In [1]:

!

pip install heuristicsearch

Collecting heuristicsearch

Downloading heuristicsearch-0.1.1-py3-none-any.whl (5.5 kB)

Installing collected packages: heuristicsearch

Successfully installed heuristicsearch-0.1.1

In [3]: from heuristicsearch.a\_star\_search import AStar aj\_list={'A': [('B', 6), ('F', 3)], 'B': [('C', 3), ('D', 2)],

'C': [('D', 1), ('E', 5)], 'D': [('C', 1),('E', 8)],

'E': [('I', 5), ('J', 5)],

'F': [('G', 1),('H', 7)],

'G': [('I', 3)],

'H': [('I', 2)],

'I': [('E', 5), ('J', 3)],

}

heuristics={'A': 10,'B': 8,'C': 5,'D': 7,'E': 3,'F': 6,'G': 5,'H':

3,'I': 1,'J': 0}

graph=AStar(aj\_list,heuristics) graph.apply\_a\_star(start='A',stop='J')

Path

A -> F -> G -> I -> J

Cost

0 -> 3 -> 4 -> 7 -> 10

In [4]: from heuristicsearch.ao\_star import AOStar print("Graphs-1") heuristic={'A':1,'B':6,'C':2,'D':12,'E':2,'F':1,'G':5,'H':7,'J':

1,'T':3}

aj\_list={'A':[[('B',1),('C',1)],[('D',1)]],

'B':[[('G',1)],[('H',1)]],

'C':[[('J',1)]],

'D':[[('E',1),('F',1)]],

'G':[[('I',1)]]

} graph=AOStar(aj\_list,heuristic,'A') graph.applyAOStar()

Graphs-1

PROCESSING NODE : A

-----------------------------------------------------------------------

------------------

10 ['B', 'C']

PROCESSING NODE : B

-----------------------------------------------------------------------

------------------

6 ['G']

PROCESSING NODE : A

-----------------------------------------------------------------------

------------------

10 ['B', 'C']

PROCESSING NODE : G

-----------------------------------------------------------------------

------------------

1. ['I']

PROCESSING NODE : B

-----------------------------------------------------------------------

------------------

1. ['G']

PROCESSING NODE : A

-----------------------------------------------------------------------

------------------

6 ['B', 'C']

PROCESSING NODE : I

-----------------------------------------------------------------------

------------------

1. []

PROCESSING NODE : G

-----------------------------------------------------------------------

------------------

1. ['I']

PROCESSING NODE : B

-----------------------------------------------------------------------

------------------

1. ['G']

PROCESSING NODE : A

-----------------------------------------------------------------------

------------------

6 ['B', 'C']

PROCESSING NODE : C

-----------------------------------------------------------------------

------------------

2 ['J']

PROCESSING NODE : A -----------------------------------------------------------------------

------------------

6 ['B', 'C']

PROCESSING NODE : J

-----------------------------------------------------------------------

------------------

1. []

PROCESSING NODE : C

-----------------------------------------------------------------------

------------------

1. ['J']

PROCESSING NODE : A

-----------------------------------------------------------------------

------------------

5 ['B', 'C']

FOR THE SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

------------------------------------------------------------

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

------------------------------------------------------------

In [5]:

import

numpy

as

np

import

pandas

as

pd

data

=

pd

.

DataFrame

(

data

=

pd

.

read\_csv

(

'rashika.csv'

))

In [6]:

data

Out[6]:

Sky

Air Temp

Humidity

Wind

Water

Forecast

Enjoy Sport

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 Sunny | Warm | Normal Strong | Warm | Same | Yes |
| 1 Sunny | Warm | High Strong | Warm | Same | Yes |
| 2 Rainy | Cold | High Strong | Warm | Change | No |
| 3 Sunny | Warm | High Strong | Cold | Change | Yes |

In [9]: concepts = np.array(data.iloc[:,0:-1]) target = np.array(data.iloc[:,-1]) def learn(concepts, target): specific\_h = concepts[0].copy()

print("initialization of specific\_h and general\_h") print(specific\_h)

general\_h = [["?" for i in range(len(specific\_h))] for i in range

(len(specific\_h))] print(general\_h) for i, h in enumerate(concepts): if target[i] == "Yes": for x in range(len(specific\_h)): if h[x] != specific\_h[x]: specific\_h[x] = '?' general\_h[x][x] = '?' if target[i] == "No": for x in range(len(specific\_h)): if h[x] != specific\_h[x]: general\_h[x][x] = specific\_h[x] else: general\_h[x][x] = '?' print(" steps of Candidate Elimination Algorithm",i+1) print("Specific\_h ",i+1,"\n ") print(specific\_h)

print("general\_h ", i+1, "\n ") print(general\_h)

indices = [i for i, val in enumerate(general\_h) if val == ['?',

'?', '?', '?', '?', '?']] for i in indices: general\_h.remove(['?', '?', '?', '?', '?', '?']) return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target) print("Final Specific\_h:", s\_final, sep="\n") print("Final General\_h:", g\_final, sep="\n")

initialization of specific\_h and general\_h

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',

'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',

'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']] steps of Candidate Elimination Algorithm 4

Specific\_h 4

['Sunny' 'Warm' '?' 'Strong' '?' '?'] general\_h 4

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']] Final Specific\_h:

['Sunny' 'Warm' '?' 'Strong' '?' '?'] Final General\_h:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

In [15]: import pandas as pd from pprint import pprint

from sklearn.feature\_selection import mutual\_info\_classif from collections import Counter

def id3(df, target\_attribute, attribute\_names, default\_class=None): cnt=Counter(x for x in df[target\_attribute]) if len(cnt)==1: return next(iter(cnt))

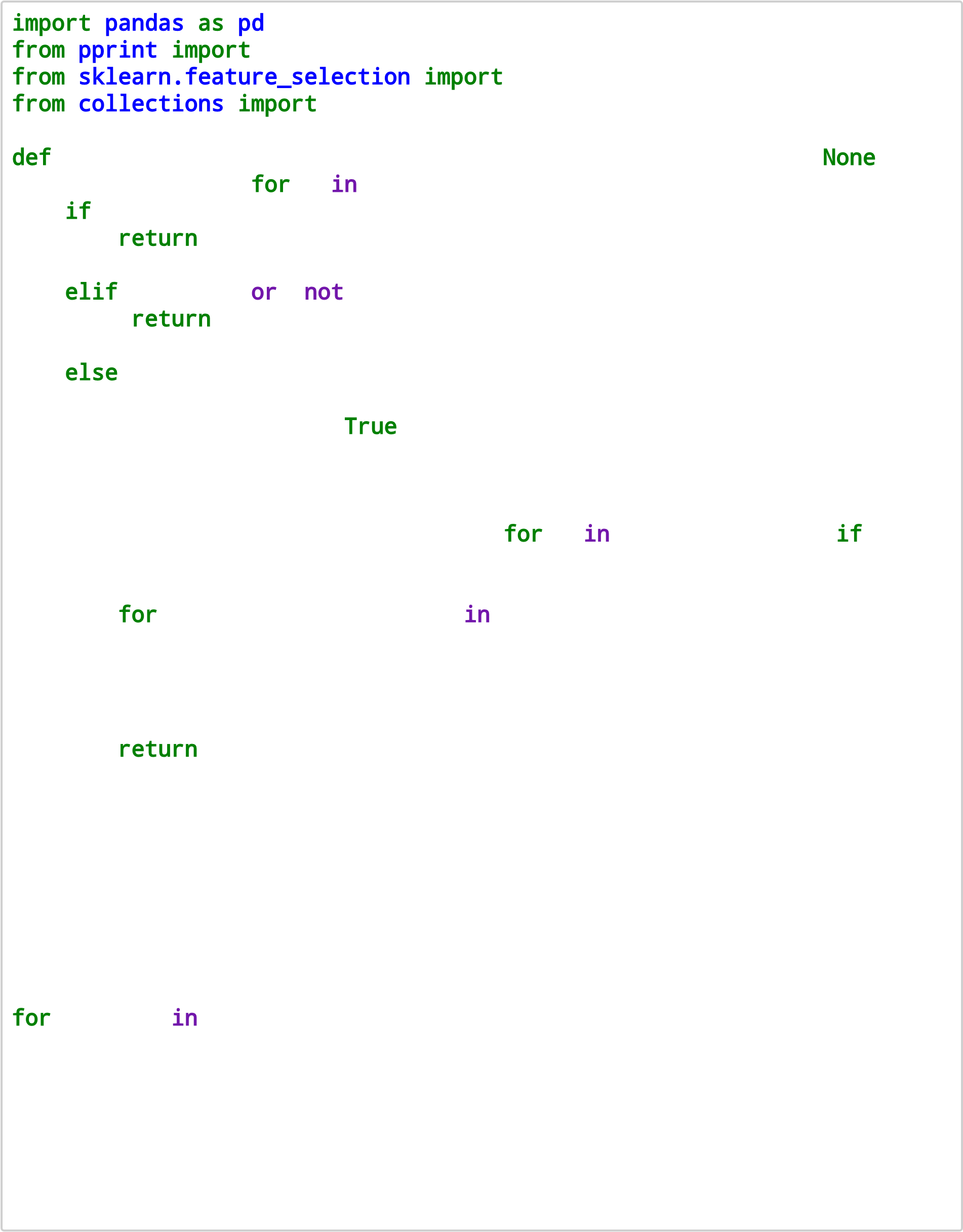
elif df.empty or (not attribute\_names): return default\_class

else: gainz = mutual\_info\_classif(df[attribute\_names],df[target\_attr

ibute],discrete\_features=True)

index\_of\_max=gainz.tolist().index(max(gainz)) best\_attr=attribute\_names[index\_of\_max] tree={best\_attr:{}}

remaining\_attribute\_names=[i for i in attribute\_names if i!=be

st\_attr] for attr\_val, data\_subset in df.groupby(best\_attr): subtree=id3(data\_subset, target\_attribute, remaining\_attri

bute\_names,default\_class)

tree[best\_attr][attr\_val]=subtree

return tree df=pd.read\_csv("kirana.csv")

attribute\_names=df.columns.tolist() print("List of attribut name") attribute\_names.remove("PlayTennis")

for colname in df.select\_dtypes("object"): df[colname], \_ = df[colname].factorize()

print(df)

tree= id3(df,"PlayTennis", attribute\_names) print("The tree structure") pprint(tree)

List of attribut name

Outlook Temperature Humidity Wind PlayTennis 0 0 0 0 0 0

1. 0 0 0 1 0
2. 1 0 0 0 1
3. 2 1 0 0 1
4. 2 2 1 0 1
5. 2 2 1 1 0
6. 1 2 1 1 1
7. 0 1 0 0 0
8. 0 2 1 0 1
9. 2 1 1 0 1
10. 0 1 1 1 1
11. 1 1 0 1 1
12. 1 0 1 0 1
13. 2 1 0 1 0

The tree structure {'Outlook': {0: {'Humidity': {0: 0, 1: 1}}, 1: 1, 2: {'Wind': {0: 1, 1:

0}}}} In [16]:

In [19]: import numpy as np

X=np.array(([2,9],[1,5],[3,6]),dtype=float) y=np.array(([92],[86],[89]),dtype=float)

X=X/np.amax(X,axis=0) y=y/100 def sigmoid(x): return 1/(1+np.exp(-x))

def derivatives\_sigmoid(x):

return x\*(1-x)

epoch=7000 lr=0.25

inputlayer\_neurons=2 hiddenlayer\_neurons=3 output\_neurons=1

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons)) bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons)) bout=np.random.uniform(size=(1,output\_neurons)) for i in range(epoch): hinp1=np.dot(X,wh) hinp=hinp1+bh hlayer\_act=sigmoid(hinp) outinp1=np.dot(hlayer\_act,wout) outinp=outinp1+bout output=sigmoid(outinp) EO=y-output

outgrad=derivatives\_sigmoid(output) d\_output=EO\*outgrad EH=d\_output.dot(wout.T)

hiddengrad=derivatives\_sigmoid(hlayer\_act) d\_hiddenlayer=EH\*hiddengrad wout+=hlayer\_act.T.dot(d\_output)\*lr wh+=X.T.dot(d\_hiddenlayer)\*lr

print("Input=\n"+str(X)) print("Actual output:\n"+str(y)) print("predicated output:",output)

Input=

[[0.66666667 1. ] [0.33333333 0.55555556]

[1. 0.66666667]] Actual output:

[[0.92] [0.86]

[0.89]] predicated output: [[0.89574228]

[0.87965404]



In [21]: data

Out[21]:

Outlook Temperature Humidity Wind PlayTennis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | Sunny | Hot | High | Weak | No |
| 1 | Sunny | Hot | High | Strong | No |
| 2 | Overcast | Hot | High | Weak | Yes |
| 3 | Rain | Mild | High | Weak | Yes |
| 4 | Rain | Cool | Normal | Weak | Yes |
| 5 | Rain | Cool | Normal | Strong | No |
| 6 | Overcast | Cool | Normal | Strong | Yes |
| 7 | Sunny | Mild | High | Weak | No |
| 8 | Sunny | Cool | Normal | Weak | Yes |
| 9 | Rain | Mild | Normal | Weak | Yes |
| 10 | Sunny | Mild | Normal | Strong | Yes |
| 11 | Overcast | Mild | High | Strong | Yes |
| 12 | Overcast | Hot | Normal | Weak | Yes |
| 13 | Rain | Mild | High | Strong | No |

In [24]: print("The first 5 Values of data is :\n", data.head())

1. = data.iloc[:, :-1] print("\nThe First 5 values of the train attributes is\n", X.head())
2. = data.iloc[:, -1]

print("\nThe First 5 values of target values is\n", Y.head())

obj1= LabelEncoder()

X.Outlook = obj1.fit\_transform(X.Outlook)

print("\n The Encoded and Transformed Data in Outlook\n",X.Outlook)

obj2 = LabelEncoder()

X.Temperature = obj2.fit\_transform(X.Temperature)

obj3 = LabelEncoder()

X.Humidity = obj3.fit\_transform(X.Humidity)

obj4 = LabelEncoder()

X.Wind = obj4.fit\_transform(X.Wind)

print("\n The Encoded and Transformed Training Examples \n", X.head())

obj5 = LabelEncoder() Y = obj5.fit\_transform(Y) print("The class Label encoded in numerical form is",Y)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y, test\_size =

0.20)

from sklearn.naive\_bayes import GaussianNB classifier = GaussianNB() classifier.fit(X\_train, Y\_train) from sklearn.metrics import accuracy\_score

print("Accuracy is:", accuracy\_score(classifier.predict(X\_test), Y\_tes t))

The first 5 Values of data is :

Outlook Temperature Humidity Wind PlayTennis

1. Sunny Hot High Weak No
2. Sunny Hot High Strong No
3. Overcast Hot High Weak Yes
4. Rain Mild High Weak Yes
5. Rain Cool Normal Weak Yes

The First 5 values of the train attributes is Outlook Temperature Humidity Wind

1. Sunny Hot High Weak
2. Sunny Hot High Strong
3. Overcast Hot High Weak
4. Rain Mild High Weak
5. Rain Cool Normal Weak

The First 5 values of target values is

1. No
2. No
3. Yes
4. Yes
5. Yes

Name: PlayTennis, dtype: object

The Encoded and Transformed Data in Outlook

1. 2
2. 2
3. 0
4. 1
5. 1
6. 1
7. 0
8. 2
9. 2
10. 1
11. 2
12. 0
13. 0
14. 1

Name: Outlook, dtype: int32

The Encoded and Transformed Training Examples

Outlook Temperature Humidity Wind

1. 2 1 0 1
2. 2 1 0 0
3. 0 1 0 1
4. 1 2 0 1
5. 1 0 1 1

The class Label encoded in numerical form is [0 0 1 1 1 0 1 0 1 1 1 1 1

0]

Accuracy is: 0.6666666666666666 C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:5303:

SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandasdocs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

self[name] = value

In [29]: import matplotlib.pyplot as plt from sklearn import datasets from sklearn.cluster import KMeans import pandas as pd import numpy as np iris = datasets.load\_iris() X = pd.DataFrame(iris.data)

X.columns =['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width'] y = pd.DataFrame(iris.target)

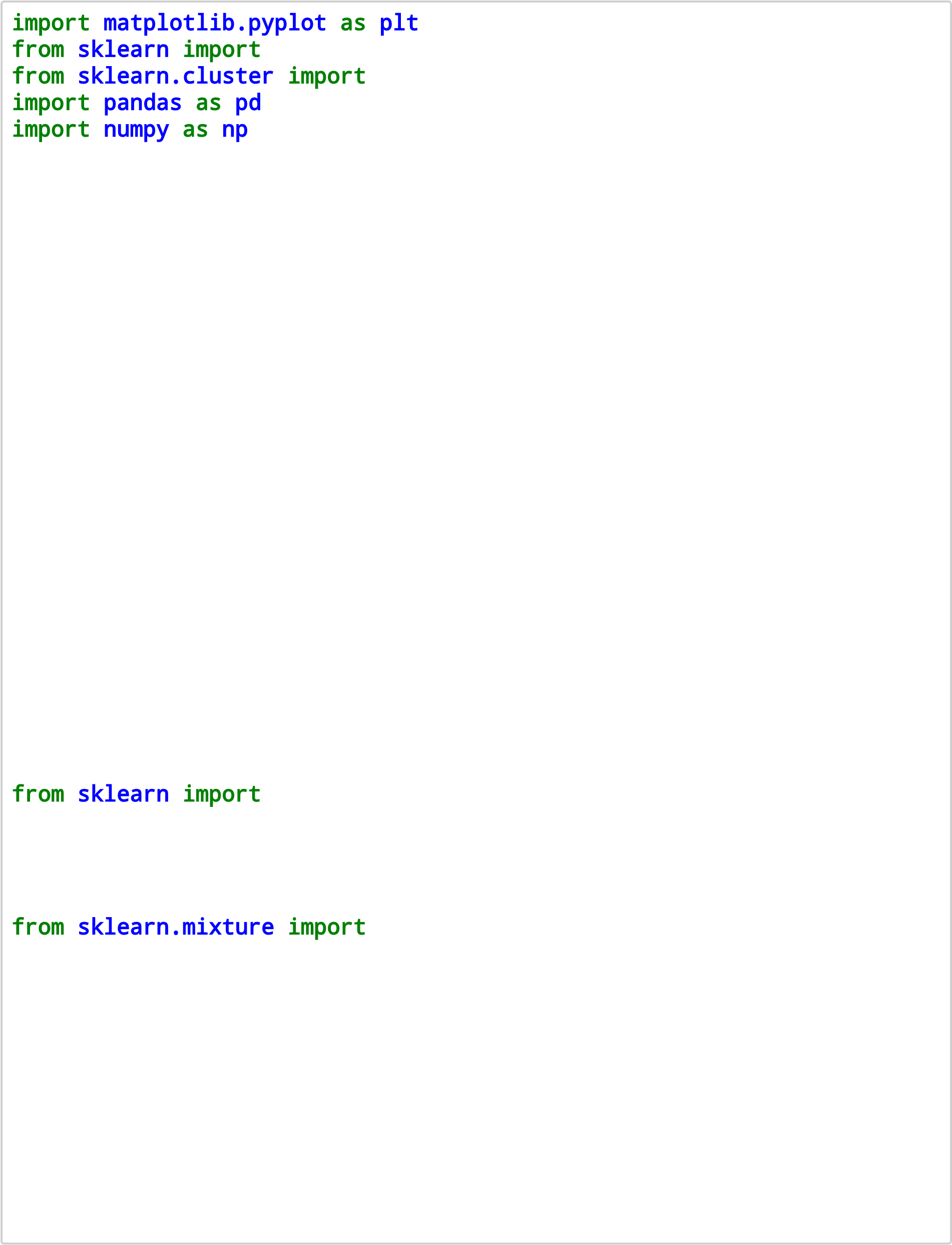
y.columns = ['Targets']

model = KMeans(n\_clusters=3) model.fit(X) # model.labels\_ : Gives cluster no for which samples belo

ngs to

plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black']) plt.subplot(1, 3, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40) plt.title('Real Clusters') plt.xlabel('Petal Length') plt.ylabel('Petal Width') plt.subplot(1, 3, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40) plt.title('K-Means Clustering') plt.xlabel('Petal Length') plt.ylabel('Petal Width')

from sklearn import preprocessing scaler = preprocessing.StandardScaler() scaler.fit(X) xsa = scaler.transform(X)

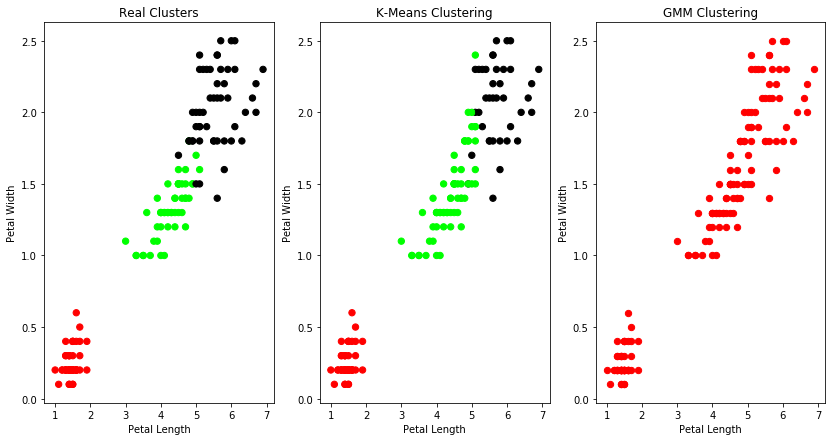
xs = pd.DataFrame(xsa, columns = X.columns) from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=40) gmm.fit(xs) plt.subplot(1, 3, 3)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[0], s=40) plt.title('GMM Clustering') plt.xlabel('Petal Length') plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matche d the true labels more closely than the Kmeans.')

Observation: The GMM using EM algorithm based clustering matched the tr ue labels more closely than the Kmeans.



In [31]: from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier from sklearn import datasets iris=datasets.load\_iris() print("Iris Data set loaded...")

x\_train, x\_test, y\_train, y\_test =train\_test\_split(iris.data,iris.targ et,test\_size=0.1) #random\_state=0 for i in range(len(iris.target\_names)): print("Label", i , "-",str(iris.target\_names[i]))

classifier = KNeighborsClassifier(n\_neighbors=2) classifier.fit(x\_train, y\_train) y\_pred=classifier.predict(x\_test)

print("Results of Classification using K-nn with K=1 ") for r in range(0,len(x\_test)): print(" Sample:", str(x\_test[r]), " Actual-label:",

str(y\_test[r])," Predicted-label:", str(y\_pred[r]))

print("Classification Accuracy :" ,

classifier.score(x\_test,y\_test));

Iris Data set loaded...

Label 0 - setosa

Label 1 - versicolor

Label 2 - virginica

Results of Classification using K-nn with K=1

Sample: [6.6 2.9 4.6 1.3] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 1.0

Sample: [6.9 3.1 4.9 1.5] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 1.0

Sample: [6.7 3.3 5.7 2.5] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 1.0

Sample: [5.7 2.5 5. 2. ] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 1.0

Sample: [4.8 3. 1.4 0.3] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 1.0

Sample: [5. 2. 3.5 1. ] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 1.0

Sample: [5.1 3.8 1.5 0.3] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 1.0

Sample: [6.4 3.1 5.5 1.8] Actual-label: 2 Predicted-label: 2

Classification Accuracy : 1.0

Sample: [6.3 3.3 4.7 1.6] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 1.0

Sample: [5.2 4.1 1.5 0.1] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 1.0

Sample: [4.9 2.4 3.3 1. ] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 1.0

Sample: [5. 3.5 1.6 0.6] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 1.0

Sample: [5.5 2.5 4. 1.3] Actual-label: 1 Predicted-label: 1

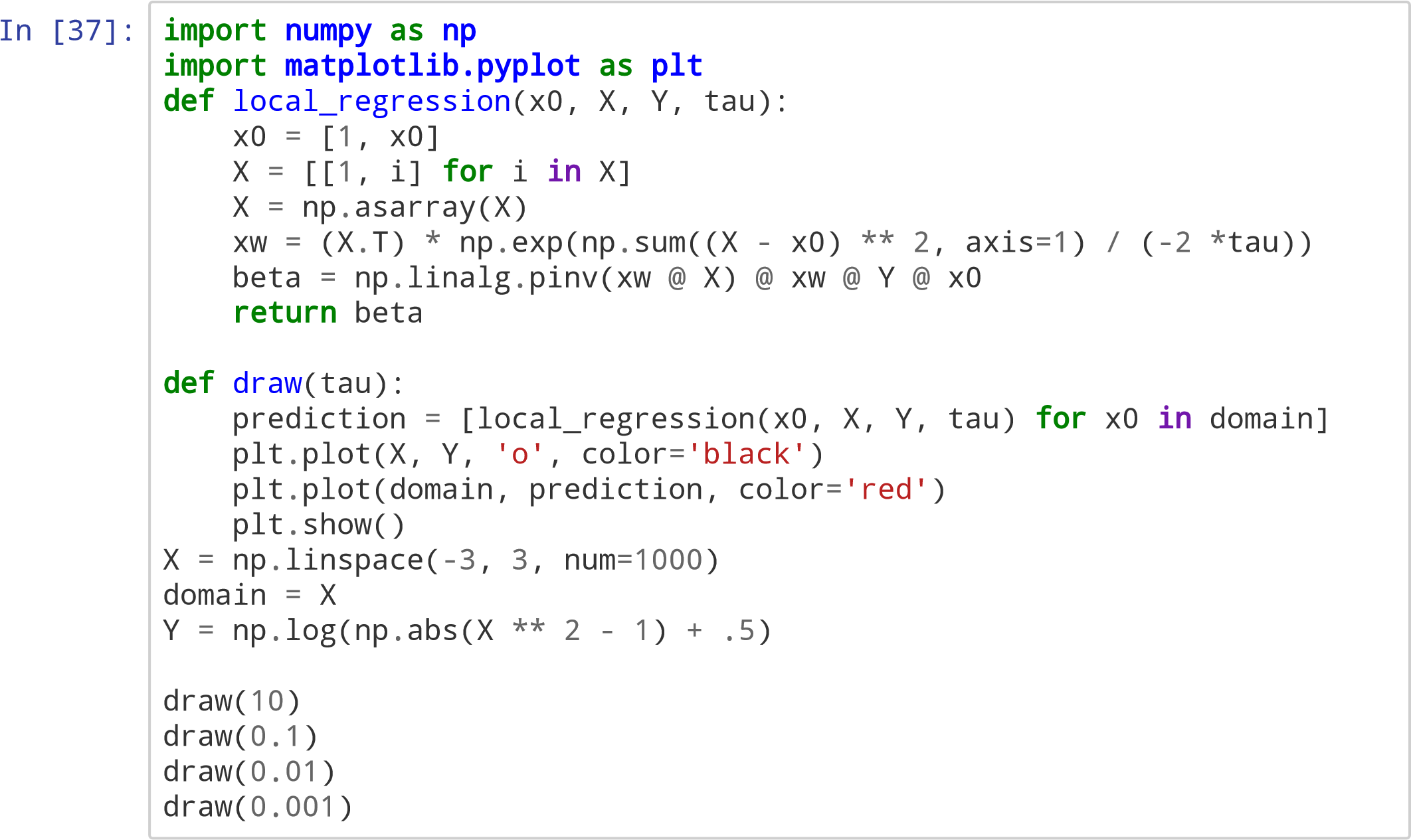
Classification Accuracy : 1.0

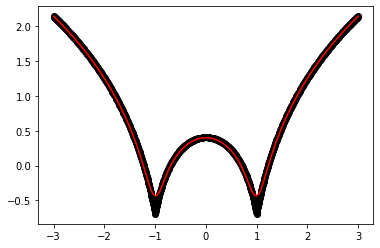
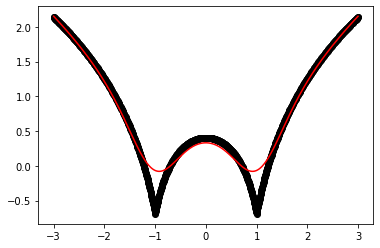
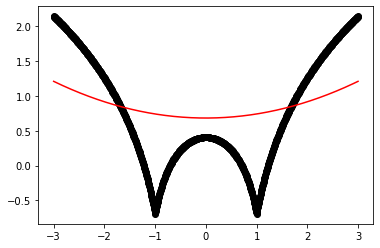
Sample: [6. 2.9 4.5 1.5] Actual-label: 1 Predicted-label: 1

Classification Accuracy : 1.0

Sample: [4.9 3.6 1.4 0.1] Actual-label: 0 Predicted-label: 0

Classification Accuracy : 1.0





In [ ]:

