# ProyectoMBD

October 18, 2024

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import geopandas as gpd
import folium
from folium.plugins import HeatMap
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import silhouette_score, davies_bouldin_score,␣
 ↪mean_squared_error, mean_absolute_error
from prophet import Prophet
from shapely.geometry import MultiPoint, Point
from shapely.ops import unary_union
from libpysal import weights
import warnings
from esda import G
import plotly.express as px
from sklearn.ensemble import IsolationForest
from tensorflow import keras
from tensorflow.keras import layers

warnings.filterwarnings('ignore')
```

Primero se comienza importando diversas bibliotecas en Python, como **pandas** y **numpy** para el manejo y análisis de datos, y **matplotlib** y **seaborn** para la visualización de información. A continuación, se incluye **geopandas** para trabajar con datos geoespaciales y **folium** para crear mapas interactivos. También se importan herramientas de aprendizaje automático como **DBSCAN** para realizar clustering y **Prophet** para el pronóstico de series temporales. Además, se utilizan módulos de **tensorflow** para construir redes neuronales. Por último, se configuran advertencias para que no se muestren, preparando así el entorno para realizar análisis y modelado de datos de manera efectiva.

[2]:
```
file_path = 'BDD_ENE_17_ABR_24.csv'

accidentes = pd.read_csv(file_path, delimiter=",", encoding='latin-1')
accidentes.head()
```

[2]:
```
   Numero    ANIO      SINIESTROS  LESIONADOS  FALLECIDOS  \
0     1.0  2017.0  DMQ00001012017         1.0         0.0
1     2.0  2017.0  ATM00002012017         1.0         0.0
2     3.0  2017.0  PNE00003012017         1.0         0.0
3     4.0  2017.0  DMQ00004012017         0.0         0.0
4     5.0  2017.0  DMQ00005012017         0.0         0.0

                                    ENTE_DE_CONTROL  LATITUD_Y  LONGITUD_X  \
0   AGENCIA METROPOLITANA DE TRANSITO DE QUITO - AMT  -0.083501  -78.417742
1  AGENCIA DE TRANSITO Y MOVILIDAD DE GUAYAQUIL -…  -2.246682  -79.897754
2                      POLICIA NACIONAL DEL ECUADOR  -0.253881  -79.217405
3   AGENCIA METROPOLITANA DE TRANSITO DE QUITO - AMT  -0.116059  -78.464188
4   AGENCIA METROPOLITANA DE TRANSITO DE QUITO - AMT  -0.239721  -78.512058

   DPA_1                       PROVINCIA  …  TRICIMOTO  \
0   17.0                       PICHINCHA  …        0.0
1    9.0                          GUAYAS  …        0.0
2   23.0  SANTO DOMINGO DE LOS TSACHILAS  …        0.0
3   17.0                       PICHINCHA  …        0.0
4   17.0                       PICHINCHA  …        0.0

  VEHICULO_DEPORTIVO_UTILITARIO  SUMA_DE_VEHICULOS        TIPO_ID_1 EDAD_1  \
0                           0.0                 1.0           CEDULA   36.0
1                           0.0                 1.0           CEDULA   12.0
2                           0.0                 1.0           CEDULA   26.0
3                           0.0                 1.0  NO IDENTIFICADO   -1.0
4                           0.0                 1.0           CEDULA   11.0

            SEXO_1      CONDICION_1     PARTICIPANTE_1 CASCO_1 CINTURON_1
0            MUJER        LESIONADO             PEATON      NO         NO
1           HOMBRE        LESIONADO             PEATON      NO         NO
2           HOMBRE        LESIONADO             PEATON      NO         NO
3  NO IDENTIFICADO  NO IDENTIFICADO  CONDUCTOR AUSENTE      NO         NO
4           HOMBRE             ILESO           PASAJERO      NO         NO

[5 rows x 56 columns]
```

Se procede con la carga de un conjunto de datos sobre accidentes de tránsito desde un archivo CSV llamado BDD_ENE_17_ABR_24.csv. Utilizando la biblioteca pandas, se especifica el delimitador y la codificación para asegurar que los datos se importen correctamente. Los datos se almacenan en un DataFrame denominado accidentes, que contiene diversas columnas relacionadas con los siniestros, incluyendo información sobre el año, la cantidad de lesionados y fallecidos, ubicación geográfica y otros detalles relevantes. Finalmente, se muestra el contenido del DataFrame para su revisión.

[3]:
```
df = accidentes
```

Se crea una nueva variable llamada df que almacena el DataFrame de accidentes. A continuación

se utiliza el método info() de pandas para generar un resumen del DataFrame, donde se puede observar que el DataFrame contiene 166685 entradas y 56 columnas.

[4]:
```
df = df.applymap(lambda x: x.upper() if isinstance(x, str) else x)
```

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

[5]:
```
Numero                    1
ANIO                      1
SINIESTROS                1
LESIONADOS                1
FALLECIDOS                1
ENTE_DE_CONTROL           1
LATITUD_Y                 1
LONGITUD_X                1
DPA_1                     1
PROVINCIA                 1
DPA_2                     1
CANTON                    1
DPA_3                     1
PARROQUIA                 1
DIRECCION                 1
ZONA_PLANIFICACION        1
ZONA                      1
ID_DE_LA_VIA              1
NOMBRE_DE_LA_VIA          1
UBICACION_DE_LA_VIA       1
JERARQUIA_DE_LA_VIA       1
FECHA                     1
HORA                      1
PERIODO_1                 1
PERIODO_2                 1
DIA_1                     1
DIA_2                     1
MES_1                     1
MES_2                     1
FERIADO                   1
CODIGO_CAUSA              1
CAUSA_PROBABLE            1
TIPO_DE_SINIESTRO         1
TIPO_DE_VEHICULO_1        1
SERVICIO_1                1
AUTOMOVIL                 1
BICICLETA                 1
BUS                       1
CAMION                    1
CAMIONETA                 1
```

```
EMERGENCIAS                        1
ESPECIAL                           1
FURGONETA                          1
MOTOCICLETA                        1
NO_IDENTIFICADO                    1
SCOOTER_ELECTRICO                  1
TRICIMOTO                          1
VEHICULO_DEPORTIVO_UTILITARIO      1
SUMA_DE_VEHICULOS                  1
TIPO_ID_1                          1
EDAD_1                             1
SEXO_1                             1
CONDICION_1                        0
PARTICIPANTE_1                     1
CASCO_1                            1
CINTURON_1                         1
dtype: int64
```

Se aplica una transformación al DataFrame df para convertir todos los valores de tipo cadena a mayúsculas, utilizando la función applymap, con el fin de estandarizar los datos textuales y a prevenir problemas relacionados con el uso de mayúsculas y minúsculas en el análisis posterior. Luego, se verifica la presencia de valores nulos en cada columna mediante el método isnull().sum(), revelando que todas las columnas, excepto CONDICION_1 tienen un valor nulo.

[6]:
```
df = df.dropna()
```

[7]:
```
df = df.drop_duplicates()
```

Se lleva a cabo un proceso de limpieza de datos en el DataFrame df. Primero, se eliminan todas las filas que contienen valores nulos utilizando el método dropna(), lo que asegura que no haya registros incompletos que puedan afectar los resultados del análisis. Luego, se eliminan las filas duplicadas con drop_duplicates(), garantizando que cada entrada en el DataFrame sea única.

[8]:
```
df.dtypes
```

[8]:
```
Numero                      float64
ANIO                        float64
SINIESTROS                   object
LESIONADOS                  float64
FALLECIDOS                  float64
ENTE_DE_CONTROL              object
LATITUD_Y                   float64
LONGITUD_X                  float64
DPA_1                       float64
PROVINCIA                    object
DPA_2                       float64
CANTON                       object
DPA_3                       float64
```

```
PARROQUIA                          object
DIRECCION                          object
ZONA_PLANIFICACION                 object
ZONA                               object
ID_DE_LA_VIA                       object
NOMBRE_DE_LA_VIA                   object
UBICACION_DE_LA_VIA                object
JERARQUIA_DE_LA_VIA                object
FECHA                              object
HORA                               object
PERIODO_1                          object
PERIODO_2                         float64
DIA_1                              object
DIA_2                             float64
MES_1                              object
MES_2                             float64
FERIADO                            object
CODIGO_CAUSA                       object
CAUSA_PROBABLE                     object
TIPO_DE_SINIESTRO                  object
TIPO_DE_VEHICULO_1                 object
SERVICIO_1                         object
AUTOMOVIL                         float64
BICICLETA                         float64
BUS                               float64
CAMION                            float64
CAMIONETA                         float64
EMERGENCIAS                       float64
ESPECIAL                          float64
FURGONETA                         float64
MOTOCICLETA                       float64
NO_IDENTIFICADO                   float64
SCOOTER_ELECTRICO                 float64
TRICIMOTO                         float64
VEHICULO_DEPORTIVO_UTILITARIO     float64
SUMA_DE_VEHICULOS                 float64
TIPO_ID_1                          object
EDAD_1                            float64
SEXO_1                             object
CONDICION_1                        object
PARTICIPANTE_1                     object
CASCO_1                            object
CINTURON_1                         object
dtype: object
```

Se examinan los tipos de datos de cada columna en el DataFrame df utilizando df.dtypes. El análisis revela que el DataFrame contiene una combinación de tipos de datos: 27 columnas son de tipo numérico (float64), que incluyen información cuantitativa como la cantidad de lesionados

y fallecidos, así como coordenadas geográficas. También hay 29 columnas de tipo objeto, que almacenan datos categóricos o de texto, tales como nombres de siniestros, provincias, y tipos de vehículos.

[9]:
```
columnas_interes = [
    'ANIO', 'LESIONADOS', 'FALLECIDOS', 'LATITUD_Y', 'LONGITUD_X',
    'PROVINCIA', 'CANTON', 'PARROQUIA', 'ZONA_PLANIFICACION', 'ZONA',
    'FECHA', 'HORA', 'FERIADO', 'CODIGO_CAUSA', 'CAUSA_PROBABLE',␣
 ↪'TIPO_DE_SINIESTRO',
    'TIPO_DE_VEHICULO_1', 'SERVICIO_1', 'AUTOMOVIL', 'BICICLETA', 'BUS',
    'CAMION', 'CAMIONETA', 'EMERGENCIAS', 'ESPECIAL', 'FURGONETA',
    'MOTOCICLETA', 'NO_IDENTIFICADO', 'SCOOTER_ELECTRICO', 'TRICIMOTO',
    'VEHICULO_DEPORTIVO_UTILITARIO', 'SUMA_DE_VEHICULOS'
]

df = df[columnas_interes]

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 166684 entries, 0 to 166683
Data columns (total 32 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   ANIO                    166684 non-null  float64
 1   LESIONADOS              166684 non-null  float64
 2   FALLECIDOS              166684 non-null  float64
 3   LATITUD_Y               166684 non-null  float64
 4   LONGITUD_X              166684 non-null  float64
 5   PROVINCIA               166684 non-null  object
 6   CANTON                  166684 non-null  object
 7   PARROQUIA               166684 non-null  object
 8   ZONA_PLANIFICACION      166684 non-null  object
 9   ZONA                    166684 non-null  object
 10  FECHA                   166684 non-null  object
 11  HORA                    166684 non-null  object
 12  FERIADO                 166684 non-null  object
 13  CODIGO_CAUSA            166684 non-null  object
 14  CAUSA_PROBABLE          166684 non-null  object
 15  TIPO_DE_SINIESTRO       166684 non-null  object
 16  TIPO_DE_VEHICULO_1      166684 non-null  object
 17  SERVICIO_1              166684 non-null  object
 18  AUTOMOVIL               166684 non-null  float64
 19  BICICLETA               166684 non-null  float64
 20  BUS                     166684 non-null  float64
 21  CAMION                  166684 non-null  float64
 22  CAMIONETA               166684 non-null  float64
 23  EMERGENCIAS             166684 non-null  float64
```

```
 24  ESPECIAL                          166684 non-null  float64
 25  FURGONETA                         166684 non-null  float64
 26  MOTOCICLETA                       166684 non-null  float64
 27  NO_IDENTIFICADO                   166684 non-null  float64
 28  SCOOTER_ELECTRICO                 166684 non-null  float64
 29  TRICIMOTO                         166684 non-null  float64
 30  VEHICULO_DEPORTIVO_UTILITARIO     166684 non-null  float64
 31  SUMA_DE_VEHICULOS                 166684 non-null  float64
dtypes: float64(19), object(13)
memory usage: 42.0+ MB
```

Se seleccionan las columnas de interés del DataFrame df, creando una nueva lista llamada columnas_interes que incluye variables relevantes para el análisis de accidentes de tránsito. El DataFrame se filtra para que contenga solo estas columnas, resultando en un DataFrame con 166684 entradas y 34 columnas.

[10]:
```
numeric_cols = [
    'ANIO', 'LESIONADOS', 'FALLECIDOS', 'AUTOMOVIL', 'BICICLETA', 'BUS',
    'CAMION', 'CAMIONETA', 'EMERGENCIAS', 'ESPECIAL', 'FURGONETA',
    'MOTOCICLETA', 'NO_IDENTIFICADO', 'SCOOTER_ELECTRICO', 'TRICIMOTO',
    'VEHICULO_DEPORTIVO_UTILITARIO', 'SUMA_DE_VEHICULOS'
]

for col in numeric_cols:
    df[col] = df[col].fillna(0).astype(int)
```

A continuación, se convierten las columnas numéricas a tipo entero (int32) para optimizar el uso de memoria y facilitar el análisis. Finalmente, se verifica la estructura del DataFrame, confirmando que las conversiones se han realizado correctamente y que el uso de memoria se ha reducido.

[11]:
```
lat_min, lat_max = -5.0, 1.5
lon_min, lon_max = -92.0, -75.0

df = df[(df['LATITUD_Y'] >= lat_min) & (df['LATITUD_Y'] <= lat_max) &
                (df['LONGITUD_X'] >= lon_min) & (df['LONGITUD_X'] <= lon_max)]

print(f"Filtrados {len(df)} registros dentro de los límites de Ecuador")
```

```
Filtrados 166682 registros dentro de los límites de Ecuador
```

Se establecen los límites geográficos de Ecuador mediante coordenadas de latitud y longitud. A continuación, se filtra el DataFrame df para mantener únicamente los registros que se encuentran dentro de estos límites. Después del filtrado, se verifica el tamaño del DataFrame y se puede observar que quedan 166682 registros dentro de los límites geográficos de Ecuador.

[12]:
```
df['FECHA'] = pd.to_datetime(df['FECHA'], dayfirst=True)
df['HORA'] = pd.to_datetime(df['HORA'], format='%H:%M:%S').dt.time

df['LATITUD_Y'] = pd.to_numeric(df['LATITUD_Y'], errors='coerce')
```

```
df['LONGITUD_X'] = pd.to_numeric(df['LONGITUD_X'], errors='coerce')
```

Se convierte la columna FECHA al formato de fecha datetime, especificando que el día aparece primero en el formato de las fechas (por ejemplo, "dd/mm/yyyy") utilizando el parámetro dayfirst=True. Además, convierte la columna HORA, que contiene solo la hora en formato "HH:MM", al formato de tiempo (time) utilizando pd.to_datetime() con el formato específico '%H:%M:%S'.

Además se convierte las columnas 'LATITUD_Y' y 'LONGITUD_X' a tipo numérico y maneja cualquier error convirtiendo valores no numéricos a NaN.

[13]:
```
df.describe()
```

[13]:

| | ANIO | LESIONADOS | FALLECIDOS | LATITUD_Y |
|---|---|---|---|---|
| count | 166682.000000 | 166682.000000 | 166682.000000 | 166682.000000 |
| mean | 2019.948339 | 0.814329 | 0.092691 | -1.472170 |
| min | 2017.000000 | 0.000000 | 0.000000 | -4.966098 |
| 25% | 2018.000000 | 0.000000 | 0.000000 | -2.195446 |
| 50% | 2020.000000 | 1.000000 | 0.000000 | -1.682596 |
| 75% | 2022.000000 | 1.000000 | 0.000000 | -0.279203 |
| max | 2024.000000 | 48.000000 | 22.000000 | 1.366128 |
| std | 2.175354 | 1.105809 | 0.353764 | 1.092668 |

| | LONGITUD_X | FECHA | AUTOMOVIL |
|---|---|---|---|
| count | 166682.000000 | 166682 | 166682.000000 |
| mean | -79.323644 | 2020-06-11 04:58:19.223671296 | 0.572713 |
| min | -91.108467 | 2017-01-01 00:00:00 | 0.000000 |
| 25% | -79.905261 | 2018-07-05 00:00:00 | 0.000000 |
| 50% | -79.442778 | 2020-03-07 00:00:00 | 0.000000 |
| 75% | -78.551378 | 2022-05-09 00:00:00 | 1.000000 |
| max | -75.864644 | 2024-04-30 00:00:00 | 9.000000 |
| std | 0.762532 | NaN | 0.729074 |

| | BICICLETA | BUS | CAMION | CAMIONETA |
|---|---|---|---|---|
| count | 166682.000000 | 166682.000000 | 166682.000000 | 166682.000000 |
| mean | 0.014765 | 0.064950 | 0.106082 | 0.176348 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 50% | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 75% | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| max | 3.000000 | 4.000000 | 7.000000 | 7.000000 |
| std | 0.122338 | 0.257417 | 0.337680 | 0.414217 |

| | EMERGENCIAS | ESPECIAL | FURGONETA | MOTOCICLETA |
|---|---|---|---|---|
| count | 166682.000000 | 166682.000000 | 166682.000000 | 166682.000000 |
| mean | 0.000768 | 0.007955 | 0.014363 | 0.312655 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 50% | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

```
75%          0.000000       0.000000       0.000000       1.000000
max          2.000000       2.000000       2.000000       6.000000
std          0.028131       0.091564       0.120733       0.500672

       NO_IDENTIFICADO  SCOOTER_ELECTRICO      TRICIMOTO  \
count    166682.000000      166682.000000  166682.000000
mean          0.237728           0.000762       0.001188
min           0.000000           0.000000       0.000000
25%           0.000000           0.000000       0.000000
50%           0.000000           0.000000       0.000000
75%           0.000000           0.000000       0.000000
max           4.000000           1.000000       2.000000
std           0.467088           0.027593       0.035306

       VEHICULO_DEPORTIVO_UTILITARIO  SUMA_DE_VEHICULOS
count                 166682.000000      166682.000000
mean                       0.099087           1.609364
min                        0.000000           1.000000
25%                        0.000000           1.000000
50%                        0.000000           2.000000
75%                        0.000000           2.000000
max                        6.000000          10.000000
std                        0.324222           0.624199
```

Se utiliza df.describe() para obtener un resumen estadístico de las columnas numéricas del DataFrame df. Se observa que el año promedio de los registros es 2019.95, con un rango de años de 2017 a 2024. En cuanto a los lesionados y fallecidos, el promedio es de aproximadamente 0.81 y 0.09, respectivamente, con máximos de 48 y 22. Las coordenadas geográficas se encuentran dentro de los límites esperados para Ecuador, y se presenta información sobre la cantidad de vehículos involucrados en los accidentes.

[14]:
```
df['DIA_SEMANA'] = df['FECHA'].dt.day_name()

df['DIA_SEMANA'].unique()
```

[14]:
```
array(['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
       'Saturday'], dtype=object)
```

[15]:
```
df['HORA']
```

[15]:
```
0         00:15:00
1         00:25:00
2         00:25:00
3         00:25:00
4         00:30:00
             …
166679    14:00:00
166680    10:15:00
```

```
166681    08:50:00
166682    01:45:00
166683    03:17:00
Name: HORA, Length: 166682, dtype: object
```

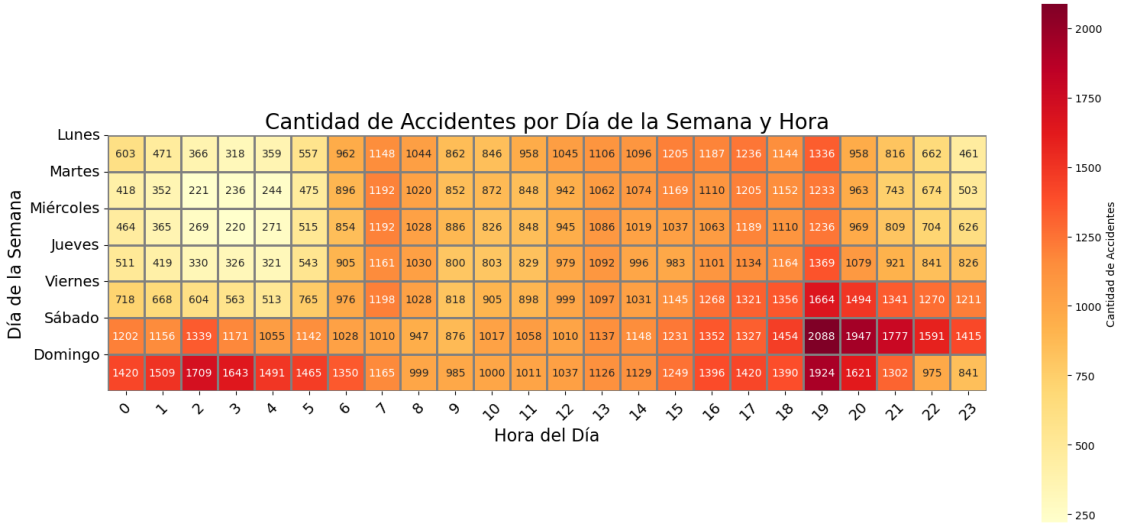
[16]:
```
df['HORA'] = pd.to_datetime(df['HORA'], format='%H:%M:%S', errors='coerce').dt.
 ↪hour

df['DIA_SEMANA'] = df['FECHA'].dt.day_name()

print("Valores únicos en DIA_SEMANA:", df['DIA_SEMANA'].unique())

accidentes_por_dia_hora = df.groupby(['DIA_SEMANA', 'HORA']).size().unstack().
 ↪fillna(0)
```

```
Valores únicos en DIA_SEMANA: ['Sunday' 'Monday' 'Tuesday' 'Wednesday'
'Thursday' 'Friday' 'Saturday']
```

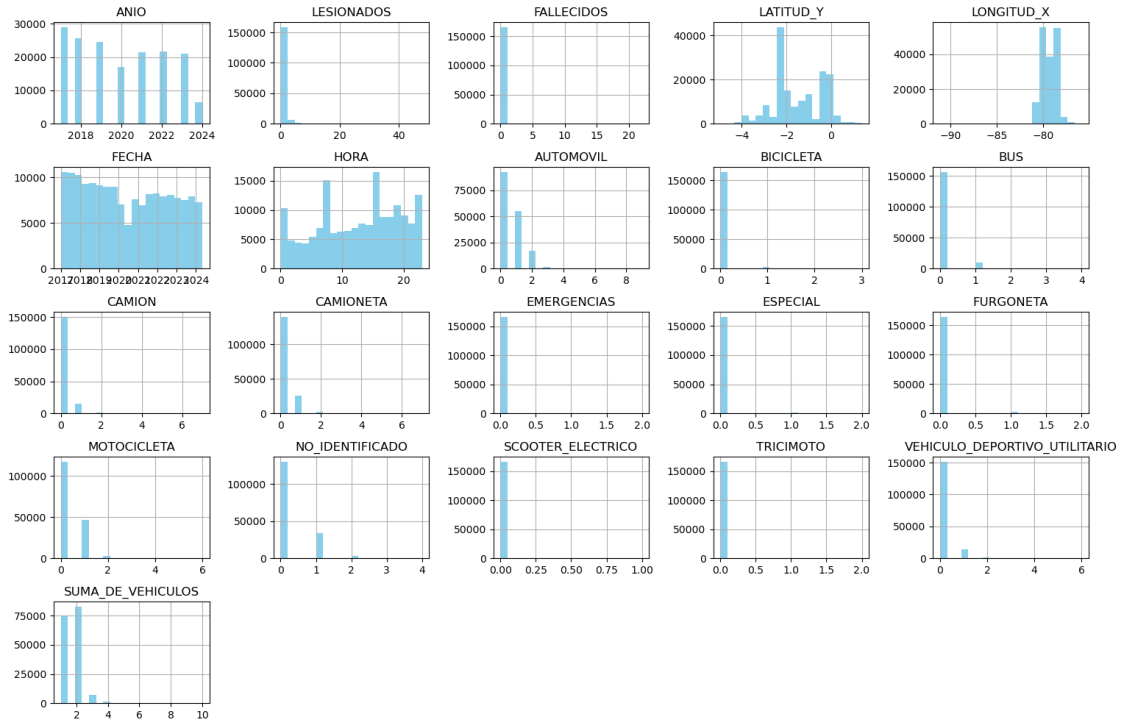
[17]:
```
dias_ordenados = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday']
accidentes_por_dia_hora = accidentes_por_dia_hora.reindex(dias_ordenados)
```

[18]:
```
import seaborn as sns
import matplotlib.pyplot as plt

dias_ordenados = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday']
accidentes_por_dia_hora = accidentes_por_dia_hora.reindex(dias_ordenados)

plt.figure(figsize=(16, 10))  # Aumentar el tamaño de la figura

sns.heatmap(accidentes_por_dia_hora, cmap="YlOrRd", linewidths=1,␣
 ↪linecolor='grey',
            annot=True, fmt=".0f", cbar_kws={'label': 'Cantidad de␣
 ↪Accidentes', 'shrink': 0.7},
            square=True)  # Reducir la barra de color con 'shrink'

plt.title('Cantidad de Accidentes por Día de la Semana y Hora', fontsize=20)
plt.xlabel('Hora del Día', fontsize=16)
plt.ylabel('Día de la Semana', fontsize=16)

plt.yticks(ticks=[0, 1, 2, 3, 4, 5, 6], labels=['Lunes', 'Martes', 'Miércoles',␣
 ↪'Jueves', 'Viernes', 'Sábado', 'Domingo'], fontsize=14)

plt.xticks(fontsize=14, rotation=45)

plt.tight_layout()
```

```
plt.show()
```




[19]:
```
df.hist(figsize=(15, 10), bins=20, color='skyblue')
plt.tight_layout()
plt.show()
```



**Año** - El histograma muestra una distribución de accidentes por año desde 2018 hasta 2024. - Se observa un aumento en la cantidad de registros en 2020 y 2024, lo que indica un aumento en los accidentes de tránsito, se tiene que tomar en cuenta que en el 2020 debido a la pandemia mundial COVID dismuyeron los accidentes de tránsito pero que después de este año comienzan a elevar nuevamente.

**Lesionados y fallecidos** - El histograma de lesionados muestra que la mayoría de los registros tienen cero o un lesionado, con pocos casos que superan los 5 lesionados. La concentración en valores bajos (0-1) sugiere que, aunque hay accidentes, la mayoría no resultan en lesiones graves, lo que puede ser un indicador positivo en términos de seguridad vial. - El histograma de fallecidos muestra que la mayoría de los accidentes no resultan en muertes, con la mayoría de los registros teniendo cero fallecidos, sin embargo hay algunos casos aislados con un número mayor de fallecidos (hasta 22), lo que indica que, aunque raros, los accidentes fatales ocurren y representan un riesgo significativo.

**Latitud (LATITUD_Y) y Longitud (LONGITUD_X)** - El histograma de latitud muestra que la mayoría de los accidentes se producen entre -2 y -1.5, lo cual está dentro de los límites geográficos de Ecuador, indicando que los datos son relevantes geográficamente. - El histograma de longitud muestra una concentración en valores alrededor de -80 a -78, lo que también se alinea con las coordenadas geográficas de Ecuador.

**Suma de Vehículos** - El histograma de la suma de vehículos muestra que la mayoría de los accidentes involucran entre 0 y 2 vehículos, lo que puede reflejar que muchos accidentes son simples colisiones o incidentes de un solo vehículo. - La distribución indica que hay pocos casos con un número significativo de vehículos involucrados, es decir, la mayoría de los accidentes son relativamente menores en términos de la cantidad de vehículos implicados.

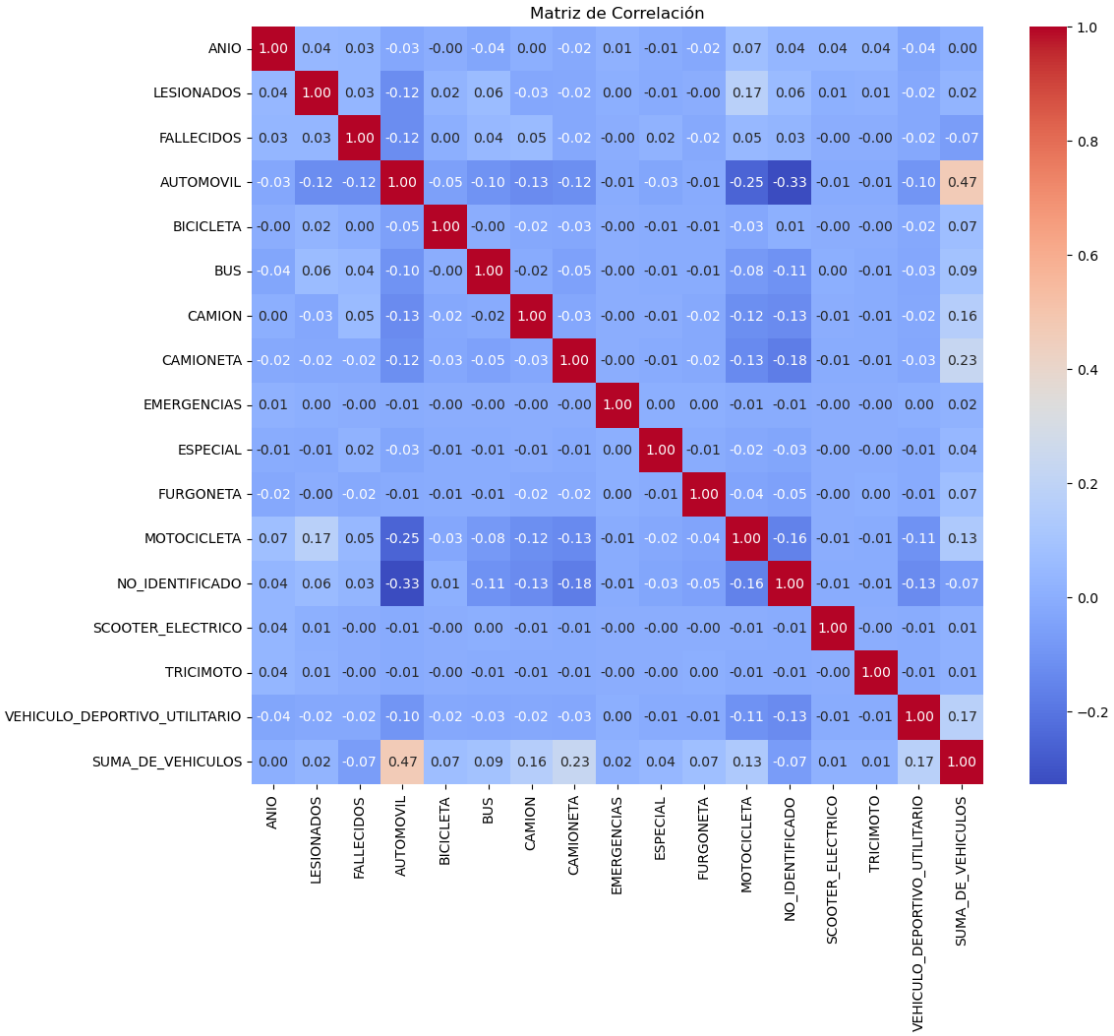
**Tipos de Vehículos** - El histograma que representa la cantidad de vehículos involucrados en los accidentes (automóvil, bicicleta, bus, camión, camioneta, emergencias, especial, furgoneta, motocicleta, no identificado, scooter, tricimoto y vehículo deportivo utilitario) muestra que la mayoría de los accidentes involucran un número reducido de vehículos. - La mayoría de los tipos de vehículos tienen una frecuencia baja, con una clara concentración en valores cercanos a cero, lo que quiere decir que la mayoría de los accidentes no involucran muchos vehículos al mismo tiempo. - Para automóviles, camiones y camionetas, los registros son más frecuentes, lo que refleja su predominancia en las vías y su uso habitual en el contexto de tráfico. - Automóviles muestran un número relativamente alto de registros, lo que es consistente con su prevalencia en la vía. - Bicicletas y motocicletas también tienen registros, aunque en menor frecuencia, lo que indica un menor uso en comparación con los automóviles. - Buses y camiones tienen una frecuencia moderada, sugiriendo su uso en el transporte público y de carga, respectivamente. - Tipos como scooters, tricimotos y vehículos deportivos utilitarios tienen una baja presencia en los accidentes, lo que podría reflejar su menor cantidad en circulación o su uso en situaciones menos propensas a accidentes. - La categoría "no identificado" presenta registros, lo que indica casos donde el tipo de vehículo no se documentó adecuadamente.

Los datos quieren decir que aunque hay un número considerable de accidentes, la mayoría son de bajo impacto en términos de lesiones y fatalidades. Sin embargo, los picos en los años recientes y las variaciones en la suma de vehículos involucrados resaltan áreas donde se podría enfocar la atención para mejorar la seguridad vial.

La distribución de los tipos de vehículos involucrados en accidentes señalan que los automóviles predominan, seguidos de camiones y otros tipos de vehículos. La baja frecuencia de vehículos

menos comunes puede ser indicativa de su menor participación en el tráfico o de patrones de uso específicos.

[20]:
```
plt.figure(figsize=(12, 10))
correlation_matrix = df[numeric_cols].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Matriz de Correlación')
plt.savefig('matriz_correlacion.png', format='png')
plt.show()
```




La matriz de correlación muestra las relaciones entre varias variables relacionadas con incidentes vehiculares, lesiones y muertes.

**1. Lesionados y Automóviles:** La correlación entre el número de lesionados y los automóviles involucrados en accidentes es negativa (-0.122), es decir, en general cuando más automóviles están involucrados en un accidente, no necesariamente hay más lesionados. Puede ser que los accidentes

con automóviles resulten menos graves en comparación con otros vehículos, como motocicletas, que suelen estar más expuestas.

**2. Motocicletas y Lesionados:** Tiene una correlación positiva (0.174), lo que indica que, cuando hay accidentes que involucran motocicletas, es más probable que haya personas lesionadas, lo que confirma que los motociclistas están en mayor riesgo de sufrir lesiones cuando están involucrados en un accidente.

**3. Latitud/Longitud y Fallecidos:** La correlación entre la ubicación geográfica (latitud y longitud) y el número de fallecidos es muy baja, cercana a 0, es decir, la ubicación exacta de un accidente no está fuertemente relacionada con cuántas personas fallecen en él, sin embargo, pequeños patrones pueden estar presentes, indicando que algunos lugares pueden tener un riesgo ligeramente mayor.

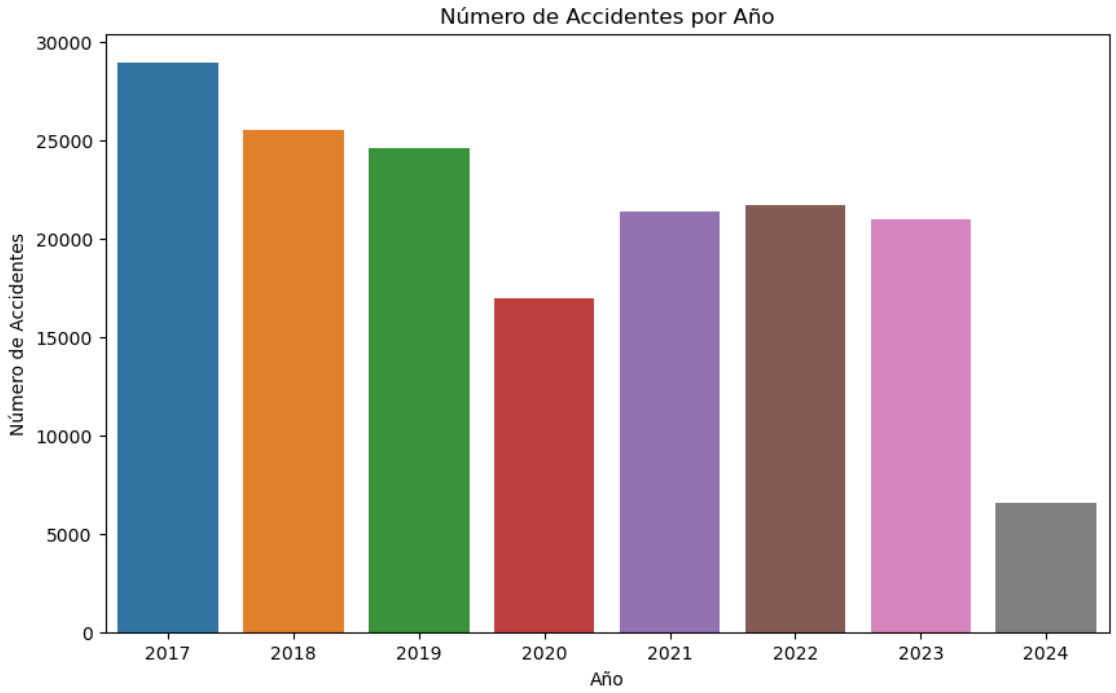
**4. Suma de Vehículos Involucrados y Tipo de Vehículo:** Existe una correlación fuerte y positiva entre el número total de vehículos involucrados en un accidente y el tipo de vehículo, especialmente automóviles (0.467), es decir, los automóviles son frecuentemente parte de los accidentes en los que hay muchos vehículos involucrados.

**5. Fallecidos y Tipo de Vehículo:** La correlación entre el número de fallecidos y el tipo de vehículo no es muy fuerte, ningún tipo de vehículo específico está claramente vinculado a un aumento en la probabilidad de fallecimientos en los accidentes. Sin embargo, algunos vehículos, como los camiones, pueden estar ligeramente más involucrados en accidentes con fallecidos debido a su tamaño y la gravedad potencial del impacto.

**6. Motocicletas y Accidentes Mortales:** Aunque la correlación entre motocicletas y fallecidos es más débil que con lesionados, todavía hay una tendencia a que los accidentes con motocicletas puedan ser más graves.

**7. Accidentes y Ubicaciones:** Aunque la correlación entre accidentes y ubicación (latitud y longitud) no es muy alta, esto podría ser un indicio de que ciertas áreas del país son más propensas a accidentes.

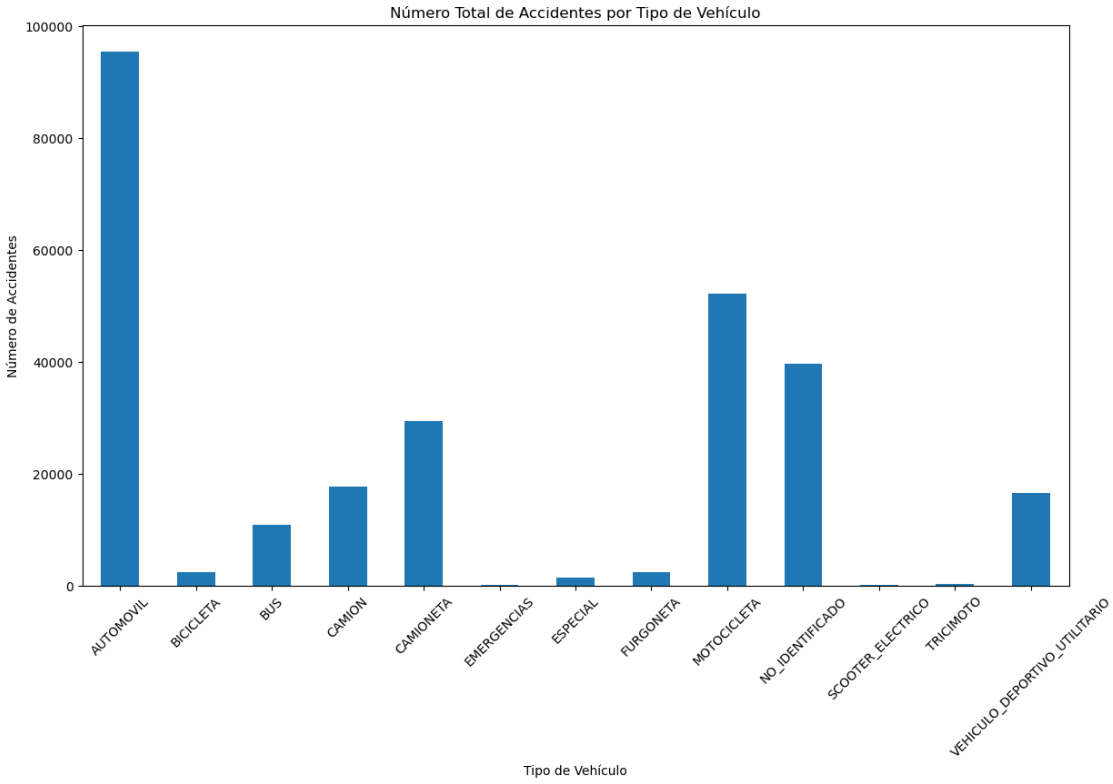
[21]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='ANIO', data=df)
plt.title('Número de Accidentes por Año')
plt.xlabel('Año')
plt.ylabel('Número de Accidentes')
plt.show()
```




El gráfico de barras ilustra el número de accidentes por año desde 2017 hasta 2024.

El año 2017 tuvo el mayor número de accidentes, superando los 30000. El número de accidentes disminuyó gradualmente desde 2018 hasta 2020, con 2019 mostrando una ligera caída en comparación con 2018. El año 2020 experimentó una reducción significativa en los accidentes, debido a factores como los confinamientos por la pandemia COVID-19. En los años 2021 y 2022 se registra una alza en los accidentes de tránsito. En el 2023 se puede observar un ligero decrecimiento a diferencia del 2022, pero en el 2024 existe un decrecimiento total en relación a los otros años, debido a que solo se encuentran registros hasta el 30 de abril del mismo año, por lo que representa el 1/3 de los otros años.

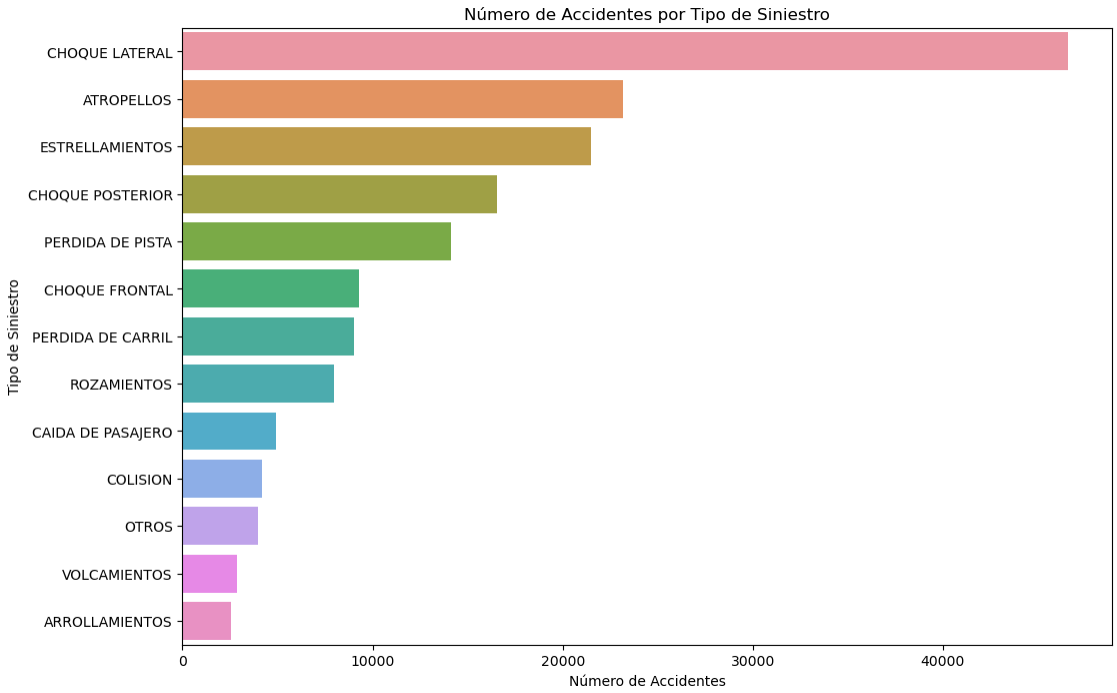
[22]:
```
plt.figure(figsize=(12, 8))
vehicle_types = [
    'AUTOMOVIL', 'BICICLETA', 'BUS', 'CAMION', 'CAMIONETA',
    'EMERGENCIAS', 'ESPECIAL', 'FURGONETA', 'MOTOCICLETA',
    'NO_IDENTIFICADO', 'SCOOTER_ELECTRICO', 'TRICIMOTO',
    'VEHICULO_DEPORTIVO_UTILITARIO'
]
df[vehicle_types].sum().plot(kind='bar', figsize=(14, 8))
plt.title('Número Total de Accidentes por Tipo de Vehículo')
plt.xlabel('Tipo de Vehículo')
plt.ylabel('Número de Accidentes')
plt.xticks(rotation=45)
plt.show()
```




El gráfico de barras muestra el número total de accidentes categorizados por tipo de vehículo.

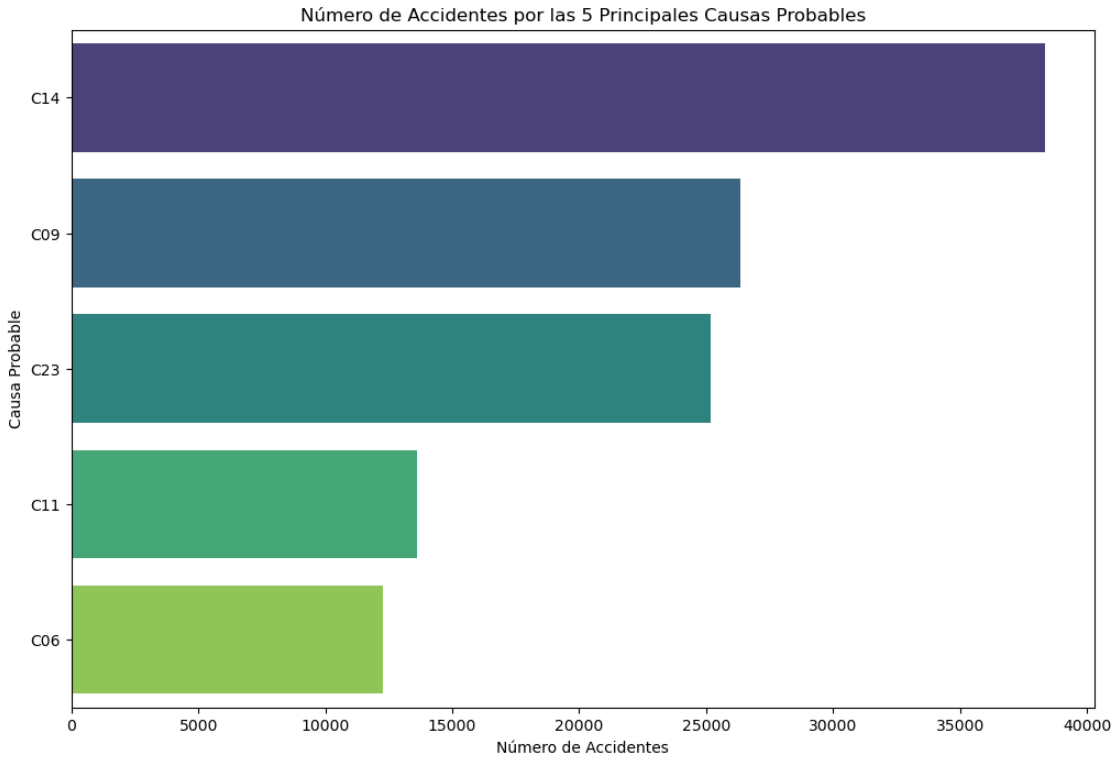
1. Automóvil tiene el mayor número de accidentes, significativamente más que otros tipos.
2. Motocicleta y bicicleta también presentan un número notable de incidentes.
3. Otros tipos de vehículos como camión, camioneta y bus tienen menos accidentes, pero aún contribuyen al total.
4. Algunas categorías como scooter eléctrico y tricimoto tienen conteos de accidentes muy bajos.

[23]:
```
plt.figure(figsize=(12, 8))
sns.countplot(y='TIPO_DE_SINIESTRO', data=df, order=df['TIPO_DE_SINIESTRO'].
 ↪value_counts().index)
plt.title('Número de Accidentes por Tipo de Siniestro')
plt.xlabel('Número de Accidentes')
plt.ylabel('Tipo de Siniestro')
plt.show()
```




El gráfico muestra el número de accidentes categorizados por tipo de siniestro, siendo los con mayor frecuencia choque lateral, atropellos, estrellamiento, choque posterior y pérdida de pista.

[24]:
```
top_5_causas = df['CODIGO_CAUSA'].value_counts().head(5)

plt.figure(figsize=(12, 8))
sns.barplot(y=top_5_causas.index, x=top_5_causas.values, palette='viridis')
plt.title('Número de Accidentes por las 5 Principales Causas Probables')
plt.xlabel('Número de Accidentes')
plt.ylabel('Causa Probable')
plt.show()
```




[25]:
```
pd.set_option('display.max_colwidth', None)

top_5_causas = df['CODIGO_CAUSA'].value_counts().head(5).index
top_5_causas_df = df[df['CODIGO_CAUSA'].isin(top_5_causas)]

top_5_causas_info = top_5_causas_df[['CODIGO_CAUSA', 'CAUSA_PROBABLE']].
↪drop_duplicates().set_index('CODIGO_CAUSA')
top_5_causas_info = top_5_causas_info.loc[top_5_causas]

top_5_causas_info
```

[25]:
```
                                                    CAUSA_PROBABLE
CODIGO_CAUSA
C14           CONDUCIR DESATENTO A LAS CONDICIONES DE TRANSITO (CELULAR,
PANTALLAS DE VIDEO, COMIDA, MAQUILLAJE O CUALQUIER OTRO ELEMENTO DISTRACTOR).
C09
CONDUCIR VEHICULO SUPERANDO LOS LIMITES MAXIMOS DE VELOCIDAD.
C23                                              NO RESPETAR LAS SEÑALES
REGLAMENTARIAS DE TRANSITO. (PARE, CEDA EL PASO, LUZ ROJA DEL SEMAFORO, ETC).
C11                                                                   NO
MANTENER LA DISTANCIA PRUDENCIAL CON RESPECTO AL VEHICULO QUE LE ANTECEDE.
C06                                               CONDUCE BAJO LA INFLUENCIA DE
```

ALCOHOL, SUSTANCIAS ESTUPEFACIENTES O PSICOTROPICAS Y/O MEDICAMENTOS.

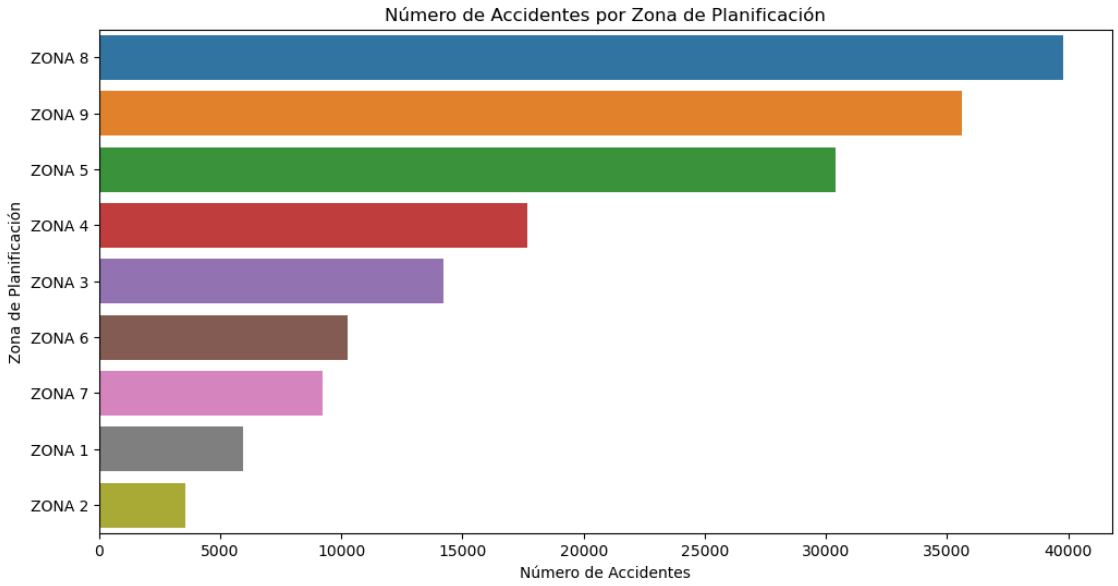
El gráfico presenta el número de accidentes categorizados por las cinco principales causas probables. Los datos se representan en un gráfico de barras horizontal, donde el eje y indica las causas probables (C06, C11, C14, C09, C23) y el eje x muestra el número de accidentes.

Del gráfico se puede observar que:

- La causa etiquetada como C14 tiene el mayor número de accidentes.
- C09, C23, C11 y C06 siguen en orden descendente, siendo C06 la que tiene la menor cantidad.

Las causas con el código correspondiente se puede observar en la tabla anterior.

[26]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=df, y='ZONA_PLANIFICACION', order=df['ZONA_PLANIFICACION'].
 ↪value_counts().index)
plt.title('Número de Accidentes por Zona de Planificación')
plt.xlabel('Número de Accidentes')
plt.ylabel('Zona de Planificación')
plt.show()
```



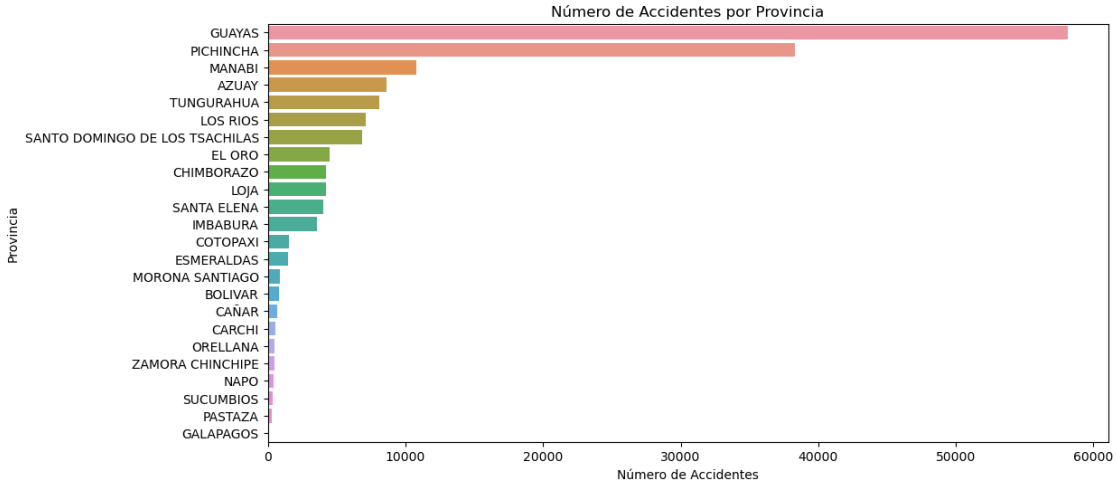

Las Zonas de Planificación en Ecuador son áreas territoriales definidas por el Estado para facilitar la gestión y planificación del desarrollo regional.

El gráfico de barras muestra el número de accidentes por zona de planificación.

- **ZONA 8** (Guayaquil, Samborondón, Durán) tiene el mayor número de accidentes.
- **ZONA 9** (Distrito Metropolitano de Quito) le sigue con una cifra significativa.
- **ZONA 5** (Santa Elena, Guayas, Bolívar, Los Ríos, Galápagos) y **ZONA 4** (Manabí, Santo Domingo de los Tsáchilas) también presentan números considerables, pero inferiores a ZONA 9.

- Las zonas restantes (**ZONA 3, ZONA 6, ZONA 7, ZONA 1 y ZONA 2**) muestran progresivamente menos accidentes, siendo ZONA 2 la que tiene la cifra más baja.

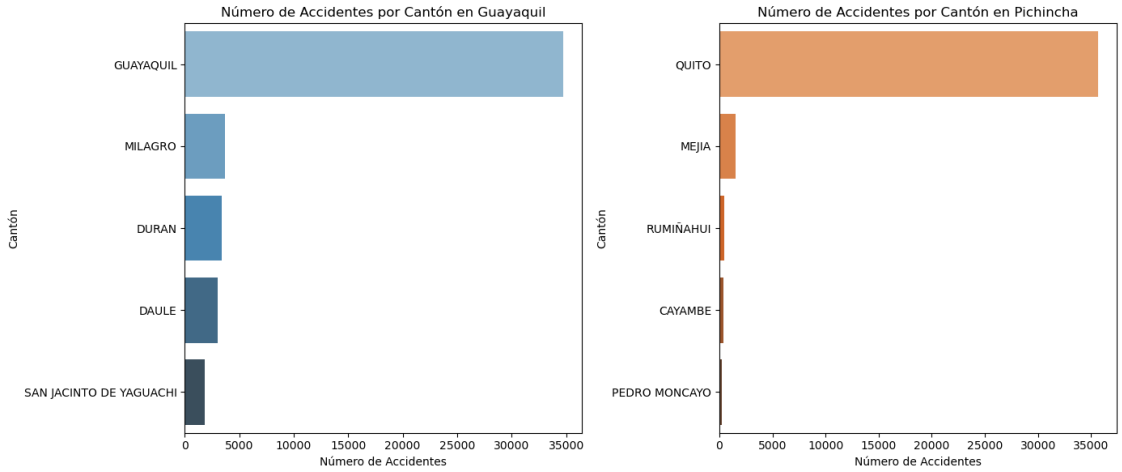
[27]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=df, y='PROVINCIA', order=df['PROVINCIA'].value_counts().
 ↪index)
plt.title('Número de Accidentes por Provincia')
plt.xlabel('Número de Accidentes')
plt.ylabel('Provincia')
plt.show()
```




El gráfico de barras muestra el número de accidentes por provincia, destacando que Guayas tiene la mayor cantidad, seguido de Pichincha y Manabí. Los datos reflejan una disparidad significativa en el número de accidentes, especialmente en la parte superior de la lista. Otras provincias, como Azuay y Tungurahua, también presentan cifras notables, pero son mucho más bajas en comparación con Guayas. Las provincias restantes, como Galápagos y Pastaza, tienen considerablemente menos accidentes.

[28]:
```
df_guayaquil = df[df['PROVINCIA'] == 'GUAYAS']
df_pichincha = df[df['PROVINCIA'] == 'PICHINCHA']

accidentes_guayaquil = df_guayaquil['CANTON'].value_counts().head(5)
accidentes_pichincha = df_pichincha['CANTON'].value_counts().head(5)

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.barplot(y=accidentes_guayaquil.index, x=accidentes_guayaquil.values,␣
 ↪palette='Blues_d')
plt.title('Número de Accidentes por Cantón en Guayaquil')
plt.xlabel('Número de Accidentes')
```

```
plt.ylabel('Cantón')

plt.subplot(1, 2, 2)
sns.barplot(y=accidentes_pichincha.index, x=accidentes_pichincha.values,␣
 ↪palette='Oranges_d')
plt.title('Número de Accidentes por Cantón en Pichincha')
plt.xlabel('Número de Accidentes')
plt.ylabel('Cantón')

plt.tight_layout()
plt.show()
```




Al tener la mayor cantidad de registros de accidentes de tránsito en Guayas y en Pichincha se analiza por cantón en dichas provincia.

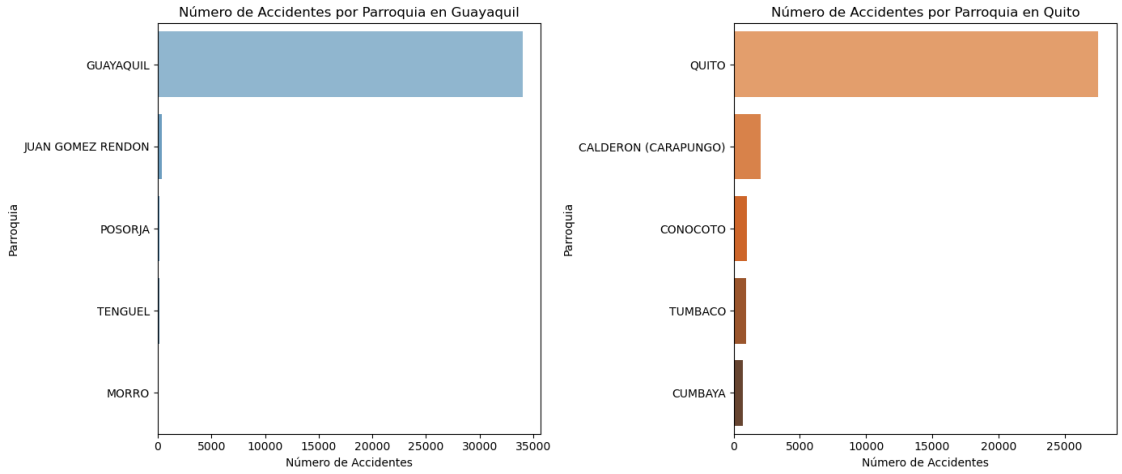
**Guayaquil (gráfico de la izquierda):** - Guayaquil tiene el mayor número de accidentes, superando significativamente a otros cantones. - Otros cantones como Milagro, Durán, Daule y San Jacinto de Yaguachi muestran un número de accidentes considerablemente más bajo.

**Pichincha (gráfico de la derecha):** - Quito destaca con el mayor número de accidentes, superando con creces a los otros cantones. - Cantones como Mejía, Rumiñahui, Cayambe y Pedro Moncayo tienen totales mucho más bajos en comparación.

En general, ambas regiones muestran un cantón dominante con un número desproporcionadamente alto de accidentes en comparación con los demás.

[29]:
```
df_guayaquil = df[(df['PROVINCIA'] == 'GUAYAS') & (df['CANTON'] == 'GUAYAQUIL')]
df_quito = df[(df['PROVINCIA'] == 'PICHINCHA') & (df['CANTON'] == 'QUITO')]

accidentes_guayaquil = df_guayaquil['PARROQUIA'].value_counts().head(5)
accidentes_quito = df_quito['PARROQUIA'].value_counts().head(5)
```

```
df_parroquia_gye = df[(df['PARROQUIA']  == 'GUAYAQUIL')]
df_parroquia_uio = df[(df['PARROQUIA'] == 'QUITO')]

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.barplot(y=accidentes_guayaquil.index, x=accidentes_guayaquil.values,␣
 ↪palette='Blues_d')
plt.title('Número de Accidentes por Parroquia en Guayaquil')
plt.xlabel('Número de Accidentes')
plt.ylabel('Parroquia')

plt.subplot(1, 2, 2)
sns.barplot(y=accidentes_quito.index, x=accidentes_quito.values,␣
 ↪palette='Oranges_d')
plt.title('Número de Accidentes por Parroquia en Quito')
plt.xlabel('Número de Accidentes')
plt.ylabel('Parroquia')

plt.tight_layout()
plt.show()
```



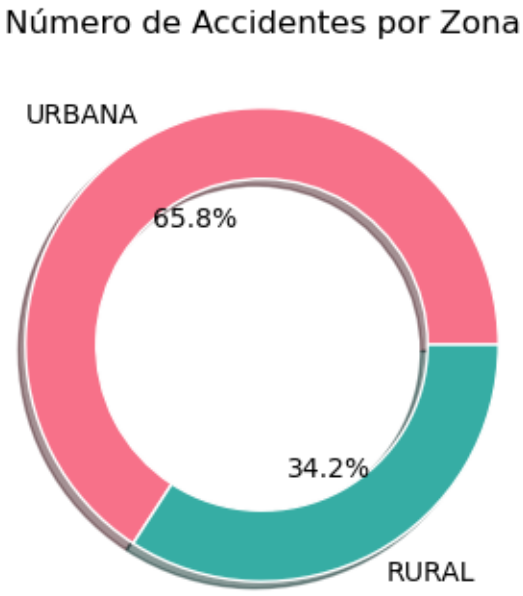

Se procede a analizar la cantidad de accidentes de los cantones de Guayaquil y Quito.

**Guayaquigráfico de la izquierda):** - En la parroquia GUAYAQUIL es significativamente más alto la cantidad de accidentes que se reportan a diferencia de los demás. - Otras parroquias como JUAN GOMEZ RENDON, POSORJA, TENGUEL y MORRO tienen recuentos de accidentes mucho más bajos en comparación.

**Quito (gráfico de la derecha):** - QUITO también presenta el mayor número de accidentes, pero no es tan abrumadoramente dominante como en Guayaquil. - Otras parroquias como CALDERON (CARAPUNGO), CONOCOTO, TUMBACO, y CUMBAYA muestran números de accidentes no-

tables, pero más bajos.

[30]:
```
accidentes_por_zona = df['ZONA'].value_counts()

plt.figure(figsize=(5, 4))
colors = sns.color_palette('husl', n_colors=len(accidentes_por_zona))
plt.pie(accidentes_por_zona, labels=accidentes_por_zona.index, autopct='%1.
↪1f%%', colors=colors, wedgeprops=dict(width=0.3, edgecolor='w'), shadow=True)
plt.title('Número de Accidentes por Zona')
plt.show()
```




El gráfico de pastel muestra la distribución de accidentes por área, indicando que:

- Las áreas urbanas representan el 65.8% del total de accidentes.
- Las áreas rurales representan el 34.2%.

Lo que destaca que una proporción significativamente mayor de accidentes ocurre en entornos urbanos en comparación con los rurales.

[31]:
```
parroquias_guayaquil = [
    'AYACUCHO', 'BOLÍVAR-SAGRARIO', 'CARBO-CONCEPCIÓN', 'FEBRES CORDERO',
    'GARCÍA MORENO', 'LETAMENDI', '9 DE OCTUBRE', 'OLMEDO-SAN ALEJO',
    'ROCA', 'ROCAFUERTE', 'SUCRE', 'TARQUI', 'URDANETA', 'XIMENA',
    'CHONGÓN', 'PASCUALES', 'GUAYAQUIL'
]

df_parroquia_gye = df[df['PARROQUIA'].isin(parroquias_guayaquil)]
```

```
lat_min, lat_max = -2.25, -2.10
lon_min, lon_max = -79.95, -79.85

df_parroquia_gye = df_parroquia_gye[(df_parroquia_gye['LATITUD_Y'] >= lat_min)␣
 ↪& (df_parroquia_gye['LATITUD_Y'] <= lat_max) &
(df_parroquia_gye['LONGITUD_X'] >= lon_min) & (df_parroquia_gye['LONGITUD_X']␣
 ↪<= lon_max)]


parroquias_quito = [
    'ALANGASÍ', 'AMAGUAÑA', 'ATAHUALPA', 'CALACALÍ', 'CALDERÓN', 'CONOCOTO',␣
 ↪'CUMBAYÁ',
    'CHAVEZPAMBA', 'CHECA', 'EL QUINCHE', 'GUALEA', 'GUANGOPOLO',␣
 ↪'GUAYLLABAMBA',
    'LA MERCED', 'LLANO CHICO', 'LLOA', 'NANEGAL', 'NANEGALITO', 'NAYÓN',␣
 ↪'NONO',
    'PACTO', 'PERUCHO', 'PIFO', 'PÍNTAG', 'POMASQUI', 'PUÉLLARO', 'PUEMBO',
    'SAN ANTONIO DE PICHINCHA', 'SAN JOSÉ DE MINAS', 'TABABELA', 'TUMBACO',␣
 ↪'YARUQUÍ',
    'ZÁMBIZA', 'BELISARIO QUEVEDO', 'EL CONDADO', 'LA MENA', 'EL INCA',␣
 ↪'MAGDALENA', 'CARCELÉN',
    'GUAMANÍ', 'MARISCAL SUCRE', 'CENTRO HISTÓRICO', 'IÑAQUITO', 'PONCEANO',␣
 ↪'CHILIBULO',
    'ITCHIMBÍA', 'PUENGASÍ', 'CHILLOGALLO', 'JIPIJAPA', 'QUITUMBE',␣
 ↪'CHIMBACALLE',
    'KENNEDY', 'RUMIPAMBA', 'COCHAPAMBA', 'LA ARGELIA', 'SAN BARTOLO', 'COMITÉ␣
 ↪DEL PUEBLO',
    'LA ECUATORIANA', 'SAN JUAN', 'CONCEPCIÓN', 'LA FERROVIARIA', 'SOLANDA',␣
 ↪'COTOCOLLAO',
    'LA LIBERTAD', 'TURUBAMBA', 'QUITO'
]

df_parroquia_uio = df[df['PARROQUIA'].isin(parroquias_quito)]

lat_min, lat_max = -0.5, -0.1
lon_min, lon_max = -78.6, -78.2

df_parroquia_uio = df_parroquia_uio[(df_parroquia_uio['LATITUD_Y'] >= lat_min)␣
 ↪& (df_parroquia_uio['LATITUD_Y'] <= lat_max) &
(df_parroquia_uio['LONGITUD_X'] >= lon_min) & (df_parroquia_uio['LONGITUD_X']␣
 ↪<= lon_max)]
```

[32]:
```
df
```

[32]:

| | ANIO | LESIONADOS | FALLECIDOS | LATITUD_Y | LONGITUD_X | \ |
|---|---|---|---|---|---|---|
| 0 | 2017 | 1 | 0 | -0.083501 | -78.417742 | |
| 1 | 2017 | 1 | 0 | -2.246682 | -79.897754 | |
| 2 | 2017 | 1 | 0 | -0.253881 | -79.217405 | |
| 3 | 2017 | 0 | 0 | -0.116059 | -78.464188 | |
| 4 | 2017 | 0 | 0 | -0.239721 | -78.512058 | |
| … | … | … | … | … | … | |
| 166679 | 2024 | 0 | 1 | -0.232671 | -78.340748 | |
| 166680 | 2024 | 1 | 0 | -0.261181 | -78.488094 | |
| 166681 | 2024 | 0 | 0 | -0.255168 | -78.484198 | |
| 166682 | 2024 | 0 | 0 | -0.220589 | -78.338811 | |
| 166683 | 2024 | 2 | 0 | -0.358812 | -78.469731 | |

| | PROVINCIA | CANTON | \ |
|---|---|---|---|
| 0 | PICHINCHA | QUITO | |
| 1 | GUAYAS | GUAYAQUIL | |
| 2 | SANTO DOMINGO DE LOS TSACHILAS | SANTO DOMINGO | |
| 3 | PICHINCHA | QUITO | |
| 4 | PICHINCHA | QUITO | |
| … | … | … | |
| 166679 | PICHINCHA | QUITO | |
| 166680 | PICHINCHA | QUITO | |
| 166681 | PICHINCHA | QUITO | |
| 166682 | PICHINCHA | QUITO | |
| 166683 | PICHINCHA | RUMIÑAHUI | |

| | PARROQUIA | ZONA_PLANIFICACION | ZONA | … | \ |
|---|---|---|---|---|---|
| 0 | CALDERON (CARAPUNGO) | ZONA 9 | RURAL | … | |
| 1 | GUAYAQUIL | ZONA 8 | URBANA | … | |
| 2 | SANTO DOMINGO DE LOS COLORADOS | ZONA 4 | URBANA | … | |
| 3 | QUITO | ZONA 9 | RURAL | … | |
| 4 | QUITO | ZONA 9 | URBANA | … | |
| … | … | … | … | … | |
| 166679 | PIFO | ZONA 9 | RURAL | … | |
| 166680 | CONOCOTO | ZONA 9 | RURAL | … | |
| 166681 | CONOCOTO | ZONA 9 | RURAL | … | |
| 166682 | PIFO | ZONA 9 | RURAL | … | |
| 166683 | SANGOLQUI | ZONA 2 | RURAL | … | |

| | EMERGENCIAS | ESPECIAL | FURGONETA | MOTOCICLETA | NO_IDENTIFICADO | \ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | |
| 1 | 0 | 0 | 0 | 0 | 0 | |
| 2 | 0 | 0 | 0 | 1 | 0 | |
| 3 | 0 | 0 | 0 | 0 | 0 | |
| 4 | 0 | 0 | 0 | 0 | 1 | |
| … | … | … | … | … | … | |
| 166679 | 0 | 0 | 0 | 1 | 0 | |

```
166680           0         0         0           1                0
166681           1         0         0           0                0
166682           0         0         0           0                0
166683           0         0         0           0                0

       SCOOTER_ELECTRICO TRICIMOTO VEHICULO_DEPORTIVO_UTILITARIO  \
0                      0         0                             0
1                      0         0                             0
2                      0         0                             0
3                      0         0                             0
4                      0         0                             0
…                    …       …                               …
166679                 0         0                             0
166680                 0         0                             1
166681                 0         0                             1
166682                 0         0                             0
166683                 0         0                             0

        SUMA_DE_VEHICULOS  DIA_SEMANA
0                       1      Sunday
1                       1      Sunday
2                       1      Sunday
3                       1      Sunday
4                       1      Sunday
…                     …          …
166679                  1    Saturday
166680                  2      Sunday
166681                  3     Tuesday
166682                  2      Friday
166683                  2      Sunday

[166682 rows x 33 columns]
```

## 0.1 ANALISIS GEOESPACIAL

[33]:
```
mapa = folium.Map(location=[df['LATITUD_Y'].mean(), df['LONGITUD_X'].mean()],␣
 ↪zoom_start=6)
heat_data = [[row['LATITUD_Y'], row['LONGITUD_X']] for index, row in df.
 ↪iterrows() if not pd.isnull(row['LATITUD_Y']) and not pd.
 ↪isnull(row['LONGITUD_X'])]
HeatMap(heat_data).add_to(mapa)
mapa.save('mapa_calor.html')
mapa
```

[33]:
```
<folium.folium.Map at 0x2bb8d6f1d10>
```

Se analiza más detalladamente el mapa de calor relacionado con accidentes de tránsito en Ecuador:

- El mapa muestra las áreas con mayor incidencia de accidentes de tránsito. Las zonas en rojo indican una alta concentración de accidentes, mientras que las áreas en azul y verde tienen menos incidentes. Esto puede ayudar a identificar las regiones más problemáticas.
- Las áreas cercanas a las principales ciudades o rutas de alto tráfico suelen mostrar más accidentes. Las carreteras nacionales y las zonas urbanas son puntos clave que podrían requerir mayor vigilancia y medidas de seguridad.
- Factores contribuyentes como la infraestructura (calidad de las carreteras, señalización e iluminación) puede influir en la frecuencia de accidentes.
- Factores como la velocidad, el uso de alcohol y la distracción son cruciales.
- El clima también juega un papel importante, ya que condiciones adversas pueden aumentar el riesgo de accidentes.

Se define el modelo de clustering utilizando DBSCAN, estableciendo los parámetros de eps (distancia máxima entre dos muestras para que se consideren en el mismo vecindario) y min_samples (número mínimo de muestras en un vecindario para que se considere un núcleo). Posteriormente, se aplican estos parámetros al conjunto de coordenadas de latitud y longitud del DataFrame, lo que permite identificar los clusters de accidentes. Luego, se añaden las etiquetas de los clusters resultantes al DataFrame, donde cada punto se clasifica en un cluster o se identifica como ruido (etiquetado como -1). Finalmente, se filtran los datos para conservar únicamente los clusters relevantes, excluyendo aquellos puntos que no pertenecen a ningún cluster y que son considerados ruido, lo que facilita el análisis posterior de las áreas con alta densidad de accidentes.

[34]:
```
scaler = StandardScaler()
coords_scaled = scaler.fit_transform(df[['LATITUD_Y', 'LONGITUD_X']])

dbscan = DBSCAN(eps=0.012, min_samples=45)
clustering = dbscan.fit(coords_scaled)

df['cluster'] = clustering.labels_

filtered_coords = coords_scaled[clustering.labels_ != -1]
filtered_labels = clustering.labels_[clustering.labels_ != -1]

silhouette = silhouette_score(filtered_coords, filtered_labels)
davies_bouldin = davies_bouldin_score(filtered_coords, filtered_labels)

n_clusters = len(set(df['cluster'])) - (1 if -1 in df['cluster'].values else 0)
print(f"Número de clústeres (excluyendo ruido): {n_clusters}")

n_noise = np.sum(df['cluster'] == -1)
n_points = len(df)
percentage_assigned = (n_points - n_noise) / n_points * 100
print(f"Porcentaje de puntos asignados a clústeres (excluyendo ruido):␣
  ↪{percentage_assigned:.2f}%")

print(f'Silhouette Score (sin ruido): {silhouette}')
```

```
print(f'Davies-Bouldin Index (sin ruido): {davies_bouldin}')
```

```
Número de clústeres (excluyendo ruido): 131
Porcentaje de puntos asignados a clústeres (excluyendo ruido): 83.54%
Silhouette Score (sin ruido): 0.43919231762703476
Davies-Bouldin Index (sin ruido): 0.31052723918310077
```

El código utiliza el algoritmo de clustering DBSCAN para identificar grupos de accidentes basándose en sus coordenadas geográficas. Primero, se normalizan las coordenadas de latitud y longitud utilizando StandardScaler, y luego se aplica DBSCAN con parámetros específicos para determinar los clústeres. La salida muestra que se identificaron 131 clústeres, lo que indica una diversidad significativa en la distribución de los accidentes. Además, el 83.54% de los puntos se asignaron a clústeres, lo que quiere decir que la mayoría de los accidentes tienen una ubicación geográfica coherente.

Las métricas de evaluación del clustering, como el Silhouette Score de aproximadamente 0.44, indican una separación moderada entre los clústeres, mientras que el índice Davies-Bouldin de 0.31 sugiere que los clústeres están relativamente bien separados en comparación con su dispersión interna.

[35]:
```
accidentes_por_cluster = df.groupby('cluster').agg({
    'FALLECIDOS': 'sum',
    'LESIONADOS': 'sum'
}).reset_index()

accidentes_por_cluster
```

[35]:

| | cluster | FALLECIDOS | LESIONADOS |
|---|---|---|---|
| 0 | -1 | 6943 | 26936 |
| 1 | 0 | 1967 | 20786 |
| 2 | 1 | 1834 | 38729 |
| 3 | 2 | 318 | 3567 |
| 4 | 3 | 91 | 984 |
| .. | … | … | … |
| 127 | 126 | 10 | 41 |
| 128 | 127 | 1 | 64 |
| 129 | 128 | 9 | 49 |
| 130 | 129 | 5 | 52 |
| 131 | 130 | 7 | 48 |

[132 rows x 3 columns]

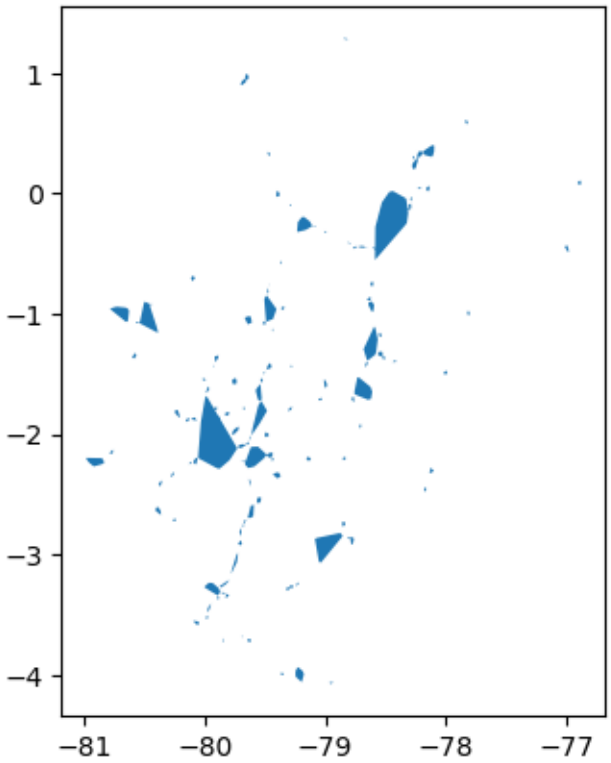
[36]:
```
print(df['cluster'].unique())

# Verificar el número de ocurrencias de cada cluster
df['cluster'].value_counts()
```

```
[  0   1   2   3   4   5   6   7   8   9  10  11  -1  12  13  14  15  17
  16  39  56  18  19  20 107  21  54  22  23  24  25  26  27  28  29  30
```

```
 31  72  32  33  34  35  74 119  43  36  37  38  53  40  41  42  64 113
 44  45  46  47  48  49  50  51  92  52 124  55 105 114  57  58 116  59
 60  79  68  61 117 109  62  63 110 103  65  66  82  67 106  69  70  71
 73  75  76  88  91  77 104 102 111 120  98 128  78  83  81 121 127  80
101  86  84  85  90  99  87  89  93  94  95  96  97 100 126 108 118 115
129 112 122 125 123 130]
```

[36]:
```
cluster
 1      41003
 0      35776
-1      27443
 8       6502
 9       6181
        …
 129       45
 128       45
 130       45
 109       42
 124       39
Name: count, Length: 132, dtype: int64
```

[37]:
```
clusters = df[df['cluster'] != -1]

gdf_clusters = gpd.GeoDataFrame(clusters, geometry=gpd.
 ↪points_from_xy(clusters['LONGITUD_X'], clusters['LATITUD_Y']))

poligonos = gdf_clusters.dissolve(by='cluster').convex_hull

gdf_poligonos = gpd.GeoDataFrame(poligonos, geometry=poligonos.geometry)

gdf_poligonos.plot()
plt.show()
```




Se crea un GeoDataFrame a partir de los datos filtrados de los clusters, utilizando las coordenadas de longitud y latitud para definir la geometría de los puntos. Luego, se agrupan los datos por cluster y se utilizan las funciones de disolución y el envolvente convexo (convex hull) para crear polígonos que abarcan cada uno de los clusters, lo que proporciona una representación visual de las áreas densamente agrupadas de accidentes. A continuación, se genera otro GeoDataFrame que contiene estos polígonos como geometría, facilitando así su visualización. Finalmente, se muestran los polígonos en un gráfico, lo que permite observar claramente la distribución espacial de los clusters de accidentes y su extensión geográfica, lo que es crucial para el análisis geoespacial y la identificación de hotspots.

En el gráfico se pueden observar polígonos que son las áreas donde existen mayor cantidad de accidentes.

[38]:
```
for _, row in gdf_poligonos.iterrows():
    # Convertir las coordenadas del polígono en una lista de latitudes y␣
 ↪longitudes
    folium.Polygon(locations=[(point[1], point[0]) for point in row.geometry.
 ↪exterior.coords],
                   color='blue', fill=True, fill_opacity=0.2).add_to(mapa)

mapa.save('mapa_poligonos.html')
mapa
```

[38]:
```
<folium.folium.Map at 0x2bb8d6f1d10>
```

Se itera sobre cada fila del GeoDataFrame que contiene los polígonos de los clusters, y para cada polígono se convierten sus coordenadas en una lista de pares de latitud y longitud, ajustando el orden de los puntos para que sean compatibles con el formato requerido por Folium. Luego, se añade cada polígono al mapa utilizando la clase Polygon, definiendo su ubicación, color y opacidad de relleno, lo que permite visualizar de forma clara las áreas de alta densidad de accidentes. Finalmente, se guarda el mapa resultante como un archivo HTML, denominado mapa_poligonos.html, lo que permite a los usuarios interactuar con el mapa en un navegador web y observar la distribución espacial de los clusters de accidentes a través de las áreas resaltadas.

Se puede observar un mapa de calor con los polígonos donde hay mayor cantidad de accidentes.

[39]:
```
gdf_poligonos['area_m2'] = gdf_poligonos.geometry.area * 10**6  # Convertir de␣
↪grados a metros cuadrados (aproximado)
gdf_poligonos[['area_m2']]
```

[39]:

| cluster | area_m2 |
|---|---|
| 0 | 92889.496462 |
| 1 | 110451.268803 |
| 2 | 10044.632252 |
| 3 | 5917.467670 |
| 4 | 11155.565505 |
| … | … |
| 126 | 156.477332 |
| 127 | 156.292586 |
| 128 | 126.243488 |
| 129 | 28.622563 |
| 130 | 128.066628 |

[131 rows x 1 columns]

Se calcula el área de cada polígono en el GeoDataFrame gdf_poligonos multiplicando la geometría del polígono por un factor de conversión de (10^6) para convertir los resultados de grados cuadrados a metros cuadrados, teniendo en cuenta que el área calculada en sistemas de coordenadas geográficas es una aproximación. Posteriormente, se muestra un resumen de las áreas calculadas en un DataFrame, permitiendo observar las dimensiones de cada polígono correspondiente a los clusters, lo que puede ser útil para identificar no solo la concentración de accidentes, sino también para evaluar la magnitud de la problemática en términos de área afectada en cada cluster.

Los resultados muestran una amplia gama de áreas, desde más de 73000 $m^2$ para algunos polígonos grandes hasta menos de 10 $m^2$ para los más pequeños.

[40]:
```
gdf_poligonos['area_m2'].describe()
```

[40]:
```
count       131.000000
mean       3524.656898
std       13099.294493
```

```
min          26.787247
25%         311.592344
50%         541.251698
75%        1189.418933
max      110451.268803
Name: area_m2, dtype: float64
```

Se calculan las estadísticas descriptivas del área de los polígonos en metros cuadrados utilizando el método describe() de pandas, lo que proporciona un resumen completo que incluye el conteo, la media, la desviación estándar, el mínimo, el máximo y los percentiles del área.

Aunque la media es de aproximadamente 2489.78 $m^2$, la gran desviación estándar sugiere que hay una considerable variabilidad en el tamaño de los clusters. A continuación, se genera un histograma que visualiza la distribución de las áreas, mostrando la frecuencia de los diferentes rangos de tamaño de los polígonos.

La imagen muestra un histograma que ilustra la distribución de áreas de polígonos. El eje x representa los valores de área, mientras que el eje y indica la frecuencia de esos valores. Se puede observar un pico significativo en el extremo inferior del rango de áreas, lo que quiere decir que la mayoría de los polígonos tienen áreas pequeñas. Muy pocos polígonos parecen tener áreas más grandes, como lo indican los escasos recuentos de frecuencia en los rangos superiores. Tiene una distribución sesgada a la derecha, donde la mayoría de los puntos de datos se agrupan alrededor de valores más pequeños.

[41]:
```
geometry = [Point(xy) for xy in zip(df['LONGITUD_X'], df['LATITUD_Y'])]
gdf = gpd.GeoDataFrame(df, geometry=geometry)

gdf.head()
```

[41]:

| | ANIO | LESIONADOS | FALLECIDOS | LATITUD_Y | LONGITUD_X | \ |
|---|---|---|---|---|---|---|
| 0 | 2017 | 1 | 0 | -0.083501 | -78.417742 | |
| 1 | 2017 | 1 | 0 | -2.246682 | -79.897754 | |
| 2 | 2017 | 1 | 0 | -0.253881 | -79.217405 | |
| 3 | 2017 | 0 | 0 | -0.116059 | -78.464188 | |
| 4 | 2017 | 0 | 0 | -0.239721 | -78.512058 | |

| | PROVINCIA | CANTON | \ |
|---|---|---|---|
| 0 | PICHINCHA | QUITO | |
| 1 | GUAYAS | GUAYAQUIL | |
| 2 | SANTO DOMINGO DE LOS TSACHILAS | SANTO DOMINGO | |
| 3 | PICHINCHA | QUITO | |
| 4 | PICHINCHA | QUITO | |

| | PARROQUIA | ZONA_PLANIFICACION | ZONA | … | FURGONETA | \ |
|---|---|---|---|---|---|---|
| 0 | CALDERON (CARAPUNGO) | ZONA 9 | RURAL | … | 0 | |
| 1 | GUAYAQUIL | ZONA 8 | URBANA | … | 0 | |
| 2 | SANTO DOMINGO DE LOS COLORADOS | ZONA 4 | URBANA | … | 0 | |
| 3 | QUITO | ZONA 9 | RURAL | … | 0 | |
| 4 | QUITO | ZONA 9 | URBANA | … | 0 | |

```
   MOTOCICLETA NO_IDENTIFICADO SCOOTER_ELECTRICO TRICIMOTO  \
0            0               1                 0         0
1            0               0                 0         0
2            1               0                 0         0
3            0               0                 0         0
4            0               1                 0         0

  VEHICULO_DEPORTIVO_UTILITARIO SUMA_DE_VEHICULOS DIA_SEMANA  cluster  \
0                             0                 1     Sunday        0
1                             0                 1     Sunday        1
2                             0                 1     Sunday        2
3                             0                 1     Sunday        0
4                             0                 1     Sunday        0

                     geometry
0   POINT (-78.41774 -0.0835)
1  POINT (-79.89775 -2.24668)
2   POINT (-79.2174 -0.25388)
3  POINT (-78.46419 -0.11606)
4  POINT (-78.51206 -0.23972)

[5 rows x 35 columns]
```

Se crea un GeoDataFrame utilizando la biblioteca geopandas, donde se construye la geometría de cada punto a partir de las coordenadas de longitud y latitud del DataFrame original. Posteriormente, el nuevo GeoDataFrame, gdf, se genera al combinar el DataFrame original con la geometría de los puntos, permitiendo así el análisis y la visualización geoespacial de los datos. La verificación de la estructura del GeoDataFrame se lleva a cabo al mostrar las primeras filas del mismo, lo que permite observar que las columnas originales, junto con la nueva columna de geometría, están correctamente integradas.

[42]:
```
muestra = gdf.groupby('cluster', group_keys=False).apply(lambda x: x.
 ↪sample(frac=0.1, random_state=1))

print(f"Número total de registros en la muestra: {len(muestra)}")
```

```
Número total de registros en la muestra: 16668
```

Se lleva a cabo un muestreo estratificado utilizando la función groupby del GeoDataFrame, agrupando los datos por la columna cluster. A través de la aplicación de una función lambda, se toma una muestra del 10% de los registros de cada cluster, lo que permite asegurar que todos los grupos estén representados en la muestra final. El número total de registros en la muestra se imprime, revelando que se han seleccionado 16672 registros, lo que proporciona una base suficiente para realizar análisis y visualizaciones adicionales sin comprometer la representatividad de los datos.

[43]:
```
coords_muestra = np.array(list(zip(muestra.geometry.x, muestra.geometry.y)))
```

```
w_muestra = weights.KNN(coords_muestra, k=10)
```

Se extraen las coordenadas de la muestra en forma de un array de NumPy, utilizando zip para combinar las longitudes y latitudes de los puntos geoespaciales, lo que permite una manipulación más sencilla de los datos. A continuación, se crea una matriz de pesos espaciales utilizando el método KNN (K-Nearest Neighbors) de la biblioteca libpysal, especificando un parámetro k de 5, que determina la cantidad de vecinos más cercanos a considerar para cada punto. La elección de k puede ajustarse según las características del conjunto de datos y los objetivos del análisis, lo que proporciona flexibilidad en el estudio de la distribución geográfica de los accidentes.

[44]:
```
print(muestra['FALLECIDOS'].describe())
```

```
count    16668.000000
mean         0.090593
std          0.335961
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max         11.000000
Name: FALLECIDOS, dtype: float64
```

[45]:
```
g_muestra = G(muestra['FALLECIDOS'], w_muestra)

muestra['FALLECIDOS_scaled'] = (muestra['FALLECIDOS'] - muestra['FALLECIDOS'].
  ↪min()) / (muestra['FALLECIDOS'].max() - muestra['FALLECIDOS'].min())

g_muestra = G(muestra['FALLECIDOS_scaled'], w_muestra)

print(f"Índice de Getis-Ord Gi*: {g_muestra.G}, p-value: {g_muestra.p_sim}")
```

```
Índice de Getis-Ord Gi*: 0.001327432462922759, p-value: 0.001
```

[46]:
```
print(muestra['FALLECIDOS'].describe())
```

```
count    16668.000000
mean         0.090593
std          0.335961
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max         11.000000
Name: FALLECIDOS, dtype: float64
```

Se calcula el índice de Getis-Ord Gi* utilizando el conjunto de datos de la muestra que contiene la variable de FALLECIDOS y la matriz de pesos espaciales creada previamente con el método KNN. El resultado, que se almacena en la variable g_muestra, incluye el valor del índice Gi* y el p-valor correspondiente. En este caso, se obtiene un índice de Getis-Ord Gi* de aproximadamente 0.00064

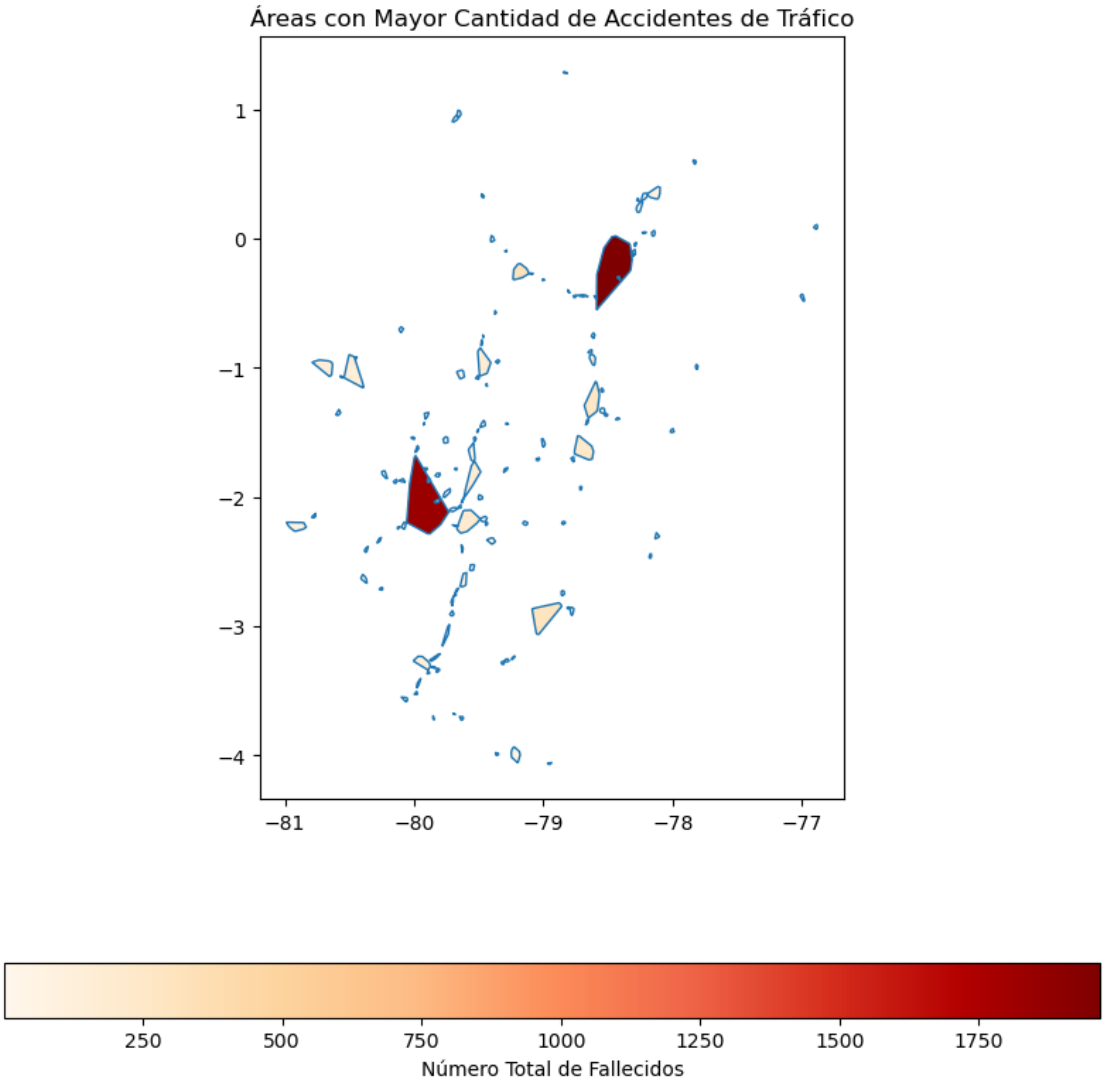
y un p-valor de 0.001, lo que quiere decir que hay una concentración significativa de fallecidos en ciertas áreas geográficas que supera lo que se esperaría en una distribución aleatoria.

[47]:
```
accidentes_por_poligono = gdf_clusters.groupby('cluster').agg({
    'FALLECIDOS': 'sum',
    'LESIONADOS': 'sum'
}).reset_index()

gdf_poligonos = gdf_poligonos.merge(accidentes_por_poligono, on='cluster',␣
 ↪how='left', suffixes=('', '_total'))
```

[48]:
```
gdf_poligonos_sorted = gdf_poligonos.sort_values(by='FALLECIDOS',␣
 ↪ascending=False)
print(gdf_poligonos_sorted[['cluster', 'FALLECIDOS', 'LESIONADOS']])
```

```
     cluster  FALLECIDOS  LESIONADOS
0          0        1967       20786
1          1        1834       38729
2          2         318        3567
9          9         301        5379
8          8         263        3567
..       …          …          …
121      121           4          37
120      120           3          44
123      123           3          46
114      114           3         107
127      127           1          64

[131 rows x 3 columns]
```

[49]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
gdf_poligonos.boundary.plot(ax=ax, linewidth=1)
gdf_poligonos.plot(column='FALLECIDOS', ax=ax, legend=True,
                   legend_kwds={'label': "Número Total de Fallecidos",
                                'orientation': "horizontal"},
                   cmap='OrRd')  # Usar un mapa de colores de rojo a naranja

plt.title('Áreas con Mayor Cantidad de Accidentes de Tráfico')
plt.show()
```




[50]:
```
analisis_hotspots = df.groupby('cluster').agg({
    'FALLECIDOS': 'sum',
    'LESIONADOS': 'sum',
    'HORA': lambda x: x.mode()[0]  # Obtener la hora más frecuente
}).reset_index()

analisis_hotspots
```

[50]:

| | cluster | FALLECIDOS | LESIONADOS | HORA |
|---|---|---|---|---|
| 0 | -1 | 6943 | 26936 | 19 |
| 1 | 0 | 1967 | 20786 | 7 |
| 2 | 1 | 1834 | 38729 | 19 |
| 3 | 2 | 318 | 3567 | 19 |

```
4          3          91          984     7
..       …          …           …    …
127      126          10           41    22
128      127           1           64    21
129      128           9           49    17
130      129           5           52    21
131      130           7           48     0

[132 rows x 4 columns]
```

Se agrupa el DataFrame por la columna cluster y se calculan varias estadísticas descriptivas utilizando el método agg(). El análisis revela que existen 136 clusters, cada uno con su respectivo total de fallecidos y lesionados, así como la hora más frecuente en que ocurrieron los accidentes. Por ejemplo, el cluster -1, que representa los puntos considerados como ruido o no pertenecientes a un cluster, tiene 7713 fallecidos y 30573 lesionados, con la hora más común de los accidentes siendo a las 19:30:00. En contraste, el cluster 1 presenta 1682 fallecidos y 36864 lesionados, con la hora más frecuente a las 16:00:00.

[51]:
```
accidentes_por_dia = df.groupby(df['FECHA'].dt.date).size().
  ↪reset_index(name='Número de Accidentes')

accidentes_por_dia
```

[51]:
```
            FECHA  Número de Accidentes
0      2017-01-01                   150
1      2017-01-02                    71
2      2017-01-03                    67
3      2017-01-04                    55
4      2017-01-05                    68
…           …                     …
2672   2024-04-26                    46
2673   2024-04-27                    93
2674   2024-04-28                    93
2675   2024-04-29                    46
2676   2024-04-30                    30

[2677 rows x 2 columns]
```

[52]:
```
gdf_accidentes = gpd.GeoDataFrame(df, geometry=gpd.
  ↪points_from_xy(df['LONGITUD_X'], df['LATITUD_Y']))

gdf_accidentes.set_crs(epsg=4326, inplace=True)

if gdf_poligonos.crs is None:
    gdf_poligonos.set_crs(epsg=4326, inplace=True)
    print("CRS de gdf_poligonos definido como EPSG:4326")

if gdf_accidentes.crs != gdf_poligonos.crs:
```

```
    gdf_accidentes = gdf_accidentes.to_crs(gdf_poligonos.crs)

gdf_joined = gpd.sjoin(gdf_accidentes, gdf_poligonos, how='inner',␣
 ↪predicate='within', lsuffix='_acc', rsuffix='_poly')

columns = [col for col in gdf_joined.columns if isinstance(col, str)]

columns_to_drop = [col for col in columns if 'cluster' in col and col.
 ↪endswith('_right')]

gdf_joined = gdf_joined.drop(columns=columns_to_drop, errors='ignore')
```

```
CRS de gdf_poligonos definido como EPSG:4326
```

[53]:
```
gdf_accidentes = gpd.GeoDataFrame(df, geometry=gpd.
 ↪points_from_xy(df['LONGITUD_X'], df['LATITUD_Y']), crs='EPSG:4326')
gdf_poligonos = gpd.GeoDataFrame(gdf_poligonos, geometry=gdf_poligonos.
 ↪geometry, crs='EPSG:4326')
```

[54]:
```
print(gdf_joined.columns)
accidentes_por_poligono = gdf_joined.groupby('cluster__poly').size().
 ↪reset_index(name='num_accidentes')
gdf_poligonos = gdf_poligonos.merge(accidentes_por_poligono, left_on='cluster',␣
 ↪right_on='cluster__poly', how='left')
```

```
Index([                          'ANIO',                'LESIONADOS__acc',
                    'FALLECIDOS__acc',                     'LATITUD_Y',
                         'LONGITUD_X',                     'PROVINCIA',
                            'CANTON',                     'PARROQUIA',
                 'ZONA_PLANIFICACION',                          'ZONA',
                             'FECHA',                          'HORA',
                           'FERIADO',                 'CODIGO_CAUSA',
                    'CAUSA_PROBABLE',              'TIPO_DE_SINIESTRO',
                 'TIPO_DE_VEHICULO_1',                     'SERVICIO_1',
                          'AUTOMOVIL',                     'BICICLETA',
                                'BUS',                        'CAMION',
                          'CAMIONETA',                   'EMERGENCIAS',
                           'ESPECIAL',                     'FURGONETA',
                        'MOTOCICLETA',                'NO_IDENTIFICADO',
                  'SCOOTER_ELECTRICO',                     'TRICIMOTO',
       'VEHICULO_DEPORTIVO_UTILITARIO',              'SUMA_DE_VEHICULOS',
                         'DIA_SEMANA',                  'cluster__acc',
                           'geometry',                   'index__poly',
                      'cluster__poly',                             0,
                           'area_m2',                'FALLECIDOS__poly',
                  'LESIONADOS__poly'],
      dtype='object')
```

[55]:
```
print(gdf_accidentes.crs)
print(gdf_poligonos.crs)

if gdf_accidentes.crs != gdf_poligonos.crs:
    gdf_accidentes = gdf_accidentes.to_crs(gdf_poligonos.crs)
```

```
EPSG:4326
EPSG:4326
```

[56]:
```
top_poligonos = gdf_poligonos.nlargest(10, 'num_accidentes')

print(top_poligonos[['num_accidentes', 'area_m2']])
```

```
    num_accidentes        area_m2
1            41685  110451.268803
0            36127   92889.496462
8             6601   18688.214526
9             6312   28384.924555
2             5137   10044.632252
11            4253   19491.161547
4             4015   11155.565505
13            3260   20002.320114
41            3192    5499.029401
17            2835    7275.297276
```

[57]:
```
mapa_denso = folium.Map(location=[df['LATITUD_Y'].mean(), df['LONGITUD_X'].
 ↪mean()], zoom_start=12)

for _, row in top_poligonos.iterrows():
    folium.Polygon(
        locations=[(point[1], point[0]) for point in row.geometry.exterior.
 ↪coords],
        color='red',
        fill=True,
        fill_opacity=0.5,
        popup=f"Accidentes: {row['num_accidentes']}, Área: {row['area_m2']:.2f}␣
 ↪m²"
    ).add_to(mapa_denso)

mapa_denso.save('GYE1.html')
mapa_denso
```

[57]:
```
<folium.folium.Map at 0x2bbc9c48510>
```

[58]:
```
gdf_poligonos['densidad_accidentes'] = gdf_poligonos['num_accidentes'] /␣
 ↪gdf_poligonos['area_m2']

top_densos = gdf_poligonos.nlargest(10, 'densidad_accidentes')
```

```
print(top_densos[['densidad_accidentes', 'num_accidentes', 'area_m2']])
```

```
     densidad_accidentes  num_accidentes       area_m2
129             1.222812              35     28.622563
124             1.157267              31     26.787247
121             0.638572              39     61.073786
67              0.619655              74    119.421308
41              0.580466            3192   5499.029401
105             0.569723             112    196.586666
112             0.536932             121    225.354321
2               0.511417            5137  10044.632252
104             0.484496              33     68.112007
109             0.463626              32     69.021152
```

## 0.2 QUITO

[59]:
```
df_parroquia_uio['LATITUD_Y'] = pd.to_numeric(df_parroquia_uio['LATITUD_Y'],␣
 ↪errors='coerce')
df_parroquia_uio['LONGITUD_X'] = pd.to_numeric(df_parroquia_uio['LONGITUD_X'],␣
 ↪errors='coerce')
```

[60]:
```
centro_quito = [-0.2313, -78.5285]  # Latitud y longitud aproximada del centro␣
 ↪de Quito

quito_map = folium.Map(location=centro_quito, zoom_start=12)

df_parroquia_uio = df_parroquia_uio.dropna(subset=['LATITUD_Y', 'LONGITUD_X'])

heat_data = [[row['LATITUD_Y'], row['LONGITUD_X']] for index, row in␣
 ↪df_parroquia_uio.iterrows()]

HeatMap(heat_data).add_to(quito_map)

quito_map.save('mapa_calor_quito.html')
quito_map
```

[60]:
```
<folium.folium.Map at 0x2bbc9c461d0>
```

[61]:
```
scaler = StandardScaler()
coords_scaled = scaler.fit_transform(df_parroquia_uio[['LATITUD_Y',␣
 ↪'LONGITUD_X']])

dbscan = DBSCAN(eps=0.035, min_samples=50)
clustering = dbscan.fit(coords_scaled)
df_parroquia_uio['cluster'] = clustering.labels_
```

```
filtered_coords = coords_scaled[clustering.labels_ != -1]
filtered_labels = clustering.labels_[clustering.labels_ != -1]

silhouette = silhouette_score(filtered_coords, filtered_labels)
davies_bouldin = davies_bouldin_score(filtered_coords, filtered_labels)

n_clusters = len(set(df_parroquia_uio['cluster'])) - (1 if -1 in␣
 ↪df_parroquia_uio['cluster'].values else 0)
print(f"Número de clústeres (excluyendo ruido): {n_clusters}")

n_noise = np.sum(df_parroquia_uio['cluster'] == -1)
n_points = len(df_parroquia_uio)
percentage_assigned = (n_points - n_noise) / n_points * 100

print(f"Porcentaje de puntos asignados a clústeres (excluyendo ruido):
 ↪{percentage_assigned:.2f}%")
print(f'Silhouette Score (sin ruido): {silhouette}')
print(f'Davies-Bouldin Index (sin ruido): {davies_bouldin}')
```

```
Número de clústeres (excluyendo ruido): 116
Porcentaje de puntos asignados a clústeres (excluyendo ruido):62.14%
Silhouette Score (sin ruido): 0.33633185534200283
Davies-Bouldin Index (sin ruido): 0.5392981205788716
```

[62]:
```
scaler = StandardScaler()
coords_scaled = scaler.fit_transform(df_parroquia_uio[['LATITUD_Y',␣
 ↪'LONGITUD_X']])

dbscan = DBSCAN(eps=0.03, min_samples=50)
clustering = dbscan.fit(coords_scaled)
df_parroquia_uio['cluster'] = clustering.labels_

filtered_coords = coords_scaled[clustering.labels_ != -1]
filtered_labels = clustering.labels_[clustering.labels_ != -1]

silhouette = silhouette_score(filtered_coords, filtered_labels)
davies_bouldin = davies_bouldin_score(filtered_coords, filtered_labels)

n_clusters = len(set(df_parroquia_uio['cluster'])) - (1 if -1 in␣
 ↪df_parroquia_uio['cluster'].values else 0)

n_noise = np.sum(df_parroquia_uio['cluster'] == -1)
n_points = len(df_parroquia_uio)
percentage_assigned = (n_points - n_noise) / n_points * 100

print(f"Número de clústeres (excluyendo ruido): {n_clusters}")
```

```
print(f"Porcentaje de puntos asignados a clústeres (excluyendo ruido):␣
  ↪{percentage_assigned:.2f}%")
print(f'Silhouette Score (sin ruido): {silhouette}')
print(f'Davies-Bouldin Index (sin ruido): {davies_bouldin}')
```

```
Número de clústeres (excluyendo ruido): 121
Porcentaje de puntos asignados a clústeres (excluyendo ruido): 52.24%
Silhouette Score (sin ruido): 0.5811086461417003
Davies-Bouldin Index (sin ruido): 0.4113963755719634
```

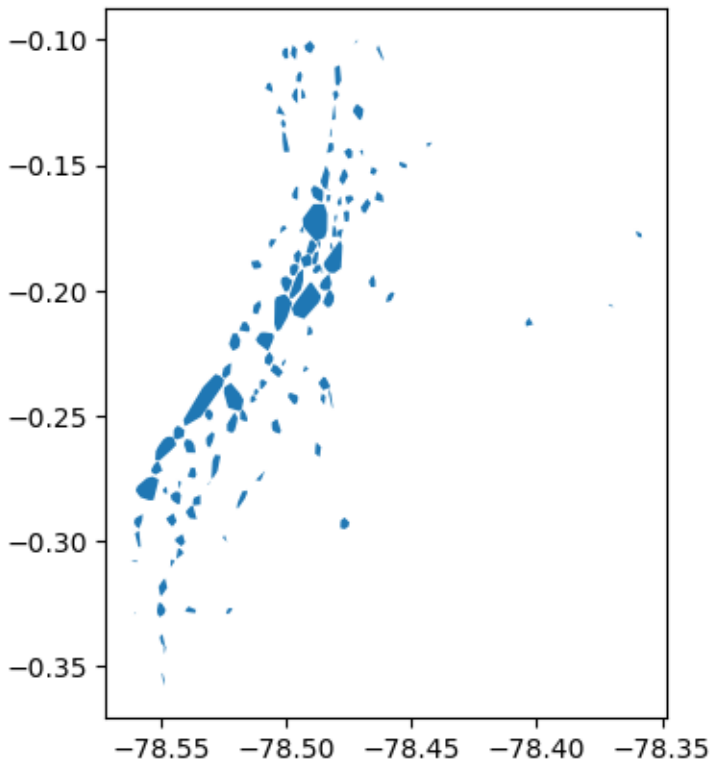
[63]:
```
accidentes_por_cluster = df_parroquia_uio.groupby('cluster').agg({
'FALLECIDOS': 'sum',
'LESIONADOS': 'sum'
}).reset_index()
# Mostrar la tabla
accidentes_por_cluster
```

[63]:

| | cluster | FALLECIDOS | LESIONADOS |
|---|---|---|---|
| 0 | -1 | 824 | 8355 |
| 1 | 0 | 13 | 61 |
| 2 | 1 | 6 | 74 |
| 3 | 2 | 6 | 91 |
| 4 | 3 | 5 | 170 |
| .. | … | … | … |
| 117 | 116 | 1 | 27 |
| 118 | 117 | 4 | 22 |
| 119 | 118 | 1 | 34 |
| 120 | 119 | 0 | 0 |
| 121 | 120 | 5 | 26 |

[122 rows x 3 columns]

[64]:
```
clusters = df_parroquia_uio[df_parroquia_uio['cluster'] != -1]
gdf_clusters = gpd.GeoDataFrame(clusters, geometry=gpd.
  ↪points_from_xy(clusters['LONGITUD_X'], clusters['LATITUD_Y']))
poligonos = gdf_clusters.dissolve(by='cluster').convex_hull
gdf_poligonos = gpd.GeoDataFrame(poligonos, geometry=poligonos.geometry)
gdf_poligonos.plot()
plt.show()
```




[65]:
```
gdf_poligonos['area_m2'] = gdf_poligonos.geometry.area * 10**6 # Convertir de␣
↪grados a metros cuadrados (aproximado)
gdf_poligonos[['area_m2']]
```

[65]:

| cluster | area_m2 |
|---|---|
| 0 | 4.165757 |
| 1 | 18.929323 |
| 2 | 14.116614 |
| 3 | 58.025534 |
| 4 | 92.380447 |
| … | … |
| 116 | 0.025701 |
| 117 | 1.378556 |
| 118 | 1.978941 |
| 119 | 0.141754 |
| 120 | 2.614161 |

[121 rows x 1 columns]

[66]:
```
geometry = [Point(xy) for xy in zip(df_parroquia_uio['LONGITUD_X'],␣
↪df_parroquia_uio['LATITUD_Y'])]
```

```
gdf = gpd.GeoDataFrame(df_parroquia_uio, geometry=geometry)
```

[67]:
```
muestra = gdf.groupby('cluster', group_keys=False).apply(lambda x: x.
  ↪sample(frac=0.1, random_state=1))
print(f"Número total de registros en la muestra: {len(muestra)}")
```

```
Número total de registros en la muestra: 2976
```

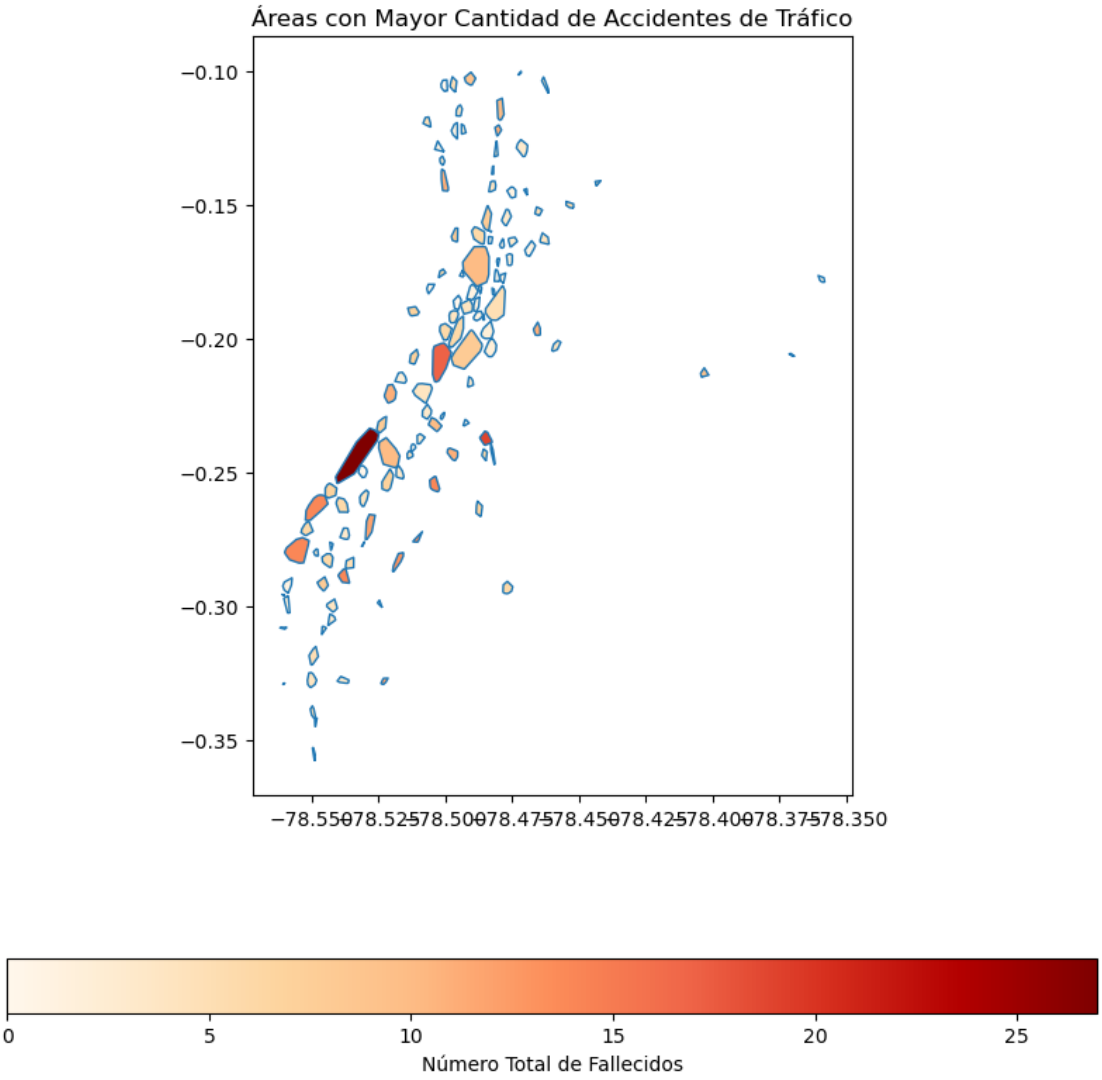
[68]:
```
coords_muestra = np.array(list(zip(muestra.geometry.x, muestra.geometry.y)))
w_muestra = weights.KNN(coords_muestra, k=10)
```

[69]:
```
g_muestra = G(muestra['FALLECIDOS'], w_muestra)
muestra['FALLECIDOS_scaled'] = (muestra['FALLECIDOS'] - muestra['FALLECIDOS'].
  ↪min()) / (muestra['FALLECIDOS'].max() - muestra['FALLECIDOS'].min())
g_muestra = G(muestra['FALLECIDOS_scaled'], w_muestra)
print(f"Índice de Getis-Ord Gi*: {g_muestra.G}, p-value: {g_muestra.p_sim}")
```

```
Índice de Getis-Ord Gi*: 0.005940113953206456, p-value: 0.001
```

[70]:
```
accidentes_por_poligono = gdf_clusters.groupby('cluster').agg({
'FALLECIDOS': 'sum',
'LESIONADOS': 'sum'
}).reset_index()

gdf_poligonos = gdf_poligonos.merge(accidentes_por_poligono, on='cluster',␣
  ↪how='left', suffixes=('', '_total'))
```

[71]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
gdf_poligonos.boundary.plot(ax=ax, linewidth=1)
gdf_poligonos.plot(column='FALLECIDOS', ax=ax, legend=True,␣
  ↪legend_kwds={'label': "Número Total de Fallecidos", 'orientation':␣
  ↪"horizontal"}, cmap='OrRd')
plt.title('Áreas con Mayor Cantidad de Accidentes de Tráfico')
plt.show()
```




[72]:
```
gdf_poligonos_sorted = gdf_poligonos.sort_values(by='FALLECIDOS',␣
↪ascending=False)

print(gdf_poligonos_sorted[['cluster', 'FALLECIDOS', 'LESIONADOS']])
```

```
     cluster  FALLECIDOS  LESIONADOS
35        35          27         412
28        28          19         134
40        40          17         286
96        96          14          59
15        15          14         292
..       ...         ...         ...
113      113           0           4
20        20           0          34
```

```
21        21           0          54
119      119           0           0
85        85           0          56

[121 rows x 3 columns]
```

[73]:
```
analisis_hotspots = df_parroquia_uio.groupby('cluster').agg({
    'FALLECIDOS': 'sum',
'LESIONADOS': 'sum',
'HORA': lambda x: x.mode()[0] # Obtener la hora más frecuente
}).reset_index()

analisis_hotspots
```

[73]:
```
     cluster  FALLECIDOS  LESIONADOS  HORA
0         -1         824        8355    19
1          0          13          61     5
2          1           6          74     7
3          2           6          91    18
4          3           5         170     8
..       …          …         …   …
117      116           1          27    19
118      117           4          22    21
119      118           1          34    12
120      119           0           0     3
121      120           5          26    22

[122 rows x 4 columns]
```

[74]:
```
accidentes_por_dia = df_parroquia_uio.groupby(df_parroquia_uio['FECHA'].dt.
 ↪date).size().reset_index(name='Número de Accidentes')

accidentes_por_dia
```

[74]:
```
           FECHA  Número de Accidentes
0     2017-01-01                    22
1     2017-01-02                     6
2     2017-01-03                    15
3     2017-01-04                    15
4     2017-01-05                    23
…          …                   …
2659  2024-04-26                     9
2660  2024-04-27                    21
2661  2024-04-28                    18
2662  2024-04-29                     7
2663  2024-04-30                     5

[2664 rows x 2 columns]
```

[75]:
```
gdf_accidentes = gpd.GeoDataFrame(df_parroquia_uio, geometry=gpd.
 ↪points_from_xy(df_parroquia_uio['LONGITUD_X'],␣
 ↪df_parroquia_uio['LATITUD_Y']))

gdf_accidentes.set_crs(epsg=4326, inplace=True)

if gdf_poligonos.crs is None:
    gdf_poligonos.set_crs(epsg=4326, inplace=True)
    print("CRS de gdf_poligonos definido como EPSG:4326")

if gdf_accidentes.crs != gdf_poligonos.crs:
    gdf_accidentes = gdf_accidentes.to_crs(gdf_poligonos.crs)

gdf_joined = gpd.sjoin(gdf_accidentes, gdf_poligonos,␣
 ↪how='inner',predicate='within', lsuffix='_acc', rsuffix='_poly')

columns = [col for col in gdf_joined.columns if isinstance(col, str)]

columns_to_drop = [col for col in columns if 'cluster' in col and col.
 ↪endswith('_right')]

gdf_joined = gdf_joined.drop(columns=columns_to_drop, errors='ignore')
```

```
CRS de gdf_poligonos definido como EPSG:4326
```

[76]:
```
gdf_accidentes = gpd.GeoDataFrame(df_parroquia_uio, geometry=gpd.
 ↪points_from_xy(df_parroquia_uio['LONGITUD_X'],␣
 ↪df_parroquia_uio['LATITUD_Y']), crs='EPSG:4326')
gdf_poligonos = gpd.GeoDataFrame(gdf_poligonos, geometry=gdf_poligonos.
 ↪geometry, crs='EPSG:4326')
```

[77]:
```
print(gdf_joined.columns)

accidentes_por_poligono = gdf_joined.groupby('cluster__poly').size().
 ↪reset_index(name='num_accidentes')

gdf_poligonos = gdf_poligonos.merge(accidentes_por_poligono, left_on='cluster',␣
 ↪right_on='cluster__poly', how='left')
```

```
Index([                        'ANIO',              'LESIONADOS__acc',
                  'FALLECIDOS__acc',                    'LATITUD_Y',
                       'LONGITUD_X',                    'PROVINCIA',
                           'CANTON',                    'PARROQUIA',
                'ZONA_PLANIFICACION',                         'ZONA',
                           'FECHA',                         'HORA',
                         'FERIADO',                'CODIGO_CAUSA',
                    'CAUSA_PROBABLE',           'TIPO_DE_SINIESTRO',
                'TIPO_DE_VEHICULO_1',                  'SERVICIO_1',
```

```
                          'AUTOMOVIL',                    'BICICLETA',
                                'BUS',                       'CAMION',
                          'CAMIONETA',                  'EMERGENCIAS',
                           'ESPECIAL',                    'FURGONETA',
                        'MOTOCICLETA',                'NO_IDENTIFICADO',
                  'SCOOTER_ELECTRICO',                      'TRICIMOTO',
       'VEHICULO_DEPORTIVO_UTILITARIO',             'SUMA_DE_VEHICULOS',
                         'DIA_SEMANA',                  'cluster__acc',
                           'geometry',                   'index__poly',
                      'cluster__poly',                               0,
                            'area_m2',               'FALLECIDOS__poly',
                 'LESIONADOS__poly'],
      dtype='object')
```

[78]:
```
top_poligonos = gdf_poligonos.nlargest(10, 'num_accidentes')
print(top_poligonos[['num_accidentes', 'area_m2']])
```

```
    num_accidentes     area_m2
17           835.0  109.293865
35           668.0  121.640137
4            663.0   92.380447
40           476.0   69.636921
15           451.0   58.287909
3            384.0   58.025534
56           356.0   57.623673
34           282.0   45.217574
24           271.0   32.109440
28           261.0   14.611803
```

[79]:
```
mapa_denso = folium.Map(location=[df_parroquia_uio['LATITUD_Y'].mean(),␣
 ↪df_parroquia_uio['LONGITUD_X'].mean()], zoom_start=12)

for _, row in top_poligonos.iterrows():
    folium.Polygon(
        locations=[(point[1], point[0]) for point in row.geometry.exterior.
 ↪coords],
        color='red',
        fill=True,
        fill_opacity=0.5,
        popup=f"Accidentes: {row['num_accidentes']}, Área: {row['area_m2']:.2f}␣
 ↪m²"
    ).add_to(mapa_denso)

mapa_denso.save('mapa_denso.html')
mapa_denso
```

[79]:
```
<folium.folium.Map at 0x2bb87369b50>
```

[80]:
```
gdf_poligonos['densidad_accidentes'] = gdf_poligonos['num_accidentes'] /␣
↪gdf_poligonos['area_m2']
top_densos = gdf_poligonos.nlargest(10, 'densidad_accidentes')
print(top_densos[['densidad_accidentes', 'num_accidentes', 'area_m2']])
```

```
     densidad_accidentes  num_accidentes   area_m2
67            350.918573            61.0  0.173829
21            136.364884            77.0  0.564662
114           102.377309            38.0  0.371176
82             66.212466            53.0  0.800453
49             43.319777            83.0  1.915984
64             32.891990            52.0  1.580932
31             32.851706           188.0  5.722686
81             29.205122           101.0  3.458298
112            27.424195            43.0  1.567959
93             26.082720            50.0  1.916978
```

[81]:
```
df_parroquia_gye['LATITUD_Y'] = pd.to_numeric(df_parroquia_gye['LATITUD_Y'],␣
↪errors='coerce')
df_parroquia_gye['LONGITUD_X'] = pd.to_numeric(df_parroquia_gye['LONGITUD_X'],␣
↪errors='coerce')
```

[82]:
```
scaler = StandardScaler()
coords_scaled = scaler.fit_transform(df_parroquia_uio[['LATITUD_Y',␣
↪'LONGITUD_X']])

dbscan = DBSCAN(eps=0.035, min_samples=50)
clustering = dbscan.fit(coords_scaled)
df_parroquia_uio['cluster'] = clustering.labels_

filtered_coords = coords_scaled[clustering.labels_ != -1]
filtered_labels = clustering.labels_[clustering.labels_ != -1]

silhouette = silhouette_score(filtered_coords, filtered_labels)
davies_bouldin = davies_bouldin_score(filtered_coords, filtered_labels)

n_clusters = len(set(df_parroquia_uio['cluster'])) - (1 if -1 in␣
↪df_parroquia_uio['cluster'].values else 0)
print(f"Número de clústeres (excluyendo ruido): {n_clusters}")

n_noise = np.sum(df_parroquia_uio['cluster'] == -1)
n_points = len(df_parroquia_uio)
percentage_assigned = (n_points - n_noise) / n_points * 100

print(f"Porcentaje de puntos asignados a clústeres (excluyendo ruido):
↪{percentage_assigned:.2f}%")
print(f'Silhouette Score (sin ruido): {silhouette}')
```

```
print(f'Davies-Bouldin Index (sin ruido): {davies_bouldin}')
```

```
Número de clústeres (excluyendo ruido): 116
Porcentaje de puntos asignados a clústeres (excluyendo ruido):62.14%
Silhouette Score (sin ruido): 0.33633185534200283
Davies-Bouldin Index (sin ruido): 0.5392981205788716
```

# 1 GUAYAQUIL

[83]:
```
df_parroquia_gye['LATITUD_Y'] = pd.to_numeric(df_parroquia_gye['LATITUD_Y'],␣
 ↪errors='coerce')
df_parroquia_gye['LONGITUD_X'] = pd.to_numeric(df_parroquia_gye['LONGITUD_X'],␣
 ↪errors='coerce')
```

[84]:
```
guayaquil_map = folium.Map(location=[-2.1700, -79.9221], zoom_start=12)  #␣
 ↪Coordenadas aproximadas de Guayaquil

df_parroquia_gye = df_parroquia_gye.dropna(subset=['LATITUD_Y', 'LONGITUD_X'])

heat_data = [[row['LATITUD_Y'], row['LONGITUD_X']] for index, row in␣
 ↪df_parroquia_gye.iterrows()]

HeatMap(heat_data).add_to(guayaquil_map)

guayaquil_map.save('mapa_calor_guayaquil.html')
guayaquil_map
```

[84]:
```
<folium.folium.Map at 0x2bb988bac10>
```

[85]:
```
df_parroquia_gye['LATITUD_Y'] = pd.to_numeric(df_parroquia_gye['LATITUD_Y'],␣
 ↪errors='coerce')
df_parroquia_gye['LONGITUD_X'] = pd.to_numeric(df_parroquia_gye['LONGITUD_X'],␣
 ↪errors='coerce')

df_parroquia_gye = df_parroquia_gye.dropna(subset=['LATITUD_Y', 'LONGITUD_X'])

scaler = StandardScaler()
coords_scaled = scaler.fit_transform(df_parroquia_gye[['LATITUD_Y',␣
 ↪'LONGITUD_X']])

dbscan = DBSCAN(eps=0.04, min_samples=40)
clustering = dbscan.fit(coords_scaled)

df_parroquia_gye['cluster'] = clustering.labels_

filtered_coords = coords_scaled[clustering.labels_ != -1]
```

```
filtered_labels = clustering.labels_[clustering.labels_ != -1]

silhouette = silhouette_score(filtered_coords, filtered_labels)
davies_bouldin = davies_bouldin_score(filtered_coords, filtered_labels)

n_clusters = len(set(df_parroquia_gye['cluster'])) - (1 if -1 in␣
 ↪df_parroquia_gye['cluster'].values else 0)

n_noise = np.sum(df_parroquia_gye['cluster'] == -1)
n_points = len(df_parroquia_gye)
percentage_assigned = (n_points - n_noise) / n_points * 100

print(f"Número de clústeres (excluyendo ruido): {n_clusters}")
print(f"Porcentaje de puntos asignados a clústeres (excluyendo ruido):␣
 ↪{percentage_assigned:.2f}%")
print(f'Silhouette Score (sin ruido): {silhouette}')
print(f'Davies-Bouldin Index (sin ruido): {davies_bouldin}')
```

```
Número de clústeres (excluyendo ruido): 92
Porcentaje de puntos asignados a clústeres (excluyendo ruido): 50.91%
Silhouette Score (sin ruido): 0.33953880819514126
Davies-Bouldin Index (sin ruido): 0.46360031632855636
```

[86]:
```
accidentes_por_cluster = df_parroquia_uio.groupby('cluster').agg({
'FALLECIDOS': 'sum',
'LESIONADOS': 'sum'
}).reset_index()

accidentes_por_cluster
```

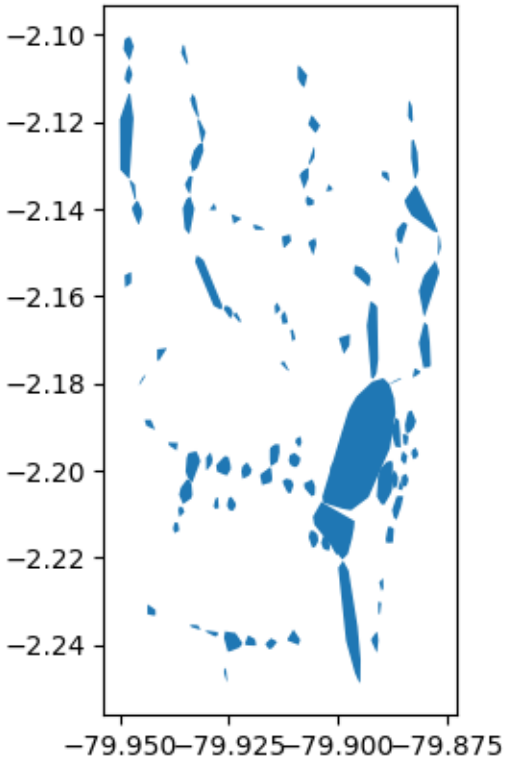
[86]:

| | cluster | FALLECIDOS | LESIONADOS |
|---|---|---|---|
| 0 | -1 | 726 | 6705 |
| 1 | 0 | 13 | 62 |
| 2 | 1 | 13 | 95 |
| 3 | 2 | 7 | 85 |
| 4 | 3 | 15 | 76 |
| .. | … | … | … |
| 112 | 111 | 4 | 22 |
| 113 | 112 | 3 | 39 |
| 114 | 113 | 0 | 3 |
| 115 | 114 | 1 | 29 |
| 116 | 115 | 0 | 2 |

[117 rows x 3 columns]

[87]:
```
clusters = df_parroquia_gye[df_parroquia_gye['cluster'] != -1]
```

```
gdf_clusters = gpd.GeoDataFrame(clusters, geometry=gpd.
  ↪points_from_xy(clusters['LONGITUD_X'], clusters['LATITUD_Y']))
poligonos = gdf_clusters.dissolve(by='cluster').convex_hull
gdf_poligonos = gpd.GeoDataFrame(poligonos, geometry=poligonos.geometry)
gdf_poligonos.plot()
plt.show()
```




[88]:
```
gdf_poligonos['area_m2'] = gdf_poligonos.geometry.area * 10**6 # Convertir de␣
  ↪grados a metros cuadrados (aproximado)
gdf_poligonos[['area_m2']]
```

[88]:

| cluster | area_m2 |
|---|---|
| 0 | 11.577693 |
| 1 | 302.160649 |
| 2 | 6.699105 |
| 3 | 14.592137 |
| 4 | 15.862259 |
| … | … |
| 87 | 2.458254 |
| 88 | 1.310353 |
| 89 | 2.194736 |

```
90          2.218768
91          1.550603

[92 rows x 1 columns]
```

[89]:
```
geometry = [Point(xy) for xy in zip(df_parroquia_gye['LONGITUD_X'],␣
 ↪df_parroquia_gye['LATITUD_Y'])]
gdf = gpd.GeoDataFrame(df_parroquia_gye, geometry=geometry)
```

[90]:
```
muestra = gdf.groupby('cluster', group_keys=False).apply(lambda x: x.
 ↪sample(frac=0.1, random_state=1))

print(f"Número total de registros en la muestra: {len(muestra)}")
```

```
Número total de registros en la muestra: 2551
```

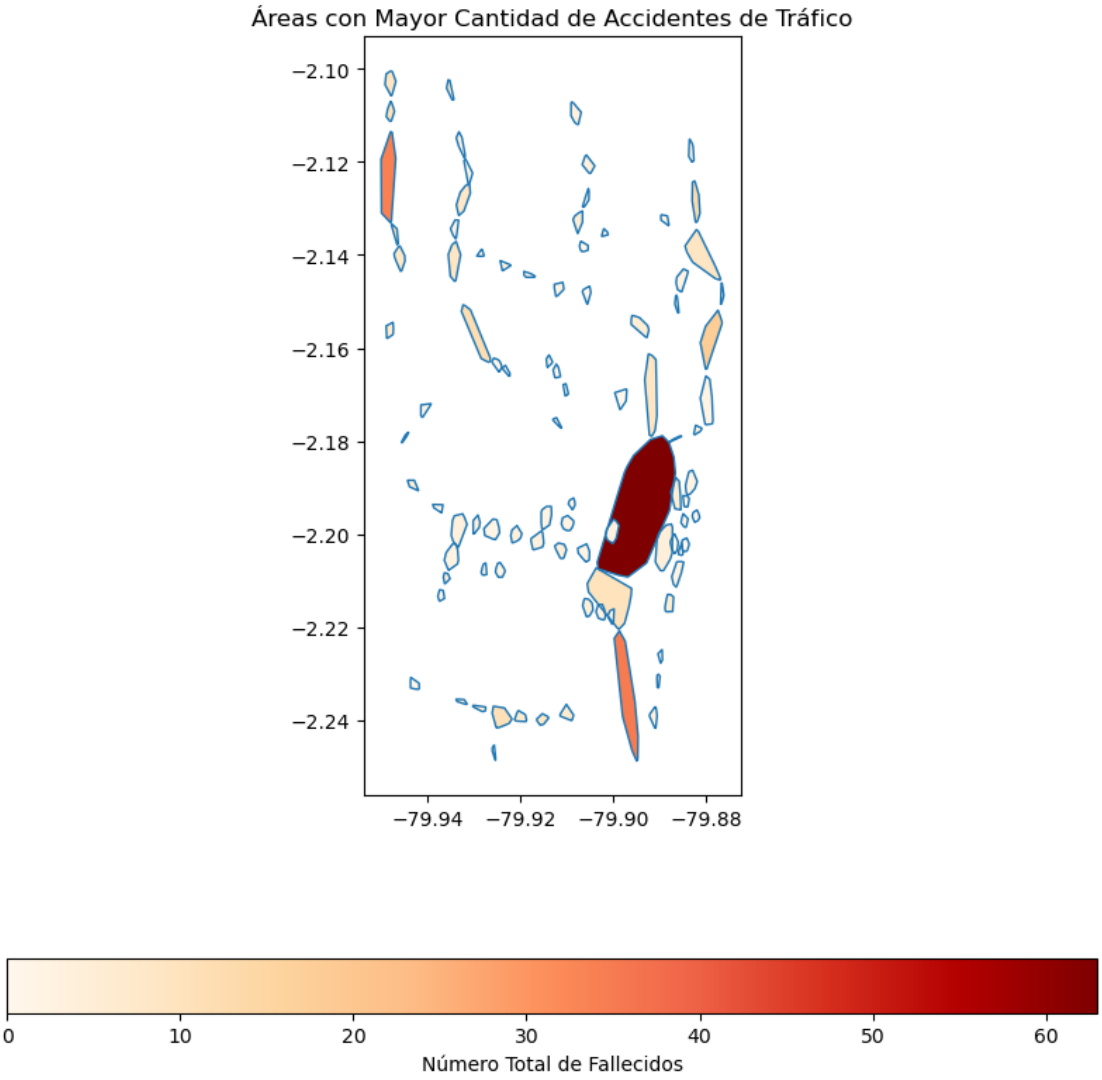
[91]:
```
coords_muestra = np.array(list(zip(muestra.geometry.x, muestra.geometry.y)))

w_muestra = weights.KNN(coords_muestra, k=10)
```

[92]:
```
g_muestra = G(muestra['FALLECIDOS'], w_muestra)

muestra['FALLECIDOS_scaled'] = (muestra['FALLECIDOS'] - muestra['FALLECIDOS'].
 ↪min()) / (muestra['FALLECIDOS'].max() - muestra['FALLECIDOS'].min())

g_muestra = G(muestra['FALLECIDOS_scaled'], w_muestra)

print(f"Índice de Getis-Ord Gi*: {g_muestra.G}, p-value: {g_muestra.p_sim}")
```

```
Índice de Getis-Ord Gi*: 0.007900280898876405, p-value: 0.004
```

[93]:
```
accidentes_por_poligono = gdf_clusters.groupby('cluster').agg({
'FALLECIDOS': 'sum',
'LESIONADOS': 'sum'
}).reset_index()

gdf_poligonos = gdf_poligonos.merge(accidentes_por_poligono, on='cluster',␣
 ↪how='left', suffixes=('', '_total'))
```

[94]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
gdf_poligonos.boundary.plot(ax=ax, linewidth=1)
gdf_poligonos.plot(column='FALLECIDOS', ax=ax, legend=True,
legend_kwds={'label': "Número Total de Fallecidos",
             'orientation': "horizontal"},
cmap='OrRd') # Usar un mapa de colores de rojo a naranja
plt.title('Áreas con Mayor Cantidad de Accidentes de Tráfico')
plt.show()
```




[95]:
```
gdf_poligonos_sorted = gdf_poligonos.sort_values(by='FALLECIDOS',␣
↪ascending=False)

print(gdf_poligonos_sorted[['cluster', 'FALLECIDOS', 'LESIONADOS']])
```

```
    cluster  FALLECIDOS  LESIONADOS
1         1          63        2896
8         8          35         600
5         5          34         451
46       46          18         239
50       50          13         121
..      ...         ...         ...
16       16           0          51
39       39           0          53
```

```
41        41            0           40
47        47            0           73
72        72            0           38

[92 rows x 3 columns]
```

[96]:
```
analisis_hotspots = df_parroquia_gye.groupby('cluster').agg({
'FALLECIDOS': 'sum',
'LESIONADOS': 'sum',
'HORA': lambda x: x.mode()[0]
}).reset_index()

analisis_hotspots
```

[96]:
```
    cluster  FALLECIDOS  LESIONADOS  HORA
0        -1         409       11937    15
1         0           2         169    19
2         1          63        2896     7
3         2           3         107     6
4         3           9         209     8
..      …          …          …   …
88       87           1          45     7
89       88           1          30    15
90       89           2          44     1
91       90           1          53    11
92       91           1          32    12

[93 rows x 4 columns]
```

[97]:
```
accidentes_por_dia = df_parroquia_gye.groupby(df_parroquia_gye['FECHA'].dt.
 ↪date).size().reset_index(name='Número de Accidentes')

accidentes_por_dia
```

[97]:
```
           FECHA  Número de Accidentes
0     2017-01-01                    15
1     2017-01-02                    15
2     2017-01-03                    12
3     2017-01-04                     7
4     2017-01-05                     5
…          …                    …
2667  2024-04-26                     8
2668  2024-04-27                    13
2669  2024-04-28                    14
2670  2024-04-29                    11
2671  2024-04-30                     5

[2672 rows x 2 columns]
```

[98]:
```
gdf_accidentes = gpd.GeoDataFrame(df_parroquia_gye, geometry=gpd.
 ↪points_from_xy(df_parroquia_gye['LONGITUD_X'],␣
 ↪df_parroquia_gye['LATITUD_Y']))

gdf_accidentes.set_crs(epsg=4326, inplace=True)

if gdf_poligonos.crs is None:
    gdf_poligonos.set_crs(epsg=4326, inplace=True)
    print("CRS de gdf_poligonos definido como EPSG:4326")

if gdf_accidentes.crs != gdf_poligonos.crs:
    gdf_accidentes = gdf_accidentes.to_crs(gdf_poligonos.crs)

gdf_joined = gpd.sjoin(gdf_accidentes, gdf_poligonos,␣
 ↪how='inner',predicate='within', lsuffix='_acc', rsuffix='_poly')

columns = [col for col in gdf_joined.columns if isinstance(col, str)]
columns_to_drop = [col for col in columns if 'cluster' in col and col.
 ↪endswith('_right')]

gdf_joined = gdf_joined.drop(columns=columns_to_drop, errors='ignore')
```

```
CRS de gdf_poligonos definido como EPSG:4326
```

[99]:
```
gdf_accidentes = gpd.GeoDataFrame(df_parroquia_gye, geometry=gpd.
 ↪points_from_xy(df_parroquia_gye['LONGITUD_X'],␣
 ↪df_parroquia_gye['LATITUD_Y']), crs='EPSG:4326')
gdf_poligonos = gpd.GeoDataFrame(gdf_poligonos, geometry=gdf_poligonos.
 ↪geometry, crs='EPSG:4326')
```

[100]:
```
print(gdf_joined.columns)

accidentes_por_poligono = gdf_joined.groupby('cluster__poly').size().
 ↪reset_index(name='num_accidentes')

gdf_poligonos = gdf_poligonos.merge(accidentes_por_poligono, left_on='cluster',␣
 ↪right_on='cluster__poly', how='left')
```

```
Index([                      'ANIO',              'LESIONADOS__acc',
                 'FALLECIDOS__acc',                    'LATITUD_Y',
                     'LONGITUD_X',                    'PROVINCIA',
                         'CANTON',                    'PARROQUIA',
               'ZONA_PLANIFICACION',                         'ZONA',
                          'FECHA',                        'HORA',
                        'FERIADO',                 'CODIGO_CAUSA',
                 'CAUSA_PROBABLE',           'TIPO_DE_SINIESTRO',
               'TIPO_DE_VEHICULO_1',                 'SERVICIO_1',
                       'AUTOMOVIL',                  'BICICLETA',
```

```
                                'BUS',                          'CAMION',
                           'CAMIONETA',                      'EMERGENCIAS',
                            'ESPECIAL',                        'FURGONETA',
                          'MOTOCICLETA',                   'NO_IDENTIFICADO',
                     'SCOOTER_ELECTRICO',                         'TRICIMOTO',
          'VEHICULO_DEPORTIVO_UTILITARIO',                'SUMA_DE_VEHICULOS',
                           'DIA_SEMANA',                     'cluster__acc',
                              'geometry',                      'index__poly',
                         'cluster__poly',                                  0,
                               'area_m2',                   'FALLECIDOS__poly',
                     'LESIONADOS__poly'],
      dtype='object')
```

[101]:
```
print(gdf_accidentes.crs)
print(gdf_poligonos.crs)

if gdf_accidentes.crs != gdf_poligonos.crs:
    gdf_accidentes = gdf_accidentes.to_crs(gdf_poligonos.crs)
```

```
EPSG:4326
EPSG:4326
```

[102]:
```
top_poligonos = gdf_poligonos.nlargest(10, 'num_accidentes')

print(top_poligonos[['num_accidentes', 'area_m2']])
```

```
    num_accidentes     area_m2
1             3262  302.160649
9              728   70.388144
8              687   64.685115
5              503   44.162261
7              397   34.064706
22             339   34.185848
32             306   23.450519
46             305   28.241863
27             245   27.517505
35             225   19.376418
```

[103]:
```
mapa_denso = folium.Map(location=[df_parroquia_gye['LATITUD_Y'].mean(),␣
 ↪df_parroquia_gye['LONGITUD_X'].mean()], zoom_start=12)

for _, row in top_poligonos.iterrows():
    folium.Polygon(
        locations=[(point[1], point[0]) for point in row.geometry.exterior.
 ↪coords],
        color='red',
        fill=True,
        fill_opacity=0.5,
```

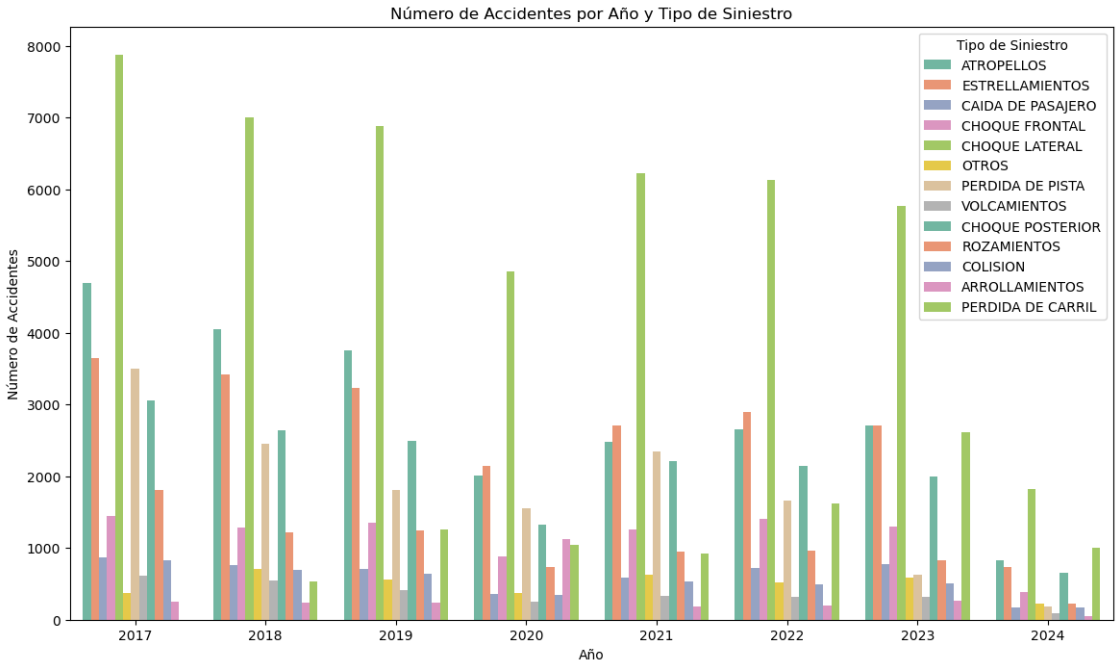
```
        popup=f"Accidentes: {row['num_accidentes']}, Área: {row['area_m2']:.2f}␣
↪m²"
    ).add_to(mapa_denso)

    mapa_denso
```

[104]:
```
gdf_poligonos['densidad_accidentes'] = gdf_poligonos['num_accidentes'] /␣
↪gdf_poligonos['area_m2']
top_densos = gdf_poligonos.nlargest(10, 'densidad_accidentes')

print(top_densos[['densidad_accidentes', 'num_accidentes', 'area_m2']])
```

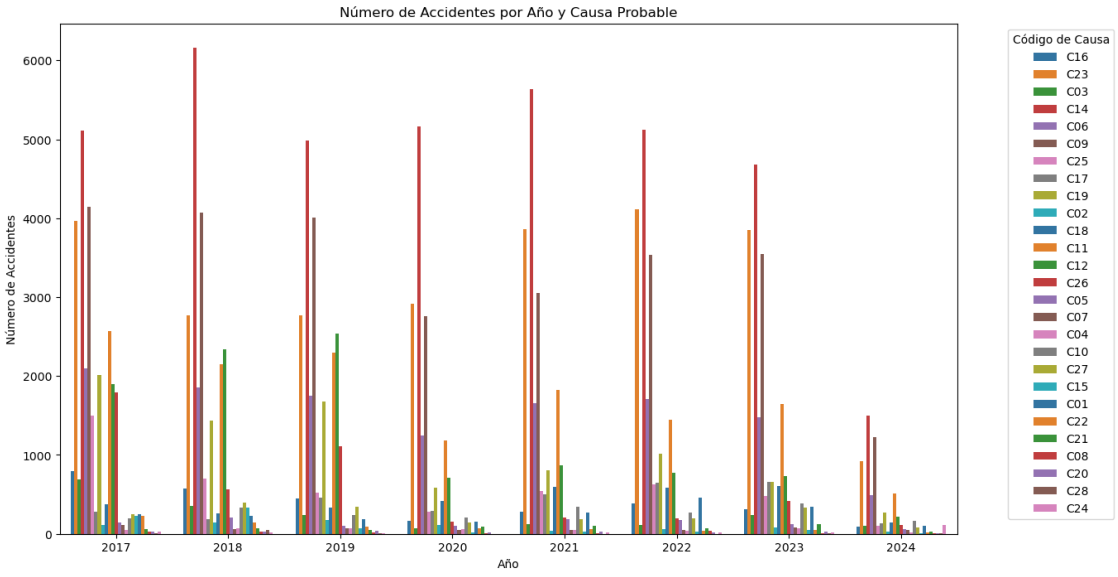
| | densidad_accidentes | num_accidentes | area_m2 |
|---|---|---|---|
| 24 | 93.610662 | 57 | 0.608905 |
| 83 | 57.882653 | 42 | 0.725606 |
| 74 | 36.850626 | 43 | 1.166873 |
| 70 | 35.156824 | 50 | 1.422199 |
| 64 | 35.006325 | 62 | 1.771109 |
| 14 | 34.448728 | 64 | 1.857833 |
| 59 | 31.988272 | 47 | 1.469289 |
| 60 | 30.167476 | 39 | 1.292783 |
| 55 | 29.313656 | 80 | 2.729103 |
| 34 | 28.640301 | 130 | 4.539058 |

## 1.1 ANALISIS TEMPORAL

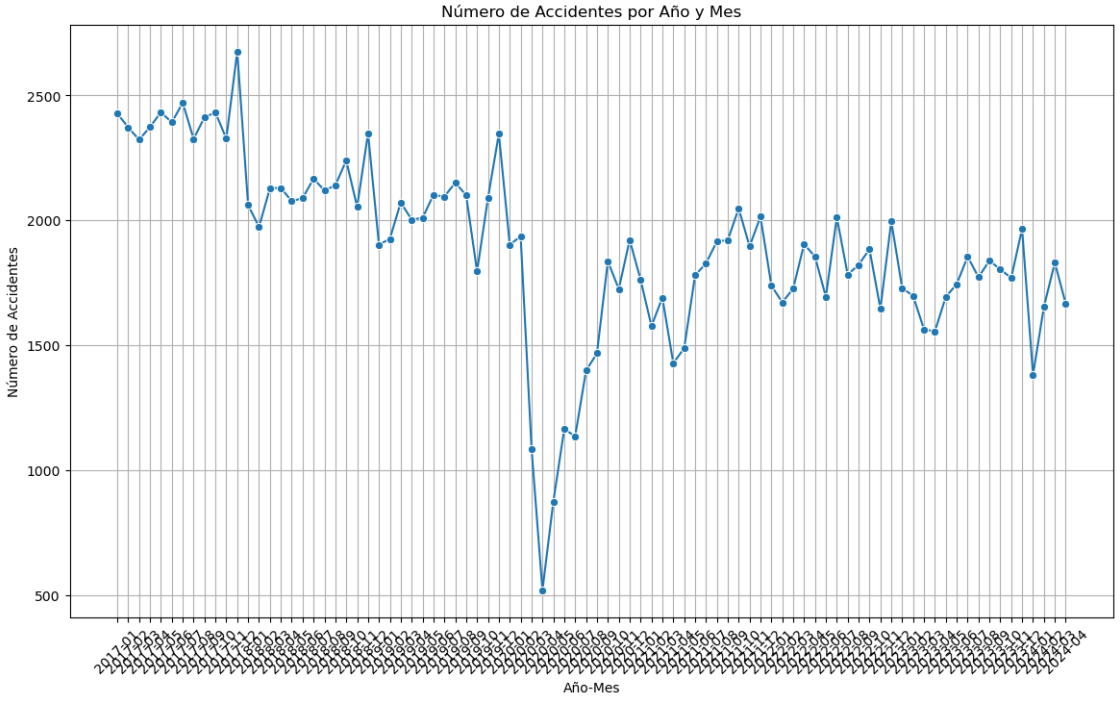
[105]:
```
plt.figure(figsize=(14, 8))
sns.countplot(data=df, x='ANIO', hue='TIPO_DE_SINIESTRO', palette='Set2')
plt.title('Número de Accidentes por Año y Tipo de Siniestro')
plt.xlabel('Año')
plt.ylabel('Número de Accidentes')
plt.legend(title='Tipo de Siniestro')
plt.show()
```




[106]:
```
plt.figure(figsize=(14, 8))
sns.countplot(data=df, x='ANIO', hue='CODIGO_CAUSA', palette='tab10')
plt.title('Número de Accidentes por Año y Causa Probable')
plt.xlabel('Año')
plt.ylabel('Número de Accidentes')
plt.legend(title='Código de Causa', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

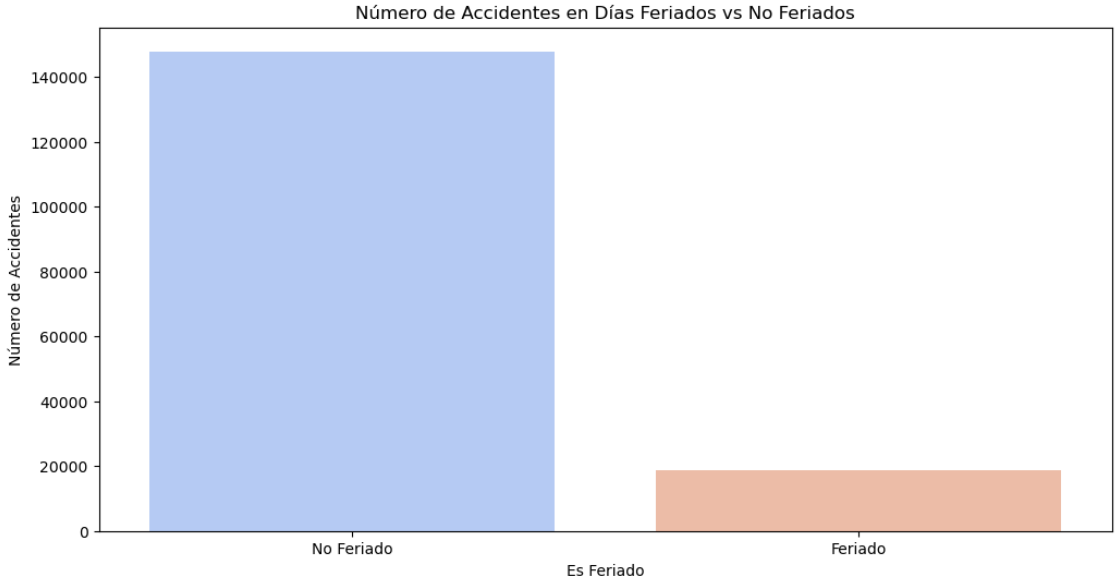


[107]:
```
df['FECHA'] = pd.to_datetime(df['FECHA'], format='%d/%m/%Y', dayfirst=True)

df['AÑO'] = df['FECHA'].dt.year
df['MES'] = df['FECHA'].dt.month
df['DIA'] = df['FECHA'].dt.day

accidentes_por_anio_mes = df.groupby(['AÑO', 'MES']).size().
 ↪reset_index(name='Número de Accidentes')

accidentes_por_anio_mes['AÑO_MES'] = accidentes_por_anio_mes['AÑO'].astype(str)␣
 ↪+ '-' + accidentes_por_anio_mes['MES'].astype(str).str.zfill(2)

plt.figure(figsize=(14, 8))
sns.lineplot(data=accidentes_por_anio_mes, x='AÑO_MES', y='Número de␣
 ↪Accidentes', marker='o')
plt.title('Número de Accidentes por Año y Mes')
plt.xlabel('Año-Mes')
plt.ylabel('Número de Accidentes')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



[108]:
```
df['FECHA'] = pd.to_datetime(df['FECHA'], format='%d/%m/%Y')

df['ES_FERIADO'] = df['FERIADO'].str.lower().apply(lambda x: x in ['sí', 'si',␣
↪'yes', '1'])

print(df[['FECHA', 'FERIADO', 'ES_FERIADO']].head())

plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='ES_FERIADO', palette='coolwarm')
plt.title('Número de Accidentes en Días Feriados vs No Feriados')
plt.xlabel('Es Feriado')
plt.ylabel('Número de Accidentes')
plt.xticks([0, 1], ['No Feriado', 'Feriado'])
plt.show()
```

```
       FECHA FERIADO  ES_FERIADO
0 2017-01-01      SI        True
1 2017-01-01      SI        True
2 2017-01-01      SI        True
3 2017-01-01      SI        True
4 2017-01-01      SI        True
```



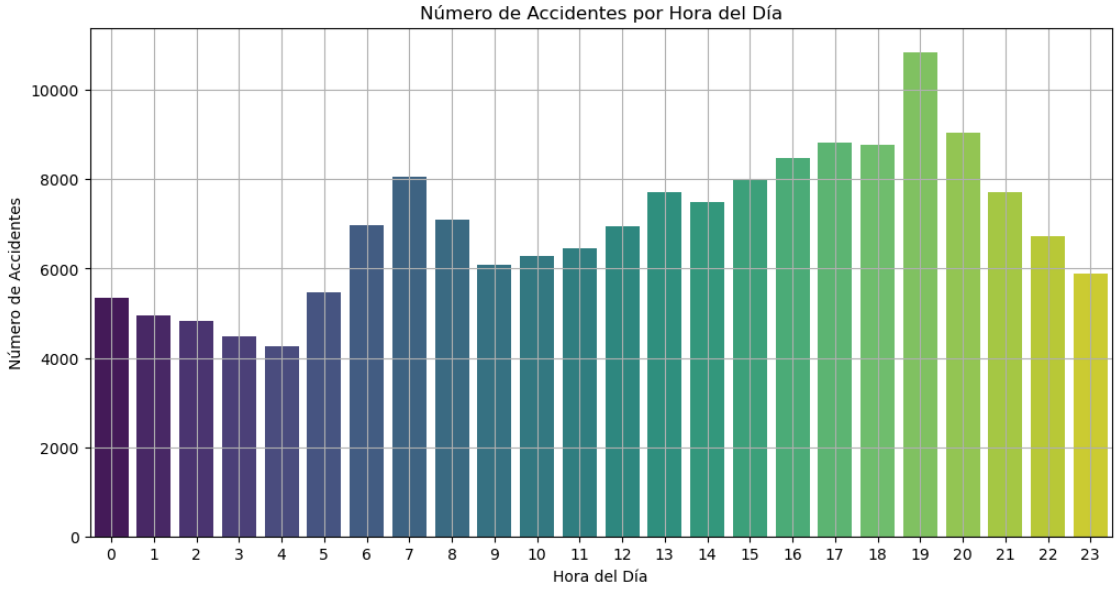

[109]:
```
print(df['HORA'].unique())
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

[110]:
```
print(df['HORA'].head())
print(df['HORA'].dtype)
```

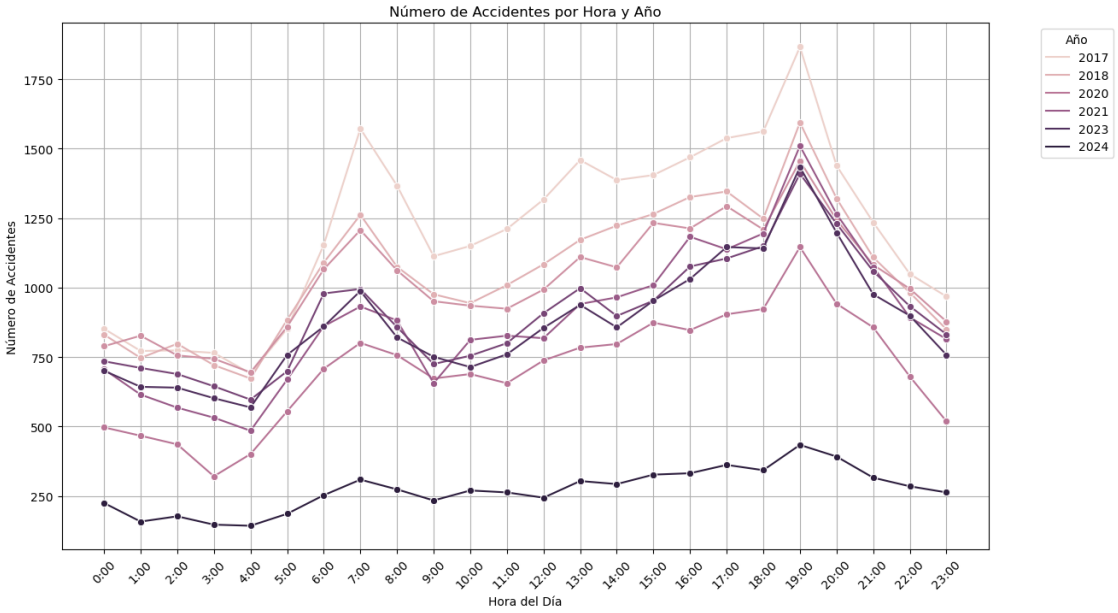
```
0    0
1    0
2    0
3    0
4    0
Name: HORA, dtype: int32
int32
```

[111]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='HORA', palette='viridis')
plt.title('Número de Accidentes por Hora del Día')
plt.xlabel('Hora del Día')
plt.ylabel('Número de Accidentes')
plt.grid(True)
plt.xticks(range(24))
plt.show()
```

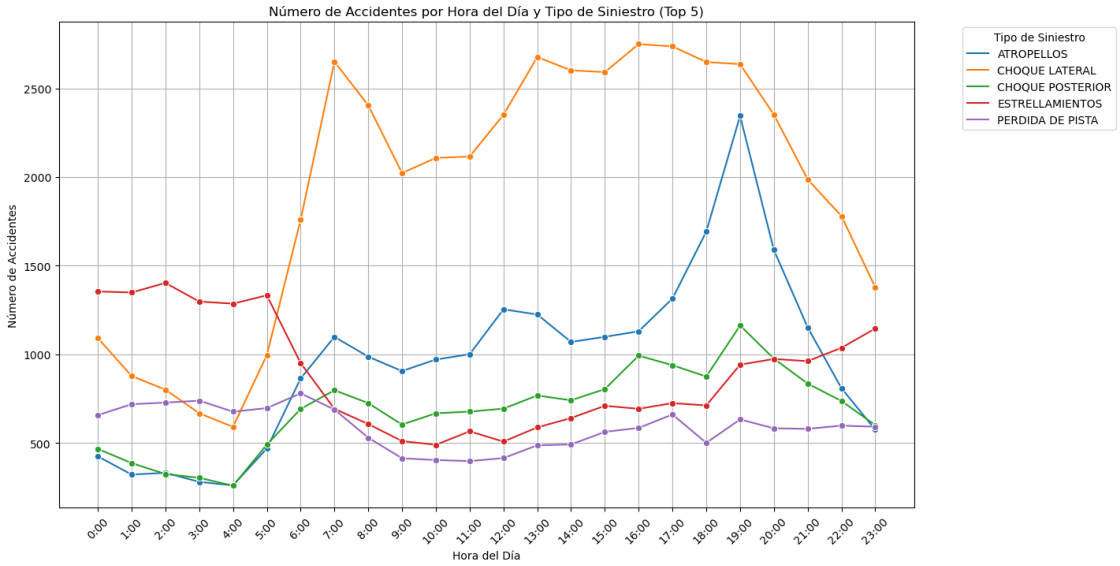



[112]:
```
accidentes_por_hora_anio = df.groupby(['HORA', 'ANIO']).size().
 ↪reset_index(name='Número de Accidentes')

plt.figure(figsize=(14, 8))
sns.lineplot(data=accidentes_por_hora_anio, x='HORA', y='Número de Accidentes',␣
 ↪hue='ANIO', marker='o')
plt.title('Número de Accidentes por Hora y Año')
plt.xlabel('Hora del Día')
plt.ylabel('Número de Accidentes')
plt.xticks(range(24), [f'{h}:00' for h in range(24)], rotation=45)
```

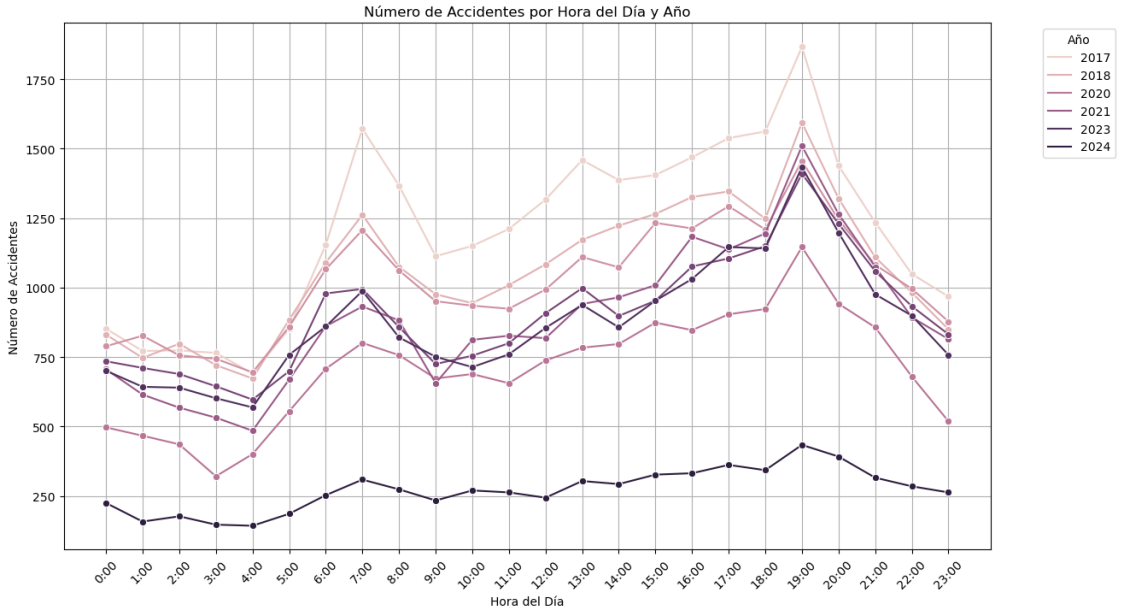
```
plt.legend(title='Año', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
```




[113]:
```
top_5_siniestros = df['TIPO_DE_SINIESTRO'].value_counts().head(5).index

df_top_5 = df[df['TIPO_DE_SINIESTRO'].isin(top_5_siniestros)]

accidentes_por_hora_tipo = df_top_5.groupby(['HORA', 'TIPO_DE_SINIESTRO']).
 ↪size().reset_index(name='Número de Accidentes')

plt.figure(figsize=(14, 8))
sns.lineplot(data=accidentes_por_hora_tipo, x='HORA', y='Número de Accidentes',␣
 ↪hue='TIPO_DE_SINIESTRO', marker='o')
plt.title('Número de Accidentes por Hora del Día y Tipo de Siniestro (Top 5)')
plt.xlabel('Hora del Día')
plt.ylabel('Número de Accidentes')
plt.xticks(range(24), [f'{h}:00' for h in range(24)], rotation=45)
plt.legend(title='Tipo de Siniestro', bbox_to_anchor=(1.05, 1), loc='upper␣
 ↪left')
plt.grid(True)
plt.show()
```




[114]:
```
accidentes_por_hora_anio = df.groupby(['HORA', 'ANIO']).size().
  ↪reset_index(name='Número de Accidentes')

plt.figure(figsize=(14, 8))
sns.lineplot(data=accidentes_por_hora_anio, x='HORA', y='Número de Accidentes',␣
  ↪hue='ANIO', marker='o')
plt.title('Número de Accidentes por Hora del Día y Año')
plt.xlabel('Hora del Día')
plt.ylabel('Número de Accidentes')
plt.xticks(range(24), [f'{h}:00' for h in range(24)], rotation=45)
plt.legend(title='Año', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
```



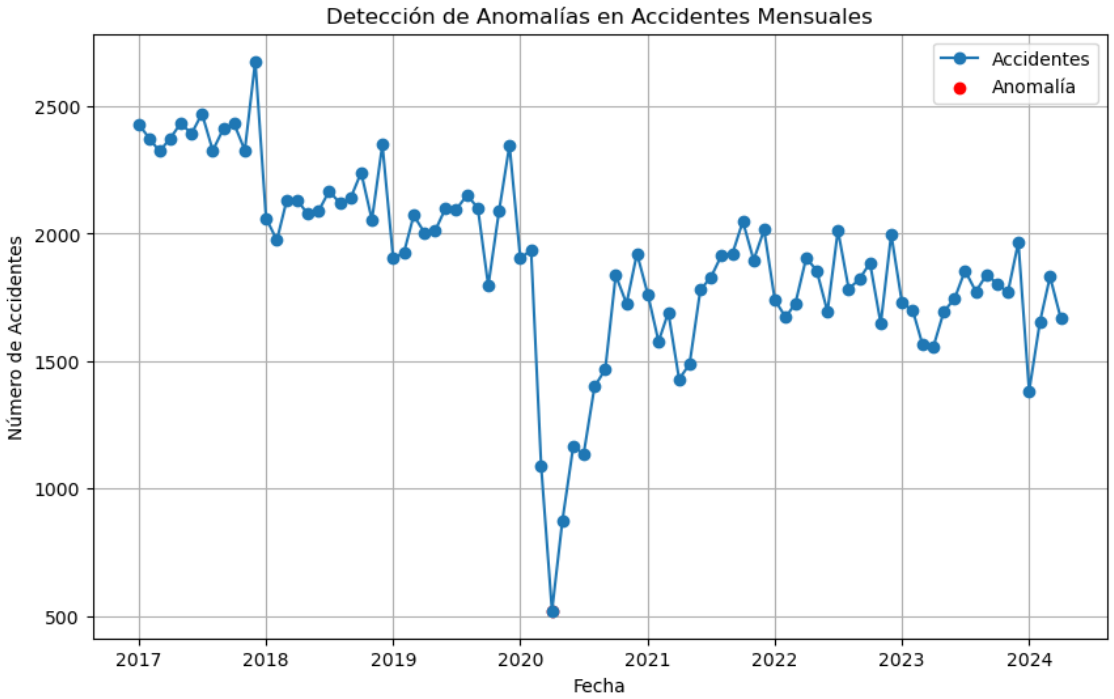

[115]:
```
df_mensual = df.groupby(['AÑO', 'MES']).size().
 ↪reset_index(name='num_accidentes')

df_mensual = df_mensual.rename(columns={'AÑO': 'year', 'MES': 'month'})

df_mensual['FECHA'] = pd.to_datetime(df_mensual.assign(day=1)[['year', 'month',␣
 ↪'day']])
```

[116]:
```
mean_accidentes = df_mensual['num_accidentes'].mean()
std_accidentes = df_mensual['num_accidentes'].std()

upper_bound = mean_accidentes + 3 * std_accidentes
lower_bound = mean_accidentes - 3 * std_accidentes
```

[117]:
```
df_mensual['anomaly'] = (df_mensual['num_accidentes'] > upper_bound) |␣
 ↪(df_mensual['num_accidentes'] < lower_bound)
```

[118]:
```
plt.figure(figsize=(10, 6))
plt.plot(df_mensual['FECHA'], df_mensual['num_accidentes'], marker='o',␣
 ↪label='Accidentes')
plt.scatter(df_mensual[df_mensual['anomaly']]['FECHA'],␣
 ↪df_mensual[df_mensual['anomaly']]['num_accidentes'], color='red',␣
 ↪label='Anomalía', marker='o')
plt.title('Detección de Anomalías en Accidentes Mensuales')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
```

```
plt.legend()
plt.grid(True)
plt.show()
```




[119]:
```
upper_bound = mean_accidentes + 2 * std_accidentes
lower_bound = mean_accidentes - 2 * std_accidentes

df_mensual['anomaly'] = (df_mensual['num_accidentes'] > upper_bound) |␣
↪(df_mensual['num_accidentes'] < lower_bound)
```

[120]:
```
iso_forest = IsolationForest(contamination=0.05, random_state=42)

df_mensual['anomaly'] = iso_forest.fit_predict(df_mensual[['num_accidentes']])

df_mensual['anomaly'] = df_mensual['anomaly'] == -1
```

[121]:
```
print(f"Media de accidentes: {mean_accidentes}")
print(f"Desviación estándar de accidentes: {std_accidentes}")
print(f"Límite superior: {upper_bound}")
print(f"Límite inferior: {lower_bound}")
print(df_mensual['num_accidentes'].describe())
```

```
Media de accidentes: 1894.1136363636363
Desviación estándar de accidentes: 361.44443403152866
Límite superior: 2617.0025044266936
```

```
Límite inferior: 1171.224768300579
count      88.000000
mean     1894.113636
std       361.444434
min       520.000000
25%      1726.500000
50%      1904.000000
75%      2101.250000
max      2676.000000
Name: num_accidentes, dtype: float64
```

[122]:
```
df_mensual['z_score'] = (df_mensual['num_accidentes'] - mean_accidentes) /␣
↪std_accidentes

df_mensual['anomaly'] = df_mensual['z_score'].abs() > 2

print(df_mensual[df_mensual['anomaly']])
```

```
    year  month  num_accidentes      FECHA  anomaly   z_score
11  2017     12            2676 2017-12-01     True  2.163227
38  2020      3            1086 2020-03-01     True -2.235789
39  2020      4             520 2020-04-01     True -3.801729
40  2020      5             875 2020-05-01     True -2.819558
41  2020      6            1165 2020-06-01     True -2.017222
42  2020      7            1136 2020-07-01     True -2.097456
```

[123]:
```
print("Total de anomalías detectadas:", df_mensual['anomaly'].sum())
```

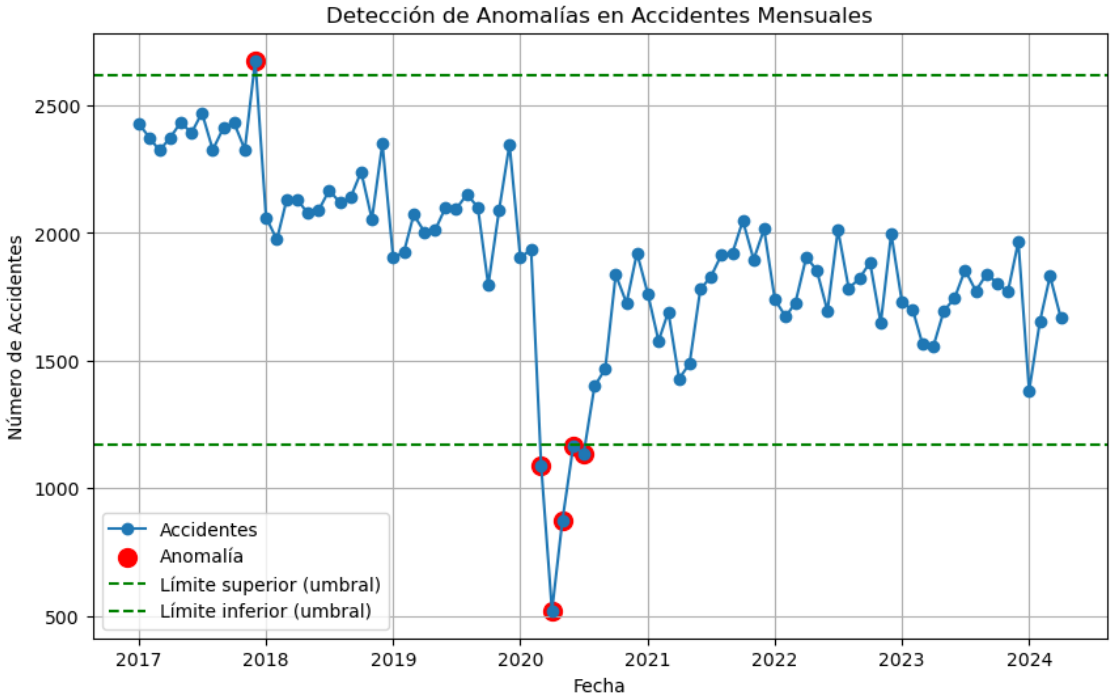
```
Total de anomalías detectadas: 6
```

[124]:
```
print(df_mensual[df_mensual['anomaly']][['FECHA', 'num_accidentes']])
```

```
        FECHA  num_accidentes
11 2017-12-01            2676
38 2020-03-01            1086
39 2020-04-01             520
40 2020-05-01             875
41 2020-06-01            1165
42 2020-07-01            1136
```

[125]:
```
plt.figure(figsize=(10, 6))

plt.plot(df_mensual['FECHA'], df_mensual['num_accidentes'], marker='o',␣
↪label='Accidentes')

anomalies = df_mensual[df_mensual['anomaly']]  # Filtrar solo las anomalías
plt.scatter(anomalies['FECHA'], anomalies['num_accidentes'], color='red',␣
↪label='Anomalía', s=100, marker='o')
```

```
plt.axhline(upper_bound, color='green', linestyle='--', label='Límite superior␣
 ↪(umbral)')
plt.axhline(lower_bound, color='green', linestyle='--', label='Límite inferior␣
 ↪(umbral)')

plt.title('Detección de Anomalías en Accidentes Mensuales')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.legend()
plt.grid(True)
plt.show()
```




Informe de Anomalías en Accidentes Mensuales

Anomalías Detectadas:

Diciembre 2017: 2676 accidentes (alto) Marzo 2020: 1086 accidentes (bajo) Abril 2020: 520 accidentes (bajo) Mayo 2020: 875 accidentes (bajo) Junio 2020: 1165 accidentes (bajo) Julio 2020: 1136 accidentes (bajo) Posibles Explicaciones:

Diciembre 2017: Incremento debido a eventos festivos y vacaciones. Marzo a Julio 2020: Reducción de accidentes relacionada con el confinamiento y las restricciones de movilidad durante la pandemia de COVID-19. Impacto en el Análisis:

Las anomalías pueden indicar cambios significativos en el comportamiento del tráfico y deben ser consideradas al modelar futuros accidentes.

[126]:
```
df_mensual['FECHA'] = pd.to_datetime(df_mensual['FECHA'])

df_mensual['mes'] = df_mensual['FECHA'].dt.month

media_por_mes = df_mensual[~df_mensual['anomaly']].
 ↪groupby('mes')['num_accidentes'].mean()

print(media_por_mes)
```

```
mes
1     1863.500000
2     1851.125000
3     1905.285714
4     1866.285714
5     1926.166667
6     1967.000000
7     2071.500000
8     1924.000000
9     1957.714286
10    2006.000000
11    1929.714286
12    2099.333333
Name: num_accidentes, dtype: float64
```

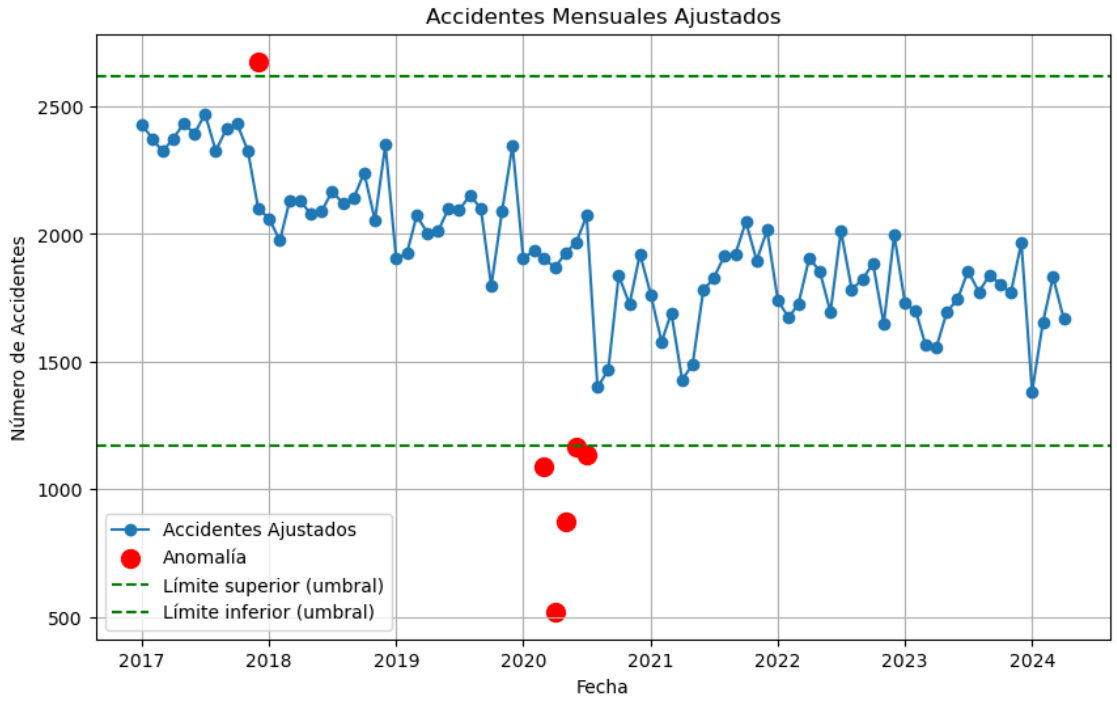
[127]:
```
for index, row in df_mensual.iterrows():
    if row['anomaly']:  # Si es una anomalía
        mes = row['mes']
        df_mensual.at[index, 'num_accidentes'] = media_por_mes[mes]

print(df_mensual[['FECHA', 'num_accidentes', 'anomaly']].head(20))
```

```
        FECHA  num_accidentes  anomaly
0  2017-01-01     2428.000000    False
1  2017-02-01     2372.000000    False
2  2017-03-01     2323.000000    False
3  2017-04-01     2374.000000    False
4  2017-05-01     2433.000000    False
5  2017-06-01     2392.000000    False
6  2017-07-01     2471.000000    False
7  2017-08-01     2326.000000    False
8  2017-09-01     2413.000000    False
9  2017-10-01     2432.000000    False
10 2017-11-01     2327.000000    False
11 2017-12-01     2099.333333     True
12 2018-01-01     2061.000000    False
13 2018-02-01     1974.000000    False
14 2018-03-01     2129.000000    False
15 2018-04-01     2130.000000    False
```

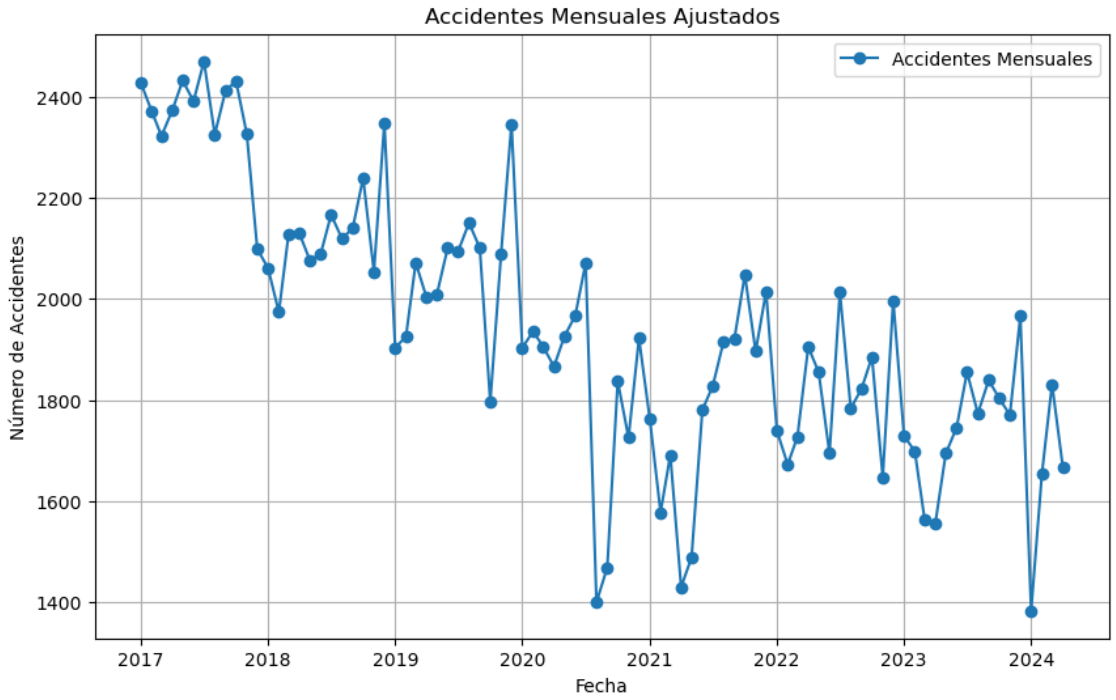
```
16 2018-05-01     2077.000000    False
17 2018-06-01     2089.000000    False
18 2018-07-01     2167.000000    False
19 2018-08-01     2120.000000    False
```

[128]:
```
df_sin_anomalias = df_mensual
```

[129]:
```
plt.figure(figsize=(10, 6))

plt.plot(df_mensual['FECHA'], df_mensual['num_accidentes'], marker='o',␣
 ↪label='Accidentes Ajustados')

plt.scatter(anomalies['FECHA'], anomalies['num_accidentes'], color='red',␣
 ↪label='Anomalía', s=100, marker='o')

plt.axhline(upper_bound, color='green', linestyle='--', label='Límite superior␣
 ↪(umbral)')
plt.axhline(lower_bound, color='green', linestyle='--', label='Límite inferior␣
 ↪(umbral)')

plt.title('Accidentes Mensuales Ajustados')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.legend()
plt.grid(True)
plt.show()
```



[130]:
```
plt.figure(figsize=(10, 6))

plt.plot(df_mensual['FECHA'], df_mensual['num_accidentes'], marker='o',␣
↪label='Accidentes Mensuales')

plt.title('Accidentes Mensuales Ajustados')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.legend()
plt.grid(True)
plt.show()
```




[131]:
```
df_prophet = df_sin_anomalias.rename(columns={'FECHA': 'ds', 'num_accidentes':␣
↪'y'})

print(df_prophet.head())
```

```
   year  month       y         ds  anomaly   z_score  mes
0  2017      1  2428.0 2017-01-01    False  1.477091    1
1  2017      2  2372.0 2017-02-01    False  1.322157    2
2  2017      3  2323.0 2017-03-01    False  1.186590    3
3  2017      4  2374.0 2017-04-01    False  1.327691    4
```

```
4  2017      5  2433.0 2017-05-01    False  1.490925    5
```

[132]:
```
df_train = df_sin_anomalias[:-12]  # Datos para entrenamiento
df_test = df_sin_anomalias[-12:]    # Datos para prueba

df_train = df_train.rename(columns={'FECHA': 'ds', 'num_accidentes': 'y'})
```

[133]:
```
df_prophet = df_mensual.rename(columns={'FECHA': 'ds', 'num_accidentes': 'y'})

model = Prophet()
model.fit(df_prophet)
```

```
21:39:00 - cmdstanpy - INFO - Chain [1] start processing
21:39:00 - cmdstanpy - INFO - Chain [1] done processing
```

[133]:
```
<prophet.forecaster.Prophet at 0x2bb8cc56810>
```

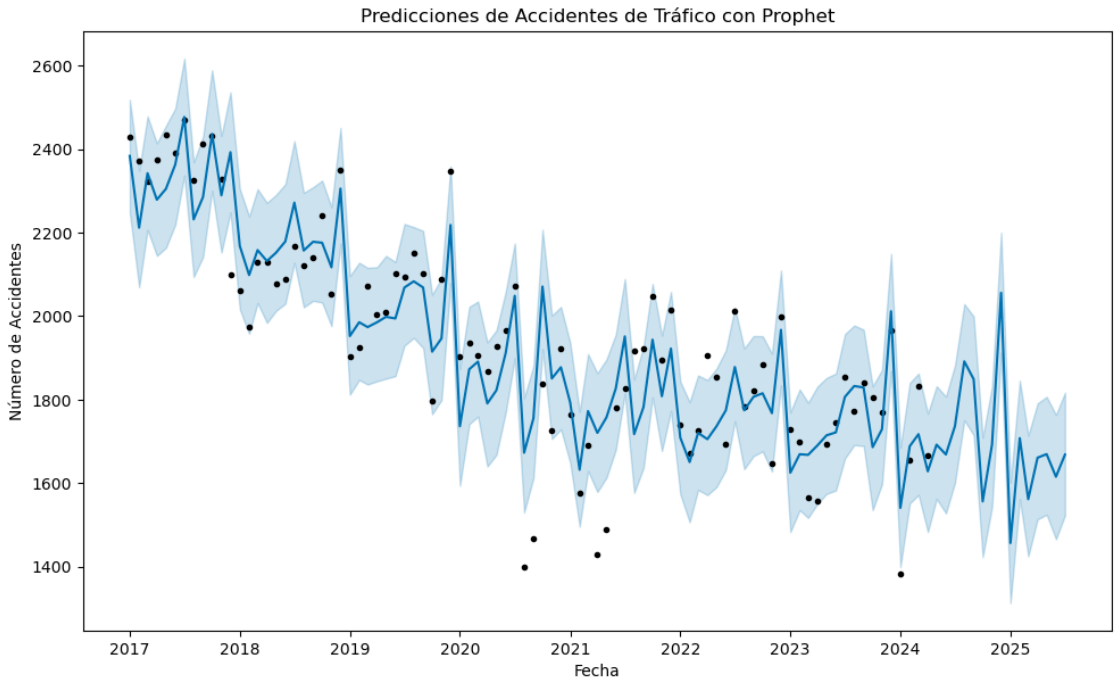
[134]:
```
future = model.make_future_dataframe(periods=12, freq='M')  # 'M' indica que es␣
↪mensual

forecast = model.predict(future)

print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail(12))  # Mostrar␣
↪solo las predicciones para los próximos 12 meses
```

```
           ds         yhat   yhat_lower   yhat_upper
88 2024-04-30  1692.085053  1547.577745  1827.393053
89 2024-05-31  1668.503402  1531.896674  1811.888016
90 2024-06-30  1736.511685  1587.312899  1875.713059
91 2024-07-31  1891.529343  1746.622646  2035.239072
92 2024-08-31  1848.891908  1707.069252  1990.975326
93 2024-09-30  1556.252283  1410.926068  1699.789436
94 2024-10-31  1692.315240  1547.221876  1836.145141
95 2024-11-30  2055.673398  1921.643146  2200.660091
96 2024-12-31  1456.457154  1306.064044  1600.117455
97 2025-01-31  1707.655415  1564.754181  1849.732505
98 2025-02-28  1561.358682  1422.623398  1702.958358
99 2025-03-31  1660.800655  1514.517094  1793.140534
```

[135]:
```
df = df_sin_anomalias[['FECHA', 'num_accidentes']].rename(columns={'FECHA':␣
↪'ds', 'num_accidentes': 'y'})

model = Prophet()
model.fit(df)

future = model.make_future_dataframe(periods=15, freq='M')  # 15 meses hacia el␣
↪futuro
```

```
forecast = model.predict(future)

fig = model.plot(forecast)
plt.title('Predicciones de Accidentes de Tráfico con Prophet')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.grid()
plt.show()
```

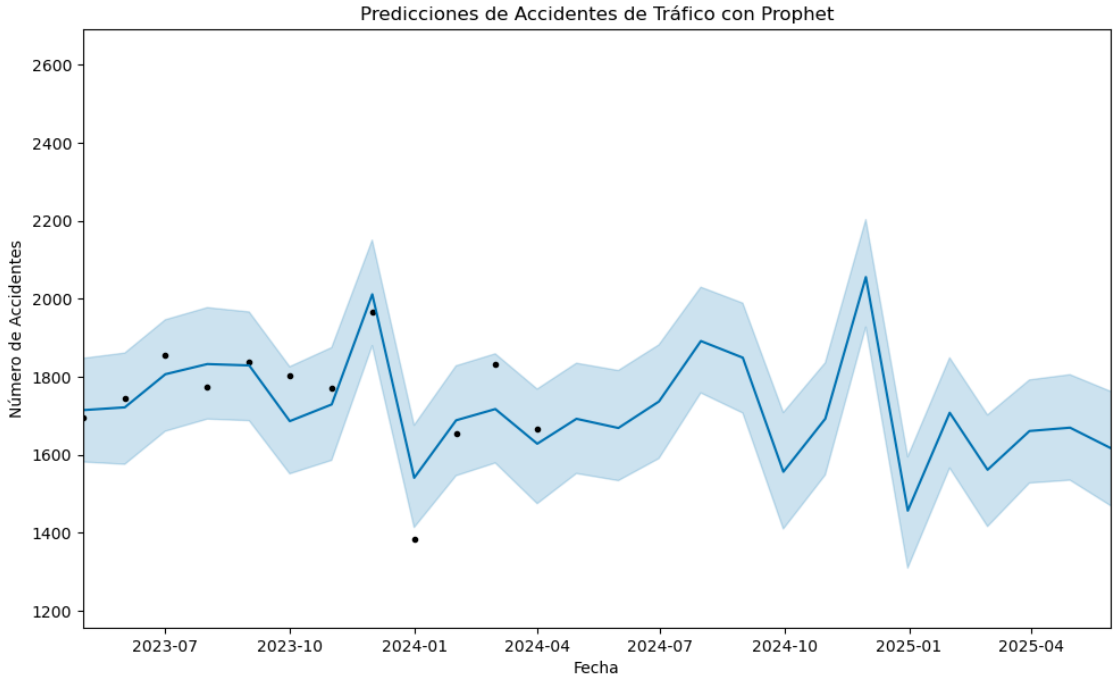
```
21:39:00 - cmdstanpy - INFO - Chain [1] start processing
21:39:01 - cmdstanpy - INFO - Chain [1] done processing
```




[136]:
```
df = df_sin_anomalias[['FECHA', 'num_accidentes']].rename(columns={'FECHA':␣
↪'ds', 'num_accidentes': 'y'})

model = Prophet()
model.fit(df)

future = model.make_future_dataframe(periods=24, freq='M')  # 24 meses hacia el␣
↪futuro

forecast = model.predict(future)

fig = model.plot(forecast)
```

```
plt.xlim(pd.Timestamp('2023-05-01'), pd.Timestamp('2025-05-30'))

plt.title('Predicciones de Accidentes de Tráfico con Prophet')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.grid()
plt.show()
```

```
21:39:02 - cmdstanpy - INFO - Chain [1] start processing
21:39:03 - cmdstanpy - INFO - Chain [1] done processing
```



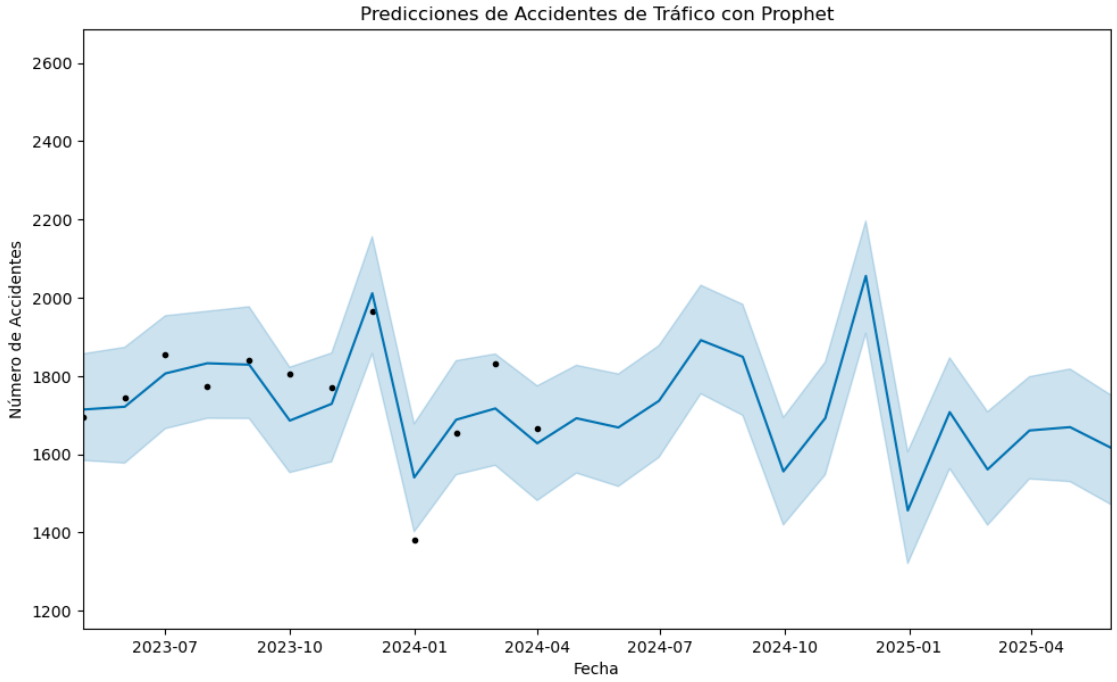

[137]:
```
df = df_sin_anomalias[['FECHA', 'num_accidentes']].rename(columns={'FECHA':␣
↪'ds', 'num_accidentes': 'y'})

model = Prophet()
model.fit(df)

future = model.make_future_dataframe(periods=24, freq='M')  # 24 meses hacia el␣
↪futuro

forecast = model.predict(future)

predicciones_matriz = forecast[['ds', 'yhat']].rename(columns={'ds': 'Fecha',␣
↪'yhat': 'Predicción'})
```

```
predicciones_matriz = predicciones_matriz[predicciones_matriz['Fecha'] >=␣
 ↪'2023-05-01']

predicciones_matriz = predicciones_matriz.reset_index(drop=True)

print(predicciones_matriz)

fig = model.plot(forecast)

plt.xlim(pd.Timestamp('2023-05-01'), pd.Timestamp('2025-05-30'))

plt.title('Predicciones de Accidentes de Tráfico con Prophet')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.grid()
plt.show()
```

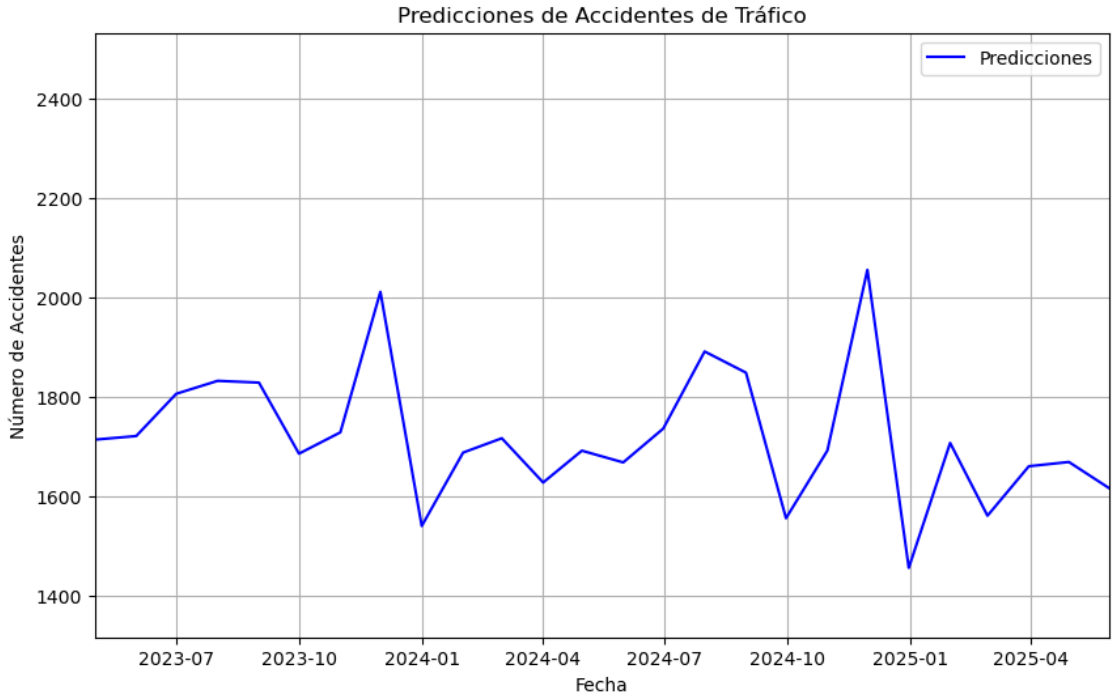
```
21:39:03 - cmdstanpy - INFO - Chain [1] start processing
21:39:04 - cmdstanpy - INFO - Chain [1] done processing
```

| | Fecha | Predicción |
|---|---|---|
| 0 | 2023-05-01 | 1714.275779 |
| 1 | 2023-06-01 | 1721.485701 |
| 2 | 2023-07-01 | 1806.446531 |
| 3 | 2023-08-01 | 1832.491854 |
| 4 | 2023-09-01 | 1828.960828 |
| 5 | 2023-10-01 | 1686.078686 |
| 6 | 2023-11-01 | 1728.914834 |
| 7 | 2023-12-01 | 2011.433247 |
| 8 | 2024-01-01 | 1540.709528 |
| 9 | 2024-02-01 | 1688.261879 |
| 10 | 2024-03-01 | 1717.080333 |
| 11 | 2024-04-01 | 1628.123747 |
| 12 | 2024-04-30 | 1692.085053 |
| 13 | 2024-05-31 | 1668.503402 |
| 14 | 2024-06-30 | 1736.511685 |
| 15 | 2024-07-31 | 1891.529343 |
| 16 | 2024-08-31 | 1848.891908 |
| 17 | 2024-09-30 | 1556.252283 |
| 18 | 2024-10-31 | 1692.315240 |
| 19 | 2024-11-30 | 2055.673398 |
| 20 | 2024-12-31 | 1456.457154 |
| 21 | 2025-01-31 | 1707.655415 |
| 22 | 2025-02-28 | 1561.358682 |
| 23 | 2025-03-31 | 1660.800655 |
| 24 | 2025-04-30 | 1669.357332 |
| 25 | 2025-05-31 | 1615.266672 |
| 26 | 2025-06-30 | 1668.260851 |

```
27 2025-07-31  1951.400720
28 2025-08-31  1866.392309
29 2025-09-30  1425.791754
30 2025-10-31  1657.711669
31 2025-11-30  2099.581059
32 2025-12-31  1372.118206
33 2026-01-31  1727.304662
34 2026-02-28  1507.797455
35 2026-03-31  1646.198203
```




[138]:
```
fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(forecast['ds'], forecast['yhat'], label='Predicciones', color='blue')

ax.set_xlim(pd.Timestamp('2023-05-01'), pd.Timestamp('2025-05-30'))

plt.title('Predicciones de Accidentes de Tráfico')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.grid()
plt.legend()

plt.show()
```




[139]:
```
df_train = df_sin_anomalias[:-12]
df_test = df_sin_anomalias[-12:]

scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(df_sin_anomalias[['num_accidentes']])

def crear_secuencias(data, time_steps=1):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:(i + time_steps), 0])
        y.append(data[i + time_steps, 0])
    return np.array(X), np.array(y)

time_steps = 12
X, y = crear_secuencias(data_scaled, time_steps)

X_train, y_train = X[:-12], y[:-12]
X_test, y_test = X[-12:], y[-12:]

X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

model = keras.Sequential([
```

```
    layers.LSTM(100, activation='relu', return_sequences=True,␣
 ↪input_shape=(X_train.shape[1], 1)),
    layers.LSTM(100, activation='relu'),
    layers.Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X_train, y_train, epochs=300, batch_size=16, verbose=1)

predicciones = model.predict(X_test)
predicciones = scaler.inverse_transform(predicciones)

def generar_predicciones_futuras(model, scaler, last_data, num_months):
    predicciones_futuras = []
    input_data = last_data[-time_steps:].reshape(-1)  # Usar los últimos␣
 ↪`time_steps` para iniciar

    for _ in range(num_months):
        input_data_reshaped = input_data.reshape((1, time_steps, 1))
        prediccion = model.predict(input_data_reshaped)
        predicciones_futuras.append(prediccion[0][0])
        input_data = np.append(input_data[1:], prediccion[0][0])  # Desplazar␣
 ↪el input

    return scaler.inverse_transform(np.array(predicciones_futuras).reshape(-1,␣
 ↪1))

predicciones_futuras = generar_predicciones_futuras(model, scaler, data_scaled,␣
 ↪15)

ultima_fecha = df_sin_anomalias['FECHA'].iloc[-1]

fechas_futuras = pd.date_range(start=ultima_fecha - pd.DateOffset(months=1),␣
 ↪periods=15, freq='M')

plt.figure(figsize=(14, 7))
plt.plot(df_sin_anomalias['FECHA'], df_sin_anomalias['num_accidentes'],␣
 ↪label='Datos Reales', color='blue')
plt.plot(df_test['FECHA'], predicciones, label='Predicciones LSTM',␣
 ↪color='red', marker='o')

plt.plot(fechas_futuras, predicciones_futuras, label='Predicciones Futuras',␣
 ↪color='green', marker='x')

plt.title('Predicciones de Accidentes de Tráfico con LSTM')
```

```
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.legend()
plt.grid()
plt.xticks(rotation=45)

plt.show()
```

```
Epoch 1/300
4/4             4s 24ms/step - loss:
0.2219
Epoch 2/300
4/4             0s 16ms/step - loss:
0.1547
Epoch 3/300
4/4             0s 20ms/step - loss:
0.0495
Epoch 4/300
4/4             0s 14ms/step - loss:
0.0461
Epoch 5/300
4/4             0s 12ms/step - loss:
0.0316
Epoch 6/300
4/4             0s 12ms/step - loss:
0.0329
Epoch 7/300
4/4             0s 13ms/step - loss:
0.0335
Epoch 8/300
4/4             0s 13ms/step - loss:
0.0273
Epoch 9/300
4/4             0s 14ms/step - loss:
0.0253
Epoch 10/300
4/4             0s 20ms/step - loss:
0.0297
Epoch 11/300
4/4             0s 19ms/step - loss:
0.0307
Epoch 12/300
4/4             0s 19ms/step - loss:
0.0315
Epoch 13/300
4/4             0s 16ms/step - loss:
0.0332
Epoch 14/300
```

```
4/4             0s 12ms/step - loss:
0.0262
Epoch 15/300
4/4             0s 14ms/step - loss:
0.0250
Epoch 16/300
4/4             0s 20ms/step - loss:
0.0286
Epoch 17/300
4/4             0s 19ms/step - loss:
0.0190
Epoch 18/300
4/4             0s 21ms/step - loss:
0.0253
Epoch 19/300
4/4             0s 21ms/step - loss:
0.0282
Epoch 20/300
4/4             0s 18ms/step - loss:
0.0269
Epoch 21/300
4/4             0s 20ms/step - loss:
0.0281
Epoch 22/300
4/4             0s 15ms/step - loss:
0.0290
Epoch 23/300
4/4             0s 19ms/step - loss:
0.0229
Epoch 24/300
4/4             0s 19ms/step - loss:
0.0258
Epoch 25/300
4/4             0s 17ms/step - loss:
0.0252
Epoch 26/300
4/4             0s 23ms/step - loss:
0.0209
Epoch 27/300
4/4             0s 23ms/step - loss:
0.0288
Epoch 28/300
4/4             0s 23ms/step - loss:
0.0266
Epoch 29/300
4/4             0s 23ms/step - loss:
0.0217
Epoch 30/300
```

```
4/4             0s 23ms/step - loss:
0.0242
Epoch 31/300
4/4             0s 15ms/step - loss:
0.0343
Epoch 32/300
4/4             0s 26ms/step - loss:
0.0224
Epoch 33/300
4/4             0s 22ms/step - loss:
0.0272
Epoch 34/300
4/4             0s 23ms/step - loss:
0.0268
Epoch 35/300
4/4             0s 20ms/step - loss:
0.0240
Epoch 36/300
4/4             0s 20ms/step - loss:
0.0253
Epoch 37/300
4/4             0s 22ms/step - loss:
0.0213
Epoch 38/300
4/4             0s 20ms/step - loss:
0.0223
Epoch 39/300
4/4             0s 21ms/step - loss:
0.0258
Epoch 40/300
4/4             0s 20ms/step - loss:
0.0254
Epoch 41/300
4/4             0s 13ms/step - loss:
0.0256
Epoch 42/300
4/4             0s 12ms/step - loss:
0.0216
Epoch 43/300
4/4             0s 18ms/step - loss:
0.0239
Epoch 44/300
4/4             0s 18ms/step - loss:
0.0257
Epoch 45/300
4/4             0s 15ms/step - loss:
0.0258
Epoch 46/300
```

```
4/4             0s 20ms/step - loss:
0.0273
Epoch 47/300
4/4             0s 20ms/step - loss:
0.0218
Epoch 48/300
4/4             0s 22ms/step - loss:
0.0241
Epoch 49/300
4/4             0s 14ms/step - loss:
0.0265
Epoch 50/300
4/4             0s 21ms/step - loss:
0.0245
Epoch 51/300
4/4             0s 17ms/step - loss:
0.0292
Epoch 52/300
4/4             0s 20ms/step - loss:
0.0299
Epoch 53/300
4/4             0s 16ms/step - loss:
0.0299
Epoch 54/300
4/4             0s 19ms/step - loss:
0.0261
Epoch 55/300
4/4             0s 20ms/step - loss:
0.0224
Epoch 56/300
4/4             0s 21ms/step - loss:
0.0244
Epoch 57/300
4/4             0s 20ms/step - loss:
0.0218
Epoch 58/300
4/4             0s 21ms/step - loss:
0.0210
Epoch 59/300
4/4             0s 12ms/step - loss:
0.0253
Epoch 60/300
4/4             0s 15ms/step - loss:
0.0236
Epoch 61/300
4/4             0s 19ms/step - loss:
0.0282
Epoch 62/300
```

```
4/4             0s 17ms/step - loss:
0.0215
Epoch 63/300
4/4             0s 19ms/step - loss:
0.0225
Epoch 64/300
4/4             0s 19ms/step - loss:
0.0196
Epoch 65/300
4/4             0s 18ms/step - loss:
0.0198
Epoch 66/300
4/4             0s 17ms/step - loss:
0.0291
Epoch 67/300
4/4             0s 14ms/step - loss:
0.0221
Epoch 68/300
4/4             0s 19ms/step - loss:
0.0220
Epoch 69/300
4/4             0s 18ms/step - loss:
0.0221
Epoch 70/300
4/4             0s 19ms/step - loss:
0.0230
Epoch 71/300
4/4             0s 16ms/step - loss:
0.0290
Epoch 72/300
4/4             0s 19ms/step - loss:
0.0232
Epoch 73/300
4/4             0s 13ms/step - loss:
0.0247
Epoch 74/300
4/4             0s 13ms/step - loss:
0.0241
Epoch 75/300
4/4             0s 16ms/step - loss:
0.0242
Epoch 76/300
4/4             0s 16ms/step - loss:
0.0271
Epoch 77/300
4/4             0s 15ms/step - loss:
0.0229
Epoch 78/300
```

```
4/4             0s 14ms/step - loss:
0.0199
Epoch 79/300
4/4             0s 14ms/step - loss:
0.0225
Epoch 80/300
4/4             0s 15ms/step - loss:
0.0222
Epoch 81/300
4/4             0s 20ms/step - loss:
0.0242
Epoch 82/300
4/4             0s 20ms/step - loss:
0.0246
Epoch 83/300
4/4             0s 19ms/step - loss:
0.0205
Epoch 84/300
4/4             0s 17ms/step - loss:
0.0246
Epoch 85/300
4/4             0s 22ms/step - loss:
0.0287
Epoch 86/300
4/4             0s 19ms/step - loss:
0.0248
Epoch 87/300
4/4             0s 12ms/step - loss:
0.0216
Epoch 88/300
4/4             0s 12ms/step - loss:
0.0219
Epoch 89/300
4/4             0s 17ms/step - loss:
0.0206
Epoch 90/300
4/4             0s 19ms/step - loss:
0.0229
Epoch 91/300
4/4             0s 20ms/step - loss:
0.0298
Epoch 92/300
4/4             0s 19ms/step - loss:
0.0259
Epoch 93/300
4/4             0s 20ms/step - loss:
0.0223
Epoch 94/300
```

```
4/4             0s 20ms/step - loss:
0.0229
Epoch 95/300
4/4             0s 14ms/step - loss:
0.0270
Epoch 96/300
4/4             0s 21ms/step - loss:
0.0186
Epoch 97/300
4/4             0s 12ms/step - loss:
0.0230
Epoch 98/300
4/4             0s 13ms/step - loss:
0.0261
Epoch 99/300
4/4             0s 20ms/step - loss:
0.0243
Epoch 100/300
4/4             0s 23ms/step - loss:
0.0223
Epoch 101/300
4/4             0s 23ms/step - loss:
0.0192
Epoch 102/300
4/4             0s 23ms/step - loss:
0.0254
Epoch 103/300
4/4             0s 19ms/step - loss:
0.0221
Epoch 104/300
4/4             0s 17ms/step - loss:
0.0242
Epoch 105/300
4/4             0s 19ms/step - loss:
0.0223
Epoch 106/300
4/4             0s 14ms/step - loss:
0.0211
Epoch 107/300
4/4             0s 13ms/step - loss:
0.0240
Epoch 108/300
4/4             0s 19ms/step - loss:
0.0212
Epoch 109/300
4/4             0s 21ms/step - loss:
0.0234
Epoch 110/300
```

```
4/4             0s 17ms/step - loss:
0.0162
Epoch 111/300
4/4             0s 21ms/step - loss:
0.0237
Epoch 112/300
4/4             0s 24ms/step - loss:
0.0243
Epoch 113/300
4/4             0s 21ms/step - loss:
0.0191
Epoch 114/300
4/4             0s 22ms/step - loss:
0.0314
Epoch 115/300
4/4             0s 15ms/step - loss:
0.0189
Epoch 116/300
4/4             0s 14ms/step - loss:
0.0179
Epoch 117/300
4/4             0s 20ms/step - loss:
0.0236
Epoch 118/300
4/4             0s 21ms/step - loss:
0.0209
Epoch 119/300
4/4             0s 21ms/step - loss:
0.0218
Epoch 120/300
4/4             0s 23ms/step - loss:
0.0232
Epoch 121/300
4/4             0s 24ms/step - loss:
0.0205
Epoch 122/300
4/4             0s 22ms/step - loss:
0.0221
Epoch 123/300
4/4             0s 13ms/step - loss:
0.0206
Epoch 124/300
4/4             0s 16ms/step - loss:
0.0206
Epoch 125/300
4/4             0s 20ms/step - loss:
0.0239
Epoch 126/300
```

```
4/4             0s 21ms/step - loss:
0.0195
Epoch 127/300
4/4             0s 18ms/step - loss:
0.0202
Epoch 128/300
4/4             0s 21ms/step - loss:
0.0196
Epoch 129/300
4/4             0s 23ms/step - loss:
0.0175
Epoch 130/300
4/4             0s 17ms/step - loss:
0.0164
Epoch 131/300
4/4             0s 12ms/step - loss:
0.0190
Epoch 132/300
4/4             0s 13ms/step - loss:
0.0215
Epoch 133/300
4/4             0s 20ms/step - loss:
0.0190
Epoch 134/300
4/4             0s 15ms/step - loss:
0.0203
Epoch 135/300
4/4             0s 16ms/step - loss:
0.0171
Epoch 136/300
4/4             0s 24ms/step - loss:
0.0160
Epoch 137/300
4/4             0s 21ms/step - loss:
0.0155
Epoch 138/300
4/4             0s 13ms/step - loss:
0.0178
Epoch 139/300
4/4             0s 13ms/step - loss:
0.0159
Epoch 140/300
4/4             0s 21ms/step - loss:
0.0198
Epoch 141/300
4/4             0s 21ms/step - loss:
0.0191
Epoch 142/300
```

```
4/4             0s 19ms/step - loss:
0.0176
Epoch 143/300
4/4             0s 21ms/step - loss:
0.0198
Epoch 144/300
4/4             0s 13ms/step - loss:
0.0169
Epoch 145/300
4/4             0s 12ms/step - loss:
0.0155
Epoch 146/300
4/4             0s 19ms/step - loss:
0.0227
Epoch 147/300
4/4             0s 21ms/step - loss:
0.0156
Epoch 148/300
4/4             0s 22ms/step - loss:
0.0178
Epoch 149/300
4/4             0s 20ms/step - loss:
0.0185
Epoch 150/300
4/4             0s 16ms/step - loss:
0.0153
Epoch 151/300
4/4             0s 19ms/step - loss:
0.0206
Epoch 152/300
4/4             0s 12ms/step - loss:
0.0201
Epoch 153/300
4/4             0s 13ms/step - loss:
0.0172
Epoch 154/300
4/4             0s 18ms/step - loss:
0.0161
Epoch 155/300
4/4             0s 22ms/step - loss:
0.0149
Epoch 156/300
4/4             0s 19ms/step - loss:
0.0158
Epoch 157/300
4/4             0s 15ms/step - loss:
0.0182
Epoch 158/300
```

```
4/4             0s 11ms/step - loss:
0.0172
Epoch 159/300
4/4             0s 16ms/step - loss:
0.0144
Epoch 160/300
4/4             0s 21ms/step - loss:
0.0197
Epoch 161/300
4/4             0s 21ms/step - loss:
0.0148
Epoch 162/300
4/4             0s 20ms/step - loss:
0.0109
Epoch 163/300
4/4             0s 22ms/step - loss:
0.0138
Epoch 164/300
4/4             0s 12ms/step - loss:
0.0162
Epoch 165/300
4/4             0s 12ms/step - loss:
0.0144
Epoch 166/300
4/4             0s 20ms/step - loss:
0.0164
Epoch 167/300
4/4             0s 16ms/step - loss:
0.0159
Epoch 168/300
4/4             0s 18ms/step - loss:
0.0154
Epoch 169/300
4/4             0s 22ms/step - loss:
0.0131
Epoch 170/300
4/4             0s 12ms/step - loss:
0.0136
Epoch 171/300
4/4             0s 16ms/step - loss:
0.0213
Epoch 172/300
4/4             0s 20ms/step - loss:
0.0165
Epoch 173/300
4/4             0s 21ms/step - loss:
0.0163
Epoch 174/300
```

```
4/4             0s 22ms/step - loss:
0.0179
Epoch 175/300
4/4             0s 12ms/step - loss:
0.0141
Epoch 176/300
4/4             0s 15ms/step - loss:
0.0149
Epoch 177/300
4/4             0s 20ms/step - loss:
0.0122
Epoch 178/300
4/4             0s 20ms/step - loss:
0.0140
Epoch 179/300
4/4             0s 24ms/step - loss:
0.0190
Epoch 180/300
4/4             0s 13ms/step - loss:
0.0183
Epoch 181/300
4/4             0s 13ms/step - loss:
0.0189
Epoch 182/300
4/4             0s 19ms/step - loss:
0.0179
Epoch 183/300
4/4             0s 21ms/step - loss:
0.0189
Epoch 184/300
4/4             0s 19ms/step - loss:
0.0135
Epoch 185/300
4/4             0s 15ms/step - loss:
0.0167
Epoch 186/300
4/4             0s 13ms/step - loss:
0.0189
Epoch 187/300
4/4             0s 19ms/step - loss:
0.0182
Epoch 188/300
4/4             0s 19ms/step - loss:
0.0149
Epoch 189/300
4/4             0s 20ms/step - loss:
0.0167
Epoch 190/300
```

```
4/4             0s 13ms/step - loss:
0.0136
Epoch 191/300
4/4             0s 15ms/step - loss:
0.0133
Epoch 192/300
4/4             0s 15ms/step - loss:
0.0154
Epoch 193/300
4/4             0s 16ms/step - loss:
0.0135
Epoch 194/300
4/4             0s 17ms/step - loss:
0.0139
Epoch 195/300
4/4             0s 17ms/step - loss:
0.0146
Epoch 196/300
4/4             0s 18ms/step - loss:
0.0141
Epoch 197/300
4/4             0s 14ms/step - loss:
0.0128
Epoch 198/300
4/4             0s 14ms/step - loss:
0.0114
Epoch 199/300
4/4             0s 15ms/step - loss:
0.0155
Epoch 200/300
4/4             0s 19ms/step - loss:
0.0151
Epoch 201/300
4/4             0s 15ms/step - loss:
0.0166
Epoch 202/300
4/4             0s 14ms/step - loss:
0.0150
Epoch 203/300
4/4             0s 14ms/step - loss:
0.0152
Epoch 204/300
4/4             0s 15ms/step - loss:
0.0152
Epoch 205/300
4/4             0s 16ms/step - loss:
0.0128
Epoch 206/300
```

```
4/4             0s 20ms/step - loss:
0.0146
Epoch 207/300
4/4             0s 13ms/step - loss:
0.0172
Epoch 208/300
4/4             0s 13ms/step - loss:
0.0146
Epoch 209/300
4/4             0s 19ms/step - loss:
0.0166
Epoch 210/300
4/4             0s 20ms/step - loss:
0.0138
Epoch 211/300
4/4             0s 14ms/step - loss:
0.0171
Epoch 212/300
4/4             0s 19ms/step - loss:
0.0135
Epoch 213/300
4/4             0s 18ms/step - loss:
0.0140
Epoch 214/300
4/4             0s 21ms/step - loss:
0.0154
Epoch 215/300
4/4             0s 15ms/step - loss:
0.0181
Epoch 216/300
4/4             0s 14ms/step - loss:
0.0173
Epoch 217/300
4/4             0s 13ms/step - loss:
0.0117
Epoch 218/300
4/4             0s 19ms/step - loss:
0.0122
Epoch 219/300
4/4             0s 18ms/step - loss:
0.0111
Epoch 220/300
4/4             0s 12ms/step - loss:
0.0141
Epoch 221/300
4/4             0s 15ms/step - loss:
0.0164
Epoch 222/300
```

```
4/4             0s 17ms/step - loss:
0.0135
Epoch 223/300
4/4             0s 20ms/step - loss:
0.0144
Epoch 224/300
4/4             0s 14ms/step - loss:
0.0177
Epoch 225/300
4/4             0s 14ms/step - loss:
0.0186
Epoch 226/300
4/4             0s 19ms/step - loss:
0.0168
Epoch 227/300
4/4             0s 19ms/step - loss:
0.0161
Epoch 228/300
4/4             0s 15ms/step - loss:
0.0126
Epoch 229/300
4/4             0s 12ms/step - loss:
0.0176
Epoch 230/300
4/4             0s 17ms/step - loss:
0.0186
Epoch 231/300
4/4             0s 20ms/step - loss:
0.0171
Epoch 232/300
4/4             0s 20ms/step - loss:
0.0155
Epoch 233/300
4/4             0s 14ms/step - loss:
0.0175
Epoch 234/300
4/4             0s 12ms/step - loss:
0.0164
Epoch 235/300
4/4             0s 15ms/step - loss:
0.0142
Epoch 236/300
4/4             0s 15ms/step - loss:
0.0170
Epoch 237/300
4/4             0s 15ms/step - loss:
0.0132
Epoch 238/300
```

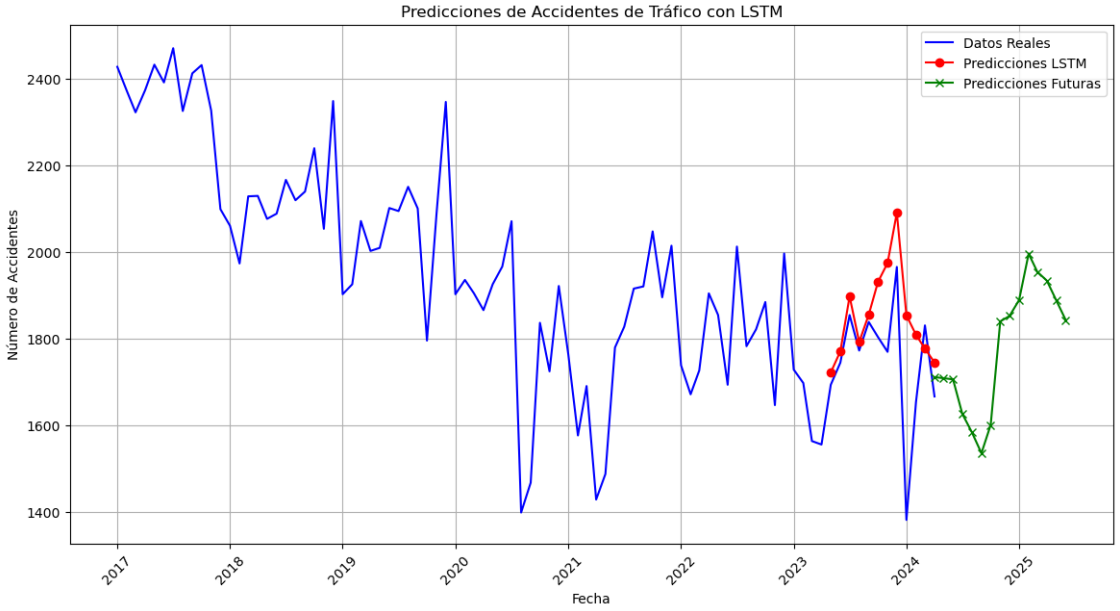
```
4/4             0s 22ms/step - loss:
0.0150
Epoch 239/300
4/4             0s 19ms/step - loss:
0.0116
Epoch 240/300
4/4             0s 14ms/step - loss:
0.0154
Epoch 241/300
4/4             0s 20ms/step - loss:
0.0116
Epoch 242/300
4/4             0s 13ms/step - loss:
0.0141
Epoch 243/300
4/4             0s 13ms/step - loss:
0.0129
Epoch 244/300
4/4             0s 19ms/step - loss:
0.0120
Epoch 245/300
4/4             0s 22ms/step - loss:
0.0117
Epoch 246/300
4/4             0s 18ms/step - loss:
0.0149
Epoch 247/300
4/4             0s 12ms/step - loss:
0.0178
Epoch 248/300
4/4             0s 15ms/step - loss:
0.0158
Epoch 249/300
4/4             0s 19ms/step - loss:
0.0109
Epoch 250/300
4/4             0s 21ms/step - loss:
0.0113
Epoch 251/300
4/4             0s 14ms/step - loss:
0.0141
Epoch 252/300
4/4             0s 14ms/step - loss:
0.0118
Epoch 253/300
4/4             0s 20ms/step - loss:
0.0147
Epoch 254/300
```

```
4/4             0s 22ms/step - loss:
0.0106
Epoch 255/300
4/4             0s 13ms/step - loss:
0.0144
Epoch 256/300
4/4             0s 13ms/step - loss:
0.0124
Epoch 257/300
4/4             0s 19ms/step - loss:
0.0144
Epoch 258/300
4/4             0s 20ms/step - loss:
0.0161
Epoch 259/300
4/4             0s 13ms/step - loss:
0.0153
Epoch 260/300
4/4             0s 15ms/step - loss:
0.0151
Epoch 261/300
4/4             0s 21ms/step - loss:
0.0146
Epoch 262/300
4/4             0s 19ms/step - loss:
0.0139
Epoch 263/300
4/4             0s 13ms/step - loss:
0.0137
Epoch 264/300
4/4             0s 15ms/step - loss:
0.0183
Epoch 265/300
4/4             0s 24ms/step - loss:
0.0127
Epoch 266/300
4/4             0s 13ms/step - loss:
0.0128
Epoch 267/300
4/4             0s 13ms/step - loss:
0.0175
Epoch 268/300
4/4             0s 22ms/step - loss:
0.0174
Epoch 269/300
4/4             0s 16ms/step - loss:
0.0135
Epoch 270/300
```

```
4/4             0s 13ms/step - loss:
0.0148
Epoch 271/300
4/4             0s 12ms/step - loss:
0.0123
Epoch 272/300
4/4             0s 17ms/step - loss:
0.0151
Epoch 273/300
4/4             0s 21ms/step - loss:
0.0134
Epoch 274/300
4/4             0s 14ms/step - loss:
0.0128
Epoch 275/300
4/4             0s 13ms/step - loss:
0.0151
Epoch 276/300
4/4             0s 16ms/step - loss:
0.0111
Epoch 277/300
4/4             0s 16ms/step - loss:
0.0155
Epoch 278/300
4/4             0s 13ms/step - loss:
0.0165
Epoch 279/300
4/4             0s 15ms/step - loss:
0.0125
Epoch 280/300
4/4             0s 13ms/step - loss:
0.0147
Epoch 281/300
4/4             0s 13ms/step - loss:
0.0140
Epoch 282/300
4/4             0s 13ms/step - loss:
0.0096
Epoch 283/300
4/4             0s 17ms/step - loss:
0.0142
Epoch 284/300
4/4             0s 20ms/step - loss:
0.0152
Epoch 285/300
4/4             0s 20ms/step - loss:
0.0133
Epoch 286/300
```

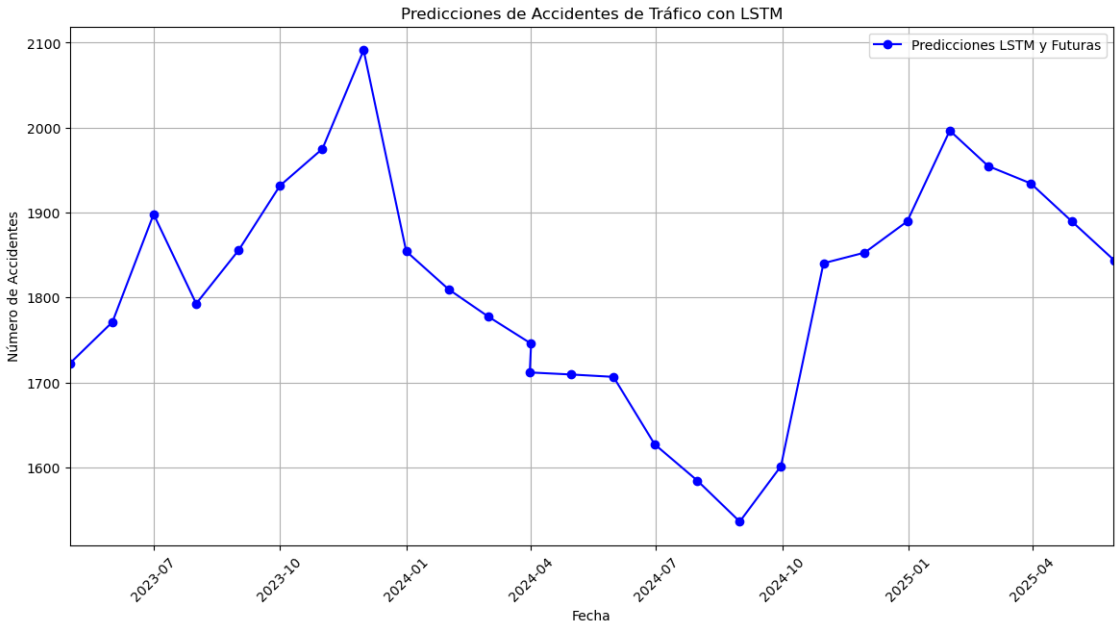
```
4/4             0s 13ms/step - loss:
0.0146
Epoch 287/300
4/4             0s 13ms/step - loss:
0.0111
Epoch 288/300
4/4             0s 15ms/step - loss:
0.0138
Epoch 289/300
4/4             0s 13ms/step - loss:
0.0151
Epoch 290/300
4/4             0s 13ms/step - loss:
0.0162
Epoch 291/300
4/4             0s 17ms/step - loss:
0.0134
Epoch 292/300
4/4             0s 16ms/step - loss:
0.0128
Epoch 293/300
4/4             0s 14ms/step - loss:
0.0153
Epoch 294/300
4/4             0s 12ms/step - loss:
0.0167
Epoch 295/300
4/4             0s 17ms/step - loss:
0.0118
Epoch 296/300
4/4             0s 19ms/step - loss:
0.0111
Epoch 297/300
4/4             0s 19ms/step - loss:
0.0133
Epoch 298/300
4/4             0s 13ms/step - loss:
0.0129
Epoch 299/300
4/4             0s 14ms/step - loss:
0.0129
Epoch 300/300
4/4             0s 15ms/step - loss:
0.0137
1/1             0s 452ms/step
1/1             0s 390ms/step
1/1             0s 34ms/step
1/1             0s 46ms/step
```

```
1/1             0s 40ms/step
1/1             0s 43ms/step
1/1             0s 36ms/step
1/1             0s 28ms/step
1/1             0s 29ms/step
1/1             0s 44ms/step
1/1             0s 23ms/step
1/1             0s 46ms/step
1/1             0s 50ms/step
1/1             0s 36ms/step
1/1             0s 37ms/step
1/1             0s 30ms/step
```




[140]:
```
fechas_futuras = pd.date_range(start=ultima_fecha - pd.DateOffset(months=1),␣
 ↪periods=15, freq='M')

predicciones = predicciones.flatten()  # Convertir a 1D si es necesario
predicciones_futuras = predicciones_futuras.flatten()  # Convertir a 1D si es␣
 ↪necesario

fechas_totales = pd.concat([df_test['FECHA'], pd.Series(fechas_futuras)])
predicciones_totales = pd.concat([pd.Series(predicciones), pd.
 ↪Series(predicciones_futuras)])

plt.figure(figsize=(14, 7))
```

```
plt.plot(fechas_totales, predicciones_totales, label='Predicciones LSTM y␣
 ↪Futuras', color='blue', marker='o')

plt.title('Predicciones de Accidentes de Tráfico con LSTM')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.legend()
plt.grid()
plt.xticks(rotation=45)

plt.xlim(pd.Timestamp('2023-05-01'), pd.Timestamp('2025-05-30'))

plt.show()
```




[141]:
```
fechas_futuras = pd.date_range(start=ultima_fecha - pd.DateOffset(months=1),␣
 ↪periods=15, freq='M')

predicciones = predicciones.flatten()  # Convertir a 1D si es necesario
predicciones_futuras = predicciones_futuras.flatten()  # Convertir a 1D si es␣
 ↪necesario

fechas_totales = pd.concat([df_test['FECHA'], pd.Series(fechas_futuras)]).
 ↪reset_index(drop=True)
predicciones_totales = pd.concat([pd.Series(predicciones), pd.
 ↪Series(predicciones_futuras)]).reset_index(drop=True)
```

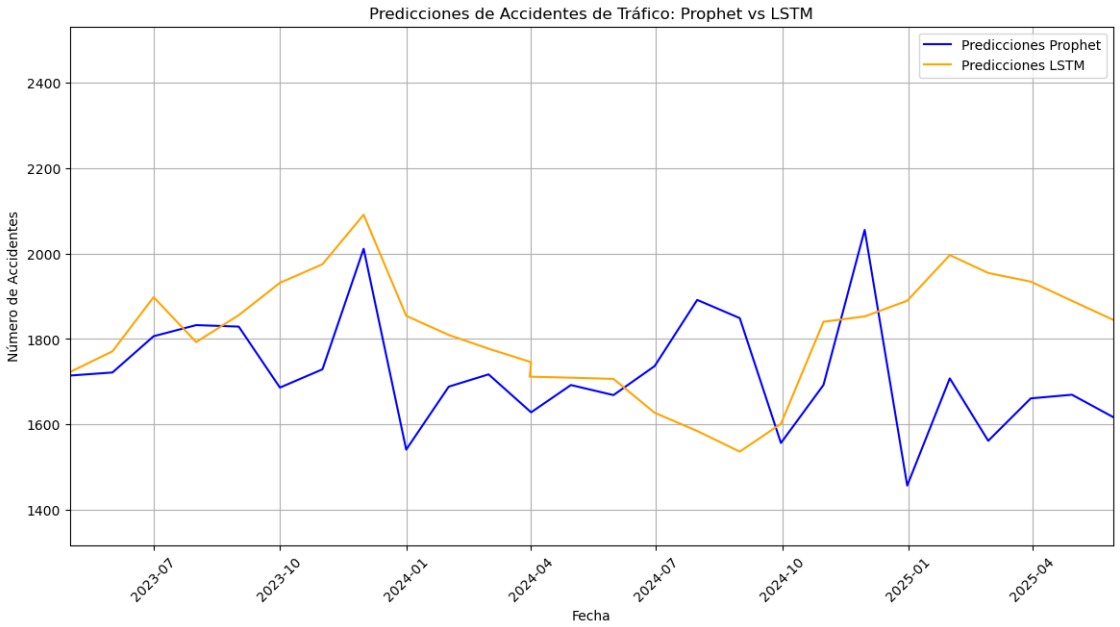
```
tabla_predicciones_lstm = pd.DataFrame({
    'Fecha': fechas_totales,
    'Predicción': predicciones_totales
})

tabla_predicciones_lstm
```

[141]:

| | Fecha | Predicción |
|---|---|---|
| 0 | 2023-05-01 | 1722.386719 |
| 1 | 2023-06-01 | 1771.094238 |
| 2 | 2023-07-01 | 1898.042969 |
| 3 | 2023-08-01 | 1792.699951 |
| 4 | 2023-09-01 | 1855.797119 |
| 5 | 2023-10-01 | 1931.570068 |
| 6 | 2023-11-01 | 1975.100708 |
| 7 | 2023-12-01 | 2091.249023 |
| 8 | 2024-01-01 | 1854.342529 |
| 9 | 2024-02-01 | 1809.445923 |
| 10 | 2024-03-01 | 1777.246582 |
| 11 | 2024-04-01 | 1745.621582 |
| 12 | 2024-03-31 | 1711.669434 |
| 13 | 2024-04-30 | 1709.270752 |
| 14 | 2024-05-31 | 1706.471069 |
| 15 | 2024-06-30 | 1627.044678 |
| 16 | 2024-07-31 | 1584.426514 |
| 17 | 2024-08-31 | 1536.031860 |
| 18 | 2024-09-30 | 1601.280273 |
| 19 | 2024-10-31 | 1840.334717 |
| 20 | 2024-11-30 | 1852.808228 |
| 21 | 2024-12-31 | 1889.579590 |
| 22 | 2025-01-31 | 1996.489990 |
| 23 | 2025-02-28 | 1954.735474 |
| 24 | 2025-03-31 | 1934.314209 |
| 25 | 2025-04-30 | 1889.466675 |
| 26 | 2025-05-31 | 1843.426147 |

[142]:
```
fechas_futuras = pd.date_range(start=ultima_fecha - pd.DateOffset(months=1),␣
 ↪periods=15, freq='M')

predicciones = predicciones.flatten()  # Convertir a 1D si es necesario
predicciones_futuras = predicciones_futuras.flatten()  # Convertir a 1D si es␣
 ↪necesario

fechas_totales = pd.concat([df_test['FECHA'], pd.Series(fechas_futuras)])
predicciones_totales = pd.concat([pd.Series(predicciones), pd.
 ↪Series(predicciones_futuras)])
```

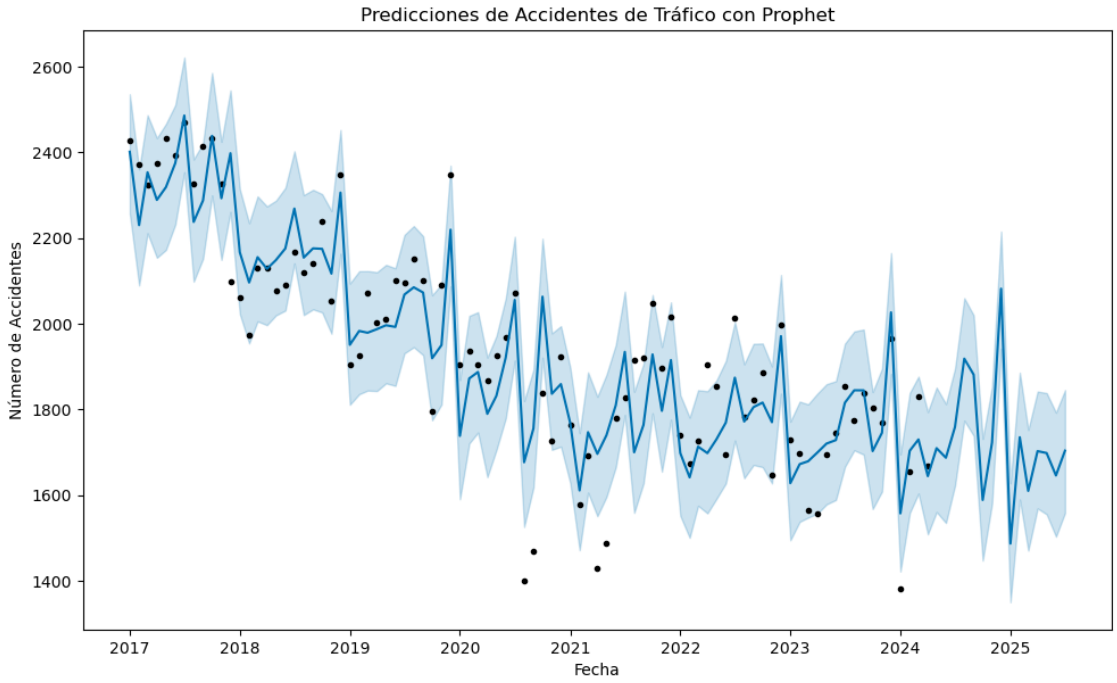
```
plt.figure(figsize=(14, 7))

plt.plot(forecast['ds'], forecast['yhat'], label='Predicciones Prophet',␣
 ↪color='blue', linestyle='-')

plt.plot(fechas_totales, predicciones_totales, label='Predicciones LSTM',␣
 ↪color='orange', linestyle='-')

# Añadir título y etiquetas
plt.title('Predicciones de Accidentes de Tráfico: Prophet vs LSTM')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.legend()
plt.grid()
plt.xticks(rotation=45)

plt.xlim(pd.Timestamp('2023-05-01'), pd.Timestamp('2025-05-30'))

plt.show()
```




[143]:
```
df = df_sin_anomalias[['FECHA', 'num_accidentes']].rename(columns={'FECHA':␣
 ↪'ds', 'num_accidentes': 'y'})

model = Prophet(changepoint_prior_scale=0.1)  # Ajusta este valor según sea␣
 ↪necesario
model.fit(df)
```

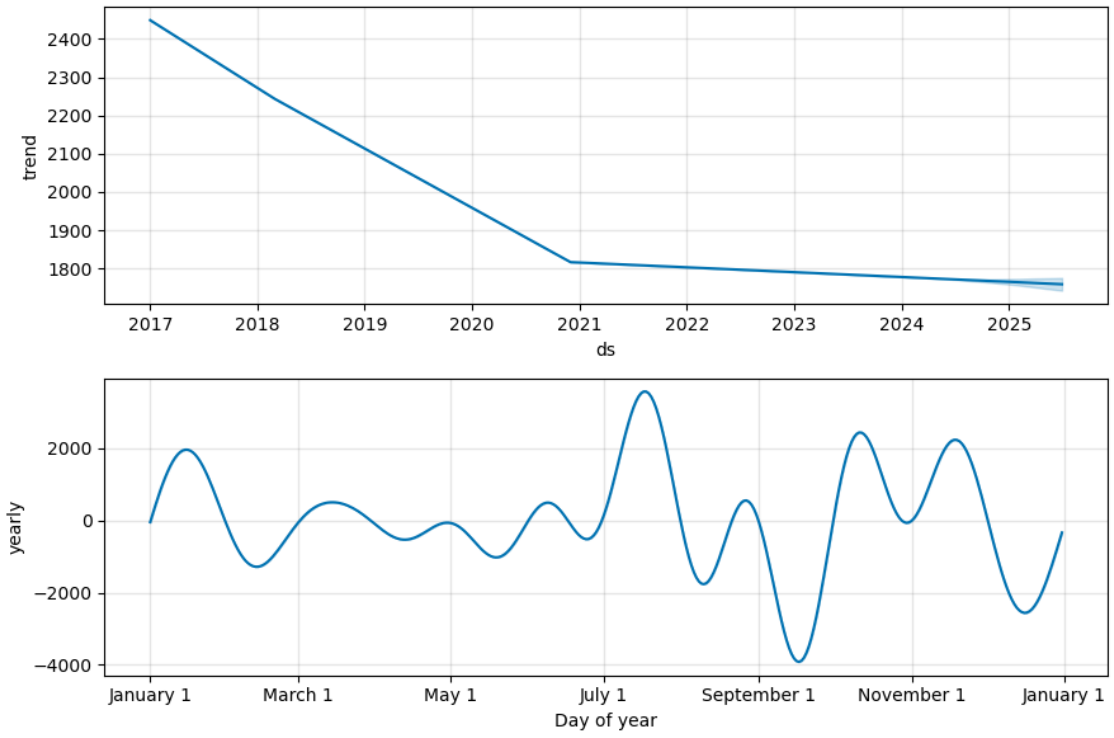
```
future = model.make_future_dataframe(periods=15, freq='M')  # 15 meses hacia el␣
↪futuro

forecast = model.predict(future)

fig = model.plot(forecast)
plt.title('Predicciones de Accidentes de Tráfico con Prophet')
plt.xlabel('Fecha')
plt.ylabel('Número de Accidentes')
plt.grid()
plt.show()

changepoints = model.changepoints
print("Puntos de Cambio Identificados:")
print(changepoints)

fig2 = model.plot_components(forecast)
plt.show()
```

```
21:39:49 - cmdstanpy - INFO - Chain [1] start processing
21:39:50 - cmdstanpy - INFO - Chain [1] done processing
```




```
Puntos de Cambio Identificados:
3    2017-04-01
```

```
6     2017-07-01
8     2017-09-01
11    2017-12-01
14    2018-03-01
17    2018-06-01
19    2018-08-01
22    2018-11-01
25    2019-02-01
28    2019-05-01
30    2019-07-01
33    2019-10-01
36    2020-01-01
39    2020-04-01
41    2020-06-01
44    2020-09-01
47    2020-12-01
50    2021-03-01
52    2021-05-01
55    2021-08-01
58    2021-11-01
61    2022-02-01
63    2022-04-01
66    2022-07-01
69    2022-10-01
Name: ds, dtype: datetime64[ns]
```



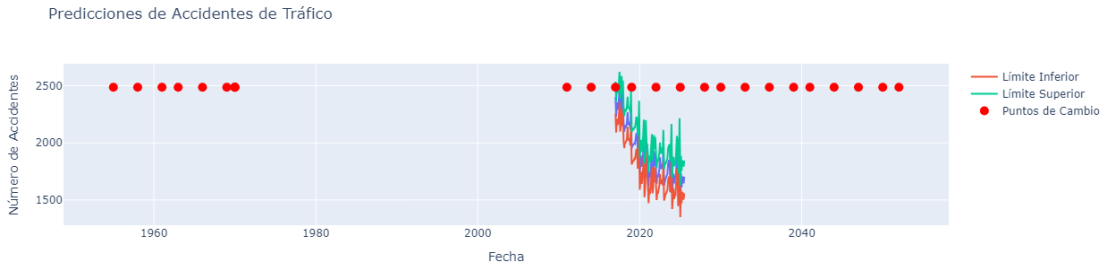
[144]:
```
print("Puntos de Cambio Identificados:")
print(model.changepoints)
```

```
Puntos de Cambio Identificados:
3    2017-04-01
6    2017-07-01
8    2017-09-01
11   2017-12-01
14   2018-03-01
17   2018-06-01
19   2018-08-01
22   2018-11-01
25   2019-02-01
28   2019-05-01
30   2019-07-01
33   2019-10-01
36   2020-01-01
39   2020-04-01
41   2020-06-01
44   2020-09-01
47   2020-12-01
50   2021-03-01
52   2021-05-01
55   2021-08-01
58   2021-11-01
61   2022-02-01
63   2022-04-01
66   2022-07-01
69   2022-10-01
Name: ds, dtype: datetime64[ns]
```

[145]:
```
changepoints_dates = model.changepoints.index  # Si las fechas están en el␣
↪índice
```

[146]:
```
fig = px.line(forecast, x='ds', y='yhat', title='Predicciones de Accidentes de␣
↪Tráfico',
              labels={'yhat': 'Número de Accidentes', 'ds': 'Fecha'})
fig.add_scatter(x=forecast['ds'], y=forecast['yhat_lower'], mode='lines',␣
↪name='Límite Inferior')
fig.add_scatter(x=forecast['ds'], y=forecast['yhat_upper'], mode='lines',␣
↪name='Límite Superior')

changepoints_dates = model.changepoints.index  # Usar el índice si no hay␣
↪columna 'ds'
fig.add_scatter(x=changepoints_dates,
```

```
                y=[forecast['yhat'].max()]*len(changepoints_dates),
                mode='markers',
                name='Puntos de Cambio',
                marker=dict(color='red', size=10))

fig.show()
```




[147]:
```
print(model.changepoints)
```

```
3    2017-04-01
6    2017-07-01
8    2017-09-01
11   2017-12-01
14   2018-03-01
17   2018-06-01
19   2018-08-01
22   2018-11-01
25   2019-02-01
28   2019-05-01
30   2019-07-01
33   2019-10-01
36   2020-01-01
39   2020-04-01
41   2020-06-01
44   2020-09-01
47   2020-12-01
50   2021-03-01
52   2021-05-01
55   2021-08-01
58   2021-11-01
61   2022-02-01
63   2022-04-01
66   2022-07-01
69   2022-10-01
Name: ds, dtype: datetime64[ns]
```

[148]:
```
changepoints_dates
```

[148]:
```
Index([ 3,  6,  8, 11, 14, 17, 19, 22, 25, 28, 30, 33, 36, 39, 41, 44, 47, 50,
       52, 55, 58, 61, 63, 66, 69],
      dtype='int64')
```

[149]:
```
predicciones_prophet = forecast['yhat'][-12:].values  # Cambia el rango según␣
↪tus datos
datos_reales = df_sin_anomalias['num_accidentes'].values[-12:]  # Cambia el␣
↪rango según tus datos

mse_prophet = mean_squared_error(datos_reales, predicciones_prophet)
rmse_prophet = np.sqrt(mse_prophet)
mae_prophet = mean_absolute_error(datos_reales, predicciones_prophet)

print(f"Métricas para Prophet:")
print(f"MSE: {mse_prophet}")
print(f"RMSE: {rmse_prophet}")
print(f"MAE: {mae_prophet}")
```

```
Métricas para Prophet:
MSE: 47561.07720840036
RMSE: 218.08502288878154
MAE: 184.62032130750666
```