In [1119]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import math
```

In [1120]:
```
pima=pd.read_csv("diabetes.csv")
```

In [1121]:
```
pima
```

Out[1121]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

768 rows × 9 columns

In [1122]:
```
#Taking required features for training into a list
features=["Glucose","BloodPressure","SkinThickness","Outcome"]
#Creating a DataFrame with the list features
X_temp=pima[features]
#Spliting the Datasets into 2 parts i.e Training Set and Test Set
X_train,X_test = train_test_split(X_temp, test_size=0.5)
```

In [1123]:
```
X_train
```

Out[1123]:

	Glucose	BloodPressure	SkinThickness	Outcome
744	153	88	37	0
654	106	70	28	0
112	89	76	34	0
98	93	50	30	0
721	114	66	36	0
...	...	...	...	...
710	158	64	13	0
526	97	64	19	0
160	151	90	38	0
134	96	68	13	0
65	99	74	27	0

384 rows × 4 columns

In [1124]:
```
#Method to calculating Euclidean Distance for two Vectors
def dist(x,y):
    temp=(x-y)**2
    s=np.sum(temp,axis=0)
    d=np.sqrt(s)
    return d
```

In [1125]:
```
#Method to Perform KNN
def knn(x,k,x_old):
    #Making a copy of the Dataset to temparily store the distance values as a
 separate column
    x1=x_old.copy()
    #Temp List stores the distance of the numpy array "x" to all the sample fr
om the given Dataset"x_old"
    temp=[]
    for i in range(x_old.shape[0]):
        d=dist(x,x_old.iloc[i,:].to_numpy().reshape(X_train.shape[1],1))
        temp.append(d)
    #Adding new column as the distance to the "x1" DataFrame
    x1["distance"]=temp
    #Sorts the DataFrame based upon the distance values
    x2=x1.sort_values(by=['distance'])
    #sorted_k_indexes stores the K Nearest Neighbours to the test sample
    sorted_k_indexes=[]
    for j in range(k):
        sorted_k_indexes.append(x2.iloc[j,4])
    classA=0
    classB=0
    #calculating how many classA and ClassB outcomes were there in these K val
ues
    for p in range(len(sorted_k_indexes)):
        if(sorted_k_indexes[p]==1):
            classB+=1
        else:
            classA+=1
    #if ClassB values were more than assign the test sample to ClassB (i.e out
come 1) else ClassA
    if(classA>classB):
        return 0
    else:
        return 1
```

In [1126]:
```
#Accuracy for K=1
predicted_outcome=[]
for k in range(X_test.shape[0]):
    #Changing the test sample from Pandas Series to numpy Array and reshaping
 it
    a=X_test.iloc[k,:].to_numpy().reshape(X_train.shape[1],1)
    #Calling the knn method and storing the return in a label
    label=knn(a,1,X_train)
    if(label==1):
        predicted_outcome.append(1)
    else:
        predicted_outcome.append(0)
XTest=X_test.copy()
#adding a column to the test set DataFrame with Predicted values
XTest["predictedOutcome"]=predicted_outcome
correct=0
wrong=0
for k in range(XTest.shape[0]):
    #If predicted values and actual outcome is same then increment the correct
variable
    if(XTest.iloc[k,3]==XTest.iloc[k,4]):
        correct+=1
    else:
        wrong+=1
print("Total no of correctly predicted values:" +str(correct))
Accuracy_k1=correct/X_test.shape[0]
print("Accuracy: "+str(Accuracy_k1))
```

```
Total no of correctly predicted values:242
Accuracy: 0.6302083333333334
```

In [1127]:
```
#Accuracy for K=5
predicted_outcome=[]
for k in range(X_test.shape[0]):
    a=X_test.iloc[k,:].to_numpy().reshape(X_train.shape[1],1)
    label=knn(a,5,X_train)
    if(label==1):
        predicted_outcome.append(1)
    else:
        predicted_outcome.append(0)
XTest=X_test.copy()
XTest["predictedOutcome"]=predicted_outcome
correct=0
wrong=0
for k in range(XTest.shape[0]):
    if(XTest.iloc[k,3]==XTest.iloc[k,4]):
        correct+=1
    else:
        wrong+=1
print("Total no of correctly predicted values:" +str(correct))
Accuracy_k5=correct/X_test.shape[0]
print("Accuracy: "+str(Accuracy_k5))
```

```
Total no of correctly predicted values:247
Accuracy: 0.6432291666666666
```

In [1128]:
```
#Accuracy for K=11
predicted_outcome=[]
for k in range(X_test.shape[0]):
    a=X_test.iloc[k,:].to_numpy().reshape(X_train.shape[1],1)
    label=knn(a,11,X_train)
    if(label==1):
        predicted_outcome.append(1)
    else:
        predicted_outcome.append(0)
XTest=X_test.copy()
XTest["predictedOutcome"]=predicted_outcome
correct=0
wrong=0
for k in range(XTest.shape[0]):
    if(XTest.iloc[k,3]==XTest.iloc[k,4]):
        correct+=1
    else:
        wrong+=1
print("Total no of correctly predicted values:" +str(correct))
Accuracy_k11=correct/X_test.shape[0]
print("Accuracy: "+str(Accuracy_k11))
```

```
Total no of correctly predicted values:247
Accuracy: 0.6432291666666666
```

In [1129]:
```
#storing the accuracy of 10 iterations in a list
#ListAccuracy_k1=[]
#ListAccuracy_k5=[]
#ListAccuracy_k11=[]
```

In [1130]:
```
ListAccuracy_k1.append(Accuracy_k1)
ListAccuracy_k5.append(Accuracy_k5)
ListAccuracy_k11.append(Accuracy_k11)
```

In [1131]:
```
ListAccuracy_k1
```

Out[1131]:
```
[0.640625,
 0.6536458333333334,
 0.6432291666666666,
 0.6796875,
 0.6197916666666666,
 0.6354166666666666,
 0.6458333333333334,
 0.6380208333333334,
 0.6666666666666666,
 0.6041666666666666,
 0.6302083333333334]
```

In [1132]:
```
ListAccuracy_k5
```

Out[1132]:
```
[0.6510416666666666,
 0.6614583333333334,
 0.6588541666666666,
 0.6927083333333334,
 0.6302083333333334,
 0.6588541666666666,
 0.6614583333333334,
 0.6510416666666666,
 0.6822916666666666,
 0.6197916666666666,
 0.6432291666666666]
```

In [1133]:
```
ListAccuracy_k11
```

Out[1133]:
```
[0.6510416666666666,
 0.6614583333333334,
 0.6588541666666666,
 0.6927083333333334,
 0.6302083333333334,
 0.6588541666666666,
 0.6588541666666666,
 0.6510416666666666,
 0.6822916666666666,
 0.6171875,
 0.6432291666666666]
```

In [1134]:
```
#mean accuracy
Mean_k1=sum(ListAccuracy_k1)/len(ListAccuracy_k1)
Mean_k5=sum(ListAccuracy_k5)/len(ListAccuracy_k5)
Mean_k11=sum(ListAccuracy_k11)/len(ListAccuracy_k11)

print("Mean for K==1: "+str(Mean_k1))
print("Mean for K==5: "+str(Mean_k5))
print("Mean for K==11: "+str(Mean_k11))
```

```
Mean for K==1: 0.6415719696969696
Mean for K==5: 0.6555397727272728
Mean for K==11: 0.655066287878788
```

In [1135]:
```
std_dev=0
for p in range(len(ListAccuracy_k1)):
    std_dev+=(ListAccuracy_k1[p]-Mean_k1)**2
std_d=std_dev/(len(ListAccuracy_k1)-1)
std_deviation_k1=math.sqrt(std_d)
#Standard Deviation of the accuracy
print(std_deviation_k1)
```

```
0.020744364955307508
```

In [1136]:
```
std_dev=0
for p in range(len(ListAccuracy_k5)):
    std_dev+=(ListAccuracy_k5[p]-Mean_k5)**2
std_d=std_dev/(len(ListAccuracy_k5)-1)
std_deviation_k5=math.sqrt(std_d)
#Standard Deviation of the accuracy
print(std_deviation_k5)
```

```
0.020738420159382223
```

In [1137]:
```
std_dev=0
for p in range(len(ListAccuracy_k11)):
    std_dev+=(ListAccuracy_k11[p]-Mean_k11)**2
std_d=std_dev/(len(ListAccuracy_k11)-1)
std_deviation_k11=math.sqrt(std_d)
#Standard Deviation of the accuracy
print(std_deviation_k11)
```

```
0.021135941483918244
```