Car Accident Severity Prediction Project Report

1. Introduction:

This is the final project from Coursera IBM Data Science Capstone Moduel. This project is focusing on building a machine learning model to predict the severity code of a car accident based on the big data (raw dataset can be found here: https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv).

2. Business Understanding:

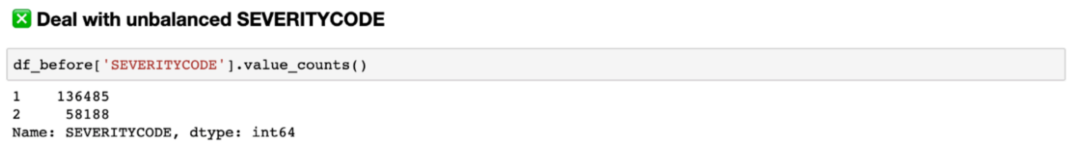
With a rising number of car accidents happened in our community, an predictor must be built in order to give people a guidance of the potential of how severe a car accident might happen given some prerequisites like: road conditions, weather conditions, location, etc. When the predictor gives a severe code prediction, it could alert the driver to pay attention when driving in order to avoid the car collision happening.

3. Data understanding:

This dataset consists of 37 attributes and those are known as independent variables and the 'SEVERITYCODE' is known as dependent variable. The aim is building a machine learning model to predict the 'SEVERITYCODE' based on those independent variables.

By looking at this dataset, there are some problems need to be fixed:

- It has too many attributes (37)
- It is an unbalanced dataset (SEVERITYCODE column)
- Some of the data is NaN



❎ **Deal with unbalanced SEVERITYCODE**

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

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

SEVERITYCODE 1 has a number which is far more bigger than SEVERITYCODE 2, hence this is an unbalanced dataset

Severity Code as shown as follows as a reference:

0: Little to no Probability (Clear Conditions)

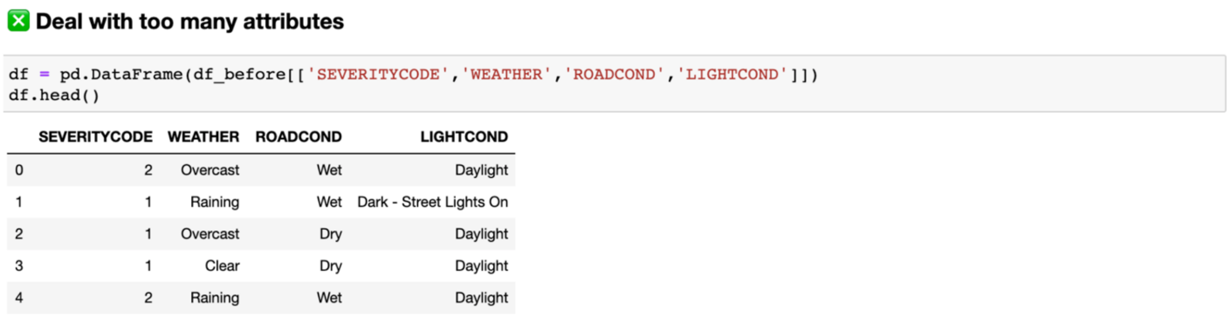
1: Very Low Probability — Chance or Property Damage

2: Low Probability — Chance of Injury

3: Mild Probability — Chance of Serious Injury

4: High Probability — Chance of Fatality

🔧The first problem can be fixed by removing the unwanted attributes and only keep the useful variables:



❎ Deal with too many attributes

```
df = pd.DataFrame(df_before[['SEVERITYCODE','WEATHER','ROADCOND','LIGHTCOND']])
df.head()
```

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND
0	2	Overcast	Wet	Daylight
1	1	Raining	Wet	Dark - Street Lights On
2	1	Overcast	Dry	Daylight
3	1	Clear	Dry	Daylight
4	2	Raining	Wet	Daylight

Another problem can be observed is that the type of those data is object which can be difficult to implement further model training and testing. By coding those conditions with a corresponding number can fix this problem, codes are shown below:

```
encoding_weather = {'WEATHER':{'Clear': 1, 'Partly Cloudy': 2, 'Overcast':3,  'Fog/Smog/Smoke':4, 'Severe Crosswind':5,
df.replace(encoding_weather, inplace=True)
df.head()
```

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND
0	2	3.0	Wet	Daylight
1	1	6.0	Wet	Dark - Street Lights On
2	1	3.0	Dry	Daylight
3	1	1.0	Dry	Daylight
4	2	6.0	Wet	Daylight

```
encoding_roadcond = {'ROADCOND':{'Dry': 1, 'Sand/Mud/Dirt':2, 'Oil':3, 'Wet':4,'Standing Water':5,'Snow/Slush':6,'Ice':
df.replace(encoding_roadcond, inplace=True)
df.head()
```

	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND
0	2	3.0	4.0	Daylight
1	1	6.0	4.0	Dark - Street Lights On
2	1	3.0	1.0	Daylight
3	1	1.0	1.0	Daylight
4	2	6.0	4.0	Daylight

```
encoding_lightcond = {'LIGHTCOND':{'Daylight':1, 'Dawn':2, 'Dusk':3, 'Dark - Street Lights On':4, 'Dark - Street Lights
df.replace(encoding_lightcond, inplace=True)
df.head()
```

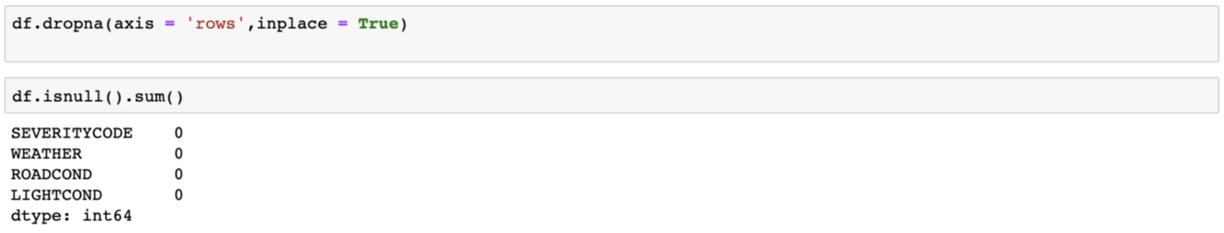
	SEVERITYCODE	WEATHER	ROADCOND	LIGHTCOND
0	2	3.0	4.0	1.0
1	1	6.0	4.0	4.0
2	1	3.0	1.0	1.0
3	1	1.0	1.0	1.0
4	2	6.0	4.0	1.0

🔧 The second problem can be fixed by balancing out this dataset, what we want is to have a equal number of SEVERITYCODE 1 and SEVERITYCODE 2, codes are shown below:

```
from sklearn.utils import resample

df_before_maj = df_before[df_before.SEVERITYCODE==1]
df_before_min = df_before[df_before.SEVERITYCODE==2]

df_before_maj_upsampled = resample(df_before_maj,
                                 replace=True,     # sample with replacement
                                 n_samples=58188,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
balanced_df = pd.concat([df_before_maj_upsampled, df_before_min])
balanced_df.SEVERITYCODE.value_counts()
```

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

The third goal is to deal with the NaN value and in this case, I decided to drop the NaN value:



```
df.dropna(axis = 'rows',inplace = True)
```

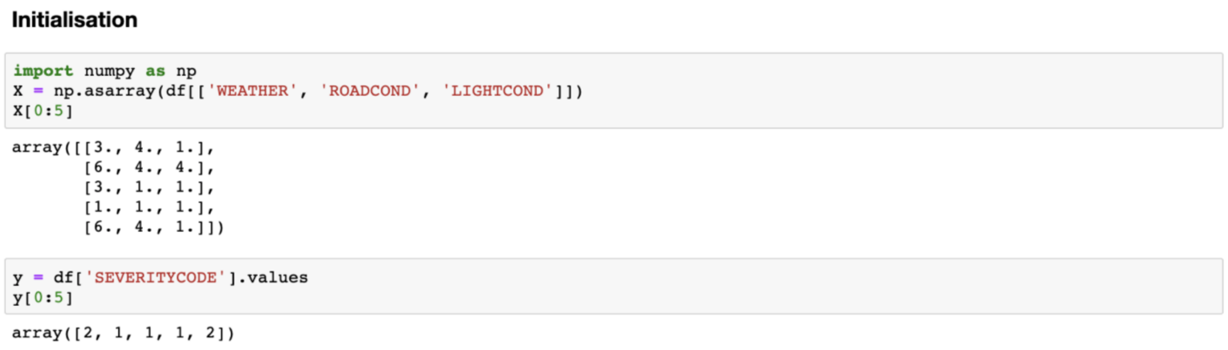
```
df.isnull().sum()
```

```
SEVERITYCODE     0
WEATHER          0
ROADCOND         0
LIGHTCOND        0
dtype: int64
```

4. Data Preparation:

After finishing dealing with unprepared dataset, we are ready to move on to the data initialisation section.

We use the 'asarray' method from numpy library to create an array for both independent and dependent variables for the sake of convenient model implementation.



Initialisation

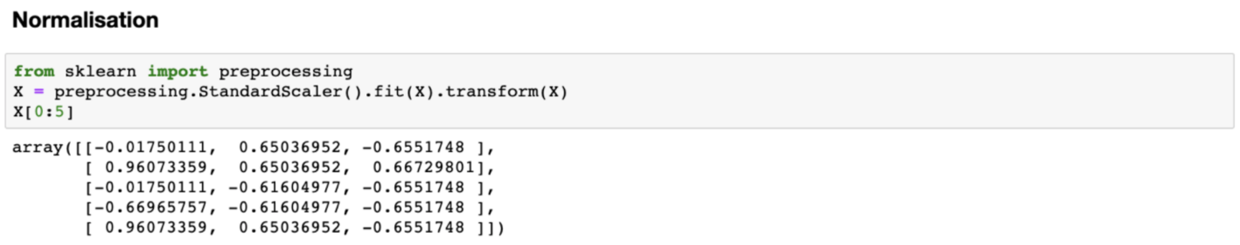
```
import numpy as np
X = np.asarray(df[['WEATHER', 'ROADCOND', 'LIGHTCOND']])
X[0:5]
```

```
array([[3., 4., 1.],
       [6., 4., 4.],
       [3., 1., 1.],
       [1., 1., 1.],
       [6., 4., 1.]])
```

```
y = df['SEVERITYCODE'].values
y[0:5]
```

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

Next, do the data normalisation for the independent variables:

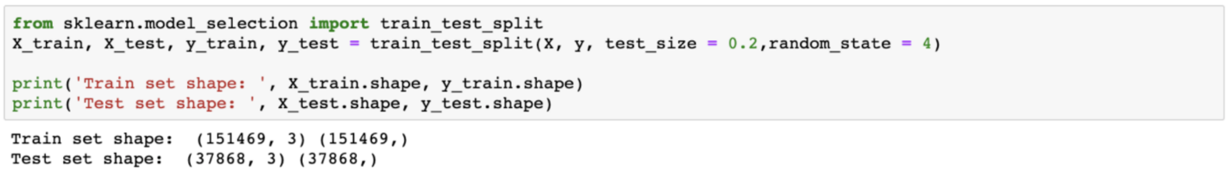


Normalisation

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

```
array([[-0.01750111,  0.65036952, -0.6551748 ],
       [ 0.96073359,  0.65036952,  0.66729801],
       [-0.01750111, -0.61604977, -0.6551748 ],
       [-0.66965757, -0.61604977, -0.6551748 ],
       [ 0.96073359,  0.65036952, -0.6551748 ]])
```

🔧Then, this dataset needs to be split into training and testing set:



```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state = 4)

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

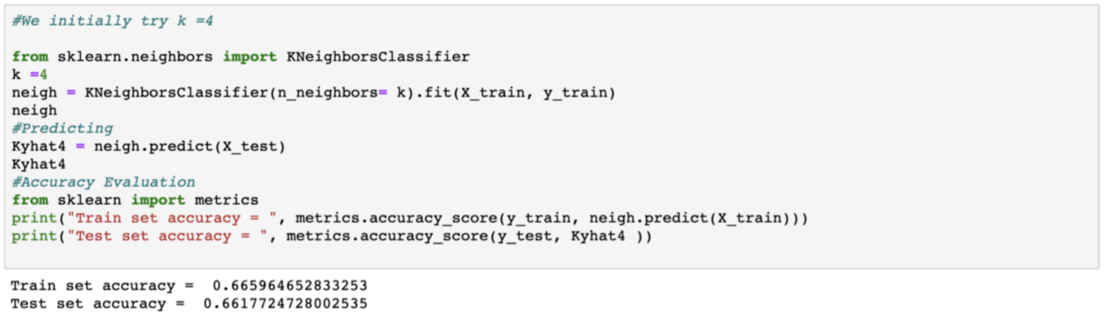
```
Train set shape:  (151469, 3) (151469,)
Test set shape:  (37868, 3) (37868,)
```

5. Modeling:

In this case, I come up with four methodologies which are:

- K Nearest Neighbours(KNN)
- Decision Tree
- Support Vector Machine(SVM)
- Logistic Regression

K Nearest Neighbours (KNN):

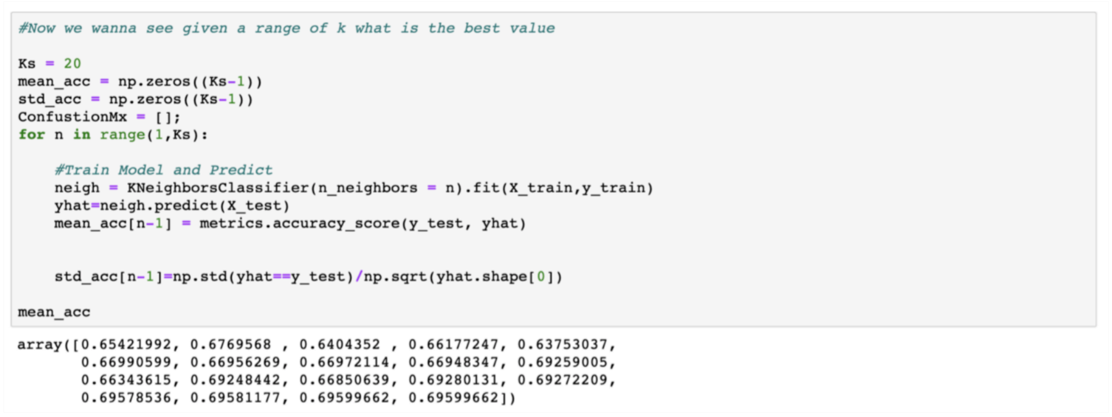


```
#We initially try k =4

from sklearn.neighbors import KNeighborsClassifier
k =4
neigh = KNeighborsClassifier(n_neighbors= k).fit(X_train, y_train)
neigh
#Predicting
Kyhat4 = neigh.predict(X_test)
Kyhat4
#Accuracy Evaluation
from sklearn import metrics
print("Train set accuracy = ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set accuracy = ", metrics.accuracy_score(y_test, Kyhat4 ))
```

```
Train set accuracy =  0.665964652833253
Test set accuracy =  0.6617724728002535
```

Firstly, try k = 4 case and see the train set accuracy and test set accuracy

Screen Shot 2020-09-22 at 4.55.25 PM

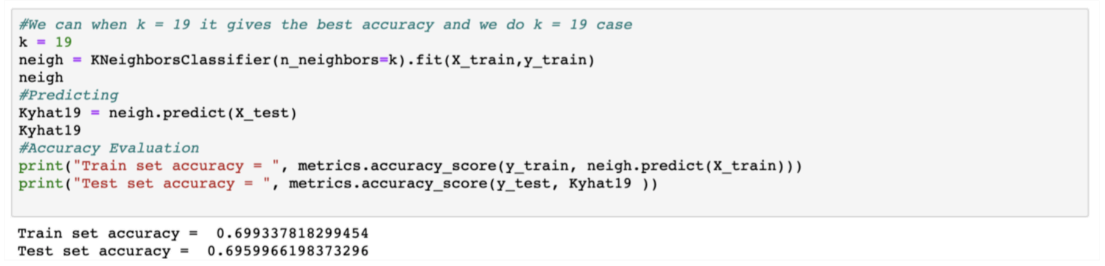


```
#Now we wanna see given a range of k what is the best value

Ks = 20
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

```
array([0.65421992, 0.6769568 , 0.6404352 , 0.66177247, 0.63753037,
       0.66990599, 0.66956269, 0.66972114, 0.66948347, 0.69259005,
       0.66343615, 0.69248442, 0.66850639, 0.69280131, 0.69272209,
       0.69578536, 0.69581177, 0.69599662, 0.69599662])
```

Given a range of k from 1 to 20 and it is obvious when k = 19 it gives the optimal training, testing set accuracy

Screen Shot 2020-09-22 at 4.55.39 PM



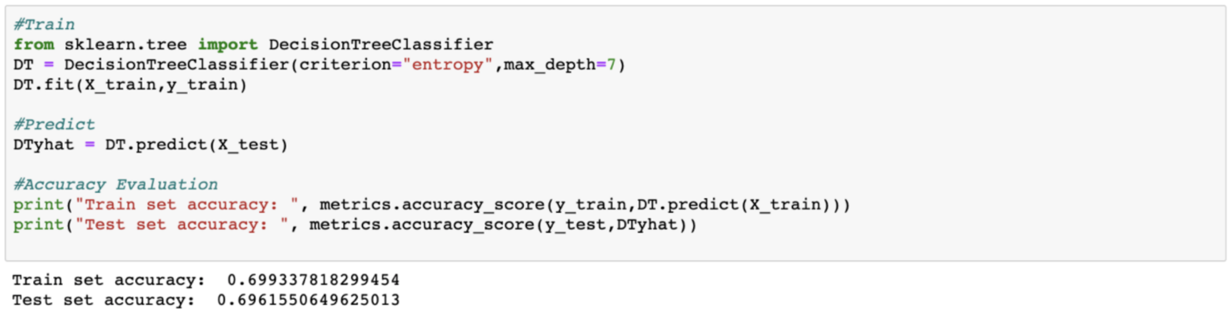
```
#We can when k = 19 it gives the best accuracy and we do k = 19 case
k = 19
neigh = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
neigh
#Predicting
Kyhat19 = neigh.predict(X_test)
Kyhat19
#Accuracy Evaluation
print("Train set accuracy = ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set accuracy = ", metrics.accuracy_score(y_test, Kyhat19 ))
```

```
Train set accuracy =  0.699337818299454
Test set accuracy =  0.6959966198373296
```

Finally, implement the k = 19 case

Screen Shot 2020-09-22 at 4.55.49 PM

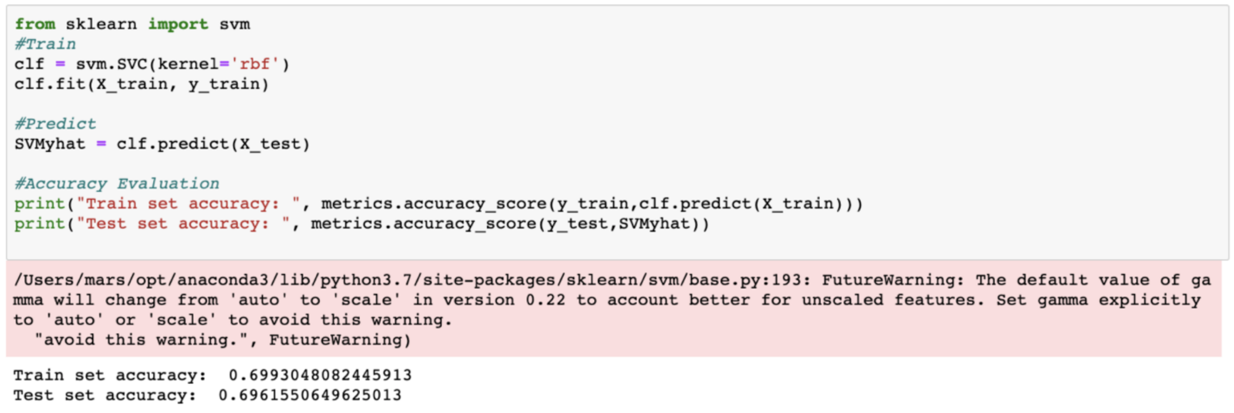
Decision Tree



```
#Train
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier(criterion="entropy",max_depth=7)
DT.fit(X_train,y_train)

#Predict
DTyhat = DT.predict(X_test)

#Accuracy Evaluation
print("Train set accuracy: ", metrics.accuracy_score(y_train,DT.predict(X_train)))
print("Test set accuracy: ", metrics.accuracy_score(y_test,DTyhat))
```

```
Train set accuracy:  0.699337818299454
Test set accuracy:  0.6961550649625013
```

Support Vector Machine (SVM)

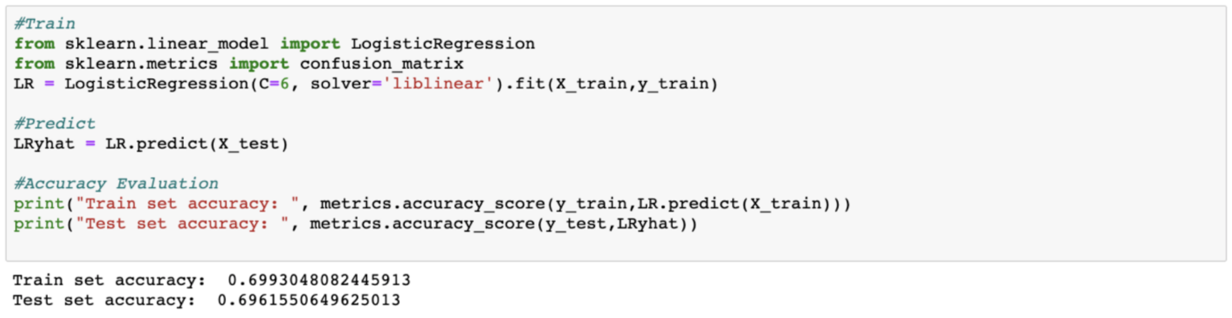


```
from sklearn import svm
#Train
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)

#Predict
SVMyhat = clf.predict(X_test)

#Accuracy Evaluation
print("Train set accuracy: ", metrics.accuracy_score(y_train,clf.predict(X_train)))
print("Test set accuracy: ", metrics.accuracy_score(y_test,SVMyhat))
```

```
/Users/mars/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193: FutureWarning: The default value of ga
mma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly
to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

```
Train set accuracy:  0.6993048082445913
Test set accuracy:  0.6961550649625013
```

Logistic Regression



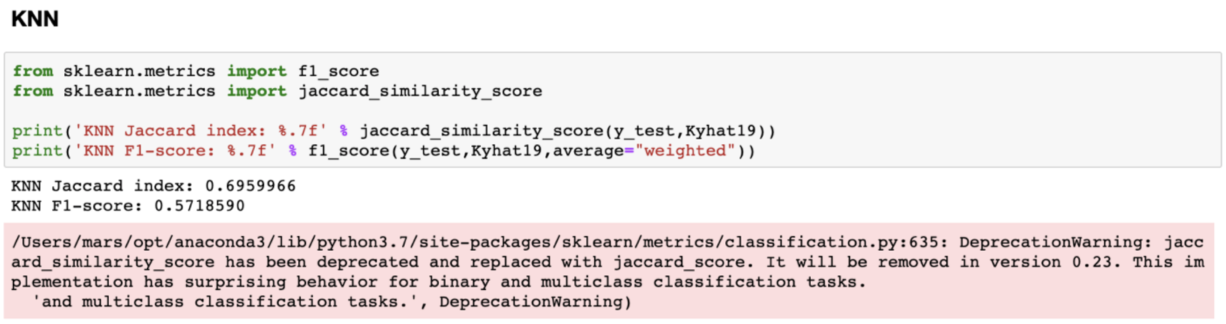
```
#Train
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=6, solver='liblinear').fit(X_train,y_train)

#Predict
LRyhat = LR.predict(X_test)

#Accuracy Evaluation
print("Train set accuracy: ", metrics.accuracy_score(y_train,LR.predict(X_train)))
print("Test set accuracy: ", metrics.accuracy_score(y_test,LRyhat))
```

```
Train set accuracy:  0.6993048082445913
Test set accuracy:  0.6961550649625013
```

6. Evaluation:

For each of the models we will calculate the **Jaccard index and F1-Score**:

- *The Jaccard Index, also known as the Jaccard similarity coefficient, is a statistic used in understanding the similarities between sample sets. The measurement emphasizes similarity between finite sample sets, and is formally defined as the size of the intersection divided by the size of the union of the sample sets.*
- *It is calculated from the precision and recall of the test, where the* ***precision is the number of correctly identified positive results divided by the number of all positive results, including those not identified correctly,\*** *and the* ***recall is the number of correctly identified positive results divided by the number of all samples that should have been identified as positive.\*** *The highest possible value of F1 is 1, indicating perfect precision and recall, and the lowest possible value is 0, if either the precision or the recall is zero.*

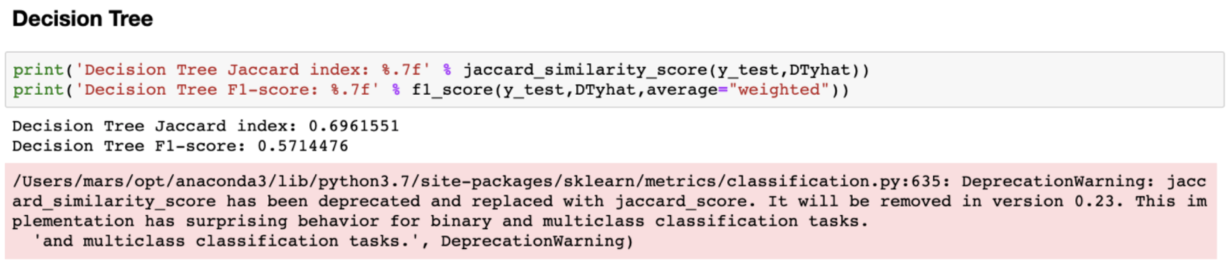


KNN

```
from sklearn.metrics import f1_score
from sklearn.metrics import jaccard_similarity_score

print('KNN Jaccard index: %.7f' % jaccard_similarity_score(y_test,Kyhat19))
print('KNN F1-score: %.7f' % f1_score(y_test,Kyhat19,average="weighted"))
```

```
KNN Jaccard index: 0.6959966
KNN F1-score: 0.5718590
```

```
/Users/mars/opt/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:635: DeprecationWarning: jacc
ard_similarity_score has been deprecated and replaced with jaccard_score. It will be removed in version 0.23. This im
plementation has surprising behavior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

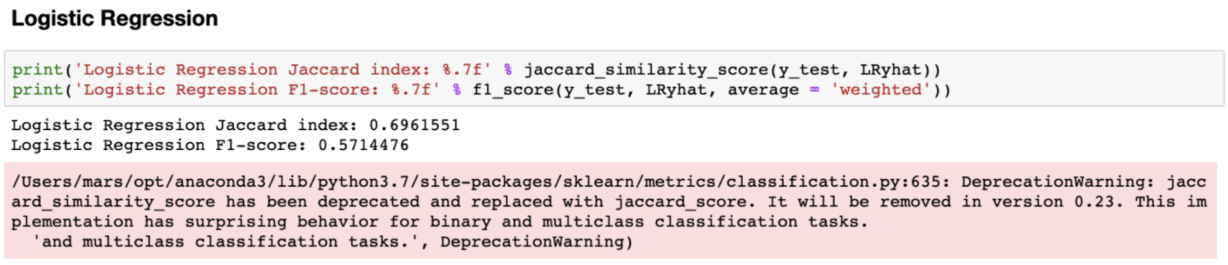


Decision Tree

```
print('Decision Tree Jaccard index: %.7f' % jaccard_similarity_score(y_test,DTyhat))
print('Decision Tree F1-score: %.7f' % f1_score(y_test,DTyhat,average="weighted"))
```

```
Decision Tree Jaccard index: 0.6961551
Decision Tree F1-score: 0.5714476
```

```
/Users/mars/opt/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:635: DeprecationWarning: jacc
ard_similarity_score has been deprecated and replaced with jaccard_score. It will be removed in version 0.23. This im
plementation has surprising behavior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

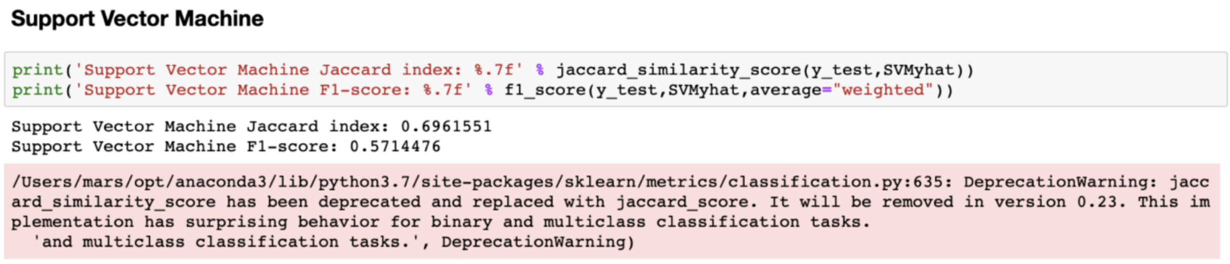


Logistic Regression

```
print('Logistic Regression Jaccard index: %.7f' % jaccard_similarity_score(y_test, LRyhat))
print('Logistic Regression F1-score: %.7f' % f1_score(y_test, LRyhat, average = 'weighted'))
```

```
Logistic Regression Jaccard index: 0.6961551
Logistic Regression F1-score: 0.5714476
```

```
/Users/mars/opt/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:635: DeprecationWarning: jacc
ard_similarity_score has been deprecated and replaced with jaccard_score. It will be removed in version 0.23. This im
plementation has surprising behavior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```



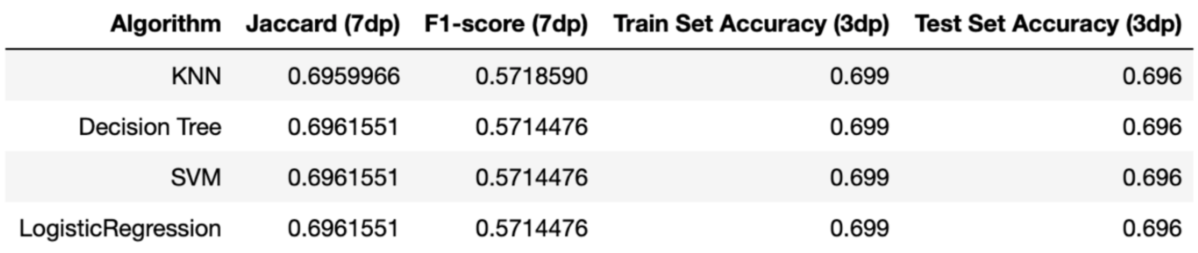
Support Vector Machine

```
print('Support Vector Machine Jaccard index: %.7f' % jaccard_similarity_score(y_test,SVMyhat))
print('Support Vector Machine F1-score: %.7f' % f1_score(y_test,SVMyhat,average="weighted"))
```

```
Support Vector Machine Jaccard index: 0.6961551
Support Vector Machine F1-score: 0.5714476
```

```
/Users/mars/opt/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:635: DeprecationWarning: jacc
ard_similarity_score has been deprecated and replaced with jaccard_score. It will be removed in version 0.23. This im
plementation has surprising behavior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

Results can be clearly shown in a table:



Algorithm	Jaccard (7dp)	F1-score (7dp)	Train Set Accuracy (3dp)	Test Set Accuracy (3dp)
KNN	0.6959966	0.5718590	0.699	0.696
Decision Tree	0.6961551	0.5714476	0.699	0.696
SVM	0.6961551	0.5714476	0.699	0.696
LogisticRegression	0.6961551	0.5714476	0.699	0.696

7. Conclusion:

In this project, by using the data to train and test the model, those four models are now ready to use. This could be helpful especially for those cars whose are executing special missions like: ambulance cars, police cars, school bus, etc. This model could give the driver an alert so that some bad car collision cases could be avoided. By looking at this model, the severity of a car collision could be reduced by, for example, improving the road conditions or increasing the awareness of the population.

8. Advice on Further Analysis:

However, this model is not that optimal because the Jaccard index can only go up to about 70% for each of these models. This is due to less attributes are used to train the model. If more attributes, such as, car speed, location can be introduced when training this model. They could make this algorithm to be more accurate in predicting the severity code of a car accident. Also, the model's executing time can be optimized.