PREDICTING CAR ACCIDENT'SEVERITY

IBM Applied Data Science Capstone Project

JEFRY HAMJAYA

1. INTRODUCTION / BUSINESS PROBLEM

In a big city where car accidents happen all the time, it can be a challenge to deploy necessary number or type of personnel on time with the limited numbers of personnel on our disposal.

The idea is to classify the severity of a car accident, in this case we will use two level of severity, 1 for Property Damage Only Collision and 2 for Injury Collision. The severity prediction will be based on the information received at the time an accident is reported.

With this simplification of early accident classification, the Dispatch Center can decide which personnel should be dispatched for the accident. For example, for accident with severity of 1 Property Damage Only Collision, the healthcare personnels are not needed on site, and they can be allocated to another injury related accident.

2.
DATA

The data that will be used is to approach the problem is the sample data set from:

https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv

This is a Seattle's car accident data from 2004 to 2020 which contains a number of information for each accident, such as the time, location, and the number of people / vehicle involved in each accident. Based on this historical data, we will try to build a model that is able to predict the severity of an accident based on the initial data collected from the accident site.

The data contains 1 target column & 37 feature columns with a total of 194673 rows of data.

Target Column:

'SEVERITYCODE'

Feature Columns:

'X', 'Y', 'OBJECTID', 'INCKEY', 'COLDETKEY', 'REPORTNO', 'STATUS', 'ADDRTYPE', 'INTKEY', 'LOCATION', 'EXCEPTRSNCODE', 'EXCEPTRSNDESC', 'SEVERITYCODE.1', 'SEVERITYDESC', 'COLLISIONTYPE', 'PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT', 'INCDATE', 'INCDTTM', 'JUNCTIONTYPE', 'SDOT_COLCODE', 'SDOT_COLDESC', 'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND', 'PEDROWNOTGRNT', 'SDOTCOLNUM', 'SPEEDING', 'ST_COLCODE', 'ST_COLDESC', 'SEGLANEKEY', 'CROSSWALKKEY', 'HITPARKEDCAR'

The explanation for each column can be found in:

https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Metadata.pdf

We exclude the columns that are entered by the state as they won't be available in the initial report ('PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT', 'INJURIES', 'SERIOUSINJURIES', 'FATALITIES').

We also exclude the 'LOCATION' column as this is a free text column and is already represented by the coordinates ('X', 'Y').

These are the columns that we'll use for our model building:

X	Double	Longitude
Y	Double	Latitude
ADDRTYPE	Text, 12	Collision address type: Alley, Block, Intersection
INTKEY	Double	Key that corresponds to the intersection associated with a collision
PERSONCOUNT	Double	The total number of people involved in the collision
SDOT_COLCODE	Text, 10	A code given to the collision by SDOT.
INATTENTIONIND	Text, 1	Whether or not collision was due to inattention. (Y/N)
UNDERINFL	Text, 10	Whether or not a driver involved was under the influence of drugs or alcohol.

WEATHER	Text, 300	A description of the weather conditions during the time of the collision.
ROADCOND	Text, 300	The condition of the road during the collision.
LIGHTCOND	Text, 300	The light conditions during the collision.
SPEEDING	Text, 1	Whether or not speeding was a factor in the collision. (Y/N)
ST_COLCODE	Text, 10	A code provided by the state that describes the collision.
SEGLANEKEY	Long	A key for the lane segment in which the collision occurred.
CROSSWALKKEY	Long	A key for the crosswalk at which the collision occurred.
HITPARKEDCAR	Text, 1	Whether or not the collision involved hitting a parked car. (Y/N)

3. METHODOLOGY

IMPORTING THE DATA

We start by importing the data in the notebook and importing some necessary packages into it.

```
path = "./DATA/Data-Collisions.csv"
```

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

```
data = pd.read_csv(path)
```

```
data.head()
```

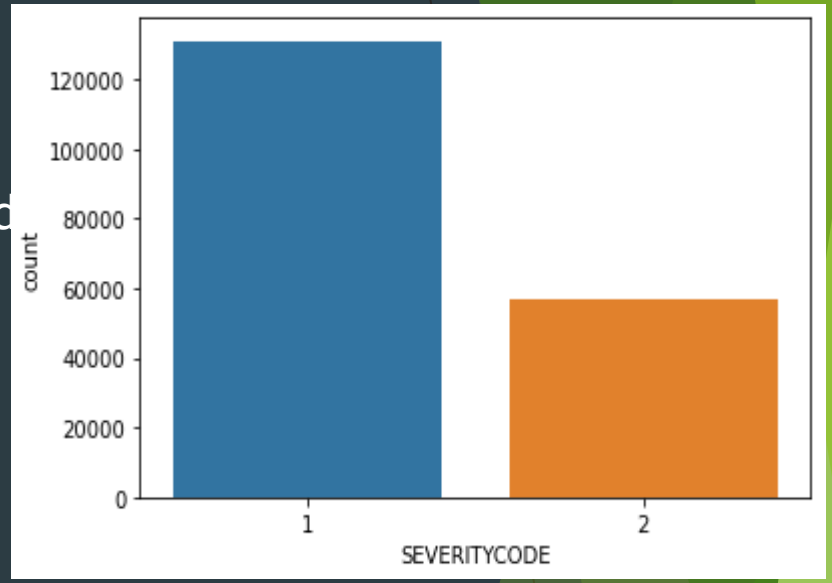
	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT	SDOTCOLNUM	SPEEDING	ST_COLCODE	ST_COLDESC	SEGLANEKEY	CROSSWALKKEY	HITPARI
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight	NaN	NaN	NaN	10	Entering at angle	0	0	
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...	0	0	
2	1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight	NaN	4323031.0	NaN	32	One parked--one moving	0	0	
3	1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight	NaN	NaN	NaN	23	From same direction - all others	0	0	
4	2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight	NaN	4028032.0	NaN	10	Entering at angle	0	0	

PRELIMINARY TARGET CHECK

Then, we'll check the target column SEVERITYCODE.

Using countplot, we can see that the data is very unbalanced, and towards SEVERITYCODE = 1. Training our model with this kind recommended due to the bias.

We'll address this issue when we start to train our model later.



PRELIMINARY FEATURES CHECK

Next, we examine the features that we decided to use one by one to see their significance on the Severity value.

X, Y, ADDRTYPE, INTKEY, PERSONCOUNT, SDOT_COLCODE,
INATTENTIONIND, UNDERINFL, WEATHER, ROADCOND, LIGHTCOND,
SPEEDING, ST_COLCODE, SEGLANEKEY, CROSSWALKKEY, HITPARKEDCAR

X, Y

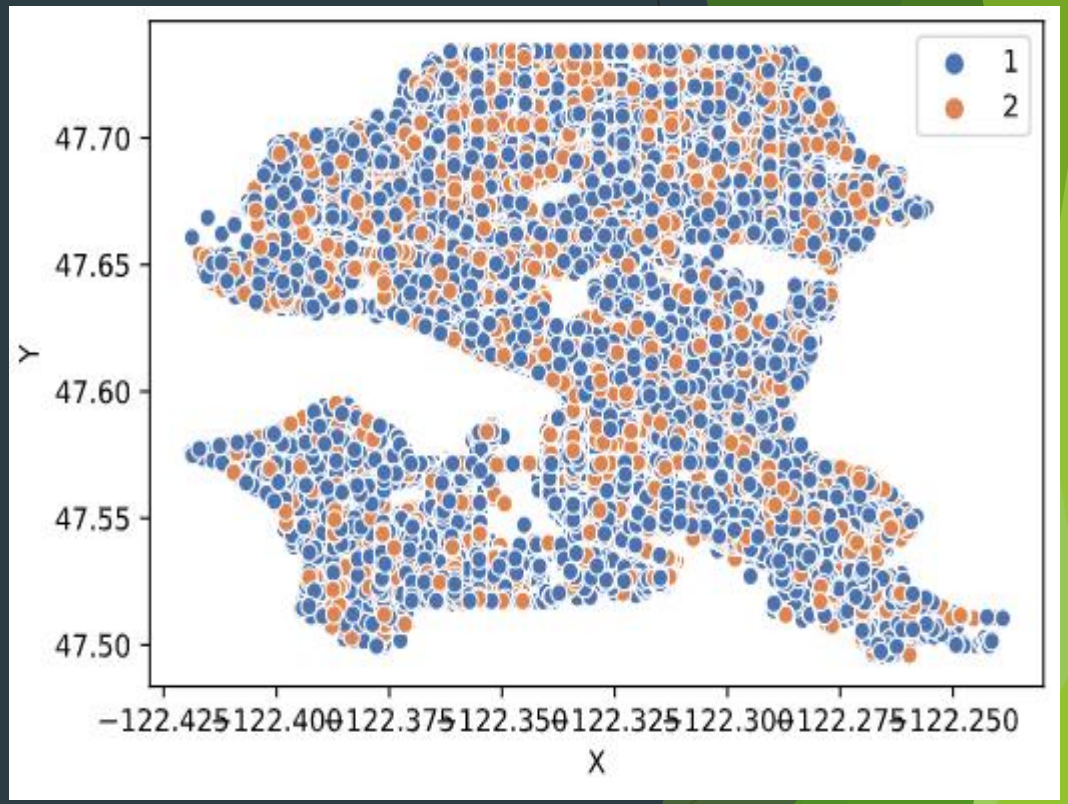
By plotting X and Y, we can see that there is no clear separation between areas with SEVERITYCODE = 1 and 2.

Also there are 5334 rows with missing X and Y information.

```
pre_data[['X', 'Y']].isna().sum()
```

```
X     5334
Y     5334
dtype: int64
```

We can conclude that we can drop these columns as they pose little significance for predicting SEVERITYCODE values.



INTKEY

INTKEY refers to intersection number related to the accident. Since more than half of the information are missing, we'll drop this column

```
pre_data['INTKEY'].isna().sum()
```

```
127677
```

```
pre_data.drop('INTKEY', axis = 1, inplace = True)
```

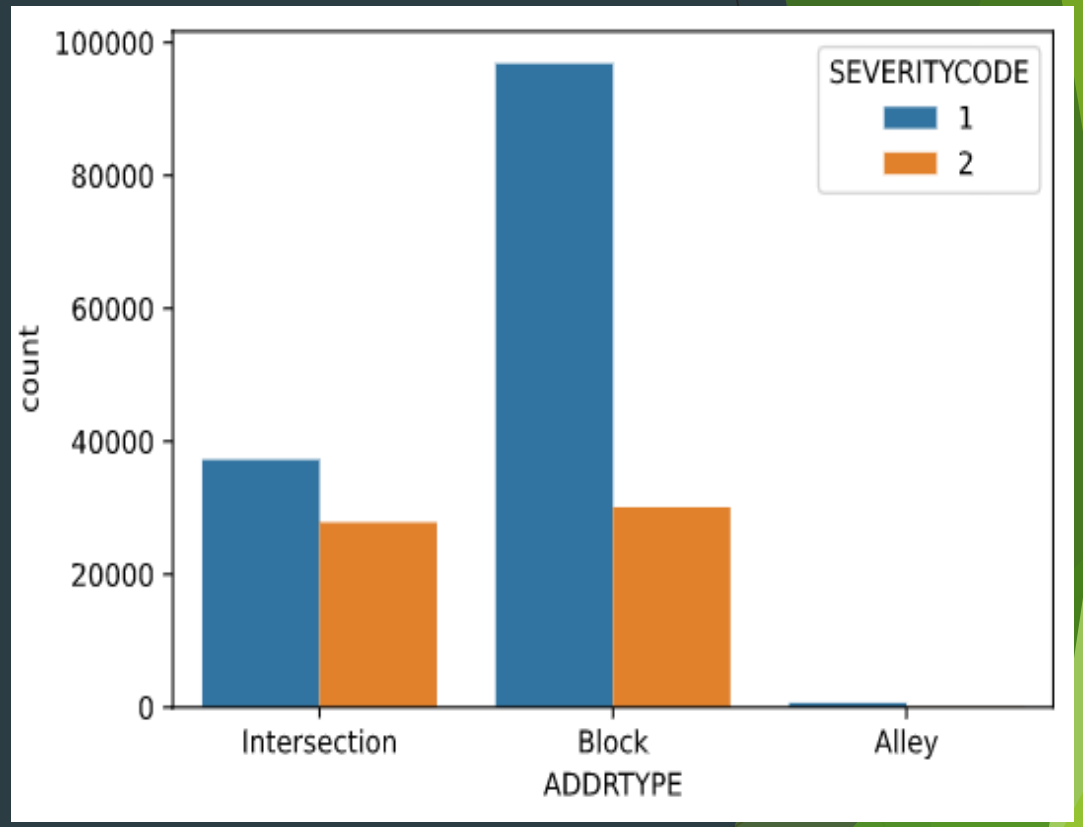
ADDRTYPE

There is a large number of accidents where the ADDRTYPE is 'Block' and most of them are under SEVERITYCODE = 1.

There are 1926 rows with missing ADDRTYPE that we'll drop from the dataframe.

```
pre_data['ADDRTYPE'].value_counts(dropna = False)
```

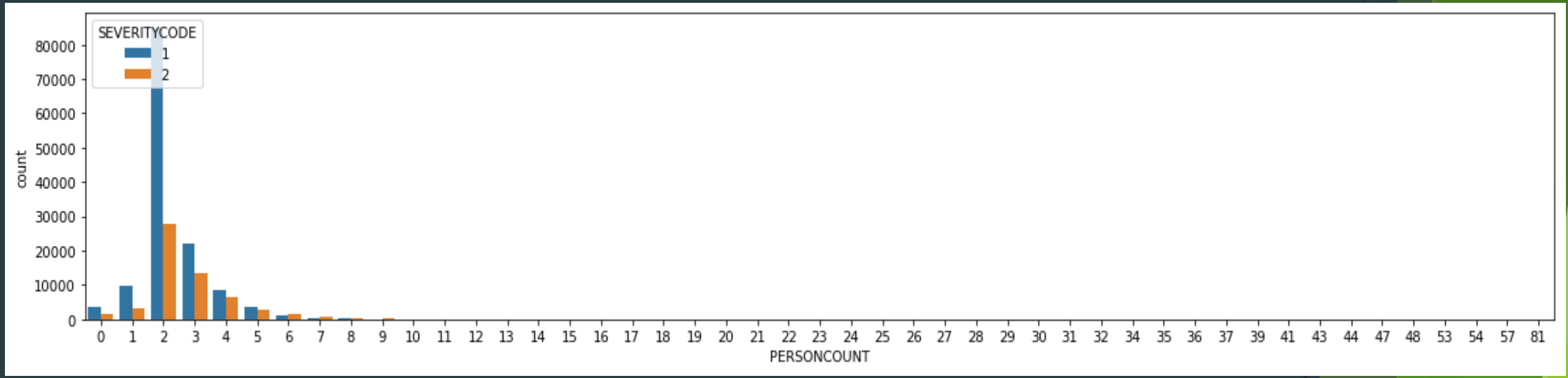
```
Block           126926
Intersection     65070
NaN               1926
Alley              751
Name: ADDRTYPE, dtype: int64
```



PERSONCOUNT

We can see that most car accidents involves two people.

Nothing to be done on this column as everything is in place and no missing value.



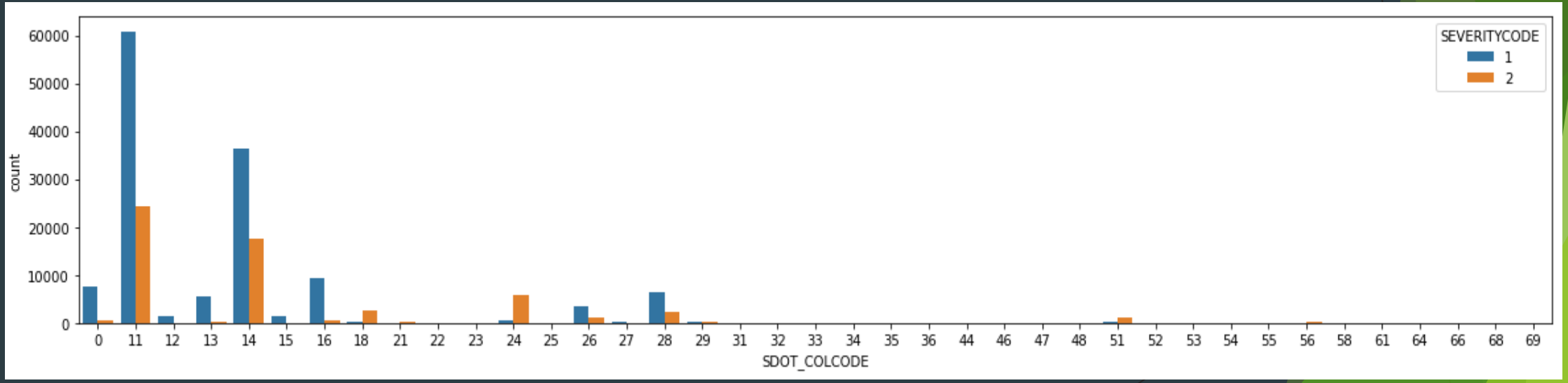
SDOT_COLCODE

The majority of accidents happen with SDOT_COLCODE = 11 & 14

SDOT_COLCODE 11 - motor vehicle struck another motor vehicle in front end

SDOT_COLCODE 14 - motor vehicle struck another motor vehicle in rear end

The data in this column are in order and no missing value either.



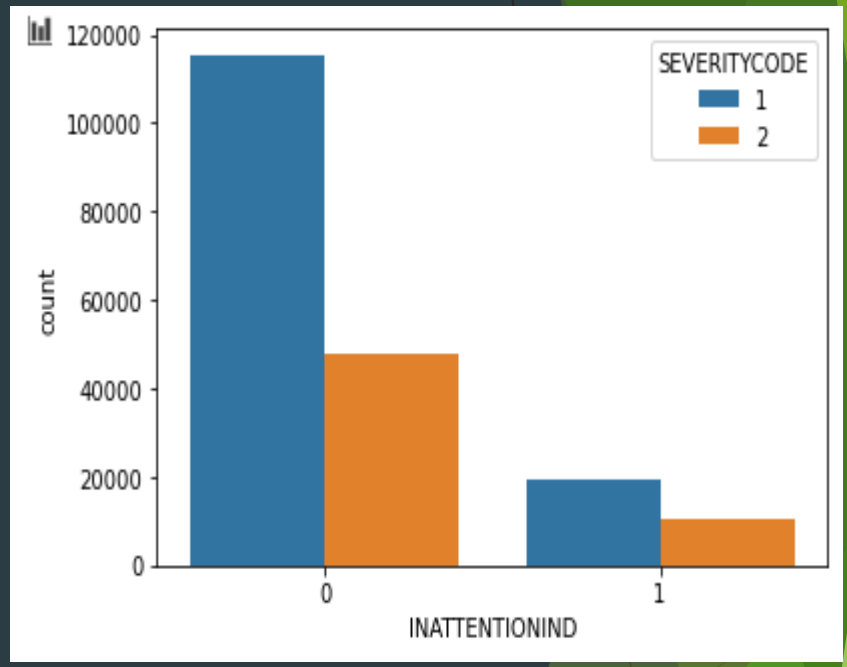
INATTENTIONIND

Since this column contains NaN and ‘Y’, we’ll need to convert them to binary value of 1 using replace function.

```
pre_data['INATTENTIONIND'].unique()
```

```
array([nan, 'Y'], dtype=object)
```

```
pre_data['INATTENTIONIND'].replace([np.nan, 'Y'], [0,1], inplace = True)
```



UNDERINFL

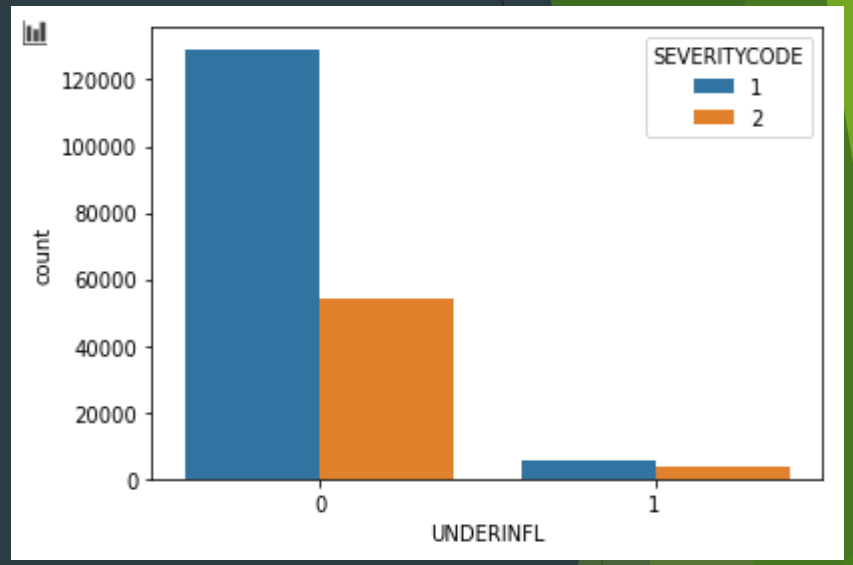
As with INATTENTIONIND, this column also needs to be tidied up since it contains multiple type of values (‘N’, ‘0’, NaN, ‘1’, ‘Y’)

```
pre_data['UNDERINFL'].unique()
```

```
array(['N', '0', nan, '1', 'Y'], dtype=object)
```

We’ll also use replace function to convert it to binary values.

```
pre_data['UNDERINFL'].replace(['N', '0', np.nan, '1', 'Y'], [0, 0, 0, 1, 1], inplace = True)
```



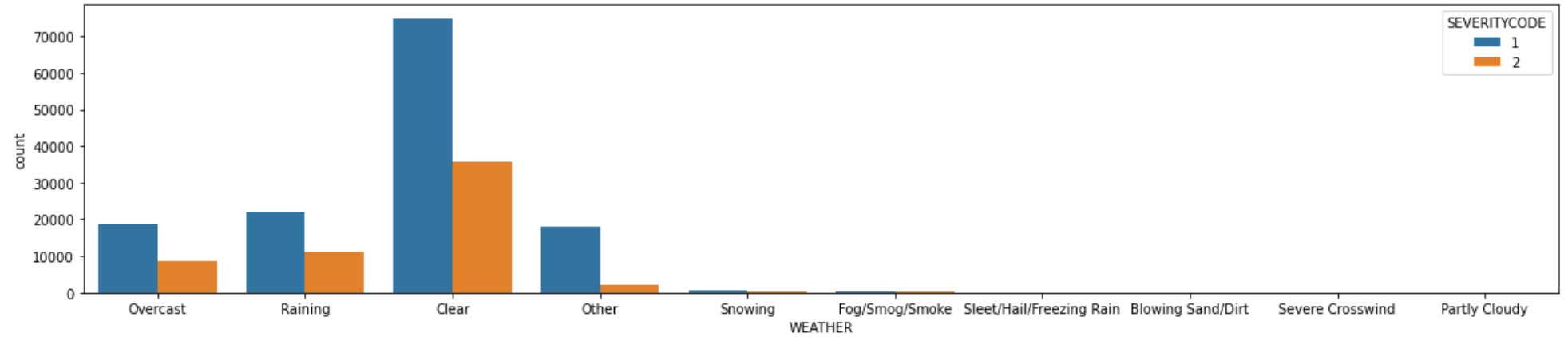
WEATHER

Here we'll group together NaN, 'Unknown', and 'Other' as Other. Interestingly most accidents happened on clear days.

```
pre_data['WEATHER'].replace([np.nan, 'Unknown'], ['Other', 'Other'], inplace = True)
```

```
plt.figure(figsize = (20,4))
sns.countplot(pre_data['WEATHER'], hue = pre_data['SEVERITYCODE'])
```

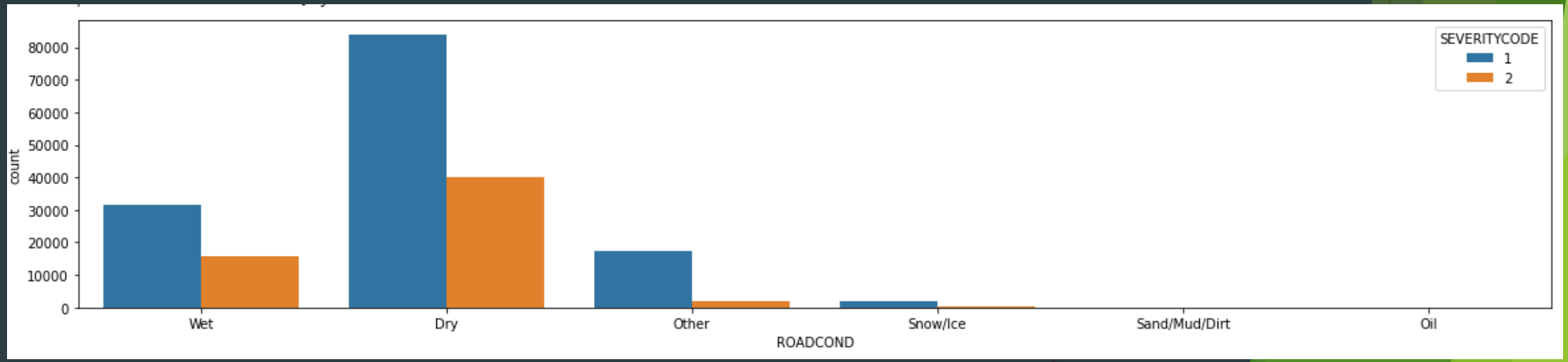
```
<AxesSubplot:xlabel='WEATHER', ylabel='count'>
```



ROADCOND

There are some values that can be grouped together:

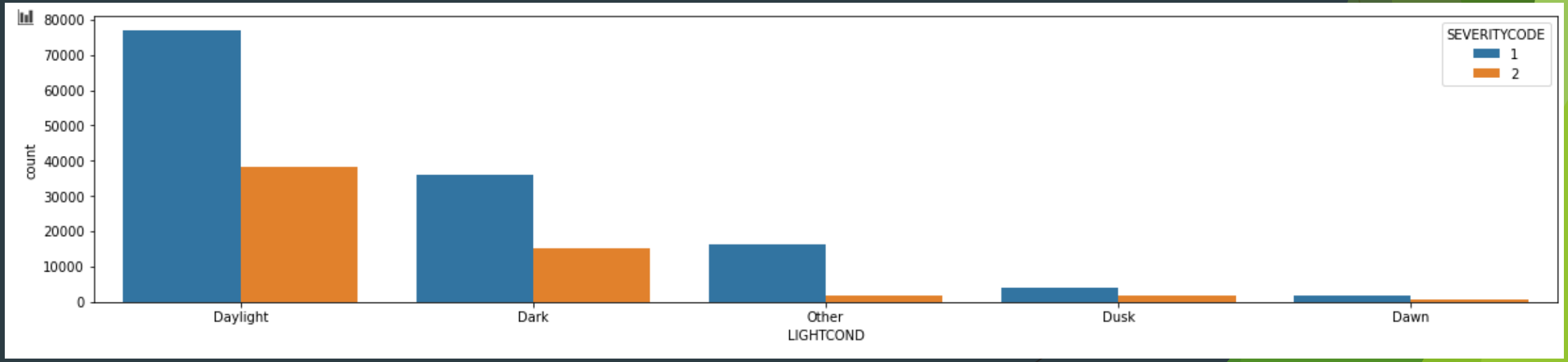
- Wet (Wet, Standing Water)
- Dry
- Other (nan, Unknown, Other)
- Snow/Ice (Snow/Slush, Ice)
- Sand/Mud/Dirt
- Oil



LIGHTCOND

Again, we grouped some similar values together:

- Daylight
- Dark (Dark - Street Lights On, Dark - No Street Lights, Dark - Street Lights Off, Dark - Unknown Lighting)
- Dusk
- Dawn
- Other (nan, Other, Unknown)



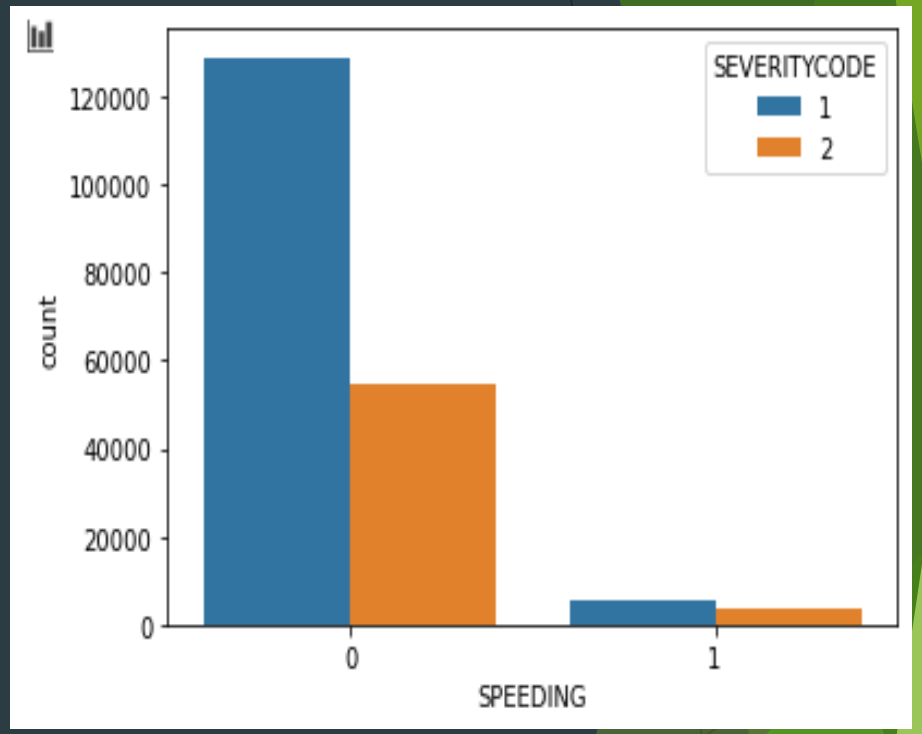
SPEEDING

We convert the values into binary data using replace function.

```
pre_data['SPEEDING'].value_counts(dropna = False)
```

```
NaN    183468
Y        9279
Name: SPEEDING, dtype: int64
```

```
pre_data['SPEEDING'].replace([np.nan, 'Y'], [0, 1], inplace = True)
```

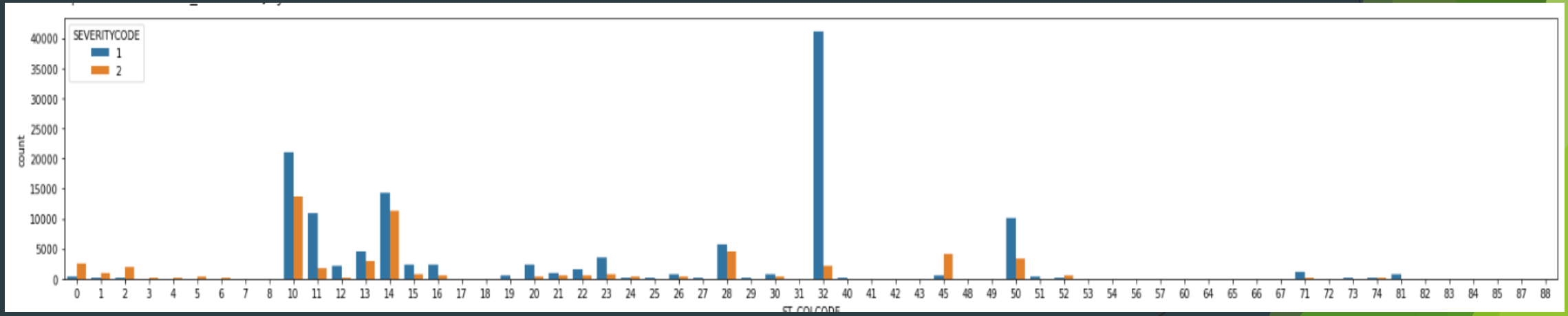


ST_COLCODE

There are 18 rows with NaN and blank space (‘ ‘) in ST_COLCODE column that we removed.

```
pre_data.dropna(subset = ['ST_COLCODE'], inplace = True)
```

```
pre_data.drop(pre_data.index[pre_data['ST_COLCODE'] == ' '], inplace = True)
```



SEGLANEKEY, CROSSWALKKEY

Almost every rows have 0 for both SEGLANEKEY and CROSSWALKKEY. As they don’t identify anything, we dropped these two columns.

```
pre_data[pre_data['SEGLANEKEY'] == 0]['SEGLANEKEY'].count()
```

```
185220
```

```
pre_data[pre_data['CROSSWALKKEY'] == 0]['CROSSWALKKEY'].count()
```

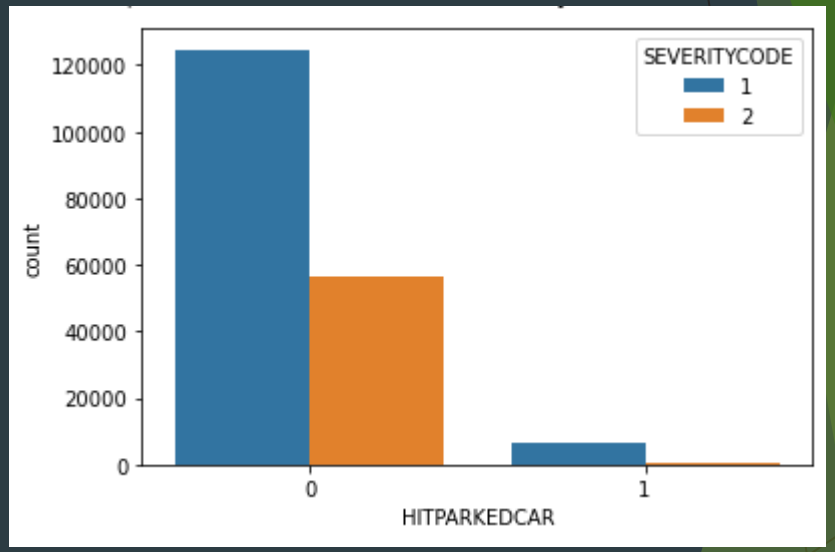
```
184187
```

HITPARKEDCAR

We'll convert HITPARKEDCAR into binary data by replacing the values with 0 and 1.

```
pre_data['HITPARKEDCAR'].replace(['N', 'Y'], [0, 1], inplace = True)
pre_data['HITPARKEDCAR'].unique()
```

```
array([0, 1], dtype=int64)
```



REVIEWING THE CLEANED UP DATA

```
pre_data.head()
```

	SEVERITYCODE	ADDRTYPE	PERSONCOUNT	SDOT_COLCODE	INATTENTIONIND	UNDERINFL	WEATHER	ROADCOND	LIGHTCOND	SPEEDING	ST_COLCODE	HITPARKEDCAR
0	2	Intersection	2	11	0	0	Overcast	Wet	Daylight	0	10	0
1	1	Block	2	16	0	0	Raining	Wet	Dark	0	11	0
2	1	Block	4	14	0	0	Overcast	Dry	Daylight	0	32	0
3	1	Block	3	11	0	0	Clear	Dry	Daylight	0	23	0
4	2	Intersection	2	11	0	0	Raining	Wet	Daylight	0	10	0

ONE-HOT ENCODING

Before we can create the train and test dataset, we need to convert the following categorical features into numerical values using one-hot encoding techniques.

- ADDRTYPE
- WEATHER
- ROADCOND
- LIGHTCOND

GET_DUMMIES

```
addrtype_dummy = pd.get_dummies(pre_data['ADDRTYPE']).drop('Alley', axis = 1)
weather_dummy = pd.get_dummies(pre_data['WEATHER']).drop('Other', axis = 1)
roadcond_dummy = pd.get_dummies(pre_data['ROADCOND']).drop('Other', axis = 1)
lightcond_dummy = pd.get_dummies(pre_data['LIGHTCOND']).drop('Other', axis = 1)
```

```
pre_data = pd.concat([pre_data, addrtype_dummy, weather_dummy, roadcond_dummy, lightcond_dummy], axis = 1)
```

```
pre_data.drop(['ADDRTYPE', 'WEATHER', 'ROADCOND', 'LIGHTCOND'], axis = 1, inplace = True)
```

	SEVERITYCODE	PERSONCOUNT	SDOT_COLCODE	INATTENTIONIND	UNDERINFL	SPEEDING	ST_COLCODE	HITPARKEDCAR	Block	Intersection	...	Snowing	Dry	Oil	Sand/Mud/Dirt	Snow/Ice	Wet	Dark	Dawn	Daylight	Dusk
0	2	2	11	0	0	0	10	0	0	1	...	0	0	0	0	0	1	0	0	1	0
1	1	2	16	0	0	0	11	0	1	0	...	0	0	0	0	0	1	1	0	0	0
2	1	4	14	0	0	0	32	0	1	0	...	0	1	0	0	0	0	0	0	1	0
3	1	3	11	0	0	0	23	0	1	0	...	0	1	0	0	0	0	0	0	1	0
4	2	2	11	0	0	0	10	0	0	1	...	0	0	0	0	0	1	0	0	1	0

TEST, TRAIN SPLIT

Now we'll split the data into training and test dataset using test_train_split function.

```
X = pre_data.loc[:,'PERSONCOUNT':]
y = pre_data['SEVERITYCODE']
```

```
from sklearn.model_selection import train_test_split
from sklearn import metrics

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
print('X_train.shape() = ', X_train.shape, ', y_train.shape() = ', y_train.shape)
print('X_test.shape()  = ', X_test.shape, ', y_test.shape()  = ', y_test.shape)
```

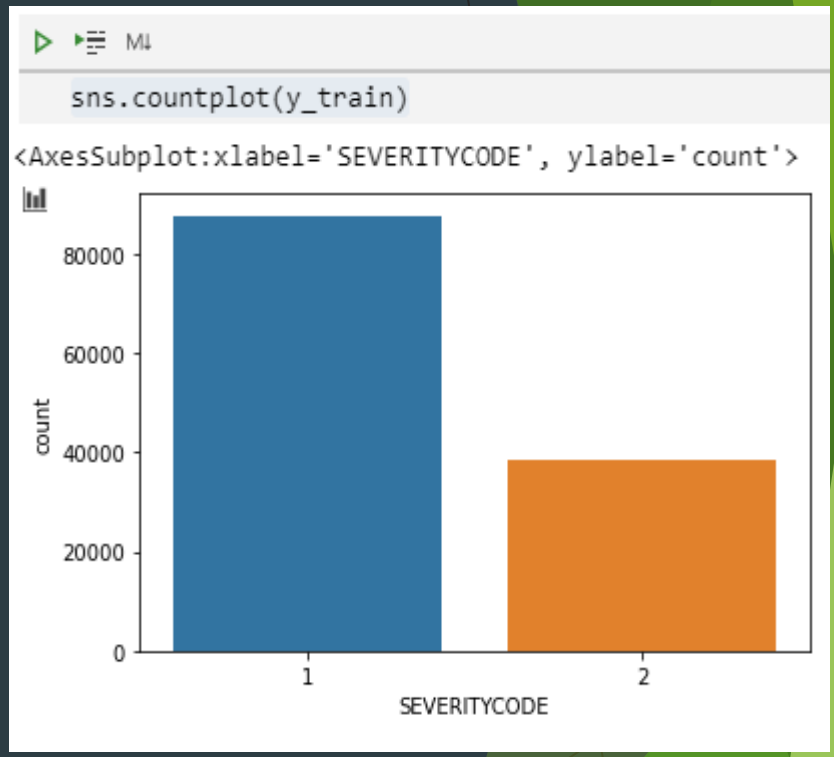
```
X_train.shape() =  (125926, 27) , y_train.shape() =  (125926,)
X_test.shape()  =  (62024, 27) , y_test.shape()  =  (62024,)
```

UNBALANCED DATA

Remember how our target values are very skewed towards SEVERITYCODE = 1? Now we need to address this problem before training our data.

There are several way to address this, but it this project we'll upsample the minority part of the train data so that the number of data for SEVERITYCODE = 1 & 2 are the same. This way we don't lose any data.

We'll accomplish this using resample function from sklearn package.



UPSAMPLING TRAINING DATA

First we will need to recombine X_train and y_train using pd.concat.

```
X_train = pd.concat([X_train, y_train], axis = 1)
```

Then we upsample the data for SEVERITYCODE = 2 to the number of data for SEVERITYCODE = 1.

```
print('SEVERITYCODE 1 = ',X_train[X_train['SEVERITYCODE'] == 1]['SEVERITYCODE'].count())
print('SEVERITYCODE 2 = ',X_train[X_train['SEVERITYCODE'] == 2]['SEVERITYCODE'].count())
```

```
SEVERITYCODE 1 =  87675
SEVERITYCODE 2 =  38251
```

```
{}
```

```
Then we will upsample the data for SEVERITYCODE 2 using resample function from sklearn.
```

```
from sklearn.utils import resample

X_1 = X_train[X_train['SEVERITYCODE'] == 1]
X_2 = X_train[X_train['SEVERITYCODE'] == 2]

X_2_upsample = resample(X_2, replace=True, n_samples=len(X_1), random_state=42)
len(X_2_upsample)
```

```
87675
```

RECREATE TRAINING DATASET

Now we can split the training dataset into X_train and y_train again.

```
X_train_upsample = pd.concat([X_1, X_2_upsample], axis = 0)
len(X_train_upsample)
```

```
175350
```

```
y_train_upsample = X_train_upsample['SEVERITYCODE']
X_train_upsample.drop('SEVERITYCODE', axis = 1, inplace = True)
```

This will be the dataset for training our model.

BUILDING THE MODELS

Due to computanional limitation, we'll only use the following models for our project:

- Logistic Regression
- Decision Tree
- Support Vector Machine

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression

mod_log_r = LogisticRegression()
mod_log_r.fit(X_train_upsample, y_train_upsample)
yhat_log_r = mod_log_r.predict(X_test)
yhat_log_r_proba = mod_log_r.predict_proba(X_test)
print("Logistic Regression's Accuracy: ", metrics.accuracy_score(y_test, yhat_log_r))
```

```
Logistic Regression's Accuracy:  0.6494421514252547
```

DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier

mod_tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4).fit(X_train_upsample, y_train_upsample)
yhat_tree = mod_tree.predict(X_test)
print("Decision Trees's Accuracy: ", metrics.accuracy_score(y_test, yhat_tree))
```

```
Decision Trees's Accuracy:  0.7171739971623887
```

SUPPORT VECTOR MACHINE

```
from sklearn import svm

mod_svm = svm.SVC(kernel='rbf', gamma = 'scale').fit(X_train_upsample, y_train_upsample)
yhat_svm = mod_svm.predict(X_test)
print("Decision Trees's Accuracy: ", metrics.accuracy_score(y_test, yhat_svm))
```

```
Decision Trees's Accuracy:  0.6362859538243261
```

4. RESULTS

COMPARING MODEL'S ACCURACY

We'll compare the model's performance against the test dataset using Jaccard Score and F1-Score as the metrics.

```
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score

report = pd.DataFrame(index = ['LogisticRegression', 'Decision Tree', 'SVM'], columns = ['Jaccard', 'F1-score'])

report.loc['LogisticRegression', 'Jaccard'] = jaccard_score(y_test, yhat_log_r)
report.loc['LogisticRegression', 'F1-score'] = f1_score(y_test, yhat_log_r, average = 'weighted')

report.loc['Decision Tree', 'Jaccard'] = jaccard_score(y_test, yhat_tree)
report.loc['Decision Tree', 'F1-score'] = f1_score(y_test, yhat_tree, average = 'weighted')

report.loc['SVM', 'Jaccard'] = jaccard_score(y_test, yhat_svm)
report.loc['SVM', 'F1-score'] = f1_score(y_test, yhat_svm, average = 'weighted')


report.index.name = 'Algorithm'
report
```

RESULT COMPARISION

Here’s the result. Decision Tree gives us the best performance, which is unsurprising since it handles unbalanced dataset better compared to Logistic Regression and Support Vector Machine.

Using confusion matrixes, we can also see how the models perform in dealing with the data.

Algorithm	Jaccard	F1-score
LogisticRegression	0.56196	0.663106
Decision Tree	0.659848	0.719192
SVM	0.522793	0.650411

```
Confusion Matrix for Logistic Regression:
[[27894 15404]
 [ 6339 12387]]

Confusion Matrix for Decision Tree:
[[34029  9269]
 [ 8273 10453]]

Confusion Matrix for Support Vector Machine:
[[24714 18584]
 [ 3975 14751]]
```

5. DISCUSSION

While the models cannot fully predict the severity of an accident by using the data available from the accident report, they are able to give us an adequate result, especially using Decision Tree model. This prediction can help the Dispatch Centre to better allocate their personnel in a moment notice using the information from the initial report, of course, provided that the information received is accurate and complete enough.

6. CONCLUSSION

The data we use have an unbalanced number of SEVERITYCODE values and is heavily skewed toward SEVERITYCODE = 1. Also, some of the lines are missing some information. Due to those, we needed to remove rows with missing information and resampled the training data to reinforce the signal of the data in the minor category (SEVERITYCODE = 2).

With those limitations, we managed to build three classification models, Logistic Regression, Decision Tree, and Support Vector Machine, even though there are still room to improve the model's performances.

Comparing the scores for those models, we have the Decision Tree model that gives us the best accuracy score. With better dataset, we sure can improve the performance of the model.