Machine Learning

A Machine Learning Approach for Enhancing Defence Against Global Terrorism

I. Getting Started

The objective of this work is to predict the region and country of a terrorist attack using machine learning approaches. The work has been carried out upon the Global Terrorism Database (GTD), which is an open database containing list of terrorist activities from 1970 to 2017. Six machine learning algorithms have been applied on a selected set of features from the dataset to achieve an accuracy of up to 82%. The results suggest that it is possible to train machine learning models in order to predict the region and country of terrorist attack if certain parameters are known. It is postulated that the work can be used for enhancing security against terrorist attacks in the world.

In [1]:
```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display # Allows the use of display() for DataFram
es

# Pretty display for notebooks
%matplotlib inline
plt.style.use('fivethirtyeight')

# Load the accepted loan dataset
# low_memory and skiprows in read_csv because the file is large and it leads t
o the Lending Club website
try:
    #"https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"
    #gtd_data = pd.read_csv("gdt.csv", low_memory = False, skiprows = 1, encod
ing = "ISO-8859-1")
    gtd_data = pd.read_csv("gdt.csv", low_memory = False, skiprows = 1, encodi
ng = "ISO-8859-1")
    print("The GTD dataset has {} samples with {} features.".format(*gtd_data.
shape))
except Exception as e:
    print(str(e))
    print("The GTD  dataset could not be loaded. Is the dataset missing?")
```

```
The GTD dataset has 181691 samples with 135 features.
```

Introduction To The Data

Prediction of terrorism activities is an important area of concern for researchers. . The large number of events makes it difficult to predict terrorist group responsible for some terrorist activity. The work in has tested machine learning approaches for classifying and analyzing global terrorist activity. The authors have explored supervised machine learning approaches to study terrorist activity, and then developed a model to classify historical events in Global Terrorism Database. They have released a new dataset as well named QFactors Terrorism, which collaborates event-specific features derived from the GTD with population-level demographic data from sources like United Nations and World Bank. Naive Bayes, decision trees, Linear Discriminant Analysis, k-nearest neighbors and random forest approaches have been implemented.

In [2]:
```
gtd_data.head()
```

Out[2]:

	eventid	iyear	imonth	iday	approxdate	extended	resolution	country	country_txt	re
0	197000000001	1970	7	2	NaN	0	NaN	58	Dominican Republic	
1	197000000002	1970	0	0	NaN	0	NaN	130	Mexico	
2	197001000001	1970	1	0	NaN	0	NaN	160	Philippines	
3	197001000002	1970	1	0	NaN	0	NaN	78	Greece	
4	197001000003	1970	1	0	NaN	0	NaN	101	Japan	

5 rows × 135 columns

In [3]:
```
gtd_data.iloc[55]
```

Out[3]:
```
eventid                                          197002080001
iyear                                                    1970
imonth                                                      2
iday                                                        8
approxdate                                                NaN
                                   ...
INT_LOG                                                     0
INT_IDEO                                                    1
INT_MISC                                                    0
INT_ANY                                                     1
related        197002080001, 197002080002, 197002090003
Name: 55, Length: 135, dtype: object
```

In [4]:
```
gtd_data = gtd_data.drop(['addnotes', 'scite1','scite2','scite2'],axis=1)
```

In [5]:
```
gtd_data.describe()
```

Out[5]:

	eventid	iyear	imonth	iday	extended	country
count	1.816910e+05	181691.000000	181691.000000	181691.000000	181691.000000	181691.00000(
mean	2.002705e+11	2002.638997	6.467277	15.505644	0.045346	131.96850
std	1.325957e+09	13.259430	3.388303	8.814045	0.208063	112.41453
min	1.970000e+11	1970.000000	0.000000	0.000000	0.000000	4.00000(
25%	1.991021e+11	1991.000000	4.000000	8.000000	0.000000	78.00000(
50%	2.009022e+11	2009.000000	6.000000	15.000000	0.000000	98.00000(
75%	2.014081e+11	2014.000000	9.000000	23.000000	0.000000	160.00000(
max	2.017123e+11	2017.000000	12.000000	31.000000	1.000000	1004.00000(

8 rows × 77 columns

Another notable thing to remove is to remove columns with more than 50% missing values. It would be time consumming and inefficient to deal with the tremendous amount of missing values from these columns.

The Cleaned Data will stores in gtd_data.csv file

In [6]:
```
# count half point of the dataset.
half_point = len(gtd_data) / 2
gtd_data = gtd_data.dropna(thresh=half_point, axis=1)
# we save the new file
gtd_data.to_csv('gtd_data.csv', index=False)
```

We reload the data in the notebook and take a look at the first row.

In [7]:
```
gtd_data = pd.read_csv('gtd_data.csv', low_memory = False)
gtd_data.drop_duplicates()
print("The GTD dataset has {} samples with {} features.".format(*gtd_data.shap
e))
gtd_data.iloc[0]
```

```
The GTD dataset has 181691 samples with 57 features.
```

Out[7]:
```
eventid                          197000000001
iyear                                    1970
imonth                                      7
iday                                        2
extended                                    0
country                                    58
country_txt                Dominican Republic
region                                      2
region_txt         Central America & Caribbean
provstate                                 NaN
city                           Santo Domingo
latitude                              18.4568
longitude                            -69.9512
specificity                                 1
vicinity                                    0
summary                                   NaN
crit1                                       1
crit2                                       1
crit3                                       1
doubtterr                                   0
multiple                                    0
success                                     1
suicide                                     0
attacktype1                                 1
attacktype1_txt                 Assassination
targtype1                                  14
targtype1_txt      Private Citizens & Property
targsubtype1                               68
targsubtype1_txt               Named Civilian
corp1                                     NaN
target1                          Julio Guzman
natlty1                                    58
natlty1_txt                Dominican Republic
gname                                  MANO-D
guncertain1                                 0
individual                                  0
nperps                                    NaN
nperpcap                                  NaN
claimed                                   NaN
weaptype1                                  13
weaptype1_txt                         Unknown
weapsubtype1                              NaN
weapsubtype1_txt                          NaN
weapdetail                                NaN
nkill                                       1
nkillus                                   NaN
nkillter                                  NaN
nwound                                      0
nwoundus                                  NaN
nwoundte                                  NaN
property                                    0
ishostkid                                   0
dbsource                                 PGIS
INT_LOG                                     0
INT_IDEO                                    0
INT_MISC                                    0
```

```
INT_ANY                                          0
Name: 0, dtype: object
```

In [8]:
```
gtd_data.shape[1]
```

Out[8]:
```
57
```

As we have seen the Dataframe is cumbersome and we had to set the `low_memory` to `False` to avoid a warning message from the notebook. This is due to the numerous columns of the dataset. Let us now explore the dataset with the data dictionary as this will be useful as we go through the data and try to clean it.

One important thing to keep in mind is that we will need to be careful about data from the future, this type of leakage could throw off the predictions of our model. A clear example is information about the borrower after the loan was approved, this is not data that we would have at our disposal.

II. Analysis

Features Meaning and Usefulness

We will use the first entry of the `gtd_data.csv` file to explore the meaning of the remaining 52 columns.

In [9]:
```
first_entry = gtd_data.iloc[0]
first_entry.to_csv('first_entry.csv', index = True)
```

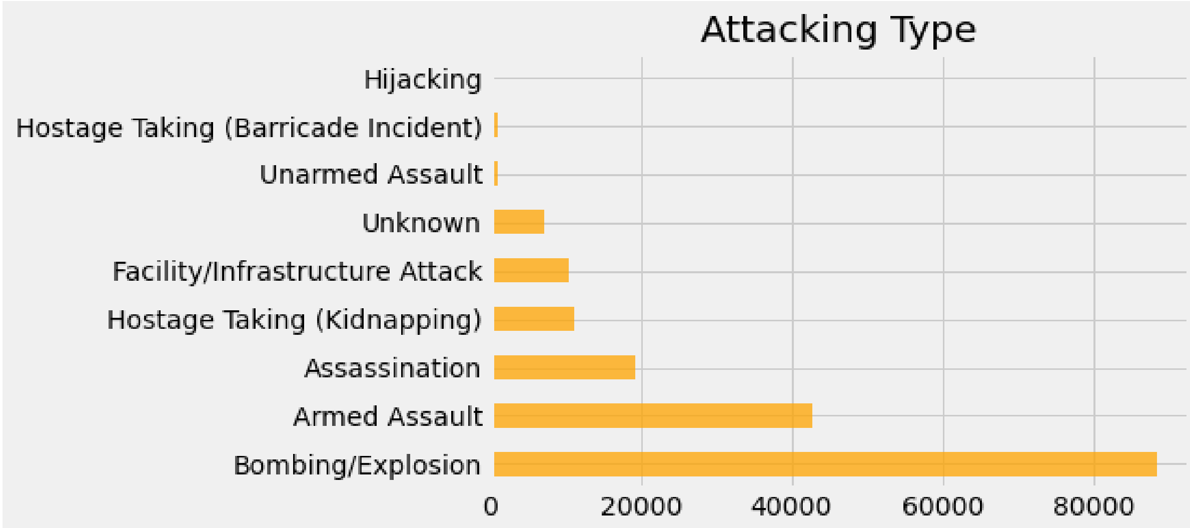
Target Column

The target column is a critical part when fitting this type of data to machine learning algorithms because it tries to make prediction based on the outcome that we want. In this particular case, we want to predict the loan status (*`attacktype1_txt`*) which can take many values in total.

In [10]:
```
gtd_data['attacktype1_txt'].value_counts()
```

Out[10]:
```
Bombing/Explosion                      88255
Armed Assault                          42669
Assassination                          19312
Hostage Taking (Kidnapping)            11158
Facility/Infrastructure Attack         10356
Unknown                                 7276
Unarmed Assault                         1015
Hostage Taking (Barricade Incident)      991
Hijacking                                659
Name: attacktype1_txt, dtype: int64
```

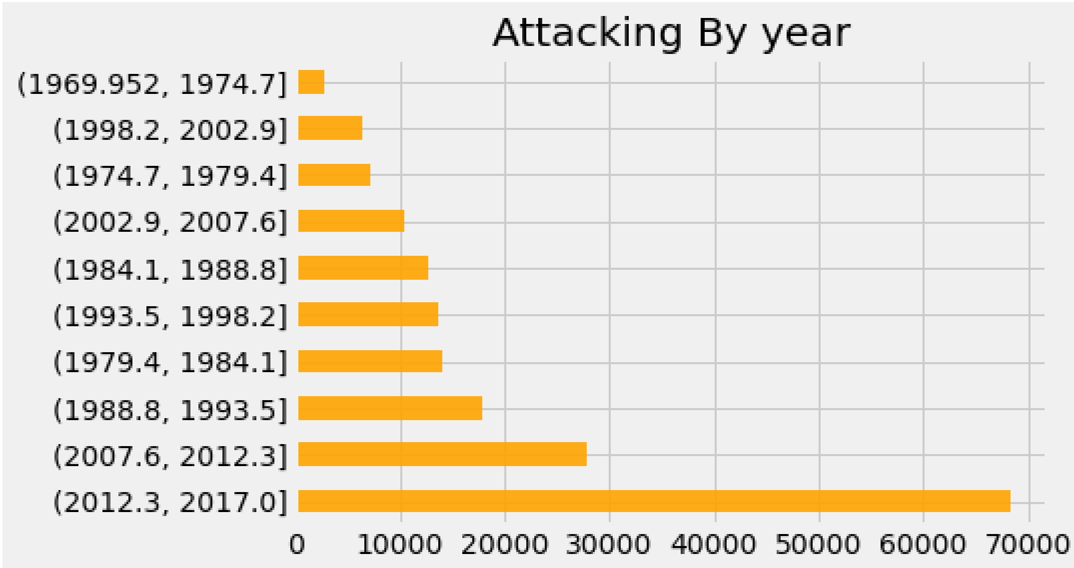
In [11]:
```
gtd_data['attacktype1_txt'].value_counts().plot(kind= 'barh', color = 'orange'
, title = 'Attacking Type', alpha = 0.75)
plt.show()
```



In [12]:
```
gtd_data['iyear'].value_counts(dropna=False).head(20)
```

Out[12]:
```
2014    16903
2015    14965
2016    13587
2013    12036
2017    10900
2012     8522
2011     5076
1992     5071
2010     4826
2008     4805
2009     4721
1991     4683
1989     4324
1990     3887
1988     3721
1984     3495
1994     3456
2007     3242
1997     3197
1987     3183
Name: iyear, dtype: int64
```

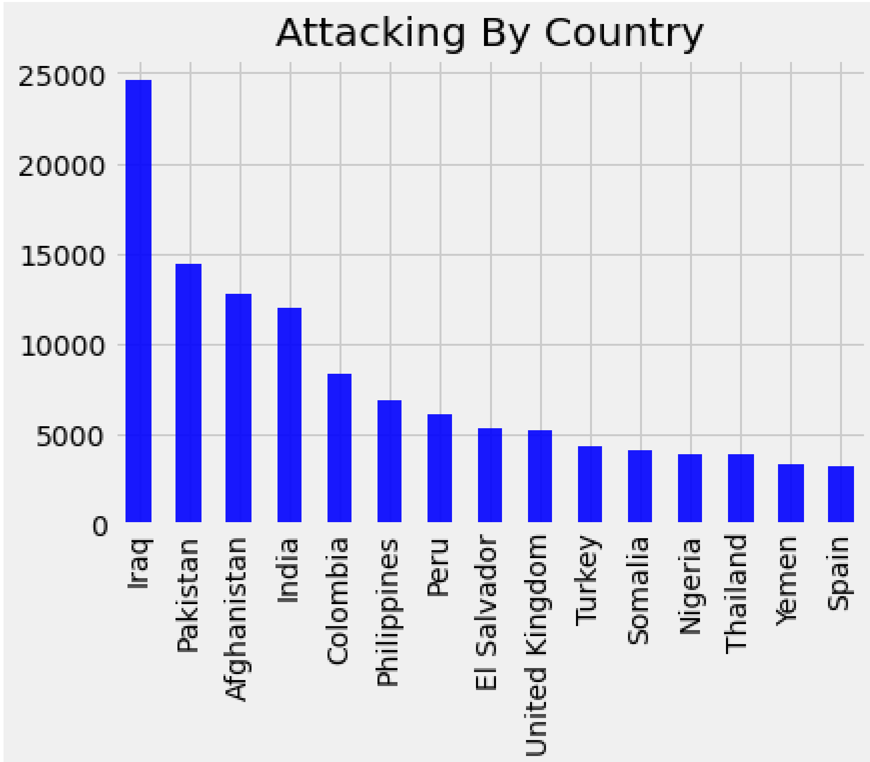
In [13]:
```
gtd_data['iyear'].value_counts(bins=10,dropna=False).head(20).plot(kind= 'bar
h', color = 'orange', title = 'Attacking By year', alpha = 0.90)
plt.show()
```



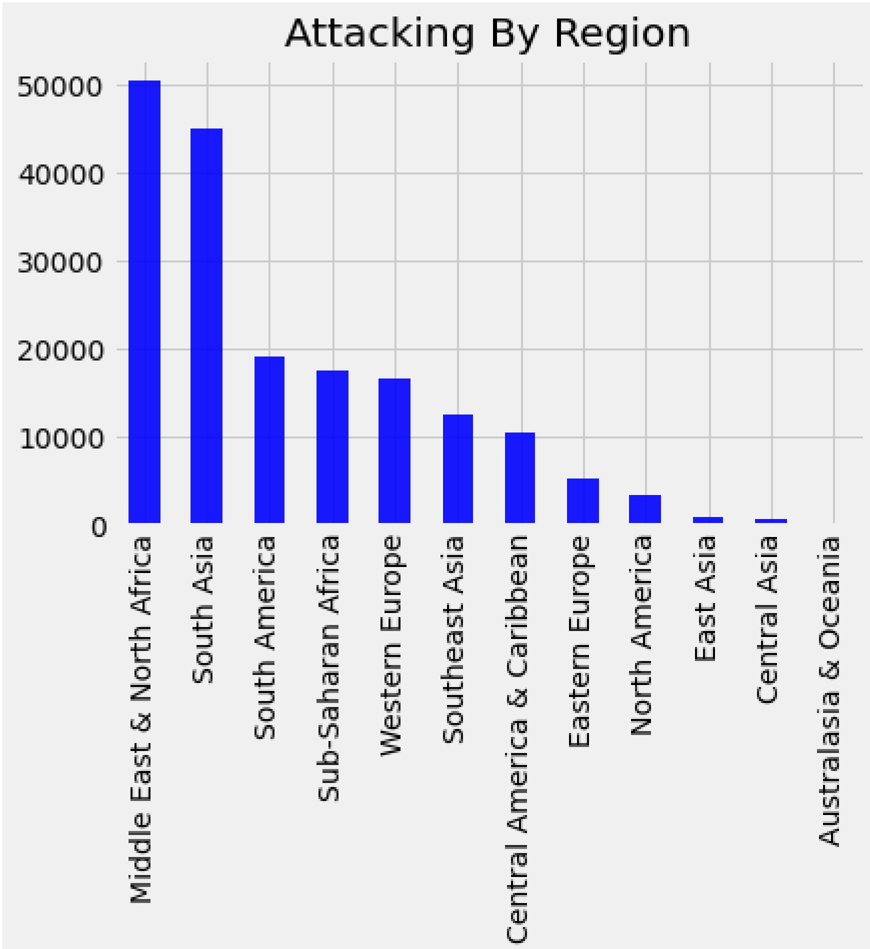
In [14]:
```
gtd_data['country_txt'].value_counts(dropna=True)
```

Out[14]:
```
Iraq                     24636
Pakistan                 14368
Afghanistan              12731
India                    11960
Colombia                  8306
                         ...
Wallis and Futuna            1
Antigua and Barbuda          1
Andorra                      1
New Hebrides                 1
International                1
Name: country_txt, Length: 205, dtype: int64
```

In [15]:
```
gtd_data['country_txt'].value_counts(dropna=False).head(15).plot(kind= 'bar',
color = 'Blue', title = 'Attacking By Country', alpha = 0.90)
plt.show()
```



In [16]:
```
gtd_data['region_txt'].value_counts()
gtd_data['region_txt'].value_counts().plot(kind= 'bar', color = 'Blue', title
= 'Attacking By Region', alpha = 0.90)
plt.show()
```



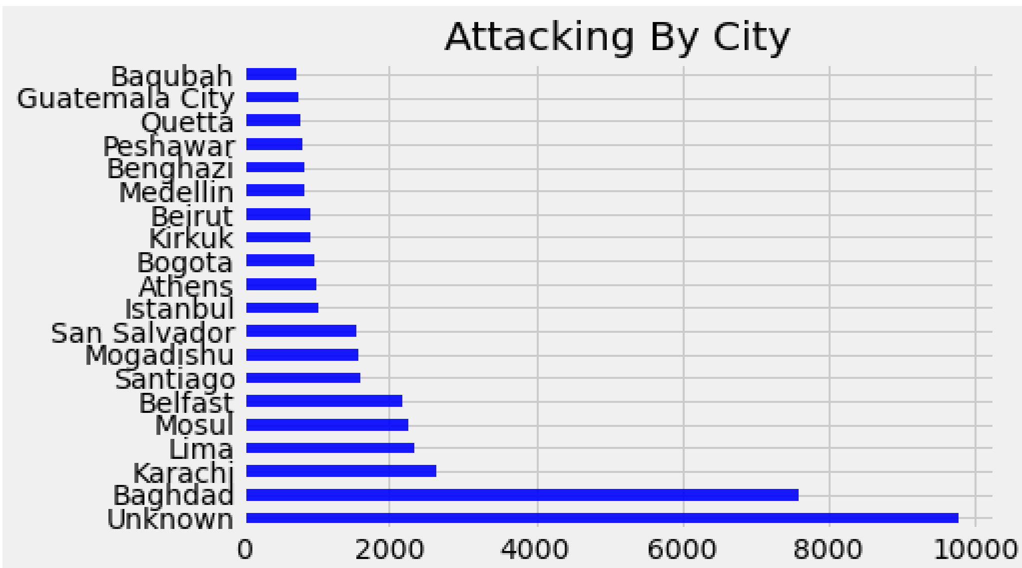
In [17]:
```
gtd_data['city'].value_counts()
```

Out[17]:
```
Unknown        9775
Baghdad        7589
Karachi        2652
Lima           2359
Mosul          2265
               ... 
Bushalingwa       1
Harem             1
Bowri Tana        1
Tabon-tabon       1
Selpang           1
Name: city, Length: 36674, dtype: int64
```

In [18]:
```
gtd_data['city'].value_counts()
```

Out[18]:
```
Unknown        9775
Baghdad        7589
Karachi        2652
Lima           2359
Mosul          2265
               ...
Bushalingwa       1
Harem             1
Bowri Tana        1
Tabon-tabon       1
Selpang           1
Name: city, Length: 36674, dtype: int64
```

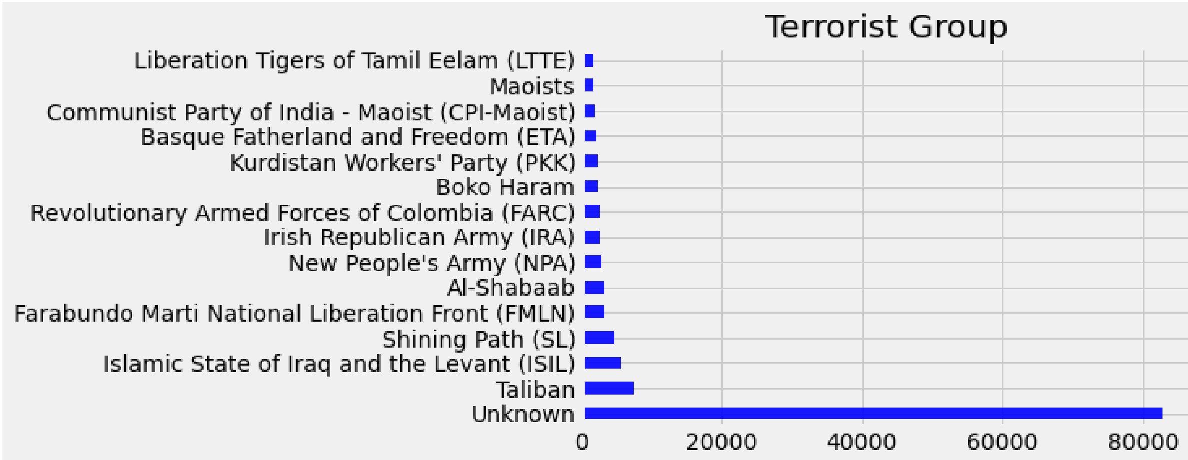
In [19]:
```
gtd_data['city'].value_counts().head(20).plot(kind= 'barh', color = 'Blue', title = 'Attacking By City', alpha = 0.90)
plt.show()
```



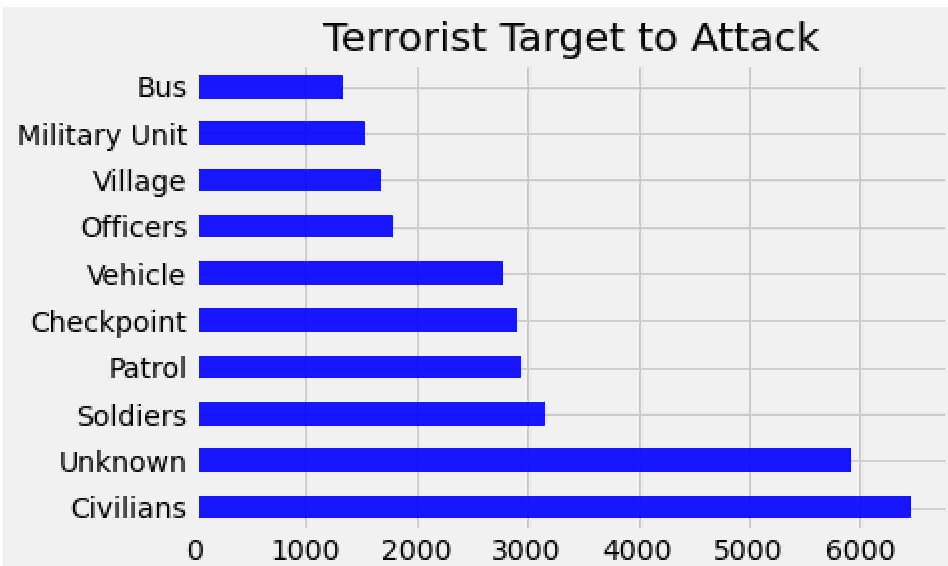
In [20]:
```
gtd_data['gname'].value_counts()
```

Out[20]:
```
Unknown                                             82782
Taliban                                              7478
Islamic State of Iraq and the Levant (ISIL)          5613
Shining Path (SL)                                    4555
Farabundo Marti National Liberation Front (FMLN)     3351
                                                    ...
Abu Salim Martyr's Brigade                              1
Khmer Serei Guerrillas                                  1
Revolted Persons of the Polytech School                 1
Comrades Organized in Partisan Nuclei                   1
Boer Sentries                                           1
Name: gname, Length: 3537, dtype: int64
```

In [21]:
```
gtd_data['gname'].value_counts().head(15).plot(kind= 'barh', color = 'Blue', t
itle = 'Terrorist Group', alpha = 0.90)
plt.show()
```

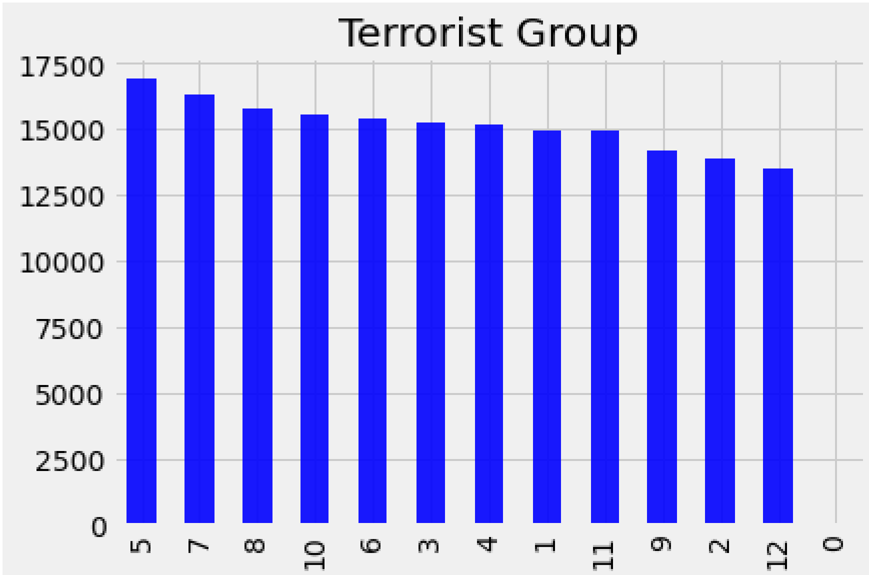


In [22]:
```
gtd_data['target1'].value_counts()
gtd_data['target1'].value_counts().head(10).plot(kind= 'barh', color = 'Blue',
title = 'Terrorist Target to Attack', alpha = 0.90)
plt.show()
```

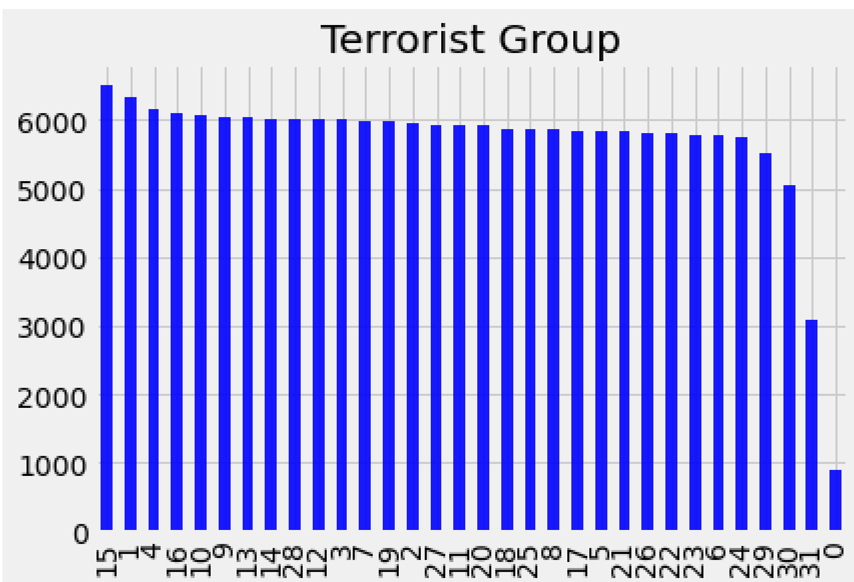


We have 9 possibility for `loan_status` and only 2 values are important in our model's binary classification; fully paid and charged off. These 2 values indicate the result of the loan outcome. We will remove the other possibilities and avoid "translating" those values into the binary possibility (fully paid or charged off).

In [23]:
```
gtd_data['imonth'].value_counts()
gtd_data['imonth'].value_counts().plot(kind= 'bar', color = 'Blue', title = 'T
errorist Group', alpha = 0.90)
plt.show()
```



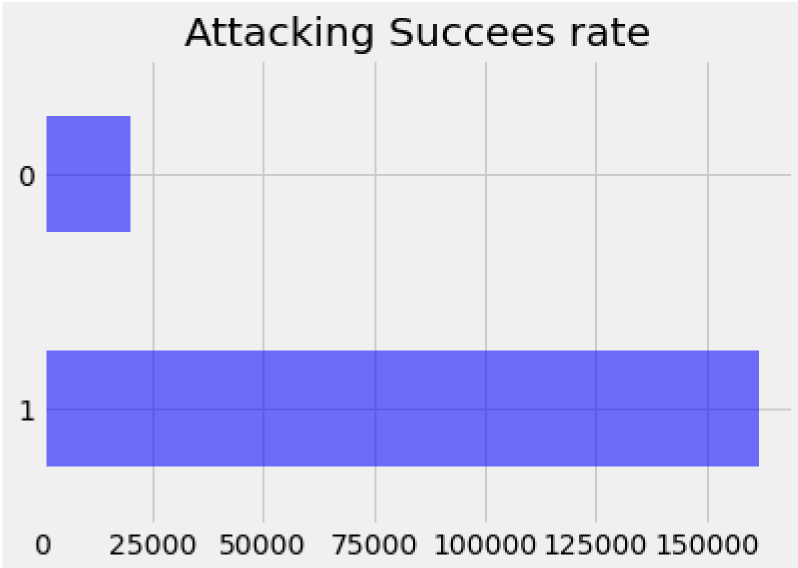
In [24]:
```
gtd_data['iday'].value_counts()
gtd_data['iday'].value_counts().plot(kind= 'bar', color = 'Blue', title = 'Ter
rorist Group', alpha = 0.90)
plt.show()
```



In [25]:
```
gtd_data = gtd_data[(gtd_data['success'] == 0) | (gtd_data['success'] == 1)]
```

1. Predict Attacking is success or failed.

In [26]:
```
gtd_data['success'].value_counts().plot(kind= 'barh', color = 'blue', title =
'Attacking Succees rate', alpha = 0.55)
plt.show()
```



We need to change the object value to numerical for the algorithm processing. Let's use a dictionary.

In [27]:
```
status_replace = {
    "success" : {
        1: "Success",
        0: "Filed",
    }
}
gtd_data = gtd_data.replace(status_replace)
```

In [28]:
```
gtd_data['success'].value_counts()
```

Out[28]:
```
Success    161632
Filed       20059
Name: success, dtype: int64
```

In [29]:
```
gtd_data.shape
```

Out[29]:
```
(181691, 57)
```

Final Data Cleaning

In [30]:
```
orig_columns = gtd_data.columns
drop_columns = []
for col in orig_columns:
    col_series = gtd_data[col].dropna().unique()
    if len(col_series) == 1:
        drop_columns.append(col)
gtd_data = gtd_data.drop(drop_columns, axis = 1)
drop_columns
```

Out[30]:
```
[]
```

In [31]:
```
#loan_data.drop(['pymnt_plan','initial_list_status','tax_liens','delinq_amnt','collections_12_mths_ex_med','application_type','acc_now_delinq','chargeoff_within_12_mths'],axis=1)

gtd_data.shape
```

Out[31]:
```
(181691, 57)
```

III. Methodology

Preparing The Features: Dealing With Missing Values

In [32]:
```
null_counts = gtd_data.isnull().sum()
null_counts
```

Out[32]:
```
eventid                 0
iyear                   0
imonth                  0
iday                    0
extended                0
country                 0
country_txt             0
region                  0
region_txt              0
provstate             421
city                  434
latitude             4556
longitude            4557
specificity             6
vicinity                0
summary             66129
crit1                   0
crit2                   0
crit3                   0
doubtterr               1
multiple                1
success                 0
suicide                 0
attacktype1             0
attacktype1_txt         0
targtype1               0
targtype1_txt           0
targsubtype1        10373
targsubtype1_txt    10373
corp1               42550
target1               636
natlty1              1559
natlty1_txt          1559
gname                   0
guncertain1           380
individual              0
nperps              71115
nperpcap            69489
claimed             66120
weaptype1               0
weaptype1_txt           0
weapsubtype1        20768
weapsubtype1_txt    20768
weapdetail          67670
nkill               10313
nkillus             64446
nkillter            66958
nwound              16311
nwoundus            64702
nwoundte            69143
property                0
ishostkid             178
dbsource                0
INT_LOG                 0
INT_IDEO                0
INT_MISC                0
```

```
INT_ANY                0
dtype: int64
```

In [33]:
```
gtd_data.shape
```

Out[33]:
```
(181691, 57)
```

Handling Non-Numeric Data Types

The data types of columns are important to look at and we will need to deal with non-numeric values in order to encode and use them in our machine learning algorithms.

In [34]:
```
print(gtd_data.dtypes.value_counts())
```

```
int64      21
float64    19
object     17
dtype: int64
```

In [35]:
```
object_columns_df = gtd_data.select_dtypes(include=["object"])
print(object_columns_df.iloc[0])
```

```
country_txt                   Dominican Republic
region_txt          Central America & Caribbean
provstate                                    NaN
city                               Santo Domingo
summary                                      NaN
success                                  Success
attacktype1_txt                    Assassination
targtype1_txt       Private Citizens & Property
targsubtype1_txt                  Named Civilian
corp1                                        NaN
target1                             Julio Guzman
natlty1_txt                   Dominican Republic
gname                                     MANO-D
weaptype1_txt                            Unknown
weapsubtype1_txt                             NaN
weapdetail                                   NaN
dbsource                                    PGIS
Name: 0, dtype: object
```

In [36]:
```
gtd_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 181691 entries, 0 to 181690
Data columns (total 57 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   eventid           181691 non-null  int64  
 1   iyear             181691 non-null  int64  
 2   imonth            181691 non-null  int64  
 3   iday              181691 non-null  int64  
 4   extended          181691 non-null  int64  
 5   country           181691 non-null  int64  
 6   country_txt       181691 non-null  object 
 7   region            181691 non-null  int64  
 8   region_txt        181691 non-null  object 
 9   provstate         181270 non-null  object 
 10  city              181257 non-null  object 
 11  latitude          177135 non-null  float64
 12  longitude         177134 non-null  float64
 13  specificity       181685 non-null  float64
 14  vicinity          181691 non-null  int64  
 15  summary           115562 non-null  object 
 16  crit1             181691 non-null  int64  
 17  crit2             181691 non-null  int64  
 18  crit3             181691 non-null  int64  
 19  doubtterr         181690 non-null  float64
 20  multiple          181690 non-null  float64
 21  success           181691 non-null  object 
 22  suicide           181691 non-null  int64  
 23  attacktype1       181691 non-null  int64  
 24  attacktype1_txt   181691 non-null  object 
 25  targtype1         181691 non-null  int64  
 26  targtype1_txt     181691 non-null  object 
 27  targsubtype1      171318 non-null  float64
 28  targsubtype1_txt  171318 non-null  object 
 29  corp1             139141 non-null  object 
 30  target1           181055 non-null  object 
 31  natlty1           180132 non-null  float64
 32  natlty1_txt       180132 non-null  object 
 33  gname             181691 non-null  object 
 34  guncertain1       181311 non-null  float64
 35  individual        181691 non-null  int64  
 36  nperps            110576 non-null  float64
 37  nperpcap          112202 non-null  float64
 38  claimed           115571 non-null  float64
 39  weaptype1         181691 non-null  int64  
 40  weaptype1_txt     181691 non-null  object 
 41  weapsubtype1      160923 non-null  float64
 42  weapsubtype1_txt  160923 non-null  object 
 43  weapdetail        114021 non-null  object 
 44  nkill             171378 non-null  float64
 45  nkillus           117245 non-null  float64
 46  nkillter          114733 non-null  float64
 47  nwound            165380 non-null  float64
 48  nwoundus          116989 non-null  float64
 49  nwoundte          112548 non-null  float64
 50  property          181691 non-null  int64  
 51  ishostkid         181513 non-null  float64
```

```
 52  dbsource          181691 non-null  object
 53  INT_LOG           181691 non-null  int64
 54  INT_IDEO          181691 non-null  int64
 55  INT_MISC          181691 non-null  int64
 56  INT_ANY           181691 non-null  int64
dtypes: float64(19), int64(21), object(17)
memory usage: 80.4+ MB
```

In [37]:

```
status_replace = {
    "success" : {
        1: "Success",
        0: "Filed",
    }
}
gtd_data = gtd_data.replace(status_replace)
```

In [38]:

```
predictions = pd.Series(np.ones(gtd_data.shape[0]))

false_positive_filter = (predictions == 1) & ((gtd_data['success'] == 'Succes
s'))

false_positive = len(predictions[false_positive_filter])

true_positive_filter = (predictions == 1) & ((gtd_data['success'] == 'Filed'))
true_positive = len(predictions[true_positive_filter])

predictions = pd.Series(np.zeros(gtd_data.shape[0]))
false_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Succes
s'))
false_negative = len(predictions[false_negative_filter])


true_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Filed'))
true_negative = len(predictions[true_negative_filter])


false_positive_rate = true_positive / (true_positive + false_negative)
true_positive_rate = false_positive / (false_positive + true_negative)

print (float(true_positive_rate) )
print (float(false_positive_rate))
```

```
0.8895982739926579
0.11040172600734213
```

In [39]:

```
accuracy = float(false_positive + false_negative)/float(true_positive + false_
positive+ false_negative + true_negative)
accuracy
```

Out[39]:

```
0.8895982739926579
```

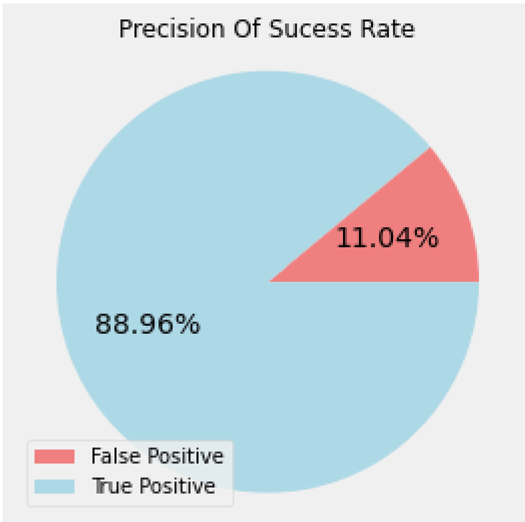
In [40]:

```
precision = float(false_positive)/float(false_positive + true_negative)
precision
```

Out[40]:

```
0.8895982739926579
```

In [41]:
```
# Data to plot
labels = 'False Positive', 'True Positive'
sizes = [1-precision, precision]
colors = ['lightcoral', 'lightblue']
# Plot
plt.figure(figsize=(4,4))
plt.pie(sizes, colors=colors, autopct='%1.2f%%', shadow=False, startangle=0)
plt.title('Precision Of Sucess Rate', fontsize=12)
plt.legend(labels, loc='lower left', fontsize=10)
plt.axis('equal')
plt.show()
```



Logistic Regression Classification

Let's try to improve the predictions with logistic regression.

In [42]:
```
gtd_data.iloc[0]
```

Out[42]:
```
eventid                          197000000001
iyear                                    1970
imonth                                      7
iday                                        2
extended                                    0
country                                    58
country_txt                Dominican Republic
region                                      2
region_txt          Central America & Caribbean
provstate                                 NaN
city                           Santo Domingo
latitude                              18.4568
longitude                            -69.9512
specificity                                 1
vicinity                                    0
summary                                   NaN
crit1                                       1
crit2                                       1
crit3                                       1
doubtterr                                   0
multiple                                    0
success                               Success
suicide                                     0
attacktype1                                 1
attacktype1_txt                 Assassination
targtype1                                  14
targtype1_txt       Private Citizens & Property
targsubtype1                               68
targsubtype1_txt               Named Civilian
corp1                                     NaN
target1                          Julio Guzman
natlty1                                    58
natlty1_txt                Dominican Republic
gname                                  MANO-D
guncertain1                                 0
individual                                  0
nperps                                    NaN
nperpcap                                  NaN
claimed                                   NaN
weaptype1                                  13
weaptype1_txt                         Unknown
weapsubtype1                              NaN
weapsubtype1_txt                          NaN
weapdetail                                NaN
nkill                                       1
nkillus                                   NaN
nkillter                                  NaN
nwound                                      0
nwoundus                                  NaN
nwoundte                                  NaN
property                                    0
ishostkid                                   0
dbsource                                 PGIS
INT_LOG                                     0
INT_IDEO                                    0
INT_MISC                                    0
```

```
INT_ANY                                     0
Name: 0, dtype: object
```

In [43]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
gtd_data.dropna(axis='columns')
cols = gtd_data.columns
#print(type(cols))
train_cols = cols.drop([ 'eventid', 'iyear', 'imonth', 'iday', 'extended', 'co
untry',
       'country_txt', 'region', 'region_txt', 'provstate', 'city', 'latitude',
       'longitude', 'specificity', 'vicinity', 'summary', 'crit1', 'crit2',
       'crit3', 'doubtterr', 'multiple',  'suicide', 'attacktype1',
       'attacktype1_txt', 'targtype1', 'targtype1_txt', 'targsubtype1',
       'targsubtype1_txt', 'corp1', 'target1', 'natlty1', 'natlty1_txt',
       'gname', 'guncertain1', 'individual', 'nperps', 'nperpcap', 'claimed',
       'weaptype1', 'weaptype1_txt', 'weapsubtype1', 'weapsubtype1_txt',
       'weapdetail',
       'nwoundte', 'property', 'ishostkid', 'dbsource', 'INT_LOG', 'INT_IDEO',
       'INT_MISC', 'INT_ANY'])
features = gtd_data[train_cols]
target = []
ans = features['success']
for val in ans=='Success':

    if val==True:
        target.append(int(True))
    else:
        target.append(int(False))

cols.drop([ 'success'])

status_replace = {
    "success" : {
        1: "Success",
        0: "Filed",
    },
}
gtd_data = gtd_data.replace(status_replace)
features = features.iloc[:, [1,2,3,4]]
features.dropna(axis='columns')
features = features.replace(0,np.NaN)
features = features.fillna(0)
lr.fit(features, target)
predictions = lr.predict(features)
```

In [44]:
```
#from sklearn.cross_validation import cross_val_predict, KFold
#lr = LogisticRegression()
#kf = KFold(features.shape[0], random_state=42)
#predictions = cross_val_predict(lr, features, target, cv=kf)
#predictions = pd.Series(predictions)
```

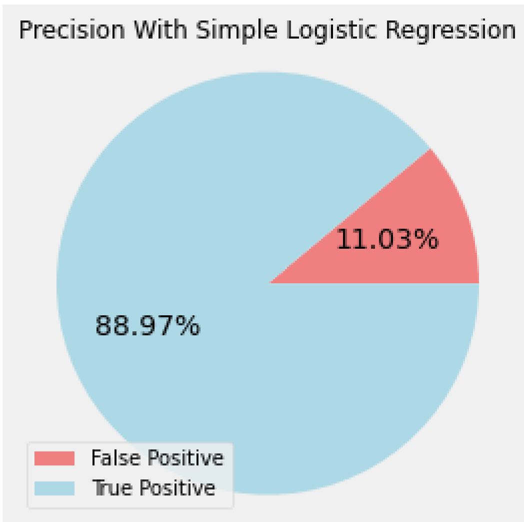
In [45]:
```
false_positive_filter =(predictions == 1) & ((gtd_data['success'] == 'Success'
))
false_positive = len(predictions[false_positive_filter])

true_positive_filter = (predictions == 1) & ((gtd_data['success'] == 'Filed'))
true_positive = len(predictions[true_positive_filter])

false_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Succes
s'))
false_negative = len(predictions[false_negative_filter])

true_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Filed'))
true_negative = len(predictions[true_negative_filter])

true_positive_rate = float(true_positive)/float((true_positive + false_negativ
e))
false_positive_rate = float(false_positive)/float((false_positive + true_negat
ive))

print (float(true_positive_rate) )
print (float(false_positive_rate))
```

```
0.9951282561145357
0.999746247872505
```

In [46]:
```
precision = float(false_positive)/float(true_positive + false_positive)
precision
```

Out[46]:
```
0.8897395787432801
```

In [47]:
```
 # Data to plot
labels = 'False Positive', 'True Positive'
sizes = [1-precision, precision]
colors = ['lightcoral', 'lightblue']
# Plot
plt.figure(figsize=(4,4))
plt.pie(sizes, colors=colors, autopct='%1.2f%%', shadow=False, startangle=0)
plt.title('Precision With Simple Logistic Regression', fontsize=12)
plt.legend(labels, loc='lower left', fontsize=10)
plt.axis('equal')
plt.show()
```



In [48]:
```
accuracy = float(false_positive + true_positive)/float(true_positive + false_p
ositive + false_negative + true_negative)
accuracy
```

Out[48]: 0.9992349648579181

Weighting Errors To Improve Performance

We will add weight to mistakes in order to penalize the model when it overfits, that way we can improve the performance of the model.

In [49]:
```
#lr = LogisticRegression(class_weight="balanced")
#kf = KFold(features.shape[0], random_state=1)
#predictions = cross_val_predict(lr, features, target, cv=kf)
#predictions = pd.Series(predictions)


false_positive_filter = (predictions == 1) & ((gtd_data['success'] == 'Succes
s'))
false_positive = len(predictions[false_positive_filter])

true_positive_filter = (predictions == 1) & ((gtd_data['success'] == 'Filed'))
true_positive = len(predictions[true_positive_filter])

false_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Succes
s'))
false_negative = len(predictions[false_negative_filter])

true_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Filed'))
true_negative = len(predictions[true_negative_filter])

true_positive_rate = float(true_positive)/float((true_positive + false_negativ
e))
false_positive_rate = float(false_positive)/float((false_positive + true_negat
ive))

print (float(true_positive_rate) )
print (float(false_positive_rate))
```

```
0.9951282561145357
0.999746247872505
```

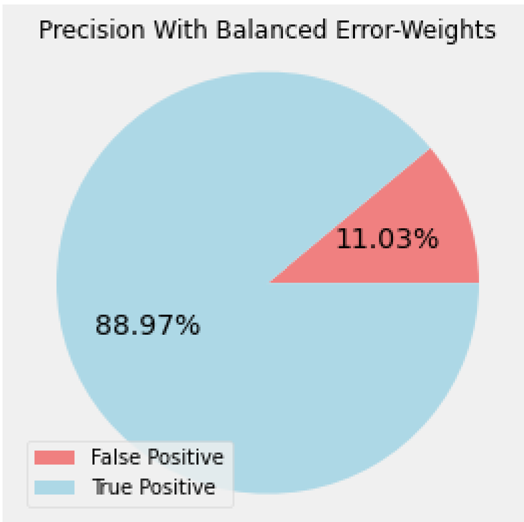
In [50]:
```
accuracy = float(false_negative + false_positive)/float((true_positive + false
_positive+ false_negative + true_negative))
accuracy
```

Out[50]: 0.8895982739926579

In [51]:
```
precision = float(false_positive)/float(true_positive + false_positive)
precision
```

Out[51]: 0.8897395787432801

In [52]:
```
# Data to plot
labels = 'False Positive', 'True Positive'
sizes = [1-precision, precision]
colors = ['lightcoral', 'lightblue']
# Plot
plt.figure(figsize=(4,4))
plt.pie(sizes, colors=colors, autopct='%1.2f%%', shadow=False, startangle=0)
plt.title('Precision With Balanced Error-Weights', fontsize=12)
plt.legend(labels, loc='lower left', fontsize=10)
plt.axis('equal')
plt.show()
```



In [53]:
```
#from sklearn.linear_model import LogisticRegression
#from sklearn.cross_validation import cross_val_predict
'''
cross_val_predict, KFold

penalty = {0: 10,
           1: 1
          }

lr = LogisticRegression(class_weight=penalty)
kf = KFold(features.shape[0], random_state=42)
predictions = cross_val_predict(lr, features, target, cv= kf)
predictions = pd.Series(predictions)
'''
```

Out[53]:
```
'\ncross_val_predict, KFold\n\npenalty = {0: 10,\n           1: 1\n
}\n\nlr = LogisticRegression(class_weight=penalty)\nkf = KFold(features.shape
[0], random_state=42)\npredictions = cross_val_predict(lr, features, target,
cv= kf)\npredictions = pd.Series(predictions)\n'
```

In [54]:
```
false_positive_filter = (predictions == 1) & ((gtd_data['success'] == 'Succes
s'))
false_positive = len(predictions[false_positive_filter])

true_positive_filter = (predictions == 1) & ((gtd_data['success'] == 'Filed'))
true_positive = len(predictions[true_positive_filter])

false_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Succes
s'))
false_negative = len(predictions[false_negative_filter])

true_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Filed'))
true_negative = len(predictions[true_negative_filter])

true_positive_rate = float(true_positive)/float((true_positive + false_negativ
e))
false_positive_rate = float(false_positive)/float((false_positive + true_negat
ive))

print( float(true_positive_rate) )
print( float(false_positive_rate))
```

```
0.9951282561145357
0.999746247872505
```

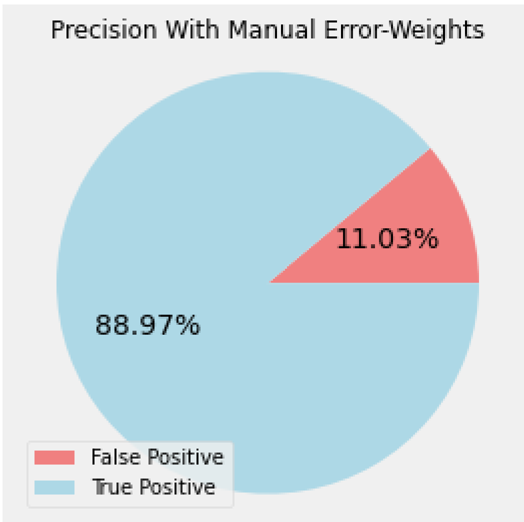
In [55]:
```
accuracy = float(false_positive + false_negative)/float(true_positive + false_
positive+ false_negative + true_negative)
accuracy
```

Out[55]:
```
0.8895982739926579
```

In [56]:
```
precision = float(false_positive)/float(true_positive + false_positive)
precision
```

Out[56]:
```
0.8897395787432801
```

In [57]:
```
# Data to plot
labels = 'False Positive', 'True Positive'
sizes = [1-precision, precision]
colors = ['lightcoral', 'lightblue']
# Plot
plt.figure(figsize=(4,4))
plt.pie(sizes, colors=colors, autopct='%1.2f%%', shadow=False, startangle=0)
plt.title('Precision With Manual Error-Weights', fontsize=12)
plt.legend(labels, loc='lower left', fontsize=10)
plt.axis('equal')
plt.show()
```



Try Random Forest

We try to fit the data with the random forest classifier of scikit-learn in order to increase the performance of our model.

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
#from sklearn.cross_validation import cross_val_predict

rf = RandomForestClassifier(class_weight="balanced", random_state=1)
#kf = KFold(features.shape[0], random_state=42)

#predictions = cross_val_predict(rf, features, target, cv=kf)
predictions = pd.Series(predictions)
```

In [59]:
```
false_positive_filter = (predictions == 1) & ((gtd_data['success'] == 'Succes
s'))
false_positive = len(predictions[false_positive_filter])

true_positive_filter = (predictions == 1) & ((gtd_data['success'] == 'Filed'))
true_positive = len(predictions[true_positive_filter])

false_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Succes
s'))
false_negative = len(predictions[false_negative_filter])

true_negative_filter = (predictions == 0) & ((gtd_data['success'] == 'Filed'))
true_negative = len(predictions[true_negative_filter])

true_positive_rate = float(true_positive)/float((true_positive + false_negativ
e))
false_positive_rate = float(false_positive)/float((false_positive + true_negat
ive))

print (float(true_positive_rate) )
print (float(false_positive_rate))
```

```
0.9951282561145357
0.999746247872505
```

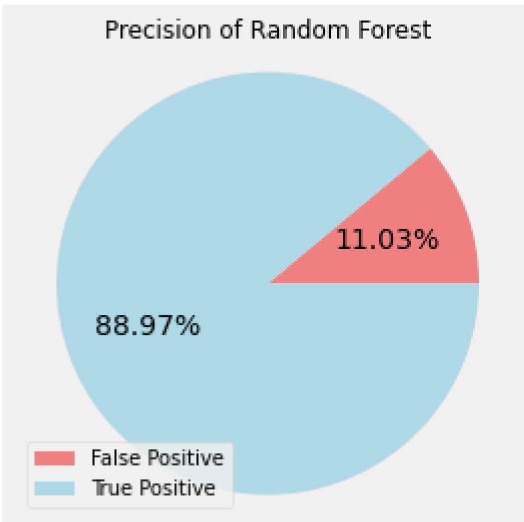
In [60]:
```
accuracy = float(false_positive + false_negative)/float(true_positive + false_
positive+ false_negative + true_negative)
accuracy
```

Out[60]:
```
0.8895982739926579
```

In [61]:
```
precision = float(false_positive)/float(true_positive + false_positive)
precision
```

Out[61]:
```
0.8897395787432801
```

In [62]:
```
# Data to plot
labels = 'False Positive', 'True Positive'
sizes = [1-precision, precision]
colors = ['lightcoral', 'lightblue']
# Plot
plt.figure(figsize=(4,4))
plt.pie(sizes, colors=colors, autopct='%1.2f%%', shadow=False, startangle=0)
plt.title('Precision of Random Forest', fontsize=12)
plt.legend(labels, loc='lower left', fontsize=10)
plt.axis('equal')
plt.show()
```



This is a similar pie as the one we started with.

Decision Trees:

Decision trees classifiers applies questions and conditions in a tree structure. This approach applies decision rules inferred from the data features to predict the value of target variable and create model accordingly. The condition for categorization is included in the root and internal nodes. Inputs are entered at the top and tree is traversed down, following the branches. Once the input node reaches the terminal node, a class is assigned. The advantage of decision trees is that they can be easily visualized and they can easily handle continuous and discrete data. When the training set is small in comparison with the number of classes, it also leads to higher classification error rate, hence causing overfitting.

In [63]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
gtd_data = pd.read_csv('gtd_data.csv')
gtd_data.dropna(axis='columns')
cols = gtd_data.columns
#print(type(cols))
train_cols = cols.drop([ 'eventid', 'iyear', 'imonth', 'iday', 'extended', 'co
untry',
       'country_txt', 'region', 'region_txt', 'provstate', 'city', 'latitude',
       'longitude', 'specificity', 'vicinity', 'summary', 'crit1', 'crit2',
       'crit3', 'doubtterr', 'multiple',  'suicide', 'attacktype1',
       'attacktype1_txt', 'targtype1', 'targtype1_txt', 'targsubtype1',
       'targsubtype1_txt', 'corp1', 'target1', 'natlty1', 'natlty1_txt',
       'gname', 'guncertain1', 'individual', 'nperps', 'nperpcap', 'claimed',
       'weaptype1', 'weaptype1_txt', 'weapsubtype1', 'weapsubtype1_txt',
       'weapdetail',
       'nwoundte', 'property', 'ishostkid', 'dbsource', 'INT_LOG', 'INT_IDEO',
       'INT_MISC', 'INT_ANY'])
features = gtd_data[train_cols]
target = []
ans = features['success']
for val in ans==1:

    if val==True:
        target.append(int(True))
    else:
        target.append(int(False))




#gtd_data = gtd_data.replace(status_replace)
features = features.iloc[:, [1,2,3,4]]
features.dropna(axis='columns')
features = features.replace(0,np.NaN)
features = features.fillna(0)

# Fitting Decision Tree Regression to the dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
X_train, X_test, y_train, y_test = train_test_split(features, target, test_siz
e = 0.3, random_state = 100)

#print(len(X_train)," = ",len(y_train))
regressor.fit(X_train, y_train)

# Predicting a new result
y_pred = regressor.predict(X_test)
#print("Confusion Matrix: ",confusion_matrix(y_pred.round(),y_test))
#print(type(y_pred.round()),type(y_test))
accuracy = accuracy_score( y_pred.round(),y_test)
print("Decision Tree Accuracy ",accuracy)
```

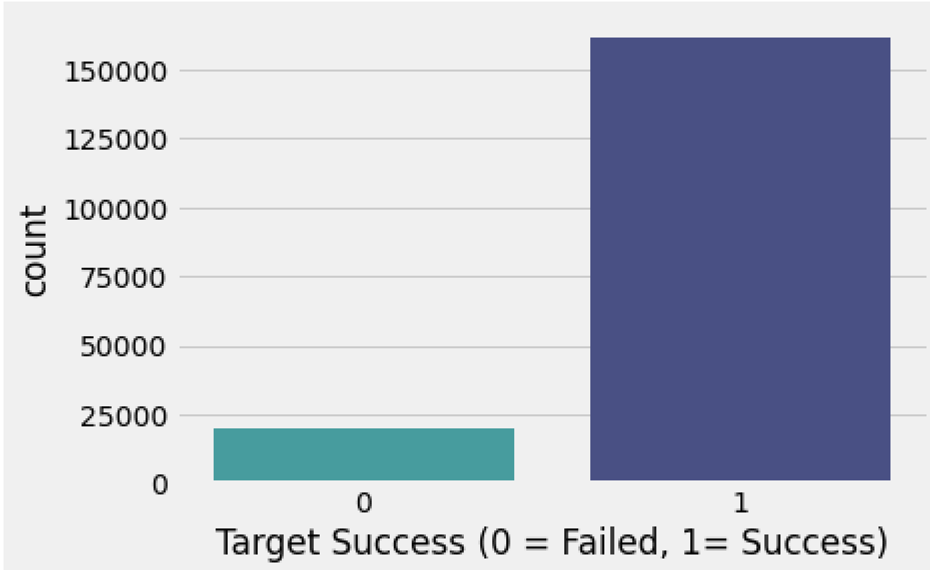
```
Decision Tree Accuracy  0.8895208042856094
```

K- Nearest Neighbours

k-NN is another algorithm commonly used for supervised classification problems. First introduced in 1951, the algorithm aims to identify homogeneous subgroups such that observations in the same group (clusters) are more similar to each other than others. Each data points' k-closest neighbors are found by calculating Euclidean or Hamming distance and grouped into clusters. The k-closest data points are then analyzed to determine which class label is the most common among the set. The most common class is then classified to the data point being tested. For k-NN classification, an input is classified by a majority vote of its neighbors. That is, the algorithm obtains the classification of its k neighbors and outputs the class that represents a majority of the k neighbors.

In [64]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics

data = pd.read_csv('gtd_data.csv')
data.head()
data.success.value_counts()
sns.countplot(x="success", data=data, palette="bwr")
#plt.show()

sns.countplot(x='success', data=data, palette="mako_r")
plt.xlabel("Target Success (0 = Failed, 1= Success)")
#plt.show()

'''
plt.scatter(x=data.success[data.success==1], y=data.success[(data.success==
1)], c="green")
plt.scatter(x=data.success[data.success==0], y=data.success[(data.success==
0)], c = 'black')
plt.legend(["Attack", "Not Attack"])
plt.xlabel("Success")
plt.ylabel("Maximum Heart Rate")
plt.show()'''



X_train, X_test, y_train, y_test =  train_test_split(features, target,test_siz
e = 0.25, random_state= 0)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2
)
classifier = classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
#check accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print('KNN Accuracy: ',accuracy)
```

```
KNN Accuracy:  0.8864231776853136
```



Linear Discriminant Analysis

LDA is also based on Bayes' Theorem. But instead of directly calculating posterior probability, it estimates multivariate distribution of its distribution. If we see its mathematical aspect, the algorithm does training by first setting the linear combination of predictors (features) that is helpful in separating different classes. The predicted class is classified by detecting the training samples which falls into linear decision boundaries. The advantage of LDA is it always produces an explicit solution and is feasible due to its low-dimensionality, but suffers from the assumption that linear separability is achievable in all classifications.

In [65]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
lda = LinearDiscriminantAnalysis()
regressor =  LinearDiscriminantAnalysis()
X_train, X_test, y_train, y_test = train_test_split(features, target, test_siz
e = 0.3, random_state = 100)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
#print("Confusion Matrix: ",confusion_matrix(y_pred.round(),y_test))
accuracy = accuracy_score( y_pred.round(),y_test)
print("Linear Discriminant Analysis Accuracy ",accuracy)
```

```
Linear Discriminant Analysis Accuracy  0.8907132897923241
```

Gaussian Naive Bayes

Gaussian Naive Bayes: Naive Bayes classifier has been considered as one of the simplest supervised approaches. In this, Bayes theorem provides a way to calculate probability of hypothesis (given prior information), hence the presence of one feature does not affect the presence of other feature. The advantage of NB is it can be easily trained with small and large datasets and the execution time is relatively fast

In [66]:
```
from sklearn.naive_bayes import GaussianNB
regressor = GaussianNB()


X_train, X_test, y_train, y_test = train_test_split(features, target, test_siz
e = 0.3, random_state = 5)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
#print("Confusion Matrix: ",confusion_matrix(y_pred.round(),y_test))
accuracy = accuracy_score( y_pred.round(),y_test)
print("Gaussian Naive Bayes Accuracy ",accuracy)
```

```
Gaussian Naive Bayes Accuracy  0.187568797240772
```

Support Vector Machines

Support Vector Machines: In machine learning, they basically comes under the category of supervised learning which analyze data used for classification and regression analysis. SVM model is a representation of points in space, mapped properly so that the categories get divided by a wide gap. If new examples are mapped, then they fall accordingly into the right side of the gap.

In [67]:
```
from sklearn import svm
regressor = svm.SVC()
X_train, X_test, y_train, y_test = train_test_split(features, target, test_siz
e = 0.3, random_state = 5)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
#print("Confusion Matrix: ",confusion_matrix(y_pred.round(),y_test))
accuracy = accuracy_score( y_pred.round(),y_test)
print("Support Vector Machine Accuracy ",accuracy)
```

```
Support Vector Machine Accuracy  0.8899060688339326
```

CONCLUSION

After training our models on the variables month, Traget_type, attack_type to predict the region of attack and country of attack it is estimated that Logistic regression, LDA, Naïve Bayes and SVM gives higher accuracy of 82 % in both the cases on predicting Region and country of terrorist attack. The results of the presented work can be used for enhancing defense against terrorist attacks in coming times.