Politechnic Institute of Setúbal

Barreiro School of Technology

"Big Data" Project

BSc in Bioinformatics

Road Safety Data - Accidents 2019

January 2021

Group II
Bernardo Augusto | 201800128
Miguel Cisneiros | 201800010

Contents

1 Introduction

Road traffic crashes result in over 1 million deaths of people around the world each year and leave between 20 and 50 million people with non-fatal injuries [1]. Most of the road traffic deaths and injuries involve vulnerable road users, such as pedestrians, cyclists and motorcyclists and their passengers.
Young people are the most vulnerable age group on the road, at a global level, and road traffic injuries are the main cause of death for people aged between 5 and 29. Statistics say that young males under 25 years are more likely to be involved in road traffic crashes than females, with 73 per cent of all road traffic deaths occurring among young males in that age.
Developing countries record higher rates of road traffic injuries, with 93 per cent of fatalities coming from low- and middle- income countries.
In addition to the human suffering, these injuries also incur a heavy economic burden on victims and their families, both through treatment costs for the injured and through loss of productivity of those killed or disabled. More broadly, road traffic injuries have a serious impact on national economies, costing countries 3 per cent of their annual gross domestic product.
There has been a continuous implementation of proven measures to reduce the risk of road traffic injuries and deaths, as well as the 2030 Agenda for Sustainable Development has set ambitious targets for reducing road traffic injuries.

For this project, a dataset with road safety data from the United Kingdom (UK) regarding the year of 2019 - available on UK's Government Open Data website - was assigned to this group.

Given the dataset, the main goals for this project are:

- to consolidate the learnings and applications of Pyspark (Python API for Spark's parallel and batch processing);
- to comprehend its usefulness for Big Data processing and Machine Learning computation.

In order to achieve these goals, the following tasks were executed:

1. Data Loading and Cleaning;
2. Exploratory Data Analysis;
3. Machine Learning Algorithms Implementation.

2 Data Cleaning

Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted. These data is usually not necessary or helpful when it comes to data analysis because it may hinder the process or provide inaccurate results. There are several methods for cleaning data depending on how it is stored along with the answers being sought. We began by looking at the null values. It could be observed that there were some missing values (figure 1) and that these could interfere with our data analysis. For that reason they had to be removed. For that removal was used the dropna() method, figure 2. For analysis proposes, the decision of dropping some columns was taken. The columns removed were, "Accident_Index" and "LSOA_of_Accident_Location", figure 3. These columns only contain index information, that has no use for data analysis. There were noticed some dummy values on some columns. These dummy values do not have meaning for the data analysis section and so they must be removed. For that procedure, we used the index position where the dummy values were and removed that lines, figure 4.

```
# discover how many data is missing
missing = dataset_pd.isnull().sum()

print(missing)
```

```
Accident_Index                                  0
Location_Easting_OSGR                          28
Location_Northing_OSGR                         28
Longitude                                      28
Latitude                                       28
Police_Force                                    0
Accident_Severity                               0
Number_of_Vehicles                              0
Number_of_Casualties                            0
Date                                            0
Day_of_Week                                     0
Time                                           63
Local_Authority_(District)                      0
Local_Authority_(Highway)                       0
1st_Road_Class                                  0
1st_Road_Number                                 0
Road_Type                                       0
Speed_limit                                     0
Junction_Detail                                 0
Junction_Control                                0
2nd_Road_Class                                  0
2nd_Road_Number                                 0
Pedestrian_Crossing-Human_Control               0
Pedestrian_Crossing-Physical_Facilities         0
Light_Conditions                                0
Weather_Conditions                              0
Road_Surface_Conditions                         0
Special_Conditions_at_Site                      0
Carriageway_Hazards                             0
Urban_or_Rural_Area                             0
Did_Police_Officer_Attend_Scene_of_Accident     0
LSOA_of_Accident_Location                    5714
dtype: int64
```

Figure 1: Missing values

```
# drop all the rows that have null values
dataset_pd = dataset_pd.dropna()
```

Figure 2: dropna() method

```
# we shall drop the Accident_Index and LSOA_of_Accident_Location collumn
# because these collumns will not be relevant for the exploratory analysis
dataset_pd = dataset_pd.drop(['Accident_Index', 'LSOA_of_Accident_Location'], axis=1)
```

Figure 3: Drop collumns, Accident_Index and LSOA_of_Accident_Location

```
# removes the -1 value that is a mistake on the dataset

# Get indexes where column has value -1
dataset_pd = dataset_pd[dataset_pd['Light_Conditions'] != -1]

# Get indexes where column has value -1
dataset_pd = dataset_pd[dataset_pd['Junction_Control'] != -1]

# Get indexes where column has value -1
dataset_pd = dataset_pd[dataset_pd['2nd_Road_Class'] != -1]

# Get indexes where column has value -1
dataset_pd = dataset_pd[dataset_pd['Pedestrian_Crossing-Human_Control'] != -1]

# Get indexes where column has value -1
dataset_pd = dataset_pd[dataset_pd['Pedestrian_Crossing-Physical_Facilities'] != -1]

# Get indexes where column has value -1
dataset_pd = dataset_pd[dataset_pd['Road_Surface_Conditions'] != -1]

# Get indexes where column has value -1
dataset_pd = dataset_pd[dataset_pd['Special_Conditions_at_Site'] != -1]

# Get indexes where column has value -1
dataset_pd = dataset_pd[dataset_pd['Carriageway_Hazards'] != -1]
```

Figure 4: Removal of the dummy values

3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) can help on the identification of obvious errors, as well as better understand patterns within the data, detect outliers or anomalous events, find interesting relations among the variables. Many data scientists use EDA to ensure the results they produce are valid and applicable to any desired business outcomes and goals. To avoid misleading information, data cleaning must be performed before the proceed to the analysis. That part was done, and so we can move to the data analysis section. We began this section by using the describe() method from pandas, in order to obtain some values, like count, mean, maximum and minimum, figure 5.

```
# Metrics
dataset_pd.describe()
```

	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitude	Police_Force	Accident_Severity	Number_of_Vehicles
count	62765.000000	6.276500e+04	62765.000000	62765.000000	62765.000000	62765.000000	62765.000000
mean	451654.501107	2.791784e+05	-1.260034	52.400516	26.832582	2.785709	1.868127
std	91611.309190	1.476246e+05	1.341546	1.329861	24.407876	0.433629	0.609778
min	75706.000000	2.056800e+04	-7.379107	50.042087	1.000000	1.000000	1.000000
25%	388162.000000	1.762400e+05	-2.178496	51.472876	4.000000	3.000000	2.000000
50%	457470.000000	2.223020e+05	-1.151018	51.885882	20.000000	3.000000	2.000000
75%	528068.000000	3.873040e+05	-0.154495	53.380262	44.000000	3.000000	2.000000
max	655244.000000	1.162603e+06	1.757476	60.343852	98.000000	3.000000	13.000000

8 rows × 27 columns

Figure 5: describe(), pandas method

After using the describe() method, we decided to take a graphical approach. For that, Seaborn, NumPy and Matplotlib were used. We began, by analysing the accident severity distribution on the year 2019, figure 6. We can notice that most of the accidents were located at level 3, which is the highest level of severity. Then, we decided to look at the distribution of the accident by light conditions, figure 7. In this distribution, it is possible to notice that most the accidents happen when the light conditions are lower (between level 1 and 2).
After this, we decided to evaluate if the day of the week has any influence on the number of accidents. But before we did that analysis, the professor gave us the suggestion of changing the number for the names of the days of the week, and that's what we did. For this, we used the replace() method, figure 8. And then we did the same as previously to obtain a graph of the distribution of accident during the week from 2019, figure 9. The distribution is almost even but Monday appears to be the day where there is less number of accidents.
We also decided to look if the road conditions had any influence on the number of accidents. The conclusion was that it had. According to the, figure 10, we can conclude that the worse the condition of the road, the higher the number of accidents.
In terms of weather conditions, there was a relation between the number of accidents and this variable. The worst the conditions the higher is the number of accidents, figure 11.
As for the number of casualties, it is visible on figure 12 that usually there is only one casualty.
On the number of vehicles, usually the accidents are between two vehicles, figure 13. Then we decided to verify if there was any variable correlated to another. For this a good option is to use a heatmap, which is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space. We can observe, in figure 14 that, there is a correlation between "Location_Easting_OSGR" and "Longitude", "Location_Northing_OSGR" and Latitude, and "Local_Authority" and "Police_Force". The location variables are

related because they all represent the same geographical data. We decided to remove the "Location_Easting_OSGR" and "Location_Northing_OSGR" columns because of personal preference. We came to the conclusion that the latitude and longitude may be a better option for geographical analysis if necessary. In terms of the others columns that were correlated we decided to drop the "Police_Force". After dropping these columns we obtained a heatmap, figure 15.

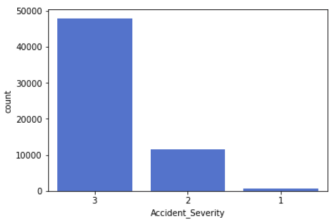

Figure 6: Number of Accidents by Accident Severity

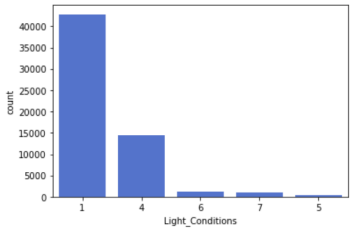

Figure 7: Number of Accidents by Light Conditions

```
# set the day of the week to names instead of numbers
dataset_pd["Day_of_Week"].replace({1: "Sun", 2: "Mon", 3: "Tue", 4: "Wed", 5: "Thu", 6: "Fri", 7: "Sat"}, inplace=True)
```

Figure 8: Change from numbers to names of the week

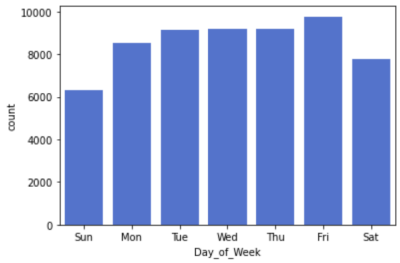

Figure 9: Number of Accidents by Day of the Week

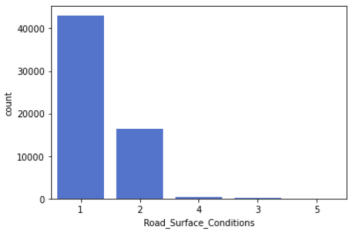

Figure 10: Number of Accidents by Road Surface Conditions

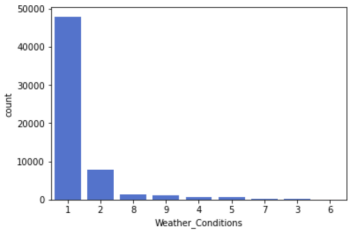

Figure 11: Number of Accidents by Weather Conditions

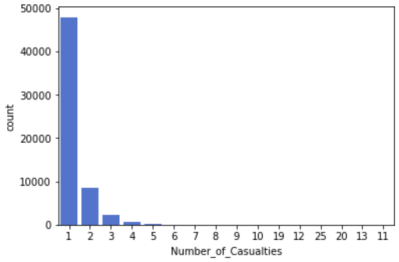

Figure 12: Number of Accidents by Number of Casualties

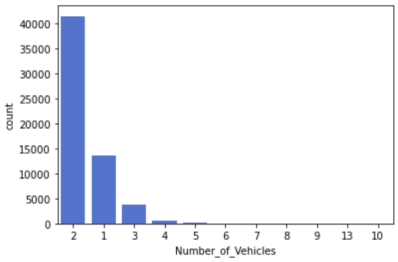

Figure 13: Number of Accidents by Number of Vehicles

Figure 14: Heatmap before the removal of the columns

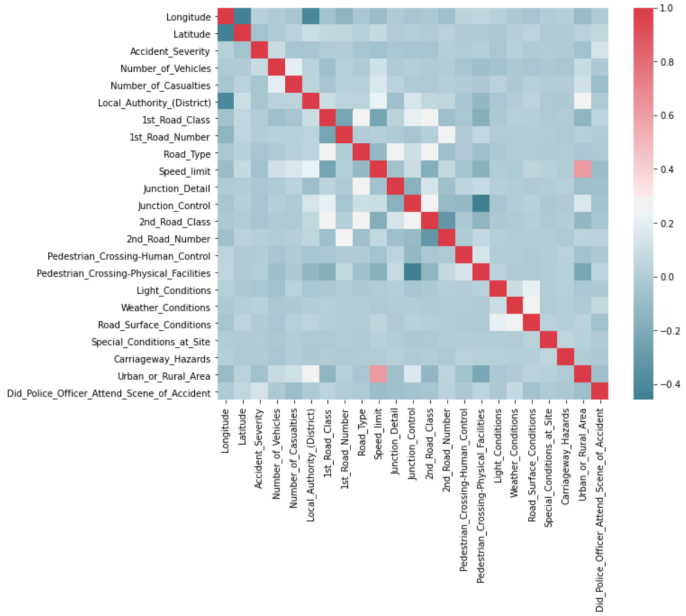

Figure 15: Heatmap after the removal of the columns

4 Machine Learning Implementation

Before the implementation of the Machine Learning (ML) algorithm/model, we had to drop the null and dummy values on the pyspark dataset, figure 16 and figure 17. The first ML algorithm we decided to use was a Logistic Regression. For that we had to remove 10 features, since some were indexes and other's type was not supported (figure 18). After this procedure, we had to build our Vector Assembler (figure 19). Then we standardized the features (figure 20). Afterwards, we had to convert the target variable into a binary. For that we decided to set the values 1 and 2 of Accident Severity as our 0 and value 3 as our 1 (figure 21). Then we applied the undersampling technique. This technique cuts some of the records of the major class to even the number of records in all classes. On this case, 1 (after the binary conversion). This will, basically, leverage the ratio to 1 (previously was 3, between the major and minor class). Then we performed the train and test split. We decided to split, 70 and 30 (figure 23). On figure 24, we are fitting the model to the data. As the accuracy was too high, the model was overfit to the data (figure 25). For this reason, besides the undersampling procedure, we decided to remove some features and to leave just our target variable (Accident_Severity) and Light, Weather and Road Conditions (figure 26). Then, we followed the same steps as before: undersampling, train and test split, fitting the model and accuracy verification. It was evident that the feature removal worsened the overfitting (figure 27).

After careful examination of the group and the professor, we came to the conclusion that the Vector Assembler was incorrectly built. That revealed to be the true cause of overfitting. The correct vector can be observed on figure 28. After this adjustment we obtained a much more reliable value for the accuracy (figure 29).

For our second algorithm, we decided to implement Clustering. Our group also attempted to implement Random Forest as a third ML algorithm but the accuracy for this model was 100 percent and, for that reason, we concluded that this was not a credible value and we rejected this algorithm (figure 30).

For Clustering, we used a "group by" function to group the features of interest, Accident_Severity and Number_of_Casualties (figure 31). Then we created our Vector Assembler, this time with only two features: the ones that were of our interest for analysis (figure 32). After training and testing, we concluded that the optimal number of clusters, in this case, was 2. This number was achieved using the Silhouette method, which is more detailed in figure 33. In simple terms, the higher the score of the Silhouette method, the more spaced are the clusters. On figure 33 we trained the model. After some time analysing the data, we came to the conclusion that these two clusters could match with the groups highlighted in figure 34. So, one cluster would be the one that contains the level 3 of accident severity with 1 casualty and the other would contain all the other levels with all the combinations with number of casualties.

```
dataset.na.drop("all")
```

```
DataFrame[Accident_Index: string, Location_Easting_OSGR: int, Location_Northing_OSGR: int, Longitude: double, Latitude: double, Police_Force: int, Accident_Severity: int, Number_of_Vehicles: int, Number_of_Casualties: int, Date: string, Day_of_Week: int, Time: string, Local_Authority_(District): int, Local_Authority_(Highway): string, 1st_Road_Class: int, 1st_Road_Number: int, Road_Type: int, Speed_limit: int, Junction_Detail: int, Junction_Control: int, 2nd_Road_Class: int, 2nd_Road_Number: int, Pedestrian_Crossing-Human_Control: int, Pedestrian_Crossing-Physical_Facilities: int, Light_Conditions: int, Weather_Conditions: int, Road_Surface_Conditions: int, Special_Conditions_at_Site: int, Carriageway_Hazards: int, Urban_or_Rural_Area: int, Did_Police_Officer_Attend_Scene_of_Accident: int, LSOA_of_Accident_Location: string]
```

```
from pyspark.sql.functions import isnan, when, count, col
dataset.select([count(when(isnan(c), c)).alias(c) for c in dataset.columns]).show()
```

Figure 16: Drop of the null values from the pyspark dataset

```
# Drop the -1 values from the spark dataset
dataset = dataset.filter((dataset["Light_Conditions"] != -1) & (dataset["Junction_Control"] != -1) & (dataset["2nd_Road_Class"] != -1)
                         & (dataset["Pedestrian_Crossing-Human_Control"] != -1) & (dataset["Pedestrian_Crossing-Physical_Facilities"] != -1)
                         & (dataset["Road_Surface_COnditions"] != -1) & (dataset["Special_Conditions_at_Site"] != -1)
                         & (dataset["Carriageway_Hazards"] != -1)
                         )
```

Figure 17: Drop of the dummy values from the pyspark dataset

```
# drop the indexes and the correlated variables
dataset = dataset.drop("Accident_Index", "Location_Easting_OSGR", "Location_Northing_OSGR", "Police_Force", "LSOA_of_Accident_Location")
```

```
# also discovered that these column have a type that is not supported
"""
Local_Authority_(Highway)
Time
Date
"""

dataset = dataset.drop("Local_Authority_(Highway)", "Time", "Date")
```

```
dataset = dataset.drop("Longitude", "Latitude")
```

Figure 18: Removal of 10 features from the pyspark dataset

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

```
# transformer
vector_assembler = VectorAssembler(inputCols=["Accident_Severity", "Number_of_Vehicles",
                                              "Number_of_Casualties", "Day_of_Week",
                                              "Local_Authority_(District)", "1st_Road_Class", "1st_Road_Number",
                                              "Road_Type", "Speed_limit", "Junction_Detail", "Junction_Control", "2nd_Road_Class",
                                              "2nd_Road_Number", "Pedestrian_Crossing-Human_Control", "Pedestrian_Crossing-Physical_Facilities",
                                              "Light_Conditions", "Weather_Conditions", "Road_Surface_Conditions", "Special_Conditions_at_Site",
                                              "Carriageway_Hazards", "Urban_or_Rural_Area", "Did_Police_Officer_Attend_Scene_of_Accident"
                                             ],outputCol="features")
output = vector_assembler.transform(dataset)
output = output.withColumn('target', output.Accident_Severity)
output.show(5)
```

Figure 19: Construction of the Vector Assembler

```
from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=False)

scalerModel = scaler.fit(output).transform(output)

scalerModel.select("features","scaledFeatures").show(5)
```

```
+--------------------+--------------------+
|            features|      scaledFeatures|
+--------------------+--------------------+
|[3.0,2.0,3.0,2.0,...|[6.91872245494149...|
|[3.0,2.0,1.0,3.0,...|[6.91872245494149...|
|[2.0,1.0,1.0,3.0,...|[4.61248163662766...|
|[3.0,2.0,2.0,3.0,...|[6.91872245494149...|
|[3.0,1.0,1.0,3.0,...|[6.91872245494149...|
+--------------------+--------------------+
only showing top 5 rows
```

Figure 20: Standardizing the features

```
scalerModel = scalerModel.withColumn("Accident_Severity", F.when(F.col("Accident_Severity")<=2,0).otherwise(F.when(F.col("Accident_Severity")>2,1)))
```

```
scalerModel = scalerModel.withColumn("Binary Target", scalerModel.Accident_Severity)

scalerModel.select("Target", "Binary Target").show(5)
```

```
+------+-------------+
|Target|Binary Target|
+------+-------------+
|     3|            1|
|     3|            1|
|     2|            0|
|     3|            1|
|     3|            1|
+------+-------------+
only showing top 5 rows
```

Figure 21: Convert target into binary

```
from pyspark.sql.functions import col, explode, array, lit

major_df = scalerModel.filter(col("Accident_Severity") == 1)
minor_df = scalerModel.filter(col("Accident_Severity") == 0)
ratio = int(major_df.count()/minor_df.count())
print("ratio: {}".format(ratio))
```

```
ratio: 3
```

```
sampled_majority_df = major_df.sample(False, 1/ratio)
combined_df_2 = sampled_majority_df.unionAll(minor_df)
scalerModel = combined_df_2
scalerModel.show()
```

Figure 22: Undersampling the data

```
final_data = scalerModel.select("scaledFeatures","Accident_Severity")

final_data.show(5)
```

```
+--------------------+-----------------+
|      scaledFeatures|Accident_Severity|
+--------------------+-----------------+
|[6.91872245494149...|                1|
|[6.91872245494149...|                1|
|[6.91872245494149...|                1|
|[6.91872245494149...|                1|
|[6.91872245494149...|                1|
+--------------------+-----------------+
only showing top 5 rows
```

```
train, test = final_data.randomSplit([0.7, 0.3],seed=1000)
print("training dataset:", str(train.count()))
print("test dataset:", str(test.count()))
```

```
training dataset: 20592
test dataset: 8964
```

Figure 23: Train and test split

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator



from sklearn import metrics
from sklearn.metrics import classification_report
```

```
lr = LogisticRegression(featuresCol = 'scaledFeatures',labelCol="Accident_Severity")

lrModel = lr.fit(train)
```

```
predict_train=lrModel.transform(train)
predict_test=lrModel.transform(test)
```

Figure 24: Logistic Regression

```
evaluator=BinaryClassificationEvaluator(rawPredictionCol="rawPrediction",labelCol="Accident_Severity")

predict_test.select("Accident_Severity","prediction").show(5)

print("Train score {}".format(evaluator.evaluate(predict_train)))
print("Test score {}".format(evaluator.evaluate(predict_test)))
```

```
+-----------------+----------+
|Accident_Severity|prediction|
+-----------------+----------+
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
+-----------------+----------+
only showing top 5 rows

Train score 0.9999997555481346
Test score 0.999999618852917
```

Figure 25: Accuracy before the feature removal

```
dataset = dataset.drop("Number_of_Vehicles", "Number_of_Casualties", "Day_of_Week","Local_Authority_(District)", "1st_Road_Class", "1st_Road_Number",
                       "Road_Type", "Speed_limit", "Junction_Detail", "Junction_Control", "2nd_Road_Class",
                       "2nd_Road_Number", "Pedestrian_Crossing-Human_Control", "Pedestrian_Crossing-Physical_Facilities",
                       "Special_Conditions_at_Site", "Carriageway_Hazards", "Urban_or_Rural_Area", "Did_Police_Officer_Attend_Scene_of_Accident")
```

```
# Schema
dataset.printSchema()
```

```
root
 |-- Accident_Severity: integer (nullable = true)
 |-- Light_Conditions: integer (nullable = true)
 |-- Weather_Conditions: integer (nullable = true)
 |-- Road_Surface_Conditions: integer (nullable = true)
```

Figure 26: Feature removal

```
evaluator=BinaryClassificationEvaluator(rawPredictionCol="rawPrediction",labelCol="Accident_Severity")

predict_test.select("Accident_Severity","prediction").show(5)

print("Train score {}".format(evaluator.evaluate(predict_train)))
print("Test score {}".format(evaluator.evaluate(predict_test)))
```

```
+-----------------+----------+
|Accident_Severity|prediction|
+-----------------+----------+
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
+-----------------+----------+
only showing top 5 rows

Train score 1.0
Test score 1.0
```

Figure 27: Accuracy after the feature removal

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

# transformer
vector_assembler = VectorAssembler(inputCols=["Number_of_Vehicles", "Number_of_Casualties", "Day_of_Week",
                                              "Local_Authority_(District)", "1st_Road_Class", "1st_Road_Number",
                                              "Road_Type", "Speed_limit", "Junction_Detail", "Junction_Control", "2nd_Road_Class",
                                              "2nd_Road_Number", "Pedestrian_Crossing-Human_Control", "Pedestrian_Crossing-Physical_Facilities",
                                              "Light_Conditions", "Weather_Conditions", "Road_Surface_Conditions", "Special_Conditions_at_Site",
                                              "Carriageway_Hazards", "Urban_or_Rural_Area", "Did_Police_Officer_Attend_Scene_of_Accident"
                                             ],outputCol="features")
output = vector_assembler.transform(dataset)
output = output.withColumn('target', output.Accident_Severity)
output.show(5)
```

Figure 28: Correct Vector Assembler without the Accident Severity

```
evaluator=BinaryClassificationEvaluator(rawPredictionCol="rawPrediction",labelCol="Accident_Severity")

predict_test.select("Accident_Severity","prediction").show(5)

print("Train score {}".format(evaluator.evaluate(predict_train)))
print("Test score {}".format(evaluator.evaluate(predict_test)))
```

```
+-----------------+----------+
|Accident_Severity|prediction|
+-----------------+----------+
|                1|       1.0|
|                1|       1.0|
|                1|       1.0|
|                1|       0.0|
|                1|       0.0|
+-----------------+----------+
only showing top 5 rows

Train score 0.6302466268000029
Test score 0.6337443454305766
```

```
print(classification_report(test.select("Accident_Severity").toPandas(), predict_test.select("prediction").toPandas()))
```

```
              precision    recall  f1-score   support

           0       0.56      0.37      0.44      3841
           1       0.63      0.78      0.70      5195

    accuracy                           0.61      9036
   macro avg       0.59      0.58      0.57      9036
weighted avg       0.60      0.61      0.59      9036
```

Figure 29: Accuracy of the Logistic Regression with the correct Vector

```
evaluator = MulticlassClassificationEvaluator(labelCol='labelIndexed', predictionCol='labelIndexed')

accuracy = evaluator.evaluate(predictions)

print(f"Accuracy = {accuracy*100}%")
```

```
Accuracy = 100.0%
```

Figure 30: Accuracy of the Random Forest with the correct Vector

```
col1_col2 = dataset.groupBy("Accident_Severity", "Number_of_Casualties").count()
```

```
print(col1_col2)
```

```
DataFrame[Accident_Severity: int, Number_of_Casualties: int, count: bigint]
```

Figure 31: Group by of the features that we want to cluster

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

# transformer
vector_assembler = VectorAssembler(inputCols=["Accident_Severity",
                                              "Number_of_Casualties"
                                             ],outputCol="features")
output = vector_assembler.transform(col1_col2)
output.show(5)
```

```
+-----------------+---------------------+-----+-----------+
|Accident_Severity|Number_of_Casualties|count|  features|
+-----------------+---------------------+-----+-----------+
|                3|                    1|40078| [3.0,1.0]|
|                2|                    2| 1667| [2.0,2.0]|
|                2|                   19|    1|[2.0,19.0]|
|                1|                    7|    2| [1.0,7.0]|
|                2|                    3|  528| [2.0,3.0]|
+-----------------+---------------------+-----+-----------+
only showing top 5 rows
```

Figure 32: Vector for the Kmeans algorithm

```
# Train a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(output.select("features"))
predictions = model.transform(output.select("features"))
```

```
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))
```

```
Silhouette with squared euclidean distance = 0.7674905938258503
```

Figure 33: Silhouette Score and training the kmeans algorithm

```
+-----------------+---------------------+-----+-----------+
|Accident_Severity|Number_of_Casualties|count|  features|
+-----------------+---------------------+-----+-----------+
|                3|                    1|40078| [3.0,1.0]|
|                2|                    2| 1667| [2.0,2.0]|
|                2|                   19|    1|[2.0,19.0]|
|                1|                    7|    2| [1.0,7.0]|
|                2|                    3|  528| [2.0,3.0]|
+-----------------+---------------------+-----+-----------+
only showing top 5 rows
```

Figure 34: Identifying Clusters

5 Conclusion

Road traffic crashes result in over 1 million deaths of people around the world each year and leave between 20 and 50 million people with non-fatal injuries [1]. Machine Learning and Data Analysis, on the issue of road traffic accidents, are particularly useful tools when it comes to road safety and prevention by identifying patterns and predicting outcomes. To learn with the tragedies of the past, for a brighter tomorrow.

With this work, we can conclude that the goals for this project were achieved with success. Despite the first ML algorithm not reaching our expectations at first, the mentoring received from Professor Raquel was determinant for the correct implementation of that algorithm. The second algorithm (Clustering) demonstrated to be an interesting approach for data analysis. A practical example of the applications of Custering, when using our dataset, is an automatic

classification of the accident severity by the emergency services upon contact, based on the description provided by the caller. As said before, we also tried to implement a third algorithm without success. This made us conclude that not all algorithms may fit our dataset and that, in the future, we must try to focus on optimizing those who have a better performance for completing the task. We also learned new things, beyond the scope of the subject, as over-sampling/undersampling that will certainly add value to our career/academic paths.

References

[1] World Health Organization. (2018). Global status report on road safety 2018. Retrieved January 9, 2021, from Who.int website:https://www.who.int/publications/i/item/9789241565684

[2] Databricks website, https://databricks.com/glossary/pyspark

[3] United Kingdom Governament website, https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data

[4] IBM website, https://www.ibm.com/cloud/learn/exploratory-data-analysis

[5] SISENSE website, https://www.sisense.com/glossary/data-cleaning/

[6] GeeksforGeeks website, https://www.geeksforgeeks.org/clustering-in-machine-learning/

[7] Medium website, https://medium.com/swlh/k-means-clustering-using-pyspark-on-data-bricks-fd65e207154a

[8] Medium website, https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d

[9] Medium website, https://medium.com/python-in-plain-english/decision-trees-random-forests-in-pyspark-d07546e4fa7d

[10] Medium website, https://medium.com/swlh/logistic-regression-with-pyspark-60295d41221