About the Data

```
1. Day_of_week: The day of the week when the accident occurred.
2.Age_band_of_driver: The age group or band of the driver involved in the
accident.
3.Sex_of_driver: The gender of the driver involved in the accident.
4.Educational_level: The educational level of the driver involved in the
accident.
5.Vehicle_driver_relation: Relationship of the driver with the vehicle (e.g.,
owner, renter).
6.Driving_experience: Experience level of the driver in terms of years.
7.Type_of_vehicle: Type of vehicle involved in the accident (e.g., car, truck,
motorcycle).
8.Owner_of_vehicle: Ownership status of the vehicle (e.g., self-owned, company-
owned).
9.Service_year_of_vehicle: Number of years the vehicle has been in service.
10.Vehicle_movement: Movement or action of the vehicle before or during the
accident.
11.Casualty_class: Classification of the casualty (e.g., driver, passenger,
pedestrian).
12.Sex_of_casualty: Gender of the casualty involved in the accident.
13.Age_band_of_casualty: Age group or band of the casualty involved in the
accident.
14.Casualty_severity: Severity of the casualty (e.g., minor injury, serious
injury, fatality).
15.Work_of_casuality: Occupation or work status of the casualty.
16.Fitness_of_casuality: Fitness status of the casualty.
17.Pedestrian_movement: Movement of any pedestrians involved in the accident.
18.Cause_of_accident: The cause or reason for the accident.
19.Accident_severity: Severity of the accident itself.
```

Importing the libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
```

In [2]:
```
#Reading the dataset
df = pd.read_csv('RTA Dataset.csv')
```

In [3]:
```
df.head(5)
```

Out[3]:

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

5 rows × 32 columns

In [4]:
```
#Checking the information
df.info()
```

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

Their are "29" object columns and "2" numerical(Integer) Columns

In [5]:
```
#checking for null values
df.isnull().sum()
```

Out[5]:
```
Time                              0
Day_of_week                       0
Age_band_of_driver                0
Sex_of_driver                     0
Educational_level               741
Vehicle_driver_relation         579
Driving_experience              829
Type_of_vehicle                 950
Owner_of_vehicle                482
Service_year_of_vehicle        3928
Defect_of_vehicle              4427
Area_accident_occured           239
Lanes_or_Medians                385
Road_allignment                 142
Types_of_Junction               887
Road_surface_type               172
Road_surface_conditions           0
Light_conditions                  0
Weather_conditions                0
Type_of_collision               155
Number_of_vehicles_involved       0
Number_of_casualties              0
Vehicle_movement                308
Casualty_class                    0
Sex_of_casualty                   0
Age_band_of_casualty              0
Casualty_severity                 0
Work_of_casuality              3198
Fitness_of_casuality           2635
Pedestrian_movement               0
Cause_of_accident                 0
Accident_severity                 0
dtype: int64
```

In [6]:
```
print(31-15) #Their are 16 columns with null values present
```

```
16
```

In [7]:
```
#describing the data for numerical data
df.describe()
```

Out[7]:

	Number_of_vehicles_involved	Number_of_casualties
count	12316.000000	12316.000000
mean	2.040679	1.548149
std	0.688790	1.007179
min	1.000000	1.000000
25%	2.000000	1.000000
50%	2.000000	1.000000
75%	2.000000	2.000000
max	7.000000	8.000000

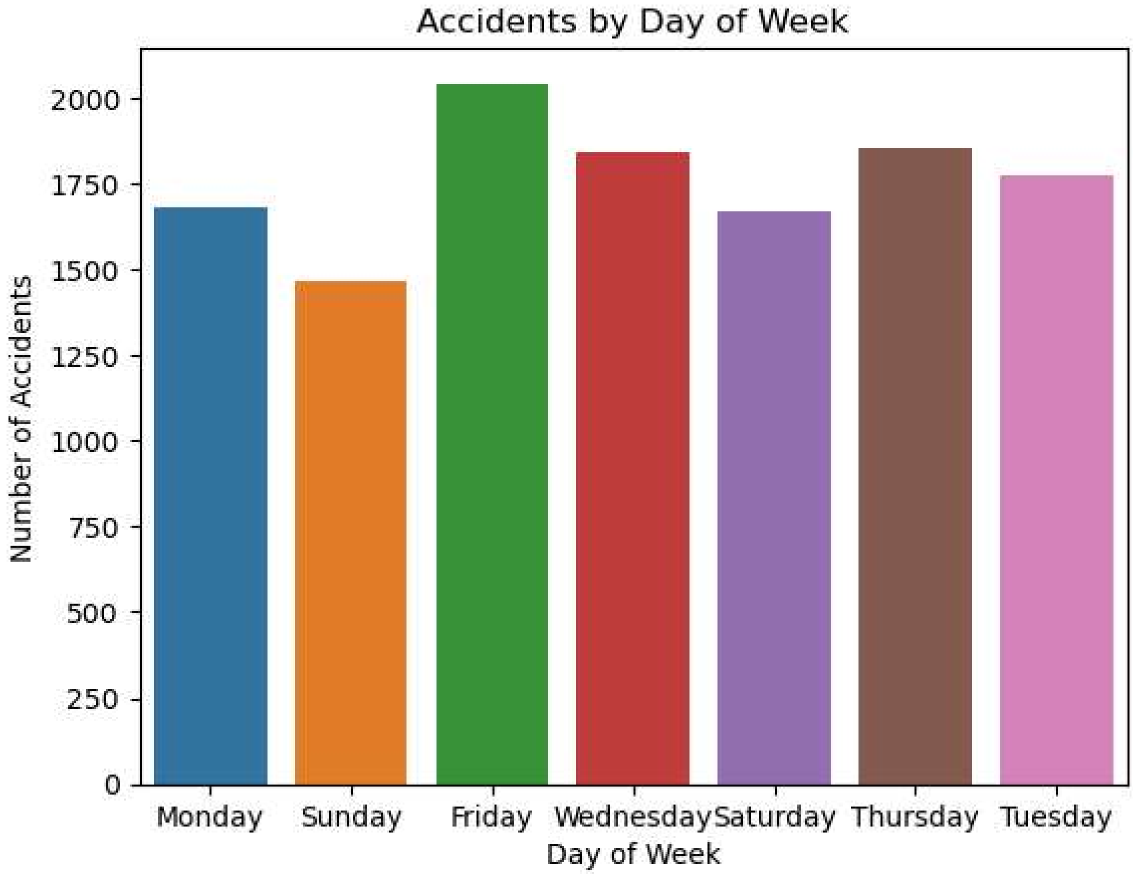
In [8]:
```
# Describing the data including categorical columns
df.describe(include='object')
```

Out[8]:

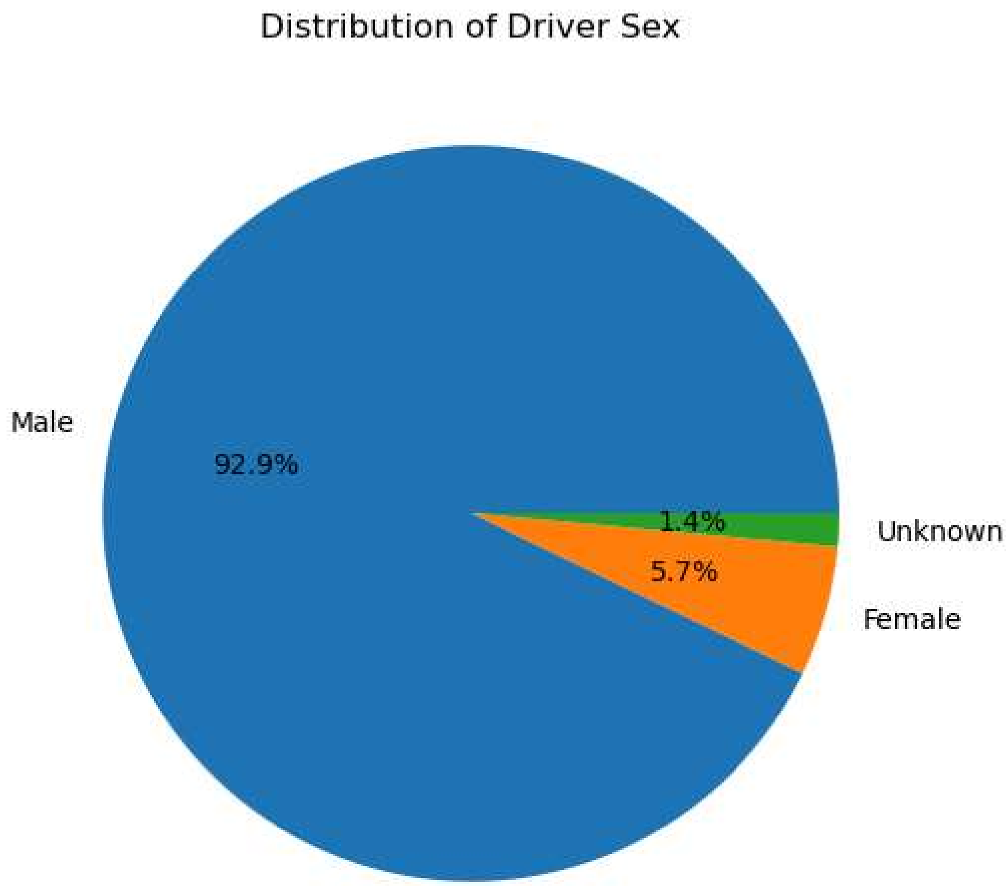
	Time	Day_of_week	Age_band_of_driver	Sex_of_driver	Educational_level	Vehicle_driver_
count	12316	12316	12316	12316	11575	
unique	1074	7	5	3	7	
top	15:30:00	Friday	18-30	Male	Junior high school	Er
freq	120	2041	4271	11437	7619	

4 rows × 30 columns

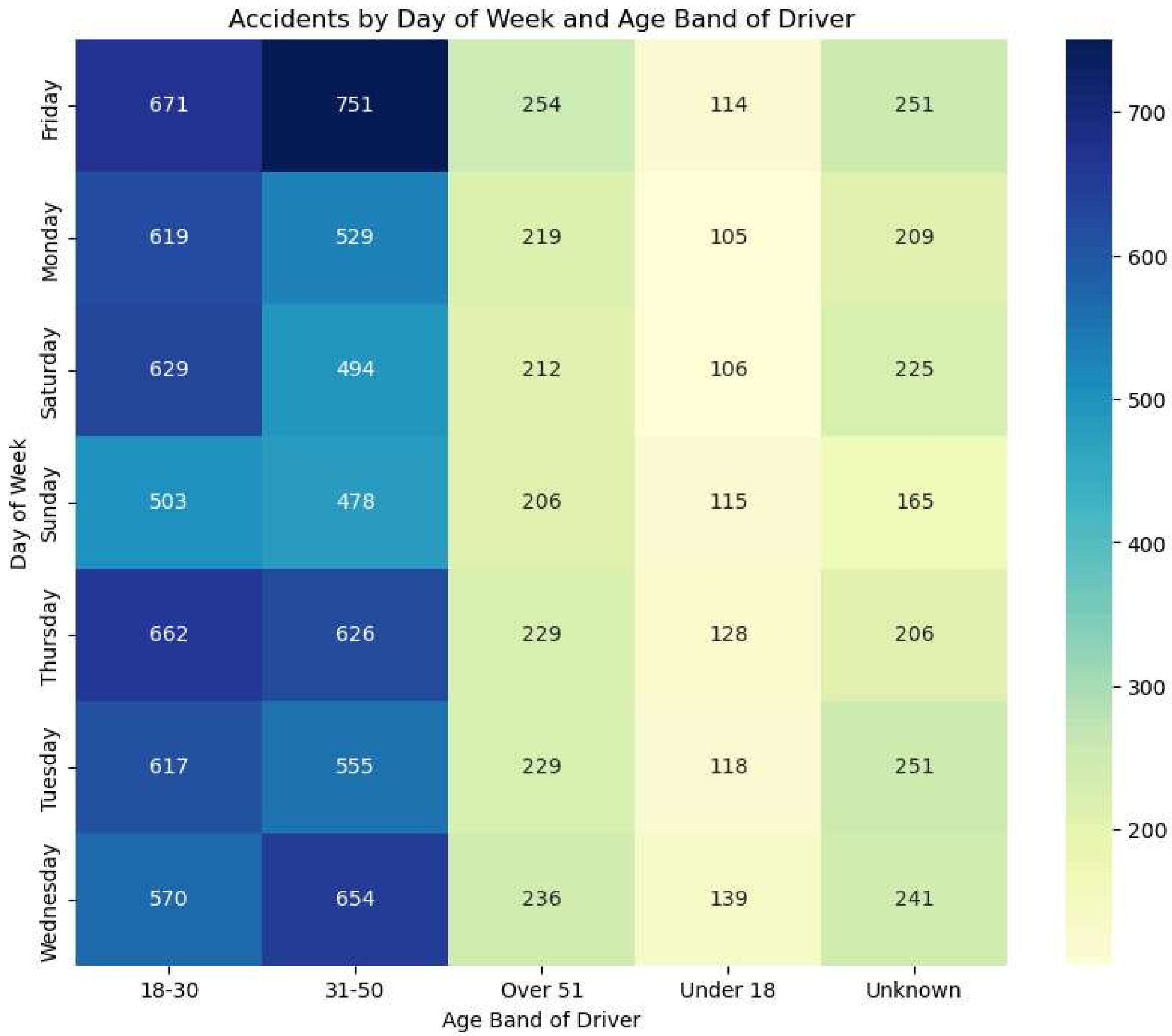
In [9]:
```
sns.countplot(x='Day_of_week', data=df)
plt.title('Accidents by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Number of Accidents')
plt.show()
```



In [10]:
```
plt.figure(figsize=(8, 6))
df['Sex_of_driver'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Distribution of Driver Sex')
plt.ylabel('')
plt.show()
```



In [11]:
```
plt.figure(figsize=(10, 8))
heatmap_data = df.groupby(['Day_of_week', 'Age_band_of_driver']).size().unstack(f
sns.heatmap(heatmap_data, cmap='YlGnBu', annot=True, fmt='d')
plt.title('Accidents by Day of Week and Age Band of Driver')
plt.xlabel('Age Band of Driver')
plt.ylabel('Day of Week')
plt.show()
```



In [12]:
```
#Checking for unique/ different type of values present in "Educational_level" colu
df['Educational_level'].unique()
```

Out[12]:
```
array(['Above high school', 'Junior high school', nan,
       'Elementary school', 'High school', 'Unknown', 'Illiterate',
       'Writing & reading'], dtype=object)
```

In [13]:
```
#Calculating the mode of "Educational_level"
ELM=df['Educational_level'].mode()
ELM
```

Out[13]:
```
0    Junior high school
Name: Educational_level, dtype: object
```

In [14]:
```
#Calculating the mode of "Vehicle_driver_relation"
VDRM = df['Vehicle_driver_relation'].mode()
VDRM
```

Out[14]:
```
0    Employee
Name: Vehicle_driver_relation, dtype: object
```

In [15]:
```
#Calculating the mode of "Driving_experience"
DE = df['Driving_experience'].mode()
DE
```

Out[15]:
```
0    5-10yr
Name: Driving_experience, dtype: object
```

In [16]:
```
print("The percentage of data that is null:")
df.isnull().sum()/len(df)*100
```

```
The percentage of data that is null:
```

Out[16]:
```
Time                            0.000000
Day_of_week                     0.000000
Age_band_of_driver              0.000000
Sex_of_driver                   0.000000
Educational_level               6.016564
Vehicle_driver_relation         4.701202
Driving_experience              6.731082
Type_of_vehicle                 7.713543
Owner_of_vehicle                3.913608
Service_year_of_vehicle        31.893472
Defect_of_vehicle              35.945112
Area_accident_occured           1.940565
Lanes_or_Medians                3.126015
Road_allignment                 1.152972
Types_of_Junction               7.202014
Road_surface_type               1.396557
Road_surface_conditions         0.000000
Light_conditions                0.000000
Weather_conditions              0.000000
Type_of_collision               1.258525
Number_of_vehicles_involved     0.000000
Number_of_casualties            0.000000
Vehicle_movement                2.500812
Casualty_class                  0.000000
Sex_of_casualty                 0.000000
Age_band_of_casualty            0.000000
Casualty_severity               0.000000
Work_of_casuality              25.966223
Fitness_of_casuality           21.394933
Pedestrian_movement             0.000000
Cause_of_accident               0.000000
Accident_severity               0.000000
dtype: float64
```

In [17]:
```
# created an function to replace the null values with mode value
def null_value_treatment(col):
    for i in df:
        if df[i].dtypes=='object':
            df[i].fillna(df[i].mode()[0],inplace=True)
        else:
            df[i].fillna(df[i].median(),inplace=True)
```

In [18]:
```
for i in df:
    null_value_treatment(i)
```

In [19]:
```
df.isnull().sum()/len(df)*100
```

Out[19]:
```
Time                           0.0
Day_of_week                    0.0
Age_band_of_driver             0.0
Sex_of_driver                  0.0
Educational_level              0.0
Vehicle_driver_relation        0.0
Driving_experience             0.0
Type_of_vehicle                0.0
Owner_of_vehicle               0.0
Service_year_of_vehicle        0.0
Defect_of_vehicle              0.0
Area_accident_occured          0.0
Lanes_or_Medians               0.0
Road_allignment                0.0
Types_of_Junction              0.0
Road_surface_type              0.0
Road_surface_conditions        0.0
Light_conditions               0.0
Weather_conditions             0.0
Type_of_collision              0.0
Number_of_vehicles_involved    0.0
Number_of_casualties           0.0
Vehicle_movement               0.0
Casualty_class                 0.0
Sex_of_casualty                0.0
Age_band_of_casualty           0.0
Casualty_severity              0.0
Work_of_casuality              0.0
Fitness_of_casuality           0.0
Pedestrian_movement            0.0
Cause_of_accident              0.0
Accident_severity              0.0
dtype: float64
```

In [20]:
```
df.sample(10)
```

Out[20]:

	Time	Day_of_week	Age_band_of_driver	Sex_of_driver	Educational_level	Vehicle_driver_re
7716	11:10:00	Thursday	Unknown	Male	Junior high school	
11313	17:40:00	Sunday	Over 51	Male	Junior high school	
36	20:30:00	Friday	18-30	Male	Above high school	Em
5185	18:11:00	Thursday	18-30	Male	Junior high school	
2133	1:32:00	Monday	31-50	Male	Junior high school	Em
4016	8:56:00	Saturday	Under 18	Male	High school	Em
526	13:28:00	Monday	18-30	Male	Junior high school	
4410	16:25:00	Monday	18-30	Male	Unknown	Em
7672	6:50:00	Monday	Under 18	Male	Junior high school	Em
7827	12:51:00	Wednesday	Over 51	Male	Junior high school	Em

10 rows × 32 columns

After filling the null values with mode, their are some columns that has "na" and "unknown" present.

In [21]:
```
df['Age_band_of_casualty'].unique()
```

Out[21]:
```
array(['na', '31-50', '18-30', 'Under 18', 'Over 51', '5'], dtype=object)
```

In [22]:
```
df['Service_year_of_vehicle'].unique()
```

Out[22]:
```
array(['Above 10yr', '5-10yrs', 'Unknown', '1-2yr', '2-5yrs', 'Below 1yr'],
      dtype=object)
```

In [23]:
```
df['Service_year_of_vehicle'].mode()[0]
```

Out[23]:
```
'Unknown'
```

In [24]:
```
df['Age_band_of_casualty'].mode()[0]
```

Out[24]:
```
'na'
```

As we can se that the mode value of that column is null

In [25]:
```
#Checking the correlation
df.corr()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_16032\18339900.py:2: FutureWarning: The
default value of numeric_only in DataFrame.corr is deprecated. In a future versi
on, it will default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  df.corr()
```

Out[25]:

	Number_of_vehicles_involved	Number_of_casualties
Number_of_vehicles_involved	1.000000	0.213427
Number_of_casualties	0.213427	1.000000

As we can see that "Number_of_vehicles_involved" has correlation with "Number_of_casualties" of 0.2134. A correlation of 0.213 is not particularly strong, but it’s not necessarily “bad.

In [26]:
```
df.skew()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_16032\1665899112.py:1: FutureWarning: T
he default value of numeric_only in DataFrame.skew is deprecated. In a future ve
rsion, it will default to False. In addition, specifying 'numeric_only=None' is
deprecated. Select only valid columns or specify the value of numeric_only to si
lence this warning.
  df.skew()
```

Out[26]:
```
Number_of_vehicles_involved    1.323454
Number_of_casualties           2.344769
dtype: float64
```

A skewness value of 1.3234, 2.344 indicates that the data is positively skewed (right-skewed) and -3.833 indicates that the data is left skewed

In [27]:
```
# Converting the categorical data into numerical data
LE = LabelEncoder()
```

In [28]:
```
def Categorical_numerical(col):
    for i in df:
        df[col] = LE.fit_transform(df[col])
```

In [29]:
```
for i in df:
    Categorical_numerical(i)
```

In [30]:
```
df.head(4)
```

Out[30]:

	Time	Day_of_week	Age_band_of_driver	Sex_of_driver	Educational_level	Vehicle_driver_relation
0	420	1	0	1	0	0
1	420	1	1	1	4	0
2	420	1	0	1	4	0
3	594	3	0	1	4	0

4 rows × 32 columns

In [31]:
```
#treating the outlier with zscore
from scipy.stats import zscore
def outlier_treatment(col):
    zscore1=(abs(zscore(df[col])))
    outlier=zscore1>+3
    median=df[col].median()
    df.loc[outlier,col]=median
```

In [32]:
```
for i in df:
    outlier_treatment(i)
```

Training the Model

In [33]:
```
X = df.drop('Accident_severity',axis=1)
```

In [34]:
```
y = df.Accident_severity
```

Balancing the data using smote

In [35]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)
df_resampled = pd.concat([pd.DataFrame(X_resampled), pd.DataFrame(y_resampled)], 
```

In [36]:
```
# Splitting the data into training and testing
from sklearn.model_selection import train_test_split
```

In [37]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.8,random_state=0
```

Logistic Regression

In [38]:
```
from sklearn.linear_model import LogisticRegression
LR=LogisticRegression()
```

In [39]:
```
LR.fit(x_train,y_train)
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: C
onvergenceWarning: lbfgs failed to converge (status=1):
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 (https://scikit-l
earn.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-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
  n_iter_i = _check_optimize_result(
```

Out[39]:
```
▾ LogisticRegression
LogisticRegression()
```

In [40]:
```
pred = LR.predict(x_test)
```

Evaluating the Model

In [41]:
```
accuracy_score(y_test,pred)*100
```

Out[41]:
```
85.86217395717041
```

In [43]:
```
predict = LR.predict(x_test)
```

In [44]:
```
confusion_matrix(y_test,predict)
```

Out[44]:
```
array([[   0, 1393],
       [   0, 8460]], dtype=int64)
```

DecisionTreeClassifier()

In [45]:
```
#Decision Tree classifier
DTC = DecisionTreeClassifier()
```

In [46]:
```
DTC.fit(x_train,y_train)
```
Out[46]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [47]:
```
y_pred = DTC.predict(x_test)
```

In [48]:
```
y_pred
```
Out[48]:
```
array([2, 1, 2, ..., 2, 2, 2], dtype=int64)
```

Evaluating the model

In [49]:
```
accuracy_score(y_test,y_pred)*100
```
Out[49]:
```
75.55059372779864
```

In [50]:
```
f1_score(y_test,y_pred,average='weighted')*100
```
Out[50]:
```
76.20773383900621
```

In [51]:
```
confusion_matrix(y_test,y_pred)
```
Out[51]:
```
array([[ 286, 1107],
       [1302, 7158]], dtype=int64)
```

K NEIGHBORS CLASSIFIER

In [52]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [53]:
```
KNN = KNeighborsClassifier(n_neighbors=25)
```

In [54]:
```
KNN.fit(x_train,y_train)
```
Out[54]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=25)
```

In [55]:
```
knn_pred=KNN.predict(x_test)
```

In [56]:
```
y_test
```

Out[56]:
```
8348    2
5386    2
1783    2
7077    2
5437    2
       ..
9012    2
886     2
5878    2
5876    2
6558    2
Name: Accident_severity, Length: 9853, dtype: int64
```

Evaluating the Model

In [57]:
```
accuracy_score(y_test,knn_pred)*100
```

Out[57]:
```
85.86217395717041
```

In [58]:
```
f1_score(y_test,knn_pred,average='weighted')*100
```

Out[58]:
```
79.33096616367189
```

In [59]:
```
confusion_matrix(y_test,knn_pred)
```

Out[59]:
```
array([[   0, 1393],
       [   0, 8460]], dtype=int64)
```

In [60]:
```
print("The true predictions are:")
8460+0
```

```
The true predictions are:
```

Out[60]:
```
8460
```

In [61]:
```
print("The False predictions are:")
1393+0
```

```
The False predictions are:
```

Out[61]:
```
1393
```

In [62]:
```
KNN.score(x_train,y_train)
```

Out[62]:
```
0.857896873731222
```

In [63]:
```
KNN.score(x_test,y_test)
```

Out[63]:
```
0.8586217395717041
```

Random Forest Classifier

In [64]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [65]:
```
RFC = RandomForestClassifier()
RFC.fit(x_train,y_train)
```

Out[65]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [66]:
```
model_pred = RFC.predict(x_test)
```

In [67]:
```
y_test
```

Out[67]:
```
8348    2
5386    2
1783    2
7077    2
5437    2
       ..
9012    2
886     2
5878    2
5876    2
6558    2
Name: Accident_severity, Length: 9853, dtype: int64
```

Evaluating the model

In [68]:
```
accuracy_score(y_test,model_pred)*100
```

Out[68]:
```
85.87232315030955
```

In [69]:
```
confusion_matrix(y_test,model_pred)
```

Out[69]:
```
array([[   5, 1388],
       [   4, 8456]], dtype=int64)
```

In [70]:
```
print("The True predictions are:")
(4+845)
```

```
The True predictions are:
```

Out[70]:
```
849
```

In [71]:
```
print("The False predictions are:")
2+1389
```

```
The False predictions are:
```

Out[71]:
```
1391
```

In [72]:
```
precision = precision_score(y_test, model_pred)
recall = recall_score(y_test, model_pred)

# Print the results
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

```
Precision: 0.56
Recall: 0.00
```

In [73]:
```
f1_score(y_test,model_pred)*100
```

Out[73]:
```
0.7132667617689015
```

Conclusion

```
1. Logistic Regression Model:
Algorithm Used: Logistic Regression
Performance Metrics:
Accuracy: 85.8621%
F1-score: 76.32%

2. Random Forest Model:
Algorithm Used: Random Forest
Performance Metrics:
Accuracy: 85.88%
F1-score: 0.57%

3. K-nearest Neighbour:
Algorithm Used: KNN Classifier
Performance Metrics:
Accuracy: 85.86%
F1-score: 79.33%

4. Decision Tree
Algorithm Used: Decsison tree Classifier
Performance Metrics:
Accuracy: 75.69%
F1-score: 76.32%
```

The analysis concludes with the following results:

Logistic Regression achieved an accuracy of approximately 85.86%. Decision Tree Classifier had an accuracy of around 75.69%. K-Nearest Neighbors (KNN) achieved an accuracy of 85.86%. Random Forest Classifier had an accuracy of approximately 85.88%. The models were evaluated based on accuracy, F1-score, and confusion matrices.

Further fine-tuning and feature engineering may improve model performance.

In [ ]:

```

```