Objectif: A- Collection du data et faire de l'analyse B- Separer et Nomaliser notre data C- Creation du notre model basant sur le svm D- Creer un interface de notre application basé sur le streamlit

1. Impoting Dependencies :

In [ ]:
```
import numpy as np #Library of basic table
import pandas as pd #Pandas library of importing a csv/text/excel
from sklearn.preprocessing import StandardScaler #importer la classe StandardSca
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

2. Data Collcetion and Analysis:

A- Loading dataset to a panda dataframe:

In [ ]:
```
#Chargement des données dans un DataFrame pandas à partir d'un fichier CSV.
data = pd.read_csv('C:/Machine_learning Python/projets/Diabet/diabetes.csv')
```

B- View the data (head)

In [ ]:
```
#Affichage des premières lignes de données pour comprendre la structure.
data.head()
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
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	

C-Number of row & columns:

In [ ]:
```
#Nombre de ligne et colonne dans notre dataFrrame
data.shape
```

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

2. Statisctical measures :

A. General Statistic:

In [ ]:
```
#Analyse descriptive pour obtenir des statistiques telles que la moyenne, l'écar
data.describe()
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
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

B. Value of the Diabetic & Nom Diabetic Person

In [ ]:
```
#Comptage des valeurs pour les résultats diabétiques et non diabétiques.
data['Outcome'].value_counts()
# 0 ---> Nom Diabetic
# 1 --->Diabetic
```

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

C. Grouping by the Outcome

In [ ]:
```
#Grouper les données par résultat pour analyser les différences moyennes entre l
data.groupby('Outcome').mean()
```

Out[ ]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
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

D. Separating the data and labels

In [ ]:
```
#Separer nos données entre les parametres a calculé (X) et le output (Y)
X = data.drop(columns = 'Outcome', axis = 1)
Y = data['Outcome']
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 [ ]:
```
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
```

3. Data Standarization:

A. Creat the variable of strandarization

In [ ]:
```
#Standardisation des données (normaliser nos de data)
scaler = StandardScaler() #Création d'un objet de standardisation.
scaler.fit(X) #traini cette objet pas nos parametre X
```

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

B. Fiting and transforming on the new data:

In [ ]:
```
standardized_data = scaler.transform(X) #Transfomer notre data en la nouvelle da
```

C. The new data:

In [ ]:
```
print(standardized_data) #View of the head of our new data
```

```
[[ 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]]
```

D. Creating the new X and Y

In [ ]:
```
X = standardized_data # notre X (nos parametre) devient nomaliser et standariser
Y = data['Outcome']   # Y inchanché
```

In [ ]:
```
print(X)
```

```
[[ 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 [ ]:
```
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
```

4. Train test split:

A. Variables of testing and training:

In [ ]:
```
X_train,X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2, stratify=Y
#Utilisation de train_test_split pour diviser les données entre les donnée X et
#80% des données pour l'entrainement et 20% pour le test
```

In [ ]:
```
print(X.shape, X_train.shape, X_test.shape) #dimmenssion of the new X et Y
```

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

4. Training the model:

A. Variable of classification

In [ ]:
```
classifier = svm.SVC(kernel='linear') #Création  d'un modèle SVM (machine à vect
```

Explication: Le modèle SVM (Support Vector Machine) est un outil puissant pour la classification et la régression dans le domaine du machine learning. L'un des aspects les plus importants et puissants de SVM est son utilisation des fonctions noyau, ou "kernels", qui permettent de traiter des données non linéaires. Lors de l'utilisation d'un SVM avec un noyau linéaire, voici ce qu'il est essentiel de comprendre :

Noyau Linéaire Un noyau linéaire est le type de noyau le plus simple dans les SVM. Il est utilisé quand on suppose que les données sont linéairement séparables, c'est-à-dire qu'il est possible de séparer les classes à l'aide d'une droite (en 2D), d'un plan (en 3D) ou d'un hyperplan dans des dimensions plus élevées.

Avantages du Noyau Linéaire : Simplicité : Étant donné que le noyau linéaire n'implique que des opérations de produits scalaires, il est très rapide et efficace en termes de calcul, surtout pour des grands jeux de données où la linéarité est une hypothèse raisonnable. Interprétabilité : Les modèles utilisant un noyau linéaire sont souvent plus faciles à interpréter, car ils reposent directement sur les caractéristiques originales sans transformations complexes.

B.Training the support vector Machine Classifier:

In [ ]:
```
classifier.fit(X_train,Y_train) #Entrainer notre modele avec notre variable X_tr
```

Out[ ]:
```
SVC
SVC(kernel='linear')
```

5. Evaluate the model:

A. Accuracy Score of training:

In [ ]:
```
#évaluer la performance du modèle par rapport à l'entrainent (training)
X_train_predicition = classifier.predict(X_train) #Creation de la variable de pr
training_data_accuracy = accuracy_score(X_train_predicition, Y_train) #creation
print('Accuracy score of the training data:', training_data_accuracy)
```

```
Accuracy score of the training data: 0.7866449511400652
```

B. Accuracy Score of testing:

In [ ]:
```
#évaluer la performance du modèle par rapport au test (training)
X_test_predicition = classifier.predict(X_test) #Creation de la variable de prea
training_data_accuracy = accuracy_score(X_test_predicition, Y_test) #creation de
print('Accuracy score of the training data:', training_data_accuracy)
```

```
Accuracy score of the training data: 0.7727272727272727
```

C. Application:

In [ ]:
```
#Definir notre fonction avec les parametre à recupere du notre patient

def  predictionF(Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,Dia

    #Enter les donnée dans une tuble (like a tableau)
    input_data = (Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,Di
    #Input the data into the numpy array (transofmrer nos donnée en un numpy tab
    input_dataNumpuy = np.asarray(input_data)
    #Reshape the data (redimenssioner notre data)
    input_dataReshaped = input_dataNumpuy.reshape(1,-1)
    #Standariser the data (normaliser notre data)
    std_data = scaler.transform(input_dataReshaped)
    #Predict the model (utilisation notre model pour faire la predicition de not
    prediction = classifier.predict(std_data)

    print(prediction[0])
    if(prediction[0] == 1):
        print("The person is diabetic")
    else:
        print("The person is not diabetic")

predictionF(0,131,0,0,0,43.2,0.27,26)
```

```
1
The person is diabetic
c:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\bas
e.py:465: UserWarning: X does not have valid feature names, but StandardScaler wa
s fitted with feature names
  warnings.warn(
```

II.Deployment:

1. Saving the trained model:

In [ ]:
```
#Bibiliotheque qui permer de saver les models
import pickle
```

In [ ]:
```
#Save notre model dans un path nomé trained_model.sav
filename = 'trained_model.sav'
#Save le model
pickle.dump(classifier , open(filename , 'wb'))
```

2. Load the saved model:

In [ ]:
```
#loading the model
loaded_model = pickle.load(open('trained_model.sav', 'rb'))
```