Data Science Capstone: Car Accident Severity Report

By **IBM Data Science**

Introduction & Business Understanding:

In an effort to reduce the frequency of car collisions in a community, a machine learning model must be developed to predict the severity of an accident given the current weather, road and visibility conditions. When conditions are bad, Whether or not a driver involved was under the influence of drugs or alcohol, this model will alert drivers to remind them to be more careful on the Roads.

Data Understanding:

Data source: https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv
The meta-data: https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Metadata.pdf

The data was collected by SDOT Traffic Management Division, Traffic Records Group from 2004 to present.
The data consists of 37 independent variable and 194,673 rows, and the variable thats gonna be our target will be SEVERITYCODE because it is used measure the severity of an accident from 0 to 5 within the dataset. Attributes used to weigh the severity of an accident are WEATHER, ROADCOND and LIGHTCOND.

- Severity codes are as follows:
  0. Unknown (Clear Conditions)
  1. Property damage
  2. injury
  3. serious injury
  4. fatality

```
In [6]: df.shape
Out[6]: (194673, 38)

In [7]: df.head()
```

Out[7]:

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

5 rows × 38 columns

```
In [8]: print(df.columns)
Index(['SEVERITYCODE', 'X', 'Y', 'OBJECTID', 'INCKEY', 'COLDETKEY', 'REPORTNO',
       'STATUS', 'ADDRTYPE', 'INTKEY', 'LOCATION', 'EXCEPTRSNCODE',
       'EXCEPTRSNDESC', 'SEVERITYCODE.1', 'SEVERITYDESC', 'COLLISIONTYPE',
       'PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT', 'INCDATE',
       'INCDTTM', 'JUNCTIONTYPE', 'SDOT_COLCODE', 'SDOT_COLDESC',
       'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND',
       'PEDROWNOTGRNT', 'SDOTCOLNUM', 'SPEEDING', 'ST_COLCODE', 'ST_COLDESC',
       'SEGLANEKEY', 'CROSSWALKKEY', 'HITPARKEDCAR'],
      dtype='object')
```

Other important variables include:
• ADDRTYPE: Collision address type: Alley, Block, Intersection
• LOCATION: Description of the general location of the collision
• PERSONCOUNT: The total number of people involved in the collision helps identify severity involved
• PEDCOUNT: The number of pedestrians involved in the collision helps identify severity involved
• PEDCYLCOUNT: The number of bicycles involved in the collision helps identify severity involved
• VEHCOUNT: The number of vehicles involved in the collision identify severity involved
• JUNCTIONTYPE: Category of junction at which collision took place helps identify where most collisions occur
• WEATHER: A description of the weather conditions during the time of the collision
• ROADCOND: The condition of the road during the collision
• LIGHTCOND: The light conditions during the collision
• SPEEDING: Whether or not speeding was a factor in the collision (Y/N)
• SEGLANEKEY: A key for the lane segment in which the collision occurred
• CROSSWALKKEY: A key for the crosswalk at which the collision occurred
• HITPARKEDCAR: Whether or not the collision involved hitting a parked car

Data Preprocessing:

The dataset from the original is not ready to begin analysis. First thing first, we need to drop the non-relevant columns. In addition, most of the features are of object data types (String) that need to be converted into numerical data types(float or int).

After analyzing the data, I conclude to focus precisely on only four features, severity, weather conditions, road conditions, and light conditions, among others.

To get a clear vision on the dataset, I have seen different values in the features. The results show in the image below, the variable *SEVERITYCODE* is imbalance, so we use a simple technique to balance the two different category .

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

[30]: 1    136485
      2     58188
      Name: SEVERITYCODE, dtype: int64
```

As you can see, the number of rows in *category 1* is greater than the number of rows in *category 2.*

```
31]: from sklearn.utils import resample
```

```
33]: df_maj = df[df.SEVERITYCODE == 1]
     df_min = df[df.SEVERITYCODE == 2]
     df_2_samples = resample(df_maj, replace=False, n_samples=58188, random_state=0)

     df_balanced = pd.concat([df_2_samples, df_min])
     df_balanced['SEVERITYCODE'].value_counts()
```

```
33]: 2    58188
     1    58188
     Name: SEVERITYCODE, dtype: int64
```

As e can see on the screenshot below there is 2795 row in our dataset that has WEATHER or ROADCOND FEATURE is UNKNOW (Nan) we use a simple method in ***SKLEARN*** to remove the rows that had nan values, this method used only if we have a large dataset if not we can use the average method or the mod frequency value to replace the missing value on our dataset.

```
[54]: df2.isna().sum()
```

```
[54]: SEVERITYCODE       0
      WEATHER          2795
      ROADCOND         2758
      dtype: int64
```

```
[59]: df2.dropna(inplace=True)
      df2.isna().sum()
```

```
/Users/Mehdi/opt/anaconda3/lib/python3.
A value is trying to be set on a copy o

See the caveats in the documentation: h
view-versus-a-copy
  """Entry point for launching an IPyth
```

```
[59]: SEVERITYCODE     0
      WEATHER          0
      ROADCOND         0
      dtype: int64
```

as we can see the dataset not all the feature or independent variable in our dataset are numerical so we need to convert the columns who are type object to a numerical the solution for data is the use the oneHotEncode method in sklearn to get the dummy variable

```
[122]: X = df2.iloc[:, 1:].values
       y = df2.iloc[:, 0].values
       print(X)
       print(y)
```

```
[['Raining' 'Wet']
 ['Clear' 'Ice']
 ['Clear' 'Dry']
 ...
 ['Clear' 'Dry']
 ['Clear' 'Dry']
 ['Clear' 'Dry']]
[1 1 1 ... 2 2 2]
```

```
[123]: print(X.shape)
       print(y.shape)
```

```
(113549, 2)
(113549,)
```

```
[124]: from sklearn.preprocessing import OneHotEncoder
       from sklearn.compose import ColumnTransformer
```

```
[125]: cl = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[0, 1])], remainder='passthrough')
```

```
[126]: X = cl.fit_transform(X)
       X
```

```
[126]: <113549x20 sparse matrix of type '<class 'numpy.float64'>'
              with 227098 stored elements in Compressed Sparse Row format>
```

```
[127]: X.shape
```

```
[127]: (113549, 20)
```

```
[109]: print(X)
  (0, 6)          1.0
  (0, 19)         1.0
  (1, 1)          1.0
  (1, 12)         1.0
  (2, 1)          1.0
  (2, 11)         1.0
  (3, 6)          1.0
  (3, 19)         1.0
  (4, 10)         1.0
  (4, 18)         1.0
  (5, 1)          1.0
  (5, 11)         1.0
  (6, 1)          1.0
  (6, 11)         1.0
  (7, 1)          1.0
  (7, 11)         1.0
  (8, 4)          1.0
  (8, 11)         1.0
  (9, 6)          1.0
  (9, 19)         1.0
  (10, 1)         1.0
  (10, 11)        1.0
  (11, 1)         1.0
  (11, 11)        1.0
  (12, 10)        1.0
  :       :
  (113536, 11)    1.0
  (113537, 6)     1.0
  (113537, 19)    1.0
  (113538, 1)     1.0
  (113538, 11)    1.0
  (113539, 1)     1.0
  (113539, 11)    1.0
  (113540, 4)     1.0
  (113540, 19)    1.0
```

Methodology:

Our data is now ready to be fed into machine learning models.

We will use the following 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.

Decision Tree

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

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.

Define X and y:

```
[27]: X = df_balanced.iloc[:, 4:].values
      y = df_balanced.iloc[:, 3].values

[28]: X[:5]

[28]: array([[ 1,  0,  2],
             [ 3,  7,  8],
             [10,  7,  8],
             [ 6,  8,  5],
             [ 1,  0,  5]], dtype=int8)

[29]: y[: 5]

[29]: array([1, 1, 1, 1, 1])
```

Normalize the dataset:

```
[30]: from sklearn.preprocessing import StandardScaler
      X = StandardScaler().fit_transform(X)
      X[: 5]

[30]: array([[-0.71908766, -0.69400808, -1.43997571],
             [ 0.0169217 ,  1.22836933,  2.21133186],
             [ 2.59295448,  1.22836933,  2.21133186],
             [ 1.12093575,  1.50299467,  0.38567807],
             [-0.71908766, -0.69400808,  0.38567807]])
```

Train/Test Split:

We Will use 70% for training set and 30% for Test set.

[32]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=0)
```

[33]:
```
print("X_train size: ", X_train.shape)
print("y_train size: ", y_train.shape)
print("X_test size: ", X_test.shape)
print("y_test size: ", y_test.shape)
```

```
X_train size:  (80668, 3)
y_train size:  (80668,)
X_test size:  (34572, 3)
y_test size:  (34572,)
```

Building the models:

Building the KNN Model

[35]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[36]:
```
KNN = KNeighborsClassifier(5)
```

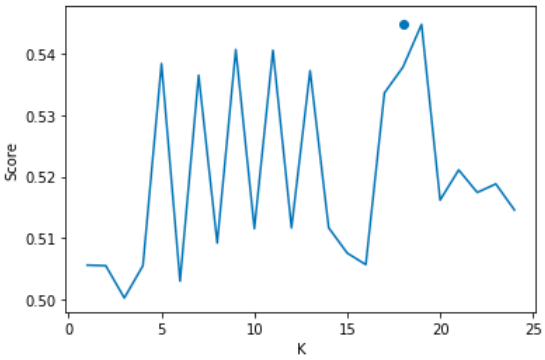
[37]:
```
KNN.fit(X_train, y_train)
```

[37]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

Finding the best K for the KNN:

[48]:
```
best_k = []
score = []
for k in range(1, 25):
    KNN = KNeighborsClassifier(k)
    KNN.fit(X_train, y_train)
    best_k.append(k)
    ## The score method in the instance of KNN is just same as f1_score
    score.append(KNN.score(X_test, y_test))
```

[53]:
```
plt.plot(range(1, 25), score)
plt.scatter(score.index(max(score)), max(score))
plt.xlabel('K')
plt.ylabel('Score')
plt.show()
```



[56]:
```
KNN = KNeighborsClassifier(18).fit(X_train, y_train)
```

[57]:
```
y_predKNN = KNN.predict(X_test)
```

[58]:
```
y_predKNN[:5]
```

[58]:
```
array([1, 2, 2, 1, 2])
```

Building The Decision Tree Model

for Decision Tree we don't need to Scale our Feature but it will work fine with Scalled feature

]:
```
from sklearn.tree import DecisionTreeClassifier
DC = DecisionTreeClassifier(criterion="entropy", max_depth = 7)
DC.fit(X_train,y_train)
```

]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=7, 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='deprecated',
                       random_state=None, splitter='best')
```

]:
```
DT_ypredt = DC.predict(X_test)
print (DT_ypredt [0:5])
print (y_test [0:5])
```

```
[1 2 2 2 2]
[2 2 1 1 2]
```

Logistic Regression

[61]:
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(C=0.1).fit(X_train,y_train)
LR
```

[61]:
```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[62]:
```
LR_pred = LR.predict(X_test)
LR_pred[:5]
```

[62]:
```
array([1, 2, 2, 1, 1])
```

[63]:
```
LR_predPro =  LR.predict_proba(X_test)
```

Results and Evaluations:

[64]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

[65]:
```
print('The Jaccard Score:')
print('KNN Jaccard:{:.2f}'.format(jaccard_similarity_score(y_test, y_predKNN)))
print('Decision Tree Jaccard:{:.2f}'.format(jaccard_similarity_score(y_test, DT_ypredt)))
print('Logistic Regression Jaccard:{:.2f}'.format(jaccard_similarity_score(y_test, LR_pred)))
print('\nThe f1_score')
print('KNN f1_score:{:.2f}'.format(f1_score(y_test, y_predKNN)))
print('Decision Tree f1_score:{:.2f}'.format(f1_score(y_test, DT_ypredt)))
print('Logistic Regression f1_score:{:.2f}'.format(f1_score(y_test, LR_pred)))
```

```
The Jaccard Score:
KNN Jaccard:0.54
Decision Tree Jaccard:0.56
Logistic Regression Jaccard:0.53

The f1_score
KNN f1_score:0.53
Decision Tree f1_score:0.43
Logistic Regression f1_score:0.45
```

LOSS for Logistic Regression:

[71]:
```
print('Logistic Regression LOGLOSS: {:.2f}'.format(log_loss(y_test, LR_predPro)))
```

```
Logistic Regression LOGLOSS: 0.68
```

Discussion:

In the beginning of this Jupyter-notebook, we had categorical data of type 'object'. This is not a data type that we could have fed through a Machine learning, so label encoding was used to created dummy variable as categorical of numerical data type.

After solving that the problem we were left with another imbalanced data. As we see earlier, the class 1 was greater than class 2. The solution to this was to resample the classes with Sklearn's resample tool. We down sampled to match smallest class.
After we analyzed and cleaned the data-set, it was then fed through Machine Learning models,
K-NearestNeighborClassifier, Decision Tree and Logistic Regression. Although the first two are ideal for this project.
Evaluation metrics used to test the accuracy of our models were jaccard index, f-1 score and *Logloss* for logistic regression. Choosing The right k, max depth and hypermeter C=0.1 values helped to improve our accuracy to be the best possible.

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 (code | class 1) or injury (code | class 2).