Global Terrorism Database (GTD)

A Multi-Class Classification Approach

By: Group 1

Members: Akansha, Anand, Aseem, Vedanshi, Shehzad khan, Sanjeet

Course: Data Science

Institution: Great Learning

Date: 07 August, 2025

Abstract

The growing threat of global terrorism poses significant challenges to national security and counter-terrorism agencies worldwide. One of the critical issues in post-attack investigations is identifying the terrorist organization responsible, as many groups do not claim responsibility. This uncertainty delays countermeasures, prolongs investigations, and wastes valuable resources. To address this challenge, this project leverages machine learning to predict the responsible terrorist group based on historical attack data from the Global Terrorism Database, which includes over **2,14,666 incidents** and **135 features** such as location, attack type, target type, and weapon used. We implemented multiple classification models, including Logistic Regression, Decision Tree and Random Forest, and evaluated them using accuracy ,recall and precision . Among the tested models, Random Forest delivered the best performance with an accuracy of 83% and recall of 83%, demonstrating the potential of predictive analytics in accelerating counter-terrorism decision-making. These findings highlight how data-driven insights can support law enforcement and intelligence agencies in responding swiftly and effectively to global security threats.

Table of Content

Introduction

Terrorism remains a persistent and evolving global threat, causing significant loss of life, disruption, and fear across societies. The growing complexity and frequency of terrorist incidents have prompted the need for advanced analytical tools to assist law enforcement and intelligence agencies. One major challenge is the frequent lack of accountability, as responsible groups often do not claim their attacks, complicating investigative and response efforts. This project leverages the Global Terrorism Database (GTD), a comprehensive open-source repository containing detailed records of over 2,14,666 terrorist incidents worldwide from 1970 to 2021, with 135 features covering aspects like date, location, weapon type, attack method, and target profile. The primary objective is to develop and evaluate machine learning models that can accurately predict the group responsible for an attack, thus potentially reducing investigation time and aiding timely decision-making. Several models—such as Logistic Regression, Decision Tree and Random Forest, —were tested and benchmarked using accuracy, precision and recall, with the final model achieving an accuracy 83%. These results demonstrate the promise of data-driven approaches in supporting counter-terrorism and public safety strategies.

Problem Statement

Many terrorist incidents remain unattributed because perpetrator groups often do not claim responsibility, resulting in delayed countermeasures and inefficient use of investigative resources. This project seeks to leverage structured historical data from the Global Terrorism Database (GTD), which includes detailed characteristics of attacks such as location, weapon type, attack type, and target, to build a robust machine learning model that predicts the likely responsible group. Accurate prediction of perpetrator groups can help authorities narrow investigative leads, optimize resource allocation, and deploy preventive strategies more effectively. Key challenges include managing class imbalance, noisy or incomplete data, and overlapping features among multiple groups. Model success will be evaluated using classification accuracy, precision and recall across different terrorist groups. This approach aims to demonstrate the feasibility of data-driven attribution to support faster, data-informed counter-terrorism efforts and improve public safety outcomes.

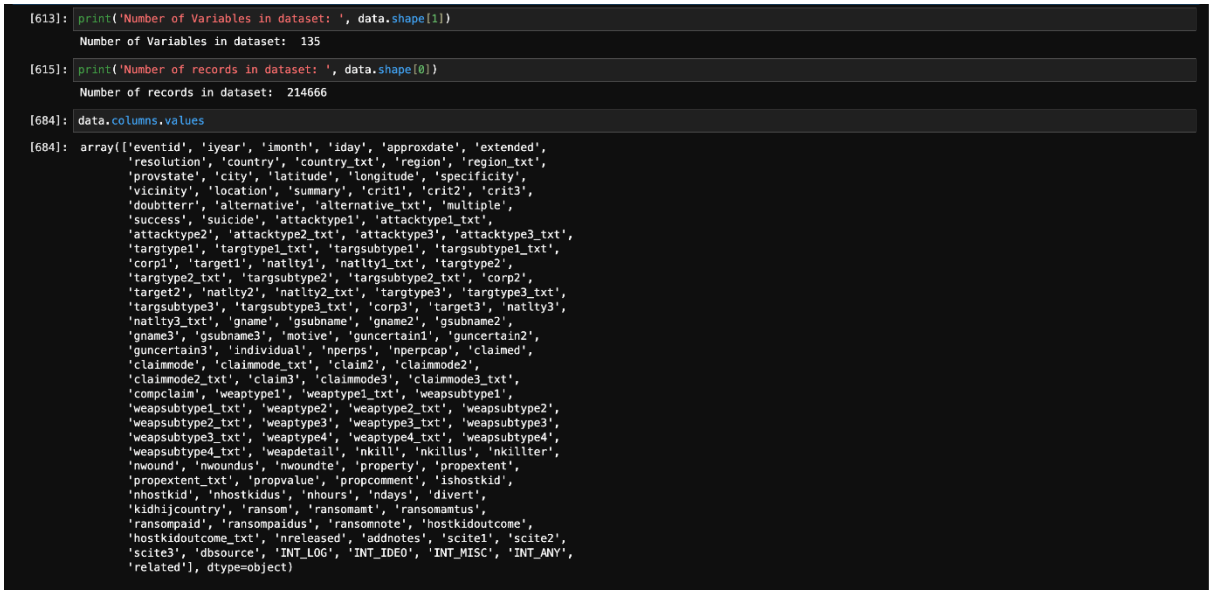
Dataset Description

Source

The Global Terrorism Database™ (GTD) is an open-source database including information on terrorist events around the world from 1970 through 2020. Unlike many other event databases, the GTD includes systematic data on domestic as well as transnational and international terrorist incidents that have occurred during this time period and now includes more than 200,000 cases. For each GTD incident, information is available on the date and location of the incident, the weapons used and nature of the target, the number of casualties, and--when identifiable--the group or individual responsible.
https:/www.start.umd.edu/data-tools/GTD

Detail

The dataset has 135 columns and 2,14,666 rows.



```
[613]: print('Number of Variables in dataset: ', data.shape[1])
       Number of Variables in dataset:  135
[615]: print('Number of records in dataset: ', data.shape[0])
       Number of records in dataset:  214666
[684]: data.columns.values
[684]: array(['eventid', 'iyear', 'imonth', 'iday', 'approxdate', 'extended',
       'resolution', 'country', 'country_txt', 'region', 'region_txt',
       'provstate', 'city', 'latitude', 'longitude', 'specificity',
       'vicinity', 'location', 'summary', 'crit1', 'crit2', 'crit3',
       'doubtterr', 'alternative', 'alternative_txt', 'multiple',
       'success', 'suicide', 'attacktype1', 'attacktype1_txt',
       'attacktype2', 'attacktype2_txt', 'attacktype3', 'attacktype3_txt',
       'targtype1', 'targtype1_txt', 'targsubtype1', 'targsubtype1_txt',
       'corp1', 'target1', 'natlty1', 'natlty1_txt', 'targtype2',
       'targtype2_txt', 'targsubtype2', 'targsubtype2_txt', 'corp2',
       'target2', 'natlty2', 'natlty2_txt', 'targtype3', 'targtype3_txt',
       'targsubtype3', 'targsubtype3_txt', 'corp3', 'target3', 'natlty3',
       'natlty3_txt', 'gname', 'gsubname', 'gname2', 'gsubname2',
       'gname3', 'gsubname3', 'motive', 'guncertain1', 'guncertain2',
       'guncertain3', 'individual', 'nperps', 'nperpcap', 'claimed',
       'claimmode', 'claimmode_txt', 'claim2', 'claimmode2',
       'claimmode2_txt', 'claim3', 'claimmode3', 'claimmode3_txt',
       'compclaim', 'weaptype1', 'weaptype1_txt', 'weapsubtype1',
       'weapsubtype1_txt', 'weaptype2', 'weaptype2_txt', 'weapsubtype2',
       'weapsubtype2_txt', 'weaptype3', 'weaptype3_txt', 'weapsubtype3',
       'weapsubtype3_txt', 'weaptype4', 'weaptype4_txt', 'weapsubtype4',
       'weapsubtype4_txt', 'weapdetail', 'nkill', 'nkillus', 'nkillter',
       'nwound', 'nwoundus', 'nwoundte', 'property', 'propextent',
       'propextent_txt', 'propvalue', 'propcomment', 'ishostkid',
       'nhostkid', 'nhostkidus', 'nhours', 'ndays', 'divert',
       'kidhijcountry', 'ransom', 'ransomamt', 'ransomamtus',
       'ransompaid', 'ransompaidus', 'ransomnote', 'hostkidoutcome',
       'hostkidoutcome_txt', 'nreleased', 'addnotes', 'scite1', 'scite2',
       'scite3', 'dbsource', 'INT_LOG', 'INT_IDEO', 'INT_MISC', 'INT_ANY',
       'related'], dtype=object)
```

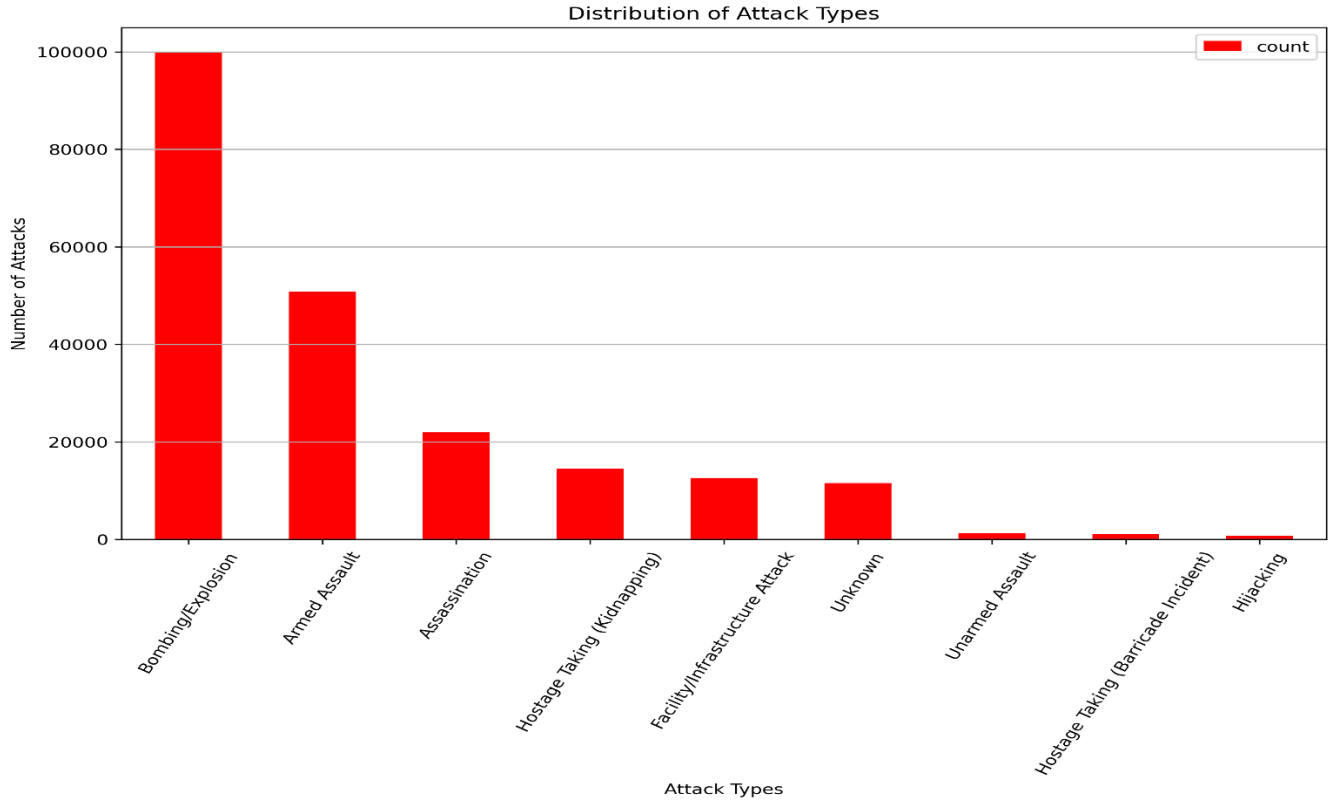
Features and Target variable

- **success**: Denotes if the attack was successful according to defined criteria per attack type.
- **suicide**: Indicates if the attack was a suicide attack (1=Yes, 0=No).
- **attacktype1, attacktype1_txt**: Primary method of attack (e.g., Bombing/Explosion, Armed Assault).
- **targtype1, targtype1_txt**: General type of the first target/victim (e.g., Government, Military, Private Citizens).

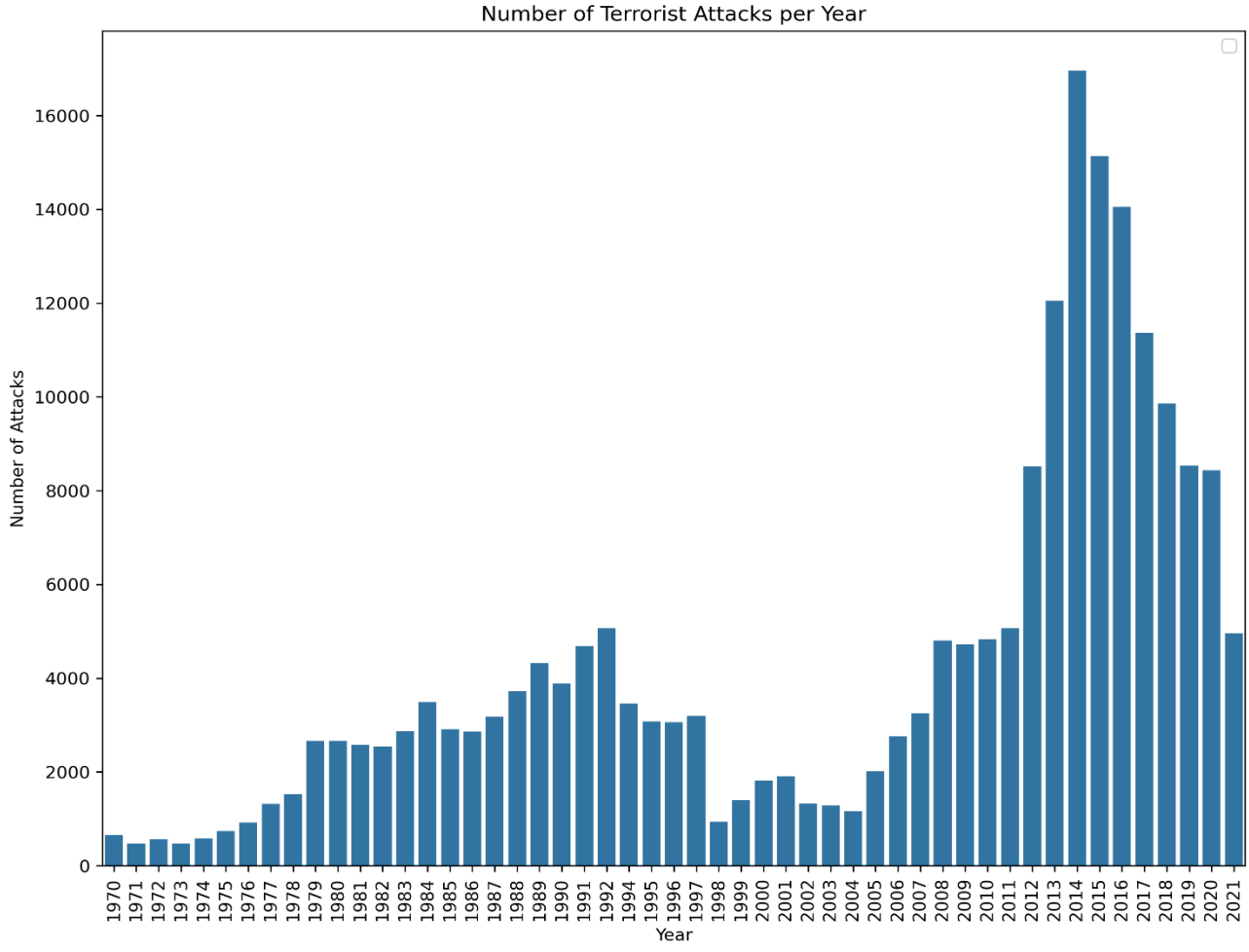
- **targsubtype1, targsubtype1_txt**: More specific categorization of target subtype.
- **natlty1, natlty1_txt**: Nationality of the first target.
- **gname**: Name of the perpetrator group responsible for the incident.
- **gsubname**: Sub-group or faction of the perpetrator group.
- **nperps**: Number of perpetrators involved.
- **weaptype1, weaptype1_txt**: Primary weapon type used (e.g., Explosives, Firearms).
- **weapsubtype1, weapsubtype1_txt**: Specific subtype of the first weapon type (e.g., Grenade, Handgun).
- **nkillus**: Number of US citizens killed.
- **nkillter**: Number of perpetrators killed.

Data Exploration

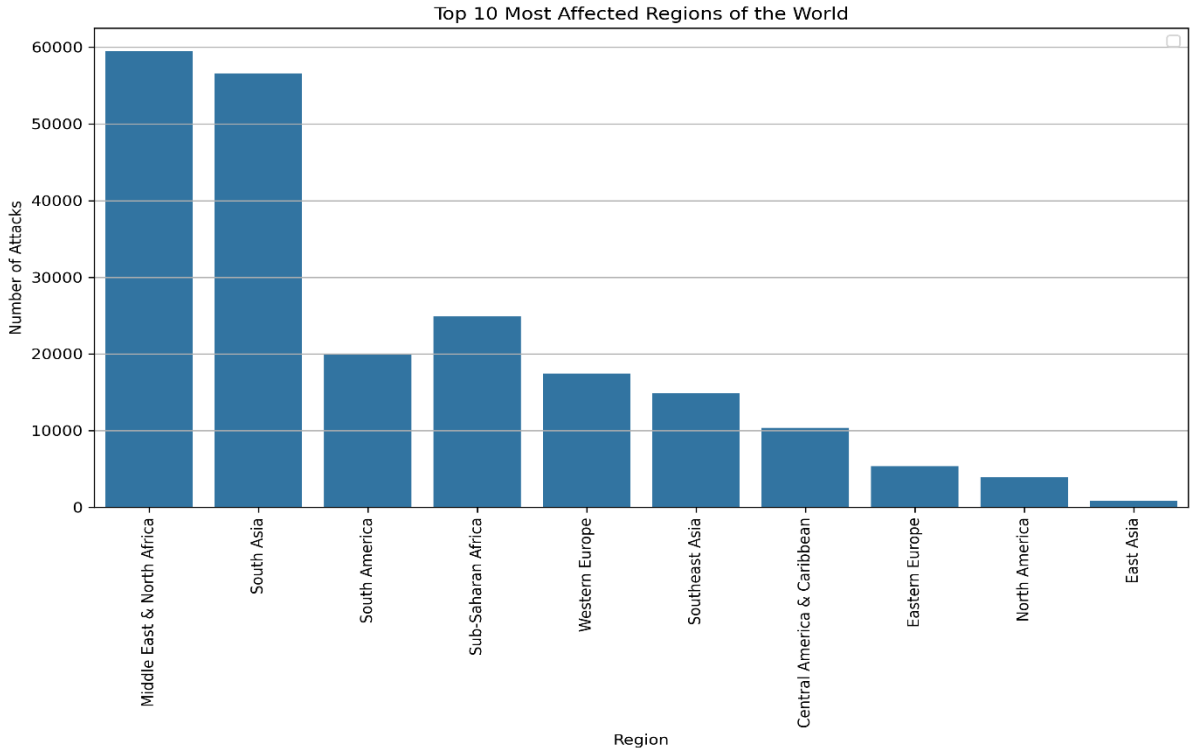
Type of attacks and their frequency.



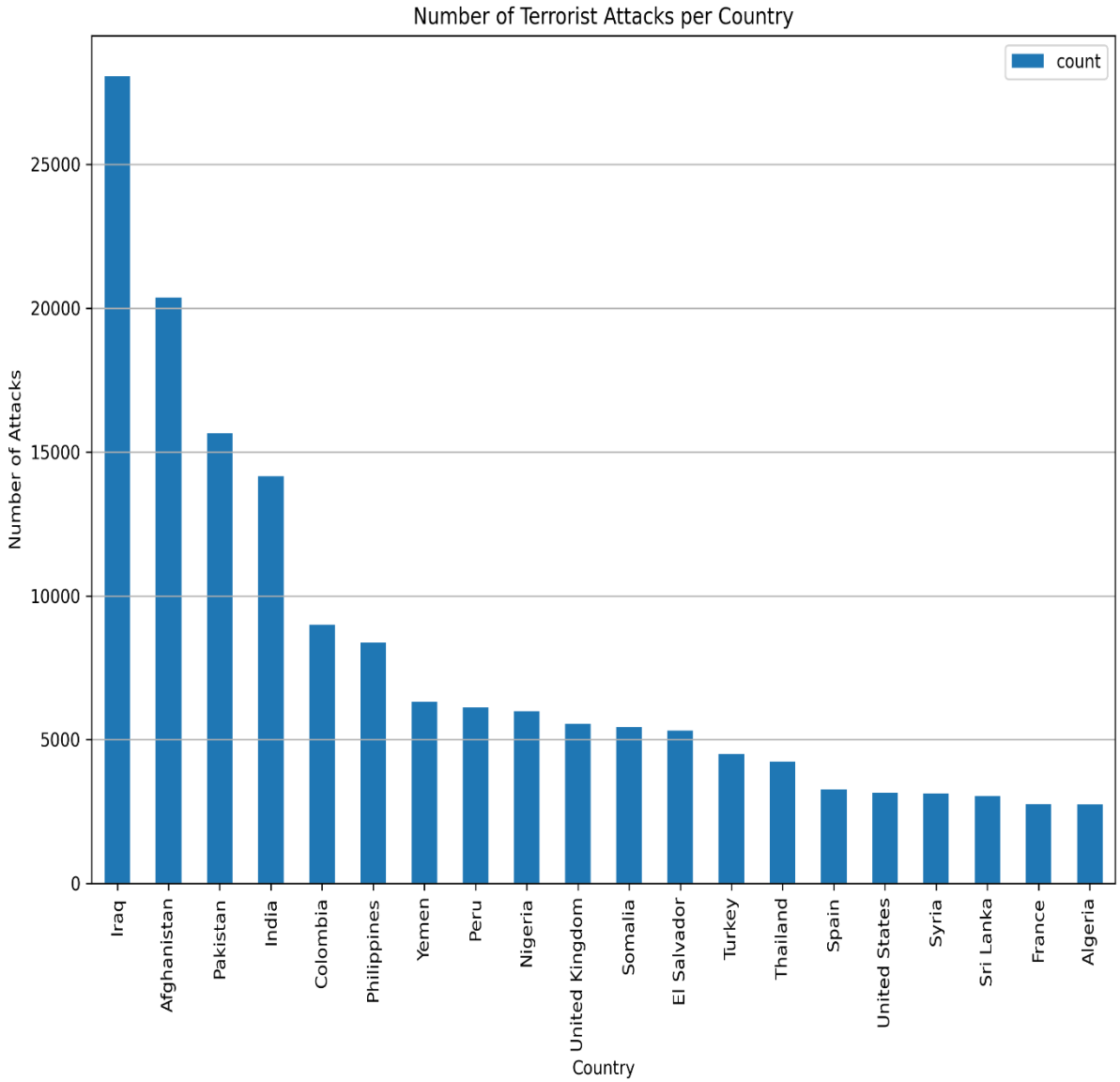
Count of attacks by year.



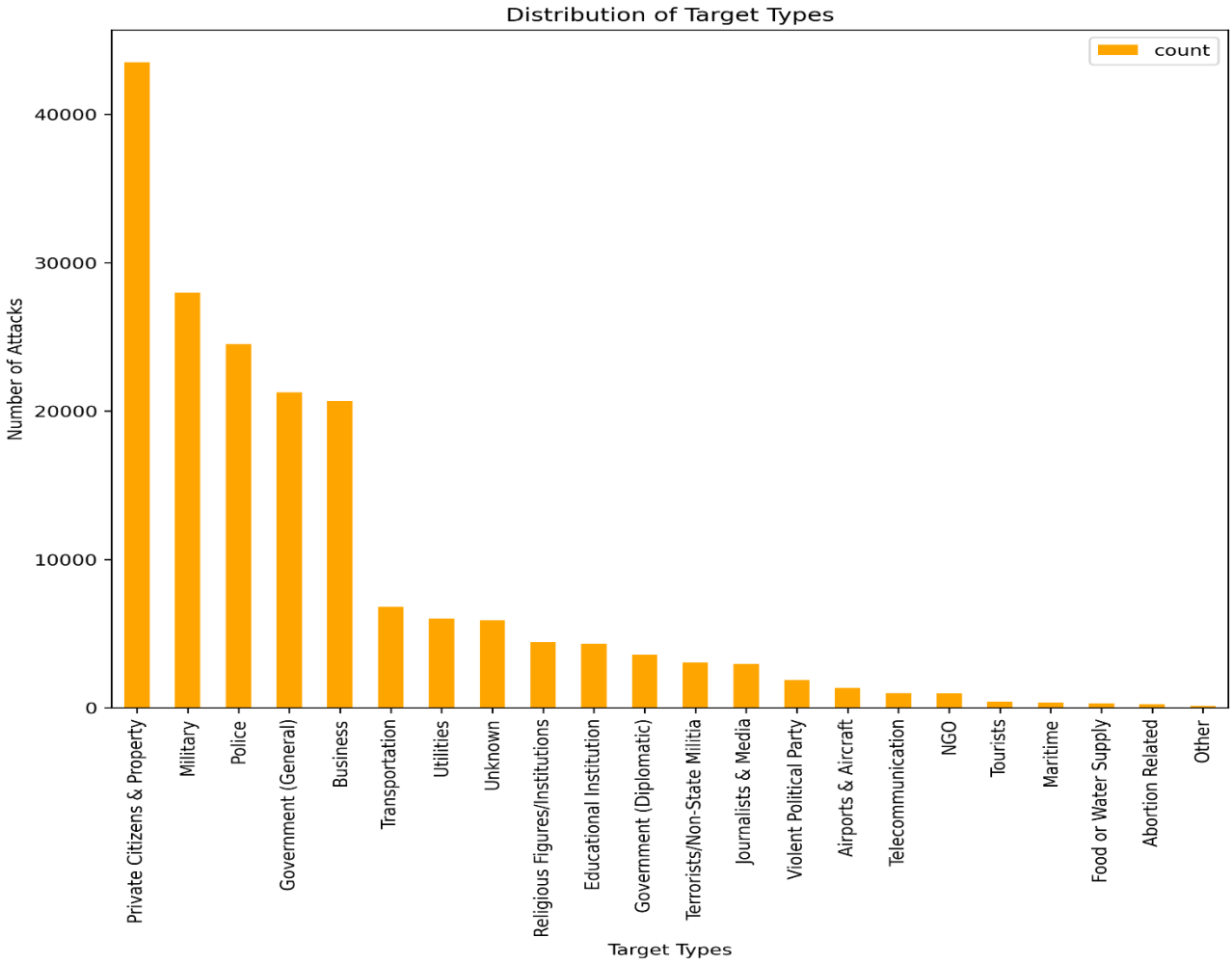
Region wise count of attacks.



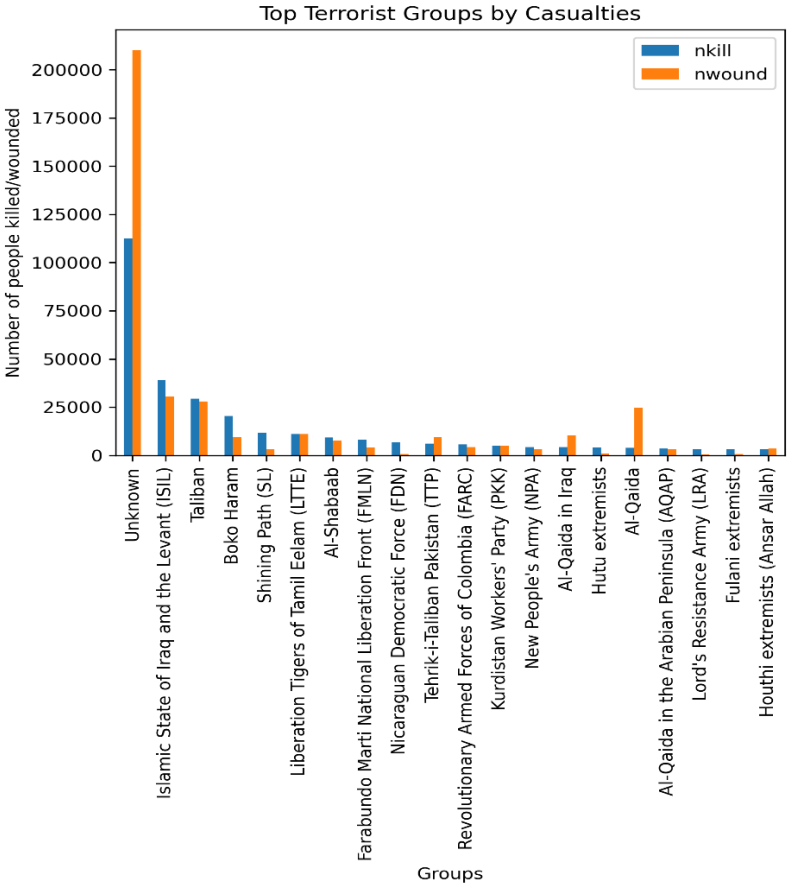
Count of attack in top 20 countries from 1970 to 2021.



Target of terrorist groups.



Top terrorist groups in list and casualty affected by them.



Attacks with label unknown are, for which no group has claimed responsibility, is used as test data for predictive modelling.

Exploratory Data Analysis

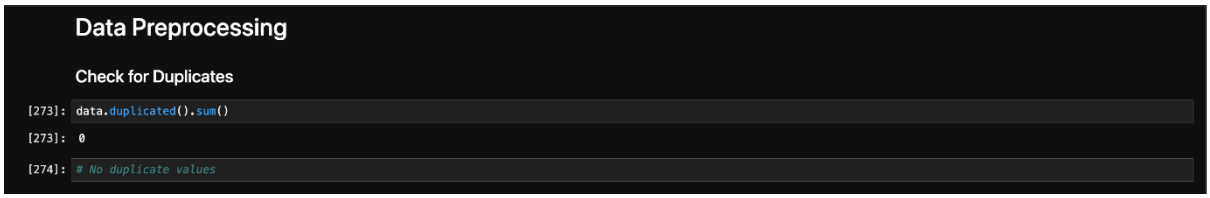
Data Overview

Source: https://www.start.umd.edu/gtd-download

Data Shape: 135 columns and 214666 rows.

Target Variable: gname

Step 1: Checking for duplicated Values



No duplicates Found.

Step 2: Check for Null Values

Observation: 104 variables contain NULL VALUES.

NULL VALUE TREATMENT

We dropped 91 columns with over 5% missing values due to high Null Value presence.
The remaining 41 columns, each with over 95% complete data, were retained for analysis.

Columns Left: 135-94 = 41

Step 3: Checking for Correlation between variables

Multicollinearity occurs when two or more independent variables are highly correlated, leading to a situation where one variable can be predicted from the others with high accuracy. In models like linear regression, this can result in:

- Inflated standard errors
- Unstable and unreliable coefficient estimates
- Difficulty in determining the effect of individual predictors

Treating multicollinearity ensures that the model remains statistically sound, interpretable, and avoids misleading inferences. Common solutions include dropping correlated variables, principal component analysis (PCA), or using regularization techniques like Ridge or Lasso regression.

```
high_corr = corr_matrix.where((corr_matrix > 0.5) & (corr_matrix < 1.0))

high_corr_pairs = high_corr.stack().reset_index()
high_corr_pairs.columns = ['Feature_1', 'Feature_2', 'Correlation']
high_corr_pairs = high_corr_pairs.sort_values(by='Correlation', ascending=False)

print('Highly correlated feature pairs (0.5 < corr < 1.0)')
print(high_corr_pairs)
```

```
Highly correlated feature pairs (0.5 < corr < 1.0)
      Feature_1    Feature_2  Correlation
0       eventid        iyear     0.999996
2         iyear      eventid     0.999996
10      INT_LOG     INT_IDEO     0.996426
12     INT_IDEO      INT_LOG     0.996426
13     INT_IDEO      INT_ANY     0.899411
15      INT_ANY     INT_IDEO     0.899411
11      INT_LOG      INT_ANY     0.896726
14      INT_ANY      INT_LOG     0.896726
7   attacktype1    weaptype1     0.698660
9     weaptype1  attacktype1     0.698660
4       country      natlty1     0.634821
8       natlty1      country     0.634821
1       eventid    longitude     0.524264
5     longitude      eventid     0.524264
3         iyear    longitude     0.524248
6     longitude        iyear     0.524248
```

Observation: 10 variables have 16 relations where correlation.

MULTICOLLINIARITY TREATMENT

Dropping these 10 columns.

Columns Left: 41-10 = 31

Step 4: Dropping Irrelevant Columns.

Observation: Upon further exploration, the dbsource column has been identified as irrelevant to the modelling process, as it exclusively indicates the database origin. Therefore, we recommend its exclusion from the dataset

TREATMENT

Dropping the column: dbsource.

Columns Left: 31-1 = 30

Step 5: Analysing Target Variable

Observation: We have 3537 classes in target variable. Since, this is huge number to predict we will take groups that have carries out attacks greater than the total average of all terrorist groups.

```
gname_breakdown = data_cleaned.groupby('gname')[['gname']].count().rename(columns={'gname': 'count'})
gname_total = gname_breakdown.index
gname_above_avg = gname_breakdown[gname_breakdown['count'] > gname_breakdown['count'].mean() ]

print('Total Groups: ', len(gname_total))
print('Count of groups responsible for more than the Average attacks: ', len(gname_above_avg))
```

```
Total Groups:  3767
Count of groups responsible for more than the Average attacks:  218
```

Total number of groups that have conducted attacks greater than average is 218.

TREATMENT

Dropping the rows with less Important/ less dangerous/ small groups.

```
print('Total groups in new dataset: ', len(gname_total))
print('\n', 'Total number of records after processing: ', len(data_cleaned))
```

```
Total groups in new dataset:  218

 Total number of records after processing:  198005
```

Rows Left: 214666 -16661= 198005

Step 6: Splitting Train and Test Data

Observation: Out target variable has one class as UNKNOWN, which is of no use in predictive modelling so we convert it to as TEST DATA.

```
# Remove records where gname is 'Unknown'
# Cut data_cleaned
g_list = gname_breakdown.index.to_list()
g_list.remove('Unknown')
```

```
[82]:
data_cleaned = data_cleaned[data_cleaned['gname'].isin(g_list)]
print("Total number of records after removing 'unknown' groups: ", len(data_cleaned))
```

```
Total number of records after removing 'unknown' groups:  104330
```

TRAIN TEST SPLIT

Train=data_cleaned [data_cleaned['gname']! = 'Unknown']

Test = data_cleaned [data_cleaned['gname'] =='Unknown']

Rows in Train = 104330

Rows in Test =93675

Step 7: Handling Categorical Variables

Observation: Out of 30 columns 10 are categorical. Which have different distinct values.

Handle Categorical Variables

```
[298]: # view levels of each categorical variable
           # Only those with levels <30 will be included (ensures python doesn't crash in OneHotEncoding)

       cat_vars = data_cleaned.loc[:, data_cleaned.dtypes == object]
       cat_vars_count = cat_vars.nunique()
       cat_vars_count
```

```
[298]: country_txt          144
       region_txt            12
       provstate           1436
       city               21912
       attacktype1_txt        9
       targtype1_txt         22
       target1            38862
       natlty1_txt          169
       gname                203
       weaptype1_txt         11
       dtype: int64
```

```
[371]: # For Rest of the Variables like country_txt, weapontype1_txt, natlty1_txt,
       # We are going to use Label Encoding.
       # Since, There are too many categories in rest of the columns like provstate, city, target1, we are going to drop those columns
```

```
[373]: data_cleaned = data_cleaned.drop(labels=['provstate', 'city', 'target1'], axis=1)
```

TREATMENT

Dropping those columns that have value count > 30.

Dropping [' provstate ', ' city ', ' target1 '] from both Train and Test Data.

Step 8: Handling Missing Values

Observation:

In Train data 5 columns still have few missing values.

In Test data 7 columns still have few missing values.

MISSING TREATMENT

Dropping the rows having missing values

Shape of Train = 100249, 27

Shape of Test = 90668, 27

Step 9: Label Encoding

Observation: We have 6 categorical columns other than target variable, we need to do one hot encoding for better predictive modelling.

```
[393]: df_encoded = pd.get_dummies(data_cleaned, columns=columns, drop_first=True, dtype=int)
       df_encoded.shape

[393]: (80297, 73)
```

ONE HOT ENCODING

After One Hot Encoding.

Shape of Train = 100249, 73

Shape of Test = 90668, 74

Step 10: Balancing the data

Observation: Train data has 73 columns and Test data has 74 columns, after checking we found that Test data has one columns for Radiological weapon (“weapontype1_txt_Radiological”). It came with UNKOWN gname, our Model will not be trained on this and hence we can predict the gname. Therefore, we will remove this column.

```
missing_cols = [col for col in X_test_encoded.columns if col not in df_encoded.columns]
print("Columns in X_test_encoded but not in df_encoded:\n", missing_cols)
```

```
Columns in X_test_encoded but not in df_encoded:
 ['weaptype1_txt_Radiological']
```

[100]:

```
X_test_encoded = X_test_encoded.drop(columns=missing_cols)
```

[101]:

```
X_test_encoded.shape
```

[101]:

```
(90668, 73)
```

Shape of Train = 100249, 73

Shape of Test = 90668, 73

Step 12: Ordinal Encoding

Observation: Handling Unseen Categories in Test Data. When using encoders like Label Encoder, an issue can arise if the test data contains categories that were not present during training. For example, a country like "Dominican Republic" may appear in X_test['country_txt'] but not in the training data df_encoded['country_txt']. In such cases, Label Encoder will raise a Value Error, as it cannot handle unseen categories.

Recommended Solution

To avoid this problem, it is advisable to use Ordinal Encoder from sklearn with the parameter: handle unknown='use_encoded_value', unknown_value = -1. This allows the encoder to handle unseen categories by encoding them as -1, thereby preventing crashes during prediction.

Note: Ordinal Encoder is intended for feature variables (X) and not for target variables (y).

You can later identify and handle these -1 values in your pipeline or prediction logic, depending on how you want to treat unseen categories.

```
# Ordinal Encoding Variables with more than 30 categories (High Cardinality)
# natlty1_txt, country_txt , Since these are relevent columns for prediction

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder

cols = ['country_txt', 'natlty1_txt']
oe = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
df_encoded[cols] = oe.fit_transform(df_encoded[cols].astype(str))
X_test_encoded[cols] = oe.transform(X_test_encoded[cols].astype(str))
```

ORDINAL ENCODING

After Ordinal Encoding.

Shape of Train = 100249, 73

Shape of Test = 90668, 73

Step 11: Encoding Target Variable

Observation: We have 217 group names to predict for the Unknown attacks. We will encode gname with Label encoding.

```
le_encoder = LabelEncoder()

df_encoded['gname_encoded'] = le_encoder.fit_transform(df_encoded['gname'])
```

```
[105]:
df_encoded['gname_encoded'].unique()
```

```
[105]:
array([ 37, 114, 194, 216, 147, 165, 148, 156, 101, 181, 124,  86, 211,
       184, 208,  61, 153, 169, 146,  36,  87,  38, 210, 131, 140, 166,
       176, 172,  51, 163,  67,  21,  63,  27,  85, 174, 126, 115, 192,
        72, 118, 178, 129,   2, 151,  46,  19, 159, 167, 171,  64,  43,
        54, 187, 106, 144,  62, 201,  53, 104, 130, 113, 132,  57, 145,
       136, 177, 158, 120, 185,  84,  73, 200, 188,  49, 149,  78,  56,
        14, 214,  71, 134, 119, 198,  23, 180, 207, 138, 109, 123, 135,
        22, 189, 186, 125,  92,  15, 150,  74, 133, 160, 203, 103, 213,
       128, 164,  39, 155,  75, 105,  80, 137,  93,  65, 173,  32,  52,
        97, 154, 170,   4,  88, 152,  82, 141, 139,  42,   7, 179,  26,
         0, 102, 116, 197, 108, 209, 111,  60, 112,  13, 205, 183, 215,
        95,  12, 100, 142,   3,  58, 122,  34,  68,   8,   9,  29, 202,
        44,  81, 127,  99,  76,  98,  90,  10,  55,  28,  11, 199,  31,
        47, 143, 182, 161,  30, 110,  40, 193, 157,  69, 191,  16, 212,
        66,   6, 195, 168, 162,  33,  17, 204, 121,  91,  20, 196,   5,
        59, 117, 190,  35, 206, 107,  45,   1,  24,  94,  89,  48,  77,
       175,  96,  70,  25,  83,  18,  79,  41,  50])
```

Step 12: Splitting the Train data

We split the dataset into training and testing sets to evaluate the model's performance on unseen data. The training set is used to teach the model, while the test set is used to assess how well it generalizes. This helps prevent overfitting and ensures a fair evaluation. A typical split is 80% for training and 20% for testing. We use train_test_split from sklearn. model_selection with a fixed random_state for reproducibility.

```
# Splitting data into independent and dependent variables
x = df_encoded.drop(columns=['gname', 'gname_encoded'])
y = df_encoded['gname_encoded']
```

```
[108]:
```

```
# Splitting data usingt train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)

# Train and Test Shape
print('Training Data: x=%s, y=%s ' % (x_train.shape, y_train.shape))
print('Test Data: x=%s, y=%s' % (x_test.shape, y_test.shape))
```

```
Training Data: x=(80199, 72), y=(80199,)
Test Data: x=(20050, 72), y=(20050,)
```

AFTER SPLIT

X_train = 80199 ,72

Y_train = 80199

X_test = 20050 ,72

Y_test = 20050

Model Training

ADA Boost

```
ypred_ada = model_ada.predict(x_test)
```

[214]:

```
accuracy_ada = accuracy_score(y_test, ypred_ada)

print('Accuracy ADA: ', accuracy_ada)
```

```
Accuracy ADA:  0.8052369077306734
```

Logistic Regression: -

```
ypred_h_lr = model_lr.predict(x_test_scaled)
ypred_s_lr = model_lr.predict_proba(x_test_scaled)[:, 1]
```

[220]:

```
accuracy_lr = accuracy_score(y_test, ypred_h_lr)
recall_lr = recall_score(y_test, ypred_h_lr, average='weighted')
precision_lr = precision_score(y_test, ypred_h_lr, average='weighted')

print(f'Accuracy Score LR: {accuracy_lr}')
print(f'Recall Score LR: {recall_lr}')
print(f'Precision Score LR: {precision_lr}')
```

```
Accuracy Score LR: 0.6786533665835411
Recall Score LR: 0.6786533665835411
Precision Score LR: 0.6188329542366131
```

Decision Tree: -

```
ypred_h_dt = model_dt.predict(x_test)
ypred_s_dt = model_dt.predict_proba(x_test)[:, 1]
```

[250]:

```
accuracy_dt = accuracy_score(y_test, ypred_h_dt)
recall_dt = recall_score(y_test, ypred_h_dt, average='weighted')
precision_dt = precision_score(y_test, ypred_h_dt, average='weighted')

print(f'Accuracy Score LR: {accuracy_dt}')
print(f'Recall Score LR: {recall_dt}')
print(f'Precision Score LR: {precision_dt}')
```

```
Accuracy Score LR: 0.7809476309226933
Recall Score LR: 0.7809476309226933
Precision Score LR: 0.7856895695217953
```

Random Forest: -

```
ypred_h_rf = model_rf.predict(x_test)
```

```
[232]:
```

```
accuracy_rf = accuracy_score(y_test, ypred_h_rf)
recall_rf = recall_score(y_test, ypred_h_rf, average='weighted')
precision_rf = precision_score(y_test, ypred_h_rf, average='weighted')

print(f'Accuracy Score LR: {accuracy_rf}')
print(f'Recall Score LR: {recall_rf}')
print(f'Precision Score LR: {precision_rf}')
```

```
Accuracy Score LR: 0.816708229426434
Recall Score LR: 0.816708229426434
Precision Score LR: 0.8080266821490387
```

XG Boost: -

```
ypred_xgb = model_xgb.predict(x_test)

accuracy_xgb = accuracy_score(y_test, ypred_xgb)
# recall_xgb = recall_score(y_test, ypred_xgb)
# precision_xgb = precision_score(y_test, ypred_xgb)

print(f'Accuracy Score LR: {accuracy_xgb}')
# print(f'Recall Score LR: {recall_xgb}', ave)
# print(f'Precision Score LR: {precision_xgb}')
```

```
Accuracy Score LR: 0.5222942643391522
```