nasa-ptml

July 4, 2022

1 Introduction

Authors:

- Baptiste Bourdet
- Philippe Bernet
- Marius Dubosc
- Hugo Levy

The dataset comes from Kaggle: https://www.kaggle.com/datasets/sameepvani/nasa-nearest-earth-objects.

In this report we will try to analyze this data and compute models of both supervised and unsupervised learning to respond to a problem that was highlighted in many movies of science fiction: are any objects currently in orbit a danger to either satellites or earth.

1.1 Loading the data

First we need to load the data and analyse the different component it is made out of.

```
             id                 name  est_diameter_min  est_diameter_max  \
0       2162635  162635 (2000 SS164)          1.198271          2.679415
1       2277475    277475 (2005 WK4)          0.265800          0.594347
2       2512244   512244 (2015 YE18)          0.722030          1.614507
3       3596030          (2012 BV13)          0.096506          0.215794
4       3667127          (2014 GE35)          0.255009          0.570217
…           …                   …               …                 …
90831   3763337           (2016 VX1)          0.026580          0.059435
90832   3837603           (2019 AD3)          0.016771          0.037501
90833  54017201           (2020 JP3)          0.031956          0.071456
90834  54115824           (2021 CN5)          0.007321          0.016370
90835  54205447           (2021 TW7)          0.039862          0.089133

       relative_velocity  miss_distance orbiting_body  sentry_object  \
0           13569.249224   5.483974e+07         Earth          False
1           73588.726663   6.143813e+07         Earth          False
2          114258.692129   4.979872e+07         Earth          False
3           24764.303138   2.543497e+07         Earth          False
```

```
4           42737.733765   4.627557e+07        Earth          False
…                  …             …          …            …
90831       52078.886692   1.230039e+07        Earth          False
90832       46114.605073   5.432121e+07        Earth          False
90833        7566.807732   2.840077e+07        Earth          False
90834       69199.154484   6.869206e+07        Earth          False
90835       27024.455553   5.977213e+07        Earth          False

       absolute_magnitude  hazardous
0                   16.73      False
1                   20.00       True
2                   17.83      False
3                   22.20      False
4                   20.09       True
…                   …        …
90831               25.00      False
90832               26.00      False
90833               24.60      False
90834               27.80      False
90835               24.12      False

[90836 rows x 10 columns]
```

The data is composed of the following columns:

- id: index number
- name: name of the object
- est_dimater_min: smallest size of the object in km
- est_dimater_max: biggest size of the object in km
- relative_velocity: velocity relative to Earth in km/h
- orbiting_body: the body the object is orbiting (Earth, Sun, the Moon …)
- sentry_object: whether or not the object is tracked by the sentry system of the nasa
- absolute_magnitude: visibility index, the smaller it is, the brigther the object it, the magnitude of the sun is -27 for example
- **hazardous**: whether or not the object is considerer a potential threat by the nasa, it is this column we will want to monitor

Let's start by checking Null values

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90836 entries, 0 to 90835
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  90836 non-null  object
```

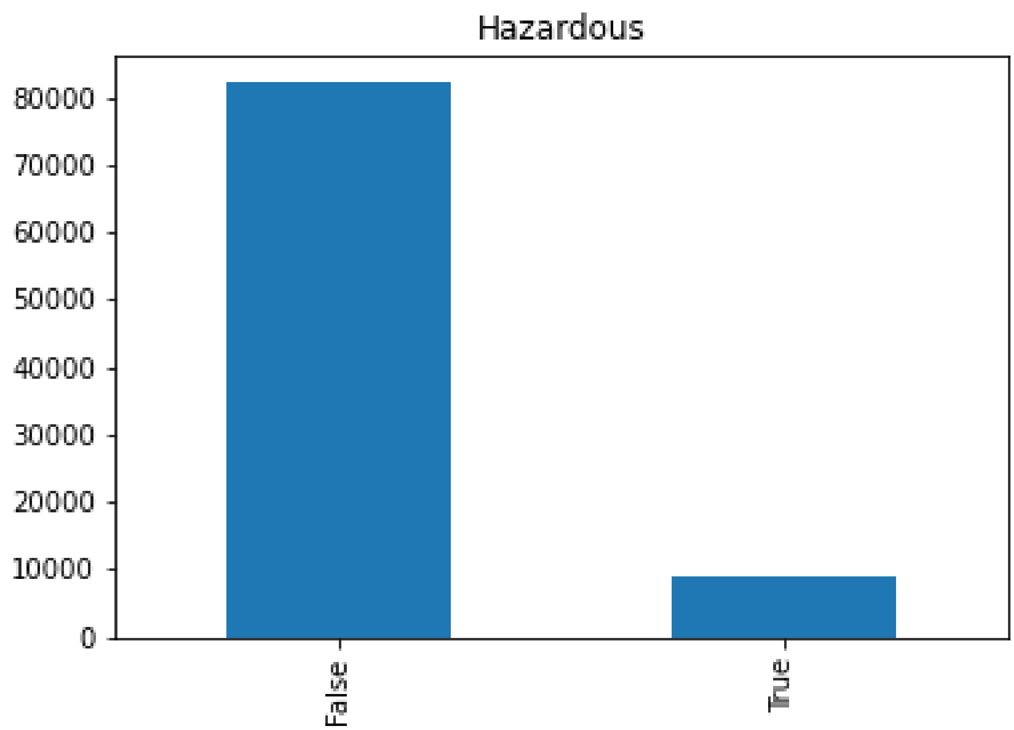
```
 1   name                 90836 non-null  object
 2   est_diameter_min     90836 non-null  float64
 3   est_diameter_max     90836 non-null  float64
 4   relative_velocity    90836 non-null  float64
 5   miss_distance        90836 non-null  float64
 6   orbiting_body        90836 non-null  category
 7   sentry_object        90836 non-null  bool
 8   absolute_magnitude   90836 non-null  float64
 9   hazardous            90836 non-null  bool
dtypes: bool(2), category(1), float64(5), object(2)
memory usage: 5.1+ MB
```

2 Analysis

2.1 Basic statistics

The start of the analysis will be purely on the statistic to gain a batter comprehension of the data.

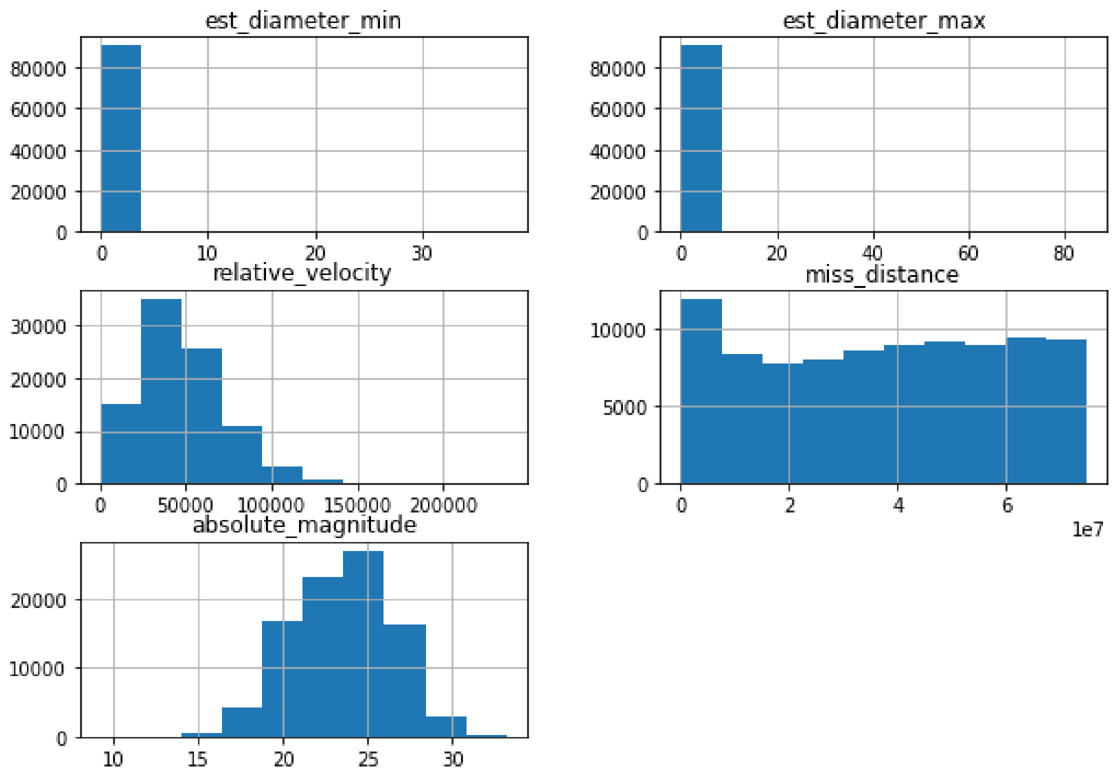
```
Length of the dataset is 90836
```




```
Summary of all numerical values :

       est_diameter_min  est_diameter_max  relative_velocity  miss_distance  \
count      90836.000000      90836.000000       90836.000000   9.083600e+04
```

mean	0.127432	0.284947	48066.918918	3.706655e+07
std	0.298511	0.667491	25293.296961	2.235204e+07
min	0.000609	0.001362	203.346433	6.745533e+03
25%	0.019256	0.043057	28619.020645	1.721082e+07
50%	0.048368	0.108153	44190.117890	3.784658e+07
75%	0.143402	0.320656	62923.604633	5.654900e+07
max	37.892650	84.730541	236990.128088	7.479865e+07

	absolute_magnitude
count	90836.000000
mean	23.527103
std	2.894086
min	9.230000
25%	21.340000
50%	23.700000
75%	25.700000
max	33.200000

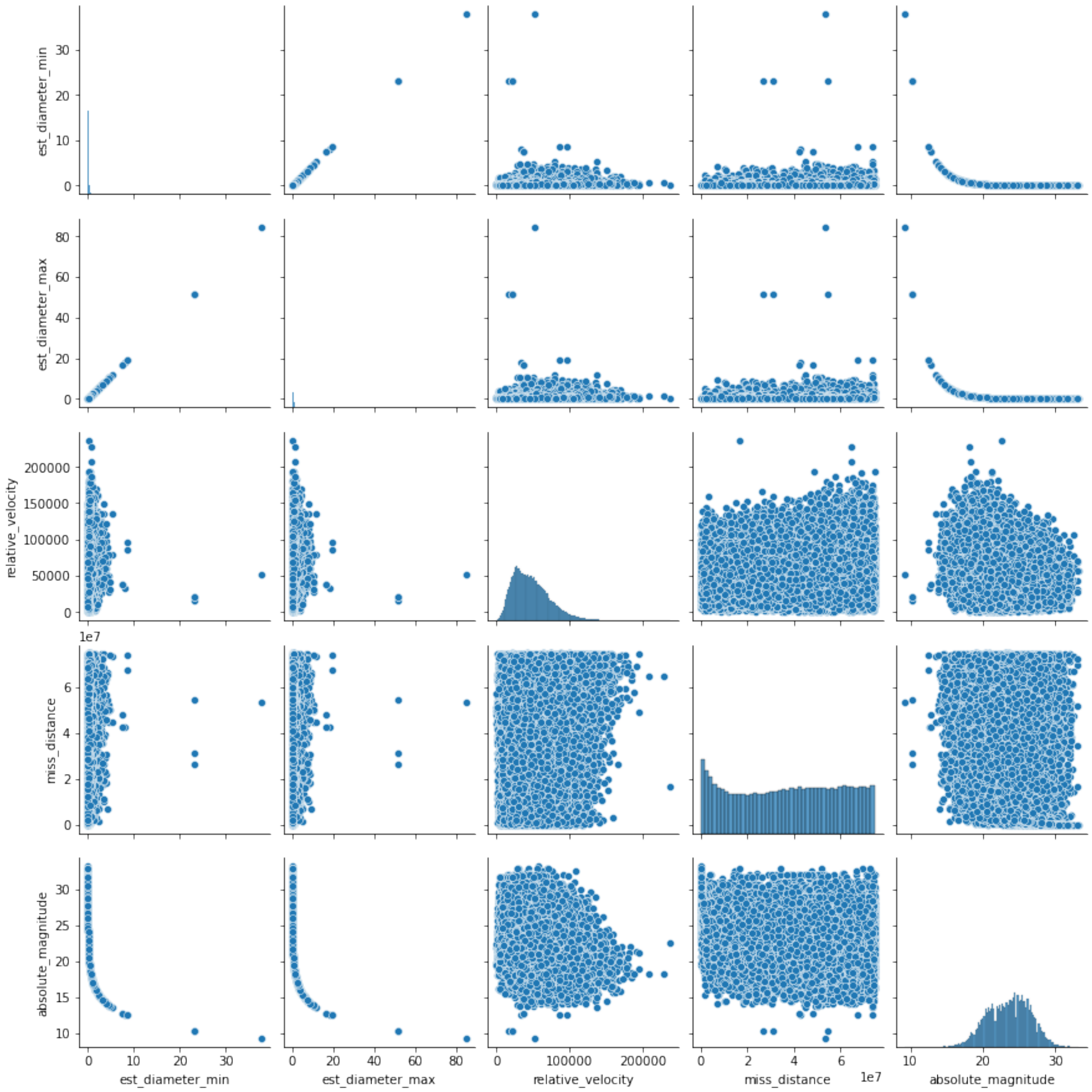


Visualization of categorical values :

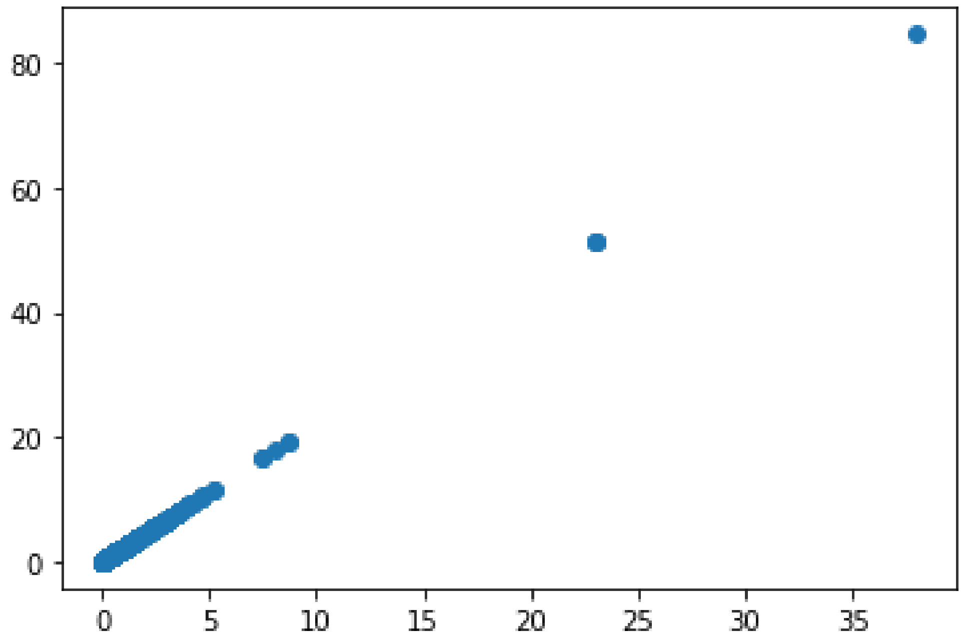
	orbiting_body	sentry_object	hazardous
count	90836	90836	90836
unique	1	1	2
top	Earth	False	False

```
freq            90836          90836      81996
```

As we can see we have only one `orbiting body` and only one value for the `sentry`. Those two columns can therefore be removed from the dataset safely. The final column, the one we want to predict with the models, seems to have 10% of dangerous objects, the outliers we will try to identify.




```
<matplotlib.collections.PathCollection at 0x7fb7fa1f91c0>
```



We see that est_diameter_min and est_diamater_max are completly linearly correlated. We can therefore remove one.

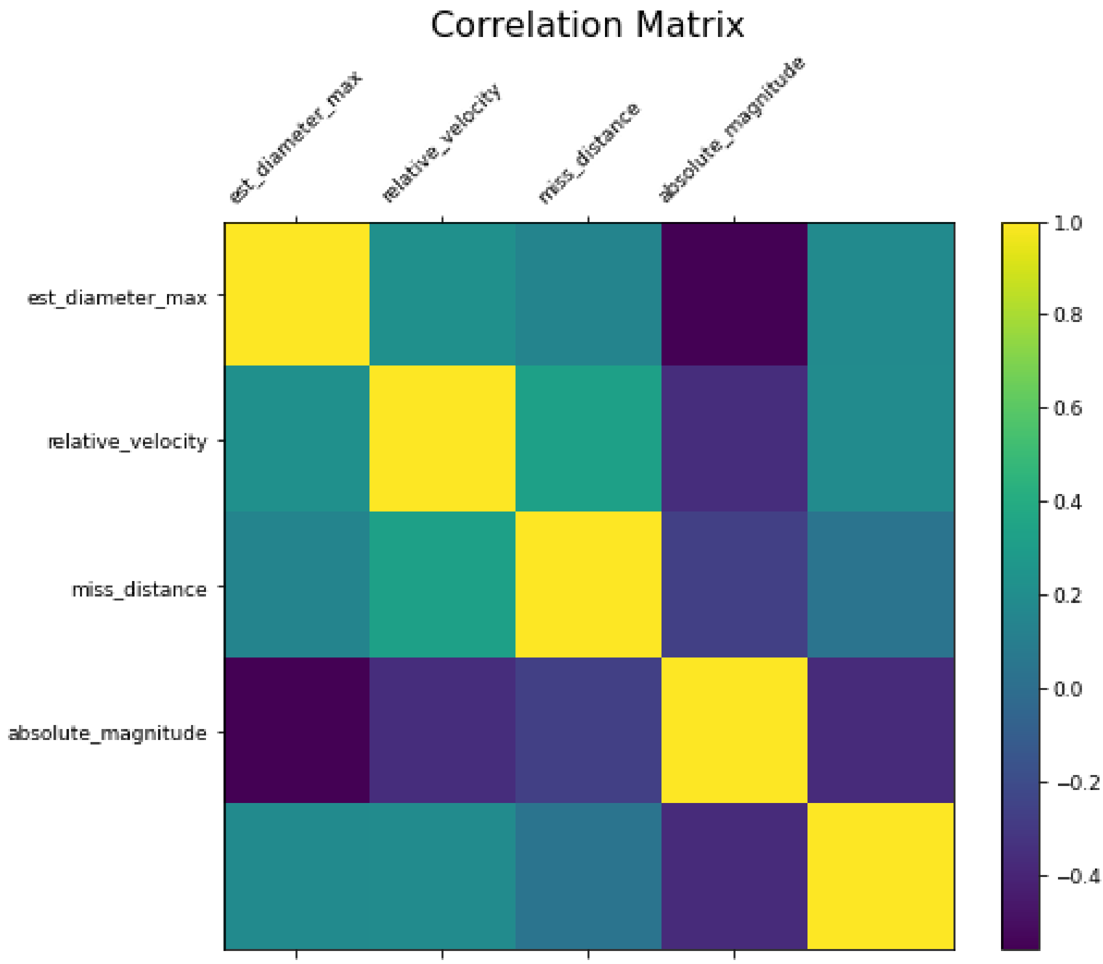
```
             id                 name  est_diameter_max  relative_velocity  \
0       2162635  162635 (2000 SS164)          2.679415       13569.249224
1       2277475    277475 (2005 WK4)          0.594347       73588.726663
2       2512244   512244 (2015 YE18)          1.614507      114258.692129
3       3596030          (2012 BV13)          0.215794       24764.303138
4       3667127          (2014 GE35)          0.570217       42737.733765
…           …                    …                 …                …
90831   3763337           (2016 VX1)          0.059435       52078.886692
90832   3837603           (2019 AD3)          0.037501       46114.605073
90833  54017201           (2020 JP3)          0.071456        7566.807732
90834  54115824           (2021 CN5)          0.016370       69199.154484
90835  54205447           (2021 TW7)          0.089133       27024.455553

       miss_distance  absolute_magnitude  hazardous
0       5.483974e+07               16.73      False
1       6.143813e+07               20.00       True
2       4.979872e+07               17.83      False
3       2.543497e+07               22.20      False
4       4.627557e+07               20.09       True
…                …                   …        …
90831   1.230039e+07               25.00      False
90832   5.432121e+07               26.00      False
90833   2.840077e+07               24.60      False
```

```
90834   6.869206e+07                   27.80      False
90835   5.977213e+07                   24.12      False

[90836 rows x 7 columns]
```

2.2 Correlation

The correlation matrix of our data is the following:

```
                    est_diameter_max  relative_velocity  miss_distance  \
est_diameter_max            1.000000           0.221553       0.142241
relative_velocity           0.221553           1.000000       0.327169
miss_distance               0.142241           0.327169       1.000000
absolute_magnitude         -0.560188          -0.353863      -0.264168
hazardous                   0.183363           0.191185       0.042302

                    absolute_magnitude  hazardous
est_diameter_max             -0.560188   0.183363
relative_velocity            -0.353863   0.191185
miss_distance                -0.264168   0.042302
absolute_magnitude            1.000000  -0.365267
hazardous                    -0.365267   1.000000
```



<Figure size 1368x1080 with 0 Axes>

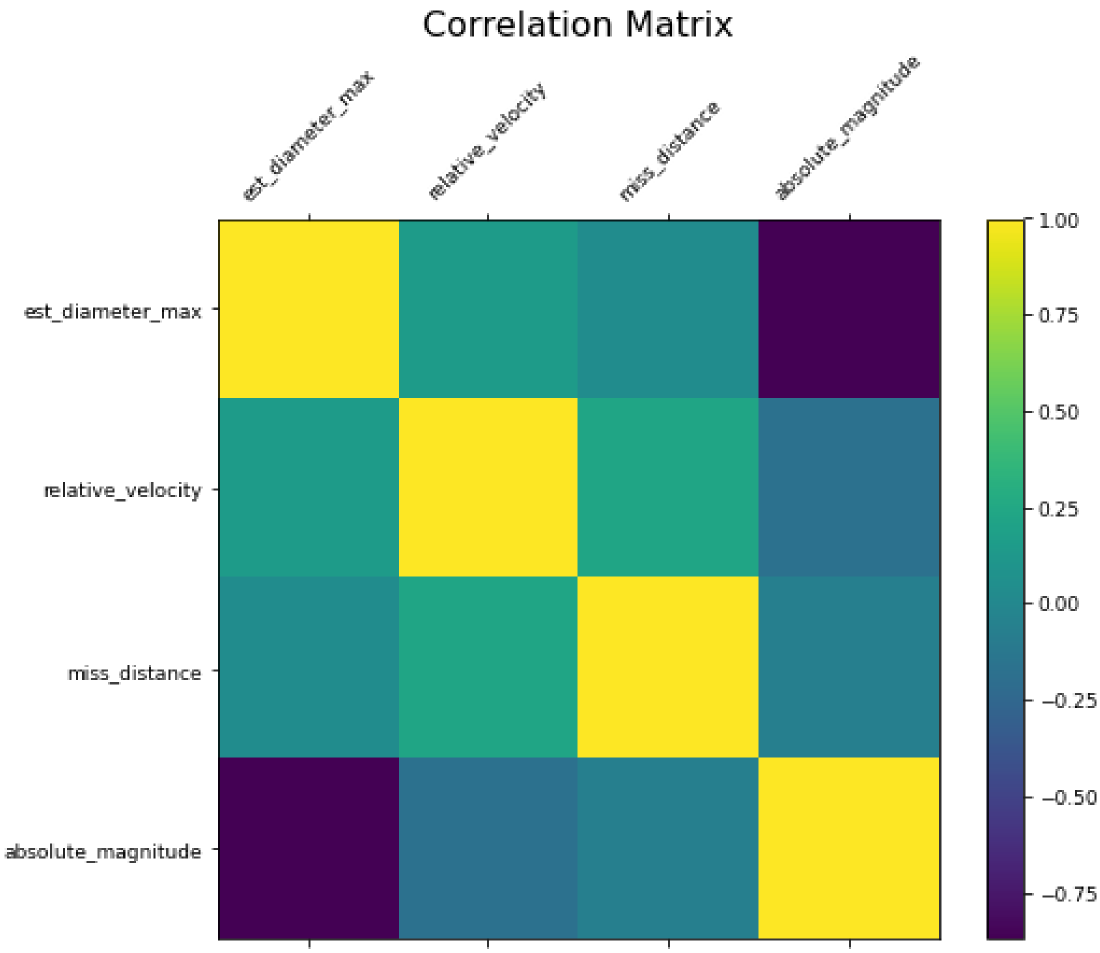
(90836, 4)

	id	name	est_diameter_max	relative_velocity	\
0	2162635	162635 (2000 SS164)	2.679415	13569.249224	
1	2277475	277475 (2005 WK4)	0.594347	73588.726663	
2	2512244	512244 (2015 YE18)	1.614507	114258.692129	
3	3596030	(2012 BV13)	0.215794	24764.303138	
4	3667127	(2014 GE35)	0.570217	42737.733765	
…	…	…	…	…	
90831	3763337	(2016 VX1)	0.059435	52078.886692	
90832	3837603	(2019 AD3)	0.037501	46114.605073	
90833	54017201	(2020 JP3)	0.071456	7566.807732	
90834	54115824	(2021 CN5)	0.016370	69199.154484	
90835	54205447	(2021 TW7)	0.089133	27024.455553	

```
       miss_distance  absolute_magnitude  hazardous
0        5.483974e+07               16.73      False
1        6.143813e+07               20.00       True
2        4.979872e+07               17.83      False
3        2.543497e+07               22.20      False
4        4.627557e+07               20.09       True
…              …                  …        …
90831    1.230039e+07               25.00      False
90832    5.432121e+07               26.00      False
90833    2.840077e+07               24.60      False
90834    6.869206e+07               27.80      False
90835    5.977213e+07               24.12      False

[90836 rows x 7 columns]
```

We can see in the matrix that the magnitude is negatively correlated with mos of the other variables. This implies that objects with a high magntiude, meaning not very visible objects, are usually smaller and slower. They also more importently do not consistute a threat seeing how the magnitude is negatively correlated with the hazardous.

When computing the correlation matrix only for the hazardous objects, we can see an even greater correlation between the magnitude and the size.



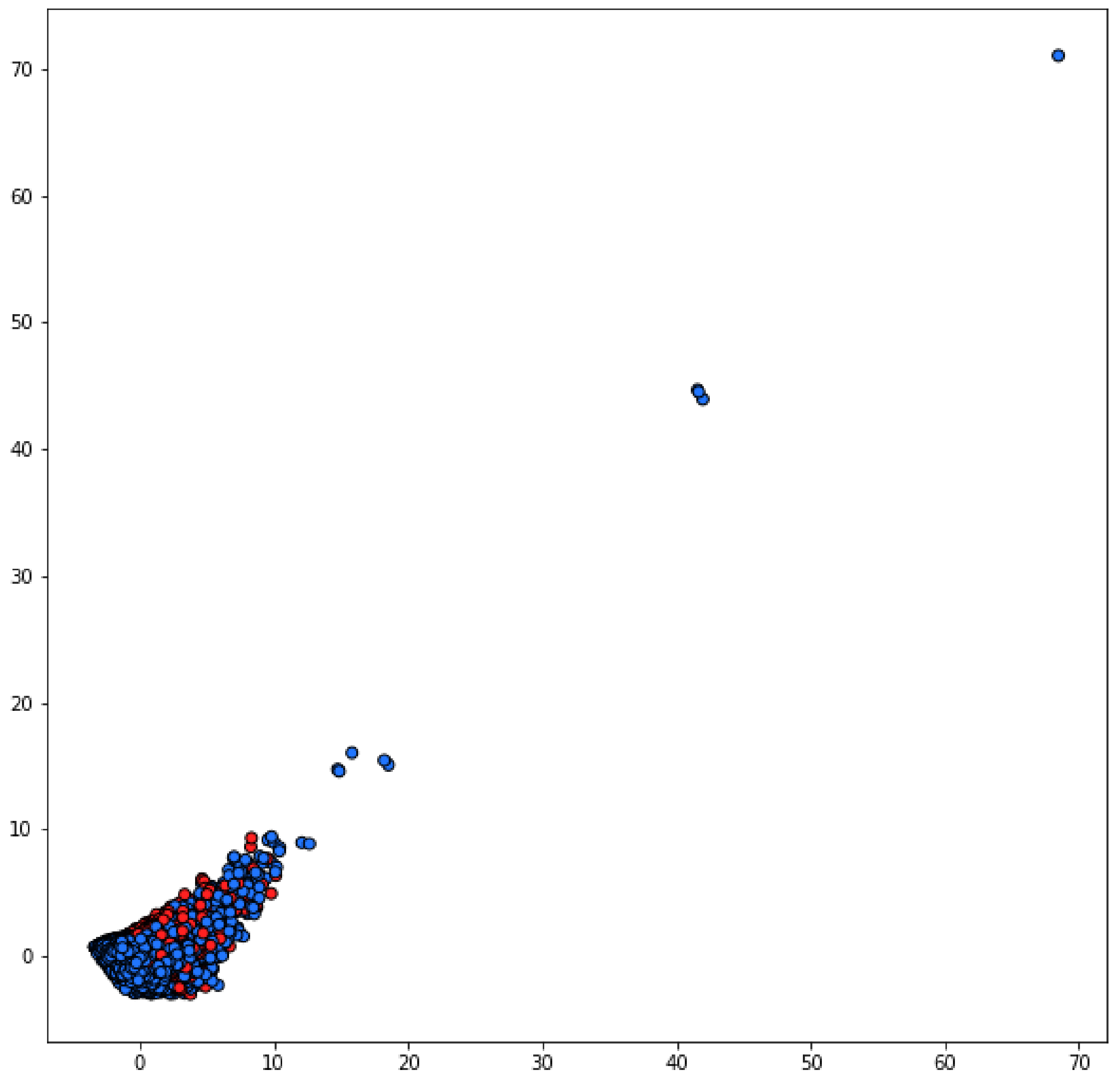

```
<Figure size 1368x1080 with 0 Axes>
```

2.2.1 Dimension Reduction

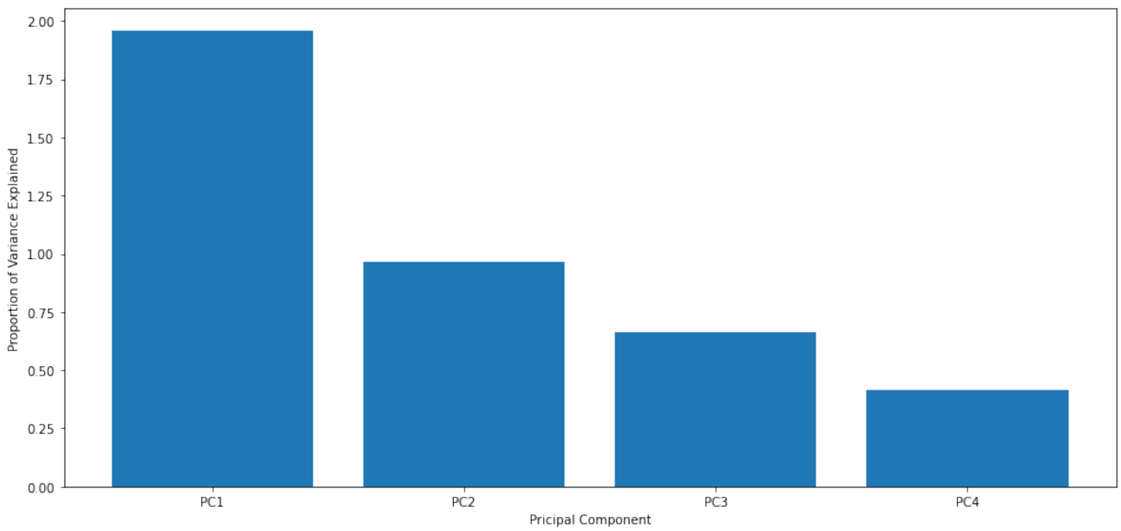
It seems some variables are more important than other. Let's see how much information we can keep while reducing the dimension.

	id	name	est_diameter_min	est_diameter_max	\
0	2162635	162635 (2000 SS164)	1.198271	2.679415	
1	2277475	277475 (2005 WK4)	0.265800	0.594347	
2	2512244	512244 (2015 YE18)	0.722030	1.614507	
3	3596030	(2012 BV13)	0.096506	0.215794	
4	3667127	(2014 GE35)	0.255009	0.570217	
…	…	…	…	…	
90831	3763337	(2016 VX1)	0.026580	0.059435	
90832	3837603	(2019 AD3)	0.016771	0.037501	
90833	54017201	(2020 JP3)	0.031956	0.071456	
90834	54115824	(2021 CN5)	0.007321	0.016370	

```
90835  54205447          (2021 TW7)         0.039862          0.089133

       relative_velocity  miss_distance orbiting_body  sentry_object  \
0           13569.249224   5.483974e+07         Earth          False
1           73588.726663   6.143813e+07         Earth          False
2          114258.692129   4.979872e+07         Earth          False
3           24764.303138   2.543497e+07         Earth          False
4           42737.733765   4.627557e+07         Earth          False
…                    …            …            …             …
90831       52078.886692   1.230039e+07         Earth          False
90832       46114.605073   5.432121e+07         Earth          False
90833        7566.807732   2.840077e+07         Earth          False
90834       69199.154484   6.869206e+07         Earth          False
90835       27024.455553   5.977213e+07         Earth          False

       absolute_magnitude
0                   16.73
1                   20.00
2                   17.83
3                   22.20
4                   20.09
…                    …
90831               25.00
90832               26.00
90833               24.60
90834               27.80
90835               24.12

[90836 rows x 9 columns]
```



70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70



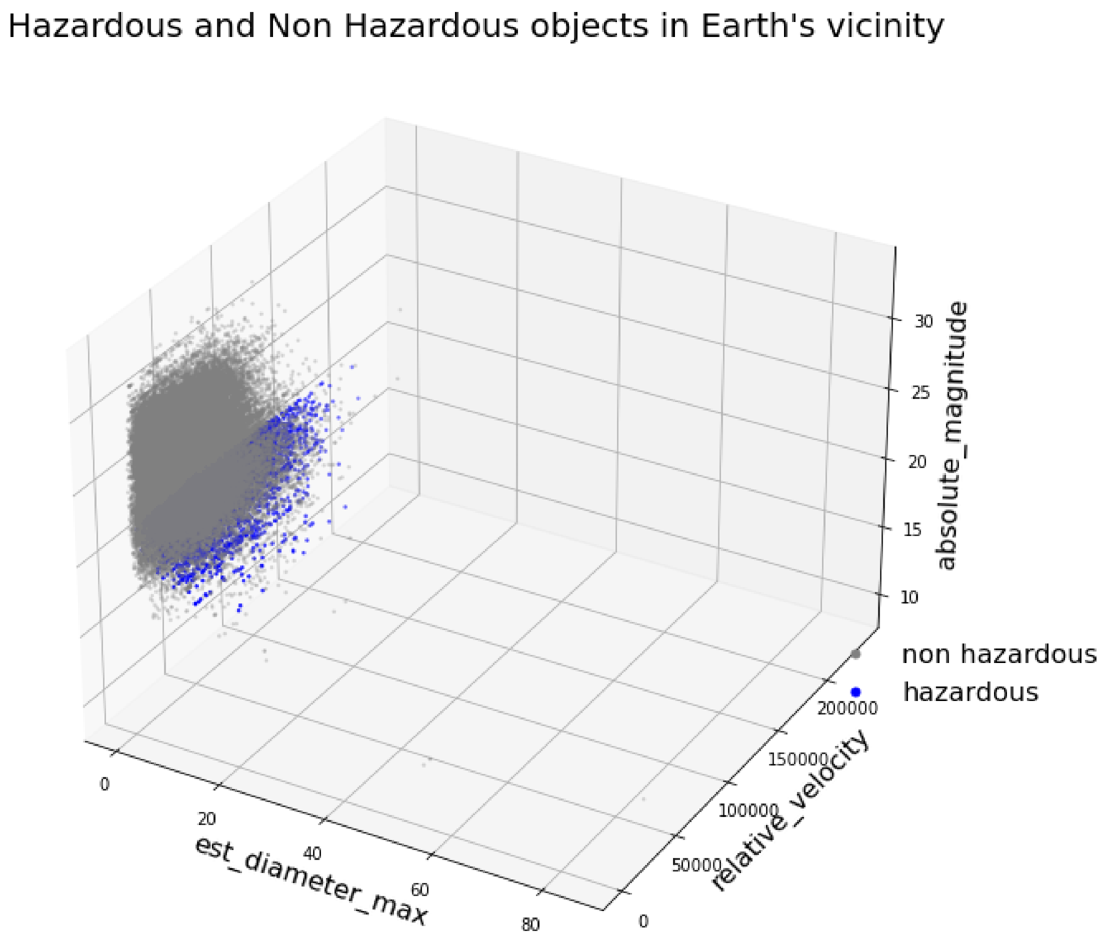

```
array([0.48894661, 0.24174391, 0.16526435, 0.10404514])
```

```
48.8946605366815% variance explained for 0 axes
73.06905105546092% variance explained for 1 axes
89.59548566091324% variance explained for 2 axes
100.0% variance explained for 3 axes
```

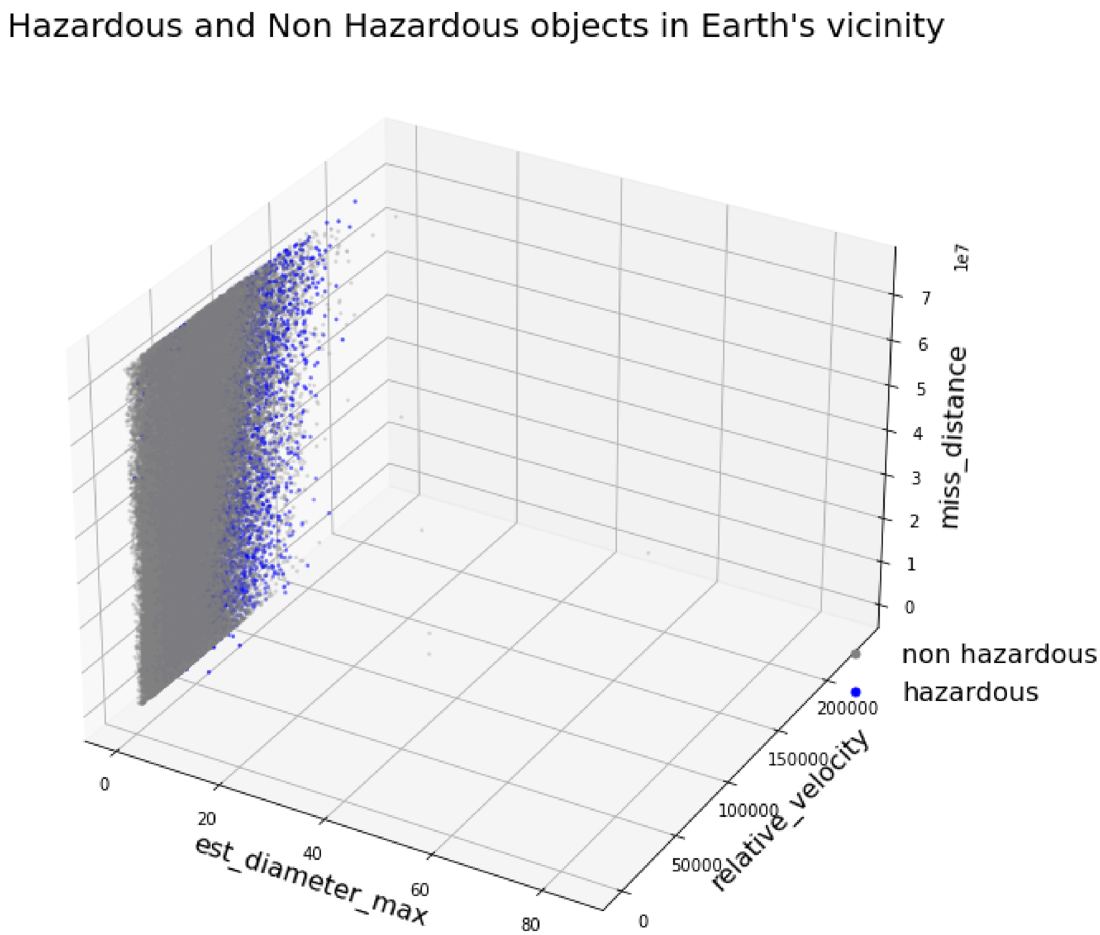
We see that two axes are enough to explain 76% of the variance. This could be a way to better represent and explain the data for the models.

2.3 3D representation

Another approach to visualization is to see how the different columns interact with each other.



Has can be seen on this first representation, the velocity seems to be slightly higher for objects considered a threat compared to the rest. The magnitude also seems to be capped at 20 for the hazardous objects, a high magnitude impliying a very dim object in the darkness of space.



On this second graph we can further see that the velocity seems to be an important factor but on the other end the miss distance is not very representative. Indeed the miss distance is not that important considering objects could be shifted our of their orbit very easily by the cosmic billiard played in the solar system by gravity. An oibject that missed earth by a lot could still be a threat.

3 Supervised Learning

Now that the statistical analysis of the data is done, the next step is to try to predict the `hazardous` values from the rest of the data. Our first approach was with supervised learning. First, we clean the dataset to get quantitative data to determine wether or not the object is considerer a potential threat by the nasa, it is this column we will want to monitor. We remove constant values (sentry_object=False and orbiting_body=Earth).

```
         est_diameter_min  est_diameter_max  relative_velocity  \
id
2162635          1.198271          2.679415       13569.249224
2277475          0.265800          0.594347       73588.726663
```

```
2512244            0.722030            1.614507      114258.692129
3596030            0.096506            0.215794       24764.303138
3667127            0.255009            0.570217       42737.733765
…                     …                  …                …
3763337            0.026580            0.059435       52078.886692
3837603            0.016771            0.037501       46114.605073
54017201           0.031956            0.071456        7566.807732
54115824           0.007321            0.016370       69199.154484
54205447           0.039862            0.089133       27024.455553

          miss_distance  absolute_magnitude  hazardous
id
2162635    5.483974e+07               16.73      False
2277475    6.143813e+07               20.00       True
2512244    4.979872e+07               17.83      False
3596030    2.543497e+07               22.20      False
3667127    4.627557e+07               20.09       True
…                 …                  …         …
3763337    1.230039e+07               25.00      False
3837603    5.432121e+07               26.00      False
54017201   2.840077e+07               24.60      False
54115824   6.869206e+07               27.80      False
54205447   5.977213e+07               24.12      False

[90836 rows x 6 columns]
```

3.1 Initialize train and test sets

To properly train our models on the data, we need to split the data in two parts, one for testing and one for training, this will allow proper scoring with data the models were not trained on.

```
Number of training data: 60860
```

```
 False    54969
 True      5891
 Name: hazardous, dtype: int64
```

```
 89.28305044661536
```

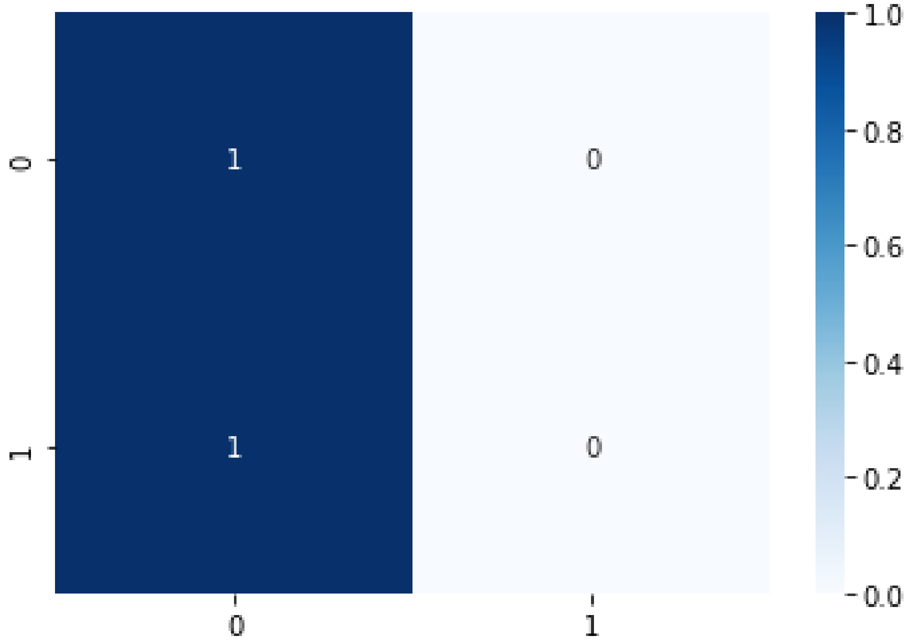
The data has been analyzed and cleaned; now, its time to build those machine learning models.

The train set contains 60860 data instances and has 5891 cases labelled as 1(hazardous), which means that if a model predicts all values as 0, then the accuracy will be 89.28%. This will be considered as baseline accuracy for the train set. Similarly, the baseline accuracy for the test set will be 89.08%. Our model should do better than these accuracies or should be robust enough to deal with the class imbalance.

3.2 Let's try multiple models

Let's start with a logistic reression.

```
=============== LogisticRegressionCV() ===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support

       False       0.90      1.00      0.95     54969
        True       0.00      0.00      0.00      5891

    accuracy                           0.90     60860
   macro avg       0.45      0.50      0.47     60860
weighted avg       0.82      0.90      0.86     60860

test :
              precision    recall  f1-score   support

       False       0.90      1.00      0.95     27027
        True       0.00      0.00      0.00      2949

    accuracy                           0.90     29976
   macro avg       0.45      0.50      0.47     29976
weighted avg       0.81      0.90      0.85     29976

------------ Accuracy Score ------------
train :
0.9032040749260598
test :
0.90162129703763

------------ Confusion matrix on test ------------
[[1. 0.]
 [1. 0.]]

False negative : 1.0
False positive : 0.0
Test accuracy : 0.90162129703763
```

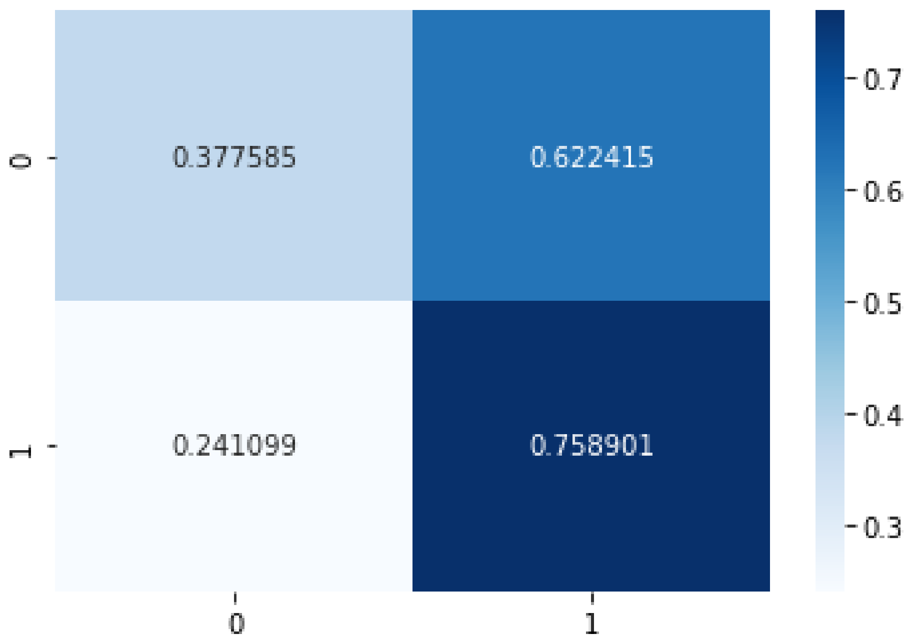


The confusion matrices clearly show that the model fails to predict even a single data instance as 1 (hazardous) and hence the model is not robust enough. We can ask the model to balance the data :

The "balanced" mode of the *class_weight* parameter uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as $n\_samples/(n\_classes * np.bincount(y))$

```
=============== LogisticRegressionCV(class_weight='balanced') ===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support

       False       0.94      0.38      0.54     54969
        True       0.12      0.76      0.20      5891

    accuracy                           0.41     60860
   macro avg       0.53      0.57      0.37     60860
weighted avg       0.86      0.41      0.50     60860

test :
              precision    recall  f1-score   support

       False       0.93      0.38      0.54     27027
        True       0.12      0.76      0.20      2949

    accuracy                           0.42     29976
```

```
   macro avg       0.53      0.57      0.37     29976
weighted avg       0.85      0.42      0.51     29976

------------ Accuracy Score ------------
train :
0.41390075583305946
test :
0.4150987456631972

------------ Confusion matrix on test ------------
[[0.37758538 0.62241462]
 [0.24109868 0.75890132]]

False negative : 0.24109867751780265
False positive : 0.6224146224146224
Test accuracy : 0.4150987456631972
```

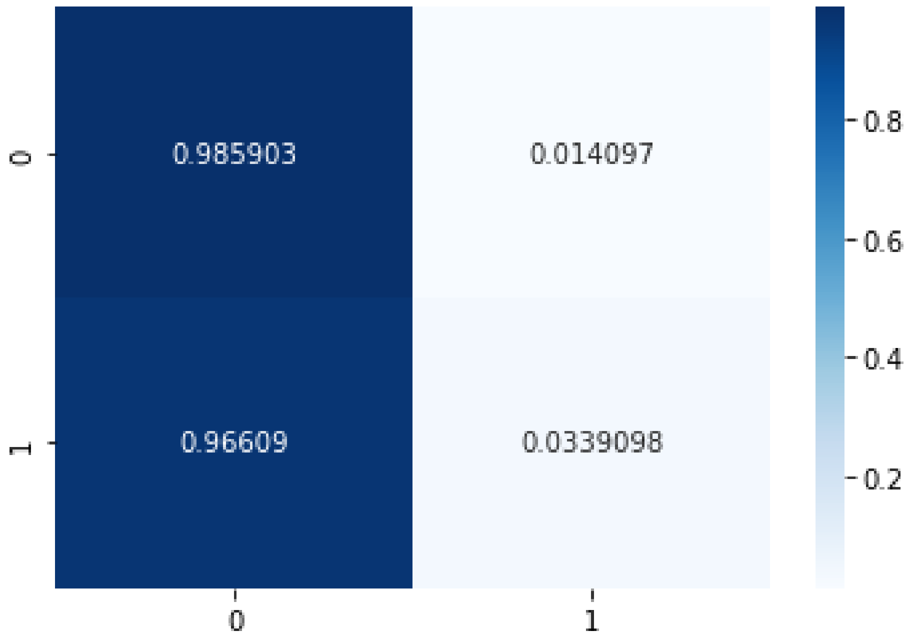


We observe really bad performances : the model achieves 41.5% accuracy, way below the 89% baseline accuracy.

Let's try another strategy : K neighbors

```
=============== KNeighborsClassifier() ===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support
```

```
       False       0.91      0.99      0.95     54969
        True       0.66      0.11      0.18      5891

    accuracy                           0.91     60860
   macro avg       0.79      0.55      0.57     60860
weighted avg       0.89      0.91      0.88     60860

test :
              precision    recall  f1-score   support

       False       0.90      0.99      0.94     27027
        True       0.21      0.03      0.06      2949

    accuracy                           0.89     29976
   macro avg       0.56      0.51      0.50     29976
weighted avg       0.83      0.89      0.86     29976

------------ Accuracy Score ------------
train :
0.9083470259612225
test :
0.8922471310381639

------------ Confusion matrix on test ------------
[[0.98590299 0.01409701]
 [0.9660902  0.0339098 ]]

False negative : 0.9660902000678196
False positive : 0.014097014097014098
Test accuracy : 0.8922471310381639
```

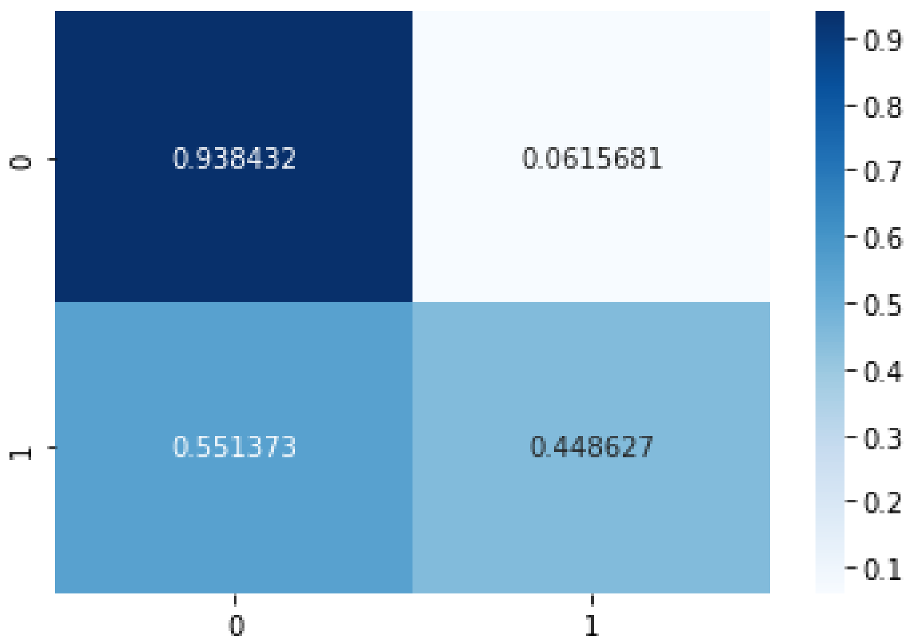


If we look at the table for Naive Bayes, then we see that the accuracies for the test set and train set are equal to baseline accuracies. Moreover, if we look at the confusion matrix, we remark that the model almost always classifies as negative, meaning the model is broken and has predicted all values as 0 (not hazardous). Such a model is of zero significance because there's no point in using a model when it can never fulfil its purpose.

The results we observe are surely caused by the fact that the dataset is un balanced. Let's use another strategy : Decision tree and ensembling.

```
=============== DecisionTreeClassifier(random_state=0) ===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support

       False       1.00      1.00      1.00     54969
        True       1.00      1.00      1.00      5891

    accuracy                           1.00     60860
   macro avg       1.00      1.00      1.00     60860
weighted avg       1.00      1.00      1.00     60860

test :
              precision    recall  f1-score   support

       False       0.94      0.94      0.94     27027
        True       0.44      0.45      0.45      2949
```

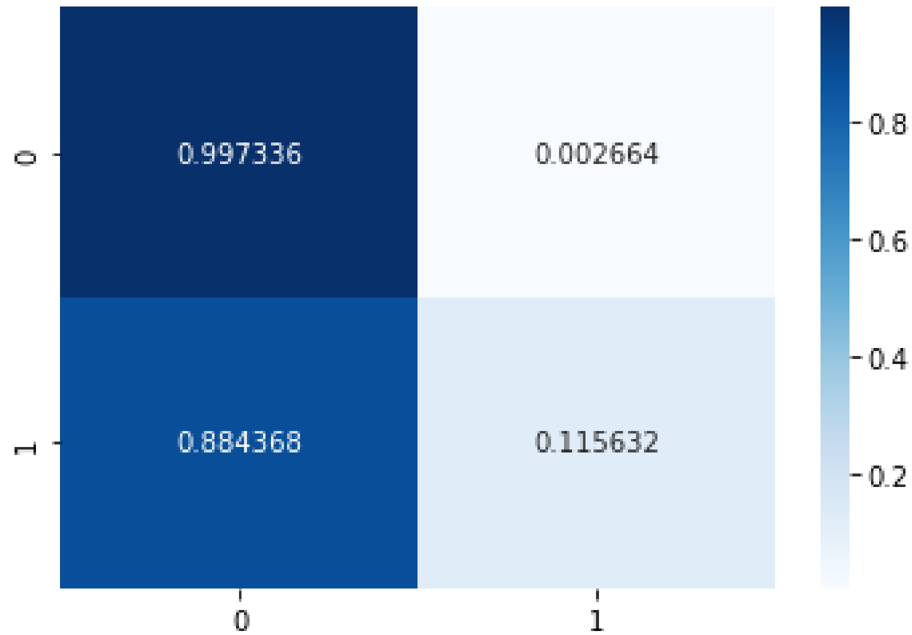
```
    accuracy                           0.89     29976
   macro avg       0.69      0.69      0.69     29976
weighted avg       0.89      0.89      0.89     29976

------------ Accuracy Score ------------
train :
1.0
test :
0.890245529757139

------------ Confusion matrix on test ------------
[[0.93843194 0.06156806]
 [0.55137335 0.44862665]]

False negative : 0.5513733468972533
False positive : 0.061568061568061 57
Test accuracy : 0.890245529757139
```



The accuracy for the test dataset remains equal to the baseline accuracy, but we can see that the accuracy of the train set is perfect ; the tree has overfit. To avoid that, we can choose a maximul depth.

```
=============== DecisionTreeClassifier(max_depth=3, random_state=0)
===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support
```

```
       False       0.91      1.00      0.95     54969
        True       0.85      0.13      0.23      5891

    accuracy                           0.91     60860
   macro avg       0.88      0.56      0.59     60860
weighted avg       0.91      0.91      0.88     60860

test :
              precision    recall  f1-score   support

       False       0.91      1.00      0.95     27027
        True       0.83      0.12      0.20      2949

    accuracy                           0.91     29976
   macro avg       0.87      0.56      0.58     29976
weighted avg       0.90      0.91      0.88     29976

------------ Accuracy Score ------------
train :
0.9135228393033191
test :
0.9105951427808914

------------ Confusion matrix on test ------------
[[0.997336   0.002664  ]
 [0.88436758 0.11563242]]

False negative : 0.8843675822312649
False positive : 0.002664002664002664
Test accuracy : 0.9105951427808914
```



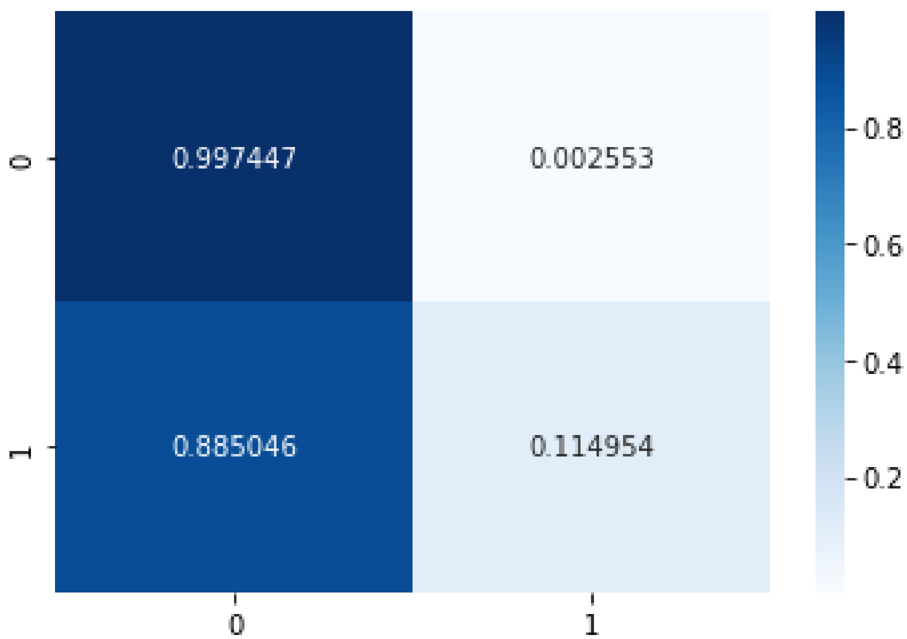
We now have a test accuracy of 91, which is better than the baseline accuracy. An other way to avoid overfit is **bagging**.

3.2.1 Bagging

The principle of bagging is to choose **n** random subsets from the training set, and train **n** decision trees on it. For each candidate in the test set, Random Forest uses the class with the majority vote as this candidate's final prediction.

```
=============== RandomForestClassifier(max_depth=3, n_estimators=50,
random_state=0) ===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support

       False       0.91      1.00      0.95     54969
        True       0.85      0.13      0.22      5891

    accuracy                           0.91     60860
   macro avg       0.88      0.56      0.59     60860
weighted avg       0.91      0.91      0.88     60860

test :
              precision    recall  f1-score   support

       False       0.91      1.00      0.95     27027
        True       0.83      0.11      0.20      2949
```

```
    accuracy                           0.91     29976
   macro avg       0.87      0.56      0.58     29976
weighted avg       0.90      0.91      0.88     29976

------------ Accuracy Score ------------
train :
0.9135721327637201
test :
0.9106285028022418

------------ Confusion matrix on test ------------
[[0.997447   0.002553  ]
 [0.88504578 0.11495422]]

False negative : 0.8850457782299085
False positive : 0.002553002553002553
Test accuracy : 0.9106285028022418
```

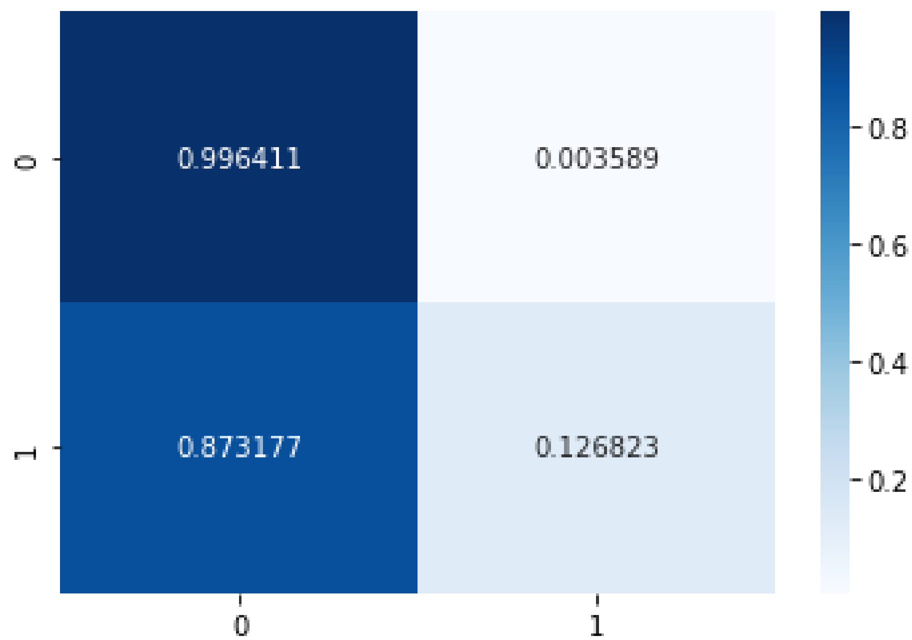


It improves the result only by 0.003% . Let's try another strategy, **Boosting**

3.2.2 Boosting

Boosting model's key is learning from the previous mistakes, e.g. misclassification data points. **n** estimators are trained sequentially ; they are trained with the residual errors of the previous tree, and the final prediction is made by simply adding up the predictions (of all trees).

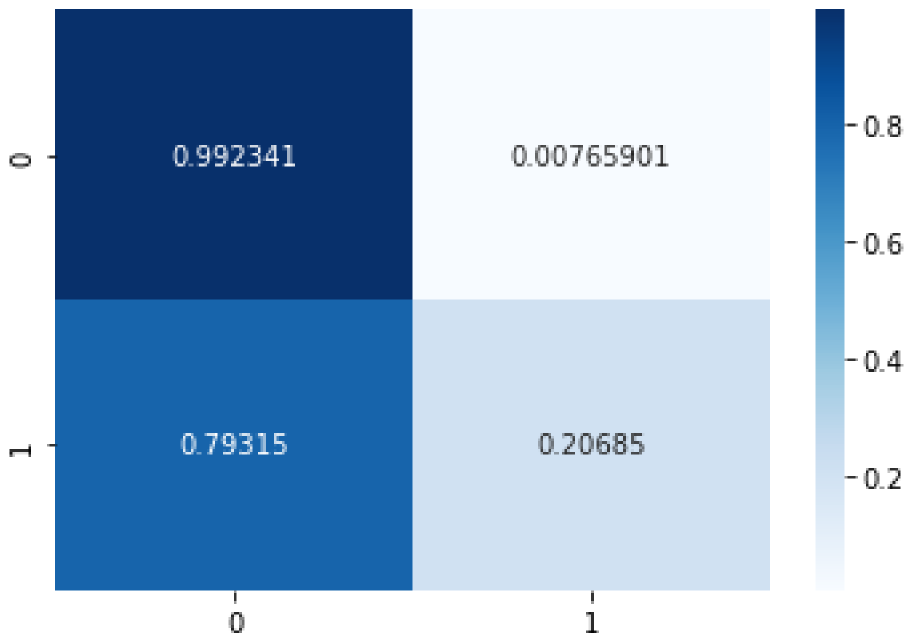
```
=============== GradientBoostingClassifier(n_estimators=50) ===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support

       False       0.92      1.00      0.95     54969
        True       0.81      0.14      0.24      5891

    accuracy                           0.91     60860
   macro avg       0.87      0.57      0.60     60860
weighted avg       0.91      0.91      0.89     60860

test :
              precision    recall  f1-score   support

       False       0.91      1.00      0.95     27027
        True       0.79      0.13      0.22      2949

    accuracy                           0.91     29976
   macro avg       0.85      0.56      0.59     29976
weighted avg       0.90      0.91      0.88     29976

------------ Accuracy Score ------------
train :
0.9139336181399934
test :
0.9108620229516947

------------ Confusion matrix on test ------------
[[0.996411   0.003589  ]
 [0.87317735 0.12682265]]

False negative : 0.8731773482536453
False positive : 0.003589003589003589
Test accuracy : 0.9108620229516947
```



We improved the result by 0.7 %. We can improve the accuracy further by increasing the number of estimators :

```
=============== GradientBoostingClassifier(n_estimators=500) ===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support

       False       0.93      0.99      0.96     54969
        True       0.83      0.26      0.40      5891

    accuracy                           0.92     60860
   macro avg       0.88      0.63      0.68     60860
weighted avg       0.92      0.92      0.90     60860

test :
              precision    recall  f1-score   support

       False       0.92      0.99      0.95     27027
        True       0.75      0.21      0.32      2949

    accuracy                           0.92     29976
   macro avg       0.83      0.60      0.64     29976
weighted avg       0.90      0.92      0.89     29976

------------ Accuracy Score ------------
train :
```

```
0.9233815313835031
test :
0.9150653856418468

------------ Confusion matrix on test ------------
[[0.99234099 0.00765901]
 [0.79315022 0.20684978]]

False negative : 0.7931502204136995
False positive : 0.007659007659007659
Test accuracy : 0.9150653856418468
```



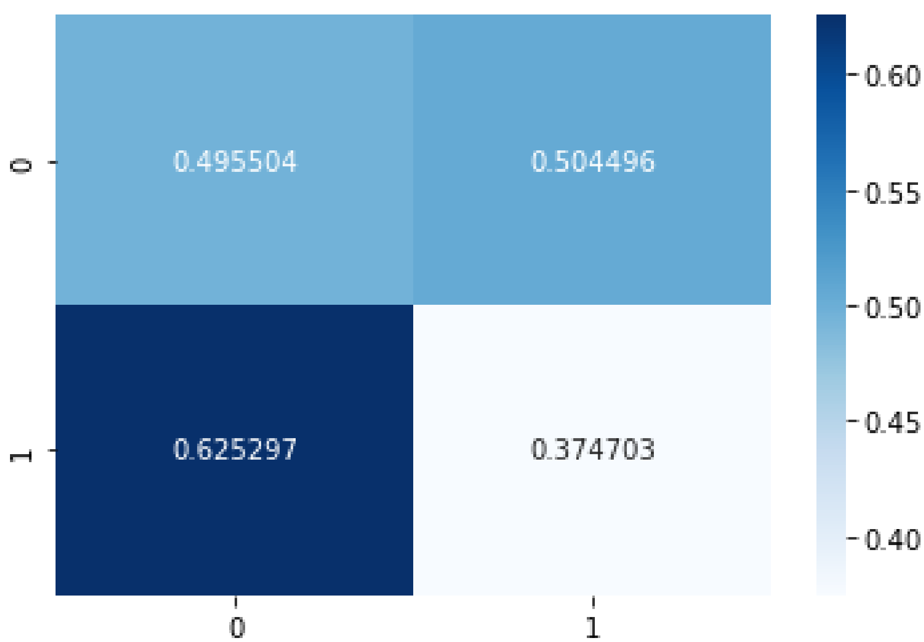
The accuracy improved by 0.42 %

3.2.3 Dimension Reduction to facilitate classification

We can try to reduce the dimension in order to accelerate the learning.

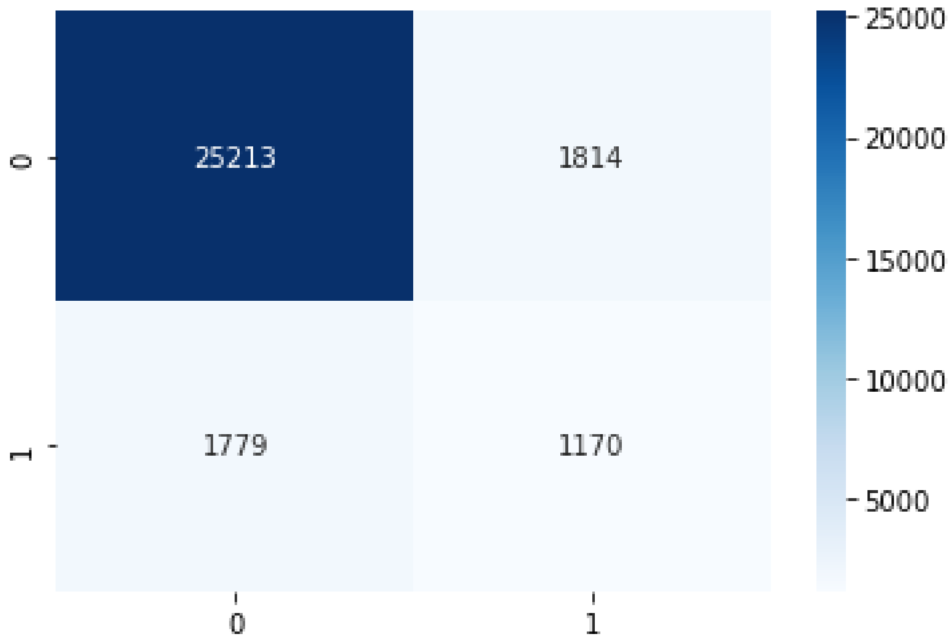
```
=============== Pipeline(steps=[('tree', LinearSVC(class_weight='balanced',
loss='hinge'))]) ===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support

       False       0.88      0.50      0.64     54969
        True       0.07      0.37      0.12      5891
```

```
    accuracy                           0.49     60860
   macro avg       0.48      0.43      0.38     60860
weighted avg       0.80      0.49      0.59     60860

test :
              precision    recall  f1-score   support

       False       0.88      0.50      0.63     27027
        True       0.07      0.37      0.12      2949

    accuracy                           0.48     29976
   macro avg       0.48      0.44      0.38     29976
weighted avg       0.80      0.48      0.58     29976

------------ Accuracy Score ------------
train :
0.4856720341767992
test :
0.4836202295169469

------------ Confusion matrix on test ------------
[[0.4955045  0.5044955 ]
 [0.62529671 0.37470329]]

False negative : 0.6252967107494066
False positive : 0.5044955044955045
Test accuracy : 0.4836202295169469
```

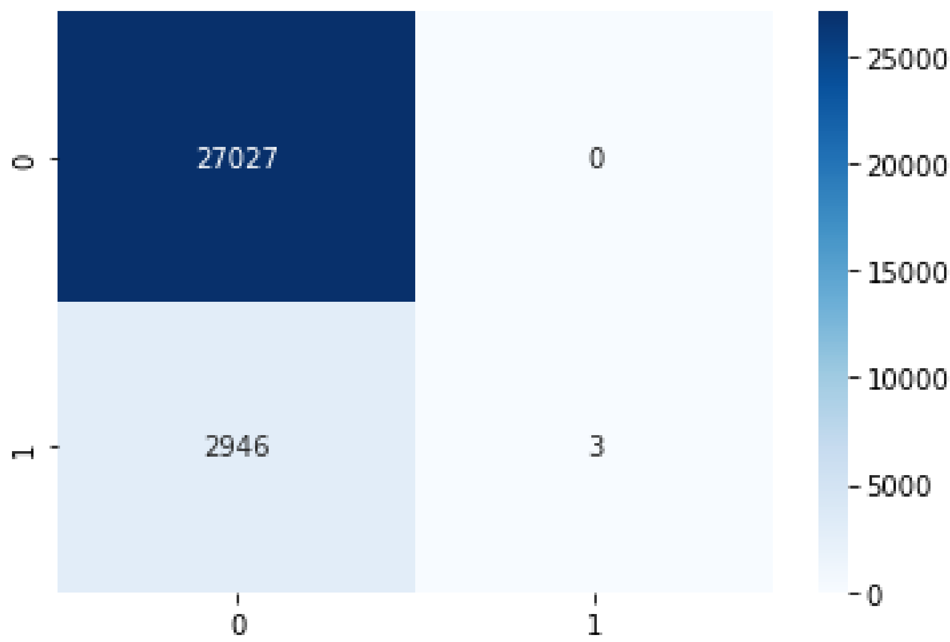


```
Score on training set : 1.0
Score on test set : 0.8801374432879637

Score on training set : 0.9034341110745975
Score on test set : 0.9017213771016813
```

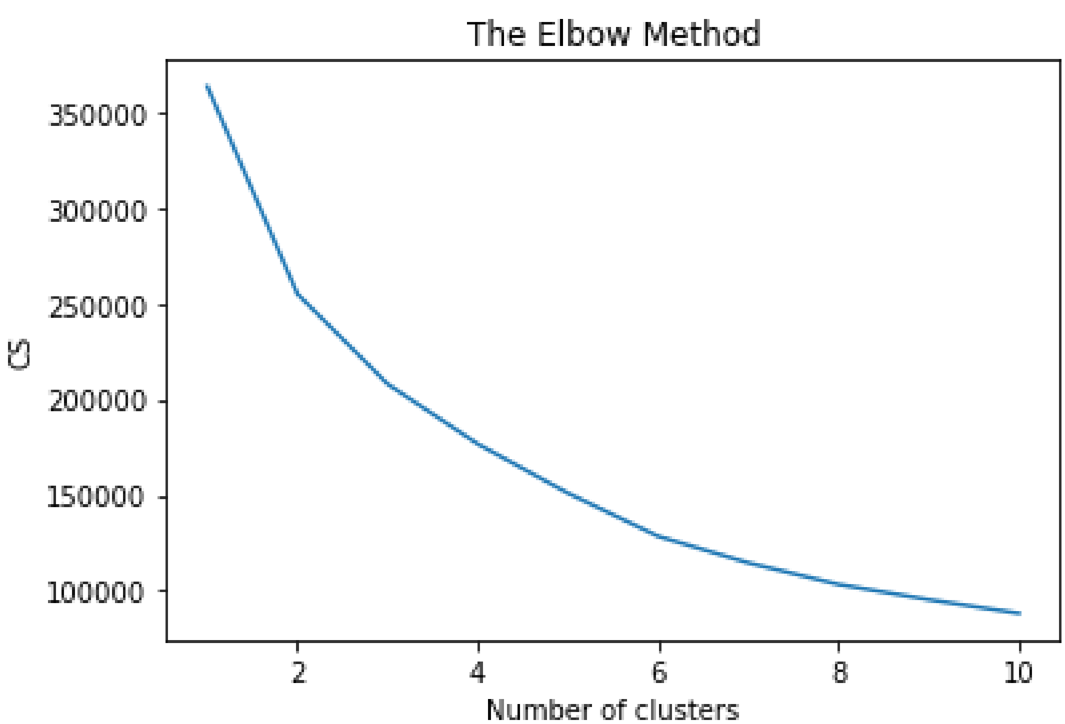
```
[[25213  1814]
 [ 1779  1170]]
False negative : 1779
False positive : 1814
```



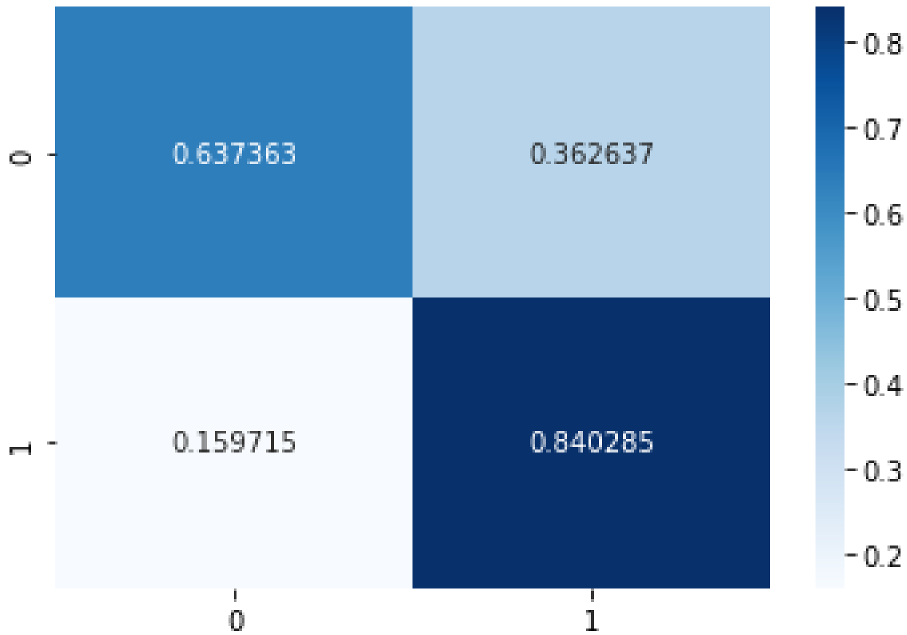

```
[[27027     0]
 [ 2946     3]]
False negative : 2946
False positive : 0
```



4 Unsupervised



```
Result: 29898 out of 90836 samples were correctly labeled.
=============== Pipeline(steps=[('scale', StandardScaler()),
                ('tree', KMeans(n_clusters=2, random_state=0))]) ===============
------------ Classification Report ------------
train :
              precision    recall  f1-score   support

       False       0.97      0.64      0.77     54969
        True       0.20      0.83      0.32      5891

    accuracy                           0.66     60860
   macro avg       0.59      0.73      0.55     60860
weighted avg       0.90      0.66      0.73     60860

test :
              precision    recall  f1-score   support

       False       0.97      0.64      0.77     27027
        True       0.20      0.84      0.33      2949

    accuracy                           0.66     29976
   macro avg       0.59      0.74      0.55     29976
weighted avg       0.90      0.66      0.73     29976

------------ Accuracy Score ------------
train :
0.6575254682878738
test :
0.6573258606885508

------------ Confusion matrix on test ------------
[[0.63736264 0.36263736]
 [0.15971516 0.84028484]]

False negative : 0.15971515768056968
False positive : 0.3626373626373626
Test accuracy : 0.6573258606885508
```



5 Conclusion

In the end we managed to try multiple models on our dataset, both supervised and unsupervised. In our case supervised learning seemed to be more appropriate and we decided to go further in this direction by trying more models. The unsupervised approach would probably require more data on each object to lead to a better detection of outliers. Another reason for the bad results of the unsupervised model may also be the unbaIn any case we managed to have a good accuracy with the supervised way. Finally our tentative to reduce the number of dimensions lead to acceptable results with a better speed.