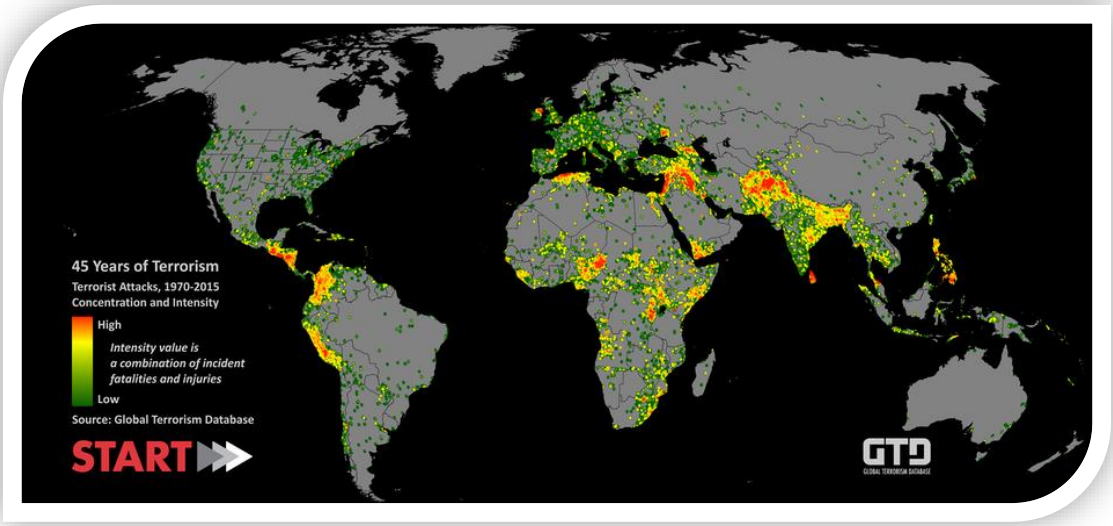
STATISTICS AND PROBABLITY TERM PROJECT

Name:	Muhammed Yahya Avar
Number:	1921221004
Date:	5.06.2021
Department:	Computer Engineering

STATISTICALLY ANALYSING GLOBAL TERROR ATTACKS PER YEAR

The Global Terrorism Database (GTD) is an open-source database including information on terrorist attacks around the world from 1970 through 2017, except for 1993. Unfortunately, terrorism is a big issue in today's world. The reason why I picked up this database is to analyze the near 50-year continuation of terror attacks worldwide. The database is maintained by researchers at the National Consortium for the Study of Terrorism and Responses to Terrorism (START), headquartered at the University of Maryland.



The column choose is "year", because statistically analyzing it can show the density of attacks per time. Each terrorist attack data on the GTD has its year and more the same year repeats, it means the more attacks happened in that year. It starts with the year 1970 and ends with 2017, with the exception of 1993, that's because the original PGIS data consisted of hard-copy index cards, which were subsequently coded electronically by START researchers. Unfortunately, **the set of cards for 1993 was lost prior** to PGIS handing the data over to START.

STATISTICAL CALCULATIONS OF DATA

Inserting the Data

```
import pandas as pd
import math

mainData = pd.read_csv("/Users/Yahya/Desktop/CE/JpyNote/Statistics Project/globalterrorismdb_0718dist.csv", engine='python')
mainData=mainData.rename(columns={'iyear': 'year' })
year=mainData.year
```

At first, I imported the pandas and math libraries for data analysis and some calculations like square root or sum. And renamed the “iyear” column to “year” and got the column data on “year” variable.

Mean

Sum of values of a data set divided by number of values.

$$\overline{X} = \frac{\sum X}{N}$$

```
In [34]: def Mean(data):
             mean=sum(data)/len(data)
             return(mean)

         Mean(year)

Out[34]: 2002.6389969783863
```

Divided the sum of years by the total number of years (length of the data).

Median

Variable separating the higher half from the lower half of a data sample, central value.

$$\mathrm{Med}(X) = \begin{cases} X[\frac{n}{2}] & \text{if n is even} \\ \frac{(X[\frac{n-1}{2}]+X[\frac{n+1}{2}])}{2} & \text{if n is odd} \end{cases}$$

X = ordered list of values in data set

n = number of values in data set

```
In [31]: def Median(data):
             sortedData=data.sort_values()
             length = len(sortedData)
             if length % 2 == 0:
                 tempFirst = sortedData[int(length/2)]
                 tempSecond = sortedData[int(length/2)-1]
                 median=(tempFirst + tempSecond)/2
                 return(median)
             else:
                 median=sortedData[int(length/2)]
                 return(median)

         Median(year)

Out[31]: 2009
```

Sorted the data, got its length and if the number of years on data are even, calculated the median by finding the average of mid-most values and returned it. If the number of years on data are odd, returned the middle year on the data.

Shape of Distribution

The shape of a distribution is described by its number of peaks and by its possession of symmetry, its tendency to skew. Distributions that are skewed have more points plotted on one side of the graph than on the other. We can decide the shape of the distribution by looking into the mean and median.

If Mean=Median, it's symmetric, If Mean>Median, it's right-skewed, If Mean<Median, it's left-skewed.

```
In [16]: def Shape(data):
             if Mean(data)==Median(data):
                 return "symmetric"
             if Mean(data)>Median(data):
                 return "right-skewed"
             if Mean(data)<Median(data):
                 return "left-skewed"

         Shape(year)

Out[16]: 'left-skewed'
```

Variance and Standard Deviation

Variance (s^2) measures variability among observations and estimates the population variance. Standard deviation (s) is a square root of a sample variance and it measures variability in the same units as X and estimates the population standard deviation.

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

```
In [33]: def Variance(data):
             length = len(data)
             mean=Mean(data)
             deviations = [pow((x - mean),2) for x in data]
             variance = sum(deviations) / length
             return variance

         Variance(year)
```

```
Out[33]: 175.8115286439676
```

```
In [34]: def Stdev(data):
             variance = Variance(data)
             stdDev = math.sqrt(variance)
             return stdDev

         Stdev(year)
```

```
Out[34]: 13.259393977251284
```

While calculating the variance, got the length and mean of data and calculated the deviations by taking the square powers of sample minus mean for all sample in data. And divided it by length of the data to find variance. For calculating the standard deviation, took the square root of variance.

Standard Error

It shows precision and reliability of estimators, and how much estimators of the same parameter can vary if they are computed from different samples. The standard error of the mean equals the standard deviation divided by the square root of the sample size.

$$SE = \frac{\sigma}{\sqrt{n}}$$

```
In [37]: def StanErr(data):
             stdDev=Stdev(data)
             length = len(data)
             stanErr=stdDev/math.sqrt(length)
             return stanErr

         StanErr(year)
```

```
Out[37]: 0.031106916785692926
```

Divided the standard deviation of data by square root of its length.

Outliers

Sample mean, variance, and standard deviation are sensitive to outliers. If an extreme observation (an outlier) erroneously appears in our data set, it can rather significantly affect the values of them. A “rule of thumb” for identifying outliers is the rule of 1.5(IQR). Measure 1.5(Q3-Q1) down from the first quartile and up from the third quartile. IQR (interquartile range) is defined as the difference between the first and the third quartiles. It measures the variability of data.

In [9]:
```
def Outlier(data):
    outliers=[]
    outCount=0
    sortedData=data.sort_values()
    Q1=sortedData[int((len(sortedData)+1)/4)]
    Q3=sortedData[int((len(sortedData)*3)/4)]
    IQR = Q3-Q1
    lower = Q1-(1.5*(IQR))
    upper = Q3+(1.5*(IQR))
    for x in sortedData:
        if x < lower or x > upper:
            outliers.append(x)
            outCount+=1
    if outCount==0:
        return "No outlier found on data."
    else:
        return "Number of outliers: ", outCount, outliers


Outlier(year)
```

Out[9]: 'No outlier found on data.'

Defined a list of outliers and a count of outliers, sorted the data, found the Q1 and Q3 by taking the length of data and further modifying it to find where the quantiles are and assigned them to their variables. Calculated the IQR and calculated the lower and upper range afterwards. Scanned the whole sorted data for finding out values outside of range and if found, increased the outlier count. Returned the number of outliers and a list of outliers at the end. If none found, returned a message of it.

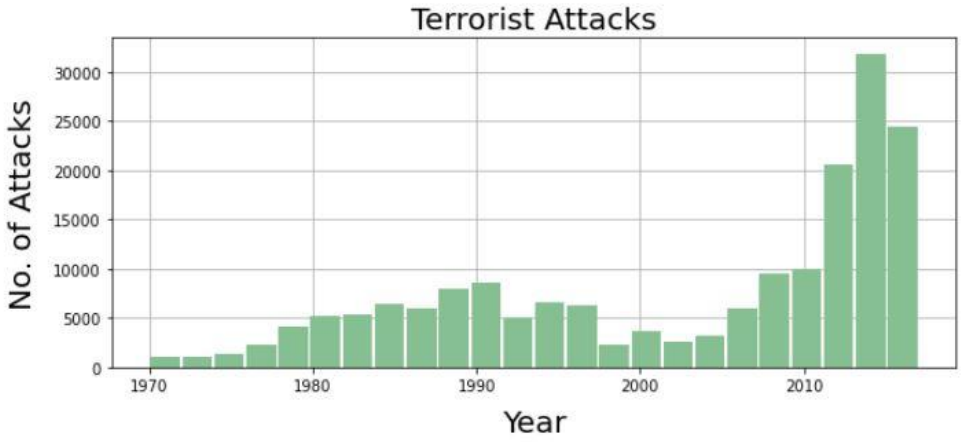
Our data has no outliers, that’s because of the closeness of values to each other with very little difference, due to them being “years”.

Histogram

A histogram shows the shape of a probability mass function or a probability density function of data, checks for homogeneity, and suggests possible outliers. To construct a histogram, we split the range of data into equal intervals, “bins,” and count how many observations fall into each bin.

```
In [9]: def drawHist(data):
            edt =data.hist(bins=24, grid=True, figsize=(10,4), color='#86bf91', zorder=4, rwidth=0.92)
            edt.set_title("Terrorist Attacks", size=20)
            edt.set_xlabel("Year", labelpad=10, size=20)
            edt.set_ylabel("No. of Attacks", labelpad=10, size=20)

        drawHist(year)
```



Using the Pandas library's “.hist” function, generated the histogram with 24 bins, and modified its size, color, width between bins etc. and labeled it.

Looking into our density histogram, we can see that it’s a mixture histogram and indeed a left-skewed one. The number attacks decreased after early 90’s up until 2003-2004 where it started to rise. That’s due to the start of Iraqi & Afghani insurgencies that started after the US invasion. We can also see the peak at the mid-2010's, that’s due to start of Syrian Civil War in 2011 and global resurgence of ISIS in 2014.

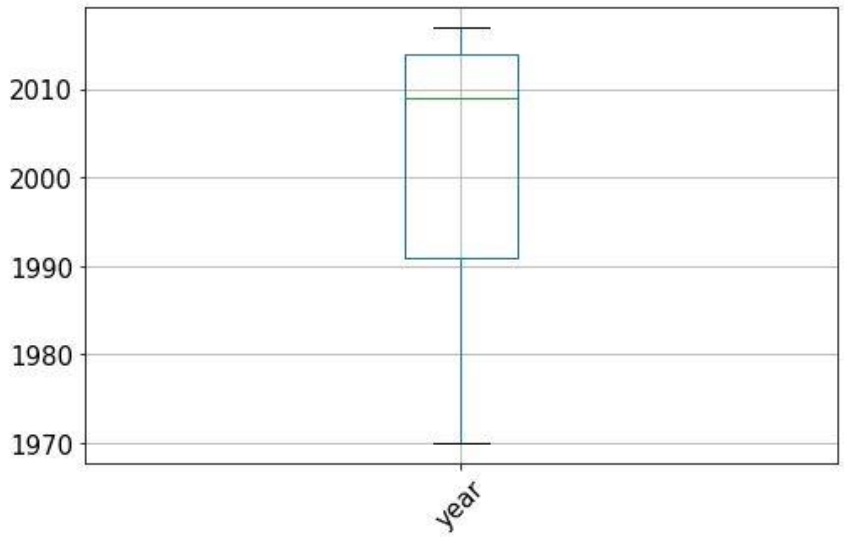
Boxplot

The main descriptive statistics of a sample can be represented graphically by a boxplot. To construct a boxplot, we draw a box between the first and the third quartiles, a line inside a box for a median, and extend whiskers to the smallest and the largest observations.

```
In [37]: def drawBoxByMainData(data):
             string = input()
             data.boxplot(column=string,rot=45,fontsize=15,figsize=(8,5))

         drawBoxByMainData(mainData)
```

```
year
```



Again, using the Pandas library we used the ".boxplot" function and generated a boxplot with "year" column and modified the size of font, figure, rotated it.

By looking at our boxplot we can tell that there are no outliers, median is 2009 and the gap between Median and Q1 is bigger than Q3 because of the high frequency of attacks in early-mid 2010's.

Confidence Interval

Gives a range of values for an unknown parameter (for example, a population mean). The interval has an associated confidence level that gives the probability with which an estimated interval will contain the true value of the parameter. The confidence level is chosen by the investigator. For a given estimation in a given sample, using a higher confidence level generates a wider (i.e., less precise) confidence interval. In general terms, a confidence interval for an unknown parameter is based on sampling the distribution of a corresponding estimator.

The Confidence Rule is applicable if a sample comes from any distribution, but the sample size n is large, then sample mean has an approximately Normal distribution according to the Central Limit Theorem. And this is our case.

Confidence interval for the mean; σ is known

$$\bar{X} \pm z_{\alpha/2}\frac{\sigma}{\sqrt{n}}$$

Confidence interval for the variance is also the same function, swapping mean with variance.

We know that for %95 confidence interval, $z_{0,025}$=1.96

```
In [94]: def findConfidenceInterval95(data,n):
             conf=1.96
             datan=data.sample(n)
             m=Mean(datan)
             v=Variance(datan)
             stdDev=Stdev(data)
             se=stdDev/math.sqrt(n)
             he=se*conf
             return"of sample mean=",(m-he, m+he),"of sample variance=",(v-he, v+he)

         findConfidenceInterval95(year,50000)

Out[94]: ('of sample mean=',
          (2002.4768162874075, 2002.7092637125925),
          'of sample variance=',
          (175.9400398458199, 176.1724872710048))
```

Got the z-value on “conf” variable, got the sample of data using “.sample” function from Pandas library. Calculated the mean and variance of sample data, calculated the standard deviation of population. Divided the standard deviation with square root of sample size and multiplied it with z-value. The result will be the right side of confidence interval formula. Returned the confidence interval for sample mean and variance by adding-removing the result of our calculations.

Sample size for a given precision

In order to attain a margin of error Δ for estimating a population mean with a confidence level $(1-\alpha)$,

a sample of size $n \geq \left(\frac{z_{\alpha/2} \cdot \sigma}{\Delta}\right)^2$ is required.

We know that for %90 confidence interval, $z_{0,05}$=1.645

```
In [91]: def findSample01marg90(data):
             marg=0.1
             conf=1.645
             stdDev=Stdev(data)
             nmin=pow(((conf*stdDev)/marg),2)
             return int(nmin)+1

         findSample01marg90(year)
```

```
Out[91]: 47576
```

Got our given 0.1 margin on “marg” value, got the z-value on “conf” variable, calculated the standard deviation for population. Multiplied the z-value with standard deviation, divided it and took the second-degree power of it. Returned the result by casting it into an integer (float result was 47575.039 but we need an integer value for sample size) and adding 1.