Assignment 2 (Machine Learning using Sklearn and TensorFlow)

Angel Team:

- Andres Salguero - C0932873
- Andrea Franco - C0931897
- Vishv Patel - C0938107
- Rajkumar Patel - C0934637
- Harpreet kaur - C0936410
- Gurpreet kaur - C0936411

Project: Traffic collision analysis

This project aims to build a machine learning classification model in oder to predict whether the person suffered a fatal or non fatal injury in a collision based on various features.

The classification models which are used in this project are evaluated on the basis of classification report and accuracy score to know which model is performing the best.

The classification models which will be used are Random Forest classifier, Gradient boosting Classifier, SVM amd logistic regression. Then the results from all these model are saved in csv file and then uploaded to competition to know whether the models are under-fitting or over-fitting.

Dataset

Killed or Seriously Injured (KSI) dataset

This dataset includes all traffic collisions events where a person was either Killed or Seriously Injured (KSI) from 2006 – 2022.

This Killed or Seriously Injured (KSI) dataset is a subset from all traffic collision events.

The source of the data comes from police reports where an officer attended an event related to a traffic collision. Please note that this dataset does not include all traffic collision events. The KSI data only includes events where a person sustained a major or fatal injury in a traffic collision event.

Data Fields Description:

- INDEX_ : Unique Identifier
- ACCNUM : Accident Number
- YEAR : Year Collision Occurred

- DATE : Date Collision Occurred (time is displayed in UTC format)
- TIME : Time Collision Occurred
- STREET1 : Street Collision Occurred
- STREET2 : Street Collision Occurred
- OFFSET : Distance and direction of the Collision
- ROAD_CLASS : Road Classification
- DISTRICT : City District
- WARDNUM : City of Toronto Ward collision occurred
- LATITUDE : Latitude
- LONGITUDE : Longitude
- LOCCOORD : Location Coordinate
- ACCLOC : Collision Location
- TRAFFCTL : Traffic Control Type
- VISIBILITY : Environment Condition
- LIGHT : Light Condition
- RDSFCOND : Road Surface Condition
- ACCLASS : Classification of Accident
- IMPACTYPE : Initial Impact Type
- INVTYPE : Involvement Type
- INVAGE : Age of Involved Party
- INJURY : Severity of Injury
- FATAL_NO : Sequential Number
- INITDIR : Initial Direction of Travel
- VEHTYPE : Type of Vehicle
- MANOEUVER : Vehicle Manoeuver
- DRIVACT : Apparent Driver Action
- DRIVCOND : Driver Condition
- PEDTYPE : Pedestrian Crash Type - detail
- PEDACT : Pedestrian Action
- PEDCOND : Condition of Pedestrian
- CYCLISTYPE : Cyclist Crash Type - detail
- CYCACT : Cyclist Action
- CYCCOND : Cyclist Condition
- PEDESTRIAN : Pedestrian Involved In Collision
- CYCLIST : Cyclists Involved in Collision
- AUTOMOBILE : Driver Involved in Collision
- MOTORCYCLE : Motorcyclist Involved in Collision
- TRUCK : Truck Driver Involved in Collision
- TRSN_CITY_VEH : Transit or City Vehicle Involved in Collision
- EMERG_VEH : Emergency Vehicle Involved in Collision
- PASSENGER : Passenger Involved in Collision
- SPEEDING : Speeding Related Collision
- AG_DRIV : Aggressive and Distracted Driving Collision

- REDLIGHT : Red Light Related Collision
- ALCOHOL : Alcohol Related Collision
- DISABILITY : Medical or Physical Disability Related Collision
- HOOD_158 Unique ID : for City of Toronto Neighbourhood (new)
- NEIGHBOURHOOD_158 : City of Toronto Neighbourhood name (new)
- HOOD_140 : Unique ID for City of Toronto Neighbourhood (old)
- NEIGHBOURHOOD_140 : City of Toronto Neighbourhood name (old)
- DIVISION : Toronto Police Service Division
- ObjectID : Unique Identifier (auto generated)

Credits to: Toronto Police Service Public Safety Data Portal

Objective: Build a Binary classification model based on certain features would predict if the incident would result in fatality or not.

Libraries:

In this code cell will be importing the libraries which are necessary for the project.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# import machine learning libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier ,
GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import
accuracy_score,roc_curve,auc,classification_report,confusion_matrix
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import learning_curve

# Tensorflow libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
```

This dataset is loaded using pandas function (pd.read_csv)

```
# loading dataset
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
```

The copy of test dataset is made using (copy) function in oder to use the object id make comparison and check the model's performance, over-fitting and under-fitting.

```
# creating copy for accessing object id from this
object_id_col = test_df.copy()
object_id_col.head()
```

```
             X            Y  OBJECTID    INDEX_  ACCNUM  \
0  637398.2785  4849101.813     15001  80972086     NaN
1  637398.2785  4849101.813     15002  80972617     NaN
2  639017.8028  4843417.954     15003  80972182     NaN
3  639017.8028  4843417.954     15004  80972183     NaN
4  620810.2466  4838690.153     15005  80972485     NaN

                     DATE  TIME              STREET1       STREET2  \
0  2018/09/26 08:00:00+00  2053  3850 SHEPPARD AVE E           NaN
1  2018/09/26 08:00:00+00  2053  3850 SHEPPARD AVE E           NaN
2  2018/09/28 08:00:00+00   806       EGLINTON AVE E  ROSEMOUNT DR
3  2018/09/28 08:00:00+00   806       EGLINTON AVE E  ROSEMOUNT DR
4  2018/09/28 08:00:00+00  1018         1277 JANE ST           NaN

         OFFSET  ... SPEEDING AG_DRIV  REDLIGHT  ALCOHOL DISABILITY
HOOD_158  \
0  90 m East of  ...      NaN     Yes       NaN      NaN        NaN
118
1  90 m East of  ...      NaN     Yes       NaN      NaN        NaN
118
2           NaN  ...      NaN     NaN       NaN      NaN        NaN
125
3           NaN  ...      NaN     NaN       NaN      NaN        NaN
125
4  4 m North of  ...      NaN     NaN       NaN      NaN        NaN
115

        NEIGHBOURHOOD_158 HOOD_140             NEIGHBOURHOOD_140
DIVISION
0  Tam O'Shanter-Sullivan      118  Tam O'Shanter-Sullivan (118)
D42
1  Tam O'Shanter-Sullivan      118  Tam O'Shanter-Sullivan (118)
D42
2                 Ionview      125                 Ionview (125)
D41
3                 Ionview      125                 Ionview (125)
D41
4           Mount Dennis      115            Mount Dennis (115)
D12

[5 rows x 53 columns]
```

Exploratory data analysis

In this step we will be using some function like head, info to get insights of our datasets.

```
train_df.head()

              X            Y  OBJECTID   INDEX_    ACCNUM  \
0  635468.3685  4839880.764         1  3389067  893184.0
1  635468.3685  4839880.764         2  3389068  893184.0
2  635468.3685  4839880.764         3  3389069  893184.0
3  635468.3685  4839880.764         4  3389070  893184.0
4  635468.3685  4839880.764         5  3389071  893184.0

                     DATE  TIME       STREET1      STREET2 OFFSET  ...
\
0  2006/01/01 10:00:00+00   236  WOODBINE AVE  O CONNOR DR    NaN  ...

1  2006/01/01 10:00:00+00   236  WOODBINE AVE  O CONNOR DR    NaN  ...

2  2006/01/01 10:00:00+00   236  WOODBINE AVE  O CONNOR DR    NaN  ...

3  2006/01/01 10:00:00+00   236  WOODBINE AVE  O CONNOR DR    NaN  ...

4  2006/01/01 10:00:00+00   236  WOODBINE AVE  O CONNOR DR    NaN  ...


  SPEEDING AG_DRIV  REDLIGHT  ALCOHOL DISABILITY HOOD_158
NEIGHBOURHOOD_158  \
0      Yes     Yes       NaN      Yes        NaN       60  Woodbine-
Lumsden
1      Yes     Yes       NaN      Yes        NaN       60  Woodbine-
Lumsden
2      Yes     Yes       NaN      Yes        NaN       60  Woodbine-
Lumsden
3      Yes     Yes       NaN      Yes        NaN       60  Woodbine-
Lumsden
4      Yes     Yes       NaN      Yes        NaN       60  Woodbine-
Lumsden

  HOOD_140      NEIGHBOURHOOD_140 DIVISION
0       60  Woodbine-Lumsden (60)      D55
1       60  Woodbine-Lumsden (60)      D55
2       60  Woodbine-Lumsden (60)      D55
3       60  Woodbine-Lumsden (60)      D55
4       60  Woodbine-Lumsden (60)      D55

[5 rows x 54 columns]

# number of raws and columns
train_df.shape
```

```
(15000, 54)
```

```python
# information of dataset such as how many non-null values , datatypes of columns, number of rows and columns
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 54 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   X                  15000 non-null  float64
 1   Y                  15000 non-null  float64
 2   OBJECTID           15000 non-null  int64
 3   INDEX_             15000 non-null  int64
 4   ACCNUM             11302 non-null  float64
 5   DATE               15000 non-null  object
 6   TIME               15000 non-null  int64
 7   STREET1            15000 non-null  object
 8   STREET2            13657 non-null  object
 9   OFFSET             1928 non-null   object
 10  ROAD_CLASS         14643 non-null  object
 11  DISTRICT           14984 non-null  object
 12  LATITUDE           15000 non-null  float64
 13  LONGITUDE          15000 non-null  float64
 14  ACCLOC             9550 non-null   object
 15  TRAFFCTL           14971 non-null  object
 16  VISIBILITY         14986 non-null  object
 17  LIGHT              15000 non-null  object
 18  RDSFCOND           14981 non-null  object
 19  ACCLASS            15000 non-null  object
 20  IMPACTYPE          15000 non-null  object
 21  INVTYPE            14990 non-null  object
 22  INVAGE             15000 non-null  object
 23  INJURY             7811 non-null   object
 24  FATAL_NO           593 non-null    float64
 25  INITDIR            10502 non-null  object
 26  VEHTYPE            12944 non-null  object
 27  MANOEUVER          8486 non-null   object
 28  DRIVACT            7425 non-null   object
 29  DRIVCOND           7421 non-null   object
 30  PEDTYPE            2460 non-null   object
 31  PEDACT             2450 non-null   object
 32  PEDCOND            2445 non-null   object
 33  CYCLISTYPE         635 non-null    object
 34  CYCACT             621 non-null    object
 35  CYCCOND            620 non-null    object
 36  PEDESTRIAN         5966 non-null   object
 37  CYCLIST            1578 non-null   object
 38  AUTOMOBILE         13672 non-null  object
```

```
 39  MOTORCYCLE          1162 non-null   object
 40  TRUCK               933 non-null    object
 41  TRSN_CITY_VEH       923 non-null    object
 42  EMERG_VEH           19 non-null     object
 43  PASSENGER           5633 non-null   object
 44  SPEEDING            1998 non-null   object
 45  AG_DRIV             7696 non-null   object
 46  REDLIGHT            1275 non-null   object
 47  ALCOHOL             672 non-null    object
 48  DISABILITY          420 non-null    object
 49  HOOD_158            15000 non-null  object
 50  NEIGHBOURHOOD_158   15000 non-null  object
 51  HOOD_140            15000 non-null  object
 52  NEIGHBOURHOOD_140   15000 non-null  object
 53  DIVISION            15000 non-null  object
dtypes: float64(6), int64(3), object(45)
memory usage: 6.2+ MB
```

Unique values

In this we are making a function that will tell us about all the unique values in our datset and printing it

```
#created function for printing unique value from all column
def count_func():
    count= 0
    for column in train_df.columns:# loop which call each column one by one
        unique_value= train_df[column].unique() # getting unique value in each column
        unique_number= train_df[column].nunique() # getting number of unique value in each column
        count+=1
        print(f"{count}. unique value number in {column} : {unique_number} \n") # print the number of unique values
        print(f"{count}. unique value in {column}: {unique_value} \n") # print the unique value
        print("-------" * 10)

count_func()

1. unique value number in X : 4695

1. unique value in X: [635468.3685 635711.8004 628520.911  ... 641202.6999 627158.8849
 638360.8419]

----------------------------------------------------------------------
2. unique value number in Y : 4695
```

```
2. unique value in Y: [4839880.764 4838250.056 4834554.582 ...
4842218.457 4836916.84
 4852316.818]

----------------------------------------------------------------------
3. unique value number in OBJECTID : 15000

3. unique value in OBJECTID: [    1     2     3 ... 14998 14999 15000]


----------------------------------------------------------------------
4. unique value number in INDEX_ : 15000

4. unique value in INDEX_: [ 3389067  3389068  3389069 ... 80972829
80972190 80972191]

----------------------------------------------------------------------
5. unique value number in ACCNUM : 3822

5. unique value in ACCNUM: [8.93184000e+05 9.09646000e+05
8.84090000e+05 ... 4.00356472e+09
 1.78057130e+08 1.88016123e+08]

----------------------------------------------------------------------
6. unique value number in DATE : 3082

6. unique value in DATE: ['2006/01/01 10:00:00+00' '2006/01/02
10:00:00+00'
 '2006/01/04 10:00:00+00' ... '2018/09/23 08:00:00+00'
 '2018/09/24 08:00:00+00' '2018/09/26 08:00:00+00']

----------------------------------------------------------------------
7. unique value number in TIME : 1276

7. unique value in TIME: [ 236  315  705 ... 1712  729  408]

----------------------------------------------------------------------
8. unique value number in STREET1 : 1547

8. unique value in STREET1: ['WOODBINE AVE' 'DANFORTH AVE' 'BATHURST
ST' ... '2265 MIDLAND AVE'
 'DEWHURST BLVD' 'MELITA AVE']

----------------------------------------------------------------------
9. unique value number in STREET2 : 2344

9. unique value in STREET2: ['O CONNOR DR' 'WEST LYNN AVE' 'DUNDAS ST
W' ... 'DOCTOR O LANE'
 'CHESTNUT ST' 'BRIMWOOD BLVD']

----------------------------------------------------------------------
```

```
10. unique value number in OFFSET : 335

10. unique value in OFFSET: [nan '60 NORTH OF' '1 m West of' '234 m
South ' '450 m West o'
 '7 m West of' '314 m  of' '192 m East o' '2 m North of' '100 m East
o'
 '51 m South o' '43 m West of' '5 m North of' '8 m North of' '6 m West
of'
 '30 m North o' '25 m East of' '60 m East of' '44 m North o'
 '132 m West o' '48 m East of' '30 m South o' '500 m East o'
 '1 m North of' '58 m North o' '500 m North ' '12 m West of'
 '80 m West of' '97 m South o' '100 m South ' '17 m West of'
 '280 m East o' '4 m North of' '10 m East of' '25 m West of'
 '18 m East of' '41 m South o' '50 m East of' '39 m East of'
 '10 m South o' '76 m West of' '50 m West of' '15 m East of'
 '220 m South ' '10 m North o' '4 m South of' '100 m North '
 '55 m North o' '25 m South o' '14 m West of' '200 m North '
 '65 m East of' '27 m East of' '64 m North o' '28 m East of'
 '85 m West of' '55 m West of' '200 m South ' '40 m West of'
 '58 m West of' '46 m North o' '4 m West of' '70 m West of' '15 m
South o'
 '246 m North ' '49 m East of' '37 m West of' '23 m East of'
 '60 m North o' '6 m North of' '3 m  of' '2 m West of' '20 m North o'
 '120 m West o' '50 m South o' '545 m North ' '5 m East of' '185 m
South '
 '98 m West of' '3 m West of' '6 m South of' '12 m South o' '31 m
North o'
 '20 m East of' '24 m South o' '125 m East o' '9 m West of' '20 m
South o'
 '4 m East of' '35 m North o' '17 m North o' '33 m West of' '374 m
West o'
 '80 m East of' '300 m East o' '38 m North o' '1 m East of' '5 m West
of'
 '11 m South o' '5 m South of' '100 m West o' '40 m East of'
 '22 m East of' '10 m West of' '26 m South o' '73 m West of'
 '167 m East o' '12 m East of' '15 m West of' '18 m South o'
 '40 m North o' '9 m East of' '18 m North o' '30 m East of' '20 m West
of'
 '40 m South o' '53 m South o' '24 m West of' '165 m South ' '24 m
of'
 '57 m East of' '3 m East of' '9 m South of' '3 m South of' '42 m
South o'
 '33 m East of' '358 m North ' '64 m West of' '64 m East of'
 '16 m West of' '3 m North of' '35 m East of' '51 m East of'
 '60 m West of' '14 m East of' '120 m North ' '14 m North o'
 '19 m South o' '90 m North o' '94 m North o' '19 m East of'
 '134 m West o' '7 m East of' '119 m West o' '7 m North of' '1 m South
of'
 '16 m East of' '196 m North ' '265 m East o' '13 m West of'
```

```
 '130 m South ' '70 m East of' '57 m North o' '90 m West of'
 '85 m South o' '245 m East o' '172 m East o' '16 m North o'
 '39 m South o' '13 m North o' '450 m East o' '81 m West of'
 '11 m West of' '67 m South o' '99 m South o' '18 m West of'
 '110 m South ' '2 m South of' '150 m East o' '420 m West o'
 '324 m East o' '63 m West of' '80 m South o' '51 m West of' '2 m East
of'
 '400 m North ' '90 m South o' '30 m West of' '37 m South o'
 '120 m South ' '58 m South o' '45 m West of' '11 m North o'
 '143 m West o' '92.1 m south' '32 m East of' '65 m South o'
 '200 m West o' '75 m East of' '65.6 M E of' '105 m North ' '15 m
North o'
 '8 m South of' '27 m West of' '137 m South ' '12.5 M S of' '130 m
East o'
 '121 m North ' '282 m South ' '40 m East' '219 m North ' '139 m South
'
 '66 m North o' '68 m North o' '368 m East o' '25M' '37 m East of'
 '50 m North o' '72 m North o' '55 m East of' '160 m West o'
 '23 m West of' '458 m West o' '500 m West o' '131 m South '
 '51 m North o' '76 m South o' '34 m East of' '8 m West of' '63 m
North o'
 '300 m South ' '69 m West of' '84 m South o' '900 m West o'
 '252 m South ' '113 m North ' '20 m North' '350 m West' '350 m North
'
 '250 m West o' '42 m East of' '177 m West o' '112 m North' '150 m
North '
 '8 m East of' '7 m South of' '6.5 m West o' '8.6 m East o' '386 m
South '
 '17 m East of' '176 m South ' '6 m East of' '34 m South o' '10 m
west'
 '150 m East' '50 m North' '47 m East of' '47 m West of' '29 m East
of'
 '700 m East o' 'north of' '80 m North o' '21 m East of' '26 m West
of'
 '12 m North o' '69 m North o' '35 m South o' '233 m East o'
 '9 m North of' '101 m South ' '65 m West of' '29 m West of'
 '98 m South o' '240 m North ' '107 m East o' '150 m West o' '20 m
of'
 '95 m South o' '30  m East o' '38 m East of' '120 m East o'
 '213 m West o' '71 m West of' '84 m West of' '297 m East o'
 '45 m South o' '88 m North o' '200 m East o' '21 m North o'
 '99 m East of' '84 m North o' '153 m North ' '22 m West of'
 '378 m South ' '60 m South o' '147 M North' '75 m East' '31 m East
of'
 '92 m East of' '41 m West of' '140 m East o' '403 m N of' '408 m East
o'
 'E of' 'W of' '195 m South' '74 m South o' '16 m South o' '99 m West
of'
 '11 m East of' '52 m East of' '28 m North o' '45 m  of' '192 m West
```

```
o'
 '185 m East o' '107 m North ' '45 m North o' '70 m South o'
 '400 m East o' '180 m East o' '77 m South o' '75 m North o'
 '23 m North o' '297 m South ' '73 m North o' '49 m West of'
 '44 m West of' '33 m North o' '87 m East of' '158 m South '
 '620 HURON ST' '31.8 m East ' '42 m South' '260 m West o' '600 m
East'
 '365 m East o' '13 m East of' '31 m West of' '81 m North o'
 '37 m North o' '59 m North o' '17 meters so' '34 meters so'
 '52 m West of' '338 m West o' '150 m South ']

----------------------------------------------------------------------
11. unique value number in ROAD_CLASS : 9

11. unique value in ROAD_CLASS: ['Major Arterial' 'Minor Arterial'
'Collector' 'Local' nan 'Other'
 'Pending' 'Laneway' 'Expressway' 'Expressway Ramp']

----------------------------------------------------------------------
12. unique value number in DISTRICT : 4

12. unique value in DISTRICT: ['Toronto and East York' 'North York'
'Scarborough' 'Etobicoke York' nan]

----------------------------------------------------------------------
13. unique value number in LATITUDE : 3475

13. unique value in LATITUDE: [43.699595 43.684874 43.652892 ...
43.719565 43.674388 43.810984]

----------------------------------------------------------------------
14. unique value number in LONGITUDE : 3901

14. unique value in LONGITUDE: [-79.318797 -79.316188 -79.406253 ... -
79.247051 -79.42258  -79.279712]

----------------------------------------------------------------------
15. unique value number in ACCLOC : 9

15. unique value in ACCLOC: ['Intersection Related' nan 'At
Intersection' 'Non Intersection'
 'Private Driveway' 'At/Near Private Drive' 'Underpass or Tunnel'
 'Overpass or Bridge' 'Trail' 'Laneway']

----------------------------------------------------------------------
16. unique value number in TRAFFCTL : 10

16. unique value in TRAFFCTL: ['No Control' 'Traffic Signal'
'Pedestrian Crossover' 'Stop Sign' nan
 'Yield Sign' 'Traffic Controller' 'School Guard' 'Police Control'
```

```
 'Traffic Gate' 'Streetcar (Stop for)']

----------------------------------------------------------------------
17. unique value number in VISIBILITY : 8

17. unique value in VISIBILITY: ['Clear' 'Snow' 'Other' 'Rain' 'Strong
wind' 'Fog, Mist, Smoke, Dust'
 'Drifting Snow' 'Freezing Rain' nan]

----------------------------------------------------------------------
18. unique value number in LIGHT : 9

18. unique value in LIGHT: ['Dark' 'Dark, artificial' 'Daylight'
'Dusk' 'Dawn' 'Dusk, artificial'
 'Dawn, artificial' 'Daylight, artificial' 'Other']

----------------------------------------------------------------------
19. unique value number in RDSFCOND : 9

19. unique value in RDSFCOND: ['Wet' 'Slush' 'Dry' 'Ice' 'Loose Snow'
'Other' 'Packed Snow'
 'Spilled liquid' 'Loose Sand or Gravel' nan]

----------------------------------------------------------------------
20. unique value number in ACCLASS : 2

20. unique value in ACCLASS: ['Non-Fatal Injury' 'Fatal']

----------------------------------------------------------------------
21. unique value number in IMPACTYPE : 10

21. unique value in IMPACTYPE: ['Approaching' 'SMV Other' 'Pedestrian
Collisions' 'Angle'
 'Turning Movement' 'Cyclist Collisions' 'Rear End' 'Sideswipe'
 'SMV Unattended Vehicle' 'Other']

----------------------------------------------------------------------
22. unique value number in INVTYPE : 18

22. unique value in INVTYPE: ['Passenger' 'Driver' 'Vehicle Owner'
'Other Property Owner' 'Pedestrian'
 'Cyclist' 'Other' 'Motorcycle Driver' 'Truck Driver' 'In-Line Skater'
 'Driver - Not Hit' 'Motorcycle Passenger' nan 'Moped Driver'
'Wheelchair'
 'Pedestrian - Not Hit' 'Trailer Owner' 'Witness' 'Cyclist Passenger']


----------------------------------------------------------------------
23. unique value number in INVAGE : 21

23. unique value in INVAGE: ['50 to 54' '15 to 19' '55 to 59' '20 to
```

```
24' 'unknown' '25 to 29'
 '10 to 14' '30 to 34' '45 to 49' '75 to 79' '35 to 39' '40 to 44'
 '80 to 84' '60 to 64' '85 to 89' '65 to 69' '70 to 74' '5 to 9' '0 to
4'
 '90 to 94' 'Over 95']

----------------------------------------------------------------------
24. unique value number in INJURY : 4

24. unique value in INJURY: ['Major' 'Minor' nan 'Fatal' 'Minimal']

----------------------------------------------------------------------
25. unique value number in FATAL_NO : 78

25. unique value in FATAL_NO: [nan  1.  2.  3.  4.  5.  6.  7.  8. 12.
10.  9. 11. 13. 14. 15. 16. 17.
 18. 19. 20. 21. 22. 26. 23. 24. 25. 27. 28. 29. 30. 31. 32. 33. 34.
35.
 36. 37. 38. 39. 40. 41. 42. 43. 44. 46. 45. 47. 48. 49. 50. 51. 52.
53.
 54. 55. 57. 56. 58. 59. 60. 61. 62. 63. 65. 64. 78. 66. 67. 68. 69.
70.
 71. 72. 73. 74. 75. 76. 77.]

----------------------------------------------------------------------
26. unique value number in INITDIR : 5

26. unique value in INITDIR: [nan 'North' 'South' 'East' 'West'
'Unknown']

----------------------------------------------------------------------
27. unique value number in VEHTYPE : 27

27. unique value in VEHTYPE: [nan 'Automobile, Station Wagon' 'Other'
'Passenger Van'
 'Municipal Transit Bus (TTC)' 'Taxi' 'Bicycle' 'Delivery Van'
 'Motorcycle' 'Truck - Open' 'Moped' 'Pick Up Truck' 'Tow Truck'
 'Police Vehicle' 'Truck-Tractor' 'Street Car'
 'Truck - Closed (Blazer, etc)' 'Truck - Dump'
 'Bus (Other) (Go Bus, Gray Coa' 'Construction Equipment' 'Intercity
Bus'
 'Truck (other)' 'Fire Vehicle' 'School Bus' 'Other Emergency Vehicle'
 'Off Road - 2 Wheels' 'Truck - Tank' 'Truck - Car Carrier']

----------------------------------------------------------------------
28. unique value number in MANOEUVER : 16

28. unique value in MANOEUVER: [nan 'Going Ahead' 'Changing Lanes'
'Turning Right' 'Slowing or Stopping'
 'Turning Left' 'Other' 'Stopped' 'Unknown' 'Parked' 'Overtaking'
```

```
 'Making U Turn' 'Reversing' 'Pulling Away from Shoulder or Curb'
 'Pulling Onto Shoulder or towardCurb' 'Merging' 'Disabled']

----------------------------------------------------------------------
29. unique value number in DRIVACT : 13

29. unique value in DRIVACT: [nan 'Driving Properly' 'Lost control'
'Improper Lane Change'
 'Disobeyed Traffic Control' 'Failed to Yield Right of Way' 'Other'
 'Speed too Fast For Condition' 'Exceeding Speed Limit' 'Improper
Turn'
 'Following too Close' 'Improper Passing' 'Wrong Way on One Way Road'
 'Speed too Slow']

----------------------------------------------------------------------
30. unique value number in DRIVCOND : 10

30. unique value in DRIVCOND: [nan 'Normal' 'Ability Impaired, Alcohol
Over .08' 'Inattentive' 'Unknown'
 'Medical or Physical Disability' 'Had Been Drinking' 'Fatigue'
'Other'
 'Ability Impaired, Alcohol' 'Ability Impaired, Drugs']

----------------------------------------------------------------------
31. unique value number in PEDTYPE : 16

31. unique value in PEDTYPE: [nan 'Pedestrian hit at mid-block'
 'Vehicle is going straight thru inter.while ped cross without ROW'
 'Vehicle is going straight thru inter.while ped cross with ROW'
 'Pedestrian hit a PXO/ped. Mid-block signal'
 'Pedestrian involved in a collision with transit vehicle anywhere
along roadway'
 'Vehicle turns left while ped crosses with ROW at inter.'
 'Other / Undefined'
 'Vehicle turns left while ped crosses without ROW at inter.'
 'Vehicle turns right while ped crosses with ROW at inter.'
 'Vehicle hits the pedestrian walking or running out from between
parked vehicles at mid-block'
 'Unknown' 'Vehicle turns right while ped crosses without ROW at
inter.'
 'Pedestrian hit on sidewalk or shoulder'
 'Vehicle is reversing and hits pedestrian'
 'Pedestrian hit at private driveway' 'Pedestrian hit at parking lot']


----------------------------------------------------------------------
32. unique value number in PEDACT : 15

32. unique value in PEDACT: [nan 'Crossing without right of way'
'Crossing with right of way'
```

```
 'Crossing, Pedestrian Crossover' 'Crossing, no Traffic Control'
'Other'
 'Running onto Roadway' 'Coming From Behind Parked Vehicle'
 'Pushing/Working on Vehicle' 'On Sidewalk or Shoulder'
 'Walking on Roadway Against Traffic' 'Playing or Working on Highway'
 'Person Getting on/off Vehicle' 'Walking on Roadway with Traffic'
 'Crossing marked crosswalk without ROW'
 'Person Getting on/off School Bus']

----------------------------------------------------------------------
33. unique value number in PEDCOND : 10

33. unique value in PEDCOND: [nan 'Inattentive' 'Normal' 'Unknown'
'Medical or Physical Disability'
 'Had Been Drinking' 'Ability Impaired, Alcohol' 'Other'
 'Ability Impaired, Alcohol Over .80' 'Ability Impaired, Drugs'
'Fatigue']

----------------------------------------------------------------------
34. unique value number in CYCLISTYPE : 22

34. unique value in CYCLISTYPE: [nan 'Motorist turned left across
cyclists path.'
 'Motorist turning right on green or amber at signalized intersection
strikes cyclist.'
 'Cyclist struck opened vehicle door'
 'Cyclist and Driver travelling in same direction. One vehicle rear-
ended the other.'
 'Motorist turns right at non-signal Inter.(stop, yield, no cont.,and
dwy) and strikes cyclist.'
 'Cyclist makes u-turn in-front of driver.'
 'Cyclist and Driver travelling in same direction. One vehicle
sideswipes the other.'
 'Cyclist strikes pedestrian.'
 'Cyclist loses control and strikes object (pole, ttc track)'
 'Cyclist without ROW rides into path of motorist at inter, lnwy, dwy-
Cyclist not turn.'
 'Cyclist turns right across motorists path'
 'Motorist turning right on red at signalized intersection strikes
cyclist.'
 'Cyclist turned left across motorists path.'
 'Motorist without ROW drives into path of cyclist at inter, lnwy,
dwy-Driver not turn.'
 'Cyclist rode off sidewalk into road at midblock.'
 'Insufficient information (to determine cyclist crash type).'
 'Cyclist struck at PXO(cyclist either travel in same dir. as veh. or
ride across xwalk)'
 'Motorist reversing struck cyclist.'
 'Motorist loses control and strikes cyclist.'
 'Cyclist strikes a parked vehicle.'
```

```
 'Motorist makes u-turn in-front of cyclist.'
 'Cyclist falls off bike - no contact with motorist.']

----------------------------------------------------------------------
35. unique value number in CYCACT : 11

35. unique value in CYCACT: [nan 'Driving Properly' 'Other' 'Improper
Turn' 'Improper Passing'
 'Disobeyed Traffic Control' 'Lost control' 'Failed to Yield Right of
Way'
 'Improper Lane Change' 'Following too Close'
 'Speed too Fast For Condition' 'Wrong Way on One Way Road']

----------------------------------------------------------------------
36. unique value number in CYCCOND : 10

36. unique value in CYCCOND: [nan 'Normal' 'Inattentive' 'Had Been
Drinking' 'Unknown'
 'Ability Impaired, Drugs' 'Ability Impaired, Alcohol Over .80'
 'Medical or Physical Disability' 'Ability Impaired, Alcohol' 'Other'
 'Fatigue']

----------------------------------------------------------------------
37. unique value number in PEDESTRIAN : 1

37. unique value in PEDESTRIAN: [nan 'Yes']

----------------------------------------------------------------------
38. unique value number in CYCLIST : 1

38. unique value in CYCLIST: [nan 'Yes']

----------------------------------------------------------------------
39. unique value number in AUTOMOBILE : 1

39. unique value in AUTOMOBILE: ['Yes' nan]

----------------------------------------------------------------------
40. unique value number in MOTORCYCLE : 1

40. unique value in MOTORCYCLE: [nan 'Yes']

----------------------------------------------------------------------
41. unique value number in TRUCK : 1

41. unique value in TRUCK: [nan 'Yes']

----------------------------------------------------------------------
42. unique value number in TRSN_CITY_VEH : 1

42. unique value in TRSN_CITY_VEH: [nan 'Yes']
```

```
----------------------------------------------------------------------
43. unique value number in EMERG_VEH : 1

43. unique value in EMERG_VEH: [nan 'Yes']

----------------------------------------------------------------------
44. unique value number in PASSENGER : 1

44. unique value in PASSENGER: ['Yes' nan]

----------------------------------------------------------------------
45. unique value number in SPEEDING : 1

45. unique value in SPEEDING: ['Yes' nan]

----------------------------------------------------------------------
46. unique value number in AG_DRIV : 1

46. unique value in AG_DRIV: ['Yes' nan]

----------------------------------------------------------------------
47. unique value number in REDLIGHT : 1

47. unique value in REDLIGHT: [nan 'Yes']

----------------------------------------------------------------------
48. unique value number in ALCOHOL : 1

48. unique value in ALCOHOL: ['Yes' nan]

----------------------------------------------------------------------
49. unique value number in DISABILITY : 1

49. unique value in DISABILITY: [nan 'Yes']

----------------------------------------------------------------------
50. unique value number in HOOD_158 : 159

50. unique value in HOOD_158: ['60' '64' '78' '83' '47' '144' '166'
'5' '126' '129' '157' '43' '22'
 '100' '89' '38' '136' '128' '95' '119' '143' '98' '80' '160' '96'
'148'
 '88' '1' '149' '125' '66' '54' '110' '59' '4' '172' '56' '85' '145'
'159'
 '101' '11' '73' '70' '138' '57' '87' '161' '146' '124' '81' '6' '116'
 '171' '152' '27' '91' 'NSA' '142' '30' '29' '111' '92' '115' '165'
'44'
 '42' '170' '32' '139' '169' '25' '120' '103' '39' '122' '102' '21'
'112'
 '37' '65' '40' '9' '99' '16' '97' '106' '154' '35' '53' '168' '24'
```

```
'8'
 '71' '63' '94' '135' '174' '162' '151' '18' '33' '10' '150' '107'
'156'
 '164' '84' '114' '58' '48' '153' '20' '61' '50' '123' '109' '167'
'130'
 '34' '108' '163' '52' '23' '55' '13' '113' '7' '31' '68' '86' '2'
'118'
 '3' '147' '72' '155' '41' '140' '158' '105' '28' '46' '141' '62' '90'
 '79' '36' '134' '133' '69' '121' '49' '19' '15' '12' '67' '74' '173']


----------------------------------------------------------------------
51. unique value number in NEIGHBOURHOOD_158 : 159

51. unique value in NEIGHBOURHOOD_158: ['Woodbine-Lumsden' 'Woodbine
Corridor' 'Kensington-Chinatown'
 'Dufferin Grove' 'Don Valley Village' 'Morningside Heights'
 'St Lawrence-East Bayfront-The Islands' 'Elms-Old Rexdale' 'Dorset
Park'
 'Agincourt North' 'Bendale South' 'Victoria Village' 'Humbermede'
 'Yonge-Eglinton' 'Runnymede-Bloor West Village' 'Lansing-Westgate'
 'West Hill' 'Agincourt South-Malvern West' 'Annex' 'Wexford/Maryvale'
 'West Rouge' 'Rosedale-Moore Park' 'Palmerston-Little Italy'
 'Mimico-Queensway' 'Casa Loma' "East L'Amoreaux" 'High Park North'
 'West Humber-Clairville' "Parkwoods-O'Connor Hills" 'Ionview'
'Danforth'
 "O'Connor-Parkview" 'Keelesdale-Eglinton West' 'Danforth East York'
 'Rexdale-Kipling' 'Dovercourt Village' 'Leaside-Bennington'
 'South Parkdale' 'Malvern West' 'Etobicoke City Centre'
 'Forest Hill South' 'Eringate-Centennial-West Deane' 'Moss Park'
 'South Riverdale' 'Eglinton East' 'Broadview North' 'High Park-
Swansea'
 'Humber Bay Shores' 'Malvern East' 'Kennedy Park' 'Trinity-Bellwoods'
 'Kingsview Village-The Westway' 'Steeles' 'Junction-Wallace Emerson'
 'East Willowdale' 'York University Heights' 'Weston-Pelham Park'
'NSA'
 'Woburn North' 'Brookhaven-Amesbury' 'Maple Leaf' 'Rockcliffe-Smythe'
 'Corso Italia-Davenport' 'Mount Dennis' 'Harbourfront-CityPlace'
 'Flemingdon Park' 'Banbury-Don Mills' 'Yonge-Bay Corridor'
 'Englemount-Lawrence' 'Scarborough Village' 'Bay-Cloverhill'
 'Glenfield-Jane Heights' 'Clairlea-Birchmount' 'Lawrence Park South'
 'Bedford Park-Nortown' 'Birchcliffe-Cliffside' 'Forest Hill North'
 'Humber Summit' 'Beechborough-Greenbrook' 'Willowdale West'
 'Greenwood-Coxwell' 'St.Andrew-Windfields' 'Edenbridge-Humber Valley'
 'Mount Pleasant East' 'Stonegate-Queensway' 'Yonge-St.Clair'
 'Humewood-Cedarvale' 'Oakdale-Beverley Heights' 'Westminster-Branson'
 'Henry Farm' 'Downtown Yonge East' 'Black Creek'
 'Humber Heights-Westmount' 'Cabbagetown-South St.James Town'
 'The Beaches' 'Wychwood' 'Morningside' 'South Eglinton-Davisville'
 'West Queen West' 'Yonge-Doris' 'New Toronto' 'Clanton Park'
```

```
 'Princess-Rosethorn' 'Fenside-Parkwoods' 'Oakwood Village'
 'Bendale-Glen Andrew' 'Wellington Place' 'Little Portugal'
 'Lambton Baby Point' 'Old East York' 'Hillcrest Village' 'Avondale'
 'Alderwood' 'Taylor-Massey' 'Newtonbrook East' 'Cliffcrest'
 'Caledonia-Fairbank' 'Church-Wellesley' 'Milliken' 'Bathurst Manor'
 'Briar Hill-Belgravia' 'Fort York-Liberty Village' 'Bayview Village'
 'Pelmo Park-Humberlea' 'Thorncliffe Park' 'Etobicoke West Mall'
'Weston'
 'Willowridge-Martingrove-Richview' 'Yorkdale-Glen Park' 'North
Riverdale'
 'Roncesvalles' 'Mount Olive-Silverstone-Jamestown'
 "Tam O'Shanter-Sullivan" 'Thistletown-Beaumond Heights' "L'Amoreaux
West"
 'Regent Park' 'Downsview' 'Bridle Path-Sunnybrook-York Mills'
'Guildwood'
 'Islington' 'Lawrence Park North' 'Rustic' 'Pleasant View'
 'Golfdale-Cedarbrae-Woburn' 'East End-Danforth' 'Junction Area'
 'University' 'Newtonbrook West' 'Highland Creek' 'Centennial
Scarborough'
 'Blake-Jones' 'Oakridge' 'Bayview Woods-Steeles' 'Long Branch'
 'Kingsway South' 'Markland Wood' 'Playter Estates-Danforth'
 'North St.James Town' 'North Toronto']

----------------------------------------------------------------------
52. unique value number in HOOD_140 : 141

52. unique value in HOOD_140: ['60' '64' '78' '83' '47' '131' '77' '5'
'126' '129' '127' '43' '22' '100'
 '89' '38' '136' '128' '95' '119' '98' '80' '17' '96' '117' '88' '1'
'45'
 '125' '66' '54' '110' '59' '4' '93' '56' '85' '132' '12' '101' '11'
'73'
 '70' '138' '57' '87' '81' '6' '116' '51' '27' '91' 'NSA' '137' '28'
'29'
 '111' '92' '115' '44' '42' '76' '32' '139' '75' '25' '120' '103' '39'
 '122' '14' '102' '21' '112' '37' '65' '40' '9' '99' '16' '97' '106'
'26'
 '35' '53' '24' '8' '71' '63' '94' '135' '30' '104' '82' '18' '33'
'86'
 '10' '107' '84' '114' '58' '48' '20' '61' '50' '123' '109' '130' '34'
 '108' '52' '23' '55' '13' '113' '7' '31' '68' '2' '118' '3' '72' '41'
 '124' '140' '105' '46' '62' '90' '79' '36' '134' '133' '69' '121'
'49'
 '19' '15' '67' '74']

----------------------------------------------------------------------
53. unique value number in NEIGHBOURHOOD_140 : 141

53. unique value in NEIGHBOURHOOD_140: ['Woodbine-Lumsden (60)'
'Woodbine Corridor (64)'
```

```
 'Kensington-Chinatown (78)' 'Dufferin Grove (83)'
 'Don Valley Village (47)' 'Rouge (131)'
 'Waterfront Communities-The Island (77)' 'Elms-Old Rexdale (5)'
 'Dorset Park (126)' 'Agincourt North (129)' 'Bendale (127)'
 'Victoria Village (43)' 'Humbermede (22)' 'Yonge-Eglinton (100)'
 'Runnymede-Bloor West Village (89)' 'Lansing-Westgate (38)'
 'West Hill (136)' 'Agincourt South-Malvern West (128)' 'Annex (95)'
 'Wexford/Maryvale (119)' 'Rosedale-Moore Park (98)'
 'Palmerston-Little Italy (80)' 'Mimico (includes Humber Bay Shores)
(17)'
 'Casa Loma (96)' "L'Amoreaux (117)" 'High Park North (88)'
 'West Humber-Clairville (1)' 'Parkwoods-Donalda (45)' 'Ionview (125)'
 'Danforth (66)' "O'Connor-Parkview (54)" 'Keelesdale-Eglinton West
(110)'
 'Danforth East York (59)' 'Rexdale-Kipling (4)'
 'Dovercourt-Wallace Emerson-Junction (93)' 'Leaside-Bennington (56)'
 'South Parkdale (85)' 'Malvern (132)' 'Markland Wood (12)'
 'Forest Hill South (101)' 'Eringate-Centennial-West Deane (11)'
 'Moss Park (73)' 'South Riverdale (70)' 'Eglinton East (138)'
 'Broadview North (57)' 'High Park-Swansea (87)' 'Trinity-Bellwoods
(81)'
 'Kingsview Village-The Westway (6)' 'Steeles (116)'
 'Willowdale East (51)' 'York University Heights (27)'
 'Weston-Pellam Park (91)' 'NSA' 'Woburn (137)' 'Rustic (28)'
 'Maple Leaf (29)' 'Rockcliffe-Smythe (111)' 'Corso Italia-Davenport
(92)'
 'Mount Dennis (115)' 'Flemingdon Park (44)' 'Banbury-Don Mills (42)'
 'Bay Street Corridor (76)' 'Englemount-Lawrence (32)'
 'Scarborough Village (139)' 'Church-Yonge Corridor (75)'
 'Glenfield-Jane Heights (25)' 'Clairlea-Birchmount (120)'
 'Lawrence Park South (103)' 'Bedford Park-Nortown (39)'
 'Birchcliffe-Cliffside (122)' 'Islington-City Centre West (14)'
 'Forest Hill North (102)' 'Humber Summit (21)'
 'Beechborough-Greenbrook (112)' 'Willowdale West (37)'
 'Greenwood-Coxwell (65)' 'St.Andrew-Windfields (40)'
 'Edenbridge-Humber Valley (9)' 'Mount Pleasant East (99)'
 'Stonegate-Queensway (16)' 'Yonge-St.Clair (97)'
 'Humewood-Cedarvale (106)' 'Downsview-Roding-CFB (26)'
 'Westminster-Branson (35)' 'Henry Farm (53)' 'Black Creek (24)'
 'Humber Heights-Westmount (8)' 'Cabbagetown-South St.James Town (71)'
 'The Beaches (63)' 'Wychwood (94)' 'Morningside (135)'
 'Brookhaven-Amesbury (30)' 'Mount Pleasant West (104)' 'Niagara (82)'
 'New Toronto (18)' 'Clanton Park (33)' 'Roncesvalles (86)'
 'Princess-Rosethorn (10)' 'Oakwood Village (107)' 'Little Portugal
(84)'
 'Lambton Baby Point (114)' 'Old East York (58)' 'Hillcrest Village
(48)'
 'Alderwood (20)' 'Taylor-Massey (61)' 'Newtonbrook East (50)'
 'Cliffcrest (123)' 'Caledonia-Fairbank (109)' 'Milliken (130)'
```

```
 'Bathurst Manor (34)' 'Briar Hill-Belgravia (108)' 'Bayview Village (52)'
 'Pelmo Park-Humberlea (23)' 'Thorncliffe Park (55)'
 'Etobicoke West Mall (13)' 'Weston (113)'
 'Willowridge-Martingrove-Richview (7)' 'Yorkdale-Glen Park (31)'
 'North Riverdale (68)' 'Mount Olive-Silverstone-Jamestown (2)'
 "Tam O'Shanter-Sullivan (118)" 'Thistletown-Beaumond Heights (3)'
 'Regent Park (72)' 'Bridle Path-Sunnybrook-York Mills (41)'
 'Kennedy Park (124)' 'Guildwood (140)' 'Lawrence Park North (105)'
 'Pleasant View (46)' 'East End-Danforth (62)' 'Junction Area (90)'
 'University (79)' 'Newtonbrook West (36)' 'Highland Creek (134)'
 'Centennial Scarborough (133)' 'Blake-Jones (69)' 'Oakridge (121)'
 'Bayview Woods-Steeles (49)' 'Long Branch (19)' 'Kingsway South (15)'
 'Playter Estates-Danforth (67)' 'North St.James Town (74)']

----------------------------------------------------------------------
54. unique value number in DIVISION : 17

54. unique value in DIVISION: ['D55' 'D14' 'D11' 'D33' 'D42' 'D51' 'D23' 'D41' 'D31' 'D53' 'D32' 'D43'
 'D22' 'D13' 'D52' 'D12' 'NSA']

----------------------------------------------------------------------
```

From this information, we can drop some columns that are not relevant for the machine learning algorithm:

- **X and Y features** will be dropped since we don't have information about the meaning of those two variables
- **INDEX and ACCNUMBER** can be dropped as we already have another identifier for each sample (OBJECTID)
- **STREET1, STREET2, OFFSET, HOOD_158, NEIGHBOURHOOD_158, HOOD_140, NEIGHBOURHOOD_140** have too many unique categorical values (more than 2000), so they won't be useful for the model
- **PEDTYPE, PEDACT, CYCLISTYPE, DIVISION, 'INITDIR'** are not relevant
- **'FATAL_NO'** Is not relevant as it is the result of a fatal accident, not the cause (number of disease)
- **VEHTYPE, DRIVCOND, PEDCOND, CYCCOND** are already partially covered with other boolean variables that generalize the characteristics of those features to any type of person involved (REDLIGHT : Red Light Related Collision, ALCOHOL : Alcohol Related Collision, DISABILITY : Medical or Physical Disability Related Collision, etc.)
- As **INJURY** has a class called "Fatal" it will produce a data leakage to the target variable, so we have to discard it.

Also we gain some valuables insights as:

- We have a **DATE** column that is not in a datetime format so we can convert it and use it to generate new numerical features as **DAY, MONTH and YEAR**

- The **TIME** feature is not in a time format and also is a number between 0000 to 2359 (), we can map it to make it a numerical feature but with a reduced range so it wont affect the performance of the model. This is particularly true for models that are sensitive to the scale of input features, such as linear regression, k-nearest neighbors, and neural networks, so it becomes necessary.

- We can group similar road classes from the **ROAD_CLASS** together based on their characteristics. For instance:

  a. Combine 'Expressway' and 'Expressway Ramp': These could be grouped as "Expressway."
  b. Combine 'Major Arterial' and 'Minor Arterial': These could be grouped as "Arterial."
  c. Combine 'Collector' and 'Local': These could be grouped as "Local Roads."
  d. Combine "Laneway," "Other," and "Pending" into "Other"

- Similarly, we can do it for **ACCLOC**:

  a. **Intersection-Related**: Combine Intersection Related, At Intersection, and At/Near Private Drive into one category since they all relate to intersections or nearby areas.
  b. **Non-Intersection**: Combine Non Intersection, Private Driveway, and Laneway into another category as they are not related to intersections and represent different non-major road types.
  c. **Structures**: Combine Underpass or Tunnel and Overpass or Bridge into a "Structures" category as they represent structural elements in the road network.
  d. Keep **Trail** as a separate category

And so on for the other similar features:

- **TRAFFCTL (Traffic Control)**
  a. No Control
  b. Signals: Traffic Signal, Traffic Controller, Traffic Gate
  c. Signs: Stop Sign, Yield Sign
  d. Pedestrian Controls: Pedestrian Crossover, School Guard, Police Control, Streetcar (Stop for)
- **VISIBILITY**
  a. Clear Conditions: Clear
  b. Precipitation: Rain, Snow, Freezing Rain, Drifting Snow
  c. Obstructions: Fog, Mist, Smoke, Dust, Strong wind
  d. Other: Other
- **LIGHT**
  a. Dark: Dark, Dark, artificial
  b. Daylight: Daylight, Daylight, artificial
  c. Twilight: Dusk, Dawn, Dusk, artificial, Dawn, artificial
  d. Other: Other
- **RDSFCOND (Road Surface Condition)**

a. Dry: Dry
b. Wet/Slippery: Wet, Slush, Ice, Spilled liquid
c. Snow: Loose Snow, Packed Snow, Loose Sand or Gravel
d. Other: Other

- **IMPACTYPE (Impact Type)**
    a. Vehicle-Vehicle: Approaching, Rear End, Sideswipe, Angle, Turning Movement
    b. Vehicle-Person: Pedestrian Collisions, Cyclist Collisions
    c. Single Vehicle Movement: SMV Other, SMV Unattended Vehicle
    d. Other: Other
- **INVTYPE (Involved Type)**
    a. Occupants: Passenger, Driver, Vehicle Owner, Motorcycle Driver, Truck Driver, Motorcycle Passenger, Moped Driver
    b. Non-Occupants: Pedestrian, Cyclist, In-Line Skater, Wheelchair, Pedestrian - Not Hit, Cyclist Passenger
    c. Other: Other Property Owner, Other, Driver - Not Hit, Trailer Owner, Witness

Also we need to convert the following categorical features into a numerical format:

- **Ordinal features** like INVAGE or INJURY
- **Nominal features** like ROAD_CLASS, ACCLOC or DISTRICT

As well as our **TARGET FEATURE: ACCLASS (Accident Class)**

1. Non-Fatal: Non-Fatal Injury
2. Fatal: Fatal

And **normalize** some continuos values like **LONGITUDE and LATITUDE**

Null values

We can search for null values and sort them in descending order to see if there are columns with too many null values that must be eliminated

```
# Count null values in each column
null_counts = train_df.isnull().sum()

# Sort the counts in descending order
sorted_null_counts = null_counts.sort_values(ascending=False)

# Display the result
print([sorted_null_counts[:25]])
```

```
[EMERG_VEH        14981
DISABILITY       14580
FATAL_NO         14407
CYCCOND          14380
CYCACT           14379
CYCLISTYPE       14365
ALCOHOL          14328
```

```
TRSN_CITY_VEH    14077
TRUCK            14067
MOTORCYCLE       13838
REDLIGHT         13725
CYCLIST          13422
OFFSET           13072
SPEEDING         13002
PEDCOND          12555
PEDACT           12550
PEDTYPE          12540
PASSENGER         9367
PEDESTRIAN        9034
DRIVCOND          7579
DRIVACT           7575
AG_DRIV           7304
INJURY            7189
MANOEUVER         6514
ACCLOC            5450
dtype: int64]
```

We could think about deleting this columns with more than 50% of the values being null values but doing some exploration, this is because some of them are boolean variables which take a null value instead of using False, or a numerical feature that uses a missing value instead of a cero.

This is the case for:

- EMERG_VEH, DISABILITY, FATAL_NO, ALCOHOL, TRSN_CITY_VEH, TRUCK, MOTORCYCLE, REDLIGHT, CYCLIST, SPEEDING, INJURY, PASSENGER

On the other hand, that is not the case for features like:

- CYCCOND, CYCACT, CYCLISTYPE, OFFSET, PEDCOND, PEDACT, PEDTYPE, PEDESTRIAN, DRIVCOND, DRIVACT, AG_DRIV, ACCLOC, MANOEUVER

and can be droped

Dropping columns

```
#X, INDEX, ACCNUMBER, STREET1, STREET2, OFFSET, HOOD_158,
NEIGHBOURHOOD_158, HOOD_140, NEIGHBOURHOOD_140,PEDTYPE, PEDACT,
CYCLISTYPE, DIVISION,VEHTYPE, DRIVCOND, PEDCOND, CYCCOND, CYCCOND,
CYCACT, CYCLISTYPE, OFFSET, PEDCOND, PEDACT, PEDTYPE, PEDESTRIAN,
DRIVCOND, DRIVACT, AG_DRIV, ACCLOC
columns_to_drop = ['X', 'Y', 'ACCLOC', 'ACCNUM', 'AG_DRIV', 'CYCACT',
'CYCCOND', 'CYCLISTYPE', 'DIVISION',
                   'DRIVACT', 'DRIVCOND', 'PEDACT', 'PEDCOND',
'PEDTYPE', 'PEDESTRIAN', 'INDEX_', 'NEIGHBOURHOOD_140',
                   'NEIGHBOURHOOD_158', 'OFFSET', 'HOOD_140',
'HOOD_158', 'STREET1', 'STREET2', 'VEHTYPE', 'MANOEUVER', 'FATAL_NO',
'INITDIR', 'INJURY']
```

```python
print(len(columns_to_drop))

# Drop the columns and update the original dataframe
train_df.drop(columns=columns_to_drop, inplace=True)
test_df.drop(columns=columns_to_drop, inplace=True)
```

```
28
```

Final feature selection

```python
train_df.head()
```

```
   OBJECTID                    DATE  TIME      ROAD_CLASS  \
0         1  2006/01/01 10:00:00+00   236  Major Arterial
1         2  2006/01/01 10:00:00+00   236  Major Arterial
2         3  2006/01/01 10:00:00+00   236  Major Arterial
3         4  2006/01/01 10:00:00+00   236  Major Arterial
4         5  2006/01/01 10:00:00+00   236  Major Arterial

                DISTRICT   LATITUDE  LONGITUDE    TRAFFCTL VISIBILITY
LIGHT  \
0  Toronto and East York  43.699595 -79.318797  No Control      Clear
Dark
1  Toronto and East York  43.699595 -79.318797  No Control      Clear
Dark
2  Toronto and East York  43.699595 -79.318797  No Control      Clear
Dark
3  Toronto and East York  43.699595 -79.318797  No Control      Clear
Dark
4  Toronto and East York  43.699595 -79.318797  No Control      Clear
Dark

   ... AUTOMOBILE MOTORCYCLE TRUCK TRSN_CITY_VEH EMERG_VEH PASSENGER
SPEEDING  \
0  ...        Yes        NaN   NaN           NaN       NaN       Yes
Yes
1  ...        Yes        NaN   NaN           NaN       NaN       Yes
Yes
2  ...        Yes        NaN   NaN           NaN       NaN       Yes
Yes
3  ...        Yes        NaN   NaN           NaN       NaN       Yes
Yes
4  ...        Yes        NaN   NaN           NaN       NaN       Yes
Yes

  REDLIGHT ALCOHOL DISABILITY
0      NaN     Yes        NaN
1      NaN     Yes        NaN
2      NaN     Yes        NaN
```

```
3      NaN     Yes        NaN
4      NaN     Yes        NaN

[5 rows x 26 columns]
```

```
test_df.head()
```

```
   OBJECTID                    DATE  TIME      ROAD_CLASS
DISTRICT  \
0     15001  2018/09/26 08:00:00+00  2053  Major Arterial
Scarborough
1     15002  2018/09/26 08:00:00+00  2053  Major Arterial
Scarborough
2     15003  2018/09/28 08:00:00+00   806  Major Arterial
Scarborough
3     15004  2018/09/28 08:00:00+00   806  Major Arterial
Scarborough
4     15005  2018/09/28 08:00:00+00  1018  Major Arterial  Etobicoke
York

    LATITUDE  LONGITUDE        TRAFFCTL VISIBILITY
LIGHT  ...  \
0  43.782229 -79.292499      No Control      Clear  Dark,
artificial  ...
1  43.782229 -79.292499      No Control      Clear  Dark,
artificial  ...
2  43.730773 -79.273853  Traffic Signal      Clear
Daylight  ...
3  43.730773 -79.273853  Traffic Signal      Clear
Daylight  ...
4  43.691409 -79.500911      No Control      Clear
Daylight  ...

  AUTOMOBILE MOTORCYCLE TRUCK TRSN_CITY_VEH EMERG_VEH PASSENGER
SPEEDING  \
0        Yes        NaN   NaN           NaN       NaN       NaN
NaN
1        Yes        NaN   NaN           NaN       NaN       NaN
NaN
2        Yes        NaN   NaN           NaN       NaN       NaN
NaN
3        Yes        NaN   NaN           NaN       NaN       NaN
NaN
4        Yes        NaN   NaN           NaN       NaN       NaN
NaN

  REDLIGHT ALCOHOL DISABILITY
0      NaN     NaN        NaN
1      NaN     NaN        NaN
2      NaN     NaN        NaN
```

```
3      NaN     NaN          NaN
4      NaN     NaN          NaN

[5 rows x 25 columns]
```

```
train_df.columns
```

```
Index(['OBJECTID', 'DATE', 'TIME', 'ROAD_CLASS', 'DISTRICT',
'LATITUDE',
       'LONGITUDE', 'TRAFFCTL', 'VISIBILITY', 'LIGHT', 'RDSFCOND',
'ACCLASS',
       'IMPACTYPE', 'INVTYPE', 'INVAGE', 'CYCLIST', 'AUTOMOBILE',
'MOTORCYCLE',
       'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING',
       'REDLIGHT', 'ALCOHOL', 'DISABILITY'],
      dtype='object')
```

After that initial analysis we still have 25 features excluding the target feature and object_id, so we should consider choosing only the most important ones as having a large number of features can lead to overfitting, where the model learns noise rather than patterns. Also, many of the current features need to be transform into one-hot encoding so the number of features will increase even more.

We can eliminate DATE, TIME, LATITUDE, LONGITUDE, INVTYPE

```
columns_to_drop = ['DATE', 'TIME','LATITUDE', 'LONGITUDE','INVTYPE']

# Drop the columns and update the original dataframe
train_df.drop(columns=columns_to_drop, inplace=True)
test_df.drop(columns=columns_to_drop, inplace=True)

train_df.columns
```

```
Index(['OBJECTID', 'ROAD_CLASS', 'DISTRICT', 'TRAFFCTL', 'VISIBILITY',
'LIGHT',
       'RDSFCOND', 'ACCLASS', 'IMPACTYPE', 'INVAGE', 'CYCLIST',
'AUTOMOBILE',
       'MOTORCYCLE', 'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH',
'PASSENGER',
       'SPEEDING', 'REDLIGHT', 'ALCOHOL', 'DISABILITY'],
      dtype='object')
```

Cleansing

Fill nulls with 'No'

As some of the variables are boolean, but don't have the proper format, first we can fill the null values with a String "No" to determine a False case for that sample, and make the dtype consistent so we can transform both labels into a numerical binary mapping later on.

```
boolean_columns = ['CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK',
'TRSN_CITY_VEH', 'EMERG_VEH',
       'PASSENGER', 'SPEEDING', 'REDLIGHT', 'ALCOHOL', 'DISABILITY']

# Fill null values with 'No' for the train DataFrame
train_df.loc[:, boolean_columns] = train_df.loc[:,
boolean_columns].fillna('No')

# Fill null values with 'No' for the test DataFrame
test_df.loc[:, boolean_columns] = test_df.loc[:,
boolean_columns].fillna('No')

train_df.head()
```

```
   OBJECTID      ROAD_CLASS               DISTRICT    TRAFFCTL
VISIBILITY  \
0         1  Major Arterial  Toronto and East York  No Control
Clear
1         2  Major Arterial  Toronto and East York  No Control
Clear
2         3  Major Arterial  Toronto and East York  No Control
Clear
3         4  Major Arterial  Toronto and East York  No Control
Clear
4         5  Major Arterial  Toronto and East York  No Control
Clear

  LIGHT RDSFCOND            ACCLASS    IMPACTYPE    INVAGE  ...
AUTOMOBILE  \
0  Dark      Wet  Non-Fatal Injury  Approaching  50 to 54  ...
Yes
1  Dark      Wet  Non-Fatal Injury  Approaching  15 to 19  ...
Yes
2  Dark      Wet  Non-Fatal Injury  Approaching  55 to 59  ...
Yes
3  Dark      Wet  Non-Fatal Injury  Approaching  20 to 24  ...
Yes
4  Dark      Wet  Non-Fatal Injury  Approaching  15 to 19  ...
Yes

  MOTORCYCLE TRUCK TRSN_CITY_VEH EMERG_VEH PASSENGER SPEEDING REDLIGHT
\
0         No    No            No        No       Yes      Yes       No

1         No    No            No        No       Yes      Yes       No

2         No    No            No        No       Yes      Yes       No

3         No    No            No        No       Yes      Yes       No
```

```
4          No     No             No         No        Yes       Yes       No


  ALCOHOL DISABILITY
0     Yes         No
1     Yes         No
2     Yes         No
3     Yes         No
4     Yes         No

[5 rows x 21 columns]
```

Check and drop the null values left

```
# number of null value in all column using isnull function
print(train_df.isnull().sum())
```

```
OBJECTID           0
ROAD_CLASS       357
DISTRICT          16
TRAFFCTL          29
VISIBILITY        14
LIGHT              0
RDSFCOND          19
ACCLASS            0
IMPACTYPE          0
INVAGE             0
CYCLIST            0
AUTOMOBILE         0
MOTORCYCLE         0
TRUCK              0
TRSN_CITY_VEH      0
EMERG_VEH          0
PASSENGER          0
SPEEDING           0
REDLIGHT           0
ALCOHOL            0
DISABILITY         0
dtype: int64
```

```
test_df.isnull().sum()
```

```
OBJECTID           0
ROAD_CLASS       129
DISTRICT         213
TRAFFCTL          46
VISIBILITY        10
LIGHT              4
RDSFCOND          10
IMPACTYPE         27
```

```
INVAGE                 0
CYCLIST                0
AUTOMOBILE             0
MOTORCYCLE             0
TRUCK                  0
TRSN_CITY_VEH          0
EMERG_VEH              0
PASSENGER              0
SPEEDING               0
REDLIGHT               0
ALCOHOL                0
DISABILITY             0
dtype: int64
```

We drop the null values left in order to remove inconsistency as their proportion is too small compared to the whole dataset and dropping the null values will not affect too much the model's performance.

```python
# Drop rows with any missing values
train_df = train_df.dropna()
```

Impute Null values

As the Kaggle competition demands to submit a complete number of samples from the original test dataset, we will impute the null values for this test set instead of dropping them.

```python
# Mode imputation
for column in test_df.columns:
    mode_value = test_df[column].mode()[0]
    test_df[column].fillna(mode_value, inplace=True)
```

```
C:\Users\Andrea FS\AppData\Local\Temp\ipykernel_28136\2996424292.py:4:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  test_df[column].fillna(mode_value, inplace=True)
```

```python
test_df.isnull().sum()
```

```
OBJECTID         0
ROAD_CLASS       0
DISTRICT         0
TRAFFCTL         0
VISIBILITY       0
LIGHT            0
RDSFCOND         0
IMPACTYPE        0
INVAGE           0
CYCLIST          0
AUTOMOBILE       0
MOTORCYCLE       0
TRUCK            0
TRSN_CITY_VEH    0
EMERG_VEH        0
PASSENGER        0
SPEEDING         0
REDLIGHT         0
ALCOHOL          0
DISABILITY       0
dtype: int64
```

```
test_df.shape
```

```
(3956, 20)
```

Check for duplicates

Remove all the duplicate values from the dataset in order to decrease the inconsistency from our dataset.

```
# Check for duplicate rows
duplicates = train_df.duplicated()

print(train_df[duplicates])  # Display duplicate rows
```

```
Empty DataFrame
Columns: [OBJECTID, ROAD_CLASS, DISTRICT, TRAFFCTL, VISIBILITY, LIGHT,
RDSFCOND, ACCLASS, IMPACTYPE, INVAGE, CYCLIST, AUTOMOBILE, MOTORCYCLE,
TRUCK, TRSN_CITY_VEH, EMERG_VEH, PASSENGER, SPEEDING, REDLIGHT,
ALCOHOL, DISABILITY]
Index: []

[0 rows x 21 columns]
```

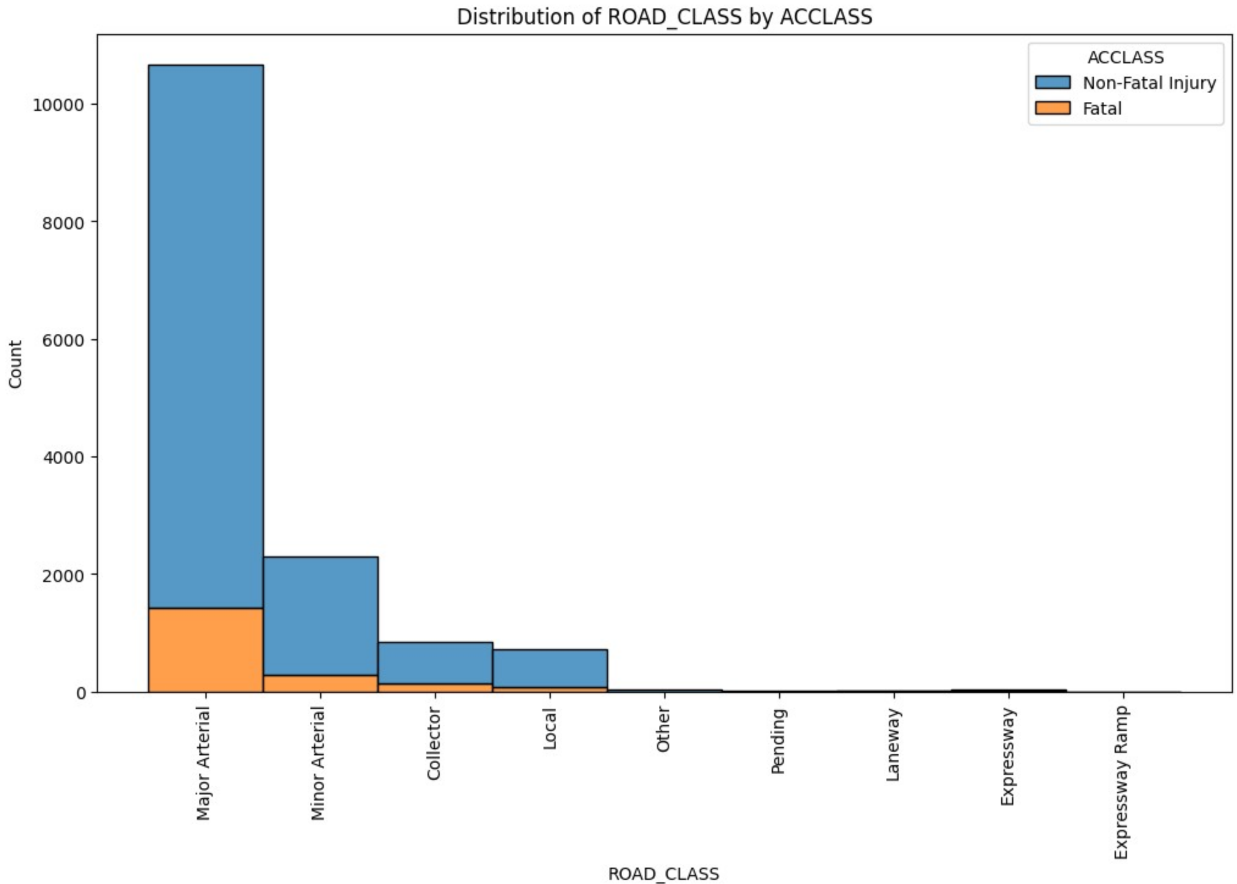
```
# Check for duplicate rows
duplicates = test_df.duplicated()

print(test_df[duplicates])  # Display duplicate rows
```

```
Empty DataFrame
Columns: [OBJECTID, ROAD_CLASS, DISTRICT, TRAFFCTL, VISIBILITY, LIGHT,
RDSFCOND, IMPACTYPE, INVAGE, CYCLIST, AUTOMOBILE, MOTORCYCLE, TRUCK,
TRSN_CITY_VEH, EMERG_VEH, PASSENGER, SPEEDING, REDLIGHT, ALCOHOL,
DISABILITY]
Index: []
```

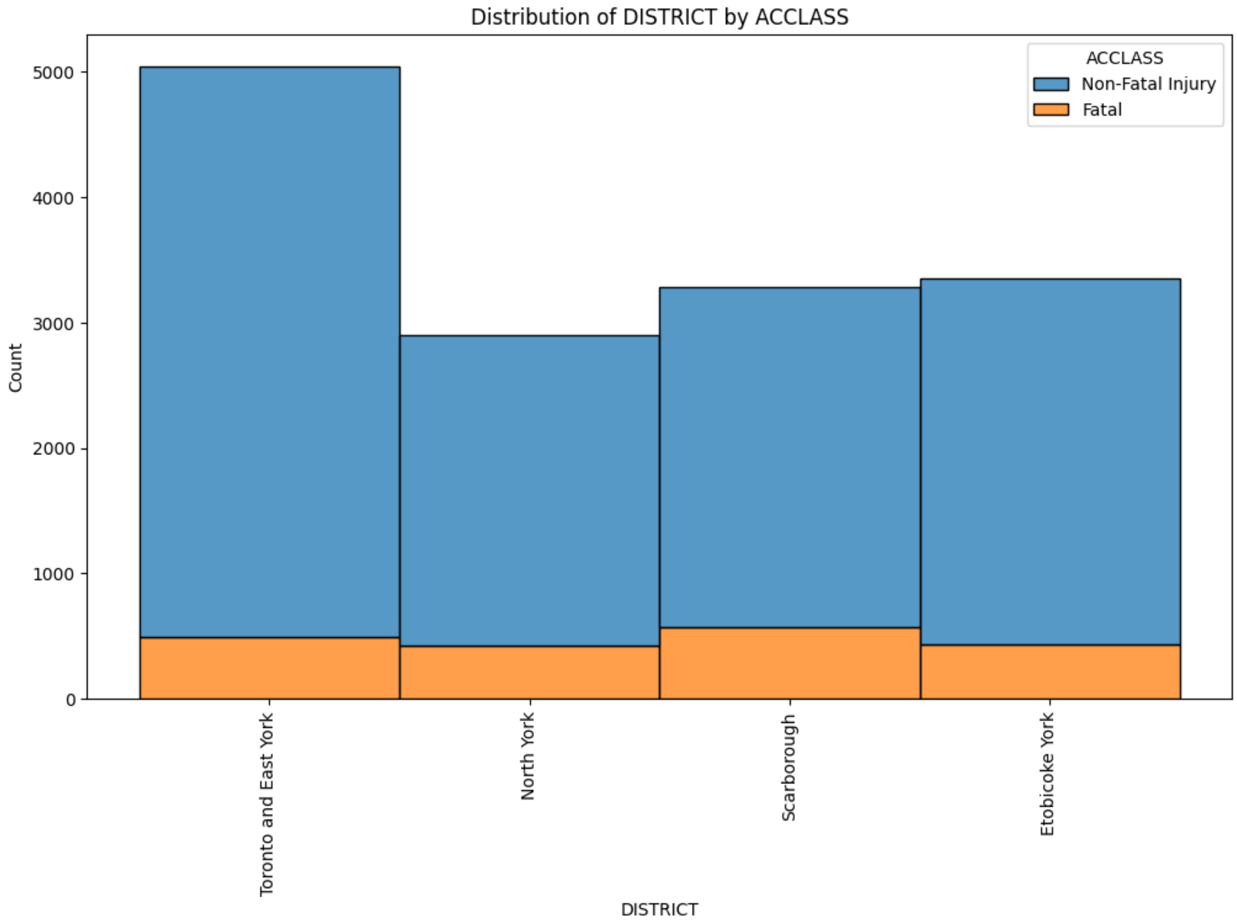
We can see above that there were no duplicate rows, which means that no duplicate values were deleted

Visualization

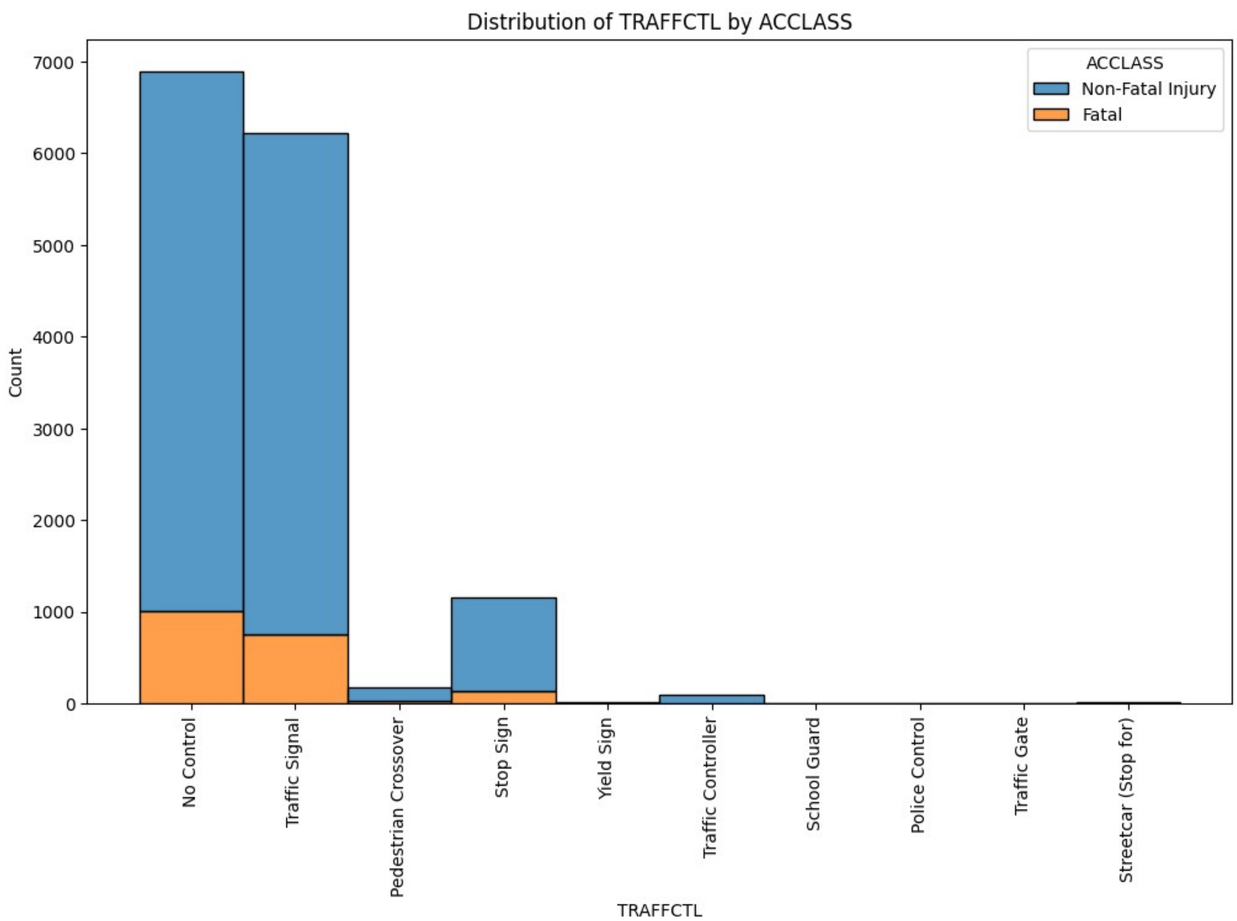
```python
# visualize the distribution of invage by class
# created function for histplot
def histplot_fun(INVAGE,rotation=90):
    plt.figure(figsize=(12,7))
    sns.histplot(data=train_df, x=INVAGE, hue='ACCLASS',
multiple='stack')
    plt.title(f"Distribution of {INVAGE} by ACCLASS")
    plt.xlabel(INVAGE)
    plt.ylabel('Count')
    plt.xticks(rotation=rotation)
    plt.show()

#showing the relationship between ACCLASS and ROAD_CLASS
histplot_fun('ROAD_CLASS')

#showing the relationship between ACCLASS and DISTRICT
histplot_fun('DISTRICT')

#showing the relationship between ACCLASS and TRAFFCTL
histplot_fun('TRAFFCTL')

#showing the relationship between ACCLASS and VISIBILITY
histplot_fun('VISIBILITY')

#showing the relationship between ACCLASS and LIGHT
histplot_fun('LIGHT')

#showing the relationship between ACCLASS and IMPACTYPE
histplot_fun('IMPACTYPE')

histplot_fun('INVAGE')
```

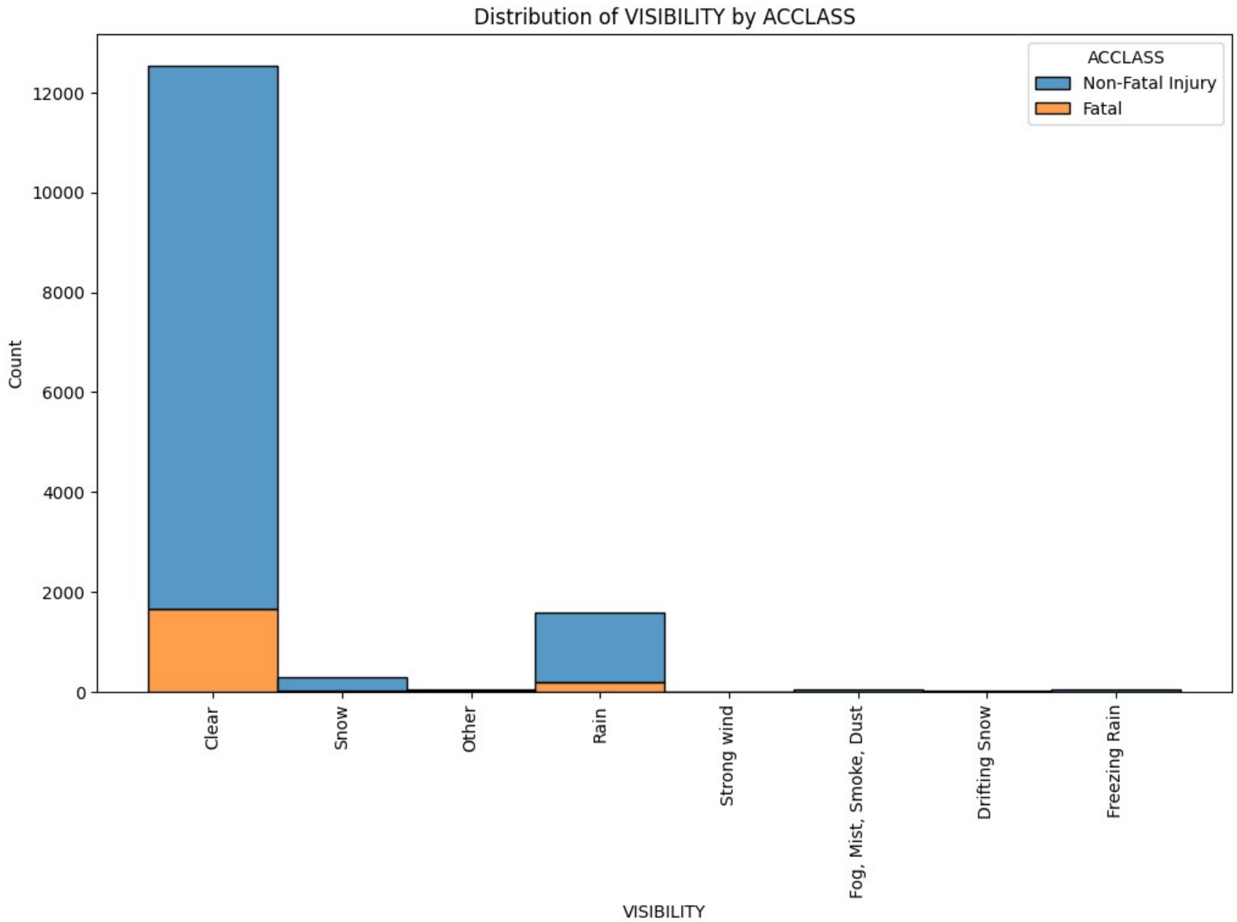


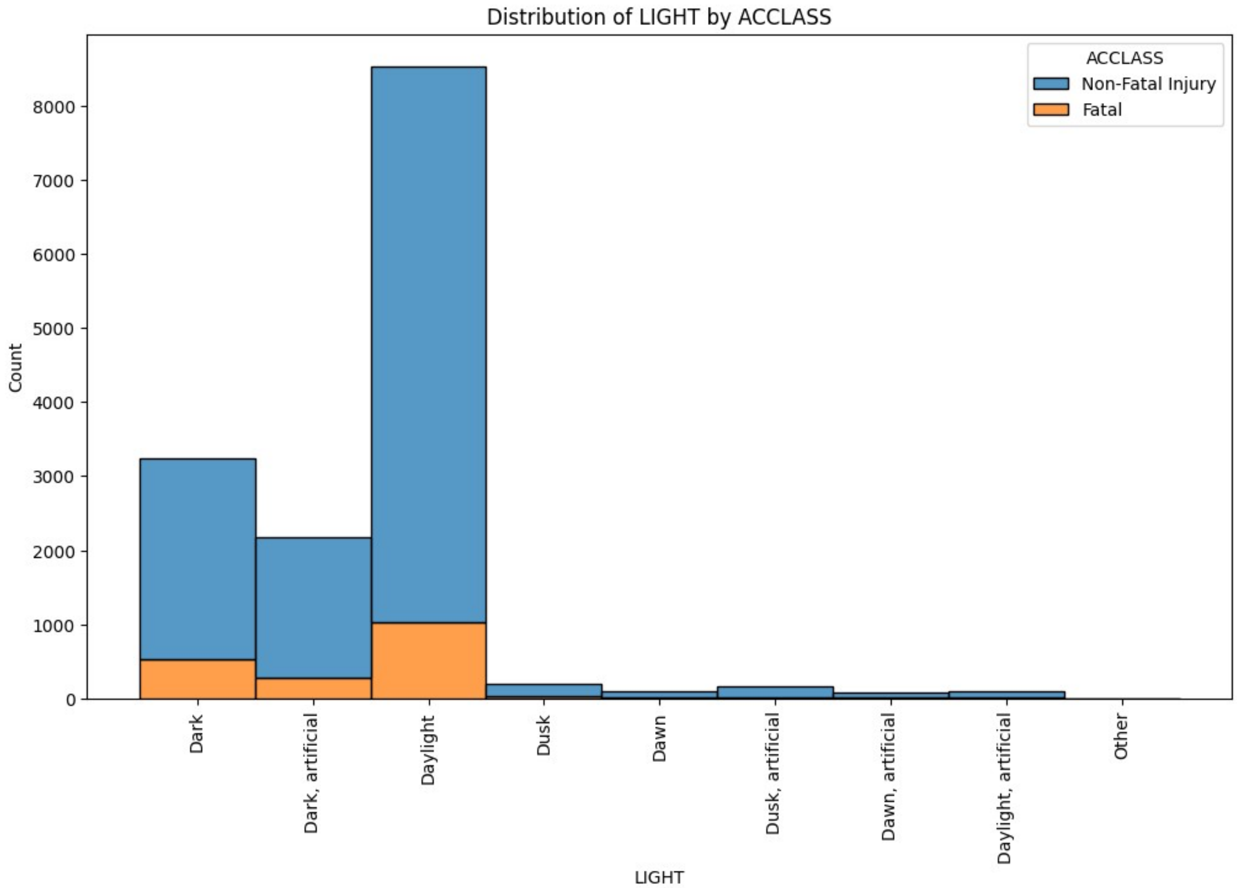
Distribution of ROAD_CLASS by ACCLASS
ACCLASS
Non-Fatal Injury
Fatal
Count
0
2000
4000
6000
8000
10000
Major Arterial
Minor Arterial
Collector
Local
Other
Pending
Laneway
Expressway
Expressway Ramp
ROAD_CLASS

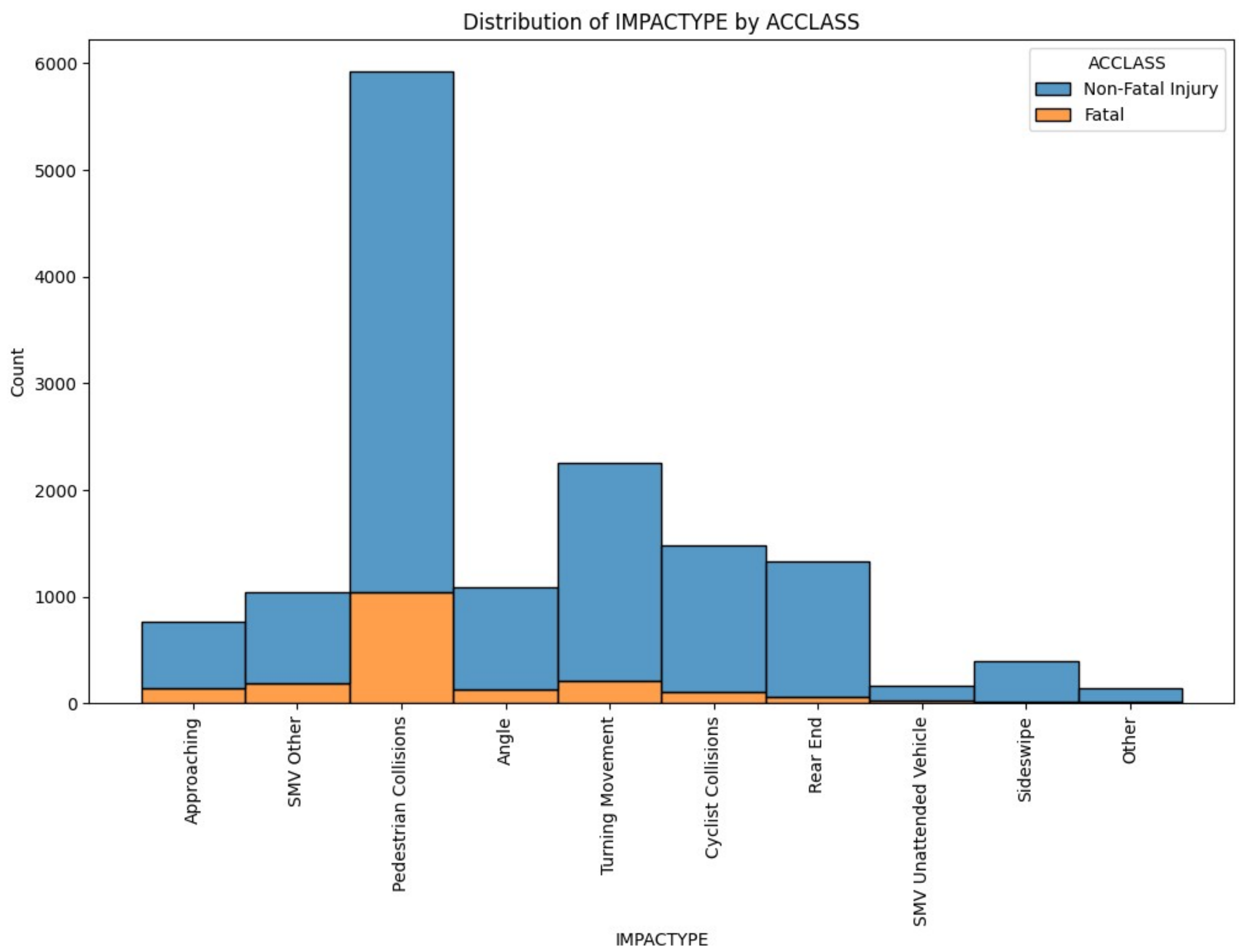


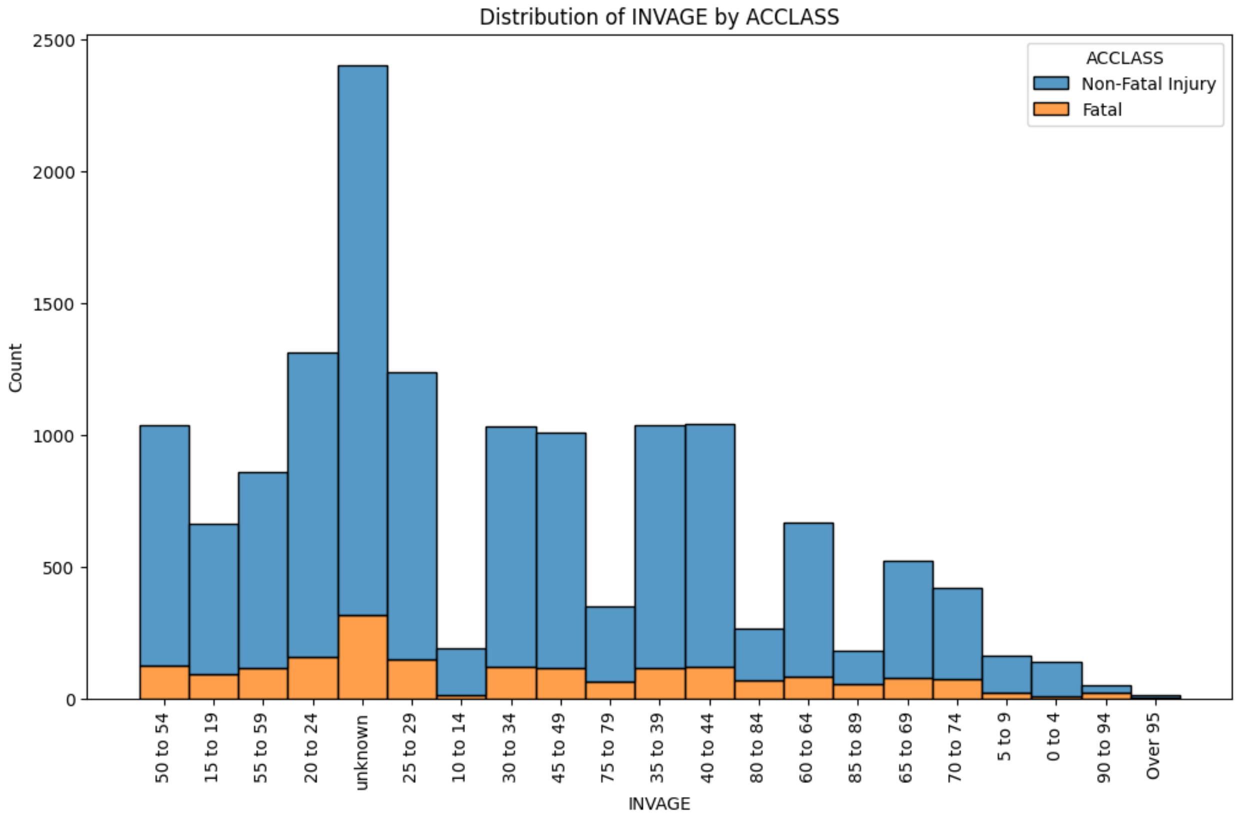
Distribution of DISTRICT by ACCLASS
ACCLASS
Non-Fatal Injury
Fatal
Count
0
1000
2000
3000
4000
5000
Toronto and East York
North York
Scarborough
Etobicoke York
DISTRICT


Distribution of TRAFFCTL by ACCLASS
ACCLASS
Non-Fatal Injury
Fatal
Count
0
1000
2000
3000
4000
5000
6000
7000
No Control
Traffic Signal
Pedestrian Crossover
Stop Sign
Yield Sign
Traffic Controller
School Guard
Police Control
Traffic Gate
Streetcar (Stop for)
TRAFFCTL


Distribution of VISIBILITY by ACCLASS
ACCLASS
Non-Fatal Injury
Fatal
Count
0
2000
4000
6000
8000
10000
12000
Clear
Snow
Other
Rain
Strong wind
Fog, Mist, Smoke, Dust
Drifting Snow
Freezing Rain
VISIBILITY


Distribution of LIGHT by ACCLASS
ACCLASS
Non-Fatal Injury
Fatal
Count
8000
7000
6000
5000
4000
3000
2000
1000
0
Dark
Dark, artificial
Daylight
Dusk
Dawn
Dusk, artificial
Dawn, artificial
Daylight, artificial
Other
LIGHT


Distribution of IMPACTYPE by ACCLASS
ACCLASS
Non-Fatal Injury
Fatal
Count
0
1000
2000
3000
4000
5000
6000
Approaching
SMV Other
Pedestrian Collisions
Angle
Turning Movement
Cyclist Collisions
Rear End
SMV Unattended Vehicle
Sideswipe
Other
IMPACTYPE



Histogram explanation of ACCLASS AND ROAD_CLASS:

In this figure, we can observe that most of non fatal injuries occur in major Arterial area of road class that is 8000 followed by minorArterial,Collector and local(count is below 2000). In fatal Injuries all the values in these columns are are less than 2000 and in other remaining columns we can only seem some values (like very close to zero). We can observe the RIGHT skewness as well

Histogram Explanation of District And Acclass:

AS we observe that there are highest non fatal injuries are in Toronto north york region But other region values are also near 3000. While for fatal injuries the count is near 500.The distribution of data is Multimodal.

Histogram explanation of Acclass and TRAFFCTL:

We can see that No control and traffic signal have highest number of Non Fatal injuries between range of 5000 to 6000. While there regions have values which are less than 1000. We can also observe the that data distribution is Skewed toward the right.

Histogram Explanation of Acclass and Visibility:

In this, clear visibility have most Non fatal Fatal injuries among the other and Count of injuries in rain ans snow is less than 500 while other parameter values are very close to zero or zero.the data distribution in this is Bimodal.

Histogram Explanation of Acclass and Light:

In this ,day light had largest non fatal injuries above 7000 followed by dark (near 3800) and Dark, artificial (near 2000).The fatal injury range of these are below 1000.While the other parameter values in range of (0-200).The data distribution in this Multimodal.

Histogram Explanation of Acclass and ImpactType:

In this we can observe that pedestrian collision is most number of non fatal injuries but its value for fatal injury is near 1000.while other parameters values range for non fatal is between 500 to 2200, for fatal its between 0 to 300. The data distribution in this is also Bimodal.

Histogram explanation of Acclass and Invage:

in This we can observe that ,Unknown ages have most number non fatal injuries among the other because other ages range for non fatal is between 0-1150.but fatal injuries are all less than 400.the data distribution in this is Multimodal

Pre-processing pipeline

Group mapping

The first step in the pre-processing pipeline is to perform a group mapping. As each variable has its own categorical classes that need to be reduced in order to get the minimum amount of features once the hot encoding is performance, without sacrificing the accuracy of the classes in describing its nature, we map some classes that had more than 5 features into 5 or less features.

```
# Define mapping functions for each feature
def group_age(age):
    if age in ['0 to 4', '5 to 9', '10 to 14']:
        return 'Children'
    elif age in ['15 to 19', '20 to 24', '25 to 29', '30 to 34', '35 to 39',
                 '40 to 44', '45 to 49', '50 to 54', '55 to 59', '60 to 64']:
        return 'Adults'
    elif age in ['65 to 69', '70 to 74', '75 to 79', '80 to 84', '85 to 89',
                 '90 to 94', 'Over 95']:
        return 'Seniors'
    else:
        return 'Unknown'

def map_road_class(value):
    if value in ['Major Arterial', 'Minor Arterial']:
        return 'Arterials'
    elif value in ['Expressway', 'Expressway Ramp']:
        return 'Expressways'
    elif value in ['Collector', 'Local', 'Laneway']:
```

```
        return 'Local Roads'
    else:
        return 'Other'

def map_traffctl(value):
    if value == 'No Control':
        return 'No Control'
    elif value in ['Traffic Signal', 'Traffic Gate']:
        return 'Automated Control'
    elif value in ['Stop Sign', 'Yield Sign', 'Pedestrian Crossover']:
        return 'Signage'
    elif value in ['Traffic Controller', 'School Guard', 'Police
Control']:
        return 'Human Control'
    else:
        return 'Other'

def map_visibility(value):
    if value == 'Clear':
        return 'Clear'
    elif value in ['Snow', 'Rain', 'Fog, Mist, Smoke, Dust', 'Drifting
Snow', 'Freezing Rain']:
        return 'Obstructed'
    else:
        return 'Other'

def map_light(value):
    if value in ['Daylight', 'Dusk', 'Dawn']:
        return 'Natural Light'
    elif value in ['Dark, artificial', 'Dusk, artificial', 'Dawn,
artificial', 'Daylight, artificial']:
        return 'Artificial Light'
    elif value == 'Dark':
        return 'Dark'
    else:
        return 'Other'

def map_rdsfcond(value):
    if value == 'Dry':
        return 'Clear'
    elif value in ['Wet', 'Slush', 'Loose Snow', 'Packed Snow',
'Spilled liquid', 'Loose Sand or Gravel']:
        return 'Wet'
    elif value == 'Ice':
        return 'Icy'
    else:
        return 'Other'

def map_impactype(value):
    if value in ['Approaching', 'Rear End', 'Sideswipe', 'Angle',
```

```
'Turning Movement']:
        return 'Vehicle Collisions'
    elif value in ['Pedestrian Collisions', 'Cyclist Collisions']:
        return 'Special Cases'
    elif value in ['SMV Other', 'SMV Unattended Vehicle']:
        return 'Static or Other Objects'
    else:
        return 'Other'

# Apply mapping functions
def apply_group_mapping(df):
    # Apply the mapping function to group
    df['INVAGE'] = df['INVAGE'].apply(group_age)
    df['ROAD_CLASS'] = df['ROAD_CLASS'].apply(map_road_class)
    df['TRAFFCTL'] = df['TRAFFCTL'].apply(map_traffctl)
    df['VISIBILITY'] = df['VISIBILITY'].apply(map_visibility)
    df['LIGHT'] = df['LIGHT'].apply(map_light)
    df['RDSFCOND'] = df['RDSFCOND'].apply(map_rdsfcond)
    df['IMPACTYPE'] = df['IMPACTYPE'].apply(map_impactype)
    return df

train_df = apply_group_mapping(train_df)
test_df = apply_group_mapping(test_df)

# Print the DataFrame to verify the grouping
print(train_df)
```

```
       OBJECTID ROAD_CLASS               DISTRICT
TRAFFCTL  \
0             1  Arterials  Toronto and East York         No Control

1             2  Arterials  Toronto and East York         No Control

2             3  Arterials  Toronto and East York         No Control

3             4  Arterials  Toronto and East York         No Control

4             5  Arterials  Toronto and East York         No Control

...         ...        ...                    ...                ...

14995     14996  Arterials            Scarborough  Automated Control

14996     14997  Arterials  Toronto and East York         No Control

14997     14998  Arterials  Toronto and East York         No Control

14998     14999  Arterials            Scarborough  Automated Control

14999     15000  Arterials            Scarborough  Automated Control
```

```
      VISIBILITY          LIGHT RDSFCOND           ACCLASS  \
0          Clear           Dark      Wet  Non-Fatal Injury
1          Clear           Dark      Wet  Non-Fatal Injury
2          Clear           Dark      Wet  Non-Fatal Injury
3          Clear           Dark      Wet  Non-Fatal Injury
4          Clear           Dark      Wet  Non-Fatal Injury
...          ...            ...      ...               ...
14995      Clear  Natural Light    Clear             Fatal
14996      Clear  Natural Light    Clear  Non-Fatal Injury
14997      Clear  Natural Light    Clear  Non-Fatal Injury
14998      Clear  Natural Light    Clear  Non-Fatal Injury
14999      Clear  Natural Light    Clear  Non-Fatal Injury

                IMPACTYPE   INVAGE  ... AUTOMOBILE MOTORCYCLE TRUCK  \
0      Vehicle Collisions   Adults  ...        Yes         No    No
1      Vehicle Collisions   Adults  ...        Yes         No    No
2      Vehicle Collisions   Adults  ...        Yes         No    No
3      Vehicle Collisions   Adults  ...        Yes         No    No
4      Vehicle Collisions   Adults  ...        Yes         No    No
...                   ...      ...  ...        ...        ...   ...
14995       Special Cases  Unknown  ...        Yes         No    No
14996       Special Cases   Adults  ...        Yes         No    No
14997       Special Cases   Adults  ...        Yes         No    No
14998       Special Cases   Adults  ...        Yes         No    No
14999       Special Cases   Adults  ...        Yes         No    No

      TRSN_CITY_VEH EMERG_VEH PASSENGER SPEEDING REDLIGHT ALCOHOL
DISABILITY
0                No        No       Yes      Yes       No     Yes
No
1                No        No       Yes      Yes       No     Yes
No
2                No        No       Yes      Yes       No     Yes
No
3                No        No       Yes      Yes       No     Yes
No
4                No        No       Yes      Yes       No     Yes
No
...             ...       ...       ...      ...      ...     ...
...
14995            No        No        No       No      Yes      No
No
14996            No        No        No       No       No      No
No
14997            No        No        No       No       No      No
No
14998            No        No        No       No       No      No
No
14999            No        No        No       No       No      No
```

```
No

[14584 rows x 21 columns]
```

We can see the new classes inside each feature:

```
count_func()

1. unique value number in OBJECTID : 14584

1. unique value in OBJECTID: [    1     2     3 ... 14998 14999 15000]


----------------------------------------------------------------------
2. unique value number in ROAD_CLASS : 4

2. unique value in ROAD_CLASS: ['Arterials' 'Local Roads' 'Other'
'Expressways']

----------------------------------------------------------------------
3. unique value number in DISTRICT : 4

3. unique value in DISTRICT: ['Toronto and East York' 'North York'
'Scarborough' 'Etobicoke York']

----------------------------------------------------------------------
4. unique value number in TRAFFCTL : 5

4. unique value in TRAFFCTL: ['No Control' 'Automated Control'
'Signage' 'Human Control' 'Other']

----------------------------------------------------------------------
5. unique value number in VISIBILITY : 3

5. unique value in VISIBILITY: ['Clear' 'Obstructed' 'Other']

----------------------------------------------------------------------
6. unique value number in LIGHT : 4

6. unique value in LIGHT: ['Dark' 'Artificial Light' 'Natural Light'
'Other']

----------------------------------------------------------------------
7. unique value number in RDSFCOND : 4

7. unique value in RDSFCOND: ['Wet' 'Clear' 'Icy' 'Other']

----------------------------------------------------------------------
8. unique value number in ACCLASS : 2

8. unique value in ACCLASS: ['Non-Fatal Injury' 'Fatal']
```

```
--------------------------------------------------------------------
9. unique value number in IMPACTYPE : 4

9. unique value in IMPACTYPE: ['Vehicle Collisions' 'Static or Other
Objects' 'Special Cases' 'Other']

--------------------------------------------------------------------
10. unique value number in INVAGE : 4

10. unique value in INVAGE: ['Adults' 'Unknown' 'Children' 'Seniors']

--------------------------------------------------------------------
11. unique value number in CYCLIST : 2

11. unique value in CYCLIST: ['No' 'Yes']

--------------------------------------------------------------------
12. unique value number in AUTOMOBILE : 2

12. unique value in AUTOMOBILE: ['Yes' 'No']

--------------------------------------------------------------------
13. unique value number in MOTORCYCLE : 2

13. unique value in MOTORCYCLE: ['No' 'Yes']

--------------------------------------------------------------------
14. unique value number in TRUCK : 2

14. unique value in TRUCK: ['No' 'Yes']

--------------------------------------------------------------------
15. unique value number in TRSN_CITY_VEH : 2

15. unique value in TRSN_CITY_VEH: ['No' 'Yes']

--------------------------------------------------------------------
16. unique value number in EMERG_VEH : 2

16. unique value in EMERG_VEH: ['No' 'Yes']

--------------------------------------------------------------------
17. unique value number in PASSENGER : 2

17. unique value in PASSENGER: ['Yes' 'No']

--------------------------------------------------------------------
18. unique value number in SPEEDING : 2

18. unique value in SPEEDING: ['Yes' 'No']
```

```
-------------------------------------------------------------------------
19. unique value number in REDLIGHT : 2

19. unique value in REDLIGHT: ['No' 'Yes']

-------------------------------------------------------------------------
20. unique value number in ALCOHOL : 2

20. unique value in ALCOHOL: ['Yes' 'No']

-------------------------------------------------------------------------
21. unique value number in DISABILITY : 2

21. unique value in DISABILITY: ['No' 'Yes']

-------------------------------------------------------------------------
```

Categorical to numerical mapping

Once the number of classes are reduced, we can start performing one-hot encoding for the nominal features and mapping to cero or one the binary categorical features. The other two features that will only be used to evaluate the model and not to train it, won't be transformed; this are OBJECT_ID' and 'ACCLASS' (target feature).

```
test_df.columns

Index(['OBJECTID', 'ROAD_CLASS', 'DISTRICT', 'TRAFFCTL', 'VISIBILITY',
'LIGHT',
       'RDSFCOND', 'IMPACTYPE', 'INVAGE', 'CYCLIST', 'AUTOMOBILE',
       'MOTORCYCLE', 'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH',
'PASSENGER',
       'SPEEDING', 'REDLIGHT', 'ALCOHOL', 'DISABILITY'],
      dtype='object')

categorical_columns = ['ROAD_CLASS', 'DISTRICT', 'TRAFFCTL',
'VISIBILITY', 'LIGHT', 'RDSFCOND', 'IMPACTYPE']
boolean_columns = ['CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK',
'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER',
       'SPEEDING', 'REDLIGHT', 'ALCOHOL', 'DISABILITY']
ordinal_columns = ['INVAGE', 'INJURY']

# Define the order for ordinal encoding
age_order = ['Unknown','Children', 'Adults', 'Seniors']

import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
```

```
from sklearn.preprocessing import FunctionTransformer
from sklearn.base import BaseEstimator, TransformerMixin

class YesNoToBinary(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Convert 'Yes' to 1 and 'No' to 0
        return np.where(X == 'Yes', 1, 0)

# Define the order for ordinal encoding
age_order = ['Unknown', 'Children', 'Adults', 'Seniors']


def cat_to_num(df, train):
    # define if the input df is the training or test to include the
"ACCLASS" feature
    static_labels = []

    if train == True:
        static_labels = ['OBJECT_ID','ACCLASS']
    else:
        static_labels = ['OBJECT_ID']

    # Define the transformers
    preprocessor = ColumnTransformer(
        transformers=[
            ('categorical', Pipeline(steps=[
                ('imputer', SimpleImputer(strategy='most_frequent')),
# Impute missing values with the most frequent value
                ('onehot', OneHotEncoder(handle_unknown='ignore'))  #
One-hot encode categorical variables
            ]), ['ROAD_CLASS', 'DISTRICT', 'TRAFFCTL', 'VISIBILITY',
'LIGHT', 'RDSFCOND', 'IMPACTYPE']),

            ('boolean', Pipeline(steps=[
                ('imputer', SimpleImputer(strategy='constant',
fill_value='No')),  # Replace missing values with 'No'
                ('binary', YesNoToBinary())  # Convert 'Yes'/'No' to
1/0
            ]), ['CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK',
'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER',
                'SPEEDING', 'REDLIGHT', 'ALCOHOL', 'DISABILITY']),

            ('ordinal', Pipeline(steps=[
                ('imputer', SimpleImputer(strategy='most_frequent')),
# Impute missing values with the most frequent value
                ('ordinal', OrdinalEncoder(categories=[age_order]))  #
Ordinal encode age
```

```
            ]), ['INVAGE'])
        ],
    remainder='passthrough'  # This ensures that columns not specified
in transformers will be included unchanged
    )

    # Fit and transform the data
    transformed_df = preprocessor.fit_transform(df)

    # Convert the result back to DataFrame if needed
    # Get feature names from OneHotEncoder
    onehot_feature_names =
preprocessor.named_transformers_['categorical'].named_steps['onehot'].
get_feature_names_out(['ROAD_CLASS', 'DISTRICT', 'TRAFFCTL',
'VISIBILITY', 'LIGHT', 'RDSFCOND', 'IMPACTYPE'])

    # Combine feature names
    feature_names = (list(onehot_feature_names) +
                    ['CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK',
'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER',
                    'SPEEDING', 'REDLIGHT', 'ALCOHOL', 'DISABILITY'] +
                    ['INVAGE'] +
                     static_labels)

    # Create DataFrame with new feature names
    transformed_df = pd.DataFrame(transformed_df,
columns=feature_names)

    # Print the transformed DataFrame
    print(transformed_df)

    return transformed_df

new_train_df = train_df.copy()
new_test_df = test_df.copy()

new_train_df = new_train_df.reindex(train_df.index)
new_test_df = new_test_df.reindex(test_df.index)

new_test_df = cat_to_num(new_test_df, train= False)
new_train_df = cat_to_num(new_train_df, train = True)
```

```
      ROAD_CLASS_Arterials  ROAD_CLASS_Expressways  ROAD_CLASS_Local
Roads  \
0                      1.0                     0.0
0.0
1                      1.0                     0.0
0.0
2                      1.0                     0.0
0.0
```

```
3                        1.0                        0.0
0.0
4                        1.0                        0.0
0.0
...                      ...                        ...
...
3951                     1.0                        0.0
0.0
3952                     1.0                        0.0
0.0
3953                     1.0                        0.0
0.0
3954                     0.0                        0.0
1.0
3955                     0.0                        0.0
1.0

      ROAD_CLASS_Other  DISTRICT_Etobicoke York  DISTRICT_North
York  \
0                  0.0                      0.0                     0.0

1                  0.0                      0.0                     0.0

2                  0.0                      0.0                     0.0

3                  0.0                      0.0                     0.0

4                  0.0                      1.0                     0.0

...                ...                      ...                     ...

3951               0.0                      0.0                     0.0

3952               0.0                      0.0                     0.0

3953               0.0                      0.0                     0.0

3954               0.0                      0.0                     0.0

3955               0.0                      0.0                     0.0


      DISTRICT_Scarborough  DISTRICT_Toronto and East York  \
0                      1.0                             0.0
1                      1.0                             0.0
2                      1.0                             0.0
3                      1.0                             0.0
4                      0.0                             0.0
...                    ...                             ...
3951                   0.0                             1.0
```

```
3952                   0.0                             1.0
3953                   0.0                             1.0
3954                   1.0                             0.0
3955                   1.0                             0.0

      TRAFFCTL_Automated Control  TRAFFCTL_Human Control  ...
TRUCK  \
0                            0.0                     0.0  ...    0.0

1                            0.0                     0.0  ...    0.0

2                            1.0                     0.0  ...    0.0

3                            1.0                     0.0  ...    0.0

4                            0.0                     0.0  ...    0.0

...                          ...                     ...  ...    ...

3951                         1.0                     0.0  ...    0.0

3952                         0.0                     0.0  ...    0.0

3953                         0.0                     0.0  ...    0.0

3954                         0.0                     0.0  ...    0.0

3955                         0.0                     0.0  ...    0.0


      TRSN_CITY_VEH  EMERG_VEH  PASSENGER  SPEEDING  REDLIGHT  ALCOHOL
\
0               0.0        0.0        0.0       0.0       0.0      0.0

1               0.0        0.0        0.0       0.0       0.0      0.0

2               0.0        0.0        0.0       0.0       0.0      0.0

3               0.0        0.0        0.0       0.0       0.0      0.0

4               0.0        0.0        0.0       0.0       0.0      0.0

...             ...        ...        ...       ...       ...      ...

3951            0.0        0.0        0.0       0.0       0.0      0.0

3952            0.0        0.0        0.0       0.0       0.0      0.0

3953            0.0        0.0        0.0       0.0       0.0      0.0

3954            0.0        0.0        0.0       0.0       0.0      0.0
```

```
3955            0.0         0.0         0.0        0.0        0.0      0.0


      DISABILITY  INVAGE  OBJECT_ID
0            0.0     0.0    15001.0
1            0.0     3.0    15002.0
2            0.0     0.0    15003.0
3            0.0     2.0    15004.0
4            0.0     2.0    15005.0
...          ...     ...        ...
3951         0.0     2.0    18953.0
3952         0.0     3.0    18954.0
3953         0.0     2.0    18955.0
3954         0.0     3.0    18956.0
3955         0.0     2.0    18957.0

[3956 rows x 40 columns]
      ROAD_CLASS_Arterials ROAD_CLASS_Expressways ROAD_CLASS_Local
Roads  \
0                      1.0                    0.0
0.0
1                      1.0                    0.0
0.0
2                      1.0                    0.0
0.0
3                      1.0                    0.0
0.0
4                      1.0                    0.0
0.0
...                    ...                    ...                    .
..
14579                  1.0                    0.0
0.0
14580                  1.0                    0.0
0.0
14581                  1.0                    0.0
0.0
14582                  1.0                    0.0
0.0
14583                  1.0                    0.0
0.0

      ROAD_CLASS_Other DISTRICT_Etobicoke York DISTRICT_North York  \
0                  0.0                     0.0                 0.0
1                  0.0                     0.0                 0.0
2                  0.0                     0.0                 0.0
3                  0.0                     0.0                 0.0
4                  0.0                     0.0                 0.0
...                ...                     ...                 ...
```

```
14579               0.0                        0.0                     0.0
14580               0.0                        0.0                     0.0
14581               0.0                        0.0                     0.0
14582               0.0                        0.0                     0.0
14583               0.0                        0.0                     0.0

      DISTRICT_Scarborough DISTRICT_Toronto and East York  \
0                      0.0                            1.0
1                      0.0                            1.0
2                      0.0                            1.0
3                      0.0                            1.0
4                      0.0                            1.0
...                    ...                            ...
14579                  1.0                            0.0
14580                  0.0                            1.0
14581                  0.0                            1.0
14582                  1.0                            0.0
14583                  1.0                            0.0

      TRAFFCTL_Automated Control TRAFFCTL_Human Control  ...
TRSN_CITY_VEH  \
0                            0.0                    0.0  ...
0
1                            0.0                    0.0  ...
0
2                            0.0                    0.0  ...
0
3                            0.0                    0.0  ...
0
4                            0.0                    0.0  ...
0
...                          ...                    ...  ...
...
14579                        1.0                    0.0  ...
0
14580                        0.0                    0.0  ...
0
14581                        0.0                    0.0  ...
0
14582                        1.0                    0.0  ...
0
14583                        1.0                    0.0  ...
0

      EMERG_VEH PASSENGER SPEEDING REDLIGHT ALCOHOL DISABILITY INVAGE
\
0             0         1        1        0       1          0    2.0

1             0         1        1        0       1          0    2.0
```

```
2                0         1         1         0        1           0     2.0

3                0         1         1         0        1           0     2.0

4                0         1         1         0        1           0     2.0

...            ...       ...       ...       ...      ...         ...     ...

14579            0         0         0         1        0           0     0.0

14580            0         0         0         0        0           0     2.0

14581            0         0         0         0        0           0     2.0

14582            0         0         0         0        0           0     2.0

14583            0         0         0         0        0           0     2.0


      OBJECT_ID           ACCLASS
0             1  Non-Fatal Injury
1             2  Non-Fatal Injury
2             3  Non-Fatal Injury
3             4  Non-Fatal Injury
4             5  Non-Fatal Injury
...         ...               ...
14579     14996             Fatal
14580     14997  Non-Fatal Injury
14581     14998  Non-Fatal Injury
14582     14999  Non-Fatal Injury
14583     15000  Non-Fatal Injury

[14584 rows x 42 columns]
```

```
new_test_df.shape
```

```
(3956, 40)
```

```
new_train_df.shape
```

```
(14584, 42)
```

Because there were no values for TRAFFCTL_Other in the validation dataframe, this feature was not created. Because of that, we will create the column and fill it with zeros to match the shape of the train data that will determine the input shape for the neural network model

```
# Get the columns in df1 but not in df2
columns_only_in_df1 = set(new_train_df.columns) -
set(new_test_df.columns)

print("Columns only in df1:", columns_only_in_df1)
```

```
Columns only in df1: {'TRAFFCTL_Other', 'ACCLASS'}
```

We added it in the same location that the train dataset so both have same index in the columns

```
# Find the position of column 'TRAFFCTL_Other'
position = new_train_df.columns.get_loc('TRAFFCTL_Other')
print(f"Position of column 'TRAFFCTL_Other': {position}")

Position of column 'TRAFFCTL_Other': 11

# Insert the new column at the same position as column
'TRAFFCTL_Other' in df
new_column = [0] * len(new_test_df)  # Create a list of zeros with
that length
new_test_df.insert(position, 'TRAFFCTL_Other', new_column)

new_test_df['TRAFFCTL_Other']

0       0
1       0
2       0
3       0
4       0
       ..
3951    0
3952    0
3953    0
3954    0
3955    0
Name: TRAFFCTL_Other, Length: 3956, dtype: int64

# Compare columns order between new_train_df and new_test_df
columns_match_1 =
new_train_df.drop(columns=['ACCLASS']).columns.equals(new_test_df.colu
mns)
print(f"Columns order match between df1 and df2: {columns_match_1}")

Columns order match between df1 and df2: True
```

Now we only have one column different that is the labels, that are not available for the validation test.

```
print(new_train_df.shape)

(14584, 42)

new_test_df.shape

(3956, 41)
```

Mapping

In this we made a function that maps the INVAGE column (age) to numerical indices so that we can use it in the predictions. We are using the map function for mapping it.

Making manual mapping in target feature so that we can use it in prediction (encoding)

```
new_train_df['ACCLASS'].unique()
```

```
array(['Non-Fatal Injury', 'Fatal'], dtype=object)
```

```
train_df['ACCLASS'].isnull().sum()
```

```
0
```

```
# Mapping the target feature
new_train_df['ACCLASS'] = new_train_df['ACCLASS'].map({'Non-Fatal
Injury': 0, 'Fatal': 1})
```

```
new_train_df['ACCLASS'].unique()
```

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

```
new_train_df['ACCLASS'].isnull().sum()
```

```
0
```

```
train_df['ACCLASS'].shape
```

```
(14584,)
```

```
new_train_df['ACCLASS'].shape
```

```
(14584,)
```

Train_test split

First we drop the OBJECT_ID' and 'ACCLASS' features to get only the features which will be used for the model to be train and save it in the X variable, and only save the target feature (labels) in the Y variable

```
X = new_train_df.drop(columns=['ACCLASS', 'OBJECT_ID'])
y = new_train_df['ACCLASS']
```

```
X.shape
```

```
(14584, 40)
```

As the validation set doesn't have the labels, we only drop the OBJECT_ID column

```
# same for val set
X_val = new_test_df.drop(columns=['OBJECT_ID'])

X_val.shape
```

```
(3956, 40)
```

```
X_train,X_test,y_train,y_test = train_test_split(X, y,
test_size=0.2,random_state=42)
```

Conversion to an acceptable format for TensorFlow

As Tensorflow doesn't accept pandas series as an input format for training the NNs, we transform the features into numpy arrays of type float32

```
# Convert to float32
X_train = np.array(X_train, dtype=np.float32)
y_train = np.array(y_train, dtype=np.float32)
X_test = np.array(X_test, dtype=np.float32)
y_test = np.array(y_test, dtype=np.float32)

X_val = np.array(X_val, dtype=np.float32)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (11667, 40)
y_train shape: (11667,)
X_test shape: (2917, 40)
y_test shape: (2917,)
```

```
print("X_val shape:", X_val.shape)
```

```
X_val shape: (3956, 40)
```

Feature importance

Before training the model we can determine the feature importance to finally choose the most important feature and reduce even more the dataset

```
# Train a Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Get feature importances
feature_importances = model.feature_importances_

# Create a DataFrame for better visualization
```

```python
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

print(importance_df)
```

```
                                Feature  Importance
39                               INVAGE    0.098590
34                            PASSENGER    0.086484
35                             SPEEDING    0.052042
31                                TRUCK    0.039811
10                  TRAFFCTL_No Control    0.038900
18                  LIGHT_Natural Light    0.037465
37                              ALCOHOL    0.037018
17                           LIGHT_Dark    0.036379
8            TRAFFCTL_Automated Control    0.033269
6                  DISTRICT_Scarborough    0.032624
36                             REDLIGHT    0.031509
16               LIGHT_Artificial Light    0.031027
4                DISTRICT_Etobicoke York   0.030019
5                   DISTRICT_North York    0.029972
32                         TRSN_CITY_VEH    0.029110
7       DISTRICT_Toronto and East York     0.029068
29                           AUTOMOBILE    0.024600
30                           MOTORCYCLE    0.024542
28                              CYCLIST    0.024346
25              IMPACTYPE_Special Cases    0.023994
20                       RDSFCOND_Clear    0.023949
23                         RDSFCOND_Wet    0.023278
27         IMPACTYPE_Vehicle Collisions    0.022968
12                     TRAFFCTL_Signage    0.022532
2                ROAD_CLASS_Local Roads    0.021444
0                 ROAD_CLASS_Arterials     0.021063
38                           DISABILITY    0.018118
13                     VISIBILITY_Clear    0.017823
14                VISIBILITY_Obstructed    0.016591
26   IMPACTYPE_Static or Other Objects     0.016492
22                       RDSFCOND_Other    0.008574
15                     VISIBILITY_Other    0.005755
24                      IMPACTYPE_Other    0.004458
1                ROAD_CLASS_Expressways    0.002269
9                TRAFFCTL_Human Control    0.001767
21                         RDSFCOND_Icy    0.001286
3                      ROAD_CLASS_Other    0.000440
33                            EMERG_VEH    0.000244
11                       TRAFFCTL_Other    0.000103
19                          LIGHT_Other    0.000075
```

```python
new_train_df.columns
```

```
Index(['ROAD_CLASS_Arterials', 'ROAD_CLASS_Expressways',
       'ROAD_CLASS_Local Roads', 'ROAD_CLASS_Other',
'DISTRICT_Etobicoke York',
       'DISTRICT_North York', 'DISTRICT_Scarborough',
       'DISTRICT_Toronto and East York', 'TRAFFCTL_Automated Control',
       'TRAFFCTL_Human Control', 'TRAFFCTL_No Control',
'TRAFFCTL_Other',
       'TRAFFCTL_Signage', 'VISIBILITY_Clear',
'VISIBILITY_Obstructed',
       'VISIBILITY_Other', 'LIGHT_Artificial Light', 'LIGHT_Dark',
       'LIGHT_Natural Light', 'LIGHT_Other', 'RDSFCOND_Clear',
'RDSFCOND_Icy',
       'RDSFCOND_Other', 'RDSFCOND_Wet', 'IMPACTYPE_Other',
       'IMPACTYPE_Special Cases', 'IMPACTYPE_Static or Other Objects',
       'IMPACTYPE_Vehicle Collisions', 'CYCLIST', 'AUTOMOBILE',
'MOTORCYCLE',
       'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING',
       'REDLIGHT', 'ALCOHOL', 'DISABILITY', 'INVAGE', 'OBJECT_ID',
'ACCLASS'],
      dtype='object')
```

```
type(importance_df)
```

```
pandas.core.frame.DataFrame
```

We only choose the first 17 features as they are the ones with a feature importance greater than 2,5%

```
important_features = list(importance_df['Feature'][:16])
important_features
```

```
['INVAGE',
 'PASSENGER',
 'SPEEDING',
 'TRUCK',
 'TRAFFCTL_No Control',
 'LIGHT_Natural Light',
 'ALCOHOL',
 'LIGHT_Dark',
 'TRAFFCTL_Automated Control',
 'DISTRICT_Scarborough',
 'REDLIGHT',
 'LIGHT_Artificial Light',
 'DISTRICT_Etobicoke York',
 'DISTRICT_North York',
 'TRSN_CITY_VEH',
 'DISTRICT_Toronto and East York']
```

```
important_features_idx = list(importance_df[:16].index)
```

```
X_train2 = X_train[:,important_features_idx]
X_test2 = X_test[:,important_features_idx]

important_features_idx

[39, 34, 35, 31, 10, 18, 37, 17, 8, 6, 36, 16, 4, 5, 32, 7]
```

Now we can validate that the shape of the new X contains only 17 features

```
X_val2 = X_val[:,important_features_idx]

X_train2.shape[1:]

(16,)

X_train2.shape

(11667, 16)

X_val2.shape[1:]

(16,)

X_val2.shape

(3956, 16)
```

Classification with various models

Definition of the Class SupervisedModels

In this class we define the attributes of the SupervisedMLModels instance and the required methods to perform training of the sk.learn models and evaluation, visualization and prediction for all the models including the tensorflow model

```
class SupervisedMLModels:
    # defining function for preprocessing, hyperparameter tuning
    def __init__(self, new_train_df, tensorflow_model=None):
        self.new_train_df = new_train_df
        self.tensorflow_model = tensorflow_model  # Add this line to
accept a TensorFlow model

        # making dictionary for models
        self.models = {
            'RandomForest': RandomForestClassifier(random_state=42),
            'GradientBoosting':
GradientBoostingClassifier(random_state=42),
            'SVM': SVC(probability=True, random_state=42),
```

```
            'LogisticRegression': LogisticRegression(random_state=42)
        }

        # define parameters for all model
        self.param_grids = {
            'RandomForest': {
                'n_estimators': [100, 200, 300],
                'max_depth': [10, 20, 30, None],
                'min_samples_split': [2, 5, 10],
                'min_samples_leaf': [1, 2, 4]
            },
            'GradientBoosting': {
                'n_estimators': [100, 200, 300],
                'learning_rate': [0.01, 0.1, 0.2],
                'max_depth': [3, 4, 5]
            },
            'SVM': {
                'C': [0.1, 1, 10],
                'gamma': ['scale', 0.1, 1, 10],
                'kernel': ['rbf']
            },
            'LogisticRegression': {
                'C': [0.1, 1, 10],
                'solver': ['liblinear', 'saga']
            }
        }
        self.best_models = {}  # empty dictionary to store values of
the best model

    # function to train and evaluate each model
    def train_and_evaluate(self):
        results = {}  # dictionary to store results of each model
evaluations

        for name, model in self.models.items():
            print(f'training {name}....')
            grid_search = GridSearchCV(estimator=model,
param_grid=self.param_grids[name], cv=3, n_jobs=-1, verbose=2)
            grid_search.fit(X_train2, y_train)
            best_model = grid_search.best_estimator_
            self.best_models[name] = best_model

            # doing model prediction
            y_pred = best_model.predict(X_test2)
            accuracy = accuracy_score(y_test, y_pred)
            print(f'{name} Accuracy after tuning parameter is:
{accuracy:.2f}')
            class_report = classification_report(y_test, y_pred,
zero_division=0)
```

```
            print(f'{name} Classification Report for model is :\
n{class_report}')
            conf_matrix = confusion_matrix(y_test, y_pred)
            print(f"confusion matrix for {name} is:\n{conf_matrix}")

            if name == 'GradientBoosting':
                y_prob = best_model.predict_proba(X_test2)[:, 1]
                fpr, tpr, _ = roc_curve(y_test, y_prob)
                auc_score = auc(fpr, tpr)
                print(f"Auc score {name} is :{auc_score:.2f}")

            results[name] = accuracy

        return results

    # defining a method for prediction using TensorFlow model
    def predict_with_tensorflow(self, X_test2, y_test):
        if self.tensorflow_model is not None:
            print("Predicting with TensorFlow model...")
            y_pred = (self.tensorflow_model.predict(X_test2) >
0.5).astype("int32")
            accuracy = accuracy_score(y_test, y_pred)
            print(f'TensorFlow Model Accuracy: {accuracy:.2f}')
            class_report = classification_report(y_test, y_pred,
zero_division=0)
            print(f'TensorFlow Classification Report:\
n{class_report}')
            conf_matrix = confusion_matrix(y_test, y_pred)
            print(f"Confusion Matrix for TensorFlow Model:\
n{conf_matrix}")

            # Assuming a binary classification for ROC curve and AUC
score
            if self.tensorflow_model.output_shape[-1] == 1:
                y_prob =
self.tensorflow_model.predict(X_test2).ravel()
                fpr, tpr, _ = roc_curve(y_test, y_prob)
                auc_score = auc(fpr, tpr)
                print(f"TensorFlow AUC Score: {auc_score:.2f}")
        else:
            print("No TensorFlow model provided.")

    def predict_and_save(self, X_val2, object_id_col,
output_file_prefix='submission'):
        predictions = {}

        for name, model in self.best_models.items():
            y_pred = model.predict(X_val2)
            predictions[name] = y_pred
```

```
            y_pred_final_df = pd.DataFrame({'OBJECTID': object_id_col,
'ACCLASS': y_pred})
            acclass_mapping_rev = {0: 'Non-Fatal Injury', 1: 'Fatal'}
            y_pred_final_df['ACCLASS'] =
y_pred_final_df['ACCLASS'].map(acclass_mapping_rev)

            output_file = f'{output_file_prefix}_{name}.csv'
            y_pred_final_df.to_csv(output_file, index=False)
            print(f'Submission file for {name} is created
successfully: {output_file}')

        # TensorFlow predictions
        if self.tensorflow_model is not None:
            y_pred = (self.tensorflow_model.predict(X_val2) >
0.5).astype("int32")
            y_pred_final_df = pd.DataFrame({'OBJECTID': object_id_col,
'ACCLASS': y_pred.flatten()})
            y_pred_final_df['ACCLASS'] =
y_pred_final_df['ACCLASS'].map(acclass_mapping_rev)

            output_file = f'{output_file_prefix}_TensorFlow.csv'
            y_pred_final_df.to_csv(output_file, index=False)
            print(f'Submission file for TensorFlow model is created
successfully: {output_file}')

    def plot_roc_curve(self):
        plt.figure(figsize=(12, 8))

        for name, model in self.best_models.items():
            if hasattr(model, 'predict_proba'):
                y_prob = model.predict_proba(X_test2)[:, 1]
            else:
                y_prob = model.decision_function(X_test2)

            fpr, tpr, _ = roc_curve(y_test, y_prob)
            auc_score = auc(fpr, tpr)
            plt.plot(fpr, tpr, label=f'{name} (AUC={auc_score:.2f})')

        # TensorFlow model ROC curve
        if self.tensorflow_model is not None:
            y_prob = self.tensorflow_model.predict(X_test2).ravel()
            fpr, tpr, _ = roc_curve(y_test, y_prob)
            auc_score = auc(fpr, tpr)
            plt.plot(fpr, tpr, label=f'TensorFlow
(AUC={auc_score:.2f})')

        plt.plot([0, 1], [0, 1], "k--")
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel("False Positive Rate")
```

```
        plt.ylabel("True Positive Rate")
        plt.title("Receiver Operating Characteristic (ROC) Curve")
        plt.legend(loc="lower right")
        plt.show()

    def plot_training_validation_curve(self, model_name):
        model = self.best_models[model_name]

        train_sizes, train_scores, validation_scores = learning_curve(
            model, X_train2, y_train, cv=3, n_jobs=-1,
train_sizes=np.linspace(0.1, 1.0, 10))

        plt.figure(figsize=(12, 6))
        plt.plot(train_sizes, np.mean(train_scores, axis=1), 'o-',
color='r', label='Training Score')
        plt.plot(train_sizes, np.mean(validation_scores, axis=1),
'o-', color='g', label='Validation Score')

        plt.xlabel("Training Size")
        plt.ylabel("Score")
        plt.title(f"Training and Validation Curves for {model_name}")
        plt.legend(loc='best')
        plt.show()



    def plot_tf_training_validation_curve(self, history):
        """ Plot training and validation curves for the TensorFlow
model.
        """
        # Extract data from history
        acc = history.history['accuracy']
        val_acc = history.history['val_accuracy']
        loss = history.history['loss']
        val_loss = history.history['val_loss']

        # Get number of epochs
        epochs = range(1, len(acc) + 1)

        # Plot training and validation accuracy
        plt.figure(figsize=(14, 5))
        plt.subplot(1, 2, 1)
        plt.plot(epochs, acc, 'bo-', label='Training accuracy')
        plt.plot(epochs, val_acc, 'r-', label='Validation accuracy')
        plt.title('Training and Validation Accuracy')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()

        # Plot training and validation loss
```

```
        plt.subplot(1, 2, 2)
        plt.plot(epochs, loss, 'bo-', label='Training loss')
        plt.plot(epochs, val_loss, 'r-', label='Validation loss')
        plt.title('Training and Validation Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()

        plt.tight_layout()
        plt.show()



    def plot_feature_importance(self, feature_names):
        """
        Plot feature importance for each model in the class.
        """
        # Loop through each model and compute feature importances
        for name, model in self.best_models.items():
            if hasattr(model, 'feature_importances_'):  # Tree-based
models
                importances = model.feature_importances_
            elif hasattr(model, 'coef_'):  # Linear models
                importances = np.abs(model.coef_).ravel()
            else:
                print(f"Feature importance is not available for
{name}")
                continue

            # Sort feature importances in descending order
            indices = np.argsort(importances)[::-1]

            # Plot the feature importances
            plt.figure(figsize=(12, 6))
            plt.title(f"Feature Importances for {name}")
            plt.bar(range(len(importances)), importances[indices],
align="center")
            plt.xticks(range(len(importances)), [feature_names[i] for
i in indices], rotation=90)
            plt.xlim([-1, len(importances)])
            plt.xlabel('Feature')
            plt.ylabel('Importance')
            plt.tight_layout()
            plt.show()
```

Creation of the Neural Networks model

In this case we use Neural networks to make the classification. For this, we use a simple model of dense layers with relu activation functions with some dropout layers to reduce the overfitting and improve model's performance.

The dense layers start with 1200 neurons in the first layer and start descending by half of the neurons in each of the subsequent layers.

The last one has a sigmoid activation function as we are working with a binary classification problem.

```
print("TensorFlow version:", tf.__version__)

def get_model():
    model = Sequential([
        keras.layers.Input(shape=X_train2.shape[1:]),
        keras.layers.Dense(1200, activation='relu',),
        keras.layers.Dropout(0.2),
        keras.layers.Dense(600, activation='relu'),
        keras.layers.Dense(300, activation='relu'),
        keras.layers.Dropout(0.2),
        keras.layers.Dense(1, activation='sigmoid')
    ])

    return model
```

```
TensorFlow version: 2.17.0
```

We initialize the model and compile it while defining the optimizer, the loss function and the evaluation metrics.

We will be using *ADAMAX* as it is an improved version of the ADAM optimizer. Overall, while ADAM is generally more popular and widely used, ADAMAX can be a beneficial alternative in scenarios where gradient stability and sparse data handling are critical factors.

Also, there might be situations where, empirically, ADAMAX performs better than ADAM on specific datasets or tasks. It's always a good idea to experiment with both optimizers and see which one works better for that particular use case.

In our case, we observed better performance using ADAMAX, so decided to keep it.

As for the loss functions, the best alternative for a binary classification problem is using Binary Cross Entropy.

And the accuracy will be our metric as it is the same metric that we are being evaluated in the Kaggle competition

```
'''best performance: tf_model.compile(optimizer='adamax',
              loss=keras.losses.BinaryCrossentropy(),
             metrics=['accuracy'])
```

```
                17 features
                current architecture
                30 epochs
                default learning rate
                '''

tf_model = get_model()

# Compile the model
tf_model.compile(optimizer='adamax',
                 loss=keras.losses.BinaryCrossentropy(),
                metrics=['accuracy'])

tf_model.summary()

Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ dense_4 (Dense)                 │ (None, 1200)           │
20,400 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout_2 (Dropout)             │ (None, 1200)           │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dense_5 (Dense)                 │ (None, 600)            │
720,600 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dense_6 (Dense)                 │ (None, 300)            │
180,300 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout_3 (Dropout)             │ (None, 300)            │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dense_7 (Dense)                 │ (None, 1)              │
301 │
└─────────────────────────────────┴────────────────────────┴──────────
─────┘

 Total params: 921,601 (3.52 MB)
```

```
 Trainable params: 921,601 (3.52 MB)

 Non-trainable params: 0 (0.00 B)
```

We save the history of the model to make further plot analysis of the accuracy and loss for both training and test sets

```
history = tf_model.fit(X_train2, y_train, epochs=30,
validation_data=(X_test2, y_test), verbose=1)

Epoch 1/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8653 - loss:
0.3954 - val_accuracy: 0.8711 - val_loss: 0.3659
Epoch 2/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8672 - loss:
0.3722 - val_accuracy: 0.8732 - val_loss: 0.3708
Epoch 3/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8693 - loss:
0.3677 - val_accuracy: 0.8738 - val_loss: 0.3579
Epoch 4/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8679 - loss:
0.3595 - val_accuracy: 0.8738 - val_loss: 0.3575
Epoch 5/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8714 - loss:
0.3569 - val_accuracy: 0.8690 - val_loss: 0.3589
Epoch 6/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8752 - loss:
0.3507 - val_accuracy: 0.8756 - val_loss: 0.3547
Epoch 7/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.8731 - loss:
0.3539 - val_accuracy: 0.8738 - val_loss: 0.3546
Epoch 8/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8774 - loss:
0.3430 - val_accuracy: 0.8759 - val_loss: 0.3556
Epoch 9/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8753 - loss:
0.3400 - val_accuracy: 0.8725 - val_loss: 0.3552
Epoch 10/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.8735 - loss:
0.3439 - val_accuracy: 0.8762 - val_loss: 0.3480
Epoch 11/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.8802 - loss:
0.3318 - val_accuracy: 0.8756 - val_loss: 0.3521
Epoch 12/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.8758 - loss:
0.3401 - val_accuracy: 0.8780 - val_loss: 0.3448
Epoch 13/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8759 - loss:
0.3311 - val_accuracy: 0.8780 - val_loss: 0.3469
```

```
Epoch 14/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8787 - loss:
0.3313 - val_accuracy: 0.8769 - val_loss: 0.3473
Epoch 15/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8806 - loss:
0.3265 - val_accuracy: 0.8793 - val_loss: 0.3473
Epoch 16/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8781 - loss:
0.3289 - val_accuracy: 0.8773 - val_loss: 0.3399
Epoch 17/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8765 - loss:
0.3303 - val_accuracy: 0.8804 - val_loss: 0.3364
Epoch 18/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8798 - loss:
0.3217 - val_accuracy: 0.8776 - val_loss: 0.3357
Epoch 19/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8863 - loss:
0.3113 - val_accuracy: 0.8800 - val_loss: 0.3398
Epoch 20/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8787 - loss:
0.3295 - val_accuracy: 0.8804 - val_loss: 0.3406
Epoch 21/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.8776 - loss:
0.3281 - val_accuracy: 0.8783 - val_loss: 0.3445
Epoch 22/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.8798 - loss:
0.3228 - val_accuracy: 0.8793 - val_loss: 0.3364
Epoch 23/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8827 - loss:
0.3139 - val_accuracy: 0.8797 - val_loss: 0.3396
Epoch 24/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8805 - loss:
0.3201 - val_accuracy: 0.8800 - val_loss: 0.3346
Epoch 25/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.8816 - loss:
0.3166 - val_accuracy: 0.8780 - val_loss: 0.3375
Epoch 26/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.8761 - loss:
0.3285 - val_accuracy: 0.8783 - val_loss: 0.3327
Epoch 27/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8797 - loss:
0.3165 - val_accuracy: 0.8776 - val_loss: 0.3384
Epoch 28/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.8862 - loss:
0.3118 - val_accuracy: 0.8769 - val_loss: 0.3355
Epoch 29/30
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8793 - loss:
0.3202 - val_accuracy: 0.8783 - val_loss: 0.3361
Epoch 30/30
```

```
365/365 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8811 - loss:
0.3157 - val_accuracy: 0.8769 - val_loss: 0.3374
```

We can also perform and individual evaluation of the trained model only with our test set

```
tf_model.evaluate(X_test2, y_test)
```

```
92/92 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8720 - loss:
0.3447

[0.33735784888267517, 0.8769283294677734]
```

Classification for all the models

```
# Initialize the class with your data and TensorFlow model
ml_models = SupervisedMLModels(new_train_df=X_train2,
tensorflow_model=tf_model)
```

Results of all model

Display the accuracy, classification report and confusion matrix for all the models

```
# Train and evaluate the scikit-learn models
results = ml_models.train_and_evaluate()
print("Training and evaluation results:", results)
```

```
training RandomForest....
Fitting 3 folds for each of 108 candidates, totalling 324 fits
RandomForest Accuracy after tuning parameter is: 0.88
RandomForest Classification Report for model is :
              precision    recall  f1-score   support

         0.0       0.88      0.99      0.93      2541
         1.0       0.73      0.10      0.17       376

    accuracy                           0.88      2917
   macro avg       0.81      0.55      0.55      2917
weighted avg       0.86      0.88      0.84      2917

confusion matrix for RandomForest is:
[[2528   13]
 [ 340   36]]
training GradientBoosting....
Fitting 3 folds for each of 27 candidates, totalling 81 fits
GradientBoosting Accuracy after tuning parameter is: 0.87
GradientBoosting Classification Report for model is :
              precision    recall  f1-score   support
```

```
         0.0       0.88      0.99      0.93      2541
         1.0       0.59      0.08      0.14       376

    accuracy                           0.87      2917
   macro avg       0.73      0.54      0.54      2917
weighted avg       0.84      0.87      0.83      2917

confusion matrix for GradientBoosting is:
[[2520   21]
 [ 346   30]]
Auc score GradientBoosting is :0.72
training SVM....
Fitting 3 folds for each of 12 candidates, totalling 36 fits
SVM Accuracy after tuning parameter is: 0.88
SVM Classification Report for model is :
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.94      2541
         1.0       0.86      0.08      0.15       376

    accuracy                           0.88      2917
   macro avg       0.87      0.54      0.54      2917
weighted avg       0.88      0.88      0.83      2917

confusion matrix for SVM is:
[[2536    5]
 [ 346   30]]
training LogisticRegression....
Fitting 3 folds for each of 6 candidates, totalling 18 fits
LogisticRegression Accuracy after tuning parameter is: 0.87
LogisticRegression Classification Report for model is :
              precision    recall  f1-score   support

         0.0       0.87      1.00      0.93      2541
         1.0       1.00      0.02      0.04       376

    accuracy                           0.87      2917
   macro avg       0.94      0.51      0.49      2917
weighted avg       0.89      0.87      0.82      2917

confusion matrix for LogisticRegression is:
[[2541    0]
 [ 368    8]]
Training and evaluation results: {'RandomForest': 0.8789852588275625,
'GradientBoosting': 0.8741858073363045, 'SVM': 0.8796708947548851,
'LogisticRegression': 0.8738429893726432}

# Make predictions using the TensorFlow model
ml_models.predict_with_tensorflow(X_test2, y_test)
```

```
Predicting with TensorFlow model...
92/92 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
TensorFlow Model Accuracy: 0.88
TensorFlow Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.99      0.93      2541
         1.0       0.61      0.12      0.21       376

    accuracy                           0.88      2917
   macro avg       0.75      0.56      0.57      2917
weighted avg       0.85      0.88      0.84      2917

Confusion Matrix for TensorFlow Model:
[[2511   30]
 [ 329   47]]
92/92 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
TensorFlow AUC Score: 0.74
```

Saving predictions in csv file

We save the labels in a variable to add it to validate the predictions

```
# tranfering value of object id to a variable
object_id_col = object_id_col['OBJECTID']
```

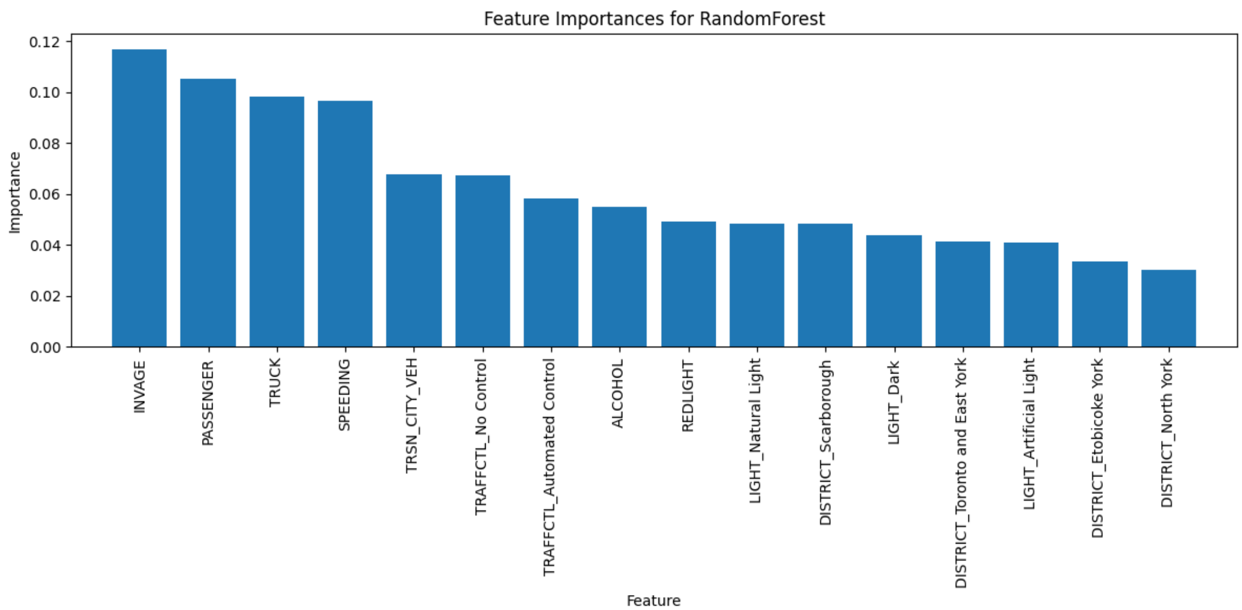
Then make predictions and save the file for the best-performing model

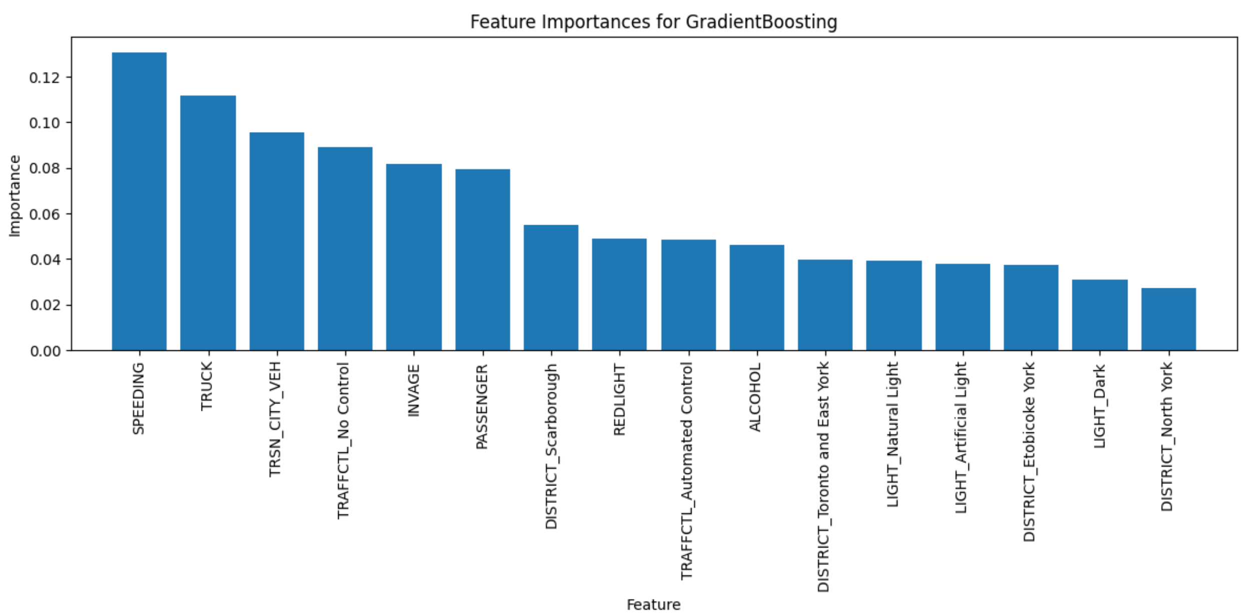
```
# Save predictions for validation set
ml_models.predict_and_save(X_val2=X_val2, object_id_col=object_id_col)
```

```
Submission file for RandomForest is created successfully:
submission_RandomForest.csv
Submission file for GradientBoosting is created successfully:
submission_GradientBoosting.csv
Submission file for SVM is created successfully: submission_SVM.csv
Submission file for LogisticRegression is created successfully:
submission_LogisticRegression.csv
124/124 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Submission file for TensorFlow model is created successfully:
submission_TensorFlow.csv
```

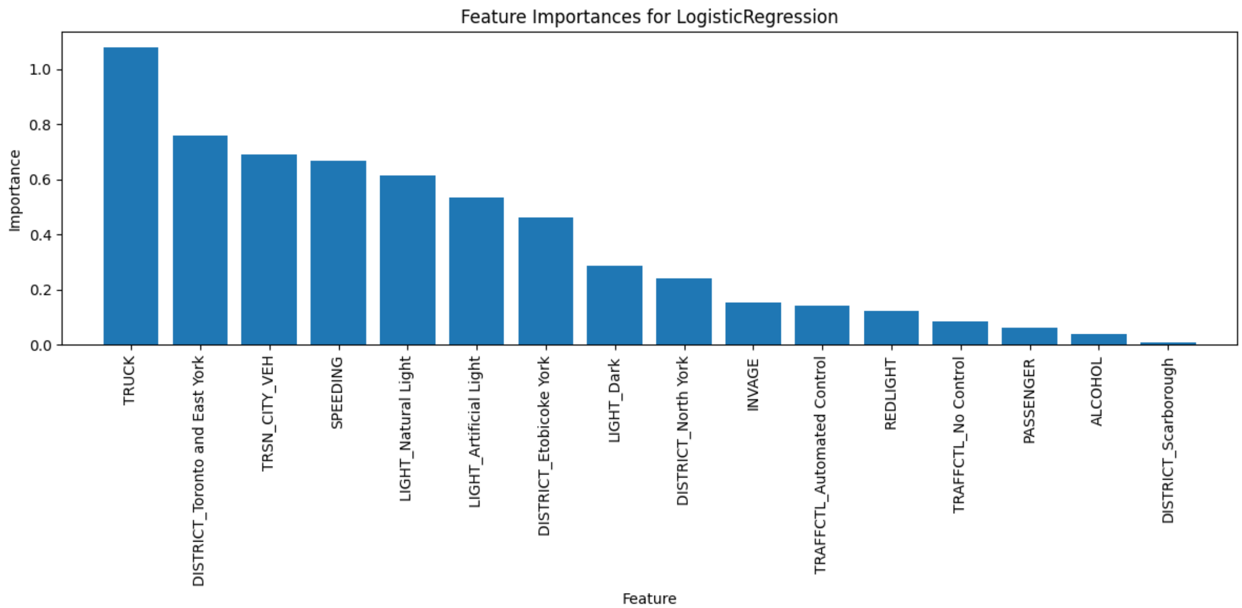
Feature importance for all models

```
# Plot feature importances
ml_models.plot_feature_importance(important_features)
```






```
Feature importance is not available for SVM
```

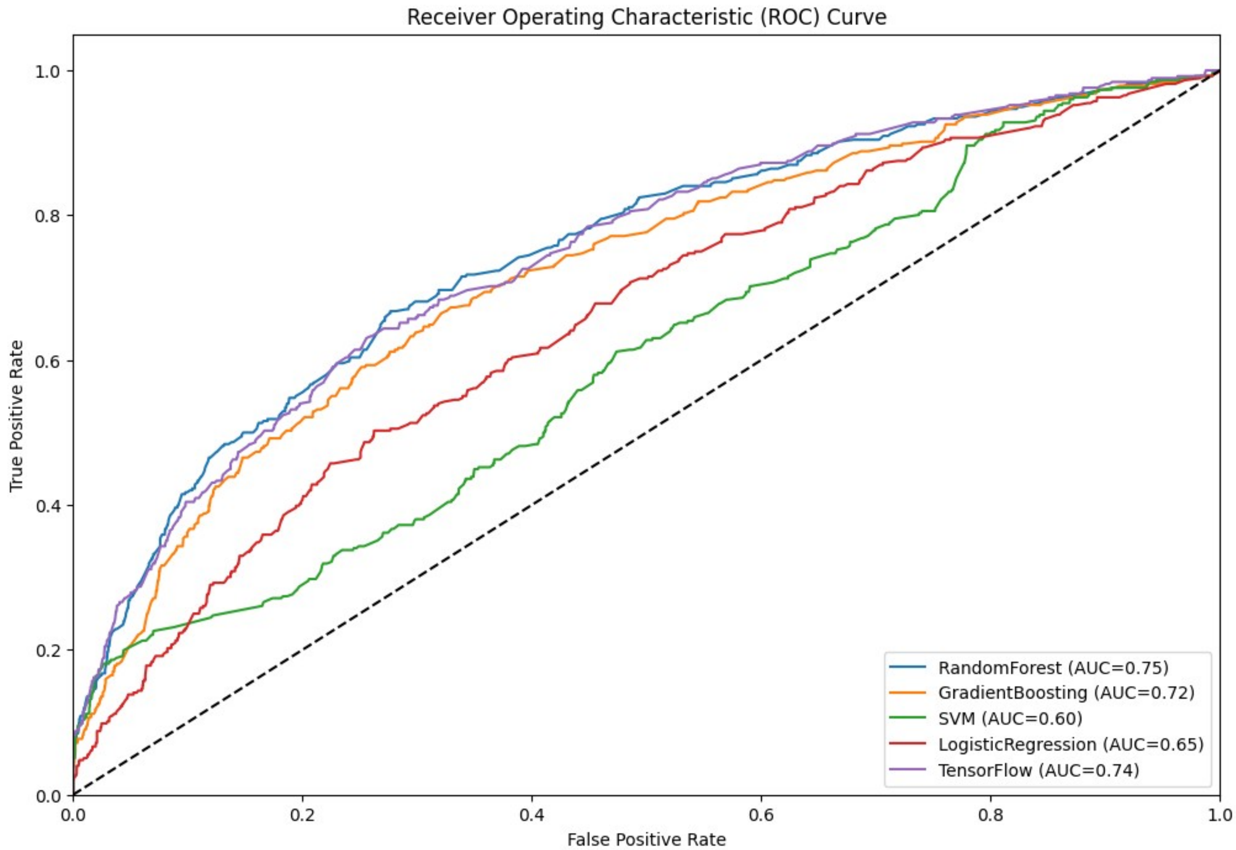


Feature importance is not built into neural networks (because of the complex, often non-linear interactions between features in neural networks) neither in SVM model because of the nature of its algorithm. For this, we won't be showing feature importance for this two algorithms

ROC curves for all model

```
# Plot ROC curves for all models
ml_models.plot_roc_curve()
```

```
92/92 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



The AUC score represents the overall ability of the model to discriminate between classes.

AUC Random forest:

This model is performing well as a score of 75% indicates that the model's ability to distinguish between the classes is good. It indicates that, on average, 75% of the time, the model gives a positive instance a higher ranking than to a negative instance. To check whether the this model is overfitting or not we will use the training and validation curve which we will explain in the following cells

AUC Gradient boosting:

the model performance is almost similar to random forest as auc score is near 72 but to know about its overfitting we will take a look at its training and validation curves.

AUC SVM:

this model is performance is not as goods as above two models as score is 60 its could be a good model to use but its less effective than other two models. Again to know its overfitting and underfitting we need the training and validation curves.

AUC Logistic regression:

This model performance is very poor as score is 65 and could be simple model to know about the underlying patterns in dataset. This could underfit the model but we will check it using training and validation curves.

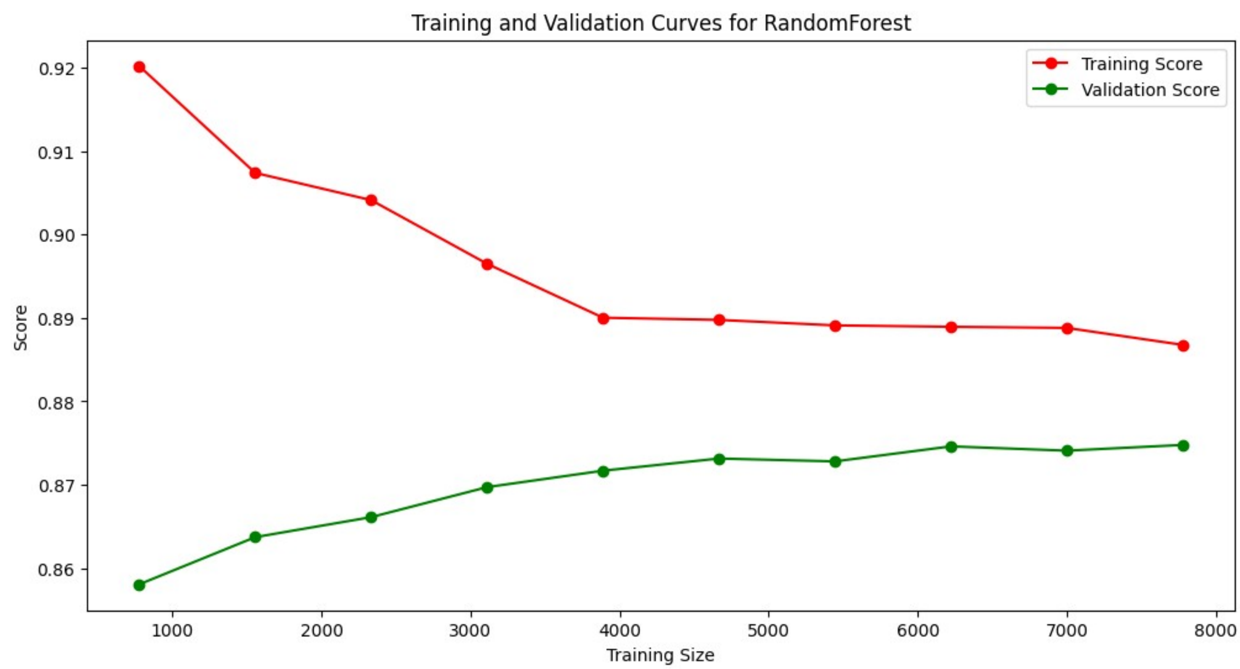
AUC Tensor flow :

this model is performing well as score is almost the same of the Random forest (75). Overfitting will be checked using the curves.

Training and validation curve

```
# calling training and validation curve method to plot the curve
print("curve for random forest :")
ml_models.plot_training_validation_curve('RandomForest')
```

```
curve for random forest :
```

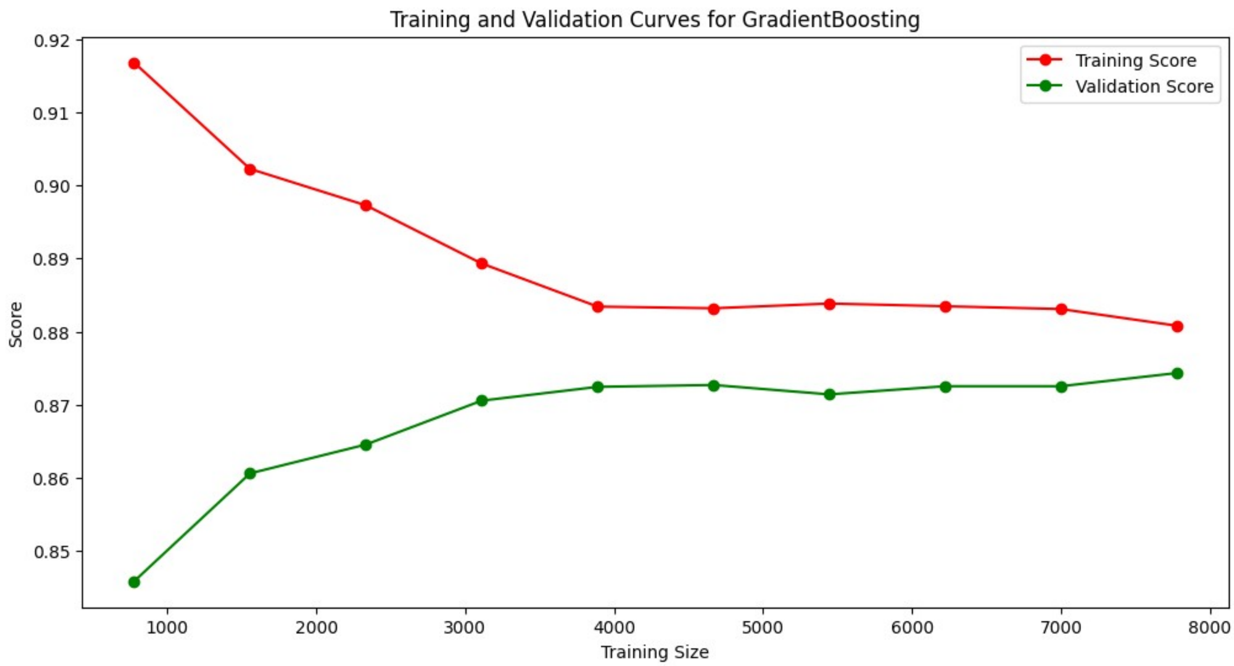


As we can see that there is covergence between the curve and they become parllel after 4000 training size and an appropiate model from our analysis.The model had caputered most of relevant information from dataset which was trained but not an complex enough to be an overfit model.

It also tells that hyper parameter tuning done by us is robust enough to make the balance between bias and variance.

```
# calling training and validation curve method to plot the curve
print("curve for random GradientBoosting :")
ml_models.plot_training_validation_curve('GradientBoosting')
```

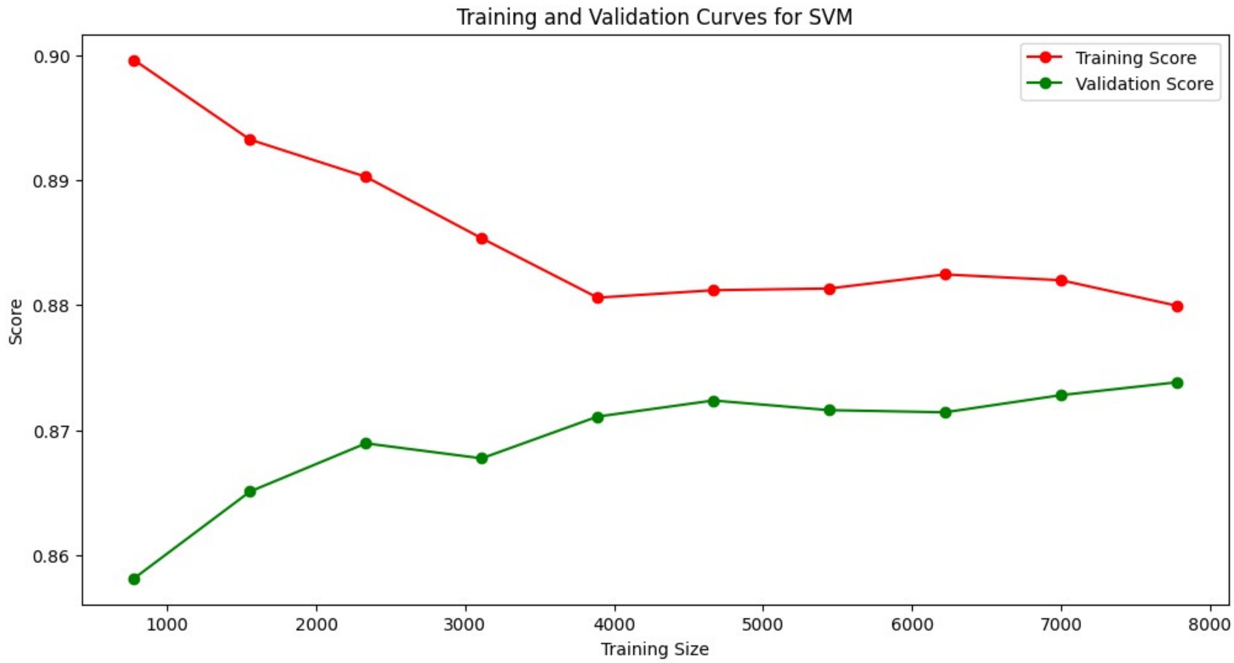
```
curve for random GradientBoosting :
```



The Similar results can be seen from above graph which means this is good fit model not an overfit model for our analysis.

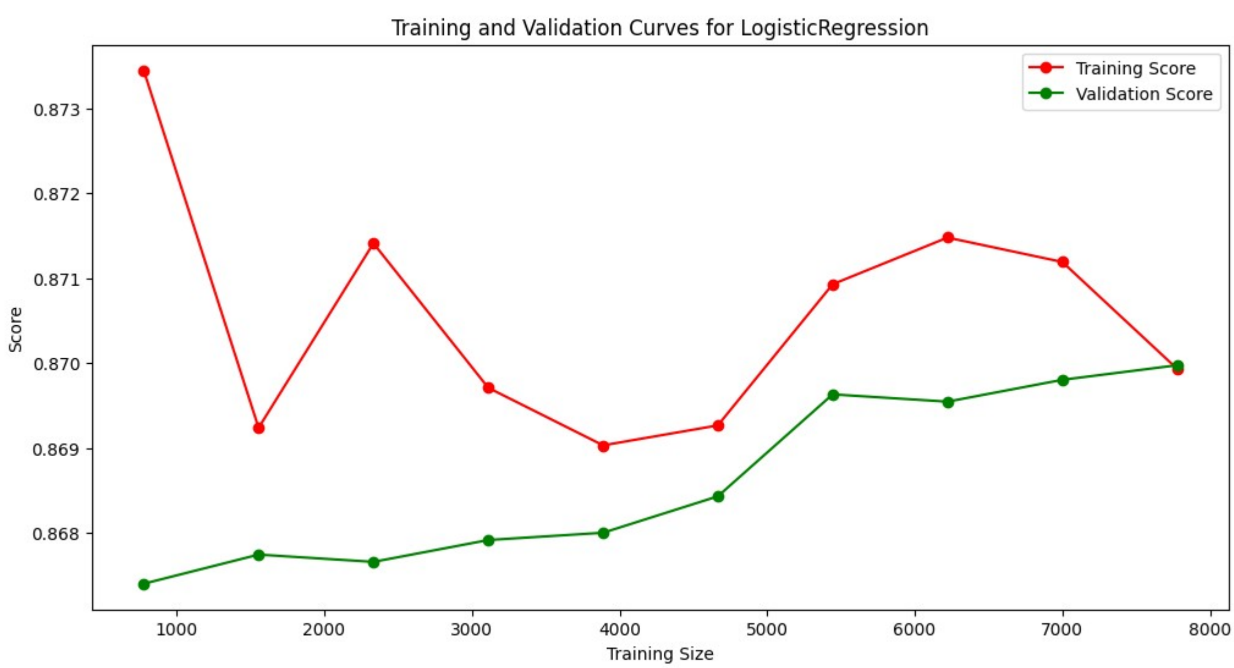
```
# calling training and validation curve method to plot the curve
print("curve for random SVM :")
ml_models.plot_training_validation_curve('SVM')
```

```
curve for random SVM :
```



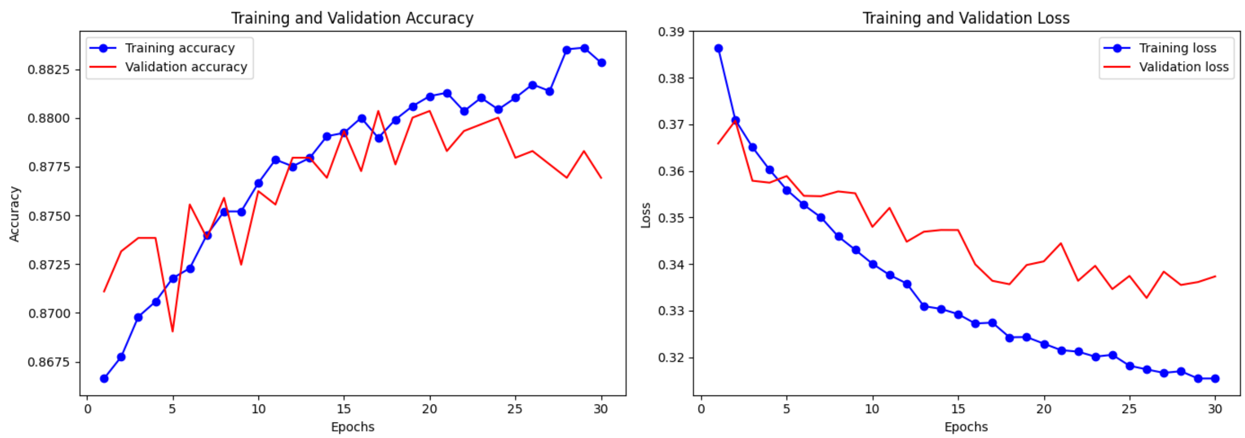
As we can see that there is significant decrease in values of training score and increase in validation score and both of curves come near at 0.88 but become parallel after that, so this model would also be good fit for our analysis.

```
# calling training and validation curve method to plot the curve
print("curve for random LogisticRegression :")
ml_models.plot_training_validation_curve('LogisticRegression')

curve for random LogisticRegression :
```



As we can see that the curves have some ups and down initially which shows that model struggle to stabilize learning path but after 4000 training size it get stabilize but 5000 there is increase in scores. After 7000 the training score starts to decrease and meet validation curve near 8000.This tell that there of need more hyper parameter tuning in order to achieve balance between the bias and variance in the dataset.

```
# Plot the training and validation curves for the TensorFlow model
ml_models.plot_tf_training_validation_curve(history)
```



From the above plot we can clearly see that after some time the training accuracy becomes higher than validation while the validation loss has become significant at 0.340 while training remain to decrease which means that after 25 epochs, the model will start to overfit.

Generating the model file

We can save our model in a SavedModel format as it follows:

```
# Save the model as an .h5 (deprecated)
#tf_model.save('my_model.h5')

# Save the entire model as a `.keras` zip archive.
tf_model.save('my_model.keras')
```

And loaded it again to make new predictions without having to train the model again with the following function:

```
new_model = tf.keras.models.load_model('my_model.keras')

# Show the model architecture
new_model.summary()

Model: "sequential_1"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━
━━━━━━┓
┃ Layer (type)                         ┃ Output Shape               ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━
━━━━━━┩
│ dense_4 (Dense)                      │ (None, 1200)               │
20,400 │
├──────────────────────────────────────┼────────────────────────────┼───────────
──────┤
│ dropout_2 (Dropout)                  │ (None, 1200)               │
0 │
├──────────────────────────────────────┼────────────────────────────┼───────────
──────┤
│ dense_5 (Dense)                      │ (None, 600)                │
720,600 │
├──────────────────────────────────────┼────────────────────────────┼───────────
──────┤
│ dense_6 (Dense)                      │ (None, 300)                │
180,300 │
├──────────────────────────────────────┼────────────────────────────┼───────────
──────┤
│ dropout_3 (Dropout)                  │ (None, 300)                │
0 │
├──────────────────────────────────────┼────────────────────────────┼───────────
──────┤
│ dense_7 (Dense)                      │ (None, 1)                  │
301 │
└──────────────────────────────────────┴────────────────────────────┴───────────
──────┘

 Total params: 2,764,805 (10.55 MB)

 Trainable params: 921,601 (3.52 MB)

 Non-trainable params: 0 (0.00 B)

 Optimizer params: 1,843,204 (7.03 MB)
```