Traffic Accident Hotspot Detection and Severity Prediction Using NYC Crash and Weather Data

Pranav Patil (pbp86) Vishal Nagamalla (vn218)

CS 210

December 2025

Motivation and Goals

- NYC releases large crash datasets, but raw CSV analysis is hard to scale and reproduce.
- Weather is a meaningful contextual factor that may influence crash risk and severity.
- **Goal 1:** Build a clean PostgreSQL database with reliable ETL.
- **Goal 2:** Use SQL to identify spatial and temporal crash hotspots.
- **Goal 3:** Predict crash severity (minor vs. serious) using time, location, and weather.

Datasets

- **NYC Motor Vehicle Collisions** (`nyc_crashes.csv`)
  - One row per crash with date/time, borough/zip, latitude/longitude.
  - Counts of injuries and deaths used to define severity.
- **NYC Daily Weather** (`nyc_weather_daily.csv`)
  - Daily precipitation and temperature range.
  - Simple weather description when available.
- Crashes are linked to weather by matching on `crash_date`.

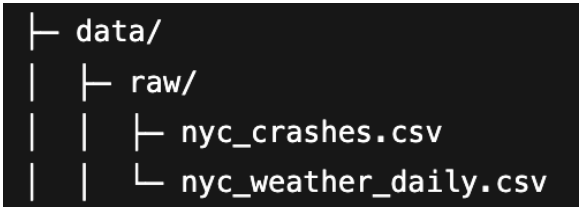


Figure: Data Location

End-to-End System Overview

- Raw CSVs (`nyc_crashes.csv`, `nyc_weather_daily.csv`) $\Rightarrow$ `etl.py` cleans and loads into PostgreSQL `traffic_db`.
- `schema.sql` defines tables, keys, and indexes.
- `sql_queries.sql` runs hotspot and severity summaries directly in SQL.
- `ml_pipeline.py` pulls features from the DB and trains:
  - Logistic Regression (baseline)
  - Random Forest (non-linear ensemble)
- Pipeline is fully reproducible on another machine with the same DB + CSVs.

Database Design

- **Tables**
  - `weather(weather_id, date, precipitation, temp_max, temp_min, ...)`
  - `accidents(accident_id, crash_datetime, borough, lat, lon, injuries, deaths, severity, weather_id)`
- **Integrity**
  - Primary keys on both tables.
  - Foreign key: `accidents.weather_id` $\rightarrow$ `weather.weather_id`.
- **Performance**
  - Indexes on time, severity, and location to speed hotspot queries.

ETL and Feature Engineering

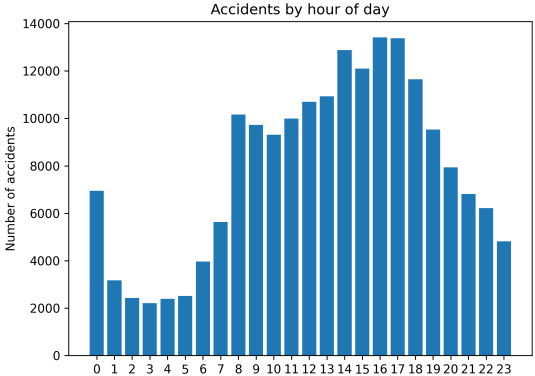
- Normalized columns and types from raw CSV inputs.
- Removed invalid rows (missing dates/times or missing coordinates).
- Built `crash_datetime` from `CRASH_DATE + CRASH_TIME`.
- Defined binary severity:

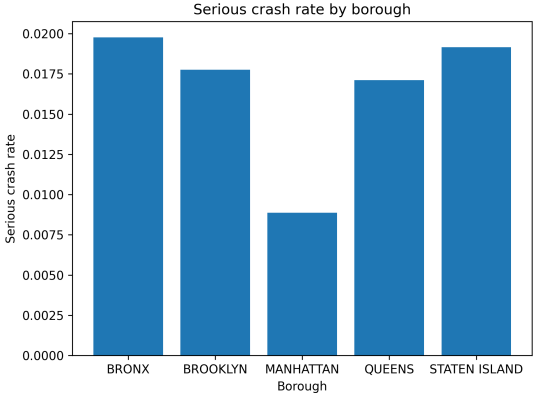
$$\texttt{severity} = 1 \text{ if deaths} > 0 \text{ or injuries} \geq 3, \quad \texttt{severity} = 0 \text{ otherwise.}$$

- Joined each crash to daily weather using `crash_date`.
- Derived ML features:
  - hour, day-of-week, month, weekend
  - precipitation, temp_max, temp_min

Hotspot Analysis (SQL + Plots)

- SQL in `sql_queries.sql`:
  - Top borough–hour combinations by total and serious crashes.
  - Severity distributions by borough and zip code.
- Visual summaries computed from query outputs:



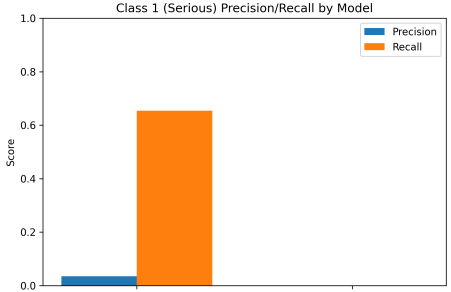


Machine Learning Setup

- Task: binary classification of crash severity (minor vs. serious).
- Models:
  - Logistic Regression (baseline)
  - Random Forest (non-linear ensemble)
- Features:
  - Time features from `crash_datetime`
  - Location features (lat/long, borough/zip when available)
  - Weather features (precipitation, temp_max/min)
- Evaluation:
  - Chronological train/test split
  - Precision, recall, F1, confusion matrix

ML Results Summary

- Strong class imbalance: serious crashes $\approx 2.8\%$ of test set.
- **Logistic Regression**
  - Class 1 (serious): precision ≈ 0.04, recall ≈ 0.65.
  - Catches many serious crashes but with many false positives.
- **Random Forest**
  - Very high accuracy on class 0 (minor).
  - Almost never predicts serious crashes (recall ≈ 0 for class 1).



Limitations, Future Work, and Contributions

Limitations

- Severe class imbalance reduces model effectiveness for class 1.
- Severity label is a simplified rule-based definition.

Future Work

- Class weights or resampling.
- Threshold tuning for better serious-crash detection.
- Richer spatial features (intersection clustering).

Contributions

- Pranav Patil: project definition, initial schema and ETL structure, baseline ML pipeline.
- Vishal Nagamalla: PostgreSQL setup, weather reintegration, ETL robustness, final runs and results capture.
- Both: joint debugging and GitHub-based integration.