**HEART\_DISEASE\_DETECTION\_TCR\_INNOVATION**

#NAME= SOUVIK DAS

#CERTIFICATION CODE=TCRIG02R84

#EMAIL=souvik702080@gmail.com

#BATCH= MACHINE LEARNING WITH PYTHON

#PROJECT NAME- LOAN\_PREDICTION\_

**SOURCE CODE**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

data = pd.read\_csv("/content/train\_u6lujuX\_CVtuZ9i (1).csv")

data.head

<bound method NDFrame.head of Loan\_ID Gender Married ... Credit\_History Property\_Area Loan\_Status

0 LP001002 Male No ... 1.0 Urban Y

1 LP001003 Male Yes ... 1.0 Rural N

2 LP001005 Male Yes ... 1.0 Urban Y

3 LP001006 Male Yes ... 1.0 Urban Y

4 LP001008 Male No ... 1.0 Urban Y

.. ... ... ... ... ... ... ...

609 LP002978 Female No ... 1.0 Rural Y

610 LP002979 Male Yes ... 1.0 Rural Y

611 LP002983 Male Yes ... 1.0 Urban Y

612 LP002984 Male Yes ... 1.0 Urban Y

613 LP002990 Female No ... 0.0 Semiurban N

data.info

<bound method DataFrame.info of Loan\_ID Gender Married ... Credit\_History Property\_Area Loan\_Status

0 LP001002 Male No ... 1.0 Urban Y

1 LP001003 Male Yes ... 1.0 Rural N

2 LP001005 Male Yes ... 1.0 Urban Y

3 LP001006 Male Yes ... 1.0 Urban Y

4 LP001008 Male No ... 1.0 Urban Y

.. ... ... ... ... ... ... ...

609 LP002978 Female No ... 1.0 Rural Y

610 LP002979 Male Yes ... 1.0 Rural Y

611 LP002983 Male Yes ... 1.0 Urban Y

612 LP002984 Male Yes ... 1.0 Urban Y

613 LP002990 Female No ... 0.0 Semiurban N

Data Cleaning and filling missing values

data.apply(lambda x: sum(x.isnull()),axis=0)

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

data['Gender'].value\_counts()

Male 489

Female 112

Name: Gender, dtype: int64

data.Gender = data.Gender.fillna('Male')

data['Married'].value\_counts()

Yes 398

No 213

Name: Married, dtype: int64

data.Married = data.Married.fillna('Yes')

data['Dependents'].value\_counts()

0 345

1 102

2 101

3+ 51

Name: Dependents, dtype: int64

data.Dependents = data.Dependents.fillna('0')

data['Self\_Employed'].value\_counts()

No 500

Yes 82

Name: Self\_Employed, dtype: int64

data.Self\_Employed = data.Self\_Employed.fillna('No')

data.LoanAmount = data.LoanAmount.fillna(data.LoanAmount.mean())

data['Loan\_Amount\_Term'].value\_counts()

360.0 512

180.0 44

480.0 15

300.0 13

84.0 4

240.0 4

120.0 3

36.0 2

60.0 2

12.0 1

Name: Loan\_Amount\_Term, dtype: int64

data.Loan\_Amount\_Term = data.Loan\_Amount\_Term.fillna(360.0)

data['Credit\_History'].value\_counts()

1.0 475

0.0 89

Name: Credit\_History, dtype: int64

data.Credit\_History = data.Credit\_History.fillna(1.0)

data.apply(lambda x: sum(x.isnull()),axis=0)

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

data.head()

|  | **Loan\_ID** | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | 146.412162 | 360.0 | 1.0 | Urban | Y |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.000000 | 360.0 | 1.0 | Rural | N |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.000000 | 360.0 | 1.0 | Urban | Y |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.000000 | 360.0 | 1.0 | Urban | Y |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.000000 | 360.0 | 1.0 | Urban | Y |

X = data.iloc[:, 1: 12].values

y = data.iloc[:, 12].values

X

array([['Male', 'No', '0', ..., 360.0, 1.0, 'Urban'],

['Male', 'Yes', '1', ..., 360.0, 1.0, 'Rural'],

['Male', 'Yes', '0', ..., 360.0, 1.0, 'Urban'],

...,

['Male', 'Yes', '1', ..., 360.0, 1.0, 'Urban'],

['Male', 'Yes', '2', ..., 360.0, 1.0, 'Urban'],

['Female', 'No', '0', ..., 360.0, 0.0, 'Semiurban']], dtype=object)

y

array(['Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y',

'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y',

'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y',

'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',

'N', 'N', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N',

'N', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N',

'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',

'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',

'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',

'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N',

'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'N', 'N', 'Y', 'Y',

'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'Y',

'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N',

'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N',

'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y',

'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',

'Y', 'N', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N',

'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',

'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y',

'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'N',

'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',

'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y',

'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N',

'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'Y', 'Y',

'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',

'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',

'N', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',

'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y',

'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',

'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',

'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'N', 'Y',

'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y',

'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',

'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',

'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'N', 'Y', 'N', 'Y', 'Y',

'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',

'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',

'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',

'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',

'N', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'N', 'N',

'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',

'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',

'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y',

'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'N',

'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N',

'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N',

'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',

'Y', 'Y', 'N'], dtype=object)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0)

X\_train

array([['Male', 'Yes', '3+', ..., 360.0, 1.0, 'Rural'],

['Male', 'Yes', '0', ..., 360.0, 1.0, 'Rural'],

['Male', 'Yes', '3+', ..., 180.0, 1.0, 'Rural'],

...,

['Male', 'Yes', '3+', ..., 360.0, 1.0, 'Semiurban'],

['Male', 'Yes', '0', ..., 360.0, 1.0, 'Urban'],

['Female', 'Yes', '0', ..., 360.0, 1.0, 'Semiurban']], dtype=object)

from sklearn.preprocessing import LabelEncoder

labelencoder\_X = LabelEncoder()

for i in range(0, 5):

    X\_train[:,i] = labelencoder\_X.fit\_transform(X\_train[:,i])

X\_train[:,10] = labelencoder\_X.fit\_transform(X\_train[:,10])

labelencoder\_y = LabelEncoder()

y\_train = labelencoder\_y.fit\_transform(y\_train)

X\_train

array([[ 1.25232703e+00, -1.47124819e+00],

[ 8.03573140e-01, 1.17530917e-01],

[ 5.00517279e-01, -2.14474656e+00],

[ 1.88908783e-02, -2.36622879e-01],

[ 4.63728087e-01, -8.34329288e-01],

[ 1.01804227e+00, 1.40671658e-01],

[ 6.46244427e+00, 4.26395753e+00],

[-2.30050297e+00, 1.61896623e+00],

[-6.01605471e-02, -1.03574728e+00],

[-3.12070301e-01, -1.30692410e+00],

[-2.55395441e+00, 8.24108152e-01],

[-2.09654658e-01, -3.56113753e-01],

[ 3.08071630e-03, -1.06345864e+00],

[ 7.04392847e-02, -1.87330109e+00],

[ 1.70019018e+00, -2.46381146e+00],

[-2.17581410e-01, -2.25065808e-01],

[-5.76391821e-01, -4.88154112e-01],

[ 1.04628868e+00, 2.59567921e-01],

[ 2.65790605e-01, 2.57483677e-01],

[ 3.79944595e+00, 1.00893637e+00],

[ 3.68163271e+00, 2.00363062e+00],

[ 4.79476021e-01, 5.52254342e-01],

[ 7.76057640e+00, 4.80406947e+00],

[-1.72743347e+00, 1.95971654e+00],

[-3.35747366e-01, 1.43373858e-01],

[ 3.42521509e-01, -5.13866712e-01],

[ 1.22559869e-02, -7.36399644e-01],

[-5.81712639e-01, 9.36813033e-01],

[-3.52718050e-02, -5.86909608e-02],

[ 9.45717587e-01, -5.07606160e-01],

[-7.57696092e-01, 3.90364185e-01],

[ 1.22540962e+00, -9.02202679e-01],

[ 9.19902547e-01, 6.50090370e-01],

[-1.73344005e+00, 2.07379082e-01],

[ 1.74212916e+00, 2.67134635e-01],

[-1.74690582e+00, 1.74218595e+00],

[ 1.40455715e+00, -9.38653405e-01],

[ 1.20694244e-01, -2.16508100e+00],

[ 2.30794503e-01, -1.67992435e+00],

[ 6.83431357e+00, 2.48758926e+00],

[ 2.30781898e-02, -2.55226711e-01],

[ 3.02413244e+00, 2.40574581e+00],

[-6.72719090e-01, 7.53316515e-01],

[-8.59882241e-01, -6.31601087e-01],

[-3.91899965e-01, -7.75561389e-01],

[-7.05469105e-02, -2.01088683e+00],

[ 1.61335718e+00, 2.50057385e+00],

[ 7.84135440e-01, -2.86288083e-01],

[-2.47968873e+00, 1.42474581e+00],

[-1.66329762e+00, 1.80383022e+00],

[ 6.81204708e-01, 4.43704506e-01],

[ 6.26627926e-01, -2.53302490e+00],

[ 5.06221534e-01, -6.69355148e-01],

[ 2.16474848e+00, 5.72842692e-01],

[-1.53099490e-01, -1.73773219e+00],

[-4.97850192e-01, -4.69810994e-01],

[-1.95749886e-01, -3.99259666e-01],

[ 5.59649811e-01, -8.16984907e-01],

[-9.38248422e-01, 3.61016468e-01],

[-1.78988081e+00, 5.87878247e-01],

[-4.36672324e-01, 1.39141467e+00],

[-6.95326633e-01, -1.40912011e+00],

[-1.74867033e+00, 4.26078755e-01],

[-1.48832177e+00, -4.82606925e-01],

[-1.80701614e+00, 2.69611734e+00],

[-1.00890154e-01, -4.21394426e-01],

[-2.99554536e-01, 7.50302545e-01],

[ 1.41406596e+00, -6.39443786e-01],

[ 1.83829473e+00, -5.45611656e-02],

[ 2.06713708e-01, -9.27331087e-01],

[ 3.11313365e-01, -1.10520243e+00],

[ 7.79962167e-01, -7.59672041e-01],

[ 5.64135479e-02, -1.02873972e+00],

[-1.75684550e+00, 4.40935124e-01],

[ 3.36650436e-01, -1.85826251e+00],

[ 3.21881881e+00, 1.34245917e+00],

[-1.16696925e+00, 5.69294706e-01],

[-1.22311126e+00, 4.06810184e-01],

[-1.09121150e+00, 5.45834359e-01],

[-2.04202912e+00, 1.73555315e+00],

[-1.46425126e-01, -1.28918739e-01],

[-4.20139745e-01, 6.57567589e-01],

[-1.75323797e-01, 2.09113776e-01],

[ 2.01342266e+00, -2.98081722e-01],

[ 2.84700541e-01, -1.77078930e+00],

[-9.54351777e-01, 9.55067659e-01],

[ 1.52594292e+00, -8.31479031e-02],

[-6.59368585e-01, -5.62075576e-01],

[ 1.33855183e+00, 5.27970040e-01],

[-1.38041288e-01, -3.05898612e-01],

[ 2.34690655e+00, 2.72265195e-01],

[-8.11329084e-01, 1.01986164e+00],

[ 2.18401146e+00, -2.45517137e-01],

[ 2.40156457e+00, -7.70528326e-01],

[-1.12279232e+00, 8.56073799e-02],

[ 4.28471642e-01, -4.95920155e-01],

[-4.90654535e-01, -1.14883460e+00],

[ 4.93817306e-01, 5.25352628e-03],

[ 1.03097902e+00, 9.04432211e-01],

[-1.64340827e+00, 4.83402253e-01],

[ 9.62966700e-01, -1.47419379e-01],

[-3.21098240e-01, -2.69486108e-01],

[-7.39895176e-01, -2.68043374e-01],

[ 1.41138895e-01, -1.49321869e-01],

[ 4.18618843e-01, -4.02123334e-03],

[ 7.28380859e-01, 5.23905568e-02],

[-3.44023670e-01, -1.61497753e-01],

[ 3.59436992e-01, -8.46117135e-01],

[-1.34511317e+00, 7.74610633e-01],

[ 1.89992480e+00, -2.63831882e-01],

[-8.59576618e-01, -1.47424845e+00],

[-6.95707451e-02, -1.99853659e+00],

[-1.66457100e+00, 1.40629826e+00],

[ 4.68272643e-01, -3.47060145e-01],

[ 4.08535691e-02, -8.31569708e-01],

[-1.26045502e+00, -8.54142515e-02],

[ 4.85058336e-01, -4.74798373e-01],

[-3.55846859e-01, 1.38400786e+00],

[-7.70242761e-01, 8.65280847e-01],

[-1.68699337e+00, 2.58982860e-01],

[ 1.51158302e+00, -5.85115781e-01],

[-1.96505162e-02, -6.85527701e-01],

[ 1.63520483e+00, 2.65019978e-02],

[ 6.48122946e-01, -6.77388197e-01],

[ 4.47007937e-01, -1.51425710e-01],

[-1.33515667e-01, -6.54055264e-01],

[ 3.78707059e+00, 5.19607303e-01],

[ 1.12657288e+00, -3.49682711e-01],

[-2.47523241e+00, 1.53384397e+00],

[-2.11055798e+00, 1.83902317e+00],

[-4.16683963e-01, -6.11053712e-01],

[-1.17515837e+00, 7.32390819e-01],

[-5.94068760e-02, -1.38032896e-01],

[-8.91624828e-01, 8.83083676e-01],

[-2.13999130e+00, 3.25513957e-01],

[-3.14326261e-02, -1.03443075e-01],

[-9.94587489e-01, 7.08380080e-01],

[-1.04324360e+00, 5.33218990e-01],

[-8.83590681e-01, 1.05970407e+00],

[-1.48918632e+00, -4.09151823e-01],

[ 6.70635648e-01, 3.18871663e-01],

[ 4.17766813e+00, 3.90503262e+00],

[ 2.96884076e+00, -6.22932172e-01],

[ 3.53062342e-01, -1.85791801e+00],

[ 1.45117235e+00, -1.94399161e+00],

[ 2.25424973e+00, 5.83019278e-01],

[-1.71315099e+00, 2.11370242e+00],

[-1.56729487e-01, 1.33186016e-01],

[ 8.15310359e-01, -2.12853490e+00],

[ 4.68596093e-01, -9.02605744e-01],

[ 1.09278489e+00, -8.75316369e-01],

[-3.73293037e-02, 7.76569207e-01],

[-3.34998600e-01, 1.27981869e+00],

[ 1.18844683e-01, -2.26310513e+00],

[ 7.69519111e-01, -1.11643301e-01],

[-4.77420688e-01, -9.06449050e-01],

[-1.13984447e+00, 7.74938795e-01],

[-1.00386697e+00, -1.54477616e+00],

[ 1.18961449e+00, 2.63185790e-01],

[ 2.90675246e-01, -1.20562541e+00],

[ 5.18437808e-01, -4.43377039e-01],

[ 2.61613585e-01, -1.11293770e+00],

[-2.76266287e+00, 4.75748905e-01],

[-1.74774217e+00, 1.08552899e+00],

[ 1.49396362e-01, -1.91899497e+00],

[-1.56169715e+00, -3.79837400e-01],

[-2.17238226e+00, 1.66502882e+00],

[ 5.38417684e-01, -2.58920315e-01],

[ 5.21978014e-01, -3.33255042e-01],

[-1.77170753e+00, -1.24727108e+00],

[-1.26750831e+00, 7.36699177e-01],

[-1.73955916e+00, 3.78261373e-01],

[ 8.53617863e-01, 2.60357618e+00],

[-5.77123210e-01, -4.33403521e-01],

[-1.84826272e+00, -8.42449629e-01],

[ 3.67913460e-03, -2.54933723e-01],

[-3.99185199e-01, -2.58677355e-01],

[-5.17222089e-02, 1.27619640e+00],

[-1.36178016e+00, -4.73196272e-01],

[-3.23847115e-03, -9.29857313e-01],

[-2.77856417e-01, -3.18978038e-01],

[-3.12953056e-01, 1.37528175e+00],

[-5.21121870e-02, -3.11975687e-01],

[-1.58137950e-01, -6.61956123e-01],

[-2.97381505e-01, 1.68661928e+00],

[-2.55612321e+00, 1.37909352e+00],

[ 4.83259544e-01, -1.24473433e+00],

[-3.79036366e-01, 1.62279721e+00],

[-2.48892232e+00, 1.53863261e+00],

[ 3.30235454e-01, -9.25784373e-01],

[-1.32263420e+00, 6.65975999e-01],

[ 1.61864661e-01, -8.16534496e-02],

[-1.38480862e+00, 6.59842305e-01],

[-2.01321484e-01, 1.51008408e+00],

[ 2.59988384e-01, -7.03979340e-01],

[-1.79589413e+00, 1.86666449e+00],

[-2.38035402e-01, -1.15364214e-01],

[-6.38838072e-01, 6.90986037e-01],

[ 4.26706541e-01, -1.95787868e+00],

[ 2.58935081e-01, -2.72927944e+00],

[ 7.36745382e-01, -1.15988368e+00],

[-1.49525217e-01, -2.53981024e-01],

[-3.25833799e-01, 8.96337234e-01],

[-1.16078975e+00, 1.08334116e+00],

[ 5.75191071e-01, -1.16237817e+00],

[-6.56593698e-01, 4.93200810e-01],

[ 1.46377048e+00, -6.20920398e-01],

[ 7.82976275e-01, -1.40236651e+00],

[ 4.27338124e-01, -3.55217661e-01],

[-1.40074927e-01, -2.44647125e-01],

[ 4.24484500e-02, -1.86803752e+00],

[-1.42323234e+00, 6.34336557e-01],

[ 1.04650469e+00, 1.04755109e-01],

[ 4.72665053e-01, -2.76013623e-01],

[ 8.37464338e-01, 4.53696441e-01],

[-2.47475865e+00, 6.13052232e-01],

[ 5.13941938e-01, 2.10705234e+00],

[ 1.69362783e+00, 7.76472053e-01],

[-4.17474209e-02, -6.04808518e-02],

[ 4.23098815e+00, 1.05938786e+00],

[-4.53377397e-01, -2.29839437e+00],

[-8.31343505e-01, 7.85722965e-01],

[ 3.27549790e-01, -2.27200174e+00],

[-1.33908383e+00, -1.70830112e+00],

[ 8.14734089e-01, -3.29086271e-01],

[-8.74701939e-02, -2.66176029e-01],

[-5.69393782e-01, -1.22600962e+00],

[ 8.04186771e-02, -1.83275512e+00],

[ 2.29634188e-01, -5.57574511e-01],

[ 5.23912475e-01, -2.06475064e+00],

[ 1.75470778e+00, 6.61236240e-01],

[ 4.19952520e-01, -5.20895455e-01],

[ 4.11902378e-01, 1.35194041e+00],

[-3.19890996e-01, -4.89175402e-01],

[-6.66108536e-01, -1.28694644e+00],

[-9.32094376e-01, -1.18830891e+00],

[ 3.67742022e-01, 1.88998508e-01],

[ 4.53603092e-01, 3.28308164e-02],

[ 5.01870147e-01, -2.93498264e-01],

[-8.84533067e-01, 7.92267320e-01],

[-1.07806190e+00, 8.41849373e-01],

[ 1.24545755e+00, -7.55016219e-01],

[-1.75264045e+00, -6.65616265e-01],

[ 4.22938293e-01, 2.21853974e+00],

[-1.53033103e+00, 5.75317994e-01],

[ 1.68907017e-01, -9.47289284e-01],

[-1.34750820e+00, -8.34278859e-01],

[ 1.20315299e+00, -1.66135948e+00],

[ 1.51643295e+00, -5.32645854e-01],

[ 1.44294987e+00, 6.43157670e-01],

[ 1.01447065e+00, 6.43891657e-01],

[-1.15716839e+00, 4.26184418e-01],

[-1.66070297e-01, -3.29350591e-01],

[-8.94911497e-01, 1.01938738e+00],

[-2.48277530e+00, 1.51794808e+00],

[-1.23688440e+00, 1.23731989e+00],

[-1.90015392e-01, 4.35180647e-01],

[-1.49174326e+00, 1.57869477e+00],

[-1.84709217e+00, 1.38118903e+00],

[ 9.65291251e-01, 6.56112061e-01],

[ 9.53369627e-01, 3.84727934e-01],

[ 8.06937767e-01, -1.82611965e-02],

[-2.67069498e-01, -7.75812076e-01],

[-2.06746313e+00, 1.66108515e+00],

[ 3.32278388e-01, -2.27045984e+00],

[-5.62772640e-02, -1.42215743e+00],

[-1.19725387e+00, 8.86182117e-01],

[ 1.16999488e+00, 9.80330261e-01],

[-1.03470966e+00, 8.93349189e-01],

[-1.67641851e-01, -1.63137286e-01],

[-6.49882442e-01, -1.47336308e+00],

[-4.20968059e-02, 1.54063224e+00],

[-1.18505431e+00, -1.12660501e+00],

[ 5.01209544e-01, 1.38595867e+00],

[-1.41903332e+00, 5.67051413e-01],

[-3.20417872e-02, -1.94635676e+00],

[-7.09282384e-01, 2.29253099e+00],

[ 1.27362527e+00, -1.51790270e+00],

[ 3.85383922e+00, 1.60820958e+00],

[ 2.65155698e-01, -1.88012927e+00],

[-1.40997680e+00, 2.37248605e+00],

[-3.30702907e-01, -2.43862327e+00],

[ 1.84704822e+00, 2.87820408e-01],

[ 6.42159642e-01, -1.42331948e+00],

[-2.67543890e+00, 1.44116449e+00],

[-5.54959176e-01, -4.89296712e-01],

[-1.71835796e+00, 1.64756528e+00],

[ 4.72872151e-01, -8.06157241e-01],

[-8.20827597e-02, -1.88896060e-01],

[ 2.35641959e-01, -1.76592494e+00],

[-1.72593815e+00, 8.13521517e-01],

[ 3.17877522e-02, 1.24802026e+00],

[-9.73276448e-01, 6.21664533e-01],

[-2.42836968e-01, -4.01449502e-01],

[-1.01198991e+00, 8.01509799e-01],

[ 7.68077014e-01, -3.67520831e-02],

[-1.82522227e+00, 1.36652655e+00],

[-1.84086458e+00, 1.27881557e+00],

[-2.90479012e-01, -6.60307883e-01],

[ 1.28752614e-01, -5.01719622e-01],

[-1.82111241e+00, -1.58207642e+00],

[-6.07243065e-01, -1.35818232e+00],

[ 1.83247163e+00, 1.04741734e+00],

[-2.14874083e+00, 1.63808880e+00],

[-1.39327229e-02, 3.00780547e-01],

[-1.38196012e+00, 1.91207629e+00],

[ 5.57177801e-01, -2.43063394e-01],

[ 6.56447920e-01, -1.94749553e+00],

[ 9.73328980e-01, -3.41483169e-01],

[-1.20503876e+00, 8.00798805e-01],

[ 6.96103696e-01, -6.22518073e-01],

[-2.82757237e-01, -4.02305830e-01],

[ 3.72055049e-02, -6.79078729e-01],

[-2.15924218e-02, -8.17589933e-01],

[-3.19711558e-01, -1.02366357e+00],

[-2.35565766e-01, 9.34248452e-01],

[-2.05122466e+00, 1.74946782e+00],

[-1.96739822e+00, 1.84347892e+00],

[ 2.92176017e-01, -9.93798082e-01],

[-6.72014349e-02, -2.39484348e-01],

[ 7.08299514e-03, -2.03295803e+00],

[-4.05857495e-01, -1.88822114e-01],

[ 8.32461213e-01, -8.04570008e-02],

[-1.79384547e+00, 2.96140774e-01],

[ 7.31321446e-02, 1.18962809e-01],

[-1.18996555e+00, 7.06279804e-01],

[-7.13639519e-01, 2.47179154e+00],

[ 4.38672449e-02, -1.04028928e+00],

[-3.83254488e-01, -1.90601920e+00],

[-2.27473691e-01, -1.49982671e-01],

[ 3.89829538e+00, 4.77941948e+00],

[ 2.64380959e-01, -1.77150466e+00],

[-1.07709429e-01, -1.04908586e-01],

[-9.92130007e-01, 2.85870754e+00],

[ 6.22525331e-01, -8.04973372e-01],

[-3.35303488e-01, 1.55115213e+00],

[ 9.23228577e-03, 5.67719310e-01],

[ 5.61515435e-01, -1.65165290e-01],

[-2.08726875e-01, -1.01743547e+00],

[ 4.86768611e-01, -8.49166790e-01],

[-1.23935681e+00, 1.96574890e-02],

[-2.31638773e+00, 5.99066889e-01],

[-8.64538917e-01, 1.02448641e+00],

[ 2.04833353e+00, 1.68540273e+00],

[ 1.19205766e+00, -2.76800456e-01],

[-1.50951473e+00, -2.89943939e-01],

[-4.63335333e-01, -1.57903067e-01],

[ 1.25554479e+00, -4.08144906e-01],

[-2.53412470e-01, -3.52066561e-01],

[ 1.19507062e+00, -1.18217364e-01],

[-1.54808509e+00, 3.86034013e-01],

[-6.47132270e-02, 3.15676012e-02],

[ 2.72701791e+00, 1.93406883e-01],

[-1.82328104e-01, -1.51055334e+00],

[-2.95482447e-01, -1.19907640e+00],

[-1.45086572e+00, -4.37283692e-01],

[ 1.64868695e+00, -1.30647673e+00],

[-1.71061368e-01, 2.28549652e-01],

[-4.87259924e-01, 2.45685440e+00],

[-6.38240436e-01, 2.75133961e+00],

[ 5.91301088e-01, -5.35102426e-01],

[-7.40192637e-02, -3.01205350e-01],

[ 7.89278005e-01, -1.10911656e+00],

[ 6.00510308e-01, -1.12407635e+00],

[-4.71921278e-01, -3.92891937e-01],

[-1.02458587e-01, -1.14958901e+00],

[-7.64272151e-01, -4.42323550e-01],

[-1.92308673e-01, -3.01630889e-01],

[ 3.79628259e-01, 1.14856108e-01],

[-5.64634929e-02, -2.58395634e-01],

[ 1.45776872e+00, -1.42441061e-01],

[ 1.53690786e+00, 2.15510893e-01],

[ 4.80566995e-01, -7.02988338e-01],

[-8.13040304e-01, 7.51668297e-01],

[ 2.01043318e+00, 7.71123989e-02],

[-2.30128028e+00, -7.43368523e-01],

[ 3.00869892e+00, 1.88181423e+00],

[ 2.49231693e-01, 2.15305732e-01],

[-1.55812298e+00, -4.76967491e-01],

[ 7.19995598e-01, 2.30759240e-01],

[-3.56678250e-01, -1.40107245e+00],

[-2.17811871e-01, -1.66619326e+00],

[-1.56174852e+00, -2.97209908e-01],

[-7.13183515e-01, 6.84745728e-01],

[-2.00525031e+00, 1.78194216e+00],

[ 4.00918392e-01, -1.33287272e-01],

[-6.83350580e-01, -1.02573686e+00],

[ 4.27033032e+00, 2.12706443e-01],

[-1.54264757e-01, -4.29704055e-01],

[ 8.25444773e-01, -6.08945535e-01],

[-9.56378675e-01, 1.02488432e+00],

[ 1.49935450e+00, -3.21118557e-01],

[-1.40438965e+00, 3.12737231e-01],

[-6.16839583e-01, 2.23136085e-02],

[-1.58283046e-01, -7.77537661e-01],

[-3.82975502e-01, -4.35019110e-01],

[-1.66309873e-02, -1.64420481e+00],

[-6.31492906e-01, 1.24258145e-01],

[ 3.42461671e-03, -2.54705871e-01],

[-2.26387022e+00, 1.83340424e+00],

[ 3.43757006e+00, -5.95242632e-01],

[ 6.37827205e-01, -1.17075260e+00],

[ 1.54768270e-01, -1.34108219e-01],

[ 9.52171211e-01, -4.12101079e-01],

[-5.94521668e-01, -4.24688120e-01],

[ 2.92532577e+00, 5.15354800e-01],

[ 1.56142632e+00, -8.75631785e-01],

[-2.61723939e-01, -9.09252656e-01],

[-5.10329717e-01, 1.03322634e+00]])

y\_train

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

0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1,

0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,

0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,

1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1,

1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0,

0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1,

1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,

0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1,

1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1,

1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1,

1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,

1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,

1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1,

1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1,

1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,

0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,

1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1])

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder\_X = LabelEncoder()

for i in range(0, 5):

    X\_test[:,i] = labelencoder\_X.fit\_transform(X\_test[:,i])

X\_test[:,10] = labelencoder\_X.fit\_transform(X\_test[:,10])

labelencoder\_y = LabelEncoder()

y\_test = labelencoder\_y.fit\_transform(y\_test)

X\_test

array([[1, 0, 0, ..., 360.0, 1.0, 1],

[0, 0, 0, ..., 360.0, 1.0, 1],

[1, 1, 0, ..., 360.0, 1.0, 2],

...,

[1, 1, 0, ..., 180.0, 1.0, 0],

[1, 1, 2, ..., 180.0, 0.0, 2],

[1, 1, 0, ..., 360.0, 1.0, 0]], dtype=object)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.fit\_transform(X\_test)

Applying PCA

from sklearn.decomposition import PCA

pca = PCA(n\_components = 2)

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.fit\_transform(X\_test)

explained\_variance = pca.explained\_variance\_ratio\_

Classification Algorithms

Logistic Regression

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

y\_pred

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1])

from sklearn import metrics

print('The accuracy of Logistic Regression is: ', metrics.accuracy\_score(y\_pred, y\_test))

The accuracy of Logistic Regression is: 0.7073170731707317

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

cm

array([[ 0, 60],

[ 0, 145]])

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Training set)')

plt.xlabel('PC1')

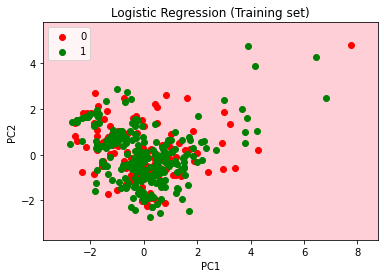
plt.ylabel('PC2')

plt.legend()

plt.show()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with \*x\* & \*y\*. Please use the \*color\* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with \*x\* & \*y\*. Please use the \*color\* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.



from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Test set)')

plt.xlabel('PC1')

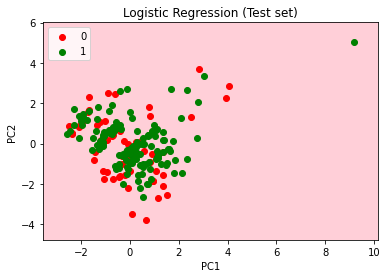
plt.ylabel('PC2')

plt.legend()

plt.show()

c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with \*x\* & \*y\*. Please use the \*color\* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with \*x\* & \*y\*. Please use the \*color\* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.



K-NN

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

classifier.fit(X\_train, y\_train)

KNeighborsClassifier()

y\_pred = classifier.predict(X\_test)

y\_pred

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

1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,

1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 1, 1, 1, 1, 0, 0, 1,

1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,

1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 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,

0, 0, 1, 1, 0, 1, 1])

from sklearn import metrics

print('The accuracy of KNN is: ', metrics.accuracy\_score(y\_pred, y\_test))

The accuracy of KNN is: 0.6292682926829268

from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

[[ 11 49]

[ 27 118]]

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

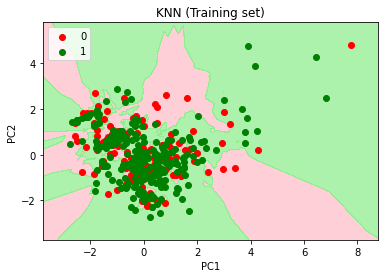
plt.title('KNN (Training set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()



from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

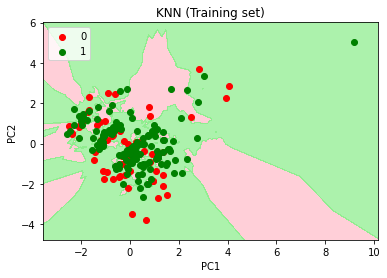
                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('KNN (Training set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()



SVM

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

classifier.fit(X\_train, y\_train)

SVC(kernel='linear', random\_state=0)

y\_pred = classifier.predict(X\_test)

y\_pred

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1])

from sklearn import metrics

print('The accuracy of SVM is: ', metrics.accuracy\_score(y\_pred, y\_test))

The accuracy of SVM is: 0.7073170731707317

from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

[[ 0 60]

[ 0 145]]

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

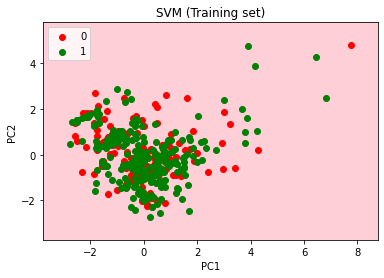
plt.title('SVM (Training set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()



from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

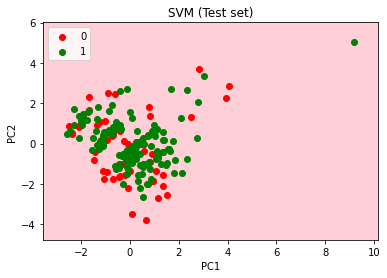
plt.title('SVM (Test set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()



Naive Bayes

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

GaussianNB()

y\_pred = classifier.predict(X\_test)

y\_pred

y\_pred

from sklearn import metrics

print('The accuracy of Naive Bayes is: ', metrics.accuracy\_score(y\_pred, y\_test))

The accuracy of Naive Bayes is: 0.7121951219512195

from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

[[ 3 57]

[ 2 143]]

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

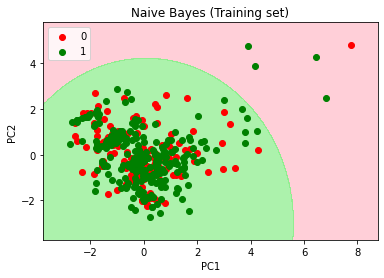
plt.title('Naive Bayes (Training set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()



from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

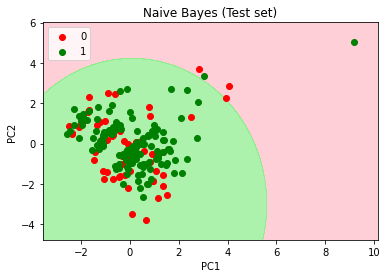
plt.title('Naive Bayes (Test set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()



Decision Tree Classification

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy', random\_state = 0)

classifier.fit(X\_train, y\_train)

DecisionTreeClassifier(criterion='entropy', random\_state=0)

y\_pred = classifier.predict(X\_test)

y\_pred

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

1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1,

1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,

1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,

1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,

0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,

0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,

0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0,

0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0,

1, 0, 1, 1, 0, 1, 1])

from sklearn import metrics

print('The accuracy of Decision Tree Classifier is: ', metrics.accuracy\_score(y\_pred, y\_test))

The accuracy of Decision Tree Classifier is: 0.5365853658536586

from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

[[20 40]

[55 90]]

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

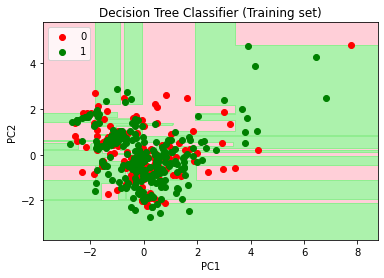
plt.title('Decision Tree Classifier (Training set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()



from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

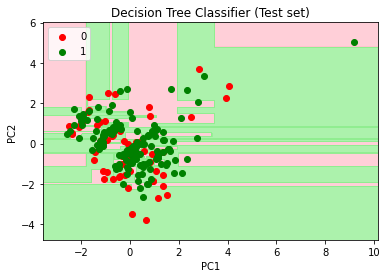
plt.title('Decision Tree Classifier (Test set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()



Random Forest Classification

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy', random\_state = 0)

classifier.fit(X\_train, y\_train)

RandomForestClassifier(criterion='entropy', n\_estimators=10, random\_state=0)

y\_pred = classifier.predict(X\_test)

y\_pred

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

1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,

1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,

1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0,

1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1,

0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1,

0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0,

0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0,

1, 0, 1, 1, 1, 1, 1])

from sklearn import metrics

print('The accuracy of Random Forest Classification is: ', metrics.accuracy\_score(y\_pred, y\_test))

The accuracy of Random Forest Classification is: 0.5853658536585366

from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

[[22 38]

[47 98]]

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

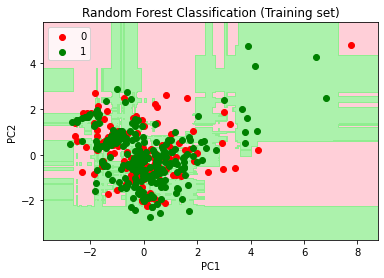
plt.title('Random Forest Classification (Training set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()



from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

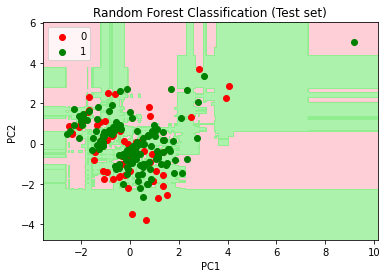
plt.title('Random Forest Classification (Test set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()



Results:

The accuracy of Logistic Regression is: 70.73 %

The accuracy of KNN is: 62.92 %

The accuracy of SVM is: 70.73 %

The accuracy of Naive Bayes is: 71.21 %

The accuracy of Decision Tree Classifier is: 53.63 %

The accuracy of Random Forest Classification is: 58.53 %