Open in Colab

Road Accidents Fatality Prediction in Kenya

Project Overview

The goal of this project is to build a machine learning model that predicts the probability of fatal outcomes in road crashes using historical crash data. Insights from this model will help transportation agencies, public safety departments, and urban planners develop data-driven interventions to reduce fatalities.

Main Objective(s)

- Analyze historical crash data from 2012-2023 to identify fatality patterns.
- Train a machine learning model to predict fatality likelihood in road crashes.
- Deploy a web-based interface for real-time fatality risk prediction.

Specific Objectives

- Identify key factors influencing fatality risks (e.g., location, time of day).
- Provide data-driven insights to support road safety campaigns and infrastructure planning

1 Business Understanding

Road traffic accidents are a significant public safety concern in Kenya, contributing to high fatality rates. Effective measures to reduce fatalities require identifying the factors that increase the likelihood of death in accidents.The goal of this project is to build a machine learning model that predicts the probability of fatal outcomes in road crashes using historical crash data. Insights from this model will help transportation agencies, public safety departments, and urban planners develop data-driven interventions to reduce fatalities.

1.2 Stakeholders

- Transportation agencies i.e Supermetro sacco, Utimo sacco
- Public safety departments i.e NTSA
- Urban planners i.e KURA, KENHA
- Cyclists and Pedestrians
- Private car-owners

2 Data Understanding

The project will utilize crash data collected from Kenya's road accidents between 2012-2023, sourced from the World Bank microdata platform. The dataset contains multiple features that describe road crashes and their outcomes. Dataset Overview:

- Crash Date and Time: When the crash occurred (time of day, date).
- Location: Geographic details such as latitude, longitude.
- Crash Description Keywords: Keywords that describe the nature of the crash (e.g., "fatality," "pedestrian," "motorcycle").
- Crash Outcome: Binary indicator (fatal or non-fatal).

2.1 Importing the necessary libraries

```
import pandas as pd
import numpy as np
#Data plotting
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
# from sklearn
from xgboost import XGBClassifier
import sklearn.metrics as metrics
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.ensemble  import GradientBoostingClassifier,RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report, confusio
from statsmodels.stats.outliers_influence import variance_inflation_factor
!pip install streamlit
import streamlit as st
import joblib
```

```
Collecting streamlit
  Downloading streamlit-1.40.0-py2.py3-none-any.whl.metadata (8.5 kB)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (4.2.2)
Requirement already satisfied: blinker<2,>=1.0.0 in /usr/lib/python3/dist-packages (from streamlit) (1.4)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (5.5.0)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (8.1.7)
Requirement already satisfied: numpy<3,>=1.20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (1.26.4)
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (24.1)
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (2.2.2)
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (10.4.0)
Requirement already satisfied: protobuf<6,>=3.20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (3.20.3)
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (17.0.0)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.10/dist-packages (from streamlit) (2.32.3)
Requirement already satisfied: rich<14,>=10.14.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (13.9.3)
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (9.0.0)
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (from streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.3.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (4.12.2)
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in /usr/local/lib/python3.10/dist-packages (from streamlit) (3.1.43
Collecting pydeck<1,>=0.8.0b4 (from streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local/lib/python3.10/dist-packages (from streamlit) (6.3.3)
Collecting watchdog<6,>=2.1.5 (from streamlit)
  Downloading watchdog-5.0.3-py3-none-manylinux2014_x86_64.whl.metadata (41 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41.9/41.9 kB 3.3 MB/s eta 0:00:00
Requirement already satisfied: entrypoints in /usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0->streamlit) (0.4)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0->streamlit) (3.1.4)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0->streamlit) (4.2
Requirement already satisfied: toolz in /usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0->streamlit) (0.12.1)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.10/dist-packages (from gitpython!=3.1.19,<4,>=3.0.7->s
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1.4.0->stream
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1.4.0->streamlit) (2024
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1.4.0->streamlit) (20
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->str
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->streamlit) (3.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->streamlit
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->streamlit
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich<14,>=10.14.0->stream
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich<14,>=10.14.0->stre
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from gitdb<5,>=4.0.1->gitpython!=3.1
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->altair<6,>=4.0->streaml
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair<6,>
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair<6,>=4.0-
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich<14,>=10.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas<3,>=1.4
Downloading streamlit-1.40.0-py2.py3-none-any.whl (8.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.6/8.6 MB 88.7 MB/s eta 0:00:00
Downloading pydeck-0.9.1-py2.py3-none-any.whl (6.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 93.4 MB/s eta 0:00:00
Downloading watchdog-5.0.3-py3-none-manylinux2014_x86_64.whl (79 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.3/79.3 kB 6.9 MB/s eta 0:00:00
Installing collected packages: watchdog, pydeck, streamlit
Successfully installed pydeck-0.9.1 streamlit-1.40.0 watchdog-5.0.3
```

LOADING THE DATASET

```
data = pd.read_csv('ma3route_crashes_algorithmcode.csv')
data.head(20)
```

	crash_id	crash_datetime	crash_date	latitude	longitude	n_crash_reports	contains_fatality_words	contains_pedestrian
0	1	06/06/2018 20:39	06/06/2018	-1.263030	36.764374	1	0	
1	2	17/08/2018 06:15	17/08/2018	-0.829710	37.037820	1	1	
2	3	25/05/2018 17:51	25/05/2018	-1.125301	37.003297	1	0	
3	4	25/05/2018 18:11	25/05/2018	-1.740958	37.129025	1	0	
4	5	25/05/2018 21:59	25/05/2018	-1.259392	36.842321	1	1	
5	6	26/05/2018 07:11	26/05/2018	-1.215499	36.835150	1	0	
6	7	26/05/2018 07:42	26/05/2018	-1.372556	36.920491	1	1	
7	8	26/05/2018 07:52	26/05/2018	-1.209940	36.833173	1	0	
8	9	26/05/2018 11:51	26/05/2018	-1.314351	36.807909	1	0	
9	10	26/05/2018 15:42	26/05/2018	-1.206788	36.854991	1	0	
10	11	26/05/2018 18:21	26/05/2018	-1.244783	36.866854	1	0	
11	12	27/05/2018 14:03	27/05/2018	-1.233802	36.873170	1	0	
12	13	27/05/2018 14:04	27/05/2018	-1.258863	36.913780	1	1	
13	14	27/05/2018 14:49	27/05/2018	-1.067962	37.051240	1	0	
14	15	27/05/2018 17:00	27/05/2018	-1.260873	36.712982	1	0	
15	16	28/05/2018 19:42	28/05/2018	-1.397699	36.941652	1	0	
16	17	29/05/2018 16:33	29/05/2018	-1.319633	36.837982	1	0	
17	18	29/05/2018 16:44	29/05/2018	-1.305472	36.825128	1	0	
18	19	29/05/2018 17:18	29/05/2018	-1.250170	36.846601	1	0	
19	20	29/05/2018 18:32	29/05/2018	-1.503181	37.087346	1	0	

Next steps: Generate code with data | View recommended plots | New interactive sheet

CLASS CREATION

```
#load the dataset
class DataUnderstanding:
    def __init__(self, data = None):
      self.df = data
    #Load Data
    def load_data(self,path):
      # Try reading with 'latin-1' encoding and specifying the delimiter as '\t'
      self.df = pd.read_csv(path, encoding='latin-1', delimiter='\t')
      return self.df
    def data_understanding(self):
      # First five rows of the dataset
      print('n\n\First five rows of the dataset')
      print('-' * 5)
      print(self.df.head())

      # Dataset Info
      print('n\n\Dataset Info')
      print('-' * 5)
      print(self.df.info())

      # Statistical Sumary of the dataset
      print('n\n\Stastical summary')
      print('-' * 5)
      print(self.df.describe())
# Total Null values per column
      print('\n\nNull values per column')
      print('-' * 5)
      print(self.df.isnull().sum())

      # Unique Values
      print("\n\nUNIQUE VALUES")
      print("-"*12)
      for col in self.df.columns:
          print(f"Column *{col}* has {self.df[col].nunique()} unique values")
          if self.df[col].nunique() < 12:
              print(f"Top unique values in the *{col}* include:")
              for idx in self.df[col].value_counts().index:
                  print(f"- {idx}")
          print("")

      # Total Duplicates in the dataset
      print('n\n\Total duplicated rows in the dataset')
      print('-'* 5)
      print(self.df.duplicated().sum())


data_understanding = DataUnderstanding()
df = data_understanding.load_data('ma3route_crashes_algorithmcode.csv')
data_understanding.data_understanding()
```

```
n
\First five rows of the dataset
-----
  crash_id,crash_datetime,crash_date,latitude,longitude,n_crash_reports,contains_fatality_words,contains_pedestrian_words,conta
0  1,06/06/2018 20:39,06/06/2018,-1.26302986,36.7...
1  2,17/08/2018 06:15,17/08/2018,-0.829710012,37....
2  3,25/05/2018 17:51,25/05/2018,-1.12530079,37.0...
3  4,25/05/2018 18:11,25/05/2018,-1.740957808,37....
4  5,25/05/2018 21:59,25/05/2018,-1.259392311,36....
n
\Dataset Info
-----
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31064 entries, 0 to 31063
Data columns (total 1 columns):
 #   Column
---  ------
 0   crash_id,crash_datetime,crash_date,latitude,longitude,n_crash_reports,contains_fatality_words,contains_pedestrian_words,co
dtypes: object(1)
memory usage: 242.8+ KB
None
n
\Stastical summary
-----
       crash_id,crash_datetime,crash_date,latitude,longitude,n_crash_reports,contains_fatality_words,contains_pedestrian_words,
```

```
count                                                31064
unique                                               31064
top     1,06/06/2018 20:39,06/06/2018,-1.26302986,36.7...
freq                                                     1


Null values per column
-----
crash_id,crash_datetime,crash_date,latitude,longitude,n_crash_reports,contains_fatality_words,contains_pedestrian_words,contain
dtype: int64


UNIQUE VALUES
------------
Column *crash_id,crash_datetime,crash_date,latitude,longitude,n_crash_reports,contains_fatality_words,contains_pedestrian_words

n
\Total duplicated rows in the dataset
-----
0
```

3 DATA CLEANING

In this section, we perform comprehensive data cleaning including handling missing values, outliers, duplicates, and ensuring data types are consistent.

```
#looking at the columns
df.columns
```

```
Index(['crash_id,crash_datetime,crash_date,latitude,longitude,n_crash_reports,contains_fatality_words,contains_pedestrian_words
dtype='object')
```

3.1 Handling Missing Values

```
#checking for missing values
df.isna().sum()
```

crash_id,crash_datetime,crash_date,latitude,longitude,n_crash_reports,contains_fatality_words,contains_pedestrian_words,contains_matatu_wo

dtype: int64

Dataset has no missing values

3.2 Handling Duplicates

```
#checking for duplicates
df.duplicated().sum()
```

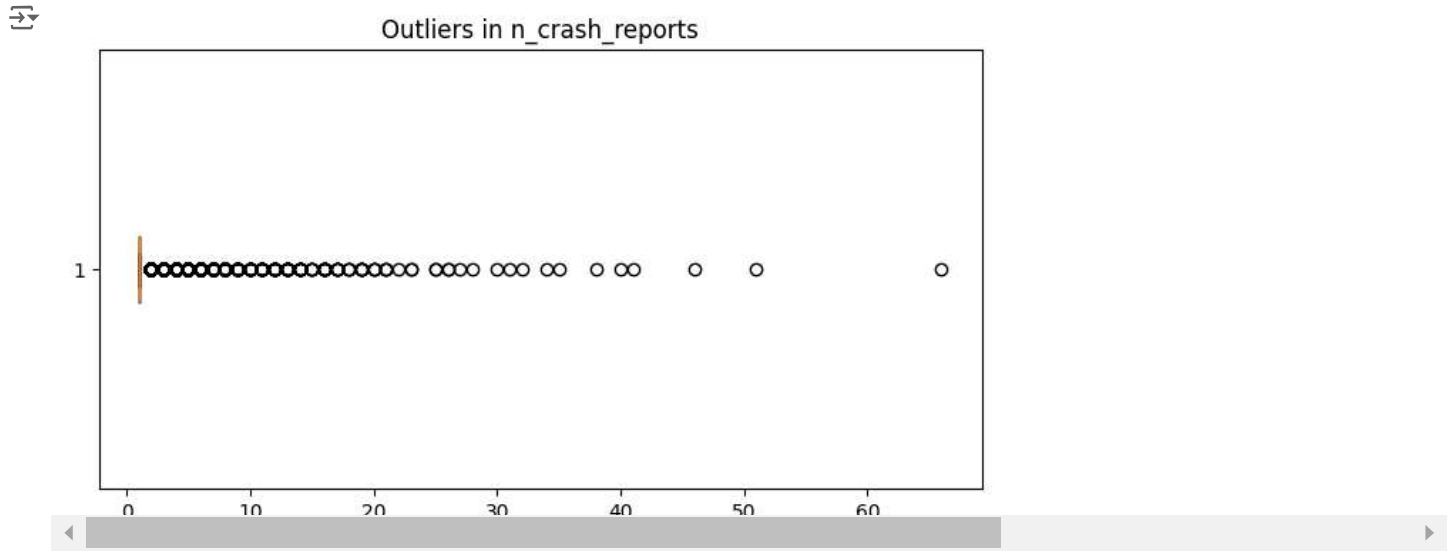
```
0
```

Dataset has no duplicated values

4 Uniformity

4.1 Handling Outliers

```
# Visualizing outliers in 'n_crash_reports'
plt.figure(figsize=(8, 4))
plt.boxplot(data['n_crash_reports'], vert=False)
plt.title('Outliers in n_crash_reports')
plt.show()
```




```
# Removing outliers using the IQR method
Q1 = data['n_crash_reports'].quantile(0.25)
Q3 = data['n_crash_reports'].quantile(0.75)
IQR = Q3 - Q1

filter_outliers = (data['n_crash_reports'] >= (Q1 - 1.5 * IQR)) & \
                  (data['n_crash_reports'] <= (Q3 + 1.5 * IQR))
data = data[filter_outliers].reset_index(drop=True)
```

Dealt with the outliers using the IQR method

4.2 Correcting Data Types

```
# Converting date columns to datetime format
data['crash_date'] = pd.to_datetime(data['crash_date'], errors='coerce')
data['crash_datetime'] = pd.to_datetime(data['crash_datetime'], errors='coerce')


# Converting binary categorical columns to integer type
binary_columns = ['contains_fatality_words', 'contains_pedestrian_words',
                  'contains_matatu_words', 'contains_motorcycle_words']
for col in binary_columns:
    data[col] = data[col].astype(int)
```

4.3 Standardize Categorical Values

```
# Ensuring consistent labeling in categorical columns
data['contains_pedestrian_words'].replace({'yes': 1, 'no': 0}, inplace=True)
```

4.4 Handling Text and Keyword Based Columns

```
# Convert columns indicating presence of keywords (e.g., 'yes', 'no') to binary (1, 0)
text_indicators = ['contains_pedestrian_words', 'contains_matatu_words',
                   'contains_motorcycle_words']
for col in text_indicators:
    data[col] = data[col].apply(lambda x: 1 if x == 'yes' else 0)
```

4.5 Geospatial Cleaning

```
# Ensure latitude and longitude are within valid range for Kenya
valid_latitude = (data['latitude'] >= -4.7) & (data['latitude'] <= 5.0)
valid_longitude = (data['longitude'] >= 33.5) & (data['longitude'] <= 42.0)
data = data[valid_latitude & valid_longitude].reset_index(drop=True)
```

```
# Display cells with whitespace in specific columns
for col in df.select_dtypes(include=['object']).columns:
    print(f"Whitespace in column '{col}':")
    print(df[df[col].str.contains(r'^\s|\s$', na=False)][col])
```

```
Whitespace in column 'crash_id,crash_datetime,crash_date,latitude,longitude,n_crash_reports,contains_fatality_words,contains_pe
Series([], Name: crash_id,crash_datetime,crash_date,latitude,longitude,n_crash_reports,contains_fatality_words,contains_pedestr
```

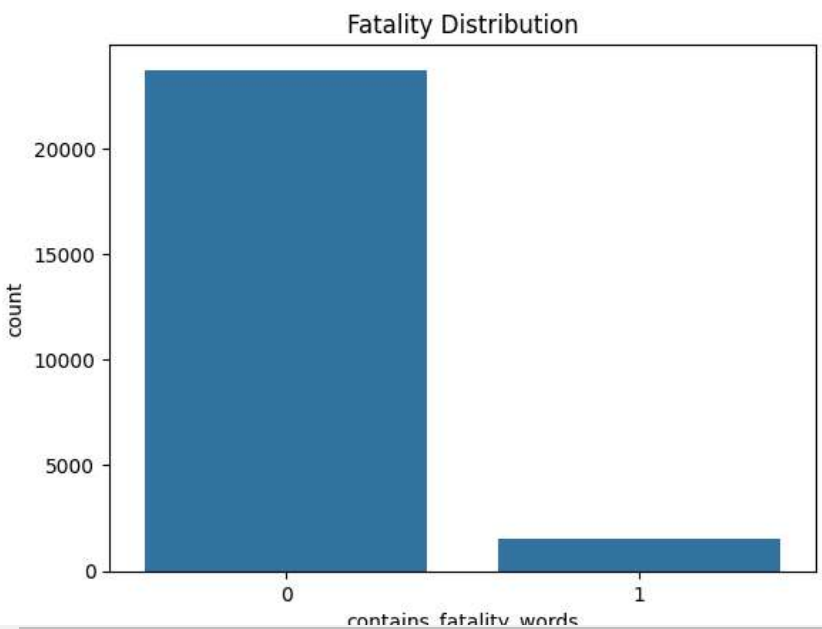
```
# Strip whitespace from column names
df.columns = df.columns.str.strip()
```

5 Exploratory data anaylsis

5.1 Target variable anaylsis

Checking for the distribution of the target variable to check for class imbalance

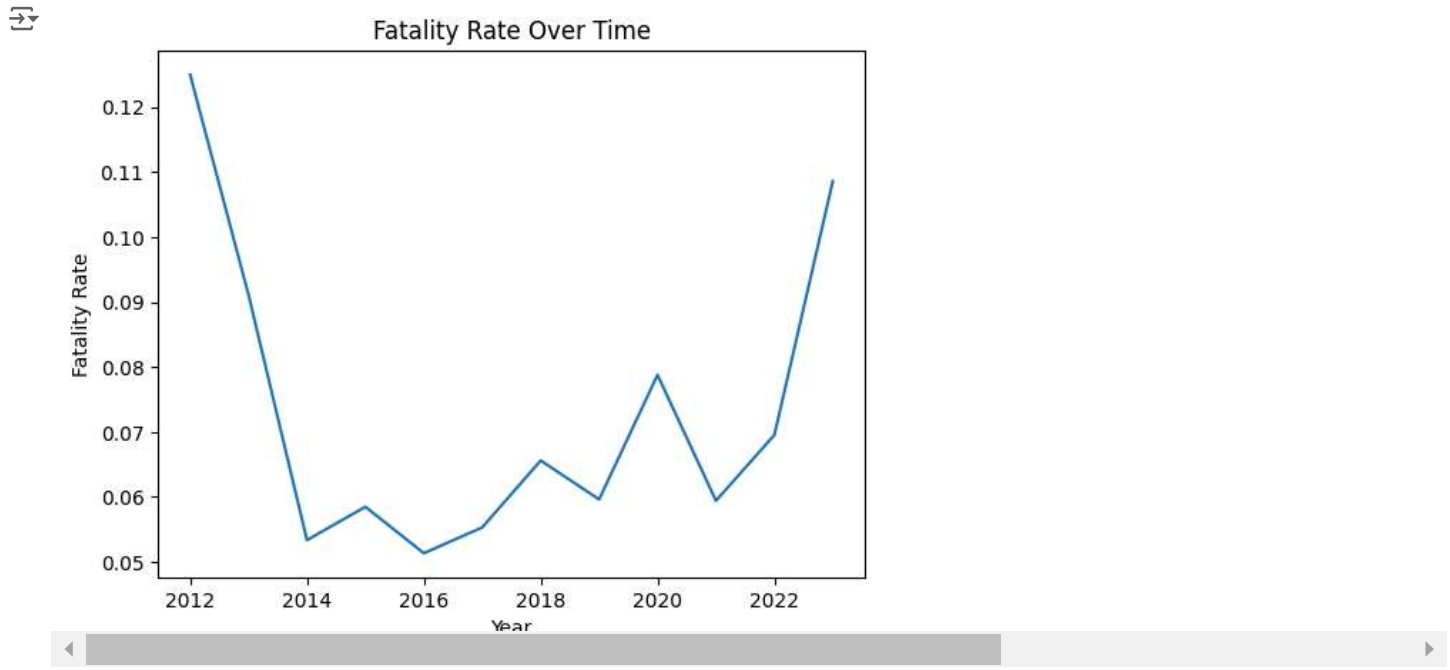
```
import matplotlib.pyplot as plt
import seaborn as sns

# Distribution of the target variable
sns.countplot(x='contains_fatality_words', data=data)
plt.title('Fatality Distribution')
plt.show()
```



There is class imbalance which will be dealt with using SMOTE

```
# Fatalities over time (e.g., per year)
data['crash_date'] = pd.to_datetime(data['crash_date'], errors='coerce')
```

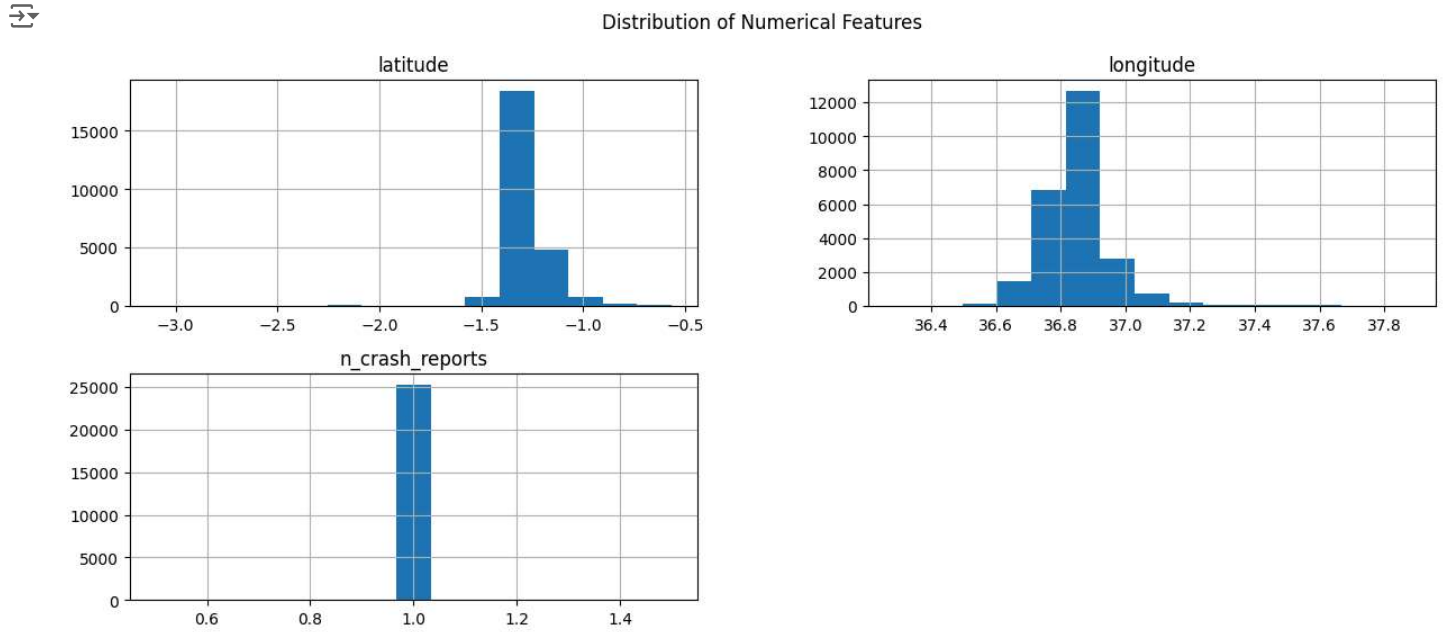
```
data['year'] = data['crash_date'].dt.year
fatality_trend = data.groupby('year')['contains_fatality_words'].mean()

fatality_trend.plot(kind='line', title='Fatality Rate Over Time')
plt.ylabel('Fatality Rate')
plt.xlabel('Year')
plt.show()
```



Over the past decade, the fatality rate has decreased

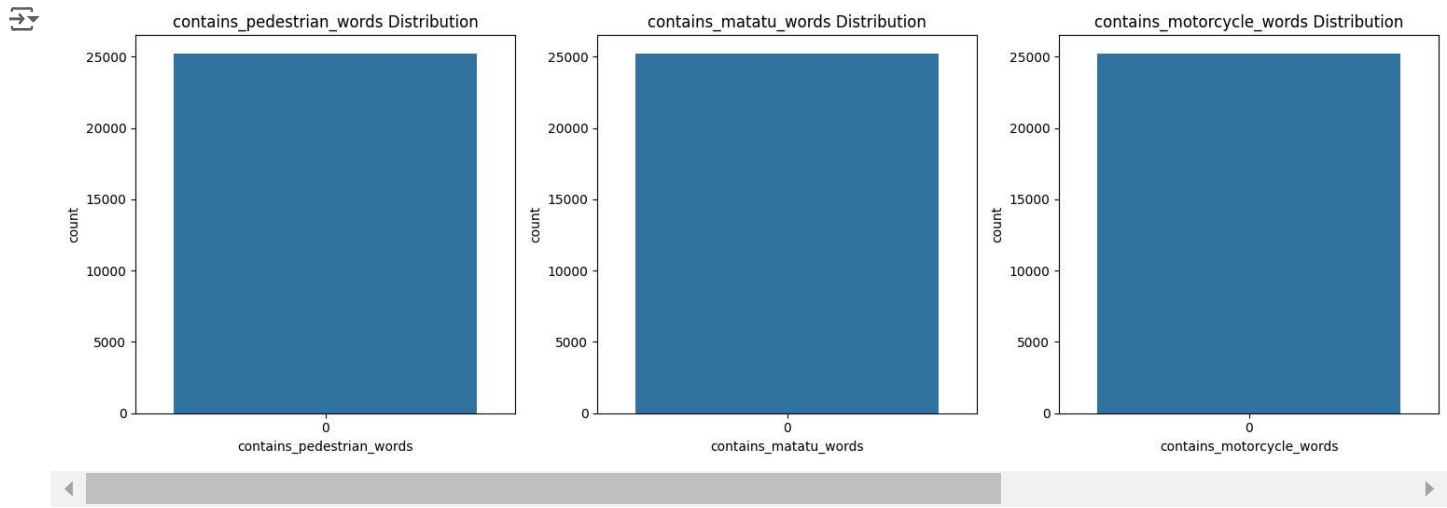
5.2 Univariate Analysis

```
# Histogram for numerical features
numerical_features = ['latitude', 'longitude', 'n_crash_reports']
data[numerical_features].hist(bins=15, figsize=(15, 6))
plt.suptitle('Distribution of Numerical Features')
plt.show()
```



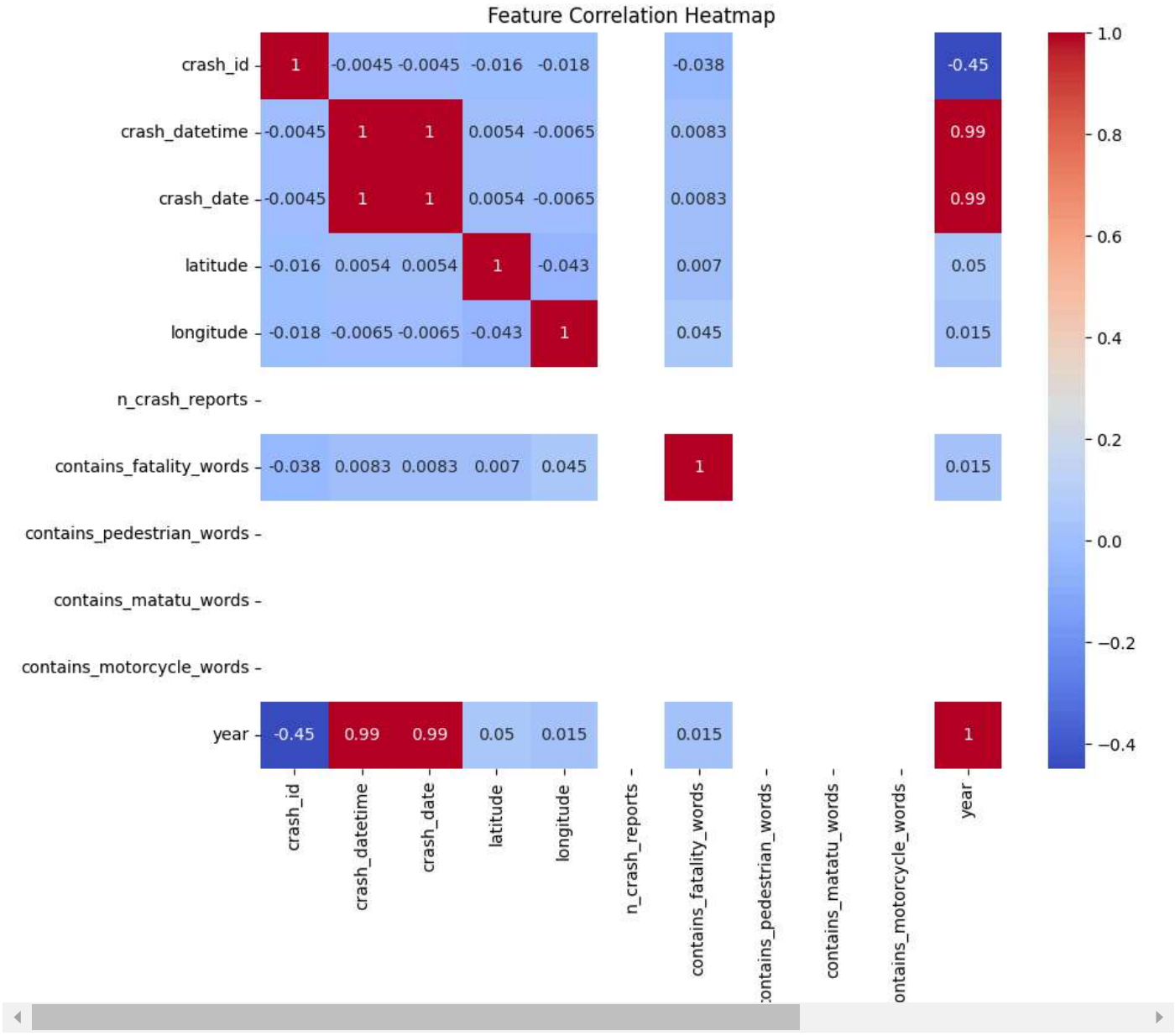
The distributions indicate that the dataset is geographically focused on a particular region with limited variability in crash report frequency making it suitable for studying crash patterns in that specific area.

```
# Bar plots for categorical features
categorical_features = ['contains_pedestrian_words', 'contains_matatu_words',
                        'contains_motorcycle_words']
fig, axes = plt.subplots(1, len(categorical_features), figsize=(15, 5))
for i, col in enumerate(categorical_features):
    sns.countplot(x=col, data=data, ax=axes[i])
    axes[i].set_title(f'{col} Distribution')
plt.tight_layout()
plt.show()
```



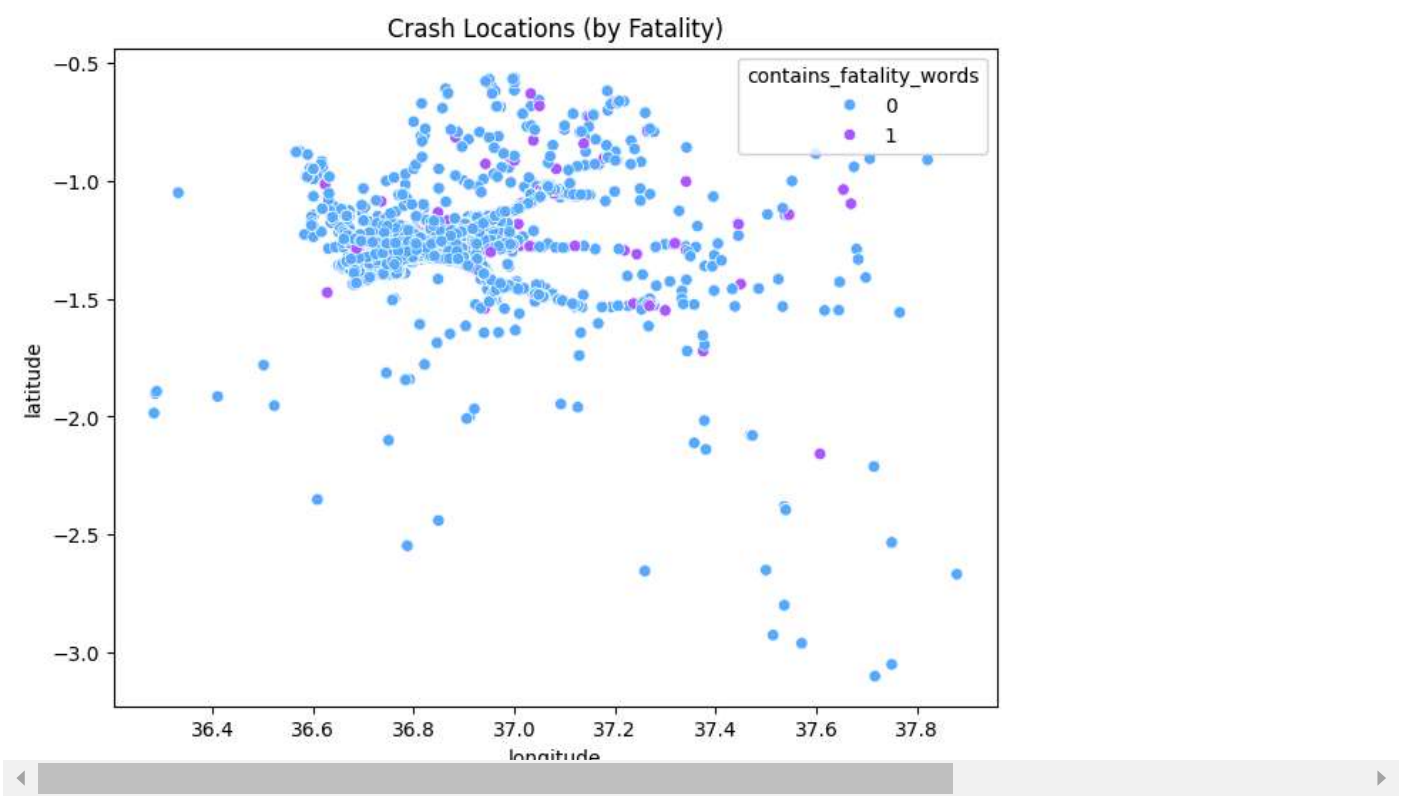
5.3 Bivariate Analysis

```
# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Feature Correlation Heatmap')
plt.show()
```



The blue color represents negative correlations, red positive while white indicates a near-zero correlation

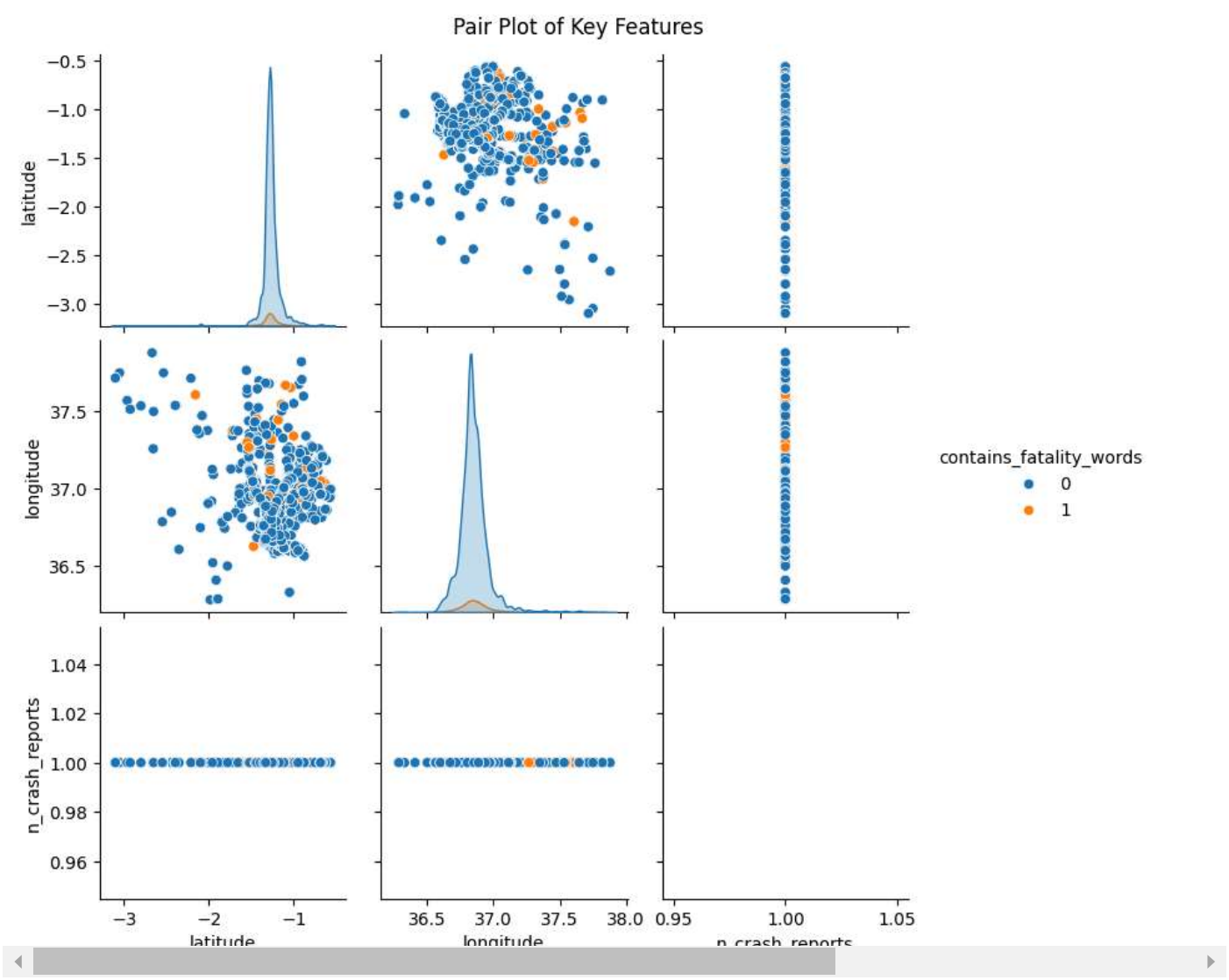
```
# Scatter plot: latitude vs. longitude
plt.figure(figsize=(8, 6))
sns.scatterplot(x='longitude', y='latitude', hue='contains_fatality_words',
                data=data, palette='cool')
plt.title('Crash Locations (by Fatality)')
plt.show()
```



The scatter plot above highlights where fatalities are most and least concentrated thereby showing the spatial distribution of fatal accidents at a glance

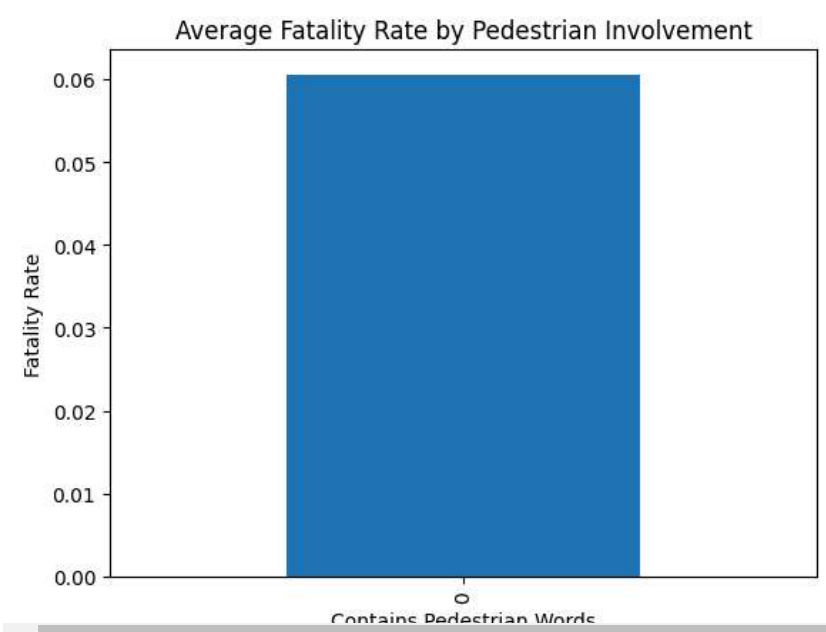
5.4 Multivariate Analysis

```
# Pair plot for key features with target variable
sns.pairplot(data, hue='contains_fatality_words',
             vars=['latitude', 'longitude', 'n_crash_reports'])
plt.suptitle('Pair Plot of Key Features', y=1.02)
plt.show()
```



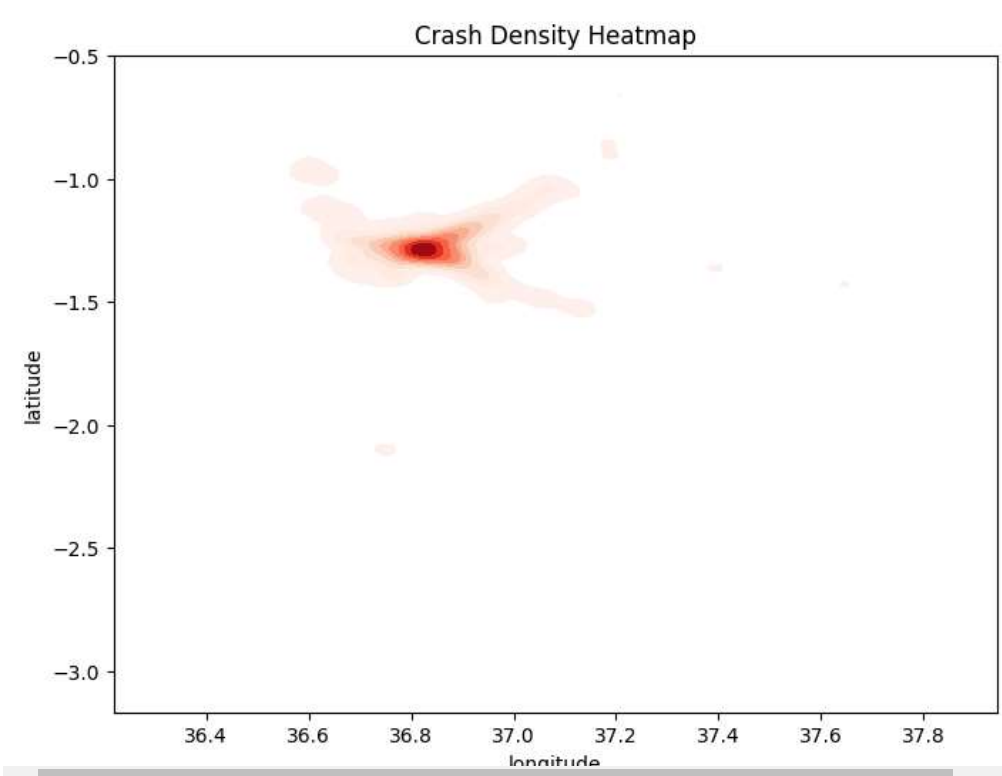
The pair plot implies that the dataset is geographiically clustered as most records involve a single crash report and fatal incidences are distributed across the area without a geographical pattern. This shows that fatal incidences are not associated with any particular location within the scope of the dataset

```
# Grouped analysis: mean fatality rate by contains_pedestrian_words
grouped = data.groupby('contains_pedestrian_words')['contains_fatality_words'].mean()
grouped.plot(kind='bar', title='Average Fatality Rate by Pedestrian Involvement')
plt.ylabel('Fatality Rate')
plt.xlabel('Contains Pedestrian Words')
plt.show()
```



5.5 Geospatial Analysis

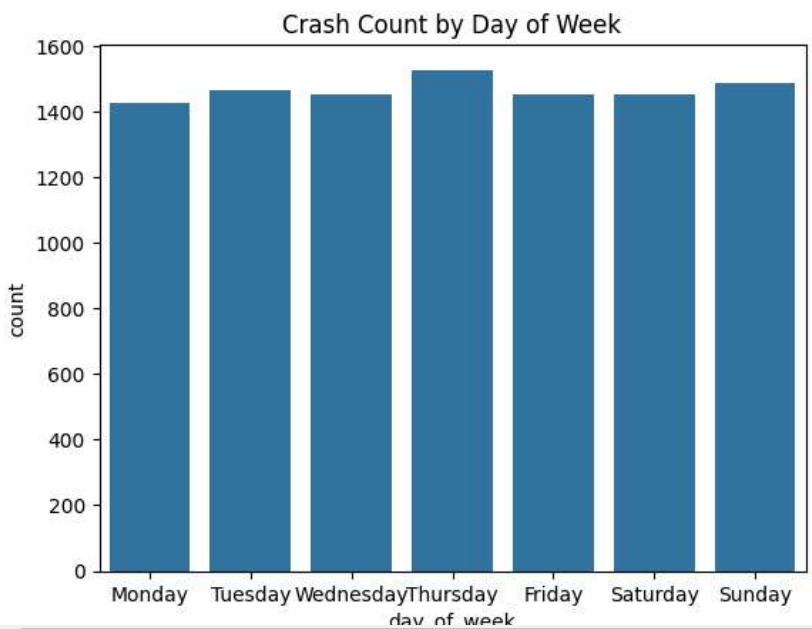
```
# use matplotlib scatter for location density
plt.figure(figsize=(8, 6))
sns.kdeplot(x='longitude', y='latitude', data=data, cmap='Reds', fill=True)
plt.title('Crash Density Heatmap')
plt.show()
```



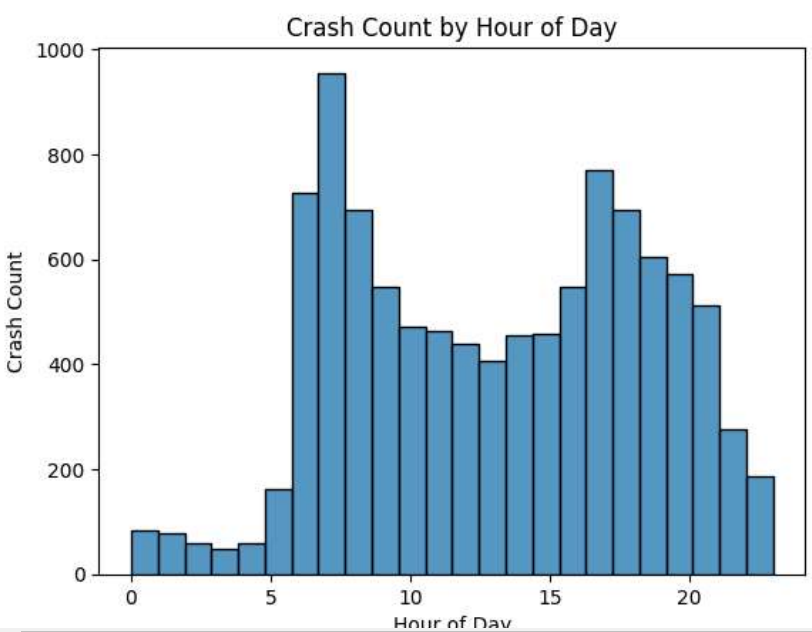
The heatmap provides valuable insight into the geographical distribution of crash incidences with a clear hotspot around a specific latitude and longitude. This is useful for targeted interventions in traffic safety and resource allocation

5.6 Temporal Analysis

```
# Analyzing crash frequency over the days of the week
data['day_of_week'] = data['crash_date'].dt.day_name()

sns.countplot(x='day_of_week', data=data, order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.title('Crash Count by Day of Week')
plt.show()
```




```
# Analyzing crash frequency by time of day
data['hour'] = data['crash_datetime'].dt.hour

sns.histplot(data['hour'], bins=24)
plt.title('Crash Count by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Crash Count')
plt.show()
```



6 FEATURE ENGINEERING

In this section, we create new features that may help improve model performance by capturing patterns in the data.

6.1 Date-Time Feature Engineering

By doing this the model gains insight into time based patterns that may influence crash likelihood i.e hourly and daily patterns and seasonal/yearly trends

```
# Convert crash_datetime to datetime and extract time components
data['crash_datetime'] = pd.to_datetime(data['crash_datetime'], errors='coerce')

# Extracting date-time features
data['hour'] = data['crash_datetime'].dt.hour
data['day'] = data['crash_datetime'].dt.day
data['weekday'] = data['crash_datetime'].dt.dayofweek
data['month'] = data['crash_datetime'].dt.month
data['year'] = data['crash_datetime'].dt.year
data['is_weekend'] = data['weekday'].apply(lambda x: 1 if x >= 5 else 0)

# Day-part feature
def categorize_time(hour):
    if 5 <= hour < 12:
        return 'morning'
    elif 12 <= hour < 17:
        return 'afternoon'
    elif 17 <= hour < 21:
        return 'evening'
    else:
        return 'night'

data['day_part'] = data['hour'].apply(categorize_time)
```

6.2 Spatial Features

This enriches the dataset by adding a geographical context and help to capture patterns related to the location of each crash

```
#  calculating distance from a city center (e.g., Nairobi)
def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Earth radius in kilometers
    phi1, phi2 = np.radians(lat1), np.radians(lat2)
    dphi = np.radians(lat2 - lat1)
    dlambda = np.radians(lon2 - lon1)

    a = np.sin(dphi/2)**2 + np.cos(phi1)*np.cos(phi2)*np.sin(dlambda/2)**2
    return 2 * R * np.arctan2(np.sqrt(a), np.sqrt(1 - a))

# Calculate distance from Nairobi (approx latitude: -1.2921, longitude: 36.8219)
nairobi_lat, nairobi_lon = -1.2921, 36.8219
data['distance_to_nairobi'] = data.apply(lambda row: haversine(row['latitude'], row['longitude'], nairobi_lat, nairobi_lon), axis=1
```

6.3 Traffic and Vehicle-Related Features

The two features add crucial information about involvement of high-risk vehicles and crash severity which enriches thedataset by prividing context on traffic patterns and potential which are essential for accurate predictions

```
# Combining vehicle-related keywords into a single feature
data['contains_vehicle_words'] = data['contains_matatu_words'] | data['contains_motorcycle_words']
```

```
# Creating a severity indicator by combining fatality-related keywords
data['severity_index'] = data[['contains_fatality_words', 'contains_pedestrian_words']].sum(axis=1)
```

6.4 Crash Density Features

The featur provides insights into the distribution of crashes, identifying high density areas, high risk areas, spatial pattern recognition and improving prediction accuracy thus helps the model to understand and predict patterns related to high risk areas

```
# Create a local crash density feature by counting crashes within a grid
from sklearn.neighbors import BallTree

# Using latitude and longitude
coords = np.radians(data[['latitude', 'longitude']])
tree = BallTree(coords, metric='haversine')

# Radius (5 km) in radians
radius = 5 / 6371  # Earth radius in km
data['local_crash_density'] = tree.query_radius(coords, r=radius, count_only=True) - 1
```

6.5 Seasonal and Environmental Features

This feature captures the potential impact of seasonal weather patterns on crash occurences i.e influence of weather conditions, weather improvement and targeted interventions

```
# Season feature from month
def map_season(month):
    if month in [12, 1, 2]:
        return 'rainy'
    elif month in [6, 7, 8]:
        return 'dry'
    else:
        return 'transitional'

data['season'] = data['month'].apply(map_season)
```

6.6 Crash Report Aggregation Features

This feature helps to capture broader patterns i.e temporal fatality patterns, simplified vehicle and severity indicators and data streamlining

```
# Fatality mention ratio
region_counts = data.groupby('day_part')['contains_fatality_words'].mean()
data['fatality_mention_ratio'] = data['day_part'].map(region_counts)
```

```
# Combining keywords into single features
data['contains_vehicle_words'] = data['contains_matatu_words'] | data['contains_motorcycle_words']
data['severity_index'] = data[['contains_fatality_words', 'contains_pedestrian_words']].sum(axis=1)
```

```
# Fatality mention ratio
region_counts = data.groupby('day_part')['contains_fatality_words'].mean()
data['fatality_mention_ratio'] = data['day_part'].map(region_counts)
```

```
# Step 3: Drop Old Columns
data = data.drop(columns=['crash_id', 'crash_datetime', 'crash_date', 'latitude', 'longitude',
                          'contains_matatu_words', 'contains_motorcycle_words'])
```

6.7 Generating new dataset with engeered features

```
# Generating new dataset
data.to_csv('processed_crash_data.csv', index=False)
print("Processed data saved to 'processed_crash_data.csv'")
data.head()
```

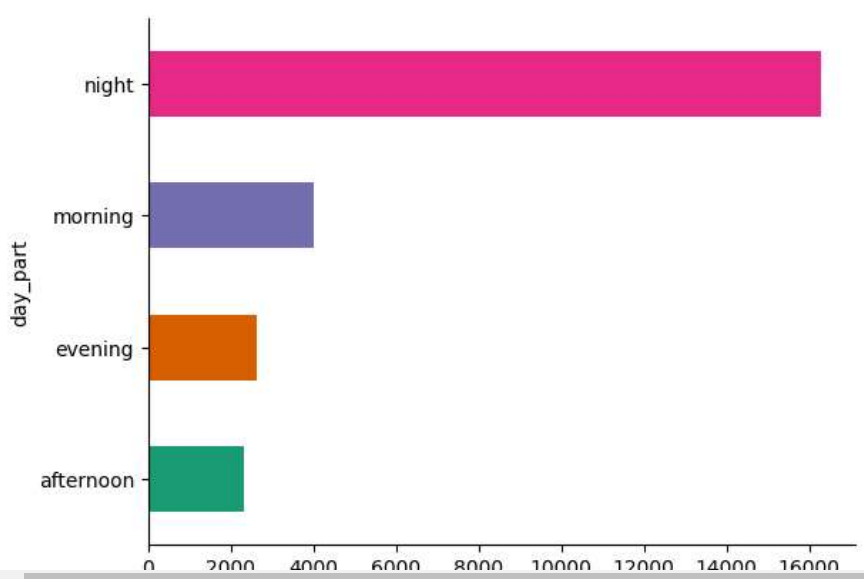
```
Processed data saved to 'processed_crash_data.csv'
```

	n_crash_reports	contains_fatality_words	contains_pedestrian_words	year	day_of_week	hour	day	weekday	month	is_weeke
0	1	0	0	2018.0	Wednesday	20.0	6.0	2.0	6.0	
1	1	1	0	NaN	NaN	NaN	NaN	NaN	NaN	
2	1	0	0	NaN	NaN	NaN	NaN	NaN	NaN	
3	1	0	0	NaN	NaN	NaN	NaN	NaN	NaN	
4	1	1	0	NaN	NaN	NaN	NaN	NaN	NaN	

Next steps: Generate code with data | View recommended plots | New interactive sheet

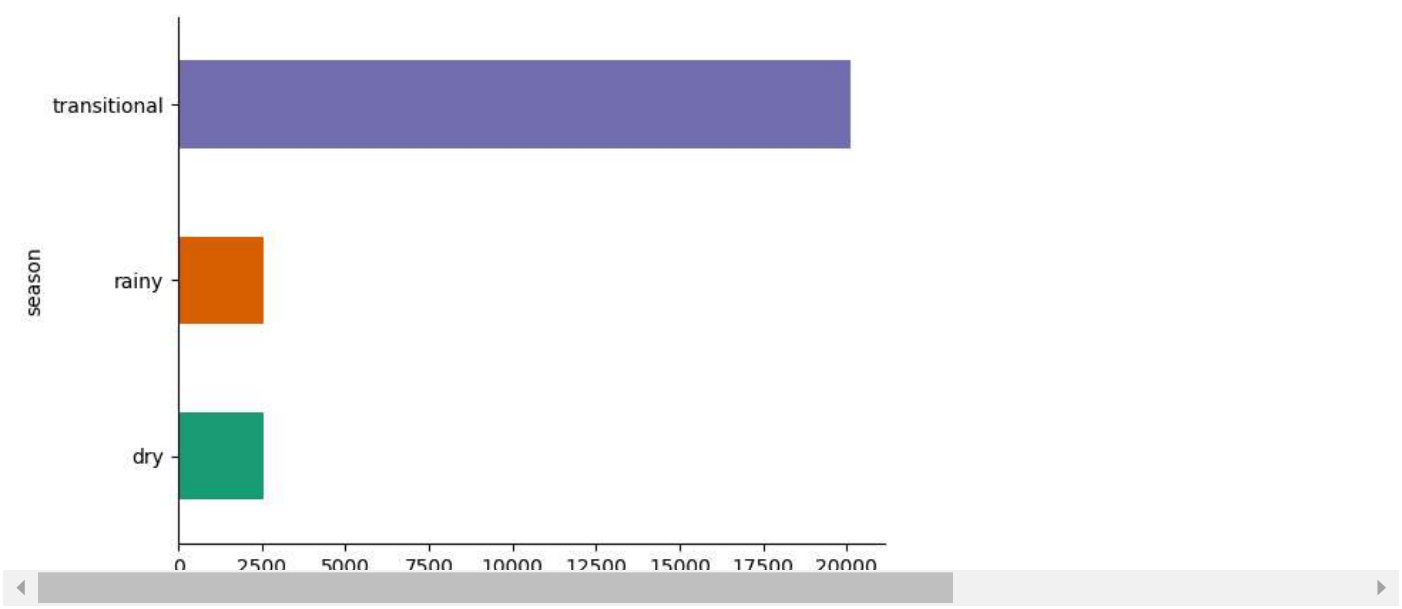
Understaning the new generated dataset with the new features

```
#day_part

from matplotlib import pyplot as plt
import seaborn as sns
data.groupby('day_part').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```



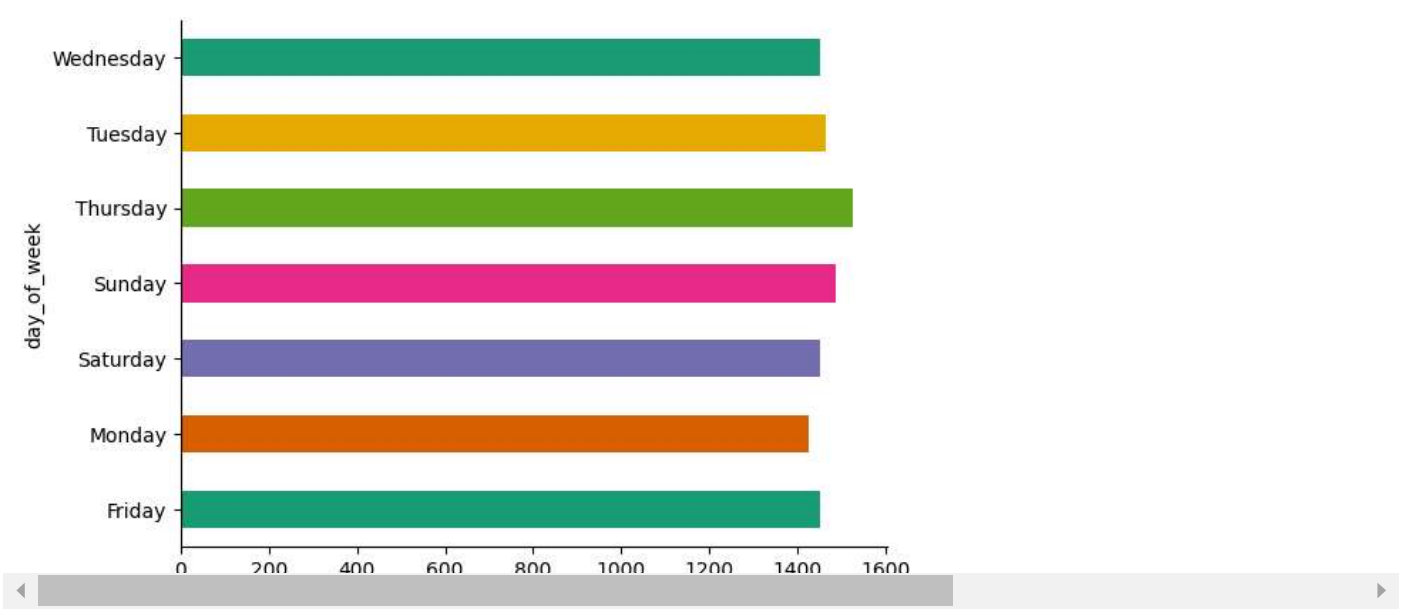
the data shows more crash happen during at night followed by morning

```
#season
from matplotlib import pyplot as plt
import seaborn as sns
data.groupby('season').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```



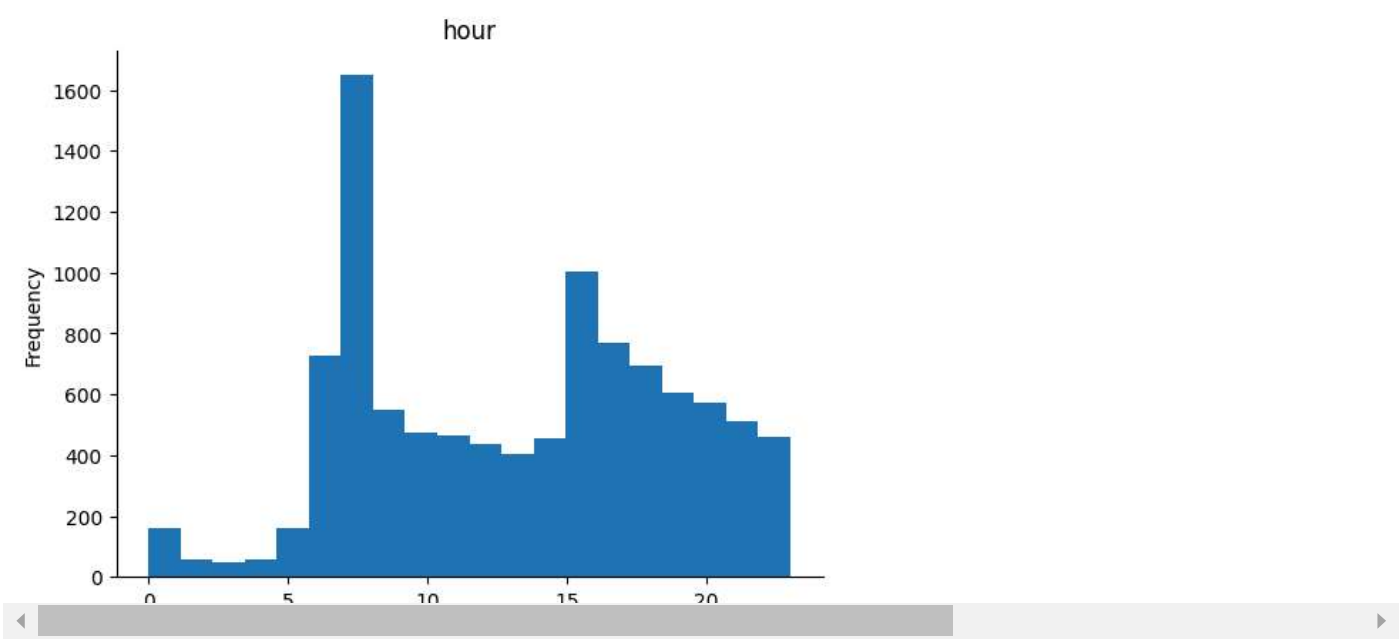
Transitional season has the highest reported crashes

```
#day_of_week
from matplotlib import pyplot as plt
import seaborn as sns
data.groupby('day_of_week').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```

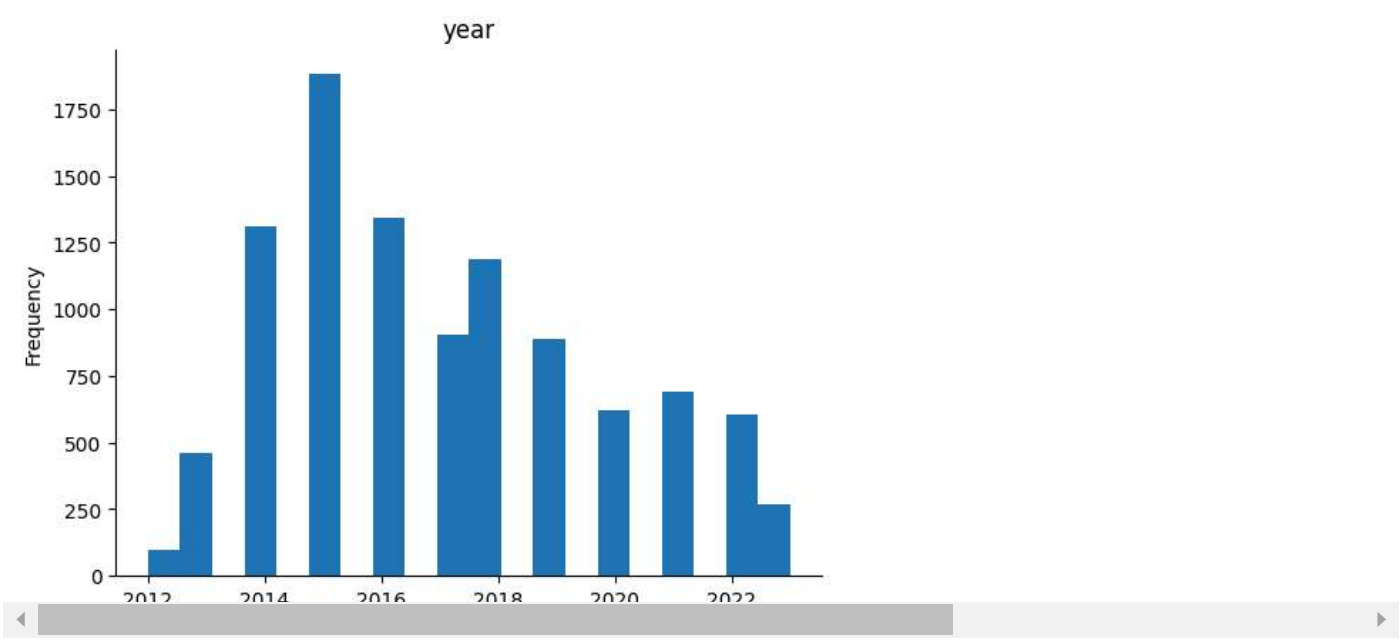


Thursady has the highest number of reported crashes

```
#hour
from matplotlib import pyplot as plt
data['hour'].plot(kind='hist', bins=20, title='hour')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

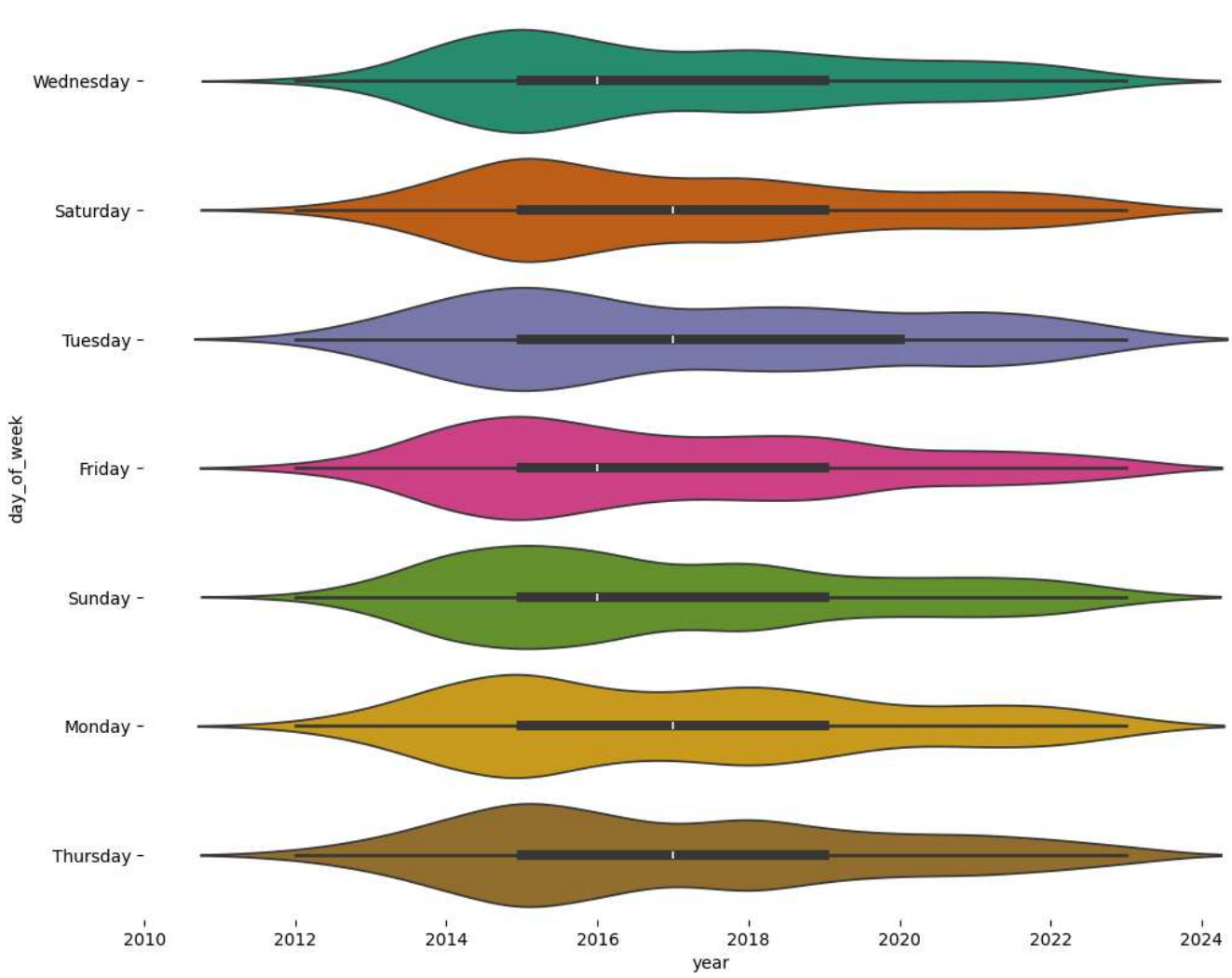



```
#frequency of accidents per year

from matplotlib import pyplot as plt
data['year'].plot(kind='hist', bins=20, title='year')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



the year 2015 had the highest number of crash reports

```
# day_of_week vs year

from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(data['day_of_week'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(data, x='year', y='day_of_week', inner='box', palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```



7 MODELING

7.1 Data Preparation for Modeling

Here, we split the data into training and testing sets and standardize the features for models sensitive to feature scales.

```
# Features and target variable selection
X = data.drop(columns=['contains_fatality_words'])  # Dropping target column
y = data['contains_fatality_words']

# Get the column names of the DataFrame X
columns_to_check = X.columns

#drop rows with NAt values in all columns
X = X.dropna(subset=columns_to_check)  # Now using the list of columns
y = y[X.index]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert datetime columns to Unix timestamp
for col in X_train.select_dtypes(include=['datetime64']).columns:
    X_train[col] = X_train[col].astype('int64') // 10**9  # Convert to Unix timestamp
    X_test[col] = X_test[col].astype('int64') // 10**9    # Apply the same to X_test
```

```
# ----> ONE-HOT ENCODING FOR CATEGORICAL FEATURES <----
from sklearn.preprocessing import OneHotEncoder
categorical_features = X_train.select_dtypes(include=['object']).columns # Select object type columns

# Create and fit OneHotEncoder
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore') # sparse=False for numpy array
encoded_data_train = encoder.fit_transform(X_train[categorical_features])
encoded_data_test = encoder.transform(X_test[categorical_features])

# Create DataFrames from encoded data
encoded_df_train = pd.DataFrame(encoded_data_train, columns=encoder.get_feature_names_out(categorical_features), index=X_train.inde
encoded_df_test = pd.DataFrame(encoded_data_test, columns=encoder.get_feature_names_out(categorical_features), index=X_test.index)

# Concatenate encoded features with numerical features
X_train = pd.concat([X_train.drop(columns=categorical_features), encoded_df_train], axis=1)
X_test = pd.concat([X_test.drop(columns=categorical_features), encoded_df_test], axis=1)


# Select only numerical features for scaling
numerical_features = X_train.select_dtypes(include=['number']).columns
X_train_num = X_train[numerical_features]
X_test_num = X_test[numerical_features]

# Standardizing features
scaler = StandardScaler()
X_train_num = scaler.fit_transform(X_train_num) # Scale only numerical features
X_test_num = scaler.transform(X_test_num) # Scale only numerical features

# Update original DataFrames with scaled numerical features
X_train[numerical_features] = X_train_num
X_test[numerical_features] = X_test_num
```

7.2 Dealing with class imbalance using smote

```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder


# Create a LabelEncoder object
encoder = LabelEncoder()

# Iterate through columns and encode object type columns
for column in X.select_dtypes(include=['object']).columns:
    X[column] = encoder.fit_transform(X[column])

# Now apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)
```

7.3 Checking and dealing with Multicollinearity

```
def calculate_vif(data):
        vif = pd.DataFrame()
        vif["Feature"] = data.columns
        vif["VIF"] = [variance_inflation_factor(data.values, i) for i in range(data.shape[1])]
        return vif

vif_df = calculate_vif(pd.DataFrame(X_train))
# identying and dropping highly correlated features
high_vif_features = vif_df[vif_df["VIF"] > 10]["Feature"].tolist()
X_train = X_train.drop(columns=high_vif_features)
X_test = X_test.drop(columns=high_vif_features)
```

7.4 Baseline Model

Creating a baseline model to establish a performance benchmark.

```
from sklearn.dummy import DummyClassifier

# Dummy Classifier as a baseline
dummy_clf = DummyClassifier(strategy='most_frequent')
dummy_clf.fit(X_train, y_train)
baseline_accuracy = dummy_clf.score(X_test, y_test)

print(f'Baseline Accuracy (Most Frequent Class): {baseline_accuracy:.2f}')
```

```
Baseline Accuracy (Most Frequent Class): 0.50
```

7.5 Model Selection

Here, we train several models and use cross-validation to determine which one performs best for our dataset.

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import OneHotEncoder,StandardScaler
from sklearn.model_selection import train_test_split



# 'X_train' is your training DataFrame
X_train = pd.get_dummies(X_train)

# Initialize the model
model = RandomForestClassifier(random_state=42)


cv_score = cross_val_score(model, X_train, y_train, cv=5, error_score='raise')


# Model candidates
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Support Vector Machine': SVC(probability=True)
}

# Cross-validation
for name, model in models.items():
    cv_score = cross_val_score(model, X_train, y_train, cv=5).mean()
    print(f'{name} CV Score: {cv_score:.2f}')
```

```
Logistic Regression CV Score: 1.00
Random Forest CV Score: 1.00
Gradient Boosting CV Score: 1.00
Support Vector Machine CV Score: 0.53
```

7.6 Regulirazation of the logistic regression model

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

log_reg = LogisticRegression(penalty='l2', solver='liblinear')
log_reg.fit(X_train_scaled, y_train)

y_pred = log_reg.predict(X_test_scaled)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2862
           1       1.00      1.00      1.00      2907

    accuracy                           1.00      5769
   macro avg       1.00      1.00      1.00      5769
weighted avg       1.00      1.00      1.00      5769
```

7.7 Hyperparameter Tuning

Using GridSearchCV to find the best hyperparameters for selected models, such as Random Forest.

```
# Hyperparameter tuning for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Best hyperparameters and score
print(f'Best parameters: {grid_search.best_params_}')
print(f'Best cross-validation score: {grid_search.best_score_:.2f}')
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 50}
Best cross-validation score: 1.00
```

7.8 Model Evaluation Metrics

Evaluating the model with various metrics to understand its performance on the test set.

```
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)

# Align columns in both datasets (fill missing columns with 0)
X_train, X_test = X_train.align(X_test, join='left', axis=1, fill_value=0)

best_model = RandomForestClassifier(random_state=42)
best_model.fit(X_train, y_train)


y_pred = best_model.predict(X_test)
y_pred_prob = best_model.predict_proba(X_test)[:, 1]


# Using the best model from GridSearch
best_model = grid_search.best_estimator_

# Predictions on the test set
y_pred = best_model.predict(X_test)
y_pred_prob = best_model.predict_proba(X_test)[:, 1]

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)

print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

```
print(f'F1 Score: {f1:.2f}')
print(f'ROC-AUC Score: {roc_auc:.2f}')
```

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
ROC-AUC Score: 1.00
```

7.9 Ensemble Methods

Combining multiple models for potentially better predictive power.

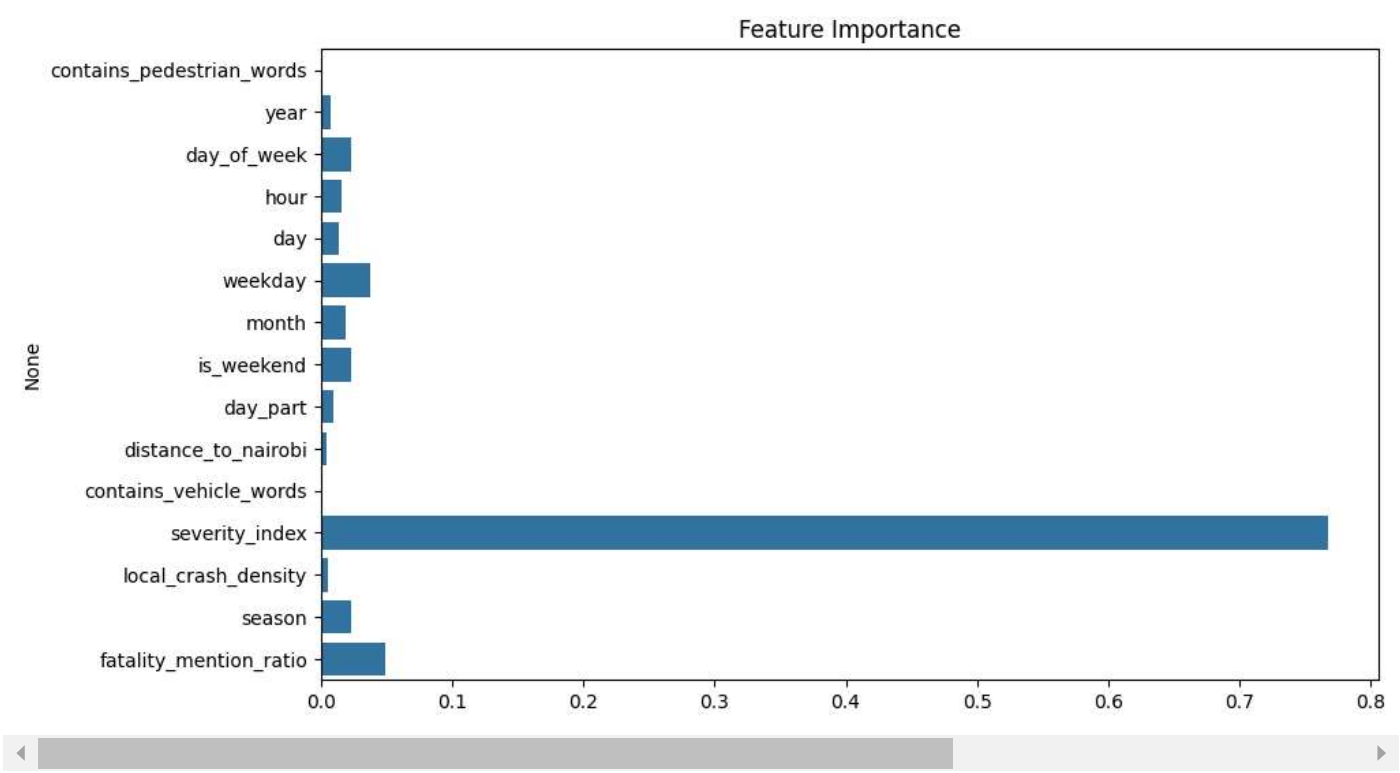
```
from sklearn.ensemble import VotingClassifier

# Voting Classifier
voting_clf = VotingClassifier(
    estimators=[('lr', LogisticRegression(max_iter=1000)),
                ('rf', RandomForestClassifier()),
                ('gb', GradientBoostingClassifier())],
    voting='soft'
)

# Fit and evaluate ensemble model
voting_clf.fit(X_train, y_train)
ensemble_score = voting_clf.score(X_test, y_test)

print(f'Ensemble Model Test Score: {ensemble_score:.2f}')
```

```
Ensemble Model Test Score: 1.00
```

8 Model Interpretation

Analyzing feature importance to interpret the model and understand the key features contributing to predictions.

```
importances = best_model.feature_importances_  # Or best_model.coef_ for linear models
features = X_train.columns  # Ensure features match the training data used to fit best_model

# Check if importances and features have the same length. If not, trim importances
if len(importances) != len(features):
    importances = importances[:len(features)] # Trim importances to match the length of features

# Plotting feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x=importances, y=features)
plt.title('Feature Importance')
plt.show()
```



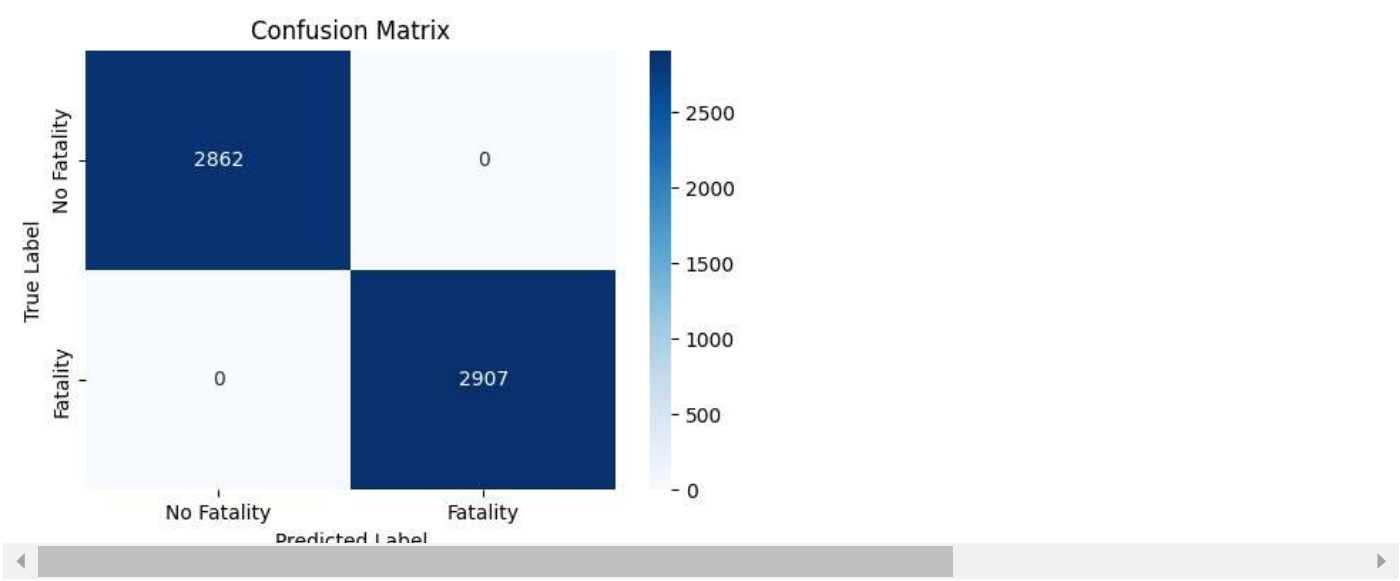
9 Modeling evaluation

Here, we evaluate the model's performance using multiple metrics and visualizations

9.1 Confusion Matrix

Visualizing the confusion matrix to understand the types of errors made by the model.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns


# Confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Fatality', 'Fatality'], yticklabels=['No Fatality', 'Fatality']
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



9.2 Classification Report

Displaying precision, recall, and F1-score for each class, providing insights into model performance on each class.

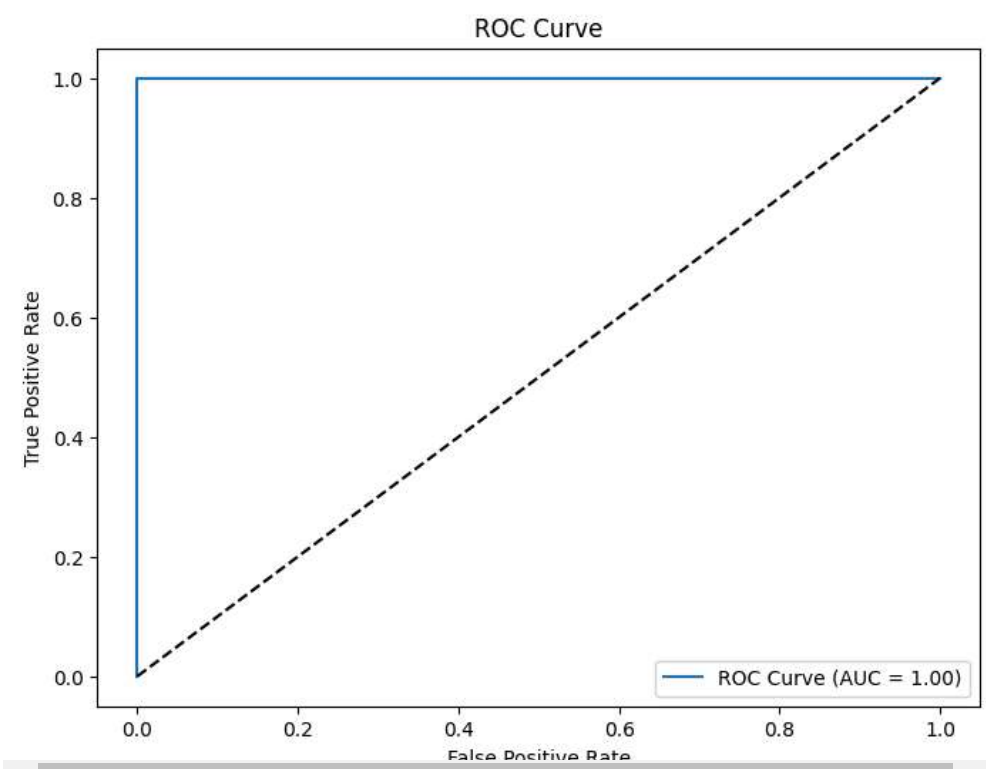
```
from sklearn.metrics import classification_report

# Classification report
print('Classification Report:')
print(classification_report(y_test, y_pred, target_names=['No Fatality', 'Fatality']))
```

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

   No Fatality       1.00      1.00      1.00      2862
      Fatality       1.00      1.00      1.00      2907

      accuracy                           1.00      5769
     macro avg       1.00      1.00      1.00      5769
  weighted avg       1.00      1.00      1.00      5769
```

9.3 ROC-AUC and ROC Curve

Evaluating the model's ability to distinguish between classes across different thresholds using ROC-AUC and plotting the ROC curve.

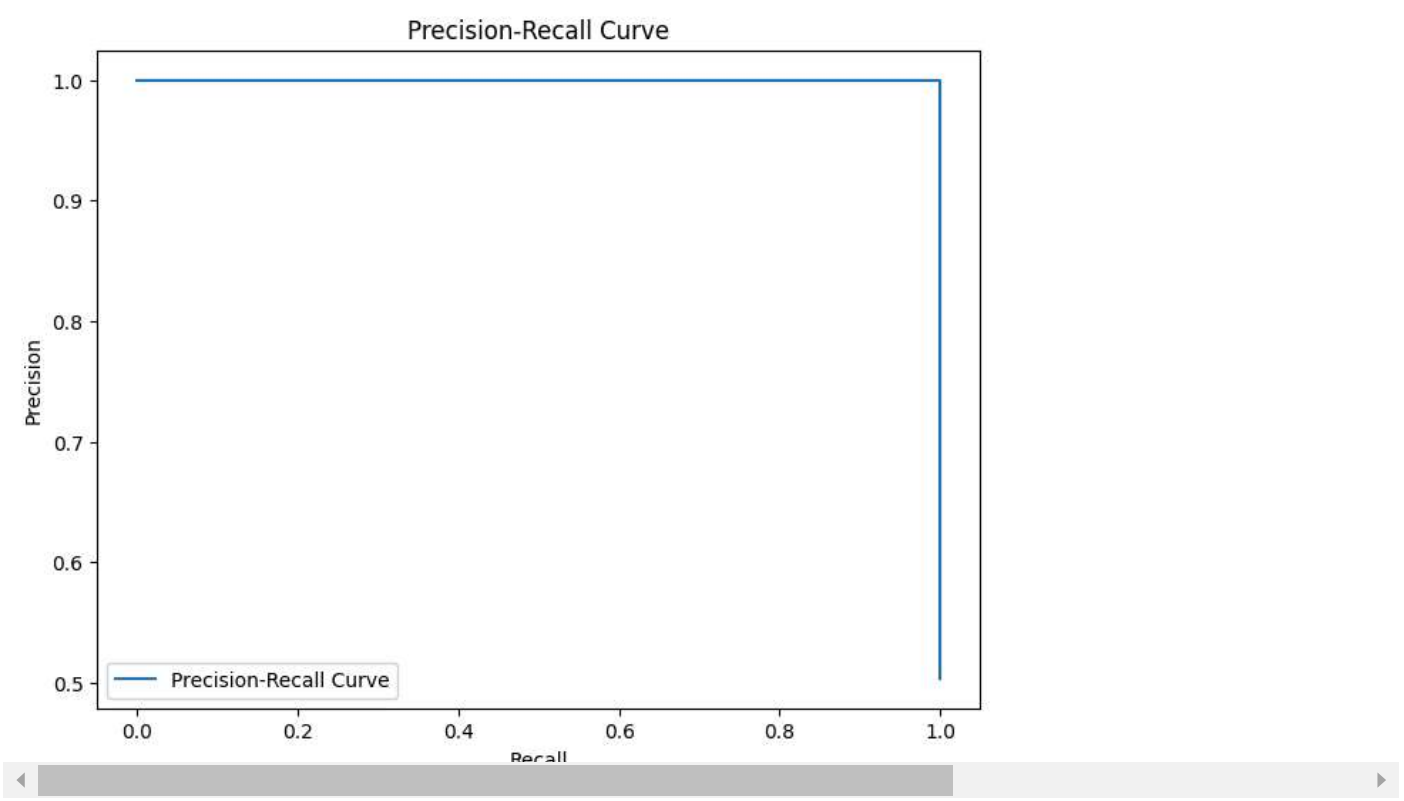
```
from sklearn.metrics import roc_curve, auc

# ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Plotting the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



9.4 Precision-Recall Curve

For imbalanced classes, the Precision-Recall curve can provide better insights than the ROC curve by focusing on the trade-off between precision and recall.

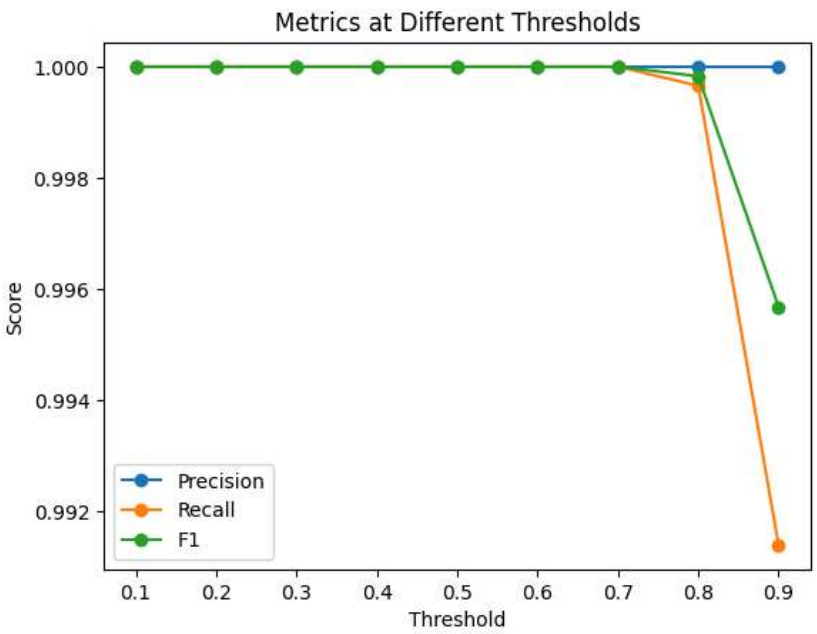
```
from sklearn.metrics import precision_recall_curve

# Precision-recall curve
precision, recall, thresholds_pr = precision_recall_curve(y_test, y_pred_prob)

# Plotting the Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label='Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()
```



9.5 Threshold Analysis

Analyzing the effect of different decision thresholds on precision, recall, and F1-score to find the most practical threshold for the model.

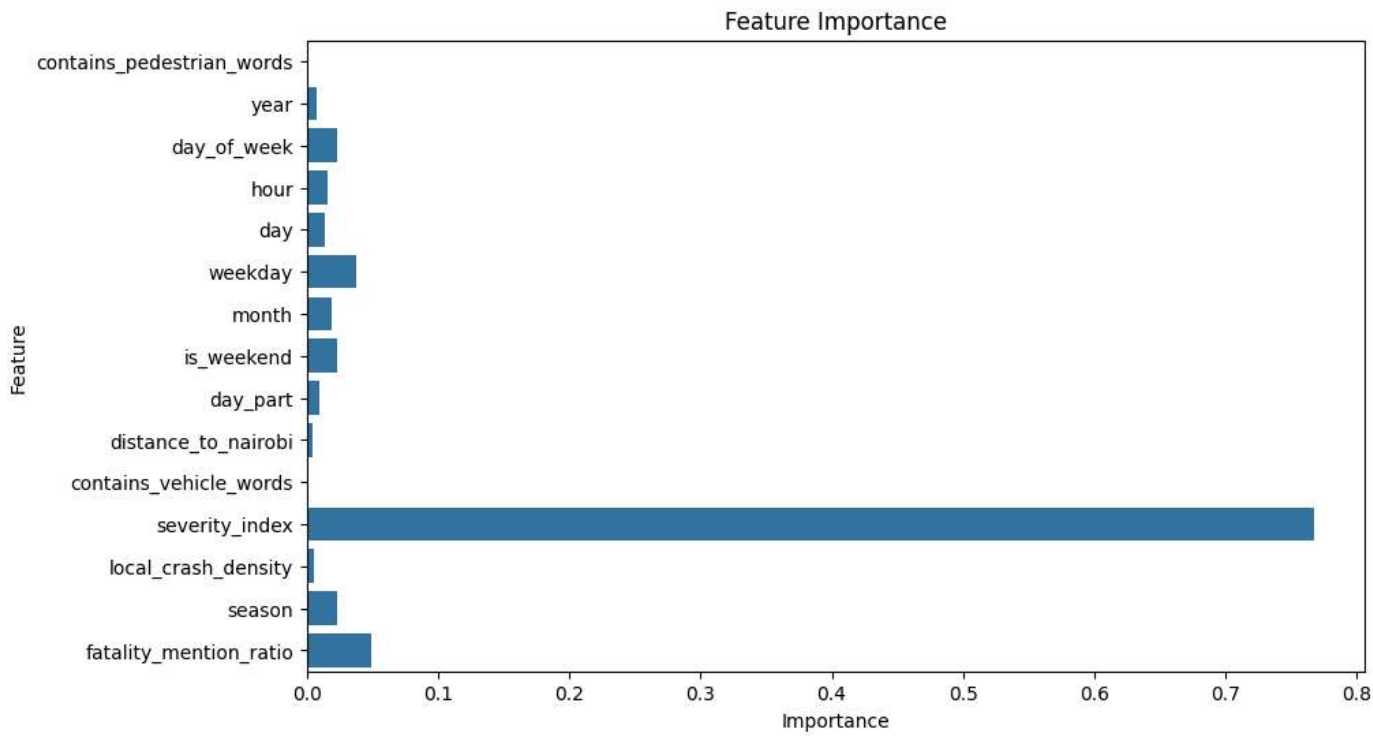
```
# Function to calculate metrics at different thresholds
threshold_metrics = []

for threshold in np.arange(0.1, 1.0, 0.1):
    y_pred_threshold = (y_pred_prob >= threshold).astype(int)
    precision = precision_score(y_test, y_pred_threshold)
    recall = recall_score(y_test, y_pred_threshold)
    f1 = f1_score(y_test, y_pred_threshold)
    threshold_metrics.append((threshold, precision, recall, f1))

# Converting to DataFrame

threshold_df = pd.DataFrame(threshold_metrics, columns=['Threshold', 'Precision', 'Recall', 'F1'])
threshold_df.plot(x='Threshold', y=['Precision', 'Recall', 'F1'], marker='o')
plt.title('Metrics at Different Thresholds')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.show()
```



9.6 Feature Impact Analysis

Using SHAP values and feature importance to interpret model predictions and assess the impact of each feature on predictions.

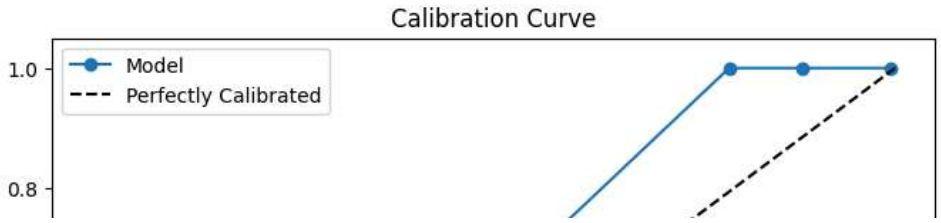
```
importances = best_model.feature_importances_  # Extract feature importances
features = X_train.columns                      # Column names of training features

# Adjust lengths if there's a mismatch
min_length = min(len(importances), len(features))
importances = importances[:min_length]
features = features[:min_length]


plt.figure(figsize=(10, 6))
sns.barplot(x=importances, y=features)
plt.title('Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



9.7 Model Calibration

Evaluating if the model's probability estimates align with actual observed probabilities and calibrating the model if needed.

```
from sklearn.calibration import calibration_curve

# Calibration curve
prob_true, prob_pred = calibration_curve(y_test, y_pred_prob, n_bins=10)

# Plotting calibration curve
plt.figure(figsize=(8, 6))
plt.plot(prob_pred, prob_true, marker='o', label='Model')
plt.plot([0, 1], [0, 1], 'k--', label='Perfectly Calibrated')
plt.xlabel('Predicted Probability')
plt.ylabel('True Probability')
plt.title('Calibration Curve')
plt.legend()
plt.show()
```


Calibration Curve
Model
Perfectly Calibrated
1.0
0.8