Data analysis and model building for traffic accident prediction

Introduction

Imagine you are driving on a highway, you might feel lucky that a traffic jam happened on your opposite direction, but, what if the jam happened on your side? Would it be super annoying? According to a survey, traffic jams are mostly caused by accidents. If the severe level of accidents can be determined with involved factors, anticipated warnings or actions can be implemented to prevent further high volume of traffic inflow, which can not only avoid traffic jams but also accelerate the cleanse of accident site. Any further policies could also be drawn on places where the occurrence of traffic accidents was frequent.

Data description

The dataset that will be used in this analysis is retrieved from Seattle Department of Transportation. It records all types of collisions since 2004. It includes the number of injuries or fatalities in every accident, whether a pedestrian or a bicycle or other traffic participants were involved, and variables where the accidents happened such as road condition, weather condition, light condition, etc.
The dataset has 194,673 records and 38 columns, the target variable is 'SEVERITYCODE', which uses numbers to indicate the levels severity related to the accident. The details of 'SEVERITYCODE' is as follows:

> 3 - fatality
>
> 2b - serious injury
>
> 2 - injury
>
> 1 - prop damage
>
> 0 - unknown

In our dataset, we only have two categories to work with (1 and 2).

Methodology

Preprocessing

This dataset has many columns with code or ID information, which do not provide useful insights for our model building, so the first step is to remove these code or ID columns. Notice that there are two target variable columns('SEVERITYCODE' and 'SEVERITYCODE.1'), we then performed a comparison to find if they are distinct.

```
comparison = pd.Series(df1['SEVERITYCODE'] == df1['SEVERITYCODE.1'])
comparison.value_counts()
```

```
True    194673
dtype: int64
```

And the answer is yes, so we removed one of them.
Next, we continued to clean our dataset, we removed 'SEVERITYDESC' and 'LOCATION' columns since these information has been either reflected through severity code or can be concluded using ADDRTYPE variable.
Then, we started to deal with missing values.

```
SEVERITYCODE           0
X                   5334
Y                   5334
STATUS                 0
ADDRTYPE            1926
COLLISIONTYPE       4904
PERSONCOUNT            0
PEDCOUNT               0
PEDCYLCOUNT            0
VEHCOUNT               0
INCDATE                0
INCDTTM                0
JUNCTIONTYPE        6329
INATTENTIONIND    164868
UNDERINFL           4884
WEATHER             5081
ROADCOND            5012
LIGHTCOND           5170
PEDROWNOTGRNT     190006
SPEEDING          185340
HITPARKEDCAR           0
dtype: int64
```

We can find that some variables have more than 80% of missing values, so we decided to remove these features once and for all.

```
# drop 3 columns
df1 = df1.drop(columns=['INATTENTIONIND','PEDROWNOTGRNT','SPEEDING'], axis=1)
```

We then replaced the rest missing values with proper values, for example, we used the mode value to replace ADDRTYPE, COLLISIONTYPE and other features; we used median value to replace geographical information.

```
# replace other missing values
values = {'X':df1['X'].median(), 'Y':df1['Y'].median(),
'ADDRTYPE':df1['ADDRTYPE'].mode().iloc[0],
'COLLISIONTYPE':df1['COLLISIONTYPE'].mode().iloc[0],
'JUNCTIONTYPE':df1['JUNCTIONTYPE'].mode().iloc[0],
'UNDERINFL':df1['UNDERINFL'].mode().iloc[0],
'WEATHER':df1['WEATHER'].mode().iloc[0],
'ROADCOND':df1['ROADCOND'].mode().iloc[0],
'LIGHTCOND':df1['LIGHTCOND'].mode().iloc[0]}
df1 = df1.fillna(value=values, axis=0)
```

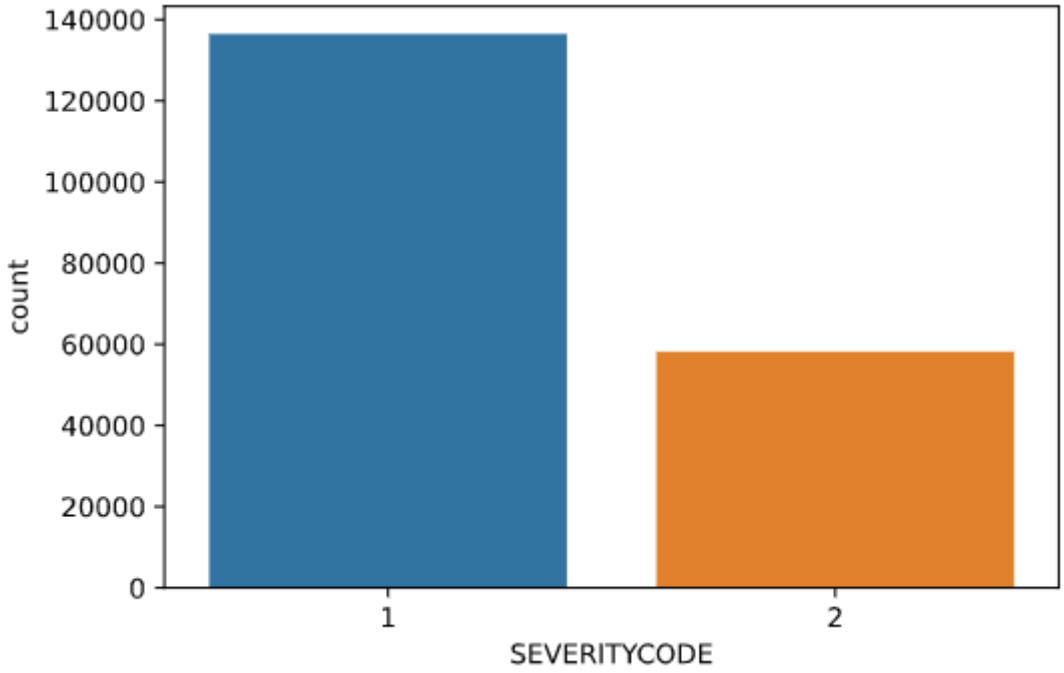
The last step of data wrangling was to transform the time format. We kept INCDTTM variable and transformed it to standard format.

```
# Drop date column
df1.drop(columns='INCDATE', axis=1, inplace=True)
# Convert DTTM
df1['INCDTTM'] = pd.to_datetime(df1['INCDTTM'])
```

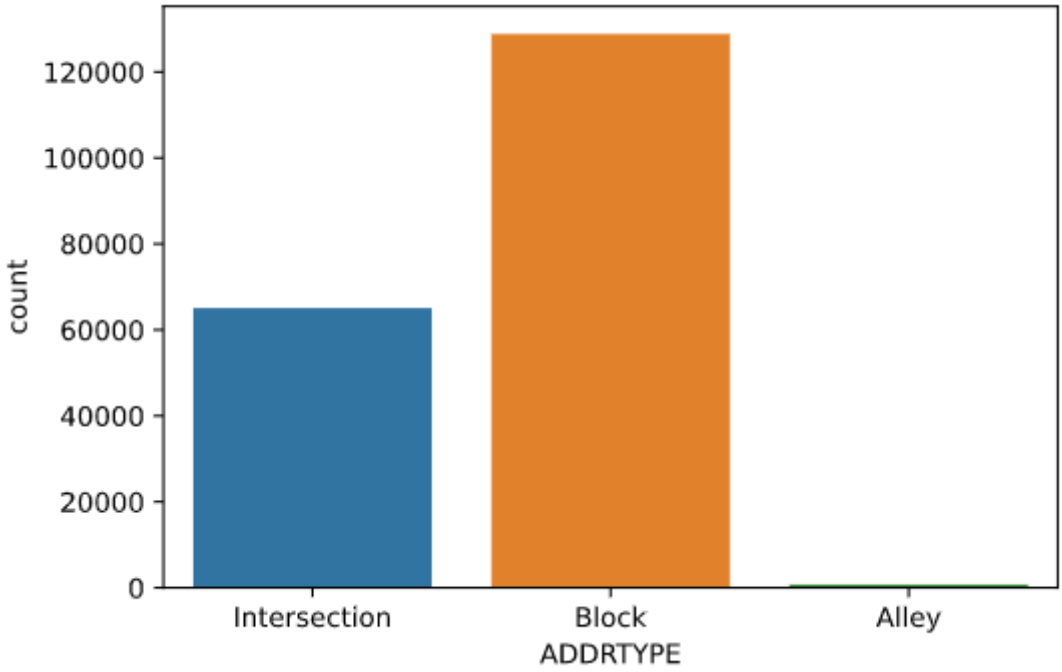
Now, we have finished data wrangling steps.

Visualization

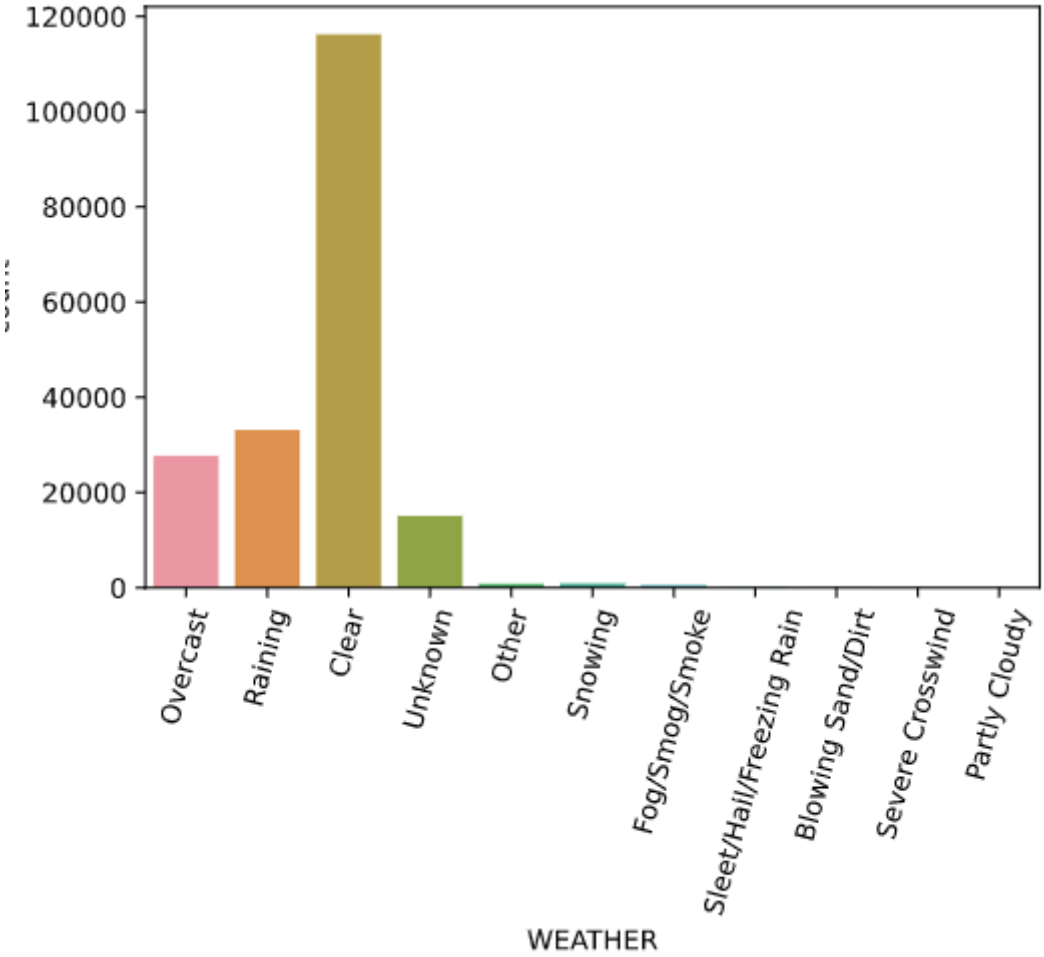
Since we have more than 190,000 records to discover, it is impossible to check the relationship between and within variables without asking for help from visualization tools. We first checked the distribution of two different severity levels.

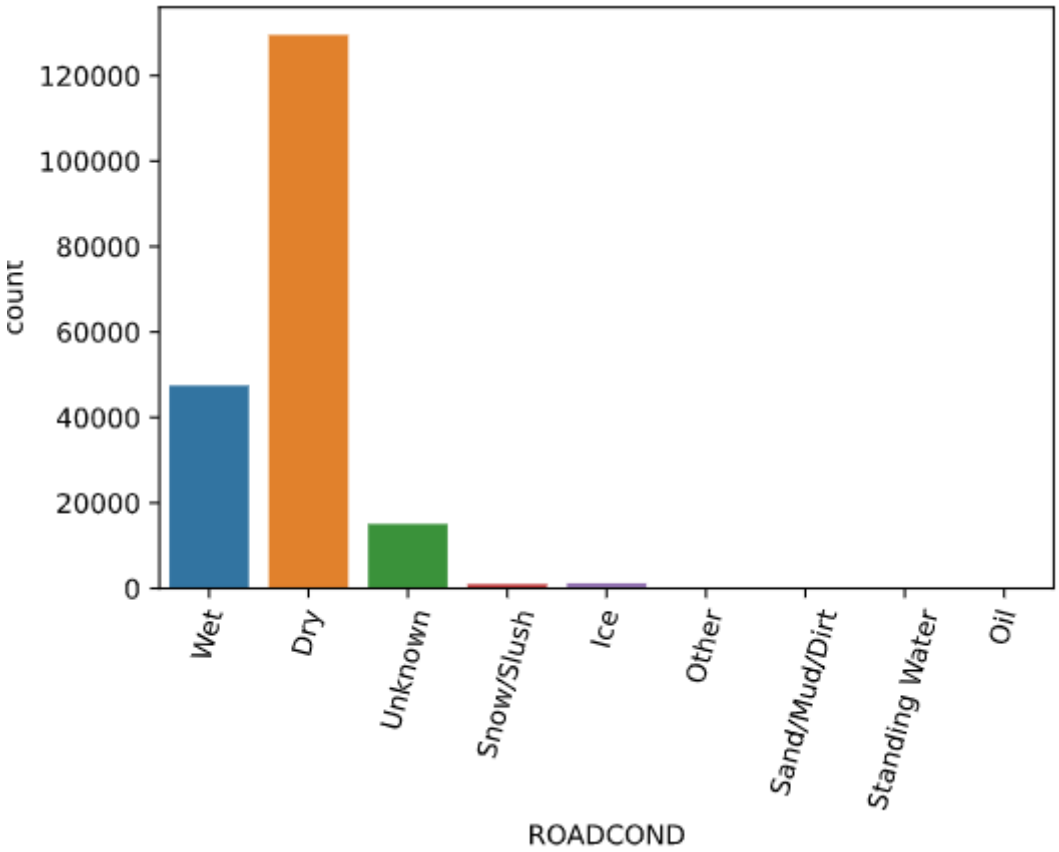


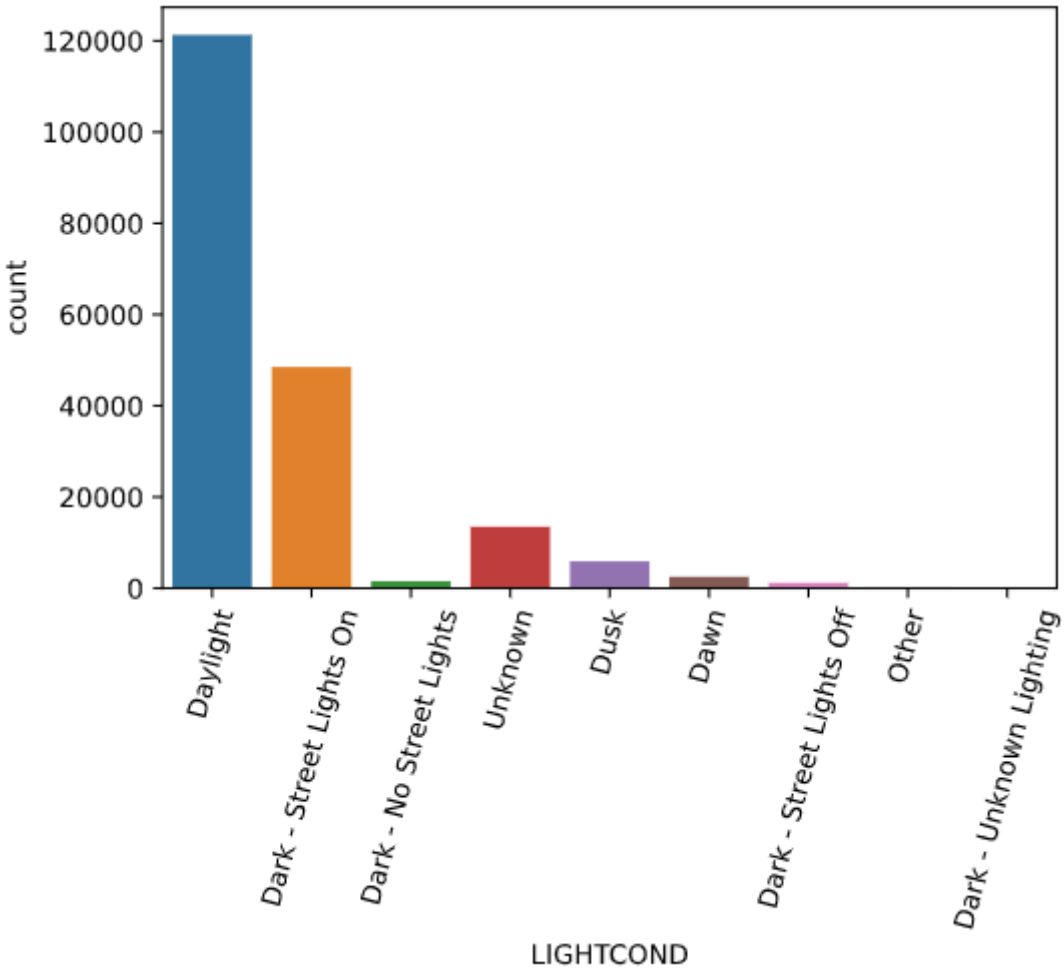
Second, we tried to understand where these accidents were occurred based on the category of address.



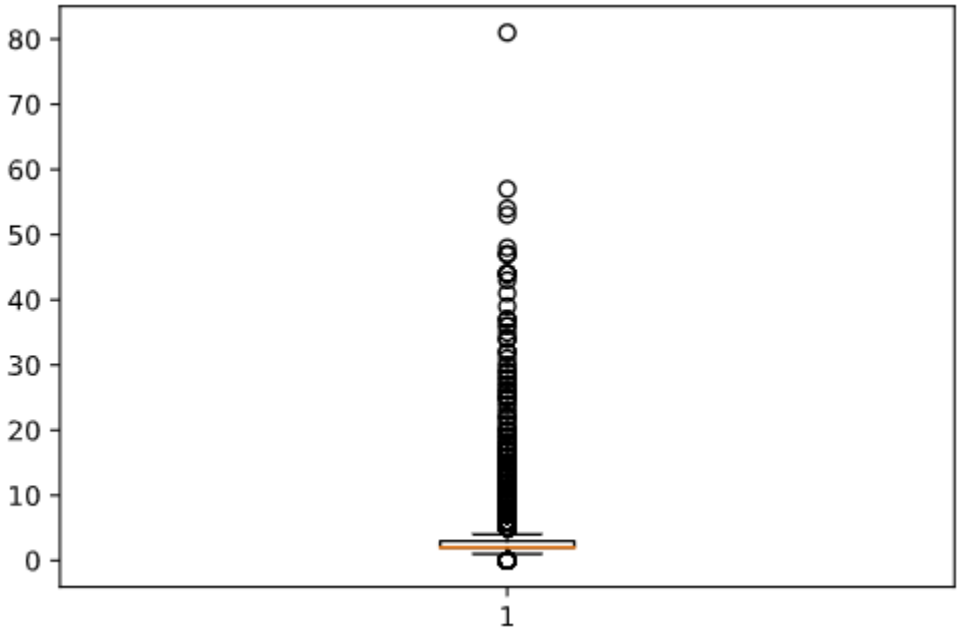
It is obvious that most accidents happened around blocks and intersections, only a few happened in an alley.
Then, let's see what was the condition at the moment of accidents.







We were also interested in the number of persons involved in the accidents. So we built a boxplot to get an idea.



It seems the majority part of accidents had less then 10 persons involved, but we still have quite a large part of accidents involved a lot more people.

Model building

Now, we would like to build our models. Before that, we had a last step to encode the categorical variables. We then had the ultimate independent variable set just as below:

```
X.head()
```

	STATUS	ADDRTYPE	COLLISIONTYPE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	JUNCTIONTYPE	UNDERINFL	WEATHER	ROADCOND	LIGHTCOND	HITPARKEDCAR
0	0	2	0	2	0	0	2	1	0	4	8	5	0
1	0	1	9	2	0	0	2	4	0	6	8	2	0
2	0	1	5	4	0	0	3	4	0	4	0	5	0
3	0	1	4	3	0	0	3	4	0	1	0	5	0
4	0	2	0	2	0	0	2	1	0	6	8	5	0

Next, we standardized these variables.

```
from sklearn import preprocessing
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
array([[-0.16046824,  1.39942178, -1.64247496, -0.33020207, -0.18743029,
        -0.16958841,  0.12553783, -1.28460737, -0.2217116 ,  0.34206956,
         1.50722693,  0.34496947, -0.19619929],
       [-0.16046824, -0.69049979,  1.63045406, -0.33020207, -0.18743029,
        -0.16958841,  0.12553783,  0.90108032, -0.2217116 ,  1.04697121,
         1.50722693, -1.42641774, -0.19619929],
       [-0.16046824, -0.69049979,  0.17581894,  1.15576451, -0.18743029,
        -0.16958841,  1.7102107 ,  0.90108032, -0.2217116 ,  0.34206956,
        -0.69828325,  0.34496947, -0.19619929],
       [-0.16046824, -0.69049979, -0.18783984,  0.41278122, -0.18743029,
        -0.16958841,  1.7102107 ,  0.90108032, -0.2217116 , -0.71528292,
        -0.69828325,  0.34496947, -0.19619929],
       [-0.16046824,  1.39942178, -1.64247496, -0.33020207, -0.18743029,
        -0.16958841,  0.12553783, -1.28460737, -0.2217116 ,  1.04697121,
         1.50722693,  0.34496947, -0.19619929]])
```

Since the dependent variable has an imbalanced distribution, we would like to do oversampling for our training and test set.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=2)
!pip install -U imbalanced-learn
from imblearn.over_sampling import SMOTE
over_samples = SMOTE(random_state=0)
over_samples_X,over_samples_y = over_samples.fit_sample(X_train,
y_train.values.reshape(-1,1))
```

Because my computer took a lot of time running models, I will only do K-Nearest Neighbors for model building

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(over_samples_X, over_samples_y)
y_pred_knn = knn.predict(X_test)
```

```
# I would like to run different models but my computer took a lot of time running them, sometimes it just stuck. So I will only use KNN as an example.
```

Results

> Train set Accuracy: 0.6934465308853666
> Test set Accuracy: 0.7213280367110715

Discussion

Although we ran only one model, we can get an idea that the severity level of an accident can be somewhat predicted according to the weather conditions, number of persons, bicycles or vehicles involved. It is useful for transportation departments to utilize these information to decide the best dealing method for both saving lives and ensuring the communication remains fluent.

Conclusion

Due to the computational power limitation, I am unable to run more models, however, there could be some improvements to make in the future. For example, we can split large dataset to pieces and run the models separately, which also provides a good way for cross validation. If we can have the geo API information, we can also use geographical coordinates to map the accidents and find which places have the highest risk for traffic accidents.