PREDICTING CAR ACCIDENT'SEVERITY

IBM APPLIED DATA SCIENCE CAPSTONE PROJECT

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 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.

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 / 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)*

3. METHODOLOGY

3.1 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.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	

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 194673 entries, 0 to 194672
Data columns (total 38 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   SEVERITYCODE    194673 non-null  int64
 1   X               189339 non-null  float64
 2   Y               189339 non-null  float64
 3   OBJECTID        194673 non-null  int64
 4   INCKEY          194673 non-null  int64
 5   COLDETKEY       194673 non-null  int64
 6   REPORTNO        194673 non-null  object
 7   STATUS          194673 non-null  object
 8   ADDRTYPE        192747 non-null  object
 9   INTKEY          65070 non-null   float64
 10  LOCATION        191996 non-null  object
 11  EXCEPTRSNCODE   84811 non-null   object
 12  EXCEPTRSNDESC   5638 non-null    object
 13  SEVERITYCODE.1  194673 non-null  int64
 14  SEVERITYDESC    194673 non-null  object
 15  COLLISIONTYPE   189769 non-null  object
 16  PERSONCOUNT     194673 non-null  int64
 17  PEDCOUNT        194673 non-null  int64
 18  PEDCYLCOUNT     194673 non-null  int64
```

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

len(data)

```
194673
```

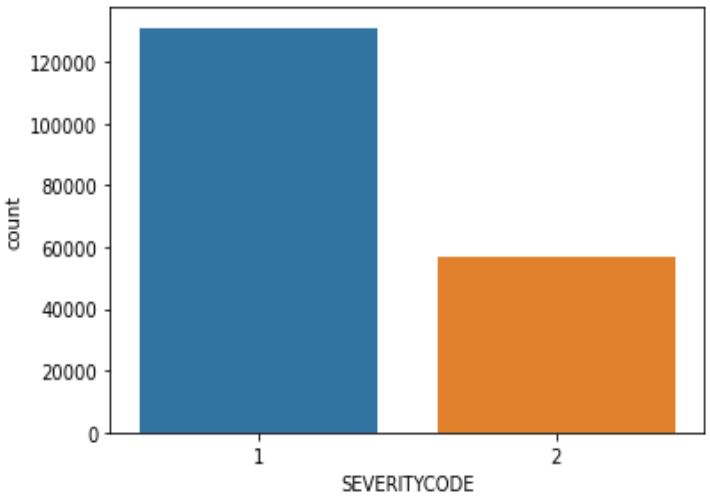
data.columns

```
Index(['SEVERITYCODE', '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'],
      dtype='object')
```

3.2 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.

3.3 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()

	SEVERITYCODE	X	Y	ADDRTYPE	INTKEY	PERSONCOUNT	SDOT_COLCODE	INATTENTIONIND	UNDERINFL	WEATHER	ROADCOND	LIGHTCOND	SPEEDING	ST_COLCODE	SEGLANEKEY	CROSSWALKKEY	HITPARKEDCAR
0	2	-122.323148	47.703140	Intersection	37475.0	2	11	NaN	N	Overcast	Wet	Daylight	NaN	10	0	0	N
1	1	-122.347294	47.647172	Block	NaN	2	16	NaN	0	Raining	Wet	Dark - Street Lights On	NaN	11	0	0	N
2	1	-122.334540	47.607871	Block	NaN	4	14	NaN	0	Overcast	Dry	Daylight	NaN	32	0	0	N
3	1	-122.334803	47.604803	Block	NaN	3	11	NaN	N	Clear	Dry	Daylight	NaN	23	0	0	N
4	2	-122.306426	47.545739	Intersection	34387.0	2	11	NaN	0	Raining	Wet	Daylight	NaN	10	0	0	N

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

pre_data.isna().sum()

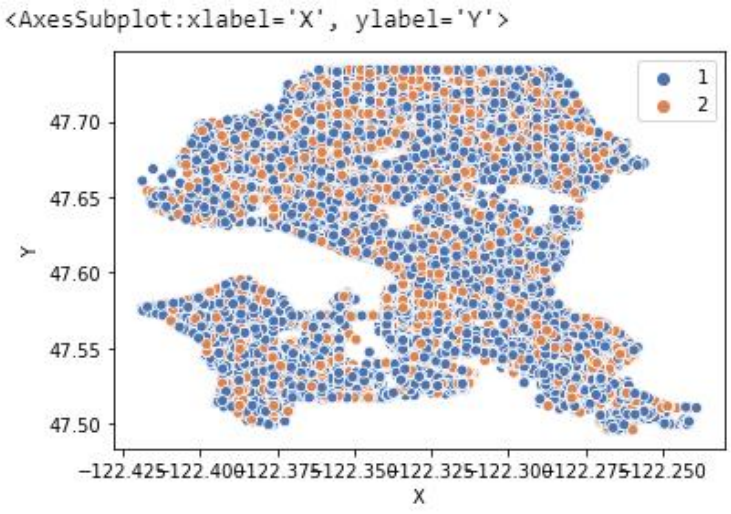
```
SEVERITYCODE           0
X                   5334
Y                   5334
ADDRTYPE            1926
INTKEY            129603
PERSONCOUNT            0
SDOT_COLCODE           0
INATTENTIONIND    164868
UNDERINFL           4884
WEATHER             5081
ROADCOND            5012
LIGHTCOND           5170
SPEEDING          185340
ST_COLCODE            18
SEGLANEKEY             0
CROSSWALKKEY           0
HITPARKEDCAR           0
dtype: int64
```

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.

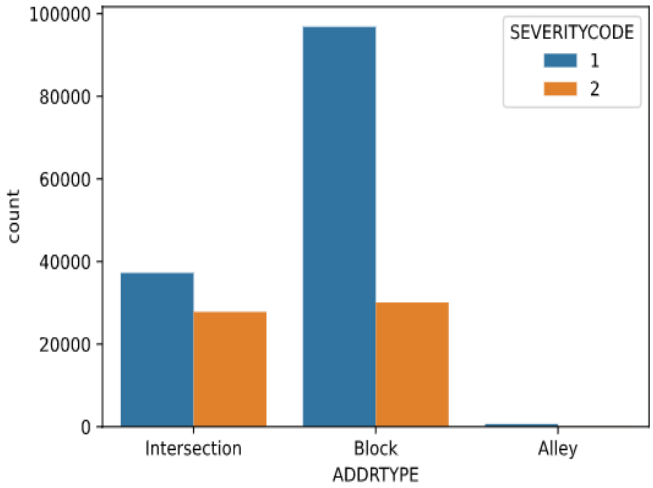
pre_data['ADDRTYPE'].unique()

```
array(['Intersection', 'Block', 'Alley', nan], dtype=object)
```

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

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

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)

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

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()

127677

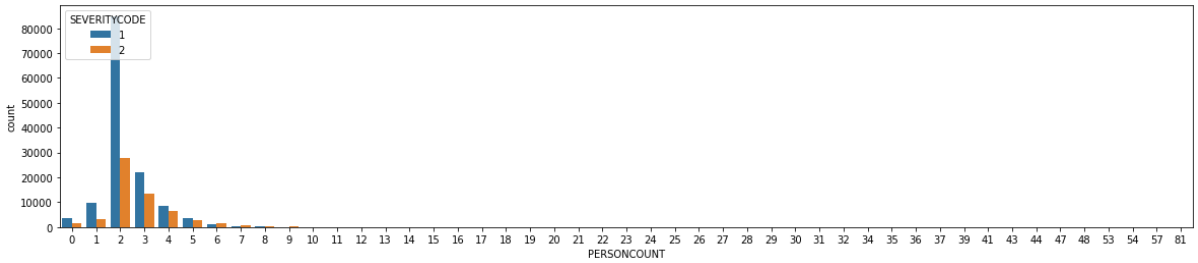
pre_data.drop('INTKEY', axis = 1, inplace = True)

PERSONCOUNT

Let us try to visualize this column.

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

sns.countplot(pre_data['PERSONCOUNT'], hue = pre_data['SEVERITYCODE'])



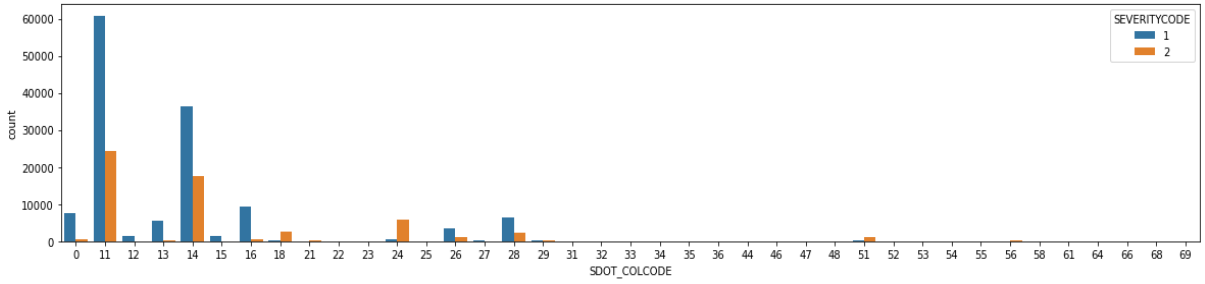
We can see that most car accidents involve two people. Nothing to be done on this column as everything is in place and no missing value.

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()

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

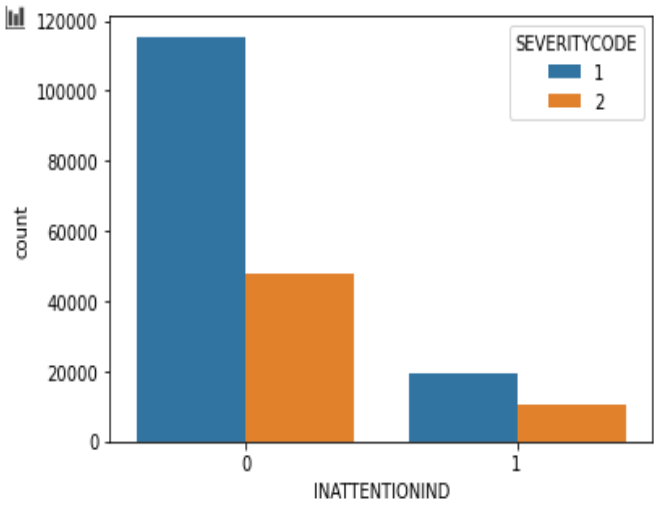
pre_data['INATTENTIONIND'].isna().sum()

```
163076
```

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()

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

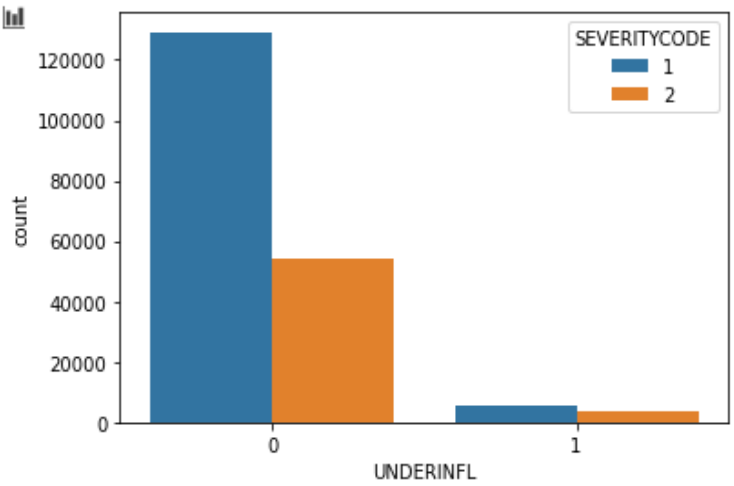
pre_data['UNDERINFL'].value_counts(dropna = False)

```
N      99155
0      79724
Y       5099
NaN     4777
1       3992
Name: UNDERINFL, dtype: int64
```

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)

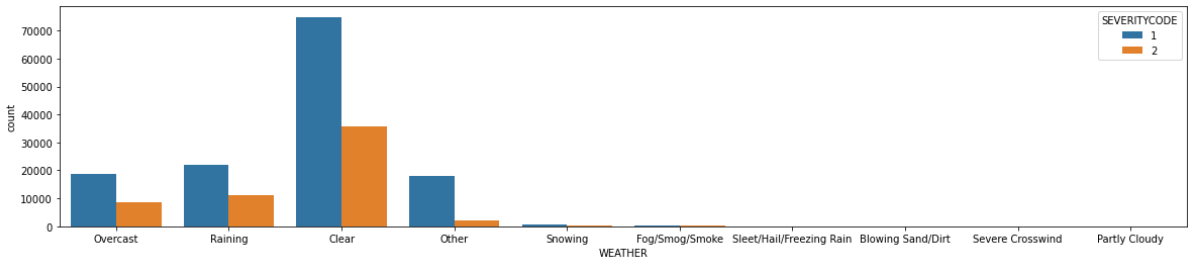
```
Clear                       110626
Raining                      33004
Overcast                     27584
Unknown                      14107
NaN                           4971
Snowing                        902
Other                          798
Fog/Smog/Smoke                 563
Sleet/Hail/Freezing Rain       112
Blowing Sand/Dirt               50
Severe Crosswind                25
Partly Cloudy                    5
Name: WEATHER, dtype: int64
```

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)

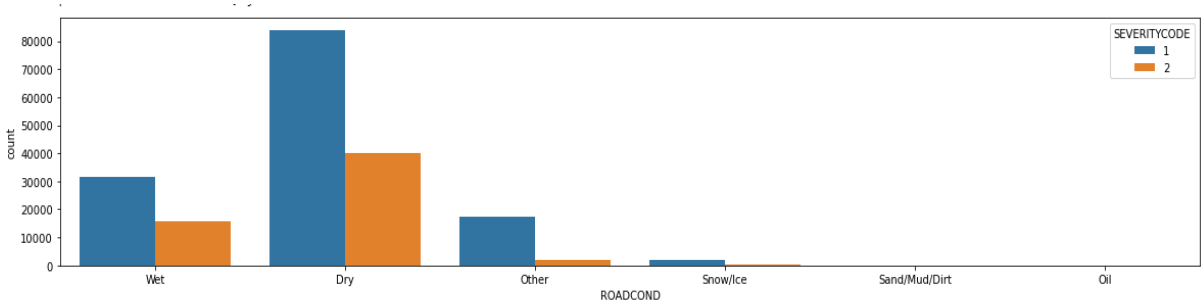
```
Dry              123945
Wet               47279
Unknown           14053
NaN                4903
Ice                1196
Snow/Slush          997
Other               125
Standing Water      111
Sand/Mud/Dirt        74
Oil                  64
Name: ROADCOND, dtype: int64
```

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)

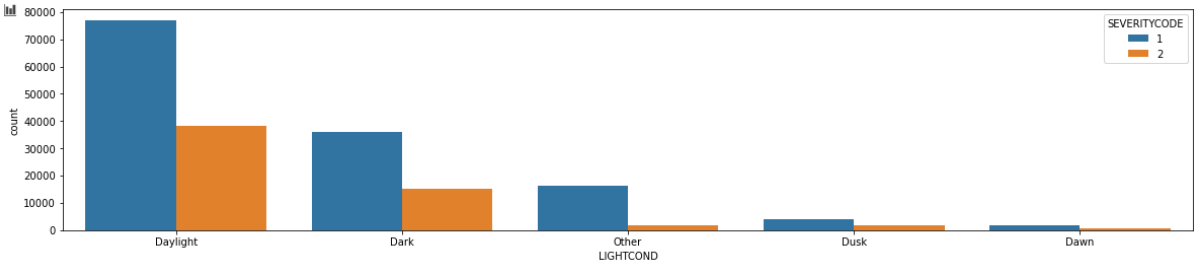
pre_data['LIGHTCOND'].value_counts(dropna = False)

```
Daylight                    115468
Dark - Street Lights On      48301
Unknown                      12616
Dusk                          5856
NaN                           5058
Dawn                          2491
Dark - No Street Lights       1528
Dark - Street Lights Off      1191
Other                          227
Dark - Unknown Lighting         11
Name: LIGHTCOND, dtype: int64
```

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.

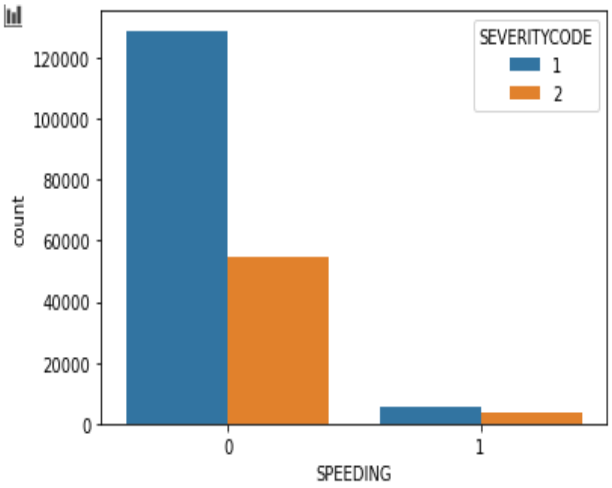
pre_data['SPEEDING'].value_counts(dropna = False)

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

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()

```
18
```

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()

```
array(['10', '11', '32', '23', '5', '22', '14', '30', ' ', '28', '51',
       '13', '50', '12', '45', '0', '20', '21', '1', '52', '16', '15',
       '74', '81', '26', '19', '2', '66', '71', '3', '24', '40', '57',
       '6', '83', '25', '27', '4', '72', '29', '56', '73', '41', '17',
       '65', '82', '67', '49', '84', '31', '43', '42', '48', '64', '53',
       32, 50, 15, 10, 14, 20, 13, 22, 51, 11, 28, 12, 52, 21, 0, 19, 30,
       16, 40, 26, 27, 83, 2, 45, 65, 23, 24, 71, 1, 29, 81, 25, 4, 73,
       74, 72, 3, 84, 64, 57, 42, 41, 48, 66, 56, 82, 67, '54', '60', 53,
       31, 43, 87, 54, '87', nan, '7', '8', '85', '88', '18'],
      dtype=object)
```

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()

```
4779
```

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

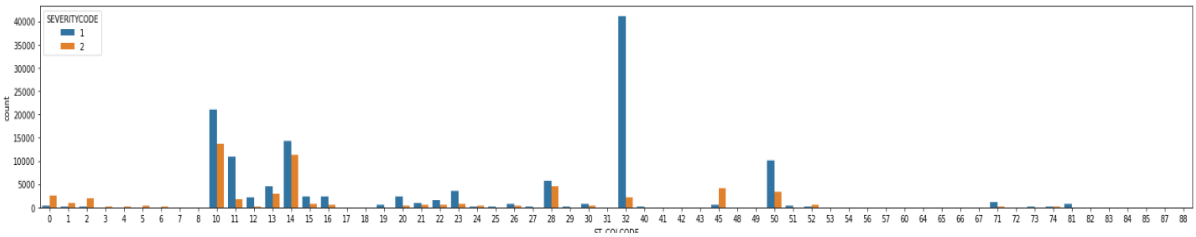
Next, we will convert everything else to int to make it easier when building the model.

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()

```
array([    0,  6855, 25242, ..., 42190, 11583, 10319], dtype=int64)
```

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

```
185220
```

pre_data['CROSSWALKKEY'].unique()

```
array([     0, 520838, 521466, ..., 525046, 523792, 523322], dtype=int64)
```

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

```
184187
```

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.

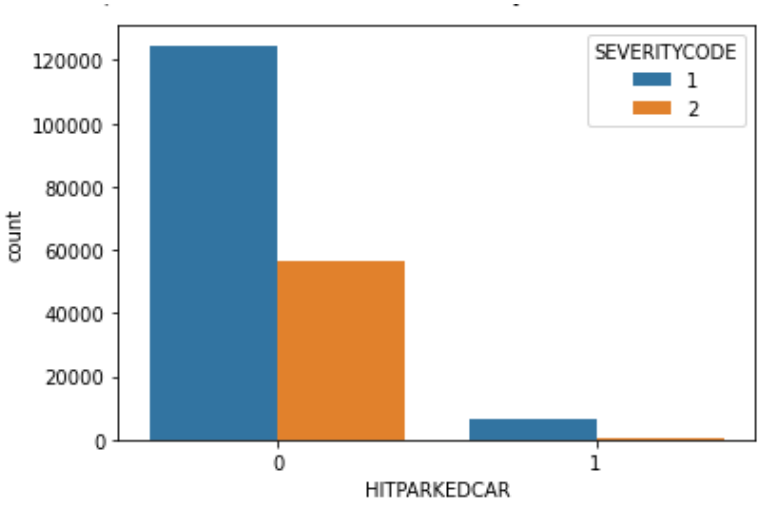
pre_data['HITPARKEDCAR'].unique()

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

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

And we can plot it.

sns.countplot(pre_data['HITPARKEDCAR'], hue = pre_data['SEVERITYCODE'])



3.4 REVIEWING THE CLEANED-UP DATA

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

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

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()

	SEVERITYCODE	ADDRTYPE	PERSONCOUNT	SDOT_COLCODE	INATTENTIONIND	UNDERINFL	WEATHER	ROADCOND	LIGHTCOND	SPEEDING	...	Snowing	Dry	Oil	Sand/Mud/Dirt	Snow/Ice	Wet	Dark	Dawn	Daylight	Dusk
0	2	Intersection	2	11	0	0	Overcast	Wet	Daylight	0	...	0	0	0	0	0	1	0	0	1	0
1	1	Block	2	16	0	0	Raining	Wet	Dark	0	...	0	0	0	0	0	1	1	0	0	0
2	1	Block	4	14	0	0	Overcast	Dry	Daylight	0	...	0	1	0	0	0	0	0	0	1	0
3	1	Block	3	11	0	0	Clear	Dry	Daylight	0	...	0	1	0	0	0	0	0	0	1	0
4	2	Intersection	2	11	0	0	Raining	Wet	Daylight	0	...	0	0	0	0	0	1	0	0	1	0

5 rows × 32 columns

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()

	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

5 rows × 28 columns

3.5 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)

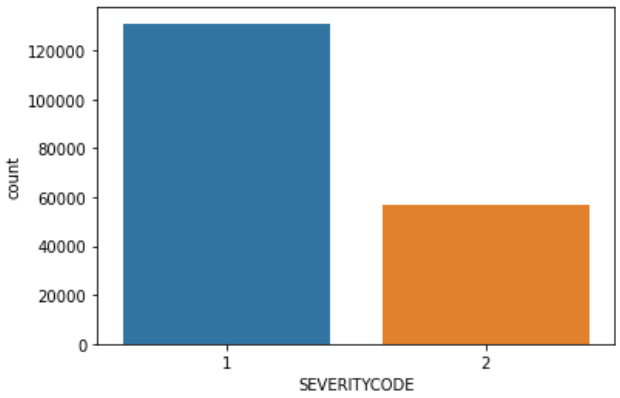
```
X_train.shape() =  (125926, 27) , y_train.shape() =  (125926,)
```

print('X_test.shape() = ', X_test.shape, ', y_test.shape() = ', y_test.shape)

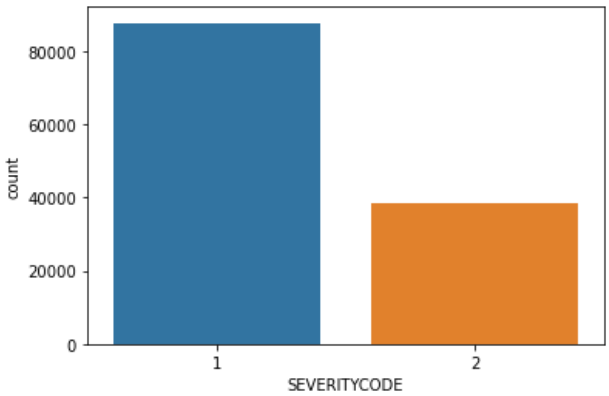
```
X_test.shape()  =  (62024, 27) , y_test.shape()  =  (62024,)
```

Remember that we have unbalanced dataset?

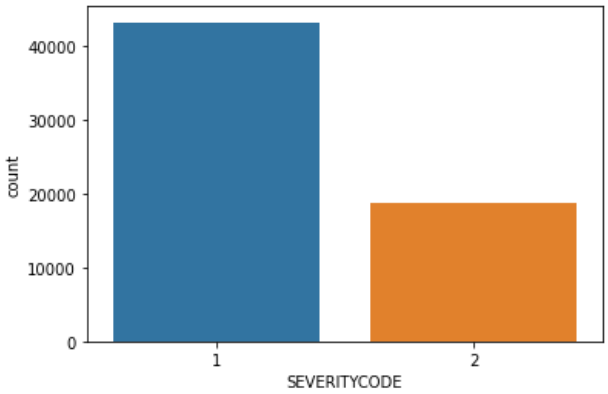
sns.countplot(pre_data['SEVERITYCODE'])



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()

	PERSONCOUNT	SDOT_COLCODE	INATTENTIONIND	UNDERINFL	SPEEDING	ST_COLCODE	HITPARKEDCAR	Block	Intersection	Blowing Sand/Dirt	...	Dry	Oil	Sand/Mud/Dirt	Snow/Ice	Wet	Dark	Dawn	Daylight	Dusk	SEVERITYCODE
164180	4	11	1	0	0	10	0	0	1	0	...	1	0	0	0	0	0	0	1	0	1
97934	5	11	0	0	0	20	0	1	0	0	...	0	0	0	0	1	0	1	0	0	1
118892	1	0	1	0	0	50	0	1	0	0	...	0	0	0	0	1	1	0	0	0	1
31489	2	14	0	0	0	12	0	0	1	0	...	1	0	0	0	0	0	0	1	0	1
171344	1	28	0	1	0	50	0	0	1	0	...	1	0	0	0	0	1	0	0	0	2

5 rows × 28 columns

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

```
SEVERITYCODE 1 =  87675
```

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

```
SEVERITYCODE 2 =  38251
```

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)

```
87675
```

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)
```

3.6 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))
```

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

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))
```

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

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))
```

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

4. 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
```

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

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))
```

```
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]]
```

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

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. 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.