SCSA2608 - AI and Visualization Lab

2.Classification Techniques

2.a. Naïve Bayes Classifier

import pandas as pd

f = pd.DataFrame({'Weather':['Sunny', 'Rainy', 'Sunny', 'Sunny'],

'Wind':['Mild', 'Mild', 'High', 'Mild'],

'Temp':['Moderate', 'Mild', 'Moderate', 'Mild'],

'go':['Yes', 'No', 'Yes', 'Yes']})

print(f.columns)

from sklearn.naive\_bayes import GaussianNB as g

from sklearn.preprocessing import LabelEncoder as le

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

l = le()

for i in f.columns:

f[i] = l.fit\_transform(f[i])

x = f.iloc[:, :3]

y = f.iloc[:, 3]

xtr, xte, ytr, yte = tt(x, y, test\_size=0.3)

gg = g()

gg.fit(xtr, ytr)

y\_pred = gg.predict(xte)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(yte, y\_pred))

OP

Index(['Weather', 'Wind', 'Temp', 'go'], dtype='object')

0.5

2.b.Support vector machine

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv("Social\_Network \_Ads1.csv")

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

X\_test = sc.transform(X\_test)

from sklearn.svm import SVC

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

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix, accuracy\_score

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

print(cm)

accuracy\_score(y\_test,y\_pred)

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(('red', 'green')))

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(('pink', 'green'))(i), label = j)

plt.title('SVM (Test set)')

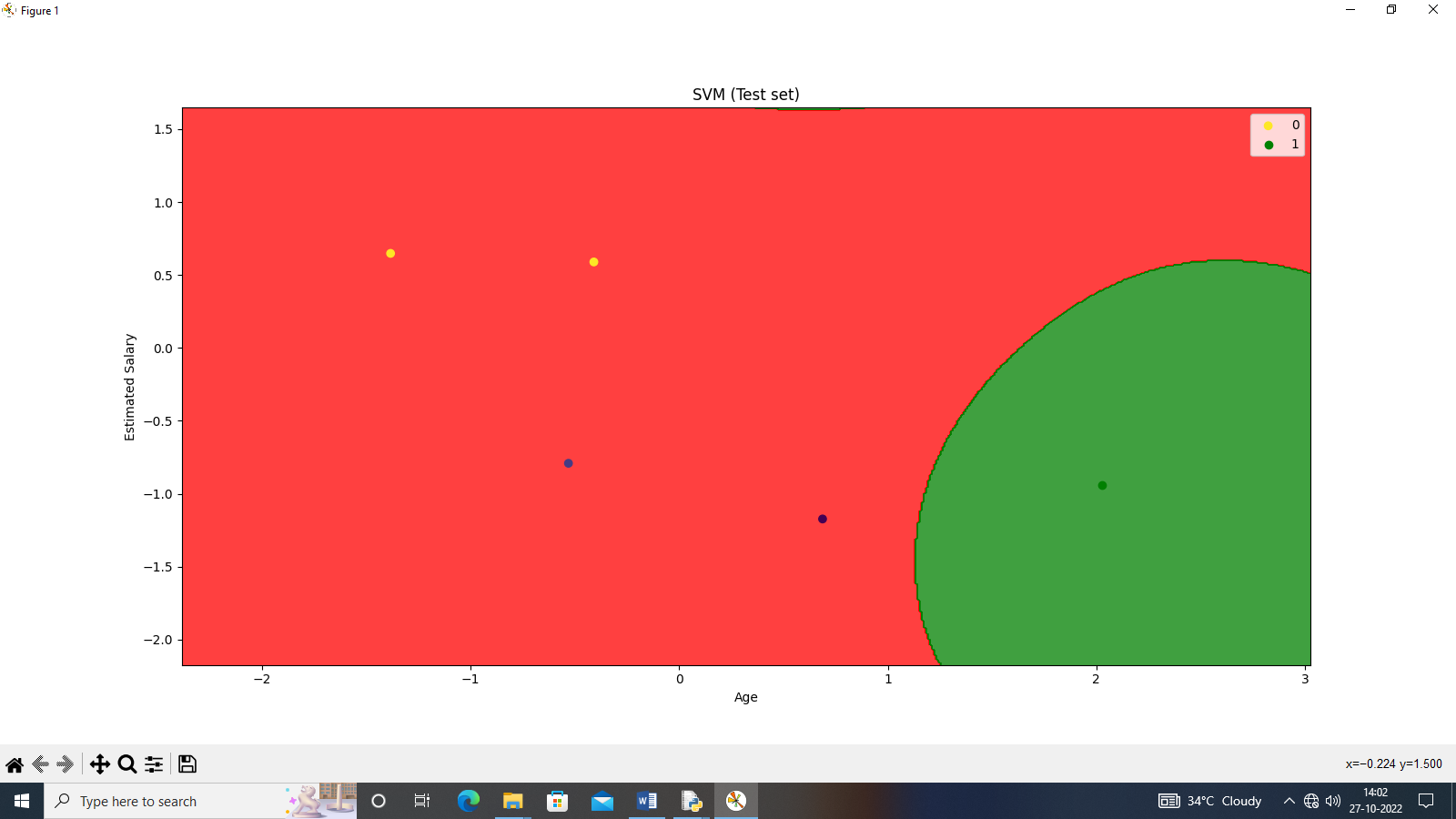
plt.xlabel('Age')

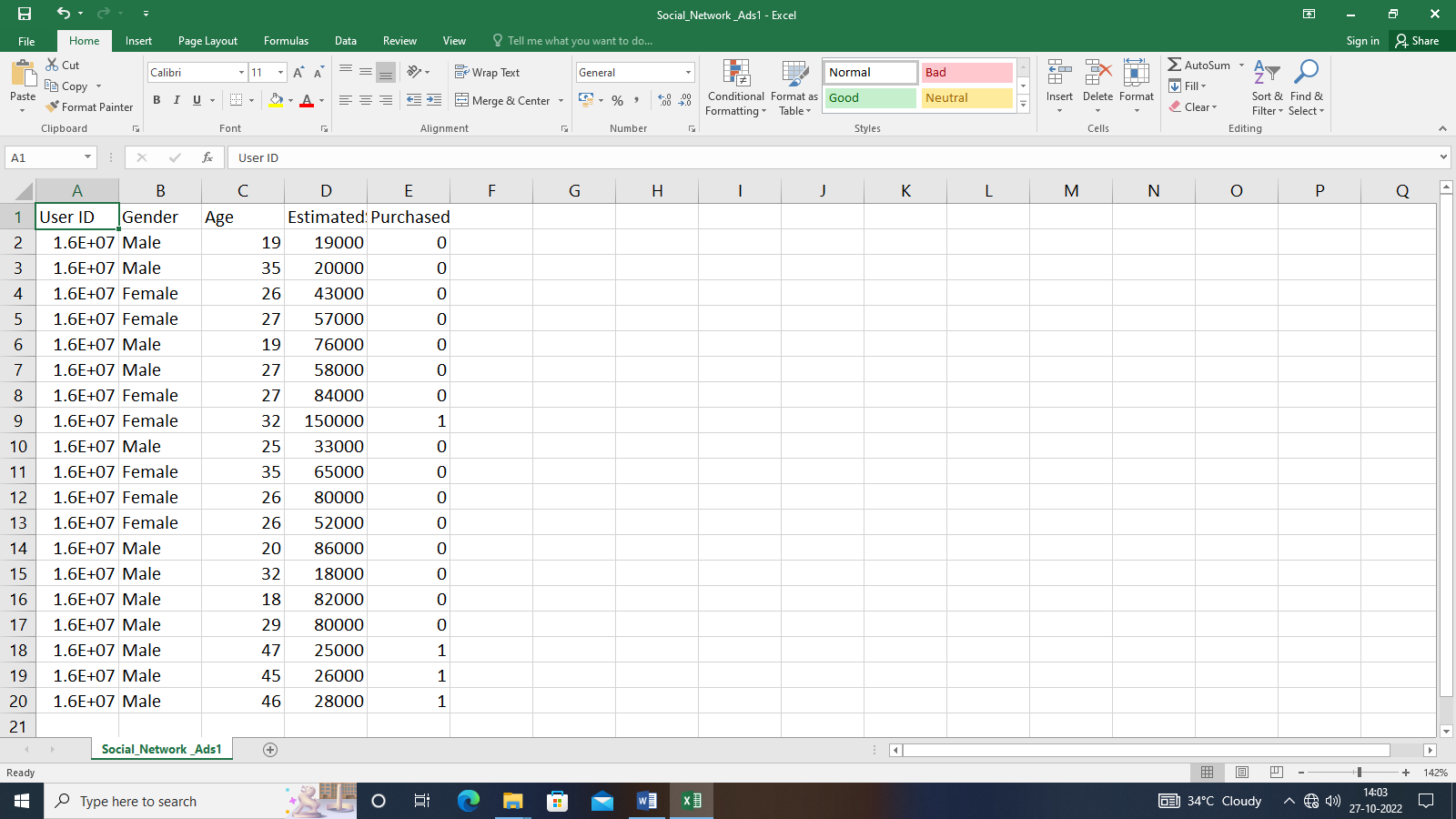
plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

[[4 0]

[0 1]] 



2.c.Logistic regression

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from matplotlib.colors import ListedColormap

dataset = pd.read\_csv("diabetes1.csv")

x = dataset.iloc[:, [4, 7]].values

y = dataset.iloc[:, 8].values

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

sc\_x = StandardScaler()

xtrain = sc\_x.fit\_transform(xtrain)

xtest = sc\_x.transform(xtest)

print (xtrain[0:10, :])

classifier = LogisticRegression(random\_state = 0)

classifier.fit(xtrain, ytrain)

y\_pred = classifier.predict(xtest)

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

print ("Confusion Matrix : \n", cm)

print ("Accuracy : ", accuracy\_score(ytest, y\_pred))

X\_set, y\_set = xtest, ytest

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(('red', 'green')))

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)

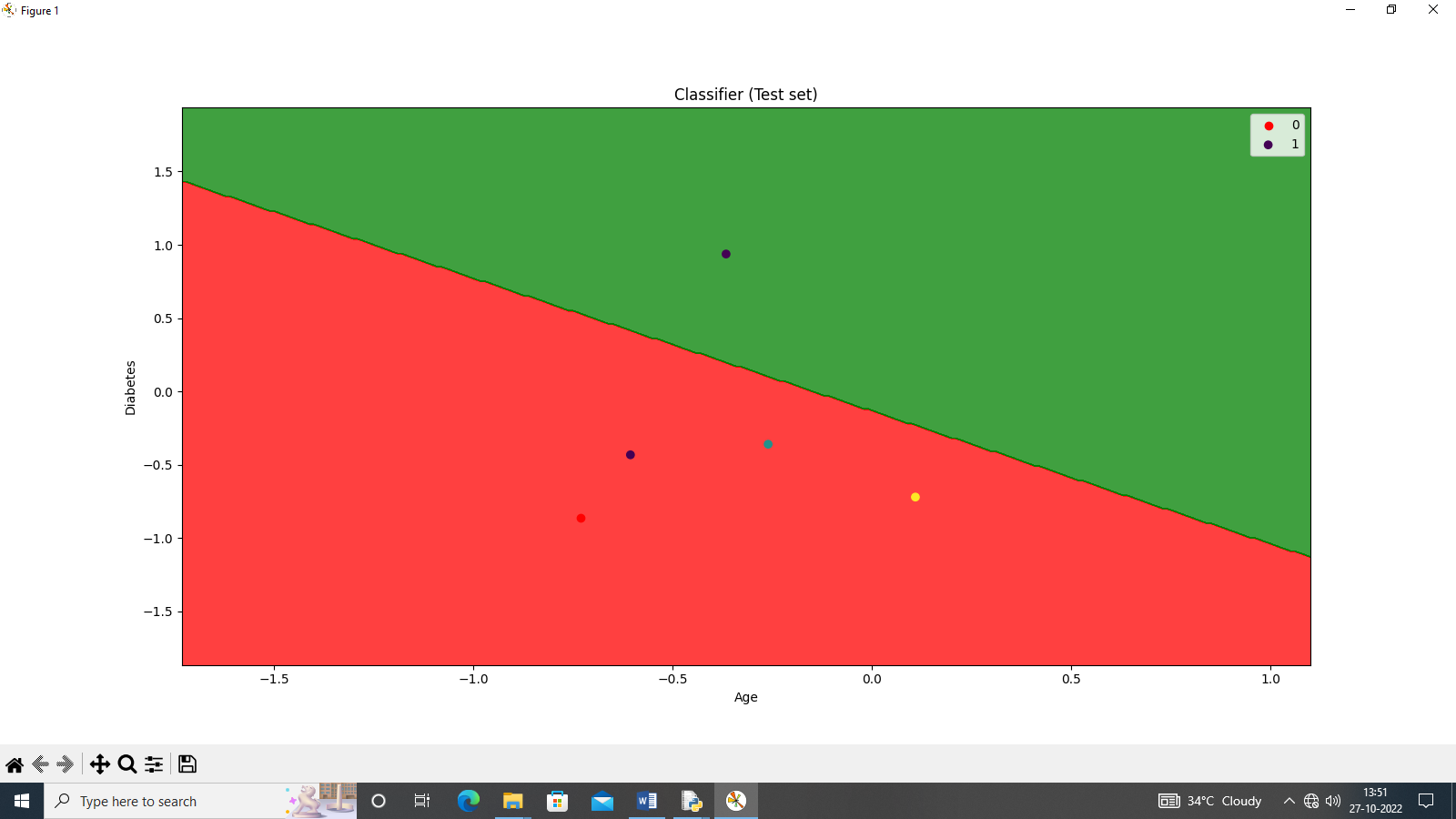
plt.title('Classifier (Test set)')

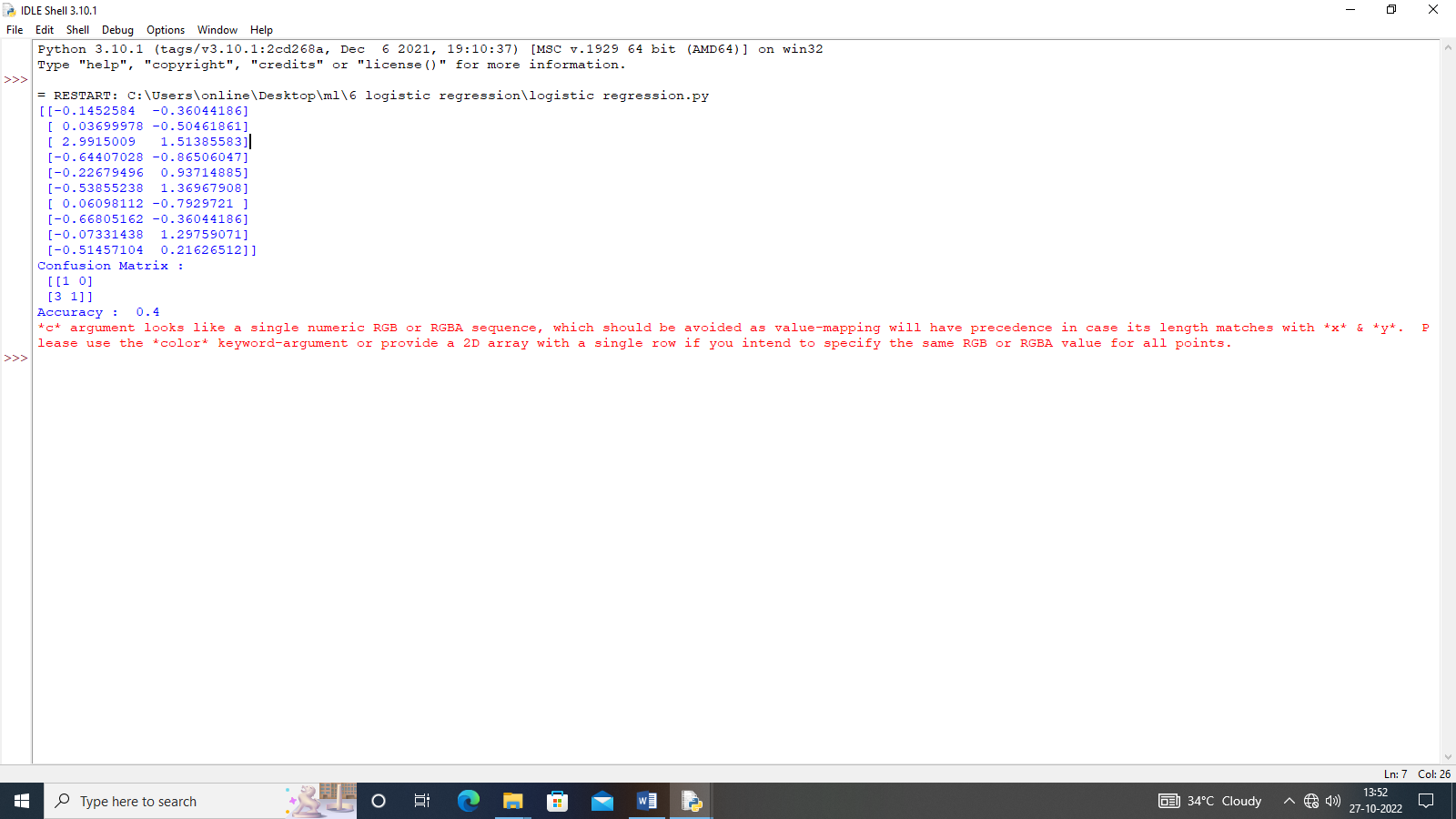
plt.xlabel('Age')

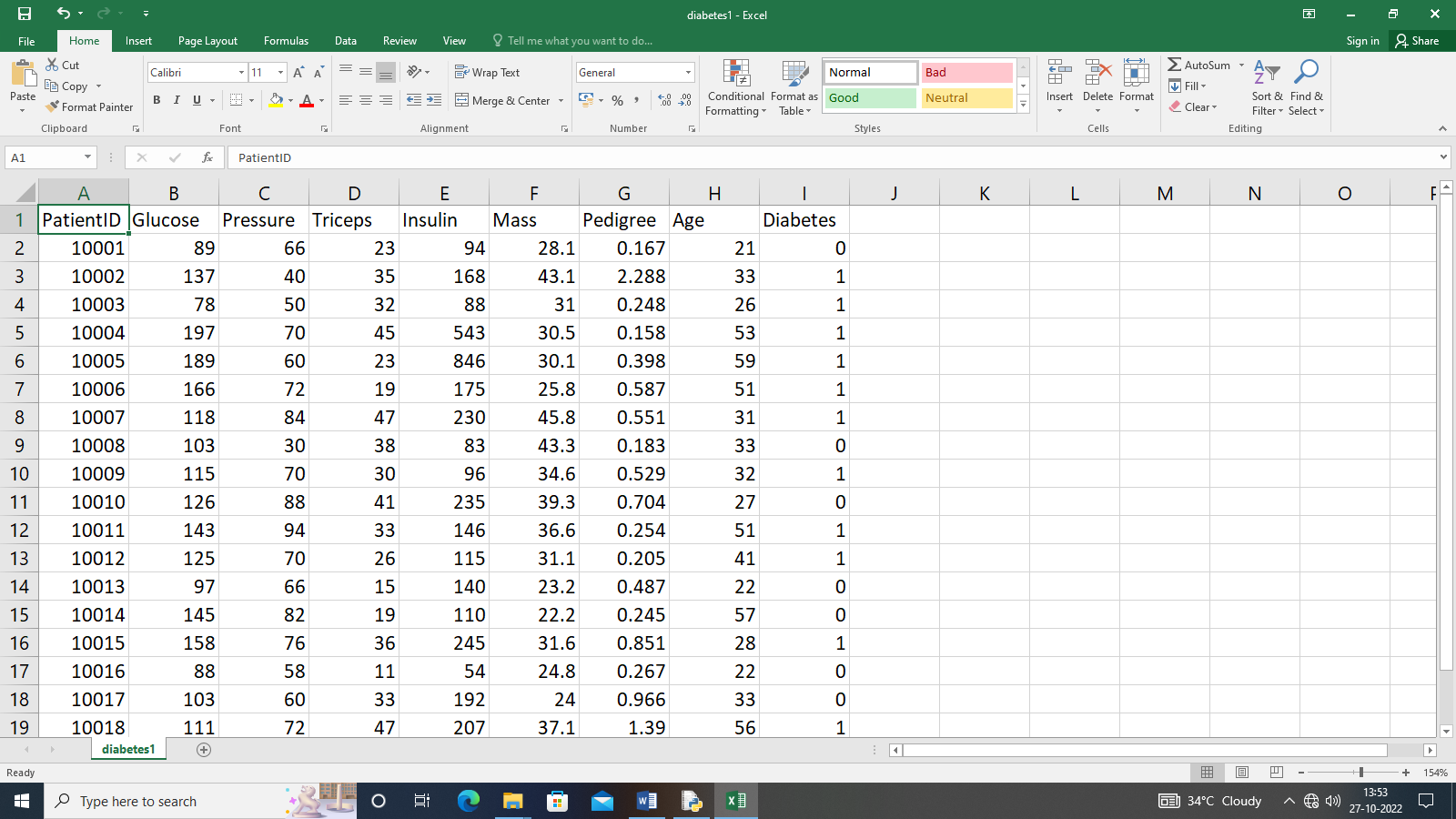
plt.ylabel('Diabetes')

plt.legend()

plt.show()







2.d.DECISION TREE

import pandas as pd

import math

import numpy as np

data = pd.read\_csv("3-dataset.csv")

features = [feat for feat in data]

features.remove("answer")

class Node:

def \_\_init\_\_(self):

self.children = []

self.value = ""

self.isLeaf = False

self.pred = ""

def entropy(examples):

pos = 0.0

neg = 0.0

for \_, row in examples.iterrows():

if row["answer"] == "yes":

pos += 1

else:

neg += 1

if pos == 0.0 or neg == 0.0:

return 0.0

else:

p = pos / (pos + neg)

n = neg / (pos + neg)

return -(p \* math.log(p, 2) + n \* math.log(n, 2))

def info\_gain(examples, attr):

uniq = np.unique(examples[attr])

gain = entropy(examples)

for u in uniq:

subdata = examples[examples[attr] == u]

sub\_e = entropy(subdata)

gain -= (float(len(subdata)) / float(len(examples))) \* sub\_e

return gain

def ID3(examples, attrs):

root = Node()

max\_gain = 0

max\_feat = ""

for feature in attrs:

gain = info\_gain(examples, feature)

if gain > max\_gain:

max\_gain = gain

max\_feat = feature

root.value = max\_feat

uniq = np.unique(examples[max\_feat])

for u in uniq:

subdata = examples[examples[max\_feat] == u]

if entropy(subdata) == 0.0:

newNode = Node()

newNode.isLeaf = True

newNode.value = u

newNode.pred = np.unique(subdata["answer"])

root.children.append(newNode)

else:

dummyNode = Node()

dummyNode.value = u

new\_attrs = attrs.copy()

new\_attrs.remove(max\_feat)

child = ID3(subdata, new\_attrs)

dummyNode.children.append(child)

root.children.append(dummyNode)

return root

def printTree(root: Node, depth=0):

for i in range(depth):

print("\t", end="")

print(root.value, end="")

if root.isLeaf:

print(" -> ", root.pred)

print()

for child in root.children:

printTree(child, depth + 1)

def classify(root: Node, new):

for child in root.children:

if child.value == new[root.value]:

if child.isLeaf:

print ("Predicted Label for new example", new," is:", child.pred)

exit

else:

classify (child.children[0], new)

root = ID3(data, features)

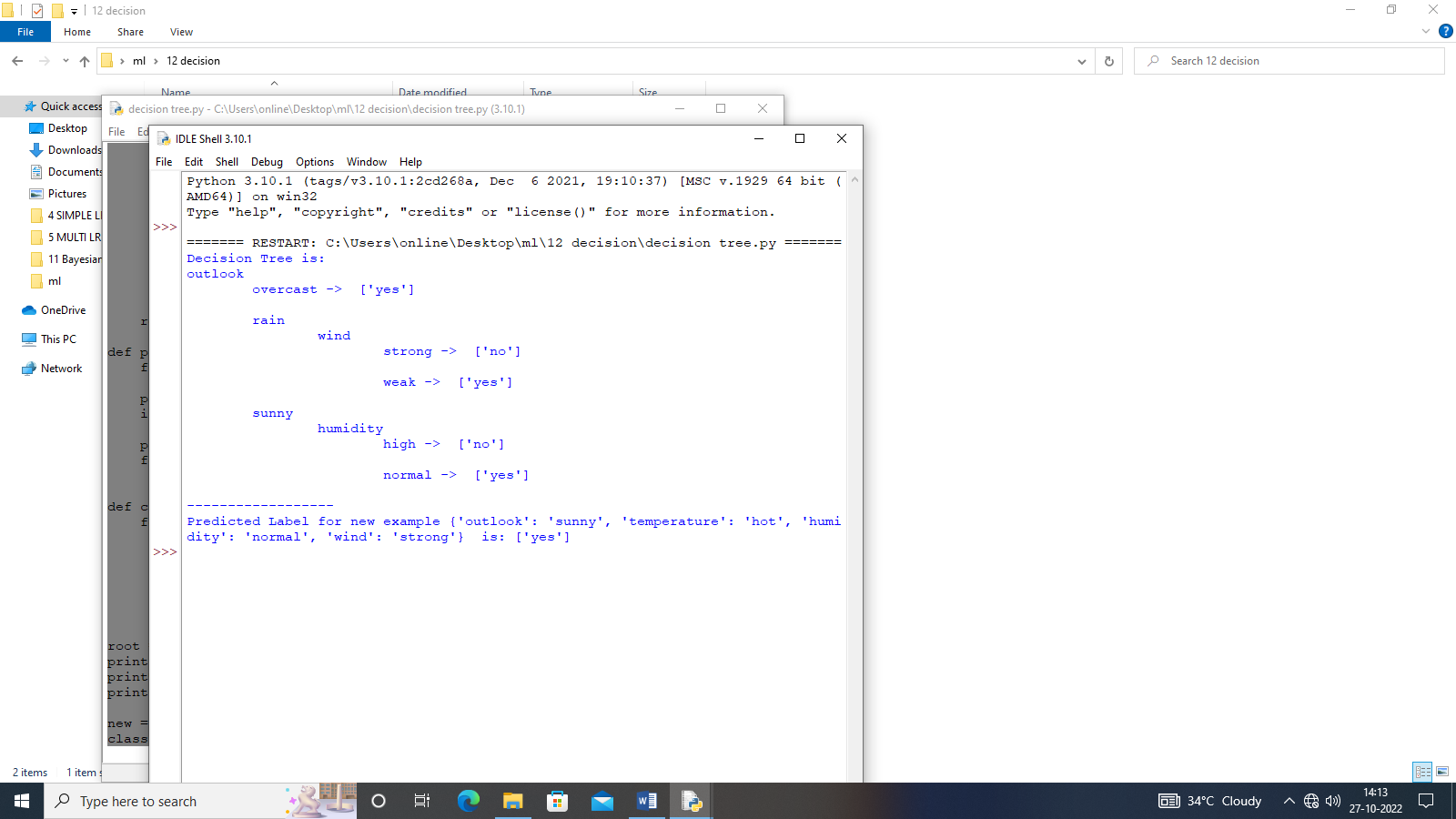
print("Decision Tree is:")

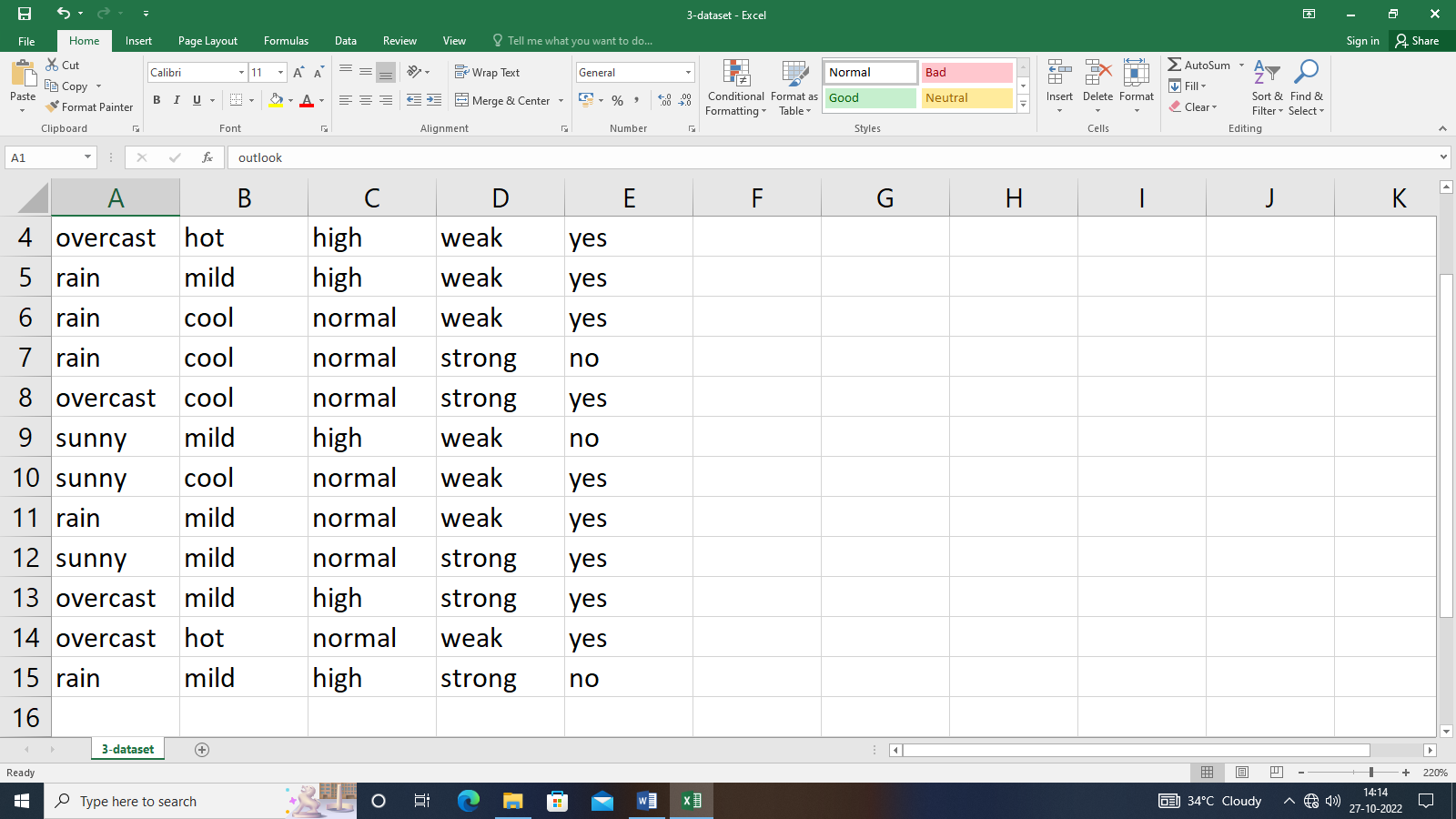
printTree(root)

print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}

classify (root, new)





2.e.Random forest

import pandas as pd

data=pd.read\_csv("HeartDisease1.csv")

X =data.iloc[:,[1,2,3,4,5,6,7,8,9,10,11,12]].values

y =data.iloc[:,13].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()

rfc.fit(X\_train, y\_train)

y\_pred=rfc.predict(X\_test)

from sklearn import metrics

print("Classification Accuracy:", metrics.accuracy\_score(y\_test, y\_pred)\*100)

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

print(cm)

import seaborn as sn

from matplotlib import pyplot as plt

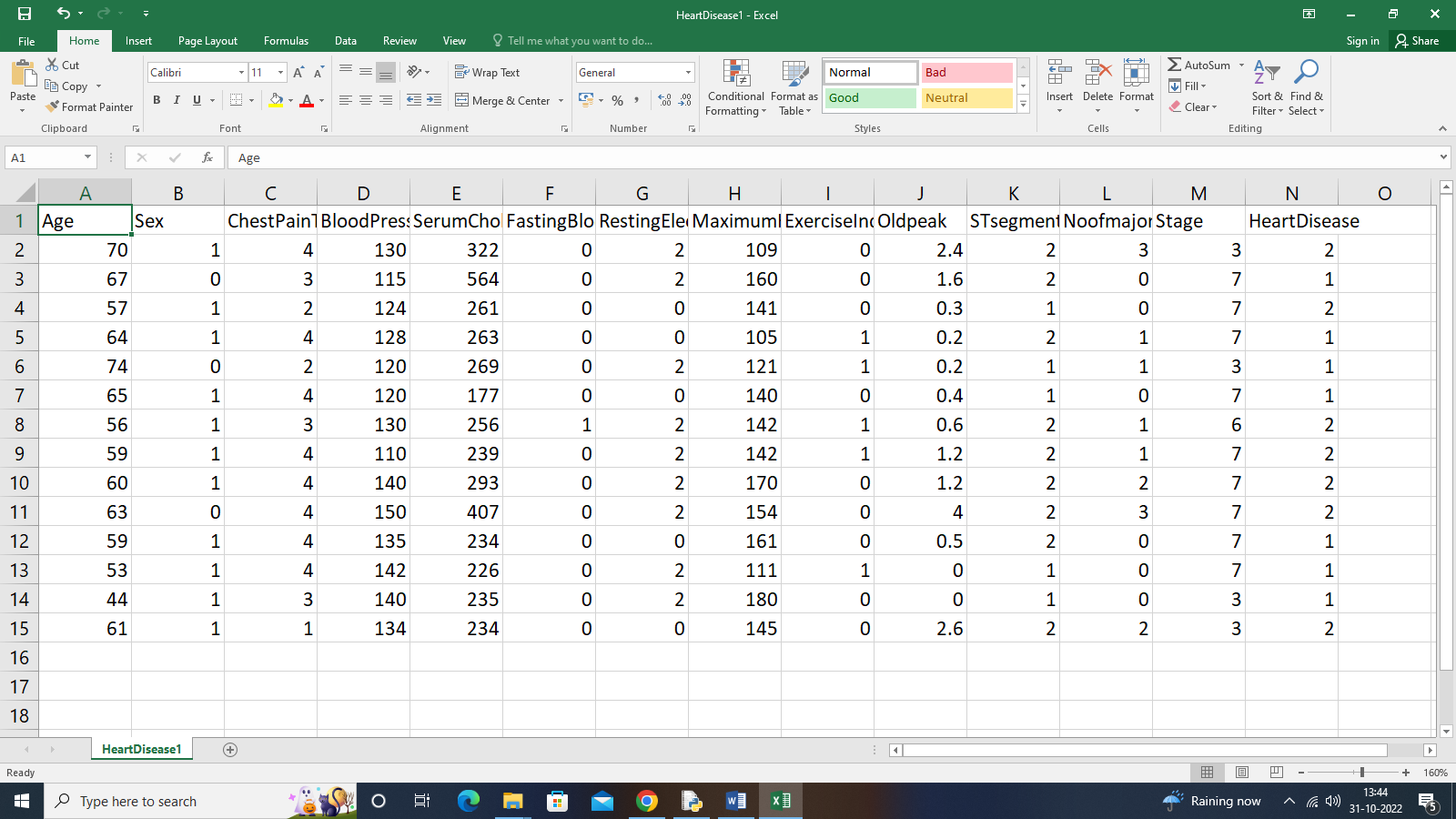
plt.figure(figsize=(5,4))

sn.heatmap(cm,annot=True)

plt.xlabel('Predicted value')

plt.ylabel('Actual value')

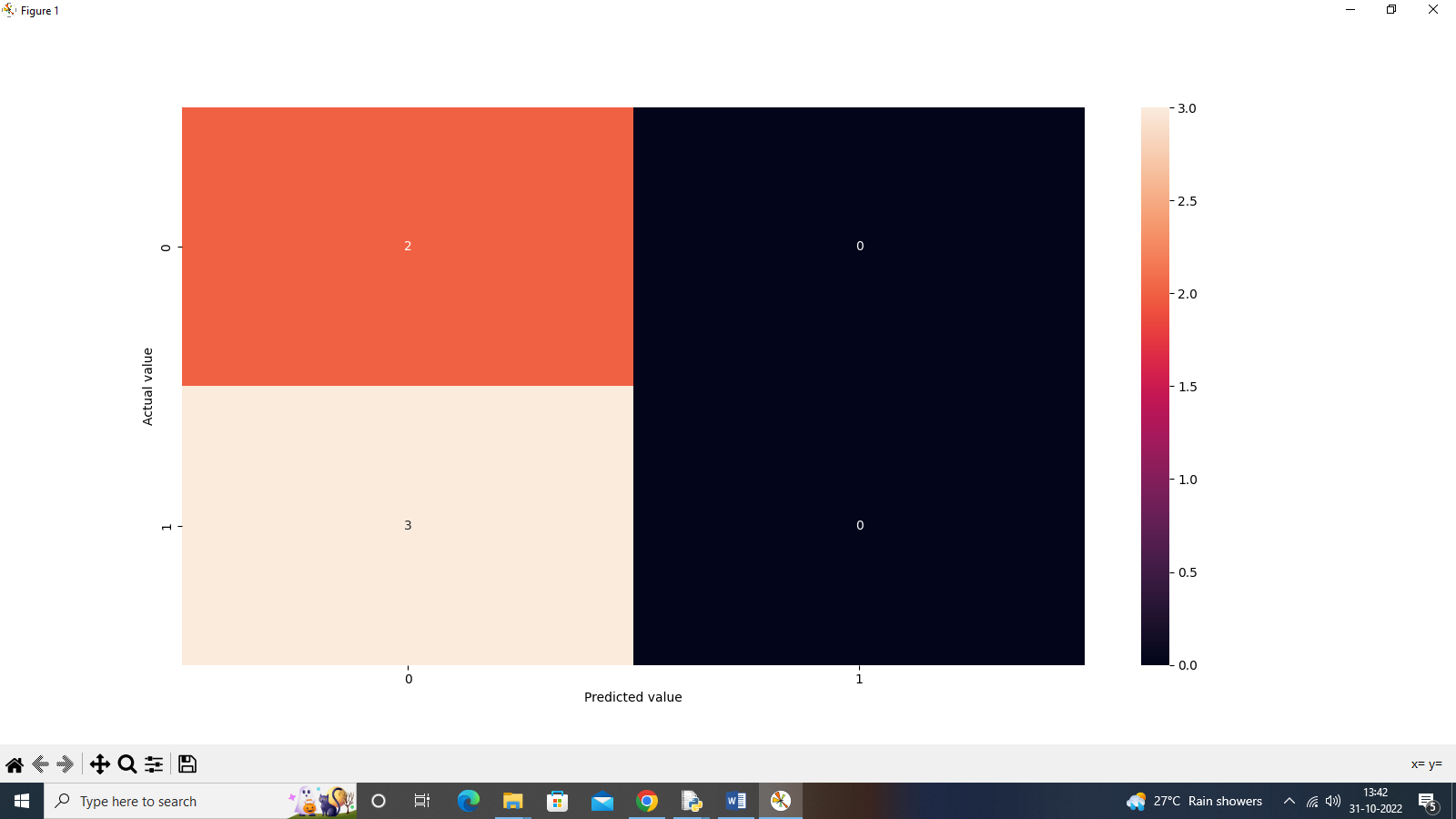
plt.show()



Classification Accuracy: 40.0

[[2 0]

[3 0]]



3.K-MEANS

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.preprocessing import MinMaxScaler

iris = pd.read\_csv("Iris1.csv")

x = iris.iloc[:, [ 1,2,3,4]].values

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)

kmeans.fit(x)

wcss.append(kmeans.inertia\_)

kmeans = KMeans(n\_clusters = 3, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)

y\_kmeans = kmeans.fit\_predict(x)

plt.scatter(x[y\_kmeans == 0, 0], x[y\_kmeans == 0, 1], s = 100, c = 'blue', label = 'Iris-setosa')

plt.scatter(x[y\_kmeans == 1, 0], x[y\_kmeans == 1, 1], s = 100, c = 'orange', label = 'Iris-versicolour')

plt.scatter(x[y\_kmeans == 2, 0], x[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:,1], s = 100, c = 'red', label = 'Centroids')

plt.legend()

plt.show()

