In [181…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import functools as fc
import functools as ft
import math
from IPython.display import Image
```

IMPORT THE EXCEL FILE

In [46]:
```
data=pd.read_excel("/Users/apple/Desktop/Data analyst/python fallsem lab/datas.xlsx")
data.head()
```

Out[46]:

	WBC_area	WBC_convex_area	WBC_peri	ecc_wbc	solidity_wbc	orient_wbc	nuc_area	nuc_ratio	peri_nuc	round_nuc	...
0	3798	3849	244.485281	0.580090	0.986750	-79.012451	2720	0.716166	237.764502	0.604623	...
1	3521	3590	236.727922	0.602939	0.980780	77.502581	2373	0.673956	204.066017	0.716088	...
2	2805	2805	208.000000	0.374387	1.000000	0.000000	1976	0.704456	173.195959	0.827792	...
3	2595	2616	200.242641	0.433734	0.991972	77.086906	1912	0.736802	176.267027	0.773313	...
4	2662	2704	209.313708	0.239700	0.984467	-33.831764	2025	0.760706	168.024387	0.901343	...

5 rows × 22 columns

Basic data information

In [99]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   WBC_area          10 non-null     int64
 1   WBC_convex_area   10 non-null     int64
 2   WBC_peri          10 non-null     float64
 3   ecc_wbc           10 non-null     float64
 4   solidity_wbc      10 non-null     float64
 5   orient_wbc        10 non-null     float64
 6   nuc_area          10 non-null     int64
 7   nuc_ratio         10 non-null     float64
 8   peri_nuc          10 non-null     float64
 9   round_nuc         10 non-null     float64
 10  ecc_nuc           10 non-null     float64
 11  solidity_nuc      10 non-null     float64
 12  convex_area_nuc   10 non-null     int64
 13  avg_cyt_re        10 non-null     float64
 14  avg_cyt_gr        10 non-null     float64
 15  avg_cyt_bl        10 non-null     float64
 16  entropy_cyt       10 non-null     float64
 17  minoraxis         10 non-null     float64
 18  majoraxis         10 non-null     float64
 19  minoraxis_nuc     10 non-null     float64
 20  majoraxis_nuc     10 non-null     float64
 21  axismeanratio     10 non-null     float64
dtypes: float64(18), int64(4)
memory usage: 1.8 KB
```

In [100…
```
data.describe()
```

Out[100…

	WBC_area	WBC_convex_area	WBC_peri	ecc_wbc	solidity_wbc	orient_wbc	nuc_area	nuc_ratio	peri_nuc	round
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.00
mean	3071.200000	3135.800000	220.380613	0.448796	0.979959	-18.928289	2210.600000	0.722930	194.332208	0.74
std	479.483936	494.119824	18.208200	0.145287	0.020192	63.355073	283.112149	0.032191	24.062707	0.10
min	2595.000000	2616.000000	200.242641	0.239700	0.931750	-85.255857	1912.000000	0.673956	168.024387	0.59
25%	2693.250000	2733.500000	206.242641	0.385612	0.981076	-73.921656	2030.750000	0.695345	176.559921	0.67
50%	2876.000000	2939.000000	213.162951	0.433133	0.985609	-38.849140	2107.000000	0.726484	184.630988	0.76
75%	3415.500000	3524.000000	237.627825	0.546599	0.990963	27.520138	2323.250000	0.746451	210.762716	0.81
max	3848.000000	3895.000000	244.485281	0.699188	1.000000	77.502581	2720.000000	0.767616	237.764502	0.90

8 rows × 22 columns

In [101…
```
data.shape
```

Out[101…
```
(10, 22)
```

In [103…
```
data.columns
```

Out[103…
```
Index(['WBC_area', 'WBC_convex_area', 'WBC_peri', 'ecc_wbc', 'solidity_wbc',
       'orient_wbc', 'nuc_area', 'nuc_ratio', 'peri_nuc', 'round_nuc',
       'ecc_nuc', 'solidity_nuc', 'convex_area_nuc', 'avg_cyt_re',
       'avg_cyt_gr', 'avg_cyt_bl', 'entropy_cyt', 'minoraxis', 'majoraxis',
       'minoraxis_nuc', 'majoraxis_nuc', 'axismeanratio'],
      dtype='object')
```

APPLY THE DATA STRUCTURE

There are many data structures in python some of them include lists,tupels,sets ect.For this assessment i chose to use lists as they are mutable.But tupels demonstration has been given.

In [49]:
```
first_row=data.iloc[0]
print(first_row)
```

```
WBC_area           3798.000000
WBC_convex_area    3849.000000
WBC_peri            244.485281
ecc_wbc               0.580090
solidity_wbc          0.986750
orient_wbc          -79.012451
nuc_area           2720.000000
nuc_ratio             0.716166
peri_nuc            237.764502
round_nuc             0.604623
ecc_nuc               0.672656
solidity_nuc          0.908787
convex_area_nuc    2993.000000
avg_cyt_re          124.895176
avg_cyt_gr          114.916512
avg_cyt_bl          145.590909
entropy_cyt         -18.413519
minoraxis            63.942810
majoraxis            78.500533
minoraxis_nuc        51.410517
majoraxis_nuc        69.477876
axismeanratio         1.689070
Name: 0, dtype: float64
```

In [180…
```
# converting first row from the dataframe to list
print(type(first_row))


first_list=tuple(first_list)
print(type(first_list))

first_list=list(first_row)
print(type(first_list))
summ=sum(first_row)
print(summ)
# Manipulating the values in list
first_list.clear() # clears the list
print(first_list)
```

```
<class 'pandas.core.series.Series'>
<class 'tuple'>
<class 'list'>
14399.716288155389
[]
```

EVALUATE THE VARIANT, CORRELATTION AND REGRESSION WITH HIGHER ORDER FUNCTIONS

I have considered the two columns WBC_area and WBC_convec_area.

Variant:

To locate the variation, I created a function that calculates the variance of the data, which aids us in understanding the dispersion of the data.

$$\mathrm{SD} = \sqrt{\frac{\sum |x - \bar{x}|^2}{n}}$$

Corelation:

I created the corelation function by combining higher order functions such as lambda, reduce, and map. They have proven extremely handy when repeating the same technique. The reduction function did the aggregation.

$$r_{xy} = \frac{\Sigma(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma(x_i - \bar{x})^2 \, \Sigma(y_i - \bar{y})^2}}$$

r_{xy} = correlation coefficient between x and y

x_i = the values of x within a sample

y_i = the values of y within a sample

$\bar{x}$ = the average of the values of x within a sample

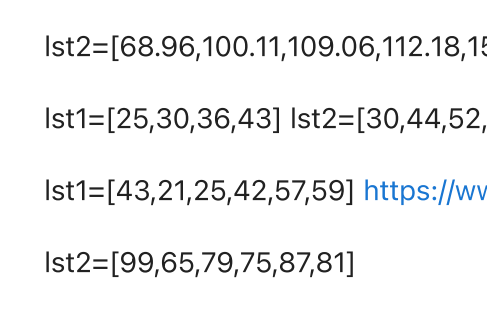
$\bar{y}$ = the average of the values of y within a sample

Linear Regression:

Using the statistical formula, I created the Linear Regression function.Higher order functions like as map, reduce, and lambda are used in the code.

$$A = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$B = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$



lst1[1691.75,1977.8,1884.09,2151.13,2519.36]https://corporatefinanceinstitute.com/resources/knowledge/finance/correlation/

lst2=[68.96,100.11,109.06,112.18,154.12]

lst1=[25,30,36,43] lst2=[30,44,52,70]

lst1=[43,21,25,42,57,59] https://www.statisticshowto.com/probability-and-statistics/correlation-coefficient-formula/

lst2=[99,65,79,75,87,81]

lst1=[1,2,3,4,5,6,7] lst2=[8,9,10,11,12,13,14] https://www.brainkart.com/article/Solved-Example-Problems-for-Regression-Analysis_37036/

test=[24,53,53,36,21,84,64,34,77,54] https://www.cuemath.com/variance-formula/

These are some of the test cases i have used to check the correctness of the code output.

In [166…
```
x=list(data["WBC_area"])
y=list(data["WBC_convex_area"])


def corr(lst1,lst2):

    sx=fc.reduce(lambda x,y:x+y,lst1,0)
    meanls1=sx/len(lst1)
    sy=fc.reduce(lambda x,y:x+y,lst2,0)
    meanls2=sy/len(lst1)
    #print(meanls1)
    #print(meanls2)
    dx=[]
    dy=[]
    nm=[]
    for item in lst1:
        dx.append(item-meanls1)

    for item in lst2:
        dy.append(item-meanls2)

    itdx=iter(dx)
    itdy=iter(dy)
    n=len(dx)
    #print(dx,dy)
    while(n>0):
        nm.append(next(itdx)*next(itdy))
        n=n-1
    #print(sum(nm)) #Numerator
    #Denominator
    denx=list(map(lambda x:x**2,dx))
    deny=list(map(lambda x:x**2,dy))
    #print(sum(denx),sum(deny))
    rxy=round((sum(nm))/(math.sqrt(sum(denx)*sum(deny))),2)
    print(rxy)

def linear_regression(lst1,lst2):
    #Y=a+bX
    summ=0
    sigmax=fc.reduce(lambda x,y:x+y,lst1)
    sigmay=fc.reduce(lambda x,y:x+y,lst2)
    sigmaxsq=fc.reduce(lambda x,y:x+y,list(map(lambda x:x**2,lst1)))
    for i in range(0,len(lst1)):
        x=lst1[i]*lst2[i]
        summ=summ+x
    sigmaxy=summ
    sigmaxwsq=sigmax**2
    n=len(lst1)

    #print(sigmax,sigmay,sigmaxsq,sigmaxy,sigmaxwsq)

    a=round(((sigmay*sigmaxsq)-(sigmax*sigmaxy))/((n*sigmaxsq)-sigmaxwsq),2)
    b=round(((n*sigmaxy)-sigmax*sigmay)/((n*sigmaxsq)-sigmaxwsq),2)

    #print(a,b)

    print("The linear regression equation is Y={}x+{}".format(a,b))

    return a,b

def variant(lst1):

    mean=fc.reduce(lambda x,y:x+y,lst1)/len(lst1)
    #print(mean)
    n=len(lst1)
    num=sum(list(list(map(lambda x:x**2,list(map(lambda x:x-mean,lst1))))))
    #print(num)
    #print(n-1)

    print(math.sqrt(num/(n-1)))

def test_case_lr(lst1,lst2,var1):

    #Linear Regression Line

    a,b=linear_regression(lst1,lst2)


    Y=a+(b*var1)

    print(Y)
```

When we want to find the predected value of y for given value x, we use the linear regression to generate a line that satisfies the datta points.Here is the example regression line for the given data and the value of y for given X(3466)

In [167…
```
x=list(data["WBC_area"])
y=list(data["WBC_convex_area"])
print("The solution from the regression line when the value is 3466")
test_case_lr(x,y,3466)

print("Corelation value:")
z=corr(lst1,lst2)

print("variant value:")
variant(x)
```

```
The solution from the regression line when the value is 3466
The linear regression equation is Y=-0.86x+1.02
3534.46
Corelation value:
0.53
variant value:
479.48393554366805
```

EVALUATE THE VARIANT, CORRELATTION AND REGRESSION USING RECURSIVE FUNCTION

For regression, I utilised recursive functions to calculate the total of the list, sum of squares of the list, and product of congruent parts in the list.

For corelation i used the similar recursive functions

In [169…
```
# x=list(data["x"])
# y=list(data["y"])
# lst1=[43,21,25,42,57,59]
# lst2=[99,65,79,75,87,81]
x=list(data["WBC_area"])
y=list(data["WBC_convex_area"])

def addno(lst,N):
    if N==0:
        return 0

    else:
        return addno(lst,N-1)+lst[N-1]

def numerator(lst1,lst2,mean1,mean2):
    if len(lst1)==1 and len(lst2)==1: # base condition
        return ((lst1[0]-mean1)*(lst2[0]-mean2))
    return ( numerator([lst1[0]],[lst2[0]],mean1,mean2) + (numerator(lst1[1:],lst2[1:],mean1,mean2)) ) # recur

def denp(lst1,mean):
    if len(lst1)==1:
        return((lst1[0]-mean)**2)

    else:
        return denp([lst1[0]],mean)+denp(lst1[1:],mean)


def corcursion(lst1,lst2):
    sx=addno(lst1,len(lst1))
    meanls1=sx/len(lst1)
    #print(sx,meanls1)
    sy=addno(lst2,len(lst2))
    meanls2=sy/len(lst1)
    #print(sy,meanls2)
    num=numerator(lst1,lst2,meanls1,meanls2)
    #print(num)
    denp1=denp(lst1,meanls1)
    #print(denp1)
    denp2=denp(lst2,meanls2)

    core=round((num/(math.sqrt(denp1)*math.sqrt(denp2))),2)
    print(core)

x=list(data["WBC_area"])
y=list(data["WBC_convex_area"])

def addno(lst,N):
    if N==0:
        return 0

    else:
        return addno(lst,N-1)+lst[N-1]

def sumprod(lst1,lst2): # Recursive function to calculate the product (xi*yi)

    if len(lst1)==1 and len(lst2)==1:
        return lst1[0]*lst2[0]

    else:
        return sumprod([lst1[0]],[lst2[0]])+sumprod(lst1[1:],lst2[1:])


def regression_recurr(lst1,lst2):
    summ=0
    n=len(lst1)
    sigmax=addno(lst1,len(lst1))
    sigmay=addno(lst2,len(lst2))
    meanx=sigmax/len(lst1)
    meany=sigmay/len(lst2)
#   print(lst1,lst2)
    sigmaxy=sumprod(lst1,lst2)
    sigmaxsq=fc.reduce(lambda x,y:x+y,list(map(lambda x:x**2,lst1)))
    for i in range(0,len(lst1)):
        x=lst1[i]*lst2[i]
        summ=summ+x
    sigmaxy=summ
    sigmaxwsq=sigmax**2
    n=len(lst1)
    a=round(((sigmay*sigmaxsq)-(sigmax*sigmaxy))/((n*sigmaxsq)-sigmaxwsq),2)
    b=round(((n*sigmaxy)-sigmax*sigmay)/((n*sigmaxsq)-sigmaxwsq),2)
    #print(a,b)

    print("The linear regression equation is Y={}x+{}".format(a,b))

def test_case_lr(lst1,lst2,var1):

    #Linear Regression Line

    a,b=linear_regression(lst1,lst2)


    Y=a+(b*var1)

    print(Y)

def Variance(z):
    #variance=clsquaresum/n
    def summ(lst):
        if len(lst)==1:
            return lst[0]
        else:
            return lst[0]+sum(lst[1:])
    def lS(lst):
        if len(lst)== 1:
            return [lst[0]**2]  # Returning a list
        else:
            return [lst[0]**2] + lS(lst[1:])
    def sub(lst,z):
        if len(lst)== 1:
            return [lst[0]-z]   # Returning a list
        else:
            return [lst[0]-z] + sub(lst[1:],z)

    n=len(z)
    X1sum=summ(z)
    X1mean=X1sum/n
    c1=sub(z,X1mean)#c1=X1-X1mean
    clsquare=lS(c1)
    clsquaresum=sum(clsquare)
    Variance=clsquaresum/(n-1)
    print( math.sqrt(Variance))
```

In [170…
```
print("The solution from the regression line when the value is 3466")
test_case_lr(x,y,3466)

print("Corelation value:")
z=corcursion(lst1,lst2)

print("variant value:")
Variance(x)
```

```
The solution from the regression line when the value is 3466
The linear regression equation is Y=-0.86x+1.02
3534.46
Corelation value:
0.53
variant value:
479.48393554366805
```

This code is for calculating the variance in the row after applying the quicksort function and using recursive techniques/higher order.

In [175…
```
 # quicksort code
def quicksort(first_row):
    if first_row == []:
        return []
    pivot = first_row[0]
    less=[]
    greater=[]
    for item in first_row[1:]:
        if item>pivot:
            less.append(item)
        else:
            greater.append(item)
    all=quicksort(greater)+[pivot]+quicksort(less)
    return all
# applying quicksort to the above list(i.e first row in excel file)
sorted_row=quicksort(list(first_row))
print("Sorted row:")
print(sorted_row)

Variance(sorted_row)
variant(sorted_row)
```

```
Sorted row:
[-79.01245087128778, -18.413519365941077, 0.5800897671102989, 0.6046231023145049, 0.6726561322059249, 0.7161664
033701949, 0.9087871700634814, 0.9867498051441933, 1.6890702514650064, 51.41051668955135, 63.94280966993384, 6
9.47787553973552, 78.50053309706678, 114.91651205936593692, 124.8951762523191, 145.5909090909091, 237.7645019878174
8, 244.48528137423864, 2720.0, 2993.0, 3798.0, 3849.0]
1315.8663046752333
1315.8663046752333
```

In [ ]:

In [ ]: