The project focuses on predicting whether a person has diabetes based on health-related data, using the PIMA Diabetes Dataset. The machine learning algorithm Support Vector Machine (SVM) is used to classify individuals as diabetic or non-diabetic. This project demonstrates how machine learning can help in healthcare by providing an early indication of potential health issues.

IMPORTING THE LIBRARIES

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
```

This project builds a predictive system using machine learning to classify individuals as diabetic or not based on specific health parameters, such as blood glucose levels, BMI, and age. The model is trained using SVM, a powerful algorithm for classification problems.

Data Collection and Analysis

In [3]:

```
# loading the diabetes dataset to a pandas DataFrame
dataset = pd.read_csv('/content/diabetes.csv')
```

In [4]:

```
#PRINTING FIRST FIVE ROWS
diabetes_dataset.head()
```

Out[4]:

	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 [5]:

```
#TO FIND NUMBER OF ROWS AND COLUMNS
dataset.shape
```

Out[5]:

```
(768, 9)
```

In [6]:

```
#TO GET STATISTICAL MEASURE OF DATA STRUCTURE.
dataset.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	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	0.471876	33.240885

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

In [7]:

```
dataset.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 [8]:

```
#COUNTING 0 AND 1 NON-DIABETIC AND DIABATIC VALUES RESPECTIVELY.
dataset['Outcome'].value_counts()
```

Out[8]:

	count
Outcome	
0	500
1	268

dtype: int64

In [9]:

```
diabetes_dataset.groupby('Outcome').mean()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
Outcome								
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.19000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.06716

In [10]:

```
#SPLITTING X AND Y(TARGET)
X = dataset.drop(columns = 'Outcome', axis=1)
Y = dataset['Outcome']
```

In [11]:

```
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 [12]:

```
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
```

STANDARDIZING THE DATA

Standardization ensures that all feature values are scaled to the same range

In [13]:

```
scaler = StandardScaler()
```

In [14]:

```
scaler.fit(X)
```

Out[14]:

▼ StandardScaler i ?

StandardScaler()

In [15]:

```
standardized_data = scaler.transform(X)
```

In [16]:

```
print(standardized_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]]
```

In [17]:

```
X = standardized_data
Y = dataset['Outcome']
```

In [18]:

```
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
```

TRAIN TEST SPLIT

In [21]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.3, stratify=Y, ra
ndom_state=2)
```

In [22]:

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(768, 8) (537, 8) (231, 8)
```

Importance of Preprocessing: Steps like standardization and splitting data correctly improve model performance.

TRAINING THE SVM CLASSIFIER MODEL

SVM is effective for binary classification tasks like this.

In [23]:

```
classifier = svm.SVC(kernel='linear')
```

In [24]:

```
classifier.fit(X_train, Y_train)
```

Out[24]:

▾ SVC i ?

SVC(kernel='linear')

MODEL EVALUATION

In [25]:

```
#ACCURACY SCORE FOR THE TRAINING DATA
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [26]:

```
print('Accuracy score of the training data : ', training_data_accuracy)
```

```
Accuracy score of the training data :  0.7821229050279329
```

In [27]:

```
##ACCURACY SCORE FOR THE TESTING DATA
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

Accuracy on training data: How well the model performs on data it has seen.

Accuracy on test data: How well the model generalizes to unseen data.

In [28]:

```
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the test data :  0.7748917748917749
```

MAKE A PREDICTIVE SYSTEM

In [29]:

```
input_data = (5,166,72,19,175,25.8,0.587,51)

#CHANGEINPUT DATA INTO NUMPY ARRAY
input_data_as_numpy_array = np.asarray(input_data)

#RESHAPING THE ARRAY WE ARE SPLIITTING FOR A INSTANCE
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

#STANDARDIZE 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')
```

```
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person is diabetic
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

CONCLUSION:

CONCLUSION:

This project uses machine learning to predict diabetes based on patient data. The model achieved reasonable accuracy on both training and test data, and can differentiate between diabetic and non-diabetic individuals.

Real-world Applicability: This model can assist in early diabetes detection but should not replace professional medical diagnosis.The dataset typically contains 768 rows and 9 columns, which is sufficient for basic models like Logistic Regression or SVM in an educational or exploratory context.If the dataset is small:1)Use cross-validation to make the most of the available data. 2)Focus on simpler models (e.g., Logistic Regression) that require less data to perform well.

Limitations:

Predicting diabetes is a critical task; even small errors can have serious consequences.

also More advanced techniques, such as **hyperparameter tuning or ensemble models, could improve performance.**

This project showcases the potential of machine learning in healthcare, but accuracy and reliability are essential when dealing with life-threatening conditions.so Always combine machine predictions with expert medical opinions.