King Saud University

College of Computer and Information Sciences

Information Technology Department

IT326: Data Mining

Section: 71680

1st Semester 1446 H

Car Crash Data Mining

Name	ID
Noor Algumlas	444200811
Lama Albugami	444201026
Hour Alajaji	444203559
Alanoud Alshayea	444200869
Danyh Alotaibi	444201030

Table of Contents

1.Problem

Car crashes are a serious issue that impact people's lives every day, causing injuries, loss of life, and financial challenges. While there is a lot of data collected about crashes, it's often difficult to identify clear patterns and causes due to the variety of factors involved, like when and where the crash happened or what caused it.

In this project, we will analyze a car crash dataset with details such as the date, time, location, type of collision, and primary causes. Our goal is to find trends, like the most dangerous times or locations for crashes, and understand what factors contribute the most to accidents.

This is important because understanding these patterns can help improve road safety by giving policymakers and planners the information they need to make better decisions. By reducing crashes, we can help save lives and lower the overall impact of accidents on society.

2.Data Mining Task

The car crash analysis involves two key data mining tasks classification and clustering. The classification task focuses on predicting the *Risk Level* (Low Risk, Moderate Risk, High Risk) as the class attribute. And build an accurate model for assessing crash severity based on features such as time, location, collision type, and primary factors. Clustering analysis was performed on the preprocessed dataset using **K-Means** clustering. This technique was chosen for its simplicity and efficiency in identifying natural groupings within the data and then these clusters will help predict new results.

.

Goals of Data Mining Tasks:

- Develop a classification model to predict crash severity levels based on critical factors.
- Identify natural groupings in the data to reveal high-risk zones, time periods, or contributing factors.
- Optimize the classification and clustering models to ensure high accuracy and meaningful insights.

- Provide actionable recommendations to improve road safety and reduce accidents.

3.Data

Dataset Description

Source: Kaggle (Car Accidents in USA, specifically LA-Monroe)

Dataset Name: Monroe County Car Crashes (2003–2015)

Number of objects: 53,943

Number of attributes: 11 (plus the derived *Risk Level* class label)

This dataset contains detailed information on car crashes, including the year, month, day, time, collision type, injury type, primary factors, location, and more. The dataset is structured with both numerical and categorical attributes, with key features like *Risk Level* serving as a derived class label for classification tasks.

Main Characteristics of Attributes

Data Types:

Numerical Attributes: Year, Month, Day, Hour, Latitude, Longitude (continuous variables)

Categorical Attributes: Weekend?, Collision Type, Injury Type, Primary Factor, Reported_Location(discrete categories)

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from sklearn.feature_selection import VarianceThreshold, SelectFromModel
from sklearn.model_selection import train_test_split

# Read the dataset
data1 = pd.read_csv("/workspaces/IT326/Dataset/Original_Dataset.csv", encoding="ISO-8859-1")
df = pd.DataFrame(data1)  # Convert data1 to DataFrame

# Display basic information about the dataset
print(df.info())
print(df.describe())

# Class label to classify collision risk
def classify_collision(row):
    # Indented code block for the function body
    if (row['Hour'] >= 1600 and row['Hour'] <= 2000) or (row['Hour'] >= 700 and row['Hour'] <= 900)
    or (row['Weekend?'] == 'Weekend' and row['Injury Type'] == 'Incapacitating'):
        return 'High Risk'
    elif row['Injury Type'] == 'No injury/unknown':
        return 'Moderate Risk'
    else:
        return 'Low Risk'

# Apply the function to the DataFrame
data1['Risk Level'] = data1.apply(classify_collision, axis=1)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53943 entries, 0 to 53942
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Year               53943 non-null  int64
 1   Month              53943 non-null  int64
 2   Day                53943 non-null  int64
 3   Weekend?           53875 non-null  object
 4   Hour               53718 non-null  float64
 5   Collision Type     53937 non-null  object
 6   Injury Type        53943 non-null  object
 7   Primary Factor     52822 non-null  object
 8   Reported_Location  53908 non-null  object
 9   Latitude           53913 non-null  float64
 10  Longitude          53913 non-null  float64
dtypes: float64(3), int64(3), object(5)
memory usage: 4.5+ MB
None
               Year         Month           Day          Hour      Latitude  \
count  53943.000000  53943.000000  53943.000000  53718.000000  53913.000000
mean    2008.968059      6.662162      4.196912   1347.265349     35.582109
std        3.789760      3.514630      1.909440    531.654039     11.289883
min     2003.000000      1.000000      1.000000      0.000000      0.000000
25%     2006.000000      4.000000      3.000000   1000.000000     39.142048
50%     2009.000000      7.000000      4.000000   1400.000000     39.164430
75%     2012.000000     10.000000      6.000000   1700.000000     39.173344
max     2015.000000     12.000000      7.000000   2300.000000     41.228665

          Longitude
count  53913.000000
mean     -78.619224
std       24.957587
min      -88.959213
25%      -86.551520
50%      -86.530992
75%      -86.508288
max       86.596363
```

Missing values:

Column-Wise Missing Values: The missing_values variable shows the count of missing values for each column in data1. This helps identify specific columns that may require data cleaning or imputation.

Total Missing Values: By summing the values in missing_values, the total number of missing entries in the entire dataset is obtained. This provides an overview of the extent of missing data and the potential impact on data analysis.

```
# Missing Values
missing_values = data1.isna().sum()
print("\nMissing values in each column:\n", missing_values)
print("\nTotal number of missing values:\n", missing_values.sum())
```

```
Missing values in each column:
 Year                    0
Month                    0
Day                      0
Weekend?                68
Hour                   225
Collision Type           6
Injury Type              0
Primary Factor        1121
Reported_Location       35
Latitude                30
Longitude               30
Risk Level               0
dtype: int64

Total number of missing values:
 1515
```

Statistical measures:

Below are the summary statistics for each numerical column, which help us understand the central tendency and distribution of the data. As we can see the mean for Year column is 2008 which indicates that most crashes fall under this period. The mean Hour of 1347 (1:47 PM) implies that incidents are more frequent in the afternoon. The meadian for the Hour column is 1400 (2:00 PM) which aligns closely to the mean and confirms most accidents happen in the afternoon. The large variance in Hour (282,656) shows that incidents occur throughout the day. On the contrary the year variance (14.36) is relatively low, indicating that incidents occur within a tight range of years. The mode shows us that the most frequent month that accidents occured at is October and the day is the 6th. Lastly, The standard deviation for the year (3.79) indicates incidents occur within a few years around 2008.

```python
# Statistical summaries (Central tendency measurements such as mean and variance)
numeric_data = data1.select_dtypes(include=['number'])

print("\nMean:\n", numeric_data.mean())
print("\nMedian:\n", numeric_data.median())
print("\nVariance:\n", numeric_data.var())
print("\nstander deviation:\n", numeric_data.std())
print("\nmode:\n", numeric_data.mode())
```

```
Mean:
 Year         2008.968059
Month           6.662162
Day             4.196912
Hour         1347.265349
Latitude       35.582109
Longitude     -78.619224
dtype: float64

Median:
 Year         2009.000000
Month           7.000000
Day             4.000000
Hour         1400.000000
Latitude       39.164430
Longitude     -86.530992
dtype: float64

Variance:
 Year             14.362277
Month            12.352624
Day               3.645961
Hour         282656.017271
Latitude        127.461469
Longitude       622.881135
dtype: float64

stander deviation:
 Year           3.789760
Month          3.514630
Day            1.909440
Hour         531.654039
Latitude      11.289883
Longitude     24.957587
dtype: float64

mode:
    Year  Month  Day    Hour  Latitude  Longitude
0  2003     10    6  1700.0       0.0        0.0
```

Outliers:

```
outliers = {}

for column in numeric_columns:
    # Calculate Q1 (25th percentile) and Q3 (75th percentile)
    Q1 = data1[column].quantile(0.25)
    Q3 = data1[column].quantile(0.75)

    # Calculate the IQR
    IQR = Q3 - Q1

    # Define the lower and upper bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers[column] = data1[(data1[column] < lower_bound) | (data1[column] > upper_bound)]

# Display outliers for each column
for column, outlier_data in outliers.items():
    print(f'Outliers for {column}:')
    print(outlier_data)
    print('-' * 40)
```

```
Outliers for Year:
Empty DataFrame
Columns: [Year, Month, Day, Weekend?, Hour, Collision Type, Injury Type, Primary Factor, Reported_Location, Lati
tude, Longitude, Risk Level]
Index: []
----------------------------------------
Outliers for Month:
Empty DataFrame
Columns: [Year, Month, Day, Weekend?, Hour, Collision Type, Injury Type, Primary Factor, Reported_Location, Lati
tude, Longitude, Risk Level]
Index: []
----------------------------------------
Outliers for Day:
Empty DataFrame
Columns: [Year, Month, Day, Weekend?, Hour, Collision Type, Injury Type, Primary Factor, Reported_Location, Lati
tude, Longitude, Risk Level]
Index: []
----------------------------------------
Outliers for Hour:
Empty DataFrame
Columns: [Year, Month, Day, Weekend?, Hour, Collision Type, Injury Type, Primary Factor, Reported_Location, Lati
tude, Longitude, Risk Level]
Index: []
```

Columns year, month, day, and hour show no outliers

Outliers for latitude:

	Latitude	Longitude	Risk Level
23	39.028430	-86.407807	High Risk
33	38.992016	-86.537248	High Risk
44	39.088157	-86.669012	Low Risk
50	39.263024	-86.522640	Moderate Risk
58	0.000000	0.000000	Moderate Risk
...	...	...	...
53938	0.000000	0.000000	High Risk
53939	0.000000	0.000000	High Risk
53940	0.000000	0.000000	Moderate Risk
53941	0.000000	0.000000	High Risk
53942	0.000000	0.000000	High Risk

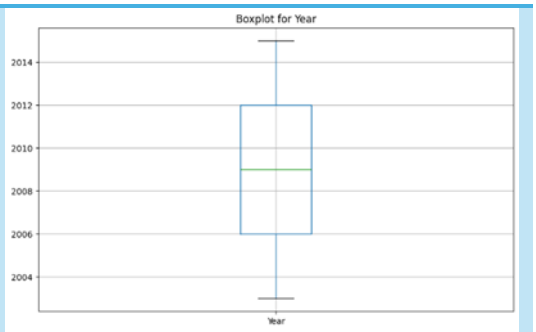
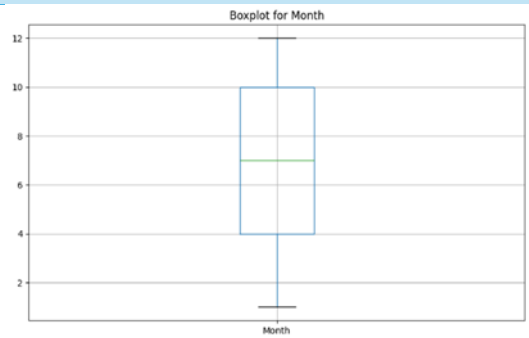
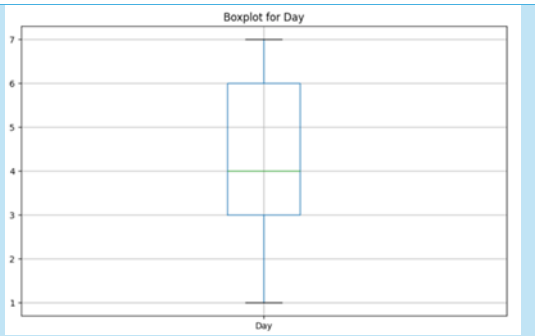
Outliers for longitude:

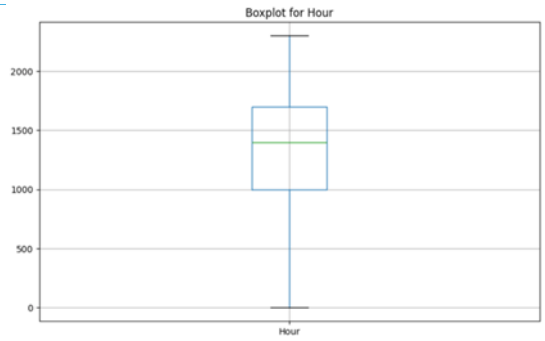
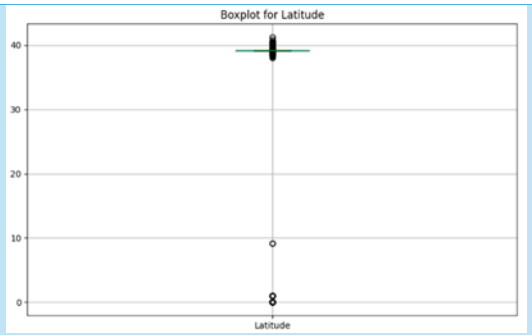
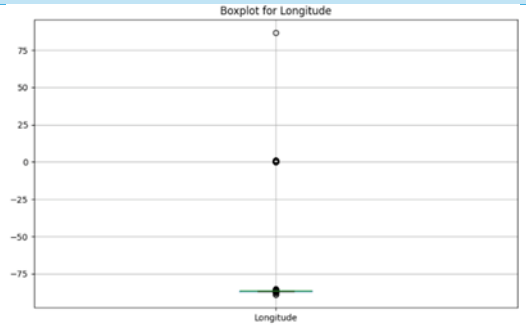
	Latitude	Longitude	Risk Level
7	39.199272	-86.637024	Low Risk
23	39.028430	-86.407807	High Risk
30	39.171870	-86.416711	High Risk
37	39.134144	-86.642864	Moderate Risk
44	39.088157	-86.669012	Low Risk
...	...	...	...
53938	0.000000	0.000000	High Risk
53939	0.000000	0.000000	High Risk
53940	0.000000	0.000000	Moderate Risk
53941	0.000000	0.000000	High Risk
53942	0.000000	0.000000	High Risk

Box Plots:

```
numeric_columns = data1.select_dtypes(include='number').columns

# Loop through numeric columns and create a boxplot for each
for column in numeric_columns:
    plt.figure(figsize=(10, 6))
    data1.boxplot(column=column)
    plt.title(f'Boxplot for {column}')
    plt.show()
```

Description	Graph
The Year Box-plot illustrates that there are no outliers in the year attribute. The minimum is 2003, Q1 is 2006, the median is 2009, Q3 is 2012, and the maximum is 2015. The broad range of years suggests variability in the data, and normalization might be necessary to ensure the data fits within a more consistent and comparable range for analysis.	
The Month Box-plot shows no outliers across the months, with a uniform distribution from January (1) to December (12). The minimum is January (1), Q1 is April (4), the median is August (7), Q3 is October (10), and the maximum is December (12). This indicates that the dataset contains full yearly data without extreme values. Given this, no transformation is needed for the month attribute, but encoding might improve its utility for models.	
The Day Boxplot illustrates a concentration of incidents across all days of the week, from Sunday (1) to Saturday (7). The minimum is Sunday (1), Q1 is Tuesday (3), the median is Wednesday (4), Q3 is Friday (6), and the maximum is Saturday (7). While there are no significant outliers, slight variations across specific days suggest that grouping into broader categories like weekdays vs. weekends could streamline the analysis and enhance prediction accuracy.	

The Hour Box-plot illustrates no outliers in the hour attribute. The minimum is 0 (12:00 AM), Q1 is 1000 (10:00 AM), the median is 1400 (2:00 PM), Q3 is 1700 (5:00 PM), and the maximum is 2400 (12:00 AM). The wide range suggests that binning the hours into broader intervals, such as morning, afternoon, and evening, could make the data more manageable and improve model interpretation.	
The Latitude Box-plot shows the geographic spread of incidents. The minimum is 0 (which could indicate missing or erroneous data), Q1 is approximately 39.0, the median is around 39.2, Q3 is 39.5, and the maximum is around 41.0. Outliers in latitude should be examined more closely as they may point to rare but significant locations of risk or potential data issues requiring preprocessing.	
The Longitude Boxplot also shows the geographic spread of incidents. The minimum is 0 (which could also indicate missing or erroneous data), Q1 is approximately -86.5, the median is around -86.6, Q3 is -86.2, and the maximum is about -84.0. Similar to Latitude, outliers in longitude should be carefully examined, as they may indicate data anomalies or rare accident locations that need further investigation.	

Plotting methods:

Bar Plot:

The bar plot shows the frequency of different Collision Types (e.g., 2-Car, 1-Car, Pedestrian) and their association with Injury Types. Each collision type is represented by a bar, with the color indicating the severity of the injury (e.g., Fatal, Non-incapacitating, No injury/Unknown). The y-axis represents the count of accidents, while the x-axis shows different collision categories.

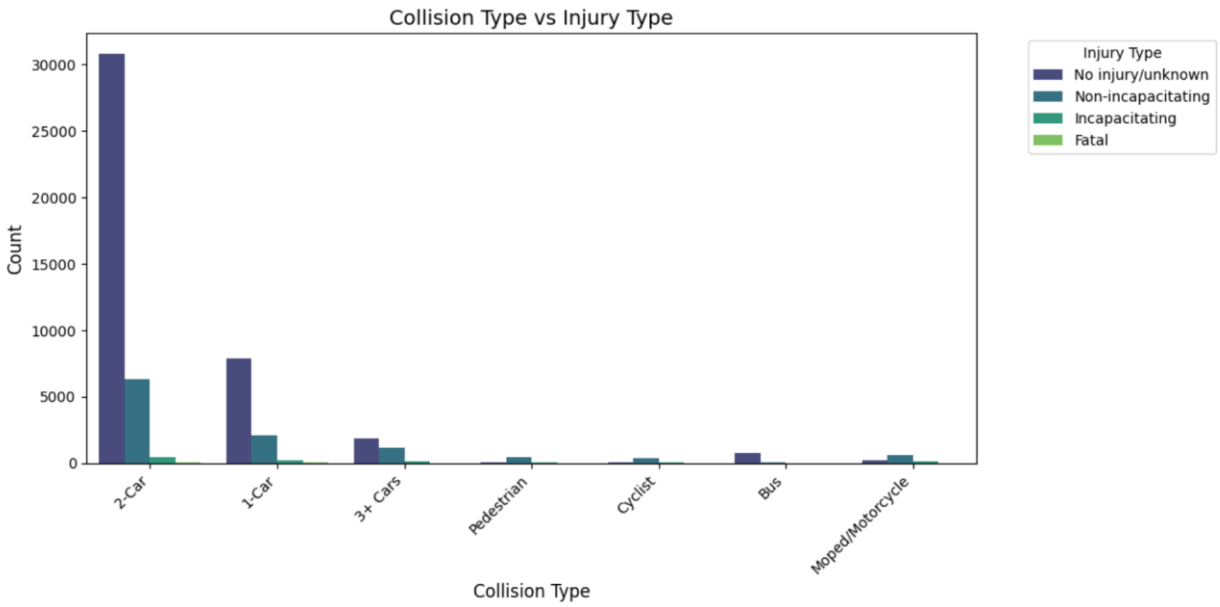
```
# Bar plot showing the frequency of each Collision Type and Injury Type
plt.figure(figsize=(12, 6))

# Create a count plot
sns.countplot(data=data1, x='Collision Type', hue='Injury Type', palette='viridis')

# Add title and labels
plt.title('Collision Type vs Injury Type', fontsize=14)
plt.xlabel('Collision Type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=45, ha='right')

# Show the legend
plt.legend(title='Injury Type', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show plot
plt.tight_layout()
plt.show()
```



Insights and Preprocessing Decisions:

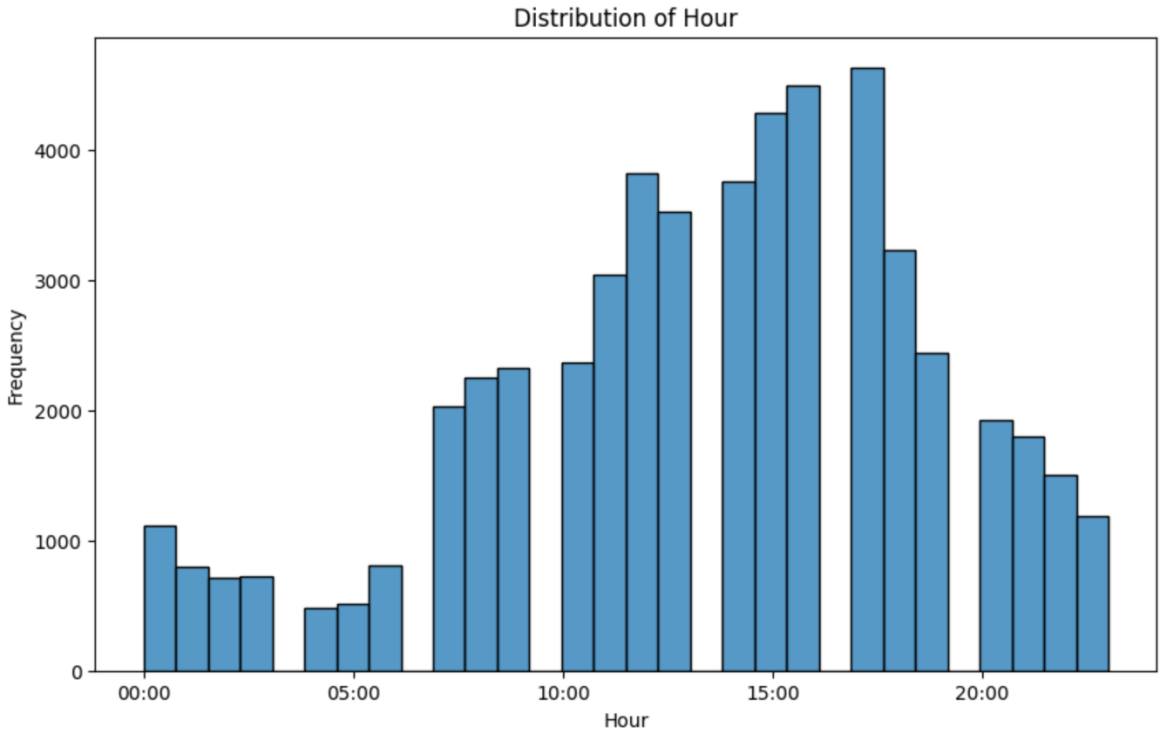
- 2-Car collisions are most frequent but usually result in non-incapacitating or no injury/unknown, while Moped/Motorcycle collisions have higher rates of incapacitating injuries.
- There is a class imbalance, with fatal injuries being underrepresented.

- Sparse data for collision types like Bus, Moped/Motorcycle, and Cyclist, which could indicate data issues or outliers.
- The "No injury/Unknown" category suggests the presence of missing injury data.

Histogram

The **x-axis** represents the hours of the day (from 00:00 to 23:00) while The **y-axis** shows the frequency of accidents occurring in each hour, with values ranging from 0 to 4,500, bars are grouped to reflect the number of incidents during specific time periods, with peaks around the **morning rush hours** and **evening rush hours**

```
    # Histogram
plt.figure(figsize=(10, 6))
sns.histplot(data1['Hour'], bins=30)
plt.title('Distribution of Hour')
plt.xlabel('Hour')
plt.xticks([0, 500, 1000, 1500, 2000], ['00:00', '05:00', '10:00', '15:00', '20:00'])
plt.ylabel('Frequency')
plt.show()
```



Insights and Preprocessing Decisions:

1. **Peak Hours:** More accidents during rush hours, indicating the role of traffic congestion.
2. **Skewness:** Right-skewed distribution suggests potential data imbalance, requiring sampling techniques.

3. **Missing Data:** Gaps in the dataset would show as lower bars, indicating a need for data cleaning.
4. **Normalization:** Skewed hours may need normalization for predictive modeling.

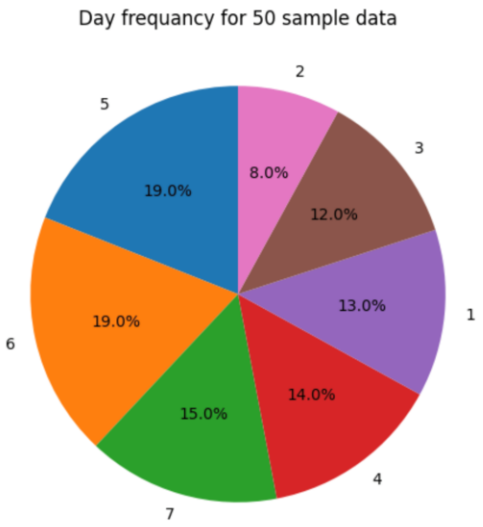
The histogram shows the need for data cleaning, addressing imbalanced data, and potential feature engineering for better model performance.

Pie Chart

The pie chart visualizes the frequency of incidents across the days of the week, using a random sample of 50 data points from the dataset. Each segment of the pie represents a different day of the week (from 1 to 7), with the size of the segment showing the percentage of incidents that occurred on that particular day.

The chart shows that certain days, such as Day 5 and Day 6, have a higher percentage of incidents, while Day 2 has a noticeably lower frequency. This highlights the variation in incident occurrence across different days, which may reflect underlying patterns such as weekends having more or fewer accidents compared to weekdays.

```
    # pie chart for 'Hour' column from the sample
# sample from  dataset
sampled_data = data1.sample(n=100, random_state=1)
Day_frequancy = sampled_data['Day'].value_counts(normalize=True)*100
Day_frequancy.plot.pie(autopct='%1.1f%%', figsize=(10,6),startangle=90)
plt.ylabel('')
plt.title('Day frequancy for 50 sample data')
plt.show()
```



insights and Preprocessing Decisions:

1. **Weekday vs Weekend Trends:** If weekends (e.g., Day 6 and Day 7) show a higher percentage of accidents, this could indicate a need to account for the differences in incident frequency when building predictive models. Preprocessing steps like adjusting for the day of the week may improve model performance.
2. **Data Imbalance:** If certain days are underrepresented, further sampling methods could be used to balance the dataset.

This pie chart helps reveal day-of-week trends in the dataset and suggests that preprocessing (such as normalization or balancing) may be needed to improve the model's handling of such patterns.

4. Data preprocessing

Data Preprocessing is a crucial step in the data analysis and machine learning pipeline, ensuring that raw data is cleaned, transformed, and made ready for modeling. Techniques such as encoding categorical variables, normalizing numerical features, and discretizing continuous variables are applied to make the data suitable for algorithms. Preprocessing also includes detecting and addressing imbalanced data. By enhancing the dataset's structure and relevance, data preprocessing ensures accurate, efficient, and reliable model performance.

1- Imputation of Missing Values

Imputing missing values in our dataset is a crucial step, as it helps with the dataset quality, model accuracy and data consistency.

- **Numeric Columns**: Mean imputation was chosen to maintain the central tendency of the data without introducing bias, especially since variables like `Hour` are continuous and show a relatively normal distribution.
- **Categorical Columns**: Most frequent value imputation ensures that the categorical distributions remain consistent while filling missing values

```python
# 1. Impute missing values for categorical columns
imputer_cat = SimpleImputer(strategy='most_frequent')
data1[['Weekend?', 'Collision Type', 'Primary Factor', 'Reported_Location']] = imputer_cat.fit_transform(data1[['Weekend?',

   # 2. Impute missing values for numeric columns with mean
imputer_num = SimpleImputer(strategy='mean')
data1[['Hour', 'Latitude', 'Longitude']] = imputer_num.fit_transform(data1[['Hour', 'Latitude', 'Longitude']])

      # -Check missing values again
missing_values = data1.isna().sum()
print("\nMissing values after imputation:\n", missing_values)
```

```
Missing values in each column:
 Year                   0
Month                   0
Day                     0
Weekend?               68
Hour                  225
Collision Type          6
Injury Type             0
Primary Factor       1121
Reported_Location      35
Latitude               30
Longitude              30
Risk Level              0
dtype: int64

Total number of missing values:
 1515
```

```
Missing values after imputation:
 Year                 0
Month                 0
Day                   0
Weekend?              0
Hour                  0
Collision Type        0
Injury Type           0
Primary Factor        0
Reported_Location     0
Latitude              0
Longitude             0
Risk Level            0
```

2- Encoding Categorical Columns

Convert categorical columns to numerical format using label encoding, to make feature selection faster and more efficient. As feature selection methods often rely on mathematical operations.

```python
 # Encoding
le = LabelEncoder()
data1['Weekend?']= le.fit_transform(data1['Weekend?'])
data1['Collision Type'] = le.fit_transform(data1['Collision Type'])
data1['Injury Type'] = le.fit_transform(data1['Injury Type'])
data1['Primary Factor'] = le.fit_transform(data1['Primary Factor'])
data1['Reported_Location'] = le.fit_transform(data1['Reported_Location'])
```

```python
# Encode target variable
le_target = LabelEncoder()
y_encoded = le_target.fit_transform(y)
```

Dataset after encoding:

data after encoding

```
print(data1.head())
```

✓ 0.0s

```
   Year  Month  Day  Weekend?    Hour  Collision Type  Injury Type  \
0  2015      1    5         0     0.0               1            2
1  2015      1    6         0  1500.0               1            2
2  2015      1    6         1  2300.0               1            3
3  2015      1    7         1   900.0               1            3
4  2015      1    7         1  1100.0               1            2

   Primary Factor  Reported_Location   Latitude  Longitude     Risk Level  \
0              27               1301  39.159207 -86.525874  Moderate Risk
1              11               1612  39.161440 -86.534848  Moderate Risk
2               5               4438  39.149780 -86.568890       Low Risk
3              10              10505  39.165655 -86.575956      High Risk
4              10              18788  39.164848 -86.579625  Moderate Risk

   Hour_Bin
0         3
1         0
2         1
3         2
4         2
```

Later, we encode the target variable ('Risk Level') using a LabelEncoder to convert the categorical labels into numerical values, making it suitable for model training.

3- Discretization of the 'Hour' Column

We divide the Hour column into bins for different times of day (Night, Morning, Afternoon, Evening), making the data easier to interpret and analyze. Then we encode the categories into numeric format.

```
 # Discretization
data1['Hour_Bin'] = pd.cut(data1['Hour'], bins=[0, 600, 1200, 1800, 2400], labels=['Night', 'Morning', 'Afternoon', 'Evening
  # Encode 'Hour_Bin'
data1['Hour_Bin'] = le.fit_transform(data1['Hour_Bin'])
```

Dataset after discretization:

Data after discretization

```
print(data1.head())
```

✓ 0.0s

```
   Year  Month  Day  Weekend?    Hour  Collision Type  Injury Type  \
0  2015      1    5         0     0.0               1            2
1  2015      1    6         0  1500.0               1            2
2  2015      1    6         1  2300.0               1            3
3  2015      1    7         1   900.0               1            3
4  2015      1    7         1  1100.0               1            2

   Primary Factor  Reported_Location   Latitude  Longitude     Risk Level  \
0              27               1301  39.159207 -86.525874  Moderate Risk
1              11               1612  39.161440 -86.534848  Moderate Risk
2               5               4438  39.149780 -86.568890       Low Risk
3              10              10505  39.165655 -86.575956      High Risk
4              10              18788  39.164848 -86.579625  Moderate Risk

   Hour_Bin
0         3
1         0
2         1
3         2
4         2
```

4- Normalization of Longitude and Latitude

Normalize Latitude and Longitude columns to scale them between 0 and 1. to make the values smaller, more comparable and to not dominate the features due to their large ranges.

```
# Normalization of Latitude and Longitude
scaler = MinMaxScaler()
data1[['Hour','Latitude', 'Longitude']] = scaler.fit_transform(data1[['Hour','Latitude', 'Longitude']])
```

Dataset after Normalization:

data after normalization

```
print(data1.head())
✓ 0.0s
```

	Year	Month	Day	Weekend?	Hour	Collision Type	Injury Type	\
0	2015	1	5	0	0.000000	1	2	
1	2015	1	6	0	0.652174	1	2	
2	2015	1	6	1	1.000000	1	3	
3	2015	1	7	1	0.391304	1	3	
4	2015	1	7	1	0.478261	1	2	

	Primary Factor	Reported_Location	Latitude	Longitude	Risk Level	\
0	27	1301	0.949805	0.013861	Moderate Risk	
1	11	1612	0.949860	0.013810	Moderate Risk	
2	5	4438	0.949577	0.013616	Low Risk	
3	10	10505	0.949962	0.013576	High Risk	
4	10	18788	0.949942	0.013555	Moderate Risk	

	Hour_Bin
0	3
1	0
2	1
3	2
4	2

5- Outlier Detection in the "Hour" column

Outliers are critical in understanding rare or extreme events that could indicate high-risk collision times. Here, we apply two methods to detect outliers:

1. Standard Deviation Method:

This method is effective for identifying extreme deviations in normalized data. The detected outliers may correspond to rare collision times (e.g., very early or late hours).

Outliers are values that deviate more than 2 standard deviations from the mean.

Suitable for data with a normal distribution.

2. Interquartile Range (IQR) Method:

This method did not initially detect outliers due to normalization

Outliers are values below Q1 - 1.5 IQR or above Q3 + 1.5 IQR.

Effective for data that is not normalized or is skewed.

The Hour column has been normalized, which can reduce the effectiveness of the IQR method. To address this, we apply IQR to the original scale of Hour.

- Both methods provide complementary views of outliers.

- Normalization can obscure variability, making IQR less effective without reverting to the original scale.
- Retained outliers as they represent rare but critical events for collision risk analysis.

```
# Create a backup of the original Hour column before normalization
data1['Original_Hour'] = data1['Hour'] * (2400 - 0)  # Revert normalization

# Outlier Detection using Standard Deviation
threshold = 2
mean_hour = data1['Hour'].mean()
std_hour = data1['Hour'].std()

# Identify outliers based on standard deviation
outliers_std = data1[data1['Hour'].apply(lambda x: abs(x - mean_hour) > threshold * std_hour)]
print(f"Outliers detected using Standard Deviation: {len(outliers_std)}")

# Outlier Detection using Interquartile Range (IQR) on Original_Hour
Q1 = data1['Original_Hour'].quantile(0.25)
Q3 = data1['Original_Hour'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers based on IQR
outliers_iqr = data1[(data1['Original_Hour'] < lower_bound) | (data1['Original_Hour'] > upper_bound)]
print(f"Outliers detected using IQR (on Original Hour): {len(outliers_iqr)}")

# Compare the results
print("\nOutliers using Standard Deviation:")
print(outliers_std.head())

print("\nOutliers using IQR:")
print(outliers_iqr.head())

# Visualization using Box Plot and Strip Plot
# Sampling data for visualization to reduce computational overhead
sampled_data = data1.sample(n=1000, random_state=42)
```

```
Outliers detected using Standard Deviation: 2632
Outliers detected using IQR (on Original Hour): 0

Outliers using Standard Deviation:
    Year  Month  Day  Weekend?      Hour  Collision Type  Injury Type  \
0   2015      1    5         0  0.000000               1            2
43  2015      1    4         0  0.000000               0            2
57  2015      1    7         1  0.000000               1            2
73  2015      1    4         0  0.000000               1            1
82  2015      1    1         1  0.043478               1            2

    Primary Factor  Reported_Location  Latitude  Longitude     Risk Level  \
0               27               1301  0.949805   0.013861  Moderate Risk
43              43               7016  0.950990   0.013845  Moderate Risk
57              10               9001  0.949927   0.014036  Moderate Risk
73              10               3648  0.950995   0.013697       Low Risk
82              49               3280  0.950036   0.013788  Moderate Risk

    Hour_Bin  Original_Hour
0          3       0.000000
43         3       0.000000
57         3       0.000000
73         3       0.000000
82         3     104.347826

Outliers using IQR:
```

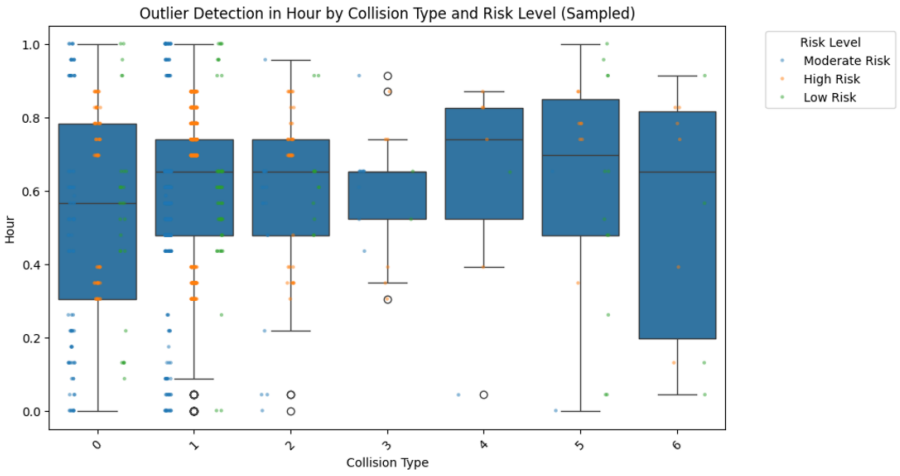
Visualizing Outliers:

combined box plot and strip plot illustrate:

- How outliers (dots) deviate from the median Hour for each Collision Type.
- The color coding by Risk Level helps highlight patterns in high-risk incidents.

```python
# Visualization using Box Plot and Strip Plot
# Sampling data for visualization to reduce computational overhead
sampled_data = data1.sample(n=1000, random_state=42)

plt.figure(figsize=(10, 6))
sns.boxplot(data=sampled_data, x='Collision Type', y='Hour')
sns.stripplot(data=sampled_data, x='Collision Type', y='Hour', hue='Risk Level', dodge=True, alpha=0.5, size=3)
plt.title('Outlier Detection in Hour by Collision Type and Risk Level (Sampled)')
plt.xlabel('Collision Type')
plt.ylabel('Hour')
plt.xticks(rotation=45)
plt.legend(title='Risk Level', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



6- Feature Selection

- Feature selection helps improve model performance by:
- Reducing redundancy and noise in the dataset.
- Simplifying the model, making it easier to interpret.
- Reducing overfitting, especially for datasets with many features. In this project, we use several feature selection techniques, including:

    1- Correlation-based feature removal.
    2- SelectKBest (Chi-Squared).
    3- Recursive Feature Elimination (RFE).
    4- L1 Regularization (Lasso)

We First Separate Features and Target

```python
# Separate the target variable ('Risk Level') from the features
X = data1.drop(['Risk Level'], axis=1)  # Features
y = data1['Risk Level']  # Target
```

1. Correlation Coefficient Calculation

We compute the correlation matrix to identify highly correlated features. If two features have a correlation above a threshold (0.8), we drop one of them to reduce multicollinearity and improve model performance.

```
# 1. Correlation-based feature selection
corr_matrix = X.corr()  # Compute correlation matrix
corr_threshold = 0.8  # Threshold for high correlation
high_corr = corr_matrix[(corr_matrix.abs() > corr_threshold) & (corr_matrix != 1.0)]  # Find highly correlated pairs

# Drop one feature from each correlated pair
to_drop = [col for col in high_corr.columns if any(high_corr[col].abs() > corr_threshold)]
X_reduced = X.drop(columns=to_drop)  # Reduced feature set
```

Features selected after removing highly correlated attributes:
['Year', 'Month', 'Day', 'Weekend?', 'Hour', 'Collision Type', 'Injury Type', 'Primary Factor', 'Reported_Location', 'Latitude', 'Risk Level', 'Hour_Bin']

2. SelectKBest Feature Selection (Chi-Squared Test)

We use SelectKBest to select the top 5 features based on the chi-squared test. This method ranks features according to their relevance for classification, helping us identify the most important features.

```
# 2. SelectKBest: Select top 10 features using chi-squared test
selector = SelectKBest(score_func=chi2, k=5)
X_new = selector.fit_transform(X_reduced, y)
selected_features = selector.get_support(indices=True)
print("\nSelected Features using SelectKBest:\n", X_reduced.columns[selected_features])
```

```
Selected Features using SelectKBest:
 Index(['Weekend?', 'Collision Type', 'Injury Type', 'Primary Factor',
       'Reported_Location'],
      dtype='object')
```

3. Recursive Feature Elimination (RFE) with Logistic Regression

Recursive Feature Elimination (RFE) is a feature selection technique that recursively removes the least important features to improve model performance. It ranks the features based on their importance to the target variable (Risk Level) as determined by the logistic regression model.

Why Use RFE?

- Interpretable Ranking: RFE provides a clear ranking of features based on their contribution to the model.
- Handles Redundancy: By removing less important features, RFE helps improve model generalization.
- Logistic Regression: Chosen for its robustness and suitability for classification tasks.

```
# Ensure features are scaled
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Scale the features

# Initialize Logistic Regression model with adjustments
model = LogisticRegression(max_iter=1000, solver='saga', tol=1e-4)

# Apply RFE
num_features_to_select = 5  # Specify the number of features to select
rfe = RFE(estimator=model, n_features_to_select=num_features_to_select)
rfe.fit(X_scaled, y)  # Fit RFE on the scaled data

# Output selected features
selected_features_rfe = X.columns[rfe.get_support()]
print(f"Features selected by RFE: {selected_features_rfe.tolist()}")
```

```
Features selected by RFE: ['Injury Type', 'Latitude', 'Longitude', 'Hour_Bin', 'Original_Hour']
```

4. L1 Regularization (Lasso) for Embedded Feature Selection

Lasso (L1 regularization) is applied to the model to shrink less important feature coefficients to zero, effectively performing feature selection. The features selected by Lasso are considered the most important.

```
lasso = Lasso(alpha=0.1)
lasso.fit(X, y_encoded)  # Fit Lasso model
selector = SelectFromModel(lasso, prefit=True)
selected_features = X.columns[selector.get_support()]
print("Selected Features based on L1 Regularization:", selected_features)
```

```
Selected Features based on L1 Regularization: Index(['Primary Factor', 'Original_Hour'], dtype='object')
```

Why Lasso?

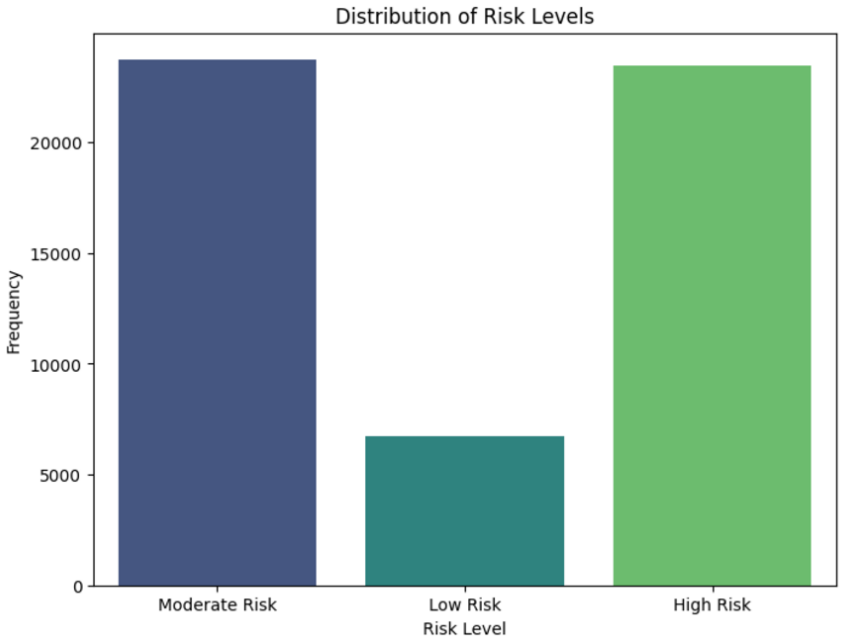
- Efficient for high-dimensional data.
- Automatically removes less relevant features by shrinking their coefficients to zero.

7- Balancing Data (Sampling)

Moderate Risk and High Risk have significantly higher frequencies compared to Low Risk. Low Risk is underrepresented, which may lead to challenges in accurately predicting this category, so balancing is needed to ensure that the model does not become biased towards the more frequent categories (Moderate Risk and High Risk) and can accurately predict all classes, including the underrepresented Low Risk category.

Visualizing Unbalanced Data:

```
plt.figure(figsize=(8, 6))
sns.countplot(data=data1, x='Risk Level', palette='viridis')
plt.title('Distribution of Risk Levels')
plt.xlabel('Risk Level')
plt.ylabel('Frequency')
plt.show()
```



Sampling:

```
print("Original class distribution:\n" , data1['Risk Level'].value_counts())

# Use the processed data1 DataFrame for sampling
total_sample_size = 10000  # Example size for training

# Get the class distribution of the 'Risk Level' column
risk_levels = data1['Risk Level'].value_counts()

# Calculate sample sizes for each class based on the total sample size
sample_sizes = {level: int(total_sample_size * (count / len(data1))) for level, count in risk_levels.items()}

# Adjust if there are not enough samples in any class
for level in risk_levels.index:
    class_count = data1[data1['Risk Level'] == level].shape[0]
    if class_count < sample_sizes[level]:
        sample_sizes[level] = class_count

# Sample the data to balance the classes
balanced_sample = pd.concat([
    data1[data1['Risk Level'] == level].sample(sample_sizes[level], replace=True, random_state=42)
    for level in risk_levels.index
])

# Verify the balanced sample distribution
print("Balanced sample class distribution:\n", balanced_sample['Risk Level'].value_counts())
print(balanced_sample)
```

```
Original class distribution:
 Risk Level
Moderate Risk    23746
High Risk        23475
Low Risk          6722
Name: count, dtype: int64
Balanced sample class distribution:
 Risk Level
Moderate Risk    4402
High Risk        4351
Low Risk         1246
Name: count, dtype: int64
       Year  Month  Day  Weekend?      Hour  Collision Type  Injury Type  \
53762  2003      1    6         0  0.608696               0            2
35228  2007      6    5         0  0.478261               1            2
1921   2015      6    7         1  0.478261               1            2
11829  2013      3    2         0  0.478261               1            2
49103  2004      4    1         0  0.913043               1            2
...     ...    ...  ...       ...       ...             ...          ...
12314  2013      7    1         1  0.086957               0            3
9148   2013      8    5         0  0.434783               6            3
4349   2014      4    4         0  0.260870               6            3
20405  2011      5    4         0  0.608696               1            3
23641  2010      8    2         0  0.608696               1            3

       Primary Factor  Reported_Location  Latitude  Longitude     Risk Level  \
53762              38              10670  0.000000   0.506730  Moderate Risk
35228              48               5896  0.949927   0.014017  Moderate Risk
1921               49               6393  0.949749   0.013537  Moderate Risk
11829              48              17297  0.949242   0.013903  Moderate Risk
49103              10               2918  0.000000   0.506730  Moderate Risk
...               ...                ...       ...        ...           ...
12314              38              13417  0.946437   0.013803       Low Risk
9148               27               4112  0.950378   0.013757       Low Risk
4349               35              19264  0.953537   0.013058       Low Risk
20405              10              10695  0.946285   0.013752       Low Risk
23641              11              12741  0.949464   0.013566       Low Risk

       Hour_Bin  Original_Hour
53762         0    1460.869565
35228         2    1147.826087
1921          2    1147.826087
11829         2    1147.826087
49103         1    2191.304348
...         ...            ...
12314         3     208.695652
9148          2    1043.478261
4349          2     626.086957
20405         0    1460.869565
23641         0    1460.869565

[9999 rows x 14 columns]
```

Dataset Before and After Preprocessing:

Data before preprocessing

```
print(data1.head())
```

✓ 0.0s

```
   Year  Month  Day Weekend?    Hour Collision Type         Injury Type  \
0  2015      1    5  Weekday     0.0          2-Car   No injury/unknown
1  2015      1    6  Weekday  1500.0          2-Car   No injury/unknown
2  2015      1    6  Weekend  2300.0          2-Car  Non-incapacitating
3  2015      1    7  Weekend   900.0          2-Car  Non-incapacitating
4  2015      1    7  Weekend  1100.0          2-Car   No injury/unknown

                        Primary Factor      Reported_Location   Latitude  \
0  OTHER (DRIVER) - EXPLAIN IN NARRATIVE            1ST & FESS  39.159207
1                 FOLLOWING TOO CLOSELY          2ND & COLLEGE  39.161440
2             DISREGARD SIGNAL/REG SIGN  BASSWOOD & BLOOMFIELD  39.149780
3          FAILURE TO YIELD RIGHT OF WAY         GATES & JACOBS  39.165655
4          FAILURE TO YIELD RIGHT OF WAY                W 3RD  39.164848

   Longitude     Risk Level
0 -86.525874  Moderate Risk
1 -86.534848  Moderate Risk
2 -86.568890       Low Risk
3 -86.575956      High Risk
4 -86.579625  Moderate Risk
```

data after preprocessiing

```
print(data1.head())
```

✓ 0.0s

```
   Year  Month  Day  Weekend?      Hour  Collision Type  Injury Type  \
0  2015      1    5         0  0.000000               1            2
1  2015      1    6         0  0.652174               1            2
2  2015      1    6         1  1.000000               1            3
3  2015      1    7         1  0.391304               1            3
4  2015      1    7         1  0.478261               1            2

   Primary Factor  Reported_Location  Latitude  Longitude     Risk Level  \
0              27               1301  0.949805   0.013861  Moderate Risk
1              11               1612  0.949860   0.013810  Moderate Risk
2               5               4438  0.949577   0.013616       Low Risk
3              10              10505  0.949962   0.013576      High Risk
4              10              18788  0.949942   0.013555  Moderate Risk

   Hour_Bin  Original_Hour
0         3       0.000000
1         0    1565.217391
2         1    2400.000000
3         2     939.130435
4         2    1147.826087
```

5. Data Mining Technique

We applied both supervised and unsupervised learning to our dataset using **classification** and **clustering** techniques.

Classification

For classification, we used a **Decision Tree**, which is a recursive algorithm that produces a tree with leaf nodes representing the final decisions. Our model predicts the class label (Risk Level) with three categories: *Low Risk*, *Moderate Risk*, and *High Risk*. The prediction is made based on the remaining attributes (Year, Month, Day, Weekend? , Hour, Collision Type, Injury Type, Primary Factor, Latitude, Longitude, and Hour_Bin).

This technique involved dividing the dataset into two subsets:

- **Training Dataset**: Used for building the decision tree.
- **Testing Dataset**: Used to evaluate the constructed model.

To evaluate our model, we measured accuracy, precision, recall, and F1-score. We also visualized the results using a confusion matrix and interpreted the outcomes for each split.

We used Python's **scikit-learn** package for classification. Specifically, the following methods were utilized:

- train_test_split for splitting the dataset into training and testing subsets.
- DecisionTreeClassifier for constructing the decision tree.
- predict for making predictions on the test set.
- classification_report and confusion_matrix for evaluating the model's performance.

Training Procedure

Since classification is supervised learning, we required training data to build the model. We experimented with three different train-test splits (70%-30%, 80%-20%, and 90%-10%) to identify the configuration that yields the best accuracy. Given the dataset size, we prioritized a larger training subset to improve the model's ability to generalize and predict new data accurately.

The decision tree was built using both the **Gini Index** and **Entropy** criteria to split nodes. After training, the models were evaluated using the test set, and the results (accuracy and confusion matrix) were compared to identify the best-performing split and splitting criterion.

Decision Tree Visualization

The constructed decision tree was visualized to provide insights into how the model makes decisions based on feature values. Each node represents a decision point (whether Hour <= threshold), and the leaves represent final classifications (*Low Risk*, *Moderate Risk*, or *High Risk*).

Clustering

For clustering, we applied **K-Means Clustering**, an unsupervised learning algorithm. Unlike classification, clustering does not use a target class label. Thus, we excluded the Risk Level attribute and used the remaining features (Year, Month, Day, Weekend?, Hour, Collision Type, Injury Type, Primary Factor, Latitude, Longitude, and Hour_Bin) for clustering.

The algorithm partitions the dataset into K clusters, where each cluster is represented by a centroid. Each observation is assigned to the nearest cluster, and the centroids are iteratively recalculated until convergence (when cluster assignments no longer change).

We use K-Means because:

- **Scalability**: Efficient for large datasets.
- **Compactness**: Groups similar observations into tight clusters.
- **Ease of Implementation**: Simple yet effective for numeric and categorical data.

We used Python's **scikit-learn** library for clustering and **yellowbrick** for visualization. Specifically:

- StandardScaler was used to normalize the features.
- KMeans was used to create the clusters.
- The **Elbow Method** and **Silhouette Analysis** were applied to determine the optimal number of clusters (K).
- yellowbrick.cluster.SilhouetteVisualizer and scatterplot from seaborn were used to visualize the clustering results.

Validation and Visualization

To validate the clustering results:

- The **Elbow Method** was used to identify the number of clusters that minimizes the inertia (compactness of clusters).
- **Silhouette Analysis** measured the quality of clustering by evaluating how well-separated the clusters are.
- The clusters were visualized in a 2D space using features like Latitude and Longitude.

Clustering Steps

1. Normalize the dataset using StandardScaler to ensure all features contribute equally.
2. Apply K-Means clustering for different values of K (2 to 10).
3. Evaluate cluster quality using the Elbow Method and Silhouette Analysis.
4. Visualize the clusters and interpret the groupings.

Summary

- **Classification**:
    - Method: Decision Tree (Gini and Entropy).
    - Python Packages: scikit-learn.
    - Purpose: Predict Risk Level based on features and evaluate model performance.
    - Key Results: Accuracy across different splits was consistent (~89%), with minimal differences between Gini and Entropy criteria.
- **Clustering**:
    - Method: K-Means Clustering.
    - Python Packages: scikit-learn, yellowbrick, and seaborn.
    - Purpose: Group similar observations to identify hidden patterns in the data.
    - Key Results: The Elbow Method and Silhouette Analysis identified the optimal number of clusters (K=4). Cluster visualizations provided insights into spatial and temporal patterns.

Both techniques provide complementary insights, with classification focusing on predictions and clustering uncovering natural groupings within the dataset. Together, these approaches enhance our understanding of car accident risk levels and their contributing factors.

6. Evaluation and Comparison

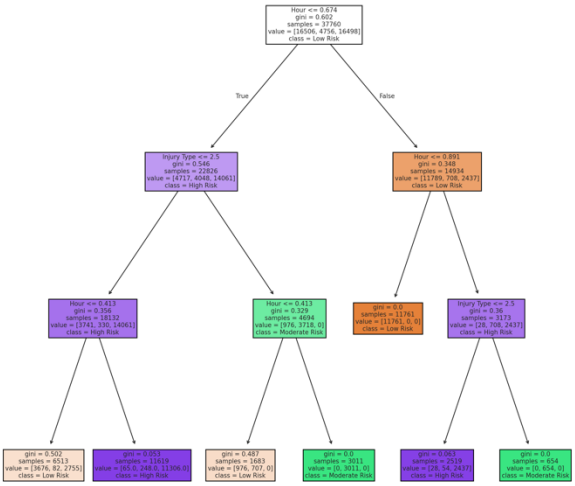
In this section, we compare the performance of the Decision Tree classifier using two different criteria: Information Gain (Entropy) and Gini Index. We evaluated the models using three different training-test splits: 90% training, 10% testing, 80% training, 20% testing, and 70% training, 30% testing.

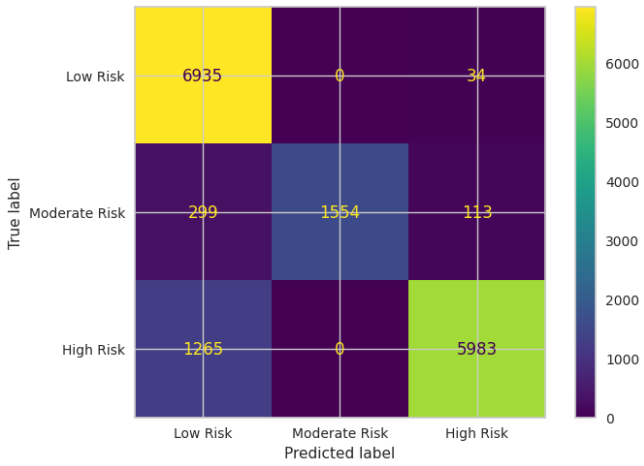
6.1 Classification

- *Classification [70% training, 30% testing]:*

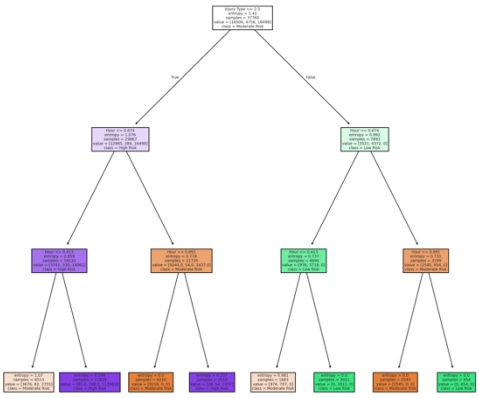
Gini index:

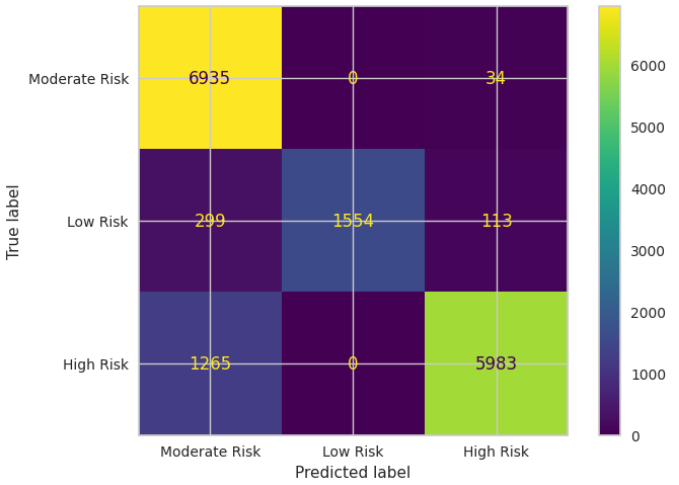
Figure (1) Decision Tree, Confusion matrix:





Entropy:

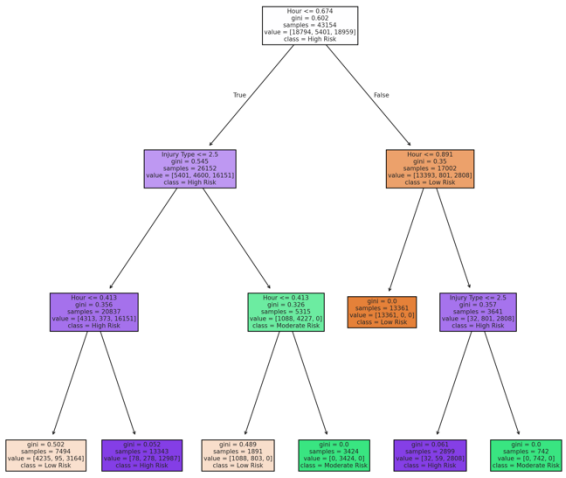


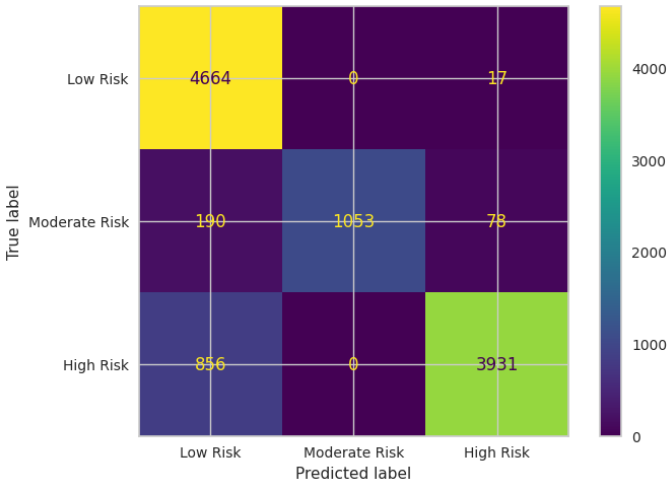


- *Classification [80% training, 20% testing]:*

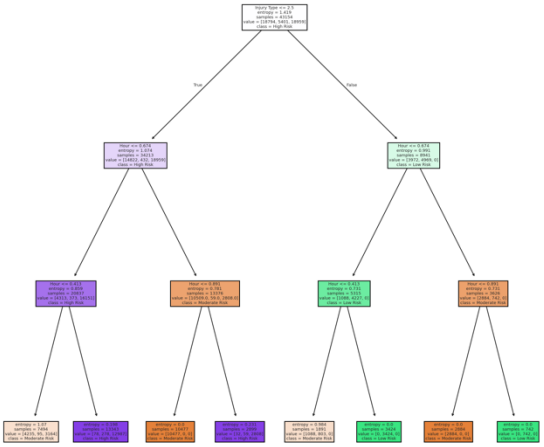
Gini index:

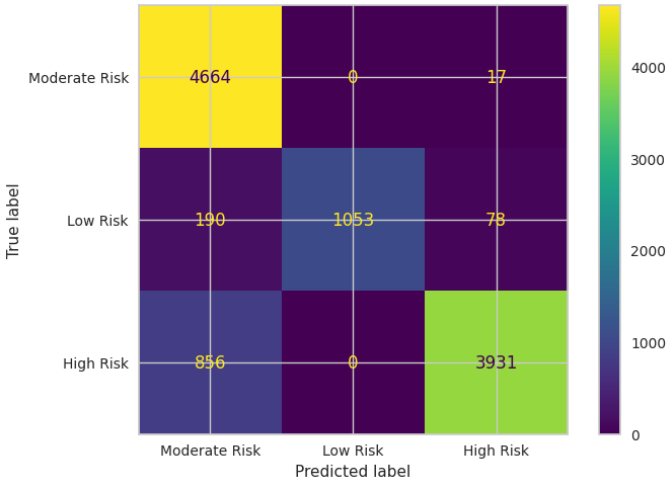
Figure (2) Decision Tree, confusion matrix:





Entropy:

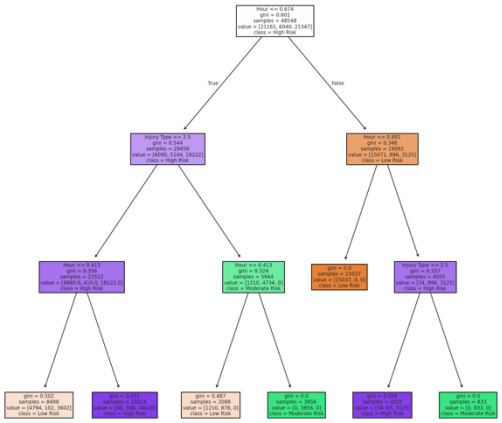


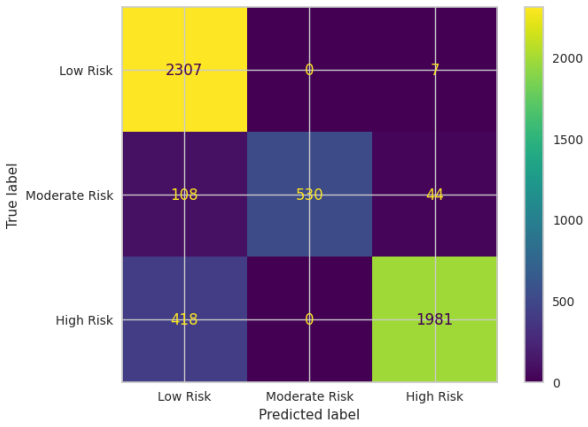


- *Classification [90% training, 10% testing]:*

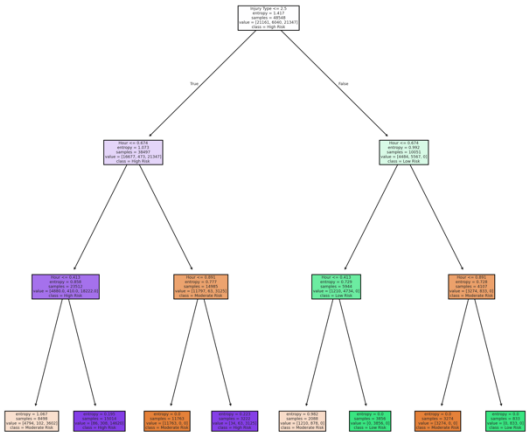
Gini index:

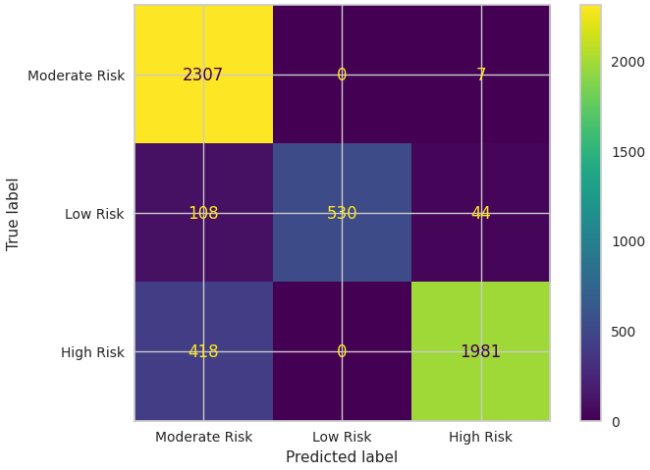
Figure (3) Decision Tree:





Entropy:





Minning Task	*Comparison Criteria*
Classification	***We tried 3 different sizes for dataset splitting to create the decision tree:***

Evaluation metrics: Accuracy

	Information Gain	Gini index
90 %t raining set 10% testing set:	89.3%	89.3%
80 % training set 20% testing set:	89.42%	89.42%
70 % training set 30% testing set:	89.43%	89.43%

As shown, both **Information Gain** and **Gini Index** produced **identical accuracy values** across all splits, indicating that the splitting criterion did not significantly impact accuracy for this dataset.

6.2 Clustering

- *Clustering [k=3]:*

Figure (1): scatter plot

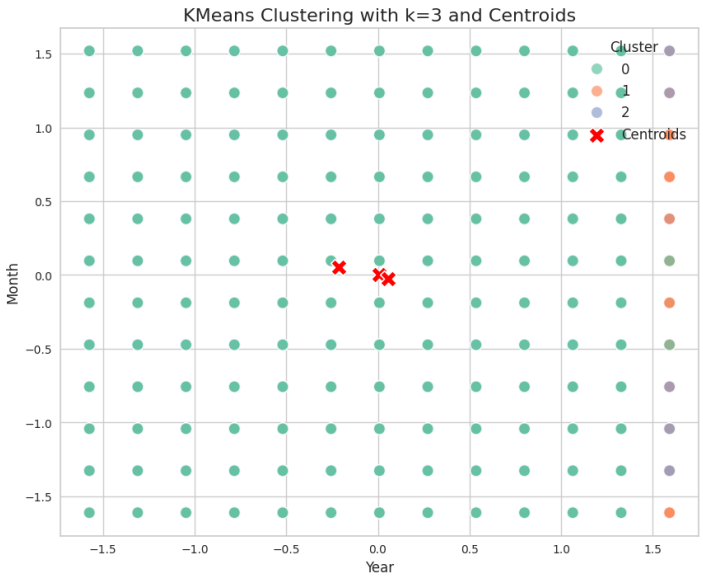
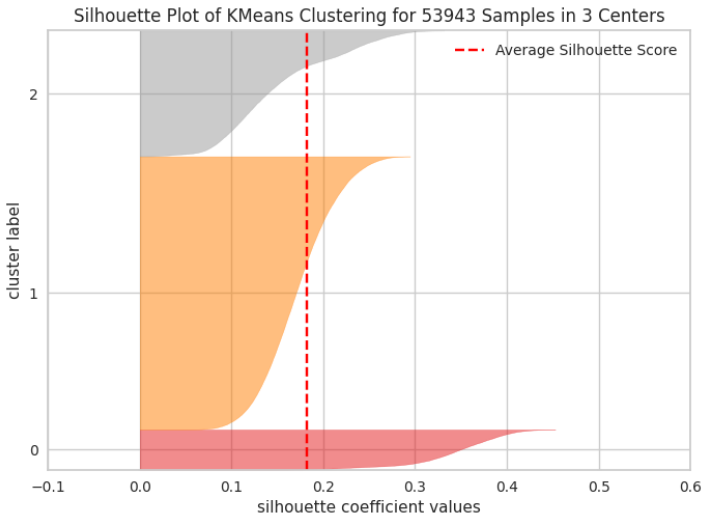


Figure (2): average silhouette

The average silhouette score for k=3 is: 0.18195072479886865



- *Clustering [k=4]:*

Figure (1):

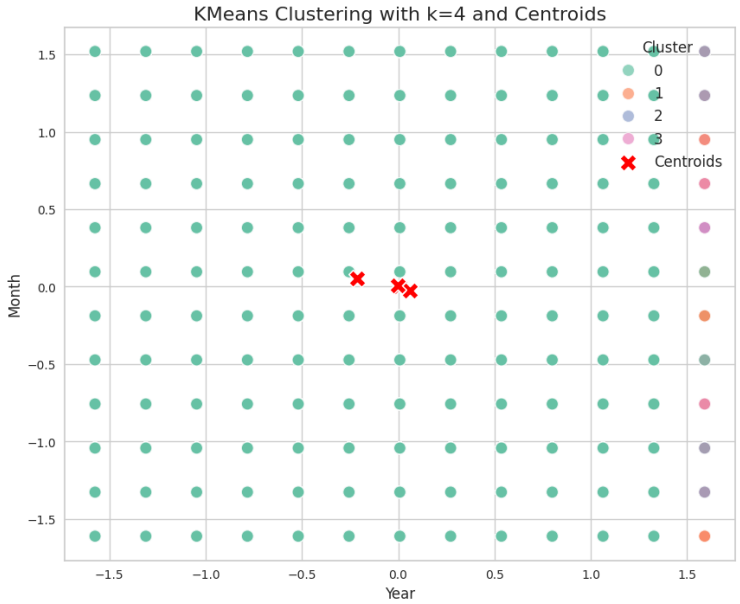
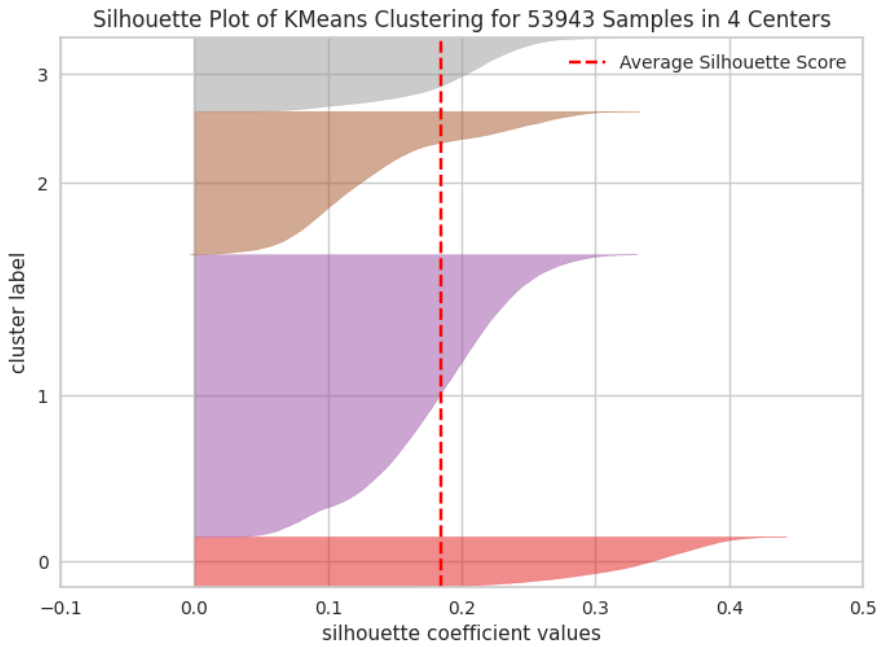


Figure (2): average silhouette

The average silhouette score for k=4 is: 0.1842830218004177



- *Clustering [k=5]:*

Figure (1):

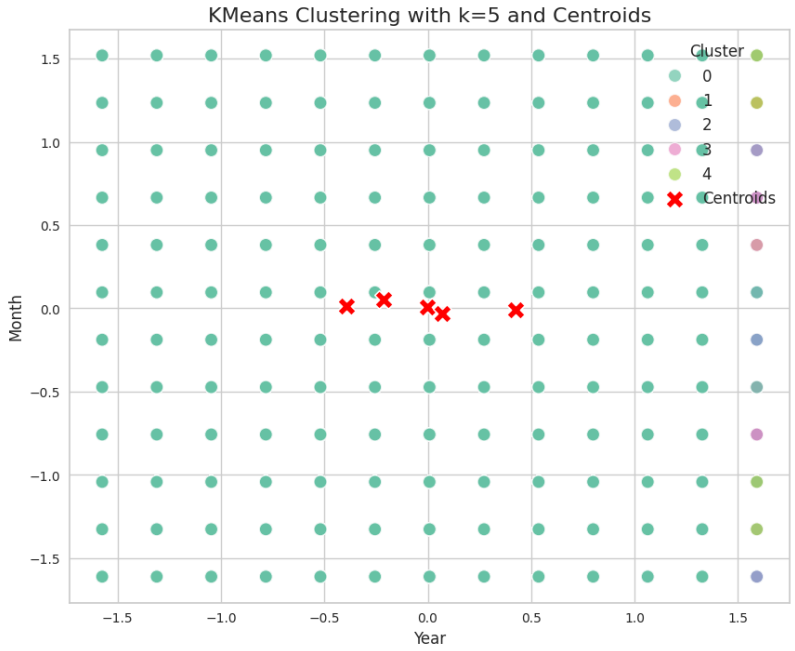
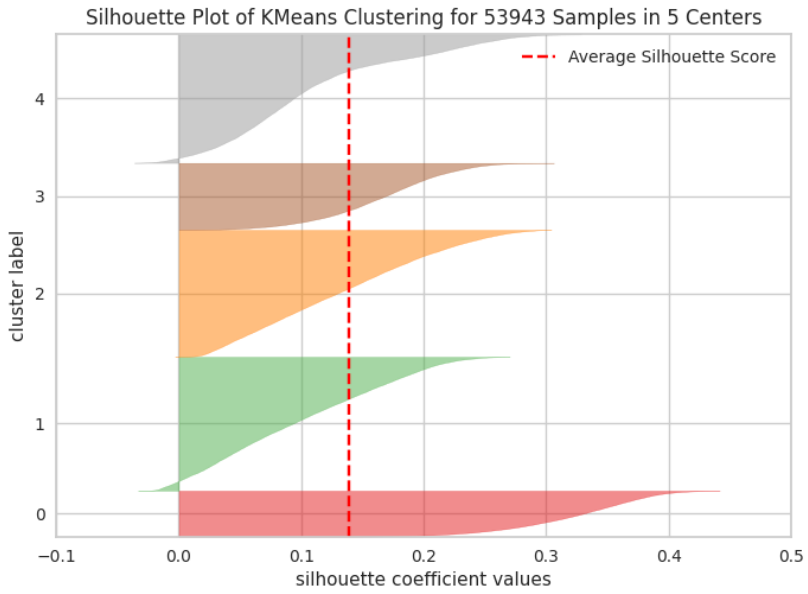
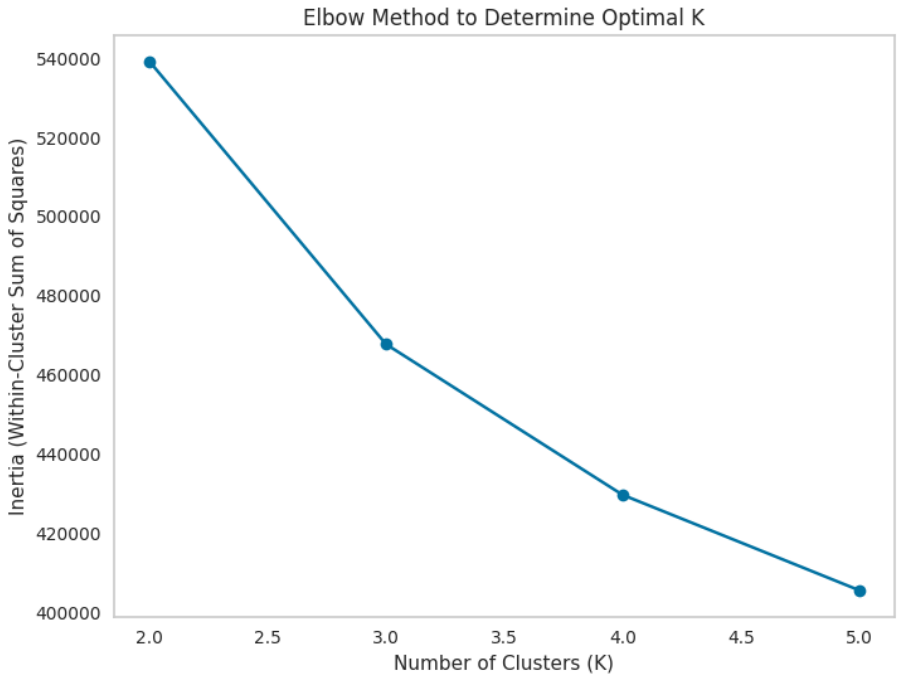


Figure (2): average silhouette width

The average silhouette score for k=5 is: 0.13926206310860578



- *Optimal number of clusters:*



<table>
<tr><th>Mining Task</th><th>Comparison criteria</th></tr>
<tr><td>Clustering</td><td>
We tried 3 different numbers of clusters:

K= 3, K=4, K=5
<table>
<tr><th>No. of clusters</th><th>3(BEST)</th><th>4</th><th>5</th></tr>
<tr><td>Silhouette width for all clusters</td><td>0.181</td><td>0.184</td><td>0.139</td></tr>
<tr><td>Total within-cluster sum of square.</td><td>K=3: 467562.53763123794</td><td>K=4: 429497.90052027325</td><td>K=5: 405395.48596480227</td></tr>
<tr><td>Visualization</td><td colspan="3">In the figures above.</td></tr>
</table>
</td></tr>
</table>

7. Findings

In the beginning, we selected a dataset that captures various factors contributing to car accidents. By leveraging this data, we aim to develop predictive models that classify and cluster the likelihood of future accidents. The ultimate objective is to enhance road safety, optimize traffic management, and provide valuable insights to inform policy decisions.

To ensure accurate and efficient results, we applied several preprocessing techniques to improve data quality. We utilized various plotting methods, such as boxplots and histograms, to visually explore the data, which helped us better understand its structure and identify the most suitable preprocessing steps. Based on these visualizations and other analyses, we removed all null, missing, and outlier values, as these could negatively impact the results. Additionally, we applied data transformation, so we normalized and discretized some attributes to give them equal weight and to facilitate handling the data during data mining tasks.

Consequently, we applied the data mining tasks which are classification and clustering.

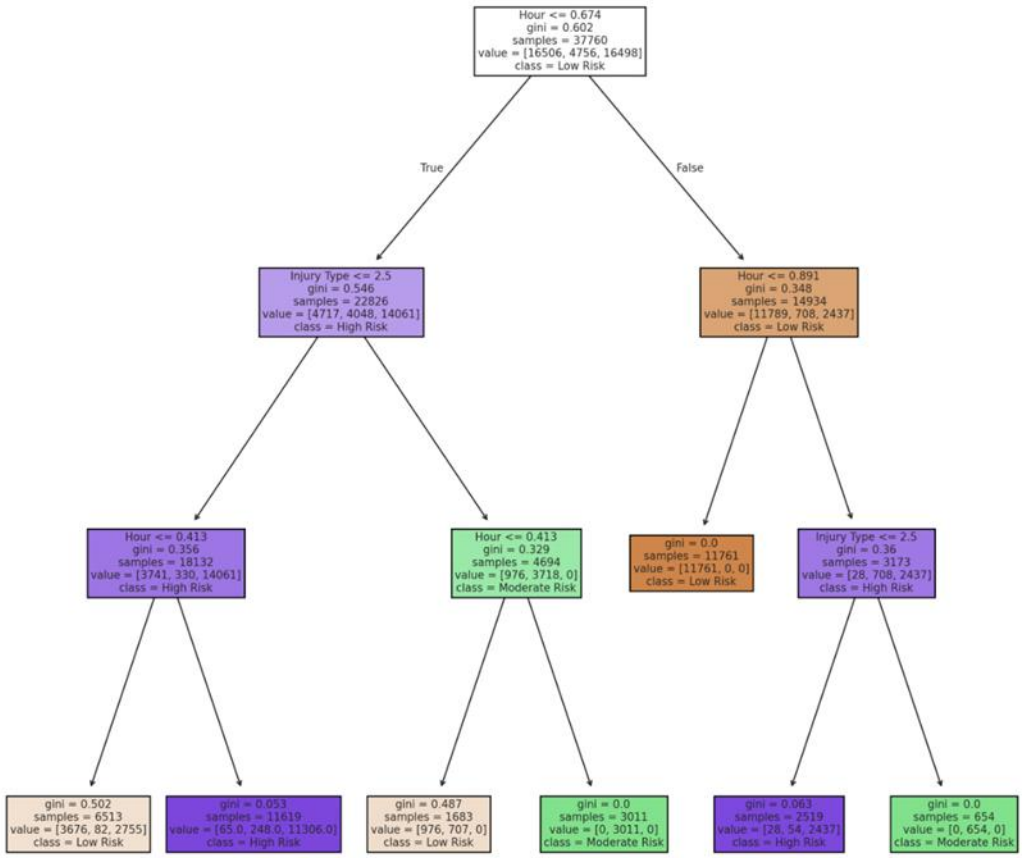
For classification, we use the decision tree method to construct our model using both Gini and Entropy, we tried 3 different sizes of training and testing data to get the best result for construction and evaluation and we concluded the following results:

Gini and Entropy

- 70% Training data, 30% Test data, Criterion= Gini, accuracy = 89.43%
- 70% Training data, 30% Test data, Criterion=Entropy, accuracy = 89.43%
- 80% Training data, 20% Test data, Criterion= Gini, accuracy = 89.42%
- 80% Training data, 20% Test data Criterion=Entropy, accuracy = 89.42%
- 90% Training data, 10% Test data, Criterion= Gini, accuracy = 89.30%
- 90% Training data, 10% Test data, Criterion=Entropy, accuracy = 89.30%

The model that has the best accuracy was the first model with 70% training data and 30% test data which means that most tuples were correctly classified.

Gini Index:

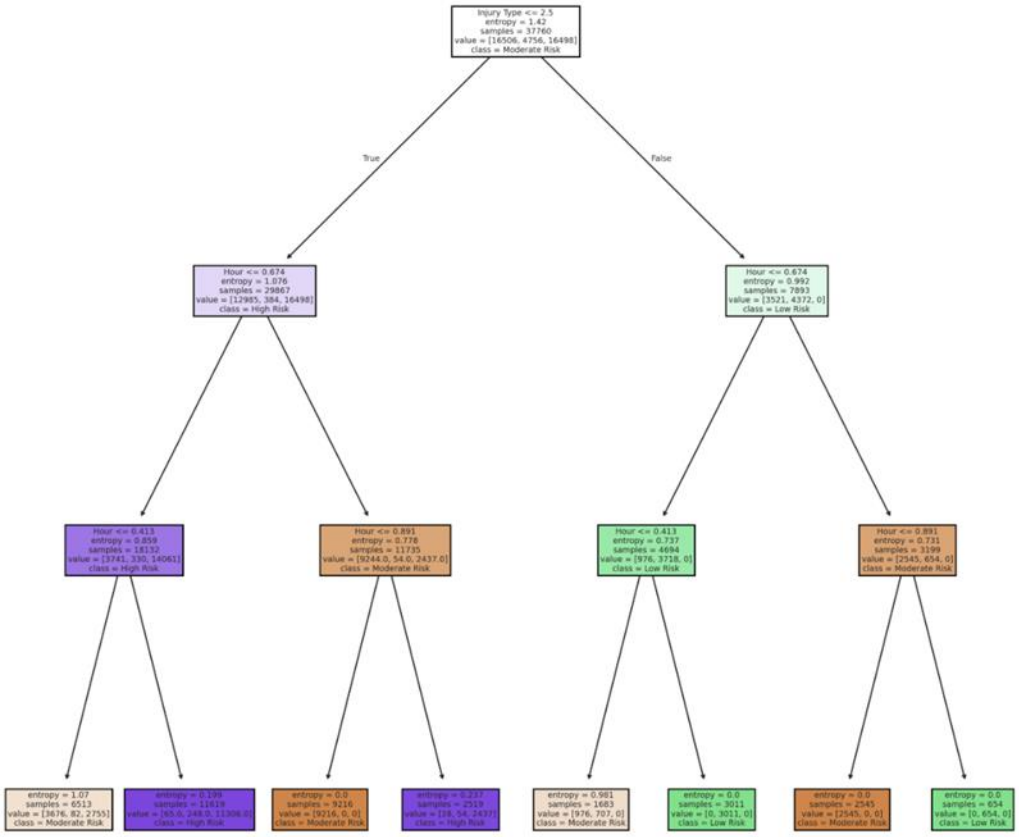


From the plot of the tree, we concluded the following results:

- Nodes: Represent decision points based on feature values.
- Leaves: Represent final classifications (Low Risk, Moderate Risk, High Risk).
- Colors represent the class distribution at each node
- The root node might split on "Hour":
    - If Hour <= 0.674 = Yes, go to the left branch.
    - If Hour <= 0.674 = No, go to the right branch.
- According to the tree the first node were the Hour <= 0.674 = no ,needs further portioning and was partitioned based on the Hour, if Hour <=0.891 yes go to the left and if no go to the right. if Hour <=0.891 yes you will have the leaf were gini=0.0 and it is **Low Risk** , if Hour <=0.891 no ,needs further portioning and was partitioned based on the Injury type, if Injury type<=2.5 yes you have gini=0.063 and **High Risk** .if Injury type<=2.5 no you have gini=0.0 and **Moderate Risk**.
- And for the second node were the Hour <= 0.674 = Yes, needs further portioning and was partitioned based on the Injury type. if Injury type<=2.5 yes go to the left and if no go to right.
- in left needs further portioning and was partitioned based on the Hour, if Hour <=0.413 yes then go to the left were gini=0.502 and **LowRisk** .if Hour <=0.413NO go to right were gini =0.053 and **High Risk**.

- and in the right needs further portioning and was partitioned based on the Hour, if Hour<=0.413 yes then go to the left were gini=0.487 and **Low Risk. if** Hour <=0.413NO go to right were gini =0.0 and **Moderate Risk**.
- The tree uses Hour and Injury Type features to classify samples into Low Risk, Moderate Risk, or High Risk. Leaf nodes represent pure or almost pure classifications based on the Gini index, with predictions derived from class distributions.
- If the accident occurs early in the day (Hour <= 0.674) and involves less severe injuries (Injury Type <= 2.5), the risk of the accident is typically Low Risk, unless it's later in the day (Hour > 0.413), where High Risk is predicted.
- If the accident happens later in the day (Hour > 0.674), there is a greater chance of High Risk or Moderate Risk depending on the exact hour and injury type.
- High Injury Type (greater than 2.5) is often linked with Moderate Risk based on the hour.

Entropy:



From the plot of the tree, we concluded the following results:

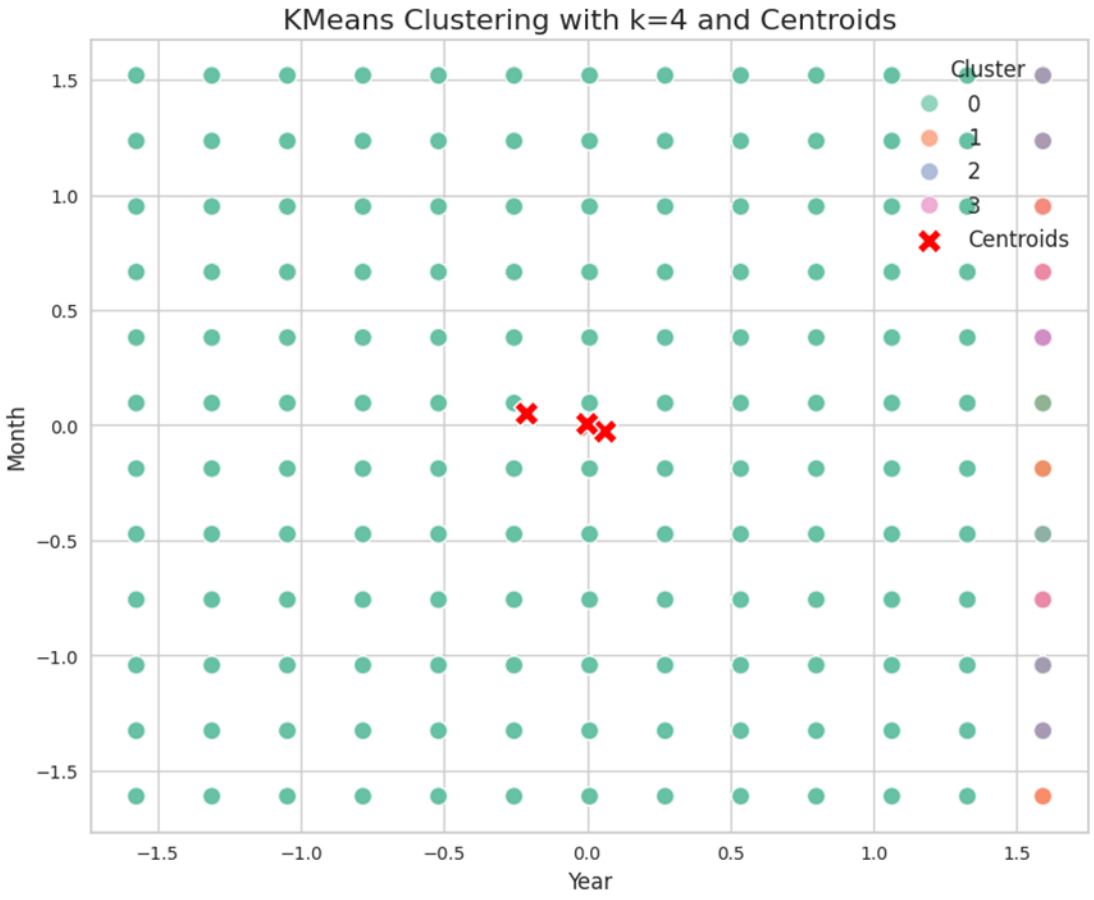
- Nodes: Represent decision points based on feature values.
- Leaves: Represent final classifications (Low Risk, Moderate Risk, High Risk).
- Colors represent the class distribution at each node
- The root node might split on "injury type":
    - If injury type <= 2.5 = Yes, go to the left branch.
    - If injury type <= 2.5 = No, go to the right branch.
- If injury type <= 2.5 = Yes, needs further portioning and was partitioned based on the Hour, if Hour<=0.674 =yes go to the left and if the Hour<=0.674 =no go to the right
- if Hour<=0.674 =yes, needs further portioning and was partitioned based on the Hour, if Hour<=0.413=yes go to the left were entropy=1.07 and **Moderate Risk**. if Hour<=0.413=no go to the right were entropy=0.199 and **High Risk**.
- if Hour<=0.674 =no, needs further portioning and was partitioned based on the Hour, if Hour<=0.891=yes go to the left were entropy=0.0 and **Moderate Risk. if** Hour<=0.891=no go to the right were entropy=0.237 and **High Risk**.
- If injury type <= 2.5 = no, needs further portioning and was partitioned based on the Hour, if Hour<=0.674 =yes go to the left and if the Hour<=0.674 =no go to the right
- if Hour<=0.674 =yes, needs further portioning and was partitioned based on the Hour, if Hour<=0.413=yes go to the left were entropy=0.981 and **Moderate Risk**. if Hour<=0.413=no go to the right were entropy=0.0 and **Low Risk.**

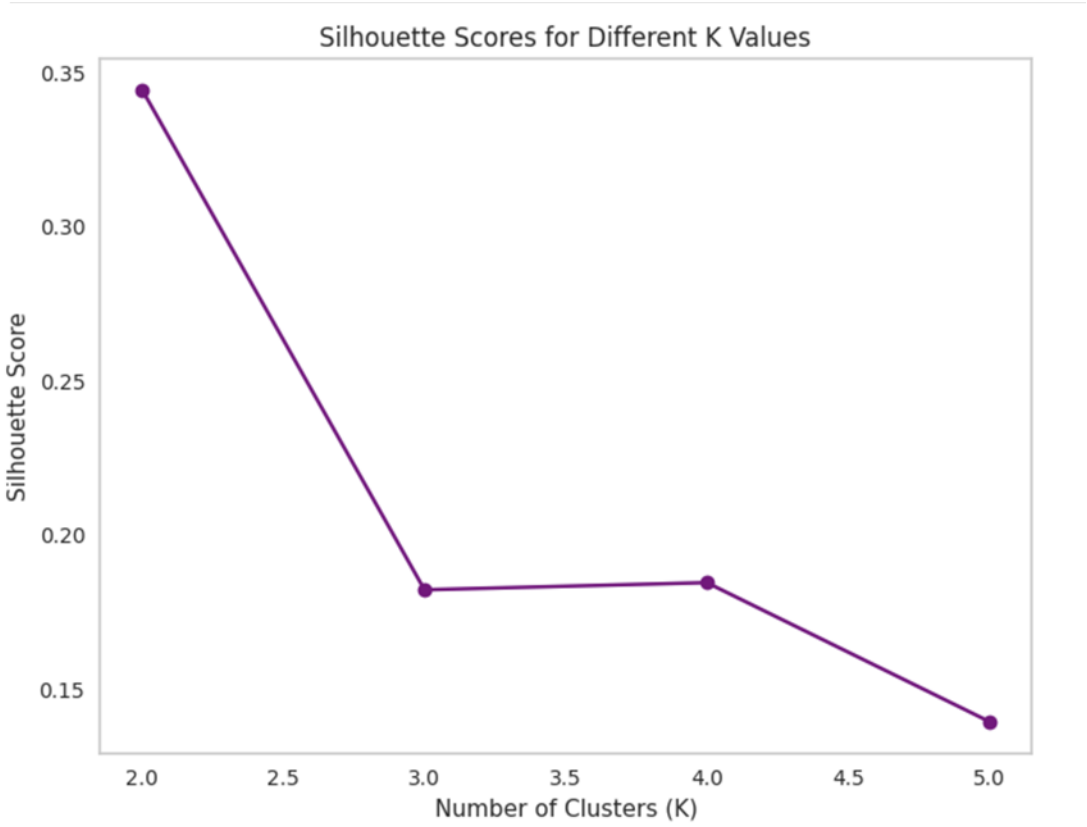
- if Hour<=0.674 =no, needs further portioning and was partitioned based on the Hour,if Hour<=0.891=yes go to the left were entropy=0.0 and **Moderate Risk.** if Hour<=0.891=no go to the right were entropy=0.0 and **Low Risk**.

In the **clustering**, we used the **K-means** algorithm with three different values of *K* to determine the optimal number of clusters. We calculated the **average silhouette width** for each value of K, and the results were as follows:

- **K = 3**: Silhouette width = 0.181
- **K = 4**: Silhouette width = 0.184
- **K = 5**: Silhouette width = 0.13

After analyzing the **silhouette score**, we decided to choose **K = 4** as the optimal number of clusters. This is because the **silhouette width** was highest at K=4, indicating that the clusters at this value had the best internal cohesion and separation between them. In contrast, the silhouette score was lower for K=3and K=5, suggesting that the clusters at these values were less distinct.





Finally, both models are helpful for capturing various factors contributing to car accidents and have helped us reach our goal of enhancing road safety, optimizing traffic management, and providing valuable insights to inform policy decisions. However, since our data contains class labels, this makes Supervised Learning models (such as classification) more accurate and suitable for this problem compared to Unsupervised Learning models (such as clustering). The key advantage is that with class labels, we can use the known expected output to train the model, leading to more reliable and precise results in predicting accident-related outcomes.

The results shown in the tree indicate the following:

If an accident occurs early in the day (Hour ≤ 0.674) and involves less severe injuries (Injury Type ≤ 2.5), the risk of the accident is typically classified as Low Risk. However, if the accident occurs later in the day (Hour > 0.413), the risk increases and is predicted to be High Risk.

If the accident happens later in the day (Hour > 0.674), there is a greater chance of High Risk or Moderate Risk, depending on the exact hour and injury type. Additionally, High Injury Type (greater than 2.5) is often linked with Moderate Risk, based on the time of the accident.

Solutions:

To address these findings, the following measures are recommended:

Target enforcement and traffic management measures during high-risk hours (e.g., after 0.413 and during evening/nighttime).

Increased police presence and traffic control during these hours can reduce reckless driving and improve road safety.

Focus on improving infrastructure in accident hotspots.

Enhance lighting, install clear signage, and implement speed control mechanisms (such as speed bumps or speed cameras) to improve visibility and reduce speeding, especially in high-risk areas.

Use data-driven traffic management systems to predict and mitigate risks in real-time.

Implement adaptive traffic signals and predictive traffic management systems that respond dynamically to traffic conditions and accident patterns, helping prevent congestion and manage risks efficiently.

Increase public awareness campaigns and training programs to encourage safer driving habits.

Launch public awareness campaigns that emphasize safe driving practices, especially during high-risk times (like early mornings and late evenings). Additionally, provide training programs focused on defensive driving and fatigue management to reduce accidents caused by driver error or drowsiness.

By focusing on these solutions, we can enhance road safety, reduce the likelihood of high-risk accidents, and improve overall traffic management.

8. References

1. J. Divakar, "Car Crash Dataset," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/datasets/jacksondivakarr/car-crash-dataset?resource=download. [Accessed: 16-Nov-2024].