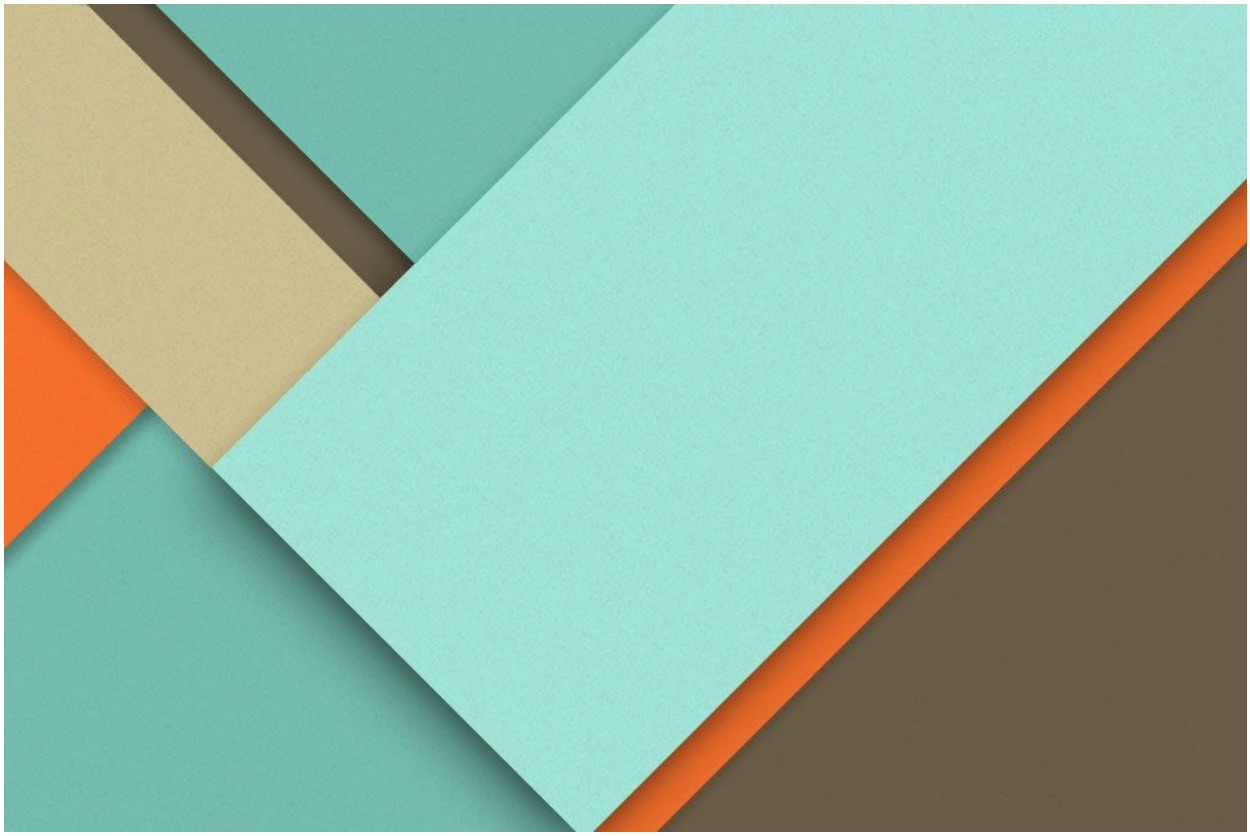


Capstone Project - Car Accident Severity

IBM Applied Data Science Capstone

By :

- Sandeep krishna Donepudi,

October 2020.

1. Introduction

1.1 Background

Road accidents constitute a major problem in our societies around the world. The World Health Organization (WHO) estimated that 1.25 million deaths were related to road traffic injuries in the year 2010. For the year 2016, the USA alone had recorded 37, 461 motor vehicle crash-related deaths, averaging around 102 people per day. In Europe, the statistics also indicate that each minute, there are 50 road deaths recorded in the year 2017.

1.2 Problem

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.

1.3 Target audience of this project

The target audience of the project is local government, police, rescue groups, and last but not least, car insurance institutes. These can get useful information from the model regarding the severity of an accident basis the factors mentioned above. Its results are going to provide some advice for the target audience to make insightful decisions for reducing the number of accidents and injuries. To be specific , for a given accident that just happened or a potential one, this model is supposed to be able to predict the likelihood of this accident being a severe one.

2. Data acquisition and Processing

2.1 Data Sources

The data for this project is taken from an open source website - https://data-seattlecitygis.opendata.arcgis.com/datasets/5b5c745e0f1f48e7a53acec63a0022ab_0/data

The data consists of 39 independent variables and 221525 rows. The dependent variable, "**SEVERITYCODE**", contains codes that correspond to different levels of severity caused by an accident.

```
df['SEVERITYCODE'].value_counts()
```

```
1     137671
2      58783
0      21615
2b      3105
3        350
Name: SEVERITYCODE, dtype: int64
```

Severity codes are as follows:

- 0 : Unknown
- 1 : Property Damage Only Collision
- 2 : Injury Collision
- 2b : Serious Injury Collision
- 3 : Fatality Collision

```
df['SEVERITYDESC'].value_counts()
```

```
Property Damage Only Collision    137671
Injury Collision                   58783
Unknown                            21616
Serious Injury Collision            3105
Fatality Collision                   350
Name: SEVERITYDESC, dtype: int64
```

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

2.2 Data 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, most of the features are of object data types that need to be converted into numerical data types. We have to convert the 'SEVERITYCODE' our target variable into numerical data type too.

For this we have updated the value of SEVERITYCODE '2b' to '4' and then updated the column data type.

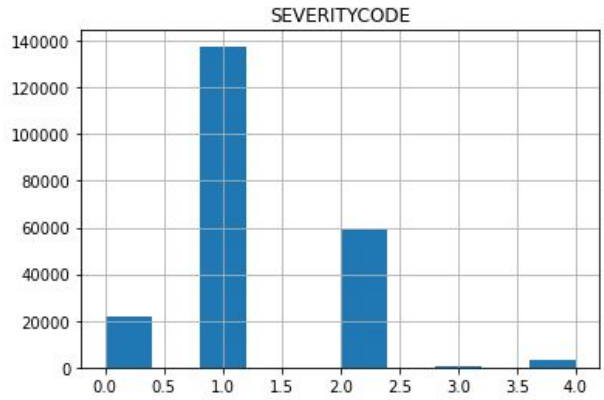
```
df['SEVERITYCODE'].value_counts()
```

```
1    137671
2     58783
0     21615
4      3105
3       350
Name: SEVERITYCODE, dtype: int64
```

To get a good understanding of the dataset, I have checked different values in the features. The results show the target feature is imbalance, so we use a simple statistical technique to balance it.

```
df.hist(column='SEVERITYCODE')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001A62DC566D0>]],
      dtype=object)
```



The number of rows in class 1 is almost three times bigger than the number of rows in class 2. It is possible to solve the issue by downsampling the class 1.

Post down sampling the data for class 1 we get a balanced target feature. Though the overall record size reduced but the model built would not be biased.

```
from sklearn.utils import resample

df_maj = df[df.SEVERITYCODE==1]
df_min = df[df.SEVERITYCODE!=1]

df_dsample = resample(df_maj,
                      replace=False,
                      n_samples= 65000,
                      random_state=123)

balanced_df = pd.concat([df_dsample, df_min])

balanced_df['SEVERITYCODE'].value_counts()
```

```
1.0    65000
2.0    58783
0.0    21615
4.0     3105
3.0      350
Name: SEVERITYCODE, dtype: int64
```

2.3 Feature Selection

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

Once I have loaded data into Pandas Dataframe, used 'dtypes' attribute to check the feature names and their data types. Then I have selected the most important features to predict the severity of accidents in Seattle. Among all the features, the following features have the most influence in the accuracy of the predictions:

- COLLISIONTYPE
- PERSONCOUNT
- PEDCOUNT
- PEDCYLCOUNT
- VEHCOUNT
- INJURIES
- SERIOUSINJURIES
- FATALITIES
- JUNCTIONTYPE
- WEATHER
- ROADCOND
- LIGHTCOND
- SPEEDING

Also, as I mentioned earlier, "**SEVERITYCODE**" is the target variable.

I have run a value count on road ('ROADCOND') and weather condition ('WEATHER') to get ideas of the different road and weather conditions. I also have run a value count on light conditions ('LIGHTCOND'), to see the breakdowns of accidents occurring during the different light conditions. Similarly we have run value count for other attributes as well. The results of a few can be seen below:

```
df_car['WEATHER'].value_counts()
```

```
Clear                       74459
Raining                     22350
Overcast                    18447
Unknown                      7650
Snowing                       511
Other                         487
Fog/Smog/Smoke                378
Sleet/Hail/Freezing Rain       72
Blowing Sand/Dirt              35
Severe Crosswind               17
Partly Cloudy                   7
Blowing Snow                    1
Name: WEATHER, dtype: int64
```

```
df_car['ROADCOND'].value_counts()
```

```
Dry                83395
Wet                31883
Unknown             7641
Ice                  726
Snow/Slush           558
Other                 86
Standing Water        75
Sand/Mud/Dirt         54
Oil                   49
Name: ROADCOND, dtype: int64
```

```
df_car['LIGHTCOND'].value_counts()
```

```
Daylight                    77912
Dark - Street Lights On     32097
Unknown                     6781
Dusk                        3962
Dawn                        1721
Dark - No Street Lights      954
Dark - Street Lights Off     793
Other                        138
Dark - Unknown Lighting       17
Name: LIGHTCOND, dtype: int64
```

```
df_car['JUNCTIONTYPE'].value_counts()
```

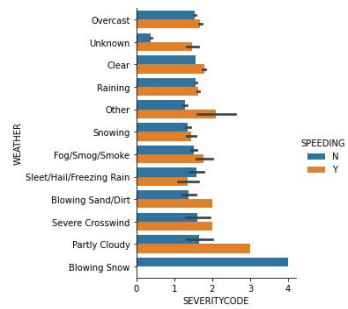
```
Mid-Block (not related to intersection)              64023
At Intersection (intersection related)               50359
Mid-Block (but intersection related)                 16256
Driveway Junction                                     7555
At Intersection (but not related to intersection)     1729
Ramp Junction                                          123
Unknown                                                 19
Name: JUNCTIONTYPE, dtype: int64
```

3. Exploratory Data Analysis

The below graph shows severity levels of accidents based on the different attributes like WEATHER, ROADCOND, LIGHTCOND and JUNCTIONTYPE.

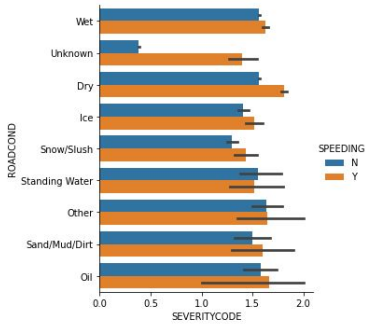
```
plt.figure(figsize=(20,5))
sns.catplot(x="SEVERITYCODE", y="WEATHER", hue="SPEEDING", kind="bar", data=df_car)
```

```
<seaborn.axisgrid.FacetGrid at 0x1cf074775e0>
<Figure size 1440x360 with 0 Axes>
```




```
plt.figure(figsize=(20,5))
sns.catplot(x="SEVERITYCODE", y="ROADCOND", hue="SPEEDING", kind="bar", data=df_car)
```

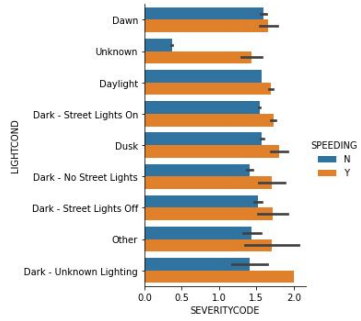
```
<seaborn.axisgrid.FacetGrid at 0x1cf12af83a0>
<Figure size 1440x360 with 0 Axes>
```



```
plt.figure(figsize=(20,5))
sns.catplot(x="SEVERITYCODE", y="LIGHTCOND", hue="SPEEDING", kind="bar", data=df_car)
```

```
<seaborn.axisgrid.FacetGrid at 0x1cf12b3ffd0>
```

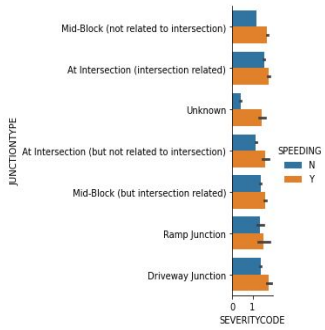
```
<Figure size 1440x360 with 0 Axes>
```




```
plt.figure(figsize=(20,5))
sns.catplot(x="SEVERITYCODE", y="JUNCTIONTYPE", hue="SPEEDING", kind="bar", data=df_car)
```

```
<seaborn.axisgrid.FacetGrid at 0x1cf12e153d0>
```

```
<Figure size 1440x360 with 0 Axes>
```



3.1 Data Preprocessing

Using df_car as the final data after removing extra fields, declare the following variables:

- X as the Feature Matrix (data of df_car)
- y as the response vector (SEVERITYCODE)

As you may figure out, some features in this dataset are categorical such as WEATHER, ROADCOND, LIGHTCOND, etc. Unfortunately; Sklearn Decision Trees do not handle categorical variables. But still we can convert these features to numerical values. pandas.get_dummies() Convert categorical variable into dummy/indicator variables.

```
from sklearn import preprocessing
le_Col = preprocessing.LabelEncoder()
le_Col.fit(['Angles','Rear Ended','Parked Car', 'Other', 'Sideswipe', 'Left Turn', 'Pedestrian', 'Cycles', 'Right Turn', 'Head Or
X[:,0] = le_Col.transform(X[:,0])


le_Jun = preprocessing.LabelEncoder()
le_Jun.fit([ 'Mid-Block (not related to intersection)', 'At Intersection (intersection related)', 'Mid-Block (but intersection re
X[:,8] = le_Jun.transform(X[:,8])


le_Wthr = preprocessing.LabelEncoder()
le_Wthr.fit([ 'Clear', 'Raining', 'Overcast', 'Unknown', 'Snowing', 'Other', 'Fog/Smog/Smoke', 'Sleet/Hail/Freezing Rain', 'Blowi
X[:,9] = le_Wthr.transform(X[:,9])

le_Rcnd = preprocessing.LabelEncoder()
le_Rcnd.fit([ 'Dry', 'Wet', 'Unknown', 'Ice', 'Snow/Slush', 'Other', 'Standing Water', 'Sand/Mud/Dirt', 'Oil'])
X[:,10] = le_Rcnd.transform(X[:,10])

le_Lcnd = preprocessing.LabelEncoder()
le_Lcnd.fit([ 'Daylight', 'Dark - Street Lights On', 'Unknown', 'Dusk', 'Snow/Slush', 'Dawn', 'Dark - No Street Lights', 'Dark -
X[:,11] = le_Lcnd.transform(X[:,11])

le_Spd = preprocessing.LabelEncoder()
le_Spd.fit([ 'Y', 'N'])
X[:,12] = le_Spd.transform(X[:,12])

X[0:5]
```

```
array([[0, 2, 0, 0, 2, 0, 0, 0, 2, 2, 0, 2, 0],
       [0, 2, 0, 0, 2, 0, 0, 0, 0, 11, 7, 9, 0],
       [5, 2, 0, 0, 2, 0, 0, 0, 6, 11, 7, 9, 0],
       [5, 2, 0, 0, 2, 0, 0, 0, 4, 11, 7, 9, 0],
       [5, 2, 0, 0, 2, 0, 0, 0, 4, 11, 7, 9, 0]], dtype=object)
```

3.2 Normalize Data

Data Standardization give data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on distance of cases:

```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

```
array([[ 0.81883233, -0.11190296, -0.22670164, -0.19763718,  0.41224259,
        -0.6672496 , -0.1176568 , -0.04614688,  0.7345875 , -0.03049773,
         1.27613092, -0.49199936, -0.22042029],
       [-0.08243785, -0.11190296, -0.22670164, -0.19763718,  0.41224259,
        -0.6672496 , -0.1176568 , -0.04614688, -1.18119427,  1.6350295 ,
         1.0071595 ,  1.62673697, -0.22042029],
       [-0.08243785, -0.11190296, -0.22670164, -0.19763718,  0.41224259,
        -0.6672496 , -0.1176568 , -0.04614688,  0.7345875 , -0.86326135,
        -0.87564045, -0.06825209, -0.22042029],
       [ 1.11925572,  0.52270657, -0.22670164, -0.19763718,  0.41224259,
        -0.6672496 , -0.1176568 , -0.04614688,  0.7345875 , -0.86326135,
        -0.87564045, -0.06825209, -0.22042029],
       [-0.38286124, -0.74651249, -0.22670164, -0.19763718, -0.68330057,
        -0.6672496 , -0.1176568 , -0.04614688,  0.7345875 ,  0.52467801,
         1.27613092, -0.06825209, -0.22042029]])
```

3.3 Train - Test Split

Out of Sample Accuracy is the percentage of correct predictions that the model makes on data that the model has NOT been trained on. Doing a train and test on the same dataset will most likely have low out-of-sample accuracy, due to the likelihood of being over-fit.

It is important that our models have a high, out-of-sample accuracy, because the purpose of any model, of course, is to make correct predictions on unknown data. So how can we improve out-of-sample accuracy? One way is to use an evaluation approach called Train/Test Split. Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (119169, 13) (119169,)
Test set: (29793, 13) (29793,)
```

4. Modelling

After balancing SEVERITYCODE feature, and standardizing the input feature, the data has been ready for building machine learning models.

I have employed three machine learning models:

- K Nearest Neighbor (KNN)
- Decision Tree
- Logistic Regression

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

4.1 K-Nearest Neighbor (KNN)

```
from sklearn.neighbors import KNeighborsClassifier

k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
KNeighborsClassifier(n_neighbors=4)
```

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

```
array([1., 0., 1., 1., 1.])
```

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.9912057771426928
Test set Accuracy:  0.9917205576316319
```

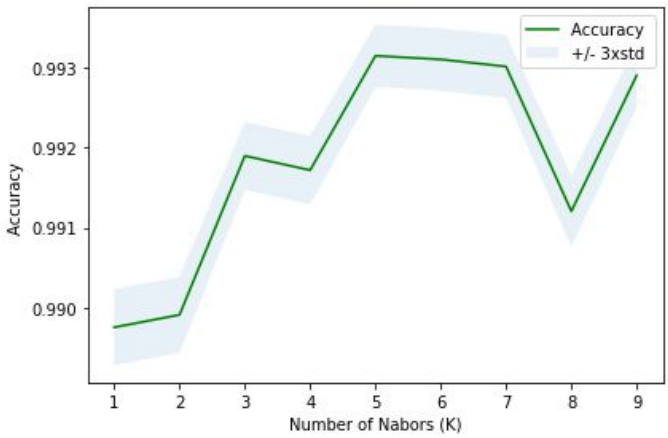
We can calculate the accuracy of KNN for different Ks.

```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

```
array([0.98351962, 0.99177659, 0.9934884 , 0.99177659, 0.99325345,
       0.99328701, 0.99311919, 0.99311919, 0.99308562])
```

Plot model accuracy for Different number of Neighbors:

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```




```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.993152677392647 with k= 5
```

Then we rebuild the model with the best value for K=5.

```
k = 5
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
KNeighborsClassifier()
```

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

```
array([1., 0., 1., 1., 1.])
```

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.9925579967968697
Test set Accuracy:  0.993152677392647
```

```
from sklearn.metrics import f1_score

print("DT F1-score: %.2f" % f1_score(y_test, yhat, average='weighted') )
```

```
DT F1-score: 0.99
```

4.2 Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
DT_model = DecisionTreeClassifier(criterion="entropy", max_depth = 5)
DT_model.fit(X_train,y_train)
DT_model
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5)
```

```
yhat = DT_model.predict(X_test)
yhat
```

```
array([1., 0., 1., ..., 2., 2., 0.])
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
DecisionTrees's Accuracy:  0.9931750542639128
```

```
from sklearn.metrics import f1_score

print("DT F1-score: %.2f" % f1_score(y_test, yhat, average='weighted') )
```

```
DT F1-score: 0.99
```

4.3 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
LR_model = LogisticRegression(C=0.01).fit(X_train,y_train)
LR_model
```

```
LogisticRegression(C=0.01)
```

```
yhat = LR_model.predict(X_test)
yhat
```

```
array([1., 0., 1., ..., 2., 2., 0.])
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss

LR_yhat = LR_model.predict(X_test)
LR_yhat_prob = LR_model.predict_proba(X_test)

print("LR F1-score: %.2f" % f1_score(y_test, LR_yhat, average='weighted'))
print("LR LogLoss: %.2f" % log_loss(y_test, LR_yhat_prob))
```

```
LR F1-score: 0.99
LR LogLoss: 0.04
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
DecisionTrees's Accuracy:  0.990914990266061
```

5. Results and Evaluations

ML Model	F1-Score	Accuracy
KNN	0.99	0.993
Decision Tree	0.99	0.993
Logistic Regression	0.99	0.990

6. Conclusion

Based on the dataset provided for this capstone from weather, road, light conditions, etc. 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. These models can be very helpful to reduce the economic and social impact of car accidents.