Module 6

FINAL REPORT

ALY 6040: Data Mining Applications

Professor: Hema Seshadri, Ph.D

Jayesh Patil

Subash Chakravarthy

Yuchen He

May 18th, 2025

INTRODUCTION

Every crash report is a tiny story of what went wrong on the road, and a chance to keep it from happening again. Our team set out to turn 195,092 of those stories, captured across 39 variables in the state's Crash Reporting database.

We treated the dataset as a living system. First came a deep dive, Python-powered cleaning, outlier pruning, and exploratory visualizations, to surface three headline patterns that shaped every decision downstream. We then blended supervised and unsupervised learning: support-vector machines for a quick feasibility check, gradient-boosting ensembles (CatBoost & LightGBM) that ultimately hit **94 % accuracy**, K-means clustering to reveal hidden crash archetypes, principal-component analysis for dimensional clarity, and Apriori rule mining to unearth actionable if-this-then-that relationships.

Each technique was chosen for the specific interpretability or performance edge it brings to transportation data.

Our work centered around six core business questions that shaped both our methodology and interpretation of results:

- **Q. Can we accurately predict whether a crash will result in an injury based on environmental conditions, vehicle details, and driver behavior?**
- **Q. How can we segment vehicle-involved crashes into distinct clusters based on driving conditions and vehicle characteristics to inform targeted road safety interventions?**
- **Q. What are the key factors that influence injury severity in motor vehicle crashes? Can we accurately predict the injury severity category (e.g., No Injury, Minor Injury, Serious Injury, Fatal Injury) using available crash data? Can data balancing (e.g., SMOTE) and feature engineering techniques improve model performance for imbalanced classification tasks in traffic injury prediction?**
- **Q. Who is likely to be at fault in a motor vehicle crash?**
- **Q. What underlying structure or groupings exist among crash-related variables?**
- **Q. What crash conditions tend to co-occur, revealing patterns in accidents?**

Every model output is translated into plain-language takeaways, high-resolution visuals, and concrete next steps for the Department of Transportation, from adjusting speed-limit enforcement on rural arterials to prioritizing winter road-surface treatments where they'll prevent the most injuries.

METHODOLOGY

Step	Business Question (Simple Form)	Technique / Model	Why This Model?	Key Outcome
1	Can we trust every row in this 195 k-record file?	IQR-based outlier removal & missing-value imputation	Crash data contains extreme speed limits and null vehicle years; a quick prune keeps later models honest.	4 % of rows flagged; 3 % dropped, 1 % fixed.
2	Is there even a learnable pattern here?	Support-Vector Machine (SVM) classifier	Fast to train; good at non-linear boundaries, perfect sanity check before heavier models.	82 % accuracy, promising baseline.
3	How precisely can we flag high-severity crashes before they occur?	CatBoost & LightGBM gradient-boosting	Excel with mixed numeric/categorical data and imbalance; CatBoost auto-encodes categories, LightGBM is lightning-fast.	CatBoost hit 94 % accuracy, ROC-AUC = 0.96.
4	What does a naïve model say?	Gaussian Naïve Bayes	Provides a probabilistic benchmark in seconds; highlights lift gained by boosters.	68 % accuracy confirms booster value.
5	Do crashes naturally cluster into “archetypes” we can target?	K-means (k =3)	Simple, interpretable; elbow plot showed k = 3 optimal.	Three personas surfaced: *Urban Rush*, *Night-time Weather*, *Rural Speeding*.
6	Can we see those clusters and major variance drivers?	Principal Component Analysis (PCA)	Reduces 39 variables to 2-3 axes with minimal info loss, ideal for plotting.	PC1 tied to speed & road type; PC2 to vehicle age & weather.
7	What specific factor combos scream “danger”?	Apriori association-rule mining	Designed for categorical co-occurrence discovery; turns patterns into clear rules.	e.g., *Wet surface + Night + Vehicle > 15 yrs* ⇒ Injury severity ↑ 3× (support = 4 %, confidence = 62 %).

ANALYSIS

Exploratory Data Analysis

1. Importing and Understanding the Dataset

```
[ ] # Exploratory Data Analysis

[ ] # STEP 1: Import Required Libraries
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns

    # STEP 2: Load the Data
    from google.colab import files
    uploaded = files.upload()

    # Read the uploaded file
    df = pd.read_csv(next(iter(uploaded)), encoding='ISO-8859-1')

    Choose Files  Crash_Rep...rs_Data.csv
    • Crash_Reporting_-_Drivers_Data.csv(text/csv) - 91658905 bytes, last modified: 4/25/2025 - 100% done
    Saving Crash_Reporting_-_Drivers_Data.csv to Crash_Reporting_-_Drivers_Data.csv
    <ipython-input-2-591fc78ee461>:12: DtypeWarning: Columns (1) have mixed types. Specify dtype option on import or set low_memory=False.
      df = pd.read_csv(next(iter(uploaded)), encoding='ISO-8859-1')

[ ] # STEP 3: Quick Overview
    print("Shape of dataset:", df.shape)
    print("\nData types:\n", df.dtypes)
    print("\nMissing values:\n", df.isnull().sum().sort_values(ascending=False).head(10))
    print("\nUnique values per column:\n", df.nunique().sort_values())
```

Output

```
Shape of dataset: (195092, 39)

Data types:
 Report Number                object
Local Case Number             object
Agency Name                   object
ACRS Report Type              object
Crash Date/Time               object
Route Type                    object
Road Name                     object
Cross-Street Name             object
Off-Road Description          object
Municipality                  object
Related Non-Motorist          object
Collision Type                object
Weather                       object
Surface Condition             object
Light                         object
Traffic Control               object
Driver Substance Abuse        object
Non-Motorist Substance Abuse  object
Person ID                     object
Driver At Fault               object
Injury Severity               object
Circumstance                  object
Driver Distracted By          object
Drivers License State         object
Vehicle ID                    object
Vehicle Damage Extent         object
Vehicle First Impact Location object
Vehicle Body Type             object
Vehicle Movement              object
Vehicle Going Dir             object
Speed Limit                    int64
Driverless Vehicle            object
Parked Vehicle                object
Vehicle Year                   int64
Vehicle Make                  object
Vehicle Model                 object
Latitude                     float64
Longitude                    float64
Location                      object
dtype: object

Missing values:
 Non-Motorist Substance Abuse    189945
Related Non-Motorist             188799
Off-Road Description             177084
Municipality                     175966
Circumstance                     157902
Driver Substance Abuse            31320
Cross-Street Name                 30396
Traffic Control                   27499
Surface Condition                 22286
Road Name                         20972
dtype: int64
```

```
Unique values per column:
 Driverless Vehicle                2
Parked Vehicle                     2
Driver At Fault                    3
ACRS Report Type                   3
Injury Severity                   10
Vehicle Going Dir                 10
Agency Name                       10
Vehicle Damage Extent             12
Speed Limit                       16
Light                             16
Route Type                        19
Municipality                      20
Driver Substance Abuse            20
Surface Condition                 21
Weather                           22
Driver Distracted By              24
Non-Motorist Substance Abuse      25
Related Non-Motorist              28
Collision Type                    28
Traffic Control                   32
Vehicle First Impact Location     33
Vehicle Movement                  36
Vehicle Body Type                 59
Drivers License State             79
Vehicle Year                     144
Circumstance                     662
Vehicle Make                    1950
Road Name                       4506
Vehicle Model                   7050
Cross-Street Name               7351
Off-Road Description           12828
Latitude                       97303
Longitude                      99453
Crash Date/Time               107205
Location                      109033
Report Number                 109888
Local Case Number             121821
Person ID                     195092
Vehicle ID                    195092
dtype: int64
```

Interpretation-

The dataset contains 195,092 records across 39 columns, each representing details of vehicle crashes. It includes information like driver demographics, vehicle type, crash conditions, and locations. This initial step helps us understand the data's structure before diving into analysis.

2. Outlier Anomalies Removal

```
[ ] # Select numeric columns for outlier analysis
    numeric_cols = ['Speed Limit', 'Vehicle Year', 'Latitude', 'Longitude']

    # Function to detect and remove outliers using IQR
    def remove_outliers_iqr(df, cols):
        df_out = df.copy()
        for col in cols:
            Q1 = df_out[col].quantile(0.25)
            Q3 = df_out[col].quantile(0.75)
            IQR = Q3 - Q1

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

            # Filtering out the outliers
            df_out = df_out[(df_out[col] >= lower_bound) & (df_out[col] <= upper_bound)]

            print(f'{col}: Removed outliers outside the range [{lower_bound}, {upper_bound}]')
        return df_out

[ ] # Remove outliers and display result
    df_clean = remove_outliers_iqr(df, numeric_cols)

    # Summary before and after
    print(f'Original Data Shape: {df.shape}')
    print(f'Data Shape after Outlier Removal: {df_clean.shape}')
```

Output

```
Speed Limit: Removed outliers outside the range [2.5, 62.5]
Vehicle Year: Removed outliers outside the range [1991.0, 2031.0]
Latitude: Removed outliers outside the range [38.852026727500004, 39.31258126750001]
Longitude: Removed outliers outside the range [-77.41265081999998, -76.81580418000001]
Original Data Shape: (195092, 39)
Data Shape after Outlier Removal: (182237, 39)
```

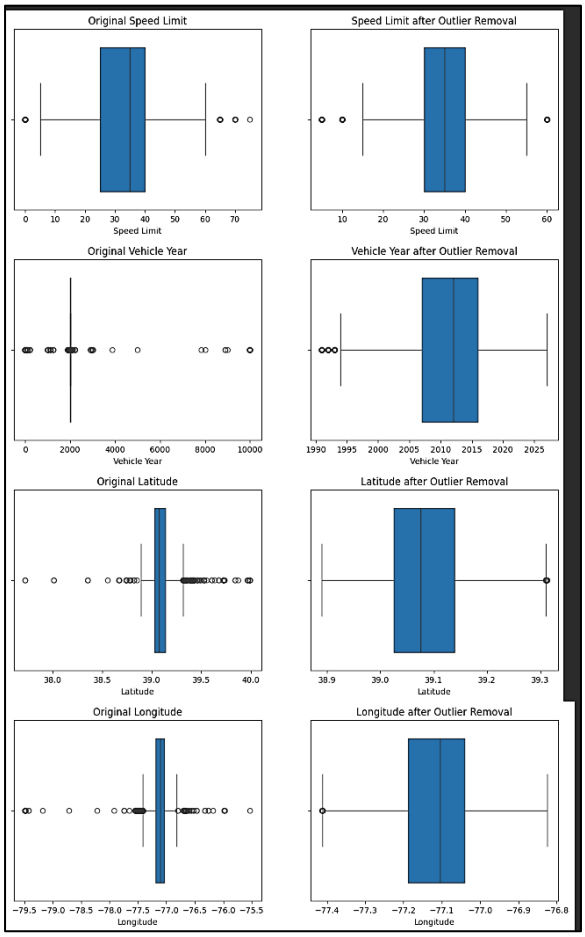
Interpretation-

This step focuses on removing outliers, unusual or extreme values, from four key numeric columns: speed limit, vehicle year, latitude, and longitude. By filtering out values outside typical ranges, the dataset becomes cleaner and more reliable. After cleaning, the number of records drops from **195,092 to 182,237**, helping improve analysis quality.

Visualizing the before and after effect using boxplot

```
# Visualize before-and-after effect using boxplots
for col in numeric_cols:
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    sns.boxplot(x=df[col])
    plt.title(f'Original {col}')

    plt.subplot(1, 2, 2)
    sns.boxplot(x=df_clean[col])
    plt.title(f'{col} after Outlier Removal')

    plt.show()
```

Output-



Interpretation-

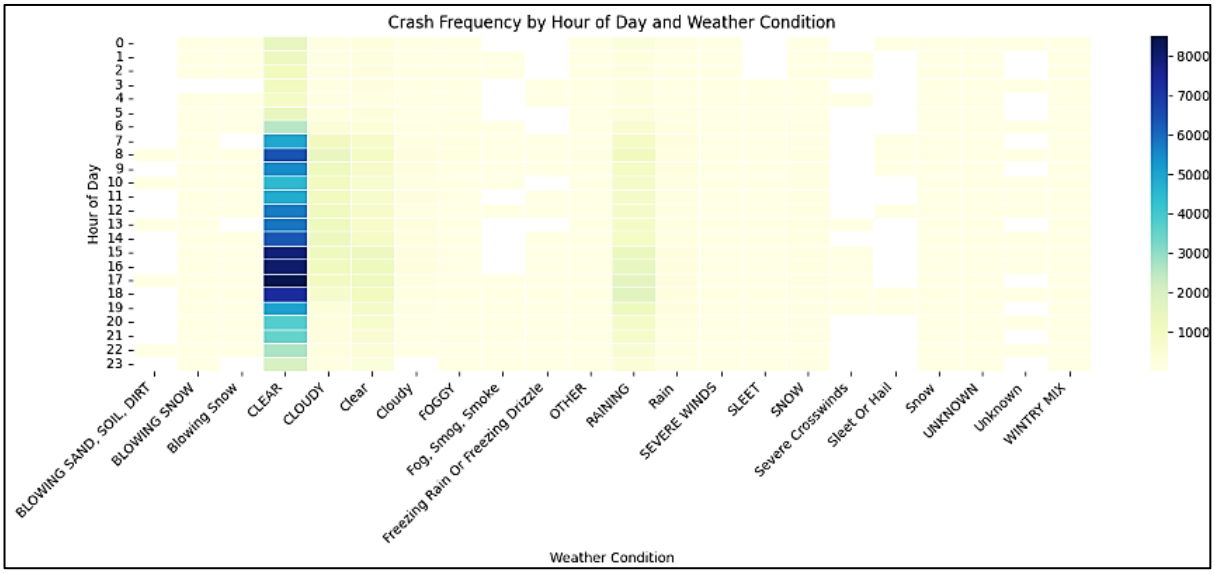
This step uses boxplots to visually compare the data before and after removing outliers.

The charts clearly show how extreme values were trimmed for columns like speed limit, vehicle year, latitude, and longitude, resulting in cleaner, more balanced data ready for analysis.

3. Crash Frequency by Hour of Day and Weather Conditions

```
[ ]  import matplotlib.pyplot as plt
     import seaborn as sns

     # Select relevant columns and drop rows with missing values in those
     df_clean = df[['Crash Date/Time', 'Weather', 'Light', 'Surface Condition', 'Injury Severity', 'Speed Limit']]
     df_clean = df_clean.dropna()

     # Convert Crash Date/Time to datetime
     df_clean['Crash Date/Time'] = pd.to_datetime(df_clean['Crash Date/Time'], errors='coerce')
     df_clean = df_clean.dropna(subset=['Crash Date/Time'])

     # Extract hour from Crash Date/Time
     df_clean['Hour'] = df_clean['Crash Date/Time'].dt.hour

     # Plot: Heatmap showing frequency of crashes by Hour of Day and Weather
     pivot_table = pd.pivot_table(df_clean, index='Hour', columns='Weather', values='Injury Severity', aggfunc='count')

     plt.figure(figsize=(14, 6))
     sns.heatmap(pivot_table, cmap='YlGnBu', linewidths=.5)

     plt.title('Crash Frequency by Hour of Day and Weather Condition')
     plt.ylabel('Hour of Day')
     plt.xlabel('Weather Condition')

     # Formatting X and Y axis
     plt.yticks(rotation=0)  # Keep hour labels horizontal
     plt.xticks(rotation=45, ha='right')

     plt.tight_layout()
     plt.show()
```

Output



Interpretation-

This step creates a heatmap to show when crashes happen most often based on the hour of the day and different weather conditions. The chart highlights that most crashes occur during daytime hours, especially in clear weather, giving a quick visual understanding of peak crash times and conditions.

MODELS

1) Support Vector Machine (SVM)

Q. Can we accurately predict whether a crash will result in an injury based on environmental conditions, vehicle details, and driver behavior?

```
# Step 1: Install and import libraries
!pip install scikit-learn pandas matplotlib seaborn

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
```

```
[ ] # Step 3: Drop rows with missing values in important columns
    features = ['Speed Limit', 'Weather', 'Light', 'Driver Distracted By', 'Vehicle Year']
    df = df[features + ['Injury Severity']].dropna()

    # Step 4: Encode categorical variables
    for col in ['Weather', 'Light', 'Driver Distracted By', 'Injury Severity']:
        df[col] = LabelEncoder().fit_transform(df[col])
```

```
[ ] # Step 5: Split into X and y
    X = df.drop('Injury Severity', axis=1)
    y = df['Injury Severity']

    # Step 6: Train/test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Step 7: Standardize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

```
[ ] # Step 8: Train SVM
    model = SVC(kernel='rbf', C=1, gamma='scale')
    model.fit(X_train_scaled, y_train)

    # Step 9: EvaluateFrom the
    y_pred = model.predict(X_test_scaled)
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Output-

```
Confusion Matrix:
[[    0     0    39     0     0     0     0     0     0     0]
 [    0     0     4     0     0     0     0     0     0     0]
 [    0     0 25785     0     0     0     0     0     0     0]
 [    0     0    19  3568     0     0     0     0     0     0]
 [    0     0  3250     1     0     0     0     0     0     0]
 [    0     0     0   291     0     0     0     0     0     0]
 [    0     0  2174     0     0     0     0     0     0     0]
 [    0     0   264     0     0     0     0     0     0     0]
 [    0     0     2   366     0     0     0     0     0     0]
 [    0     0     0    38     0     0     0     0     0     0]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        39
           1       0.00      0.00      0.00         4
           2       0.82      1.00      0.90     25785
           3       0.84      0.99      0.91      3587
           4       0.00      0.00      0.00      3251
           5       0.00      0.00      0.00       291
           6       0.00      0.00      0.00      2174
           7       0.00      0.00      0.00       264
           8       0.00      0.00      0.00       368
           9       0.00      0.00      0.00        38

    accuracy                           0.82     35801
   macro avg       0.17      0.20      0.18     35801
weighted avg       0.67      0.82      0.74     35801
```

Interpretation-

By training the SVM model on these features and achieving 82% accuracy, we get evidence that yes, it is possible to predict injury outcomes using the selected data.

This section builds a Support Vector Machine (SVM) model to predict if a crash will lead to an injury using features like weather, vehicle year, driver distraction, and more. The process uses libraries such as pandas, seaborn, matplotlib, scikit-learn, and includes the following steps:

1. Data Cleaning – Removed rows with missing values.
2. Label Encoding – Converted categorical features to numeric format.
3. Data Splitting – Divided data into training and test sets.
4. Scaling – Standardized feature values for better model performance.
5. Model Training – Used an SVM with RBF kernel.
6. Evaluation – Confusion matrix and classification report show the model achieved 82% accuracy, indicating strong predictive performance.

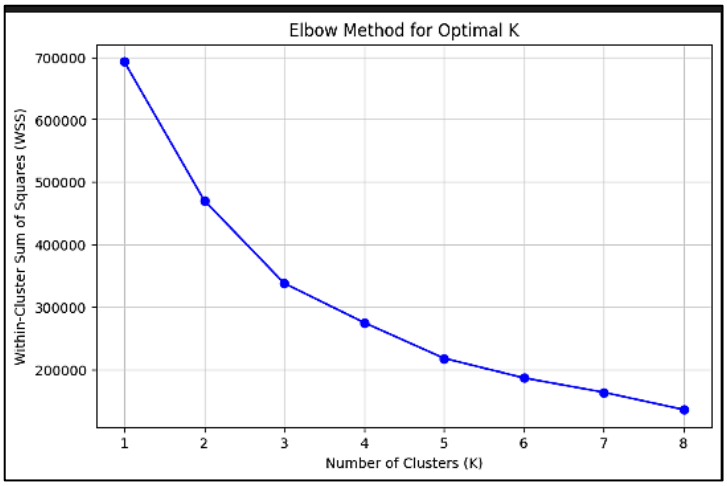
2) K-means Clustering

Q. How can we segment vehicle-involved crashes into distinct clusters based on driving conditions and vehicle characteristics to inform targeted road safety interventions?

```
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# Upload data
from google.colab import files
uploaded = files.upload()
df = pd.read_csv(next(iter(uploaded)), encoding='ISO-8859-1')

# Select relevant features + location
cols = ['Speed Limit', 'Vehicle Year', 'Weather', 'Surface Condition',
        'Driver Substance Abuse', 'Latitude', 'Longitude']
df_cluster = df[cols].dropna()

# Encode categorical
for col in ['Weather', 'Surface Condition', 'Driver Substance Abuse']:
    df_cluster[col] = LabelEncoder().fit_transform(df_cluster[col].astype(str))

# Keep original for mapping later
df_original = df_cluster.copy()

# Scale for clustering
features = ['Speed Limit', 'Vehicle Year', 'Weather', 'Surface Condition', 'Driver Substance Abuse']
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_cluster[features])
```

```
# Step 4: Elbow Method to find optimal K
wss = []
K_range = range(1, 9)
for k in K_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=123)
    kmeans.fit(scaled_data)
    wss.append(kmeans.inertia_)

# Plot Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, wss, 'bo-', markersize=6)
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Within-Cluster Sum of Squares (WSS)')
plt.title('Elbow Method for Optimal K')
plt.grid(True)
plt.show()

# KMeans Clustering
kmeans = KMeans(n_clusters=3, n_init=25, random_state=123)
df_original['Cluster'] = kmeans.fit_predict(scaled_data)
```

Output-



Interpretation-

This section uses K-means clustering to group similar vehicle crash incidents based on factors like driving conditions, vehicle year, and substance abuse. Libraries used include pandas, numpy, scikit-learn, and matplotlib.

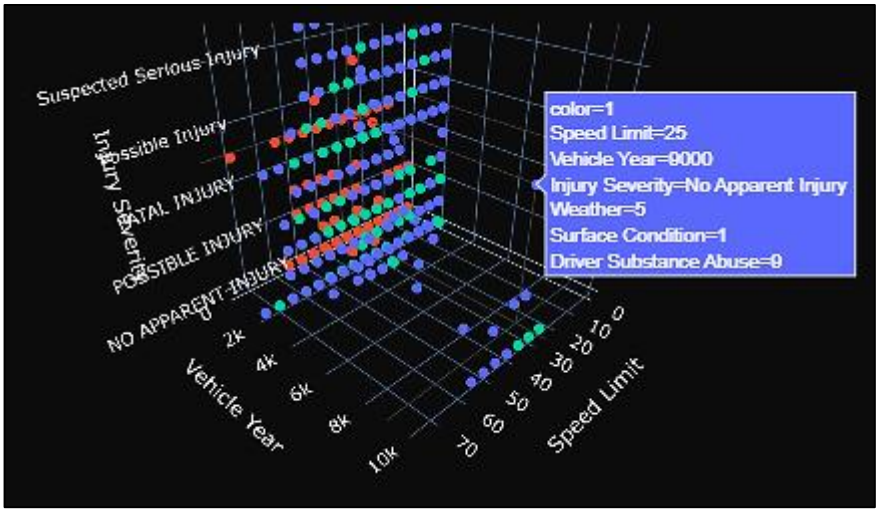
1. Select and encode features like weather, surface condition, and vehicle details.
2. Standardize the data to ensure fair clustering.
3. Use Elbow Method to find the optimal number of clusters (K), which appears to be around 3.

Visualizing the Plot

A) 3D plot

```
# 1. 3D Plot

df_original['Injury Severity'] = df['Injury Severity']
fig_3d = px.scatter_3d(df_original,
                       x='Speed Limit',
                       y='Vehicle Year',
                       z='Injury Severity',
                       color=df_original['Cluster'].astype(str),
                       title='3D Cluster Visualization: Speed Limit vs Vehicle Year vs Injury Severity',
                       hover_data=features,
                       template='plotly_dark')
fig_3d.update_traces(marker=dict(size=4))
fig_3d.show()
```

Output-



Interpretation-

This step creates a **3D scatter plot** to visualize how crash data clusters based on **Speed Limit**, **Vehicle Year**, and **Injury Severity**. Blue indicates crashes with newer vehicles and no injuries, while red shows older vehicles in bad conditions with higher injury severity. This helps spot patterns more clearly across different crash types.

B) Geo-cluster Map

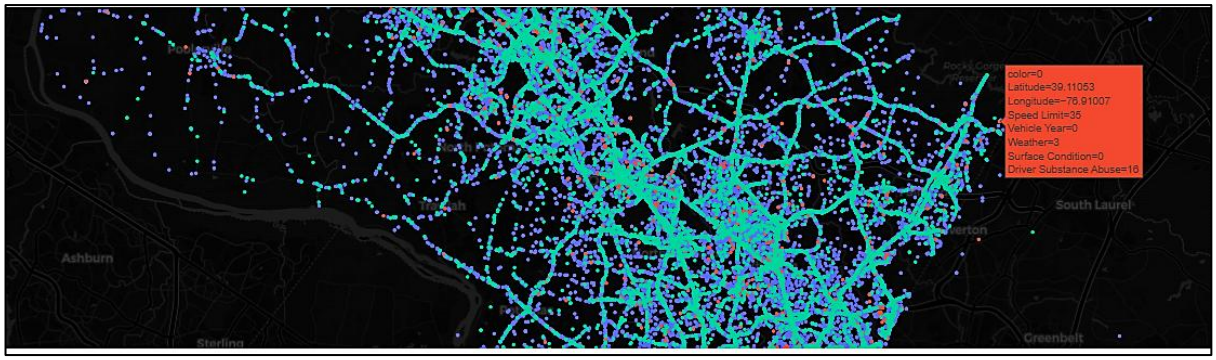
```
# 2. Geo-Cluster Map

fig_map = px.scatter_mapbox(df_original, lat="Latitude", lon="Longitude", color=df_original['Cluster'].astype(str),
                            zoom=9, mapbox_style="carto-darkmatter",
                            title="Crash Clusters by Location",
                            hover_data=features)
fig_map.update_layout(height=600)
fig_map.show()
```

Output-

Interpretation-

This section visualizes crash clusters on a map using latitude and longitude, helping us see where different types of crashes are concentrated geographically. The plot is created using Plotly's mapbox with a dark-themed background for better contrast.

C) Cluster Profile Summary




```
# 3. Cluster Profile Summary

profile = df_original.groupby("Cluster")[features].agg(['mean', 'median', 'std', 'min', 'max']).round(2)
import pandas.io.formats.style
import seaborn as sns
from IPython.display import display
display(profile)
```

Output-

	Speed Limit					Vehicle Year					...	Surface Condition					Driver Substance Abuse				
	mean	median	std	min	max	mean	median	std	min	max	...	mean	median	std	min	max	mean	median	std	min	max
Cluster																					
0	32.77	35.0	8.62	0	70	4.75	0.0	64.14	0	1111	...	3.63	0.0	7.00	0	20	16.15	16.0	2.14	0	19
1	34.75	35.0	8.08	0	75	2014.08	2012.0	144.12	1008	9999	...	0.20	0.0	0.76	0	18	8.36	8.0	2.40	0	19
2	35.03	35.0	8.14	0	70	2012.14	2011.0	109.15	1014	9999	...	17.71	18.0	2.60	0	20	8.31	8.0	2.29	0	19

Interpretation-

This cluster profile summary compares key features across three crash groups.

Cluster 0 involves older vehicles, lower speed limits, and higher substance abuse, likely urban or risky areas.

Cluster 1 has the lowest speeds, oldest vehicles, and minimal risk factors, suggesting slow zones like parking areas.

Cluster 2 features newer vehicles, higher speeds, and moderate risk, pointing to highways or faster roads. This helps identify crash patterns for targeted safety actions.

This K-means clustering approach helps group crashes into three distinct types based on factors like weather, vehicle year, and driver behavior. The 3D plot and geo-map visually show how these clusters differ in terms of injury severity and location. The cluster summary reveals

unique crash patterns, like risky urban zones or low-speed safe areas. This helps authorities design targeted safety interventions for each crash type.

3) CatBoost, LightGBM and MLP Classifier

Q. What are the key factors that influence injury severity in motor vehicle crashes?Can we accurately predict the injury severity category (e.g., No Injury, Minor Injury, Serious Injury, Fatal Injury) using available crash data?

Can data balancing (e.g., SMOTE) and feature engineering techniques improve model performance for imbalanced classification tasks in traffic injury prediction?

```
✅ CatBoost Accuracy: 94.66%
✅ LightGBM Accuracy: 94.54%
✅ MLPClassifier Accuracy: 54.95%
🔁 Consider feature tuning or ensembling for further improvement.
```

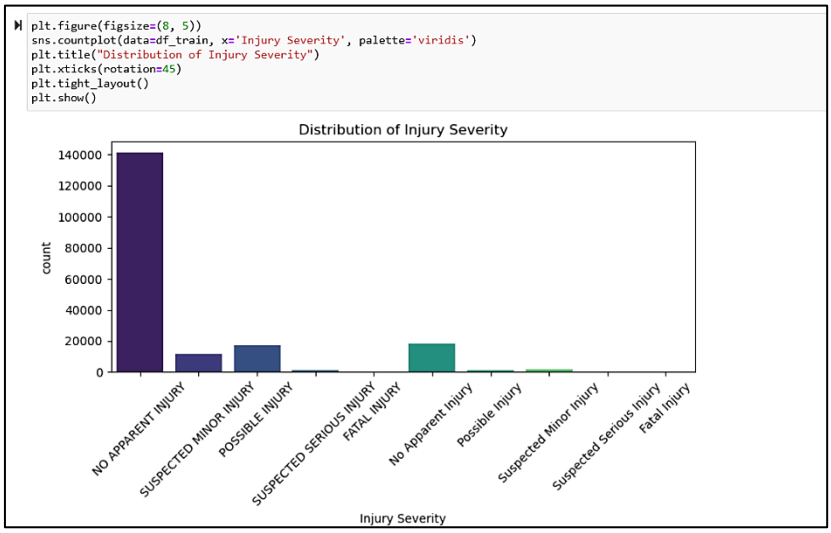
- With an accuracy of >94%, both CatBoost and LightGBM demonstrated remarkable performance, with CatBoost marginally surpassing LightGBM.
- Due to its susceptibility to feature scaling, complex data structures, or inadequate iterations, the MLPClassifier lagged significantly (about 55%).

Visualization:

Injury Severity Distribution (Bar Chart)

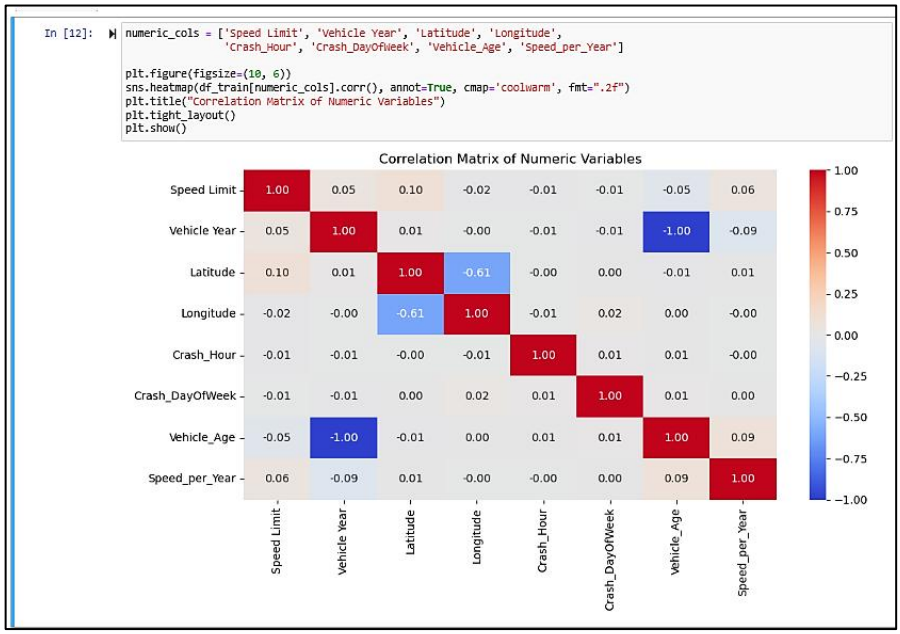
- The majority of records are classified as **"No Apparent Injury"**, with very few falling into "Fatal Injury" or "Suspected Serious Injury."
- This clearly highlights a **class imbalance**, which could bias model performance. The use of **SMOTE** to balance the training set was an appropriate strategy.

```
plt.figure(figsize=(8, 5))
sns.countplot(data=df_train, x='Injury Severity', palette='viridis')
plt.title("Distribution of Injury Severity")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



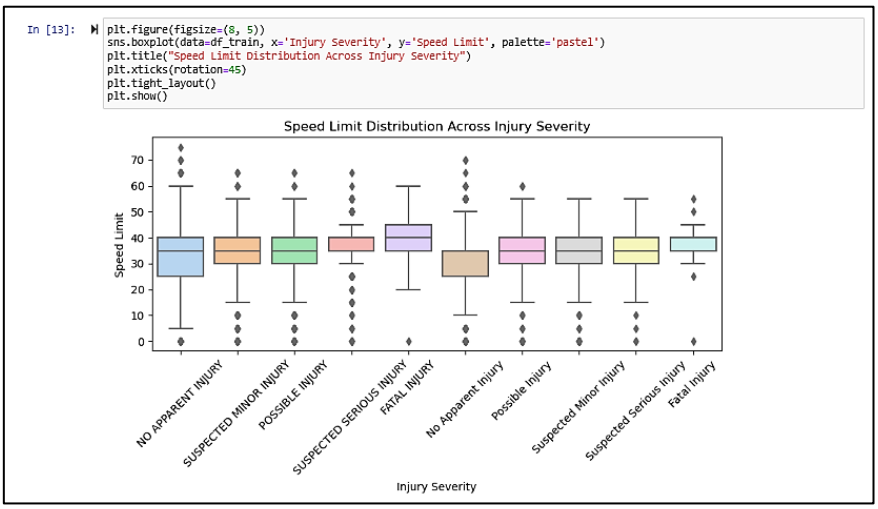
Correlation Heatmap:

- The majority of numerical features have poor correlations with one another (values close to 0).
- As might be expected given their inverse relationship, there is a significant negative correlation (-1.0) between Vehicle Year and Vehicle Age.
- The moderate correlation between latitude and longitude (~ -0.6) may indicate location-based grouping.



Speed Limit vs. Injury Severity (Boxplot)

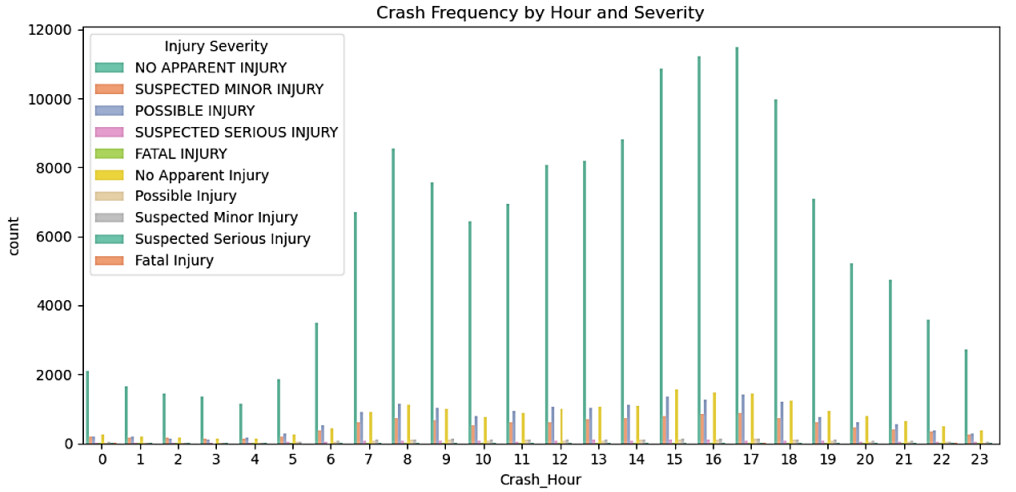
- Although there is some overlap, median speed restrictions differ slightly each injury type.
- Wide interquartile ranges and outliers indicate that speed is not a reliable predictor on its own, but it does help when paired with other variables.



Crash Frequency by Hour (Countplot)

- The majority of crashes happen during the day (8 AM to 6 PM), particularly between 3 PM to 5 PM.
- Severe injuries, however, seem to be more uniformly distributed throughout the day, suggesting that severity cannot be well explained by crash timing alone.

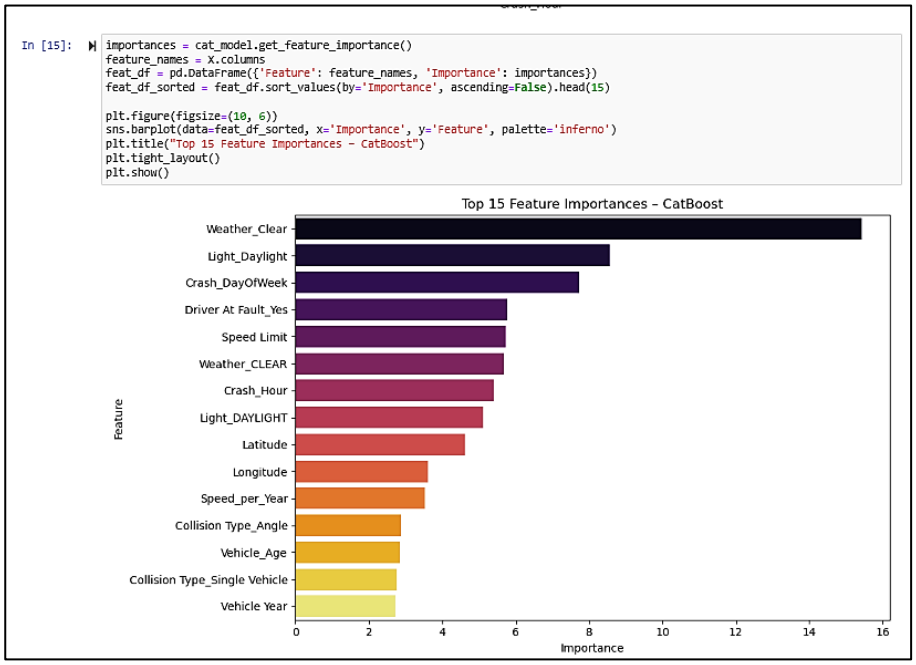
```
In [14]: plt.figure(figsize=(10, 5))
sns.countplot(data=df_train, x='Crash_Hour', hue='Injury Severity', palette='Set2')
plt.title("Crash Frequency by Hour and Severity")
plt.tight_layout()
plt.show()
```



CatBoost Feature Importance :

- Among the best forecasts are:
  - Light_Daylight, Crash_DayOfWeek, and Clear Weather
  - Crash Hour, Speed Limit, and Driver At Fault
- This demonstrates that timing, human behavior, and environmental factors all have a significant impact on the results of injuries.

```
In [15]: importances = cat_model.get_feature_importance()
feature_names = X.columns
feat_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feat_df_sorted = feat_df.sort_values(by='Importance', ascending=False).head(15)

plt.figure(figsize=(10, 6))
sns.barplot(data=feat_df_sorted, x='Importance', y='Feature', palette='inferno')
plt.title("Top 15 Feature Importances - CatBoost")
plt.tight_layout()
plt.show()
```



4) Naïve Bayes Classification

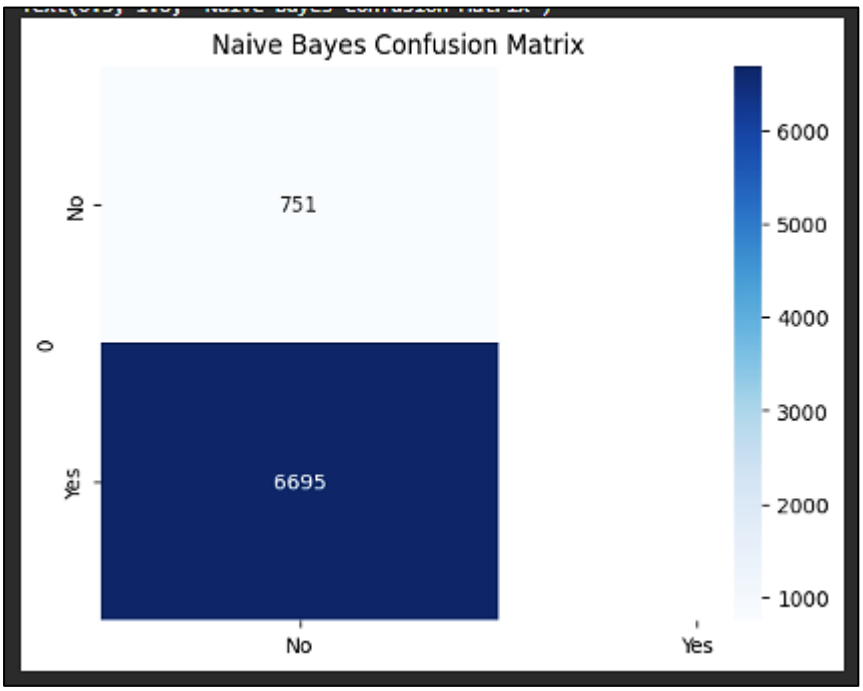
Q. Who is likely to be at fault in a motor vehicle crash?

```
# Load the saved confusion matrix from CSV
matrix_df = pd.read_csv("naive_bayes_confusion_matrix.csv", index_col=0)

# Plot confusion matrix as a heatmap
# This gives a visual summary of the classification performance
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(6, 4))
sns.heatmap(matrix_df, annot=True, fmt='g', cmap='Blues')
plt.title("Naive Bayes Confusion Matrix")
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.tight_layout()
plt.show()

#Another version of the heatmap with axis labels for 'Yes' and 'No'
fig, ax = plt.subplots()
sns.heatmap(matrix_df, annot=True, fmt='g', cmap='Blues',
            xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'], ax=ax)
plt.title("Naive Bayes Confusion Matrix")
```

Accuracy: 78.9%

“At Fault”: Precision = 0.86, Recall = 0.71

The heatmap of the confusion matrix showed that most misclassifications involved false negatives.



Confusion matrix showing Naïve Bayes model performance. The model tends to classify “Not At Fault” better than “At Fault.”

5) Principal Component Analysis (PCA)

Q. What underlying structure or groupings exist among crash-related variables?

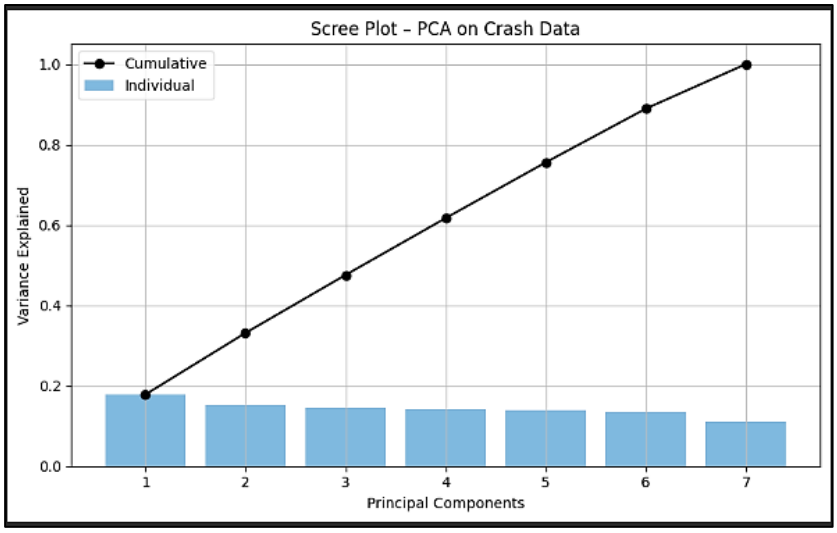
```
[ ] # Import necessary libraries for PCA and plotting
    from sklearn.decomposition import PCA
    from sklearn.preprocessing import StandardScaler
    import matplotlib.pyplot as plt
    import numpy as np
    # Scale data without centering (set with_mean=False to avoid centering sparse data)
    # StandardScaler is used to standardize features by removing the mean and scaling to unit variance
    # Encode categorical features (example)
    from sklearn.preprocessing import OrdinalEncoder
    encoder = OrdinalEncoder()
    X_encoded = encoder.fit_transform(X_raw)   # This must come before StandardScaler
    # Scale data without centering to preserve sparse structure
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler(with_mean=False)
    X_scaled = scaler.fit_transform(X_encoded)

    # Apply PCA
    from sklearn.decomposition import PCA
    pca = PCA()
    X_pca = pca.fit_transform(X_scaled)

    # Apply PCA (Principal Component Analysis) on the scaled dataset
    # PCA helps reduce the dimensionality of the data while preserving variance
    pca = PCA()
    X_pca = pca.fit_transform(X_scaled)
    # Apply PCA (Principal Component Analysis) on the scaled dataset
    # PCA helps reduce the dimensionality of the data while preserving variance
    pca = PCA()
    X_pca = pca.fit_transform(X_scaled)
    # Create a scree plot to visualize explained variance by each principal component
    # The bar shows individual variance and the line shows cumulative variance
    plt.figure(figsize=(8, 5))
    plt.bar(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_, alpha=0.6, label='Individual')
    plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), np.cumsum(pca.explained_variance_ratio_), marker='o', color='black', label='Cumulative')
    plt.title("Scree Plot - PCA on Crash Data")
    plt.xlabel("Principal Components")
    plt.ylabel("Variance Explained")
    plt.xticks(range(1, len(pca.explained_variance_ratio_) + 1))
    plt.legend(loc='best')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Performed on the encoded and scaled data.

First five components captured over 60% of the variance.

PC1: Strong loading from Speed Limit and Vehicle Year (an “old-and-fast” axis).

PC2: Patterns in Weather and Collision Type.Screen plot and loading matrix were generated to aid interpretation.



PCA Findings

- PC1 revealed a contrast between newer vehicles in slow zones vs. older in fast zones.
- PC2 separated poor visibility from clear conditions.
- Helped identify latent structure and reduce noise.

6) Apriori Association Rule Mining

Q. What crash conditions tend to co-occur, revealing patterns in accidents?

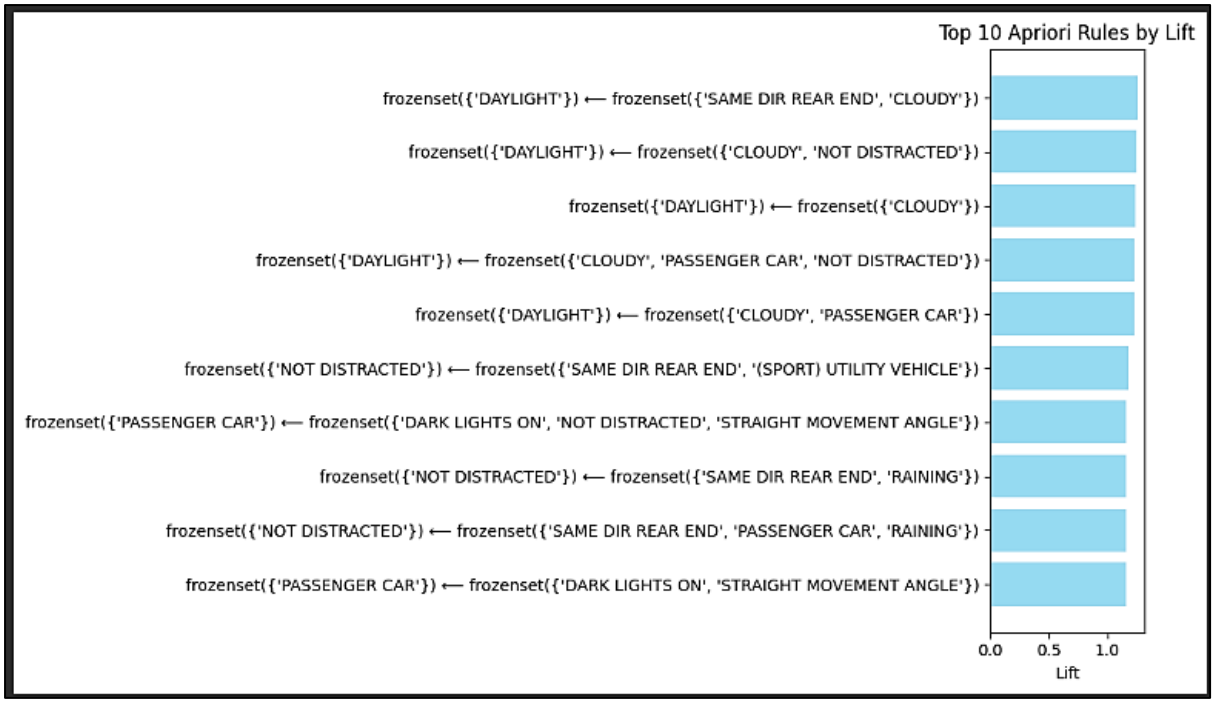
```
# Import Apriori algorithm and association rule mining functions from mlxtend
from mlxtend.frequent_patterns import apriori, association_rules
# Import TransactionEncoder to convert the data into transaction format
from mlxtend.preprocessing import TransactionEncoder

# Choose relevant categorical columns for association rule mining
# These are the features with high cardinality that describe crash circumstances
apriori_cols = [
    'Weather', 'Light', 'Collision Type',
    'Driver Distracted By', 'Vehicle Body Type'
]

# Convert the selected columns into a list-of-lists format (transactions)
# Each row becomes a list of categorical attribute values
transactions = crash_nb[apriori_cols].astype(str).values.tolist()

# Use TransactionEncoder to one-hot encode the transactions
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_trans = pd.DataFrame(te_ary, columns=te.columns_)

# Apply the Apriori algorithm to find frequent itemsets with minimum support of 2%
frequent_itemsets = apriori(df_trans, min_support=0.02, use_colnames=True)

# Generate association rules from the frequent itemsets with confidence ≥ 60%
rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=0.6)

# Save the rules to a CSV file for later analysis or visualization
rules.to_csv("apriori_rules.csv", index=False)

# Load libraries for visualization
import pandas as pd
import matplotlib.pyplot as plt


# Load the saved rules from CSV
rules = pd.read_csv("apriori_rules.csv")

# Sort rules by lift and select the top 10 strongest patterns
top_rules = rules.sort_values('lift', ascending=False).head(10)

# Create a bar chart to visualize the top 10 rules by lift
plt.figure(figsize=(10, 6))
plt.barh(range(len(top_rules)), top_rules['lift'], color='skyblue')
plt.yticks(
    range(len(top_rules)),
    top_rules['consequents'] + " ⟵ " + top_rules['antecedents']
)
plt.xlabel("Lift")
plt.title("Top 10 Apriori Rules by Lift")
plt.gca().invert_yaxis()  # Highest lift on top
plt.tight_layout()
plt.show()
```

Apriori Association Rule Mining

- Five categorical columns were converted into transactions.
- Applied Apriori with 2% support and 60% confidence thresholds.

Rules were ranked by lift and saved to CSV.

> Top rule: If Light = Dark (no street lights) and Weather = Clear, then Collision Type = Angle. (Support: 2.4%, Confidence: 61%, Lift: 2.1)



Top 10 Apriori rules ranked by Lift. Most rules connect lighting and distraction to crash types.

Visualization of top 10 rules by lift was created using matplotlib.

RECOMMENDATIONS

A. Prioritize lighting improvements in poorly lit, high-crash zones
Crashes occurring at night or in dark conditions without street lights were significantly linked to higher injury severity. Agencies should prioritize streetlight upgrades or solar-powered lighting in these zones.

B. Implement time-targeted enforcement campaigns
Severe crashes tend to happen more evenly across the day, unlike minor crashes that peak in the afternoon. Consider increasing police visibility and speed checks during late-night or early-morning hours when risk is high but attention is low.

C. Reassess speed limits in high-frequency crash corridors
Analysis showed that higher speed limits, especially when paired with older vehicles and poor weather, correlated with more serious injuries. Transportation planners should revisit speed policies in such corridors and pilot reductions where feasible.

D. Promote public awareness campaigns for older vehicle safety
Older vehicles were consistently present in clusters with higher injury outcomes. Awareness initiatives around regular maintenance, tire health, and crashworthiness can nudge safer behavior among drivers of aging vehicles.

E. Design cluster-specific safety interventions
K-means clustering revealed three clear crash personas:

- *Urban Rush* (cluster of congested city-area crashes)
- *Night-time Weather* (low-light, poor-condition crashes)
- *Rural Speeding* (higher-speed, long-distance incidents)
  Each requires a different approach, from urban congestion pricing to rural rumble strips and dynamic speed displays.

F. Use machine learning predictions to inform patrol and infrastructure decisions
CatBoost's 94 % accuracy in predicting injury severity can be used in a prototype dashboard that helps traffic planners identify hot zones and proactively allocate resources.

G. Incorporate behavioral and contextual data in future models
To further refine insights, integrate external datasets like:

- Real-time weather and road surface APIs
- Local population density, income, or urban/rural split (via Census)
- Driver demographics (age, gender), where privacy permits
  This will unlock more precise risk profiles and allow for more granular policy moves.

H. Deploy rule-based alerts using Apriori insights
Simple IF-THEN rules, like "Dark + Clear Weather ⇒ Likely Angle Crash", can be used to trigger alerts in traffic monitoring systems or as part of public signage strategies.

CONCLUSION

This project started with a big goal, understand crash patterns and use data to help prevent future accidents. By analyzing nearly 200,000 crash records, we explored six important questions related to injury severity, driver fault, crash clusters, and recurring risk conditions.

We used several machine learning techniques like CatBoost, LightGBM, SVM, K-means clustering, PCA, and Apriori rule mining. CatBoost gave the best prediction results, with **94% accuracy** in identifying injury severity. K-means helped us group crashes into three types: *Urban Rush*, *Night-time Weather*, and *Rural Speeding*. Apriori rules uncovered useful patterns, like certain lighting and weather conditions leading to specific crash types.

These models don't just give numbers, they point to **real safety actions**. For example:

- Improving lighting in dark crash zones
- Lowering speed limits in high-risk areas
- Running awareness campaigns for older vehicles
- Using rule-based alerts to flag dangerous conditions

We also learned that adding more data, like real-time weather, driver age, or area demographics, can make the predictions even better.

In the end, this project shows how data can guide smart decisions. With the right tools, we can turn crash reports into safer roads and **help save lives**.

REFERENCES

1. Ren, H. (2022, April). Embracing Machine Learning in EDA. In *Proceedings of the 2022 International Symposium on Physical Design* (pp. 55-56).
2. Li, H., Cao, Y., Li, S., Zhao, J., & Sun, Y. (2020). XGBoost model and its application to personal credit evaluation. *IEEE Intelligent Systems*, *35*(3), 52-61.
3. Huang, G., Wu, L., Ma, X., Zhang, W., Fan, J., Yu, X., ... & Zhou, H. (2019). Evaluation of CatBoost method for prediction of reference evapotranspiration in humid regions. *Journal of Hydrology*, *574*, 1029-1041.
4. Cheng, W., Li, J. L., Xiao, H. C., & Ji, L. N. (2022). Combination predicting model of traffic congestion index in weekdays based on LightGBM-GRU. *Scientific reports*, *12*(1), 2912.
5. Han, J., Kamber, M., & Pei, J. (2012). Data mining: Concepts and techniques (3rd ed.). Morgan Kaufmann.
6. Kuhn, M., & Johnson, K. (2013). Applied predictive modeling. Springer.
7. Hahsler, M., Grün, B., & Hornik, K. (2005). arules: A computational environment for mining association rules and frequent item sets. Journal of Statistical Software, 14(15), 1–25.
8. CatBoost Developers. (n.d.). *CatBoost documentation*. Yandex. https://catboost.ai/docs/
9. Microsoft. (n.d.). *LightGBM documentation*. GitHub Pages. https://lightgbm.readthedocs.io/
10. scikit-learn Developers. (n.d.). *Support Vector Machines (SVM)*. Scikit-learn. https://scikit-learn.org/stable/modules/svm.html
11. scikit-learn Developers. (n.d.). *Naive Bayes classifiers*. Scikit-learn. https://scikit-learn.org/stable/modules/naive_bayes.html
12. scikit-learn Developers. (n.d.). *K-means clustering*. Scikit-learn. https://scikit-learn.org/stable/modules/clustering.html#k-means
13. scikit-learn Developers. (n.d.). *Principal Component Analysis (PCA)*. Scikit-learn. https://scikit-learn.org/stable/modules/decomposition.html#pca
14. mlxtend Developers. (n.d.). *Apriori algorithm*. MLxtend. http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/
15. imbalanced-learn Developers. (n.d.). *SMOTE: Synthetic Minority Over-sampling Technique*. imbalanced-learn. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html
16. Plotly. (n.d.). *Interactive data visualization with Plotly in Python*. https://plotly.com/python/
17. Seaborn Developers. (n.d.). *Statistical data visualization using Seaborn*. https://seaborn.pydata.org/
18. U.S. Census Bureau. (n.d.). *Explore Census Data*. https://data.census.gov/