



Barcelona StreetWatch

Extracting, Loading and Transforming traffic accident data in the city of Barcelona

NoSQL Lab with Python & MongoDB

Daniel Herrera-Russert

November 6, 2024

1	Summary	2
2	Requirements & Configuration	3
3	ELT Process	5
3.1	DB Setup	5
3.2	Extract	5
3.3	Load	9
3.4	Transform	11
3.5	Datastructure	20
4	Data analysis	21
4.1	Where in the city are accidents recorded?	21
4.2	Do more accidents happen in more populous neighborhoods?	29
4.3	What are the temporal patterns of the recorded street accidents?	33
4.4	Does a higher rate of male to female inhabitants per neighborhood correlate with accident counts?	40
5	Conclusions	46
6	Learnings	47

1 Summary

The following project focuses on a 2023 data set on urban accidents in Barcelona, with the purpose of delving into the causes and patterns of traffic incidents across different times and locations in the city.

Data published in the [Annual Statistics of Traffic Accidents of Catalunya](#) reveals that the majority of traffic incidents in the Spanish region take place within urban limits, with the worst cases (fatalities) overwhelmingly involving motorcyclists, and individuals over 65 years of age constituting the largest portion of pedestrian victimization. Additionally, the increase in bicycle usage having been correlated with a rise in bicycle-related incidents as well as low compliance rates for mandatory practices such as seat belt and child restraint system usage, are suggestive of a need for stricter enforcement of traffic laws.

To gain deeper insights into the factors contributing to these accidents, analyzing the 2023 traffic accident data collected by Barcelona's Municipal Police Force (Guàrdia Urbana) is a logical first step. Comparing this data with population statistics from various neighborhoods can be particularly insightful, as it would highlight the demographic and socio-economic characteristics of different areas and help identify specific trends and risk factors associated with traffic violations. For example, neighborhoods with higher pedestrian traffic may exhibit different accident patterns compared to those with more vehicle congestion.



The data basis pertaining both topics has been sourced from the [Open Data portal of the City Council of Barcelona](#). The portal is an initiative by the Barcelona City Council aimed at promoting transparency, citizen participation, and innovation through open access to public data. This digital platform offers a wide array of datasets on various city aspects such as transportation, environment, demographics, and economy, enabling developers, researchers, and the general public to analyze, create, and share applications that enhance the understanding and management of the city.

This nuanced approach enables the development of targeted safety initiatives tailored to the unique needs of each community, ultimately helping to reduce traffic incidents and improve overall road safety in Barcelona.

2 Requirements & Configuration

```
[1]: ! pip3 list --disable-pip-version-check | findstr "pymongo dnspython pandas"
```

dnspython	2.7.0
pandas	2.2.3
pymongo	4.10.1

```
[2]: import pymongo
import pprint as pp
import pandas as pd
import numpy as np
import requests
import json
import time
import string
import datetime
import seaborn as sns
import matplotlib.pyplot as plt
from adjustText import adjust_text # to make some adjustments to the labels in plots
from pymongo import GEOSPHERE
import kaleido

# For the maps:
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
import chart_studio.plotly as py
```

```
[3]: # Made updates to the code: added display.
# pandas configuration
pd.set_option('display.precision', 2)
pd.set_option('display.max_rows', 30)
pd.set_option('display.max_colwidth', 50)
# pd.describe_option('max_rows')
# pd.describe_option('precision')
# pd.describe_option('max_colwidth')
```

The topic of choice is based on the data set ‘Accidents by driver cause managed by the Guàrdia Urbana in the city of Barcelona’, which can be obtained via API from the Open Data project of the City of Barcelona, without the need of a personal token or API key. The file in question is 2023_ACCIDENTS_CAUSA_CONDUCTOR_GU_BCN_.csv, which corresponds to the id 5a040155-38b3-4b19-a4b0-c84a0618d363 from their resource catalogue.

The data set contains records collected by the Guàrdia Urbana (Barcelona’s municipal police force) pertaining traffic accidents, and contains information regarding location (coordinates), neighborhood and district, time of the day, date and reason for police involvement.

Additionally, in order to enhance the value of the insights, a second data set is also obtained

via API from the same Open Data Portal, corresponding to the ‘Population by sex’. The file in question is found under 2023_pad_mdbas_sexe.json with 468882a5-ec03-48b2-af05-d77df540226d in the resource catalogue.

The descriptive data, names of keys/variables and strings throughout the sets are in Catalan, one of the official languages spoken in the city, and the official language of institutional documents. For the sake of the exercise, the data will be kept in its original form, and interpreted and translated when necessary.

The data from both sources corresponds to the year 2023.

3 ELT Process

3.1 DB Setup

The following code snippets provide the details and credentials for accessing the already existing MongoDB database, as well as dumping all contents if anything is existing beforehand.

```
[4]: # API URLs
API_URL_1 = "https://opendata-ajuntament.barcelona.cat/data/api/action/
            →datastore_search?resource_id=5a040155-38b3-4b19-a4b0-c84a0618d363"
API_URL_2 = "https://opendata-ajuntament.barcelona.cat/data/api/action/
            →datastore_search?resource_id=d0e4ec78-e274-4300-a3bc-cb85cf79014d"

# Database data
CNX_STR = "mongodb+srv://waherrer:admin@mongocluster.pcvay.mongodb.net/"
DB_NAME = "BarcelonaDB"
COLL_NAME_1 = "traffic_accidents"
COLL_NAME_2 = "population"
```

```
[5]: # Connection to MongoDB
client = pymongo.MongoClient(CNX_STR)
db = client[DB_NAME]
accidents_collection = db[COLL_NAME_1]
population_collection = db[COLL_NAME_2]
```

```
[6]: # Remove all existing documents from collection 'traffic_accidents'
accidents_collection.drop()
accidents_collection.count_documents({})
```

```
[6]: 0
```

```
[7]: # Remove all existing documents from collection 'population'
population_collection.drop()
population_collection.count_documents({})
```

```
[7]: 0
```

3.2 Extract

Accidents by Driver Cause Data Set We begin by extracting and processing the ‘Accidents by Driver Cause’ data set from the API source. A selection of variables has been made that is more relevant to the future data analysis. Additionally, the latitude and longitude have been nested as a substructure within a new coordinates variable.

```
[8]: # Download all accidents data
# Limit has been set to the one shown in the documentation
url = f'{API_URL_1}&limit=12794'
```

```

# Sending a GET request to the API and checking for success
response = requests.get(url)
response.raise_for_status() # Raise an error for bad responses (4XX or 5XX)

data = response.json()

# Print the length of the raw data:
print(f'The total number of top-level keys in the raw JSON data: {len(data)}')

# Number of records:
if 'result' in data and 'records' in data['result']:
    num_records = len(data['result']['records'])
    print(f'The total number of accident records: {num_records}')
else:
    print("No records found in the data.")

```

The total number of top-level keys in the raw JSON data: 3

The total number of accident records: 12794

```
[9]: # Define a function to extract specific data items from the records
def extract_values_accident(accident):
    values = {
        'id': int(accident['_id']),
        'district_name': accident['Nom_districte'],
        'neighborhood_name': accident['Nom_barri'],
        'street_name': accident['Nom_carrer'],
        'date': f'{accident['NK_Any']}-{accident['Mes_any']}-{accident['Dia_mes']}',
        'day_of_week': accident['Descripcio_dia_setmana'],
        'time_of_day': accident['Hora_dia'],
        'cause': accident['Descripcio_causa_mediata'],
        'coordinates': {
            'latitude': float(accident['Latitud_WGS84']),
            'longitude': float(accident['Longitud_WGS84'])}
    }
    return values
```

```
[10]: # Access the list of accident records
accidents_list = data['result']['records'] # This should be a list of accidents

# Initialize an empty list to hold processed accident records
accident_list = []

# Loop through each accident record and apply the extraction function
for accident in accidents_list:
    extracted_accident = extract_values_accident(accident)
    # Append accident to the accident list
    accident_list.append(extracted_accident)
```

```
# Check the first item
accident_list[0]
```

[10]:

```
{'id': 1,
'district_name': 'Sants-Montjuïc',
'neighborhood_name': 'la Marina del Prat Vermell',
'street_name': 'Litoral (Llobregat)',
'date': '2023-6-14',
'day_of_week': 'Dimecres',
'time_of_day': '18',
'cause': 'Gir indegit o sense precaució',
'coordinates': {'latitude': 41.343237595, 'longitude': 2.1309991}}
```

[11]:

```
print(len(accident_list))
```

```
12794
```

Population by District Next we extract the ‘Population by sex’ data set from the API source. In the transformation step we ensure the items are uniformly named for further processing.

```
[12]: # Download all population data
# Limit has been set to the one shown in the documentation
url = f'{API_URL_2}&limit=2136'

# Sending a GET request to the API and checking for success
response = requests.get(url)
response.raise_for_status() # Raise an error for bad responses (4XX or 5XX)

data = response.json()

# Print the length of the raw data:
print(f'The total number of top-level keys in the raw JSON data: {len(data)}')

# Number of records:
if 'result' in data and 'records' in data['result']:
    num_records = len(data['result']['records'])
    print(f'The total number of population records: {num_records}')
else:
    print("No records found in the data.")
```

```
The total number of top-level keys in the raw JSON data: 3
The total number of population records: 2136
```

[13]:

```
# First item
data['result']['records'][0]
```

```
[13]: {'Codi_Districte': '1',
       'SEXЕ': '1',
       'Nom_Districte': 'Ciutat Vella',
       'Codi_Barri': '1',
       'Nom_Barri': 'el Raval',
       'AEB': '1',
       'Seccio_Censal': '1001',
       'Valor': '606',
       'Data_Refencia': '2023-01-01T00:00:00',
       '_id': 1}
```

```
[14]: # Define a function to extract specific data items from the records
def extract_values_population(population):
    values = {
        'id': int(population['_id']),
        'district_name': population['Nom_Districte'],
        'neighborhood_name': population['Nom_Barri'],
        'Population': int(population['Valor']),
        'Sex': population['SEXЕ']
    }
    return values
```

```
[15]: # Access the list of population records
population_list_raw = data['result']['records'] # This should be a list of_
→population records

# New empty list to hold selected and processed population records
population_list = [] # Use a new list for processed data

# Loop through each population record and apply the extraction function
for population in population_list_raw:
    extracted_population = extract_values_population(population)
    # Append processed record to the new list
    population_list.append(extracted_population)

print(population_list[0])

{'id': 1, 'district_name': 'Ciutat Vella', 'neighborhood_name': 'el Raval',
'Population': 606, 'Sex': '1'}
```

```
[16]: print(len(population_list))
```

2136

3.3 Load

In the second stage of the ELT process, we proceed by loading the processed data, stored as dictionary structures (key pairs) in Python, into the MongoDB database by specifying the appropriate target collections. To prevent duplicate entries or data overlap, we first ensure that the collections are emptied before loading new data. This step is performed for both data sets to maintain data integrity and consistency within the database, first with the traffic accident data:

```
[17]: # Check if the accidents collection is empty
if accidents_collection.count_documents({}) == 0:
    accidents_collection.insert_many(incident_list);
else:
    print("Accidents collection is not empty. No documents were inserted.")
```

```
[18]: # count number of documents inserted
accidents_collection.count_documents({})
```

```
[18]: 12794
```

And next with the population data:

```
[19]: # Check if the accidents collection is empty
if population_collection.count_documents({}) == 0:
    population_collection.insert_many(population_list);
else:
    print("Population collection is not empty. No documents were inserted.")
```

```
[20]: # count number of documents inserted
population_collection.count_documents({})
```

```
[20]: 2136
```

We extract one record from each data set, and can observe that the structure has remained the same as in the original data sets, except for the variables that were not selected:

```
[21]: # Get one accident record from MongoDB
accidents_collection.find_one()
```

```
[21]: {'_id': ObjectId('6729244fb4731f602a97daea'),
'id': 1,
'district_name': 'Sants-Montjuïc',
'neighborhood_name': 'la Marina del Prat Vermell',
'street_name': 'Litoral (Llobregat)',
'date': '2023-6-14',
'day_of_week': 'Dimecres',
'time_of_day': '18',
'cause': 'Gir indegit o sense precaució',
'coordinates': {'latitude': 41.343237595, 'longitude': 2.1309991}}
```

```
[22]: # Get one population record from MongoDB
population_collection.find_one()
```

```
[22]: {'_id': ObjectId('6729245cb4731f602a980ce4'),
       'id': 1,
       'district_name': 'Ciutat Vella',
       'neighborhood_name': 'el Raval',
       'Population': 606,
       'Sex': '1'}
```

```
[23]: # Get 5 accident entries from MongoDB and display as dataframe
r_accidents = accidents_collection.aggregate([
    {"$limit": 5},
])
pd.DataFrame(r_accidents)
```

```
[23]:          _id   id  district_name      neighborhood_name \
0  6729244fb4731f602a97daea  1  Sants-Montjuïc  la Marina del Prat Vermell
1  6729244fb4731f602a97daeb  2  Sants-Montjuïc  la Marina del Prat Vermell
2  6729244fb4731f602a97daec  3  Sants-Montjuïc  la Marina del Prat Vermell
3  6729244fb4731f602a97daed  4  Sants-Montjuïc  la Marina del Prat Vermell
4  6729244fb4731f602a97daee  5  Sants-Montjuïc  la Marina del Prat Vermell

          street_name      date day_of_week time_of_day \
0  Litoral (Llobregat)  2023-6-14     Dimecres        18
1  Litoral (Besòs)    2023-6-23     Divendres        14
2  Litoral (Besòs)    2023-2-8     Dimecres         7
3  Litoral (Besòs)    2023-1-4     Dimecres        19
4  Litoral (Besòs)  2023-10-25     Dimecres        17

          cause \
0  Gir indegut o sense precaució
1                  Altres
2  Canvi de carril sense precaució
3  No respectar distàncies
4  No respectar distàncies

          coordinates
0  {'latitude': 41.343237595, 'longitude': 2.1309... .
1  {'latitude': 41.351558031, 'longitude': 2.1534... .
2  {'latitude': 41.348267377, 'longitude': 2.1450... .
3  {'latitude': 41.347100121, 'longitude': 2.1417... .
4  {'latitude': 41.347100121, 'longitude': 2.1417... .
```

```
[24]: # Get 5 population entries from MongoDB and display as dataframe
r_population = population_collection.aggregate([
```

```

        {"$limit": 5},
    ])
pd.DataFrame(r_population)

```

[24]:

	_id	id	district_name	neighborhood_name	Population	\
0	6729245cb4731f602a980ce4	1	Ciutat Vella	el Raval	606	
1	6729245cb4731f602a980ce5	2	Ciutat Vella	el Raval	640	
2	6729245cb4731f602a980ce6	3	Ciutat Vella	el Raval	588	
3	6729245cb4731f602a980ce7	4	Ciutat Vella	el Raval	650	
4	6729245cb4731f602a980ce8	5	Ciutat Vella	el Raval	1555	

	Sex
0	1
1	2
2	1
3	2
4	1

3.4 Transform

Quality Assessment: accident data We first direct our attention towards the accidents collection by projecting its 5 first elements, to get a better understanding of the structure in table format.

[25]:

```
# Check document structure
r_accidents = accidents_collection.aggregate([
    {"$project": {"_id": 0}},
    {"$limit": 5},
])
pd.DataFrame(r_accidents)
```

[25]:

	id	district_name	neighborhood_name	street_name	\
0	1	Sants-Montjuïc	la Marina del Prat Vermell	Litoral (Llobregat)	
1	2	Sants-Montjuïc	la Marina del Prat Vermell	Litoral (Besòs)	
2	3	Sants-Montjuïc	la Marina del Prat Vermell	Litoral (Besòs)	
3	4	Sants-Montjuïc	la Marina del Prat Vermell	Litoral (Besòs)	
4	5	Sants-Montjuïc	la Marina del Prat Vermell	Litoral (Besòs)	

	date	day_of_week	time_of_day	cause	\
0	2023-6-14	Dimecres	18	Gir indegit o sense precaució	
1	2023-6-23	Divendres	14	Altres	
2	2023-2-8	Dimecres	7	Canvi de carril sense precaució	
3	2023-1-4	Dimecres	19	No respectar distàncies	
4	2023-10-25	Dimecres	17	No respectar distàncies	

	coordinates
--	-------------

```

0 {'latitude': 41.343237595, 'longitude': 2.1309... .
1 {'latitude': 41.351558031, 'longitude': 2.1534... .
2 {'latitude': 41.348267377, 'longitude': 2.1450... .
3 {'latitude': 41.347100121, 'longitude': 2.1417... .
4 {'latitude': 41.347100121, 'longitude': 2.1417... .

```

We now check for missing fields in the collection:

```
[26]: # Check for missing fields in accidents_collection
pipeline_accidents = [
    {"$match": {
        "$or": [
            {"district_name": {"$exists": False}},
            {"neighborhood_name": {"$exists": False}},
            {"street_name": {"$exists": False}},
            {"date": {"$exists": False}},
            {"day_of_week": {"$exists": False}},
            {"time_of_day": {"$exists": False}},
            {"cause": {"$exists": False}}
        ]
    }},
    {"$count": "missing_fields"}
]
```

We now execute the MongoDB aggregation pipeline on the collections accidents_collection. The aggregate() function runs the respective pipeline, which checks for missing fields in the documents. The result of each aggregation is converted into a list, allowing further inspection and manipulation of the aggregation results. Specifically, missing_accidents will store the results of any documents that match the conditions for missing fields in each collection.

```
[27]: missing_accidents = list(accidents_collection.aggregate(pipeline_accidents))

if missing_accidents:
    print("Missing fields in accidents_collection:",_
          →missing_accidents[0]['missing_fields'])
else:
    print("No missing fields in accidents_collection.)
```

No missing fields in accidents_collection.

The results show that the data does not have any missing entries, however the quality and level of completeness can be also addressed at the content level. We will now focus on the causes of accidents recorded in the collection, found in the variable cause, for which we will perform an aggregation pipeline which will show the count for each cause. For the sake of better interpretability the 15 unique string values will be translated from Catalan to English. This will involve firstly creating a translation dictionary, and then replacing all string values accordingly.

It must be noted that the variable cause contains a list predefined accident description strings for each record, meaning a given incident could be assigned a selection of the following causes.

```
[28]: # Dictionary to map Catalan 'causes' to English
causes_translation_dict = {
    "Manca d'atenció a la conducció": "Lack of attention while driving",
    "Altres": "Other",
    "No respectar distàncies": "Not respecting distances",
    "Gir indegit o sense precaució": "Improper or careless turn",
    "No determinada": "Undetermined",
    "Canvi de carril sense precaució": "Lane change without caution",
    "Desobeir semàfor": "Disobeying traffic lights",
    "Desobeir altres senyals": "Disobeying other signs",
    "Desconeugut": "Unknown",
    "Avançament defectuós/improcedent": "Improper/defective overtaking",
    "No respectat pas de vianants": "Not respecting pedestrian crossing",
    "Manca precaució incorporació circulació": "Lack of caution when merging",
    "Manca precaució efectuar marxa enrera": "Lack of caution when reversing",
    "No cedir la dreta": "Not yielding to the right",
    "Envair calçada contrària": "Crossing into oncoming traffic",
    "Fallada mecànica o avaria": "Mechanical failure or breakdown"
}
```

```
[29]: # Fetch the causes from MongoDB
r_causes = accidents_collection.aggregate([
    {"$group": {"_id": "$cause", "count": {"$sum": 1}}},
    {"$sort": {"count": -1}}
])

# Convert to DataFrame
df_causes = pd.DataFrame(r_causes)

# Replace the cause in Catalan with its English translation and rename '_id' to 'cause'
df_causes['_id'] = df_causes['_id'].map(causes_translation_dict)
df_causes.rename(columns={'_id': 'cause'}, inplace=True)

df_causes
```

	cause	count
0	Lack of attention while driving	2948
1	Other	1615
2	Not respecting distances	1416
3	Improper or careless turn	1151
4	Undetermined	1045
5	Lane change without caution	963
6	Disobeying traffic lights	922
7	Disobeying other signs	651
8	Unknown	550
9	Improper/defective overtaking	419

10	Not respecting pedestrian crossing	354
11	Lack of caution when merging	343
12	Lack of caution when reversing	256
13	Not yielding to the right	100
14	Crossing into oncoming traffic	38
15	Mechanical failure or breakdown	23

Continuing on from this result, we can map all the inconclusive causes of accident into a single ‘Cause specified’ category, and the rest into a ‘Cause not specified’ one. This will provide an insightful estimation of the quality of the data in terms of its content.

```
[30]: # Define the unspecified causes:
not_specified_causes = ['Other', 'Unknown', 'Undetermined']
# Create new column, and use lambda function to assign new value. Then group by new_column value.
df_causes['cause_category'] = df_causes['cause'].apply(lambda x: 'Cause not_specified' if x in not_specified_causes else 'Cause specified')
df_grouped = df_causes.groupby('cause_category')['count'].sum().reset_index()

df_grouped
```

```
[30]:      cause_category  count
0  Cause not specified  3210
1    Cause specified   9584
```

```
[31]: count_not_specified = df_grouped[df_grouped['cause_category'] == 'Cause not_specified']['count'].iloc[0]
count_specified = df_grouped[df_grouped['cause_category'] == 'Cause specified']['count'].iloc[0]

perc = count_not_specified * 100 / count_specified
print(f"{perc:.1f}% of the records do not specify the cause of the accident.")
```

33.5% of the records do not specify the cause of the accident.

The result shows that about one third of the causes for accidents are not specified, which shows lacking quality in the data in this respect.

Quality Assessment: population data We perform the same quality assessment with the population data stored under the population_collection. :

```
[32]: # Check document structure
r_population = population_collection.aggregate([
    {"$project": {"_id": 0}},
    {"$limit": 5},
])
pd.DataFrame(r_population)
```

```
[32]:   id district_name neighborhood_name  Population Sex
  0    1 Ciutat Vella          el Raval      606    1
  1    2 Ciutat Vella          el Raval     640    2
  2    3 Ciutat Vella          el Raval     588    1
  3    4 Ciutat Vella          el Raval     650    2
  4    5 Ciutat Vella          el Raval    1555    1
```

```
[33]: # Check for missing fields in population_collection
pipeline_population = [
    {"$match": {
        "$or": [
            {"district_name": {"$exists": False}},
            {"neighborhood_name": {"$exists": False}},
            {"Population": {"$exists": False}},
            {"Sex": {"$exists": False}}
        ]
    }},
    {"$count": "missing_fields"}
]

missing_population = list(population_collection.aggregate(pipeline_population))

if missing_population:
    print("Missing fields in population_collection:",_
          →missing_population[0]['missing_fields'])
else:
    print("No missing fields in population_collection.")
```

No missing fields in population_collection.

As is the case in the previous collection, there are no missing records for the population data set, indicating a good level of completeness across both sets from this source. There are however issues pertaining the structure which will have to be dealt with and modified for our specific use case, which are mainly due to the extraction method.

It becomes noticeable that for the same value of district_name and neighborhood_name there are several records of Population and Sex. This is due to the original data having an additional categorical variable for the censal code within each neighborhood, which for the sake of this project has been deemed unnecessary. In order to clean this structure, the data must be aggregated by summing the values of Population by neighborhood_name. Additionally, the Sex variable will be remapped as a string indicating female for a current value of 1 and male for 2, as is described in the [documentation](#) of the data set.

```
[34]: # Aggregation pipeline to group by neighborhood_name, district_name, and Sex while_
       →summing up Population
r_population = population_collection.aggregate([
    {
        "$group": {
            "_id": {
```

```

        "district_name": "$district_name",
        "neighborhood_name": "$neighborhood_name",
        "Sex": "$Sex"
    },
    "total_population": {"$sum": "$Population"}
}
},
{
    "$project": {
        "_id": 0, # Remove the default _id
        "district_name": "$_id.district_name",
        "neighborhood_name": "$_id.neighborhood_name",
        # Correct the mapping: 1 -> female, 2 -> male. Notice how the values are
→strings!
        "Sex": {
            "$cond": {
                "if": {"$eq": ["$_id.Sex", '1']},
                "then": "female",
                "else": "male"
            }
        },
        "total_population": 1
    }
},
{
    "$sort": {"district_name": 1, "neighborhood_name": 1, "Sex": 1} # Sort by
→district, neighborhood, and Sex
])

```

r_population_list = list(r_population)

Convert the result into a DataFrame for analysis

df_population = pd.DataFrame(r_population_list)

df_population

```
[34]:   total_population      district_name \
0          11173      Ciutat Vella
1          10868      Ciutat Vella
2          9276      Ciutat Vella
3         15184      Ciutat Vella
4         21629      Ciutat Vella
..          ...
141         2384  Sarrià-Sant Gervasi
142         16295  Sarrià-Sant Gervasi
143         13735  Sarrià-Sant Gervasi
144          8691  Sarrià-Sant Gervasi
145          7585  Sarrià-Sant Gervasi
```

```

neighborhood_name      Sex
0   Sant Pere, Santa Caterina i la Ribera female
1   Sant Pere, Santa Caterina i la Ribera    male
2               el Barri Gòtic    female
3               el Barri Gòtic    male
4               el Raval    female
...
141  Vallvidrera, el Tibidabo i les Planes    male
142          el Putxet i el Farró    female
143          el Putxet i el Farró    male
144        les Tres Torres    female
145        les Tres Torres    male

```

[146 rows x 4 columns]

It is important to note that in the above step, the cursor containing the aggregation pipeline has been saved to a list before being cast to a Pandas dataframe. This is due to cursors being iterators, and being consumed and reset after one action. Saving it as a list allows to preserve it for visualizing it as a dataframe, as well as reloading it to the collection, which is the next step. We now proceed to purge the collection and repopulate it with the transformed data:

```

[35]: # Purge the existing collection
population_collection.delete_many({}) # This deletes all documents in the collection

# Insert the new aggregated data into the collection
population_collection.insert_many(r_population_list) # Use the list for insertion
print("Population collection successfully purged and repopulated with the new_"
→aggregated data!")

```

Population collection successfully purged and repopulated with the new aggregated data!

If a cursor is converted to a list and then cast to a Pandas dataframe object afterwards, a more readable table representation shows the data has been stored correctly, and contains separate records for female and male residents of each neighborhood.

```

[36]: listed_head = list(population_collection.find().limit(2))
listed_head

```

```

[36]: [ {_id': ObjectId('67292466b4731f602a98153c'),
  'total_population': 11173,
  'district_name': 'Ciutat Vella',
  'neighborhood_name': 'Sant Pere, Santa Caterina i la Ribera',
  'Sex': 'female'},
  {_id': ObjectId('67292466b4731f602a98153d'),
  'total_population': 10868,
  'district_name': 'Ciutat Vella',
  'neighborhood_name': 'Sant Pere, Santa Caterina i la Ribera',
  'Sex': 'male'}]

```

```
[37]: pd.DataFrame(listed_head)
```

```
[37]:          _id  total_population district_name \
0  67292466b4731f602a98153c           11173  Ciutat Vella
1  67292466b4731f602a98153d           10868  Ciutat Vella

               neighborhood_name      Sex
0  Sant Pere, Santa Caterina i la Ribera  female
1  Sant Pere, Santa Caterina i la Ribera    male
```

Managing location data as GeoJSON In this transformation step, the focus will be shifted to preparing accident location data in a format compatible with MongoDB's geospatial capabilities. Each record in the accidents_collection initially contains coordinates as separate latitude and longitude values. By using MongoDB's update_many function, the transformation adds a new location field to each document, structuring it as a GeoJSON Point object with coordinates. This format is essential for geospatial queries in MongoDB, as it follows the GeoJSON standard, which is widely supported for handling location-based data in NoSQL databases. This transformation allows MongoDB to recognize the location field as spatial data, enabling for some advanced spatial operations.

```
[38]: # Update all documents to include location in GeoJSON format
accidents_collection.update_many(
    {},
    [
        {
            "$set": {
                "location": {
                    "type": "Point",
                    "coordinates": ["$coordinates.longitude", "$coordinates.latitude"]
                }
            }
        }
    ]
)
```

```
[38]: UpdateResult({'n': 12794, 'electionId': ObjectId('7fffffff0000000000002d6'),
  'opTime': {'ts': Timestamp(1730749554, 225), 't': 726}, 'nModified': 12794,
  'ok': 1.0, '$clusterTime': {'clusterTime': Timestamp(1730749554, 232),
  'signature': {'hash':
b'\x80\x6\xfb\xde\x025:$\xc8\xcf\xf83e^v\x06\xb3\x18\x4', 'keyId':
7393024685182550042}}, 'operationTime': Timestamp(1730749554, 225),
  'updatedExisting': True}, acknowledged=True)
```

Following the transformation of the data format, a 2dsphere index is created on the location field to support efficient geospatial queries. The 2dsphere index enables MongoDB to perform complex geospatial calculations, such as finding points within a specific distance or locating points within a defined polygon. By preparing the data in GeoJSON format and indexing it, this transformation step ensures that subsequent geospatial analysis and filtering operations on traffic accident data

in Barcelona are both feasible and efficient.

```
[39]: # Create 2dsphere index  
accidents_collection.create_index([("location", GEOSPHERE)])
```

```
[39]: 'location_2dsphere'
```

To avoid redundancy of data, we proceed to remove the original coordinates item, in favor of the new GeoJSON standardized location:

```
[40]: # Remove the 'coordinates' field from all documents in the collection  
accidents_collection.update_many({}, {"$unset": {"coordinates": ""}})
```

```
[40]: UpdateResult({'n': 12794, 'electionId': ObjectId('7fffffff00000000000002d6'),  
'opTime': {'ts': Timestamp(1730749566, 963), 't': 726}, 'nModified': 12794,  
'ok': 1.0, '$clusterTime': {'clusterTime': Timestamp(1730749566, 968),  
'signature': {'hash':  
b'\xb9d\xaa\x1a\xb1X\x13\x1a\xd6Q7\x9f\xdemd\xf3\xfe\x12\x89XF', 'keyId':  
7393024685182550042}}, 'operationTime': Timestamp(1730749566, 963),  
'updatedExisting': True}, acknowledged=True)
```

```
[41]: # Get one accident record from MongoDB  
accidents_collection.find_one()
```

```
[41]: {'_id': ObjectId('6729244fb4731f602a97daea'),  
'id': 1,  
'district_name': 'Sants-Montjuïc',  
'neighborhood_name': 'la Marina del Prat Vermell',  
'street_name': 'Litoral (Llobregat)',  
'date': '2023-6-14',  
'day_of_week': 'Dimecres',  
'time_of_day': '18',  
'cause': 'Gir indegit o sense precaució',  
'location': {'type': 'Point', 'coordinates': [2.1309991, 41.343237595]}}
```

3.5 Datastructure

The structure of the data stored on the MongoDB collections after the transformations is as follows:

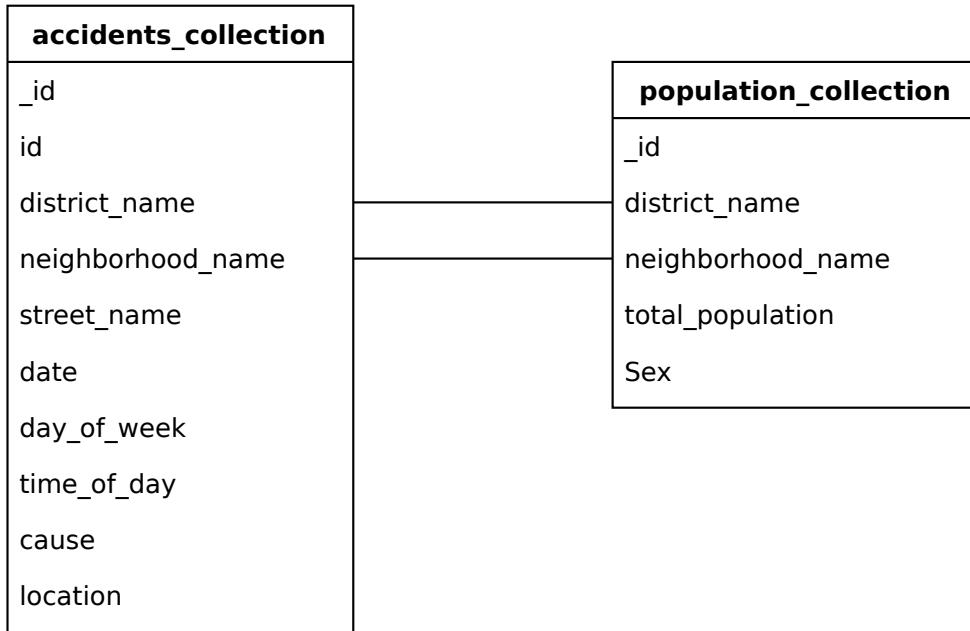
Traffic Accidents Collection This collection stores detailed information about each accident, including location (formatted as GeoJSON for geospatial querying), district and neighborhood names, street name, date, time, and cause of the accident.

Field	Data Type	Description
_id	ObjectId	Unique identifier generated by MongoDB
id	Integer	ID from the original dataset
district_name	String	Name of the district where the accident occurred
neighborhood_name	String	Name of the neighborhood
street_name	String	Name of the street
date	String (ISO format)	Date of the accident
day_of_week	String	Day of the week when the accident happened
time_of_day	Integer	Time of day recorded as an integer (24-hour format)
cause	String	Cause of the accident
location	GeoJSON (Point)	GeoJSON object containing type (Point) and coordinates (longitude, latitude)

Population Data Collection This collection houses population statistics per neighborhood, broken down by gender, which can be used to analyze demographics in relation to accident data. It also includes a column to the parent district.

Field	Data Type	Description
_id	ObjectId	Unique identifier generated by MongoDB
district_name	String	Name of the district
neighborhood_name	String	Name of the neighborhood
total_population	Integer	Total population of the neighborhood
Sex	Integer	Gender identifier (1 for female and 2 for male)

Although the original datasets, sourced from the same provider, reference a neighborhood ID encoded as a numerical value, the high level of data cleanliness and consistency between these datasets has led to the decision to use district_name and neighborhood_name as mapping variables or keys for the following aggregations. This decision is also due to these additional variables containing essentially the same information, and therefore having been discarded at the API extraction phase to avoid unnecessary redundancy and storage size. While it's generally understood that mapping on string variables is not ideal, given these specific circumstances, this approach has been deemed acceptable for this use case.



4 Data analysis

This section presents a series of data analysis procedures conducted using MongoDB aggregations, focusing on five different research questions. These analyses will be complemented by visualizations created with Pandas dataframes and Matplotlib and Seaborn plots, as well as some interactive map charts.

4.1 Where in the city are accidents recorded?

To start our data analysis, we will begin with a geographical interpretation of the dataset. This process begins with defining a MongoDB query for the accidents collection, which retrieves location information such as district, neighborhood, and street names, along with coordinates stored as a nested structure within MongoDB records in the standardized WGS84 format. Additionally, some descriptive data will be included. The query is as follows:

```
[42]: accidents_query_01 = [
    {
        "$group": {
            "_id": {
                "district_name": "$district_name",
                "neighborhood_name": "$neighborhood_name",
                "street_name": "$street_name",
                "date": "$date" # Include date in the group
            },
            "num_accidents": {"$sum": 1}, # Count the number of accidents per ↴neighborhood and street
        }
    }
]
```

```

        "causes": {"$push": "$cause"}, # Gather all causes in an array
        "location": {"$first": "$location"} # Keep the first occurrence of
    →location
    }
},
{
    "$project": {
        "_id": 0,
        "district_name": "$_id.district_name",
        "neighborhood_name": "$_id.neighborhood_name",
        "street_name": "$_id.street_name",
        "date": "$_id.date", # Project the date field
        "num_accidents": 1,
        "causes": 1,
        "location": 1 # Use location instead of coordinates
    }
}
]

```

[43]: # Accident aggregation query
accidents_agg_01 = accidents_collection.aggregate(accidents_query_01)
accidents_df = pd.DataFrame(list(accidents_agg_01))

As a disclaimer, for the current and the following research questions, all procedures beyond querying and aggregation in MongoDB will be conducted using the versatility and visualization capabilities of Pandas dataframes, along with selected visualization libraries.

Casting the aggregated results into a Pandas dataframe shows some transformations that must be undertaken, such as unpacking the coordinates into separate columns and translating the causes into English with the previously defined dictionary.

[44]: # 1. Translate causes
accidents_df['translated_causes'] = accidents_df['causes'].apply(lambda x:
→[causes_translation_dict.get(cause, cause) for cause in x])

2. Unpack location coordinates into separate latitude and longitude columns,
→preserving all decimals
accidents_df['longitude'] = accidents_df['location'].apply(lambda x:
→float(x['coordinates'][0]))
accidents_df['latitude'] = accidents_df['location'].apply(lambda x:
→float(x['coordinates'][1]))

3. Remove the original location and causes columns
accidents_df.drop(columns=['location', 'causes'], inplace=True)

Set pandas to display more decimal places, to visualize coordinates accurately
pd.set_option('display.float_format', '{:.10f}'.format) # Adjust number of decimal
→places as needed

```
# Convert lists in 'translated_causes' to strings
accidents_df['translated_causes'] = accidents_df['translated_causes'].apply(lambda x: ', '.join(x))

# Display the updated DataFrame
accidents_df.head()
```

```
[44]:   num_accidents      district_name      neighborhood_name \
0            1             Eixample    l'Antiga Esquerra de l'Eixample
1            1           Ciutat Vella                  el Raval
2            2        Horta-Guinardó          el Baix Guinardó
3            2  Sarrià-Sant Gervasi  Sant Gervasi - Galvany
4            1           Sant Andreu          el Bon Pastor

      street_name      date      translated_causes \
0       Londres  2023-10-28  Improper/defective overtaking
1  Sant Rafael  2023-2-15  Lack of attention while driving
2     Lepant    2023-4-4  Lane change without caution, Lane change witho...
3  Marià Cubí  2023-9-17  Lack of attention while driving, Lack of atten...
4  Monterrey  2023-6-28                Undetermined

      longitude      latitude
0  2.1471485750  41.3907448610
1  2.1694813860  41.3791276050
2  2.1690034560  41.4103077130
3  2.1454417640  41.3956951640
4  2.2060131730  41.4434421740
```

The first visualization in the series is a scatter map displaying all accident records across the city, providing an initial assessment of the dataset. Two colors are used to differentiate records with specified causes from those with undefined causes, offering a quick view of data quality, as discussed in a previous chapter of the project.

```
[45]: # Set the default renderer for JupyterLab
pio.renderers.default = 'iframe'

# Define the unspecified causes
not_specified_causes = ['Other', 'Unknown', 'Undetermined']

# Create a new column for cause category
accidents_df['cause_category'] = accidents_df['translated_causes'].apply(
    lambda x: 'Cause not specified' if x in not_specified_causes else 'Cause specified'
)

# Create the scatter map with cause categories
```

```

fig1 = px.scatter_mapbox(
    accidents_df,
    lat='latitude',
    lon='longitude',
    color='cause_category', # Color by specified/unspecified cause
    hover_name='date', # Show date as the main hover label
    hover_data={
        'neighborhood_name': True, # Show neighborhood name on hover
        'street_name': True, # Include street name in the hover
        'cause_category': True, # Include cause category in the hover
        'latitude': False,
        'longitude': False
    },
    title='Accidents in Barcelona in 2023',
    mapbox_style='open-street-map', # Using OpenStreetMap tiles
    zoom=12, # Adjust zoom level as necessary
    height=800, # Set height to 800 pixels
)

# Update hover text to format it correctly
fig1.update_traces(
    hovertemplate=
        "<b>Date:</b> %{hovertext}<br>" + # Show date
        "<b>Neighborhood:</b> %{customdata[0]}<br>" + # Neighborhood name
        "<b>Street:</b> %{customdata[1]}<br>" + # Street name
        "<b>Cause Category:</b> %{customdata[2]}<br>" + # Cause category
        "<extra></extra>", # Removes the default hover info
)

# Set custom data for the hover template
fig1.data[0].customdata = accidents_df[['neighborhood_name', 'street_name', ...'cause_category']].values

# Change the legend title for the color scale
fig1.for_each_trace(lambda t: t.update(name=t.name.split('=')[1].strip() if '=' in ...t.name else t.name))

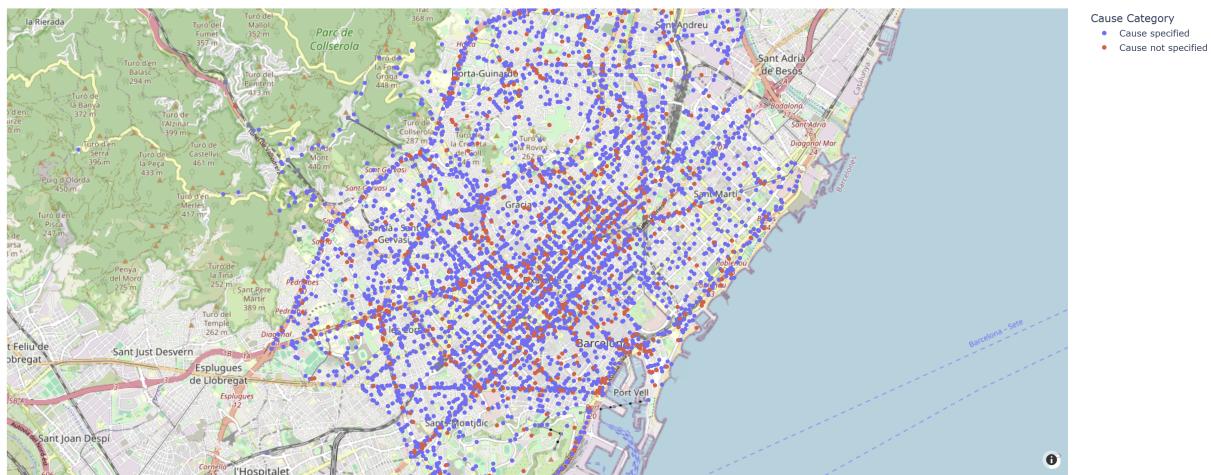
# Manually set the color legend title
fig1.update_layout(
    legend_title_text="Cause Category"
)

# Show the map
fig1.show()

# Save the Plotly figure as a standalone HTML file
fig1.write_html("images/fig1_accidents.html", full_html=True)

```

Accidents in Barcelona in 2023



The results reveal a predictable concentration of accident records along the main arteries of the city, with a marked increase in density within the city center and along the main traffic rings. There appears to be no discernible pattern among non-specific accident records, except that most occur within the perimeter of the traffic rings.

Next, we will follow up wby visualizing the data as a density map.

```
[46]: # Step 2: Create the density map based on individual accident locations
density_fig = px.density_mapbox(
    accidents_df,
    lat='latitude',
    lon='longitude',
    radius=6, # Adjust radius for visual clarity and influence size
    mapbox_style='open-street-map',
    zoom=12,
    height=800,
    title='Density Map of Accidents in Barcelona by Location in 2023'
)

# Update hover text to show street and neighborhood details
density_fig.update_traces(
    hovertemplate=
    "<b>Street:</b> %{customdata[0]}<br>" + # Show street name
    "<b>Neighborhood:</b> %{customdata[1]}<br>" + # Neighborhood name
    "<extra></extra>", # Removes the default hover info
)

# Set custom data for the hover template
density_fig.data[0].customdata = accidents_df[['street_name', 'neighborhood_name']].values
```

```

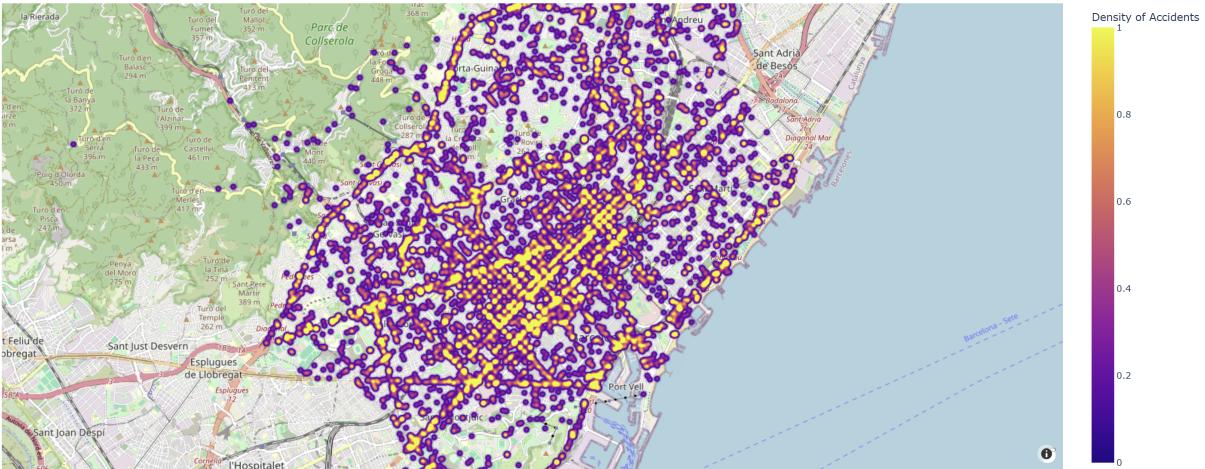
# Change the legend title for the color scale
density_fig.update_layout(
    coloraxis_colorbar=dict(
        title="Density of Accidents" # Updated legend title
    )
)

# Show the density map
density_fig.show()

# Save the Plotly figure as a standalone HTML file
density_fig.write_html("images/fig2_accidents.html", full_html=True)

```

Density Map of Accidents in Barcelona by Location in 2023



This visualization clearly highlights the main arteries of Barcelona, with Avinguda Diagonal and Gran Via de les Corts Catalanes prominently standing out, along with the surrounding ring roads. The city center, where these major axes intersect, shows a high concentration of accidents, as expected in densely trafficked areas.

One particularly interesting observation is the concentration of accidents in the Eixample district, visible as bright yellow zones on the map. These hotspots correspond primarily to intersections between streets within Eixample, a district renowned for its 19th-century expansion, which extended beyond the old city in a meticulously planned orthogonal grid. This design, known as Eixample or Ensanche, features a high number of straight, cross-street intersections, a layout that has been shown to increase the likelihood of traffic collisions across many urban settings around the world.

The density map thus not only reflects the flow of traffic along Barcelona's main routes but also emphasizes the risks inherent in Eixample's urban design. This grid pattern, while iconic, contributes to the high accident density at intersections, making it a crucial area of focus for traffic safety improvements.

How many accidents have been recorded in the vicinity of a touristic landmark, the Sagrada Família? This subquestion will be tackled by leveraging the enhanced functionalities of the Geo-

JSON format, to query results based on geographical variables. The landmark of choice will be the Sagrada Família Basilica, one of the most celebrated architectural works of the city. The query will be based on the coordinates of its location, which have been sourced from its [Wikipedia article](#). The following query will make use of the location item stored within each record, and with the \$near clause and the point coordinates for the location, as well as an indicated radius of 100 meters, we should obtain as a result all accident records found within that designated area.

```
[47]: radius = 250 # meters

nearby_accidents = accidents_collection.find(
    {
        "location": {
            "$near": {
                "$geometry": {"type": "Point", "coordinates": [2.17433, 41.40369]}, ←
                ←# Coordinates of Barcelona center
                "$maxDistance": radius
            }
        }
    }
)

nearby_accidents_list= list(nearby_accidents)
print(f"There are {len(nearby_accidents_list)} accidents recorded within {radius}←
    ←meters of the Sagrada Família Basilica in 2023.")
```

There are 82 accidents recorded within 250 meters of the Sagrada Família Basilica in 2023.

```
[48]: # Convert the list of accident records to a DataFrame
accident_df = pd.DataFrame(nearby_accidents_list)

# Separate latitude and longitude from the location field
accident_df['longitude'] = accident_df['location'].apply(lambda x:←
    ←x['coordinates'][0] if x else None)
accident_df['latitude'] = accident_df['location'].apply(lambda x:←
    ←x['coordinates'][1] if x else None)

accident_df.drop(columns=['location'], inplace=True)
```

To improve the map representation of the queried accidents, a small jitter factor has been added to account for the overlap of multiple accident points. This overlap is likely due to the limited precision of the original recorded coordinates. Although adding jitter slightly reduces the positional accuracy, it has been deemed appropriate to create a more visually informative and comprehensible graphic.

```
[49]: # Add a small jitter to latitude and longitude
jitter_amount = 0.0001
accident_df['longitude'] += np.random.normal(0, jitter_amount, ←
    ←size=len(accident_df))
```

```

accident_df['latitude'] += np.random.normal(0, jitter_amount, size=len(accident_df))

# Create the scatter map with additional hover data (street and date)
fig3 = px.scatter_mapbox(
    accident_df,
    lat='latitude',
    lon='longitude',
    hover_name='neighborhood_name', # Show neighborhood as the main hover label
    title=f'Accidents within {radius} meters of Sagrada Família Basilica in 2023',
    mapbox_style='open-street-map', # Use OpenStreetMap tiles
    zoom=16, # Adjust zoom level for a close-up view
    height=800 # Set the height of the plot
)

# Set custom data for hover information
fig3.data[0].customdata = accident_df[['street_name', 'date']].values

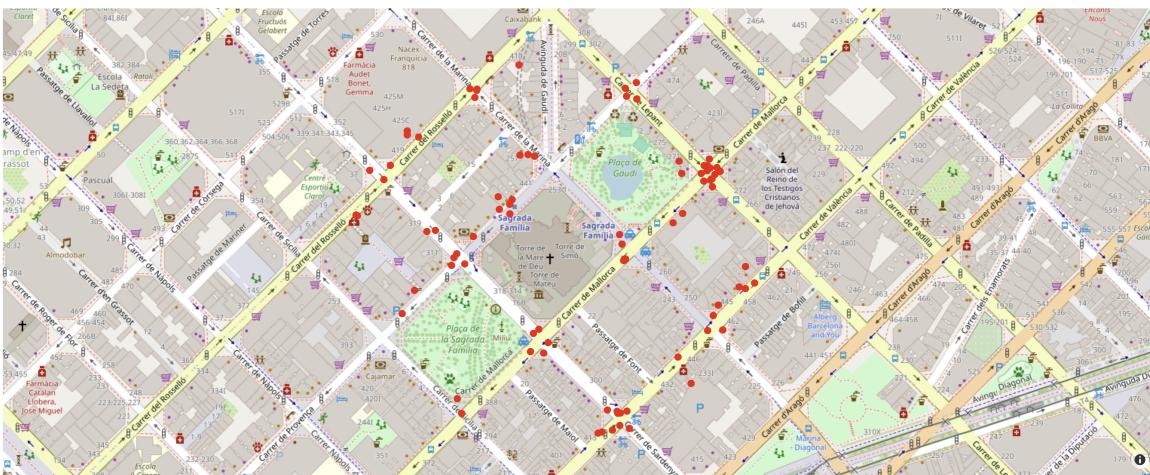
# Customize hover text to include street, neighborhood, and date
fig3.update_traces(
    marker=dict(size=10, color="red"), # Set size and color of the markers
    hovertemplate=
        "<b>Neighborhood:</b> %{hovertext}<br>" + # Neighborhood name
        "<b>Street:</b> %{customdata[0]}<br>" + # Street name
        "<b>Date:</b> %{customdata[1]}<br>" + # Date
        "<b>Latitude:</b> %{lat:.6f}<br>" + # Latitude with 6 decimal places
        "<b>Longitude:</b> %{lon:.6f}<br>" + # Longitude with 6 decimal places
        "<extra></extra>" # Removes the default hover info
)

# Show the map
fig3.show()

# Save the Plotly figure as a standalone HTML file
fig3.write_html("images/fig3_accidents.html", full_html=True)

```

Accidents within 250 meters of Sagrada Família Basilica in 2023



4.2 Do more accidents happen in more populous neighborhoods?

A straightforward question, which will rely on querying and integrating data from both collections, we now aim to observe potential trends in the number of accidents per neighborhood and the official population counts. For this purpose, we will have to extract the relevant variables `district_name`, `neighborhood_name`, while aggregating the counts of accident records into a new `num_accidents` column, and aggregate the populations from both sexes into one single `total_population`. Following is the construction of the query pipelines:

```
[50]: accidents_query_02 = [
    {
        "$group": {
            "_id": {
                "district_name": "$district_name",
                "neighborhood_name": "$neighborhood_name"
            },
            "num_accidents": {"$sum": 1} # Count the number of accidents per ↴neighborhood
        }
    },
    {
        "$project": {
            "_id": 0,
            "district_name": "$_id.district_name",
            "neighborhood_name": "$_id.neighborhood_name",
            "num_accidents": 1
        }
    }
]
```

```

population_query_02 = [
    {
        '$group': {
            '_id': '$neighborhood_name',
            'total_population': {'$sum': '$total_population'}
        }
    },
    {
        '$project': {
            'neighborhood_name': '_id',
            'total_population': 1,
            '_id': 0
        }
    }
]

```

```

[51]: # Accident aggregation query
accidents_agg = accidents_collection.aggregate(accidents_query_02)
accidents_df = pd.DataFrame(list(accidents_agg))

# Population aggregation query
population_agg = population_collection.aggregate(population_query_02)
population_df = pd.DataFrame(list(population_agg))

```

We now merge both Pandas dataframes built from the queried data for analysis and visualization.

```

[52]: # Merge accidents data with population data
merged_data = pd.merge(accidents_df, population_df, on=['neighborhood_name'],
                       how='inner')
merged_data.head(5)

```

	num_accidents	district_name	neighborhood_name	total_population
0	164	Sants-Montjuïc	Hostafrancs	15969
1	98	Sants-Montjuïc	Sants - Badal	24366
2	125	Sant Andreu	la Trinitat Vella	10286
3	118	Sant Andreu	la Sagrera	29908
4	297	Sants-Montjuïc	la Marina del Prat Vermell	1678

Additionally, we can compute a new accidents_per_100_inhabitants directly from the DataFrame, to obtain a more comprehensive interpretation of both values.

```

[53]: # Calculate accidents per 100 inhabitants
merged_data["accidents_per_100_inhabitants"] = (merged_data["num_accidents"] /_
                                                 merged_data["total_population"]) * 100
merged_data.head()

```

	num_accidents	district_name	neighborhood_name
0	164	Sants-Montjuïc	Hostafrancs

```

1          98  Sants-Montjuïc           Sants - Badal
2         125  Sant Andreu           la Trinitat Vella
3         118  Sant Andreu           la Sagrera
4        297  Sants-Montjuïc  la Marina del Prat Vermell

  total_population  accidents_per_100_inhabitants
0            15969                1.0269897927
1            24366                0.4021997866
2            10286                1.2152440210
3            29908                0.3945432660
4            1678                 17.6996424315

```

```
[54]: # Figure size
plt.figure(figsize=(12, 8))

# Create a scatterplot with enhanced transparency and smaller points
sns.scatterplot(
    data=merged_data,
    x='total_population',
    y='num_accidents',
    hue='district_name', # Color by 'district_name'
    alpha=0.6, # Increased transparency
    s=70, # Reduced point size
    edgecolor='black' # Edge color to make points stand out
)

# Add a lighter regression line using regplot
sns.regplot(
    x='total_population',
    y='num_accidents',
    data=merged_data,
    scatter=False, # Do not overlay scatter points again
    line_kws={'color': 'red', 'alpha': 0.3}, # Lighter regression line
    ci=None # Remove confidence interval shading for decluttering
)

# Add text labels to every nth point
n = 4 # Set the spacing of the labels
for i in range(0, len(merged_data), n):
    plt.text(
        merged_data['total_population'].iloc[i],
        merged_data['num_accidents'].iloc[i],
        merged_data['neighborhood_name'].iloc[i],
        fontsize=9,
        ha='right',
        va='bottom'
    )

```

```

# Set title and axis labels
plt.title('Scatterplot of Total Population vs. Number of Accidents', fontsize=16)
plt.xlabel('Total Population', fontsize=14)
plt.ylabel('Number of Accidents', fontsize=14)

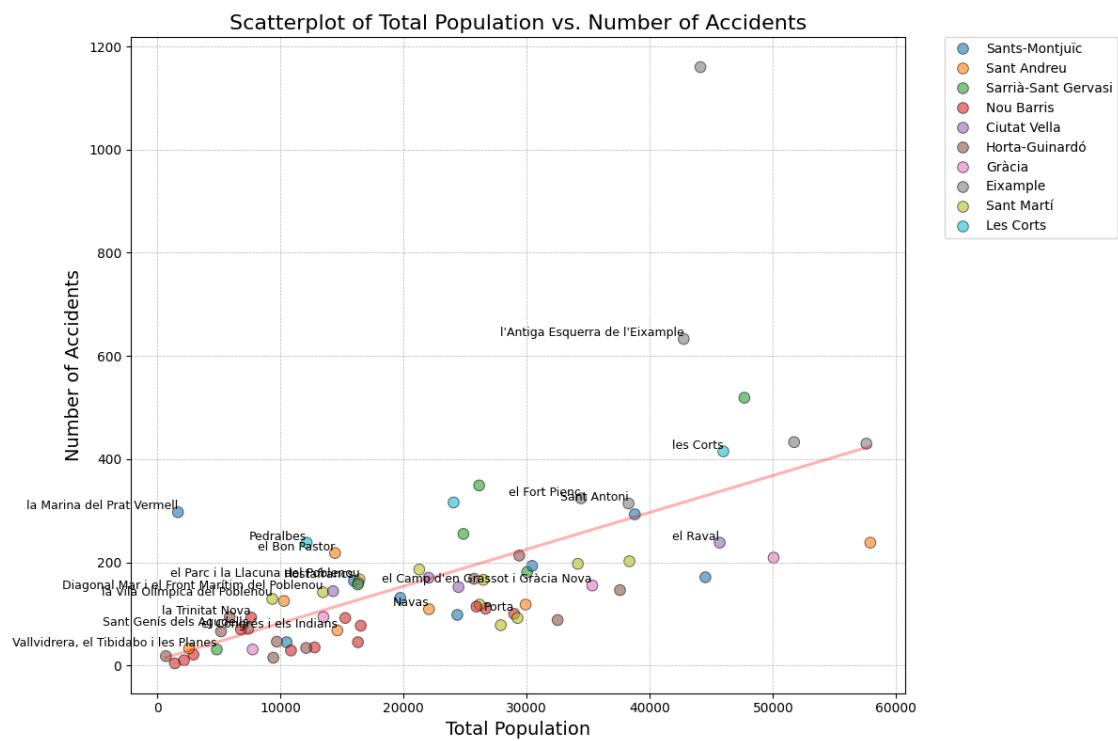
# Add a grid for better readability
plt.grid(True, which="both", linestyle="--", linewidth=0.5) # Add gridlines for both major and minor ticks

# Move the legend outside the plot to reduce clutter
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0)

# Adjust layout for readability
plt.tight_layout()

# Show the plot
plt.show()

```



The above plot shows each neighborhood's recorded accident count in relation to its population. The color of each point has been assigned to its district, which represents a higher administrative category than neighborhood. What becomes apparent is the linear trend, which supports the hypothesis that higher populations per neighborhood do in fact correlate with higher number of accidents. Based on this analysis, neighborhoods located in the Eixample district of Barcelona,

which roughly covers most of the city center and busiest streets, holds both some of the highest population counts as well as recorded street traffic incidents.

However, if we consider the relative number of accidents per inhabitant count, the results become more comprehensive, as the above plot doesn't take into account factors such as neighborhood size or population density.

```
[55]: # Sort by accidents_per_100_inhabitants in descending order and display the head
sorted_data = merged_data.sort_values(by="accidents_per_100_inhabitants",_
→ascending=False).head()
sorted_data
```

```
[55]:    num_accidents  district_name      neighborhood_name \
4            297  Sants-Montjuïc  la Marina del Prat Vermell
34           1160        Eixample  la Dreta de l'Eixample
27            18  Horta-Guinardó      la Clota
68            238        Les Corts      Pedralbes
62            94  Horta-Guinardó  la Vall d'Hebron

total_population  accidents_per_100_inhabitants
4              1678          17.6996424315
34             44105          2.6300872917
27              699          2.5751072961
68             12130          1.9620774938
62             5874          1.6002723868
```

Sorting the results by the top 5 highest values for accidents_per_100_inhabitants reveals that global count of accidents isn't the only comprehensive assessment. For example, la Marina del Prat Vermell, a relatively small yet centric neighborhood of the city, concentrates a staggering 17.70 accidents per 100 inhabitants, followed by la Dreta de l'Eixample's 2.63 value. The latter also tops the chart for global accident count.

4.3 What are the temporal patterns of the recorded street accidents?

For this exploration chapter of the data stored on the MongoDB database, we will now focus exclusively on the traffic accidents collection, and extract insights pertaining the temporal data. As with the previous research questions, the first step will be to build the corresponding MongoDB aggregation query, without the need of having to later merge it with the data from the other collection.

```
[56]: accidents_query_03 = [
  {
    "$project": {
      "_id": 0,
      "year": {"$year": {"$dateFromString": {"dateString": "$date"}},  #_
→Extract year
      "month": {"$month": {"$dateFromString": {"dateString": "$date"}},  #_
→Extract month
    }
  }
]
```

```

        "day_of_month": {"$dayOfMonth": {"$dateFromString": {"dateString": .._  
→ "$date"}}, # Extract day of the month  
        "day_of_week": "$day_of_week", # Use existing day of week field  
        "time_of_day": "$time_of_day", # Time of day from the field  
        "cause": 1, # Include cause for reference  
        "neighborhood_name": 1, # Include neighborhood for reference  
        "district_name": 1, # Include district for reference  
    }  
}  
]

```

For the sake of better interpretation, we again create a small dictionary to translate the days of the week from Catalan to English. We will also refer to the translated accident causes dictionary from earlier in the project, in order to produce a more complete assessment.

[57]:

```
# Translation dictionary for days of the week (Catalan to English)  
days_translation_dict = {  
    'Dilluns': 'Monday',  
    'Dimarts': 'Tuesday',  
    'Dimecres': 'Wednesday',  
    'Dijous': 'Thursday',  
    'Divendres': 'Friday',  
    'Dissabte': 'Saturday',  
    'Diumenge': 'Sunday'  
}
```

[58]:

```
# Accident aggregation query  
accidents_temporal = accidents_collection.aggregate(accidents_query_03)  
accidents_temporal_df = pd.DataFrame(list(accidents_temporal))  
  
# Apply the translations to the DataFrame  
accidents_temporal_df['day_of_week'] = accidents_temporal_df['day_of_week'].  
    →map(days_translation_dict)  
accidents_temporal_df['cause'] = accidents_temporal_df['cause'].  
    →map(causes_translation_dict)  
  
# Display the DataFrame  
accidents_temporal_df.head()
```

[58]:

	district_name	neighborhood_name	cause \
0	Sants-Montjuïc	la Marina del Prat Vermell	Improper or careless turn
1	Sants-Montjuïc	la Marina del Prat Vermell	Other
2	Sants-Montjuïc	la Marina del Prat Vermell	Lane change without caution
3	Sants-Montjuïc	la Marina del Prat Vermell	Not respecting distances
4	Sants-Montjuïc	la Marina del Prat Vermell	Not respecting distances

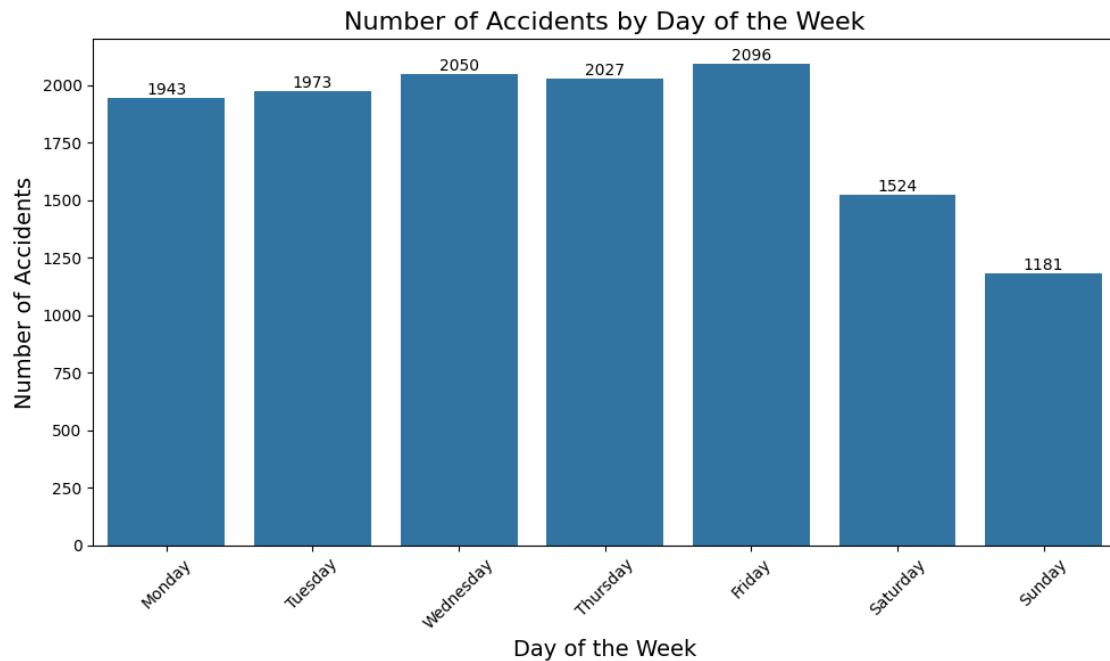
year month day_of_month day_of_week time_of_day

0	2023	6	14	Wednesday	18
1	2023	6	23	Friday	14
2	2023	2	8	Wednesday	7
3	2023	1	4	Wednesday	19
4	2023	10	25	Wednesday	17

```
[59]: # Bar plot: Number of accidents by day of the week
plt.figure(figsize=(10, 6))
ax = sns.countplot(data=accidents_temporal_df, x='day_of_week', order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

# Add count labels on top of the bars
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=10)

plt.title('Number of Accidents by Day of the Week', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Accidents', fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



The first visualization we produce corresponds to the overall accident count sorted by day of the week. At a first glance it becomes apparent that throughout the week, an escalation in accident

reports takes place, followed by a sudden sinking on the weekend, with Sundays having the lowest accident counts. This would be explained by a likely reduced level of traffic during the weekend, as well as less street police activity.

```
[60]: # Rename 'day_of_month' to 'day'
accidents_temporal_df.rename(columns={'day_of_month': 'day'}, inplace=True)

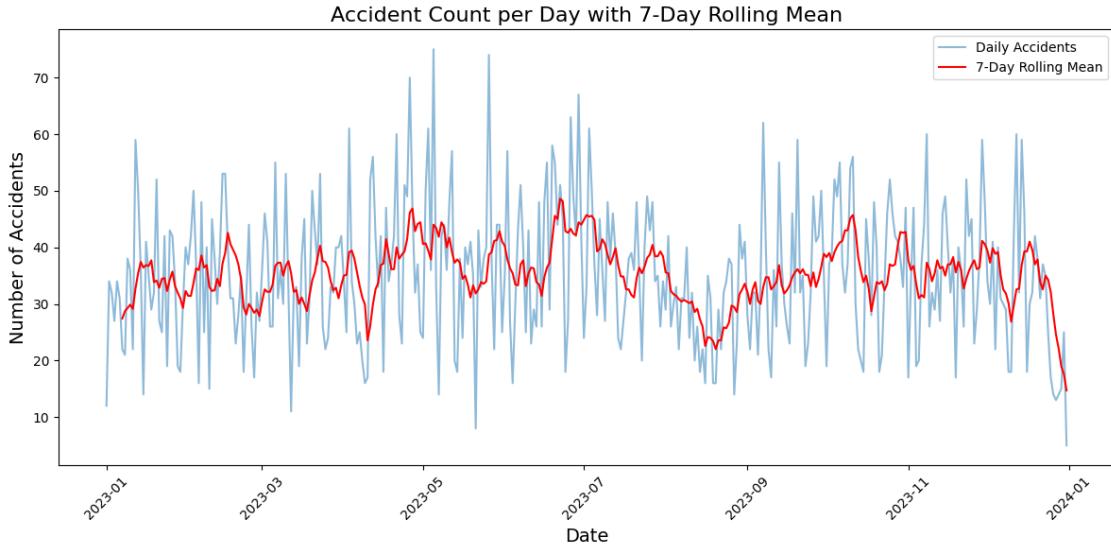
# Create a 'date' column by combining year, month, and day
accidents_temporal_df['date'] = pd.to_datetime(accidents_temporal_df[['year',_
→'month', 'day']])

# Group by date to count the number of accidents per day
accidents_per_day = accidents_temporal_df.groupby('date').size().
→reset_index(name='accident_count')

# Calculate a 7-day rolling mean for accidents per day
accidents_per_day['rolling_mean'] = accidents_per_day['accident_count'].
→rolling(window=7).mean()

# Plot the time series of accidents per day with rolling mean
plt.figure(figsize=(12, 6))
sns.lineplot(data=accidents_per_day, x='date', y='accident_count', label='Daily_-
→Accidents', alpha=0.5)
sns.lineplot(data=accidents_per_day, x='date', y='rolling_mean', label='7-Day_-
→Rolling Mean', color='red')

plt.title('Accident Count per Day with 7-Day Rolling Mean', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Number of Accidents', fontsize=14)
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```



The plot above illustrates a time series of daily accident counts in Barcelona. To clarify the underlying trends, a 7-day rolling mean (in red) is included. The blue line indicates daily accident counts, which display significant spikes and dips, reflecting the city's dynamic traffic conditions. The red line smooths out these fluctuations, providing a clearer understanding of long-term trends in traffic safety. This analysis is vital for city planners and public safety officials working to enhance road safety and traffic management in Barcelona.

```
[61]: # Create a dictionary to map month numbers to month names
month_mapping = {
    1: 'January', 2: 'February', 3: 'March', 4: 'April',
    5: 'May', 6: 'June', 7: 'July', 8: 'August',
    9: 'September', 10: 'October', 11: 'November', 12: 'December'
}

# Check if the 'month' column contains numeric values
if pd.api.types.is_numeric_dtype(accidents_temporal_df['month']):
    # Ensure the 'month' column is numeric and handle any non-integer or missing values
    accidents_temporal_df['month'] = pd.to_numeric(accidents_temporal_df['month'],
                                                   errors='coerce')

    # Replace month numbers with month names in the DataFrame
    accidents_temporal_df['month'] = accidents_temporal_df['month'].map(month_mapping)

# Check for any NaN values after the mapping
accidents_temporal_df = accidents_temporal_df.dropna(subset=['month'])

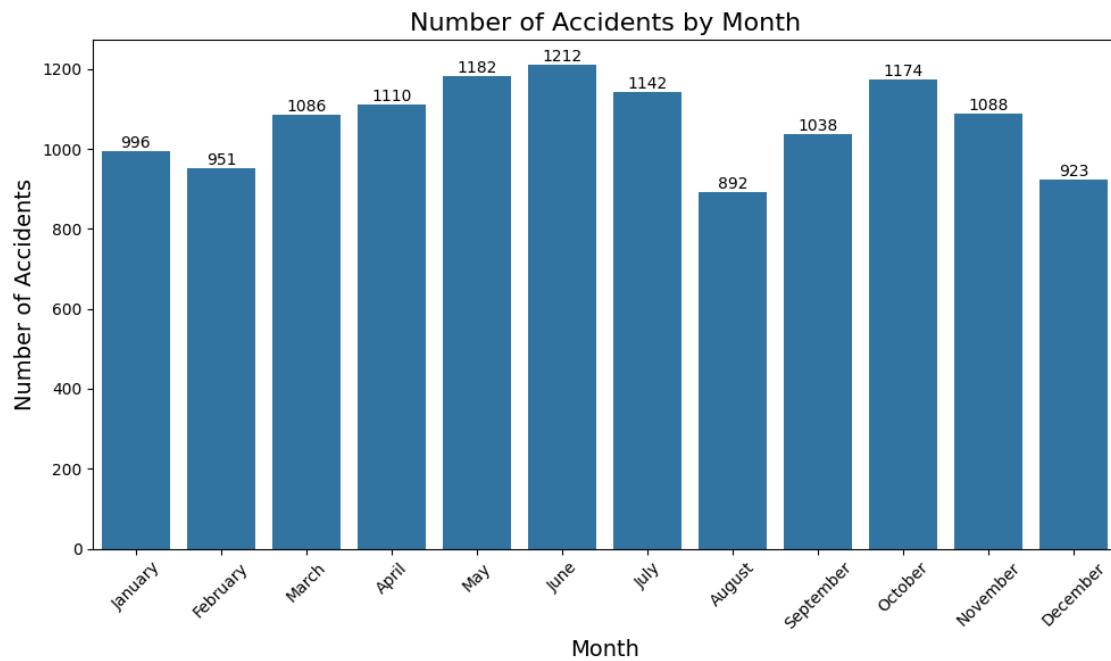
# Bar plot: Number of accidents by month
```

```

plt.figure(figsize=(10, 6))
ax = sns.countplot(data=accidents_temporal_df, x='month', order=list(month_mapping.
    ↪values()))
# Add count labels on top of the bars
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.
        ↪get_height()),
                ha='center', va='bottom', fontsize=10)

plt.title('Number of Accidents by Month', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Accidents', fontsize=14)
plt.xticks(rotation=45) # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()

```



We can also aggregate the accident counts by month of incidence, and provide a clearer visualization of the observations from the previous chart. A barplot shows two humps throughout the year of 2023, happening first in June, and later on in October. Also, seasons of lower accident reporting appear to happen during the winter months, with December having a low, and then in August the lowest count of records.

[62]: # Ensure that time_of_day is treated as an integer for correct sorting

```

accidents_temporal_df['time_of_day'] = accidents_temporal_df['time_of_day'].  

    ↪astype(int)

# Create a pivot table to count accidents by day of the week and time of day
heatmap_data = accidents_temporal_df.pivot_table(index='day_of_week',  

    ↪columns='time_of_day', aggfunc='size', fill_value=0)
# Reorder the days of the week for correct display
heatmap_data = heatmap_data.reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday',  

    ↪'Friday', 'Saturday', 'Sunday'])

# Plot the heatmap with original color scheme and smaller annotation font size
plt.figure(figsize=(12, 3.8))
sns.heatmap(  

    heatmap_data,  

    cmap='Blues',  

    annot=True,  

    fmt='d',  

    cbar_kws={'label': 'Number of Accidents'},  

    annot_kws={"size": 8} # Set the font size for annotations
)

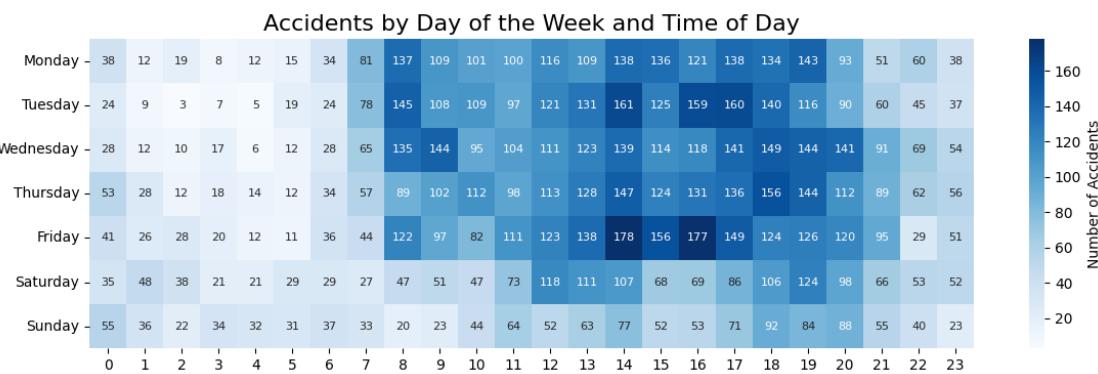
# Rotate the y-axis (day of the week) labels for better readability
plt.yticks(rotation=0)

# Set the title and axis labels
plt.title('Accidents by Day of the Week and Time of Day', fontsize=16)
plt.xlabel('', fontsize=14)
plt.ylabel('')

# Adjust layout for better spacing
plt.tight_layout()

# Show the plot
plt.show()

```



And lastly, the accidents by weekday and time of the day heatmap shows clear patterns in accident frequency that correspond to daily travel dynamics, particularly during rush hours. Accidents spike on weekdays during the 07:00 to 09:00 morning commute, especially on Wednesday, Thursday, and Friday. A more pronounced increase occurs during the evening rush hours from 16:00 to 19:00, with Friday at 17:00 showing the highest number of accidents, likely due to a combination of rush hour and the start of the weekend. While weekends generally have fewer accidents, there's a slight rise between 12:00 and 15:00, possibly linked to midday leisure travel. Overall, weekday rush hours are the most dangerous periods for accidents, particularly on Fridays.

4.4 Does a higher rate of male to female inhabitants per neighborhood correlate with accident counts?

Although the question is based on a stereotypical assumption, it is often observed that behaviors commonly associated with risk-taking, more frequently attributed to men, may correlate with a higher number of traffic incidents. With the availability of relevant data in our constructed database, we can explore this hypothesis.

Therefore, the goal for this chapter is to determine whether a higher male-to-female population ratio in Barcelona's districts correlates with a greater number of traffic accidents in those neighborhoods. To investigate this, we will aggregate traffic incidents by neighborhood and then combine this data with population figures to compare the male-to-female ratios recorded in the second dataset.

The choice of this question is purely motivated by the drive to perform an interesting data exploration which deals with both sources, rather than endorse or prove certain societal perceptions.

```
[63]: population_query_04 = [
    {
        "$group": {
            "_id": {
                "district_name": "$district_name",
                "neighborhood_name": "$neighborhood_name"
            },
            "total_population": {"$sum": "$total_population"},
            "male_population": {
                "$sum": {
                    "$cond": [
                        {"$eq": ["$Sex", "male"]},
                        "$total_population",
                        0
                    ]
                }
            },
            "female_population": {
                "$sum": {
                    "$cond": [
                        {"$eq": ["$Sex", "female"]},
                        "$total_population",
                        0
                    ]
                }
            }
        }
    }
]
```

```

        "$total_population",
        0
    ]
}
}
}
]
]

accidents_query_04 = [
{
    "$group": {
        "_id": {
            "district_name": "$district_name",
            "neighborhood_name": "$neighborhood_name"
        },
        "accident_count": {"$sum": 1}
    }
}
]

population_data = list(population_collection.aggregate(population_query_04))
accidents_data = list(accidents_collection.aggregate(accidents_query_04))

```

[64]:

```

combined_data = []
for pop in population_data:
    for acc in accidents_data:
        if (pop['_id']['district_name'] == acc['_id']['district_name'] and
            pop['_id']['neighborhood_name'] == acc['_id']['neighborhood_name']):
            combined_data.append({
                "district_name": pop['_id']['district_name'], # Include district_name
                "neighborhood_name": pop['_id']['neighborhood_name'],
                "total_population": pop['total_population'],
                "male_population": pop['male_population'],
                "female_population": pop['female_population'],
                "accident_count": acc['accident_count']
            })
# Create DataFrame from combined data
df_combined = pd.DataFrame(combined_data)

```

[65]:

```

# Calculate male-to-female ratio, handling division by zero by replacing it with NaN
df_combined['male_to_female_ratio'] = df_combined['male_population'] / \
    df_combined['female_population'].replace(0, pd.NA).astype(float)

df_combined['accidents_per_100_inhabitants'] = (df_combined['accident_count'] / \
    df_combined['total_population']) * 100

```

```

df_combined_sorted = df_combined.sort_values(by='accident_count', ascending=False)

result = df_combined_sorted[['district_name', 'neighborhood_name',_
    →'accident_count', 'male_to_female_ratio', 'accidents_per_100_inhabitants']]
result.head()

```

```
[65]:      district_name      neighborhood_name accident_count \
42          Eixample        la Dreta de l'Eixample      1160
2           Eixample  l'Antiga Esquerra de l'Eixample      633
57  Sarrià-Sant Gervasi      Sant Gervasi - Galvany      519
61          Eixample        la Sagrada Família      433
29          Eixample  la Nova Esquerra de l'Eixample      430

      male_to_female_ratio  accidents_per_100_inhabitants
42          0.8772079166            2.6300872917
2           0.8882116514            1.4806671189
57          0.8305175856            1.0886437052
61          0.8458682848            0.8373298267
29          0.8796801045            0.7467352042
```

At first glance, it can be noted that females slightly outnumber males in the municipal census, with the male-to-female ratio remaining close yet inferior to 1 in the top 10 neighborhoods by accident count. To assess the potential correlation between these variables, we can create a scatterplot comparing accident counts with the male-to-female ratio. By plotting a linear regression line, we can examine whether any trend or relationship between the two variables emerges from the data.

```

[66]: # Set figure size
plt.figure(figsize=(12, 8))

# Create a scatterplot with enhanced transparency and smaller points
sns.scatterplot(
    data=df_combined_sorted,
    x='male_to_female_ratio',
    y='accident_count',
    hue='district_name', # Color by 'district_name'
    alpha=0.6, # Increase transparency to avoid overlap
    s=70, # Decrease point size
    edgecolor='black' # Add an edge color to make points stand out
)

# Add a lighter regression line using regplot
sns.regplot(
    x='male_to_female_ratio',
    y='accident_count',
    data=df_combined_sorted,
    scatter=False, # Do not overlay scatter points again
    line_kws={'color': 'red', 'alpha': 0.3}, # Lighter regression line
)
```

```

        ci=None # Remove the confidence interval shading
    )

# Add text labels to every nth point
n = 8 # Set the spacing of the labels
for i in range(0, len(df_combined_sorted), n):
    plt.text(
        df_combined_sorted['male_to_female_ratio'].iloc[i],
        df_combined_sorted['accident_count'].iloc[i],
        df_combined_sorted['neighborhood_name'].iloc[i],
        fontsize=9,
        ha='right',
        va='bottom'
    )

# Plot settings
plt.title('Correlation between Accident Count and Male-to-Female Ratio', fontsize=16)
plt.xlabel('Male-to-Female Ratio', fontsize=14)
plt.ylabel('Accident Count', fontsize=14)
plt.ylim(bottom=0) # Set the minimum value of y-axis to 0

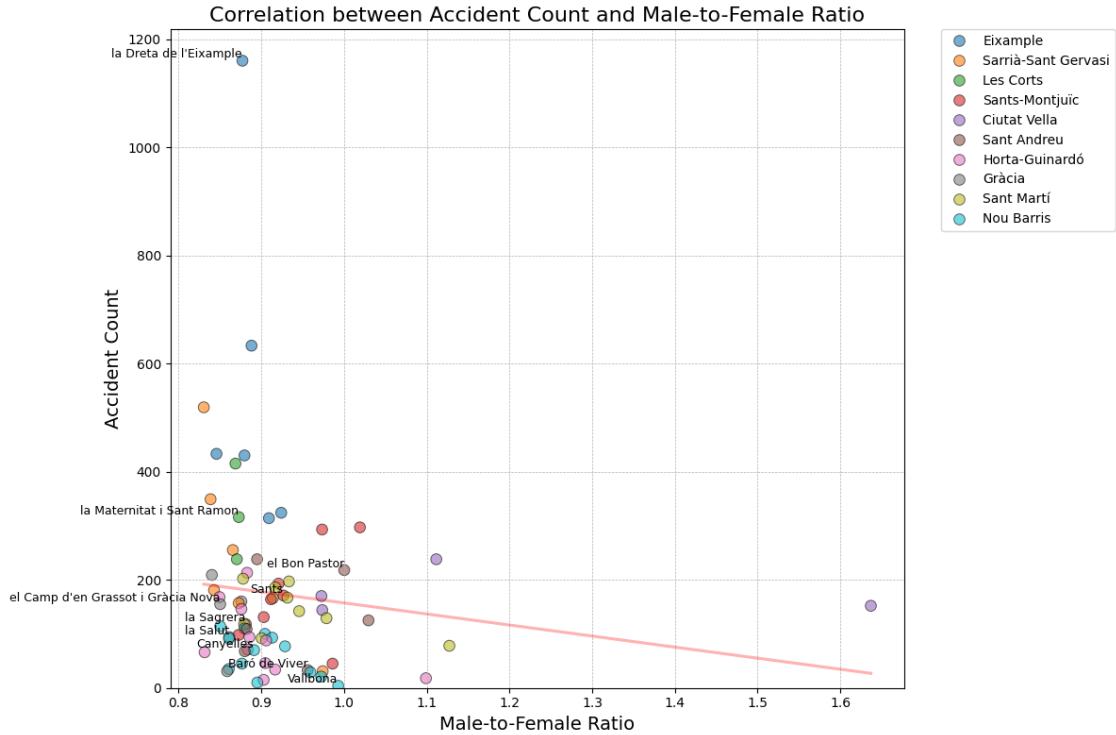
# Add a grid for better readability
plt.grid(True, which="both", linestyle="--", linewidth=0.5) # Grid on both major
# and minor ticks

# Move legend outside the plot
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0)

# Adjust layout for readability
plt.tight_layout()

# Show the plot
plt.show()

```



These results suggest that the initial hypothesis is not supported, as there is actually a slight negative trend between the male-to-female ratio and overall accident counts. However, as has been stated earlier, this sort of assessment ignores the observations as relative to number of inhabitants. We can reproduce the plot, but this time reflecting the accident per inhabitant ratio rather than the overall count.

```
[67]: # Set figure size
plt.figure(figsize=(12, 8))

# Create a scatterplot with enhanced transparency and smaller points
sns.scatterplot(
    data=df_combined_sorted,
    x='male_to_female_ratio',
    y='accidents_per_100_inhabitants',
    hue='district_name', # Color by 'district_name'
    alpha=0.6, # Increase transparency to avoid overlap
    s=70, # Decrease point size
    edgecolor='black' # Add an edge color to make points stand out
)

# Optionally, remove or lighten regression line if it's too dominant
sns.regplot(
    x='male_to_female_ratio',
    y='accidents_per_100_inhabitants',
```

```

    data=df_combined_sorted,
    scatter=False,
    line_kws={'color': 'red', 'alpha': 0.3}, # Lighter regression line
    ci=None # Remove the confidence interval shading
)

# Remove log scale adjustment
plt.xscale('linear') # Change x-axis to linear scale
plt.yscale('linear') # Keep y-axis on a linear scale

# Add text labels to selected or outlier points for better readability
selected_points = df_combined_sorted.nlargest(5, 'accidents_per_100_inhabitants') #
→# Label top 5 points
for i, row in selected_points.iterrows():
    plt.text(
        row['male_to_female_ratio'],
        row['accidents_per_100_inhabitants'],
        row['neighborhood_name'],
        fontsize=10,
        ha='right',
        va='bottom'
    )

# Plot settings
plt.title('Correlation between Accidents per 100 Inhabitants and Male-to-Female_
→Ratio', fontsize=16)
plt.xlabel('Male-to-Female Ratio', fontsize=14)
plt.ylabel('Accidents per 100 Inhabitants', fontsize=14)

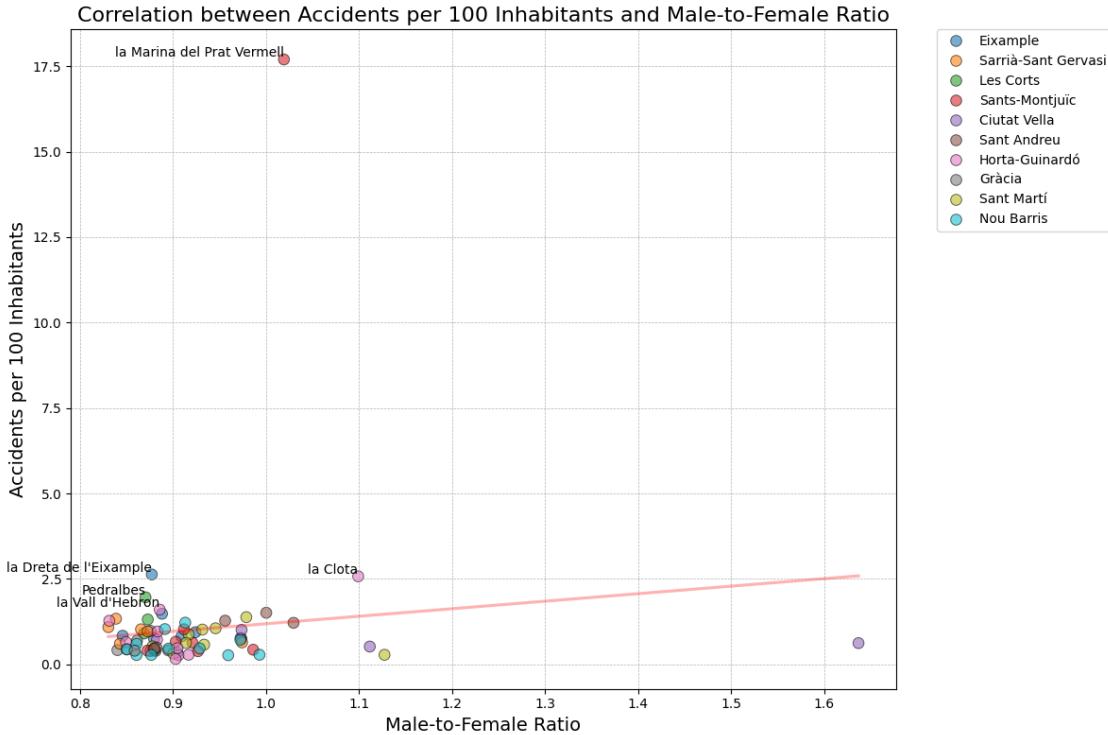
# Adjust grid and ticks for better readability
plt.grid(True, which="both", linestyle="--", linewidth=0.5) # Grid on both major_
→and minor ticks

# Move legend outside the plot
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0)

# Adjust layout for readability
plt.tight_layout()

# Show the plot
plt.show()

```



This time the trend appears to be increasing, suggesting some basis to the hypothesis that stereotypical reckless behaviors commonly associated with men may correlate with more accidents.

However, it's important to recognize that generalizations like these often stem from oversimplified interpretations of data. Correlation does not imply causation, and numerous other factors likely play a significant role in shaping accident rates. Additionally, the dataset used may not be well-suited for exploring this hypothesis, as many of the individuals using the streets in a given neighborhood may not actually reside there, meaning the population data may not accurately reflect the demographic makeup of the area's street users. Therefore, drawing conclusions from this data requires caution, and further analysis would need to account for a broader range of variables to better understand the underlying causes.

5 Conclusions

This project offers a comprehensive exploration of traffic accident patterns across Barcelona, utilizing open data sources and leveraging MongoDB for data storage and transformation. By integrating data on both traffic incidents and population demographics, we achieved a multidimensional view of the factors contributing to road safety issues in the city. The density map and various other visualizations helped identify key accident hotspots, with notable concentrations along the main arteries and intersections within the Eixample district. These insights align with known urban patterns, where areas with high traffic flow or complex intersections are more susceptible to accidents.

Furthermore, the analysis highlighted the correlation between accident occurrences and factors like population density and neighborhood traffic dynamics. By merging demographic information, we were able to pinpoint areas where safety interventions could be most impactful, whether by redesigning intersections, enhancing traffic law enforcement, or focusing public awareness campaigns. Although further research is necessary to account for external factors (such as visitor traffic and non-resident commuters), this project lays a strong foundation for data-informed urban safety strategies in Barcelona.

6 Learnings

Throughout this project, valuable knowledge and skills were acquired in multiple new areas, notably in modeling and building a MongoDB database, data querying, and the integration of these data engineering techniques with an Python-based ELT pipeline.

Working with MongoDB provided practical experience in handling semi-structured data within a NoSQL database. Learning to perform efficient aggregation pipelines was crucial in retrieving and transforming data efficiently, especially when dealing with nested structures like coordinates and multiple fields across collections. The ability to structure complex queries, such as grouping, filtering, and joining datasets on specific keys, was essential in extracting actionable insights.

As a second chapter, integrating MongoDB within an ELT pipeline broadened our understanding of managing data from extraction to transformation. Python's capabilities were leveraged to perform ETL tasks, such as fetching data from API sources, transforming it to a suitable structure, and loading it into MongoDB. This provided hands-on experience in building modular, reusable scripts that can be applied to other data-centric projects. Additionally, this knowledge of MongoDB integration can now be extended to more sophisticated data workflows, making it possible to store, manipulate, and analyze complex datasets with ease.

The project emphasized the importance of visual representation in understanding data trends, especially in spatial contexts. Tools like Plotly and Pandas enhanced our ability to visualize accident data in maps, scatter plots, and heatmaps, making patterns more accessible and interpretable. Through visualization, we were able to identify spatial and temporal trends in accident occurrence, leading to a more nuanced understanding of urban traffic issues.

In sum, this project has been a valuable exercise in practical data engineering, analytics, and visualization, reinforcing key technical skills in MongoDB and ETL workflows, as well as advancing knowledge in urban safety data analysis. These learnings will be instrumental in future projects where large, complex datasets need to be processed and analyzed to generate impactful insights.

As a final disclaimer, ChatGPT has been incorporated as a valuable tool for accelerating troubleshooting and refining analytical approaches. Its use was carefully balanced within fair and reasonable limits, ensuring that it served as a supportive resource rather than a substitute for in-depth understanding. By leveraging AI for quick debugging, technical guidance, and procedural suggestions, I was able to streamline workflows and generate insights more efficiently. Importantly, the theoretical foundations and detailed methodologies in each step were thoroughly understood and applied by me. AI integration merely enhanced my capabilities, allowing me to focus more on the critical aspects of data interpretation and insight generation, ultimately producing a more robust and timely report.