Phase-3

Student Name: Sowparnikashree P

Register Number: 410723104084

Institution: DHANALAKSHMI COLLEGE OF ENGINEERING

Department: COMPUTER SCIENCE AND ENGINEERING

Date of Submission: 16-05-2025

Github Repository Link:

https://github.com/Sowparnikashree-2005/NM_sowparnika

1. Problem Statement

Despite advances in transportation infrastructure, road traffic accidents remain a major global issue, causing significant loss of life, injury, and economic impact. Traditional methods of traffic safety management often rely on historical data and reactive measures, which are insufficient for proactive risk mitigation. There is a critical need for intelligent systems that can analyze complex traffic patterns, predict accident-prone areas, and provide timely alerts to prevent collisions.

2. Abstract

- Road traffic accidents are a leading cause of injury and death worldwide, posing a critical challenge to public safety and urban mobility. Traditional approaches to accident prevention often rely on reactive strategies and fail to leverage the full potential of modern data analytics. This study explores the development of an AI-driven framework for traffic accident analysis and prediction, aiming to shift from reactive to proactive road safety management.
- This work highlights the transformative potential of artificial intelligence in traffic safety, offering a scalable and intelligent solution to one of the most pressing issues in urban transportation systems

3. System Requirements:

- **Hardware**: RAM :64 GB, **Processor :** Intel Xeon Silver/Gold or AMD EPYC
- **Software**: Python ,IDE (Colab)

4. Objectives:

- **Develop an AI-Powered Predictive Model**

Design and implement machine learning and deep learning models to accurately predict traffic accidents based on historical and real-time data (e.g., traffic volume, weather, time, location).

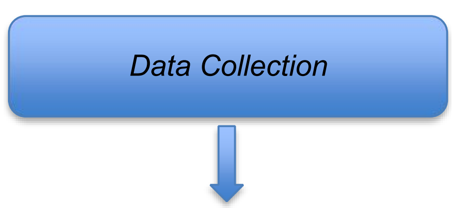
- **Identify High-Risk Zones and Time Periods**

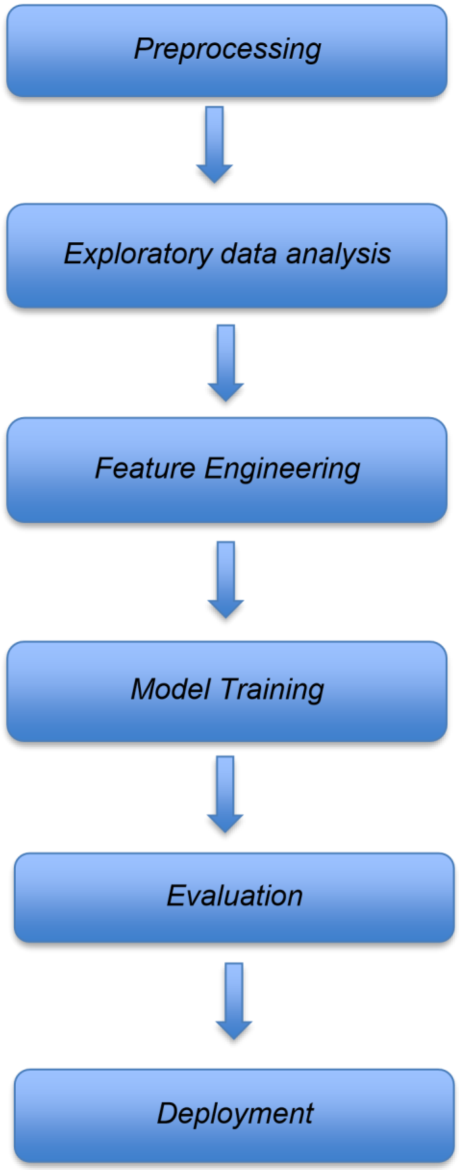
Use spatial and temporal analysis to detect accident hotspots and peak risk windows, enabling proactive safety interventions by traffic authorities.

- **Integrate and Analyze Multi-Source Data**

Collect and unify data from various sources—traffic sensors, GPS, weather APIs, CCTV, and accident databases—for comprehensive risk modeling.

5.flowchart of project work flow:





6. Dataset Description:

source: Kaggle – driver response

Type: Public
Size:5*21 columns

```python
import pandas as pd
import numpy as numpy
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("/driverresponse.csv")
df.head()
```

	index	sno	stateut	region	regionid	alcintake2014	overspeed2014	overtaking2014	lanejumping2014	wrongside2014	...	asleep2014	othercause
0	0	1	Andhra Pradesh	south	2	594	12747.0	507	328	668	...	154	
1	1	2	Arunachal Pradesh	northeast	5	11	16.0	0	0	0	...	0	
2	2	3	Assam	northeast	5	613	4596.0	129	104	156	...	3	
3	3	4	Bihar	north	1	1680	1496.0	278	236	308	...	72	
4	4	5	Chhattisgarh	centre	9	335	6720.0	188	313	266	...	81	

5 rows × 21 columns

7. Data Preprocessing:

1.Handling missing values
2. Data normalization
3. Feature engineering (extracting relevant features)
4. Encoding categorical variables
5. Data transformation (converting data types)
6. Removing duplicates and outliers.

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

index	0
sno	0
stateut	0
region	0
regionid	0
alcintake2014	0
overspeed2014	0
overtaking2014	0
lanejumping2014	0
wrongside2014	0
signalavoid2014	0
asleep2014	0
othercause2014	0
alcintake2016	0
overspeed2016	0
signalavoid2016	0
wrongside2016	0
lanejumping2016	0
overtaking2016	0
asleep2016	0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   index            36 non-null     int64  
 1   sno              36 non-null     int64  
 2   stateut          36 non-null     object 
 3   region           36 non-null     object 
 4   regionid         36 non-null     int64  
 5   alcintake2014    36 non-null     int64  
 6   overspeed2014    36 non-null     float64
 7   overtaking2014   36 non-null     int64  
 8   lanejumping2014  36 non-null     int64  
 9   wrongside2014    36 non-null     int64  
 10  signalavoid2014  36 non-null     int64  
 11  asleep2014       36 non-null     int64  
 12  othercause2014   36 non-null     int64  
 13  alcintake2016    36 non-null     int64  
 14  overspeed2016    36 non-null     int64  
 15  signalavoid2016  36 non-null     int64  
 16  wrongside2016    36 non-null     int64  
 17  lanejumping2016  36 non-null     int64  
 18  overtaking2016   36 non-null     int64  
 19  asleep2016       36 non-null     int64  
 20  othercause2016   36 non-null     int64  
dtypes: float64(1), int64(18), object(2)
```

```
13] df.describe()
```

	index	sno	regionid	alcintake2014	overspeed2014	overtaking2014	lanejumping2014	wrongside2014	signalavoid2014	asleep2014	othercause2014	alcintake2016	overspeed2016	s
count	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000	36.000000	
mean	17.500000	18.500000	4.000000	525.444444	5785.305556	312.833333	283.361111	302.000000	37.000000	92.027778	1007.944444	413.722222	7453.916667	
std	10.535654	10.535654	2.746426	767.133866	8610.175276	514.513279	462.370115	389.184642	81.196411	234.019472	1780.826436	907.754202	11689.392631	
min	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	8.750000	9.750000	1.750000	11.750000	50.250000	1.500000	0.000000	6.250000	0.000000	0.000000	19.000000	24.000000	140.500000	
50%	17.500000	18.500000	4.000000	82.500000	2407.000000	75.500000	86.500000	113.000000	4.500000	5.000000	156.000000	105.000000	2009.500000	
75%	26.250000	27.250000	5.000000	706.500000	6379.500000	350.000000	316.750000	531.750000	34.500000	133.000000	810.000000	412.500000	10966.500000	
max	35.000000	36.000000	9.000000	3540.000000	29790.000000	2171.000000	1869.000000	1529.000000	451.000000	1393.000000	6874.000000	4633.000000	47055.000000	

```
print(df.duplicated().sum())
```

```
0
```

8. Exploratory Data Analysis (EDA):

1. Data cleaning and preprocessing
2. Summary statistics (mean, median, mode, etc.)
3. Data visualization (plots, charts, etc.)
4. Correlation analysis
5. Pattern identification

```python
import pandas as pd

df = pd.read_csv("/driverresponse.csv")

numeric_df= df.select_dtypes (include=['number'])

if numeric_df.empty:
    print("\nNo numeric columns found in the dataset.")
else:
    mean = numeric_df.mean()
    median = numeric_df.median()
    var = numeric_df.var()
    std = numeric_df.std()
    print("\nMean:\n", mean)
    print("\nMedian:\n", median)
    print("\nVariance: \n", var)
    print("\nStandard Deviation:\n", std)
```

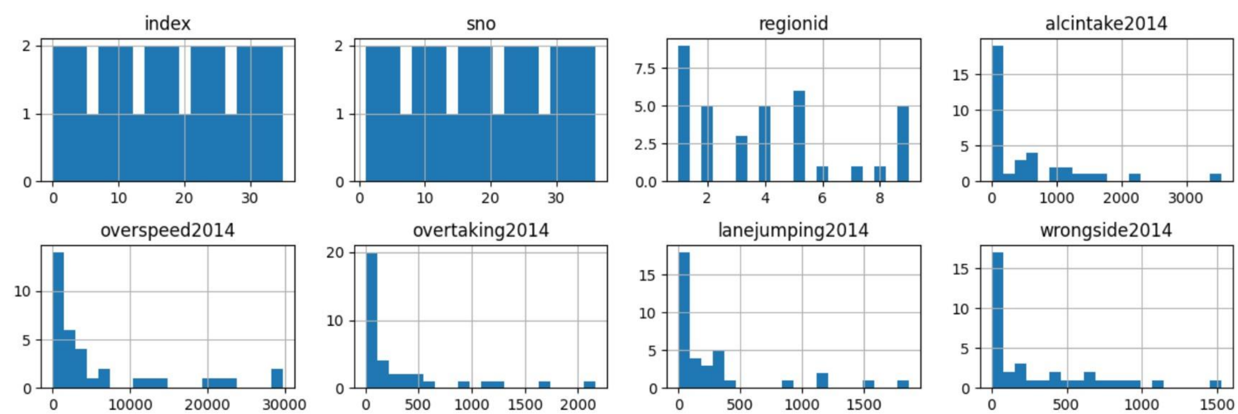
```
Mean:
 index                17.500000
sno                  18.500000
regionid              4.000000
alcintake2014       525.444444
overspeed2014      5950.600000
overtaking2014      312.833333
lanejumping2014     283.361111
wrongside2014       302.000000
signalavoid2014      37.000000
asleep2014           92.027778
othercause2014     1007.944444
alcintake2016       413.722222
overspeed2016      7453.916667
signalavoid2016     124.750000
wrongside2016       490.388889
lanejumping2016     236.472222
overtaking2016      823.527778
asleep2016          126.444444
othercause2016     1403.611111
dtype: float64
```

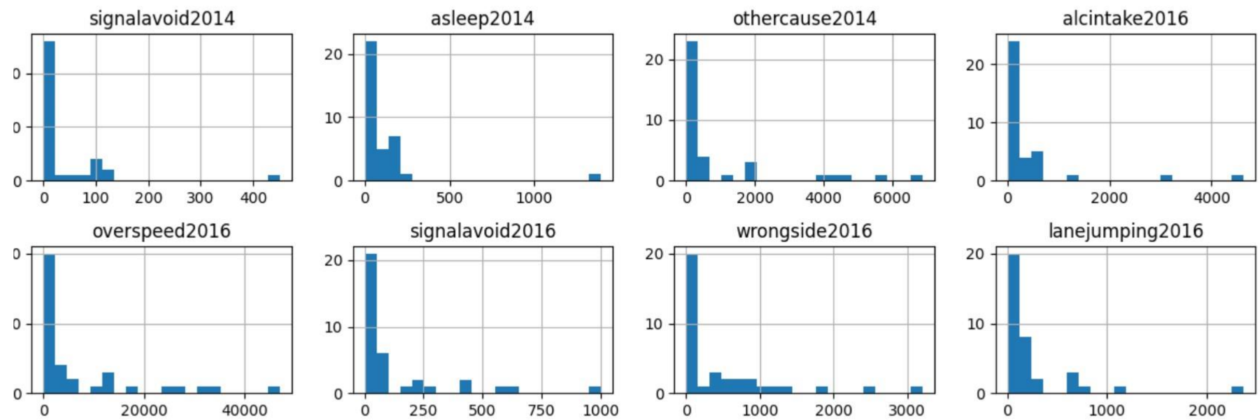
```
Median:
 index              17.5
sno                18.5
regionid            4.0
alcintake2014      82.5
overspeed2014    2561.0
overtaking2014     75.5
lanejumping2014    86.5
wrongside2014     113.0
signalavoid2014     4.5
asleep2014          5.0
othercause2014    156.0
alcintake2016     105.0
overspeed2016    2009.5
signalavoid2016     9.5
wrongside2016     118.0
lanejumping2016    65.5
overtaking2016    175.0
asleep2016          7.5
othercause2016    555.0
dtype: float64
```

```
Variance:
 index                1.110000e+02
sno                  1.110000e+02
regionid             7.542857e+00
alcintake2014        5.884944e+05
overspeed2014        7.530303e+07
overtaking2014       2.647239e+05
lanejumping2014      2.137861e+05
wrongside2014        1.514647e+05
signalavoid2014      6.592857e+03
asleep2014           5.476511e+04
othercause2014       3.171343e+06
alcintake2016        8.240177e+05
overspeed2016        1.366419e+08
signalavoid2016      5.071425e+04
wrongside2016        5.689074e+05
lanejumping2016      2.064571e+05
overtaking2016       2.182989e+06
asleep2016           6.194300e+04
othercause2016       3.714890e+06
dtype: float64
```

```
Standard Deviation:
 index                  10.535654
sno                    10.535654
regionid                2.746426
alcintake2014         767.133866
overspeed2014        8677.731983
overtaking2014        514.513279
lanejumping2014       462.370115
wrongside2014         389.184642
signalavoid2014        81.196411
asleep2014            234.019472
othercause2014       1780.826436
alcintake2016         907.754202
overspeed2016       11689.392631
signalavoid2016       225.198246
wrongside2016         754.259534
lanejumping2016       454.375457
overtaking2016       1477.494162
asleep2016            248.883501
othercause2016       1927.404966
dtype: float64
```

```python
df.hist(bins=20, figsize=(12, 10))
plt.tight_layout()
plt.show()
```





9. Feature Engineering:

1. Extract time-based features (hour, day, month)
2. Create location-based features (latitude, longitude, proximity to intersections)
3. Calculate traffic-related features (traffic volume, speed)
4. Encode categorical variables (weather, road conditions)
5. Derive accident severity features (injury/fatality rates)

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load your data
df = pd.read_csv('/driverresponse.csv')

# Step 1: Select only numeric columns
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Step 2: Initialize the scaler
scaler = StandardScaler()

# Step 3: Fit and transform the numeric data
df_scaled = df.copy()
df_scaled[numeric_cols] = scaler.fit_transform(df[numeric_cols])

# Step 4: See the result
print("\nBefore Scaling (first 5 rows):")
print(df[numeric_cols].head())

print("\nAfter Standardization (first 5 rows):")
print(df_scaled[numeric_cols].head())
```

```
Before Scaling (first 5 rows):
   index  sno  regionid  alcintake2014  overspeed2014  overtaking2014  \
0      0    1         2            594        12747.0             507
1      1    2         5             11           16.0               0
2      2    3         5            613         4596.0             129
3      3    4         1           1680         1496.0             278
4      4    5         9            335         6720.0             188

   lanejumping2014  wrongside2014  signalavoid2014  asleep2014  \
0              328            668               29         154
1                0              0                0           0
2              104            156                7           3
3              236            308                7          72
4              313            266               10          81

   othercause2014  alcintake2016  overspeed2016  signalavoid2016  \
0            1938            128          17286               40
1              22             15             45                0
2             628            352           3520               64
3             439            593           2323                8
4             520            145           6660               62

   wrongside2016  lanejumping2016  overtaking2016  asleep2016  othercause2016
0            667              325            1024         306            2868
1              7               11               8           8              18
2            334              156             371           3             873
3            458              156             573         122             612
4            410              175             467         144             953


After Standardization (first 5 rows):
      index       sno  regionid  alcintake2014  overspeed2014  overtaking2014  \
0 -1.684588 -1.684588 -0.738549       0.090633       0.794634        0.382732
1 -1.588326 -1.588326  0.369274      -0.680118      -0.693873       -0.616643
2 -1.492064 -1.492064  0.369274       0.115752      -0.158380       -0.362364
3 -1.395802 -1.395802 -1.107823       1.526374      -0.520831       -0.068662
4 -1.299540 -1.299540  1.846372      -0.251776       0.089958       -0.246066

   lanejumping2014  wrongside2014  signalavoid2014  asleep2014  \
0         0.097913       0.953768        -0.099924    0.268573
1        -0.621538      -0.786989        -0.462149   -0.398827
2        -0.393419      -0.380465        -0.374715   -0.385825
3        -0.103884       0.015636        -0.374715   -0.086796
4         0.065011      -0.093813        -0.337244   -0.047792

   othercause2014  alcintake2016  overspeed2016  signalavoid2016  \
0        0.529669      -0.319222       0.853043        -0.381673
1       -0.561498      -0.445471      -0.642806        -0.561814
2       -0.216379      -0.068959      -0.341311        -0.273589
3       -0.324015       0.200297      -0.445164        -0.525786
4       -0.277886      -0.300229      -0.068881        -0.282596

   wrongside2016  lanejumping2016  overtaking2016  asleep2016  othercause2016
0       0.237473         0.197598        0.137609    0.731678        0.770550
1      -0.649970        -0.503263       -0.559797   -0.482654       -0.729098
2      -0.210282        -0.179617       -0.310625   -0.503029       -0.279203
3      -0.043550        -0.179617       -0.171968   -0.018111       -0.416539
4      -0.108092        -0.137209       -0.244729    0.071538       -0.237108
```

10. Model Building

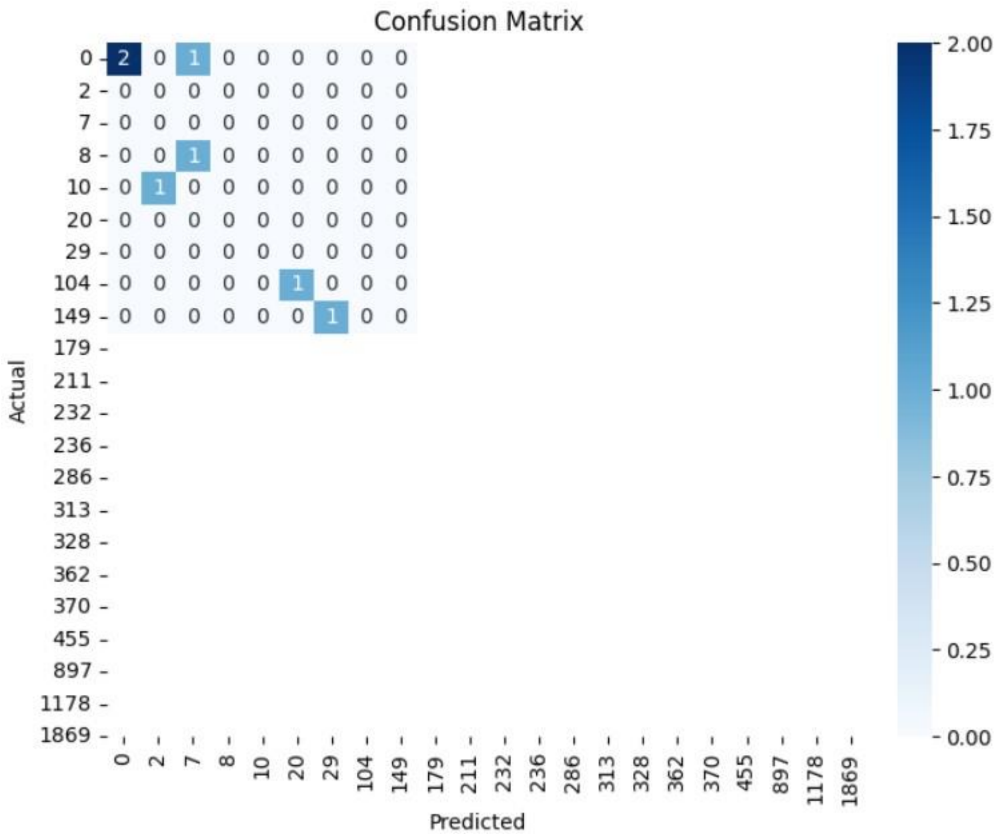
1. Select algorithms: Choose suitable machine learning algorithms (e.g., logistic regression, decision trees, random forest, neural networks)
2. Train models: Train models using the training dataset

3. Hyperparameter tuning: Optimize model hyperparameters for better performance
4. Model evaluation: Evaluate model performance using metrics (accuracy, precision, recall, F1score)
5. Model selection: Select the best-performing model for deployment.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, roc_curve, auc, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import cross_val_score
import numpy as np

# Load the dataset
df = pd.read_csv("/driverresponse.csv")

# Initialize LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

# Encode categorical columns
for col in df.select_dtypes(include='object').columns:
    df[col] = le.fit_transform(df[col])

# Set 'lanejumping2014' as the target column (adjust based on your choice of target column)
X = df.drop('lanejumping2014', axis=1)  # Drop the target column from features
y = df['lanejumping2014']  # This is the target column

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.182, random_state=42)
```

```python
# Initialize and train the Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Make predictions
y_pred = rf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Calculate F1 Score
# Calculate F1 Score - Specify average for multiclass
f1 = f1_score(y_test, y_pred, average='weighted')  # Use 'weighted' or another suitable average

# Calculate RMSE (Root Mean Squared Error)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

# Calculate ROC Curve and AUC (for binary classification)
if len(rf.classes_) == 2:
    y_prob = rf.predict_proba(X_test)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc_score = auc(fpr, tpr)
else:
    auc_score = "N/A"  # Not applicable for multi-class problems

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
```

```python
if len(rf.classes_) == 2:
    plt.figure(figsize=(8,6))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {auc_score:.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('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()

# Cross-validation to validate model performance
cv_scores = cross_val_score(rf, X, y, cv=5)

# Summarize all evaluation metrics in a table format
evaluation_results = {
    'Metric': ['Accuracy', 'F1 Score', 'ROC AUC', 'RMSE'],
    'Value': [accuracy, f1, auc_score, rmse]
}

evaluation_df = pd.DataFrame(evaluation_results)

# Print evaluation metrics and model comparison table
print("Evaluation Metrics and Model Comparison:")
print(evaluation_df)

# Cross-validation scores
print(f"\nCross-Validation Scores: {cv_scores}")
print(f"Mean Cross-Validation Score: {cv_scores.mean()}")
```



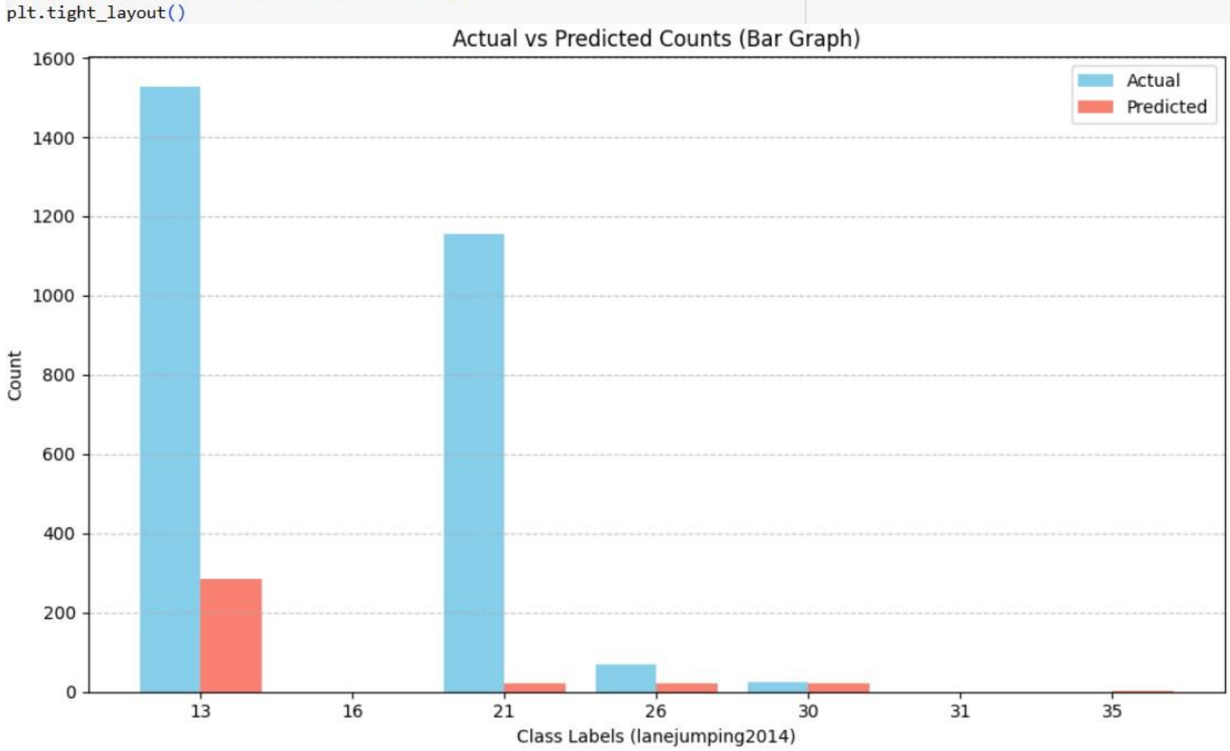
11.Model Evaluation

1.SCIKIT - learn*: Model evaluation metrics (Python)

2. Metrics: Accuracy, precision, recall, F1-score

```
# Load and preprocess dataset
df = pd.read_csv("/driverresponse.csv")
le = LabelEncoder()
for col in df.select_dtypes(include='object').columns:
    df[col] = le.fit_transform(df[col])

# Define features and target
X = df.drop('lanejumping2014', axis=1)
y = df['lanejumping2014']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.182, random_state=42)

# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # or strategy='median', 'most_frequent', 'constant'
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test) # Use the same imputer fitted on training data

# Train model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```python
# Create comparison_df DataFrame
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
comparison_df = comparison_df.groupby(comparison_df.index).first() # Ensure index is unique

# Now you can use comparison_df
labels = comparison_df.index.astype(str)
x = range(len(labels))

plt.figure(figsize=(10, 6))
plt.bar(x, comparison_df['Actual'], width=0.4, label='Actual', align='center', color='skyblue')
plt.bar([i + 0.4 for i in x], comparison_df['Predicted'], width=0.4, label='Predicted', align='center', color='salmon')
plt.xticks([i + 0.2 for i in x], labels)
plt.xlabel('Class Labels (lanejumping2014)')
plt.ylabel('Count')
plt.title('Actual vs Predicted Counts (Bar Graph)')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
```



12. Deployment

Deployment Plan:

1. Integration with Existing Infrastructure: Integrate the AI system with existing traffic management systems, surveillance cameras, and sensors.
2. Real-time Data Feed: Establish a real-time data feed from various sources, including traffic cameras, sensors, and emergency services.
3. Cloud-based Deployment: Deploy the system on a cloud-based platform for scalability, reliability, and maintenance.

13. Source code

```
import pandas as pd
import numpy as numpy import
matplotlib.pyplot as plt import
seaborn as sns
df=pd.read_csv("/driverresponse.csv")
df.head() df.isnull().sum()
df.fillna(0,inplace=True)
df.isnull().sum() df.info() df.describe()
print(df.duplicated().sum()) print("Original
shape:",df.shape)
numeric_df= df.select_dtypes (include=['number']) if
numeric_df.empty:     print("\nNo numeric columns
found in the dataset.") else:
    mean = numeric_df.mean()
median = numeric_df.median()     var
= numeric_df.var()     std =
numeric_df.std()     print("\nMean:\n",
mean)     print("\nMedian:\n", median)
print("\nVariance: \n", var)
print("\nStandard Deviation:\n", std)

for col in df.columns:     print(f"{col}:
{df[col].nunique()} unique values") for col in
df.select_dtypes(include='object').columns:
    print(f"\n{col} value counts:")
print(df[col].value_counts()) print(df.shape)
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler() df_scaled = df.copy()
df_scaled[numeric_cols] = scaler.fit_transform(df[numeric_cols])
print("\nBefore Scaling (first 5 rows):") print(df[numeric_cols].head())
print("\nAfter Standardization (first 5 rows):")
print(df_scaled[numeric_cols].head()) df.hist(bins=20,
figsize=(12, 10))
plt.tight_layout() plt.show() le = LabelEncoder() for
col in df.select_dtypes(include='object').columns:
    df[col] = le.fit_transform(df[col])
df = df.dropna()
X = df.drop('lanejumping2014', axis=1)  # Drop the target column from features y
= df['lanejumping2014']  # This is the target column
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.182, random_state=42) rf
= RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test) class_labels
= sorted(y.unique())
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report: \n", classification_report(y_test, y_pred, labels=class_labels,
zero_division=1))
```

```
from sklearn.preprocessing import LabelEncoder le
= LabelEncoder()

for col in df.select_dtypes(include='object').columns:
    df[col] = le.fit_transform(df[col])
X = df.drop('lanejumping2014', axis=1)  # Drop the target column from features y
= df['lanejumping2014']  # This is the target column
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.182, random_state=42)
rf = RandomForestClassifier(n_estimators=100, random_state=42) rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')  # Use 'weighted' or another suitable average
rmse = np.sqrt(mean_squared_error(y_test, y_pred)) if len(rf.classes_) == 2:     y_prob =
rf.predict_proba(X_test)[:, 1]     fpr, tpr, thresholds = roc_curve(y_test, y_prob)     auc_score =
auc(fpr, tpr) else:     auc_score = "N/A"  # Not applicable for multi-class problems cm =
confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=rf.classes_,
yticklabels=rf.classes_) plt.xlabel('Predicted') plt.ylabel('Actual')
plt.title('Confusion Matrix') plt.show() if len(rf.classes_) == 2:
plt.figure(figsize=(8,6))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {auc_score:.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('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')     plt.show()
cv_scores = cross_val_score(rf, X, y, cv=5) evaluation_results
= {
    'Metric': ['Accuracy', 'F1 Score', 'ROC AUC', 'RMSE'],
    'Value': [accuracy, f1, auc_score, rmse]
}

evaluation_df = pd.DataFrame(evaluation_results) print("Evaluation
Metrics and Model Comparison:") print(evaluation_df)
print(f"\nCross-Validation Scores: {cv_scores}")
print(f"Mean Cross-Validation Score: {cv_scores.mean()}")

from sklearn.model_selection import train_test_split from
sklearn.linear_model import LogisticRegression from
sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer  # Import SimpleImputer
df = pd.read_csv("/driverresponse.csv") le = LabelEncoder() for col
in df.select_dtypes(include='object').columns:
    df[col] = le.fit_transform(df[col])
X = df.drop('lanejumping2014', axis=1) y
= df['lanejumping2014']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.182, random_state=42)
imputer = SimpleImputer(strategy='mean') # or strategy='median', 'most_frequent', 'constant'
```

```
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test) # Use the same imputer fitted on training data

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train) y_pred
= model.predict(X_test)
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
comparison_df = comparison_df.groupby(comparison_df.index).first() # Ensure index is unique
labels = comparison_df.index.astype(str) x = range(len(labels)) plt.figure(figsize=(10, 6))
plt.bar(x, comparison_df['Actual'], width=0.4, label='Actual', align='center', color='skyblue')
plt.bar([i + 0.4 for i in x], comparison_df['Predicted'], width=0.4, label='Predicted', align='center',
color='salmon')
plt.xticks([i + 0.2 for i in x], labels) plt.xlabel('Class
Labels (lanejumping2014)')
plt.ylabel('Count')
plt.title('Actual vs Predicted Counts (Bar Graph)') plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7) plt.tight_layout()
plt.show()
```

14. Future scope

1. Reduced Accidents: Further reduction in traffic accidents and fatalities.
2. Improved Traffic Flow: Enhanced traffic flow and reduced congestion.
3. Data-Driven Policy Making: Informed policy decisions for urban planning, transportation, and safety.
4. Increased Efficiency: Improved emergency response times and resource allocation.

15. Team Members and Roles:

Google colab:

h
t
t
p
s
:
/
/
c
o
l
a
b
.

research.google.com/notebooks/intro.ipynb

SN NO	NAMES	ROLES	RESPONSIBILITY
1.	Silpha S	Team Leader	Data Collection & Data Preprocessing

2.	Sowparnikashree P	Team Member	Exploratory Data Analysis & Feature Engineering
3.	Shalini S	Team Member	Model Building & Model Evaluation
4.	Thiriveni N	Team Member	Visualization Interpretion , Deployment