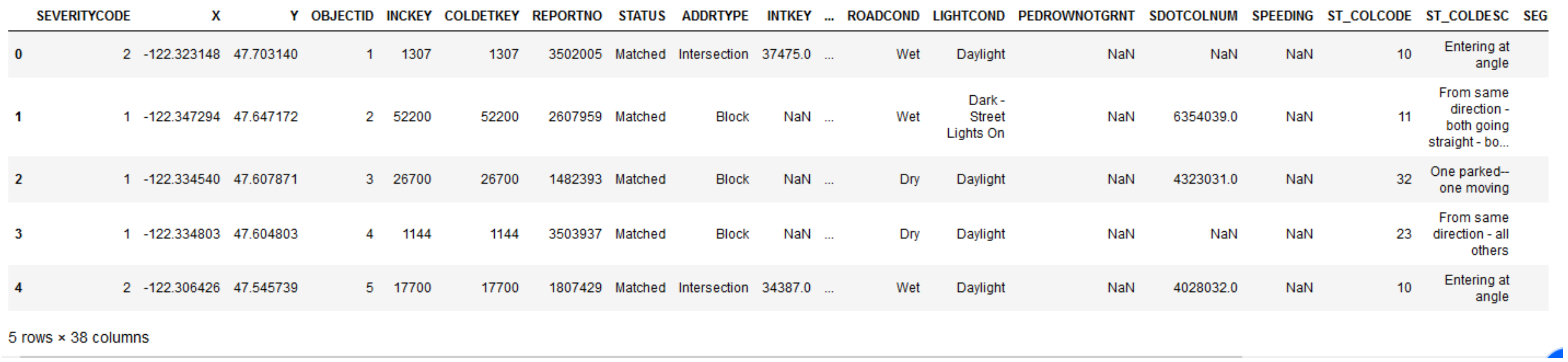
Introduction

Traveling in and around Seattle City is not unlike any situation traveling around any city or on any public roadway. When traffic stops or backs up for miles, drivers are often wondering how long it will take to clear and if an alternate route should be found. Modern vehicles with modern GPS systems use Predictive Traffic feature which gives the driver an understanding of of the traffic patterns and possible delays using real time traffic monitoring and integrated navigation to find a more efficient route in the event of an accident or slowdown. In this analysis I want to use the shared sample data set to predict the severity of an accident. This could be used to feed such a system or an electronic highway sign to re-route traffic in the case of a severe accident or fatality.

Data

I plan to use the share data set example of Seattle City for my analysis. The data requires balancing, pre-processing and cleaning before it can be used to train and test the model. The model will use the severity code as the target label and the rest of the labels, as determined by pre-processing and exploratory analysis, will be used as predictors. The following is an exploration of the data set.

Initial look at the columns and label types produces a table of 37 attributes, which cannot be fully displayed. Many of these attributes are unnecessary for my analysis. I only want to deal with the attributes that contribute to the severity of an accident. Below is an initial look at the data to visually determine the types.



	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT	SDOTCOLNUM	SPEEDING	ST_COLCODE	ST_COLDESC	SEG
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight	NaN	NaN	NaN	10	Entering at angle	
1	1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On	NaN	6354039.0	NaN	11	From same direction - both going straight - bo...	
2	1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight	NaN	4323031.0	NaN	32	One parked-- one moving	
3	1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight	NaN	NaN	NaN	23	From same direction - all others	
4	2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight	NaN	4028032.0	NaN	10	Entering at angle	

5 rows × 38 columns

I used the following code to make the data set more manageable by dropping the unnecessary attributes

```
# Dropping of some of the label determined as non essential to the analysis (some may be added back at a later time)
```

```
df.drop(['X','Y','OBJECTID','INCKEY','COLDETKEY','REPORTNO','STATUS','INTKEY','LOCATION','EXCEPTRSNCODE','EXCEPTRSNDESC','SEVERITYCODE.1','PEDROWNOTGRNT','INCDATE','SDOTCOLNUM','S
```

I then replaced objects with Int.

```
df['ADDRTYPE'] = df['ADDRTYPE'].replace(['Block','Intersection','Alley'],['1','2','3',])
```

```
df['SEVERITYDESC'] = df['SEVERITYDESC'].replace(['Property Damage Only Collision','Injury Collision'],['1','2'])
```

```
df['COLLISIONTYPE'] = df['COLLISIONTYPE'].replace(['Parked Car','Angles','Rear Ended','Other','Sideswipe','Left Turn','Pedestrian','Cycles','Right Turn','Head On'],['1','2','3','4
```

```
df['WEATHER'] = df['WEATHER'].replace(['Clear','Raining','Overcast','Unknown','Snowing','Other','Fog/Smog/Smoke','Sleet/Hail/Freezing Rain','Blowing Sand/Dirt','Severe Crosswind',
```

```
df['ROADCOND'] = df['ROADCOND'].replace(['Dry','Wet','Unknown','Ice','Snow/Slush','Other','Standing Water','Sand/Mud/Dirt','Oil'],['1','2','3','4','5','6','7','8','9'])
```

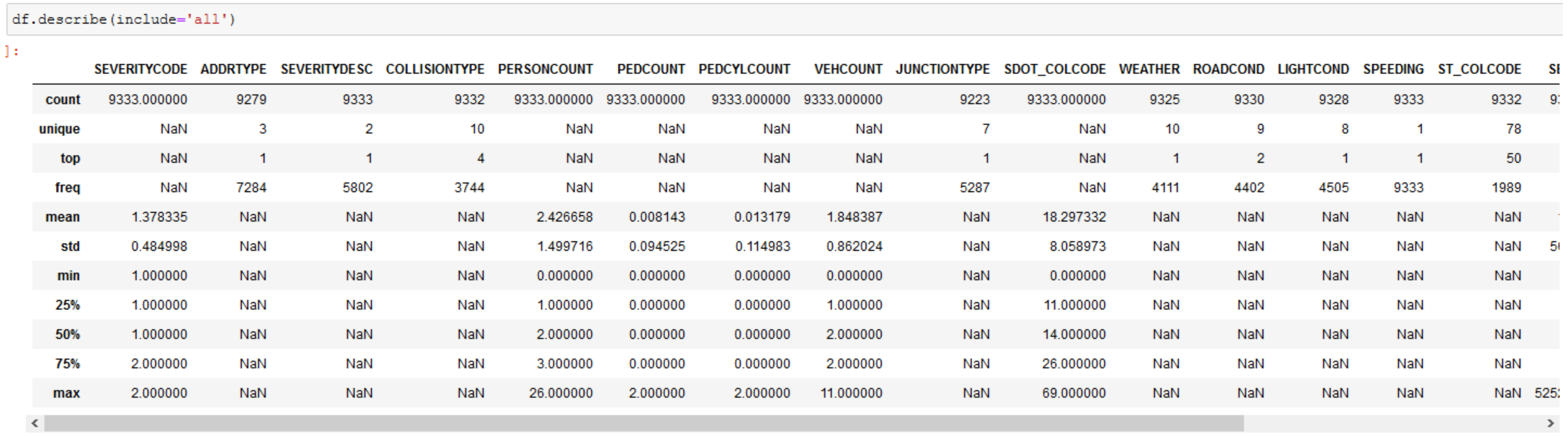
```
df['LIGHTCOND'] = df['LIGHTCOND'].replace(['Daylight','Dark - Street Lights On','Unknown','Dusk','Dawn','Dark - No Street Lights','Dark - Street Lights Off','Other','Dark - Unknow
```

```
df['SPEEDING'] = df['SPEEDING'].replace(['Y'],['1'])
```

```
df['JUNCTIONTYPE'] = df['JUNCTIONTYPE'].replace(['Mid-Block (not related to intersection)','At Intersection (intersection related)','Mid-Block (but intersection related)','Drivewa
```

```
df['HITPARKEDCAR'] = df['HITPARKEDCAR'].replace(['Y','N'],['1','2'])
```

I Looked at the description of the remaining data and the statistics to determine if I should replace missing data with an average, frequency, or completely drop the missing data.

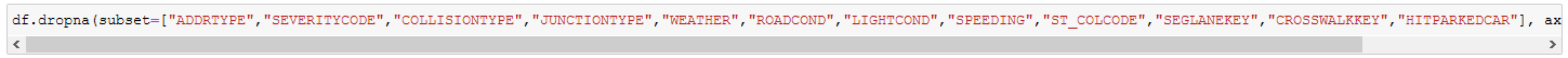


```
df.describe(include='all')
```

]:

	SEVERITYCODE	ADDRTYPE	SEVERITYDESC	COLLISIONTYPE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	JUNCTIONTYPE	SDOT_COLCODE	WEATHER	ROADCOND	LIGHTCOND	SPEEDING	ST_COLCODE	SI
count	9333.000000	9279	9333	9332	9333.000000	9333.000000	9333.000000	9333.000000	9223	9333.000000	9325	9330	9328	9333	9332	9
unique	NaN	3	2	10	NaN	NaN	NaN	NaN	7	NaN	10	9	8	1	78	
top	NaN	1	1	4	NaN	NaN	NaN	NaN	1	NaN	1	2	1	1	50	
freq	NaN	7284	5802	3744	NaN	NaN	NaN	NaN	5287	NaN	4111	4402	4505	9333	1989	
mean	1.378335	NaN	NaN	NaN	2.426658	0.008143	0.013179	1.848387	NaN	18.297332	NaN	NaN	NaN	NaN	NaN	
std	0.484998	NaN	NaN	NaN	1.499716	0.094525	0.114983	0.862024	NaN	8.058973	NaN	NaN	NaN	NaN	NaN	5
min	1.000000	NaN	NaN	NaN	0.000000	0.000000	0.000000	0.000000	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	
25%	1.000000	NaN	NaN	NaN	1.000000	0.000000	0.000000	1.000000	NaN	11.000000	NaN	NaN	NaN	NaN	NaN	
50%	1.000000	NaN	NaN	NaN	2.000000	0.000000	0.000000	2.000000	NaN	14.000000	NaN	NaN	NaN	NaN	NaN	
75%	2.000000	NaN	NaN	NaN	3.000000	0.000000	0.000000	2.000000	NaN	26.000000	NaN	NaN	NaN	NaN	NaN	
max	2.000000	NaN	NaN	NaN	26.000000	2.000000	2.000000	11.000000	NaN	69.000000	NaN	NaN	NaN	NaN	NaN	525

In the end I decided to drop the missing data as it constituted such a small part of each attribute as a whole.



```
df.dropna(subset=["ADDRTYPE","SEVERITYCODE","COLLISIONTYPE","JUNCTIONTYPE","WEATHER","ROADCOND","LIGHTCOND","SPEEDING","ST_COLCODE","SEGLANEKEY","CROSSWALKKEY","HITPARKEDCAR"], ax
```

Methodology

At this point the data is clean enough to be used for analysis. I will use supervised learning approach of KNN, Decision Tree, and Logistic Regression.

KNN - The K-Nearest Neighbors algorithm is a classification algorithm that takes a bunch of labeled points and uses them to learn how to label other points. This algorithm classifies cases based on their similarity to other cases. In K-Nearest   Neighbors, data points that are near each other are said to be neighbors.

KNN Method

```
#This methodology will focus on using attributes that may contribute or predict the severity of an accident./
#The values used are, addrtype, personcount, vehcount, junctiontype, weather, roadcond, lightcond, speeding to predict the severitcode.
#The targe, severitycode, has two possible values that correspond to accident severity, Basic Property Damage or Injury Collision
#My objective is to build a classifier, to predict the accident severity using K nearest neighbour.
```

```
df.columns
```

```
]: Index(['SEVERITYCODE', 'ADDRTYPE', 'PERSONCOUNT', 'VEHCOUNT', 'JUNCTIONTYPE',
          'WEATHER', 'ROADCOND', 'LIGHTCOND', 'SPEEDING'],
         dtype='object')
```

```
#Convert Pandas dataframe to Numpy array
```

```
X = df[['ADDRTYPE', 'PERSONCOUNT', 'VEHCOUNT', 'JUNCTIONTYPE',
       'WEATHER', 'ROADCOND', 'LIGHTCOND', 'SPEEDING']] .values  #.astype(float)
X[0:5]
```

```
]: array([['2', 3, 2, '2', '1', '1', '1', '1'],
          ['1', 1, 1, '1', '2', '2', '2', '1'],
          ['1', 4, 4, '3', '1', '1', '1', '1'],
          ['1', 2, 2, '1', '2', '2', '1', '1'],
          ['2', 2, 2, '2', '1', '1', '1', '1']], dtype=object)
```

```
#Labels
```

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

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

```
#Normalize the Data
```

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

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype object was converted to float64 by StandardS
caler.
  warnings.warn(msg, DataConversionWarning)
```

```
]: array([[ 1.88271082,  0.37794197,  0.17043397,  0.29672887, -0.82338369,
          -0.75637945, -0.67495399,  0.        ],
         [-0.51981008, -0.9558087 , -0.98844136, -0.73968014,  0.13597832,
           0.27101362,  0.21120026,  0.        ],
         [-0.51981008,  1.0448173 ,  2.48818462,  1.33313788, -0.82338369,
          -0.75637945, -0.67495399,  0.        ],
         [-0.51981008, -0.28893337,  0.17043397, -0.73968014,  0.13597832,
           0.27101362, -0.67495399,  0.        ],
         [ 1.88271082, -0.28893337,  0.17043397,  0.29672887, -0.82338369,
          -0.75637945, -0.67495399,  0.        ]])
```

```
#Split the Data into a train and test 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:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (7354, 8) (7354,)
Test set: (1839, 8) (1839,)
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
#Training
```

```
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

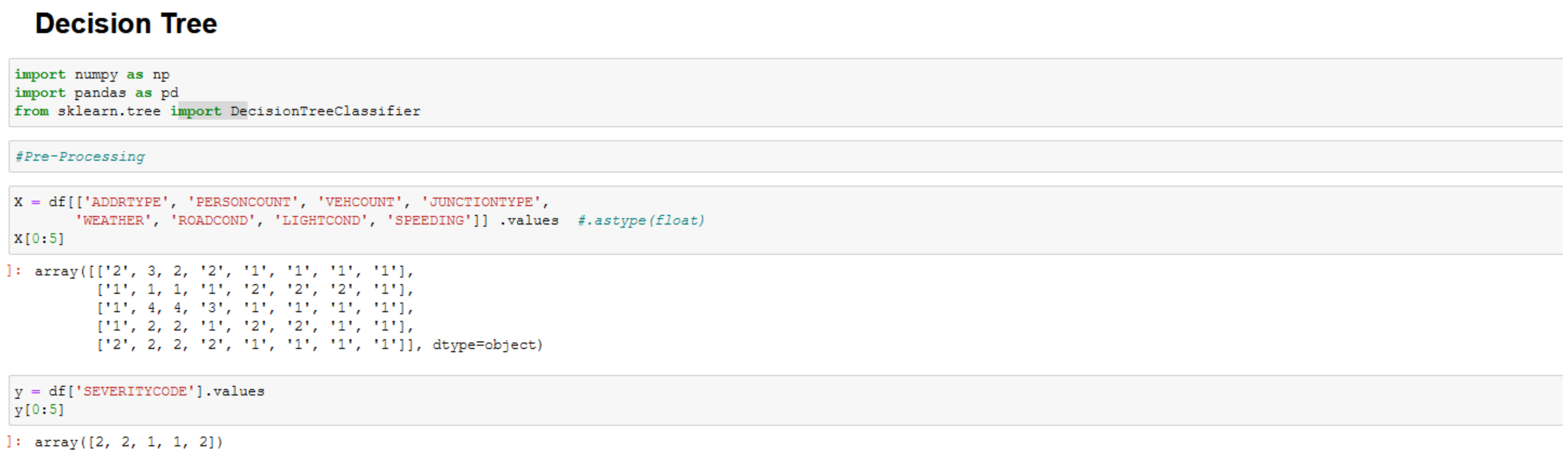
```
]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
               metric_params=None, n_jobs=None, n_neighbors=4, p=2,
               weights='uniform')
```

```
#Predicting
```

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

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

Decision Tree - Decision trees are built using recursive partitioning to classify the data. The algorithm chooses the most predictive feature to split the data on. What is important in making a decision tree, is to determine which attribute is the best or more predictive to split data based on the feature.



Decision Tree

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

```
#Pre-Processing
```

```
X = df[['ADDRTYPE', 'PERSONCOUNT', 'VEHCOUNT', 'JUNCTIONTYPE',
       'WEATHER', 'ROADCOND', 'LIGHTCOND', 'SPEEDING']] .values  #.astype(float)
X[0:5]
```

```
]: array([['2', 3, 2, '2', '1', '1', '1', '1'],
          ['1', 1, 1, '1', '2', '2', '2', '1'],
          ['1', 4, 4, '3', '1', '1', '1', '1'],
          ['1', 2, 2, '1', '2', '2', '1', '1'],
          ['2', 2, 2, '2', '1', '1', '1', '1']], dtype=object)
```

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

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

Decision Tree

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

```
#Pre-Processing
```

```
X = df[['ADDRTYPE', 'PERSONCOUNT', 'VEHCOUNT', 'JUNCTIONTYPE',
       'WEATHER', 'ROADCOND', 'LIGHTCOND', 'SPEEDING']] .values  #.astype(float)
X[0:5]
```

```
]: array([['2', 3, 2, '2', '1', '1', '1', '1'],
          ['1', 1, 1, '1', '2', '2', '2', '1'],
          ['1', 4, 4, '3', '1', '1', '1', '1'],
          ['1', 2, 2, '1', '2', '2', '1', '1'],
          ['2', 2, 2, '2', '1', '1', '1', '1']], dtype=object)
```

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

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

```python
#Setting up the Tree/Splitting the Data
```

```python
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3, random_state=3)
```

```python
print("X_trainset:", X_trainset.shape)
print("y_trainset:", y_trainset.shape)
```

```
X_trainset: (6435, 8)
y_trainset: (6435,)
```

```python
print("X_testset:", X_testset.shape)
print("y_testset:", y_testset.shape)
```

```
X_testset: (2758, 8)
y_testset: (2758,)
```

```python
#Modeling the Data
```

```python
accidentTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
accidentTree
```

```
]: 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')
```

```python
accidentTree.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')
```

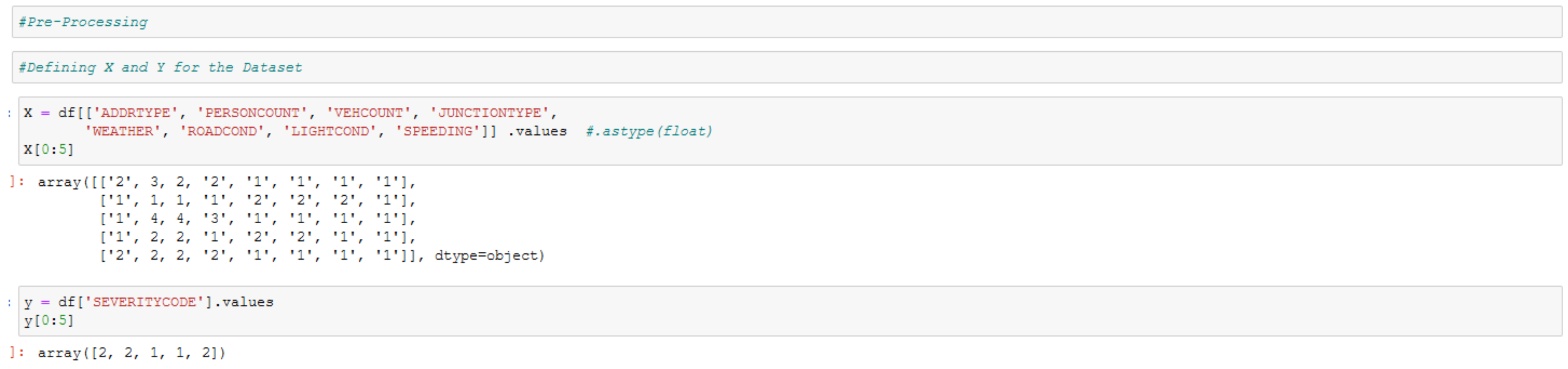
```python
#Predict on the Test Dataset
```

```python
accidentTree = accidentTree.predict(X_testset)
```

```python
print (accidentTree [0:5])
print (y_testset [0:5])
```

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

Logistic Regression - Logistic regression is a statistical and machine learning technique for classifying records of a dataset based on the values of the input fields. In logistic regression, we use one or more independent variables to predict an outcome. In logistic regression, we predict a variable which is binary. In logistic regression independent variables should be continuous. If categorical, they should be dummy or indicator coded. This means we have to transform them to some
continuous value.



```
#Pre-Processing
```

```
#Defining X and Y for the Dataset
```

```
: X = df[['ADDRTYPE', 'PERSONCOUNT', 'VEHCOUNT', 'JUNCTIONTYPE',
         'WEATHER', 'ROADCOND', 'LIGHTCOND', 'SPEEDING']] .values  #.astype(float)
  X[0:5]
```

```
]: array([['2', 3, 2, '2', '1', '1', '1', '1'],
          ['1', 1, 1, '1', '2', '2', '2', '1'],
          ['1', 4, 4, '3', '1', '1', '1', '1'],
          ['1', 2, 2, '1', '2', '2', '1', '1'],
          ['2', 2, 2, '2', '1', '1', '1', '1']], dtype=object)
```

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

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

```
#Normalize the Data
```

```
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/validation.py:595: DataConversionWarning: Data with input dtype object was converted to float64 by StandardS
caler.
  warnings.warn(msg, DataConversionWarning)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype object was converted to float64 by StandardS
caler.
  warnings.warn(msg, DataConversionWarning)
```

```
: array([[ 1.88271082,  0.37794197,  0.17043397,  0.29672887, -0.82338369,
         -0.75637945, -0.67495399,  0.        ],
        [-0.51981008, -0.9558087 , -0.98844136, -0.73968014,  0.13597832,
          0.27101362,  0.21120026,  0.        ],
        [-0.51981008,  1.0448173 ,  2.48818462,  1.33313788, -0.82338369,
         -0.75637945, -0.67495399,  0.        ],
        [-0.51981008, -0.28893337,  0.17043397, -0.73968014,  0.13597832,
          0.27101362, -0.67495399,  0.        ],
        [ 1.88271082, -0.28893337,  0.17043397,  0.29672887, -0.82338369,
         -0.75637945, -0.67495399,  0.        ]])
```

```
#Split the Dataset into Train/Test
```

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

```
Train set: (7354, 8) (7354,)
Test set: (1839, 8) (1839,)
```

```
#Modeling
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

```
: LogisticRegression(C=0.01, 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)
```

```
#Predicitng
```

```
yhat = LR.predict(X_test)
yhat
```

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

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

```
: array([[0.39408398, 0.60591602],
         [0.5727948 , 0.4272052 ],
         [0.69583502, 0.30416498],
         ...,
         [0.65325331, 0.34674669],
         [0.39601753, 0.60398247],
         [0.57933837, 0.42066163]])
```

Results

The results for each methodology are as follows:

KNN

```
#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, yhat))
```

```
  Train set Accuracy:  0.6558335599673647
  Test set Accuracy:  0.601957585644372
```

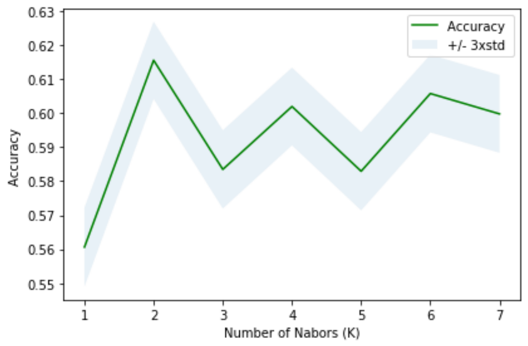
```
#Accuracy of KNN for differnt K's
```

```
Ks = 8
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([0.56063078, 0.61555193, 0.58346928, 0.60195759, 0.5829255 ,
         0.605764  , 0.59978249])
```

```
#Plotting Accuracy using best K from above
```

```
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 Nabors (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 0.6155519303969549 with k= 2
```

Decision Tree

```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, accidentTree))
```

```
DecisionTrees's Accuracy:  0.631979695431472
```

Logistic Regression

```
#Accuracy
```

```
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat)
```

```
: 0.6286025013594345
```

```
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

```
: 0.6533870111035934
```

```
from sklearn.metrics import f1_score
f1_score(y_test, yhat, average='weighted')
```

```
: 0.5661237417407249
```

Discussion

The accuracy of the predictions for all three methodologies yielded lower than expected results. The attributes that were chosen were based on the likelihood that they would be contributing indicators for the severity of an accident. Weather, road conditions, lighting conditions, and speeding I saw as major contributors to indicting severity. The attributes of address type, person count, vehicle count, and junction type were left in the data set with the thought that those would be present in number in the most severe accidents and would make the model more accurate.

Conclusion

In conclusion I feel that the full data set should have been used or further scraping of other accident severity data sets should have been collected and at least minimally explored for their veracity to be used as meaningful data. I am not sure that all the attributes left in the cleaned data were necessary. In the end I should have left the top four of weather, road conditions, light conditions and speeding and then determined if they resulted in over fitting.