Predicting Traffic Accident Severity

NAME Ayush Singh **REGISTER NUMBER** 19BCE1813

NAME Aayush Kumar Singh **REGISTER NUMBER** 19BCE1113

A project report submitted to

Dr. S. Asha

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

in fulfilment of the requirements for the course of

CSE3021 – SOCIAL INFORMATION AND NETWORK

in

B. Tech. COMPUTER SCIENCE ENGINEERING

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Vandalur – Kelambakkam Road

Chennai – 600127

DECEMBER 2021

BONAFIDE CERTIFICATE

Certified that this project report entitled "**PREDICTING TRAFFIC ACCIDENT SEVERITY"** is a bonafide work of **AYUSH SINGH-19BCE1813 and AAYUSH KUMAR SINGH-19BCE1113** who carried out the Project work under my supervision and guidance for **CSE3021-SOCIAL INFORMATION AND NETWORKS.**

Dr. S. ASHA

Associate Professor Senior

School of Computer Science and Engineering (SCOPE),

VIT University, Chennai

Chennai – 600 127.

TABLE OF CONTENTS

1 Introduction

1.1 Background

Every year car accidents cause hundreds of thousands of deaths worldwide. Ac- cording to research conducted by the World Health Organization (WHO) there were 1.35 million road traffic deaths globally in 2016, with millions more sus- training serious injuries and living with long-term adverse health consequences. Globally, road traffic crashes are a leading cause of death among young people, and the main cause of death among those aged 15–29 years. Road traffic injuries are currently estimated to be the eighth leading cause of death across all age groups globally, and are predicted to become the seventh leading cause of death by 2030[1].

Leveraging the tools and all the information nowadays available, an extensive analysis to predict traffic accidents and its severity would make a difference to the death toll. Analyzing a significant range of factors, including weather conditions, locality, type of road and lighting among others, an accurate prediction of the severity of the accidents can be performed. Thus, trends that commonly lead to severe traffic incidents can help identifying the highly severe accidents. This kind of information could be used by emergency services, to send the exact required staff and equipment to the place of the accident, leaving more resources available for accidents occurring simultaneously. Moreover, this severe accident situation can be warned to nearby hospitals which can have all the equipment ready for a severe intervention in advance.

Consequently, road safety should be a prior interest for governments, local authorities and private companies investing in technologies that can help reduce accidents and improve overall driver safety.

1.2 Problem

Data that might contribute to determining the likeliness of a potential accident occurring might include information on previous accidents such as road conditions, weather conditions, exact time and place of the accident, type of vehicles involved in the accident, information on the users involved in the accident and off course the severity of the accident. This project aims to forecast the severity of accidents with previous information that could be given by a witness informing the emergency services.

1.3 Interest

Governments should be highly interested in accurate predictions of the severity of an accident, in order to reduce the time of arrival and to make a more efficient use of the resources, and thus save a significant amount of people each year. Others interested could be private companies investing in technologies aiming to improve road safeness.

2 Data

2.1 Data source

The data can be found in the following Kaggle data set click here.

2.2 Feature Selection

The data is divided in 5 different data sets, consisting of all the recorded accidents in France from 2005 to 2016. The *characteristics* data set contains information on the time, place, and type of collision, weather and lighting conditions and type of intersection where it occurred. The *places* data set has the road specifics such as the gradient, shape and category of the road, the traffic regime, surface conditions and infrastructure. On the *user* data set it can be found the place occupied by the users of the vehicle, information on the users involved in the accident, reason of traveling, severity of the accident, the use of safety equipment and information on the pedestrians. The *vehicle* data set contains the flow and type of vehicle, and the *holiday* one labels the accidents occurring in a holiday. All five data sets share the accident identifications number.

An initial analysis of the data was performed for the selection of the most relevant features for this specific problem, reducing the size of the dataset and avoiding redundancy, click here. With this process the number of features was reduced from 54 to 28.

2.3 Description

The dataset that resulted from the feature selection consisted in 839,985 samples, each one describing an accident and 29 different features.

These features where the following:

From the *characteristics* dataset: lighting, localization, type of intersection, atmospheric conditions, type of collisions, department, time and the coordinates which are described in the Kaggle dataset here. In addition, two new features were crafted, date to perform a seasonality analysis of the accident severity and weekend indicating if the accident occurred during the weekend or not.

Regarding the place's dataset, the selected features where: road category, traffic regime, number of traffic lanes, road profile, road shape, surface condition, situation, school nearby and infrastructure.

The user's dataset was used to craft some new features:

- number of users: total number of people involved in the accident.
- pedestrians: whether there were pedestrians involved (1) or not (0).
- critical age: whether there were users between 17 or 31 yr. involved inthe accident.
- severity: maximum gravity suffered by any user involved in the accident. Unscathed or light injury (0), hospitalized wounded or death (1)

The holiday dataset was used to add a last feature, labeling the accidents which occurred in a holiday.

2.4 Data Cleaning

The data cleaning is the process of giving a proper format to the data for its further analysis. The first step was to deal with missing values and outliers. Initially the latitude, longitude and road number were dropped from the data frame as more than a 50% of its values where NaN or 0 which is an outlier inthis case.

Then keeping with replacing the missing values, the analysis was divided in two groups of features. The first group had in all features a label which described *other cases*, for instance the feature describing the atmospheric conditions had a value of 9 for any other atmospheric condition not labeled with the other 8 values. Therefore, the missing values and outliers were replaced with the *other cases* label for the features of atmospheric conditions, type of collision, road category and the surface conditions. For the second group of features instead, the distribution of their values was analyzed. Then two features were dropped, the infrastructures and reserved lanes, as the outliers represented more than 75% of its data. Finally with the rest of the features with missing values, the traffic regime, the number of lanes, the road profile and shape and the situation at the time of the accident, the NaN and outliers were replaced with the feature'smost popular value.

Last format changes were performed to the school and department values. The school feature had all samples divided either in the 0 or the 100 values, thus all the 100 values were replaced with a 1. Similarly, the department feature had an extra 0 added at the unit's position, so all values were divided by 10.

Regarding the type of the data, all features had a coherent data type except for the date feature which was defined with the string type. I used the two data function of pandas to define the date feature with the *datetime* type. After all, 24 features remained.

3 Literature Review

Kumar et al. implemented k-means and Association Rule data mining approaches to identify the frequency of accident severity locations and to extract hidden information. From the total 158 locations; 87 of them were selected after removing accident location frequency count less than 20. Then k-means were applied to cluster into three groups, Number of clusters are determined by gap statistics. To get rules, they used minimum support of 5 percent. As a result, curved and slop on the hilly surface were revealed as accident prone locations.

These stated works concerned mostly on road accident analysis and pedestrian severity using Statistical methods. on the other hand some studies employed a data mining techniques (Decision Tree and MLP) on Weka tool focusing mainly on driver responsibility. Another study employed J48 and PART a data mining algorithm on driver and vehicle information considering as a major risk in accident severity on Weka tool.

4 Exploratory Data Analysis

First, the distribution of the target's values was visualized. The plot confirmed that it is a balanced labeled dataset as the samples are divided 56-54 with more cases of lower severity. Then a seasonality analysis was performed, visualizing the global trend of daily accidents as well as the number of accidents grouped by years, month of the year, and day of the week

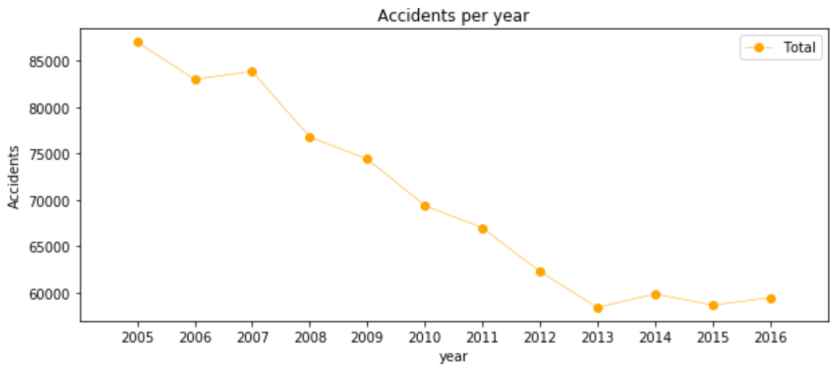


Figure 1: Lineplot of total amount of accidents per year.

The previous image show that the number of traffic accidents decreased over the years from 2005 to 2013, after which the trend became stable. Analyzing the yearly trend there is a seasonal pattern where the number of accidents increase around March and then again in September. This pattern can be seen in following two figures.

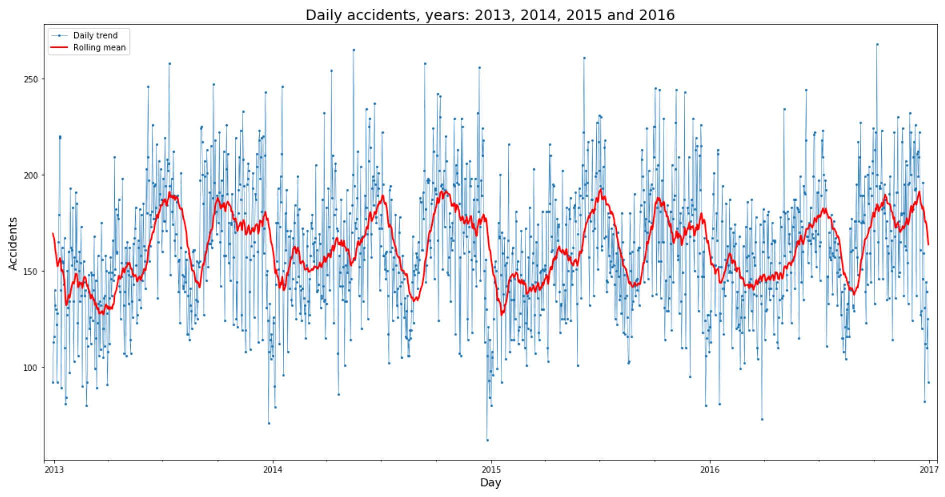


Figure 2: Lineplot of the amount of accident per day during the 2013, 2014, 2015 and 2016. The plot includes the rolling mean, with a window size of 30 days.

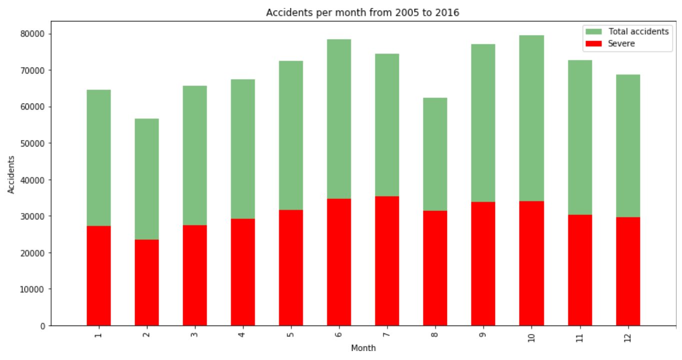


Figure 3: Barplot: Amount of accident per month from 2005 to 2016.

Regarding the day of the week there is not a significant difference between them, **Figure4**. There is a steady trend during the week with more accidents on Friday, and Sunday is the day with less recorded accident of all.

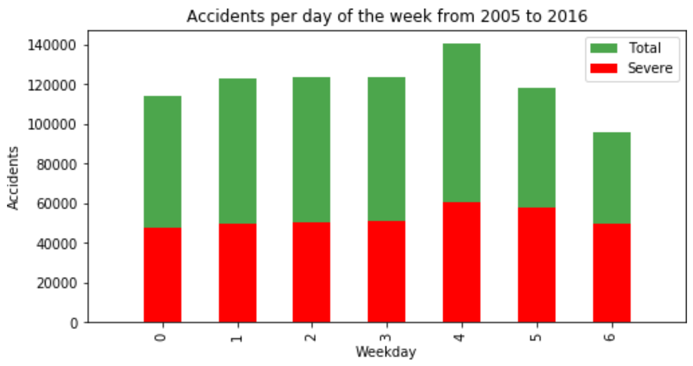


Figure 4: Barplot: Amount of accident per day of the week from 2005 to 2016.

Lastly analyzing the accidents per hour there are clearly two spikes, one at 8am, the time people go to work and another one between 5 and 6pm, time when people return home. The number of accidents decreases between these two spikes, nothing unusual but it proves there is a pattern here.

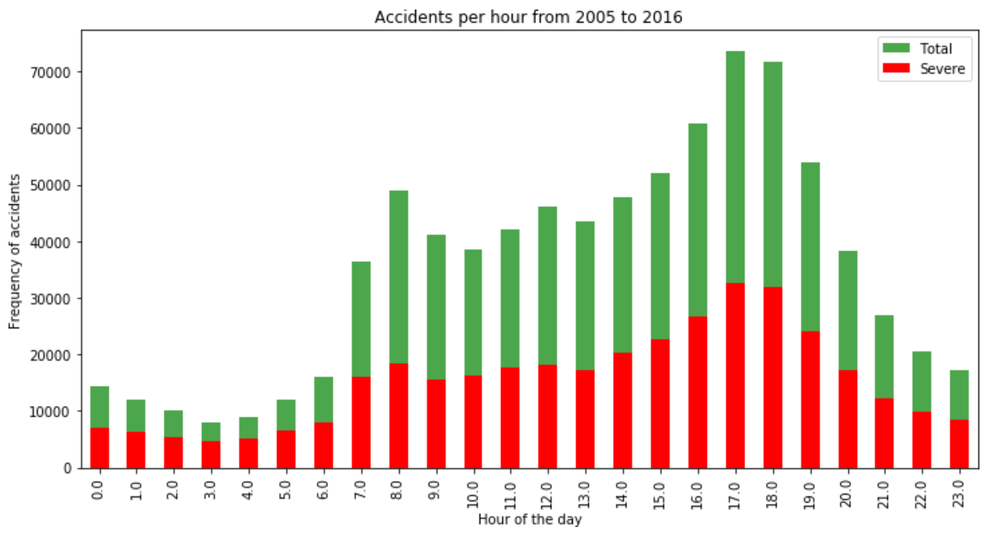


Figure 5: Lineplot of total amount of accidents per year.

The trend of highly severe accidents is proportional to the global trend, for both the accidents divided per month of the year and per day of the week. Same thing happens with the amount of highly severe accidents by hour of the day as we can see on **Figure:**5. One aspect to highlight from the hourly trend is that the proportion of severe accidents from noon to morning is higher, to be precise, the percentage of severe accidents from 9pm to 6am is 50.67% of the total amount of accidents occurring between these hours, while from 7am to 8pm is 42.41%. Due to the results of the former analysis, to features were added; month and day as the day of the month.
The next statistical analysis was the correlation of the features with the severity of an accident. The Pearson correlation showed weak or null correlation with all features. Further visualizations were performed for a better understanding of the data. Some conclusions of this analysis were for instance that accidents involving people above 84 years old tend to have a high severity.

5 Predictive Modeling

Different classification algorithms have been tuned and built for the prediction of the level of accident severity. These algorithms provided a supervised learning approach predicting with certain accuracy and computational time. These two properties have been compared in order to determine the best suited algorithm for his specific problem.

Firstly, the 839.985 rows where split 80/20 between the training and test sets, afterwards an additional 80/20 split was performed among the training samples creating the validation set for the development of the models. Then the data was standardized giving zero mean and unit variance to all features.

Four different approaches were used:

- Decision Tree, Random Forest
- Logistic Regression
- K-Nearest Neighbor
- Supervised Vector Machine

The same *modus operandi* was performed with each algorithm. With the train and validation sets the best hyperparameters were selected and using the test set the accuracy and computational time for the development of the models were calculated.

The decision tree model was upgraded to the random forest. With the default random forest, the features were sorted by impurity-based importance in the prediction of the severity. Thus, the 10 least important features were dropped to decrease the computation complexity for the **KNN** and **SVM** models. Keeping with 13 features the accuracy stayed the same and the computational time decreased significantly. After evaluating the parameters for each algorithm these were the models.

- Random Forest: 10 decision trees, maximum depth of 12 features and maximum of 8 features compared for the split.
- Logistic Regression: c=0.001.
- KNN: k=16
- SVM: size of the training set= 75,000 samples.

The following visualizations show how the parameters for KNN and SVM models were selected. The **SVM** model is computationally inefficient with huge sample sets. Therefore, an equilibrium between accuracy and computational time was a found evaluating different training sizes. The training set was reduced from 537,590 to 75,000 rows.

On **Figure:**7, the accuracy is increasing as the training size does, however **Figure:**9 shows how this comes with an important increasing of the computational time.

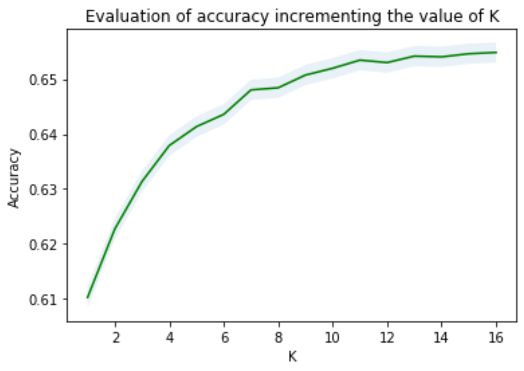


Figure 6: Accuracy of KNN models increasing the value of K.

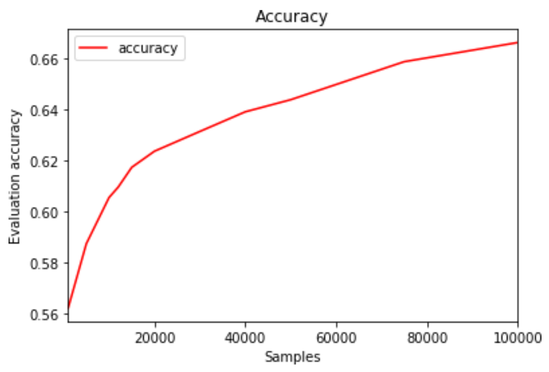


Figure 7: Accuracy of SVM increasing the training sample's size.

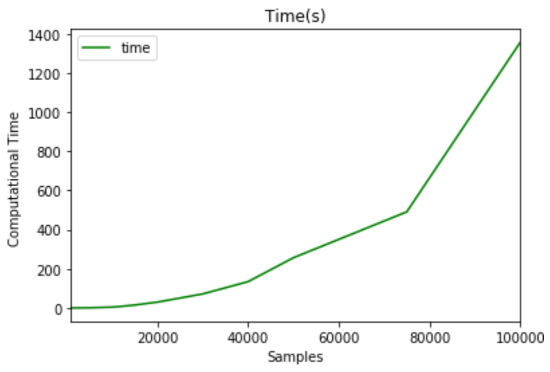


Figure 8: Computational time of SVM increasing the training sample's size.

6 Results

The metrics used to compare the accuracy of the models are the *Jaccard Score, f1-score, Precision*[1] and *Recall* [2]. This table reports the results of the evaluation of each model.

Algorithm	Jaccard	f1-score	Precision	Recall	Time (s)
Random Forest	0.722	0.72	0.724	0.591	6.588
Logistic Regression	0.661	0.65	0.667	0.456	6.530
KNN	0.664	0.66	0.652	0.506	200.58
SVM	0.659	0.65	0.630	0.528	403.92

In this case, the recall is more important than the precision as a high recall will favor that all required resources will be equipped up to the severity of the accident. The *logistic regression*, *KNN*, and *SVM* models have similar accuracy, however the computational time from the regression is far better than the other two models. With no doubt the *Random Forest* is the best model, in the same time as the *log. res.* it improves the accuracy from 0.66 to 0.72 and the recall from 0.45 to 0.59.

1.Proportion of predicted severe accidents that were truly severe

2.Proportion of truly severe accidents that were properly predicted

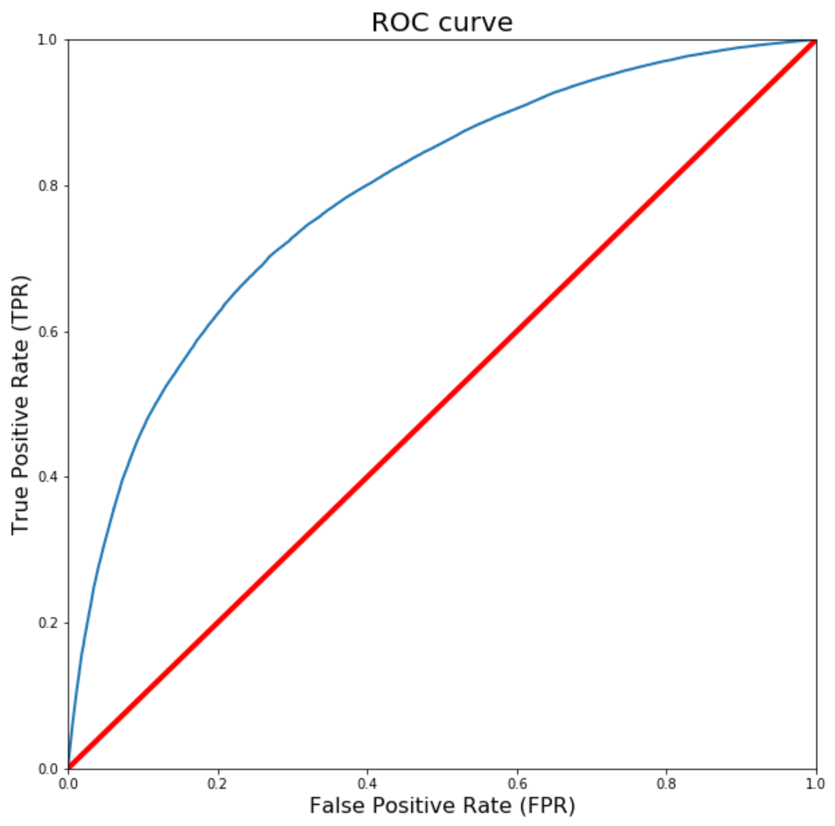


Figure 9: Representation of the ROC curve from the results of the Random Forest model.

We have also evaluated the best model using their ROC curves. In this particular problem, lower false positive rate is less important than higher true positive rate. In other words, it is more important to properly predict the high-severity accidents properly, if there is room for doubt it is better to prevent.

7 Conclusion

In this study, we analyzed the relationship between severity of an accident and some characteristics which describe the situation that involved the accident. We built and compared 4 different classification models to predict whether an accident would have a high or low severity. These models can have multiple application in real life. For instance, imagine that emergency services have an application with some default features such as date, time and department/municipality and then with the information given by the witness calling to inform on the accident they could predict the severity of the accident before getting there and so alert nearby hospitals and prepare with the necessary equipment and staff. Also, by identifying the features that favor the most the gravity of an accident, these could be tackled by improving road conditions or increasing the awareness of the population.

8. Appendices

8.1 Code

feature selection

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
df_char = pd.read_csv('caracteristics.csv', encoding='latin-1',
low_memory=False)
df_pl = pd.read_csv('places.csv')
df_users = pd.read_csv('users.csv')
df_veh = pd.read_csv('vehicles.csv')
df_holi = pd.read_csv('holidays.csv')
df_char.columns
df_char.drop(['adr','com', 'gps'], axis=1, inplace=True)
df_pl.drop(['v1', 'v2', 'pr', 'pr1', 'lartpc', 'larrout'], axis=1,  inplace=True)
df_veh.drop(['senc', 'occutc', 'obs', 'obsm', 'choc', 'manv', 'num_veh'],
axis=1, inplace=True)
print(df_char.shape, df_pl.shape, df_veh.shape, df_users.shape)
df_pl
df_char.columns = ['ID', 'year', 'month', 'day', 'time', 'lum', 'agg', 'int',
'atm', 'col', 'lat', 'long', 'dep']
df_pl.columns = ['ID','road_cat', 'road_num', 'traf_reg', 'num_lanes',
'res_lane', 'long_prof', 'shape', 'surf', 'infra', 'situation', 'school']
df = df_char.merge(df_pl, how='inner',on='ID')
df.head(5)
categories = df_veh['catv'].value_counts().sort_index()
print(categories.shape)
categories
df_users = pd.read_csv('users.csv')
df_users.head(5)
df_users.drop(['place', 'sexe', 'trajet', 'locp', 'actp', 'etatp', 'num_veh'],
axis=1, inplace=True)
df_users.columns = ['ID', 'catu', 'grav', 'secu', 'year_birth']
df_users.sort_values(by='ID').head(10)
```

```
num_users = df_users.ID.value_counts().sort_index(ascending=True)
num_users
df_users['ped'] = df_users['catu'].apply(lambda x: 1 if x==3 else 0)
df_users.head(5)
df_users2 = df_users.groupby('ID').sum()
ped = df_users2.ped
print('Accidents in which pedestrians have been involved:')
df_users2.ped.value_counts()
acc_year = df_users.ID.astype(str).str[:4]
age = acc_year.astype(int) - df_users['year_birth']
df_users['age']= age
df_users
df2 = df_users[df_users['grav']==2]
deaths = df2['age'].value_counts()
deaths.sort_values(ascending=False)

total, suma, i = deaths.values.sum(), 0, 0

for num in deaths.values:
    suma += num
    per = (suma/total)*100
    if per<=50:
        i += 1
        percentage = per
print('A % 2.2f percent of a total of %4i deaths is found on the %2i first ages, being the deaths array sorted by number of deaths.' %(percentage,total,i))
topdeaths = deaths.head(15)
print('Half of the deaths in a car accident are aged between % 2d and % 2.0i y.o.'
      % (topdeaths.index.min(), topdeaths.index.max()))
topdeaths
pd.DataFrame(deaths)
dplot=deaths.sort_index()
dplot.plot.bar(figsize=(14,6))
plt.annotate('  50% of deaths',
             color='r',
            xy=(17,1650),
            xytext=(31,1635),
            arrowprops=dict(arrowstyle='<->', color='r')
            )
```

```
plt.axvline(x=17, color='g', linestyle='-')
plt.axvline(x=31, color='g', linestyle='-')
plt.xlabel('Age', size=12)
plt.ylabel('number of deaths', size=12)
plt.title('Total number of deahts by age from 2005 to 2016', size=20)
death = df2['age'].value_counts().sort_index()
total = df_users['age'].value_counts().sort_index()
death_prop = (deaths/total)*100
d = death_prop.dropna()
minimum = d[d>9.9].index.min()
d.plot.bar(figsize=(14,6))
plt.title('Percentage of fatal accidents by age', size=20)
plt.xlabel('Age', size=12)
plt.ylabel('%', size=12)
plt.axhline(y=10, color='r')
plt.axvline(x=84, color='r')
print('People older than {0:2} are more likely to die\
 with at least 1 out of 10 of the times, rising to beyond 35% in some
cases.'.format(int(minimum)))
df_users['crit_age'] = df_users['age'].apply(lambda x: 1 if 17<=x<=31
else 0)
df_users['dead_age'] = df_users['age'].apply(lambda x: 1 if x>84 else 0)
df_users.head(5)
df2 = df_users.groupby('ID').sum().sort_index(ascending=True)
df2['crit_age1'] = df2['crit_age'].apply(lambda x: 1 if x>0 else 0)
df2['dead_age1'] = df2['dead_age'].apply(lambda x: 1 if x>0 else 0)
df2['ped1'] = df2['ped'].apply(lambda x: 1 if x>0 else 0)
df2['num_us'] = num_users
df2
def change_grav(x):
    if x==1 or x==4:
        return 0
    else:
        return 1
df3 = df_users[['ID', 'grav']].copy()
df3['grav'] = df_users['grav'].apply(change_grav)
df3.sort_values(by='ID')
severity=df3[['grav','ID']].groupby('ID').max().sort_index
(ascending=True)
severity
df2['sev'] = severity
df2.reset_index(inplace=True)
```

```
df2
print('Both data frames having same number of rows is an indicator that
any accident has been left behind')
df.shape, df2.shape
df = df.merge(df2[['ID','crit_age1', 'ped1','dead_age1', 'num_us', 'sev']],
on='ID', how='left')
df
from datetime import datetime
dt = df[['ID','year', 'month', 'day', 'time' ]]
date = (dt.year+2000)*10000+dt.month*100+dt.day
dt['date'] = pd.to_datetime(date, format='%Y%m%d')
dt['weekday'] = dt['date'].dt.weekday
dt['weekend'] = dt['weekday'].apply(lambda x: 1 if x>4 else 0)
dt.loc[dt.date.isin(df_holi.ds) , 'holiday'] = 1
dt.holiday.fillna(0, inplace=True)
dt.head(5)
df = df.merge(dt[['ID', 'date', 'weekend', 'holiday']],on='ID', how='left')
df.drop(['year', 'month', 'day'], axis=1, inplace=True)
df.head()
df['time'] = df.time.div(100).apply(np.floor)
df.rename(columns={'crit_age1':'crit_age','dead_age1':'dead_age',
'ped1':'ped'},inplace=True)
df.to_csv('Data.csv')
```

Main notebook

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import datetime as dt
import warnings
warnings.filterwarnings('ignore')
df = pd.read_csv('Data.csv', index_col=0)
df.head(10)
df.info()
df['road_num'].describe()
df.drop(['lat', 'long', 'road_num'], axis=1, inplace=True)
print('Missing values in atm:', df["atm"].isna().sum(),'\n'
   'Missing values in collision:', df["col"].isna().sum(), '\n'
   'Missing values in road_cat:', df["road_cat"].isna().sum(),'\n'
   'Missing values in surf:', df["surf"].isna().sum())
df['atm'].hist(alpha=0.5, rwidth=0.35, align='mid', figsize=(12,6), label='atm')
df['col'].hist(alpha=0.5, rwidth=0.35, align='mid', label='collision')
df['road_cat'].hist(alpha=0.6, rwidth=0.35, align='left', label='road_cat')
df['surf'].hist(alpha=0.3,rwidth=0.35, align='left', label='surf')
plt.title('Frequency of the values in 4 different features', size=12)
plt.xticks(range(10))
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
df['atm'].fillna(9, inplace=True)
df['col'].fillna(6, inplace=True)
df['road_cat'].fillna(9, inplace=True)
df['surf'].fillna(9, inplace=True)
df['surf'].replace(0,9, inplace=True)
df.surf.value_counts()
df[['traf_reg', 'num_lanes','res_lane', 'long_prof', 'shape', 'infra', 'situation']].describe()
df.drop(['infra', 'res_lane'], axis=1, inplace=True)
df['num_lanes'].value_counts()
df.num_lanes.fillna(0, inplace=True)
df['num_lanes'] = df['num_lanes'].apply(lambda x: 2 if x>6 or x==0 else x)
df.num_lanes.value_counts()
```

```
df['traf_reg'].hist(alpha=0.5,   rwidth=0.35,   align='left',   figsize=(12,6), label='traf_reg')
df['long_prof'].hist(alpha=0.5,rwidth=0.35, align='left', label='long_prof')
df['shape'].hist(alpha=0.5,rwidth=0.35, align='mid', label='shape')
df['situation'].hist(alpha=0.5,rwidth=0.35, align='mid', label='situation')
plt.title('Frequency of the values in 4 different features', size=12)
plt.xticks(range(6))
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
df['traf_reg'].fillna(0, inplace=True)
df['traf_reg'] = df['traf_reg'].replace(0,2)
df['long_prof'].fillna(0, inplace=True)
df['long_prof'] = df['long_prof'].replace(0,1)
df['shape'].fillna(0, inplace=True)
df['shape'] = df['shape'].replace(0,1)
df['situation'].fillna(0, inplace=True)
df['situation'] = df['situation'].replace(0,1)
df.school.describe(), df.school.hist()
plt.title('School feature values')
df.school.fillna(0, inplace=True)
df['school'] = df.school.apply(lambda x:1 if x>0 else 0)
df["dep"] = df["dep"].div(10).apply(np.floor)
df["dep"] = df["dep"].astype(int)
df.info()
df.sev.plot.hist(figsize=(4,3))
plt.title('Target feature values')
plt.xlabel('Severity')
plt.ylabel('Frequency')
print('Accidents classified in each level of severity:')
print(df.sev.value_counts())
df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')
date = df[['ID','sev', 'date']]
date.date
date['year'] = df.date.dt.year
date['month'] = df.date.dt.month
date['weekday'] = df.date.dt.weekday
high_sev = date[date['sev']==1]
season = date[['date', 'ID']].groupby('date').count()
season['rolling'] = season.ID.rolling(window=30).mean()
season['ID'][365*8:].plot(figsize=(20,10),    marker='o',    markersize=2, linewidth=0.5, label='Daily trend')
```

```
season['rolling'][365*8:].plot(color='r', linewidth=2, label='Rolling
mean')
plt.title('Daily accidents, years: 2013, 2014, 2015 and 2016', size=18)
plt.xlabel('Day', size=14)
plt.ylabel('Accidents', size=14)
t0 = dt.datetime.strptime('2012-12-15', '%Y-%m-%d')
t1 = dt.datetime.strptime('2017-01-15', '%Y-%m-%d')
plt.xlim(t0,t1)
plt.legend()
plt.show()
yearly = date[['year', 'ID']].groupby('year').count()
yearly['ID'].plot.line(figsize=(10,4), marker='o', linewidth=0.5,
color='orange', label='Total')
plt.title('Accidents per year')
plt.xticks(range(2005,2017))
plt.xlim(2004,2017)
plt.ylabel('Accidents')
plt.legend()
monthly = date[['month', 'ID']].groupby(['month']).count()
monthly['high_sev']=high_sev[['month', 'ID']].groupby(['month']).count()
monthly['ID'].plot.bar(figsize=(14,7), alpha=0.5, color='g', label='Total
accidents')
monthly['high_sev'].plot.bar(color='r', label='Severe')
plt.title('Accidents per month from 2005 to 2016')
plt.xticks(range(13))
plt.xlim(-1,12)
plt.xlabel('Month')
plt.ylabel('Accidents')
plt.legend()
weekday = date[['weekday', 'ID']].groupby('weekday').count()
weekday['high_sev'] = high_sev[['weekday',
'ID']].groupby(['weekday']).count()
weekday['ID'].plot.bar(figsize=(8,4), alpha=0.7, color='g', label='Total')
weekday['high_sev'].plot.bar(color='r', label='Severe')
plt.title('Accidents per day of the week from 2005 to 2016')
plt.xticks(range(7))
plt.xlim(-1,7)
plt.xlabel('Weekday')
plt.ylabel('Accidents')
plt.legend()
hourly = df[['ID', 'time']].groupby('time').count()
hourly['high_sev'] = df[df.sev==1][['ID', 'time']].groupby('time').count()
```

```
hourly['ID'].plot.bar(figsize=(12,6), alpha=0.7, color='g', label='Total')
hourly['high_sev'].plot.bar(color='r', label='Severe')
plt.xticks(range(24))
plt.title('Accidents per hour from 2005 to 2016')
plt.xlabel('Hour of the day')
plt.ylabel('Frequency of accidents')
plt.legend()
hourly['ID'].sum()
hourly['high_sev'].plot.bar(figsize=(12,6),color='r', label='Fatal')
plt.xticks(range(24))
plt.ylim((0,35000))
plt.title('Fatal accidents per hour from 2005 to 2016')
plt.xlabel('Hour of the day')
plt.ylabel('Frequency of accidents')
plt.legend()
noon_morn_severe =
hourly.high_sev.loc[0:6].sum()+hourly.high_sev.loc[21:23].sum()
day_severe = hourly.high_sev.loc[7:20].sum()
noon_morn = hourly.ID.loc[0:6].sum()+hourly.ID.loc[21:23].sum()
day = hourly.ID.loc[7:20].sum()
noon_morn_prop = (noon_morn_severe/noon_morn)*100
day_prop = (day_severe/day)*100
print('The percentage of severe accidents from 9pm to 6am is {0:0.2f}%
of the total amount of accidents ocurring between this hours,\
    while the percentage of deathly accidents from 7am to 8pm is
{1:2.2f}%.'.format(noon_morn_prop.round(2), day_prop))
df['month'] = df.date.dt.month
df['day'] = df.date.dt.day
df.day.value_counts
df[['sev','lum', 'agg', 'int', 'atm',
   'col', 'dep', 'road_cat', 'traf_reg',
   'num_lanes', 'long_prof', 'shape', 'surf',
   'situation', 'school', 'crit_age','dead_age', 'ped',
   'num_us', 'weekend', 'holiday', 'month',
'day']].corr()['sev'].sort_values(ascending=False)
bins = np.linspace(df.atm.min(), df.atm.max(), 10)
g = sns.FacetGrid(df, col="crit_age", hue="sev", palette="Set1",
col_wrap=2)
g.map(plt.hist,'road_cat', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
bins = np.linspace(df.lum.min(), df.lum.max(), 10)
```

```
g = sns.FacetGrid(df, col="crit_age", hue="sev", palette="Set1", col_wrap=2)
g.map(plt.hist, 'num_us', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
bins = np.linspace(df.col.min(), df.col.max(), 10)
g = sns.FacetGrid(df, col="crit_age", hue="sev", palette="Set1", col_wrap=2)
g.map(plt.hist, 'col', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
bins = np.linspace(df.surf.min(), df.surf.max(), 10)
g = sns.FacetGrid(df, col="crit_age", hue="sev", palette="Set1", col_wrap=2)
g.map(plt.hist, 'surf', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
df['sev'][df['dead_age']==1].plot.hist(color='r')
plt.title('Accident severity for poeple above 84 y.o.')
plt.xlabel('Severity')
df['sev'][df['dead_age']==0].plot.hist()
plt.title('Accident severity for poeple under 84 y.o.')
plt.xlabel('Severity')
df.drop(['ID', 'date'], axis=1, inplace=True)
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(df.drop('sev', axis=1), df['sev'], test_size=0.2, random_state=8)
xtrain, xval, ytrain, yval = train_test_split(xtrain, ytrain, test_size=0.2)
print('Size of training set:', xtrain.shape[0],'\n'
      'Size of test set:',xtest.shape[0],'\n'
      'Size of evaluation set:', xval.shape[0])
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import time
from sklearn.metrics import accuracy_score, log_loss, jaccard_score, classification_report
from sklearn.metrics import precision_score, recall_score, roc_curve
t0=time.time()
```

```
tree = DecisionTreeClassifier(criterion='entropy')
tree.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = tree.predict(xval)
score_tree = accuracy_score(yval,yhat)
print('Accuracy :',score_tree)
t0=time.time()
model_rf                                                                         =
RandomForestClassifier(n_estimators=100,criterion='entropy',random_st
ate=0, n_jobs=-1)
model_rf.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = model_rf.predict(xval)
score_rf = accuracy_score(yval,yhat)
print('Accuracy :',score_rf)
importances=pd.DataFrame({'feature':df.drop('sev',
axis=1).columns,'importance':np.round(model_rf.feature_importances_,3
)})
importances=importances.sort_values('importance',ascending=False).set
_index('feature')
importances
xtrain = pd.DataFrame(xtrain)
xtrain.drop(['dead_age', 'holiday', 'ped', 'situation', 'weekend', 'school',
'shape', 'crit_age', 'surf', 'long_prof','lum','atm'], axis=1, inplace=True)
xval.drop(['dead_age', 'holiday', 'ped', 'situation', 'weekend', 'school',
'shape', 'crit_age', 'surf', 'long_prof', 'lum', 'atm'], axis=1, inplace=True)
xtest.drop(['dead_age', 'holiday', 'ped', 'situation', 'weekend', 'school',
'shape', 'crit_age','surf', 'long_prof', 'lum', 'atm'], axis=1, inplace=True)
t0=time.time()
model_rf                                                                         =
RandomForestClassifier(n_estimators=100,criterion='entropy',random_st
ate=0, n_jobs=-1)
model_rf.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = model_rf.predict(xval)
score_rf = accuracy_score(yval,yhat)
print('Accuracy :',score_rf)
t0=time.time()
model_rf = RandomForestClassifier(n_estimators=50, max_features=5,
max_depth =10 ,criterion='entropy',random_state=0, n_jobs=-1)
model_rf.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = model_rf.predict(xval)
score_rf = accuracy_score(yval,yhat)
```

```
print('Accuracy :',score_rf)
t0=time.time()
model_rf = RandomForestClassifier(n_estimators=10, max_features=8,
max_depth =12,criterion='entropy',random_state=0, n_jobs=-1)
model_rf.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = model_rf.predict(xval)
score_rf = accuracy_score(yval,yhat)
print('Accuracy :',score_rf)
t0=time.time()
model_rf = RandomForestClassifier(n_estimators=10, max_features=8,
max_depth =12,criterion='entropy',random_state=0, n_jobs=-1)
model_rf.fit(xtrain,ytrain)
t_rf = time.time()-t0
print('Time taken :' , t_rf)
yhat_rf = model_rf.predict(xtest)
jaccard_rf = jaccard_score(ytest,yhat_rf)
c_rf = classification_report(ytest,yhat_rf)
prec_rf = precision_score(ytest, yhat_rf)
rec_rf = recall_score(ytest, yhat_rf)
print('Jaccard :',jaccard_rf,'\n',
     c_rf)
acc=np.zeros(6)
i=0
for c in [0.5, 0.1, 0.01, 0.001, 10, 100]:
    lr = LogisticRegression(C=c, solver='liblinear').fit(xtrain, ytrain)
    yhat = lr.predict(xval)
    acc[i] = accuracy_score(yval,yhat)
    i+=1
acc
t0=time.time()
lr = LogisticRegression(C=0.001, solver='liblinear').fit(xtrain, ytrain)
t_lr = time.time()-t0
print('Time taken :' , t_lr)
yhat = lr.predict(xtest)
jaccard_lr = jaccard_score(ytest,yhat)
c_lr = classification_report(ytest,yhat)
prec_lr = precision_score(ytest, yhat)
rec_lr = recall_score(ytest, yhat)
print('Jaccard :',jaccard_lr,'\n',
     c_lr)
```

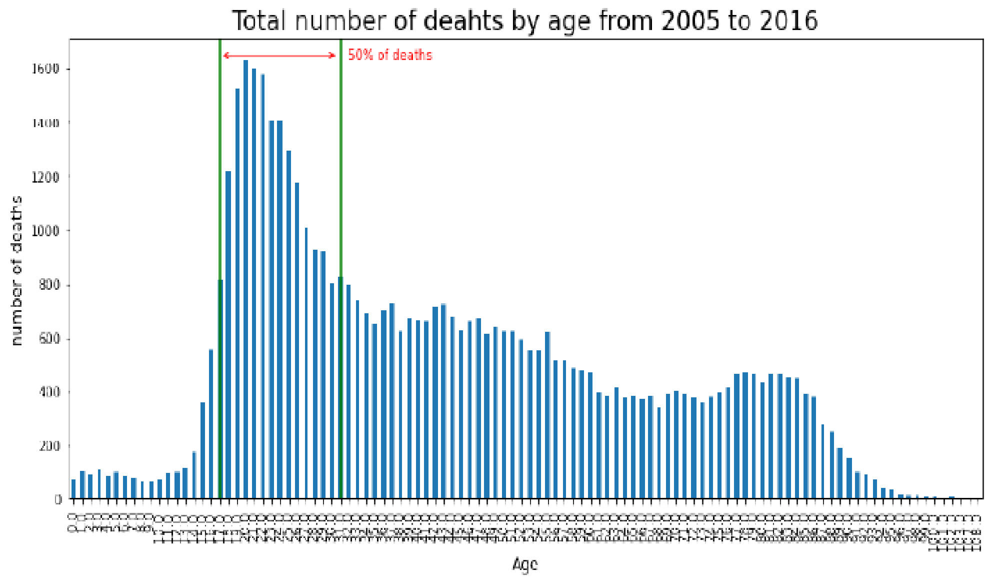
```
tt = xtrain.shape[0]
```

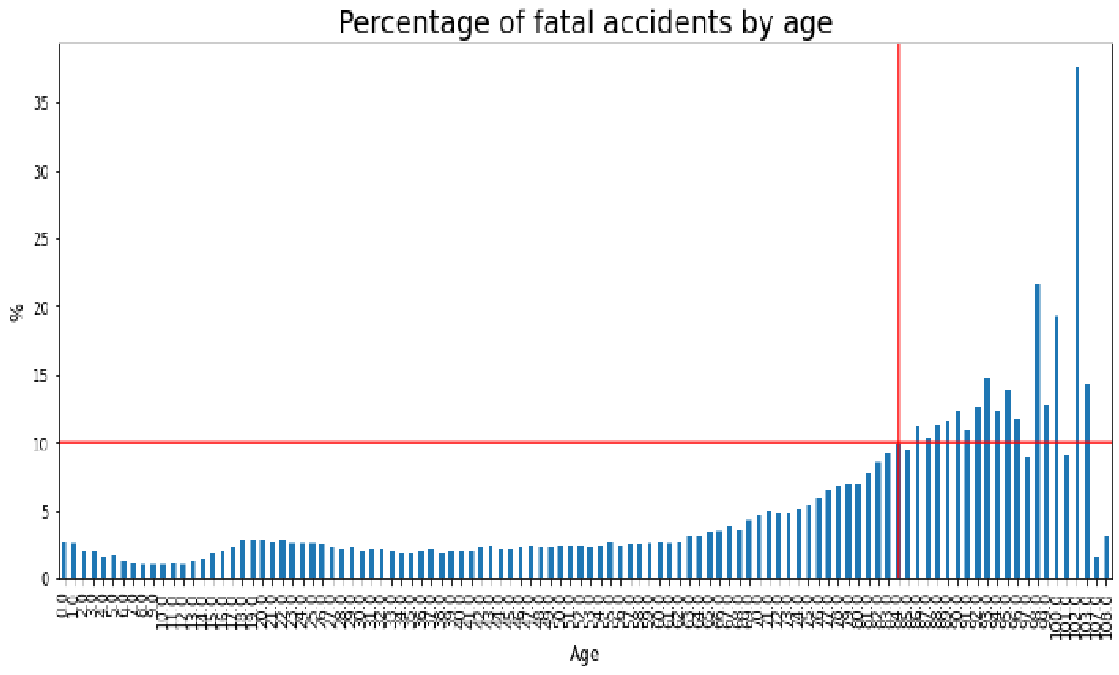
```
tv = xval.shape[0]
xtrain[int(tt*0.5):].shape[0], xval[int(tv*0.5):].shape[0]
ks = 17
mean_acc = np.zeros(ks-1)
std_acc = np.zeros(ks-1)
for n in range(1,ks):
   neigh = KNeighborsClassifier(n_neighbors =
n).fit(xtrain[int(tt*0.5):],ytrain[int(tt*0.5):])
   yhat = neigh.predict(xval[int(tv*0.5):])
   mean_acc[n-1] = accuracy_score(yval[int(tv*0.5):],yhat)
   std_acc[n-1] = np.std(yhat==yval[int(tv*0.5):])/np.sqrt(yhat.shape[0])
print('Best performing K is '+ str(mean_acc.argmax()+1) + ' with an
accuracy of ' +str(mean_acc.max()))
plt.plot(range(1,ks),mean_acc,'g')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.title('Evaluation of accuracy incrementing the value of K')
plt.fill_between(range(1,ks),mean_acc-1*std_acc,mean_acc+1*std_acc,
alpha=0.1)
t0=time.time()
model_knn = KNeighborsClassifier(n_neighbors = 16, n_jobs=-1)
model_knn.fit(xtrain,ytrain)
t_knn = time.time()-t0
print('Time taken :' , t_knn)
yhat = model_knn.predict(xtest)
jaccard_knn = jaccard_score(ytest,yhat)
c_knn = classification_report(ytest,yhat)
prec_knn = precision_score(ytest, yhat)
rec_knn = recall_score(ytest, yhat)
print('Jaccard :',jaccard_knn,'\n',
    c_knn)
size =
[1000,5000,10000,12000,15000,20000,30000,40000,50000,75000,10000
0]
acc = []
t = []
for s in size:
   t0=time.time()
   sv = SVC().fit(xtrain[:s],ytrain[:s])
   t.append(time.time()-t0)
   yhat = sv.predict(xval[:s])
   acc.append(jaccard_score(yval[:s],yhat))
```

```
performance = pd.DataFrame({'acc':acc, 'time':t}, index=size)
performance
performance.plot(y='acc', color='r', label='accuracy')
plt.xlabel('Samples')
plt.ylabel('Evaluation accuracy')
plt.title('Accuracy')
performance.plot(y='time', color='green', label='time')
plt.xlabel('Samples')
plt.ylabel('Computational Time ')
plt.title('Time(s)')
s=75000
t0=time.time()
sv = SVC().fit(xtrain[:s],ytrain[:s])
t_svm = time.time()-t0
print('Time taken :' , t_svm)
yhat = sv.predict(xtest[:s])
jaccard_svm = jaccard_score(ytest[:s],yhat)
c_svm = classification_report(ytest[:s],yhat)
prec_svm = precision_score(ytest[:s], yhat)
rec_svm = recall_score(ytest[:s], yhat)
print('Jaccard :',jaccard_lr,'\n',
    c_lr)
print('Jaccard:',jaccard_rf,'Precision:',prec_rf,'Recall:',rec_rf)
print('Jaccard:',jaccard_lr,'Precision:',prec_lr,'Recall:',rec_lr)
print('Jaccard:',jaccard_knn,'Precision:',prec_knn,'Recall:',rec_knn)
print('Jaccard:',jaccard_svm,'Precision:',prec_svm,'Recall:',rec_svm)
yscores = model_rf.predict_proba(xtest)
false_positive_rate,    true_positive_rate,    thresholds    =
roc_curve(ytest.values, yscores[:,1])
def plot_roc_curve(false_positive_rate, true_positive_rate, label=None):
   plt.plot(false_positive_rate, true_positive_rate, linewidth=2, label='a')
   plt.plot([0, 1], [0, 1], 'r', linewidth=4)
   plt.axis([0, 1, 0, 1])
   plt.xlabel('False Positive Rate (FPR)', fontsize=16)
   plt.ylabel('True Positive Rate (TPR)', fontsize=16)
plt.figure(figsize=(10, 10))
plt.title('ROC curve', fontsize=20)
plot_roc_curve(false_positive_rate, true_positive_rate)
plt.show()
```

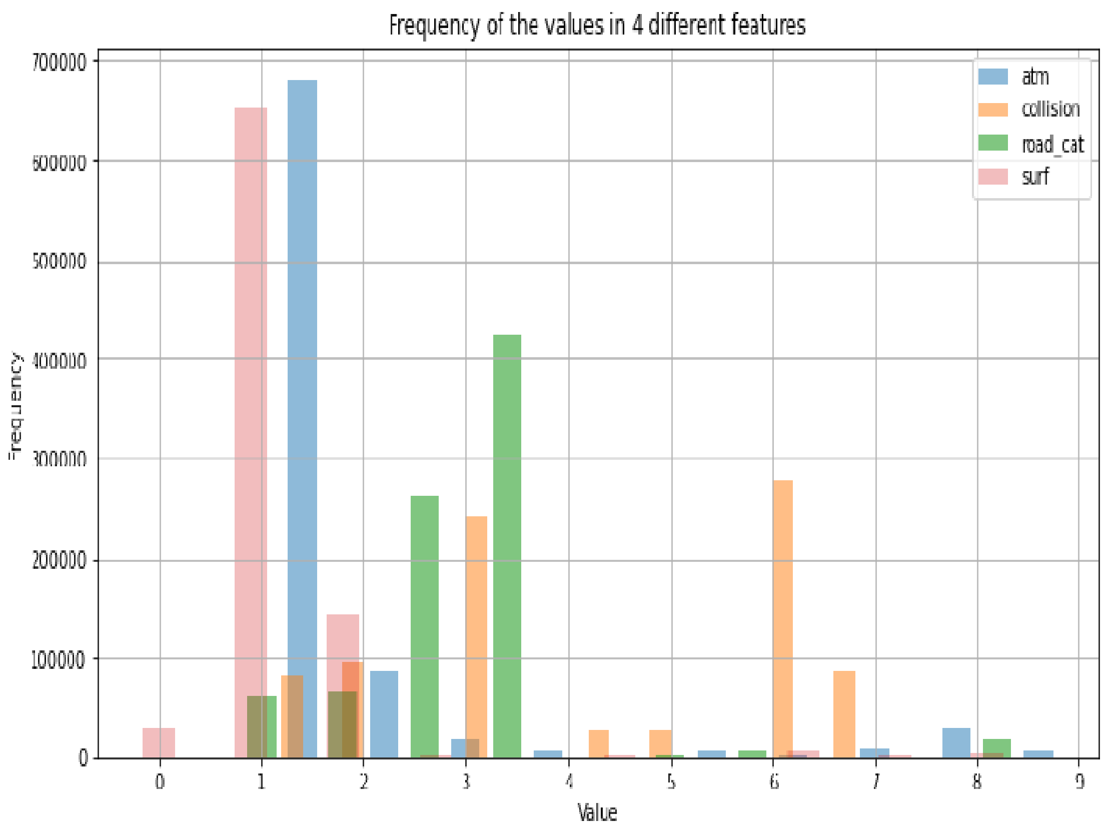
8.2 Screenshots

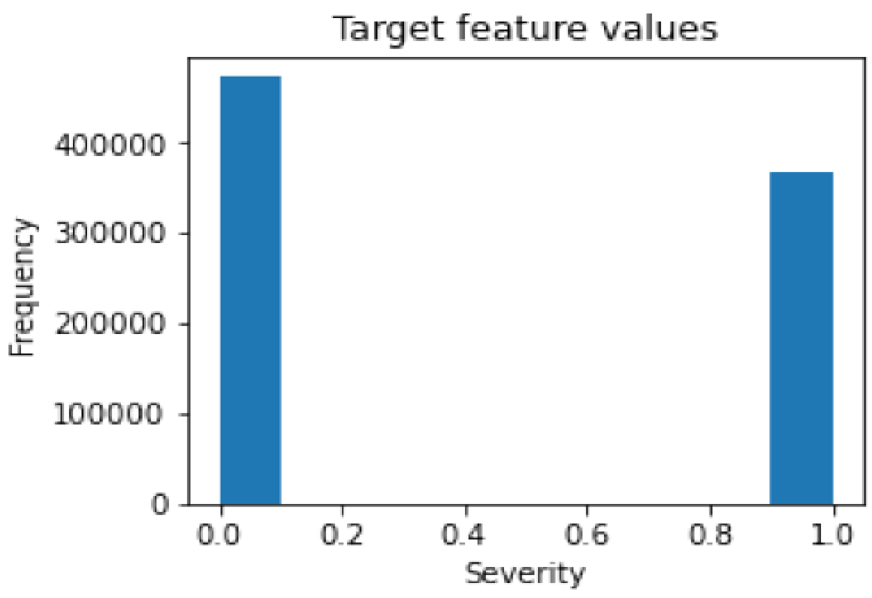
feature selection

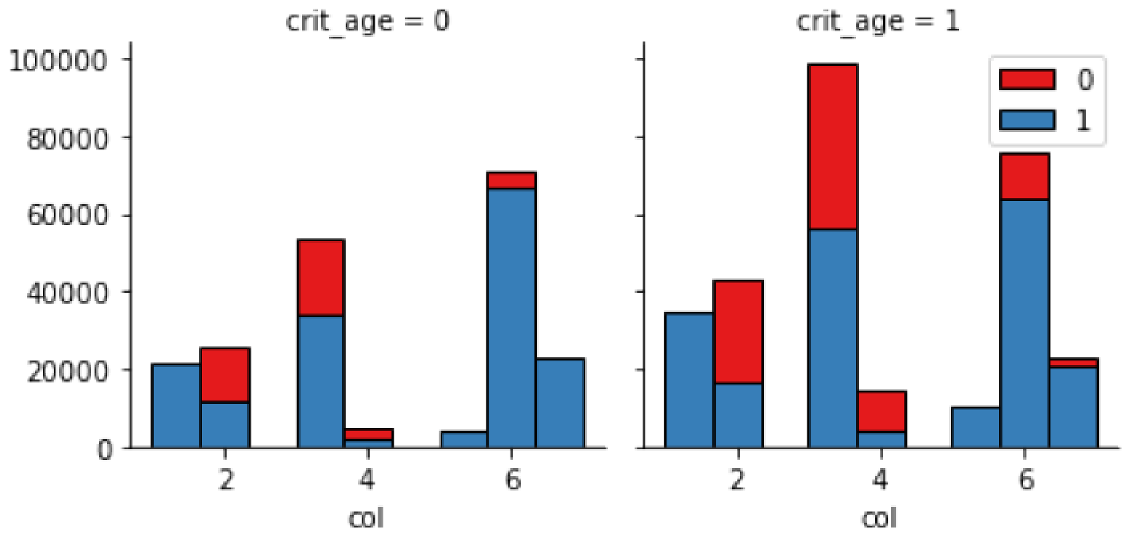


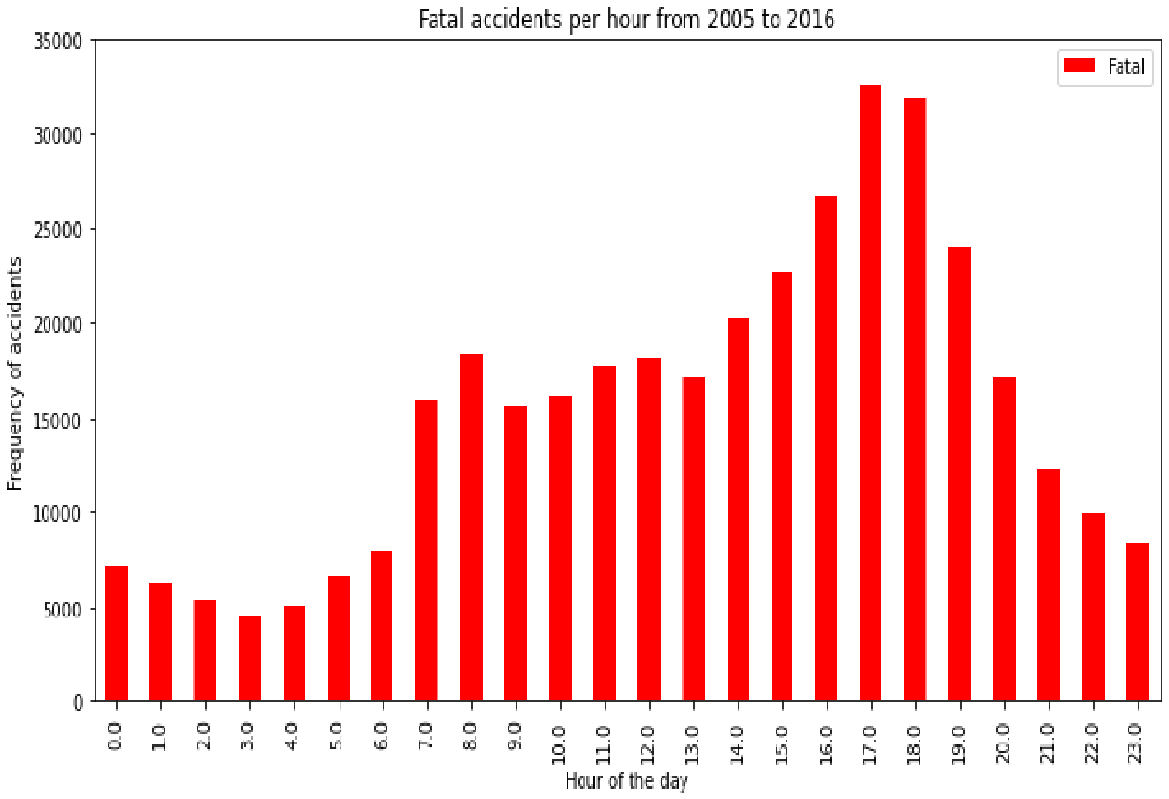


Main notebook






crit_age = 0
crit_age = 1
100000
80000
60000
40000
20000
0
2
4
6
col
0
1


Fatal accidents per hour from 2005 to 2016
Fatal
Frequency of accidents
Hour of the day
35000
30000
25000
20000
15000
10000
5000
0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0

9. References

1. https://www.nrso.ntua.gr/accident-prediction-modelling-a-literature-review-2017/
2. https://www.icevirtuallibrary.com/doi/10.1680/jtran.16.00067
3. https://link.springer.com/article/10.1007/s42452-020-3125-1
4. https://www.nrso.ntua.gr/accident-prediction-modelling-a-literature-review-2017/
5. https://www.researchgate.net/publication/316808843_Road_traffic_accident_prediction_modelling_a_literature_review

10. GOOGLE DRIVE LINK

https://drive.google.com/file/d/1QLeTP2rjek0maWlx4IcFfe
U1TL5ZpO_1/view?usp=sharing