The Role of Right-Turn-on-Red Traffic Lights in Pedestrian Accidents

TABLE OF CONTENTS

The Role of Right-Turn-on-Red Traffic Lights in Pedestrian Accidents

By Onyeukwu Emeka

PROJECT BACKGROUND

Since the 1970s, many cities in America have allowed cars to turn right at red lights if they stop first. This can help traffic move faster, but it can also be dangerous for people walking on the streets. Sometimes, drivers don't see or stop for people in crosswalks, which can lead to accidents.

San Francisco is a good example to study because lots of people walk there, and there is data to help understand the problem. In 2020, a study showed that many accidents in the city happen when drivers turn right at red lights and don't stop for people walking. The city is trying to fix this and stop accidents by aiming for no serious injuries or deaths by 2024.

Some people want to stop or limit the right turn on red rule at busy places to keep walkers safer. But even with the city's efforts, accidents still happen in San Francisco.

Objectives

- **Analyze Historical Data:** Study past accident data to find patterns and factors contributing to RTOR-related pedestrian accidents, including variables like time of day, weather, and traffic volume.
- **Predict Accidents Using Machine Learning:** Use machine learning models to predict the likelihood of pedestrian accidents at intersections with RTOR, helping to identify high-risk areas for future incidents.

Motivation/Expected Outcome

- **Goal:** This project focuses on improving urban safety, using San Francisco as a case study for RTOR traffic lights and pedestrian accidents.
- **Approach:** By applying data analysis, predictive modeling, and community input, the project aims to understand how RTOR policies affect pedestrian safety.
- **Expected Results:** The project aims to identify accident trends, pinpoint high-risk intersections, and provide evidence-based suggestions for policy changes to prioritize pedestrian safety.
- **Ultimate Goal:** The goal is to reduce RTOR-related pedestrian accidents, contributing to the broader effort of eliminating traffic-related fatalities and injuries. With stakeholder involvement, the project seeks to implement practical safety measures in North American urban areas.

DATASET

Data Sources

- **Dataset Owner:** Open Data (USA)
- **Dataset Name:** TransBASE.sfgov.org

- **Data Link:** Traffic Crashes Resulting in Injury Victims

Data Collection Method

The data is filled out by police officers within 30 days of a crash happening. After it is collected from the California Highway Patrol's 555 Crash Report.

- **Date Collected:** October 3, 2024
- The data was downloaded in CSV format from the website.
- It includes data from 2022 to the collection date.

Initial Data Overview

- The dataset has 76,511 rows and 91 columns.
- The columns contain different types of data, including:
    - Categorical data ○ Unique Identifiers ○ Time data ○ Geographical data ○ Text data
    - Numerical data

Data Processing

· The data required some measure of cleaning, also for the **Dataset Features** we Identified the dependent and independent variables

Data Exploration Steps

- Identifying the most crucial variables needed for our analysis.
- Verifying variable datatypes.
- Checking for how many missing rows each variable had.
- Checking Data summarisation and aggregation for each variable, such as minimum, mean, median, maximum values.
- Checking for unique values in each column and identifying naming inconsistencies.

Data cleaning steps

- Dropping unwanted columns.
- Removing rows with null and missing values.
- Correcting all negative values in the age column.
- Changed datatype for required columns.
- Standardizing string columns such as (inattention, type of collision etc.) that have inconsistent unique value naming convention, to ensure that all values which mean the same thing have the same name.
- Data imputation for necessary columns.

- Filtering all data based on right turn on red conditions and creating a new data frame from it.

Tools Used

- Microsoft Office Suite (Word, PowerPoint, Teams, Excel)
- Python
- Tableau

The independent variables are most of the columns in the dataset, later in this project, we would be creating a dependent variable based on conditions from 2 or more independent variables for predictive modeling and analysis.

Data Exploration and Findings

We conducted an exploratory data analysis (EDA) to examine both univariate and bivariate distributions within the dataset. We assessed the relationships between variables, identifying correlations and key patterns. Following a thorough data cleaning process, we exported the refined dataset for further analysis. Additionally, we utilized Tableau to create visualizations that provided deeper insights into the data, enabling more effective interpretation and decision-making.

Figure 1 . Correlation heat map for Numerical Features

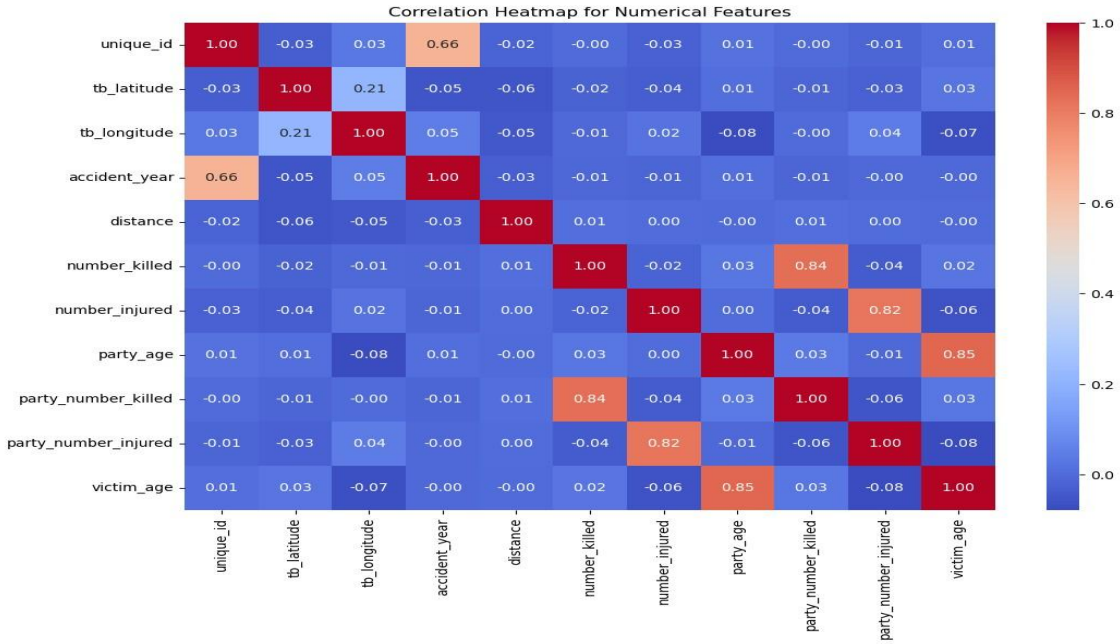


Figure 2. Type of Collision vs Collision Severity

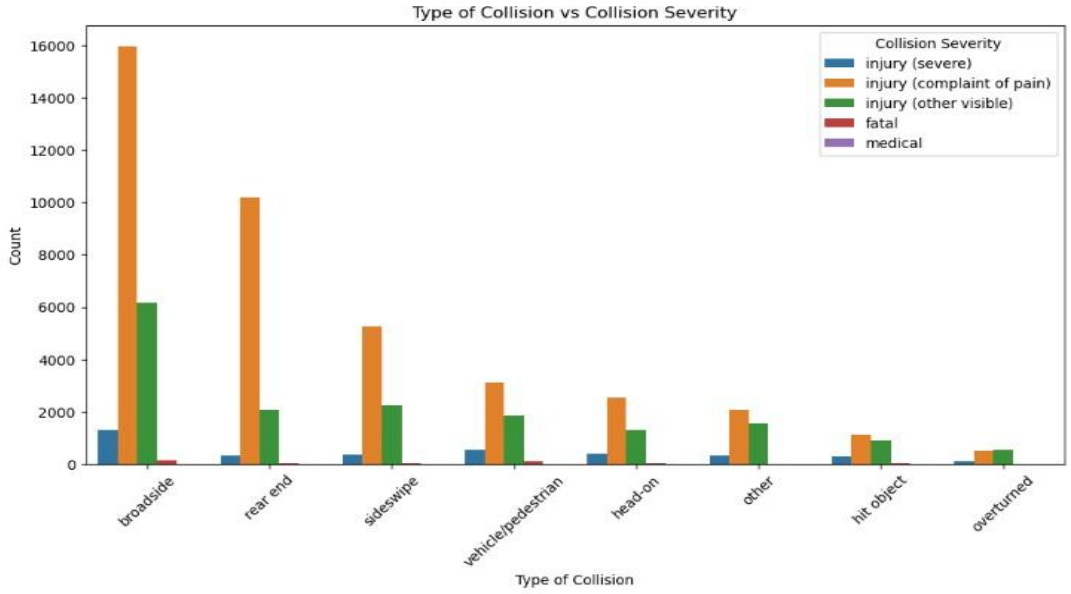


Figure 3. Distribution of Collision Severity

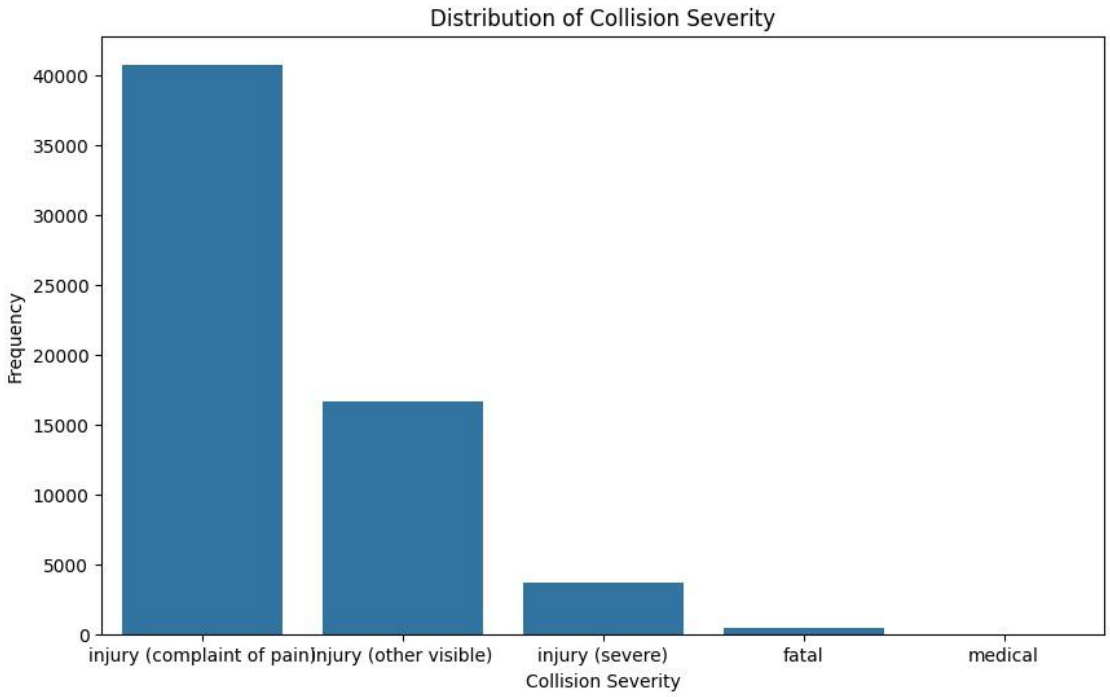
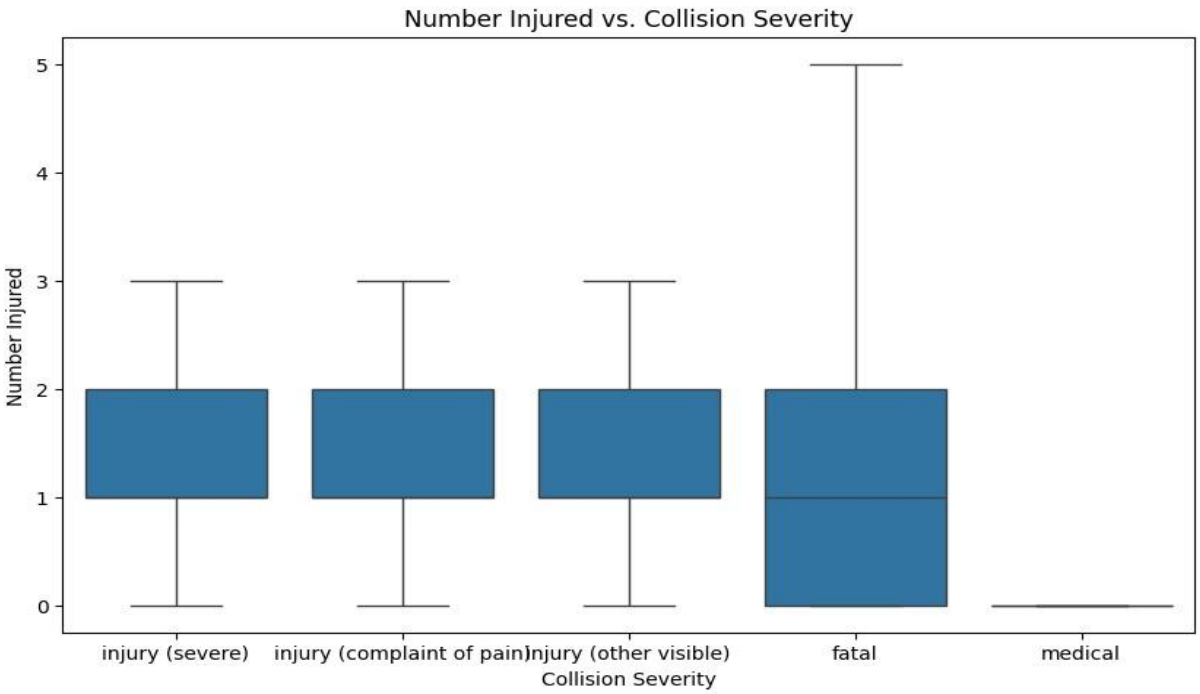


Figure 4. Boxplots of Injured vs Collision Severity



Our Tableau visualization revealed several key insights:

- **Age Distribution of Accident Victims:** Individuals aged 20-29 were the most frequently affected, followed by those in the 30-39 and 40-49 age groups. This suggests that younger, more active populations are at higher risk in accident-prone areas.
- **Top Accident-Prone Routes:** Mission Street and Market Street had the highest accident rates, followed by Van Ness Avenue and Geary Boulevard. This could be due to high traffic density, inadequate pedestrian infrastructure, or poor enforcement of right-turn regulations.
- **Weather Conditions During Accidents:** Most accidents occurred in clear weather, indicating that visibility or road conditions were not the primary contributing factors. Instead, human error, traffic design, or intersection control likely played a significant role.
- **Collision Severity by Day of the Week:** Injury-related accidents dominated across all days, while fatal accidents remained low. A slight decline in weekend incidents suggests that reduced traffic volumes might contribute to fewer accidents.

These insights provided valuable information on accident patterns, helping to identify potential areas for traffic safety improvements.

Figure 5. Main Dashboard

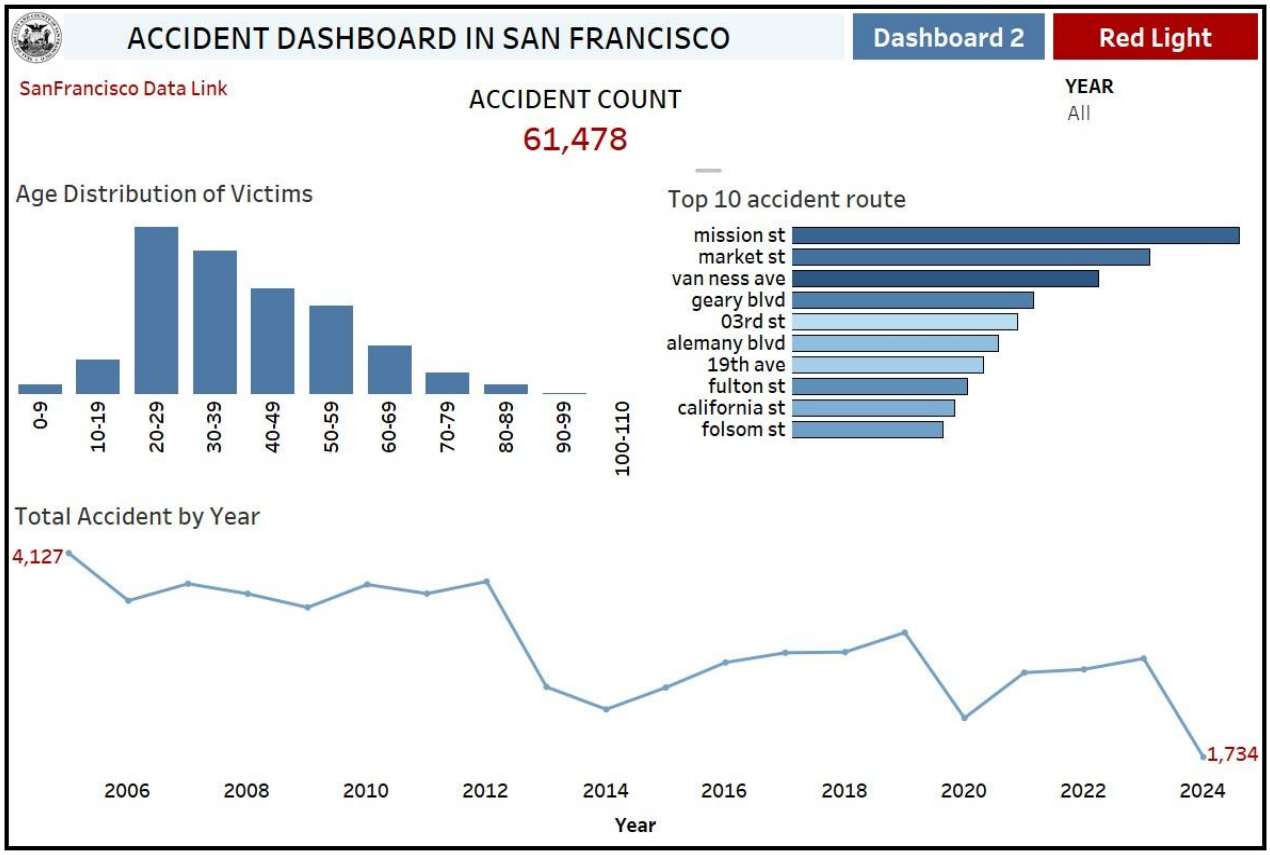


Figure 6 Dashboard 2

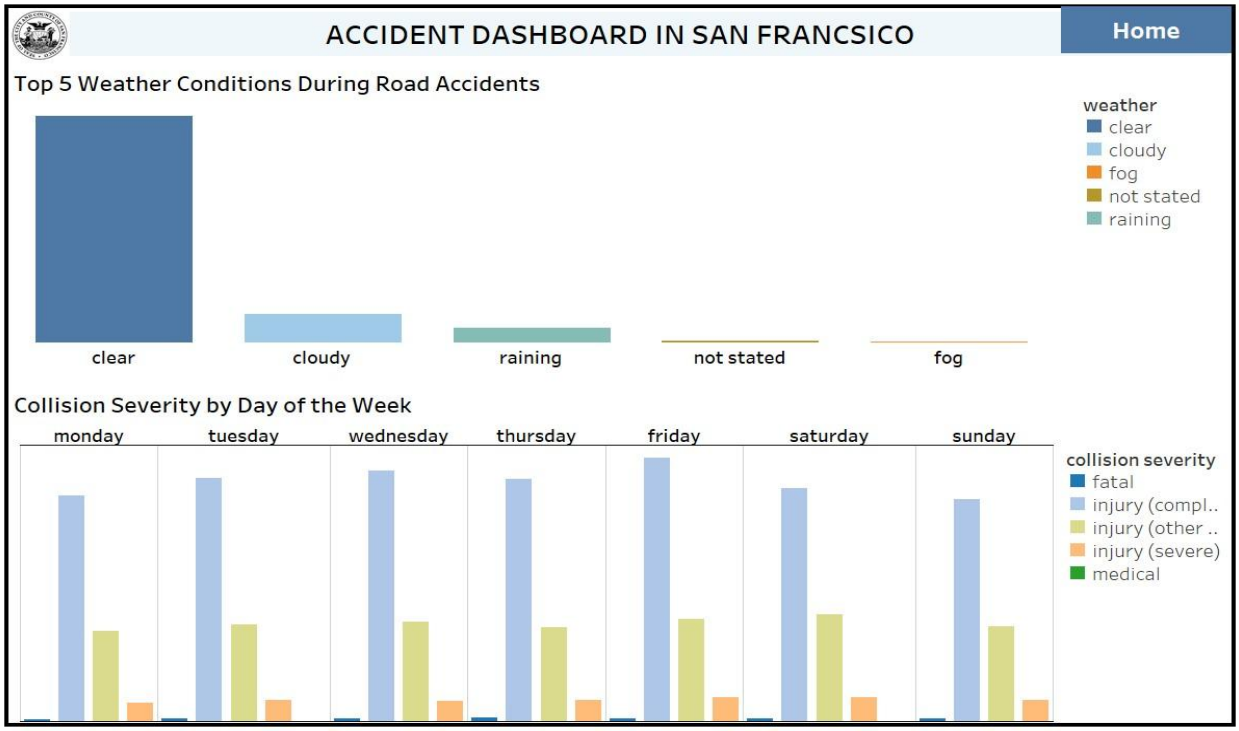
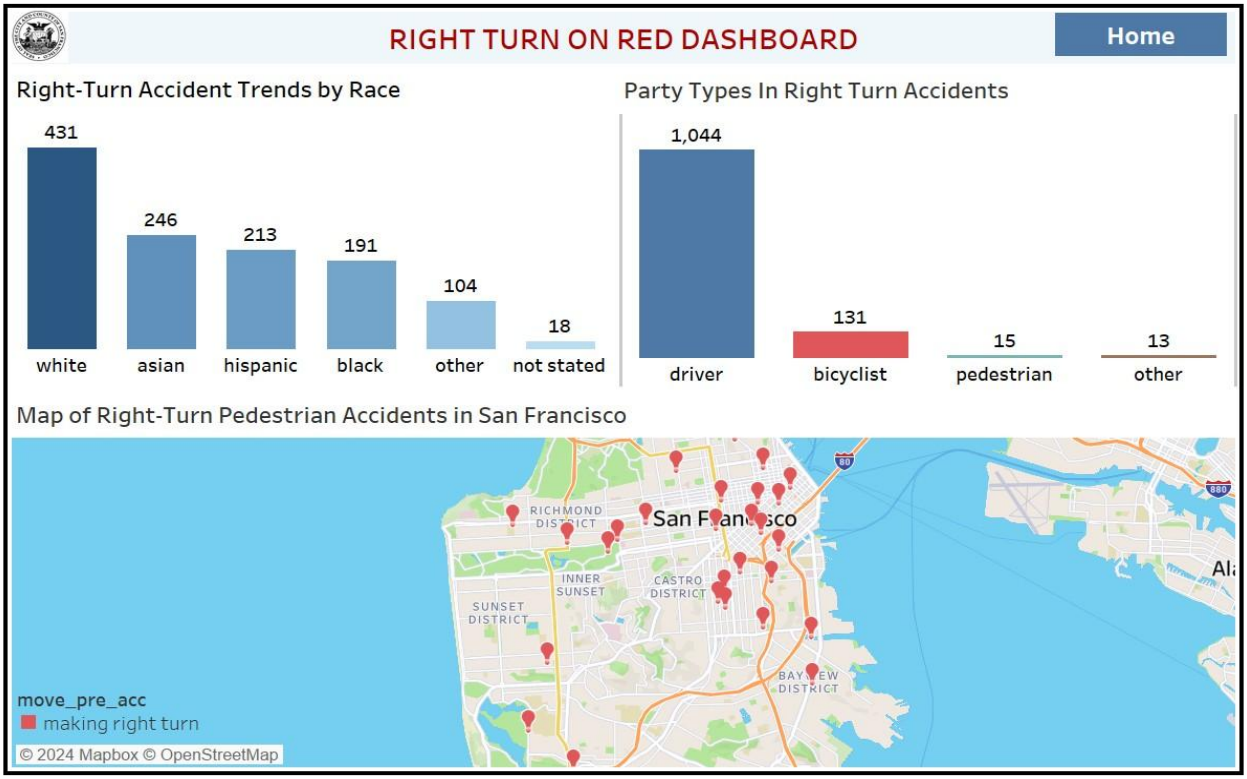


Figure 7 Red Dashboard



ADVANCED ANALYTICS / PREDICTIVE ML MODEL

For the purpose of this project, we will be incorporating images of the respective intersections into our existing dataset. We would be applying machine learning models used for computer vision tasks such as Convolutional Neural Networks (CNN) to predict the likelihood of pedestrian accidents at intersections with RTOR.

Approach (Methodology)

Obtaining/Downloading San Fransico Intersection Images

- Based on exploratory analysis, we realized that although we had over 76,000 rows each representing data of an accident occurrence, there were only about 6300 unique intersections in the entire dataset.
- The above discovery drastically reduced the number of images required for this project.
- **Data Source**: After numerous research on reliable sources for image datasets, we settled on modifying the street view links in the initial dataset, replicating same for all rows to achieve the desired images.
- **Automated Downloading**: Using the modified links we researched and explored the use of scripts or APIs to automate the download process. API'S were mostly paid and expensive, hence we resorted to the use of scripts on the command line using Google Street View to fetch images of the specific corresponding intersections.

Merging resized Images with Existing dataset.

To merge resized images with the existing dataset, we stored the paths of resized images in the dataset dataframe, ensuring each image path corresponds to the appropriate row in the dataset.

Figure 8

```
# 🔹 Remove NaN before conversion
df = df.dropna(subset=["cnn_intrsctn_fkey"])

# Convert intersection keys to integer strings (fixing float issue)
df["cnn_intrsctn_fkey"] = df["cnn_intrsctn_fkey"].astype(float).astype(int).astype(str)

# Convert filenames to lowercase and strip spaces
image_files = {str(int(os.path.splitext(f)[0])).strip(): os.path.join(image_dir, f)
               for f in os.listdir(image_dir) if f.endswith(('.jpg', '.png'))}

# Re-run mapping
df["image_path"] = df["cnn_intrsctn_fkey"].map(image_files)

# 🔹 Debugging: Check how many were mapped
missing_images = df["image_path"].isna().sum()
total_rows = len(df)

print(f"\n✅ Rows successfully mapped: {total_rows - missing_images}/{total_rows}")

# Drop rows where image_path is missing
df = df.dropna(subset=["image_path"])

# Save updated CSV
merged_csv_file = "merged_intersections.csv"
df.to_csv(merged_csv_file, index=False)

print(f"\n📂 Merged CSV saved to {merged_csv_file}")
```

```
✅ Rows successfully mapped: 76306/76490

📂 Merged CSV saved to merged_intersections.csv
```

Resizing Intersection Images and Label Encoding

To ensure all images are of the same size, which is crucial for feeding into machine learning models, we employed image processing libraries such as OpenCV or PIL to resize images to a standard dimension 512x512 pixels. Optimizing to reduce image size while preserving image quality. Label encoding and standard scaling were applied to structured data.

Figure 9

Feature Engineering : To **indicate whether a pedestrian was harmed** (e.g., injured or killed) during a **Right Turn on Red (RTOR)** event. This helps capture safety risk outcomes for predictive modeling or pattern analysis.

Figure 10

Feature Engineering

```
In [14]: data['harm'] = data.apply(
             lambda row: 1 if (row['number_killed'] > 0 or row['number_injured'] > 0) and row['move_pre_acc'] == 'making right turn' else
             axis=1
         )
```

```
In [16]: harm_check = data['harm'].value_counts()
         harm_check
```

```
Out[16]: 1    71566
         0      206
         Name: harm, dtype: int64
```

Selecting and Building a Machine Learning Model

- Considering the problem type which we're trying to solve which is more of a classification problem and the data characteristics of the dataset we have, we would be applying one of the common models for image classification, the Convolutional Neural Networks (CNNs) .CNN architecture was implemented using **Functional API**.

Figure 11

In [31]:
```
# Pass the combined input through convolutional layers
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu')(combined_input)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)
x = tf.keras.layers.Flatten()(x)

# Add fully connected layers
x = tf.keras.layers.Dense(128, activation='relu')(x)
output = tf.keras.layers.Dense(len(np.unique(data['harm'])), activation='softmax')(x)  # Adjust for number of classes
```

In [32]:
```
model = tf.keras.Model(inputs=[image_input, structured_input], outputs=output)

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [33]:
```
# Train the model
model.fit(
    [images, combined_features_scaled],  # Inputs: images and structured data
    data['harm'],                        # Labels: target variable
    epochs=10,
    batch_size=32,
    validation_split=0.2
)
```

```
Epoch 1/10
```

Model Evaluation Strategy

To assess model performance, **accuracy** was initially selected as the primary evaluation metric. Given the binary classification task, accuracy provided a straightforward measure of correct predictions over total predictions. During initial evaluation, the model achieved **high accuracy on the training set** but performed **substantially worse on the validation set**, suggesting the presence of **overfitting**. No early stopping or regularization had yet been applied at this stage.

Although additional metrics such as **precision**, **recall**, and **F1-score** were considered, they were deferred in this phase due to a preliminary focus on overall classification success across classes. However, the presence of class imbalance likely inflated the training accuracy.

Figure 12

In [34]:
```
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(
    [images, combined_features_scaled],  # Inputs: images and structured data
    data['harm']                         # Ground truth labels for harm
)

# Print evaluation metrics
print(f"Test Loss: {test_loss:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
337/337 ━━━━━━━━━━━━━━━━━━━━ 102s 299ms/step - accuracy: 0.9960 - loss: 0.0292
Test Loss: 0.03
Test Accuracy: 1.00
```

Overfitting Mitigation: SMOTE and Augmentation

To address the overfitting issue and improve generalization, two corrective actions were implemented:

1. **SMOTE (Synthetic Minority Oversampling Technique):**
   Applied to the structured (tabular) data, SMOTE synthetically generated new instances of the minority class by interpolating between existing minority samples and their k-nearest neighbors. This technique helped **rebalance the class distribution** and reduce bias in favor of the majority class during training.

2. **Image Augmentation with K-Nearest Neighbor Expansion:**
   For the image input pipeline, augmentation was performed using randomized transformations (e.g., rotation, brightness scaling, flipping). Additionally, synthetic augmentation via a **KNN-based technique** was applied to extend underrepresented class samples by identifying nearest neighbors in the image feature space and generating perturbed variations.

Following these preprocessing steps, the model was retrained, and **accuracy was re-evaluated**. The results demonstrated significantly improved validation accuracy and reduced variance between training and validation performance—indicating improved model generalization and reduced overfitting risk.

Figure 13

```
: from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_struct_resampled, y_resampled = sm.fit_resample(X_struct_train_imputed, y_train)
```

Figure 14

In [ ]:
```python
from sklearn.impute import SimpleImputer
from sklearn.neighbors import NearestNeighbors
import numpy as np
import tensorflow as tf

# # Impute missing values in structured data
# imputer = SimpleImputer(strategy='median')
# X_struct_train = imputer.fit_transform(X_struct_train)

# Fit NearestNeighbors on clean structured data
nn = NearestNeighbors(n_neighbors=1)
nn.fit(X_struct_train)

# Define image augmentation
def augment_image_batch(image_batch):
    augmented = []
    for img in image_batch:
        img = tf.image.random_flip_left_right(img)
        img = tf.image.random_brightness(img, max_delta=0.1)
        img = tf.image.random_contrast(img, lower=0.8, upper=1.2)
        img = tf.image.random_saturation(img, lower=0.9, upper=1.1)
        img = tf.image.random_crop(img, size=[200, 200, 3])
        img = tf.image.resize(img, [224, 224])
        augmented.append(img)
    return tf.stack(augmented)

# Define generator that safely casts images on-the-fly
def generate_augmented_batches(X_struct_resampled, y_resampled, X_struct_train, X_img_train, batch_size=32):
    for i in range(0, len(X_struct_resampled), batch_size):
        # Get batch
        batch_struct = X_struct_resampled[i:i+batch_size]
```

Figure 15

In [54]:
```python
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(
    [X_img_test, X_struct_test],  # Inputs: preprocessed test data
    y_test                        # Ground truth labels
)

# Print evaluation metrics
print(f"Test Loss: {test_loss:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
68/68 ━━━━━━━━━━━━━━━━━━━━ 20s 291ms/step - accuracy: 0.6744 - loss: 1.5669
Test Loss: 2.85
Test Accuracy: 0.67
```

Model Predictions

Our preliminary prediction indicates that :

- Intersection ID 30070000.0 has the highest number of predictions (31).
- Intersection IDs 22556000.0, 30739000.0, and 24922000.0 follow closely with 27 predictions each.

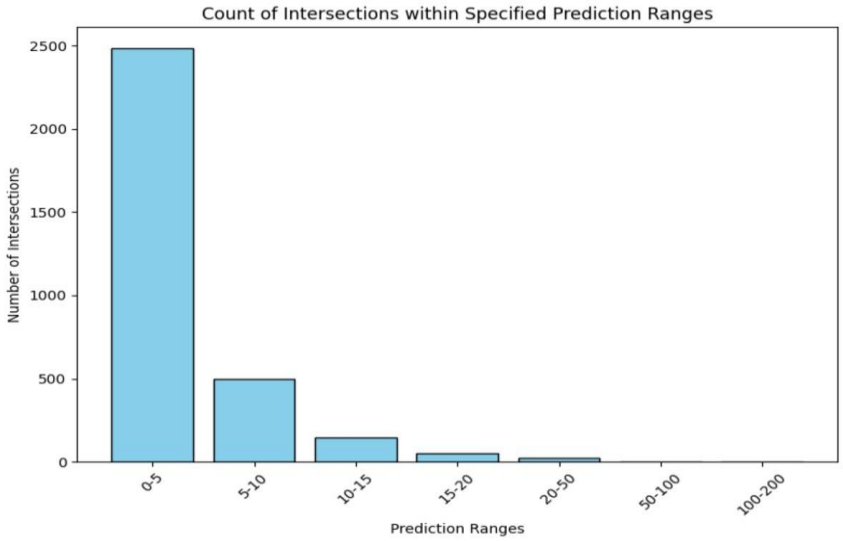
Figure 16

Dangerous Intersections (as a DataFrame)

Intersection ID	Total Predictions
30070000.0	31.0
22556000.0	27.0
30739000.0	27.0
24922000.0	27.0
26621000.0	26.0
24429000.0	26.0
23897000.0	26.0
24241000.0	25.0
23149000.0	25.0
24450000.0	24.0
24388000.0	24.0
23906000.0	23.0
28106000.0	23.0
24293000.0	23.0
26345000.0	22.0
24020000.0	22.0
24142000.0	22.0
33027000.0	22.0
26547000.0	21.0
24459000.0	21.0
27444000.0	20.0
23799000.0	20.0
24022000.0	20.0
25967000.0	19.0
25179000.0	19.0
30756000.0	19.0
23919000.0	19.0
26347000.0	19.0
23953000.0	18.0
26973000.0	18.0
25186000.0	18.0
30738000.0	18.0
26031000.0	18.0

Predictions Frequency Chart

Figure 17



CHALLENGES AND LIMITATIONS

This project encountered several key challenges during the data acquisition and integration phases, particularly concerning image handling and model training efficiency:

1. **Image Acquisition and Integration Complexity**
   The process of sourcing, downloading, and incorporating image data into the structured dataset presented multiple obstacles. These included **limited data availability**, **API rate limits**, and **inconsistencies in file formats** and naming conventions across sources. Additionally, attempts at web scraping were constrained by website restrictions and content variability, further complicating reliable image collection.

2. **Record-to-Image Mapping Difficulties**
   Aligning images with their corresponding structured data records posed a significant challenge due to **non-standardized or misaligned unique identifiers**, **missing image files**, and **label inconsistencies**. These discrepancies reduced the completeness and reliability of the multimodal dataset, requiring extensive manual inspection and validation.

3. **Resource Constraints for Image Processing**
   The large volume of image data led to **heavy demands on storage capacity and memory (RAM)** during preprocessing and training. Moreover, **model training on CPU hardware proved to be prohibitively slow**, particularly for deep learning workflows involving convolutional neural networks (CNNs). This computational bottleneck limited experimentation and extended model iteration times.

Recommendations and Future Research

Analysis revealed that **left-turn manoeuvres are associated with a higher frequency of collisions** compared to other movements. This finding highlights the need for focused research on left-turn-related incidents and the development of targeted interventions to reduce risk.

To enhance intersection safety and reduce turn-related accidents, the following recommendations are proposed:

1. **Expand Research on Left-Turn Accidents**
   Conduct a dedicated study to investigate the underlying causes of high collision rates during leftturn movements, considering variables such as signal phasing, intersection design, and visibility constraints.
2. **Implement Red-Light Cameras and Automated Enforcement**
   Deploy **automated traffic enforcement systems**—such as red-light cameras—at intersections identified as high-risk based on historical accident data. These tools can help deter traffic violations and support evidence-based enforcement.

3. **Restrict Right Turns on Red During Peak Hours**

At intersections with elevated accident rates, consider implementing **time-based restrictions on right turns during peak traffic periods** to reduce pedestrian and vehicle conflict points.

4. **Introduce Dedicated Right-Turn Lanes with Improved Signage**
   Where feasible, install **dedicated right-turn lanes** with clear pavement markings and highly visible signage. This can help streamline vehicle movements, reduce hesitation or confusion, and lower the likelihood of collisions.

These strategies should be integrated with ongoing traffic engineering studies and evaluated through future pilot programs to validate their effectiveness and scalability.

Appendix

Libraries

```
import pandas as pd import
numpy as np import
tensorflow as tf
from sklearn.preprocessing import StandardScaler, LabelEncoder from
sklearn.model_selection import train_test_split
from datetime import datetime, time Data
```

Cleaning Phase 1

```
#Dropping unnecessary columns columns_to_drop = [
   "geocode_source", "juris", "officer_id", "beat_number", "vz_pcf_code", "vz_pcf_link",
   "street_view", "party_number_ckey", "victim_id", "data_as_of",
   "data_updated_at", "data_loaded_at","Current Police Districts", "Current Supervisor Districts",
   "Analysis Neighborhoods", "Neighborhoods", "SF Find Neighborhoods",
   "oaf_viol_section", "oaf_violation_code", "oaf_violation_suffix","special_info_f",
   "special_info_g","stwd_vehicle_type","geocode_location","collision_datetime",
   "intersection","victim_ejected","victim_seating_position","victim_safety_equip_1",
      "victim_safety_equip_2", "point","supervisor_district','pre_link', 'vehicle_year', 'post_link',
'full_streetview_link','collision_datetime'
]

data.drop(columns=columns_to_drop, inplace=True, errors='ignore')

# Display remaining columns and shape after dropping
remaining_columns = data.columns
df_shape_after_dropping = data.shape
```

```
remaining_columns, df_shape_after_dropping #
Assuming your DataFrame is named 'data'
missing_case_id_rows = data[data['case_id_pkey'].isna()]

# To display the rows where 'case_id_pkey' is NaN print(missing_case_id_rows['case_id_pkey'])
# Convert negative age values to positive in 'party_age' and 'victim_age'
data['party_age'] = data['party_age'].abs() data['victim_age'] =
data['victim_age'].abs()

# Replace NaN with 0 and inf with a large number or -1
data[['party_age', 'victim_age' ]] = data[['party_age', 'victim_age' ]].fillna(0)  # Replace NaN with 0
data[['party_age', 'victim_age' ]] = data[['party_age', 'victim_age' ]].replace(np.inf, -1)  # Replace inf with 1

# Clean the victim_age column
data = data[(data['victim_age'].astype(int) <= 100) & (data['victim_age'].astype(int) >= 1)]

# Clean the party_age column
data = data[(data['party_age'].astype(int) <= 100) & (data['party_age'].astype(int) >= 16)]

# Convert to integer and filter out unrealistic years in 'vehicle_year' column
#df['vehicle_year'] = pd.to_numeric(df['vehicle_year'], errors='coerce')  # Convert all values to numeric
#df['vehicle_year'] = df['vehicle_year'].apply(lambda x: x if (x >= 1900 and x <= 2024) else None)  # Set outof-
range to NaN data['victim_age'] = pd.to_numeric(data['victim_age'], errors='coerce').fillna(0).astype(int)
data['party_age'] = pd.to_numeric(data['party_age'], errors='coerce').fillna(0).astype(int)
data['number_killed'] = pd.to_numeric(data['number_killed'], errors='coerce').fillna(0).astype(int)

# Replace null values in 'oaf_viol_cat' with "Not Stated"
# data['oaf_viol_cat'].fillna("Not Stated", inplace=True)

# make'inattention' values more readable
data['inattention'] = data['inattention'].replace({     "Cell
Phone Hands Free": "Cellphone Handsfree",
   "Unknown": "Not Stated",
   " Cell Phone": "Cell Phone Handheld",
   "Electronic Equip": "Electronic Equipment"
})
```

```
# Replace NaN and 'Not Stated' values with 'Other' in the 'oaf_2' column data['oaf_2'] =
data['oaf_2'].replace(['Not Stated', None], 'Other').fillna('Other')

#  Replace NaNs in 'street_of_travel' with "Not Stated" and standardize values
data['street_of_travel'] = data['street_of_travel'].replace({"Not Collected": "Not Stated"}).fillna("Not Stated")

#  Replace " Not Collected" in 'vehicle_autonomous' with "Not Stated"
data['vehicle_autonomous'] = data['vehicle_autonomous'].replace({"Not Collected": "Not Stated"})

# Replace " -" in 'vehicle_make' with "Unknown"
data['vehicle_make'] = data['vehicle_make'].replace({"-": "Unknown"})

#  Replace 'not stated' in 'weather_1'' with "other"
data['weather_1'] = data['weather_1'].replace({'not stated': 'other'})

#  Replace 'not stated' in 'type_of_collision'' with "other"
data['type_of_collision'] = data['type_of_collision'].replace({'not stated': 'other'})

from sklearn.impute import SimpleImputer from
sklearn.neighbors import NearestNeighbors import
numpy as np
import tensorflow as tf

# # Impute missing values in structured data
# imputer = SimpleImputer(strategy='median')
# X_struct_train = imputer.fit_transform(X_struct_train)

# Fit NearestNeighbors on clean structured data
nn = NearestNeighbors(n_neighbors=1)
nn.fit(X_struct_train)

# Define image augmentation def
augment_image_batch(image_batch):    augmented = []    for img in
image_batch:        img = tf.image.random_flip_left_right(img)
img = tf.image.random_brightness(img, max_delta=0.1)        img =
tf.image.random_contrast(img, lower=0.8, upper=1.2)        img =
tf.image.random_saturation(img, lower=0.9, upper=1.1)        img =
tf.image.random_crop(img, size=[200, 200, 3])
      img = tf.image.resize(img, [224, 224])
```

```
        augmented.append(img)
    return tf.stack(augmented)
```

Apply Smote

```
# Define generator that safely casts images on-the-fly def
generate_augmented_batches(X_struct_resampled, y_resampled, X_struct_train, X_img_train,
batch_size=32):     for i in range(0, len(X_struct_resampled), batch_size):
        # Get batch
        batch_struct = X_struct_resampled[i:i+batch_size]
batch_labels = y_resampled[i:i+batch_size]

        # Find nearest real structured neighbors
        _, idx = nn.kneighbors(batch_struct)

        # Safe float32 cast only for current batch
        matched_imgs = np.asarray(X_img_train[idx.flatten()], dtype=np.float32)

        # Augment matched images
        augmented_imgs = augment_image_batch(matched_imgs)

        yield (augmented_imgs, batch_struct), batch_labels import
tensorflow as tf

def get_tf_dataset():
    output_signature = (
        (tf.TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32),
tf.TensorSpec(shape=(None, 62), dtype=tf.float32)),         tf.TensorSpec(shape=(None,),
dtype=tf.int32)
    )

    return tf.data.Dataset.from_generator(
        lambda: generate_augmented_batches(X_struct_resampled, y_resampled, X_struct_train, X_img_train,
batch_size=32),
        output_signature=output_signature
    )

# Evaluate the model on the test data test_loss,
test_accuracy = model.evaluate(
    [X_img_test, X_struct_test],  # Inputs: preprocessed test data
```

```
    y_test                    # Ground truth labels
)

# Print evaluation metrics print(f"Test
Loss: {test_loss:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")
# Step 1: Predict danger levels using the trained model
predictions = model.predict([images, combined_features_scaled])  # Assuming `images` and
`combined_features_scaled` are inputs
predicted_classes = np.argmax(predictions, axis=1)  # Extract the predicted harm levels (classes)

# Step 2: Combine predictions with intersection identifiers
# Assuming `cnn_intrsctn_fkey` is present in the structured data (`combined_features_scaled`)
data_with_predictions = pd.DataFrame({
    'cnn_intrsctn_fkey': data['cnn_intrsctn_fkey'],  # Intersection key
    'predicted_harm': predicted_classes            # Predicted harm levels
})

# Step 3: Aggregate danger metrics by intersection #
Group by intersection and calculate metrics
intersection_ranking = data_with_predictions.groupby('cnn_intrsctn_fkey').agg(
total_predictions=('predicted_harm', 'count'),     avg_predicted_danger=('predicted_harm',
'mean'),     max_predicted_danger=('predicted_harm', 'max')
).reset_index()

# Step 4: Scale danger levels to a scale of 1 to 10
# Normalize average predicted danger to a scale of 1 to 10 intersection_ranking['danger_score'] = (
    10 * (intersection_ranking['avg_predicted_danger'] / intersection_ranking['avg_predicted_danger'].max())
)

# Step 5: Rank intersections by danger score
intersection_ranking = intersection_ranking.sort_values(by='danger_score', ascending=False)

# Display the top 10 most dangerous intersections print("Top 10
Most Dangerous Intersections:")
print(intersection_ranking.head(10))
```

REFERENCES

1. *Traffic crashes resulting in injury: Victims involved | DataSF | City and County of San Francisco*. (2024, October 3). https://data.sfgov.org/Public-Safety/Traffic-Crashes-Resulting-in-InjuryVictims-Involv/nwes-mmgh/about_data
2. Baldini, G., Amerini, I., Dimc, F., & Bonavitacola, F. (2023). Convolutional Neural Networks combined with feature selection for radio-frequency fingerprinting. *Computational Intelligence*, *39*(5), 734–758. https://doi.org/10.1111/coin.12592
3. PrathamModi. (2023, October 19). Keras CNN tutorial: Classifying images made easy. Medium. https://medium.com/@prathammodi001/keras-cnn-tutorial-classifying-images-made-easy-fb55cc8892ec