IBM Data Science Capstone: Car Accident Severity Report

Ibrahem Nofal

Introduction | Business Understanding

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 its 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.

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

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

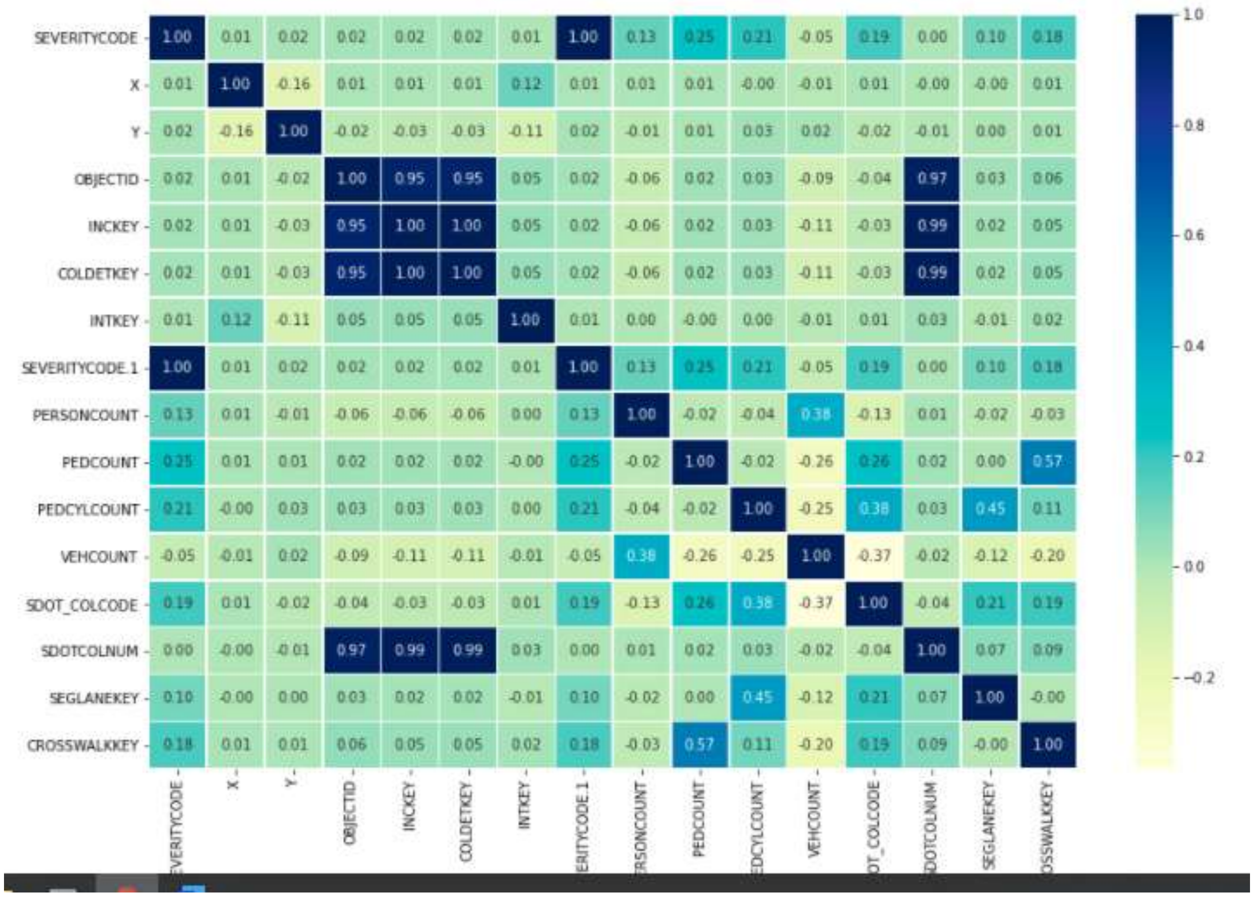
EDA

Correlation between independent variables

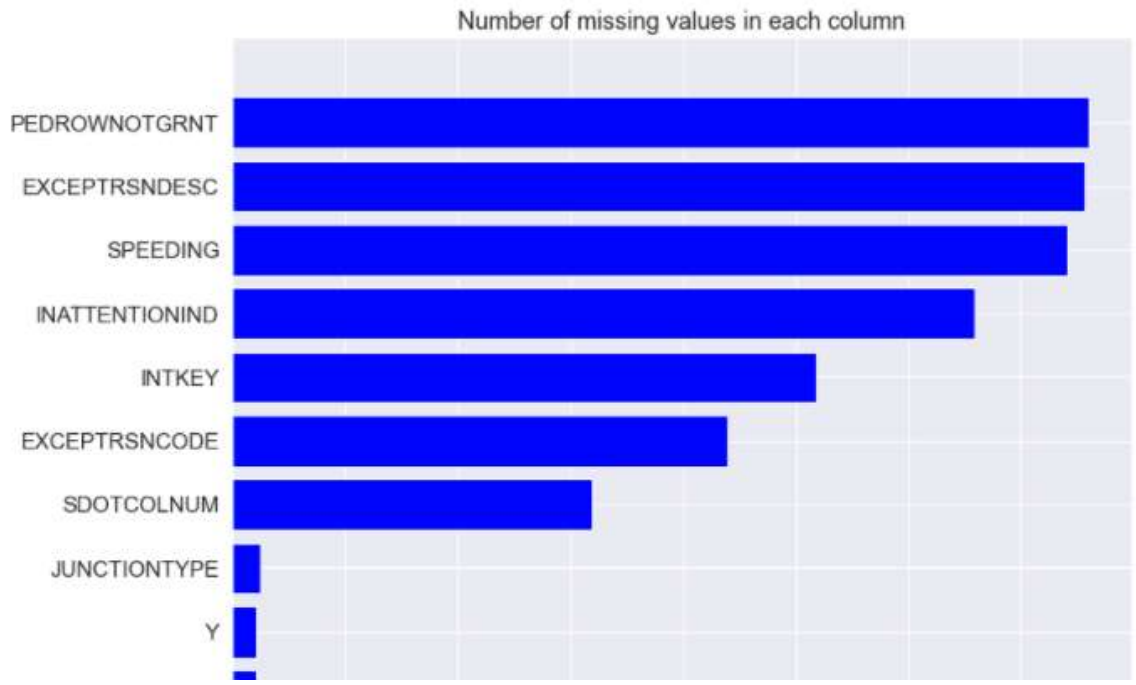
Finally, we'll compare all of the independent variables in one hit.

Why?

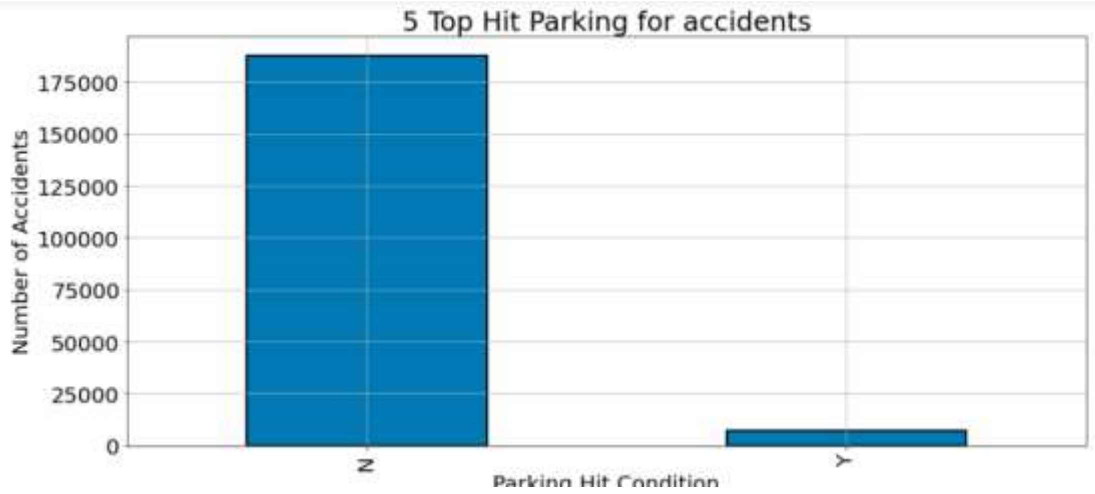
Because this may give an idea of which independent variables may or may not have an impact on our target variable.



Number of missing values in each columns



Hit Parking

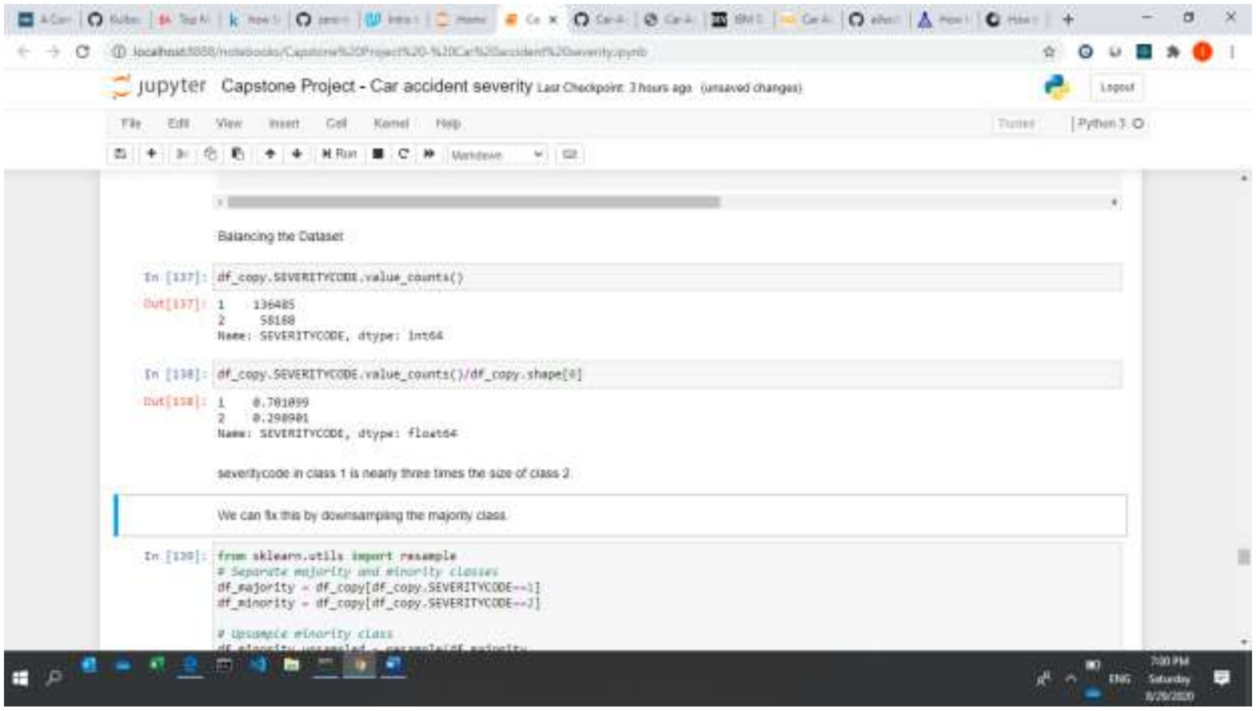


And more………

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.


jupyter Capstone Project - Car accident severity Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Help
Python 3

Balancing the Dataset

```
In [137]: df_copy.SEVERITYCODE.value_counts()
Out[137]: 1    136485
          2     58188
          Name: SEVERITYCODE, dtype: int64

In [138]: df_copy.SEVERITYCODE.value_counts()/df_copy.shape[0]
Out[138]: 1    0.701099
          2    0.298901
          Name: SEVERITYCODE, dtype: float64
```

severitycode in class 1 is nearly three times the size of class 2

We can fix this by downsampling the majority class.

```
In [139]: from sklearn.utils import resample
          # Separate majority and minority classes
          df_majority = df_copy[df_copy.SEVERITYCODE==1]
          df_minority = df_copy[df_copy.SEVERITYCODE==2]

          # Upsample minority class
```

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 like 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. Its context, the decision tree observes all possible outcomes of different weather conditions.

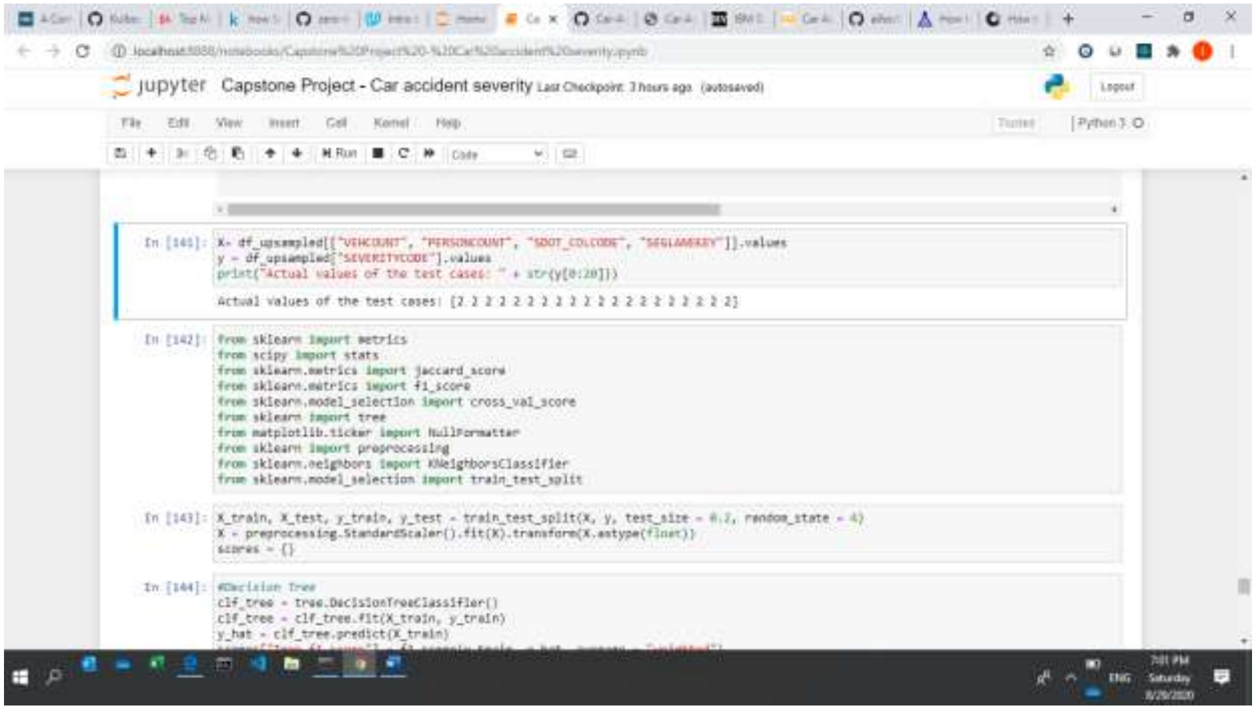
Logistic Regression

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 [141]: X= df_upsampled[["VEHCOUNT", "PERSONCOUNT", "SDOT_COLCODE", "SEGLANEKEY"]].values
          y = df_upsampled["SEVERITYCODE"].values
          print("Actual values of the test cases: " + str(y[0:20]))

          Actual values of the test cases: [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

In [142]: from sklearn import metrics
          from scipy import stats
          from sklearn.metrics import jaccard_score
          from sklearn.metrics import f1_score
          from sklearn.model_selection import cross_val_score
          from sklearn import tree
          from matplotlib.ticker import NullFormatter
          from sklearn import preprocessing
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.model_selection import train_test_split

In [143]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 4)
          X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
          scores = {}

In [144]: #Decision Tree
          clf_tree = tree.DecisionTreeClassifier()
          clf_tree = clf_tree.fit(X_train, y_train)
          y_hat = clf_tree.predict(X_train)
```

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])
```

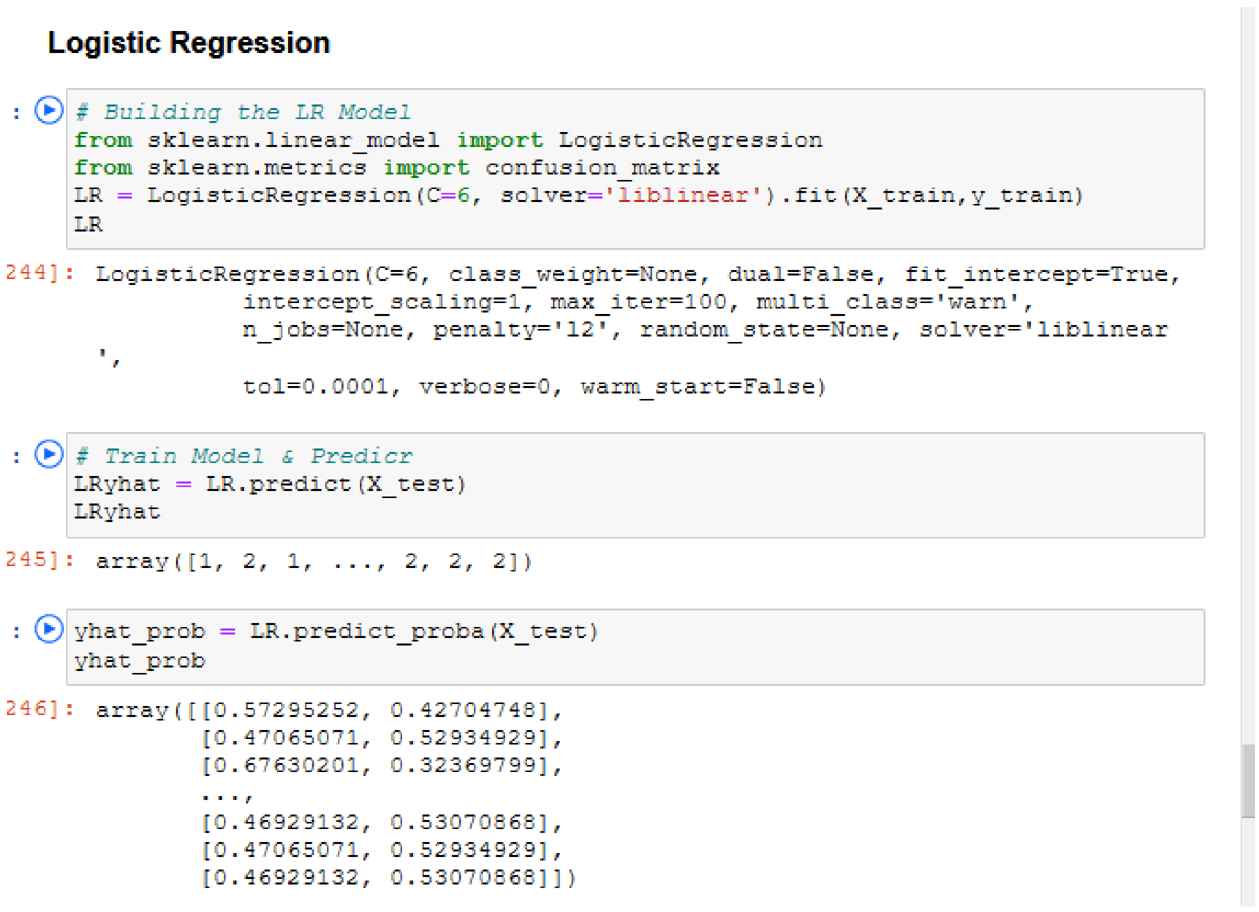
Decision Tree

```
# 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
```

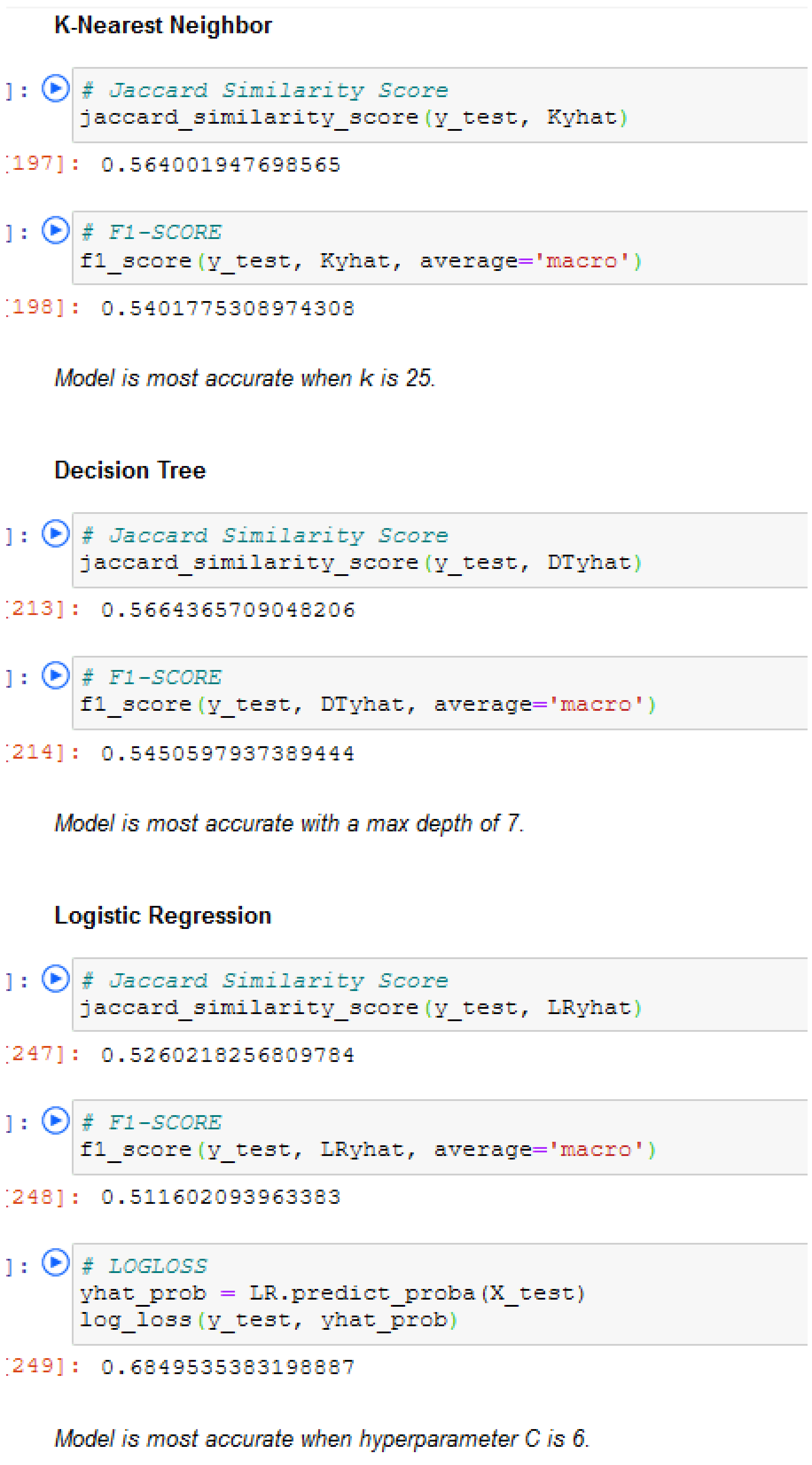
```
245]: array([1, 2, 1, ..., 2, 2, 2])
```

```
: 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.

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 downsampled 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 accident conditions have a somewhat impact could result in property damage (class 1) or injury (class 2).