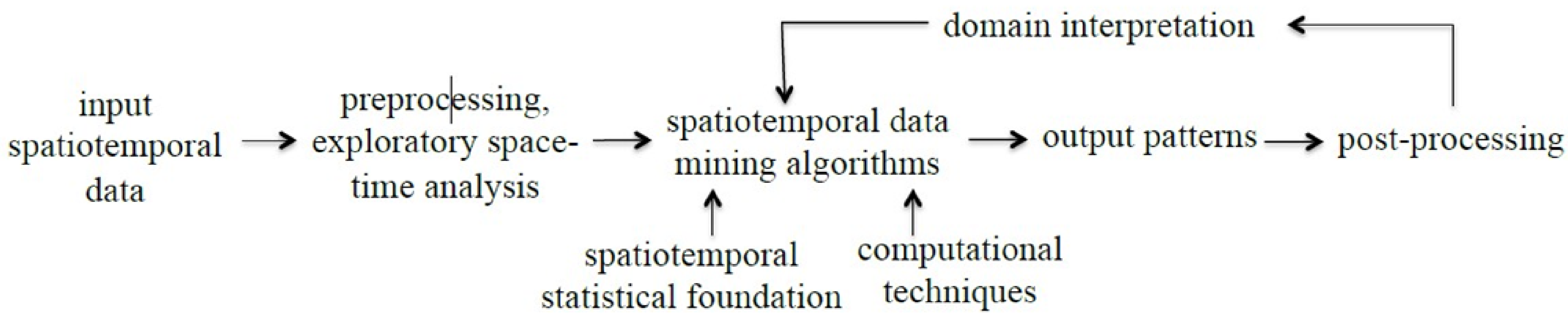
- Using the existing data to make the predictions of future armed conflict in Kenya.
- This is done by using the SpatioTemporal data mining techniques.



In [4]:

```
#Librarie for manupulating, storing data and performing mathematical calculation
import pandas as pd
import numpy as np
#importing visualization libraries
import matplotlib.pyplot as plt
import plotly.express as px
from plotly.subplots import make_subplots
%matplotlib inline
import seaborn as sns
#prediction libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

In [5]:

```
df=pd.read_csv("G:\Project\Final_YearProject👌❤\DATA\The_Armed_Conflict_Location_&_Event_Data_Project.csv")
```

In [6]:

```
df["event_date"]=pd.to_datetime(df["event_date"])
df["month"]=df["event_date"].apply(lambda time:time.month)
df["day"]=df["event_date"].apply(lambda time:time.day)
df['DayOfweek']=df["event_date"].dt.day_name()
```

In [7]:

```
groupby_df=pd.DataFrame(df.groupby(['event_date','year','Constituency','Ward','event_type','latitude','longitude','County
groupby_df.reset_index(inplace=True)
```

In [10]:
```
mciPerYear = df_ct.loc['All']
df_ct = df_ct.iloc[:-1,:]
mciPerYear = mciPerYear[:-1]
mciPerYear.tail()
```

Out[10]:
```
year
2018     524
2019     340
2020     446
2021     397
2022    1085
Name: All, dtype: int64
```

In [11]:
```
df_annual = pd.concat([pd.Series(mciPerYear.index, name='year'),
                       pd.Series(mciPerYear.values, name='event_type')], axis=1).reset_index()
df_annual = df_annual.drop(columns=['index'])
df_annual.sample(4)
```

Out[11]:

	year	event_type
15	2012	374
11	2008	416
18	2015	321
23	2020	446

Machine learning Model

- this is the subset of Artificial Interligence use to train machine to learn and make the changes.
- the type of machine learning use here is supervised machine learning. This is becouse there is Historical and label data that is used to train machine.

In [12]:
```
#Linear regression function
def LinearPredict(x,y,years):
    #reshape data
    x = x.reshape((-1, 1))
    y = y.reshape((-1, 1))
    #build model and train
    model = LinearRegression()
```

```
    model.fit(x, y)
    #evaluate error
    r_sq = model.score(x, y)
    #make predictions
    y_pred = model.predict(years)

    print('intercept:', model.intercept_)
    print('slope:', model.coef_)
    print('coefficient of determination:', r_sq)
    print('Prediction of Armed Conflict Events in Kenya in', years, 'will be', np.round(y_pred,0))

    return model.coef_, model.intercept_, y_pred
```

In [13]:

```
#Years to predict
yearsToPredict = np.array([[2023],[2024],[2025]])
```

In [14]:

```
#get MCI data
x = df_annual['year'].values
y = df_annual['event_type'].values
m,b, pred = LinearPredict(x,y,yearsToPredict)
#print("data", m,b,pred)
```

```
intercept: [-43352.86290598]
slope: [[21.7408547]]
coefficient of determination: 0.46089049870064336
Prediction of Armed Conflict Events in Kenya in [[2023]
 [2024]
 [2025]] will be [[629.]
 [651.]
 [672.]]
```

In [15]:

```
# Reshape Data
x = (df_annual['year'].values).reshape(-1, 1)
y = df_annual['event_type'].values

#build and train model
poly_reg = PolynomialFeatures(degree=2)
x_poly = poly_reg.fit_transform(x)
pol_reg = LinearRegression()
pol_reg.fit(x_poly, y)

#make prediction
```

```
pred = pol_reg.predict(poly_reg.fit_transform(yearsToPredict))
pred
```

Out[15]:
```
array([755.94384617, 805.91974362, 857.91242982])
```

In [16]:
```
#build model
poly_reg = PolynomialFeatures(degree=3)
x_poly = poly_reg.fit_transform(x)
pol_reg = LinearRegression()
#train
pol_reg.fit(x_poly, y)
#predict
pred3 = pol_reg.predict(poly_reg.fit_transform(yearsToPredict))
pred3
```

Out[16]:
```
array([784.38667449, 847.00279549, 913.89282027])
```

In [17]:
```
from sklearn import svm
```

In [18]:
```
x = (df_annual['year'].values).reshape((-1, 1))
y = (df_annual['event_type'].values)
yearsToPredict
```

Out[18]:
```
array([[2023],
       [2024],
       [2025]])
```

In [19]:
```
clf = svm.SVR(gamma='auto')
clf.fit(x,y)


resSVR = clf.predict(yearsToPredict)

resSvc_df =pd.DataFrame({'Predict': resSVR[:]})
resSvc_df
```

Out[19]:

	Predict
0	246.886319

	Predict
1	246.518439
2	246.500124

Armed Conflict prediction model

I will build a model that will predict the total number of event types in Kenya based on location of crime.

The following alogrithms will be tested

- KNeighborsClassifier
- DecisionTreeClassifier/ RandomForest &
- LogisticRegression

In [20]:

```
x = groupby_df[[ 'latitude','longitude']].values
print(type(x))
x
```

```
<class 'numpy.ndarray'>
```

Out[20]:

```
array([[ 2.1667, 36.5167],
       [-0.3072, 36.0723],
       [-4.05  , 39.6667],
       ...,
       [-4.0547, 39.6636],
       [ 0.2239, 34.808 ],
       [-1.1216, 37.1518]])
```

In [21]:

```
y = groupby_df[['location']].values.flatten()
print(type(y))
y
```

```
<class 'numpy.ndarray'>
```

Out[21]:

```
array(['Suguta Valley', 'Nakuru', 'Mombasa', ..., 'Mombasa', 'Shinyalu',
       'Komo'], dtype=object)
```

In [22]:

```
#Split data into training and testing datasets
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)
```

In [23]:

```
#Evaluation function
def Evaluate(predicted, y_test):
    true_pred = np.sum(predicted == y_test)
    total_pred = predicted.shape[0]
    print('True predictions', true_pred, 'out of', total_pred)
    print('Percent of correct predictions', round(100*true_pred/total_pred,2), '%')
```

KNeighborsClassifier

- use to predict categorical values

In [25]:

```
from sklearn.neighbors import KNeighborsClassifier
```

Defining and Training the model

In [27]:

```
#Defining Model
KNClassifier = KNeighborsClassifier(n_neighbors =15) #  n_neighbors : int, optional (default = 5)
#Training the model
KNClassifier = KNClassifier.fit(x_train, y_train)
```

Predicting and testing the model

In [30]:

```
#Predicting
resKN = KNClassifier.predict(x_test)
#Evaluate
Evaluate(resKN,y_test)
#convert resKN to dataframe
KNClassifier_df =pd.DataFrame({'KNClassifier': resKN[:]})
KNClassifier_df.head()
```

```
True predictions 1235 out of 1744
Percent of correct predictions 70.81 %
```

```
C:\Users\USER\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:211: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy
1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic
is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this
```

```
warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Out[30]:

	KNClassifier
0	Kabete
1	Tigania East
2	Kom
3	Kakamega
4	Mau Narok

In [24]:

```
from sklearn import metrics
#metrics.mean_absolute_error(y_test,y_train)
```

I have tested different values for n_neighbors, if n_neighbors =5 the error is higher than when n_neighbors =15. Further increase of n_neighbors
does not produce better results.

Using decision tree

- this is a method that oparates by constracting multiple Decision Trees during phase.
- the decision of the majority of the trees is chosen by the Random Forest as the final decision.

In [31]:

```
from sklearn.tree import DecisionTreeClassifier
```

Defining and Training the model

In [33]:

```
#defining model
DTClassifier = DecisionTreeClassifier(criterion='entropy',max_depth=20,random_state=1)
#Training model
DTClassifier = DTClassifier.fit(x_train,y_train)
```

Predicting and testing the model

In [34]:
```
#pr
resDtc = DTClassifier.predict(x_test)
Evaluate(resDtc,y_test)
DTClassifier_df =pd.DataFrame({'DTClassifier': resDtc[:]})
DTClassifier_df.head()
```

```
True predictions 1593 out of 1744
Percent of correct predictions 91.34 %
```

Out[34]:

	DTClassifier
0	Wangige
1	Tigania East
2	Kom
3	Kakamega
4	Mau Narok

I have tested different max_depth, found that increasing max_depth pass 20 does not produce better results.

Logistic Regression

In [35]:
```
from sklearn.linear_model import LogisticRegression
```

In [36]:
```
lr = LogisticRegression(random_state=1)
lr = lr.fit(x_train, y_train)
resLr = lr.predict(x_test)
Evaluate(resLr,y_test)

lr_df =pd.DataFrame({'lr': resLr[:]})
lr_df.head()
```

```
True predictions 294 out of 1744
Percent of correct predictions 16.86 %
C:\Users\USER\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to conv
erge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

```
  https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
```

Out[36]:

	lr
0	Nairobi
1	Nairobi
2	Kisumu
3	Nairobi
4	Nairobi

Voting for most frequent prediction

I have 3 predictions. To improve prediction results I will create voting algorithm.

If 2 out of 3 predictions are the same, I will choose that prediction, otherwise I will select 1st column by default The logic is this

If (a==b) or (a==c) then select a

Ifelse (b==c) then select b

Else select a by default

Above can be transformed to the following statement

If (b==c) then select b

Else select a by default

In [37]:

```
df_predict = KNClassifier_df.join(DTClassifier_df)
df_predict = df_predict.join(lr_df)
df_predict.head()
```

Out[37]:

	KNClassifier	DTClassifier	lr
0	Kabete	Wangige	Nairobi
1	Tigania East	Tigania East	Nairobi

	KNClassifier	DTClassifier	lr
2	Kom	Kom	Kisumu
3	Kakamega	Kakamega	Nairobi
4	Mau Narok	Mau Narok	Nairobi

In [38]:

```
df_predict_votes = df_predict.copy()
df_predict_votes.head()
```

Out[38]:

	KNClassifier	DTClassifier	lr
0	Kabete	Wangige	Nairobi
1	Tigania East	Tigania East	Nairobi
2	Kom	Kom	Kisumu
3	Kakamega	Kakamega	Nairobi
4	Mau Narok	Mau Narok	Nairobi

In [39]:

```
df_predict_votes['vote'] = 'none'
df_predict_votes.head()
```

Out[39]:

	KNClassifier	DTClassifier	lr	vote
0	Kabete	Wangige	Nairobi	none
1	Tigania East	Tigania East	Nairobi	none
2	Kom	Kom	Kisumu	none
3	Kakamega	Kakamega	Nairobi	none
4	Mau Narok	Mau Narok	Nairobi	none

Here is the execution of the function itself/ voting chamber

In [40]:

```
for items, vals in df_predict_votes.iterrows():
    if(vals['DTClassifier'] == vals['KNClassifier']):
```

```
        vals['vote'] = vals['KNClassifier']
    else:
        vals['vote'] = vals['DTClassifier']

    #print(items, vals['knc'],vals['dtc'],vals['lr'], vals['vote'])
```

In [41]:

```
y_voted =df_predict_votes[['vote']].values.flatten()
print(type(y_voted))
y_voted
```

```
<class 'numpy.ndarray'>
```

Out[41]:

```
array(['Wangige', 'Tigania East', 'Kom', ..., 'Nairobi', 'Nakuru',
       'Nyamira'], dtype=object)
```

In [42]:

```
Evaluate(y_voted,y_test)
```

```
True predictions 1593 out of 1744
Percent of correct predictions 91.34 %
```

calling the previous dataset with the requared column so that the output data will be complete

In [43]:

```
county=groupby_df[['County','Constituency','Ward','location','latitude','longitude']]
```

Counting the total number of predicted Armed conflict events in each location.

In [113…

```
#total number of events in predicted location
count=DTClassifier_df['DTClassifier'].value_counts()
count.head()
```

Out[113…

```
Nairobi    165
Mombasa     49
Mandera     36
Garissa     32
Nakuru      30
Name: DTClassifier, dtype: int64
```

In [114…

```
count=count.rename_axis('location')#giving name to a first column
count.head(2)
```

Out[114…

```
location
Nairobi    165
Mombasa     49
Name: DTClassifier, dtype: int64
```

In [115…

```
#giving name to the second column and make the dataframe
count=count.reset_index(name='values')
count.sort_values('location')
```

Out[115…

	location	values
472	Ahero	1
147	Aiyam	3
80	Alale	4
482	Aldai	1
121	Amagoro	3
...	...	...
126	Westlands	3
76	Witu	4
225	Wote	2
241	Wundanyi	2
555	Yala	1

556 rows × 2 columns

In [120…

```
#Saving in excel
excel_file=pd.ExcelWriter("Predictions.xlsx")
count.to_excel(excel_file)
excel_file.save()
```

In [104…

```
#cheking/testing some location if it exist in a predicted dataframe
count.loc[count['location']=='Kuresoi']
```

Out[104…

	location	values
123	Kuresoi	3

Combination of the predicted values with historical data so it can be easily understood. Here, I will be merging the predicted location with the corresponding County, constituency, ward, latitude and longitude.

In [105…

```
#checking unique values in a predicted results
Unique_values=count['location'].unique()
Unique_values.shape
```

Out[105… (556,)

In [106…

```
# combining the unique values of a predicted results(location) with the corresponding County, latitude and longitude
isinDTC=county[county['location'].isin(Unique_values)]
isinDTC.tail()
```

Out[106…

	County	Constituency	Ward	location	latitude	longitude
8711	Muranga	Kandara	Ithiru	Kandara	-0.9000	37.0000
8712	Kisumu	Kisumu Central	Railways	Kisumu	-0.1000	34.7500
8715	Migori	Suna West	Ragana-oruba	Migori	-1.0667	34.4667
8716	Machakos	Machakos	Township	Machakos	-1.5167	37.2667
8717	Mombasa	Mvita	Majengo	Mombasa	-4.0547	39.6636

In [107…

```
#counting the total number of duplicate rows
isinDTC.duplicated().sum()
```

Out[107… 6573

In [108…

```
#Total number of duplicates rows in the column of location
isinDTC['location'].duplicated().sum()
```

Out[108…

```
6804
```

In [109…

```
#droping the duplicate rows in the column of location
isinDTC['location'].drop_duplicates(inplace=False).shape
```

Out[109…

```
(556,)
```

In [117…

```
results=isinDTC.drop_duplicates(inplace= False)
results.sort_values('location')
```

Out[117…

	County	Constituency	Ward	location	latitude	longitude
2027	Kisumu	Nyando	Ahero	Ahero	-0.1833	34.9166
2286	Laikipia	Laikipia East	Sosian	Aiyam	0.3667	36.5500
7634	West Pokot	Pokot North	Alale	Alale	2.3333	35.0176
1755	West Pokot	Kacheliba	Kiwawa	Alale	2.3333	35.0176
2162	Siaya	Alego Usonga	Usonga	Aldai	0.0833	34.0667
...	...	...	...	...	...	...
7247	Nairobi	Westlands	Parklands	Westlands	-1.2682	36.8091
3131	Lamu	Lamu West	Witu	Witu	-2.3889	40.4382
4268	Makueni	Makueni	Wote	Wote	-1.7808	37.6288
1983	Taita Taveta	Wundanyi	Wundanyi/Mbale	Wundanyi	-3.4000	38.3667
1588	Siaya	Gem	Yala Township	Yala	0.0991	34.5376

787 rows × 6 columns

In [111…

```
787-556
```

Out[111…

```
231
```

>There are 231 repeated rows still, in this case the dataset can be exported to excel so that this reduntancies can be removed completely

- its the saved in an exel file format for further preprocessing

In [118…

```
excel_file=pd.ExcelWriter("Final_results.xlsx")
results.to_excel(excel_file)
excel_file.save()
```

Importing the complete processed dataset results from excel to jupyter

In [123…

```
pred_locations=pd.read_csv('G:\PROJECT\Final_YearProject👌❤️\DATA\Complete.csv')
pred_locations.head()
```

Out[123…

	County	Constituency	Ward	location	latitude	longitude	Total_Pred
0	Kisumu	Nyando	Ahero	Ahero	-0.1833	34.9166	1
1	Laikipia	Laikipia East	Sosian	Aiyam	0.3667	36.5500	3
2	West Pokot	Kacheliba	Kiwawa	Alale	2.3333	35.0176	4
3	Siaya	Alego Usonga	Usonga	Aldai	0.0833	34.0667	1
4	Busia	Teso North	Malaba Central	Amagoro	0.6333	34.3333	3

In [130…

```
pred_locations.loc[pred_locations['location']=='Eldoret']#count.loc[count['location']=='Kuresoi']
```

Out[130…

	County	Constituency	Ward	location	latitude	longitude	Total_Pred
86	Uasin Gishu	Ainabkoi	Kapsoya	Eldoret	0.5167	35.2833	20

In [ ]:

```

```