GR20 Regulations

III B.Tech II Semester

Machine Learning Lab

(GR20A3112)

Department of Computer Science and Engineering

GOKARAJU RANGARAJU
INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Autonomous)

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

MACHINE LEARNING AND APPLICATIONS LAB

Course code: GR20A3122 **L/T/P/C: 0/0/3/1.5**

Course Objectives:

- Learn usage of Libraries for Machine Learning in Python
- Demonstrate Dimensionality reduction methods
- Describe appropriate supervised learning algorithms for a given problem.
- Explore back propagation algorithm and ensemble methods
- Discuss different unsupervised learning algorithms

Course Outcomes:

- Illustrate the applications of Python Machine Learning Libraries.
- Apply Dimensionality reduction methods for Machine Learning Tasks.
- Design and analyze various supervised learning mechanisms.
- Develop back propagation algorithm and Random Forest Ensemble method.
- Design and analyze various unsupervised learning algorithms.

Note: Implement the following Machine Learning Tasks using Python / R-Tool

Task 1: Write a python program to import and export data using Pandas library functions.

Task 2: Demonstrate various data pre-processing techniques for a given dataset.

Task 3: Implement Dimensionality reduction using Principle Component Analysis (PCA) method.

Task 4: Write a Python program to demonstrate various Data Visualization Techniques.

Task 5: Implement Simple and Multiple Linear Regression Models.

Task 6: Develop Logistic Regression Model for a given dataset.

Task 7: Develop Decision Tree Classification model for a given dataset and use it to classify a new sample.

Task 8: Implement Naïve Bayes Classification in Python

Task 9: Build KNN Classification model for a given dataset.

Task 10: Build Artificial Neural Network model with back propagation on a given dataset.

Task 11: a) Implement Random Forest ensemble method on a given dataset.

b) Implement Boosting ensemble method on a given dataset.

Task 12: Write a python program to implement K-Means clustering Algorithm

INDEX

TASK-1

Task-1: Write a python program to import and export data using Pandas

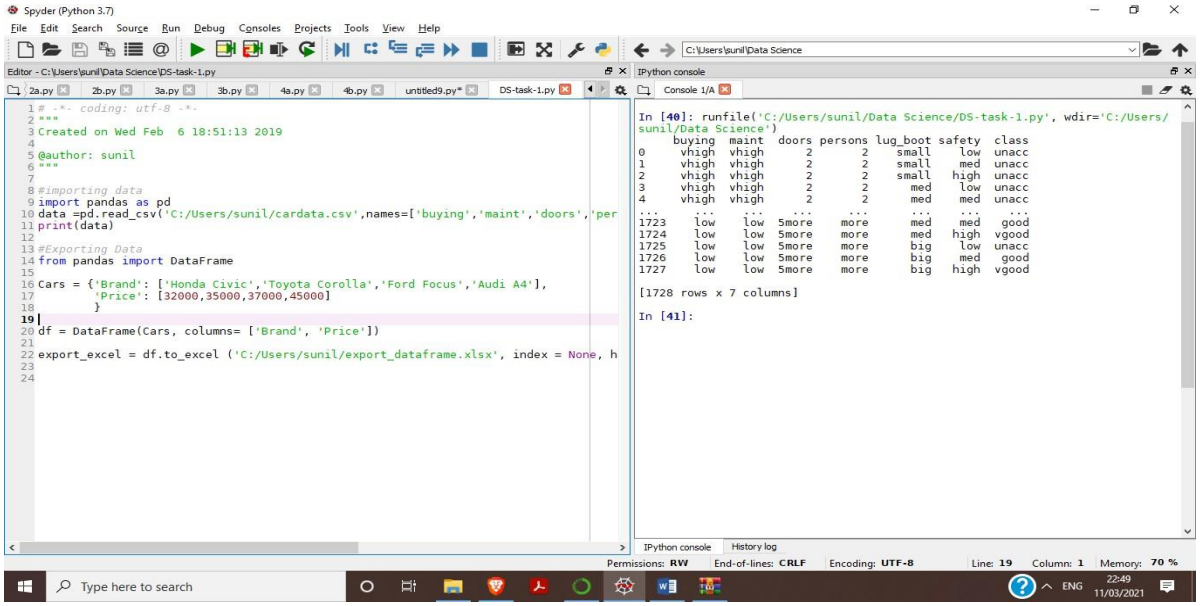
library functions

Aim: To implement a Python script for importing and exporting data using Pandas in Python

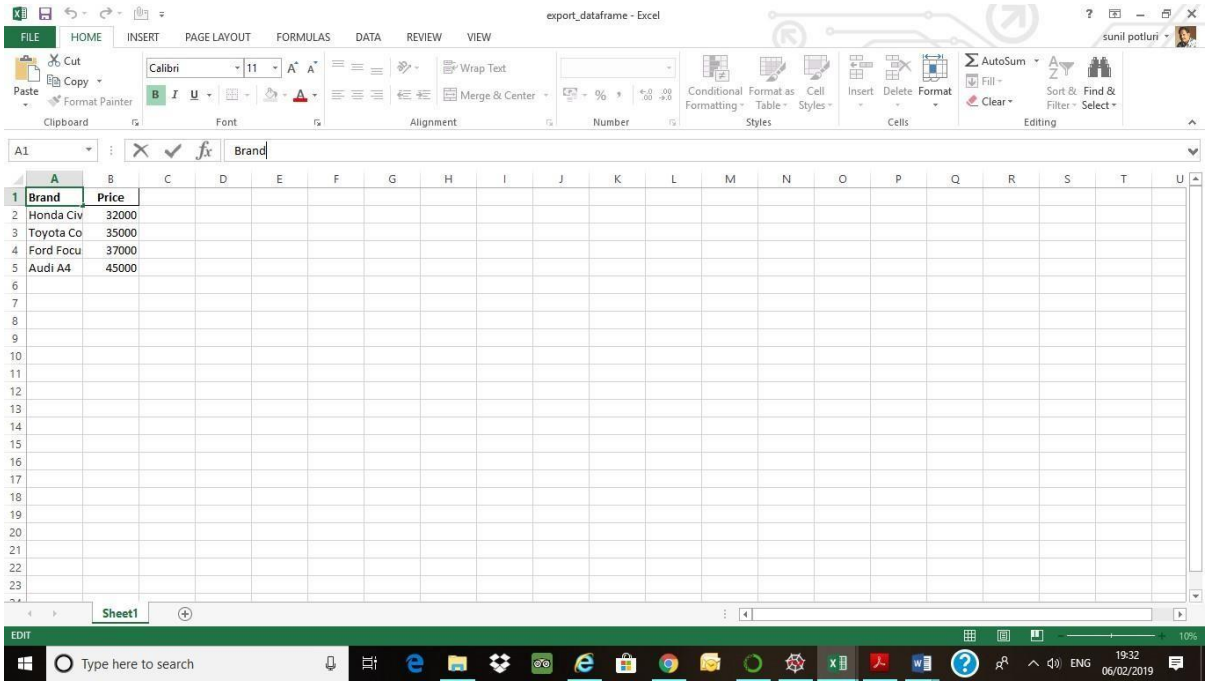
Program:

```
#importing
data import
pandas as pd
data
=pd.read_csv('C:/Users/sunil/cardata.csv',names=['buying','maint','doors','persons','lug_boot','safety',
'c
las
s'])
pri
nt(
dat
a)


#Exporting Data
from pandas import DataFrame


Cars = {'Brand': ['Honda Civic','Toyota Corolla','Ford
      Focus','Audi A4'], 'Price': [32000,35000,37000,45000]
      }


df = DataFrame(Cars, columns= ['Brand', 'Price'])


export_excel = df.to_excel ('C:/Users/sunil/export_dataframe.xlsx', index = None,
header=True) #Don't forget to add '.xlsx' at the end of the path
```

Output:



After Export:



TASK-2

Task-2: Demonstrate various data pre-processing techniques for a given dataset

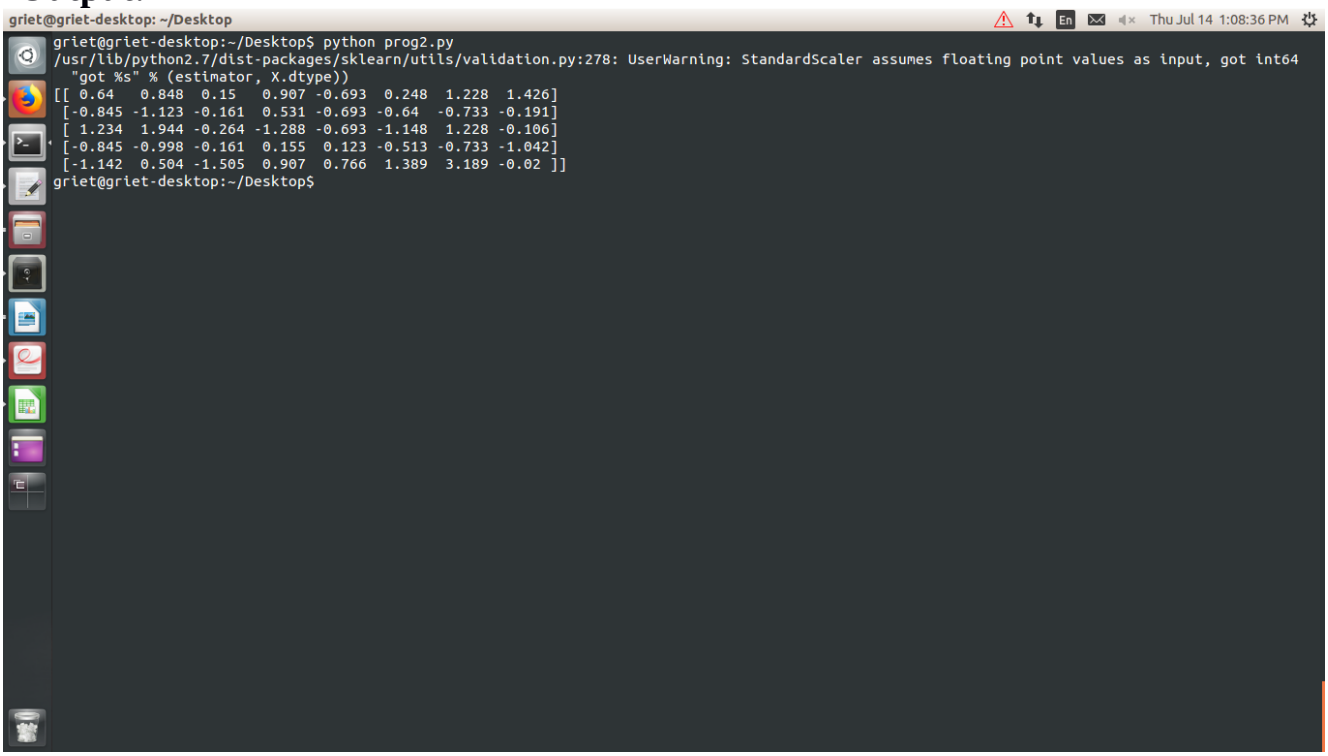
Aim: To provide various data pre-processing techniques for a given data set are there. They are

a) Standard Scaler b) Binarizer c) Min MaxScaler

Program: a) Standard Scaler

```
# importing libraries
from sklearn.preprocessing import StandardScaler
import pandas
import numpy
# data set link
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
# preparating of dataframe using the data at given link and defined
columnslist
dataframe = pandas.read_csv('//home//griet//Desktop//ML//diabetes.csv',
names = names)
array = dataframe.values
# separate array into input and output components
X = array[:, 0:8]
Y = array[:, 8]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarize transformed data
numpy.set_printoptions(precision = 3)
print(rescaledX[0:5,:])
```

Output:



b) Binarizer

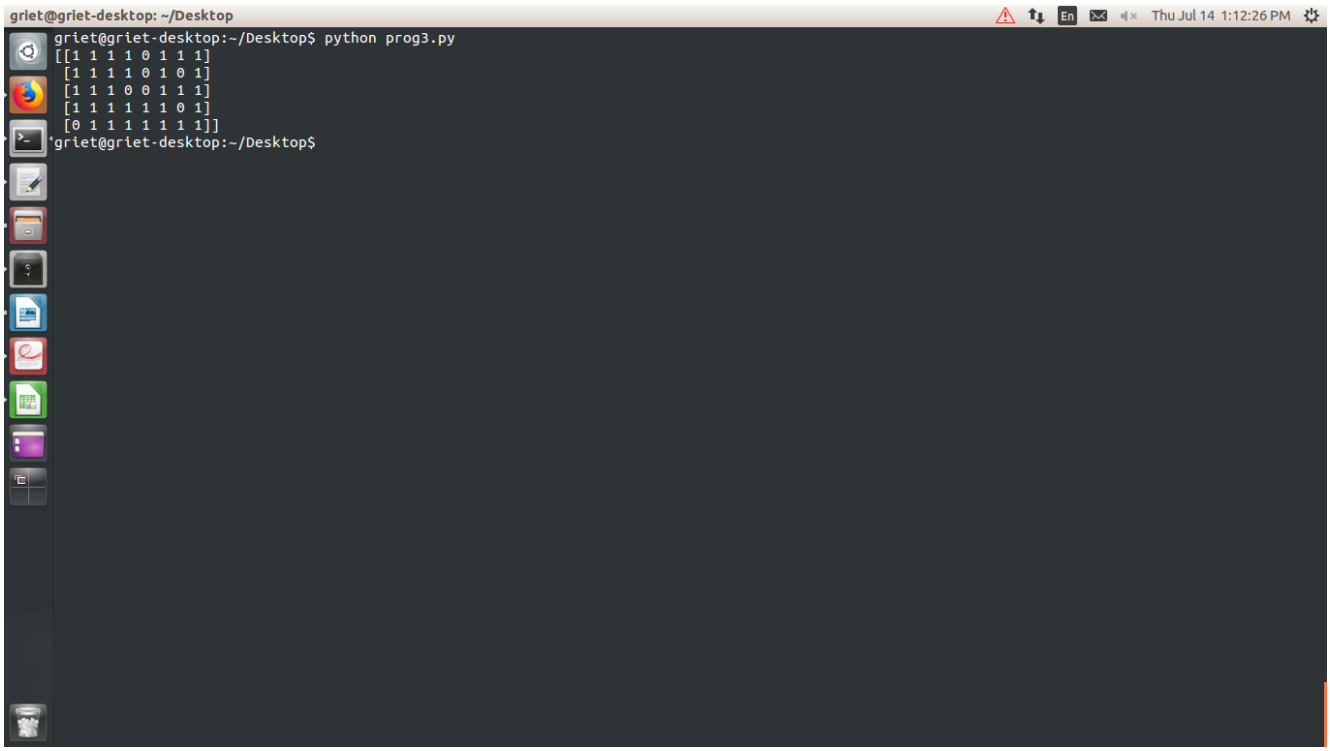
```
# import libraries
from sklearn.preprocessing import Binarizer
import pandas
import numpy

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
```

```
# preparating of dataframe using the data at given link and defined columns list
dataframe = pandas.read_csv('//home//griet//Desktop//ML//diabetes.csv', names = names)
array = dataframe.values
# separate array into input and output components
X = array[:, 0:8]
Y = array[:, 8]
binarizer = Binarizer(threshold = 0.0).fit(X)
binaryX = binarizer.transform(X)
# summarize transformed data
numpy.set_printoptions(precision = 3)
print(binaryX[0:5,:])
```

Output:

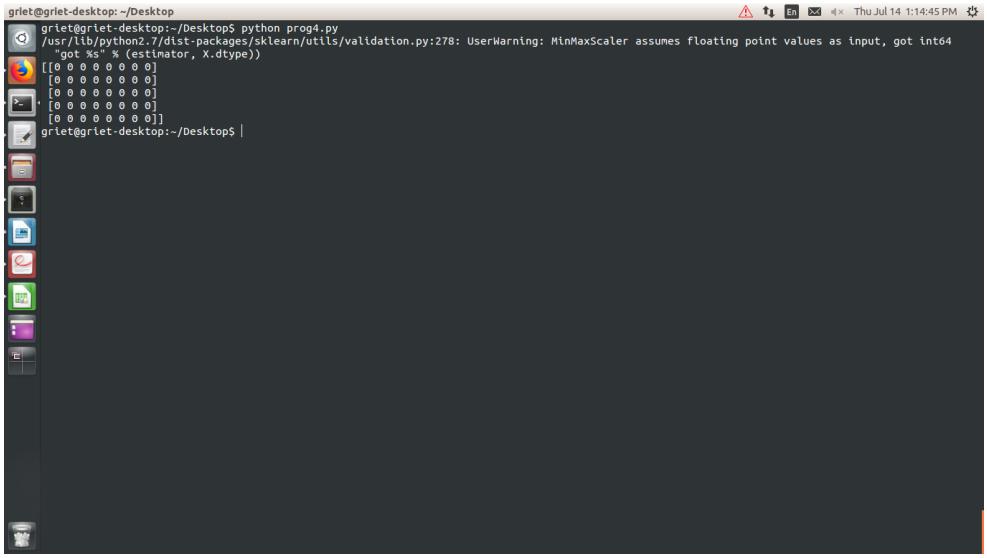
c)MinMaxScaler

```
# importing libraries

import pandas
```



```
import scipy
import numpy
from sklearn.preprocessing import MinMaxScaler
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi','age','class']
# preparating of dataframe using the data at given link and
defined columns list
dataframe =
pandas.read_csv('//home//griet//Desktop//ML//diabetes.csv',
names = names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
# initialising the MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
# learning the statistical parameters for each of the data and
transforming
rescaledX = scaler.fit_transform(X)
# summarize transformed data
numpy.set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

Output:


griet@griet-desktop: ~/Desktop
Thu Jul 14 1:14:45 PM
griet@griet-desktop:~/Desktop$ python prog4.py
/usr/lib/python2.7/dist-packages/sklearn/utils/validation.py:278: UserWarning: MinMaxScaler assumes floating point values as input, got int64
"got %s" % (estimator, X.dtype))
[[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]]
griet@griet-desktop:~/Desktop$

TASK-3

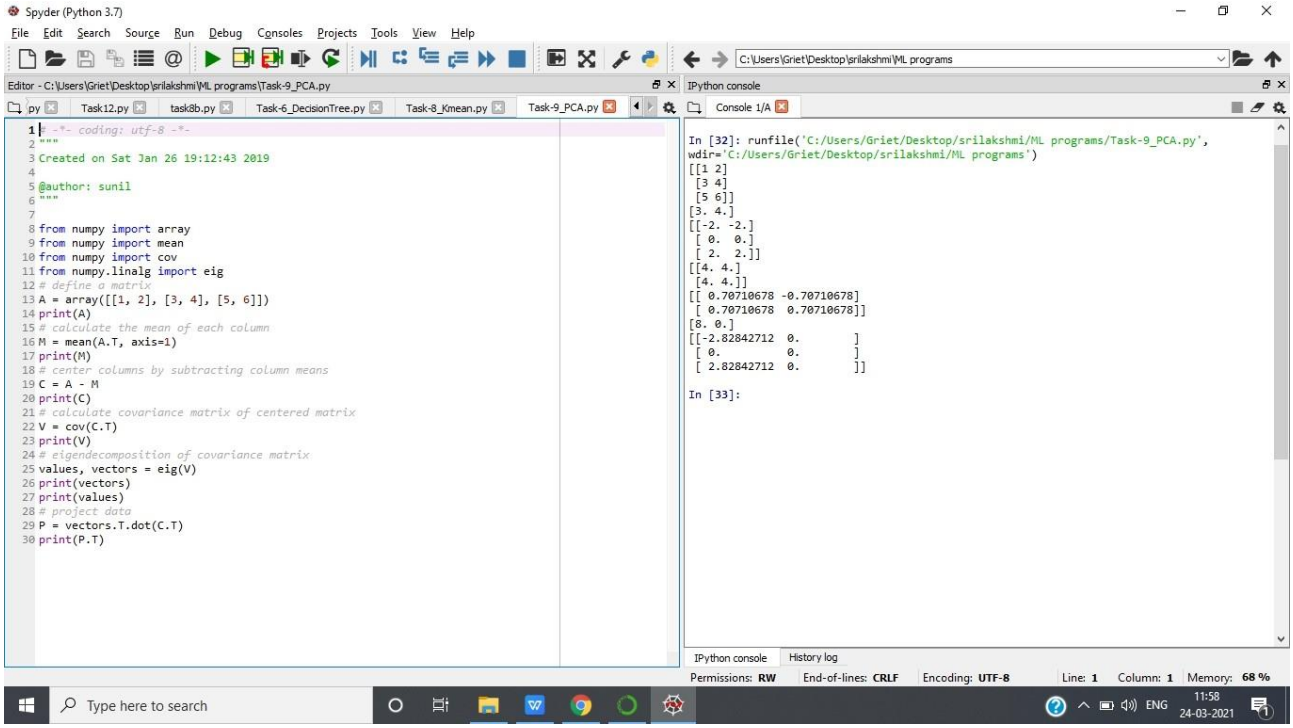
Task-3: Implement Dimensionality reduction using Principle Component Analysis (PCA)method.

Aim: To Implement Principle Component Analysis for Dimensionality Reduction

Program:

```
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
# center columns by subtracting column means
C = A - M
print(C)
# calculate covariance matrix of centered matrix
V = cov(C.T)
print(V)
# eigendecomposition of covariance matrix
values, vectors = eig(V)
print(vectors)
print(values)
# project data
P = vectors.T.dot(C.T)
print(P.T)
```

Output:


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\Griet\Desktop\srilakshmi\ML programs\Task-9_PCA.py
Task12.py
task8b.py
Task-6_DecisionTree.py
Task-8_Kmean.py
Task-9_PCA.py
-*- coding: utf-8 -*-
"""
Created on Sat Jan 26 19:12:43 2019
@author: sunil
"""
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
# center columns by subtracting column means
C = A - M
print(C)
# calculate covariance matrix of centered matrix
V = cov(C.T)
print(V)
# eigendecomposition of covariance matrix
values, vectors = eig(V)
print(vectors)
print(values)
# project data
P = vectors.T.dot(C.T)
print(P.T)
IPython console
Console 1/A
In [32]: runfile('C:/Users/Griet/Desktop/srilakshmi/ML programs/Task-9_PCA.py',
wdir='C:/Users/Griet/Desktop/srilakshmi/ML programs')
[[1 2]
[3 4]
[5 6]]
[3. 4.]
[[-2. -2.]
[0. 0.]
[2. 2.]]
[[4. 4.]
[4. 4.]]
[[0.70710678 -0.70710678]
[0.70710678 0.70710678]]
[8. 0.]
[[-2.82842712 0.]
[0. 0.]
[2.82842712 0.]]
In [33]:
IPython console
History log
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 1
Column: 1
Memory: 68 %
Type here to search
ENG
11:58
24-03-2021

TASK-4

Task 4: Write a Python program to demonstrate various Data Visualization Techniques

Aim: To implement various visualization techniques in python.

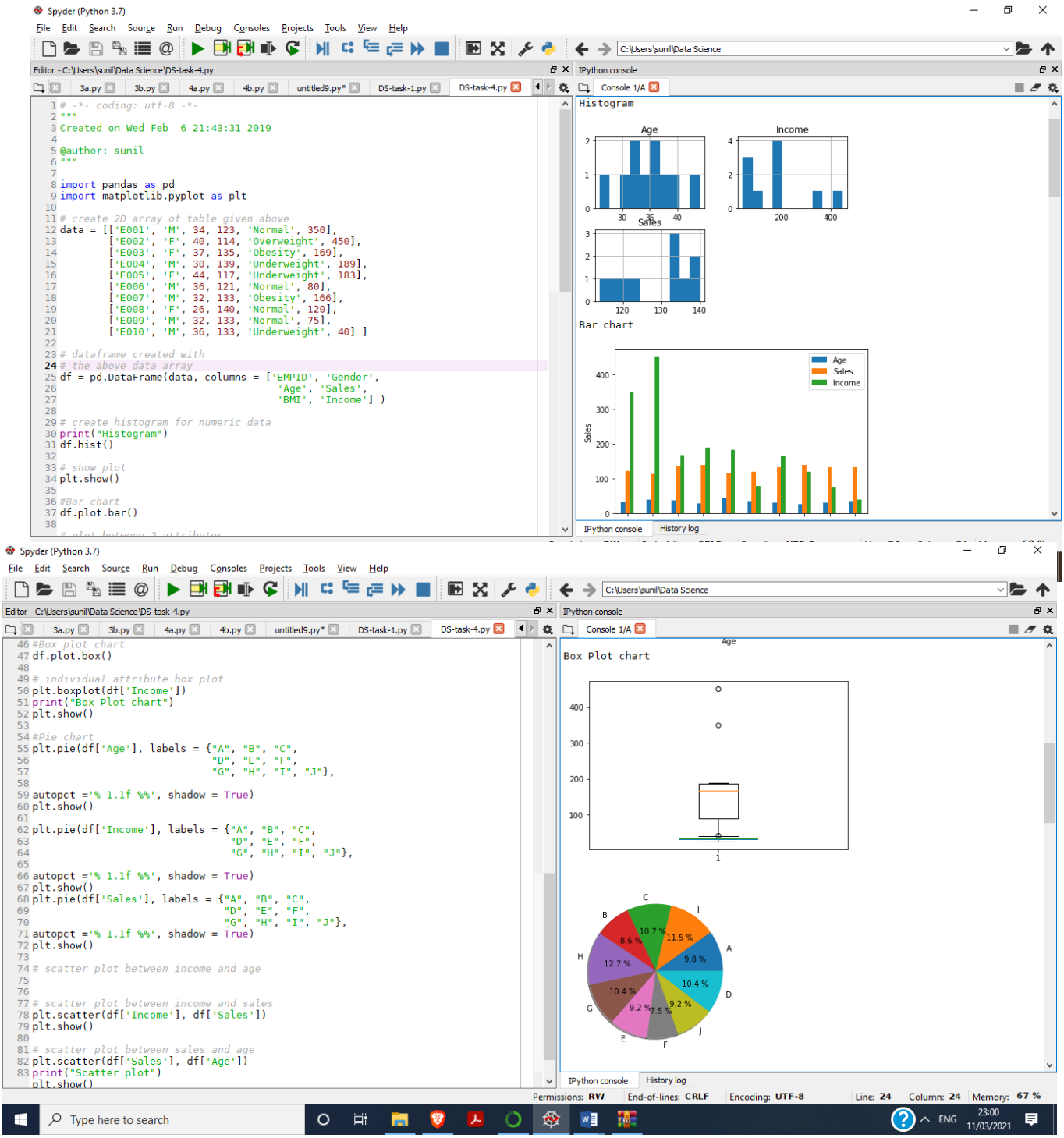
Program:

```
import pandas as pd
import matplotlib.pyplot as plt
# create 2D array of table given above
data = [['E001', 'M', 34, 123, 'Normal', 350],
        ['E002', 'F', 40, 114, 'Overweight', 450],
        ['E003', 'F', 37, 135, 'Obesity', 169],
        ['E004', 'M', 30, 139, 'Underweight', 189],
        ['E005', 'F', 44, 117, 'Underweight', 183],
        ['E006', 'M', 36, 121, 'Normal', 80],
        ['E007', 'M', 32, 133, 'Obesity', 166],
        ['E008', 'F', 26, 140, 'Normal', 120],
        ['E009', 'M', 32, 133, 'Normal', 75],
        ['E010', 'M', 36, 133, 'Underweight', 40] ]
# dataframe created with
# the above data array
df = pd.DataFrame(data, columns = ['EMPID', 'Gender',
                                   'Age', 'Sales',
                                   'BMI', 'Income'] )
# create histogram for numeric data
print("Histogram")
df.hist()
# show plot
plt.show()
#Bar chart
df.plot.bar()
# plot between 2 attributes
plt.bar(df['Age'], df['Sales'])
plt.xlabel("Age")
plt.ylabel("Sales")
print("Bar chart")
plt.show()
```

```
#Box
plotchart
df.plot.box(
)
    # individual attribute
    box plot
    plt.boxplot(df['Incom
    e']) print("Box Plot
    chart") plt.show()


    #Pie chart
    plt.pie(df['Age'], labels = {"A", "B", "C",
                            "D", "E", "F",
                            "G", "H", "I", "J"},

    autopct ='% 1.1f %%', shadow
    = True) plt.show()


    plt.pie(df['Income'], labels = {"A", "B", "C",
                              "D", "E", "F",
                              "G", "H", "I", "J"},

    autopct ='% 1.1f %%', shadow
    = True) plt.show()
    plt.pie(df['Sales'], labels = {"A", "B", "C",
                            "D", "E", "F",
                            "G", "H", "I", "J"},
    autopct ='% 1.1f %%', shadow
    = True) plt.show()

    # scatter plot between income and age


    # scatter plot between income
    and sales
    plt.scatter(df['Income'],
    df['Sales']) plt.show()
```

```
# scatter plot between sales and
age plt.scatter(df['Sales'],
df['Age']) print("Scatter plot")
plt.show()
```

Output:



Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\sunil\Data Science

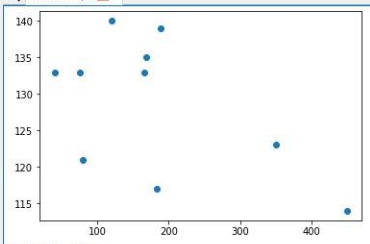
Editor - C:\Users\sunil\Data Science\DS-task-4.py

3a.py 3b.py 4a.py 4b.py untitled9.py* DS-task-1.py DS-task-4.py

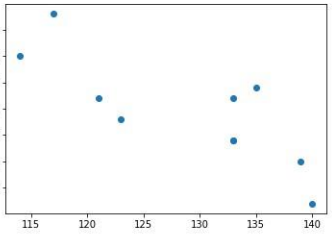
```
# individual attribute box plot
plt.boxplot(df['Income'])
print("Box Plot chart")
plt.show()

#Pie chart
plt.pie(df['Age'], labels = {"A", "B", "C",
                             "D", "E", "F",
                             "G", "H", "I", "J"},

autopct ='% 1.1f %%', shadow = True)
plt.show()

plt.pie(df['Income'], labels = {"A", "B", "C",
                                "D", "E", "F",
                                "G", "H", "I", "J"},

autopct ='% 1.1f %%', shadow = True)
plt.show()
plt.pie(df['Sales'], labels = {"A", "B", "C",
                               "D", "E", "F",
                               "G", "H", "I", "J"},
autopct ='% 1.1f %%', shadow = True)
plt.show()

# scatter plot between income and age


# scatter plot between income and sales
plt.scatter(df['Income'], df['Sales'])
plt.show()

# scatter plot between sales and age
plt.scatter(df['Sales'], df['Age'])
print("Scatter plot")
plt.show()


```

IPython console

Console 1/A



Scatter plot



In [43]:

IPython console History log

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 24 Column: 24 Memory: 67 %

Type here to search

ENG 23:00 11/03/2021

TASK-5

Task-5: Implement Simple and Multiple Linear Regression Models.

Aim: To Develop a python program for Simple and Multiple Linear Regression

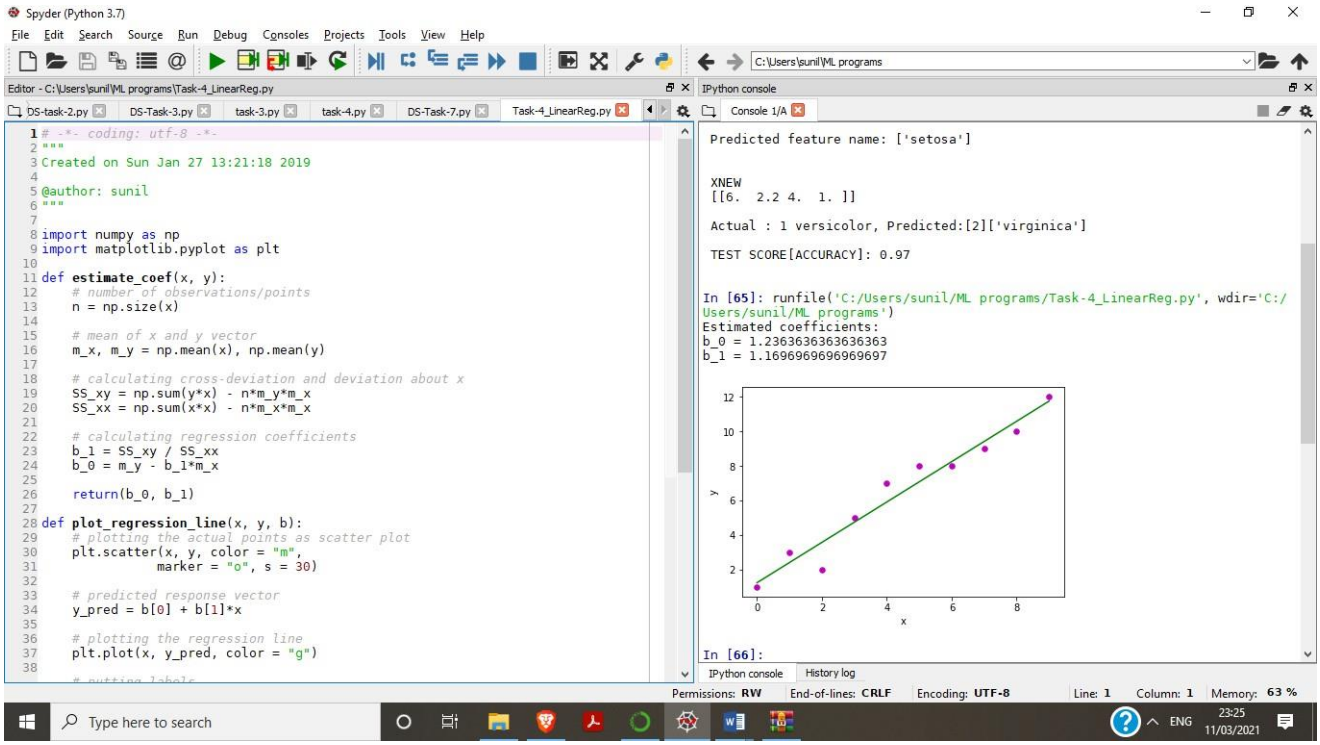
a) Simple Linear Regression Model:

Program:

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of
    observations/pointsn =
    np.size(x)

    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)

    # calculating cross-deviation and deviation
    about xSS_xy = np.sum(y*x) -
    n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression
    coefficientsb_1 = SS_xy /
    SS_xx
    b_0 = m_y -

    b_1*m_x

    return(b_0,

    b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as
    scatter plotplt.scatter(x, y, color
    = "m",
            marker = "o", s = 30)

    # predicted
```

```
    response vector
    y_pred = b[0] +
    b[1]*x

    # plotting the
    regression line
    plt.plot(x, y_pred,
    color = "g")

    #
    putting
    labels
    plt.xlab
    el('x')
    plt.ylabel('y')

    # function to
    show plot
    plt.show()

def main():
    # observations
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating
    coefficientsb =
    estimate_coef(x,
    y)
    print("Estimated coefficients:\nb_0 = {} \nb_1 = {}".format(b[0], b[1]))

    # plotting regression
    line
    plot_regression_line
    (x, y, b)

if__name__==
    main ":main()
```

Output:

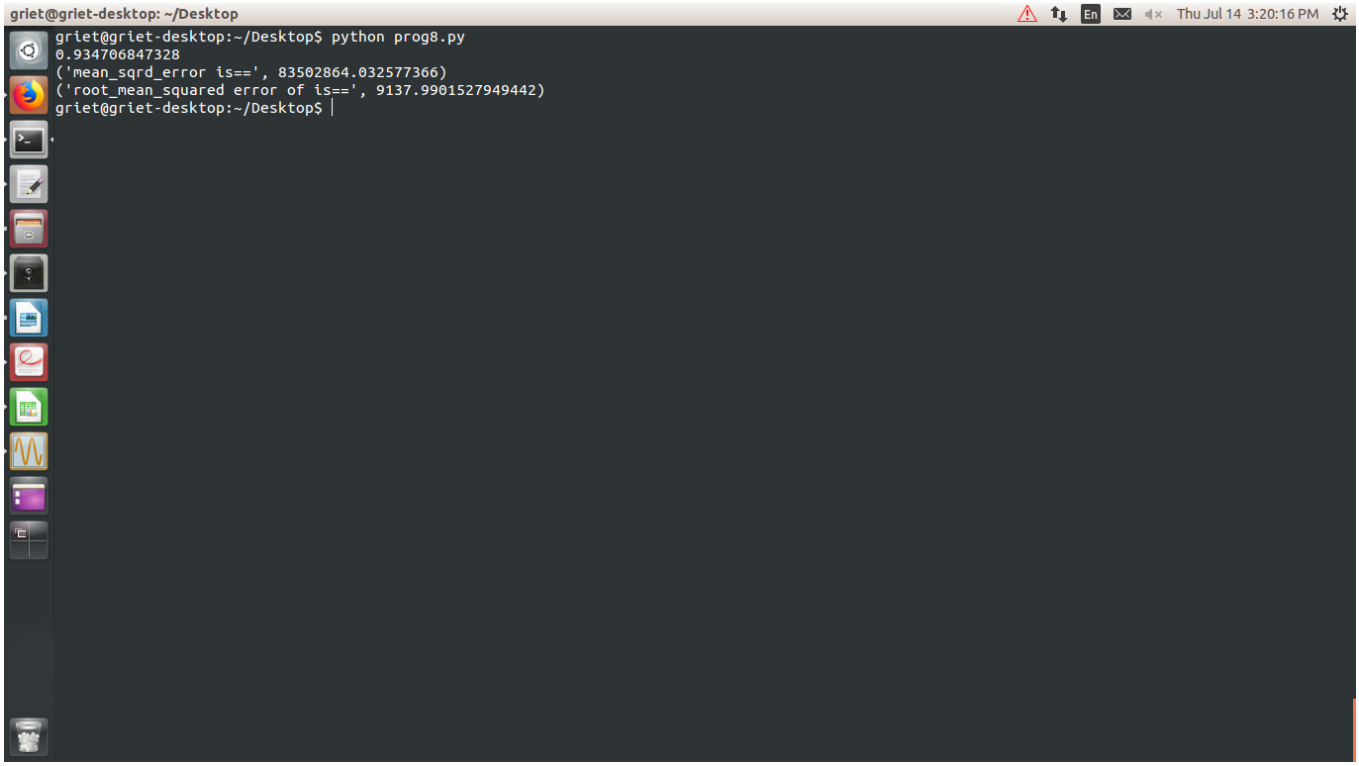

Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\sunil\ML programs
Editor - C:\Users\sunil\ML programs\Task-4_LinearReg.py
DS-task-2.py
DS-Task-3.py
task-3.py
task-4.py
DS-Task-7.py
Task-4_LinearReg.py
-*- coding: utf-8 -*-
"""
Created on Sun Jan 27 13:21:18 2019
@author: sunil
"""
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
# number of observations/points
n = np.size(x)
# mean of x and y vector
m_x, m_y = np.mean(x), np.mean(y)
# calculating cross-deviation and deviation about x
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
# calculating regression coefficients
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
return(b_0, b_1)
def plot_regression_line(x, y, b):
# plotting the actual points as scatter plot
plt.scatter(x, y, color = "m",
marker = "o", s = 30)
# predicted response vector
y_pred = b[0] + b[1]*x
# plotting the regression line
plt.plot(x, y_pred, color = "g")
IPython console
Console 1/A
Predicted feature name: ['setosa']
XNEW
[[6. 2.2 4. 1.]]
Actual : 1 versicolor, Predicted:[2]['virginica']
TEST SCORE[ACCURACY]: 0.97
In [65]: runfile('C:/Users/sunil/ML programs/Task-4_LinearReg.py', wdir='C:/Users/sunil/ML programs')
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
12
10
8
6
4
2
y
0
2
4
6
8
x
In [66]:
IPython console
History log
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 1
Column: 1
Memory: 63 %
Type here to search
ENG
23:25
11/03/2021

b) Multiple Linear Regression Models:

Program:

```
# Multiple Linear Regression
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import cross_validation
# Importing the dataset
dataset = pd.read_csv('//home//griet//Desktop//startups.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 4]
#Convert the column into categorical columns
states=pd.get_dummies(X['State'])
# Drop the state coulmn
X=X.drop('State',axis=1)
# concat the dummy variables
X=pd.concat([X,states],axis=1)
# Splitting the dataset into the Training set and Test set
#from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, y, test_size = 0.2, random_state =0)
# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# Predicting the Test set results
y_pred = regressor.predict(X_test)
from sklearn.metrics import r2_score
score=r2_score(y_test,y_pred)
print(score)
print('mean_sqrd_error is==',mean_squared_error(y_test,y_pred))
print('root_mean_squared error of is==',np.sqrt(mean_squared_error(y_test,y_pred)))
```

Output:


griet@griet-desktop: ~/Desktop
Thu Jul 14 3:20:16 PM
griet@griet-desktop:~/Desktop$ python prog8.py
0.934706847328
('mean_sqrd_error is==', 83502864.032577366)
('root_mean_squared error of is==', 9137.9901527949442)
griet@griet-desktop:~/Desktop$

TASK-6

Task-6: Develop Logistic Regression Model for a given dataset.

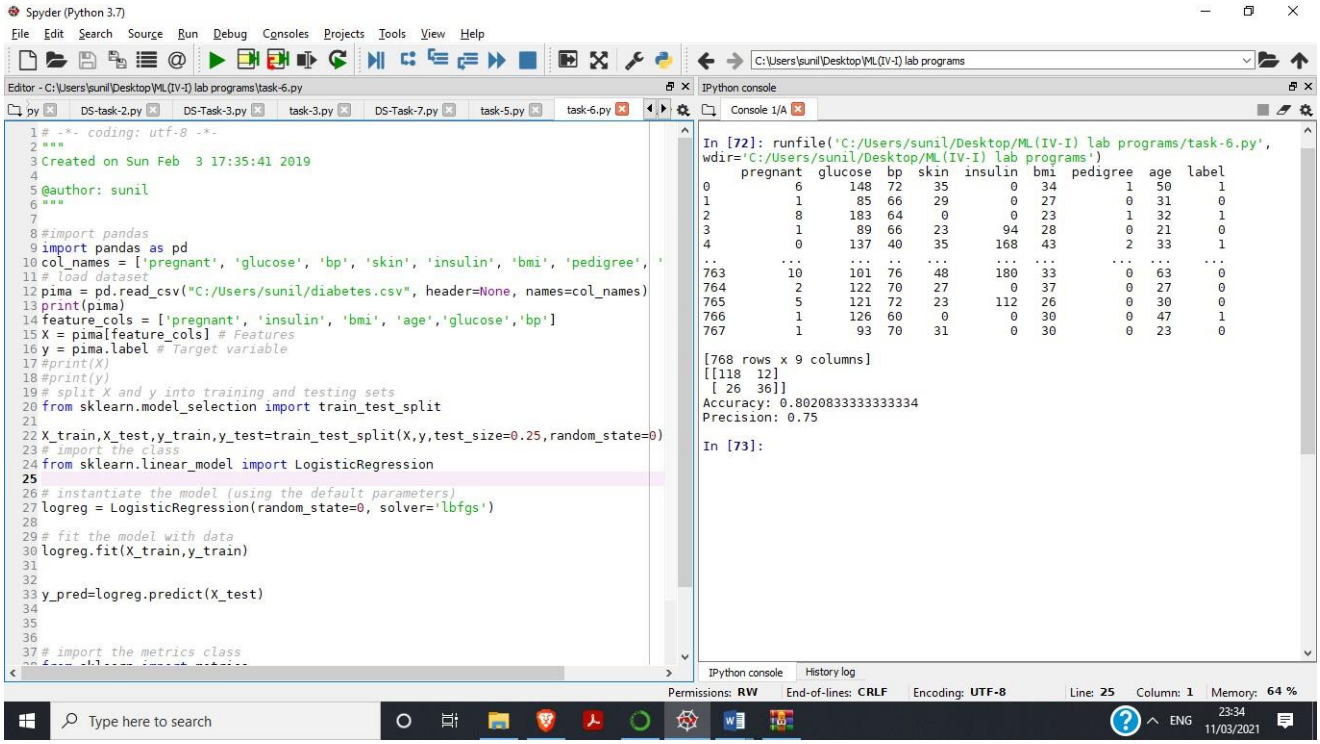
Aim: Develop a python program for Logistic Regression

Program:

```
import pandas as pd

col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age',
'label'] # load dataset
pima = pd.read_csv("C:/Users/swapnika/diabetes.csv", header=None, names=col_names)
print(pima)

feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp']
X = pima[feature_cols] #
Features y = pima.label #
Target variable print(X)
print(y)

# split X and y into training and testing sets

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_s
tate=0) # import the class

from sklearn.linear_model import LogisticRegression


# instantiate the model (using the default parameters)

logreg = LogisticRegression(random_state=0, solver='lbfgs')


# fit the model with data
logreg.fit(X_train,y_train)


y_pred=logreg.predict(X_test)


# import the metrics
class from sklearn
import metrics

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)
```

```
print("Accuracy:",metrics.accuracy_score(y_test,
y_pred))
print("Precision:",metrics.precision_score(y_test,
y_pred))
```

Output:



TASK-7

Task-7: Develop Decision Tree Classification model for a given dataset and use it to classify a new sample.

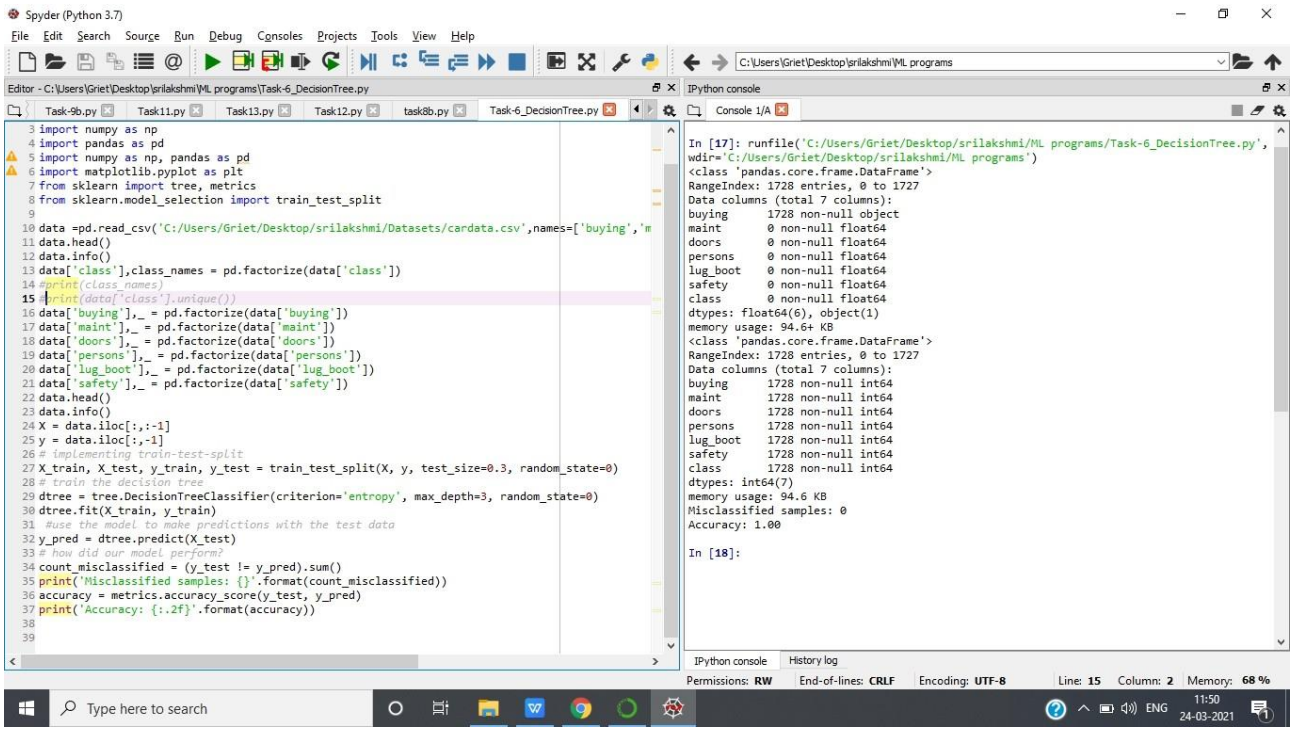
Aim: To Construct Decision Tree for Classification of any data set.

Program:

```
import os

import numpy as np
import pandas as pd
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from sklearn import tree, metrics

from sklearn.model_selection import train_test_split
import pylab

data

=pd.read_csv('C:/Users/sunil/cardata.csv',names=['buying','maint','doors','persons','lug_boot','safety','
class'])
data.head()
data.info()
data['class'],class_names = pd.factorize(data['class'])
print(class_names)

print(data['class'].unique())

data['buying'],_ = pd.factorize(data['buying'])
data['maint'],_ = pd.factorize(data['maint'])
data['doors'],_ = pd.factorize(data['doors'])
data['persons'],_ = pd.factorize(data['persons'])
data['lug_boot'],_ = pd.factorize(data['lug_boot'])
data['safety'],_ = pd.factorize(data['safety'])
data.head()
data.info()

X = data.iloc[:,:-1]

y = data.iloc[:,-1]

# implementing train-test-split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
# train the decision tree
```

```
dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
dtree.fit(X_train, y_train)

 #use the model to make predictions with the test data
y_pred = dtree.predict(X_test)

# how did our model perform?
count_misclassified = (y_test != y_pred).sum()

print('Misclassified samples: {}'.format(count_misclassified))

accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
```

Output:



TASK-8

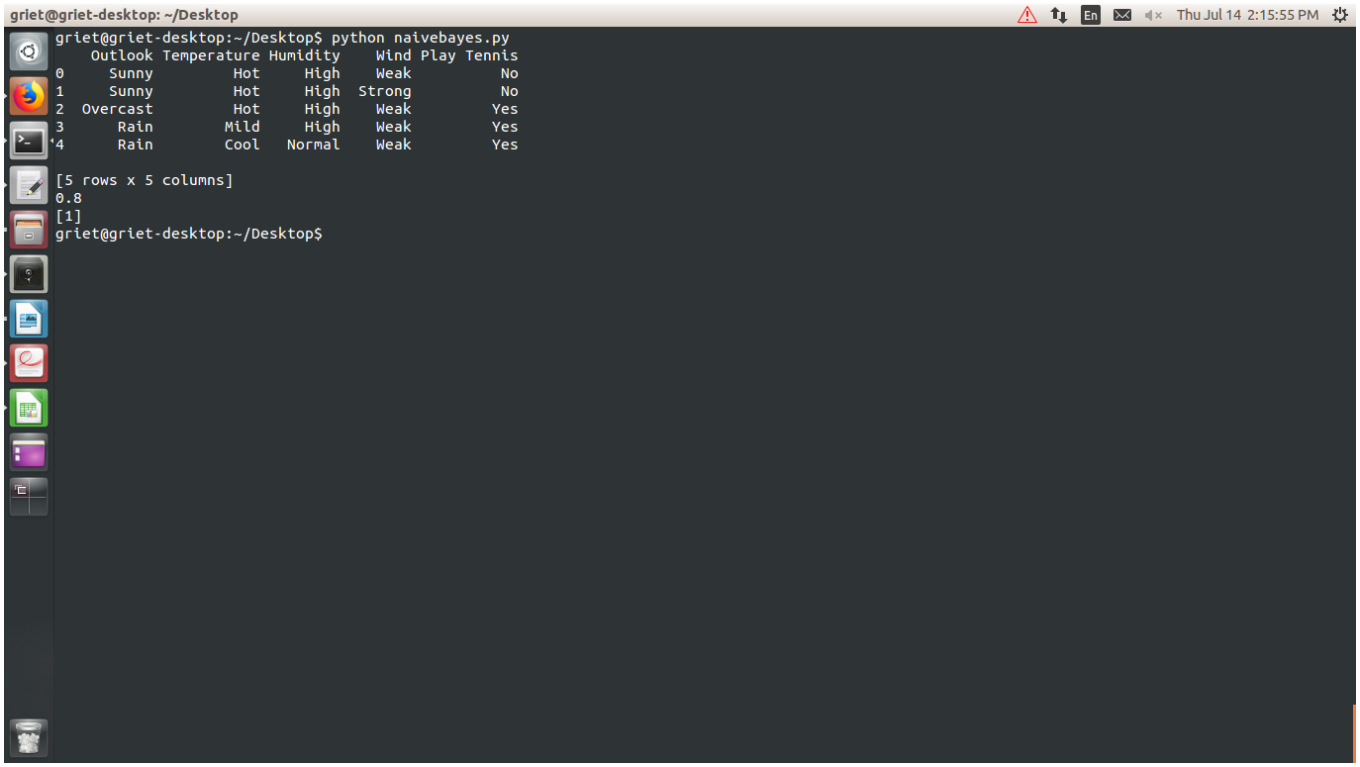
Task-8: Implement Naïve Bayes Classification in Python

Aim: To Implement Naïve Bayes Classification in Python

Program:

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn import cross_validation

play_tennis = pd.read_csv("//home//griet//Downloads//PlayTennis.csv")
print(play_tennis.head())
number = LabelEncoder()
play_tennis['Outlook'] = number.fit_transform(play_tennis['Outlook'])
play_tennis['Temperature'] = number.fit_transform(play_tennis['Temperature'])
play_tennis['Humidity'] = number.fit_transform(play_tennis['Humidity'])
play_tennis['Wind'] = number.fit_transform(play_tennis['Wind'])
play_tennis['Play Tennis'] = number.fit_transform(play_tennis['Play Tennis'])
features = ["Outlook", "Temperature", "Humidity", "Wind"]
target = "Play Tennis"
features_train, features_test, target_train, target_test =
cross_validation.train_test_split(play_tennis[features],play_tennis[target],test_size = 0.33,
        random_state = 54)
model = GaussianNB()
model.fit(features_train, target_train)
pred = model.predict(features_test)
accuracy = accuracy_score(target_test, pred)
print(accuracy)
print model.predict([[1,2,0,1]])
```

Output:


griet@griet-desktop: ~/Desktop
Thu Jul 14 2:15:55 PM
griet@griet-desktop:~/Desktop$ python naivebayes.py
Outlook Temperature Humidity Wind Play Tennis
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 rows x 5 columns]
0.8
[1]
griet@griet-desktop:~/Desktop$

TASK-9

Task-9: Build KNN Classification model for a given dataset.

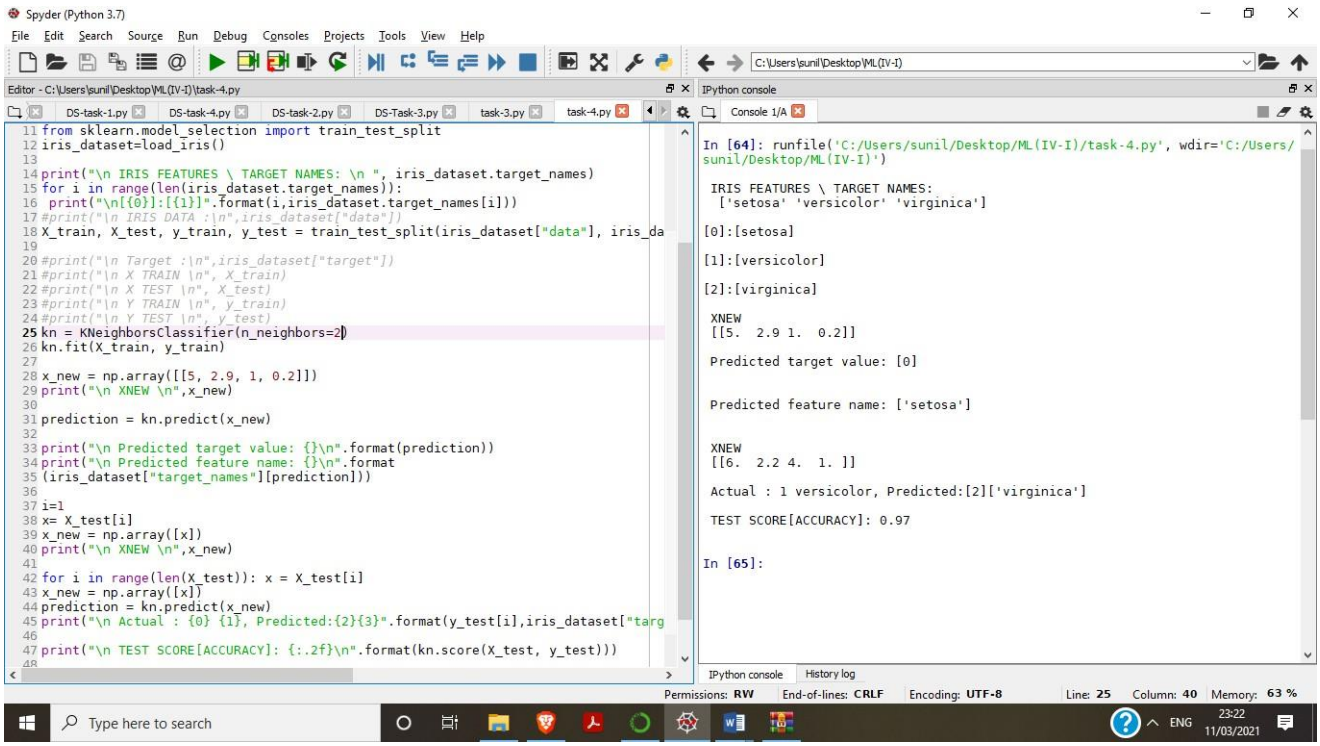
Aim: Develop a python program for KNN algorithm to classify Iris data set.

Program:

```
from sklearn.datasets import load_iris
from sklearn.neighbors import
KNeighborsClassifier import numpy as np
from sklearn.model_selection import train_test_split
iris_dataset=load_iris()

print("\n IRIS FEATURES \ TARGET NAMES: \n ",
iris_dataset.target_names) for i in range(len(iris_dataset.target_names)):
   print("\n[{0}]:[{1}]".format(i,iris_dataset.target_na

mes[i])) print("\n IRIS DATA

:\n",iris_dataset["data"])


X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"],
random_state=0)

print("\n Target
:\n",iris_dataset["target"]) print("\n
X TRAIN \n", X_train)
print("\n X TEST \n",
X_test) print("\n Y
TRAIN \n", y_train)
print("\n Y TEST \n",
y_test)
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)

x_new = np.array([[5, 2.9,
1, 0.2]]) print("\n XNEW
\n",x_new)


prediction = kn.predict(x_new)
```

```
print("\n Predicted target value: {}\n".format(prediction))
print("\n Predicted feature name: {}\n".format
    (iris_dataset["target_names"][prediction]))

i=1
x= X_test[i]
x_new = np.array([x])
print("\n XNEW
\n",x_new)
for i in range(len(X_test)):
    x = X_test[i]
    x_new = np.array([x])
    prediction =
    kn.predict(x_new)
    print("\n Actual : {0} {1}, Predicted
:{2}{3}".format(y_test[i],iris_dataset["target_names"][y_test[i]],prediction,iris_dataset["targ
et_names"][prediction]))

print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X_test, y_test)))
```

Output:-



TASK-10

Task-10: Build Artificial Neural Network model with backup propagation on a given dataset.

Aim: Develop a Python program to build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Program:

```
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) # maximum of X array longitudinallyy
= y/100 #Sigmoid Function
def sigmoid (x): return(1/(1 +
np.exp(- x)))#Derivative of
Sigmoid Function

def derivatives_sigmoid(x):return(x * (1 - x))#Variable
initialization epoch=7000 #Setting training

iterationslr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in
data set hiddenlayer_neurons = 3 #number of
hidden layers

neuronsoutput_neurons = 1 #number of neurons
at output layer #weight and bias initialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_ne
urons)) bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neu
rons)) bout=np.random.uniform(size=(1,output_neurons))

#draws a random range of numbers uniformly
of dim x*yfor i in range(epoch):

#Forward Propogation hinp1=np.dot(X,wh) hinp=hinp1 + bh hlayer_act =
 sigmoid(hinp) outinp1=np.dot(hlayer_act,wout)outinp= outinp1+ bout
```

output
sigmoid(outinp)#Backpropagation
EO = y-output

outgrad = derivatives_sigmoid(output)d_output = EO* outgrad

EH = d_output.dot(wout.T)

hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden
layer wtsd_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror and
currentlayerop # bout += np.sum(d_output, axis=0,keepdims=True)

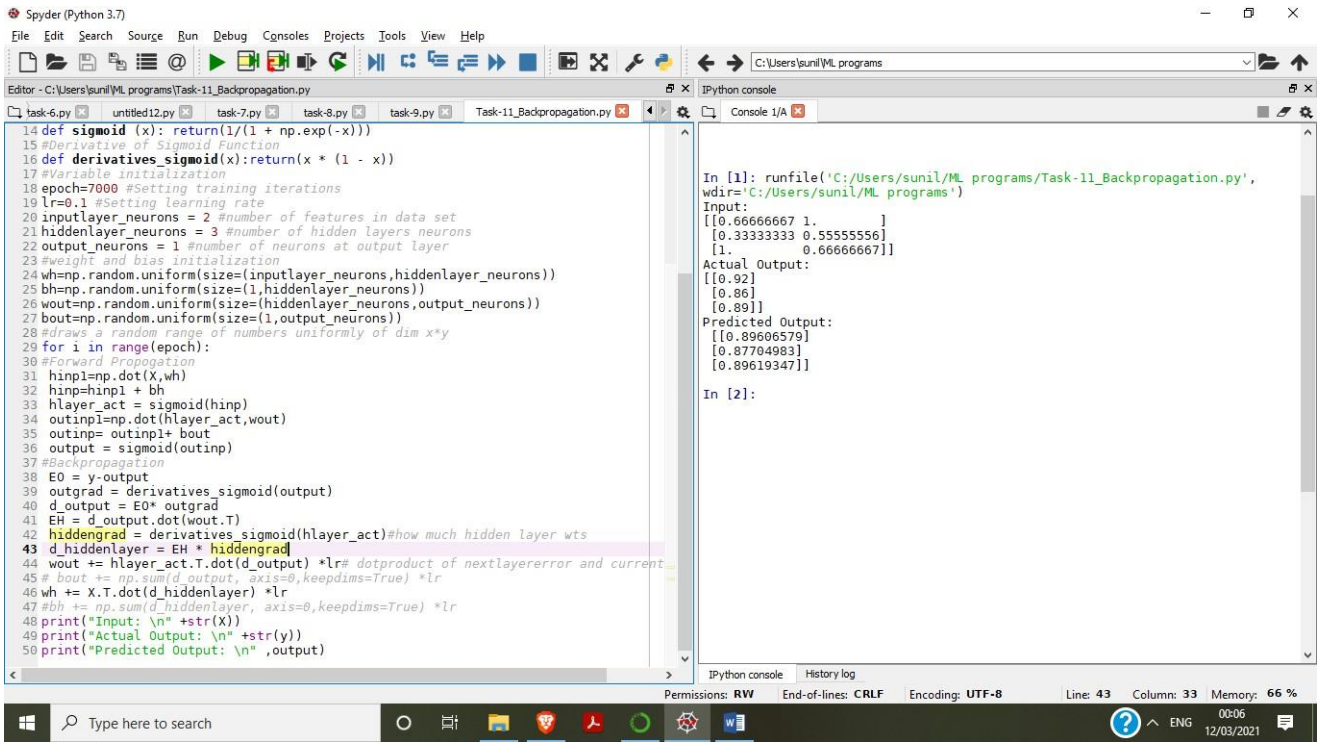
Python program to implement Boosting ensemble method*lrwh += X.T.dot(d_hiddenlayer) *lr

#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True)

*lrprint("Input: \n" + str(X))

print("Actual Output: \n" + str(y)) print("Predicted Output: \n" ,output)

Output:-



TASK-11

Task-11: a) Implement Random Forest ensemble method on a given dataset.

b) Implement Boosting ensemble method on a given dataset.

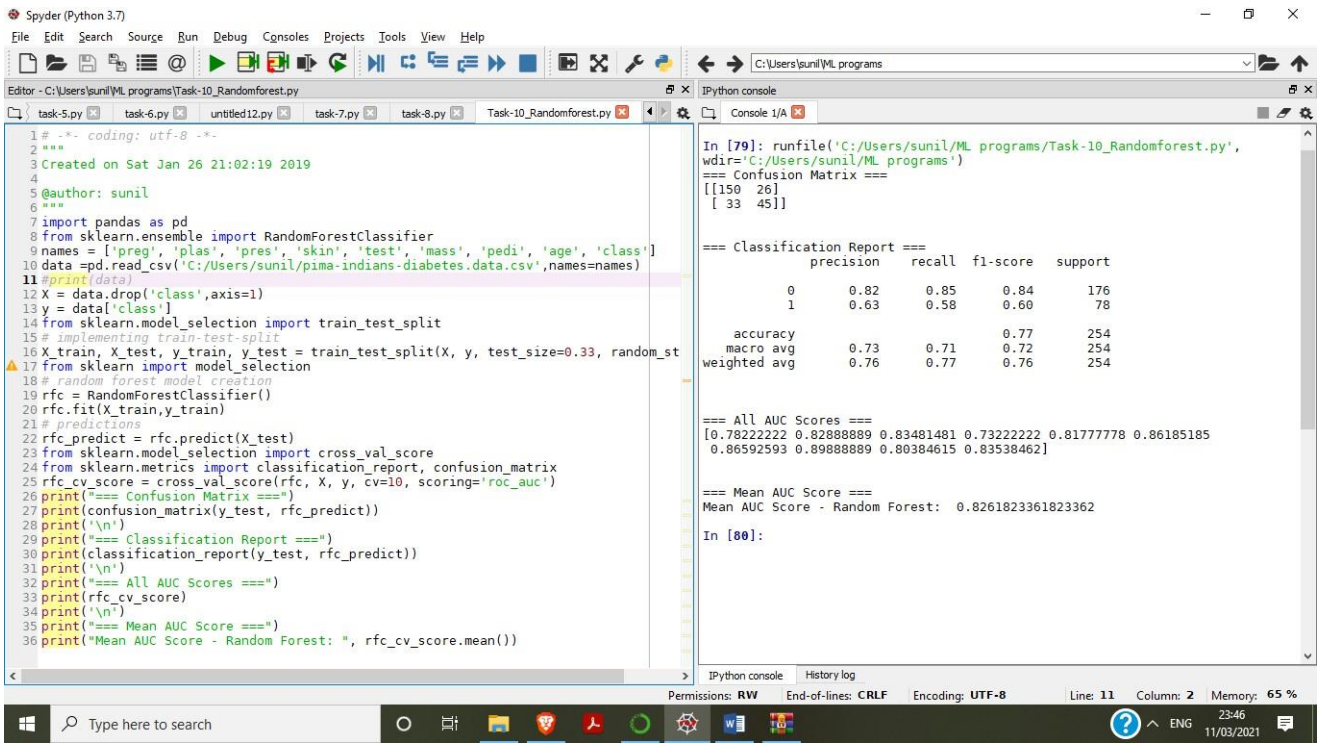
Aim: a) Develop a Python program to implement Random Forest ensemble method

Program:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class'] data =pd.read_csv('C:/Users/sunil/pima-indians-
diabetes.data.csv',names=names)print(data)
X =
data.drop('class',axi
s=1)y = data['class']
from sklearn.model_selection import
train_test_split# implementing train-test-
split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=66)from sklearn import
model_selection
#random forest
modelcreation
rfc=RandomForestCla
ssssifier()
rfc.fit(X_train,y_train)
# predictions
rfc_predict = rfc.predict(X_test)
from sklearn.model_selection import cross_val_score
from sklearn.metrics import
classification_report,
confusion_matrixrfc_cv_score = cross_val_score(rfc, X, y,
cv=10, scoring='roc_auc')print("=== Confusion Matrix
===") print(confusion_matrix(y_test,
rfc_predict))print('\n')
```

```
print("=== Classification
Report ===")
print(classification_report(y_tes
t, rfc_predict))print('\n')
print("=== All AUC Scores
===")print(rfc_cv
_score) print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

Output:



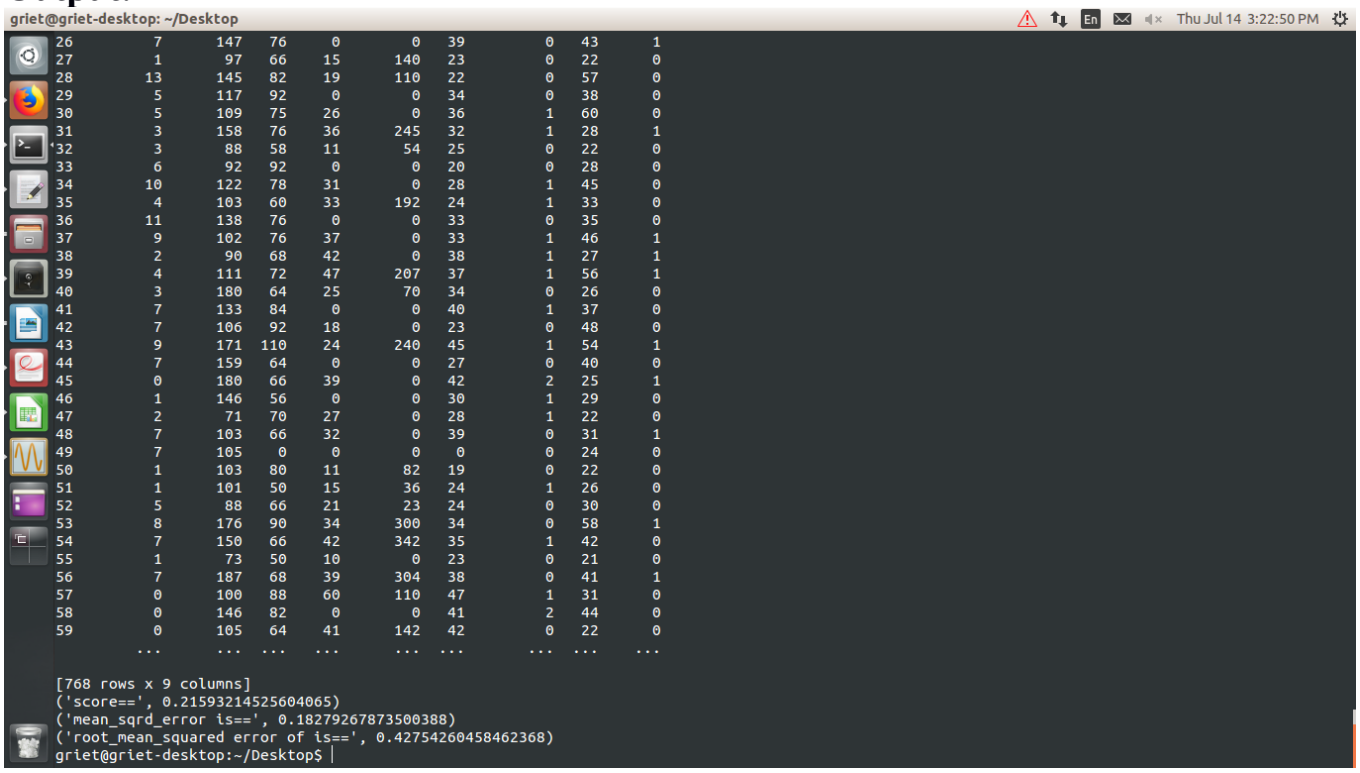
b) Implement Boosting ensemble method on a given dataset

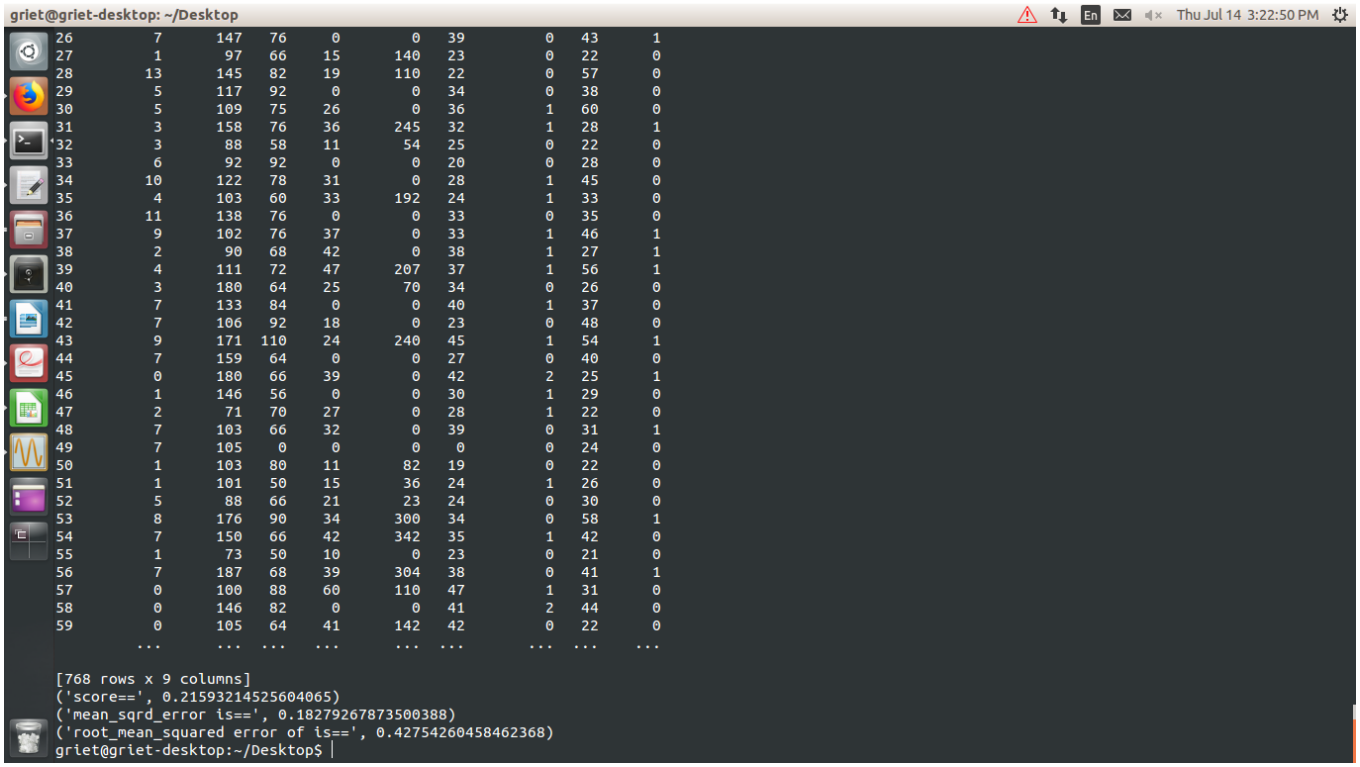
Aim: Develop a Python program to implement Boosting ensemble method

Program:

```
# importing utility modules
import pandas as pd
import numpy as np
from sklearn import cross_validation
from sklearn.metrics import mean_squared_error


# importing machine learning models for prediction
from sklearn.ensemble import GradientBoostingRegressor
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("//home//griet//Desktop//diabetes.csv", header=None,
names=col_names)
print(pima)
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp']
X = pima[feature_cols] # Features
y = pima.label # Target variable

# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, y, test_size=0.20)
# initializing the boosting module with default parameters
model = GradientBoostingRegressor()

# training the model on the train dataset
model.fit(X_train, y_train)

# predicting the output on the test dataset
y_pred = model.predict(X_test)

# printing the root mean squared error between real value and predicted value

from sklearn.metrics import r2_score

print('score==',r2_score(y_test,y_pred))
print('mean_sqrd_error is==',mean_squared_error(y_test,y_pred))
print('root_mean_squared error of is==',np.sqrt(mean_squared_error(y_test,y_pred)))
```

Output:


griet@griet-desktop: ~/Desktop
Thu Jul 14 3:22:50 PM
26 7 147 76 0 0 39 0 43 1
27 1 97 66 15 140 23 0 22 0
28 13 145 82 19 110 22 0 57 0
29 5 117 92 0 0 34 0 38 0
30 5 109 75 26 0 36 1 60 0
31 3 158 76 36 245 32 1 28 1
32 3 88 58 11 54 25 0 22 0
33 6 92 92 0 0 20 0 28 0
34 10 122 78 31 0 28 1 45 0
35 4 103 60 33 192 24 1 33 0
36 11 138 76 0 0 33 0 35 0
37 9 102 76 37 0 33 1 46 1
38 2 90 68 42 0 38 1 27 1
39 4 111 72 47 207 37 1 56 1
40 3 180 64 25 70 34 0 26 0
41 7 133 84 0 0 40 1 37 0
42 7 106 92 18 0 23 0 48 0
43 9 171 110 24 240 45 1 54 1
44 7 159 64 0 0 27 0 40 0
45 0 180 66 39 0 42 2 25 1
46 1 146 56 0 0 30 1 29 0
47 2 71 70 27 0 28 1 22 0
48 7 103 66 32 0 39 0 31 1
49 7 105 0 0 0 0 0 24 0
50 1 103 80 11 82 19 0 22 0
51 1 101 50 15 36 24 1 26 0
52 5 88 66 21 23 24 0 30 0
53 8 176 90 34 300 34 0 58 1
54 7 150 66 42 342 35 1 42 0
55 1 73 50 10 0 23 0 21 0
56 7 187 68 39 304 38 0 41 1
57 0 100 88 60 110 47 1 31 0
58 0 146 82 0 0 41 2 44 0
59 0 105 64 41 142 42 0 22 0
... ...
[768 rows x 9 columns]
('score==', 0.21593214525604065)
('mean_sqrd_error is==', 0.18279267873500388)
('root_mean_squared error of is==', 0.42754260458462368)
griet@griet-desktop:~/Desktop$


griet@griet-desktop: ~/Desktop
Thu Jul 14 3:22:50 PM
26 7 147 76 0 0 39 0 43 1
27 1 97 66 15 140 23 0 22 0
28 13 145 82 19 110 22 0 57 0
29 5 117 92 0 0 34 0 38 0
30 5 109 75 26 0 36 1 60 0
31 3 158 76 36 245 32 1 28 1
32 3 88 58 11 54 25 0 22 0
33 6 92 92 0 0 20 0 28 0
34 10 122 78 31 0 28 1 45 0
35 4 103 60 33 192 24 1 33 0
36 11 138 76 0 0 33 0 35 0
37 9 102 76 37 0 33 1 46 1
38 2 90 68 42 0 38 1 27 1
39 4 111 72 47 207 37 1 56 1
40 3 180 64 25 70 34 0 26 0
41 7 133 84 0 0 40 1 37 0
42 7 106 92 18 0 23 0 48 0
43 9 171 110 24 240 45 1 54 1
44 7 159 64 0 0 27 0 40 0
45 0 180 66 39 0 42 2 25 1
46 1 146 56 0 0 30 1 29 0
47 2 71 70 27 0 28 1 22 0
48 7 103 66 32 0 39 0 31 1
49 7 105 0 0 0 0 0 24 0
50 1 103 80 11 82 19 0 22 0
51 1 101 50 15 36 24 1 26 0
52 5 88 66 21 23 24 0 30 0
53 8 176 90 34 300 34 0 58 1
54 7 150 66 42 342 35 1 42 0
55 1 73 50 10 0 23 0 21 0
56 7 187 68 39 304 38 0 41 1
57 0 100 88 60 110 47 1 31 0
58 0 146 82 0 0 41 2 44 0
59 0 105 64 41 142 42 0 22 0
... ...
[768 rows x 9 columns]
('score==', 0.21593214525604065)
('mean_sqrd_error is==', 0.18279267873500388)
('root_mean_squared error of is==', 0.42754260458462368)
griet@griet-desktop:~/Desktop$

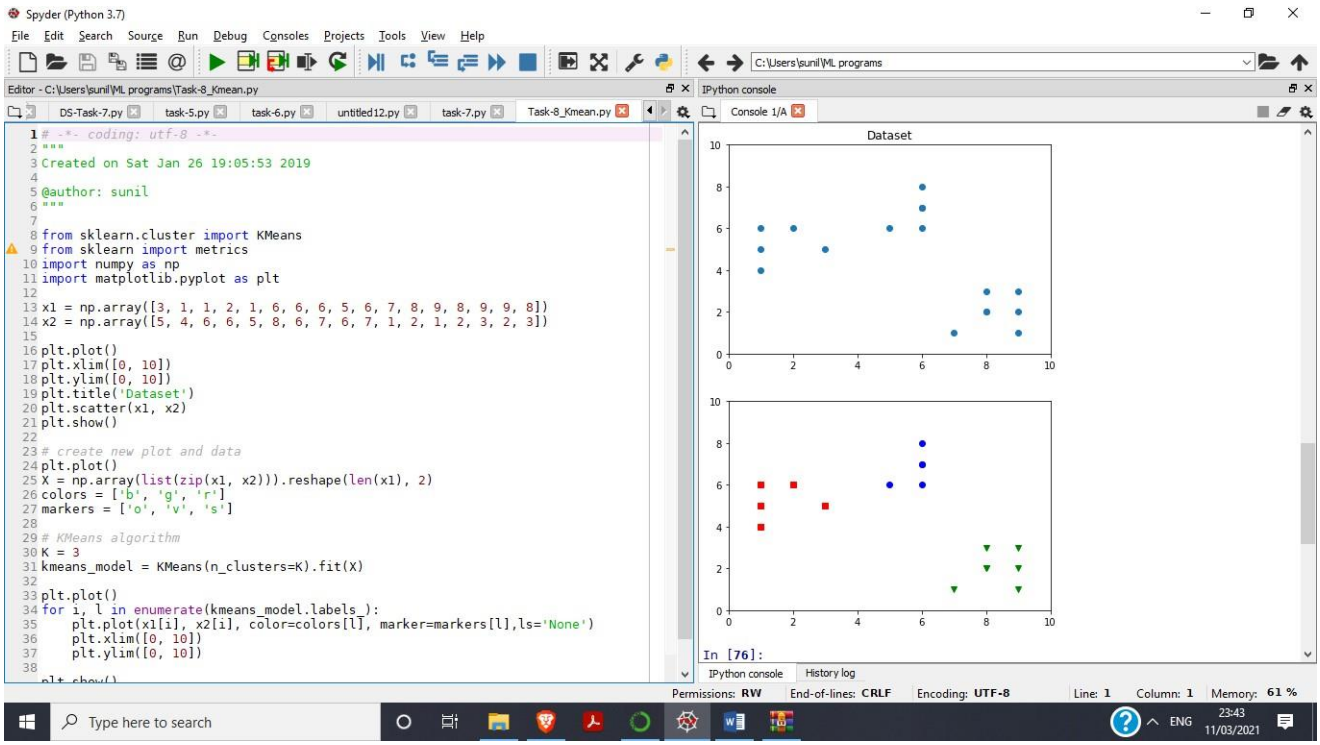
TASK-12

Task-12: Write a python program to implement K-Means Clustering Algorithm

Aim: Develop a Python program to implement K-means Clustering Algorithm

Program:

```
from sklearn.cluster import KMeans
from sklearn import metrics

import numpy as np
import matplotlib.pyplot as plt


x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8,  9, 9, 8])

x2 = np.array([5, 4, 6, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2,  3])


plt.plot()
plt.xlim([0, 10])

plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()


# create new plot and data
plt.plot()

X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']

markers = ['o', 'v', 's']


# KMeans algorithmK

= 3

kmeans_model = KMeans(n_clusters=K).fit(X)


plt.plot()

for i, l in enumerate(kmeans_model.labels_):
    plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l],ls='None')
    plt.xlim([0, 10])
    plt.ylim([0, 10])


plt.show()
```


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\sunil\ML programs
Editor - C:\Users\sunil\ML programs\Task-8_Kmean.py
IPython console
DS-Task-7.py
task-5.py
task-6.py
untitled12.py
task-7.py
Task-8_Kmean.py
Console 1/A
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sat Jan 26 19:05:53 2019
4
5 @author: sunil
6 """
7
8 from sklearn.cluster import KMeans
9 from sklearn import metrics
10 import numpy as np
11 import matplotlib.pyplot as plt
12
13 x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
14 x2 = np.array([5, 4, 6, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])
15
16 plt.plot()
17 plt.xlim([0, 10])
18 plt.ylim([0, 10])
19 plt.title('Dataset')
20 plt.scatter(x1, x2)
21 plt.show()
22
23 # create new plot and data
24 plt.plot()
25 X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
26 colors = ['b', 'g', 'r']
27 markers = ['o', 'v', 's']
28
29 # KMeans algorithm
30 K = 3
31 kmeans_model = KMeans(n_clusters=K).fit(X)
32
33 plt.plot()
34 for i, l in enumerate(kmeans_model.labels_):
35 plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l],ls='None')
36 plt.xlim([0, 10])
37 plt.ylim([0, 10])
38
Dataset
In [76]:
IPython console
History log
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 1
Column: 1
Memory: 61 %
Type here to search
ENG
23:43
11/03/2021