Capstone Project (BDA 106)

By Mahwish Khalid Disaster Dataset with python

To start exploring data, we need to start by actually loading our data.Pandas library, make it easy task for programmers,so we have imported the package as "panda as pd", also we have used the read_csv() function, to which we have passed the directory in which the data can be found. We need to make sure that or data is is read correctly. ¶

Import Libraries

In [1]:
```python
import numpy
import pandas as pd
import scipy
import matplotlib
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
sns.set(style="darkgrid")
import csv
# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

```

Importing The Data

In [2]:
```
# this for the operation system
import os
# to show the directory that related to the Data
os.getcwd()
```

Out[2]: 'C:\\Users\\Mahwish\\Desktop\\mc master university\\BDA 106\\submission to professor'

Load in the data with read_csv()

In [3]:
```
# open CSV file including the direcotry
df = pd.read_csv("Data_em1.csv")
```

What is Exploratory Data Analysis (EDA)?

```
Exploratory data analysis (EDA) is a crucial component of data science which allows us to develop the gist
of what our data look like.EDA is use to test business assumptions, generate hypotheses for further
analysis. On the other hand, we can also use it to prepare our data for modeling. So good knowledge of
data to either get the answers that we need or to develop  for interpreting the results of future
modeling.so for acheiving our purpose we should know basic basic description of the data, visualize it and
identify patterns in it.
```

Basic Description of the Data

1)Shape of dataset

It shows the number of columns and rows in our data set

In [4]:
```
df.shape
```

Out[4]: (2734, 12)

Note:-

Columns=12 Rows=2734

2)Describing The Data

We use the describe() function to get various summary statistics that exclude NaN values.

In [5]:
```
1 df = df.rename(columns={"Total damage  ('000 US$)": "Total damage"})
```

In [6]:
```
1  print (df.describe())
```

```
              Year   occurrence   Total deaths       Injured      Affected  \
count  2734.000000  2734.000000    2043.000000  8.710000e+02  1.465000e+03
mean   1989.504755     1.876737     436.892805  3.648246e+03  2.809973e+05
std      23.853371     2.118296    5555.286696  6.285201e+04  2.497475e+06
min    1900.000000     1.000000       1.000000  1.000000e+00  1.000000e+00
25%    1980.000000     1.000000      10.500000  2.000000e+01  2.000000e+03
50%    1996.000000     1.000000      32.000000  6.400000e+01  1.300000e+04
75%    2007.000000     2.000000     105.000000  2.180000e+02  7.618600e+04
max    2018.000000    27.000000  222570.000000  1.800000e+06  8.501880e+07

           Homeless  Total affected  Total damage
count  4.910000e+02    1.925000e+03  9.580000e+02
mean   2.528397e+04    2.219497e+05  1.478785e+06
std    8.997038e+04    2.185625e+06  9.060460e+06
min    1.000000e+01    1.000000e+00  3.000000e+00
25%    4.500000e+02    4.000000e+02  6.000000e+03
50%    2.092000e+03    5.650000e+03  5.100000e+04
75%    1.300000e+04    4.771400e+04  4.000000e+05
max    1.166000e+06    8.501947e+07  1.711100e+08
```

Note:-

So we can see that this function returns the count, mean, standard deviation, minimum and maximum values and the quantiles of the data.

3)First and Last DataFrame Rows

Now that you have got a general idea about our data set, it’s also a good idea to take a closer look at the data itself. With the help of the head() and tail() functions of the Pandas library, we can easily check out the first and last lines of our DataFrame, respectively.

In [7]:
```
print(df.head(5))

```

```
   Year  ISO              country_name Disaster disaster subgroup  occurrence  \
0  1900  JAM                   Jamaica  Natural        Biological           1
1  1900  JAM                   Jamaica  Natural      Hydrological           1
2  1900  USA  United States of America  Natural    Meteorological           1
3  1902  SLV               El Salvador  Natural       Geophysical           1
4  1902  GTM                 Guatemala  Natural       Geophysical           3

   Total deaths  Injured  Affected  Homeless  Total affected  Total damage
0          30.0      NaN       NaN       NaN             NaN           NaN
1         300.0      NaN       NaN       NaN             NaN           NaN
2        6000.0      NaN       NaN       NaN             NaN       30000.0
3         185.0      NaN       NaN       NaN             NaN           NaN
4        9000.0      NaN       NaN       NaN             NaN           NaN
```

In [8]:
```
print(df.tail(5))
```

```
      Year  ISO                        country_name       Disaster  \
2729  2018  USA            United States of America        Natural
2730  2018  USA            United States of America  Technological
2731  2018  URY                             Uruguay        Natural
2732  2018  VEN  Venezuela (Bolivarian Republic of)        Natural
2733  2018  VEN  Venezuela (Bolivarian Republic of)  Technological

     disaster subgroup  occurrence  Total deaths  Injured  Affected  Homeless  \
2729    Meteorological           6         136.0      NaN       NaN       NaN
2730     Technological           2          37.0      NaN       NaN       NaN
2731    Climatological           1           NaN      NaN   11135.0       NaN
2732      Hydrological           2           2.0      NaN   10700.0       NaN
2733     Technological           1          17.0      NaN       NaN       NaN

      Total affected  Total damage
2729             NaN    12800000.0
2730             NaN           NaN
2731         11135.0      500000.0
2732         10700.0           NaN
2733             NaN           NaN
```

4) Sampling The Data

If we have a large dataset, we might consider taking a sample of our data as an easy way to get a feel for our data quickly. As a first and easy way to do this, we can make use of the sample() function.

In [9]:
```
print(df.sample(5))
```

```
      Year  ISO  country_name      Disaster disaster subgroup  occurrence  \
1198  1992  URY       Uruguay  Technological     Technological           1
932   1987  PER          Peru        Natural      Hydrological           4
2535  2015  BRA        Brazil  Technological     Technological           4
410   1967  NIC     Nicaragua  Technological     Technological           1
1521  1998  PRI   Puerto Rico        Natural    Meteorological           1

      Total deaths  Injured  Affected  Homeless  Total affected  Total damage
1198          20.0     40.0       NaN       NaN            40.0           NaN
932          268.0    564.0   27500.0     840.0         28904.0           NaN
2535          93.0    221.0       NaN     637.0           858.0           NaN
410            NaN      NaN     130.0       NaN           130.0         500.0
1521           NaN      NaN       NaN       NaN             NaN     1750000.0
```

5)Column Name

In [10]:
```
print(df.columns)
```

```
Index(['Year', 'ISO', ' country_name', 'Disaster', 'disaster subgroup',
       'occurrence', 'Total deaths', 'Injured', 'Affected', 'Homeless',
       'Total affected', 'Total damage'],
      dtype='object')
```

Note:-

```
The total affected is the sum of the injured + Affected + homeless
```

6)Data Type

In [11]:
```
print (df.dtypes)
```

```
Year                   int64
ISO                   object
 country_name         object
Disaster              object
disaster subgroup     object
occurrence             int64
Total deaths         float64
Injured              float64
Affected             float64
Homeless             float64
Total affected       float64
Total damage         float64
dtype: object
```

7)The summary for categorical value

This will shows thesummary of categorical variables.

In [12]:
```
categorical = df.dtypes[df.dtypes == "object"].index
print(categorical)
df[categorical].describe()
```

```
Index(['ISO', ' country_name', 'Disaster', 'disaster subgroup'], dtype='object')
```

Out[12]:

	ISO	country_name	Disaster	disaster subgroup
count	2734	2734	2734	2734
unique	50	50	3	7
top	USA	United States of America	Natural	Hydrological
freq	304	304	2111	722

The Challenges of Data

Now I have gathered some basic information on my data, it’s a good idea to just go a little bit deeper into the challenges that our data might pose.Now I am finding the irrugularities in the dataset because missing values finding is another important part of analysis.

1)Missing Values

Checking missing values is the important part when we are exploring our dataset is whether or not the data set has any missing values because the dataset can lose expressiveness, which can lead to weak or biased analyses if we do not treat missing values. The chances of our classification or predictions for the data is increased if we treat missing values.So NaN counts for each column tells us the total number of missing values in specific column.

Note:-

Total deaths,Injured,Affected,Homeless,Total Affected & Total damages have number of missing values.Now we are planning to treat the missing values we need to follow some strategies.like mean,median,mode,0 ,backfill or manymore.Here I use fillna() function.

In [13]:
```
df.isna().sum()
```

Out[13]:
```
Year                    0
ISO                     0
 country_name           0
Disaster                0
disaster subgroup       0
occurrence              0
Total deaths          691
Injured              1863
Affected             1269
Homeless             2243
Total affected        809
Total damage         1776
dtype: int64
```

2)Treating missing values in Total damage:-

For doing this here I find the mean of total damage column then I p;aced 70% of this mean value will be filled to NaN values.

In [14]:
```
df["Total damage"]=df["Total damage"].fillna(df["Total damage"].mean()*0.70)
```

In [15]:
```
1 df.head()
```

Out[15]:

	Year	ISO	country_name	Disaster	disaster subgroup	occurrence	Total deaths	Injured	Affected	Homeless	Total affected	Total damage
0	1900	JAM	Jamaica	Natural	Biological	1	30.0	NaN	NaN	NaN	NaN	1.035150e+06
1	1900	JAM	Jamaica	Natural	Hydrological	1	300.0	NaN	NaN	NaN	NaN	1.035150e+06
2	1900	USA	United States of America	Natural	Meteorological	1	6000.0	NaN	NaN	NaN	NaN	3.000000e+04
3	1902	SLV	El Salvador	Natural	Geophysical	1	185.0	NaN	NaN	NaN	NaN	1.035150e+06
4	1902	GTM	Guatemala	Natural	Geophysical	3	9000.0	NaN	NaN	NaN	NaN	1.035150e+06

In [16]:
```
1 df.isna().sum()
```

Out[16]:
```
Year                    0
ISO                     0
 country_name           0
Disaster                0
disaster subgroup       0
occurrence              0
Total deaths          691
Injured              1863
Affected             1269
Homeless             2243
Total affected        809
Total damage            0
dtype: int64
```

3)Treating the missing value of Total deaths

In [17]:
```
1 df["Total deaths"]= df["Total deaths"].fillna(df["Total deaths"].mean())
```

In [18]:
```
df.isna().sum()
```

Out[18]:
```
Year                    0
ISO                     0
 country_name           0
Disaster                0
disaster subgroup       0
occurrence              0
Total deaths            0
Injured              1863
Affected             1269
Homeless             2243
Total affected        809
Total damage            0
dtype: int64
```

4)Now we filling the missing values of Total affected with fillna of nearest preceding non missing values.

In [19]:
```
df["Total affected"]= df["Total affected"].fillna(method='backfill')
```

In [20]:
```
df.isna().sum()
```

Out[20]:
```
Year                    0
ISO                     0
 country_name           0
Disaster                0
disaster subgroup       0
occurrence              0
Total deaths            0
Injured              1863
Affected             1269
Homeless             2243
Total affected          1
Total damage            0
dtype: int64
```

In [21]:
```
df.fillna(0, inplace=True)

```

Note:-

```
Fill the other missing values with 0 ,if we need to replace with some other crteria that may require for
our data analysis we will do  later.
```

Exporting the dataset into new csv file

```
Use: datasetname.to_CVS(‘outputfilename.cvs’)
```

In [22]:
```
df.to_csv("df9.csv")#clean dataset
```

In [23]:
```
df.isna().sum()
```

Out[23]:
```
Year                 0
ISO                  0
 country_name        0
Disaster             0
disaster subgroup    0
occurrence           0
Total deaths         0
Injured              0
Affected             0
Homeless             0
Total affected       0
Total damage         0
dtype: int64
```

CORRELATION BETWEEN COLUMNS:-

Correlation values range between -1 and 1.There are two key components of a correlation value: magnitude – The larger the magnitude (closer to 1 or -1), the stronger the correlation sign – If negative, there is an inverse correlation. If positive, there is a regular correlation.

In [24]:
```
df.corr()
```

Out[24]:

	Year	occurrence	Total deaths	Injured	Affected	Homeless	Total affected	Total damage
Year	1.000000	0.129977	-0.045711	0.013002	0.039463	0.002364	0.039229	0.066678
occurrence	0.129977	1.000000	-0.016084	-0.009905	0.130124	0.050892	0.125103	0.344985
Total deaths	-0.045711	-0.016084	1.000000	0.177602	0.048417	0.064354	0.052293	0.024817
Injured	0.013002	-0.009905	0.177602	1.000000	0.012275	0.027909	0.031369	0.004772
Affected	0.039463	0.130124	0.048417	0.012275	1.000000	0.054482	0.989218	0.123211
Homeless	0.002364	0.050892	0.064354	0.027909	0.054482	1.000000	0.073804	0.109217
Total affected	0.039229	0.125103	0.052293	0.031369	0.989218	0.073804	1.000000	0.123411
Total damage	0.066678	0.344985	0.024817	0.004772	0.123211	0.109217	0.123411	1.000000

HEAT MAP:-

A heat map is a two-dimensional representation of data in which values are represented by colors. A simple heat map provides an immediate visual summary of information. More elaborate heat maps allow the viewer to understand complex data sets

In [25]:

```python

corr=df.corr()

sns.set(font_scale=1.15)
plt.figure(figsize=(14, 10))

sns.heatmap(corr, vmax=.8, linewidths=0.01,
            square=True,annot=True,cmap='YlGnBu',linecolor="black")
plt.title('Correlation between features');
```

PAIRPLOT

In [26]:
```
sns.pairplot(df)
```

Out[26]: <seaborn.axisgrid.PairGrid at 0x1cb66200dd8>

Now different sub dataset will be used based on the location of the countries

North America(USA, Canada,Mexico)

1)North_America (USA, Canada,Mexico)

In [27]:
```
North_America = ['USA','CAN','MEX']
North_America = df[df.ISO.isin(North_America)]


```

a) Total number of north american countries name in dataset

In [28]:
```
USA=['USA']
CAN=['CAN']
MEX=['MEX']
USA=North_America[North_America.ISO.isin(USA)]
CAN=North_America[North_America.ISO.isin(CAN)]
MEX=North_America[North_America.ISO.isin(MEX)]
print(USA.shape);print(CAN.shape);print(MEX.shape)
```

```
(304, 12)
(171, 12)
(176, 12)
```

In [29]:
```
print(North_America.head(10))
```

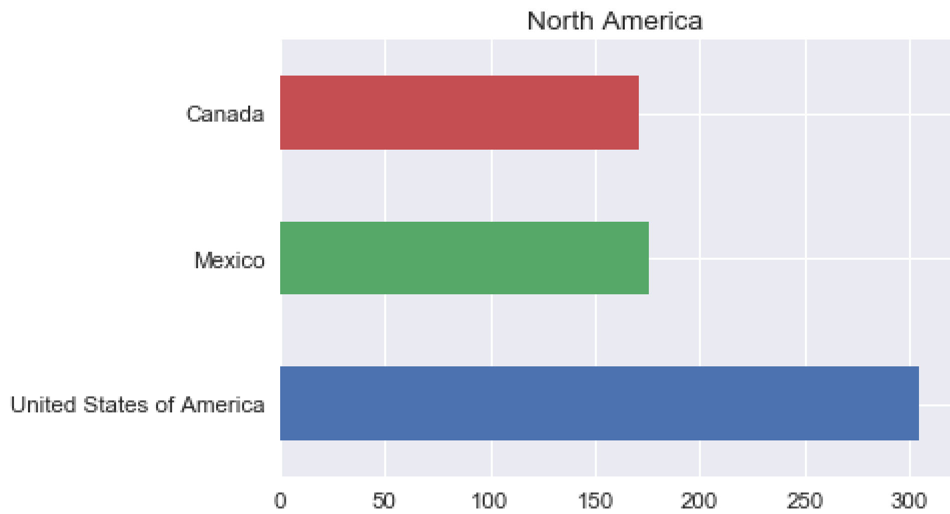
```
    Year  ISO              country_name      Disaster disaster subgroup  \
2   1900  USA  United States of America       Natural    Meteorological
7   1902  USA  United States of America  Technological     Technological
8   1903  CAN                    Canada       Natural       Geophysical
10  1903  USA  United States of America       Natural      Hydrological
11  1903  USA  United States of America       Natural    Meteorological
12  1903  USA  United States of America  Technological     Technological
14  1904  USA  United States of America  Technological     Technological
15  1905  CAN                    Canada       Natural       Geophysical
21  1906  USA  United States of America       Natural       Geophysical
22  1906  USA  United States of America       Natural    Meteorological

    occurrence  Total deaths  Injured  Affected  Homeless  Total affected  \
2            1        6000.0      0.0       0.0       0.0            23.0
7            1         115.0      0.0       0.0       0.0            23.0
8            1          76.0     23.0       0.0       0.0            23.0
10           2         250.0      0.0       0.0       0.0            18.0
11           1          98.0      0.0       0.0       0.0            18.0
12           1         602.0      0.0       0.0       0.0            18.0
14           2        1096.0      0.0       0.0       0.0            18.0
15           1          18.0     18.0       0.0       0.0            18.0
21           1        2000.0      0.0       0.0       0.0         90000.0
22           2         298.0      0.0       0.0       0.0         90000.0

    Total damage
2   3.000000e+04
7   1.035150e+06
8   1.035150e+06
10  4.800000e+05
11  1.035150e+06
12  1.035150e+06
14  1.035150e+06
15  1.035150e+06
21  5.240000e+05
22  1.035150e+06
```

In [30]:
```
North_America.shape
```

Out[30]: (651, 12)

In [31]:
```
(North_America [' country_name']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('North America')
)
```

Out[31]: Text(0.5,1,'North America')



In [32]:
```
df_NA=North_America
```

In [33]:
```
df_NA.to_csv("df_NA.csv")
```

2) South_America

In [34]:
```
South_America = ['BLZ','CRI','SLV','GTM','HND','NIC','PAN']
South_America = df[df.ISO.isin(South_America)]
```

In [35]:
```
South_America.shape
```

Out[35]: (471, 12)

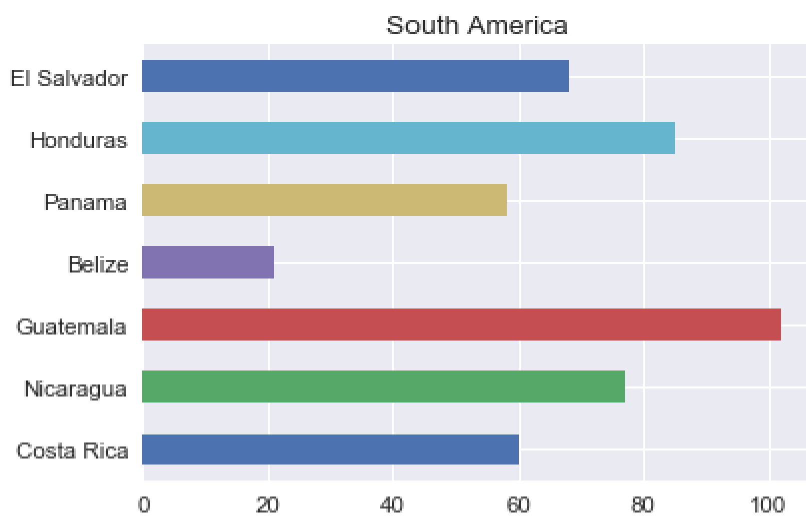
In [36]:
```
print(South_America.head(5))
```

```
    Year  ISO  country_name Disaster disaster subgroup  occurrence  \
3   1902  SLV   El Salvador  Natural       Geophysical           1
4   1902  GTM     Guatemala  Natural       Geophysical           3
19  1906  SLV   El Salvador  Natural       Geophysical           1
20  1906  NIC     Nicaragua  Natural       Geophysical           1
36  1910  CRI    Costa Rica  Natural       Geophysical           1

    Total deaths  Injured  Affected  Homeless  Total affected  Total damage
3     185.000000      0.0       0.0       0.0            23.0  1.035150e+06
4    9000.000000      0.0       0.0       0.0            23.0  1.035150e+06
19    436.892805      0.0       0.0       0.0         90000.0  1.035150e+06
20   1000.000000      0.0       0.0       0.0         90000.0  1.035150e+06
36   1750.000000      0.0       0.0       0.0           200.0  1.035150e+06
```

In [ ]:
```

```

In [37]:
```
(South_America [' country_name']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('South America')
)
```

Out[37]: Text(0.5,1,'South America')



In [38]:
```
df_SA=South_America
df_SA.to_csv("df_SA.csv")
```

3) Central_America

In [39]:
```
Central_America = ['ARG','BOL','BRA','CHL','COL','ECU','GUF','GUY','PRY','PER','SUR','URY','VEN']
Central_America = df[df.ISO.isin(Central_America)]
```

In [40]:
```
print(Central_America.head(5))
```

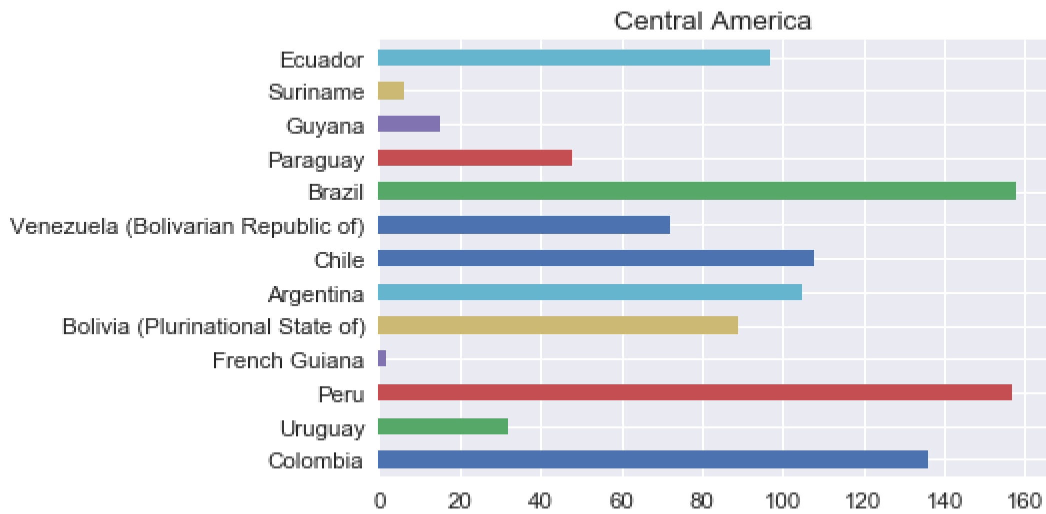
```
    Year  ISO  country_name Disaster disaster subgroup  occurrence  \
13  1904  ECU       Ecuador  Natural       Geophysical           1
16  1906  CHL         Chile  Natural       Geophysical           1
17  1906  COL      Colombia  Natural       Geophysical           1
18  1906  ECU       Ecuador  Natural       Geophysical           1
43  1913  PER          Peru  Natural       Geophysical           1

    Total deaths  Injured  Affected  Homeless  Total affected  Total damage
13    436.892805      0.0       0.0       0.0            18.0  1.035150e+06
16  20000.000000      0.0       0.0       0.0         90000.0  1.000000e+05
17    400.000000      0.0       0.0       0.0         90000.0  1.035150e+06
18    436.892805      0.0       0.0       0.0         90000.0  1.035150e+06
43    150.000000      0.0       0.0       0.0            22.0  1.035150e+06
```

In [41]:
```
Central_America.shape
```

Out[41]: (1025, 12)

In [42]:
```
(Central_America [' country_name']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('Central America')
)
```

Out[42]: Text(0.5,1,'Central America')



In [43]:
```
df_CA=Central_America
```

In [44]:
```
df_CA.to_csv("df_CA.csv")
```

In [45]:
```
N_C_S_America = ['ARG','BOL','BRA','CHL','COL','ECU','GUF','GUY','PRY','PER','SUR','URY','VEN',
                'BLZ','CRI','SLV','GTM','HND','NIC','PAN',
                'USA','CAN', 'MEX']
# the (~) symbole is to select anything except this list
Caribbean = df[~df.ISO.isin(N_C_S_America)]
```

In [46]:
```
df_Caribbean=Caribbean
df_Caribbean.to_csv("df_Caribbean.csv")

```

In [47]:
```
print(Caribbean.head(10))
```

```
    Year  ISO                      country_name Disaster disaster subgroup  \
0   1900  JAM                           Jamaica  Natural        Biological
1   1900  JAM                           Jamaica  Natural      Hydrological
5   1902  MTQ                        Martinique  Natural       Geophysical
6   1902  VCT  Saint Vincent and the Grenadines  Natural       Geophysical
9   1903  JAM                           Jamaica  Natural     Meteorological
25  1907  JAM                           Jamaica  Natural       Geophysical
29  1909  HTI                             Haiti  Natural     Meteorological
30  1909  JAM                           Jamaica  Natural      Hydrological
41  1912  JAM                           Jamaica  Natural     Meteorological
47  1915  HTI                             Haiti  Natural     Meteorological

    occurrence  Total deaths  Injured  Affected  Homeless  Total affected  \
0            1          30.0      0.0       0.0       0.0            23.0
1            1         300.0      0.0       0.0       0.0            23.0
5            1       30000.0      0.0       0.0       0.0            23.0
6            1        1565.0      0.0       0.0       0.0            23.0
9            1          65.0      0.0       0.0       0.0            18.0
25           1        1200.0      0.0   90000.0       0.0         90000.0
```

In [48]:
```
Caribbean.shape
```

Out[48]: (587, 12)

Types of Disaster in Different region of North America,South America,Central Region & Carribean Countries

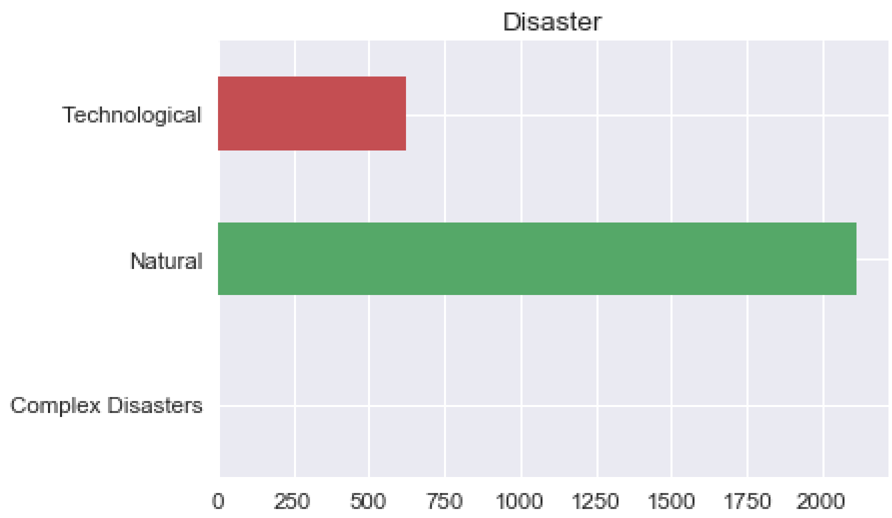
1)Overall Disasters(Full Datasets)

In [49]:
```
print(df['Disaster'].value_counts())
```

```
Natural              2111
Technological         621
Complex Disasters       2
Name: Disaster, dtype: int64
```

In [50]:
```
(df['Disaster']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('Disaster')
)
```

Out[50]: Text(0.5,1,'Disaster')



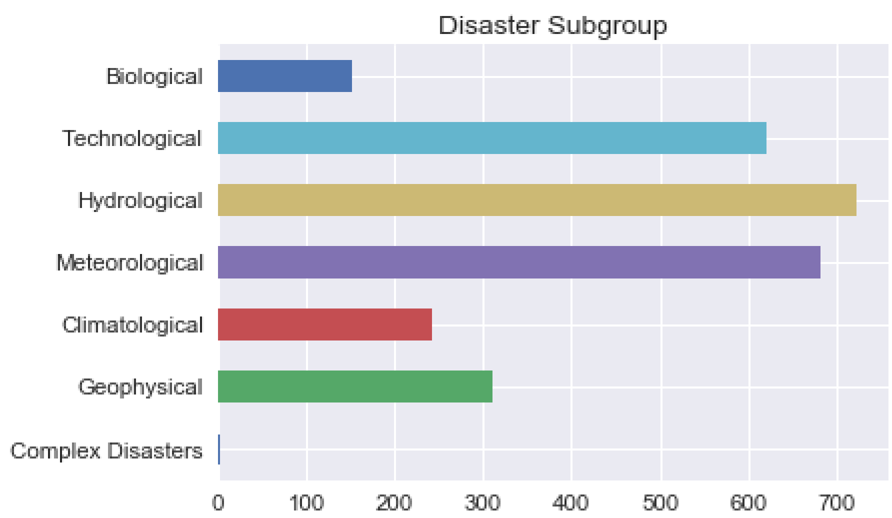
1i)Disasters subgroup(Full Dataset)

In [51]:
```
print(df['disaster subgroup'].value_counts())
```

```
Hydrological        722
Meteorological      683
Technological       621
Geophysical         310
Climatological      243
Biological          153
Complex Disasters     2
Name: disaster subgroup, dtype: int64
```

In [52]:
```
(df['disaster subgroup']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('Disaster Subgroup')
)
```

Out[52]:
```
Text(0.5,1,'Disaster Subgroup')
```



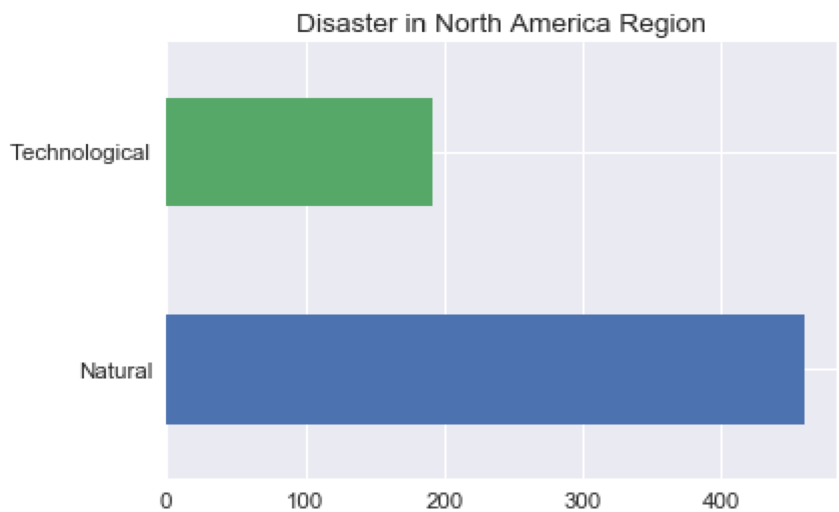
2)Disasters in North America

In [53]:
```
print (North_America['Disaster'].value_counts())
```

```
Natural          460
Technological    191
Name: Disaster, dtype: int64
```

In [54]:
```
(North_America['Disaster']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('Disaster in North America Region')
)
```

Out[54]: Text(0.5,1,'Disaster in North America Region')



2i)Sub-Disaster in North Ameirca

In [55]:
```
1 print(North_America['disaster subgroup'].value_counts())
```

```
Technological     191
Meteorological    179
Hydrological      127
Climatological     70
Geophysical        70
Biological         14
Name: disaster subgroup, dtype: int64
```

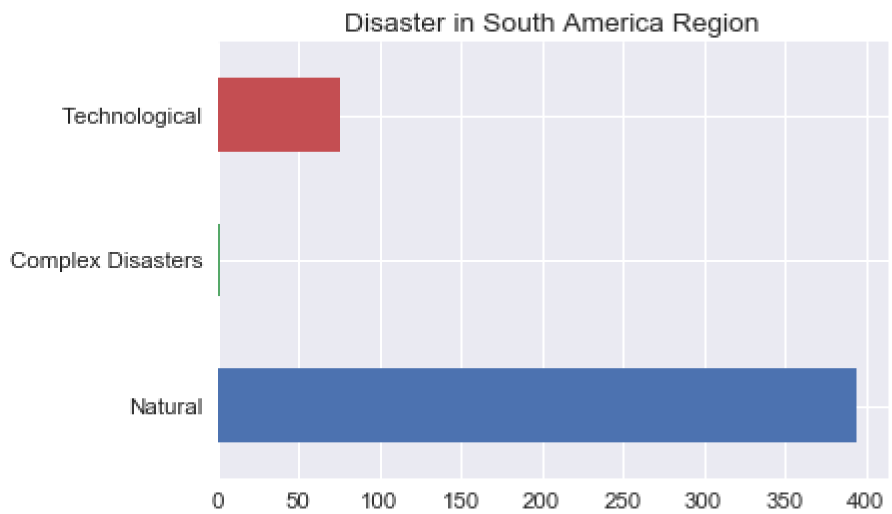
3)Disaster in South America

In [56]:
```
1 print (South_America['Disaster'].value_counts())
```

```
Natural              394
Technological         75
Complex Disasters      2
Name: Disaster, dtype: int64
```

In [57]:
```
(South_America['Disaster']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('Disaster in South America Region')
)
```

Out[57]: Text(0.5,1,'Disaster in South America Region')



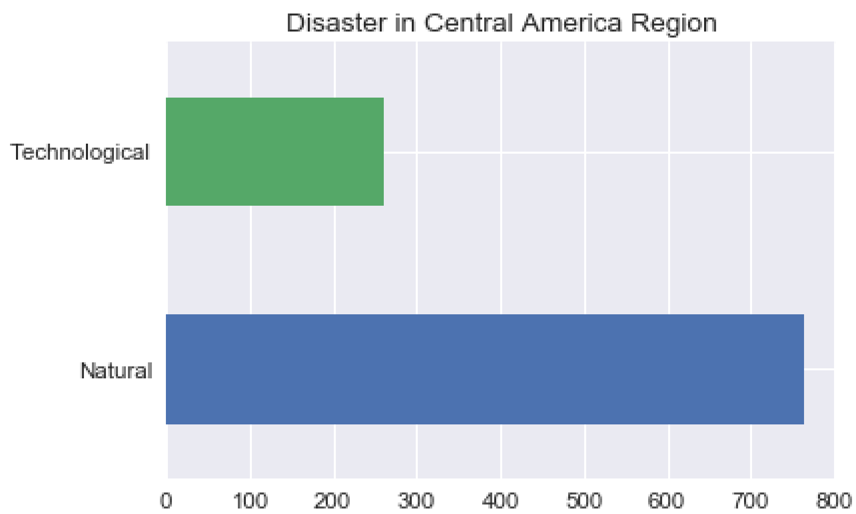
4)Disaster in Central America

In [58]:
```
print (Central_America['Disaster'].value_counts())
```

```
Natural          764
Technological    261
Name: Disaster, dtype: int64
```

In [59]:
```
(Central_America['Disaster']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('Disaster in Central America Region')
)
```

Out[59]: Text(0.5,1,'Disaster in Central America Region')



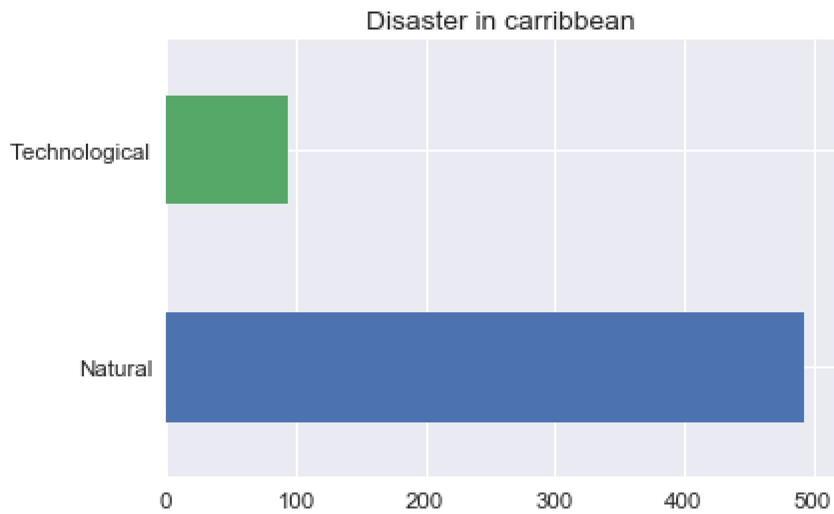
5)Disaster in Carribbean

In [60]:
```
print (Caribbean['Disaster'].value_counts())
```

```
Natural          493
Technological     94
Name: Disaster, dtype: int64
```

In [61]:
```
(Caribbean['Disaster']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('Disaster in carribbean')
)
```

Out[61]: Text(0.5,1,'Disaster in carribbean')

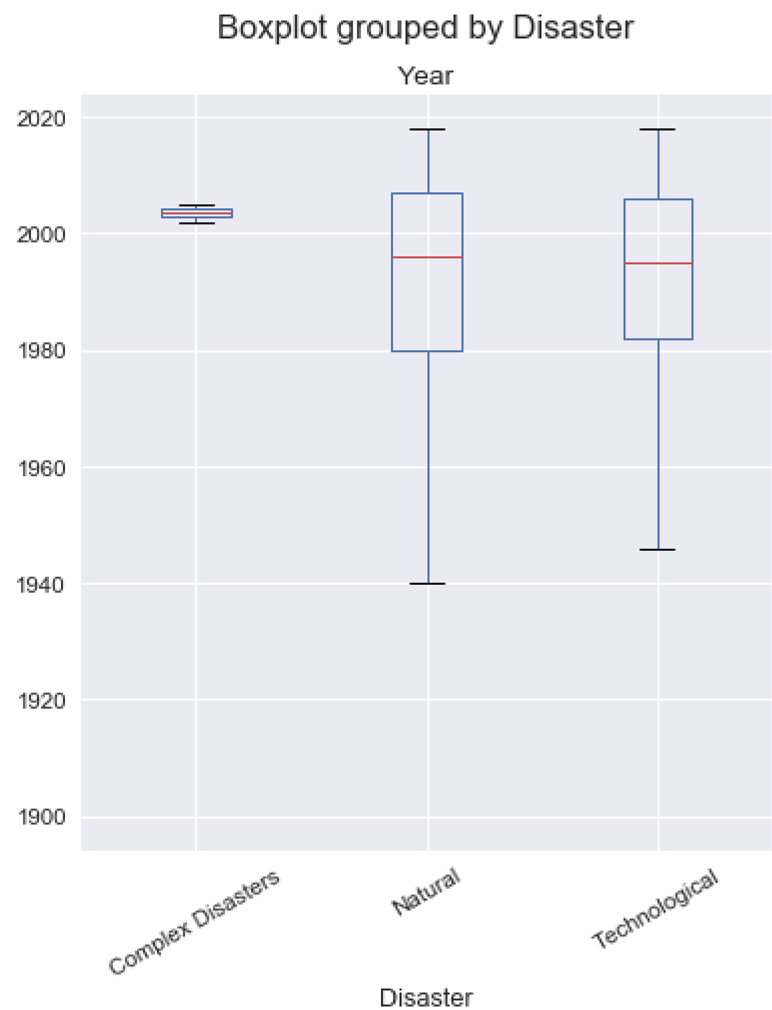


Visualization(Box plot):-

Box plot for each numeric variable will give us a clearer idea of the distribution of the input variables:

In [62]:
```
df.boxplot('Year','Disaster',rot = 30,figsize=(6,7))
```

Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb69d52898>



North America dataset Analysis

Model, predict and solve:-

Now we are ready to train a model and predict the required solution. There are 60+ predictive modelling algorithms to choose from. We must understand the type of problem and solution requirement to narrow down to a select few models which we can evaluate. Our problem is a classification and regression problem.

In [63]:
```
North_America.shape
```
Out[63]:
```
(651, 12)
```

In [64]:
```
df3=North_America
```

In [65]:
```
print(df3.head(10))
```

```
    Year  ISO              country_name       Disaster disaster subgroup  \
2   1900  USA  United States of America        Natural    Meteorological   
7   1902  USA  United States of America  Technological     Technological   
8   1903  CAN                    Canada        Natural       Geophysical   
10  1903  USA  United States of America        Natural      Hydrological   
11  1903  USA  United States of America        Natural    Meteorological   
12  1903  USA  United States of America  Technological     Technological   
14  1904  USA  United States of America  Technological     Technological   
15  1905  CAN                    Canada        Natural       Geophysical   
21  1906  USA  United States of America        Natural       Geophysical   
22  1906  USA  United States of America        Natural    Meteorological   

    occurrence  Total deaths  Injured  Affected  Homeless  Total affected  \
2            1        6000.0      0.0       0.0       0.0            23.0   
7            1         115.0      0.0       0.0       0.0            23.0   
8            1          76.0     23.0       0.0       0.0            23.0   
10           2         250.0      0.0       0.0       0.0            18.0   
11           1          98.0      0.0       0.0       0.0            18.0   
12           1         602.0      0.0       0.0       0.0            18.0   
14           2        1096.0      0.0       0.0       0.0            18.0   
15           1          18.0     18.0       0.0       0.0            18.0   
21           1        2000.0      0.0       0.0       0.0         90000.0   
22           2         298.0      0.0       0.0       0.0         90000.0   

    Total damage  
2   3.000000e+04  
7   1.035150e+06  
8   1.035150e+06  
10  4.800000e+05  
11  1.035150e+06  
12  1.035150e+06  
14  1.035150e+06  
15  1.035150e+06  
21  5.240000e+05  
22  1.035150e+06  
```

In [66]:

```
1 df3.corr()
```

Out[66]:

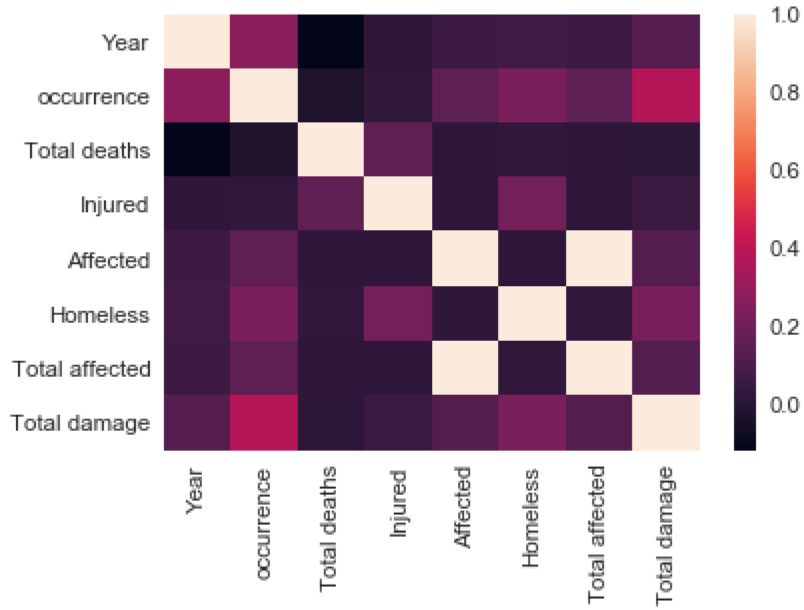
	Year	occurrence	Total deaths	Injured	Affected	Homeless	Total affected	Total damage
Year	1.000000	0.276257	-0.114404	0.027261	0.062955	0.077900	0.063509	0.137691
occurrence	0.276257	1.000000	-0.024107	0.032253	0.158864	0.233778	0.157217	0.384232
Total deaths	-0.114404	-0.024107	1.000000	0.163979	0.024091	0.033167	0.024355	0.017239
Injured	0.027261	0.032253	0.163979	1.000000	0.021861	0.212280	0.022494	0.059566
Affected	0.062955	0.158864	0.024091	0.021861	1.000000	0.026002	0.998895	0.126929
Homeless	0.077900	0.233778	0.033167	0.212280	0.026002	1.000000	0.030404	0.225587
Total affected	0.063509	0.157217	0.024355	0.022494	0.998895	0.030404	1.000000	0.127200
Total damage	0.137691	0.384232	0.017239	0.059566	0.126929	0.225587	0.127200	1.000000

Heat map

A heat map is a two-dimensional representation of data in which values are represented by colors. A simple heat map provides an immediate visual summary of information. More elaborate heat maps allow the viewer to understand complex data sets.Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate together. A positive correlation indicates the extent to which those variables increase or decrease in parallel; a negative correlation indicates the extent to which one variable increases as the other decreases.When the r value is closer to +1 or -1, it indicates that there is a stronger linear relationship between the two variables. A correlation of -0.97 is a strong negative correlation while a correlation of 0.10 would be a weak positive correlation.

In [67]:
```
corr = df3.corr()
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)
```

Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb69ea7550>



In [68]:
```
df3.head()
```

Out[68]:

	Year	ISO	country_name	Disaster	disaster subgroup	occurrence	Total deaths	Injured	Affected	Homeless	Total affected	Total damage
2	1900	USA	United States of America	Natural	Meteorological	1	6000.0	0.0	0.0	0.0	23.0	3.000000e+04
7	1902	USA	United States of America	Technological	Technological	1	115.0	0.0	0.0	0.0	23.0	1.035150e+06
8	1903	CAN	Canada	Natural	Geophysical	1	76.0	23.0	0.0	0.0	23.0	1.035150e+06
10	1903	USA	United States of America	Natural	Hydrological	2	250.0	0.0	0.0	0.0	18.0	4.800000e+05
11	1903	USA	United States of America	Natural	Meteorological	1	98.0	0.0	0.0	0.0	18.0	1.035150e+06

Dealing with Categorical variables

```
As you will only be dealing with categorical features in this tutorial, it's better to filter them out.
You can create a separate DataFrame consisting of only these features by running the following command.
The method .copy() is used here so that any changes made in new DataFrame don't get reflected in the
original one.
```

In [69]:
```
cat_df3 = df3.select_dtypes(include=['object']).copy()
cat_df3.head(5)


```

Out[69]:

	ISO	country_name	Disaster	disaster subgroup
2	USA	United States of America	Natural	Meteorological
7	USA	United States of America	Technological	Technological
8	CAN	Canada	Natural	Geophysical
10	USA	United States of America	Natural	Hydrological
11	USA	United States of America	Natural	Meteorological

In [70]:
```
print(cat_df3.isnull().values.sum())
```

```
0
```

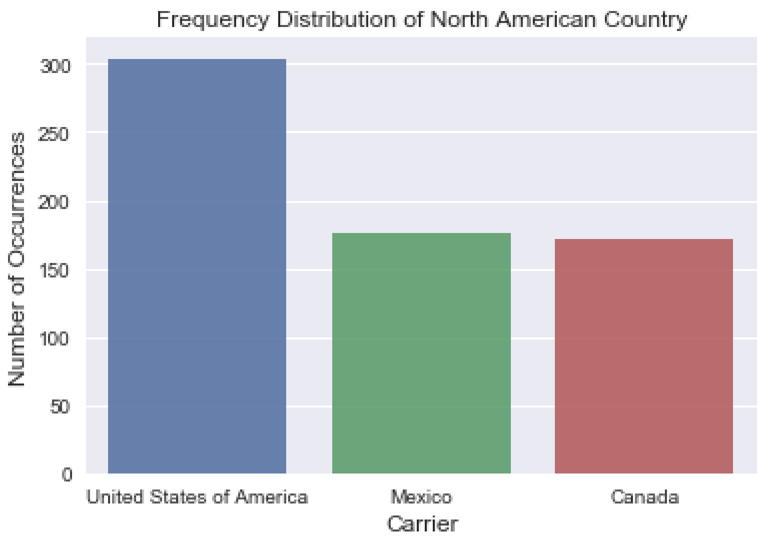
In [71]:
```
print(cat_df3[' country_name'].value_counts())
```

```
United States of America    304
Mexico                      176
Canada                      171
Name:  country_name, dtype: int64
```

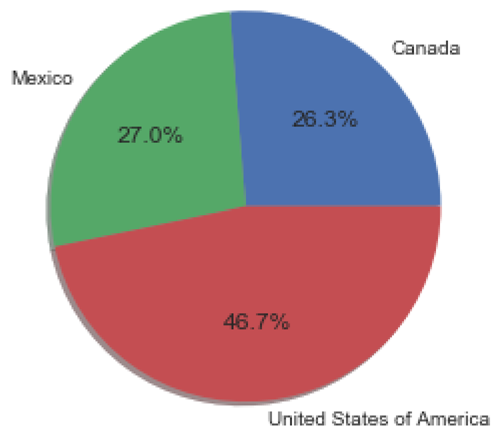
In [72]:
```
print(cat_df3[' country_name'].value_counts().count())
```

```
3
```

In [73]:
```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
country_name_count = cat_df3[' country_name'].value_counts()
sns.set(style="darkgrid")
sns.barplot(country_name_count.index, country_name_count.values, alpha=0.9)
plt.title('Frequency Distribution of North American Country')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Carrier', fontsize=12)
plt.show()
```



In [74]:
```
labels = cat_df3[' country_name'].astype('category').cat.categories.tolist()
counts = cat_df3[' country_name'].value_counts()
sizes = [counts[var_cat] for var_cat in labels]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True) #autopct is show the % on plot
ax1.axis('equal')
plt.show()
```



Binary Encoding

```
In this technique, first the categories are encoded as ordinal, then those integers are converted into binary code, then the digits from that binary string are split into separate columns. This encodes the data in fewer dimensions than one-hot.You can do binary encoding via a number of ways but the simplest one is using the category_encoders library. You can install category_encoders via pip install category_encoders on cmd or just download and extract the .tar.gz file from the site.
You have to first import the category_encoders library after installing it. Invoke the BinaryEncoder function by specifying the columns you want to encode and then call the .fit_transform() method on it with the DataFrame as the argument.
```

In [75]:

```
cat_df3_country_ce = cat_df3.copy()

!pip install category_encoders
import category_encoders as ce

encoder = ce.BinaryEncoder(cols=[' country_name'])
df3_binary = encoder.fit_transform(cat_df3_country_ce)

df3_binary.head()
```

```
Requirement already satisfied: category_encoders in c:\users\mahwish\anaconda3\lib\site-packages (1.3.0)
Requirement already satisfied: patsy>=0.4.1 in c:\users\mahwish\anaconda3\lib\site-packages (from category_enco
ders) (0.5.0)
Requirement already satisfied: statsmodels>=0.6.1 in c:\users\mahwish\anaconda3\lib\site-packages (from categor
y_encoders) (0.8.0)
Requirement already satisfied: scikit-learn>=0.17.1 in c:\users\mahwish\anaconda3\lib\site-packages (from categ
ory_encoders) (0.19.1)
Requirement already satisfied: scipy>=0.17.0 in c:\users\mahwish\anaconda3\lib\site-packages (from category_enc
oders) (1.0.0)
Requirement already satisfied: numpy>=1.11.1 in c:\users\mahwish\anaconda3\lib\site-packages (from category_enc
oders) (1.14.2)
Requirement already satisfied: pandas>=0.20.1 in c:\users\mahwish\anaconda3\lib\site-packages (from category_en
coders) (0.22.0)
Requirement already satisfied: six in c:\users\mahwish\anaconda3\lib\site-packages (from patsy>=0.4.1->category
_encoders) (1.11.0)
Requirement already satisfied: python-dateutil>=2 in c:\users\mahwish\anaconda3\lib\site-packages (from pandas>
=0.20.1->category_encoders) (2.6.1)
Requirement already satisfied: pytz>=2011k in c:\users\mahwish\anaconda3\lib\site-packages (from pandas>=0.20.1
->category_encoders) (2017.3)

You are using pip version 19.0.2, however version 19.0.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

Out[75]:

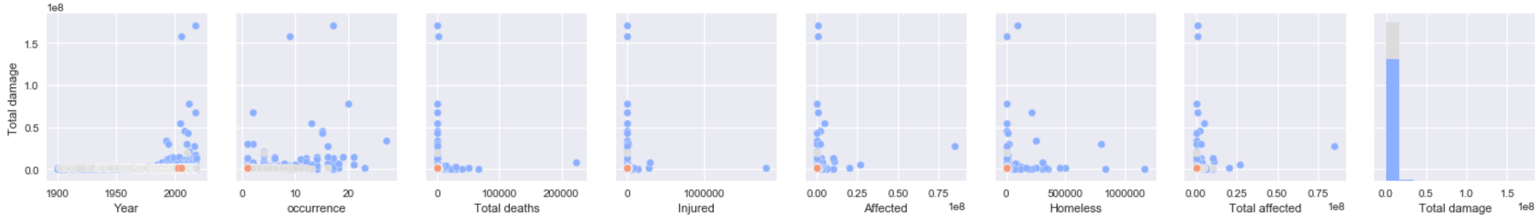
	country_name_0	country_name_1	country_name_2	ISO	Disaster	disaster subgroup
2	0	0	1	USA	Natural	Meteorological
7	0	0	1	USA	Technological	Technological
8	0	1	0	CAN	Natural	Geophysical
10	0	0	1	USA	Natural	Hydrological
11	0	0	1	USA	Natural	Meteorological

```
 
```

PairPlot of Disaster

In [76]:
```
sns.pairplot(data=df,hue='Disaster',palette = 'coolwarm')
```

Out[76]: <seaborn.axisgrid.PairGrid at 0x1cb69dc3f98>



Correlation Matrix Visualization between features

In [77]:
```
corr=df3.corr()

sns.set(font_scale=1.15)
plt.figure(figsize=(14, 10))

sns.heatmap(corr, vmax=.8, linewidths=0.01,
            square=True,annot=True,cmap='YlGnBu',linecolor="black")
plt.title('Correlation between features');
```


Correlation between features
Year
occurrence
Total deaths
Injured
Affected
Homeless
Total affected
Total damage
1
0.28
-0.11
0.027
0.063
0.078
0.064
0.14
0.38
-0.024
0.032
0.16
0.23
0.024
0.033
0.017
0.022
0.21
0.06
0.026
0.13
0.03
0.8
0.6
0.4
0.2
0.0

```
In [ ]: 1
```

Note:-

Three new columns are created in place of the "Country name" column with binary encoding for each category in the feature.

```
Now first we remove the column names "ISO,Disaster and disaster subgroup" from df3 binary dataset
```

In [78]:
```
df3_binary_new=df3_binary.drop(["ISO","Disaster","disaster subgroup"],axis=1)
```

In [79]:
```
df3_binary_new.head()
```

Out[79]:

	country_name_0	country_name_1	country_name_2
2	0	0	1
7	0	0	1
8	0	1	0
10	0	0	1
11	0	0	1

Note:-

Now join df3_binary_new with df3 but before that we need to remove the country_name from df3 dataset as we already make country name as binary encoding

In [80]:
```
print(df3.head(2))
```

```
   Year  ISO             country_name      Disaster disaster subgroup  \
2  1900  USA  United States of America       Natural    Meteorological
7  1902  USA  United States of America  Technological     Technological

   occurrence  Total deaths  Injured  Affected  Homeless  Total affected  \
2           1        6000.0      0.0       0.0       0.0            23.0
7           1         115.0      0.0       0.0       0.0            23.0

   Total damage
2  3.000000e+04
7  1.035150e+06
```

In [81]:
```
df3=df3.drop([" country_name"],axis =1)


```

In [82]:
```
df3=df3.drop(["ISO"],axis=1)
df3=df3.drop(["disaster subgroup"],axis=1)
```

In [83]:
```
print(df3.head())
```

```
    Year       Disaster  occurrence  Total deaths  Injured  Affected  \
2   1900        Natural           1        6000.0      0.0       0.0
7   1902  Technological           1         115.0      0.0       0.0
8   1903        Natural           1          76.0     23.0       0.0
10  1903        Natural           2         250.0      0.0       0.0
11  1903        Natural           1          98.0      0.0       0.0

    Homeless  Total affected  Total damage
2        0.0            23.0  3.000000e+04
7        0.0            23.0  1.035150e+06
8        0.0            23.0  1.035150e+06
10       0.0            18.0  4.800000e+05
11       0.0            18.0  1.035150e+06
```

Note:-

we have dropped country name from df3 country no we will join df3 with df3_binary_new

In [84]:
```

df3=df3.join(df3_binary_new)
```

In [85]:
```
df3.shape
```

Out[85]:
```
(651, 12)
```

In [86]:
```
df3.columns
```

Out[86]:
```
Index(['Year', 'Disaster', 'occurrence', 'Total deaths', 'Injured', 'Affected',
       'Homeless', 'Total affected', 'Total damage', ' country_name_0',
       ' country_name_1', ' country_name_2'],
      dtype='object')
```

In [87]:
```
df3.head()
```

Out[87]:

	Year	Disaster	occurrence	Total deaths	Injured	Affected	Homeless	Total affected	Total damage	country_name_0	country_name_1	coun
2	1900	Natural	1	6000.0	0.0	0.0	0.0	23.0	3.000000e+04	0	0	
7	1902	Technological	1	115.0	0.0	0.0	0.0	23.0	1.035150e+06	0	0	
8	1903	Natural	1	76.0	23.0	0.0	0.0	23.0	1.035150e+06	0	1	
10	1903	Natural	2	250.0	0.0	0.0	0.0	18.0	4.800000e+05	0	0	
11	1903	Natural	1	98.0	0.0	0.0	0.0	18.0	1.035150e+06	0	0	

Applying several different supervised machine learning techniques to this data set, and see which one yields the highest accuracy as measured with K-Fold cross validation (K=10).

In [88]:
```
from sklearn.model_selection import train_test_split

```

In [89]:
```
X = df3.drop("Disaster",axis = 1)
y = df3["Disaster"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,random_state=1435)

```

In [90]:
```
X.head()
```

Out[90]:

	Year	occurrence	Total deaths	Injured	Affected	Homeless	Total affected	Total damage	country_name_0	country_name_1	country_name_2
2	1900	1	6000.0	0.0	0.0	0.0	23.0	3.000000e+04	0	0	1
7	1902	1	115.0	0.0	0.0	0.0	23.0	1.035150e+06	0	0	1
8	1903	1	76.0	23.0	0.0	0.0	23.0	1.035150e+06	0	1	0
10	1903	2	250.0	0.0	0.0	0.0	18.0	4.800000e+05	0	0	1
11	1903	1	98.0	0.0	0.0	0.0	18.0	1.035150e+06	0	0	1

In [91]:
```
features = ["Year","occurrence","Total deaths","Injured","Affected","Homeless","Total affected","Total damag
features
```

Out[91]:
```
['Year',
 'occurrence',
 'Total deaths',
 'Injured',
 'Affected',
 'Homeless',
 'Total affected',
 'Total damage',
 'country_name_0',
 'country_name_1',
 'country_name_2']
```

Decision Trees

In [92]:

```
from sklearn.tree import DecisionTreeClassifier

clf= DecisionTreeClassifier(random_state=0)

# Train the classifier on the training set
clf.fit(X_train, y_train)
```

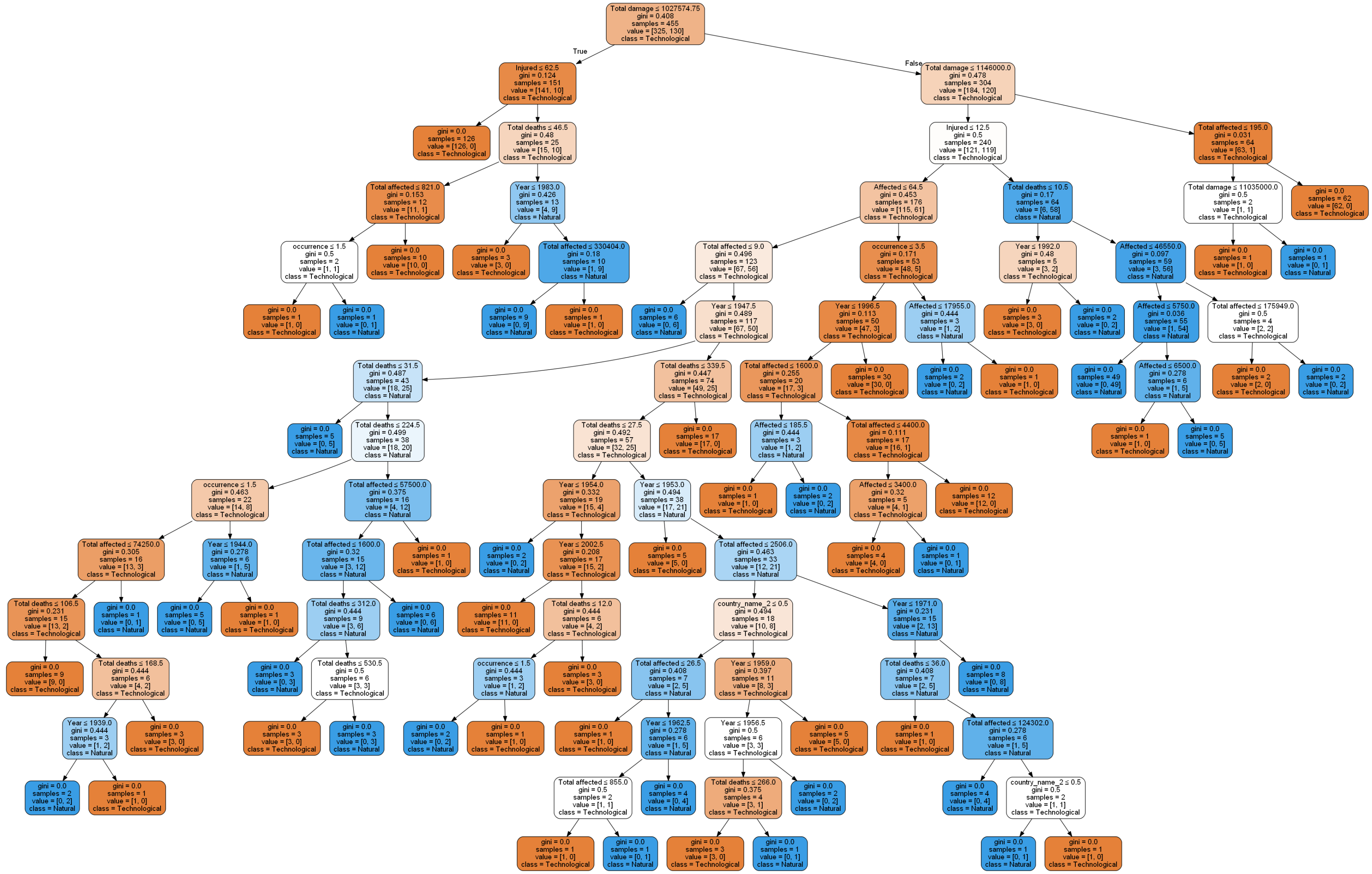
Out[92]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

In [93]:

```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn import tree
from pydotplus import graph_from_dot_data

dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data,
                         feature_names=features,class_names=["Technological","Natural","Complex Disasters"],
graph = graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

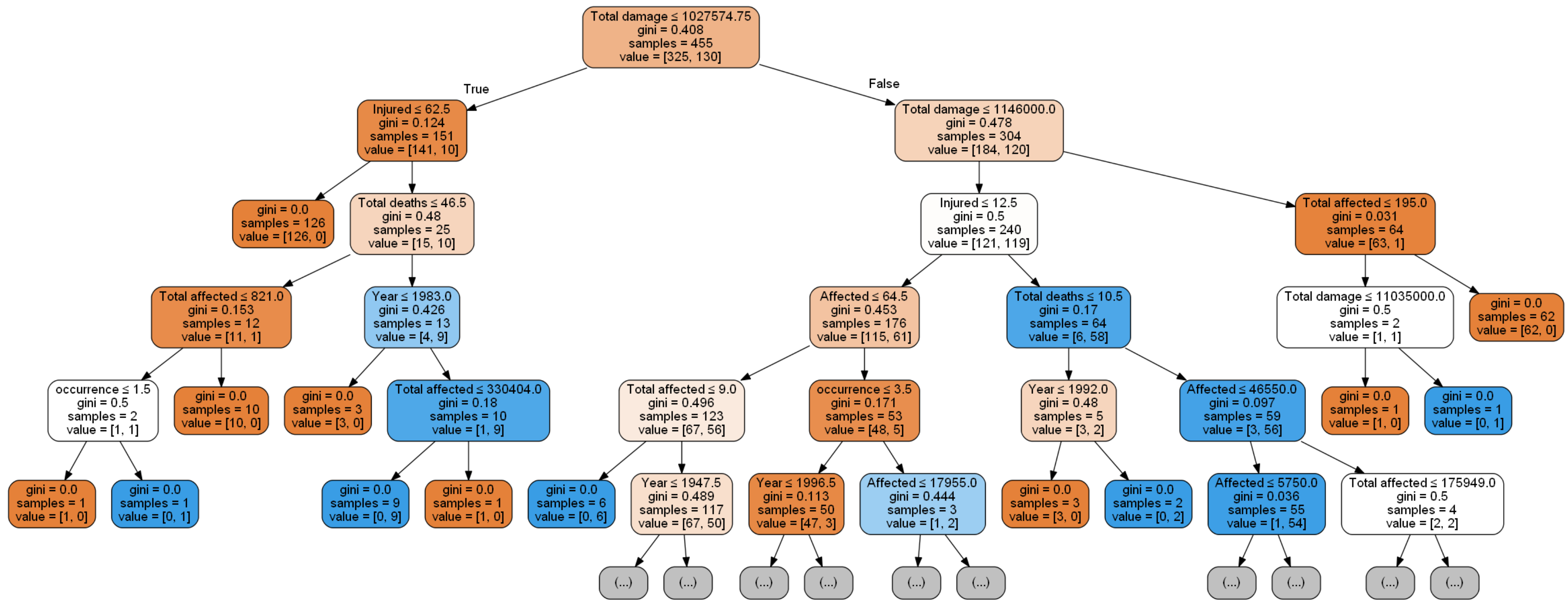
Out[93]:



using max depth =5 for shrink tree

In [94]:
```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn import tree
from pydotplus import graph_from_dot_data

dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data,
                        feature_names=features,filled = True,rounded = True,special_characters = True,max_d
graph = graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[94]:



Measuring the accuracy of the resulting decision tree model using your test data

In [95]:
```

clf.score(X_test, y_test)
```

Out[95]: 0.8061224489795918

Now instead of a single train/test split, let us use K-Fold cross validation to get a better measure of the model's accuracy (K=10).

In [96]:
```
from sklearn.model_selection import cross_val_score

clf = DecisionTreeClassifier(random_state=0)

cv_scores = cross_val_score(clf, X_test, y_test, cv=10)

cv_scores.mean()
```

Out[96]: 0.7956641604010024

HYPER-PARAMETER TUNING USING GRIDSEARCHCV¶

In [97]:
```
from sklearn.model_selection import GridSearchCV
```

In [98]:
```
param_grid = {'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1,5,8]
             }
```

In [99]:
```
grid = GridSearchCV(clf, param_grid,cv = 5)
```

In [100]:
```
grid.fit(X_train,y_train)
```

Out[100]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_features': ['log2', 'sqrt', 'auto'], 'criterion': ['entropy', 'gini'], 'max_depth': [2,
3, 5, 10], 'min_samples_split': [2, 3, 5], 'min_samples_leaf': [1, 5, 8]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [101]:
```
grid.best_params_
```

Out[101]:
```
{'criterion': 'entropy',
 'max_depth': 10,
 'max_features': 'log2',
 'min_samples_leaf': 5,
 'min_samples_split': 2}
```

In [102]:
```
grid.best_score_
```

Out[102]:
```
0.8021978021978022
```

In [103]:
```
clf_1 = grid.best_estimator_
```

lets now check the accuracy of the model by the new hyper-parameters

In [104]:
```
clf_1.fit(X_train,y_train)
```

Out[104]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=10,
            max_features='log2', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

In [105]:
```
clf_1.score(X_test,y_test)
```

Out[105]:
```
0.7653061224489796
```

In [106]:
```
predict = clf_1.predict(X_test)
```

CLASSIFICATION REPORT

In [107]:
```
from sklearn.metrics import classification_report
```

In [108]:
```
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

      Natural       0.80      0.87      0.84       135
Technological       0.65      0.52      0.58        61

  avg / total       0.76      0.77      0.76       196
```

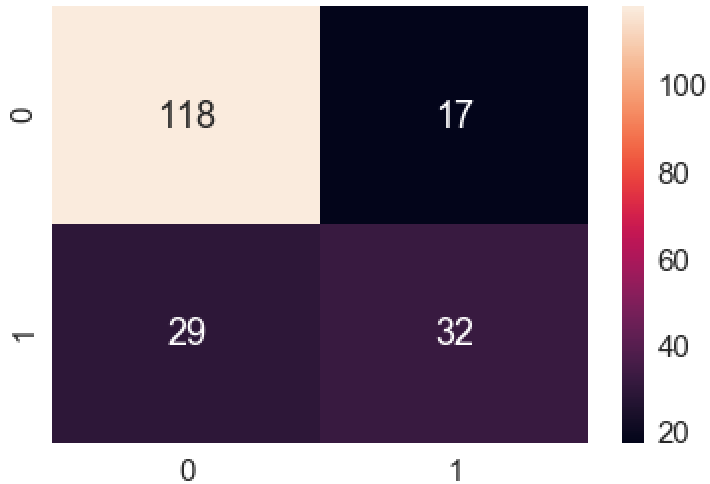
CONFUSION MATRIX

In [109]:
```
from sklearn.metrics import confusion_matrix
```

In [110]:
```
confusion_matrix(predict,y_test)
```

Out[110]:
```
array([[118,  29],
       [ 17,  32]], dtype=int64)
```

In [111]:
```
sns.set(font_scale=1.5)
cm = confusion_matrix(y_test,predict)
sns.heatmap(cm, annot=True, fmt='g')
plt.show()
```



ACCURACY SCORE

In [112]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,predict)
```

Out[112]: 0.7653061224489796

In [113]:
```
TP=118;FP=17;FN=29;TN=32
```

Now TP=118,FP=17,FN=29,TN=32

ACCURACY

In [114]:
```
(TP+TN)/(TP+FP+TN+FN)
```

Out[114]: 0.7653061224489796

PRECISION

In [115]:
```
TP/(TP+FP)
```

Out[115]: 0.8740740740740741

RECALL OR SENSITIVITY

In [116]:
```
TP/(TP+FN)
```

Out[116]: 0.8027210884353742

SPECIFICITY

In [117]:
```
TN/(TN+FP)
```

Out[117]: 0.6530612244897959

Now lets try a RandomForestClassifier instead and see if it performs better? ¶

In [118]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=10, random_state=0)
cv_scores = cross_val_score(clf, X_test, y_test, cv=10)

cv_scores.mean()
```

Out[118]:
```
0.7746365914786967
```

SVM

Next let us try using svm.SVC with a linear kernel and compare to the decision tree SVM also require the input data to be normalized first¶

In [119]:
```
df3.columns
```

Out[119]:
```
Index(['Year', 'Disaster', 'occurrence', 'Total deaths', 'Injured', 'Affected',
       'Homeless', 'Total affected', 'Total damage', ' country_name_0',
       ' country_name_1', ' country_name_2'],
      dtype='object')
```

In [120]:
```



independent_features = df3[['Year','occurrence','Total deaths','Injured','Affected',
       'Homeless','Total affected', 'Total damage',' country_name_0',
       ' country_name_1',' country_name_2']].values


target_feature = df3['Disaster'].values
```

In [121]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
all_features_scaled = scaler.fit_transform(independent_features)
all_features_scaled
```

Out[121]:
```
array([[-2.70520177, -0.56764218,  2.83418474, ...,  0.        ,
        -1.0683854 ,  0.59686682],
       [-2.63753535, -0.56764218, -0.0888751 , ...,  0.        ,
        -1.0683854 ,  0.59686682],
       [-2.60370214, -0.56764218, -0.10824627, ...,  0.        ,
         0.93599182, -1.67541563],
       ...,
       [ 1.28711725, -0.56764218, -0.13556459, ...,  0.        ,
        -1.0683854 ,  0.59686682],
       [ 1.28711725,  0.80711622, -0.07844447, ...,  0.        ,
        -1.0683854 ,  0.59686682],
       [ 1.28711725, -0.2926905 , -0.12761744, ...,  0.        ,
        -1.0683854 ,  0.59686682]])
```

In [122]:
```
from sklearn import svm

C = 1.0
svc = svm.SVC(kernel='linear', C=C)
```

In [123]:
```
cv_scores = cross_val_score(svc, all_features_scaled, target_feature, cv=10)

cv_scores.mean()
```

Out[123]:
```
0.7035198135198135
```

KNN

Using neighbors.KNeighborsClassifier . Starting with a K of 10. K is an example of a hyperparameter - a parameter on the model itself which may need to be tuned for best results on the data set

In [124]:
```
from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(n_neighbors=10)
cv_scores = cross_val_score(clf, all_features_scaled, target_feature, cv=10)

cv_scores.mean()
```

Out[124]: 0.6286480186480187

Choosing K is tricky, so we can't discard KNN until we've tried different values of K. We will write a for loop to run KNN with K values ranging from 1 to 50 and see if K makes a substantial difference.¶

In [125]:
```
for n in range(1, 50):
    clf = neighbors.KNeighborsClassifier(n_neighbors=n)
    cv_scores = cross_val_score(clf, all_features_scaled, target_feature, cv=10)
    print (n, cv_scores.mean())
```

```
1 0.5241025641025641
2 0.6117482517482518
3 0.5779487179487179
4 0.6025407925407926
5 0.601025641025641
6 0.6256177156177156
7 0.617925407925408
8 0.6286480186480187
9 0.6117482517482518
10 0.6286480186480187
11 0.6163636363636364
12 0.6378554778554779
13 0.6194405594405594
14 0.6470862470862471
15 0.6486713286713287
16 0.6516783216783217
17 0.6501631701631702
18 0.6715617715617717
19 0.6639160839160839
20 0.6822843822843824
21 0.6853846153846155
22 0.6991142191142192
23 0.6807459207459208
24 0.7021911421911422
25 0.6991375291375291
26 0.7052214452214451
27 0.7128671328671329
28 0.7097668997668998
29 0.7021911421911422
30 0.7112820512820514
31 0.7175291375291376
32 0.7097902097902098
33 0.7068065268065269
34 0.7051981351981352
35 0.7068531468531469
36 0.7113053613053614
37 0.7097668997668999
```

```
38 0.7113053613053614
39 0.7097902097902098
40 0.7051515151515153
41 0.7097902097902098
42 0.7066899766899768
43 0.7097668997668999
44 0.7143822843822845
45 0.715920745920746
46 0.7174592074592075
47 0.7174592074592075
48 0.7174592074592075
49 0.718997668997669
```

Naive Bayes

Now lets try naive_bayes.MultinomialNB and see how does its accuracy stack up¶

In [126]:

```python
from sklearn.naive_bayes import MultinomialNB

scaler = preprocessing.MinMaxScaler()
all_features_minmax = scaler.fit_transform(independent_features)

clf = MultinomialNB()
cv_scores = cross_val_score(clf, all_features_minmax, target_feature, cv=10)

cv_scores.mean()
```

Out[126]:

```
0.7050815850815851
```

Revisiting SVM

As of now SVM got the highest accuracy . lets now try if the efficiency of the model can be increased by Hyperparameter tuning svm.SVC may perform differently with different kernels. We wil try the rbf, sigmoid, and poly kernels and see what the best-performing kernel is.

In [127]:
```
C = 1.0
svc = svm.SVC(kernel='rbf', C=C)
cv_scores = cross_val_score(svc, all_features_scaled, target_feature, cv=10)
cv_scores.mean()
```

Out[127]: 0.7067365967365968

In [128]:
```
C = 1.0
svc = svm.SVC(kernel='sigmoid', C=C)
cv_scores = cross_val_score(svc, all_features_scaled, target_feature, cv=10)
cv_scores.mean()
```

Out[128]: 0.6636829836829838

In [129]:
```
C = 1.0
svc = svm.SVC(kernel='poly', C=C)
cv_scores = cross_val_score(svc, all_features_scaled, target_feature, cv=10)
cv_scores.mean()
```

Out[129]: 0.698974358974359

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```