In [1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

In [2]:
```
# data collection and analysis
```

In [3]:
```
diabetic_dataset=pd.read_csv('diabetes.csv')
```

In [4]:
```
diabetic_dataset.head()
```

Out[4]:

	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 [5]:
```
# no of row and column in dataset
diabetic_dataset.shape
```

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

In [6]:
```
#gettting the statiscal measure of data
diabetic_dataset.describe()
```

Out[6]:

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

In [7]:
```
diabetic_dataset['Outcome'].value_counts()
```

```
0    500
```

Out[7]:
```
1    268
Name: Outcome, dtype: int64
```

In [8]:
```
diabetic_dataset.groupby('Outcome').mean()
```

Out[8]:

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

In [9]:
```
#seperate data and labels
x=diabetic_dataset.drop(columns='Outcome',axis=1)
y=diabetic_dataset['Outcome']
```

In [10]:
```
print(x)
```

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

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

In [11]:
```
print(y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [12]:
```
# Data standardization
scaler=StandardScaler()
scaler.fit(x)
standard_data=scaler.transform(x)
```

In [13]:
```
print(standard_data)# we can see here all the value are in similar range..to make be
```

```
[[ 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 [14]:
```
x=standard_data
```

In [15]:
```
#train test split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,stratify=y,random_st
```

In [16]:
```
print(x.shape,x_train.shape,x_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

In [17]:
```
#making model
models = {
    "                    Logistic Regression": LogisticRegression(),
    "                    K-Nearest Neighbors": KNeighborsClassifier(),
    "                          Decision Tree": DecisionTreeClassifier(),
    "Support Vector Machine (Linear Kernel)": LinearSVC(),
    "   Support Vector Machine (RBF Kernel)": SVC(),
    "                          Neural Network": MLPClassifier(),
    "                           Random Forest": RandomForestClassifier(),
    "                       Gradient Boosting": GradientBoostingClassifier()
}

for name, model in models.items():
    model.fit(x_train, y_train)
    print(name + " trained.")
```

```
                    Logistic Regression trained.
                    K-Nearest Neighbors trained.
                          Decision Tree trained.
Support Vector Machine (Linear Kernel) trained.
   Support Vector Machine (RBF Kernel) trained.
C:\Users\gtgau\anaconda3\lib\site-packages\sklearn\svm\_base.py:976: ConvergenceWarn
ing: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
C:\Users\gtgau\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  warnings.warn(
                          Neural Network trained.
                           Random Forest trained.
                       Gradient Boosting trained.
```

In [18]:
```
for name, model in models.items():
    print(name + ": {:.2f}%".format(model.score(x_test, y_test) * 100))
```

```
                    Logistic Regression: 75.97%
                    K-Nearest Neighbors: 72.08%
                          Decision Tree: 67.53%
Support Vector Machine (Linear Kernel): 75.97%
   Support Vector Machine (RBF Kernel): 72.73%
```

```
Neural Network: 75.32%
Random Forest: 74.03%
Gradient Boosting: 70.78%
```

In [19]:
```
#model evaluation
# accuracy score on training data
classifier=LinearSVC()
classifier.fit(x_train, y_train)
x_train_pridiction=classifier.predict(x_train)
training_data_accuracy=accuracy_score(x_train_pridiction,y_train)
```

```
C:\Users\gtgau\anaconda3\lib\site-packages\sklearn\svm\_base.py:976: ConvergenceWarn
ing: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

In [20]:
```
print("accuracy score of training data is",training_data_accuracy)
```

```
accuracy score of training data is 0.7899022801302932
```

In [21]:
```
#accuracy score on test data
x_test_pridiction=classifier.predict(x_test)
test_data_accuracy=accuracy_score(x_test_pridiction,y_test)
```

In [22]:
```
print("accuracy score of test data is",test_data_accuracy)
```

```
accuracy score of test data is 0.7597402597402597
```

------>Making a predictive system

In [23]:
```
input_data = (5,166,72,19,175,25.8,0.587,51)
print(input_data)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)
print(input_data_as_numpy_array)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)
print(input_data_reshaped)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

```
(5, 166, 72, 19, 175, 25.8, 0.587, 51)
[  5.    166.     72.     19.    175.     25.8     0.587  51.   ]
[[  5.    166.     72.     19.    175.     25.8     0.587  51.   ]]
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person is diabetic
```

In [ ]:
```
```