SEATTLE TRAFFICK DATA ANALYSIS AND RISK PREDICTION

IBM Final Capstone Project

Puja Misra (10th Oct 2020)

1.Introduction

This project is our Final submission to IBM Data Science Professional Certificate course on Coursera. The goal of the project is to detail and use Data Science tool set for Predictive analysis.

We will be working on a real-life problem and demonstrate how Machine Learning can help us predict and process the value by applying the learned skills.

2.Business Understanding

- **2.1 Background:**

According to 2017 WSDOT data, a car accident occurs every 4 minutes and a person dies due to a car crash every 20 hours. Fatal crashes went from 508 in 2016 to 525 in 2017, resulting in the death of 555 people. This number has stayed relatively steady for the past decade. According to 2017 WSDOT data, a car accident occurs every 4 minutes and a person dies due to a car crash every 20 hours. Fatal crashes went from 508 in 2016 to 525 in 2017, resulting in the death of 555 people. This number has stayed relatively steady for the past decade.

- **2.2 Problem Statement:**

As we see in the background statement above, the numbers of fatal crashes are having an upward trend or has been steady for past decade for Seattle as per WSDOT.

- **2.3 Objective of the Project:**

- The purpose of the project is to gather the data and determine what causes the accident and the attributes that leads to the severity.

- Through data visualization and machine learning algorithm we will be analyzing a significant range of attributes, including weather conditions, road condition, speeding, special events, roadworks, traffic jams among others and we will try to predict what are the conditions that can contribute to high severity accidents which may cause loss of life or loss of property .WSOT can use the model to take precaution to minimize the loss of property and life.

- Reducing the insurance cost and Preventing fatalities

- **2.4 Stakeholders:**

- Government Officials
- Emergency Responders (911 dispatchers)
- Common People
- Insurance Companies

3.Data understanding

We chose the public data from open source available with labeled columns and attributes and observations data to help us do our analysis better.

Sample Data Below

Link to the Data

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

https://www.seattle.gov/Documents/Departments/SDOT/GIS/Collisions_OD.pdf

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
SEVERITYC	X	Y	OBJECTID	INCKEY	COLDETKE	REPORTNC	STATUS	ADDRTYPE	INTKEY	LOCATION	EXCEPTRSI	EXCEPTRSI	SEVERITYC	SEVERITYD	COLLISIC
2	-122.323	47.70314	1	1307	1307	3502005	Matched	Intersectic	37475	5TH AVE N			2	Injury Coll	Angles
1	-122.347	47.64717	2	52200	52200	2607959	Matched	Block		AURORA BR BETWEEN RAYE ST			1	Property D	Sideswip
1	-122.335	47.60787	3	26700	26700	1482393	Matched	Block		4TH AVE BETWEEN SENECA ST A			1	Property D	Parked C
1	-122.335	47.6048	4	1144	1144	3503937	Matched	Block		2ND AVE E			1	Property D	Other
2	-122.306	47.54574	5	17700	17700	1807429	Matched	Intersectic	34387	SWIFT AVE S AND SWIFT AV OFI			2	Injury Coll	Angles

The data consists of 40 independent variables and 221738 rows. The dependent variable, "SEVERITYCODE", contains numbers that correspond to different levels of severity caused by an accident from 0 to 4.

Severity codes are as follows:

0: Unknown

1: Property Damage

2: Injury

2b: Serious Injury

3: Fatality

4.Data Preparation

- 4.1 Public Traffic data for Seattle city USA is available from the Open source (Link Mentioned above)

- 4.2 After the data has been extracted, keeping the columns required in the data frame.

- 4.3 Excluding the rows with null values.

- 4.4 Transform the data type for analysis.

- Load Data to the Data frame.

    - <u>Original Size of the Data frame</u>

```
[221738 rows x 40 columns]>

[5]: #Saving the size of original DF
     print("Size  of DF :"+str(df.shape[0])+'x'+str(df.shape[1]))

     Size  of DF :221738x40
```

4.5 Missing Values from the Data frame

```
[7]: #Check where there are NaNs in the dataframe
print(df.isnull().sum(axis=0))
```

```
X                     7478
Y                     7478
OBJECTID                 0
INCKEY                   0
COLDETKEY                0
REPORTNO                 0
STATUS                   0
ADDRTYPE              3714
INTKEY              149711
LOCATION              4593
EXCEPTRSNCODE       120403
EXCEPTRSNDESC       209953
SEVERITYCODE             1
SEVERITYDESC             0
COLLISIONTYPE        26451
PERSONCOUNT              0
PEDCOUNT                 0
PEDCYLCOUNT              0
VEHCOUNT                 0
INJURIES                 0
SERIOUSINJURIES          0
FATALITIES               0
INCDATE                  0
INCDTTM                  0
JUNCTIONTYPE         11979
SDOT_COLCODE             1
SDOT_COLDESC             1
INATTENTIONIND      191550
UNDERINFL            26431
WEATHER              26641
ROADCOND             26560
LIGHTCOND            26730
PEDROWNOTGRNT       216543
SDOTCOLNUM           94533
SPEEDING            211802
ST_COLCODE            9413
ST_COLDESC           26451
SEGLANEKEY               0
CROSSWALKKEY             0
HITPARKEDCAR             0
dtype: int64
```

4.6 Preparing the Data after doing the Data Cleansing

```
16]: #Data Cleansing
#Find missing WEATHERCOND
todrop1 = df["WEATHER"] == 'Unknown'
noweatherinfo = todrop1.values.sum()

#Find missing ROADCOND
todrop2 = df["ROADCOND"] == "Unknown"
noroadcondinfo = todrop2.values.sum()

#Find missing LIGHTCOND
todrop3 = df["LIGHTCOND"] == "Unknown"
nolightinfo = todrop3.values.sum()


#Find missing LIGHTCOND
todrop4 = df["SEVERITYCODE"] == "2b"
noseveritycode = todrop4.values.sum()

#Collate these and remove
df["TODROP"] = 0
count_noweather = 0
count_noinfo = 0
for i in range(0,len(todrop1)):
    if todrop1[i] == True or todrop2[i] == True or todrop3[i] == True or todrop4[i] == True:
        df["TODROP"][i] = 1

print("There are "+str(noweatherinfo)+" accidents with no weather information.")
print("There are "+str(noroadcondinfo)+" accidents with no road condition information.")
print("There are "+str(nolightinfo)+" accidents with no information about light conditions.")
print("There are "+str(df["TODROP"].values.sum())+" accidents without one or more of the above.\n Deleting now...")

#Delete the temporary column "TODROP" and re-index the data
shape0 = df.shape
todrop = df["TODROP"] == 1
df.drop(df.index[todrop], inplace=True)
```

```
17]: #Take columns with mixed boolean data types ([1, "Y", True],[0, "N", False] etc) and cast them as numerical variables
#SPEEDING, INATTENTIONIND, UNDERINFL, PEDROWNOTGRNT, HITPARKEDCAR
df["SPEEDING"].replace(np.nan, 0, inplace=True)
df["SPEEDING"].replace("Y", 1, inplace=True)

df["INATTENTIONIND"].replace(np.nan, 0, inplace=True)
df["INATTENTIONIND"].replace("Y", 1, inplace=True)

df["INATTENTIONIND"].replace(np.nan, 0, inplace=True)
df["INATTENTIONIND"].replace("Y", 1, inplace=True)

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

df["PEDROWNOTGRNT"].replace(np.nan, 0, inplace=True)
df["PEDROWNOTGRNT"].replace("Y", 1, inplace=True)

df["HITPARKEDCAR"].replace("N", 0, inplace=True)
df["HITPARKEDCAR"].replace(np.nan, 0, inplace=True)
df["SPEEDING"].replace(np.nan, 0, inplace=True)
df["SPEEDING"].replace("Y", 1, inplace=True)
df["HITPARKEDCAR"].replace("Y", 1, inplace=True)
```

4.7 Data Cleansing-Dropping the unwanted columns.

```
[13]: if 'OBJECTID' in df:
    del df["OBJECTID"]
if 'COLDETKEY' in df:
    del df["COLDETKEY"]
if 'REPORTNO' in df:
    del df["REPORTNO"]
if 'STATUS' in df:
    del df["STATUS"]
if 'EXCEPTRSNCODE' in df:
    del df["EXCEPTRSNCODE"]
if 'EXCEPTRSNDESC' in df:
    del df["EXCEPTRSNDESC"]
if 'SDOTCOLNUM' in df:
    del df["SDOTCOLNUM"]
if 'STCOLCODE' in df:
    del df["STCOLCODE"]
if 'INTKEY' in df:
    del df["INTKEY"]
if 'LOCATION' in df:
    del df["LOCATION"]
if 'SEVERITYDESC' in df:
    del df["SEVERITYDESC"]
```

4.8 Casting the columns to right datatype (numerical variables) for calculations.

```
[17]: #Take columns with mixed boolean data types ([1, "Y", True],[0, "N", False] etc) and cast them as numerical variables
#SPEEDING, INATTENTIONIND, UNDERINFL, PEDROWNOTGRNT, HITPARKEDCAR
df["SPEEDING"].replace(np.nan, 0, inplace=True)
df["SPEEDING"].replace("Y", 1, inplace=True)

df["INATTENTIONIND"].replace(np.nan, 0, inplace=True)
df["INATTENTIONIND"].replace("Y", 1, inplace=True)

df["INATTENTIONIND"].replace(np.nan, 0, inplace=True)
df["INATTENTIONIND"].replace("Y", 1, inplace=True)

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

df["PEDROWNOTGRNT"].replace(np.nan, 0, inplace=True)
df["PEDROWNOTGRNT"].replace("Y", 1, inplace=True)

df["HITPARKEDCAR"].replace("N", 0, inplace=True)
df["HITPARKEDCAR"].replace(np.nan, 0, inplace=True)
df["SPEEDING"].replace(np.nan, 0, inplace=True)
df["SPEEDING"].replace("Y", 1, inplace=True)
df["HITPARKEDCAR"].replace("Y", 1, inplace=True)
```

```
[21]: #Descriptive Stats
descriptive_stats= df.describe(include="all")
```

```
[22]: print(df.SEVERITYCODE.value_counts())

1    95913
2    49569
3      240
0        2
Name: SEVERITYCODE, dtype: int64
```

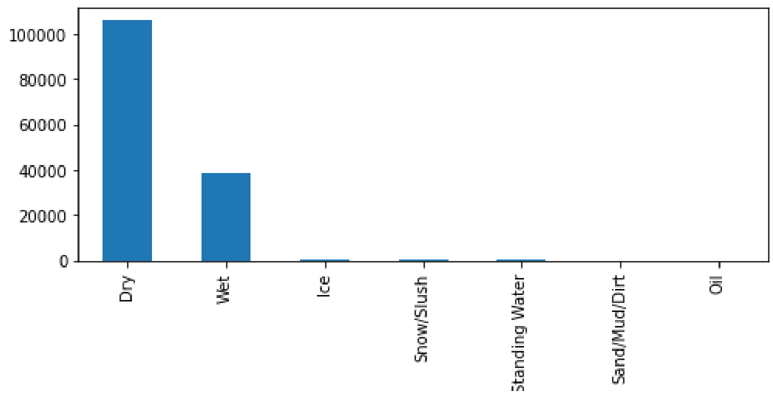
4.9 Count Based on the Road Condition

[25]:
```
print(df.ROADCOND.value_counts())
```

```
Dry                105934
Wet                 38373
Ice                   687
Snow/Slush            629
Standing Water         50
Sand/Mud/Dirt          30
Oil                    21
Name: ROADCOND, dtype: int64
```

[27]:
```
df.ROADCOND.value_counts().plot.bar(figsize=(8,3))
```

[27]: <AxesSubplot:>



5.Exploratory Data Analysis

We will run a value count on road ('ROADCOND') and weather condition ('WEATHER') to get ideas of the different road and weather conditions. We will also check the value count on light condition ('LIGHTCOND'), to see the breakdowns of accidents occurring during the different light conditions. The results will then be used for data modeling.

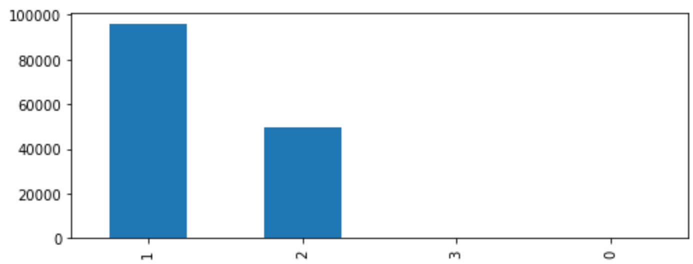
5.1 SEVERITY CODE count

```
[22]: print(df.SEVERITYCODE.value_counts())
```

```
1    95913
2    49569
3      240
0        2
Name: SEVERITYCODE, dtype: int64
```

```
[23]: df.SEVERITYCODE.value_counts().plot.bar(figsize=(8,3))
```

```
[23]: <AxesSubplot:>
```



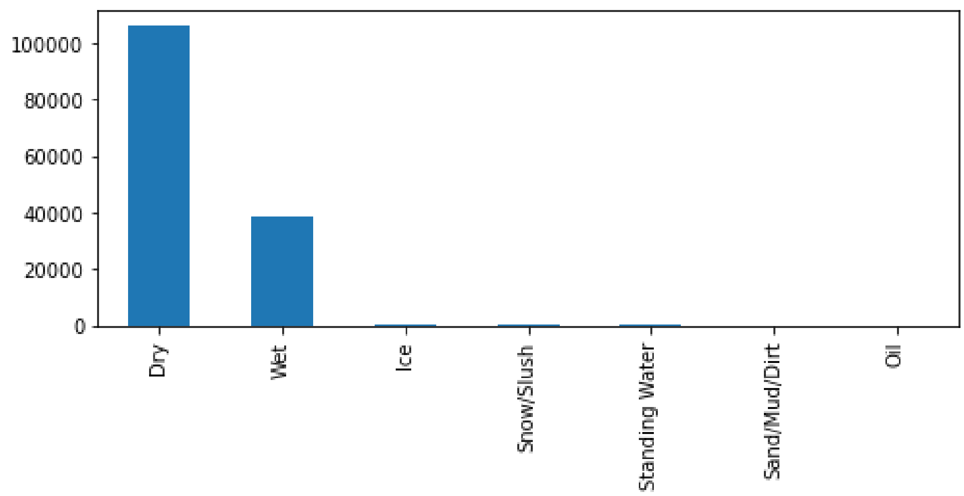
5.2 ROADCOND count

```
[24]: print(df.ROADCOND.value_counts())
```

```
Dry               105934
Wet                38373
Ice                  687
Snow/Slush           629
Standing Water        50
Sand/Mud/Dirt         30
Oil                   21
Name: ROADCOND, dtype: int64
```

```
[25]: df.ROADCOND.value_counts().plot.bar(figsize=(8,3))
```

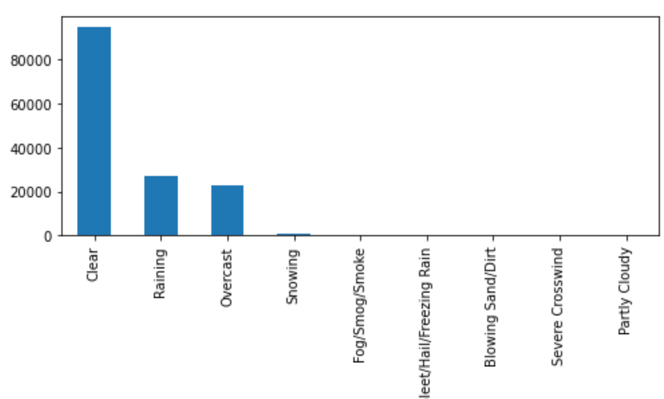
```
[25]: <AxesSubplot:>
```



5.3 WEATHER count

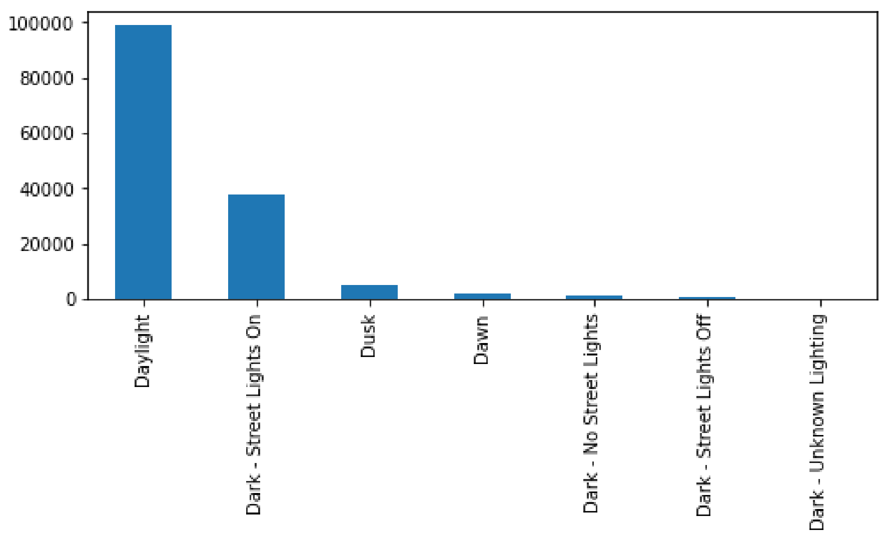
```
[25]: df.WEATHER.value_counts().plot.bar(figsize=(8,3))
```

```
[25]: <AxesSubplot:>
```



5.4 LIGHTCOND count

```
[24]: df.LIGHTCOND.value_counts().plot.bar(figsize=(8,3))

[24]: <AxesSubplot:>
```



Graphs built with seaborn library below to check how the severity is impacted by various attributes.

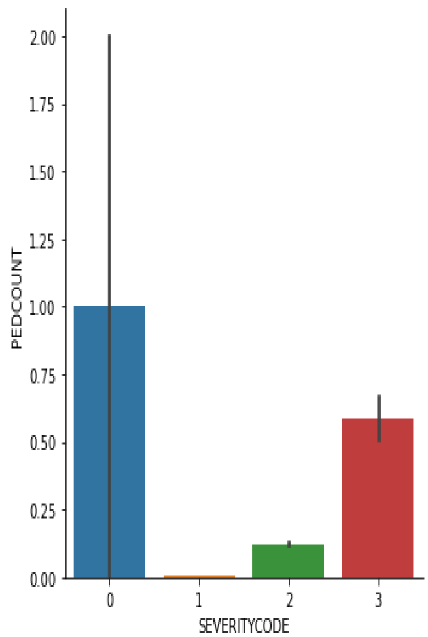
<u>OBSERVATIONS:</u>

- **1.1 Number of pedestrians involved in the collision (PEDCOUNT) and Severity**

- **1.2 Severity 2 (Injury) accidents has happened to Pedestrians, compared to SERV 1 accidents**

```
28]: # The number of pedestrians involved in the collision.
     import seaborn as sns
     sns.catplot(x="SEVERITYCODE", y="PEDCOUNT", data=df, kind="bar")
```

```
28]: <seaborn.axisgrid.FacetGrid at 0x7f2c1bf59198>
```

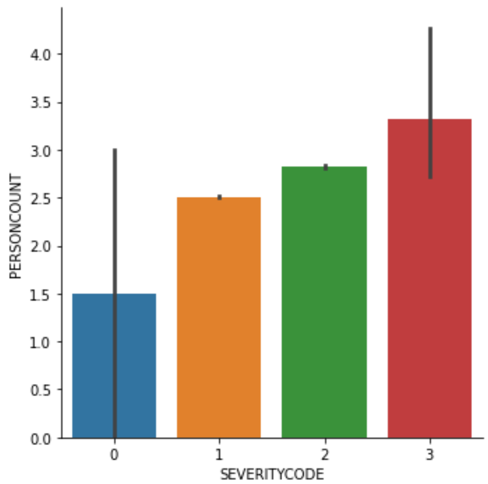


- **1.3 Number of people involved in the collision (PERSONCOUNT) and Severity¶**

1.4 The data shows that the severity of the accident is high with person count.

```
[27]: # The number of pedestrians involved in the collision.
import seaborn as sns
sns.catplot(x="SEVERITYCODE", y="PERSONCOUNT", data=df, kind="bar")
```

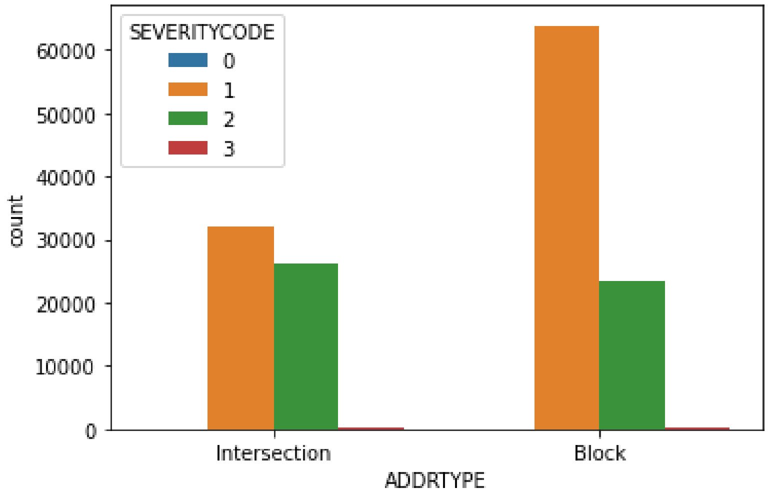
```
[27]: <seaborn.axisgrid.FacetGrid at 0x7fa4e90fbda0>
```



- **1.5 The below data shows accidents are happening more near the blocks and less at intersection. Severity 2 is almost same on block and on Intersection**

```
]: # Count plot for Address type :
sns.countplot(x='ADDRTYPE', data=df, hue='SEVERITYCODE')
```

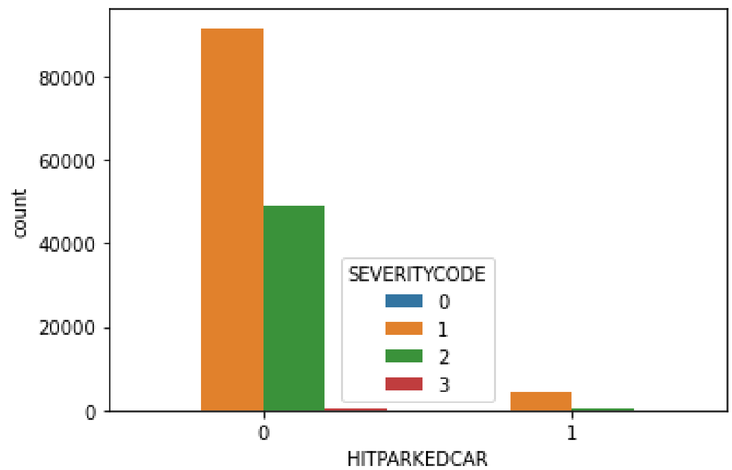
```
]: <AxesSubplot:xlabel='ADDRTYPE', ylabel='count'>
```



- **1.6 Number of cases of accidents when car was parked**


```
[32]: sns.countplot(x='HITPARKEDCAR', data=df, hue='SEVERITYCODE')
```

```
[32]: <AxesSubplot:xlabel='HITPARKEDCAR', ylabel='count'>
```

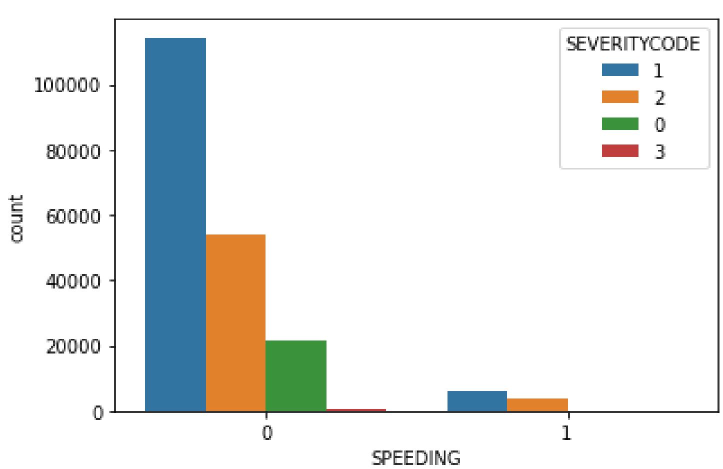


1.7 Looking at the Speeding cases, there were 9381 speeding cases.

HITPARKEDCAR

```
[51]: sns.countplot(x='SPEEDING', data=df, hue='SEVERITYCODE')
```

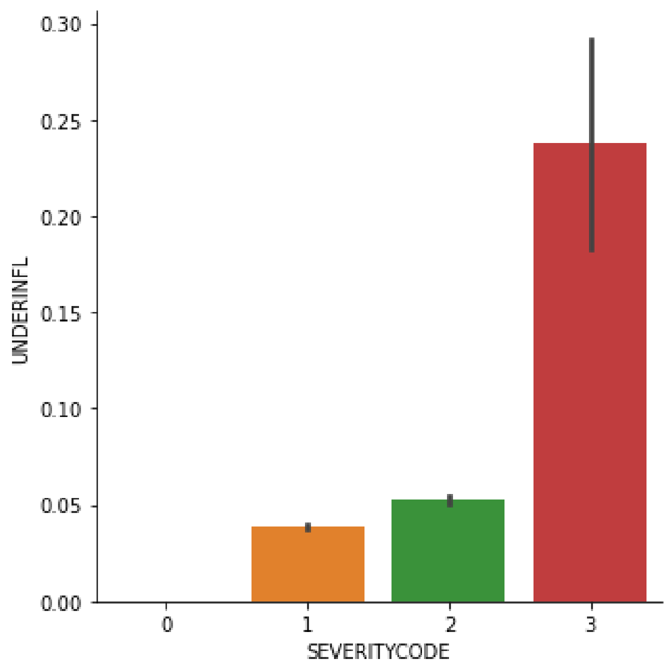
```
[51]: <AxesSubplot:xlabel='SPEEDING', ylabel='count'>
```



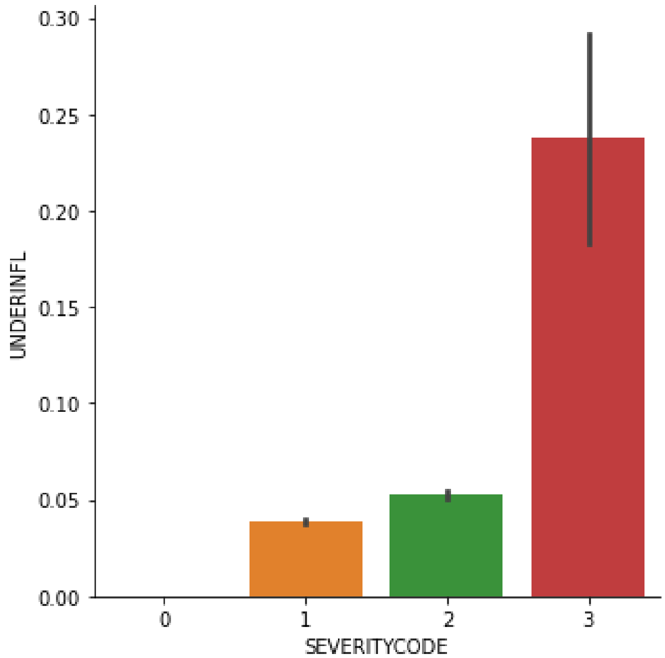
- **1.8 Figure shows the high number of accidents when people were under influence.**

[28]:
```python
# The number of pedestrians involved in the collision.
import seaborn as sns
sns.catplot(x="SEVERITYCODE", y="UNDERINFL", data=df, kind="bar")
```

[28]: <seaborn.axisgrid.FacetGrid at 0x7f5eadc8fe10>



```
[28]: # The number of pedestrians involved in the collision.
      import seaborn as sns
      sns.catplot(x="SEVERITYCODE", y="UNDERINFL", data=df, kind="bar")

[28]: <seaborn.axisgrid.FacetGrid at 0x7f5eadc8fe10>
```



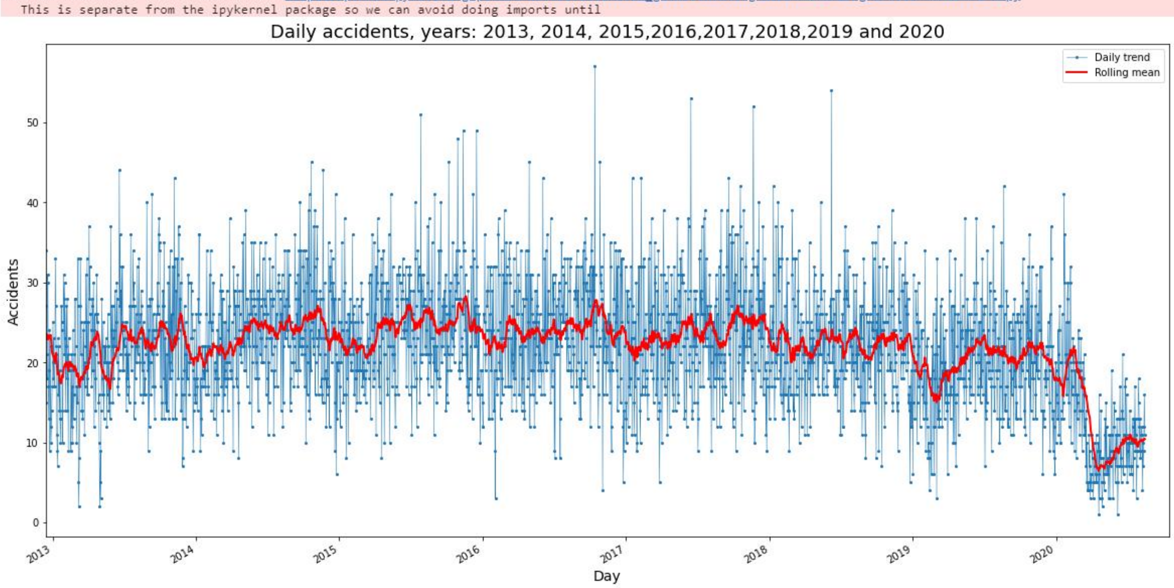
For further analysis we created an Incident dataset.

Following features were used: "Incident Date", "Incident Time”, “Collision Time”, “Collision condition”, “Light Condition” indicating the Incident Detail.

```
26]: #Daily accidents based on the Year
date['year'] = df.INCDATE.dt.year
date['month'] = df.INCDATE.dt.month
date['weekday'] = df.INCDATE.dt.weekday
high_sev = date[date['SEVERITYCODE']==1]

season = date[['INCDATE', 'SEVERITYCODE']].groupby('INCDATE').count()
season['rolling'] = season.SEVERITYCODE.rolling(window=30).mean()
season['SEVERITYCODE'][365*8:].plot(figsize=(20,10), marker='o', markersize=2, linewidth=0.5, label='Daily trend')
season['rolling'][365*8:].plot(color='r', linewidth=2, label='Rolling mean')
plt.title('Daily accidents, years: 2013, 2014, 2015,2016,2017,2018,2019 and 2020', size=18)
plt.xlabel('Day', size=14)
plt.ylabel('Accidents', size=14)

t0 = dt.datetime.strptime('2012-12-15', '%Y-%m-%d')
t1 = dt.datetime.strptime('2020-10-15', '%Y-%m-%d')

plt.xlim(t0,t1)
plt.legend()
```

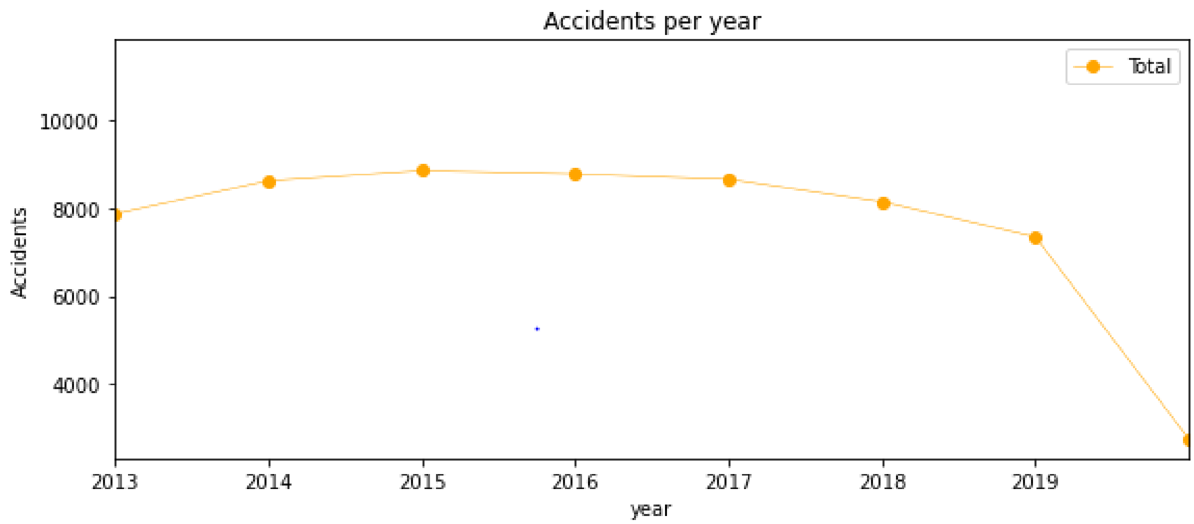
1.9 The Below graph shows the daily accidents from 2013 till 2020 with high around end of 2016 and beginning of 2017 and the number of accidents low in 2020.

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```



1.10 Plotting the accidents Per year, confirms the same trend as above

[33]:
```python
#By year

yearly = date[['year', 'SEVERITYCODE']].groupby('year').count()
yearly['SEVERITYCODE'].plot.line(figsize=(10,4), marker='o', linewidth=0.5, color='orange', label='Total')
plt.title('Accidents per year')
plt.xticks(range(2012,2020))
plt.xlim(2013,2020)
plt.ylabel('Accidents')
plt.legend()
plt.show()
```



6.Modeling

6.1 Using NumPy, Scalar, Linear regression on the clean transformed data (Shown Above)

6.2 After importing necessary packages and splitting preprocessed data into test and train sets, for each machine learning model, we will build and evaluate the model with the techniques as follow:

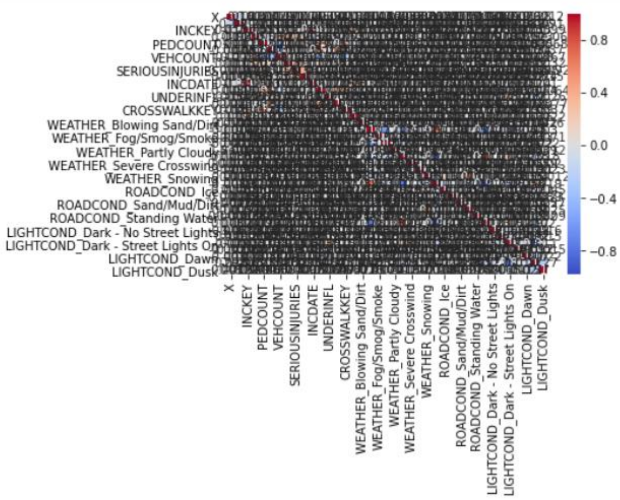
6.3 Data frame with features below created [“ADDRTYPE”,” COLLISIONTYPE,”

JUNCTIONTYPE”,” WEATHER”,” ROADCOND”,” LIGHTCOND”,” UNDERINFL”,” HITPARKEDCAR”]

```
[66]: X = df_balanced.iloc[:,1:]

      #Encoding Categorical Features - Training Dataset
      X = pd.get_dummies(data=X, columns=['ADDRTYPE','COLLISIONTYPE','JUNCTIONTYPE','WEATHER',
                                          'ROADCOND','LIGHTCOND','UNDERINFL','HITPARKEDCAR'])

      Y = df_balanced[['SEVERITYCODE']]
      print(X.info())
      X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.25,random_state=0)
      xtrain, xval, ytrain, yval = train_test_split(X_train, Y_train, test_size=0.25)
      print(Y_train.info())
      print(Y_test.info())
```

Heatmap

```
[53]: sns.heatmap(df_corr,annot=True,cmap='coolwarm')
      plt.savefig('corr.png')
```



7.Machine Learning Models

- 7.1 GitHub as a repository and running Jupiter Notebook are used to process data and build Machine Learning models
- 7.2 Python and its popular packages such as Pandas, NumPy and Sklearn is used for determining the accuracy.
- 7.3 The dataset x and y are constructed. After normalization they are split into x_train,y_train,x_test and y_test using train_test split.75% of the data is used for training and 25% is used for testing as below

```
[41]: X = df_balanced.iloc[:,1:]

      #Encoding Categorical Features - Training Dataset
      X = pd.get_dummies(data=X, columns=['ADDRTYPE','COLLISIONTYPE','JUNCTIONTYPE','WEATHER',
                                          'ROADCOND','LIGHTCOND','UNDERINFL','HITPARKEDCAR'])

      Y = df_balanced[['SEVERITYCODE']]
      print(X.info())
      X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.25,random_state=0)
      xtrain, xval, ytrain, yval = train_test_split(X_train, Y_train, test_size=0.25)
      print(Y_train.info())
      print(Y_test.info())
```

- K Nearest Neighbour (KNN)

```
Best Hyperparameter KNN :  {'n_neighbors': 6, 'p': 1}
[[    0     1     0     0]
 [    0 22073  1991     0]
 [    0 11024  1280     0]
 [    0    57     5     0]]

/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/metr
-score are ill-defined and being set to 0.0 in labels with no predicted sa
  'precision', 'predicted', average, warn_for)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.67      0.92      0.77     24064
           2       0.39      0.10      0.16     12304
           3       0.00      0.00      0.00        62

   micro avg       0.64      0.64      0.64     36431
   macro avg       0.26      0.26      0.23     36431
weighted avg       0.57      0.64      0.57     36431
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/model_selection/_split.py:2053: FutureWarning: You sho
ue for 'cv' instead of relying on the default value. The default value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/model_selection/_split.py:652: Warning: The least popu
has only 1 members, which is too few. The minimum number of members in any class cannot be less than n_splits=3.
  % (min_groups, self.n_splits)), Warning)
Best Hyperparameter KNN :  {'n_neighbors': 6, 'p': 1}
[[    0     1     0     0]
 [    0 22073  1991     0]
 [    0 11024  1280     0]
 [    0    57     5     0]]

/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning
-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.67      0.92      0.77     24064
           2       0.39      0.10      0.16     12304
           3       0.00      0.00      0.00        62

   micro avg       0.64      0.64      0.64     36431
   macro avg       0.26      0.26      0.23     36431
weighted avg       0.57      0.64      0.57     36431

0.6410200104306771
```

- LINEAR REGRESSION

```
  "this warning.", FutureWarning)
['1' '1' '1' ... '1' '1' '1']

[[    0     1     0     0]
 [    0 24064     0     0]
 [    0 12304     0     0]
 [    0    62     0     0]]

/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precis
-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.66      1.00      0.80     24064
           2       0.00      0.00      0.00     12304
           3       0.00      0.00      0.00        62

   micro avg       0.66      0.66      0.66     36431
   macro avg       0.17      0.25      0.20     36431
weighted avg       0.44      0.66      0.53     36431


0.6605363563997694
```

- DECISION TREE

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/model_selectio
ue for 'cv' instead of relying on the default value. The default value will change fr
  warnings.warn(CV_WARNING, FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/model_selectio
has only 1 members, which is too few. The minimum number of members in any class cann
  % (min_groups, self.n_splits)), Warning)
Best Hyperparameter DTC :  {'criterion': 'gini', 'random_state': 0}
[[    1     0     0     0]
 [    0 24064     0     0]
 [    0     0 12304     0]
 [    0     0     0    62]]

              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00     24064
           2       1.00      1.00      1.00     12304
           3       1.00      1.00      1.00        62

   micro avg       1.00      1.00      1.00     36431
   macro avg       1.00      1.00      1.00     36431
weighted avg       1.00      1.00      1.00     36431


1.0
```

Based on the data sample of train and test data we see there is an Overfitting, meaning our model is learning the noise from the data and its ability to generalize the results is very low. In this case you have a small training error but very large validation error. So, we will try to prune the data set and run the model again.

We changed our dataset and started again to find the accuracy and which model fits the best.

- Dataframe:data_clean

```
memory usage: 15.0+ MB

[68]: data_clean = pd.concat([data_clean.drop(['WEATHER', 'ROADCOND', 'LIGHTCOND'], axis=1),
                 pd.get_dummies(data_clean['ROADCOND']),
                 pd.get_dummies(data_clean['LIGHTCOND']),
                 pd.get_dummies(data_clean['WEATHER'])], axis=1)

[69]: data_clean = data_clean.sample(frac=1).reset_index(drop=True)

[70]: from sklearn import preprocessing
      x = data_clean.drop(['SEVERITYCODE'], axis=1)
      y = data_clean[['SEVERITYCODE']]
      data_clean_scaled = preprocessing.StandardScaler().fit(x).transform(x)
      data_clean_scaled[0:3]
```

- DECISION TREE

  Accuracy Score :0.63

```
[72]: #DECISION TREE ON SECOND SET OF DATA
from sklearn.tree import DecisionTreeClassifier
dTreeModel = DecisionTreeClassifier(criterion='entropy', max_depth=5)
dTreeModel.fit(x_train, y_train)
dTreeModel
```

```
[72]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
[73]: yHat = dTreeModel.predict(x_test)
```

```
[74]: print(classification_report(y_test, yHat))
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWa
0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.71      0.98      0.83     19147
           2       0.86      0.24      0.37      9945
           3       0.00      0.00      0.00        52

   micro avg       0.73      0.73      0.73     29145
   macro avg       0.39      0.31      0.30     29145
weighted avg       0.76      0.73      0.67     29145
```

```
[75]: #Accuracy
acc = accuracy_score(Y_test,yHat)
print(acc,'\n')
accDict['RFT'] = acc
```

```
0.6296791902556185
```

- LINEAR REGRESSION

  Accuracy Score: 0.72

```
nged to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/utils/validation.py:761: DataConversionWarning: A column-vector
assed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class
e changed to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
['1' '1' '1' ... '1' '1' '1']

[[18632   556     0]
 [ 7430  2480     0]
 [   16    31     0]]

/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precisio
-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
              precision    recall  f1-score   support

           1       0.71      0.97      0.82     19188
           2       0.81      0.25      0.38      9910
           3       0.00      0.00      0.00        47

   micro avg       0.72      0.72      0.72     29145
   macro avg       0.51      0.41      0.40     29145
weighted avg       0.75      0.72      0.67     29145


0.7243781094527363
```

- RANDOM FOREST

Accuracy Score:0.73

```
yHat = rfcModel.predict(x_test)
```

```
print(classification_report(y_test, yHat))
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricW
samples.
  'precision', 'predicted', average, warn_for)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.72      0.96      0.83     19147
           2       0.79      0.28      0.42      9945
           3       0.00      0.00      0.00        52

   micro avg       0.73      0.73      0.73     29145
   macro avg       0.38      0.31      0.31     29145
weighted avg       0.74      0.73      0.69     29145
```

```
#Accuracy
acc = accuracy_score(yHat,y_test)
print(acc,'\n')
accDict = {}
accDict['RF1'] = acc
```

```
0.7301423914908217
```

- KNN

  Accuracy Score:0.72

```
[87]: #KNN ON SECOND SET OF DATA

      from sklearn.neighbors import KNeighborsClassifier
      classifier = KNeighborsClassifier(n_neighbors = 8)
      classifier.fit(x_train, y_train)

/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please chang
), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports until

[87]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                 metric_params=None, n_jobs=None, n_neighbors=8, p=2,
                 weights='uniform')

[88]: y_pred = classifier.predict(x_test)

[89]: #Accuracy
      acc = accuracy_score(y_pred,y_test)
      print(acc,'\n')
      accDict['KNN'] = acc

      0.7234517069823297
```

8.CONCLUSION

With the help of machine learning algorithms, we were able to predict the impact of weather, road condition and light condition on the seriousness of the accidents. Property damage (class 1) or injury (class 2) with graphs.

From the Machine Learning model, we saw except Decision Tree the other three models have a close accuracy of 72–73% which means that the model has trained well.

KNN, Logistic Regression, Random forest have an accuracy of 72–74%, this is because of the similarity of the features for both types of accidents (1 and 2)

Though with 73% of accuracy from Random Forest we can say that the model has trained well and performs well on the testing as well as the trained data.

The model is ready to be used.

9.FUTURE DIRECTIONS

There is more scope of improvement in terms of accuracy by adding additional features in the data set and by having fewer null values and more data for speeding and other important columns (like attention id, under influence and collision type). Missing Data List shown below.

[7]:
```
#Check where there are NaNs in the dataframe
print(df.isnull().sum(axis=0))
```

```
X                    7478
Y                    7478
OBJECTID                0
INCKEY                  0
COLDETKEY               0
REPORTNO                0
STATUS                  0
ADDRTYPE             3714
INTKEY             149711
LOCATION             4593
EXCEPTRSNCODE      120403
EXCEPTRSNDESC      209953
SEVERITYCODE            1
SEVERITYDESC            0
COLLISIONTYPE       26451
PERSONCOUNT             0
PEDCOUNT                0
PEDCYLCOUNT             0
VEHCOUNT                0
INJURIES                0
SERIOUSINJURIES         0
FATALITIES              0
INCDATE                 0
INCDTTM                 0
JUNCTIONTYPE        11979
SDOT_COLCODE            1
SDOT_COLDESC            1
INATTENTIONIND     191550
UNDERINFL           26431
WEATHER             26641
ROADCOND            26560
LIGHTCOND           26730
PEDROWNOTGRNT      216543
SDOTCOLNUM          94533
SPEEDING           211802
ST_COLCODE           9413
ST_COLDESC          26451
SEGLANEKEY              0
CROSSWALKKEY            0
HITPARKEDCAR            0
dtype: int64
```