

Final ETL project



William Alejandro Botero Florez

Juan Manuel Lopez Rodriguez

Javier Alejandro Vergara Zorrilla

ETL (Extract, Transform, Load)

Autonomous University of the West

Faculty of Engineering

Santiago de Cali

2024

Documentation and evidence of the Traffic Accident Analysis Project

1. Introduction

This project focuses on the implementation of a traffic accident analysis system using multiple technologies and data analysis tools, with the aim of offering a comprehensive solution for data extraction, transformation, analysis and visualization in real time. The project is divided into six main phases:

- **Data Collection:** Gathering data from a static dataset and an external API to ensure a rich and diverse dataset.
- **Dimensional Modeling:** Designing a dimensional model to organize and store data efficiently for analysis.
- **Kafka Integration:** Implementing Kafka for real-time data streaming and seamless communication between system components.
- **Real-Time Visualization with Looker Studio:** Creating a dynamic dashboard to visualize the latest data updates in real-time.
- **Airflow Orchestration:** Developing an Airflow DAG to automate ETL processes, ensuring a streamlined workflow.
- **Conclusion:** Summarizing the outcomes and the effectiveness of the implemented solution.

2. Data Sources

In this phase, two different data sources were selected to feed the analysis system:

1. **US Car Accident Dataset:** This is historical data on traffic accidents in the USA from 2016 to 2023, stored in CSV format. This data includes detailed information about the accidents, such as the date, location, and number of people injured or killed.

For this data set the following process was carried out

Database

All this information was stored in a postgresql database, with a total of 500 thousand rows and 46 columns

There was information such as accident id, accident severity, accident start time and end time, accident coordinates, accident description, accident location, city, state, region, continent, weather conditions, time zone, time of day etc...

1 SELECT * FROM public.us_accidents
2 ORDER BY id ASC

Data Output Messages Notifications

	id [PK] character varying (100)	source character varying (50)	severity double precision	start_time timestamp without time zone	end_time timestamp without time zone	start_lat double precision	start_lng double precision	enc dou
1	A-1000009	Source2		2021-06-17 08:09:19	2021-06-17 09:23:35	38.86628	-77.323746	
2	A-1000025	Source3		2021-06-17 11:25:06	2021-06-17 13:10:17	40.596088	-77.86238900000002	
3	A-1000039	Source2		2021-06-17 16:23:07	2021-06-17 18:57:54	38.909958	-77.218094	
4	A-1000045	Source2		2021-06-17 19:19:22	2021-06-17 21:35:19	38.670273	-77.25183100000002	
5	A-100006	Source2		2016-04-05 16:06:41	2016-04-05 17:06:41	34.034119	-118.013443	
6	A-1000061	Source2		2021-06-17 13:13:03	2021-06-17 14:59:00	38.348675	-85.48835799999998	
7	A-1000079	Source2		2021-06-17 06:09:44	2021-06-17 06:34:00	39.440098	-84.337097	
8	A-1000099	Source2		2021-06-17 13:14:10	2021-06-17 14:15:00	38.051193	-84.34904499999998	
9	A-1000104	Source2		2021-06-17 15:25:14	2021-06-17 17:24:58	40.000938	-82.984604	
10	A-1000113	Source2		2021-06-17 17:37:02	2021-06-17 18:36:17	38.041481	-84.473122	
11	A-1000118	Source2		2021-06-17 18:01:54	2021-06-17 19:01:39	39.368912	-84.28649899999998	
12	A-1000131	Source2		2021-06-17 05:14:23	2021-06-17 07:18:14	35.108231	-80.881676	
13	A-1000135	Source2		2021-06-17 07:20:52	2021-06-17 08:50:53	36.947983	-80.923965	
14	A-1000153	Source2		2021-06-17 09:20:35	2021-06-17 10:50:12	35.20216	-80.78872700000002	
15	A-1000171	Source2		2021-06-17 15:56:23	2021-06-17 16:56:17	35.11005	-81.871437	

Total rows: 1000 of 500000 Query complete 00:00:19.264 Ln 1, Col 1

EDA

Database connection:

In order to use this information and bring it to our domain we had to make a connection to this postgresql database, for this we use the python sqlalchemy libraries and also dotenv, dotenv creates a file for us to have the credentials that we are going to use, for example in this case our .env file would look like this:

PostgreSQL database connection settings

DB_HOST=localhost

DB_PORT=5432

DB_USER=your_postgresql_user

DB_PASS=your_postgresql_password

DB_NAME=ETL_Project1

Once we have our .env file configured we can finish with the connection to our database by establishing a connection

```
src > db_conexion.py > ...
willyb, last month | 2 authors (willyb and one other)
1 from sqlalchemy import create_engine
2 from sqlalchemy.orm import sessionmaker
3 from dotenv import load_dotenv willyb, last month * Añadir DAG de ETL para datos de accidente
4 import os
5
6 # Cargar las variables de entorno desde el archivo .env
7 load_dotenv()
8
9 # Obtener las credenciales desde las variables de entorno
10 db = os.getenv('DB_NAME')
11 usuario = os.getenv('DB_USER')
12 token = os.getenv('DB_PASS')
13 host1 = os.getenv('DB_HOST')
14 puerto = os.getenv('DB_PORT')
15
16 def establecer_conexion():
17     # Crear la cadena de conexión para SQLAlchemy
18     connection_string = f'postgresql+psycopg2://{usuario}:{token}@{host1}:{puerto}/{db}'
19     engine = create_engine(connection_string)
20
21     # Crear una sesión (opcional, si necesitas trabajar con ORM más adelante)
22     Session = sessionmaker(bind=engine)
23     session = Session()
24     print("Conexión exitosa a la base de datos")
25     return engine, session # Devolver tanto el engine como la session si es necesario
26
27 def cerrar_conexion(session):
28     session.close()
29     print("Conexión cerrada a la base de datos")
30
31 # Establecer la conexión
32 engine, session = establecer_conexion()
33
34 # Ahora puedes usar `engine` con pandas sin problemas
35
```

Extracting the database:

Here we establish the connection again by importing the db_connection and we make a sql query to extract the data and we load the dataset to then save this data in a csv file

```
src > extract.py > extract_data
willyb, last month | 2 authors (willyb and one other)
1 import pandas as pd
2 from db_conexion import establecer_conexion, cerrar_conexion
3
4 def extract_data():
5     # Establece la conexión usando SQLAlchemy
6     engine, session = establecer_conexion() # Ahora engine es el primero que se recibe
7
8     # Consulta SQL para extraer datos
9     query = "SELECT * FROM us_accidents"
10
11     # Carga los datos en un DataFrame de pandas usando el engine de SQLAlchemy
12     df = pd.read_sql(query, con=engine)
13
14     # Cierra la sesión y la conexión
15     cerrar_conexion(session)
16     print("Datos cargados con éxito")
17
18     # Guarda los datos en un archivo CSV para la transformación
19     df.to_csv('data/us_accidents_raw.csv', index=False)
20     print("Datos guardados en 'data/us_accidents_raw.csv'")
21
22     # Puedes llamar a la función `extract_data()` si deseas ejecutar el script directamente
23 if __name__ == "__main__":
24     extract_data()
25
26
```

Data cleaning:

In this data cleaning we do the following:

- Connect to the database
- Delete the columns that we will not use later:
- 'id', 'source', 'country', 'description', 'end_lat', 'end_lng', 'civil_twilight', 'nautical_twilight', 'astronomical_twilight'
- Impute some missing values
- Perform other cleaning processes
- Check to see if there are no nulls after

```

src > data_cleaning.py > clean_data
willyb, last month | 3 authors (willyb and others)
1 import pandas as pd
2 from db_conexion import establecer_conexion, cerrar_conexion
3
4 def clean_data():
5     # Establece la conexión usando SQLAlchemy
6     engine, session = establecer_conexion()
7
8     # SQL query para seleccionar todos los datos de la tabla 'us_accidents'
9     query = "SELECT * FROM us_accidents"
10
11     # Lee los datos en un DataFrame de pandas usando el engine de SQLAlchemy
12     df = pd.read_sql(query, con=engine)
13
14     # Configura pandas para mostrar más filas y columnas
15     pd.set_option('display.max_rows', 100) # Mostrar hasta 100 filas
16     pd.set_option('display.max_columns', None) # Mostrar todas las columnas sin truncar
17     pd.set_option('display.width', None) # Ajustar automáticamente el ancho de visualización
18
19     # Mostrar las primeras 20 filas en formato tabular
20     print(df.head(20))
21
22     # Columnas a eliminar
23     columns_to_drop = ['id', 'source', 'country', 'description', 'end_lat', 'end_lng',
24                       'civil_twilight', 'nautical_twilight', 'astronomical_twilight']
25
26     # Eliminar las columnas especificadas
27     df_cleaned = df.drop(columns=columns_to_drop)
28
29     # Imputar valores faltantes en columnas numéricas con la media
30     df_cleaned['temperature_f'].fillna(df_cleaned['temperature_f'].mean(), inplace=True)
31
32     # Imputar valores faltantes en columnas categóricas con la moda (valor más frecuente)
33     df_cleaned['weather_condition'].fillna(df_cleaned['weather_condition'].mode()[0], inplace=True)
34
35     # Imputar valores faltantes en múltiples columnas numéricas con la media
36     num_cols = ['wind_chill_f', 'humidity_percent', 'pressure_in', 'visibility_mi', 'wind_speed_mph', 'precipitation_in']
37     df_cleaned[num_cols] = df_cleaned[num_cols].apply(lambda col: col.fillna(col.mean()))

```

After this data cleaning we wanted to see some analysis with the graphics

Dashboard:



2. New York accidents API: To enrich the data, an external API called `data.cityofnewyork.us` was used, which provides additional information such as the weather conditions at the time of each accident, among other things. The API was integrated into the ETL flow to ensure that each record included the corresponding weather information.

For this API, the following process was carried out

Connection to the API

We defined the function to bring the API to our domain and we defined the parameters that the data limit it brings is 200,000 to finally save that data in a csv in the data folder that we will use later for data cleaning. This API had information such as the date, time of the accident, locations, injured people, dead people, vehicle, street, etc...

```
src > API_connection.py > ...
willyb, last month | 1 author (willyb)
1 import requests
2 import pandas as pd
3
4 def get_data_from_api():
5     # URL of the dataset (API endpoint)
6     url = "https://data.cityofnewyork.us/resource/h9gi-nx95.json"
7
8     # Parameters to limit the response to 200,000 records
9     params = {
10         "$limit": 200000
11     }
12
13     # Send a GET request to the API
14     response = requests.get(url, params=params)
15
16     # Check if the request was successful
17     if response.status_code == 200:
18         data = response.json() # Convert the response to JSON format
19         df = pd.DataFrame(data) # Create a pandas DataFrame from the data
20         print(df.head()) # Display the first few records
21         # Save the DataFrame to a CSV file
22         df.to_csv('data/API_data.csv', index=False, encoding='utf-8')
23         print("Data downloaded and saved to data/API_data.csv")
24     else:
25         # If the request fails, print the error code
26         print(f"Error in the request: {response.status_code}")
27
```

Cleaning the data from the New York API:

In this cleaning, the following transformations were performed:

- The API CSV was loaded
- Crash date and crash time were converted to datetime format
- Duplicate rows were removed
- Unnecessary columns were deleted
- A new column called city was added, in which the only data would be “new york”
- At the end, the clean CSV will be saved in the data folder

```
src > API_cleaning.py > clean_api_data
4 def clean_api_data():
5     # Get the absolute path of the file
6     file_path = os.path.abspath(os.path.join('data/API_data.csv'))
7
8     # Load the CSV file using pandas
9     data = pd.read_csv(file_path)
10
11     # Set pandas to display all columns
12     pd.set_option('display.max_columns', None)
13
14     # Convert `crash_date` and `crash_time` to datetime format
15     data['crash_date'] = pd.to_datetime(data['crash_date'], errors='coerce')
16     data['crash_time'] = pd.to_datetime(data['crash_time'], format='%H:%M', errors='coerce')
17
18     # Fix inconsistent values (e.g., remove whitespace or correct capitalization in the `borough` column)
19     data['borough'] = data['borough'].str.strip().str.title()
20
21     # Filter data to keep only rows with valid crash dates and from the year 2021 or later
22     data = data[data['crash_date'].notna() & (data['crash_date'].dt.year >= 2021)]
23
24     # Remove duplicates based on the `collision_id` column (assuming it's unique for each accident)
25     data = data.drop_duplicates(subset='collision_id')
26
27     # Convert `crash_date` to just the date (drop the time part)
28     data['crash_date'] = data['crash_date'].dt.date
29
30     # Drop unnecessary columns
31     data = data.drop(['vehicle_type_code_5', 'contributing_factor_vehicle_5',
32                     'vehicle_type_code_4', 'contributing_factor_vehicle_4',
33                     'vehicle_type_code_3', 'contributing_factor_vehicle_3',
34                     'cross_street_name'], axis=1)
35
36     # Drop rows with any missing values
37     data = data.dropna()
38
39     # Add a `city` column with the value "New York"
40     data['city'] = "New York"
```


Merge:

A merge was performed between the `Accidents_Usa_clean` and `API_data_cleaned` csv files, the following transformations were made:

- **Data loading:** Two CSV files were loaded
- **Date conversion:** The date columns in both datasets were converted to datetime format, keeping only the date part.
- **Filtering by city:** Both datasets were filtered to include only data related to New York City.
- **Dataset merge:** An inner join was performed using the date columns (`crash_date` and `start_time`), thus combining climate data with accident information.
- **Duplicate column removal:** The duplicate city columns (`city_x` and `city_y`) generated after the merge were removed.
- **Date format and new column:** The date column was formatted to include only year and month, and a 'city' column was added with the fixed value "New York".
- **Column reorganization:** Moved the 'city' column to the beginning of the DataFrame.
- **Crash time adjustment:** Converted the `crash_time` column to display only the time.
- **Integer conversion:** Converted the `number_of_persons_injured` column to integer type, handling null values.
- **Unnecessary columns removal:** Removed columns not relevant to the analysis, such as coordinates, distances, secondary identifiers, and redundant data.
- **Field merging:** Joined the `borough` and `zip_code` columns, and merged city with state.
- **Unique ID assignment:** Assigned a unique identifier to each row and moved it to the beginning of the DataFrame.
- **Final validation:** Checked the number of rows after the merge and checked for null values.

- **Saved to CSV:** The final, cleaned DataFrame was saved to a file named `merged_data.csv`.

```
src > merge_data.py > merge_data
willyb, 8 hours ago | 2 authors (willyb and one other)
1 import pandas as pd
2
3 def merge_data():
4     # Load the CSV files
5     API_merge = pd.read_csv('data/API_data_Cleaned.csv')
6     db_merge = pd.read_csv('data/us_accidents_cleaned.csv')
7
8     # Convert date columns to datetime format and use only the date
9     API_merge['crash_date'] = pd.to_datetime(API_merge['crash_date']).dt.date # Use only the date
10    db_merge['start_time'] = pd.to_datetime(db_merge['start_time']).dt.date # Use only the date
11
12    # Filter both datasets for rows where the city is 'New York'
13    api_data_ny = API_merge[API_merge['city'] == 'New York']
14    us_accidents_ny = db_merge[db_merge['city'] == 'New York']
15
16    # Merge the two datasets based on the date (inner join)
17    merged_df = pd.merge(api_data_ny, us_accidents_ny, left_on='crash_date', right_on='start_time', how='inner')
18
19    # Drop duplicate city columns ('city_x' and 'city_y')
20    merged_df = merged_df.drop(columns=['city_x', 'city_y'])
21
22    # 1. Asegúrate de que la columna de fecha esté en formato datetime
23    merged_df['crash_date'] = pd.to_datetime(merged_df['crash_date'], errors='coerce')
24
25    # 2. Crear una nueva columna que contenga el mes y el año
26    # Aquí se formatea como "YYYY-MM"
27    merged_df['crash_date'] = merged_df['crash_date'].dt.to_period('M')
28
29    # Add a new column 'city' with the value "New York"
30    merged_df['city'] = "New York"
31
32    merged_df = merged_df.sort_values(by='crash_date', ascending=True)
33
34    # Move the 'city' column to the beginning of the DataFrame
35    cols = ['city'] + [col for col in merged_df.columns if col != 'city']
36    merged_df = merged_df[cols]
```

Merge analysis:

Comparison between day and night accidents

Comparison between day and night accidents:

	sunrise_sunset
Day	31947
Night	15277

Relationship between weather conditions and accident severity

Relationship between climate and number of injured people:

weather_condition

Fair	17746
Cloudy	6142
Light Rain	1807
Mostly Cloudy	1760
Partly Cloudy	1310
Heavy Rain	293
Rain	261
Fog	149
Light Snow	128
Haze	64
Snow	15

Distribution by type of vehicle involved

Distribution by type of vehicle involved:

vehicle_type_code1

Sedan	22405
Station Wagon/Sport Utility Vehicle	16225
Taxi	1386
Bus	1099
Pick-up Truck	1003
...	
TRACTOR	1
Commercial	1
MTA bus	1
REFG	1
Van Camper	1

Distribution by time of day

2. Distribution by time of day:

	crash_time	count
0	00:00	690
828	14:00	457
888	15:00	440
1008	17:00	413
768	13:00	398
...
226	03:48	1
283	04:47	1
233	03:56	1
234	03:57	1
275	04:38	1

Most common contributing factors in serious accidents

Most common contributing factors in serious accidents:

contributing_factor_vehicle_1	
Driver Inattention/Distraction	5855
Unspecified	3572
Failure to Yield Right-of-Way	2820
Traffic Control Disregarded	1733
Following Too Closely	1218
Unsafe Speed	820
Passing or Lane Usage Improper	757
Turning Improperly	623
Other Vehicular	490
Pedestrian/Bicyclist/Other Pedestrian Error/Confusion	431
Driver Inexperience	366
Unsafe Lane Changing	334
Alcohol Involvement	317
View Obstructed/Limited	301

Accidents by (borough)

Accidents by (borough):

borough

Brooklyn - 11207.0 1427

Brooklyn - 11236.0 922

Brooklyn - 11234.0 831

Queens - 11434.0 771

Brooklyn - 11208.0 758

...

Manhattan - 10280.0 5

Manhattan - 10115.0 4

Manhattan - 10069.0 4

Queens - 11109.0 2

Manhattan - 10169.0 1

Correlation between type of vehicle and number of injured people

Correlation between type of vehicle and number of injured people:

vehicle_type_code1

Sedan 14633

Station Wagon/Sport Utility Vehicle 10300

Taxi 971

Bike 617

Pick-up Truck 536

...

MINIVAN 0

MINI BUS 0

Lunch Wagon 0

LOCOMOTIVE 0

van 0

Accidents by Wind Speed

Accidents by Wind Speed:

```
wind_speed_mph
0.000000      9099
3.000000      8589
5.000000      7806
6.000000      7099
7.000000      3737
7.681347      2513
8.000000      2403
9.000000      2011
10.000000     1356
13.000000       709
12.000000       685
15.000000       416
18.000000       308
16.000000       306
14.000000       187
```

Comparison between Injured Persons, Pedestrians, Cyclist and Motorist

```
Comparison between injured pedestrians, cyclists and motorcyclists:
{'Injured People': 29675, 'Injured Pedestrians': 219, 'Injured Cyclist': 4104, 'Injured Motorist': 23444}
```

Correlation between temperature and number of accidents

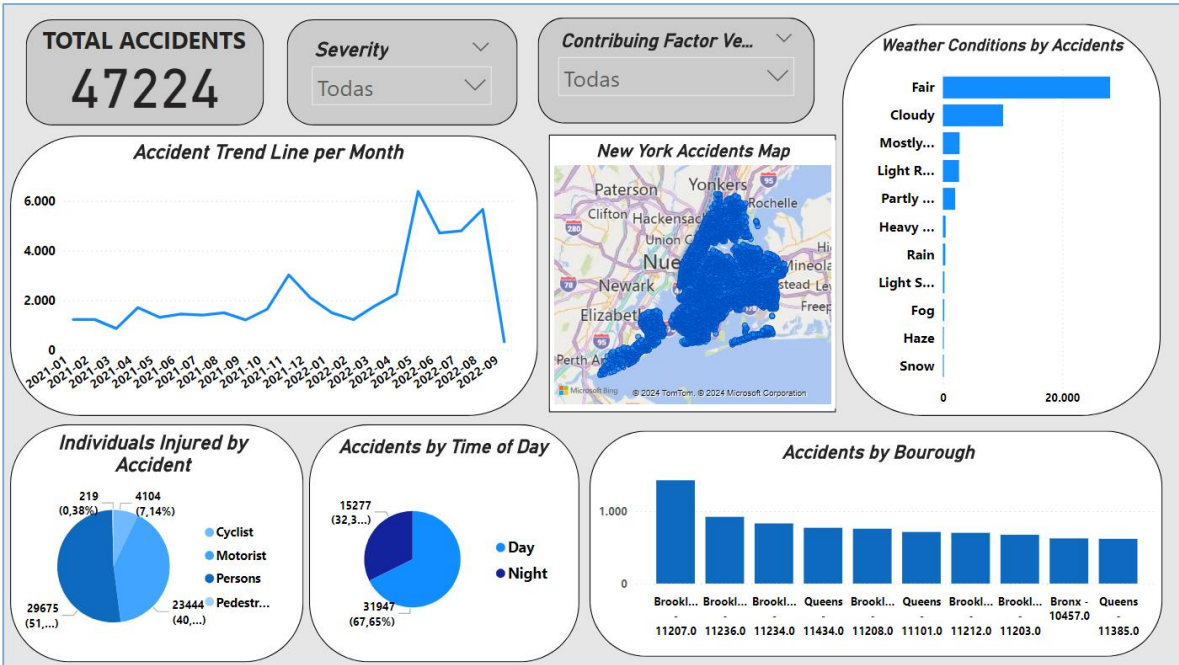
Correlation between type of vehicle and number of injured people:

```
vehicle_type_code1
Sedan                                14633
Station Wagon/Sport Utility Vehicle  10300
Taxi                                  971
Bike                                  617
Pick-up Truck                         536
...
MINIVAN                               0
MINI BUS                              0
Lunch Wagon                           0
LOCOMOTIVE                            0
van                                    0
```

Most common factors in accidents without injuries

Most common factors in accidents without injuries:	
contributing_factor_vehicle_1	
Driver Inattention/Distraction	6780
Unspecified	4674
Failure to Yield Right-of-Way	1879
Passing or Lane Usage Improper	1701
Following Too Closely	1582
Passing Too Closely	1239
Traffic Control Disregarded	1091
Turning Improperly	1040
Backing Unsafely	911
Unsafe Speed	789
Other Vehicular	690
Alcohol Involvement	591
Driver Inexperience	545
Unsafe Lane Changing	530
Reaction to Uninvolved Vehicle	323

Data merge dashboard:



3. Dimensional model:

The goal of this model is to organize data in a way that makes it easy to query and analyze. It is based on the star schema methodology, where a central fact table is surrounded by several dimension tables. In this case, the fact table is fact_accidents, and it is related to the dimension tables dim_time, dim_weather, dim_vehicle, and dim_location.

First we create the tables that we already mentioned with the sql that is inside the repository so that we can fill those tables with this merge.py code

dim_time table

- Purpose: Stores details related to the date and time of the accident.
- Content:
- date: Year and month of the accident (YYYY-MM format).
- day: Name of the day (e.g. "Monday").
- month: Name of the month (e.g. "January").
- year: Year of the accident.
- time: Specific time of the accident.

This table allows for temporal analysis, such as identifying patterns on specific days of the week, months, or years.

dim_climate table

- Purpose: Contains weather information relevant to the time of the accident.
- Contents:
- temperature_f: Temperature in degrees Fahrenheit.
- wind_chill_f: Wind chill.
- humidity_percent: Relative humidity in percentage.
- pressure_in: Atmospheric pressure in inches.
- visibility_mi: Visibility in miles.
- wind_direction: Wind direction.
- wind_speed_mph: Wind speed in miles per hour.
- precipitation_in: Precipitation in inches.

This table helps to understand the influence of weather on the occurrence of accidents.

dim_vehicle table

- Purpose: Describes the type of vehicle and contributing factors to the accident.
- Contents:
- vehicle_type_code1: Type of vehicle involved.
- contributing_factor_vehicle_1: Contributing factor to the vehicle-related accident (e.g. "Distracted Driving").

Allows for analysis based on vehicle types and common causes of accidents.

dim_location table

- Purpose: Contains geographic and location information for accidents.
- Content:
- city: City where the accident occurred (in this case, always "New York").
- borough: District or area of the city (e.g. "Manhattan").
- location: Coordinates or specific details of the accident location.

Facilitates the analysis of accidents by specific geographic locations.

fact_accidents table

- Purpose: This is the central table of the dimensional model that records the specific details of each accident.
- Content:
- Accident details such as date (crash_date), time (crash_time), number of people injured or killed, severity (severity), and weather conditions (weather_condition).
- Foreign keys that connect to the dimension tables:
 - dim_tiempo_id: Relates to the dim_tiempo table.
 - dim_climate_id: Relates to the dim_climate table.
 - dim_vehicle_id: Relates to the dim_vehicle table.
 - dim_location_id: Relates to the dim_location table.

Model Population Process

The code you provided performs the following steps to populate the dimensional model in PostgreSQL:

1. Connecting to the database using SQLAlchemy, with connection parameters stored in a .env file.
2. Loading the merged dataset (merged_data_cleaned.csv), which contains accident information along with climate and geographic data.
3. Converting and creating unique IDs for each dimension table (dim_weather, dim_climate, dim_vehicle, dim_location), ensuring that each row in these tables has a unique identifier.
4. Inserting data into the dimension tables using to_sql to send the data to the PostgreSQL database.
5. Assigning dimension IDs in the fact_accidents fact table by performing index-based merges to ensure foreign key integrity.
6. Filtering data to ensure consistency in the fact_accidents table, removing rows with null or inconsistent IDs.
7. Final insertion of records into the fact_accidents table.

```
src > load_dimensional_model.py > dimensional_model
5 import os
6
7 def dimensional_model():
8
9
10 # Cargar las variables de entorno del archivo .env
11 load_dotenv()
12
13 # Obtener las variables de entorno
14 DB_HOST = os.getenv('DB_HOST')
15 DB_PORT = os.getenv('DB_PORT')
16 DB_USER = os.getenv('DB_USER')
17 DB_PASS = os.getenv('DB_PASS')
18 DB_NAME = os.getenv('DB_NAME')
19
20 # Configurar la conexión a la base de datos
21 DATABASE_URL = f'postgresql://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}'
22 engine = create_engine(DATABASE_URL)
23
24 # Cargar el archivo CSV de accidentes
25 merged_df = pd.read_csv('data/merged_data_cleaned.csv', encoding='utf-8')
26
27
28 # Convertir crash_date y crash_time a datetime
29 merged_df['crash_datetime'] = pd.to_datetime(merged_df['crash_date'] + ' ' + merged_df['crash_time'], errors='coerce')
30
31 # Verificar si hay errores de conversión de fecha
32 if merged_df['crash_datetime'].isnull().any():
33     print("Advertencia: Algunas fechas no se pudieron convertir.")
34
35 # Extraer información de tiempo
36 merged_df['dim_tiempo_id'] = range(1, len(merged_df) + 1)
37 merged_df['fecha'] = merged_df['crash_datetime'].dt.strftime('%Y-%m') # Convertir a formato de cadena 'YYYY-MM'
38 merged_df['hora'] = merged_df['crash_datetime'].dt.strftime('%H:%M') # Formato HH:MM
39 merged_df['dia'] = merged_df['crash_datetime'].dt.day_name()
40 merged_df['mes'] = merged_df['crash_datetime'].dt.month_name()
41 merged_df['año'] = merged_df['crash_datetime'].dt.year
42
```

Rationale for the Dimensional Model

This model is designed to improve the efficiency of querying and analyzing accident data:

- **Fast query:** By segmenting the data into specific dimensions, faster queries and analysis can be performed for common questions, such as "In which months do more accidents occur?" or "What type of weather is most associated with serious accidents?".
- **Flexibility:** Allows new dimensions to be added in the future without affecting the core structure of the model.
- **Clarity in analysis:** By separating data into dimension tables, redundancy is minimized and clarity in the presentation of information is improved.
- **Storage optimization:** Dimension tables store information that is repeated in multiple records in the fact table, saving space and making maintenance easier.

4. Kafka Implementation

4.1. Configuration with Docker

To manage Apache Kafka and Zookeeper, Docker Compose was used. The configuration included:

4.2. Productor de Kafka

- The Kafka Producer For the Kafka implementation, we use the metric of total injured people per month from data collected in our traffic accident database.

```
5 import json
6 import time # Importa el módulo time para usar sleep
7 from kafka import KafkaProducer
8
9 def kafka_producer():
10     # Cargar las variables de entorno del archivo .env
11     load_dotenv()
12
13     # Obtener las variables de entorno
14     DB_HOST = os.getenv('DB_HOST')
15     DB_PORT = os.getenv('DB_PORT')
16     DB_USER = os.getenv('DB_USER')
17     DB_PASS = os.getenv('DB_PASS')
18     DB_NAME = os.getenv('DB_NAME')
19
20     # Configurar la conexión a la base de datos
21     DATABASE_URL = f'postgresql://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}'
22     engine = create_engine(DATABASE_URL)
23
24     # Conexión a PostgreSQL para obtener el número total de personas heridas
25     try:
26         # Consulta SQL para obtener el total de personas heridas por día
27         query = """
28         SELECT
29             crash_date AS date,
30             SUM(number_of_persons_injured +
31                 number_of_pedestrians_injured +
32                 number_of_cyclist_injured +
33                 number_of_motorist_injured) AS total_injured
34         FROM
35             fact_accidents
36         GROUP BY
37             crash_date
38         ORDER BY
39             crash_date;
40         """
41
42         # Ejecutar la consulta usando pandas para obtener el resultado en un DataFrame
```

The Kafka producer takes daily traffic accident data from our PostgreSQL database and streams the total injured people metric to the consumer. Each message contains the date of the accident and the total number of injured people on that day.

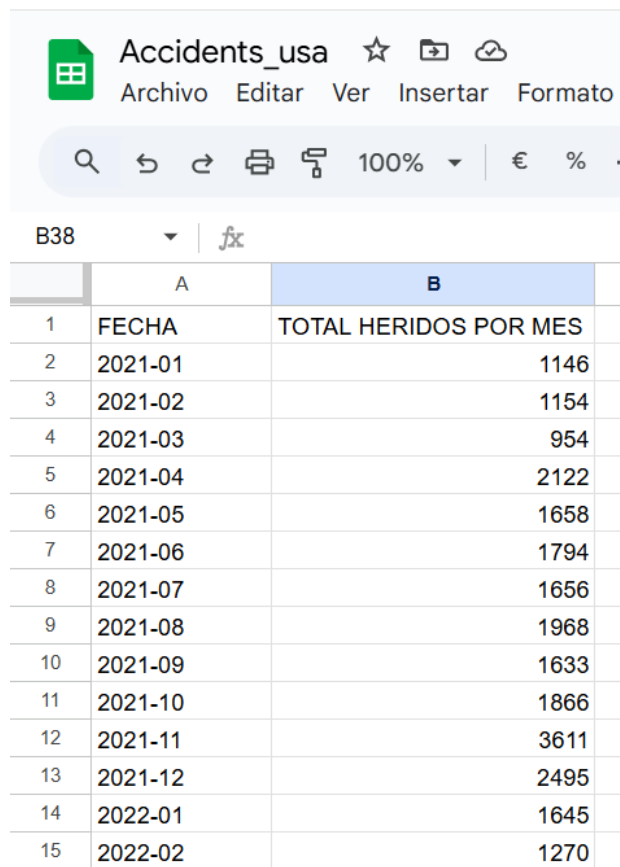
```
● $ python src/kafka_producer.py
Enviado a Kafka: {'date': '2021-01', 'total_injured': 1146}
Enviado a Kafka: {'date': '2021-02', 'total_injured': 1154}
Enviado a Kafka: {'date': '2021-03', 'total_injured': 954}
Enviado a Kafka: {'date': '2021-04', 'total_injured': 2122}
Enviado a Kafka: {'date': '2021-05', 'total_injured': 1658}
Enviado a Kafka: {'date': '2021-06', 'total_injured': 1794}
Enviado a Kafka: {'date': '2021-07', 'total_injured': 1656}
Enviado a Kafka: {'date': '2021-08', 'total_injured': 1968}
Enviado a Kafka: {'date': '2021-09', 'total_injured': 1633}
Enviado a Kafka: {'date': '2021-10', 'total_injured': 1866}
Enviado a Kafka: {'date': '2021-11', 'total_injured': 3611}
Enviado a Kafka: {'date': '2021-12', 'total_injured': 2495}
Enviado a Kafka: {'date': '2022-01', 'total_injured': 1645}
Enviado a Kafka: {'date': '2022-02', 'total_injured': 1270}
Enviado a Kafka: {'date': '2022-03', 'total_injured': 1945}
Enviado a Kafka: {'date': '2022-04', 'total_injured': 2907}
```

4.3. Kafka Consumer

- The Kafka Consumer The Kafka Consumer receives real-time accident data and sends it to a spreadsheet in Google Sheets. This data, organized by date, contains the metric of the total number of injured people.
- Each consumer message has a time.sleep

```
src > kafka_consumer.py > kafka_consumer_to_google_sheets
5 from google.oauth2.service_account import Credentials
6 from googleapiclient.errors import HttpError
7
8 def kafka_consumer_to_google_sheets():
9     # Configuración del consumidor de Kafka
10    consumer = KafkaConsumer(
11        'injuries_by_day', # Nombre del topic de Kafka
12        bootstrap_servers='localhost:9092', # Servidor de Kafka
13        value_deserializer=lambda x: json.loads(x.decode('utf-8')) # Deserialización de mensajes
14    )
15
16    # Ruta al archivo de credenciales JSON de la cuenta de servicio de Google
17    creds = Credentials.from_service_account_file(
18        'accidents_usa.json', # Ruta del archivo de credenciales JSON
19        scopes=["https://www.googleapis.com/auth/spreadsheets"]
20    )
21
22    # ID de la hoja de cálculo de Google Sheets y el rango donde escribir los datos
23    SPREADSHEET_ID = '1Dflw-mewAhm6Nqa60dxc9BgLTklrMrbnyWf485XW0FE' # ID de la hoja de cálculo
24    RANGE_NAME = 'Hoja1!A:A' # Columna A para referencia
25
26    # Crear el servicio de Google Sheets
27    service = build('sheets', 'v4', credentials=creds)
28    sheet = service.spreadsheets()
29
30    # Función para obtener la última fila de la hoja de cálculo
31    def get_last_row():
32        try:
33            result = sheet.values().get(
34                spreadsheetId=SPREADSHEET_ID,
35                range=RANGE_NAME
36            ).execute()
37            values = result.get('values', [])
38            return len(values) + 1 # La próxima fila disponible
39        except HttpError as err:
40            print(f'Error al obtener la última fila: {err}')
41            return 2 # Retorna la fila 2 en caso de error
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
259
```

Here the Google Sheets table is filled in



The screenshot shows a Google Sheets interface with a spreadsheet titled 'Accidents_usa'. The spreadsheet has two columns: 'FECHA' (Date) and 'TOTAL HERIDOS POR MES' (Total Injured per Month). The data is organized by month from January 2021 to February 2022. The values represent the number of injured persons for each month.

	A	B
1	FECHA	TOTAL HERIDOS POR MES
2	2021-01	1146
3	2021-02	1154
4	2021-03	954
5	2021-04	2122
6	2021-05	1658
7	2021-06	1794
8	2021-07	1656
9	2021-08	1968
10	2021-09	1633
11	2021-10	1866
12	2021-11	3611
13	2021-12	2495
14	2022-01	1645
15	2022-02	1270

To implement data integration from Kafka to Google Sheets, a process was followed that involved setting up credentials in Google Cloud, authorizing access to the Google Sheets spreadsheet, and creating the script that consumes the data from Kafka to insert it into Google Sheets.

Setting up the Google Sheets API:

- In the Google Cloud console, enable the Google Sheets API.
- Create a service account in Google Cloud and download the JSON credentials file.
- Share the Google Sheets spreadsheet with the email generated by the service account (e.g. accidents@accidents-usa.iam.gserviceaccount.com).

Preparing the Kafka Consumer:

- Set up a Kafka consumer in Python to connect to the specific topic (in this case, injuries_by_day).
- Deserialize the messages received from Kafka using json.

Authentication in Google Sheets:

- Use the credentials from the JSON file to authenticate and connect to the Google Sheets API with googleapiclient.
- Specify the spreadsheet ID and cell range to write data to.

Google Sheets Update:

- Read the last available row in Google Sheets to avoid overwrites.
- Insert data received from Kafka into the cells of the specified sheet.

Data Consumption and Update:

- Consume Kafka messages in a continuous loop.
- Send each received data to Google Sheets, updating the sheet in real time.
- Include a 3-second wait between messages to avoid overload.

5. Real-Time Visualization with Looker Studio

The dashboard in Looker Studio is configured to display updated information in real time. Below is a brief description of the process and the tools used:

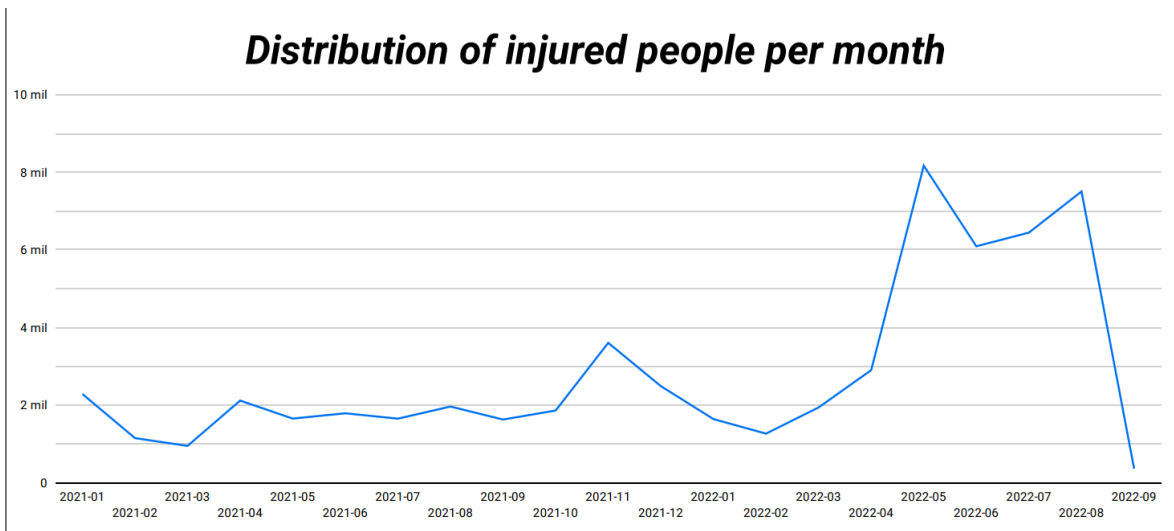
1. Data Connection:

- The Google Sheets file is used as a data source for the dashboard.
- Whenever data is updated in Google Sheets (via the Kafka consumer), Looker Studio is automatically refreshed to reflect the latest information.

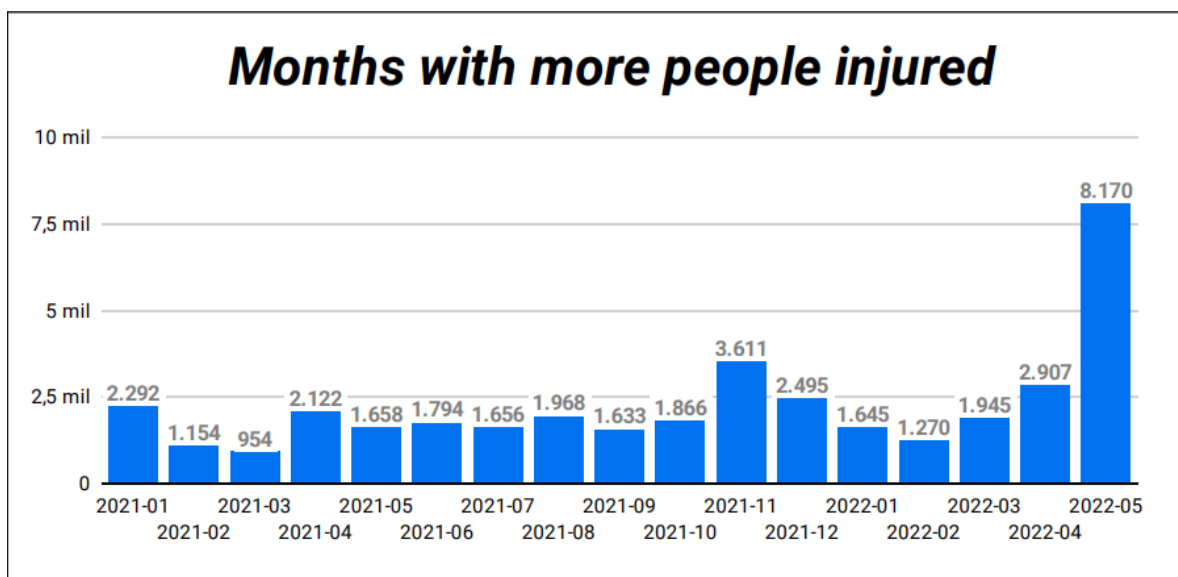
2. Dashboard Setup:

Several key visualizations are created to display accident and injury metrics:

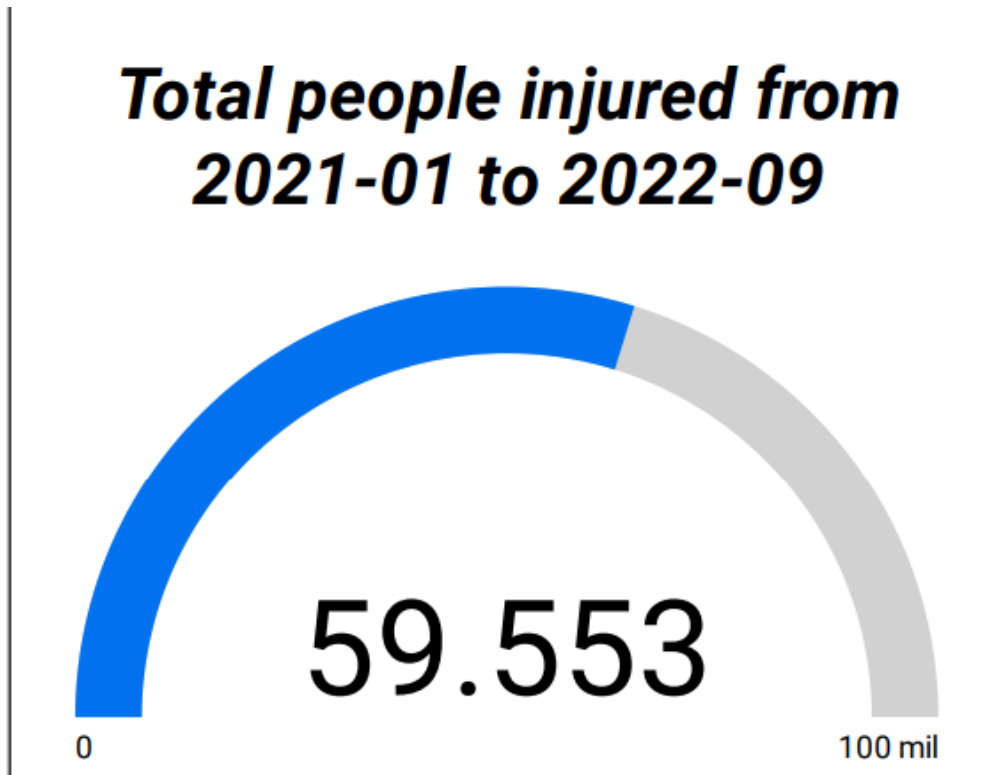
Line Chart: Visualizes injury and accident trends over time, allowing you to identify spikes and patterns.



Bar Chart: Comparison of categorical data, such as the number of accidents by day of the week or the distribution of injuries.



Indicator: Displays critical metrics, such as the total number of injuries or fatalities, providing a clear view of current values.



6. DAG Implementation in Airflow

For the orchestration of the ETL process, Apache Airflow was used on a Linux Ubuntu virtual machine. The tasks of the created DAG are detailed below:

DAG Tasks

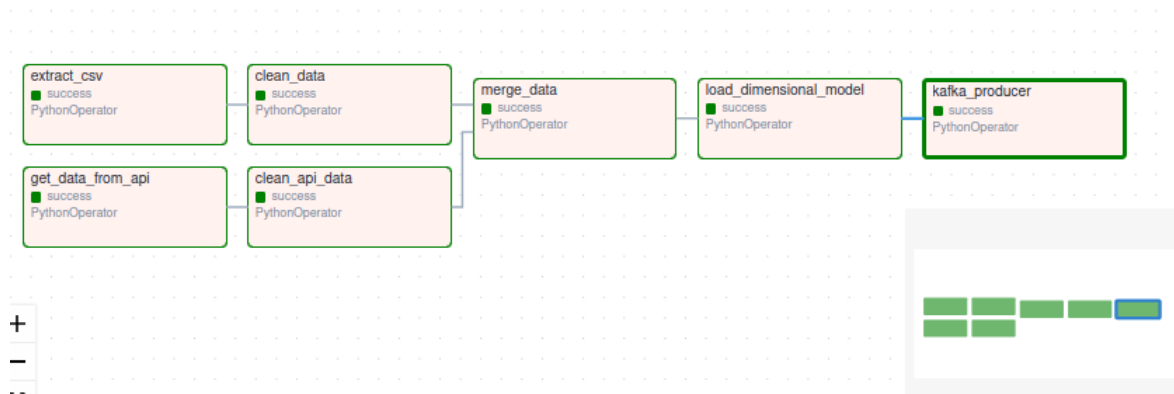
1. Data Extraction from the API
2. Data Cleaning from the API
3. Data Extraction from the CSV
4. Data Cleaning from the CSV
5. Data Unification
6. Loading into the Dimensional Model
7. Data Streaming with Kafka

```

dags > etl_dag.py > ...
willyb, 9 hours ago | 2 authors (willyb and one other)
1  import sys
2  import os
3
4  # Agregar la ruta al paquete src
5  sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '../src')))
6
7  from airflow import DAG
8  from airflow.operators.python import PythonOperator
9  from datetime import datetime, timedelta
10
11  # Importar funciones desde los módulos en src
12  from API.connection import get_data_from_api
13  from API.cleaning import clean_api_data
14  from extract import extract_data
15  from data_cleaning import clean_data
16  from merge_data import merge_data
17  from load_dimensional_model import dimensional_model
18  from kafka.producer import kafka_producer
19
20
21  # Definir el DAG
22  default_args = {
23      'owner': 'airflow',
24      'depends_on_past': False,
25      'start_date': datetime(2024, 11, 13),
26      'email_on_failure': False,
27      'email_on_retry': False,
28      'retries': 1,
29      'retry_delay': timedelta(minutes=1)
30  }
31
32  with DAG(
33      'accidents_etl_pipeline',
34      default_args=default_args,
35      description='ETL Pipeline for Accidents Data',
36      schedule_interval='@daily',
37  ) as dag:

```

In the Ariflow interface it would look like this after having completed all the tasks:



7. Conclusion

This project provides a comprehensive solution for managing and analyzing traffic accident data, integrating multiple technologies into a single workflow. The combination of Airflow, Kafka, PostgreSQL and Looker Studio allowed not only to store and analyze data, but also to visualize metrics in real time, offering valuable insights for decision making.

Technologies used:

- **Airflow**
- **Kafka**
- **Docker**
- **PostgreSQL**
- **Google Sheets y Looker Studio**
- **Power BI**
- **Github**
- **Jupyter notebooks**
- **Kaggle**
- **Pandas**