Car Accident Severity

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 Probablility - 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 dataframe. 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, severitycode in class 1 is nearly three times the size of class 2.

We can fix this by downsampling 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 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.

Let's get started!

Initialization

Define X and y

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

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

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

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

Normalize the dataset

```
In [105]: from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validat
ion.py:595: DataConversionWarning: Data with input dtype int8 was converte
d to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validat
ion.py:595: DataConversionWarning: Data with input dtype int8 was converte
d to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

```
Out[105]: 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.

```
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, rand
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,)
```

Here we will begin our modeling and predictions...

K-Nearest Neighbors (KNN)

```
# Building the KNN Model
from sklearn.neighbors import KNeighborsClassifier

k = 25
```

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

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

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

Decision Tree

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

```
.1]: 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=Non
     e,
                 splitter='best')
```

```
# Train Model & Predict
DTyhat = colDataTree.predict(X_test)
print (predTree [0:5])
print (y_test [0:5])
```

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

Logistic Regression

```
# 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
```

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

```
# Train Model & Predicr
LRyhat = LR.predict(X_test)
LRyhat
```

```
245]: array([1, 2, 1, ..., 2, 2, 2])
```

```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

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

Results & Evaluation

Now we will check the accuracy of our models.

K-Nearest Neighbor

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

[197]: 0.564001947698565

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

[198]: 0.5401775308974308

Model is most accurate when k is 25.

Decision Tree

]:
```
# Jaccard Similarity Score
jaccard_similarity_score(y_test, DTyhat)
```

[213]: 0.5664365709048206

]:
```
# F1-SCORE
f1_score(y_test, DTyhat, average='macro')
```

[214]: 0.5450597937389444

Model is most accurate with a max depth of 7.

Logistic Regression

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

[247]: 0.5260218256809784

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

[248]: 0.511602093963383

]:
```
# LOGLOSS
yhat_prob = LR.predict_proba(X_test)
log_loss(y_test, yhat_prob)
```

[249]: 0.6849535383198887

Model is most accurate when hyperparameter C is 6.