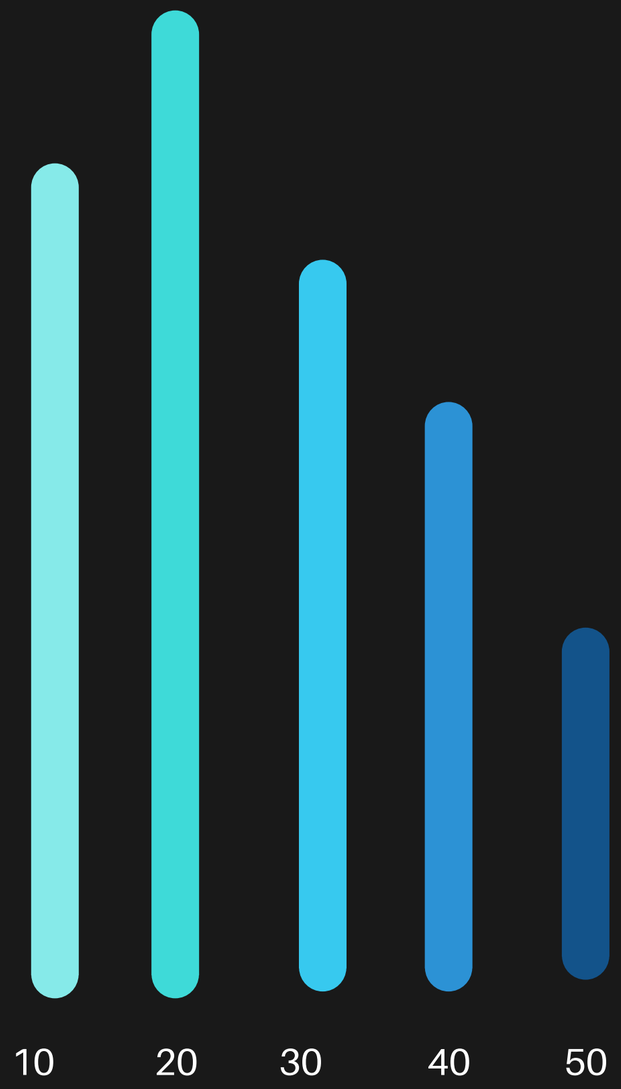


TERROR ATTACKS AROUND THE WORLD

Lahav Harary, Shelly Jurbinsky

Research questions

After considering a few topics we came across the question: Can we predict casualties in terror attacks?

We decided that it's not enough to just check casualties so we added another question:

1. Can we predict if there will be injured in terror attack?
2. Can we predict if there will be fatalities in terror attack?

Crawling

In order to retrieve the data we decided to use crawling method.

We went into the following website:

https://www.start.umd.edu/gtd/

we saw that only 20 results appear in each page and that a little button on the bottom of the screen says "20".

we changed the results to 100 and it appears that parameters in the URL are changing.

We decided to take our chances and we tried to change the count parameter in the URL to a greater number (2000).

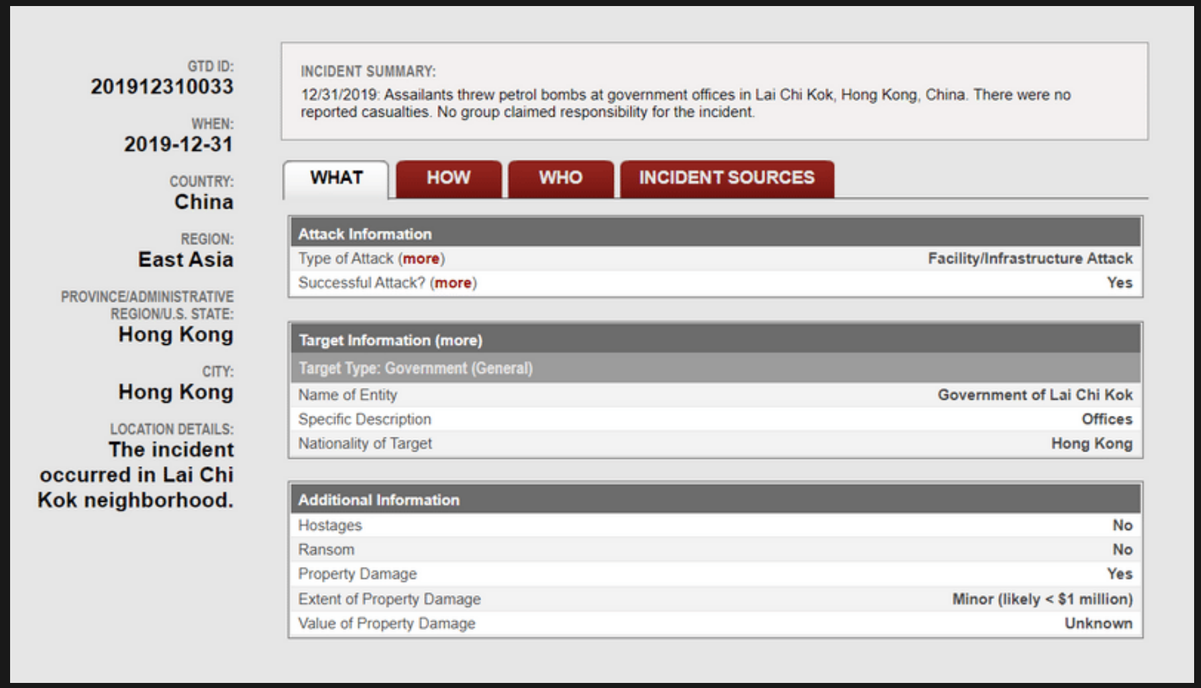
that made our table huge and allowed us to crawl over a lot less pages in total.

GTD ID	DATE	COUNTRY	CITY	PERPETRATOR GROUP	FATALITIES	INJURED	TARGET TYPE
201912310033	2019-12-31	China	Hong Kong	Unknown	0	0	Government (General)
201912310032	2019-12-31	India	Bagiot Dora	Unknown	0	1	Private Citizens & Property
201912310031	2019-12-31	Sudan	El Geneina	Unknown	2	0	Government (General),Police
201912310030	2019-12-31	Sudan	El Geneina	Unknown	2	1	Police
201912310028	2019-12-31	Iraq	Baghdad	Unknown	0	0	Private Citizens & Property
201912310027	2019-12-30	Iraq	Nada	Islamic State of Iraq and the Levant (ISIL)	1	0	Military
201912310026	2019-12-31	Cameroon	Njap	Separatists	1	0	Private Citizens & Property
201912310025	2019-12-31	Myanmar	Unknown	Arakan Army (AA)	0	0	Government (General)
201912310024	2019-12-31	Philippines	Inug-ug	Unknown	0	0	Private Citizens & Property
201912310023	2019-12-31	Russia	Magas	Caucasus Province of the Islamic State	2	4	Police
201912310022	2019-12-31	Philippines	Kinamaybay	New People's Army (NPA)	1	0	Private Citizens & Property
201912310021	2019-12-31	Philippines	Polomolok	Unknown	1	0	Government (General)
201912310019	2019-12-31	Syria	Tabqah	Islamic State of Iraq and the Levant (ISIL)	3	0	Private Citizens & Property
201912310018	2019-12-31	Yemen	Ataq district	Al-Islah Party	Unknown	Unknown	Government (General)
201912310017	2019-12-31	India	Munkeri	Tritiya Prastuti Committee (TPC)	0	1	Private Citizens & Property
201912310016	2019-12-31	India	Malhan	Tritiya Prastuti Committee (TPC)	0	0	Business
201912310015	2019-12-31	Afghanistan	Qala-e-Rig	Taliban	1	2	Government (General)
201912310014	2019-12-31	Afghanistan	Gulran district	Unknown	1	1	Private Citizens & Property
201912310013	2019-12-31	Afghanistan	Konjak	Taliban (suspected)	1	0	Private Citizens & Property
201912310012	2019-12-31	Afghanistan	Gurziwan district	Taliban	4	5	Police

Multi-Crawling

The first crawling gave us a lot of general information which is needed for training a model.
But, after playing with the website a little we saw that for every one of the rows inside our data frame we can retrieve a lot of additional relevant data such as: Type of weapon, If the terror attack was successful or not, was the attack type suicide attack? and a lot more.

In order to achieve the additional data we used MULTI-CRAWL method for every one of our GTD_ID's inside the original table.
In the additional info page of the specific attack we got the relevant data and saved it into a new better CSV file.



GTD ID:
201912310033

WHEN:
2019-12-31

COUNTRY:
China

REGION:
East Asia

PROVINCE/ADMINISTRATIVE REGION/U.S. STATE:
Hong Kong

CITY:
Hong Kong

LOCATION DETAILS:
The incident occurred in Lai Chi Kok neighborhood.

INCIDENT SUMMARY:
12/31/2019: Assailants threw petrol bombs at government offices in Lai Chi Kok, Hong Kong, China. There were no reported casualties. No group claimed responsibility for the incident.

WHAT | HOW | WHO | INCIDENT SOURCES

Attack Information	
Type of Attack (more)	Facility/Infrastructure Attack
Successful Attack? (more)	Yes

Target Information (more)	
Target Type: Government (General)	
Name of Entity	Government of Lai Chi Kok
Specific Description	Offices
Nationality of Target	Hong Kong

Additional Information	
Hostages	No
Ransom	No
Property Damage	Yes
Extent of Property Damage	Minor (likely < $1 million)
Value of Property Damage	Unknown

Data Cleaning and Visualization:

At first, we had a lot of tables from each of the crawls we did.
Each one of our crawls represents a part of the final table.
We merged all of our tables together into one.
After doing so we had plenty of data (201,183 rows X 17 columns) but some of the data was incoherent (Missing days in date, a lot on Unknown values, some duplicates and so on).
in this section we needed to get rid of some of the data and "clean" it.

Uses load_csv to merge all of the tables into one DataFrame

In [3]:
```
def unionDf():
    mainDf,firstCrawlDf,secondCrawlDf,thirdCrawlDf,fourthCrawlDf,lastCrawlDf=load_csv()
    #adding attackType
    attackTypeList=firstCrawlDf.attackTypeList.tolist()
    attackTypeList.extend(secondCrawlDf.attackTypeList.tolist())
    attackTypeList.extend(thirdCrawlDf.attackTypeList.tolist())
    attackTypeList.extend(fourthCrawlDf.attackTypeList.tolist())
    attackTypeList.extend(lastCrawlDf.attackTypeList.tolist())
    #adding successfulAttack
    successfulAttackList=firstCrawlDf.successfulAttackList.tolist()
    successfulAttackList.extend(secondCrawlDf.successfulAttackList.tolist())
    successfulAttackList.extend(thirdCrawlDf.successfulAttackList.tolist())
    successfulAttackList.extend(fourthCrawlDf.successfulAttackList.tolist())
    successfulAttackList.extend(lastCrawlDf.successfulAttackList.tolist())
    #adding hostagesList
    hostagesList=firstCrawlDf.hostagesList.tolist()
    hostagesList.extend(secondCrawlDf.hostagesList.tolist())
    hostagesList.extend(thirdCrawlDf.hostagesList.tolist())
```

	GTD_ID	DATE	COUNTRY	CITY	PERPETRATOR_GROUP	FATALITIES	INJURED	TARGET_TYPE	ATTACK_TYPE	SUCCESSFUL_ATTACK
0	201912310033	2019-12-31	China	Hong Kong	Unknown	0	0	Government (General)	Facility/Infrastructure Attack	Yes
1	201912310032	2019-12-31	India	Bagiot Dora	Unknown	0	1	Private Citizens & Property	Bombing/Explosion	Yes
2	201912310031	2019-12-31	Sudan	El Geneina	Unknown	2	0	Government (General),Police	Armed Assault	Yes
3	201912310030	2019-12-31	Sudan	El Geneina	Unknown	2	1	Police	Unknown	Yes
4	201912310028	2019-12-31	Iraq	Baghdad	Unknown	0	0	Private Citizens & Property	Bombing/Explosion	Yes
...	...	...	...	...	...	...	...	...	...	...
201178	197001000003	1970-01-00	Japan	Fukouka	Unknown	Unknown	Unknown	Government (Diplomatic)	Facility/Infrastructure Attack	Yes
201179	197001000002	1970-01-00	Greece	Athens	Unknown	Unknown	Unknown	Government (Diplomatic)	Bombing/Explosion	Yes
201180	197001000001	1970-01-00	Philippines	Unknown	Unknown	1	0	Journalists & Media	Assassination	Yes
201181	197000000002	1970-00-0	Mexico	Mexico city	23rd of September Communist League	0	0	Government (Diplomatic)	Hostage Taking (Kidnapping)	Yes
201182	197000000001	1970-07-02	Dominican Republic	Santo Domingo	MANO-D	1	0	Private Citizens & Property	Assassination	Yes

201183 rows × 17 columns

Data Cleaning and Visualization:

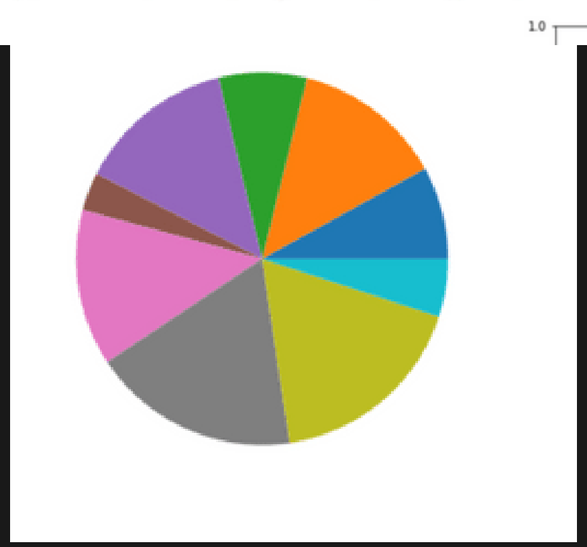
After cleaning the data it is time for us (the data scientists) to look at the data and try to understand it before processing it any further.

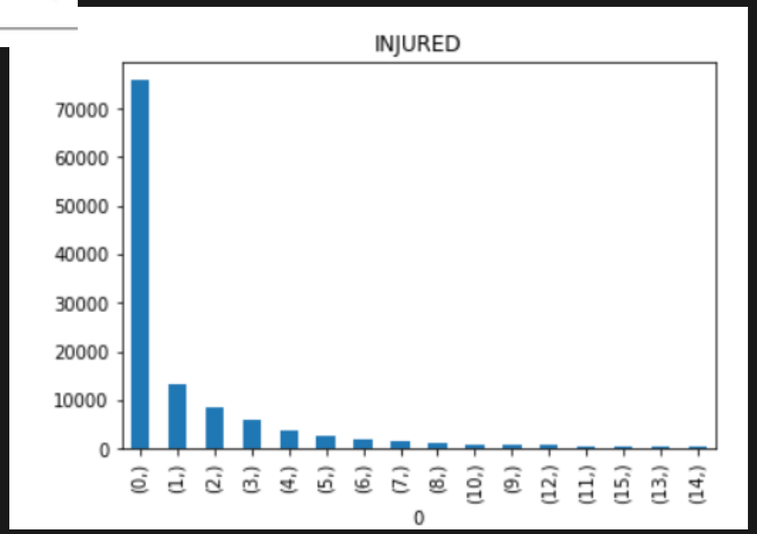
We started with understating how to treat outliers in our data, check if some of our columns are correlated:
we used a few methods to do so - one of them include heatmaps.
After that its time for us to visualize the data - we used a few bar plots and a pie chart to do so.

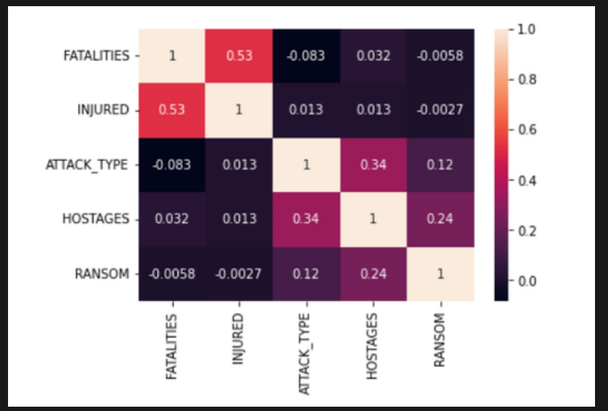
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201183 entries, 0 to 201182
Data columns (total 17 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   GTD_ID                      201183 non-null  int64
 1   DATE                        201183 non-null  object
 2   COUNTRY                     201183 non-null  object
 3   CITY                        200757 non-null  object
 4   PERPETRATOR_GROUP           201179 non-null  object
 5   FATALITIES                  201183 non-null  object
 6   INJURED                     201183 non-null  object
 7   TARGET_TYPE                 201183 non-null  object
 8   ATTACK_TYPE                 201183 non-null  object
 9   SUCCESSFUL_ATTACK           201183 non-null  object
 10  HOSTAGES                    201183 non-null  object
 11  RANSOM                      201183 non-null  object
 12  PROPERTY_DAMAGE             201183 non-null  object
 13  SUICIDE_ATTACK              201183 non-null  object
 14  PART_OF_MULTIPLE_INCIDENT   201183 non-null  object
 15  TYPE_OF_WEAPON_LIST         201183 non-null  object
 16  SUB_TYPE_OF_WEAPON          201183 non-null  object
dtypes: int64(1), object(16)
memory usage: 26.1+ MB
```

```
['Iraq', 'Pakistan', 'India', 'Afghanistan', 'Philippines', 'Colombia', 'Peru', 'Turkey', 'Thailand', 'El Salvador']
```







Machine learning

After cleaning the data and making it suitable to work with we now can start the research.
Our first question was: Can we predict if there will be FATALITIES in terror attack according to our info?
Our second question was: Can we predict if there will be INJURED in terror attack according to our info?

to do so we used 3 Machine learning algorithms:

1. Logistic Regression
2. KNN
3. NaiveBayes

Machine learning-Logistic Regression

Fatalities question:
We started by changing the FATALITIES column data to 1 if there were fatalities and 0 otherwise.
At first we tried to use standard scalar to train the model and we got some pretty good results: 0.77 !
We thought to ourselves that maybe we can do a little better if we change the scalar to: minmax scalar
After training the model with minmax we got similar results (but a little worse then before): 0.74

standard scalar usage

We want to see the results with standard scalar

```
target_column='FATALITIES'

X,y= split_df_to_x_y(df,target_column)

test_ratio, rand_state =0.3,11
X_train, X_test, y_train, y_test =split_to_train_and_test(X,y,test_ratio,rand_state)

scale_type= 'standard'
standard_scaler, X_train_standard_scaled = scale_features(X_train, scale_type)

X_test_standard_scaled = scale_test_features(X_test, standard_scaler)

classification_standard_model=train_model(X_train_standard_scaled, y_train)

df_standard_res,y_predicted = predict(classification_standard_model, X_test_standard_scaled, y_test)

evaluate = evaluate_performance(y_test,y_predicted)
```

```
evaluate
```

```
0.7753419852683269
```

minmax scalar usage

after standrad scalar we wondered if there is any way to make our prediction a little better.
Thats when we came across minmax scalar.

```
target_column='FATALITIES'
X,y= split_df_to_x_y(df,target_column)

test_ratio, rand_state =0.3,11
X_train, X_test, y_train, y_test =split_to_train_and_test(X,y,test_ratio,rand_state)

scale_type= 'minmax'
minmax_scaler, X_train_minmax_scaled = scale_features(X_train, scale_type)

X_test_minmax_scaled = scale_test_features(X_test, minmax_scaler)

classification_minmax_model = train_model(X_train_minmax_scaled, y_train)

df_minmax_res,y_predicted = predict(classification_minmax_model, X_test_minmax_scaled, y_test)

evaluate = evaluate_performance(y_test,y_predicted)
```

```
evaluate
```

```
0.7491250635637582
```

Machine learning-Logistic Regression

Injured question:
We started by changing the INJURED column data to 1 if there were injured and 0 otherwise.
At first we tried to use standard scalar to train the model and we got some pretty terrible results: 0.52
We tried to train the model with minmax scalar in order to improve the results.
After training the model with minmax we got similar results: 0.51 which means that its not very helpful.

standard scalar usage

```
target_column='INJURED'

X,y= split_df_to_x_y(df,target_column)

test_ratio, rand_state =0.3,11
X_train, X_test, y_train, y_test =split_to_train_and_test(X,y,test_ratio,rand_state)

scale_type= 'standard'
standard_scaler, X_train_standard_scaled = scale_features(X_train, scale_type)

X_test_standard_scaled = scale_test_features(X_test, standard_scaler)

classification_standard_model=train_model(X_train_standard_scaled, y_train)

df_standard_res,y_predicted = predict(classification_standard_model, X_test_standard_scaled, y_test)

evaluate = evaluate_performance(y_test,y_predicted)
```

```
evaluate
```

```
0.5211326081541144
```

```
target_column='INJURED'
X,y= split_df_to_x_y(df,target_column)

test_ratio, rand_state =0.3,11
X_train, X_test, y_train, y_test =split_to_train_and_test(X,y,test_ratio,rand_state)

scale_type= 'minmax'
minmax_scaler, X_train_minmax_scaled = scale_features(X_train, scale_type)

X_test_minmax_scaled = scale_test_features(X_test, minmax_scaler)

classification_minmax_model = train_model(X_train_minmax_scaled, y_train)

df_minmax_res,y_predicted = predict(classification_minmax_model, X_test_minmax_scaled, y_test)

evaluate = evaluate_performance(y_test,y_predicted)
```

```
evaluate
```

```
0.5147553231228987
```

Machine learning-Logistic Regression

Injured question:
We decided to try to get rid of some data in order to see if it will change something.
After looking at our Data cleaning and visualization notebook we decided that it might be OK if we got rid of the following columns:
HOSTAGES, RANSOM, SUICIDE_ATTACK
because most of their values are 0 and it appears that they don't contribute to the ML algorithm (Imbalanced Data).
After getting rid of the columns, minmax scalar model returned a result of 0.54
which means our assumption was wrong.

We understood that the Logistic Regression model isn't very robust to changes and isn't very strong ML algorithm
We decided to try KNN and NaiveBayes as well.

```
df=df.drop(columns=["HOSTAGES","RANSOM","SUICIDE_ATTACK"])
```

```
target_column='INJURED'
X,y= split_df_to_x_y(df,target_column)

test_ratio, rand_state =0.3,11
X_train, X_test, y_train, y_test =split_to_train_and_test(X,y,test_ratio,rand_state)

scale_type= 'minmax'
minmax_scaler, X_train_minmax_scaled = scale_features(X_train, scale_type)

X_test_minmax_scaled = scale_test_features(X_test, minmax_scaler)

classification_minmax_model = train_model(X_train_minmax_scaled, y_train)

df_minmax_res,y_predicted = predict(classification_minmax_model, X_test_minmax_scaled, y_test)

evaluate = evaluate_performance(y_test,y_predicted)
```

```
evaluate
```

```
0.5442284807616026
```

Machine learning-KNN

injured question:
After working with Logistic Regression we started our journey with KNN oh and what a journey it was.
First attempt:
we didn't get rid of any data and we changed the FATALITIES and INJURED column to 0's and 1's (0 means no injured/fatalities).
We used a function named: find_best_k_for_KNN which runs KNN with different values every time and gives us back the best k to run KNN with.
The first attempt came back with a result of 0.64 - not a bad first try per se.
our notes to ourselves was:
1. KNN has preformed a lot better than Logistic regression when it comes to predicting INJURED in terror attack.
(Reminder Logistic regression got only 0.54 and KNN get 0.64 that a great improvement!)
2. KNN took a little longer to run than Logistic regression due to running a function named: find_best_k_for_KNN
which takes some time to get the best result.

```
def find_best_k_for_KNN(XTrain,yTrain,XTest):
    k_s=[]
    train_accuracies=[]
    test_accuracies=[]
    for k in range(3,25,2):
        clf = KNeighborsClassifier(n_neighbors=k)
        clf.fit(XTrain, yTrain)
        y_pred_train=clf.predict(XTrain)
        y_pred=clf.predict(XTest)
        k_s.append(k)
        train_accuracies.append(metrics.accuracy_score(y_true = yTrain, y_pred = y_pred_train))
        test_accuracies.append(metrics.accuracy_score(y_true = yTest, y_pred = y_pred))

    return pd.DataFrame({"k":k_s,"train_accuracy":train_accuracies,"test_accuracy":test_accuracies})
```

```
df_best_k[df_best_k['test_accuracy']==df_best_k.test_accuracy.max()]
```

	k	train_accuracy	test_accuracy
8	19	0.684911	0.645231

```
evaluate_value = f1_score(yTest,y_pred,average='micro')
print(evaluate_value)
```

```
0.6452305803962827
```

Machine learning-KNN

injured question:

Second attempt:
What will happen if we decide to drop the columns that we talked about in the previous pages ("HOSTAGES","RANSOM","SUICIDE_ATTACK")?
As we see it a good Data Scientist needs to be able to ask some hard questions, get rid of data or change the data if it feels like he can achieve better conclusions out of it.
To do so - we decided to once again try to drop those columns and train a KNN model.
Unfortunately it didn't help at all and we got the same 0.64 result as earlier.

```
df=df.drop(columns=["HOSTAGES","RANSOM","SUICIDE_ATTACK"])
```

```
df_best_k[df_best_k['test_accuracy']==df_best_k.test_accuracy.max()]
```

	k	train_accuracy	test_accuracy
8	19	0.684757	0.645055

```
evaluate_value = f1_score(yTest,y_pred,average='micro')
print(evaluate_value)
```

```
0.6450552340873225
```

Machine learning-KNN

Fatalities question:

First attempt:
we took FATALITIES and INJURED columns once again and set them to 0's and 1's.
We trained a model based on fatalities (this time) and got the best k with the usage of our find_best_k_for_KNN function.
This time the results were 0.66 (which is pretty much ok),
but we couldn't help ourselves and we decided to try again without ("HOSTAGES","RANSOM","SUICIDE_ATTACK") columns.

```
df_best_k[df_best_k['test_accuracy']==df_best_k.test_accuracy.max()]
```

	k	train_accuracy	test_accuracy
2	7	0.74657	0.660354

```
evaluate_value = f1_score(yTest,y_pred,average='micro')
print(evaluate_value)
```

```
0.6603541995440996
```

Machine learning-KNN

Fatalities question:

Second attempt:
This time we dropped ("HOSTAGES","RANSOM","SUICIDE_ATTACK") columns.
Used find_best_k_for_KNN to find k and later on trained a knn model with the same k.
The following results were devastating 0.51 and we decided not to drop any data in our last ML section.

```
df=df.drop(columns=["HOSTAGES","RANSOM","SUICIDE_ATTACK"])
```

```
df_best_k[df_best_k['test_accuracy']==df_best_k.test_accuracy.max()]
```

	k	train_accuracy	test_accuracy
10	23	0.537241	0.510828

```
evaluate_value = f1_score(yTest,y_pred,average='micro')
print(evaluate_value)
```

```
0.5108276345782922
```

Machine learning-KNN

Our final conclusion after KNN section:

1. Without giving up any data we got better results when it comes to INJURED compared to Logistic regression

2. Without giving up any data we got worse results when it comes to FATALITIES compared to Logistic regression

3. While giving up on some data we got similar results in INJURED compared to KNN without giving up on data (1)

4. While giving up on some data we got worse results in FATALITIES compared to KNN without giving up on data (2)

Machine learning-NaiveBayes

After our Logistic Regression and KNN section we wanted a strong ML algorithm one that is robust to changes and will lead to overall better results (for each one of our questions).
A reminder:
Logistic regression gave us pretty good results when we asked about INJURED but a lousy results when we asked about FATALITIES.
KNN gave us overall the same results for INJURED and FATALITIES but they were not that high.

We were eager to get better results we couldn't stop until we got them and after reading online about some ML algorithms we decided to give NaiveBayes a try.

Machine learning-NaiveBayes

Injured question:

We tried GaussianNB, BernoulliNB and CategoricalNB

GaussianNB - gave us result of 0.65

BernoulliNB - gave us results of 0.65

CategoricalNB - gave us results of 0.69

Overall pretty consistent results (and not so bad either)

```
gnb = CategoricalNB()


# Train classifier
gnb.fit(XTrain,yTrain)

y_pred = gnb.predict(XTest)
y_pred_train = gnb.predict(XTrain)

# Print results
print('Accuracy on Train data= ', metrics.accuracy_score(y_true = yTrain, y_pred = y_pred_train))
print('Accuracy on test data= ', metrics.accuracy_score(y_true = yTest, y_pred = y_pred))
```

```
Accuracy on Train data=  0.7466134759545833
Accuracy on test data=  0.6939768542872172
```

```
gnb = GaussianNB()

# Train classifier
gnb.fit(XTrain,yTrain)

y_pred = gnb.predict(XTest)
y_pred_train = gnb.predict(XTrain)

# Print results
print('Accuracy on Train data= ', metrics.accuracy_score(y_true = yTrain, y_pred = y_pred_train))
print('Accuracy on test data= ', metrics.accuracy_score(y_true = yTest, y_pred = y_pred))
```

```
Accuracy on Train data=  0.6568431896891851
Accuracy on test data=  0.6488251797299667
```

```
from sklearn.naive_bayes import BernoulliNB
gnb = BernoulliNB()


# Train classifier
gnb.fit(XTrain,yTrain)

y_pred = gnb.predict(XTest)
y_pred_train = gnb.predict(XTrain)

# Print results
print('Accuracy on Train data= ', metrics.accuracy_score(y_true = yTrain, y_pred = y_pred_train))
print('Accuracy on test data= ', metrics.accuracy_score(y_true = yTest, y_pred = y_pred))
```

```
Accuracy on Train data=  0.6593529437552058
Accuracy on test data=  0.6563650710152551
```

Machine learning-NaiveBayes

Fatalities question:

once again we used GaussianNB, BernoulliNB and CategoricalNB

GaussianNB - gave us result of 0.76

BernoulliNB - gave us results of 0.75

CategoricalNB - gave us results of 0.77

Overall Amazing results

```
gnb = CategoricalNB()


# Train classifier
gnb.fit(XTrain,yTrain)

y_pred = gnb.predict(XTest)
y_pred_train = gnb.predict(XTrain)

# Print results
print('Accuracy on Train data= ', metrics.accuracy_score(y_true = yTrain, y_pred = y_pred_train))
print('Accuracy on test data= ', metrics.accuracy_score(y_true = yTest, y_pred = y_pred))
```

```
Accuracy on Train data=  0.7907369251676822
Accuracy on test data=  0.7735840785551464
```

```
gnb = GaussianNB()


# Train classifier
gnb.fit(XTrain,yTrain)

y_pred = gnb.predict(XTest)
y_pred_train = gnb.predict(XTrain)

# Print results
print('Accuracy on Train data= ', metrics.accuracy_score(y_true = yTrain, y_pred = y_pred_train))
print('Accuracy on test data= ', metrics.accuracy_score(y_true = yTest, y_pred = y_pred))
```

```
Accuracy on Train data=  0.7644776642847749
Accuracy on test data=  0.7664387164650184
```

```
gnb = BernoulliNB()


# Train classifier
gnb.fit(XTrain,yTrain)

y_pred = gnb.predict(XTest)
y_pred_train = gnb.predict(XTrain)

# Print results
print('Accuracy on Train data= ', metrics.accuracy_score(y_true = yTrain, y_pred = y_pred_train))
print('Accuracy on test data= ', metrics.accuracy_score(y_true = yTest, y_pred = y_pred))
```

```
Accuracy on Train data=  0.7485314102845119
Accuracy on test data=  0.7539014553743644
```

Machine learning-NaiveBayes

Our final conclusions regarding the NaiveBayes algorithm :

1. The NaiveBayes has amazing results to each one of our questions.
2. In general it feels like CategoricalNB is getting the best results (for our data).

Final Conclusions

1. Getting the data took us a lot of resources (but was fun) - it took us a lot of time to do so (almost a week including writing the code and running it with 3 different computers at the same time!)
2. Cleaning the data might be the most important part of this work - if the data isn't clean and has a lot of outliers it affects directly on the ML algorithm and makes it worse.
3. Choosing a correct ML algorithm is the key to finish the research - we spent a lot of time trying to figure out which ML algorithms are suitable for us.