Data Mining Project Report

Irene Dovichi
Giacomo Lagomarsini
Alice Nicoletta

A.Y. 2023/2024

Contents

1 Data Understanding

The first phase of the project is the data understanding as it involves gaining insights into the nature, structure, and content of the data that will be used for analysis. We have 3 datasets available: one contains data on gun incidents in the USA, another contains data on state poverty over the years, and the last contains information on congressional elections. We have 239677, 884, and 10441 records, respectively.

1.1 Features Description

The first task performed is to understand the semantics of each feature and cast them into the correct data type. The type casting operation was carried out where there were incorrect or generic data types. In Table 1, Table 2, and Table 3 we show the initial data type, the cast type, and a brief description of the attributes for each dataset.

1.2 Incoherent Data

We proceed with processing the data that are at first sight incorrect.

1.2.1 Incidents Dataset

We noticed that some incidents occurred in the **years** 2028-2030 which is certainly impossible today; we decided to set the incorrect dates as unknown (`None`). Then, we noticed that 5 incidents had the **longitude** feature of opposite sign and were in China, so we reversed the sign. We then examined the **ages** of the participants (features: `participant_age1`, `min_age_participants`, `max_age_participants`, `avg_age_participants`), imposing that they fell between 0 and 100, and putting `None` when this was not true. Moreover, we found that the minimum and maximum number of **participants** in the incidents were 0 and 103, respectively. We then proceeded to set the values outside this range to null for the features `n_participants_child`, `n_participants_teen`, and `n_participants_adult`.

1.2.2 Elections Dataset

In the original dataset there were 6 different parties, which we reduced to the 3 classes 'DEMOCRAT', 'REPUBLICAN', and 'INDEPENDENT'. We also identified 3 incorrect values for the `candidatevotes` and `totalvotes` features: -1, 0, and 1, which we set to null.

1.3 Duplicate Data

We found duplicate rows only in the incident dataset (365 instances), and we left only one copy for each duplicate data.

1.4 Missing Values

Another major part of dataset analysis is dealing with missing data. We looked at the percentage of missing values for each feature and at the heatmap of the DataFrames: in this way we were able to understand which features have the same rows with missing data.

Features of Incidents DataFrame			
Feature Name	Initial Type	Cast Type	Description
date	`object`	`Datetime`	Date of incident occurrence
state	`object`	`String`	State where incident took place
city_or_county	`object`	`String`	City or county where incident took place
address	`object`	`String`	Address where incident took place
latitude	`float64`	`float64`	Latitude of the incident
longitude	`float64`	`float64`	Longitude of the incident
congressional_district	`float64`	`int64`	Congressional district where the incident took place
state_house_district	`float64`	`int64`	State house district
state_senate_district	`float64`	`int64`	State senate district where the incident took place
participant_age1	`float64`	`int64`	Exact age of one (randomly chosen) participant in the incident
participant_age_group1	`object`	`String`	Exact age group of one (randomly chosen) participant in the incident
participant_gender1	`object`	`String`	Exact gender of one (randomly chosen) participant in the incident
min_age_participants	`object`	`int64`	Minimum age of the participants in the incident
avg_age_participants	`object`	`float64`	Average age of the participants in the incident
max_age_participants	`object`	`int64`	Maximum age of the participants in the incident
n_participants_child	`object`	`int64`	Number of child participants 0-11
n_participants_teen	`object`	`int64`	Number of teen participants 12-17
n_participants_adult	`object`	`int64`	Number of adult participants (18+)
n_males	`float64`	`int64`	Number of males participants
n_females	`float64`	`int64`	Number of females participants
n_killed	`int64`	`int64`	Number of people killed
n_injured	`int64`	`int64`	Number of people injured
n_arrested	`float64`	`int64`	Number of arrested participants
n_unharmed	`float64`	`int64`	Number of unharmed participants
n_participants	`float64`	`int64`	Number of participants in the incident
notes	`object`	`String`	Additional notes about the incident
incident_characteristics1	`object`	`String`	Incident characteristics
incident_characteristics2	`object`	`String`	Incident characteristics (not all incidents have two available characteristics)

Table 1: Features overview for the Incidents DataFrame.

Features of Poverty DataFrame			
Feature Name	Initial Type	Cast Type	Description
state	`object`	`String`	-
year	`int64`	`int64`	-
povertyPercentage	`float64`	`float64`	Poverty percentage for the corresponding state and year

Table 2: Features overview for the Poverty DataFrame.

Features of Elections DataFrame			
Feature Name	Initial Type	Cast Type	Description
year	`int64`	`int64`	-
state	`object`	`String`	-
congressional_district	`int64`	`int64`	-
party	`object`	`String`	Winning party for the corresponding congressional district in the state, in the corresponding year
candidatevotes	`int64`	`int64`	Number of votes obtained by the winning party in the corresponding election
totalvotes	`int64`	`int64`	Number total votes for the corresponding election

Table 3: Features overview for the Elections DataFrame.

1.4.1 Incidents Dataset

Some values are missing for the **latitude** and **longitude** features, but we have all the values of the `city_or_county` feature. So, we decided to substitute these missing values with the mean of the latitude and longitude of the other incidents that occurred in the same city/county. However, in 172 cases we couldn't do that because there weren't other incidents that occurred in that city/county; in these cases we opted to set the two attributes as the mean of the entire dataset.
We also noticed that we could retrieve some information about the number of **unharmed** people looking at the total number of participants and at the number of injured and killed participants, since for these features there are not missing values. In particular, we set that the number of unharmed people in an incident is equal to the total number of participants minus the killed and injured:

$$N_{unharmed} = N_{participants} - N_{killed} - N_{injured}.$$

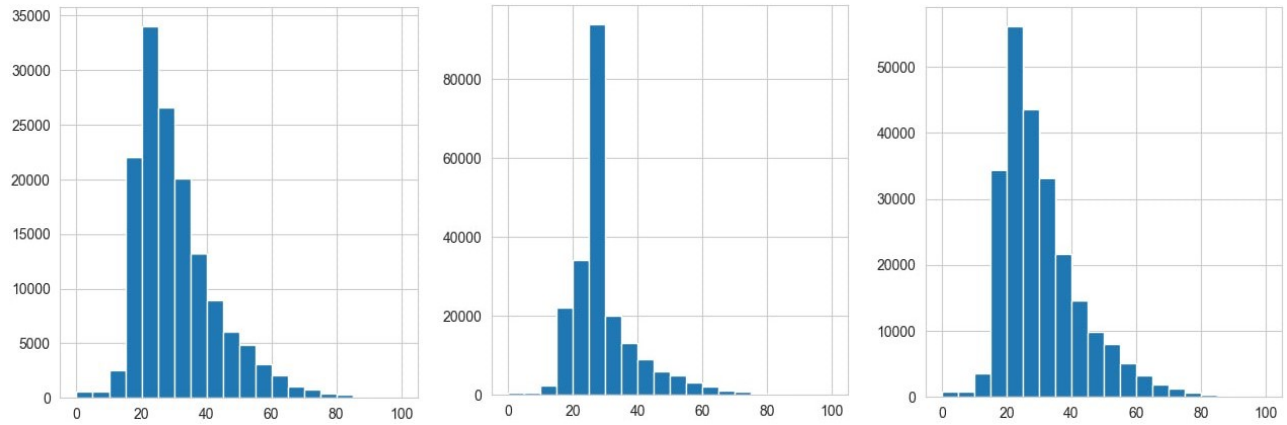
This choice was made because it seemed reasonable to us that every person involved was either unharmed or injured or killed, and furthermore more than 60% of the non-null instances satisfied this condition.
It was not trivial to deal with the **age** attributes (i.e., `min_age_participants`, `avg_age_participants`, `max_age_participants`, and `participant_age1`) given that 38% of the data was missing. We only considered the rows with `n_participants` > 0 and filled the null values for the ages with the mean and median. However, the distributions became completely unbalanced (peaking around the mean/median, see Figure 1b) and so we followed a different approach by focusing on `avg_age_participants`:

1. For each incident we counted the number of participants on the basis of the age groups, adding the columns `n_participants_child`, `n_participants_teen`, and `n_participants_adult`.

2. For each incident we could then calculate the percentage of participants in each age group.

3. We defined the distribution of each age group using a Series where:
    - the *values* were the unique ages in the subset of the incidents that had the `avg_age_participants` in the given age group
    - the *counts* were the number of occurrences of each age normalized by the total number of occurrences in the subset.
4. For each age group we sampled an age from the distribution (each *value* was drawn with probability given by the *count*).
5. The average age of the participants was computed as the weighted mean of the 3 drawn ages, with weights given by the percentages computed at Step 2.

In Figure 1c we can see the distribution of `avg_age_participants` after having carried out this filling operation.



(a) (b) (c)

Figure 1: Distribution of the feature `avg_age_participants` in the original dataset (1a), after substituting the missing values with the mean (1b), and after the replacement based on the age groups that we have described (1c).

Regarding the **gender** of the participants, we calculated the ratio of males and females over the sum of males and females; then, we considered the incidents with `n_participants` > 0 and missing values for `n_males` and `n_females` (these two features share the rows with null values). For each participant in the incident we associate the male/female gender with probabilities given by the calculated ratios. We decided to set the number of **arrested** people to 0 in the missing values, since we don't have any information about this feature and we can't infer it from the others.
Finally, we dropped the rows with `n_participants` $= 0$ because they were not too many and had too many missing values.

1.4.2 Poverty Dataset

In the poverty dataset the only missing data are in the **povertyPercentage** column and correspond exactly to the year 2012. We decided to fill this data, for each state, with a linear interpolation.

1.4.3 Elections Dataset

The elections dataset had some missing values in the **candidatevotes** and **totalvotes** columns, and we filled them with the average values of these features for the same state and district.

1.5 Dropping Useless Columns

We decided to remove some attributes that we would not have used in the study. We eliminated the **state_house_district** and **state_senate_district** features because they give geographical information similar to `congressional_district`; given that this last feature is present in the elections dataset too, we keep it. Then, the two attributes **participant_age_group1** and **participant_gender1** are redundant because we have other attributes that give a more precise information about age groups and genders. Finally, we dropped the **notes** and **incident_characteristics2** columns since there were too many missing values (34% and 41% respectively), and the `incident_characteristics1` feature is pretty representative by itself.

2 Data Preparation

2.1 Joining Datasets

We decided to join the incidents and poverty datasets in order to have, associated to each incident, the poverty percentage of that state in the year the incident occurred. Then, we added the elections dataset to have, for each incident, also the data related to the election happened in the year, state, and congressional district of the incident. Note that we have the election data just for even years, so we decided to set the values corresponding to the previous even year for odd years.

2.2 New Indicators

Examining the available data we decided to define new features:

- **severity**: this value is indicative of how serious the accident was. We considered the ratio of killed and injured participants with respect to the sum of killed, injured and unharmed participants in the incident. Then, we defined the severity as the weighted mean of these ratios (weight 0.7 for the killed and 0.3 for the injured).
- **severity_minor**: weighted mean of the ratios of children and teens involved with respect to the sum of children, teens and adults involved in the incident. We assigned the weight 0.7 to children, and 0.3 to teens.
- **killed_by_district**: ratio of killed people in the incident with respect to the total killed people in the same congressional district (and state) in the same year-month. We impose value 0 if the denominator is 0.
- **injured_by_district**: ratio of injured people in the incident with respect to the total injured people in the same congressional district (and state) in the same year-month. We impose value 0 if the denominator is 0.
- **female_ratio**: ratio of females with respect to the total number of participants in the incident.
- **arrested_ratio**: ratio of arrested people with respect to the total number of participants in the incident.
- **unharmed_ratio**: ratio of unharmed people with respect to the total number of participants in the incident.
- **right_or_left**: a number between 0 and 1 that tells which was the winning party (Democrat/Republican) and with what percentage it won. We calculated the ratio of voters of the

party of the candidate who won the election in the district w.r.t. the total number of voters in the district. If the winner is a Republican, we assign this ratio to `right_or_left`, otherwise we assign 1 - the ratio.

- **voters_over_population**: ratio of the number of voters in the state with respect to the total population in the state for that year.

2.2.1 Distance from the mean of the district

We also defined other indicators by computing the distance of a certain feature from the mean of that feature. We did it for the number of **killed** and **injured**, for the **females** and **males**, and for the number of **adults**, **teens**, and **children** participating.
Moreover, we considered the mean of these features in each state: in this way we were able to calculate the distance between each feature and the average of that feature in the state in which the incident occurred.
Going even more specifically, we considered the distance between each feature and the mean of that feature in the district where the incident occurred.
As one might expect, looking at the correlations of the distances from the global mean, the mean of the state, and the mean of the district, we found out that these three are almost equal for each feature. We looked at the entropy to decide which one to keep: it turned out that the distance from the mean of the district has higher entropy, therefore we kept it.

2.3 Correlation of indicators

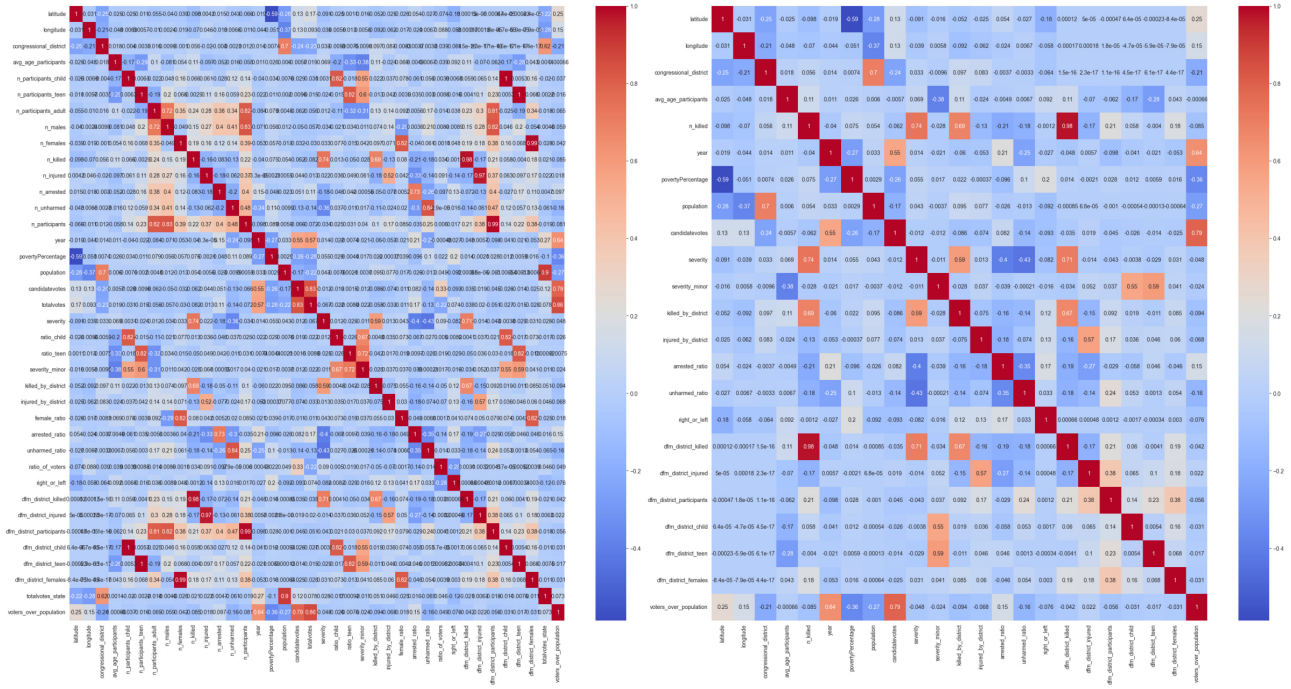
In Figure 2a we can see the correlations between all the features: those of the original dataset and the new indicators. We analyzed pairs with correlation > 0.6 and decided to drop some features among these: we ended up with 32 features with correlations shown in Figure 2b. Finally, we eliminated the (very few) rows that still had missing values.

2.4 Feature Engineering

To simplify the subsequent tasks we have created a new feature **age_group** which consists in the discretization of the average age of the participants. We divided the ages between 0 and 100 into 9 classes, and each incident was assigned to a class on the basis of the value of `avg_age_participants`. Moreover, we decided to reduce **incident_characteristics1** to just 5 labels: the 4 most common and a new one called 'Other' which includes all the others.

3 Clustering

In this Section we are going to explore different clustering algorithms: K-Means, DBSCAN, and hierarchical clustering. The last two techniques are computationally expensive, so we restricted ourselves to just one state: California, which we chose because it is the most populated state.
We focused on the capability of the algorithms to capture the severity of the incidents in each cluster, studying the distribution of the feature `incident_characteristics1` in each cluster and looking at the scatter plots of some relevant features.
For all these techniques we tried both the StandardScaler and the MinMaxScaler normalization techniques, and we always obtained better results with the MinMaxScaler.



(a) (b)

Figure 2: Correlation matrix for all the features (2a) and for the chosen ones (2b).

3.1 K-Means

Starting from the dataset obtained during the Preparation phase, we selected the features to use for the clustering based on correlations. We chose:

- `avg_age_participants`
- `povertyPercentage`
- `severity`
- `severity_minor`
- `killed_by_district`
- `injured_by_district`
- `arrested_ratio`
- `unharmed_ratio`
- `right _or_left`

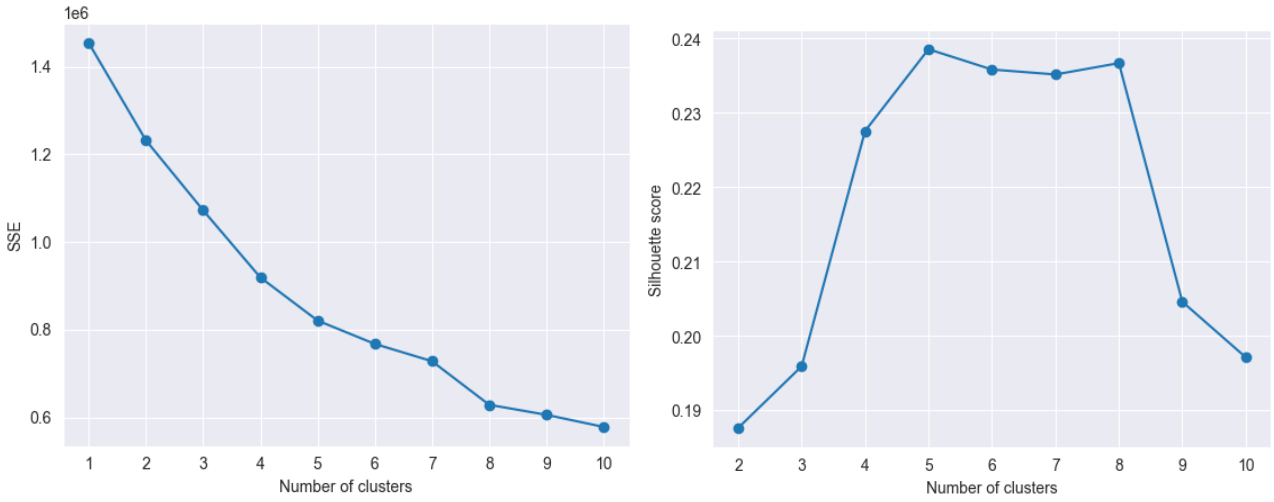
3.1.1 StandardScaler

We used the elbow method and the silhouette score (on a subsample of the dataset) to select the number k of clusters. We obtained the plots in Figure 3 and we chose $k = 5$ because it was located on the elbow and for that value of k the silhouette is maximum. We can visualize the 5 clusters obtained on the 2-PCA plot in Figure 4.

3.1.2 MinMaxScaler

We performed the same analysis to choose the best k for the dataset scaled with MinMaxScaler (Figure 5), which led to taking $k = 4$. Notice that the silhouette score for this k is 0.36 which is not very high, meaning that there are not well-defined natural clusters in the dataset; however, the result is better than with StandardScaler.
We can visualize the distribution of killed people and of the feature `incident_characteristics1` in each cluster to gain some insights (Figure 6). We deduce that Cluster 3 captures the most severe

incidents, while in the Cluster 0 we have incidents with injured people, in Cluster 1 we find minor accidents, and Cluster 2 is a mix of all of them.



(a) (b)

Figure 3: K-Means with StandardScaler: SSE (3a) and silhouette scores (3b) for $k \leq 10$.

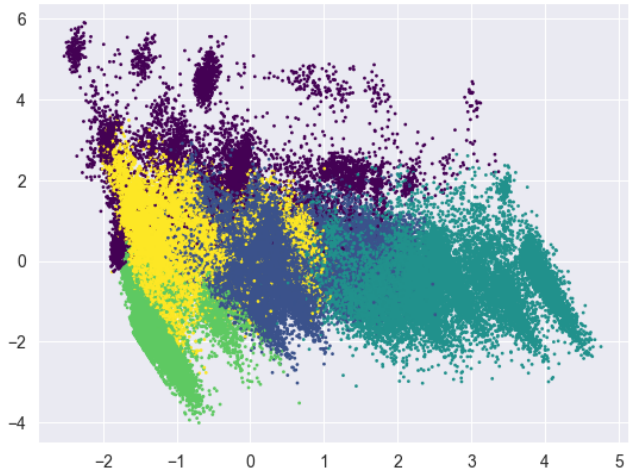


Figure 4: K-Means with StandardScaler: clusters on the 2-PCA.

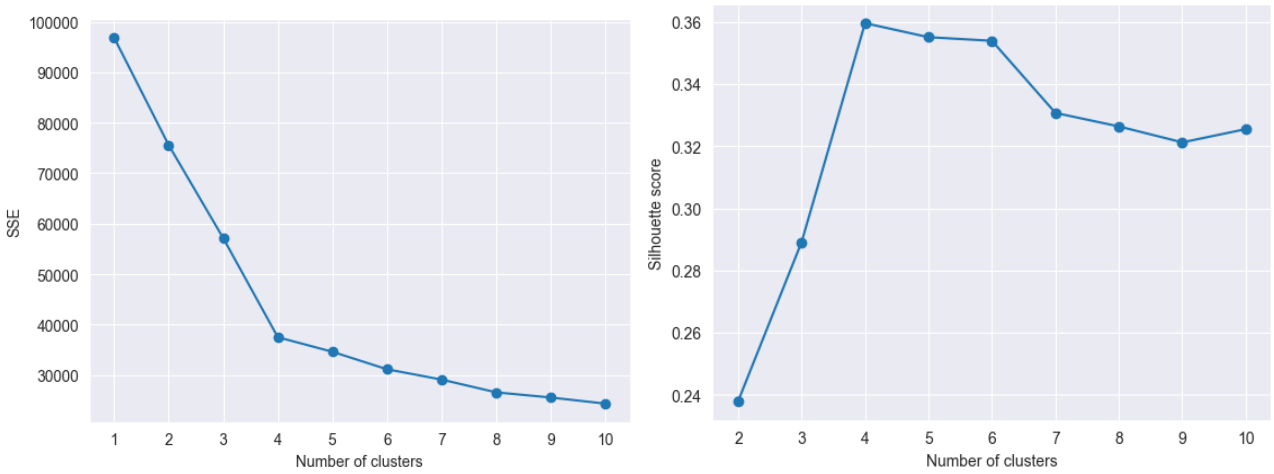
3.1.3 Comparison Between Scalers

In addition to obtaining a higher silhouette value, the MinMaxScaler was also better based on the Calinski-Harabasz index. This index is defined as:

$$CH = \frac{BCSS}{WCSS} * \frac{k-1}{n-k}$$

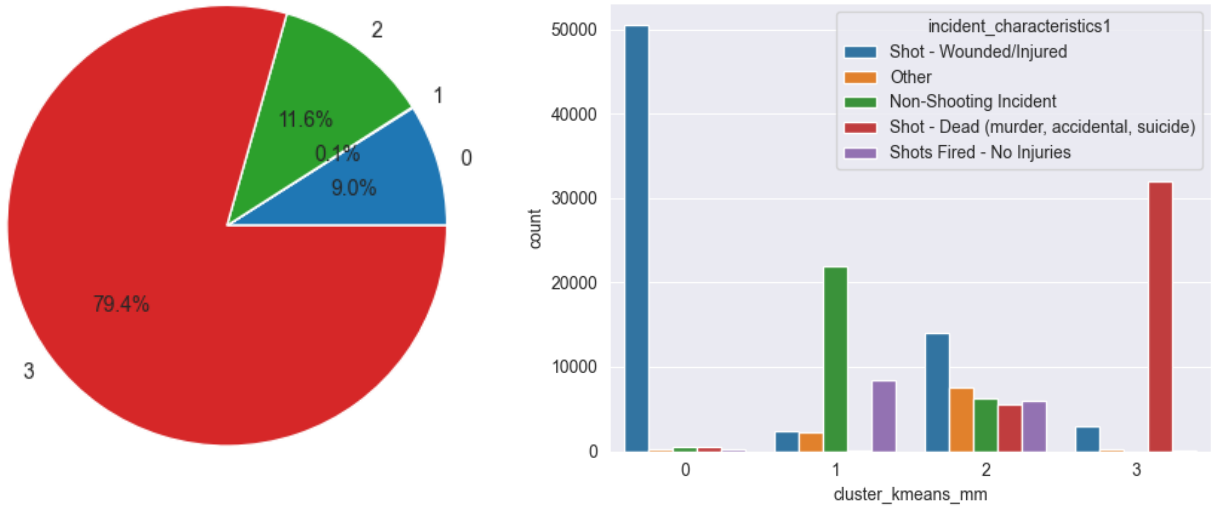
where:

- $BCSS$ is the between-cluster sum of squares, i.e., the weighted sum of the squared Euclidean distances between each cluster centroid and the overall dataset centroid, where the weights are given by the number of instances in the cluster



(a) (b)

Figure 5: K-Means with MinMaxScaler: SSE (5a) and silhouette scores (5b) for $k \leq 10$.



(a) (b)

Figure 6: K-Means with MinMaxScaler: percentage of killed in each cluster (6a) and distribution of `incident_characteristics1` (6b).

- $WCSS$ is the within-cluster sum of squares, i.e., the sum of the squared Euclidean distances between each instance and its cluster centroid
- k is the number of clusters
- n is the number of instances

and it came out 32256 for the StandardScaler, and 85271 for the MinMaxScaler. The higher the CH, the better the clusters; indeed, it means that the clusters are dense and well separated, which relates to a standard concept of a cluster.

3.2 DBSCAN

As announced we will focus on California and we selected the following features:

- `avg_age_participants`
- `severity_minor`
- `injured_by_district`
- `severity`
- `killed_by_district`
- `right _or_left`

3.2.1 StandardScaler

We need to select the parameter `eps` that defines the neighborhood of a point. In order to do so, we plotted the k-distance graph for some values of k, i.e., the plot of the distance of each point from its k-th nearest neighbor. We then chose 0.25 as the value of `eps` since it corresponded more or less to the elbow of these graphs. Moreover, we picked `min_samples` $= 10$.
This method produced 40 clusters, the largest being that of noise points (4270 elements), followed by cluster 0 (3363); the other clusters have between 700 and 5 elements, so they are rather unbalanced. In Figure 7 we have the clusters on the 2-PCA plot of California.

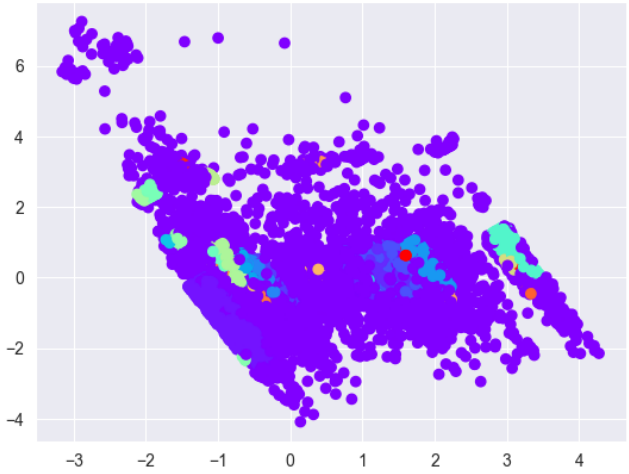


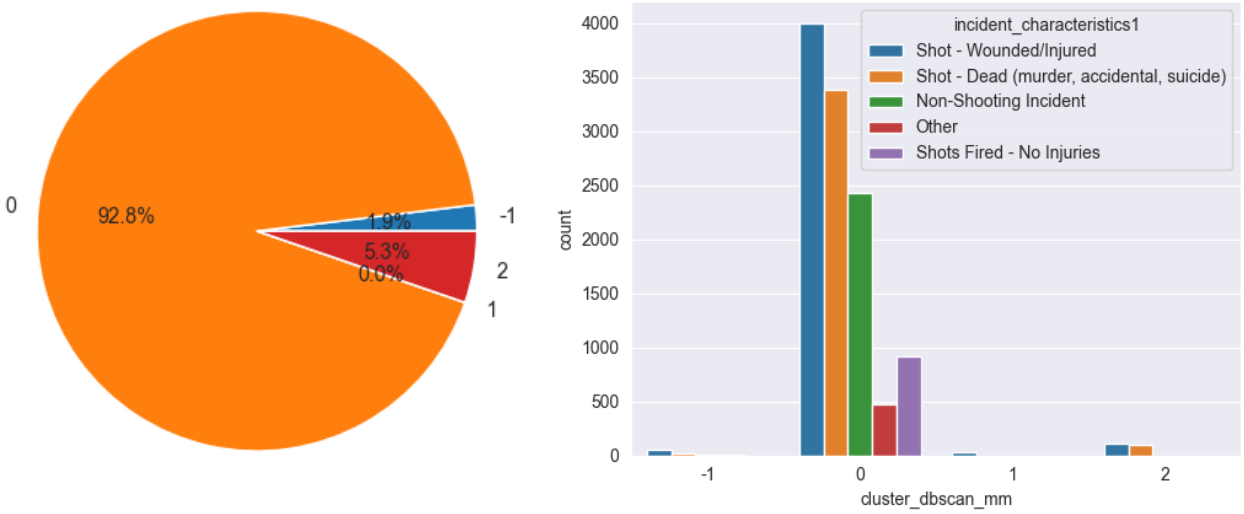
Figure 7: DBSCAN with StandardScaler: clusters on the 2-PCA.

3.2.2 MinMaxScaler

We repeated the same process to choose the parameter `eps`, which led to set again `eps` $= 0.25$; we chose as well `min_samples` $= 10$.
This time 4 clusters emerged: Cluster 0 contains the majority of elements (11208), followed by Cluster 2 (202), then there are 84 noise points and 34 instances in Cluster 1. The reason why there were far fewer noise points is probably because MinMaxScaler scales the data to the range $[0, 1]$, so the points were more concentrated in space.
We show in Figure 8 the number of killed people in each cluster, and the distribution of `incident_characteristics1`.

3.3 Hierarchical Clustering

We consider different configurations of linkage methods and metrics for the hierarchical clustering in the state of California, using the same features as in the previous section.



(a) (b)

Figure 8: DBSCAN with MinMaxScaler: percentage of killed in each cluster (8a) and distribution of `incident_characteristics1` (8b).

3.3.1 StandardScaler

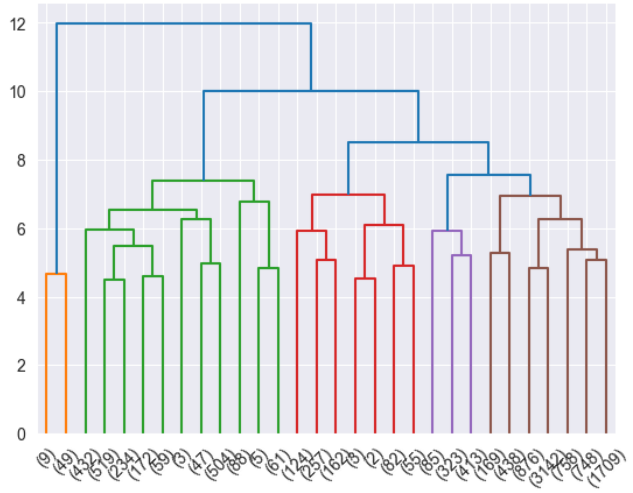
We tried 5 different combinations of methods and metrics, for which we plotted the dendrograms in Figure 9. We selected the number of clusters (choosing the parameter `color_threshold`) looking for a compromise between the height of the intervals in the diagram and the number of clusters. The configurations in Figure 9 are:

1. **complete** linkage, **euclidean** distance, **5** clusters
2. **complete** linkage, **cosine** distance, **7** clusters
3. **ward** linkage, **euclidean** distance, **4** clusters
4. **average** linkage, **euclidean** distance, **4** clusters
5. **average** linkage, **cosine** distance, **4** clusters

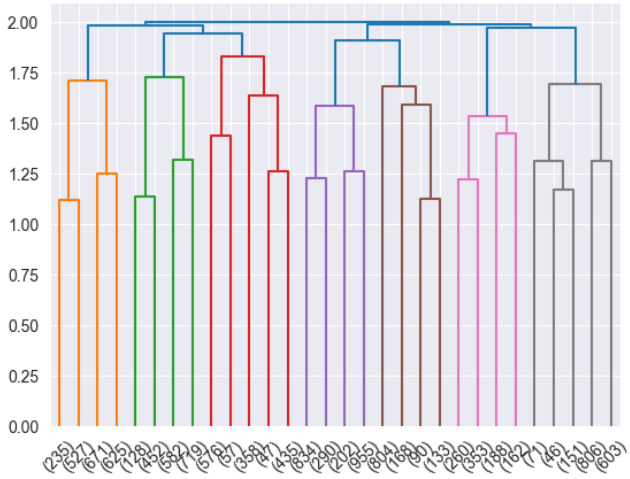
We also tested the **single** linkage method, but we didn't report its dendograms since they were not very informative.
We calculated the silhouette score for each configuration, and the one that performed the best (0.40) was the fifth: **average** linkage, **cosine** distance, **4** clusters. We show in Figure 10 the clusters obtained with this method on the 2-PCA plot of California.

3.3.2 MinMaxScaler

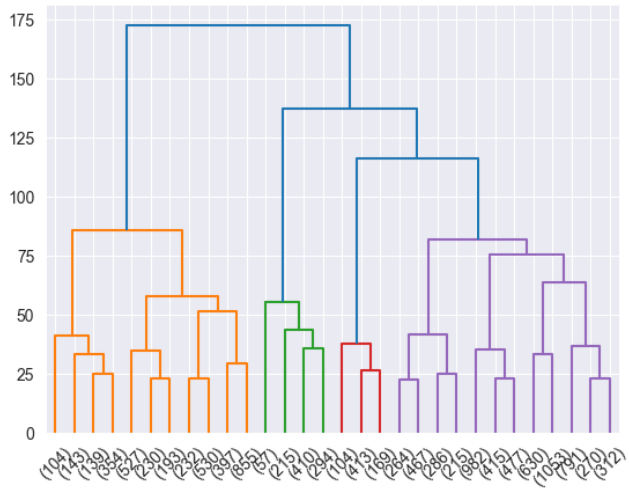
We tried the same 5 configurations as in StandardScaler, except that the number of clusters changed. Moreover, we calculated the silhouette score for each configuration, and the one that performed the best was the fifth: **average** linkage, **cosine** distance, **5** clusters. This configuration had a score of 0.50, which is higher than what happened for the StandarScaler since in that case it was 0.40. Figure 11 shows the dendogram obtained for this model.



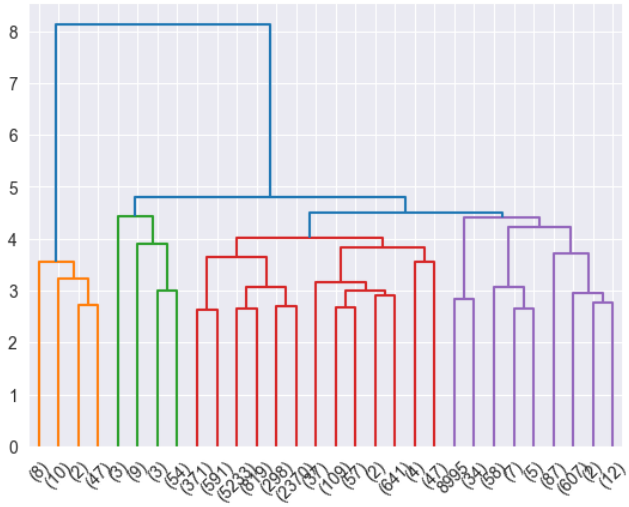
(a) Complete, euclidean, 5



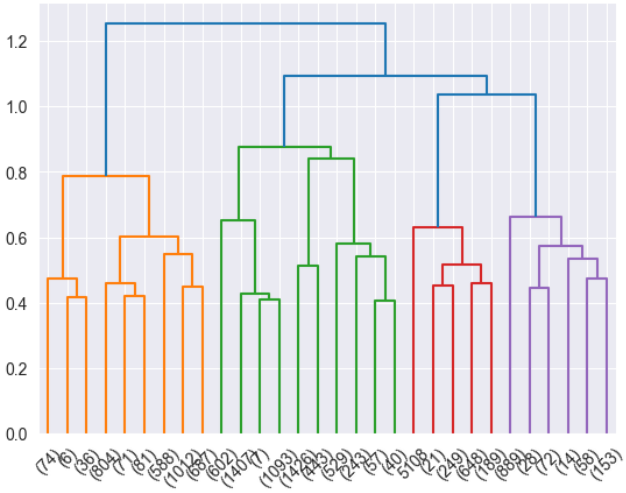
(b) Complete, cosine, 7



(c) Ward, euclidean, 4



(d) Average, euclidean, 4



(e) Average, cosine, 4

Figure 9: Hierarchical clustering with StandardScaler: dendograms.

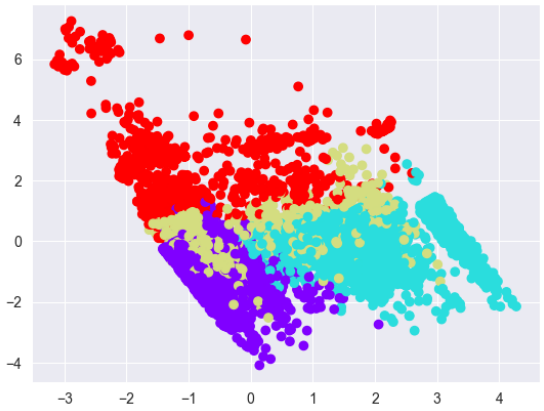


Figure 10: Hierarchical clustering with StandardScaler: clusters on the 2-PCA.

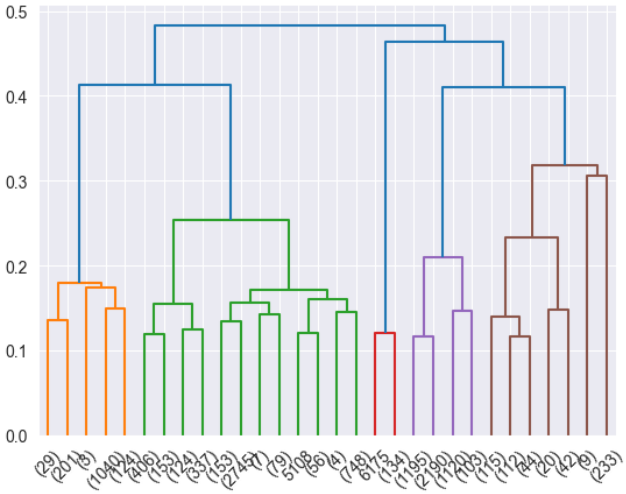


Figure 11: Hierarchical clustering with MinMaxScaler: dendogram for average, cosine, 5.

Let's visualize for this hierarchical clustering the number of killed people in each cluster, and the distribution of the feature `incident_characteristics1`: Figure 12.

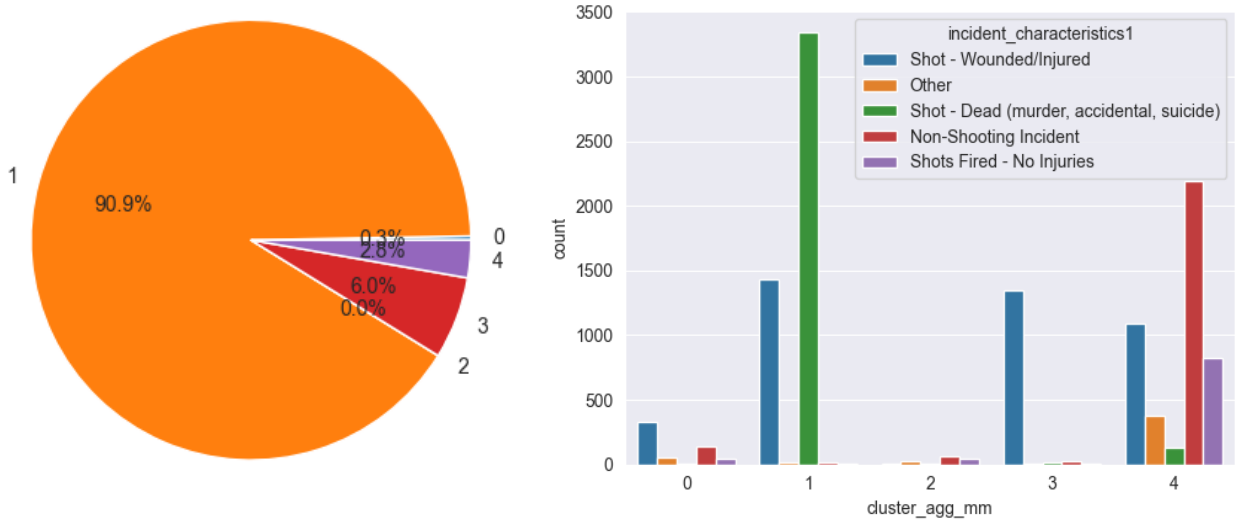
3.4 K-Means++

Using the library `pyclustering`, we finally tried the K-Means approach with the K-Means++ optimization. We used the data scaled with MinMaxScaler, and we set the number of clusters as the optimal number found previously, i.e., $k = 4$. The K-Means++ initialization seems to improve a lot the silhouette score of the clusters, with a score of 0.64. On the other hand, the best score found with random initialization was 0.36. This proves the goodness of this initialization technique. In Figure (13) we can see the 4 clusters plotted in the 2-PCA and 3-PCA plots. We can notice that the clusters look quite cohesive.

3.5 Conclusions

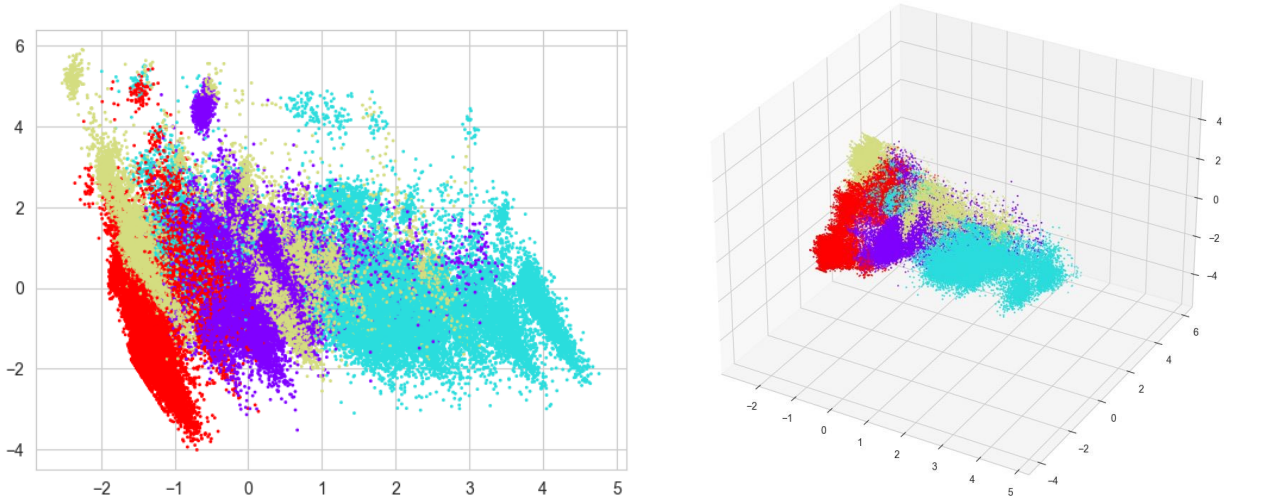
We tested several clustering algorithms and for all of them we obtained better results with the MinMaxScaler. The chosen hierarchical model: **average** linkage, **cosine** metric and **5** clusters, sepa-

rated pretty well the incidents of type 'Shot - Dead' and 'Non-Shooting Incident', but the 'Shot - Wounded/Injured' incidents were not separated (Figure 12). On the other hand, the chosen DBSCAN model: `eps` = 0.25 and `min_samples` = 10, produced 4 clusters, but in practice all the elements were concentrated in a single cluster, leading to insignificant results (Figure 8). The K-Means algorithm with 4 clusters, instead, produced a good separation of the incidents with different severity (Figure 6), that's why we conclude that it is the best choice.



(a) (b)

Figure 12: Hierarchical clustering with MinMaxScaler: percentage of killed in each cluster (12a) and distribution of `incident_characteristics1` (12b).



(a) (b)

Figure 13: K-Means++ with MinMaxScaler: clusters on the 2-PCA (13a) and 3-PCA (13b) plots.

4 Predictive Analysis

In this task we want to predict whether in a given incident there is at least a person killed or not. In this regard, we have introduced the `isKilled` feature to predict. We used the features defined in the Data Preparation phase, and some of the original features like: `n_participants`, `n_participants_teen`, `n_participants_child`, `n_females`, `n_injured`, `n_arrested`, and `n_unharmed`. However, some of the features defined in Section 2 derived directly from `n_killed`, so we did not use them. In particular, `severity` and `killed_by_district` are out of the games.
The current dataset, consisting of more than $160,000$ examples, is unbalanced in the labels: in nearly 78% of the incidents there were no killed people. We divided the dataset in training and test set, with a proportion of $80 - 20$. For models where a robust validation on the hyperparameters was needed, we used a 3-fold cross-validation approach.

4.1 Using All Available Features

The first model that we tried for the experimentation is a **logistic regression** approach. Initially, we intended to use this model as a baseline, but the results were surprising: without adjusting any hyperparameter, the model achieved an accuracy of 97% on the test set. This means that the task is nearly linearly separable.
By checking the weights of the logistic regression model, at the end of the fitting phase, we noticed behavior that we expected: some features had much higher weights and by combining them you could indeed trivially infer the `isKilled` label.
In particular, we saw that the order of magnitude of the weights was between $O(1)$ and $O(10)$ for the features `n_participants`, `n_injured`, `n_unharmed`, and `n_arrested`, while it was between $O(10^{-1})$ and $O(10^{-3})$ for almost all the others. Moreover, the weight of `n_participants` was positive, while the weights of the other three features were negative. This means that the leading relationship is:

$$N_{participants} - \big(N_{injured} + N_{unharmed} + N_{arrested}\big). \tag{1}$$

We also trained a **decision tree** on the same data, which led to similar results as shown in Table 4.

Model	Accuracy	Precision (True-False)	Recall (True-False)	F1-Score (weigthed avg)
Logistic Regr.	0.97	0.94 - 0.99	0.98 - 0.98	0.98
Decision Tree	0.98	0.96 - 0.99	0.96 - 0.98	0.98

Table 4: Model evaluation for Logistic Regression and Decision Tree.

Therefore, we proceed with the classification task using different models and dropping a subset of these attributes at each round. Doing so we can check which of them are more significant, and also detect new relevant features. We also considered that, from the point of view of analysis and knowledge discovery, it makes sense to assume that if we do not know whether there were killed or not, we do not have the other information about the status of the participants in the event either.

4.2 Discarding the Injured, Unharmed and Arrested

We tried to perform the prediction task without three of the features in Equation (1). We chose to keep `n_participants` because we thought that it was pretty reasonable to have this information. We tested several classifiers:

- **Logistic regression** showed a decline in the accuracy and a huge drop in recall and F1-score, especially on positive example (the minority class). In fact, it almost always predicts False.

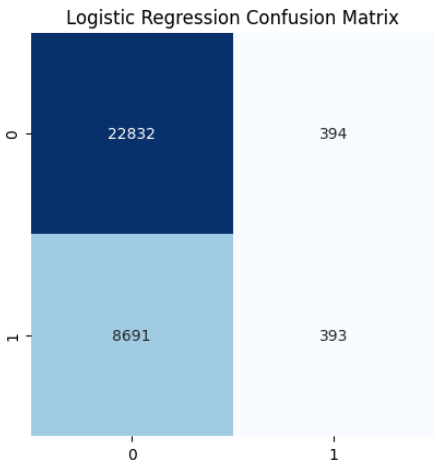
Model	Parameters
Logistic Reg.	`penalty` = L2
Decision Tree	`criterion` = `gini`, `max_depth` = 16, `min_sample_split` = 2
KNN	`n_neighbors` = 5, `metric` = `euclidean`
Random Forest	`max_depth` = 32, `min_samples_split` = 2, `n_estimators` = 30

Table 5: Hyperparameters of the models in Section 4.2.

- **Decision trees** did a little better in the average F1-score. The grid search showed that the most complex model tested was the best: `max_depth` = 16 and `min_sample_split`=2. [1] However, trying any more complex model led to overfitting on the training set, and equal or worse results on the test set.
- **KNN** also performed poorly on the positive instances. The best number of neighbors found was $K = 5$, which showed higher accuracy than decision tree but lower than logistic regression.
- The same pattern occurred with **random forest** classifiers. In this case, the best parameters found were: `criterion` = `gini`, `max_depth` = 32, `min_samples_split` = 2, and `n_estimators` = 30. Increasing the complexity of the model by using a higher number of estimators, or deeper trees, only led to overfitting.

Table 5 shows a summary of the optimal parameters for the tested models. In Table 6 we can see that all models performed poorly on all metrics. It is also significant to notice that all the accuracy scores are worse than the baseline **majority-class classifier** that always outputs `False`, and that has an accuracy score equal to the percentage of negative examples, i.e., 72%.

The only method comparable to the majority-class classifier is the logistic regression, and the value of the recall shows its abysmal true positive rate (0.04), meaning that logistic regression almost always outputs `False` too. This can also be seen immediately from the confusion matrix alongside.
In Figure 14 we can also visualize the ROC curves of the models, and notice that they are not much better than random classifiers. The poor results of all the models tried lead us to believe that the provided features are not sufficient to find significant patterns for classification.
Therefore, we decided to settle on a middle ground in the next Section: we gave the model access to the features `n_injured` and `n_participants`, leaving the features `n_unharmed` and `n_arrested` out. This setting may be less realistic than the other two, but hopefully it allows us to capture some patterns and gain some insights into the problem.



4.3 Discarding the Unharmed and Arrested

To perform the prediction having the features `n_injured` and `n_participants` we selected the following models:

- **Logistic regression** model;

[1] The `min_sample_split` parameter governs how many samples a node must have in order to be split.

Model	Accuracy	Precision (True-False)	Recall (True-False)	F1-Score (weighted avg)
Logistic Reg.	0.72	0.50 - 0.72	0.04 - 0.98	0.62
Decision Tree	0.66	0.38 - 0.75	0.35 - 0.78	0.65
KNN	0.70	0.44 - 0.75	0.28 - 0.86	0.67
Random Forest	0.70	0.44 - 0.76	0.30 - 0.85	0.68

Table 6: Performances of the models trained discarding `n_injured`, `n_unharmed`, and `n_arrested`.

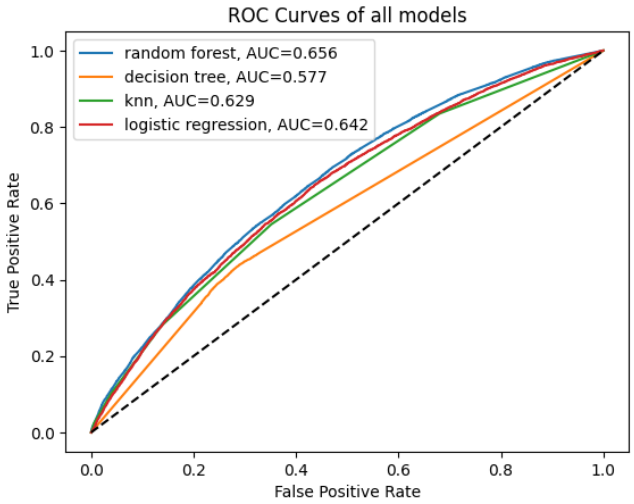


Figure 14: ROC curves of the models of Section 4.2, trained on dataset without `n_injured`, `n_unharmed`, and `n_arrested`.

- **Decision tree**, whose optimal parameters were decided with a grid search;
- **Random forest ensemble**, whose parameters were decided with a grid search;
- **Feedforward Neural Network**, whose hyperparameters were decided with a grid search.

The *boosting* approach **AdaBoost** was tested too, but it led to very poor results, probably due to the high imbalance in the dataset.
In Table 8 we reported the performances of the best grid search model on the test set. The Random Forest was the best model, followed by the Neural Network. Random forest can achieve an accuracy of 80%: an improvement of about 8% over the majority-class baseline. Adding `n_injured` to the features, indeed, made the prediction at least feasible. This can already be seen as a result in the direction of explaining the decisions of the models. Further explanations on the "black box" models are given in Section 5.

Model	Parameters
Logistic Reg.	`penalty` =l2
Decision Tree	`max_depth`=16, `min_sample_split`=16.
Random Forest	`max_depth`=16, `min_sample_split`=2, `n_estimators=50`.
Neural Network	`optimizer`=Adam, `weight_decay`=10^{-4}, `learning_rate` = 10^{-3}, `activation` =ReLU, architecture: two hidden layers of 32 units.

Table 7: Hyperparameters of the models in Section 4.3.

Model	Accuracy	Precision (True-False)	Recall (True-False)	F1-Score (macro-weighted)
Logistic Reg.	0.73	0.55 - 0.75	0.22 - 0.93	0.57 - 0.69
Decision Tree	0.78	0.61 - 0.83	0.53 - 0.87	0.71 - 0.77
Random Forest	0.80	0.68 - 0.83	0.51 - 0.91	0.72 - 0.79
Neural Network	0.79	0.65 - 0.82	0.50 - 0.90	0.71 - 0.78

Table 8: Performance of the models trained discarding `n_unharmed` and `n_arrested`.

The major issue of all the models is that they tend to classify the patterns as negative more often than necessary. In fact, the recall on the positive examples never exceeded 0.6 for any of the models. This can be due to the unbalanced data.
We then tried several techniques for balancing the dataset, but all of them led to poorer results in classification. We provide, as an example, the results for a decision tree trained with random undersampling and oversampling, and with the Synthetic Minority Over-sampling TEchnique (SMOTE) (see Table 9). All the models, indeed, had less false negatives, but way more false positives, leading to a worse accuracy than the majority-class classifier.

Balancing Technique	Accuracy	Precision (True-False)	Recall (True-False)
Random undersampling	0.63	0.94 - 0.43	0.53 - 0.91
Random oversampling	0.74	0.88 - 0.53	0.74 - 0.74
SMOTE	0.68	0.91 - 0.46	0.62 - 0.84

Table 9: Performances of decision trees on balanced data.

4.4 Conclusions

Although some models in Section 4.3 have improved the classification, the results are not optimal. The main conclusion we can draw from this task is that: either the classification is almost trivial, and therefore does not lead to the discovery of any new knowledge, or it is arbitrarily difficult if some more discriminating features are not allowed.
We think that one of the main reasons why this phenomenon occurs is that some features, even if perhaps they are significant, could be more useful at a higher level: for example for an analysis of incidents in a given state.
As an example of this, we show in Figure 16 that the poverty of the state is somehow correlated to the number of deaths in the incidents (correlation of 0.61). State poverty, however, obviously becomes a less important feature when single incidents are analyzed, as done in this Section.

5 Explanation Analysis

The last task addresses the problem of explainability, i.e., the ability to interpret why and how the models produce certain predictions. Since we are dealing with tabular data, we want to know the influence of each feature on the prediction, focusing on the models seen during the Classification phase.
So far we used models for which the interpretability is not always straightforward; we focused on the prediction score, but ignored how the model reasons to compute the prediction, effectively treating the algorithms as a black box. To address this issue, we initially used post-hoc methods, like LIME and SHAP, on the already trained models, to extract information for both global and local interpretability.

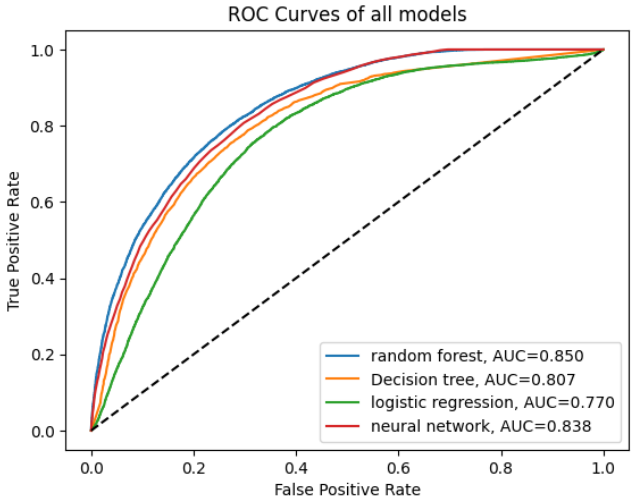


Figure 15: ROC curves for the model of Section 4.3, trained with the feature `n_injured` available.

Next, we trained an interpretable model by design: EBM, looking for more interesting insights on feature importance and on the prediction process.
For this analysis we used the same dataset as in Section 4.3: we kept `n_injured` and discarded `n_unharmed` and `n_arrested`.

5.1 Decision Tree

We consider the decision tree that gave the best results in the Classification task, that had parameters: `criterion` = `gini`, `max_depth` = 16, and `min_samples_split` = 16.

5.1.1 LIME

LIME (Local Interpretable Model-agnostic Explanations) is an explainer that can be applied to any pre-trained machine learning model to provide a measure of significance among the features of a given instance. Its real strenght lies in the ability to mark the most relevant (with respect to the prediction) portions of an image for visual data, but it works well on tabular datasets too. Moreover, this technique is model agnostic, meaning that it doesn't depend on the type of classifier used for the prediction.
Let's now discuss an example of explanation plot for a specific instance. In Figure 17 we can see on the right the first 10 most significant feature that influenced the prediction with their value, while on the left we have the assigned weights for each of them. These weights are applied to the prediction probabilities; the blue ones on the left side add to the probability of classifying the incident as False (0), while the orange for True (1).
We chose this instance since `n_participants` = 9 and `n_injured` = 0, so we would also heuristically expect that `isKilled` = False, which is actually the case. Indeed the left plot tells us that the predominant feature for the prediction was `n_injured`, as it boosts the prediction probability of 0 by 0.38. We might claim that the model learned to exploit these two features for a right prediction, even though they were poorly correlated in the correlation matrix.

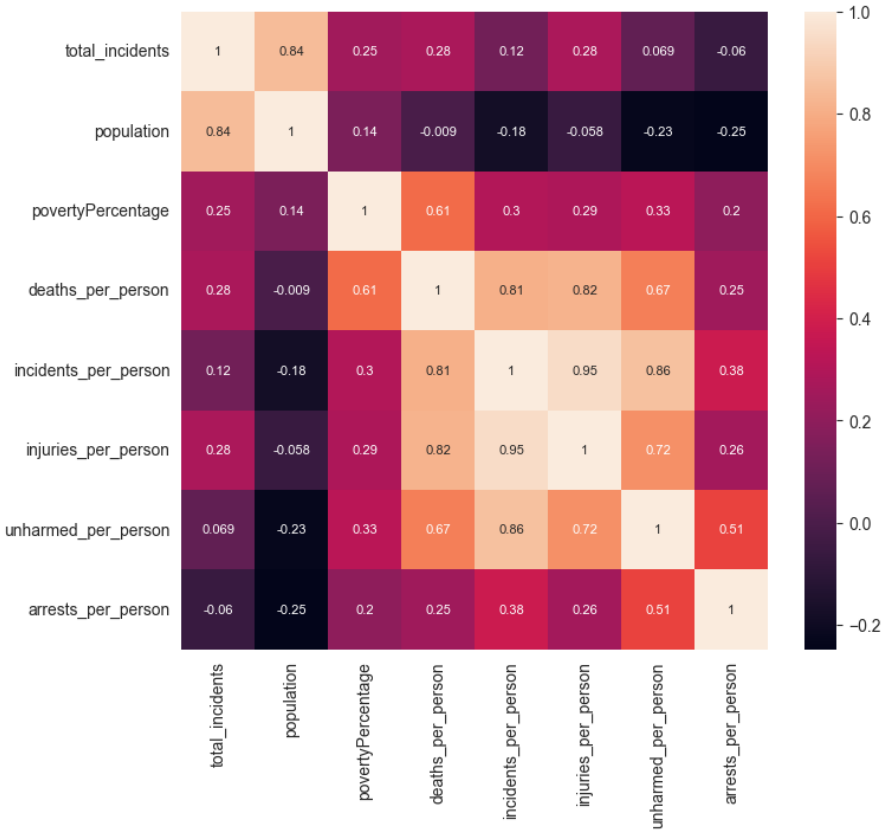


Figure 16: Correlation of some features of the aggregated dataset at the state level: the poverty of each state is positively correlated with the number of deaths per person in firearm-related incidents (i.e., we divided the number of killed in each state by the population of the state). This is an example of information that cannot be found analyzing the single incidents.

5.1.2 SHAP

SHAP (SHapley Additive exPlanations) is an approach based on game theory: Shapley values are used to assign a reward to the most important features, based on predictions of the test set. They are calculated for each feature of each instance, for both classes 0 and 1. Through these values, we can locally explain how the features interact on a single instance to infer the prediction, based on the initial probability of the label to be either True or False.
We can use the SHAP values for a global explanation: the summary plot in Figure 18a shows that the most relevant features for the prediction were `n_injured` and `n_participants`. This could explain why discarding `n_injured` brought to bad performances in the Classification task. We can also say that the importance of `n_participants` is coherent with the fact that the more participants there are, the more likely it is that someone gets killed.
Looking at the beeswarm plot in Figure 18b, we notice that a lower `n_injured` value pushes the prediction probability towards True, which we could expect since the incidents are mostly of 1 or 2 participants. Surprisingly, a higher presence of males and the victory of Democrats in the corresponding congressional district are associated with an increased probability of the incident to have a death. But, it's important to keep in mind that this behavior is not predominant, as the "swarm" for these features gathers in the center, thus being insignificant for most of the instances.

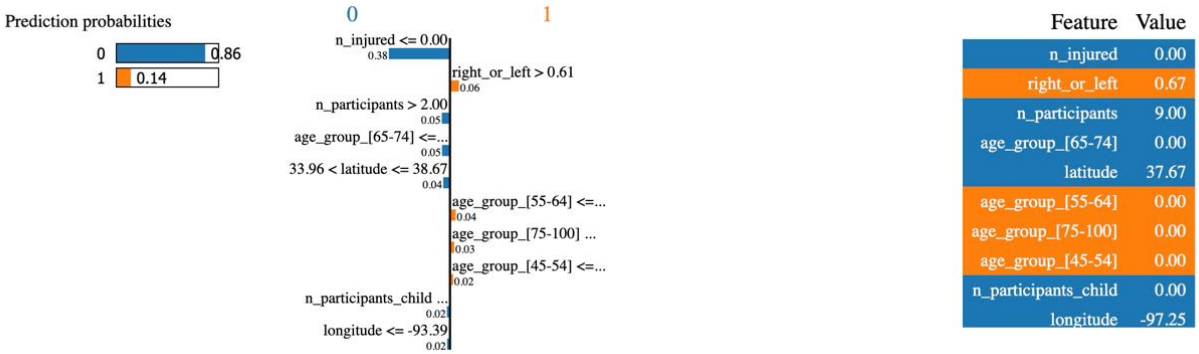
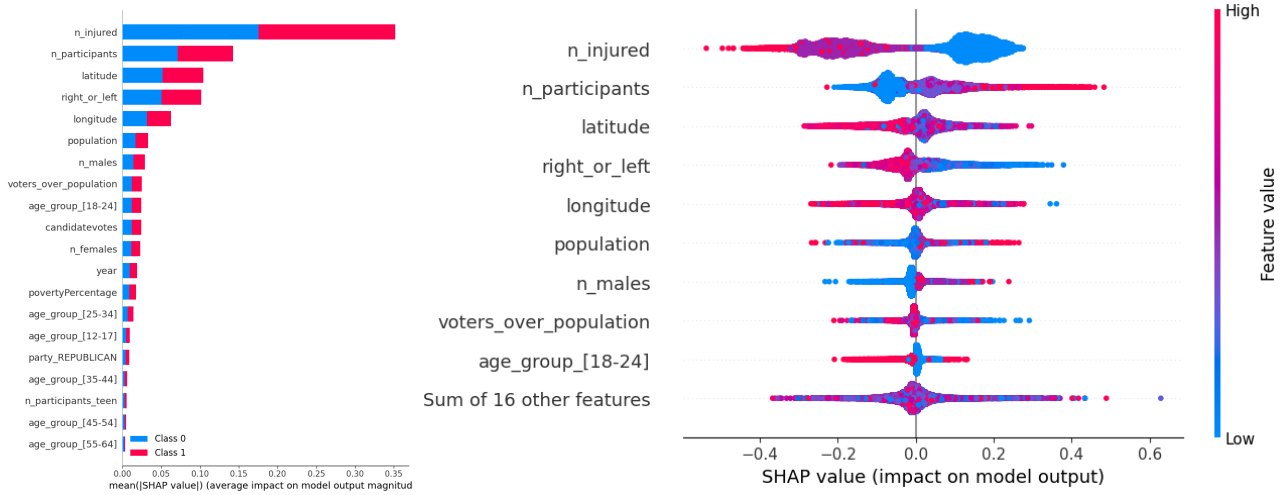


Figure 17: Decision Tree: Local explanation for an instance of the test set correctly classified as False.



(a) (b)

Figure 18: Decision Tree: SHAP values for the most relevant features (18a) and beeswarm plot (18b).

Another interesting visualization is the force plot shown in Figure 19, which contains the waterfall plots of the first 100 instances of the test set. We can see, in Figure 20, the waterfall plot of the first instance of the test set showing the contribution of each feature, similarly to what LIME did.

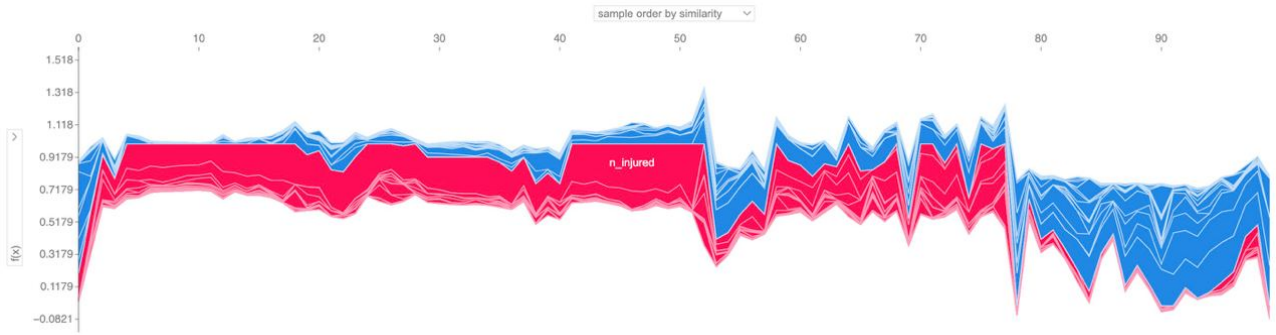


Figure 19: Decision Tree: Force plot for the first 100 instances of the test set.

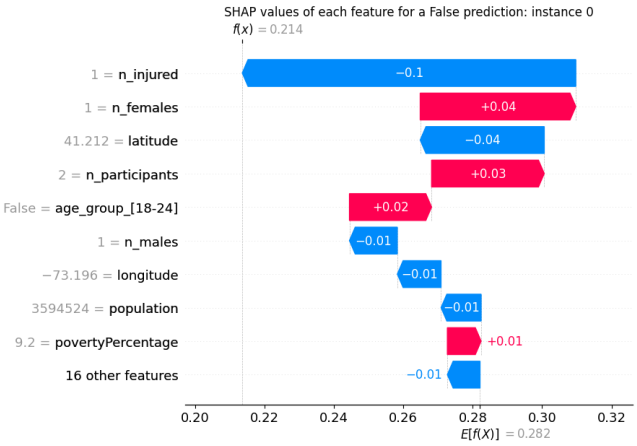


Figure 20: Decision Tree: Local explanation for the first instance of the test set.

5.2 Random Forest

Moving on to random forests, the parameters we refer to are: `criterion` = `entropy`, `max_depth` = 16, `min_samples_split` = 2, and `n_estimators` = 50.

5.2.1 LIME

In Figure 21 we can see an alternative visualization for the LIME technique. We focused on the same instance considered for the decision tree, which is again correctly classified as False. With random forest, the most significant features were `n_injured`, `n_participants`, and `n_males`; the latter was not as relevant in the case of decision trees, where instead it was `right_or_left` that was in the top 3. Also note that in this plot the orange features push towards the False class, and the blue ones towards the True class, so the colors are reversed compared to the other visualization.

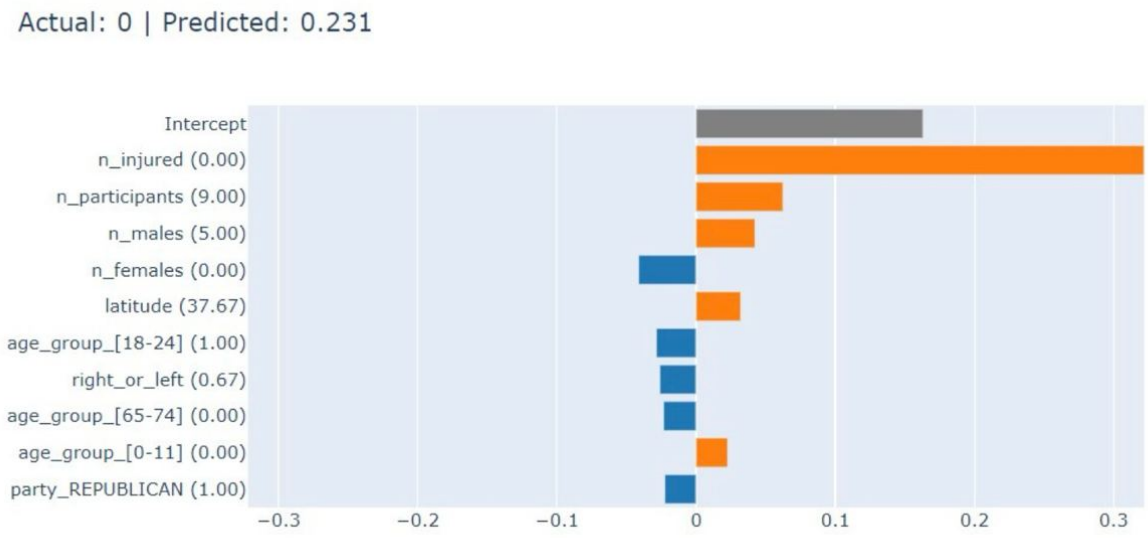
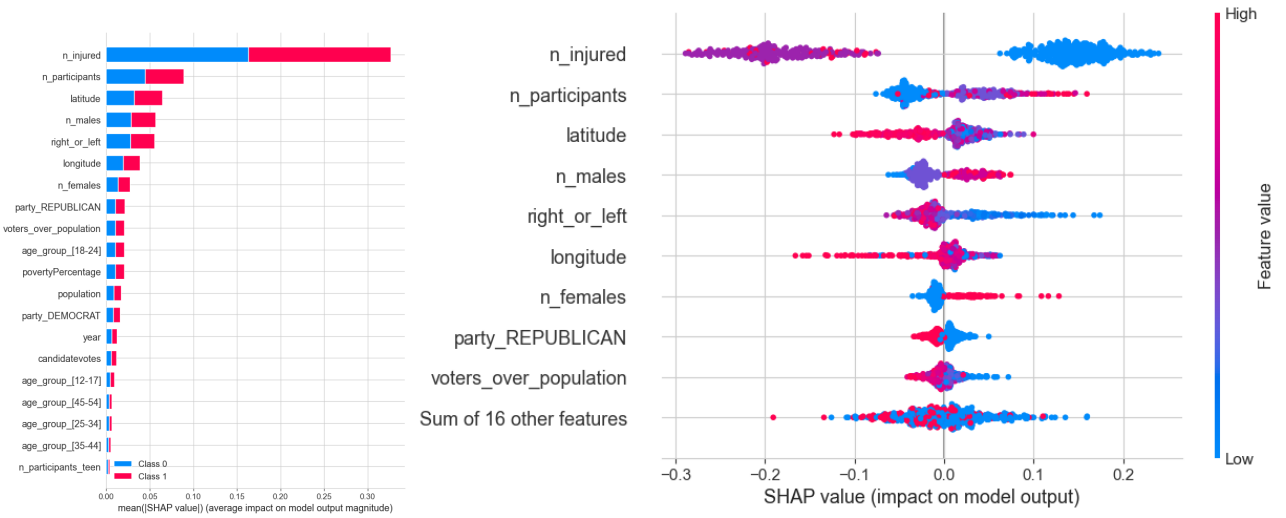


Figure 21: Random Forest: Local explanation for the same instance of Figure 17.

5.2.2 SHAP

We can see in Figure 22a that, except a few changes, the top features are still `n_injured` and `n_participants`. Since we used a subsample of 500 incidents, this time we can clearly see in the beeswarm plot how well separated are the violet and blue swarms for `n_injured`. This is easily explained by the fact that for most of the incidents the number of participants is 0 or 1, so this small sample probably captured mostly these values, thus making `n_injured` more prominent for the prediction.



(a) (b)

Figure 22: Random Forest: SHAP values for the most relevant features (22a) and beeswarm plot (22b).

5.3 Neural Network

We finally considered a neural network with 3 layers: on the first 2 layers we put `activation` = `relu` and `kernel_regularizer` = `l2(0.0001)`, and on the last layer `activation` = `sigmoid` and `kernel_regularizer` = `l2(0.0001)`.

5.3.1 LIME

The feature analyzed for the decision tree and the random forest is again correctly classified as False, as we can see in figure 23. We can see that this time the weight of the various features is more uniform, while in the other two models `n_injured` was clearly the predominant feature, here, although it is still in first place, the others do not differ much.

5.3.2 SHAP

In the case of the Neural network (Figure 24), the number of participants seems to be less important. The model weights more the geographical coordinates and the political tendencies of the areas of the incidents. This could mean that the model captured some interesting geographical patterns, but given also the performances, we think that this is actually a sign that the Neural network gives less reliable predictions.

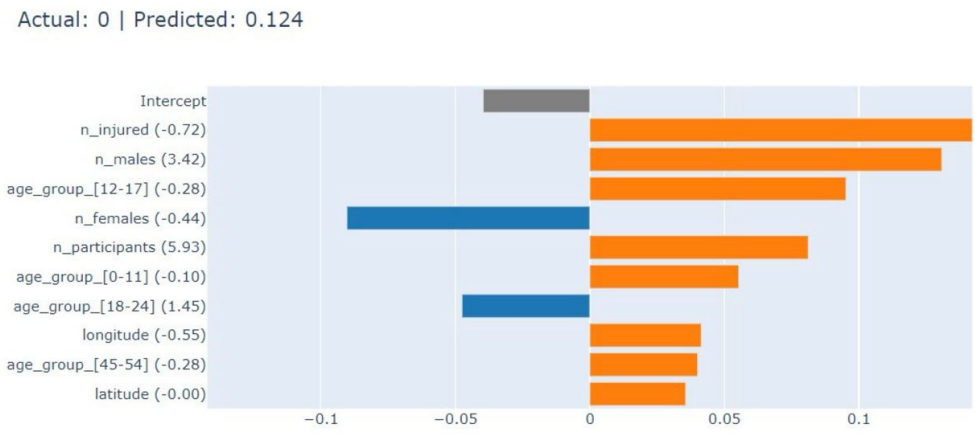


Figure 23: Neural Network: Local explanation for the same instance of Figure 17. Notice that the data are scaled.

5.4 EBM

We trained the EBM model on the same training/test split as in Section 4.3. From the Table 10 we can assert that the model performance is comparable to the previous ones.

Model	Accuracy	Precision (True-False)	Recall (True-False)	F1-Score (macro-weighted)
EBM	0.79	0.65 - 0.83	0.51 - 0.90	0.72 - 0.78

Table 10: Performance of the EBM model.

From the Figure 25, it is notable how this explainer shows the importance of jointed features. While `n_injured` is always on top, we can see that the second place is given to `n_participants & n_injured`, surpassing all the other features below by a lot. This confirms that the model fully captures the importance of the relation between the two features and the correct prediction. Again, we notice that `n_participants` alone doesn't weigh so much on the prediction, but this changes when `n_injured` is present.
As last, in Figure 26, the local explanation for the same instance of the previous models further confirms the relevance of `n_participants & n_injured` and `n_injured` alone, the two having almost opposite weights on the prediction.

5.5 Conclusions

Having explored the three explainers, we have seen that the EBM is able to catch the importance of joint features and put them in the global explanation, while the others can't directly do this. SHAP and EBM were definitely able to correlate `n_injured` and `n_participants`, which represented the two most important features for most of the models. A major drawback of the LIME method is indeed the limitation to local explanation alone (only one instance), since it's meant to be used on image data, so its contribution is not much more significant than the SHAP local analysis. Compared to EBM, the versatility of the SHAP values allows to observe more interesting insights through several plots. In conclusion, SHAP proved to be the most useful for the explanation analysis, but it suffers from the dependency of the model on which it is applied, as its model agnostic method, i.e., the Kernel Explainer, is too slow; on the other hand EBM has the advantage of being interpretable by design.

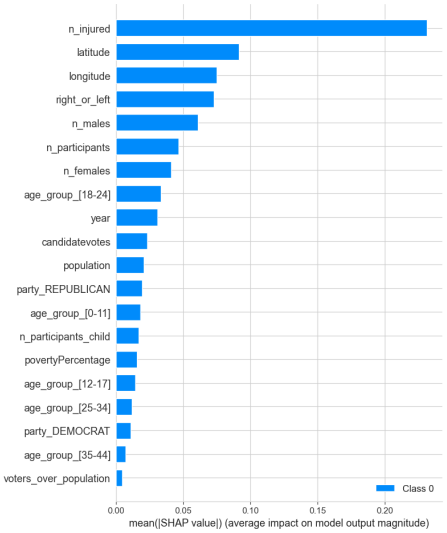


Figure 24: Neural Network: SHAP values.

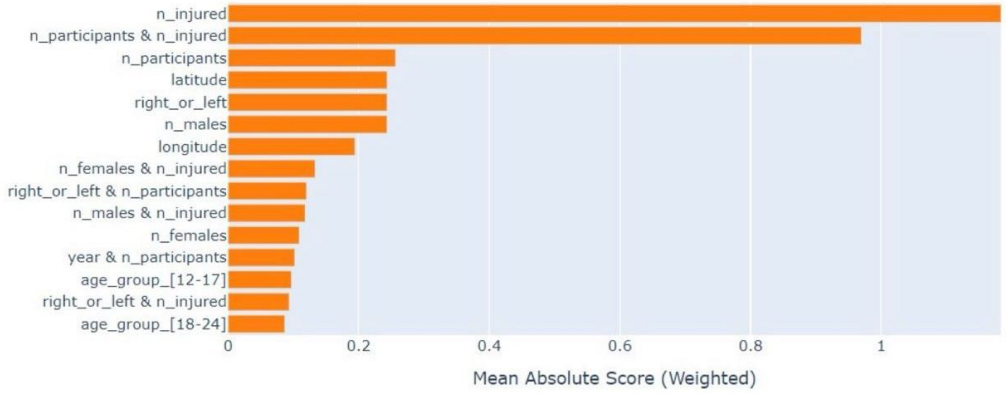


Figure 25: EBM: Global explanation.

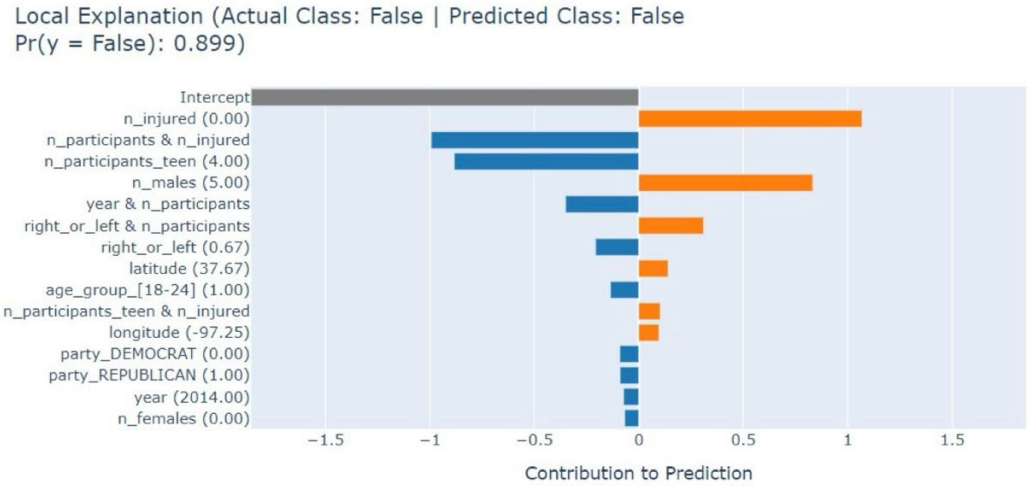


Figure 26: EBM: Local explanation for the same instance of Figure 17.