BDM 1034 - Application Design for Big Data

Report

STUDENT'S NAME	STUDENT'S ID
Andres Santa	C0931978
Johan Rodriguez	C0931102
Amandeep Singh	C0937432
Amita	C0935607
Manjot Kaur	C0938094
Anju Bala	C0935847
Sarita Rani	C0938516
Brian Martinez	C0940439

Table of contents

GitHub link:

https://github.com/Andressanta09/Project_Aplication_Designe

Project Board: **https://github.com/users/Andressanta09/projects/1**

---

1. Introduction

Objective:

- This project integrates real-time and static data sources for comprehensive analysis and visualization.
- It combines weather API data and accident history to predict accident severity.

Scope:

- The integration of real-time data with static datasets enables actionable insights into traffic safety and
- weather-related impacts. The interactive visualizations and predictions aim to assist decision-making.

Dataset Sources:

- - Real-Time Data: OpenWeatherMap API for live weather conditions.
- - Static Data: US Accidents dataset from Kaggle, containing accident records from 2016 to 2021.

---

2. Data Pipeline Architecture

Overview:

The architecture integrates real-time API data with historical accident data. Key stages include:

- Real-time data retrieval from OpenWeatherMap API.

- Cloud storage for processed data.

- Preprocessing, modeling, and real-time prediction pipelines.

- Deployment using Streamlit for UI integration.

---

3. Data Collection

Real-Time Data:

API: OpenWeatherMap API

URL: https://api.openweathermap.org/data/2.5/weather

Parameters: city, latitude, longitude, API key

Example Raw Data:

{

"coord": {"lon": -123.26, "lat": 44.56},

"weather": [{"description": "light rain", "main": "Rain"}],

"main": {"temp": 290.15, "pressure": 1013, "humidity": 80},

"visibility": 10000,

"wind": {"speed": 4.12, "deg": 120}

}

Static Data:

Dataset: US Accidents dataset from Kaggle.

Size: 1.5 million records.

Time Range: 2016-2021.

Preprocessing:

- Removed irrelevant columns.

- Handled missing values with statistical imputation.

- Normalized numerical features.

---

4. Data Preprocessing and Engineering

Steps:

1. Cleaning: Removed duplicates and handled missing values using SimpleImputer.

2. Transformation: Encoded categorical columns using LabelEncoder and normalized numerical features

using StandardScaler.

3. Feature Engineering: Derived accident severity levels based on visibility and wind speed thresholds.

Libraries/Tools:

Pandas, NumPy, PySpark, Scikit-learn.

Sample Code:

```
from sklearn.preprocessing import StandardScaler, LabelEncoder

scaler = StandardScaler()

data['scaled_temp'] = scaler.fit_transform(data[['temp']])

le = LabelEncoder()

data['city_encoded'] = le.fit_transform(data['city'])
```

---

5. Modeling

Model Selection Explanation:

Purpose

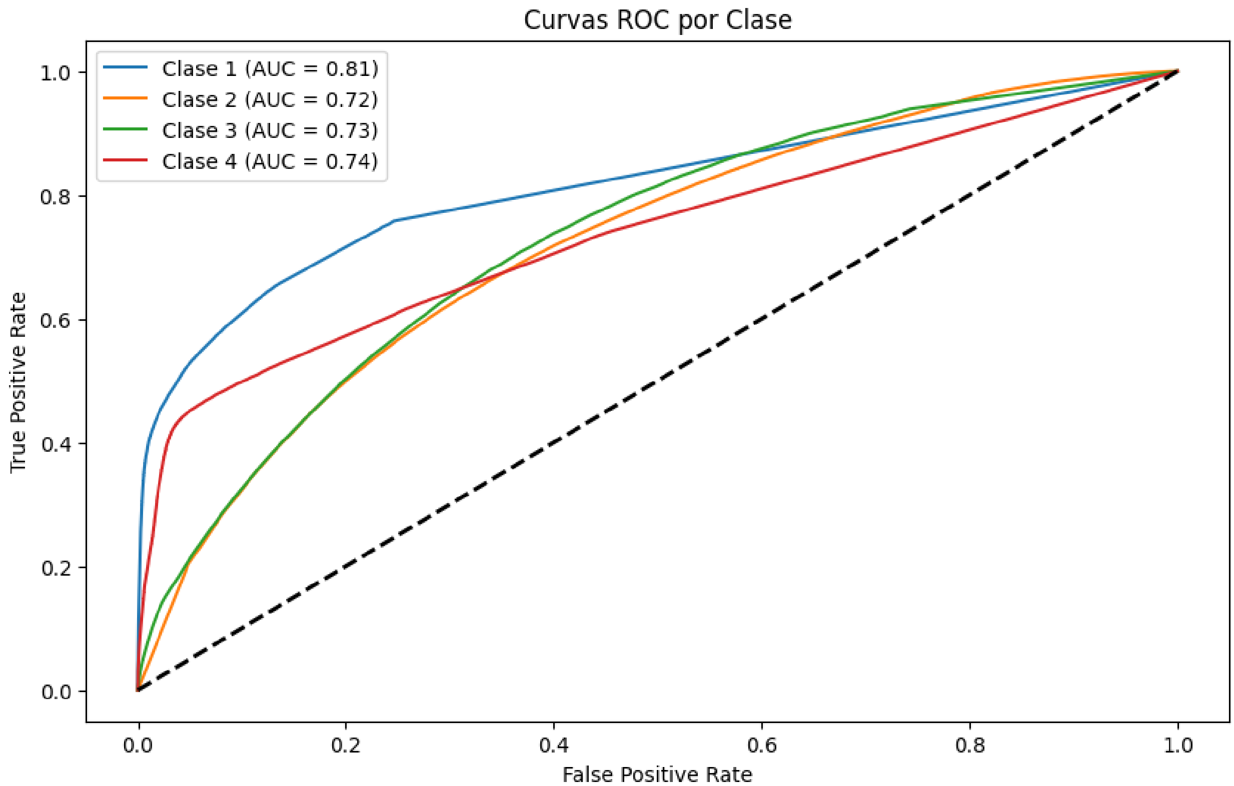
The project evaluated multiple machine learning models to predict accident severity effectively. The selection process aimed to balance accuracy, interpretability, and computational efficiency.

Chosen Models

1. Random Forest Classifier:

• Why Chosen: Random Forest is robust, interpretable, and handles feature importance well. It's ideal for datasets with mixed types of features.

- **Accuracy:** 84.5%
- **Explanation:**
    - The ensemble nature of Random Forest reduces overfitting compared to individual decision trees.
    - Strong performance is seen in predicting dominant classes (e.g., Class 2), but the model struggles with minority classes, leading to a low macro-average F1-score.
    - Feature importance suggests `Pressure(in)` and `Humidity(%)` are key predictors, likely because these weather conditions significantly impact severity.



• Best For: Baseline predictions with insights into feature importance.

2. Gradient Boosting:

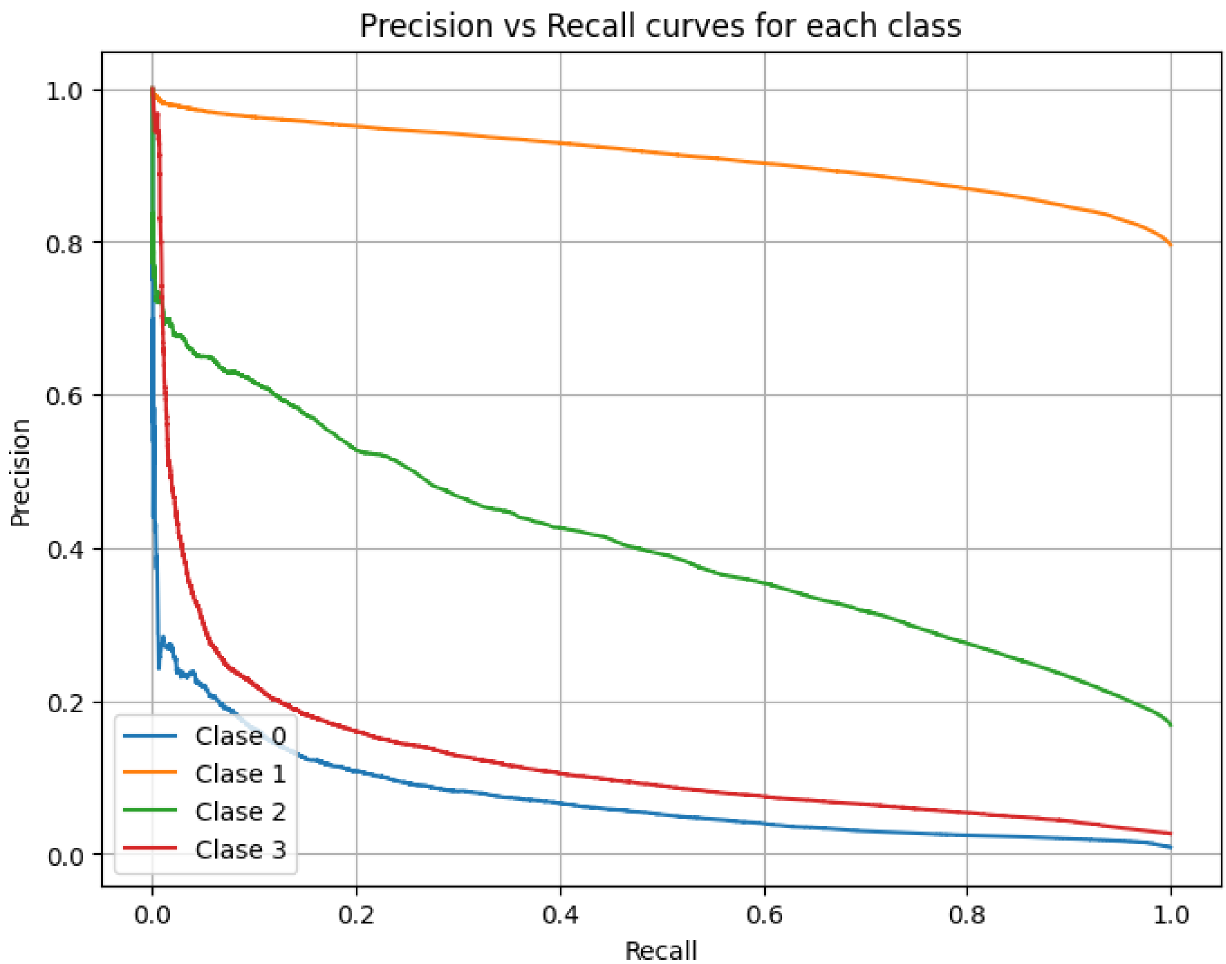
• Why Chosen: Gradient Boosting excels in handling complex feature interactions and provides higher accuracy than Random Forest with fine-tuned hyperparameters.

• Performance:

• Accuracy: 0.80140831311029

Explanation:

- XGBoost optimizes performance by correcting errors made by previous iterations, making it robust for both linear and non-linear patterns.
- High interpretability of feature importance highlights `Humidity(%)` and `Temperature(F)` as critical contributors.



• Best For: High-accuracy predictions where overfitting is controlled.

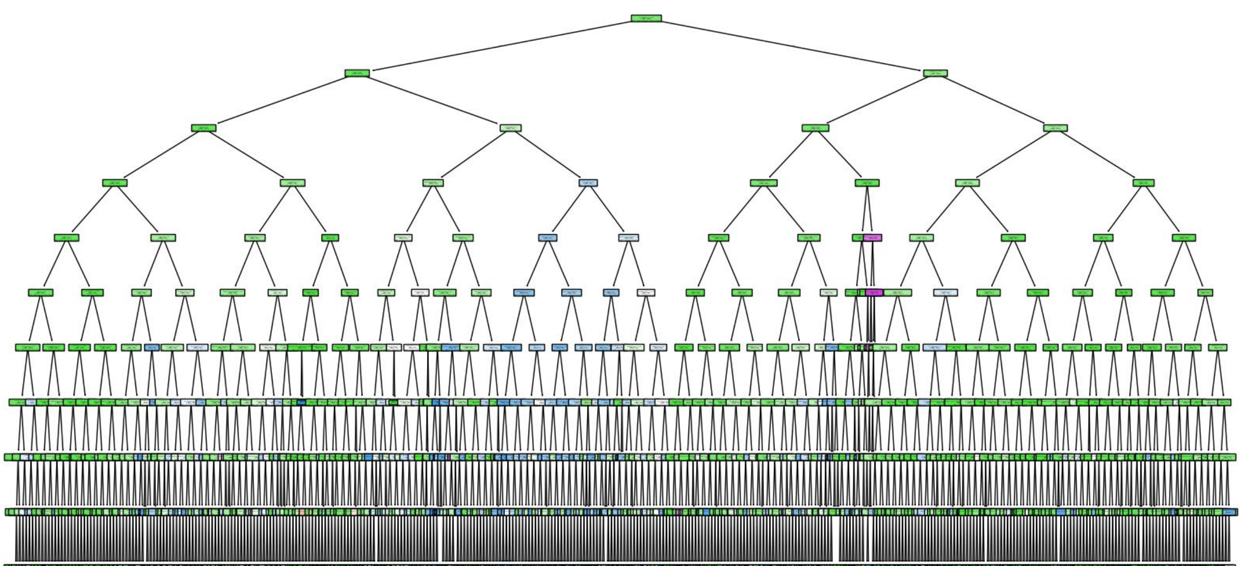
Decision tree Classifier:

The **Decision Tree Classifier** achieved an accuracy of **80.02%**. Here's the explanation for this result:

Performance Insights

- **Accuracy (80.02%):**
  - Moderate accuracy indicates that the Decision Tree is fitting the dataset to some extent but may not generalize well to unseen data.
  - Decision Trees tend to overfit the training data, especially when the tree depth is not constrained.
- **Strengths:**
  - Simple and interpretable model.
  - Captures non-linear relationships between features and target variables.
- **Weaknesses:**
  - Prone to overfitting, especially in datasets with high feature variance or noise.
  - Struggles with class imbalance, likely leading to biased predictions favoring dominant classes.

Decision Tree



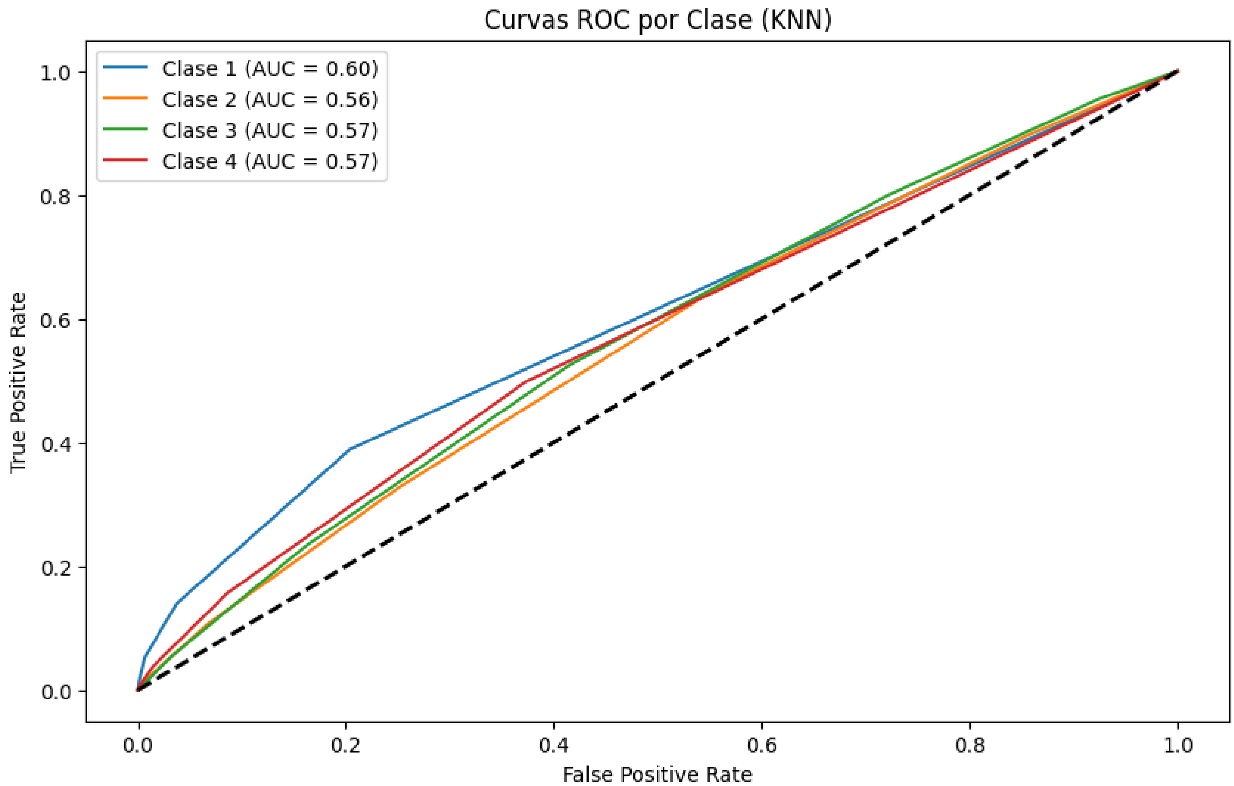
K-Nearest Neighbors (KNN)

Overall Metrics:

- **Accuracy: 46%** (not ideal, close to random guessing for imbalanced classes).
- **Macro Average:** (Average across classes):
    - Precision: **34%**, Recall: **30%**, F1-score: **30%**.
    - Indicates poor generalization across all classes.
- **Weighted Average:** (Weighted by class sizes):
    - Precision: **44%**, Recall: **46%**, F1-score: **44%**.
    - Skewed towards better performance on majority classes (2 and 3).

Analysis

1. **Class Imbalance:**
    a. Classes 1 and 4 are underrepresented in the dataset, leading to poor recall and F1-scores for these classes.
2. **KNN Limitations:**
    a. KNN is sensitive to imbalanced datasets, as it relies on majority voting in the neighborhood.
    b. Performance depends heavily on the choice of k (number of neighbors) and feature scaling.



6. Deployment

Cloud Hosting:

The app was deployed on Streamlit Cloud, enabling public access and demonstrating the scalability of the solution.

Real-Time Predictions:

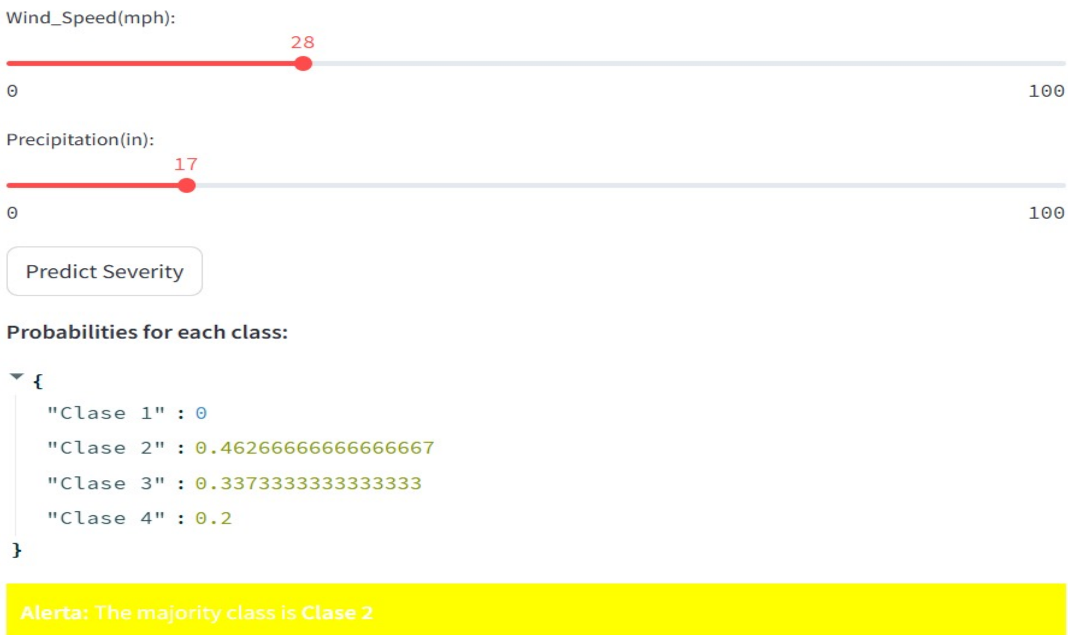
- Weather data fetched in real-time is passed to trained models for severity predictions.

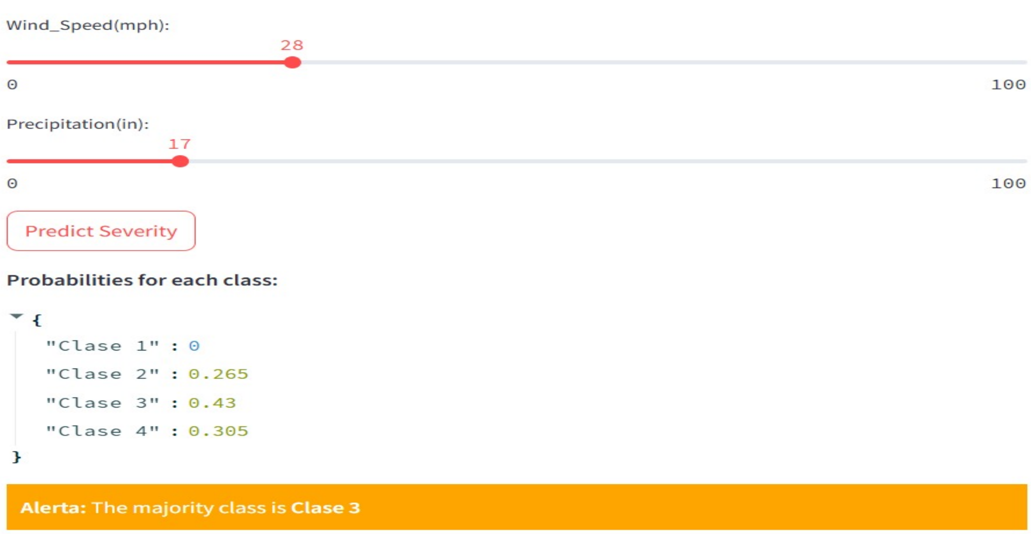
Integration:

- Streamlit app connects to the Flask API for visualizations and predictions.

1. Deployment Overview :The application was deployed online using Streamlit Cloud, a platform designed for hosting Python-based web applications. This ensured the

application was accessible via a public URL, allowing seamless interaction with the predictive models developed during the project.





Key Features

1. Prediction Functionality:

• Users input features (e.g., Wind Speed and Precipitation) using sliders and dropdown menus.

• Upon clicking the “Predict Severity” button, the app uses the trained Random Forest Classifier to predict the severity of traffic accidents.

2. Output Probabilities:

• The app provides class probabilities for each severity level, ensuring transparency in predictions.

• **Example probabilities:**

• Class 1: 0.265

• Class 2: 0.243

• Class 3: 0.397 (majority class, highlighted)

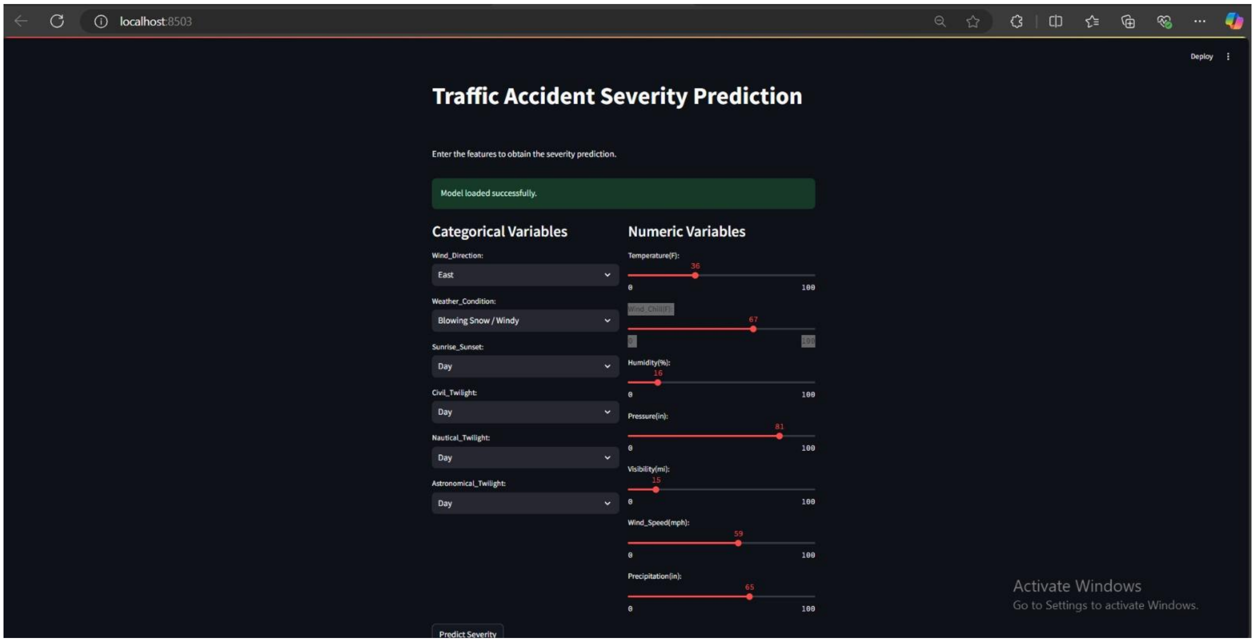
• Class 4: 0.095

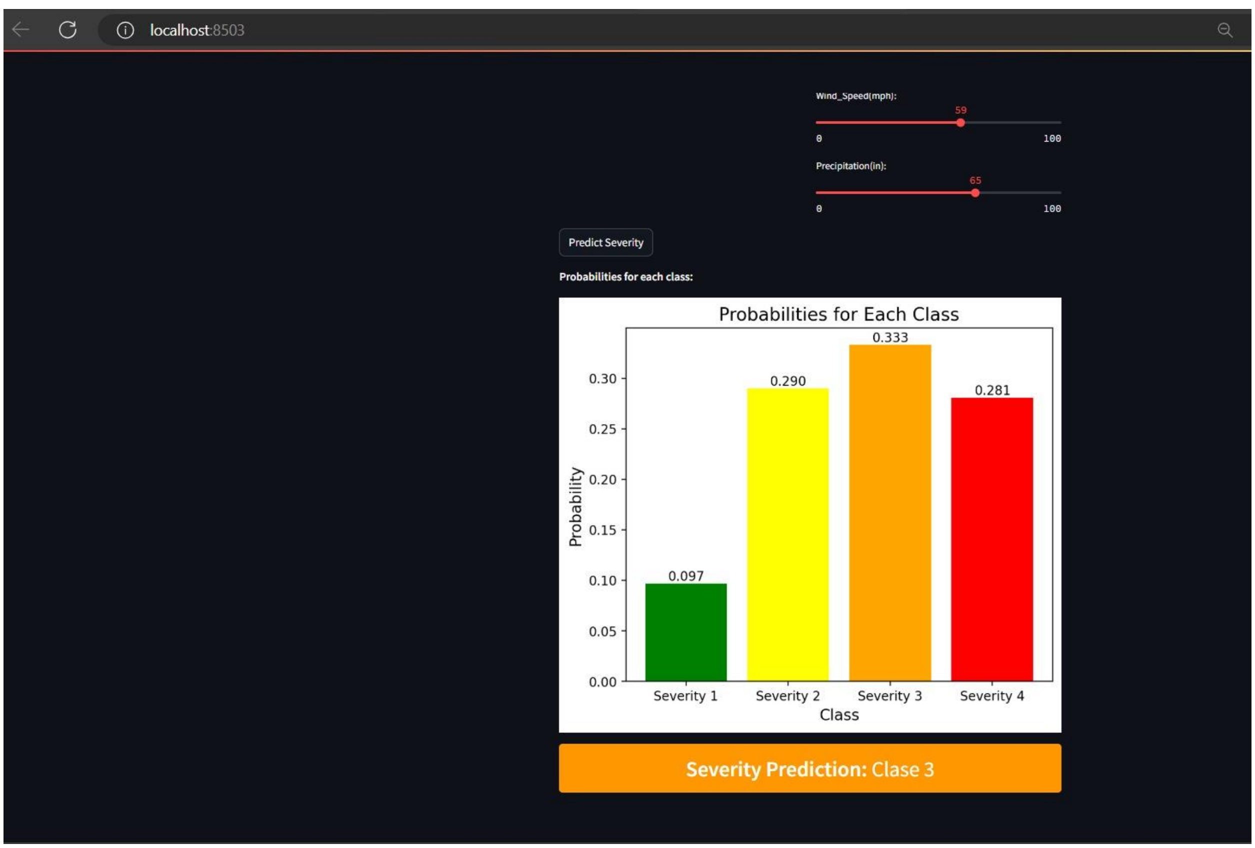
Key Benefits of Deployment

• Accessibility: The online deployment eliminates the need for users to set up local environments, making the app easily accessible on any device with an internet connection.
• Usability: The intuitive interface ensures that users with minimal technical expertise can interact with the application.

---

7. Visualization and UI





The UI is designed to prioritize user interaction and visualization:

• Features are organized into categorical and numerical sections.

• Probabilities for each class are displayed in a graphical format.

• Alerts highlight critical information, such as the predicted severity class.

8. challenges and learnings:

Challenges:

- Managing API rate limits and handling missing data in real-time streams.

- Optimizing cloud deployment for low-latency predictions.

Learnings:

- Real-time data integration significantly enhances analysis.

- Scalable architecture is crucial for handling high-frequency data streams.

---

9. Conclusion

This project successfully integrated real-time and static data for predictive analysis and visualization.

Future enhancements include adding traffic and demographic data, improving model interpretability,

and optimizing deployment for scalability.

---

10. References

1. OpenWeatherMap API documentation.

2. US Accidents dataset on Kaggle: https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents/data

3. Libraries: Pandas, NumPy, Scikit-lea