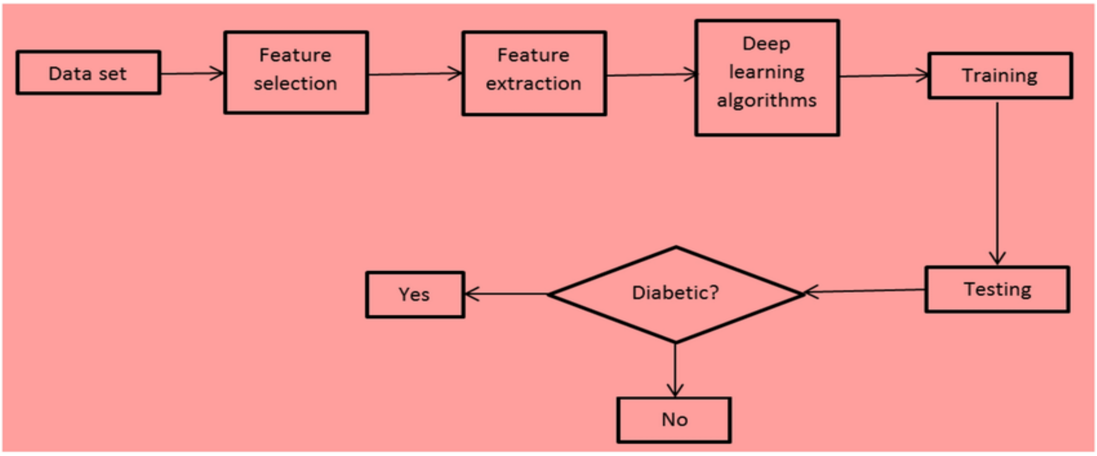
AI BASED DIABETES PREDICTION SYSTEM

Diabetes is a chronic disease that affects millions of people worldwide. Early detection and prevention of diabetes is essential for reducing the risk of complications. Machine learning and deep learning techniques have been shown to be effective in predicting diabetes.



This abstract presents an overview of innovative diabetes prediction techniques using ensemble methods and deep learning. Ensemble methods combine predictions from multiple machine learning models to produce a more accurate and robust prediction. Deep learning models are able to learn complex patterns from data, making them well-suited for diabetes prediction.

Modules

The following modules are typically involved in developing a diabetes prediction system using ensemble methods and deep learning:

- Data collection and preparation: The first step is to collect a dataset of diabetic and non-diabetic patients. The dataset should include relevant features such as age, gender, weight, height, blood glucose levels, and family history of diabetes. The data should be preprocessed to remove outliers and missing values.
- Feature engineering: This module involves creating new features from the existing data that may be more informative for the prediction model. For example, new features could be created to represent the patient's body mass index, waist circumference, or blood pressure.

- Model selection and training: This module involves selecting the appropriate machine learning or deep learning model for the prediction task. The model is then trained on the preprocessed data.
- Model evaluation: The trained model is evaluated on a held-out test set to assess its performance. The evaluation metrics used may include accuracy, precision, recall, and F1 score.
- Ensemble learning: This module involves combining predictions from multiple machine learning models to produce a more accurate and robust prediction. Various ensemble methods such as boosting, bagging, and stacking can be used.
- Deep learning: This module involves using deep learning models to predict diabetes. Deep learning models are able to learn complex patterns from data, making them well-suited for diabetes prediction.

Conclusion

Ensemble methods and deep learning techniques are innovative and effective techniques for diabetes prediction. These techniques have the potential to improve the early detection and prevention of diabetes, which can lead to better health outcomes for patients.

Example

The following example demonstrates how ensemble methods can be used to improve diabetes prediction accuracy:

- Train three separate machine learning models, such as a random forest, support vector machine, and decision tree, to predict diabetes.
- Make predictions for each patient on the test set using each of the three models.
- Combine the predictions from the three models using a weighted average approach. The weights can be assigned based on the performance of each model on the training set.
- The combined prediction is the final prediction for the patient.

This ensemble method is likely to be more accurate than any of the three individual models because it combines the strengths of each model.

Future directions

Future research in diabetes prediction using ensemble methods and deep learning could focus on the following areas:

- Developing new ensemble methods that are specifically tailored for diabetes prediction.
- Developing new deep learning models that can incorporate additional data sources, such as medical images and genomic data.
- Deploying diabetes prediction systems in clinical settings to improve the early detection and prevention of diabetes

Program

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [5]:

```
data=pd.read_csv('../input/diabetes-data-set/diabetes.csv')
data.head(10)
```

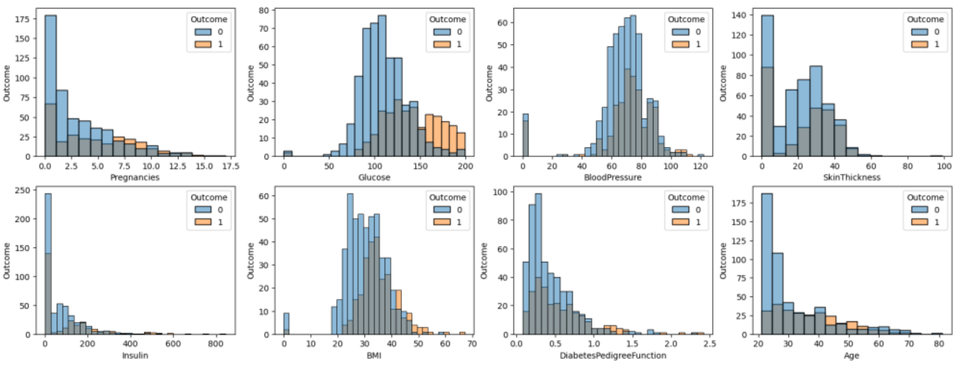
Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes Pedigree Function	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
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1

7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

In [4]:

```
plotnumber=1
featureList=['Pregnancies','Glucose','BloodPressure',
'SkinThickness','Insulin','BMI','DiabetesPedigreeFunc
tion','Age']
plt.figure(figsize=(20,15),facecolor='white')
for i in featureList:
    if(plotnumber<=8):
        plt.subplot(4,4,plotnumber)
        sns.histplot(x=i,data=data,hue='Outcome')
        plt.xlabel(i)
        plt.ylabel('Outcome')
        plotnumber+=1
```



In [5]:

```
data.Glucose.value_counts()
```

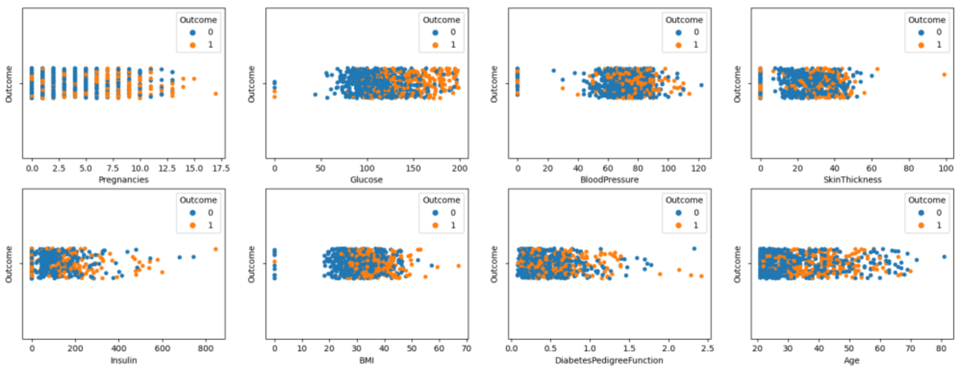
Out[5]:

```
Glucose
99     17
100    17
111    14
129    14
125    14
       ..
191     1
177     1
```

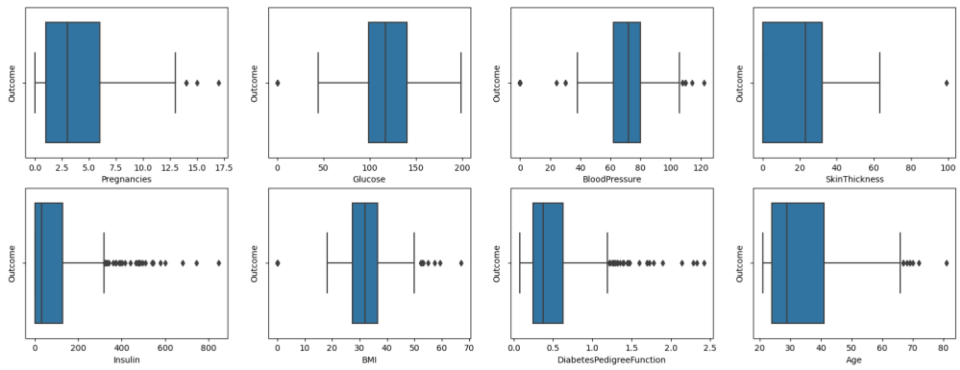
```
44       1
62       1
190      1
Name: count, Length: 136, dtype: int64
```

In [6]:

```
plotnumber=1
featureList=['Pregnancies','Glucose','BloodPressure',
'SkinThickness','Insulin','BMI','DiabetesPedigreeFunc
tion','Age']
plt.figure(figsize=(20,15),facecolor='white')
for i in featureList:
    if(plotnumber<=8):
        plt.subplot(4,4,plotnumber)
        sns.stripplot(x=i,data=data,hue='Outcome')
        plt.xlabel(i)
        plt.ylabel('Outcome')
        plotnumber+=1
```




```
plotnumber=1
featureList=['Pregnancies','Glucose','BloodPressure','SkinThickness','I
nsulin','BMI','DiabetesPedigreeFunction','Age']
plt.figure(figsize=(20,15),facecolor='white')
for i in featureList:
    if(plotnumber<=8):
        plt.subplot(4,4,plotnumber)
        sns.boxplot(x=i,data=data,hue='Outcome')
        plt.xlabel(i)
        plt.ylabel('Outcome')
        plotnumber+=1
```



In [6]:

```
data.loc[data['Glucose']==0,'Glucose']=np.median(data.Glucose)
data.loc[data['BloodPressure']==0,'BloodPressure']=np.median(data.BloodPressure)
data.loc[data['DiabetesPedigreeFunction']==0,'DiabetesPedigreeFunction']=np.median(data.DiabetesPedigreeFunction)
data.loc[data['BMI']==0,'BMI']=np.median(data.BMI)
data.loc[data['Insulin']==0,'Insulin']=np.median(data.Insulin)
data.loc[data['SkinThickness']==0,'SkinThickness']=np.median(data.SkinThickness)
```

In [9]:

```
data.head(20)
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
0	6	148	72	35	30.5	33.6	0.627	50	1
1	1	85	66	29	30.5	26.6	0.351	31	0
2	8	183	64	23	30.5	23.3	0.672	32	1
3	1	89	66	23	94.0	28.1	0.167	21	0
4	0	137	40	35	168.0	43.1	2.288	33	1
5	5	116	74	23	30.5	25.6	0.201	30	0
6	3	78	50	32	88.0	31.0	0.248	26	1
7	10	115	72	23	30.5	35.3	0.134	29	0
8	2	197	70	45	543.0	30.5	0.158	53	1
9	8	125	96	23	30.5	32.0	0.232	54	1
10	4	110	92	23	30.5	37.6	0.191	30	0
11	10	168	74	23	30.5	38.0	0.537	34	1
12	10	139	80	23	30.5	27.1	1.441	57	0

13	1	189	60	23	846.0	30.1	0.398	59	1
14	5	166	72	19	175.0	25.8	0.587	51	1
15	7	100	72	23	30.5	30.0	0.484	32	1
16	0	118	84	47	230.0	45.8	0.551	31	1
17	7	107	74	23	30.5	29.6	0.254	31	1
18	1	103	30	38	83.0	43.3	0.183	33	0
19	1	115	70	30	96.0	34.6	0.529	32	1

In [7]:

```
from sklearn.preprocessing import MinMaxScaler
temp1=['Glucose','BloodPressure','SkinThickness','Insulin','BMI','Diabe
tesPedigreeFunction']
scaling=MinMaxScaler()
data.loc[:,temp1]=scaling.fit_transform(data.loc[:,temp1])
```

In [11]:

```
data.head(10)
```

Out[11]:

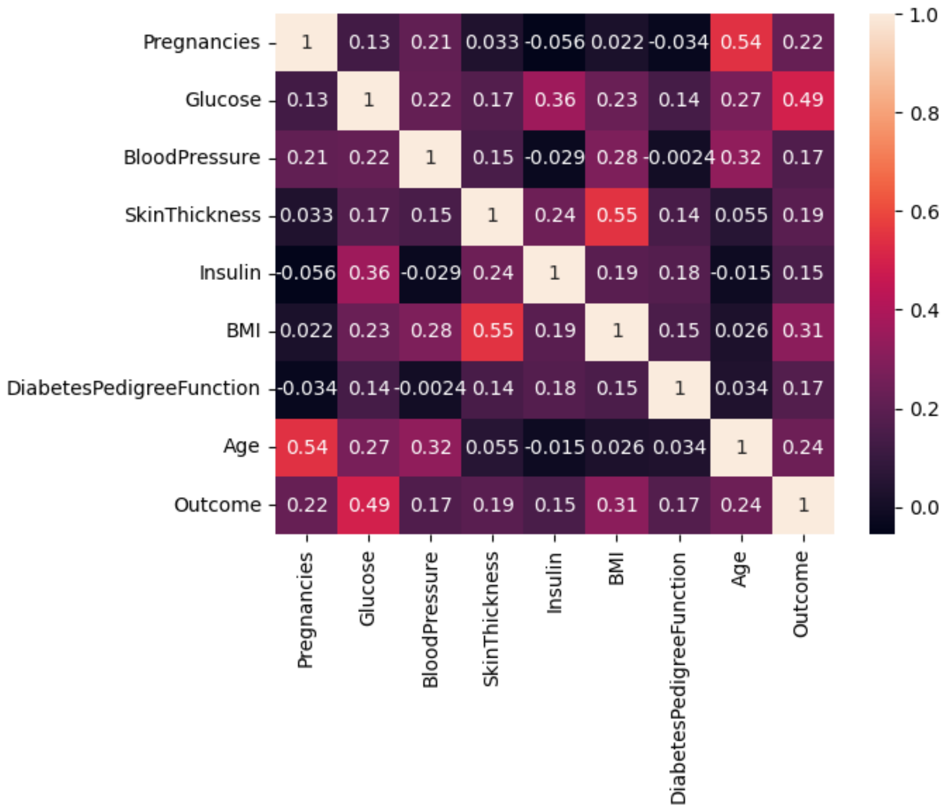
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	0.670968	0.489796	0.304348	0.019832	0.314928	0.234415	50	1
1	1	0.264516	0.428571	0.239130	0.019832	0.171779	0.116567	31	0
2	8	0.896774	0.408163	0.173913	0.019832	0.104294	0.253629	32	1
3	1	0.290323	0.428571	0.173913	0.096154	0.202454	0.038002	21	0
4	0	0.600000	0.163265	0.304348	0.185096	0.509202	0.943638	33	1
5	5	0.464516	0.510204	0.173913	0.019832	0.151329	0.052519	30	0
6	3	0.219355	0.265306	0.271739	0.088942	0.261759	0.072588	26	1
7	10	0.458065	0.489796	0.173913	0.019832	0.349693	0.023911	29	0
8	2	0.987097	0.469388	0.413043	0.635817	0.251534	0.034159	53	1
9	8	0.522581	0.734694	0.173913	0.019832	0.282209	0.065756	54	1

In [12]:

```
heat=data.corr()
sns.heatmap(heat,annot=True)
```

Out[12]:

```
<Axes: >
```



Logistic Reggresion

In [8]:

```
x=data.iloc[:,:-1]
y=data.Outcome
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random
_state=12)
```

In [14]:

```
y_train.value_counts()
```

Out[14]:

```
Outcome
0    401
1    213
Name: count, dtype: int64
```

In [9]:

```
from imblearn.over_sampling import SMOTE
smote=SMOTE()
x_smote,y_smote=smote.fit_resample(x_train,y_train)
```

In [16]:

```
y_smote.value_counts()
```

Out[16]:

```
Outcome
0    401
1    401
Name: count, dtype: int64
```

In [15]:

```
#Building Model
from sklearn.linear_model import LogisticRegression
LR=LogisticRegression()
LR.fit(x_smote,y_smote)
logisticY_predict=LR.predict(x_test)
```

In [16]:

```
from sklearn.metrics import accuracy_score,classification_report
print("Logistic Reggresion----->")
print("Accuracy Score--->",end='')
print(accuracy_score(logisticY_predict,y_test))
print("classification_report---->")
print(classification_report(logisticY_predict,y_test))
```

```
Logistic Reggresion----->
Accuracy Score--->0.7987012987012987
classification_report---->
              precision    recall  f1-score   support

           0       0.82      0.86      0.84        94
           1       0.76      0.70      0.73        60

    accuracy                           0.80       154
   macro avg       0.79      0.78      0.78       154
weighted avg       0.80      0.80      0.80       154
```

Support Vector Machine

In [18]:

```
from sklearn.svm import SVC
model1=SVC()
model1.fit(x_smote,y_smote)
scmY_predit=model1.predict(x_test)
```

In [19]:

```
print("Support Vector Machine----->")
print("Accuracy Score--->",end='')
```

```
print(accuracy_score(scmY_predit,y_test))
print("classification_report---->")
print(classification_report(scmY_predit,y_test))
```

```
Support Vector Machine----->
Accuracy Score--->0.7012987012987013
classification_report---->
              precision    recall  f1-score   support

           0       0.71      0.80      0.75        87
           1       0.69      0.57      0.62        67

    accuracy                           0.70       154
   macro avg       0.70      0.69      0.69       154
weighted avg       0.70      0.70      0.70       154
```

Using a Decision Tree Algorithm Without Hyperparameter Tuning

In [20]:

```
from sklearn.tree import DecisionTreeClassifier
model2=DecisionTreeClassifier()
model2.fit(x_smote,y_smote)
decisionwithotY_predict=model2.predict(x_test)
```

In [21]:

```
print("Decision Tree without hyperpara----->")
print("Accuracy Score--->",end='')
print(accuracy_score(decisionwithotY_predict,y_test))
print("classification_report---->")
print(classification_report(decisionwithotY_predict,y_test))
```

```
Decision Tree without hyperpara----->
Accuracy Score--->0.6298701298701299
classification_report---->
              precision    recall  f1-score   support

           0       0.67      0.73      0.70        90
           1       0.56      0.48      0.52        64

    accuracy                           0.63       154
   macro avg       0.62      0.61      0.61       154
weighted avg       0.62      0.63      0.62       154
```


Using a Decision Tree Algorithm With Hyperparameter Tuning

In [26]:

```
parameter={'criterion':['gini', 'entropy',
'log_loss'],'splitter':['best', 'random']}
from sklearn.model_selection import GridSearchCV
gscv=GridSearchCV(model2,parameter,verbose=2)
gscv.fit(x_smote,y_smote)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV] END ......................criterion=gini, splitter=best; total
time=   0.0s
[CV] END ......................criterion=gini, splitter=best; total
time=   0.0s
[CV] END ......................criterion=gini, splitter=best; total
time=   0.0s
[CV] END ......................criterion=gini, splitter=best; total
time=   0.0s
[CV] END ......................criterion=gini, splitter=best; total
time=   0.0s
[CV] END ....................criterion=gini, splitter=random; total
time=   0.0s
[CV] END ....................criterion=gini, splitter=random; total
time=   0.0s
[CV] END ....................criterion=gini, splitter=random; total
time=   0.0s
[CV] END ....................criterion=gini, splitter=random; total
time=   0.0s
[CV] END ....................criterion=gini, splitter=random; total
time=   0.0s
[CV] END ...................criterion=entropy, splitter=best; total
time=   0.0s
[CV] END ...................criterion=entropy, splitter=best; total
time=   0.0s
[CV] END ...................criterion=entropy, splitter=best; total
time=   0.0s
[CV] END ...................criterion=entropy, splitter=best; total
time=   0.0s
[CV] END ...................criterion=entropy, splitter=best; total
time=   0.0s
[CV] END .................criterion=entropy, splitter=random; total
time=   0.0s
[CV] END .................criterion=entropy, splitter=random; total
time=   0.0s
[CV] END .................criterion=entropy, splitter=random; total
```

```
time=   0.0s
[CV] END .................criterion=entropy, splitter=random; total
time=   0.0s
[CV] END .................criterion=entropy, splitter=random; total
time=   0.0s
[CV] END ..................criterion=log_loss, splitter=best; total
time=   0.0s
[CV] END ..................criterion=log_loss, splitter=best; total
time=   0.0s
[CV] END ..................criterion=log_loss, splitter=best; total
time=   0.0s
[CV] END ..................criterion=log_loss, splitter=best; total
time=   0.0s
[CV] END ..................criterion=log_loss, splitter=best; total
time=   0.0s
[CV] END ................criterion=log_loss, splitter=random; total
time=   0.0s
[CV] END ................criterion=log_loss, splitter=random; total
time=   0.0s
[CV] END ................criterion=log_loss, splitter=random; total
time=   0.0s
[CV] END ................criterion=log_loss, splitter=random; total
time=   0.0s
[CV] END ................criterion=log_loss, splitter=random; total
time=   0.0s
```

Out[26]:

GridSearchCV

estimator: DecisionTreeClassifier

DecisionTreeClassifier

In [27]:

```
gscv.best_params_
```

Out[27]:

```
{'criterion': 'log_loss', 'splitter': 'random'}
```

In [28]:

```
model3=DecisionTreeClassifier(criterion='log_loss',splitter='random')
model3.fit(x_smote,y_smote)
decisionY_predict=model3.predict(x_test)
```

In [29]:

```
print("Decision Tree with hyperpara----->")
print("Accuracy Score--->",end='')
```

```
print(accuracy_score(decisionY_predict,y_test))
print("classification_report---->")
print(classification_report(decisionY_predict,y_test))
```

```
Decision Tree with hyperpara----->
Accuracy Score--->0.7207792207792207
classification_report---->
              precision    recall  f1-score   support

           0       0.79      0.78      0.78       100
           1       0.60      0.61      0.61        54

    accuracy                           0.72       154
   macro avg       0.69      0.70      0.69       154
weighted avg       0.72      0.72      0.72       154
```

Using a Random Forest Algorithm Without Hyperparameter Tuning

In [31]:

```
from sklearn.ensemble import RandomForestClassifier
model4=RandomForestClassifier()
model4.fit(x_smote,y_smote)
randomwithotY_predict=model4.predict(x_test)
```

In [31]:

```
print("Random Forest without hyperpara----->")
print("Accuracy Score--->",end='')
print(accuracy_score(randomwithotY_predict,y_test))
print("classification_report---->")
print(classification_report(randomwithotY_predict,y_test))
```

```
Random Forest without hyperpara----->
Accuracy Score--->0.8051948051948052
classification_report---->
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        91
           1       0.80      0.70      0.75        63

    accuracy                           0.81       154
   macro avg       0.80      0.79      0.79       154
weighted avg       0.80      0.81      0.80       154
```

Using a Random Forest Algorithm With Hyperparameter Tuning

In [33]:

```
parameter={'criterion':['gini', 'entropy',
'log_loss'],'n_estimators':[10,15,20,25,30,35,40,45,50,55,60,65,70,75,8
0,85,90,95,100,105,110,115,120,125]}
from sklearn.model_selection import GridSearchCV
gscv2=GridSearchCV(model4,parameter)
gscv2.fit(x_smote,y_smote)
```

Out[33]:

GridSearchCV

estimator: RandomForestClassifier

RandomForestClassifier

In [34]:

```
gscv2.best_params_
```

Out[34]:

```
{'criterion': 'log_loss', 'n_estimators': 125}
```

In [32]:

```
## with hyperparameter
model5=RandomForestClassifier(criterion='log_loss',n_estimators=125)
model5.fit(x_smote,y_smote)
randomY_predict=model5.predict(x_test)
```

In [36]:

```
print("Random Forest hyperpara----->")
print("Accuracy Score--->",end='')
print(accuracy_score(randomY_predict,y_test))
print("classification_report---->")
print(classification_report(randomY_predict,y_test))
```

```
Random Forest hyperpara----->
Accuracy Score--->0.8181818181818182
classification_report---->
              precision    recall  f1-score   support

           0       0.83      0.88      0.85        93
           1       0.80      0.72      0.76        61

    accuracy                           0.82       154
   macro avg       0.81      0.80      0.81       154
weighted avg       0.82      0.82      0.82       154
```

k-Nearest Neighbors

In [37]:

```
from sklearn.neighbors import KNeighborsClassifier
model6=KNeighborsClassifier()
model6.fit(x_smote,y_smote)
neighbourY_predict=model6.predict(x_test)
```

In [38]:

```
print("KNeighbours----->")
print("Accuracy Score--->",end='')
print(accuracy_score(neighbourY_predict,y_test))
print("classification_report---->")
print(classification_report(neighbourY_predict,y_test))
```

```
KNeighbours----->
Accuracy Score--->0.6818181818181818
classification_report---->
              precision    recall  f1-score   support

           0       0.67      0.80      0.73        82
           1       0.71      0.54      0.61        72

    accuracy                           0.68       154
   macro avg       0.69      0.67      0.67       154
weighted avg       0.69      0.68      0.68       154
```