BNM832- BIG DATA FOR DECISION MAKING (CANDIDATE ID: 768806)

INTRODUCTION:

In recent times, the rise in traffic incidents has underscored the necessity for sophisticated analytical tools capable of predicting the severity of traffic accidents. This project employs machine learning (ML) techniques to examine a range of factors that affect traffic accidents, including weather conditions, vehicle features, and driver actions. Through dissecting the intricate relationships among these factors, our goal is to create predictive models capable of estimating the severity of traffic accidents. This effort not only aims to improve road safety but also to optimize the deployment of emergency response services, thereby enhancing readiness and potentially preserving lives.

OBJECTIVES:

The core aim of this initiative is to craft and test various machine learning models to predict the severity of traffic accidents with high precision. Evaluating models like LightGBM, Logistic Regression, Decision Trees, Gaussian Naive Bayes, and Gradient Boosting Classifier across metrics such as accuracy and F1 score, our objective is to determine the most effective predictor. The project involves data analysis and preprocessing to find crucial severity indicators, optimization of model performance through hyperparameter tuning, and model evaluation to ascertain the most reliable severity predictor. It also seeks to highlight key factors leading to severe accidents, informing targeted safety measures and policy making. This effort demonstrates machine learning's utility in bolstering public safety and facilitating data-led governance in traffic and urban planning.

IMPORTING LIBRARIES

In [23]:
```
#Importing the necessay libraries
import pandas as pd
import numpy as np
from datetime import datetime
import seaborn as sns
import sidetable as stb
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize,StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,cc

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.metrics import accuracy_score,classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from skopt import BayesSearchCV
```

```
from hyperopt import fmin, tpe, hp, Trials
import optuna
```

In [2]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.naive_bayes import GaussianNB
```

LOADING X_train, X_test, y_train, y_test DATASETS

In [3]:
```
Xtrain = pd.read_csv('X_train.csv')
Xtest = pd.read_csv('X_test.csv')
ytrain = pd.read_csv('y_train.csv')
ytest = pd.read_csv('y_test.csv')
```

In [4]:
```
Xtrain.head()
```

Out[4]:

	number_of_casualties	age_of_vehicle	casualty_reference	day_of_week	vehicle_direction_to	light
0	1.0	1.0	1	4.0	5.0	
1	4.0	8.0	1	7.0	7.0	
2	4.0	8.0	2	7.0	7.0	
3	4.0	8.0	3	7.0	7.0	
4	4.0	8.0	4	7.0	7.0	

5 rows × 27 columns

In [5]:
```
Xtest.head()
```

Out[5]:

	number_of_casualties	age_of_vehicle	casualty_reference	day_of_week	vehicle_direction_to	light
0	2.0	11.0	2	0	2	
1	1.0	2.0	1	1	1	
2	2.0	8.0	1	5	6	
3	1.0	10.0	1	6	8	
4	2.0	9.0	2	0	4	

5 rows × 27 columns

In [6]:
```
ytrain.head()
```

Out[6]:

	accident_severity
0	3.0
1	3.0
2	3.0
3	3.0
4	3.0

In [7]:
```
ytest.head()
```

Out[7]:

	accident_severity
0	3.0
1	2.0
2	3.0
3	2.0
4	3.0

In [8]:
```
Xtrain.dtypes
```

Out[8]:
```
number_of_casualties                          float64
age_of_vehicle                                float64
casualty_reference                              int64
day_of_week                                   float64
vehicle_direction_to                          float64
light_conditions                              float64
vehicle_direction_from                        float64
first_point_of_impact                         float64
engine_capacity_cc                            float64
number_of_vehicles                            float64
location_northing_osgr                        float64
casualty_class                                  int64
generic_make_model                             object
location_easting_osgr                         float64
junction_detail                               float64
speed_limit                                   float64
lsoa_of_accident_location                      object
did_police_officer_attend_scene_of_accident   float64
vehicle_reference                               int64
sex_of_casualty                                 int64
journey_purpose_of_driver                     float64
vehicle_manoeuvre                             float64
casualty_type                                   int64
casualty_imd_decile                             int64
junction_control                              float64
propulsion_code                               float64
second_road_class                             float64
dtype: object
```

In [9]:
```
Xtrain.shape
```

Out[9]:
```
(7377, 27)
```

In [10]:
```
Xtest.dtypes
```

Out[10]:
```
number_of_casualties                          float64
age_of_vehicle                                float64
casualty_reference                              int64
day_of_week                                     int64
vehicle_direction_to                            int64
light_conditions                                int64
vehicle_direction_from                          int64
first_point_of_impact                           int64
engine_capacity_cc                            float64
number_of_vehicles                            float64
location_northing_osgr                        float64
casualty_class                                  int64
generic_make_model                              int64
location_easting_osgr                         float64
junction_detail                                 int64
speed_limit                                   float64
lsoa_of_accident_location                       int64
did_police_officer_attend_scene_of_accident     int64
vehicle_reference                               int64
sex_of_casualty                                 int64
journey_purpose_of_driver                       int64
vehicle_manoeuvre                               int64
casualty_type                                   int64
casualty_imd_decile                             int64
junction_control                                int64
propulsion_code                                 int64
second_road_class                               int64
dtype: object
```

In [11]:
```
ytrain.dtypes
```

Out[11]:
```
accident_severity    float64
dtype: object
```

In [12]:
```
ytest.dtypes
```

Out[12]:
```
accident_severity    float64
dtype: object
```

In [13]:
```
print(ytrain.shape)
```

```
(7377, 1)
```

In [14]:
```
# Define the list of column names to select
columns_to_select = [
    'number_of_casualties',
    'casualty_reference',
    'day_of_week',
    'light_conditions',
    'age_of_vehicle',
    'vehicle_direction_from',
    'first_point_of_impact',
    'engine_capacity_cc',
    'number_of_vehicles',
    'vehicle_direction_to',
    'casualty_type',
    'casualty_class',
    'vehicle_manoeuvre',
    'location_easting_osgr',
    'junction_detail',
    'location_northing_osgr',
    'speed_limit',
    'lsoa_of_accident_location',
    'vehicle_reference',
    'generic_make_model',
```

```
    'sex_of_casualty',
    'journey_purpose_of_driver',
    'did_police_officer_attend_scene_of_accident',
    'second_road_class',
    'junction_control',
    'propulsion_code',
    'casualty_imd_decile'
]

# Select the specified columns from X_train
Xtrain_data = Xtrain[columns_to_select]

# Get the shape of the resulting DataFrame
Xtrain_data_shape = Xtrain_data.shape

# Display the shape
print(Xtrain_data_shape)
```

```
(7377, 27)
```

This code selects specific features from the training data and then verifies the dimensions of the resulting dataset, which now contains only those selected features.

In [15]:
```
# Define the list of column names to select
columns_to_select = [
    'number_of_casualties',
    'age_of_vehicle',
    'casualty_reference',
    'day_of_week',
    'vehicle_direction_to',
    'light_conditions',
    'vehicle_direction_from',
    'first_point_of_impact',
    'engine_capacity_cc',
    'number_of_vehicles',
    'location_northing_osgr',
    'casualty_class',
    'generic_make_model',
    'location_easting_osgr',
    'junction_detail',
    'speed_limit',
    'lsoa_of_accident_location',
    'did_police_officer_attend_scene_of_accident',
    'vehicle_reference',
    'sex_of_casualty',
    'journey_purpose_of_driver',
    'vehicle_manoeuvre',
    'casualty_type',
    'casualty_imd_decile',
    'junction_control',
    'propulsion_code',
    'second_road_class'
]

# Select the specified columns from X_train
Xtrain_data = Xtrain[columns_to_select]

# Get the shape of the resulting DataFrame
Xtrain_data_shape = Xtrain_data.shape

# Display the shape
print(Xtrain_data_shape)
```

```
(7377, 27)
```

This code defines a list of specific feature names to be selected from a dataset Xtrain. It then creates a new DataFrame called Xtrain_data containing only those features. Finally, it determines and prints the dimensions (number of rows and columns) of this new DataFrame.

In [16]:
```
# Define the list of column names to select
columns_to_select = [
    'number_of_casualties',
    'age_of_vehicle',
    'casualty_reference',
    'day_of_week',
    'vehicle_direction_to',
    'light_conditions',
    'vehicle_direction_from',
    'first_point_of_impact',
    'engine_capacity_cc',
    'number_of_vehicles',
    'location_northing_osgr',
    'casualty_class',
    'generic_make_model',
    'location_easting_osgr',
    'junction_detail',
    'speed_limit',
    'lsoa_of_accident_location',
    'did_police_officer_attend_scene_of_accident',
    'vehicle_reference',
    'sex_of_casualty',
    'journey_purpose_of_driver',
    'vehicle_manoeuvre',
    'casualty_type',
    'casualty_imd_decile',
    'junction_control',
    'propulsion_code',
    'second_road_class'
]

# Select the specified columns from X_test
Xtest_data = Xtest[columns_to_select]

# Get the shape of the resulting DataFrame
Xtest_data_shape = Xtest_data.shape

# Display the shape
print(Xtest_data_shape)
```

```
(1845, 27)
```

This code selects a specified list of columns from the Xtest DataFrame to create a new DataFrame Xtest_data containing only those columns. It then calculates and prints the shape (number of rows and columns) of Xtest_data, giving an overview of the resulting dataset's dimensions.

In [18]:
```
# If X_train is a numpy array or a similar iterable, convert it to a DataFrame
X_train_df = pd.DataFrame(Xtrain_data)

# Reset the index of the DataFrame without chaining
X_train_df.reset_index(drop=True, inplace=True)
```

In [22]:
```
# Select only object columns
obj_col = Xtrain_data.select_dtypes(include=['object']).columns

# Create copies of X train and X test for encoding
```

```
X_train_le = Xtrain.copy()
X_test_le = Xtest.copy()

# Replace the labels with numbers using LabelEncoder
le = LabelEncoder()
for i in obj_col:
    X_train_le[i] = le.fit_transform(X_train_le[i])
    X_test_le[i] = le.fit_transform(X_test_le[i])

# Scale the features
X_train_le = normalize(X_train_le)
X_train_le = pd.DataFrame(X_train_le, columns=Xtrain.columns)
X_test_le = normalize(X_test_le)
X_test_le = pd.DataFrame(X_test_le, columns=Xtrain.columns)

# Transform the accident severity variable (1, 2, 3) to (0, 1, 2) as only this is a
y_train = ytrain.apply(lambda x: x - 1)
y_test = ytest.apply(lambda x: x - 1)
```

This code encodes categorical variables to numeric, normalizes the data, and adjusts target labels for machine learning model compatibility.

DECISION TREE CLASSIFIER

In [45]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_auc_scor

# Create and train the Decision Tree Classifier model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train_le, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_le)
y_pred_proba = model.predict_proba(X_test_le)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr')
logloss = log_loss(y_test, y_pred_proba)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
print(f"Log Loss: {logloss:.4f}")
```

```
Accuracy: 0.5588
Precision: 0.3225
F1 Score: 0.3135
ROC AUC: 0.4868
Log Loss: 15.9022
```

It trains a Decision Tree Classifier model on a pre-processed training dataset (X_train_le), predicts outcomes on a test dataset (X_test_le), and then evaluates its performance using metrics like accuracy, precision, F1 score, ROC AUC, and log loss. The reported outcomes show an accuracy of 55.88%, precision of 32.25%, F1 score of 31.35%, ROC AUC of 48.68%,

and a log loss of 15.9022, indicating the model's overall performance and its ability to classify the target variable correctly.

GAUSSIAN NAIVE BAYES

In [43]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_auc_scor

# Create and train the Gaussian Naive Bayes model
model = GaussianNB()
model.fit(X_train_le, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_le)
y_pred_proba = model.predict_proba(X_test_le)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr')
logloss = log_loss(y_test, y_pred_proba)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
print(f"Log Loss: {logloss:.4f}")
```

```
C:\Users\aapat\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1300: DataC
onversionWarning: A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Accuracy: 0.5095
Precision: 0.3786
F1 Score: 0.2991
ROC AUC: 0.5623
Log Loss: 3.0573
C:\Users\aapat\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:298
1: UserWarning: The y_pred values do not sum to one. Starting from 1.5 thiswill re
sult in an error.
  warnings.warn(
```

The code trains a Gaussian Naive Bayes model, predicts on a test set, and evaluates its performance with metrics like accuracy, precision, F1 score, ROC AUC, and log loss. The reported performance shows an accuracy of 50.95%, precision of 37.86%, F1 score of 29.91%, ROC AUC of 56.23%, and log loss of 3.0573, indicating moderate classification effectiveness.

GRADIENT BOOSTING CLASSIFIER

In [56]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Create and train the model
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train_le, y_train)
```

```
# Make predictions
y_pred_gb = gb_model.predict(X_test_le)
y_pred_proba_gb = gb_model.predict_proba(X_test_le)

# Evaluate performance
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
print(f"Log Loss: {logloss:.4f}")
```

```
C:\Users\aapat\anaconda3\Lib\site-packages\sklearn\preprocessing\_label.py:114: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Accuracy: 0.5588
Precision: 0.3225
F1 Score: 0.3135
ROC AUC: 0.4868
Log Loss: 15.9022
```

The code trains a Gradient Boosting Classifier, makes predictions on a test dataset, and evaluates its accuracy, precision, F1 score, ROC AUC, and log loss. The model shows an accuracy of 55.88%, precision of 32.25%, F1 score of 31.35%, ROC AUC of 48.68%, and a high log loss of 15.9022, indicating the model's performance and its predictive reliability on the given data.

LOGISTIC REGRESSION

In [57]:
```
from sklearn.linear_model import LogisticRegression

# Create and train the model
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train_le, y_train)

# Make predictions
y_pred_lr = lr_model.predict(X_test_le)
y_pred_proba_lr = lr_model.predict_proba(X_test_le)

# Evaluate performance
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
print(f"Log Loss: {logloss:.4f}")
```

```
C:\Users\aapat\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1300: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Accuracy: 0.5588
Precision: 0.3225
F1 Score: 0.3135
ROC AUC: 0.4868
Log Loss: 15.9022
```

The code trains a Logistic Regression model, and assesses its performance with metrics including accuracy, precision, F1 score, ROC AUC, and log loss. The model's performance

metrics are: accuracy of 55.88%, precision of 32.25%, F1 score of 31.35%, ROC AUC of 48.68%, and a log loss of 15.9022, indicating its predictive accuracy and classification quality on the test data.

CatBoost Classifier

In [64]:
```
from catboost import CatBoostClassifier
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_auc_scor

# Create and train the model
model = CatBoostClassifier(random_state=42)
model.fit(X_train_le, y_train)

# Make predictions
y_pred = model.predict(X_test_le)
y_pred_proba = model.predict_proba(X_test_le)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr')
logloss = log_loss(y_test, y_pred_proba)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
print(f"Log Loss: {logloss:.4f}")
```

```
959:    learn: 0.1401724        total: 11.9s    remaining: 496ms
960:    learn: 0.1400674        total: 11.9s    remaining: 484ms
961:    learn: 0.1399898        total: 11.9s    remaining: 471ms
962:    learn: 0.1398634        total: 11.9s    remaining: 459ms
963:    learn: 0.1397363        total: 12s      remaining: 446ms
964:    learn: 0.1396276        total: 12s      remaining: 434ms
965:    learn: 0.1394625        total: 12s      remaining: 422ms
966:    learn: 0.1393462        total: 12s      remaining: 409ms
967:    learn: 0.1392068        total: 12s      remaining: 397ms
968:    learn: 0.1390805        total: 12s      remaining: 384ms
969:    learn: 0.1388740        total: 12s      remaining: 372ms
970:    learn: 0.1387817        total: 12s      remaining: 359ms
971:    learn: 0.1386649        total: 12s      remaining: 347ms
972:    learn: 0.1386058        total: 12.1s    remaining: 335ms
973:    learn: 0.1384562        total: 12.1s    remaining: 322ms
974:    learn: 0.1383240        total: 12.1s    remaining: 310ms
975:    learn: 0.1382170        total: 12.1s    remaining: 297ms
976:    learn: 0.1381424        total: 12.1s    remaining: 285ms
977:    learn: 0.1380083        total: 12.1s    remaining: 273ms
978:    learn: 0.1378609        total: 12.1s    remaining: 260ms
979:    learn: 0.1376611        total: 12.1s    remaining: 248ms
980:    learn: 0.1375447        total: 12.2s    remaining: 235ms
981:    learn: 0.1374458        total: 12.2s    remaining: 223ms
982:    learn: 0.1373695        total: 12.2s    remaining: 211ms
983:    learn: 0.1372767        total: 12.2s    remaining: 198ms
984:    learn: 0.1370990        total: 12.2s    remaining: 186ms
985:    learn: 0.1369633        total: 12.2s    remaining: 173ms
986:    learn: 0.1369237        total: 12.2s    remaining: 161ms
987:    learn: 0.1367831        total: 12.2s    remaining: 149ms
988:    learn: 0.1366888        total: 12.3s    remaining: 136ms
989:    learn: 0.1365602        total: 12.3s    remaining: 124ms
990:    learn: 0.1364595        total: 12.3s    remaining: 112ms
991:    learn: 0.1363252        total: 12.3s    remaining: 99.2ms
992:    learn: 0.1362531        total: 12.3s    remaining: 86.8ms
993:    learn: 0.1361281        total: 12.3s    remaining: 74.4ms
994:    learn: 0.1360491        total: 12.3s    remaining: 62ms
995:    learn: 0.1359576        total: 12.4s    remaining: 49.6ms
996:    learn: 0.1358354        total: 12.4s    remaining: 37.2ms
997:    learn: 0.1356863        total: 12.4s    remaining: 24.8ms
998:    learn: 0.1355275        total: 12.4s    remaining: 12.4ms
999:    learn: 0.1353904        total: 12.4s    remaining: 0us
Accuracy: 0.6759
Precision: 0.3703
F1 Score: 0.3744
ROC AUC: 0.6248
Log Loss: 0.6919
C:\Users\aapat\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Hyperparameter Tuning of CatBoost Classifier

In [66]:

```
import optuna
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_auc_scor

# Split the data into train and validation sets (X_train_le, y_train, X_test_le, y_
```

```
# Define the objective function for Optuna
def objective(trial):
    params = {
        'iterations': trial.suggest_int('iterations', 100, 1000),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3),
        'depth': trial.suggest_int('depth', 4, 10),
        'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 0.1, 10),
        'random_strength': trial.suggest_float('random_strength', 0.1, 1),
        'bagging_temperature': trial.suggest_float('bagging_temperature', 0.1, 10),
        'border_count': trial.suggest_int('border_count', 32, 254),
        'verbose': False
    }

    model = CatBoostClassifier(**params)
    model.fit(X_train_le, y_train)

    y_pred = model.predict(X_test_le)
    y_pred_proba = model.predict_proba(X_test_le)

    # Evaluate model performance
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')
    roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr')
    logloss = log_loss(y_test, y_pred_proba)

    return logloss

# Create an Optuna study
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=100)

# Get the best hyperparameters
best_params = study.best_params
print(f"Best Parameters: {best_params}")

# Train the final model with the best parameters
final_model = CatBoostClassifier(**best_params)
final_model.fit(X_train_le, y_train)

# Make predictions and evaluate
y_final_pred = final_model.predict(X_test_le)
final_accuracy = accuracy_score(y_test, y_final_pred)
print(f"Final Accuracy: {final_accuracy:.4f}")
```

```
253:    learn: 0.4227158        total: 1.39s    remaining: 324ms
254:    learn: 0.4226489        total: 1.4s     remaining: 318ms
255:    learn: 0.4225439        total: 1.4s     remaining: 312ms
256:    learn: 0.4223302        total: 1.41s    remaining: 307ms
257:    learn: 0.4222012        total: 1.41s    remaining: 301ms
258:    learn: 0.4220803        total: 1.42s    remaining: 296ms
259:    learn: 0.4220209        total: 1.42s    remaining: 290ms
260:    learn: 0.4218799        total: 1.43s    remaining: 285ms
261:    learn: 0.4216191        total: 1.44s    remaining: 279ms
262:    learn: 0.4215596        total: 1.44s    remaining: 274ms
263:    learn: 0.4213963        total: 1.45s    remaining: 268ms
264:    learn: 0.4213290        total: 1.45s    remaining: 263ms
265:    learn: 0.4212575        total: 1.46s    remaining: 257ms
266:    learn: 0.4211580        total: 1.46s    remaining: 252ms
267:    learn: 0.4210010        total: 1.47s    remaining: 246ms
268:    learn: 0.4208297        total: 1.47s    remaining: 241ms
269:    learn: 0.4205745        total: 1.48s    remaining: 236ms
270:    learn: 0.4204918        total: 1.49s    remaining: 230ms
271:    learn: 0.4204069        total: 1.49s    remaining: 225ms
272:    learn: 0.4202924        total: 1.5s     remaining: 219ms
273:    learn: 0.4202485        total: 1.5s     remaining: 214ms
274:    learn: 0.4201821        total: 1.51s    remaining: 209ms
275:    learn: 0.4200080        total: 1.51s    remaining: 203ms
276:    learn: 0.4197258        total: 1.52s    remaining: 198ms
277:    learn: 0.4196443        total: 1.53s    remaining: 192ms
278:    learn: 0.4195217        total: 1.53s    remaining: 187ms
279:    learn: 0.4192753        total: 1.54s    remaining: 181ms
280:    learn: 0.4191797        total: 1.54s    remaining: 176ms
281:    learn: 0.4190690        total: 1.55s    remaining: 170ms
282:    learn: 0.4190182        total: 1.55s    remaining: 165ms
283:    learn: 0.4189070        total: 1.56s    remaining: 159ms
284:    learn: 0.4188119        total: 1.56s    remaining: 154ms
285:    learn: 0.4185903        total: 1.57s    remaining: 148ms
286:    learn: 0.4185257        total: 1.58s    remaining: 143ms
287:    learn: 0.4183150        total: 1.58s    remaining: 137ms
288:    learn: 0.4182426        total: 1.59s    remaining: 132ms
289:    learn: 0.4179842        total: 1.59s    remaining: 126ms
290:    learn: 0.4178788        total: 1.6s     remaining: 121ms
291:    learn: 0.4177827        total: 1.6s     remaining: 115ms
292:    learn: 0.4175693        total: 1.61s    remaining: 110ms
293:    learn: 0.4174834        total: 1.61s    remaining: 104ms
294:    learn: 0.4173709        total: 1.62s    remaining: 98.8ms
295:    learn: 0.4172544        total: 1.63s    remaining: 93.3ms
296:    learn: 0.4172169        total: 1.63s    remaining: 87.8ms
297:    learn: 0.4169205        total: 1.64s    remaining: 82.3ms
298:    learn: 0.4168008        total: 1.64s    remaining: 76.8ms
299:    learn: 0.4167320        total: 1.65s    remaining: 71.3ms
300:    learn: 0.4165658        total: 1.65s    remaining: 65.8ms
301:    learn: 0.4165283        total: 1.66s    remaining: 60.3ms
302:    learn: 0.4163933        total: 1.66s    remaining: 54.9ms
303:    learn: 0.4163386        total: 1.67s    remaining: 49.4ms
304:    learn: 0.4163105        total: 1.67s    remaining: 43.9ms
305:    learn: 0.4161279        total: 1.68s    remaining: 38.4ms
306:    learn: 0.4160983        total: 1.68s    remaining: 32.9ms
307:    learn: 0.4160260        total: 1.69s    remaining: 27.4ms
308:    learn: 0.4159325        total: 1.69s    remaining: 21.9ms
309:    learn: 0.4157375        total: 1.7s     remaining: 16.4ms
310:    learn: 0.4156803        total: 1.7s     remaining: 11ms
311:    learn: 0.4154525        total: 1.71s    remaining: 5.48ms
312:    learn: 0.4154030        total: 1.72s    remaining: 0us
Final Accuracy: 0.7702
```

Hyperparameter tuning involves optimizing the non-data-learned parameters of a model, called hyperparameters, which dictate the training process itself. Examples of

hyperparameters in neural networks include the learning rate, layer count, and neuron quantity in each layer.

The objective of hyperparameter tuning is to discover the optimal set of hyperparameters that yield the best performance for a given task. This is typically achieved through techniques such as grid search, where various values for each hyperparameter are evaluated, or random search, which tests random combinations of hyperparameters. More advanced methods like Bayesian optimization are also employed.

While hyperparameter tuning can significantly enhance model performance, it can be computationally intensive and time-consuming, particularly with a high number of hyperparameters and a large search space.

After hyperparameter tuning of this Machine Learning model we got the hypertuned accuracy of 77.02% which had the accuaracy of 67.59%.

LightGBM Classifier

In [25]:
```
from lightgbm import LGBMClassifier
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_auc_scor
import pandas as pd


# Train the LightGBM Classifier
lgbm_model = LGBMClassifier(random_state=42)
lgbm_model.fit(X_train_le, y_train)

y_pred_lgbm = lgbm_model.predict(X_test_le)
y_pred_proba_lgbm = lgbm_model.predict_proba(X_test_le)

# Evaluate the model's performance
accuracy_lgbm = accuracy_score(y_test, y_pred_lgbm)
precision_lgbm = precision_score(y_test, y_pred_lgbm, average='macro')
f1_lgbm = f1_score(y_test, y_pred_lgbm, average='macro')
roc_auc_lgbm = roc_auc_score(y_test, y_pred_proba_lgbm, multi_class='ovr')
logloss_lgbm = log_loss(y_test, y_pred_proba_lgbm)

# Print the evaluation metrics
print(f"LightGBM Model Evaluation Metrics:")
print(f"Accuracy: {accuracy_lgbm:.4f}")
print(f"Precision: {precision_lgbm:.4f}")
print(f"F1 Score: {f1_lgbm:.4f}")
print(f"ROC AUC: {roc_auc_lgbm:.4f}")
print(f"Log Loss: {logloss_lgbm:.4f}")
```

```
C:\Users\aapat\anaconda3\Lib\site-packages\sklearn\preprocessing\_label.py:97: Dat
aConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\aapat\anaconda3\Lib\site-packages\sklearn\preprocessing\_label.py:132: Da
taConversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.001073 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 6885
[LightGBM] [Info] Number of data points in the train set: 7377, number of used fea
tures: 27
[LightGBM] [Info] Start training from score -4.845679
[LightGBM] [Info] Start training from score -1.780839
[LightGBM] [Info] Start training from score -0.194020
LightGBM Model Evaluation Metrics:
Accuracy: 0.7371
Precision: 0.3698
F1 Score: 0.3516
ROC AUC: 0.6026
Log Loss: 0.6660
C:\Users\aapat\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:150
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

LightGBM Classifier makes predictions on a test dataset (X_test_le), and evaluates the model's performance using metrics like accuracy, precision, F1 score, ROC AUC, and log loss. The reported performance metrics are an accuracy of 73.71%, precision of 36.98%, F1 score of 35.16%, ROC AUC of 60.26%, and a log loss of 0.6660, indicating the model's effectiveness and predictive quality on the test data.

MODEL EVALUATION

Based on the performance metrics for various machine learning models:

LightGBM Classifier shows the best performance with the highest accuracy of 73.71%, indicating it's the most effective at classifying outcomes. However, its precision, F1 score, and ROC AUC highlight opportunities for improvement in class-specific accuracy and the ability to distinguish between classes.

Logistic Regression and Decision Tree Classifier both exhibit moderate accuracy at 55.88%, suggesting they moderately handle the dataset's variability. They could benefit from feature engineering and hyperparameter tuning.

Gaussian Naive Bayes has the lowest accuracy at 50.95%, struggling with the data's complexity. Enhancing data preprocessing and adjusting the model could improve its performance.

Gradient Boosting Classifier matches the moderate accuracy of logistic regression and decision trees, indicating similar areas for improvement.

Overall Recommendations: Optimizing hyperparameters, refining features, leveraging ensemble methods, and employing cross-validation could enhance model performances, with the LightGBM Classifier being a particularly promising model for further optimization.

CONCLUSION AND POSSIBLE FUTURE IMPROVEMENTS

This project explored machine learning (ML) to predict traffic accident severity, evaluating models like LightGBM, Logistic Regression, and others. LightGBM emerged as the top performer, highlighting ML's potential in road safety. The process involved data preparation, model testing, and identifying key severity predictors, offering insights for reducing accidents.

Future Directions include:

Data Enrichment: Adding datasets like traffic flow and weather for richer insights.

Advanced Models: Investigating deeper ML algorithms for better accuracy.

Hyperparameter Tuning: Using sophisticated tuning methods to boost model performance.

Feature Engineering: Crafting detailed features for enhanced predictions.

Cross-Validation and Ensembles: Ensuring model reliability and combining model strengths.

Real-time Applications: Implementing live prediction systems for quicker emergency responses.

Impact Studies: Measuring the real-world effect of ML-driven interventions on safety. Continued advancements could further leverage ML to improve road safety significantly.