Capstone Project – Car Accident Severity

Introduction:-

In an effort to reduce the frequency of car collisions in a community, an algorithm must be developed to predict the severity of an accident given the current weather, road and visibility conditions. When conditions are bad, this model will alert drivers to remind them to be more careful.

Problem Understanding:-

To complete capstone, you will be working on a case study which is to predict the severity of an accident. Say you are driving to another city for work or to visit some friends. It is rainy and windy, and on the way, you come across a terrible traffic jam on the other side of the highway. Long lines of cars barely moving. As you keep driving, police car start appearing from a far shutting down the highway. Oh, it is an accident and there's a helicopter transporting the ones involved in the crash to the nearest hospital. They must be in critical condition for all of this to be happening. Now, wouldn't it be great if there is something in place that could warn you, given the weather and the road conditions about the possibility of you getting into a car accident and how severe it would be, so that you would drive more carefully or even change your travel if you are able to. Well, this is exactly what you will be working on in this course.

Data Section :-

Our predictor or target variable 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 : Little to no Probability (Clear Conditions)`
- `1 : Very Low Probability - Chance or Property Damage`
- `2 : Low Probability - Chance of Injury`
- `3 : Mild Probability - Chance of Serious Injury`
- `4 : High Probability - Chance of Fatality`

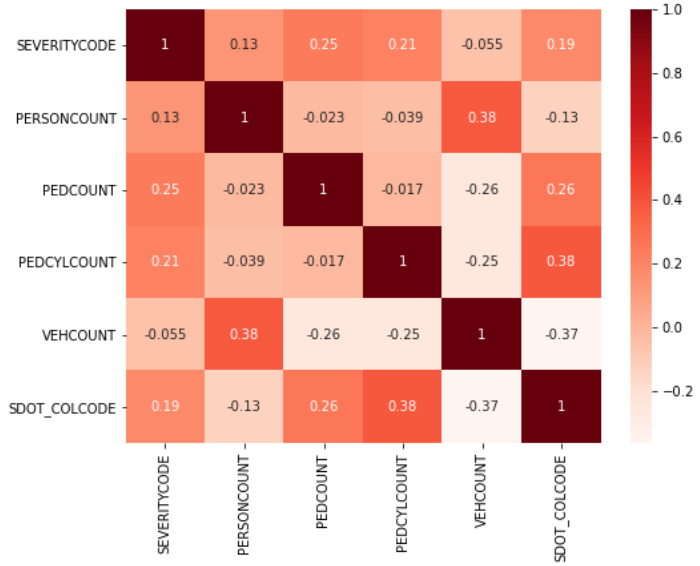
Feature Selection & Label Encoding :-

In it's original form, this data is not fit for analysis. For one, there are many columns that we will not use for this model. Also, most of the features are of type object, when they should be numerical type.

We must use label encoding to covert the features to our desired data type.

Feature selection

```
cor=df.corr()
plt.figure(figsize=(8,6))
sns.heatmap(cor,annot=True,cmap=plt.cm.Reds)
plt.show()
```



Label Encoding

The features are of type object, when they should be categorial type.We must use label encoding to covert the features to our desired data type.

In [13]:
```
features["WEATHER"]=features["WEATHER"].astype('category')
features["ROADCOND"]=features["ROADCOND"].astype('category')
features["LIGHTCOND"]=features["LIGHTCOND"].astype('category')
features.dtypes
```

Out[13]:
```
SEVERITYCODE       int64
WEATHER         category
ROADCOND        category
LIGHTCOND       category
dtype: object
```

In [14]:
```
features["WEATHER_CAT"]=features["WEATHER"].cat.codes
features["ROADCOND_CAT"]=features["ROADCOND"].cat.codes
features["LIGHTCOND_CAT"]=features["LIGHTCOND"].cat.codes
features.head()
```

Out[14]:

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND	WEATHER_CAT	ROADCOND_CAT	LIGHTCOND_CAT
0	2	Overcast	Wet	Daylight	4	8	5
1	1	Raining	Wet	Dark - Street Lights On	6	8	2
2	1	Overcast	Dry	Daylight	4	0	5
3	1	Clear	Dry	Daylight	1	0	5
4	2	Raining	Wet	Daylight	6	8	5

With the new columns, we can now use this data in our analysis and ML models!

Cleaning & Balancing the Dataset :-

We will remove the rows with null values and duplicate values. Our target variable SEVERITYCODE is only 42% balanced. In fact, severitycode in class 1 is nearly three times the size of class 2.

We can fix this by down sampling the majority class.

```
2    57052
1    57052
Name: SEVERITYCODE, dtype: int64
```

Perfectly balanced.

Methodology :-

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

We will use 30% of our data for testing and 70% for training.

Train and Test Split

We will use 30% of our data for testing and 70% for training.

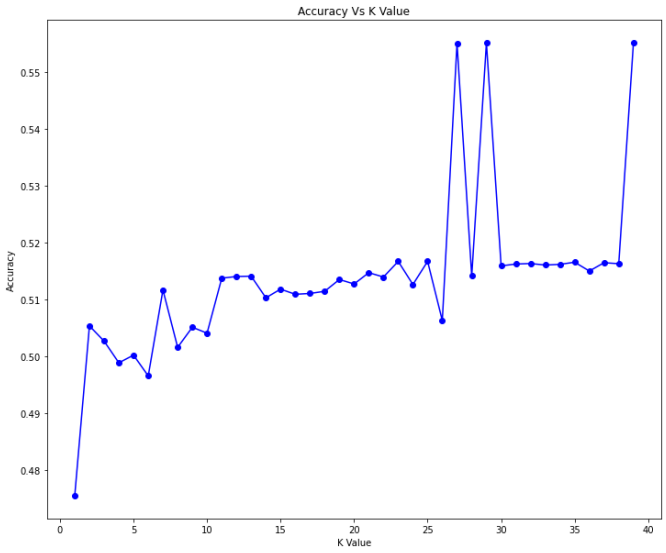
```
In [21]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=10)
         print("train data",x_train.shape,y_train.shape)
         print("test data",x_test.shape,y_test.shape)

         train data (79872, 3) (79872,)
         test data (34232, 3) (34232,)
```

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.




```
In [25]: print("Maximum Accuracy",max(acc),"at k =",acc.index(max(acc))+1)

         Maximum Accuracy 0.5551822855807431 at k = 39
```

```
k=39
neigh = KNeighborsClassifier(n_neighbors=k).fit(x_train,y_train)
pred_y=neigh.predict(x_test)
a_k=accuracy_score(y_test,pred_y)
j_k=jaccard_score(y_test, pred_y)
f1_k=f1_score(y_test, pred_y, average='weighted')
print("accuracy_score",a_k)
print("jaccard_score",j_k)
print("f1_score",f1_k)
```

```
accuracy_score 0.5551822855807431
jaccard_score 0.21639563606422396
f1_score 0.508039428734879
```

Support Vector Machine (SVM)

The algorithm creates a line or a hyperplane which separates the data into classes. In our model, we use SVM to find a separating line (or hyperplane) between data of two classes.

Support Vector Machine (SVM)

```
]: from sklearn import svm
clf=svm.SVC(kernel='rbf')
clf.fit(x_train,y_train)
pred_y2=clf.predict(x_test)
a_s=accuracy_score(y_test,pred_y2)
j_s=jaccard_score(y_test, pred_y2)
f1_s=f1_score(y_test, pred_y2, average='weighted')
print("accuracy_score",a_s)
print("jaccard_score",j_s)
print("f1_score",f1_s)
```

```
accuracy_score 0.5644426267819584
jaccard_score 0.2687591956841589
f1_score 0.5367905000793085
```

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.

Logistic Regression

```
]: from sklearn.linear_model import LogisticRegression
   LR = LogisticRegression(C=0.01, solver='liblinear').fit(x_train,y_train)
   pred_y3 = LR.predict(x_test)

   a_l=accuracy_score(y_test,pred_y3)
   j_l=jaccard_score(y_test, pred_y3)
   f1_l=f1_score(y_test, pred_y3, average='weighted')
   print("accuracy_score",a_l)
   print("jaccard_score",j_l)
   print("f1_score",f1_l)
```

```
accuracy_score 0.5360773545220846
jaccard_score 0.289694963771357
f1_score 0.5242431492424555
```

```
]: from sklearn.metrics import log_loss
   pred_y3_prob = LR.predict_proba(x_test)
   print("LR LogLoss: %.2f" % log_loss(y_test, pred_y3_prob))
```

```
LR LogLoss: 0.68
```

Results :-

Now we will check the accuracy of our models.

Final Report

```
df1 = {'Algorithm': ['KNN', 'SVM', 'LR'],'Accuracy_Score':Accuracy_Score,'Jaccard_Score':Jaccard_Score,'F1_Score':F1_Score}
Report = pd.DataFrame(data=df1, columns=['Algorithm', 'Accuracy_Score', 'Jaccard_Score', 'F1_Score'], index=None)
ReportReport.set_index("Algorithm")
Report
```

5]:

Algorithm	Accuracy_Score	Jaccard_Score	F1_Score
KNN	0.555182	0.216396	0.508039
SVM	0.564443	0.268759	0.536791
LR	0.536077	0.289695	0.524243

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.

After solving that issue, we were presented with another - imbalanced data. As mentioned earlier, class 1 was nearly three times larger than class 2. The solution to this was downsampling the majority class with sklearn's resample tool. We down sampled to match the minority class exactly with 57052 values each.

Once we analysed and cleaned the data, it was then fed through three ML models; K-Nearest Neighbor, SVM and Logistic Regression. Although the first two are ideal for this project, logistic regression made most sense because of its binary nature.

Evaluation metrics used to test the accuracy of our models were Jaccard index, f-1 score and logloss for logistic regression. Choosing different k, max depth and hypermeter C 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 (class 1) or injury (class 2).