SCHOOL OF COMPUTER SCIENCE

MACHINE LEARNING

BACHELOR OF COMPUTER SCIENCE ENGINEERING WITH SPECIALIZATION IN ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

BATCH: B3 (Hons.)

SEMESTER: IV

SUBMITTED TO: PROF. GOPAL S PHARTIYAL

SUBMITTED BY: HARSHIT RAHEJA

SAP ID: 500086226

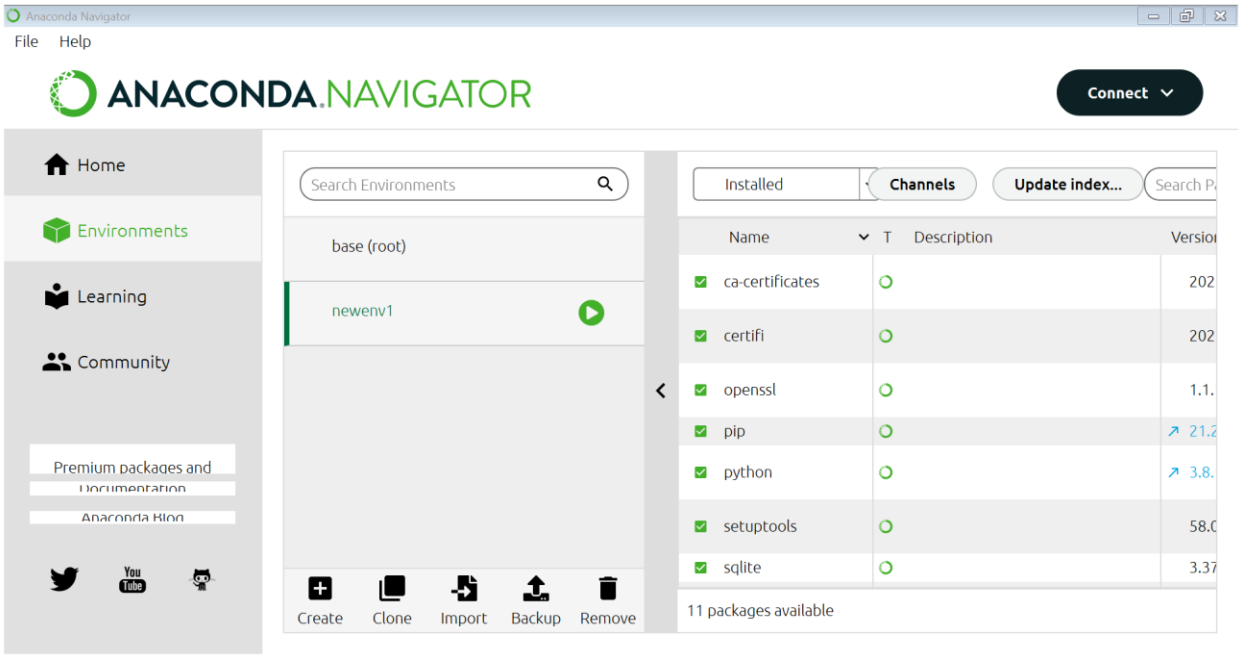
ROLL NO: R2142201556

Index

Experiment 1: Introduction to Scikit learn library

Installation Steps

1. Install Anaconda
2. Open the Anaconda Navigator
3. Go to environment and create a new environment.
4. Go to environments in the left panel and check the installed libraries.
5. Install numpy
6. Install scipy
7. Install sklearn
8. Installation is completed.



Experiment 2: Linear regression

Aim: Linear regression with input as vector

Code:

```
from sklearn.datasets import make_regression


import matplotlib.pyplot as plt
```

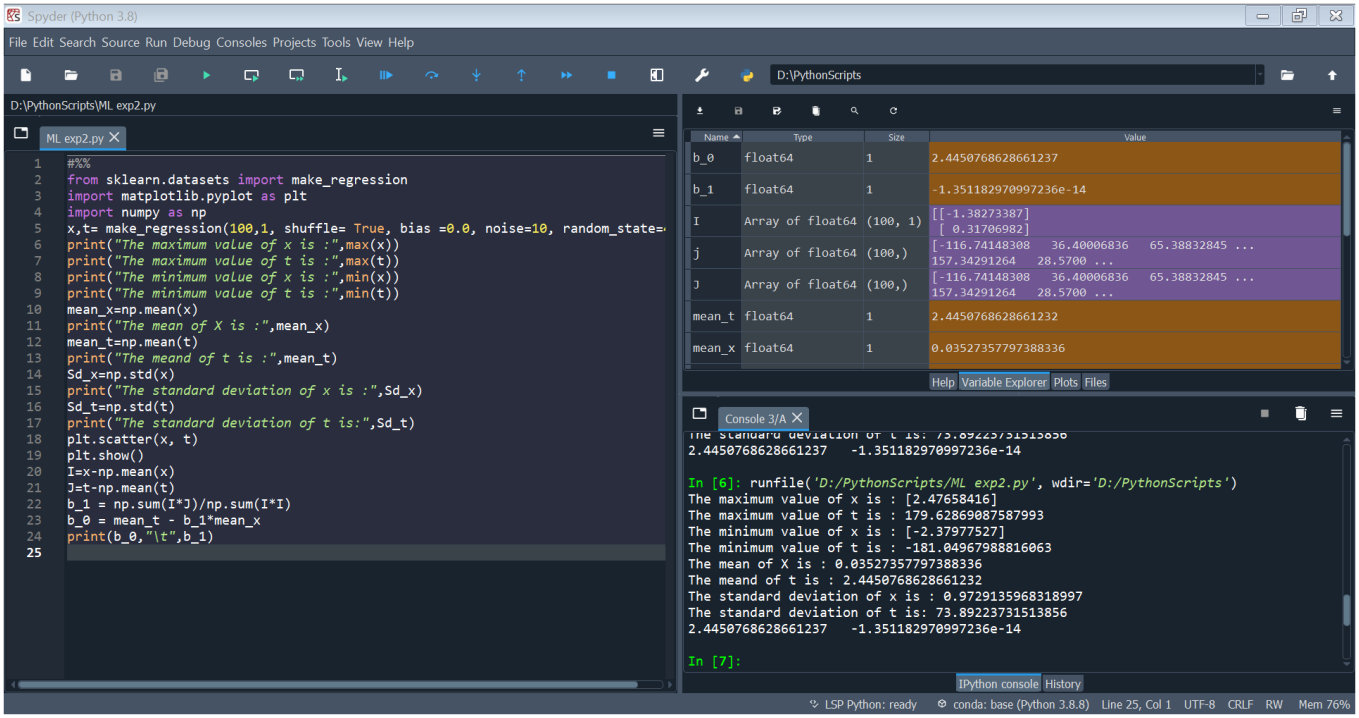
```
import numpy as np

x,t= make_regression(100,1, shuffle= True, bias =0.0, noise=10, random_state=4 )

print("The maximum value of x is :",max(x))

print("The maximum value of t is :",max(t))

print("The minimum value of x is :",min(x))

print("The minimum value of t is :",min(t))

mean_x=np.mean(x)

print("The mean of X is :",mean_x)

mean_t=np.mean(t)

print("The meand of t is :",mean_t)

Sd_x=np.std(x)

print("The standard deviation of x is :",Sd_x)

Sd_t=np.std(t)

print("The standard deviation of t is:",Sd_t)

plt.scatter(x, t)

plt.show()

I=x-np.mean(x)

J=t-np.mean(t)
```

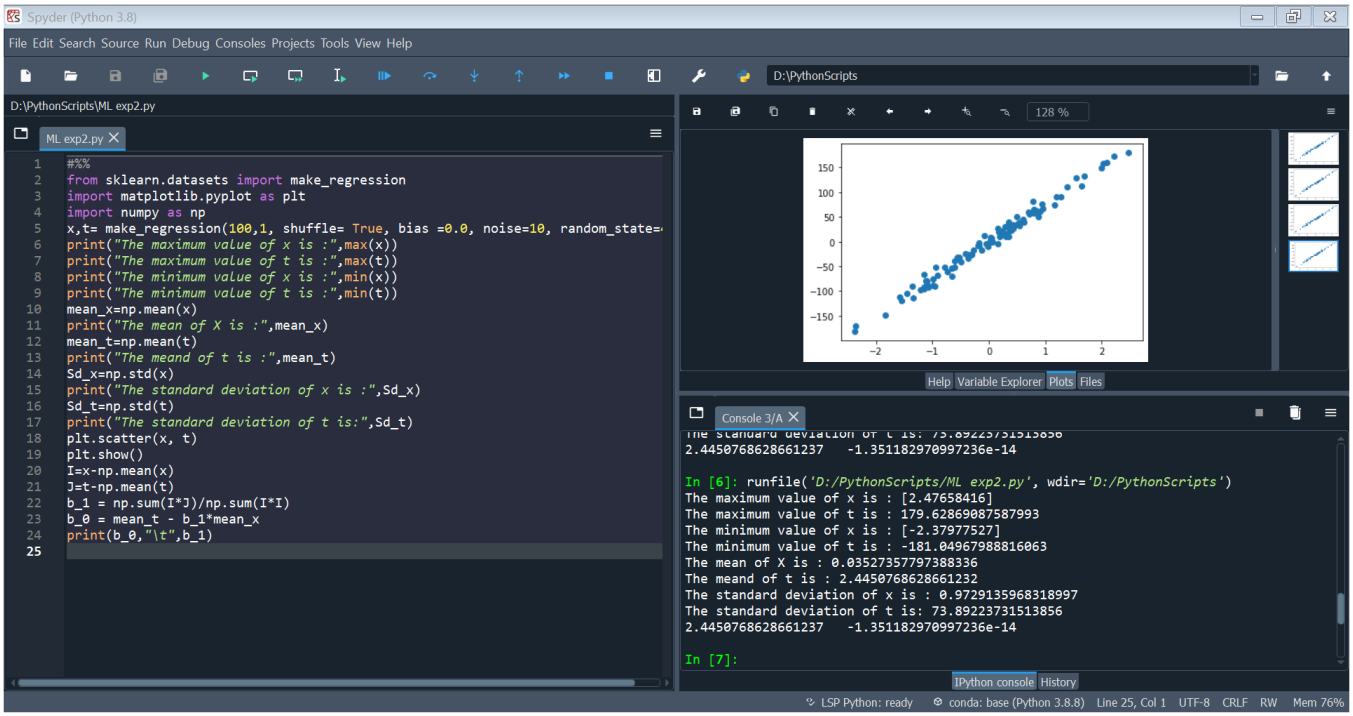
b_1 = np.sum(I*J)/np.sum(I*I)

b_0 = mean_t - b_1*mean_x

print(b_0,"\t",b_1)

Output:





Experiment 2.1: Linear regression with input as vector

Aim: Linear regression with input as vector

Code:

from sklearn.datasets import make_regression

import numpy as np

x,t= make_regression(n_samples=100,n_features=5,shuffle=True, bias =0.0, noise=10,

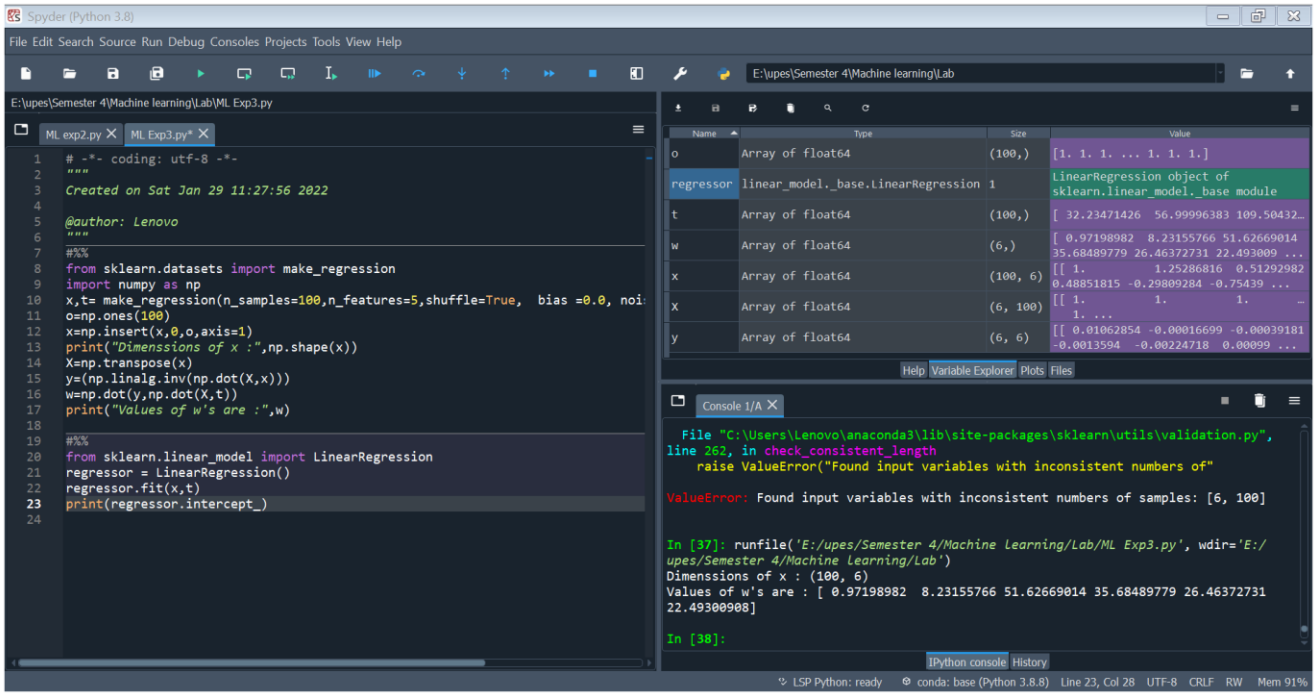
random_state=1)

o=np.ones(100)

x=np.insert(x,0,o,axis=1)

```
print("Dimenssions of x :",np.shape(x))

X=np.transpose(x)

y=(np.linalg.inv(np.dot(X,x)))

w=np.dot(y,np.dot(X,t))

print("Values of w's are :",w)

#%%

from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(x,t)

print(regressor.intercept_)
```

Output:



Experiment 3: Linear regression with x and y as vector

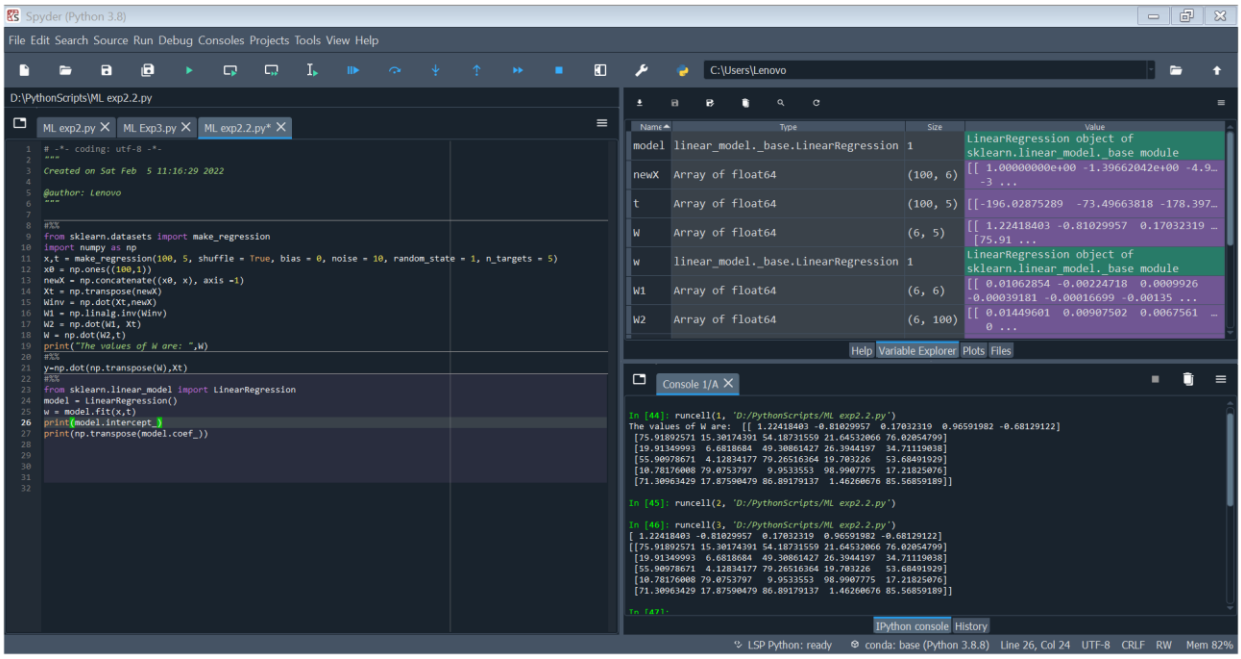
Aim: Linear regression with x and y as vector

Code:

```
#%%

from sklearn.datasets import make_regression

import numpy as np

x,t = make_regression(100, 5, shuffle = True, bias = 0, noise = 10, random_state = 1, n_targets =

5)

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

newX = np.concatenate((x0, x), axis =1)

Xt = np.transpose(newX)

Winv = np.dot(Xt,newX)

W1 = np.linalg.inv(Winv)

W2 = np.dot(W1, Xt)

W = np.dot(W2,t)

print("The values of W are: ",W)

#%%

y=np.dot(np.transpose(W),Xt)
```

```
#%%

from sklearn.linear_model import LinearRegression

model = LinearRegression()

w = model.fit(x,t)

print(model.intercept_)

print(np.transpose(model.coef_))
```

Output:



Experiment 4:

Part 1: Coefficient of Determination

Aim: Coefficient of Determination

Code:

```
from sklearn.datasets import make_regression

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

n=[0,10,20,40,80]

q=[]

for i in n:

    x,t = make_regression(100, 5, shuffle = True, bias = 0, noise = i, random_state=1)

    model = LinearRegression()

    w = model.fit(x,t)

    y=model.predict(x)

    score = r2_score(t,y)

    q.append(score)
```

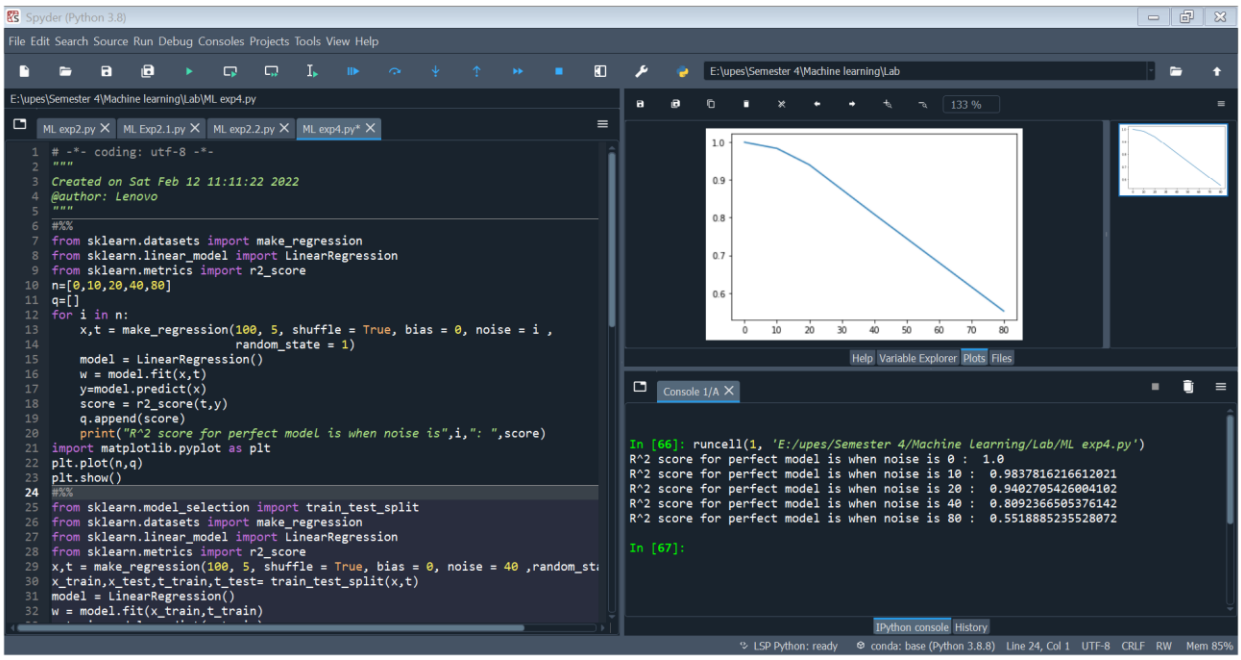
print("R^2 score for perfect model is when noise is",i,": ",score)

import matplotlib.pyplot as plt

plt.plot(n,q)

plt.show()

Output:



Part 2: Split Train and Test data

Part 2.1: Coefficient of determination of Train and Test data

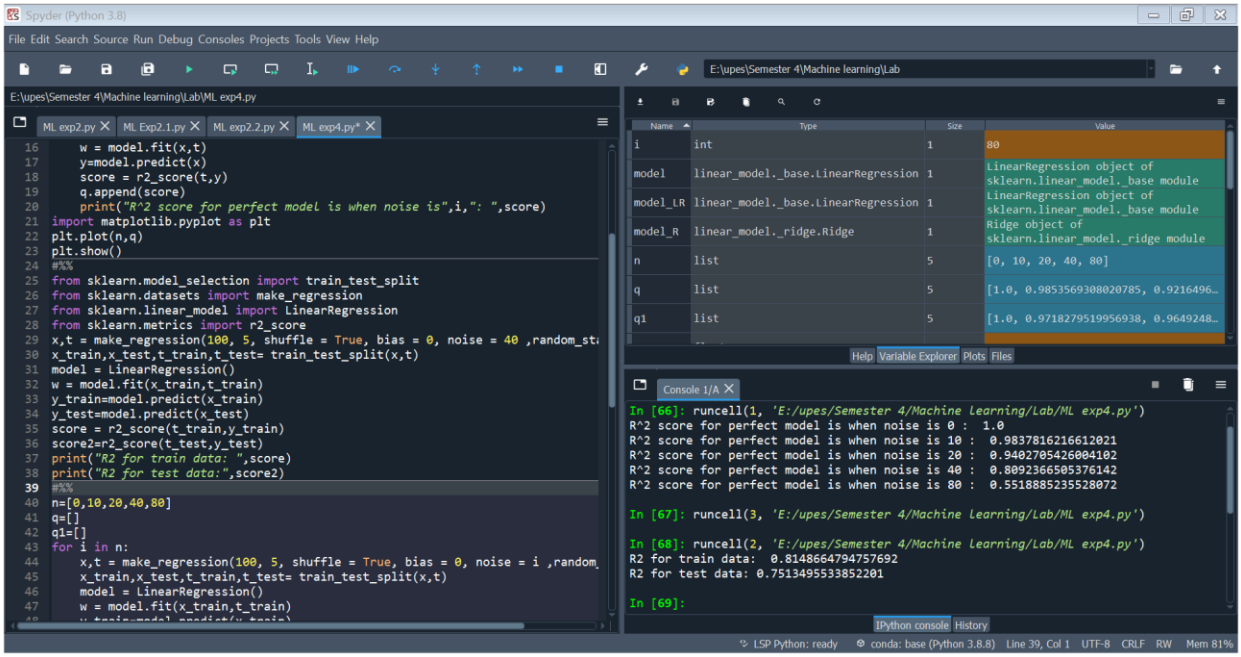
Aim: Coefficient of determination of Train and Test **data**

Code:

from sklearn.model_selection import train_test_split

```
from sklearn.datasets import make_regression

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

x,t = make_regression(100, 5, shuffle = True, bias = 0, noise = 40 ,random_state = 1)

x_train,x_test,t_train,t_test= train_test_split(x,t)

model = LinearRegression()

w = model.fit(x_train,t_train)

y_train=model.predict(x_train)

y_test=model.predict(x_test)

score = r2_score(t_train,y_train)

score2=r2_score(t_test,y_test)

print("R2 for train data: ",score)

print("R2 for test data:",score2)
```

Output:



Part 2.2: Coefficient of determination of Train and Test data with variable noise

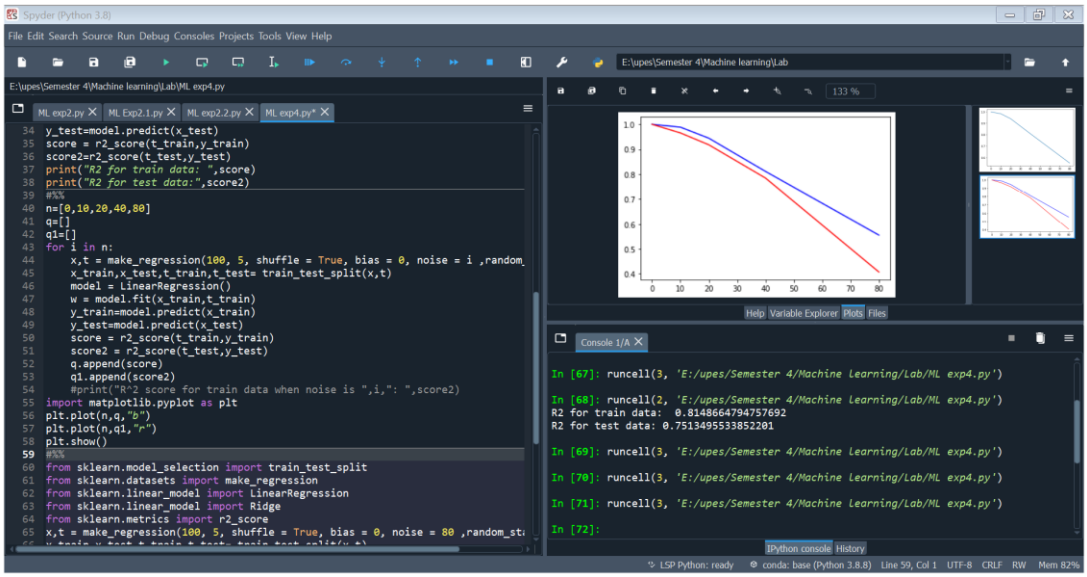
Aim: Coefficient of determination of Train and Test data with variable noise

Code:

```
n=[0,10,20,40,80]

q=[]

q1=[]

for i in n:

    x,t = make_regression(100, 5, shuffle = True, bias = 0, noise = i ,random_state = 1)

    x_train,x_test,t_train,t_test= train_test_split(x,t)

    model = LinearRegression()

    w = model.fit(x_train,t_train)
```

```
    y_train=model.predict(x_train)
    y_test=model.predict(x_test)
    score = r2_score(t_train,y_train)
    score2 = r2_score(t_test,y_test)
    q.append(score)
    q1.append(score2)
    #print("R^2 score for train data when noise is ",i,": ",score2)
import matplotlib.pyplot as plt
plt.plot(n,q,"b")
plt.plot(n,q1,"r")
plt.show()
```

Output:



Part 3: LR and R model

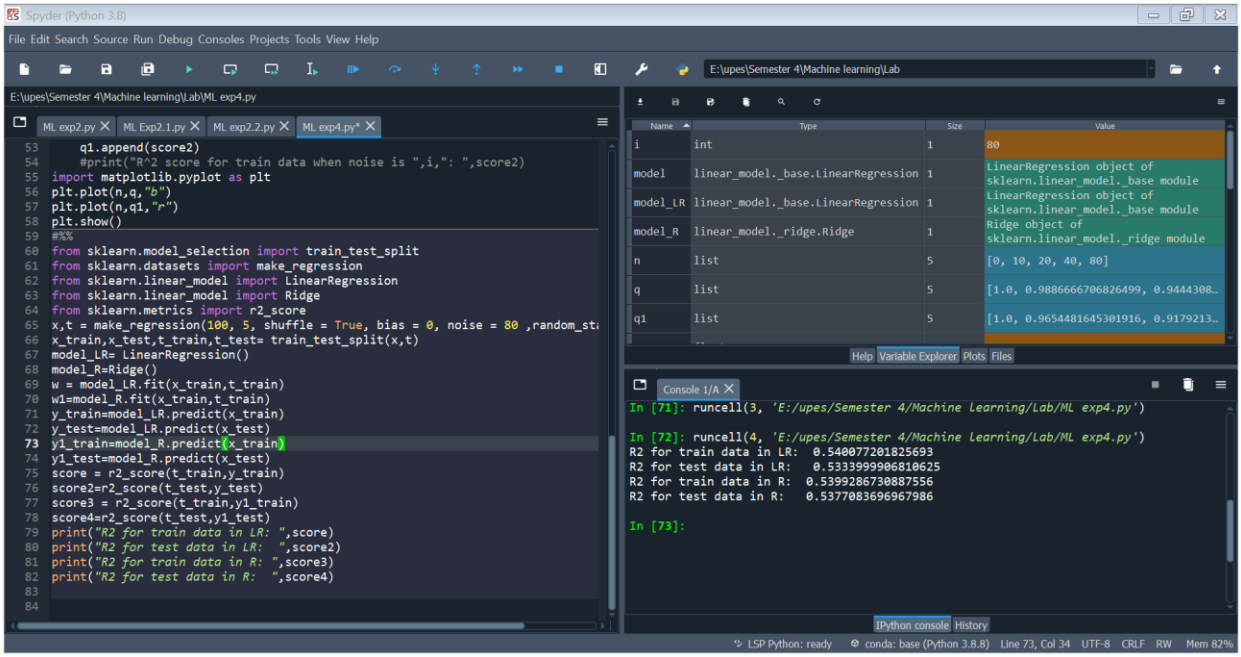
Aim: LR and R model

Code:

```
from sklearn.model_selection import train_test_split

from sklearn.datasets import make_regression

from sklearn.linear_model import LinearRegression

from sklearn.linear_model import Ridge

from sklearn.metrics import r2_score

x,t = make_regression(100, 5, shuffle = True, bias = 0, noise = 80 ,random_state = 1)

x_train,x_test,t_train,t_test= train_test_split(x,t)

model_LR= LinearRegression()

model_R=Ridge()
```

```
w = model_LR.fit(x_train,t_train)

w1=model_R.fit(x_train,t_train)

y_train=model_LR.predict(x_train)

y_test=model_LR.predict(x_test)

y1_train=model_R.predict(x_train)

y1_test=model_R.predict(x_test)

score = r2_score(t_train,y_train)

score2=r2_score(t_test,y_test)

score3 = r2_score(t_train,y1_train)

score4=r2_score(t_test,y1_test)

print("R2 for train data in LR: ",score)

print("R2 for test data in LR:  ",score2)

print("R2 for train data in R: ",score3)

print("R2 for test data in R:  ",score4)
```

Output:



Experiment 5: Classification

Aim: To understand the concept of Classification

Code:

#%%

from sklearn.datasets import make_classification

x,t=make_classification(100,5,n_classes=2)

#%%

from sklearn.model_selection import train_test_split

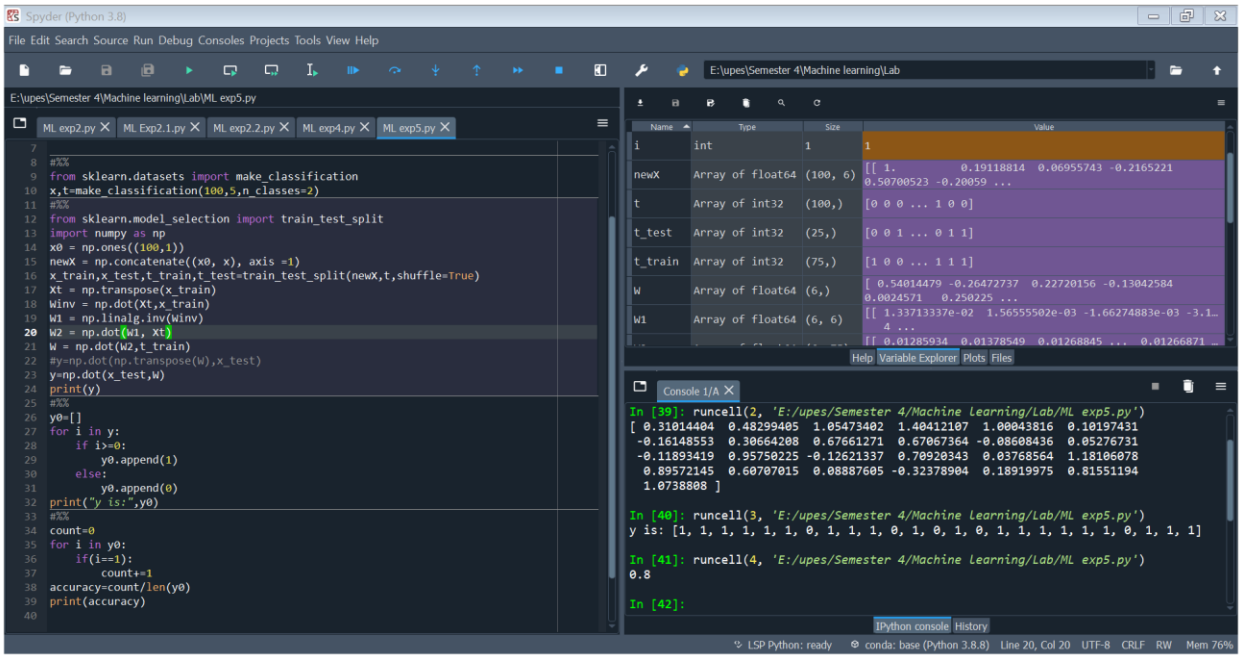
import numpy as np

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

```
newX = np.concatenate((x0, x), axis =1)

x_train,x_test,t_train,t_test=train_test_split(newX,t,shuffle=True)

Xt = np.transpose(x_train)

Winv = np.dot(Xt,x_train)

W1 = np.linalg.inv(Winv)

W2 = np.dot(W1, Xt)

W = np.dot(W2,t_train)

#y=np.dot(np.transpose(W),x_test)

y=np.dot(x_test,W)

print(y)

#%%

y0=[]

for i in y:

    if i>=0:

        y0.append(1)

    else:

        y0.append(0)

print("y is:",y0)
```

```
#%%

count=0

for i in y0:

    if(i==1):

        count+=1

accuracy=count/len(y0)

print(accuracy)
```

Output:



Experiment 6: Multiclass Classification

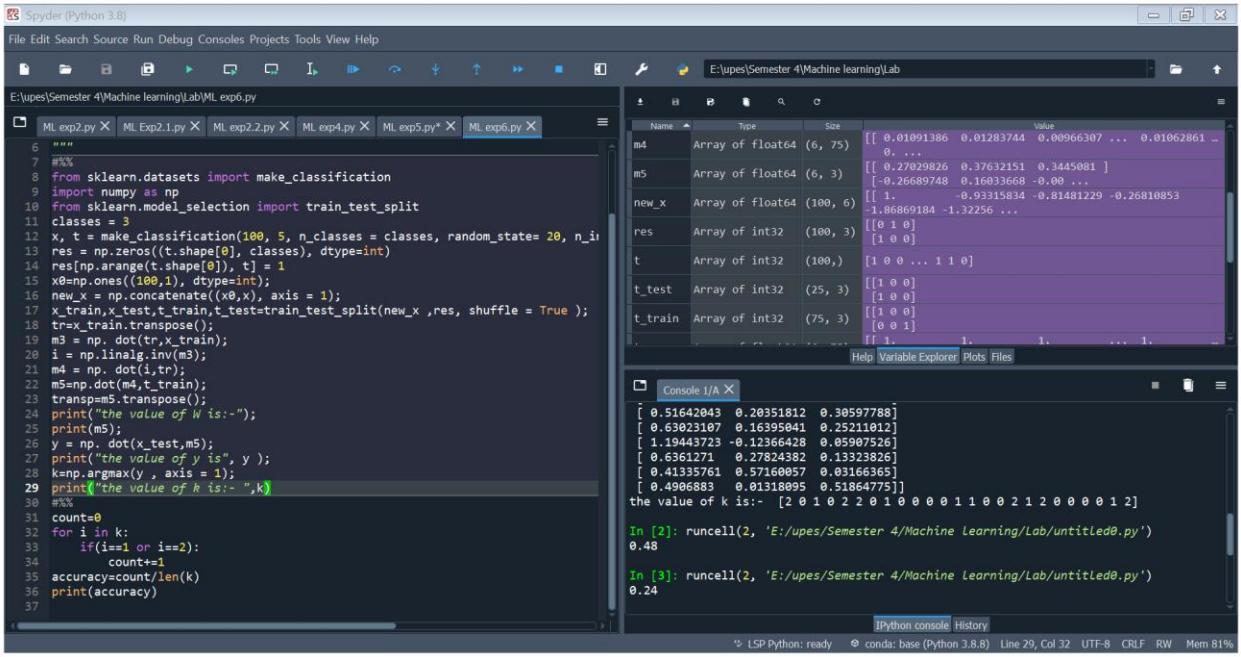
Aim: Multiclass Classification

Code:

```
#%%

from sklearn.datasets import make_classification

import numpy as np

from sklearn.model_selection import train_test_split

classes = 3

x, t = make_classification(100, 5, n_classes = classes, random_state= 20, 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

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

new_x = np.concatenate((x0,x), axis = 1);

x_train,x_test,t_train,t_test=train_test_split(new_x ,res, shuffle = True );

tr=x_train.transpose();

m3 = np. dot(tr,x_train);

i = np.linalg.inv(m3);

m4 = np. dot(i,tr);
```

```
m5=np.dot(m4,t_train);

transp=m5.transpose();

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

print(m5);

y = np. dot(x_test,m5);

print("the value of y is", y );

k=np.argmax(y , axis = 1);

print("the value of k is:- ",k)

#%%

count=0

for i in k:

    if(i==1 or i==2):

        count+=1

accuracy=count/len(k)

print(accuracy)
```

Output:



Experiment 7: Decision Trees

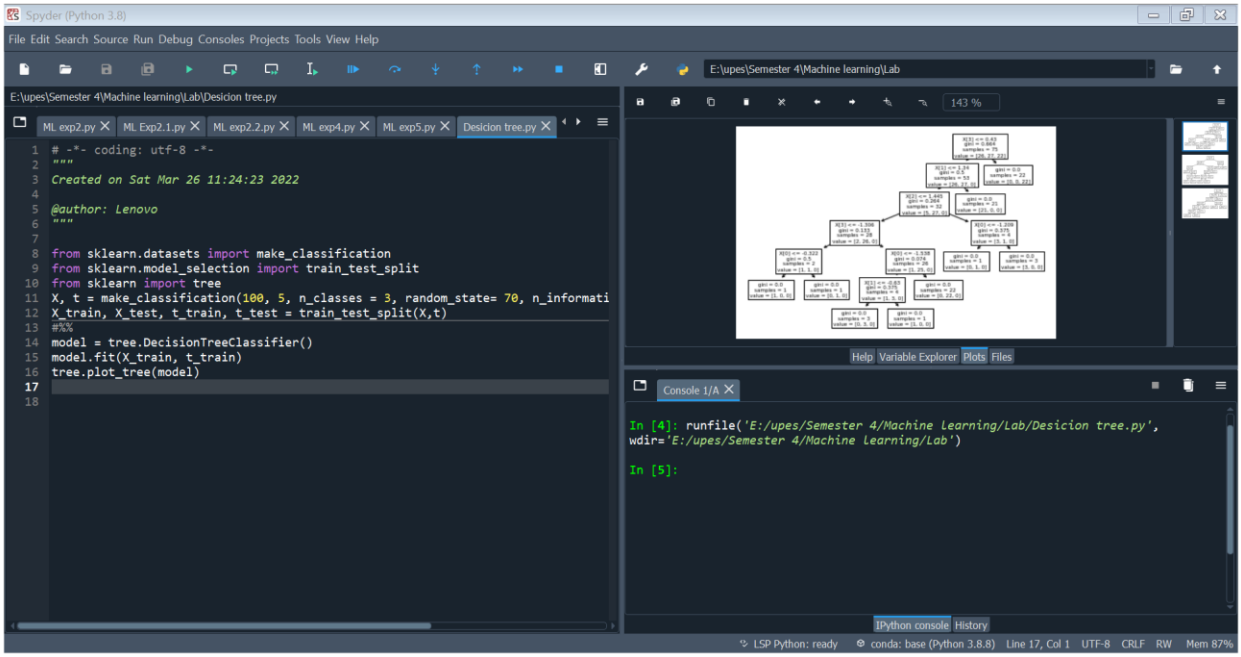
Aim: To understand the concept of Decision trees using classification and regression

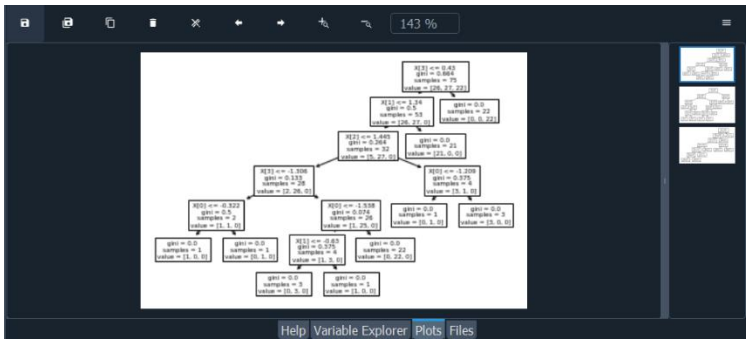
Code:

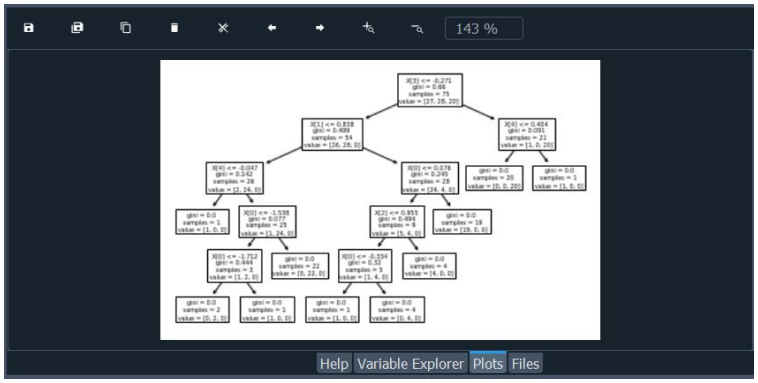
```
from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn import tree

X, t = make_classification(100, 5, n_classes = 3, random_state= 70, n_informative = 2, n_clusters_per_class = 1)

X_train, X_test, t_train, t_test = train_test_split(X,t)

#%%

model = tree.DecisionTreeClassifier()
```

```
model.fit(X_train, t_train)

tree.plot_tree(model)

#%%

from sklearn.datasets import make_regression

X, t = make_regression(100, 5, random_state= 70,noise=10,bias=True)

X_train, X_test, t_train, t_test = train_test_split(X,t)

#%%

from sklearn import tree

model = tree.DesicionTreeRegressor(criterion="ginni")

model.fit(X_train, t_train)

tree.plot_tree(model)
```

Output:


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\upes\Semester 4\Machine learning\Lab
E:\upes\Semester 4\Machine learning\Lab\Desicion tree.py
ML exp2.py
ML Exp2.1.py
ML exp2.2.py
ML exp4.py
ML exp5.py
Desicion tree.py
-*- coding: utf-8 -*-
"""
Created on Sat Mar 26 11:24:23 2022
@author: Lenovo
"""
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn import tree
X, t = make_classification(100, 5, n_classes = 3, random_state= 70, n_informati
X_train, X_test, t_train, t_test = train_test_split(X,t)
#%%
model = tree.DecisionTreeClassifier()
model.fit(X_train, t_train)
tree.plot_tree(model)
143 %
Help
Variable Explorer
Plots
Files
Console 1/A
In [4]: runfile('E:/upes/Semester 4/Machine learning/Lab/Desicion tree.py',
wdir='E:/upes/Semester 4/Machine learning/Lab')
In [5]:
IPython console
History
LSP Python: ready
conda: base (Python 3.8.8)
Line 17, Col 1
UTF-8
CRLF
RW
Mem 87%

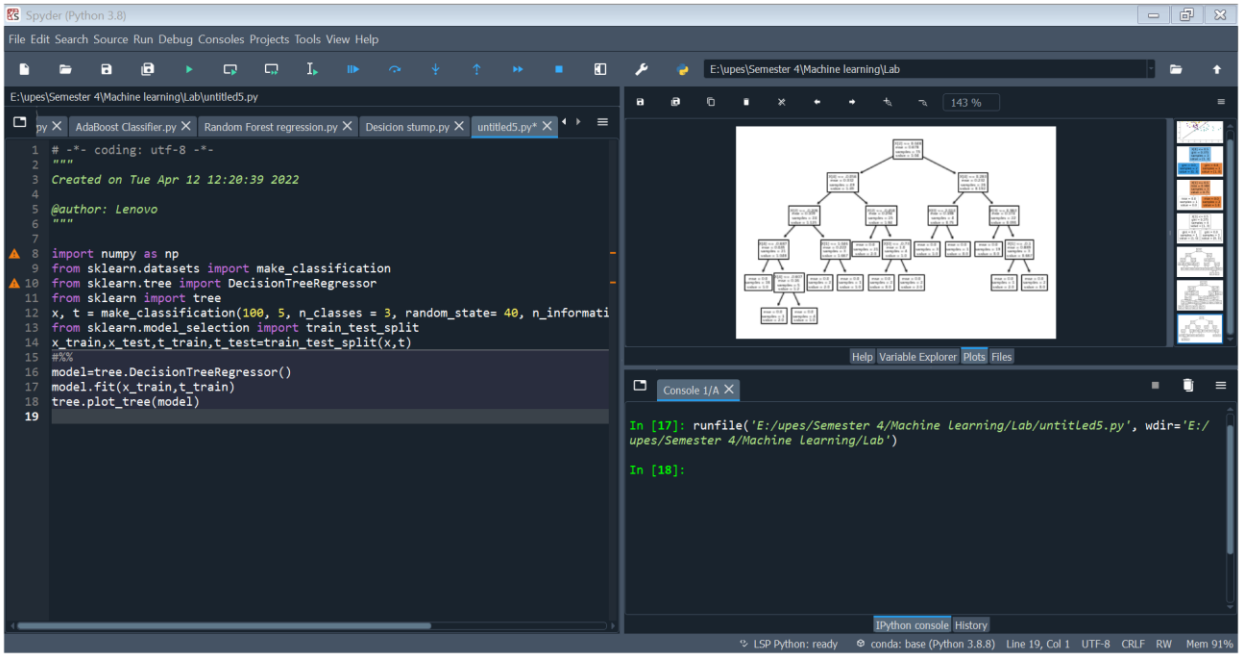

143 %
Help
Variable Explorer
Plots
Files


143 %
Help
Variable Explorer
Plots
Files

Code of Decision tree using regression:

```
import numpy as np

from sklearn.datasets import make_classification

from sklearn.tree import DecisionTreeRegressor

from sklearn import tree

x, t = make_classification(100, 5, n_classes = 3, random_state= 40, n_informative = 2, n_clusters_per_class = 1)

from sklearn.model_selection import train_test_split

x_train,x_test,t_train,t_test=train_test_split(x,t)

#%%

model=tree.DecisionTreeRegressor()

model.fit(x_train,t_train)

tree.plot_tree(model)
```

Output:



Experiment 8: Random Forest AdaBoost

Aim: To understand the concept of Random Forest and Adaboost using classification and regression

Code of Random Forest Classifier:

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])
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")
```

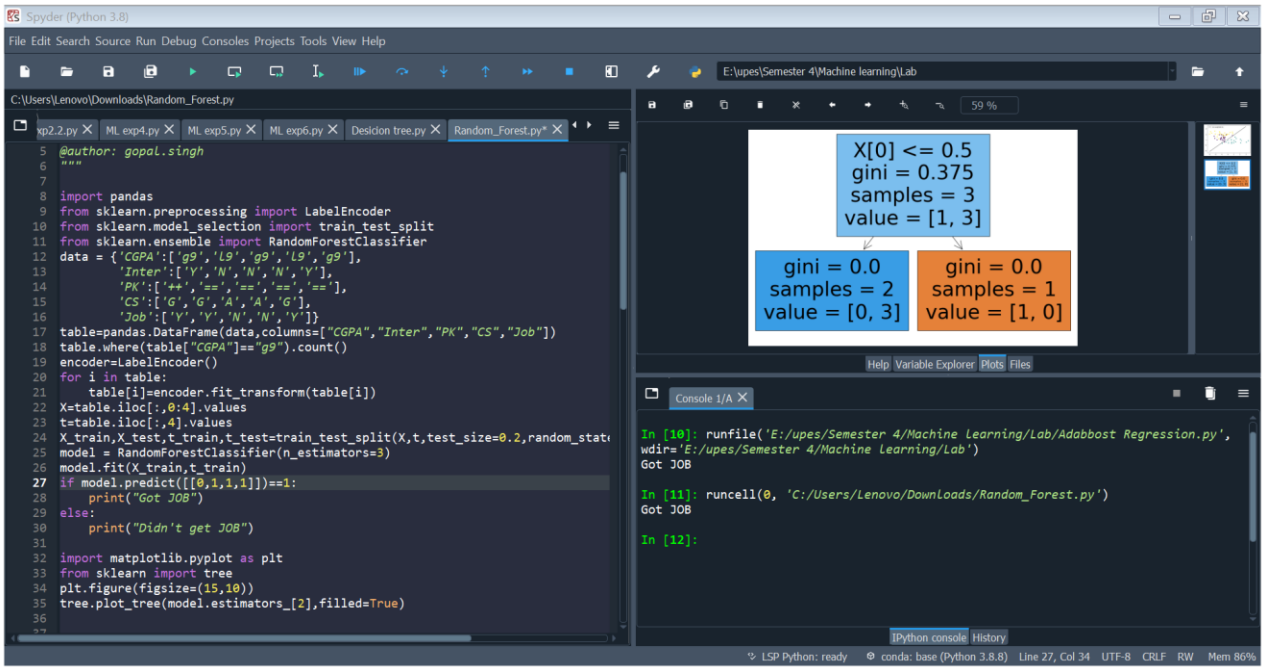
import matplotlib.pyplot as plt

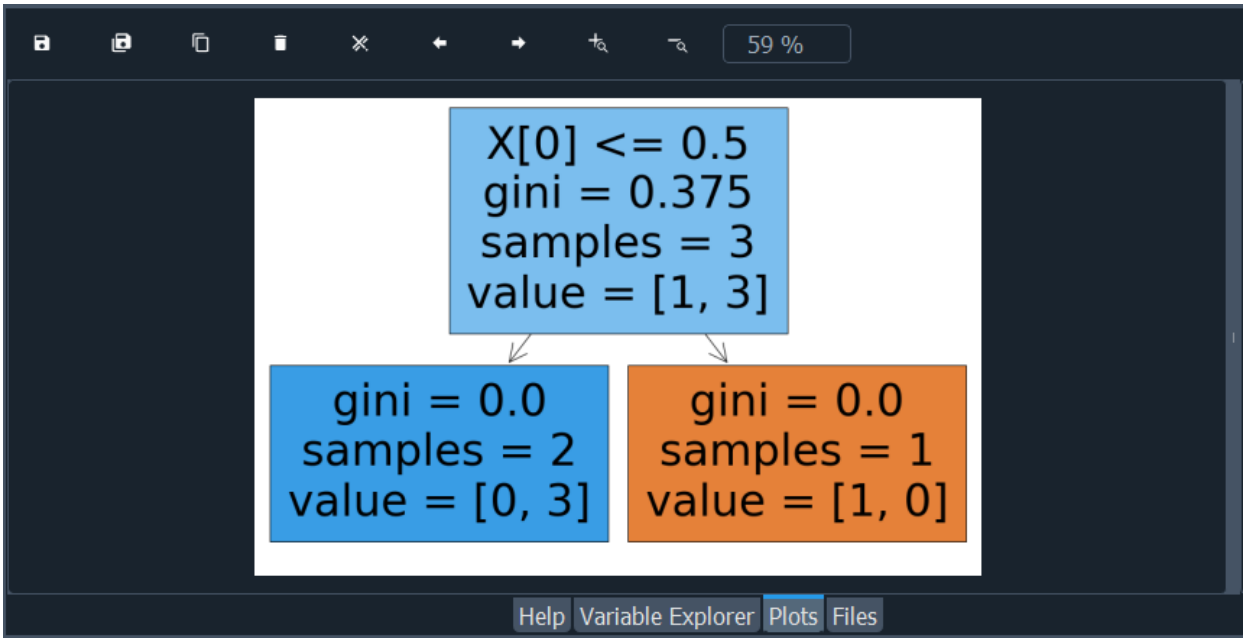
from sklearn import tree

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

tree.plot_tree(model.estimators_[2],filled=True)

Output:



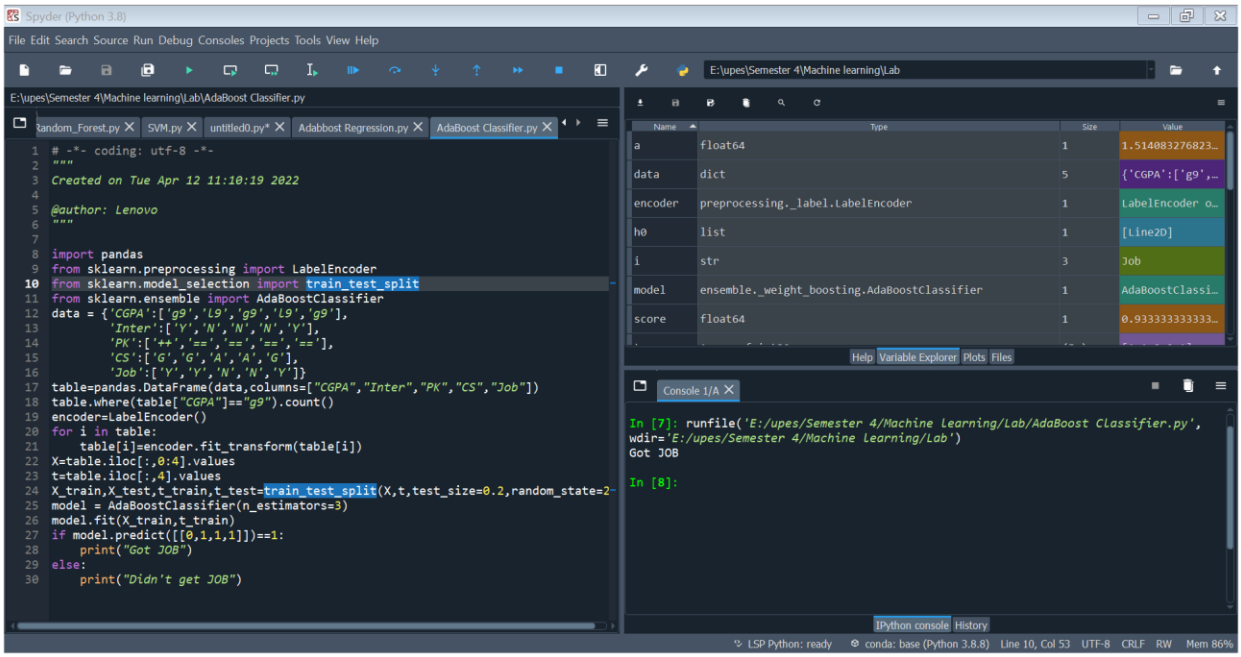


Code of AdaBoost Classifier:

```
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")
```

Output:

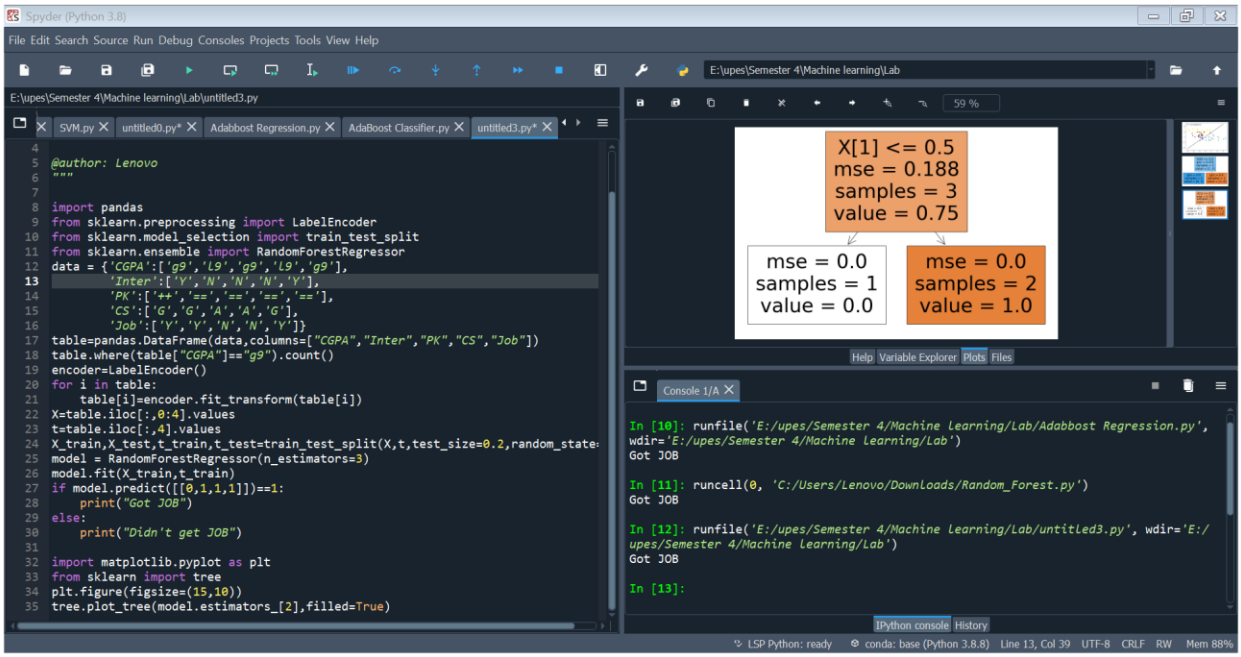


Code of Random Forest using Regression:

```
import pandas

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestRegressor

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 = RandomForestRegressor(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")

import matplotlib.pyplot as plt

from sklearn import tree

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

tree.plot_tree(model.estimators_[2],filled=True)
```

Output:

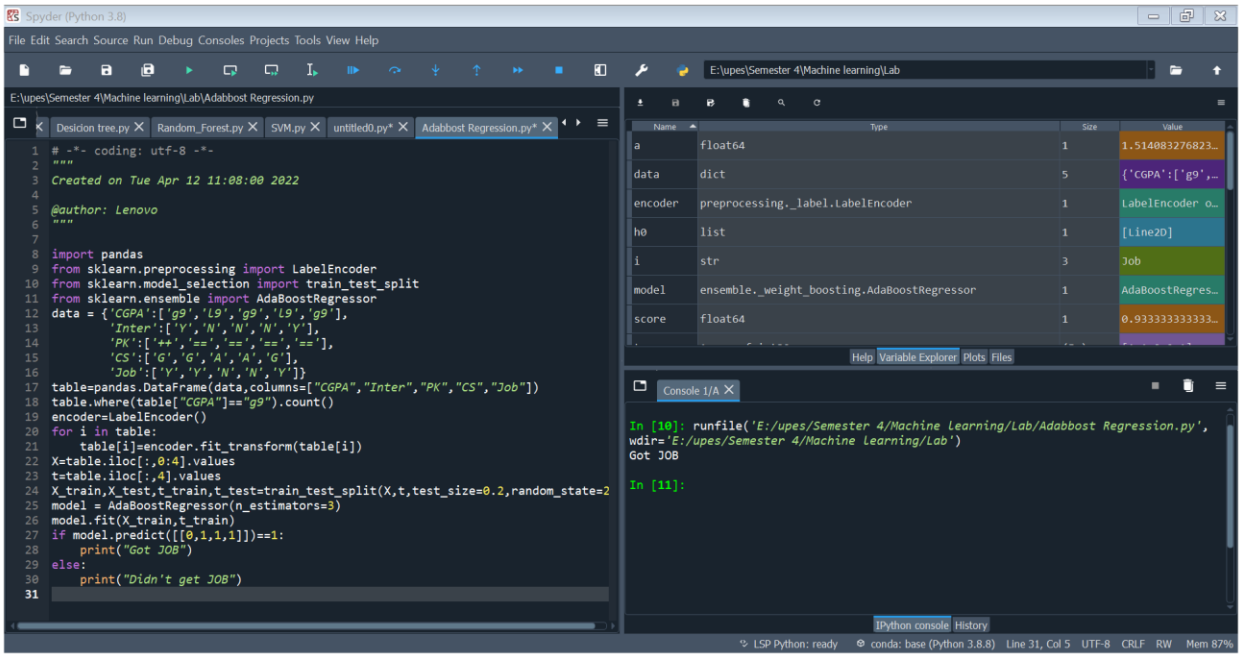


Code of AdaBoost using Regression:

```
import pandas

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostRegressor

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 = AdaBoostRegressor(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")
```

Output:



Code of Decision Stump:

```
from sklearn.metrics import accuracy_score

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

import pandas

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

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

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

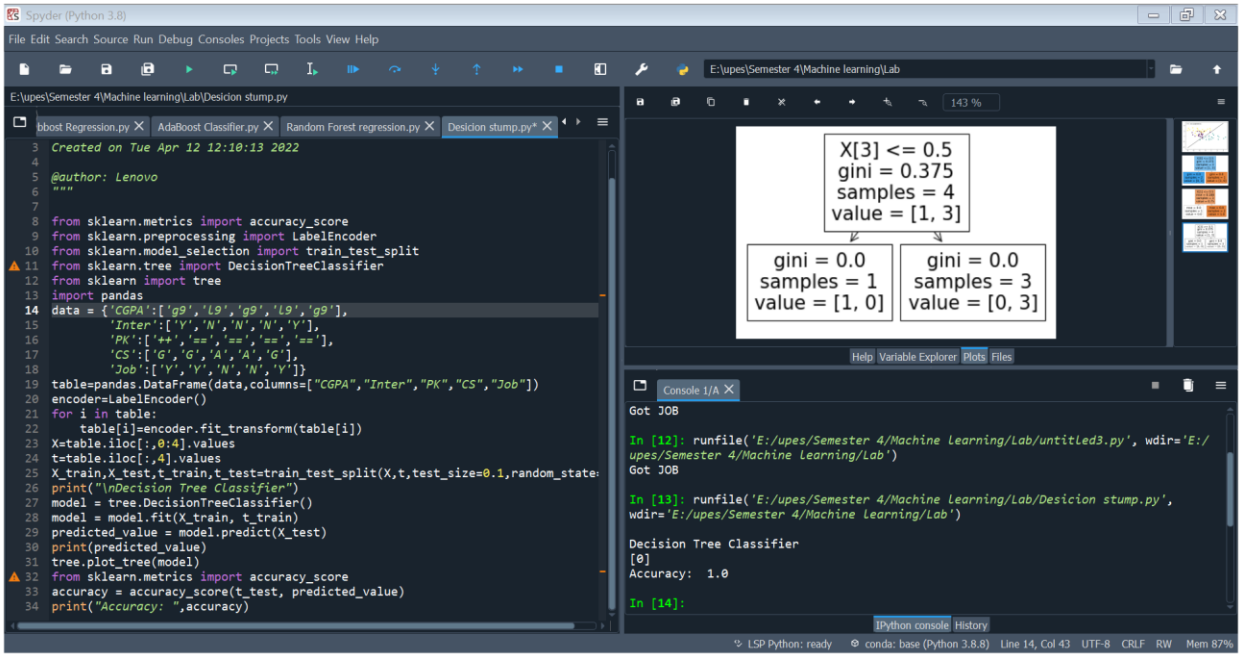
```
        'CS':['G','G','A','A','G'],
        'Job':['Y','Y','N','N','Y']}
table=pandas.DataFrame(data,columns=["CGPA","Inter","PK","CS","Job
"])
encoder=LabelEncoder()
for i in table:
    table[i]=encoder.fit_transform(table[i])
X=table.iloc[:,0:4].values
t=table.iloc[:,4].values
X_train,X_test,t_train,t_test=train_test_split(X,t,test_size=0.1,random_state=
2)
print("\nDecision Tree Classifier")
model = tree.DecisionTreeClassifier()
model = model.fit(X_train, t_train)
predicted_value = model.predict(X_test)
print(predicted_value)
tree.plot_tree(model)
```

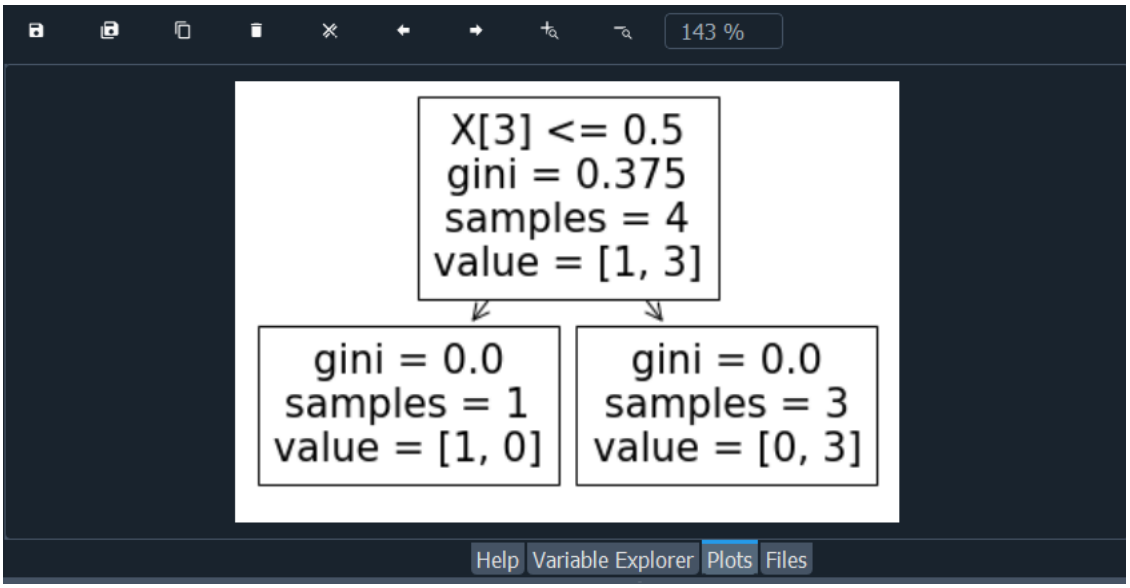
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(t_test, predicted_value)

print("Accuracy: ",accuracy)

Output:





Experiment 9: Neural Network

Aim: To understand the concept of Neural Network

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 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

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

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

epoch = 0
```

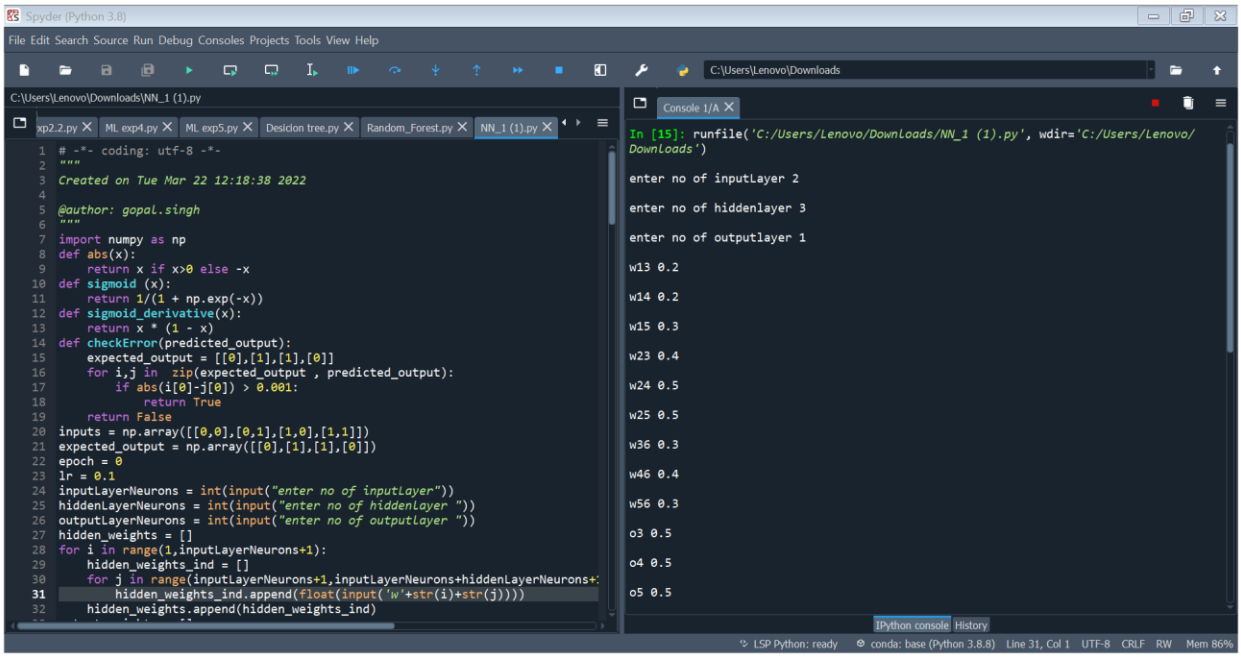
```
lr = 0.1

inputLayerNeurons = 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+ou
tputLayerNeurons+1):

                output_weights_ind.append(float(input('w'+str(i)+str(j))))

        output_weights.append(output_weights_ind)
```

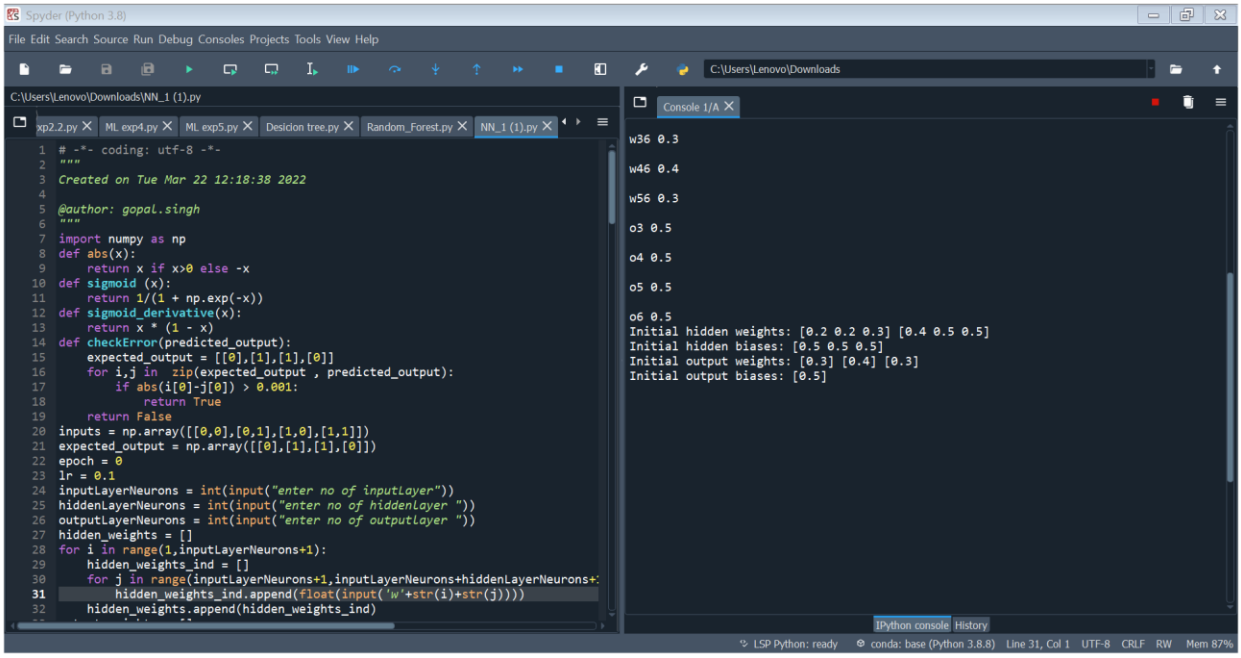
```
hidden_bias = []
output_bias = []
for i in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+outputLayerNeurons+1):
    if i > inputLayerNeurons+hiddenLayerNeurons:
        output_bias.append(float(input("o"+str(i))))
    else:
        hidden_bias.append(float(input("o"+str(i))))
hidden_weights = 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]]

while checkError(predicted_output):

        epoch += 1

        #Forward Propagation

        hidden_layer_activation = np.dot(inputs,hidden_weights)

        hidden_layer_activation += hidden_bias

        hidden_layer_output = sigmoid(hidden_layer_activation)

        output_layer_activation = np.dot(hidden_layer_output,output_weights)

        output_layer_activation += output_bias

        predicted_output = sigmoid(output_layer_activation)

        error = expected_output - predicted_output

        d_predicted_output = error * sigmoid_derivative(predicted_output)

        error_hidden_layer = d_predicted_output.dot(output_weights.T)

        d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

        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:


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Lenovo\Downloads
C:\Users\Lenovo\Downloads\NN_1 (1).py
xp2.2.py
ML exp4.py
ML exp5.py
Desicion tree.py
Random_Forest.py
NN_1 (1).py
-*- coding: utf-8 -*-
"""
Created on Tue Mar 22 12:18:38 2022
@author: gopal.singh
"""
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 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
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 = 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+
hidden_weights_ind.append(float(input('w'+str(i)+str(j))))
hidden_weights.append(hidden_weights_ind)
Console 1/A
In [15]: runfile('C:/Users/Lenovo/Downloads/NN_1 (1).py', wdir='C:/Users/Lenovo/Downloads')
enter no of inputLayer 2
enter no of hiddenlayer 3
enter no of outputlayer 1
w13 0.2
w14 0.2
w15 0.3
w23 0.4
w24 0.5
w25 0.5
w36 0.3
w46 0.4
w56 0.3
o3 0.5
o4 0.5
o5 0.5
IPython console
History
LSP Python: ready
conda: base (Python 3.8.8)
Line 31, Col 1
UTF-8
CRLF
RW
Mem 86%


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Lenovo\Downloads
C:\Users\Lenovo\Downloads\NN_1 (1).py
xp2.2.py
ML exp4.py
ML exp5.py
Desicion tree.py
Random_Forest.py
NN_1 (1).py
-*- coding: utf-8 -*-
"""
Created on Tue Mar 22 12:18:38 2022
@author: gopal.singh
"""
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 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
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 = 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+
hidden_weights_ind.append(float(input('w'+str(i)+str(j))))
hidden_weights.append(hidden_weights_ind)
Console 1/A
w36 0.3
w46 0.4
w56 0.3
o3 0.5
o4 0.5
o5 0.5
o6 0.5
Initial hidden weights: [0.2 0.2 0.3] [0.4 0.5 0.5]
Initial hidden biases: [0.5 0.5 0.5]
Initial output weights: [0.3] [0.4] [0.3]
Initial output biases: [0.5]
IPython console
History
LSP Python: ready
conda: base (Python 3.8.8)
Line 31, Col 1
UTF-8
CRLF
RW
Mem 87%

```
In [15]: runfile('C:/Users/Lenovo/Downloads/NN_1 (1).py', wdir='C:/Users/Lenovo/
Downloads')

enter no of inputLayer 2

enter no of hiddenlayer 3

enter no of outputlayer 1

w13 0.2

w14 0.2

w15 0.3

w23 0.4

w24 0.5

w25 0.5

w36 0.3

w46 0.4

w56 0.3

o3 0.5

o4 0.5

o5 0.5
```

Console 1/A

```
w25 0.5

w36 0.3

w46 0.4

w56 0.3

o3 0.5

o4 0.5

o5 0.5

o6 0.5
Initial hidden weights: [0.2 0.2 0.3] [0.4 0.5 0.5]
Initial hidden biases: [0.5 0.5 0.5]
Initial output weights: [0.3] [0.4] [0.3]
Initial output biases: [0.5]
```

Experiment 10: Support vector machine

Aim: To understand the concept of support vector machine.

Code:

```
import numpy as np

from sklearn.datasets import make_classification

x, t = make_classification(100, 5, n_classes = 3, random_state= 50, n_informative = 3, n_clusters_per_class = 1)

from sklearn.model_selection import train_test_split

x_train,x_test, y_train , y_test = train_test_split(x,t,test_size = 0.3 )

from sklearn import svm

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

model.fit(x_train,y_train)

y_pred = model.predict(x_test)

from sklearn import metrics

score = metrics.accuracy_score(y_test,y_pred)

print(score)

#%%

import matplotlib.pyplot as plt

w = model.coef_[0]
```

a = -w[0]/w[1]

xx = np.linspace(-3, 3)

yy = a *xx - model.intercept_[0]/w[1]

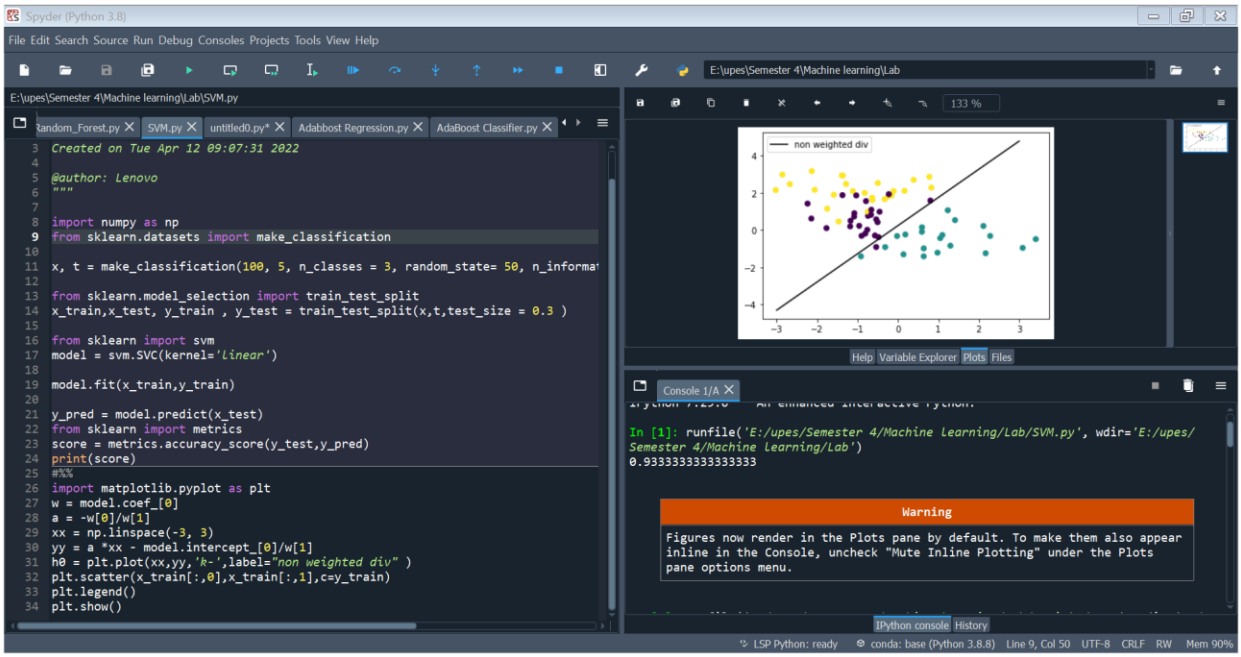
h0 = plt.plot(xx,yy,'k-',label="non weighted div")

plt.scatter(x_train[:,0],x_train[:,1],c=y_train)

plt.legend()

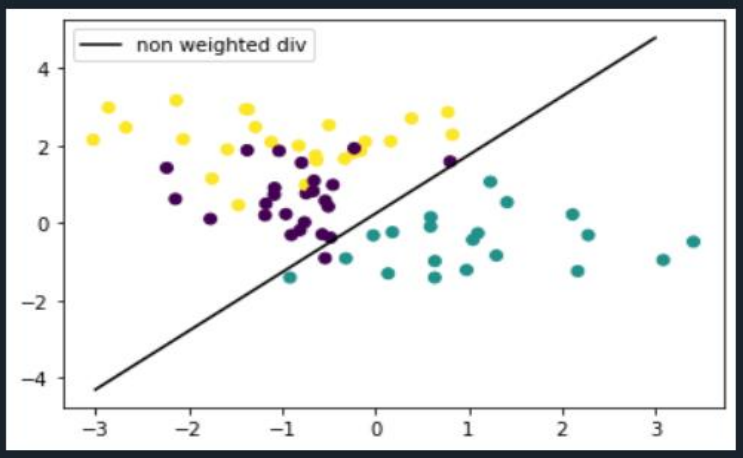
plt.show()

Output:



Console 1/A

```
In [1]: runfile('E:/upes/Semester 4/Machine learning/Lab/SVM.py', wdir='E:/upes/
Semester 4/Machine learning/Lab')
0.9333333333333333
```



Help | Variable Explorer | Plots | Files