```
1a……………………………




graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
visited = set()
def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

2……………………..

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
visited = []
queue = []
def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)
  while queue:
    m = queue.pop(0)
    print (m, end = " ")
    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)
print("Following is the BFS")
bfs(visited, graph, '5')
```

2.

```
 def aStarAlgo(start_node, stop_node):
open_set = set(start_node)
closed_set = set()
g = {}
parents = {}
g[start_node] = 0
parents[start_node] = start_node
while len(open_set) > 0:
n = None
for v in open_set:
if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
n = v
if n == stop_node or Graph_nodes[n] == None:
pass
else:
for (m, weight) in get_neighbors(n):
if m not in open_set and m not in closed_set:
open_set.add(m)
parents[m] = n
g[m] = g[n] + weight
else:
if g[m] > g[n] + weight:
g[m] = g[n] + weight
parents[m] = n
if m in closed_set:
closed_set.remove(m)
open_set.add(m)
if n == None:
print('Path does not exist!')
return None
if n == stop_node:
```

```
path = []
while parents[n] != n:
path.append(n)
n = parents[n]
path.append(start_node)
path.reverse()
print('Path found: {}'.format(path))
return path
open_set.remove(n)
closed_set.add(n)
print('Path does not exist!')
return None
def get_neighbors(v):
if v in Graph_nodes:
return Graph_nodes[v]
else:
return None
def heuristic(n):
H_dist = {'A': 11,
'B': 6,
'C': 99,
'D': 1,
'E': 7,
'G': 0,
}
return H_dist[n]
Graph_nodes = { 'A': [('B', 2), ('E', 3)],
'B': [('C', 1),('G', 9)],
'C': None,
'E': [('D', 6)],
'D': [('G', 1)],
}
aStarAlgo('A', 'G')
```

3

```
from sklearn datasets import load_iris
from sklearn.naive_bayes import gaussian NB
from sklearn.model_selection import train_test.split
from sklearn.matrices import accuracy_score
iris=load_iris()
x_train,x_test_y_train,y_test=train_test_split
(iris.data,iris.target,test_size=0.3.random_state=4.2)
nb=gaussian NB()
nb.fit(x_train,y_train)
y_pred=nd.predict(x_test)
accuracy=accuracy_score(y_test,y_pred)
print("accuracy{:2f}%",format(accuracy*100);
```

4……….

```
 # Importing library
import math
import random
import csv
# the categorical class names are changed to numberic data
# eg: yes and no encoded to 1 and 0
def encode_class(mydata):
classes = []
for i in range(len(mydata)):
if mydata[i][-1] not in classes:
classes.append(mydata[i][-1])
for i in range(len(classes)):
for j in range(len(mydata)):
if mydata[j][-1] == classes[i]:
mydata[j][-1] = i
return mydata
# Splitting the data
def splitting(mydata, ratio):
train_num = int(len(mydata) * ratio)
train = []
# initially testset will have all the dataset
test = list(mydata)
while len(train) < train_num:
# index generated randomly from range 0
# to length of testset
index = random.randrange(len(test))
# from testset, pop data rows and put it in train
train.append(test.pop(index))
return train, test
# Group the data rows under each class yes or
# no in dictionary eg: dict[yes] and dict[no]
def groupUnderClass(mydata):
```

```
dict = {}
for i in range(len(mydata)):
if (mydata[i][-1] not in dict):
dict[mydata[i][-1]] = []
dict[mydata[i][-1]].append(mydata[i])
return dict
# Calculating Mean
def mean(numbers):
return sum(numbers) / float(len(numbers))
# Calculating Standard Deviation
def std_dev(numbers):
avg = mean(numbers)
variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
return math.sqrt(variance)
def MeanAndStdDev(mydata):
info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
# eg: list = [ [a, b, c], [m, n, o], [x, y, z]]
# here mean of 1st attribute =(a + m+x), mean of 2nd attribute = (b + n+y)/3
# delete summaries of last class
del info[-1]
return info
# find Mean and Standard Deviation under each class
def MeanAndStdDevForClass(mydata):
info = {}
dict = groupUnderClass(mydata)
for classValue, instances in dict.items():
info[classValue] = MeanAndStdDev(instances)
return info
# Calculate Gaussian Probability Density Function
def calculateGaussianProbability(x, mean, stdev):
expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo
# Calculate Class Probabilities
def calculateClassProbabilities(info, test):
probabilities = {}
for classValue, classSummaries in info.items():
probabilities[classValue] = 1
for i in range(len(classSummaries)):
mean, std_dev = classSummaries[i]
x = test[i]
probabilities[classValue] *= calculateGaussianProbability(x, mean, std_dev)
return probabilities
# Make prediction - highest probability is the prediction
def predict(info, test):
probabilities = calculateClassProbabilities(info, test)
bestLabel, bestProb = None, -1
for classValue, probability in probabilities.items():
if bestLabel is None or probability > bestProb:
bestProb = probability
bestLabel = classValue
return bestLabel
# returns predictions for a set of examples
def getPredictions(info, test):
predictions = []
for i in range(len(test)):
result = predict(info, test[i])
predictions.append(result)
return predictions
# Accuracy score
def accuracy_rate(test, predictions):
correct = 0
for i in range(len(test)):
if test[i][-1] == predictions[i]:
correct += 1
return (correct / float(len(test))) * 100.0
# driver code
# add the data path in your system
filename = r'E:\pythonProject1\pima-indians-diabetes.csv'
```

```
# load the file and store it in mydata list
mydata = csv.reader(open(filename, "rt"))
mydata = list(mydata)
mydata = encode_class(mydata)
for i in range(len(mydata)):
mydata[i] = [float(x) for x in mydata[i]]
# split ratio = 0.7
# 70% of data is training data and 30% is test data used for testing
ratio = 0.7
train_data, test_data = splitting(mydata, ratio)
print('Total number of examples are: ', len(mydata))
print('Out of these, training examples are: ', len(train_data))
print("Test examples are: ", len(test_data))
# prepare model
info = MeanAndStdDevForClass(train_data)
# test model
predictions = getPredictions(info, test_data)
accuracy = accuracy_rate(test_data, predictions)
print("Accuracy of your model is: ", accuracy)
```

5a……………..

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.arr([2, 4, 6, 8, 10])
regressor = LinearRegression()
regressor.fit(x, y)
y_pred = regressor.predict(x)
print('Coefficients:', regressor.coef_)
print('Intercept:', regressor.intercept_)
plt.scatter(x, y, color = 'black')
plt.plot(x, y_pred, color = 'blue', linewidth = 3)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

5b…………….

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
X = np.array([[1, 2], [2, 3], [4, 5], [5, 6]])
y = np.array([0, 0, 1, 1])
classifier = LogisticRegression()
classifier.fit(X, y)
print('Coefficient:', classifier.coef_)
print('Intercept:', classifier.intercept_)
xx, yy = np.meshgrid(np.arange(0, 6, 0.01), np.arange(0, 8, 0.01))
z = classifier.predict(np.c_[xx.ravel(),yy.ravel()])
z = z.reshape(xx.shape)
plt.contourf(xx, yy, z, cmap = plt.cm.RdBu)
plt.scatter(X[:, 0], X[:, 1], c = y, cmap = plt.cm.RdBu_r, edgecolors = 'k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Logistic Regression')
plt.show()
```

6a……………..

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
iris=load_iris()
clf = DecisionTreeClassifier(random_state = 0)
clf.fit(iris.data,iris.taget)
plot_tree(clf,filled=True)
plt.show()
```

6b………………

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrices import accuracy_score
import matplotlib.pyplot as plt
iris = load_iris()
df = pd.DataFrame(data = iris.data, columns = iris.feature_names)
df[‘target’] = iris.target
X_train, X_test, y_train, y_test = train_test_split(df[iris.feature_names], df[‘target’], test_size =
0.3)
rfc = RandomForestClassifier(n_estimators = 100, max_depth = 2, random_state = 0)
rfc.fit(X_train,y_train)
y_pred = rfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
importances = rfc.feature_importances_
indices = list(range(len(importances)))
plt.bar(indices, importances, color = ‘r’)
plt.xticks(indices, iris.feature_names, rotation = 90)
plt.title(‘Feature Importance’)
plt.show()
```