Help the Government of India's Transport Department to predict accident severity!

Introduction | Business Understanding

The government is going to prevent avoidable car accidents by employing methods that alert drivers, health system, and police to remind them to be more careful in critical situations.

In most cases, not paying enough attention during driving, abusing drugs and alcohol or driving at very high speed are the main causes of occurring accidents that can be prevented by enacting harsher regulations. Besides the aforementioned reasons, weather, visibility, or road conditions are the major uncontrollable factors that can be prevented by revealing hidden patterns in the data and announcing warning to the local government, police and drivers on the targeted roads.

The target audience of the project is local government, police, rescue groups, and last but not least, car insurance institutes. The model and its results are going to provide some advice for the target audience to make insightful decisions for reducing the number of accidents and injuries for the city.

Data Understanding

The data was collected by https://www.kaggle.com/c/accidentseverity. The data consists of 17 independent variables. The dependent variable, “Collision_Severity”, contains numbers that correspond to different levels of severity caused by an accident from 1 to 3. The evaluation is the accuracy metric.

Collision Severity:

1. Fatal injury collision
2. Serious injury collision
3. Slight injury collision

File descriptions

- Accident_train_head.csv - the head (first 5 records) of the training set
- Accident_train.csv - the train set
- Accident_test.csv - the test set
- AttributeLevelsDescription.csv - Detailed description about the attributes the data

Data fields

- Collision_Ref_No - a Collision Reference Number
- Policing_Area - the Policing area
- Collision_Severity - the Collision Severity
- etc...

```
data_train = pd.read_csv("./dataset/Accident_train.csv")
data_test = pd.read_csv("./dataset/Accident_test.csv")
```

Furthermore, because of the existence of null values in some records, the data needs to be pre-processed before any further processing.

Data Pre-Processing

The dataset in the original form is not ready for data analysis. In order to prepare the data, first, we need to drop the non-relevant columns. In addition, some of the features are of object data types that need to be converted into numerical data types.

```
In [19]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
         # lable encoder for data_train
         data_train['Policing_Area'] = le.fit_transform(data_train['Policing_Area'].astype(str))

In [20]: data_train.replace(['MON','TUE','WED','THU','FRI','SAT','SUN'], [1,2,3,4,5,6,7], inplace= True)
```

After analysing the data set, I have decided to focus on only four features, severity, weather conditions, road conditions, and light conditions, among others.

To get a good understanding of the dataset, I have checked different values in the features.

Methodology

For implementing the solution, I have used GitHub as a repository and running Jupyter Notebook to pre-process data and build Machine Learning models. Regarding coding, I have used Python and its popular packages such as Pandas, Numpy, Matplotlib and Sklearn.

Once I have load data into Pandas Dataframe, used '*dtypes'* attribute to check the feature names and their data types. I also checked the null values in the dataset using '*info*'. Then I have selected the most important features to predict the severity of accidents.

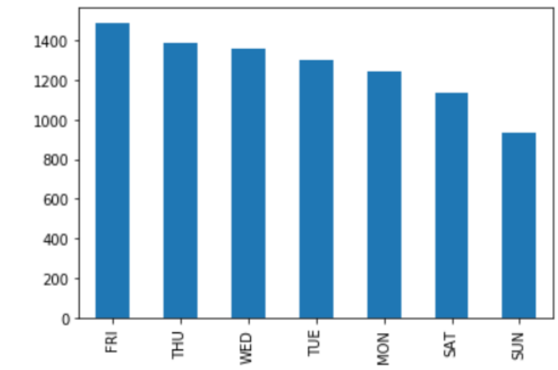
```
Data columns (total 17 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Collision_Ref_No              8849 non-null   int64  
 1   Policing_Area                 8562 non-null   object 
 2   Collision_Severity            8849 non-null   int64  
 3   Weekday_of_Collision          8849 non-null   object 
 4   Day_of_Collision              8849 non-null   int64  
 5   Month_of_Collision            8849 non-null   int64  
 6   Hour_of_Collision             8601 non-null   float64
 7   Carriageway_Type              8849 non-null   int64  
 8   Speed_Limit                   8849 non-null   int64  
 9   Junction_Detail               8593 non-null   float64
 10  Junction_Control              8590 non-null   float64
 11  Ped_Crossing_HC               8573 non-null   float64
 12  Ped_Crossing_PC               8584 non-null   float64
 13  Light_Conditions              8849 non-null   int64  
 14  Weather_Conditions            8849 non-null   int64  
 15  Road_Surface_Conditions       8575 non-null   float64
 16  Special_Conditions_at_Site    8600 non-null   float64
dtypes: float64(7), int64(8), object(2)
```

Exploratory Data Analysis

I have run a value count on weekday ('Weekday_of_Collision'), month ('Month_of_Collision') to get ideas of the accident cases pattern. I also have run a value count on light condition ('Light_Conditions'), Weather Condition ('Weather_Conditions'), Road Condition ('Road_Surface_Conditions') to see the breakdowns of accidents occurring during the different light conditions, weather conditions and road conditions.

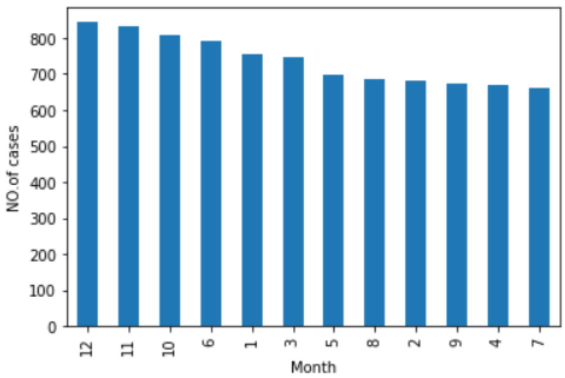
The results can be seen below:

```
In [7]: data_train['Weekday_of_Collision'].value_counts()

Out[7]: FRI    1489
        THU    1388
        WED    1355
        TUE    1302
        MON    1243
        SAT    1138
        SUN     934
        Name: Weekday_of_Collision, dtype: int64
```



As you can see, the accident cases increases as the week passes from Monday to Friday and it’s getting highest on Friday and then gets minimum on weekends. It can be understood that the roads are more busy on the office days and thus more likely to have chances of accidents.

```
In [10]: data_train['Month_of_Collision'].value_counts()

Out[10]: 12    845
         11    832
         10    809
         6     792
         1     754
         3     747
         5     699
         8     686
         2     683
         9     674
         4     668
         7     660
         Name: Month_of_Collision, dtype: int64
```



In winter season, the number of accident cases can be seen higher than rest of the year. It might be possible because of fog, low visibility and the wheather conditions of the season.

```
In [15]: data_train['Light_Conditions'].value_counts()

Out[15]: 1    2276
         2    2042
         3    1223
         7    1129
         5     919
         4     718
         6     542
         Name: Light_Conditions, dtype: int64
```

Light Conditions	1 : Daylight : street lights present
	2 : Daylight : no street lighting
	3 : Daylight : street lighting unknown
	4 : Darkness : street lights present and lit
	5 : Darkness : street lights present but unlit
	6 : Darkness : no street lighting
	7 : Darkness : street lighting unknown

Surprisingly but the maximum accidents happened in daylight. The number of cases in darkness are quite less as compared to that of daylight. This clearly shows that the maximum possibility of accidents are likely to be because of careless extra- relaxed attitude of the drivers when there is low possibility of accidents.

In [16]:
```
data_train['Weather_Conditions'].value_counts()
```

Out[16]:
```
1     2211
10    1695
2     1681
3     1606
9     1590
5       30
4       14
8       12
7        7
6        3
Name: Weather_Conditions, dtype: int64
```

Weather Conditions	1 : Fine without high winds
	2 : Raining without high winds
	3 : Snowing without high winds
	4 : Fine with high winds
	5 : Raining with high winds
	6 : Snowing with high winds
	7 : Fog or mist - if hazard
	8 : Strong sun (glaring)
	9 : Other
	10 : Unknown

In [17]:
```
data_train['Road_Surface_Conditions'].value_counts()
```

Out[17]:
```
1.0     5284
2.0     2760
4.0      141
10.0     105
9.0       87
6.0       85
3.0       50
5.0       28
7.0       26
8.0        9
Name: Road_Surface_Conditions, dtype: int64
```

Road Surface Conditions	1 : Dry
	2 : Wet / damp
	3 : Snow
	4 : Frost / ice
	5 : Flood
	6 : Oil
	7 : Mud
	8 : Leaves
	9 : Slippery (after dry spell)
	10 : Other

Null Values replacement

Here I replaced null values present in the dataset with the most frequent values in the columns i.e. mode value.

In [24]:
```
# Converting Nan Values to mean/mode etc.,
data_train.isna().any()
#data_train.mean()
# Taking care of missing values with most frequent values(mode)
data_train = data_train.apply(lambda x:x.fillna(x.value_counts().index[0]))
```

Train/Test Split

We will use 25% of our data for testing and 75% for training. Also, we standardized the data. **StandardScaler()** normalized the features i.e. each column of X, **INDIVIDUALLY**, so that each column/feature/variable will have

$\mu = 0$ and $\sigma = 1$.

[28]:
```
from sklearn.model_selection import train_test_split
from sklearn import (svm, preprocessing)
# splitting test vales into parts to find get lables
data_variables = train_test_split(data_train_x, labels_train, test_size = 0.25, random_state = 42)
data_train_l, data_test_l, labels_train_l, labels_test_l = data_variables

scaler = preprocessing.StandardScaler()
train_data_scaled = scaler.fit_transform(data_train_l)
test_data_scaled = scaler.transform(data_test_l)

test_data_scaled_x = scaler.transform(data_test_x)
```

After importing necessary packages and splitting pre-processed data into test and train sets, for each machine learning model, I have built and evaluated the model and shown the results as follow:

Support Vector Machines

Support Vector Machines

In [29]:
```
import collections
from sklearn.svm import SVC, LinearSVC
#Support Vector Machines
svc = SVC()
svc.fit(train_data_scaled, labels_train_l)
Y_pred = svc.predict(test_data_scaled)
a = np.array(Y_pred)
predict_svc = collections.Counter(a)
acc_svc = round(svc.score(test_data_scaled, labels_test_l) * 100, 2)
acc_svc
```

Out[29]:
```
89.47
```

Linear SVC

Linear SVC

[30]:
```
# Linear SVC
linear_svc = LinearSVC()
linear_svc.fit(train_data_scaled, labels_train_l)
Y_pred = linear_svc.predict(test_data_scaled)
a = np.array(Y_pred)
predict_linear_svc = collections.Counter(a)
acc_linear_svc = round(linear_svc.score(test_data_scaled, labels_test_l) * 100, 2)
acc_linear_svc
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\svm\_base.py:947: ConvergenceWarning:
the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

[30]:
```
89.47
```

Stochastic Gradient Descent

Gradient Descent

[31]:
```
# Stochastic Gradient Descent
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd.fit(train_data_scaled, labels_train_l)
Y_pred = sgd.predict(test_data_scaled)
a = np.array(Y_pred)
predict_sgd = collections.Counter(a)
acc_sgd = round(sgd.score(test_data_scaled, labels_test_l) * 100, 2)
acc_sgd
```

[31]:
```
89.47
```

KNN

K Nearest Neighbors

[32]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(train_data_scaled, labels_train_l)
Y_pred = knn.predict(test_data_scaled)
a = np.array(Y_pred)
predict_knn = collections.Counter(a)
acc_knn = round(knn.score(test_data_scaled, labels_test_l) * 100, 2)
acc_knn
```

[32]:
```
86.94
```

Logistic Regression

Logistic Regression

[33]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(train_data_scaled, labels_train_l)
Y_pred = logreg.predict(test_data_scaled)
a = np.array(Y_pred)
predict_log = collections.Counter(a)
acc_log = round(logreg.score(test_data_scaled, labels_test_l) * 100, 2)
acc_log
```

[33]: 89.47

Gaussian Naive Bayes

Naive Bayes

[34]:
```
#Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
gaussian = GaussianNB()
gaussian.fit(train_data_scaled, labels_train_l)
Y_pred = gaussian.predict(test_data_scaled)
a = np.array(Y_pred)
predict_gaussian = collections.Counter(a)
acc_gaussian = round(gaussian.score(test_data_scaled, labels_test_l) * 100, 2)
acc_gaussian
```

[34]: 9.67

Decision Tree

Decision Tree

[35]:
```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier()
decision_tree.fit(train_data_scaled, labels_train_l)
Y_pred = decision_tree.predict(test_data_scaled)
a = np.array(Y_pred)
predict_decision_tree = collections.Counter(a)
acc_decision_tree = round(decision_tree.score(test_data_scaled, labels_test_l) * 100, 2)
acc_decision_tree
```

[35]: 85.81

Random Forest

Random Forest

[47]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(train_data_scaled, labels_train_l)
Y_pred = random_forest.predict(test_data_scaled)
a = np.array(Y_pred)
predict_random_forest = collections.Counter(a)
acc_random_forest = round(random_forest.score(test_data_scaled, labels_test_l) * 100, 2)
acc_random_forest
```

[47]: 90.06

Results and Evaluations

The final results of the model evaluations are summarized in the following table:

	Model	Score	Count
3	Random Forest	90.06	{3: 2183, 2: 30}
0	Support Vector Machines	89.47	{3: 2213}
2	Logistic Regression	89.47	{3: 2213}
5	Stochastic Gradient Decent	89.47	{3: 2213}
6	Linear SVC	89.47	{3: 2213}
1	KNN	86.94	{3: 2104, 2: 85, 1: 24}
7	Decision Tree	85.27	{3: 1932, 2: 239, 1: 42}
4	Naive Bayes	9.67	{1: 1981, 3: 190, 2: 42}

Based on the above table, Random Forest is the best model to predict car accident severity.

Discussion

In the beginning of this notebook, we had categorical data that was of type 'object'. This is not a data type that we could have fed through an algorithm, so label encoding was used to created new classes that were of type int8; a numerical data type.

Once we analysed and cleaned the data, it was then fed through some ML models. The Random Forest model made most sense for this project.

Evaluation metric used to test the accuracy of our models.

Conclusion

Most crashes happened in clear, dry, and bright conditions. Most days are clear, dry, and bright, so it's no surprise that most car crashes occur under these conditions. I also found out that crashes with a distracted driver or an impaired driver are statistically more likely to result in injury, which is also not a surprise. The results of the data indicate to city officials that they should ask drivers to be more alert in ideal conditions.

Based on the dataset provided for this capstone from weather, road, and light conditions pointing to certain classes, we can conclude that particular conditions have a somewhat impact on whether or not travel could result in property damage or injury.