# Coursera Capstone Car Accident Severity Classification Project

Melissa Melancon
September 2020

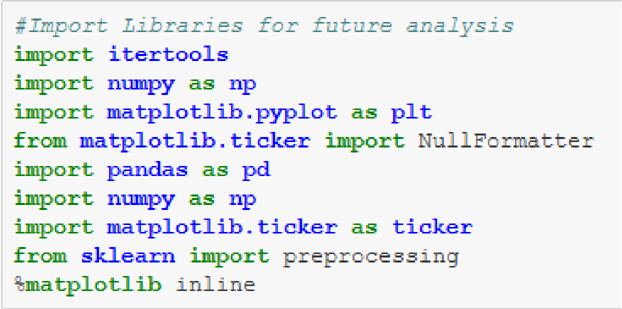
## Introduction and Business Understanding

According to the National Highway Traffic Safety Administration [1], more than 6 million car accidents have occurred across the United States per year resulting in the death of more than 36,500 people and injuring more than 2.7 million people. These figures are astounding when you consider than many accidents could be prevented through improved driver safety or road conditions. This project aims to address possible ways to detect the severity of a crash using data from Seattle, WA. By analyzing several factors in the crashes, we may be able to better predict the likelihood of a severe crash that could cause injury or result in a fatality. Using machine learning techniques, we can model multiple crash factors to see which may have the greatest ability to predict accident severity. The benefit of this modelling could result in improved road conditions, training and education, or even distribution of police and safety teams during adverse conditions. Police departments, insurance agencies, and drivers could all benefit from this type of analysis to better understand how conditions impact accident severity in the hopes of reducing not only accidents, but also the severity of accidents.

## Data Understanding

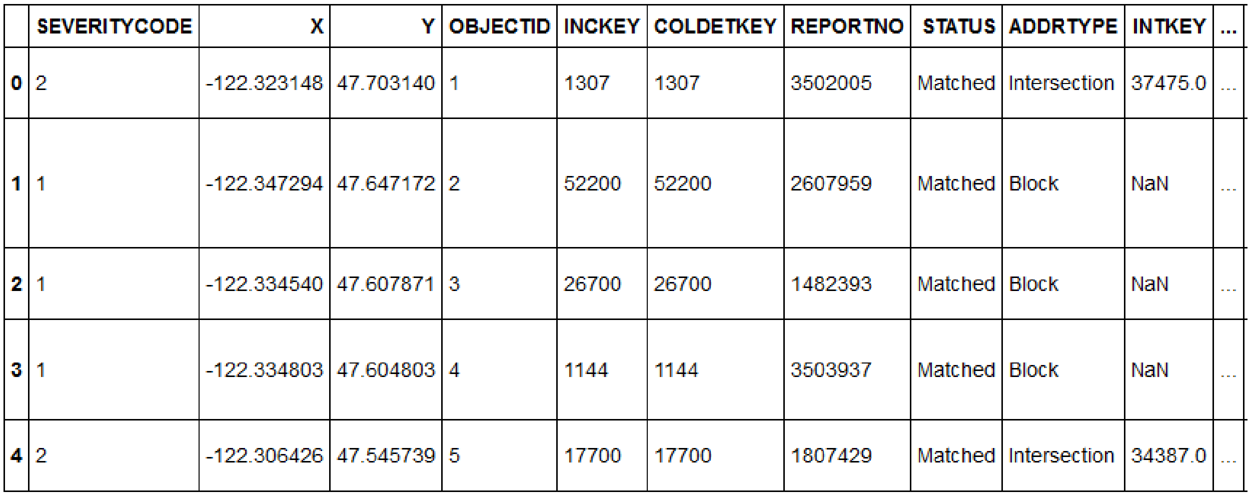
The data in this project specifically addresses crashed in Seattle, WA from 2004 to present [2] but could be modelled using similar data form other US cities. Coursera provided the data via a .csv file which then I translated into a pandas dataframe in Python. For the project, I used Watson Studio to create Jupyter Notebook where I performed the data cleanup and modelling. The data cleanup included reviewing the data cells that have the greatest impact on the reported car accidents. For example, found that weather conditions and road conditions had a high correlation to the severity. In order to process the data and create models, I needed to remove some of the 38 columns and 194,673 rows. Removing unnecessary cells allowed for a simpler dataframe which can then trained the models. At the onset of the project, I decided I would like to use three classifier regression algorithms: K-Nearest Neighbors, Decision Tree, and Support Vector Machine to help determine which model may be the best predictor of crash severity (via Jaccard index and F1 scores).

After importing the appropriate Python libraries, I installed the data provided from the Cousera link to then start building the dataframe that I would use for the project.



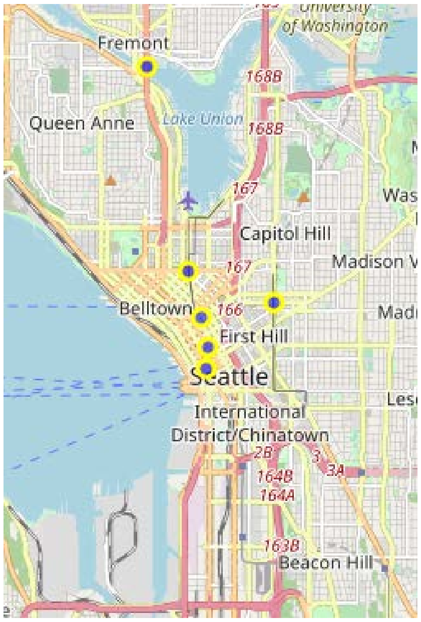
```
#Import Libraries for future analysis
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

Here is the very first header of 5 rows from the raw data file that I created into the dataframe:



| | SEVERITYCODE | X | Y | OBJECTID | INCKEY | COLDETKEY | REPORTNO | STATUS | ADDRTYPE | INTKEY | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | -122.323148 | 47.703140 | 1 | 1307 | 1307 | 3502005 | Matched | Intersection | 37475.0 | ... |
| **1** | 1 | -122.347294 | 47.647172 | 2 | 52200 | 52200 | 2607959 | Matched | Block | NaN | ... |
| **2** | 1 | -122.334540 | 47.607871 | 3 | 26700 | 26700 | 1482393 | Matched | Block | NaN | ... |
| **3** | 1 | -122.334803 | 47.604803 | 4 | 1144 | 1144 | 3503937 | Matched | Block | NaN | ... |
| **4** | 2 | -122.306426 | 47.545739 | 5 | 17700 | 17700 | 1807429 | Matched | Intersection | 34387.0 | ... |

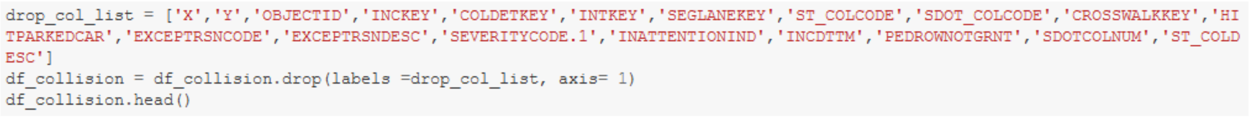
Here is a sample set of the accident locations using the “Folium” tool suite (this is just for illustrative purposes).



## Data Preparation

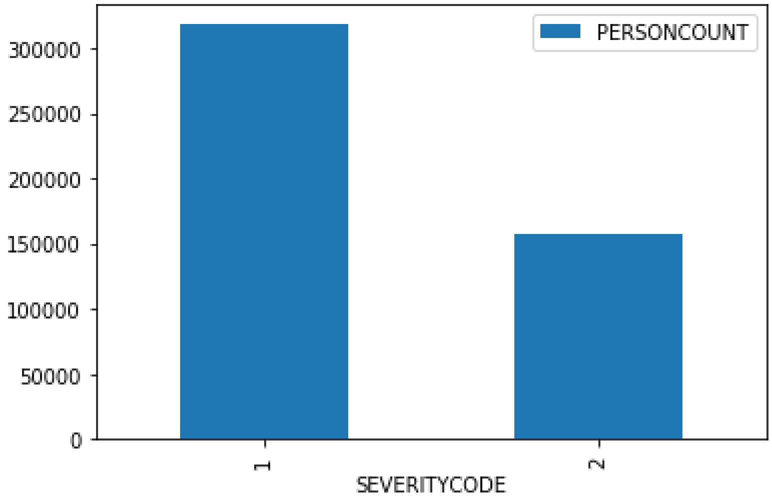
During data preparation, I reviewed the data populated across the 38 different columns. As with any data set, not all the columns yielded useful data with which to address the problem – creating a classification system to determine if select criteria can determine the severity of an accident.

Here is an example of one of the ways I dropped these less significant columns in the dataframe:

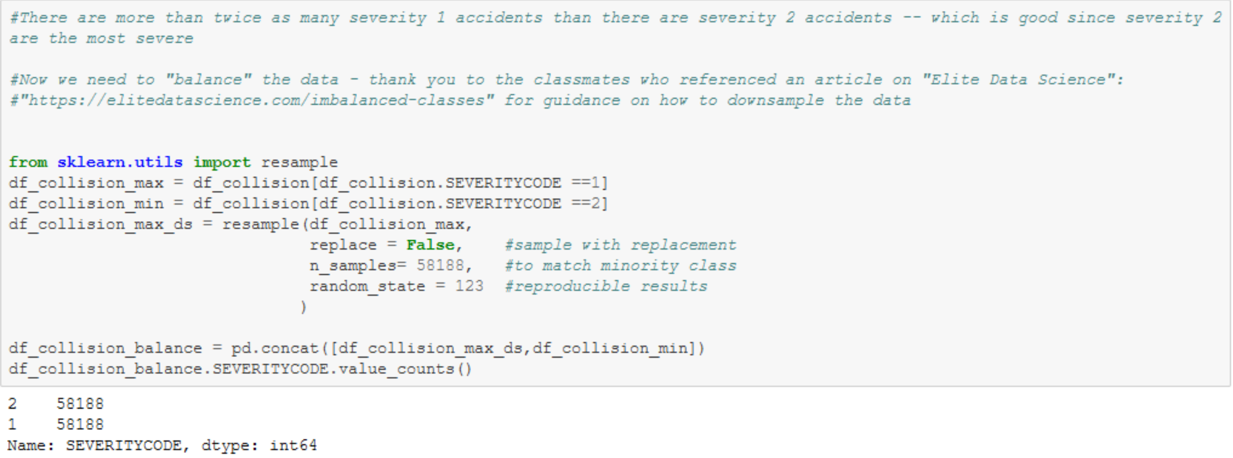


```
drop_col_list = ['X','Y','OBJECTID','INCKEY','COLDETKEY','INTKEY','SEGLANEKEY','ST_COLCODE','SDOT_COLCODE','CROSSWALKKEY','HI
TPARKEDCAR','EXCEPTRSNCODE','EXCEPTRSNDESC','SEVERITYCODE.1','INATTENTIONIND','INCDTTM','PEDROWNOTGRNT','SDOTCOLNUM','ST_COLD
ESC']
df_collision = df_collision.drop(labels =drop_col_list, axis= 1)
df_collision.head()
```

Before performing a great deal of data cleaning, I started looking at any early patterns that emerged.  For example, this simple bar chart shows that there were twice as many people involved in Severity Code 1 accidents than in Severity Code 2 accidents:



In order to account for that wide disparity in counts, I followed some carefully defined examples to downsample the data which normalized it:



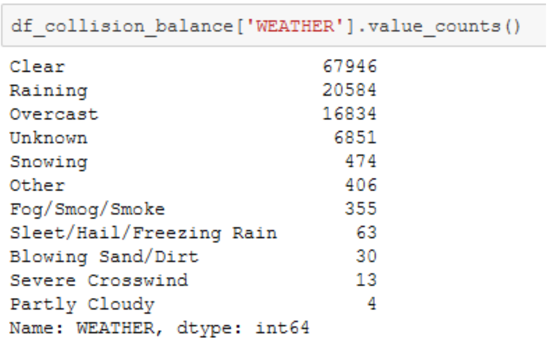
```
#There are more than twice as many severity 1 accidents than there are severity 2 accidents -- which is good since severity 2
are the most severe

#Now we need to "balance" the data - thank you to the classmates who referenced an article on "Elite Data Science":
#"https://elitedatascience.com/imbalanced-classes" for guidance on how to downsample the data


from sklearn.utils import resample
df_collision_max = df_collision[df_collision.SEVERITYCODE ==1]
df_collision_min = df_collision[df_collision.SEVERITYCODE ==2]
df_collision_max_ds = resample(df_collision_max,
                               replace = False,     #sample with replacement
                               n_samples= 58188,    #to match minority class
                               random_state = 123   #reproducible results
                              )

df_collision_balance = pd.concat([df_collision_max_ds,df_collision_min])
df_collision_balance.SEVERITYCODE.value_counts()
```

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

In this way, we saw a more balanced number of Severity 1 and Severity 2 line items.  I reran the same bar chart to evaluate the impact, and it was curious that now, there were more people involved in involved in Severity Code 2 accidents than there were Severity Code 1 accidents.

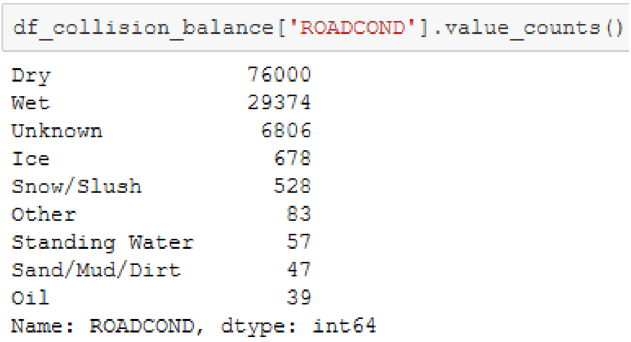
From there I started looking at patterns in the causes of accidents – from weather, road conditions, lighting conditions, address types, collision types, and junction types.  I should note that the first time I ran the models, I did not include collision types or junction types since I didn't see a strong delineation of data indicating one particular field had a remarkable impact.

But, after my initial F1 and Jaccard scores were in the 60% range, I decided to add in a few more columns to see if they would impact the accuracy (and they did!). A few examples are listed below:



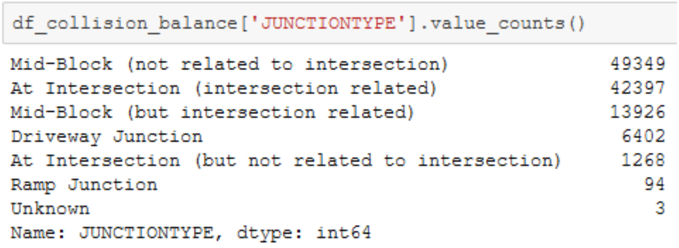
```
df_collision_balance['WEATHER'].value_counts()
```

```
Clear                       67946
Raining                     20584
Overcast                    16834
Unknown                      6851
Snowing                       474
Other                         406
Fog/Smog/Smoke                355
Sleet/Hail/Freezing Rain       63
Blowing Sand/Dirt              30
Severe Crosswind               13
Partly Cloudy                   4
Name: WEATHER, dtype: int64
```



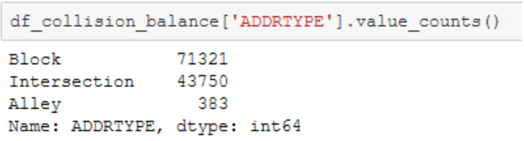
```
df_collision_balance['ROADCOND'].value_counts()
```

```
Dry               76000
Wet               29374
Unknown            6806
Ice                 678
Snow/Slush          528
Other                83
Standing Water       57
Sand/Mud/Dirt        47
Oil                  39
Name: ROADCOND, dtype: int64
```



```
df_collision_balance['JUNCTIONTYPE'].value_counts()
```

```
Mid-Block (not related to intersection)              49349
At Intersection (intersection related)               42397
Mid-Block (but intersection related)                 13926
Driveway Junction                                     6402
At Intersection (but not related to intersection)     1268
Ramp Junction                                           94
Unknown                                                  3
Name: JUNCTIONTYPE, dtype: int64
```



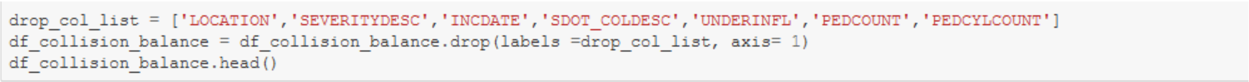
```
df_collision_balance['ADDRTYPE'].value_counts()
```

```
Block           71321
Intersection    43750
Alley             383
Name: ADDRTYPE, dtype: int64
```

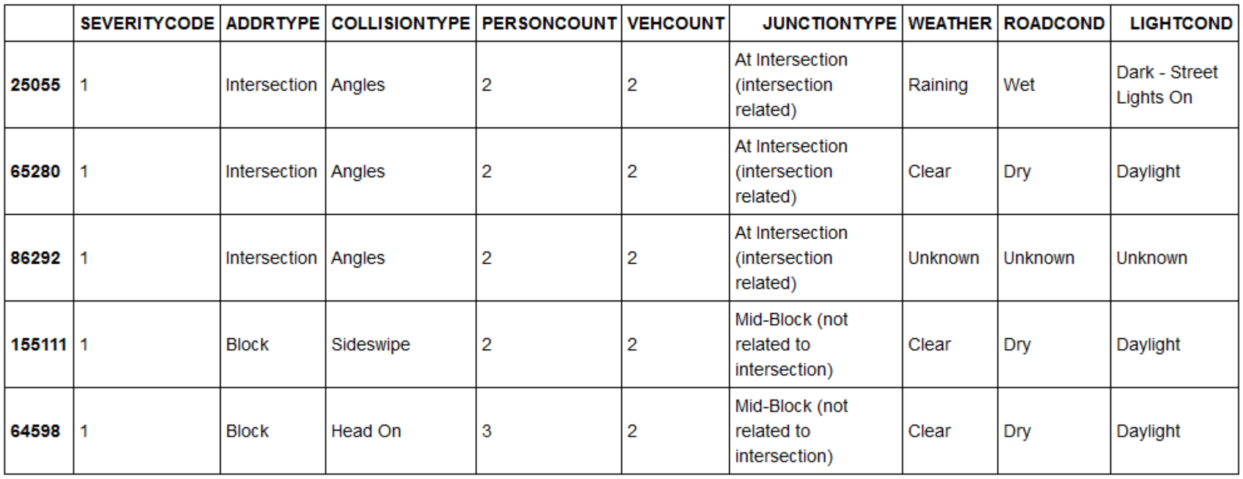
Through this review, I noted that some of the most interesting data points were the address type (“ADDRTYPE”), location (“LOCATION”), weather (“WEATHER”), road conditions (“ROADCOND”), and light conditions (“LIGHTCOND”), junction type (“JUNCTIONTYPE”), and collision type (“COLLISIONTYPE”).

I was surprised to find that such high numbers of accidents occurred on days with clear skies (60%) dry roads (65%). Given that this data came from Seattle, I expected quite the opposite! I also found that very few pedestrians and bicyclists were involved in car accidents, and eventually removed those from consideration. I also removed a couple additional fields such as speeding (“SPEEDING”) and under the influence (“UNDERINFL”) since those data points were not complete across the number of accidents and didn’t lead to a logical conclusion for contribution to the reported accidents.

From there, I started downselecting the best categories and settled on these for my modelling:

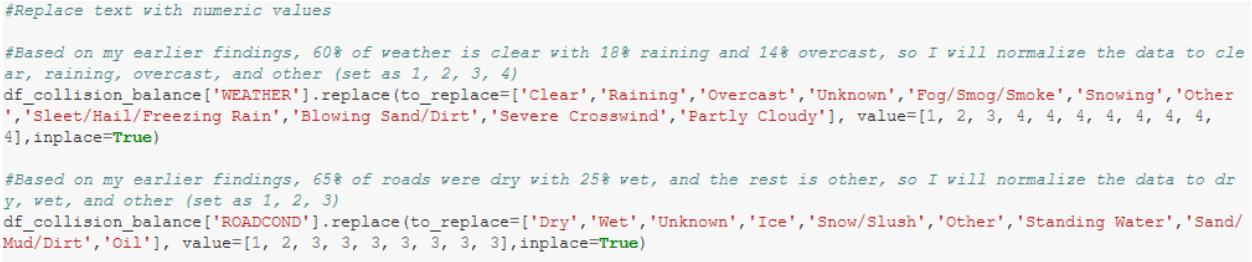


```
drop_col_list = ['LOCATION','SEVERITYDESC','INCDATE','SDOT_COLDESC','UNDERINFL','PEDCOUNT','PEDCYLCOUNT']
df_collision_balance = df_collision_balance.drop(labels =drop_col_list, axis= 1)
df_collision_balance.head()
```

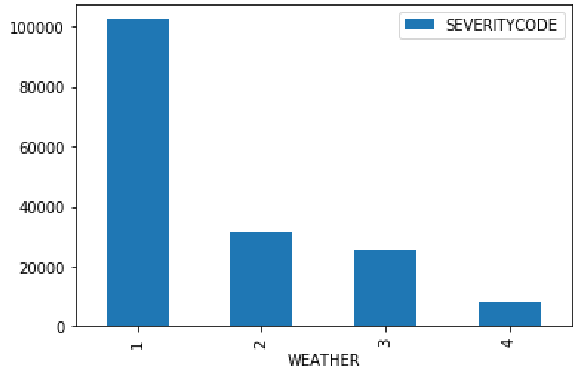


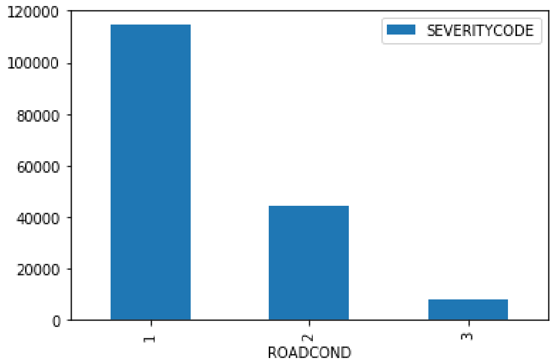
| | SEVERITYCODE | ADDRTYPE | COLLISIONTYPE | PERSONCOUNT | VEHCOUNT | JUNCTIONTYPE | WEATHER | ROADCOND | LIGHTCOND |
|---|---|---|---|---|---|---|---|---|---|
| **25055** | 1 | Intersection | Angles | 2 | 2 | At Intersection (intersection related) | Raining | Wet | Dark - Street Lights On |
| **65280** | 1 | Intersection | Angles | 2 | 2 | At Intersection (intersection related) | Clear | Dry | Daylight |
| **86292** | 1 | Intersection | Angles | 2 | 2 | At Intersection (intersection related) | Unknown | Unknown | Unknown |
| **155111** | 1 | Block | Sideswipe | 2 | 2 | Mid-Block (not related to intersection) | Clear | Dry | Daylight |
| **64598** | 1 | Block | Head On | 3 | 2 | Mid-Block (not related to intersection) | Clear | Dry | Daylight |

AS you will notice, the data is not numerical which makes modelling difficult. I created a new matrix that associated each category with numerical values. Here are a couple of examples: as I mentioned earlier, during the reported accidents, 60% of weather conditioned indicated clear with 18% raining and 14% overcast, so I normalized the data to clear, raining, overcast, and other (set as 1, 2, 3, 4). In a similar way, I noticed that in the reported accidents, 65% of roads were dry with 25% wet, and the rest is other, so I normalize that data to dry, wet, and other (set as 1, 2, 3). I produced similar metrics with the other fields, but show these as examples.

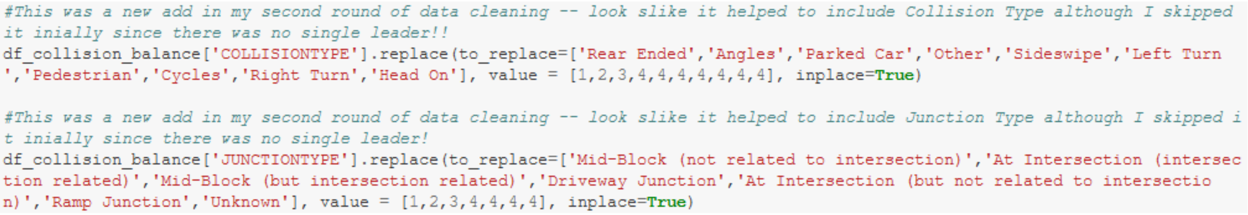


```
#Replace text with numeric values

#Based on my earlier findings, 60% of weather is clear with 18% raining and 14% overcast, so I will normalize the data to cle
ar, raining, overcast, and other (set as 1, 2, 3, 4)
df_collision_balance['WEATHER'].replace(to_replace=['Clear','Raining','Overcast','Unknown','Fog/Smog/Smoke','Snowing','Other
','Sleet/Hail/Freezing Rain','Blowing Sand/Dirt','Severe Crosswind','Partly Cloudy'], value=[1, 2, 3, 4, 4, 4, 4, 4, 4, 4,
4],inplace=True)

#Based on my earlier findings, 65% of roads were dry with 25% wet, and the rest is other, so I will normalize the data to dr
y, wet, and other (set as 1, 2, 3)
df_collision_balance['ROADCOND'].replace(to_replace=['Dry','Wet','Unknown','Ice','Snow/Slush','Other','Standing Water','Sand/
Mud/Dirt','Oil'], value=[1, 2, 3, 3, 3, 3, 3, 3, 3],inplace=True)
```



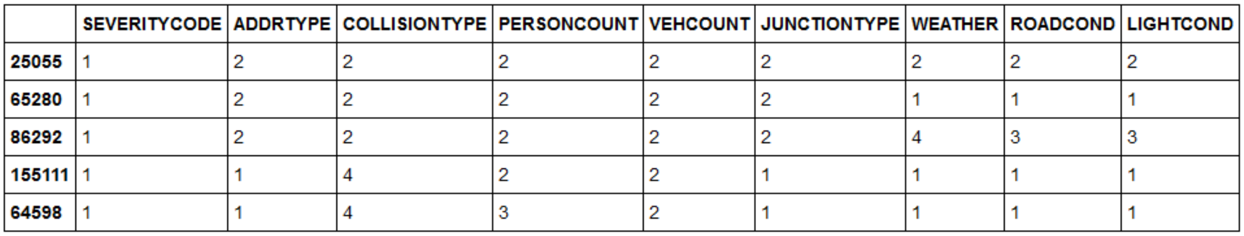


As noted earlier, when I first ran my models, I had not included  junction type (“JUNCTIONTYPE”) or and collision type (“COLLISIONTYPE”), so when I went back through the data cleaning phase, I added these to the numerical normalizing process as well:



```
#This was a new add in my second round of data cleaning -- look slike it helped to include Collision Type although I skipped
it inially since there was no single leader!!
df_collision_balance['COLLISIONTYPE'].replace(to_replace=['Rear Ended','Angles','Parked Car','Other','Sideswipe','Left Turn
','Pedestrian','Cycles','Right Turn','Head On'], value = [1,2,3,4,4,4,4,4,4,4], inplace=True)

#This was a new add in my second round of data cleaning -- look slike it helped to include Junction Type although I skipped i
t inially since there was no single leader!
df_collision_balance['JUNCTIONTYPE'].replace(to_replace=['Mid-Block (not related to intersection)','At Intersection (intersec
tion related)','Mid-Block (but intersection related)','Driveway Junction','At Intersection (but not related to intersectio
n)','Ramp Junction','Unknown'], value = [1,2,3,4,4,4,4], inplace=True)
```

After normalizing the data, you can see the dataframe that I prepared for the 3 models:



| | SEVERITYCODE | ADDRTYPE | COLLISIONTYPE | PERSONCOUNT | VEHCOUNT | JUNCTIONTYPE | WEATHER | ROADCOND | LIGHTCOND |
|---|---|---|---|---|---|---|---|---|---|
| **25055** | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| **65280** | 1 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 |
| **86292** | 1 | 2 | 2 | 2 | 2 | 2 | 4 | 3 | 3 |
| **155111** | 1 | 1 | 4 | 2 | 2 | 1 | 1 | 1 | 1 |
| **64598** | 1 | 1 | 4 | 3 | 2 | 1 | 1 | 1 | 1 |

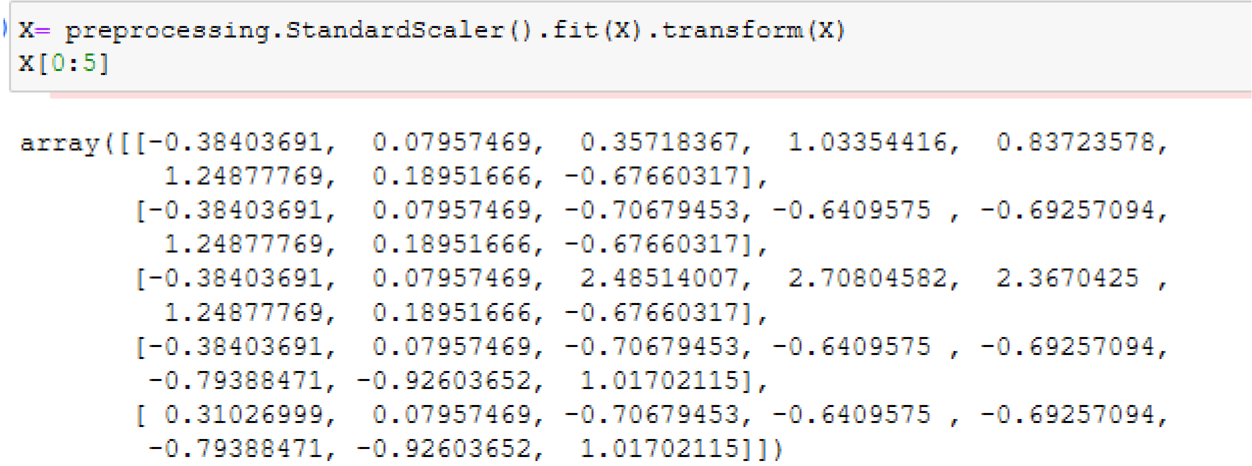
And that concludes data preparation.  I did have to redo several of my assumptions and rerum dataframes to consider more columns to improve my model accuracy.  A Senior Data Scientist and IBM once told me, “Data Scientists spend 80% of their time cleaning data.”  I hadn’t believed him at the time, but after running this project I certainly have a much better understanding of the multiple loops and iterations one will run to ensure the highest accuracy with models.

### Data Modelling and Evaluation

The intent of machine learning is to enable predictions to solve a particular problem or develop a pattern to determine how to improve a system.  In this particular case, I tried to used machine classification algorithms to determine the correlation between car accident factors and the severity of the accident -- where Severity Code 1 = property damage, and Severity Code 2 = injury to a person.  I designed the models to that the Severity Code was our target variable (Y) with eight independent variables (X).

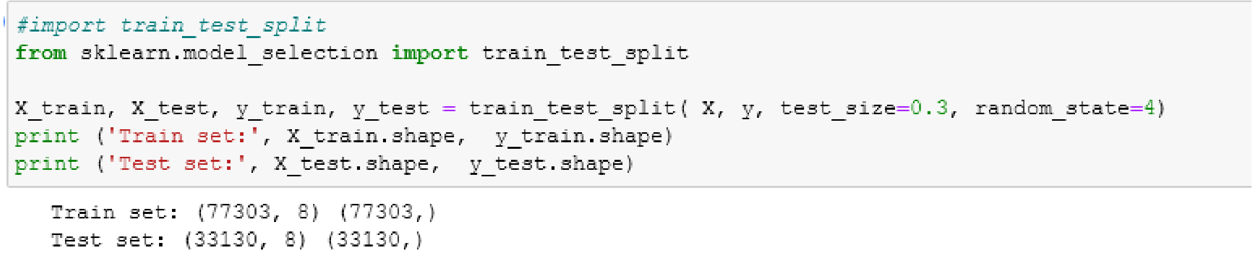
The classification algorithms I used included K-Nearest Neighbors, Decision Tree, and Support Vector Machine.  K-Nearest Neighbors technique evaluates a pattern in the neighboring datapoints to see if a behavior by similar data points will yield a similar result for a new data point. Sometimes, it is not always the “nearest” neighbor (K=1) that yield the best results, so there are some creative ways to address the algorithm by using the technique to count the best approximator of neighbors.  A Decision Tree algorithm looks at all the options for a particular set of attributes and split data until it finds the most comparable set of conditions.  There are multiple ways to set up a decision tree algorithm, but the ideal way is to calculate the lowest levels of entropy in order to determine the purest tree branch. By reducing entropy, you can develop a clearer set of patterns from which to model the data. And as a final example, Support Vector Machine classification allows for data to be distinctly classified by a “Separator” in order to determine a prediction. Support Vector Modelling tends to have quite high accuracy scores. In fact, scores such as F1 and Jaccard index scores are what I will use to evaluate these models to determine which one has the highest accuracy that might determine the severity of a car accident in the Seattle area.

Before running the classification algorithm, I transformed the data and created a train/test split:



```
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

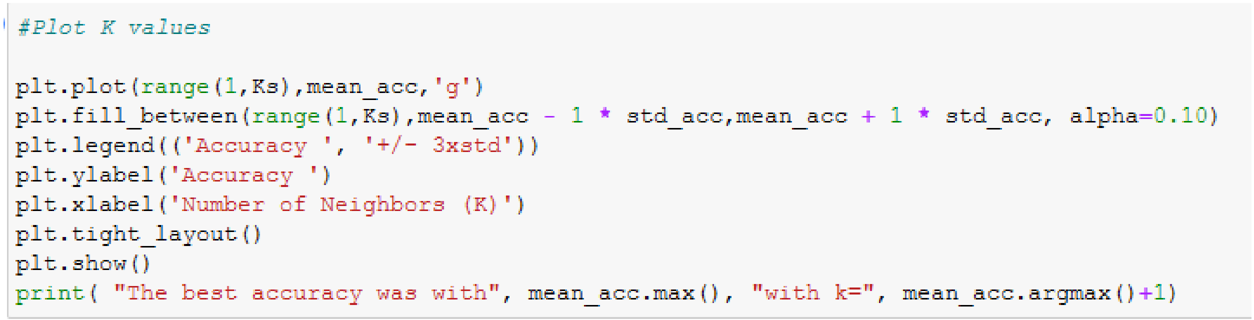
```
array([[-0.38403691,  0.07957469,  0.35718367,  1.03354416,  0.83723578,
         1.24877769,  0.18951666, -0.67660317],
       [-0.38403691,  0.07957469, -0.70679453, -0.6409575 , -0.69257094,
         1.24877769,  0.18951666, -0.67660317],
       [-0.38403691,  0.07957469,  2.48514007,  2.70804582,  2.3670425 ,
         1.24877769,  0.18951666, -0.67660317],
       [-0.38403691,  0.07957469, -0.70679453, -0.6409575 , -0.69257094,
        -0.79388471, -0.92603652,  1.01702115],
       [ 0.31026999,  0.07957469, -0.70679453, -0.6409575 , -0.69257094,
        -0.79388471, -0.92603652,  1.01702115]])
```



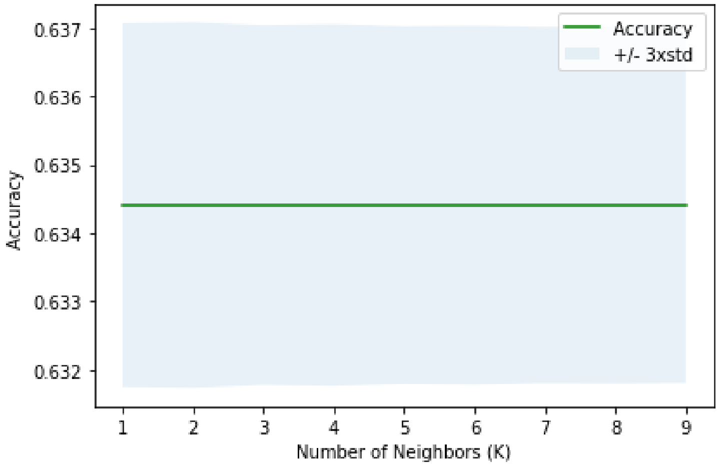
```
#import train_test_split
from sklearn.model_selection import train_test_split

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: (77303, 8) (77303,)
Test set: (33130, 8) (33130,)
```

With training and testing data split, I conducted the K-Nearest Neighbors Algorithm first. I was surprised that my “best” K-neighbors ended up to be one:

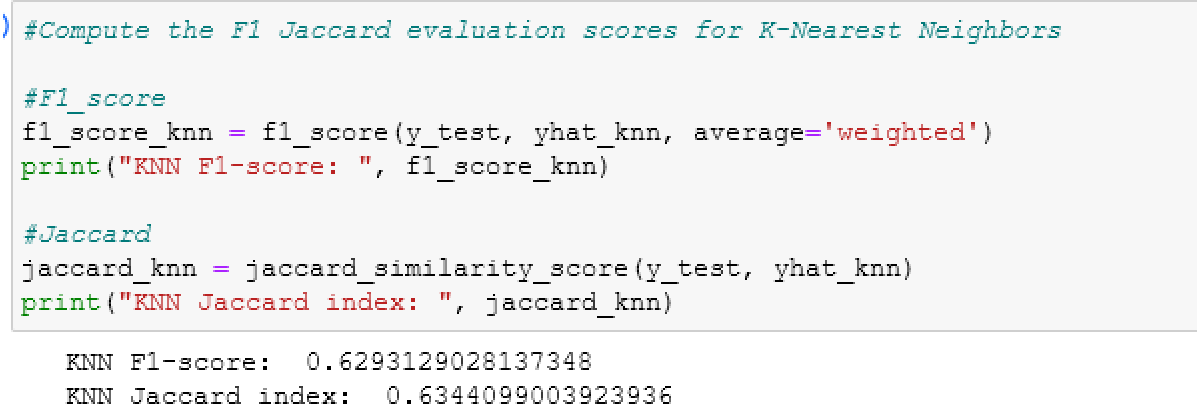


```
#Plot K values

plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```



```
The best accuracy was with 0.6344099003923936 with k= 1
```

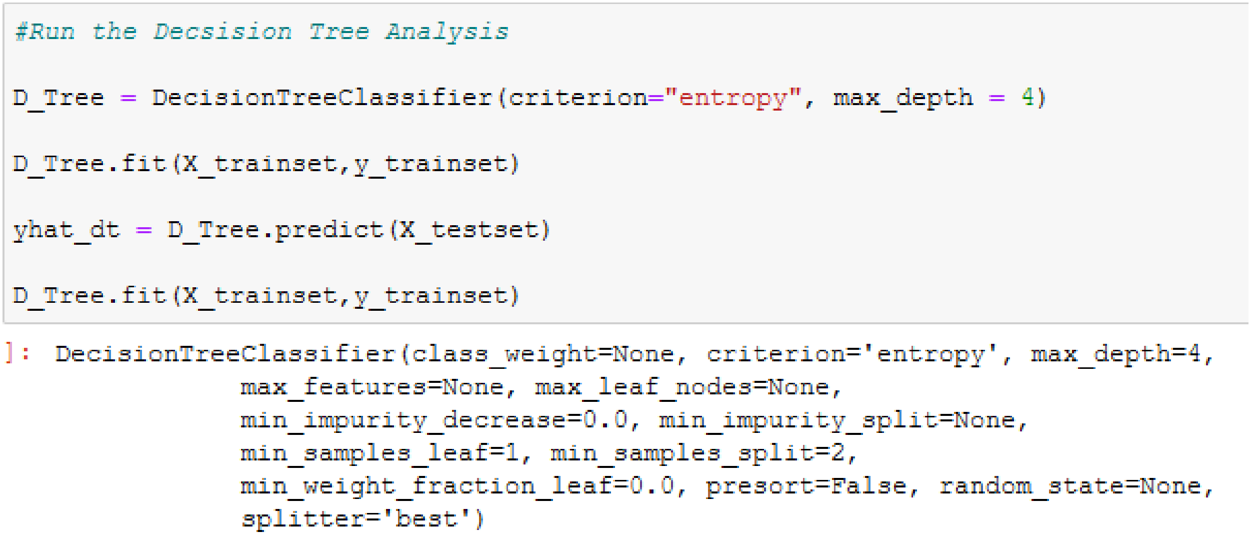
Here is the calculation of the F1-score and Jaccard index for the K-Nearest Neighbor accuracy ratings:



```
#Compute the F1 Jaccard evaluation scores for K-Nearest Neighbors

#F1_score
f1_score_knn = f1_score(y_test, yhat_knn, average='weighted')
print("KNN F1-score: ", f1_score_knn)

#Jaccard
jaccard_knn = jaccard_similarity_score(y_test, yhat_knn)
print("KNN Jaccard index: ", jaccard_knn)
```

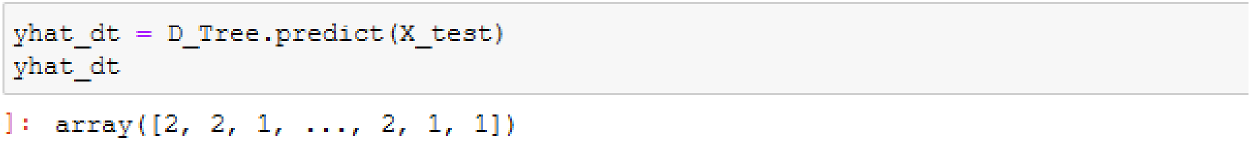
```
KNN F1-score:  0.6293129028137348
KNN Jaccard index:  0.6344099003923936
```

Then I created my Decision Tree algorithm:



```
#Run the Decsision Tree Analysis

D_Tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)

D_Tree.fit(X_trainset,y_trainset)

yhat_dt = D_Tree.predict(X_testset)

D_Tree.fit(X_trainset,y_trainset)
```

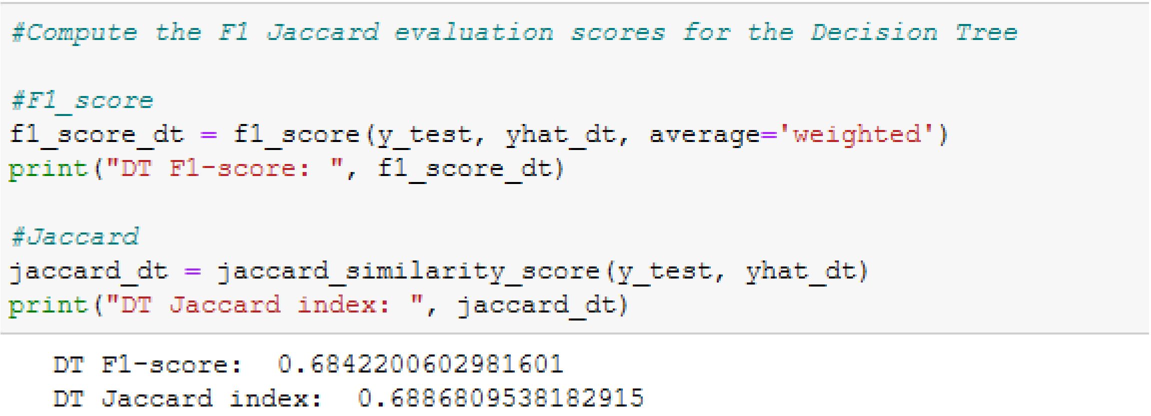
```
]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
               max_features=None, max_leaf_nodes=None,
               min_impurity_decrease=0.0, min_impurity_split=None,
               min_samples_leaf=1, min_samples_split=2,
               min_weight_fraction_leaf=0.0, presort=False, random_state=None,
               splitter='best')
```



```
yhat_dt = D_Tree.predict(X_test)
yhat_dt
```

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

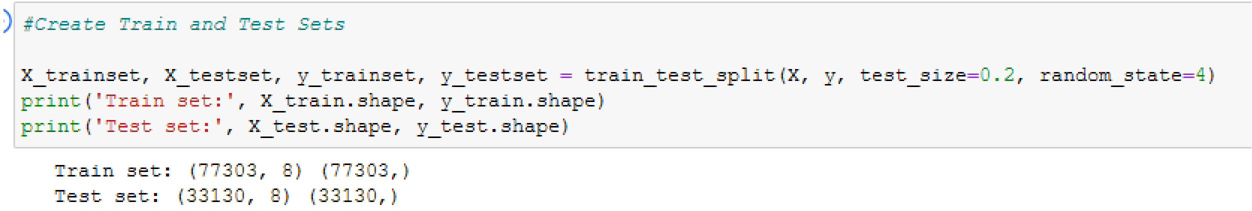
Here is the calculation of the F1-score and Jaccard index for the Decision Tree accuracy ratings:



```
#Compute the F1 Jaccard evaluation scores for the Decision Tree

#F1_score
f1_score_dt = f1_score(y_test, yhat_dt, average='weighted')
print("DT F1-score: ", f1_score_dt)

#Jaccard
jaccard_dt = jaccard_similarity_score(y_test, yhat_dt)
print("DT Jaccard index: ", jaccard_dt)
```

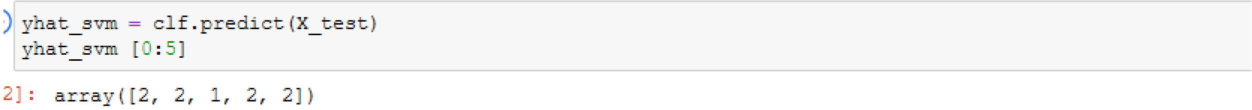
```
DT F1-score:  0.6842200602981601
DT Jaccard index:  0.6886809538182915
```

And finally, I developed the Support Vector Machine algorithm:



```
#Create Train and Test Sets

X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.2, random_state=4)
print('Train set:', X_train.shape, y_train.shape)
print('Test set:', X_test.shape, y_test.shape)
```

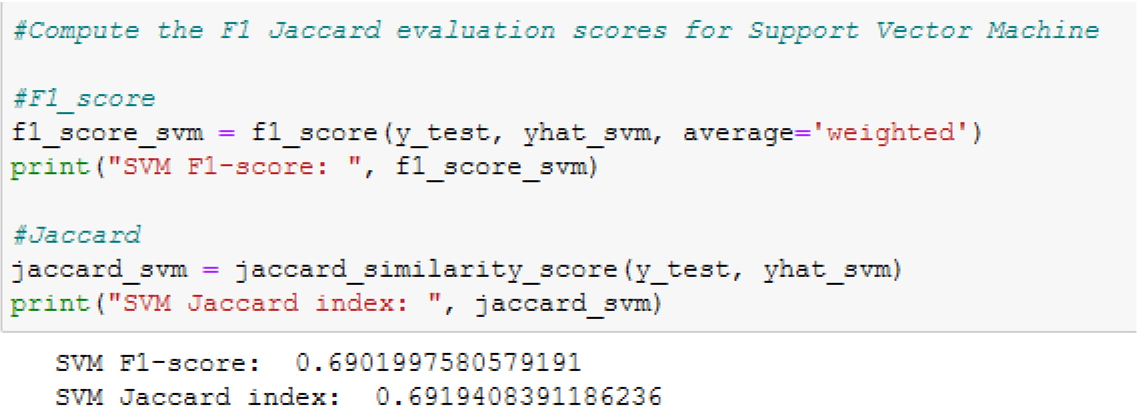
```
   Train set: (77303, 8) (77303,)
   Test set: (33130, 8) (33130,)
```



```
yhat_svm = clf.predict(X_test)
yhat_svm [0:5]
```

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

Here is the calculation of the F1-score and Jaccard index for the Support Vector Machine accuracy ratings:

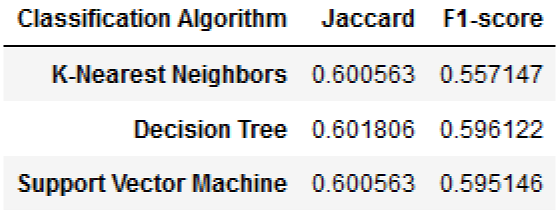


```
#Compute the F1 Jaccard evaluation scores for Support Vector Machine

#F1_score
f1_score_svm = f1_score(y_test, yhat_svm, average='weighted')
print("SVM F1-score: ", f1_score_svm)

#Jaccard
jaccard_svm = jaccard_similarity_score(y_test, yhat_svm)
print("SVM Jaccard index: ", jaccard_svm)
```

```
   SVM F1-score:  0.6901997580579191
   SVM Jaccard index:  0.6919408391186236
```

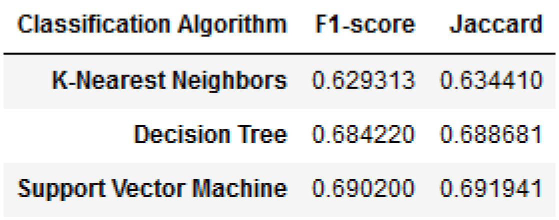
Not surprisingly, the Support Vector Machine algorithm was the highest.

As I mentioned earlier, the first time I ran the three algorithms with a smaller set of independent variables, I received significantly lower F1 and Jaccard scores:



| Classification Algorithm | Jaccard | F1-score |
|---|---|---|
| K-Nearest Neighbors | 0.600563 | 0.557147 |
| Decision Tree | 0.601806 | 0.596122 |
| Support Vector Machine | 0.600563 | 0.595146 |

Fortunately, after reevaluating the variables and columns for modelling,  I decided to add junction type (“JUNCTIONTYPE”) or and collision type (“COLLISIONTYPE”) which improved my NEW F1 and Jaccard scores:



| Classification Algorithm | F1-score | Jaccard |
|---|---|---|
| K-Nearest Neighbors | 0.629313 | 0.634410 |
| Decision Tree | 0.684220 | 0.688681 |
| Support Vector Machine | 0.690200 | 0.691941 |

### Data Conclusions

After redoing the analysis and redeveloping the models, I was able to produce results that would indicate that there is an opportunity to predict the severity of a car accident to closer to 69%. The Support Vector Machine classification produced the highest set of scores. In my first analysis, the F1 and Jaccard scores were closer to 60% across all three classification models (in the first set of data, I only modelled people counts, weather, light conditions, address types). After seeing F1 and Jaccard scores in the 60% range, I decided I needed to reevaluate the data fields included. By adding vehicle counts, junction types, and collision types to the model test / training set, I was able to produce much better F1 and Jaccard scores from 63-69%. While 69% isn't a high score, it did strongly show a correlation among the data categories.

In order to improve the model further, I would think new data types would need to be introduced. Perhaps data fields such as the age or a driver or the type of vehicle could have a larger impact on the crash severity.

[1] - National Highway Traffic Safety Administration, https://crashstats.nhtsa.dot.gov/#!/#%2F

[2] – Coursera data set and Metadata file with explanations, https://www.coursera.org/learn/applied-data-science-capstone/supplement/Nh5uS/downloading-example-dataset