----------بسم الله الرحمن الرحيم ----------

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn import svm
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import  LogisticRegression
from sklearn.metrics import accuracy_score as ac
import pickle
from matplotlib import pyplot as plt
```

Understanding the data

In [2]:
```
df = pd.read_csv("diabetes.csv")
df.head()
datafr = df
```

In [3]:
```
df.tail()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outco
763	10	101	76	48	180	32.9	0.171	63	
764	2	122	70	27	0	36.8	0.340	27	
765	5	121	72	23	112	26.2	0.245	30	
766	1	126	60	0	0	30.1	0.349	47	
767	1	93	70	31	0	30.4	0.315	23	

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [5]:
```
df.shape
```

Out[5]:
```
(768, 9)
```

In [6]:
```
df.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctic
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.4718
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.3313
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0780
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.2437
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.3725
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.6262
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.4200

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

1 ----------> diabtic 0 ----------> undiabtic

In [8]:
```
df.Outcome.value_counts()
```

Out[8]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [9]:
```
df.groupby("Outcome").mean()
```

Out[9]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550

perprocessing

In [10]:
```
df.columns =df.columns.str.lower()
```

In [11]:
```
str_col = df.columns
str_target = ['outcome']
X = df[str_col[:-1]]
Y = df[str_target]
```

In [12]:
```
X.head()
```

Out[12]:

	pregnancies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [13]:
```
Y.tail()
```

Out[13]:

	outcome
763	0
764	0
765	0
766	1
767	0

Data Standardization

In [14]:
```
scaler = StandardScaler()
```

In [15]:
```
scaler.fit(X)
```

Out[15]:
```
▼ StandardScaler
StandardScaler()
```

In [16]:
```
standardized_data = scaler.transform(X)
```

In [17]:
```
standardized_data
```

Out[17]:
```
array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
        -0.47378505, -0.87137393]])
```

In [18]:
```
X = standardized_data
```

In [19]:
```
X.shape
```

Out[19]:
```
(768, 8)
```

Spliting the data

In [20]:
```
X_train , X_test,y_train,y_test =train_test_split(X,Y,test_size=0.2,stratify = Y,random_
```

In [21]:
```
print(X_train.shape , X_test.shape,y_train.shape,y_test.shape)
```

```
(614, 8) (154, 8) (614, 1) (154, 1)
```

Training the model

In [22]:
```
classifier = svm.SVC(kernel = 'linear')
```

In [23]:
```
classifier.fit(X_train,y_train)
```

```
C:\Users\PC\conda\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarn
ing: A column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[23]:

▼ SVC

```
SVC(kernel='linear')
```

Model evaluation

In [24]:
```
prediction = classifier.predict(X_train )
prediction[2]
```

Out[24]:
```
1
```

accuracy score

In [25]:
```
prediction = classifier.predict(X_train )
score = ac(prediction,y_train)
score
#prediction
```

Out[25]:
```
0.7866449511400652
```

In [26]:
```
prediction = classifier.predict(X_test )
score = ac(prediction,y_test)
score
#prediction
```

Out[26]:
```
0.7727272727272727
```

DONE

Building a predictive system

all our data is numerical so our input will be numerical as well it mighit be an array ,a list or even a dictionary

In [27]:
```
# as atuple
x11 = (1,126,60,0,0,30.1,0.349,47)
```

In [28]:
```
def is_diabtic (x):
   xx= pd.DataFrame(x)
   X = np.asarray(xx)

   scaler.fit(X)
   X_train =scaler.transform(X)
   prediction = classifier.predict (X_train.T)

   if prediction == [1]:
      print ("This person has  diabetes ")
   else:
      print ("This person doesn't have  diabetes ")
```

In [29]:
```
is_diabtic(x11)
```

```
This person has  diabetes
```

Save the function of the model

In [30]:
```
pickle.dump(is_diabtic,open("diabetes_disease.pkl","wb"))
```

In [31]:
```
fun = pickle.load(open("diabetes_disease.pkl",'rb'))
```

In [32]:
```
fun(x11)
```

```
This person has  diabetes
```

DONE ALHAMDULLAH

In [33]:
```
df
```

Out[33]:

	pregnancies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [34]:
```
datafr.head()
```

Out[34]:

	pregnancies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [35]:

```
def Is_Diabtic() :
   print("this folloeing data must be filledout by the doctor  ")
   Pregnancies = float(input("pregnanancies : "))
   Glucose = float(input("glucose : "))
   BloodPressure = float(input("glucose : "))
   SkinThickness= float(input("SkinThickness : "))
   Insulin = float(input("Insulin : "))
   BMI= float(input("BMI : "))
   DiabetesPedigreeFunction = float(input("glucose : "))
   Age = float(input("Age : "))
   listt = [Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigree
   fun(listt)
```

In [ ]:

```
Is_Diabtic()
```

```
this folloeing data must be filledout by the doctor
```

In [ ]: