Data Warehousing and Data Mining
Complex Engineering Problem

Import important Libraries and Load dataset

The first step in dealing with any data science problem is to import the important data handling libraries i.e., NumPy and Pandas and then load the respective data set

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

In [2]:
```
terrorism_dataset = pd.read_csv("globalterrorismdb_0718dist.csv",encoding = "ISO-8859-1")
```

Examine Dataset

After loading the dataset, we must examine to know the total number of rows and columns and the amount of NaN values associated with in each column

In [4]:
```
terrorism_dataset
```

Out[4]:

	eventid	iyear	imonth	iday	approxdate	extended	resolution	country	country_txt	region	...	addnotes	scite1	scite2	scite3
0	197000000001	1970	7	2	NaN	0	NaN	58	Dominican Republic	2	...	NaN	NaN	NaN	NaN
1	197000000002	1970	0	0	NaN	0	NaN	130	Mexico	1	...	NaN	NaN	NaN	NaN
2	197001000001	1970	1	0	NaN	0	NaN	160	Philippines	5	...	NaN	NaN	NaN	NaN
3	197001000002	1970	1	0	NaN	0	NaN	78	Greece	8	...	NaN	NaN	NaN	NaN
4	197001000003	1970	1	0	NaN	0	NaN	101	Japan	4	...	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
181686	201712310022	2017	12	31	NaN	0	NaN	182	Somalia	11	...	NaN	"Somalia: Al-Shabaab Militants Attack Army Che...	"Highlights: Somalia Daily Media Highlights 2 ...	"Highlights: Somalia Daily Media Highlights 1 ...
181687	201712310029	2017	12	31	NaN	0	NaN	200	Syria	10	...	NaN	"Putin's 'victory' in Syria has turned into a ...	"Two Russian soldiers killed at Hmeymim base i...	"Two Russian servicemen killed in Syria mortar...
181688	201712310030	2017	12	31	NaN	0	NaN	160	Philippines	5	...	NaN	"Maguindanao clashes trap tribe members," Phil...	NaN	NaN
181689	201712310031	2017	12	31	NaN	0	NaN	92	India	6	...	NaN	"Trader escapes grenade attack in Imphal," Bus...	NaN	NaN
181690	201712310032	2017	12	31	NaN	0	NaN	160	Philippines	5	...	NaN	"Security tightened in Cotabato following IED ...	"Security tightened in Cotabato City," Manila ...	NaN

181691 rows × 135 columns

Data Cleaning

Data cleaning or Data cleansing is very important from the perspective of building intelligent automated systems. Data cleansing is a preprocessing step that improves the data validity, accuracy, completeness, consistency, and uniformity. It is essential for building reliable machine learning models that can produce good results. Otherwise, no matter how good the model is, its results cannot be trusted. In short, data cleaning means fixing bad data in your data set. Bad data could be:

1. Empty cells
2. Data in wrong format
3. Wrong data
4. Duplicates

So initially we apply a loop to filter out the columns whose 1/3rd values are NaN

```
td2=terrorism_dataset
for i in terrorism_dataset.columns:
    x =terrorism_dataset[f"{i}"].isna().sum()
    if(x>50000):
        td2 = td2.drop(f"{i}",axis=1)
td2
```

Secondly, we remove repetitive columns from our data set i.e. In our data set we have one column with title country and other with country_txt both provide same information however one is numeric and other is alphabetic so we will get rid of one

```
td2.drop("attacktype1_txt",axis=1,inplace=True)
td2.drop("targtype1_txt",axis=1,inplace=True)
td2.drop("targsubtype1_txt",axis=1,inplace=True)
td2.drop("natlty1_txt",axis=1,inplace=True)
td2.drop("region_txt",axis=1,inplace=True)
td2.drop("country_txt",axis=1,inplace=True)
td2.drop("weaptype1_txt",axis=1,inplace=True)
td2.drop("weapsubtype1_txt",axis=1,inplace=True)
```

Thirdly we will remove columns that are not giving much information regarding our defined task after analysis of dataset

```
td2.drop("INT_LOG",axis=1,inplace=True)
td2.drop("INT_IDEO",axis=1,inplace=True)
td2.drop("INT_MISC",axis=1,inplace=True)
td2.drop("INT_ANY",axis=1,inplace=True)
td2.drop("dbsource",axis=1,inplace=True)
td2.drop("corp1",axis=1,inplace=True)
td2.drop("ishostkid",axis=1,inplace=True)
td2.dropna(subset=["doubtterr"],axis=0 , inplace=True)
td2.dropna(subset=["multiple"],axis=0 , inplace=True)
td2.dropna(subset=["specificity"],axis=0 , inplace=True)

td2.drop("guncertain1",axis=1,inplace=True)
td2.drop("longitude",axis=1,inplace=True)
td2.drop("latitude",axis=1,inplace=True)
td2.drop("targsubtype1",axis=1,inplace=True)
td2.drop("weapsubtype1",axis=1,inplace=True)
```

Our data consist upon more than 180000 rows hence getting rid of few row that hold NaN values will not affect the data set much hence we will remove such rows too

```
td2.dropna(subset=["provstate"],axis=0 , inplace=True)
td2.dropna(subset=["city"],axis=0 , inplace=True)
```

Lastly, we will fill NaN values of some important columns calculating their mean or mode or hardcode some value based on our analysis

```
td2["nwound"].fillna(3,inplace=True)
td2["nkill"].fillna(2,inplace=True)
td2.natlty1.fillna(0,inplace=True)
td2.target1.fillna("Unknown",inplace=True)
```

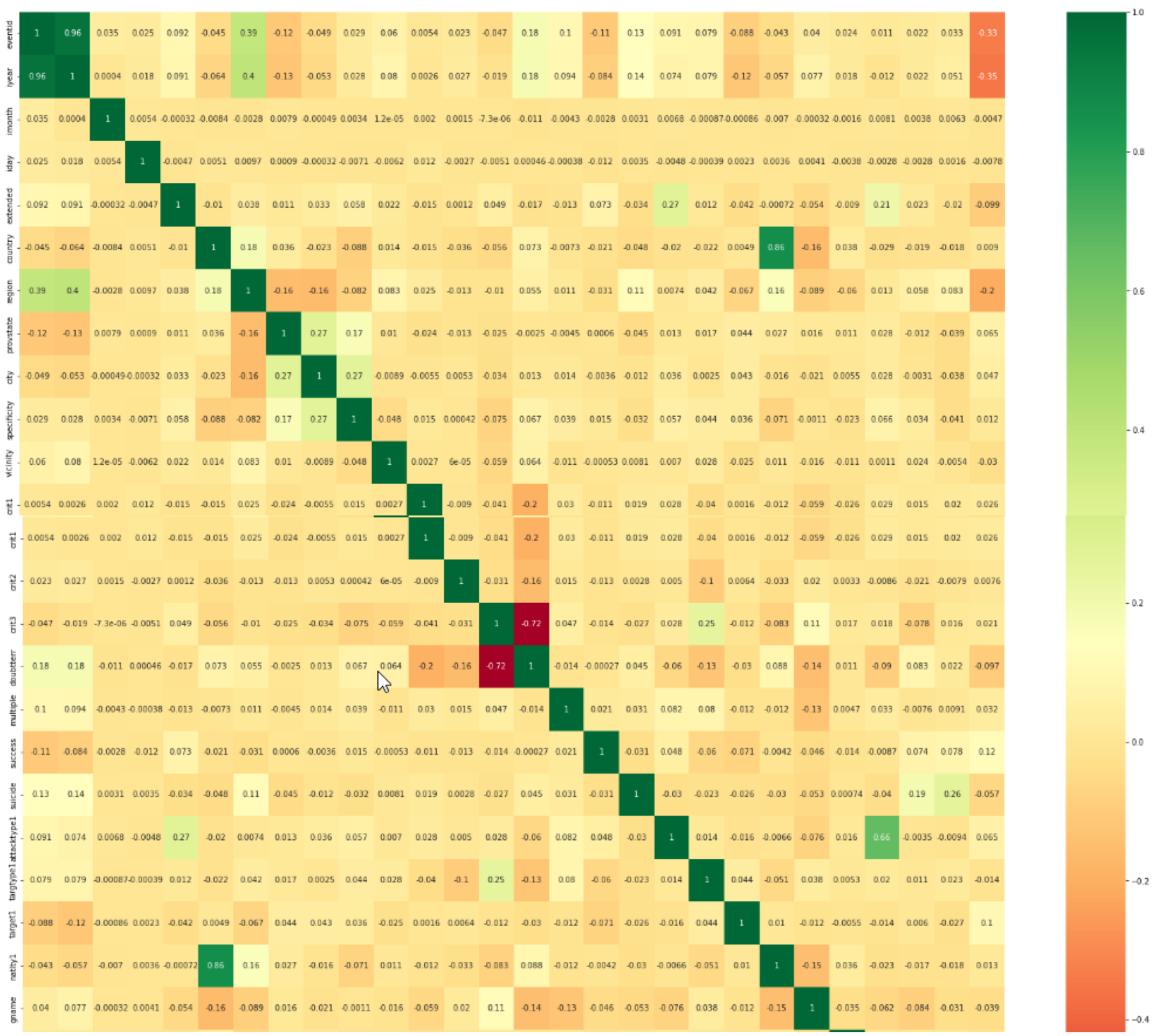
Apply Label Encoder

Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

```python
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
encode_result = td2.apply(label_encoder.fit_transform)
```

Feature Selection

Correlation states how the features are related to each other or the target variable. Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable) Heatmap makes it easy to identify which features are most related to the target variable.



Apply Machine Learning Model

Firstly, we define x with set of features that will be input for our model and y with the output feature, in our case we are supposed to predict the attack type therefore we will put that in y and some other selected features in x

```
x=encode_result[["iyear","country","region","city","provstate","gname","target1","weaptype1"]]
y=encode_result["attacktype1"]
```

Secondly, we Apply train_test_split to x and y The train-test split is used to estimate the performance of machine learning algorithms that are applicable for prediction-based Algorithms/Applications. This method is a fast and easy procedure to perform such that we can compare our own machine learning model results to machine results.

```
from sklearn.model_selection import train_test_split
np.random.seed(42)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
```

Lastly, we setup ML model and train it and see the score to know its accuracy

```
In [13]: # Setup model
         from sklearn.ensemble import RandomForestClassifier
         model = RandomForestClassifier()


         model.fit(x_train, y_train)
         model.score(x_test, y_test)

Out[13]: 0.8342642260686833
```

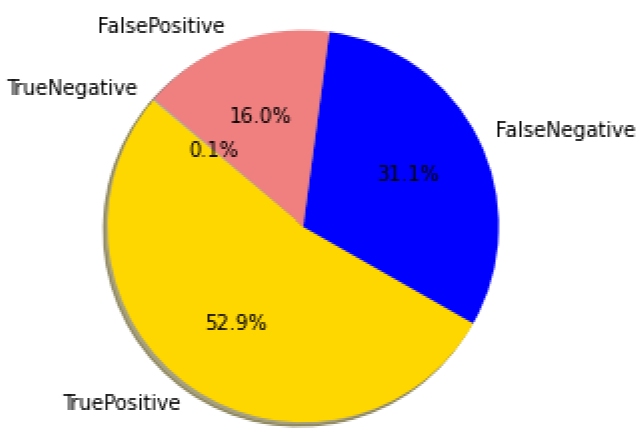
Confusion Matrix

A confusion matrix is a table that is used to define the performance of a classification algorithm. A confusion matrix visualizes and summarizes the performance of a classification algorithm.

```
from sklearn.metrics import confusion_matrix
rclassifier_pred = model.predict(x_test)
a=confusion_matrix(y_test,rclassifier_pred)
a1=a.flatten()
x=a1[0:4]
print(x)
```

```
[1871 1100  565    2]
```

```
labels = 'TruePositive', 'FalseNegative', 'FalsePositive', 'TrueNegative'
colors = ['gold', 'blue', 'lightcoral', 'lightskyblue']

# Plot
plt.pie(x, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```



Spatial-Temporal Analysis Using Clustering

Feature scaling using MinMaxScaler.

Feature scaling through standardization can be an important preprocessing step for many machine learning algorithms. Standardization involves rescaling the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.

The MinMaxscaler is a type of scaler that scales the minimum and maximum values to be 0 and 1 respectively. While the StandardScaler scales all values between min and max so that they fall within a range from min to max.

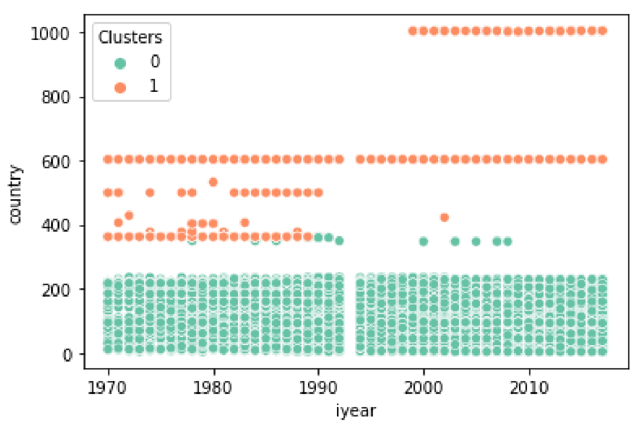
```
from sklearn.preprocessing import normalize
from sklearn.preprocessing import MinMaxScaler
scalar = MinMaxScaler()

scale = scalar.fit_transform(td2[['iyear','country']])
df_scale = pd.DataFrame(scale, columns = ['iyear','country']);
df_scale.head(5)
```

K-Mean Algorithm

K-means Algorithm is an Iterative algorithm that divides a group of n datasets into k subgroups or clusters based on the similarity and their mean distance from the centroid of that subgroup formed.

```
: from sklearn.cluster import KMeans
import sklearn.cluster as cluster
km=KMeans(n_clusters=2)
y_predicted = km.fit_predict(td2[['iyear','region','attacktype1','country','imonth','iday','nkill','nwound']])
td2['Clusters'] = km.labels_
sns.scatterplot(x="iyear", y="country",hue = 'Clusters',
data=td2,palette='Set2')
```

```
: <AxesSubplot:xlabel='iyear', ylabel='country'>
```

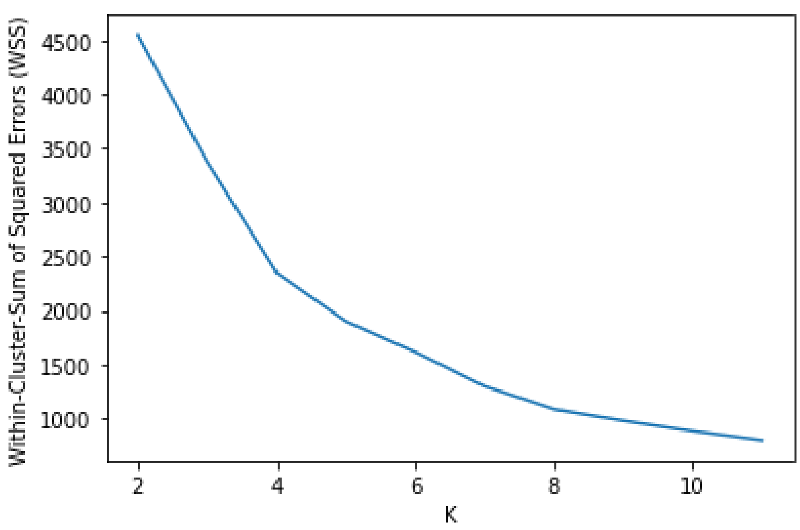


Optimum number of Clusters in K Means

Imagine we set k to its maximum value n (where n is number of samples) each sample will form its own cluster meaning sum of squared distances equals zero. Below is a plot of sum of squared distances for k in the range specified above. If the plot looks like an arm, then the elbow on the arm is optimal k.

```
#Finding optimum value of K
K=range(2,12)
wss = []
for k in K:
    kmeans=cluster.KMeans(n_clusters=k)
    kmeans=kmeans.fit(df_scale)
    wss_iter = kmeans.inertia_
    wss.append(wss_iter)

import matplotlib.pyplot as plt
#Plotting the graph
plt.xlabel('K')
plt.ylabel('Within-Cluster-Sum of Squared Errors (WSS)')
plt.plot(K,wss)
```

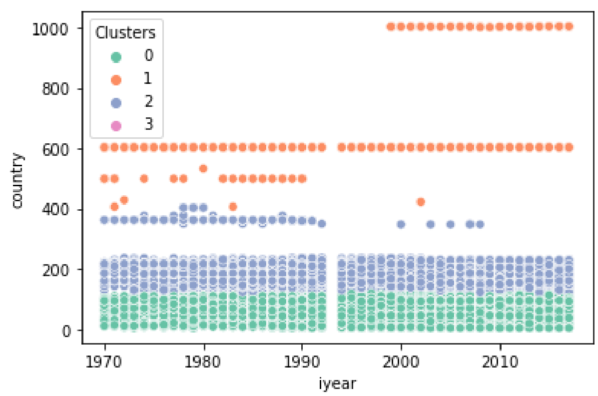
```
[<matplotlib.lines.Line2D at 0x17c869fc7c0>]
```



Applying Optimum value of K for Clustering

```
km=KMeans(n_clusters=4)
y_predicted = km.fit_predict(td2[['iyear','region','attacktype1','country','imonth','iday','nkill','nwound']])
y_predicted
td2['Clusters'] = km.labels_
sns.scatterplot(x="iyear", y="country",hue = 'Clusters',
data=td2,palette='Set2')
```

```
<AxesSubplot:xlabel='iyear', ylabel='country'>
```



Description of Graph

The Graph show clustering based on Space and time with time along x axis and country code along y axis the list pf countries along with their codes has been attached below.

4 = Afghanistan	22 = Belize	41 = Central African Republic
5 = Albania	23 = Benin	42 = Chad
6 = Algeria	24 = Bermuda*	43 = Chile
7 = Andorra	25 = Bhutan	44 = China
8 = Angola	26 = Bolivia	45 = Colombia
10 = Antigua and Barbuda	28 = Bosnia-Herzegovina	46 = Comoros
11 = Argentina	29 = Botswana	47 = Republic of the Congo
12 = Armenia	30 = Brazil	49 = Costa Rica
14 = Australia	31 = Brunei	50 = Croatia
15 = Austria	32 = Bulgaria	51 = Cuba
16 = Azerbaijan	33 = Burkina Faso	53 = Cyprus
17 = Bahamas	34 = Burundi	54 = Czech Republic
18 = Bahrain	35 = Belarus	55 = Denmark
19 = Bangladesh	36 = Cambodia	56 = Djibouti
20 = Barbados	37 = Cameroon	57 = Dominica
21 = Belgium	38 = Canada	58 = Dominican Republic

59 = Ecuador
60 = Egypt
61 = El Salvador
62 = Equatorial Guinea
63 = Eritrea
64 = Estonia
65 = Ethiopia
66 = Falkland Islands
67 = Fiji
68 = Finland
69 = France
70 = French Guiana
71 = French Polynesia
72 = Gabon
73 = Gambia
74 = Georgia
75 = Germany
76 = Ghana
78 = Greece
79 = Greenland*
80 = Grenada
81 = Guadeloupe
83 = Guatemala
84 = Guinea
85 = Guinea-Bissau
86 = Guyana
87 = Haiti
88 = Honduras
89 = Hong Kong
90 = Hungary
91 = Iceland
92 = India
93 = Indonesia
94 = Iran
95 = Iraq
96 = Ireland
97 = Israel
98 = Italy
99 = Ivory Coast
100 = Jamaica
101 = Japan
102 = Jordan
103 = Kazakhstan
104 = Kenya
106 = Kuwait
107 = Kyrgyzstan
108 = Laos
109 = Latvia
110 = Lebanon
111 = Lesotho
112 = Liberia
113 = Libya
114 = Liechtenstein*
115 = Lithuania
116 = Luxembourg
117 = Macau
118 = Macedonia
119 = Madagascar
120 = Malawi
121 = Malaysia
122 = Maldives
123 = Mali
124 = Malta
125 = Man, Isle of*
126 = Marshall Islands*
127 = Martinique
128 = Mauritania
129 = Mauritius
130 = Mexico
132 = Moldova
134 = Mongolia*
136 = Morocco
137 = Mozambique
138 = Myanmar
139 = Namibia
141 = Nepal
142 = Netherlands
143 = New Caledonia
144 = New Zealand
145 = Nicaragua
146 = Niger
147 = Nigeria
149 = North Korea
151 = Norway
152 = Oman*
153 = Pakistan
155 = West Bank and Gaza Strip
156 = Panama
157 = Papua New Guinea
158 = Paraguay
159 = Peru
160 = Philippines
161 = Poland
162 = Portugal
163 = Puerto Rico*
164 = Qatar
166 = Romania
167 = Russia
168 = Rwanda
169 = Saba (Netherlands Antilles)*
173 = Saudi Arabia
174 = Senegal
175 = Serbia-Montenegro
176 = Seychelles
177 = Sierra Leone
178 = Singapore
179 = Slovak Republic
180 = Slovenia
181 = Solomon Islands
182 = Somalia
183 = South Africa
184 = South Korea
185 = Spain
186 = Sri Lanka
189 = St. Kitts and Nevis
190 = St. Lucia
192 = St. Martin*
195 = Sudan
196 = Suriname
197 = Swaziland
198 = Sweden
199 = Switzerland
200 = Syria
201 = Taiwan
202 = Tajikistan
203 = Tanzania
204 = Togo
205 = Thailand
206 = Tonga*
207 = Trinidad and Tobago
208 = Tunisia

20

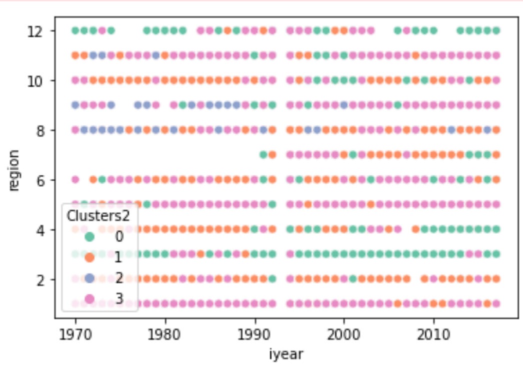
209 = Turkey
210 = Turkmenistan
212 = Tuvalu*
213 = Uganda
214 = Ukraine
215 = United Arab Emirates
216 = Great Britain*
217 = United States
218 = Uruguay
219 = Uzbekistan
220 = Vanuatu
221 = Vatican City
222 = Venezuela
223 = Vietnam
225 = Virgin Islands (U.S.)*
226 = Wallis and Futuna
228 = Yemen
229 = Democratic Republic of the Congo
230 = Zambia
231 = Zimbabwe
233 = Northern Ireland*
235 = Yugoslavia
236 = Czechoslovakia
238 = Corsica*
334 = Asian*
347 = East Timor
349 = Western Sahara
351 = Commonwealth of Independent States*
359 = Soviet Union
362 = West Germany (FRG)
377 = North Yemen
403 = Rhodesia
406 = South Yemen
422 = International
428 = South Vietnam
499 = East Germany (GDR)
520 = Sinhalese*
532 = New Hebrides
603 = United Kingdom
604 = Zaire
605 = People's Republic of the Congo
999 = Multinational*
1001 = Serbia
1002 = Montenegro
1003 = Kosovo
1004 = South Sudan

More clustering on optimal k values for spatial temporal analysis

```
km=KMeans(n_clusters=4)
y_predicted = km.fit_predict(td2[['iyear','region','attacktype1','country','imonth','iday']])
td2['Clusters2'] = km.labels_
sns.scatterplot(x="iyear", y="region",hue = 'Clusters2',
data=td2,palette='Set2')
```

```
<AxesSubplot:xlabel='iyear', ylabel='region'>
```

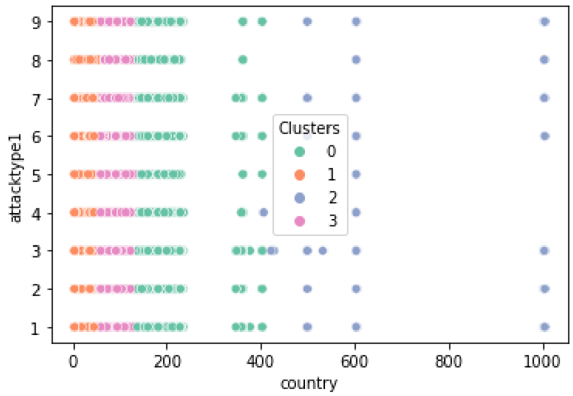
```
C:\Users\Dell\sample_project_1\env\lib\site-packages\IPython\core\pylabtools.py:151: UserWarni
t" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



```
km=KMeans(n_clusters=4)
y_predicted = km.fit_predict(td2[['iyear','region','attacktype1','country','imonth','iday','nkill','nwound']])
td2['Clusters'] = km.labels_
sns.scatterplot(x="country", y="attacktype1",hue = 'Clusters',
data=td2,palette='Set2')
```

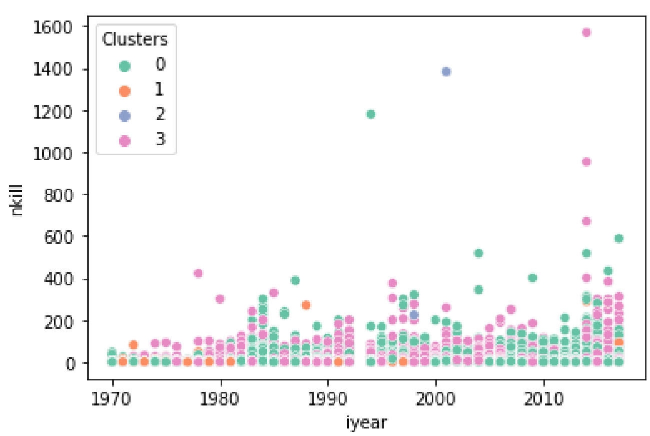
```
<AxesSubplot:xlabel='country', ylabel='attacktype1'>
```

```
C:\Users\Dell\sample_project_1\env\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Creating legend
t" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```




```
km=KMeans(n_clusters=4)
y_predicted = km.fit_predict(td2[['iyear','region','attacktype1','country','imonth','iday','nkill','nwound']])
td2['Clusters'] = km.labels_
sns.scatterplot(x="iyear", y="nkill",hue = 'Clusters',
data=td2,palette='Set2')
```

```
<AxesSubplot:xlabel='iyear', ylabel='nkill'>
```



```python
km=KMeans(n_clusters=4)
y_predicted = km.fit_predict(td2[['iyear','region','attacktype1','country','imonth','iday','nkill','nwound']])
td2['Clusters'] = km.labels_
sns.scatterplot(x="region", y="nkill",hue = 'Clusters',
data=td2,palette='Set2')
```

```
<AxesSubplot:xlabel='region', ylabel='nkill'>
```