IBM Data Science Capstone Project: Predicting Traffic Accident Severity

By Kassem Saleh
October 10, 2020

Table of content

1. Problem description

- Car accidents and the prediction of their severities given historical data on car accidents collected over many years.
- Predicting the severity of a car accident will help dispatching the appropriate emergency services to the accident scene.
  - Ultimately, this will make an effective use of resources and help save more lives.
- The exploration of collected data can help pinpointing other problems that can be identified and addressed.
  - Road quality and conditions can be improved again leading to the well being and safety of the population.

- The proposed solution can be used by various government departments and authorities to deal in an efficient and effective way with the accident itself.
- Improvements and new policies and guidelines can be developed based on the analysis of the collected data.
  - Departments and authorities that can be involved and can benefit from this system include the police department, the fire department, hospitals, civil defense, traffic and transport department and public works.

2. Data description

- Car accident data that can be useful for predicting the severity of accidents and for developing policies and making decisions that can improve the safety conditions and make efficient use of financial and other resources
- Data should include for each reported accident, among other things: the time (hour, minute) of the accident, the day of the accident, the location of the accident and its severity, the number of people involved, the road conditions and the weather condition during the accident.
- Once an accident is reported, this system would predict the severity of the reported accident based on the learning from the historical data. Based on the prediction, appropriate actions can be performed.
- The data on car accidents that we have used in this project was obtained from kaggle (**https://www.kaggle.com/ahmedlahlou/accidents-in-france-from-2005-to-2016**). It consists of five different excel sheets recording information collected about car accidents in France from 2005 to 2016.

2. Data description

The five files that we have downloaded from Kaggle.com are:

1. **characteristics.csv** file which contains data related to the characteristics of each accident recorded. The file contains 16 columns and 839985 rows. The columns include the accident number, date and time of the accident, the address, the lighting condition and other information.
2. **places.csv** file which contains data related to the places where each accident occured. The file contains 18 columns and 839985 rows. The columns include the accident number, the lane and other information.
3. **users.csv** file which contains data related to the people involved in each accident recorded. The file contains 12 columns and 1876005 rows. The columns include the accident number, the year of birth, the gravity of the injury, the gender and other information.
4. **vehicles.csv** file which contains data related to the vehicles involved in each accident recorded. The file contains 9 columns and 1433389 rows. The columns include the accident number, the type of vehicle and other information about the vehicles involved.
5. **holidays.csv** file which contains data related to the dates of holidays in France during 2005-2016. The file contains 2 columns and 132 rows. The columns include the date of the holiday and the holiday name.

```
# First the characteristics file
dataframe_characteristics = pd.read_csv('caracteristics.csv', encoding = 'latin-1', low_memory = False)
dataframe_characteristics.head(5)
```

	Num_Acc	an	mois	jour	hrmn	lum	agg	int	atm	col	com	adr	gps	lat	long	dep
0	201600000001	16	2	1	1445	1	2	1	8.0	3.0	5.0	46, rue Sonneville	M	0.0	0	590
1	201600000002	16	3	16	1800	1	2	6	1.0	6.0	5.0	1a rue du cimetière	M	0.0	0	590
2	201600000003	16	7	13	1900	1	1	1	1.0	6.0	11.0	NaN	M	0.0	0	590
3	201600000004	16	8	15	1930	2	2	1	7.0	3.0	477.0	52 rue victor hugo	M	0.0	0	590
4	201600000005	16	12	23	1100	1	2	3	1.0	3.0	11.0	rue Joliot curie	M	0.0	0	590

```
# column names in the data frame for characteristics
dataframe_characteristics.columns
```

```
Index(['Num_Acc', 'an', 'mois', 'jour', 'hrmn', 'lum', 'agg', 'int', 'atm',
       'col', 'com', 'adr', 'gps', 'lat', 'long', 'dep'],
      dtype='object')
```

```
# number of rows and columns in the data frame for characteristics
dataframe_characteristics.shape
```

```
(839985, 16)
```

```
# Second -  the places file
dataframe_places = pd.read_csv('places.csv', encoding = 'latin-1', low_memory = False)
dataframe_places.head(5)
```

	Num_Acc	catr	voie	v1	v2	circ	nbv	pr	pr1	vosp	prof	plan	lartpc	larrout	surf	infra	situ	env1
0	201600000001	3.0	39	NaN	NaN	2.0	0.0	NaN	NaN	0.0	1.0	3.0	0.0	0.0	1.0	0.0	1.0	0.0
1	201600000002	3.0	39	NaN	NaN	1.0	0.0	NaN	NaN	0.0	1.0	2.0	0.0	58.0	1.0	0.0	1.0	0.0
2	201600000003	3.0	1	NaN	NaN	2.0	2.0	NaN	NaN	0.0	1.0	3.0	0.0	68.0	2.0	0.0	3.0	99.0
3	201600000004	4.0	0	NaN	NaN	2.0	0.0	NaN	NaN	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	99.0
4	201600000005	4.0	0	NaN	NaN	0.0	0.0	NaN	NaN	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	3.0

```
# column names in the data frame for places
dataframe_places.columns
```

```
Index(['Num_Acc', 'catr', 'voie', 'v1', 'v2', 'circ', 'nbv', 'pr', 'pr1',
       'vosp', 'prof', 'plan', 'lartpc', 'larrout', 'surf', 'infra', 'situ',
       'env1'],
      dtype='object')
```

```
# number of rows and columns in the data frame for places
dataframe_places.shape
```

```
(839985, 18)
```

```
# Third -  the users file
dataframe_users = pd.read_csv('users.csv', encoding = 'latin-1', low_memory = False)
dataframe_users.head(5)
```

	Num_Acc	place	catu	grav	sexe	trajet	secu	locp	actp	etatp	an_nais	num_veh
0	201600000001	1.0	1	1	2	0.0	11.0	0.0	0.0	0.0	1983.0	B02
1	201600000001	1.0	1	3	1	9.0	21.0	0.0	0.0	0.0	2001.0	A01
2	201600000002	1.0	1	3	1	5.0	11.0	0.0	0.0	0.0	1960.0	A01
3	201600000002	2.0	2	3	1	0.0	11.0	0.0	0.0	0.0	2000.0	A01
4	201600000002	3.0	2	3	2	0.0	11.0	0.0	0.0	0.0	1962.0	A01

```
# column names in the data frame for users
dataframe_users.columns
```

```
Index(['Num_Acc', 'place', 'catu', 'grav', 'sexe', 'trajet', 'secu', 'locp',
       'actp', 'etatp', 'an_nais', 'num_veh'],
      dtype='object')
```

```
# number of rows and columns in the data frame for users
dataframe_users.shape
```

```
(1876005, 12)
```

```python
# Fourth -  the vehicles file
dataframe_vehicles = pd.read_csv('vehicles.csv', encoding = 'latin-1', low_memory = False)
dataframe_vehicles.head(5)
```

	Num_Acc	senc	catv	occutc	obs	obsm	choc	manv	num_veh
0	201600000001	0.0	7	0	0.0	0.0	1.0	1.0	B02
1	201600000001	0.0	2	0	0.0	0.0	7.0	15.0	A01
2	201600000002	0.0	7	0	6.0	0.0	1.0	1.0	A01
3	201600000003	0.0	7	0	0.0	1.0	6.0	1.0	A01
4	201600000004	0.0	32	0	0.0	0.0	1.0	1.0	B02

```python
# column names in the data frame for vehicles
dataframe_vehicles.columns
```

```
Index(['Num_Acc', 'senc', 'catv', 'occutc', 'obs', 'obsm', 'choc', 'manv',
       'num_veh'],
      dtype='object')
```

```python
# number of rows and columns in the data frame for vehicles
dataframe_vehicles.shape
```

```
(1433389, 9)
```

```python
# Fifth -  the holidays file
dataframe_holidays = pd.read_csv('holidays.csv', encoding = 'latin-1', low_memory = False)
dataframe_holidays.head(5)
```

	ds	holiday
0	2005-01-01	New year
1	2005-03-28	Easter Monday
2	2005-05-01	Labour Day
3	2005-05-05	Ascension Thursday
4	2005-05-08	Victory in Europe Day

```python
# column names in the data frame for holidays
dataframe_holidays.columns
```

```
Index(['ds', 'holiday'], dtype='object')
```

```python
# number of rows and columns in the data frame for holidays
dataframe_holidays.shape
```

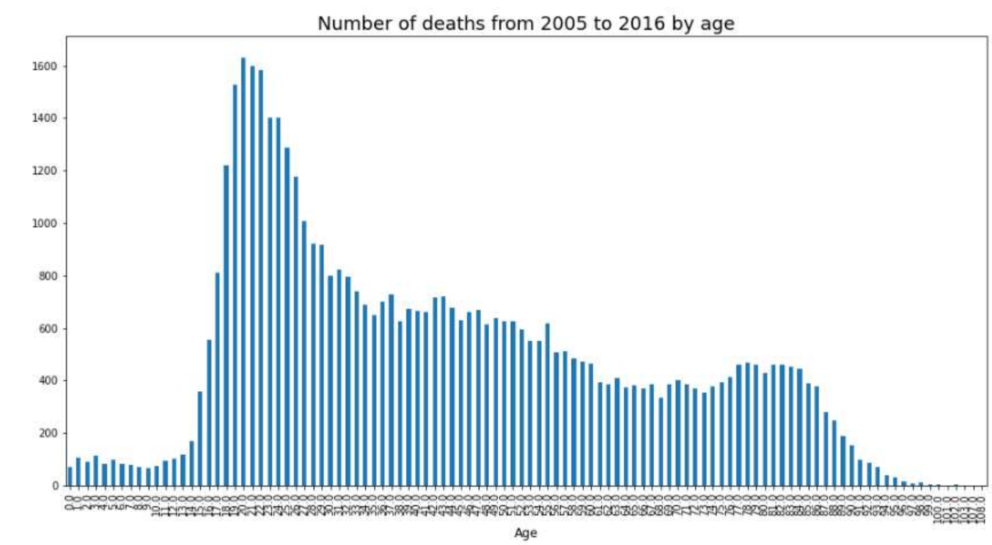
```
(132, 2)
```

3. Data cleaning

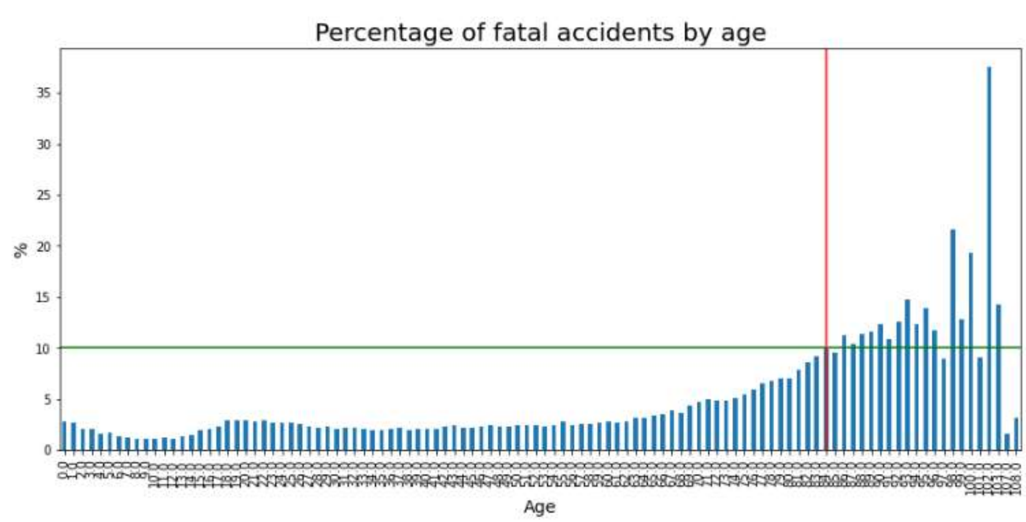
- After checking the data file contents, we can see there are a lot of data that are either not available or seem abnormal.
- Also many features are not needed or useful for the analysis and can be dropped from the data sets.
- After cleaning the data from the five data files, we joined them in one single data frame.

4. Exploratory data analysis (EDA)

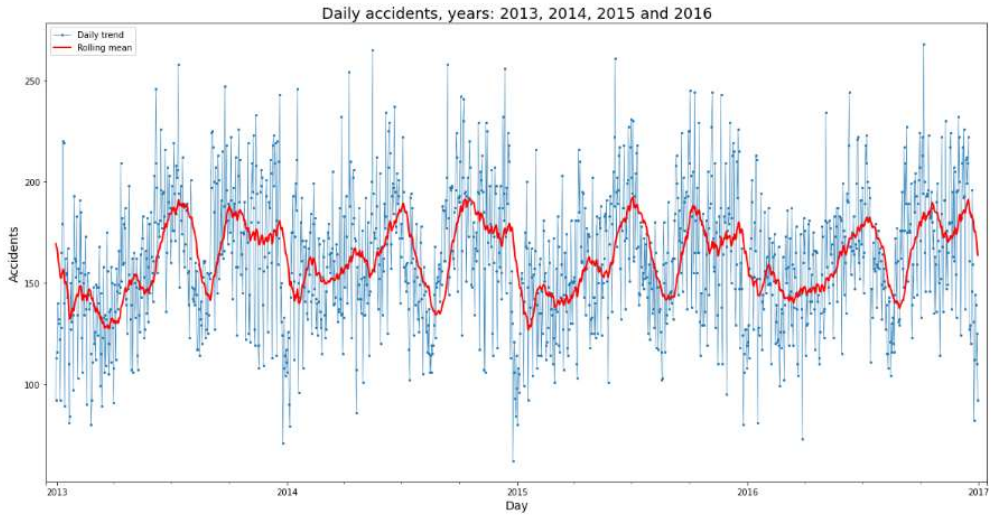
- Number of deaths by age
- Percentage of fatal accidents by age
- Number of daily accidents
- Number of accidents per year
- Distribution of accidents over the 12 months
- Distribution of accidents over the days of the week
- Distribution of accidents over the hours of the day

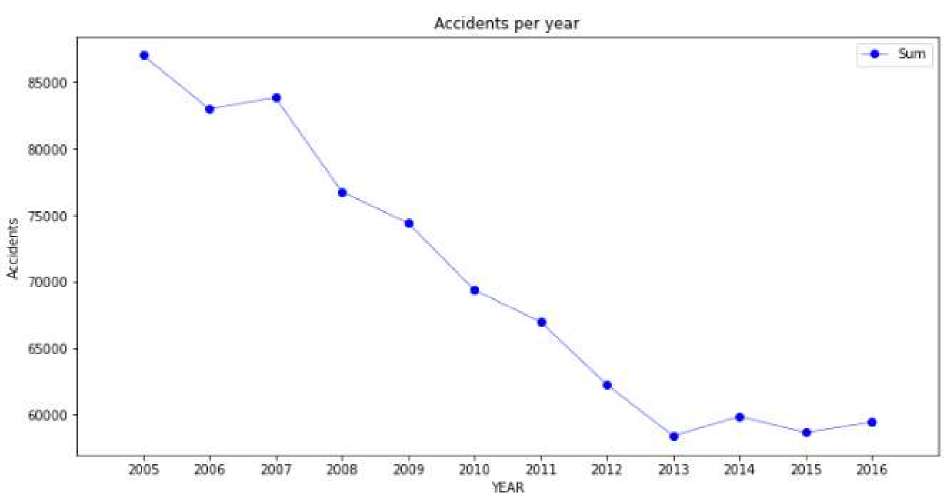


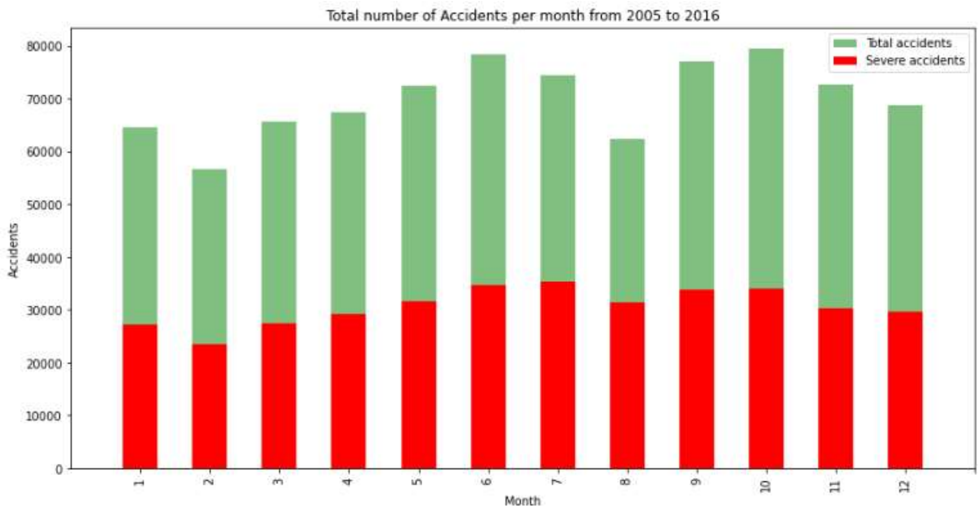
Number of deaths from 2005 to 2016 by age
1600
1400
1200
1000
800
600
400
200
0
Age

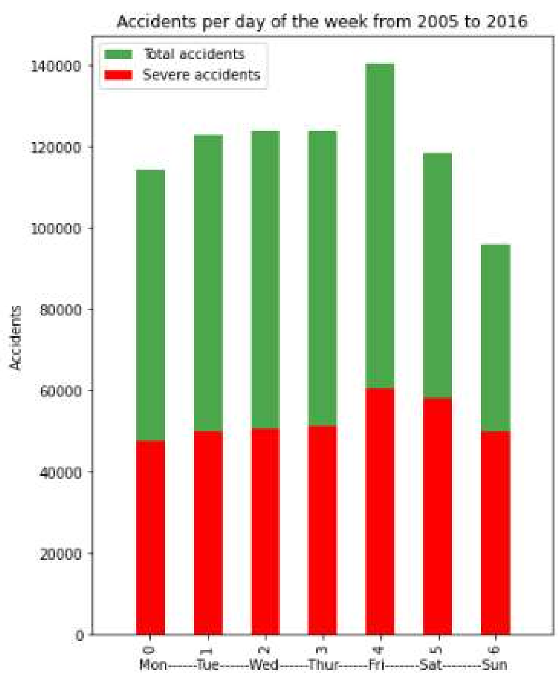


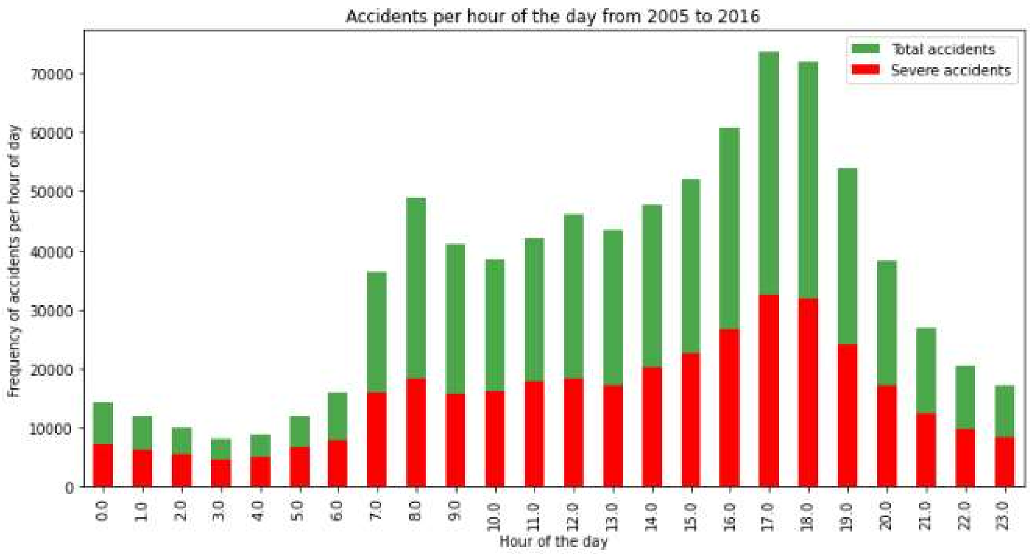
Percentage of fatal accidents by age
35
30
25
20
15
10
5
0
%
Age


Daily accidents, years: 2013, 2014, 2015 and 2016
Daily trend
Rolling mean
Accidents
250
200
150
100
2013
2014
2015
2016
2017
Day


Accidents per year
Sum
85000
80000
75000
70000
65000
60000
Accidents
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
YEAR


Total number of Accidents per month from 2005 to 2016
Total accidents
Severe accidents
Accidents
Month
0
10000
20000
30000
40000
50000
60000
70000
80000
1
2
3
4
5
6
7
8
9
10
11
12


Accidents per day of the week from 2005 to 2016
Total accidents
Severe accidents
Accidents
0
20000
40000
60000
80000
100000
120000
140000
0
1
2
3
4
5
6
Mon------Tue------Wed------Thur------Fri-------Sat--------Sun



5. Data preparation

```
#Some feature's values range from 1 to 9 while others just go either for 1 or 2,
#Normalizing the data makes that any feature has more influence in the result than others.

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(dataframe.drop('SEVERITY', axis=1), dataframe['SEVERITY'], test_size=0.2, random
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])
```

```
Size of training set: 537590
Size of test set: 167997
Size of evaluation set: 134398
```

6. Model development

We will use the following machine learning techniques and analyze their precisions.

1. Decision Tree (DT) model
2. Logistic Regression (LR) model
3. K-Nearest Neighbors (KNN) model
4. Support Vector Machine (SVM) model

6.1. Decision Tree model

- Decision tree classifier
- Random forest classifier
  - 100 decision trees
  - 50 decision trees
  - 10 decision trees

```
t0=time.time()
decisionTree = DecisionTreeClassifier(criterion='entropy')
decisionTree.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = decisionTree.predict(xval)
score_tree = accuracy_score(yval, yhat)
print('Accuracy :',score_tree)
```

```
Time taken : 11.797379970550537
Accuracy : 0.6360511317132695
```

```
#number of decision trees reduced from 50 to 10
#Limiting the number of features to look at when creating the next split to 8
#Limiting the max depth of the tree to 12

t0=time.time()
model_randomForest = RandomForestClassifier(n_estimators=10, max_features=8, max_depth =12,criterion='entropy',random_state=0, n
model_randomForest.fit(xtrain,ytrain)
print('Time taken :' , time.time()-t0)
yhat = model_randomForest.predict(xval)
score_randomForest = accuracy_score(yval,yhat)
print('Accuracy :',score_randomForest)
```

```
Time taken : 8.156516551971436
Accuracy : 0.7214988318278546
```

Dropping 10 least important features

```
({'Feature':dataframe.drop('SEVERITY', axis=1).columns,'Feature importance':np.round(model_randomForest.feature_importances_,3)})
Features.sort_values('Feature importance',ascending=False).set_index('Feature importance')
```

Feature importance	Feature
0.162	DEP
0.146	DAY
0.116	TIME
0.105	MONTH
0.053	RD_CATG
0.046	COL
0.045	num_users
0.035	AGG
0.035	N_LANES
0.032	INT
0.029	TRAFC_REGS
0.025	ATM
0.024	LIGHT
0.022	LONGIT
0.022	SURFACE
0.021	critical_age
0.019	SHAPE
0.018	SCHOOL
0.016	WEEKEND
0.014	SITUATION
0.008	pedestrian
0.005	HOLIDAY
0.004	death_age

```
xtrain = pd.DataFrame(xtrain)
xtrain.drop(['death_age', 'HOLIDAY', 'pedestrian', 'SITUATION', 'WEEKEND', 'SCHOOL', 'SHAPE', 'critical_age', 'SURFACE', 'LONGIT
xval.drop(['death_age', 'HOLIDAY', 'pedestrian', 'SITUATION', 'WEEKEND', 'SCHOOL', 'SHAPE', 'critical_age', 'SURFACE', 'LONGIT',
xtest.drop(['death_age', 'HOLIDAY', 'pedestrian', 'SITUATION', 'WEEKEND', 'SCHOOL', 'SHAPE', 'critical_age','SURFACE', 'LONGIT',
```

```
#RF 2:
#number of features reduced from 23 to 13

t0=time.time()
model_randomForest = RandomForestClassifier(n_estimators=100,criterion='entropy',random_state=0, n_jobs=-1)
model_randomForest.fit(xtrain,ytrain)
print('Time taken :' , time.time() - t0)
yhat = model_randomForest.predict(xval)
score_randomForest = accuracy_score(yval,yhat)
print('Accuracy :',score_randomForest)
```

```
Time taken : 55.78447604179382
Accuracy : 0.7064316433280257
```

Model evaluation

We can see that after reducing the number of estimators and features, we have obtained almost similar accuracy and reduced the processing time.

	# of estimators	Max # of features	Time taken	Accuracy
Random Forest 1	100	23	58.51	0.71
Random Forest 2	100	13	55.78	0.70
Random Forest 3	50	5	23.04	0.71
Random Forest 4	10	8	8.15	0.72

6.2. Logistic Regression (LR) model

- Finding the best model regularization coefficient (c= 0.001)

```
accuracy = np.zeros(6)
i=0
for c in [0.5, 0.1, 0.01, 0.001, 10, 100]:
    logisticRegression = LogisticRegression(C=c, solver='liblinear').fit(xtrain, ytrain)
    yhat = logisticRegression.predict(xval)
    accuracy[i] = accuracy_score(yval,yhat)
    i+=1
accuracy
```

```
array([0.65830593, 0.65831337, 0.65857379, 0.65904255, 0.65832825,
       0.65832825])
```

- Building the logistic regression model and evaluate it

```
t0=time.time()
logisticRegression = LogisticRegression(C=0.001, solver='liblinear').fit(xtrain, ytrain)
t_logisticRegression = time.time()-t0
print('Time taken :' , t_logisticRegression)
yhat = logisticRegression.predict(xtest)
jaccard_logisticRegression = jaccard_score(ytest,yhat)
c_logisticRegression = classification_report(ytest,yhat)
prec_logisticRegression = precision_score(ytest, yhat)
rec_logisticRegression = recall_score(ytest, yhat)
print('Jaccard :',jaccard_logisticRegression,'\n',   c_logisticRegression)
```

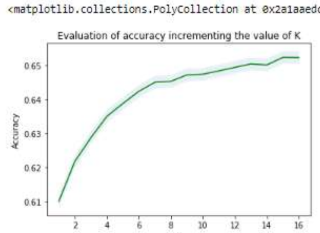
```
Time taken : 4.526005029678345
Jaccard : 0.3717329026977403
              precision    recall  f1-score   support

           0       0.66      0.82      0.73     94297
           1       0.67      0.46      0.54     73700

    accuracy                           0.66    167997
   macro avg       0.66      0.64      0.64    167997
weighted avg       0.66      0.66      0.65    167997
```

6.3. K-Nearest Neighbors (KNN) model

- Find the best performing K

```
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_accuracy = np.zeros(ks-1)
std_accuracy = 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_accuracy[n-1] = accuracy_score(yval[int(tv*0.5):],yhat)
    std_accuracy[n-1] = np.std(yhat==yval[int(tv*0.5):])/np.sqrt(yhat.shape[0])
print('Best performing K is '+ str(mean_accuracy.argmax()+1) + ' with an accuracy of ' +str(mean_accuracy.max()))

plt.plot(range(1,ks),mean_accuracy,'g')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.title('Evaluation of accuracy incrementing the value of K')
plt.fill_between(range(1,ks), mean_accuracy -1 * std_accuracy , mean_accuracy + 1 * std_accuracy, alpha=0.1)
```

```
Best performing K is 15 with an accuracy of 0.6523906605753061

<matplotlib.collections.PolyCollection at 0x2a1aaedcac0>
```



- Best at K = 15

KNN model and evaluation at k=16

```
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)
```

```
Time taken : 189.3194124698639
Jaccard : 0.39912135367118107
              precision    recall  f1-score   support

           0       0.67      0.79      0.73     94297
           1       0.65      0.51      0.57     73700

    accuracy                           0.67    167997
   macro avg       0.66      0.65      0.65    167997
weighted avg       0.66      0.67      0.66    167997
```

6.4. Support Vector Machine (SVM) model

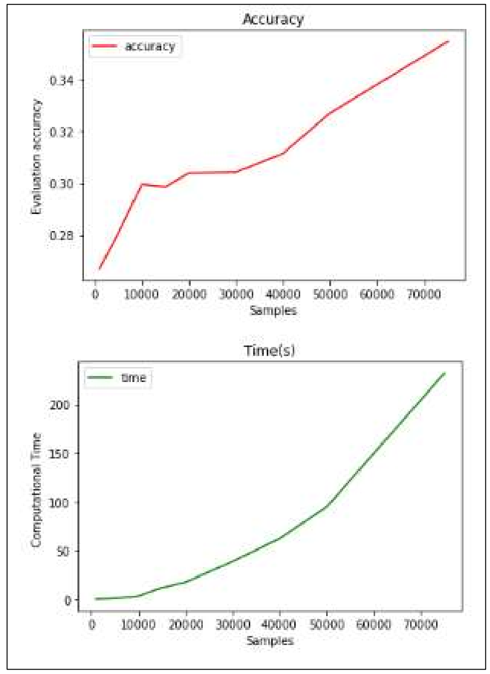
- Accuracy for 9 different sample sizes

```
size = [1000,5000,10000,15000,20000,30000,40000,50000,75000]
accuracy = []
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])
    accuracy.append(jaccard_score(yval[:s],yhat))

performance = pd.DataFrame({'acc':accuracy, 'time':t}, index=size)
performance
```

- Model evaluation

```
Time taken : 238.07018876075745
Jaccard : 0.3717329026977403
              precision    recall  f1-score   support

           0       0.66      0.82      0.73     94297
           1       0.67      0.46      0.54     73700

    accuracy                           0.66    167997
   macro avg       0.66      0.64      0.64    167997
weighted avg       0.66      0.66      0.65    167997
```

Plotting accuracy and execution time per samples



7. Results and observations

Table summarizing the 4 models

Algorithm	Jaccard	f1-score	Precision	Recall
Random Forest	0.488	0.72	0.722	0.601
Logistic Regression	0.371	0.667	0.667	0.456
KNN	0.399	0.652	0.652	0.506
SVM	0.358	0.688	0.660	0.427

Observations:

- Precision means the percentage of predicted severe accidents that were truly severe.
- Recall is the percentage of truly severe accidents that were correctly predicted.
- Recall is more important than the precision as a high recall means that all required resources will be equipped up to the severity of the accident.
- LR, KNN and SVM models have similar accuracy.
- RF model is the best model since it has the highest accuracy and highest recall.
- The best model with the highest precision is the Random Forest Decision Tree model with an precision of 0.72 and a recall of 0.59