



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

S.No	Tasks	Page No.
1	Write a python program to import and export data using Pandas library functions	1
2	Demonstrate various data pre-processing techniques for a given dataset.	3
3	Implement Dimensionality reduction using Principle Component Analysis (PCA) method.	5
4	Write a Python program to demonstrate various Data Visualization Techniques.	7
5	Implement Simple and Multiple Linear Regression Models.	11
6	Develop Logistic Regression Model for a given dataset.	14
7	Develop Decision Tree Classification model for a given dataset and use it to classify a new sample.	16
8	Implement Naïve Bayes Classification in Python	18
9	Build KNN Classification model for a given dataset.	20
10	Build Artificial Neural Network model with back propagation on a given dataset	22
11	a) Implement Random Forest ensemble method on a given dataset. b) Implement Boosting ensemble method on a given dataset.	24
12	Write a python program to implement K-Means clustering Algorithm	26

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)

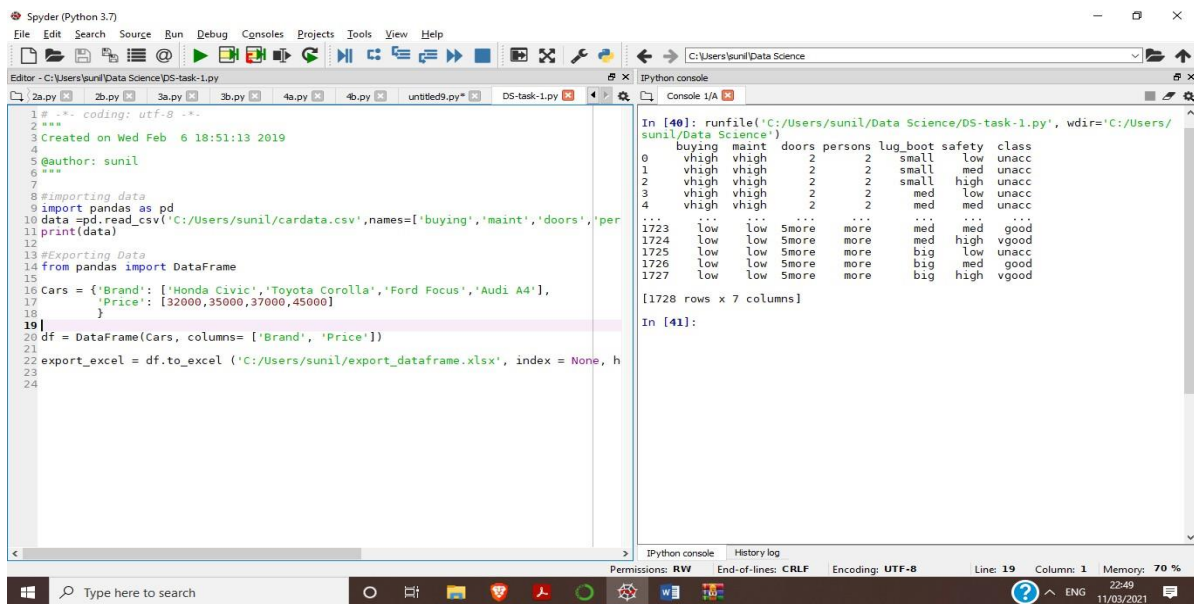
#Exporting Data
from pandas import DataFrame

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:



The screenshot shows the Spyder Python IDE interface. The editor on the left contains a Python script that reads a CSV file, creates a DataFrame, and exports it to an Excel file. The IPython console on the right displays the output of the script, showing the first few rows of the DataFrame and the dimensions of the exported Excel file.

```
1# -*- coding: utf-8 -*-
2'''
3Created on Wed Feb 6 18:51:13 2019
4
5@author: sunil
6'''
7
8#Importing data
9import pandas as pd
10data = pd.read_csv('C:/Users/sunil/cardata.csv', names=['buying', 'maint', 'doors', 'per
11print(data)
12
13#Exporting Data
14from pandas import DataFrame
15
16Cars = {'Brand': ['Honda Civic', 'Toyota Corolla', 'Ford Focus', 'Audi A4'],
17        'Price': [32000, 35000, 37000, 45000]}
18
19|
20df = DataFrame(Cars, columns= ['Brand', 'Price'])
21
22export_excel = df.to_excel('C:/Users/sunil/export_dataframe.xlsx', index = None, h
23
24
```

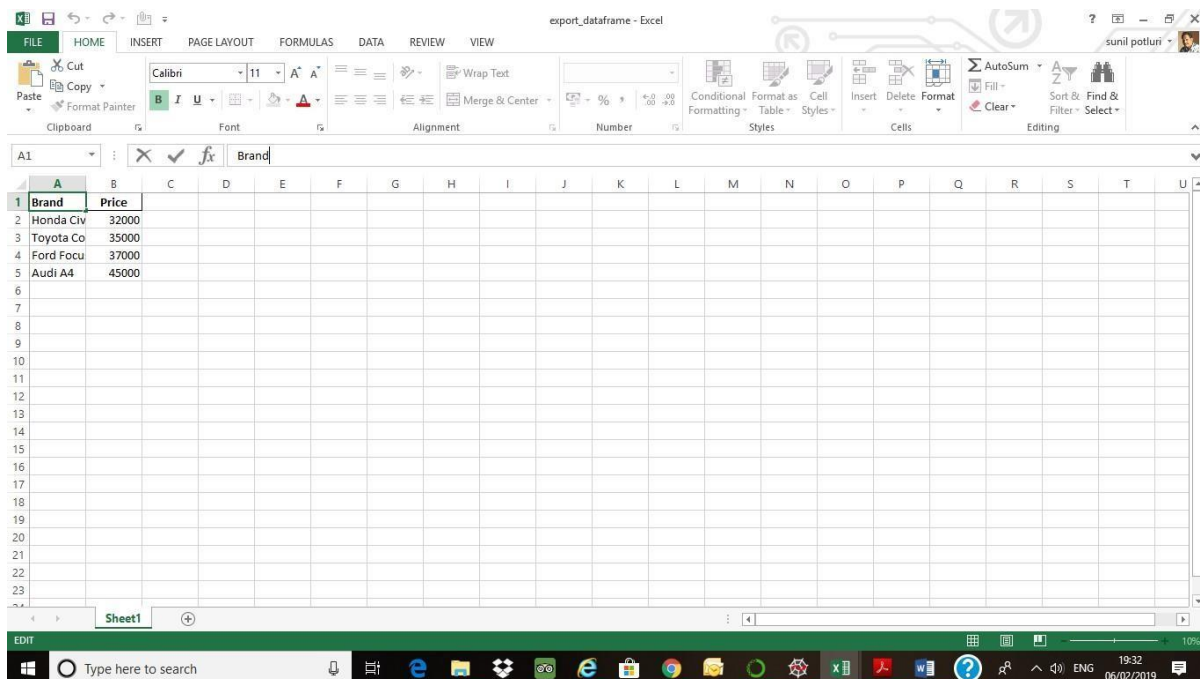
In [40]: runfile('C:/Users/sunil/Data Science/DS-task-1.py', wdir='C:/Users/sunil/Data Science')

	buying	maint	doors	persons	lug_boot	safety	class
0	vhhigh	vhhigh	2	2	small	low	unacc
1	vhhigh	vhhigh	2	2	small	med	unacc
2	vhhigh	vhhigh	2	2	small	high	unacc
3	vhhigh	vhhigh	2	2	med	low	unacc
4	vhhigh	vhhigh	2	2	med	med	unacc
...
1723	low	low	Smore	more	med	med	good
1724	low	low	Smore	more	med	high	vgood
1725	low	low	Smore	more	big	low	unacc
1726	low	low	Smore	more	big	med	good
1727	low	low	Smore	more	big	high	vgood

[1728 rows x 7 columns]

In [41]:

After Export:



The screenshot shows the Microsoft Excel application with the file 'export_dataframe - Excel' open. The data is displayed in a single sheet named 'Sheet1'. The first two columns are 'Brand' and 'Price', with four rows of data.

	Brand	Price
1	Honda Civ	32000
2	Toyota Co	35000
3	Ford Focu	37000
4	Audi A4	45000

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']

# preparing of dataframe using the data at given link and defined
columnslst

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:

```
griet@griet-desktop: ~/Desktop
griet@griet-desktop:~/Desktop$ python prog2.py
/usr/lib/python2.7/dist-packages/sklearn/utils/validation.py:278: UserWarning: StandardScaler assumes floating point values as input, got int64
"got %s" % (estimator, X.dtype))
[[ 0.64  0.848  0.15  0.907 -0.693  0.248  1.228  1.426]
 [-0.845 -1.123 -0.161  0.531 -0.693 -0.64 -0.733 -0.191]
 [ 1.234  1.944 -0.264 -1.288 -0.693 -1.148  1.228 -0.106]
 [-0.845 -0.998 -0.161  0.155  0.123 -0.513 -0.733 -1.042]
 [-1.142  0.504 -1.505  0.907  0.766  1.389  3.189 -0.02 ]]
```

b) Binarizer

```
# import libraries
```

```
from sklearn.preprocessing import Binarizer
```

```
import pandas
```

```
import numpy
```

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



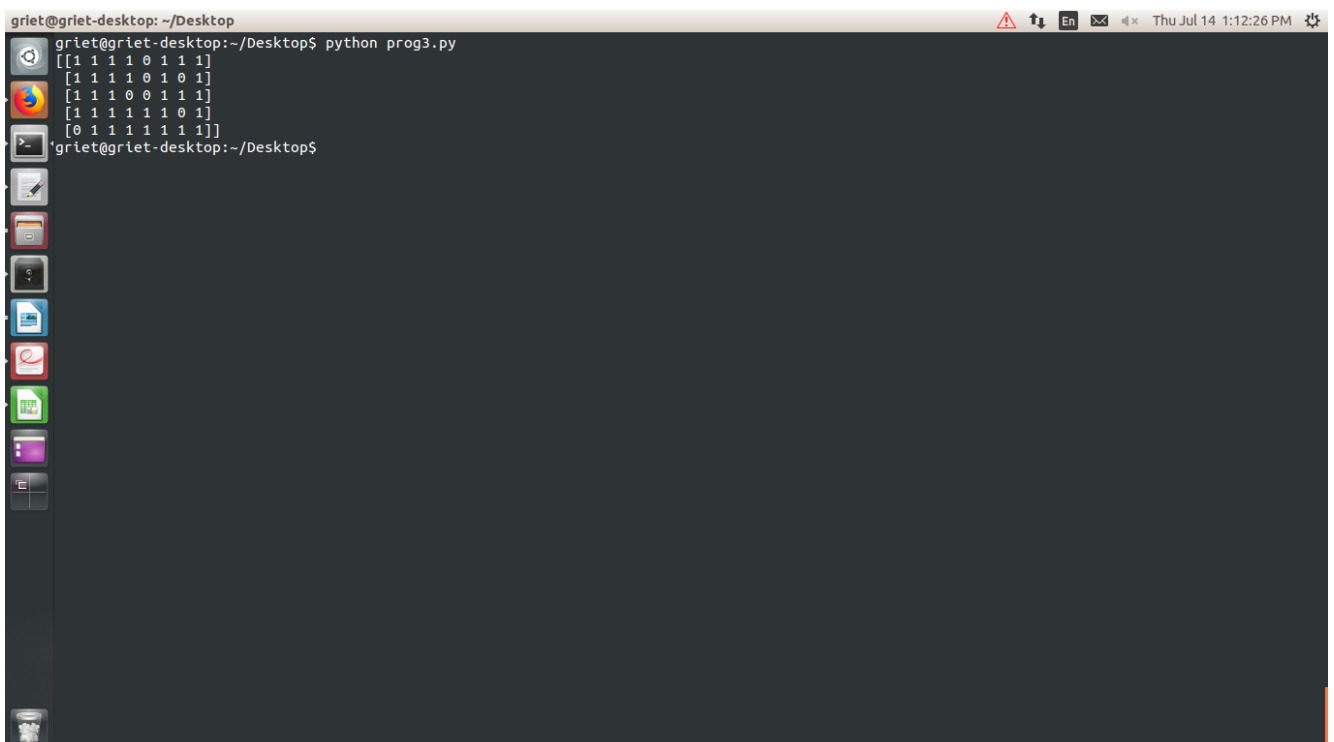
```
# preparing of dataframe using the data at given link and defined columns list
dataframe = pandas.read_csv('/home//griet/Desktop//ML//diabetes.csv', names =
names)
array = dataframe.values
# separate array into input and output components
X = array[:, 0:8]
Y = array[:, 8]
binarizer = Binarizer(threshold = 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
```

A terminal window titled 'griet@griet-desktop: ~/Desktop' shows the execution of a Python script 'prog3.py'. The script's output is displayed as a 5x7 matrix of binary values (0s and 1s). The terminal interface includes a dark background, a light-colored title bar, and a vertical dock on the left with various application icons. The system tray at the bottom right shows the date and time as 'Thu Jul 14 1:12:26 PM' along with standard window control icons.

```
griet@griet-desktop: ~/Desktop
griet@griet-desktop:~/Desktop$ python prog3.py
[[1 1 1 1 0 1 1]
 [1 1 1 1 0 1 0]
 [1 1 1 0 0 1 1]
 [1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1]]
griet@griet-desktop:~/Desktop$
```

```

import scipy

import numpy

from sklearn.preprocessing import MinMaxScaler

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

# preparing 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
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]]
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:

The image shows the Spyder Python IDE interface. The editor on the left contains a Python script for PCA. The IPython console on the right shows the output of the script. The Windows taskbar is visible at the bottom.

```
1 #-*- coding: utf-8 -*-
2 """
3 Created on Sat Jan 26 19:12:43 2019
4
5 @author: sunil
6 """
7
8 from numpy import array
9 from numpy import mean
10 from numpy import cov
11 from numpy.linalg import eig
12 # define a matrix
13 A = array([[1, 2], [3, 4], [5, 6]])
14 print(A)
15 # calculate the mean of each column
16 M = mean(A.T, axis=1)
17 print(M)
18 # center columns by subtracting column means
19 C = A - M
20 print(C)
21 # calculate covariance matrix of centered matrix
22 V = cov(C.T)
23 print(V)
24 # eigendecomposition of covariance matrix
25 values, vectors = eig(V)
26 print(vectors)
27 print(values)
28 # project data
29 P = vectors.T.dot(C.T)
30 print(P.T)
```

```
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.]
 [ 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 %
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['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

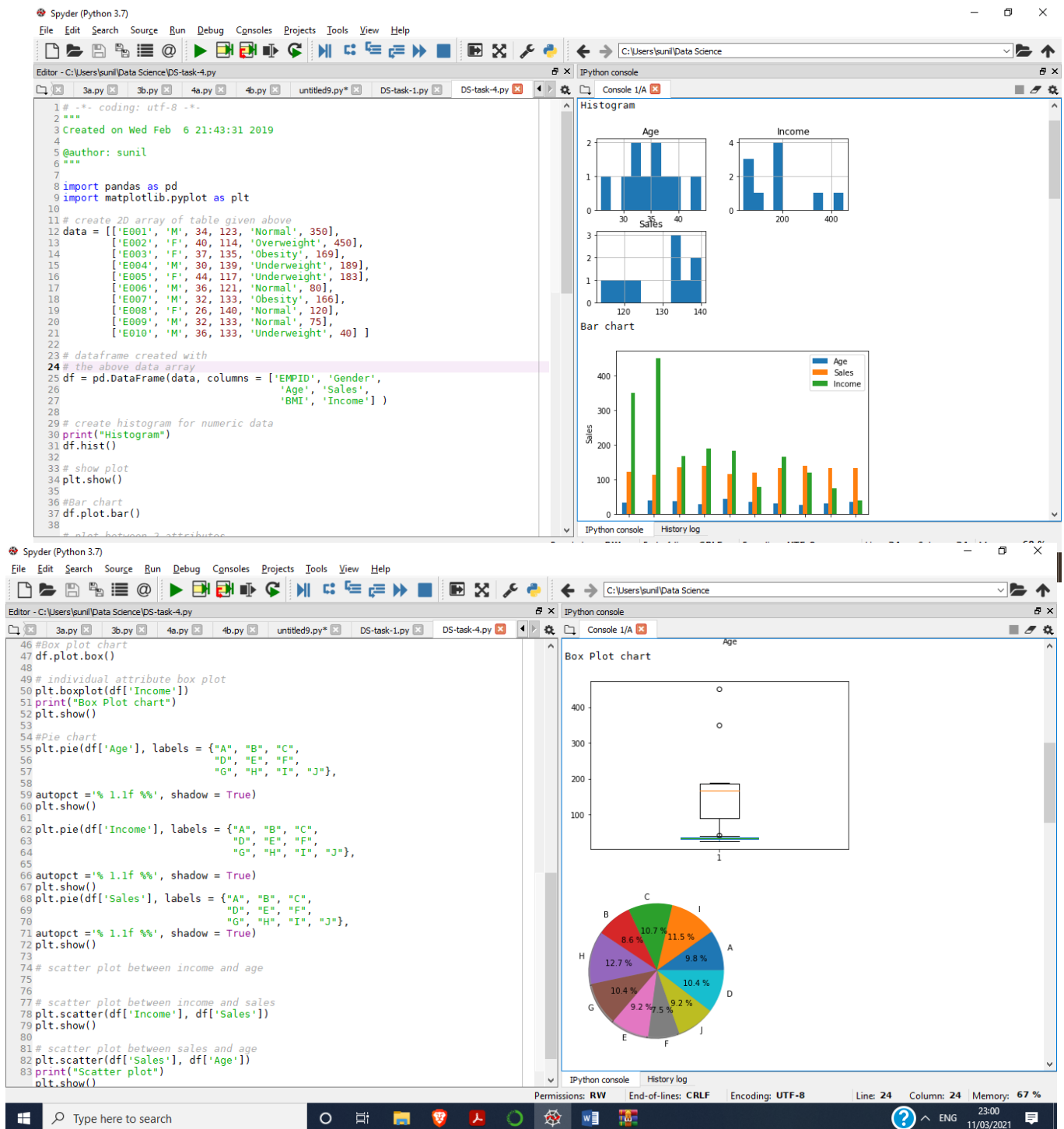
# scatter plot between income
and sales
plt.scatter(df['Income'],
df['Sales']) plt.show()

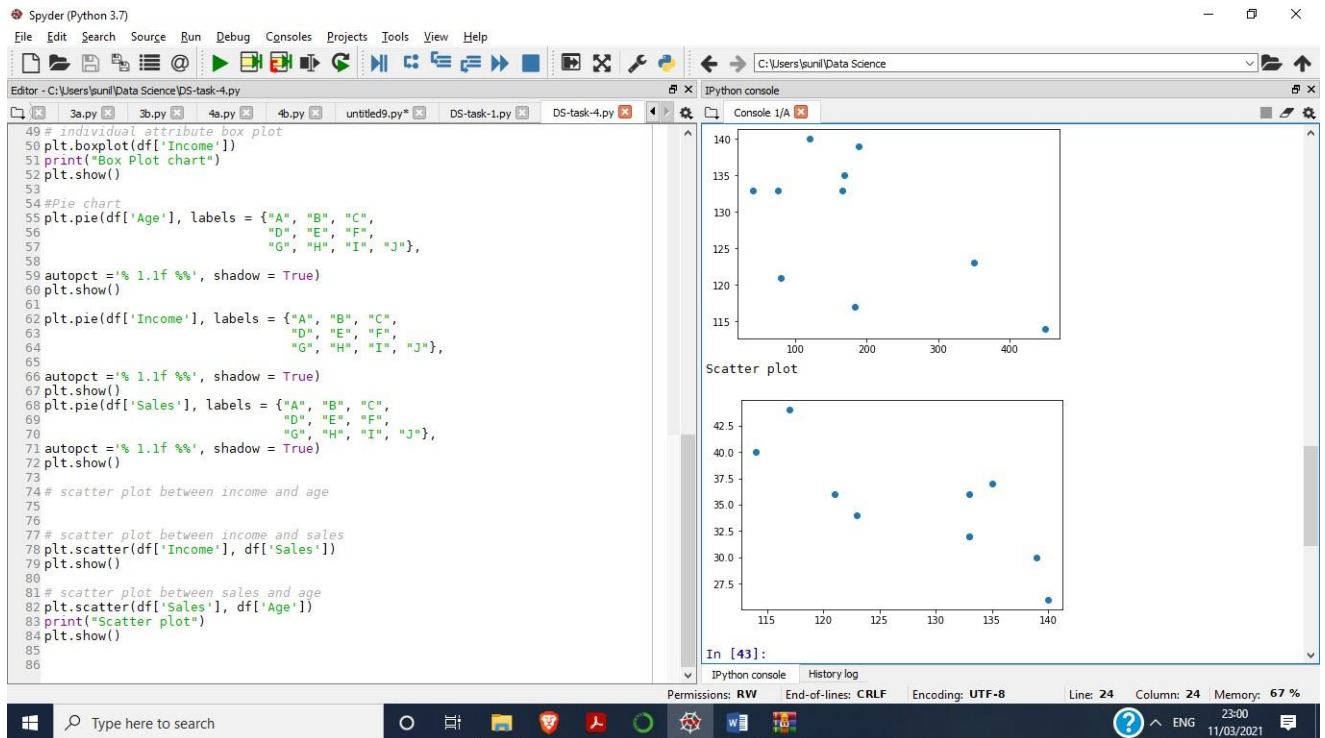
```

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

```
plt.show()
```

Output:





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/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 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.xlabel
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])

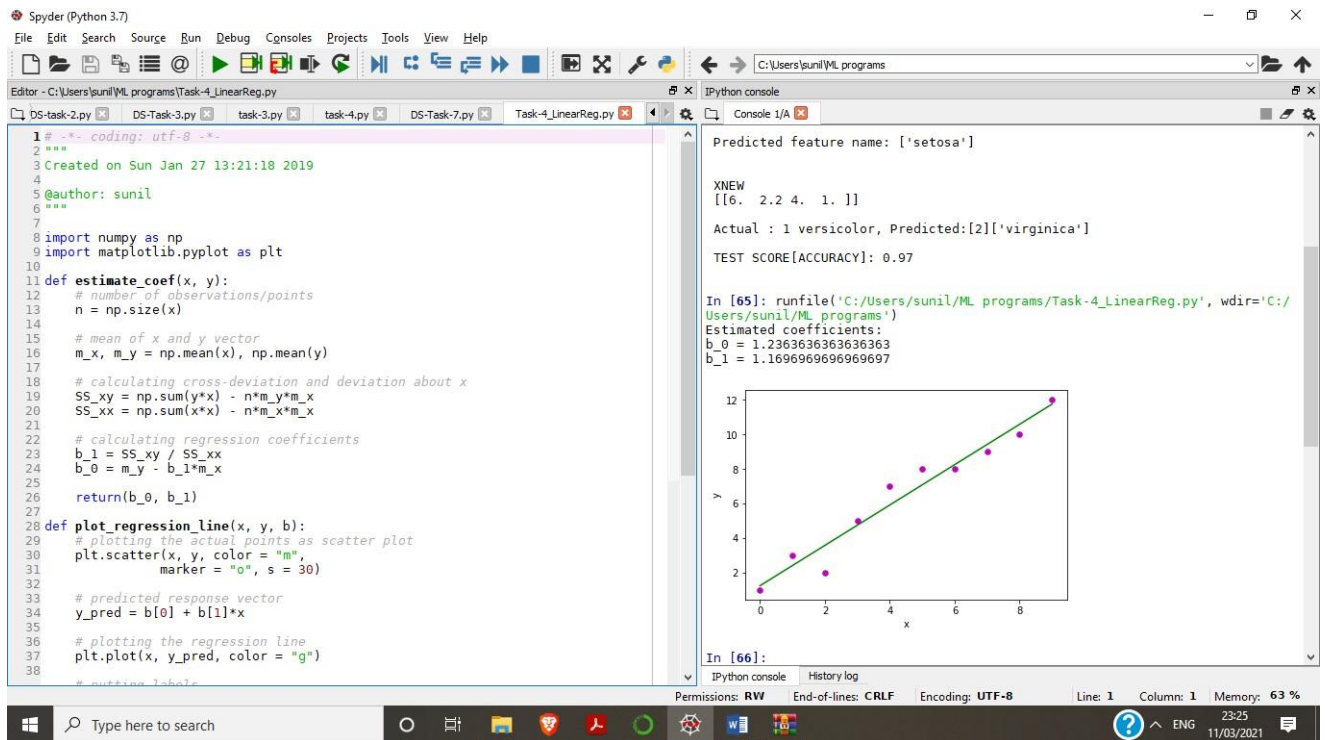
    # estimating
    coefficientsb =
    estimate_coef(x,
y)
    print("Estimated coefficients:\nb_0 = {} \nb_1 = {}".format(b[0], b[1]))

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

if __name__ == '__main__':
    main()

```

Output:



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
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:

The screenshot shows the Spyder Python IDE interface. The editor on the left contains a Python script for loading and analyzing the Pima Indians Diabetes dataset. The IPython console on the right displays the output of the script, including a preview of the dataset, its dimensions, and the accuracy and precision of a Logistic Regression model.

```
1 #-*- coding: utf-8 -*-
2 """
3 Created on Sun Feb 3 17:35:41 2019
4
5 @author: sunil
6 """
7
8 #import pandas
9 import pandas as pd
10 col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
11 # load dataset
12 pima = pd.read_csv("C:/Users/sunil/diabetes.csv", header=None, names=col_names)
13 print(pima)
14 feature_cols = ['pregnant', 'insulin', 'bmi', 'age', 'glucose', 'bp']
15 X = pima[feature_cols] # Features
16 y = pima.label # Target variable
17 #print(X)
18 #print(y)
19 # split X and y into training and testing sets
20 from sklearn.model_selection import train_test_split
21
22 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
23 # import the class
24 from sklearn.linear_model import LogisticRegression
25
26 # instantiate the model (using the default parameters)
27 logreg = LogisticRegression(random_state=0, solver='lbfgs')
28
29 # fit the model with data
30 logreg.fit(X_train,y_train)
31
32
33 y_pred=logreg.predict(X_test)
34
35
36
37 # import the metrics class
38 from sklearn import metrics
```

IPython console output:

```
In [72]: runfile('C:/Users/sunil/Desktop/ML(IV-I) lab programs/task-6.py',
wdir='C:/Users/sunil/Desktop/ML(IV-I) lab programs')
pregnant glucose bp skin insulin bmi pedigree age label
0 6 148 72 35 0 34 1 50 1
1 1 85 66 29 0 27 0 31 0
2 8 183 64 0 0 23 1 32 1
3 1 89 66 23 94 28 0 21 0
4 0 137 40 35 168 43 2 33 1
... ..
763 10 101 76 48 180 33 0 63 0
764 2 122 70 27 0 37 0 27 0
765 5 121 72 23 112 26 0 30 0
766 1 126 60 0 0 30 0 47 1
767 1 93 70 31 0 30 0 23 0

[768 rows x 9 columns]
[[118 12]
 [ 26 36]]
Accuracy: 0.8020833333333334
Precision: 0.75

In [73]:
```


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?
```

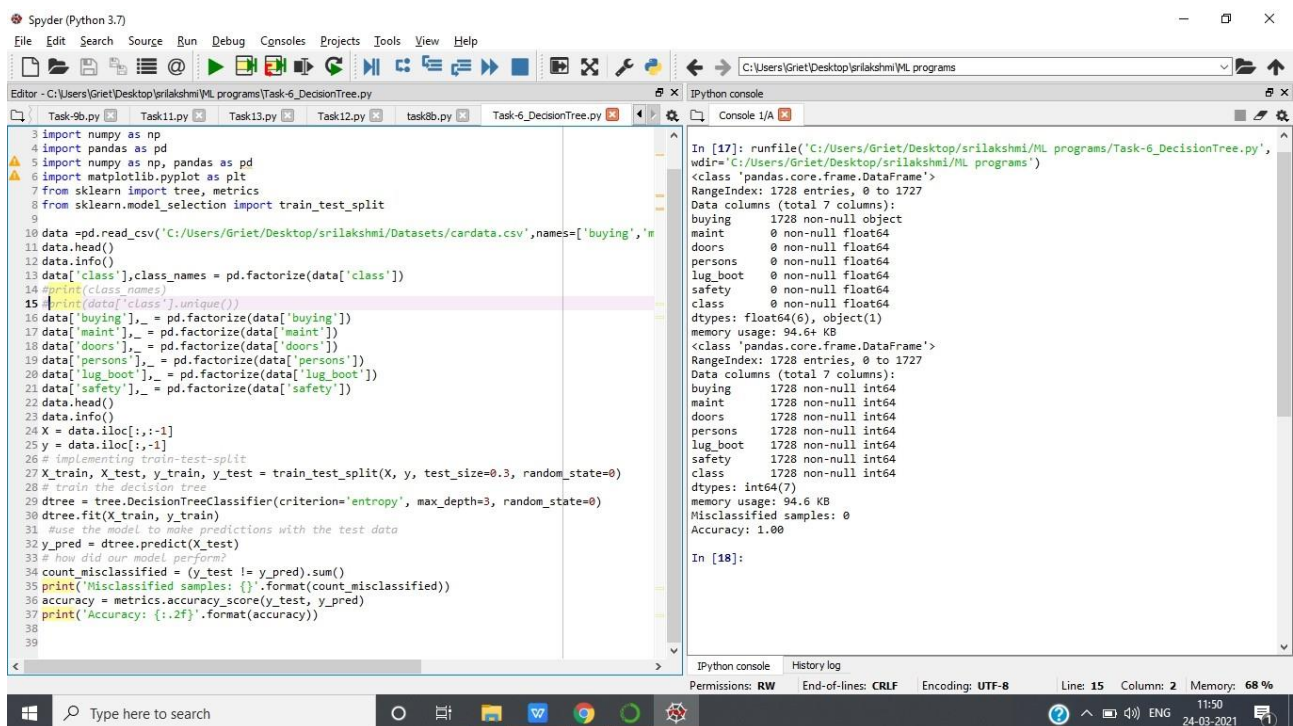
```
count_misclassified = (y_test != y_pred).sum()
```

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

```
accuracy = metrics.accuracy_score(y_test, y_pred)
```

```
print('Accuracy: {:.2f}'.format(accuracy))
```

Output:



```
3 import numpy as np
4 import pandas as pd
5 import numpy as np, pandas as pd
6 import matplotlib.pyplot as plt
7 from sklearn import tree, metrics
8 from sklearn.model_selection import train_test_split
9
10 data = pd.read_csv('C:/Users/Griet/Desktop/srilakshmi/Datasets/cardata.csv', names=['buying', 'm
11 data.head()
12 data.info()
13 data['class'], class_names = pd.factorize(data['class'])
14 #print(class_names)
15 #print(data['class'].unique())
16 data['buying'],_ = pd.factorize(data['buying'])
17 data['maint'],_ = pd.factorize(data['maint'])
18 data['doors'],_ = pd.factorize(data['doors'])
19 data['persons'],_ = pd.factorize(data['persons'])
20 data['lug_boot'],_ = pd.factorize(data['lug_boot'])
21 data['safety'],_ = pd.factorize(data['safety'])
22 data.head()
23 data.info()
24 X = data.iloc[:, :-1]
25 y = data.iloc[:, -1]
26 # implementing train-test-split
27 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
28 # train the decision tree
29 dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
30 dtree.fit(X_train, y_train)
31 #use the model to make predictions with the test data
32 y_pred = dtree.predict(X_test)
33 # how did our model perform?
34 count_misclassified = (y_test != y_pred).sum()
35 print('Misclassified samples: {}'.format(count_misclassified))
36 accuracy = metrics.accuracy_score(y_test, y_pred)
37 print('Accuracy: {:.2f}'.format(accuracy))
38
39
```

In [17]: runfile('C:/Users/Griet/Desktop/srilakshmi/ML_programs/Task-6_DecisionTree.py',
wdir='C:/Users/Griet/Desktop/srilakshmi/ML_programs')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
buying 1728 non-null object
maint 0 non-null float64
doors 0 non-null float64
persons 0 non-null float64
lug_boot 0 non-null float64
safety 0 non-null float64
class 0 non-null float64
dtypes: float64(6), object(1)
memory usage: 94.6+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
buying 1728 non-null int64
maint 1728 non-null int64
doors 1728 non-null int64
persons 1728 non-null int64
lug_boot 1728 non-null int64
safety 1728 non-null int64
class 1728 non-null int64
dtypes: int64(7)
memory usage: 94.6 KB
Misclassified samples: 0
Accuracy: 1.00

In [18]:

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
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["target_names"][prediction]))

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

```

Output:-

The screenshot shows the Spyder Python IDE with a file named 'task-4.py' open. The code in the editor implements a K-Nearest Neighbors (KNN) classifier on the Iris dataset. It loads the dataset, splits it into training and testing sets, trains a KNN model with 2 neighbors, and then iterates through the test set to make predictions and calculate the accuracy score.

The IPython console on the right displays the output of the code execution. It shows the Iris dataset features and target names, the training data, and the results of the predictions for the first few test samples. The final output is the TEST SCORE[ACCURACY]: 0.97.

```

In [64]: runfile('C:/Users/sunil/Desktop/ML(IV-I)/task-4.py', wdir='C:/Users/sunil/Desktop/ML(IV-I)')

IRIS FEATURES \ TARGET NAMES:
['setosa' 'versicolor' 'virginica']

[0]:[setosa]
[1]:[versicolor]
[2]:[virginica]

XNEW
[[5. 2.9 1. 0.2]]

Predicted target value: [0]

Predicted feature name: ['setosa']

XNEW
[[6. 2.2 4. 1. ]]

Actual : 1 versicolor, Predicted:[2]['virginica']

TEST SCORE[ACCURACY]: 0.97

In [65]:

```

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 longitudinally
= 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:-

The screenshot shows the Spyder Python IDE with a file named 'Task-11_Backpropagation.py' open. The code implements a neural network with forward and backward propagation. The output console shows the results of running the script.

```

14 def sigmoid(x): return(1/(1 + np.exp(-x)))
15 #Derivative of Sigmoid Function
16 def derivatives_sigmoid(x):return(x * (1 - x))
17 #Variable Initialization
18 epoch=7000 #Setting training iterations
19 lr=0.1 #Setting learning rate
20 inputlayer_neurons = 2 #number of features in data set
21 hiddenlayer_neurons = 3 #number of hidden layers neurons
22 output_neurons = 1 #number of neurons at output layer
23 #weight and bias initialization
24 wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
25 bh=np.random.uniform(size=(1,hiddenlayer_neurons))
26 wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
27 bout=np.random.uniform(size=(1,output_neurons))
28 #draws a random range of numbers uniformly of dim x*y
29 for i in range(epoch):
30 #Forward Propagation
31 hinpl=np.dot(X,wh)
32 hinp=hinpl + bh
33 hlayer_act = sigmoid(hinp)
34 outinp1=np.dot(hlayer_act,wout)
35 outinp= outinp1+ bout
36 output = sigmoid(outinp)
37 #Backpropagation
38 EO = y-output
39 outgrad = derivatives_sigmoid(output)
40 d_output = EO* outgrad
41 EH = d_output.dot(wout.T)
42 hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts
43 d_hiddenlayer = EH * hiddengrad
44 wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror and current
45 # bout += np.sum(d_output, axis=0,keepdims=True) *lr
46 wh += X.T.dot(d_hiddenlayer) *lr
47 #bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
48 print("Input: \n" + str(X))
49 print("Actual Output: \n" + str(y))
50 print("Predicted Output: \n" ,output)

```

IPython console output:

```

In [1]: runfile('C:/Users/sunil/ML programs/Task-11_Backpropagation.py',
wdir='C:/Users/sunil/ML programs')
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
[[0.89606579]
 [0.87704983]
 [0.89619347]]
In [2]:

```


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_test,
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:

The screenshot shows the Spyder Python IDE with a script for Random Forest classification. The script imports necessary libraries, loads the Pima Indians Diabetes dataset, splits it into training and testing sets, trains a Random Forest classifier, and prints various performance metrics including a confusion matrix, classification report, and AUC scores.

The IPython console output shows the following results:

```

In [79]: runfile('C:/Users/sunil/ML programs/Task-10_Randomforest.py',
wdir='C:/Users/sunil/ML programs')
=== Confusion Matrix ===
[[150  26]
 [ 33  45]]

=== Classification Report ===
              precision    recall  f1-score   support

      0       0.82      0.85      0.84       176
      1       0.63      0.58      0.60        78

   accuracy      0.73      0.71      0.72       254
  macro avg      0.73      0.71      0.72       254
 weighted avg      0.76      0.77      0.76       254

=== All AUC Scores ===
[0.78222222 0.82888889 0.83481481 0.73222222 0.81777778 0.86185185
 0.86592593 0.89888889 0.80384615 0.83538462]

=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.8261823361823362

In [80]:

```

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
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 ls==', 0.18279267873500388)

('root_mean_squared error of ls==', 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 algorithm
K = 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

```
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 plt.show()
```

Dataset

The top plot, titled 'Dataset', shows a scatter plot of 17 data points (x1 vs x2) with axes ranging from 0 to 10. The bottom plot shows the same data points clustered into three groups based on KMeans algorithm results. The clusters are represented by different colors and markers: blue circles (cluster 0), green downward triangles (cluster 1), and red squares (cluster 2). The axes and title are the same as the top plot.

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

23:43 11/03/2021