ACCIDENT SEVERITY REPORT

Introduction | Business Understanding:

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.

Data Understanding

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:

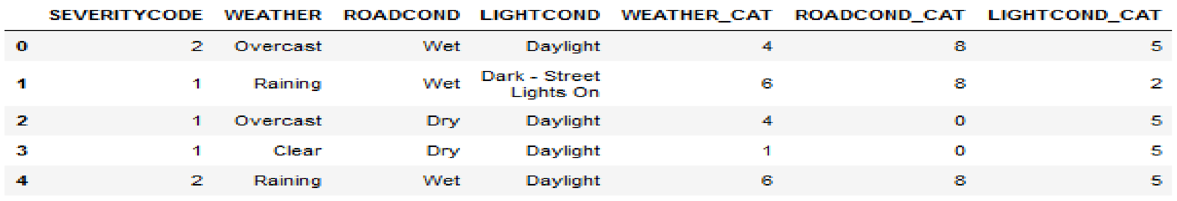
0. Little to no Probability(Clear Conditions)

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.

Extract Dataset & Convert

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.

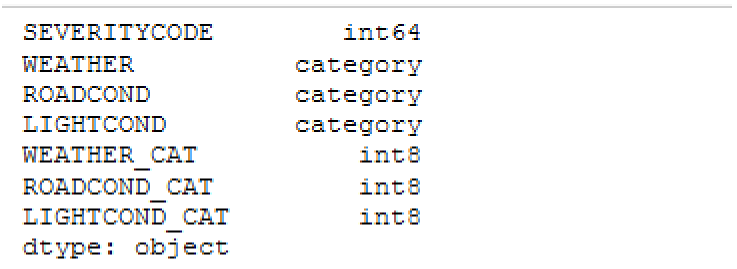
We must use label encoding to covert the features to our desired data type.



	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!

Now let's check the data types of the new columns in our data frame. Moving forward, we will only use the new columns for our analysis.

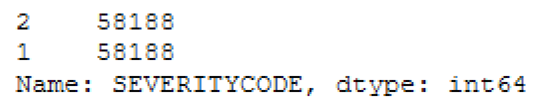


```
SEVERITYCODE        int64
WEATHER          category
ROADCOND         category
LIGHTCOND        category
WEATHER_CAT          int8
ROADCOND_CAT         int8
LIGHTCOND_CAT        int8
dtype: object
```

Balancing the Dataset

Our target variable SEVERITYCODE is only 42% balanced. In fact, severity code in class 1 is nearly three times the size of class 2.

We can fix this by down sampling the majority class.



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

Perfectly balanced.

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:

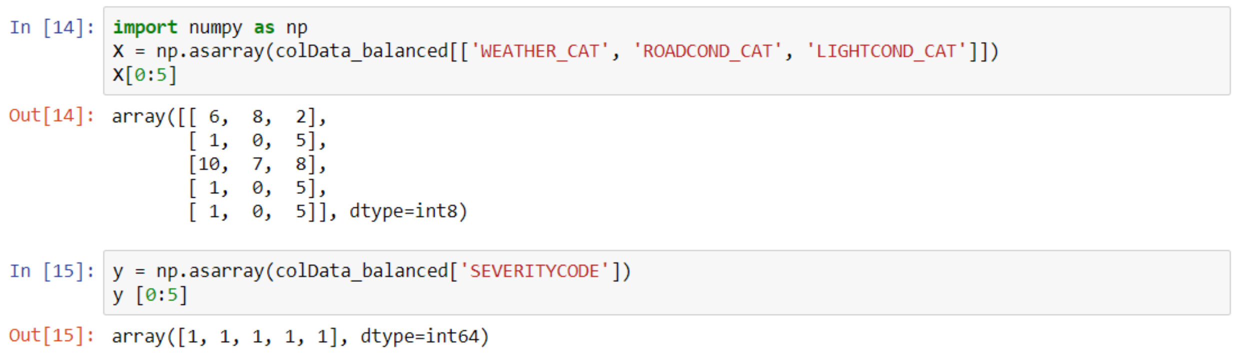
A decision tree model gives us a layout of all possible outcomes so we can fully analyze the consequences of a decision. It context, the decision tree observes all possible outcomes of different weather conditions.

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. Let's get started!

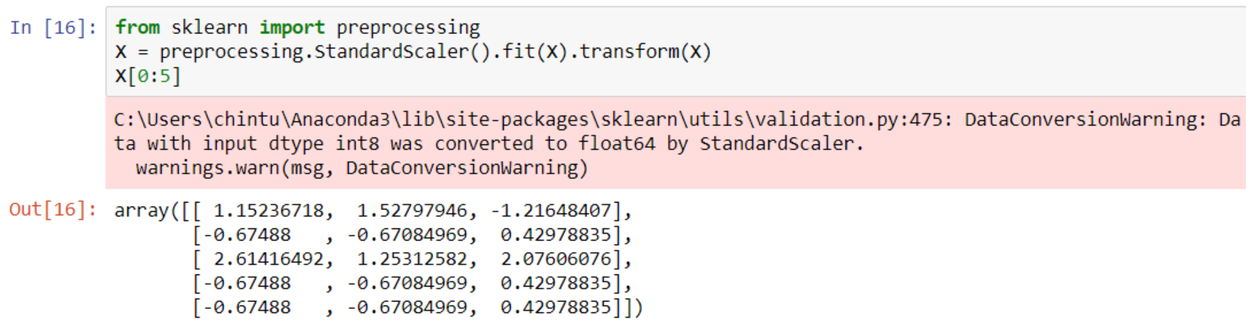
Initialization:

Define X and Y



```
In [14]: import numpy as np
         X = np.asarray(colData_balanced[['WEATHER_CAT', 'ROADCOND_CAT', 'LIGHTCOND_CAT']])
         X[0:5]

Out[14]: array([[ 6,  8,  2],
                [ 1,  0,  5],
                [10,  7,  8],
                [ 1,  0,  5],
                [ 1,  0,  5]], dtype=int8)

In [15]: y = np.asarray(colData_balanced['SEVERITYCODE'])
         y [0:5]

Out[15]: array([1, 1, 1, 1, 1], dtype=int64)
```

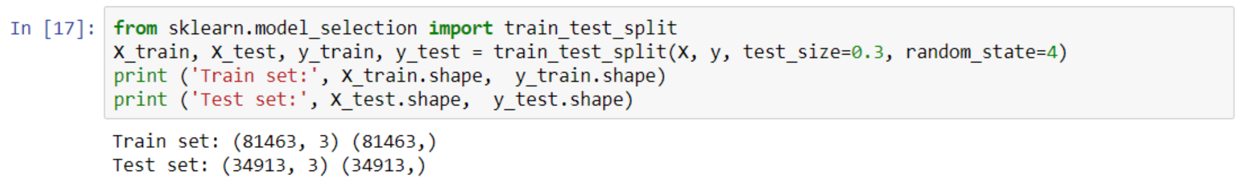
Normalize the dataset:



```
In [16]: from sklearn import preprocessing
         X = preprocessing.StandardScaler().fit(X).transform(X)
         X[0:5]

         C:\Users\chintu\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning: Da
         ta with input dtype int8 was converted to float64 by StandardScaler.
           warnings.warn(msg, DataConversionWarning)

Out[16]: array([[ 1.15236718,  1.52797946, -1.21648407],
                [-0.67488   , -0.67084969,  0.42978835],
                [ 2.61416492,  1.25312582,  2.07606076],
                [-0.67488   , -0.67084969,  0.42978835],
                [-0.67488   , -0.67084969,  0.42978835]])
```

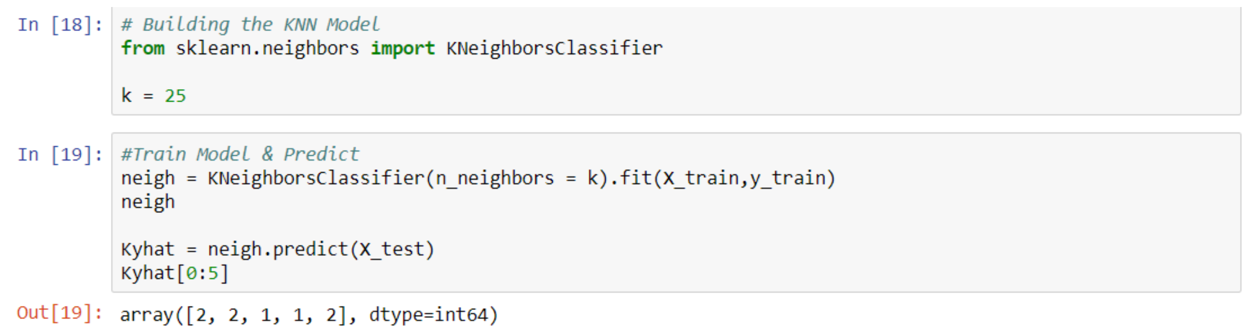
Train/Test Split:

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



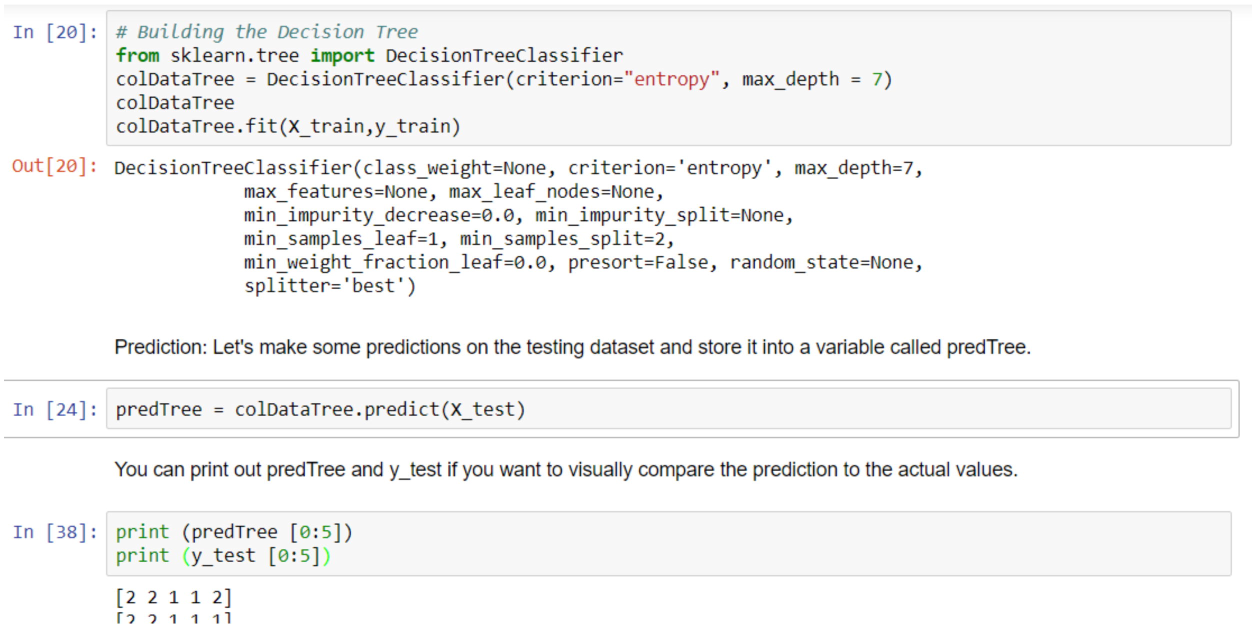
```
In [17]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4)
         print ('Train set:', X_train.shape,  y_train.shape)
         print ('Test set:', X_test.shape,  y_test.shape)

         Train set: (81463, 3) (81463,)
         Test set: (34913, 3) (34913,)
```

K-Nearest Neighbor(KNN):



```
In [18]: # Building the KNN Model
         from sklearn.neighbors import KNeighborsClassifier

         k = 25

In [19]: #Train Model & Predict
         neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
         neigh

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

Out[19]: array([2, 2, 1, 1, 2], dtype=int64)
```

Decision Tree:



```
In [20]: # Building the Decision Tree
         from sklearn.tree import DecisionTreeClassifier
         colDataTree = DecisionTreeClassifier(criterion="entropy", max_depth = 7)
         colDataTree
         colDataTree.fit(X_train,y_train)

Out[20]: DecisionTreeClassifier(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=False, random_state=None,
                     splitter='best')
```

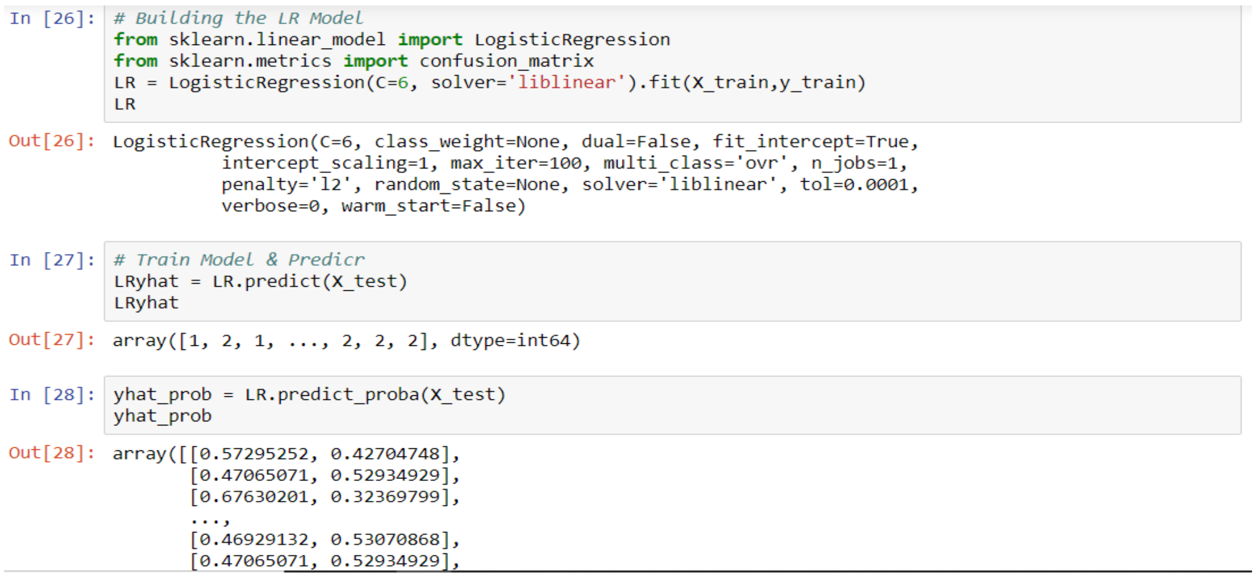
Prediction: Let's make some predictions on the testing dataset and store it into a variable called predTree.

```
In [24]: predTree = colDataTree.predict(X_test)
```

You can print out predTree and y_test if you want to visually compare the prediction to the actual values.

```
In [38]: print (predTree [0:5])
         print (y_test [0:5])

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

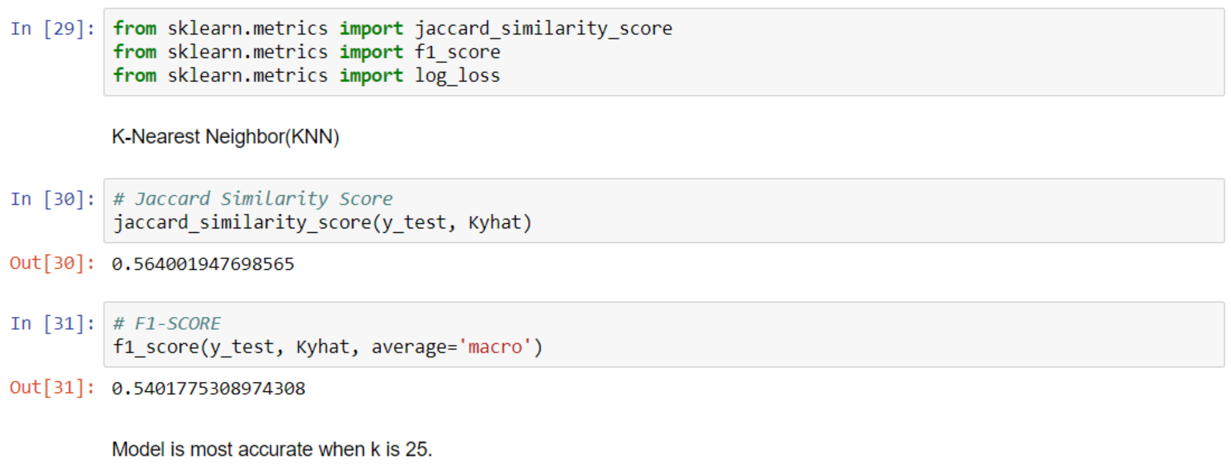
Logistic Regression:



```
In [26]: # Building the LR Model
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import confusion_matrix
         LR = LogisticRegression(C=6, solver='liblinear').fit(X_train,y_train)
         LR

Out[26]: LogisticRegression(C=6, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [27]: # Train Model & Predicr
         LRyhat = LR.predict(X_test)
         LRyhat

Out[27]: array([1, 2, 1, ..., 2, 2, 2], dtype=int64)

In [28]: yhat_prob = LR.predict_proba(X_test)
         yhat_prob

Out[28]: array([[0.57295252, 0.42704748],
                [0.47065071, 0.52934929],
                [0.67630201, 0.32369799],
                ...,
                [0.46929132, 0.53070868],
                [0.47065071, 0.52934929],
```

Results and Evaluation:

Now we will check the accuracy of our models.

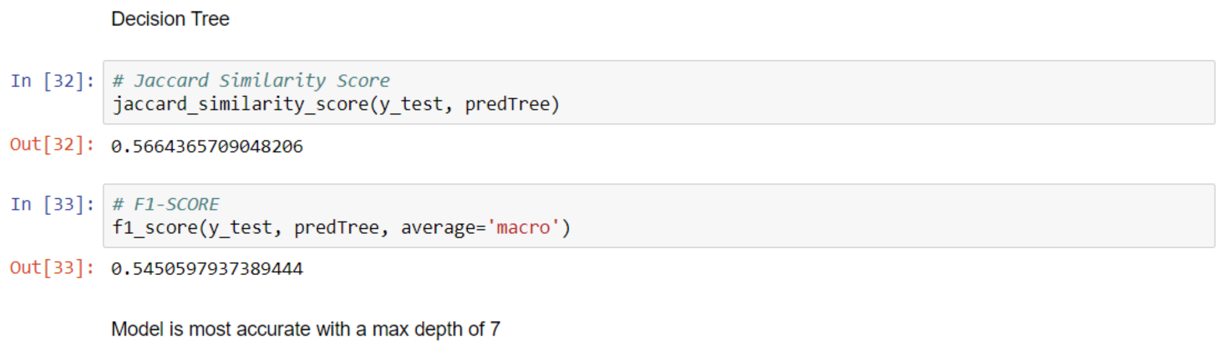


```
In [29]: from sklearn.metrics import jaccard_similarity_score
         from sklearn.metrics import f1_score
         from sklearn.metrics import log_loss
```

K-Nearest Neighbor(KNN)

```
In [30]: # Jaccard Similarity Score
         jaccard_similarity_score(y_test, Kyhat)

Out[30]: 0.564001947698565

In [31]: # F1-SCORE
         f1_score(y_test, Kyhat, average='macro')

Out[31]: 0.5401775308974308
```

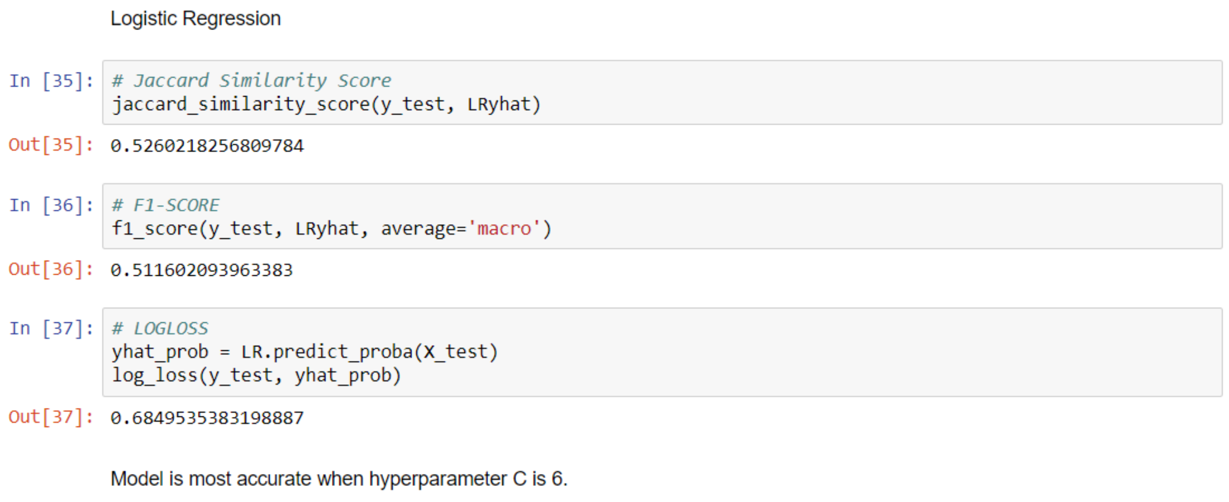
Model is most accurate when k is 25.



Decision Tree

```
In [32]: # Jaccard Similarity Score
         jaccard_similarity_score(y_test, predTree)

Out[32]: 0.5664365709048206

In [33]: # F1-SCORE
         f1_score(y_test, predTree, average='macro')

Out[33]: 0.5450597937389444
```

Model is most accurate with a max depth of 7



Logistic Regression

```
In [35]: # Jaccard Similarity Score
         jaccard_similarity_score(y_test, LRyhat)

Out[35]: 0.5260218256809784

In [36]: # F1-SCORE
         f1_score(y_test, LRyhat, average='macro')

Out[36]: 0.511602093963383

In [37]: # LOGLOSS
         yhat_prob = LR.predict_proba(X_test)
         log_loss(y_test, yhat_prob)

Out[37]: 0.6849535383198887
```

Model is most accurate when hyperparameter C is 6.

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 down sampling the majority class with sklearn's resample tool. We down sampled to match the minority class exactly with 58188 values each.

Once we analyzed and cleaned the data, it was then fed through three ML models; K-Nearest Neighbor, Decision Tree 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 hyparameter 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).