PREDICTING CAR ACCIDENT'SEVERITY

IBM APPLIED DATA SCIENCE CAPSTONE PROJECT

# 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 Centre can decide which personnel should be dispatched for the accident. For example, for accident with severity of 1 Property Damage Only Collision, the healthcare personnel are not needed on site, and they can be allocated to another injury related accident.

# 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 / vehicles 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 itself containing 1 target column & 37 feature columns, some of them are not necessarily useful for us in building the model, with a total number of 194673 rows.

The target column is SEVERITYCODE which contains the severity classification. We have 2 different severity values here:

*1 Property Damage Only Collision*

*2 Injury Collision*

These are the 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'*).

We are going to use the following feature columns in our initial model and adding or remove the features as necessary as we build the model.

*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. See the State Collision Code Dictionary in the Metadata file.*

*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)*

# 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"*

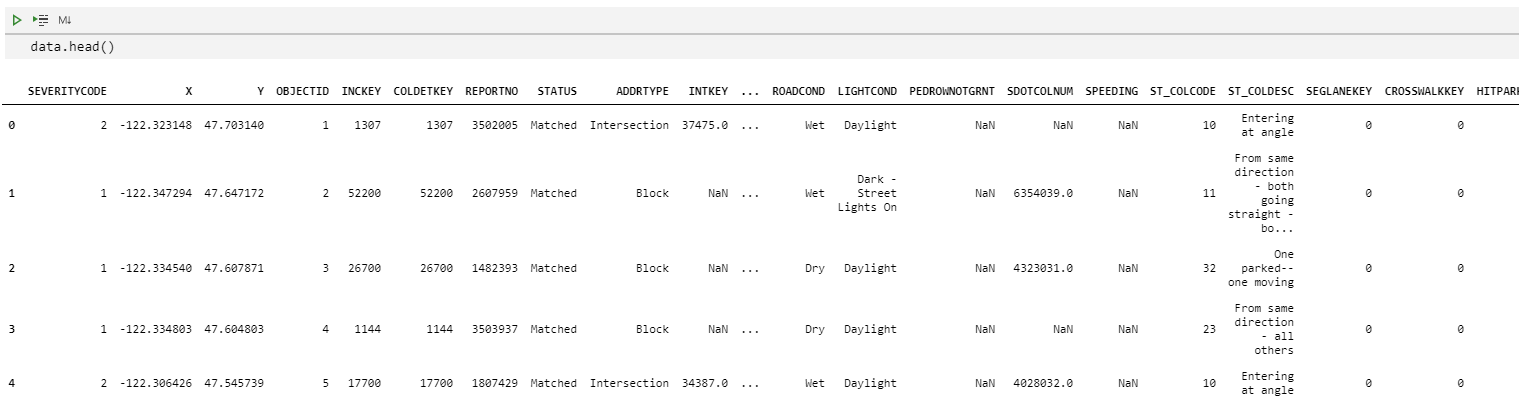
*import pandas as pd*

*import seaborn as sns*

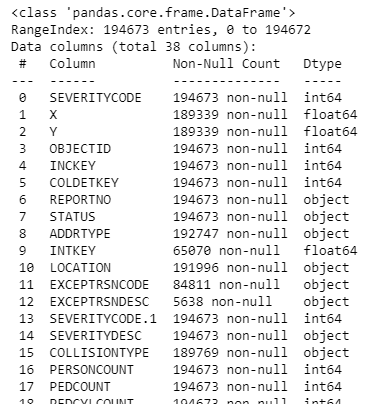
*import numpy as np*

*import matplotlib.pyplot as plt*

*%matplotlib inline*

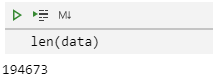
**

*data.info()*

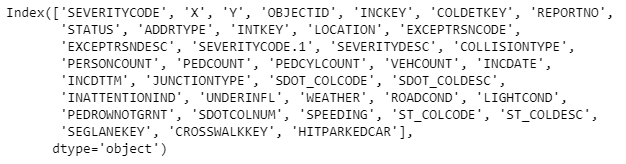


We have a total of 194673 car accident records, some of columns seem to be missing some information.

*len(data)*



*data.columns*



## PRELIMINARY TARGET CHECK

Using *countplot*, we can see that the data is very unbalanced, and highly skewed towards *SEVERITYCODE* = 1. Training our model with this kind of data is not recommended due to the bias.

*sns.countplot(pre\_data['SEVERITYCODE'])*



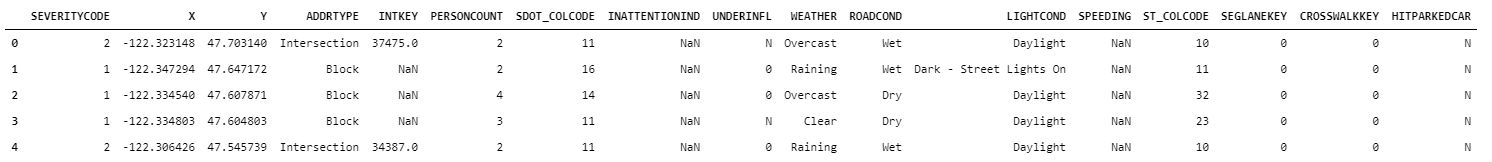
We will address this issue when we start to train our model later.

## PRELIMINARY FEATURES CHECK

Next, we examine the features that we decided to use one by one to see their significance on the *SEVERITYCODE* value. We will start by creating a copy of our original data frame containing only the columns that we decided to use at the beginning of the project.

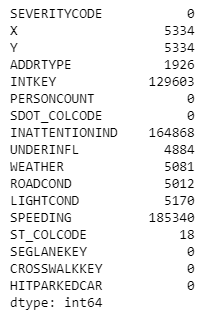
*pre\_data = data[['SEVERITYCODE', 'X', 'Y', 'ADDRTYPE', 'INTKEY', 'PERSONCOUNT', 'SDOT\_COLCODE', 'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND', 'SPEEDING', 'ST\_COLCODE', 'SEGLANEKEY', 'CROSSWALKKEY', 'HITPARKEDCAR']].copy()*

*pre\_data.head()*



We can also try to check for missing values in each column.

*pre\_data.isna().sum()*

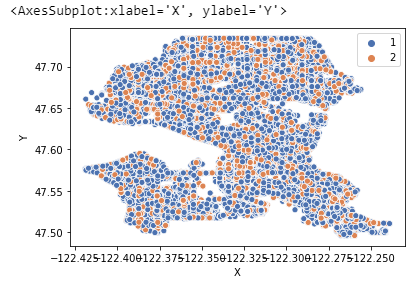


Next, we will go through each column one be by one to see if there are any necessary actions needed to clean up the data.

### X, Y

Note that there are 5334 lines without coordinates data. By plotting X and Y, we can see that there is no clear separation between areas with *SEVERITYCODE* = 1 and 2.

*sns.scatterplot(x = pre\_data['X'], y = pre\_data['Y'], hue = pre\_data['SEVERITYCODE'].tolist(), palette = 'deep')*

**

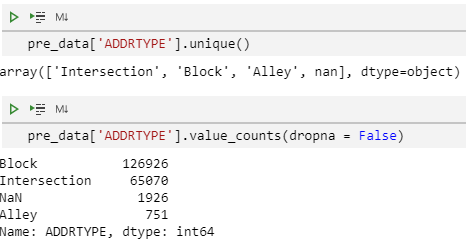
We can drop these columns as they pose little significance for predicting *SEVERITYCODE* values.

*pre\_data.drop(['X', 'Y'], axis = 1, inplace = True)*

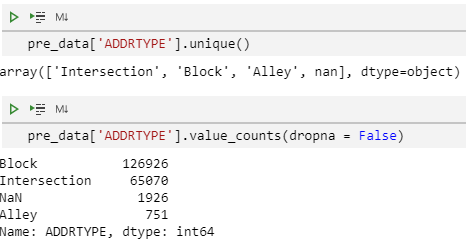
### ADDRTYPE

We can check the unique data and their counts in this column.

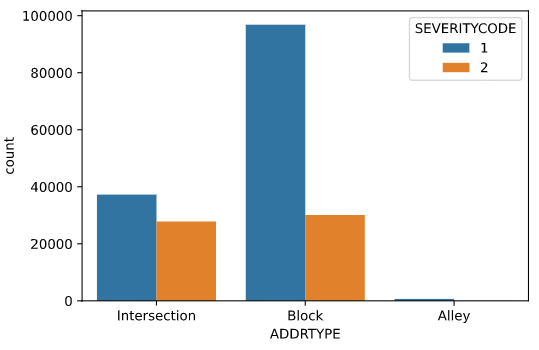
*pre\_data['ADDRTYPE'].unique()*

**

*pre\_data['ADDRTYPE'].value\_counts(dropna = False)*

**

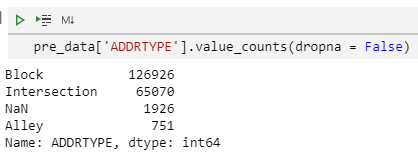
*sns.countplot(x = 'ADDRTYPE', data = pre\_data, hue = 'SEVERITYCODE')*

**

From the graph above we can see that we have more *SEVERITYCODE* 1 when the accident is happened in the blocks.

As for the 1926 rows with missing *ADDRTYPE*, we will drop from the data frame.

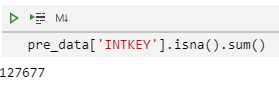
*pre\_data.dropna(subset = ['ADDRTYPE'], inplace = True)*



### INTKEY

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

*pre\_data['INTKEY'].isna().sum()*

**

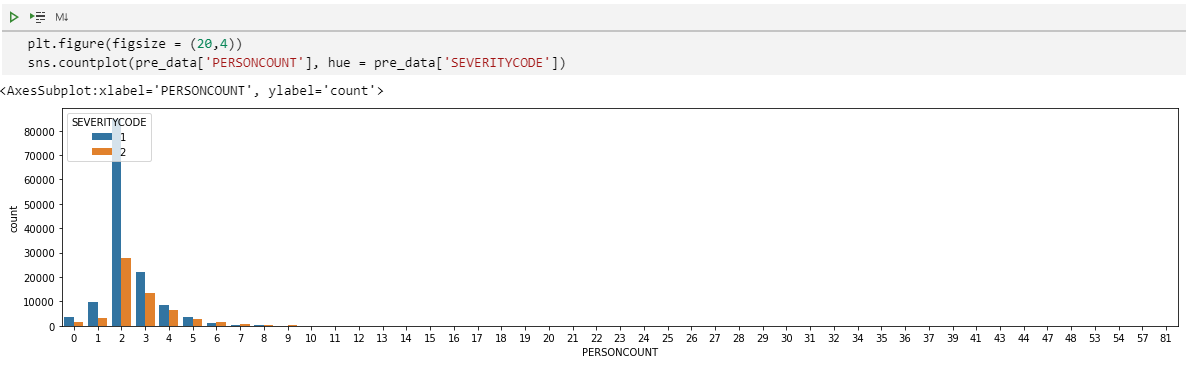
*pre\_data.drop('INTKEY', axis = 1, inplace = True)*

### PERSONCOUNT

Let us try to visualize this column.

*plt.figure(figsize = (20,4))*

*sns.countplot(pre\_data['PERSONCOUNT'], hue = pre\_data['SEVERITYCODE'])*



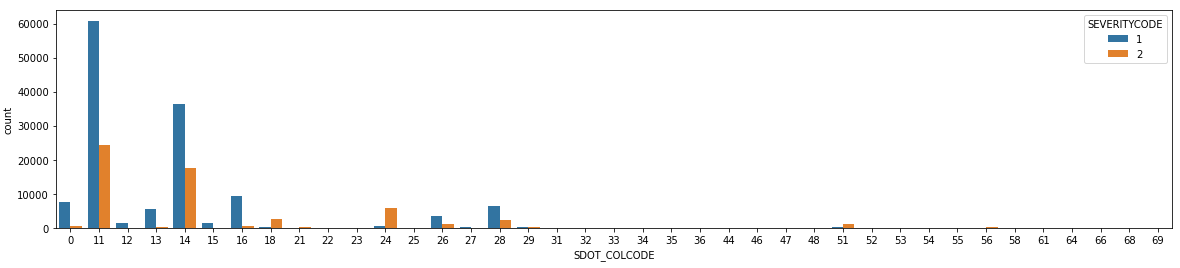
We can see that most car accidents involve two people. Nothing to be done on this column as everything is in place and no missing value.

### SDOT\_COLCODE

Let us plot this column too.

*plt.figure(figsize = (20,4))*

*sns.countplot(pre\_data['SDOT\_COLCODE'], hue = pre\_data['SEVERITYCODE'])*



We can see that most accidents happen with SDOT\_COLCODE = 11 and 14.

*SDOT\_COLCODE 11 - motor vehicle struck another motor vehicle in front end*

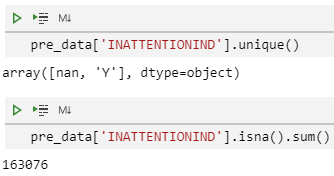
*SDOT\_COLCODE 14 - motor vehicle struck another motor vehicle in rear end*

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

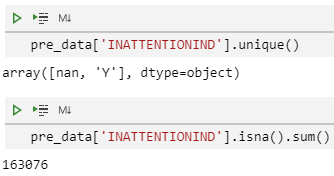
### INATTENTIONIND

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

*pre\_data['INATTENTIONIND'].unique()*



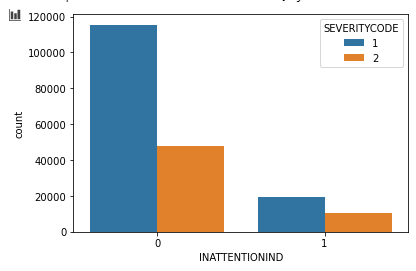
*pre\_data['INATTENTIONIND'].isna().sum()*



*pre\_data['INATTENTIONIND'].replace([np.nan, 'Y'], [0,1], inplace = True)*

Plotting the data, we can see that more *SEVERITYCODE* 1 mostly happens when *INATTENTIONIND* = 0.

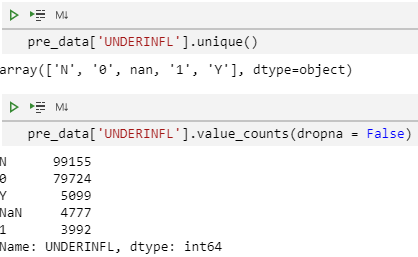
*sns.countplot(pre\_data['INATTENTIONIND'], hue = pre\_data['SEVERITYCODE'])*

**

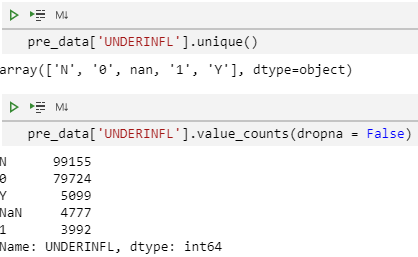
### UNDERINFL

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()*

**

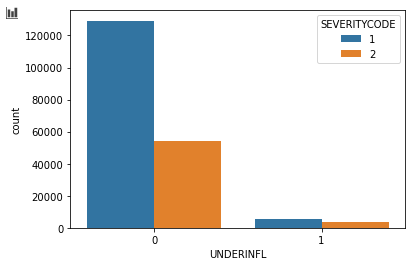
*pre\_data['UNDERINFL'].value\_counts(dropna = False)*

**

*pre\_data['UNDERINFL'].replace(['N', '0', np.nan, '1', 'Y'], [0, 0, 0, 1, 1], inplace = True)*

Let us try plotting this column too.

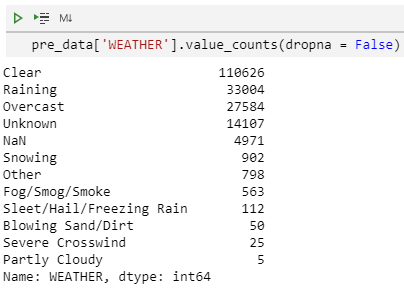
*sns.countplot(pre\_data['UNDERINFL'], hue = pre\_data['SEVERITYCODE'])*



### WEATHER

Here we will group together NaN, ‘Unknown’, and ‘Other’ as Other to simplify the categories.

*pre\_data['WEATHER'].value\_counts(dropna = False)*

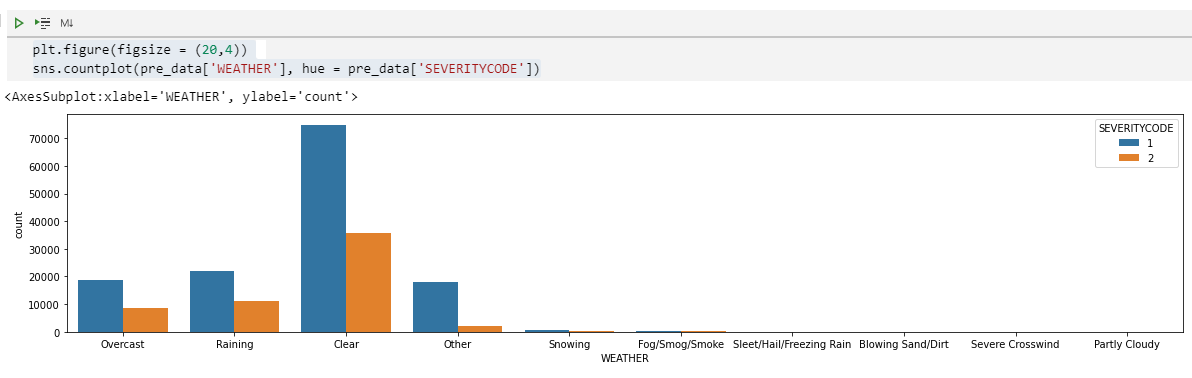


*pre\_data['WEATHER'].replace([np.nan, 'Unknown'], ['Other', 'Other'], inplace = True)*

Interestingly, most accidents happened on clear days.

*plt.figure(figsize = (20,4))*

*sns.countplot(pre\_data['WEATHER'], hue = pre\_data['SEVERITYCODE'])*

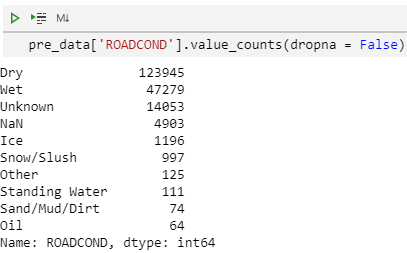
**

### ROADCOND

Looking at the unique values, 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

*pre\_data['ROADCOND'].value\_counts(dropna = False)*

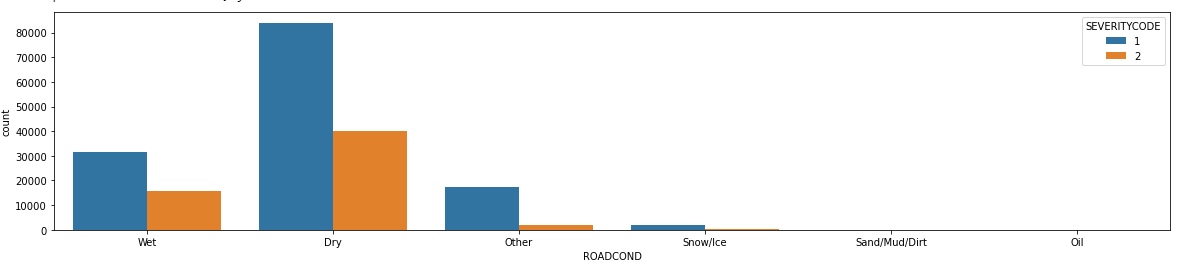


*pre\_data['ROADCOND'].replace(['Standing Water', np.nan, 'Unknown', 'Snow/Slush', 'Ice'], ['Wet', 'Other', 'Other', 'Snow/Ice', 'Snow/Ice'], inplace = True)*

Another interesting thing, most accidents happened when the road condition is dry.

*plt.figure(figsize = (20,4))*

*sns.countplot(pre\_data['ROADCOND'], hue = pre\_data['SEVERITYCODE'])*

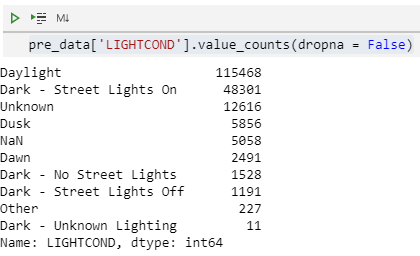
**

### LIGHTCOND

Again, we can group 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)

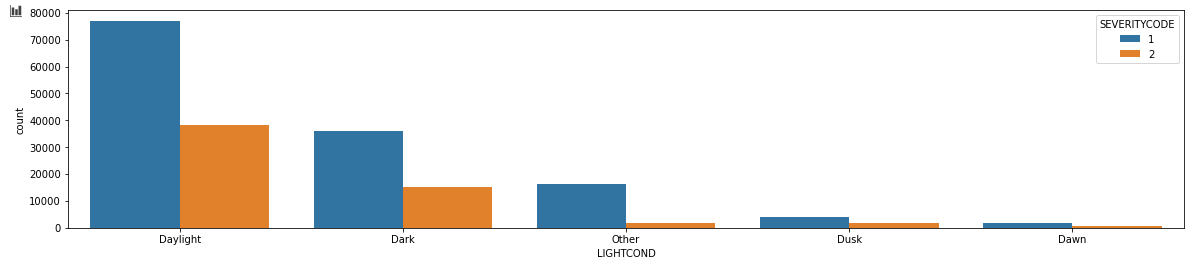
*pre\_data['LIGHTCOND'].value\_counts(dropna = False)*

**

Then we plot the data again. Somehow most of the accidents happened during daylight.

*plt.figure(figsize = (20,4))*

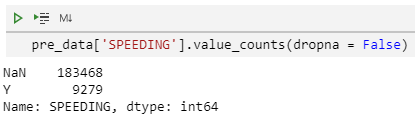
*sns.countplot(pre\_data['LIGHTCOND'], hue = pre\_data['SEVERITYCODE'])*



### SPEEDING

We convert the values into binary data using replace function.

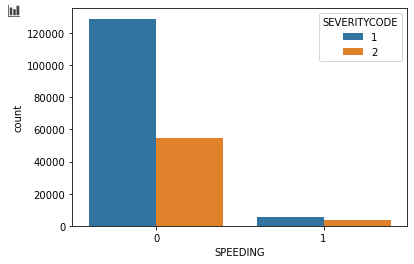
*pre\_data['SPEEDING'].value\_counts(dropna = False)*

**

*pre\_data['SPEEDING'].replace([np.nan, 'Y'], [0, 1], inplace = True)*

Again, we plot the data.

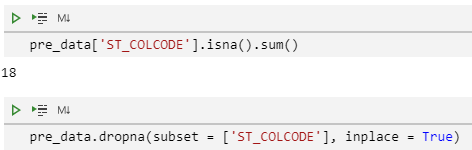
*sns.countplot(pre\_data['SPEEDING'], hue = pre\_data['SEVERITYCODE'])*

**

### ST\_COLCODE

We can see that there are 18 missing data for *ST\_COLCODE*. Since this is an insignificant number compared to the total data, we will remove the lines with missing *ST\_COLCODE* info.

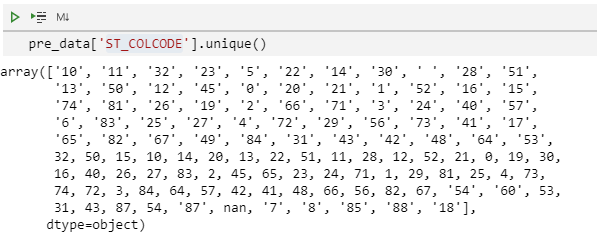
*pre\_data['ST\_COLCODE'].isna().sum()*



*pre\_data.dropna(subset = ['ST\_COLCODE'], inplace = True)*

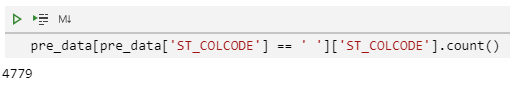
Next, we need to deal with how this column’s values are a combination of text, empty space ‘ ‘, and number.

*pre\_data['ST\_COLCODE'].unique()*



First we’ll deal with the empty space ‘ ‘. There are 4779 of them, which is not that big of a number compared to the total rows. We will just remove them from our data frame. Note that we could not fill them with estimated values as they are categorical data.

*pre\_data[pre\_data['ST\_COLCODE'] == ' ']['ST\_COLCODE'].count()*



*pre\_data.drop(pre\_data.index[pre\_data['ST\_COLCODE'] == ' '], inplace = True)*

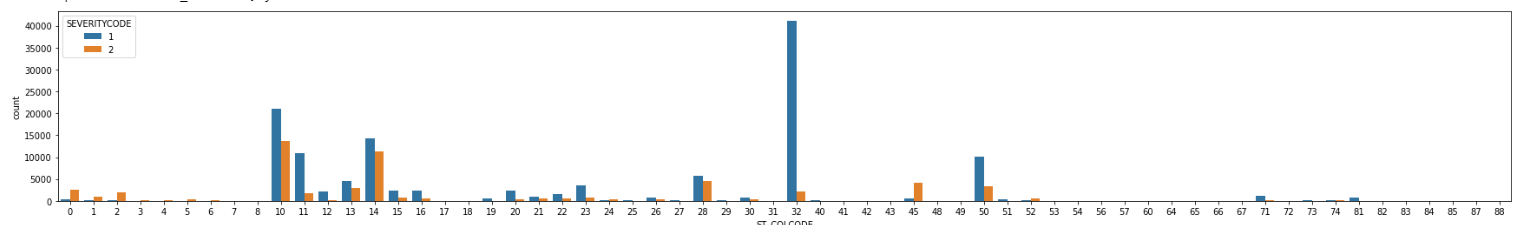
Next, we will convert everything else to int to make it easier when building the model.

*pre\_data['ST\_COLCODE'] = pre\_data['ST\_COLCODE'].astype('int64')*

We can try to plot the data now.

*plt.figure(figsize = (30,4))*

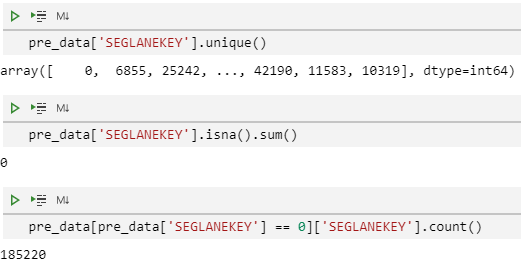
*sns.countplot(pre\_data['ST\_COLCODE'], hue = pre\_data['SEVERITYCODE'])*

**

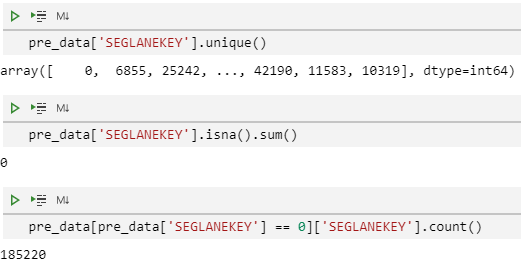
### SEGLANEKEY, CROSSWALKKEY

Almost every row has 0 for both SEGLANEKEY and CROSSWALKKEY. As they do not identify anything, we dropped these two columns.

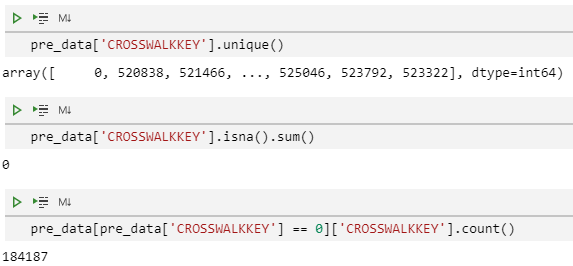
*pre\_data['SEGLANEKEY'].unique()*

**

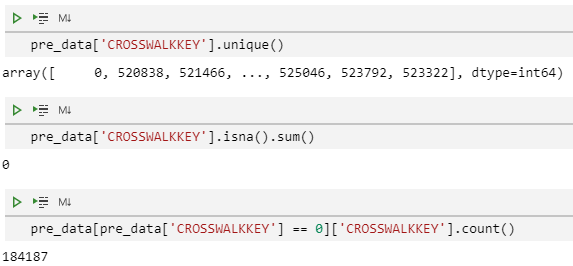
*pre\_data[pre\_data['SEGLANEKEY'] == 0]['SEGLANEKEY'].count()*

**

*pre\_data['CROSSWALKKEY'].unique()*

**

*pre\_data[pre\_data['CROSSWALKKEY'] == 0]['CROSSWALKKEY'].count()*

**

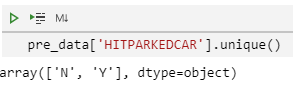
*pre\_data.drop('SEGLANEKEY', axis = 1, inplace = True)*

*pre\_data.drop('CROSSWALKKEY', axis = 1, inplace = True)*

### HITPARKEDCAR

We will convert *HITPARKEDCAR* into binary data by replacing the values with 0 and 1.

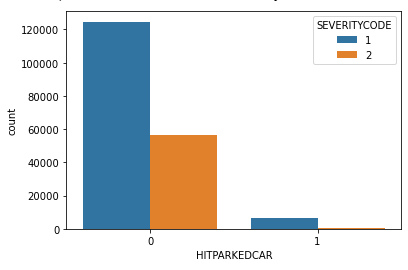
pre\_data['HITPARKEDCAR'].unique()



*pre\_data['HITPARKEDCAR'].replace(['N', 'Y'], [0, 1], inplace = True)*

And we can plot it.

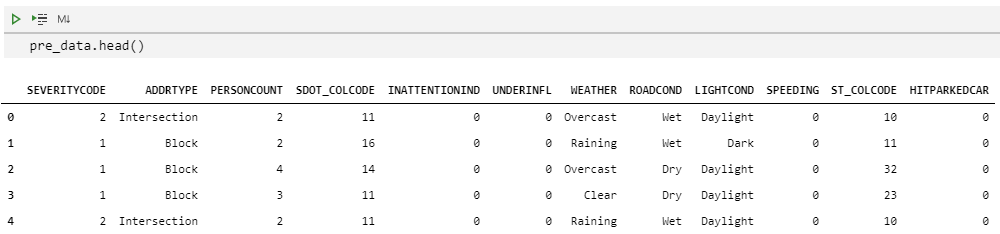
*sns.countplot(pre\_data['HITPARKEDCAR'], hue = pre\_data['SEVERITYCODE'])*



## REVIEWING THE CLEANED-UP DATA

Now, let us review our cleaned-up data. We now only have 11 feature columns.

*pre\_data.head()*



## 3.5 ONE-HOT ENCODING

Before we can pass this data to train our model, we need to convert the following categorical features into numerical values.

* *ADDRTYPE*
* *WEATHER*
* *ROADCOND*
* *LIGHTCOND*

We can do this using one-hot encoding technique.

*addrtype\_dummy = pd.get\_dummies(pre\_data['ADDRTYPE']).drop('Alley', axis = 1)*

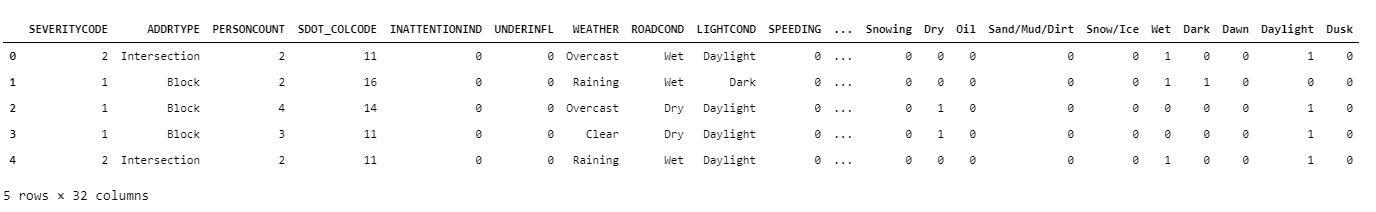
*weather\_dummy = pd.get\_dummies(pre\_data['WEATHER']).drop('Other', axis = 1)*

*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)*

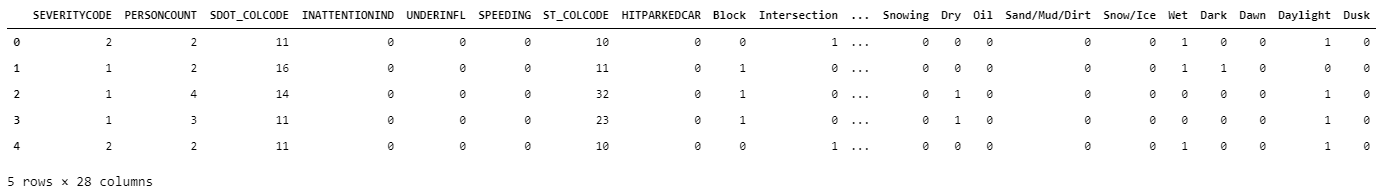
*pre\_data.head()*

**

We will drop *ADDRTYPE, WEATHER, ROADCOND*, and *LIGHTCOND* since we already have generated the dummy features from them.

*pre\_data.drop(['ADDRTYPE', 'WEATHER', 'ROADCOND', 'LIGHTCOND'], axis = 1, inplace = True)*

*pre\_data.head()*

**

## TEST, TRAIN SPLIT

Now we will 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)*

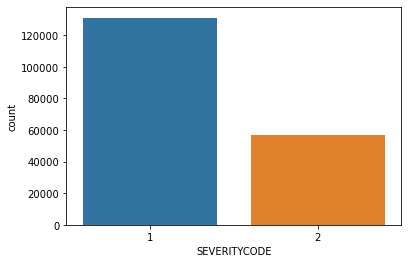
**

*print('X\_test.shape() = ', X\_test.shape, ', y\_test.shape() = ', y\_test.shape)*

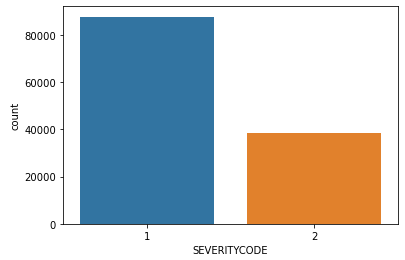
**

Remember that we have unbalanced dataset?

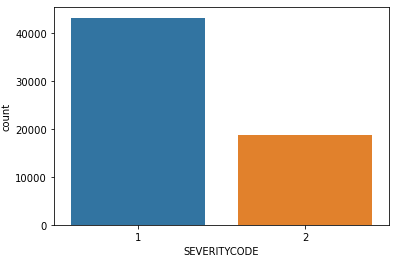
*sns.countplot(pre\_data['SEVERITYCODE'])*

**

*sns.countplot(y\_train)*

**

*sns.countplot(y\_test)*

**

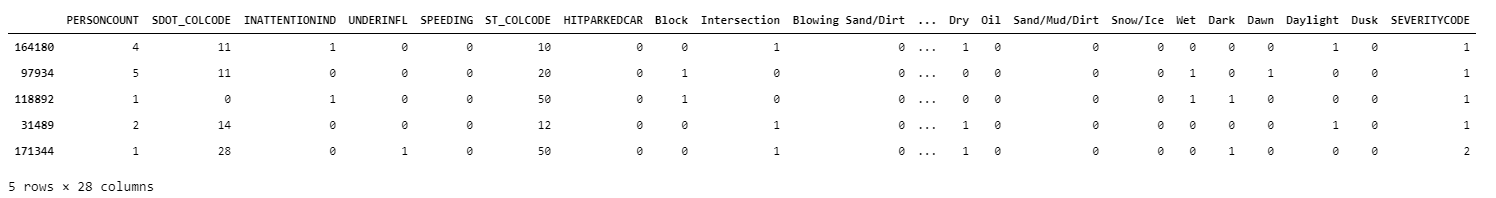
Now we need to address this issue with our training dataset.

There are several ways to do this, for this project we will up sample the training data with minority group, which is the one with *SEVERITYCODE* 2.

First, we need to recombine X\_train and y\_train.

*X\_train = pd.concat([X\_train, y\_train], axis = 1)*

*X\_train.head()*

**

*print('SEVERITYCODE 1 = ',X\_train[X\_train['SEVERITYCODE'] == 1]['SEVERITYCODE'].count())*

**

*print('SEVERITYCODE 2 = ',X\_train[X\_train['SEVERITYCODE'] == 2]['SEVERITYCODE'].count())*

**

Then we will up sample 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)*



Now we have the same number of training data for *SEVERITYCODE* 1 and 2.

We can recreate our training set.

*X\_train\_upsample = pd.concat([X\_1, X\_2\_upsample], axis = 0)*

*y\_train\_upsample = X\_train\_upsample['SEVERITYCODE']*

*X\_train\_upsample.drop('SEVERITYCODE', axis = 1, inplace = True)*

## MODEL BUILDING

Due to computational limitation, we will only use Logistic Regression, Decision Tree, and Support Vector Machine for the models.

### Logistic Regression

*from sklearn.linear\_model import LogisticRegression*

*mod\_log\_r = LogisticRegression()*

*mod\_log.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))*

**

### Decision Tree

*from sklearn.tree import DecisionTreeClassifier*

*mod\_tree = DecisionTreeClassifier(criterion="entropy", max\_depth = 4)*

*mod\_tree.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))*

**

### Support Vector Machine

*from sklearn import svm*

*mod\_svm = svm.SVC(kernel='rbf', gamma = 'scale')*

*mod\_svm.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))*

**

# RESULTS

To evaluate and comparing the results of our three models, we will use Jaccard Similarity Score and F1-Score as our 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)*

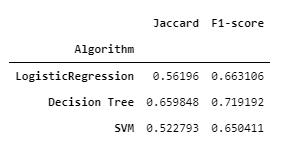
*report.loc['Decision Tree', 'F1-score'] = f1\_score(y\_test, yhat\_tree, average = 'weighted')*

*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*

**

Decision Tree gives us the best performance, which is unsurprising since its algorithm handles unbalanced dataset better compared to Logistic Regression and Support Vector Machine.

We can also try to print out the confusion matrixes for the models.

*print('Confusion Matrix for Logistic Regression:')*

*print(metrics.confusion\_matrix(y\_test, yhat\_log\_r))*

*print()*

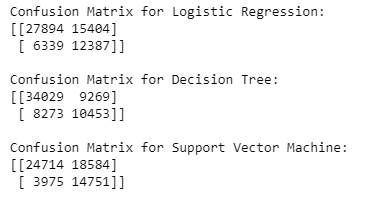
*print('Confusion Matrix for Decision Tree:')*

*print(metrics.confusion\_matrix(y\_test, yhat\_tree))*

*print()*

*print('Confusion Matrix for Support Vector Machine:')*

*print(metrics.confusion\_matrix(y\_test, yhat\_svm))*



Again, we can see that Decision Tree gives us the best True Positive numbers, although it is the worst performer in predicting True Negatives.

# 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.

# 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. Comparing the scores for those models, we have the Decision Tree model that gives us the best accuracy score.

There are still room to improve the model’s performances, though. With better dataset, we sure can do better. Another option is to try another method to address the imbalance in the dataset, for example, down sampling the majority group instead of up sampling the minority group.