Connor Hennen
Kaggle.com Terrorism Dataset Analysis

Global Terrorism 1970-2016

Machine learning, data analysis, data visualization, regex, plotting and more.

This near 50 years worth of terrorism data has endless applications. I sort of just dove deeply into a few different canals. If I could try to narrow it I'll go through all the things I did:

1) Machine learning classifier models, with the target variable being the terrorist group (Taliban, ISIL, IRA, Boko Haram, etc).

OneHotEncoding:

```
def getEncodedMatrices():
    X = df10.iloc[:, :-1].values
    #y = df10.iloc[:, 9].values ----- ends at 8 now that we deleted City feature
    y = df10.iloc[:, 8].values

    #Encode categorical variables
    from sklearn.preprocessing import LabelEncoder, OneHotEncoder

    labelencoder_X = LabelEncoder()
    #Because all the features' numbers refer to
    #types, they are actually categorical, except the Goal feautures (which are already one-hot
    #encoded), only the string features need the label econdoder step though

    X[:, 2] = labelencoder_X.fit_transform(X[:, 2])

    onehotencoder = OneHotEncoder(categorical_features = [0,1,2,3,4])#Don't one hot encode the goal columns as the
    X = onehotencoder.fit_transform(X).toarray()

    # Encoding the Dependent Variable
    labelencoder_y = LabelEncoder()
    y = labelencoder_y.fit_transform(y)

    return X,y

X,y = getEncodedMatrices()
```

OneHotEncoding, Feature Variable Matrix Outcome:

```
print type(X)
print X[:3]
print type(df10)
print df10[:3]
```

```
<type 'numpy.ndarray'>
[[ 0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   1.   0.   1.   1.]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   1.   0.   1.   1.]
 [ 0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   1.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.
   0.   1.   1.   1.]]
<class 'pandas.core.frame.DataFrame'>
   AttackType  Country Decade  TargetType  WeaponType  GoalNotHumanitarian  \
0           9      160   dec7           4          13                    0
1           9      160   dec7           4          13                    0
2           2      603   dec7          17           8                    1

   GoalPoliEconReligion  GoalSendMessage                    GroupName
0                     1                1       New People's Army (NPA)
1                     1                1       New People's Army (NPA)
2                     1                1  Irish Republican Army (IRA)
```

OneHotEncoding, Target Variable Array Outcome + Dictionary to Track Labels (useful later):

```
print [i for i in df10['GroupName'][:25]]
print y[:25]
```

```
["New People's Army (NPA)", "New People's Army (NPA)", 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', "New People's Army (NPA)", 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)', 'Irish Republican Army (IRA)']
[6 6 3 3 3 3 3 3 3 3 3 3 3 3 6 3 3 3 3 3 3 3 3 3 3]
```

```
#Get a list that has one index corresponding to each unique terror group
groupIndices = []
for g in top10Groups:
    groupIndices.append(list(df10.iloc[:, 8].values).index(g))

#GroupLabelDict will store the groups as values and their encoded labels as keys
groupLabelDict = {y[i]:df10.iloc[:, 8].values[i] for i in groupIndices}
print groupLabelDict
```

```
{0: 'Al-Shabaab', 1: 'Boko Haram', 2: 'Farabundo Marti National Liberation Front (FMLN)', 3: 'Irish Republican Army
(IRA)', 4: 'Islamic State of Iraq and the Levant (ISIL)', 5: "Kurdistan Workers' Party (PKK)", 6: "New People's Army
```

Logistic Regression Model, 99.7% accuracy:

```
#Computing accuracy score
from sklearn.metrics import accuracy_score
accScoreLogistic = accuracy_score(yPred,yTest)

print 'Logistic regression accuracy score is ' + str(round(accScoreLogistic,4)*100) + '%.\n'
print 'Our model appears to do very well, with hardly any classifications for almost every terror group.\n'
print 'The most errors (11) come from misclassifying the ' + groupLabelDict[5] + ' as\n' + groupLabelDict[4] + ', p
print 'Confusion matrix:'
print confusionMatLogistic
```

```
Accuracy score is 99.7%.

Our model appears to do very well, with hardly any classifications for almost every terror group.

The most errors (11) come from misclassifying the Kurdistan Workers' Party (PKK) as
Islamic State of Iraq and the Levant (ISIL), presumably because the Kurdistan Workers' Party and ISIL have had
significant country and decade overlap.

Confusion matrix:
[[ 538    0    0    0    0    0    0    0    0    0]
 [    0  409    0    0    0    0    0    0    0    0]
 [    0    0  708    0    0    0    0    0    0    0]
 [    0    0    0  518    0    0    0    0    0    0]
 [    1    0    0    0  835   11    1    0    0    1]
 [    0    0    0    2    2  406    0    0    0    0]
 [    0    0    0    0    0    0  475    0    0    0]
 [    0    0    0    0    0    0    0  537    0    0]
 [    0    0    0    0    0    0    0    2  887    0]
 [    0    0    0    0    0    0    0    0    0 1315]]
```

KNN Regression Model Attempt #1, 98.27% Accuracy:

```
#Accuracy score
from sklearn.metrics import accuracy_score
accScoreKNN = accuracy_score(yPred,yTest)

print '\nAccuracy score is ' + str(round(accScoreKNN,4)*100) + '%.\n'
print 'Confustion matrix:'
print confusionMatKNN
```

```
Accuracy score is 98.27%.

Confustion matrix:
[[ 536    0    1    0    0    0    0    0    0    1]
 [    3  400    0    0    2    1    0    0    0    3]
 [    1    0  702    0    0    0    0    0    3    2]
 [    2    0    0  507    0    2    1    0    5    1]
 [    1    0    0    2  823   11    1    0    1   10]
 [    0    0    0    0    9  391    0    1    4    5]
 [    0    0    0    2    1    0  465    0    3    4]
 [    1    0    0    0    0    0    3  522    3    8]
 [    2    0    2    1    0    0    0    3  880    1]
 [    2    1    0    0    2    0    1    1    1 1307]]
```

In [42]:

```
print '\nOur logistic model actually seems to be more accurate with the current parameters. Let\'s try tinkering wi
```

```
Our logistic model actually seems to be more accurate with the current parameters. Let's try tinkering with the KNN p
arameters.
```

KNN Regression Model Tuning Attempt, 98.99% Accuracy:

```
#Let's try to find the most accurate combination of parameters for n_neighbors and p, between 1 and 4 and 1 and 3,

bestAccScore = 0
#the best params end up being 1 and 1. Don't actually run this code unless you want to wait about 5 minutes (:
for n in range(1,5):
    for pVal in range(1,3):
        classifier = KNeighborsClassifier(n_neighbors = n, p = pVal)
        classifier.fit(xTrain, yTrain)

        #Test set predictions
        yPred = classifier.predict(xTest)
        currAccScore = accuracy_score(yPred,yTest)

        if currAccScore > bestAccScore:
            bestN = n
            bestP = pVal
            bestAccScore = currAccScore

classifier = KNeighborsClassifier(n_neighbors = 1, p = 1)
classifier.fit(xTrain, yTrain)
#Test set predictions
yPred = classifier.predict(xTest)
bestAccScore = accuracy_score(yPred,yTest)
```

```
print 'For KKN models, our best accuracy (without going through an too exhaustive of a tuning search) has an accu'
'racy of \n' + str(round(bestAccScore,4)*100) + '%. Though slightly improved, the logistic model remains more' \
+ ' accurate.'
```

```
For KKN models, our best accuracy (without going through an too exhaustive of a tuning search) has an accuracy of
98.99%. Though slightly improved, the logistic model remains more accurate.
```

Decoding into an interpretable dataframe:

Using, the same random seed to split the OneHotEncoded unindexed matrices and the indexed pandas Dataframe, I was able to keep track of what indexes were being predicted. Further, using the dictionary which stored the terror groups numeric labels, I was able to convert the labels back into group names and append a prediction column that makes the dataframe easily interpretable.

Decoding the prediction labels w/ dictionary created earlier:

```
predgroups = list(yPredLog)
for p in range(len(predgroups)):
    predgroups[p] = groupLabelDict[predgroups[p]]
```

Using same random seed to keep indices parallel:

```
#Logistic model implementation:

# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split


xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size = 0.2, random_state = 61495)

```

```
DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all
the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different fr
om that of this module. This module will be removed in 0.20. [cross_validation.py:41]
```

```
fakeX = df10.iloc[:, :-1]
fakeY = df10.iloc[:, 8]
xTrain1, xTest1, yTrain1, yTest1 = train_test_split(fakeX, fakeY, test_size = 0.2, random_state = 61495)

trainingIndices = xTrain1.index
testingIndices = xTest1.index


```

The first two columns are the real group versus the predicted group.

In [95]:

```
freshDF2 = freshDF.copy()
freshDF2 = freshDF2.loc[[i for i in testingIndices], :]

freshDF2 = freshDF2.reset_index(drop=True)
freshDF2['Prediction'] = predgroups

freshDF2 = freshDF2.rename(columns={'AttackTypeDef':'AttackType','CountryDef':'Country','Year':'Year','TargetTypeDe
freshDF2names = list(freshDF2)
temp = freshDF2names[0]
freshDF2names[0] = freshDF2names[8]
freshDF2names[8] = temp

temp = freshDF2names[1]
freshDF2names[1] = freshDF2names[9]
freshDF2names[9] = temp

myQuickSort(freshDF2names,2,len(freshDF2names)-1)

freshDF2 = freshDF2[freshDF2names]
freshDF2
```

	TrueTerrorGroup	PredictedTerrorGroup	AttackType	Country	GoalSendMessage	OutsideOfHumanLaw	PoliticalEconomicOrReligiousMotivated	TargetType
0	Taliban	Taliban	Unknown	Afghanistan	1	1	1	Police
1	Farabundo Marti National Liberation Front (FMLN)	Farabundo Marti National Liberation Front (FMLN)	Facility/Infrastructure Attack	El Salvador	1	1	1	Business
2	Irish Republican Army (IRA)	Irish Republican Army (IRA)	Assassination	United Kingdom	1	0	1	Military
3	Islamic State of Iraq and the Levant (ISIL)	Islamic State of Iraq and the Levant (ISIL)	Bombing/Explosion	Iraq	1	1	1	Private Citizens & Property
4	Shining Path (SL)	Shining Path (SL)	Assassination	Peru	1	1	1	Private Citizens & Property
5	Farabundo Marti National Liberation Front (FMLN)	Farabundo Marti National Liberation Front (FMLN)	Bombing/Explosion	El Salvador	1	1	1	Utilities
6	Farabundo Marti							

Over here

2) Text extraction (and applications)
without nltk

The columns for motives, summary, and (the most striking feature in the end) targets of the attacks, were unstructured text. Instead of using nltk to just grab the proper nouns, places, organizations, etc.,

I just used regex, along with a webscraped list of stop words.

```python
def extractKeyWords(df,colname,summ):
    '''
    Function that does text extraction for variable's whose elements are described by unstandardized text. Aims to
    capture meaningful words (people, religions, countries, places, etc) without the use of an NLP module, but with
    only re, urllib and numpy (just for the np.NaN).

    Param1: The terror df
    Param2: The name of feature we are trying to extract text from
    Param3: If true, we are looking at the summary feature and have to sub out a few minor things.


    Return: The same terrorDF but updated with the given column only containing keyword elements. It also
    passes the input dataframe by reference, so it is altered in the same way.
    '''
    from urllib import urlopen
    import re
    import numpy as np
    summarySample = list(df[colname])

    stopWordPage=urlopen('http://www.lextek.com/manuals/onix/stopwords1.html').read()
    stopWordPage = str(stopWordPage)
    stopWordPage = stopWordPage.split('</pre>')[0]
    stopWordPage = stopWordPage.split('#')[len(stopWordPage.split('#'))-1]
    stopWords = stopWordPage.split() + ['specific','sources','people']

    for i in range(len(summarySample)):
        if summarySample[i] == summarySample[i]:
            summarySample[i] = str(summarySample[i])
            currSample = summarySample[i].split()
            fixedSample = []
            for j in range(len(currSample)):
                if currSample[j].lower() not in stopWords and currSample[j][0] == currSample[j][0].upper() and len(
                    temp = re.sub(r'\.','',currSample[j])
                    temp = re.sub(r'\'s','',temp)
                    temp = re.sub(r'\"','',temp)
                    temp = re.sub(r'\'','',temp)
                    temp = re.sub(r'!','',temp)
                    temp = re.sub(r',','',temp)
                    temp = re.sub(r';','',temp)
                    temp = re.sub(r'[(]','',temp)
                    temp = re.sub(r'[)]','',temp)
                    motiveStr3 = re.sub(r'\.','',temp)
                    motiveStr3 = re.sub(r'  ',' ',motiveStr3)
                    motiveStr3 = re.sub(r'\"','',motiveStr3)
                    motiveStr3 = re.sub(r'\'','',motiveStr3)
                    motiveStr3 = re.sub(r',','',motiveStr3)
                    motiveStr3 = re.sub(r'[(]','',motiveStr3)
                    motiveStr3 = re.sub(r'[)]','',motiveStr3)
                    motiveStr3 = re.sub(r';','',motiveStr3)
                    motiveStr3 = re.sub(r'[)]','',motiveStr3)
                    motiveStr3 = re.sub(r'!','',motiveStr3)
                    motiveStr3 = re.sub(r'”','',motiveStr3)
                    motiveStr3 = re.sub(r'[$]\d+?(?=[^\d])','',motiveStr3)
                    motiveStr3 = re.sub(r'“','',motiveStr3)
                    motiveStr3 = re.sub(r'/',' ',motiveStr3)
                    motiveStr3 = re.sub(r'\\x.+','',motiveStr3)
                    motiveStr3 = re.sub(r'\\','',motiveStr3)
                    temp = re.sub(r'  ',' ',motiveStr3)
                    if summ == True:
                        temp = re.sub(r'\d','',temp)
                        temp = re.sub(r':','',temp)
                        temp = re.sub(r'[$]','',temp)
                    if re.search(r'\w',temp):
                        fixedSample.append(temp)

            if len(fixedSample) > 0:
                df.at[i,colname] = myMergeSort(list(set(fixedSample)))
            else:
                df.at[i,colname] = np.NaN

    return df


motiveKeys = extractKeyWords(terrorDF,'motive',False)
```

Behold…my text extraction function

Note the differnces, the second set of elements has only keywords, no dates, no stopwords, no puntcuation marks, etc

```
#Added a copy for illustration purposes (as the input df is passed by referenece)
terrorDFcopy = terrorDF.copy()
motiveKeys = extractKeyWords(terrorDFcopy,'motive',False)
```

```
summaryKeys = extractKeyWords(terrorDFcopy,'summary',True)
```

```
print terrorDF['summary'][5:10]
print summaryKeys['summary'][5:10]
```

```
5    1/1/1970: Unknown African American assailants ...
6                                                 NaN
7    1/2/1970: Unknown perpetrators detonated explo...
8    1/2/1970: Karl Armstrong, a member of the New ...
9    1/3/1970: Karl Armstrong, a member of the New ...
Name: summary, dtype: object
5    [African, American, Black, Cairo, Illinois, St...
6                                                 NaN
7    [California, Company, Edes, Electric, Gas, Oak...
8    [Armstrong, Gang, Gym, Karl, Madison, ROTC, Re...
9    [Armstrong, Gang, Headquarters, Karl, Lab, Mad...
Name: summary, dtype: object
```

```python
print 'At this point I have created a sparse density matrix that could be useful for a bag of words model. ' \
+ 'Although interesting, this was actually extremely time-consuming and at the end of the day, beyond the scope ' \
+ 'of things at this time, so Im going to have to let this pursuit go for the time being'
print newDF['MotiveEncodings'][3:8]
print newDF['Motive'][3:8]
print '\nElements 4, 5, and 7 are obviously related, and that fact is described in the binary matrix, but Im ' \
+ 'not sure where to go with this right now, aside from a bag of words classifier which isnt really within the ' \
+ 'scope of things'
```

```
At this point I have created a sparse density matrix that could be useful for a bag of words model. Although interest
ing, this was actually extremely time-consuming and at the end of the day, beyond the scope of things at this time, s
o Im going to have to let this pursuit go for the time being
3    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
4    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
5    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
6    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
7    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
Name: MotiveEncodings, dtype: object
3                              NaN
4                ['Vietnam', 'War']
5                ['Vietnam', 'War']
6                              NaN
7    ['Protest', 'Vietnam', 'War']
Name: Motive, dtype: object

Elements 4, 5, and 7 are obviously related, and that fact is described in the binary matrix, but Im not sure where to
go with this right now, aside from a bag of words classifier which isnt really within the scope of things
```

Unfortunately, I just had wasted several hours..or so I thought…

```python
print 'We see that elements 5 and 7 do have a high cosine simularity here. But again, beyond scope- letting go of ' \
+ 'for now'
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
allMotives = motiveLists
def cosine_sim(text1, text2):
    tfidf = vectorizer.fit_transform([text1, text2])
    return ((tfidf * tfidf.T).A)[0,1]

print cosine_sim(' '.join(allMotives[5]), ' '.join(allMotives[7]))
```

```
We see that elements 5 and 7 do have a high cosine simularity here. But again, beyond scope- letting go of for now
0.709297266606
```

3) Wordclouds with the extracted keywords

Then I thought of wordclouds, which does use a few libraries from within course scope – scipy, matplotlib, random, and os. And of course, wordcloud.

In [188]:

```
print 'We can however create a wordcloud of motives.'

from os import path
from scipy.misc import imread
import matplotlib.pyplot as plt
import random
from wordcloud import WordCloud

#We don't wan't to count unknwn as a motive..
motiveStr1 = re.sub('Unknown ','',motiveStr)

wordcloud = WordCloud(font_path='/Library/Fonts/Verdana.ttf',
                      relative_scaling = 0.3,
                      random_state=61495,
                      ).generate(motiveStr1)
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

```
We can however create a wordcloud of motives.
```

Then I thought of wordclouds, which does use a few libraries from within course scope – scipy, matplotlib, random, and os.

In [189]:

```
print 'We can also do a word cloud for summaries'

allSummaries = newDF['summary']
summaryStr = ''
import numpy as np

for summaryL in allSummaries:

      if summaryL == summaryL:
         summaryL = re.sub(',','',summaryL[0])
         a = re.sub('[[]','',''.join(summaryL))
         a = re.sub('[]]','',a)
         a = re.sub('\'','',a)
         summaryStr += a + ' '

wordcloud1 = WordCloud(font_path='/Library/Fonts/Verdana.ttf',
                       relative_scaling = .3,
                       max_words = 200,
                       random_state=61495,
                       ).generate(summaryStr)
plt.imshow(wordcloud1)
plt.axis("off")
plt.show()
```

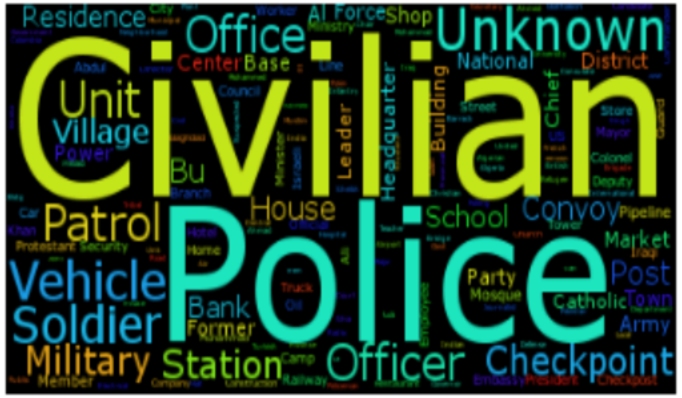
We can also do a word cloud for summaries

And now the most chilling thing I came across in the dataset… Civilian, school, office, market, house… These were some of the most frequently targeted people/places/things of 50 years of terror.

In [190]:

```
print 'Finally, a word cloud for the targets, which is the most telling of all.'

wordcloud2 = WordCloud(font_path='/Library/Fonts/Verdana.ttf',
                       relative_scaling = .3,
                       max_words = 200,
                       random_state=61495,
                       ).generate(targetStr)
plt.imshow(wordcloud2)
plt.axis("off")
plt.show()

```

```
Finally, a word cloud for the targets, which is the most telling of all.
```

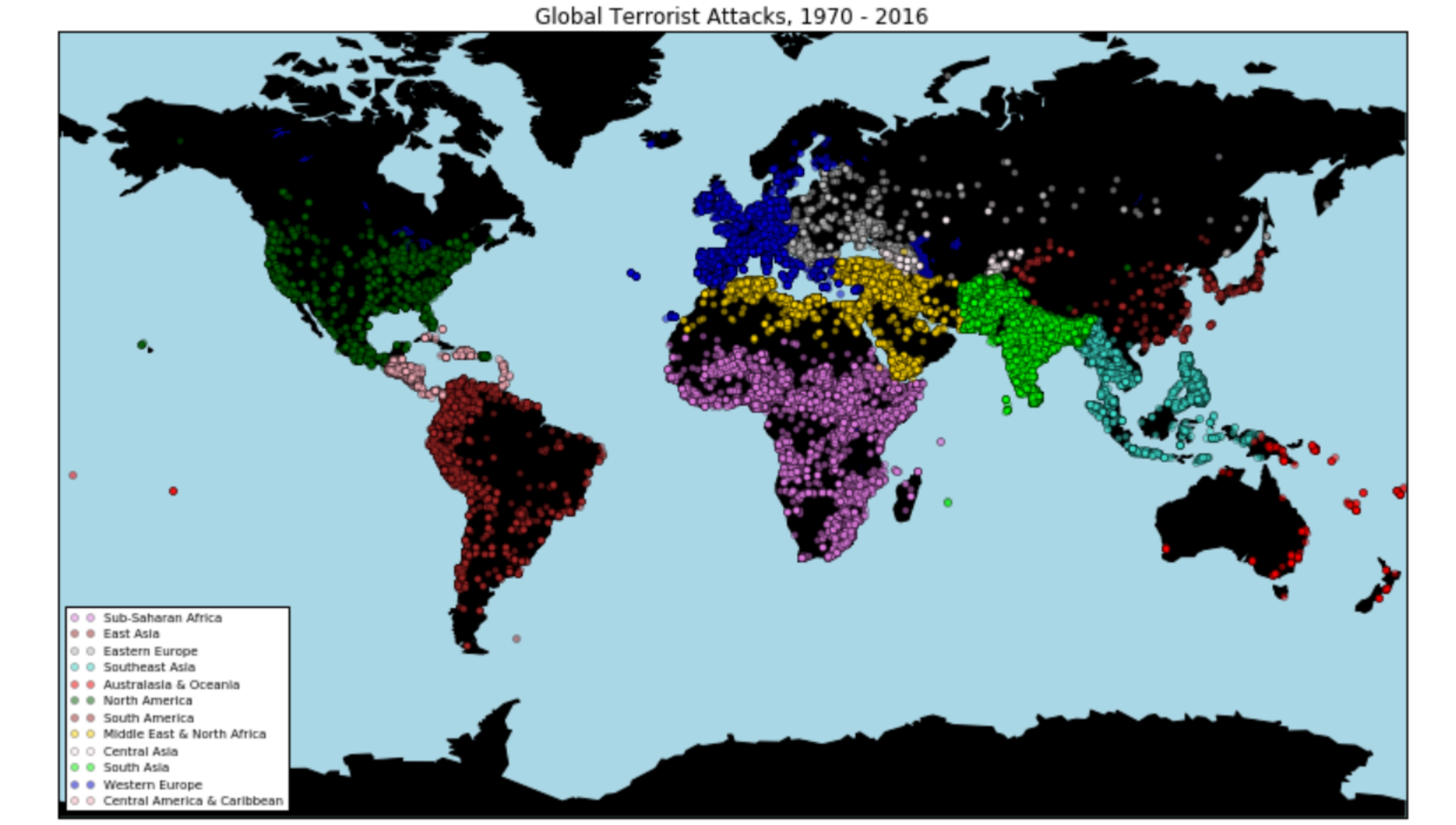


4) Plotting global terror with matplotlib and Basemap

In [213]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import matplotlib.colors

def getRGB(col):
    rbgfinder = matplotlib.colors.ColorConverter()
    return rbgfinder.to_rgb(col)


regions = list(set(terrorDF2.region_txt))

#I used the getRGB function to play with the colors and arrive at these long decimals.
colors = [(.9,.5,.9), (0.6470588235294118, 0.16470588235294117, 0.16470588235294117), (0.7529411764705882, 0.752941
terrorDF2 = terrorDF
plt.figure(figsize=(15,8))

coldWorld = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat_ts=20,resolution
coldWorld.drawcountries()
coldWorld.drawcoastlines()
coldWorld.fillcontinents(color='black',lake_color='darkblue', zorder = 1)
coldWorld.drawmapboundary(fill_color='lightblue')

def plotAttacks(clr, rgn):
    xCoord, yCoord = coldWorld([i for i in terrorDF2.longitude[terrorDF2.region_txt == rgn]]
,[j for j in terrorDF2.latitude[terrorDF2.region_txt == rgn]]
)
    coldWorld.plot(xCoord, yCoord, "o", color = clr, label = rgn, markersize =4, alpha = .5)

for c, rgn in enumerate(regions):
    plotAttacks(colors[c],rgn)

plt.legend(loc ='lower left', prop= {'size':7})
plt.title("Global Terrorist Attacks, 1970 - 2016")
plt.show()
```

Terror knows no bounds



Other things I did:

I created my own mergesorting, quicksorting, mean, median, and mode functions from scratch (and used them on my dataframes).

I webscrabed the PDF that explains how what the variable values represent.

I did a lot of descriptive statistics, and could have easily done more, but the other things seemed higher priority.

I worked with and manipulated handily all of the major data structures – lists, strings, ints, pandas dataframes, sets, dictionaries, etc. In other words, list and dict comprehension, dataframe cleaning and preprocessing, changing shape, turning into a matrix and back again, dropping rows, adding rows, splitting strings, implementing regex on strings, etc. I think scrolling through the .html or the notebook will be telling as to how much thought when into this. It is also a very cool dataset.