TripSafe: Predicting Traffic Accident Risk Using Machine Learning on US Accident Data

Mohammed Ansari (17432883) and Nurislam Saliev (018157828)
San José State University
May 2025

Abstract

Traffic accidents pose a major public safety risk, particularly in highly populated regions like California. This paper introduces TripSafe, a machine learning model designed to predict the risk level of traffic accidents based on user input data such as location, time, and weather conditions, enabling drivers to make more informed travel decisions. The data pipeline processes real-world traffic data through an initial preprocessing stage using Bash, followed by advanced data cleaning, transformation, and analysis in Python. We utilize K-means clustering to define risk levels, apply PCA for dimensionality reduction, and train multiple classifiers (Logistic Regression, SVM, Random Forest) to predict these risk categories. Our best model achieved 98% accuracy with high precision across all clusters.

Introduction

Traffic accidents are a leading cause of injury and death in the United States, resulting in around 41,910 fatalities according to NHTSA. While real-time traffic navigation apps provide live traffic updates, they often lack the ability to provide predictive insights into road safety. This is where TripSafe comes in - a machine learning model designed to predict accident risk based on location, time, and weather conditions.

Objectives

- Predict traffic accident risk levels (Low, Medium, High) for a given time and location.
- Translate raw traffic accident data into a trainable dataset.
- Develop a machine learning model that supports risk-aware travel planning.

Business Value

TripSafe's predictive capability can be of value to:

- Navigation platforms (e.g., Google Maps, Apple Maps) for advanced accident risk warnings.
- Logistics companies to reroute fleets based on safety forecasts.

Literature Review

Many researchers have used machine learning to predict traffic accidents. One recent study by M. Girija and V. Divya (2024), titled *"Deep Learning-Based Traffic Accident Prediction,"* used a deep learning model called a Convolutional Neural Network (CNN) to predict accidents. Their model used data like weather, road conditions, traffic volume, and past accident reports. They also used real-time tracking tools to follow vehicles and understand movement patterns. They focused on images and spatial data to find patterns and improve road safety.

While their work is extremely impressive, our project takes a different approach. Instead of deep learning, we use traditional machine learning models like Logistic Regression, Random Forest, and SVM, which have faster inference time and are lightweight. Also, instead of using image or real-time video data, we use historical accident data in a CSV format.

Another big difference is that they tried to predict whether an accident would happen, while we focus on **classifying the risk level** (low, medium, high) for any given situation. Our project is built for use in navigation apps and safer trip planning, making it more practical for real-world use.

Key Ideas from Existing Research:

- Use of weather, traffic, and road data for predictions
- Use of CNNs for extracting patterns from images
- Real-time tracking to improve prediction accuracy

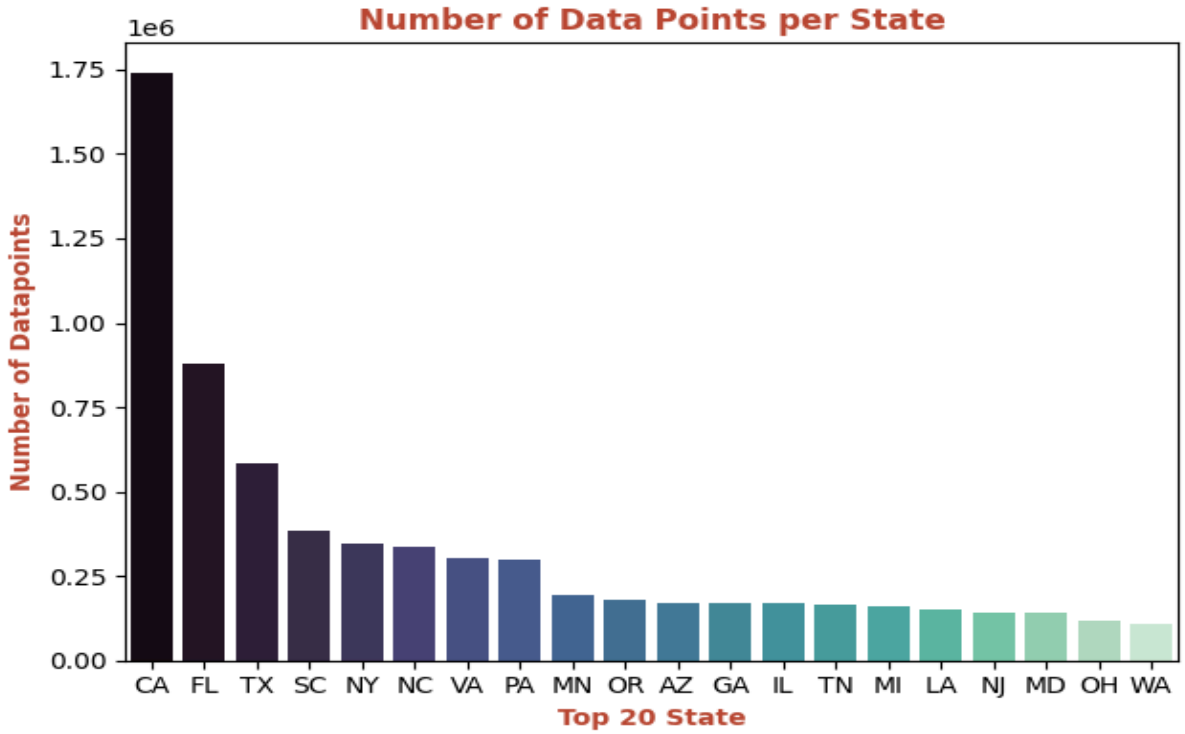
What Makes Our Project Different:

- We use simpler and faster models that still give high accuracy
- Our data is tabular and doesn't need images or video
- We focus on classifying risk levels, not just predicting if an accident will happen
- Our model is easy to use in real apps like Google Maps or fleet tracking tools

Dataset

The dataset used in this project is derived from the US Accidents (2016–2023) dataset available on Kaggle. It contains over 7.7 million accident records from across the United States, spanning 48 columns. These columns include a wide range of attributes such as state, city, location coordinates, weather conditions, and detailed time stamps.

Figure 1. Data distribution across states



Data Cleaning

Initial exploratory data analysis and cleaning was performed on Bash. And While performing our initial EDA on bash we noticed that a major part of the dataset in California accidents and the

accident from the other states is sparse(Figure 1). As a result, we narrowed our focus to California to ensure better model accuracy and consistency.

Figure 2. Raw Data

	ID	Source	Severity	Start_Time	End_Time	Start_Lat	Start_Lng	End_Lat	End_Lng	Distance(mi)	...	Roundabout	Station	Stop	Traffic_Calming	Traffic_Signal	Turning_Loop	Sunrise_Sunset	Civil_Twilight	Nautical_Twilight	Astronomical_Twilight
0	A-1	Source2	3	2016-02-08 05:46:00	2016-02-08 11:00:00	39.865147	-84.058723	NaN	NaN	0.01	...	False	False	False	False	False	False	Night	Night	Night	Night
1	A-2	Source2	2	2016-02-08 06:07:59	2016-02-08 06:37:59	39.928059	-82.831184	NaN	NaN	0.01	...	False	False	False	False	False	False	Night	Night	Night	Day
2	A-3	Source2	2	2016-02-08 06:49:27	2016-02-08 07:19:27	39.063148	-84.032608	NaN	NaN	0.01	...	False	False	False	False	True	False	Night	Night	Day	Day

The original dataset had 48 columns(Figure 2). Using logical reasoning and project relevance, we dropped 36 columns that were either redundant or not useful for our project. Only columns with potential value to the modeling process were retained.

We then removed duplicate records and dropped rows with excessive missing values to reduce noise and improve data quality(Figure 3). At the end of this step, we had about 1.7 million rows and 16 columns.

Figure 3. Some of the Bash Command used

```
# Extract California Data Only
awk -F',' 'NR==1 || $15 == "CA"' US_Accidents.csv > CA_Accidents.csv
# Removes rows with more than 9 missing values
awk -F',' 'NR>1 { empty = 0 for (i = 1; i <= NF; i++) if ($i == "") empty++ if (empty > 9) count++} END { print "Count:", count }' CA_Accidents.csv
# Check Severity Column Distribution
tail -n +2 CA_Accidents.csv | cut -d',' -f3 | sort | uniq -c
```

Then in Python, we worked on handling the missing values. We used simple and standard methods, filling in categorical columns with the most common value (mode) and numerical ones with either the mean or median, depending on how skewed the data was(if the data skewness was closer to 0 we filled the column with mean,otherwise median). If a column had more than 200,000 missing values, we just dropped it altogether, since filling in that much missing data would make the column less reliable. By the end of this step, we were left with around 1.1 million rows and 16 cleaned columns.

Figure 4. Data after cleaning

	Severity	Start_Time	End_Time	Distance(mi)	City	County	Zipcode	Temperature(F)	Wind_Chill(F)	Humidity(%)	Pressure(in)	Visibility(mi)	Wind_Direction	Wind_Speed(mph)	Precipitation(in)	Weather_Condition
230	3	2016-06-22 23:26:28	2016-06-22 23:56:28	0.0	Fairfield	Solano	94534	55.9	62.825993	80.0	29.96	10.0	WSW	9.2	0.0	Light Rain
240	3	2016-06-23 03:04:28	2016-06-23 04:04:28	0.0	Fairfield	Solano	94534	54.0	62.825993	83.0	29.98	10.0	WSW	5.8	0.0	Light Rain
919	3	2016-06-29 10:47:51	2016-06-29 11:33:44	0.0	Hayward	Alameda	94541	60.1	62.825993	67.0	30.04	10.0	West	11.5	0.0	Mostly Cloudy

Feature Engineering

Feature engineering is one of the most important steps in building a good model, and in our case, it played a major role. The raw dataset only contains details about individual traffic accidents, but to build a predictive model, we needed to extract meaningful patterns and generate new features from that raw information.

We mainly focused on time-related features and accident frequency per location (such as county, city, and zip code). When an accident happens, like the time of day, day of the week, or even the month, can tell us a lot about traffic patterns and accident risk.

From the Start_Time and End_Time columns, we created the following new features:

- Day_of_Week: Weekends may have different accident patterns than weekdays.
- Duration_Minutes: The longer an accident lasts, the more severe it might be.
- Month: Some months might have more accidents due to holidays or weather.
- Is_Holiday: Flag to mark if the accident occurred on a holiday.
- Is_Rush_Hour: Accidents are more likely during busy traffic hours (7–9 AM and 4–7 PM).

We also created a feature called Accident_Density, which counts the number of accidents per county. This helps identify areas with consistently high accident rates.

These engineered features gave our model more context and helped it make better predictions about accident risk.

Dealing with Outliers, Categorical Data, and Downcasting

Outliers

Handling outliers is crucial for building a clean and reliable dataset. Outliers can hurt model performance by reducing accuracy and generalization. In the California accidents dataset, we noticed two types of outliers:

1. Natural outliers — extreme but valid values (ex: very high wind speeds during storms).

2. Data collection errors — invalid or impossible values (ex: wind chills above 140°F).

We removed rows based on clearly invalid values, including:

- Wind Chill > 140°F
- Temperature > 140°F
- Wind Speed > 200 mph

We applied similar filters to features like pressure, distance, duration, and visibility to clean up the data without discarding meaningful edge cases.

Figure 5. Before Dealing with outliers

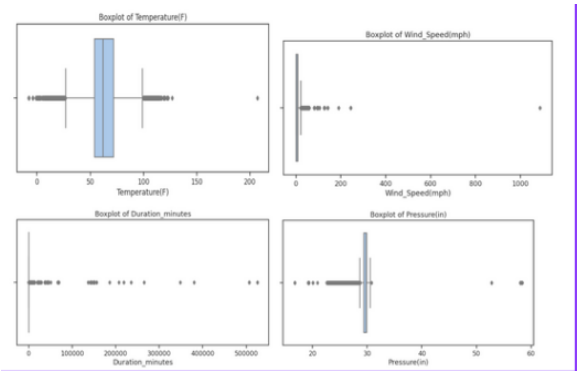
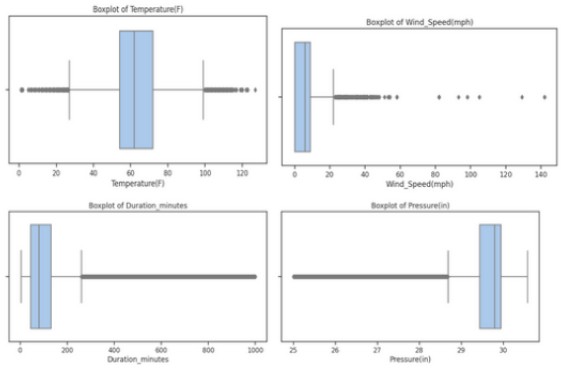


Figure 6. After Dealing with outliers



For categorical data, we needed to convert all non-numeric columns into numbers. Some columns had extremely high cardinality, which adds noise and slows down training. To address this:

- **City** values were reduced from over 1,200 to just 13 by keeping cities with over 10,000 records and grouping the rest as ‘’Other’’.
- **Weather_Condition** was trimmed from around 1,000 unique entries to 16 using a similar method.
- **Zip Code** initially had over 100,000 unique entries. We shrinked them to the first 5 digits, then grouped by the first 3 digits, ending up with 57 unique zip groups.

After reducing category counts, we used **one-hot encoding** to convert the cleaned categorical data into numerical form. This method was preferred over label encoding since, clustering

algorithms perform well one-hot encoded columns. After encoding, the dataset had 957,018 rows and 122 columns.

To speed up training, we applied **downcasting** to numeric columns, reducing memory usage by 14.6%. Finally, we standardized all features, scaling them so they have a mean of 0 and a standard deviation of 1. This step is crucial for clustering algorithms, which use distance as a core part of grouping data. Without proper scaling, clusters can become heavily biased toward features with larger numeric values.

Clustering/ Target Column

Initially, we planned to use the Severity column as the target for our classification task. However, during exploratory data analysis, we found that this column was highly imbalanced with over 75% of the records labeled as Severity 2. This imbalance made it unsuitable for training a well-generalized and reliable model.

Figure 6. Severity Class Balance

Severity	Count
2	873,408
3	91,955
1	8,346
4	6,416

We considered using sampling techniques to address this issue:

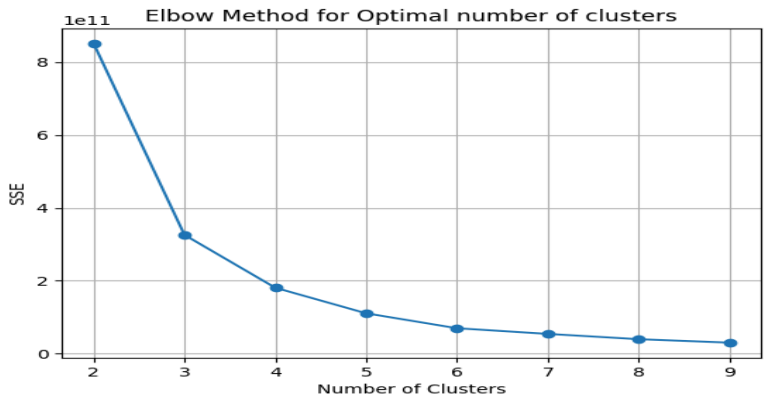
- Undersampling significantly reduced the dataset size, resulting in loss of valuable data.
- Oversampling introduced too much synthetic data, increasing the risk of overfitting.

Neither option was for building an accurate and generalized classifier.

After further research, we realized that clustering would be a more effective approach. Instead of relying on an imbalanced target, we could use unsupervised learning to find natural groupings within the data. And these natural groupings can be used to classify risk levels.

To determine the optimal number of clusters, we applied the elbow method(The point the curve bends indicates the optimal number of clusters), which showed that three clusters would be ideal.(Figure 7) We then trained a KMeans model using all columns except Severity to avoid injecting bias into the clustering process.

Figure 7. Elbow Method



Once clustering was complete, we evaluated the quality of the clusters using the silhouette score. A silhouette score above 0.5 is generally considered good and we achieved a score of 0.644, indicating well-separated and meaningful clusters.

To interpret and label the clusters, we calculated the mean values of Severity, Duration_minutes, and Accident_Density for each cluster. Based on these averages, we assigned the following risk levels:

- Cluster 1 → High Risk
- Cluster 2 → Medium Risk
- Cluster 0 → Low Risk

These cluster labels were then used as the new target column for training our classification models.

Data Sampling

After performing clustering, we noticed that the resulting clusters were not perfectly balanced, some clusters had significantly more samples than others(Figure 8). To address this imbalance,

we decided to use oversampling techniques, since we wanted to preserve as much of the original data as possible(Figure 9).

Undersampling would have reduced the dataset size and probably discarded useful information, so we chose to try oversampling instead.

We initially tested SMOTE (Synthetic Minority Oversampling Technique). It worked well and produced strong model performance during testing, so we did not test ADASYN, as SMOTE was already performing well.

Figure 8. Before Sampling

Cluster	Count	Risk Level
1	220,900	High Risk
2	386,700	Medium Risk
0	349,418	Low Risk

Figure 9. After Sampling

Cluster	Count	Risk Level
1	386,700	High Risk
2	386,700	Medium Risk
0	386,700	Low Risk

Feature Selection

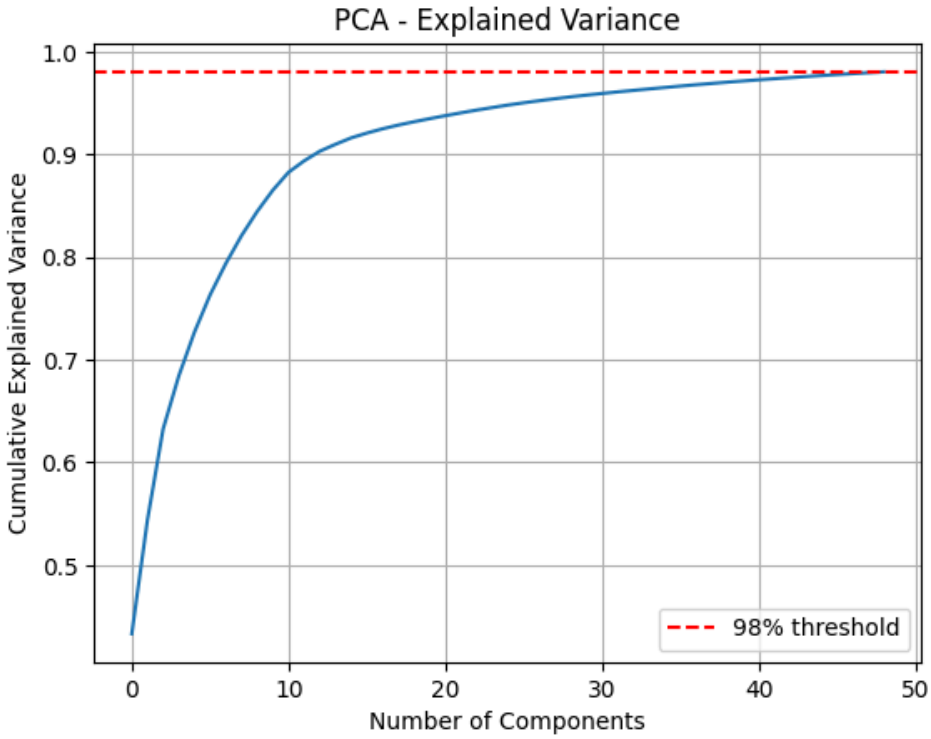
The final dataset had 122 columns after preprocessing and one-hot encoding. Training on all features could lead to issues like slow training, overfitting, and poor accuracy. To address this, we used a combination of techniques to select the most relevant features.

We explored several methods, including:

- Correlation heatmaps
- Intuition-based filtering
- Wrapper methods
- Principal Component Analysis (PCA)

In the early stages, we applied intuition-based filtering during the Bash cleaning to drop clearly irrelevant columns. Later, we used PCA to reduce dimensionality while preserving important patterns in the data. After testing various values, we chose to keep 98% of the variance, which gave us 48 principal components(Figure 10).

Figure 10. Principal Component Analysis



Although PCA improved performance and reduced dimensionality, it replaced original feature names with component numbers, making analysing more difficult. However, the benefits in training speed and model quality made it a good trade-off

Supervised ML Models

After preparing the data and creating target column through clustering, we trained supervised models to predict the accident risk level. The machine learning algorithms we tested were Logistic Regression, Support Vector Machine (SVM), and Random Forest.

Originally, we considered using a Multilayer Perceptron (MLP), but due to the large size of our dataset and computational limitations, we decided to use Random Forest instead for its speed.

Parameter selection for each model:

- Logistic Regression: We tuned the model using different values for the regularization parameter C, which controls the penalty for overfitting. We tested values like 0.01, 0.1, and 1 to find C value..
- Support Vector Machine (SVM): Similar to logistic regression, we experimented with various C values. Also, we tested different kernel types, including 'rbf', 'linear', and 'polynomial', to see which could capture non-linear patterns in the data best.
- Random Forest: For this model, we tuned two main hyperparameters:
    - n_estimators: The number of decision trees in the forest. We tested 10, 20, and 100.
    - max_depth: The maximum depth of each tree, which controls how much the model can learn from the data without overfitting.. We tested 4, 8, and 15.

Implementation

After selecting the models and defining which hyperparameters to test, we used GridSearchCV with 5-fold cross-validation to train and evaluate each machine learning algorithm. This helped make a fair comparison across all models.

For each algorithm, we reviewed the classification report, including precision, recall, and F1-score, and also compared the train and test log loss values. These metrics helped us understand the good and bad of each model, and assess whether any were overfitting or underfitting. Based on these results, we selected the most optimal model for final training and evaluation.

To implement and evaluate our models, we used the following tools from the scikit-learn library:

- train_test_split – to split the dataset into training and testing sets (80/20 split).
- GridSearchCV – for tuning hyperparameters using cross-validation.
- classification_report – to analyze metrics like precision, recall, and F1-score.
- log_loss – to evaluate how well each model predicted probabilities.

Experimental Setup

Our final dataset had over 950,000 rows, which made it difficult to run GridSearchCV directly due to limited hardware and long training times. To work around this, we created a smaller sample dataset with about 50,000 rows. We used this smaller set to train models, tune hyperparameters, and run cross-validation. This helped us find the best model setup without wasting too much time or resources.

After choosing the best model and its settings, we trained the final model on the full dataset using a Google Cloud Platform (GCP) virtual machine. This gave us the extra computing power needed to handle the full training process smoothly.

Pipeline Flow:

Data Cleaning(Bash) → Data Cleaning(Python) → Feature Engineering →Optimizations → PCA → KMeans Clustering → Sampling → Supervised Classification

Results

Figure. 11: Classification Report(Test) SVM

Class	Precision	Recall	F1-Score
0	0.96	0.92	0.94
1	0.98	0.96	0.97
2	0.94	1	0.97
Overall Accuracy	—	—	0.96

Figure. 12: Classification Report(Test) Random Forest

Class	Precision	Recall	F1-Score
0	1	0.93	0.96
1	0.98	1	0.99
2	0.95	1	0.98
Overall Accuracy	—	—	0.98

Figure. 13: Classification Report(Test) Logistic Regression

Class	Precision	Recall	F1-Score
0	1	0.94	0.97
1	0.98	1	0.99
2	0.96	1	0.98
Overall Accuracy	—	—	0.98

Figure. 14: Best Parameters

Model	Best Score	Best Parameters
svm	0.964125	{'C': 1}
randomForest	0.979375	{'max_depth': 15, 'n_estimators': 50}
logistic_regression	0.981625	{'C': 1}

We evaluated all three models: Logistic Regression, Support Vector Machine (SVM), and Random Forest using GridSearchCV and analyzed their classification reports, train/test losses, and parameter performance.

Even though both Random Forest and Logistic Regression showed similar accuracy, precision, and recall, Logistic Regression had better training and test loss, indicating more stable and confident predictions.

Random Forest(max_dept: 15, n_estimator: 50):

- Train Loss: 0.1005
- Test Loss: 0.1244

Logistic Regression (C = 1):

- Train Loss: 0.0426

- Test Loss: 0.0526

Based on these results, we selected Logistic Regression with C = 1 as the final model.

Discussion

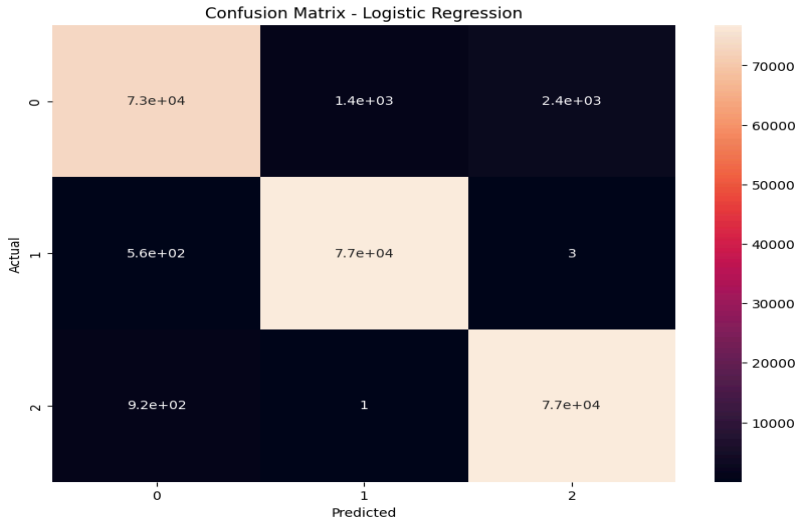
We retrained the selected model using the full dataset, applying the same number of PCA components and maintaining a consistent train-test split. The model's performance was tested through cross-validation.

- **Test Accuracy:** 0.98
- **Train Accuracy:** 0.98

Figure. 15: Classification Report - Final Model(Test Set)

Class	Precision	Recall	F1-Score
0	0.98	0.95	0.97
1	0.98	0.99	0.99
2	0.97	0.99	0.98
Overall Accuracy	—	—	0.98

Figure. 16: Confusion Matrix - Final Model



Interpretation

- **Precision** reflects how often the model's positive predictions were correct.
- **Recall** measures how well the model identified all values of each class.
- **Confusion Matrix:** Diagonal values (light color) show correct predictions; off-diagonal values represent misclassifications.

Overall, the final model performed exceptionally well across all key metrics. High precision, recall, and F1-score demonstrate that it is both accurate and well-balanced. The near-identical training and test accuracy confirms that the model is not overfitting and generalizes well. Additionally, the low and nearly equal train and test loss values support the model's reliability.

While the supervised classification aspect of the project performed impressively, the clustering step still has room for improvement. Although our silhouette score of 0.68 is decent, better clustering techniques or feature tuning could potentially lead to more meaningful clusters.

Conclusion

In this project, we built TripSafe, a machine learning model that predicts traffic accident risk based on past accident data. We started with a huge dataset and narrowed it down to focus on California, since it had the most data points. This helped us build a cleaner and more accurate model.

We went through multiple steps,cleaning data with Bash and Python, creating useful features, reducing dimensions with PCA, and generating target columns using clustering. We tried different models and, after testing, found that Logistic Regression gave the best results. It was accurate, balanced, and had very low loss compared to the others.

The supervised part of the project worked really well, and the final model performed great across all key metrics. However, there's still room to improve the clustering part, especially if we want better separation between risk levels.

Overall, TripSafe shows how machine learning can be used to turn raw data into something useful and practical. With more improvements, it has the potential to help drivers, and navigation apps, make smarter, safer decisions.

References

- [Codebasics]. (2019, February 4). Machine Learning Tutorial Python - 13: K Means Clustering Algorithm [Video]. Youtube. https://www.youtube.com/watch?v=EItlUEPCIzM&t=375s.
- [Codebasics]. (2021, September 9). Principal Component Analysis (PCA) with Python Code [Video]. Youtube. https://www.youtube.com/watch?v=EItlUEPCIzM&t=375s.
- [Codebasics]. (29, November 29). Hyper parameter Tuning (GridSearchCV) [Video]. Youtube. https://www.youtube.com/watch?v=HdlDYng8g9s&t=574s.
- Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
- Will Cukierski. Titanic - Machine Learning from Disaster. https://kaggle.com/competitions/titanic, 2012. Kaggle.
- Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, https://doi.org/10.21105/joss.03021.
- Karlin, Marc. “California Car Accident Statistics - Traffic Fatalities ca 2025.” Karlin & Karlin, APC, 9 Jan. 2025, www.karlaw.com/blog/california-car-accident-statistics/#:~:text=At%20least%20250%2C000%20accidents%20in,car%20accidents%20are%20speeding%2Drelated.
- Girija, M., & Divya, V. (2024). Deep learning-based traffic accident prediction: An investigative study for enhanced road safety. *EAI Endorsed Transactions on Internet of Things, 10*(1). https://doi.org/10.4108/eetiot.51661
- Jaadi, Z. (2024, February 23). *Principal Component Analysis (PCA): A step-by-step explanation*. Built In. https://builtin.com/data-science/step-step-explanation-principal-component-analysis