UNIVERSITÀ DI PISA

Decision Support Systems - Module II: Laboratory of data science project

Artesi S.* Incerti A.†

December 2023

*Master in Data Science & Business Informatics – s.artesi@studenti.unipi.it
†Master in Data Science & Business Informatics – a.incerti4@studenti.unipi.it

Contents

1 Introduction

This project comprises a series of tasks, divided into 3 parts, covering the entire business intelligence process. This includes the creation and population of a database, the development of an OLAP cube, and concluding with data visualization.

2 Part 1: Creation and population of the database

In this first part of the project, we are required to create and populate a database starting from different files and perform some operations on it. All details are described below.

2.1 Creation of the database schema using SQL Server Management Studio

For the solution of this first assignment, five tables were created using SQL Server Management Studio (SSMS): "Custody" (the fact table of the schema), "Gun", "Date", "Geography", and "Participant". Compared to the schema proposed in the exercise, which also included the Incident table, we chose to merge its only attribute, namely "Incident_Id", into the fact table. This choice was made to optimize the space occupied by the database, while still allowing the creation of the Incident table later if needed.
The tables were constructed using the same primary keys proposed in the exercise, and, as a rule, selecting the primary keys of each tables as foreign keys in the fact table.
In addition, we tried to make the maximum character space used in each column as efficient as possible where the attribute is represented by a string. This operation was possible by selecting a number less than 50 in the attribute type. *NVARCHAR(50)*. For each attribute, a reasonable number was chosen to prevent errors from occurring either immediately or in the future, considering the possible creation of new values to be inserted into the database.
Upon studying the presence or absence of null values in our initial CSV and noting that there are none, we selected "Non-Null" as a possible option for each attribute in each table. This option prevents the insertion of null (missing) values within the attributes.

2.2 Code description

Within this assignment, we are asked to divide the original dataset "Police.csv" into 6 separate tables and perform some operations on the data. Since these operations are necessary for the dataset division, we first carried out these operations and then created the aforementioned tables.
Firstly, we imported the "Police.csv" file into the program, storing its values within a list of lists: this data structure was chosen because a numpy array or a list of tuples would have been more computationally efficient, however the former cannot contain heterogeneous values, while the latter is not modifiable after its initialization.
Regarding the calculation of "participant_id", "gun_id", and "geo_id", we assumed not to know how many attributes referred to these IDs or in what position they were within the dataset: the only information we considered available was that each attribute relevant to a particular ID had the keyword of that ID in its metadata (for example, all attributes related to "participant_id" had the term "participant" in their metadata). In essence, we assumed to have as little information as possible to create a code that could provide suitable results in the most cases possible. Therefore, to understand how many and which attributes referred to "participant_id", we iterated through the metadata of the "Police.csv" table and saved in a dictionary, called "Dictionary_part_id", all metadata containing the word "participant" as the key, with the corresponding column index as its value. For "gun_id" and "geo_id", we followed a similar process, although for "geo_id", we searched for the features "latitude" and "longitude".
Subsequently, we created 3 tables: "part_id_table", "gun_id_table", and "geo_id_table", in which we inserted, without repetition, all combinations of attributes (captured within the respective dictionaries mentioned above) that we identified within "Police.csv". To each combination, we then assigned an ID equal to one less than its position within the corresponding table (the one less is due to the metadata, which occupies position 0 and will therefore have an index of -1 since it is not a combination eligible for an ID). To perform the operation just described, we iterated through each instance of the "Police" dataset. For each iteration, once we inserted the combination into the respective ID table, we added the ID value to the row corresponding to that iteration. In summary, this approach allowed us to entirely construct the tables containing information related to "participant_id" and "gun_id" and partially the one containing information related to "geo_id" (that will be completed later) through a single iteration of the "Police" table, thus achieving a computationally more efficient code.
To provide a solution that maintains a higher code readability, we created a function that takes as input the list of values to search for, the table within which to search, and the value (string) to be used as metadata for

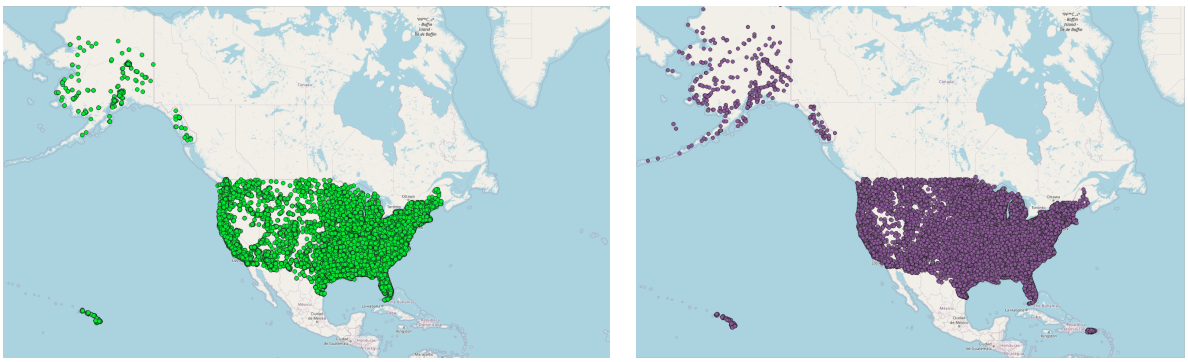
the column that will be added to the reference table (commented block within the code). This function returns an "id_table" containing all combinations of values related to the aspect we want to investigate, along with their respective IDs. As mentioned earlier, this function is a way to achieve the required result in tasks while maintaining a high degree of code readability and reusability. However, since we have to call the function for each table we want to create, it is less efficient than the first method described, which iterates through the table of interest ("police_table") only once, creating all 3 required tables. For this reason, for the purpose of the assignment, we decided to keep the more efficient code block.
Subsequently, we further iterated through the "Police" dataset, in order to calculate for each instance its corresponding "crime_gravity" as:

$$crimegravity(x) = F_1(x.participantage) \cdot F_2(x.participanttype) \cdot F_3(x.participantstatus)$$

Where F1 is a .json file containing a dictionary that associates each "participant_age" value with its corresponding value to be inserted into the "crime gravity" formula. Similarly, F2 and F3 contain dictionaries related to "participant_type" and "participant_status", respectively.
Even in this case, it would have been possible to include this calculation within that single iteration described above, but following an empirical test, we found that the time required to perform it is significantly low. Therefore, we decided to preserve an advantage in terms of code readability.
Once the data manipulation described above was performed, we proceeded with the creation of the Custody table: our fact table, which contains, for each "custody_id", the corresponding "participant_id", "gun_id", "geo_id", "date_id", "incident_id", and "crime_gravity".
To complete the creation of Geography table, we needed to locate the geographic coordinates in our possession. We proceeded in three ways:

- Initially, we attempted to locate the points using the QGIS (Quantum GIS) program, but it couldn't handle such a large number of inputs.
- Subsequently, we tested various Python libraries, such as geopy.geocoders and reverse_geocode. However, these proved to be not very efficient and, furthermore, either returned an incorrect city or were unable to fully locate it.
- We finally chose to search for a dataset of American cities and calculate the distance of our locations from those in the dataset.

This dataset was obtained from "simplemaps"[1], an organization that, in turn, compiled this information from U.S. government sources such as the U.S. Geological Survey and U.S. Census Bureau.
Once we obtained a dataset (in CSV format) containing, for each U.S. city, its respective longitude, latitude, and state of belonging, we considered the adequacy of the data. The dataset we were able to obtain contains about 30,000 records out of a total of approximately 109,000 U.S. urban settlements. Despite being a relatively small quantity, we deemed it sufficient for the purpose of our assignment, as upon observing these instances, we found a homogeneous distribution across the entire U.S. territory 1.



(a) *Our dataset* (b) *uscities dataset*

Figure 1: Comparison of the city distribution between the two datasets

We imported it into the program and reflected on any data preparation operations, such as handling missing values and duplicates. However, the only operation we deemed necessary was the removal of double quotation marks within some values and the deletion of a single instance that caused an error during dataset iteration because it did not have a numeric value but rather a string value (being only one, we decided to remove it).

[1] https://simplemaps.com/data/us-cities

To complete the creation of the "Geography" table, we assigned each pair of coordinates to their respective city and state. To do this, we calculated the Euclidean distance from each pair of coordinates in Custody to those in UsCities and for each pair of coordinates in "Custody", we selected the city in UsCities that was closest as its city of belonging.
Theoretically, to calculate the distance between two points on Earth, it would be more accurate to use geodetic distance, which considers the Earth's curvature. However, since we are aware that the UsCities dataset uniformly covers the American territory, leaving very little distance between the points of the two datasets, we chose to use Euclidean distance to simplify the calculations.
Once the construction of these tables was complete, we copied the data into their respective CSV files, specifically created for this purpose.
Finally, using the xml.etree.ElementTree module, we obtained the hierarchical structure of the "dates.xml" file and visualized its content. We found that within it, each primary key was assigned a specific date. Once we obtained these values, we also created the corresponding CSV file for this table.
Once the CSV files were correctly created, we proceeded with their loading into the database prepared in Assignment 0. First, we set the parameters for the connection and established it. Subsequently, for each table, we read the data within it and performed the upload. At the end of the process, the connection was closed.

3 Part 2: SSIS

In this section of the project, we are required to solve some problems on the database we create in the previous part using SQL Server Integration Services (SSIS) with computation on the client side.

3.1 Introduction

In all the assignments described below, the connection is established by placing a "Data Flow Task" node in the control panel. This allows access to the data flow panel, where nodes are positioned to complete various tasks. The initial node in each task is the "OLE DB Source," renamed as needed to indicate the starting table. To utilize the table, it is imperative to establish a connection with the server, select the database, and import the data into the program. Lastly, for visualization purposes, a "Flat File Destination" node is added to all queries, facilitating the printing of results to txt files, including .CSV or .txt in our case.

3.2 Assigment 0

1. Join between Custody and Date on date_id: Through the use of the Search tool, we performed a join operation between the Custody and Date tables, using date_id as the joining key.

2. part_id sorted by the number of custody: Using the Aggregation tool, we conducted two GROUP BY operations: the first based on year and the second on "participant_id". This operation generated a table where instances are initially grouped by year and subsequently by "participant_id". Within this tool, we then performed a count (COUNT) of "custody_id" for each combination of (year, "participant_id").

3. Sort by year and "participant_id": In order to obtain a list sorted in descending order of "custody_id" counts for each year, as specified, we utilized the sorting node (SORT) to arrange instances first based on the values of year and then based on the corresponding "custody_id" counts (both in descending order).

3.3 Assignment 1

1. Join between "Custody" and "Geography" on "geo_id": As this task involves linking information from the "Custody", "Gun", and "Geography" tables, in this initial step, we perform a join between "Custody" and "Geography" using "geo_id" as the common key.

2. Join with Gun on "gun_id": Subsequently: we perform the join between the table obtained in the previous step and Gun using "gun_id" as the key.

3. Multicast: Since calculating the Stolen Gravity Index requires analyzing two different quantities ("crime_gravity" for instances with "stolen_gun" and the total "crime_gravity"), we split the data into two distinct channels. Taking the table generated in the previous step as input, we output the same table on two different channels.

4. First output of the multicast:

    - Select rows with "stolen": We use the "Conditional Split" tool to obtain a table containing only instances with "Stolen" as the value of the "gun_stolen" attribute.

- Stolen gravity per state: Using the "Aggregation" tool, we calculate, for each state, the sum of the crime_gravity values for all instances in the obtained table. These values are entered into an additional column called "crime_gravity_sum," which will represent the numerator for the Stolen Gravity Index calculation.
- Sort by "state_name": We sort the obtained table alphabetically based on the state name of each instance.

5. Second output of the multicast:

   - Total gravity per state: We obtain the sum of the "crime_gravity" values for each state in the table output from the Multicast. This value will represent the denominator in the formula for calculating the Stolen Gravity Index.
   - Sort by state_name: Again, we sort the obtained table based on the value of "state_name".

6. Join on state_name: Using the "Merge Join" tool, we combine the two tables obtained in the previous branches based on the values of state_name.

7. Calculation of the Stolen Gravity Index: Now that we have both the total "crime_gravity" and that related to records with the "Stolen" value for each state, we calculate the Stolen Gravity Index. This step is performed using the "Derived Column" tool, which adds a column to the table containing the Stolen Gravity Index values for each state.

3.4 Assigment 2

1. Join between Custody and Date on DateID: Using the search tool, we perform a join between the "Custody" and "Date" tables on the "date_id" key.

2. Multicast: Similar to the previous assignment, we need to calculate two different measures: the "crime_gravity" for each month of each year and the total "crime_gravity" for the entire year. Therefore, we again use the Multicast tool to direct the input table to two distinct output channels.

3. First output of the multicast:

   - Total gravity per year: Using the Aggregate tool, we calculate the total "crime_gravity" for each year.
   - Sort by year: We sort the newly obtained table based on the "year" value.

4. Second output of the multicast:

   - Total gravity per month: We calculate, using the same procedure described in the previous steps, the "crime_gravity" for each month of each year (first grouping by "year" values and then by "month" values).
   - Sort by year: We sort the newly obtained table based on the "year" value.

5. Join on year: The two outputs from the Multicast described in point 2 have led to the calculation of two different measures, each present in its own table. We now join the two tables using the "year" attribute.

6. Percentage computation: At this point, we have obtained the two necessary measures and proceed with the calculation of the percentage as required by the assignment.

4 Part 3: Datacube creation and data visualization

In this phase of the project, our objective is to address specific business inquiries using a data cube on Analysis Services generated from the prepared database. Following this, we will formulate responses to business questions utilizing MultiDimensional eXpressions (MDX) within SQL Management Studio. Finally, we will leverage PowerBI for data visualization purposes.

4.1 Cube creation

First, we connected to the Group_ID_4_DB database, importing its structure into the program.
Upon importing the database, in the "Data Source Views" section, we had the opportunity to choose which tables from our database to integrate into the program. Not knowing in advance which tables would be necessary for our assignments, including the Data Visualization part, we opted to import all tables. This allows us to directly visualize the schema of the reference database within the program.
Subsequently, we created dimensions with their respective hierarchies. Initially, we created the "Date" dimension and selected the attributes "date_id", "Day", "Month", and Year. We then used all these attributes to create the "DayMonthYear" hierarchy, where the most general element is Year, further specifying through Month to Day, and finally to "Date_id". In the "Attribute Relationships" panel, we defined relationships between these attributes, connecting each attribute to "date_id".
Despite a potential decrease in query performance related to this dimension, we chose this hierarchy as it was not possible to define functional dependencies between the mentioned attributes. For instance, the same month can correspond to multiple different years and vice versa.
Similarly, we created the "Geography" dimension with the "CityState" hierarchy, where the most general element is State Name and becomes more specific through City to "Geo_id". We also created the "Participant" dimension with the "participant_h" hierarchy, where the most general element is "Participan_Age_Group", and the most specific is "Paricipant_Id".
Finally, through the browser panel, we verified that the hierarchies were correctly set up, for example, ensuring that Day is contained in Month and Month in Year.
For the cube creation, we selected the fact table, represented in this case by Custody: its foreign keys became dimensions, and non-key attributes became measures. Subsequently, we selected all the dimensions just created.
After creating the cube, we focused on Assignment 1, which involved manipulating "Crime_gravity". We inspected its properties, selecting "Integer" as the data type and "Sum" as the aggregation function, given its numerical measurements that can be useful for our analyses.

4.2 Query 1

```
-- Show the total crime gravity for each city and the grand total with respect to the state.

WITH member Crime_Gravity_State as
([Geography].[CityState].currentmember.parent, [Measures].[Crime Gravity])
SELECT
  {[Measures].[Crime Gravity], Crime_Gravity_State} ON COLUMNS,
  [Geography].[CityState].[City] ON ROWS
FROM [Group ID 4]
```

Figure 2: First query

Initially, we calculated a member named "Crime_Gravity_State." It is important to note the ".currentmember.parent" method, which, within the Select command, allows us to select an attribute (in this case, Crime Gravity) related to the parent of the currently analyzed element. In our specific case, when the query processes "Crime_Gravity_State" for a particular instance, it uses the parent of the "city" attribute presented in that instance, namely the state to which that city belongs. In the column section of the query, we use curly braces because, to display multiple values in the columns of the output table, we need to define a set rather than a tuple.

Below, we present an initial section of the table obtained in the output:

	Crime Gravity	Crime_Gravity_State
Abbeville	4	4261
Adamsville	1	4261
Alabaster	4	4261
Albertville	18	4261
Alexander City	32	4261
Andalusia	0	4261
Anniston	13	4261
Arab	12	4261
Ardmore	1	4261
Ariton	11	4261
Arley	0	4261

Figure 3: First query output

In the output table, we observe that for each city, the "crime_gravity" of the city itself is displayed along with the "crime_gravity" associated with the corresponding state.

4.3 Query 2

```
-- Show the percentage increase or decrease in total crime gravity answers with respect
-- to the previous year for each age group.

WITH MEMBER difference AS
([Measures].[Crime Gravity] - ([Date].[DayMonthYear].prevmember, [Measures].[Crime Gravity]))
/([Date].[DayMonthYear].prevmember, [Measures].[Crime Gravity]),
format_string="percent"

SELECT
difference on columns,
([Participant].[partecipant_h].[Participant Age Group],[Date].[DayMonthYear].[Year]) on rows
FROM [Group ID 4]
```

Figure 4: Second query

In this case as well, it was necessary to calculate a member ("difference"), where the ".prevmember" method is employed. This method is also related to a hierarchy, with the difference compared to before being that it does not refer to the "parent" but to the preceding "sibling" in the iteration of the analyzed table. Therefore, on the rows, the tuple composed of Age Group and Year will be displayed, while the "difference" value will appear on the columns.

		difference
Adult 18+	2013	inf
Adult 18+	2014	3229.33%
Adult 18+	2015	300.84%
Adult 18+	2016	48.13%
Adult 18+	2017	52.48%
Adult 18+	2018	-78.42%
Child 0-11	2013	-nan(ind)
Child 0-11	2014	inf
Child 0-11	2015	278.13%
Child 0-11	2016	-27.27%
Child 0-11	2017	30.68%
Child 0-11	2018	-73.04%
Teen 12-17	2013	inf
Teen 12-17	2014	7940.00%
Teen 12-17	2015	228.73%
Teen 12-17	2016	27.47%

Figure 5: Second query output

Note: Since data before 2013 is not available, the value of "difference" for the year 2013 will be infinite due to division by 0. A particular case concerns the "Child 0-11" group: as it has a Crime_Gravity equal to 0 in the year 2013, the "difference" value for this year will be undefined, while in 2014, it will be infinite.

4.4 Query 3

```
-- Show the ratio between the total crime gravity of each year w.r.t the previous year.

WITH member percentage_diff as
[Measures].[Crime Gravity] / ([Date].[DayMonthYear].prevmember, [Measures].[Crime Gravity]),
format_string="percent"

SELECT
percentage_diff on columns,
[Date].[DayMonthYear].[Year] on rows
FROM[Group ID 4]
```

Figure 6: Third query

In this context, we calculate a member called "percentage_diff," which represents the percentage difference between the Crime_gravity values of a specific year and the preceding year. Subsequently, we display this value in columns, with the years arranged on the rows.

	percentage_diff
2013	inf
2014	3958.82%
2015	383.18%
2016	142.48%
2017	154.00%
2018	22.01%

Figure 7: Third query output

The first row contains the value "inf" because, lacking a reference year to calculate the requested percentage, the value of the current year is divided by 0, resulting in an infinite value.

4.5 Power BI: Assignment 1

In this initial data visualization assignment, the goal was to create a dashboard showcasing the geographical distribution of "Crime Gravity" for each age group.
To represent this distribution concerning states and age groups, we opted for using a map highlighting the geographical area of interest. Subsequently, we incorporated a column chart displaying, based on the selected state in the aforementioned map, the distribution of "Crime Gravity" for each age group.
Finally, we introduced a line chart to visualize the distribution of "Crime Gravity" in each state. In the event of selecting a specific "age_group" in the preceding column chart, the line chart will reflect the distribution of "Crime Gravity" for each state, corresponding to the selected age.
This approach allows for observing the distribution relative to a particular age group or a specific state, depending on the preferences of the report viewer.

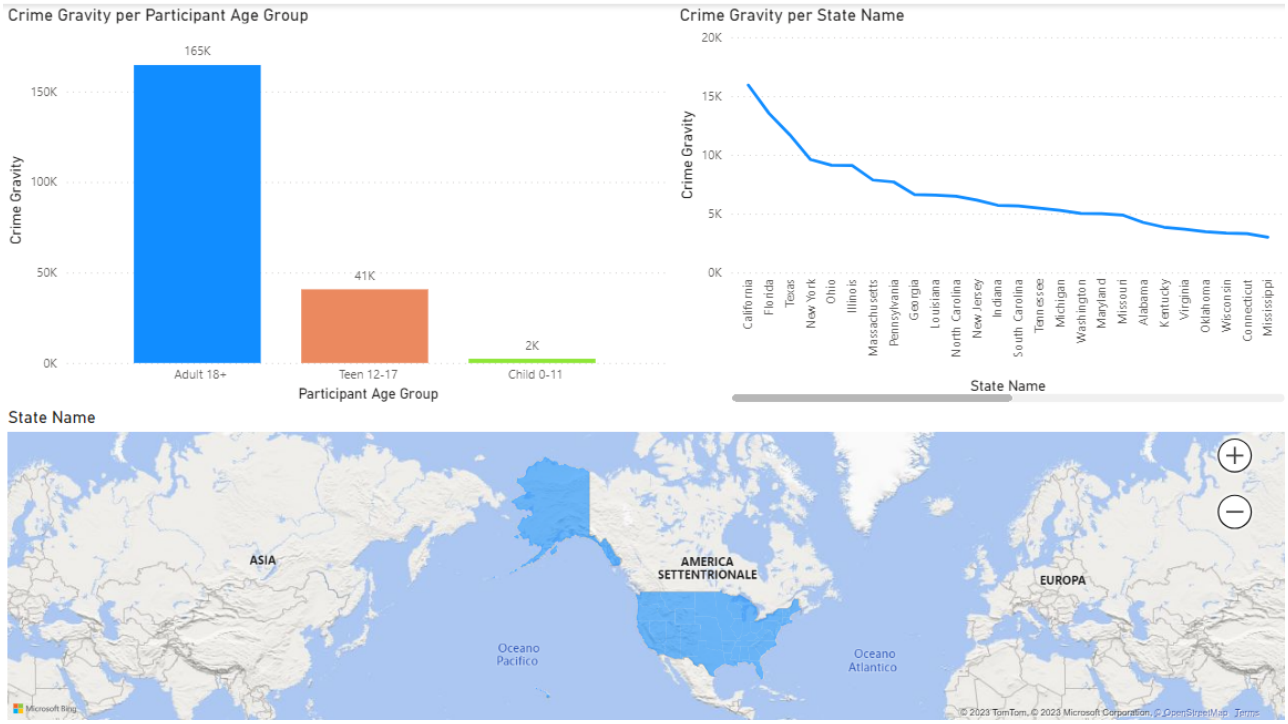


Figure 8: Geospatial Analysis of Crime Gravity Across Age Groups Dashboard(None state selected)

As highlighted in Figure 8, and as one would expect, there is a significant difference in "Crime Gravity" based on the age group. Furthermore, it is observed that the same measure for the first four states in the graph is considerably higher than the average. When considering the population of each state (data not available in our cube), we noticed that these four states are the most populous. Therefore, for a comprehensive analysis, it would be necessary to correlate the population of each state with the total "Crime Gravity."

4.6 Power BI: Assignment 2

In this second assignment, we chose to create a dashboard that relates "Crime Gravity" to participants' gender, the "YearMonthDay" hierarchy, and details about the type of weapon used. For the creation of the dashboard, as depicted in Figure 9, we included a map highlighting the American states of interest. Subsequently, we added two more graphs that display, in a line chart, how the measure varies based on the year, month, day hierarchy, and in the other based on gender, weapon type, and whether the weapon was stolen or not.

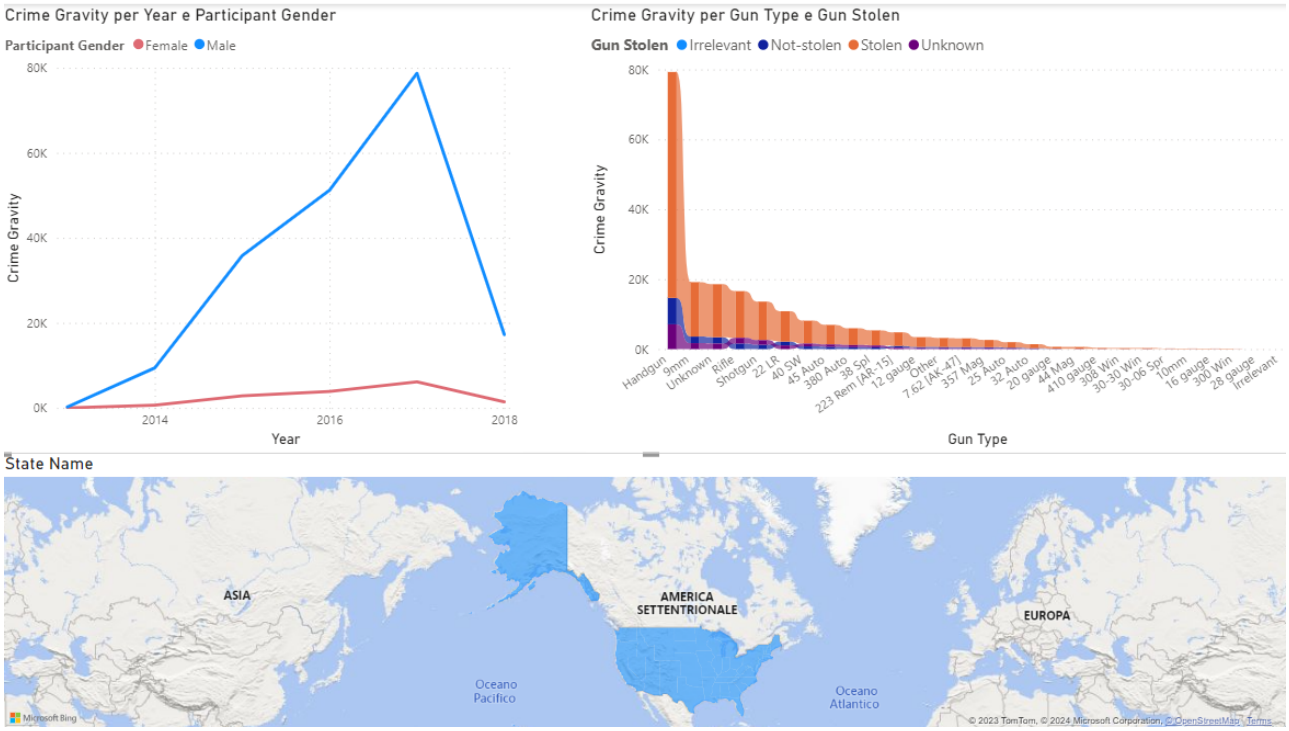


Figure 9: Crime Gravity in relation to participant gender and the type of weapon used

As highlighted in the first chart, over the years, there has been a general increase in "Crime Gravity," especially for the male gender, which exhibited a consistent surge from 2014 to 2017. Concerning the months, there is a higher "Crime Gravity" for the first three months, followed by a decrease and substantial stability throughout the rest of the year.
In the adjacent chart, there is a notable spike for the "Handgun" weapon, and overall, a higher "Crime Gravity" is observed in the case of a stolen weapon.