**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()
```

| | Time | Day_of_week | Age_band_of_driver | Sex_of_driver | Educational_level | Vehicle_driver_relation | Driving_experience | Type_of_vehicle | Owner_of_vehicle | Service_year_of_vehicle | ... | Vehicle_movemen |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 17:02:00 | Monday | 18-30 | Male | Above high school | Employee | 1-2yr | Automobile | Owner | Above 10yr | ... | Going straigh |
| 1 | 17:02:00 | Monday | 31-50 | Male | Junior high school | Employee | Above 10yr | Public (> 45 seats) | Owner | 5-10yrs | ... | Going straigh |
| 2 | 17:02:00 | Monday | 18-30 | Male | Junior high school | Employee | 1-2yr | Lorry (41?100Q) | Owner | NaN | ... | Going straigh |
| 3 | 01:06:00 | Sunday | 18-30 | Male | Junior high school | Employee | 5-10yr | Public (> 45 seats) | Governmental | NaN | ... | Going straigh |
| 4 | 01:06:00 | Sunday | 18-30 | Male | Junior high school | Employee | 2-5yr | NaN | Owner | 5-10yrs | ... | Going straigh |

5 rows × 32 columns

**Data Preprocessing**

```
# 1. Drop columns with more than 50% missing values
threshold = 0.5 * len(df)
df = df.dropna(thresh=threshold, axis=1)
```

```python
# 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.")
```

```
Time                          0
Day_of_week                   0
Age_band_of_driver            0
Sex_of_driver                 0
Educational_level             0
Vehicle_driver_relation       0
Driving_experience            0
Type_of_vehicle               0
Owner_of_vehicle              0
Service_year_of_vehicle       0
Defect_of_vehicle             0
Area_accident_occured         0
Lanes_or_Medians              0
Road_allignment               0
Types_of_Junction             0
Road_surface_type             0
Road_surface_conditions       0
Light_conditions              0
Weather_conditions            0
Type_of_collision             0
Number_of_vehicles_involved   0
Number_of_casualties          0
Vehicle_movement              0
Casualty_class                0
Sex_of_casualty               0
...
Cause_of_accident             0
Accident_severity             0
dtype: int64
Preprocessed dataset saved successfully.
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

```
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.
```

```python
# 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'")
```

```
Preprocessed dataset saved as 'Accident_Dataset_After_Preprocessing.xlsx'
```

**Load the Pre-processed Data**

```python
# Load the preprocessed dataset
df = pd.read_excel('Accident_Dataset_After_Preprocessing.xlsx')

# Display the first few rows to confirm
df.head()
```

| | Time | Day_of_week | Age_band_of_driver | Sex_of_driver | Educational_level | Vehicle_driver_relation | Driving_experience | Type_of_vehicle | Owner_of_vehicle | Service_year_of_vehicle | ... | Vehicle_movemen |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 17:02:00 | Monday | 18-30 | Male | Above high school | Employee | 1-2yr | Automobile | Owner | Above 10yr | ... | Going straigh |
| 1 | 17:02:00 | Monday | 31-50 | Male | Junior high school | Employee | Above 10yr | Public (> 45 seats) | Owner | 5-10yrs | ... | Going straigh |
| 2 | 17:02:00 | Monday | 18-30 | Male | Junior high school | Employee | 1-2yr | Lorry (41?100Q) | Owner | Unknown | ... | Going straigh |
| 3 | 01:06:00 | Sunday | 18-30 | Male | Junior high school | Employee | 5-10yr | Public (> 45 seats) | Governmental | Unknown | ... | Going straigh |
| 4 | 01:06:00 | Sunday | 18-30 | Male | Junior high school | Employee | 2-5yr | Automobile | Owner | 5-10yrs | ... | Going straigh |

**Model Building and Evaluation**

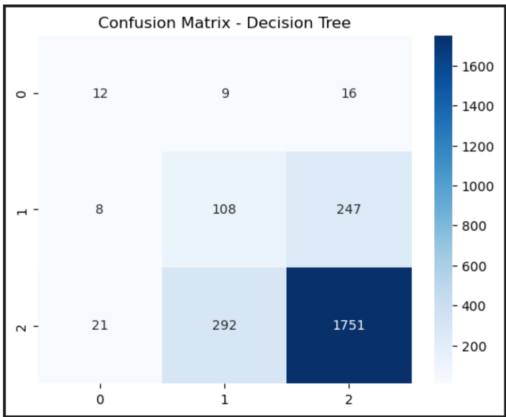
```python
# Encode categorical variables
label_encoders = {}
for column in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# 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)
```

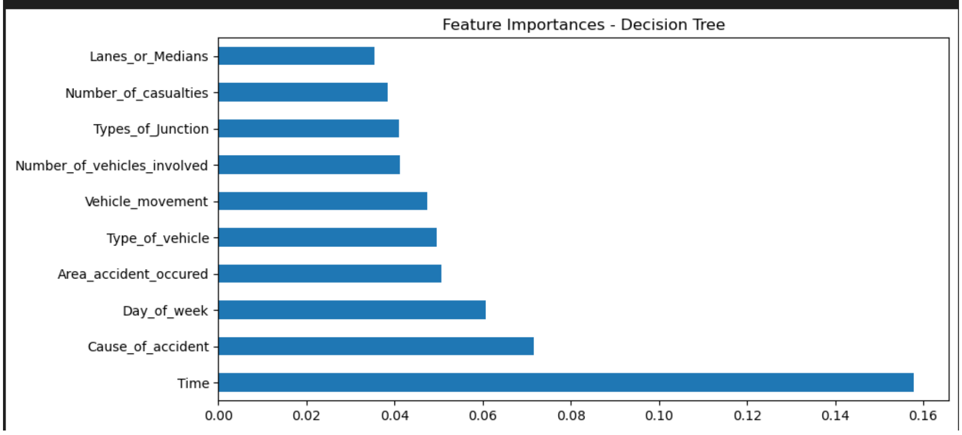
```python
dt_time = time.time() - start_time

# 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)
```

```
Performance Metrics for Decision Tree:
Accuracy: 0.7593344155844156
Precision: 0.7715719456537042
Recall: 0.7593344155844156
F1 Score: 0.7651859738258828
Cohen's Kappa: 0.17187783379883215
Training Time: 0.11581206321716309 seconds

Classification Report:
               precision    recall  f1-score   support

           0       0.29      0.32      0.31        37
           1       0.26      0.30      0.28       363
           2       0.87      0.85      0.86      2064

    accuracy                           0.76      2464
   macro avg       0.48      0.49      0.48      2464
weighted avg       0.77      0.76      0.77      2464

Performance Metrics for Logistic Regression:
Accuracy: 0.8376623376623377
Precision: 0.7016781919379322
Recall: 0.8376623376623377
F1 Score: 0.7636638979395164
Cohen's Kappa: 0.0
...
    accuracy                           0.84      2464
   macro avg       0.28      0.33      0.30      2464
weighted avg       0.70      0.84      0.76      2464
```

**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()
```



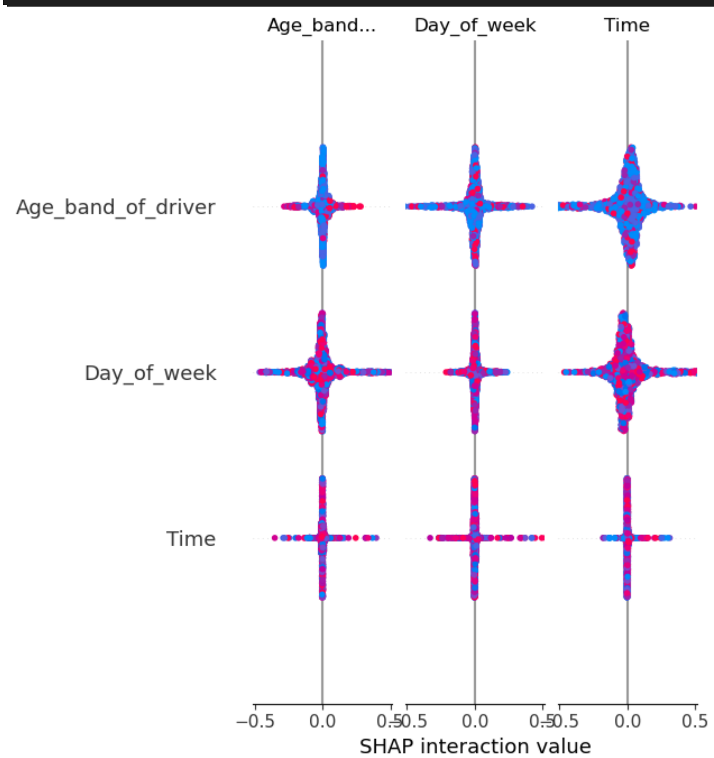

**XAI Techniques – LIME and SHAP**

```
# 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()
```


**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)
```



**MY EXTENDED SCRIPT**

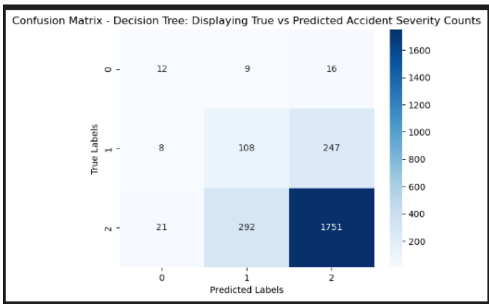
```
# Install missing packages (if needed)
!pip install lime shap matplotlib seaborn pandas scikit-learn openpyxl

# 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
```

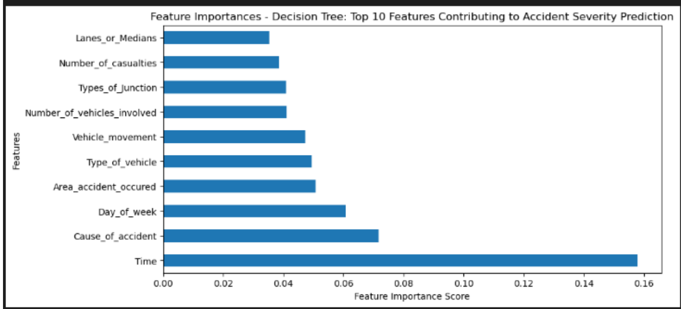
```python
print(df.isnull().sum())

# 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))
```

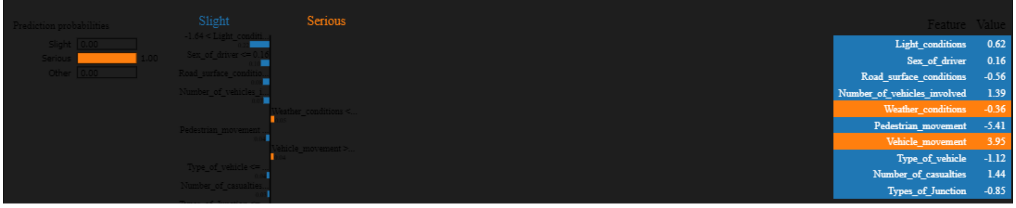
```python
    print("Precision:", precision_score(y_test, y_pred, average='weighted'))
    print("Recall:", recall_score(y_test, y_pred, average='weighted'))
    print("F1 Score:", f1_score(y_test, y_pred, average='weighted'))
    print("Cohen's Kappa:", cohen_kappa_score(y_test, y_pred))
    print("Training 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)
```

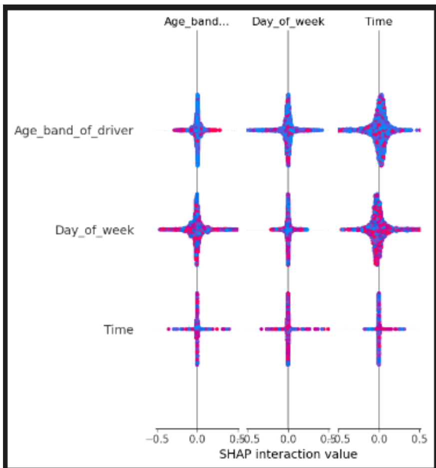
```
# Plot PDP (Partial Dependence Plot)
features_to_plot = [0, 1, 2]  # Indexes of features to plot (example)
fig, ax = plt.subplots(figsize=(10, 8))
PartialDependenceDisplay.from_estimator(dt_model, X_train,
features=features_to_plot, feature_names=X.columns, target=0, ax=ax)
plt.title(f"Partial Dependence 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()
```

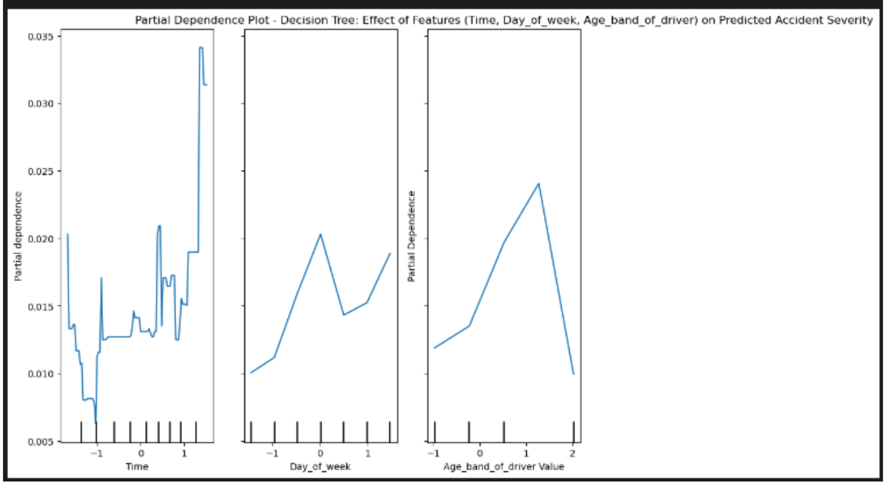
```
Time                           0
Day_of_week                    0
Age_band_of_driver             0
Sex_of_driver                  0
Educational_level              0
Vehicle_driver_relation        0
Driving_experience             0
Type_of_vehicle                0
Owner_of_vehicle               0
Service_year_of_vehicle        0
Defect_of_vehicle              0
Area_accident_occured          0
Lanes_or_Medians               0
Road_allignment                0
Types_of_Junction              0
Road_surface_type              0
Road_surface_conditions        0
Light_conditions               0
Weather_conditions             0
Type_of_collision              0
Number_of_vehicles_involved    0
Number_of_casualties           0
Vehicle_movement               0
Casualty_class                 0
Sex_of_casualty                0
...
    accuracy                           0.84      2464
   macro avg       0.28      0.33      0.30      2464
weighted avg       0.70      0.84      0.76      2464
```

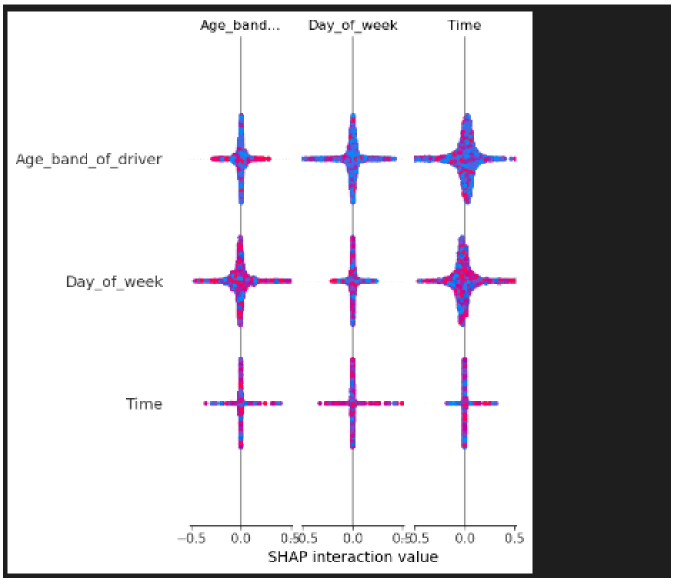

Confusion Matrix - Decision Tree: Displaying True vs Predicted Accident Severity Counts
True Labels
Predicted Labels
0
1
2
12
9
16
8
108
247
21
292
1751
1600
1400
1200
1000
800
600
400
200

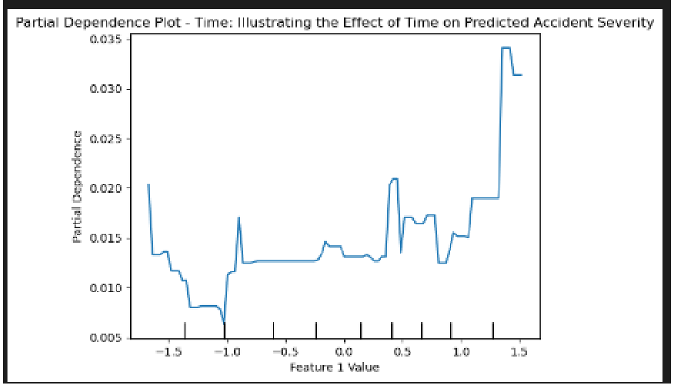

Feature Importances - Decision Tree: Top 10 Features Contributing to Accident Severity Prediction
Lanes_or_Medians
Number_of_casualties
Types_of_Junction
Number_of_vehicles_involved
Vehicle_movement
Type_of_vehicle
Area_accident_occured
Day_of_week
Cause_of_accident
Time
Features
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
Feature Importance Score


Slight
Serious
Feature
Value
Light_conditions
0.62
Sex_of_driver
0.16
Road_surface_conditions
-0.56
Number_of_vehicles_involved
1.39
Weather_conditions
-0.36
Pedestrian_movement
-5.41
Vehicle_movement
3.95
Type_of_vehicle
-1.12
Number_of_casualties
1.44
Types_of_Junction
-0.85


Age_band...
Day_of_week
Time
Age_band_of_driver
Day_of_week
Time
SHAP interaction value


Partial Dependence Plot - Decision Tree: Effect of Features (Time, Day_of_week, Age_band_of_driver) on Predicted Accident Severity
Partial dependence
Time
Day_of_week
Partial Dependence
Age_band_of_driver Value




```
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()
```