Predicting Car Accident Severity.

Adnane Aabbar

September 07, 2020

Table of Contents

Introduction & Business Understanding:

Description of the problem:

In order to reduce the frequency of car crashes in the community, an algorithm must be developed to predict accident severity given the current weather, road and light conditions.

The main goal of this study is to prevent accidents when conditions are bad, that’s why this model is meant to alert drivers to remind them to be a little bit more careful, to switch roads or postpone their car rides.

There is another possible targeted audience for this study, the local police, health institutes, insurance companies etc. They can make good use of this model to know when to be fully ready to receive bad news about a specific road, and more importantly take prevention measures to avoid accidents on certain ones.

Discussion of the background:

In most cases, carelessness while driving, using drugs and alcohol or driving too fast are some of the main causes of accidents that can be avoided by implementing stronger regulations.

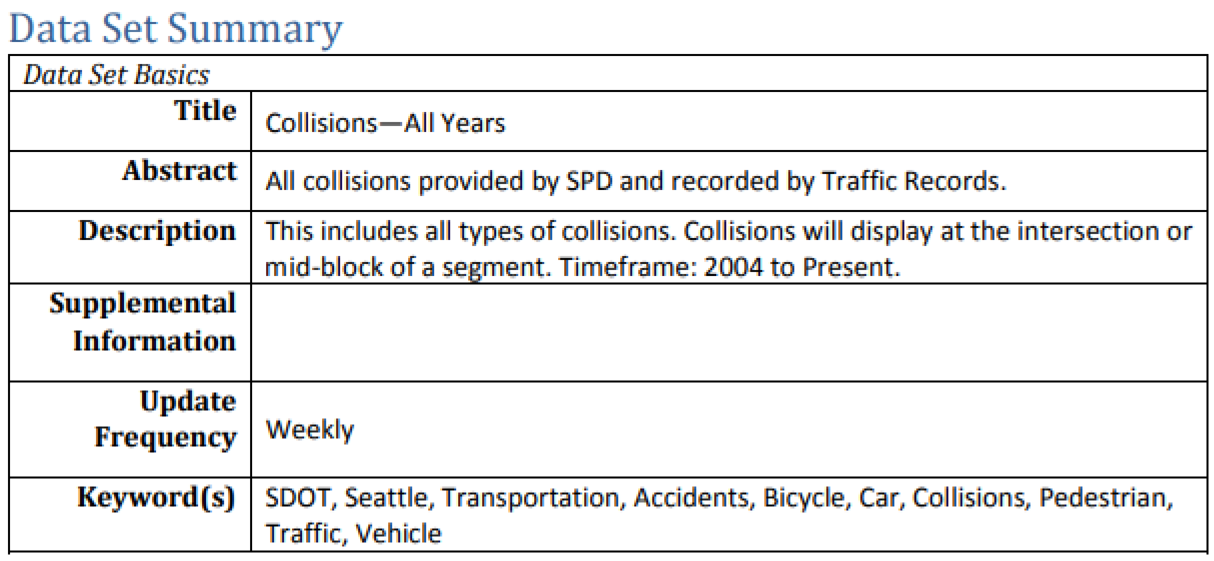
Besides the above reasons, weather, visibility or road conditions are the major uncontrollable factors which can be avoided by uncovering patterns hidden in the data and declaring a warning to local government, police and drivers on the roads. targeted routes, or alerting the drivers before the road trips.

Data Understanding:

Source of the data:

The data used in this project is the example dataset provided in the course. Including all types of collisions.

It concerns the city of Seattle, WA. It is provided by Seattle Police Dept. and recorded by Traffic Records, with a timeframe of: 2004 to Present.



Data Set Summary

Data Set Basics	
Title	Collisions—All Years
Abstract	All collisions provided by SPD and recorded by Traffic Records.
Description	This includes all types of collisions. Collisions will display at the intersection or mid-block of a segment. Timeframe: 2004 to Present.
Supplemental Information	
Update Frequency	Weekly
Keyword(s)	SDOT, Seattle, Transportation, Accidents, Bicycle, Car, Collisions, Pedestrian, Traffic, Vehicle

Description of the data:

The data consists of 37 independent variables and 194,673 rows. The dependent variable, “SEVERITYCODE”, contains numbers that correspond to different levels of severity caused by an accident from 0 to 4.

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

Furthermore, because of the existence of a huge unbalance in some attributes occurrences and the existence of null values in some records, the data needs to be preprocessed, cleaned and balanced before any further processing.

How the data will be used to solve the problem:

We have to select the most important features to weigh the severity of accidents in Seattle. Among all the features, the following features have the most influence in the accuracy of the predictions:

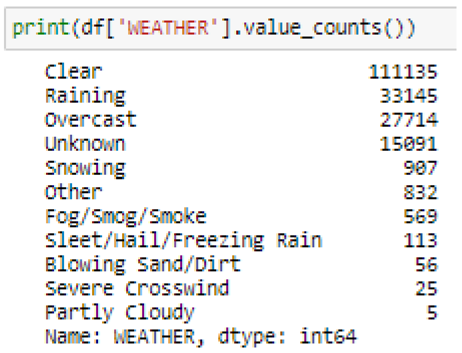
The **'WEATHER', 'ROADCOND'** and **'LIGHTCOND'** attributes.

Preprocessing:

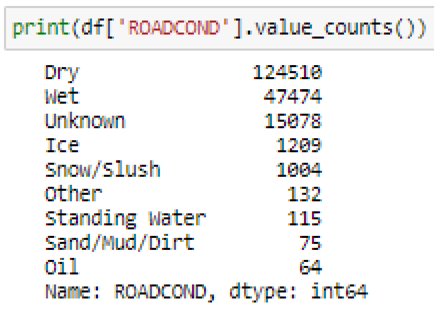
Let’s count the missing values of the attribute columns that we are using to weigh the severity of the collisions we’re going to study.

```
WEATHER          5081
ROADCOND         5012
LIGHTCOND        5170
```

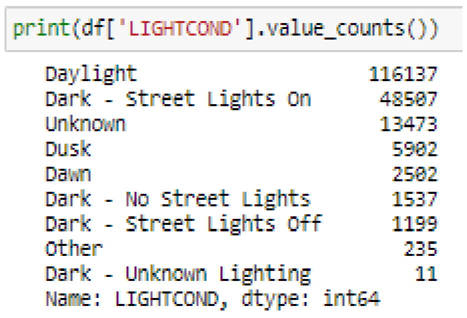
Right now, the columns are of type object, let’s count their values:



```
print(df['WEATHER'].value_counts())

Clear                        111135
Raining                       33145
Overcast                      27714
Unknown                       15091
Snowing                         907
Other                           832
Fog/Smog/Smoke                  569
Sleet/Hail/Freezing Rain        113
Blowing Sand/Dirt                56
Severe Crosswind                 25
Partly Cloudy                     5
Name: WEATHER, dtype: int64
```

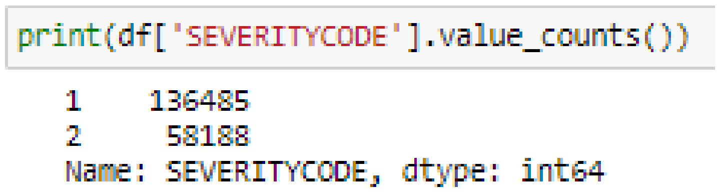


```
print(df['ROADCOND'].value_counts())

Dry               124510
Wet                47474
Unknown            15078
Ice                 1209
Snow/Slush          1004
Other                132
Standing Water       115
Sand/Mud/Dirt         75
Oil                   64
Name: ROADCOND, dtype: int64
```



```
print(df['LIGHTCOND'].value_counts())

Daylight                    116137
Dark - Street Lights On      48507
Unknown                      13473
Dusk                          5902
Dawn                          2502
Dark - No Street Lights       1537
Dark - Street Lights Off      1199
Other                          235
Dark - Unknown Lighting         11
Name: LIGHTCOND, dtype: int64
```

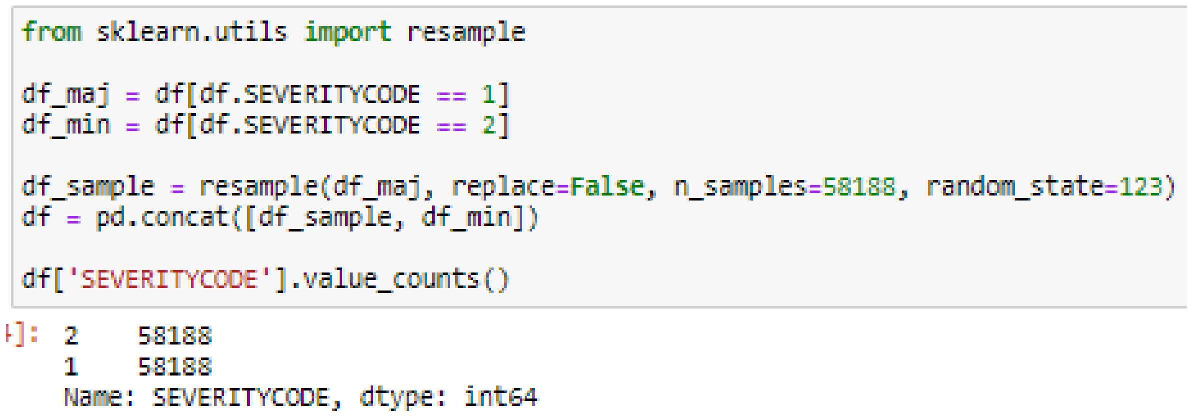
Let’s study the balance of our dataset, let’s check if the samples in the SEVERITYCODE column, have nearly equal value count.



```
print(df['SEVERITYCODE'].value_counts())
```

```
1    136485
2     58188
Name: SEVERITYCODE, dtype: int64
```

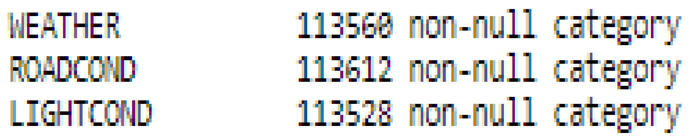
The class 1 is nearly 3 times the number of rows of class 2, let’s fix the unbalance in the SEVERITYCODE column by down sampling the data to only 58188 of each class.



```
from sklearn.utils import resample

df_maj = df[df.SEVERITYCODE == 1]
df_min = df[df.SEVERITYCODE == 2]

df_sample = resample(df_maj, replace=False, n_samples=58188, random_state=123)
df = pd.concat([df_sample, df_min])

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

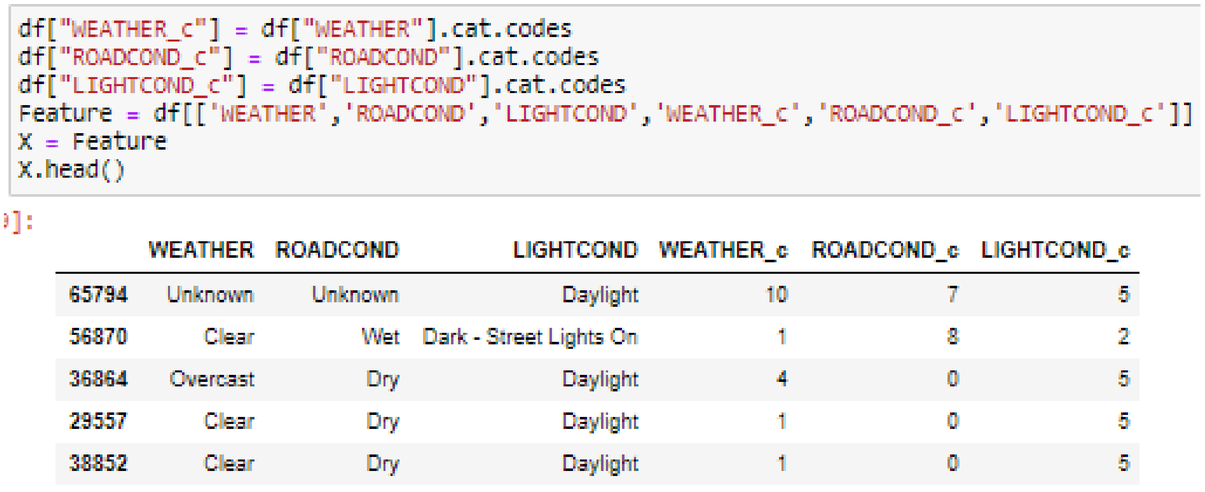
```
]: 2    58188
   1    58188
   Name: SEVERITYCODE, dtype: int64
```

In order to work with our features as categorical values, we should change the type of the columns, and we will add 3 other columns containing the label encoding of categories in the 3 column attributes.



```
WEATHER          113560 non-null category
ROADCOND         113612 non-null category
LIGHTCOND        113528 non-null category
```

Defining then our Feature DataFrame that we will work with for the rest of the study

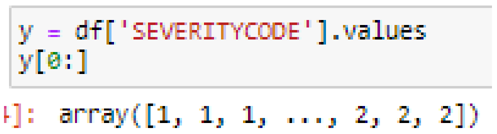


```
df["WEATHER_c"] = df["WEATHER"].cat.codes
df["ROADCOND_c"] = df["ROADCOND"].cat.codes
df["LIGHTCOND_c"] = df["LIGHTCOND"].cat.codes
Feature = df[['WEATHER','ROADCOND','LIGHTCOND','WEATHER_c','ROADCOND_c','LIGHTCOND_c']]
X = Feature
X.head()
```

]:

	WEATHER	ROADCOND	LIGHTCOND	WEATHER_c	ROADCOND_c	LIGHTCOND_c
65794	Unknown	Unknown	Daylight	10	7	5
56870	Clear	Wet	Dark - Street Lights On	1	8	2
36864	Overcast	Dry	Daylight	4	0	5
29557	Clear	Dry	Daylight	1	0	5
38852	Clear	Dry	Daylight	1	0	5

Let’s now define our labels array.



```
y = df['SEVERITYCODE'].values
y[0:]
```

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

Methodology:

Tools and Technologies:

To help implement the solution, I have used a **Github repository** and running **Jupyter Notebook** from the **IBM WATSON Studio** to preprocess data and build Machine Learning models.

Exploratory Data Analysis:

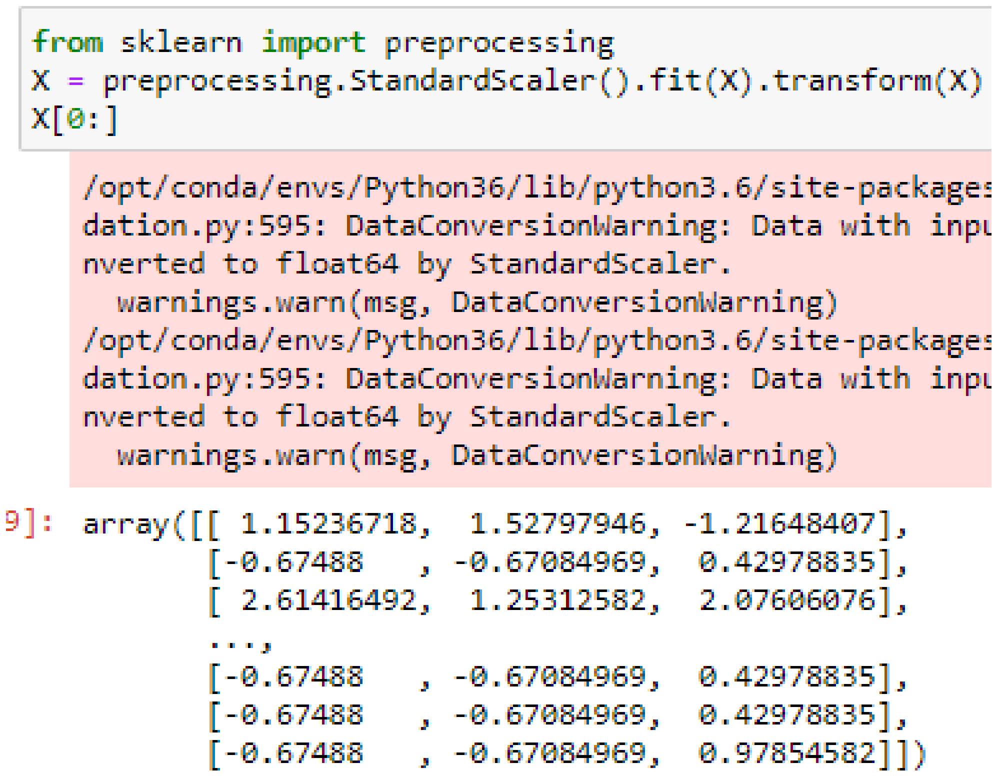
For the **exploratory data analysis**, it was all done in the previous section, where we uncovered flaws in the dataset, unbalance, missing values.

Building the models:

When it comes to coding, I have used **Python** and its popular packages such as **Pandas, NumPy and Sklearn.**

After balancing SEVERITYCODE feature, and standardizing the input feature, the data has been ready for building machine learning models.

Normalizing Data:

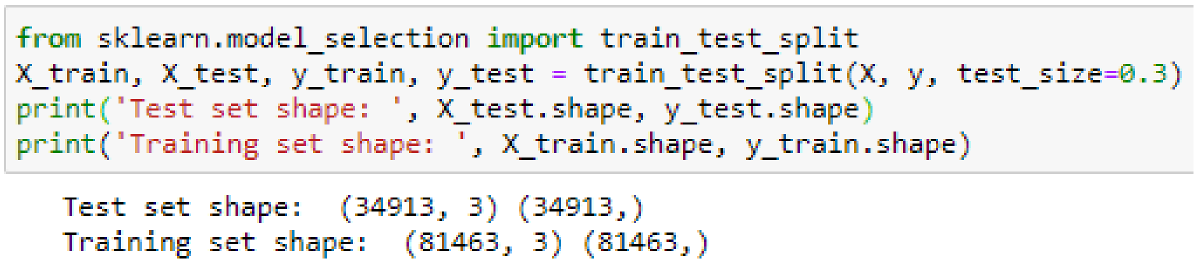


```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:]
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages
dation.py:595: DataConversionWarning: Data with inpu
nverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/opt/conda/envs/Python36/lib/python3.6/site-packages
dation.py:595: DataConversionWarning: Data with inpu
nverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

```
9]: 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],
           [-0.67488   , -0.67084969,  0.97854582]])
```

Train/Test Splitting of the Data:



```
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)
print('Test set shape: ', X_test.shape, y_test.shape)
print('Training set shape: ', X_train.shape, y_train.shape)
```

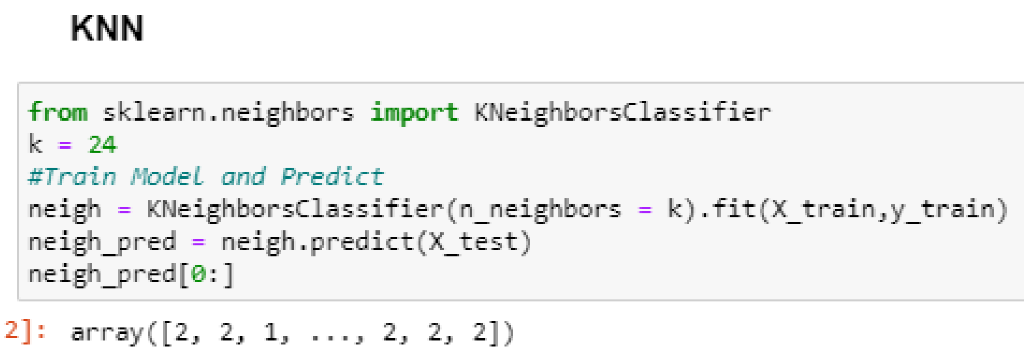
```
Test set shape:  (34913, 3) (34913,)
Training set shape:  (81463, 3) (81463,)
```

Classification Models:

I have employed three of the machine learning models for classification that have been studied in the last course of this specialization:

- K-Nearest Neighbors (KNN)
- Decision Tree
- Linear Regression

K-Nearest Neighbors :

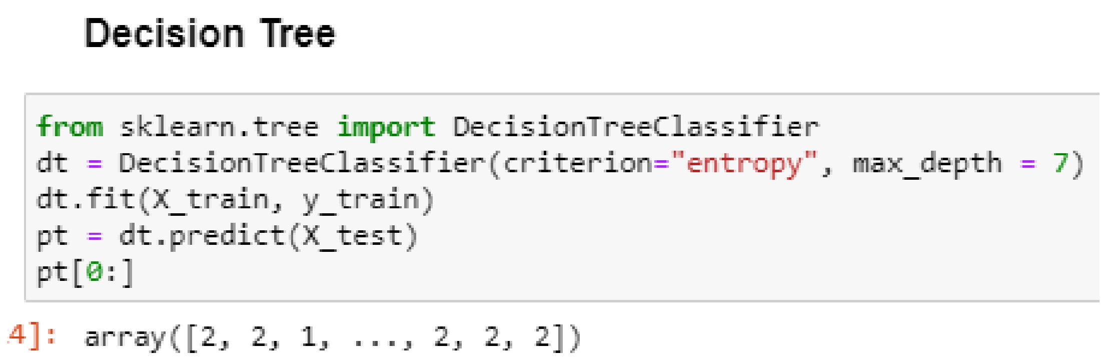


KNN

```
from sklearn.neighbors import KNeighborsClassifier
k = 24
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh_pred = neigh.predict(X_test)
neigh_pred[0:]
```

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

Decision Tree:

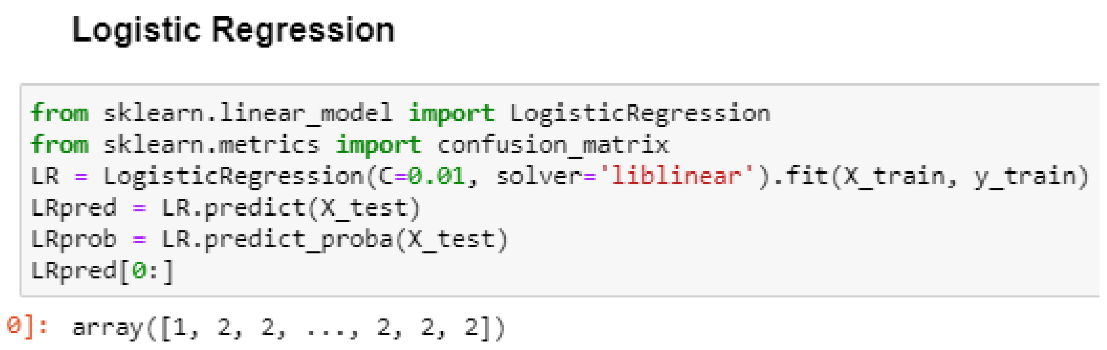


Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion="entropy", max_depth = 7)
dt.fit(X_train, y_train)
pt = dt.predict(X_test)
pt[0:]
```

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

Logistic Regression:



Logistic Regression

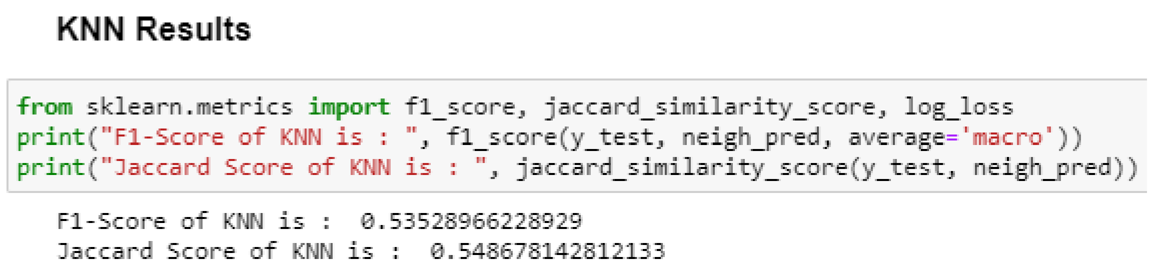
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train, y_train)
LRpred = LR.predict(X_test)
LRprob = LR.predict_proba(X_test)
LRpred[0:]
```

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

Results & Evaluation:

Results:

K-Nearest Neighbors :



KNN Results

```python
from sklearn.metrics import f1_score, jaccard_similarity_score, log_loss
print("F1-Score of KNN is : ", f1_score(y_test, neigh_pred, average='macro'))
print("Jaccard Score of KNN is : ", jaccard_similarity_score(y_test, neigh_pred))
```

```
F1-Score of KNN is :  0.53528966228929
Jaccard Score of KNN is :  0.548678142812133
```

Decision Tree:

Decision Tree Results

```
print("F1-Score of Decision Tree is : ", f1_score(y_test, pt, average='macro'))
print("Jaccard Score of Decision Tree is : ", jaccard_similarity_score(y_test, pt))
```

```
F1-Score of Decision Tree is :  0.5371512827050612
Jaccard Score of Decision Tree is :  0.5601351931945121
```

Logistic Regression:

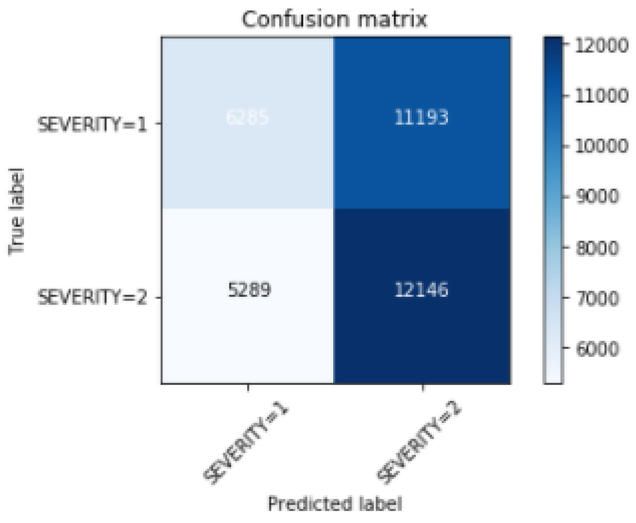
Logistic Regression Results

```
print("F1-Score of Logistic Regression is : ", f1_score(y_test, LRpred, average='macro'))
print("Jaccard Score of Logistic Regression is : ", jaccard_similarity_score(y_test, LRpred))
print("LogLoss of Logistic Regression is : ", log_loss(y_test, LRprob))
```

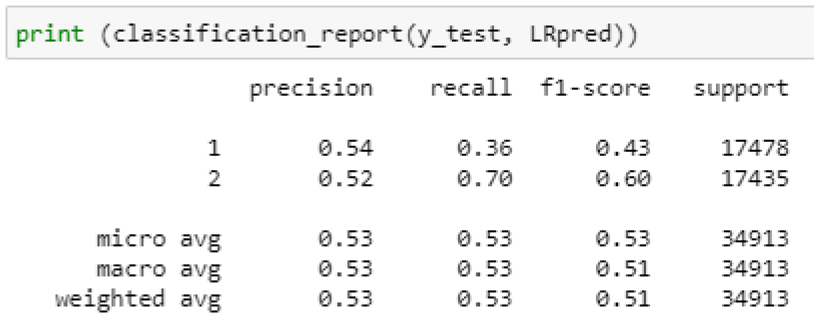
```
F1-Score of Logistic Regression is :  0.5142221323868076
Jaccard Score of Logistic Regression is :  0.527912238994071
LogLoss of Logistic Regression is :  0.6839592390386835
```

Confusion Matrix

```
Confusion matrix, without normalization
[[ 6285 11193]
 [ 5289 12146]]
```



Classification Report



```
print (classification_report(y_test, LRpred))
```

```
              precision    recall  f1-score   support

           1       0.54      0.36      0.43     17478
           2       0.52      0.70      0.60     17435

   micro avg       0.53      0.53      0.53     34913
   macro avg       0.53      0.53      0.51     34913
weighted avg       0.53      0.53      0.51     34913
```

Evaluation:

	F1-score	Jaccard-score	Log Loss
KNN	0.51	0.52	NA
Decision Tree	0.54	0.56	NA
Logistic Regression	0.52	0.53	0.68

Based on the results table, Decision Tree is the best model to predict car accident severity.

Discussion:

At the beginning of this, we had a dataset that had a lot of features of type object, we did convert the columns into category type, then we encoded the labels to contain numerical values.

Once we converted our data to the proper type that can be fed to our models, we faced the problem of unbalanced dataset, we fixed this issue through resampling the dataset.

Due to the binary aspect of the 'SEVERITYCODE' attribute we want to predict(only classes 1 & 2 were in this dataset), a Logistic Regression model was the first intuitive solution, but we also built the K-Nearest Neighbors and Decision Tree models to have more results, contrary to what we expected, the Decision Tree model had better F1-score and Jaccard-score.

We can still improve the above models, by better tuning of the hyperparameters like the **"k"** in KNN, the **"max_depth"** in the Decision Tree, and the **"C"** parameter in the Logistic Regression.

Conclusion:

Based on historical data from the collision in Seattle, we can conclude that particular weather, road and light conditions have an impact on whether or not the car ride could result in one of the two classes property damage (class 1) or injury (class 2).