IBM Data Science Capstone: Car Accident Severity Report

Akshara Gopakumar

Background

A traffic collision occurs when a vehicle collides with another vehicle, pedestrian, animal, road debris or other stationary obstacle, such as a tree, pole or building. Traffic accidents often result in injury, disability, death, and damage to property, as well as financial costs for both society and the people involved. Injuries to road traffic cause substantial economic damages to people, their communities , and nations as a whole. These losses result from medical costs as well as from loss of productivity for those killed or affected by their injuries, and for family members who need to take time off work or school to care for the injured

Problem Description

Residing in a metropolitan area my whole life, it hurts to see many people getting injured on a daily basis. I believe if we know the root cause of these injuries we can make solutions for the future. Hence, the objective of the project was to investigate accidents, to research on what types of traffic collisions are most likely to result in injury.

Data Understanding:

We will use SEVERITYCODE impacted by several variables, the most important ones being WEATHER, ROADCOND and LIGHTCOND.

Preprocessing

After importing the packages, let us understand what is in the dataset first

In [1]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

In [2]:
```
ead_csv("https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-(
()
```

```
/Users/aksharagopakumar/.local/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3049: DtypeWarning: Colum
ns (33) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[2]:

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDF
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight	
1	1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On	
2	1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight	
3	1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight	
4	2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight	

5 rows × 38 columns

Methodology

Normalize the Data

In [21]:
```
DF.groupby(['PEDCOUNT'])['SEVERITYCODE'].value_counts(normalize=True)
```

Out[21]:
```
PEDCOUNT  SEVERITYCODE
0         1               0.527439
          2               0.472561
1         2               0.948973
          1               0.051027
2         2               0.981043
          1               0.018957
3         2               0.954545
          1               0.045455
4         2               1.000000
5         2               1.000000
6         2               1.000000
Name: SEVERITYCODE, dtype: float64
```

In [22]:
```
DF.groupby(['PEDCYLCOUNT'])['SEVERITYCODE'].value_counts(normalize=True)
```

Out[22]:
```
PEDCYLCOUNT  SEVERITYCODE
0            1               0.520625
             2               0.479375
1            2               0.940735
             1               0.059265
2            2               1.000000
Name: SEVERITYCODE, dtype: float64
```

In [26]:
```
DF.groupby(['WEATHER'])['SEVERITYCODE'].value_counts(normalize=True)
```

Out[26]:
```
WEATHER                   SEVERITYCODE
Blowing Sand/Dirt         1               0.516129
                          2               0.483871
Clear                     2               0.525452
                          1               0.474548
Fog/Smog/Smoke            2               0.525281
                          1               0.474719
Other                     1               0.742222
                          2               0.257778
Overcast                  2               0.518345
                          1               0.481655
Partly Cloudy             2               0.600000
                          1               0.400000
Raining                   2               0.538888
                          1               0.461112
Severe Crosswind          1               0.562500
                          2               0.437500
Sleet/Hail/Freezing Rain  1               0.594203
                          2               0.405797
Snowing                   1               0.628261
                          2               0.371739
Unknown                   1               0.883429
                          2               0.116571
Name: SEVERITYCODE, dtype: float64
```

In [29]:
```
DF.groupby(['ROADCOND'])['SEVERITYCODE'].value_counts(normalize=True)
```
Out[29]:
```
ROADCOND        SEVERITYCODE
Dry             2               0.525711
                1               0.474289
Ice             1               0.566502
                2               0.433498
Oil             2               0.648649
                1               0.351351
Other           2               0.594203
                1               0.405797
Sand/Mud/Dirt   1               0.531915
                2               0.468085
Snow/Slush      1               0.647059
                2               0.352941
Standing Water  1               0.558824
                2               0.441176
Unknown         1               0.742222
                2               0.257778
Wet             2               0.536109
                1               0.463891
Name: SEVERITYCODE, dtype: float64
```

In [32]:
```
DF.groupby(['LIGHTCOND'])['SEVERITYCODE'].value_counts(normalize=True)
```
Out[32]:
```
LIGHTCOND                 SEVERITYCODE
Dark - No Street Lights   1               0.583979
                          2               0.416021
Dark - Street Lights Off  1               0.532526
                          2               0.467474
Dark - Street Lights On   2               0.506505
                          1               0.493495
Dark - Unknown Lighting   2               0.600000
                          1               0.400000
Dawn                      2               0.537333
                          1               0.462667
Daylight                  2               0.541513
                          1               0.458487
Dusk                      2               0.550231
                          1               0.449769
Other                     1               0.580357
                          2               0.419643
Unknown                   1               0.791923
                          2               0.208077
Name: SEVERITYCODE, dtype: float64
```

Train Test Split

In [47]:
```
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: (84316, 11) (84316,)
Test set: (21080, 11) (21080,)
```

Now we can build our models

K-Nearest Neighbor (KNN)

KNN will help us predict the severity code of an outcome by finding the most similar to data point within k distance.

In [44]:
```
from sklearn.linear_model import LinearRegression
X=DF[['WEATHER']]
Y=DF[['ROADCOND']]
lm=LinearRegression()
lm.fit(X,Y)
Yhat=lm.predict(X)
lm.score(X,Y)
```
Out[44]: 0.6092365003242208

In [45]:
```
['ADDRTYPE','PERSONCOUNT','PEDCOUNT','PEDCYLCOUNT','VEHCOUNT','WEATHER','LIGHTCOND','SPEEDING','INATTENTIONIND','UNI
et_index(drop=True,inplace=True)
d()
```
Out[45]:

	ADDRTYPE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	WEATHER	LIGHTCOND	SPEEDING	INATTENTIONIND	UNDERINFL	HITPARKEDCAI
0	2.0	3	0	0	2	3.0	1.0	0	0	0	[illegible]
1	2.0	2	0	0	2	2.0	1.0	0	0	0	[illegible]
2	1.0	2	0	0	2	1.0	2.0	0	0	0	[illegible]
3	1.0	2	0	0	2	1.0	1.0	0	1	0	[illegible]
4	1.0	2	0	0	2	2.0	3.0	0	0	1	[illegible]

Decision Tree

A decision tree model gives us a layout of all possible outcomes so we can fully analyze the consequences of a decision. Its context, the decision tree observes all possible outcomes of different weather conditions.

In [48]:
```python
from sklearn.tree import DecisionTreeClassifier
Tree = DecisionTreeClassifier(criterion="entropy", max_depth =5)
Tree.fit(X_train,y_train)
Tree
```

Out[48]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [49]:
```python
yhat = Tree.predict(X_test)
yhat
```

Out[49]:
```
array([2, 2, 2, ..., 2, 2, 2])
```

In [50]:
```python
from sklearn import metrics
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
DecisionTrees's Accuracy:  0.6509013282732448
```

In [51]:
```python
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           1       0.66      0.52      0.58      9851
           2       0.65      0.76      0.70     11229

   micro avg       0.65      0.65      0.65     21080
   macro avg       0.65      0.64      0.64     21080
weighted avg       0.65      0.65      0.65     21080
```

In [54]:
```python
X1=DF[['ADDRTYPE','WEATHER','LIGHTCOND','SPEEDING','INATTENTIONIND','UNDERINFL']]
X1.reset_index(drop=True,inplace=True)
X1.head()
```

Out[54]:

	ADDRTYPE	WEATHER	LIGHTCOND	SPEEDING	INATTENTIONIND	UNDERINFL
0	2.0	3.0	1.0	0	0	0
1	2.0	2.0	1.0	0	0	0
2	1.0	1.0	2.0	0	0	0
3	1.0	1.0	1.0	0	1	0
4	1.0	2.0	3.0	0	0	1

In [55]:
```python
X_train, X_test, y_train, y_test = train_test_split(X1, y, test_size=0.2, random_state=4)
Tree1 = DecisionTreeClassifier(criterion="entropy", max_depth =5)
Tree1.fit(X_train,y_train)
yhat1 = Tree1.predict(X_test)
yhat1
```

Out[55]:
```
array([2, 2, 2, ..., 2, 1, 2])
```

In [56]:
```python
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, yhat1))
print (classification_report(y_test, yhat1))
```

```
DecisionTrees's Accuracy:  0.5969639468690702
              precision    recall  f1-score   support

           1       0.57      0.55      0.56      9851
           2       0.62      0.64      0.63     11229

   micro avg       0.60      0.60      0.60     21080
   macro avg       0.59      0.59      0.59     21080
weighted avg       0.60      0.60      0.60     21080
```

Logistic Regression

Because our dataset only provides us with two severity code outcomes, our model will only predict one of those two classes. This makes our data binary, which is perfect to use with logistic regression.

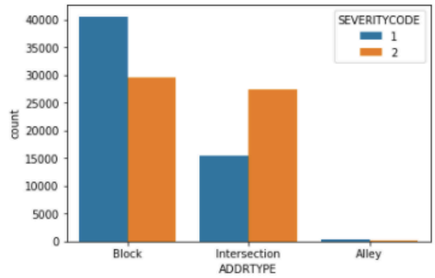
Results & Evaluation:

Then we checked the accuracy and found out that the Decision tree is the most accurate model.

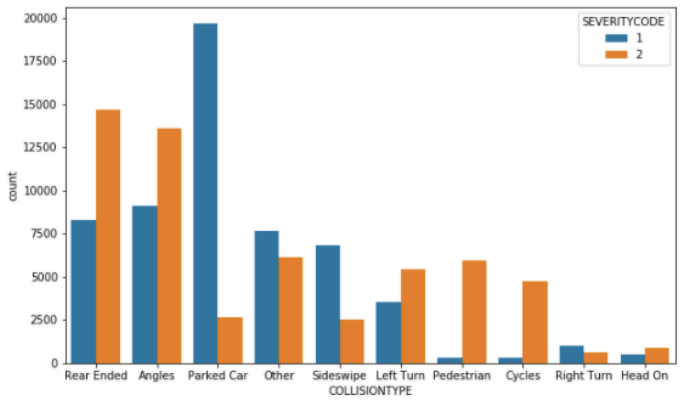
Discussion:

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.

```
In [17]: sns.countplot(x='ADDRTYPE', hue='SEVERITYCODE',data=DF, order=DF['ADDRTYPE'].value_counts().index)
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x118aabef0>
```

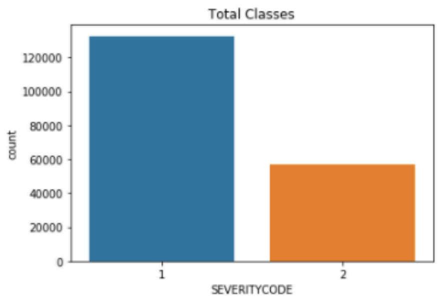



```
In [13]: f, ax = plt.subplots(figsize=(10, 6))
         sns.countplot(x='COLLISIONTYPE', hue='SEVERITYCODE',data=DF,order=DF['COLLISIONTYPE'].value_counts().index)
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x116ffbb70>
```



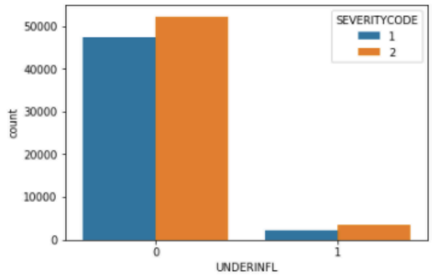

```
In [10]: import seaborn as sns
         import matplotlib.pyplot as plt
         %matplotlib inline
         sns.countplot('SEVERITYCODE', data=DF)
         plt.title('Balanced Classes')
         plt.show()
```

In [8]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.countplot('SEVERITYCODE', data=df)
plt.title('Total Classes')
plt.show()
```



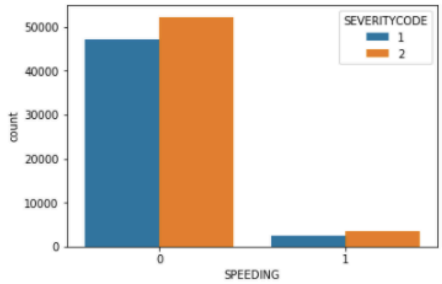
In [41]:
```
sns.countplot(x="UNDERINFL", hue="SEVERITYCODE", data=DF)
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x1172087b8>



In [43]:
```
sns.countplot(x="SPEEDING", hue="SEVERITYCODE", data=DF)
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x11994bc88>



Conclusion:

Based on historical data from weather conditions pointing to certain classes, we can conclude that particular weather conditions have a somewhat impact on whether or not travel could result in property damage or injury .