Presented by Tasneem Samy

Global Terrorism

Data Analysis

Introduce to Eng: Ibrahim El-Shal

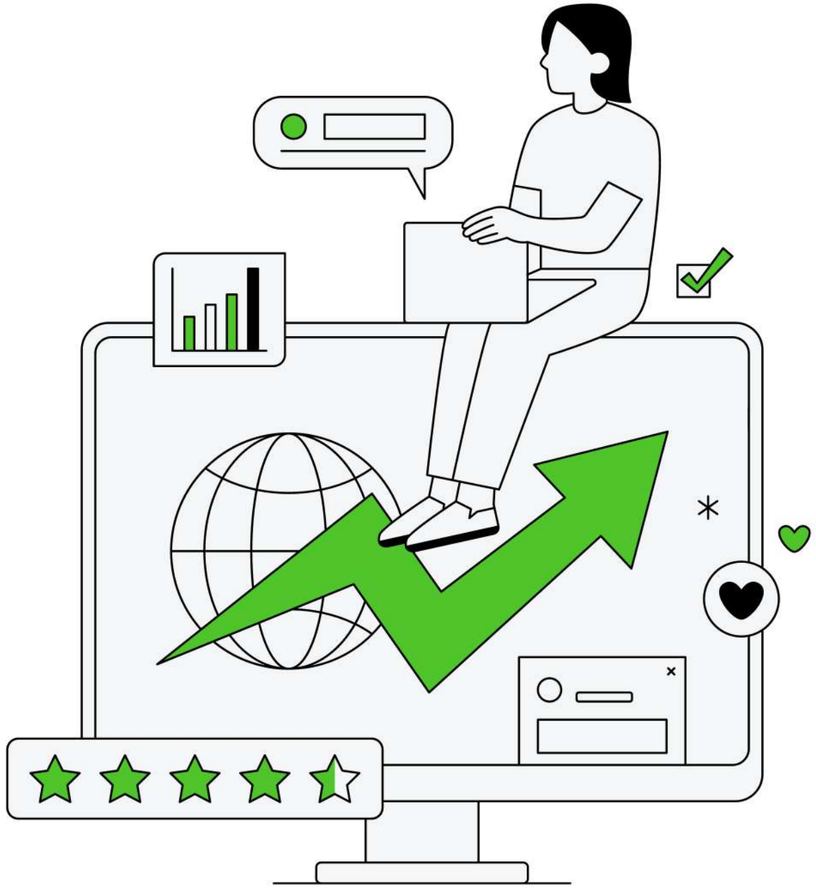


Table Of Content

01.

Performing Data Cleaning, Exploration and Visualization using Pandas

02.

Gathering all insights in Dash board using power bi

03.

Dive into using Dask and performing similar steps on it, and make the comparisons

Note That with each section it mentioned the challenges that I faced

1- Reading Dataset

1- Starting with importing used libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

✓ 1.3s

2- Try to read csv file (first challenge find the right encoding type)

Using chardet library to know the exact encoding type for the CSV file

```
import chardet
with open('globalterrorismdb_0718dist.csv','rb') as obj:
    result=chardet.detect(obj.read(20000))
print(result)
```

[2] ✓ 0.1s

```
{'encoding': 'ISO-8859-1', 'confidence': 0.73, 'language': ''}
```

Chardet library is the easiest way to Detecting files encoding

3- Load csv file to data frame

```
df=pd.read_csv('globalterrorismdb_0718dist.csv',encoding='ISO-8859-1')
pd.set_option('display.max_columns', None)
```

set options to show all columns in df

4- Display head of data frame

```
df.head(10)
```

✓ 0.0s

	eventid	iyear	imonth	iday	approxdate	extended	resolution	country	country_txt	region	region_txt	provstate	city
0	197000000001	1970	7	2	NaN	0	NaN	58	Dominican Republic	2	Central America & Caribbean	NaN	Santo Domingo
1	197000000002	1970	0	0	NaN	0	NaN	130	Mexico	1	North America	Federal	Mexico city
2	197001000001	1970	1	0	NaN	0	NaN	160	Philippines	5	Southeast Asia	Tarlac	Unknown

5- Show info of data frame

Using verbose to print the full summary

```
df.info(verbose = True)
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 181691 entries, 0 to 181690
Data columns (total 135 columns):
 #    Column              Dtype
---   ------              -----
 0    eventid             int64
 1    iyear               int64
 2    imonth              int64
 3    iday                int64
 4    approxdate          object
 5    extended            int64
 6    resolution          object
 7    country             int64
 8    country_txt         object
 9    region              int64
 10   region_txt          object
 11   provstate           object
 12   city                object
 13   latitude            float64
 14   longitude           float64
 15   specificity         float64
 16   vicinity            int64
```

2- Cleaning Dataset

1- Show the duplicates row in data (it has no duplicates)

```
df[df.duplicated()]
```

✓ 2.6s

eventid	iyear	imonth	iday	approxdate	extended	resolution	country	country_txt	region	region_txt	provstate	city

2- Show the percentage of null values

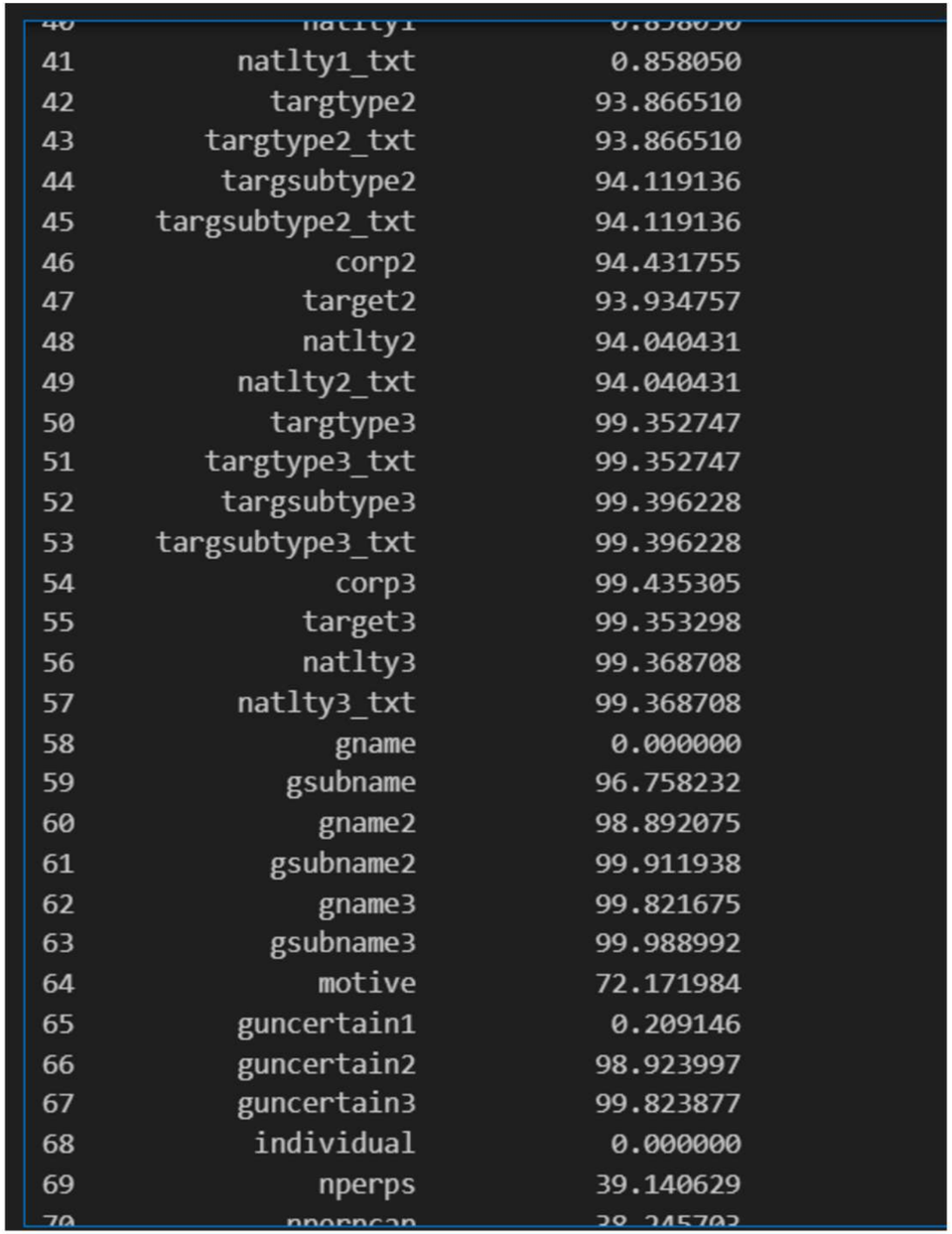
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
missing_values_percentage = (df.isnull().sum() / len(df)) * 100
missing_values_df = missing_values_percentage.reset_index()
missing_values_df.columns = ['Column', 'Missing Percentage']

print(missing_values_df)
```

Challenge2: Handling Null Values and Choose Features:

Choosing Features (Columns) was based on:

1- Percentage of null (I can’t choose columns that has more than 70 % of null even if it’s important because I can’t make insights from it)

2- Related to Success of Attack



41	natlty1_txt	0.858050
42	targtype2	93.866510
43	targtype2_txt	93.866510
44	targsubtype2	94.119136
45	targsubtype2_txt	94.119136
46	corp2	94.431755
47	target2	93.934757
48	natlty2	94.040431
49	natlty2_txt	94.040431
50	targtype3	99.352747
51	targtype3_txt	99.352747
52	targsubtype3	99.396228
53	targsubtype3_txt	99.396228
54	corp3	99.435305
55	target3	99.353298
56	natlty3	99.368708
57	natlty3_txt	99.368708
58	gname	0.000000
59	gsubname	96.758232
60	gname2	98.892075
61	gsubname2	99.911938
62	gname3	99.821675
63	gsubname3	99.988992
64	motive	72.171984
65	guncertain1	0.209146
66	guncertain2	98.923997
67	guncertain3	99.823877
68	individual	0.000000
69	nperps	39.140629

Choosing columns that I will continue with

```
cleaned_df=df[['eventid', 'iyear', 'imonth', 'iday', 'extended', 'country',
       'country_txt', 'region', 'region_txt', 'provstate', 'city', 'latitude',
       'longitude', 'specificity', 'vicinity', 'summary', 'crit1',
       'crit2', 'crit3', 'doubtterr', 'multiple', 'success', 'suicide',
       'attacktype1', 'attacktype1_txt', 'targtype1', 'targtype1_txt',
       'targsubtype1', 'targsubtype1_txt', 'corp1', 'target1', 'natlty1',
       'natlty1_txt', 'gname', 'motive', 'guncertain1', 'individual', 'nperps',
       'nperpcap', 'claimed', 'weaptype1', 'weaptype1_txt', 'weapsubtype1',
       'weapsubtype1_txt', 'weapdetail', 'nkill', 'nkillus', 'nkillter',
       'nwound', 'nwoundus', 'nwoundte', 'property', 'propextent',
       'propextent_txt', 'propcomment', 'ishostkid', 'ransom',
       'scite1', 'scite2','dbsource', 'INT_LOG', 'INT_IDEO',
       'INT_MISC', 'INT_ANY']]
```

Rename Some columns to be Clear to use

```
cleaned_df.rename(columns =
                  {'iyear':'year',
                  'imonth':'month',
                  'iday':'day',
                  'country':'country_code',
                  'country_txt' : 'country',
                  'region': 'regioncode',
                  'region_txt' : 'region',
                  'crit1' : 'crit'
```

Challenge3 : Validity of Numeric Columns

```
cleaned_df.describe()
```

vicinity	doubtterr	nperps	nperpcap	claime_dresp
181691.000000	181690.000000	110576.000000	112202.000000	115571.000000
0.068297	-0.523171	-65.361154	-1.517727	0.049666
0.284553	2.455819	216.536633	12.830346	1.093195
-9.000000	-9.000000	-99.000000	-99.000000	-9.000000
0.000000	0.000000	-99.000000	0.000000	0.000000
0.000000	0.000000	-99.000000	0.000000	0.000000
0.000000	0.000000	1.000000	0.000000	0.000000
1.000000	1.000000	25000.000000	406.000000	1.000000

and so on

Cleaning was based on Knowing it original value (0 or 1)
and know rows that have invalid value and replace it with zero

```
cleaned_df['doubtterr'].unique()
```
✓ 0.0s

```
array([ 0., -9.,  1., nan])
```

```
cleaned_df[cleaned_df['doubtterr']<0].shape[0]
```
✓ 0.0s

```
13784
```

```
cleaned_df['doubtterr'] = cleaned_df['doubtterr'].replace(-9, 0)
```
✓ 0.0s

and so on

Validity of Categorical Columns like (country, region, weopon...

```
print(cleaned_df['country'].nunique())
print(cleaned_df['country'].str.lower().str.strip().nunique())
```

[47] ✓ 0.0s

```
205
205
```

```
cleaned_df['region'].unique()
```

[53] ✓ 0.0s

```
array(['Central America & Caribbean', 'North America', 'Southeast Asia',
       'Western Europe', 'East Asia', 'South America', 'Eastern Europe',
       'Sub-Saharan Africa', 'Middle East & North Africa',
       'Australasia & Oceania', 'South Asia', 'Central Asia'],
      dtype=object)
```

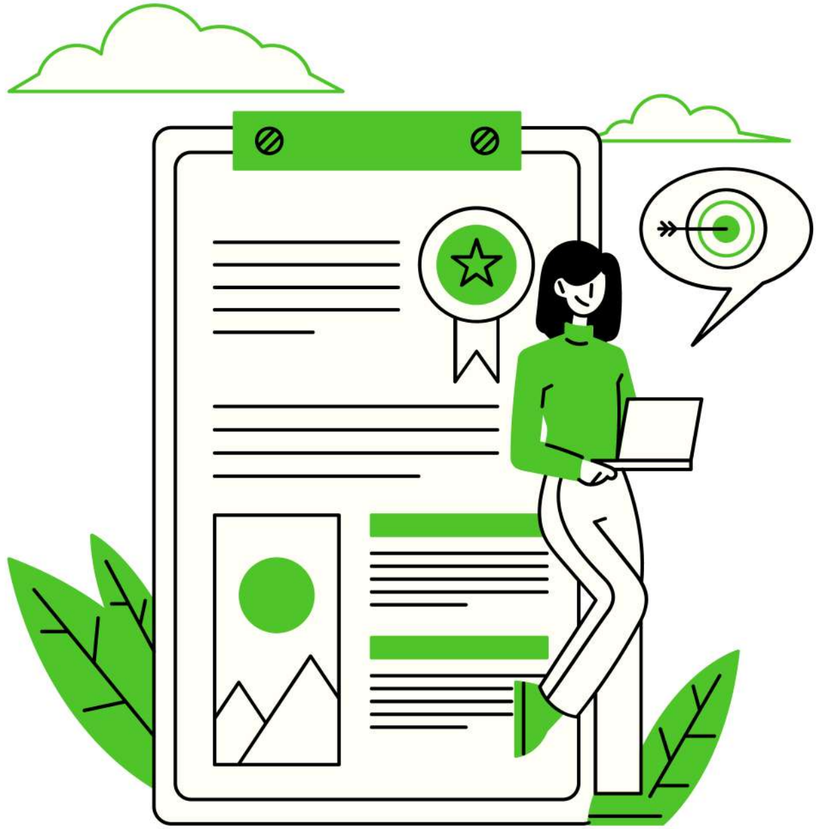
```
cleaned_df['attack_type'].unique()
```

[ ]

```
array(['Assassination', 'Hostage Taking (Kidnapping)',
       'Bombing/Explosion', 'Facility/Infrastructure Attack',
       'Armed Assault', 'Hijacking', 'Unknown', 'Unarmed Assault',
       'Hostage Taking (Barricade Incident)'], dtype=object)
```

it was already valid

3- Data Analysis



1- Choosing the most frequent Value of categorical columns using Numpy

```python
def get_most_frequent(values):
    values = np.array(values)
    unique, counts = np.unique(values, return_counts=True)
    max_count_index = np.argmax(counts)
    return unique[max_count_index]

categorical_columns = [
    'country', 'region', 'provstate', 'city', 'attack_type', 'target_type',
    'targsubtype1_txt', 'corp1', 'target1', 'nationality_of_victom',
    'organisation', 'motive', 'weaptype1', 'weapon_type', 'weapsubtype1_txt',
    'weapdetail', 'propextent_desc', 'propcomment', 'scite1', 'scite2','dbsource'
]


frequent_values = {}
for col in categorical_columns:
    frequent_values[col] = get_most_frequent(cleaned_df[col].dropna())

frequent_values_df = pd.DataFrame(frequent_values, index=['Most Frequent'])
frequent_values_df.head()
```

We see that most attack happened in **Iraq**, most of targets with attacking **civilians**, Most of the attacks were with **explosives** and most cases wasn't knowing exactly what was the motive of terrorists

	country	region	provstate	attack_type	target_type	target_subtype	organisation	motive	weapon_type
Most Frequent	Iraq	Middle East & North Africa	Baghdad	Bombing/Explosion	Private Citizens & Property	Unnamed Civilian/Unspecified	Unknown	Unknown	Explosives

2- Making insights with Pandas

Knowing numbers of attacks over years

```
attacks_per_year = cleaned_df.groupby('year').size().reset_index(name='num_attacks').sort_values(by='num_attacks',ascending=False)
attacks_per_year
```

0.0s

year	num_attacks
2014	16903
2015	14965
2016	13587
2013	12036
2017	10900

We notice that Maximum number of attacks happen in 2014 with 16903 attacks and the terrorist attacks have increased in recent years.

Knowing numbers of attacks over countries

```
tacks_per_country = cleaned_df.groupby('country').size().reset_index(name='num_attacks').sort_values(by='num_attacks',ascending=False
tacks_per_country.head(20)
```

.0s

country	num_attacks
Iraq	24636
Pakistan	14368
Afghanistan	12731
India	11960
Colombia	8306

We notice that Maximum number of attacks happen in Iraq with 24636 attacks , next was Pakistan with 14368 attacks followed by Afghanistan with 12731.

Knowing numbers of attacks over regions

```
attacks_per_region = cleaned_df.groupby('region').size().reset_index(name='num_attacks').sort_values(by='num_attacks',ascending=False)
attacks_per_region
```

region	num_attacks
Middle East & North Africa	50474
South Asia	44974
South America	18978
Sub-Saharan Africa	17550
Western Europe	16639
Southeast Asia	12485
Central America & Caribbean	10344
Eastern Europe	5144
North America	3456
East Asia	802
Central Asia	563
Australasia & Oceania	282

We notice that Maximum number of attacks happen in Middle East & North Africa with 50474 attacks next South Asia and the least region was Australasia & Oceania

Most Attack types

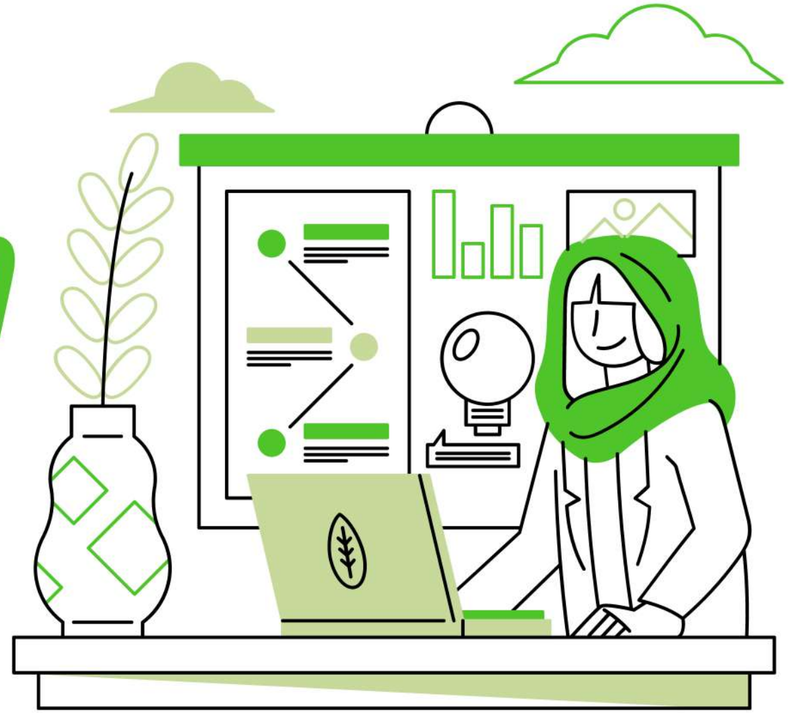
```
attacks_per_attack_type = cleaned_df.groupby('attack_type').size().reset_index(name='num_attacks').sort_values(by='num_attacks',ascending=False)
attacks_per_attack_type
```

0.0s Pyth

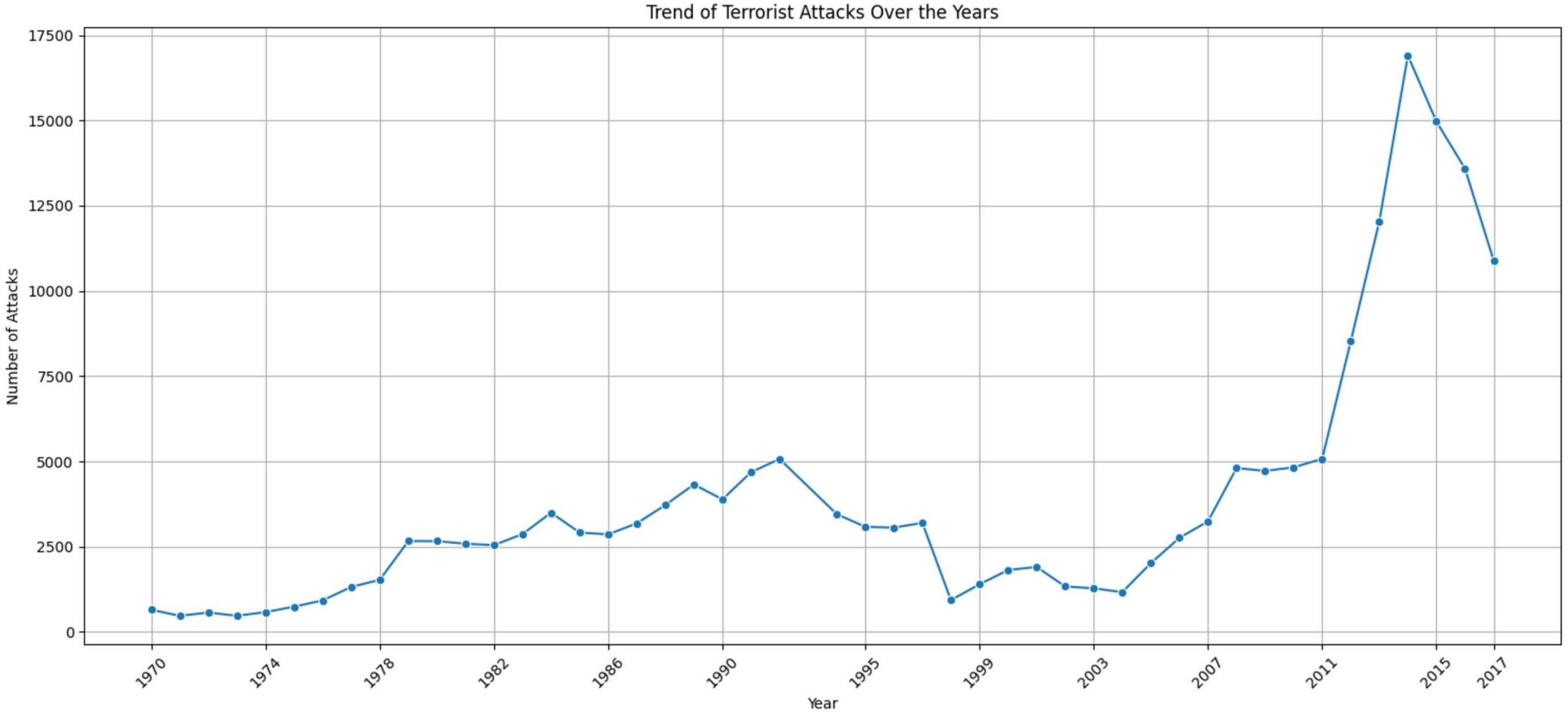
attack_type	num_attacks
Bombing/Explosion	88255
Armed Assault	42669
Assassination	19312
Hostage Taking (Kidnapping)	11158
Facility/Infrastructure Attack	10356

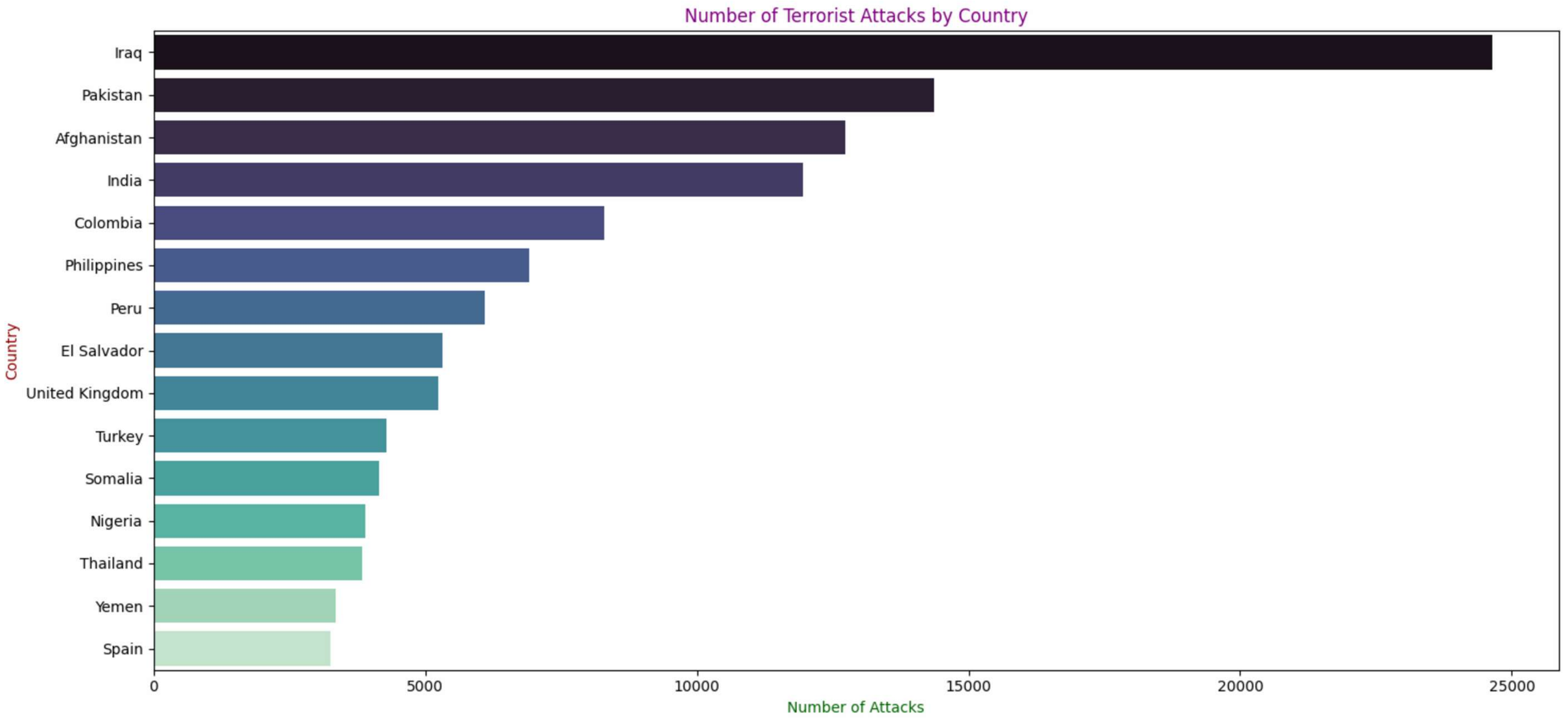
We notice that Most attacks was Bombing/Explosion with 88255 attacks next was Armed Assault with 42669 attacks .

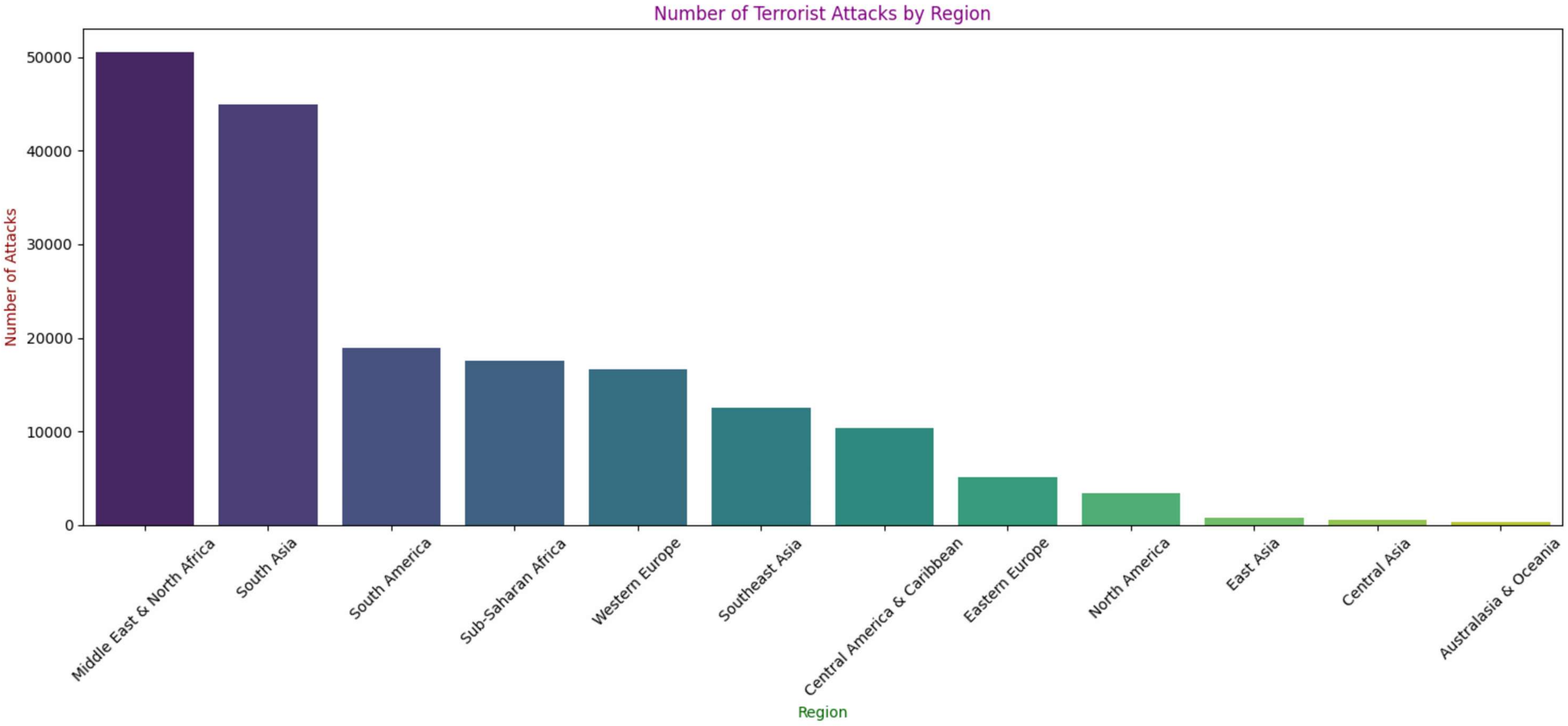
4- Data Visualization with python



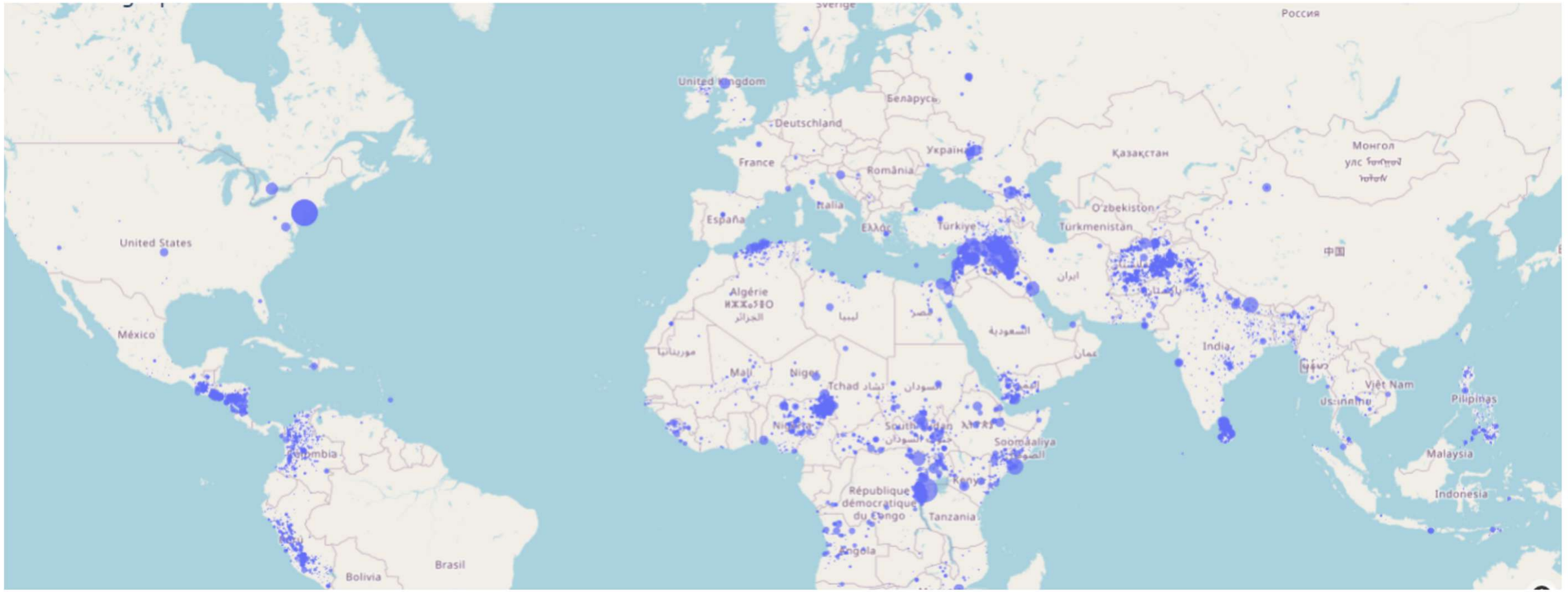
Trend of Terrorist Attacks over years



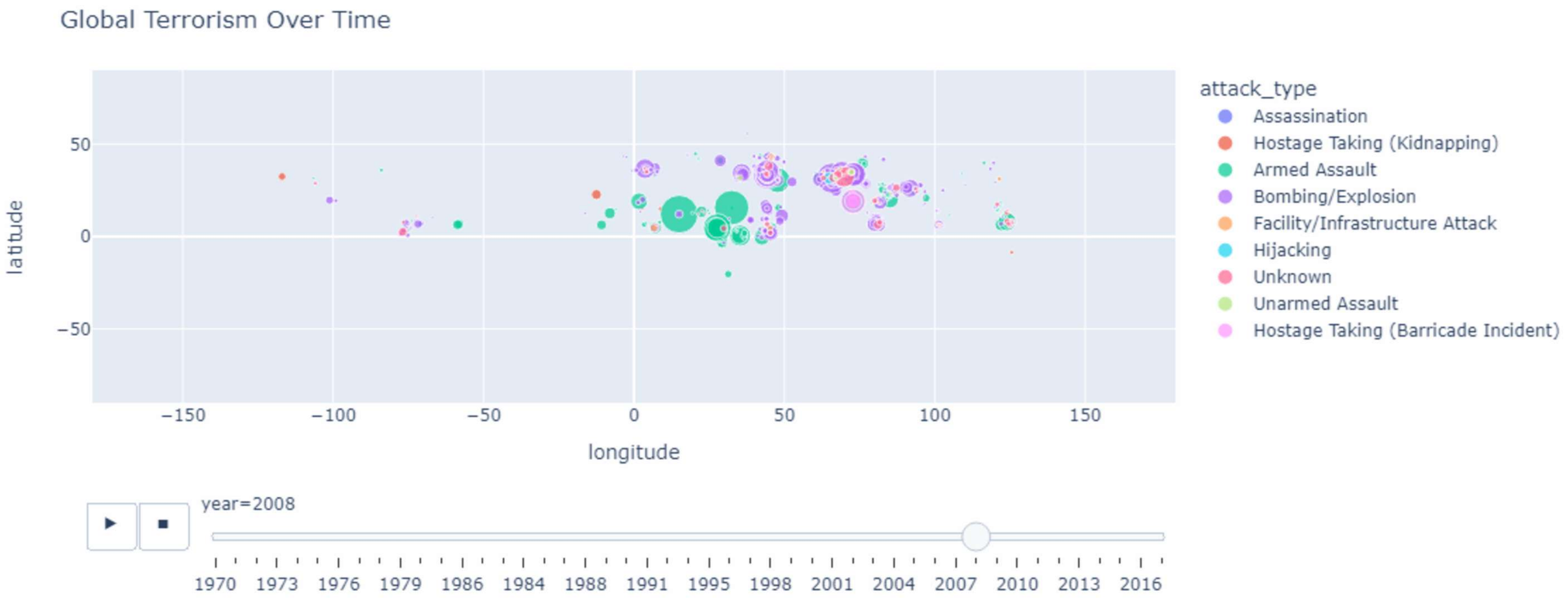

Number of Terrorist Attacks by Country
Iraq
Pakistan
Afghanistan
India
Colombia
Philippines
Peru
El Salvador
United Kingdom
Turkey
Somalia
Nigeria
Thailand
Yemen
Spain
Country
0
5000
10000
15000
20000
25000
Number of Attacks


Number of Terrorist Attacks by Region
Number of Attacks
0
10000
20000
30000
40000
50000
Middle East & North Africa
South Asia
South America
Sub-Saharan Africa
Western Europe
Southeast Asia
Central America & Caribbean
Eastern Europe
North America
East Asia
Central Asia
Australasia & Oceania
Region

Geographic Distribution of Terrorist Attacks



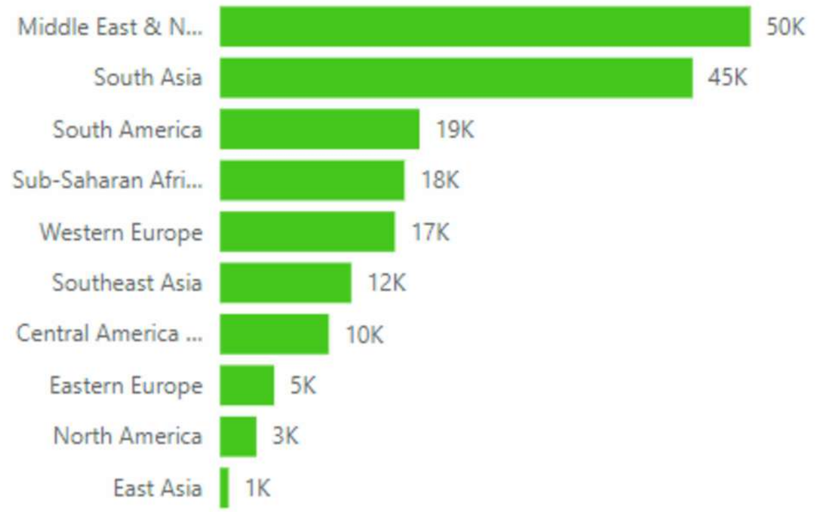
Time series animation showing the spread of terrorism over the years

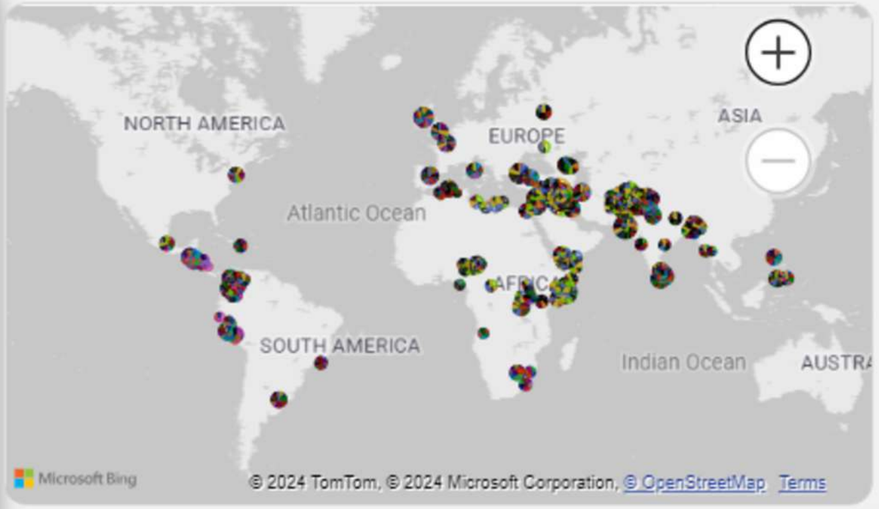


5- Power BI Dash Board

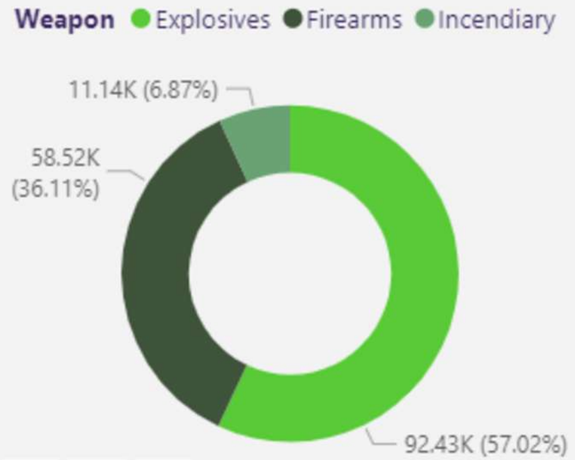
Global Terrorism

Attacks Per Region

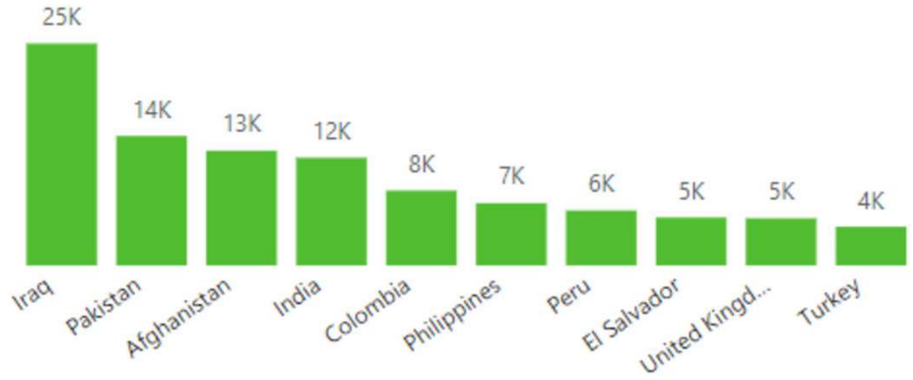




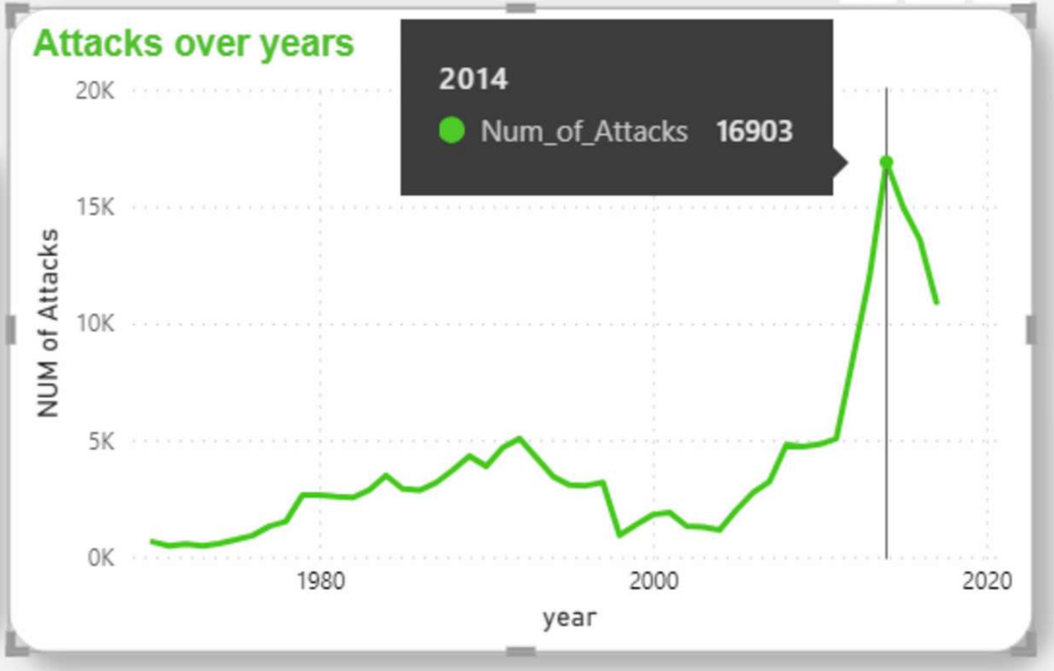
Attacks Per Weapon



Attacks Per Country



Attacks over years



412K
Total Number Killed

524K
Total Number Wounded

6- Working With Dask

1- Importing Libraries

```
import pandas as pd
import numpy as np
import dask.dataframe as dd
import time
```

2- Comparing read csv file (challenge1: Facing problems in reading with dask)

```
import warnings
warnings.filterwarnings('ignore')
```

we used it filter warning

Dask was faster in reading csv file

```
df = pd.read_csv('globalterrisom.csv')
```
✓ 2.0s

```
dask_df=dd.read_csv('globalterrisom.csv')
```
✓ 0.0s

3- Comparing Memory Usage

```
pandas_memory_usage=memory_profiler.memory_usage()[0]
df = pd.read_csv('globalterrorismdb_0718dist.csv',encoding='ISO-8859-1')
print(f"Pandas memory usage: {pandas_memory_usage}")
```
✓ 2.6s

```
Pandas memory usage: 118.2265625
```

```
dask_memory_usage=pandas_memory_usage=memory_profiler.memory_usage()[0]
dask_df=dd.read_csv('globalterrorismdb_0718dist.csv',encoding='ISO-8859-1')
print(f"Dask memory usage: {dask_memory_usage}")
```
✓ 0.1s

```
Dask memory usage: 448.0390625
```

Dask consume more memory

4- Comparing Summarizing Data

Comparing summerizing data

```
start_time = time.time()
df.describe()
end_time = time.time()
print(f"Pandas time: {end_time - start_time}")
```

✓ 0.3s

```
Pandas time: 0.3373904228210449
```

```
start_time = time.time()
dask_df.describe()
end_time = time.time()
print(f"Dask time: {end_time - start_time}")
```

✓ 0.2s

```
Dask time: 0.2223215103149414
```

Dask Faster

5-Comparing Aggregation operations

```
start_time = time.time()
attacks_per_weapon_type = df.groupby('iyear').size()
attacks_per_weapon_type = df.groupby('country_txt').size()
attacks_per_weapon_type = df.groupby('region_txt').size()
attacks_per_weapon_type = df.groupby('weaptype1_txt').size()
end_time = time.time()
print(f"Pandas time: {end_time - start_time}")
```

✓ 14.6s

```
Pandas time: 14.692779064178467
```

```
start_time = time.time()
attacks_per_weapon_type = dask_df.groupby('iyear').size().compute()
attacks_per_weapon_type = dask_df.groupby('country_txt').size().compute()
attacks_per_weapon_type = dask_df.groupby('region_txt').size().compute()
attacks_per_weapon_type = dask_df.groupby('weaptype1_txt').size().compute()
end_time = time.time()
print(f"Dask time: {end_time - start_time}")
```

✓ 0.1s

```
Dask time: 0.10119462013244629
```

Dask was faster

Thank you very much!