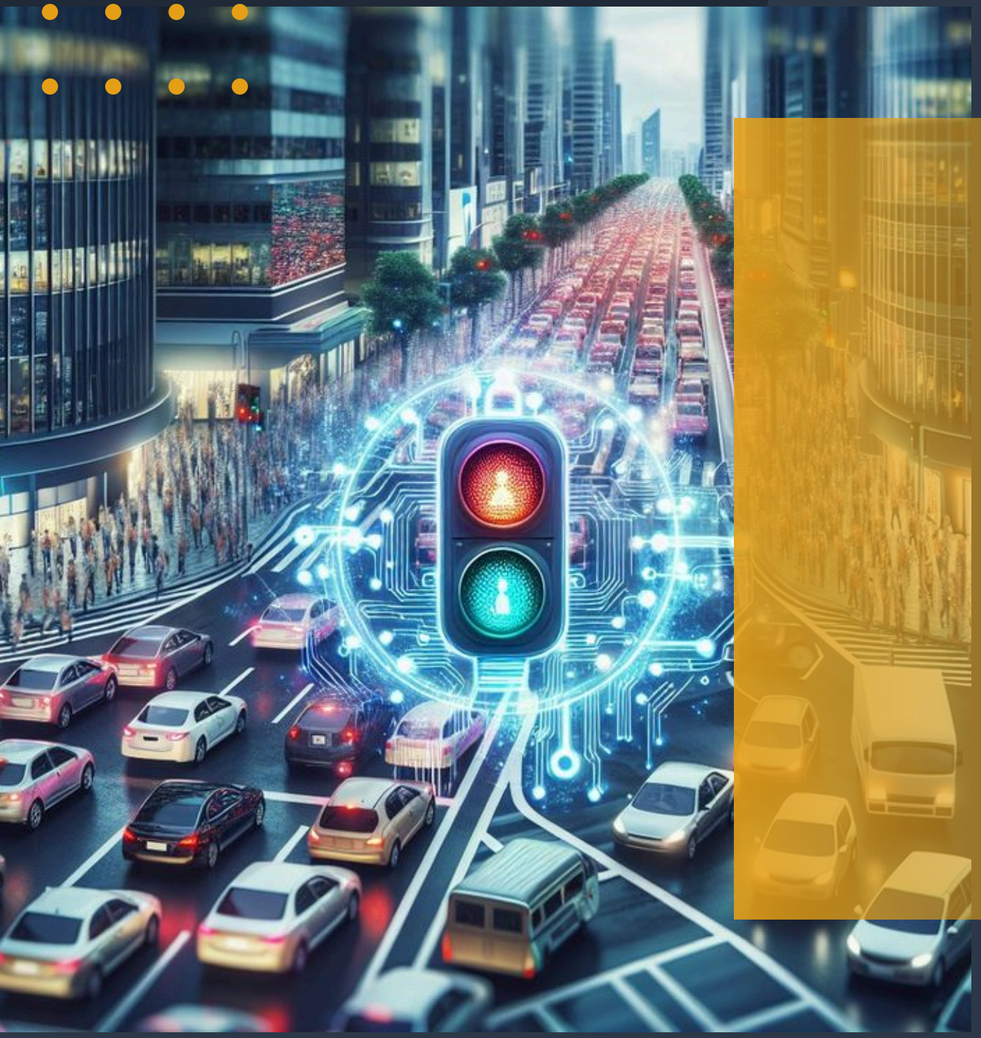
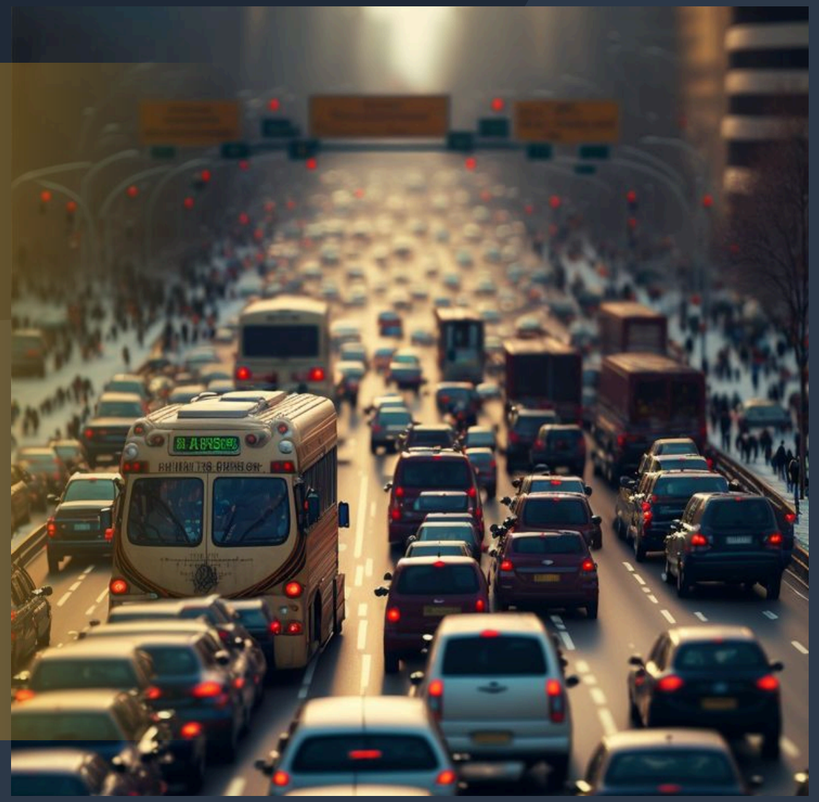
PREDICTING ACCIDENT SEVERITY BASED ON DRIVER BEHAVIORS AND ENVIRONMENTAL FACTORS

CCS1304 - Business Intelligence

TABLE OF CONTENTS

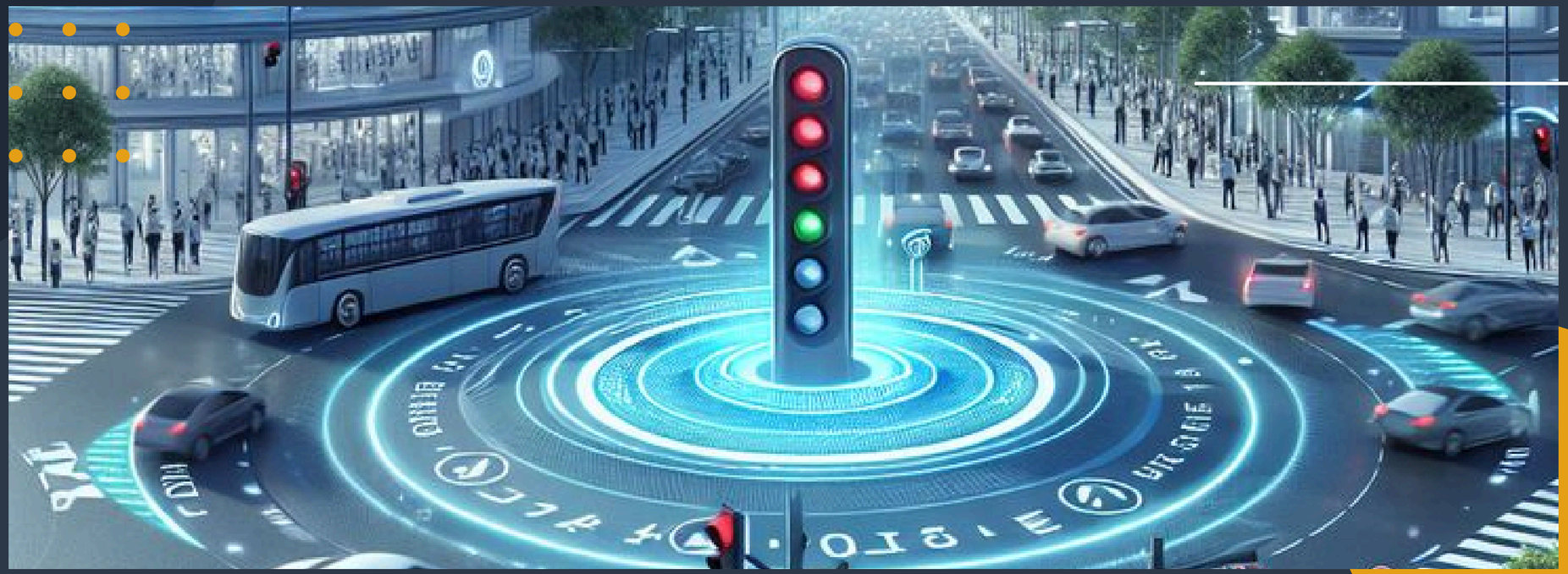




PROBLEM STATEMENT

Traffic accidents pose a major threat to public safety. Understanding the factors influencing accident severity can help authorities implement proactive safety measures.

The study applies machine learning to predict accident severity (Low, Moderate, High) based on environmental and driver-related factors.



PROBLEM DESCRIPTION

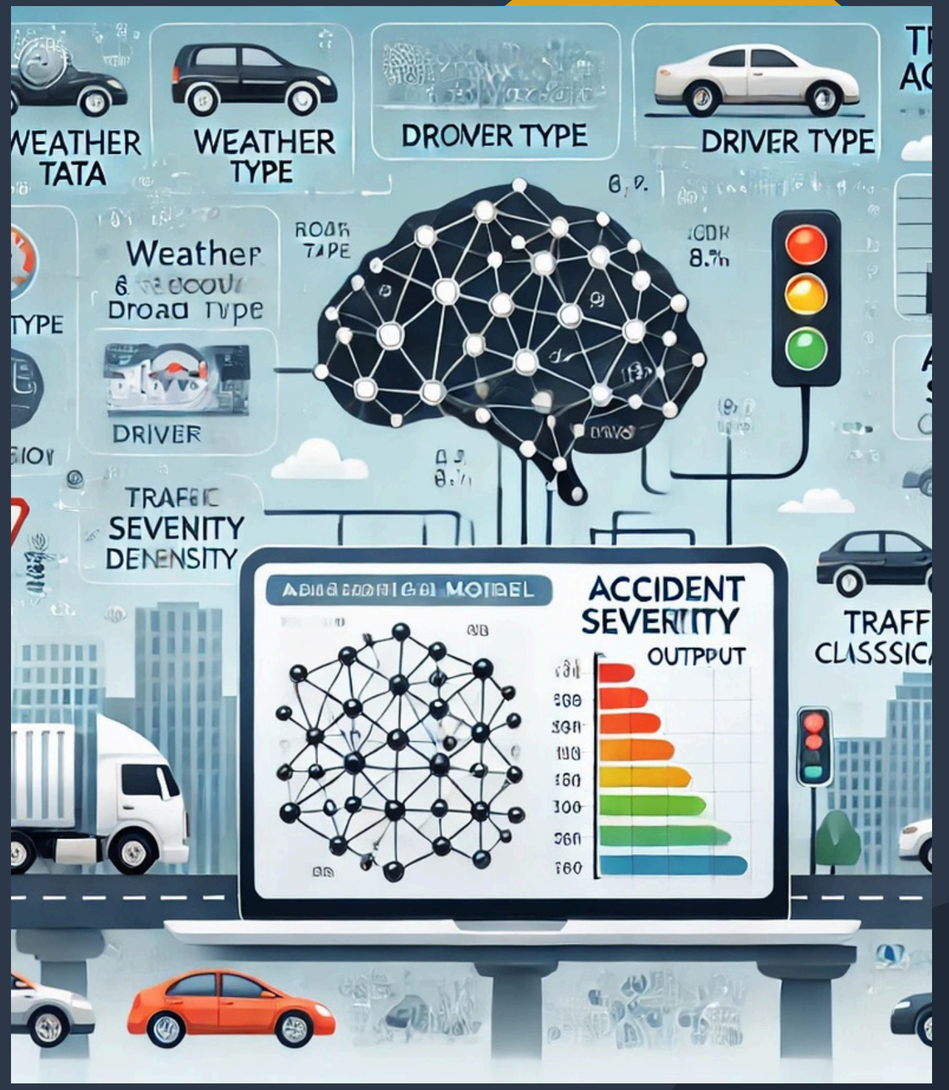
Accident severity depends on multiple factors: road conditions, traffic density, driver behavior, and vehicle type.

Traditional methods rely on historical reports, while machine learning provides a scalable, accurate approach.

ABOUT THE DATASET

- Categorical variables: Weather, Road Type, Time of Day, Accident Severity, Road Condition, Vehicle Type, Road Light Condition.

  Numerical variables: Traffic Density, Speed Limit, Driver Age, Number of Vehicles, Driver Alcohol, Driver Experience



METHODOLOGY

Data Preprocessing

```
#Filling Numerical values

df['Traffic_Density'].fillna(df['Traffic_Density'].median(), inplace=True)
df['Speed_Limit'].fillna(df['Speed_Limit'].median(), inplace=True)
df['Number_of_Vehicles'].fillna(df['Number_of_Vehicles'].median(), inplace=True)
df['Driver_Alcohol'].fillna(df['Driver_Alcohol'].median(), inplace=True)
df['Driver_Age'].fillna(df['Driver_Age'].median(), inplace=True)
df['Driver_Experience'].fillna(df['Driver_Experience'].median(), inplace=True)
df['Accident'].fillna(df['Accident'].median(), inplace=True)
```

```
#Filling categorical data

df['Weather'] = df['Weather'].fillna(df['Weather'].mode()[0])
df['Road_Type'] = df['Road_Type'].fillna(df['Road_Type'].mode()[0])
df['Time_of_Day'] = df['Time_of_Day'].fillna(df['Time_of_Day'].mode()[0])
df['Accident_Severity'] = df['Accident_Severity'].fillna(df['Accident_Severity'].mode()[0])
df['Road_Condition'] = df['Road_Condition'].fillna(df['Road_Condition'].mode()[0])
df['Vehicle_Type'] = df['Vehicle_Type'].fillna(df['Vehicle_Type'].mode()[0])
df['Road_Light_Condition'] = df['Road_Light_Condition'].fillna(df['Road_Light_Condition'].mode()[0])
```

Handling Missing Values

Check for missing data and apply appropriate imputation techniques.

Numerical values are filled using the median to reduce the effect of outliers.

Categorical values are filled using the mode to maintain consistency.

Handling Duplicates

```
[ ]  duplicates_count = df.duplicated().sum()
     print(f"Duplicates count: {duplicates_count}")
```

```
Duplicates count: 18
```

```
[ ]  df.drop_duplicates(inplace=True)
```

```
[ ]  duplicates_count = df.duplicated().sum()
     print(f"Duplicates count: {duplicates_count}")
```

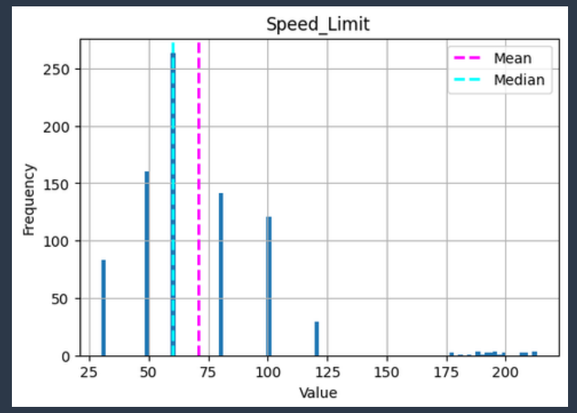
```
Duplicates count: 0
```

Encoding Categorical Variables

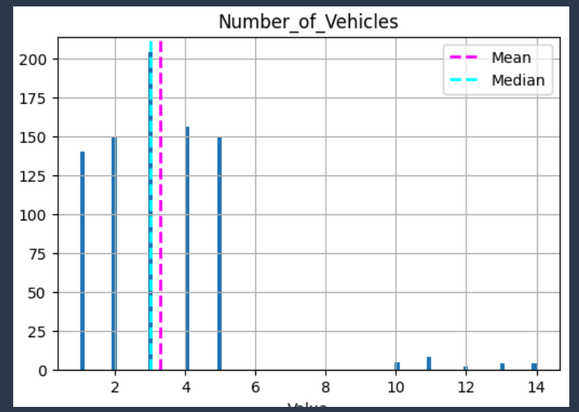
```
from sklearn.preprocessing import OneHotEncoder

# One-hot encoding for categorical features
X_environmental = pd.get_dummies(X_environmental, drop_first=True)
X_driver = pd.get_dummies(X_driver, drop_first=True)
```

Data Visualization

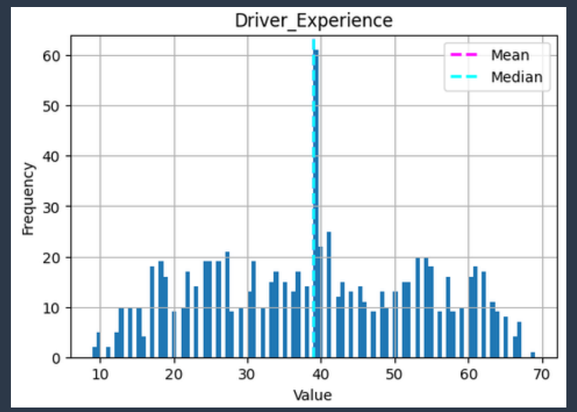
```python
# Numerical columns
Numerical_col = ['Speed_Limit', 'Number_of_Vehicles','Driver_Age', 'Driver_Experience', 'Accident']

# Plot a histogram for each numeric column
for col in Numerical_col:
    fig = plt.figure(figsize=(6, 4))
    ax = fig.gca()
    feature = df[col]

    # Plot histogram
    feature.hist(bins=100, ax=ax)

    # Add vertical lines for mean and median
    ax.axvline(feature.mean(), color='magenta', linestyle='dashed', linewidth=2, label="Mean")
    ax.axvline(feature.median(), color='cyan', linestyle='dashed', linewidth=2, label="Median")

    # Titles and labels
    ax.set_title(col)
    ax.set_xlabel("Value")
    ax.set_ylabel("Frequency")  # Set y-axis label
    ax.legend()

plt.show()
```



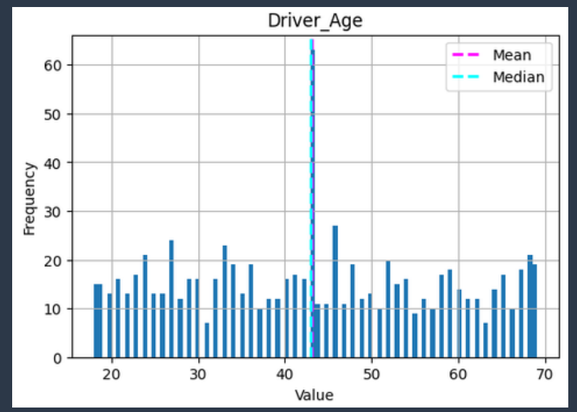
Shows the frequency of different speed limits, with vertical lines marking the mean and median



Displays the distribution of the number of vehicles involved in incidents



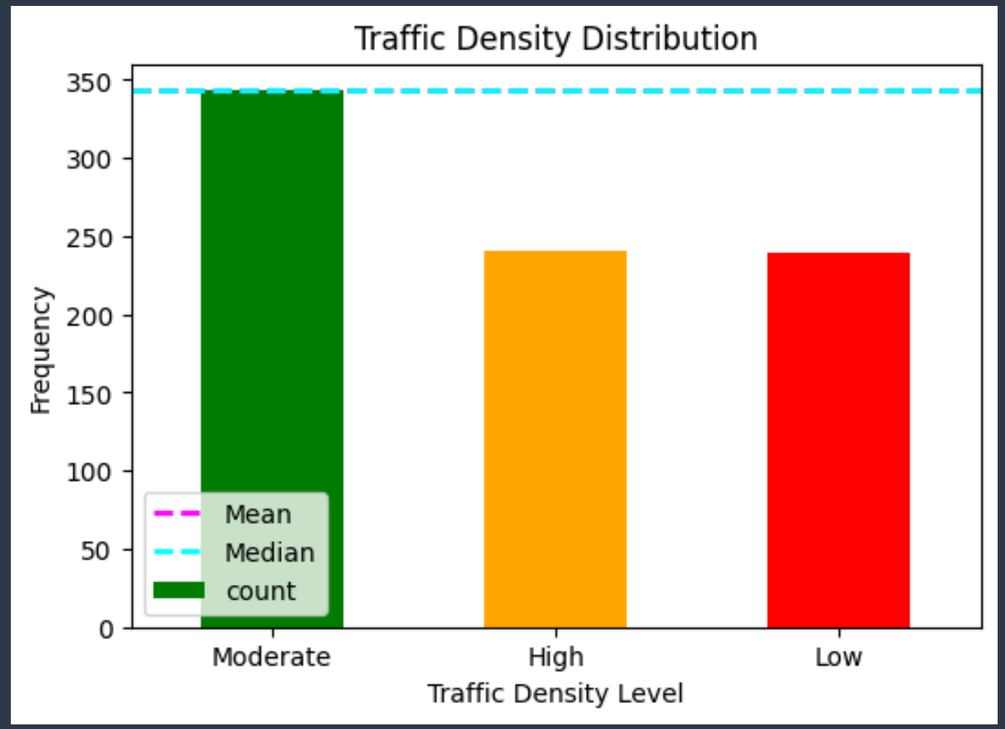
Represents the distribution of driver experience values



Shows the frequency of various driver ages

Plot for Traffic Density distribution

```python
# Define category labels
traffic_labels = {0: 'Low', 1: 'Moderate', 2: 'High'}

# Map numeric values to categorical labels
df['Traffic_Density_Category'] = df['Traffic_Density'].map(traffic_labels)

# Count occurrences of each category
density_counts = df['Traffic_Density_Category'].value_counts()

# Plot bar chart
fig, ax = plt.subplots(figsize=(6, 4))
bars = density_counts.plot(kind='bar', color=['green', 'orange', 'red'], ax=ax)

# Calculate mean and median as numerical values
mean_value = df['Traffic_Density'].mean()
median_value = df['Traffic_Density'].median()

# Convert numerical mean/median to category positions
mean_category = traffic_labels[int(round(mean_value))]
median_category = traffic_labels[int(round(median_value))]

# Add mean and median lines
ax.axhline(y=density_counts[mean_category], color='magenta', linestyle='dashed', linewidth=2, label='Mean')
ax.axhline(y=density_counts[median_category], color='cyan', linestyle='dashed', linewidth=2, label='Median')

# Labels and title
ax.set_title("Traffic Density Distribution")
ax.set_xlabel("Traffic Density Level")
ax.set_ylabel("Frequency")
plt.xticks(rotation=0)
```

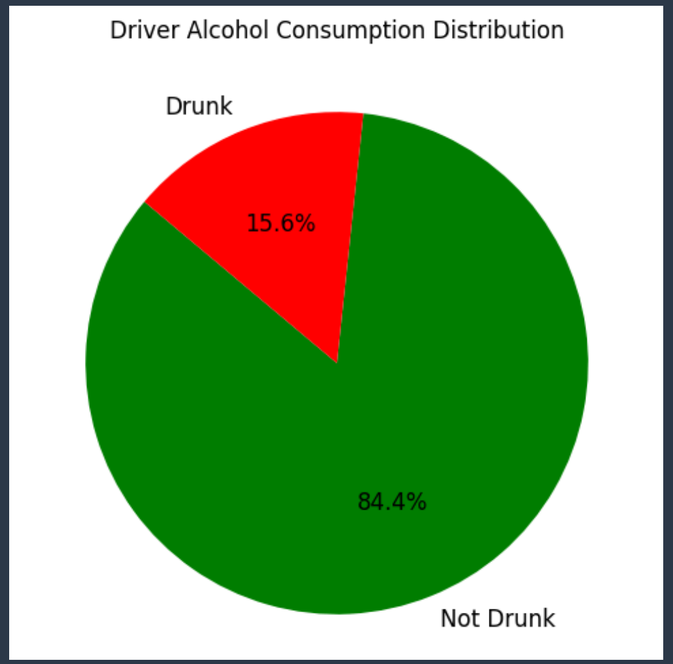


The image displays a Traffic Density Distribution plot, displaying the frequency of different traffic density levels, with distinct colors and mean and median indicators, generated using Python code.

Pie chart for driver alcohol consumption distribution

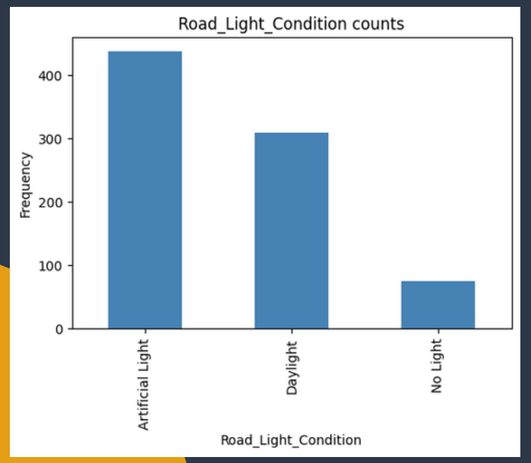
```python
import matplotlib.pyplot as plt

# Define category labels
alcohol_labels = {0: 'Not Drunk', 1: 'Drunk'}
# Map numeric values to categorical labels
df['Driver_Alcohol_Category'] = df['Driver_Alcohol'].map(alcohol_labels)

# Count occurrences of each category
alcohol_counts = df['Driver_Alcohol_Category'].value_counts()

# Define colors: Green for "Not Drunk", Red for "Drunk"
colors = {'Not Drunk': 'green', 'Drunk': 'red'}

# Plot pie chart
fig, ax = plt.subplots(figsize=(6, 6))
wedges, texts, autotexts = ax.pie(
    alcohol_counts, labels=alcohol_counts.index, autopct='%1.1f%%',
    colors=[colors[label] for label in alcohol_counts.index],
    startangle=140, textprops={'fontsize': 12}
)

# Title
ax.set_title("Driver Alcohol Consumption Distribution")
plt.show()
```

- The image displays a pie chart displaying the distribution of driver alcohol consumption
- color-coded for drunk and non-drunk drivers
- Python code using Matplotlib for visualization.

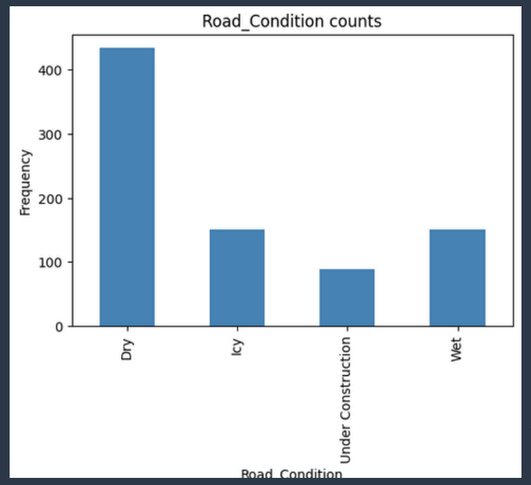


Plots for each categorical Feature

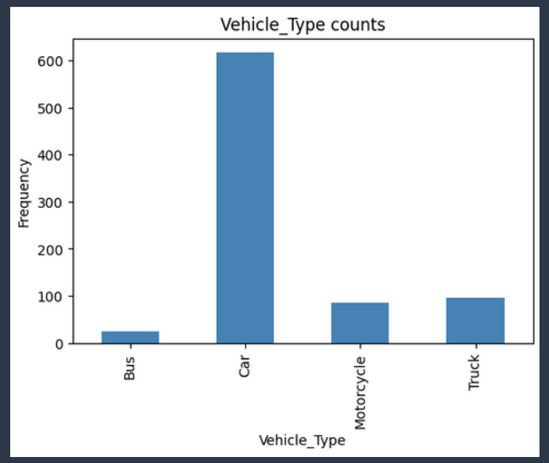
```python
# Plot a bar plot for each categorical feature count
categorical_features = ['Weather', 'Road_Type', 'Time_of_Day', 'Accident_Severity', 'Road_Condition', 'Vehicle_Type', 'Road_Light_Condition']

for col in categorical_features:
    counts = df[col].value_counts().sort_index()
    fig = plt.figure(figsize=(6, 4))
    ax = fig.gca()
    counts.plot.bar(ax = ax, color='steelblue')
    ax.set_title(col + ' counts')
    ax.set_xlabel(col)
    ax.set_ylabel("Frequency")
plt.show()
```



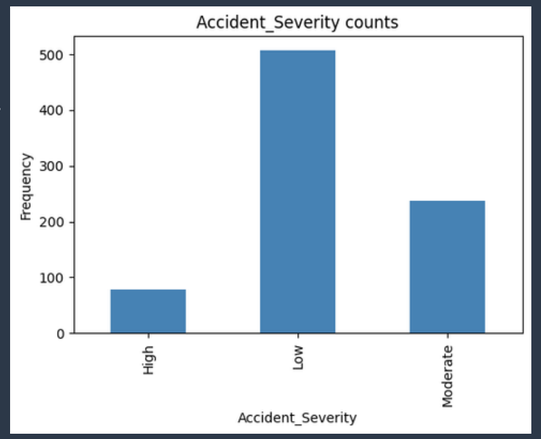
Displays the frequency of different lighting conditions



Shows the count of various road conditions

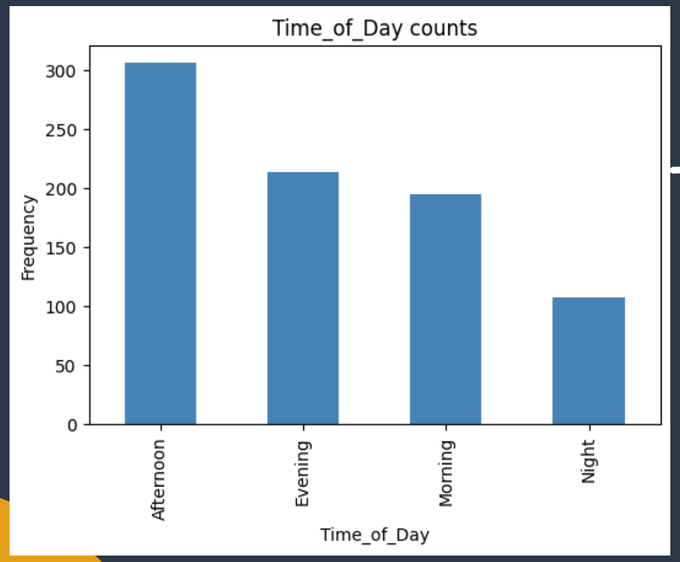


Illustrates the distribution of different types of vehicles

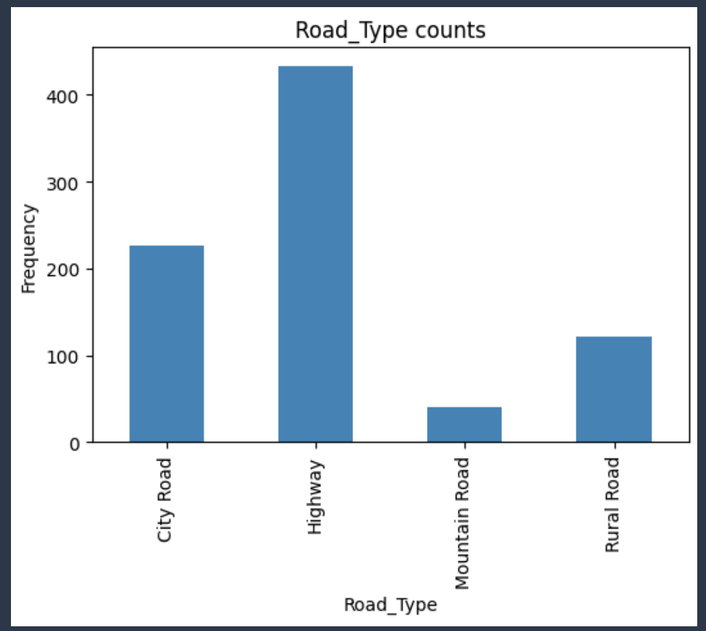


Represents the count of accidents based on severity levels

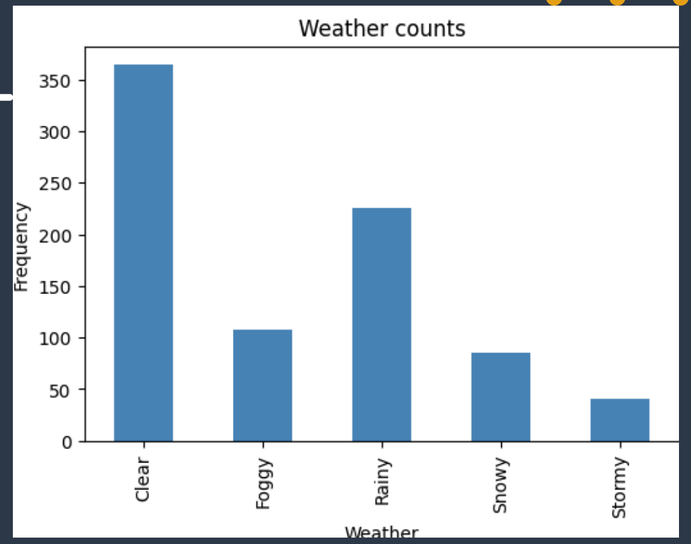
Plots for each categorical Feature



graph displays data distribution across different times of the day, categorizing it into morning, afternoon, evening, and night or using hourly intervals



Illustrates the count of different road types in the dataset



Highlights the frequency of various weather conditions

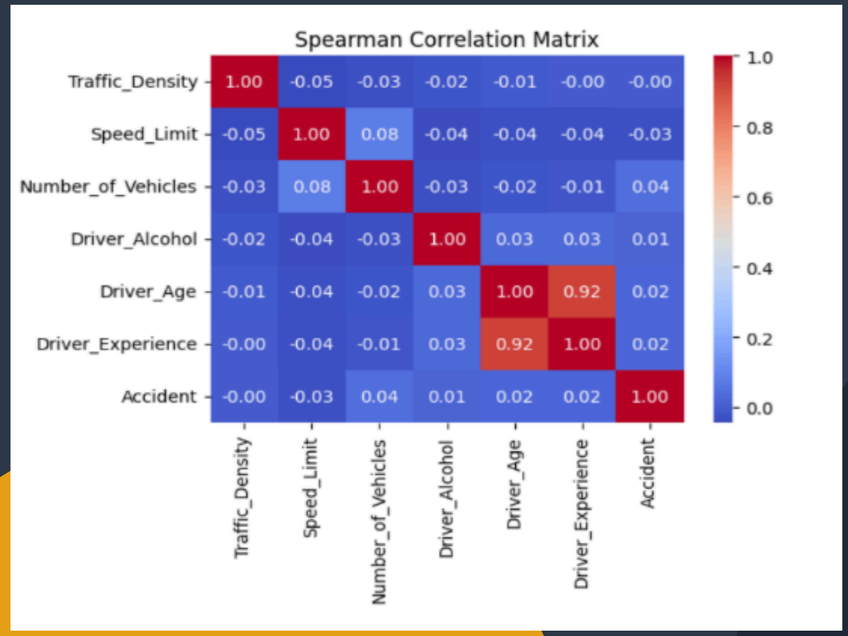
Analyze Correlations

```
numeric_df = df.select_dtypes(include=['number'])
spearman_corr = numeric_df.corr(method='spearman')

# printing corralation of numerical values
print(spearman_corr)

#Heat map for Numerical values
plt.figure(figsize=(6, 4))
sns.heatmap(spearman_corr, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Spearman Correlation Matrix')
plt.show()
```

	Traffic_Density	Speed_Limit	Number_of_Vehicles	\
Traffic_Density	1.000000	-0.045265	-0.030647	
Speed_Limit	-0.045265	1.000000	0.076297	
Number_of_Vehicles	-0.030647	0.076297	1.000000	
Driver_Alcohol	-0.022652	-0.037757	-0.030421	
Driver_Age	-0.005930	-0.036819	-0.024255	
Driver_Experience	-0.002173	-0.035221	-0.013461	
Accident	-0.001008	-0.028651	0.036608	

	Driver_Alcohol	Driver_Age	Driver_Experience	Accident
Traffic_Density	-0.022652	-0.005930	-0.002173	-0.001008
Speed_Limit	-0.037757	-0.036819	-0.035221	-0.028651
Number_of_Vehicles	-0.030421	-0.024255	-0.013461	0.036608
Driver_Alcohol	1.000000	0.025207	0.026769	0.012788
Driver_Age	0.025207	1.000000	0.924057	0.022875
Driver_Experience	0.026769	0.924057	1.000000	0.022151
Accident	0.012788	0.022875	0.022151	1.000000

A Spearman correlation analysis of numerical features in a dataset, using a codesnippet, correlation matrix table, and heatmap. It helps understand dependencies between features, such as traffic density and driver experience influencing accident rates.



Model Selection and Training

Separate Features for the Dataset

```
environmental_features = ['Weather', 'Road_Type', 'Time_of_Day',
                          'Road_Condition', 'Traffic_Density' ,'Road_Light_Condition']
X_environmental = df[environmental_features]
y = df['Accident_Severity']

driver_features = ['Vehicle_Type','Driver_Alcohol', 'Driver_Age', 'Driver_Experience']
X_driver = df[driver_features]
y = df['Accident_Severity']
```

First separate features as environmental and driver features then train and split separated data. After that following machine learning models used to predict accident severity

Train Test and split Data

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

# One-hot encoding for categorical features
X_environmental = pd.get_dummies(X_environmental, drop_first=True)
X_driver = pd.get_dummies(X_driver, drop_first=True)

# Split data into train and test sets (80% train, 20% test)
X_env_train, X_env_test, y_train, y_test = train_test_split(X_environmental, y, test_size=0.2, random_state=42)
X_driver_train, X_driver_test, y_train, y_test = train_test_split(X_driver, y, test_size=0.2, random_state=42)
```

ML model 1 - Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Logistic Regression for environmental features
log_reg_env = LogisticRegression(max_iter=1000)
log_reg_env.fit(X_env_train, y_train)
y_pred_env_log_reg = log_reg_env.predict(X_env_test)
acc_env_log_reg = accuracy_score(y_test, y_pred_env_log_reg)

# Logistic Regression for driver features
log_reg_driver = LogisticRegression(max_iter=1000)
log_reg_driver.fit(X_driver_train, y_train)
y_pred_driver_log_reg = log_reg_driver.predict(X_driver_test)
acc_driver_log_reg = accuracy_score(y_test, y_pred_driver_log_reg)

# Print Accuracy
print("Logistic Regression Accuracy (Environmental):", acc_env_log_reg)
print("Logistic Regression Accuracy (Driver):", acc_driver_log_reg)

# Print Classification Reports
print("\nClassification Report (Environmental Factors):\n", classification_report(y_test, y_pred_env_log_reg))
print("\nClassification Report (Driver Behavior):\n", classification_report(y_test, y_pred_driver_log_reg))
```

ML model 2 - Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier

# Random Forest for environmental features
rf_env = RandomForestClassifier(random_state=42)
rf_env.fit(X_env_train, y_train)
y_pred_env_rf = rf_env.predict(X_env_test)
print("Random Forest Accuracy (Environmental):", accuracy_score(y_test, y_pred_env_rf))

# Random Forest for driver features
rf_driver = RandomForestClassifier(random_state=42)
rf_driver.fit(X_driver_train, y_train)
y_pred_driver_rf = rf_driver.predict(X_driver_test)
print("Random Forest Accuracy (Driver):", accuracy_score(y_test, y_pred_driver_rf))

print("\nClassification Report (Environmental Factors):\n", classification_report(y_test, y_pred_env_rf))
print("\nClassification Report (Driver Behavior):\n", classification_report(y_test, y_pred_driver_rf))
```

ML Model 3 - Dessiosion tree classifier

```python
# Decision Tree for environmental features
dt_env = DecisionTreeClassifier(random_state=42)
dt_env.fit(X_env_train, y_train)
y_pred_env_dt = dt_env.predict(X_env_test)
print("Decision Tree Accuracy (Environmental):", accuracy_score(y_test, y_pred_env_dt))

# Decision Tree for driver features
dt_driver = DecisionTreeClassifier(random_state=42)
dt_driver.fit(X_driver_train, y_train)
y_pred_driver_dt = dt_driver.predict(X_driver_test)
print("Decision Tree Accuracy (Driver):", accuracy_score(y_test, y_pred_driver_dt))

# Print Classification Reports
print("\nClassification Report (Environmental Factors):\n", classification_report(y_test, y_pred_env_dt))
print("\nClassification Report (Driver Behavior):\n", classification_report(y_test, y_pred_driver_dt))
```

Model Evaluation

Performance Metrics like Accuracy, Precision, Recall, and F1-score to evaluate Each model and to select the most accurate one.

```
from sklearn.metrics import accuracy_score, classification_report
```

This is Model Evaluation stage in machine learning workflow, emphasizing the importance of assessing a model's performance using key metrics like accuracy, precision, recall, and F1-score. It explains that these metrics are used to evaluate each model and choose the most accurate one. The slide also provides a codesnippet in Python using the scikit-learn library to calculate these metrics.

RESULTS

ML model 1 - Logistic Regression

```
Logistic Regression Accuracy (Environmental): 0.6606060606060606
Logistic Regression Accuracy (Driver): 0.6787878787878788

Classification Report (Environmental Factors):
              precision    recall  f1-score   support

        High       0.00      0.00      0.00        10
         Low       0.68      0.97      0.80       112
    Moderate       0.00      0.00      0.00        43

    accuracy                           0.66       165
   macro avg       0.23      0.32      0.27       165
weighted avg       0.46      0.66      0.54       165


Classification Report (Driver Behavior):
              precision    recall  f1-score   support

        High       0.00      0.00      0.00        10
         Low       0.68      1.00      0.81       112
    Moderate       0.00      0.00      0.00        43

    accuracy                           0.68       165
   macro avg       0.23      0.33      0.27       165
weighted avg       0.46      0.68      0.55       165
```

ML model 2 - Random Forest Classifier

```
Random Forest Accuracy (Environmental): 0.6121212121212121
Random Forest Accuracy (Driver): 0.503030303030303

Classification Report (Environmental Factors):
              precision    recall  f1-score   support

        High       0.00      0.00      0.00        10
         Low       0.69      0.81      0.75       112
    Moderate       0.40      0.23      0.29        43

    accuracy                           0.61       165
   macro avg       0.36      0.35      0.35       165
weighted avg       0.58      0.61      0.59       165


Classification Report (Driver Behavior):
              precision    recall  f1-score   support

        High       0.00      0.00      0.00        10
         Low       0.65      0.65      0.65       112
    Moderate       0.24      0.23      0.24        43

    accuracy                           0.50       165
   macro avg       0.30      0.29      0.30       165
weighted avg       0.51      0.50      0.50       165
```

ML Model 3 - Dessiosion tree classifier

```
Decision Tree Accuracy (Environmental): 0.509090909090909
Decision Tree Accuracy (Driver): 0.4909090909090909

Classification Report (Environmental Factors):
              precision    recall  f1-score   support

        High       0.05      0.10      0.07        10
         Low       0.67      0.64      0.65       112
    Moderate       0.30      0.26      0.28        43

    accuracy                           0.51       165
   macro avg       0.34      0.33      0.33       165
weighted avg       0.53      0.51      0.52       165


Classification Report (Driver Behavior):
              precision    recall  f1-score   support

        High       0.00      0.00      0.00        10
         Low       0.68      0.61      0.64       112
    Moderate       0.30      0.30      0.30        43

    accuracy                           0.49       165
   macro avg       0.33      0.30      0.31       165
weighted avg       0.54      0.49      0.51       165
```

- Performance metrics: Accuracy, Precision, Recall, F1-score.
- Best-performing model: Logistic Regression
  - 66.1% accuracy (Environmental Factors)
  - 67.78% accuracy (Driver Behaviors)

DISCUSSION

Limitations

- Limited feature scope (lack of driver fatigue, distractions, etc.).
- Class imbalance affects severe accident predictions.
- Potential dataset biases.

Future Improvements

- Collect more real-world accident data.
- Explore deep learning models.
- Implement real-time prediction.
- Integrate geospatial analysis.

CONCLUSION

The study developed a machine learning model to predict accident severity using environmental, traffic, and driver-related factors. Driver behaviors and environmental conditions were significant predictors. Logistic Regression was the best model, with future work focusing on real-time analysis and intelligent transportation systems.

22ug2-0023	Thavisha Nipun	22ug2-0023@sltc.ac.lk
22ug2-0119	Sanduni Chethana	22ug2-0119@sltc.ac.lk
22ug2-0017	Deshani Wijewardhana	22ug2-0017@sltc.ac.lk
22ug2-0271	Muthuni Nimshi	22ug2-0271@sltc.ac.lk
22ug2-0068	Nethulya Sooriarachchi	22ug2-0068@sltc.ac.lk

THANK YOU