```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
data= pd.read_csv("/content/dataset_traffic_accident_prediction1.csv")
```

```
data.head(15)
```

	Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicles
0	Rainy	City Road	Morning	1.0	100.0	5.C
1	Clear	Rural Road	Night	NaN	120.0	3.C
2	Rainy	Highway	Evening	1.0	60.0	4.C
3	Clear	City Road	Afternoon	2.0	60.0	3.C
4	Rainy	Highway	Morning	1.0	195.0	11.C
5	Clear	Rural Road	Night	0.0	120.0	3.C
6	Foggy	Highway	Afternoon	0.0	60.0	4.C
7	Rainy	City Road	Afternoon	0.0	60.0	4.C
8	Stormy	Highway	Morning	1.0	60.0	2.C
9	Rainy	City Road	Afternoon	2.0	30.0	2.C
10	Foggy	NaN	Evening	NaN	60.0	2.C
11	Clear	Mountain Road	Night	2.0	100.0	5.C
12	NaN	Rural Road	Afternoon	0.0	60.0	4.C
13	Rainy	City Road	Night	0.0	30.0	1.C
14	Clear	Rural Road	Morning	0.0	NaN	1.C

```
data.drop_duplicates(inplace=True)
```

```
data
```

	Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicle
0	Rainy	City Road	Morning	1.0	100.0	5.
1	Clear	Rural Road	Night	NaN	120.0	3.
2	Rainy	Highway	Evening	1.0	60.0	4.
3	Clear	City Road	Afternoon	2.0	60.0	3.
4	Rainy	Highway	Morning	1.0	195.0	11.
...	...	...	...	...	...	.
835	Clear	Highway	Night	2.0	30.0	4.
836	Rainy	Rural Road	Evening	2.0	60.0	4.
837	Foggy	Highway	Evening	NaN	30.0	4.
838	Foggy	Highway	Afternoon	2.0	60.0	3.
839	Clear	Highway	Afternoon	1.0	60.0	4.

826 rows × 14 columns

```
data.columns
```

```
Index(['Weather', 'Road_Type', 'Time_of_Day', 'Traffic_Density', 'Speed_Limit',
       'Number_of_Vehicles', 'Driver_Alcohol', 'Accident_Severity',
       'Road_Condition', 'Vehicle_Type', 'Driver_Age', 'Driver_Experience',
       'Road_Light_Condition', 'Accident'],
      dtype='object')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 826 entries, 0 to 839
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Weather               784 non-null    object 
 1   Road_Type             784 non-null    object 
 2   Time_of_Day           785 non-null    object 
 3   Traffic_Density       784 non-null    float64
 4   Speed_Limit           784 non-null    float64
 5   Number_of_Vehicles    784 non-null    float64
 6   Driver_Alcohol        784 non-null    float64
 7   Accident_Severity     785 non-null    object 
 8   Road_Condition        784 non-null    object 
 9   Vehicle_Type          784 non-null    object 
 10  Driver_Age            784 non-null    float64
 11  Driver_Experience     784 non-null    float64
```

```
 12  Road_Light_Condition  784 non-null    object
 13  Accident              784 non-null    float64
dtypes: float64(7), object(7)
memory usage: 96.8+ KB
```

```
#finding missing values
data.isnull().sum()
```

	0
Weather	42
Road_Type	42
Time_of_Day	41
Traffic_Density	42
Speed_Limit	42
Number_of_Vehicles	42
Driver_Alcohol	42
Accident_Severity	41
Road_Condition	42
Vehicle_Type	42
Driver_Age	42
Driver_Experience	42
Road_Light_Condition	42
Accident	42

dtype: int64

```
data.duplicated().sum()
```

```
np.int64(0)
```

```
#dropping missing values
data.dropna()
```

Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicle:
Rainy	Highway	Evening	1.0	60.0	4.(
Clear	City Road	Afternoon	2.0	60.0	3.(
Rainy	Highway	Morning	1.0	195.0	11.(
Foggy	Highway	Afternoon	0.0	60.0	4.(
Rainy	City Road	Afternoon	0.0	60.0	4.(
...	...	...	...	...	..
Clear	Highway	Morning	1.0	100.0	2.(
Clear	Highway	Night	2.0	30.0	4.(
Rainy	Rural Road	Evening	2.0	60.0	4.(
Foggy	Highway	Afternoon	2.0	60.0	3.(
Clear	Highway	Afternoon	1.0	60.0	4.(

›ws × 14 columns

```
#filling the null values
data["Traffic_Density"].fillna(data["Traffic_Density"].mean(), inplace=True)
data["Speed_Limit"].fillna(data["Speed_Limit"].mean(), inplace=True)
data["Number_of_Vehicles"].fillna(data["Number_of_Vehicles"].mean(), inplace=True)
data["Driver_Alcohol"].fillna(data["Driver_Alcohol"].mean(), inplace=True)
data["Accident_Severity"].fillna(data["Accident_Severity"].mode()[0], inplace=True)
data["Road_Condition"].fillna(data["Road_Condition"].mode()[0], inplace=True)
data["Vehicle_Type"].fillna(data["Vehicle_Type"].mode()[0], inplace=True)
data["Driver_Age"].fillna(data["Driver_Age"].mean(), inplace=True)
data["Driver_Experience"].fillna(data["Driver_Experience"].mean(), inplace=True)
data["Road_Light_Condition"].fillna(data["Road_Light_Condition"].mode()[0], inplace=True)
data["Accident"].fillna(data["Accident"].mean(), inplace=True)
data["Weather"].fillna(data["Weather"].mode()[0], inplace=True)
data["Road_Type"].fillna(data["Road_Type"].mode()[0], inplace=True)
data["Time_of_Day"].fillna(data["Time_of_Day"].mode()[0], inplace=True)
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho


  data["Road_Condition"].fillna(data["Road_Condition"].mode()[0], inplace=True)
<ipython-input-24-230c89790859>:8: FutureWarning: A value is trying to be set on a
The behavior will change in pandas 3.0. This inplace method will never work becaus

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho


  data["Vehicle_Type"].fillna(data["Vehicle_Type"].mode()[0], inplace=True)
<ipython-input-24-230c89790859>:9: FutureWarning: A value is trying to be set on a
The behavior will change in pandas 3.0. This inplace method will never work becaus

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho


  data["Driver_Age"].fillna(data["Driver_Age"].mean(), inplace=True)
<ipython-input-24-230c89790859>:10: FutureWarning: A value is trying to be set on
The behavior will change in pandas 3.0. This inplace method will never work becaus

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho


  data["Driver_Experience"].fillna(data["Driver_Experience"].mean(), inplace=True)
<ipython-input-24-230c89790859>:11: FutureWarning: A value is trying to be set on
The behavior will change in pandas 3.0. This inplace method will never work becaus

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho


  data["Road_Light_Condition"].fillna(data["Road_Light_Condition"].mode()[0], inpl
<ipython-input-24-230c89790859>:12: FutureWarning: A value is trying to be set on
The behavior will change in pandas 3.0. This inplace method will never work becaus

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho


  data["Accident"].fillna(data["Accident"].mean(), inplace=True)
<ipython-input-24-230c89790859>:15: FutureWarning: A value is trying to be set on
The behavior will change in pandas 3.0. This inplace method will never work becaus

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.metho


  data["Time_of_Day"].fillna(data["Time_of_Day"].mode()[0], inplace=True)
```

```
data
```

	Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicle
0	Rainy	City Road	Morning	1.000000	100.0	5.
1	Clear	Rural Road	Night	0.998724	120.0	3.
2	Rainy	Highway	Evening	1.000000	60.0	4.
3	Clear	City Road	Afternoon	2.000000	60.0	3.
4	Rainy	Highway	Morning	1.000000	195.0	11.
...	...	...	...	...	...	.
835	Clear	Highway	Night	2.000000	30.0	4.
836	Rainy	Rural Road	Evening	2.000000	60.0	4.
837	Foggy	Highway	Evening	0.998724	30.0	4.
838	Foggy	Highway	Afternoon	2.000000	60.0	3.
839	Clear	Highway	Afternoon	1.000000	60.0	4.

826 rows × 14 columns

```
data.isnull().sum()
```

	0
Weather	0
Road_Type	0
Time_of_Day	0
Traffic_Density	0
Speed_Limit	0
Number_of_Vehicles	0
Driver_Alcohol	0
Accident_Severity	0
Road_Condition	0
Vehicle_Type	0
Driver_Age	0
Driver_Experience	0
Road_Light_Condition	0
Accident	0

dtype: int64

```
#categorical data
data["Road_Light_Condition"].fillna(data["Road_Light_Condition"].mode()[0], inplace = Tru
```

```
<ipython-input-26-8de2997c5d88>:2: FutureWarning: A value is trying to be set on a cc
The behavior will change in pandas 3.0. This inplace method will never work because t

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({


  data["Road_Light_Condition"].fillna(data["Road_Light_Condition"].mode()[0], inplace
```

```
data
```

	Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicle
0	Rainy	City Road	Morning	1.000000	100.0	5.
1	Clear	Rural Road	Night	0.998724	120.0	3.
2	Rainy	Highway	Evening	1.000000	60.0	4.
3	Clear	City Road	Afternoon	2.000000	60.0	3.
4	Rainy	Highway	Morning	1.000000	195.0	11.
...	...	...	...	...	...	.
835	Clear	Highway	Night	2.000000	30.0	4.
836	Rainy	Rural Road	Evening	2.000000	60.0	4.
837	Foggy	Highway	Evening	0.998724	30.0	4.
838	Foggy	Highway	Afternoon	2.000000	60.0	3.
839	Clear	Highway	Afternoon	1.000000	60.0	4.

826 rows × 14 columns

```
#removing duplicates
data.drop_duplicates(inplace=True)
```

```
data
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = data.copy()
data_scaled[["Traffic_Density","Speed_Limit"]] =scaler.fit_transform(data[["Traffic_Densi
data_scaled
```

	Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicle
0	Rainy	City Road	Morning	0.001672	0.918165	5.
1	Clear	Rural Road	Night	0.000002	1.555111	3.
2	Rainy	Highway	Evening	0.001672	-0.355726	4.
3	Clear	City Road	Afternoon	1.311322	-0.355726	3.
4	Rainy	Highway	Morning	0.001672	3.943656	11.
...	...	...	...	...	...	.
835	Clear	Highway	Night	1.311322	-1.311144	4.
836	Rainy	Rural Road	Evening	1.311322	-0.355726	4.
837	Foggy	Highway	Evening	0.000002	-1.311144	4.
838	Foggy	Highway	Afternoon	1.311322	-0.355726	3.
839	Clear	Highway	Afternoon	0.001672	-0.355726	4.

825 rows × 14 columns

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

data_scaled[["Traffic_Density","Speed_Limit"]] =scaler.fit_transform(data[["Traffic_Densi
data_scaled
```

	Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicle
0	Rainy	City Road	Morning	0.500000	0.382514	5.
1	Clear	Rural Road	Night	0.499362	0.491803	3.
2	Rainy	Highway	Evening	0.500000	0.163934	4.
3	Clear	City Road	Afternoon	1.000000	0.163934	3.
4	Rainy	Highway	Morning	0.500000	0.901639	11.
...	...	...	...	...	...	.
835	Clear	Highway	Night	1.000000	0.000000	4.
836	Rainy	Rural Road	Evening	1.000000	0.163934	4.
837	Foggy	Highway	Evening	0.499362	0.000000	4.
838	Foggy	Highway	Afternoon	1.000000	0.163934	3.
839	Clear	Highway	Afternoon	0.500000	0.163934	4.

825 rows × 14 columns

```
data_encoded = pd.get_dummies(data, columns=["Road_Light_Condition"],drop_first=True)
print(data_encoded)
```

```
    Weather   Road_Type Time_of_Day  Traffic_Density  Speed_Limit  \
0     Rainy   City Road     Morning         1.000000        100.0
1     Clear  Rural Road       Night         0.998724        120.0
2     Rainy     Highway     Evening         1.000000         60.0
3     Clear   City Road   Afternoon         2.000000         60.0
4     Rainy     Highway     Morning         1.000000        195.0
..      ...         ...         ...              ...          ...
835   Clear     Highway       Night         2.000000         30.0
836   Rainy  Rural Road     Evening         2.000000         60.0
837   Foggy     Highway     Evening         0.998724         30.0
838   Foggy     Highway   Afternoon         2.000000         60.0
839   Clear     Highway   Afternoon         1.000000         60.0

     Number_of_Vehicles  Driver_Alcohol Accident_Severity      Road_Condition  \
0                   5.0             0.0               Low                 Wet
1                   3.0             0.0          Moderate                 Wet
2                   4.0             0.0               Low                 Icy
3                   3.0             0.0               Low  Under Construction
4                  11.0             0.0               Low                 Dry
..                  ...             ...               ...                 ...
835                 4.0             0.0               Low                 Dry
836                 4.0             0.0               Low                 Dry
837                 4.0             1.0              High                 Dry
838                 3.0             0.0               Low                 Dry
839                 4.0             0.0               Low                 Dry

    Vehicle_Type  Driver_Age  Driver_Experience  Accident  \
```

```
0           Car   51.000000              48.0  0.000000
1         Truck   49.000000              43.0  0.000000
2           Car   54.000000              52.0  0.000000
3           Bus   34.000000              31.0  0.000000
4           Car   62.000000              55.0  1.000000
..          ...         ...               ...       ...
835         Car   23.000000              15.0  0.000000
836  Motorcycle   52.000000              46.0  1.000000
837         Car   43.153061              34.0  0.298469
838         Car   25.000000              19.0  0.000000
839  Motorcycle   29.000000              21.0  0.000000

     Road_Light_Condition_Daylight  Road_Light_Condition_No Light
0                            False                          False
1                            False                          False
2                            False                          False
3                             True                          False
4                            False                          False
..                             ...                            ...
835                           True                          False
836                           True                          False
837                          False                          False
838                          False                          False
839                          False                          False

[825 rows x 15 columns]
```

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data["Road_Light_Condition"] = encoder.fit_transform(data["Road_Light_Condition"])


def performance_category(Speed_Limit):
    if Speed_Limit >= 10:
      return "High"
    elif Speed_Limit >= 5:
      return "Medium"
    else:
      return "Low"

data["Performance"] = data["Speed_Limit"].apply(performance_category)
print(data)
```

```
    Weather   Road_Type Time_of_Day  Traffic_Density  Speed_Limit  \
0     Rainy   City Road     Morning         1.000000        100.0
1     Clear  Rural Road       Night         0.998724        120.0
2     Rainy     Highway     Evening         1.000000         60.0
3     Clear   City Road   Afternoon         2.000000         60.0
4     Rainy     Highway     Morning         1.000000        195.0
..      ...         ...         ...              ...          ...
835   Clear     Highway       Night         2.000000         30.0
836   Rainy  Rural Road     Evening         2.000000         60.0
837   Foggy     Highway     Evening         0.998724         30.0
838   Foggy     Highway   Afternoon         2.000000         60.0
839   Clear     Highway   Afternoon         1.000000         60.0

     Number_of_Vehicles  Driver_Alcohol Accident_Severity      Road_Condition  \
0                   5.0             0.0               Low                 Wet
```

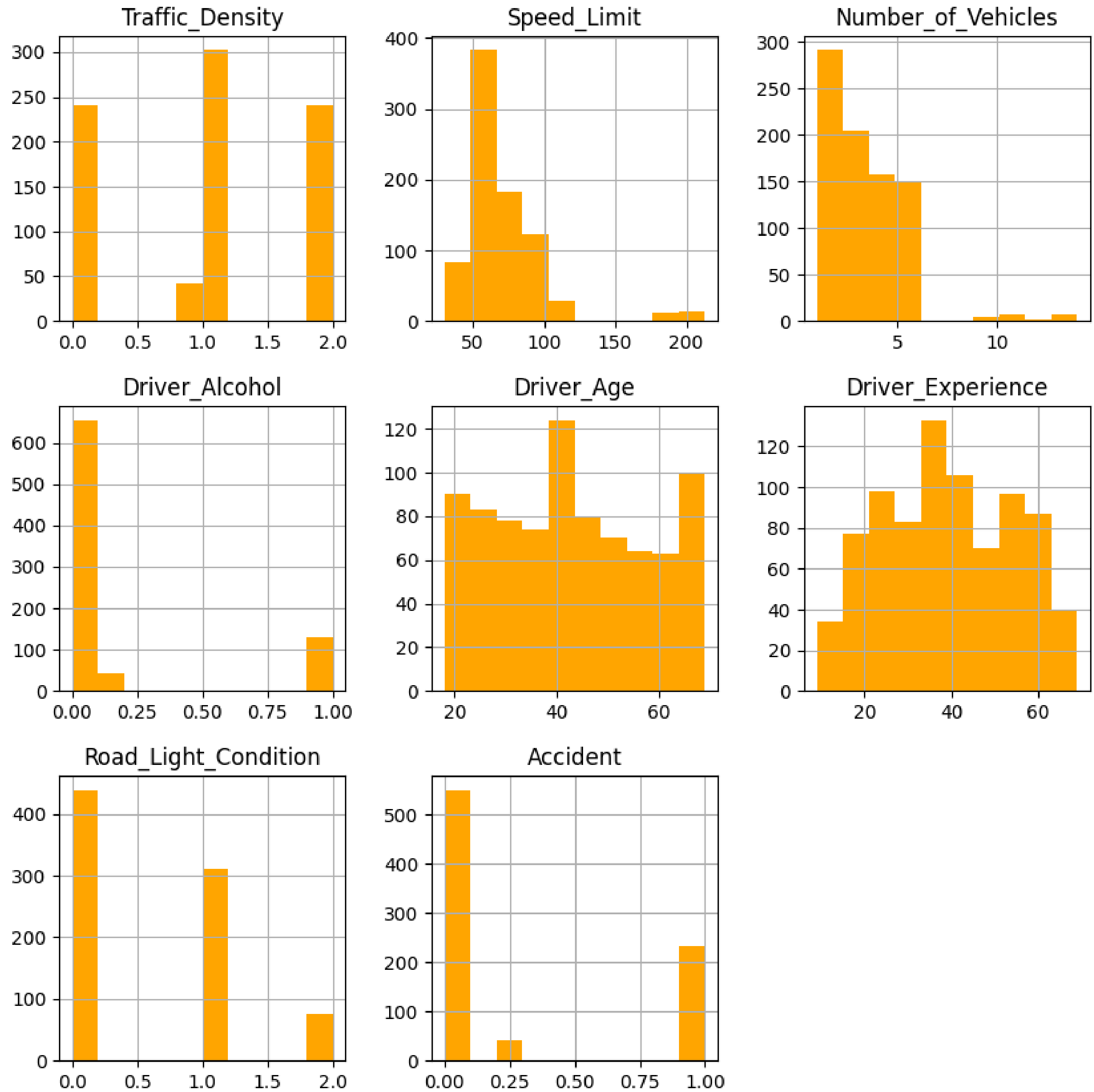
```
1                 3.0             0.0         Moderate                 Wet
2                 4.0             0.0              Low                 Icy
3                 3.0             0.0              Low  Under Construction
4                11.0             0.0              Low                 Dry
..                ...             ...              ...                 ...
835               4.0             0.0              Low                 Dry
836               4.0             0.0              Low                 Dry
837               4.0             1.0             High                 Dry
838               3.0             0.0              Low                 Dry
839               4.0             0.0              Low                 Dry

    Vehicle_Type  Driver_Age  Driver_Experience  Road_Light_Condition  \
0            Car   51.000000               48.0                     0
1          Truck   49.000000               43.0                     0
2            Car   54.000000               52.0                     0
3            Bus   34.000000               31.0                     1
4            Car   62.000000               55.0                     0
..           ...         ...                ...                   ...
835          Car   23.000000               15.0                     1
836   Motorcycle   52.000000               46.0                     1
837          Car   43.153061               34.0                     0
838          Car   25.000000               19.0                     0
839   Motorcycle   29.000000               21.0                     0

     Accident Performance
0    0.000000        High
1    0.000000        High
2    0.000000        High
3    0.000000        High
4    1.000000        High
..        ...         ...
835  0.000000        High
836  1.000000        High
837  0.298469        High
838  0.000000        High
839  0.000000        High

[825 rows x 15 columns]
```

```
data
```

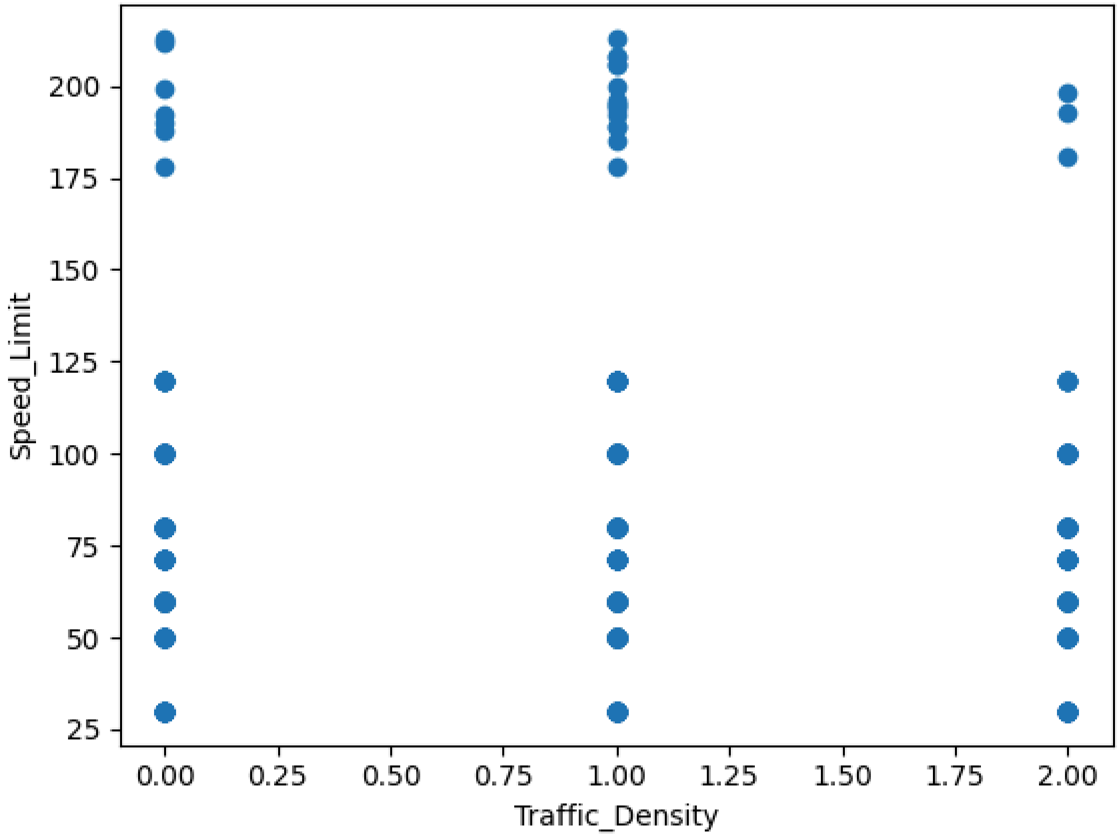
	Weather	Road_Type	Time_of_Day	Traffic_Density	Speed_Limit	Number_of_Vehicle
0	Rainy	City Road	Morning	1.000000	100.0	5.
1	Clear	Rural Road	Night	0.998724	120.0	3.
2	Rainy	Highway	Evening	1.000000	60.0	4.
3	Clear	City Road	Afternoon	2.000000	60.0	3.
4	Rainy	Highway	Morning	1.000000	195.0	11.
...	...	...	...	...	...	.
835	Clear	Highway	Night	2.000000	30.0	4.
836	Rainy	Rural Road	Evening	2.000000	60.0	4.
837	Foggy	Highway	Evening	0.998724	30.0	4.
838	Foggy	Highway	Afternoon	2.000000	60.0	3.
839	Clear	Highway	Afternoon	1.000000	60.0	4.

825 rows × 15 columns

```
#univariate analysis
data.hist(figsize=(10,10), color="Orange")
plt.show()
```




```
#scatter chart
plt.scatter(data["Traffic_Density"], data["Speed_Limit"])
plt.xlabel("Traffic_Density")
plt.ylabel("Speed_Limit")
plt.show()
```




```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data["Road_Light_Condition"] = encoder.fit_transform(data["Road_Light_Condition"])
```

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data["Accident"] = encoder.fit_transform(data["Accident"])
data["Accident_Severity"]=encoder.fit_transform(data["Accident_Severity"])
data["Road_Condition"]=encoder.fit_transform(data["Road_Condition"])
data["Vehicle_Type"]=encoder.fit_transform(data["Vehicle_Type"])
data["Driver_Alcohol"]=encoder.fit_transform(data["Driver_Alcohol"])
data["Road_Type"]=encoder.fit_transform(data["Road_Type"])
data["Time_of_Day"]=encoder.fit_transform(data["Time_of_Day"])
data["Weather"]=encoder.fit_transform(data["Weather"])
data["Performance"]=encoder.fit_transform(data["Performance"])
```

```
#model building
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
#random forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
#select target data
X = data.drop("Accident", axis=1)
```

```
y = data["Accident"]


x_test,x_train,y_test,y_train = train_test_split(X,y,test_size=0.2,random_state=42)


#logistic regression
model = LogisticRegression()
model.fit(x_train,y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: Conver
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
▾ LogisticRegression
LogisticRegression()
```

```
#prediction
y_pred = model.predict(x_test)
print("y_pred",y_pred)
```

```
y_pred [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 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 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 0 0 0 0 0 0 0 0 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 0 0 0 0 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
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 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 0 0 0 0
 0 0 0 0 0 0 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 0 0 0 0 0 0
 0 0 0 0 0 0 0 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 0 0 0 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 0 0 0 0 0 0 0 0 1 0 0 0 0
 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 0 0 0 0 0 0]
```

```
#random forest classifier
model = RandomForestClassifier()
model.fit(x_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
#prediction
y_pred_random = model.predict(x_test)
print("y_pred_random",y_pred_random)
```

```
y_pred_random [0 0 0 0 0 0 2 0 0 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0
 0 0 0 0 0 0 0 0 2 2 0 0 0 0 2 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0
 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 2 2 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 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 0 0 0 0 0 0 0 2 0 0 0 2
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 0 0 2 0 0 0 0
 0 0 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 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 0 0 0 2 0 0]
```

```
#Evaluation logistic regression
accuracy = accuracy_score(y_test,y_pred)
print("accuracy",accuracy)
classification_rep = classification_report(y_test,y_pred)
print("classification_rep",classification_rep)
confusion_mat = confusion_matrix(y_test,y_pred)
print("confusion_mat",confusion_mat)
```

```
accuracy 0.6454545454545455
classification_rep               precision    recall  f1-score   support

           0       0.66      0.96      0.78       438
           1       0.06      0.03      0.04        37
           2       0.57      0.02      0.04       185

    accuracy                           0.65       660
   macro avg       0.43      0.34      0.29       660
weighted avg       0.60      0.65      0.53       660

confusion_mat [[421  14   3]
 [ 36   1   0]
 [178   3   4]]
```

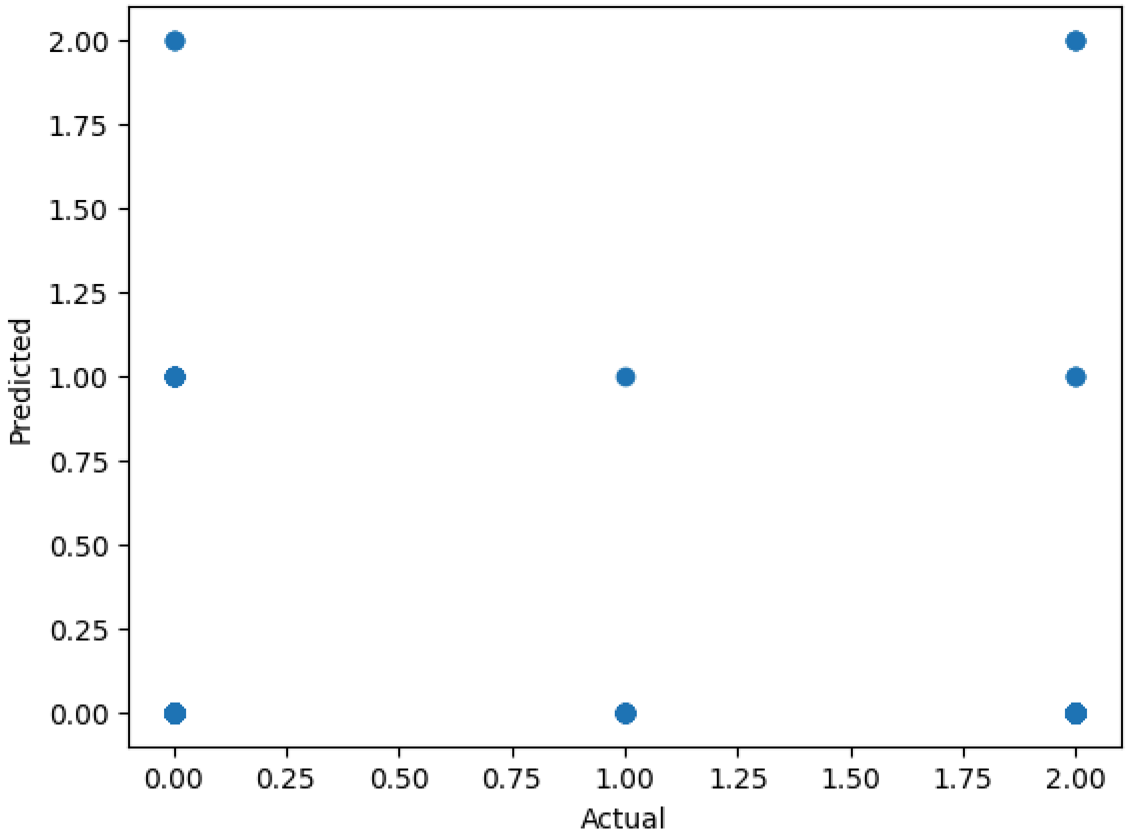
```
#evaluation random forest
accuracy_random = accuracy_score(y_test,y_pred_random)
print("accuracy_random",accuracy_random)
classification_rep_random = classification_report(y_test,y_pred_random)
print("classification_rep_random",classification_rep_random)
confusion_mat_random = confusion_matrix(y_test,y_pred_random)
print("confusion_mat_random",confusion_mat_random)
```

```
accuracy_random 0.646969696969697
classification_rep_random               precision    recall  f1-score   support

           0       0.67      0.95      0.78       438
           1       0.00      0.00      0.00        37
           2       0.29      0.05      0.08       185

    accuracy                           0.65       660
   macro avg       0.32      0.33      0.29       660
weighted avg       0.52      0.65      0.54       660

confusion_mat_random [[418   1  19]
 [ 34   0   3]
 [176   0   9]]
```

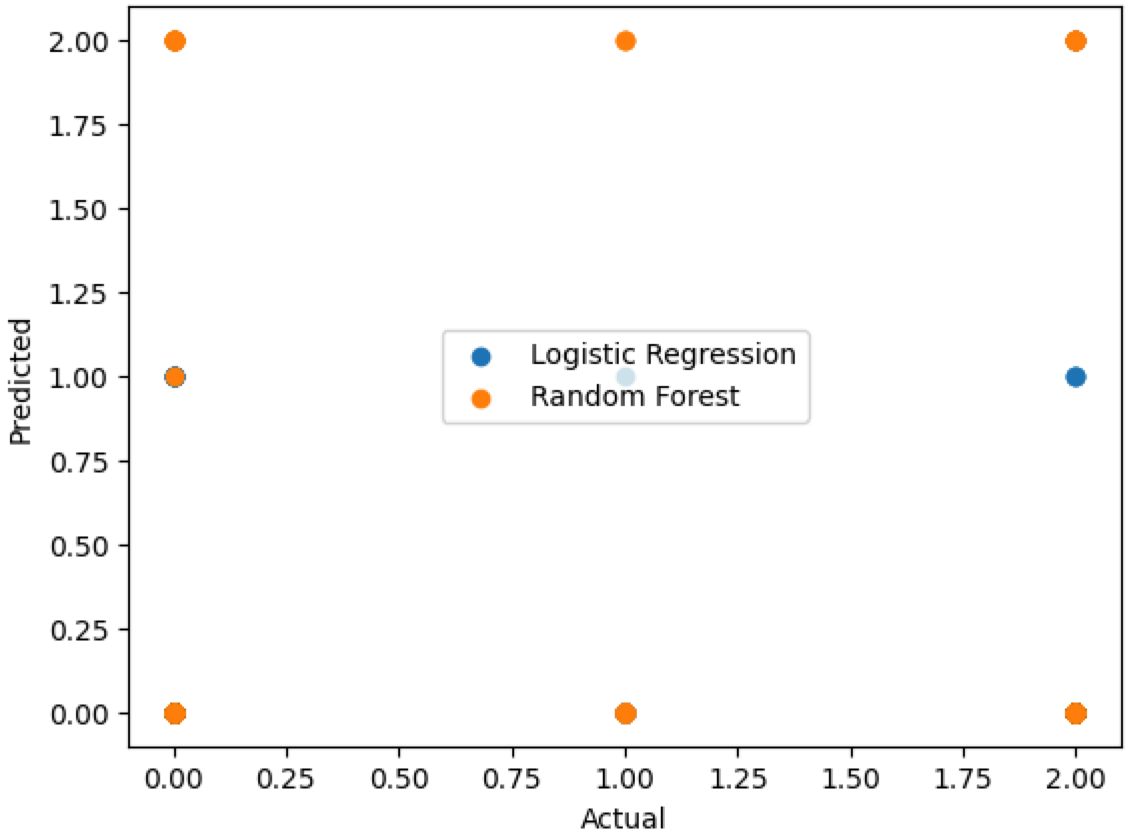
```
#prediction analysis
prediction_analysis = pd.DataFrame({"Actual":y_test,"Predicted":y_pred})
print(prediction_analysis)
```

```
     Actual  Predicted
239       0          0
701       0          0
655       2          0
345       0          0
302       2          0
..      ...        ...
71        2          0
106       0          0
272       0          0
441       0          0
102       0          0

[660 rows x 2 columns]
```

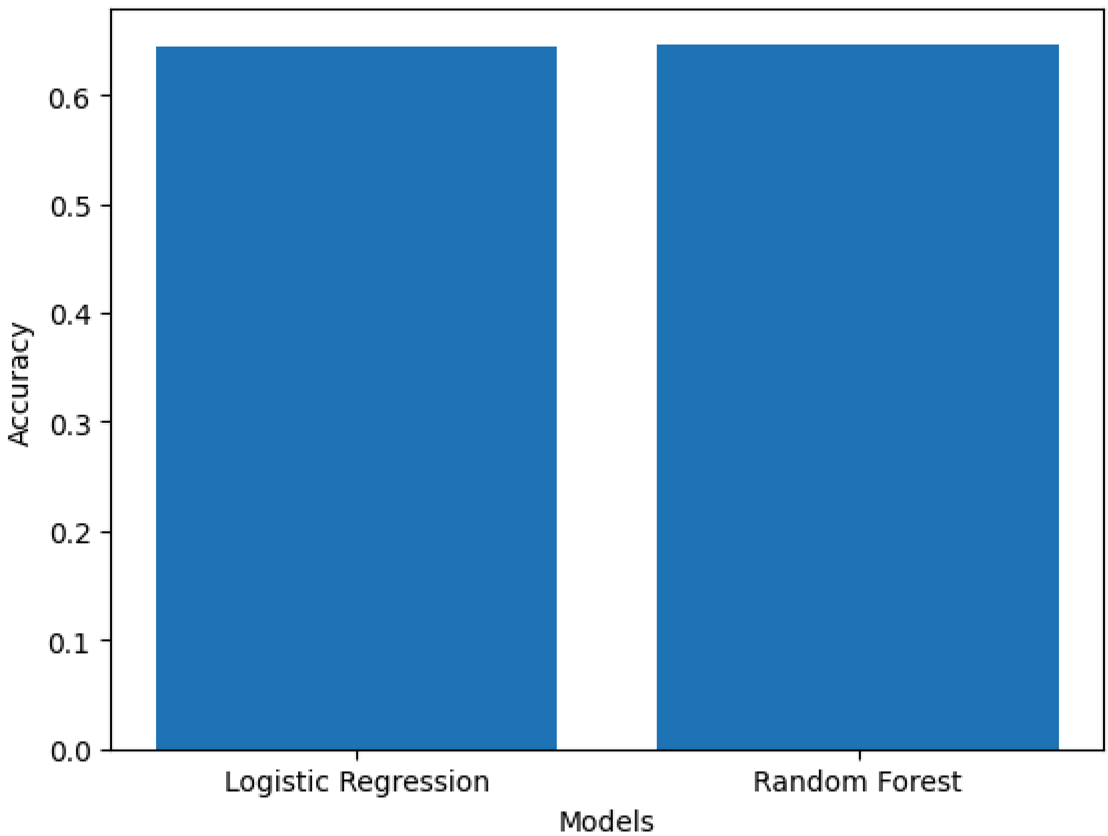
```
#visualization prediction and actual value
plt.scatter(y_test,y_pred)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



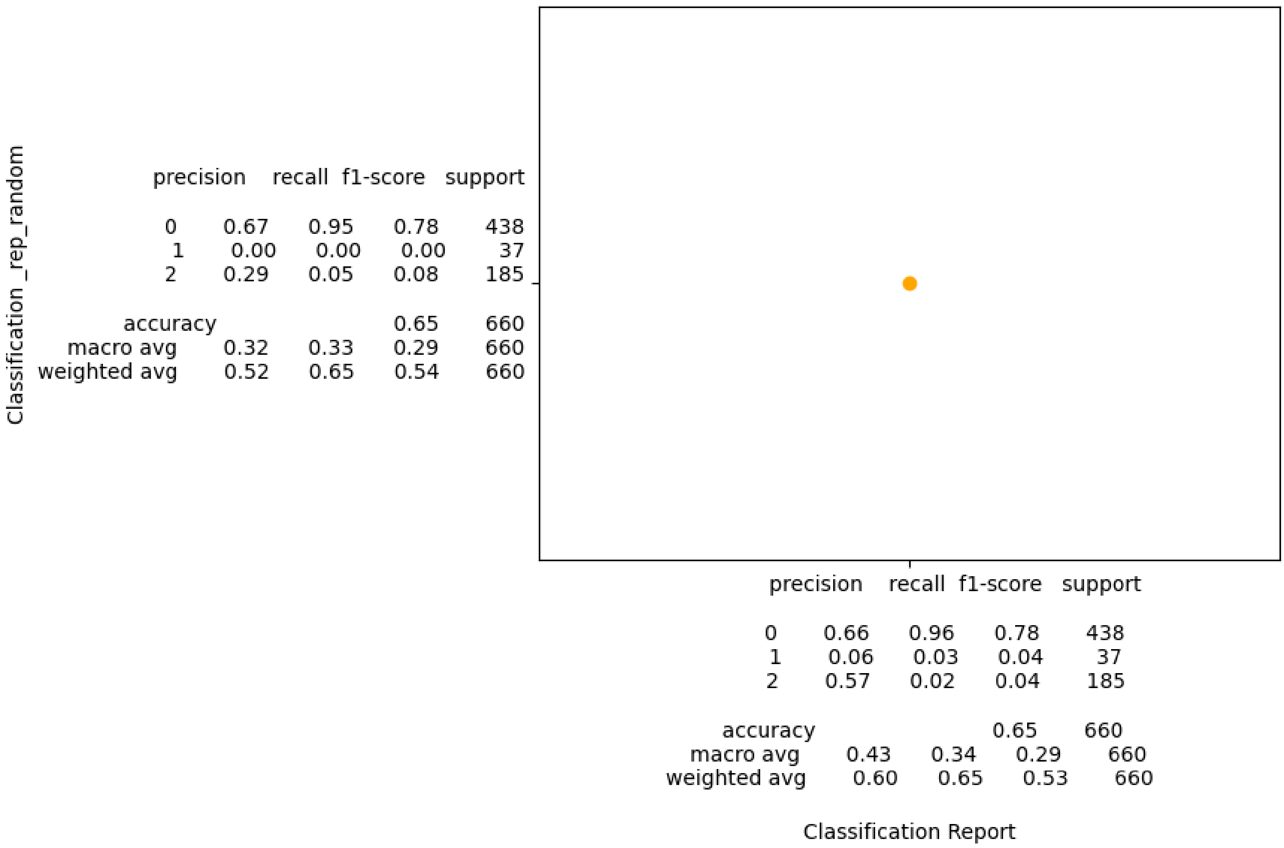

```
#visualization an two models
plt.scatter(y_test,y_pred,label="Logistic Regression")
plt.scatter(y_test,y_pred_random,label="Random Forest")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.legend()
plt.show()
```



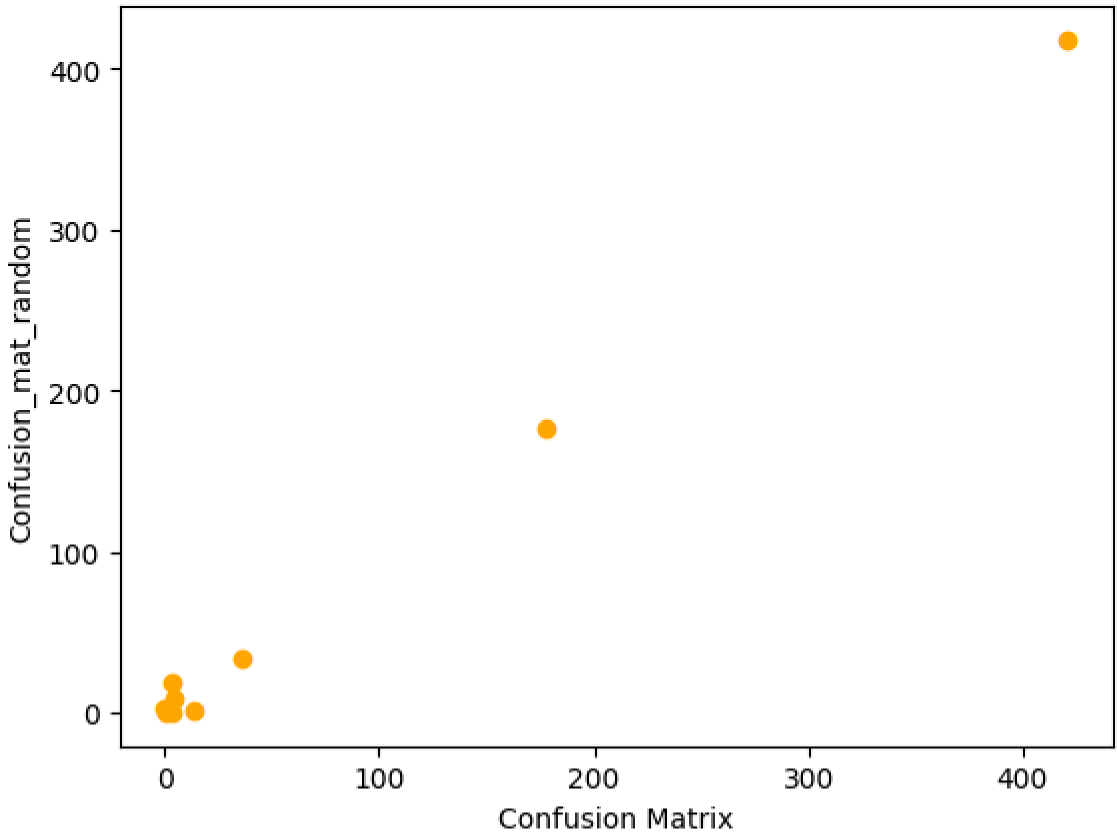

```
#visualization on evaluation two models
plt.bar(["Logistic Regression","Random Forest"],[accuracy,accuracy_random])
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.show()
```



```
#chart classification report
plt.scatter(classification_rep,classification_rep_random,color="Orange")
plt.xlabel("Classification Report")
plt.ylabel("Classification _rep_random")
plt.show()
```




```
#confusion matrix chart
plt.scatter(confusion_mat,confusion_mat_random,color="Orange")
plt.xlabel("Confusion Matrix")
plt.ylabel("Confusion_mat_random")
plt.show()
```




```
#final output prediction
final_output = pd.DataFrame({"Actual":y_test,"Logistic Regression":y_pred,"Random Forest"
print(final_output)
```

```
     Actual  Logistic Regression  Random Forest
239       0                    0              0
701       0                    0              0
655       2                    0              0
345       0                    0              0
302       2                    0              0
..      ...                  ...            ...
71        2                    0              0
106       0                    0              0
272       0                    0              2
441       0                    0              0
102       0                    0              0

[660 rows x 3 columns]
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.