DIABETES

```
import numpy as np
import pandas as pd
# first upload file
df=pd.read_csv("/content/diabetes.csv")
df
# print head tail total number of missing values
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Ins
0	6	148	72	35	
1	1	85	66	29	
2	8	183	64	0	
3	1	89	66	23	
4	0	137	40	35	
...	...	...	...	...	
763	10	101	76	48	
764	2	122	70	27	
765	5	121	72	23	
766	1	126	60	0	
767	1	93	70	31	

768 rows × 9 columns

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insul
0	6	148	72	35	
1	1	85	66	29	
2	8	183	64	0	
3	1	89	66	23	

0s completed at 2:48 AM

4	0	137	40	35	1(

```
df.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Ins
763	10	101	76	48	
764	2	122	70	27	
765	5	121	72	23	
766	1	126	60	0	
767	1	93	70	31	

```
df.shape
```

```
(768, 9)
```

```
df.isna().sum
```

```
<bound method NDFrame._add_numeric_operations.<locals>.sum
of      Pregnancies  Glucose  BloodPressure  SkinThickness
Insulin    BMI  \
0           False    False          False          False
False  False
1           False    False          False          False
False  False
2           False    False          False          False
False  False
3           False    False          False          False
False  False
4           False    False          False          False
False  False
..            ...      ...            ...            ...
...    ...
763         False    False          False          False
False  False
764         False    False          False          False
False  False
765         False    False          False          False
False  False
766         False    False          False          False
```

```
False  False
767        False    False          False          False
False  False

     DiabetesPedigreeFunction    Age  Outcome
0                       False  False    False
1                       False  False    False
2                       False  False    False
3                       False  False    False
4                       False  False    False
..                        ...    ...      ...
763                     False  False    False
764                     False  False    False
765                     False  False    False
766                     False  False    False
767                     False  False    False

[768 rows x 9 columns]>
```

```
df.columns #SELECT OUTPUT
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age',
'Outcome'],
      dtype='object')
```

```
df1=df.groupby('Outcome') ['Outcome'].count() #SELECT INPUT
df1
```

```
Outcome
0    500
1    268
Name: Outcome, dtype: int64
```

```
# INPUT ASSIGNED X AND OUTPUT ASSIGNED Y (seperation)
x=df.iloc[:,:-1].values #.values used to print as array
x
y=df.iloc[:,-1].values
y
```

```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1,
1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1
```

```
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1,
1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1,
1, 1, 0, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1,
1, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0,
1, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0,
0, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
1, 0, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0,
1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0
```

```
0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 1, 0, 0,
```

```
# dataset splits in to training data and testing data

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30

x_train
```

```
array([[1.000e+01, 1.080e+02, 6.600e+01, ..., 3.240e+01,
2.720e-01,
        4.200e+01],
       [3.000e+00, 1.080e+02, 6.200e+01, ..., 2.600e+01,
2.230e-01,
        2.500e+01],
       [0.000e+00, 1.370e+02, 4.000e+01, ..., 4.310e+01,
2.288e+00,
        3.300e+01],
       ...,
       [1.000e+00, 1.720e+02, 6.800e+01, ..., 4.240e+01,
7.020e-01,
        2.800e+01],
       [5.000e+00, 1.040e+02, 7.400e+01, ..., 2.880e+01,
1.530e-01,
        4.800e+01],
       [3.000e+00, 1.760e+02, 8.600e+01, ..., 3.330e+01,
1.154e+00,
        5.200e+01]])
```

```
# normalisation method (all values comes under same range)
# here we aply standered scalar normalisation
# equation is =z=(x-u)/s
# u=mean of traing sample
# s=standered deviation of training sample
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x_train)
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
```

```
x_test=scaler.transform(x_test)
x_train
```

```
array([[ 1.86232063e+00, -4.25046400e-01, -1.18948373e-01,
...,
         8.23448265e-02, -6.34719419e-01,  7.69203352e-01],
       [-2.44983177e-01, -4.25046400e-01, -3.18091869e-01,
...,
        -7.20272019e-01, -7.80118162e-01, -6.78046731e-01],
       [-1.14811338e+00,  5.08162057e-01, -1.41338110e+00,
...,
         1.42421987e+00,  5.34740031e+00,  3.01213184e-03],
       ...,
       [-8.47069979e-01,  1.63444813e+00, -1.93766250e-02,
...,
         1.33643365e+00,  6.41228738e-01, -4.22649657e-01],
       [ 3.57103625e-01, -5.53764808e-01,  2.79338619e-01,
...,
        -3.69127149e-01, -9.87830653e-01,  1.27999750e+00],
       [-2.44983177e-01,  1.76316653e+00,  8.76769106e-01,
...,
         1.95212820e-01,  1.98245796e+00,
1.62052693e+00]])
```

```
# create model based on training data
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=7)
# fit input and output
knn.fit(x_train,y_train)
# predict the y value
y_predict=knn.predict(x_test)
y_predict
# apply the value0
print(knn.predict([[5,140,65,30,0,30.6,30.32,25]]))
```

```
[1]
```

```
# to check the performance of the created model(b/w y_predict anc
# based on the confusion matrix
# TP:True positive=====>how many times maticne predict true as tr
# TN:true negative=====>how many times maticne predict false as f
# FP:false positive====>how many times maticne predict false as t
# FN:false negative====>how many times maticne predict true as fa

# 1laccuracy score(the value is more than 70 %)
```

```
# 1]accuracy score(the value is more than 70 %)
      # work based on confution matrix
      # 2*2 matrix  [ tp   fp
      #               fn   tn ]

# acciracy score=       TP+TN
#                      __________
#                     TP+TN+FP+FN


from sklearn.metrics import confusion_matrix,accuracy_score
mat=confusion_matrix(y_predict,y_test)
mat
score=accuracy_score(y_predict,y_test)
score
```

```
0.7748917748917749
```