UNIVERSITÀ DI PISA

Department of Computer Science:

Masters degree in Data Science and Business Informatics

Laboratory of Data Science: Project report

Pietriccioli Alice
Yilmaz Elif

Academic year 2024/2025

Contents

1 Project part 1

1.1 Data Understanding

1.1.1 Crashes, Vehicles, People: datasets introduction

In this first we will analyze all the dataset, Vehicles, Crashes, and People, in order the understand the variables' semantics, the way these three files relate to each other and eventually if there are any missing values.
The Crashes dataset contains 36 features and 138,593 rows; it shows information such as the date of the accident, the precise location and road where it occurred, the causes of the accident, road surface and weather conditions, and details on any injuries sustained by the people involved.
The Vehicles dataset contains information of all the different parties involved in the registered traffic crashes such as the type of vehicle and its model, license plate, and possible defects. It is important to note that this dataset does not only include motor vehicles but also 'units', which can include pedestrians, for example. In such cases, we will not have information relevant only to cars, such as those listed above. The feature in dataset are 17 and describes 460,437 records.
The last of our three dataset, People, is the one concerning people involved in a crash and if any injuries were sustained, their residence, the parties' physical conditions and the damage of the crash. It has 19 columns and 564565 entries.

1.1.2 Crashes, Vehicles, People: the relationship among them

As we mentioned in the introduction, each of these three CSV files contains information on individual recorded crashes. Crashes dataset and vehicles have a single attribute as a unique identifier:`rd_no` is the only feature with a unique value for each row , while in the Vehicles dataset, `crash_unit_id` serves as a unique identifier, even if `vehicle_id` is also a variable, which identify every different motor vehicle, and they can be repeated if involved in more than one crash. On the other hand, the People dataset does not have a key composed of a single attribute, as `person_id` also has duplicates. This means the same person was involved in multiple crashes and recorded with the same ID. Therefore, the key for this table is formed by combining the `person_id` column with `rd_no`, so we can have information of a person involved in one crash.
The relationship that connects the three files is the individual crash. Preliminarily, we understand that the records in these three CSV files can be linked using `rd_no`, which allows us to determine both the people and vehicles involved in the same incident. To identify whether a person was inside a motor vehicle, we can use `vehicle_id`. If this key is missing, it indicates that the person in question was not inside any vehicle. Supporting this is the fact that `vehicle_id` in the vehicles table also contains missing values when the `unit_type` variable does not pertain to motor vehicles. Additionally, in such cases, all other columns related to the vehicle will also have missing values.

1.1.3 Missing Values among the datasets

By analyzing each dataset, we noticed that each one had several columns with missing values, for example as we had already expected in the case of vehicles. The types of missing data found are not limited to standard 'NaN' values (easily detectable using the 'numpy.nan' method); many columns also include 'UNKNOWN' or 'UNKNOWN/NA' among their unique values. Therefore, we will likely have a higher percentage of missing data in many columns. Now, we will examine in detail which columns in each dataset have null values (this analysis also includes the UNKNOWN values):

- Crashes: In this dataset, the majority of columns do not have any NaN values. There are some columns with a total of nearly 10 missing values (`street_direction`, `street_name`, `beat_of_occurrence`, `most_severe_injury`). Notably, the three columns related to the crash location (`longitude`, `latitude`, and `location`) have the same number of NaN values, approximately 0.39%; finally, the other columns (`traffic_control_device`, `device_condition`, `weather_condition`, `lighting_condition`, `first_crash_type`, `roadway_surface_cond`, `trafficway_type`, `road_defect`, `report_type`) have a percentage ranging from at least 1% to a maximum of 14%;

- Vehicles: out of 17 columns, only 5 do not have any null values (of these, `unit_type` has only one NaN record), then only three have a percentage under 10% (`vehicle_id`, `travel_direction`, `occupant_cnt`) while the majority of the other columns have a percentage of null values around 10%, (`make`, `lic_plate_state`, `vehicle_year`, `vehicle_type`,`travel_direction`, maneuver, first_contact_point); finally, the remaining columns have a percentage exceeding 20%, (model, vehicle_defect, vehicle_use);

- People: This dataset has the highest number of columns with NaN values. Out of 19 columns, only 5 do not have any (person_id, person_type, rd_no, crash_date, damage_category). Some columns have a limited number of NaN values (vehicle_id, sex, safety_equipment, airbag_-deployed, ejection, injury_classification). Following these, there are some columns with a percentage of missing data ranging from 13% to 30% (damage, bac_result, age, state, city). Finally, the columns with a significantly high percentage of missing data are the two related to the driver (driver_action and driver_vision, with 36% and 51% of NaN values, respectively) and physical_condition, with 43%.

It is important to note that missing values may also stem from the type of unit, as we mentioned in the dataset descriptions (subchapter 1.1.1). For example, columns with high numbers of missing values could be due to the type of 'party involved' being described.

1.2 Data Cleaning

1.2.1 Handling Nan values

We have observed that each CSV file contains many columns with null values. We will only replace standard NaN values, substituting them with 'UNKNOWN', if they are string values, and -1 if numeric type, and for the missing date we put '1900-01-01 00:00' (to keep the same date format and to avoid problems during the uploading phase); we choose to avoid compromising real data in cases where values cannot be derived; therefore, the result we will obtain will show the same percentage of missing values as found in the initial analysis, but now the NaN values will appear as the unique value 'UNKNOWN' or -1, to avoid giving false information to the record having a missing value.
The only columns for which we modified the null values are the three related to the crash location coordinates: location, latitude and longitude. Thanks to a specific library, GeoPy, we were able to fill the missing data by using the street number and name, along with the city of Chicago, to retrieve the corresponding longitude and latitude points, and subsequently combine them into the location column; so now these three columns will be free of any null values.

1.2.2 Columns cleaning

We proceeded to make sure that each column has a correct and homogeneous data type, as required by the server, and we also decided to keep the entire data set and do not lose any information in case we have to answer various business questions in the future. Although the dataset, apart from a single removed column, has not been significantly altered, we have made a few adjustments that are outlined below:

- as we observed the presence of numerical values in the city column of the People dataset, we decided to replace those with UNKNOWN values;
- we relabeled the entries X and U in the sex column as 'Others', since these values represented a very small proportion of the total and were grouped under a single category
- the variable injuries_unknown in Crashes.csv , had a unique value of 0 for every observation, therefore, we removed it, as queries on a variable with an unchanging value would be meaningless;
- as Make column had 666 unique values in 460 thousand rows, but majority of them were appearing only once, therefore for the make and model column we decided to set a threshold and mark some of the values as "Others" instead;
- we decided to round latitude and longitude values to 5 decimals, as in that way locations are still precise (up to few meters difference compared to the original entries) but it gives us better opportunity to group very close locations;

Eventually to make our data compatible with the sql server management studio, we formatted the date related columns (using instead the 12 hour format the 24 hour format), handled necessary data type conversions, such as casting the integer values in model column as string, and casting values that should be integer but appearing as floats (such as numbers of injuries) as integers.

1.3 DW Schema and Data Preparation

The data mart we created can be seen in the image 2.4:

- the fact granularity is a person involved in one specific crash;
- the related dimension tables are: Cause, Crash, Date, Geography, Person, Vehicle and Weather (for the attributes of each tables see figure 2.4);
- in the Damage to user table, we have not only the measures (see figure 2.4), but also `damage_-category` which we decide to put here as a degenerate dimension.

To split the data from the three CSV files into the tables of our data mart, we first merged the records into a single file:

1. each record from the People file was used as a base;
2. using the `rd_no` variable, we linked each person to the crash information they were involved in, ensuring no records were lost;
3. then the third file was linked using the `vehicle_id` variable, however, `vehicle_id` contained missing values in both the People and Vehicles files;
4. finally for records where `vehicle_id` was `-1`, we added all unknown values for the columns imported from the Vehicles file, this aligns with our prior discussion that if a person is not associated with a motor vehicle, all values from the Vehicles table are not given.

As a result of these operations, we obtained a unified file where each row represents a person with all the relevant information about the specific crash they were involved in.
Then we divided the Merged file' s columns in the tables of our schema according to the different information that they give, as we can see in the figure 2.4, and we created a primary key for each dimensional table.

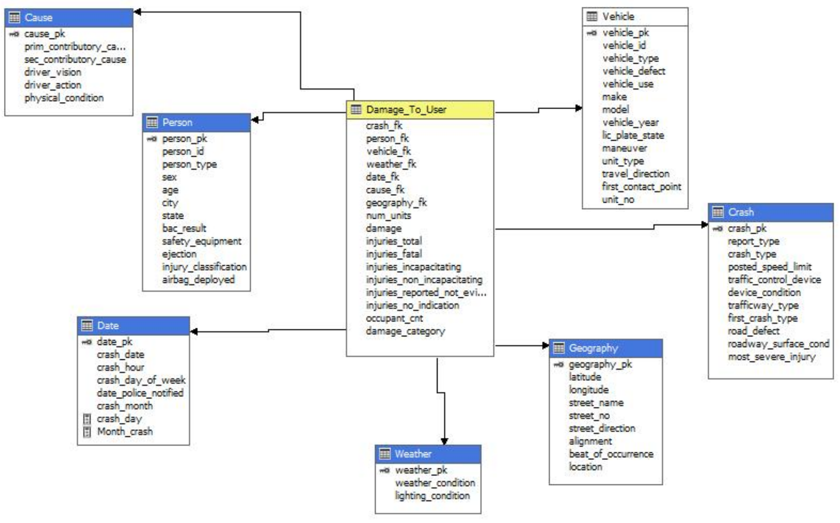

Figure 1.1: Damage to user star schema

For each table in this schema, we created primary keys. For the Cause, Weather, Crash, Geography, and Date tables, primary keys were generated by assigning an incremental key to all possible combinations of the column values. This ensured that there were no duplicate values or errors.
For the Vehicle and Person tables, we created primary keys by assigning incremental values to combinations of the variables `rd_no` and `vehicle_id` (for Vehicle) and `rd_no` and `person_id` (for Person). This approach also ensured the absence of duplicates, even in cases where a person or a vehicle was recorded in multiple crashes.
The Vehicle table, in our schema, is the only optional dimension. An individual involved in a crash might not be associated with any motor vehicle, resulting in unknown values for every column in this table. Nevertheless, to prevent errors in the SQL server and maintain foreign key constraints, we added a record with a primary key equal to -1 and all columns set to unknown values. This allows us to link

all users not associated with any vehicle (the ones having `vehicle_fk` = -1 in the fact table).
To load the tables into SQL Server using the python library' s pyodbc, we began by creating the empty tables directly in SSMS, carefully defining the correct data types for each column to avoid errors during the insertion of data from the CSV files and by adding by direct query in SSMS the record with primary key equals to -1 (and all the other attributes as -1 or "UNKNOWN") in Vehicle table to avoid any errors while the uploading of all the other datas. Then we design the code for connecting to the server, by defining the connection string and creating the cursor for the sql commands, and saving all our operation (`connection.commit` was necessary for saving our tasks at the very end of the file) and to speed up the data insertion process, we utilized the `cursors.executemany` command and also by splitting for each table the datas into "batchs" which allowed for efficient loading of data into the respective tables. Additionally, during this stage, we also prepared empty tables specifically intended for sampling purposes, to be used later in SSIS.

1.4 SSIS queries

In this part, we created 2 different projects, the first for the sampling task, which contains a package for each sampling of the table. We connect every table for the sampling the OLE DB connection, then for the sampling task we use a fixed seed to avoid having different results if the package is run more times, then we save the result in every empty 'SSIS' table created in the previous assignment. Then we create another SSIS project to solve all four queries, each of which is a different package, and we choose to save all the results as new tables on the sql server (each with the name 'query number result').
Then, for every given query, we first wrote the correct SQL query to do:

- "Show all participants ordered by the total damage costs for every vehicle type":

```
SELECT v.vehicle_type, p.person_id, SUM(d.damage) AS total_damage
FROM Damage_to_user d JOIN  Person p ON p.person_pk= d.person_fk
JOIN Vehicle v ON vehicle_pk= vehicle_fk
GROUP BY v.vehicle_type, p.person_id
ORDER BY total_damage;
```

- "For each month, calculate the percentage of the total damage costs caused by incidents occurring between 9 pm and 8 am and incidents occurring between 8 am and 9 pm, with respect to the average total damage costs for all months within the same year" [1]:

```
WITH MonthlyDamage AS (
       SELECT YEAR(crash_date) AS crash_year, crash_month,
        CASE WHEN dt.crash_hour >= 21 OR dt.crash_hour < 8 THEN 'Night' ELSE 'Day' END AS day_period,
        SUM(d.damage) AS total_damage
        FROM Damage_to_user d JOIN  Date dt ON d.date_fk = dt.date_pk
        GROUP BY YEAR(dt.crash_date), dt.crash_month,
            CASE WHEN dt.crash_hour >= 21 OR dt.crash_hour < 8 THEN 'Night' ELSE 'Day' END

    ), YearlyAverageDamage AS (
        SELECT crash_year, AVG(total_damage) AS avg_monthly_damage
        FROM MonthlyDamage
        GROUP BY crash_year
     )

    SELECT  md.crash_year, md.crash_month, md.day_period, md.total_damage, yad.avg_monthly_damage,
        (md.total_damage / yad.avg_monthly_damage) * 100 AS percentage_of_avg
    FROM MonthlyDamage md JOIN YearlyAverageDamage yad ON md.crash_year = yad.crash_year
    ORDER BY md.crash_year, md.crash_month;
```

[1]Even in this query there is no request of ordering the results, in the sql server they will be ordered by year (as it was necessary in the SSIS flow, and we also for a better view by month)

- "Show the total crash damage costs for each vehicle type and weather condition"[2]:

```sql
SELECT  v.vehicle_type, w.weather_condition, SUM(d.damage) AS total_damage_cost
FROM Damage_to_user d JOIN Vehicle v ON d.vehicle_fk = v.vehicle_pk
    JOIN Weather w ON d.weather_fk = w.weather_pk
GROUP BY v.vehicle_type, w.weather_condition;
```

From the first query, we noticed that the highest damage is associated with the `vehicle_type` unknown, which corresponds to individuals not in a motor vehicle during the crash. From the second query, we observed that most accidents occurred during daylight hours have an higher total damage. Therefore, we decided to answer this query: "show for all the crash happened from 9 am to 18, the percentage of the damage by each person type, with respect to the total damage of the crash". So our aim is to explore the percentage of damage costs attributable to the different person types in relation to the total damage and our findings showed that, in most cases, the majority of the damage share is linked to drivers. Hence, the query we implemented in SSIS is [3]:

```sql
WITH Damage_Per_Crash AS (
    SELECT d.crash_fk, SUM(d.damage) AS total_crash_damage
    FROM Damage_To_User d
    JOIN Date da ON d.date_fk = da.date_pk
    WHERE da.crash_hour BETWEEN 8 AND 18
    GROUP BY d.crash_fk
),
Damage_By_PersonT AS (
    SELECT d.crash_fk, p.person_type, SUM(d.damage) AS tot_persont_damage
    FROM Damage_To_User d JOIN Date da ON d.date_fk = da.date_pk
        JOIN Person p ON d.person_fk = p.person_pk
    WHERE da.crash_hour BETWEEN 8 AND 18
    GROUP BY d.crash_fk, p.person_type
)
SELECT pd.crash_fk, pd.person_type, pd.tot_persont_damage, cd.total_crash_damage,
    (pd.tot_persont_damage * 100.0 / cd.total_crash_damage) AS percentage_over_total_damage
FROM Damage_By_PersonT pd JOIN Damage_Per_Crash cd ON pd.crash_fk = cd.crash_fk
ORDER BY pd.crash_fk;
```

[2]Even if in this query there is not an order by, we decided to put an order by `vehicle_type`,in the SSIS data flow, to have a better view of the results

[3]In the four query there is a typo error: the crash hour should be between 9 and 18, not 8

2 Project part 2

2.1 Creation of the OLAP Cube

To create our cube we followed the instructions given on the lectures. We created the cube on Visual Studio, firsly setting our source as the database we loaded to SSMS. Then we set our dimentions and created our cube to work on. One problem we encountered was due to how we handled the missing values, we were not able to succesfully create the hiearchies within the cube. We added an arbitrary date for missing date values (01-01-1900), -1 for missing numeric values and "unknown" for missing string values which at this point of the project we realized was a problem once we tried to set up. Thus we decided to keep our hiearchies flat and move forward.

2.2 MDX queries

Out of the 4 optional queries, we decided to answer the assignments 2,3,4 and 6. For the "location" mentioned in assignments 2 and 4 we initially tried to add zip codes to each crash occurring, using Geopy library. But even if we left the code running overnight, the attempt was unsuccessful. Therefore we decided to use a bucketing technique where through the SSMS we ran a query where we divided the latitude and longitude values into 0.1 degree intervals and ended up with 16 buckets as our locations.
Although we got answers from assignments 2-3-4, we struggled to get accurate answers for assignments 6 and 8.1. Assignment 6 gave us the Vehicle Type, Year, and the highest associated damage but failed to show the person ID and person related information. Assignment 8.1 was succesfull to show the overall most common causes but failed to demonstrate the divisions between the years. To address these problems, we tried to add parts of the queries as calculated measures inside the cube (DamagePerYear, which was keeping track of total damage associated per Person Id, or WeightedCauseCountYear where we tried to assign a formula that counts the occurrence of causes and assigns them a weight to answer assignment 8.1. We also added the count measures for primary and secondary causes to allow analysis for assignment 8.1)
These attempts were not the solution, and we believe the main problem was the fact that we were not able to set the hiearchies correctly, thus the analysis were not able to go through accurately, especially why we struggled to separate most common the causes per year (we ended up getting a lump sum value whatever we tried). However, due to time constraints we were not able to go back to our database and change from how we handle missing values to be able to set the hiearchies accurately.

2.3 Dashboards

To create the three requested dashboards, we used PowerBI and opted to save our report instead of publishing it.
For the first dashboard, we utilized the "Map" feature, where we added in the commands the `latitude` and `longitude` variables instead of specifying the location feature.

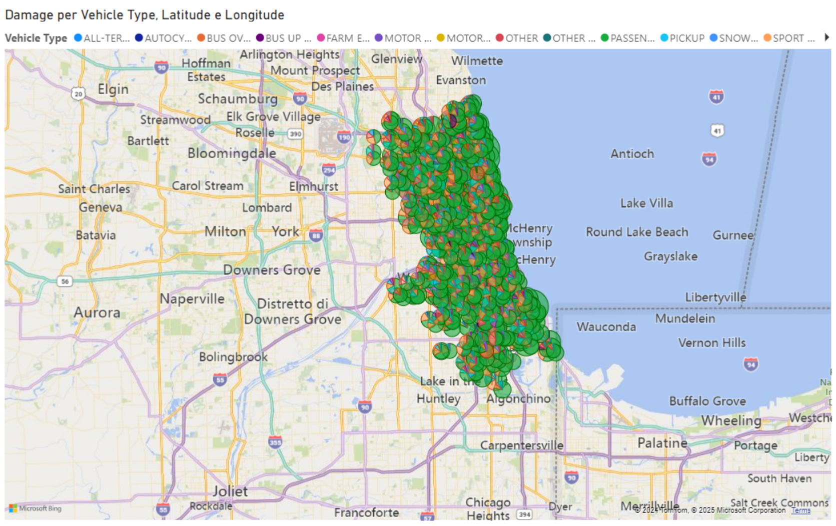
Damage per Vehicle Type, Latitude e Longitude
Vehicle Type ● ALL-TER... ● AUTOCY... ● BUS OV... ● BUS UP ... ● FARM E... ● MOTOR ... ● MOTOR... ● OTHER ● OTHER ... ● PASSEN... ● PICKUP ● SNOW... ● SPORT ...
Elgin
Schaumburg
Bloomingdale
Naperville
Aurora
Joliet
Evanston
Antioch
Wilmette

Figure 2.1: Map: geographical distribution of the total damage costs for each vehicle category

The size of the bubbles generated on the map reflects the total amount of damage caused by crashes at each exact location. Furthermore, for each bubble, we visualized the distribution of different vehicle types (by putting the corresponding variable in the legend command) involved in accidents at that specific location. To ensure accuracy, we excluded records with missing location data, allowing us to display only precise locations distributed across Chicago's streets.
The second dashboard we decided to create is a line chart displaying the semi-annual trend of total damage for each crash that occurred on a specific street within the given time frame.

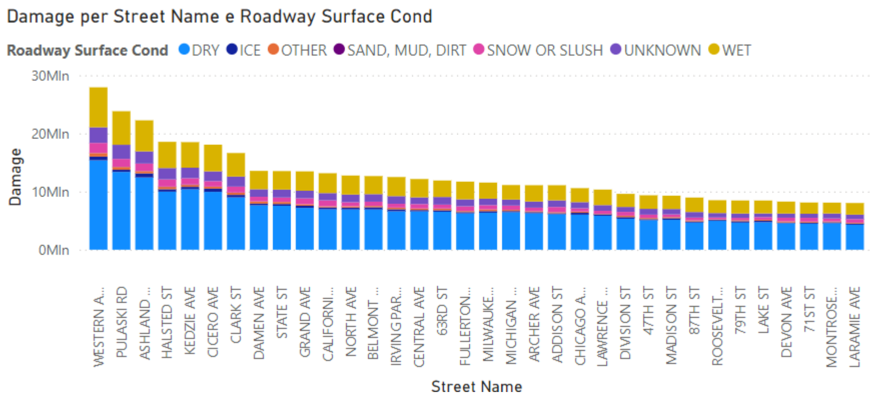

Figure 2.2: Stacked bar chart: distribution of total damage over each street, by roadway surface condition

To choose which street to analyze, we first created a stacked bar chart to identify which street reported the highest total damage in our data. We selected "Western Ave" for further analysis. Additionally, we wanted to examine the surface conditions of the road at the time of each crash. We observed that most of the time the surface condition was "dry," followed by "wet."

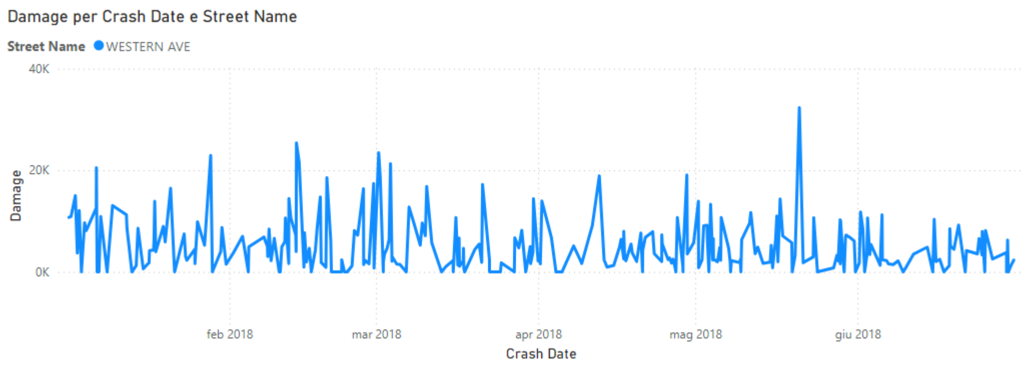

Figure 2.3: Line chart: distribution of total damage over Western Ave street, with wet roadway surface during the first half of 2018

Therefore, we focused our line chart on Western Ave with a "wet" surface condition during the first half of 2018. Thanks to the hierarchy present in the data from our OLAP cube, we were able to analyze the data not on a monthly basis but on a daily level. When examining the data aggregated for the first semester, we observed several peaks around 20k total damage, with one notable case in May where the total damage reached nearly 40k.
Finally for the third dashboard, we used a "stacked bar chart" visualization. In this graph, we represent the total amount of damage for each type of person, with the classification of injuries of each person as a legend. We excluded the distinct value "no indication of injury" because it was the most frequent value across all categories.

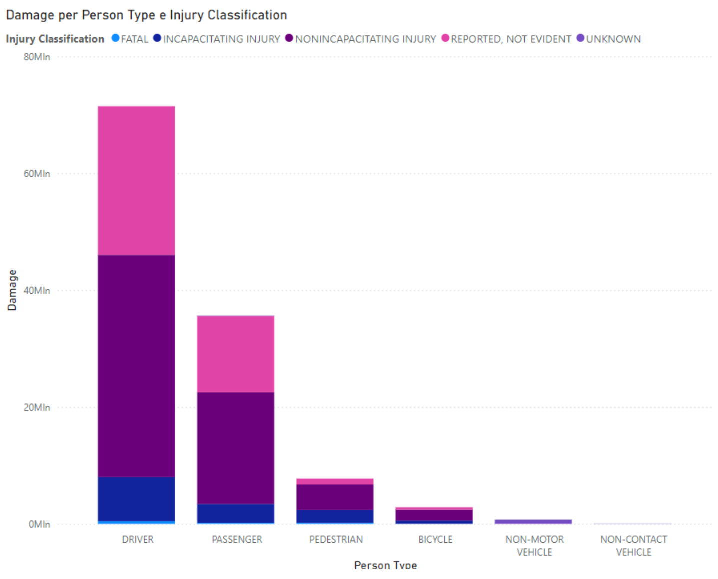

Figure 2.4: Stacked bar chart: total damage distributed over each type of person by different type of injuries

From this visualization, we can see that the category with the highest damage is the "drivers," followed by the "passengers" (both categories that was inside a motor vehicle during the crash). For both categories, the most commonly reported injuries are "non-incapacitating" injuries, followed by "reported, non-evident" injuries. Therefore, we observe that missing injuries or the mildest ones are the most frequent among individuals.