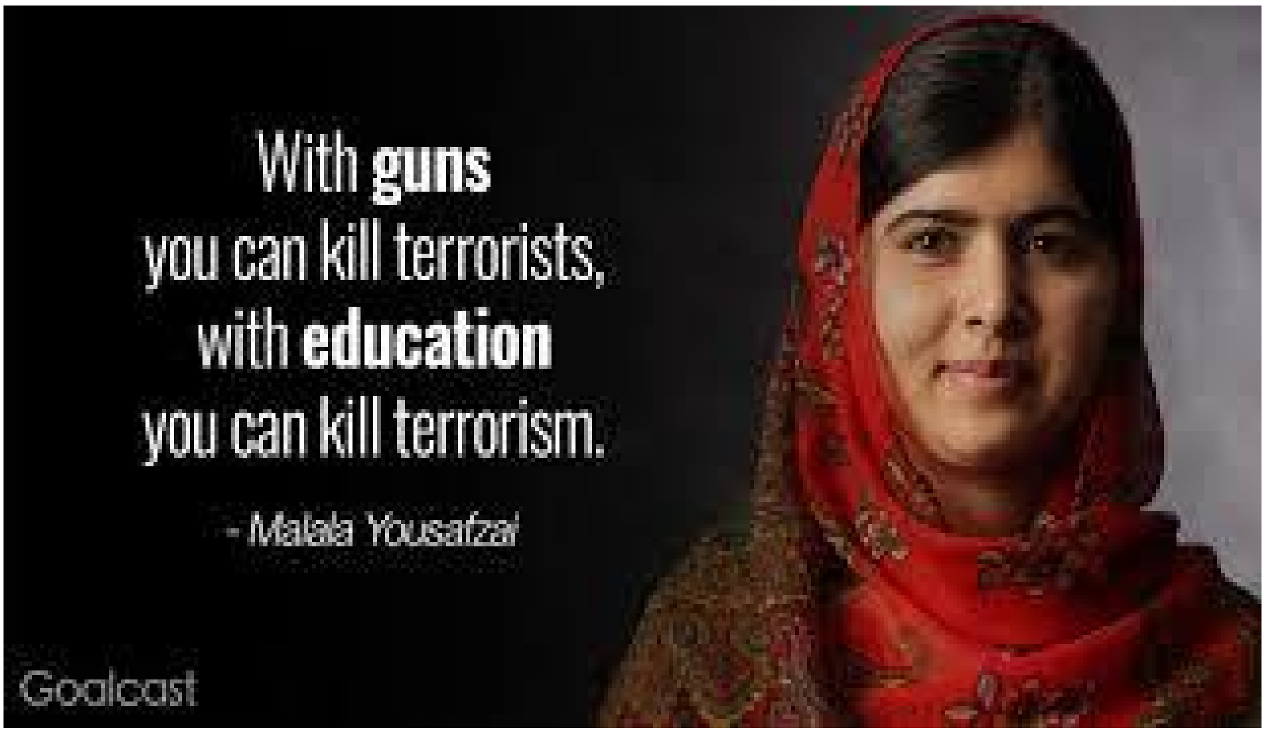
19CSE304- FOUNDATIONS OF DATA SCIENCE

GLOBAL TERRORISM

PART 1- GLOBAL TERRORISM ANALYSIS
PART 2- GLOBAL TERRORISM CLASSIFICATION



NAME	ROLL NUMBER
Bhimireddy Sai Gayathri Reddy	CB.EN.U4CSE190**12**
Harini S	CB.EN.U4CSE190**23**
Rithika Sri J	CB.EN.U4CSE190**25**

DATASET LINK : Global Terrorism Database

DATASET VARIABLES: https://start.umd.edu/gtd/downloads/Codebook.pdf

Abstract

The threatened or actual use of illegal force and violence by a non-state actor to attain a political, economic, religious, or social goal through fear, coercion, or intimidation is called Terrorism.

The Global Terrorism Database (GTD) is an open-source database ncluding 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.

Objective and motivation

In first part our project, we analyze the data available in the global terrorism database (GTD). This analysis will provide insight about terrorism occuring at different parts of the world and their impact on the society.
In the second part of our project, our main aim is to do a binary classification that classifies terror attacks as a 'successful mission' or a 'failed mission' depending upon the independent features provided in the global terrorism database.

REFERENCE

Model Building - https://en.wikipedia.org/wiki/Model_building

Mean Reversion - https://blog.quantinsti.com/mean-reversion-time-series/

Time series - https://builtin.com/data-science/time-series-python

Feature selection -

https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

GLOBAL TERRORISM ANALYSIS

GROUP - 4

CB.EN.U4CSE19012 - Gayathri Reddy

CB.EN.U4CSE19023 - Harini S

CB.EN.U4CSE19025 - Rithika Sri J

```
#Mounting drive
from google.colab import drive
drive.mount('/content/drive')
```

Install

```
!pip install pmdarima
```

```
Collecting pmdarima
  Downloading pmdarima-1.8.4-cp37-cp37m-
manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_24_x86_64.whl
(1.4 MB)
ent already satisfied: numpy>=1.19.3 in /usr/local/lib/python3.7/dist-
packages (from pmdarima) (1.19.5)
Requirement already satisfied: Cython!=0.29.18,>=0.29 in
/usr/local/lib/python3.7/dist-packages (from pmdarima) (0.29.24)
Requirement already satisfied: statsmodels!=0.12.0,>=0.11 in
/usr/local/lib/python3.7/dist-packages (from pmdarima) (0.13.1)
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.7/dist-packages (from pmdarima) (1.1.0)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
/usr/local/lib/python3.7/dist-packages (from pmdarima) (57.4.0)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.7/dist-packages (from pmdarima) (1.4.1)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.7/dist-packages (from pmdarima) (1.0.1)
Requirement already satisfied: urllib3 in
/usr/local/lib/python3.7/dist-packages (from pmdarima) (1.24.3)
Requirement already satisfied: pandas>=0.19 in
/usr/local/lib/python3.7/dist-packages (from pmdarima) (1.1.5)
Requirement already satisfied: pytz>=2017.2 in
/usr/local/lib/python3.7/dist-packages (from pandas>=0.19->pmdarima)
(2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=0.19->pmdarima)
(2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3-
>pandas>=0.19->pmdarima) (1.15.0)
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.22-
>pmdarima) (3.0.0)
Requirement already satisfied: patsy>=0.5.2 in
/usr/local/lib/python3.7/dist-packages (from statsmodels!
=0.12.0,>=0.11->pmdarima) (0.5.2)
Installing collected packages: pmdarima
Successfully installed pmdarima-1.8.4
```

```
pip install feature-engine
```

```
Requirement already satisfied: feature-engine in
/usr/local/lib/python3.7/dist-packages (1.1.2)
Requirement already satisfied: numpy>=1.18.2 in
/usr/local/lib/python3.7/dist-packages (from feature-engine) (1.19.5)
Requirement already satisfied: scipy>=1.4.1 in
/usr/local/lib/python3.7/dist-packages (from feature-engine) (1.4.1)
Requirement already satisfied: pandas>=1.0.3 in
/usr/local/lib/python3.7/dist-packages (from feature-engine) (1.1.5)
Requirement already satisfied: statsmodels>=0.11.1 in
/usr/local/lib/python3.7/dist-packages (from feature-engine) (0.13.1)
Requirement already satisfied: scikit-learn>=0.22.2 in
/usr/local/lib/python3.7/dist-packages (from feature-engine) (1.0.1)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=1.0.3->feature-
engine) (2.8.2)
Requirement already satisfied: pytz>=2017.2 in
/usr/local/lib/python3.7/dist-packages (from pandas>=1.0.3->feature-
engine) (2018.9)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3-
>pandas>=1.0.3->feature-engine) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.22.2-
>feature-engine) (3.0.0)
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.22.2-
>feature-engine) (1.1.0)
Requirement already satisfied: patsy>=0.5.2 in
/usr/local/lib/python3.7/dist-packages (from statsmodels>=0.11.1-
>feature-engine) (0.5.2)
```

INTRODUCTION

The threatened or actual use of illegal force and violence by a non-state actor to attain a political, economic, religious, or social goal through fear, coercion, or intimidation is called Terrorism.

The main objective of this case study is to analyze the data available in the global terrorism database (GTD). 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.

OVERVIEW:

A detailed analysis on how terrorism spread around the world and the impact caused.

OBJECTIVE:

- To predict future Attacks based on past successful attacks using different models & do model comparison
- To forecast the future trend of Global Terriorism - Number of Attacks using Time Series Analysis

```
#importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import folium
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from datetime import date, timedelta
from pmdarima.arima import auto_arima
from math import sqrt
from sklearn.metrics import mean_squared_error
from feature_engine.creation import CyclicalTransformer
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Understanding the dataset

```
#loading the dataset
GT = pd.read_csv('/content/drive/MyDrive/Global_Terrorism.csv',
encoding = "ISO-8859-1")
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/
interactiveshell.py:2718: DtypeWarning: Columns
(4,6,31,33,61,62,63,76,79,90,92,94,96,114,115,121) have mixed
types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
#num of columns and rows
GT.shape
```

```
(181691, 135)
```

```
GT.head()
```

```
        eventid  iyear  imonth  iday  ... INT_IDEO  INT_MISC INT_ANY
related
0  197000000001   1970       7     2  ...        0         0       0
NaN
1  197000000002   1970       0     0  ...        1         1       1
NaN
2  197001000001   1970       1     0  ...       -9         1       1
NaN
3  197001000002   1970       1     0  ...       -9         1       1
NaN
4  197001000003   1970       1     0  ...       -9         1       1
NaN

[5 rows x 135 columns]
```

```
#Checking on the list of columns
print(GT.columns.to_list())
```

```
['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']
```

DATA PREPROCESSING

```
GT = GT.loc[(GT.doubtterr == 0) & (GT.nkill >=0)]
GT.shape
```

```
(132137, 135)
```

```
#renaming the columns
GT.rename(columns={'eventid':'ID','iyear':'Year','imonth':'Month','ida
y':'Day','country_txt':'Country','provstate':'state',
'region_txt':'Region',

'attacktype1_txt':'AttackType','nkill':'Killed','target1':'Target',
'nwound':'Wounded','summary':'Summary',
                   'gname':'Group','targtype1_txt':'Target_type',
'weaptype1_txt':'Weapon_type','motive':'Motive'},inplace=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4308:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  errors=errors,
```

```
#Extracting only important columns
GT =
GT[['ID','Year','Month','Day','extended','Country','state','city',
'Region','latitude','longitude','AttackType','Killed','Wounded','Targe
t','Group',
        'success','crit1','crit2','crit3','multiple',
'Target_type','Weapon_type', 'vicinity',
'specificity','suicide','propextent_txt','ishostkid','INT_ANY']]
```

```
#Random 5 rows
GT.sample(3)
```

```
                  ID  Year  ...  ishostkid  INT_ANY
157936  201601200040  2016  ...        0.0        0
163164  201606010039  2016  ...        0.0        1
147828  201505080012  2015  ...        0.0        1

[3 rows x 29 columns]
```

```
#num of columns and rows
GT.shape
```

```
(132137, 29)
```

```
GT.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 132137 entries, 0 to 181690
Data columns (total 29 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   ID              132137 non-null  int64  
 1   Year            132137 non-null  int64  
 2   Month           132137 non-null  int64  
 3   Day             132137 non-null  int64  
 4   extended        132137 non-null  int64  
 5   Country         132137 non-null  object 
 6   state           131934 non-null  object 
 7   city            131743 non-null  object 
 8   Region          132137 non-null  object 
 9   latitude        129568 non-null  float64
 10  longitude       129568 non-null  float64
 11  AttackType      132137 non-null  object 
 12  Killed          132137 non-null  float64
 13  Wounded         127588 non-null  float64
 14  Target          131710 non-null  object 
 15  Group           132137 non-null  object 
 16  success         132137 non-null  int64  
 17  crit1           132137 non-null  int64  
 18  crit2           132137 non-null  int64  
 19  crit3           132137 non-null  int64  
 20  multiple        132136 non-null  float64
 21  Target_type     132137 non-null  object 
 22  Weapon_type     132137 non-null  object 
 23  vicinity        132137 non-null  int64  
 24  specificity     132132 non-null  float64
 25  suicide         132137 non-null  int64  
 26  propextent_txt  50322 non-null   object 
 27  ishostkid       131989 non-null  float64
 28  INT_ANY         132137 non-null  int64  
dtypes: float64(7), int64(12), object(10)
memory usage: 30.2+ MB
```

```
#total null values in each column
GT.isnull().sum()
```

```
ID                     0
Year                   0
Month                  0
Day                    0
extended               0
Country                0
state                203
```

```
city                394
Region                0
latitude           2569
longitude          2569
AttackType            0
Killed                0
Wounded            4549
Target              427
Group                 0
success               0
crit1                 0
crit2                 0
crit3                 0
multiple              1
Target_type           0
Weapon_type           0
vicinity              0
specificity           5
suicide               0
propextent_txt    81815
ishostkid           148
INT_ANY               0
dtype: int64
```

```python
#Handling Nan in text fields. Assigning Nan as 'Unknown'
GT['Target'].fillna('Unknown', inplace= True)
GT['city'].fillna('Unknown', inplace= True)
GT['state'].fillna('Unknown', inplace= True)
```

```python
#total null values in each column
GT.isnull().sum()
```

```
ID                    0
Year                  0
Month                 0
Day                   0
extended              0
Country               0
state                 0
city                  0
Region                0
latitude           2569
longitude          2569
AttackType            0
Killed                0
Wounded            4549
Target                0
Group                 0
success               0
crit1                 0
crit2                 0
```

```
crit3                  0
multiple               1
Target_type            0
Weapon_type            0
vicinity               0
specificity            5
suicide                0
propextent_txt     81815
ishostkid            148
INT_ANY                0
dtype: int64
```

```python
GT = GT.loc[(GT.ishostkid != -9) & (GT.INT_ANY !=-9) ]
```

```python
GT=GT.replace('Unknown', np.nan)
GT=GT.replace('Other',np.nan)
```

```python
GT.dropna(inplace=True)
GT.shape
```

```
(15691, 29)
```

```python
#total null values in each column
GT.isnull().sum()
```

```
ID                 0
Year               0
Month              0
Day                0
extended           0
Country            0
state              0
city               0
Region             0
latitude           0
longitude          0
AttackType         0
Killed             0
Wounded            0
Target             0
Group              0
success            0
crit1              0
crit2              0
crit3              0
multiple           0
Target_type        0
Weapon_type        0
vicinity           0
specificity        0
suicide            0
```

```
propextent_txt    0
ishostkid         0
INT_ANY           0
dtype: int64
```

```
GT.success.value_counts()
```

```
1    15262
0      429
Name: success, dtype: int64
```

```
GT['Day'].value_counts()
```

```
1     628
14    606
21    584
11    576
12    572
2     557
15    554
7     553
10    546
9     541
25    539
13    537
16    532
5     526
19    520
20    515
3     513
27    513
22    485
18    479
4     477
8     476
24    466
26    463
6     453
23    450
17    450
29    446
28    440
30    413
31    239
0      42
Name: Day, dtype: int64
```

```
'''
There was data inconsistency in GT['Day']. It had a inconsistent value
0. The actual value must range between 1 - 31.
So removing the rows with GT['Day']==0
```

```
'''
GT.drop(GT.loc[GT.Day==0].index, inplace= True)

#Creating Datetime feature with relevant features
GT['Date'] = pd.to_datetime(GT[['Year','Month','Day']], errors =
'coerce')
#GT.drop(['Day','Month'], axis=1, inplace= True)

#Replacing a large value in GT['Weapon_type']
GT['Weapon_type'].replace({"Vehicle (not to include vehicle-borne
explosives, i.e., car or truck bombs)":"Bombless Vehicle"},inplace
=True)

#Adding new column 'Casualities'
GT['Casualities'] = GT['Wounded'] + GT['Killed']

GT.shape
```

```
(15649, 31)
```

```
GT.dtypes
```

```
ID                           int64
Year                         int64
Month                        int64
Day                          int64
extended                     int64
Country                     object
state                       object
city                        object
Region                      object
latitude                   float64
longitude                  float64
AttackType                  object
Killed                     float64
Wounded                    float64
Target                      object
Group                       object
success                      int64
crit1                        int64
crit2                        int64
crit3                        int64
multiple                   float64
Target_type                 object
Weapon_type                 object
vicinity                     int64
specificity                float64
suicide                      int64
propextent_txt              object
ishostkid                  float64
INT_ANY                      int64
Date                datetime64[ns]
```

```
Casualities              float64
dtype: object
```

```
GT.sample(3)
```

```
                 ID  Year  extended  ... INT_ANY       Date
Casualities
6426   197806070003  1978         0  ...       1 1978-06-07
0.0
23733  198411130012  1984         0  ...       0 1984-11-13
0.0
6355   197805200006  1978         0  ...       1 1978-05-20
0.0

[3 rows x 29 columns]
```

UNDERSTANDING DATA - ANALYSIS

```
print("Country with the most
attacks:",GT['Country'].value_counts().idxmax())
print("Country with the least
attacks:",GT['Country'].value_counts().idxmin())
print("City with the most
attacks:",GT['city'].value_counts().index[0])
print("Region with the most
attacks:",GT['Region'].value_counts().idxmax())
print("Year with the most
attacks:",GT['Year'].value_counts().idxmax())
print("Year with the least
attacks:",GT['Year'].value_counts().idxmin())
print("Group with the most
attacks:",GT['Group'].value_counts().index[1])
print("Most Attack Types:",GT['AttackType'].value_counts().idxmax())
```

```
Country with the most attacks: Iraq
Country with the least attacks: South Yemen
City with the most attacks: Baghdad
Region with the most attacks: Middle East & North Africa
Year with the most attacks: 2014
Year with the least attacks: 1972
Group with the most attacks: Taliban
Most Attack Types: Bombing/Explosion
```

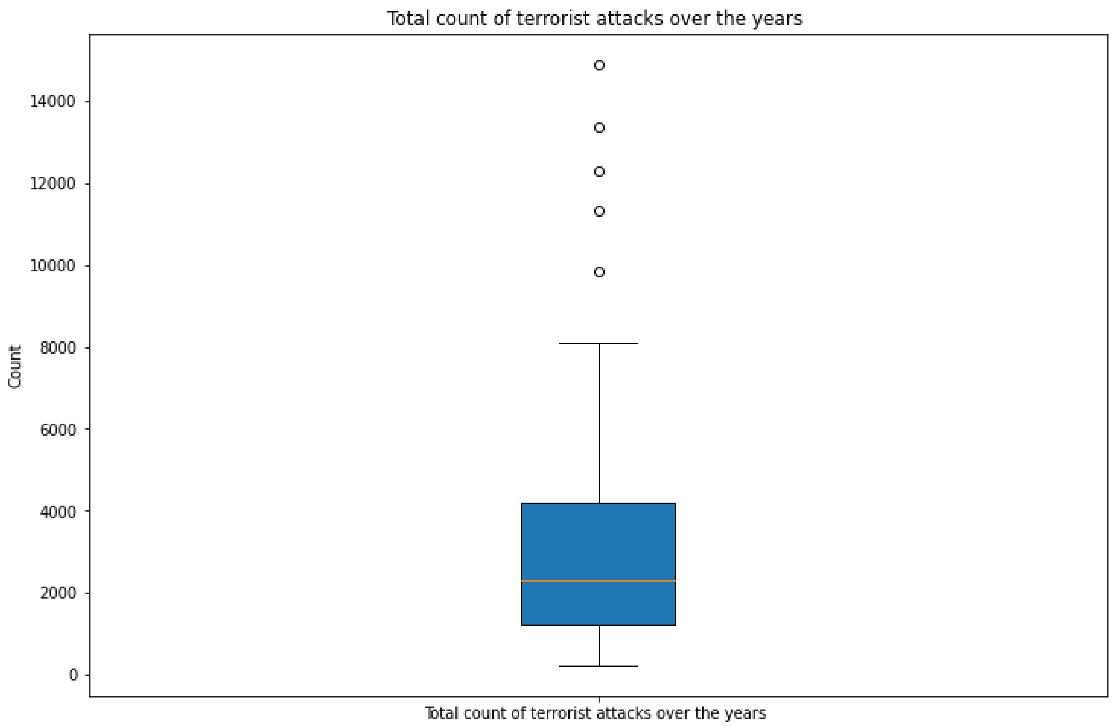
```
#Statistical info on numerical data
GT.describe()[['Killed', 'Wounded', 'Casualities']].round(2)
```

```
          Killed    Wounded  Casualities
count  160284.00  160284.00    160284.00
mean        2.10       3.20         5.30
std         9.76      36.48        42.58
min         0.00       0.00         0.00
```

```
25%         0.00       0.00         0.00
50%         0.00       0.00         1.00
75%         2.00       2.00         4.00
max      1384.00    8191.00      9574.00
```

INFERENCE FROM DESCRIBE(): Need not rescale 'Killed', 'Wounded' and 'Casualities'.

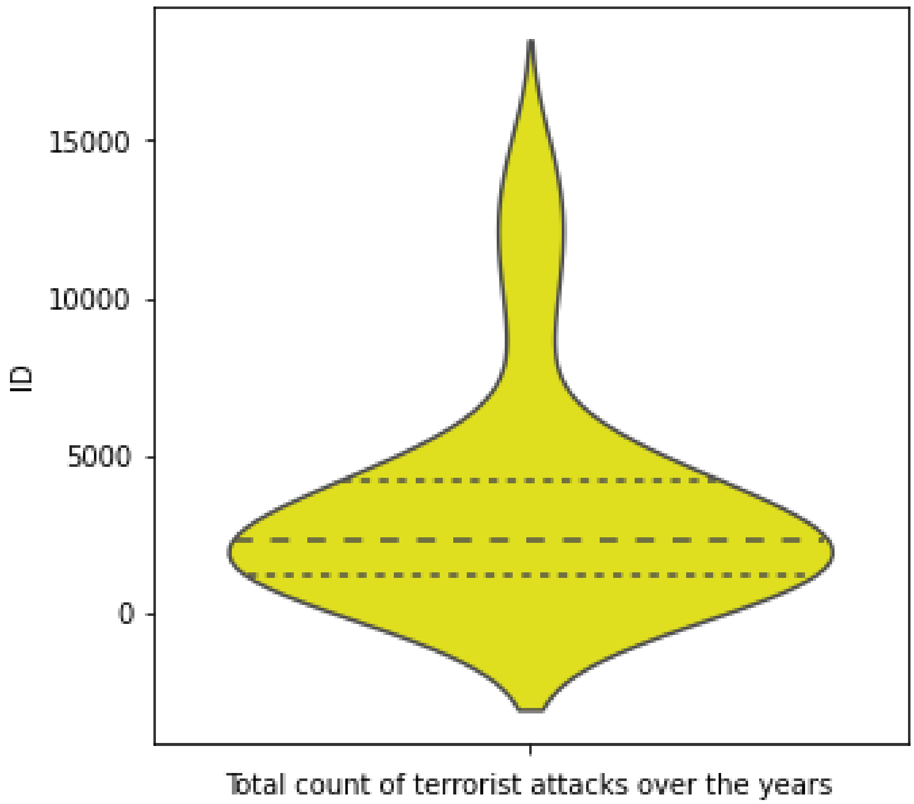
```
#Box plot for Terrorist attack over the years
year_group = GT.groupby('Year',as_index = False)['ID'].count()
plt.figure(figsize=(12,8))
plt.boxplot(year_group['ID'],patch_artist = True)
plt.title('Total count of terrorist attacks over the years ')
plt.xticks([1],['Total count of terrorist attacks over the years '])
plt.ylabel('Count')
plt.show()
```



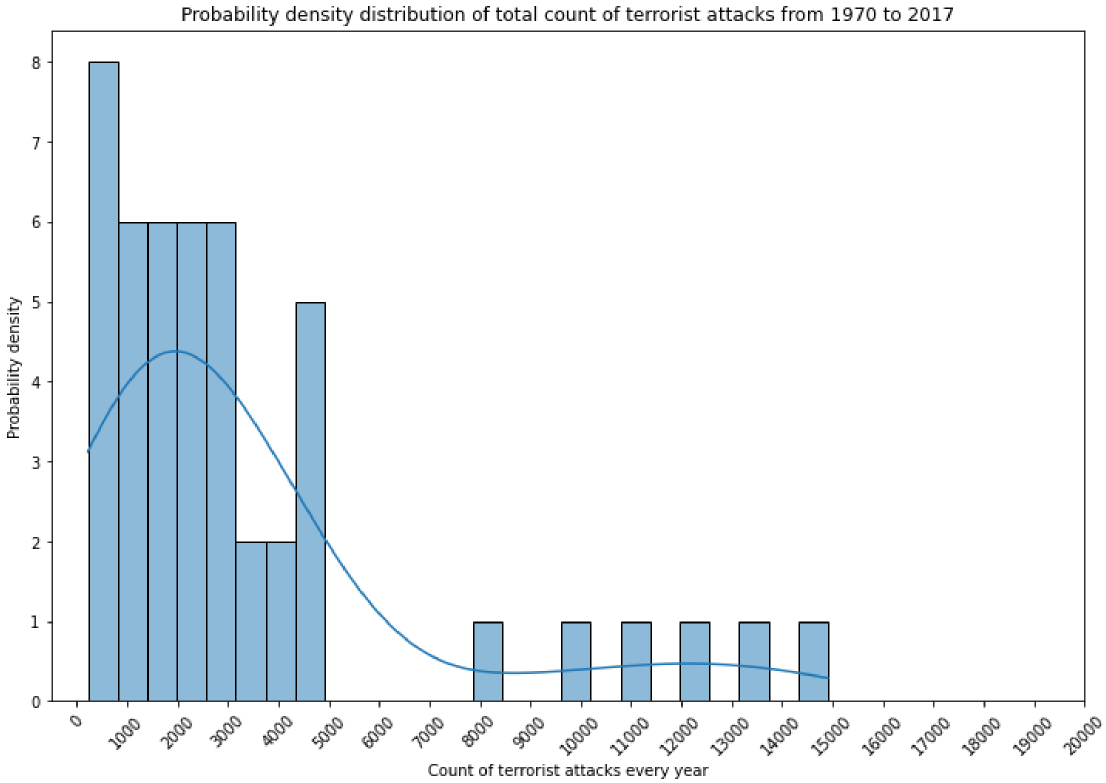

```
print('Statistics on total count of terrorist attacks every year
from',year_group['Year'].min(), 'to', year_group['Year'].max(),':')
print('\t Total: ',year_group['ID'].sum())
print('\t Average: ',round(year_group['ID'].mean()))
print('\t Maximum: ',year_group['ID'].max())
print('\t Minimum: ',year_group['ID'].min())
```

```
Statistics on total count of terrorist attacks every year from 1970 to
2017 :
	 Total:  160284
	 Average:  3410
	 Maximum:  14905
	 Minimum:  219
```

```
#Violin plot for Terrorist attack over the years
year_group = GT.groupby('Year',as_index = False)['ID'].count()
plt.figure(figsize = (5,5))
plt.xlabel('Total count of terrorist attacks over the years')
sns.violinplot(y= year_group['ID'], inner= 'quartile', color=
'yellow')
plt.show()
```



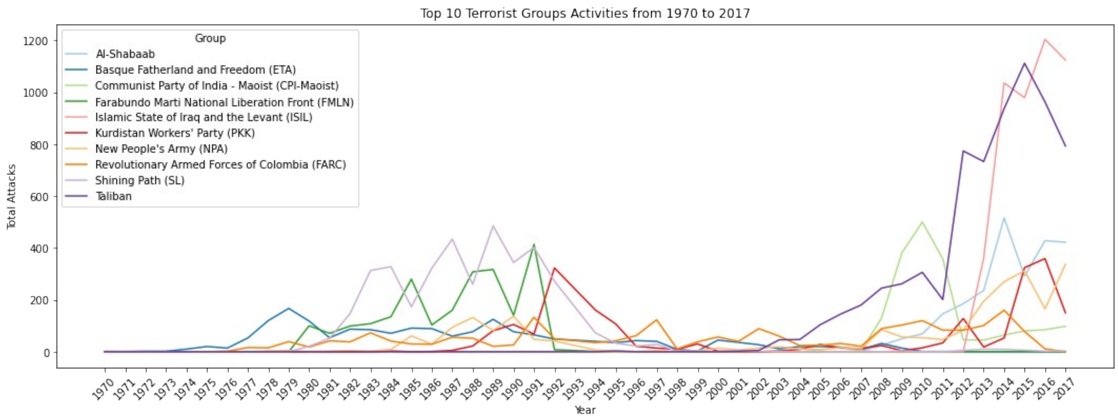

```
plt.figure(figsize=(12,8))
ax=sns.histplot(year_group['ID'], bins=25, kde= True)
plt.xlabel('Count of terrorist attacks every year')
plt.ylabel('Probability density')
plt.xticks(range(0,20001,1000),rotation =45)
plt.title('Probability density distribution of total count of
terrorist attacks from 1970 to 2017')
plt.show()
```



We can conclude that the count of terrorist attack every year is positively skewed.

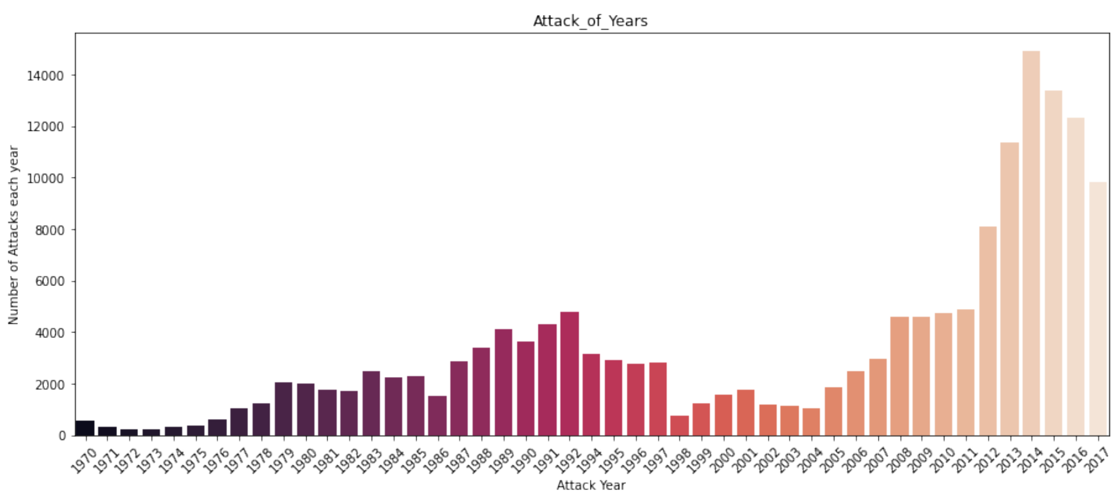
DATA VISUALIZATION

```
#Top 10 Terrorist Groups Activities from 1970 to 2017
groups_10 = GT[GT.Group.isin(GT.Group.value_counts()[1:11].index)]
pd.crosstab(groups_10.Year,
groups_10.Group).plot(color=sns.color_palette('Paired', 10))
fig=plt.gcf()
fig.set_size_inches(18,6)
plt.xticks(range(1970, 2018, 1), rotation= 45)
plt.ylabel('Total Attacks')
plt.title('Top 10 Terrorist Groups Activities from 1970 to 2017')
plt.show()
```



In the recent years, the Taliban and ISIL are more active then the past decades.

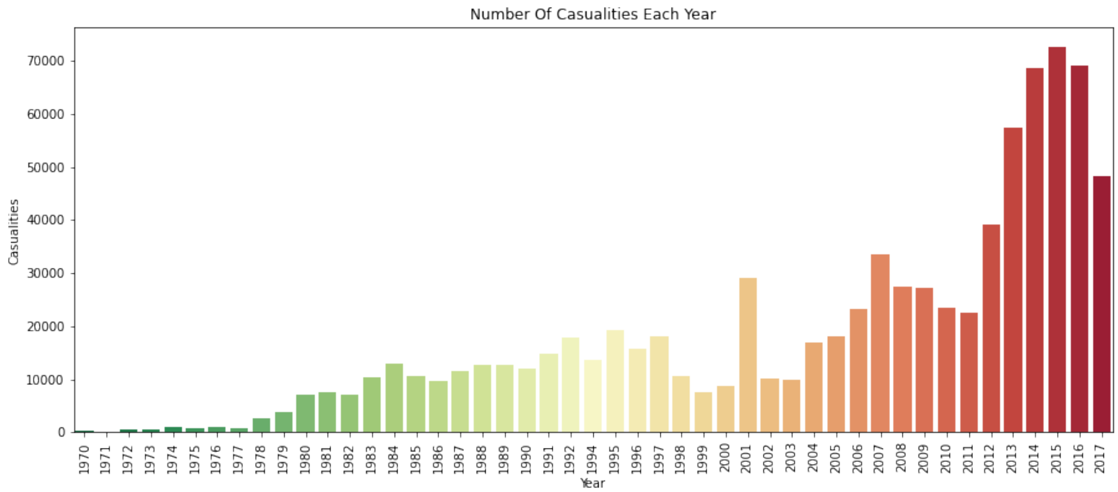
```
#Count of attacks over the years
x_year = GT['Year'].unique()
y_count_years = GT['Year'].value_counts().sort_index()
plt.subplots(figsize=(15,6))
sns.barplot(x = x_year, y = y_count_years, palette = 'rocket')
plt.xticks(rotation = 45)
plt.xlabel('Attack Year')
plt.ylabel('Number of Attacks each year')
plt.title('Attack_of_Years')
plt.show()
```



As we can see, overally, there has been a rise in the number of terrorist attacks over the years. The highest peak was at 2014 and lowest was at 1972. Seems there is an steady decrease after hitting the highest from 2014 to 2017.

```
plt.subplots(figsize=(15,6))
year_cas =
GT.groupby('Year').Casualities.sum().to_frame().reset_index()
year_cas.columns = ['Year','Casualities']
sns.barplot(x=year_cas.Year, y=year_cas.Casualities,
```

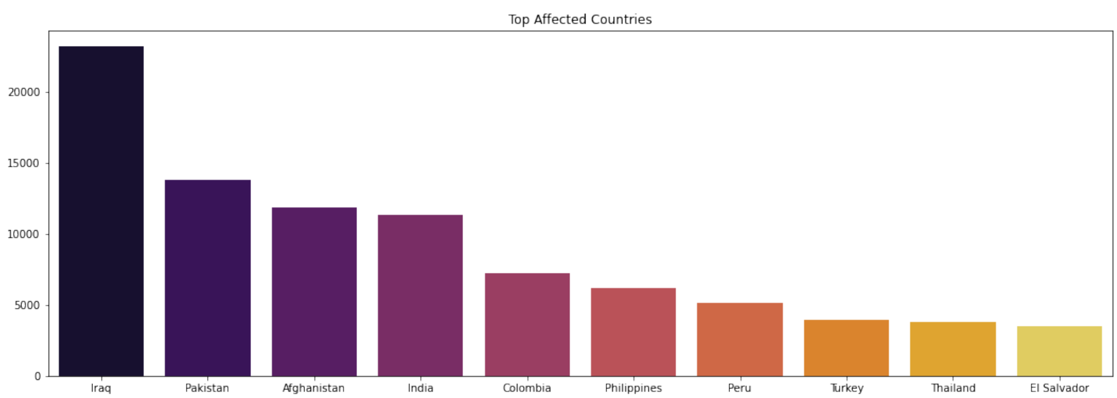
```
palette='RdYlGn_r')
plt.xticks(rotation=90)
plt.title('Number Of Casualities Each Year')
plt.show()
```



Inference:

We can see that the number of casualities is high between years 2014 and 2016 and low between the years 1970-1977

```
plt.subplots(figsize=(18,6))
sns.barplot(x= GT['Country'].value_counts()[:10].index,y=
GT['Country'].value_counts()[:10].values,palette='inferno')
plt.title('Top Affected Countries')
plt.show()
```

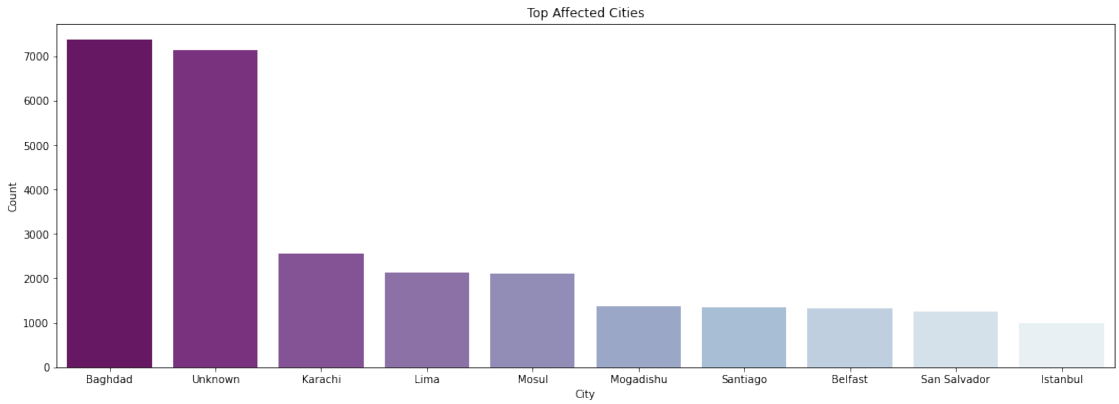


Inference:

The most effected country from Terrorism is Iraq when El Salvador is effected the least.

```
plt.subplots(figsize=(18,6))
sns.barplot(x= GT['city'].value_counts()[:10].index,y=
GT['city'].value_counts()[:10].values,palette='BuPu_r')
plt.title('Top Affected Cities')
```

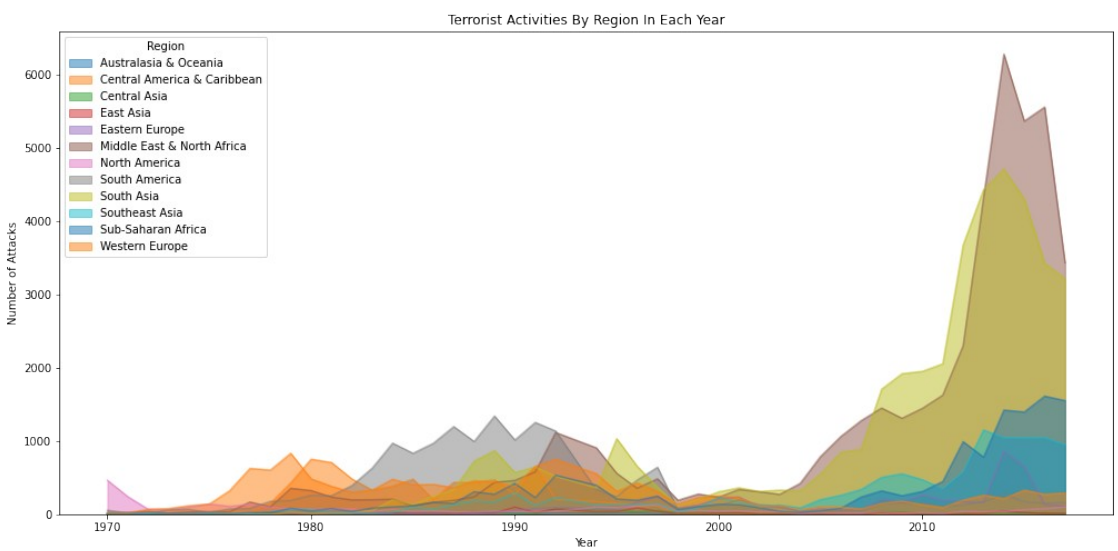
```
plt.xlabel('City')
plt.ylabel('Count')
plt.show()
```



Inference:

The most affected city from terrorism is Baghbad and the least affested city is Istanbul

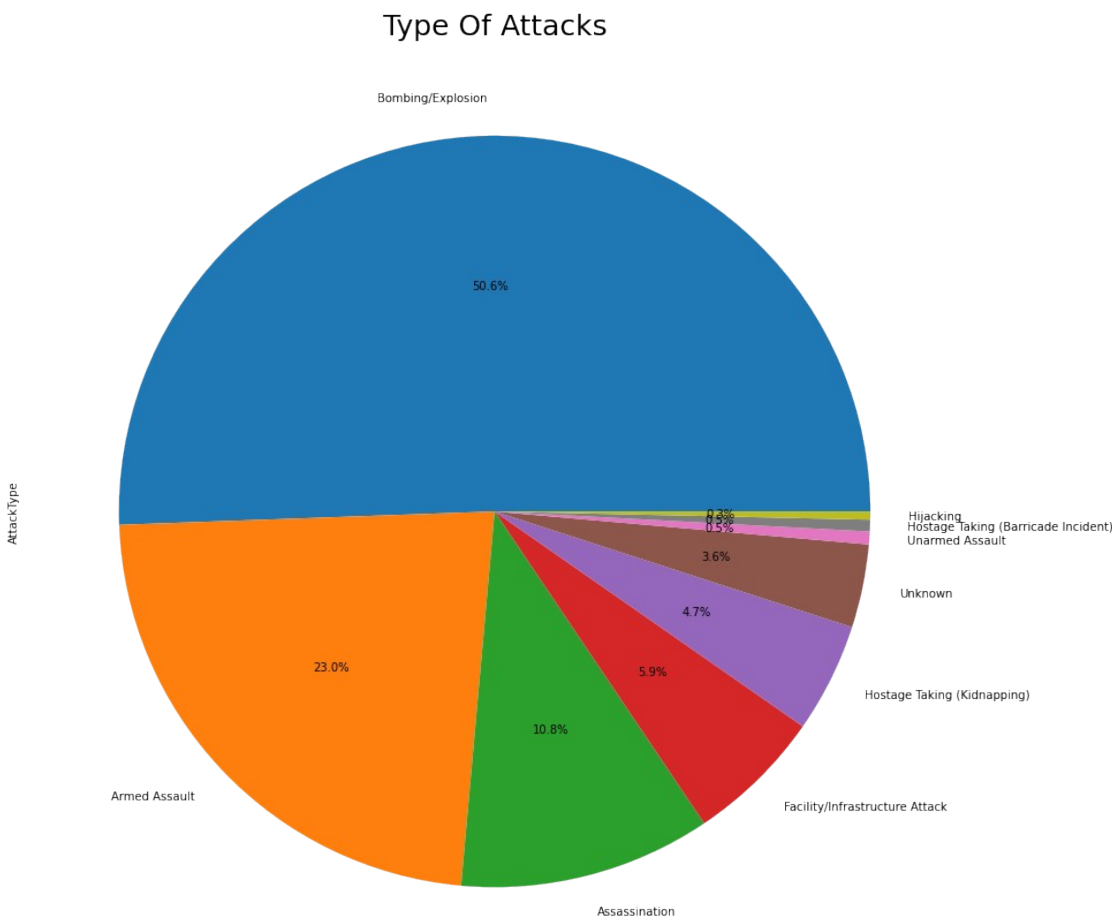
```
pd.crosstab(GT.Year,
GT.Region).plot(kind='area',stacked=False,figsize=(17,8))
plt.title('Terrorist Activities By Region In Each Year')
plt.ylabel('Number of Attacks')
plt.xlabel("Year")
plt.show()
```



Inference:

From The above graph from the year 2010 there is graduall increase of Terrorist activities in the East Asia Region,and also in South Asia Region.in between years 1980-2000 there are high number of Terrorist Activities in South America Region.

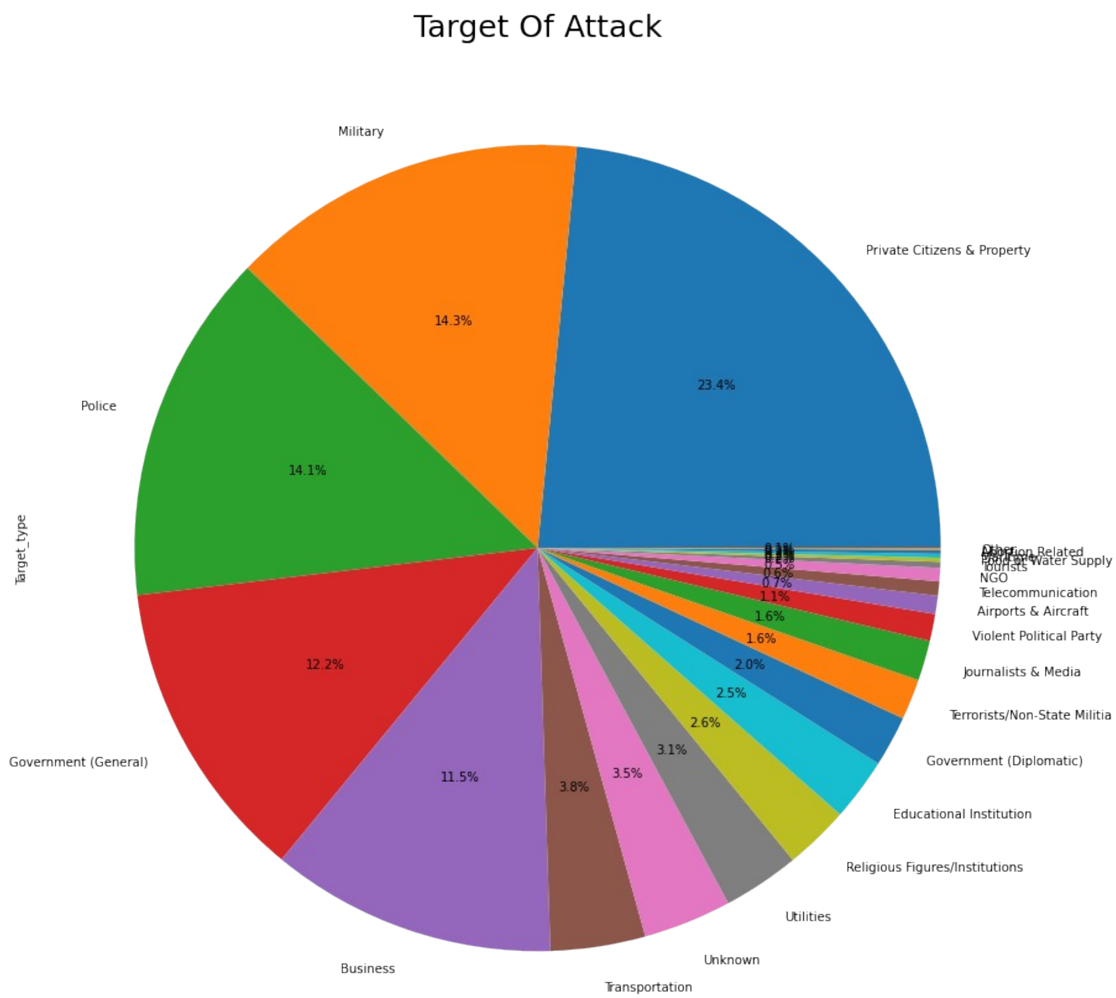
```python
plt.figure(figsize=(15,15))
GT['AttackType'].value_counts().plot.pie(autopct="%1.1f%%")
plt.title('Type Of Attacks', fontsize=25)
plt.show()
```



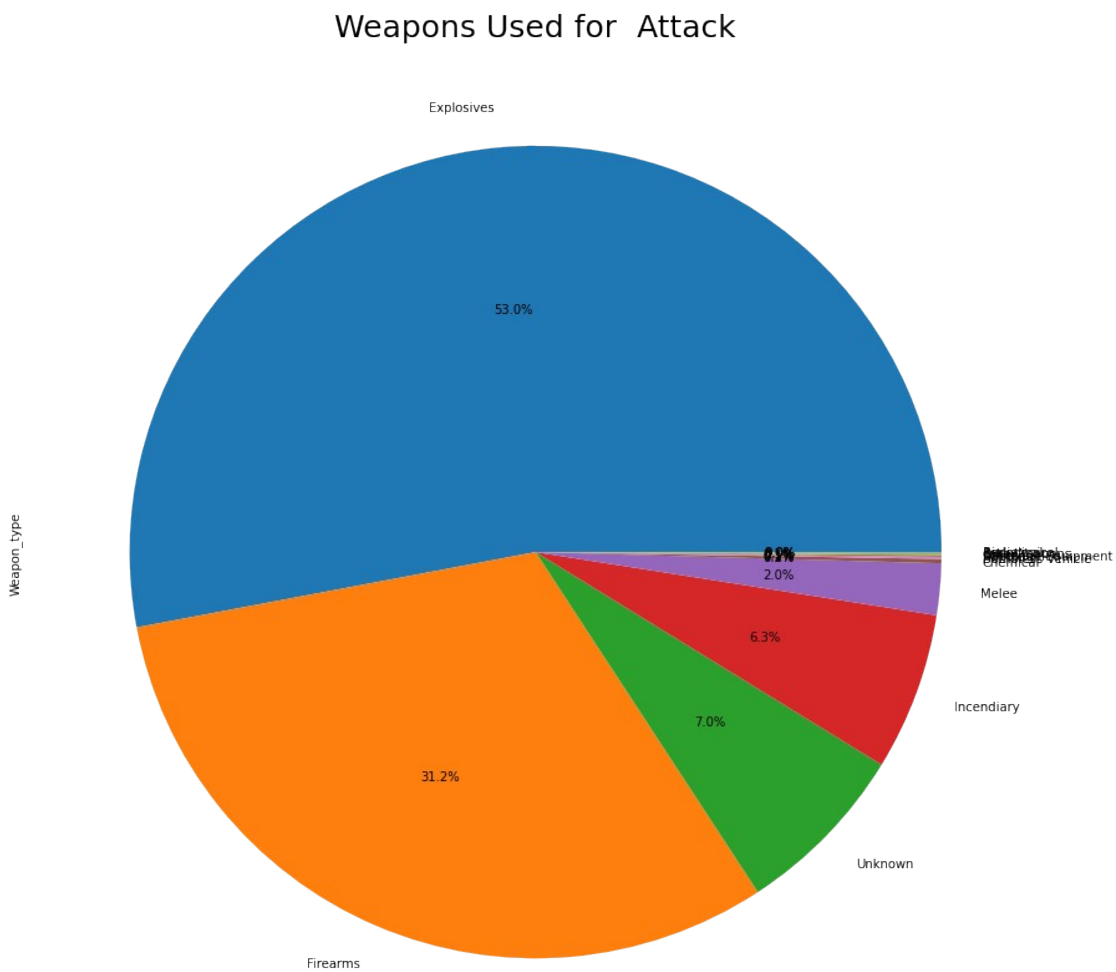
Inference:

Bombing or explosion is the most often used for attacking. Where Armed Assault is the second. Hijacking,Hostage Taking and Unarmed Assault are the least used type of Attacks.

```python
plt.figure(figsize=(15,15))
GT['Target_type'].value_counts().plot.pie(autopct="%1.1f%%")
plt.title('Target Of Attack',fontsize=25)
plt.show()
```



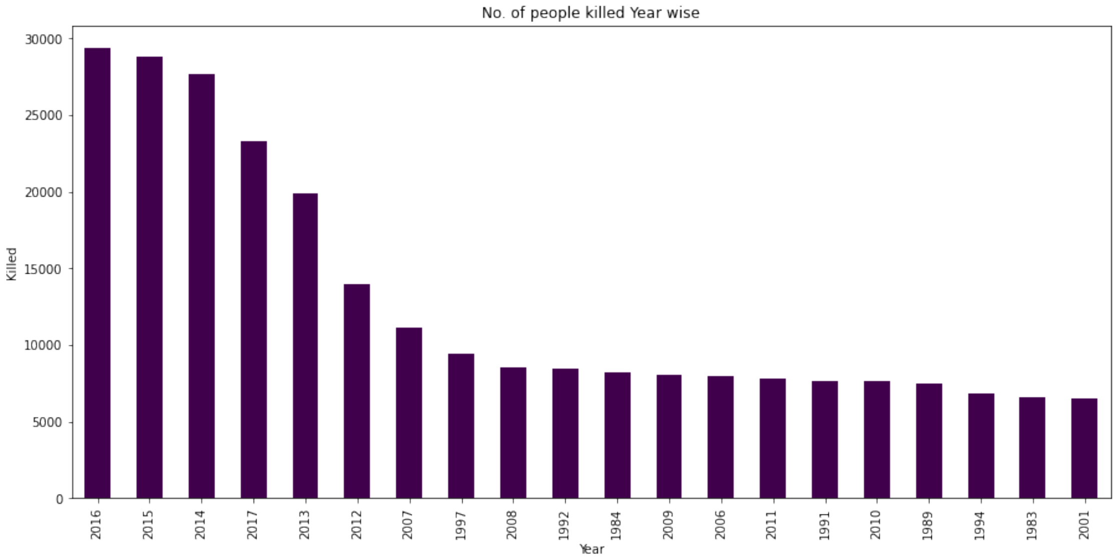

```
plt.figure(figsize=(15,15))
GT['Weapon_type'].value_counts().plot.pie(autopct="%1.1f%%")
plt.title('Weapons Used for  Attack',fontsize=25)
plt.show()
```



Inference:

For the majority of Attacks Explosives are used as weapons and Firearms are in second place.

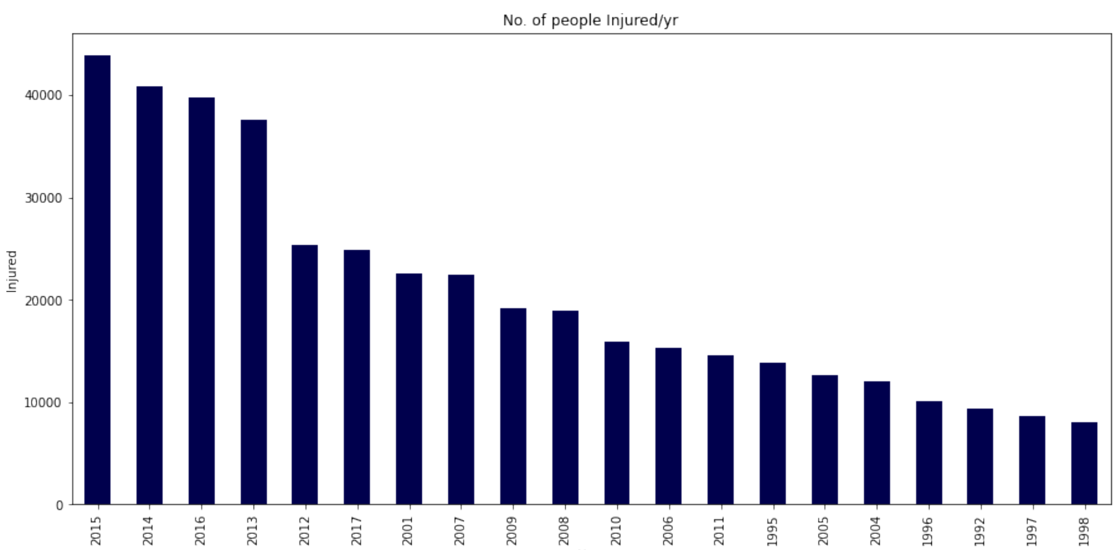
```
plt.figure(figsize = (15,7))
GT.groupby(['Year'])['Killed'].sum().sort_values(ascending =
False).head(20).plot(kind = 'bar', colormap = 'PRGn')
plt.xticks(rotation=90)
plt.title('No. of people killed Year wise')
plt.ylabel("Killed")
plt.show()
```



Inference:

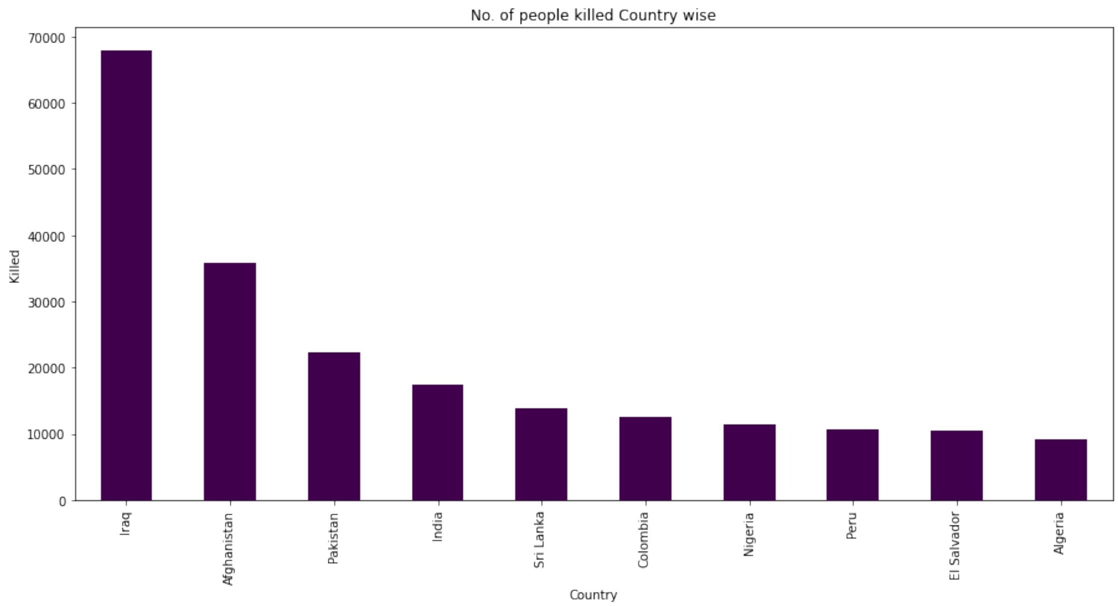
In the year 2016 major number of people are killed, and from 2012 there was a drastic increase of deadths.

```
plt.figure(figsize = (15,7))
GT.groupby(['Year'])['Wounded'].sum().sort_values(ascending =
False).head(20).plot(kind = 'bar', colormap = 'seismic')
plt.xticks(rotation=90)
plt.title('No. of people Injured/yr')
plt.ylabel("Injured")
plt.show()
```




```
plt.figure(figsize = (15,7))
GT.groupby(['Country'])['Killed'].sum().sort_values(ascending =
False).head(10).plot(kind = 'bar', colormap = 'PRGn')
```

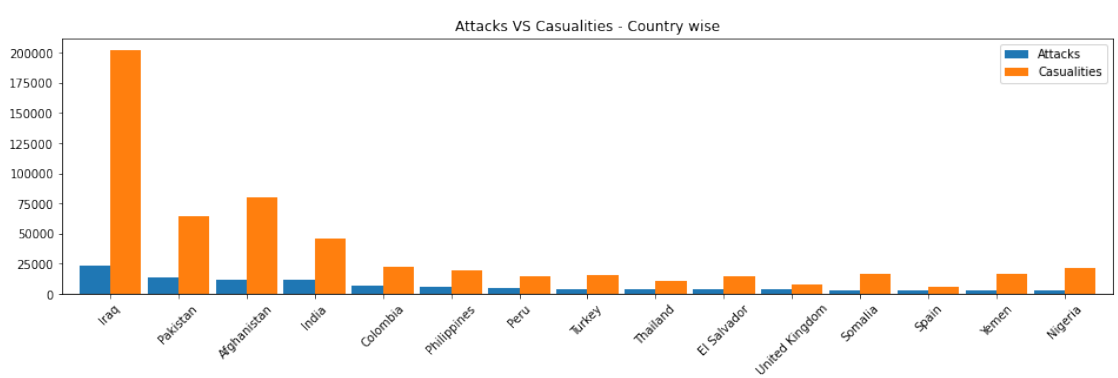
```
plt.title('No. of people killed Country wise')
plt.ylabel("Killed")
plt.show()
```



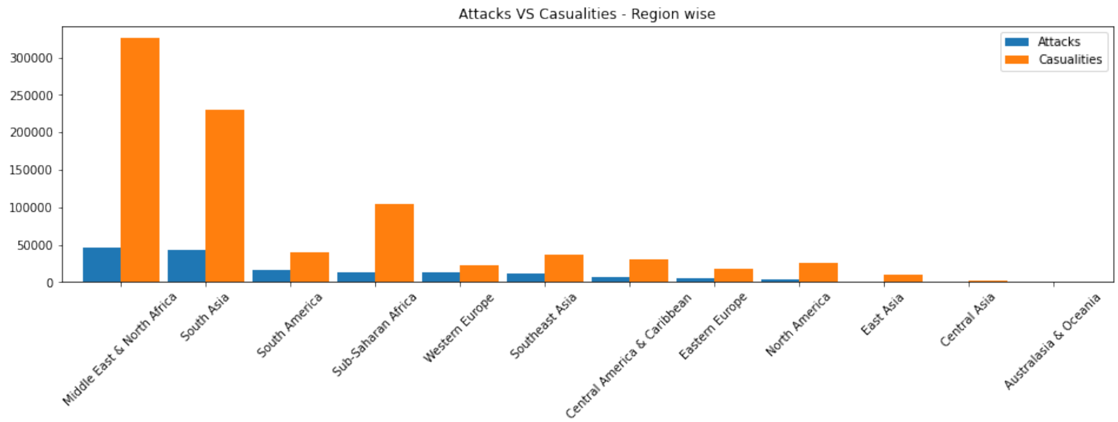

```
print(GT['Country'].unique().shape)
print(GT['Region'].unique().shape)
```

```
(202,)
(12,)
```

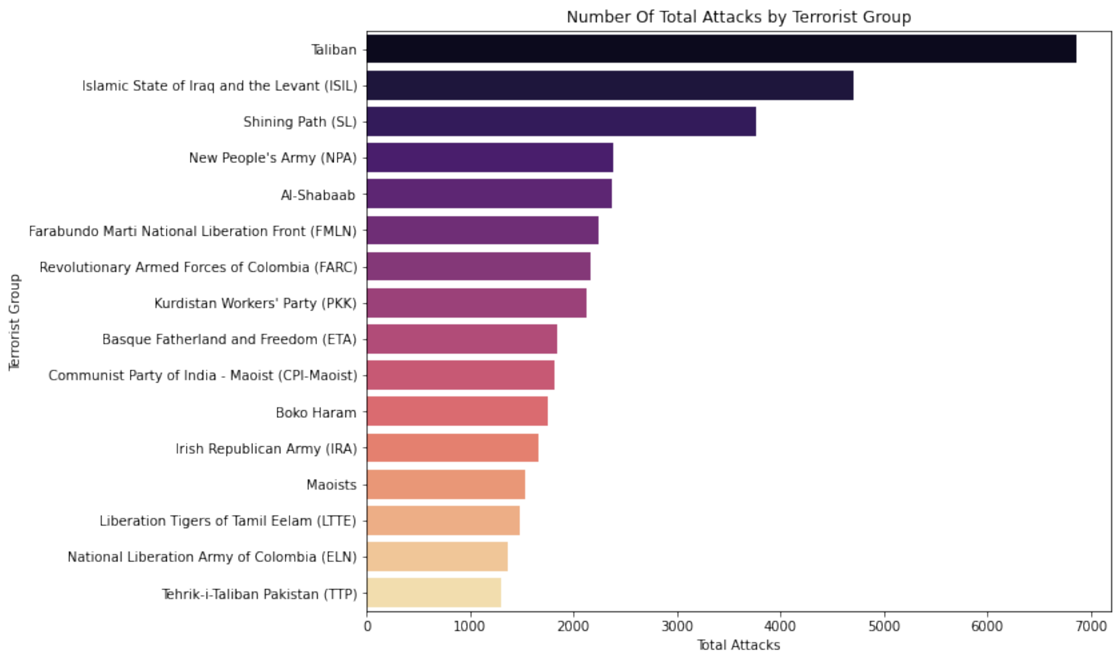
```
count_terror = GT['Country'].value_counts()[:15].to_frame()
count_terror.columns=['Attacks']
count_kill=GT.groupby ('Country')['Casualities'].sum().to_frame()
count_terror.merge(count_kill,left_index = True,right_index
=True,how='left').plot.bar(width=0.9)
plt.xticks(rotation=45)
fig=plt.gcf()
plt.title("Attacks VS Casualities - Country wise")
fig.set_size_inches(16,4)
plt.show()
```



```
count_terror = GT['Region'].value_counts()[:15].to_frame()
count_terror.columns=['Attacks']
count_kill=GT.groupby ('Region')['Casualities'].sum().to_frame()
count_terror.merge(count_kill,left_index = True,right_index
=True,how='left').plot.bar(width=0.9)
fig=plt.gcf()
plt.xticks(rotation=45)
plt.title("Attacks VS Casualities - Region wise")
fig.set_size_inches(16,4)
plt.show()
```

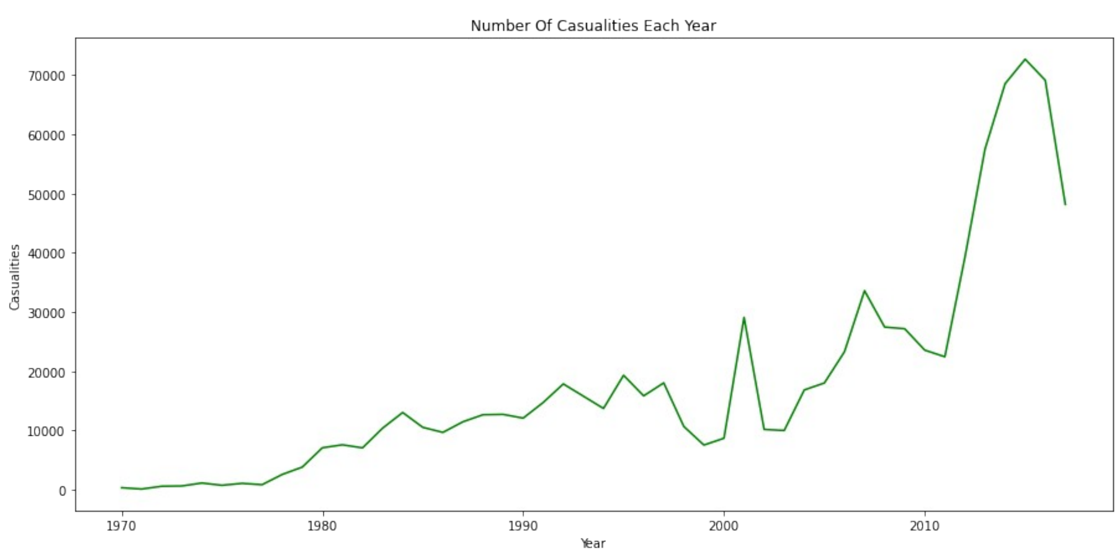



```
group_attacks =
GT.Group.value_counts().to_frame().drop('Unknown').reset_index()[:16]
group_attacks.columns = ['Terrorist Group', 'Total Attacks']
plt.subplots(figsize=(10,8))
sns.barplot(y= group_attacks['Terrorist Group'], x=
group_attacks['Total Attacks'], palette='magma')
plt.title('Number Of Total Attacks by Terrorist Group')
plt.show()
```




```
plt.subplots(figsize=(15,7))
year_casual =
GT.groupby('Year').Casualities.sum().to_frame().reset_index()
year_casual.columns = ['Year','Casualities']
plt.title('Number Of Casualities Each Year')
sns.lineplot(x='Year', y='Casualities', data=year_casual,color="g")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5ba1f7fc10>
```



=>Year and Casualities are ***independent***

```
loca = GT[['latitude','longitude']][:8000]
coun = GT['Country'][:8000]
cit = GT['city'][:8000]
```

```
kill = GT['Killed'][:8000]
wound = GT['Wounded'][:8000]

def color(x):
    if x>=30:
        color='red'
    elif ((x>0 and x<30)):
        color='blue'
    else:
        color='orange'
    return color
def size(x):
    if (x>30 and x<100):
        size=2
    elif (x>=100 and x<500):
        size=8
    elif x>=500:
        size=16
    else:
        size=0.5
    return size

map = folium.Map(location=[30,0],tiles='cartodbpositron',zoom_start=2)
for point in loca.index:
    info='<b>Country: </b>'+str(coun[point])+'<br><b>City: </b>:
'+str(cit[point])+'<br><b>Killed </b>: '+str(kill[point])
+'<br><b>Wounded</b> : '+str(wound[point])
    iframe = folium.IFrame(html=info, width=200, height=200)

folium.CircleMarker(list(loca.loc[point].values),popup=folium.Popup(if
rame),radius=size(kill[point]),color=color(kill[point])).add_to(map)

map
```

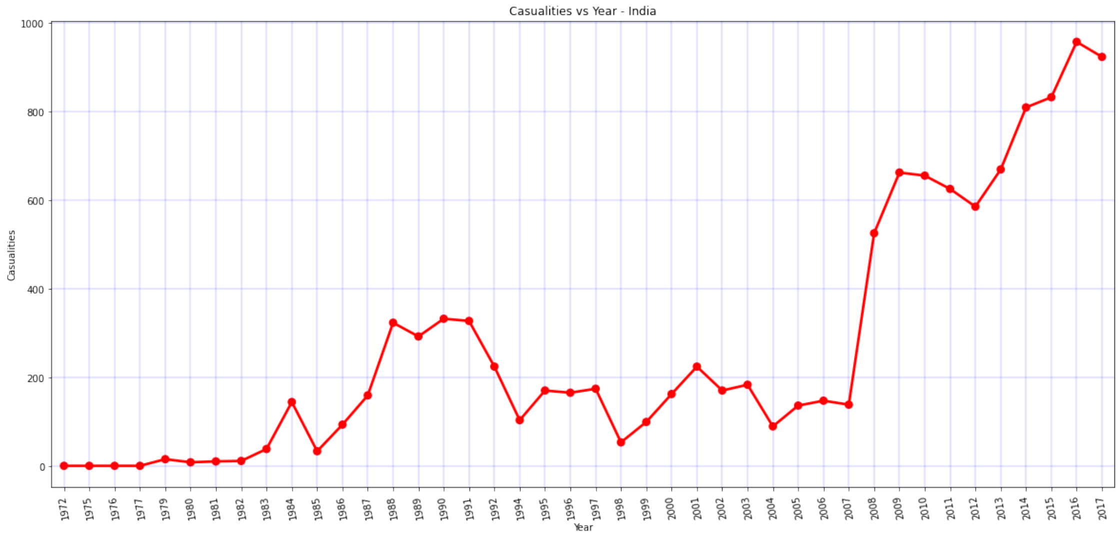
```
<folium.folium.Map at 0x7f5ba1b89c10>
```

INFERENCE:

The map shows that South West Asia has had many attacks compared to the rest of the world

```
#India - Casualities over Years
GTindia=GT[GT['Country']=='India']
year = GTindia["Year"].value_counts()
Year_Attack, Counts_attack = list(year.index), list(year.values)
plt.subplots(figsize=(20,9))
sns.pointplot(x=Year_Attack,y=Counts_attack,color="Red")
plt.xlabel("Year")
plt.xticks(rotation=100)
plt.ylabel("Casualities")
```

```
plt.title("Casualities vs Year - India")
plt.grid(color='b', linestyle='-', linewidth=0.2)
```



INFERENCE:

From the above graph we can see that the Casualities caused by Terrorism in India, same as Global Terorrism, has increased comparitively. The rate of increase in Casulities is not propotional to Years.

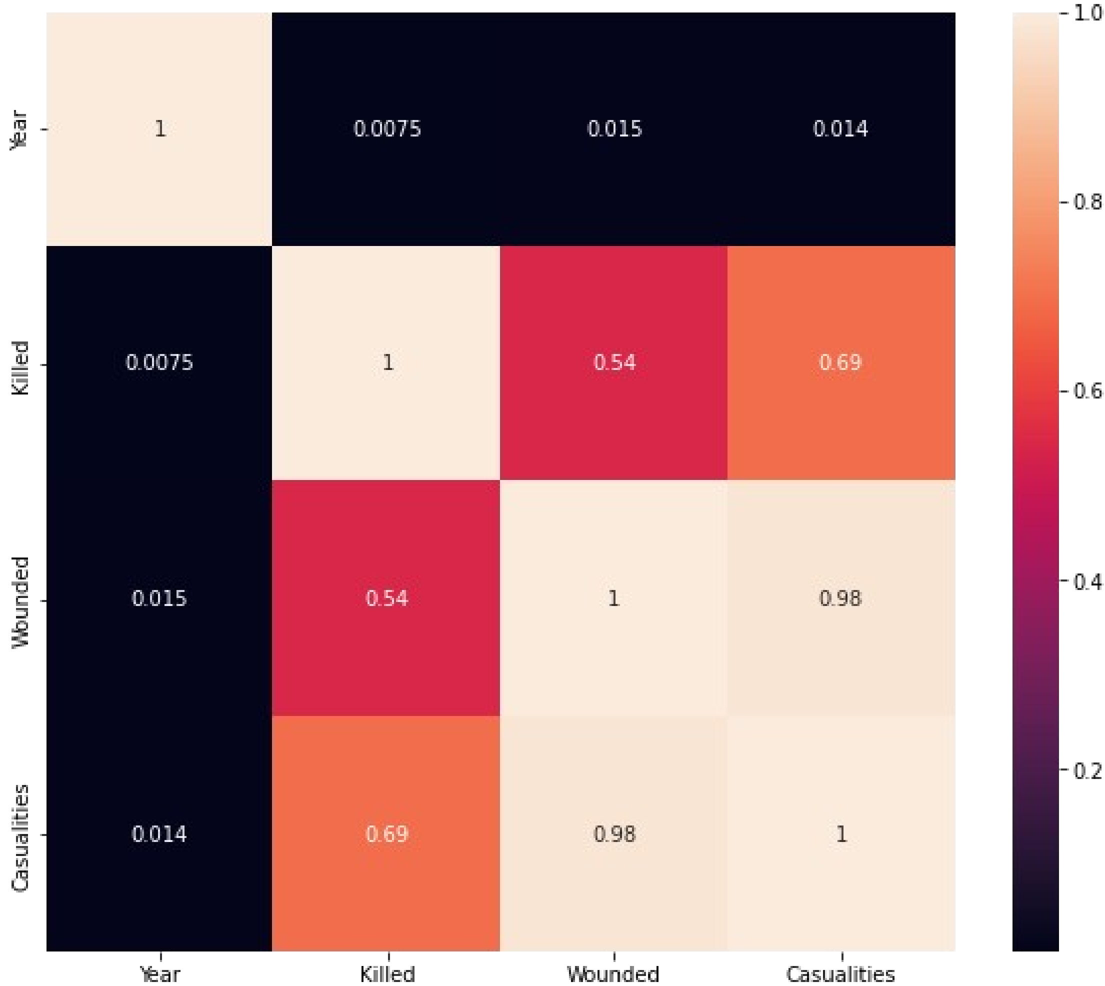
```
location_ind=GTindia[['latitude','longitude']][:5000]
city_ind=GTindia['city'][:5000]
killed_ind=GTindia['Killed'][:5000]
wound_ind=GTindia['Wounded'][:5000]

map1 = folium.Map(location=[20,
78],tiles='cartodbpositron',zoom_start=4)
for point in location_ind.index:

folium.CircleMarker(list(location_ind.loc[point].values),popup='<b>Cit
y: </b>'+str(city_ind[point])+'<br><b>Killed:
</b>'+str(killed_ind[point])+\
                        '<br><b>Injured:
</b>'+str(wound_ind[point]),radius=size(killed_ind[point]),color=color
(killed_ind[point]),fill_color=color(killed_ind[point])).add_to(map1)
map1
```

```
<folium.folium.Map at 0x7f5b9b3f7c10>
```

INFERENCE

- It is clear from the map that the places that border neighbouring countries suffer more attacks.
- Major places and capitals like Mumbai, Delhi, Gandhinagar, Chennai and Hydrabad has had more casualities (Targetted Cities and States)

```
#Correlation representation
hm = GT[['Year','Killed','Wounded','Casualities']]
plt.figure(figsize=(10,8.5))
sns.heatmap(hm.corr(), annot= True)
plt.show()
```



INFERENCE:

The features Killed - Wounded - Casuality are directly propotional

GLOBAL TERRORISM - BINARY CLASSIFICATION

The main objective of this case study is to classify the data available in the global terrorism database (GTD) as 'successful' or 'failure'. This is a binary classification problem.

```
gt = pd.read_csv('/content/drive/MyDrive/Global_Terrorism.csv',
encoding = "ISO-8859-1")
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/
interactiveshell.py:2718: DtypeWarning: Columns
(4,6,31,33,61,62,63,76,79,90,92,94,96,114,115,121) have mixed
types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
gt = gt.loc[(gt.doubtterr == 0) & (gt.nkill >=0)]
gt.shape
```

```
(132137, 135)
```

```
gt
```

```
             eventid  iyear  imonth  iday  ... INT_IDEO  INT_MISC
INT_ANY  related
0       197000000001   1970       7     2  ...        0         0
0      NaN
1       197000000002   1970       0     0  ...        1         1
1      NaN
2       197001000001   1970       1     0  ...       -9         1
1      NaN
5       197001010002   1970       1     1  ...       -9         0
-9      NaN
6       197001020001   1970       1     2  ...        0         0
0      NaN
...              ...    ...     ...   ...  ...      ...       ...
...     ...
181684  201712310019   2017      12    31  ...        0         0
0      NaN
181685  201712310020   2017      12    31  ...       -9         0
-9      NaN
181688  201712310030   2017      12    31  ...        0         0
0      NaN
181689  201712310031   2017      12    31  ...       -9         0
-9      NaN
181690  201712310032   2017      12    31  ...       -9         0
-9      NaN

[132137 rows x 135 columns]
```

```
GT =
gt[['iyear','imonth','extended','country_txt','region_txt','nkill','nw
ound','success','crit1','crit2','crit3','multiple',
     'vicinity',
'specificity','suicide','attacktype1_txt','weaptype1_txt','targtype1_t
xt','gname','propextent_txt','ishostkid','INT_ANY']]
```

```
GT
```

```
        iyear  imonth  ...  ishostkid INT_ANY
0        1970       7  ...        0.0       0
1        1970       0  ...        1.0       1
```

```
2         1970       1  ...        0.0       1
5         1970       1  ...        0.0      -9
6         1970       1  ...        0.0       0
...        ...     ...  ...        ...     ...
181684   2017      12  ...        1.0       0
181685   2017      12  ...        0.0      -9
181688   2017      12  ...        0.0       0
181689   2017      12  ...        0.0      -9
181690   2017      12  ...        0.0      -9

[132137 rows x 22 columns]
```

```
GT = GT.loc[(GT.ishostkid != -9) & (GT.INT_ANY !=-9) ]
GT.shape
```

```
(70564, 22)
```

```
GT=GT.replace('Unknown', np.nan)
```

```
GT=GT.replace('Other',np.nan)
```

```
GT.isnull().sum()
```

```
iyear                 0
imonth                0
extended              0
country_txt           0
region_txt            0
nkill                 0
nwound             3242
success               0
crit1                 0
crit2                 0
crit3                 0
multiple              0
vicinity              0
specificity           3
suicide               0
attacktype1_txt    3005
weaptype1_txt      6466
targtype1_txt      2183
gname              4494
propextent_txt    51435
ishostkid           119
INT_ANY               0
dtype: int64
```

```
GT
```

```
        iyear  imonth  ...  ishostkid INT_ANY
0        1970       7  ...        0.0       0
1        1970       0  ...        1.0       1
```

```
2         1970       1  ...         0.0       1
6         1970       1  ...         0.0       0
8         1970       1  ...         0.0       0
...        ...     ...  ...         ...     ...
181677   2017      12  ...         0.0       0
181681   2017      12  ...         0.0       0
181683   2017      12  ...         0.0       0
181684   2017      12  ...         1.0       0
181688   2017      12  ...         0.0       0

[70564 rows x 22 columns]
```

```
GT.dropna(inplace=True)
GT.shape
```

```
(16777, 22)
```

```
GT.success.value_counts()
```

```
1    16323
0      454
Name: success, dtype: int64
```

```
GT.dtypes
```

```
iyear                int64
imonth               int64
extended             int64
country_txt         object
region_txt          object
nkill              float64
nwound             float64
success              int64
crit1                int64
crit2                int64
crit3                int64
multiple           float64
vicinity             int64
specificity        float64
suicide              int64
attacktype1_txt     object
weaptype1_txt       object
targtype1_txt       object
gname               object
propextent_txt      object
ishostkid          float64
INT_ANY              int64
dtype: object
```

```
pip install feature-engine
```

```
Collecting feature-engine
  Downloading feature_engine-1.1.2-py2.py3-none-any.whl (180 kB)
odels>=0.11.1
  Downloading statsmodels-0.13.1-cp37-cp37m-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (9.8 MB)
ent already satisfied: pandas>=1.0.3 in /usr/local/lib/python3.7/dist-
packages (from feature-engine) (1.1.5)
Requirement already satisfied: scipy>=1.4.1 in
/usr/local/lib/python3.7/dist-packages (from feature-engine) (1.4.1)
Requirement already satisfied: scikit-learn>=0.22.2 in
/usr/local/lib/python3.7/dist-packages (from feature-engine) (1.0.1)
Requirement already satisfied: numpy>=1.18.2 in
/usr/local/lib/python3.7/dist-packages (from feature-engine) (1.19.5)
Requirement already satisfied: pytz>=2017.2 in
/usr/local/lib/python3.7/dist-packages (from pandas>=1.0.3->feature-
engine) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=1.0.3->feature-
engine) (2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3-
>pandas>=1.0.3->feature-engine) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.22.2-
>feature-engine) (3.0.0)
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.22.2-
>feature-engine) (1.1.0)
Requirement already satisfied: patsy>=0.5.2 in
/usr/local/lib/python3.7/dist-packages (from statsmodels>=0.11.1-
>feature-engine) (0.5.2)
Installing collected packages: statsmodels, feature-engine
  Attempting uninstall: statsmodels
    Found existing installation: statsmodels 0.10.2
    Uninstalling statsmodels-0.10.2:
      Successfully uninstalled statsmodels-0.10.2
Successfully installed feature-engine-1.1.2 statsmodels-0.13.1
```

{"pip_warning":{"packages":["statsmodels"]}}

```
from feature_engine.creation import CyclicalTransformer

df = GT[['iyear','imonth']]
cyclical = CyclicalTransformer(variables=None, drop_original=True)
X = cyclical.fit_transform(df)
X = X.groupby(X.columns,axis=1).apply(lambda x: round(x,3))

GT = pd.concat([X, GT], axis=1)
GT= GT.drop('iyear', axis=1)
```

```
GT= GT.drop('imonth', axis=1)
GT

        iyear_sin  iyear_cos  ...  ishostkid  INT_ANY
8          -0.146      0.989  ...        0.0        0
9          -0.146      0.989  ...        0.0        0
21         -0.146      0.989  ...        0.0        0
55         -0.146      0.989  ...        0.0        1
56         -0.146      0.989  ...        0.0        1
...           ...        ...  ...        ...      ...
181659     -0.000      1.000  ...        0.0        1
181665     -0.000      1.000  ...        0.0        0
181676     -0.000      1.000  ...        0.0        0
181677     -0.000      1.000  ...        0.0        0
181681     -0.000      1.000  ...        0.0        0

[16777 rows x 24 columns]

#GT= pd.get_dummies(data=GT,
columns=['extended','country_txt','region_txt','crit1','crit2','crit3'
,'multiple',
#     'vicinity',
'specificity','attacktype1_txt','weaptype1_txt','targtype1_txt','prope
xtent_txt','gname','ishostkid','INT_ANY'])

from sklearn import preprocessing
# creating instance of labelencoder
labelencoder = preprocessing.LabelEncoder()
# Assigning numerical values and storing in another column
GT['country_txt']= labelencoder.fit_transform(GT['country_txt'])
GT['region_txt']= labelencoder.fit_transform(GT['region_txt'])
GT['attacktype1_txt']=
labelencoder.fit_transform(GT['attacktype1_txt'])
GT['weaptype1_txt']= labelencoder.fit_transform(GT['weaptype1_txt'])
GT['targtype1_txt']= labelencoder.fit_transform(GT['targtype1_txt'])
GT['propextent_txt']= labelencoder.fit_transform(GT['propextent_txt'])
GT['gname']= labelencoder.fit_transform(GT['gname'])
GT

        iyear_sin  iyear_cos  imonth_sin  ...  propextent_txt
ishostkid  INT_ANY
8          -0.146      0.989       0.500  ...               2
0.0        0
9          -0.146      0.989       0.500  ...               2
0.0        0
21         -0.146      0.989       0.500  ...               2
0.0        0
55         -0.146      0.989       0.866  ...               2
0.0        1
56         -0.146      0.989       0.866  ...               2
0.0        1
```

```
...           ...        ...          ...  ...              ...         .
..      ...
181659     -0.000      1.000      -0.000  ...                2
0.0        1
181665     -0.000      1.000      -0.000  ...                2
0.0        0
181676     -0.000      1.000      -0.000  ...                2
0.0        0
181677     -0.000      1.000      -0.000  ...                2
0.0        0
181681     -0.000      1.000      -0.000  ...                2
0.0        0

[16777 rows x 24 columns]
```

```
GT.shape
```

```
(16777, 24)
```

```
GT
```

```
        iyear_sin  iyear_cos  imonth_sin  ...  propextent_txt
ishostkid  INT_ANY
8          -0.146      0.989       0.500  ...                2
0.0        0
9          -0.146      0.989       0.500  ...                2
0.0        0
21         -0.146      0.989       0.500  ...                2
0.0        0
55         -0.146      0.989       0.866  ...                2
0.0        1
56         -0.146      0.989       0.866  ...                2
0.0        1
...           ...        ...          ...  ...              ...         .
..      ...
181659     -0.000      1.000      -0.000  ...                2
0.0        1
181665     -0.000      1.000      -0.000  ...                2
0.0        0
181676     -0.000      1.000      -0.000  ...                2
0.0        0
181677     -0.000      1.000      -0.000  ...                2
0.0        0
181681     -0.000      1.000      -0.000  ...                2
0.0        0

[16777 rows x 24 columns]
```

```
GT1=
GT[['iyear_sin','iyear_cos','imonth_sin','imonth_cos','nkill','nwound'
]]
```

```
GT1['casulaties']= GT1['nkill']+GT1['nwound']
GT1
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy


        iyear_sin  iyear_cos  imonth_sin  imonth_cos  nkill  nwound
casulaties
8          -0.146      0.989       0.500       0.866    0.0     0.0
0.0
9          -0.146      0.989       0.500       0.866    0.0     0.0
0.0
21         -0.146      0.989       0.500       0.866    0.0     0.0
0.0
55         -0.146      0.989       0.866       0.500    0.0     0.0
0.0
56         -0.146      0.989       0.866       0.500    0.0     0.0
0.0
...           ...        ...         ...         ...    ...     ...
...
181659     -0.000      1.000      -0.000       1.000    0.0     0.0
0.0
181665     -0.000      1.000      -0.000       1.000    0.0     0.0
0.0
181676     -0.000      1.000      -0.000       1.000    5.0     0.0
5.0
181677     -0.000      1.000      -0.000       1.000    0.0     0.0
0.0
181681     -0.000      1.000      -0.000       1.000    1.0     5.0
6.0

[16777 rows x 7 columns]
```

```
GT=
GT.drop(['iyear_sin','iyear_cos','imonth_sin','imonth_cos','nkill','nw
ound'], axis=1)
GT = pd.concat([GT1, GT], axis=1)
GT
```

```
        iyear_sin  iyear_cos  imonth_sin  ...  propextent_txt
ishostkid  INT_ANY
8          -0.146      0.989       0.500  ...               2
0.0        0
9          -0.146      0.989       0.500  ...               2
```

```
0.0        0
21         -0.146      0.989       0.500  ...               2
0.0        0
55         -0.146      0.989       0.866  ...               2
0.0        1
56         -0.146      0.989       0.866  ...               2
0.0        1
...           ...         ...          ...  ...             ...         .
..      ...
181659     -0.000      1.000      -0.000  ...               2
0.0        1
181665     -0.000      1.000      -0.000  ...               2
0.0        0
181676     -0.000      1.000      -0.000  ...               2
0.0        0
181677     -0.000      1.000      -0.000  ...               2
0.0        0
181681     -0.000      1.000      -0.000  ...               2
0.0        0

[16777 rows x 25 columns]
```

```
GT.reset_index(drop=True, inplace=True)
```

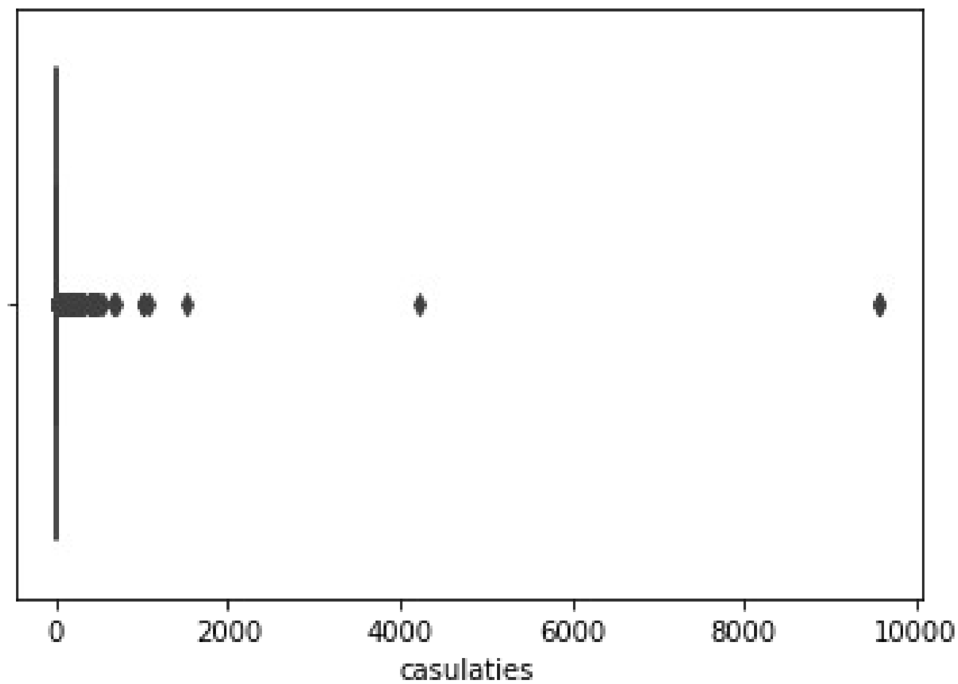
```
GT
```

```
       iyear_sin  iyear_cos  imonth_sin  ...  propextent_txt
ishostkid  INT_ANY
0         -0.146      0.989       0.500  ...               2
0.0        0
1         -0.146      0.989       0.500  ...               2
0.0        0
2         -0.146      0.989       0.500  ...               2
0.0        0
3         -0.146      0.989       0.866  ...               2
0.0        1
4         -0.146      0.989       0.866  ...               2
0.0        1
...          ...         ...          ...  ...             ...         ..
.       ...
16772     -0.000      1.000      -0.000  ...               2
0.0        1
16773     -0.000      1.000      -0.000  ...               2
0.0        0
16774     -0.000      1.000      -0.000  ...               2
0.0        0
16775     -0.000      1.000      -0.000  ...               2
0.0        0
16776     -0.000      1.000      -0.000  ...               2
0.0        0
```

```
[16777 rows x 25 columns]
```

```
#outliers
import seaborn as sns
sns.boxplot(x=GT['casulaties'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6a0f6ff890>
```




```
# calculate interquartile range
q25, q75 = np.percentile(GT['casulaties'], 25),
np.percentile(GT['casulaties'], 75)
iqr = q75 - q25
print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))

# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off

# identify outliers
for x in GT['casulaties']:
  if(x<lower or x>upper):
    GT.drop(GT.loc[(GT.casulaties)==x].index, inplace=True, axis=0)
```

```
Percentiles: 25th=0.000, 75th=5.000, IQR=5.000
```

```
GT.shape
```

```
(14640, 25)
```

```
GT.success.value_counts()
```

```
1    14226
0      414
Name: success, dtype: int64
```

```
target= GT.success
GT.drop('success', axis=1, inplace= True)
target
```

```
0        1
1        1
2        1
3        1
4        1
        ..
16772    1
16773    1
16774    1
16775    1
16776    1
Name: success, Length: 14640, dtype: int64
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(GT, target,
test_size=0.2)
```

```
from sklearn.linear_model import LogisticRegression
# all parameters not specified are set to their defaults
lg = LogisticRegression()
lg.fit(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/
_logistic.py:818: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
LogisticRegression()
```

```
y_pred_lg = lg.predict(x_test)
```

```
from sklearn.metrics import roc_curve, auc, confusion_matrix,
classification_report, accuracy_score
```

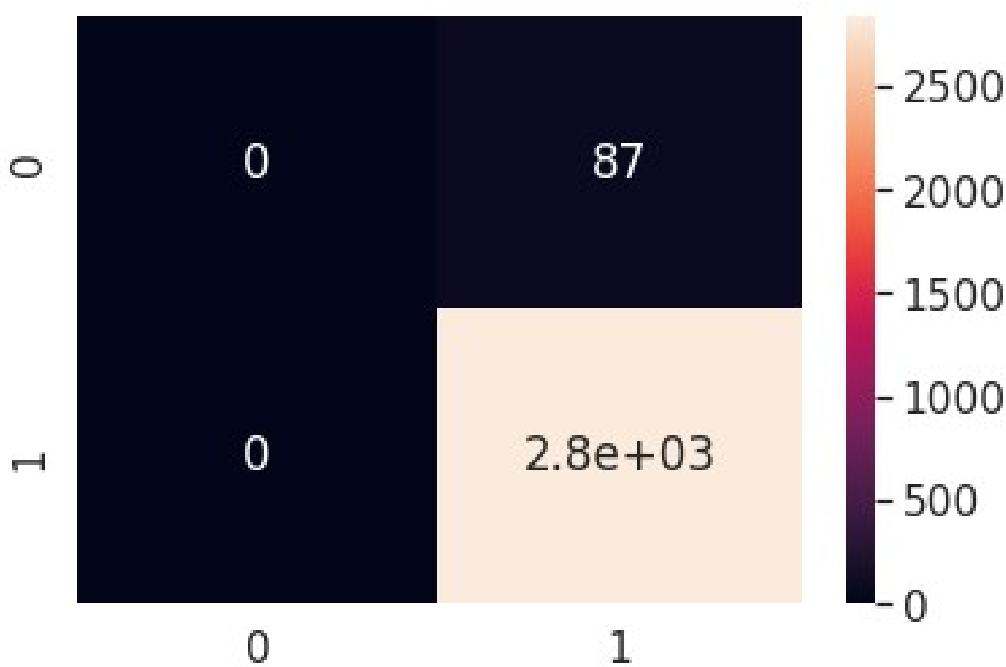
```
score_lg = accuracy_score(y_pred_lg,y_test)
score_lg
```

```
0.9702868852459017
```

```
print("train score - " + str(lg.score(x_train, y_train)))
print("test score - " + str(lg.score(x_test, y_test)))
```

```
train score - 0.9720799180327869
test score - 0.9702868852459017
```

```
#Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_lg = confusion_matrix(y_test,y_pred_lg)
sns.set(font_scale=1.4)
sns.heatmap(cm_lg, annot=True)
plt.show()
```




```
print(classification_report(y_test, y_pred_lg))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        87
           1       0.97      1.00      0.98      2841

    accuracy                           0.97      2928
   macro avg       0.49      0.50      0.49      2928
weighted avg       0.94      0.97      0.96      2928
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/
_classification.py:1308: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification
.py:1308: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification
.py:1308: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

FEATURE SELECTION

(1) Low variance filter

```
from sklearn.feature_selection import VarianceThreshold
var_thres=VarianceThreshold(threshold=0)
var_thres.fit(GT)
```

```
VarianceThreshold(threshold=0)
```

```
var_thres.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True])
```

```
GT.columns[var_thres.get_support()]
```

```
Index(['iyear_sin', 'iyear_cos', 'imonth_sin', 'imonth_cos', 'nkill',
'nwound',
       'casulaties', 'extended', 'country_txt', 'region_txt', 'crit2',
'crit3',
       'multiple', 'vicinity', 'specificity', 'suicide',
'attacktype1_txt',
       'weaptype1_txt', 'targtype1_txt', 'gname', 'propextent_txt',
       'ishostkid', 'INT_ANY'],
      dtype='object')
```

```
constant_columns = [column for column in GT.columns
                    if column not in
GT.columns[var_thres.get_support()]]
```

```
print(len(constant_columns))
```

```
1
```

```python
for feature in constant_columns:
     print(feature)
```

```
crit1
```

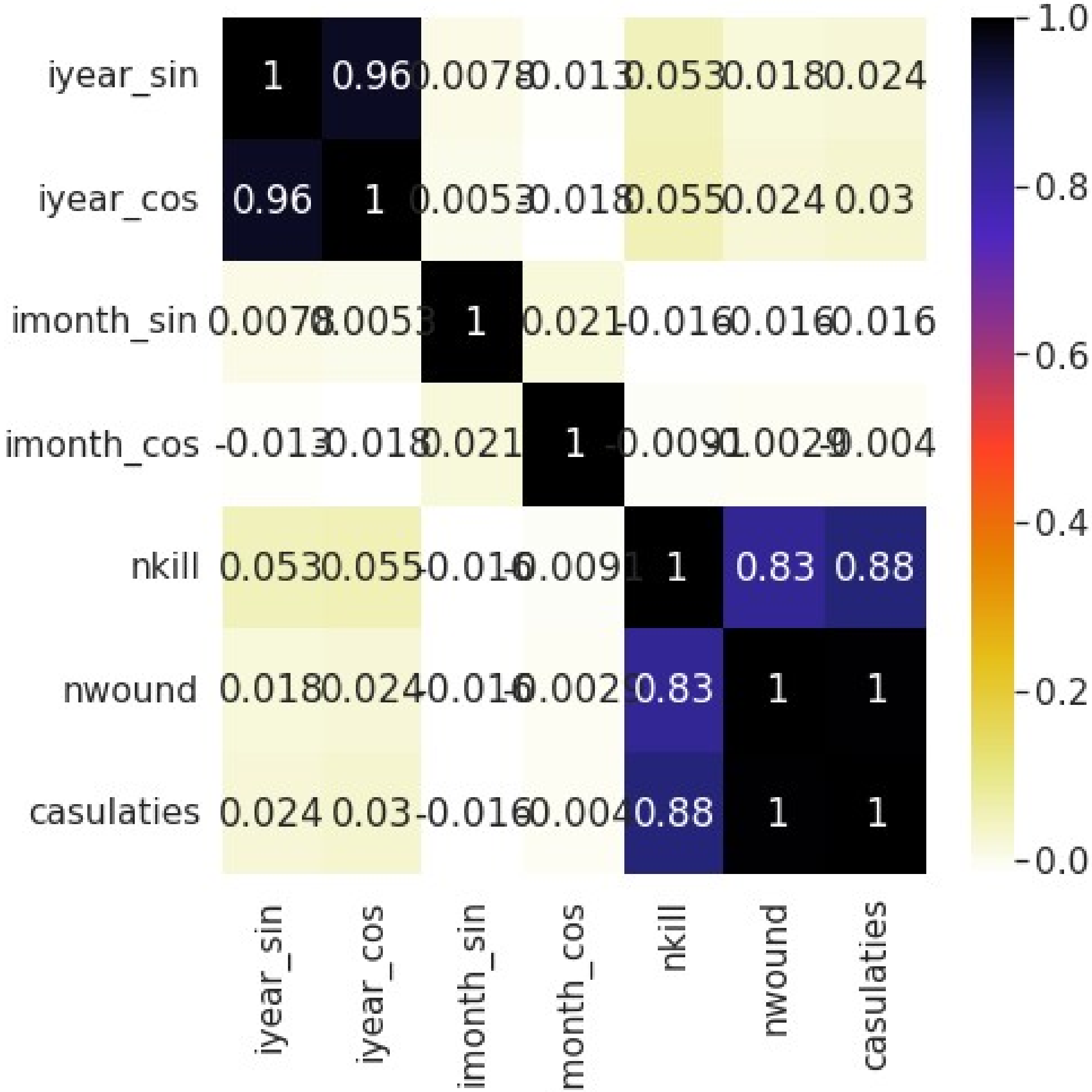
```python
GT= GT.drop(constant_columns,axis=1)
```

(2) High correlation filter

```python
 GT1.corr()
```

```
            iyear_sin  iyear_cos  imonth_sin  ...     nkill    nwound
casulaties
iyear_sin    1.000000   0.963642    0.007827  ...  0.052817  0.017822
0.024091
iyear_cos    0.963642   1.000000    0.005302  ...  0.055201  0.024010
0.029794
imonth_sin   0.007827   0.005302    1.000000  ... -0.016187 -0.015842
-0.016282
imonth_cos  -0.013271  -0.018387    0.021238  ... -0.009149 -0.002868
-0.003985
nkill        0.052817   0.055201   -0.016187  ...  1.000000  0.828968
0.877537
nwound       0.017822   0.024010   -0.015842  ...  0.828968  1.000000
0.995638
casulaties   0.024091   0.029794   -0.016282  ...  0.877537  0.995638
1.000000

[7 rows x 7 columns]
```

```python
plt.figure(figsize=(7,7))
cor =GT1.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.CMRmap_r)
plt.show()
```




```python
# with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything
other feature

def correlation(dataset, threshold):
    col_corr = []  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are
interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name
of column
                col_corr.append([colname,corr_matrix.columns[j]])
    return col_corr
```

```
corr_features = correlation(GT1, 0.8)
len(corr_features)
```

```
4
```

```
corr_features
```

```
[['iyear_cos', 'iyear_sin'],
 ['nwound', 'nkill'],
 ['casulaties', 'nkill'],
 ['casulaties', 'nwound']]
```

```
GT= GT.drop(['nwound','nkill'], axis=1)
```

(3) Chi square test

```
GT2= GT[['extended', 'country_txt', 'region_txt', 'crit2', 'crit3',
'multiple', 'vicinity', 'specificity', 'suicide',
       'attacktype1_txt', 'weaptype1_txt', 'targtype1_txt', 'gname',
'propextent_txt', 'ishostkid', 'INT_ANY']]
```

```
GT2
```

```
       extended  country_txt  region_txt  ...  propextent_txt
ishostkid  INT_ANY
0             0          123           6  ...               2
0.0        0
1             0          123           6  ...               2
0.0        0
2             0          124           7  ...               2
0.0        0
3             0          123           6  ...               2
0.0        1
4             0          123           6  ...               2
0.0        1
...         ...          ...         ...  ...             ...         .
..      ...
16772         0           61          10  ...               2
0.0        1
16773         0           49           8  ...               2
0.0        0
16774         0            0           8  ...               2
0.0        0
16775         0           92           9  ...               2
0.0        0
16776         0           92           9  ...               2
0.0        0

[14640 rows x 16 columns]
```

```
target.shape
```

```
(14640,)
```

```
from sklearn.feature_selection import chi2
f_p_values= chi2(GT2,target)
```

```
f_p_values
```

```
(array([4.62302686e+00, 4.92754250e+01, 1.14699950e+01, 7.95235548e-
06,
        1.98795306e-06, 6.04375435e+01, 4.86155311e-01,
1.72947445e+00,
        1.74338265e+01, 7.97739094e+01, 1.99010202e+01,
9.87249583e+01,
        2.66908071e-01, 1.02792188e-01, 1.17250588e+01, 5.25044448e-
01]),
 array([3.15455245e-02, 2.22431640e-12, 7.07288693e-04, 9.97749975e-
01,
        9.98875025e-01, 7.59504716e-15, 4.85647363e-01, 1.88478030e-
01,
        2.97484607e-05, 4.19799716e-19, 8.15567928e-06, 2.90122519e-
23,
        6.05413446e-01, 7.48504284e-01, 6.16640999e-04, 4.68697760e-
01]))
```

```
p_values=pd.Series(f_p_values[1])
p_values.index=GT2.columns
p_values
```

```
extended           3.154552e-02
country_txt        2.224316e-12
region_txt         7.072887e-04
crit2              9.977500e-01
crit3              9.988750e-01
multiple           7.595047e-15
vicinity           4.856474e-01
specificity        1.884780e-01
suicide            2.974846e-05
attacktype1_txt    4.197997e-19
weaptype1_txt      8.155679e-06
targtype1_txt      2.901225e-23
gname              6.054134e-01
propextent_txt     7.485043e-01
ishostkid          6.166410e-04
INT_ANY            4.686978e-01
dtype: float64
```

```
p_values.sort_index(ascending=False)
```

```
weaptype1_txt      8.155679e-06
vicinity           4.856474e-01
targtype1_txt      2.901225e-23
suicide            2.974846e-05
```

```
specificity         1.884780e-01
region_txt          7.072887e-04
propextent_txt      7.485043e-01
multiple            7.595047e-15
ishostkid           6.166410e-04
gname               6.054134e-01
extended            3.154552e-02
crit3               9.988750e-01
crit2               9.977500e-01
country_txt         2.224316e-12
attacktype1_txt     4.197997e-19
INT_ANY             4.686978e-01
dtype: float64
```

```
GT.drop(['INT_ANY','attacktype1_txt','country_txt','crit2','crit3','ex
tended'], inplace= True, axis=1)
```

```
GT['success']= target
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(GT, target,
test_size=0.2)
```

```
from sklearn.linear_model import LogisticRegression
# all parameters not specified are set to their defaults
lg = LogisticRegression()
lg.fit(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/
_logistic.py:818: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
LogisticRegression()
```

```
y_pred_lg = lg.predict(x_test)
```

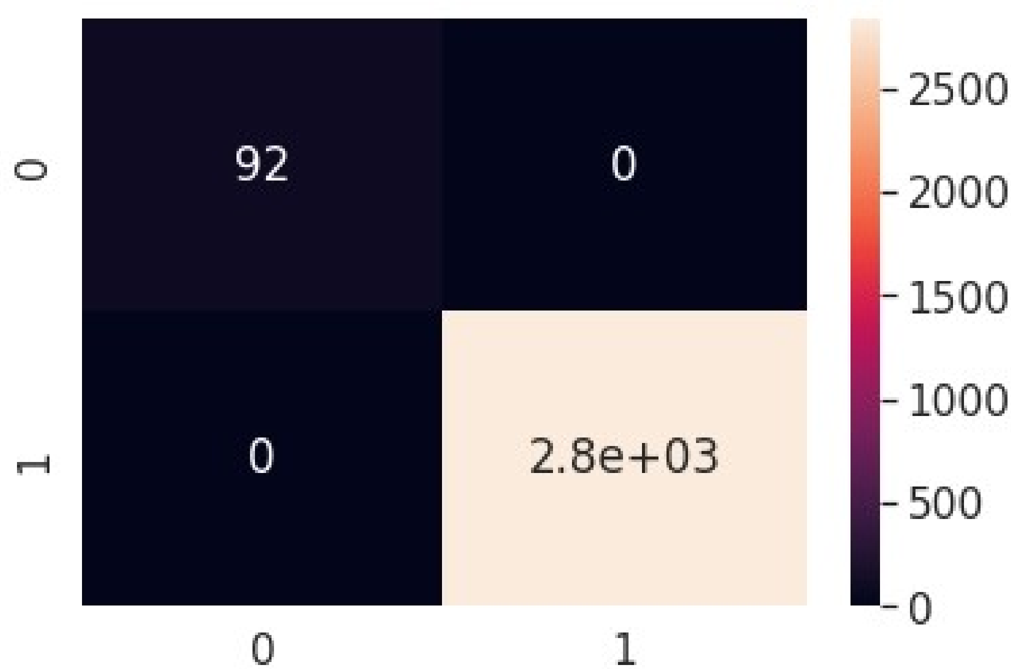
```
from sklearn.metrics import roc_curve, auc, confusion_matrix,
classification_report, accuracy_score
score_lg = accuracy_score(y_pred_lg,y_test)
score_lg
```

```
1.0
```

```
print("train score - " + str(lg.score(x_train, y_train)))
print("test score - " + str(lg.score(x_test, y_test)))
```

```
train score - 1.0
test score - 1.0
```

```
#Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm_lg = confusion_matrix(y_test,y_pred_lg)
sns.set(font_scale=1.4)
sns.heatmap(cm_lg, annot=True)
plt.show()
```




```
print(classification_report(y_test, y_pred_lg))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        92
           1       1.00      1.00      1.00      2836

    accuracy                           1.00      2928
   macro avg       1.00      1.00      1.00      2928
weighted avg       1.00      1.00      1.00      2928
```

Smote

```
pip install imbalanced-learn
```

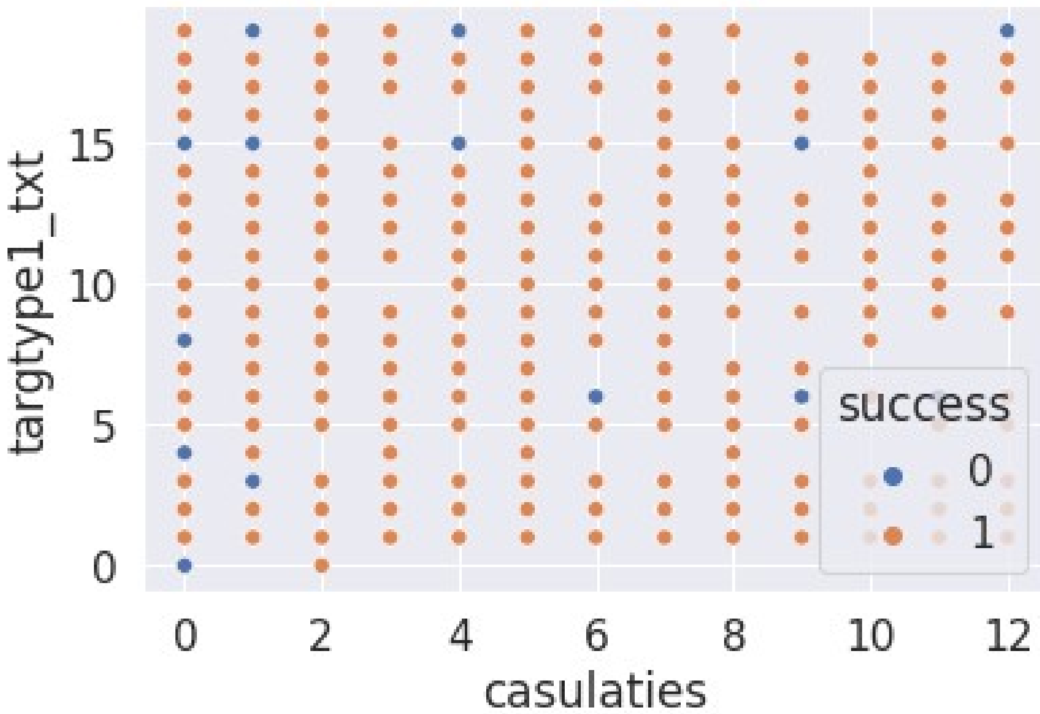
```
Requirement already satisfied: imbalanced-learn in
/usr/local/lib/python3.7/dist-packages (0.8.1)
Requirement already satisfied: scipy>=0.19.1 in
/usr/local/lib/python3.7/dist-packages (from imbalanced-learn) (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in
/usr/local/lib/python3.7/dist-packages (from imbalanced-learn)
(1.19.5)
Requirement already satisfied: scikit-learn>=0.24 in
/usr/local/lib/python3.7/dist-packages (from imbalanced-learn) (1.0.1)
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.7/dist-packages (from imbalanced-learn) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.24-
>imbalanced-learn) (3.0.0)
```

```
GT
```

```
       iyear_sin  iyear_cos  imonth_sin  ...  propextent_txt
ishostkid  success
0         -0.146      0.989       0.500  ...               2
0.0        1
1         -0.146      0.989       0.500  ...               2
0.0        1
2         -0.146      0.989       0.500  ...               2
0.0        1
3         -0.146      0.989       0.866  ...               2
0.0        1
4         -0.146      0.989       0.866  ...               2
0.0        1
...          ...        ...         ...  ...             ...         ..
.      ...
16772     -0.000      1.000      -0.000  ...               2
0.0        1
16773     -0.000      1.000      -0.000  ...               2
0.0        1
16774     -0.000      1.000      -0.000  ...               2
0.0        1
16775     -0.000      1.000      -0.000  ...               2
0.0        1
16776     -0.000      1.000      -0.000  ...               2
0.0        1

[14640 rows x 16 columns]
```

```
sns.scatterplot(data= GT, x= GT.casulaties,
                    y= GT.targtype1_txt,
                    hue= GT.success)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6a09e54c50>
```

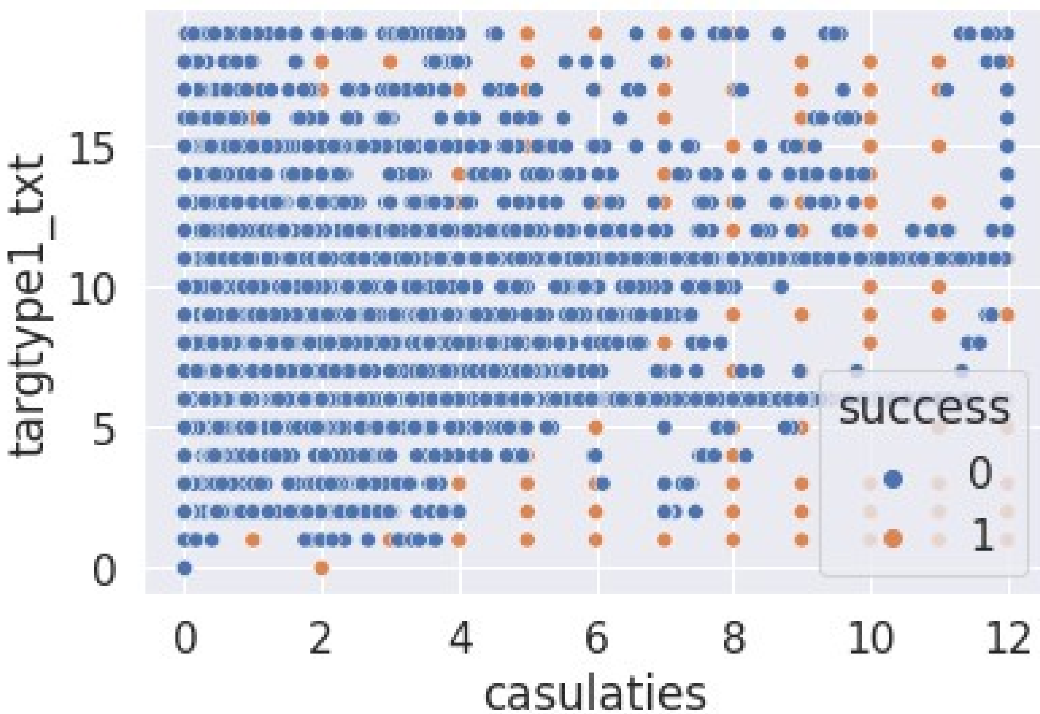



```
# Oversample and plot imbalanced dataset with SMOTE
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from numpy import where

print("Before OverSampling, counts of label '1':
{}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \
n".format(sum(y_train == 0)))

# import SMOTE module from imblearn library
# pip install imblearn (if you don't have imblearn in your system)
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
x_train_res, y_train_res = sm.fit_resample(x_train, y_train.ravel())

print('After OverSampling, the shape of train_X:
{}'.format(x_train_res.shape))
print('After OverSampling, the shape of train_y: {} \
n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1':
{}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0':
{}".format(sum(y_train_res == 0)))
```

```
Before OverSampling, counts of label '1': 11390
Before OverSampling, counts of label '0': 322

After OverSampling, the shape of train_X: (22780, 16)
After OverSampling, the shape of train_y: (22780,)

After OverSampling, counts of label '1': 11390
After OverSampling, counts of label '0': 11390
```

```
sns.scatterplot(x= x_train_res.casulaties,
                    y= x_train_res.targtype1_txt,
                    hue= x_train_res.success)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6a0f94e510>
```




```
lr1 = LogisticRegression()
lr1.fit(x_train_res, y_train_res.ravel())
predictions = lr1.predict(x_test)

# print classification report
print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	92
1	1.00	1.00	1.00	2836
accuracy			1.00	2928

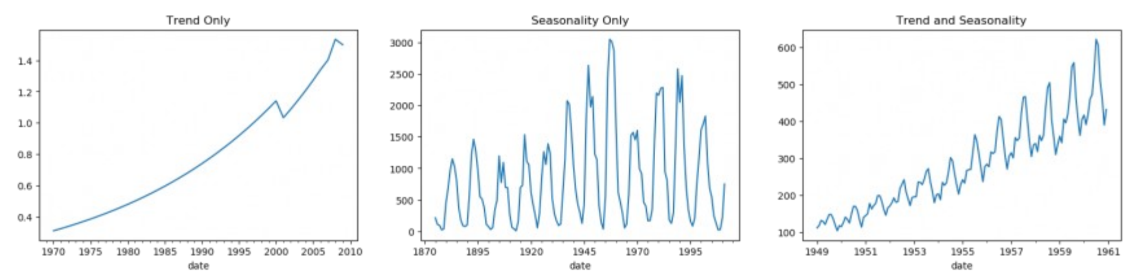
```
   macro avg       1.00      1.00      1.00      2928
weighted avg       1.00      1.00      1.00      2928


/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/
_logistic.py:818: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

TIME SERIES ANALYSIS - ANNUAL DATA

Time series data is a collection of quantities that are assembled over ***even*** intervals in time and ordered chronologically

```
GT['Date'].min(), GT['Date'].max()
```

```
(Timestamp('1970-01-02 00:00:00'), Timestamp('2017-12-31 00:00:00'))
```

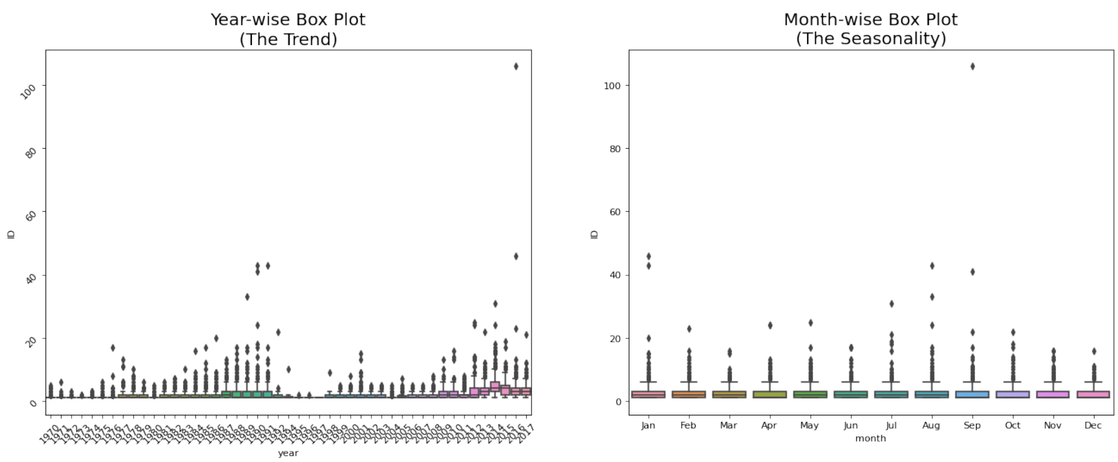



```
df = GT.groupby("Date")['ID'].count().reset_index()
df['year'] = [d.year for d in df.Date]
df['month'] = [d.strftime('%b') for d in df.Date]
df
```

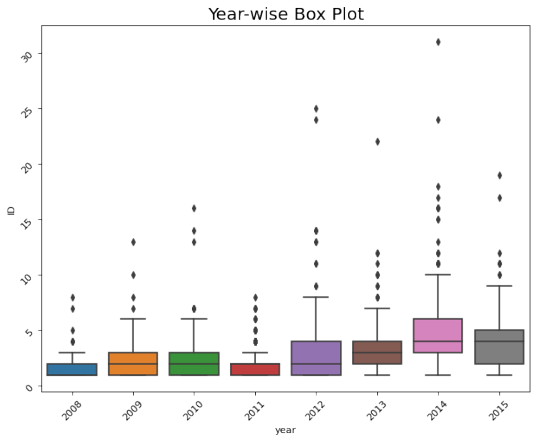
```
           Date  ID  year month
0    1970-01-02   1  1970   Jan
1    1970-01-03   1  1970   Jan
2    1970-01-15   1  1970   Jan
3    1970-02-08   2  1970   Feb
4    1970-02-13   1  1970   Feb
...         ...  ..   ...   ...
6373 2017-12-27   2  2017   Dec
6374 2017-12-28   1  2017   Dec
```

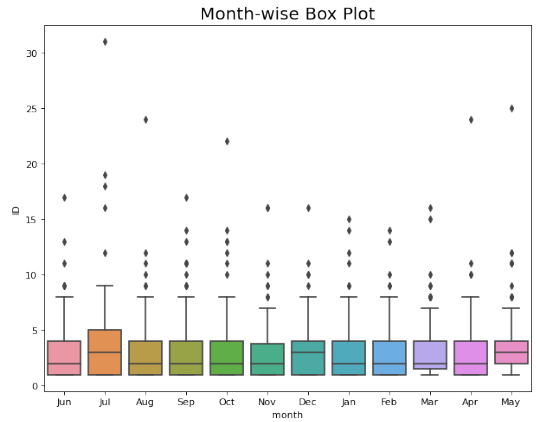
```
6375 2017-12-29   1  2017   Dec
6376 2017-12-30   2  2017   Dec
6377 2017-12-31   4  2017   Dec

[6378 rows x 4 columns]
```

```python
fig, axes = plt.subplots(1, 2, figsize=(20,7), dpi= 80)
sns.boxplot(x='year', y='ID', data=df, ax=axes[0])
sns.boxplot(x='month', y='ID', data=df)

# Set Titles
axes[0].set_title('Year-wise Box Plot\n(The Trend)', fontsize=18);
axes[0].tick_params(labelrotation=45)

axes[1].set_title('Month-wise Box Plot\n(The Seasonality)',
fontsize=18)
plt.show()
```




```python
fig, axes = plt.subplots(1, 2, figsize=(20,7), dpi= 80)
sns.boxplot(x='year', y='ID', data=df[3500:5700], ax=axes[0])
sns.boxplot(x='month', y='ID', data=df[3500:5700])

# Set Titles
axes[0].set_title('Year-wise Box Plot', fontsize=18);
axes[0].tick_params(labelrotation=45)

axes[1].set_title('Month-wise Box Plot', fontsize=18)
plt.show()
```





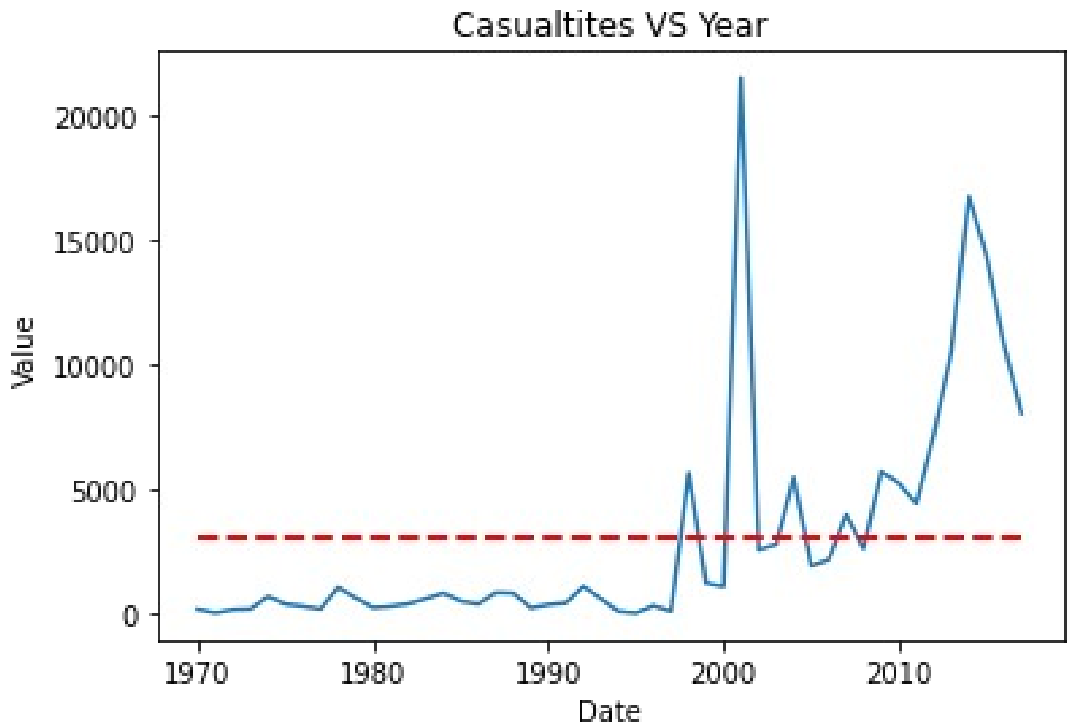
Date wise grouping does not have even interval

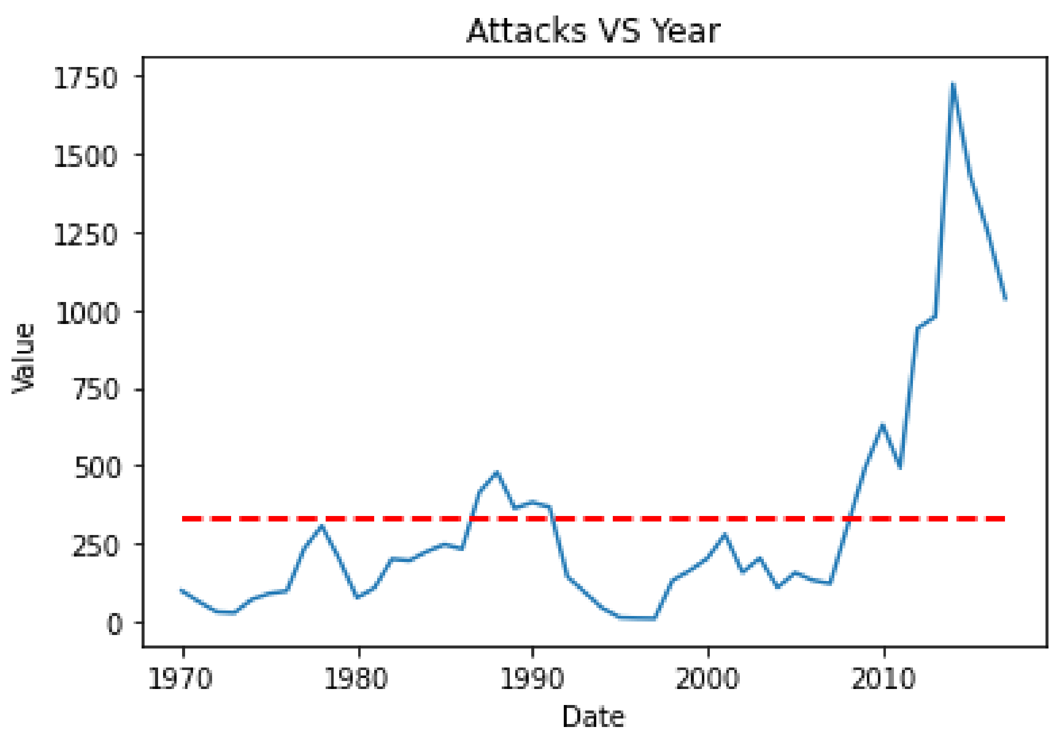
=>Year wise grouping

```
# Year - Number of attacks - Casualities
GT1 = GT.groupby("Year")['ID'].count().reset_index()
GT1["Casualities"] =
GT.groupby('Year').Casualities.sum().reset_index()["Casualities"]
GT1.head()
```

```
   Year  ID  Casualities
0  1970  98        155.0
1  1971  62          9.0
2  1972  30        144.0
3  1973  27        161.0
4  1974  70        671.0
```

```
def plot_df(df, x, y, title="", xlabel='Date', ylabel='Value',
dpi=100):
    fig, ax = plt.subplots()
    ax.plot(x, y)
    y_avg = [np.mean(y)] * len(y)
    ax.plot(x, y_avg, color='red', lw=2, ls='--')
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
    plt.show()

plot_df(GT1, x=GT1.Year, y=GT1.Casualities, title='Casualtites VS
Year')
print()
plot_df(GT1, x=GT1.Year, y=GT1.ID, title='Attacks VS Year')
```

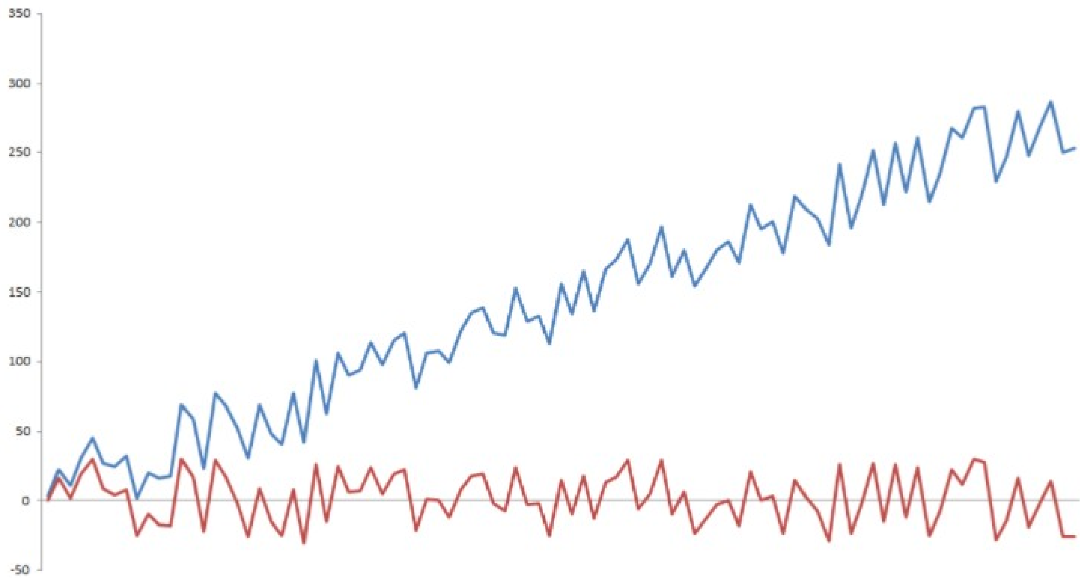




Stationary or Non-stationary ?

Stationary if its **mean and variance are time invariant**

- A stationary time series will be mean reverting in nature, i.e. it will tend to return to its mean and fluctuations around the mean will have roughly equal amplitudes.
- A stationary time series will not drift too far away from its mean because of its finite constant variance.
- A non-stationary time series, on the contrary, will have a time varying variance or a time varying mean or both, and will not tend to revert back to its mean.



Red - Stationary

Blue - Non-Stationary

```
#Making Sure all the years are included

dd = GT1[["Year","ID"]]
all_year = pd.DataFrame({'Year':list(range(1970,2018))})

#Left join your main data on dates data
dd = pd.merge(all_year, dd, on='Year', how="left")
dd.fillna(0, inplace=True)
dd = dd.set_index("Year")
dd
```

```
         ID
Year
1970   98.0
1971   62.0
1972   30.0
1973   27.0
1974   70.0
1975   89.0
1976   97.0
1977  234.0
```

```
1978   306.0
1979   199.0
1980    75.0
1981   107.0
1982   200.0
1983   195.0
1984   224.0
1985   247.0
1986   232.0
1987   415.0
1988   479.0
1989   363.0
1990   382.0
1991   367.0
1992   144.0
1993     0.0
1994    42.0
1995    12.0
1996    10.0
1997     9.0
1998   131.0
1999   163.0
2000   202.0
2001   279.0
2002   157.0
2003   203.0
2004   108.0
2005   156.0
2006   132.0
2007   121.0
2008   307.0
2009   493.0
2010   631.0
2011   493.0
2012   941.0
2013   978.0
2014  1726.0
2015  1429.0
2016  1248.0
2017  1036.0
```

How to test for stationarity?

1. `*Plotting Rolling Statistics*`

2. *Augmented Dickey Fuller test (ADH Test)*

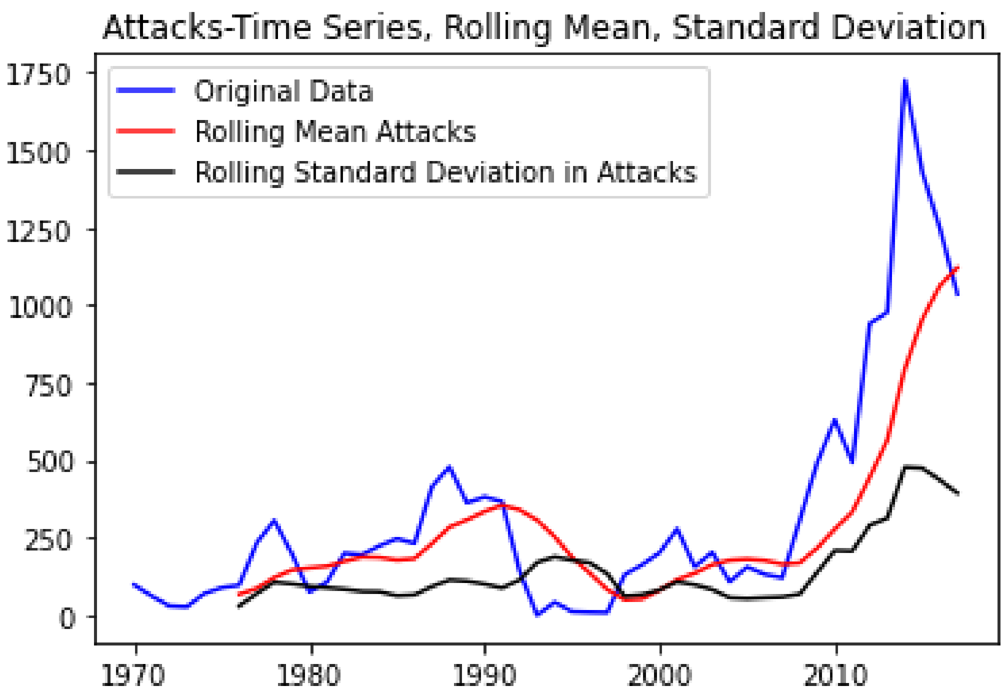
This test will generate critical values and a p-value, which will allow us to accept or reject the null hypothesis that there is no stationarity. *(Null hypothesis: Non Stationary)*

- Reject -> Stationary

- Accept -> Non Stationary

Plotting Rolling Statistics

```
def stat_plot(df):
    rolling_mean = df.rolling(7).mean()
    rolling_std = df.rolling(7).std()
    plt.plot(df, color="blue",label="Original Data")
    plt.plot(rolling_mean, color="red", label="Rolling Mean Attacks")
    plt.plot(rolling_std, color="black", label = "Rolling Standard
Deviation in Attacks")
    plt.title("Attacks-Time Series, Rolling Mean, Standard Deviation")
    plt.legend(loc="best")

stat_plot(dd)
```



Augmented Dickey Fuller test (ADF Test)

```
# ADF Test
def adf(df):
    adft = adfuller(df['ID'], autolag='AIC')
    output_df = pd.DataFrame({"Values":
[adft[0],adft[1],adft[2],adft[3], adft[4]['1%'], adft[4]['5%'],
adft[4]['10%']]  , "Metric":["Test Statistics","p-value","No. of lags
used","Number of observations used",
                                                        "critical
value (1%)", "critical value (5%)", "critical value (10%)"]})
    print(output_df)

adf(dd)
```

```
       Values                          Metric
0  -0.946668                Test Statistics
1   0.772216                        p-value
2   0.000000               No. of lags used
3  47.000000  Number of observations used
4  -3.577848            critical value (1%)
5  -2.925338            critical value (5%)
6  -2.600774           critical value (10%)
```

p-value(0.772216) > 0.05

=>Accept Null Hypothesis

=> Non Stationary

Autocorrelation

- This is a measure of how correlated time series data is at a given point in time with past values, which has huge implications across many industries.
- For example, if our GT data has strong autocorrelation, we can assume that high attack numbers today suggest a strong likelihood that they will be high tomorrow as well.

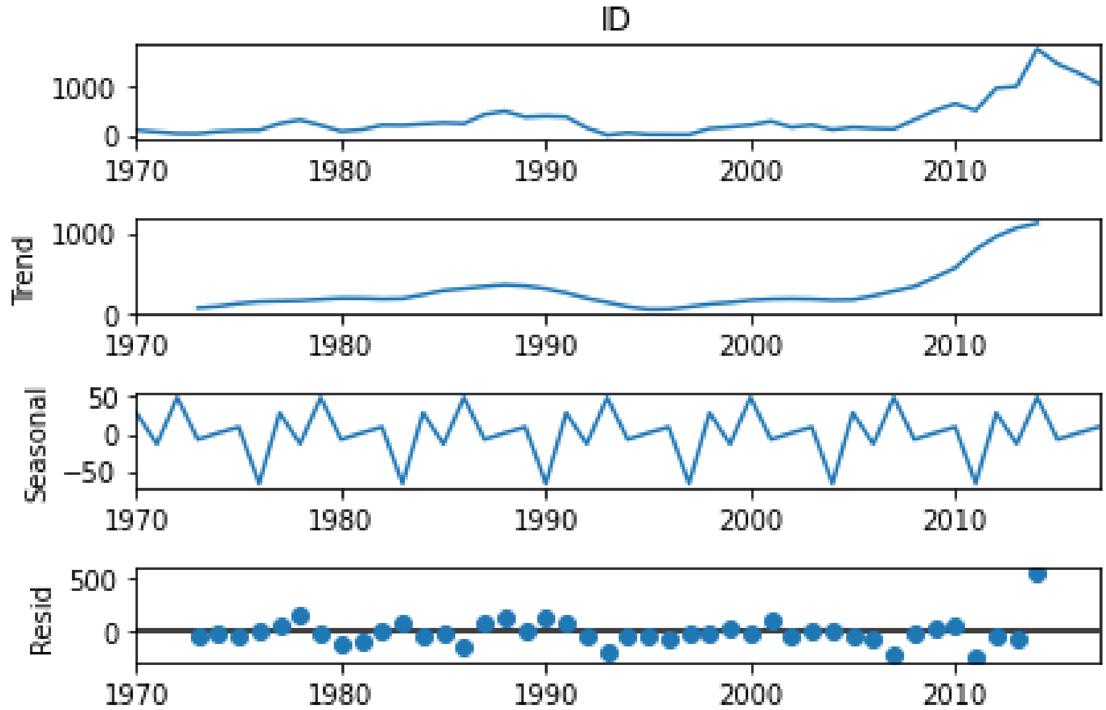
```
autocorrelation_lag1 = dd['ID'].autocorr(lag=1)
print("Lag1: ", autocorrelation_lag1)

autocorrelation_lag3 = dd['ID'].autocorr(lag=3)
print("Lag3: ", autocorrelation_lag3)

autocorrelation_lag6 = dd['ID'].autocorr(lag=6)
print("Lag6: ", autocorrelation_lag6)
```

```
Lag1:  0.9065891322854303
Lag3:  0.6947269909158585
Lag6:  0.35728230161986707
```

=> Corelation decreases long term

Decomposition

```
decompose = seasonal_decompose(dd['ID'],model='additive', period=7)
decompose.plot()
plt.show()
```

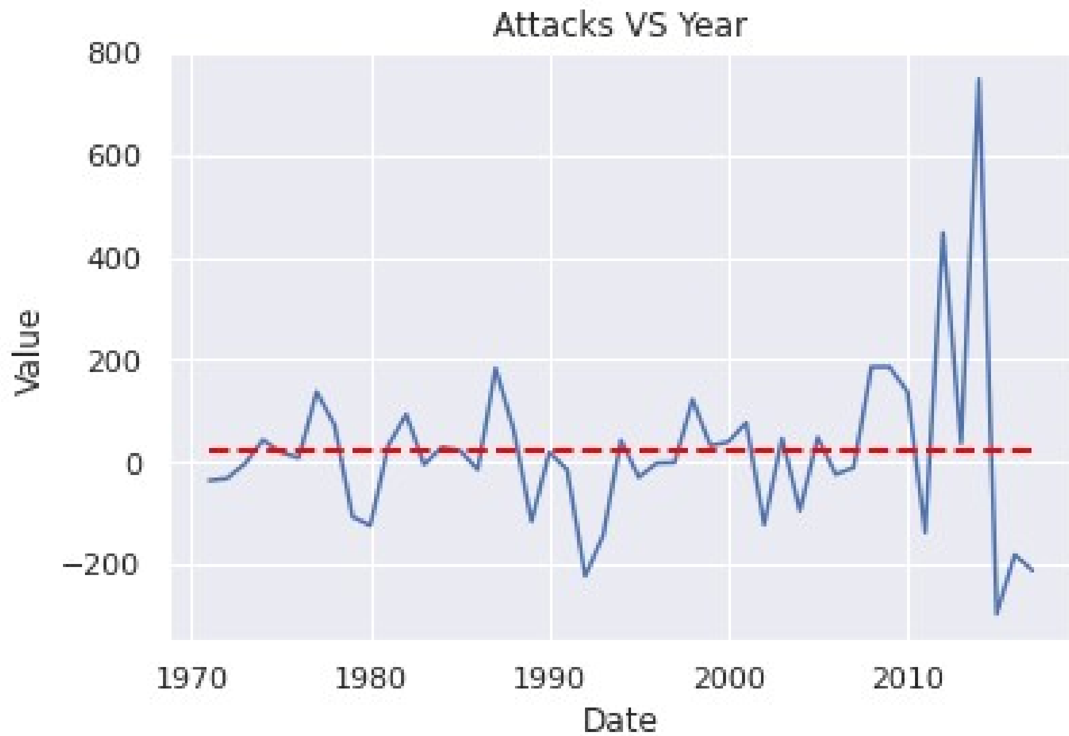


Non-Stationary to Stationary

Making the Difference

```
ddd = dd.diff().dropna()
adf(ddd)
plot_df(ddd, x=ddd.index, y=ddd.ID, title='Attacks VS Year')
```

```
      Values                       Metric
0  -4.089281              Test Statistics
1   0.001009                      p-value
2   2.000000            No. of lags used
3  44.000000  Number of observations used
4  -3.588573          critical value (1%)
5  -2.929886          critical value (5%)
6  -2.603185         critical value (10%)
```

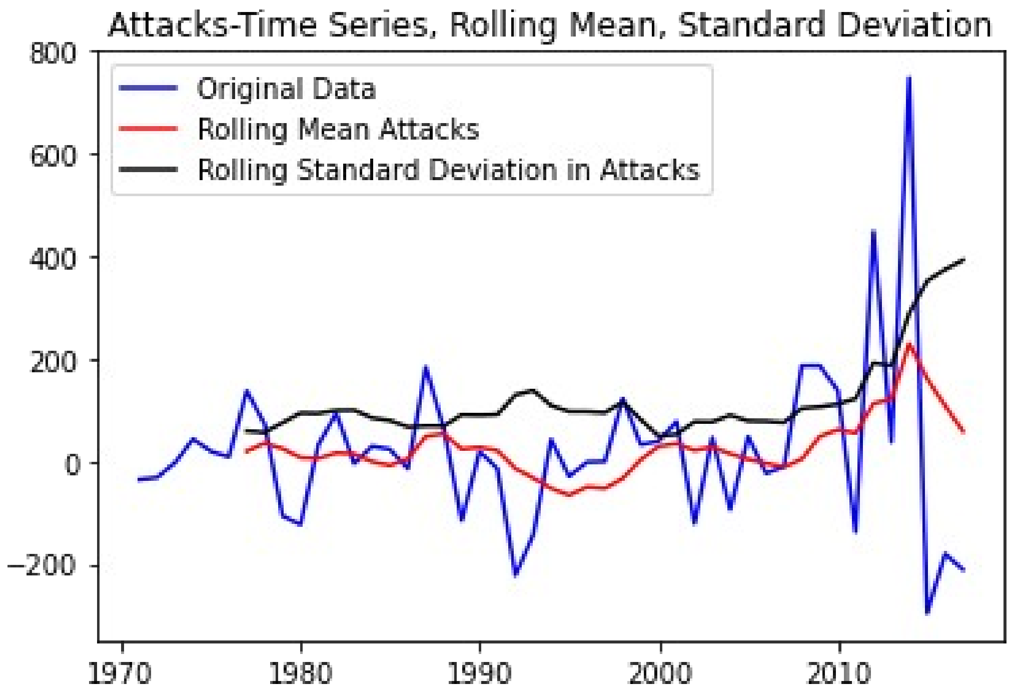


p-value (0.001009) < 0.05

=> Reject Null Hypothesis

=> Stationary

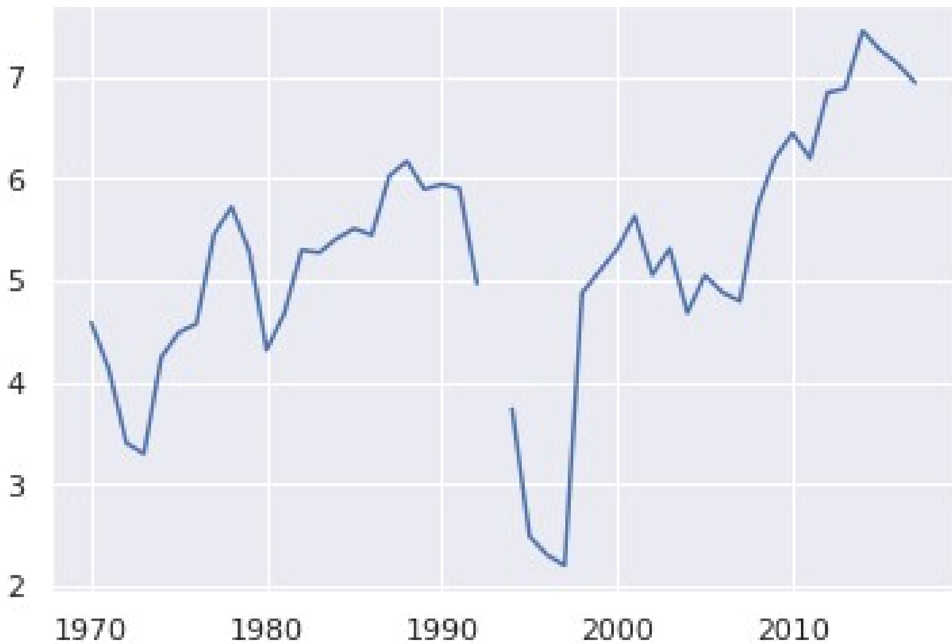
```
stat_plot(ddd)
```



Log Transform - Moving average

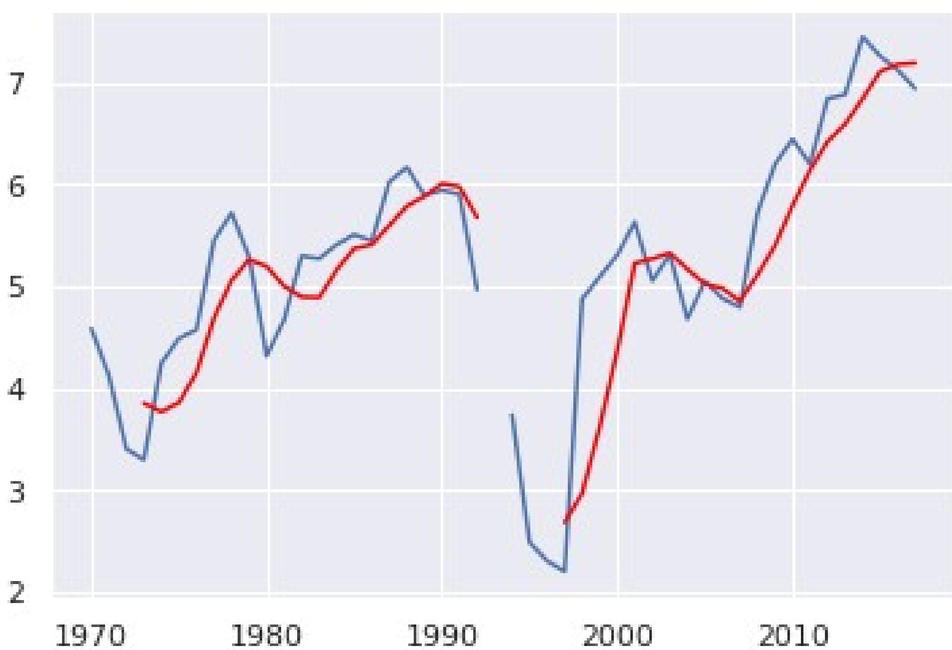
```
dd_log = np.log(dd)
plt.plot(dd_log)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
RuntimeWarning: divide by zero encountered in log
  """Entry point for launching an IPython kernel.

[<matplotlib.lines.Line2D at 0x7f21f15b98d0>]
```




```
moving_avg = dd_log.rolling(4).mean()
plt.plot(dd_log)
plt.plot(moving_avg, color='red')
```

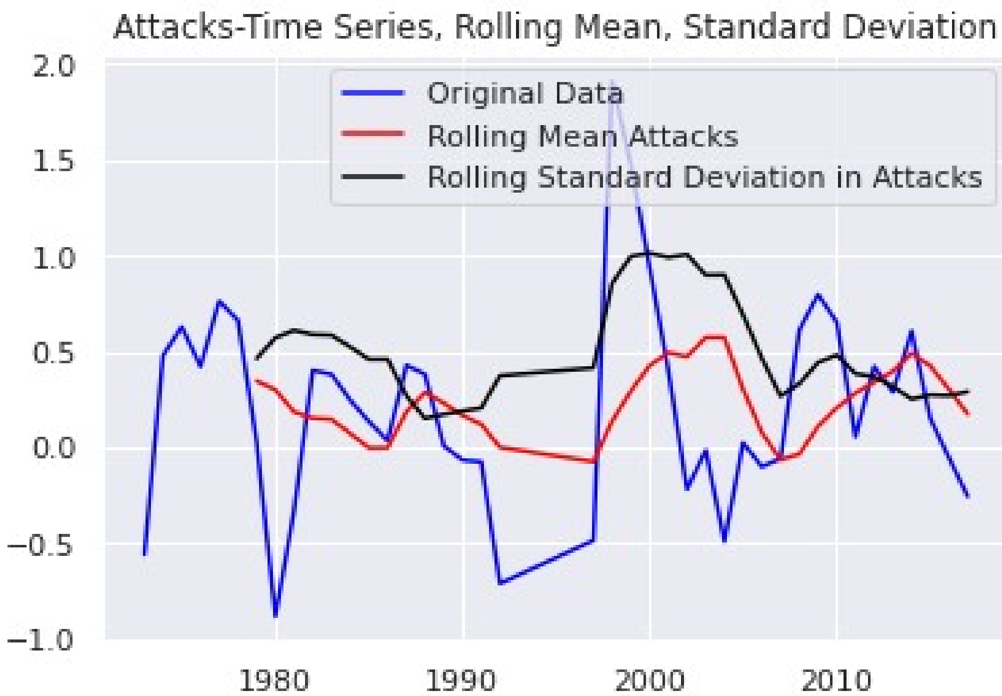
```
[<matplotlib.lines.Line2D at 0x7f21f148bf50>]
```




```
dd_log_moving_avg_diff = dd_log - moving_avg
dd_log_moving_avg_diff.dropna(inplace=True)
```

```
adf(dd_log_moving_avg_diff)
stat_plot(dd_log_moving_avg_diff)
```

```
      Values                       Metric
0  -4.581888              Test Statistics
1   0.000139                      p-value
2   1.000000             No. of lags used
3  39.000000  Number of observations used
4  -3.610400            critical value (1%)
5  -2.939109            critical value (5%)
6  -2.608063           critical value (10%)
```



p-value (0.00139) < 0.05

=> Reject Null Hypothesis

=> Stationary

FORECASTING - ARIMA

Time series forecasting allows us to predict future values in a time series given current and past data

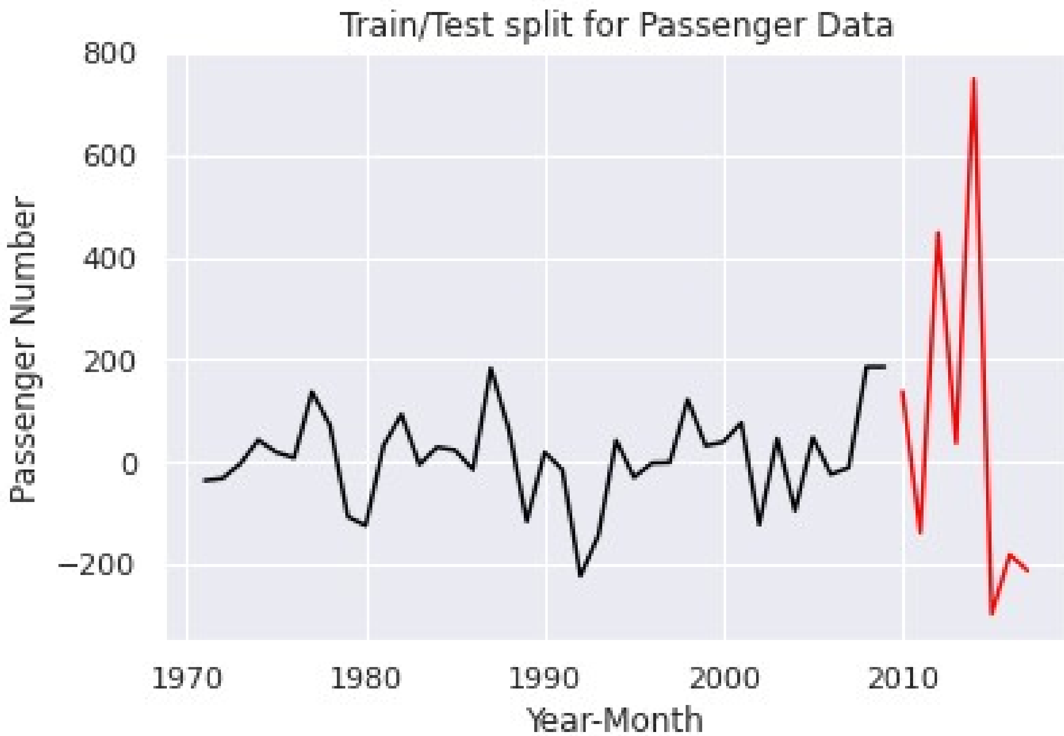
Forecast the number of attacks using ARIMA

auto_arima : Fit best ARIMA model to univariate time series (ensamble)

```
#Train-Test Split
def ttsplit(df):
    df['Year'] = df.index
```

```python
    train = df[df['Year'] < 2010]
    train['train'] = train['ID']
    del train['Year']
    del train['ID']
    test = df[df['Year'] >= 2010]
    del test['Year']
    test['test'] = test['ID']
    del test['ID']
    del df['Year']

    plt.plot(train, color = "black")
    plt.plot(test, color = "red")
    plt.title("Train/Test split for Passenger Data")
    plt.ylabel("Passenger Number")
    plt.xlabel('Year-Month')
    sns.set()
    plt.show()

    return train,test

#1
train, test = ttsplit(ddd)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  # Remove the CWD from sys.path while we load stuff.
```




```
model = auto_arima(train, trace=True, error_action='ignore',
suppress_warnings=True)
#trace - If TRUE, the list of ARIMA models considered will be
reported.
model.fit(train)
forecast = model.predict(n_periods=len(test))
forecast = pd.DataFrame(forecast,index =
test.index,columns=['Prediction'])
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=468.550, Time=0.30 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=464.802, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=465.286, Time=0.06 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=464.474, Time=0.04 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=463.304, Time=0.01 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=464.638, Time=0.11 sec

Best model:  ARIMA(0,0,0)(0,0,0)[0]
Total fit time: 0.541 seconds
```

```
forecast
```

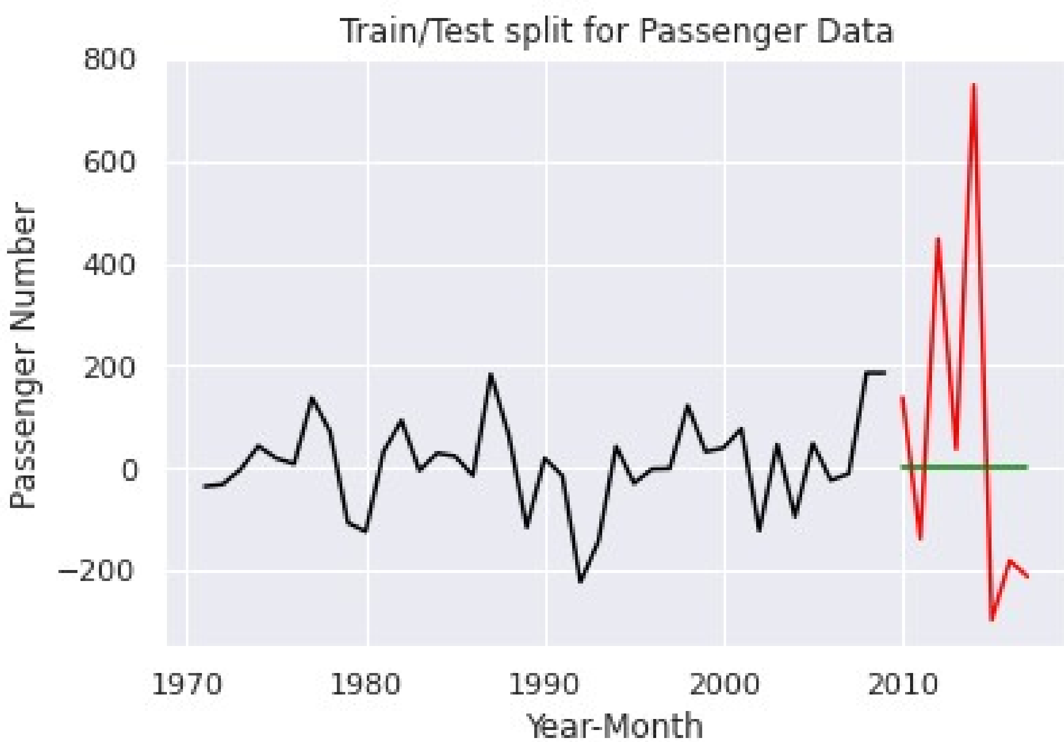
```
      Prediction
Year
2010         0.0
2011         0.0
2012         0.0
```

```
2013         0.0
2014         0.0
2015         0.0
2016         0.0
2017         0.0
```

```
rms = sqrt(mean_squared_error(test,forecast))
print("RMSE: ", rms)
```

```
RMSE:  347.41527744185345
```

```
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.plot(forecast, color = "green")
plt.title("Train/Test split for Passenger Data")
plt.ylabel("Passenger Number")
plt.xlabel('Year-Month')
sns.set()
plt.show()
```

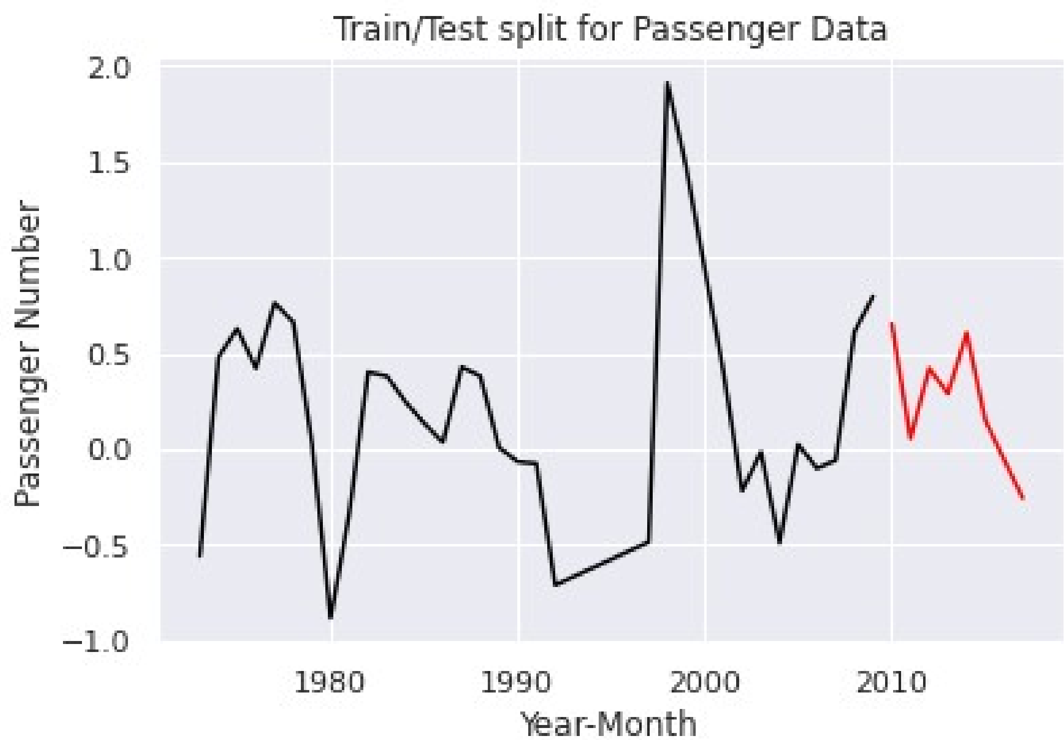


#2

```
train, test = ttsplit(dd_log_moving_avg_diff)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  # Remove the CWD from sys.path while we load stuff.
```




```
model = auto_arima(train, trace=True, error_action='ignore',
suppress_warnings=True)
#trace - If TRUE, the list of ARIMA models considered will be
reported.
model.fit(train)
forecast = model.predict(n_periods=len(test))
forecast = pd.DataFrame(forecast,index =
test.index,columns=['Prediction'])
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.26 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=61.696, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=57.021, Time=0.03 sec
```

```
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=51.570, Time=0.05 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=64.080, Time=0.02 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.14 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.10 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.23 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=inf, Time=0.06 sec

Best model:  ARIMA(0,0,1)(0,0,0)[0] intercept
Total fit time: 0.927 seconds
```

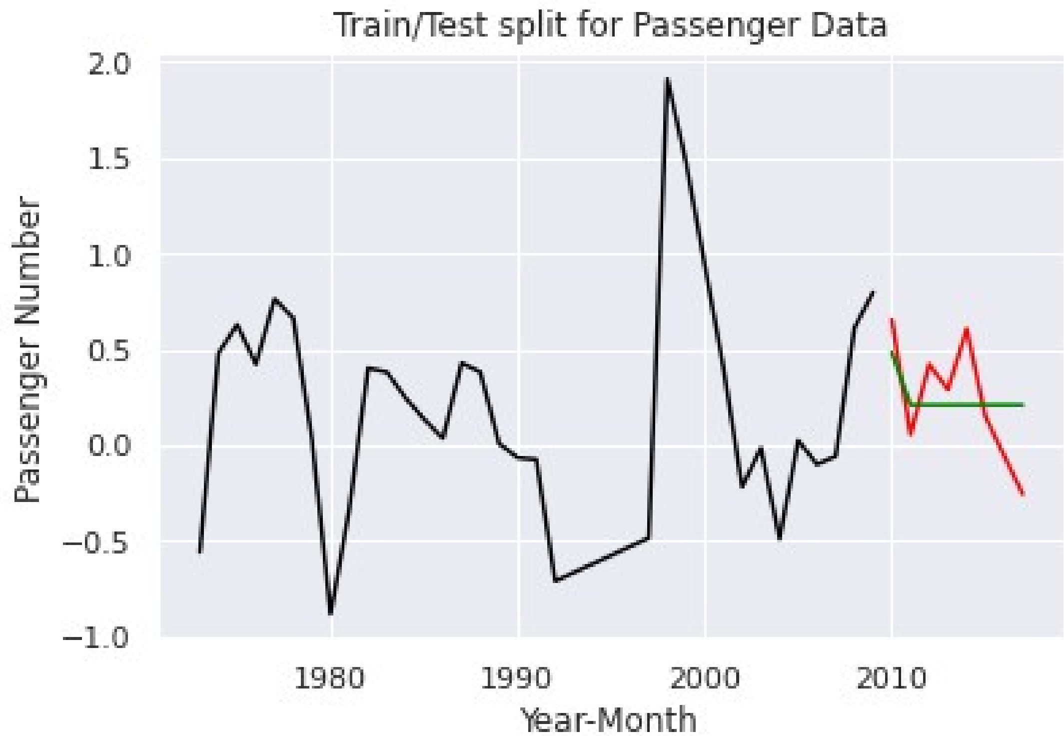
```
forecast
```

```
      Prediction
Year
2010    0.485146
2011    0.210597
2012    0.210597
2013    0.210597
2014    0.210597
2015    0.210597
2016    0.210597
2017    0.210597
```

```
rms = sqrt(mean_squared_error(test,forecast))
print("RMSE: ", rms)
```

```
RMSE:  0.26240347605662645
```

```
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.plot(forecast, color = "green")
plt.title("Train/Test split for Passenger Data")
plt.ylabel("Passenger Number")
plt.xlabel('Year-Month')
sns.set()
plt.show()
```


Train/Test split for Passenger Data
Passenger Number
Year-Month
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
1980
1990
2000
2010