- Cada año se producen 1,9 millones de muertes como consecuencia de accidentes de tráfico; En 2022, habrá 1.175 muertes por accidentes de tráfico en Nueva York

NEW YORK ACCIDENTS (2021-2022)

JUAN MANUEL LOPEZ RODRIGUEZ (2226066)

WILLIAM ALEJANDRO BOTERO FLOREZ(2225155)

TABLA DE CONTENIDO

OBJETIVO DEL PROYECTO

Realizar el análisis de datos de nuestras bases proporcionadas juntas, las cuales tienen datos muchos más completos y estructurados, que nos permitan visualizar la distribución de estos para un mejor entendimiento del contexto

USED TOOLS

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

Apache Airflow

pandas

PostgreSQL

kaggle

Power BI

WORKFLOW

Apache Airflow

USA ACCIDENTS (Database)

NEW YORK DATA (API)

Transform Data

Transform Data

NEW YORK ACCIDENTS (Merge)

Dimensional Model

Upload to Google Drive

PostgreSQL

Upload to PostgreSQL

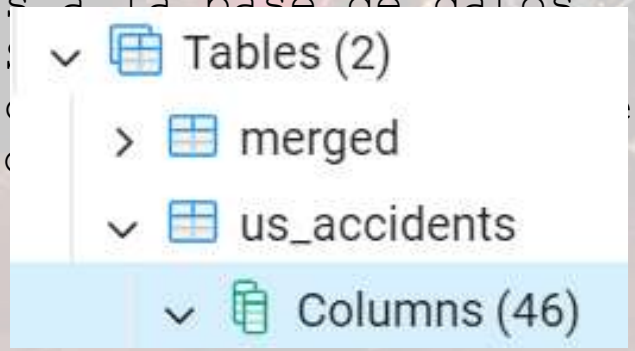
DashBoard

Power BI

ACCIDENTS IN USA

Para la obtención de los datos, los extrajimos desde la pagina de Kaggle, sobre el dataset de "US Accidents (2016-2023)", el cual nos da información sobre accidentes de tránsito que ocurrieron en Estados Unidos entre 2016 y 2023.

Una vez, descargados los datos, los subimos a la base de datos Postres la conexi trabajo y los



Una vez subida la base de datos a PostGres, importamos los datos a nuestro entorno de trabajo, para empezar a realizar todos lo relacionado al ETL de los datos

```
# 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)

- Para la limpieza de datos, lo que hicimos fue identificar datos nulos, patrones repetidos, columnas que no dieran información relevante que se puedan borrar y rellenar datos nulos con una imputación de datos.

```
# 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_i
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

- Tomando una parte más pequeña del país, decidimos enfocarnos en solo los accidentes que ocurren en la ciudad de Nueva York, así, tomamos los datos de la siguiente API: https://data.cityofnewyork.us/resource/h9gi-nx95.json
- La API, tiene una restricción de 1000 datos, por lo cual, al momento de consumirla, decidimos generar unos parámetros, para que, en vez de traer

```
# 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')
```

- Estas columnas, nos dan una vista previa de que datos podremos usar para poder realizar el merge con nuestra base de datos, y así, tener nuevos datos más completos.

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"
```

- Para la limpieza de la API, hicimos un proceso muy parecido al anterior, además de hacer una estandarización de los datos, elegir el formato correcto de las columnas relacionadas con fechas, guardar los datos que se registraron a partir del año 2021, ya que en los años anteriores se contaba con muy poquitos registros que generaban datos atípicos y por ultimo, crear una columna que se llamara ciudad, en la cual para todas las filas iban a ser iguales a New York, ya que esto nos iba a facilitar el proceso de hacer el Merge entre las 2 bases de datos.

MERGE ENTRE BASES DE DATOS

- Para la mezcla entre amabas bases de datos, lo que hicimos fue realizarlo mediante la fecha del hecho, donde concordará el día y la hora en donde ocurrió el accidente, además se aplicó un filtro donde solo nos trajera los accidentes que ocurrieron en New York, principalmente en la base de datos guardada en PostgreSQL, para ter zcla

```
# 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)

- Para la limpieza de los datos ya hecho el merge, se identificaron patrones repetidos, eliminamos columnas no importantes, eliminamos columnas repetidas que salieron a partir del merge, conversión de formatos de fechas, conversión de columnas flotantes a enteros, mezcla entre columnas para información más completa y estandarización de

```
# 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

- Luego de realizar toda la limpieza necesaria al merge de las bases de datos, quedamos con la base de datos final la cual nos habla solamente de Accidentes en la ciudad de New York, información a la cual le haremos todo lo relacionado al análisis

```
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

Comparison between day and night accidents

```
Comparison between day and night accidents:
sunrise_sunset
Day      31947
Night    15277
```

Distribution by type of vehicle involv

```
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
```

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

```
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

```
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:

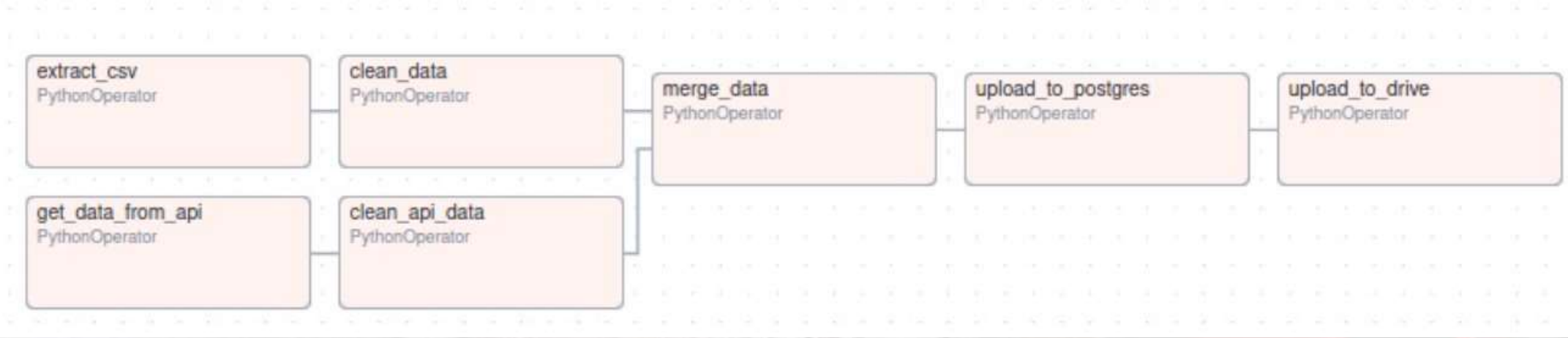
vehicle_type_code1	
Sedan	14633
Station Wagon/Sport Utility Vehicle	10300
Taxi	971
Bike	617
Pick-up Truck	536
...	...
MINIVAN	0
MINI BUS	0
Lunch Wagon	0
LOCOMOTIVE	0
van	0

Most common factors in accidents without

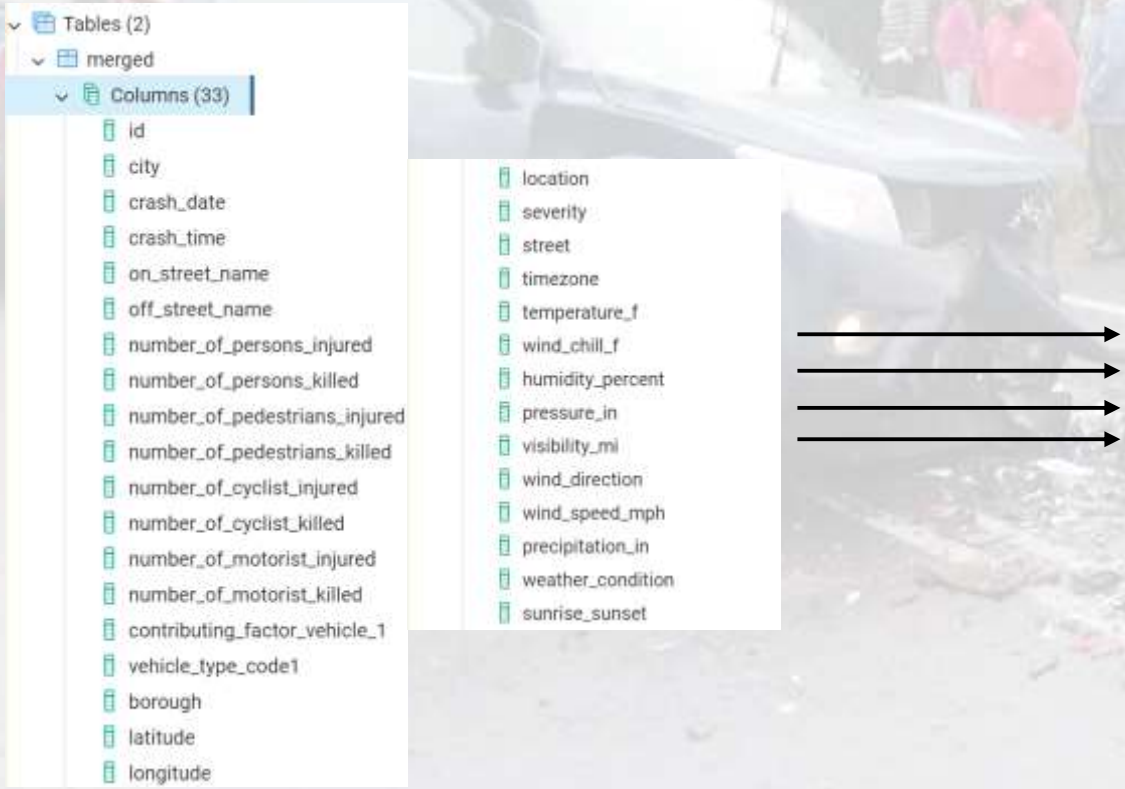
Most common factors in accidents without injuries:

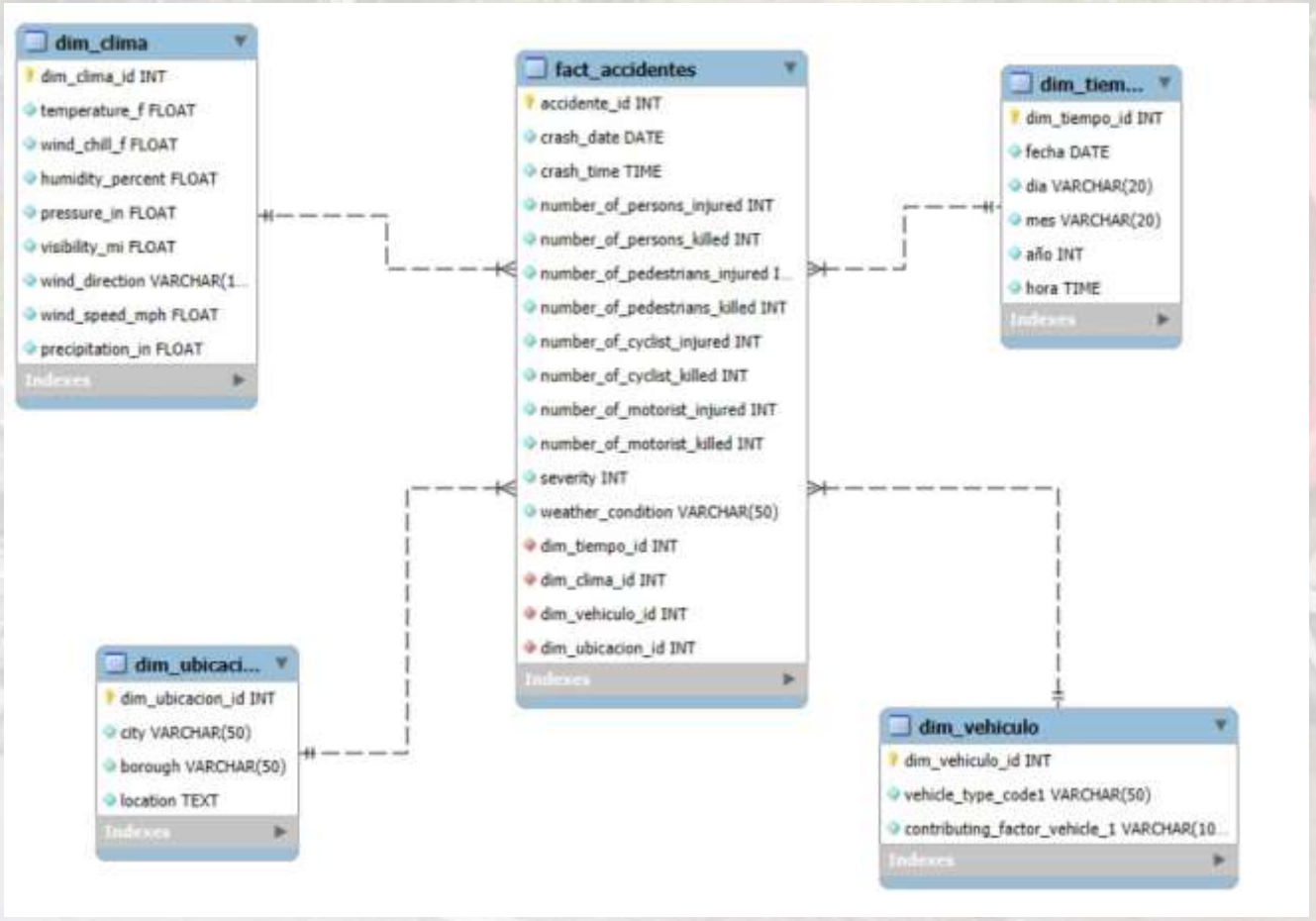
contributing_factor_vehicle_1	
Driver Inattention/Distraction	6780
Unspecified	4674
Failure to Yield Right-of-Way	1879
Passing or Lane Usage Improper	1701
Following Too Closely	1582
Passing Too Closely	1239
Traffic Control Disregarded	1091
Turning Improperly	1040
Backing Unsafely	911
Unsafe Speed	789
Other Vehicular	690
Alcohol Involvement	591
Driver Inexperience	545
Unsafe Lane Changing	530
Reaction to Uninvolved Vehicle	323

AIRFLOW


extract_csv
PythonOperator
clean_data
PythonOperator
merge_data
PythonOperator
upload_to_postgres
PythonOperator
upload_to_drive
PythonOperator
get_data_from_api
PythonOperator
clean_api_data
PythonOperator

DIMENSIONAL MODEL





DATA DASHBOARD

TOTAL ACCIDENTS

47224

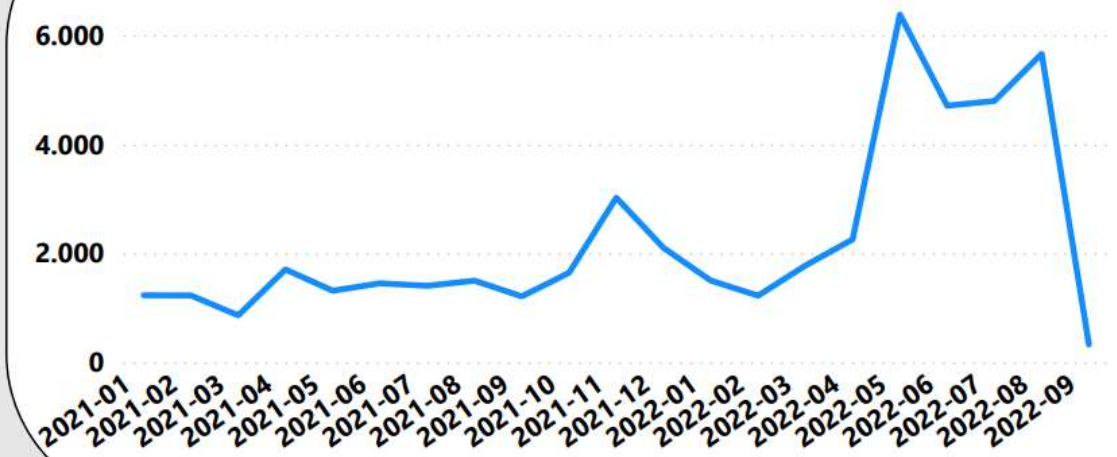
Severity

Todas

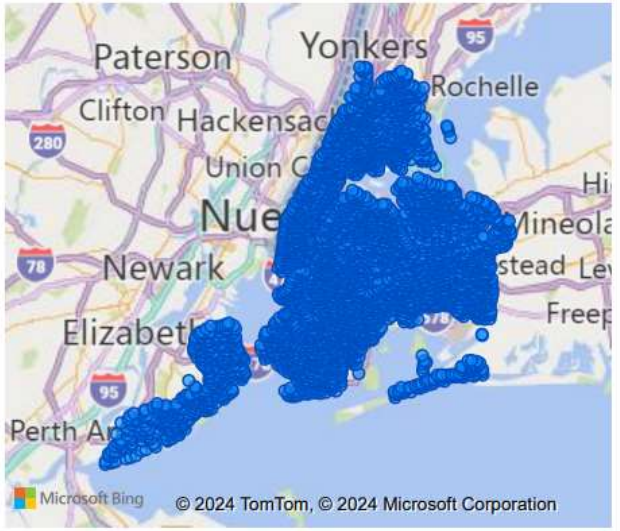
Contribuing Factor Ve...

Todas

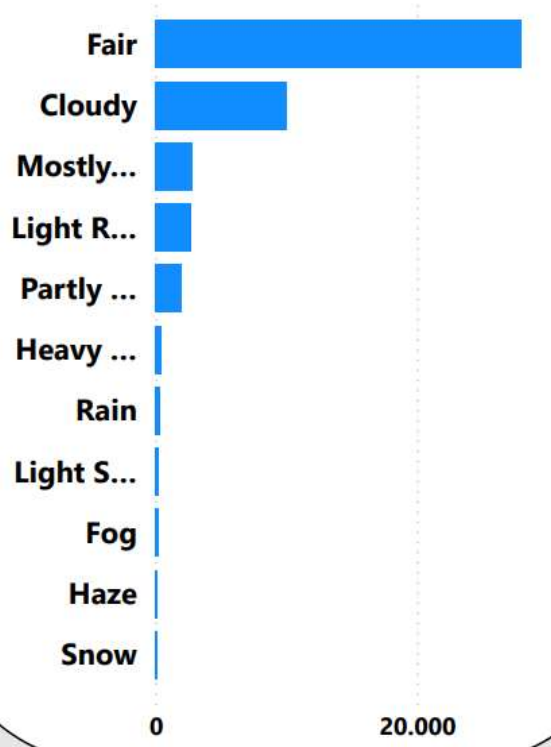
Accident Trend Line per Month



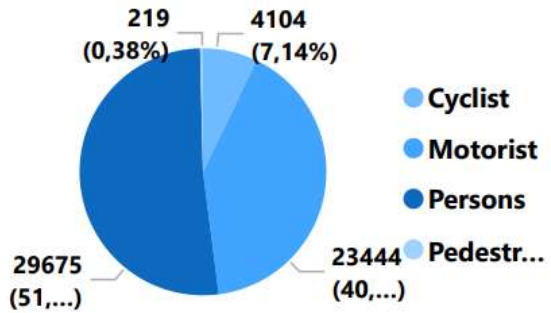
New York Accidents Map



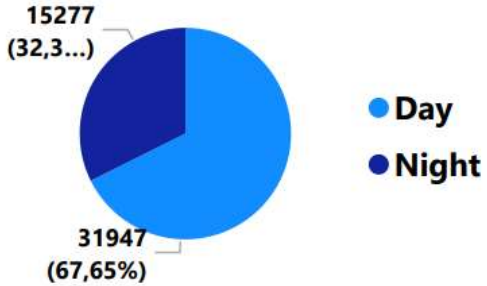
Weather Conditions by Accidents



Individuals Injured by Accident



Accidents by Time of Day



Accidents by Bourough