Project Documentation: Traffic Accidents ETL

Performed by:

- Isabella Pérez C. - 2230603
- Luz Angela Carabali M. - 2230652
- Nicolas Peña Irurita - 2232049

Version: 1.0

Update Date: March 2025

Link to repository: https://github.com/isabellaperezcav/Traffic-Accidents-ETL

---

1. Project Overview

1.1 Purpose

The **Traffic-Accidents-ETL** project aims to design and implement an Extraction, Transformation and Loading (ETL) pipeline to process a dataset of **209,306 traffic accident records** collected between **2018 and 2025**. Its main purpose is to identify patterns, trends and critical factors that allow improving road safety, using Business Intelligence (BI) tools such as **Power BI** for visualization and analysis. The processed data is stored in a **PostgreSQL** relational database to ensure its availability and scalability.

1.2 Scope

- **Data Period:** 2018 - 2025
- **Data Volume:** 209,306 records
- **Key Tools:** Python 3.12, PostgreSQL 15, Power BI Desktop
- **Deliverables:**
- Structured relational database in PostgreSQL.

- Interactive dashboards in Power BI.
- Analytical reports with trends and recommendations based on data.

1.3 Specific Objectives

- Standardize and clean data for further analysis.
- Detect temporal, climatic and severity patterns in accidents.
- Provide actionable information for road safety strategies.

1.4 Description of the Dataset

- **Source:** The dataset consists of **209,306 records** related to traffic accidents from https://www.kaggle.com/datasets/oktayrdeki/traffic-accidents
- **Time Period:** Data spans from **March 2018 to January 2025**.
- **Key Variables:**
    - `crash_date`: The date the accident occurred.
    - `traffic_control_device`: The type of traffic control device involved (e.g., traffic light, sign).
    - `weather_condition`: The weather conditions at the time of the accident.
    - `lighting_condition`: The lighting conditions at the time of the accident.
    - `first_crash_type`: The initial type of the crash (e.g., head-on, rear-end).
    - `trafficway_type`: The type of roadway involved in the accident (e.g., highway, local road).
    - `alignment`: The alignment of the road where the accident occurred (e.g., straight, curved).
    - `roadway_surface_cond`: The condition of the roadway surface (e.g., dry, wet, icy).
    - `road_defect`: Any defects present on the road surface.
    - `crash_type`: The overall type of the crash.
    - `intersection_related_i`: Whether the accident was related to an intersection.

- `damage`: The extent of the damage caused by the accident.
- `prim_contributory_cause`: The primary cause contributing to the crash.
- `num_units`: The number of vehicles involved in the accident.
- `most_severe_injury`: The most severe injury sustained in the crash.
- `injuries_total`: The total number of injuries reported.
- `injuries_fatal`: The number of fatal injuries resulting from the accident.
- `injuries_incapacitating`: The number of incapacitating injuries.
- `injuries_non_incapacitating`: The number of non-incapacitating injuries.
- `injuries_reported_not_evident`: The number of injuries reported but not visibly evident.
- `injuries_no_indication`: The number of cases with no indication of injury.
- `crash_hour`: The hour the accident occurred.
- `crash_day_of_week`: The day of the week the accident occurred.
- `crash_month`: The month the accident occurred.

---

2. Tools and Technologies

2.1 Development Environment

- **Python 3.12:** Main language for ETL and data analysis.
- Download: python.org
- **Virtual Environment (`venv`):** Isolation of project dependencies.
- **Jupyter Notebook:** Exploratory analysis and interactive documentation.
- Recommendation: Use VS Code with Jupyter extension.

2.2 Data Management

- **PostgreSQL 15:** Relational database for storage and queries.
- Download: postgresql.org

2.3 Visualization

- **Power BI Desktop:** Creation of interactive dashboards and reports.
- Download: microsoft.com

2.4 Python Dependencies

Listed in `requirements.txt`:

- `pandas`: Data manipulation.
- `numpy`: Numerical calculations.
- `matplotlib` and `seaborn`: Static visualizations.
- `sqlalchemy` and `psycopg2`: Connection to PostgreSQL.
- `jupyter`: Running notebooks.

---

3. Repository Structure

```
/Traffic-Accidents-ETL
|── 📂 config         # Future configurations
|   ├── requirements.txt   # Project dependencies
|   ├── conexion_db.py      # PostgreSQL connection configuration
|── 📂 data           # Input data
|   ├── traffic_accidentes.csv # Original dataset
|── 📂 Dashboard       # Database model and loading scripts
|   ├── proyectoETL.pbix    # Interactive Power BI dashboard
|   ├── proyectoETL.pdf  # Dashboard in PDF
|── 📂 notebooks       # Exploratory data analysis (EDA)
|   ├── 001_extract.ipynb    # Extraction and initial loading
|   └── 002_EDA.ipynb       # Transformation and exploratory analysis
|── 📂 venv          # Python virtual environment
|── .gitignore        # Files to exclude from version control
|── [README.md]       # Project documentation
```

---

4. ETL Process

4.1 Extract

- **Source:** CSV file (`traffic_accidentes.csv`) with 209,306 records.
- **Steps:**

1. Load the CSV with `pandas.read_csv()` in `001_extract.ipynb`.
2. Connect to PostgreSQL using `conexion_db.py` (using `SQLAlchemy` and `psycopg2`).
3. Insert into the `accidentes` table with `to_sql()`.

- **Validation:** SQL query in `pgAdmin` to verify the load.

4.2 Transformation

- **Notebook:** `002_EDA.ipynb`
- **Steps:**

1. **Load:** Reading the `accidents` table to a DataFrame with `pandas`.
2. **Cleaning:**

- Converting `crash_date` to `datetime` with `pd.to_datetime()`.
- Handling null values (`NaN`, `"UNKNOWN"`) by imputation or elimination.
- Creating derived columns: `year`, `month`.

1. **EDA:**

- Descriptive statistics (`df.describe()`).
- Visualizations (histograms, bars) for initial patterns.

4.3 Load

- **Destination:** `accidents` table in PostgreSQL.
- **Table Structure:**

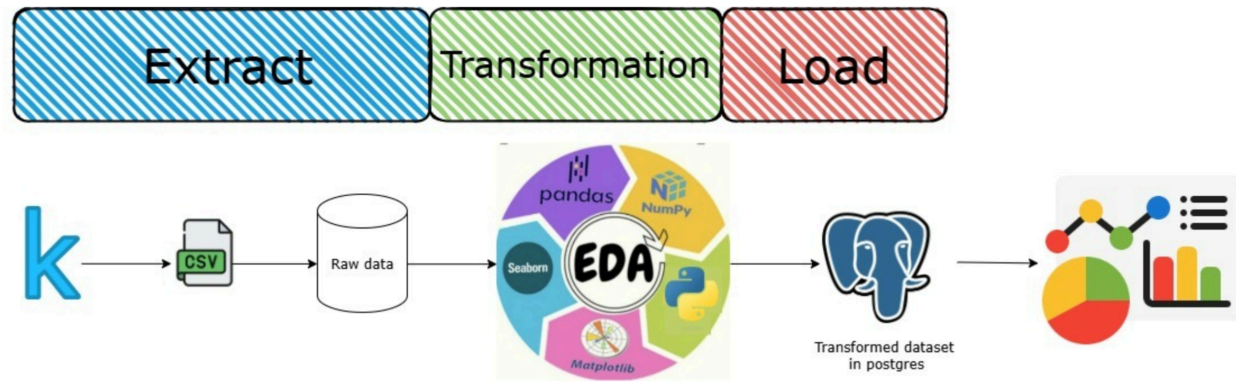
Column	Type	Description
id	SERIAL	Unique accident identifier

crash_date	TIMESTAMP	Accident date and time
traffic_control_device	TEXT	Traffic control device present
weather_condition	TEXT	Weather condition at the time of the accident
lighting_condition	TEXT	Lighting condition
first_crash_type	TEXT	Main collision type
trafficway_type	TEXT	Type of roadway where the accident occurred
alignment	TEXT	Road alignment
roadway_surface_cond	TEXT	Road surface condition
road_defect	TEXT	Road defects
crash_type	TEXT	General classification of the accident
intersection_related	CHAR(1)	Indicates if the accident occurred at an intersection (Y/N)
damage	TEXT	Recorded damage level
prim_contributory_cause	TEXT	Main cause of the accident
num_units	INT	Number of units (vehicles) involved
most_severe_injury	VARCHAR	Most severe injury reported
injuries_total	FLOAT	Total number of injured persons
injuries_fatal	FLOAT	Number of fatal injuries
injuries_incapacitating	FLOAT	Number of incapacitating injuries
injuries_non_incapacitating	FLOAT	Number of non-incapacitating injuries
injuries_reported_not_evident	FLOAT	Number of reported but non-evident injuries
injuries_no_indication	FLOAT	Number of people with no indication of injuries
crash_hour	INT	Hour when the accident occurred
crash_day_of_week	INT	Day of the week when the accident occurred
crash_month	INT	Month when the accident occurred

- **Process:** Writing transformed data to PostgreSQL.

- This table is created in `001_extract.ipynb`

4.4 Pipeline



---

5. Data Extraction (001_extract.ipynb)

5.1 Objective

Create table `accidents` in PostgreSQL and load it with the data from the csv.

5.2 Steps

- **Connecting to the database** using `conexion_db.py` (SQLAlchemy with `psycopg2`).

```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.getcwd(), "..")))
from config.conexion_db import conectar_db
```

```
✅ conexion_db.py se está ejecutando correctamente
```

- **Loading raw data** using `pandas.read_csv()`.

```python
# Cargar el CSV asegurando que el delimitador sea correcto
df = pd.read_csv(r"C:/Users/ASUS/Desktop/Traffic Accidents/data/traffic_accidents.csv")
df
```

Python

	crash_date	traffic_control_device	weather_condition	lighting_condition	first_crash_type	trafficway_type
0	07/29/2023 01:00:00 PM	TRAFFIC SIGNAL	CLEAR	DAYLIGHT	TURNING	NOT DIVIDED
1	08/13/2023 12:11:00 AM	TRAFFIC SIGNAL	CLEAR	DARKNESS, LIGHTED ROAD	TURNING	FOUR WAY

- **Storing data** in the PostgreSQL table `accidentes` via `to_sql()`.

```python
conexion = conectar_db()

df_verificacion = pd.read_sql("SELECT * FROM accidentes LIMIT 5;", conexion)
df_verificacion
```

[9] Python

```
✅ Conexión exitosa con psycopg2
C:\Users\ASUS\AppData\Local\Temp\ipykernel_11968\1011235367.py:4: UserWarning: pandas only supports SQLAlchemy connectable (engine/cc
  df_verificacion = pd.read_sql("SELECT * FROM accidentes LIMIT 5;", conexion)
```

	id	crash_date	traffic_control_device	weather_condition	lighting_condition	first_crash_type	trafficway_type	alignment	roadway_surface_cc
0	1	2023-07-29 13:00:00	TRAFFIC SIGNAL	CLEAR	DAYLIGHT	TURNING	NOT DIVIDED	STRAIGHT AND LEVEL	UNKNO
1	2	2023-08-13 00:11:00	TRAFFIC SIGNAL	CLEAR	DARKNESS, LIGHTED ROAD	TURNING	FOUR WAY	STRAIGHT AND LEVEL	D
2	3	2021-12-09 10:30:00	TRAFFIC SIGNAL	CLEAR	DAYLIGHT	REAR END	T-INTERSECTION	STRAIGHT AND LEVEL	D
3	4	2023-08-09 19:55:00	TRAFFIC SIGNAL	CLEAR	DAYLIGHT	ANGLE	FOUR WAY	STRAIGHT AND LEVEL	D
4	5	2023-08-19 14:55:00	TRAFFIC SIGNAL	CLEAR	DAYLIGHT	REAR END	T-INTERSECTION	STRAIGHT AND LEVEL	UNKNO

- **Verifying** data upload using `pgAdmin`.

---

5. Exploratory Data Analysis (002_EDA.ipynb)

5.1 Objective

Explore the data distribution and prepare the dataset for visualization.

5.2 Insights

- **Data loading**: Extracted from PostgreSQL into a Pandas DataFrame.

```python
df = pd.read_sql("SELECT * FROM accidentes LIMIT 5;", conexion)
df
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_23256\639739079.py:1: UserWarning: pandas only supports SQLAlchemy co
  df = pd.read_sql("SELECT * FROM accidentes LIMIT 5;", conexion)
```

	id	crash_date	traffic_control_device	weather_condition	lighting_condition	first_crash_type	trafficway_type	alignme
0	1	2023-07-29 13:00:00	TRAFFIC SIGNAL	CLEAR	DAYLIGHT	TURNING	NOT DIVIDED	STRAIG AI LEV
1	2	2023-08-13 00:11:00	TRAFFIC SIGNAL	CLEAR	DARKNESS, LIGHTED ROAD	TURNING	FOUR WAY	STRAIG AI LEV
2	3	2021-12-09 10:30:00	TRAFFIC SIGNAL	CLEAR	DAYLIGHT	REAR END	T-INTERSECTION	STRAIG AI LEV

- **Exploratory analysis**:

Before proceeding with further analysis, we inspect the structure of the dataset using `df.info()`. This helps us understand:

✅ The number of non-null values per column.

```python
print("Información básica del dataset:")
print(df.info())
```

```
Información básica del dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 25 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   id                        5 non-null      int64
 1   crash_date                5 non-null      datetime64[ns]
 2   traffic_control_device    5 non-null      object
 3   weather_condition         5 non-null      object
 4   lighting_condition        5 non-null      object
 5   first_crash_type          5 non-null      object
 6   trafficway_type           5 non-null      object
 7   alignment                 5 non-null      object
 8   roadway_surface_cond      5 non-null      object
 9   road_defect               5 non-null      object
 10  crash_type                5 non-null      object
 11  intersection_related      5 non-null      object
 12  damage                    5 non-null      object
 13  prim_contributory_cause   5 non-null      object
 14  num_units                 5 non-null      int64
 15  most_severe_injury        5 non-null      object
 16  injuries_total            5 non-null      float64
 17  injuries_fatal            5 non-null      float64
 18  injuries_incapacitating   5 non-null      float64
...
 24  crash_month               5 non-null      int64
dtypes: datetime64[ns](1), float64(6), int64(5), object(13)
memory usage: 1.1+ KB
None
```

- **Descriptive statistics**: Summary of numerical and categorical data.

```python
df['most_severe_injury'] = pd.Categorical(df['most_severe_injury'],
                                          categories=['NO INDICATION OF INJURY', 'REPORTED, NOT EVIDENT',
                                                      'NON-INCAPACITATING INJURY', 'INCAPACITATING INJURY',
                                          ordered=True)
```

```python
df['intersection_related'] = df['intersection_related'].apply(lambda x: 1 if x == 'Y' else 0)
```

```
print("\nValores faltantes por columna:")
print((df.isnull().sum() + (df == "UNKNOWN").sum()).to_string())
```

```
Valores faltantes por columna:
id                             0
crash_date                     0
traffic_control_device         0
weather_condition              0
lighting_condition             0
first_crash_type               0
trafficway_type                0
alignment                      0
roadway_surface_cond           2
road_defect                    2
crash_type                     0
intersection_related           0
damage                         0
prim_contributory_cause        0
num_units                      0
most_severe_injury             1
injuries_total                 0
injuries_fatal                 0
injuries_incapacitating        0
injuries_non_incapacitating    0
injuries_reported_not_evident  0
injuries_no_indication         0
crash_hour                     0
crash_day_of_week              0
crash_month                    0
```

- **Data cleaning**:
    - Converted categorical variables to appropriate formats.

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 25 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   id                             5 non-null      int64
 1   crash_date                     5 non-null      datetime64[ns]
 2   traffic_control_device         5 non-null      object
 3   weather_condition              5 non-null      object
 4   lighting_condition             5 non-null      object
 5   first_crash_type               5 non-null      object
 6   trafficway_type                5 non-null      object
 7   alignment                      5 non-null      object
 8   roadway_surface_cond           5 non-null      object
 9   road_defect                    5 non-null      object
 10  crash_type                     5 non-null      object
 11  intersection_related           5 non-null      int64
 12  damage                         5 non-null      object
 13  prim_contributory_cause        5 non-null      object
 14  num_units                      5 non-null      int64
 15  most_severe_injury             4 non-null      category
 16  injuries_total                 5 non-null      float64
 17  injuries_fatal                 5 non-null      float64
 18  injuries_incapacitating        5 non-null      float64
 19  injuries_non_incapacitating    5 non-null      float64
...
 24  crash_month                    5 non-null      int64
dtypes: category(1), datetime64[ns](1), float64(6), int64(6), object(11)
memory usage: 1.3+ KB
```

    - Handled missing values (`NaN` and "UNKNOWN" entries).

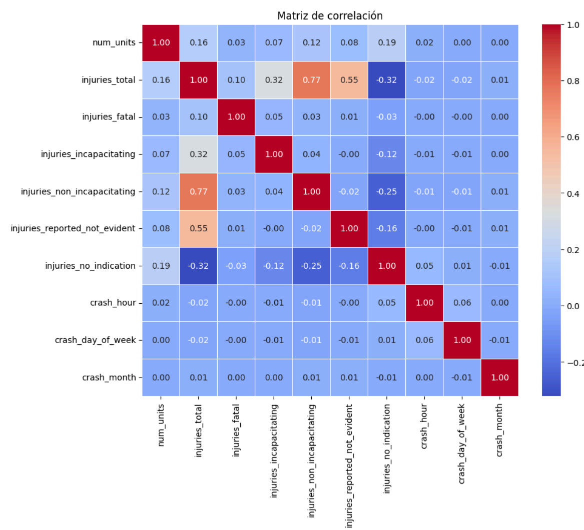
```
print("\nValores faltantes por columna:")
print((df.isnull().sum() + (df == "UNKNOWN").sum()).to_string())
```

```
Valores faltantes por columna:
id                             0
crash_date                     0
traffic_control_device         0
weather_condition              0
lighting_condition             0
first_crash_type               0
trafficway_type                0
alignment                      0
roadway_surface_cond           2
road_defect                    2
crash_type                     0
intersection_related           0
damage                         0
prim_contributory_cause        0
num_units                      0
most_severe_injury             1
injuries_total                 0
injuries_fatal                 0
injuries_incapacitating        0
injuries_non_incapacitating    0
injuries_reported_not_evident  0
injuries_no_indication         0
crash_hour                     0
crash_day_of_week              0
crash_month                    0
```

- **Handled missing values** (NaN entries) in the `most_severe_injury` column.
  Identified distribution of injury severity, confirming that most cases were labeled as `"NO INDICATION OF INJURY"`, with a few missing values.
- **Checked for specific injury classifications**, specifically `"NON-INCAPACITATING INJURY"`, but found no matching records.

5.3 Graphical Analysis of the Dataset

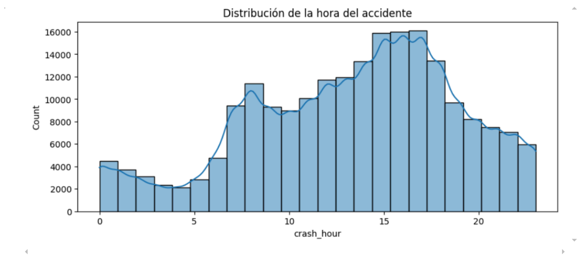
Correlation Analysis



- A strong relationship is observed between injuries_total and injuries_non_incapacitating, indicating that these types of injuries represent a significant portion of total injuries.
- The variable injuries_no_indication has a negative correlation with reported injuries, suggesting that when there is no indication of injury, other types of injuries are less likely to be reported.
- Factors such as crash_hour, crash_day_of_week, and crash_month have little or no correlation with injuries, indicating that the severity of accidents may be more dependent on other factors such as speed or road conditions.

Accident Distribution by Hour

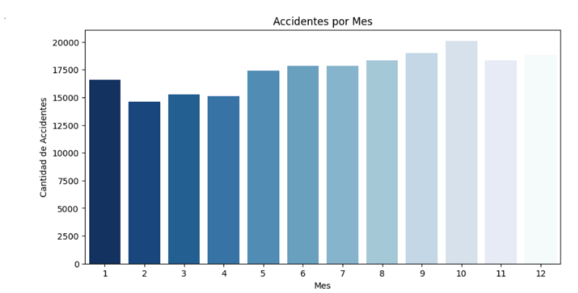
Distribución de la hora del accidente:



Specific Analysis of Accident Time Distribution

- **Low incidence at dawn (00:00 - 05:00):** Few accidents, likely due to lower traffic, but potentially linked to fatigue or alcohol consumption.
- **Increase in the morning (06:00 - 09:00):** Notable rise, peaking between 08:00 and 09:00, coinciding with rush hour for work and school.
- **Stability from mid-morning to noon (10:00 - 14:00):** Relatively steady accident rate, possibly due to more evenly distributed traffic.
- **Peak in the afternoon (15:00 - 18:00):** Highest number of accidents, especially between 16:00 and 17:00, aligning with work and school departures.
- **Decrease in the evening (19:00 - 23:00):** Gradual decline, though accidents still occur, potentially due to reduced visibility and driver fatigue.

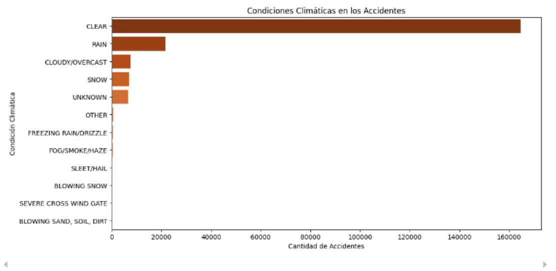
Histogram of Accidents by Month



- January and October are the most dangerous months, so preventive measures should be reinforced during these periods.
- From May to October there is a constant increase, which suggests that climatic and work factors have a strong impact.

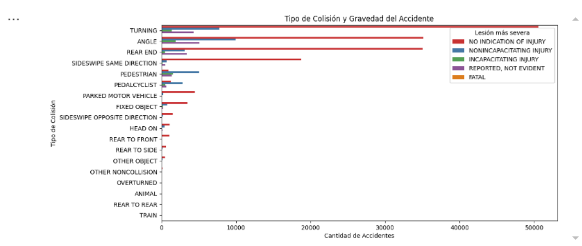
- December may have specific peaks, but in general, the month does not exceed October in total number of accidents.

Weather Conditions Analysis in Accidents



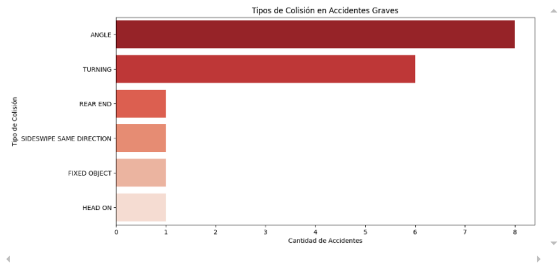
- Most accidents occur in clear conditions, suggesting that driver behavior is a key factor.
- Rain is the second most important weather cause, increasing the risk of skidding and crashes.
- More extreme conditions such as snow, fog or strong winds cause fewer accidents, but require greater caution.

Relationship between Collision Type and Accident Severity



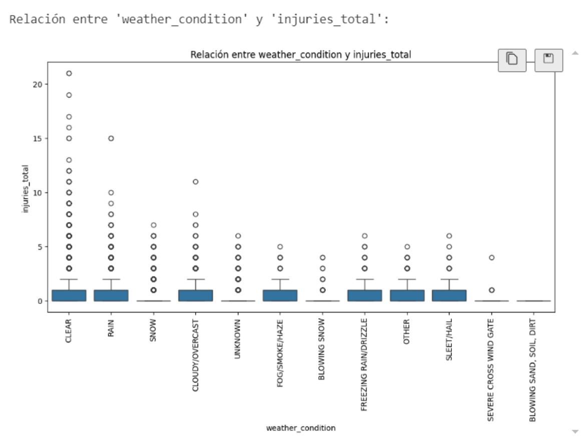
- Cornering and cornering collisions are the most common types of crashes, but most do not result in serious injuries.
- Crashes with fixed objects, rear-end collisions, and run-overs have a higher risk of disabling or fatal injuries.
- Head-on collisions and collisions with trains are less common, but their danger is much greater.

Filter Accidents with More than 10 Injuries



- Cornering and turning collisions are the most common causes of serious accidents.
- Rear-end and side collisions indicate problems with attention and safe driving.
- Head-on collisions, although less common, are the most dangerous.

Analysis of the Relationship between Categorical and Numerical Variables



- Clear weather does not prevent accidents with multiple injuries, as most accidents occur under these conditions.
- Rain and fog conditions can increase the severity of accidents.
- Extreme weather conditions (snow storms, strong winds) show fewer accidents with injuries, possibly because there is less vehicle traffic in these conditions.

5.4 Main Findings

- **Seasonality:** Higher frequency during peak hours (16:00-17:00); lower in the early morning (00:00-05:00).
- **Critical Months:** January and October lead in incidents.
- **Weather:** Accidents in clear conditions predominate, suggesting human factors.
- **Collisions:** Angle (frequent, less serious); frontal (rare, more lethal).

5.5 Visualizations

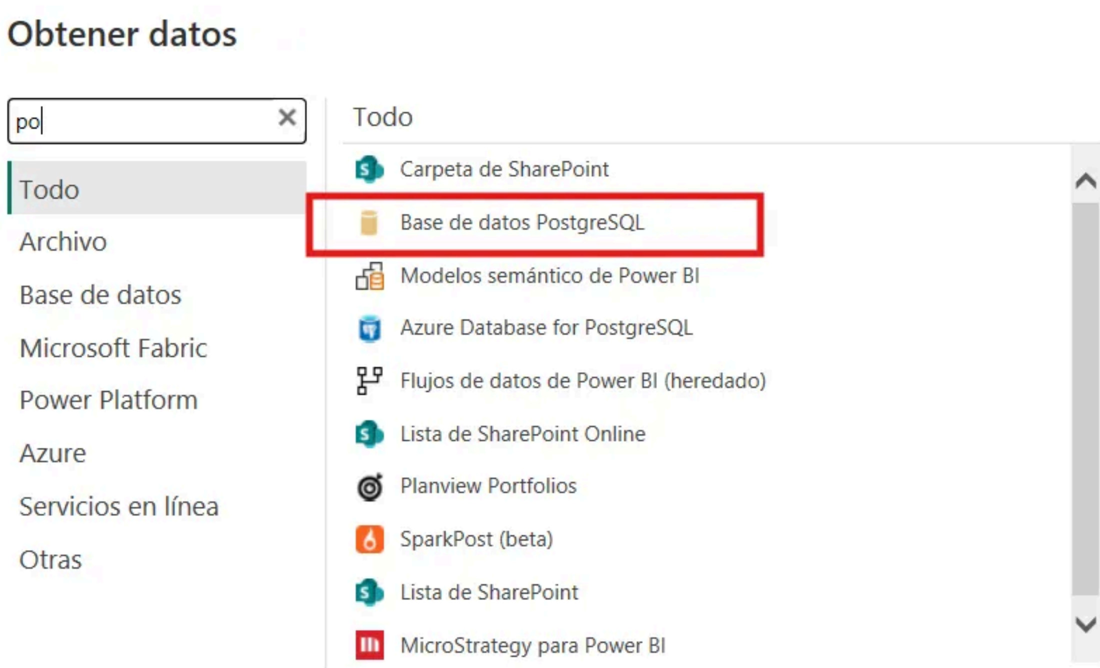
- Histograms by hour and month.
- Bar graphs by weather and type of collision.
- Correlation matrix for numerical variables.

---

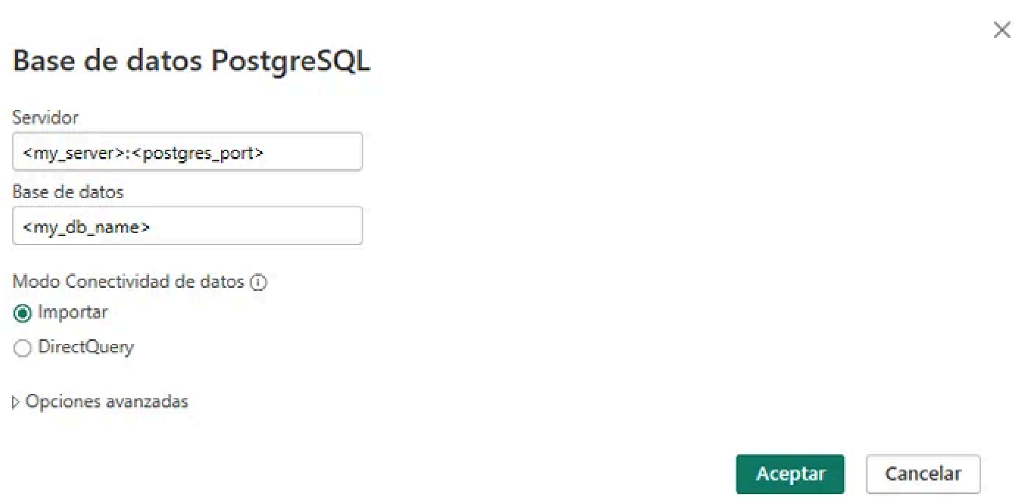
6. Visualization and Reporting

6.1 Connecting to Power BI

1. Launch Power BI Desktop.
2. Select "Get Data" > "PostgreSQL Database".



3. Set credentials and select the `accidentes` table.



6.2 Dashboards

- **Temporal Trends:** Accidents by hour, day, and month.
- **Weather and Severity:** Relationship between weather conditions and injuries.
- **Collision Types:** Distribution and associated severity.

---

7. Conclusions

- Rush hours and clear conditions are common accident scenarios.
- January and October are critical periods.
- Head-on collisions generate greater severity.

---

8. Instructions for Use

8.1 Configuration

1. Clone the repository:

```
python -m venv venv
venv\\Scripts\\activate  # Windows
```

2. Install dependencies:

```
pip install -r requirements.txt
```

8.2 Connecting to PostgreSQL

1. Install PostgreSQL and create the `traffic_accidents` database.
2. Edit `connection_db.py` with your credentials.
3. Verify the connection by running the script.

8.3 Execution

1. Start Jupyter:

```
jupyter notebook
```

2. Run in order: `001_extract.ipynb`, `002_EDA.ipynb`.
3. Open `proyectoETL.pbix` in Power BI or check `proyectoETL.pdf`.