ANALYZING AFRICA'S CONFLICT LANDSCAPE TO INFORM BUSINESS INVESTMENTS

BUSINESS UNDERSTANDING

Africa is highly marketable to multinationals due to its vast natural resources, growing consumer market, and expanding economic opportunities. However conflict and Political instability can lead to supply chain disruptions, regulatory hurdles, and heightened security expenses, making it difficult for businesses to operate efficiently. Additionally, conflicts often result in workforce displacement, financial volatility, and reputational risks. Key Risk Factors regarding the conflict such as event forecasting, conflict risk Score, conflict risk classification are key considerations that multinational organizations are going to be interested in when deciding which country to invest in and also when crafting their risk management strategies. This project examines these factors using a structured data driven approach. We used a data set from https://acleddata.com/about-acled/ to;

1. Investigate insights on key risk factors causing the conflict, event forecasting, conflict risk score, and risk classification.

2. Analyze the data set to identify conflict prone areas, forecast conflict areas, establish conflict score for countries, risk classification for different countries.

3. Perform statistical tests to understand which areas are prone to conflict, their risk scores, forecast areas in future likely to experience conflict and classify different types of conflict.

4. Provide actionable recommendations to multinationals on conflicts in Africa in terms of areas prone to conflicts and types of conflict so they can formulate good risk management strategies.

PROBLEM STATEMENT

Our data science consulting firm has been hired by a UK based multinational organization (VMagic Leather Processing Factory) that is interested in investing in Africa. The mission of the firm is to establish a manufacturing plant in a location that is less prone to conflict and political instability. The company has tasked us to analyze data for different countries in Africa and come up with a model that is going to help them decide which country to invest in. Our goal is to formulate a model that can accurately determine conflict hot spots, come up with a risk score, and forecast conflict events.

OBJECTIVES

1. Conflict Analysis in Africa: Establish a foundational understanding of conflict types, high-risk regions, and underlying causes by analyzing relevant data.

2. Dataset Exploration: Gain familiarity with the dataset's structure, terminology, and quality by examining key variables, including numerical and categorical data.

3. Business Insights for Site Selection: Assist a multinational client in identifying conflict-free locations for a manufacturing facility by:

- Developing a predictive model to forecast conflict occurrences based on historical data.

- Identifying key conflict drivers through correlation analysis of variables such as fatalities, disorder types, and actor involvement.
- Assessing spatial and temporal trends to understand conflict variations across regions and time periods.
- Evaluating conflict severity by predicting the likelihood of high-fatality events.
- Enhancing early warning systems through geographic risk assessments.

4. Statistical and Predictive Analysis
- Conduct descriptive and exploratory analysis using measures such as mean, median, variance, and frequency distribution.
- Perform correlation analysis to examine relationships between key variables.
- Apply hypothesis testing, including ANOVA, to compare disorder types across locations.
- Use predictive modeling techniques like logistic regression to assess conflict probabilities.
-Conduct trend analysis to visualize conflict patterns over time.

5. Summary and Recommendations
- Synthesize findings and provide strategic recommendations
- Identifying the most suitable country for investment and highlighting conflict-driving factors.

DATA UNDERSTANDING

In this project, we worked with the ACLED dataset—a comprehensive repository of real-time data on political violence, protests, and other politically relevant events from around the world. The dataset captures a wide array of details, including unique event identifiers, precise dates, event types and sub-types, information about the actors involved, geographic coordinates, and additional contextual attributes such as reported fatalities, tags, and sources. The raw data is collected via a global network that aggregates reports from local, regional, and national media, government publications, and over 50 local data collection partners, covering more than 100 languages and updated on a weekly basis. Data can be accessed through ACLED's export tools, curated downloads, or an API, and is accompanied by comprehensive documentation that clearly defines each feature, ensuring analytical clarity The dataset is extensive with 413,948 rows and 31 features.

Below is a table summarizing the key features of the ACLED dataset along with their meanings:

Feature	Description
event_id_cnty	A unique alphanumeric identifier combining a numeric ID with a country code (e.g., ETH9766).
event_date	The date on which the event occurred, recorded in YYYY-MM-DD format.
year	The year during which the event took place.
time_precision	A numeric code (1–3) indicating the precision of the event date, with 1 being the most precise.
disorder_type	The overarching category of the event, such as Political violence, Demonstrations, or Strategic developments.
event_type	The general classification of the event (e.g., Battles, Protests, Explosions/Remote violence).
sub_event_type	A more specific classification within the event type (e.g., Armed clash, Peaceful protest).
actor1	The primary actor involved in the event.
assoc_actor_1	Additional or supporting actor(s) associated with actor1.
inter1	A categorical code describing actor1's type (e.g., Rebel group, State forces).
actor2	The secondary actor involved in the event (which may represent a target or another involved party).
assoc_actor_2	Additional or supporting actor(s) associated with actor2.
inter2	A categorical code describing actor2's type.

Feature	Description
interaction	A combined description derived from inter1 and inter2, indicating the nature of the interaction between the actors.
civilian_targeting	An indicator specifying whether civilians were the primary target (e.g., "Civilians targeted").
iso	A three-digit ISO numeric code representing the country where the event occurred.
region	The geographic region in which the event took place (e.g., Eastern Africa).
country	The country in which the event occurred.
admin1	The primary sub-national administrative region (e.g., state or province).
admin2	The secondary sub-national administrative region (if applicable).
admin3	The tertiary sub-national administrative region (if available).
location	The specific name of the place where the event occurred.
latitude	The latitude coordinate of the event location in decimal degrees.
longitude	The longitude coordinate of the event location in decimal degrees.
geo_precision	A numeric code (1–3) indicating the precision of the geographic data, with 1 being the most precise.
source	The source(s) used to report the event, which may include multiple sources separated by semicolons.
source_scale	Indicates the geographic closeness of the source to the event (e.g., Local partner, National).
notes	Additional descriptive information about the event.
fatalities	The number of reported fatalities resulting from the event (0 if none are reported).
tags	Keywords or structured tags that provide additional context to the event (e.g., "women targeted: politicians").
timestamp	A Unix timestamp representing when the event was uploaded to the ACLED API, capturing the exact date and time of data entry.

Furthermore, ACLED categorizes its events using a detailed classification system outlined in Table 2. This table presents the different event types, their associated sub-event types, and the overarching disorder categories. The event type provides a broad categorization—such as Battles or Protests—while the sub-event types offer more granular details (for example, distinguishing between an "Armed clash" and a "Government regains territory" within Battles). The disorder type further contextualizes the event by indicating whether it falls under political violence, demonstrations, or strategic developments. This structured taxonomy is essential for accurately analyzing conflict dynamics and understanding the nuanced nature of political violence.

Event Type	Sub-Event Types	Disorder Type
Battles	Government regains territory, Non-state actor overtakes territory, Armed clash	Political violence
Protests	Excessive force against protesters, Protest with intervention, Peaceful protest	Political violence; Demonstrations
Riots	Violent demonstration, Mob violence	Political violence; Demonstrations
Explosions/Remote violence	Chemical weapon, Air/drone strike, Suicide bomb, Shelling/artillery/missile attack, Remote explosive/landmine/IED, Grenade	Political violence
Violence against civilians	Sexual violence, Attack, Abduction/forced disappearance	Political violence
Strategic developments	Agreement, Arrests, Change to group/activity, Disrupted weapons use, Headquarters or base established, Looting/property destruction, Non-violent transfer of territory, Other	Strategic developments

Importing Libraries

In [55]:

```
# Importing necessary libraries
```

```
import pandas as pd
import numpy as np
import csv
import folium
import requests
import json

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Machine learning utilities
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, Rand
omizedSearchCV
from sklearn.metrics import accuracy_score,f1_score,recall_score,precision_score,confusio
n_matrix,roc_curve,roc_auc_score,classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# performance metrics
from scipy import stats
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Algorithms for supervised learning methods
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
import xgboost as xgb

# Filtering future warnings
import warnings
warnings.filterwarnings('ignore')
```

Loading the Dataset

In [56]:

```
# Load the dataset
df = pd.read_csv('../Africa_1997-2025_Feb28.csv')
```

Exploring the structure of the Dataset

In [57]:

```
print("Dataset Overview:")
# checking the shape of the dataset
print(f"This dataset contains {df.shape[0]} rows and {df.shape[1]} columns\n")

# Checking the Columns
print(f"This are the columns available in the dataset {df.columns} columns")

#Checking the Data Type
print(f"The shows more information about the dataset, that is the entries and data type p
er column\n")
print(df.info())  # Column names, data types, and non-null counts

# Preview the first few rows of data
print("\nFirst 5 Rows:")
df.head()
```

```
Dataset Overview:
This dataset contains 413947 rows and 31 columns

This are the columns available in the dataset Index(['event_id_cnty', 'event_date', 'year
', 'time_precision',
       'disorder_type', 'event_type', 'sub_event_type', 'actor1',
       'assoc_actor_1', 'inter1', 'actor2', 'assoc_actor_2', 'inter2',
```

```
       'interaction', 'civilian_targeting', 'iso', 'region', 'country',
       'admin1', 'admin2', 'admin3', 'location', 'latitude', 'longitude',
       'geo_precision', 'source', 'source_scale', 'notes', 'fatalities',
       'tags', 'timestamp'],
      dtype='object') columns
The shows more information about the dataset, that is the entries and data type per colum
n

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 413947 entries, 0 to 413946
Data columns (total 31 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   event_id_cnty       413947 non-null  object
 1   event_date          413947 non-null  object
 2   year                413947 non-null  int64
 3   time_precision      413947 non-null  int64
 4   disorder_type       413947 non-null  object
 5   event_type          413947 non-null  object
 6   sub_event_type      413947 non-null  object
 7   actor1              413947 non-null  object
 8   assoc_actor_1       114747 non-null  object
 9   inter1              413947 non-null  object
 10  actor2              301956 non-null  object
 11  assoc_actor_2       85527 non-null   object
 12  inter2              301956 non-null  object
 13  interaction         413947 non-null  object
 14  civilian_targeting  123147 non-null  object
 15  iso                 413947 non-null  int64
 16  region              413947 non-null  object
 17  country             413947 non-null  object
 18  admin1              413930 non-null  object
 19  admin2              410127 non-null  object
 20  admin3              209892 non-null  object
 21  location            413947 non-null  object
 22  latitude            413947 non-null  float64
 23  longitude           413947 non-null  float64
 24  geo_precision       413947 non-null  int64
 25  source              413947 non-null  object
 26  source_scale        413947 non-null  object
 27  notes               413947 non-null  object
 28  fatalities          413947 non-null  int64
 29  tags                94466 non-null   object
 30  timestamp           413947 non-null  int64
dtypes: float64(2), int64(6), object(23)
memory usage: 97.9+ MB
None

First 5 Rows:
```

Out[57]:

	event_id_cnty	event_date	year	time_precision	disorder_type	event_type	sub_event_type	actor1	assoc
0	MLI33921	2025-02-28	2025	1	Political violence	Explosions/Remote violence	Remote explosive/landmine/IED	JNIM: Group for Support of Islam and Muslims	
1	BFO13376	2025-02-28	2025	1	Political violence	Battles	Armed clash	JNIM: Group for Support of Islam and Muslims	
2	MLI33922	2025-02-	2025	1	Political	Violence against	Attack	Military Forces	

	event_id_cnty	event_date	year	time_precision	disorder_type	event_type	sub_event_type	actor1	assoc
3	GHA2795	2025-02-28	2025	1	Political violence	Violence against civilians	Attack	Land Guards	
4	GHA2800	2025-02-28	2025	1	Political violence	Riots	Mob violence	Rioters (Ghana)	

5 rows × 31 columns

In [58]:

```
# Checking the Value counts for Categorical columns eg. 'region', 'country', 'event_type'
, 'disorder_type'
print("\nValue counts for 'region':")
display(df['region'].value_counts())

print("\nValue counts for 'country':")
display(df['country'].value_counts())

print("\nValue counts for 'event_type':")
display(df['event_type'].value_counts())

print("\nValue counts for 'disorder_type':")
display(df['disorder_type'].value_counts())
```

```
Value counts for 'region':

region
Eastern Africa     129294
Northern Africa     93960
Western Africa      92353
Middle Africa       65031
Southern Africa     33309
Name: count, dtype: int64

Value counts for 'country':

country
Somalia                                         48651
Nigeria                                         40480
Democratic Republic of Congo                    33961
Sudan                                           33287
South Africa                                    22662
Kenya                                           16693
Cameroon                                        16248
Ethiopia                                        14599
South Sudan                                     14422
Morocco                                         13059
Algeria                                         12712
Mali                                            12302
Burkina Faso                                    12079
Libya                                           11898
Egypt                                           11801
Tunisia                                         11203
Burundi                                         10868
Uganda                                           9183
Central African Republic                         7638
Zimbabwe                                         6710
Sierra Leone                                     4982
Mozambique                                       4761
```

```
Niger                                                4703
Madagascar                                           4110
Angola                                               4043
Ivory Coast                                          3086
Guinea                                               2988
Ghana                                                2675
Senegal                                              2114
Mauritania                                           2027
Chad                                                 1980
Liberia                                              1725
Benin                                                1648
Malawi                                               1588
Zambia                                               1544
Namibia                                              1188
Tanzania                                             1170
Rwanda                                                853
eSwatini                                              805
Reunion                                               625
Togo                                                  597
Mauritius                                             580
Republic of Congo                                     534
Gabon                                                 508
Eritrea                                               408
Guinea-Bissau                                         375
Mayotte                                               352
Gambia                                                299
Cape Verde                                            273
Lesotho                                               249
Djibouti                                              219
Comoros                                               193
Botswana                                              149
Equatorial Guinea                                      77
Sao Tome and Principe                                  42
Seychelles                                             19
Saint Helena, Ascension and Tristan da Cunha            2
Name: count, dtype: int64

Value counts for 'event_type':

event_type
Battles                       105556
Violence against civilians    100372
Protests                       94993
Riots                          42737
Strategic developments         38984
Explosions/Remote violence     31305
Name: count, dtype: int64

Value counts for 'disorder_type':

disorder_type
Political violence                    256514
Demonstrations                        115877
Strategic developments                 38984
Political violence; Demonstrations      2572
Name: count, dtype: int64
```

In [59]:

```
# Summary statistics for numerical columns
print("\nSummary Statistics:")
print(df.describe())  # Mean, min, max, etc., for numerical columns
```

```
Summary Statistics:
                year  time_precision            iso       latitude  \
count  413947.000000   413947.000000  413947.000000  413947.000000
mean     2017.527964        1.130737     510.399988       6.922263
std         6.608510        0.393477     250.075057      15.495063
min      1997.000000        1.000000      12.000000     -34.706800
25%      2015.000000        1.000000     231.000000       0.315600
50%      2020.000000        1.000000     566.000000       6.693600
75%      2022.000000        1.000000     710.000000      13.515700
max      2025.000000        3.000000     894.000000      37.281500
```

```
           longitude  geo_precision     fatalities     timestamp
count  413947.000000  413947.000000  413947.000000  4.139470e+05
mean       21.886800       1.279074       2.439153  1.676141e+09
std        16.776814       0.494604      24.320917  5.263689e+07
min       -25.163100       1.000000       0.000000  1.552576e+09
25%         8.155500       1.000000       0.000000  1.622068e+09
50%        28.043600       1.000000       0.000000  1.689711e+09
75%        33.483300       2.000000       1.000000  1.724714e+09
max        64.683200       3.000000    1350.000000  1.741072e+09
```

Data Cleaning

Data Cleaning Plan

1. Dropping irrelevant data i.e columns that do not add value to the analysis

2. Handling Missing Data Fill or drop missing values based on relevance.

3. Removing Duplicates if any exist.

4. Detect and remove extreme values that could distort analysis that is remove outliers

5. Cleaning Text Data that is renaming columns and correct of typos

NB: for the missing values this is the criteria that will be used:

> If a column has 0% missing values, no cleaning is needed.

> If a column has less than 5% missing values, you can either drop the missing rows or fill them using mean/median/mode.

> If a column has more than 50% missing values, it may be better to drop the column entirely.

The following columns will be dropped as they are irrelevant to our anlaysis

> event_id_cnty – Unique identifier for events (not useful for prediction).

> timestamp – A Unix timestamp; redundant with event_date.

> notes – Descriptive text that is difficult to quantify for modeling.

> source & source_scale – Identifies where the data was collected but may not contribute to predicting future conflict.

> tags – Sparse data and may not be structured enough for modeling. > assoc_actor_1 & assoc_actor_2 – Frequently missing values; may not provide strong predictive power. yet it is backed up by actor

> iso - The iso column represents the ISO country code, which is a numeric identifier for each country.

> time_precision - The time_precision column indicates how precise the event date is, hence it less useful.

In [60]:

```python
# Checking for duplicates
# Checking for missing values
print("\n The number of duplicates in the dataset is:")
print(df.duplicated().sum())

# Checking for missing values
print("\nMissing Values:")
print(df.isnull().sum())  # Count missing values in each column
```

```
 The number of duplicates in the dataset is:
0

Missing Values:
```

```
event_id_cnty              0
event_date                 0
year                       0
time_precision             0
disorder_type              0
event_type                 0
sub_event_type             0
actor1                     0
assoc_actor_1         299200
inter1                     0
actor2                111991
assoc_actor_2         328420
inter2                111991
interaction                0
civilian_targeting    290800
iso                        0
region                     0
country                    0
admin1                    17
admin2                  3820
admin3                204055
location                   0
latitude                   0
longitude                  0
geo_precision              0
source                     0
source_scale               0
notes                      0
fatalities                 0
tags                  319481
timestamp                  0
dtype: int64
```

In [61]:

```
df.drop(columns=[
    'event_id_cnty',
    'event_date',
    'time_precision',
    'assoc_actor_1',
    'assoc_actor_2',
    'iso',
    'source',
    'source_scale',
    'notes',
    'tags',
    'timestamp',
], inplace=True)
# preview the columns
df.columns
```

Out[61]:

```
Index(['year', 'disorder_type', 'event_type', 'sub_event_type', 'actor1',
       'inter1', 'actor2', 'inter2', 'interaction', 'civilian_targeting',
       'region', 'country', 'admin1', 'admin2', 'admin3', 'location',
       'latitude', 'longitude', 'geo_precision', 'fatalities'],
      dtype='object')
```

In [62]:

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

Out[62]:

```
year                       0
disorder_type              0
event_type                 0
sub_event_type             0
actor1                     0
inter1                     0
actor2                111991
```

```
actor2                 111991
inter2                 111991
interaction                 0
civilian_targeting     290800
region                      0
country                     0
admin1                     17
admin2                   3820
admin3                 204055
location                    0
latitude                    0
longitude                   0
geo_precision               0
fatalities                  0
dtype: int64
```

Plan to handle missing values

Column	Missing values	Solution and reason
actor2	111,991	Fill with ***Unknown*** Some events may involve only one actor (e.g., protests, riots)
inter2	111,991	Fill with ***Unknown*** If actor2 is missing, inter2 (interaction type) is also unknown
civilian_targeting	290,800	Fill with ***No*** Missing values likely mean civilians were not targeted
admin1	17	***Drop rows*** only a few missing, so dropping won't affect the dataset much
admin2	3,820	Fill with ***Unknown*** or mode District info missing, but "Unknown" preserves all events
admin3	204,055	***Drop column*** Too many missing values, making it unreliable for analysis

In [63]:

```
# fill the rows with missing values
df[['actor2', 'inter2', 'admin2',]] = df[['actor2', 'inter2', 'admin2',]].fillna("Unknown
")
df[['civilian_targeting']] = df[['civilian_targeting']].fillna("No")
```

In [64]:

```
"""
The civilian_targeting has two entries NO and civilian_targeting, for a better readibilit
y
replacing civilian_targeting with yes will be the enhance the readability of this column
"""
# replace civilian_targeting with Yes
df['civilian_targeting'] = df['civilian_targeting'].replace({'Civilian targeting': 'Yes'
})
# preview changes
df['civilian_targeting'].unique()
```

Out[64]:

```
array(['No', 'Yes'], dtype=object)
```

In [65]:

```
"""
The civilian_targeting has two entries NO and civilian_targeting, for a better readibilit
y
replacing civilian_targeting with yes will be the enhance the readability of this column
"""
# replace civilian_targeting with Yes
df['civilian_targeting'] = df['civilian_targeting'].replace({'Civilian targeting': 'Yes'
})
# preview changes
df['civilian_targeting'].unique()
```

Out[65]:

```
array(['No', 'Yes'], dtype=object)
```

In [66]:

In [66]:

```
# droping nan values and dropping column
df.dropna(subset=['admin1'], inplace=True)
# drop column
#df.drop('assoc_actor_2', axis=1, inplace=True)
```

In [67]:

```
# preview the missing values
print(f"The dataset contains {df.isna().sum().sum()} missing values")
```

```
The dataset contains 204038 missing values
```

In [68]:

```
print(df.isnull().sum())
```

```
year                      0
disorder_type             0
event_type                0
sub_event_type            0
actor1                    0
inter1                    0
actor2                    0
inter2                    0
interaction               0
civilian_targeting        0
region                    0
country                   0
admin1                    0
admin2                    0
admin3               204038
location                  0
latitude                  0
longitude                 0
geo_precision             0
fatalities                0
dtype: int64
```

In [69]:

```
# checking for duplicates
print(f"The dataset contains {df.duplicated().sum()} duplicated rows")
```

```
The dataset contains 120924 duplicated rows
```

In [70]:

```
"""
The dataset contain 120924 duplicated rows, to maintain data consistency while avoiding r
edundancy,
we will keep only the first occurrence of each duplicate row and remove the rest
"""
# dropping duplicates while keeping first occurrence
df.drop_duplicates(keep="first", inplace=True)
# preview
print(f"The dataset contains {df.duplicated().sum()} duplicated rows")
```

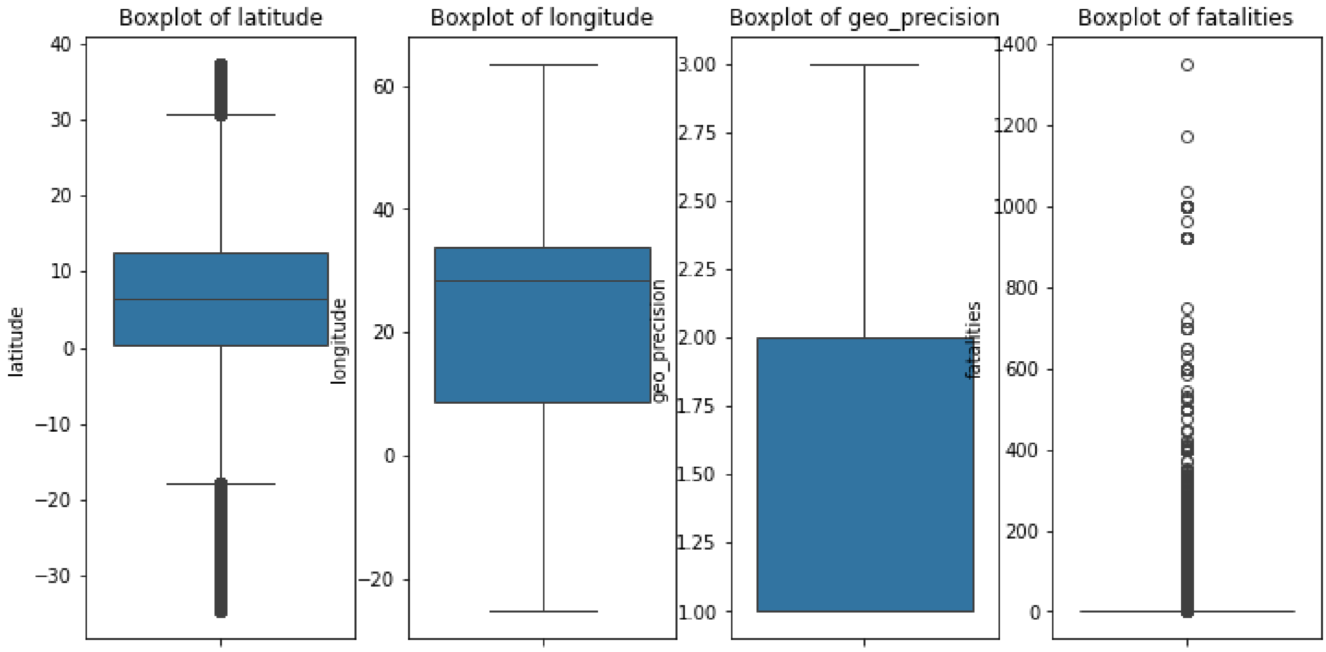
```
The dataset contains 0 duplicated rows
```

Detecting and handling outliers

In [71]:

```
# Select numerical columns
num_cols = ['latitude', 'longitude', 'geo_precision', 'fatalities']
# Plot boxplots
plt.figure(figsize=(12, 6))
for i, col in enumerate(num_cols):
    plt.subplot(1, len(num_cols), i + 1)
```

```
    sns.boxplot(y=df[col])
    plt.title(f'Boxplot of {col}')
plt.show()
```



Outlier detection was conducted on numerical columns to identify potential extreme values. However, not all numerical columns require outlier removal.

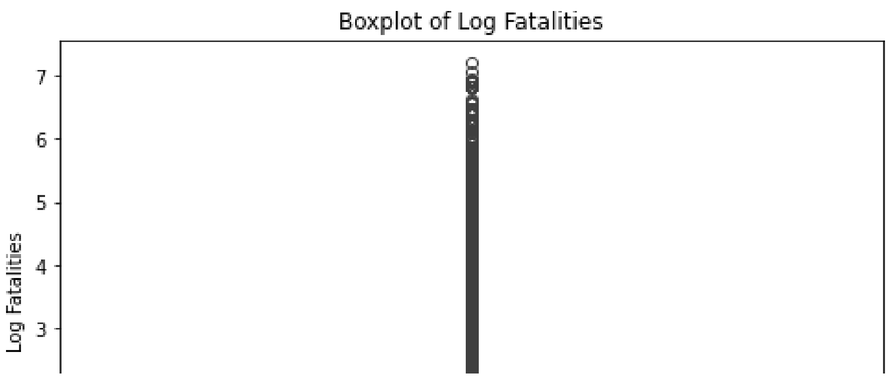
> * **Latitude & Longitude - It is a numerical column but it does not need outlier removal reason their extreme values do not necessarily indicate errors but rather represent real-world geographic locations, including conflict-prone areas. After checking for invalid coordinates none was found hence all data points were retained.**

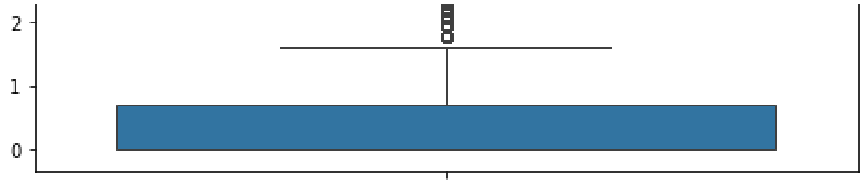
> * **Geo Precision - is a numerical column that represents categorical levels of location accuracy for example, 1 = most precise, 3 = least precise and etc hence outlier detection does not apply here, so no modifications were made.**

> * **Fatalities is a numerical column where outlier detected alot of extreme numbers. In this case extreme numbers represent real conflict events rather than errors. Instead of removing these values as outliers, applying log transformation will help to reduce skewness while preserving valuable information.**

In [72]:

```
# Apply log transformation to fatalities
df['log_fatalities'] = np.log1p(df['fatalities'])
# preview viz
# Plot the boxplot
plt.figure(figsize=(8, 5))
sns.boxplot(y=df['log_fatalities'])
plt.title("Boxplot of Log Fatalities")
plt.ylabel("Log Fatalities")
plt.show()
```





The log transformation applied to fatalities to reduced skewness and the impact of extreme values, making the distribution more manageable. This helps prevent the model from being overly influenced by very high fatality counts while preserving meaningful patterns.

Exploratory Data Analysis (EDA)

Univariate Analysis

In [73]:

```
# for this anlayis we will copy our data and rename df to df1
df1 = df.copy()
```

In [74]:

```
# check the statistical distribution
df1.describe()
```

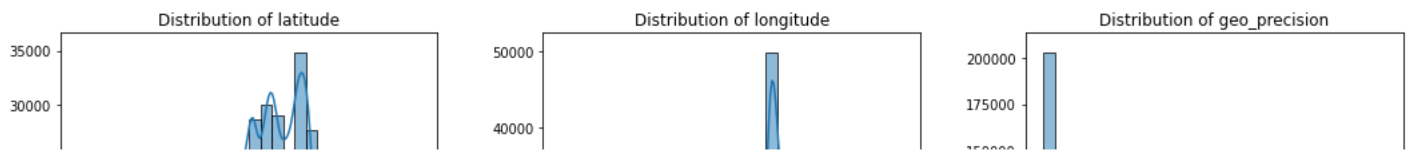
Out[74]:

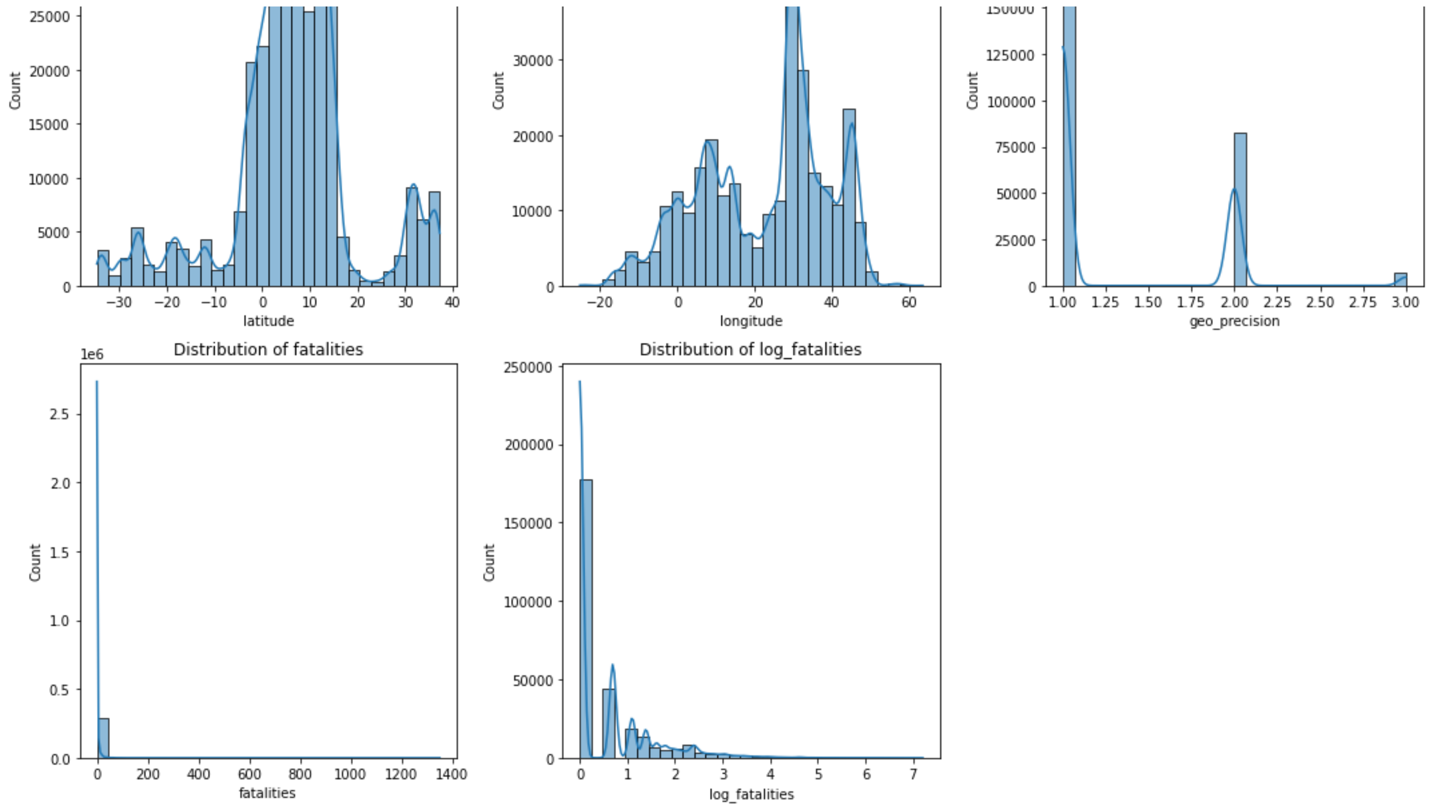
	year	latitude	longitude	geo_precision	fatalities	log_fatalities
count	293006.000000	293006.000000	293006.000000	293006.000000	293006.000000	293006.000000
mean	2017.505027	6.058650	22.679520	1.331079	2.693508	0.559398
std	6.601442	13.573443	16.250923	0.519928	18.015479	0.878351
min	1997.000000	-34.706800	-25.163100	1.000000	0.000000	0.000000
25%	2015.000000	0.340100	8.898800	1.000000	0.000000	0.000000
50%	2020.000000	6.416700	28.580000	1.000000	0.000000	0.000000
75%	2022.000000	12.495750	33.888500	2.000000	1.000000	0.693147
max	2025.000000	37.281500	63.475000	3.000000	1350.000000	7.208600

a. Numerical Variable Distribution

In [75]:

```
# Select numerical columns
num_cols = ['latitude', 'longitude', 'geo_precision', 'fatalities', 'log_fatalities']
# Create subplots
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
axes = axes.flatten()
# Plot histograms with KDE
for i, col in enumerate(num_cols):
    sns.histplot(df1[col], kde=True, bins=30, ax=axes[i])
    axes[i].set_title(f"Distribution of {col}")
# Remove empty subplot if odd number of plots
fig.delaxes(axes[-1])

plt.tight_layout()
plt.show()
```





Interpretation of Numerical Variable Distributions

> The latitude distribution shows clustering around specific values, indicating conflicts are concentrated in certain regions rather than evenly spread. Some extreme values suggest conflicts occur across a broad range. The peak of the distribution lies North of the equtor suggesting most of the conflicts occur in the northern part of the equator.

> The longitude distribution has multiple peaks, suggesting conflicts occur in distinct regions rather than being uniformly distributed. This reinforces the idea of localized conflict zones. The peaks of the distribution also suggest most conflict occur towards the eastern part of the continent.

> The geo_precision distribution shows most events have high location accuracy (1), but some have lower precision (2,3), indicating location uncertainty in certain records.

> The fatalities distribution is highly skewed, with most events having low fatalities and a few extreme cases. The log transformation compresses the scale, reducing the impact of extreme values while maintaining the trend.

Performing Univariate Analysis in Categorical columns, For redability we will split the columns as follows:

> Event Characteristics - disorder_type, event_type, sub_event_type, civilian_targeting

> Actors & Interactions - actor1, inter1, actor2, inter2, interaction

> Geographic Distribution - region, country, admin1, admin2, location

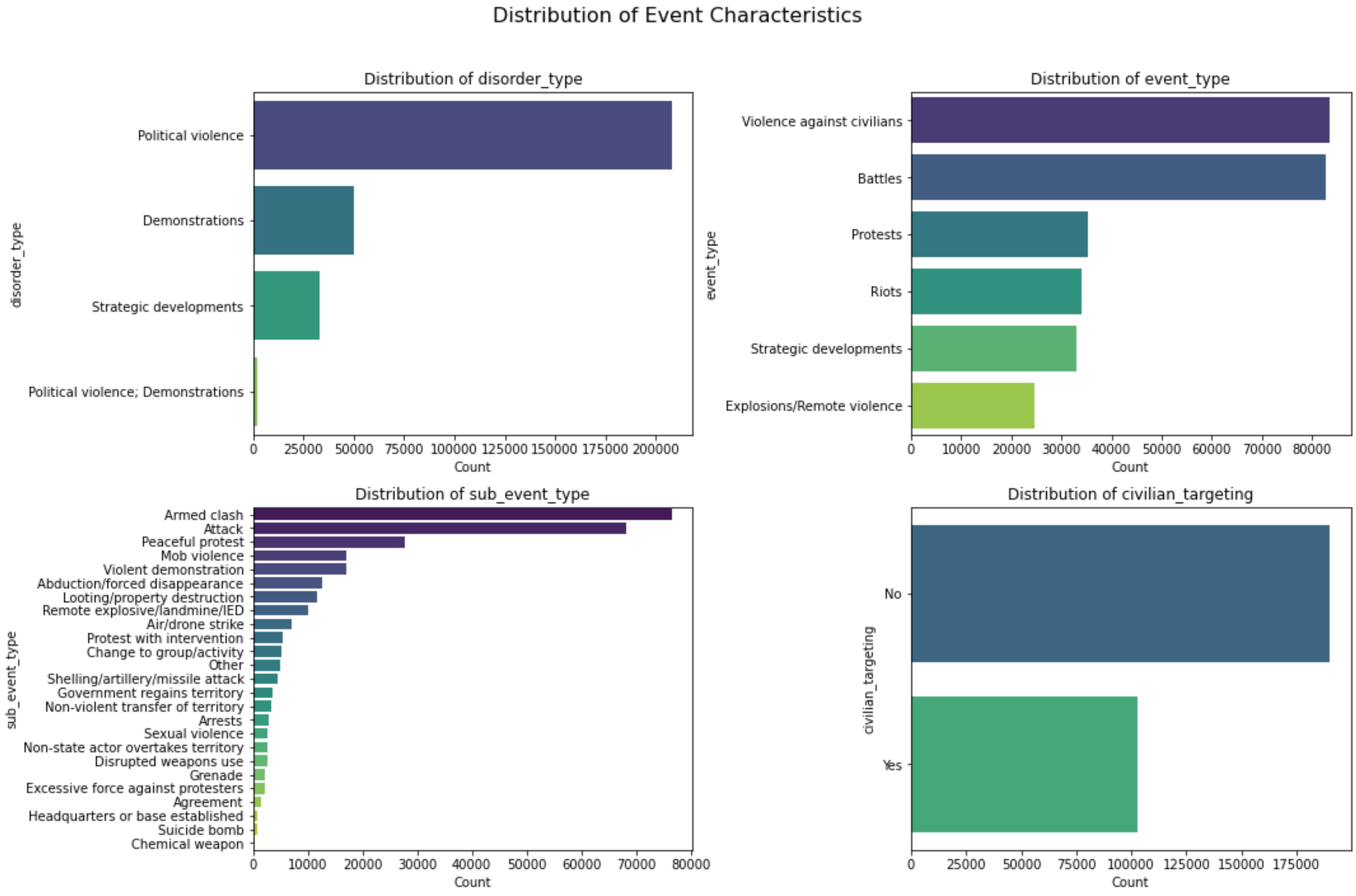
b. Categorical Variable Distribution

(1) Event Characteristics

In [76]:

```
# Define the categorical columns to visualize
event_columns = ['disorder_type', 'event_type', 'sub_event_type', 'civilian_targeting']
# Set up the figure and axes
```

```
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Distribution of Event Characteristics', fontsize=16)
# Iterate through columns and plot
for ax, col in zip(axes.flatten(), event_columns):
    sns.countplot(y=df1[col], order=df1[col].value_counts().index, ax=ax, palette="virid
is")
    ax.set_title(f'Distribution of {col}')
    ax.set_xlabel('Count')
    ax.set_ylabel(col)

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

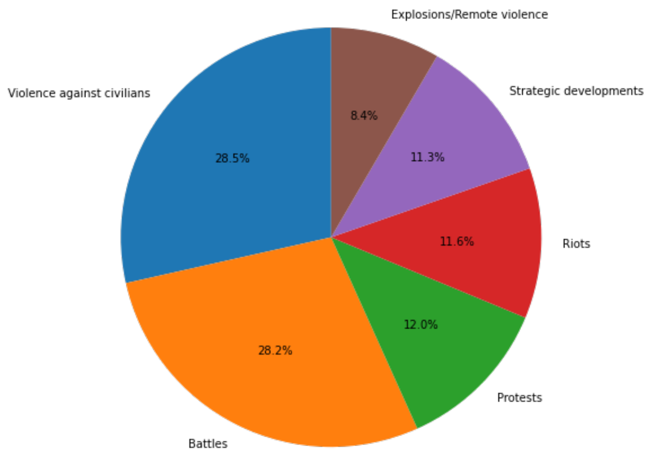


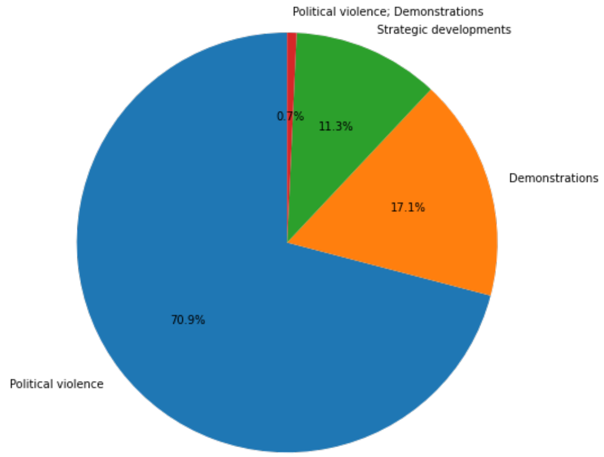
In [77]:

```
# Columns and Titles for plotting
columns = ["event_type", "disorder_type"]  # Double-check if the last one is correct
titles = [
    "Distribution of Events by Event Type",
    "Distribution of Events by Disorder Type",
]

# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(20, 7))  # 1 row, 3 columns

# Loop through each column and plot
for i in range(2):
    df1[columns[i]].value_counts().plot(
        kind='pie',
        autopct='%1.1f%%',
        startangle=90,
        ax=axs[i]
    )
    axs[i].set_title(titles[i])
    axs[i].set_ylabel('')  # Remove ylabel for cleaner look

plt.tight_layout()
plt.show()
```

Distribution of Events by Event Type

Distribution of Events by Disorder Type





Event Characteristics Interpretation

> Disorder Type - Political violence is the most dominant, followed by demonstrations. Strategic developments are less frequent.

> Event Type - Battles and violence against civilians are the most common, highlighting armed conflicts and civilian attacks.

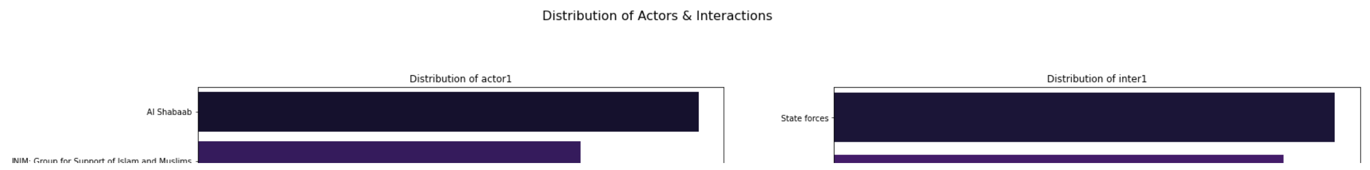
> Sub-event Type - Armed clashes, peaceful protests, and attacks dominate, showing a mix of violent and non-violent actions.

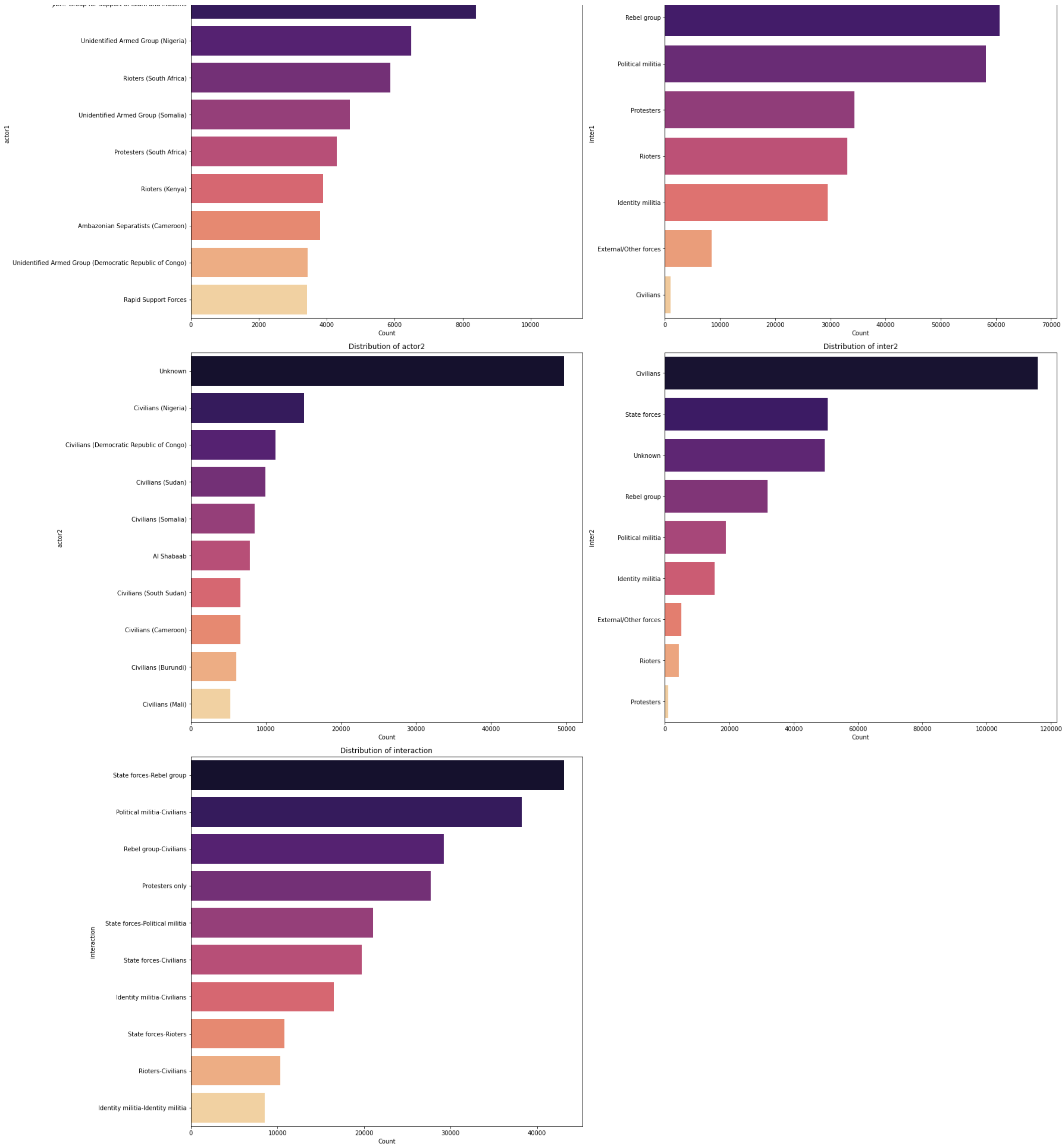
> Civilian Targeting - While most incidents don’t directly target civilians, a significant proportion still does.

(2) Actors & Interactions

In [78]:

```
# Define the categorical columns to visualize
actor_columns = ['actor1', 'inter1', 'actor2', 'inter2', 'interaction']
# Set up the figure and axes dynamically based on the number of columns
num_plots = len(actor_columns)
rows = (num_plots // 2) + (num_plots % 2)  # Ensure enough rows
fig, axes = plt.subplots(rows, 2, figsize=(25, 30))

fig.suptitle('Distribution of Actors & Interactions', fontsize=16)
# Flatten axes only if there are multiple rows
if rows > 1:
    axes = axes.flatten()
else:
    axes = [axes]
# Iterate through columns and plot
for i, col in enumerate(actor_columns):
    sns.countplot(y=df1[col], order=df1[col].value_counts().index[:10], ax=axes[i], pale
tte="magma")  # Top 10 categories
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel('Count')
    axes[i].set_ylabel(col)
# Remove any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])  # Remove empty subplot

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```





Actors & Interactions Interpretation

> Actor 1: Actor 1 being the primary actor involved in the event indicates that Al-Shabaab and protesters are the most frequently mentioned, indicating militant and civil movements.

> Inter1: Inter1 is the actor's 1 type and it indicates that protesters, state forces, and rebel groups are the main interacting entities.

> Actor 2: Actor 2 being the secondary actor involved in the event indicates that civilians are the most affected, with many attacks involving unidentified actors.

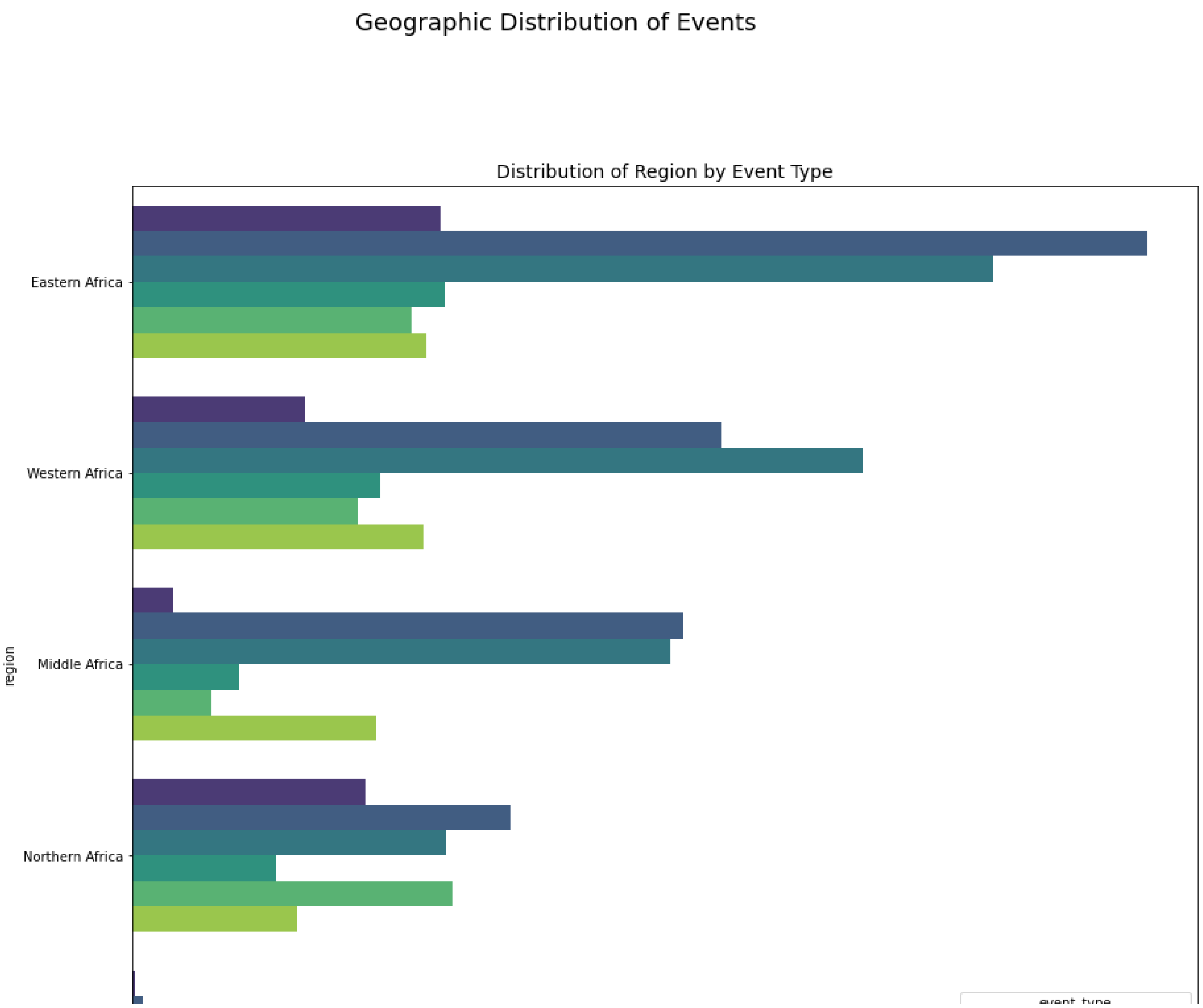
> Inter2: Inter2 describes actor2's type and from the distribution the civilians remain the most targeted group, followed by state forces and rebel groups.

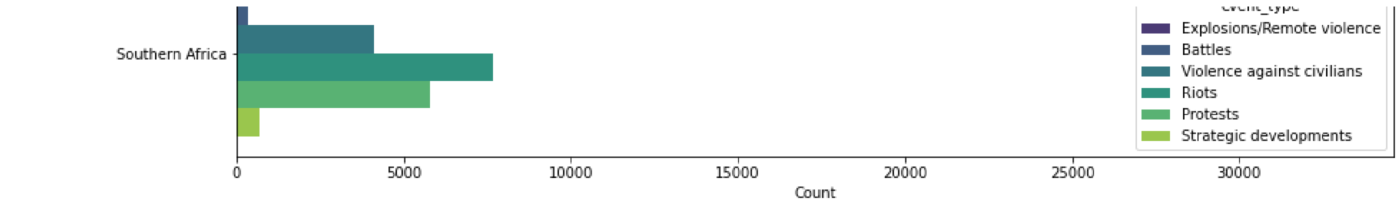
> Interaction: Interactions indicate the nature of interaction between the actors where Protesters vs. state forces and rebel groups vs. state forces are the most common conflict dynamics.

(3) Geographic Distribution

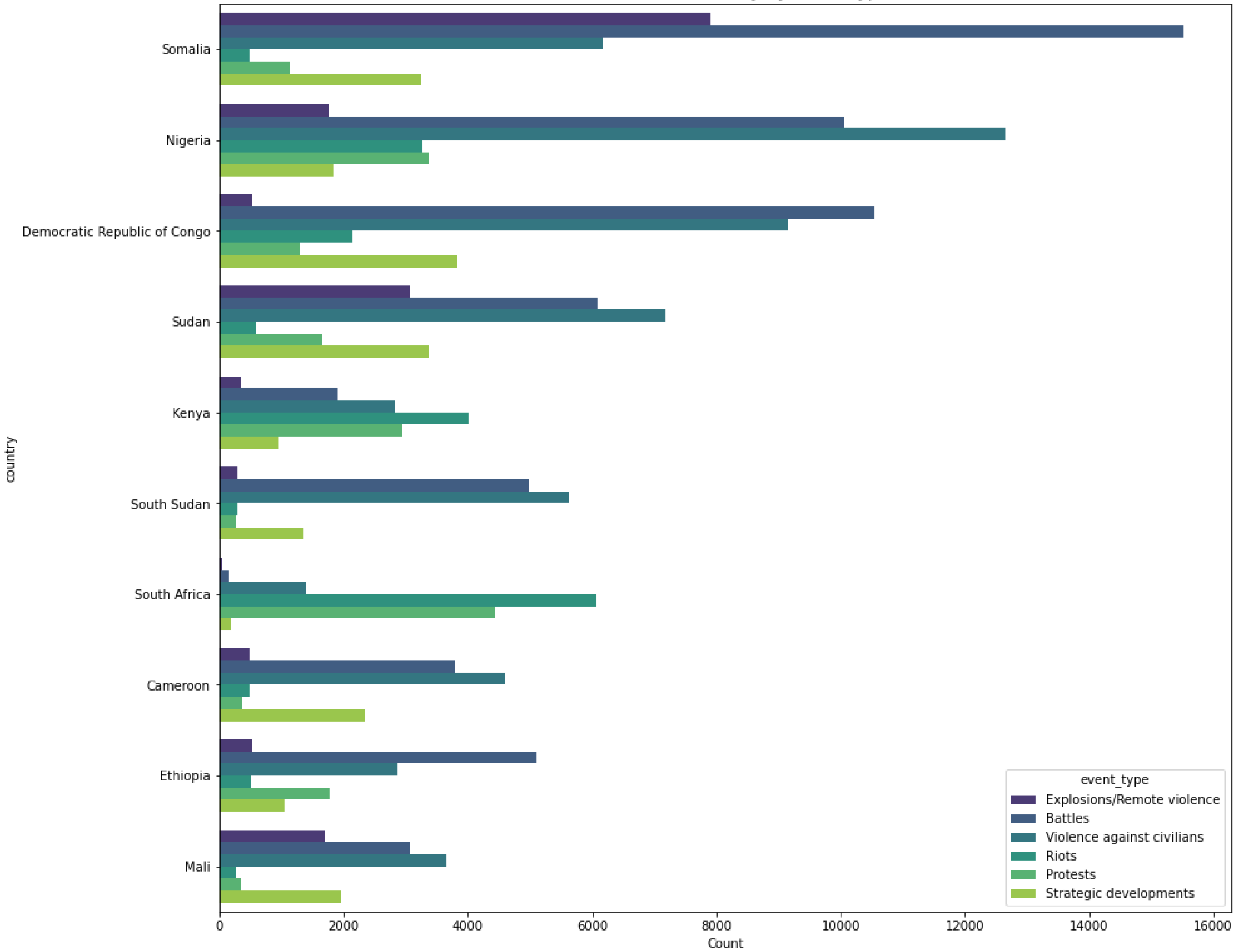
In [79]:

```
# Plan: (column to plot, hue, plot title)
plot_plan = [
    ('region', 'event_type', 'Distribution of Region by Event Type'),
    ('country', 'event_type', 'Distribution of Country by Event Type'),
    ('region', 'disorder_type', 'Distribution of Region by Disorder Type')
]

# Create the figure with len(plot_plan) subplots vertically
fig, axes = plt.subplots(len(plot_plan), 1, figsize=(14, 35))
fig.suptitle('Geographic Distribution of Events', fontsize=18)

# If only one plot, ensure axes is iterable
if len(plot_plan) == 1:
    axes = [axes]

# Loop through the plot plan
for i, (col, hue_col, title) in enumerate(plot_plan):
    sns.countplot(
        y=df1[col],
        hue=df1[hue_col],
        palette="viridis",
        order=df1[col].value_counts().index[:10],  # Optional: Top 10 for readability
        ax=axes[i],
        data=df1
    )
    axes[i].set_title(title, fontsize=14)
    axes[i].set_xlabel('Count')
    axes[i].set_ylabel(col)

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

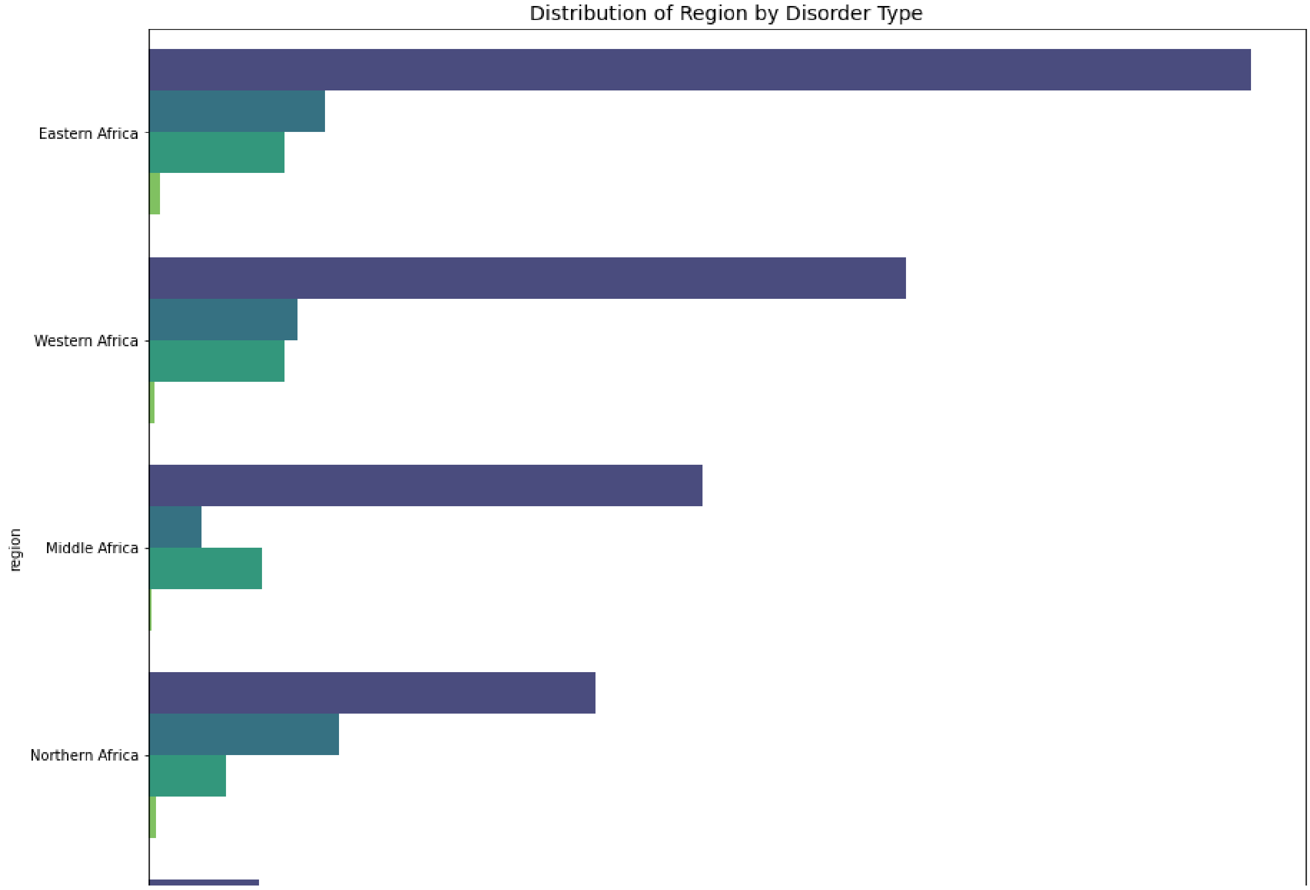


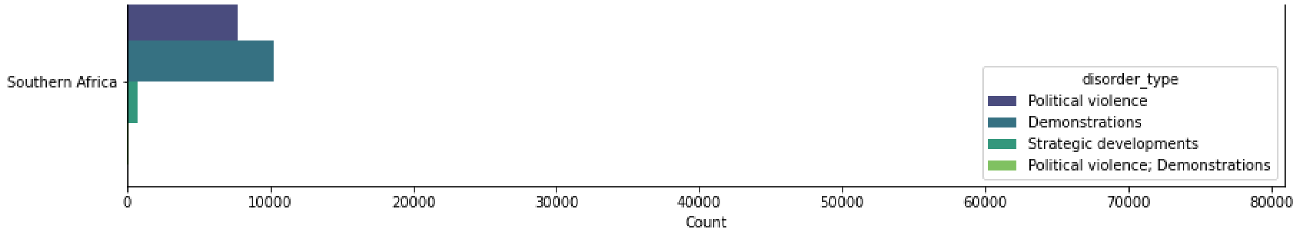


Distribution of Country by Event Type



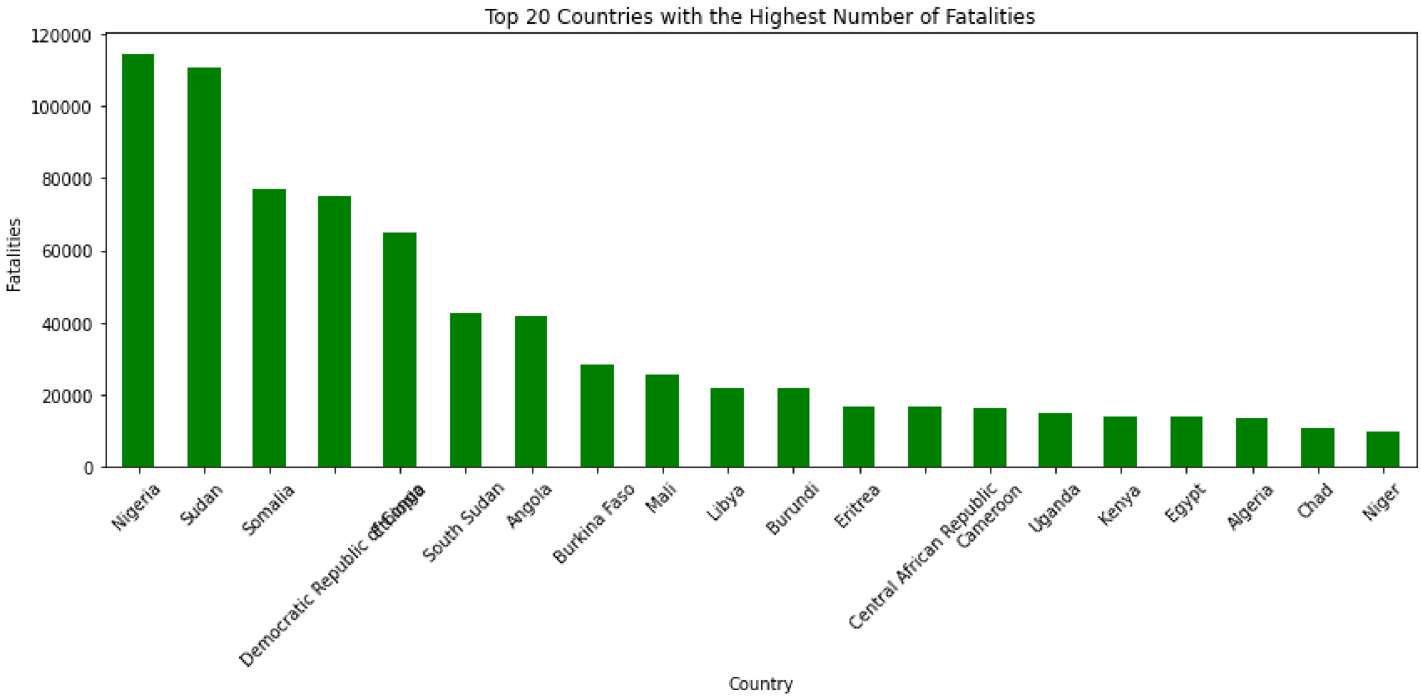
Distribution of Region by Disorder Type





In [80]:

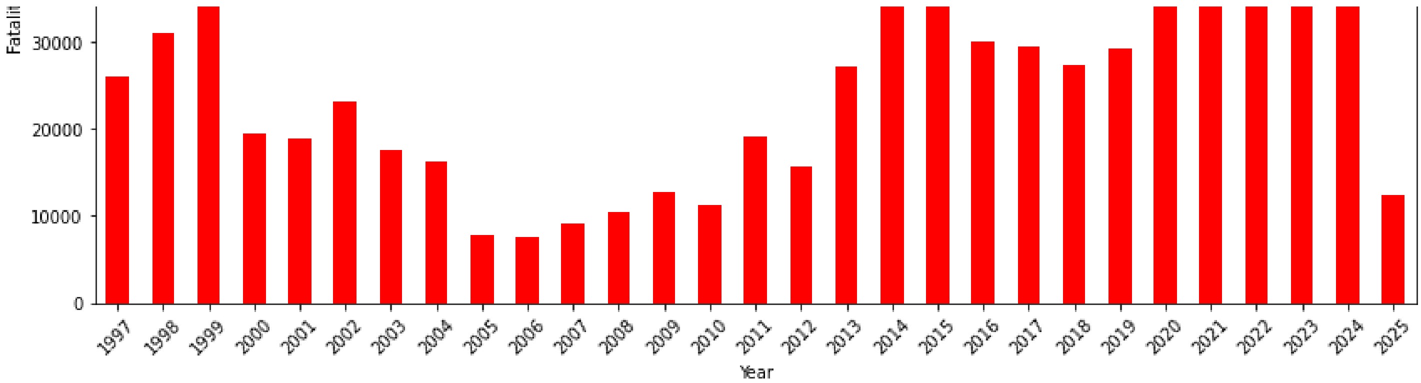
```
# Top 20 countries with the highest number of fatalities
top_countries = df1.groupby('country')['fatalities'].sum().nlargest(20)
plt.figure(figsize=(12, 6))
top_countries.plot(kind='bar', color='g')
plt.title('Top 20 Countries with the Highest Number of Fatalities')
plt.xlabel('Country')
plt.ylabel('Fatalities')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Distribution of Conflict Event over time

In [81]:

```
# Conflict trends over time
conflict_trends = df1.groupby('year')['fatalities'].sum()
plt.figure(figsize=(12, 6))
conflict_trends.plot(kind='bar', color='r')
plt.title('Conflict Trends Over Time')
plt.xlabel('Year')
plt.ylabel('Fatalities')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Mapping Conflict severity based on Fatalities per Country

In [82]:

```
# Download GeoJSON for country boundaries
url = 'https://raw.githubusercontent.com/johan/world.geo.json/master/countries.geo.json'
response = requests.get(url)

if response.status_code == 200:
    geojson_data = response.json()
    with open('countries.geojson', 'w') as f:
        json.dump(geojson_data, f)
    print("GeoJSON downloaded successfully!")
else:
    print("Failed to download")

 # Country names in conflict dataset
data_countries = set(df1['country'].unique())
#Country names in geojson
geo_countries = set(feature['properties']['name'] for feature in geojson_data['features'
])
#Identify countries in the conflict dataset that are not matching the GeoJSON country nam
es
missing_in_geojson = data_countries - geo_countries
print("MISSING countries:", missing_in_geojson)

#Create a Mapping / Replacement Dictionary
replacement_dict = {
    'Tanzania': "United Republic of Tanzania",
    'Guinea-Bissau': "Guinea Bissau",
    'Democratic Republic of Congo':'Democratic Republic of the Congo',
    'Republic of Congo': 'Republic of the Congo',
    'South Sudan': 'South Sudan',  # Only if it exists in GeoJSON, check spelling
    'eSwatini': 'Swaziland',        # Example if needed
}

# Apply the Mapping to Clean the DataFrame
df1['country'] = df1['country'].replace(replacement_dict)


# Dynamically prepare the data from your conflict DataFrame
# Group by 'country' and sum the fatalities
country_data = df1.groupby('country', as_index=False)['fatalities'].sum()

# Inspect to ensure country names align with GeoJSON country property 'name'
print(country_data.head(5))

# Initialize the Folium map
m = folium.Map(location=[1.5, 20], zoom_start=3, tiles='CartoDB positron')

# Create the Choropleth Map
folium.Choropleth(
    geo_data=geojson_data,
    data=country_data,
    columns=['country', 'fatalities'],
    key_on='feature.properties.name',  # Matches the GeoJSON country name property
    fill_color='YlOrRd',
```

```
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Total Fatalities per Country'
).add_to(m)

#Saving the map as html
```

```
, 'Republic of Congo', 'Sa
go', 'eSwatini', 'Saint He
ssau', 'Mauritius'}
```

Make this Notebook Trusted to load map: File -> Trust Notebook

Geographic Distribution Interpretation

> - **Region - Eastern Africa has the highest number of conflict events, followed by Northern and Western Africa.**
> - **Country - Somalia and Nigeria are the most affected, with ongoing insurgencies and civil unrest.**

Bivariate Analysis

Bivariate Analysis Plan

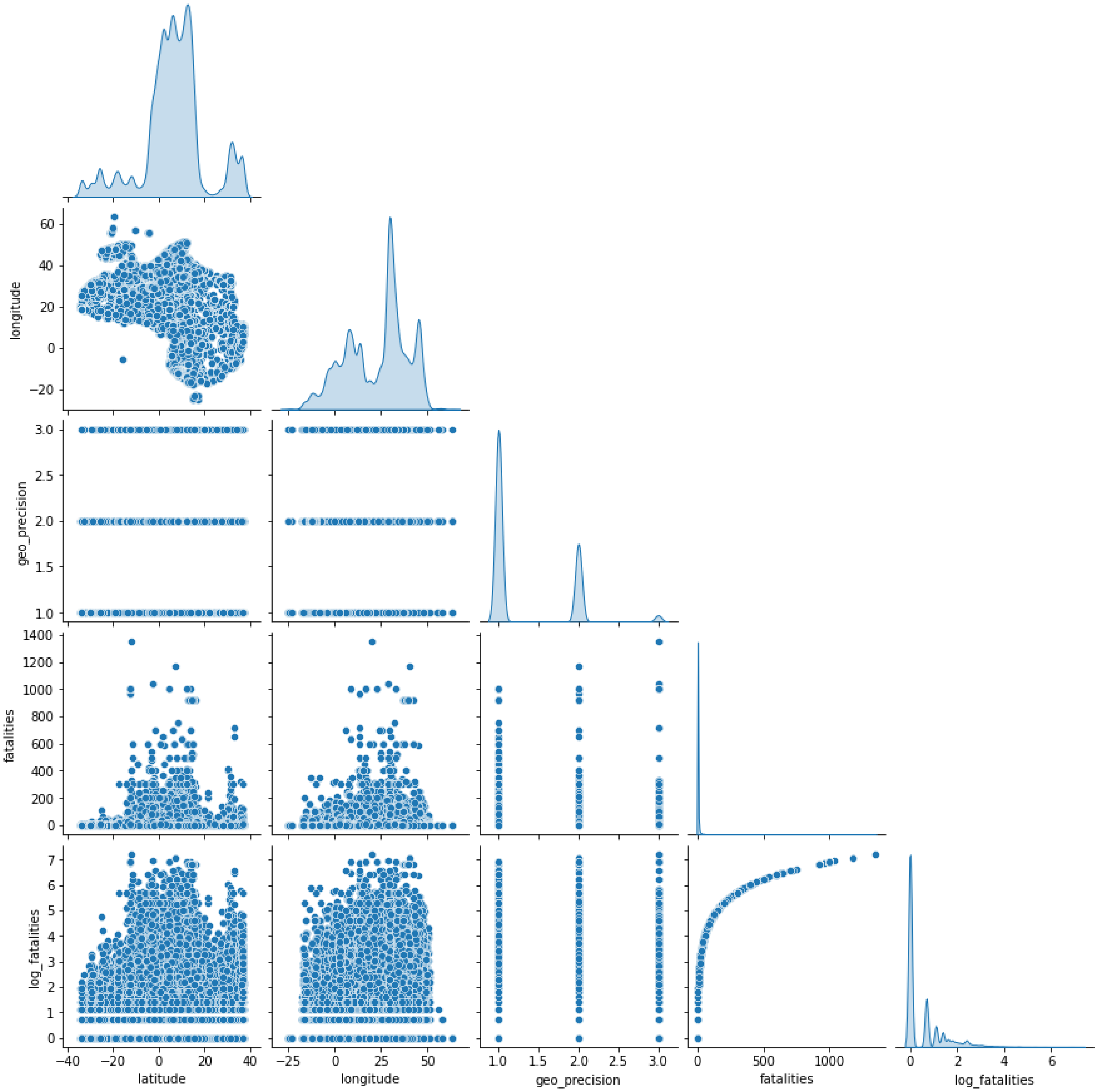
> - **Categorical vs. Categorical Analysis for example Relationship between event type and region, Use stacked bar plots or heatmaps to explore relationships between categorical variables.**
> - **Categorical vs. Numerical Analysis for example Fatalities per region, Use boxplots or violin plots to show distribution differences.**

- Numerical vs. Numerical Analysis for example Correlation between fatalities and event type, Use scatter plots and correlation matrices to check relationships between numerical features.
- Geospatial Bivariate Analysis for example Fatalities per country or region over time, Use geospatial maps to overlay conflicts and fatalities.

a. Numerical vs. Numerical Analysis

In [83]:

```python
# Selecting numerical columns for pairplot
num_cols = ['latitude', 'longitude', 'geo_precision', 'fatalities', 'log_fatalities']

# Pairplot with KDE diagonal and corner=True to avoid duplicate plots
sns.pairplot(df1[num_cols], diag_kind='kde', corner=True)

# Show the plot
plt.show()
```



Interpretation of the Pairplot

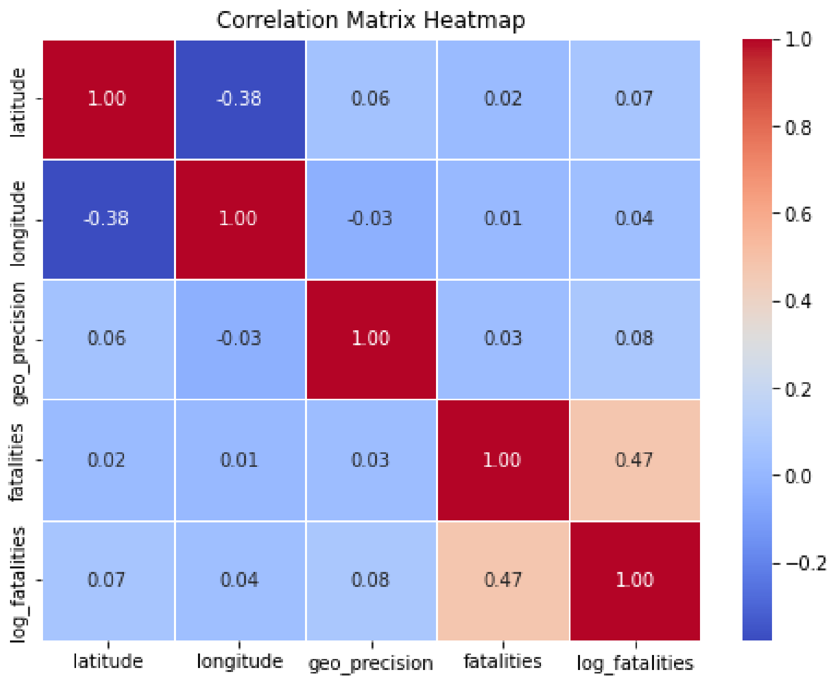
- Geospatial Distribution i.e. Latitude & Longitude, The scatter plots between latitude and longitude suggest distinct clusters, possibly representing regions with frequent conflict longitude suggest distinct clusters, possibly representing regions with frequent conflict events. These clusters may align with high-risk conflict zones in Africa.

- **Geo-Precision vs. Other Variables, geo_precision is categorical (1, 2, or 3), limiting its numerical correlation. hence, It does not show a strong relationship with other variables, indicating location precision varies independently of fatalities or geography.**
- **Fatalities vs. Log-Fatalities, fatalities is highly skewed, with a concentration of lower values and extreme outliers. but, The log transformation i.e. log_fatalities effectively normalizes this distribution, making patterns more discernible.**
- **Fatalities vs. Geographical Variables, There's no strong linear relationship between fatalities and location, though some regions may have higher conflict severity.**

b. Numerical vs. Numerical Analysis

In [84]:

```
# Selecting numerical columns
numerical_cols = ['latitude', 'longitude', 'geo_precision', 'fatalities', 'log_fatalitie
s']
# Compute correlation matrix
corr_matrix = df[numerical_cols].corr()
# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix Heatmap")
plt.show()
```



Numerical vs. Numerical Analysis Interpretation:

- **There are no strong correlations in this dataset, except for the expected relationship between fatalities and log_fatalities. Other variables show minimal influence on each other, suggesting that conflict events are widely distributed across different locations with no dominant numerical pattern.**

Hypothesis Testing

a. Event Type vs. Fatalities (ANOVA)

- **Null Hypothesis (H_0): The average number of fatalities does not significantly differ across different event types.**

- **Alternative Hypothesis (H_1): Some event types are associated with significantly higher or lower fatalities.**

In [85]:

```
# Create a subset for this hypothesis test
df_hypothesis1 = df1[['event_type', 'log_fatalities']]
# Perform ANOVA test
event_groups = [group['log_fatalities'].values for _, group in df_hypothesis1.groupby('e
vent_type')]
anova_stat, p_value = stats.f_oneway(*event_groups)
# Print the results
print(f"ANOVA Test Statistic: {anova_stat:.4f}")
print(f"P-value: {p_value:.4f}")
# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: Fatalities significantly differ across event type
s.")
else:
    print("Fail to reject the null hypothesis: No significant difference in fatalities ac
ross event types.")
```

```
ANOVA Test Statistic: 12066.1573
P-value: 0.0000
Reject the null hypothesis: Fatalities significantly differ across event types.
```

Interpretation for Event Type and Fatalities

> **Since $p < 0.05$, we reject the null hypothesis. This confirms that fatalities significantly vary by event type that is Some event types are far more lethal than others. This insight can help policymakers, security analysts, and businesses assess which types of conflicts pose the greatest risks.**

b. Region vs. Conflict Frequency (Chi-Square Test)

> - **H_0: There is no significant difference in conflict frequency across regions.**
> - **H_1: Certain regions experience significantly more or fewer conflicts than expected.**

In [86]:

```
from scipy.stats import chi2_contingency

# Group by 'region' and 'event_type' to count occurrences of each event type by region
region_event_counts = df1.groupby(['region', 'event_type']).size().unstack(fill_value=0)

# Run Chi-Square test
chi2_stat, p_val, dof, expected = chi2_contingency(region_event_counts)

# # Print results
print(f"Chi-Square Statistic: {chi2_stat:.4f}")
print(f"P-value: {p_val:.4f}")

# Interpret results
if p_val < 0.05:
    print("Reject the null hypothesis: There is a significant difference in conflict freq
uency across regions.")
else:
    print("Fail to reject the null hypothesis: Conflict occurrence is independent of the
region.")
```

```
Chi-Square Statistic: 44930.9787
P-value: 0.0000
Reject the null hypothesis: There is a significant difference in conflict frequency acros
s regions.
```

s regions:

Interpretation of Region vs. Conflict Frequency

> **The chi-square test yielded a test statistic of 44930.9787 with a p-value of 0.0000, Since the p-value is extremely small, we reject the null hypothesis (H_0). This means that conflict occurrence is not evenly distributed across regions—certain regions experience significantly higher or lower conflict frequencies than expected. This finding suggests that some regions are more prone to conflict, while others remain relatively stable.**

c. Interaction Type vs. Fatalities (ANOVA)

> - **H_0: The number of fatalities does not significantly vary based on the type of interactions (e.g., state vs. rebel group).**
> - **H_1: Certain interaction types result in significantly higher or lower fatalities.**

In [87]:

```
# Ensure the dataset contains only relevant columns and no missing values
df_anova = df1[['interaction', 'log_fatalities']]
# Performing ANOVA test
interaction_groups = [group['log_fatalities'].values for _, group in df_anova.groupby('interaction')]
anova_stat, p_value = stats.f_oneway(*interaction_groups)
# Print results
print(f"ANOVA Test Statistic: {anova_stat:.4f}")
print(f"P-value: {p_value:.4f}")
# Interpretation
if p_value < 0.05:
    print("Reject the null hypothesis: Fatalities significantly differ based on interaction type.")
else:
    print("Fail to reject the null hypothesis: No significant difference in fatalities across interaction types.")
```

```
ANOVA Test Statistic: 1129.8259
P-value: 0.0000
Reject the null hypothesis: Fatalities significantly differ based on interaction type.
```

Interpretation of Interaction Type vs. Fatalities

> **The ANOVA test yielded a test statistic of 1129.8259 with a p-value of 0.0000. Since the p-value is extremely small, we reject the null hypothesis (H_0). This means that fatalities are not evenly distributed across interaction types—certain interaction types result in significantly higher or lower fatalities than others. This finding suggests that some types of interactions, such as those involving state forces or rebel groups, may be more lethal, while others result in fewer fatalities.**

DATA PREPARATION

Feature Engineering

> **To develop a conflict occurrence prediction model, we needed a binary target variable indicating whether a conflict took place or not but the dataset does not have a binary target variable column, so we need to create one. We will use event type column**

In [88]:

```
"""
We will analyze the value counts of the event_type column to determine which event types
can
be classified as conflict-related. Based on this assessment, we will create a binary targ
et
variable (conflict_occurred) to differentiate between conflict and non-conflict events,
enabling us to build a predictive model
"""
# Check unique values in event_type
df1['event_type'].value_counts()
```

Out[88]:

```
event_type
Violence against civilians    83530
Battles                       82679
Protests                      35198
Riots                         33988
Strategic developments        32994
Explosions/Remote violence    24617
Name: count, dtype: int64
```

From the event types:

The following cleary indicate conflict since they can be defined as violent actions where the aim is a direct harm to people or infrastructure

> - **Battles**
> - **Violence against civilians**
> - **Explosions/Remote violence**

This are non conflict incidencents they can be defined as non violent actions since the aim was not to direct harm or destroy infrastructure

> - **Riots**
> - **Strategic Developments**
> - **Protests**

Feature Engineer the event type

> **Using the event type and the decision made in classifying if an event is a conflict or a non conflict, we will create a new feature called *conflict_occured* where conflict related events = 1 and non conflict related events = 0**

In [89]:

```
# Define conflict event types
conflict_events = ['Battles', 'Violence against civilians', 'Explosions/Remote violence']
# Create conflict occurrence column
df1['conflict_occurred'] = df1['event_type'].isin(conflict_events).astype(int)
# Check the distribution
df1['conflict_occurred'].value_counts()
```

Out[89]:

```
conflict_occurred
1    190826
0    102180
Name: count, dtype: int64
```

Data preprocessing

Modifying the dataset to use with the most relevant columns that contribute more towards the models to be analysed

In [90]:

```
# Select only relevant columns
selected_columns = ['year', 'latitude', 'longitude', 'fatalities','log_fatalities',
                    'disorder_type', 'event_type', 'sub_event_type',
                    'civilian_targeting', 'region','country', 'conflict_occurred']

df_modified = df1[selected_columns]

# Display the first few rows
df_modified.head()
```

Out[90]:

	year	latitude	longitude	fatalities	log_fatalities	disorder_type	event_type	sub_event_type	civilian_targetin
0	2025	18.9689	2.0041	0	0.0	Political violence	Explosions/Remote violence	Remote explosive/landmine/IED	N
1	2025	11.3693	-0.3600	0	0.0	Political violence	Battles	Armed clash	N
2	2025	18.5558	1.1113	0	0.0	Political violence	Violence against civilians	Attack	Ye
3	2025	6.4667	-2.3333	0	0.0	Political violence	Violence against civilians	Attack	Ye
4	2025	5.5560	-0.1969	0	0.0	Political violence	Riots	Mob violence	Ye

In [91]:

```
# Drop event_type and disorder_type columns
drop_cols = [col for col in df_modified.columns if 'event_type' in col or 'disorder_type
' in col]
df_modified = df_modified.drop(columns=drop_cols)
```

Checking Multicollineality

In [92]:

```
# Select only numerical columns from df_modified
num_col = df_modified.select_dtypes(include=[np.number])

# Ensure num_col is a DataFrame
num_col = num_col.copy()

# Calculate VIF
vif_data = pd.DataFrame()
vif_data["Variable"] = list(num_col)
vif_data["VIF"] = [variance_inflation_factor(num_col.values, i) for i in range(num_col.s
hape[1])]
vif_data = vif_data.sort_values(by='VIF', ascending=False)

print(vif_data)
```

```
           Variable       VIF
0              year  5.230983
2         longitude  3.521999
5  conflict_occurred  3.495478
4    log_fatalities  2.139522
1          latitude  1.432180
3        fatalities  1.331278
```

In [93]:

```
# spliting data according to the data type
```

```
# splitting data according to the data type
## assign data types variable names
num_col = df_modified.select_dtypes(include=['int64', 'float64']).columns
cat_col = df_modified.select_dtypes(include=['object', 'bool']).columns
# display the columns
display(num_col)
display(cat_col)
```

```
Index(['year', 'latitude', 'longitude', 'fatalities', 'log_fatalities',
       'conflict_occurred'],
      dtype='object')

Index(['civilian_targeting', 'region', 'country'], dtype='object')
```

Perform one hot enconding to our cateagorical variables

In [94]:

```
# Select categorical columns
cat_col = df_modified.select_dtypes(include=["object", "bool"]).columns

# One-hot encode categorical variables & convert boolean to int
df_modified = pd.get_dummies(df_modified, columns=cat_col, drop_first=True).astype(int)

# Rename columns to remove duplicate "_True"
df_modified.columns = df_modified.columns.str.replace(r'_True_True$', '_True', regex=Tru
e)

# Preview the cleaned dataframe
df_modified.head()
```

Out[94]:

	year	latitude	longitude	fatalities	log_fatalities	conflict_occurred	civilian_targeting_Yes	region_Middle Africa	region_Northern Africa	r
0	2025	18	2	0	0	1	0	0	0	
1	2025	11	0	0	0	1	0	0	0	
2	2025	18	1	0	0	1	1	0	0	
3	2025	6	-2	0	0	1	1	0	0	
4	2025	5	0	0	0	0	1	0	0	

5 rows × 67 columns

MODELLING

In [95]:

```
## Defining our X and y
X = df_modified.drop('conflict_occurred', axis=1)
y = df_modified['conflict_occurred']
## Splitting the data into train sets and test sets using 20% and random_state of 42
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2, random_state=42, st
ratify=y)
```

In [96]:

```
# Check the shape of the splits
print(f"X_train shape: {X_train.shape}, X_test shape: {X_test.shape}")
print(f"y_train distribution:\n{y_train.value_counts(normalize=True)}")
print(f"y_test distribution:\n{y_test.value_counts(normalize=True)}")
```

```
X_train shape: (234404, 66), X_test shape: (58602, 66)
y_train distribution:
conflict_occurred
```

```
1    0.651269
0    0.348731
Name: proportion, dtype: float64
y_test distribution:
conflict_occurred
1    0.651275
0    0.348725
Name: proportion, dtype: float64
```

Intergrating pipeline to the models

In [97]:

```
# Logistic Regression Pipeline
log_reg_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(random_state=42))
])

# Random Forest Pipeline
rf_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1))
])

# XGBoost Pipeline
xgb_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', xgb.XGBClassifier(eval_metric='logloss', random_state=42))
])
```

Training and making predictions of the model

In [98]:

```
# Train the models
log_reg_pipeline.fit(X_train, y_train)
rf_pipeline.fit(X_train, y_train)
xgb_pipeline.fit(X_train, y_train)

# Make predictions
y_pred_log = log_reg_pipeline.predict(X_test)
y_pred_rf = rf_pipeline.predict(X_test)
y_pred_xgb = xgb_pipeline.predict(X_test)
```

Evaluating the models

In [99]:

```
# Compute accuracy for each model
accuracy_log = accuracy_score(y_test, y_pred_log)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)

# Print results
print(f"Logistic Regression Accuracy: {accuracy_log:.4f}\n")
print(f"Random Forest Accuracy: {accuracy_rf:.4f}\n")
print(f"XGBoost Accuracy: {accuracy_xgb:.4f}")

# Detailed performance
print("\nLogistic Regression Performance:\n", classification_report(y_test, y_pred_log))
print("\nRandom Forest Performance:\n", classification_report(y_test, y_pred_rf))
print("\nXGBoost Performance:\n", classification_report(y_test, y_pred_xgb))
```

```
Logistic Regression Accuracy: 0.8070

Random Forest Accuracy: 0.8567
```

```
XGBoost Accuracy: 0.8619

Logistic Regression Performance:
              precision    recall  f1-score   support

           0       0.73      0.72      0.72     20436
           1       0.85      0.86      0.85     38166

    accuracy                           0.81     58602
   macro avg       0.79      0.79      0.79     58602
weighted avg       0.81      0.81      0.81     58602


Random Forest Performance:
              precision    recall  f1-score   support

           0       0.80      0.79      0.79     20436
           1       0.89      0.89      0.89     38166

    accuracy                           0.86     58602
   macro avg       0.84      0.84      0.84     58602
weighted avg       0.86      0.86      0.86     58602


XGBoost Performance:
              precision    recall  f1-score   support

           0       0.81      0.79      0.80     20436
           1       0.89      0.90      0.89     38166

    accuracy                           0.86     58602
   macro avg       0.85      0.85      0.85     58602
weighted avg       0.86      0.86      0.86     58602
```

Before hyperparameters interpretations

> - **Logistic Regression model achieved an accuracy of 80.71%, with a precision of 0.73 for class 0 and 0.85 for class 1. The recall values were 0.72 and 0.86, respectively, meaning the model slightly favored predicting class 1 (conflict) over class 0 (no conflict). The overall F1-scores were fairly consistent with precision and recall, indicating a fairly balanced performance.**
> - **Random Forest model achieved an accuracy of 85.67%, with a precision of 0.80 for class 0 and 0.89 for class 1. The recall values indicate that the model correctly identified 79% of class 0 instances and 89% of class 1 instances. The overall F1-score was 0.79 for class 0 and 0.89 for class 1, showing that the model was already performing well, especially in predicting class 1 events.**
> - **XGBoost model achieved an accuracy of 0.86.19, with a precision of 0.81 for class 0 and 0.89 for class 1. The recall values were 0.79 for class 0 and 0.90 for class 1, indicating that the model was slightly better at capturing positive cases than negative ones. The F1-scores, which balance precision and recall, were 0.80 for class 0 and 0.89 for class 1, contributing to an overall strong performance. However, there was still room for improvement, particularly in recall for class 0.**

Hyperparameter tunning for Logistic Regression

In [100]:

```
param_grid_logreg = {
    'classifier__C': [0.01, 0.1, 1, 10, 100],
    'classifier__penalty': ['l1', 'l2'],
    'classifier__solver': ['liblinear']
}

grid_logreg = GridSearchCV(log_reg_pipeline, param_grid_logreg, scoring='f1', cv=3, n_jo
bs=-1)
```

```
grid_logreg.fit(X_train, y_train)

# Get best parameters
best_params_logreg = grid_logreg.best_params_
print("Best Hyperparameters for Logistic Regression:", best_params_logreg)
```

```
Best Hyperparameters for Logistic Regression: {'classifier__C': 10, 'classifier__penalty'
: 'l2', 'classifier__solver': 'liblinear'}
```

Retrain the logistic regression with the best hyperparameters

In [101]:

```
# Update the pipeline with the best hyperparameters
best_log_reg_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(C=1, penalty='l1', solver='liblinear', random_stat
e=42))
])

# Train the optimized model
best_log_reg_pipeline.fit(X_train, y_train)
# Make predictions
y_pred_best_log = best_log_reg_pipeline.predict(X_test)
# Evaluate the optimized model
accuracy = accuracy_score(y_test, y_pred_best_log)

print("\nOptimized Logistic Regression Performance:\n")
print(f"Accuracy: {accuracy:.4f}")
print(classification_report(y_test, y_pred_best_log))
```

```
Optimized Logistic Regression Performance:

Accuracy: 0.8071
              precision    recall  f1-score   support

           0       0.73      0.72      0.72     20436
           1       0.85      0.86      0.85     38166

    accuracy                           0.81     58602
   macro avg       0.79      0.79      0.79     58602
weighted avg       0.81      0.81      0.81     58602
```

After Hyperparameters interpretation

> **After tuning, the accuracy remained 80.71%, showing that the optimized parameters did not significantly impact overall performance. Recall, Precision and F1-scores remained nearly unchanged. This suggests that the default hyperparameters were already effective, and further tuning may not yield significant improvements.**

Hyperparameter tunning for Random forest classifier

In [102]:

```
param_grid_rf = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [10, 20],
    'classifier__min_samples_split': [2, 5],
    'classifier__min_samples_leaf': [1, 2]
}

grid_rf = GridSearchCV(rf_pipeline, param_grid_rf, scoring='f1', cv=3, n_jobs=-1)
grid_rf.fit(X_train, y_train)

# Get best parameters
```

```
best_params_rf = grid_rf.best_params_
print("Best Hyperparameters for Random Forest:", best_params_rf)
```

```
Best Hyperparameters for Random Forest: {'classifier__max_depth': 20, 'classifier__min_sa
mples_leaf': 2, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 200}
```

Retrain random forest model using the best hyperparameters

In [103]:

```
# Update the pipeline with the best hyperparameters
best_rf_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier(
        n_estimators=200,
        max_depth=20,
        min_samples_leaf=2,
        min_samples_split=2,
        random_state=42,
        n_jobs=-1
    ))
])

# Train the optimized model
best_rf_pipeline.fit(X_train, y_train)
# Make predictions
y_pred_best_rf = best_rf_pipeline.predict(X_test)
# Calculate accuracy
rf_accuracy = accuracy_score(y_test, y_pred_best_rf)
print(f"\nOptimized Random Forest Accuracy: {rf_accuracy:.4f}")
# Print classification report
print("\nOptimized Random Forest Performance:\n")
print(classification_report(y_test, y_pred_best_rf))
```

```
Optimized Random Forest Accuracy: 0.8604

Optimized Random Forest Performance:

              precision    recall  f1-score   support

           0       0.80      0.80      0.80     20436
           1       0.89      0.89      0.89     38166

    accuracy                           0.86     58602
   macro avg       0.85      0.85      0.85     58602
weighted avg       0.86      0.86      0.86     58602
```

After Hyperparameters interpretation

> **The hyperparameter tuning appears to have led to a small improvement in performance for Class 0, particularly in recall and F1-score, but no change for Class 1. This suggests that the optimized model is slightly better at identifying Class 0 instances without affecting its performance on Class 1, which was already good. This small improvement in accuracy (from 85.67% to 86.04%) suggests that hyperparameter tuning helped the model generalize a little better, possibly by improving its ability to correctly classify both the positive and negative classes, even though the improvements in individual metrics (precision, recall, F1-score) were more noticeable for Class 0.**

Hyperparameter tunning for XGboost

In [104]:

```
# Define parameter grid
param_grid = {
    'n_estimators': [100, 200],
```

```
    'max_depth': [3, 6],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Initialize XGBoost
xgb_clf = xgb.XGBClassifier(eval_metric='logloss', random_state=42)

# Randomized Search for best hyperparameters
random_search = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_grid,
    scoring='f1',
    n_iter=20,
    cv=5,
    verbose=2,
    n_jobs=-1,
    random_state=42
)

# Fit model on training data
random_search.fit(X_train, y_train)

# Get best parameters
best_params = random_search.best_params_
print("Best Hyperparameters:", best_params)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100, 'max_depth': 6, 'learning_r
ate': 0.2, 'colsample_bytree': 1.0}
```

Retrain XGBoost using the best parameters

In [105]:

```
# Retrain XGBoost with best hyperparameters
best_xgb = xgb.XGBClassifier(
    subsample=1.0,
    n_estimators=100,
    max_depth=6,
    learning_rate=0.2,
    colsample_bytree=1.0,
    eval_metric='logloss',
    random_state=42
)

# Train the optimized model
best_xgb.fit(X_train, y_train)

# Make predictions
y_pred_best_xgb = best_xgb.predict(X_test)

# Evaluate performance
from sklearn.metrics import accuracy_score, classification_report

print(f"\nOptimized XGBoost Accuracy: {accuracy_score(y_test, y_pred_best_xgb):.4f}")
print("\nOptimized XGBoost Performance:\n")
print(classification_report(y_test, y_pred_best_xgb))
```

```
Optimized XGBoost Accuracy: 0.8598

Optimized XGBoost Performance:

              precision    recall  f1-score   support

           0       0.80      0.79      0.80     20436
           1       0.89      0.90      0.89     38166

    accuracy                           0.86     58602
   macro avg       0.85      0.84      0.85     58602
```

```
weighted avg       0.86      0.86      0.86     58602
```

After Hyperparameters interpretation

> XGBoost model achieved an accuracy of 0.86, maintaining strong predictive performance. The precision for both classes remained stable, while recall for class 0 stayed at 0.79, ensuring a balanced detection of conflict and non-conflict events. The macro and weighted averages remained consistent, indicating that the model retained its ability to generalize well across different cases. These results suggest that hyperparameter tuning fine-tuned the model without drastically altering its effectiveness.

Interpretation of Results

Logistic Regression

Accuracy: 80.71%

> - Class 0 Recall: 72%
> - Class 1 Recall: 86% Logistic Regression performs reasonably well, but it has a lower recall for class 0, meaning it struggles more with identifying areas where conflict will not occur.

Random Forest

Accuracy: 86%

> - Class 0 Recall: 80%
> - Class 1 Recall: 89% Random Forest improves both recall and precision over Logistic Regression, making it a more balanced choice for conflict prediction.

XGBoost

Accuracy: 86%

> - Class 0 Recall: 79%
> - Class 1 Recall: 90% XGBoost has the highest accuracy and F1-score, meaning it performs best overall in predicting both conflict and no-conflict events. It maintains a strong balance between precision and recall.

Best Model Recommendation

XGBoost is the best model to choose because:

> - Highest accuracy of 86% – Outperforms the other models.
> - Best recall for conflict cases of 90% – This will ensures high detection of conflict-prone areas.
> - Good balance between precision and recall – Minimizes false positives and false negatives.