index

February 6, 2025

1 Business Understanding

1.1 Introduction

Stakeholders such as traffic authorities and emergency services often face challenges in predicting and mitigating fatal/severe injuries in traffic crashes. By identifying key factors contributing to fatal/severe injury, stakeholders can gain useful insights to implement proactive measures to reduce injury severity and save lives.

1.2 Use Cases

- Use the model to identify high-risk conditions and implement measures like improved signage, speed limits or road design to reduce injury severity in traffic accidents.
- Predict whether an incident is fatal or not based on crash conditions, enhancing decision-making and resource allocation for emergency services by prioritizing high-risk incidents.

1.3 Value Proposition

This project aims to develop a classification model that predicts injury severity in traffic crashes. By identifying key high-risk contributing to severe injuries, stakeholders can implement proactive measures to: - Enhance decision-making and resource allocation for emergency services - Improve public safety and save lives

2 Data Understanding

2.1 Introduction

The dataset contains information about traffic accidents in Chicago. Stakeholders need reliable data-driven insights to mitigate injury severity and optimize their strategies. The dataset in this project is directly related to the task of predicting injury severity in traffic accidents.

2.2 Data Description

- The dataset includes detailed records of traffic accidents covering various features such as environment, crash dynamics, human factors, location factors and target variable MOST_SEVERE_INJURY.

2.3 Data Quality

- The dataset is very large with over 650,000 records and 49 features, providing a rich source of information for analysis.
- The dataset comes from the City of Chicago's open data portal and is updated daily making it a reliable source of information for stakeholders.

2.4 Data Relevance

- Use data on crash conditions (eg. weather) to identify high-risk conditions take proative measures.
- Predict injury severity to prioritize emergency services and allocate resources more effectively.

2.5 Conclusion

The dataset is robust, relevant and continually updated, making it an indispensable resource for the task of predicting injury severity in traffic accidents.

3 Data Preparation

3.1 Assembly

- The source data is comprised of three CSV files:
  - Crash Data
  - Driver Data
  - Vehicles Data
- The data will be assembled into a single dataset by joining the three tables on the common key CRASH_RECORD_ID.

3.2 Cleaning

- Irrelevant columns that do not contribute to the task will be dropped.
- Missing values that will be imputed or dropped.

3.3 Transformation

- Categorical features will be encoded using one-hot encoding.
- Numerical features will be scaled using standard scaling.

3.4 Splitting

- The dataset will be split into training and testing sets using a standard 80/20 split.

3.5 Import Libraries

[31]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import chi2
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.feature_selection import f_classif
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.sparse import hstack
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import FunctionTransformer
from sklearn.base import BaseEstimator, TransformerMixin
from scipy.stats import spearmanr
import json
from sklearn.model_selection import ParameterGrid
from scipy.stats import chi2_contingency
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
```

3.6 Load Data

[32]:
```
# load data
data = pd.read_csv('./data/Traffic_Crashes_-_Crashes_20250127.csv')
data_vehicles = pd.read_csv('./data/Traffic_Crashes_-_Vehicles_20250127.csv')
data_people= pd.read_csv('./data/Traffic_Crashes_-_People_20250127.csv')
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_88313/1241634393.py:3
: DtypeWarning: Columns (18,20,39,40,41,43,47,48,49,52,54,57,58,60,70) have
mixed types. Specify dtype option on import or set low_memory=False.
  data_vehicles = pd.read_csv('./data/Traffic_Crashes_-_Vehicles_20250127.csv')
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_88313/1241634393.py:4
: DtypeWarning: Columns (19,28) have mixed types. Specify dtype option on import
or set low_memory=False.
  data_people= pd.read_csv('./data/Traffic_Crashes_-_People_20250127.csv')
```

[33]:
```
# merge data
data = data.merge(data_vehicles, on='CRASH_RECORD_ID')
data = data.merge(data_people, on='CRASH_RECORD_ID')
```

[34]:
```
# assign target variable - MOST_SEVERE_INJURY and convert to binary 0:␣
↪NON-FATAL, 1: FATAL
data['MOST_SEVERE_INJURY'] = data['MOST_SEVERE_INJURY'].apply(lambda x: 0 if x !
↪= 'FATAL' else 1)
```

3.7 Intial Clean Up

- Drop features unlikely to influence injury severity.
  - ids
  - location
  - date/time
  - miscellaneous
  - vehicle details
  - hazmat details
  - commerical vehicle details

[35]:
```
categorical_drop = [
    # IDs
    'CRASH_RECORD_ID', 'PERSON_ID', 'USDOT_NO', 'CCMC_NO', 'ILCC_NO',␣
↪'VEHICLE_ID_x', 'VEHICLE_ID_y', 'EMS_RUN_NO', 'IDOT_PERMIT_NO', 'UNIT_NO',

    # Dates
    'DATE_POLICE_NOTIFIED', 'CRASH_DATE_EST_I', 'CRASH_DATE_x', 'CRASH_DATE_y',␣
↪'CRASH_DATE',

    # Geographic
    'CITY', 'STATE', 'ZIPCODE', 'LATITUDE', 'LONGITUDE', 'LOCATION',␣
↪'STREET_NAME', 'STREET_DIRECTION', 'CARRIER_STATE', 'CARRIER_CITY',␣
↪'STREET_NO', 'BEAT_OF_OCCURRENCE', 'TRAVEL_DIRECTION',

    # Miscellaneous
    'TOWED_BY', 'TOWED_TO', 'AREA_00_I', 'AREA_01_I', 'AREA_02_I', 'AREA_03_I',␣
↪'AREA_04_I', 'AREA_05_I', 'AREA_06_I', 'AREA_07_I', 'AREA_08_I',␣
↪'AREA_09_I', 'AREA_10_I', 'AREA_11_I', 'AREA_12_I', 'AREA_99_I',␣
↪'WORK_ZONE_TYPE', 'PHOTOS_TAKEN_I', 'STATEMENTS_TAKEN_I', 'DOORING_I',␣
↪'WIDE_LOAD_I', 'REPORT_TYPE', 'CRASH_TYPE',

    # Vehicle
    'VEHICLE_ID', 'MAKE', 'MODEL', 'LIC_PLATE_STATE', 'TRAILER1_WIDTH',␣
↪'TRAILER2_WIDTH',

    # Hazardous Materials
    'HAZMAT_PLACARDS_I', 'HAZMAT_NAME', 'HAZMAT_PRESENT_I', 'HAZMAT_REPORT_I',␣
↪'HAZMAT_REPORT_NO', 'HAZMAT_VIO_CAUSE_CRASH_I', 'HAZMAT_OUT_OF_SERVICE_I',

    # Commercial Vehicle
```

```
    'COMMERCIAL_SRC', 'CARGO_BODY_TYPE', 'VEHICLE_CONFIG', 'GVWR',␣
 ↪'CARRIER_NAME', 'MCS_VIO_CAUSE_CRASH_I', 'MCS_REPORT_I', 'MCS_REPORT_NO',␣
 ↪'MCS_OUT_OF_SERVICE_I',

    # High/Inf VIF
    'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITATING',␣
 ↪'INJURIES_NON_INCAPACITATING', 'INJURIES_REPORTED_NOT_EVIDENT',␣
 ↪'CRASH_UNIT_ID',

    # Related to Target
    'INJURY_CLASSIFICATION', 'INJURIES_NO_INDICATION', 'INJURIES_UNKNOWN',

    # Potential Drops
    'VEHICLE_USE', 'BAC_RESULT', 'DRIVERS_LICENSE_STATE',

    # Additional Drops
    'CMRC_VEH_I', 'TRAVEL_DIRECTION', 'TRAILER1_LENGTH', 'TRAILER2_LENGTH',␣
 ↪'TOTAL_VEHICLE_LENGTH', 'AXLE_CNT', 'LOAD_TYPE', 'HAZMAT_CLASS', 'SEAT_NO',␣
 ↪'DRIVERS_LICENSE_CLASS', 'HOSPITAL', 'EMS_AGENCY', 'PEDPEDAL_ACTION',␣
 ↪'PEDPEDAL_VISIBILITY', 'PEDPEDAL_LOCATION', 'BAC_RESULT VALUE',␣
 ↪'CELL_PHONE_USE', 'CMV_ID', 'TOWED_I', 'FIRE_I', 'DAMAGE'
]

# Drop columns
data.drop(columns=categorical_drop, errors='ignore', inplace=True)
```

3.8 Data Preparation

3.8.1 Remove Features with High Rate of Missing Values

- Drop features with high rate of missing values.

[36]:
```
# calculate null percentages
null_percentage = data.isnull().mean() * 100

# drop columns with more than 50% missing values
columns_to_drop = null_percentage[null_percentage > 50].index
data = data.drop(columns=columns_to_drop)
```

3.8.2 Impute Missing Values

- Median imputation for numerical features.
- Mode imputation for categorical features.

[37]:
```
# fill categorical columns with mode
for column in data.select_dtypes(include='object').columns:
    mode_value = data[column].mode()[0]
    data[column] = data[column].fillna(mode_value)
```

```
# fill numerical columns with median
for column in data.select_dtypes(include=['float64', 'int64']).columns:
    median_value = data[column].median()
    data[column] = data[column].fillna(median_value)
```

3.9 Feature Engineering

[38]:
```
# simplify contact point to FRONT, SIDE, REAR, OTHER
data['FIRST_CONTACT_POINT'] = data['FIRST_CONTACT_POINT'].apply(
    lambda x: 'Front' if 'FRONT' in x.upper() else (
        'Side' if 'SIDE' in x.upper() else (
            'Rear' if 'REAR' in x.upper() else 'Other'
        )
    )
)

# create RUSH_HOUR feature
data['CRASH_HOUR'] = pd.to_datetime(data['CRASH_HOUR'], format='%H').dt.hour
data['RUSH_HOUR'] = data['CRASH_HOUR'].apply(
    lambda x: 1 if 7 <= x <= 9 or 16 <= x <= 18 else 0
)

# create DAYLIGHT feature from LIGHTING_CONDITION and CRASH_HOUR
data['LIGHTING_CONDITION'] = data['LIGHTING_CONDITION'].replace({'DARKNESS,␣
 ↪LIGHTED ROAD': 'DARKNESS', 'DARKNESS, ROADWAY NOT LIGHTED': 'DARKNESS'})
data['DAYLIGHT'] = data['LIGHTING_CONDITION'].apply(
    lambda x: 1 if x == 'DAYLIGHT' else 0
)

data.drop(columns=['LIGHTING_CONDITION', 'CRASH_HOUR'], inplace=True)

# convert VEHICLE_YEAR to OLD, NEW
data['VEHICLE_YEAR'] = data['VEHICLE_YEAR'].apply(
    lambda x: 'Old' if x < 2010 else 'New'
)

# bin AGE into groups - child, young, middle, old
data['AGE'] = data['AGE'].apply(
    lambda x: 'Child' if x < 18 else (
        'Young' if 18 <= x < 30 else (
            'Middle' if 30 <= x < 60 else 'Old'
        )
    )
)

# bin OCCUPANT_CNT into groups - solo, small, medium, large
```

```
data['OCCUPANT_CNT'] = data['OCCUPANT_CNT'].apply(
    lambda x: 'Solo' if x == 1 else (
        'Small' if 2 <= x <= 4 else (
            'Medium' if 5 <= x <= 7 else 'Large'
        )
    )
)

# POSTED_SPEED_LIMIT          46

# simplify vehicle types
data['VEHICLE_TYPE'] = data['VEHICLE_TYPE'].replace({
    'TRUCK - SINGLE UNIT': 'Truck',
    'TRACTOR W/ SEMI-TRAILER': 'Truck',
    'TRACTOR W/O SEMI-TRAILER': 'Truck',
    'SINGLE UNIT TRUCK WITH TRAILER': 'Truck',
    'OTHER VEHICLE WITH TRAILER': 'Truck',
    'BUS OVER 15 PASS.': 'Bus',
    'BUS UP TO 15 PASS.': 'Bus',
    'MOTORCYCLE (OVER 150CC)': 'Motorcycle',
    '3-WHEELED MOTORCYCLE (2 REAR WHEELS)': 'Motorcycle',
    'AUTOCYCLE': 'Motorcycle',
    'ALL-TERRAIN VEHICLE (ATV)': 'Motorcycle'
})

# group physical conditions
data['PHYSICAL_CONDITION'] = data['PHYSICAL_CONDITION'].replace({
    'IMPAIRED - ALCOHOL': 'Impaired',
    'IMPAIRED - DRUGS': 'Impaired',
    'IMPAIRED - ALCOHOL AND DRUGS': 'Impaired',
    'MEDICATED': 'Impaired',
    'EMOTIONAL': 'Other',
    'FATIGUED/ASLEEP': 'Other',
    'ILLNESS/FAINTED': 'Other',
    'HAD BEEN DRINKING': 'Impaired',
    'NORMAL': 'Normal',
    'UNKNOWN': 'Unknown'
})
```

3.9.1 Grouping Rare Feature Values

[39]:
```
# print columns with the most unique values
unique_values = data.nunique().sort_values(ascending=False)
print(unique_values.head(10))
```

```
POSTED_SPEED_LIMIT          46
PRIM_CONTRIBUTORY_CAUSE     40
SEC_CONTRIBUTORY_CAUSE      40
```

```
MANEUVER                    28
DRIVER_ACTION               20
TRAFFICWAY_TYPE             20
TRAFFIC_CONTROL_DEVICE      19
SAFETY_EQUIPMENT            19
FIRST_CRASH_TYPE            18
VEHICLE_DEFECT              17
dtype: int64
```

[40]:
```
# simplify all categorical features by grouping rare categories into a single␣
 ↪category by proportion
def simplify_all_categorical_features(df, rare_threshold=0.01,␣
 ↪new_category='OTHER'):
    for column in df.select_dtypes(include='object').columns:
        total = len(df)
        value_counts = df[column].value_counts()
        rare_categories = value_counts[value_counts / total < rare_threshold].
 ↪index
        df[column] = df[column].replace(rare_categories, new_category)
    return df

# data = simplify_all_categorical_features(data, rare_threshold=0.005)
```

[41]:
```
# checkpoint
data.to_csv('./checkpoint/data_post_feat_eng.csv', index=False)
```

[42]:
```
# load data
data = pd.read_csv('./checkpoint/data_post_feat_eng.csv')
```

3.10 Data Preprocessing & Feature Selection

3.10.1 Significance Testing

- Spearman correlation for numerical features.
- Cramer's V for categorical features.
- Chi-squared test for categorical target.
- ANOVA for numerical target.

[43]:
```
# separate numeric and categorical features
numeric_features = data.select_dtypes(include=['float64', 'int64']).columns.
 ↪tolist()
categorical_features = data.select_dtypes(include=['object', 'category']).
 ↪columns.tolist()
```

Spearman Correlation

[44]:
```
significant_features = {}
for feature in numeric_features:
    # calculate Spearman correlation and p-value
```

```
    corr, p_value = spearmanr(data[feature], data['MOST_SEVERE_INJURY'])
    significant_features[feature] = (corr, p_value)

# filter features with p-value < 0.05
significant_features = {k: v for k, v in significant_features.items() if v[1] <␣
 ↪0.05}

# display significant features
print("Significant Features (Spearman Correlation):")
for feature, (corr, p_value) in significant_features.items():
    print(f"{feature}: Correlation={corr:.4f}, P-value={p_value:.4e}")

# checkpoint
significant_features_df = pd.DataFrame(significant_features).T
significant_features_df.columns = ['Correlation', 'P-value']
significant_features_df.to_csv('./checkpoint/spearman.csv')
```

```
Significant Features (Spearman Correlation):
POSTED_SPEED_LIMIT: Correlation=0.0095, P-value=7.6187e-84
NUM_UNITS: Correlation=0.0333, P-value=0.0000e+00
MOST_SEVERE_INJURY: Correlation=1.0000, P-value=0.0000e+00
CRASH_DAY_OF_WEEK: Correlation=-0.0021, P-value=1.2434e-05
CRASH_MONTH: Correlation=0.0015, P-value=2.0181e-03
RUSH_HOUR: Correlation=-0.0118, P-value=7.2680e-129
DAYLIGHT: Correlation=-0.0218, P-value=0.0000e+00
```

[45]:
```
# drop features with p-value > 0.05
# data.drop(columns=[feature for feature in numeric_features if feature not in␣
 ↪significant_features], inplace=True)
```

ANOVA

[46]:
```
# ANOVA test for numerical features
from scipy.stats import f_oneway

anova_results = {}

for feature in numeric_features:
    # calculate ANOVA F-statistic and p-value
    groups = [group[1] for group in data.groupby('MOST_SEVERE_INJURY')[feature]]
    f_statistic, p_value = f_oneway(*groups)
    anova_results[feature] = (f_statistic, p_value)

# filter features with p-value < 0.05
anova_results = {k: v for k, v in anova_results.items() if v[1] < 0.05}

# display significant features
print("Significant Features (ANOVA Test):")
```

```
for feature, (f_statistic, p_value) in anova_results.items():
    print(f"{feature}: F-statistic={f_statistic:.4f}, P-value={p_value:.4e}")

# checkpoint results
anova_results_df = pd.DataFrame(anova_results).T
anova_results_df.columns = ['F-statistic', 'P-value']
anova_results_df.to_csv('./checkpoint/anova.csv', index=True)
```

```
/var/folders/zp/h7t69w7n1jvg_7vxjttlw77c0000gn/T/ipykernel_88313/1670838748.py:9
: ConstantInputWarning: Each of the input arrays is constant; the F statistic is
not defined or infinite
  f_statistic, p_value = f_oneway(*groups)
```

```
Significant Features (ANOVA Test):
POSTED_SPEED_LIMIT: F-statistic=425.0732, P-value=1.9404e-94
NUM_UNITS: F-statistic=12094.5171, P-value=0.0000e+00
MOST_SEVERE_INJURY: F-statistic=inf, P-value=0.0000e+00
CRASH_DAY_OF_WEEK: F-statistic=20.8532, P-value=4.9588e-06
CRASH_MONTH: F-statistic=11.5371, P-value=6.8222e-04
RUSH_HOUR: F-statistic=583.3168, P-value=7.2680e-129
DAYLIGHT: F-statistic=2003.5118, P-value=0.0000e+00
```

Cramer's V

[47]:
```
# convert MOST_SEVERE_INJURY variable to NON-FATAL and FATAL
data['MOST_SEVERE_INJURY'] = data['MOST_SEVERE_INJURY'].apply(lambda x:␣
 ↪'NON-FATAL' if x == 0 else 'FATAL')

# calculate Cramers V for categorical features
def cramers_v(x, y):
    confusion_matrix = pd.crosstab(x, y)
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k - 1) * (r - 1)) / (n - 1))
    rcorr = r - ((r - 1) ** 2) / (n - 1)
    kcorr = k - ((k - 1) ** 2) / (n - 1)
    return np.sqrt(phi2corr / min((kcorr - 1), (rcorr - 1)))

# compute correlation for categorical features
categorical_corr = pd.Series(index=categorical_features, dtype='float64')
for feature in categorical_features:
    categorical_corr[feature] = cramers_v(data[feature],␣
 ↪data['MOST_SEVERE_INJURY'])

# display correlations sorted by absolute value
print("Categorical Feature Correlations:")
print(categorical_corr.abs().sort_values(ascending=False))
```

```
# checkpoint results
categorical_corr.to_csv('./checkpoint/cramers_v.csv', index=True)
```

```
Categorical Feature Correlations:
FIRST_CRASH_TYPE           0.069679
PRIM_CONTRIBUTORY_CAUSE    0.060734
AIRBAG_DEPLOYED            0.053426
EJECTION                   0.051340
UNIT_TYPE                  0.046917
PERSON_TYPE                0.045784
PHYSICAL_CONDITION         0.038977
SEC_CONTRIBUTORY_CAUSE     0.038758
SAFETY_EQUIPMENT           0.032656
VEHICLE_TYPE               0.022371
TRAFFICWAY_TYPE            0.019669
DRIVER_ACTION              0.019166
MANEUVER                   0.018044
OCCUPANT_CNT               0.016459
FIRST_CONTACT_POINT        0.014872
TRAFFIC_CONTROL_DEVICE     0.013418
WEATHER_CONDITION          0.012191
DRIVER_VISION              0.010694
DEVICE_CONDITION           0.010670
ALIGNMENT                  0.009686
ROADWAY_SURFACE_COND       0.009171
AGE                        0.009143
SEX                        0.007889
VEHICLE_DEFECT             0.007852
ROAD_DEFECT                0.004391
VEHICLE_YEAR               0.002113
dtype: float64
```

Chi-Square

[48]:
```
# chi-squared test for categorical features
chi2_results = {}
for feature in categorical_features:
    # calculate chi-squared statistic and p-value
    contingency_table = pd.crosstab(data[feature], data['MOST_SEVERE_INJURY'])
    chi2, p_value, _, _ = chi2_contingency(contingency_table)
    chi2_results[feature] = (chi2, p_value)

# filter features with p-value < 0.05
chi2_results = {k: v for k, v in chi2_results.items() if v[1] < 0.05}

# display significant features
print("Significant Features (Chi-Squared Test):")
```

```
for feature, (chi2, p_value) in chi2_results.items():
    print(f"{feature}: Chi2={chi2:.4f}, P-value={p_value:.4e}")

# checkpoint results
chi2_results_df = pd.DataFrame(chi2_results).T
chi2_results_df.columns = ['Chi2', 'P-value']
chi2_results_df.to_csv('./checkpoint/chi2.csv', index=True)
```

```
Significant Features (Chi-Squared Test):
TRAFFIC_CONTROL_DEVICE: Chi2=776.4059, P-value=3.3233e-153
DEVICE_CONDITION: Chi2=486.5298, P-value=6.3022e-101
WEATHER_CONDITION: Chi2=637.0532, P-value=1.6486e-129
FIRST_CRASH_TYPE: Chi2=20467.9817, P-value=0.0000e+00
TRAFFICWAY_TYPE: Chi2=1648.5028, P-value=0.0000e+00
ALIGNMENT: Chi2=400.1789, P-value=2.7146e-84
ROADWAY_SURFACE_COND: Chi2=360.3035, P-value=9.4660e-75
ROAD_DEFECT: Chi2=87.2149, P-value=1.1469e-16
PRIM_CONTRIBUTORY_CAUSE: Chi2=15576.0863, P-value=0.0000e+00
SEC_CONTRIBUTORY_CAUSE: Chi2=6366.4773, P-value=0.0000e+00
UNIT_TYPE: Chi2=9279.9496, P-value=0.0000e+00
VEHICLE_YEAR: Chi2=19.8048, P-value=8.5765e-06
VEHICLE_DEFECT: Chi2=275.6791, P-value=2.7082e-49
VEHICLE_TYPE: Chi2=2121.0831, P-value=0.0000e+00
MANEUVER: Chi2=1398.3851, P-value=4.7498e-278
OCCUPANT_CNT: Chi2=1144.0131, P-value=1.0286e-247
FIRST_CONTACT_POINT: Chi2=934.6493, P-value=2.6990e-202
PERSON_TYPE: Chi2=8834.2204, P-value=0.0000e+00
SEX: Chi2=264.1638, P-value=4.3407e-58
AGE: Chi2=355.1475, P-value=1.1457e-76
SAFETY_EQUIPMENT: Chi2=4509.9457, P-value=0.0000e+00
AIRBAG_DEPLOYED: Chi2=12029.0714, P-value=0.0000e+00
EJECTION: Chi2=11106.3368, P-value=0.0000e+00
DRIVER_ACTION: Chi2=1566.2365, P-value=0.0000e+00
DRIVER_VISION: Chi2=494.7520, P-value=1.9052e-97
PHYSICAL_CONDITION: Chi2=6404.1592, P-value=0.0000e+00
```

Variance Inflation Factor

- Check for multicollinearity using VIF.

[49]:
```
# check for multicollinearity using VIF
def calculate_vif(data):
    vif_data = pd.DataFrame()
    vif_data["feature"] = data.columns
    vif_data["VIF"] = [variance_inflation_factor(data.values, i) for i in␣
 ↪range(data.shape[1])]
    return vif_data
```

```
# vif_features = data.drop(columns=['MOST_SEVERE_INJURY']).
 ↪select_dtypes(include=['float64', 'int64'])
vif_features = data.select_dtypes(include=['float64', 'int64'])


# display VIF values
print("VIF Values:")
print(calculate_vif(vif_features))
```

```
VIF Values:
              feature        VIF
0  POSTED_SPEED_LIMIT  13.621685
1           NUM_UNITS   9.980176
2   CRASH_DAY_OF_WEEK   4.849806
3         CRASH_MONTH   4.535400
4           RUSH_HOUR   1.612996
5            DAYLIGHT   2.815999
```

3.10.2 VIF Results

- Features with high VIF (>= 10) will be dropped.
  - POSTED_SPEED_LIMIT
  - NUM_UNITS

[50]:
```
# drop POSTED_SPEED_LIMIT and NUM_UNITS
data.drop(columns=['POSTED_SPEED_LIMIT', 'NUM_UNITS'], inplace=True,␣
 ↪errors='ignore')
```

[51]:
```
# check the strongest correlations among the high VIF features
strongest_correlations = {}
for feature in vif_features.columns:
    corr = vif_features.corrwith(vif_features[feature]).abs().
 ↪sort_values(ascending=False)
    strongest_correlations[feature] = corr[corr.index != feature].head(5)

# display strongest correlations
print("Strongest Correlations:")
for feature, correlations in strongest_correlations.items():
    print(f"{feature}:")
    print(correlations)
    print()
```

```
Strongest Correlations:
POSTED_SPEED_LIMIT:
NUM_UNITS            0.061098
DAYLIGHT             0.037010
CRASH_MONTH          0.010327
CRASH_DAY_OF_WEEK    0.006360
```

```
RUSH_HOUR            0.004616
dtype: float64

NUM_UNITS:
POSTED_SPEED_LIMIT    0.061098
DAYLIGHT              0.045542
RUSH_HOUR             0.018429
CRASH_MONTH           0.003931
CRASH_DAY_OF_WEEK     0.002289
dtype: float64

CRASH_DAY_OF_WEEK:
RUSH_HOUR             0.010210
POSTED_SPEED_LIMIT    0.006360
DAYLIGHT              0.005326
CRASH_MONTH           0.003866
NUM_UNITS             0.002289
dtype: float64

CRASH_MONTH:
DAYLIGHT              0.043896
POSTED_SPEED_LIMIT    0.010327
NUM_UNITS             0.003931
CRASH_DAY_OF_WEEK     0.003866
RUSH_HOUR             0.003141
dtype: float64

RUSH_HOUR:
DAYLIGHT              0.176756
NUM_UNITS             0.018429
CRASH_DAY_OF_WEEK     0.010210
POSTED_SPEED_LIMIT    0.004616
CRASH_MONTH           0.003141
dtype: float64

DAYLIGHT:
RUSH_HOUR             0.176756
NUM_UNITS             0.045542
CRASH_MONTH           0.043896
POSTED_SPEED_LIMIT    0.037010
CRASH_DAY_OF_WEEK     0.005326
dtype: float64
```

Drop Features with Low Significance

- Drop features with p-value > 0.05.

[52]:
```
# drop low relevance features based on anova, chi2, spearman, and cramers_v
# data.drop(columns=[
    # 'VEHICLE_DEFECT',
    # 'ALIGNMENT',
    # 'Divided_Trafficway',
    # 'VEHICLE_YEAR',
    # 'Adverse_Weather',
    # 'CRASH_DAY_BINARY',
    # 'TRAFFICWAY_TYPE',
    # 'DRIVER_ACTION',
    # 'MANEUVER',
    # 'ROADWAY_SURFACE_COND',
    # 'LIGHTING_CONDITION',
    # 'SEC_CAUSE_GROUP',
    # 'ROAD_DEFECT',
    # 'CRASH_MONTH',
    # 'CRASH_HOUR',
    # 'CRASH_DAY_OF_WEEK',
    # 'WEATHER_CONDITION',
    # 'OLD_VEHICLE',
    # 'DRIVER_VISION'
    # ], errors='ignore', inplace=True)
```

[53]:
```
# print the number of features before dropping
print("Number of features after dropping:", data.shape[1])
```

```
Number of features after dropping: 31
```

[54]:
```
# checkpoint data
data.to_csv('./checkpoint/data_post_feat_sel.csv', index=False)
```

4 Modeling

4.1 Test Train Split

[98]:
```
# load data
data = pd.read_csv('./checkpoint/data_post_feat_sel.csv')
y = data['MOST_SEVERE_INJURY']
X = data.drop(columns=['MOST_SEVERE_INJURY'])
```

[99]:
```
# check if there are any missing values
missing_values = X.isnull().sum()
print("Missing Values:")
print(missing_values[missing_values > 0])

print(y.value_counts())
```

```
Missing Values:
```

```
Series([], dtype: int64)
MOST_SEVERE_INJURY
NON-FATAL    4205118
FATAL           7036
Name: count, dtype: int64
```

[100]:
```
# align indices of X and y, drop any rows with NaNs in either, and perform a␣
↪train-test split
def train_test_split_wrapper(data, y):
    # align indices of data and y
    data, y = data.align(y, join="inner", axis=0)

    # drop rows with NaNs in either the features or the target
    combined = pd.concat([data, y], axis=1)
    combined = combined.dropna()

    # split the cleaned data and target
    data_cleaned = combined.iloc[:, :-1]  # All columns except the last␣
↪(features)
    y_cleaned = combined.iloc[:, -1]  # Last column (target)

    # train-test split
    X_train, X_test, y_train, y_test = train_test_split(
        data_cleaned, y_cleaned, test_size=0.2, random_state=42,␣
↪stratify=y_cleaned
    )

    return X_train, X_test, y_train, y_test
```

[101]:
```
# perform train test split
X_train, X_test, y_train, y_test = train_test_split_wrapper(X, y)
```

[102]:
```
y_train.value_counts()
```

[102]:
```
MOST_SEVERE_INJURY
NON-FATAL    3364094
FATAL           5629
Name: count, dtype: int64
```

4.2 Preprocessing

- One-hot encoding for categorical features.
- Standard scaling for numerical features.

[103]:
```
def preprocess_data(df, scaler=None, encoder=None, train=True):
    # identify numeric & categorical features
    numeric_features = df.select_dtypes(include=['float64', 'int64']).columns.
↪tolist()
```

```
    categorical_features = df.select_dtypes(include=['object', 'category']).
↪columns.tolist()

    # scale numeric features
    if train:
        scaler = StandardScaler()
        df_numeric_scaled = pd.DataFrame(scaler.
↪fit_transform(df[numeric_features]),
                                         columns=numeric_features, index=df.
↪index)
    else:
        if scaler is None:
            raise ValueError("Scaler cannot be None when train=False")
        df_numeric_scaled = pd.DataFrame(scaler.transform(df[numeric_features]),
                                         columns=numeric_features, index=df.
↪index)

    # encode categorical features
    if train:
        encoder = OneHotEncoder(handle_unknown='ignore', drop='first',␣
↪sparse_output=False)
        df_categorical_encoded = pd.DataFrame(encoder.
↪fit_transform(df[categorical_features]),
                                              columns=encoder.
↪get_feature_names_out(categorical_features),
                                              index=df.index)
    else:
        if encoder is None:
            raise ValueError("Encoder cannot be None when train=False")
        df_categorical_encoded = pd.DataFrame(encoder.
↪transform(df[categorical_features]),
                                              columns=encoder.
↪get_feature_names_out(categorical_features),
                                              index=df.index)

    # combine processed features
    df_processed = pd.concat([df_numeric_scaled, df_categorical_encoded],␣
↪axis=1)

    return df_processed, scaler, encoder
```

[104]:
```
# ensure scaler & encoder are passed for test data
X_train_processed, scaler, encoder = preprocess_data(X_train, train=True)
X_test_processed, _, _ = preprocess_data(X_test, scaler=scaler,␣
↪encoder=encoder, train=False)
```

5 Baseline Logistic Regression Model

Logistic Regression Pros: - Fast - Interpretable - Works well with binary classification Cons: - May not capture complex relationships - May not perform well with imbalanced classes

- Random Undersampling

[105]:
```
# randomly under-sample the majority class
rus = RandomUnderSampler(random_state=42)
X_train_processed_rus, y_train_rus = rus.fit_resample(X_train_processed,␣
 ↪y_train)

# train baseline model
model = LogisticRegression(
    max_iter=1000,
    solver='saga',
    random_state=42
)

# fit the model
model.fit(X_train_processed_rus, y_train_rus)
```

[105]:
```
LogisticRegression(max_iter=1000, random_state=42, solver='saga')
```

5.0.1 Regression Coefficients

- Identify features with high impact on classifying Fatal/Non-Fatal injuries.

[106]:
```
# get regression coefficients
feature_importance = pd.DataFrame(
    {"Feature": X_train_processed.columns, "Coefficient": np.abs(model.
 ↪coef_[0])}
).sort_values("Coefficient", ascending=False)

# features with the highest coefficients (>= 1)
log_reg_coefficients = feature_importance[feature_importance["Coefficient"] >=␣
 ↪1]

# print features with the highest coefficients (>= 1)
print("Features with the highest coefficients:")
print(log_reg_coefficients)

# checkpoint feature importance
log_reg_coefficients.to_csv('./checkpoint/log_reg_coefficients.csv',␣
 ↪index=False)
```

```
Features with the highest coefficients:
                                               Feature  Coefficient
122  PRIM_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF …     3.155665
```

107	PRIM_CONTRIBUTORY_CAUSE_EXCEEDING AUTHORIZED S…	2.982513
48	FIRST_CRASH_TYPE_PEDESTRIAN	2.529505
40	FIRST_CRASH_TYPE_ANIMAL	2.401920
277	EJECTION_TOTALLY EJECTED	2.072371
17	TRAFFIC_CONTROL_DEVICE_SCHOOL ZONE	1.994622
74	TRAFFICWAY_TYPE_UNKNOWN INTERSECTION TYPE	1.950277
199	VEHICLE_TYPE_Motorcycle	1.899112
278	EJECTION_TRAPPED/EXTRICATED	1.890950
144	SEC_CONTRIBUTORY_CAUSE_EQUIPMENT - VEHICLE CON…	1.839812
52	FIRST_CRASH_TYPE_REAR TO SIDE	1.835741
101	PRIM_CONTRIBUTORY_CAUSE_DISTRACTION - FROM OUT…	1.785528
75	TRAFFICWAY_TYPE_Y-INTERSECTION	1.776450
71	TRAFFICWAY_TYPE_T-INTERSECTION	1.733798
69	TRAFFICWAY_TYPE_RAMP	1.652816
120	PRIM_CONTRIBUTORY_CAUSE_OPERATING VEHICLE IN E…	1.591822
313	PHYSICAL_CONDITION_OTHER	1.575172
153	SEC_CONTRIBUTORY_CAUSE_IMPROPER LANE USAGE	1.558567
104	PRIM_CONTRIBUTORY_CAUSE_DRIVING SKILLS/KNOWLED…	1.556687
207	VEHICLE_TYPE_UNKNOWN/NA	1.473260
270	AIRBAG_DEPLOYED_DEPLOYED, COMBINATION	1.403889
63	TRAFFICWAY_TYPE_L-INTERSECTION	1.343677
138	SEC_CONTRIBUTORY_CAUSE_DISREGARDING YIELD SIGN	1.339552
22	DEVICE_CONDITION_FUNCTIONING PROPERLY	1.299803
150	SEC_CONTRIBUTORY_CAUSE_FOLLOWING TOO CLOSELY	1.293462
155	SEC_CONTRIBUTORY_CAUSE_IMPROPER TURNING/NO SIGNAL	1.272595
7	TRAFFIC_CONTROL_DEVICE_NO CONTROLS	1.253819
197	VEHICLE_TYPE_MOPED OR MOTORIZED BICYCLE	1.234333
68	TRAFFICWAY_TYPE_PARKING LOT	1.215702
89	ROAD_DEFECT_RUT, HOLES	1.214324
113	PRIM_CONTRIBUTORY_CAUSE_IMPROPER BACKING	1.214236
116	PRIM_CONTRIBUTORY_CAUSE_IMPROPER TURNING/NO SI…	1.214067
238	OCCUPANT_CNT_Solo	1.205926
274	AIRBAG_DEPLOYED_DID NOT DEPLOY	1.201076
142	SEC_CONTRIBUTORY_CAUSE_DRIVING ON WRONG SIDE/W…	1.200413
131	PRIM_CONTRIBUTORY_CAUSE_WEATHER	1.181457
57	TRAFFICWAY_TYPE_CENTER TURN LANE	1.164641
95	PRIM_CONTRIBUTORY_CAUSE_DISREGARDING OTHER TRA…	1.154113
84	ROADWAY_SURFACE_COND_SNOW OR SLUSH	1.141065
143	SEC_CONTRIBUTORY_CAUSE_DRIVING SKILLS/KNOWLEDG…	1.135597
54	FIRST_CRASH_TYPE_SIDESWIPE SAME DIRECTION	1.124305
103	PRIM_CONTRIBUTORY_CAUSE_DRIVING ON WRONG SIDE/…	1.121768
53	FIRST_CRASH_TYPE_SIDESWIPE OPPOSITE DIRECTION	1.101175
178	UNIT_TYPE_PEDESTRIAN	1.100233
51	FIRST_CRASH_TYPE_REAR TO REAR	1.043373
29	WEATHER_CONDITION_BLOWING SNOW	1.037399
92	ROAD_DEFECT_WORN SURFACE	1.026625
265	SAFETY_EQUIPMENT_SAFETY BELT USED	1.022977
114	PRIM_CONTRIBUTORY_CAUSE_IMPROPER LANE USAGE	1.017793

```
81                          ROADWAY_SURFACE_COND_ICE     1.006277
62                          TRAFFICWAY_TYPE_FOUR WAY     1.000895
```

Findings

- **Driver's Physical Condition Significantly Affects Crash Severity**
  - PRIM_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER (**Coefficient:** 3.04)

  Actionable Steps:
  - Implement roadside impairment tests using AI-powered assessment tools.
  - Develop awareness campaigns on the risks of driving with medical impairments.

- **Speeding is a Major Predictor of Severe Crashes**
  - PRIM_CONTRIBUTORY_CAUSE_EXCEEDING AUTHORIZED SPEED LIMIT (**Coefficient:** 2.79)

  Actionable Steps:
  - Expand speed camera enforcement in high-risk areas.
  - Implement dynamic speed limits based on traffic and weather conditions.
  - Increase public awareness campaigns about the dangers of excessive speed.

- **Pedestrian, Cyclist, Animal Crashes Have High Severity**
  - FIRST_CRASH_TYPE_PEDESTRIAN (**Coefficient:** 2.55)
  - FIRST_CRASH_TYPE_PEDALCYCLIST (**Coefficient:** 1.39)
  - FIRST_CRASH_TYPE_ANIMAL (**Coefficient:** 2.428)

  Actionable Steps:
  - Improve car traffic isolation from bike lanes and pedestrian crossings. - Implement wildlife detection systems to alert drivers of animal crossings. - Increase visibility and signage at pedestrian and cyclist crossings. - Implement wildlife crossing deterrents (eg. fencing) in high-risk areas.

- **Dangerous Roadway Conditions Increase Crash Severity**
  - ROAD_DEFECT_WORN SURFACE (**Coefficient:** 0.97)
  - ROAD_DEFECT_RUT, HOLES (**Coefficient:** 0.51)
  - ROADWAY_SURFACE_COND_SNOW OR SLUSH (**Coefficient:** 0.61)

  Actionable Steps:
  - Prioritize road maintenance funding for fixing potholes and worn surfaces.
  - Implement real-time road hazard alerts using IoT and traffic apps.
  - Expand pre-winter road treatment programs to reduce ice-related crashes.

- **High-Risk Intersection Designs Need Attention**
  - TRAFFICWAY_TYPE_Y-INTERSECTION (**Coefficient:** 1.28)

  - TRAFFICWAY_TYPE_T-INTERSECTION (**Coefficient:** 1.50)

  - TRAFFICWAY_TYPE_UNKNOWN INTERSECTION TYPE (**Coefficient:** 1.93)
  - TRAFFICWAY_TYPE_L-INTERSECTION (**Coefficient:** 1.34)

  Actionable Steps:
  - Improve signage and lane markings for better driver guidance.
  - Deploy collision-prevention warning systems at intersections.
  - Implement red-light cameras at high-risk intersections. - Implement traffic slowing measures at non-standard or high-risk intersections.

- **Distracted and Reckless Driving Remains a Major Threat**
  - PRIM_CONTRIBUTORY_CAUSE_DISTRACTION - FROM OUTSIDE VEHICLE (**Coefficient:** 1.26)

  - PRIM_CONTRIBUTORY_CAUSE_OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESSIVE MANNER (**Coefficient:** 1.66)

  - PRIM_CONTRIBUTORY_CAUSE_CELL PHONE USE OTHER THAN TEXTING (**Coefficient:** 0.42)

  Actionable Steps:
  - Implement stricter penalties for distracted and reckless driving.
  - Launch public awareness campaigns on the dangers of distracted driving.
  - Install CCTV cameras at high-risk intersections to monitor driver behavior and issue fines.

5.1 Evaluation

[107]:
```
def evaluate_model(model, X, y, X_test, y_test, output_path='./checkpoint/
↪evaluation_metrics.json', cv_folds=5):
    # classification report
    print("Classification Report:")
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)

    report = classification_report(y_test, y_pred)
    print(report)

    # AUC-ROC
    roc_auc = roc_auc_score(pd.get_dummies(y_test).values, y_pred_proba,␣
↪multi_class="ovr")
    print(f"AUC-ROC: {roc_auc:.4f}")
    # confusion matrix
    print("Confusion Matrix:")
    cm = confusion_matrix(y_test, y_pred)
    print(cm)

    # cross-validation
```

```
    # print("\nRunning Cross-Validation...")
    # cv = StratifiedKFold(n_splits=cv_folds, shuffle=True, random_state=42)
    # cv_scores = cross_val_score(model, X, y, cv=cv, scoring='accuracy')

    # print cross validation scores
    # print("Cross-Validation Scores:")
    # print(cv_scores)

    # print cross-validation accuracy
    # print(f"Cross-Validation Accuracy: {cv_scores.mean():.4f} ± {cv_scores.
↪std():.4f}")

    # save evaluation metrics
    evaluation_metrics = {
        'Classification Report': classification_report(y_test, y_pred,␣
↪output_dict=True),
        'AUC-ROC': roc_auc,
        'Confusion Matrix': cm.tolist(),
        #'Cross-Validation Accuracy': {
        #    'mean': cv_scores.mean(),
        #    'std_dev': cv_scores.std(),
        #    'scores': cv_scores.tolist()
        #}
    }

    with open(output_path, 'w') as f:
        json.dump(evaluation_metrics, f, indent=4)

    return evaluation_metrics
```

[108]:
```
# evaluate the baseline model
evaluation_metrics = evaluate_model(model, X_train_processed, y_train,␣
↪X_test_processed, y_test, output_path='./checkpoint/
↪evaluation_metrics_baseline.json')
```

```
Classification Report:
              precision    recall  f1-score   support

       FATAL       0.01      0.88      0.02      1407
   NON-FATAL       1.00      0.88      0.93    841024

    accuracy                           0.88    842431
   macro avg       0.51      0.88      0.48    842431
weighted avg       1.00      0.88      0.93    842431

AUC-ROC: 0.9388
Confusion Matrix:
[[  1234    173]
```

```
[103429 737595]]
```

5.2 Baseline Evaluation

- Struggles with class imbalance, poor precision and recall for fatal class.
- Likely overfitting and will be unable to generalize to new data.
- AUC-ROC score indicates strong discriminatory power.
  - With low precision suggests that threshold tuning may be needed.

5.3 Baseline Tuning

- Threshold tuning to improve precision and recall.

[109]:
```
# train baseline model
model = LogisticRegression(
    # class_weight={'NON-FATAL': 1, 'FATAL': 10},
    max_iter=500,
    solver='saga',
    class_weight={'NON-FATAL': 1, 'FATAL': 5},
    random_state=42
)

# fit the model
model.fit(X_train_processed_rus, y_train_rus)

# evaluate the baseline model
evaluation_metrics = evaluate_model(model, X_train_processed, y_train,␣
 ↪X_test_processed, y_test, output_path='./checkpoint/
 ↪evaluation_metrics_baseline_weighted.json')
```

```
Classification Report:
              precision    recall  f1-score   support

       FATAL       0.01      0.97      0.01      1407
   NON-FATAL       1.00      0.72      0.84    841024

    accuracy                           0.72    842431
   macro avg       0.50      0.84      0.42    842431
weighted avg       1.00      0.72      0.84    842431

AUC-ROC: 0.9389
Confusion Matrix:
[[  1363     44]
 [235444 605580]]
```

5.4 Tuned Logistic Regression Evaluation

- Better:
  - captures as many fatal crashes as possible (high recall)

- Worse:
  - fewer false positives and better overall accuracy
  - more false negatives and lower precision

Ultimately, logistic regression is not the best model for this task due to class imbalance and the need for better generalization.

6 Baseline Decision Tree

- Use a decision tree classifier Pros: - Handles non-linear relationships capturing interactions and feature importance. - Works better with imbalanced data. - More flexible than logistic regression. Cons: - May overfit the training data.

[110]:
```
# train the model
model = DecisionTreeClassifier(
    class_weight='balanced',
    random_state=42,
    max_depth=5
)

# fit the model
model.fit(X_train_processed_rus, y_train_rus)

# evaluate the model
evaluation_metrics = evaluate_model(model, X_train_processed, y_train,␣
 ↪X_test_processed, y_test, output_path='./checkpoint/
 ↪evaluation_metrics_decision_tree.json')
```

```
Classification Report:
              precision    recall  f1-score   support

       FATAL       0.01      0.73      0.01      1407
   NON-FATAL       1.00      0.83      0.91    841024

    accuracy                           0.83    842431
   macro avg       0.50      0.78      0.46    842431
weighted avg       1.00      0.83      0.91    842431

AUC-ROC: 0.8126
Confusion Matrix:
[[  1024    383]
 [143369 697655]]
```

6.1 Baseline Decision Tree Evaluation

- Worse:
  - Misses more actual fatal crashes, making it less reliable for high-risk events
  - Decision Tree has higher false negatives (383 missed FATAL crashes)
  - AUC-ROC is significantly lower, indicating weaker discriminatory power

The model is largely ineffective at predicting fatal crashes and has not improved significantly over the logistic regression model.

7 Tuned Decision Tree

- Tune the decision tree model to improve performance.
  - Increase max depth to capture more complex relationships.

[111]:
```
# train the model
model = DecisionTreeClassifier(
    class_weight='balanced',
    random_state=42,
    max_depth=20,
    min_samples_split=10,
    min_samples_leaf=2
)

# fit the model
model.fit(X_train_processed_rus, y_train_rus)

# evaluate the model
evaluation_metrics = evaluate_model(model, X_train_processed, y_train,␣
 ↪X_test_processed, y_test, output_path='./checkpoint/
 ↪evaluation_metrics_decision_tree_tuned.json')
```

```
Classification Report:
              precision    recall  f1-score   support

       FATAL       0.01      0.87      0.03      1407
   NON-FATAL       1.00      0.89      0.94    841024

    accuracy                           0.89    842431
   macro avg       0.51      0.88      0.48    842431
weighted avg       1.00      0.89      0.94    842431

AUC-ROC: 0.9017
Confusion Matrix:
[[  1219    188]
 [ 94275 746749]]
```

7.1 Evaluation Results

- Decision Tree Classifier performs better than the baseline model.
  - ROC-AUC: Logistic Regression is slightly better at distinguishing between classes.
  - Weighted Average F1 Score: Decision Tree Classifier is better at predicting injury severity.
  - Accuracy: Decision Tree is overall better at predicting injury severity.

- Logistic Regression suffers from class imbalance and is not able to predict injury severity effectively.

Logistic Regression: - It heavily predicts the majority class: - Fatal injuries (FATAL): Poor recall (69%), likely due to underrepresentation. - Struggles to separate "Injury" classes (e.g., Incapacitating vs. Non-incapacitating). Decision Tree: - Better balance in predictions, but slightly worse AUC-ROC. - Higher recall for all injury types, meaning it captures more true injuries. - Slightly more false positives in injury-related classes, which could be tuned.

8 Tuning

- Hyperparameter tuning for Decision Tree Classifier.

[112]:
```
# train the model
model = DecisionTreeClassifier(
    # class_weight={'NON-FATAL': 1, 'FATAL': 10},
    class_weight='balanced',
    random_state=42,
    max_depth=20,
    min_samples_split=10,
    min_samples_leaf=2
)

# fit the model
model.fit(X_train_processed_rus, y_train_rus)

# evaluate the model
evaluation_metrics = evaluate_model(model, X_train_processed, y_train,␣
 ↪X_test_processed, y_test, output_path='./checkpoint/
 ↪evaluation_metrics_decision_tree_tuned.json')
```

```
Classification Report:
              precision    recall  f1-score   support

       FATAL       0.01      0.87      0.03      1407
   NON-FATAL       1.00      0.89      0.94    841024

    accuracy                           0.89    842431
   macro avg       0.51      0.88      0.48    842431
weighted avg       1.00      0.89      0.94    842431

AUC-ROC: 0.9017
Confusion Matrix:
[[  1219    188]
 [ 94275 746749]]
```

[113]:
```
# get model coefficients
feature_importance = pd.DataFrame(
```

```
    {"Feature": X_train_processed.columns, "Importance": model.
 ↪feature_importances_}
).sort_values("Importance", ascending=False)

# features with the highest importance
decision_tree_importance = feature_importance[feature_importance["Importance"]␣
 ↪> 0]

# print features with the highest importance
print("Features with the highest importance:")
print(decision_tree_importance)
```

```
Features with the highest importance:
                                          Feature  Importance
270          AIRBAG_DEPLOYED_DEPLOYED, COMBINATION    0.153815
48                     FIRST_CRASH_TYPE_PEDESTRIAN    0.127106
271                AIRBAG_DEPLOYED_DEPLOYED, FRONT    0.070910
242                             PERSON_TYPE_DRIVER    0.048126
265              SAFETY_EQUIPMENT_SAFETY BELT USED    0.042292
..                                             …         …
38                         WEATHER_CONDITION_SNOW    0.000102
136  SEC_CONTRIBUTORY_CAUSE_DISREGARDING STOP SIGN    0.000094
288                 DRIVER_ACTION_IMPROPER PASSING    0.000072
261                   SAFETY_EQUIPMENT_HELMET USED    0.000065
82                      ROADWAY_SURFACE_COND_OTHER    0.000044

[148 rows x 2 columns]
```

9 Tuned Decision Tree Evaluation

- Better:
  - Offers a better balance between precision and recall for all classes.
- Worse:
  - AUC-ROC is significantly lower than regression models.

10 Conclusion

- Class imbalance continues to be a challenge in predicting the minority class (FATAL injuries).
- Decision Tree Classifier offers a better balance between precision and recall for all classes.
- Logistic Regression struggles with class imbalance and is not able to predict injury severity effectively.
  - But it has a higher AUC-ROC score, indicating better discriminatory power.

Choosing a model depends on the stakeholder's priorities: - Capturing the most fatal crashes -> the Baseline Weighted model. - Balancing accuracy and precision -> the Baseline model. - Minimizing false positives and maximizing recall -> the Tuned Decision Tree model.