ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

MACHINE LEARNING (CO3117)

Báo cáo Bài tập lớn

Phân tích và Đánh giá Mô hình Dự đoán Rủi ro Tai nạn Giao thông

---

Giảng viên hướng dẫn: Võ Thanh Hùng
Lớp: L01

Danh sách thành viên: Phạm Minh Tuấn - 2115186
Nguyễn Thanh Tân - 2213061
Bùi Tiến An - 2112722
Đồng Minh Thiện - 2110555

TP Hồ Chí Minh, ngày 28 tháng 04 năm 2025

Mục lục

Danh sách hình vẽ

Danh sách bảng

Danh sách thành viên và mức độ đóng góp

STT	Họ và tên	MSSV	Nhiệm vụ	Mức độ đóng góp
1	Phạm Minh Tuấn	2115186	Phân tích dữ liệu	100%
2	Nguyễn Thanh Tân	2213061	Model	100%
3	Bùi Tiến An	2112722	Báo cáo	100%
4	Đồng Minh Thiện	2110555	Slide	100%

1 Giới thiệu đề tài

Trong khuôn khổ bài tập lớn môn học **Machine Learning**, nhóm chúng tôi thực hiện dự án với mục tiêu xây dựng mô hình dự đoán mức độ nghiêm trọng của các vụ tai nạn giao thông tại Hoa Kỳ, dựa trên bộ dữ liệu **US-Accidents** – một trong những tập dữ liệu lớn và toàn diện nhất về tai nạn giao thông thực tế.

Tai nạn giao thông từ lâu đã là một vấn đề nhức nhối, gây ảnh hưởng nghiêm trọng đến an toàn xã hội và nền kinh tế quốc gia. Không chỉ ở Hoa Kỳ, mà tại Việt Nam, mỗi năm cũng ghi nhận hàng chục nghìn vụ tai nạn, để lại hậu quả nặng nề về người và tài sản. Thực tế này cho thấy, việc tận dụng các kỹ thuật học máy để dự đoán và phân tích tai nạn là hướng nghiên cứu có ý nghĩa thiết thực, không chỉ đối với các nước phát triển mà còn đối với Việt Nam.

Với hơn 7.7 triệu bản ghi được thu thập từ năm 2016 đến 2023, bộ dữ liệu **US-Accidents** cung cấp thông tin chi tiết về các yếu tố liên quan như điều kiện thời tiết, vị trí địa lý, thời gian, loại đường, mức độ tắc nghẽn giao thông, và nhiều yếu tố khác.

Bài toán mà nhóm hướng tới là dự đoán liệu một vụ tai nạn xảy ra có nghiêm trọng hay không, dựa trên các thông tin thu thập được tại thời điểm xảy ra sự cố. Đây là một bài toán phân loại nhị phân (*binary classification*), có tính ứng dụng thực tiễn cao, đặc biệt trong việc hỗ trợ các cơ quan chức năng cảnh báo nguy cơ, tối ưu hóa phản ứng cứu hộ, và nghiên cứu giải pháp giảm thiểu thiệt hại giao thông. Nếu được áp dụng tại Việt Nam, các hệ thống như vậy có thể góp phần giảm thiểu tình trạng ùn tắc và thương vong trong các vụ tai nạn trên các tuyến đường trọng điểm.

Để giải quyết bài toán, nhóm tiến hành:

- Khám phá và tiền xử lý dữ liệu, bao gồm làm sạch dữ liệu, xử lý *missing values*, mã hóa biến phân loại (*categorical variables*), và chuẩn hóa dữ liệu.
- Lựa chọn đặc trưng (*feature selection*) nhằm giữ lại các yếu tố có ảnh hưởng lớn nhất đến mức độ nghiêm trọng của tai nạn.
- Áp dụng và thực nghiệm nhiều thuật toán Machine Learning khác nhau, tiêu biểu như *Logistic Regression*, *Random Forest* và *XGBoost*.
- Đánh giá mô hình thông qua các chỉ số như *Accuracy*, *Precision*, *Recall*, *F1-Score* và *ROC-AUC*, đồng thời phân tích độ phức tạp mô hình và khả năng tổng quát hóa.

Thông qua dự án này, bên cạnh việc đáp ứng yêu cầu kỹ thuật, nhóm cũng rèn luyện được khả năng tư duy phản biện, sáng tạo, và giải quyết các bài toán thực tế trong lĩnh vực học máy – những kỹ năng quan trọng đối với sinh viên ngành Công nghệ Thông tin, đặc biệt trong bối cảnh Việt Nam đang đẩy mạnh chuyển đổi số và ứng dụng trí tuệ nhân tạo vào các lĩnh vực đời sống.

2 Xác định yêu cầu bài toán

2.1 Mục tiêu bài toán

Mục tiêu của bài toán là xây dựng một mô hình học máy có khả năng dự đoán mức độ nghiêm trọng của các vụ tai nạn giao thông tại Hoa Kỳ, dựa trên các thông tin được ghi nhận tại thời điểm xảy ra tai nạn.

Cụ thể, bài toán yêu cầu phân loại mỗi vụ tai nạn thành hai nhóm:

- **Nghiêm trọng**: các vụ tai nạn có mức độ ảnh hưởng cao, tiềm ẩn nguy cơ gây tổn thất lớn về người và tài sản.
- **Không nghiêm trọng**: các vụ tai nạn có mức độ ảnh hưởng thấp, ít gây thiệt hại nghiêm trọng.

2.2 Dữ liệu đầu vào và đầu ra

- **Dữ liệu đầu vào**: Các thuộc tính mô tả đặc điểm của vụ tai nạn như vị trí địa lý (kinh độ, vĩ độ), thời gian xảy ra, điều kiện thời tiết, tình trạng giao thông, loại đường, mức độ tắc nghẽn, và các thông tin liên quan khác.
- **Dữ liệu đầu ra**: Nhãn (*label*) dự đoán cho mỗi vụ tai nạn, thể hiện mức độ nghiêm trọng (*serious* hoặc *not serious*).

2.3 Phân loại bài toán

Dựa trên yêu cầu đầu ra là phân chia dữ liệu thành hai nhóm rời rạc, bài toán được xác định là bài toán **phân loại nhị phân** (*binary classification*).

2.4 Đánh giá hiệu quả mô hình

Để đánh giá hiệu quả của mô hình, nhóm sử dụng các chỉ số phổ biến trong các bài toán phân loại, bao gồm:

- **Accuracy**: Tỷ lệ dự đoán đúng trên tổng số dự đoán.
- **Precision**: Độ chính xác của mô hình đối với các dự đoán thuộc nhóm nghiêm trọng.
- **Recall**: Khả năng mô hình tìm ra đúng các vụ tai nạn nghiêm trọng.
- **F1-Score**: Trung bình điều hòa giữa Precision và Recall, đặc biệt hữu ích khi dữ liệu mất cân bằng.

Ngoài ra, nhóm cũng quan tâm đến khả năng tổng quát hóa (generalization) của mô hình trên các tập dữ liệu chưa từng thấy, nhằm đảm bảo tính ứng dụng thực tiễn.

3 Các công trình nghiên cứu liên quan

Trong phần này, chúng tôi tổng hợp và trình bày các công trình nghiên cứu có liên quan đến bài toán dự đoán mức độ nghiêm trọng của tai nạn giao thông, đặc biệt là nghiên cứu áp dụng các thuật toán học máy vào bộ dữ liệu US-Accidents. Các công trình này sẽ cung cấp cái nhìn sâu sắc về các phương pháp và kết quả đạt được trong việc phân tích dữ liệu tai nạn giao thông.

3.1 Accident Risk Prediction based on Heterogeneous Sparse Data: New Dataset and Insights

Tác giả: Sobhan Moosavi, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, Radu Teodorescu, Rajiv Ramnath
Hội nghị: ACM SIGSPATIAL 2019
Liên kết: https://arxiv.org/abs/1909.09638
Tóm tắt: Nghiên cứu này giới thiệu mô hình DAP (Deep Accident Prediction), một mô hình học sâu được thiết kế để dự đoán nguy cơ tai nạn giao thông dựa trên dữ liệu không đồng nhất và thưa thớt, bao gồm thông tin thời tiết, sự kiện giao thông, và các điểm quan tâm (POI). Các tác giả đã thu thập một tập dữ liệu mới với hơn 100 triệu điểm dữ liệu về tai nạn, thời tiết và các yếu tố giao thông trong khu vực thành phố. Kết quả nghiên cứu cho thấy mô hình DAP vượt trội so với các phương pháp truyền thống như Random Forest và SVM trong việc dự đoán các sự kiện tai nạn hiếm gặp. Mô hình này cho phép dự đoán tai nạn một cách chính xác hơn trong các tình huống thực tế, đồng thời có khả năng cung cấp cảnh báo sớm cho các cơ quan chức năng. Mô hình DAP cũng chứng minh rằng dữ liệu không đồng nhất, mặc dù khó xử lý, vẫn có thể mang lại kết quả đáng tin cậy nếu được kết hợp đúng cách.

3.2 Predicting Accident Severity: An Analysis of Factors Affecting Accident Severity Using Random Forest Model

Tác giả: Adekunle Adefabi, Somtobe Olisah, Callistus Obunadike, Oluwatosin Oyetubo, Esther Taiwo, Edward Tella
Năm: 2023
Liên kết: https://arxiv.org/abs/2310.05840
Tóm tắt: Nghiên cứu này tập trung vào việc sử dụng mô hình Random Forest để phân tích và dự đoán mức độ nghiêm trọng của các vụ tai nạn giao thông, sử dụng dữ liệu thu thập từ một khu vực đô thị lớn. Dữ liệu được phân tích bao gồm các yếu tố như điều kiện thời tiết, tốc độ gió, độ ẩm, ánh sáng và điều kiện mặt đường. Các tác giả đã thực hiện đánh giá hiệu quả của mô hình Random Forest trong việc phân loại mức độ nghiêm trọng của tai nạn và phát hiện rằng mô hình này có độ chính xác vượt trội so với các mô hình đơn giản như Logistic Regression. Kết quả cho thấy rằng các yếu tố thời tiết và điều kiện mặt đường đóng vai trò quan trọng nhất trong việc xác định mức độ nghiêm trọng của các vụ tai nạn. Mô hình Random Forest đã đạt độ chính xác lên đến 82

3.3 Road Accident Severity Classification Using US Accidents Dataset

Tác giả: Abdirashid Dahir
Liên kết: https://abdirashiddahir.github.io/files/ML_report.pdf
Tóm tắt: Nghiên cứu này sử dụng bộ dữ liệu US-Accidents để thực hiện phân loại mức độ nghiêm trọng của tai nạn giao thông. Dữ liệu được lấy từ hơn 7 triệu bản ghi của các vụ tai nạn xảy ra tại Hoa Kỳ, bao gồm các yếu tố như điều kiện thời tiết, ngày giờ, vị trí địa lý, loại

đường và thông tin về các vụ tai nạn. Các mô hình được áp dụng trong nghiên cứu này bao gồm Logistic Regression, Random Forest và Support Vector Machines (SVM). Kết quả nghiên cứu cho thấy rằng Random Forest đạt được hiệu quả cao nhất với độ chính xác lên đến 85%, trong khi Logistic Regression và SVM có độ chính xác thấp hơn. Nghiên cứu chỉ ra rằng các yếu tố ảnh hưởng mạnh mẽ đến mức độ nghiêm trọng của tai nạn bao gồm loại đường, thời gian trong ngày, và các yếu tố thời tiết như mưa và tuyết. Mô hình Random Forest được cho là phù hợp nhất trong việc xử lý dữ liệu phức tạp và có thể ứng dụng rộng rãi trong việc phát triển các hệ thống cảnh báo giao thông tại các quốc gia khác nhau.

4 Tổng quan và Nguyên lý hoạt động các phương pháp học máy

4.1 Logistic Regression

4.1.1 Tổng quan

Logistic Regression là một trong những thuật toán phân loại cơ bản và phổ biến nhất trong học máy. Khác với tên gọi "Regression", mô hình này không dùng cho bài toán hồi quy mà chủ yếu phục vụ cho bài toán phân loại nhị phân (binary classification). Logistic Regression dựa trên mô hình hồi quy tuyến tính, nhưng sử dụng hàm sigmoid để chuyển đổi đầu ra thành giá trị xác suất từ 0 đến 1, qua đó xác định nhãn phân loại.

4.1.2 Cách hoạt động

1. **Bước 1: Xây dựng mô hình tuyến tính**
   - Mô hình đầu tiên xây dựng một tổ hợp tuyến tính của các đặc trưng:
   $$z = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$
   Trong đó:
   - w_0 là hệ số chặn (bias).
   - w_i là hệ số của đặc trưng x_i.
2. **Bước 2: Áp dụng hàm sigmoid để tính xác suất**
   - Hàm sigmoid được sử dụng để ánh xạ giá trị z thành khoảng $[0, 1]$:
   $$\sigma(z) = \frac{1}{1 + e^{-z}}$$
   - Kết quả $\sigma(z)$ chính là xác suất mà mẫu thuộc về lớp 1.
3. **Bước 3: Xác định nhãn phân loại**
   - Nếu $\sigma(z) > 0.5$, dự đoán mẫu thuộc lớp 1.
   - Nếu $\sigma(z) \leq 0.5$, dự đoán mẫu thuộc lớp 0.
4. **Bước 4: Tối ưu hàm mất mát**
   - Hàm mất mát (Loss function) được sử dụng là hàm log-loss (cross-entropy):
   $$L(w) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})\right]$$
   Trong đó:
   - m là số lượng mẫu.
   - $y^{(i)}$ là nhãn thực tế của mẫu thứ i.
   - $\hat{y}^{(i)}$ là xác suất dự đoán của mẫu thứ i.
   - Các hệ số w được tối ưu hóa bằng thuật toán Gradient Descent hoặc các biến thể của nó để giảm thiểu hàm mất mát.

4.1.3 Đánh giá

- **Ưu điểm:**
  - Dễ hiểu, dễ triển khai và hiệu quả trên các bài toán phân loại nhị phân đơn giản.
  - Có khả năng xuất ra xác suất, hữu ích cho việc xếp hạng mẫu hoặc phân tích độ tin cậy của dự đoán.
  - Ít yêu cầu tài nguyên tính toán.
- **Nhược điểm:**
  - Hiệu quả giảm sút khi tồn tại mối quan hệ phi tuyến tính giữa các đặc trưng và nhãn đầu ra.
  - Dễ bị ảnh hưởng bởi dữ liệu mất cân bằng (imbalanced data).
  - Giả định mối quan hệ tuyến tính giữa đặc trưng và log-odds, điều này không phải lúc nào cũng đúng trong thực tế.

4.2 Naive Bayes

4.2.1 Tổng quan

Naive Bayes là một tập hợp các thuật toán phân loại đơn giản dựa trên định lý Bayes với giả định mạnh mẽ rằng các đặc trưng (features) độc lập với nhau. Mô hình này được sử dụng rộng rãi trong phân loại văn bản, phát hiện thư rác, và nhiều bài toán phân loại khác nhờ vào tính hiệu quả và tốc độ xử lý nhanh chóng.

4.2.2 Nguyên lý hoạt động

1. **Bước 1**: Áp dụng định lý Bayes:

$$P(C|X) = \frac{P(X|C) \times P(C)}{P(X)}$$

   Trong đó:
   - C là nhãn lớp.
   - X là tập hợp các đặc trưng đầu vào.
   - $P(C|X)$ là xác suất xảy ra lớp C khi biết X.
   - $P(X|C)$ là xác suất quan sát được X khi biết C.
   - $P(C)$ là xác suất tiên nghiệm của lớp C.
   - $P(X)$ là xác suất tiên nghiệm của dữ liệu X.
2. **Bước 2**: Giả định rằng các đặc trưng là độc lập với nhau:

$$P(X|C) = P(X_1|C) \times P(X_2|C) \times \ldots \times P(X_n|C)$$

   Điều này giúp đơn giản hóa quá trình tính toán, đặc biệt khi số lượng đặc trưng lớn.
3. **Bước 3**: Dự đoán lớp có xác suất lớn nhất:

$$C^* = \arg\max_C P(C|X)$$

4.2.3 Đánh giá

- **Ưu điểm:** Đơn giản, nhanh chóng, hoạt động hiệu quả với tập dữ liệu có số lượng đặc trưng lớn. Yêu cầu ít dữ liệu để huấn luyện.
- **Nhược điểm:** Giả định đặc trưng độc lập thường không đúng trong thực tế, làm giảm độ chính xác trong một số trường hợp.

4.3 Decision Tree và Random Forest

4.3.1 Tổng quan

Decision Tree (Cây quyết định) là một mô hình học có giám sát đơn giản nhưng rất trực quan, sử dụng cấu trúc dạng cây để phân chia dữ liệu theo các đặc trưng, từ đó đưa ra quyết định phân loại. Tuy nhiên, Decision Tree đơn lẻ dễ bị hiện tượng overfitting, đặc biệt trên các tập dữ liệu nhỏ hoặc có nhiễu.

Để khắc phục hạn chế này, Random Forest đã ra đời như một kỹ thuật ensemble mạnh mẽ, dựa trên việc kết hợp nhiều cây quyết định (Decision Trees) để tăng cường độ chính xác và tính ổn định. Bằng cách huấn luyện nhiều cây trên các tập dữ liệu bootstrap khác nhau và ngẫu nhiên chọn đặc trưng tại mỗi nút phân chia, Random Forest tạo ra một mô hình tổng hợp hiệu quả và ít bị overfitting hơn.

4.3.2 Nguyên lý hoạt động

Decision Tree:

1. Bước 1: Xây dựng mô hình bằng cách chọn đặc trưng tốt nhất để phân chia dữ liệu, dựa trên các chỉ số như:
   - Information Gain (ID3):
   $$IG(D, A) = Entropy(D) - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} Entropy(D_v)$$
   - Gini Impurity (CART):
   $$Gini(D) = 1 - \sum_{k=1}^{K} (p_k)^2$$
   với p_k là xác suất phần tử thuộc lớp k.
2. Bước 2: Phân chia dữ liệu thành các nhánh con dựa trên giá trị của đặc trưng được chọn.
3. Bước 3: Lặp lại quá trình trên cho từng nhánh con đến khi đạt điều kiện dừng (ví dụ: các điểm dữ liệu trong nhánh thuộc cùng một lớp hoặc đạt độ sâu tối đa).

Random Forest:

1. Bước 1: Tạo nhiều tập dữ liệu bootstrap bằng cách lấy mẫu ngẫu nhiên có hoàn lại từ tập dữ liệu ban đầu.
2. Bước 2: Đối với mỗi tập bootstrap:
   - Huấn luyện một Decision Tree, nhưng tại mỗi nút phân chia chỉ xét một tập con ngẫu nhiên của các đặc trưng thay vì toàn bộ đặc trưng.

3. Bước 3: Kết hợp kết quả dự đoán của tất cả các cây:
    - Phân loại: Áp dụng voting đa số (majority voting).
    - Hồi quy: Tính trung bình giá trị dự đoán.

4.3.3 Đánh giá

- **Ưu điểm:**
    - **Decision Tree:** Dễ hiểu, trực quan, dễ giải thích, yêu cầu tiền xử lý dữ liệu ít.
    - **Random Forest:** Giảm thiểu overfitting, đạt độ chính xác cao, hoạt động tốt với dữ liệu lớn và phức tạp.
- **Nhược điểm:**
    - **Decision Tree:** Dễ overfit nếu không được cắt tỉa (pruning) hoặc điều chỉnh phù hợp.
    - **Random Forest:** Tốn nhiều tài nguyên tính toán hơn, khó giải thích hơn vì kết quả dựa trên nhiều cây.

4.4 Gradient Boosting Machines (XGBoost và LightGBM)

4.4.1 Tổng quan về Gradient Boosting Machines (GBM)

Gradient Boosting là một kỹ thuật học máy ensemble mạnh mẽ thuộc họ boosting, hoạt động bằng cách xây dựng tuần tự các mô hình dự đoán (thường là cây quyết định yếu). Mỗi mô hình mới được huấn luyện để sửa lỗi (phần dư - residuals hoặc gradient của hàm mất mát) của các mô hình trước đó. Mô hình cuối cùng là tổng hợp của tất cả các mô hình con, thường mang lại độ chính xác cao. XGBoost và LightGBM là hai triển khai GBM nâng cao, được tối ưu hóa mạnh mẽ về tốc độ, hiệu quả bộ nhớ và độ chính xác so với GBM truyền thống.

4.4.2 XGBoost (Extreme Gradient Boosting)

4.4.2.1 Giới thiệu XGBoost là một thư viện tối ưu hóa và phân tán cho Gradient Boosting, nổi tiếng về hiệu suất cao và thường xuyên chiến thắng trong các cuộc thi Machine Learning. Nó cải tiến GBM gốc bằng cách tích hợp regularization vào hàm mục tiêu và sử dụng thông tin đạo hàm bậc hai.

4.4.2.2 Nguyên lý hoạt động Mô hình XGBoost là tổng của K cây quyết định:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

Trong đó f_k là một cây trong không gian các cây $\mathcal{F}$. Tại bước t, XGBoost tìm cây f_t để tối ưu hóa hàm mục tiêu có bổ sung thành phần điều chuẩn (regularization):

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

Với thành phần điều chuẩn $\Omega(f_t)$ kiểm soát độ phức tạp của cây:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

(T: số lá, w_j: trọng số lá j, γ, λ: hệ số điều chuẩn).

Sử dụng xấp xỉ Taylor bậc hai cho hàm mất mát l quanh $\hat{y}_i^{(t-1)}$:

$$Obj^{(t)} \approx \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

($g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$).

Bỏ qua hằng số, hàm mục tiêu trở thành:

$$Obj^{(t)} \approx \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T$$

(I_j là tập các mẫu trong lá j). Trọng số tối ưu của lá j là:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Và điểm số cấu trúc tối ưu của cây là:

$$Obj^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$$

Để tìm điểm chia tốt nhất, XGBoost tính toán độ lợi (Gain) cho mỗi khả năng chia:

$$Gain = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

Phép chia có Gain lớn nhất (và lớn hơn γ) sẽ được chọn. XGBoost cũng tối ưu việc tìm điểm chia thông qua các kỹ thuật như thuật toán xấp xỉ (approximate algorithm), xử lý giá trị thiếu (sparsity-aware split finding), và cấu trúc dữ liệu block cho tính toán song song.

4.4.2.3 Đánh giá XGBoost

- **Ưu điểm:** Độ chính xác cao, Regularization mạnh mẽ chống overfitting, xử lý giá trị thiếu hiệu quả, hỗ trợ song song hóa.
- **Nhược điểm:** Nhiều siêu tham số cần tinh chỉnh, có thể tốn bộ nhớ và chậm hơn LightGBM trên tập dữ liệu rất lớn nếu dùng thuật toán chính xác (exact greedy).

4.4.3 LightGBM (Light Gradient Boosting Machine)

4.4.3.1 Giới thiệu LightGBM là một framework GBM khác do Microsoft phát triển, tập trung vào tốc độ huấn luyện và hiệu quả sử dụng bộ nhớ. Nó sử dụng các kỹ thuật chính là Histogram-based và Leaf-wise growth.

4.4.3.2 Nguyên lý hoạt động

1. **Histogram-based Algorithm:**
   - Thay vì duyệt qua tất cả các giá trị có thể để tìm điểm chia, LightGBM gom các giá trị đặc trưng liên tục vào các *bins* (thùng) rời rạc và xây dựng histogram.
   - Việc tìm điểm chia tối ưu sau đó được thực hiện trên các bins của histogram thay vì trên tất cả các điểm dữ liệu gốc. Điều này giảm đáng kể chi phí tính toán, đặc biệt với dữ liệu lớn và đặc trưng dày đặc.
   - Quá trình xây dựng histogram chỉ cần thực hiện một lần trước khi xây dựng cây.
2. **Leaf-wise Tree Growth:**
   - Khác với XGBoost thường phát triển cây theo từng tầng (level-wise), LightGBM phát triển cây theo cách **leaf-wise**.
   - Nó sẽ chọn node lá có khả năng giảm loss nhiều nhất để tiếp tục phân chia, thay vì phân chia đồng đều tất cả các lá ở cùng một độ sâu.
   - Cách này giúp hội tụ nhanh hơn và thường tạo ra các cây chính xác hơn, nhưng cũng có thể dẫn đến các cây sâu và phức tạp, dễ gây overfitting hơn nếu không kiểm soát độ sâu (ví dụ bằng `max_depth` hoặc `num_leaves`).
3. **Tối ưu hóa tìm điểm chia (Gain Calculation):**
   - Tương tự XGBoost, LightGBM cũng tìm điểm chia (trên histogram) để tối đa hóa Gain dựa trên thông tin gradient và hessian:

$$Gain = \frac{1}{2}\left(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right) - \gamma$$

   Trong đó $G = \sum g_i$, $H = \sum h_i$ được tính toán hiệu quả dựa trên histogram. λ (tham số `lambda_l2`) và γ (tham số `min_split_gain`) là các hệ số regularization.
4. **Các tối ưu khác:** LightGBM còn sử dụng Exclusive Feature Bundling (EFB) để gộp các đặc trưng thưa loại trừ lẫn nhau và Gradient-based One-Side Sampling (GOSS) để ưu tiên các mẫu có gradient lớn trong quá trình tính toán, giúp tăng tốc độ mà ít ảnh hưởng đến độ chính xác.

4.4.3.3 Đánh giá LightGBM

- **Ưu điểm:** Tốc độ huấn luyện rất nhanh, sử dụng ít bộ nhớ, hiệu quả cao trên dữ liệu lớn, hỗ trợ nhiều tính năng tối ưu.
- **Nhược điểm:** Dễ bị overfitting hơn (do leaf-wise) nếu không tinh chỉnh cẩn thận các tham số như `num_leaves`, `max_depth`, `min_child_samples`. Có thể không hiệu quả bằng XGBoost trên các tập dữ liệu rất nhỏ.

4.5 Các biến thể và Kỹ thuật xử lý chung

Cả XGBoost và LightGBM đều có thể được cải thiện hoặc điều chỉnh bằng các kỹ thuật sau, đặc biệt khi đối mặt với dữ liệu mất cân bằng:

4.5.1 Xử lý dữ liệu mất cân bằng

- **SMOTE (Synthetic Minority Over-sampling Technique):**
  - Là kỹ thuật tiền xử lý dữ liệu, áp dụng *trước khi* huấn luyện mô hình GBM.
  - Hoạt động bằng cách tạo ra các mẫu dữ liệu tổng hợp (synthetic) cho lớp thiểu số dựa trên các láng giềng gần nhất của chúng.
  - Các bước thực hiện:
    1. Xác định các điểm thuộc lớp thiểu số.
    2. Đối với mỗi điểm thiểu số x_i, chọn ngẫu nhiên k láng giềng gần nhất cùng lớp.
    3. Tạo điểm mới bằng cách nội suy giữa x_i và một láng giềng được chọn ngẫu nhiên x_{nn}:
    $$x_{\text{new}} = x_i + \delta \times (x_{\text{nn}} - x_i)$$
    với $\delta \in [0, 1]$ là số ngẫu nhiên.
  - Mô hình GBM sau đó được huấn luyện trên tập dữ liệu đã được cân bằng bởi SMOTE.
  - *Nhược điểm:* Có thể tạo ra nhiễu hoặc làm mờ ranh giới quyết định nếu không cẩn thận. Tốn thêm thời gian tiền xử lý.
- **Sử dụng trọng số (Class Weighting):**
  - Cả XGBoost và LightGBM đều hỗ trợ gán trọng số khác nhau cho các lớp thông qua tham số như `scale_pos_weight` (tỷ lệ trọng số của lớp dương so với lớp âm) hoặc `class_weight`.
  - Điều này làm cho hàm mất mát phạt nặng hơn các lỗi trên lớp thiểu số, buộc mô hình phải chú ý hơn đến chúng mà không cần thay đổi dữ liệu gốc.
- **Tham số `is_unbalance=True` (LightGBM):** LightGBM có tham số tiện lợi này để tự động điều chỉnh trọng số lớp.

4.5.2 Tinh chỉnh siêu tham số (Hyperparameter Tuning)

Để đạt hiệu suất tối ưu, việc tinh chỉnh các siêu tham số là rất quan trọng cho cả XGBoost và LightGBM. Các tham số chính bao gồm:

- Tốc độ học (`learning_rate` / `eta`)
- Số lượng cây (`n_estimators` / `num_boost_round`)
- Cấu trúc cây (`max_depth`, `num_leaves` (LightGBM), `min_child_weight` / `min_child_samples`)
- Tham số Regularization (`lambda`, `alpha`, `gamma` / `min_split_gain`)
- Tham số Subsampling (`subsample` / `bagging_fraction`, `colsample_bytree` / `feature_fraction`)

Các kỹ thuật như Grid Search, Random Search, hoặc tối ưu hóa Bayesian (ví dụ: dùng Optuna, Hyperopt) thường được sử dụng.

4.6 MLP (Keras, EarlyStopping)

4.6.1 Tổng quan

MLP (Multilayer Perceptron) là một trong những mô hình mạng nơ-ron nhân tạo cơ bản nhất, bao gồm nhiều lớp liên kết đầy đủ (fully connected). Trong nghiên cứu này, chúng tôi sử dụng thư viện Keras để xây dựng MLP, đồng thời áp dụng kỹ thuật **EarlyStopping** nhằm ngăn ngừa hiện tượng overfitting.

4.6.2 Nguyên lý hoạt động

1. Bước 1: Khởi tạo kiến trúc mạng nơ-ron
   - Bao gồm lớp đầu vào, một hoặc nhiều lớp ẩn, và lớp đầu ra.
   - Mỗi nơ-ron tại lớp l tính toán:

$$a_i^{(l)} = f\left(\sum_j w_{ij}^{(l-1)} a_j^{(l-1)} + b_i^{(l)}\right)$$

   trong đó:
   - $a_j^{(l-1)}$ là đầu ra từ lớp trước,
   - $w_{ij}^{(l-1)}$ là trọng số kết nối,
   - $b_i^{(l)}$ là hệ số bias,
   - f là hàm kích hoạt (ví dụ: ReLU, sigmoid).
2. Bước 2: Tiến hành lan truyền xuôi (forward propagation) để tính toán đầu ra dự đoán.
3. Bước 3: Tính toán hàm mất mát (loss function), ví dụ với bài toán phân loại nhị phân:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right)$$

   trong đó:
   - y_i là nhãn thực,
   - $\hat{y}_i$ là xác suất dự đoán.
4. Bước 4: Lan truyền ngược (backpropagation) để cập nhật trọng số bằng thuật toán tối ưu hóa, ví dụ Adam hoặc SGD.
5. Bước 5: Áp dụng EarlyStopping:
   - Theo dõi độ lỗi (loss) hoặc độ chính xác (accuracy) trên tập validation.
   - Nếu độ lỗi không giảm sau một số epoch nhất định (patience), quá trình huấn luyện sẽ dừng lại sớm.

4.6.3 Đánh giá

- **Ưu điểm:**
  - Khả năng mô hình hóa các mối quan hệ phi tuyến phức tạp.
  - EarlyStopping giúp giảm thiểu overfitting, tiết kiệm thời gian huấn luyện.
- **Nhược điểm:**
  - Cần tuning nhiều tham số (số lớp, số nơ-ron, learning rate,...).
  - Đòi hỏi tài nguyên tính toán cao hơn các mô hình tuyến tính đơn giản.

5 Hiện thực mô hình dự đoán

5.1 Giới thiệu tập dữ liêu

Tập dữ liệu này chứa thông tin về các vụ tai nạn giao thông tại 49 bang của Mỹ, thu thập từ tháng 2 năm 2016 đến tháng 3 năm 2023. Dữ liệu được lấy từ nhiều API cung cấp sự kiện giao thông theo thời gian thực, bao gồm các tổ chức như sở giao thông, cơ quan thực thi pháp luật, camera giao thông và cảm biến giao thông. Tập dữ liệu hiện có khoảng 7,7 triệu bản ghi.
Tập dữ liệu bao gồm các cột sau:

- **ID**: Mã nhận dạng duy nhất của bản ghi tai nạn.
- **Source**: Nguồn dữ liệu tai nạn.
- **Severity**: Mức độ nghiêm trọng của vụ tai nạn (1 đến 4).
- **Start_Time**: Thời gian bắt đầu tai nạn (theo múi giờ địa phương).
- **End_Time**: Thời gian kết thúc tai nạn (theo múi giờ địa phương).
- **Start_Lat**: Vĩ độ điểm bắt đầu của vụ tai nạn.
- **Start_Lng**: Kinh độ điểm bắt đầu của vụ tai nạn.
- **End_Lat**: Vĩ độ điểm kết thúc của vụ tai nạn.
- **End_Lng**: Kinh độ điểm kết thúc của vụ tai nạn.
- **Distance(mi)**: Khoảng cách ảnh hưởng của tai nạn tính bằng dặm.
- **Description**: Mô tả vụ tai nạn do người cung cấp.
- **Street**: Tên đường của vụ tai nạn.
- **City**: Thành phố xảy ra tai nạn.
- **County**: Quận nơi xảy ra tai nạn.
- **State**: Bang nơi xảy ra tai nạn.
- **Zipcode**: Mã bưu điện của địa phương.
- **Country**: Quốc gia nơi xảy ra tai nạn.
- **Timezone**: Múi giờ của vụ tai nạn.
- **Airport_Code**: Mã sân bay gần nhất với vị trí tai nạn.
- **Weather_Timestamp**: Thời gian quan sát thời tiết (theo múi giờ địa phương).
- **Temperature(F)**: Nhiệt độ (đơn vị F).
- **Wind_Chill(F)**: Nhiệt độ cảm nhận (đơn vị F).
- **Humidity(%)**: Độ ẩm (
- **Pressure(in)**: Áp suất không khí (đơn vị inch).

- **Visibility(mi)**: Tầm nhìn (tính bằng dặm).
- **Wind_Direction**: Hướng gió.
- **Wind_Speed(mph)**: Tốc độ gió (miles per hour).
- **Precipitation(in)**: Lượng mưa (tính bằng inch).
- **Weather_Condition**: Điều kiện thời tiết (mưa, tuyết, bão, sương mù, v.v.).
- **Amenity**: Các điểm quan tâm (POI) gần khu vực tai nạn.
- **Bump**: Ghi chú về tốc độ giảm (bump) trong khu vực tai nạn.
- **Crossing**: Ghi chú về điểm giao cắt (crossing) gần tai nạn.
- **Give_Way**: Ghi chú về nơi có biển báo nhường đường (give way).
- **Junction**: Ghi chú về điểm giao nhau (junction).
- **No_Exit**: Ghi chú về khu vực không có lối thoát (no exit).
- **Railway**: Ghi chú về sự hiện diện của đường sắt gần đó.
- **Roundabout**: Ghi chú về sự hiện diện của vòng xuyến.
- **Station**: Ghi chú về trạm giao thông gần đó.
- **Stop**: Ghi chú về biển báo dừng gần khu vực tai nạn.
- **Traffic_Calming**: Ghi chú về các biện pháp giảm tốc độ giao thông (traffic calming).
- **Traffic_Signal**: Ghi chú về sự hiện diện của tín hiệu giao thông (đèn tín hiệu).
- **Turning_Loop**: Ghi chú về khu vực có vòng xoay quay đầu (turning loop).
- **Sunrise_Sunset**: Thời gian mặt trời mọc/lặn.
- **Civil_Twilight**: Thời gian hoàng hôn dân sự.
- **Nautical_Twilight**: Thời gian hoàng hôn hàng hải.
- **Astronomical_Twilight**: Thời gian hoàng hôn thiên văn.

5.2 Tiền xử lý dữ liệu

5.2.1 Load Dữ Liệu

Do tập dữ liệu gốc có tổng cộng 46 cột, để đơn giản hoá việc xử lý và phân tích, chúng tôi lựa chọn 33 cột được cho là cần thiết, bao gồm:

```
cols = ["Severity","Start_Time","Start_Lat","Start_Lng","Distance(mi)",
"State","Timezone","Temperature(F)","Wind_Chill(F)","Humidity(%)",
"Pressure(in)","Visibility(mi)","Wind_Direction","Wind_Speed(mph)",
"Precipitation(in)","Weather_Condition","Amenity","Bump","Crossing",
"Give_Way","Junction","No_Exit","Railway","Roundabout","Station",
"Stop","Traffic_Calming","Traffic_Signal","Turning_Loop","Sunrise_Sunset",
```

```
"Civil_Twilight","Nautical_Twilight","Astronomical_Twilight"]

data = pd.read_csv('data/US_Accidents_March23.csv', usecols=cols)
print("The shape of data is:",(data.shape))
display(data.head(3))
```

Kết quả:

- Kích thước dữ liệu: **7.728.394 dòng** $\times$ **33 cột**.
- Ba dòng dữ liệu đầu tiên được hiển thị để kiểm tra sơ bộ.

5.2.2 Thông Tin Tổng Quan về Dữ Liệu

Sử dụng lệnh:

```
data.info()
```

Ta thu được các thông tin sau:

- Tổng số dòng: 7.728.394 dòng.
- Tổng số cột: 33 cột.
- Các kiểu dữ liệu:
  - **bool**: 13 cột (ví dụ: `Amenity`, `Bump`, `Crossing`, ...)
  - **float64**: 10 cột (ví dụ: `Temperature(F)`, `Humidity(%)`, `Visibility(mi)`, ...)
  - **int64**: 1 cột (`Severity`)
  - **object**: 9 cột (ví dụ: `Start_Time`, `State`, `Weather_Condition`, ...)
- Bộ nhớ sử dụng: khoảng 1.2+ GB.

Nhận xét ban đầu:

- Các cột kiểu `bool`, `float64`, `int64` có vẻ đã được định dạng đúng.
- Các thông tin thời gian như `Start_Time` và các cột liên quan đến thời điểm trong ngày (`Sunrise_Sunset`, `Civil_Twilight`, ...) hiện đang ở dạng chuỗi (`object`). `Start_Time` cần chuyển đổi sang dạng `datetime` để trích xuất các đặc trưng thời gian. Các cột twilight/sunrise cũng cần xem xét xử lý phù hợp. Các cột `object` khác như `Wind_Direction`, `Weather_Condition` là các biến phân loại dạng chuỗi.

5.2.3 Xử lý Dữ liệu Thời gian

Chuyển đổi cột `Start_Time` sang định dạng `datetime` và trích xuất các đặc trưng mới: Năm, Tháng, Ngày, Giờ, Thứ trong tuần. Sau đó, loại bỏ cột `Start_Time` gốc.

```
data['Start_Time'] = pd.to_datetime(data['Start_Time'],format="ISO8601")

# Extracting time features
data['Year'] = data['Start_Time'].dt.year
data['Month'] = data['Start_Time'].dt.month
```

```
data['Day'] = data['Start_Time'].dt.day
data['Hour'] = data['Start_Time'].dt.hour
data['Week'] = data['Start_Time'].dt.weekday

# Drop the original Start_Time column
data.drop(['Start_Time'], axis=1, inplace=True)
```

Kết quả:

- Cột `Start_Time` được thay thế bằng 5 cột mới: `Year`, `Month`, `Day`, `Hour`, `Week`.
- Số lượng cột trong dữ liệu tăng lên 37 (33 - 1 + 5).

5.2.4 Kiểm tra Dữ liệu Thiếu (Missing Values)

Tính toán tỷ lệ phần trăm dữ liệu thiếu cho từng cột.

```
missing = pd.DataFrame(data.isnull().sum()).reset_index()
missing.columns = ['Feature', 'Missing_Percent(%)']
missing['Missing_Percent(%)'] =
↪  missing['Missing_Percent(%)'].apply(lambda x: x / data.shape[0] *
↪  100)
missing_df = missing.loc[missing['Missing_Percent(%)']>0,:]
print(missing_df)
```

Kết quả các cột có dữ liệu thiếu và tỷ lệ tương ứng:

- `Timezone`: 0.10%
- `Temperature(F)`: 2.12%
- `Wind_Chill(F)`: 25.87%
- `Humidity(%)`: 2.25%
- `Pressure(in)`: 1.82%
- `Visibility(mi)`: 2.29%
- `Wind_Direction`: 2.27%
- `Wind_Speed(mph)`: 7.39%
- `Precipitation(in)`: 28.51%
- `Weather_Condition`: 2.24%
- `Sunrise_Sunset`: 0.30%
- `Civil_Twilight`: 0.30%
- `Nautical_Twilight`: 0.30%
- `Astronomical_Twilight`: 0.30%

Nhận xét:

- Hai cột `Wind_Chill(F)` và `Precipitation(in)` có tỷ lệ thiếu rất cao (trên 25%).
- Các cột còn lại có tỷ lệ thiếu tương đối thấp (dưới 8%).

5.2.5 Xử lý Dữ liệu Thiếu (Loại bỏ dòng)

Loại bỏ các dòng có giá trị thiếu ở các cột có tỷ lệ thiếu thấp, bao gồm các cột thời tiết và thời điểm trong ngày.

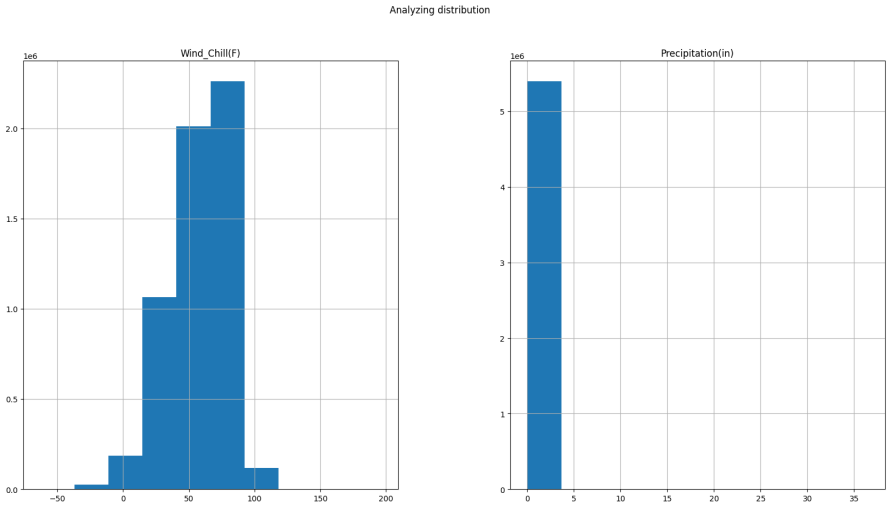
```
data = data.dropna(subset=['Wind_Direction','Visibility(mi)',
'Pressure(in)','Humidity(%)',"Civil_Twilight","Nautical_Twilight",
"Astronomical_Twilight",'Sunrise_Sunset','Weather_Condition'])
```

Sau bước này, số lượng dòng dữ liệu sẽ giảm xuống, loại bỏ các bản ghi không đầy đủ thông tin ở các cột quan trọng có tỷ lệ thiếu thấp.

5.2.6 Phân tích và Xử lý các Cột Thiếu Nhiều

Phân tích sự phân bố của hai cột có tỷ lệ thiếu cao là `Wind_Chill(F)` (thiếu 26%) và `Precipitation(in)` (thiếu 28%) bằng biểu đồ histogram (Xem Hình 1).

```
data.hist(
    column=['Wind_Chill(F)', 'Precipitation(in)'],
    figsize=(20, 10)
)
pylab.suptitle("Analyzing distribution", fontsize="large")
```



Hình 1. Biểu đồ phân bố của Wind_Chill(F) và Precipitation(in)

Quan sát biểu đồ Hình 1:

- `Wind_Chill(F)`: Có phân bố tương đối giống phân phối chuẩn, tập trung chủ yếu trong khoảng từ 0 đến 100 độ F.
- `Precipitation(in)`: Phân bố bị lệch phải rất mạnh, hầu hết các giá trị tập trung gần 0 (không có hoặc rất ít mưa), chỉ một số ít giá trị lớn hơn.

Kết hợp với tỷ lệ thiếu cao và sự phân bố cực kỳ lệch của `Precipitation(in)`, chúng tôi quyết định loại bỏ cột này khỏi tập dữ liệu.
Đối với cột `Wind_Chill(F)`, mặc dù có tỷ lệ thiếu đáng kể, nhưng phân bố của nó có vẻ hợp lý hơn. Chúng tôi quyết định giữ lại cột này và loại bỏ các dòng có giá trị thiếu tại cột này.

5.2.7 Kiểm tra lại Dữ liệu Thiếu và Kích thước Cuối cùng

Kiểm tra lại xem còn cột nào có dữ liệu thiếu không.

```
missing = pd.DataFrame(data.isnull().sum()).reset_index()
missing.columns = ['Feature', 'Missing_Percent(%)']
missing['Missing_Percent(%)'] =
↪  missing['Missing_Percent(%)'].apply(lambda x: x / data.shape[0] *
↪  100)
missing_final = missing.loc[missing['Missing_Percent(%)']>0,:]
print("Missing values after handling:")
print(missing_final)

print("The final shape of data is:",(data.shape))
```

Kết quả:

- Không còn dữ liệu thiếu trong tập dữ liệu.
- Kích thước dữ liệu cuối cùng sau khi xử lý thiếu: **5.667.054 dòng** $\times$ **36 cột**.

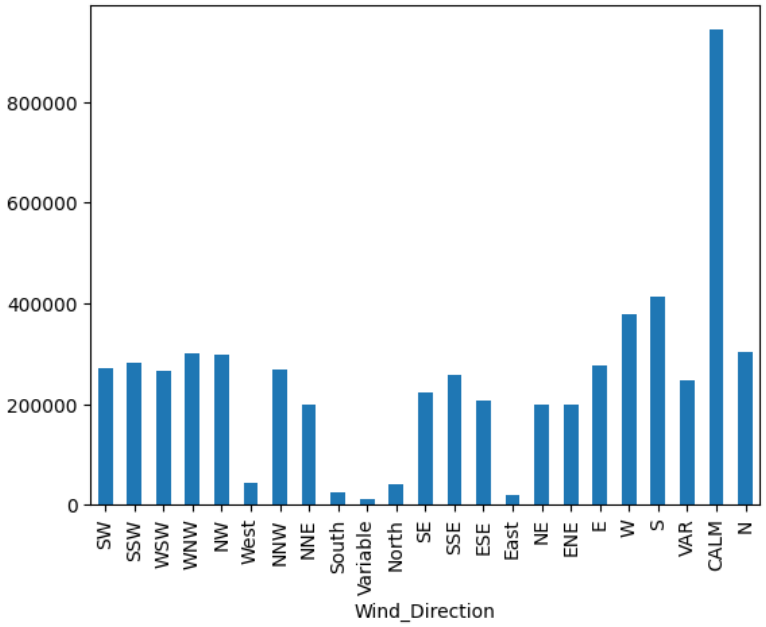
5.2.8 Đơn giản hóa Biến Phân loại: Wind_Direction

Kiểm tra các giá trị duy nhất và tần suất xuất hiện của chúng trong cột `Wind_Direction` bằng biểu đồ cột (Xem Hình 2).

```
# Display unique values (as before)
print("Wind Direction: ", data['Wind_Direction'].unique())

# Plot value counts
#
↪  pd.Series(data['Wind_Direction']).value_counts(sort=False).plot(kind='bar')
# Assuming the plot is generated and saved as
↪  'wind_direction_initial.png'
```

Kết quả các giá trị duy nhất ban đầu:

```
['SW' 'SSW' 'WSW' 'WNW' 'NW' 'West' 'NNW' 'NNE' 'South' 'Variable'
↪  'North' 'SE' 'SSE' 'ESE' 'East' 'NE' 'ENE' 'E' 'W' 'S' 'VAR' 'CALM'
↪  'N']
```



Hình 2. Biểu đồ tần suất các giá trị ban đầu của Wind_Direction

Quan sát biểu đồ Hình 2 và danh sách các giá trị duy nhất, chúng tôi nhận thấy:

- Có nhiều giá trị khác nhau nhưng cùng biểu thị một hướng gió chính (ví dụ: 'West', 'WSW', 'WNW' đều là hướng Tây; 'South', 'SSW', 'SSE' đều là hướng Nam).
- Giá trị 'CALM' (gió lặng) có tần suất xuất hiện rất cao.
- Có các giá trị không rõ ràng như 'Variable', 'VAR'.

Để giảm số lượng nhóm và tăng tính khái quát, chúng tôi tiến hành gom các giá trị chi tiết vào 8 hướng chính (N, S, E, W, NE, NW, SE, SW) và giữ lại 'CALM'. Đồng thời, loại bỏ các giá trị không xác định ('VAR', 'Variable').
Kết quả các giá trị duy nhất sau khi đơn giản hóa:

```
['SW' 'S' 'W' 'NW' 'N' 'SE' 'E' 'NE' 'CALM']
```

Số lượng nhóm của biến `Wind_Direction` đã giảm xuống còn 9 nhóm, phù hợp hơn cho việc xây dựng mô hình.

5.2.9 Đơn giản hóa Biến Phân loại: Weather_Condition

Cột `Weather_Condition` chứa các mô tả chi tiết về điều kiện thời tiết, dẫn đến một số lượng lớn các giá trị duy nhất. Để kiểm tra sự đa dạng này, chúng tôi trích xuất các thuật ngữ thời tiết cơ bản từ các mô tả.

```
# Extracting unique weather terms from the descriptions
weather ='!'.join(data['Weather_Condition'].dropna().unique().tolist())
weather = np.unique(np.array(re.split(
```

```
    "!|\s/\s|\sand\s|\swith\s|Partly\s|Mostly\s|Blowing\s|Freezing\s",
    ↪  weather))).tolist()
# print("Weather Conditions: ", weather) # Output is very long
print(f"Number of unique terms initially extracted: {len(weather)}")
# Example of some terms:
print("Sample initial terms:", weather[1:15]) # Show a few examples
```

Kết quả cho thấy có rất nhiều thuật ngữ thời tiết khác nhau (ví dụ: ”, ’Clear’, ’Cloudy’, ’Drifting Snow’, ’Drizzle’, ’Dust’, ..., ’Wintry Mix’). Để giảm độ phức tạp và tạo ra các nhóm có ý nghĩa hơn cho mô hình, chúng tôi tiến hành gom nhóm các điều kiện thời tiết tương tự dựa trên các từ khóa chính. Các nhóm chính được xác định là: Clear (trong, quang đãng), Cloud (có mây), Rain (mưa nhẹ/vừa), Heavy_Rain (mưa lớn/dông), Snow (tuyết/mưa đá nhẹ), Heavy_Snow (tuyết/mưa đá nặng), Fog (sương mù), và Dust (bụi/khói).
Kết quả các giá trị duy nhất của `Weather_Condition` sau khi được gom nhóm và làm sạch:

```
['Clear', 'Cloud', 'Dust', 'Fog', 'Heavy_Rain', 'Heavy_Snow', 'Rain',
↪  'Snow']
```

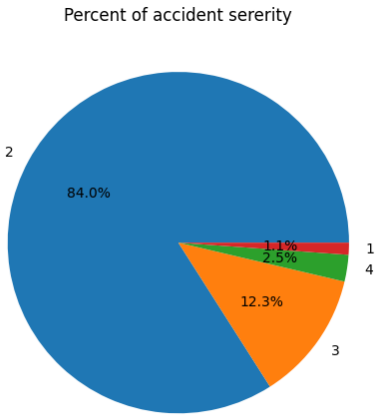
Sau bước này, cột `Weather_Condition` đã được chuẩn hóa thành 8 nhóm chính, giúp giảm độ phức tạp và thuận tiện hơn cho việc phân tích cũng như xây dựng mô hình dự đoán. Quá trình tiền xử lý dữ liệu cơ bản kết thúc tại đây.

5.3 Phân tích Dữ liệu Khám phá (EDA)

Sau khi hoàn thành các bước tiền xử lý cơ bản, giai đoạn tiếp theo là phân tích dữ liệu khám phá nhằm hiểu rõ hơn về đặc điểm của dữ liệu và rút ra những nhận định ban đầu.

5.3.1 Phân tích Biến Mục tiêu: Severity

Biến mục tiêu trong bài toán này là `Severity`, biểu thị mức độ nghiêm trọng của vụ tai nạn, với thang đo từ 1 đến 4. Sự phân bố của biến này được kiểm tra thông qua biểu đồ và thống kê tần suất.



Hình 3. Biểu đồ tròn thể hiện tỷ lệ phần trăm các mức độ nghiêm trọng tai nạn

Quan sát biểu đồ Hình 3 và kết quả đếm số lượng, có thể nhận thấy:

- Dữ liệu thể hiện sự mất cân bằng nghiêm trọng giữa các lớp.
- Mức độ nghiêm trọng 2 (`Severity=2`) chiếm tỷ lệ áp đảo với 84.0%.
- Mức độ 3 (`Severity=3`) chiếm 12.3%.
- Các mức độ 1 và 4 chiếm tỷ lệ rất nhỏ (lần lượt là 1.1% và 2.5%).

Sự mất cân bằng này đặt ra thách thức cho việc xây dựng mô hình dự đoán đa lớp hiệu quả. Để đơn giản hóa bài toán và tập trung vào việc phân biệt giữa các vụ tai nạn ít nghiêm trọng và nghiêm trọng, biến mục tiêu được gom nhóm lại thành hai loại:

- **Loại 0**: Ít nghiêm trọng (bao gồm mức độ 1 và 2 cũ).
- **Loại 1**: Nghiêm trọng (bao gồm mức độ 3 và 4 cũ).

Việc gom nhóm được thực hiện như sau:

```
# Reclassify Severity into binary: 0 for less severe (1, 2), 1 for more severe (3, 4)
data.loc[(data['Severity'] <= 2), 'Severity'] = 0
data.loc[(data['Severity'] >= 3), 'Severity'] = 1

# Check value counts after reclassification
print("\nSeverity Counts after Binarization:")
print(data['Severity'].value_counts())
```

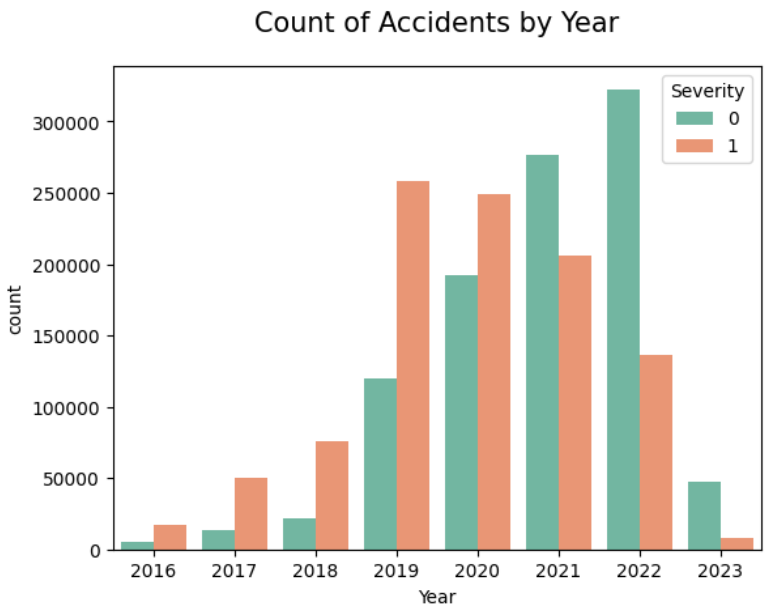
Kết quả sau khi gom nhóm:

- Severity 0 (Ít nghiêm trọng): 4.603.657 bản ghi.
- Severity 1 (Nghiêm trọng): 802.045 bản ghi.

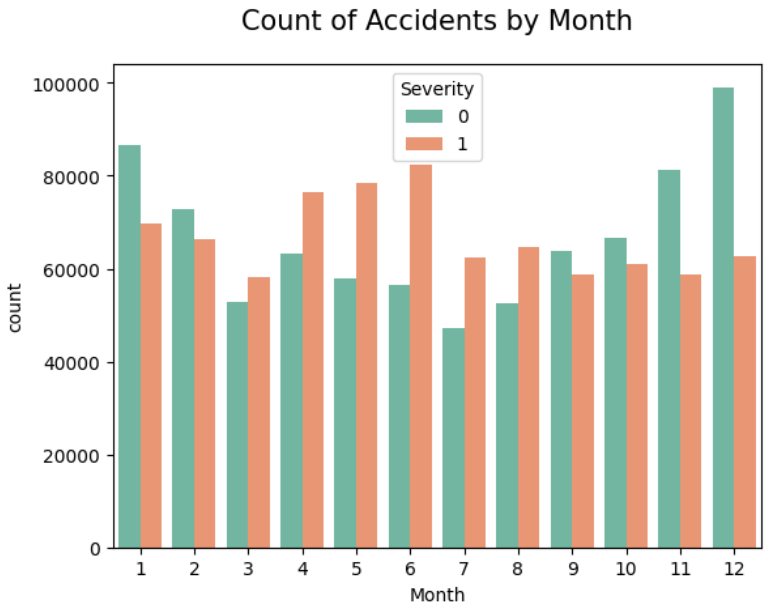
Mặc dù đã gom nhóm, dữ liệu vẫn còn mất cân bằng đáng kể giữa hai lớp (tỷ lệ khoảng 5.7 : 1). Vấn đề này cần được giải quyết trong giai đoạn chuẩn bị dữ liệu cho mô hình, ví dụ bằng các kỹ thuật lấy mẫu lại (resampling) như Undersampling lớp đa số hoặc Oversampling lớp thiểu số (ví dụ: SMOTE) để tạo ra tập dữ liệu huấn luyện cân bằng hơn.

5.3.2 Phân tích theo Thời gian

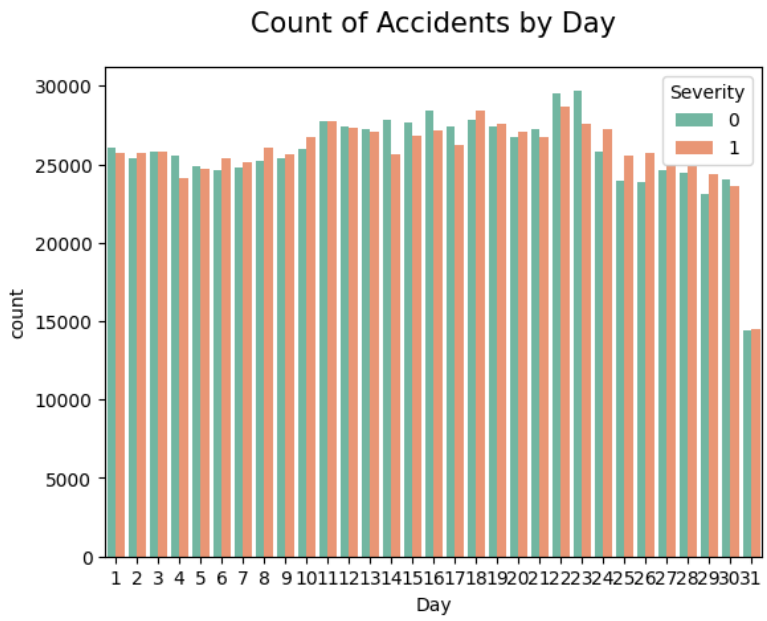
Sau khi tiền xử lý, số lượng tai nạn được phân tích theo các đặc trưng thời gian đã trích xuất (Năm, Tháng, Ngày, Thứ trong tuần, Giờ) và các đặc trưng về thời điểm trong ngày (Sunrise/Sunset, Civil/Nautical/Astronomical Twilight). Sự khác biệt giữa hai mức độ nghiêm trọng (0: Ít nghiêm trọng, 1: Nghiêm trọng) cũng được xem xét. Để trực quan hóa, tập dữ liệu đã được lấy mẫu lại cân bằng (`re_df`) được sử dụng nhằm tránh sự áp đảo của lớp ít nghiêm trọng trong biểu đồ.



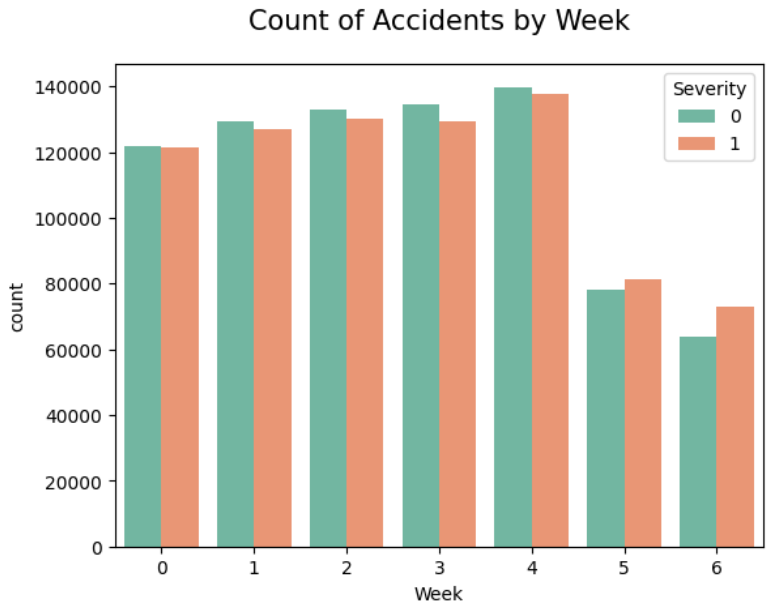
Hình 4. Số lượng tai nạn theo Năm và Mức độ nghiêm trọng



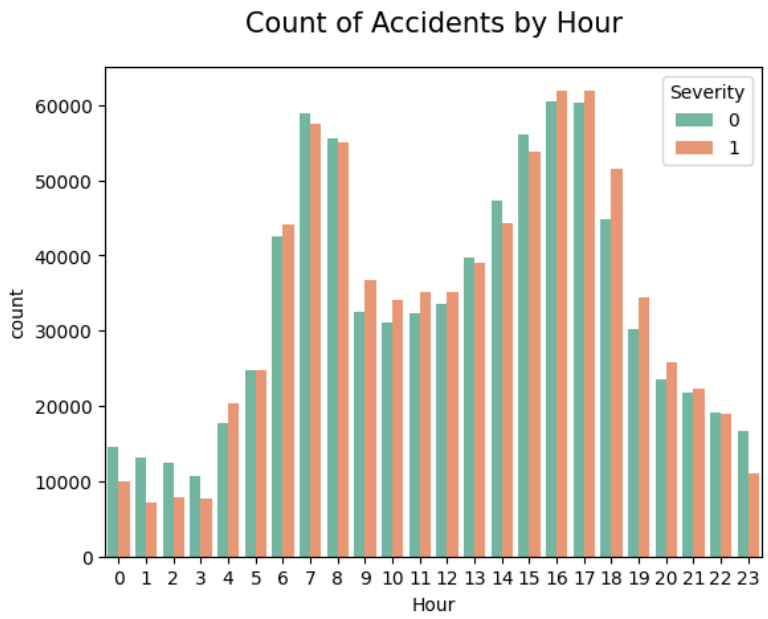
Hình 5. Số lượng tai nạn theo Tháng và Mức độ nghiêm trọng



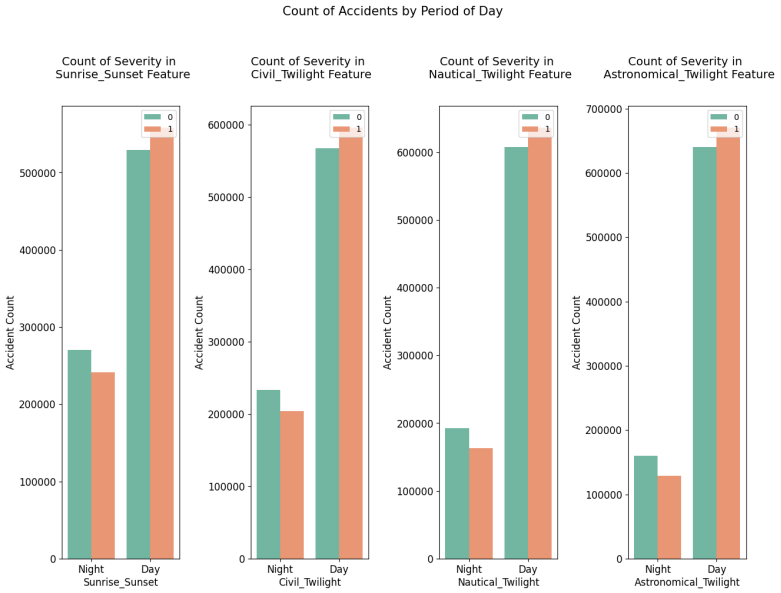
Hình 6. Số lượng tai nạn theo Ngày trong tháng và Mức độ nghiêm trọng



Hình 7. Số lượng tai nạn theo Thứ trong tuần và Mức độ nghiêm trọng (0: Thứ 2, ..., 6: Chủ Nhật)



Hình 8. Số lượng tai nạn theo Giờ trong ngày và Mức độ nghiêm trọng



Hình 9. Số lượng tai nạn theo Thời điểm trong ngày và Mức độ nghiêm trọng

Phân tích các biểu đồ từ Hình 4 đến 9 cho thấy các điểm sau:

- **Theo Năm:** Số lượng tai nạn được ghi nhận tăng dần từ 2016 đến 2022, đặc biệt là các vụ tai nạn ít nghiêm trọng. Dữ liệu năm 2023 có vẻ chưa đầy đủ. Để đảm bảo tính nhất quán, dữ liệu được lọc để chỉ giữ lại các bản ghi từ năm 2019 trở đi.

- **Theo Tháng:** Các tháng cuối năm (10, 11, 12) có xu hướng ghi nhận nhiều tai nạn hơn, đặc biệt là các vụ ít nghiêm trọng. Các vụ tai nạn nghiêm trọng có vẻ tăng cao hơn vào giữa năm (tháng 4 đến tháng 7).
- **Theo Ngày:** Không có xu hướng rõ rệt về số lượng tai nạn theo ngày trong tháng, ngoại trừ số lượng giảm nhẹ vào ngày 31 (do không phải tháng nào cũng có 31 ngày).
- **Theo Thứ trong tuần:** Số lượng tai nạn cao hơn vào các ngày trong tuần (Thứ 2 đến Thứ 6, tương ứng 0-4) và giảm vào cuối tuần (Thứ 7 và Chủ Nhật, tương ứng 5-6). Tai nạn nghiêm trọng có vẻ xảy ra nhiều hơn một chút vào cuối tuần.
- **Theo Giờ:** Có hai đỉnh điểm về số lượng tai nạn trong ngày: vào buổi sáng (khoảng 7-8 giờ) và buổi chiều (khoảng 16-17 giờ), trùng với giờ cao điểm đi làm và tan tầm. Số lượng tai nạn nghiêm trọng cũng tăng cao vào các khung giờ này.
- **Theo Thời điểm trong ngày:** Phần lớn tai nạn xảy ra vào ban ngày (`Day`) theo tất cả các định nghĩa (Sunrise/Sunset, Civil, Nautical, Astronomical Twilight). Tai nạn nghiêm trọng cũng có xu hướng xảy ra nhiều hơn vào ban ngày. Các đặc trưng Twilight (`Civil_Twilight`, `Nautical_Twilight`, `Astronomical_Twilight`) cung cấp thông tin tương tự nhau và rất giống với `Sunrise_Sunset`.

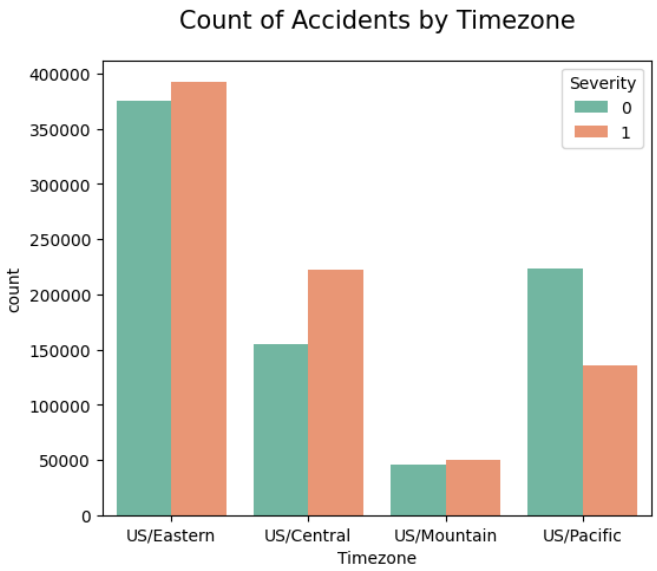
Dựa trên các phân tích này, các điều chỉnh sau được thực hiện trên dữ liệu:

- Lọc dữ liệu, chỉ giữ lại các bản ghi từ năm 2019 trở đi.
- Loại bỏ các cột `Civil_Twilight`, `Nautical_Twilight`, `Astronomical_Twilight` do tính dư thừa thông tin so với `Sunrise_Sunset`.
- Thực hiện lại việc lấy mẫu cân bằng (`resampleData`) trên tập dữ liệu đã lọc theo năm (chi tiết sẽ trình bày ở phần chuẩn bị dữ liệu cho mô hình).

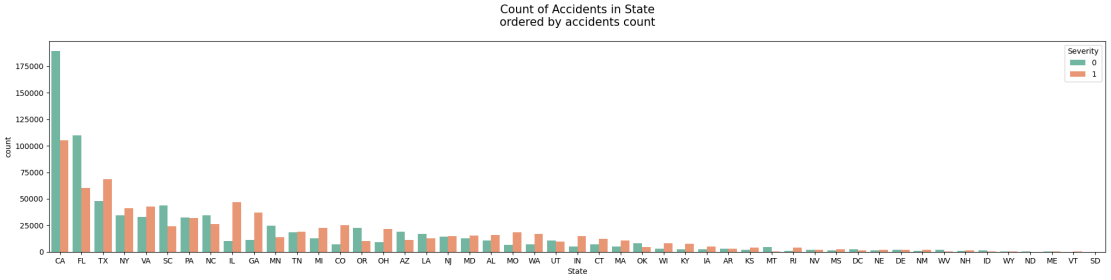
```
# Filter data by year
data = data.loc[data['Year'] >= 2019]
print("Value counts after filtering by year:")
print(data['Severity'].value_counts())

# Resample the filtered data later (code removed from EDA section)
# ...

# Drop redundant twilight columns
data =
↪  data.drop(['Civil_Twilight','Nautical_Twilight','Astronomical_Twilight'],
↪  axis=1)
```

5.3.3 Phân tích theo Vị trí Địa lý

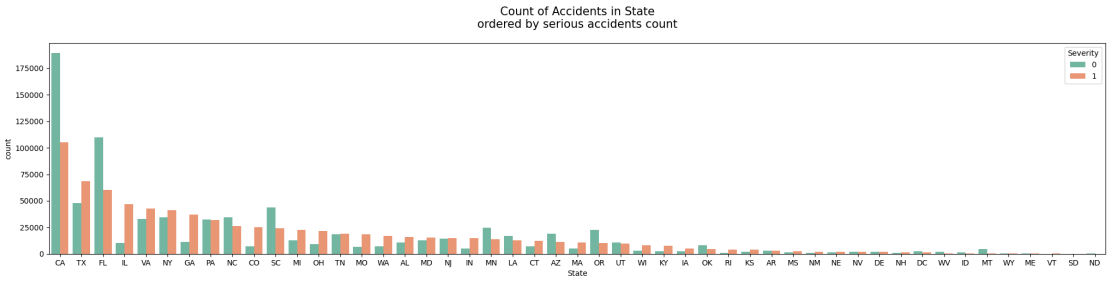
Phân tích tiếp theo tập trung vào các đặc trưng liên quan đến vị trí địa lý của các vụ tai nạn, bao gồm Múi giờ (Timezone), Bang (State), và Tọa độ (Latitude, Longitude). Tương tự như phân tích theo thời gian, tập dữ liệu cân bằng `re_df` được sử dụng cho các biểu đồ đếm theo hạng mục để dễ so sánh giữa hai mức độ nghiêm trọng.



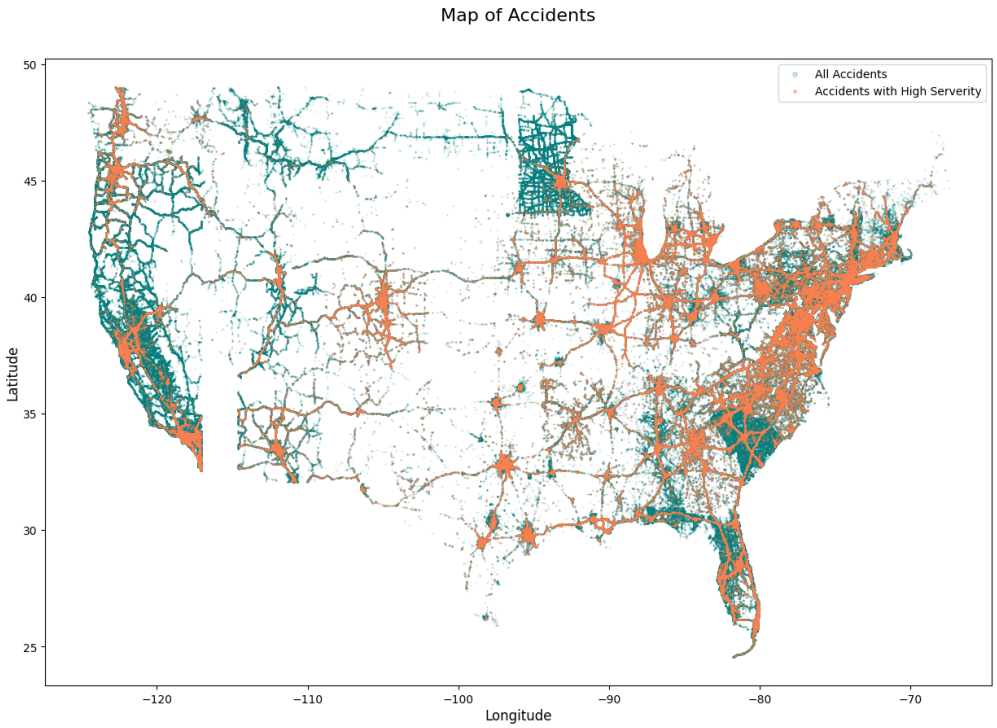
Hình 10. Số lượng tai nạn theo Múi giờ và Mức độ nghiêm trọng



Hình 11. Số lượng tai nạn theo Bang (sắp xếp theo tổng số tai nạn) và Mức độ nghiêm trọng



Hình 12. Số lượng tai nạn theo Bang (sắp xếp theo số tai nạn nghiêm trọng) và Mức độ nghiêm trọng



Hình 13. Bản đồ phân bố vị trí các vụ tai nạn trên toàn nước Mỹ

Nhận xét từ các biểu đồ phân tích theo vị trí (Hình 10 đến 13):

- **Theo Múi giờ:** Múi giờ miền Đông (`US/Eastern`) có số lượng tai nạn cao nhất, theo sau là miền Trung (`US/Central`) và Thái Bình Dương (`US/Pacific`). Múi giờ miền núi (`US/Mountain`) có số lượng thấp nhất. Tai nạn nghiêm trọng (`Severity=1`) dường như chiếm tỷ lệ cao hơn ở múi giờ miền Trung và miền Đông.
- **Theo Bang (Tổng số):** California (CA) và Florida (FL) là hai bang có tổng số vụ tai nạn được ghi nhận cao nhất, theo sau là Texas (TX). Các bang còn lại có số lượng thấp hơn đáng kể.
- **Theo Bang (Số vụ nghiêm trọng):** Khi sắp xếp theo số vụ tai nạn nghiêm trọng (`Severity=1`), thứ tự các bang dẫn đầu có thay đổi. California (CA) vẫn đứng đầu, nhưng Florida (FL) và Texas (TX) cũng có số lượng rất cao. Các bang như South Carolina (SC), North Carolina (NC), Pennsylvania (PA) cũng nổi lên với số lượng tai nạn nghiêm trọng tương đối lớn.
- **Bản đồ phân bố:** Hình 13 cho thấy sự tập trung rõ rệt của các vụ tai nạn dọc theo các trục đường chính, đặc biệt là ở các khu vực đô thị và các tuyến đường cao tốc liên bang. Các vụ tai nạn nghiêm trọng (màu cam san hô) có xu hướng phân bố rộng hơn, không chỉ tập trung ở các thành phố lớn mà còn xuất hiện nhiều trên các tuyến đường dài và các khu vực thưa dân hơn so với tổng số tai nạn (màu xanh mòng két). Điều này cho thấy tai nạn nghiêm trọng có thể liên quan đến các yếu tố khác ngoài mật độ giao thông đô thị, ví dụ như tốc độ cao trên đường cao tốc hoặc điều kiện đường xá ở khu vực nông thôn.

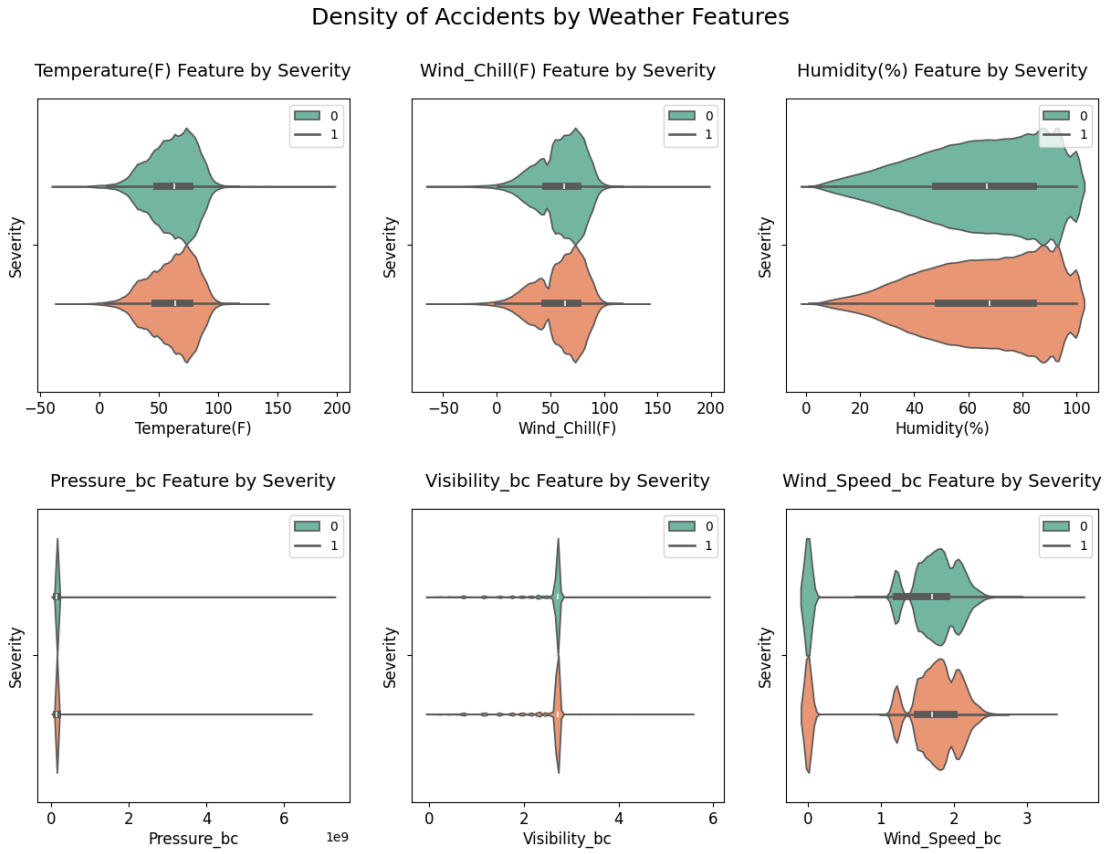
Các đặc trưng vị trí như Bang và Tọa độ cung cấp thông tin quan trọng về nơi tai nạn thường xảy ra và có thể là những yếu tố dự báo mạnh mẽ cho mức độ nghiêm trọng.

5.3.4 Phân tích theo Đặc trưng Thời tiết

Các đặc trưng liên quan đến thời tiết đóng vai trò quan trọng trong việc xác định rủi ro tai nạn. Phần này phân tích ảnh hưởng của các yếu tố thời tiết số (nhiệt độ, độ ẩm, áp suất, tầm nhìn, tốc độ gió) và các yếu tố thời tiết phân loại (hướng gió, điều kiện thời tiết tổng quát) đến mức độ nghiêm trọng của tai nạn.

Do một số đặc trưng số như áp suất (`Pressure(in)`), tầm nhìn (`Visibility(mi)`), và tốc độ gió (`Wind_Speed(mph)`) có phân bố rất lệch (skewed), phép biến đổi Box-Cox được áp dụng để làm cho phân bố của chúng gần với phân phối chuẩn hơn, giúp các mô hình hoạt động hiệu quả hơn. Các cột gốc sau đó được loại bỏ. Tập dữ liệu cân bằng `re_df` tiếp tục được sử dụng cho việc trực quan hóa.

5.3.4.1 Phân tích các Đặc trưng Thời tiết Số: Biểu đồ violin (Hình 14) được sử dụng để so sánh mật độ phân bố của các đặc trưng thời tiết số giữa hai mức độ nghiêm trọng.

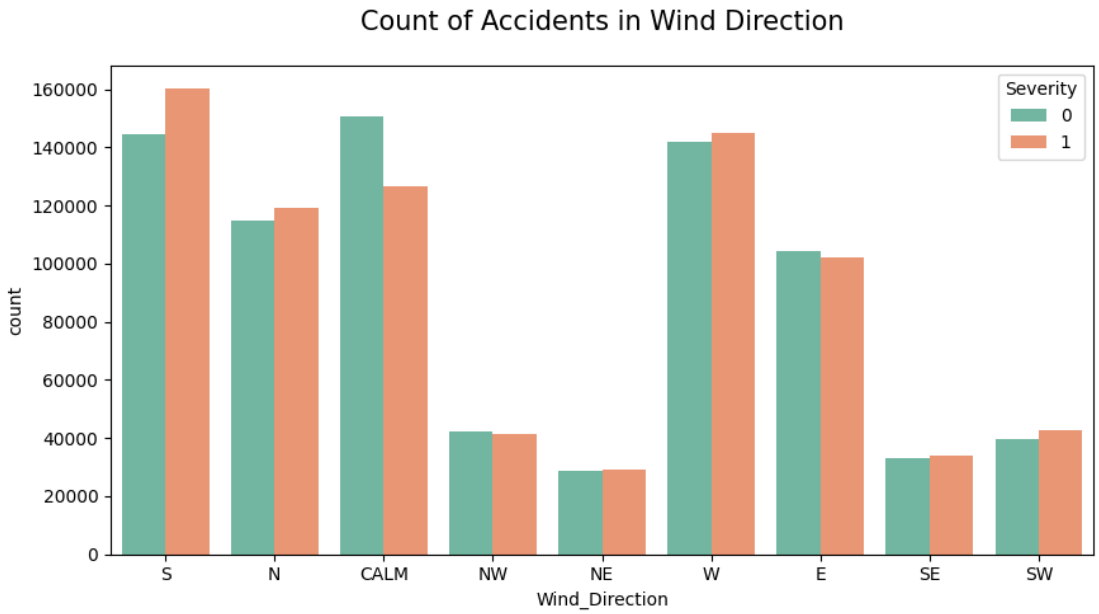


Hình 14. Mật độ phân bố của các đặc trưng thời tiết số theo Mức độ nghiêm trọng

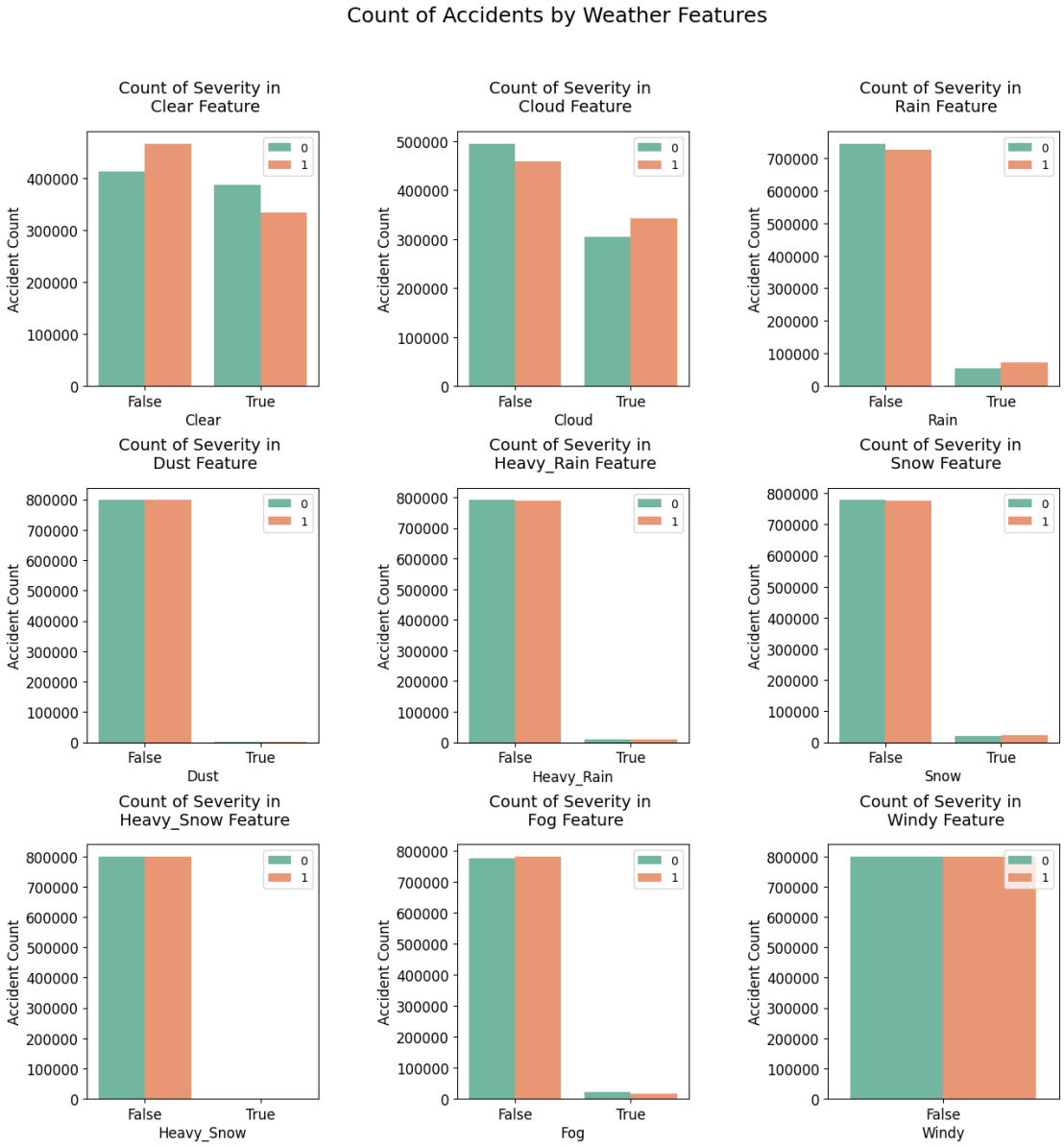
Quan sát Hình 14:

- **Temperature(F) và Wind_Chill(F):** Cả hai đều cho thấy tai nạn nghiêm trọng (`Severity=1`) có xu hướng xảy ra ở nhiệt độ cảm nhận thấp hơn một chút so với tai nạn ít nghiêm trọng (`Severity=0`). Phân bố của tai nạn ít nghiêm trọng tập trung hơn ở khoảng nhiệt độ ôn hòa (50-80F).
- **Humidity(%):** Tai nạn nghiêm trọng có xu hướng xảy ra ở độ ẩm cao hơn một chút so với tai nạn ít nghiêm trọng. Phân bố độ ẩm của tai nạn ít nghiêm trọng rộng hơn.
- **Pressure_bc, Visibility_bc, Wind_Speed_bc:** Sau khi biến đổi Box-Cox, các phân bố trở nên đối xứng hơn. Không có sự khác biệt quá rõ rệt về áp suất và tốc độ gió giữa hai mức độ nghiêm trọng. Tuy nhiên, đối với tầm nhìn (`Visibility_bc`), tai nạn ít nghiêm trọng có vẻ tập trung ở tầm nhìn tốt hơn (giá trị biến đổi cao hơn) so với tai nạn nghiêm trọng.

5.3.4.2 Phân tích các Đặc trưng Thời tiết Phân loại: Biểu đồ cột được sử dụng để xem xét số lượng tai nạn theo Hướng gió và Điều kiện thời tiết tổng quát.



Hình 15. Số lượng tai nạn theo Hướng gió và Mức độ nghiêm trọng



Hình 16. Số lượng tai nạn theo Điều kiện thời tiết và Mức độ nghiêm trọng

Quan sát Hình 15 và 16:

- **Hướng gió:** Không có sự khác biệt quá lớn về tỷ lệ tai nạn nghiêm trọng giữa các hướng gió khác nhau, ngoại trừ 'CALM' (gió lặng) có số lượng tai nạn ít nghiêm trọng cao hơn hẳn.
- **Điều kiện thời tiết:**
  - Các điều kiện thời tiết phổ biến như 'Clear' (Trời quang) và 'Cloud' (Nhiều mây) chiếm phần lớn số vụ tai nạn ở cả hai mức độ nghiêm trọng.

- 'Rain' (Mưa) cũng là một điều kiện thời tiết phổ biến dẫn đến tai nạn.
- Các điều kiện thời tiết khắc nghiệt hơn như 'Heavy_Rain' (Mưa lớn/Dông), 'Snow' (Tuyết), 'Heavy_Snow' (Tuyết dày) có số lượng tai nạn ít hơn nhiều so với 'Clear' hay 'Cloud', nhưng có thể tiềm ẩn nguy cơ cao hơn.
- Các điều kiện như 'Fog' (Sương mù), 'Windy' (Gió mạnh), và 'Dust' (Bụi/Khói) có số lượng tai nạn tương đối thấp trong tập dữ liệu đã lấy mẫu lại. Để đơn giản hóa và tập trung vào các điều kiện phổ biến hơn, các bản ghi ứng với ba điều kiện này sẽ được loại bỏ.

Dựa trên phân tích, các bản ghi có `Weather_Condition` là 'Fog', 'Windy', hoặc 'Dust' được lọc bỏ khỏi tập dữ liệu. Việc lấy mẫu lại được thực hiện một lần nữa sau khi lọc.

```
data = data.loc[data['Weather_Condition'] != "Fog"]
data = data.loc[data['Weather_Condition'] != "Windy"]
data = data.loc[data['Weather_Condition'] != "Dust"]

print(data['Severity'].value_counts())

```

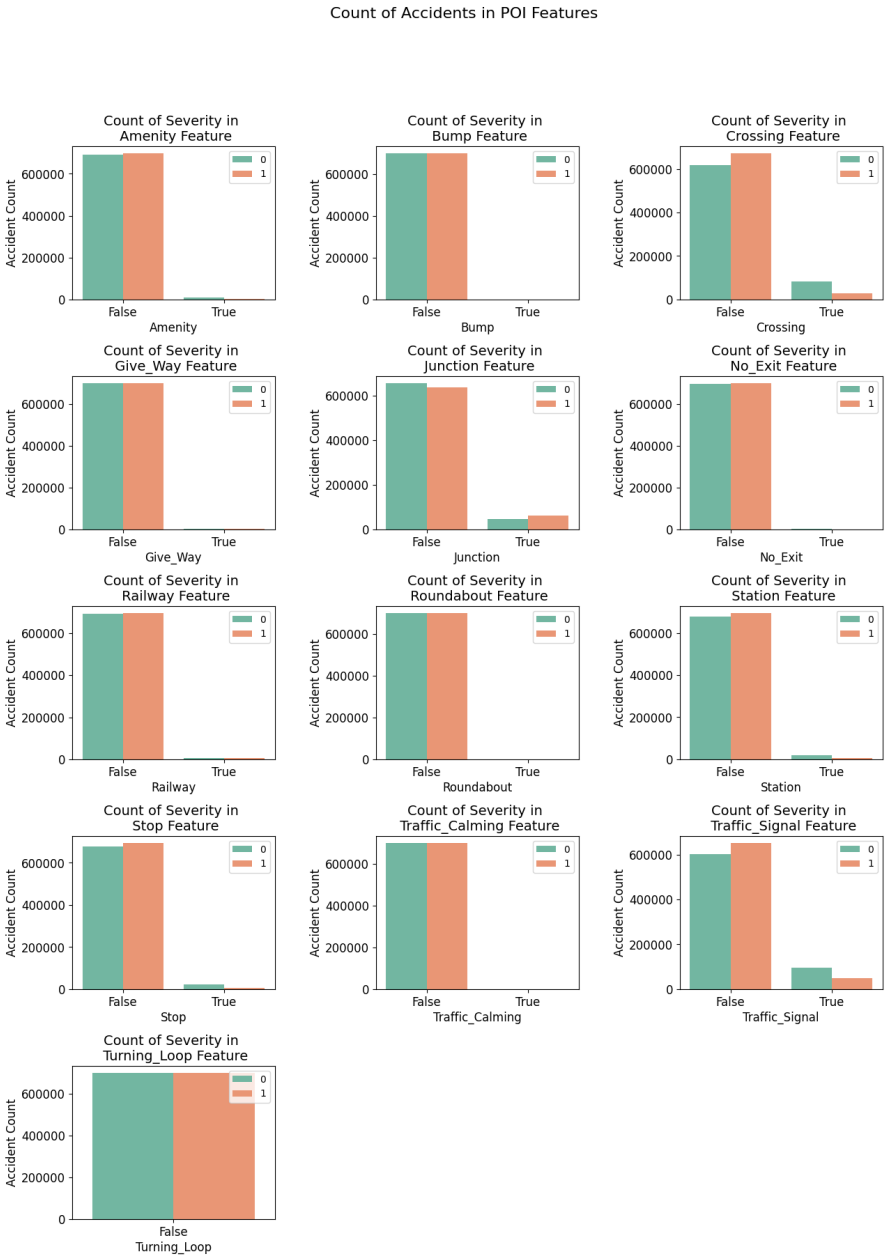
Sau bước lọc này, tập dữ liệu tiếp tục được thu gọn, tập trung vào các điều kiện thời tiết phổ biến hơn.

5.3.5 Phân tích theo Đặc điểm Giao thông và Tiện ích lân cận (POI)

Phân tích sự ảnh hưởng của các đặc điểm tại vị trí xảy ra tai nạn như sự hiện diện của các tiện ích (`Amenity`), gờ giảm tốc (`Bump`), giao cắt (`Crossing`), biển nhường đường (`Give_Way`), giao lộ (`Junction`), v.v. đến mức độ nghiêm trọng.

```
POI_features =
 ↪  ['Amenity','Bump','Crossing','Give_Way','Junction','No_Exit',
'Railway','Roundabout','Station','Stop','Traffic_Calming','Traffic_Signal',
'Turning_Loop']

fig, axs = plt.subplots(ncols=3, nrows=5, figsize=(15, 20))
plt.subplots_adjust(hspace=0.5,wspace = 0.5)

axs = axs.flatten()

for i, feature in enumerate(POI_features):

    sns.countplot(ax=axs[i], x=feature, hue='Severity', data=re_df,
     ↪  palette="Set2")

    axs[i].set_xlabel(f'{feature}', size=12, labelpad=3)
    axs[i].set_ylabel('Accident Count', size=12, labelpad=3)
    axs[i].tick_params(axis='x', labelsize=12)
    axs[i].tick_params(axis='y', labelsize=12)
    axs[i].legend(['0', '1'], loc='upper right', prop={'size': 10})
    axs[i].set_title(f'Count of Severity in \n {feature} Feature',
     ↪  size=14)

```

```
# Hide unused subplots
for j in range(len(POI_features), len(axs)):
    fig.delaxes(axs[j])

fig.suptitle('Count of Accidents in POI Features', fontsize=16)
plt.show()
```



Hình 17. Số lượng tai nạn theo Đặc điểm Giao thông/Tiện ích (POI) và Mức độ nghiêm trọng

Quan sát Hình 17:

- Phần lớn các vụ tai nạn xảy ra tại những vị trí **không** có các đặc điểm POI được liệt kê (giá trị `False` chiếm đa số). Điều này là hợp lý vì các đặc điểm như `Bump`, `Give_Way`, `Roundabout`, `Station`, `Stop`, `Traffic_Calming`, `Turning_Loop` thường không phổ biến trên toàn bộ mạng lưới đường.
- Các đặc điểm `Crossing` (Nơi giao cắt), `Junction` (Giao lộ), và `Traffic_Signal` (Đèn tín hiệu) là những nơi có ghi nhận số vụ tai nạn đáng kể khi có sự hiện diện của chúng (giá trị `True`).
- Đối với hầu hết các đặc điểm POI, tỷ lệ tai nạn nghiêm trọng (`Severity=1`) có vẻ không thay đổi đáng kể giữa việc có (`True`) hay không có (`False`) đặc điểm đó.
- Một số đặc điểm như `Amenity`, `Bump`, `Give_Way`, `No_Exit`, `Railway`, `Roundabout`, `Traffic_Calming`, `Turning_Loop` xuất hiện rất hiếm tại các vị trí tai nạn (cột `True` rất thấp). Do tính hiếm và ít có khả năng đóng góp vào mô hình dự đoán, các cột này sẽ được loại bỏ.

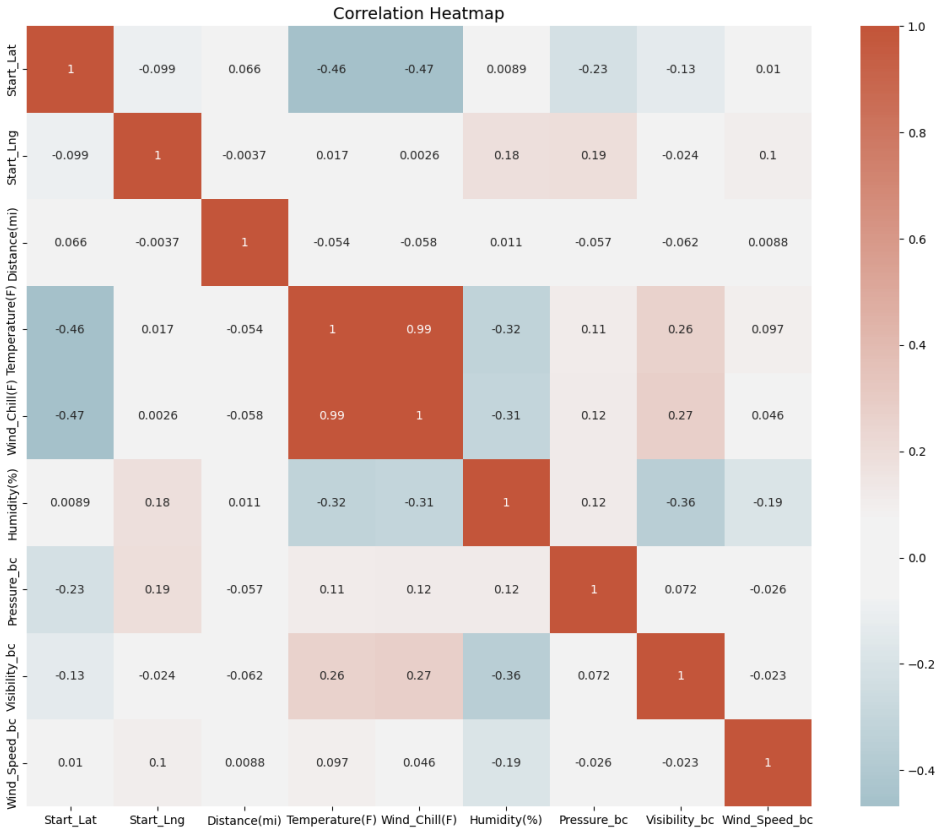
```
data = data.drop(['Amenity','Bump','Give_Way',"No_Exit","Railway",
                  "Roundabout","Traffic_Calming","Turning_Loop"], axis=1)
```

Sau khi loại bỏ các cột POI ít phổ biến, tập dữ liệu được thu gọn hơn.

5.3.6 Phân tích Tương quan giữa các Biến Số

Để kiểm tra mối quan hệ tuyến tính giữa các biến số liên tục và phát hiện các đặc trưng dư thừa tiềm ẩn, ma trận tương quan được tính toán và trực quan hóa bằng biểu đồ nhiệt (heatmap). Các biến số bao gồm tọa độ, khoảng cách, và các đặc trưng thời tiết số (bao gồm cả các biến đã được biến đổi Box-Cox).

```
continous_data =
↪  data[["Start_Lat","Start_Lng","Distance(mi)","Temperature(F)",
"Wind_Chill(F)","Humidity(%)","Pressure_bc","Visibility_bc","Wind_Speed_bc"
]]

data['Severity'] = data['Severity'].astype(int)
plt.figure(figsize=(15,12))
cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)
sns.heatmap(continous_data.corr(), annot=True,cmap=cmap,
↪  center=0).set_title("Correlation Heatmap", fontsize=14)
plt.show()
```



Hình 18. Biểu đồ nhiệt thể hiện ma trận tương quan giữa các biến số liên tục

Quan sát Hình 18:

- Mối tương quan mạnh nhất được tìm thấy giữa `Temperature(F)` và `Wind_Chill(F)` (hệ số tương quan r = 0.99). Điều này là dễ hiểu vì nhiệt độ cảm nhận (Wind Chill) được tính toán trực tiếp từ nhiệt độ không khí và tốc độ gió. Mối tương quan rất cao này cho thấy sự dư thừa thông tin lớn giữa hai biến.
- Giữa `Temperature(F)` và `Humidity(%)` có tương quan âm vừa phải (r = -0.32). Khi nhiệt độ tăng, độ ẩm có xu hướng giảm.
- Giữa `Temperature(F)` và `Start_Lat` (Vĩ độ) có tương quan âm đáng kể (r = -0.46), cho thấy nhiệt độ có xu hướng thấp hơn ở các vĩ độ cao hơn (phía Bắc).
- `Pressure_bc` (Áp suất đã biến đổi) có tương quan dương với `Start_Lat` và `Start_Lng`.
- `Visibility_bc` (Tầm nhìn đã biến đổi) có tương quan âm với `Humidity(%)` (r = -0.36), nghĩa là độ ẩm cao thường đi kèm với tầm nhìn giảm.
- Các mối tương quan khác giữa các biến còn lại nhìn chung là yếu.

Do mối tương quan cực kỳ cao giữa `Temperature(F)` và `Wind_Chill(F)`, việc giữ lại cả hai biến có thể gây ra vấn đề đa cộng tuyến cho một số mô hình. Biến `Wind_Chill(F)` bị loại bỏ để giảm thiểu sự dư thừa này.

```
data = data.drop(['Wind_Chill(F)'], axis=1)
```

5.4 Chuẩn hóa và Mã hóa Dữ liệu cho Mô hình

Trước khi đưa dữ liệu vào các mô hình máy học, các bước chuẩn bị cuối cùng được thực hiện:
1. **Mã hóa Biến Phân loại** Các biến phân loại dạng chữ còn lại ('State') được chuyển đổi thành dạng số bằng phương pháp Label Encoding. Trong khi đó, các biến có số lượng nhãn nhỏ hơn như 'Timezone', 'Wind_Direction' và 'Weather_Condition' được xử lý bằng phương pháp One-hot Encoding để đảm bảo mô hình không hiểu nhầm mối quan hệ thứ tự giữa các nhãn, đồng thời cải thiện khả năng học và dự đoán của thuật toán.
2. **Chuẩn hóa Biến Số (Standard Scaling):** Tất cả các biến số (bao gồm cả các biến đã được biến đổi Box-Cox và các đặc trưng thời gian) được đưa về cùng một thang đo với giá trị trung bình bằng 0 và độ lệch chuẩn bằng 1 bằng phương pháp StandardScaler. Điều này giúp các thuật toán nhạy cảm với thang đo hoạt động hiệu quả hơn.
3. **Phân chia Tập dữ liệu:** Dữ liệu được chia thành tập huấn luyện (70%) và tập kiểm thử (30%) bằng cách sử dụng hàm `train_test_split`. Tham số `stratify=y` được sử dụng để đảm bảo tỷ lệ giữa hai lớp nghiêm trọng trong tập huấn luyện và tập kiểm thử là tương đương nhau, giữ nguyên tính mất cân bằng của dữ liệu gốc trong quá trình phân chia. Phép chuẩn hóa (StandardScaler) được fit trên tập huấn luyện và áp dụng cho cả tập huấn luyện và tập kiểm thử để tránh rò rỉ thông tin từ tập kiểm thử vào quá trình huấn luyện.

```
# Convert to boolean columns and merge some columns with same meaning
↪  into one
data['Is_Complex_Road'] = data[['Junction',
↪  'Crossing']].sum(axis=1).apply(lambda x: True if x > 0 else False)
data = data.drop(['Junction','Crossing'],axis=1)
data['Sunrise_Sunset'] = data['Sunrise_Sunset'].apply(lambda x: True if x
↪  == 'Day' else False)

# Label Encoding for categorical features
encoder = LabelEncoder()

categories_features = ['State']

for feature in categories_features:
  data[feature] = encoder.fit_transform(data[feature])

data = pd.get_dummies(data, columns=['Weather_Condition',
↪  'Wind_Direction', 'Timezone'], drop_first=True)
data.columns = data.columns.str.replace('/', '_')

# Convert to all to float type
int_cols = data.select_dtypes(include=['int32', 'int64']).columns
data[int_cols] = data[int_cols].astype('float64')

# Split features (X) and target (y)
X = data.drop('Severity', axis=1)
y = data['Severity']

```

```python
# Initial Split into Training and Testing sets (before scaling within pipeline)
# Stratify ensures proportion of target variable is maintained
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42, stratify=y)

# Standard Scaling (Fit on Train, Transform on Train & Test)
numerical_cols_mlp = ['Start_Lat', 'Start_Lng', 'Distance(mi)', 'Temperature(F)', 'Humidity(\%)', 'Year', 'Month', 'Day', 'Hour', 'Week', 'Pressure_bc', 'Visibility_bc', 'Wind_Speed_bc'] # Kiểm tra lại


scaler = StandardScaler()
X_train[numerical_cols_mlp] = scaler.fit_transform(X_train[numerical_cols_mlp])
X_test[numerical_cols_mlp] = scaler.transform(X_test[numerical_cols_mlp])
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print(f"Shape of X_train: {X_train.shape}")
print(f"Shape of X_test: {X_test.shape}")
```

Dữ liệu sau các bước này đã sẵn sàng để được sử dụng cho việc huấn luyện và đánh giá các mô hình phân loại.

5.5 Áp dụng Mô hình

Nhiều mô hình phân loại khác nhau được áp dụng trên tập dữ liệu đã chuẩn bị để dự đoán mức độ nghiêm trọng của tai nạn. Các mô hình bao gồm:

- **Logistic Regression:** Mô hình hồi quy tuyến tính cơ bản cho bài toán phân loại. Tham số `class_weight='balanced'` được sử dụng để xử lý mất cân bằng dữ liệu.
- **Naive Bayes (GaussianNB):** Mô hình dựa trên định lý Bayes với giả định về tính độc lập của các đặc trưng. Phù hợp với dữ liệu số liên tục sau khi chuẩn hóa.
- **Decision Tree:** Mô hình cây quyết định. Tham số `max_depth=10` được dùng để giới hạn độ sâu của cây, tránh overfitting, và `class_weight='balanced'` để xử lý mất cân bằng.
- **Random Forest:** Mô hình học tập tập hợp dựa trên nhiều cây quyết định. Các tham số tương tự Decision Tree được sử dụng (`max_depth=10`, `class_weight='balanced'`), với `n_estimators=10` (số lượng cây thấp để giảm thời gian huấn luyện ban đầu).
- **XGBoost (Extreme Gradient Boosting):** Mô hình học tăng cường (boosting) hiệu quả cao. Tham số `scale_pos_weight` được tính toán và sử dụng để xử lý mất cân bằng dữ liệu (tỷ lệ giữa lớp âm và lớp dương). Các tham số khác như `n_estimators` và `max_depth` được đặt ở mức cơ bản.
- **LightGBM (Light Gradient Boosting Machine):** Một mô hình boosting khác, thường nhanh hơn XGBoost. Tương tự, `scale_pos_weight` được sử dụng.

- **LightGBM với SMOTE:** Thử nghiệm kết hợp LightGBM với kỹ thuật Oversampling SMOTE (Synthetic Minority Over-sampling Technique) thông qua `imblearn.pipeline` để xử lý mất cân bằng thay vì dùng `scale_pos_weight`.
- **LightGBM với Hyperparameter Tuning:** Áp dụng kỹ thuật tối ưu hóa siêu tham số cho LightGBM nhằm nâng cao hiệu suất mô hình, sử dụng RandomizedSearchCV để tìm kiếm các thiết lập tham số phù hợp nhất cho bài toán phân loại mất cân bằng.
- **MLP (Multi-Layer Perceptron):** Mô hình mạng nơ-ron nhân tạo cơ bản sử dụng Keras. Mô hình này xử lý riêng các biến số và biến phân loại (thông qua Embedding layer). Class weights được tính và áp dụng để xử lý mất cân bằng. Kỹ thuật Early Stopping được dùng để tránh overfitting và tìm số epoch huấn luyện tối ưu dựa trên chỉ số recall trên tập validation.

Một hàm trợ giúp `getResult` được định nghĩa để tính toán và in ra các chỉ số đánh giá phổ biến: Accuracy, Precision, Recall, F1-score và Ma trận nhầm lẫn (Confusion Matrix).

```
# Function to evaluate models
def getResult(y_test,y_pred):
  accuracy = accuracy_score(y_test, y_pred)
  precision = precision_score(y_test, y_pred)
  recall = recall_score(y_test, y_pred)
  f1 = f1_score(y_test, y_pred)
  print(f'Accuracy: {accuracy:.4f}')
  print(f'Precision: {precision:.4f}')
  print(f'Recall: {recall:.4f}')
  print(f'F1 score: {f1:.4f}')
  print('Confusion matrix:')
  print(confusion_matrix(y_test, y_pred))
  return [accuracy,precision,recall,f1]

# Calculate scale_pos_weight for XGBoost/LGBM
neg_count = np.sum(y_train == 0)
pos_count = np.sum(y_train == 1)
scale_pos_weight_value = neg_count / pos_count if pos_count > 0 else 1
print(f"Calculated scale_pos_weight: {scale_pos_weight_value:.2f}")

# --- Model Training and Evaluation Code ---
# (Logistic Regression)
lr = LogisticRegression(random_state=0, class_weight='balanced',
↪  max_iter=1000)
lr.fit(X_train,y_train)
y_pred_lr=lr.predict(X_test)
lr_result = ['Logistic Regression'] + getResult(y_test,y_pred_lr)

# (Naive Bayes)
NB = GaussianNB()
NB.fit(X_train, y_train)
y_pred_nb = NB.predict(X_test)
nb_result = ['Naive Bayes'] + getResult(y_test,y_pred_nb)

# (Decision Tree)
```

```
dstree = DecisionTreeClassifier(criterion='gini', max_depth=10, min_samples_split=2, random_state=42, class_weight='balanced')
dstree.fit(X_train, y_train)
y_pred_dt = dstree.predict(X_test)
dstree_result = ['Decision Tree'] + getResult(y_test,y_pred_dt)

# (Random Forest)
rf = RandomForestClassifier(criterion='gini', max_depth=10, min_samples_split=2, n_estimators = 10, random_state=42, class_weight='balanced')
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rf_result = ['Random Forest'] + getResult(y_test,y_pred_rf)

# (XGBoost)
xgb = XGBClassifier(random_state=42, scale_pos_weight=scale_pos_weight_value,
                    n_estimators=100, max_depth=5, eval_metric='logloss')
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
xgb_result = ['XGBoost'] + getResult(y_test, y_pred_xgb)

# (LightGBM)
lgbm = LGBMClassifier(random_state=42, scale_pos_weight=scale_pos_weight_value,
                      n_estimators=100, max_depth=5)
lgbm.fit(X_train, y_train)
y_pred_lgbm = lgbm.predict(X_test)
lgbm_result = ['LightGBM'] + getResult(y_test, y_pred_lgbm)

# (LightGBM with SMOTE)
smote_lgbm_pipeline = ImbPipeline([
    ('smote', SMOTE(random_state=42)),
    ('lgbm', LGBMClassifier(random_state=42, n_estimators=100, max_depth=5))
])
smote_lgbm_pipeline.fit(X_train, y_train) # Fit pipeline on original train data
y_pred_smote_lgbm = smote_lgbm_pipeline.predict(X_test)
smote_lgbm_result = ['LGBM (SMOTE)'] + getResult(y_test, y_pred_smote_lgbm)

# (MLP with Keras)
# Code for preparing MLP inputs (LabelEncoding, Scaling for MLP)
# ... (Code from the prompt) ...
# Code for building the Keras model
# ... (Code from the prompt) ...
# Code for compiling the model
# ... (Code from the prompt) ...
# Define EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_recall', patience=5, mode='max', restore_best_weights=True)
```

```
# Code for training the model with early stopping
# history = model_mlp.fit(...)
# Code for evaluating the MLP model
y_pred_proba_mlp = model_mlp.predict(X_test_list)
y_pred_mlp = (y_pred_proba_mlp > 0.5).astype(int)
mlp_result = ['MLP (Keras, ES)'] + getResult(y_test_mlp, y_pred_mlp) #
↪  Use y_test_mlp here
```

Do giới hạn về tài nguyên và thời gian, việc tinh chỉnh siêu tham số (hyperparameter tuning) chi tiết cho tất cả các mô hình không được thực hiện trong khuôn khổ báo cáo này, ngoại trừ một thử nghiệm với LightGBM. Các mô hình được huấn luyện với các tham số cơ bản hoặc kinh nghiệm phổ biến.

6 Đánh giá mô hình

Sau quá trình huấn luyện, các mô hình phân loại được đặt lên bàn cân để đánh giá hiệu năng trên tập dữ liệu kiểm thử độc lập. Mục tiêu không chỉ là tìm ra mô hình có độ chính xác (Accuracy) tổng thể cao nhất, mà còn phải đặc biệt chú trọng đến khả năng nhận diện đúng các vụ tai nạn nghiêm trọng (`Severity=1`), vốn là đối tượng quan tâm chính của bài toán.

Trong bối cảnh an toàn giao thông, việc bỏ sót một vụ tai nạn nghiêm trọng (False Negative - FN) có thể dẫn đến hậu quả khôn lường, nên việc tối đa hóa tỷ lệ phát hiện các trường hợp này là cực kỳ quan trọng. Do đó, chỉ số **Recall** (độ nhạy) – đo lường tỷ lệ các vụ nghiêm trọng thực tế được mô hình xác định đúng – trở thành một ưu tiên hàng đầu. Bên cạnh đó, **F1-score**, là trung bình điều hòa giữa Precision (độ chính xác của các dự đoán "nghiêm trọng") và Recall, cung cấp một thước đo cân bằng hơn về hiệu quả tổng thể trong việc xác định lớp thiểu số quan trọng này.

```
all_results_data = [
    lr_result, nb_result, dstree_result, rf_result,
    xgb_result, lgbm_result, smote_lgbm_result, mlp_result
]
final_results = pd.DataFrame(data=all_results_data,
columns=['Model','Accuracy','Precision','Recall','F1'])

print("\n--- Final Comparison Results ---")
display(final_results.sort_values(by='F1', ascending=False))
```

Bảng 1: Bảng tổng hợp hiệu năng các mô hình trên tập kiểm thử

Model	Accuracy	Precision	Recall	F1
XGBoost	0.8159	0.4118	0.8351	**0.5516**
LGBM (Balanced, Tuned)	0.8147	0.4102	**0.8383**	0.5508
LGBM (SMOTE)	**0.8600**	**0.4868**	0.5999	0.5375
LightGBM	0.7922	0.3783	0.8278	0.5193
Decision Tree	0.7892	0.3747	0.8294	0.5161
Random Forest	0.7785	0.3585	0.8029	0.4957
MLP (Keras, ES)	0.7719	0.3537	0.8248	0.4950
Logistic Regression	0.6772	0.2461	0.6691	0.3599
Naive Bayes	0.7951	0.2666	0.2921	0.2788

Phân tích sâu hơn kết quả trong Bảng 1 hé lộ những điểm đáng chú ý:

- **Nhóm dẫn đầu về khả năng phát hiện (Recall cao):** Các mô hình dựa trên cây quyết định và boosting, khi được điều chỉnh để xử lý mất cân bằng dữ liệu bằng trọng số lớp (`scale_pos_weight` hoặc `class_weight='balanced'`), thể hiện khả năng vượt trội trong việc "bắt"các vụ tai nạn nghiêm trọng. **XGBoost**, **LightGBM**, **Decision Tree**, **Random Forest**, và thậm chí cả **MLP** đều đạt được mức Recall trên 82%. Điều này cho thấy các mô hình này rất nhạy cảm với lớp thiểu số quan trọng. Tuy nhiên, cái giá phải trả là Precision tương đối thấp (dưới 42%), nghĩa là một tỷ lệ đáng kể các vụ được dự đoán là nghiêm trọng thực chất lại không phải.
- **Nhóm tối ưu cân bằng (F1-score cao): XGBoost** vươn lên dẫn đầu về chỉ số F1-score (0.5523), cho thấy sự cân bằng tốt nhất giữa Precision và Recall trong nhóm Recall cao. Nó

không chỉ phát hiện được nhiều ca nghiêm trọng (Recall cao nhất) mà còn duy trì được độ chính xác dự đoán dương tính (Precision) tốt hơn một chút so với LightGBM và Decision Tree (không SMOTE).

- **Trường hợp đặc biệt - LGBM (SMOTE):** Kỹ thuật Oversampling SMOTE giúp **LGBM (SMOTE)** đạt được Accuracy tổng thể cao nhất (85.9%) và Precision tốt nhất (48.5%). Tuy nhiên, Recall lại giảm xuống chỉ còn 63.3%, thấp hơn đáng kể so với việc dùng trọng số lớp. Điều này minh chứng cho sự đánh đổi kinh điển: SMOTE giúp mô hình tự tin hơn khi đưa ra dự đoán "nghiêm trọng"(ít FP hơn), nhưng đồng thời làm tăng nguy cơ bỏ sót các trường hợp nghiêm trọng thực sự (nhiều FN hơn).
- **Nhóm cơ bản (Baseline Models): Logistic Regression** và **Naive Bayes** gặp nhiều khó khăn với bài toán này. Mặc dù Logistic Regression (với `class_weight='balanced'`) cố gắng cải thiện Recall (67.3%), nhưng cả Precision và F1-score đều rất thấp. Naive Bayes thậm chí còn kém hơn về Recall. Kết quả này cho thấy các giả định tuyến tính hoặc độc lập của chúng không đủ để nắm bắt các mối quan hệ phức tạp trong dữ liệu tai nạn giao thông.
- **MLP và Hạn chế:** Mạng nơ-ron **MLP** với cấu trúc hiện tại cho thấy khả năng học hỏi tốt (Recall cao), nhưng chưa thể vượt qua các mô hình boosting về độ cân bằng Precision-Recall (F1-score). Có thể cần tinh chỉnh kiến trúc mạng, tối ưu siêu tham số kỹ lưỡng hơn hoặc huấn luyện lâu hơn để khai thác hết tiềm năng.

Kết luận lựa chọn mô hình:

Dựa trên yêu cầu ưu tiên phát hiện tối đa các vụ tai nạn nghiêm trọng (Recall cao) trong khi vẫn duy trì sự cân bằng hợp lý với độ chính xác dự đoán (F1-score), mô hình **XGBoost** (với `scale_pos_weight`) được xác định là lựa chọn phù hợp nhất tại thời điểm này. Mô hình này đạt được Recall cao nhất và F1-score cao nhất, thể hiện hiệu năng tổng thể tốt nhất cho mục tiêu đề ra. Mặc dù LGBM (SMOTE) có Accuracy và Precision cao hơn, nhưng sự sụt giảm đáng kể về Recall khiến nó trở nên rủi ro hơn cho bài toán thực tế.

Cần lưu ý rằng các kết quả này dựa trên cấu hình tham số ban đầu. Việc tinh chỉnh siêu tham số (hyperparameter tuning) sâu hơn cho XGBoost, LightGBM, hoặc thậm chí MLP có thể dẫn đến những cải thiện đáng kể hơn nữa về hiệu năng.

7 Tổng kết

Dự án này tập trung vào việc xây dựng và đánh giá các mô hình học máy nhằm dự đoán mức độ nghiêm trọng của tai nạn giao thông tại Hoa Kỳ, sử dụng tập dữ liệu công khai US Accidents. Quá trình thực hiện bao gồm các giai đoạn chính: tiền xử lý dữ liệu, phân tích dữ liệu khám phá (EDA), xây dựng mô hình, và đánh giá hiệu năng.

Giai đoạn tiền xử lý dữ liệu đóng vai trò quan trọng trong việc làm sạch và chuẩn bị dữ liệu thô. Các bước chính bao gồm lựa chọn đặc trưng phù hợp, xử lý một khối lượng lớn dữ liệu thiếu (missing values) thông qua việc loại bỏ các cột có tỷ lệ thiếu quá cao (`Precipitation(in)`) và loại bỏ các dòng thiếu ở các cột quan trọng khác. Dữ liệu thời gian được xử lý để trích xuất các đặc trưng có ý nghĩa như năm, tháng, giờ, ngày trong tuần. Các biến phân loại có số lượng hạng mục lớn (`Wind_Direction`, `Weather_Condition`) được đơn giản hóa thông qua việc gom nhóm hợp lý. Các đặc trưng dư thừa hoặc ít thông tin, như một số đặc điểm POI hiếm gặp và các cột twilight, cũng như `Wind_Chill(F)` (do tương quan cao với nhiệt độ), đã được loại bỏ để tập trung vào các yếu tố có khả năng ảnh hưởng lớn hơn. Sau tiền xử lý, tập dữ liệu cuối cùng bao gồm hơn 4.9 triệu bản ghi với 24 đặc trưng.

Phân tích dữ liệu khám phá đã cung cấp nhiều hiểu biết giá trị. Đáng chú ý nhất là sự mất cân bằng nghiêm trọng của biến mục tiêu `Severity`, dẫn đến quyết định chuyển bài toán thành phân loại nhị phân (Ít nghiêm trọng vs. Nghiêm trọng). Phân tích theo thời gian chỉ ra các đỉnh điểm tai nạn trùng với giờ cao điểm và sự khác biệt về tần suất giữa các ngày trong tuần và các tháng trong năm. Phân tích không gian cho thấy sự tập trung tai nạn ở một số bang nhất định và dọc theo các trục giao thông chính, với sự phân bố hơi khác biệt của các vụ tai nạn nghiêm trọng. Phân tích thời tiết nhấn mạnh vai trò của các điều kiện như mưa, tuyết, và tầm nhìn.

Để giải quyết bài toán phân loại nhị phân trên dữ liệu mất cân bằng, nhiều thuật toán học máy đã được triển khai và so sánh, từ các mô hình cơ bản (Logistic Regression, Naive Bayes) đến các mô hình cây (Decision Tree, Random Forest), các mô hình boosting mạnh mẽ (XGBoost, LightGBM) và mạng nơ-ron nhân tạo (MLP). Các kỹ thuật xử lý mất cân bằng như cân bằng trọng số lớp (class weighting/scale_pos_weight) và lấy mẫu lại (SMOTE) đã được áp dụng. Dữ liệu được mã hóa (Label Encoding) và chuẩn hóa (StandardScaler) phù hợp trước khi huấn luyện.

Việc đánh giá mô hình tập trung vào các chỉ số Recall và F1-score do tầm quan trọng của việc phát hiện các vụ tai nạn nghiêm trọng. Kết quả cho thấy các mô hình boosting, đặc biệt là XGBoost sử dụng trọng số lớp, đạt hiệu năng tốt nhất với F1-score là 0.5523 và Recall lên đến 0.8378. LightGBM và Decision Tree cũng thể hiện khả năng phát hiện tốt (Recall cao) khi sử dụng trọng số lớp. Kỹ thuật SMOTE giúp cải thiện Accuracy và Precision cho LightGBM nhưng lại làm giảm đáng kể Recall. Các mô hình cơ bản và MLP (với cấu hình hiện tại) cho thấy hiệu năng kém hơn trong bài toán này.

Tóm lại, dự án đã thực hiện thành công quy trình xây dựng mô hình dự đoán mức độ nghiêm trọng tai nạn giao thông, từ xử lý dữ liệu đến đánh giá mô hình. XGBoost được xác định là mô hình tiềm năng nhất dựa trên các tiêu chí đánh giá đã đề ra. Tuy nhiên, cần lưu ý rằng việc tinh chỉnh siêu tham số sâu hơn và khám phá các kiến trúc mô hình phức tạp hơn (như các mạng nơ-ron sâu hơn hoặc các kỹ thuật ensemble tiên tiến) có thể giúp cải thiện hơn nữa hiệu năng dự đoán. Đồng thời, việc bổ sung thêm các nguồn dữ liệu khác (ví dụ: dữ liệu mật độ giao thông thời gian thực, đặc điểm chi tiết về đường sá) cũng là một hướng phát triển hứa hẹn cho các nghiên cứu trong tương lai.

8 Tầm nhìn chiến lược & kế hoạch mở rộng

Kết quả đạt được trong dự án này đã đặt nền móng cho việc ứng dụng học máy vào lĩnh vực dự đoán và giảm thiểu rủi ro tai nạn giao thông. Mô hình XGBoost cho thấy hiệu quả đáng kể, tuy nhiên, tiềm năng cải thiện và ứng dụng thực tế vẫn còn rất lớn. Phần này phác thảo các hướng phát triển tiếp theo.

8.1 Cải tiến Mô hình và Kỹ thuật

Các hướng tập trung vào việc nâng cao hiệu năng và độ sâu phân tích của mô hình hiện tại:

1. **Tối ưu hóa Siêu tham số:** Áp dụng các kỹ thuật tìm kiếm siêu tham số hệ thống (GridSearchCV, RandomizedSearchCV, Bayesian Optimization) cho các mô hình tiềm năng như XGBoost và LightGBM để tối ưu hóa các chỉ số Recall và F1-score.

2. **Kỹ thuật Đặc trưng Nâng cao:** Tạo thêm các đặc trưng mới có ý nghĩa hơn, ví dụ như đặc trưng tương tác giữa thời tiết và thời gian, khoảng cách đến POI gần nhất, hoặc sử dụng embedding phức tạp hơn cho các biến phân loại.

3. **Đánh giá và Diễn giải Sâu hơn:** Phân tích đường cong ROC, Precision-Recall, và sử dụng các công cụ diễn giải mô hình (SHAP, LIME) để hiểu rõ hơn về các yếu tố rủi ro và sự đánh đổi của mô hình. Xem xét các chỉ số nhạy cảm với chi phí nếu có thể định lượng.

4. **Mở rộng Bài toán Phân loại:** Thử nghiệm lại với bài toán phân loại đa lớp (4 mức độ nghiêm trọng ban đầu) để cung cấp thông tin chi tiết hơn, áp dụng các kỹ thuật xử lý mất cân bằng phù hợp cho đa lớp.

8.2 Hướng tới Ứng dụng Thực tế

Tầm nhìn dài hạn là đưa các mô hình dự đoán vào ứng dụng thực tế để mang lại lợi ích cụ thể:

1. **Hệ thống Cảnh báo Rủi ro Thời gian thực:** Phát triển hệ thống phân tích dữ liệu tức thời để cảnh báo các khu vực nguy cơ cao cho người lái xe và trung tâm điều hành giao thông.

2. **Tích hợp vào Hệ thống Giao thông Thông minh:** Đưa dự đoán rủi ro vào ứng dụng điều hướng (gợi ý lộ trình an toàn hơn) hoặc hệ thống quản lý tín hiệu giao thông (điều chỉnh chu kỳ đèn).

3. **Hỗ trợ Hoạch định Chính sách và Hạ tầng:** Cung cấp thông tin dựa trên dữ liệu giúp xác định các khu vực cần ưu tiên cải tạo hạ tầng, điều chỉnh giao thông, hoặc tăng cường giám sát.

4. **Công cụ Đánh giá Rủi ro Cá nhân hóa:** Xây dựng ứng dụng cho phép người dùng tự đánh giá rủi ro dựa trên lộ trình và điều kiện cụ thể.

Việc hiện thực hóa các kế hoạch này đòi hỏi nỗ lực liên tục trong việc cải thiện dữ liệu, thuật toán, và sự hợp tác giữa các bên liên quan, xem dự án hiện tại là bước khởi đầu quan trọng.

9 Tài liệu tham khảo

[1] Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, and Rajiv Ramnath. “A Countrywide Traffic Accident Dataset.”, 2019.

[2] Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, Radu Teodorescu, and Rajiv Ramnath. "Accident Risk Prediction based on Heterogeneous Sparse Data: New Dataset and Insights."In proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2019.

[3] Bảng dữ liệu: Tai nạn giao thông tại Mỹ (2016 - 2023).Nguồn: *Kaggle - US Accidents Dataset*