In [1]:
```python
## Import basic python libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
# Import dataset
data=pd.read_csv(r'C:\Users\anith\OneDrive\Documents\anil.csv')
```

In [3]:
```python
data.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	L
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	

In [4]:
```python
# Dealing the missing values
data.isnull().sum()
```

Out[4]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [5]:
```python
# Remove the missingvalues
data.dropna(inplace=True)
data.isnull().sum()
```

Out[5]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Out[6]: (480, 13)

In [7]:
```
data
```

Out[7]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0
...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0

480 rows × 13 columns

In [8]:
```
#checking the skewness (acceptable range is -5 to +5)
data.skew()
```

Out[8]:
```
ApplicantIncome      6.917027
CoapplicantIncome    5.881622
LoanAmount           2.361437
Loan_Amount_Term    -2.333710
Credit_History      -2.013253
dtype: float64
```

In [9]:
```
data=data.drop(['Loan_ID'],axis=1)
```

In [10]:
```
data
```

Out[10]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmou
1	Male	Yes	1	Graduate	No	4583	1508.0	128
2	Male	Yes	0	Graduate	Yes	3000	0.0	66
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120
4	Male	No	0	Graduate	No	6000	0.0	141
5	Male	Yes	2	Graduate	Yes	5417	4196.0	267
...	...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	2900	0.0	71
610	Male	Yes	3+	Graduate	No	4106	0.0	40
611	Male	Yes	1	Graduate	No	8072	240.0	253
612	Male	Yes	2	Graduate	No	7583	0.0	187
613	Female	No	0	Graduate	Yes	4583	0.0	133

480 rows × 12 columns

In [11]:

```
data['Dependents'].unique()
```

Out[11]:

```
array(['1', '0', '2', '3+'], dtype=object)
```

In [12]:

```
data['Dependents']=data['Dependents'].replace('3+',4)
```

In [13]:

```
data['Dependents'].unique()
```

Out[13]:

```
array(['1', '0', '2', 4], dtype=object)
```

In [14]:

```
data['Dependents']=data['Dependents'].astype('int')
```

In [15]:

```
data['Dependents'].unique()
```

Out[15]:

```
array([1, 0, 2, 4])
```

In [16]:

```
#by using ordinal encoder converting vectors
from sklearn.preprocessing import OrdinalEncoder

encoder = OrdinalEncoder()
data[["Gender",'Married','Education','Self_Employed','Property_Area','Loan_Status']] = e
data.head()
```

Out[16]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount
1	1.0	1.0	1	0.0	0.0	4583	1508.0	128.0
2	1.0	1.0	0	0.0	1.0	3000	0.0	66.0
3	1.0	1.0	0	1.0	0.0	2583	2358.0	120.0
4	1.0	0.0	0	0.0	0.0	6000	0.0	141.0
5	1.0	1.0	2	0.0	1.0	5417	4196.0	267.0

In [17]:

```
#encoding the features are float we convert into integer
data[["Gender",'Married','Education','Self_Employed','Property_Area','Loan_Status']]=dat
```

Out[18]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmou
1	1	1	1	0	0	4583	1508.0	128
2	1	1	0	0	1	3000	0.0	66
3	1	1	0	1	0	2583	2358.0	120
4	1	0	0	0	0	6000	0.0	141
5	1	1	2	0	1	5417	4196.0	267
...	...	...	...	...	...	...	...	
609	0	0	0	0	0	2900	0.0	71
610	1	1	4	0	0	4106	0.0	40
611	1	1	1	0	0	8072	240.0	253
612	1	1	2	0	0	7583	0.0	187
613	0	0	0	0	1	4583	0.0	133

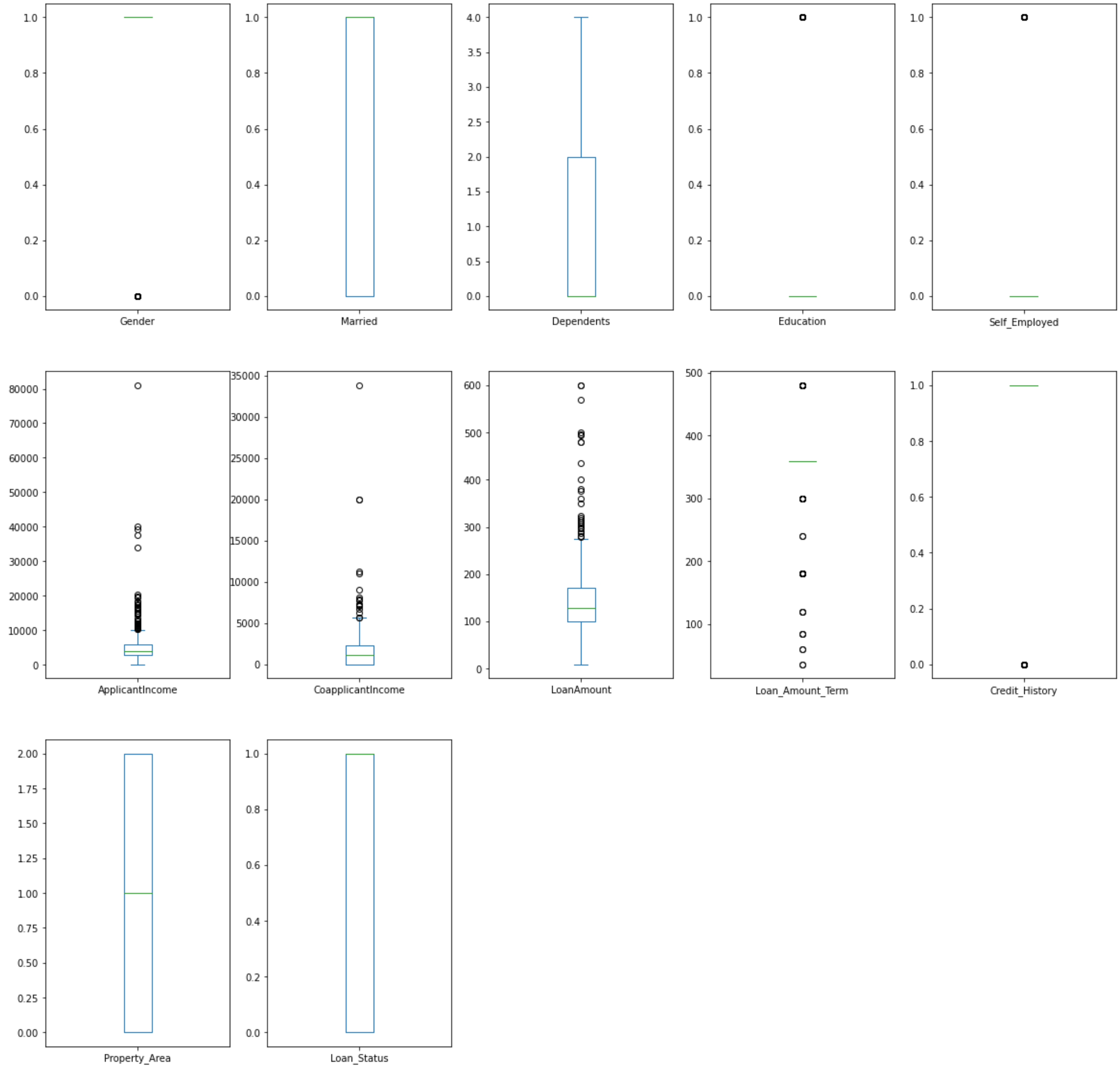
480 rows × 12 columns

In [19]:

```
#seeing the outliers
data.plot(kind='box',subplots=True,layout=(3,5),figsize=(20,20))
```

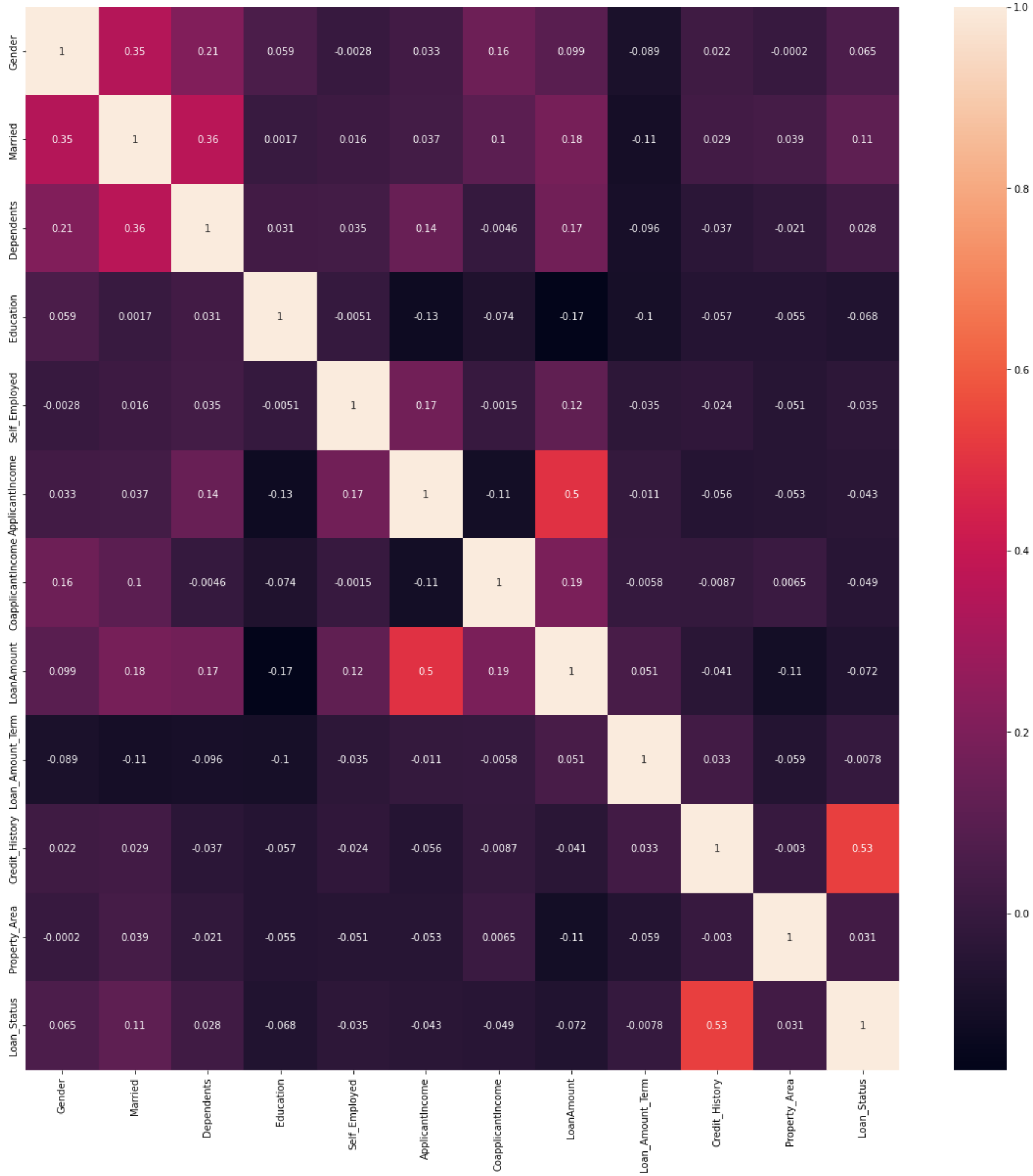
Out[19]:

```
Gender                  AxesSubplot(0.125,0.657941;0.133621x0.222059)
Married              AxesSubplot(0.285345,0.657941;0.133621x0.222059)
Dependents            AxesSubplot(0.44569,0.657941;0.133621x0.222059)
Education            AxesSubplot(0.606034,0.657941;0.133621x0.222059)
Self_Employed        AxesSubplot(0.766379,0.657941;0.133621x0.222059)
ApplicantIncome         AxesSubplot(0.125,0.391471;0.133621x0.222059)
CoapplicantIncome    AxesSubplot(0.285345,0.391471;0.133621x0.222059)
LoanAmount            AxesSubplot(0.44569,0.391471;0.133621x0.222059)
Loan_Amount_Term     AxesSubplot(0.606034,0.391471;0.133621x0.222059)
Credit_History       AxesSubplot(0.766379,0.391471;0.133621x0.222059)
Property_Area              AxesSubplot(0.125,0.125;0.133621x0.222059)
Loan_Status             AxesSubplot(0.285345,0.125;0.133621x0.222059)
dtype: object
```



In [20]:
```python
#checking correlation of dataset
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(),annot=True)
```

Out[20]: <AxesSubplot:>



In [21]:

```
#removing multicollinearity by using vif acceptable range (-10 to 10)
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif['features']=data.columns
vif['vif factor']=[variance_inflation_factor(data.values,i) for i in range(data.shape[1]
vif
```

Out[21]:

	features	vif factor
0	Gender	6.146708
1	Married	3.716146
2	Dependents	1.789241
3	Education	1.283195
4	Self_Employed	1.195656
5	ApplicantIncome	2.752965
6	CoapplicantIncome	1.554019
7	LoanAmount	6.412868
8	Loan_Amount_Term	10.747859
9	Credit_History	8.934985
10	Property_Area	2.610845
11	Loan_Status	4.633473

In [22]:

```
data=data.drop(['Loan_Amount_Term'],axis=1)
data
```

Out[22]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmou
1	1	1	1	0	0	4583	1508.0	128
2	1	1	0	0	1	3000	0.0	66
3	1	1	0	1	0	2583	2358.0	120
4	1	0	0	0	0	6000	0.0	141
5	1	1	2	0	1	5417	4196.0	267
...	...	...	...	...	...	...	...	
609	0	0	0	0	0	2900	0.0	71
610	1	1	4	0	0	4106	0.0	40
611	1	1	1	0	0	8072	240.0	253
612	1	1	2	0	0	7583	0.0	187
613	0	0	0	0	1	4583	0.0	133

480 rows × 11 columns

In [23]:

```
x=data.drop(['Loan_Status'],axis=1)
y=data['Loan_Status']
```

In [24]:

```
# feature scaling
from sklearn.preprocessing import StandardScaler
std_scaler=StandardScaler()
x_scaled=std_scaler.fit_transform(x)
```

In [25]:

```
x_scaled
```

Out[25]:
```
array([[ 0.46719815,  0.73716237,  0.11235219, ..., -0.20808917,
         0.41319694, -1.31886834],
       [ 0.46719815,  0.73716237, -0.70475462, ..., -0.97900085,
         0.41319694,  1.25977445],
       [ 0.46719815,  0.73716237, -0.70475462, ..., -0.30756164,
         0.41319694,  1.25977445],
       ...,
       [ 0.46719815,  0.73716237,  0.11235219, ...,  1.34616826,
         0.41319694,  1.25977445],
       [ 0.46719815,  0.73716237,  0.92945899, ...,  0.52552034,
         0.41319694,  1.25977445],
       [-2.14041943, -1.35655324, -0.70475462, ..., -0.14591887,
        -2.42015348, -0.02954695]])
```

In [26]:
```
# training the model
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.3,random_state=45)
```

In [27]:
```
x_train.shape
```

Out[27]:
```
(336, 10)
```

In [28]:
```
y_train.shape
```

Out[28]:
```
(336,)
```

In [29]:
```
#import varies models and metrics
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn .model_selection import cross_val_score
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_sco
```

In [30]:
```
#testing of accuracy of models
models=[LogisticRegression(),SVC(),RandomForestClassifier(), DecisionTreeClassifier()]
for m in models:
    print(f'{m}:')
    m.fit(x_train,y_train)
    print('Training score:',m.score(x_train,y_train))
    print('Testing score:',m.score(x_test,y_test))
    predm=m.predict(x_test)
    f1score=f1_score(y_test,predm)
    print('f1score:',f1score)
    acrscore=accuracy_score(y_test,predm)
    print('Accuracy score:',acrscore)
    crsv=cross_val_score(m,x_scaled,y,cv=5)
    print('Cross validation mean score:',crsv.mean())
    print("")
    print('**'*5)
    print('\n')
```

```
LogisticRegression():
Training score: 0.8154761904761905
Testing score: 0.7916666666666666
f1score: 0.8648648648648648
Accuracy score: 0.7916666666666666
Cross validation mean score: 0.8020833333333334

**********


SVC():
Training score: 0.8392857142857143
Testing score: 0.7847222222222222
f1score: 0.8597285067873304
Accuracy score: 0.7847222222222222
Cross validation mean score: 0.8104166666666666

**********


RandomForestClassifier():
Training score: 1.0
Testing score: 0.7638888888888888
f1score: 0.8440366972477066
Accuracy score: 0.7638888888888888
Cross validation mean score: 0.79375

**********


DecisionTreeClassifier():
Training score: 1.0
Testing score: 0.75
f1score: 0.826923076923077
Accuracy score: 0.75
Cross validation mean score: 0.7145833333333333

**********
```

by using hyper parameter tuning increase the select model accuracy

In [31]:
```
lr=LogisticRegression()
```

In [32]:
```
param_grid=[{'penalty':['l1','l2','elasticnet','none'],'C':np.logspace(-4,4,20),'solver'
```

In [33]:
```
from sklearn.model_selection import GridSearchCV
```

In [34]:
```
cif=GridSearchCV(lr,param_grid=param_grid,cv=3,verbose=True,n_jobs=-1)
```

In [35]:
```
import warnings
```

In [36]:
```
best_cif=cif.fit(x,y)
```

```
Fitting 3 folds for each of 1600 candidates, totalling 4800 fits
```

In [37]:
```
best_cif.best_estimator_
```

Out[37]:

```
▼ LogisticRegression
LogisticRegression(C=0.08858667904100823, penalty='l1', solver='liblinear')
```

In [38]:

```
print('Accuracy score:',best_cif.score(x,y))
```

```
Accuracy score: 0.80625
```

In [39]:

```
#dumping the model
import joblib
import pickle
```

In [40]:

```
joblib.dump(best_cif,'model.pkl')
model=joblib.load('model.pkl')
model.predict(x_test)
```

Out[40]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```