**Load the Dataset**

# Install missing packages (if needed)

!pip install lime shap matplotlib seaborn pandas scikit-learn openpyxl

# Importing libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import (

    accuracy\_score, precision\_score, recall\_score, f1\_score,

    confusion\_matrix, roc\_auc\_score, classification\_report, cohen\_kappa\_score

)

import lime.lime\_tabular

import shap

import matplotlib.pyplot as plt

import seaborn as sns

import time

# For visualizing SHAP

shap.initjs()

# Load the dataset

df = pd.read\_excel('Accident\_Dataset\_Preprocessing.xlsx', sheet\_name='Accident')

# Display the first few rows

df.head()

**A screenshot of a computer

Description automatically generated**

**Data Preprocessing**

# 1. Drop columns with more than 50% missing values

threshold = 0.5 \* len(df)

df = df.dropna(thresh=threshold, axis=1)

# 2. Drop rows where critical values are missing (target or key columns)

df = df.dropna(subset=['Accident\_severity', 'Time'])

# 3. Impute categorical features with the most frequent value (mode)

for column in df.select\_dtypes(include='object').columns:

    df[column].fillna(df[column].mode()[0], inplace=True)

# 4. Impute numerical features with the median value

for column in df.select\_dtypes(include=['int64', 'float64']).columns:

    df[column].fillna(df[column].median(), inplace=True)

# Verify if all missing values are handled

print(df.isnull().sum())

# Save the preprocessed dataset to an Excel file

df.to\_excel('Accident\_Dataset\_After\_Preprocessing.xlsx', index=False)

print("Preprocessed dataset saved successfully.")

**A screenshot of a computer program

Description automatically generated**

For the above steps:

Step 1: Drop columns where more than 50% of values are missing to avoid introducing noise.

Step 2: Drop rows where target values (Accident\_severity) or critical features are missing since they are essential for prediction.

Step 3: For categorical variables, use mode (most frequent value) to fill missing data.

Step 4: For numerical variables, use the median to avoid the effect of outliers.

# Save the preprocessed dataset to an Excel file

df.to\_excel('Accident\_Dataset\_After\_Preprocessing.xlsx', index=False)

print("Preprocessed dataset saved as 'Accident\_Dataset\_After\_Preprocessing.xlsx'")



**Load the Pre-processed Data**

# Load the preprocessed dataset

df = pd.read\_excel('Accident\_Dataset\_After\_Preprocessing.xlsx')

# Display the first few rows to confirm

df.head()

**A screenshot of a black screen

Description automatically generated**

**Model Building and Evaluation**

# Encode categorical variables

label\_encoders = {}

for column in df.select\_dtypes(include='object').columns:

    le = LabelEncoder()

    df[column] = le.fit\_transform(df[column])

    label\_encoders[column] = le

# Separate features and target

X = df.drop('Accident\_severity', axis=1)

y = df['Accident\_severity']

# Standardize numerical features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split the dataset into training and testing sets (80/20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Train the Decision Tree model

start\_time = time.time()

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

dt\_time = time.time() - start\_time

# Train the Logistic Regression model

start\_time = time.time()

lr\_model = LogisticRegression(max\_iter=1000, random\_state=42)

lr\_model.fit(X\_train, y\_train)

lr\_time = time.time() - start\_time

# Define a function to evaluate models

def evaluate\_model(model, X\_test, y\_test, model\_name, train\_time):

    y\_pred = model.predict(X\_test)

    print(f"Performance Metrics for {model\_name}:")

    print("Accuracy:", accuracy\_score(y\_test, y\_pred))

    print("Precision:", precision\_score(y\_test, y\_pred, average='weighted'))

    print("Recall:", recall\_score(y\_test, y\_pred, average='weighted'))

    print("F1 Score:", f1\_score(y\_test, y\_pred, average='weighted'))

    print("Cohen's Kappa:", cohen\_kappa\_score(y\_test, y\_pred))

    print("Training Time:", train\_time, "seconds")

    print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Evaluate both models

evaluate\_model(dt\_model, X\_test, y\_test, "Decision Tree", dt\_time)

evaluate\_model(lr\_model, X\_test, y\_test, "Logistic Regression", lr\_time)

**A screenshot of a computer

Description automatically generated**

**Visualizations**

# Confusion Matrix for Decision Tree

dt\_conf\_matrix = confusion\_matrix(y\_test, dt\_model.predict(X\_test))

sns.heatmap(dt\_conf\_matrix, annot=True, cmap='Blues', fmt='d')

plt.title('Confusion Matrix - Decision Tree')

plt.show()

# Feature Importance for Decision Tree

plt.figure(figsize=(10, 5))

feature\_importances = pd.Series(dt\_model.feature\_importances\_, index=X.columns)

feature\_importances.nlargest(10).plot(kind='barh')

plt.title('Feature Importances - Decision Tree')

plt.show()

**A screenshot of a computer screen

Description automatically generated**

**XAI Techniques – LIME and SHAP**

# Initialize LIME explainer

explainer = lime.lime\_tabular.LimeTabularExplainer(

    X\_train, feature\_names=X.columns, class\_names=['Slight', 'Serious'], mode='classification'

)

# Explain a sample prediction with LIME (Decision Tree)

i = 5  # Example index from test set

exp = explainer.explain\_instance(X\_test[i], dt\_model.predict\_proba)

exp.show\_in\_notebook()

**A black screen with orange text

Description automatically generated**

**SHAP**

# Initialize SHAP TreeExplainer

shap\_explainer = shap.TreeExplainer(dt\_model)

shap\_values = shap\_explainer.shap\_values(X\_test)

# Plot SHAP summary

shap.summary\_plot(shap\_values, X\_test, feature\_names=X.columns)

A graph of a number of lines

Description automatically generated with medium confidence

**MY EXTENDED SCRIPT**

# Install missing packages (if needed)

!pip install lime shap matplotlib seaborn pandas scikit-learn openpyxl

# Importing libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import (

    accuracy\_score, precision\_score, recall\_score, f1\_score,

    confusion\_matrix, roc\_auc\_score, classification\_report, cohen\_kappa\_score

)

from sklearn.inspection import partial\_dependence, PartialDependenceDisplay

import lime.lime\_tabular

import shap

import matplotlib.pyplot as plt

import seaborn as sns

import time

# For visualizing SHAP

shap.initjs()

# Load the dataset

df = pd.read\_excel('Accident\_Dataset\_Preprocessing.xlsx', sheet\_name='Accident')

# Display the first few rows

df.head()

# 1. Drop columns with more than 50% missing values

threshold = 0.5 \* len(df)

df = df.dropna(thresh=threshold, axis=1)

# 2. Drop rows where critical values are missing (target or key columns)

df = df.dropna(subset=['Accident\_severity', 'Time'])

# 3. Impute categorical features with the most frequent value (mode)

for column in df.select\_dtypes(include='object').columns:

    df[column].fillna(df[column].mode()[0], inplace=True)

# 4. Impute numerical features with the median value

for column in df.select\_dtypes(include=['int64', 'float64']).columns:

    df[column].fillna(df[column].median(), inplace=True)

# Verify if all missing values are handled

print(df.isnull().sum())

# Save the preprocessed dataset to an Excel file

df.to\_excel('Accident\_Dataset\_After\_Preprocessing.xlsx', index=False)

print("Preprocessed dataset saved successfully.")

# Load the preprocessed dataset

df = pd.read\_excel('Accident\_Dataset\_After\_Preprocessing.xlsx')

# Display the first few rows to confirm

df.head()

# Encode categorical variables

label\_encoders = {}

for column in df.select\_dtypes(include='object').columns:

    le = LabelEncoder()

    df[column] = le.fit\_transform(df[column])

    label\_encoders[column] = le

# Separate features and target

X = df.drop('Accident\_severity', axis=1)

y = df['Accident\_severity']

# Standardize numerical features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split the dataset into training and testing sets (80/20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Train the Decision Tree model

start\_time = time.time()

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

dt\_time = time.time() - start\_time

# Train the Logistic Regression model

start\_time = time.time()

lr\_model = LogisticRegression(max\_iter=1000, random\_state=42)

lr\_model.fit(X\_train, y\_train)

lr\_time = time.time() - start\_time

# Define a function to evaluate models

def evaluate\_model(model, X\_test, y\_test, model\_name, train\_time):

    y\_pred = model.predict(X\_test)

    print(f"Performance Metrics for {model\_name}:")

    print("Accuracy:", accuracy\_score(y\_test, y\_pred))

    print("Precision:", precision\_score(y\_test, y\_pred, average='weighted'))

    print("Recall:", recall\_score(y\_test, y\_pred, average='weighted'))

    print("F1 Score:", f1\_score(y\_test, y\_pred, average='weighted'))

    print("Cohen's Kappa:", cohen\_kappa\_score(y\_test, y\_pred))

    print("Training Time:", train\_time, "seconds")

    print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Evaluate both models

evaluate\_model(dt\_model, X\_test, y\_test, "Decision Tree", dt\_time)

evaluate\_model(lr\_model, X\_test, y\_test, "Logistic Regression", lr\_time)

# Confusion Matrix for Decision Tree

dt\_conf\_matrix = confusion\_matrix(y\_test, dt\_model.predict(X\_test))

sns.heatmap(dt\_conf\_matrix, annot=True, cmap='Blues', fmt='d')

plt.title('Confusion Matrix - Decision Tree: Displaying True vs Predicted Accident Severity Counts')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

# Feature Importance for Decision Tree

plt.figure(figsize=(10, 5))

feature\_importances = pd.Series(dt\_model.feature\_importances\_, index=X.columns)

feature\_importances.nlargest(10).plot(kind='barh')

plt.title('Feature Importances - Decision Tree: Top 10 Features Contributing to Accident Severity Prediction')

plt.xlabel('Feature Importance Score')

plt.ylabel('Features')

plt.show()

# Initialize LIME explainer

explainer = lime.lime\_tabular.LimeTabularExplainer(

    X\_train, feature\_names=X.columns, class\_names=['Slight', 'Serious'], mode='classification'

)

# Explain a sample prediction with LIME (Decision Tree)

i = 5  # Example index from test set

exp = explainer.explain\_instance(X\_test[i], dt\_model.predict\_proba)

exp.show\_in\_notebook()

# Initialize SHAP TreeExplainer

shap\_explainer = shap.TreeExplainer(dt\_model)

shap\_values = shap\_explainer.shap\_values(X\_test)

# Plot SHAP summary

shap.summary\_plot(shap\_values, X\_test, feature\_names=X.columns)

# Plot PDP (Partial Dependence Plot)

features\_to\_plot = [0, 1, 2]  # Indexes of features to plot (example)

fig, ax = plt.subplots(figsize=(10, 8))

PartialDependenceDisplay.from\_estimator(dt\_model, X\_train, features=features\_to\_plot, feature\_names=X.columns, target=0, ax=ax)

plt.title(f"Partial Dependence Plot - Decision Tree: Effect of Features ({', '.join(X.columns[features\_to\_plot])}) on Predicted Accident Severity")

plt.xlabel(f"{X.columns[2]} Value")

plt.ylabel('Partial Dependence')

plt.show()

# Visual comparison of LIME, SHAP, and PDP

# SHAP Summary Plot

shap.summary\_plot(shap\_values, X\_test, feature\_names=X.columns)

# PDP for one feature

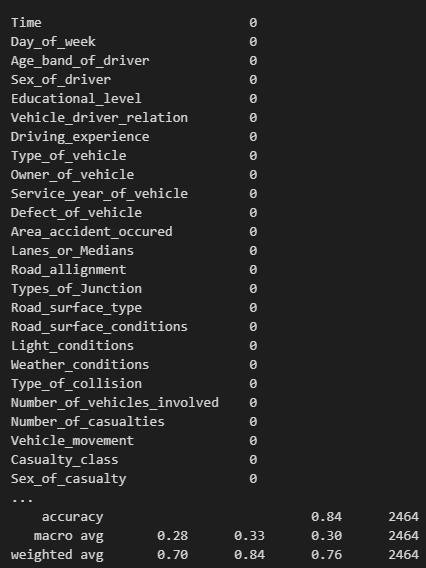
PartialDependenceDisplay.from\_estimator(dt\_model, X\_train, features=[0], feature\_names=X.columns, target=0)

plt.title(f"Partial Dependence Plot - {X.columns[0]}: Illustrating the Effect of {X.columns[0]} on Predicted Accident Severity")

plt.xlabel('Feature 1 Value')

plt.ylabel('Partial Dependence')

plt.show()

****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

from sklearn.metrics import roc\_curve, auc

# Predict probabilities for both models

y\_prob\_dt = dt\_model.predict\_proba(X\_test)

y\_prob\_lr = lr\_model.predict\_proba(X\_test)

# Compute ROC curve and ROC area for both models (using one class as an example)

fpr\_dt, tpr\_dt, \_ = roc\_curve(y\_test, y\_prob\_dt[:, 1], pos\_label=1)

fpr\_lr, tpr\_lr, \_ = roc\_curve(y\_test, y\_prob\_lr[:, 1], pos\_label=1)

# Compute AUC

roc\_auc\_dt = auc(fpr\_dt, tpr\_dt)

roc\_auc\_lr = auc(fpr\_lr, tpr\_lr)

# Plot ROC curve

plt.figure(figsize=(8, 6))

plt.plot(fpr\_dt, tpr\_dt, color='darkorange', lw=2, label=f'Decision Tree (AUC = {roc\_auc\_dt:.2f})')

plt.plot(fpr\_lr, tpr\_lr, color='blue', lw=2, label=f'Logistic Regression (AUC = {roc\_auc\_lr:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

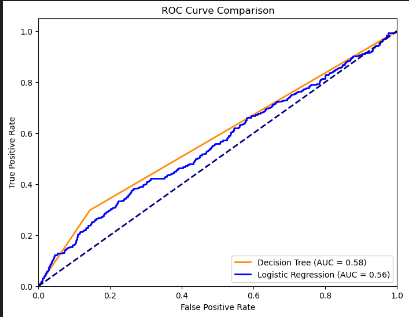
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve Comparison')

plt.legend(loc='lower right')

plt.show()

****