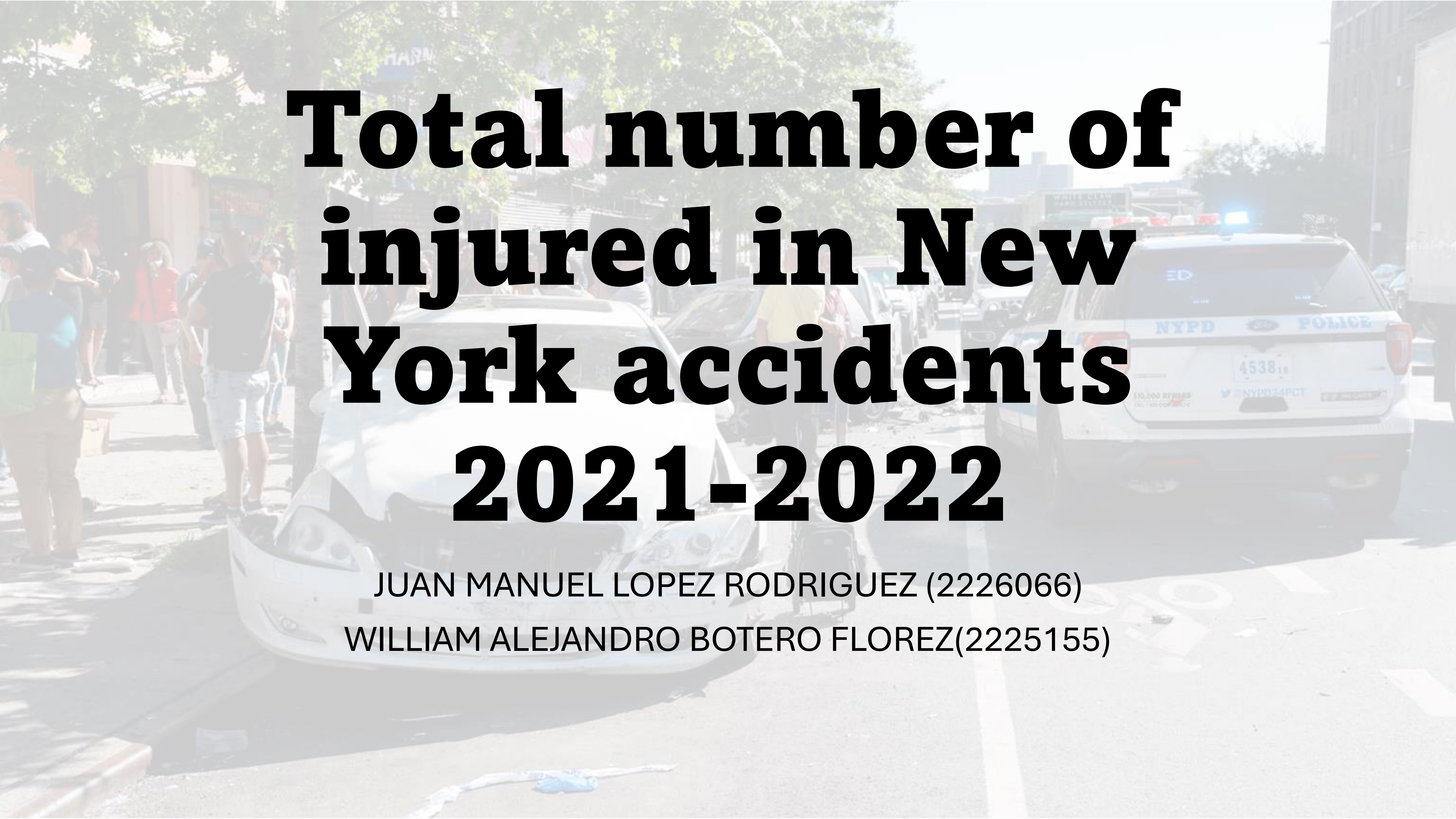
- **1.35 million annual deaths:** According to the World Health Organization (WHO), traffic accidents cause approximately 1.35 million deaths worldwide each year.
- **Vulnerable pedestrians and cyclists:** In many cities, pedestrians and cyclists account for 50% or more of traffic fatalities.
- **Drunk driving:** Drunk driving is a key factor in around 27% of traffic-related deaths globally.
- **Excessive speed:** It is estimated that speed is a determining factor in one in three serious or fatal traffic accidents.



Total number of injured in New York accidents 2021-2022

JUAN MANUEL LOPEZ RODRIGUEZ (2226066)
WILLIAM ALEJANDRO BOTERO FLOREZ(2225155)

CONTENT TABLE

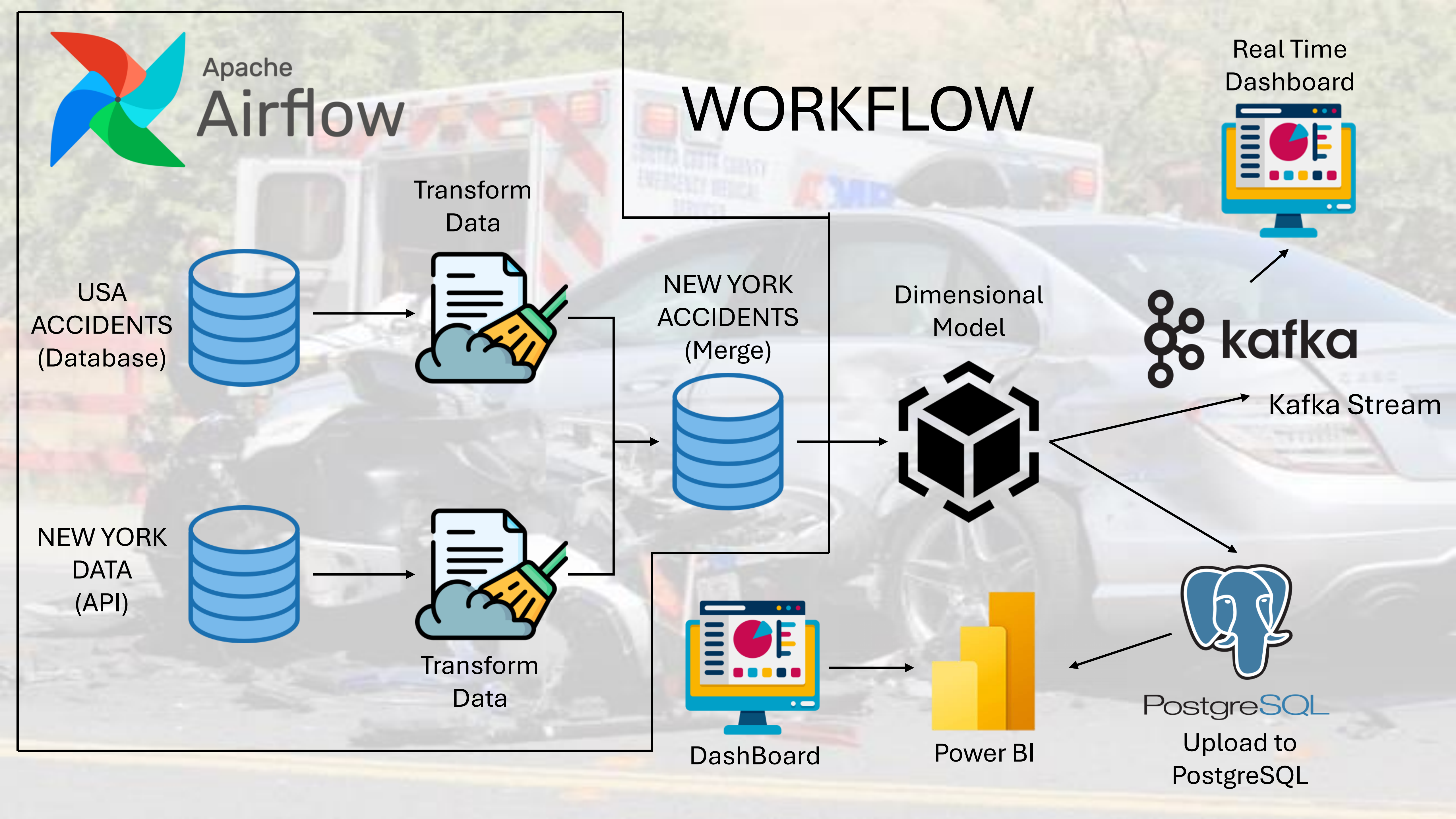
- Project Objective
- Tools Used
- WorkFlow
- Accidents in USA (DataBase)
- Data Cleaning (Database)
- New York Data (Extract API)
- Data Cleaning (API)
- Merge Between Databases
- Data Cleaning (Merged Data)
- Final Data
- Data Analysis
- Airflow
- Dimensional Model
- Data Dashboard
- Kakfa implementation
- Real time dashboard

PROJECT OBJECTIVE

Carry out an exhaustive analysis of the databases provided, which contain more complete and structured information. The analysis will seek to visualize the distribution of the data to facilitate a deeper understanding of the context and thus identify relevant patterns that guide decision making.

USED TOOLS

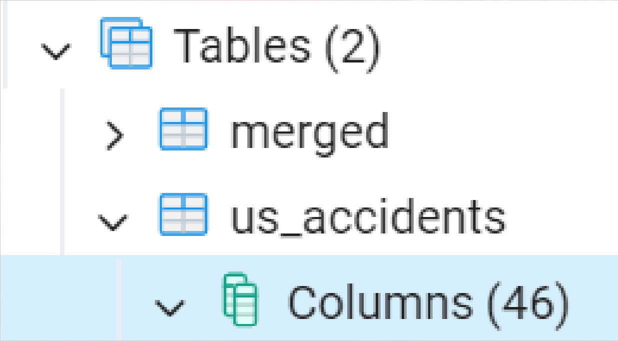
- PYTHON
- PANDAS
- AIRFLOW
- POSTGRES
- KAGGLE
- POWER BI
- KAFKA
- LOOKER


WORKFLOW
Apache Airflow
USA ACCIDENTS (Database)
NEW YORK DATA (API)
Transform Data
Transform Data
NEW YORK ACCIDENTS (Merge)
Dimensional Model
kafka
Kafka Stream
Real Time Dashboard
PostgreSQL
Upload to PostgreSQL
Power BI
DashBoard

ACCIDENTS IN USA

To obtain the data, we extracted it from the Kaggle page, on the “US Accidents (2016-2023)” dataset, which gives us information on traffic accidents that occurred in the United States between 2016 and 2023.

Once the data has been downloaded, we upload it to the PostreSQL database, which will be the connection between the workspace and the data



- Once the database is uploaded to PostGres, we import the data into our work environment, to begin carrying out everything related to the ETL of the data.

```
# Establish a connection and create a cursor
conn, cursor = establecer_conexion()  # Function to establish the database connection

# SQL query to select all data from the 'us_accidents' table
query = "SELECT * FROM us_accidents"

# Read the data into a pandas DataFrame
df = pd.read_sql_query(query, conn)
```

```
Conexion exitosa a la base de datos
```

```
Index(['severity', 'start_time', 'end_time', 'start_lat', 'start_lng',
       'distance_mi', 'street', 'city', 'county', 'state', 'zipcode',
       'timezone', 'airport_code', 'weather_timestamp', 'temperature_f',
       'wind_chill_f', 'humidity_percent', 'pressure_in', 'visibility_mi',
       'wind_direction', 'wind_speed_mph', 'precipitation_in',
       'weather_condition', 'amenity', 'bump', 'crossing', 'give_way',
       'junction', 'no_exit', 'railway', 'roundabout', 'station', 'stop',
       'traffic_calming', 'traffic_signal', 'turning_loop', 'sunrise_sunset'],
```

DATA CLEANING (DATABASE)

- To clean the data, what we did was identify null data, repeated patterns, columns that did not provide relevant information that can be deleted and fill in null data with a data imputation.

```python
# Columns to drop
columns_to_drop = ['id', 'source', 'country', 'description', 'end_lat', 'end_lng',
                   'civil_twilight', 'nautical_twilight', 'astronomical_twilight']

# Drop the specified columns
df_cleaned = df.drop(columns=columns_to_drop)

# Impute missing values in numerical columns with the mean
df_cleaned['temperature_f'].fillna(df_cleaned['temperature_f'].mean(), inplace=True)

# Impute missing values in categorical columns with the mode (most frequent value)
df_cleaned['weather_condition'].fillna(df_cleaned['weather_condition'].mode()[0], inplace=True)

# Impute missing values in multiple numerical columns with the mean
num_cols = ['wind_chill_f', 'humidity_percent', 'pressure_in', 'visibility_mi', 'wind_speed_mph', 'precipitation_in
df_cleaned[num_cols] = df_cleaned[num_cols].apply(lambda col: col.fillna(col.mean()))

# Impute the 'wind_direction' column with the most frequent value (mode)
df_cleaned['wind_direction'] = df_cleaned['wind_direction'].fillna(df_cleaned['wind_direction'].mode()[0])

# Impute 'weather_timestamp' with the previous value (forward fill) for missing timestamps
df_cleaned['weather_timestamp'] = df_cleaned['weather_timestamp'].fillna(method='ffill')

# Remove rows containing any remaining missing values
df_cleaned.dropna(inplace=True)
```

NEW YORK DATA

- Taking a smaller part of the country, we decided to focus on only accidents that occur in New York City, so we took the data from the following API: https://data.cityofnewyork.us/resource/h9gi-nx95.json
- The API has a restriction of 1000 data, so, when consuming it, we decided to generate some parameters, so that, instead of bringing only 1000 data, it would bring us 200,000, between the years 2016-2022, since until this year has updated information

```python
# URL of the dataset (API endpoint)
url = "https://data.cityofnewyork.us/resource/h9gi-nx95.json"

# Parameters to limit the response to 200,000 records
params = {
    "$limit": 200000
}

# Send a GET request to the API
response = requests.get(url, params=params)
```

- DATA API:

```
Index(['crash_date', 'crash_time', 'on_street_name', 'off_street_name',
       'number_of_persons_injured', 'number_of_persons_killed',
       'number_of_pedestrians_injured', 'number_of_pedestrians_killed',
       'number_of_cyclist_injured', 'number_of_cyclist_killed',
       'number_of_motorist_injured', 'number_of_motorist_killed',
       'contributing_factor_vehicle_1', 'contributing_factor_vehicle_2',
       'collision_id', 'vehicle_type_code1', 'vehicle_type_code2', 'borough',
       'zip_code', 'latitude', 'longitude', 'location', 'cross_street_name',
       'contributing_factor_vehicle_3', 'vehicle_type_code_3',
       'contributing_factor_vehicle_4', 'vehicle_type_code_4',
       'contributing_factor_vehicle_5', 'vehicle_type_code_5'],
      dtype='object')
```

- These columns give us a preview of what data we can use to perform the merge with our database, and thus, have new, more complete data.

DATA CLEANING (API)

```python
# Get the absolute path of the file
file_path = os.path.abspath(os.path.join('../data/API_data.csv'))

# Load the CSV file using pandas
data = pd.read_csv(file_path)

# Set pandas to display all columns
pd.set_option('display.max_columns', None)

# 2. Convert `crash_date` and `crash_time` to datetime format
# Handle errors by setting invalid parsing as NaT (Not a Time)
data['crash_date'] = pd.to_datetime(data['crash_date'], errors='coerce')
data['crash_time'] = pd.to_datetime(data['crash_time'], format='%H:%M', errors='coerce')

# 3. Fix inconsistent values (e.g., remove whitespace or correct capitalization in the `borough` column)
data['borough'] = data['borough'].str.strip().str.title()

# Filter data to keep only rows with valid crash dates and from the year 2021 or later
data = data[data['crash_date'].notna() & (data['crash_date'].dt.year >= 2021)]

# 4. Remove duplicates based on the `collision_id` column (assuming it's unique for each accident)
data = data.drop_duplicates(subset='collision_id')

# Convert `crash_date` to just the date (drop the time part)
data['crash_date'] = data['crash_date'].dt.date

# Drop unnecessary columns
data = data.drop(['vehicle_type_code_5', 'contributing_factor_vehicle_5',
                  'vehicle_type_code_4', 'contributing_factor_vehicle_4',
                  'vehicle_type_code_3', 'contributing_factor_vehicle_3',
                  'cross_street_name'], axis=1)

print("FILTERED AND CLEANED DATA: \n")

# Drop rows with any missing values
data = data.dropna()

# Add a `city` column with the value "New York"
data['city'] = "New York"
```

- To clean the API, we did a process very similar to the previous one, in addition to standardizing the data, choosing the correct format of the columns related to dates, saving the data that was recorded from the year 2021, since in In previous years there were very few records that generated atypical data and finally, we created a column called city, in which all the rows were going to be equal to New York, since this was going to facilitate the process of do the Merge between the 2 databases.

DATABASES MERGE

- To mix both databases, what we did was do it using the date of the event, where the day and time where the accident occurred will match, in addition a filter was applied where it would only bring us the accidents that occurred in New York, mainly in the database saved in PostgreSQL, to have more matches and thus make the mix.

```python
# Load the CSV files
API_merge = pd.read_csv('../data/API_data_Cleaned.csv')
db_merge = pd.read_csv('../data/us_accidents_cleaned.csv')

# Convert date columns to datetime format and use only the date
API_merge['crash_date'] = pd.to_datetime(API_merge['crash_date']).dt.date  # Use only the date
db_merge['start_time'] = pd.to_datetime(db_merge['start_time']).dt.date  # Use only the date

# Filter both datasets for rows where the city is 'New York'
api_data_ny = API_merge[API_merge['city'] == 'New York']
us_accidents_ny = db_merge[db_merge['city'] == 'New York']

# Merge the two datasets based on the date (inner join)
merged_df = pd.merge(api_data_ny, us_accidents_ny, left_on='crash_date', right_on='start_time', how='inner')
```

DATA CLEANING (MERGED DATA)

- To clean the data once the merge was done, repeated patterns were identified, we eliminated unimportant columns, we eliminated repeated columns that came out of the merge, conversion of date formats, conversion of floating columns to integers, mixing between columns for more information. completeness and data standardization.

```
# 3. Borrar 'Colishion_id'
merged_df.drop(columns=['collision_id'], inplace=True)

# 4. Borrar 'Factor contribuyente 2'
merged_df.drop(columns=['contributing_factor_vehicle_2'], inplace=True)

# 5. Borrar 'vehicle_type_code2'
merged_df.drop(columns=['vehicle_type_code2'], inplace=True)

# 6. Mezclar 'codigo postal' con 'distrito'
# Supongamos que 'codigo_postal' y 'distrito' son las columnas en merged_clean
merged_df['borough'] = merged_df['borough'] + ' - ' + merged_df['zip_code'].astype(str)


# 8. Borrar 'start time' y 'end time'
merged_df.drop(columns=['start_time', 'end_time'], inplace=True)

# 9. Borrar 'start latitud' y 'end latitud'
merged_df.drop(columns=['start_lat', 'start_lng'], inplace=True)

# 10. Borrar 'distancia en millas'
merged_df.drop(columns=['distance_mi'], inplace=True)

# 11. Borrar 'county'
merged_df.drop(columns=['county'], inplace=True)

# 12. Mezclar 'state' con 'city'
merged_df['city'] = merged_df['city'] + ', ' + merged_df['state']
```

FINAL DATA

- After carrying out all the necessary cleaning to merge the databases, we are left with the final database which tells us only about Accidents in the city of New York, information to which we will do everything related to the analysis.

```
Index(['id', 'city', 'crash_date', 'crash_time', 'on_street_name',
       'off_street_name', 'number_of_persons_injured',
       'number_of_persons_killed', 'number_of_pedestrians_injured',
       'number_of_pedestrians_killed', 'number_of_cyclist_injured',
       'number_of_cyclist_killed', 'number_of_motorist_injured',
       'number_of_motorist_killed', 'contributing_factor_vehicle_1',
       'vehicle_type_code1', 'borough', 'latitude', 'longitude', 'location',
       'severity', 'street', 'timezone', 'temperature_f', 'wind_chill_f',
       'humidity_percent', 'pressure_in', 'visibility_mi', 'wind_direction',
       'wind_speed_mph', 'precipitation_in', 'weather_condition',
       'sunrise_sunset'],
      dtype='object')
```

DATA ANALYSIS

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
```

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 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
```

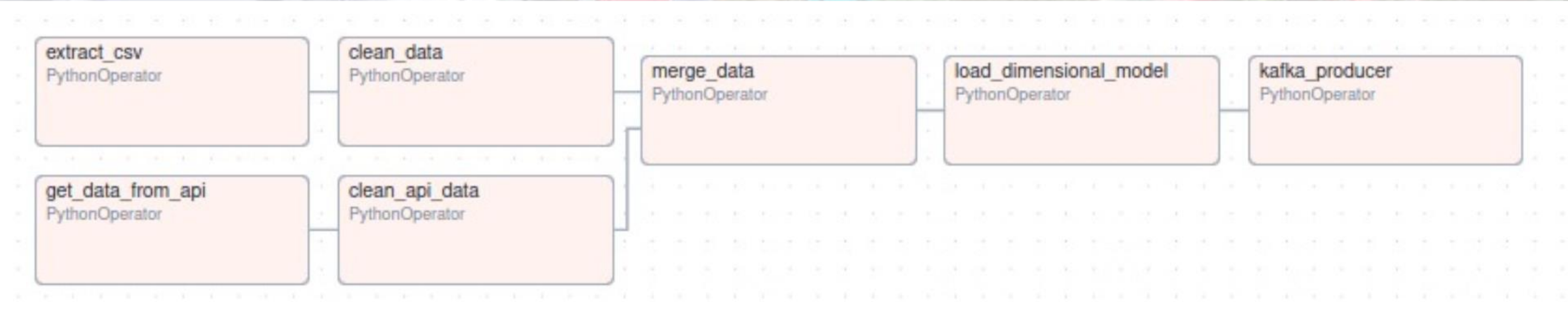
Correlation between temperature and number of accidents

```
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
```

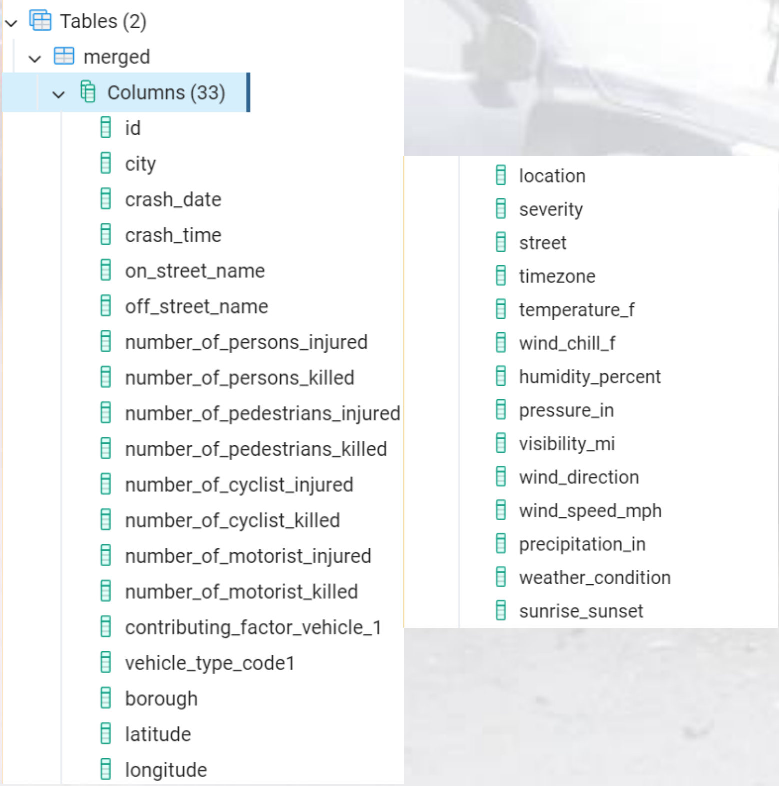
Most common factors in accidents without injuries

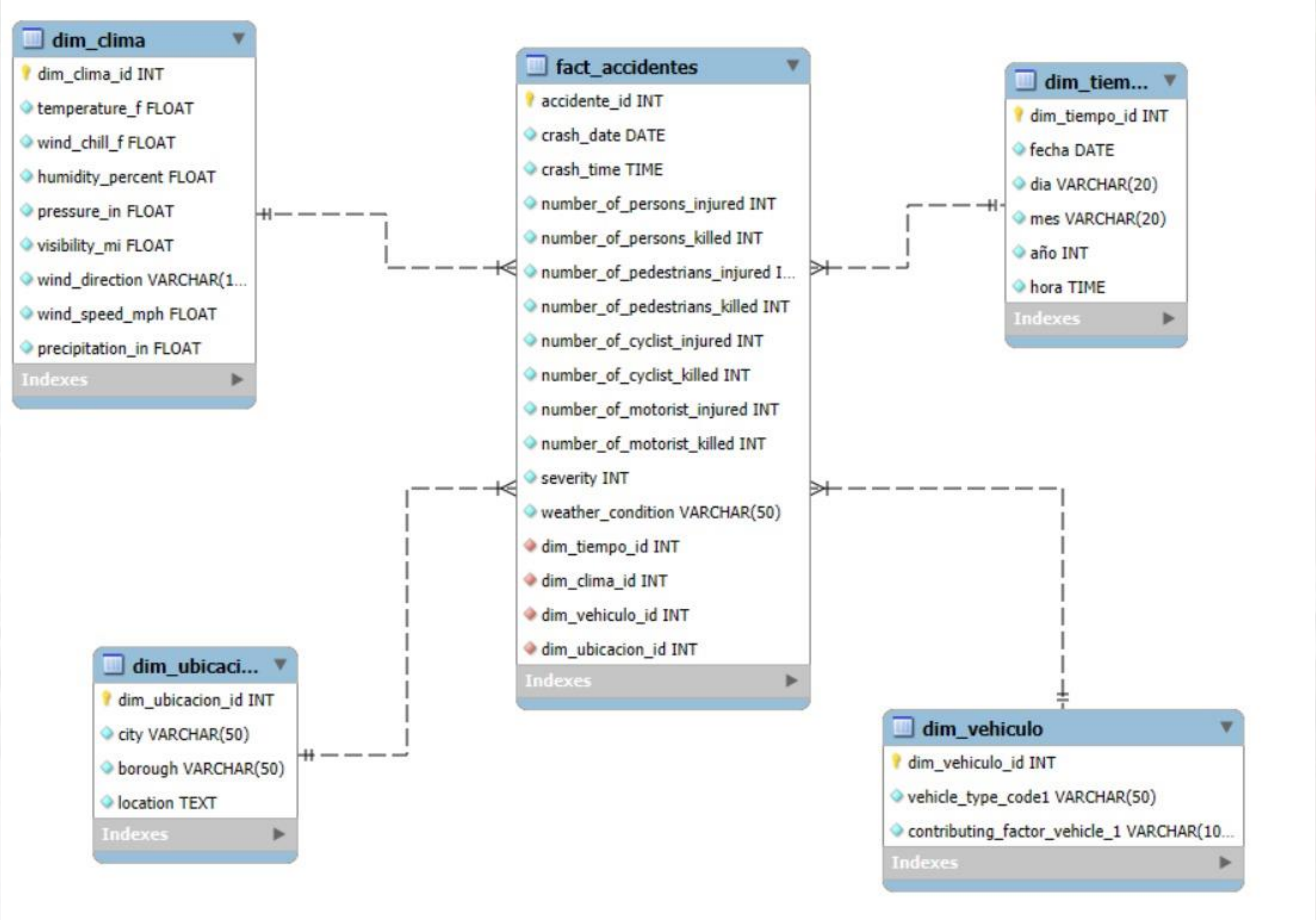
```
Most common factors in accidents without injuries:
contributing_factor_vehicle_1
Driver Inattention/Distraction                          6780
Unspecified                                             4674
Failure to Yield Right-of-Way                           1879
Passing or Lane Usage Improper                          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
```

WORKFLOW DE AIRFLOW

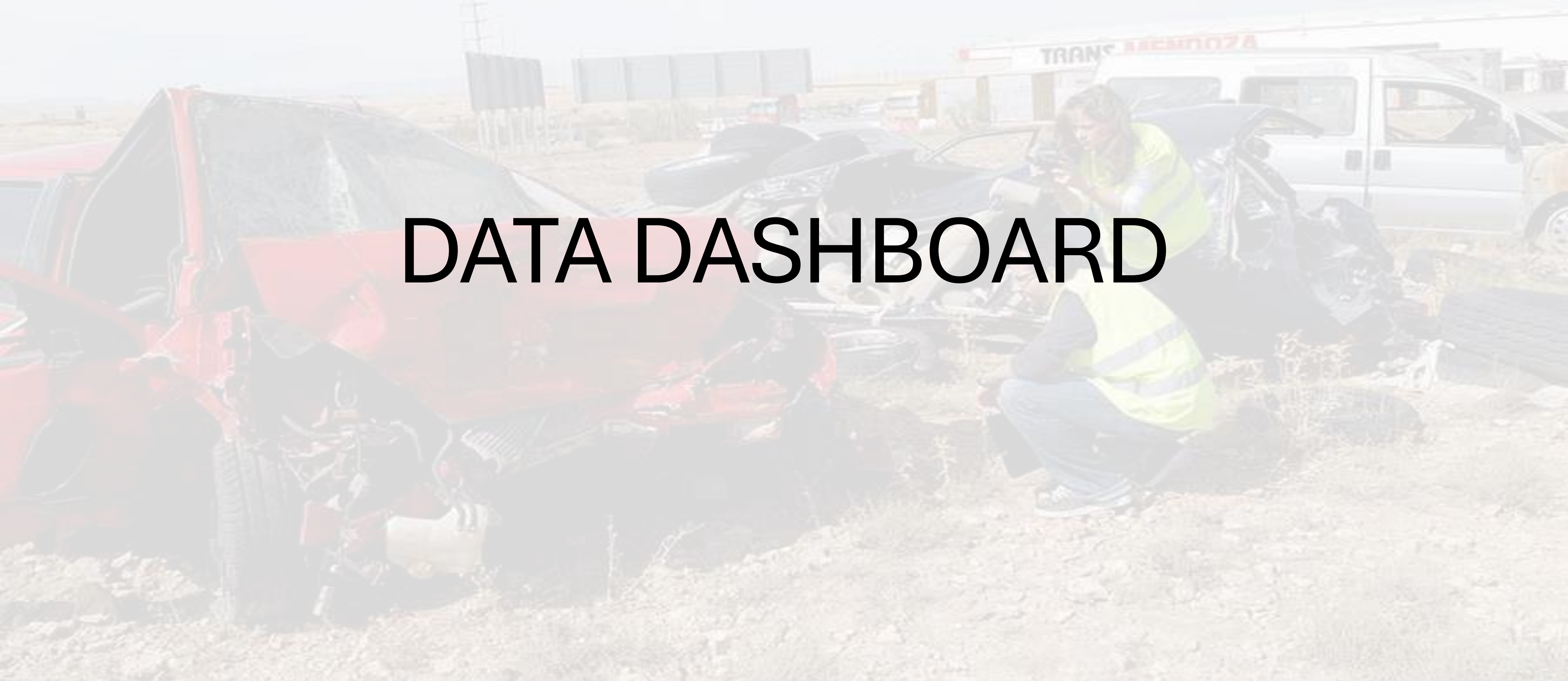


DIMENSIONAL MODEL





DATA DASHBOARD



Kafka Implementation for Accident Analysis

- For the Kafka implementation, we used the metric of total people injured per day from data collected in our traffic accident database.

```
# Consulta SQL para obtener el total de personas heridas por día
query = """
SELECT
    crash_date AS date,
    SUM(number_of_persons_injured +
        number_of_pedestrians_injured +
        number_of_cyclist_injured +
        number_of_motorist_injured) AS total_injured
FROM
    fact_accidents
GROUP BY
    crash_date
ORDER BY
    crash_date;
"""
```

How we implement it?

"The Kafka producer takes daily traffic accident data from our PostgreSQL database and transmits the metric of total people injured to the consumer. Each message contains the date of the accident and the total number of people injured 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}
```

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

Accidents_usa

Archivo Editar Ver Insertar Formato

100% € %

B38 fx

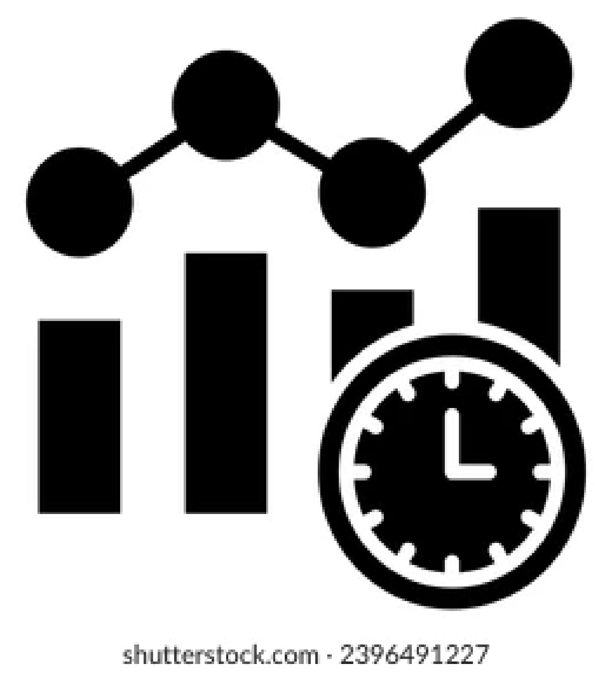
	A	B
1	FECHA	TOTAL HERIDOS POR MES
2	2021-01	1146
3	2021-02	1154
4	2021-03	954
5	2021-04	2122
6	2021-05	1658
7	2021-06	1794
8	2021-07	1656
9	2021-08	1968
10	2021-09	1633
11	2021-10	1866
12	2021-11	3611
13	2021-12	2495
14	2022-01	1645
15	2022-02	1270

```
$ python src/kafka_consumer.py
Mensaje recibido: {'date': '2021-01', 'total_injured': 1146}
Datos actualizados en Google Sheets: {'spreadsheetId': '1Dflw-mewAhm6Nqa60dxc9BgLTklrMrbnyWf485XWOFE', 'updatedRange': 'Hoja1!A2:B2',
ws': 1, 'updatedColumns': 2, 'updatedCells': 2}
Mensaje recibido: {'date': '2021-02', 'total_injured': 1154}
Datos actualizados en Google Sheets: {'spreadsheetId': '1Dflw-mewAhm6Nqa60dxc9BgLTklrMrbnyWf485XWOFE', 'updatedRange': 'Hoja1!A3:B3',
ws': 1, 'updatedColumns': 2, 'updatedCells': 2}
Mensaje recibido: {'date': '2021-03', 'total_injured': 954}
Datos actualizados en Google Sheets: {'spreadsheetId': '1Dflw-mewAhm6Nqa60dxc9BgLTklrMrbnyWf485XWOFE', 'updatedRange': 'Hoja1!A4:B4',
ws': 1, 'updatedColumns': 2, 'updatedCells': 2}
Mensaje recibido: {'date': '2021-04', 'total_injured': 2122}
Datos actualizados en Google Sheets: {'spreadsheetId': '1Dflw-mewAhm6Nqa60dxc9BgLTklrMrbnyWf485XWOFE', 'updatedRange': 'Hoja1!A5:B5',
ws': 1, 'updatedColumns': 2, 'updatedCells': 2}
Mensaje recibido: {'date': '2021-05', 'total_injured': 1658}
Datos actualizados en Google Sheets: {'spreadsheetId': '1Dflw-mewAhm6Nqa60dxc9BgLTklrMrbnyWf485XWOFE', 'updatedRange': 'Hoja1!A6:B6',
ws': 1, 'updatedColumns': 2, 'updatedCells': 2}
```

Real-Time Visualization with Looker Studio

- Every time data is updated in Google Sheets, the Looker Studio DashBoard automatically refreshes, displaying the latest information in real time.

- Types of Graphics on the DashBoard:
  - Line Chart
  - Bar Chart
  - Indicator



REAL TIME-DASHBOARD