Università degli Studi di Pisa

LDS Part 1 Group 09

Emanuele Sciancalepore (678521)
Emanuele Moscuzza (677879)
Cristian Leone (673375)

Academic year 2024/2025

Contents

1 Data Understanding Preparation

Starting with Data understanding, we have three datasets: *Crashes.csv*, *People.csv* and *Vehicles.csv*. Each subsection in Chapter 1 deals with a single dataset and only the relevant changes are mentioned, as more specific details can be found in the relevant scripts (*CrashesCleaner.py*, *VehiclesCleaner.py*, *PeopleCleaner.py*).

1.1 Crashes dataset

Concerning *Crashes.csv*, we made several changes, as the dataset had several missing values and outliers.

The column *RD NO* is the unique identifier of the crash and has no duplicate and missing values.

For the *LIGHTING CONDITION* column, we filled in the 'UNKNOWN' values based on the season and time when the incident occurred. However, in cases where the result was 'DARKNESS', we left the original value of 'UNKNOWN' as we could not know whether the road was lit or not.

The columns *STREET DIRECTION*, *STREET NAME* and *BEAT OF OCCURRENCE* had a total of only 7 missing values: they were filled in by finding occurrences of other incidents in the same geographical circumstances.

As far as injuries are concerned, we decided to remove all columns related to the injury count (*INJURIES FATAL*, *INJURIES INCAPACITATING*, ...) because they were considered redundant information that could be obtained from the column *INJURY CLASSIFICATION* in the *People.csv* dataset. This allowed us to save a lot of space resources. We left only the column *MOST SEVERE INJURIES*, of which we filled in the missing values with the pre-existing value 'NO INDICATION OF INJURY'.

For the *LATITUDE* and *LONGITUDE* columns, values were initially replaced by extracting them from the *LOCATION* column, which contained more precise information. However, there were 1022 missing values in these three columns in the dataset, which were obtained using an API (Nominatim[1]) by exploiting address information. Of these, 307 values were not found, as the addresses contained outliers that could not be traced back to a specific location. To solve this problem, as this information was needed to improve and aggregate weather data, we decided to approximate the missing values using the *ChicagoPoliceBeat.csv* dataset. This dataset provides detailed geographical information, thanks to which it was possible to calculate the centroid of the police beat where the incident occurred. In addition, we have also added a *GEOHASH* column, extracted from geographical coordinates, which may be useful for future analyses.

The *WEATHER CONDITION* column had several 'UNKNOWN' and 'OTHER' values, our strategy for this variable was to retrieve the missing information through an API (Meteostat[2]), exploiting the space (*LATITUDE* and *LONGITUDE*) and time (*CRASH DATE*) information. This tool allowed us to retrieve extra informations, such as temperature and wind speed at the time of the incident, which may be useful for future analyses.

The columns *CRASH DATE* and *DATE POLICE NOTIFIED* were decomposed by simulating a hierarchical structure and adding the relevant quarter.

1.2 Vehicles dataset

Concerning with *Vehicle.csv*, there were several inconsistencies and missing values.

[1]https://nominatim.org/
[2]https://dev.meteostat.net/

In general, for none of the columns related to the vehicle is it easy to reconstruct the actual missing values. Therefore, these values were replaced with placeholders such as 'UNKNOWN' or 'XX' as appropriate.

We modified the *VEHICLE YEAR* column in the following way: since the column had anomalous vehicle years (e.g. 9999, 3500 etc.), we decided to replace all years that were not in the range [1950, 2024] with 0.

As far as *VEHICLE ID* is concerned, rows in which the values 'DRIVER' or 'NON-CONTACT VEHICLE' are present in the *UNIT TYPE* column are in most cases linked to a *VEHICLE ID*. On the other hand, if the categories were 'BICYCLE', 'PEDESTRIAN' or 'NON-MOTOR VEHICLE', we noticed that the *VEHICLE ID* was always missing along with all the other columns. Therefore, we decided to treat these two groups differently. For the first group, we added an incremental *VEHICLE ID* from the maximum value of this last column (being an integer). For the second group, we have filled in the values with 'UNKNOWN' from the columns besides *VEHICLE ID* because, according to our assessment, there are either no suitable values for the three categories listed above or no way of tracing the actual values. For *VEHICLE ID*, we filled in the value with the string corresponding to the vehicle category. This strategy was adapted to replace these 10372 in only 3 unique rows, one for each category. This could happen because we decided to remove the columns *CRASH DATE*, *UNIT NO* and *CRASH UNIT ID* (once it will be used to solve connections problems in *People.csv*, as we will see in the next subsection).

1.3 People dataset

Concerning the *People.csv* file, we realised that different strategies were needed to solve the problem of missing values. First of all, however, we dealt with duplicates: in particular, some records appeared several times, always presenting the same value for each column, except for the *DAMAGE* column. We therefore assumed that these were insertion errors, probably due to the fact that the value of *DAMAGE* had been updated without the previous record being deleted. Consequently, we decided to keep only the first duplicate record and remove all others.

As for the *AGE* column, we decided to fill in the missing values and correct the presumably incorrect ages (e.g. 3-year-olds who turned out to be drivers) by using the distribution of corrected ages. In this way, we were able to complete the incorrect fields while preserving the proportion of the *AGE* column and making it available for future analysis.

For the *DAMAGE* column, it is important to point out that the only missing values identified were for the *DAMAGE CATEGORY* in the 'Less than 500' range. Taking advantage of this information, we decided to fill in the empty fields by generating a random number between 0 and 500. It was necessary to take this route as this feature represents the heart of what will be the fact table constructed in section 2 and therefore vital to our analysis.

As for the *CITY* column, we adopted the following strategy: after realising that many records had typing errors or similar writings, we decided to use two external lists. The first contained the names of existing American cities, while the second included the 25 most frequent cities in our source CSV file. With the first list, we identified all cities with correct names, for which no cleaning operation was necessary. For all other cities considered incorrect, we used the second list. Specifically, each record found to be incorrect was compared with each item in the list, and if the similarity score (fuzz.tokensetratio[3]) exceeded the threshold set at 80, the name was updated with that of the most similar city. Thus, for example, records with values such as 'CHG', 'CHICGO', 'CIHCG' were corrected and replaced.

With regard to the other columns, we considered it appropriate not to manipulate their internal structure, thus replacing the missing values with 'UNKNOWN'.

[3]Rapidfuzz library documentation: https://rapidfuzz.github.io/RapidFuzz/

Turning to the *VEHICLE ID* column, there are several missing values. For some of these (with *PERSON TYPE* = [‘BICYCLE’, ‘PEDESTRIAN’, ‘NON-MOTOR VEHICLE’]) we replaced the empty field with the corresponding *PERSON TYPE*, while for the others (*PERSON TYPE* = [‘DRIVER’, ‘NON-CONTACT VEHICLE’]) we exploited the link between *CRASH UNIT ID* (in *Vehicle.csv*) and *PERSON ID*(in *People.csv*). In particular, we found that, for the observations examined, removing the first character of *PERSON ID* (a letter indicating whether the person is a passenger or not), the two strings were the same. In this way, we were able to derive the associated *VEHICLE ID*.

2 Datawarehouse schema

In figure 1, it is possible to visualize the Data Warehouse constructed. The fact concerns damage to the user involved in an accident, thus a single person in a single accident.

Compared to the proposed structure, we have added an extra dimension (*road dim*) to the diagram, which contains information regarding the condition, structure and signage of the road. We then merged the dimension of the *cause* with the dimension of the *crash*. We also added a surrogate key for each dimension for vehicle-dim and person-dim despite the fact that they already had their original id. This is because we treated vehicle id and person id as plate and SSN while the surrogate key must be unique for each person and vehicle in a specific incident, since there are some fact-specific variables.

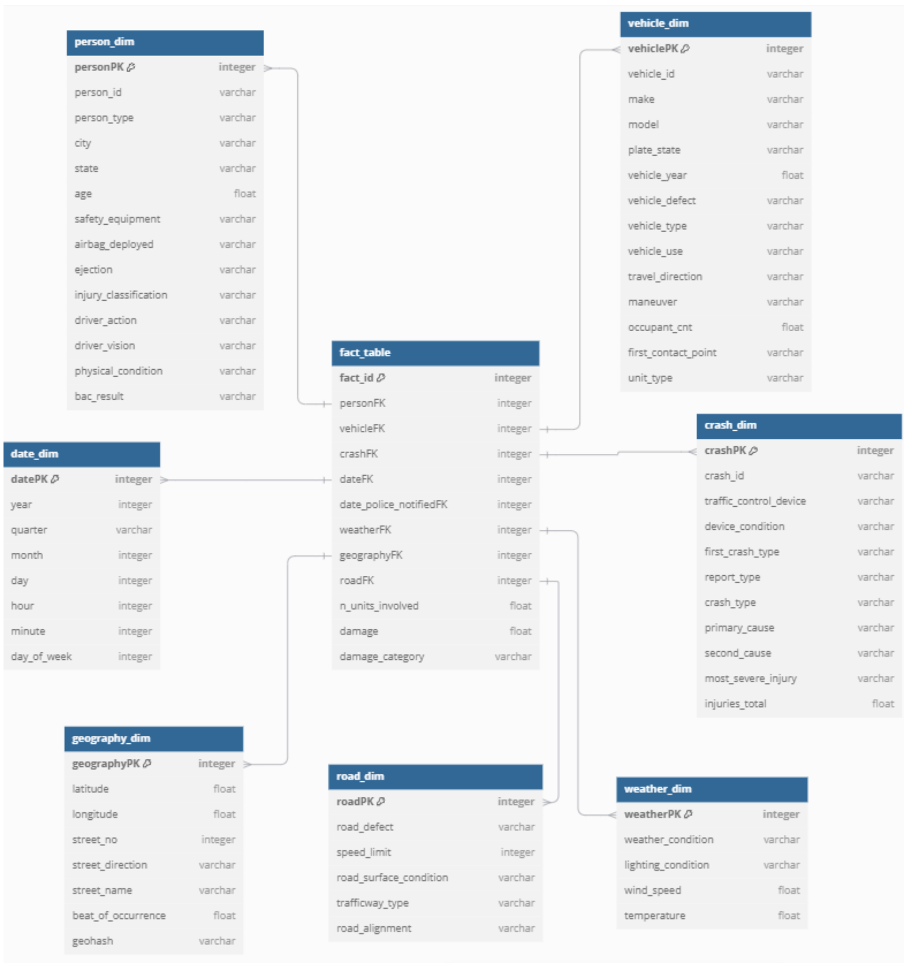

Figure 1: Data Warehouse Schema

3 Data Splitting & Uploading

To split the data into several csv files, we wrote a Python code (*splitter.py*) that after reading the 3 original csvs, places each column in the respective csv representing a dimension. It is important to note that, for each dimension, the code automatically adds an incremental surrogate key to each row not yet seen so far in order to eliminate duplicates. Furthermore it creates the fact table with the respective foreign key.

The next step is based on the Uploading section. We wrote a Python script (*upload.py*) that implements a universal function using a unique query, adaptable to any table and any set of columns.

4 Sampling with SSIS

The workflow in question can be found in the attached SSIS file (*Assignment6*). The process consists of two data flows: one dedicated to the fact table and the other to the dimensions.

In the first data flow, the fact table is read from the database, percentage sampled and loaded into a new dedicated table, named *fact_table_SSIS*.

In the second data flow, the newly loaded table is used to extract foreign keys corresponding to each dimension. For each list of foreign keys, a sorting is applied to eliminate duplicates. Next, a lookup is performed with the respective dimension table in the database. Finally, the data are loaded into SSIS in their respective dimensions.

Note: the *datePK* and *date_police_notifiedPK* keys, referring to the same dimension, were first concatenated.

5 Queries with SSIS

5.1 Assignment 6a

For the first query, in Figure 2, we joined the fact table with person dimension and date dimension using the Lookup function. Then, we grouped by *year* and *person id* and we counted the number of different facts.

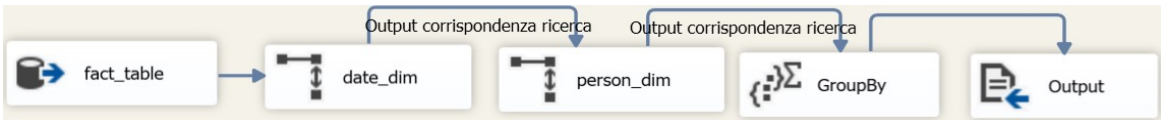

Figure 2: Query 1

5.2 Assignment 7a

For the second query, we first performed a join between the fact table with the geography dimension and date dimension using the Lookup function. We then added a derived column, called *day-or-night*, which returns 1 when it is day and 0 when it is night. We next grouped by *beat of occurrence* and by *day-or-night*, and summed the number of units involved. In the next step, we divided into two flows (one for when it is day and one for when it is night) and afterwards we joined by *beat of occurrence* so as to have the two counts on the same row. Finally, we created a derived column that calculates the ratio between the two counts.

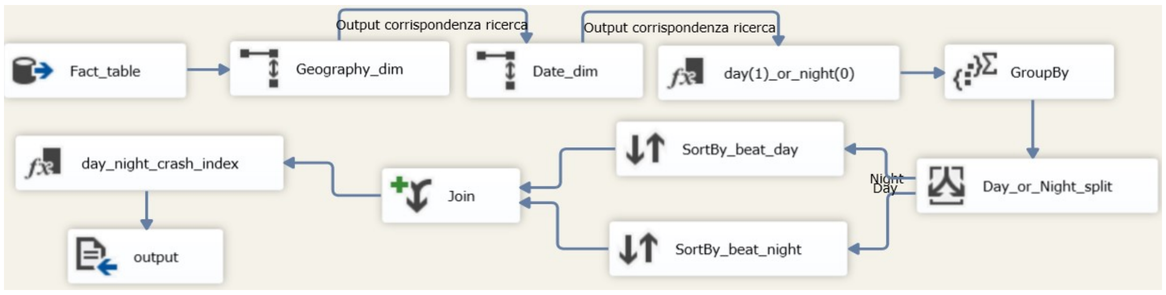

Figure 3: Query 2

5.3 Assignment 8a

For the third query, we performed joins between fact table, date dimension, geography dimension, person dimension and weather dimension. We then added two derived columns (*under21* and *over21*) that output a binary based on age. In the next step, we grouped by *quarter*, *weather condition*, *beat of occurrence* and *crashFK* to sum the two newly derived columns and obtain the two counts for each crash. Then we created a new column where we calculated the ratio for each crash (adding 1 when the denominator equals 0). Finally we performed a second grouping by *quarter*, *weather condition* and *beat of occurrence* calculating the average of the ratios for each group.

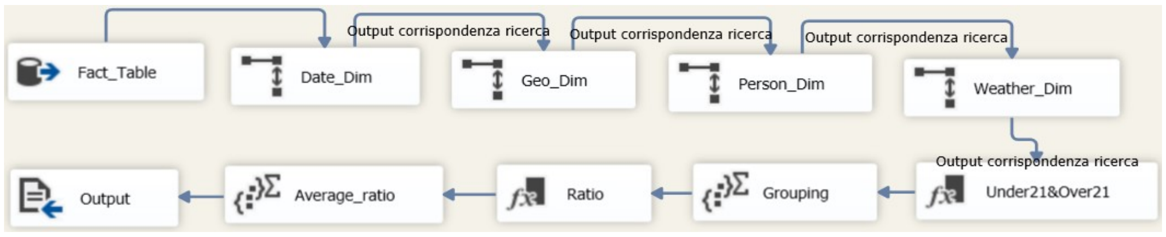
Output corrispondenza ricerca
Output corrispondenza ricerca
Output corrispondenza ricerca
Fact_Table
Date_Dim
Geo_Dim
Person_Dim
Weather_Dim
Output corrispondenza ricerca
Output
Average_ratio
Ratio
Grouping
Under21&Over21

Figure 4: Query 3

5.4 Assignment 9a

In this section, we came up with a question, namely 'What are the most dangerous areas based on the number of accidents with injuries, and what are the average temperature and wind speed recorded in those areas?' The query was structured according to two parallel flows: in the first, we joined the fact table with person dimension and geography dimension using the Lookup operation and calculated a derived column applying a weight to each injury (0. 25 to 'REPORTED, NOT EVIDENT', 0.50 to 'NON INCAPACITATING INJURY', 0.75 to 'INCAPACITATING INJURY' and 1.00 to 'INJURY FATAL'), then we grouped by *beat of occurrence* and calculated the sum of the weights of the injuries and the count of the persons involved. Then we calculated the *danger score* as the ratio between these two values; in the second flow we joined the fact table with geography dimension and weather dimension, then we grouped by *crashFK* and *beat of occurrence* (this group by was useful to have only one value of *temperature* and *wind speed* for each crash), then the second grouping by *beat of occurrence* was made and the average between *temperature* and *wind speed* was done. Finally, we grouped by *beat of occurrence* and sorted by *danger score* in descending order.

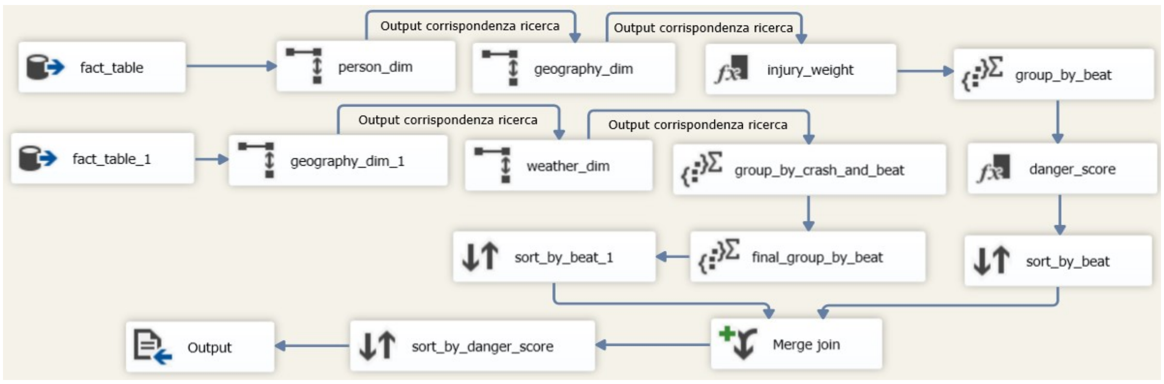

Figure 5: Query 4

6 OLAP Cube Creation

At this stage, for the creation of the OLAP Cube, we established 3 different hierarchies. The first is DayMonthQuarterYear, where relations were created between the following attributes: *Day - Month - Quarter - Year*. The second is CityState, where the hierarchical relationship between *City* and *State* has been created. Finally, in the last one, ModelMake, the relationship between *Model* and *Make* was created.

As far as measures are concerned, we have redefined *Damage*, *n-units involved* and *Fact Counting*.

7 Queries with MDX

7.1 Assignment 2

In this first query, we calculated a grand_total member by calculating the sum total of *Damage* for all months in the *DayMonthQuarterYear* hierarchy of that specific beat. Next in the main query we placed the *Damage Formatted* (the damage formatted with a '$' sign), and the *grand_total* in the columns, while in the rows we grouped by *Beat of Occurrence* and *Month*.

7.2 Assignment 3

For this query, we created two members:*crash_avg_dmg* and *year_avg_dmg*, which are respectively the average damage for each crash and the yearly average damage based on the damage averages of individual crashes. We then placed *year_avg_dmg* and year on the columns to perform a grouping.

7.3 Assignment 4

In this query, we created a member *Previous Year Damage* to calculate the values of previous years with respect to the current one. We then created the *Damage Increasing % member*, where we calculate the percentage change between the current year's damage and the previous year's damage. Finally, in the columns we entered the calculated members and the *Damage* measure, while in the rows we grouped by *Beat of Occurrence* and *Year*.

7.4 Assignment 6

In this query we created a member *Max Damage Person* where we identify the ID of the person who reported the maximum damage for a given *Vehicle Type* and *Year*. Next we calculated another member,*Max Damage*, where we calculate the maximum damage reported by a single person (that is the person with max damage). Finally we put these two members in the columns and in the rows we put *Vehicle Type* and *Year*.

7.5 Assignment 8

Given the complexity of this query, we have used several members to ensure that it worked properly. The first member created was *PrimCauseCount*, which indicates the number of accidents (*Crash PK*) for primary causes in the context of the current year, the same thing was done for secondary causes (*SecCauseCount*). Next, the total damage caused for each primary cause and each secondary cause (again in the context of the current year) was added up. Then, for both causes, the values 'UNABLE TO DETERMINE' and 'NOT APPLICABLE' were removed, as they were not useful for our analysis. Since our aim is to calculate the most frequent cause, the join made it possible to have for each cause the frequency of when it is primary causes and the fraction of when it is secondary.In fact, both the damage and the frequency are subsequently added together. Therefore, it is finally possible to calculate for each year the most frequent cause based on the *TotalCauseCount*.

8 PowerBI Dashboards

8.1 Assignment 9

In this first dashboard in Figure 6 we used a bubble map, where each bubble corresponds to a pair of coordinates where one or more incidents occurred. The bubbles have a size that increases as the damage increases, and contain information about the contributions of each *Vehicle Type* to that damage. The dashboard is unfortunately not readable because we do not have a geographical location that is more generic than coordinates but less generic than *State*, since (from our analysis) the *geohash* and *Beat of Occurrence* variables are not interpretable by PowerBI.

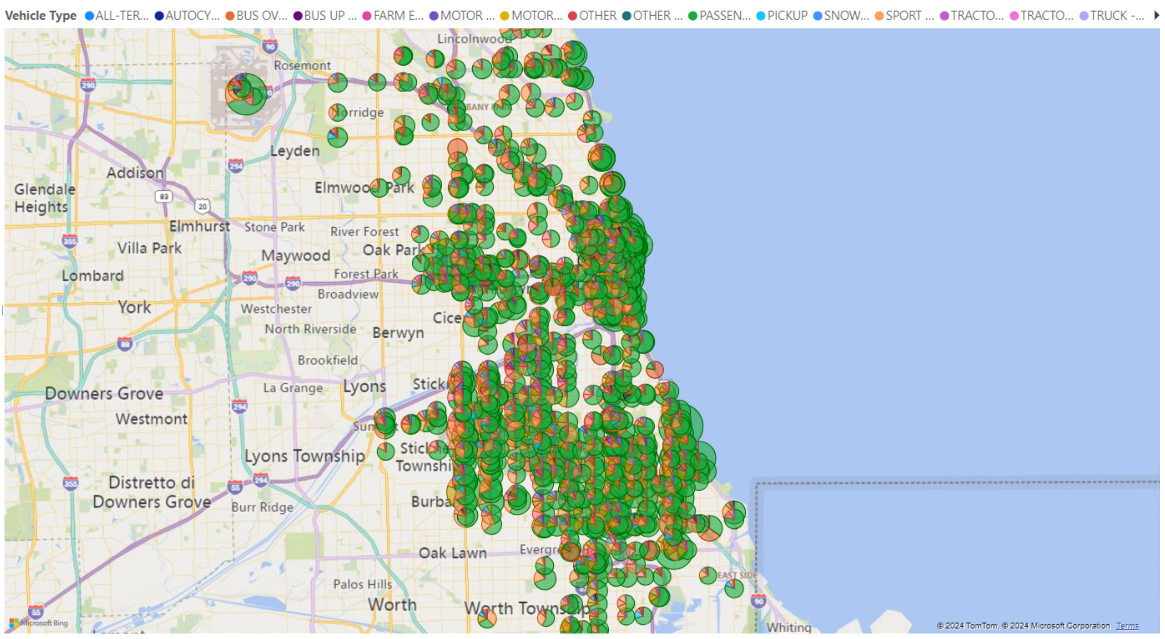

Figure 6: Damage geographical distribution based on Vehicle Category

8.2 Assignment 10

For the dashboard with respect to roads, we used several graphs. In the first one (i.e. the line graph in Figure 7) we analysed the trend in the number of people involved in an accident as the month changes, with a different line for each road condition. In the second (Figure 8) we used a donut and a stacked bar chart, to observe the distribution of Road Defect (donut graph) and how the distribution varies according to Speed Limit.

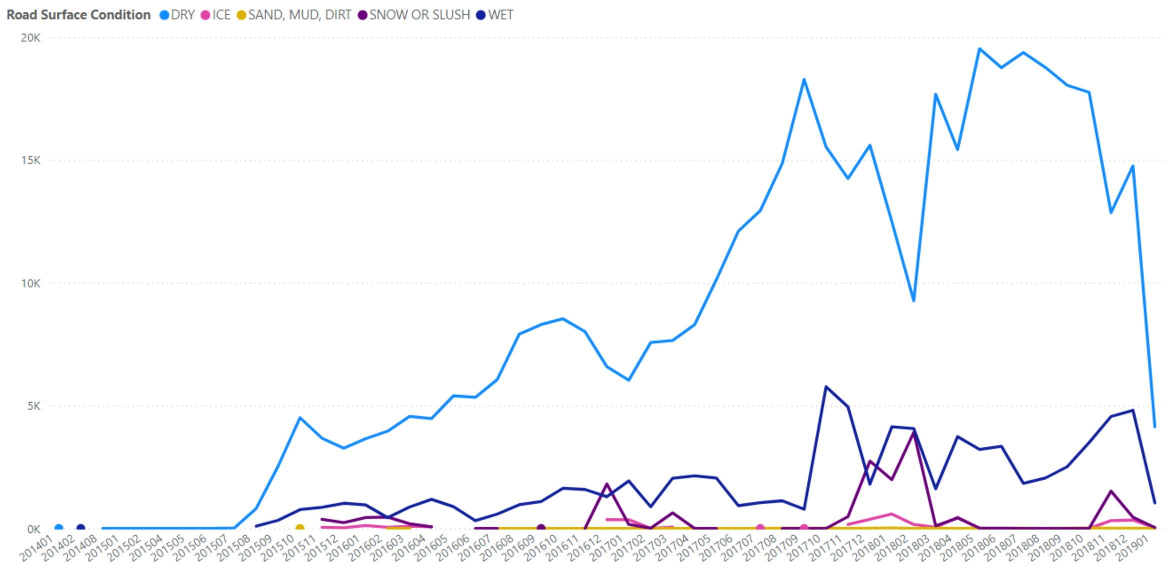

Figure 7: Number of people involved in a crash (y-axis) per Month (x-axis) based on Road Surface Condition

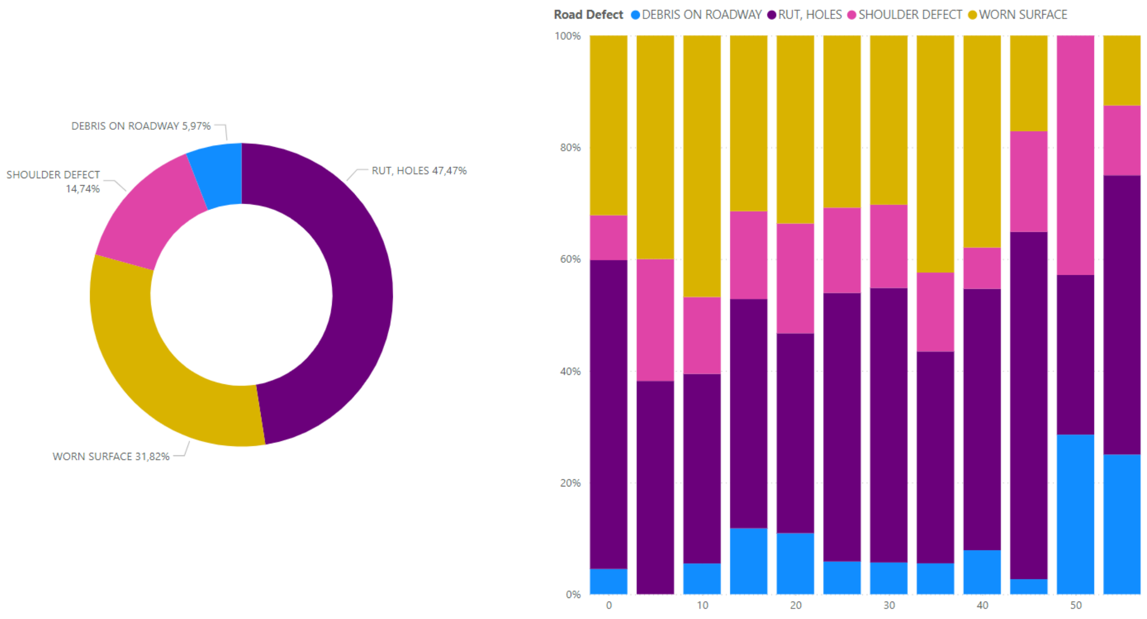

Figure 8: Percentage of person involved in a crash per Speed Limit and Road Defect

8.3 Assignment 11

For this dashboard, we also created several graphs. In the first visible graph (Figure 9), we can see a bubble map, where the size of each bubble represents the number of people from that state involved in an accident. Each bubble also contains information on the distribution of injuries for that specific state. Persons with no injuries were excluded, as they were not subject to analysis. As for states, Illinois was removed, as the high number of accidents unbalanced the scale of observation of the data. Finally, the states '*MS*', '*ME*' and '*NB*' were also excluded, as they were not correctly interpreted by PowerBI.

In the following graphs, i.e. the pie chart and tree-map (in Figure 10), one can see the physical conditions in relation to the years. From this visualization, we have eliminated some *Physical Condition* variables that we considered superfluous for the purposes of analysis: '*NORMAL*' and '*UNKNOWN*'.

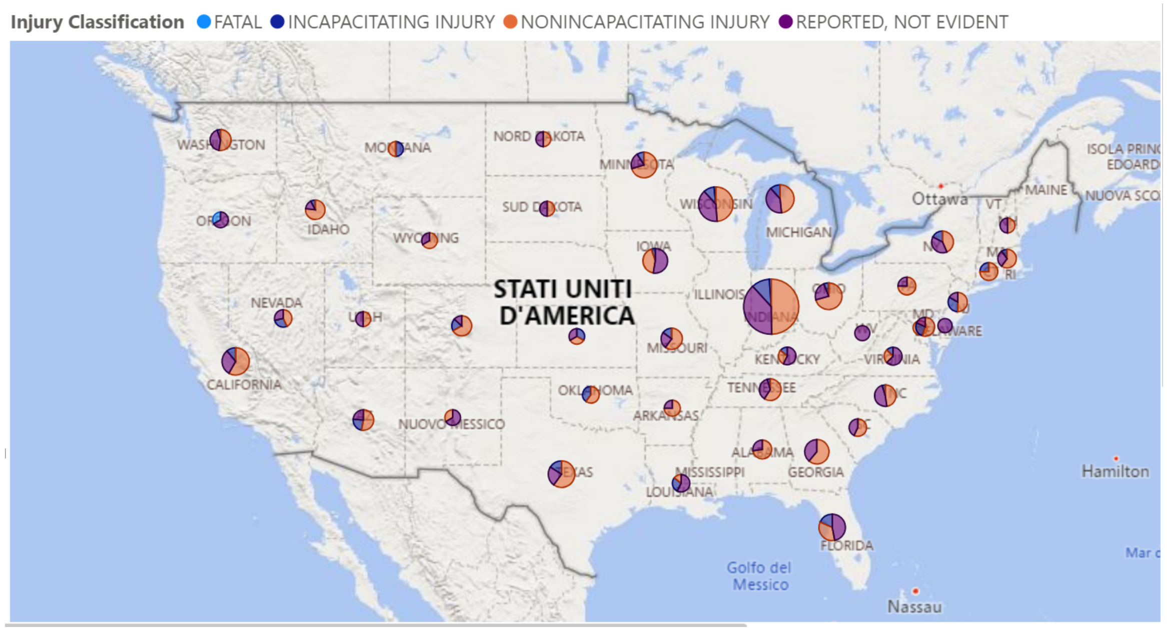

Figure 9: US State distribution of the number of people involved in a crash based on the type of Injury

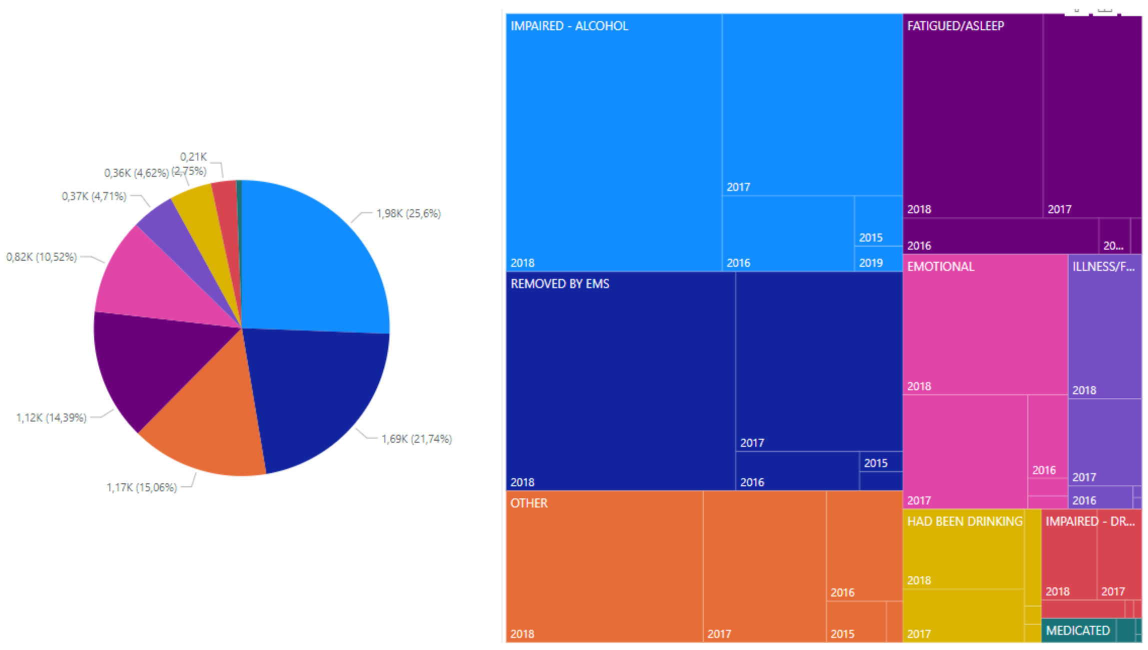

Figure 10: Physical condition by Year

These insights could help the business better understand and manage risks. Thanks to the trend analysis in Figure 7, for example, it could be possible to identify high-risk periods by showing how accident involvement varies with road conditions across different months. In this way the agency could adjust premiums or offer targeted advice during risky times.

The pie chart and the tree map (Figure 10) could offer insights into how physical conditions, such as fatigue or impairment, evolve over the years, allowing the business to design targeted awareness campaigns. Finally, knowing the state of origin of clients involved in accidents (Figure 9) is valuable for assessing risks related to drivers from specific areas. This could help the company to refine its risk profiling and tailor its services accordingly.