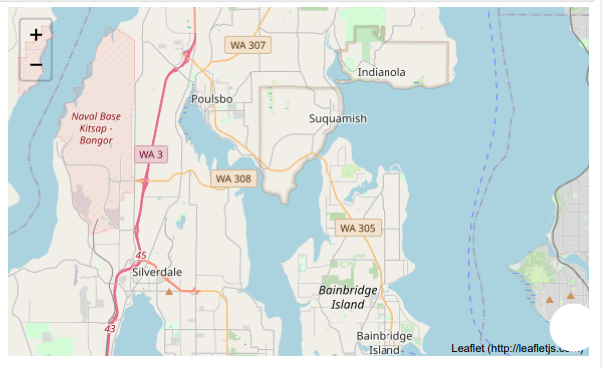
Tej Pratap



Road accidents cannot be stopped despite providing the best possible roads and intersections, however there are ways to reduce the impact of road accidents on road-users and the vehicles plying on the road. The incidence of accidental deaths has shown an increasing trend during the period 2005 – 2006 and then a decreasing trend up to 2020

Accident Severity Prediction using Machine Learning

Analysis with help of collision dataset

## Introduction | Business Understanding

The purpose of this Capstone Project is to help people to get awareness of accidents occurrences and causes of it every year in a city and helps 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.

According to preliminary estimates from National Highway Traffic Safety Administration (NHTSA), 36,120 people died in motor vehicle crashes in [2019](https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812946), down 1.2 percent from 36,560 in 2018. Reducing traffic accidents is an important public safety challenge around the world. Accident prediction is important for optimizing public transportation, enabling safer routes, and cost-effectively improving the transportation infrastructure, all in order to make the roads safer. It will help people making smart and efficient decision on selecting safe road routes to avoid accidents and be cautious.

The goal of ‘how to deal with the accidents data’- accident prediction is usually to provide a measure of the risk of accidents at different points in time and space. The severity of an accident is the label used to train the model which describes the fatality of an accident, and the proposed model can be used to identify where and when the risk of accident is significantly higher than average in order to take actions to reduce that risk.

This Capstone Project aim to analyse accident forecast basing on fatality of an accident. The severity of an accident is the label used to train the model which describes the fatality of an accident, and the proposed model can be used to identify where and when the risk of accident is significantly higher than average in order to take actions to reduce that risk.

**Data Description**

Data: Data –Collisions from Applied Data Science course

Data link: [https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv](%20%20https:/s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv)

In order to solve this probblem, that includes a collision prevention, meaning, preventing a potential unsafe road conditions from occurring in the first place. By recognizing the key factors that influence accident severity, the solution may be of great utility to various Government Departments/Authorities. The results of analysis and modeling can be used by these Departments to take appropriate measures to reduce accident impact and thereby improve traffic safety.

In order to mitigate the impact of data size on analysis and prediction, we present a new dataset, by using labels encoding to covert the features to our desired data type. Timeframe: 2004 to Present. The data has 194674 rows and 37 columns with wide range of attributes including Location, Severity Code, Vehicle Count, Injuries, Fatalities, Junction Type, Person Count, Weather, Road Condition, collision type, address type, speeding, Collisions type etc.

Our predictor or target variable will be 'SEVERITYCODE' because it is used measure the severity of an accident within the dataset. Attributes used to weigh the severity of an accident are 'WEATHER', 'ROADCOND' and 'LIGHTCOND' and few other attributes which do not correlate and have impact on target in a classification model.

In its original form, this data is not fit for analysis. Firstly, there are many columns that we will not use for this model. secondly, 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, also might need to do some feature engineering to improve the predictability of your model.

Accident Severity Prediction and Analysis

Seattle Road Map

!conda install -c conda-forge folium=0.5.0 --yes  
import folium  
from IPython.display import display  
LDN\_COORDINATES = (47.60,-122.33)  
seattle\_map = folium.Map(LDN\_COORDINATES, zoom\_start=11)  
# display map  
display(seattle\_map)

## In this project, we will be predicting the severity if an accident occurs based on some factors including Weather, Light, Road, Speeding, Inattention etc.

## Table of Contents

1. [Downloading and Prepping Data](#0)
2. [Feature Importance and Heat Map Visualization](#2)
3. [Prepaing and Exploring Datasets with *pandas*](#4)
4. [Visualizing Data using Folium](#6)
5. [Visualizing Data using Grouped Bar Charts](#8)
6. [Visualizing Data using Bubble Plots](#10)
7. [Preprocessing Data](#12)
8. [Splitting Data](#14)
9. [Training Data](#16)
10. [Scoring and Evaluation](#18)

Downloading and Prepping Data

### Loading Lbraries

Toolkits: The course heavily relies on [*pandas*](http://pandas.pydata.org/) and [**Numpy**](http://www.numpy.org/) for data wrangling, analysis, and visualization. The primary plotting library that we are exploring in the course is [Matplotlib](http://matplotlib.org/).

import pandas as pd  
import numpy as np  
import matplotlib as mpl  
import matplotlib.pyplot as plt  
%matplotlib inline  
import json  
import seaborn as sns

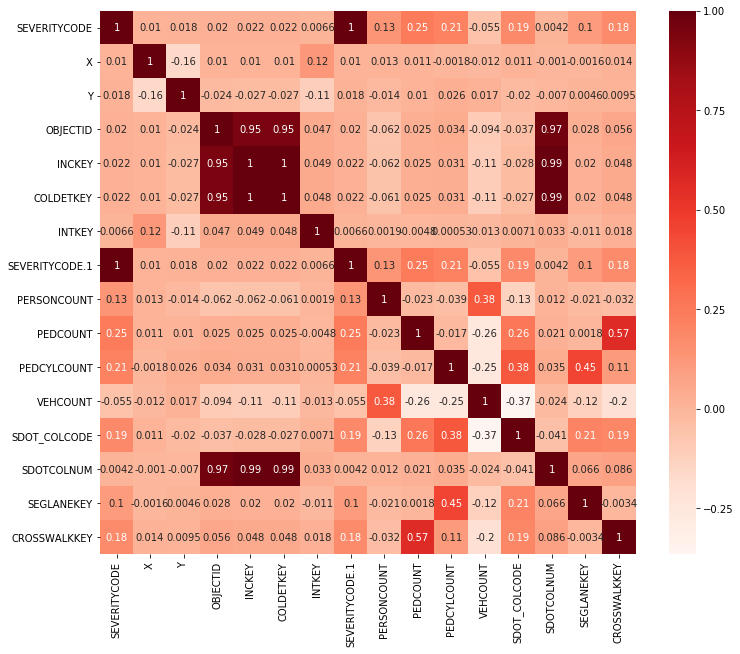
### Downloading the Datasets

df = pd.read\_csv('https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv')

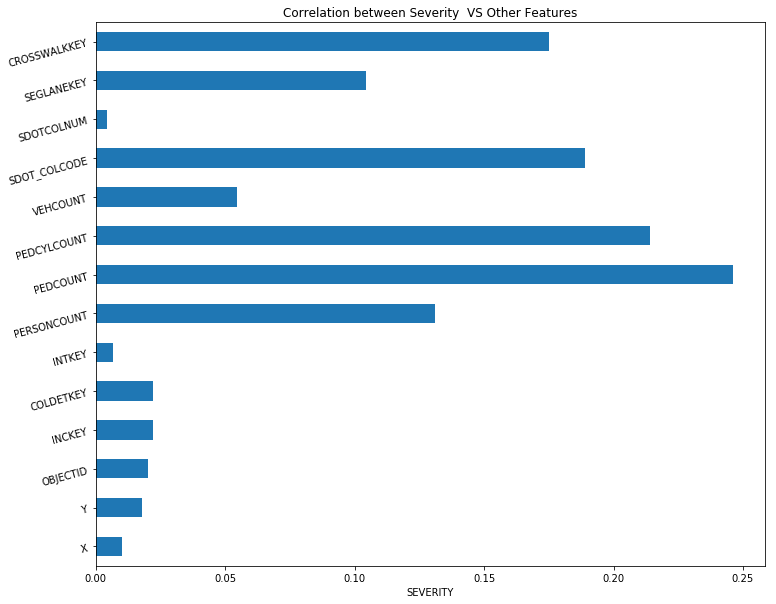
df.head(10)

Feature Importance and Heat Map Visualization

plt.figure(figsize=(12,10))  
cor = df.corr()  
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)  
plt.show()



plt.figure(figsize=(12,10))  
cor\_target = abs(cor["SEVERITYCODE"])  
#Selecting highly correlated features  
relevant\_features = cor\_target[cor\_target < 1]  
relevant\_features.plot.barh(rot=15, title="Correlation between Severity VS Other Features");  
plt.xlabel("SEVERITY")  
plt.show(block=True);



Prepaing and Exploring Datasets with *pandas*

to\_drop = [ 'OBJECTID', 'INCKEY', 'COLDETKEY',  
 'REPORTNO', 'STATUS', 'INTKEY',  
 'LOCATION', 'EXCEPTRSNCODE', 'EXCEPTRSNDESC',  
 'INCDATE', 'SDOT\_COLDESC', 'SDOTCOLNUM', 'SEVERITYCODE.1',  
 'ST\_COLDESC', 'SEGLANEKEY', 'CROSSWALKKEY'  
 ]  
  
df.drop(to\_drop,inplace=True,axis=1)

df.columns

df['INCDTTM'] = pd.to\_datetime(df['INCDTTM'], errors='coerce')  
df['Year']=df['INCDTTM'].dt.year  
df['Month']=df['INCDTTM'].dt.strftime('%b')  
df['Day']=df['INCDTTM'].dt.day  
df['Hour']=df['INCDTTM'].dt.hour  
df['Weekend']=df['INCDTTM'].dt.weekday  
df.drop(['INCDTTM'],axis = 1,inplace = True)  
df[0:5]

df.head

df.describe

df.describe(include = 'all')

df = df.replace(to\_replace = 'Unknown',value = np.NaN)  
df[['SPEEDING', 'INATTENTIONIND','UNDERINFL', 'PEDROWNOTGRNT']] = df[['SPEEDING', 'INATTENTIONIND', 'UNDERINFL', 'PEDROWNOTGRNT']].replace(to\_replace = [np.NaN, 0],value = 'N')  
df[['SPEEDING', 'INATTENTIONIND', 'UNDERINFL', 'PEDROWNOTGRNT']] = df[['SPEEDING', 'INATTENTIONIND', 'UNDERINFL', 'PEDROWNOTGRNT']].replace(to\_replace = 1,value = 'Y')  
df = df.replace(to\_replace = ' ',value = np.NaN)  
df.isna().sum()

df.describe(include = 'all')

df.columns

df = df[['SEVERITYCODE', 'X', 'Y', 'ADDRTYPE', 'COLLISIONTYPE', 'PERSONCOUNT',  
 'PEDCOUNT', 'PEDCYLCOUNT','SEVERITYDESC', 'VEHCOUNT', 'JUNCTIONTYPE', 'SDOT\_COLCODE',  
 'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND',  
 'PEDROWNOTGRNT', 'SPEEDING', 'ST\_COLCODE', 'HITPARKEDCAR', 'Year',  
 'Month', 'Day', 'Hour', 'Weekend']].dropna(how = 'any')

df.describe(include = 'all')

df.shape

df.dtypes

df.head(100)

Visualizing Data using Folium

data2 = df.copy(deep = True)  
  
  
speed = data2[data2['SPEEDING'] == "Y"]  
speed = speed[speed['Year'].isin([2017,2020])]  
speed\_sev1 = speed[speed['SEVERITYCODE'] == 1]  
speed\_sev2 = speed[speed['SEVERITYCODE'] == 2]  
speed\_sev1 = speed\_sev1[['X',"Y"]]  
speed\_sev2 = speed\_sev2[['X',"Y"]]  
speed.dropna(inplace = True)  
speed\_sev1.dropna(inplace = True)  
speed\_sev2.dropna(inplace = True)  
print(speed\_sev1.count())  
print(speed\_sev2.count())

incidents1 = folium.map.FeatureGroup()  
incidents2 = folium.map.FeatureGroup()  
#print(speed.count())  
for lat, lng, in zip(speed\_sev1.Y, speed\_sev1.X):  
 incidents1.add\_child(  
 folium.CircleMarker(  
 [lat, lng],  
 radius=0.5, # define how big you want the circle markers to be  
 color='red',  
 fill=True,  
 fill\_color='red',  
 fill\_opacity=0.6  
 )  
 )  
  
for lat, lng, in zip(speed\_sev2.Y, speed\_sev2.X):  
 incidents2.add\_child(  
 folium.CircleMarker(  
 [lat, lng],  
 radius=0.5, # define how big you want the circle markers to be  
 color='blue',  
 fill=True,  
 fill\_color='blue',  
 fill\_opacity=0.6  
 )  
 )  
seattle\_map1 = folium.Map(LDN\_COORDINATES, zoom\_start=11)  
seattle\_map1.add\_child(incidents1)  
seattle\_map1.add\_child(incidents2)

alcohol = data2[data2['UNDERINFL'] == "Y"]  
alcohol = alcohol[alcohol['Year'].isin([2017,2020])]  
  
  
alc\_sev1 = alcohol[alcohol['SEVERITYCODE'] == 1]  
alc\_sev2 = alcohol[alcohol['SEVERITYCODE'] == 2]  
alc\_sev1 = alc\_sev1[['X',"Y"]]  
alc\_sev2 = alc\_sev2[['X',"Y"]]  
  
alc\_sev1.dropna(inplace = True)  
alc\_sev2.dropna(inplace = True)  
print(alc\_sev1.count())  
print(alc\_sev2.count())

incidents1 = folium.map.FeatureGroup()  
incidents2 = folium.map.FeatureGroup()  
#print(speed.count())  
for lat, lng, in zip(alc\_sev1.Y, alc\_sev1.X):  
 incidents1.add\_child(  
 folium.CircleMarker(  
 [lat, lng],  
 radius=0.5, # define how big you want the circle markers to be  
 color='red',  
 fill=True,  
 fill\_color='red',  
 fill\_opacity=0.6  
 )  
 )  
  
for lat, lng, in zip(alc\_sev2.Y, alc\_sev2.X):  
 incidents2.add\_child(  
 folium.CircleMarker(  
 [lat, lng],  
 radius=0.5, # define how big you want the circle markers to be  
 color='blue',  
 fill=True,  
 fill\_color='blue',  
 fill\_opacity=0.6  
 )  
 )  
seattle\_map2 = folium.Map(LDN\_COORDINATES, zoom\_start=11)  
seattle\_map2.add\_child(incidents1)  
seattle\_map2.add\_child(incidents2)

billboard = data2[data2['INATTENTIONIND'] == "Y"]  
  
billboard = billboard[billboard['Year'].isin([2017,2020])]  
bill\_sev1 = billboard[billboard['SEVERITYCODE'] == 1]  
bill\_sev2 = billboard[billboard['SEVERITYCODE'] == 2]  
bill\_sev1 = bill\_sev1[['X',"Y"]]  
bill\_sev2 = bill\_sev2[['X',"Y"]]  
  
bill\_sev1.dropna(inplace = True)  
bill\_sev2.dropna(inplace = True)  
print(bill\_sev1.count())  
print(bill\_sev2.count())

incidents1 = folium.map.FeatureGroup()  
incidents2 = folium.map.FeatureGroup()  
#print(speed.count())  
for lat, lng, in zip(bill\_sev1.Y, bill\_sev1.X):  
 incidents1.add\_child(  
 folium.CircleMarker(  
 [lat, lng],  
 radius=0.5, # define how big you want the circle markers to be  
 color='red',  
 fill=True,  
 fill\_color='red',  
 fill\_opacity=0.6  
 )  
 )  
  
for lat, lng, in zip(bill\_sev2.Y, bill\_sev2.X):  
 incidents2.add\_child(  
 folium.CircleMarker(  
 [lat, lng],  
 radius=0.5, # define how big you want the circle markers to be  
 color='blue',  
 fill=True,  
 fill\_color='blue',  
 fill\_opacity=0.6  
 )  
 )  
seattle\_map3 = folium.Map(LDN\_COORDINATES, zoom\_start=11)  
seattle\_map3.add\_child(incidents1)  
seattle\_map3.add\_child(incidents2)

parked = data2[data2['HITPARKEDCAR'] == "Y"]  
parked = parked[parked['Year'].isin([2020])]  
  
  
park\_sev1 = parked[parked['SEVERITYCODE'] == 1]  
park\_sev2 = parked[parked['SEVERITYCODE'] == 2]  
park\_sev1 = park\_sev1[['X',"Y"]]  
park\_sev2 = park\_sev2[['X',"Y"]]  
  
park\_sev1.dropna(inplace = True)  
park\_sev2.dropna(inplace = True)  
print(park\_sev1.count())  
print(park\_sev2.count())

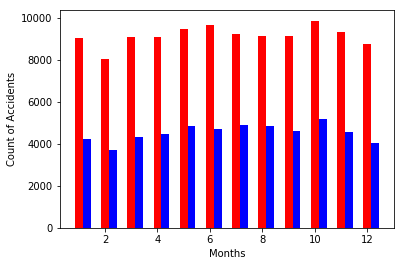
incidents1 = folium.map.FeatureGroup()  
incidents2 = folium.map.FeatureGroup()  
#print(speed.count())  
for lat, lng, in zip(park\_sev1.Y, park\_sev1.X):  
 incidents1.add\_child(  
 folium.CircleMarker(  
 [lat, lng],  
 radius=0.5, # define how big you want the circle markers to be  
 color='red',  
 fill=True,  
 fill\_color='red',  
 fill\_opacity=0.6  
 )  
 )  
  
for lat, lng, in zip(park\_sev2.Y, park\_sev2.X):  
 incidents2.add\_child(  
 folium.CircleMarker(  
 [lat, lng],  
 radius=0.5, # define how big you want the circle markers to be  
 color='blue',  
 fill=True,  
 fill\_color='blue',  
 fill\_opacity=0.6  
 )  
 )  
seattle\_map4 = folium.Map(LDN\_COORDINATES, zoom\_start=11)  
seattle\_map4.add\_child(incidents1)  
seattle\_map4.add\_child(incidents2)

Visualizing Data using Grouped Bar Charts

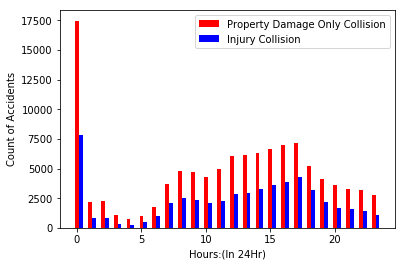
data3 = data2[data2['Year'] != 2020]  
data3['Month'] = data3['Month'].replace(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

years = data3['Month'].unique()

width = 0.3  
for year in years:  
 X2 = data3[data3['Month'] == year]  
 val1 = X2[X2['SEVERITYCODE'] == 1 ].shape[0]  
 val2 = X2[X2['SEVERITYCODE'] == 2 ].shape[0]  
 plt.bar(year, val1, width = width, color = 'red')  
 # fig , ax = plt.subplots()  
 # df.plot.bar(x = 'Month', ax = ax)  
 plt.bar( year + width, val2, width = width,color = 'blue')  
   
plt.xlabel('Months')  
plt.ylabel('Count of Accidents')  
# plt.legend(X2['SEVERITYDESC'].unique())  
plt.show()



years = data2['Hour'].unique()  
width = 0.3  
for year in years:  
 X2 = data2[data2['Hour'] == year]  
 val1 = X2[X2['SEVERITYCODE'] == 1 ].shape[0]  
 val2 = X2[X2['SEVERITYCODE'] == 2 ].shape[0]  
 plt.bar(int(year), val1, width = width, color = 'red')  
 plt.bar(int(year) + width, val2, width = width,color = 'blue')  
  
plt.xlabel('Hours:(In 24Hr)')  
plt.ylabel('Count of Accidents')  
plt.legend(X2['SEVERITYDESC'].unique())  
plt.show()



years = data2['Weekend'].unique()  
width = 0.3  
for year in years:  
 X2 = data2[data2['Weekend'] == year]  
 val1 = X2[X2['SEVERITYCODE'] == 1 ].shape[0]  
 val2 = X2[X2['SEVERITYCODE'] == 2 ].shape[0]  
 plt.bar(int(year), val1, width = width, color = 'red')  
 plt.bar(int(year) + width, val2, width = width,color = 'blue')  
  
plt.xlabel('Day of the Week')  
plt.ylabel('Count of Accidents')  
#plt.legend(X2['SEVERITYDESC'].unique())  
plt.show()

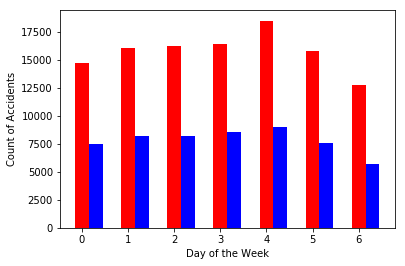
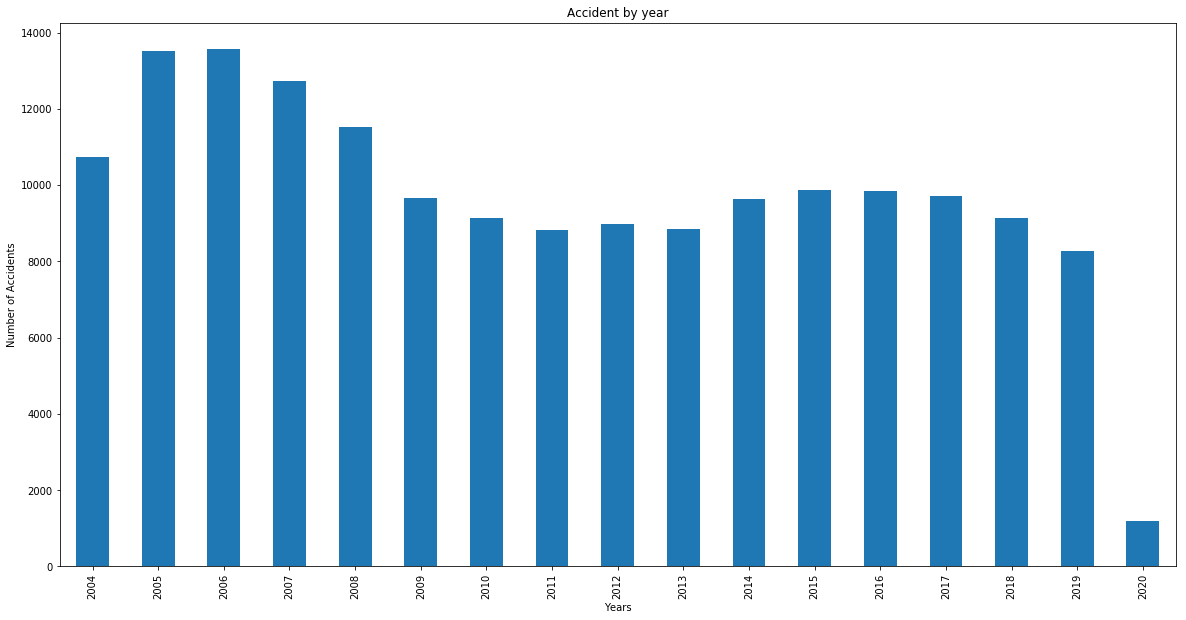


fig = plt.figure(figsize=(25,20))  
d = data2['Year'].value\_counts().sort\_index()  
d.plot(kind='bar',   
 stacked=False,  
 figsize=(20, 10), # pass a tuple (x, y) size  
 )  
  
  
plt.title('Accident by year')  
plt.ylabel('Number of Accidents')  
plt.xlabel('Years')  
  
plt.show()



Visualizing Data using Bubble Plots

d = df.pivot\_table(index = ['Month','SEVERITYCODE'],aggfunc = 'size')

li1 = d.values

li2 = list(d.index)

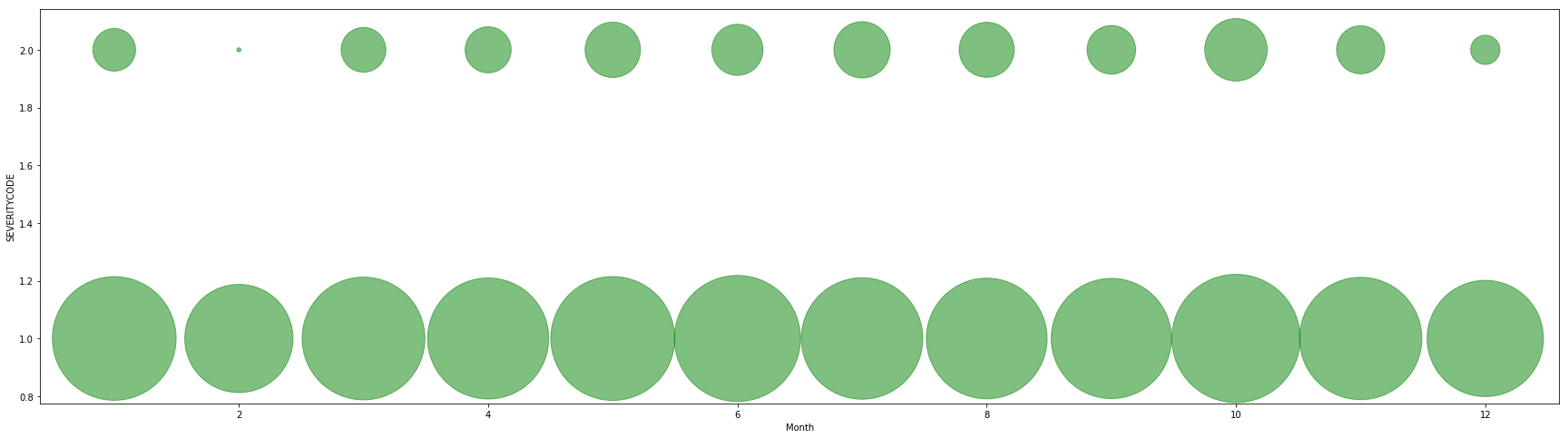
df1 = pd.DataFrame(li4,columns = ['Month', 'SEVERITYCODE'])

df1['No\_of\_Accident'] = li1  
df1

norm\_No\_of\_Accident = (df1['No\_of\_Accident'] - df1['No\_of\_Accident'].min()) / (df1['No\_of\_Accident'].max() - df1['No\_of\_Accident'].min())

df1['Month'] = df1['Month'].replace(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

ax0 = df1.plot(kind='scatter',  
 x='Month',  
 y= 'SEVERITYCODE',  
 figsize=(30, 8),  
 alpha=0.5, # transparency  
 color= 'green',  
 s= norm\_No\_of\_Accident\*20000 + 20 # pass in weights  
 )



Preprocessing Data

from sklearn.preprocessing import StandardScaler

df.drop('SEVERITYDESC' , axis =1)

df['X'] = df['X'].astype(float)  
df['Y'] = df['Y'].astype(float)  
df['LIGHTCOND'] = df['LIGHTCOND'].astype(str)  
df['ROADCOND'] = df['ROADCOND'].astype(str)  
df['WEATHER'] = df['WEATHER'].astype(str)  
df['SPEEDING'] = df['SPEEDING'].astype(str)  
df['COLLISIONTYPE'] = df['COLLISIONTYPE'].astype(str)  
df['JUNCTIONTYPE'] = df['JUNCTIONTYPE'].astype(str)  
df['ST\_COLCODE'] = df['ST\_COLCODE'].astype(str)  
df['INATTENTIONIND'] = df['INATTENTIONIND'].astype(str)  
df['UNDERINFL'] = df['UNDERINFL'].astype(str)  
df['HITPARKEDCAR'] = df['HITPARKEDCAR'].astype(str)

df.describe(include = 'all')

cols = ['SEVERITYCODE', 'X', 'Y', 'ADDRTYPE', 'COLLISIONTYPE', 'PERSONCOUNT',  
 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT', 'JUNCTIONTYPE', 'SDOT\_COLCODE',  
 'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND',  
 'PEDROWNOTGRNT', 'SPEEDING', 'ST\_COLCODE', 'HITPARKEDCAR', 'Year',  
 'Month', 'Day', 'Hour', 'Weekend']  
df[cols]= df[cols].apply(le.fit\_transform)

df.dtypes

df.head(100)

df.describe

accident = df.to\_csv('data\_collisions.csv')

y = df['SEVERITYCODE']

df.shape

df.columns

y.shape

X = df.drop(['SEVERITYCODE', 'X', 'Y', 'SEVERITYDESC'], axis = 1)

X.shape

X = StandardScaler().fit(X).transform(X.astype(float))

X[0:5]

Splitting Data

from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y,test\_size=0.2, random\_state=42)

print("Train Shape : ",X\_train.shape,y\_train.shape)

print("Train Shape : ",X\_test.shape,y\_test.shape)

Training Data

from sklearn.neighbors import KNeighborsClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.svm import SVC  
from xgboost import XGBClassifier

lr = LogisticRegression(C=0.01, solver='liblinear').fit(X\_train, y\_train)  
gbc = GradientBoostingClassifier(random\_state=4).fit(X\_train,y\_train)  
rfc = RandomForestClassifier(max\_depth=10, random\_state=4).fit(X\_train,y\_train)  
dt = DecisionTreeClassifier(criterion="entropy").fit(X\_train,y\_train)  
knn = KNeighborsClassifier(n\_neighbors = 2).fit(X\_train,y\_train)  
xgb = XGBClassifier(random\_state=4).fit(X\_train,y\_train)  
clf = SVC(gamma='auto').fit(X\_train, y\_train)  
  
print("Training Completed with different Models")

Scoring and Evaluation

from sklearn.metrics import jaccard\_similarity\_score  
from sklearn.metrics import f1\_score  
from sklearn.metrics import log\_loss  
from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report, confusion\_matrix  
import itertools  
def plot\_confusion\_matrix(cm, classes,  
 normalize=False,  
 title='Confusion matrix',  
 cmap=plt.cm.Blues):  
 """  
 This function prints and plots the confusion matrix.  
 Normalization can be applied by setting `normalize=True`.  
 """  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 print("Normalized confusion matrix")  
 else:  
 print('Confusion matrix, without normalization')  
  
 print(cm)  
  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
  
 fmt = '.2f' if normalize else 'd'  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, format(cm[i, j], fmt),  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.tight\_layout()  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')

### Logistic Regression

l = []  
l.append('Logistic Regression')   
print(l[0])  
yhat = lr.predict(X\_test)  
yhat\_prob = lr.predict\_proba(X\_test)  
yhat\_prob

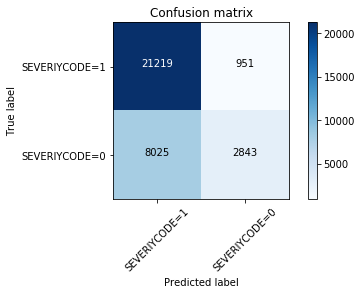
l.append(f1\_score(y\_test, yhat))  
l.append(jaccard\_similarity\_score(y\_test, yhat))  
l.append(log\_loss(y\_test,yhat))

Eval = [{'Model\_Name': l[0] ,'F1Score': l[1], 'Jaccard\_Similarity': l[2], 'Log\_Loss': l[3]}]

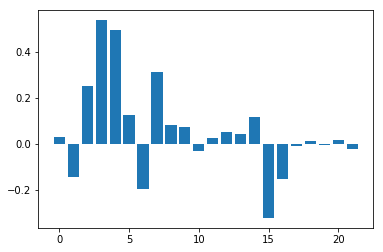
Eval\_score =pd.DataFrame.from\_dict(Eval)[list(Eval[0].keys())]

Eval\_score

# Compute confusion matrix  
cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[0,1])  
np.set\_printoptions(precision=2)  
  
  
# Plot non-normalized confusion matrix  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=['SEVERIYCODE=1','SEVERIYCODE=0'],normalize= False, title='Confusion matrix')



importance = lr.coef\_[0]  
# summarize feature importance  
for i,v in enumerate(importance):  
 print('Feature: %0d, Score: %.5f' % (i,v))  
# plot feature importance  
plt.bar([x for x in range(len(importance))], importance)  
plt.show()



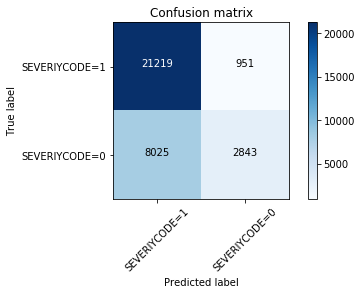
### Gradient Boosting Classifier

l[0] = 'Gradient Boosting Classifier'  
print(l[0])  
y\_hat = gbc.predict(X\_test)  
l[1] = f1\_score(y\_test,y\_hat, average='weighted')  
l[2] = jaccard\_similarity\_score(y\_test,y\_hat)  
l[3] = log\_loss(y\_test,y\_hat)

Eval\_score = Eval\_score.append({'Model\_Name': l[0] ,'F1Score': l[1], 'Jaccard\_Similarity': l[2], 'Log\_Loss': l[3]} , ignore\_index=True)

Eval\_score

# Compute confusion matrix  
cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[0,1])  
np.set\_printoptions(precision=2)  
  
  
# Plot non-normalized confusion matrix  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=['SEVERIYCODE=1','SEVERIYCODE=0'],normalize= False, title='Confusion matrix')



### Random Forest Classifier

l[0] = 'Random Forest Classifier'  
print(l[0])  
y\_hat = rfc.predict(X\_test)  
l[1] = f1\_score(y\_test,y\_hat, average='weighted')  
l[2] = jaccard\_similarity\_score(y\_test,y\_hat)  
l[3] = log\_loss(y\_test,y\_hat)

Eval\_score = Eval\_score.append({'Model\_Name': l[0] ,'F1Score': l[1], 'Jaccard\_Similarity': l[2], 'Log\_Loss': l[3]} , ignore\_index=True)

Eval\_score

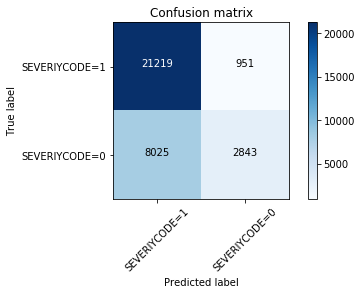
### Decision Tree Classifier

l[0] = 'Decision Tree Classifier'  
print(l[0])  
y\_hat = dt.predict(X\_test)  
l[1] = f1\_score(y\_test,y\_hat, average='weighted')  
l[2] = jaccard\_similarity\_score(y\_test,y\_hat)  
l[3] = log\_loss(y\_test,y\_hat)

Eval\_score = Eval\_score.append({'Model\_Name': l[0] ,'F1Score': l[1], 'Jaccard\_Similarity': l[2], 'Log\_Loss': l[3]} , ignore\_index=True)

Eval\_score

# Compute confusion matrix  
cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[0,1])  
np.set\_printoptions(precision=2)  
  
  
# Plot non-normalized confusion matrix  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=['SEVERIYCODE=1','SEVERIYCODE=0'],normalize= False, title='Confusion matrix')



!pip install graphviz

import graphviz  
from sklearn import tree  
print('Imported')

! pip install pydotplus  
import pydotplus

fig = plt.figure(figsize=(25,20))  
feature\_name= ['ADDRTYPE', 'COLLISIONTYPE', 'PERSONCOUNT',  
 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT', 'JUNCTIONTYPE', 'SDOT\_COLCODE',  
 'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND',  
 'PEDROWNOTGRNT', 'SPEEDING', 'ST\_COLCODE', 'HITPARKEDCAR', 'Year',  
 'Month', 'Day', 'Hour', 'Weekend']  
dot\_data = tree.export\_graphviz(dt,  
 feature\_names=feature\_name,  
 out\_file=None,  
 filled=True,  
 rounded=True)  
graph = pydotplus.graph\_from\_dot\_data(dot\_data)  
  
# colors = ('turquoise', 'orange')  
  
graph.write\_png('tree.png')

fig.savefig("decision\_tree.png")

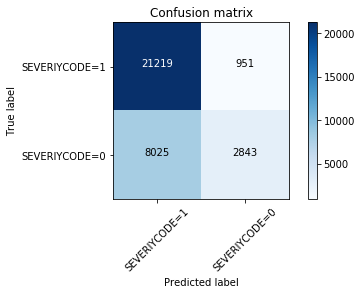
### XGBoost Classifier

l[0] = 'XGBOOST Classifier'  
print(l[0])  
y\_hat = dt.predict(X\_test)  
l[1] = f1\_score(y\_test,y\_hat, average='weighted')  
l[2] = jaccard\_similarity\_score(y\_test,y\_hat)  
l[3] = log\_loss(y\_test,y\_hat)

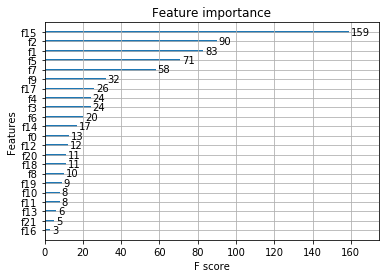
Eval\_score = Eval\_score.append({'Model\_Name': l[0] ,'F1Score': l[1], 'Jaccard\_Similarity': l[2], 'Log\_Loss': l[3]} , ignore\_index=True)

Eval\_score

# Compute confusion matrix  
cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[0,1])  
np.set\_printoptions(precision=2)

  
  
  
# Plot non-normalized confusion matrix  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=['SEVERIYCODE=1','SEVERIYCODE=0'],normalize= False, title='Confusion matrix')

from xgboost import plot\_importance  
print('Feature Importance = ',xgb.feature\_importances\_)  
plot\_importance(xgb)  
plt.show()



### KNeighbours Classifier

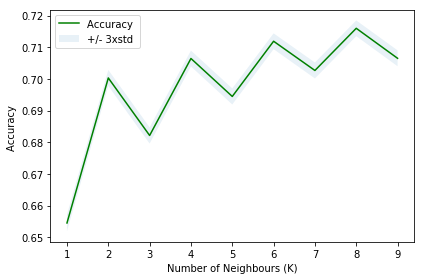
l[0] = 'KNeighbours Classifier'  
y\_hat = knn.predict(X\_test)  
l[1] = f1\_score(y\_test,y\_hat, average='weighted')  
l[2] = jaccard\_similarity\_score(y\_test,y\_hat)  
l[3] = log\_loss(y\_test,y\_hat)

Eval\_score = Eval\_score.append({'Model\_Name': l[0] ,'F1Score': l[1], 'Jaccard\_Similarity': l[2], 'Log\_Loss': l[3]} , ignore\_index=True)

Eval\_score

# Compute confusion matrix  
cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[0,1])  
np.set\_printoptions(precision=2)  
  
  
# Plot non-normalized confusion matrix  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=['SEVERIYCODE=1','SEVERIYCODE=0'],normalize= False, title='Confusion matrix')

Ks = 10  
mean\_acc = np.zeros((Ks-1))  
std\_acc = np.zeros((Ks-1))  
ConfustionMx = [];  
for n in range(1,Ks):  
   
 #Train Model and Predict   
 knn = KNeighborsClassifier(n\_neighbors = n).fit(X\_train,y\_train)  
 yhat=knn.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])  
   
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 Neighbours (K)')  
plt.tight\_layout()  
plt.show()



### Support Vector Machine

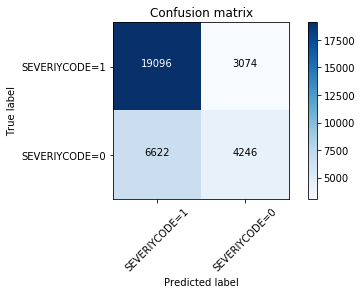
l[0] = 'Support Vector Machine'  
y\_hat = clf.predict(X\_test)  
l[1] = f1\_score(y\_test,y\_hat, average='weighted')  
l[2] = jaccard\_similarity\_score(y\_test,y\_hat)  
l[3] = log\_loss(y\_test,y\_hat)

Eval\_score = Eval\_score.append({'Model\_Name': l[0] ,'F1Score': l[1], 'Jaccard\_Similarity': l[2], 'Log\_Loss': l[3]} , ignore\_index=True)

Eval\_score

print(confusion\_matrix(y\_test, yhat, labels=[1,0]))

# Compute confusion matrix  
cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[0,1])  
np.set\_printoptions(precision=2)  
  
  
# Plot non-normalized confusion matrix  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=['SEVERIYCODE=1','SEVERIYCODE=0'],normalize= False, title='Confusion matrix')



PROJECT END