Chicago Traffic Crash Analysis & Prediction

Ziqi Wei

1. Project Goal

The goal of this analysis is to use machine learning to predict whether a traffic crash in Chicago will result in an injury (Injury vs. No Injury). We will also identify the most important risk factors (e.g., weather, lighting, speed).

This analysis supports the "Vision Zero" dashboard by providing statistical evidence for road safety interventions.

2. Setup & Data Loading

```
library(tidyverse)
library(tidymodels)
library(vip)
library(xgboost)

set.seed(123)
# Load the cleaned data
crashes_clean <- readRDS("../data/crashes_clean.rds")

# Quick check
glimpse(crashes_clean)
```

```
Rows: 98,864
Columns: 19
$ crash_record_id         <chr> "c0700cfd730b83fd7561e4b853e2f886cdc3d7fa2e5ae~
$ crash_date              <dttm> 2023-12-01 17:30:00, 2024-07-02 14:30:00, 202~
$ crash_hour              <int> 17, 14, 16, 9, 20, 22, 17, 15, 9, 10, 22, 20, ~
$ day_of_week             <ord> Fri, Tue, Thu, Fri, Tue, Sun, Thu, Mon, Fri, S~
```

```
$ is_weekend              <chr> "Weekday", "Weekday", "Weekday", "Weekday", "W~
$ season                  <chr> "Winter", "Summer", "Summer", "Spring", "Summe~
$ latitude                <dbl> 41.95221, 41.88031, 41.85331, 41.85604, 41.702~
$ longitude               <dbl> -87.64949, -87.63512, -87.73471, -87.65448, -8~
$ weather_condition       <chr> "RAIN", "CLEAR", "CLEAR", "CLEAR", "CLEAR", "C~
$ lighting_condition      <chr> "DARKNESS, LIGHTED ROAD", "DAYLIGHT", "DAYLIGH~
$ roadway_surface_cond    <chr> "WET", "DRY", "DRY", "DRY", "DRY", "DRY", "DRY~
$ posted_speed_limit      <dbl> 30, 30, 30, 30, 30, 30, 30, 20, 35, 30, 20, 30~
$ prim_contributory_cause <chr> "UNABLE TO DETERMINE", "UNABLE TO DETERMINE", ~
$ community_area          <dbl> 6, 32, 29, 31, 51, 69, 8, 7, 41, 15, 43, 2, 24~
$ community_name          <chr> "Lake View", "Loop", "North Lawndale", "Lower ~
$ city_side               <chr> "North Side", "Central", "West Side", "West Si~
$ city_side_factor        <fct> North Side, Central, West Side, West Side, Far~
$ injuries_total          <dbl> 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 1, 0~
$ injury_category         <chr> "Injury", "Injury", "No Injury", "No Injury", ~
```

3. Feature Selection & Splitting

We will use the following features to predict `injury_category`:

- **Environmental:** `weather_condition`, `lighting_condition`, `roadway_surface_cond`
- **Temporal:** `crash_hour`, `season`, `is_weekend`
- **Infrastructure:** `posted_speed_limit`
- **Geographic:** `city_side`

```r
# Convert character strings to factors (required for modeling)
model_data <- crashes_clean %>%
  mutate(
    injury_category = as.factor(injury_category),
    weather_condition = as.factor(weather_condition),
    lighting_condition = as.factor(lighting_condition),
    roadway_surface_cond = as.factor(roadway_surface_cond),
    season = as.factor(season),
    is_weekend = as.factor(is_weekend),
    city_side = as.factor(city_side) # <--- NEW: Include City Side (Region)
  ) %>%
  drop_na()

# Split data:
# 75% for Training, 25% for Testing
```

```r
data_split <- initial_split(model_data, prop = 0.75, strata = injury_category)
train_data <- training(data_split)
test_data  <- testing(data_split)

print(paste("Training Set:", nrow(train_data), "rows"))
```

```
[1] "Training Set: 73818 rows"
```

```r
print(paste("Testing Set:", nrow(test_data), "rows"))
```

```
[1] "Testing Set: 24606 rows"
```

4. Build an XGBoost Model (Gradient Boosting)

I choose **XGBoost** for this analysis because it is highly effective at handling imbalanced datasets. Instead of generating synthetic data (which can increase noise), we use the `scale_pos_weight` parameter to explicitly tell the model that correctly predicting an "Injury" is significantly more important than predicting "No Injury". This helps balance the model's sensitivity without sacrificing too much accuracy.

```r
# Define the XGBoost model specification
xg_spec <- boost_tree(
  trees = 500,
  tree_depth = 8,
  learn_rate = 0.02,
  min_n = 5
) %>%
  set_engine("xgboost", scale_pos_weight = 2.0) %>%
  set_mode("classification")

# Define the recipe (Preprocessing with Interactions)
xg_recipe <- recipe(injury_category ~ posted_speed_limit + weather_condition +
                      lighting_condition + season + crash_hour + is_weekend + city_side,
                    data = train_data) %>%
  # Group infrequent weather/lighting categories into "Other" to reduce noise
  step_other(weather_condition, threshold = 0.05) %>%
  step_other(lighting_condition, threshold = 0.05) %>%
  # Convert all categorical factors (like city_side) into dummy variables (0/1)
  step_dummy(all_nominal_predictors()) %>%
```

```
  # Interaction term: Does bad weather + bad lighting equal extra risk?
  step_interact(terms = ~ starts_with("weather_condition"):starts_with("lighting_condition"))

# Create a Workflow
xg_workflow <- workflow() %>%
  add_model(xg_spec) %>%
  add_recipe(xg_recipe)

# Train the model
xg_fit <- xg_workflow %>% fit(data = train_data)

print("Model Training Complete!")
```

```
[1] "Model Training Complete!"
```

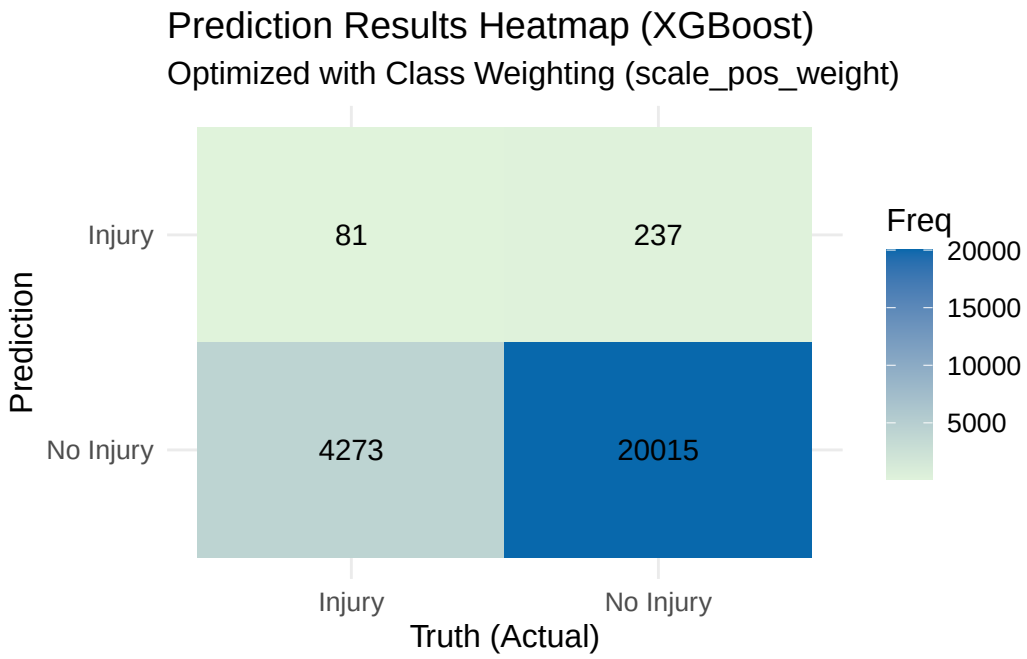
5. Evaluate Model Performance

```
# Make predictions on the Test Set
predictions <- xg_fit %>%
  predict(test_data) %>%
  bind_cols(test_data)

# Calculate Accuracy metrics
metrics_score <- predictions %>%
  metrics(truth = injury_category, estimate = .pred_class) %>%
  filter(.metric %in% c("accuracy", "kap"))

print(metrics_score)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.817
2 kap      binary         0.0108
```

```
# --- Visualization: Confusion Matrix Heatmap ---
conf_mat_result <- predictions %>%
  conf_mat(truth = injury_category, estimate = .pred_class)
```

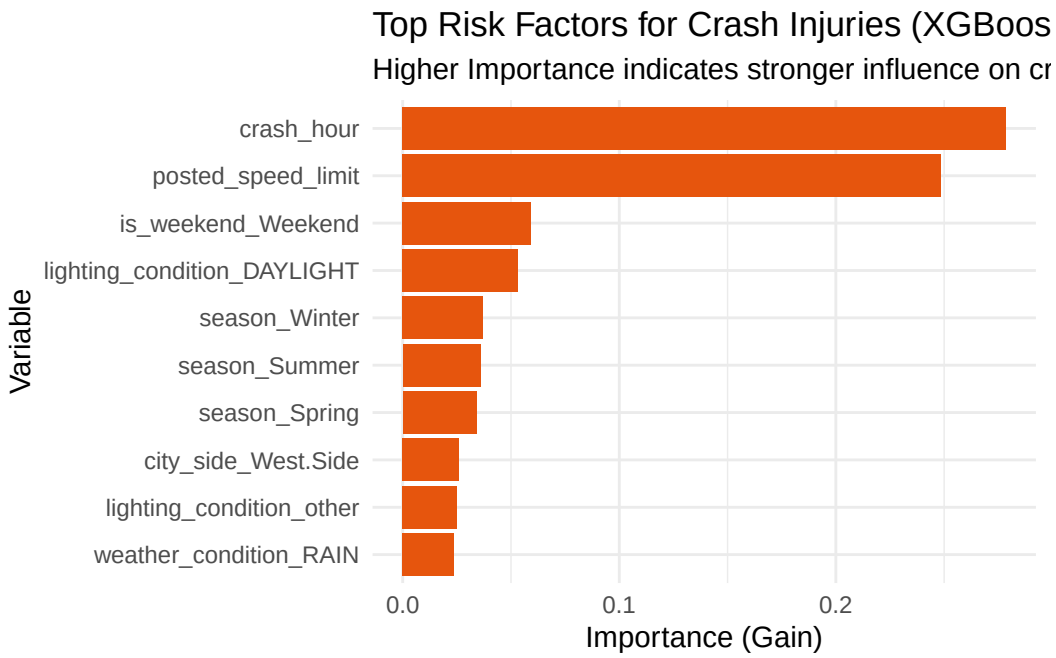
```r
# Plotting the Heatmap
heatmap_plot <- autoplot(conf_mat_result, type = "heatmap") +
  scale_fill_gradient(low = "#e0f3db", high = "#0868ac") +
  labs(
    title = "Prediction Results Heatmap (XGBoost)",
    subtitle = "Optimized with Class Weighting (scale_pos_weight)",
    x = "Truth (Actual)",
    y = "Prediction"
  ) +
  theme_minimal() +
  theme(text = element_text(size = 12))

print(heatmap_plot)
```



6. Feature Importance (Key Insights)

```r
xg_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 10, geom = "col", aesthetics = list(fill = "#e6550d")) +
```

```
  labs(
    title = "Top Risk Factors for Crash Injuries (XGBoost)",
    subtitle = "Higher Importance indicates stronger influence on crash severity",
    y = "Importance (Gain)",
    x = "Variable"
  ) +
  theme_minimal()
```




```
print(xg_fit)
```

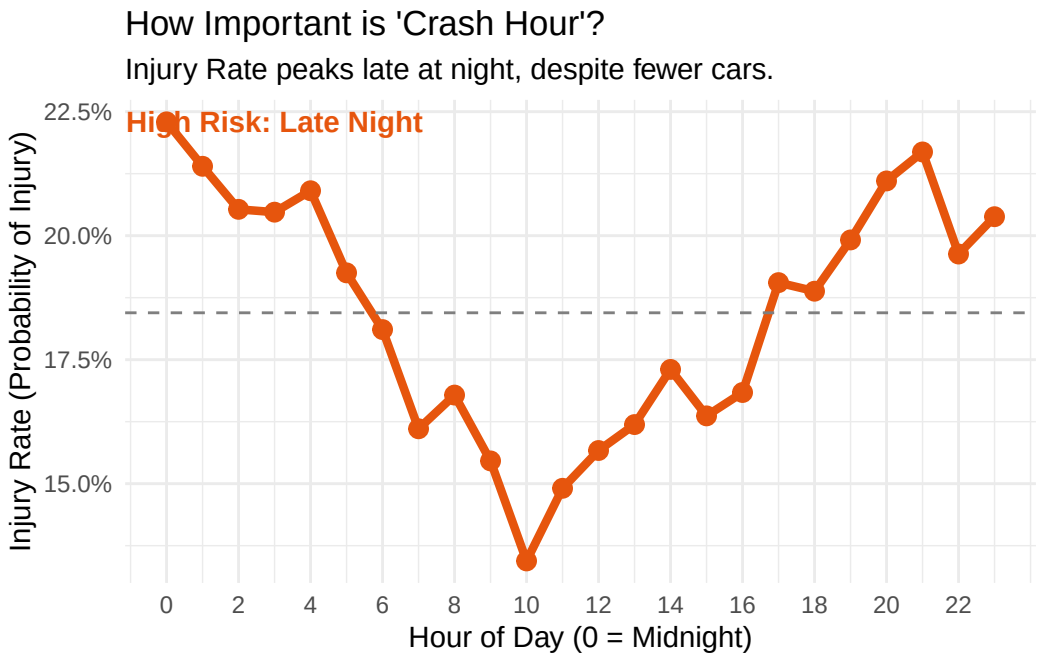
```
== Workflow [trained] ==========================================================
Preprocessor: Recipe
Model: boost_tree()

-- Preprocessor ----------------------------------------------------------------
4 Recipe Steps

* step_other()
* step_other()
* step_dummy()
* step_interact()
```

```
-- Model ---------------------------------------------------------------------
##### xgb.Booster
raw: 3.8 Mb
call:
  xgboost::xgb.train(params = list(eta = 0.02, max_depth = 8, gamma = 0,
    colsample_bytree = 1, colsample_bynode = 1, min_child_weight = 5,
    subsample = 1), data = x$data, nrounds = 500, watchlist = x$watchlist,
    verbose = 0, scale_pos_weight = 2, nthread = 1, objective = "binary:logistic")
params (as set within xgb.train):
  eta = "0.02", max_depth = "8", gamma = "0", colsample_bytree = "1", colsample_bynode = "1"
xgb.attributes:
  niter
callbacks:
  cb.evaluation.log()
# of features: 22
niter: 500
nfeatures : 22
evaluation_log:
  iter training_logloss
 <num>            <num>
     1        0.6876655
     2        0.6823646
   ---              ---
   499        0.4789096
   500        0.4789044
```

7. Deep Dive into crash_hour

```r
library(scales)
# Prepare plotting data
hourly_risk <- crashes_clean %>%
  filter(!is.na(crash_hour), !is.na(injury_category)) %>%
  group_by(crash_hour) %>%
  summarise(
    total_crashes = n(),
    injury_count = sum(injury_category == "Injury"),
    injury_rate = injury_count / total_crashes
  )

# Plot an intuitive line chart
```

```r
ggplot(hourly_risk, aes(x = crash_hour, y = injury_rate)) +
  # Plot the trend line
  geom_line(color = "#e6550d", size = 1.5) +
  geom_point(color = "#e6550d", size = 3) +

  # Add a dashed line: Average injury rate
  geom_hline(yintercept = mean(hourly_risk$injury_rate, na.rm = TRUE),
             linetype = "dashed", color = "gray50") +

  # Annotate key areas
  annotate("text", x = 3, y = max(hourly_risk$injury_rate, na.rm = TRUE),
           label = "High Risk: Late Night", color = "#e6550d", fontface = "bold") +

  # Customize the chart
  scale_x_continuous(breaks = seq(0, 23, 2)) +
  scale_y_continuous(labels = percent_format()) +
  labs(
    title = "How Important is 'Crash Hour'?",
    subtitle = "Injury Rate peaks late at night, despite fewer cars.",
    x = "Hour of Day (0 = Midnight)",
    y = "Injury Rate (Probability of Injury)"
  ) +
  theme_minimal()
```



8. Save Model for Shiny App

```
# We need to save the trained workflow so the App can use it to make live predictions.
if(!dir.exists("../models")) dir.create("../models")

# Save the XGBoost fit object
saveRDS(xg_fit, "../models/xgboost_model.rds")

print("Model successfully saved to models/xgboost_model.rds")
```

```
[1] "Model successfully saved to models/xgboost_model.rds"
```