

Importing the Dependencies

```
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
```

Data Collection and Analysis

```
#Loading the dataset to pandas dataframe
diabetes_dataset = pd.read_csv("D:\\Machine Learning\\diabetes.csv")
```

```
# printing first 5 data
diabetes_dataset.head()
```

	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

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

```
# number of rows and columns
diabetes_dataset.shape
```

```
(768, 9)
```

```
#Getting staistical measures of data
diabetes_dataset.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin \
--	-------------	---------	---------------	---------------	-----------

count	768.000000	768.000000	768.000000	768.000000
768.000000				
mean	3.845052	120.894531	69.105469	20.536458
79.799479				
std	3.369578	31.972618	19.355807	15.952218
115.244002				
min	0.000000	0.000000	0.000000	0.000000
0.000000				
25%	1.000000	99.000000	62.000000	0.000000
0.000000				
50%	3.000000	117.000000	72.000000	23.000000
30.500000				
75%	6.000000	140.250000	80.000000	32.000000
127.250000				
max	17.000000	199.000000	122.000000	99.000000
846.000000				

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

#To counts diabetes value means 1- 268 person are diabetes and 0-500 person are Non diabetes

```
diabetes_dataset["Outcome"].value_counts()
```

```
0    500
```

```
1    268
```

```
Name: Outcome, dtype: int64
```

#To get mean value

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

	Pregnancies	Glucose	BloodPressure	SkinThickness
Insulin \ Outcome				
0	3.298000	109.980000	68.184000	19.664000
68.792000				
1	4.865672	141.257463	70.824627	22.164179
100.335821				

	BMI	DiabetesPedigreeFunction	Age
Outcome			

```

0      30.304200      0.429734  31.190000
1      35.142537      0.550500  37.067164

# Separating the Data and Labels
X = diabetes_dataset.drop(columns = "Outcome",axis =1)      #Axis = 1
# to drop columns
X.head()

   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

   DiabetesPedigreeFunction  Age
0              0.627      50
1              0.351      31
2              0.672      32
3              0.167      21
4              2.288      33

# Separating the Data and Labels
Y = diabetes_dataset["Outcome"]
Y.head()

0      1
1      0
2      1
3      0
4      1
Name: Outcome, dtype: int64

```

Data Standardization

```

scaler = StandardScaler()

scaler.fit(X)      #fitting data to 0-1 range

StandardScaler()

standardized_data = scaler.transform(X)  #Transforming
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]]

```

```
X = standardized_data
```

```
Y = diabetes_dataset["Outcome"]
```

```
print(X)
```

```
print(Y)
```

```

[[ 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]]

```

```
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

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size
=0.2, stratify =Y, random_state=2) #0.2 = 20% of test Data
print(X.shape,X_train.shape,X_test.shape)
(768, 8) (614, 8) (154, 8)
```

Training the Model

```
classifier = svm.SVC(kernel= "linear")
#Training the Support Vector Machine Classifier
classifier.fit(X_train, Y_train)
SVC(kernel='linear')
```

Model Evaluation

Accuracy

```
# accuracy score on training Data
X_train_prediction = classifier.predict(X_train)
(X_train_prediction)
array([0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
      0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
0,
      0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
      0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
1,
      0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0,
      1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0,
1,
      1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
0,
      1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
0,
      1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
0,
      0,
```

```

1,      1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1,      0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0,
1,      0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
0,      0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
0,      0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0,      0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0,      0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
0,      0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1,
1,      0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1,
0,      0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,      0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
1,      0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
0,      0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0,      0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0,
1,      0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0,
0,      0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
0,      0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0,      0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0,      0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0],
dtype=int64)

```

#Checking Accuracy

```
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
print("Accuracy score of the Training Data : ",
training_data_accuracy)    #Balanced Fitting bcoz train having 78% and
test is also 77%
```

```
Accuracy score of the Training Data : 0.7866449511400652
```

```

# accuracy score on testing Data
X_test_prediction = classifier.predict(X_test)
testing_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print("Accuracy score of the Training Data : ", testing_data_accuracy)
#If Training having high % and Testing having Low % then Overfitting

Accuracy score of the Training Data :  0.7727272727272727

```

Making a Predictive system

```

input_data = (5,166,72,19,175,25.8,0.587,51)    #1    means Diabetic
# input_data = (4,110,92,0,0,37.6,0.191,30)    #0    means Not
Diabetic

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

# reshape the array as we are predicting for one example
input_data_resaped = input_data_as_numpy_array.reshape(1,-1)    #This
will tell model we are not getting result of 768 total value but for
one value

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

C:\Users\kunal\anaconda3\lib\site-packages\sklearn\base.py:420:
UserWarning: X does not have valid feature names, but StandardScaler
was fitted with feature names
  warnings.warn(

```

