Road Accident Severity Classification

In [2]:

```
!pip install imbalanced-learn
```

```
Defaulting to user installation because normal site-packages is not writeabl
e
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.10.1-py3-none-any.whl (226 kB)
     ------------------------------------ 226.0/226.0 kB 726.5 kB/s eta 0:0
0:00
Requirement already satisfied: scipy>=1.3.2 in c:\programdata\anaconda3\lib
\site-packages (from imbalanced-learn) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anacon
da3\lib\site-packages (from imbalanced-learn) (2.2.0)
Requirement already satisfied: numpy>=1.17.3 in c:\programdata\anaconda3\lib
\site-packages (from imbalanced-learn) (1.21.5)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\programdata\anacond
a3\lib\site-packages (from imbalanced-learn) (1.0.2)
Collecting joblib>=1.1.1
  Downloading joblib-1.2.0-py3-none-any.whl (297 kB)
     -------------------------------------- 298.0/298.0 kB 1.5 MB/s eta 0:0
0:00
Installing collected packages: joblib, imbalanced-learn
Successfully installed imbalanced-learn-0.10.1 joblib-1.2.0
```

In [3]:

```
#import the necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
from collections import Counter
from imblearn.over_sampling import SMOTE
import matplotlib.ticker as ticker
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,MinMaxScaler,LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold,GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoostingClas
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.model_selection import KFold # import KFold
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [4]:

```
df = pd.read_csv("RTA Dataset.csv")
```

Let's take a look at the dataset

In [5]:

```
df.head()
```

Out[5]:

	Time	Day_of_week	Age_band_of_driver	Sex_of_driver	Educational_level	Vehicle_driver_
0	17:02:00	Monday	18-30	Male	Above high school	Er
1	17:02:00	Monday	31-50	Male	Junior high school	Er
2	17:02:00	Monday	18-30	Male	Junior high school	Er
3	1:06:00	Sunday	18-30	Male	Junior high school	Er
4	1:06:00	Sunday	18-30	Male	Junior high school	Er

5 rows × 32 columns

In [6]:

```
df.sample(5)
```

Out[6]:

	Time	Day_of_week	Age_band_of_driver	Sex_of_driver	Educational_level	Vehicle_driv
2084	7:59:00	Friday	31-50	Male	Junior high school	
8376	20:00:00	Thursday	Over 51	Male	Elementary school	
8672	15:30:00	Friday	18-30	Male	Junior high school	
2825	1:05:00	Saturday	31-50	Male	Junior high school	
873	11:29:00	Saturday	Over 51	Male	Junior high school	

5 rows × 32 columns

In [7]:

```
df.shape
```

Out[7]:

```
(12316, 32)
```

In [8]:

```
df.columns
```

Out[8]:

```
Index(['Time', 'Day_of_week', 'Age_band_of_driver', 'Sex_of_driver',
       'Educational_level', 'Vehicle_driver_relation', 'Driving_experience',
       'Type_of_vehicle', 'Owner_of_vehicle', 'Service_year_of_vehicle',
       'Defect_of_vehicle', 'Area_accident_occured', 'Lanes_or_Medians',
       'Road_allignment', 'Types_of_Junction', 'Road_surface_type',
       'Road_surface_conditions', 'Light_conditions', 'Weather_conditions',
       'Type_of_collision', 'Number_of_vehicles_involved',
       'Number_of_casualties', 'Vehicle_movement', 'Casualty_class',
       'Sex_of_casualty', 'Age_band_of_casualty', 'Casualty_severity',
       'Work_of_casuality', 'Fitness_of_casuality', 'Pedestrian_movement',
       'Cause_of_accident', 'Accident_severity'],
      dtype='object')
```

In [9]:

```
df.describe(include="all")
```

Out[9]:

	Time	Day_of_week	Age_band_of_driver	Sex_of_driver	Educational_level	Vehicle_d
count	12316	12316	12316	12316	11575	
unique	1074	7	5	3	7	
top	15:30:00	Friday	18-30	Male	Junior high school	
freq	120	2041	4271	11437	7619	
mean	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	NaN	

11 rows × 32 columns

In [10]:

```
df.dtypes
```

Out[10]:

```
Time                          object
Day_of_week                   object
Age_band_of_driver            object
Sex_of_driver                 object
Educational_level             object
Vehicle_driver_relation       object
Driving_experience            object
Type_of_vehicle               object
Owner_of_vehicle              object
Service_year_of_vehicle       object
Defect_of_vehicle             object
Area_accident_occured         object
Lanes_or_Medians              object
Road_allignment               object
Types_of_Junction             object
Road_surface_type             object
Road_surface_conditions       object
Light_conditions              object
Weather_conditions            object
Type_of_collision             object
Number_of_vehicles_involved    int64
Number_of_casualties           int64
Vehicle_movement              object
Casualty_class                object
Sex_of_casualty               object
Age_band_of_casualty          object
Casualty_severity             object
Work_of_casuality             object
Fitness_of_casuality          object
Pedestrian_movement           object
Cause_of_accident             object
Accident_severity             object
dtype: object
```

In [11]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12316 entries, 0 to 12315
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Time                     12316 non-null  object
 1   Day_of_week              12316 non-null  object
 2   Age_band_of_driver       12316 non-null  object
 3   Sex_of_driver            12316 non-null  object
 4   Educational_level        11575 non-null  object
 5   Vehicle_driver_relation  11737 non-null  object
 6   Driving_experience       11487 non-null  object
 7   Type_of_vehicle          11366 non-null  object
 8   Owner_of_vehicle         11834 non-null  object
 9   Service_year_of_vehicle  8388 non-null   object
 10  Defect_of_vehicle        7889 non-null   object
 11  Area_accident_occured    12077 non-null  object
 12  Lanes_or_Medians         11931 non-null  object
 13  Road_allignment          12174 non-null  object
 14  Types_of_Junction        11429 non-null  object
 15  Road_surface_type        12144 non-null  object
 16  Road_surface_conditions  12316 non-null  object
 17  Light_conditions         12316 non-null  object
 18  Weather_conditions       12316 non-null  object
 19  Type_of_collision        12161 non-null  object
 20  Number_of_vehicles_involved  12316 non-null  int64 
 21  Number_of_casualties     12316 non-null  int64 
 22  Vehicle_movement         12008 non-null  object
 23  Casualty_class           12316 non-null  object
 24  Sex_of_casualty          12316 non-null  object
 25  Age_band_of_casualty     12316 non-null  object
 26  Casualty_severity        12316 non-null  object
 27  Work_of_casuality        9118 non-null   object
 28  Fitness_of_casuality     9681 non-null   object
 29  Pedestrian_movement      12316 non-null  object
 30  Cause_of_accident        12316 non-null  object
 31  Accident_severity        12316 non-null  object
dtypes: int64(2), object(30)
memory usage: 3.0+ MB
```

In [12]:

```
# convert the 'Date' column to datetime format
df['Time']= pd.to_datetime(df['Time'])
```

In [13]:

```
df.duplicated()
```

Out[13]:

```
0        False
1        False
2        False
3        False
4        False
         ...
12311    False
12312    False
12313    False
12314    False
12315    False
Length: 12316, dtype: bool
```

In [14]:

```
df.duplicated().sum()
```

Out[14]:

```
0
```

In [15]:

```
df.groupby('Accident_severity').size()
```

Out[15]:

```
Accident_severity
Fatal injury        158
Serious Injury     1743
Slight Injury     10415
dtype: int64
```

Data Preprocessing

In [16]:

```
df.isnull().sum()
```

Out[16]:

```
Time                          0
Day_of_week                   0
Age_band_of_driver            0
Sex_of_driver                 0
Educational_level           741
Vehicle_driver_relation     579
Driving_experience          829
Type_of_vehicle             950
Owner_of_vehicle            482
Service_year_of_vehicle    3928
Defect_of_vehicle          4427
Area_accident_occured       239
Lanes_or_Medians            385
Road_allignment             142
Types_of_Junction           887
Road_surface_type           172
Road_surface_conditions       0
Light_conditions              0
Weather_conditions            0
Type_of_collision           155
Number_of_vehicles_involved   0
Number_of_casualties          0
Vehicle_movement            308
Casualty_class                0
Sex_of_casualty               0
Age_band_of_casualty          0
Casualty_severity             0
Work_of_casuality          3198
Fitness_of_casuality       2635
Pedestrian_movement           0
Cause_of_accident             0
Accident_severity             0
dtype: int64
```

In [ ]:

```

```

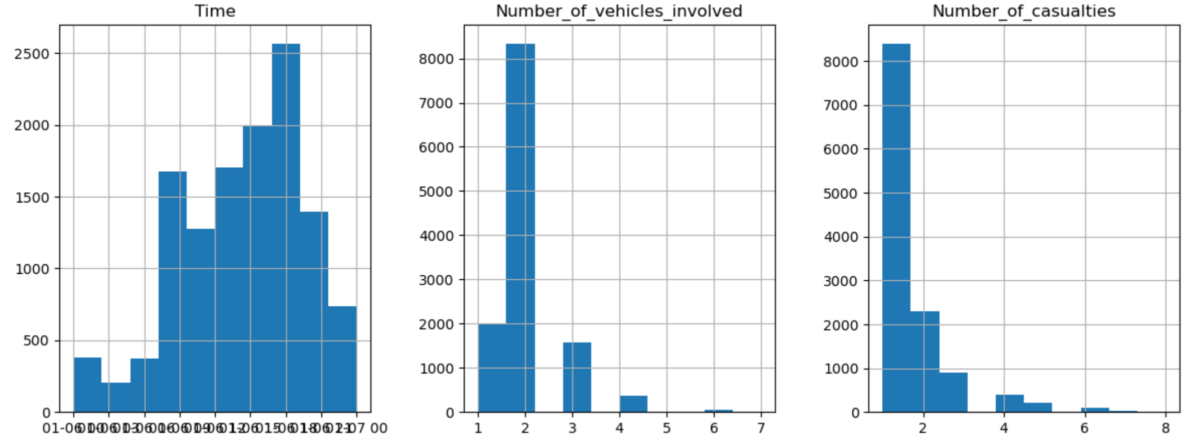
We can summarize the table as:

Number of observations: 12316 Number of columns: 32 Memory Usage: 3.0+ MB Number of int columns: 2 Number of object columns: 30 Number of columns with missing values: 16

Numerical data analysis

In [17]:

```
df.hist(layout=(1,6), figsize=(30,5))
plt.show()
```



In [18]:

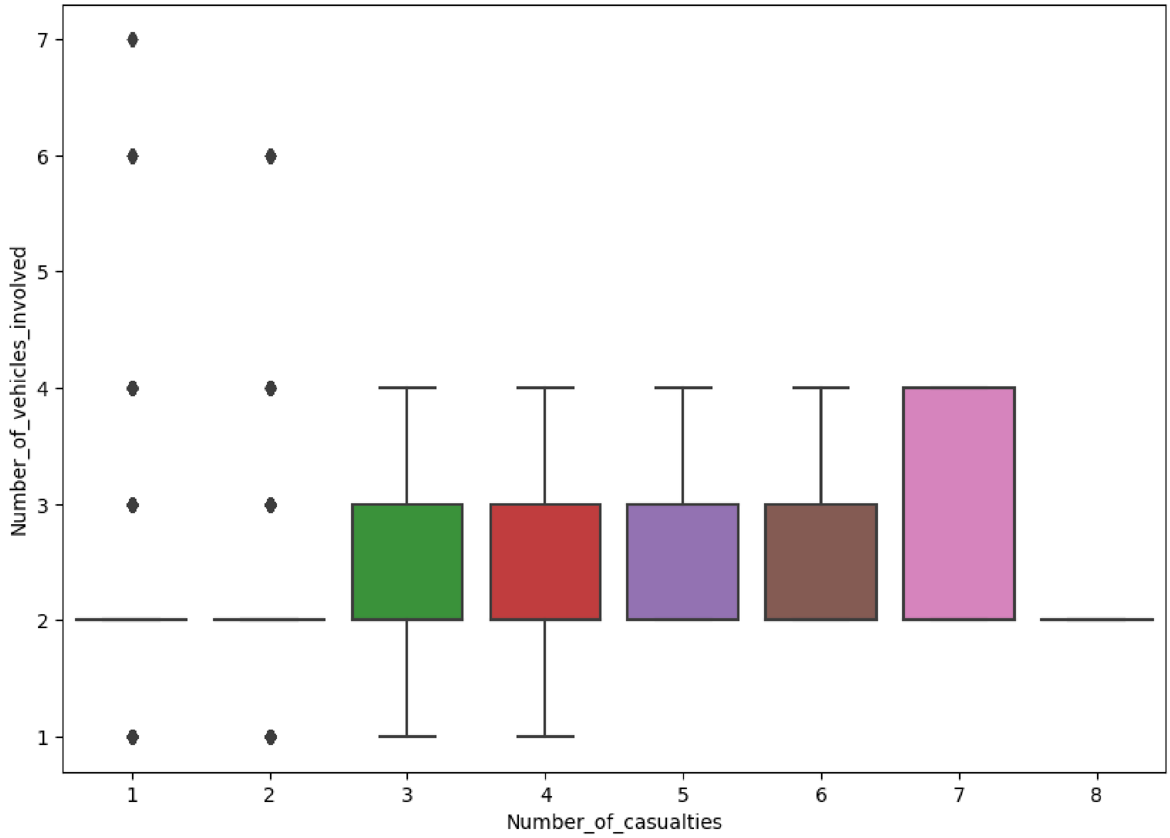
```
df['Number_of_casualties'].value_counts()
```

Out[18]:

```
1    8397
2    2290
3     909
4     394
5     207
6      89
7      22
8       8
Name: Number_of_casualties, dtype: int64
```

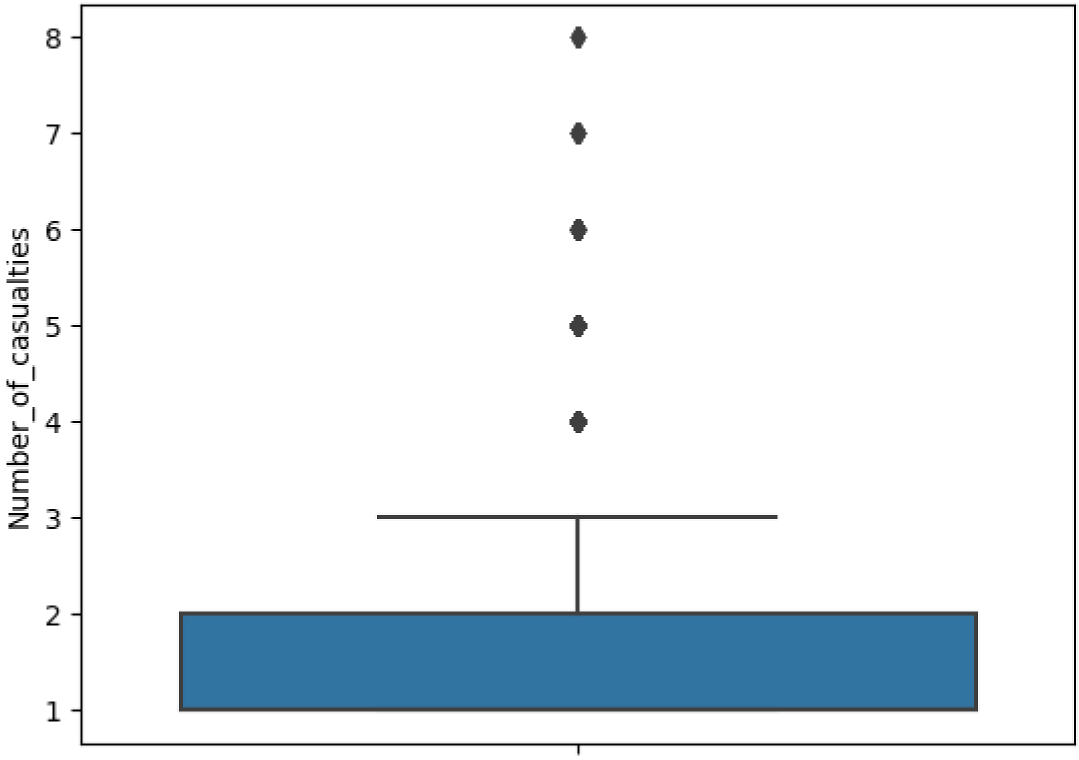
In [19]:

```
plt.figure(figsize=(10,7))
sns.boxplot(data=df, y='Number_of_vehicles_involved', x='Number_of_casualties')
plt.show()
```



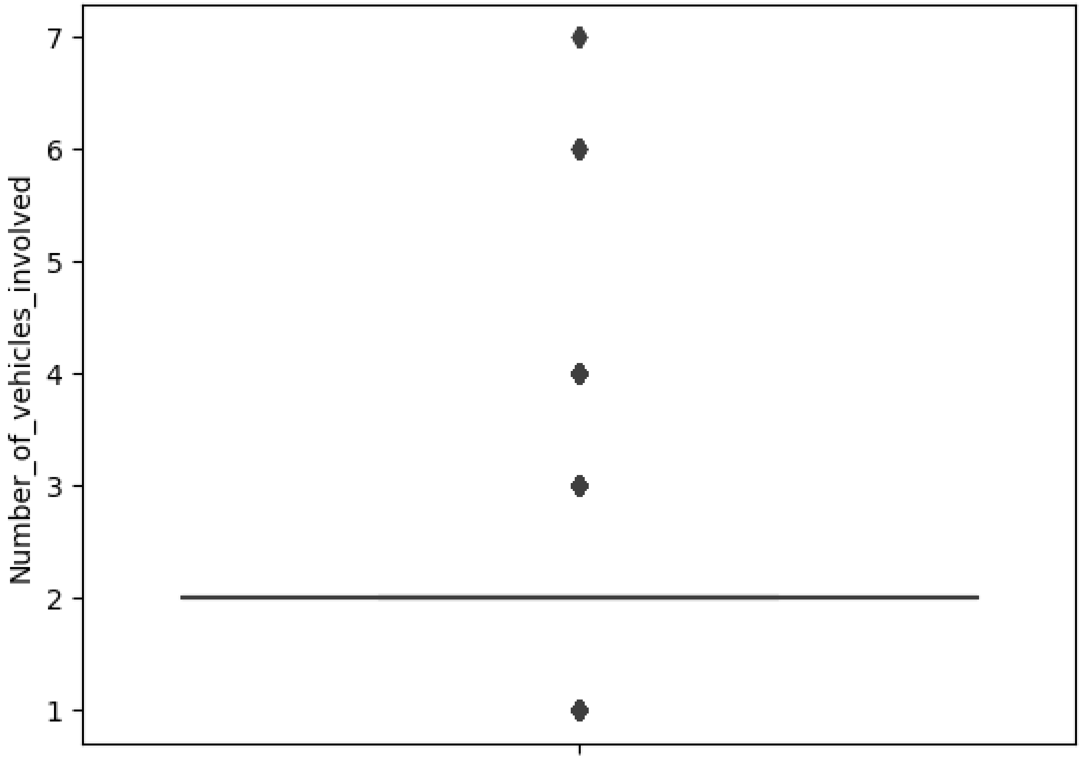
In [20]:

```
sns.boxplot(data=df, y='Number_of_casualties')
plt.show()
```



In [21]:

```
sns.boxplot(data=df, y='Number_of_vehicles_involved')
plt.show()
```



In [22]:

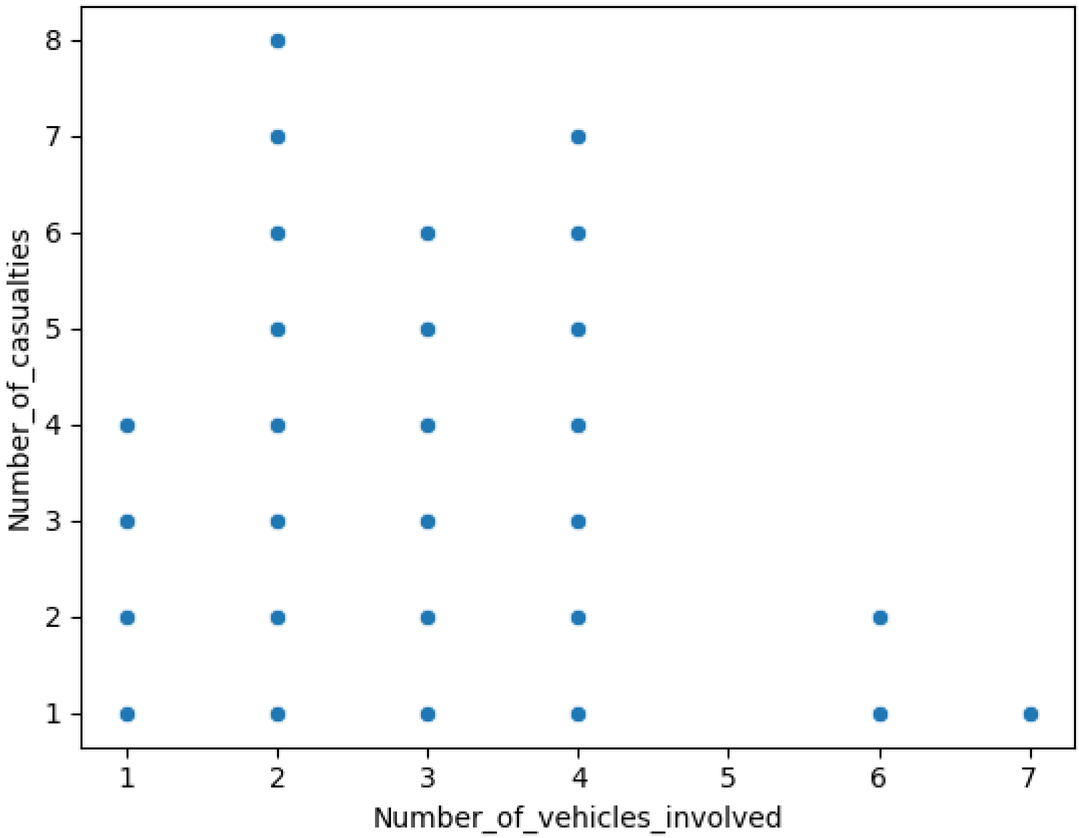
```
df['Number_of_vehicles_involved']
```

Out[22]:

```
0        2
1        2
2        2
3        2
4        2
        ..
12311    2
12312    2
12313    1
12314    2
12315    2
Name: Number_of_vehicles_involved, Length: 12316, dtype: int64
```

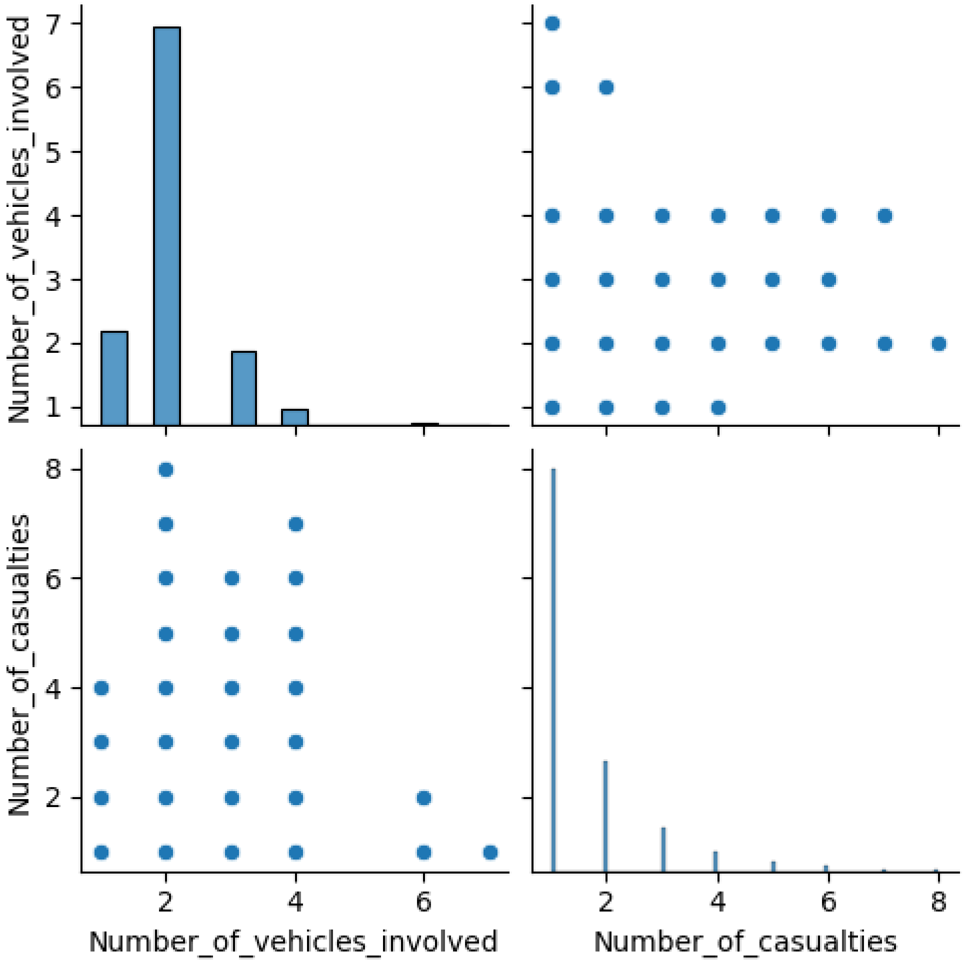
In [23]:

```
sns.scatterplot(x=df['Number_of_vehicles_involved'], y=df['Number_of_casualties'])
plt.show()
```



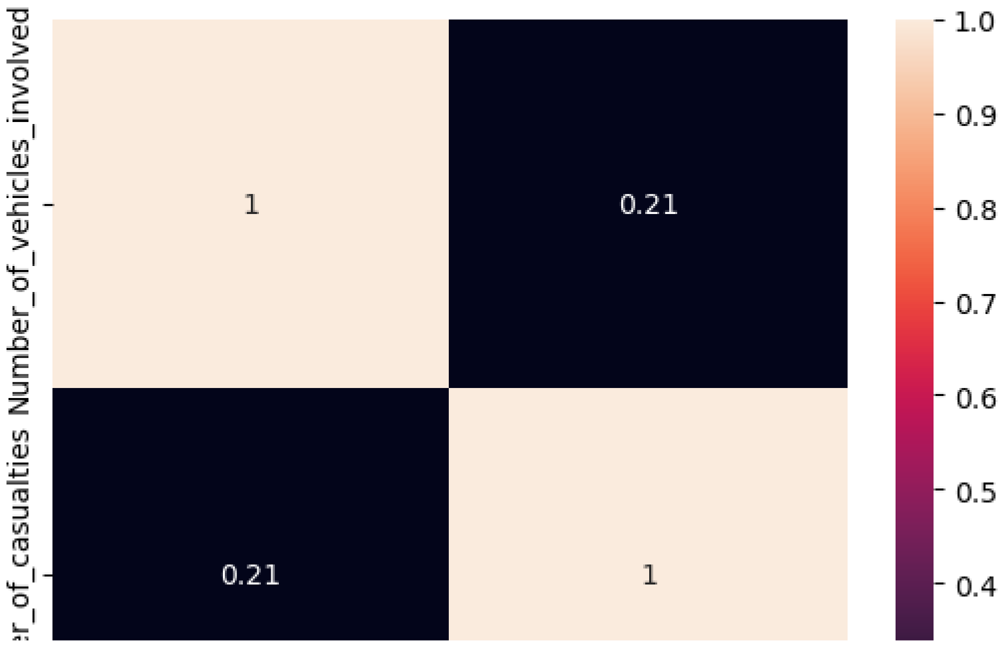
In [24]:

```
sns.pairplot(df[['Number_of_vehicles_involved','Number_of_casualties']])
plt.show()
```



In [25]:

```
correlation_matrix = df[['Number_of_vehicles_involved','Number_of_casualties']].corr()
sns.heatmap(correlation_matrix, annot=True)
plt.show()
```

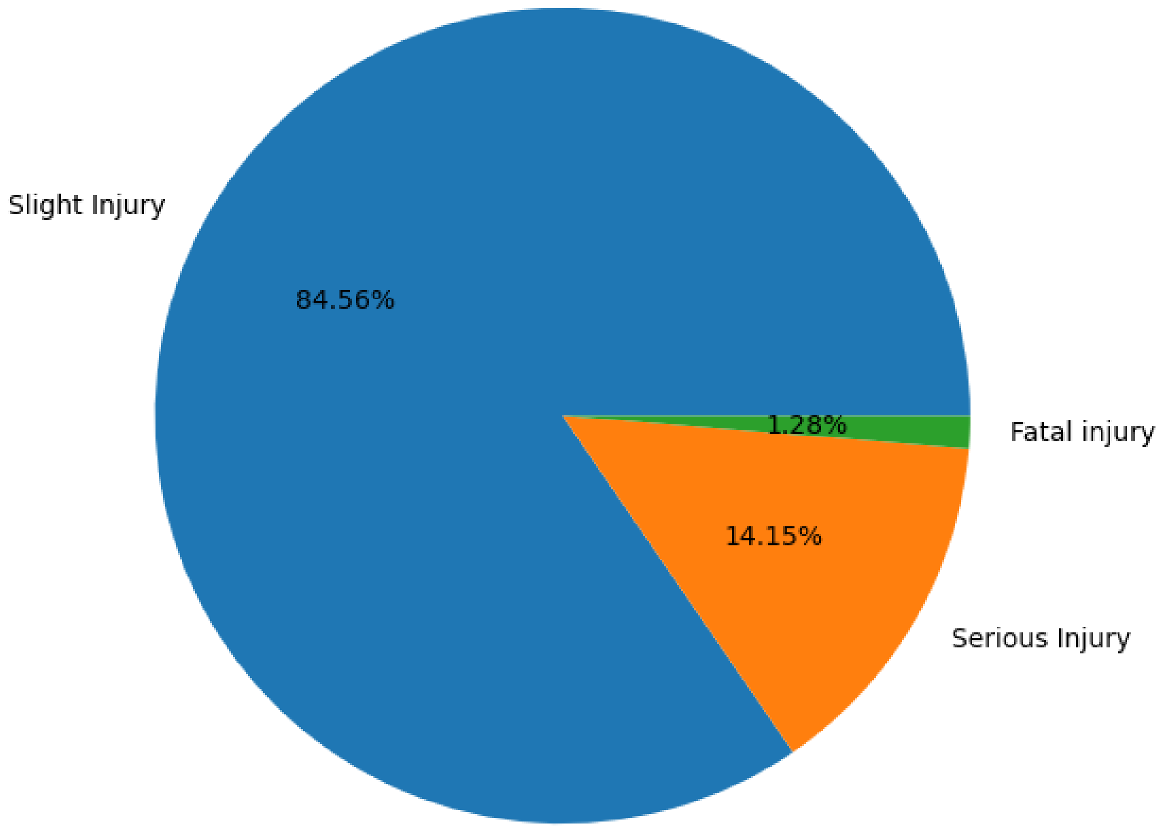


In this heatmap, we can find that these are not much extremely correlated variables

Categorical data analysis

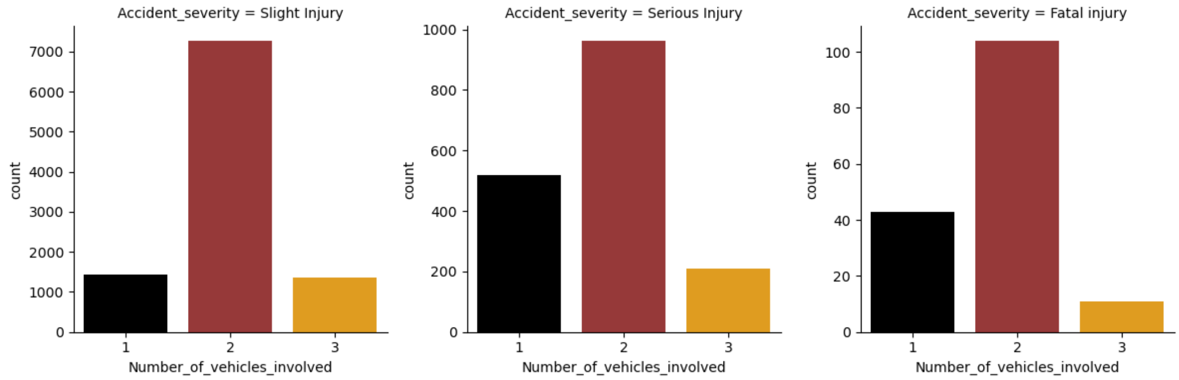
In [26]:

```
plt.figure(figsize=(10,7))
plt.pie(x=df['Accident_severity'].value_counts().values,
        labels=df['Accident_severity'].value_counts().index,
        autopct='%2.2f%%')
plt.show()
```



In [27]:

```
# creating a facet grid with columns as survived=0 and survived=1
grid = sns.FacetGrid(data=df, col='Accident_severity', height=4, aspect=1, sharey=False)
# mapping bar plot and the data on to the grid
grid.map(sns.countplot, 'Number_of_vehicles_involved', palette=['black', 'brown', 'orange']
plt.show()
```



In [28]:

```
df.columns
```

Out[28]:

```
Index(['Time', 'Day_of_week', 'Age_band_of_driver', 'Sex_of_driver',
       'Educational_level', 'Vehicle_driver_relation', 'Driving_experience',
       'Type_of_vehicle', 'Owner_of_vehicle', 'Service_year_of_vehicle',
       'Defect_of_vehicle', 'Area_accident_occured', 'Lanes_or_Medians',
       'Road_allignment', 'Types_of_Junction', 'Road_surface_type',
       'Road_surface_conditions', 'Light_conditions', 'Weather_conditions',
       'Type_of_collision', 'Number_of_vehicles_involved',
       'Number_of_casualties', 'Vehicle_movement', 'Casualty_class',
       'Sex_of_casualty', 'Age_band_of_casualty', 'Casualty_severity',
       'Work_of_casuality', 'Fitness_of_casuality', 'Pedestrian_movement',
       'Cause_of_accident', 'Accident_severity'],
      dtype='object')
```

In [29]:

```
# dropping columns that can cause imbalance while imputation
lists=['Vehicle_driver_relation', 'Work_of_casuality', 'Fitness_of_casuality','Day_of_week'
df.drop(columns = lists, inplace=True)
```

In [30]:

```
df.shape
```

Out[30]:

```
(12316, 19)
```

In [31]:

```
df.columns
```

Out[31]:

```
Index(['Age_band_of_driver', 'Driving_experience', 'Type_of_vehicle',
       'Area_accident_occured', 'Lanes_or_Medians', 'Road_allignment',
       'Types_of_Junction', 'Road_surface_conditions', 'Light_conditions',
       'Weather_conditions', 'Type_of_collision',
       'Number_of_vehicles_involved', 'Number_of_casualties',
       'Vehicle_movement', 'Casualty_class', 'Age_band_of_casualty',
       'Pedestrian_movement', 'Cause_of_accident', 'Accident_severity'],
      dtype='object')
```

Filling missing values

In [32]:

```
# fill missing values with mean column values
df['Driving_experience'].fillna(df['Driving_experience'].mode()[0], inplace=True)
df['Age_band_of_driver'].fillna(df['Age_band_of_driver'].mode()[0], inplace=True)
df['Type_of_vehicle'].fillna(df['Type_of_vehicle'].mode()[0], inplace=True)
df['Area_accident_occured'].fillna(df['Area_accident_occured'].mode()[0], inplace=True)
df['Road_allignment'].fillna(df['Road_allignment'].mode()[0], inplace=True)
df['Type_of_collision'].fillna(df['Type_of_collision'].mode()[0], inplace=True)
df['Vehicle_movement'].fillna(df['Vehicle_movement'].mode()[0], inplace=True)
df['Lanes_or_Medians'].fillna(df['Lanes_or_Medians'].mode()[0], inplace=True)
df['Types_of_Junction'].fillna(df['Types_of_Junction'].mode()[0], inplace=True)
```

In [33]:

```
df.isnull().sum()
```

Out[33]:

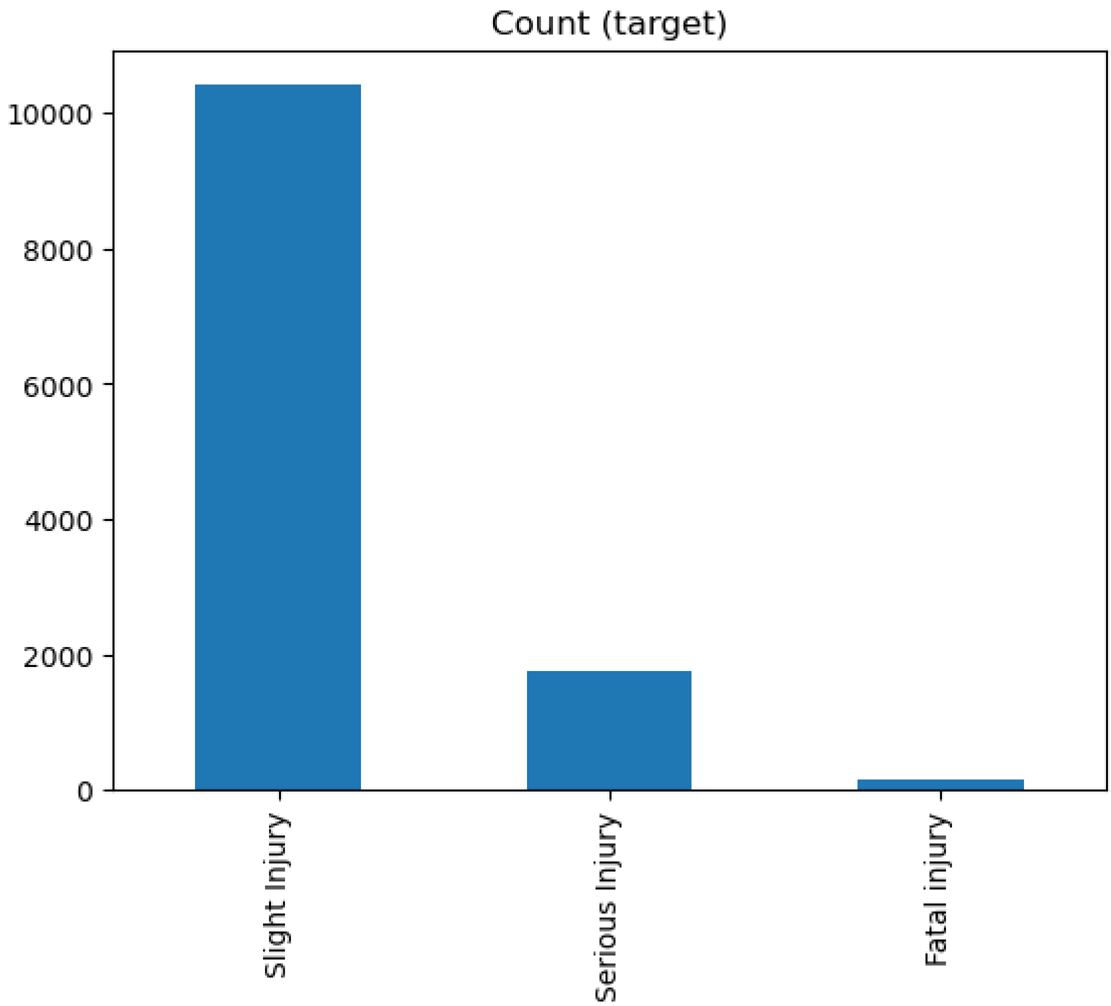
```
Age_band_of_driver             0
Driving_experience             0
Type_of_vehicle                0
Area_accident_occured          0
Lanes_or_Medians               0
Road_allignment                0
Types_of_Junction              0
Road_surface_conditions        0
Light_conditions               0
Weather_conditions             0
Type_of_collision              0
Number_of_vehicles_involved    0
Number_of_casualties           0
Vehicle_movement               0
Casualty_class                 0
Age_band_of_casualty           0
Pedestrian_movement            0
Cause_of_accident              0
Accident_severity              0
dtype: int64
```

In [34]:

```
target_count = df['Accident_severity'].value_counts()
print('Class 0:', target_count[0])
print('Class 1:', target_count[1])
print('Proportion:', round(target_count[0] / target_count[1], 2), ': 1')

target_count.plot(kind='bar', title='Count (target)');
```

```
Class 0: 10415
Class 1: 1743
Proportion: 5.98 : 1
```



Encoding

In [35]:

```
from sklearn.preprocessing import LabelEncoder          #or one hot encoder
LE = LabelEncoder()
df=df.apply(LE.fit_transform)                           #categorical values to integers
```

In [36]:

```
plt.figure(figsize=[25,15])
sns.heatmap(df.corr(),annot=True)
```

Out[36]:

```
<AxesSubplot:>
```

In [37]:

```
for col in df.select_dtypes(include='object'):
    if df[col].nunique() <= 22:
        sns.countplot(y=col, data=df)
        plt.show()
```

Upsampling

In [38]:

```
x = df.drop('Accident_severity', axis=1)
y = df['Accident_severity']

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.3, random_state=42)
print(xtrain.shape, xtest.shape, ytrain.shape, ytest.shape)
```

```
(8621, 18) (3695, 18) (8621,) (3695,)
```

In [39]:

```
# upsampling using smote

counter = Counter(ytrain)

print("=============================")

for k,v in counter.items():
    per = 100*v/len(ytrain)
    print(f"Class= {k}, n={v} ({per:.2f}%)")

oversample = SMOTE()
xtrain, ytrain = oversample.fit_resample(xtrain, ytrain)

counter = Counter(ytrain)

print("=============================")

for k,v in counter.items():
    per = 100*v/len(ytrain)
    print(f"Class= {k}, n={v} ({per:.2f}%)")

print("=============================")

print("Upsampled data shape: ", xtrain.shape, ytrain.shape)
```

```
=============================
Class= 2, n=7324 (84.96%)
Class= 1, n=1191 (13.82%)
Class= 0, n=106 (1.23%)
=============================
Class= 2, n=7324 (33.33%)
Class= 1, n=7324 (33.33%)
Class= 0, n=7324 (33.33%)
=============================
Upsampled data shape:  (21972, 18) (21972,)
```

Splitting test and train data

In [40]:

```
x=df.drop(columns=["Accident_severity"])
y=df["Accident_severity"]
```

In [41]:

```
models={"LogisticRegression":LogisticRegression(),
        "DecisionTreeClassifier":DecisionTreeClassifier(),
        "SVM":SVC(),
        "KNeighborsClassifier":KNeighborsClassifier(),
        "GNB":GaussianNB(),
       "RandomForestClassifier":RandomForestClassifier(),
        "AdaBoostClassifier":AdaBoostClassifier(),
        "GradientBoostingClassifier":GradientBoostingClassifier(),
        }
```

In [42]:

```
# models,x,y,scaleFlag=0,1,2
def modelAccuracy(models,x,y,scaleFlag):
    #train/Test
    xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=0)
    acc_result={}
    for name,model in models.items():
        #pipeline
        #1.Transformer -> 2.Model
        if(scaleFlag==1):
            model_pipeline=Pipeline([('MinMaxScler',MinMaxScaler()),('model',model)])
        elif(scaleFlag==2):
             model_pipeline=Pipeline([('StandardScaler',StandardScaler()),('model',model)])
        else:
            model_pipeline=Pipeline([('model',model)])
        #training/testing on model pipeline
        model_fit=model_pipeline.fit(xtrain,ytrain)
        ypred=model_fit.predict(xtest)
        acc=accuracy_score(ytest,ypred)
        print("The Accuracy for ",name," is :",acc)
        acc_result[name]=acc
    return acc_result
```

In [43]:

```
def bestModel(result):
    high=0
    for name,acc in result.items():
        if acc>high:
            high=acc
            model_name=name
    print("Best Model is ",model_name," with accuaracy =>",high)
```

In [44]:

```
def bestParams(model,param,xtrain,ytrain):
    #cv
    cv=RepeatedStratifiedKFold(n_splits=5,n_repeats=3)
    grid_cv=GridSearchCV(estimator=model,param_grid=param,cv=cv,scoring="f1_weighted")
    res=grid_cv.fit(xtrain,ytrain)
    print("Best Parameters are ",res.best_params_)
    print("Best Accuracy is ",res.best_score_)
```

In [45]:

```
bestParams
```

Out[45]:

```
<function __main__.bestParams(model, param, xtrain, ytrain)>
```

In [46]:

```
acc=modelAccuracy(models,x,y,1)
```

```
The Accuracy for  LogisticRegression  is : 0.84375
The Accuracy for  DecisionTreeClassifier  is : 0.7329545454545454
The Accuracy for  SVM  is : 0.84375
The Accuracy for  KNeighborsClassifier  is : 0.8262987012987013
The Accuracy for  GNB  is : 0.8145292207792207
The Accuracy for  RandomForestClassifier  is : 0.8482142857142857
The Accuracy for  AdaBoostClassifier  is : 0.8425324675324676
The Accuracy for  GradientBoostingClassifier  is : 0.8486201298701299
```

In [47]:

```
bestModel(acc)
```

```
Best Model is  GradientBoostingClassifier  with accuaracy => 0.8486201298701
299
```

In [48]:

```
model=RandomForestClassifier()
params={"n_estimators" : [100,200],
        "criterion" : ["gini","entropy"]
        }
bestParams(model,params,xtrain,ytrain)
```

```
Best Parameters are  {'criterion': 'entropy', 'n_estimators': 200}
Best Accuracy is  0.9194693388169922
```

In [49]:

```
#retrain the model with best parameters
model=RandomForestClassifier(criterion="entropy",n_estimators=200)
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [50]:

```
#Final Evaluation
print(accuracy_score(ytest,ypred))
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

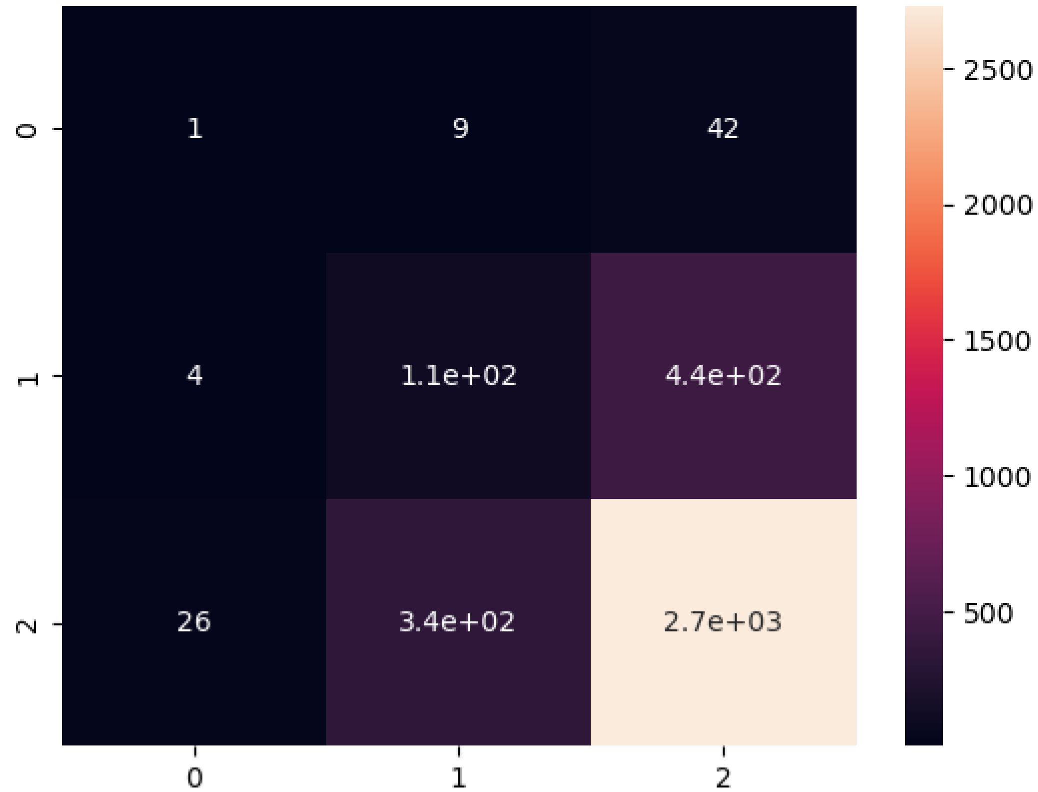
```
0.7691474966170501
              precision    recall  f1-score   support

           0       0.03      0.02      0.02        52
           1       0.24      0.20      0.22       552
           2       0.85      0.88      0.87      3091

    accuracy                           0.77      3695
   macro avg       0.38      0.37      0.37      3695
weighted avg       0.75      0.77      0.76      3695
```

Out[50]:

```
<AxesSubplot:>
```



In [ ]:

```

```

In [ ]:

```

```