NAME= SOUVIK DAS

CERTIFICATION CODE=TCRIG02R84

EMAIL=souvik702080@gmail.com

BATCH= MACHINE LEARNING WITH PYTHON PROJECT NAME- LOAN\_PREDICTION\_

GROUP-SINGLE

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 Prope

1. LP001002 Male No ... 1.0 Urban Y
2. LP001003 Male Yes ... 1.0 Rural N
3. LP001005 Male Yes ... 1.0 Urban Y
4. LP001006 Male Yes ... 1.0 Urban Y
5. LP001008 Male No ... 1.0 Urban Y .. ... ... ... ... ... ... ...
6. LP002978 Female No ... 1.0 Rural Y
7. LP002979 Male Yes ... 1.0 Rural Y
8. LP002983 Male Yes ... 1.0 Urban Y
9. LP002984 Male Yes ... 1.0 Urban Y
10. LP002990 Female No ... 0.0 Semiurban N

[614 rows x 13 columns]>

data.info

<bound method DataFrame.info of Loan\_ID Gender Married ... Credit\_History Pro

1. LP001002 Male No ... 1.0 Urban Y
2. LP001003 Male Yes ... 1.0 Rural N
3. LP001005 Male Yes ... 1.0 Urban Y
4. LP001006 Male Yes ... 1.0 Urban Y
5. LP001008 Male No ... 1.0 Urban Y .. ... ... ... ... ... ... ...
6. LP002978 Female No ... 1.0 Rural Y
7. LP002979 Male Yes ... 1.0 Rural Y
8. LP002983 Male Yes ... 1.0 Urban Y
9. LP002984 Male Yes ... 1.0 Urban Y
10. LP002990 Female No ... 0.0 Semiurban N

[614 rows x 13 columns]>

Data Cleaning and lling 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()

1. 345
2. 102
3. 101

3+ 51

Name: Dependents, dtype: int64

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

d t ['S lf l d'] l t ()

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 ApplicantIncom**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | LP001002 | Male | No | 0 | Graduate | No | 584 |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 458 |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 300 |
| **3** | LP001006 | Male | Yes | 0 | Not  Graduate | No | 258 |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 600 |

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 =

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

[-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],

[ 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],

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\_

Classi cation Algorithms

Logistic Regression

from sklearn.linear\_model import LogisticRegression classifier = LogisticRegression(random\_state = 0) classifier.fit(X\_train, y\_train)

LogisticRegression(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 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

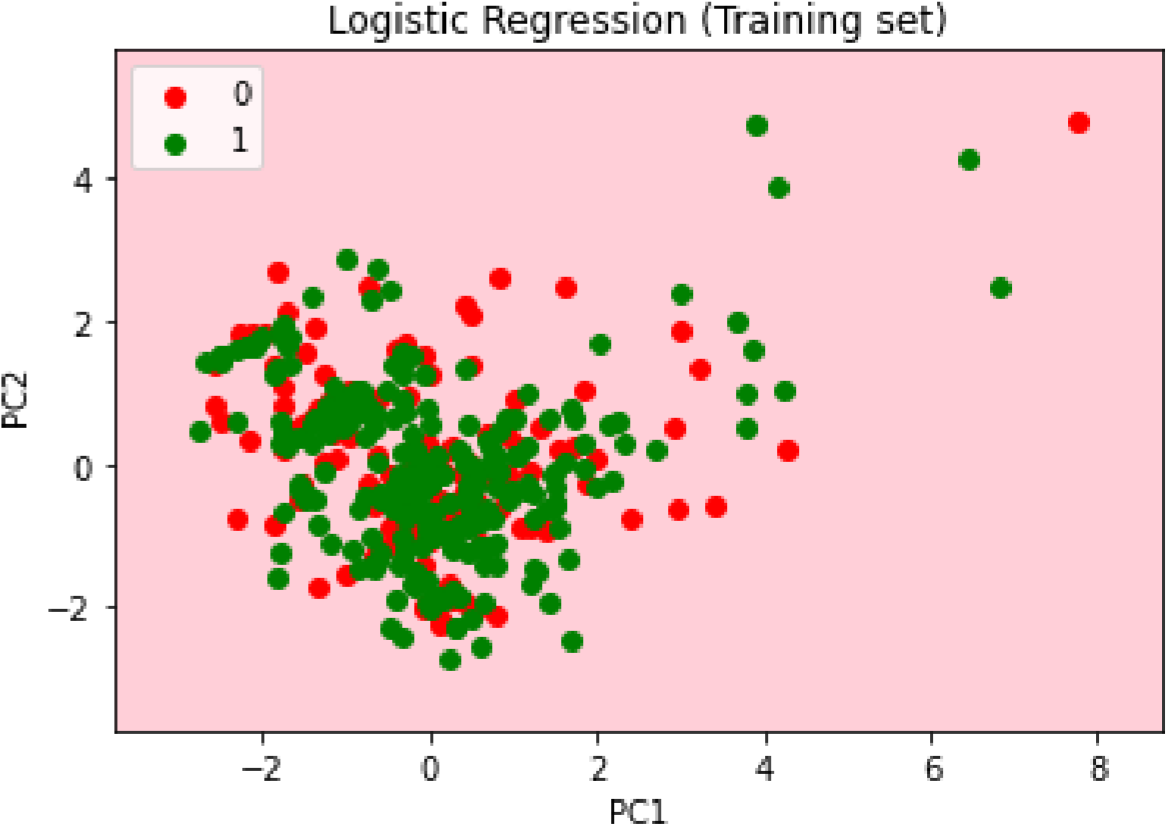
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s alpha = 0.75, cmap = ListedColormap(('pink', '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 avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



from matplotlib.colors import ListedColormap

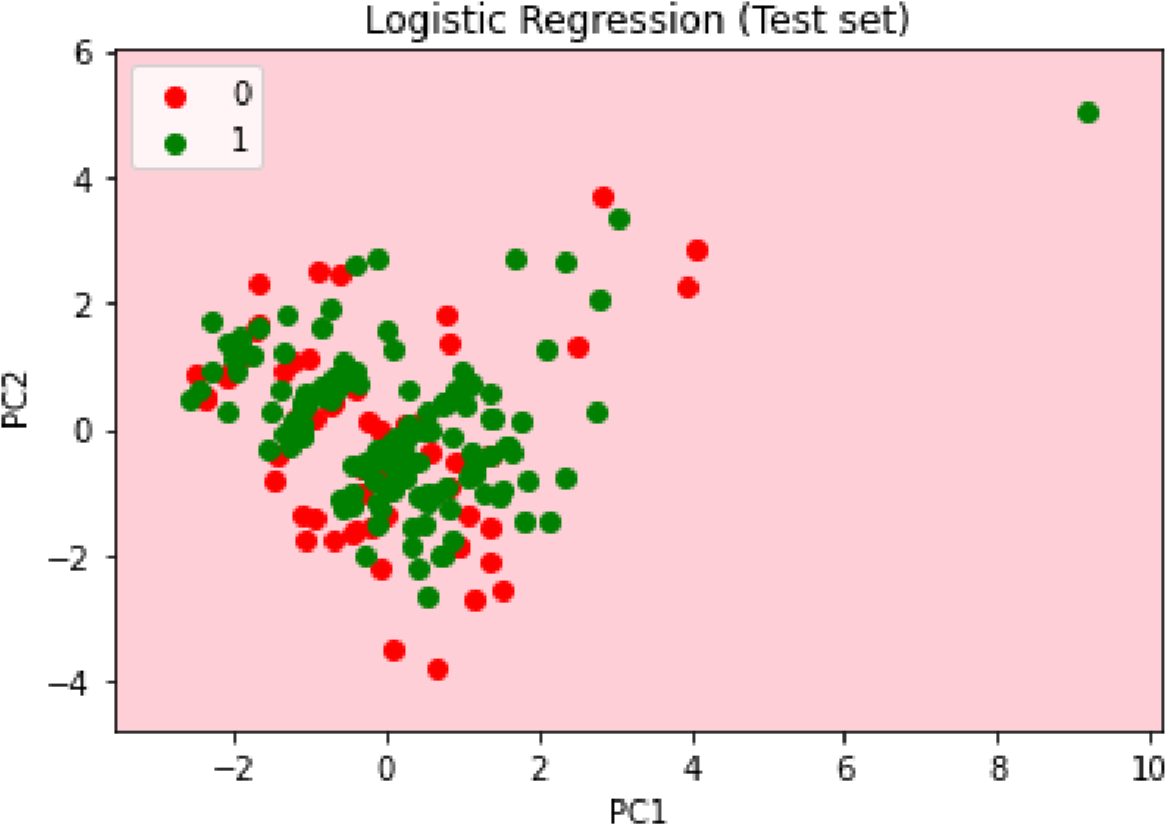
X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s 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 avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



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

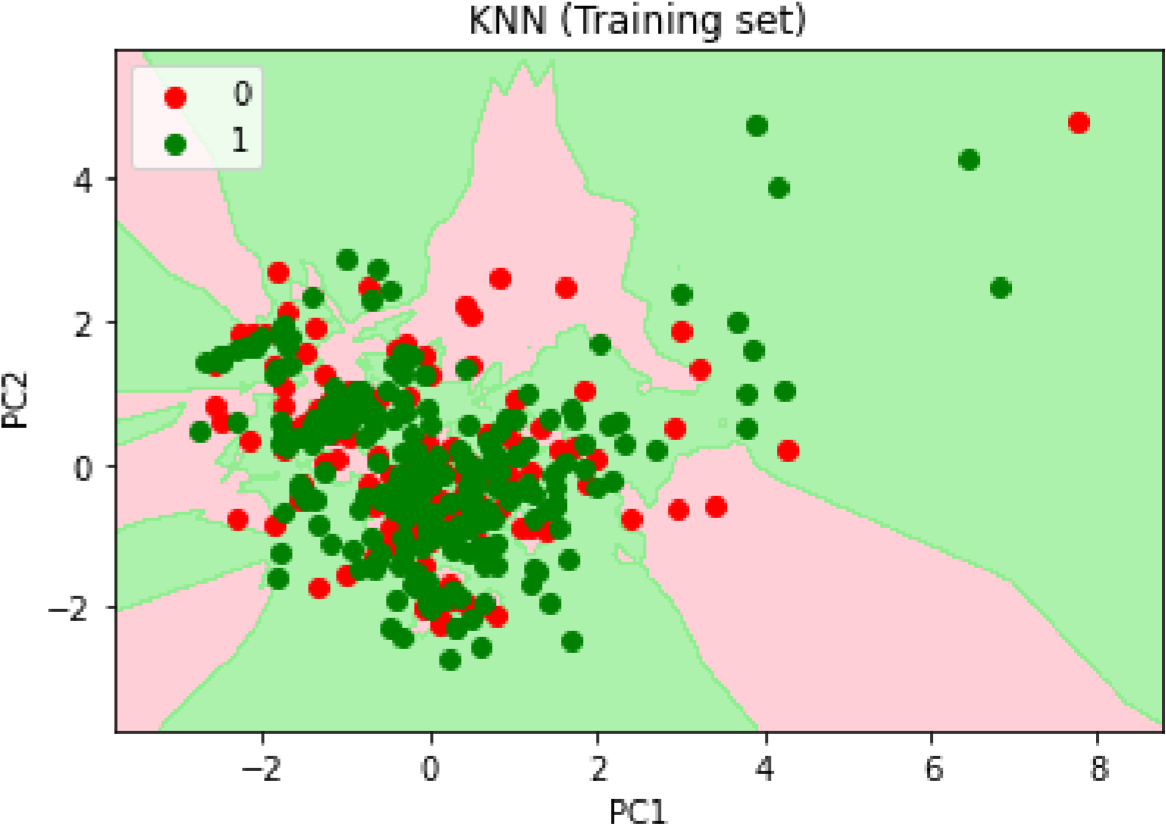
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s alpha = 0.75, cmap = ListedColormap(('pink', '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()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



from matplotlib.colors import ListedColormap

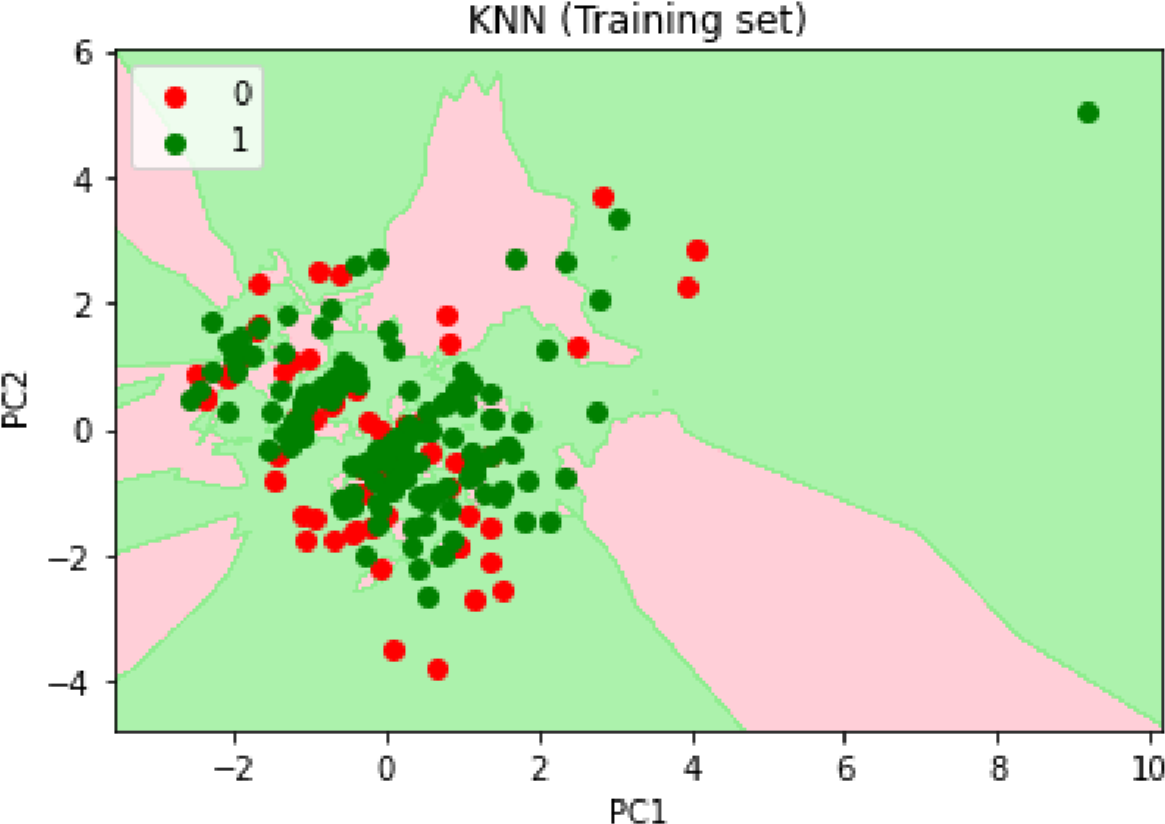
X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s 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()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



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

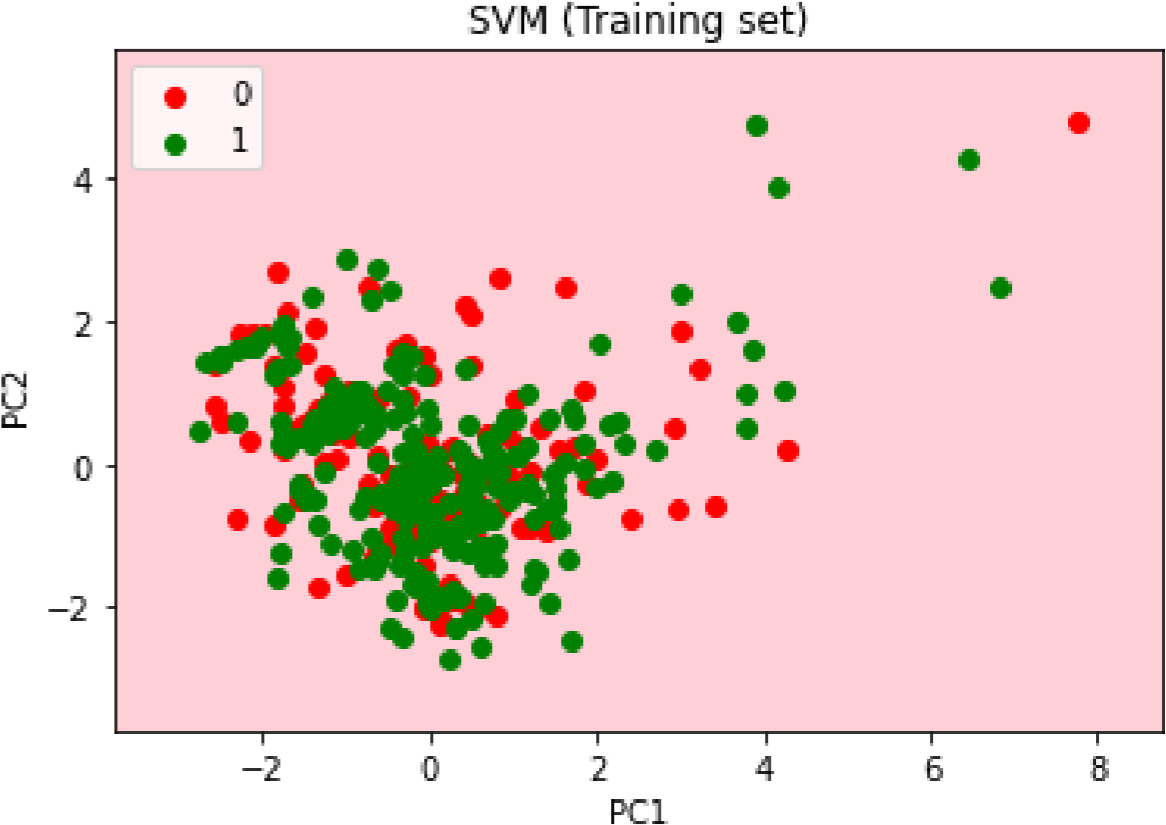
X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s 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()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



from matplotlib.colors import ListedColormap

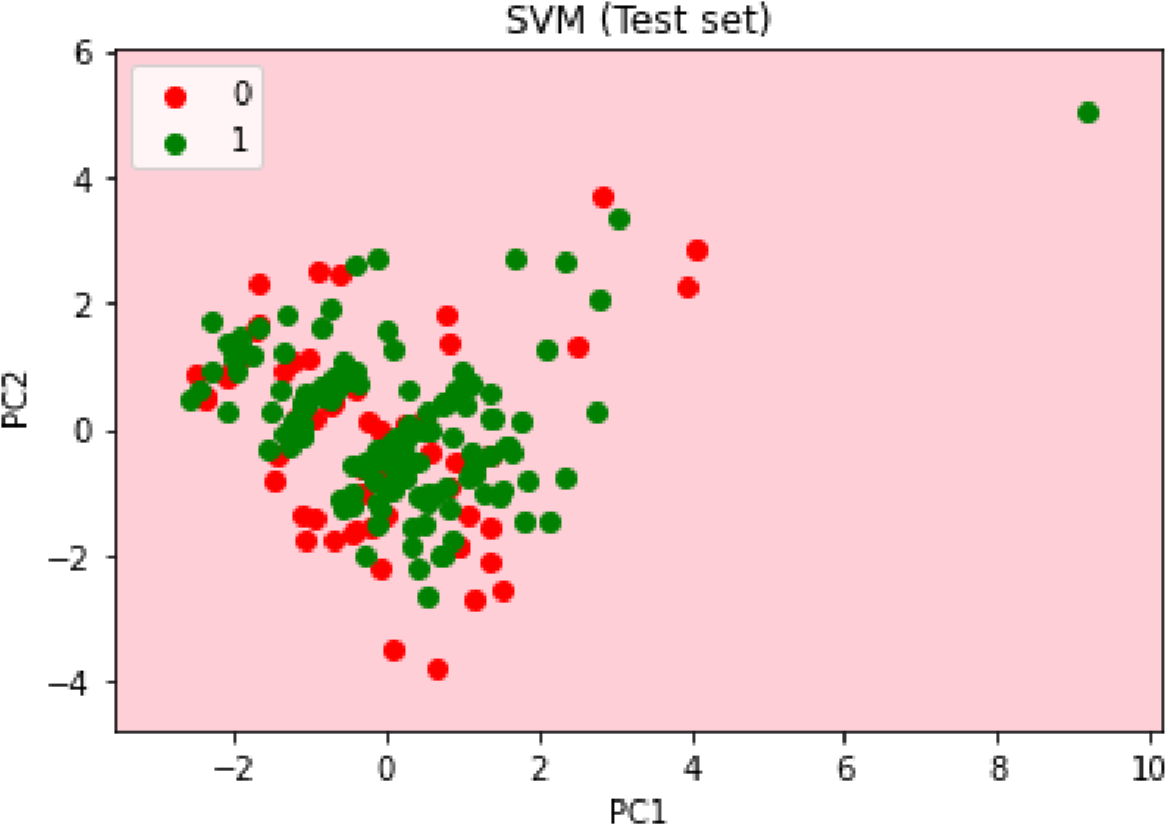
X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s 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()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



Naiv

e Ba

y

es

from

sklearn.naive\_bayes

import

GaussianNB

classifier

=

GaussianNB

()

classifier.fit

(

X\_train

,

y\_train

)

GaussianNB()

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

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

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

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

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

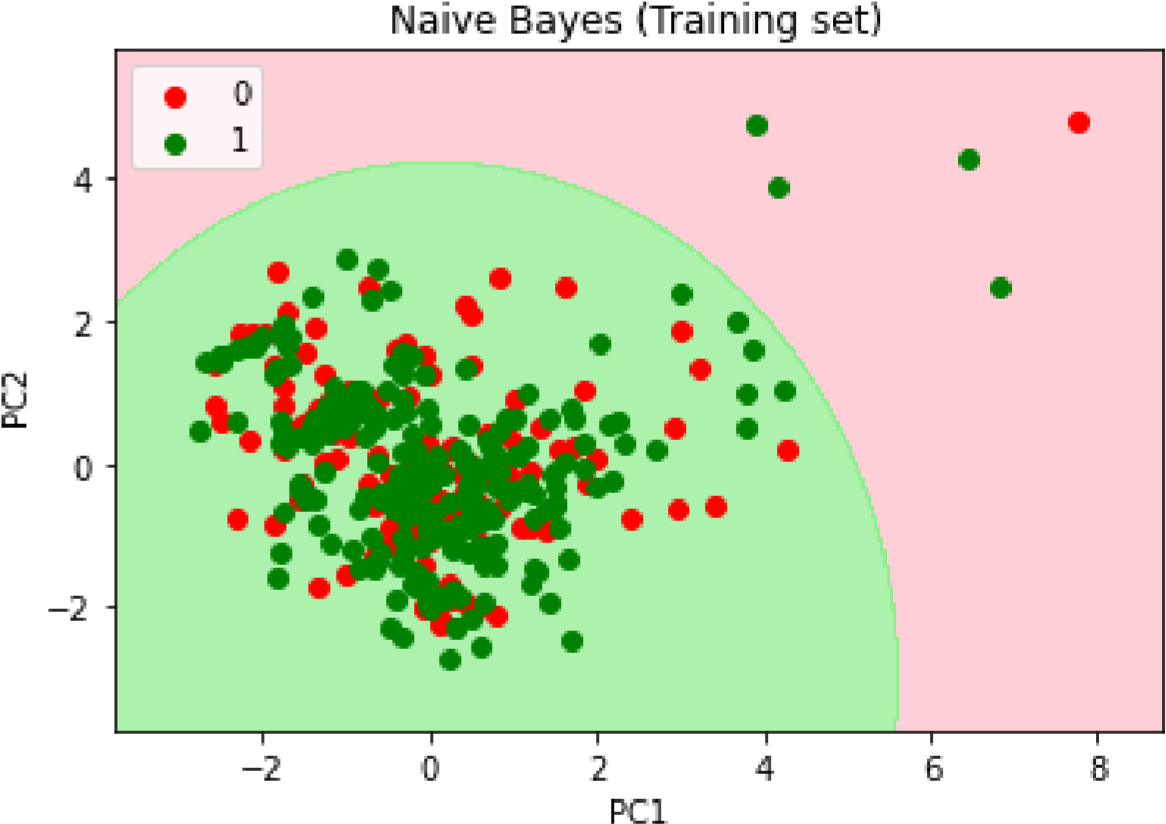
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s alpha = 0.75, cmap = ListedColormap(('pink', '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()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



from matplotlib.colors import ListedColormap

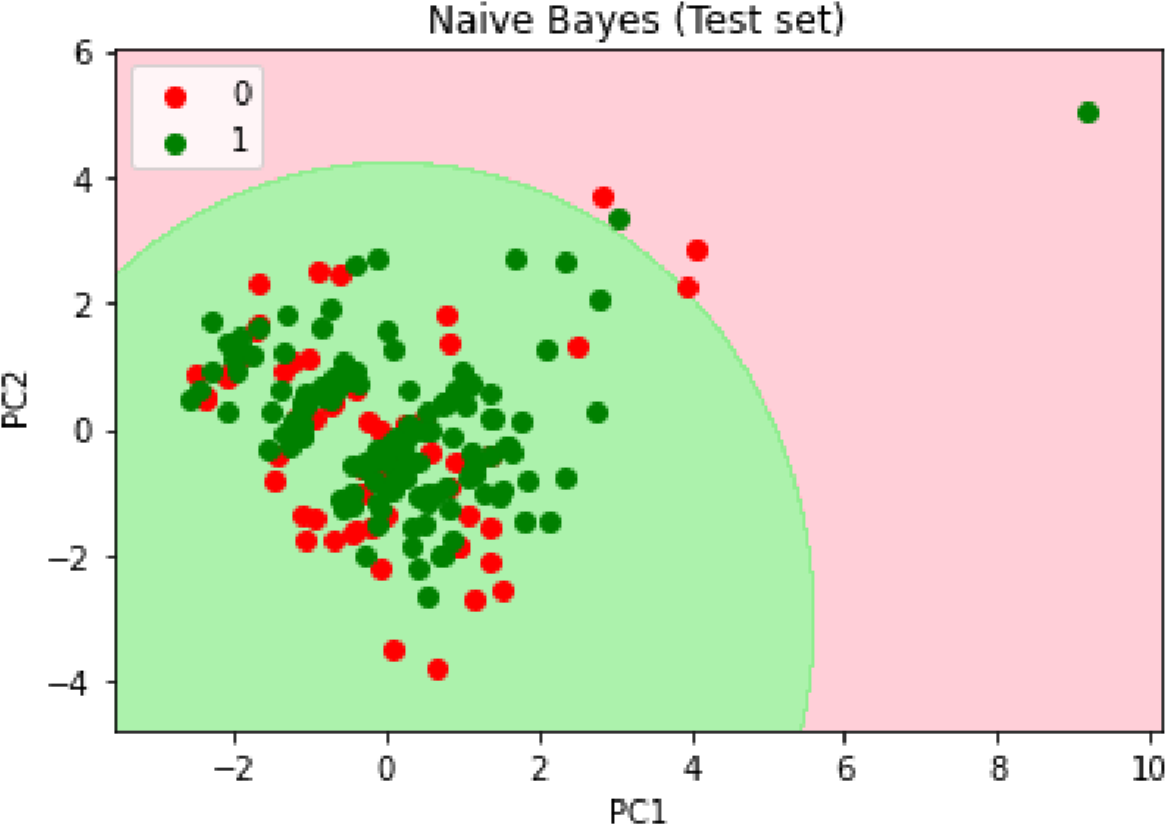
X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s 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()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



Decision T

r

ee Classi

cation

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\_te

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

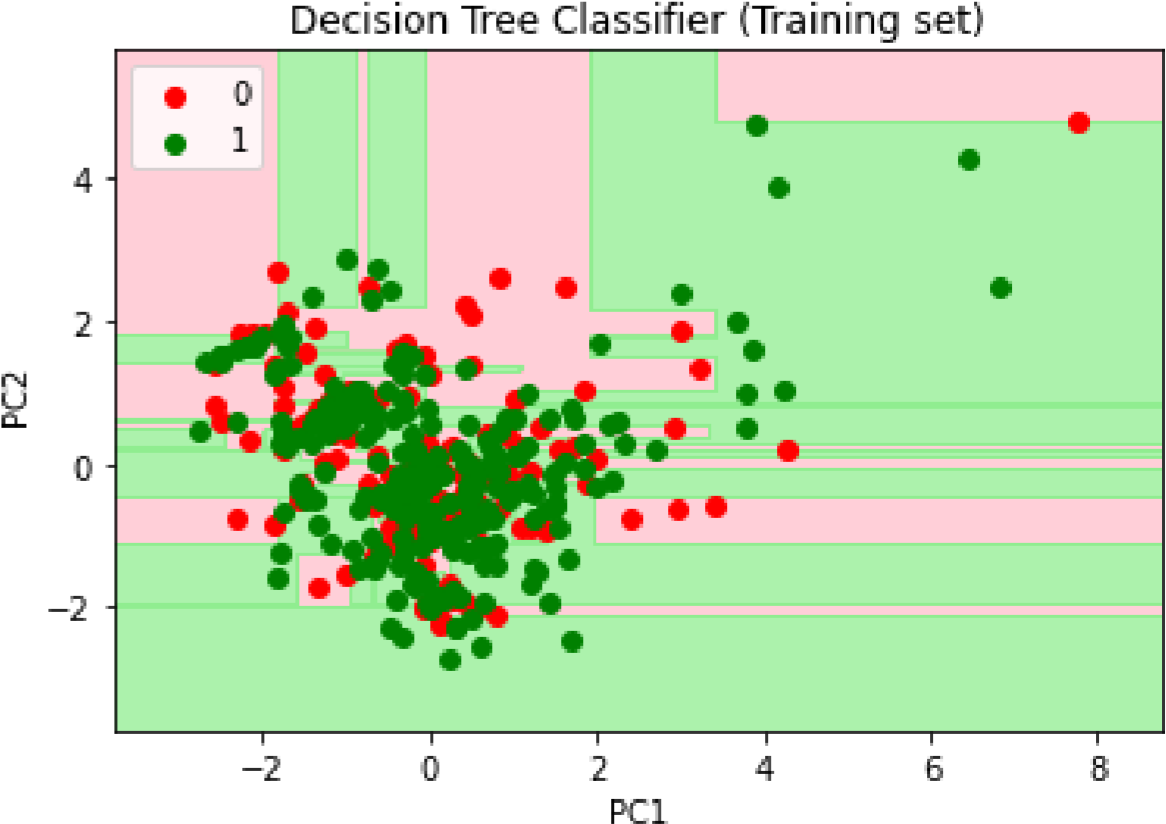
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s alpha = 0.75, cmap = ListedColormap(('pink', '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()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



from matplotlib.colors import ListedColormap

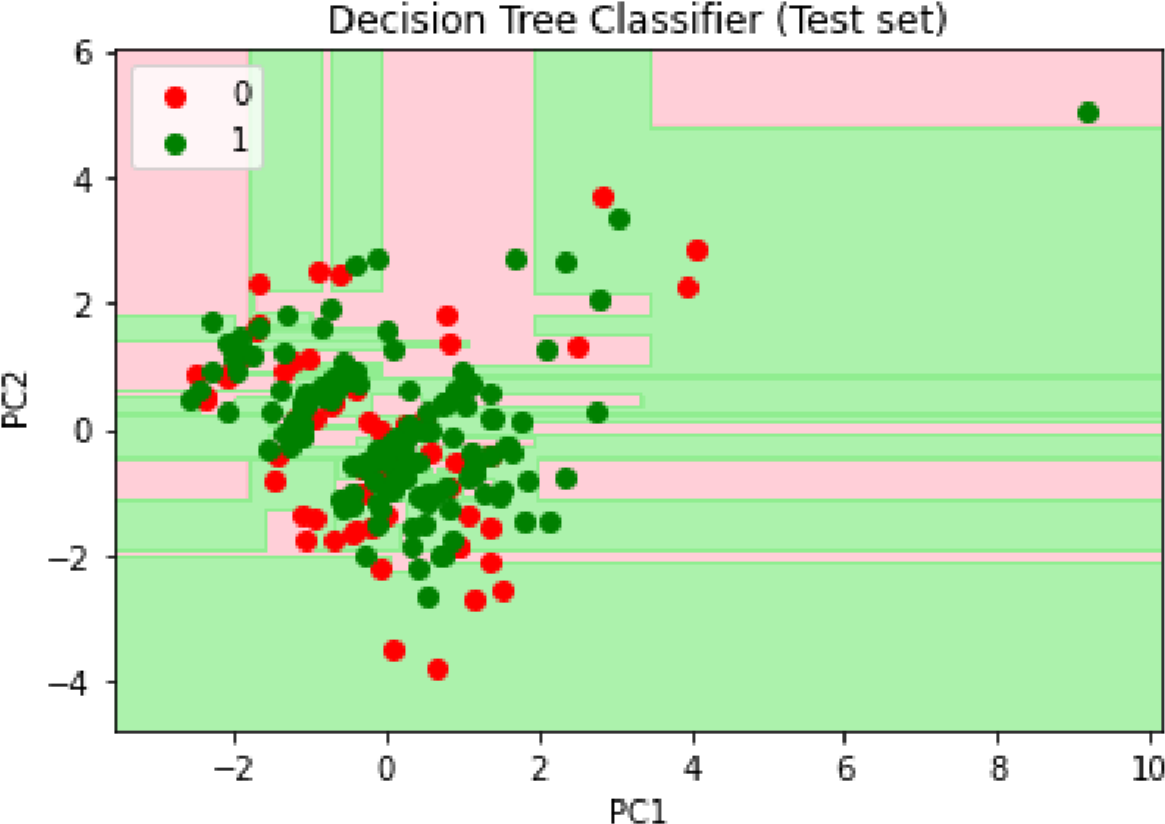
X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s 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()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



Random F

or

est Classi

cation

from

sklearn.ensemble

import

RandomForestClassifier

classifier

=

RandomForestClassifier

(

n\_estimators

=

10

,

criterion

=

'entropy'

,

random\_state

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,

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

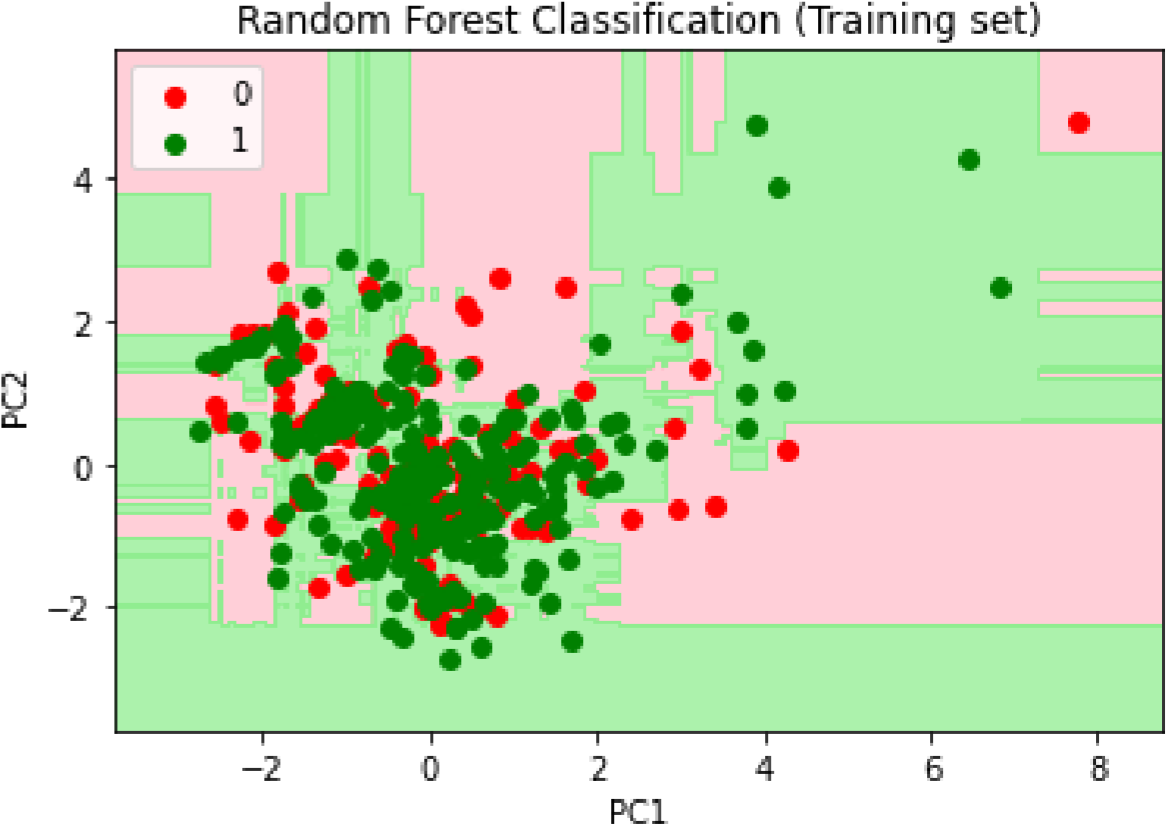
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s alpha = 0.75, cmap = ListedColormap(('pink', '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()

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoide



from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

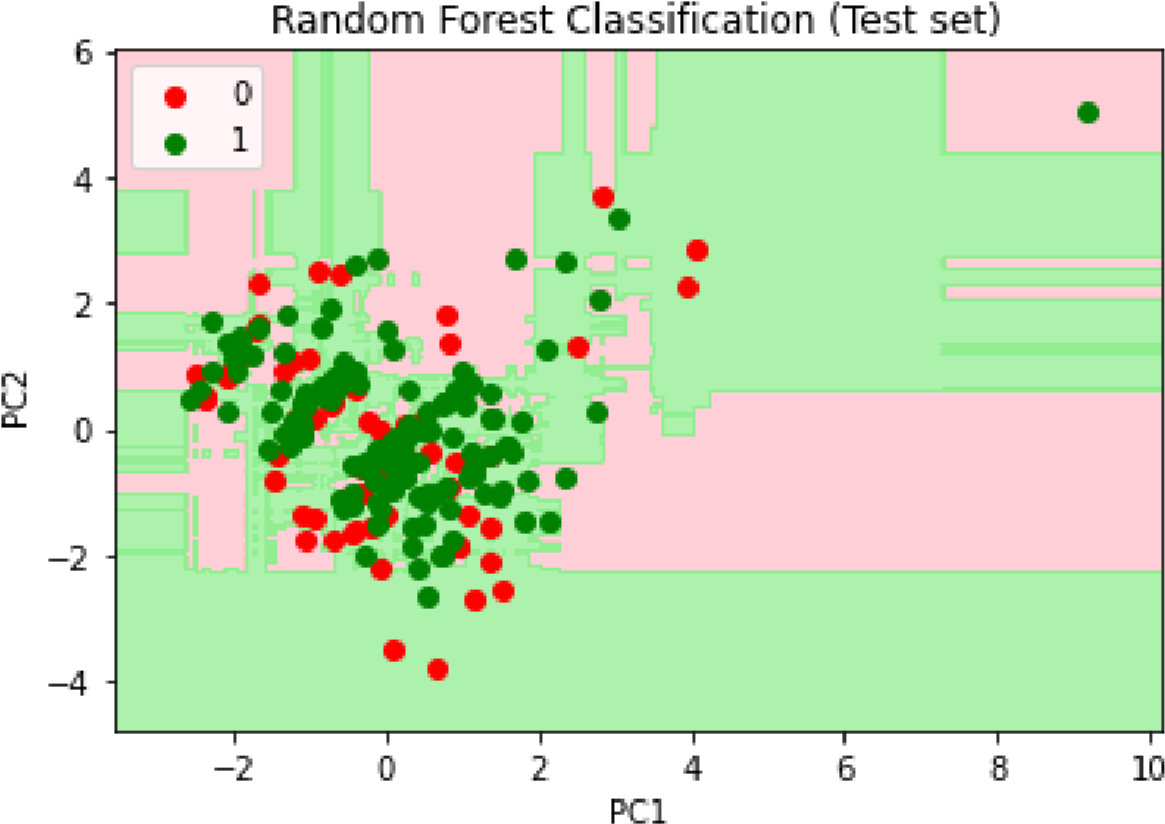
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1 np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1 plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.s alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen'))) plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y\_set)):

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoid

e

\*c\* argument looks like a single numeric RGB or RGBA sequence, which should be avoid

e



Results:

The accur

acy of Logistic Regr

ession is: 70.73 %

The accur

acy of KNN is: 62.92 %

The accur

acy of SVM is: 70.73 %

The accur

acy of Naiv

e Ba

y

es is: 71.21 %

The accur

acy of Decision T

r

ee Classi

er is: 53.63 %

The accur

acy of Random F

or

est Classi

cation is: 58.53 %

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()