

UNIVERSITY OF PETROLEUM & ENERGY STUDIES

School of Computer Science

Dehradun

LAB FILE

Programme : B.Tech (CSE+AIML)

Course : Machine Learning Lab

Subject Code : CSAI2101

No. of credits : 1

Semester : IV

Session : Jan 2022 – July 2022

Batch : 2020-2024

Submitted By:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | SAP-ID | Roll No. | Batch |
| Shudhanshu Shrotriya | 500086221 | R2142201891 | B4(Honors) |

Submitted To:

Dr Gopal Phartiyal

Associate Professor -SoCS



**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Experiments** | **Program Exercise** | **Signature** |
| 1. | Experiment 1 | Simple linear regression with one-dimensional input and output. |  |
| 2. | Experiment 2 | Linear regression with X and Y as vectors |  |
| 3. | Experiment 3 | Linear regression ( & Basis Function) with R2 vs Noise plots |  |
| 4. | Experiment 4 | Linear, Ridge, Lasso, and Elastic Net regression with R2 vs Noise plots |  |
| 5. | Experiment 5 | Logistic regression |  |
| 6. | Experiment 6 | Classification for 2 classes |  |
| 7. | Experiment 7 | Classification for Multi Classes |  |
| 8. | Experiment 8 | Decision tree |  |
| 9. | Experiment 9 | Random forest, Adaboost classification and Adaboost regression |  |
| 10. | Experiment 10 | Neural Network |  |
| 11. | Experiment 11 | SVM classification |  |
| 12. | Experiment 12 | K-Means Clustering |  |

**Experiment-1**

**Objective:** To perform the simple linear regression with one-dimensional input and output.

**Code:**

# -\*- coding: utf-8 -\*-

"""

@author: shudh

"""

#%%

import numpy as np

from sklearn.datasets import make\_regression

import matplotlib.pyplot as plt

#%%

X , t = make\_regression (100, 1, shuffle = True , bias = 0.0 ,noise = 20, random\_state= 7)

#%%

plt.scatter(X,t)

#%%

mean\_x = np.mean(np.squeeze(X))

mean\_t = np.mean(t)

std\_x = np.std(X)

std\_t = np.std(t)

#%%

d\_x= X - mean\_x

d\_t= t - mean\_t

num = np.sum(np.squeeze(d\_x) \* d\_t)

deno = np.sum(np.squeeze(d\_x) \* np.squeeze(d\_x))

B1= num / deno

B0= mean\_t - (B1 \* mean\_x)

#%%

print(B1)

print(B0)

#%%

y= B0 + B1 \* X

print(y)

print(t)

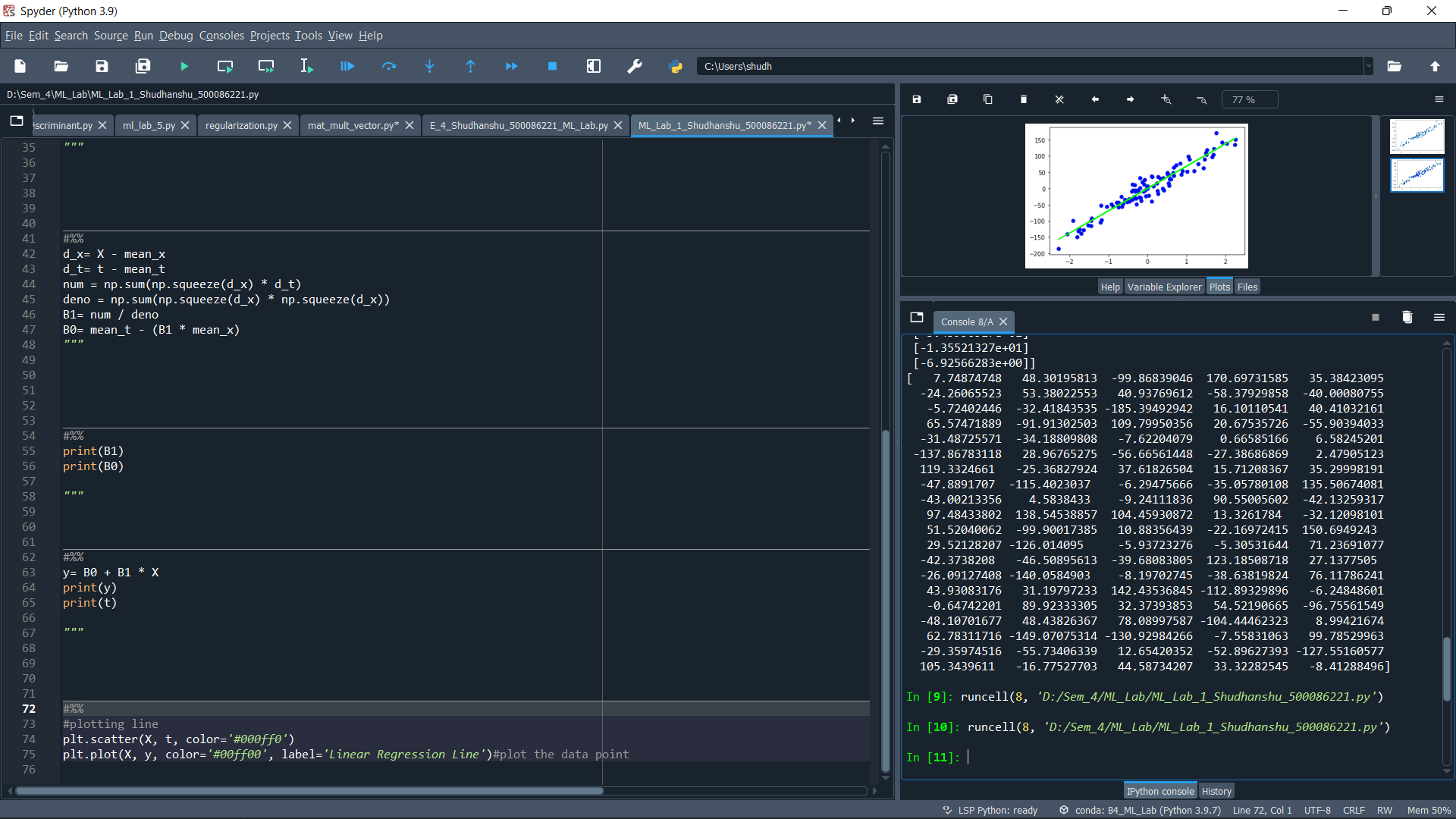
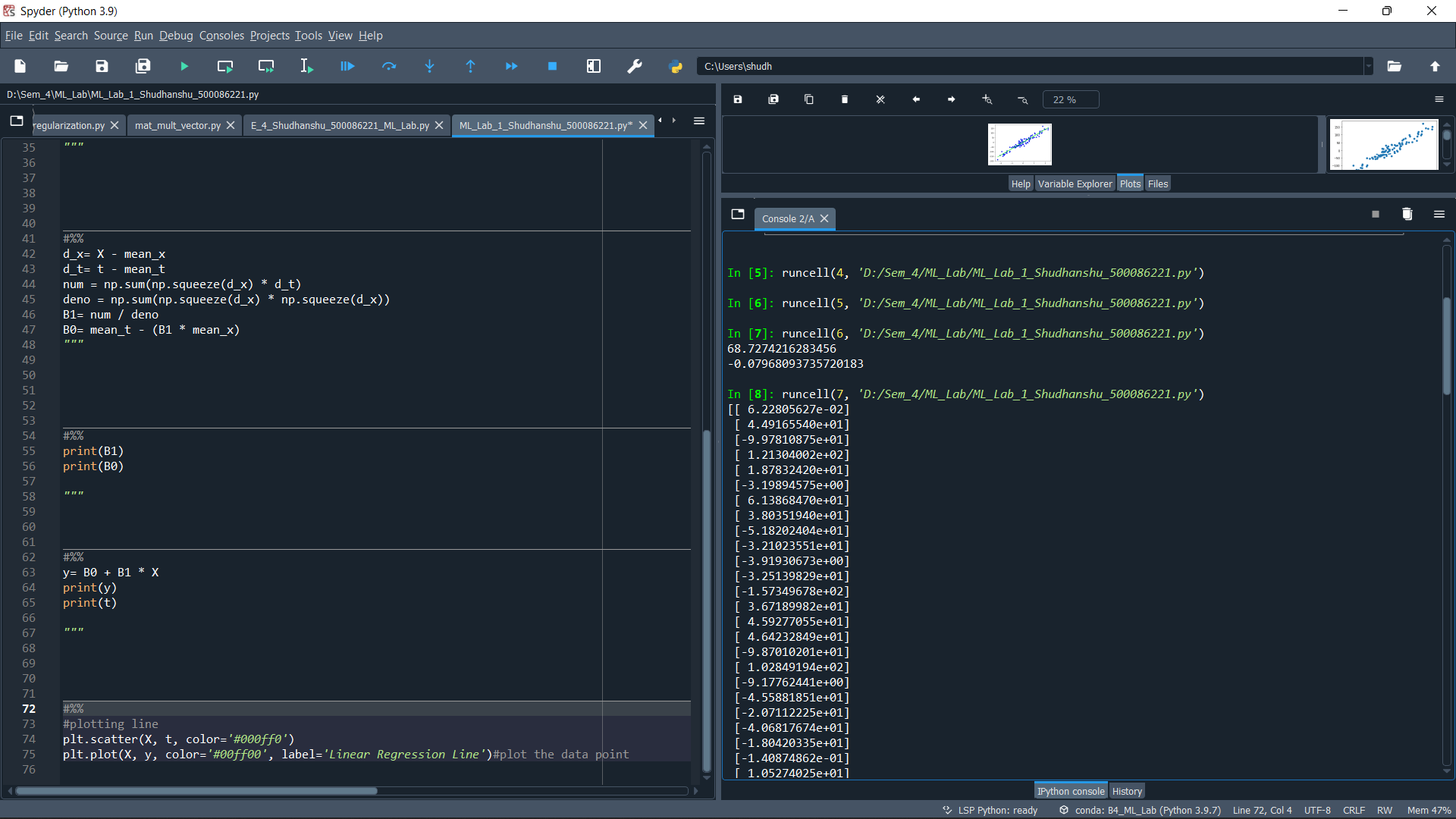
#%%

#plotting line

plt.scatter(X, t, color='#000ff0')

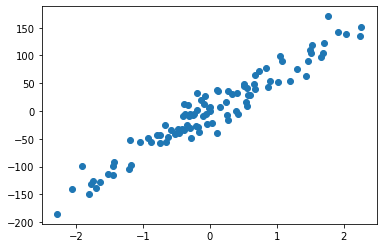
plt.plot(X, y, color='#00ff00', label='Linear Regression Line')#plot the data point

**Output:**

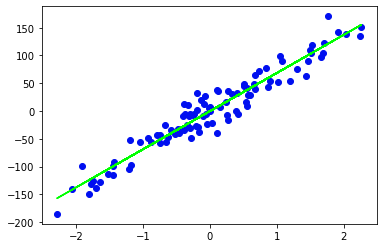


**Plots:**

**Original Dataset:**



**Linear\_Regression:**



**Experiment-2**

**Objective:** To perform the linear regression with X and Y as the vectors.

**Code:**

# -\*- coding: utf-8 -\*-

"""

@author: shudh

"""

import numpy as np

from sklearn.datasets import make\_regression

#%%

X , t = make\_regression (100,5, shuffle = True , bias = 0.0 , n\_targets=3 ,noise = 10, random\_state= 4) ;

#%%

x0=np.ones((100,1), dtype=int);

new\_x = np.concatenate((x0,X), axis = 1);

tr=new\_x.transpose();

m3 = np. dot(tr,new\_x);

#%%

i = np.linalg.inv(m3);

m4 = np. dot(i,tr);

m5=np.dot(m4,t);

transp=m5.transpose();

#%%

print("the value of W is:-");

print(m5);

#%%

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

w = model.fit(new\_x,t)

#%%

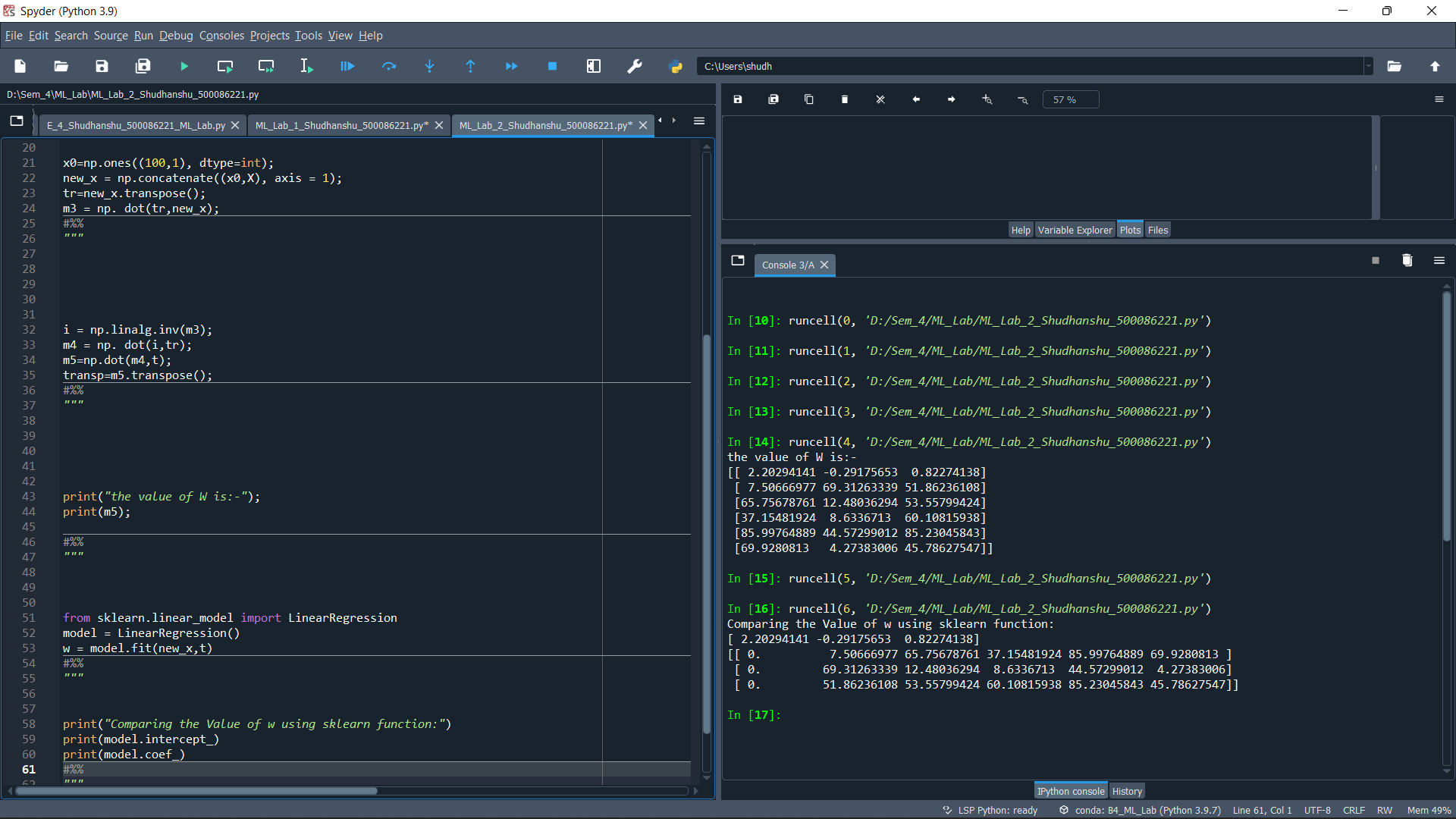
print("Comparing the Value of w using sklearn function:")

print(model.intercept\_)

print(model.coef\_)

#%%

**Output:**



**Experiment-3**

**Objective:** To perform the linear regression with Basis Function and plot the R2 score vs Noise plots for training and test data.

**Code:**

from sklearn.datasets import make\_regression

X,t = make\_regression(100, 5, shuffle = True, bias = 0, noise = 77, random\_state = 9)

X\_mean = X.mean()

t\_mean = t.mean()

X\_std = X.std()

t\_std = t.std()

print(X)

print(t)

import numpy as np

x0 = np.ones((100,1))

new\_X = np.concatenate((x0, X), axis =1)

new\_X.shape

X\_t = np.transpose(new\_X)

W\_inv = np.dot(X\_t,new\_X)

W\_1 = np.linalg.inv(W\_inv)

W\_2 = np.dot(X\_t,t)

W = np.dot(W\_1, W\_2)

print(W)

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

w = model.fit(X,t)

model.coef\_

#%%

X\_vec,t\_vec = make\_regression(100, 3, shuffle = True, bias = 0, noise = 40, random\_state = 9, n\_targets = 3)

print(X\_vec)

print(t\_vec)

x0 = np.ones((100,1))

new\_X = np.concatenate((x0, X\_vec), axis =1)

X\_vec\_t = np.transpose(new\_X)

W\_inv = np.dot(X\_vec\_t,new\_X)

W\_1 = np.linalg.inv(W\_inv)

W\_2 = np.dot(W\_1, X\_vec\_t)

W = np.dot(W\_2, t\_vec)

print(W)

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

w = model.fit(X\_vec,t\_vec)

model.coef\_

model.intercept\_

#%%

from sklearn.metrics import r2\_score

y\_predict = model.predict(X\_vec)

score = r2\_score(t\_vec, y\_predict)

print(score)

#%%

noise\_lt = [0, 10, 30, 50, 70, 90, 100]

r2\_lt = [1.0, 0.9911277146802907, 0.9233227047988551, 0.8082131795660822, 0.6769299465749685, 0.5533416629728133, 0.4983050464146044]

import matplotlib.pyplot as plt

plt.plot(noise\_lt, r2\_lt, label = 'Relation between Noise and R2\_score')

plt.legend()

plt.show()

#%%

import sklearn.model\_selection as model\_selection

X\_vec,t\_vec = make\_regression(100, 3, shuffle = True, bias = 0, noise = 40, random\_state = 9, n\_targets = 3)

X\_train, X\_test, Y\_train, Y\_test = model\_selection.train\_test\_split(X\_vec, t\_vec, test\_size=0.3)

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

w = model.fit(X\_train,Y\_train)

model.coef\_

model.intercept\_

y\_pred\_train = model.predict(X\_train)

score\_train = r2\_score(y\_pred\_train, Y\_train)

print(score\_train)

#%%

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

score\_test = r2\_score(y\_pred\_test, Y\_test)

print(score\_test)

#%%

noise\_list = [0, 10, 30, 50, 70 ,90 ,100]

score\_train\_list = [1,0.9892422892838316,0.9123535974919882,0.7851886218805498,0.4866233488454797,0.2540158327162871,-0.07161444865168864]

score\_test\_list = [1,0.9927037387643658,0.9000725805417417,0.698401043958205,0.5189450019757639,0.2725673734949044,-0.2642270301805036]

plt.plot(noise\_list, score\_train\_list, label = 'T raining\_data')

plt.plot(noise\_list, score\_test\_list, label = 'T est\_data')

plt.legend(loc="upper right")

plt.show()

fig = plt.figure()

ax1 = fig.add\_subplot(221)

ax2 = fig.add\_subplot(222)

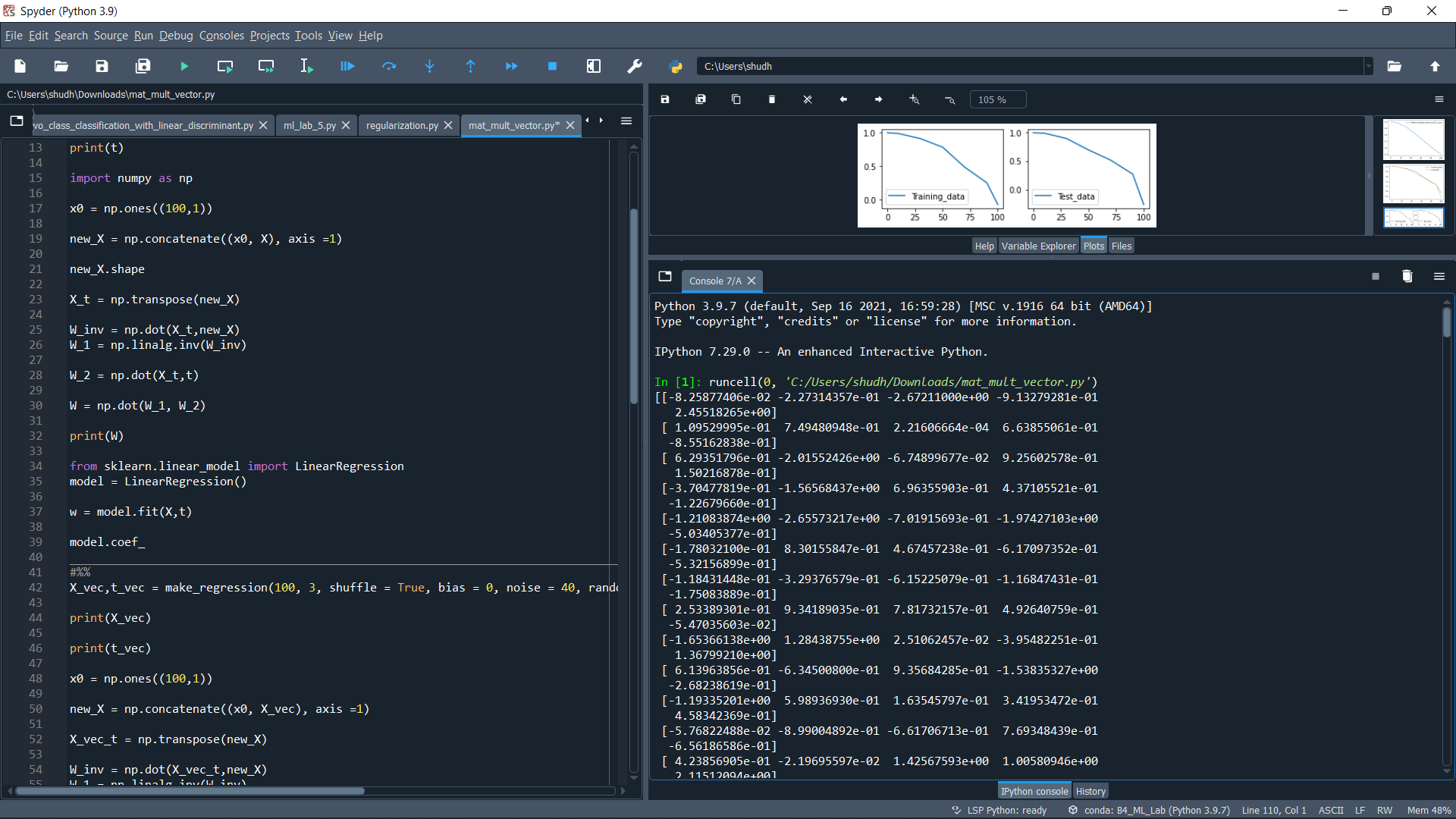
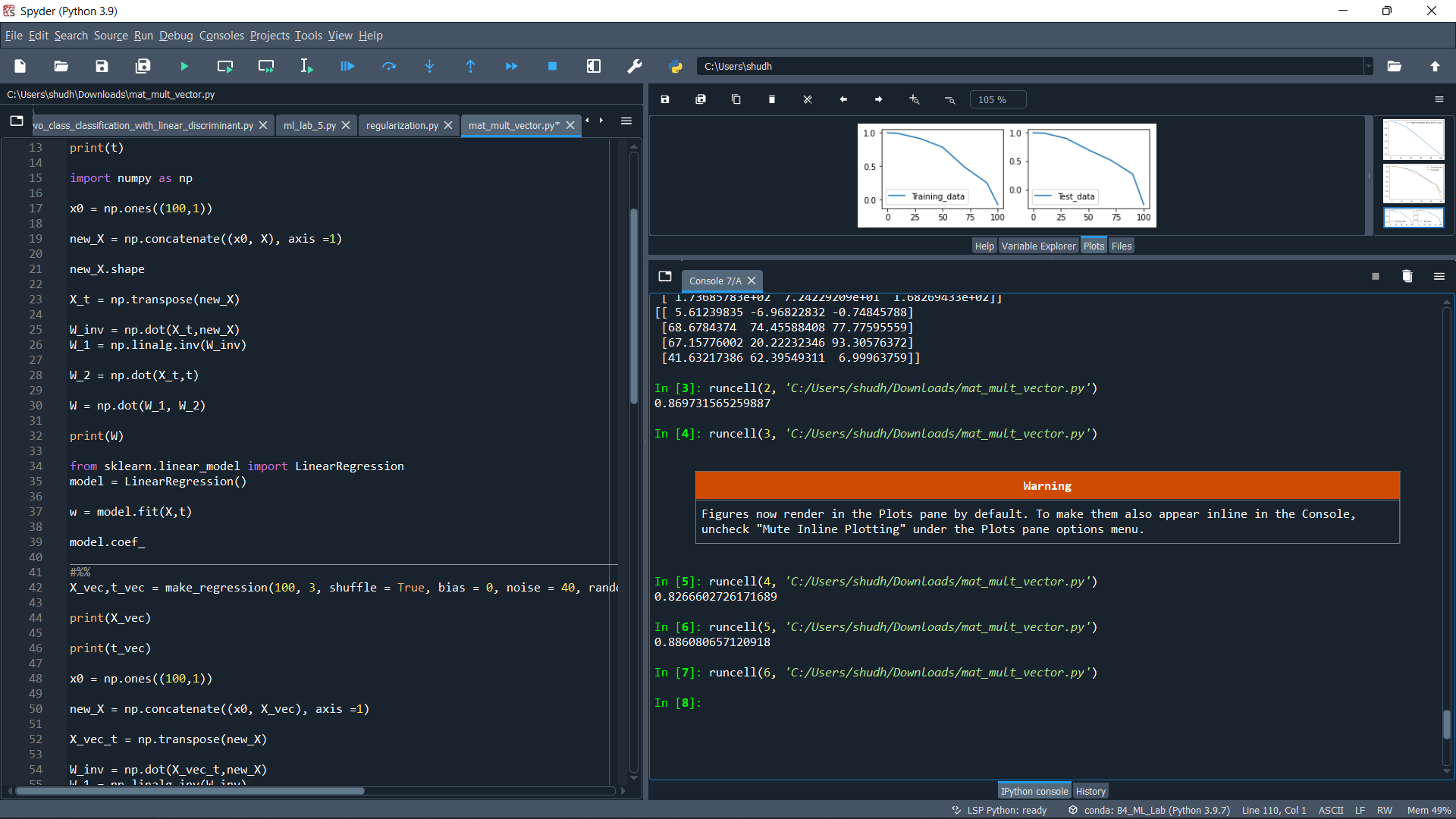
ax1.plot(noise\_list, score\_train\_list, label = 'Training\_data')

ax1.legend()

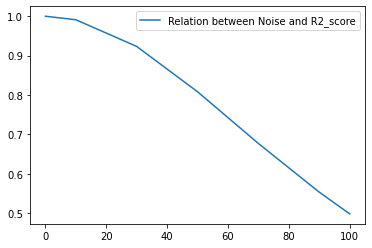
ax2.plot(noise\_list, score\_test\_list, label = 'Test\_data')

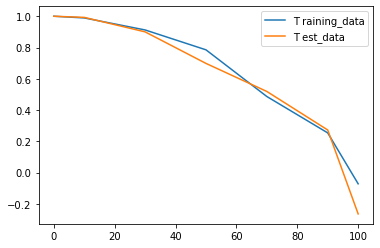
ax2.legend()

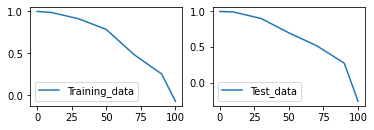
**Output:**



**Plots:**







**Experiment-4**

**Objective:** To perform the Linear regression, Ridge regression, Lasso regression, Elastic Net regression and plot the R2 score vs Noise plots for training and test data.

**Code:**

from sklearn.datasets import make\_regression

import matplotlib.pyplot as plt

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

from sklearn.metrics import r2\_score

from sklearn import linear\_model

#%%

X, y = make\_regression(100, 5, shuffle=True, random\_state=4, noise=30, bias=0.5)

x\_train, x\_test, y\_train, y\_test= train\_test\_split(X, y , test\_size=0.33)

Noise= [0,30,50,70,100]

#%%

#linear regression

from sklearn.linear\_model import LinearRegression

lin\_regressor = LinearRegression()

lin\_regressor.fit(x\_train, y\_train)

#%%

y\_pred\_train= lin\_regressor.predict(x\_train)

score\_train = r2\_score(y\_pred\_train, y\_train)

print(score\_train)

#%%

y\_pred\_test= lin\_regressor.predict(x\_test)

score\_test = r2\_score(y\_pred\_test, y\_test)

print (score\_test)

#%%

R2\_Score\_Train=[1.0,0.9338632188959995,0.7681307987876425,0.5638818102318661,0.19341579821414578]

R2\_Score\_Test=[1.0,0.8780416755714645,0.807709667337179,0.5547090201137526,0.2435815002631775]

plt.plot(Noise,R2\_Score\_Train, color ='red', label = 'T rain\_data')

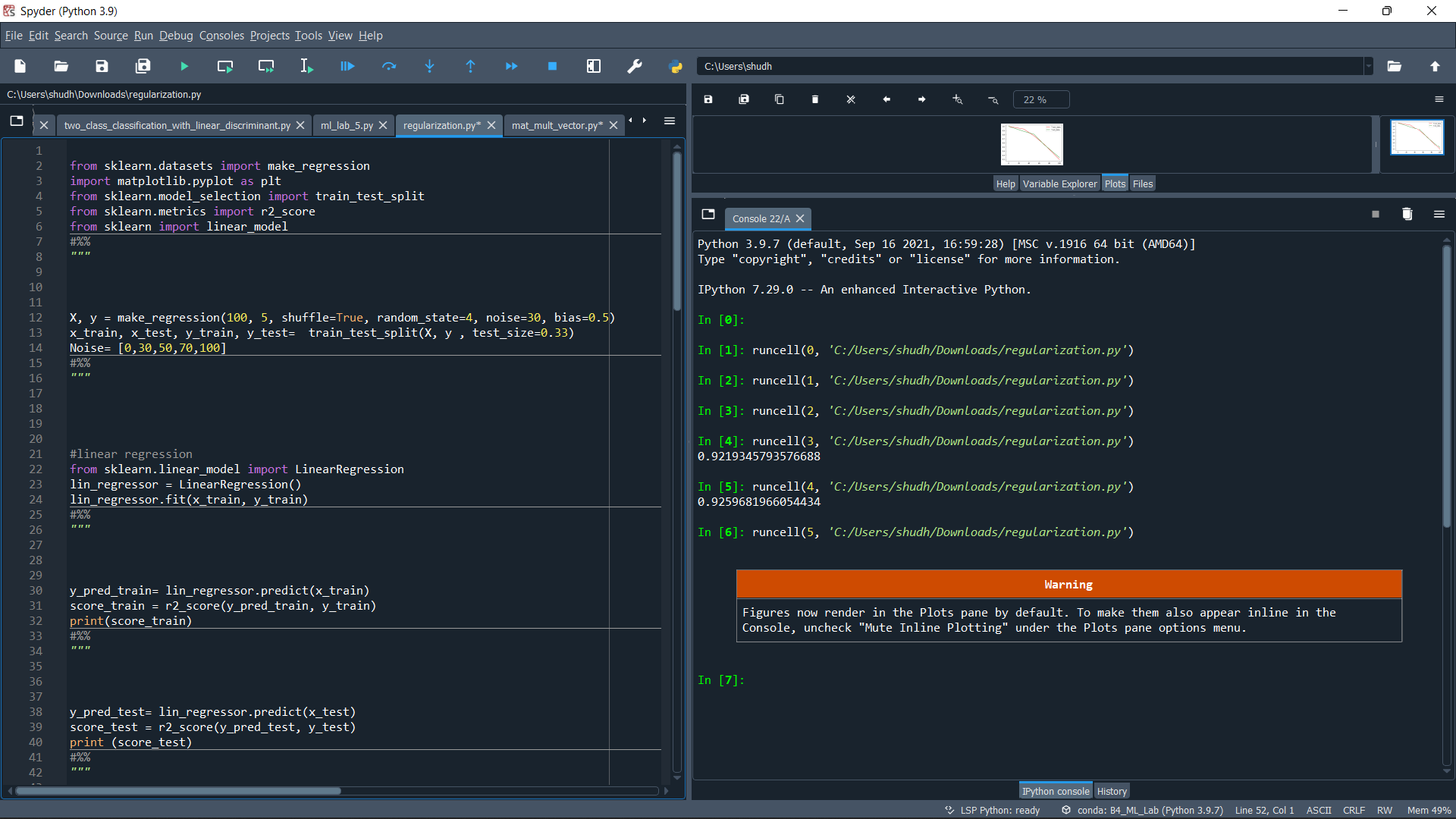
plt.plot(Noise,R2\_Score\_Test, color= 'green', label = 'T est\_data')

plt.legend()

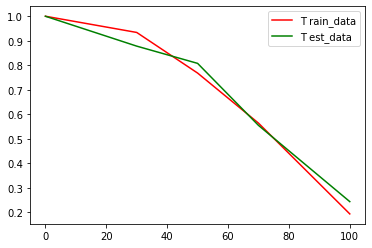
plt.show()

#%%

**Output:**



**Plots:**



#Ridge Regression

from sklearn.linear\_model import Ridge

ridge = Ridge()

ridge\_regressor = linear\_model.Lasso(alpha=0.1)

ridge\_regressor.fit(x\_train,y\_train)

#%%

y\_pred\_train= ridge\_regressor.predict(x\_train)

score\_train = r2\_score(y\_pred\_train, y\_train)

print(score\_train)

#%%

y\_pred\_test= ridge\_regressor.predict(x\_test)

score\_test = r2\_score(y\_pred\_test, y\_test)

print (score\_test)

#%%

R2\_Score\_Train=[0.999995024924167,0.8966109961110564,0.7709951023565671,0.6109128967502577,0.10285665366728136]

R2\_Score\_Test=[0.9999960410078167,0.9299234763118096,0.7625033153069063,0.4257289171127968,-0.3506965355648457]

plt.plot(Noise,R2\_Score\_Train, color ='red', label = 'T rain\_data')

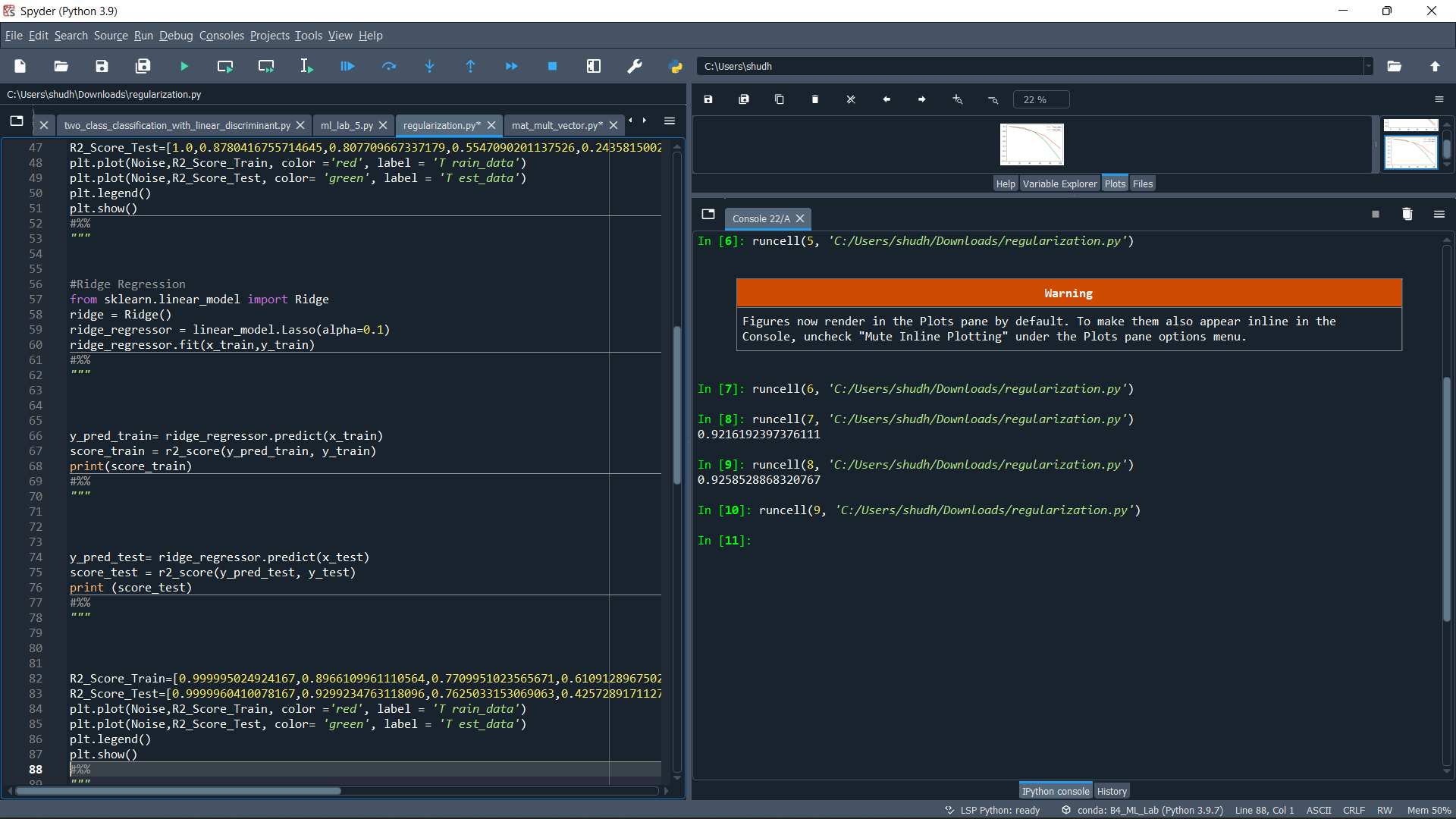
plt.plot(Noise,R2\_Score\_Test, color= 'green', label = 'T est\_data')

plt.legend()

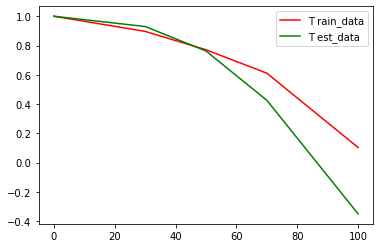
plt.show()

#%%

**Output:**



**Plots:**



#Lasso Regression

from sklearn.linear\_model import Lasso

lasso = Lasso()

lasso\_regressor = linear\_model.Lasso(alpha=0.1)

lasso\_regressor.fit(x\_train,y\_train)

#%%

y\_pred\_train= lasso\_regressor.predict(x\_train)

score\_train = r2\_score(y\_pred\_train, y\_train)

print(score\_train)

#%%

y\_pred\_test= lasso\_regressor.predict(x\_test)

score\_test = r2\_score(y\_pred\_test, y\_test)

print (score\_test)

#%%

R2\_Score\_Train=[0.999993912857805,0.9215327005213542,0.7641070653813703,0.5207553752959468,0.18465374871908924]

R2\_Score\_Test=[0.999992817349997,0.9141181129090268,0.7464025562414421,0.5136900695122311,-0.2208005962922286]

plt.plot(Noise,R2\_Score\_Train, color ='red', label = 'T rain\_data')

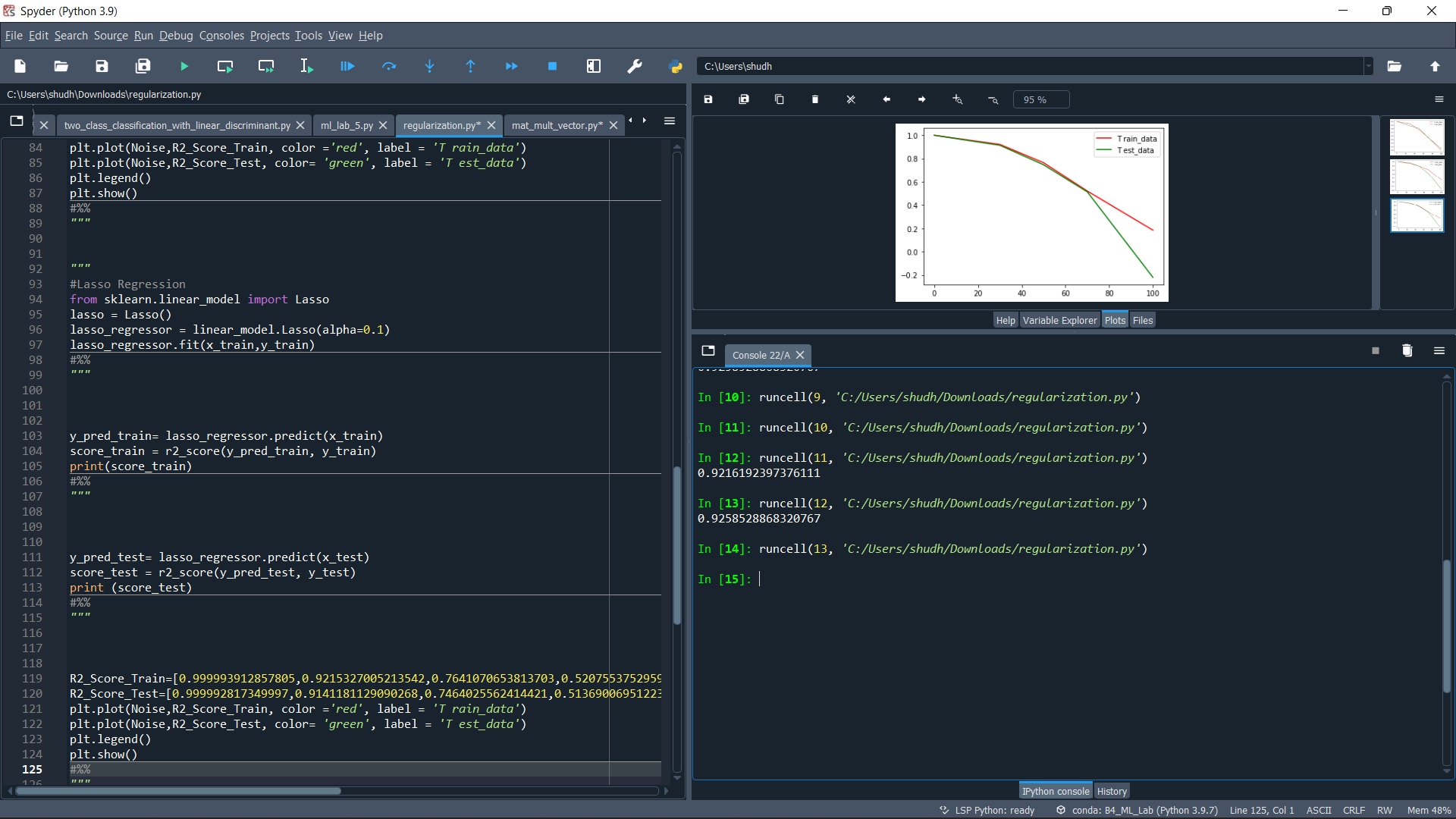
plt.plot(Noise,R2\_Score\_Test, color= 'green', label = 'T est\_data')

plt.legend()

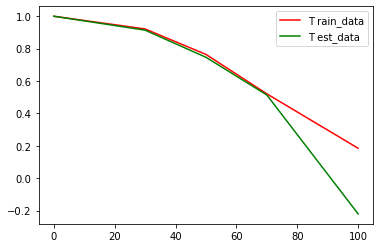
plt.show()

#%%

**Output:**



**Plots:**



#Elastic Net Regression

from sklearn.linear\_model import ElasticNet

elastic= ElasticNet()

elastic\_regressor = linear\_model.ElasticNet()

elastic\_regressor.fit(x\_train,y\_train)

#%%

y\_pred\_train= elastic\_regressor.predict(x\_train)

score\_train = r2\_score(y\_pred\_train, y\_train)

print(score\_train)

#%%

y\_pred\_test= elastic\_regressor.predict(x\_test)

score\_test = r2\_score(y\_pred\_test, y\_test)

print (score\_test)

#%%

R2\_Score\_Train=[0.6816953388621252,0.4908848531758495,0.2002958355669644,-0.05786389200116937,-0.7035044333362195]

R2\_Score\_Test=[0.6590656871439343,0.18431401702049677,-0.3858669940297983,-0.05786389200116937,-2.0147826468034955]

plt.plot(Noise,R2\_Score\_Train, color ='red', label = 'T rain\_data')

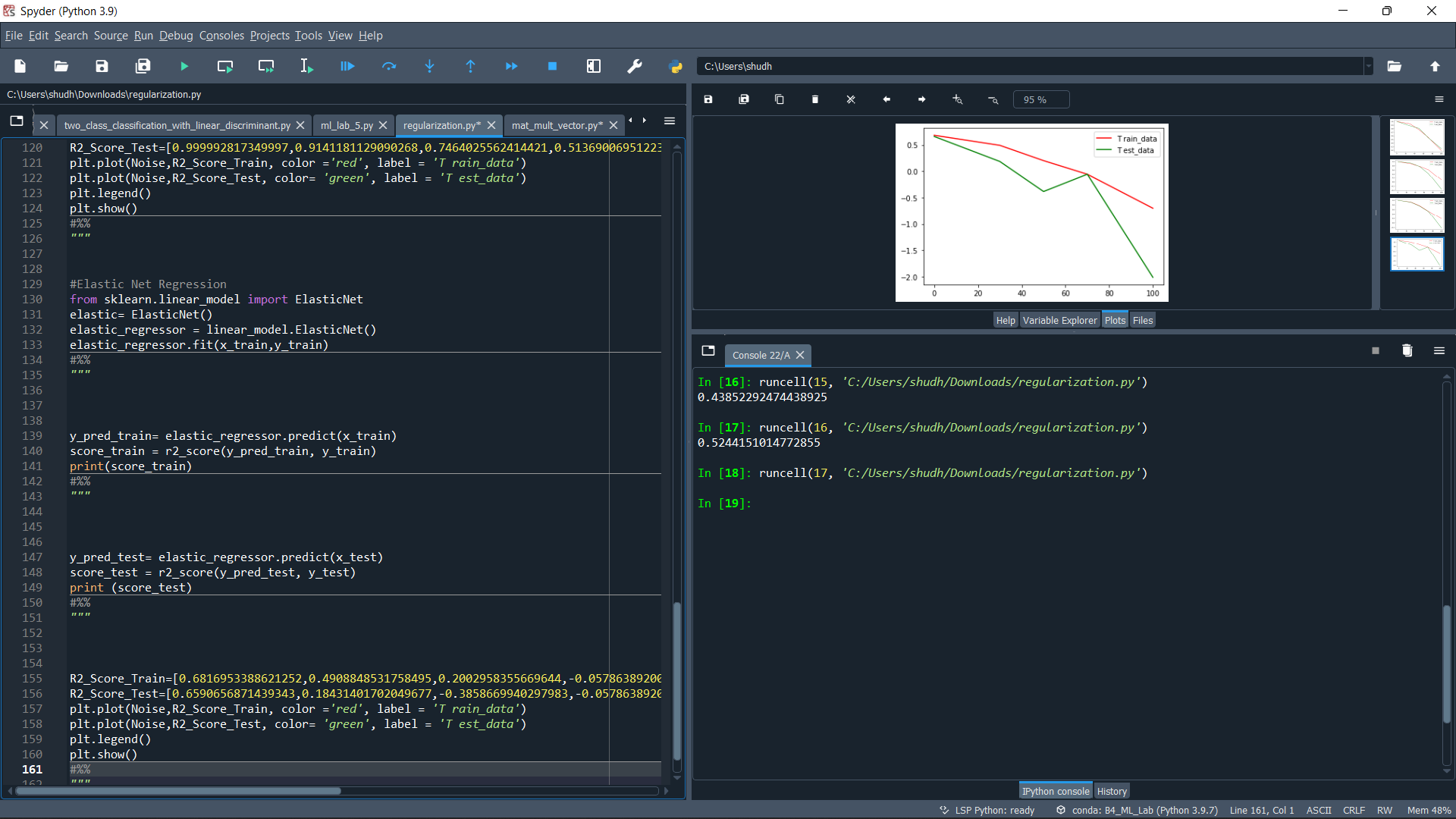
plt.plot(Noise,R2\_Score\_Test, color= 'green', label = 'T est\_data')

plt.legend()

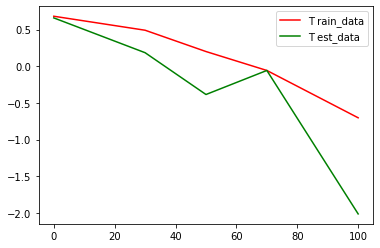
plt.show()

#%%

**Output:**



**Plots:**



**Experiment-5**

**Objective:** To perform Logistic Regression, also interpret it’s accuracy.

**Code:**

#%%

import numpy as np

from sklearn.datasets import make\_classification

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

#%%

X,t= make\_classification(100, 5, n\_classes = 2, random\_state= 15, n\_informative = 2, n\_clusters\_per\_class = 1)

#%%

X\_train, X\_test, t\_train, t\_test = train\_test\_split(X , t, test\_size=0.2)

X0=np.ones((80,1))

new\_x = np.concatenate((X0,X\_train), axis = 1)

X\_transpose = new\_x.transpose()

X\_multiply = X\_transpose.dot(new\_x)

X\_inverse = np.linalg.inv(X\_multiply)

X\_multiply1 = X\_inverse.dot(X\_transpose)

w = X\_multiply1.dot(t\_train)

#%%

Y\_model\_train = new\_x.dot(w)

print(Y\_model\_train)

#%%

x\_0 = np.ones((20,1))

new\_x\_test = np.concatenate((x\_0, X\_test), axis = 1)

y = np.dot(new\_x\_test, w)

print(y)

#%%

array = []

for i in y:

if i>=0:

array.append(1)

else:

array.append(0)

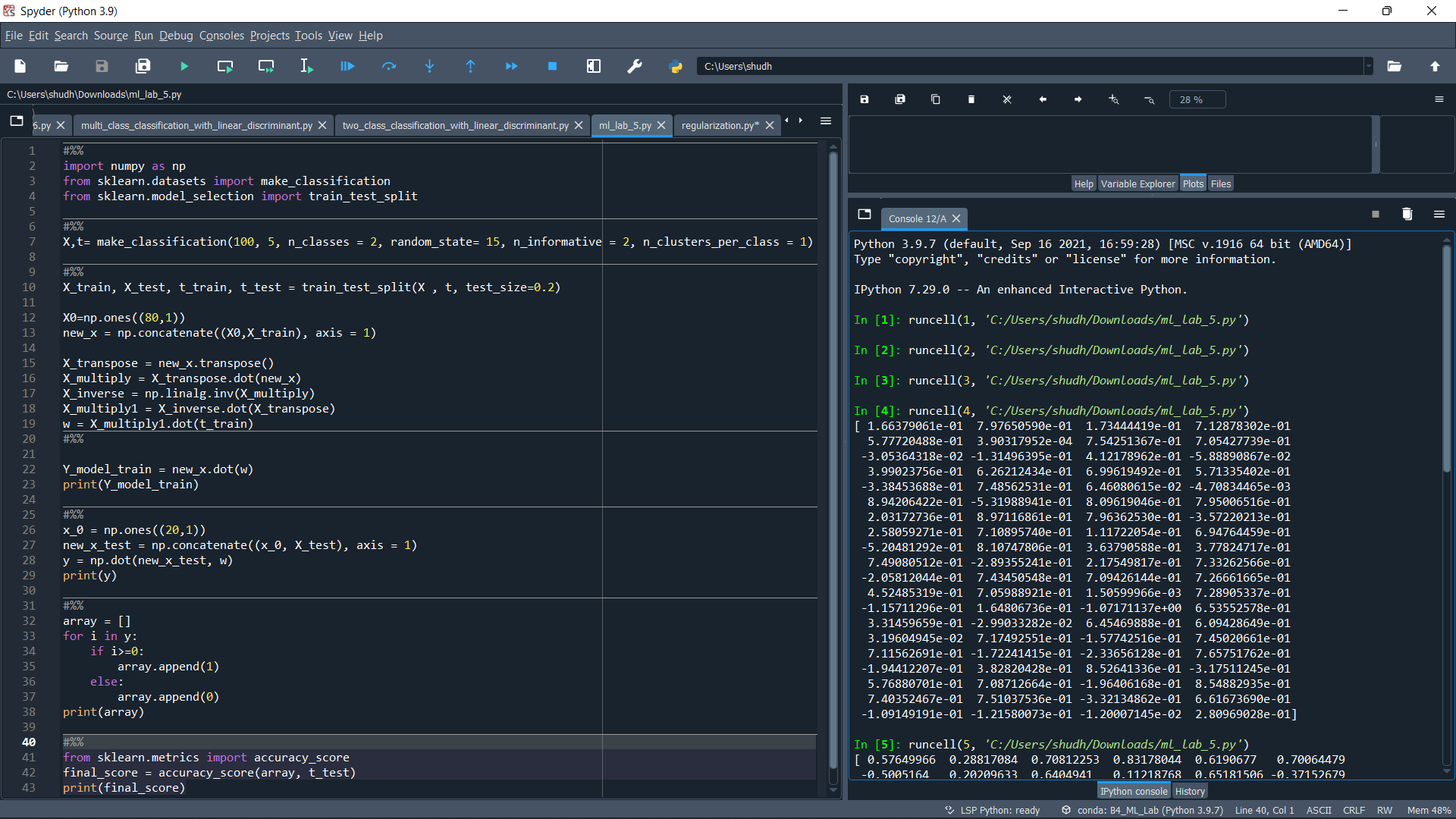
print(array)

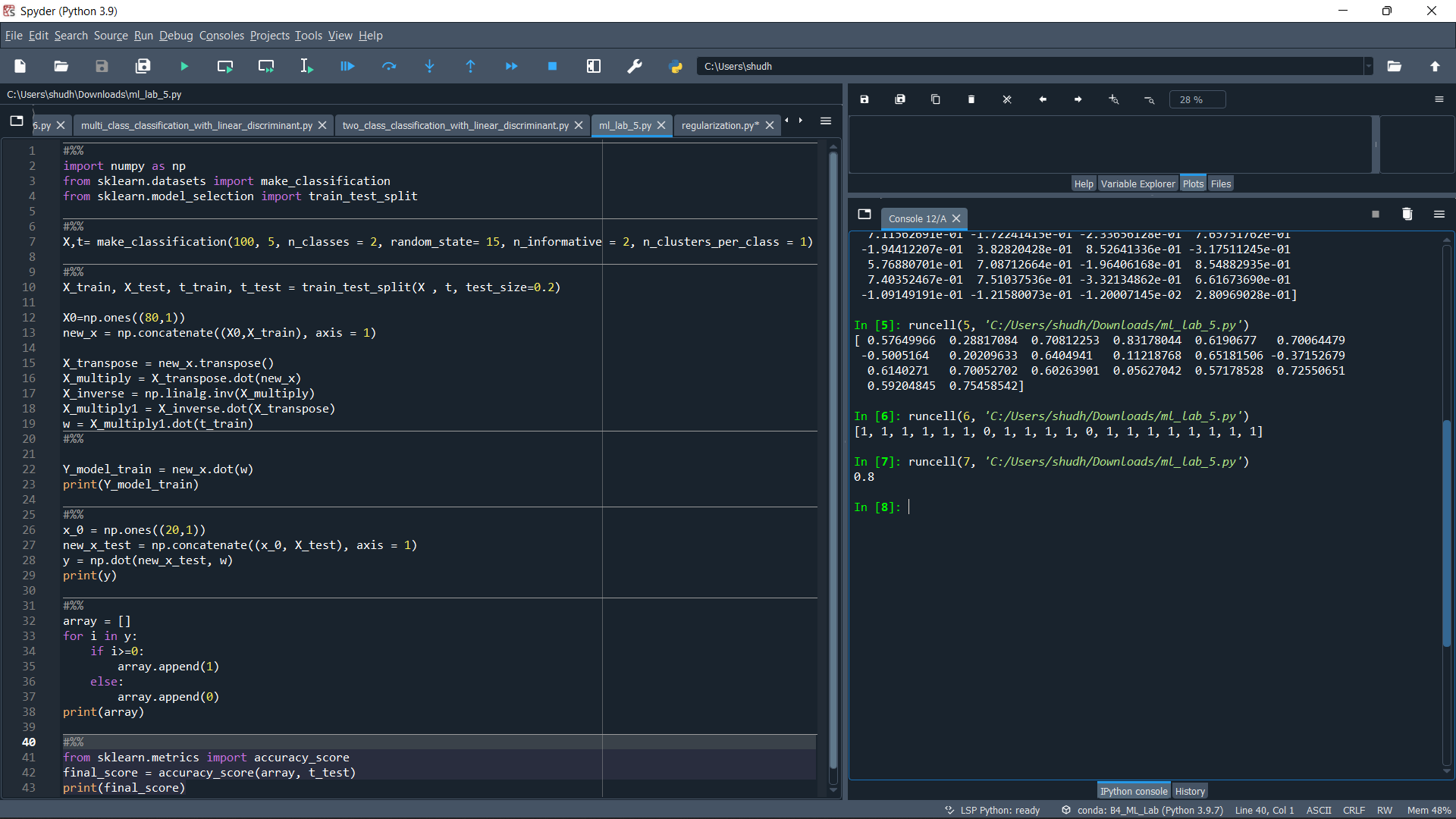
#%%

from sklearn.metrics import accuracy\_score

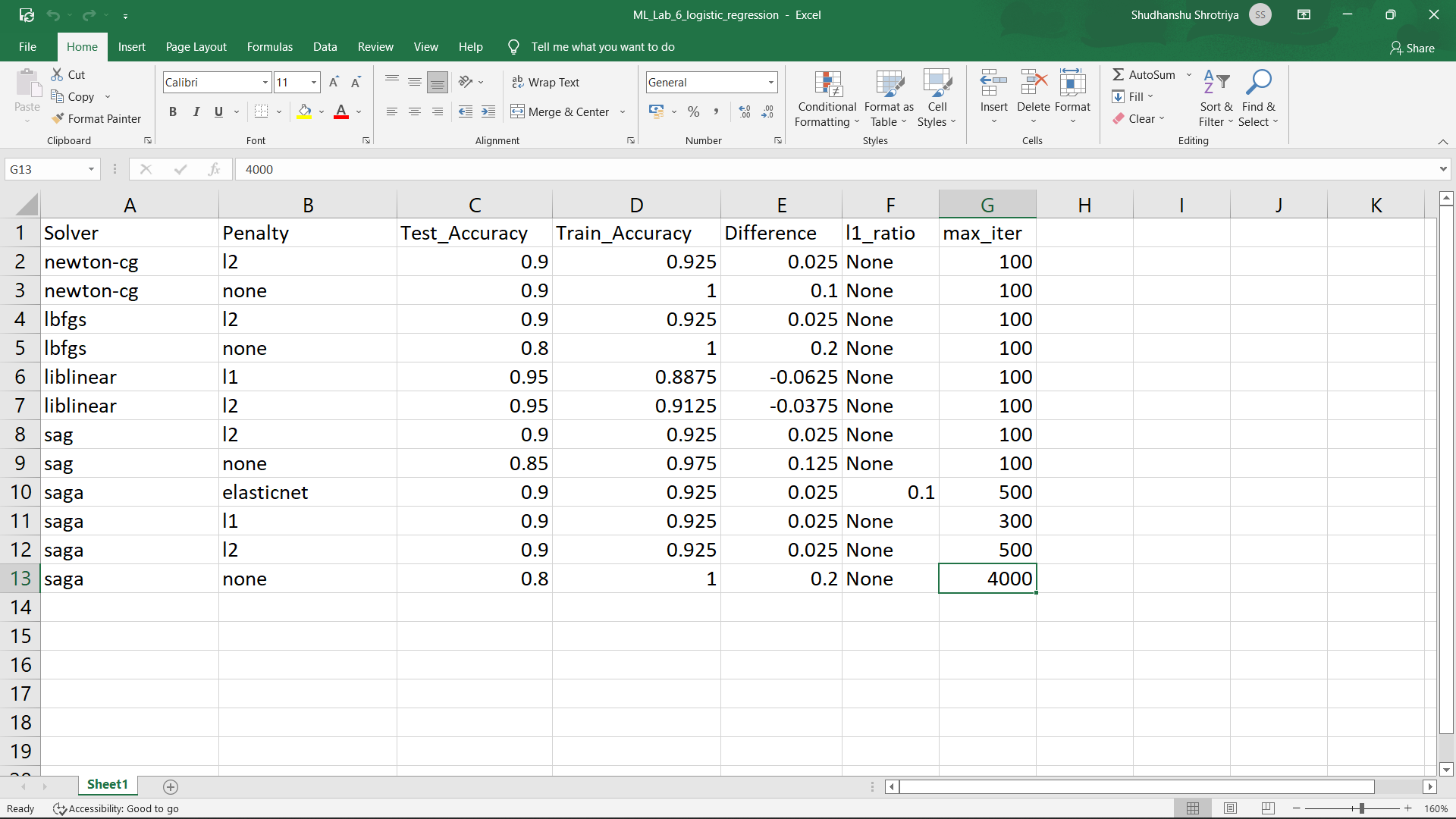
final\_score = accuracy\_score(array, t\_test)

print(final\_score)

**Output:** 



**Table:**



**Experiment-6**

**Objective:** To perform the classification in machine learning for 2 classes mathematically.

**Code:**

#%%

import numpy as np

from sklearn.datasets import make\_classification

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

#%%

X,t= make\_classification(100, 5, n\_classes = 2, random\_state= 15, n\_informative = 2, n\_clusters\_per\_class = 1)

#%%

X\_train, X\_test, t\_train, t\_test = train\_test\_split(X , t, test\_size=0.2)

X0=np.ones((80,1))

new\_x = np.concatenate((X0,X\_train), axis = 1)

X\_transpose = new\_x.transpose()

X\_multiply = X\_transpose.dot(new\_x)

X\_inverse = np.linalg.inv(X\_multiply)

X\_multiply1 = X\_inverse.dot(X\_transpose)

w = X\_multiply1.dot(t\_train)

#%%

Y\_model\_train = new\_x.dot(w)

print(Y\_model\_train)

#%%

x\_0 = np.ones((20,1))

new\_x\_test = np.concatenate((x\_0, X\_test), axis = 1)

y = np.dot(new\_x\_test, w)

print(y)

#%%

array = []

for i in y:

if i>=0:

array.append(1)

else:

array.append(0)

print(array)

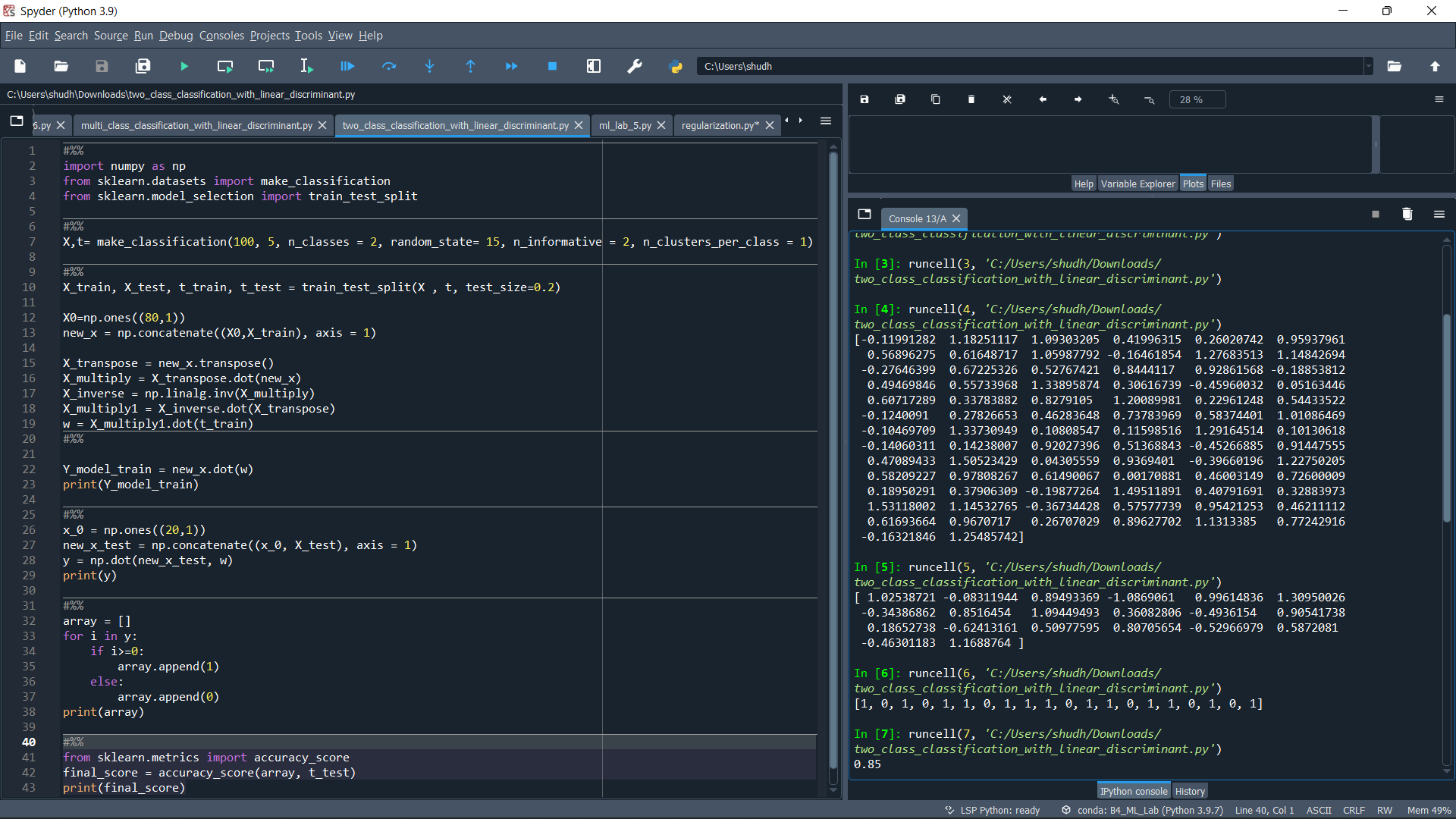
#%%

from sklearn.metrics import accuracy\_score

final\_score = accuracy\_score(array, t\_test)

print(final\_score)

**Output:**



**Experiment-7**

**Objective:** To perform the classification in machine learning for multi classes mathematically.

**Code:**

#%%

import numpy as np

from sklearn.datasets import make\_classification

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

#%%

classes = 3

X, t = make\_classification(100, 5, n\_classes = classes, random\_state= 40, n\_informative = 2, n\_clusters\_per\_class = 1)

res = np.zeros((t.shape[0], classes), dtype=int)

res[np.arange(t.shape[0]), t] = 1

print(res)

#%%

X\_train, X\_test, t\_train, t\_test = train\_test\_split(X , t, test\_size=0.2)

#%%

X0 = np.ones((100,1))

new\_x = np.concatenate((X0, X), axis = 1)

X\_transpose = new\_x.transpose()

X\_multiply = X\_transpose.dot(new\_x)

X\_inverse = np.linalg.inv(X\_multiply)

X\_multiply1 = X\_inverse.dot(X\_transpose)

#%%

w = X\_multiply1.dot(res)

print(w)

#%%

x\_0 = np.ones((20,1))

new\_x\_test = np.concatenate((x\_0, X\_test), axis = 1)

y = np.dot(new\_x\_test, w)

print(y)

#%%

z = np.argmax(y, axis = 1)

print(z)

#%%

res\_z = np.zeros((z.shape[0], classes), dtype=int)

res\_z[np.arange(z.shape[0]), z] = 1

print(res\_z)

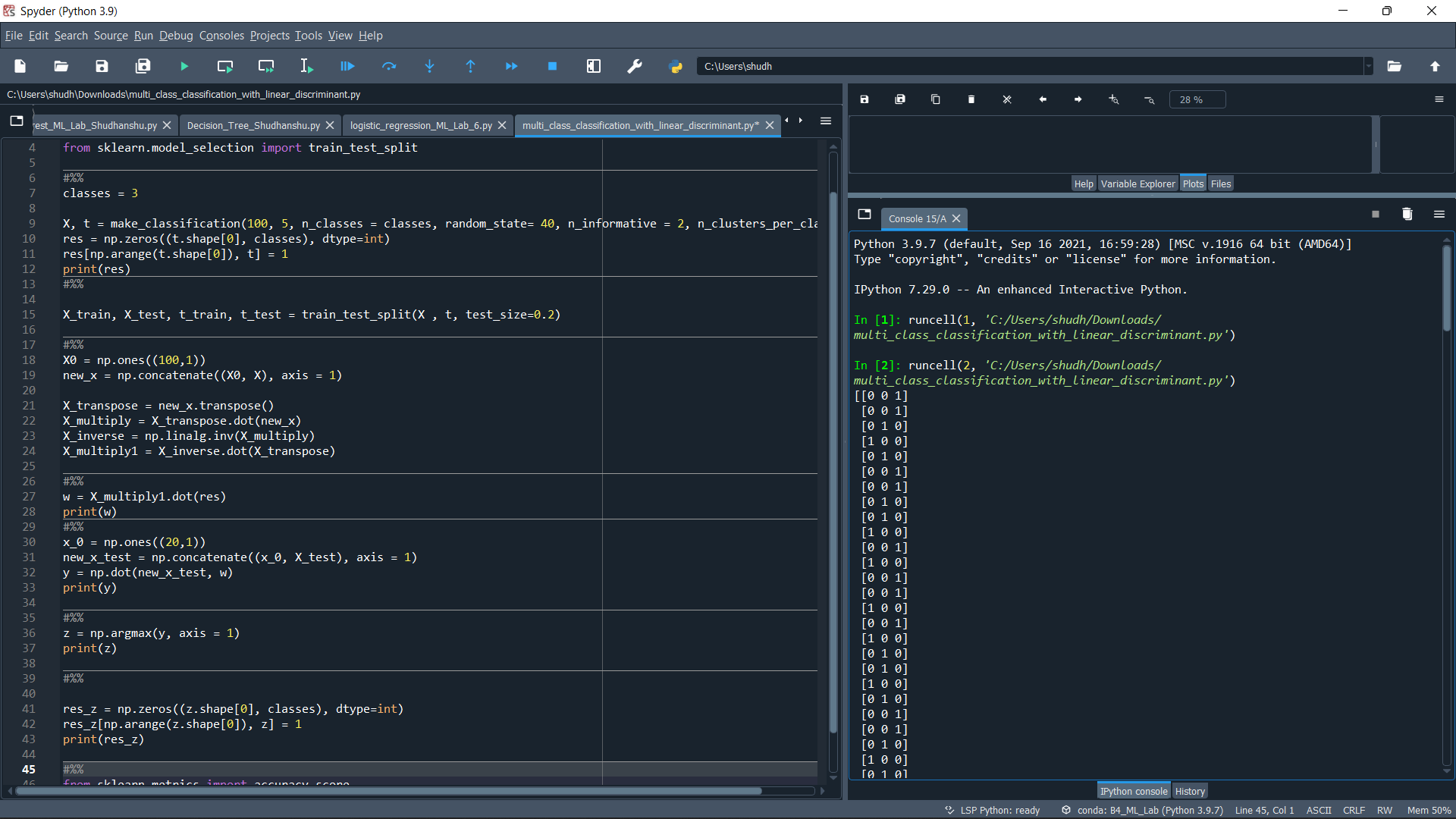
#%%

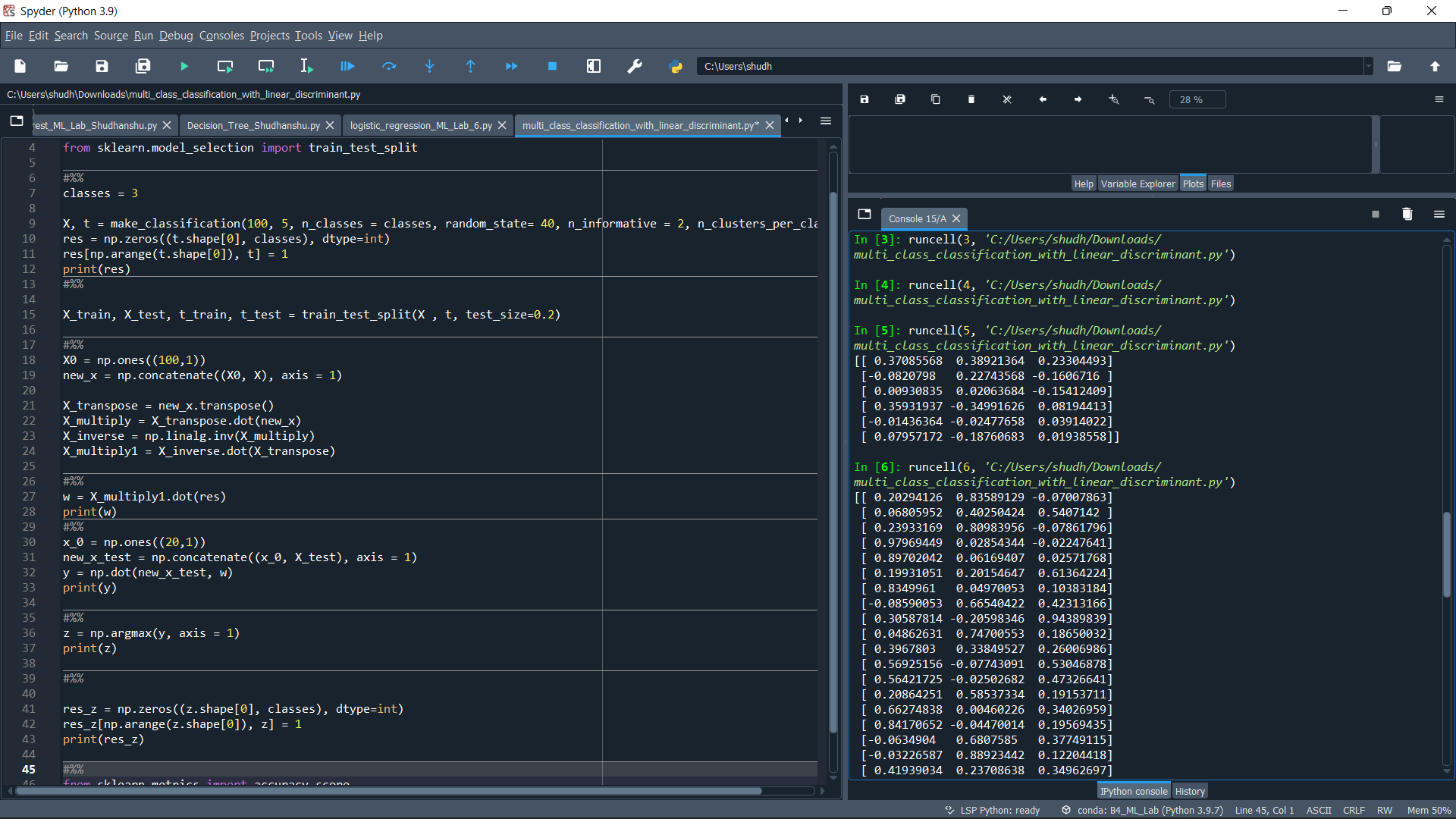
from sklearn.metrics import accuracy\_score

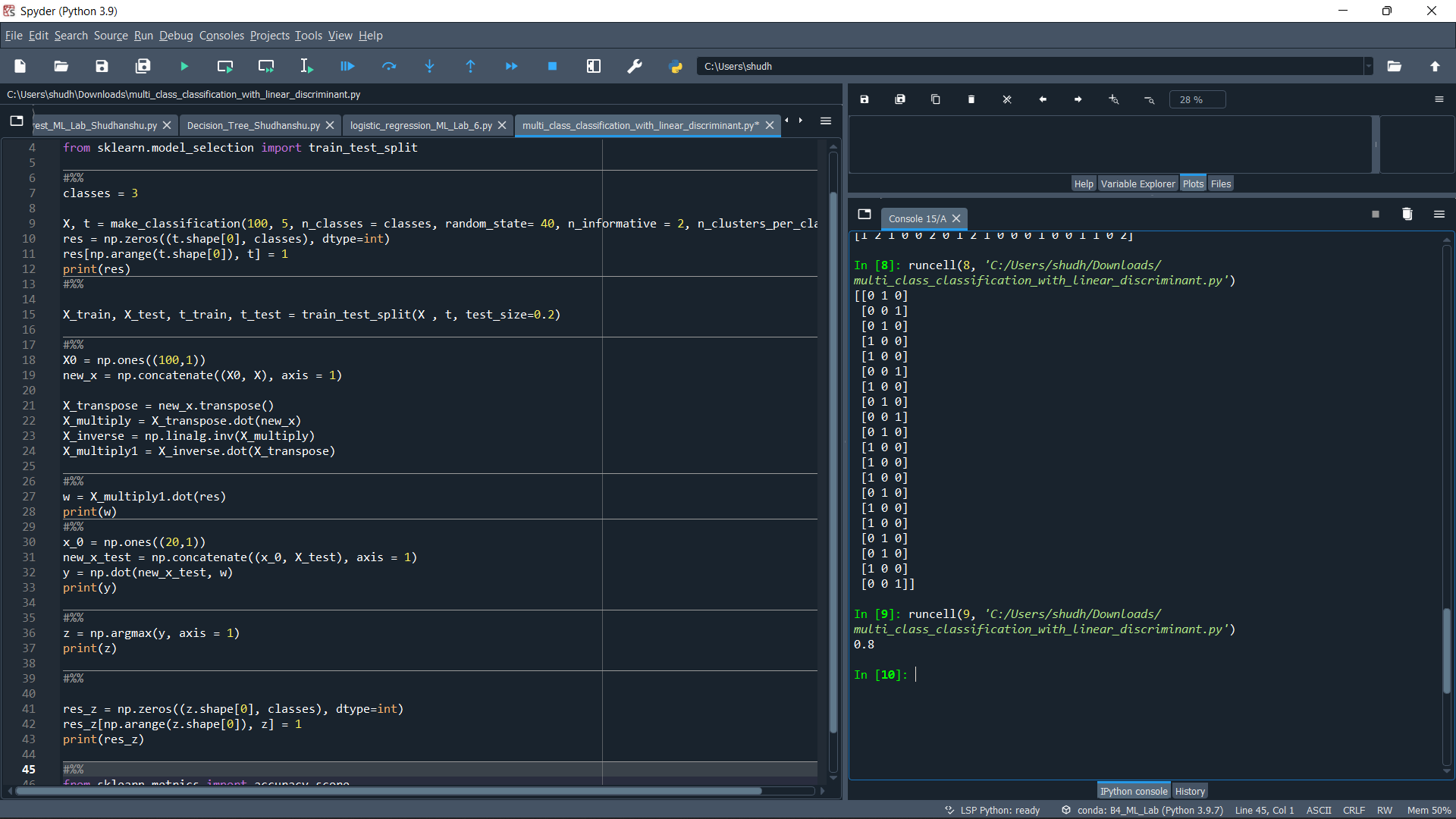
final\_score = accuracy\_score(z, t\_test)

print(final\_score)

**Output:**







**Experiment-8**

**Objective:** To implement the Decision Tree model

**Code of Decision Tree:**

#%%

from sklearn.datasets import make\_classification

from sklearn import tree

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

#%%

X, t = make\_classification(100, 5, n\_classes = 2, shuffle = True, random\_state= 10)

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

#%%

model = tree.DecisionTreeClassifier()

model = model.fit(X\_train, t\_train)

#%%

predicted\_value = model.predict(X\_test)

print(predicted\_value)

#%%

tree.plot\_tree(model)

#%%

zeroes = 0

ones = 0

for i in range(0,len(t\_train)):

if t\_train[i] == 0:

zeroes +=1

else:

ones +=1

#%%

print(zeroes)

print(ones)

#%%

val = 1 - ((zeroes/70)\*2 + (ones/70)\*2)

print("Gini :",val)

match = 0

UnMatch = 0

for i in range(30):

if predicted\_value[i] == t\_test[i]:

match += 1

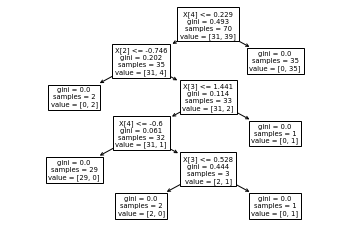
else:

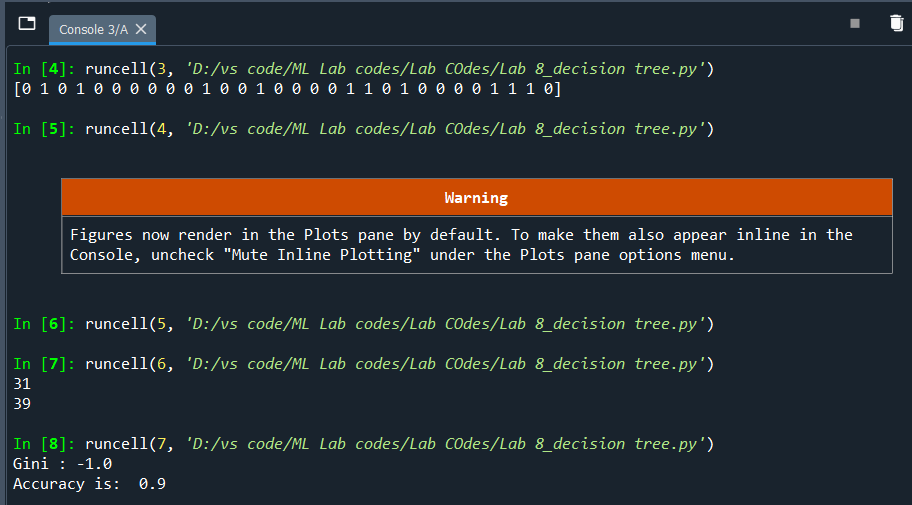
UnMatch += 1

accuracy = match/30

print("Accuracy is: ",accuracy)

**Output:**



****

**Experiment-9**

**Objective:** To implement the Random Forest Model, Adaboost Model for Classification and Adaboost Model for Regression.

**Code of Random Forest:**

import pandas

from sklearn.preprocessing import LabelEncoder

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

from sklearn.ensemble import RandomForestClassifier

#%%

data = {'CGPA':['g9','l9','g9','l9','g9'],

'Inter':['Y','N','N','N','Y'],

'PK':['++','==','==','==','=='],

'CS':['G','G','A','A','G'],

'Job':['Y','Y','N','N','Y']}

#%%

table=pandas.DataFrame(data,columns=["CGPA","Inter","PK","CS","Job"])

table.where(table["CGPA"]=="g9").count()

#%%

encoder=LabelEncoder()

for i in table:

table[i]=encoder.fit\_transform(table[i])

#%%

#Use iloc property to select by position.

X=table.iloc[:,0:4].values

t=table.iloc[:,4].values

#%%

X\_train,X\_test,t\_train,t\_test=train\_test\_split(X,t,test\_size=0.2,random\_state=2)

#%%

model = RandomForestClassifier(n\_estimators=3)

model.fit(X\_train,t\_train)

#%%

if model.predict([[0,1,1,1]])==1:

print("Got JOB")

else:

print("Didn't get JOB")

print(model.estimators\_)

#%%

import matplotlib.pyplot as plt

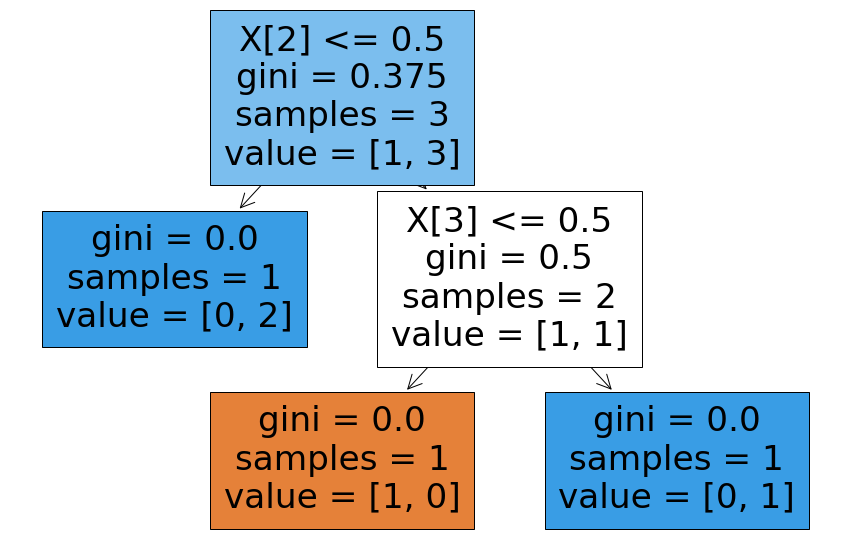
from sklearn import tree

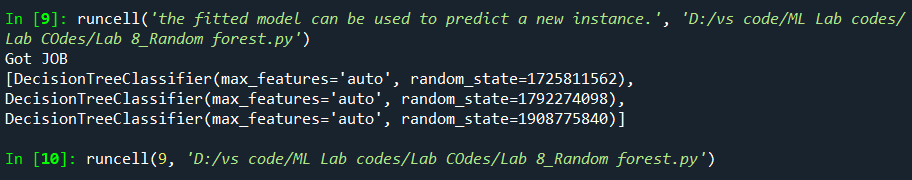
plt.figure(figsize=(15, 10))

tree.plot\_tree(model.estimators\_[2],

filled=True)

**Output:**



****

**Code of Adaboost Classification:**

import pandas

from sklearn.preprocessing import LabelEncoder

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

from sklearn.ensemble import AdaBoostClassifier

#%%

data = {'CGPA':['g9','l9','g9','l9','g9'],

'Inter':['Y','N','N','N','Y'],

'PK':['++','==','==','==','=='],

'CS':['G','G','A','A','G'],

'Job':['Y','Y','N','N','Y']}

#%%

table=pandas.DataFrame(data,columns=["CGPA","Inter","PK","CS","Job"])

table.where(table["CGPA"]=="g9").count()

#%%

encoder=LabelEncoder()

for i in table:

table[i]=encoder.fit\_transform(table[i])

#%%

X=table.iloc[:,0:4].values

t=table.iloc[:,4].values

#%%

X\_train,X\_test,t\_train,t\_test=train\_test\_split(X,t,test\_size=0.2,random\_state=2)

#%%

model = AdaBoostClassifier(n\_estimators=3)

model.fit(X\_train,t\_train)

#%%

if model.predict([[0,1,1,1]])==1:

print("Got JOB")

else:

print("Didn't get JOB")

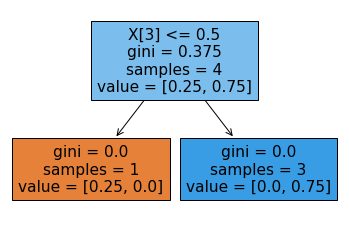
print(model.estimators\_)

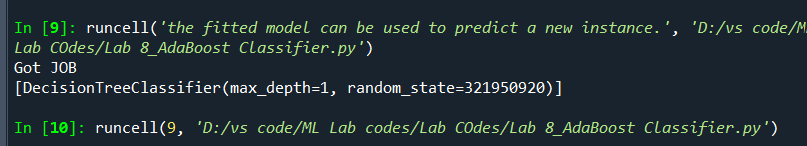
#%%

from sklearn import tree

tree.plot\_tree(model.estimators\_[0], filled=True)

**Output:**



****

**Code of Adaboost Regression:**

#%%

from sklearn.datasets import make\_regression

from sklearn.ensemble import AdaBoostRegressor

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

from sklearn.metrics import r2\_score

#%%

X\_reg,t\_reg = make\_regression(100, 3, n\_informative=15, noise=0.1, random\_state=6)

X\_train,X\_test,t\_train,t\_test=train\_test\_split(X\_reg,t\_reg,test\_size=0.3,random\_state=2)

#%%

model\_reg = AdaBoostRegressor()

model\_reg.fit(X\_train, t\_train)

#%%

y\_pred\_train = model\_reg.predict(X\_train)

score\_train = r2\_score(y\_pred\_train, t\_train)

print(score\_train)

#%%

y\_pred\_test = model\_reg.predict(X\_test)

score\_test = r2\_score(y\_pred\_test, t\_test)

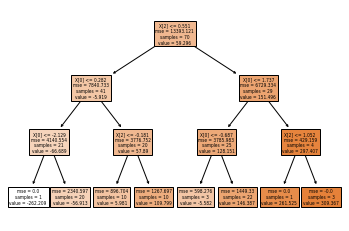
print(score\_test)

#%%

from sklearn import tree

tree.plot\_tree(model\_reg.estimators\_[0], filled=True)

**Output:**



**Experiment-10**

**Objective:** To implement the Neural Network in machine learning mathematically.

**Code:**

import numpy as np

def abs(x):

return x if x>0 else -x

def sigmoid (x):

return 1/(1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

def tanh\_act(x):

return np.tanh(x)

def tanh\_derivative(x):

return 1 - x\*\*2

def checkError(predicted\_output):

expected\_output = [[0],[1],[1],[0]]

for i,j in zip(expected\_output , predicted\_output):

if abs(i[0]-j[0]) > 0.001:

return True

return False

#Input datasets

inputs = np.array([[0,0],[0,1],[1,0],[1,1]])

expected\_output = np.array([[0],[1],[1],[0]])

epoch = 0

lr = 0.1

# inputLayerNeurons = 2

# hiddenLayerNeurons = 2

# outputLayerNeurons = 1

inputLayerNeurons = int(input("enter no of inputLayer"))

hiddenLayerNeurons = int(input("enter no of hiddenlayer "))

outputLayerNeurons = int(input("enter no of outputlayer "))

hidden\_weights = []

for i in range(1,inputLayerNeurons+1):

hidden\_weights\_ind = []

for j in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+1):

hidden\_weights\_ind.append(float(input('w'+str(i)+str(j))))

hidden\_weights.append(hidden\_weights\_ind)

output\_weights = []

for i in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+1):

output\_weights\_ind = []

for j in range(inputLayerNeurons+hiddenLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+outputLayerNeurons+1):

output\_weights\_ind.append(float(input('w'+str(i)+str(j))))

output\_weights.append(output\_weights\_ind)

hidden\_bias = []

output\_bias = []

for i in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+outputLayerNeurons+1):

if i > inputLayerNeurons+hiddenLayerNeurons:

output\_bias.append(float(input("o"+str(i))))

else:

hidden\_bias.append(float(input("o"+str(i))))

hidden\_weights = np.asarray(hidden\_weights)

hidden\_bias = np.asarray([hidden\_bias])

output\_weights = np.asarray(output\_weights)

output\_bias = np.asarray([output\_bias])

print("Initial hidden weights: ",end='')

print(\*hidden\_weights)

print("Initial hidden biases: ",end='')

print(\*hidden\_bias)

print("Initial output weights: ",end='')

print(\*output\_weights)

print("Initial output biases: ",end='')

print(\*output\_bias)

predicted\_output = [[0],[0],[0],[0]]

#Training algorithm

while checkError(predicted\_output):

epoch += 1

#Forward Propagation

hidden\_layer\_activation = np.dot(inputs,hidden\_weights)

hidden\_layer\_activation += hidden\_bias

hidden\_layer\_output = tanh\_act(hidden\_layer\_activation)

output\_layer\_activation = np.dot(hidden\_layer\_output,output\_weights)

output\_layer\_activation += output\_bias

predicted\_output = tanh\_act(output\_layer\_activation)

#Backpropagation

error = expected\_output - predicted\_output

d\_predicted\_output = error \* tanh\_derivative(predicted\_output)

error\_hidden\_layer = d\_predicted\_output.dot(output\_weights.T)

d\_hidden\_layer = error\_hidden\_layer \* tanh\_derivative(hidden\_layer\_output)

#Updating Weights and Biases

output\_weights += hidden\_layer\_output.T.dot(d\_predicted\_output) \* lr

output\_bias += np.sum(d\_predicted\_output,axis=0,keepdims=True) \* lr

hidden\_weights += inputs.T.dot(d\_hidden\_layer) \* lr

hidden\_bias += np.sum(d\_hidden\_layer,axis=0,keepdims=True) \* lr

print("Final hidden weights: ",end='')

print(\*hidden\_weights)

print("Final hidden bias: ",end='')

print(\*hidden\_bias)

print("Final output weights: ",end='')

print(\*output\_weights)

print("Final output bias: ",end='')

print(\*output\_bias)

print("\nOutput from neural network: ",end='')

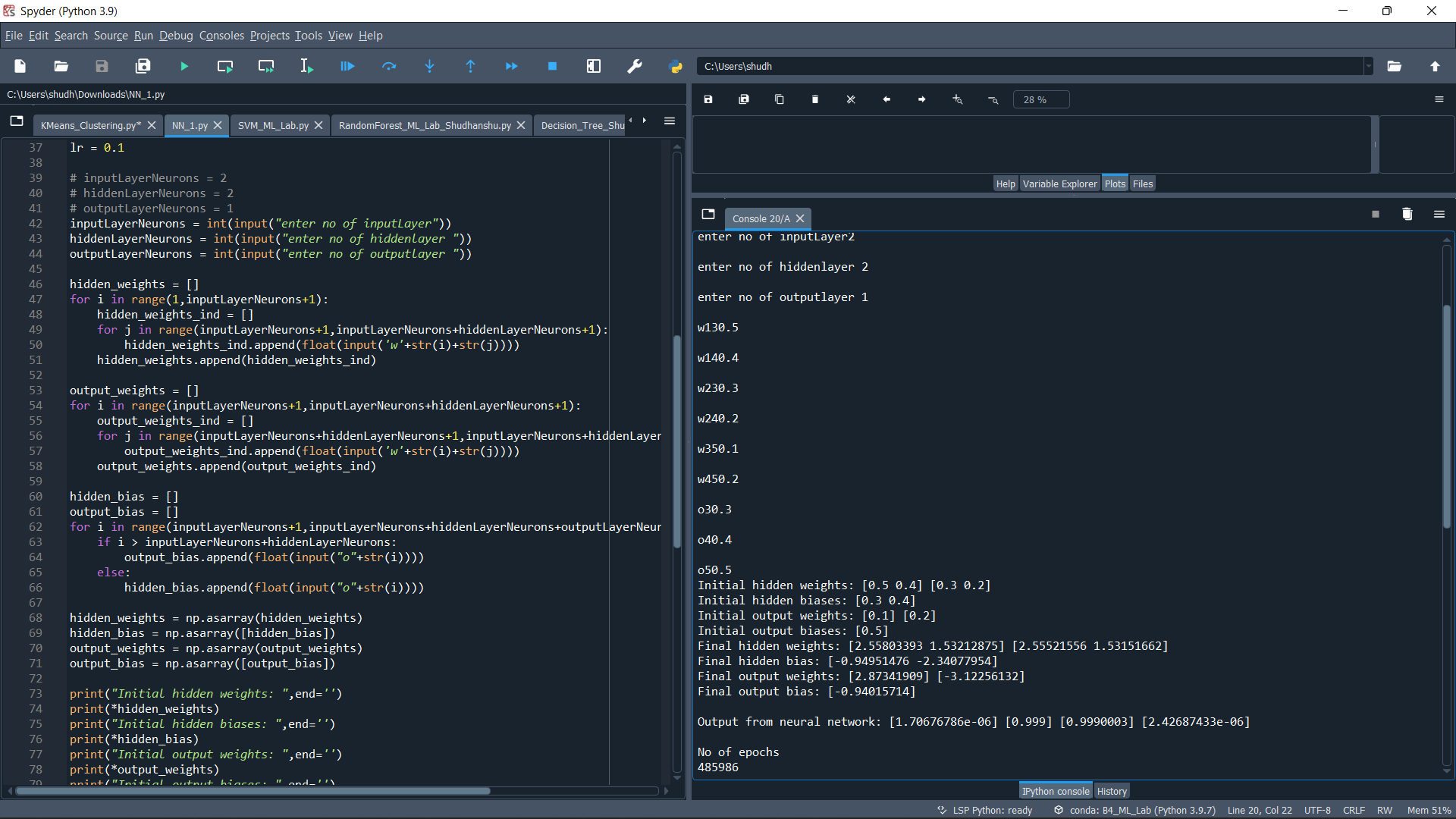
print(\*predicted\_output)

print("\nNo of epochs")

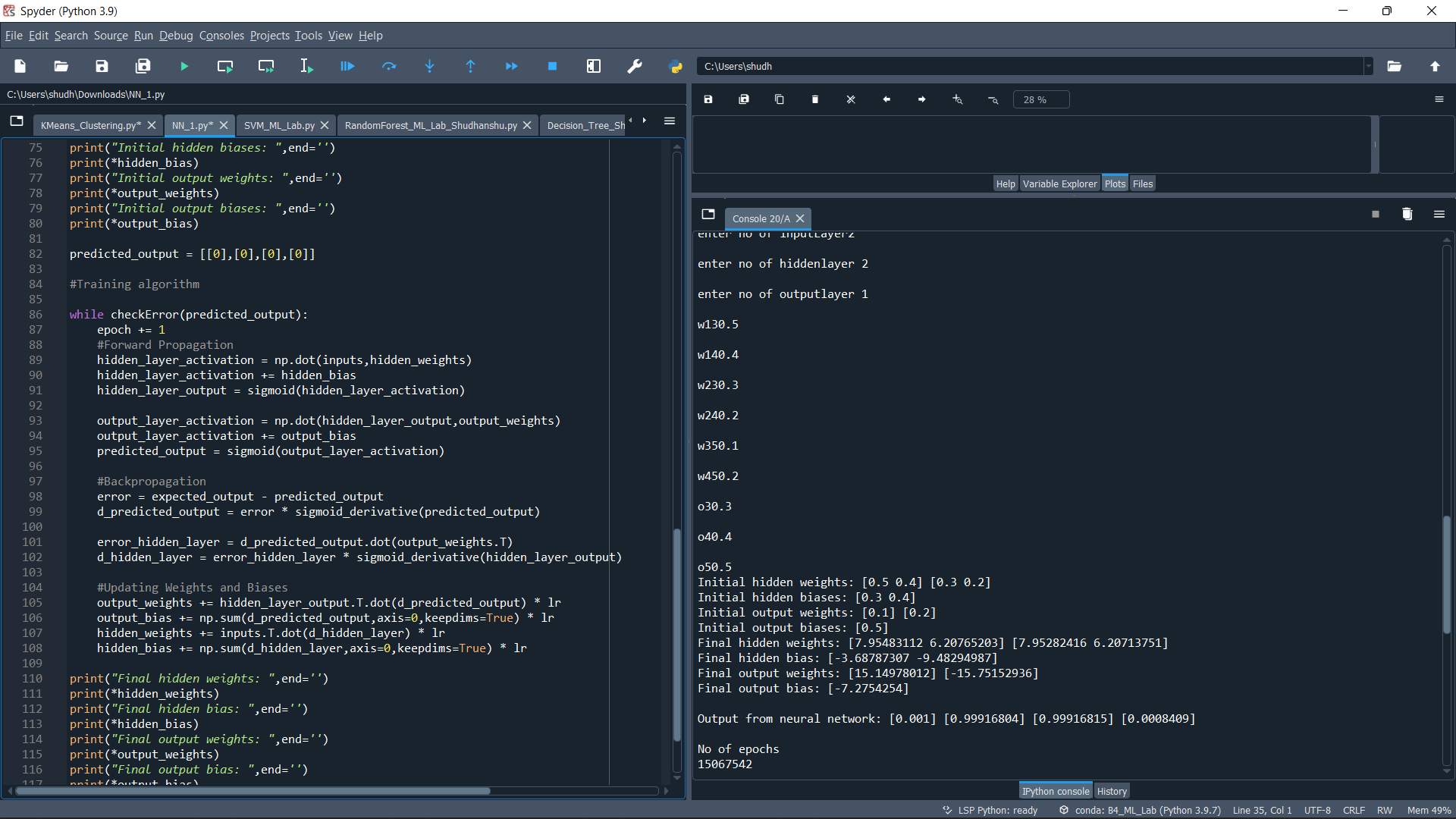
print(epoch)

**Output:**

**For Activation Function -> tanh**



**For Activation Function -> sigmoid**



**Experiment-11**

**Objective:** To implement Support Vector Machine (SVM) in machine learning.

**Code for Support Vector Machine:**

#%%

import numpy as np

from sklearn.datasets import make\_classification

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

import matplotlib.pyplot as plt

import seaborn as sns

#%%

X,t= make\_classification(500, 5, n\_classes = 3, random\_state= 50, n\_informative = 3, n\_clusters\_per\_class = 1)

#%%

X\_train, X\_test, t\_train, t\_test = train\_test\_split(X , t, test\_size=0.2, random\_state = 40)

#%%

plt.scatter(X\_train[:, 0], X\_train[:, 1], c = t\_train, marker='.')

plt.show()

#%%

from sklearn import svm

model\_svm = svm.SVC(kernel = "linear")

model\_svm.fit(X\_train, t\_train)

y\_pred = model\_svm.predict(X\_test)

#%%

from sklearn.metrics import accuracy\_score

final\_score = accuracy\_score(t\_test, y\_pred)

print(final\_score)

#%%

import random

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def generate\_random\_dataset(size):

""" Generate a random dataset and that follows a quadratic distribution

"""

x = []

y = []

target = []

for i in range(size):

# class zero

x.append(np.round(random.uniform(0, 2.5), 1))

y.append(np.round(random.uniform(0, 20), 1))

target.append(0) # class one

x.append(np.round(random.uniform(1, 5), 2))

y.append(np.round(random.uniform(20, 25), 2))

target.append(1)

x.append(np.round(random.uniform(3, 5), 2))

y.append(np.round(random.uniform(5, 25), 2))

target.append(1)

df\_x = pd.DataFrame(data=x)

df\_y = pd.DataFrame(data=y)

df\_target = pd.DataFrame(data=target)

data\_frame = pd.concat([df\_x, df\_y], ignore\_index=True, axis=1)

data\_frame = pd.concat([data\_frame, df\_target], ignore\_index=True, axis=1)

data\_frame.columns = ['x', 'y', 'target']

return data\_frame

# Generate dataset

size = 100

dataset = generate\_random\_dataset(size)

features = dataset[['x', 'y']]

label = dataset['target']# Hold out 20% of the dataset for training

test\_size = int(np.round(size \* 0.2, 0))# Split dataset into training and testing sets

x\_train = features[:-test\_size].values

y\_train = label[:-test\_size].values

x\_test = features[-test\_size:].values

y\_test = label[-test\_size:].values

#%%

fig, ax = plt.subplots(figsize=(12, 7))# removing to and right border

ax.spines['top'].set\_visible(False)

ax.spines['left'].set\_visible(False)

ax.spines['right'].set\_visible(False)# adding major gridlines

ax.grid(color='grey', linestyle='-', linewidth=0.25, alpha=0.5)

ax.scatter(features[:-test\_size]['x'], features[:-test\_size]['y'], color="#8C7298")

plt.show()

#%%

from sklearn import svm

model = svm.SVC(kernel='linear')

model.fit(x\_train, y\_train)

#%%

fig, ax = plt.subplots(figsize=(12, 7))# Removing to and right border

ax.spines['top'].set\_visible(False)

ax.spines['left'].set\_visible(False)

ax.spines['right'].set\_visible(False)# Create grid to evaluate model

xx = np.linspace(-1, max(features['x']) + 1, len(x\_train))

yy = np.linspace(0, max(features['y']) + 1, len(y\_train))

YY, XX = np.meshgrid(yy, xx)

xy = np.vstack([XX.ravel(), YY.ravel()]).T

train\_size = len(features[:-test\_size]['x'])# Assigning different colors to the classes

colors = y\_train

colors = np.where(colors == 1, '#8C7298', '#4786D1')# Plot the dataset

ax.scatter(features[:-test\_size]['x'], features[:-test\_size]['y'], c=colors)# Get the separating hyperplane

Z = model.decision\_function(xy).reshape(XX.shape)

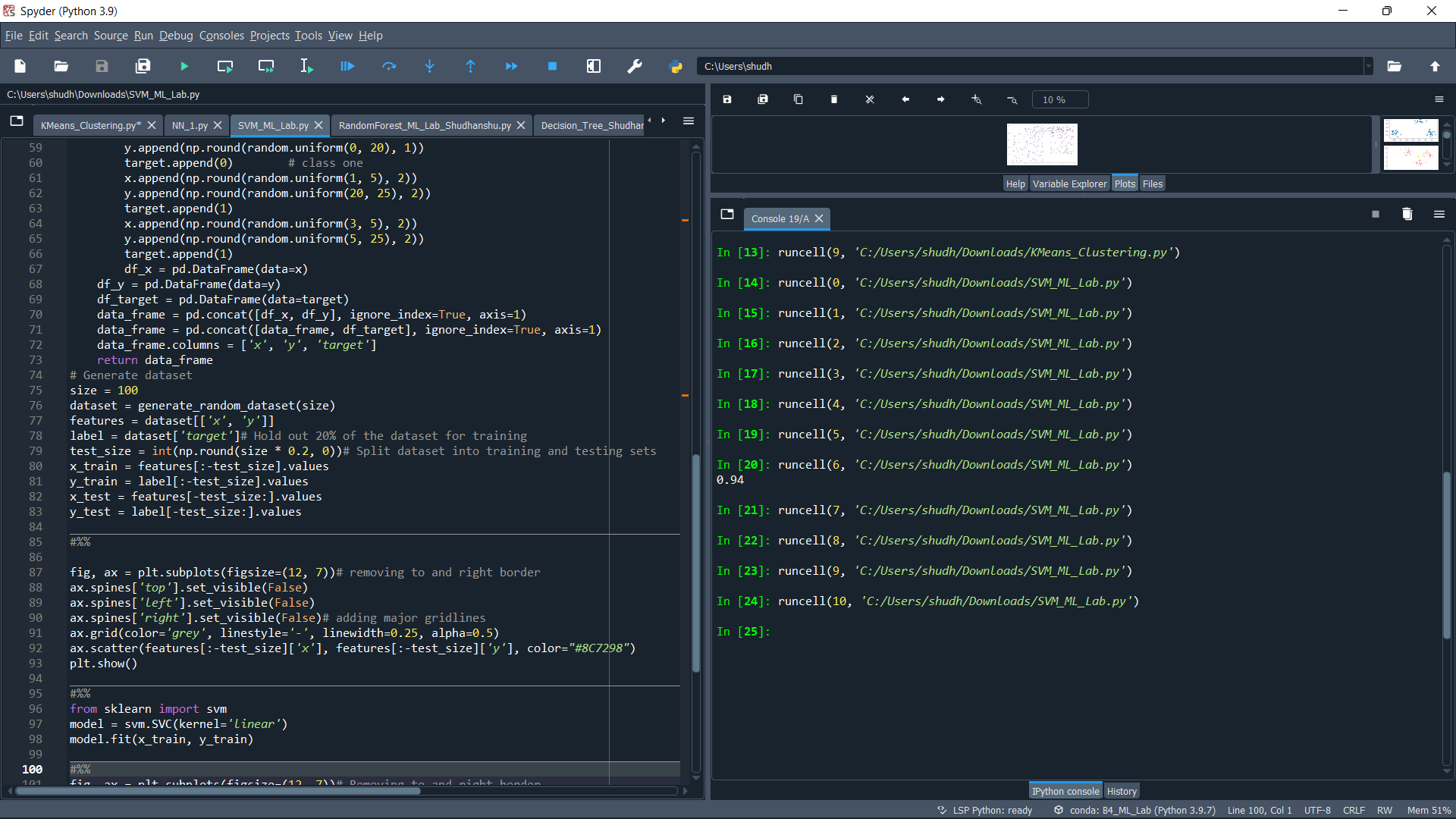
# Draw the decision boundary and margins

ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])# Highlight support vectors with a circle around them

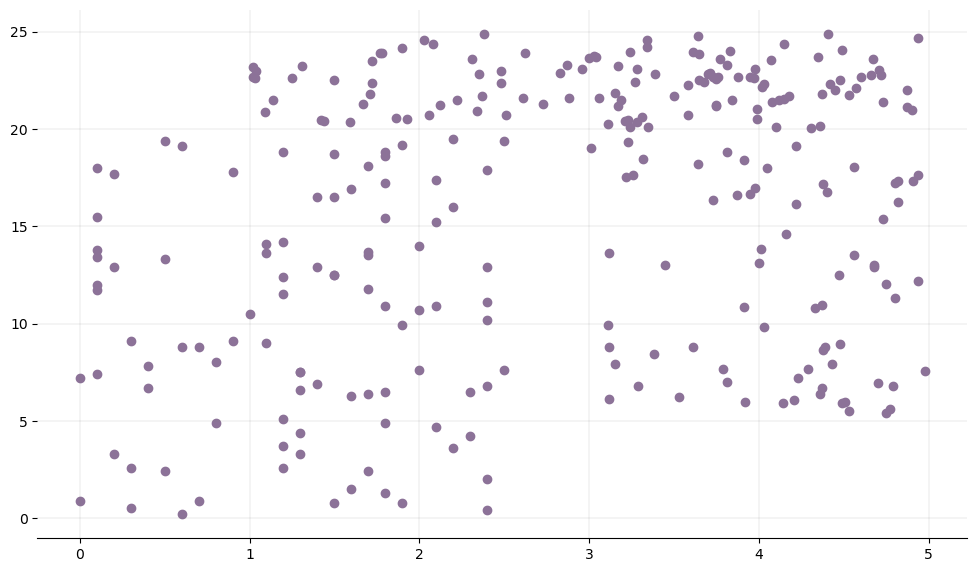
ax.scatter(model.support\_vectors\_[:, 0], model.support\_vectors\_[:, 1], s=100, linewidth=1, facecolors='none', edgecolors='k')

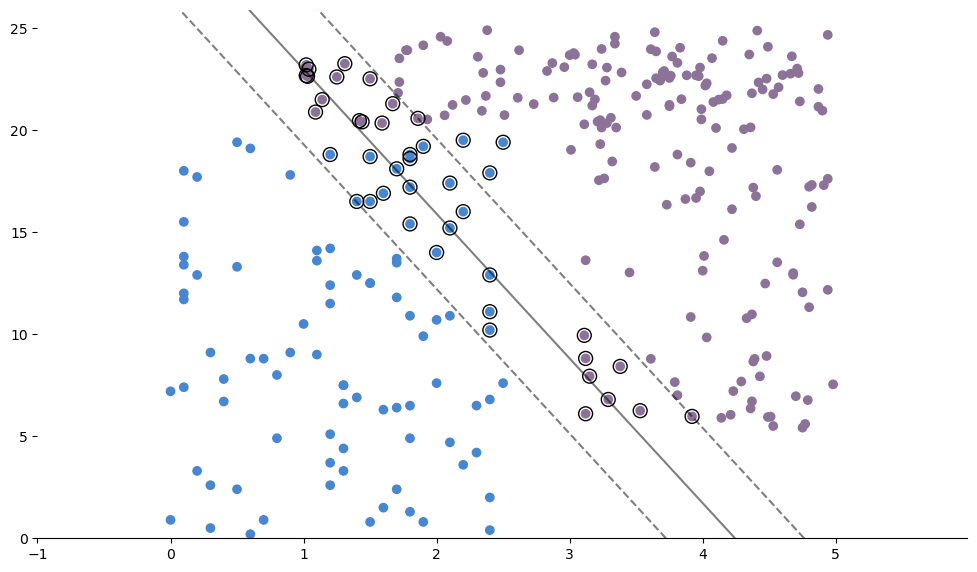
plt.show()

**Output:**



**Plots:**





**Experiment-12**

**Objective:** To implement the K-Means Clustering Algorithm in Machine Learning

**Code for KMeans:**

#%%

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

#%%

from sklearn.datasets import make\_blobs

X, y = make\_blobs(n\_samples=100, centers=5, random\_state=101)

#%%

# setting the number of training examples

m=X.shape[0]

n=X.shape[1]

n\_iter=50

#%%

# computing the initial centroids randomly

K=5

import random

# creating an empty centroid array

centroids=np.array([]).reshape(n,0)

# creating 5 random centroids

for k in range(K):

centroids=np.c\_[centroids,X[random.randint(0,m-1)]]

#%%

output={}

# creating an empty array

euclid=np.array([]).reshape(m,0)

# finding distance between for each centroid

for k in range(K):

dist=np.sum((X-centroids[:,k])\*\*2,axis=1)

euclid=np.c\_[euclid,dist]

# storing the minimum value we have computed

minimum=np.argmin(euclid,axis=1)+1

#%%

# computing the mean of separated clusters

cent={}

for k in range(K):

cent[k+1]=np.array([]).reshape(2,0)

# assigning of clusters to points

for k in range(m):

cent[minimum[k]]=np.c\_[cent[minimum[k]],X[k]]

for k in range(K):

cent[k+1]=cent[k+1].T

# computing mean and updating it

for k in range(K):

centroids[:,k]=np.mean(cent[k+1],axis=0)

#%%

# repeating the above steps again and again

for i in range(n\_iter):

euclid=np.array([]).reshape(m,0)

for k in range(K):

dist=np.sum((X-centroids[:,k])\*\*2,axis=1)

euclid=np.c\_[euclid,dist]

C=np.argmin(euclid,axis=1)+1

cent={}

for k in range(K):

cent[k+1]=np.array([]).reshape(2,0)

for k in range(m):

cent[C[k]]=np.c\_[cent[C[k]],X[k]]

for k in range(K):

cent[k+1]=cent[k+1].T

for k in range(K):

centroids[:,k]=np.mean(cent[k+1],axis=0)

final=cent

#%%

plt.scatter(X[:,0],X[:,1])

plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':100})

plt.title('Original Dataset')

#%%

for k in range(K):

plt.scatter(final[k+1][:,0],final[k+1][:,1])

plt.scatter(centroids[0,:],centroids[1,:],s=300,c='yellow')

plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':100})

plt.show()

#%%

from sklearn.datasets.samples\_generator import make\_blobs

X, y = make\_blobs(n\_samples=100, centers=5, random\_state=101)

#%%

#ELBOW\_METHOD

import seaborn as sns

from sklearn.cluster import KMeans

elbow=[]

for i in range(1, 20):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 101)

kmeans.fit(X)

elbow.append(kmeans.inertia\_)

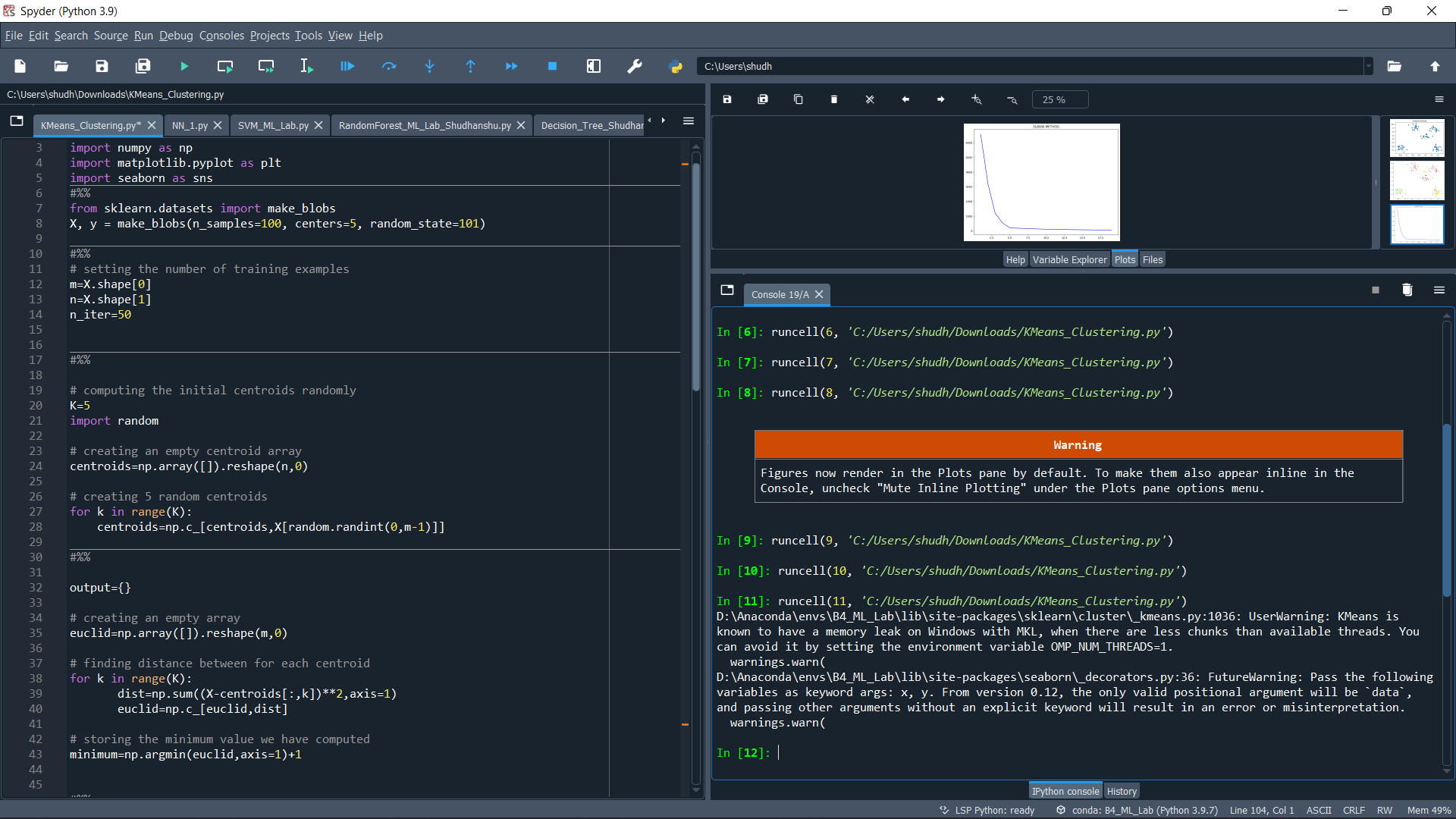
sns.lineplot(range(1, 20), elbow,color='blue')

plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':100})

plt.title('ELBOW METHOD')

plt.show()

**Output:**



**Plots:**

