GLOBAL TERRORISM EDA

ENG

Ibrahim el-shal

By

Hisham Karam Hashem

Report

Global Terrorism EDA

- **Objective:** The goal of this project is to make EDA and finding the insights derived from the data.

1.1 About Global terrorism Dataset

- **Data source :-**

Information on more than 180,000 Terrorist Attacks

The Global Terrorism Database (GTD) is an open-source database including information on terrorist attacks around the world from 1970 through 2017. The GTD includes systematic data on domestic as well as international terrorist incidents that have occurred during this time period and now includes more than 180,000 attacks. The database is maintained by researchers at the National Consortium for the Study of Terrorism and Responses to Terrorism (START), headquartered at the University of Maryland.

1.2 Methodology

☒ **Reading data**

- Importing data using pandas with encoding 'latin1'.
- Display the data columns & rows.

☒ **Feature selection (**Select some important columns for EDA**)**

(eventid, iyear, imonth, iday, country_txt, region_txt, provstate, City, attacktype1_txt, nkill, nwound, target1, summary, gname, Targtype1_txt, weaptype1_txt, motive, success).

☒ **Data quality & missing values investigation (1st preprocessing step)**

```
data.isnull().sum()

eventid                  0
iyear                    0
imonth                   0
iday                     0
country_txt              0
region_txt               0
provstate              421
city                   435
attacktype1_txt          0
nkill                10313
nwound               16311
target1                638
summary              66129
gname                    0
targtype1_txt            0
weaptype1_txt            0
motive              131130
success                  0
dtype: int64
```

☒ Handle missing values

- Drop the rows of (provstate – city – target1) as they have little nulls.

```
data = data[data['provstate'].notna()]
data = data.reset_index(drop=True)
```

```
data = data[data['city'].notna()]
data = data.reset_index(drop=True)
```

```
data = data[data['target1'].notna()]
data = data.reset_index(drop=True)
```

- Fill the rows of (nkill – nwound) with the mean value.
- Fill the rows of (motive – summary) with the mod value.

```
data["nkill"].fillna(data["nkill"].mean(),inplace=True)
```

```
data["nwound"].fillna(data["nwound"].mean(),inplace=True)
```

```
data["motive"].mode()[0]
```

```
'Unknown'
```

```
data["motive"].fillna(data["motive"].mode()[0],inplace=True)
```

```
data["summary"].fillna(data["summary"].mode()[0],inplace=True)
```

☒ Check the columns datatypes

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180204 entries, 0 to 180203
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   eventid           180204 non-null  int64
 1   iyear             180204 non-null  int64
 2   imonth            180204 non-null  int64
 3   iday              180204 non-null  int64
 4   country_txt       180204 non-null  object
 5   region_txt        180204 non-null  object
 6   provstate         180204 non-null  object
 7   city              180204 non-null  object
 8   attacktype1_txt   180204 non-null  object
 9   nkill             180204 non-null  float64
 10  nwound            180204 non-null  float64
 11  target1           180204 non-null  object
 12  summary           180204 non-null  object
 13  gname             180204 non-null  object
 14  targtype1_txt     180204 non-null  object
 15  weaptype1_txt     180204 non-null  object
 16  motive            180204 non-null  object
 17  success           180204 non-null  int64
dtypes: float64(2), int64(5), object(11)
memory usage: 24.7+ MB
```

- convert the float type of (nkill - nwound) to integer

```
# convert float (nkill & nwound) to int
data['nkill'] = data['nkill'].astype(int)
data['nwound'] = data['nwound'].astype(int)
```

☒ check the data duplication

- We observe that there are no duplicate values.

☒ Store the cleaned dataset as (‘data.csv’)

```
data.to_csv('data.csv')
```

☒ Data analysis

1- Calculate the mean, median, and standard deviation of relevant numeric columns

```
nkill_mean = 2.3811838607823224
nkill_median = 1.0
nkill_std = 11.240006377360531
----------------------------------
nwound_mean = 3.158950540212089
nwound_median = 0.0
nwound_std = 34.42633764583603
```

2- Identify the most frequent values in categorical columns.

```
most frequent value in regon : Middle East & North Africa
most frequent value in country : Iraq
most frequent value in state : Baghdad
most frequent value in weapon type : Explosives
most frequent value in attack type : Bombing/Explosion
most frequent value in target type : Private Citizens & Property
```

3- Group data by various categories (e.g., year, region, attack type) and calculate aggregate statistics

this a pivot table to show the weapon type in each region

weaptype1_txt	Biological	Chemical	Explosives	Fake Weapons	Firearms	Incendiary	Melee	Other	Radiological	Sabotage Equipment	Unknown	Vehicle (not to include vehicle-borne explosives, i.e., car or truck bombs)
region_txt												
Australasia & Oceania	0	11	74	0	68	69	9	1	0	0	26	1
Central America & Caribbean	0	2	3089	0	5581	427	64	0	0	5	985	4
Central Asia	0	2	247	1	222	15	13	0	0	0	44	0
East Asia	2	17	320	4	35	240	78	3	10	3	44	8
Eastern Europe	0	12	3040	4	1425	183	90	4	0	4	287	1

4- Identify trends over time (e.g., number of attacks per year).

```
the 5 top years of the high number of attack
```

	iyear	number of attack type
0	2014	16903
1	2015	14965
2	2016	13587
3	2013	12036
4	2017	10898

5- Determine the most affected regions and countries

```
the most affected regions by the terrorism
```

	region_txt	0
0	Middle East & North Africa	50249
1	South Asia	44710
2	South America	18910
3	Sub-Saharan Africa	17519
4	Western Europe	16511

```
the most affected countries by the terrorism
```

	country_txt	0
0	Iraq	24578
1	Pakistan	14337
2	Afghanistan	12578
3	India	11918
4	Colombia	8289

6- Identify the most common attack types and targets

```
the most common types
```

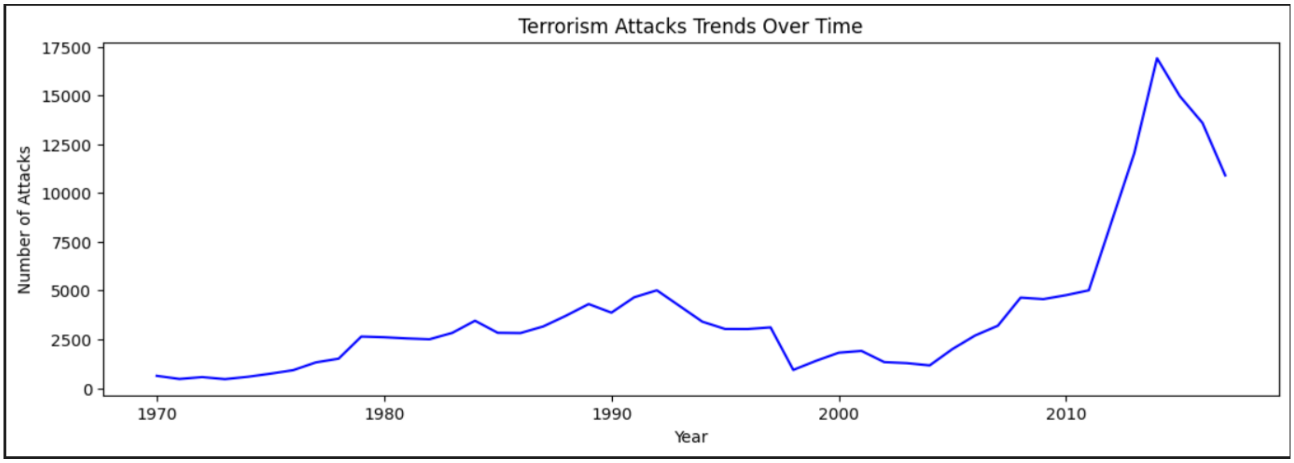
	attacktype1_txt	0
0	Bombing/Explosion	87648
1	Armed Assault	42239
2	Assassination	19164
3	Hostage Taking (Kidnapping)	11094
4	Facility/Infrastructure Attack	10278

```
the most common targets types
```

	target1	0
0	Civilians	6441
1	Unknown	5881
2	Soldiers	3156
3	Patrol	2941
4	Checkpoint	2905

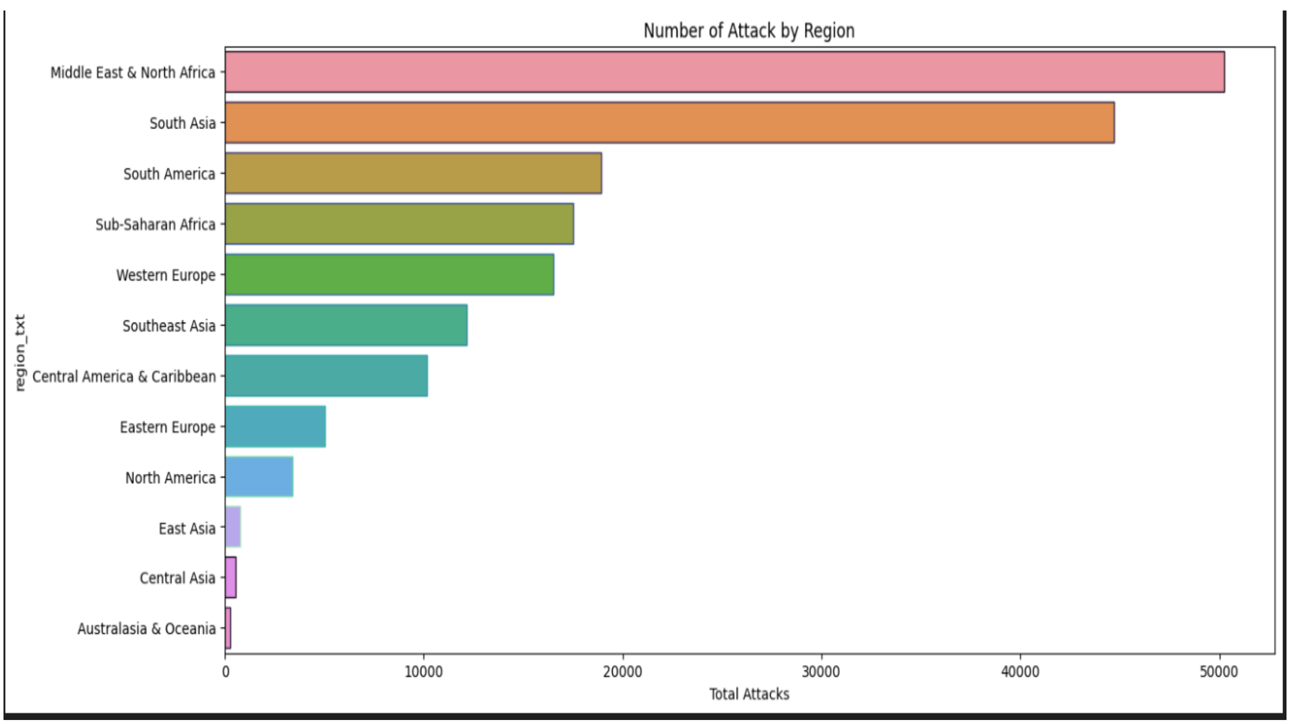
☒ Data visualization

1- Line plot showing the trend of terrorist attacks over the years



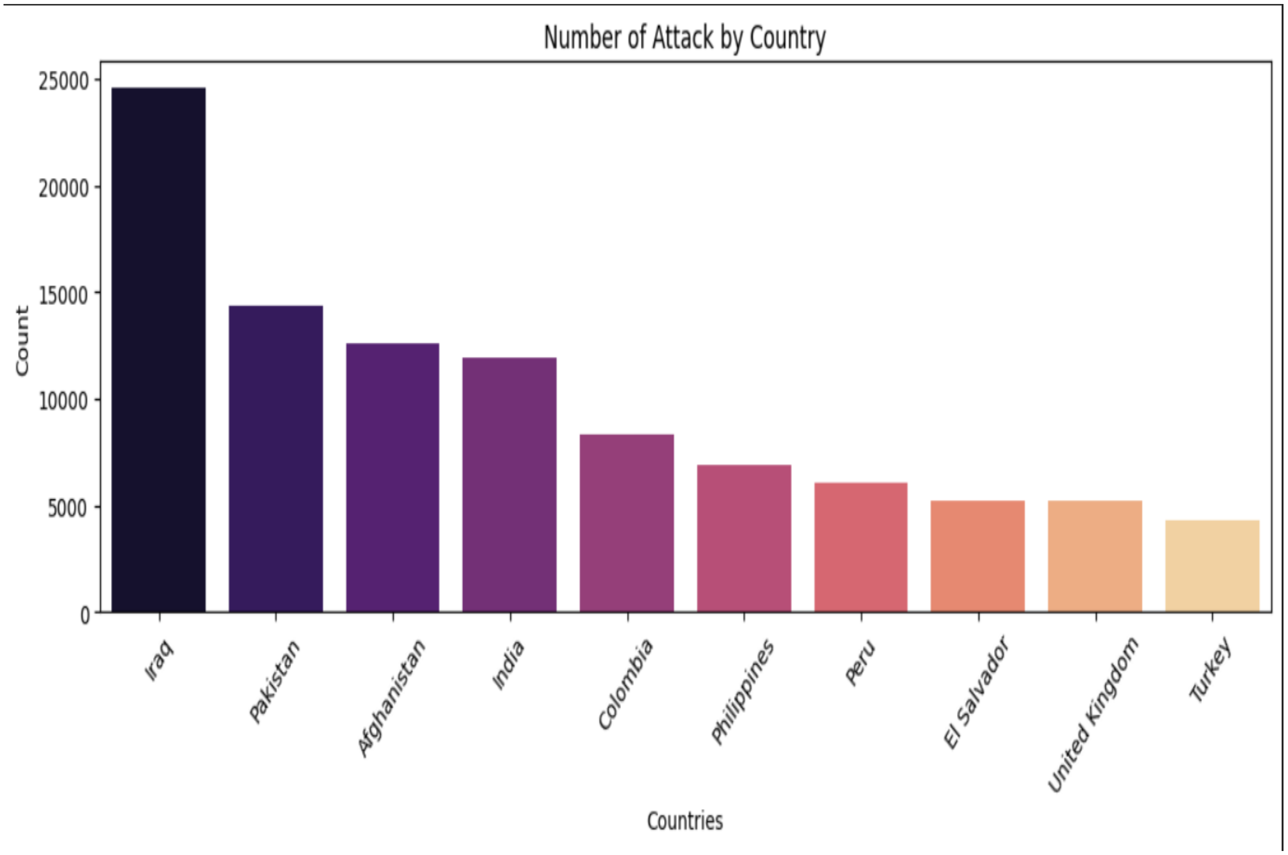
We notice that the terrorist attacks began to raise from 1970 to1990 then down from 1995 to 2000. After 2000 it began to raise until 2015 After that it began to down from 2015 to 2017

2- Bar plot of the number of attacks by region



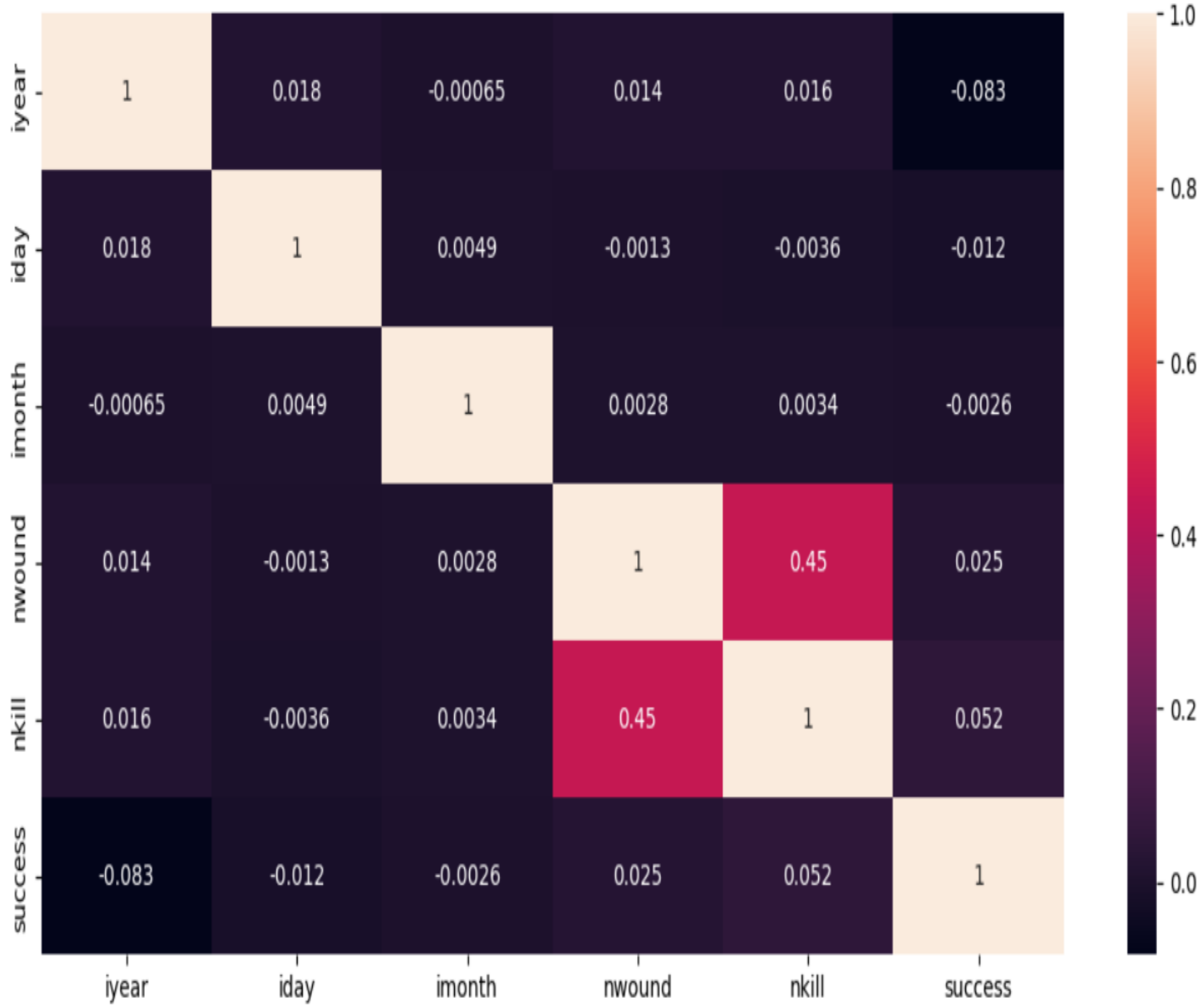
We observe that the Middle-East & North-Africa is the region that has the maximum number of attacks.

3- Bar plot of the number of attacks by country.

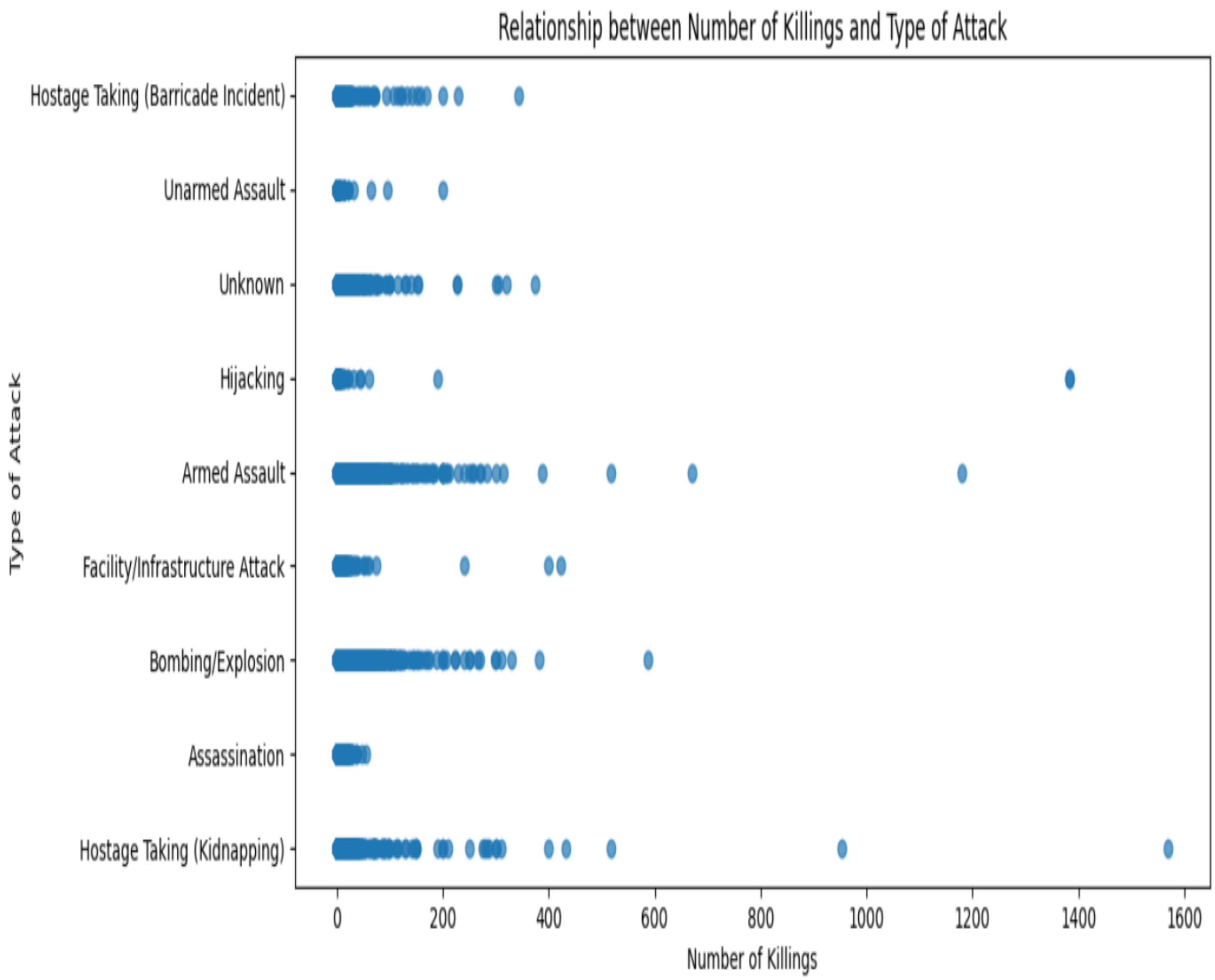


The country of (Iraq) is the top one of number of attack

4- Heatmap to visualize the correlation between different features

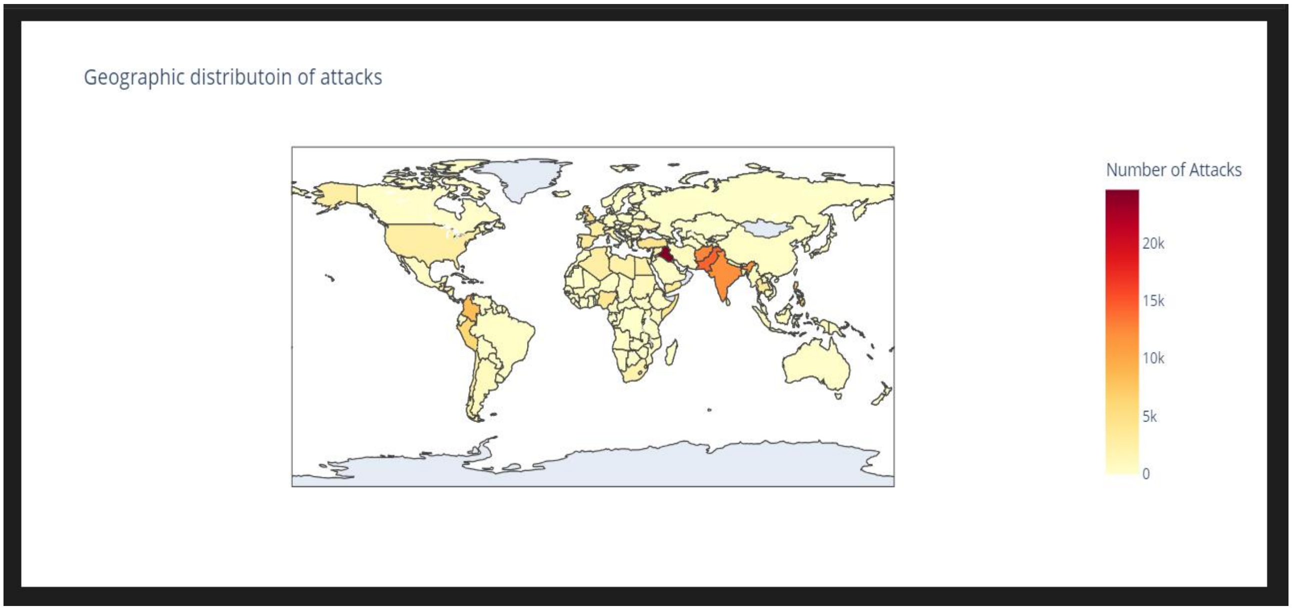


5- Scatter plot showing the relationship between the number of casualties and the type of attack.

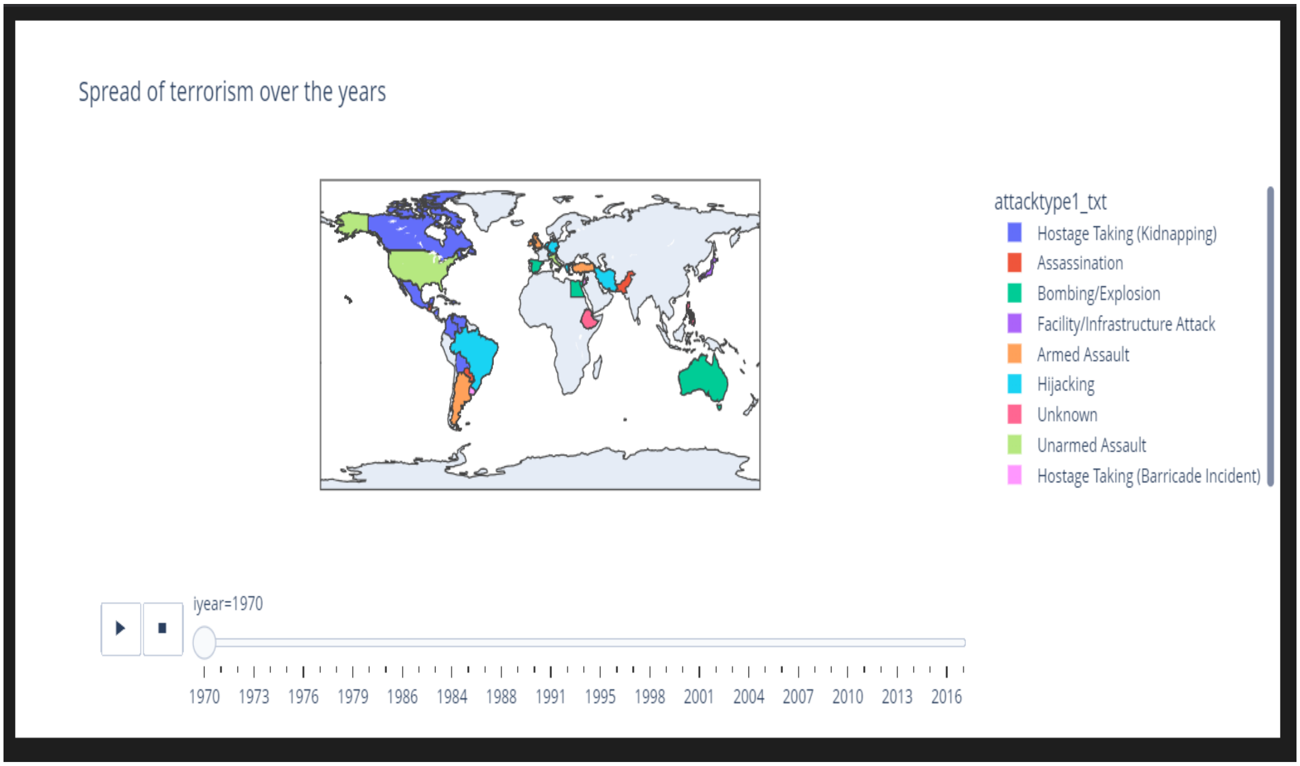


The average number of killings between (0 : 200).

- **Create interactive visualizations using Plotly (optional for advanced students)**
    1- Interactive map to show the geographic distribution of attacks.



2- Time series animation showing the spread of terrorism over the years



☒ **Demonstrate how to use Dask and Compare the performance and memory usage of Dask operations with Pandas.**

```python
import dask.dataframe as dd
from dask.distributed import Client
import time
import memory_profiler
import warnings
warnings.filterwarnings('ignore')
```

Create function (calculate ()) to calculate the time & memory usage

Try with pandas library

```python
# Define a function to calculate the time consuming and memory usage of Pandas operation
# first use the function in data reading operation

def calculate():

    mem_usage_before = memory_profiler.memory_usage()[0] # Measure memory usage

    start_time = time.time() # Measure time

    try:
        data = pd.read_csv("D:\codes/ITI/libariies/final project/globalterrorismdb_0718dist.csv", encoding='latin1', low_memory=False)
    except UnicodeDecodeError as e:
        print(f"Encoding error: {e}")

    mem_usage_after = memory_profiler.memory_usage()[0] # Measure memory usage after reading data

    end_time = time.time()

    print(f"Pandas Memory Usage: {mem_usage_after - mem_usage_before} MB")
    print(f"Pandas Time consuming: {end_time - start_time} seconds")


calculate()
```

```
Pandas Memory Usage: 264.765625 MB
Pandas Time consuming: 4.715858697891235 seconds
```

Use the function (calculate ()) to calculate the time & memory usage for Dask library

```python
# use the function to calculate the read operation with Dask

def calculate():

    mem_usage_before = memory_profiler.memory_usage()[0]

    start_time = time.time()

    try:
        ddf = dd.read_csv("D:\codes/ITI/libariies/final project/globalterrorismdb_0718dist.csv", encoding='latin1', low_memory=False)
    except UnicodeDecodeError as e:
        print(f"Encoding error: {e}")


    mem_usage_after = memory_profiler.memory_usage()[0]

    end_time = time.time()

    print(f"Dask Memory Usage: {mem_usage_after - mem_usage_before} MB")
    print(f"Dask Time consuming: {end_time - start_time} seconds")


calculate()
```

```
Dask Memory Usage: 0.00390625 MB
Dask Elapsed Time: 0.12504029273986816 seconds
```

Conclusion

The dask took less time than pandas in reading operation and used less memory than pandas.

- Perform some operations with Pandas & Dask
- Use function (calculate()) to Compare the performance between Pandas & Dask.

- Perform some operations by pandas

```
# perform some operations with pandas

def calculate():

    start_time = time.time()

    Most_country = data.groupby('country_txt').size().nlargest(10).reset_index()

    most_city = data.groupby("city")['nkill','nwound'].sum().reset_index().head()

    years = data.groupby("iyear")['nkill','nwound'].sum().reset_index().head()

    num_killings = data.groupby("iyear")['nkill'].count().nlargest(5).reset_index(name='number of attack type')

    num_wounded = data.groupby("iyear")['nwound'].count().nlargest(5).reset_index(name='number of wounded')

    end_time = time.time()

    print(f"Pandas Time consuming: {end_time - start_time} seconds")


calculate()
```

```
Pandas Time consuming: 0.12469291687011719 seconds
```

- Perform same operations with Dask

```
# perform some operations with Dask

def calculate_mean_with_dask():

    ddf=dd.from_pandas(data,npartitions=3)

    start_time = time.time()

    Most_country = ddf.groupby('country_txt').size().nlargest(10).reset_index()

    most_city = ddf.groupby("city")['nkill','nwound'].sum().reset_index().head()

    years = ddf.groupby("iyear")['nkill','nwound'].sum().reset_index().head()

    num_killings = data.groupby("iyear")['nkill'].count().nlargest(5).reset_index(name='number of attack type')

    num_wounded = data.groupby("iyear")['nwound'].count().nlargest(5).reset_index(name='number of wounded')

    end_time = time.time()

    print(f"Dask Time consuming: {end_time - start_time} seconds")


calculate()
```

```
Dask Time consuming: 0.1174476146697998 seconds
```

Conclusion

The Dask take less time than pandas in performing the operations.