LAB 1

```
graph={
    'L':['M','N','O'],
    'M':['N','O'],
    'N':['P'],
    'O':[],
    'P':[]
}
visited=set()   #set to keep track of visited nodes of graph.
def dfs(visited, graph, node):   #function for dfs
    if node not in visited:
        print(node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

#driver code
print("Following is the Depth First Search:")
dfs(visited,graph,'L')
```

```
Following is the Depth First Search:
L
M
N
P
O
```

```
graph={
    'L':['M','N','O'],
    'M':['N','O'],
    'N':['P'],
    'O':[],
    'P':[]
}

visited=[]   #list for visited nodes
queue=[]     #initialize a queue

def bfs(visited, graph, node):  #function for BFS
    visited.append(node)
    queue.append(node)

    while queue:   #creating loop to visit each node
        m=queue.pop(0)
        print(m)

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

#driver code
print("Following is the Breadth First Search:")
bfs(visited, graph, 'L')  #function calling
```

```
Following is the Breadth First Search:
L
M
N
O
P
```

Name: Ayesha Farukh Shaikh
Subject: AI
Roll No.: COTC62

Aim: 2. Implement A star Algorithm for any game search problem.

INPUT:

```
from copy import deepcopy
import numpy as np
import time

def bestsolution(state):
    bestsol = np.array([], int).reshape(-1, 9)
    count = len(state) - 1
    while count != -1:
        bestsol = np.insert(bestsol, 0, state[count]['puzzle'], 0)
        count = (state[count]['parent'])
    return bestsol.reshape(-1, 3, 3)


# checks for the uniqueness of the iteration(it).
def all(checkarray):
    set=[]
    for it in set:
        for checkarray in it:
            return 1
        else:
            return 0


# number of misplaced tiles
def misplaced_tiles(puzzle,goal):
    mscost = np.sum(puzzle != goal) - 1
    return mscost if mscost > 0 else 0


def coordinates(puzzle):
    pos = np.array(range(9))
    for p, q in enumerate(puzzle):
        pos[q] = p
    return pos


# start of 8 puzzle evaluvation, using Misplaced tiles heuristics
def evaluvate_misplaced(puzzle, goal):
    steps = np.array([('up', [0, 1, 2], -3),('down', [6, 7, 8],  3),('left', [0, 3, 6], -1),('right', [2, 5, 8],  1)],
                dtype =  [('move',  str, 1),('position', list),('head', int)])

    dtstate = [('puzzle',  list),('parent', int),('gn',  int),('hn',  int)]

    costg = coordinates(goal)
```

```
    # initializing the parent, gn and hn, where hn is misplaced_tiles  function call
    parent = -1
    gn = 0
    hn = misplaced_tiles(coordinates(puzzle), costg)
    state = np.array([(puzzle, parent, gn, hn)], dtstate)

   #priority queues with position as keys and fn as value.
    dtpriority = [('position', int),('fn', int)]

    priority = np.array([(0, hn)], dtpriority)

    while 1:
        priority = np.sort(priority, kind='mergesort', order=['fn', 'position'])
        position, fn = priority[0]
        # sort priority queue using merge sort,the first element is picked for exploring.
        priority = np.delete(priority, 0, 0)
        puzzle, parent, gn, hn = state[position]
        puzzle = np.array(puzzle)

        blank = int(np.where(puzzle == 0)[0])

        gn = gn + 1
        c = 1
        start_time = time.time()
        for s in steps:
            c = c + 1
            if blank not in s['position']:
                openstates = deepcopy(puzzle)
                openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']],
openstates[blank]

                if ~(np.all(list(state['puzzle']) == openstates, 1)).any():
                    end_time = time.time()
                    if (( end_time - start_time ) > 2):
                        print(" The 8 puzzle is unsolvable \n")
                        break

                    hn = misplaced_tiles(coordinates(openstates), costg)
                    # generate and add new state in the list
                    q = np.array([(openstates, position, gn, hn)], dtstate)
                    state = np.append(state, q, 0)
                    # f(n) is the sum of cost to reach node
                    fn = gn + hn

                    q = np.array([(len(state) - 1, fn)], dtpriority)
                    priority = np.append(priority, q, 0)

                    if np.array_equal(openstates, goal):
                        print(' The 8 puzzle is solvable \n')
                        return state, len(priority)

    return state, len(priority)
```

```
# initial state
puzzle = []

puzzle.append(2)
puzzle.append(8)
puzzle.append(3)
puzzle.append(1)
puzzle.append(6)
puzzle.append(4)
puzzle.append(7)
puzzle.append(0)
puzzle.append(5)

#goal state
goal = []

goal.append(1)
goal.append(2)
goal.append(3)
goal.append(8)
goal.append(0)
goal.append(4)
goal.append(7)
goal.append(6)
goal.append(5)


state, visited = evaluvate_misplaced(puzzle, goal)
bestpath = bestsolution(state)
print(str(bestpath).replace('[', ' ').replace(']', ''))
totalmoves = len(bestpath) - 1
print('\nSteps to reach goal:',totalmoves)
visit = len(state) - visited
print('Total nodes visited: ',visit, "\n")
```

OUTPUT:

```
The 8 puzzle is solvable

  2 8 3
  1 6 4
  7 0 5

  2 8 3
  1 0 4
  7 6 5

  2 0 3
  1 8 4
```

7 6 5

0 2 3
1 8 4
7 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5

Steps to reach goal: 5
Total nodes visited: 6

Name : Ayesha Farukh Shaikh Roll no :COTC62 Batch : C3

INPUT :

```
import sys


class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printMST(self, parent):
        print("Edge \tWeight")
        for i in range(1, self.V):
            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])

    def minKey(self, key, mstSet):
        min = sys.maxsize
        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v

    return min_index
    def primMST(self):
        key = [sys.maxsize] * self.V
        parent = [None] * self.V
        key[0] = 0
        mstSet = [False] * self.V

        parent[0] = -1
```

```
                for cout in range(self.V):
                    u = self.minKey(key, mstSet)
                        mstSet[u] = True
                    for v in range(self.V):
                        if self.graph[u][v] > 0 and mstSet[v] == False \
                        and key[v] > self.graph[u][v]:
                            key[v] = self.graph[u][v]
                            parent[v] = u

                self.printMST(parent)

if __name__== '__main__':
        g = Graph(5)
        g.graph = [[0, 2, 0, 6, 0],
                   [2, 0, 3, 8, 5],
                   [0, 3, 0, 0, 7],
                   [6, 8, 0, 0, 9],
                   [0, 5, 7, 9, 0]]

        g.primMST()
```

OUTPUT :

```
    Edge      Weight
     0 - 1      2
     1 - 2      3
     0 - 3      6
     1 - 4      5
```

Name- Ayesha Shaikh
Class-TE C Rollno-COTC62

```
from __future__ import print_function
N = 8

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end = " ")
        print()

def isSafe(row, col, slashCode, backslashCode,
           rowLookup, slashCodeLookup,
                    backslashCodeLookup):
    if (slashCodeLookup[slashCode[row][col]] or
        backslashCodeLookup[backslashCode[row][col]] or
        rowLookup[row]):
        return False
    return True

def solveNQueensUtil(board, col, slashCode, backslashCode,
                     rowLookup, slashCodeLookup,
                     backslashCodeLookup):

    if(col >= N):
        return True
    for i in range(N):
        if(isSafe(i, col, slashCode, backslashCode,
                  rowLookup, slashCodeLookup,
                  backslashCodeLookup)):

            board[i][col] = 1
            rowLookup[i] = True
            slashCodeLookup[slashCode[i][col]] = True
            backslashCodeLookup[backslashCode[i][col]] = True

            if(solveNQueensUtil(board, col + 1,
                                slashCode, backslashCode,
                                rowLookup, slashCodeLookup,
                                backslashCodeLookup)):
                return True

            board[i][col] = 0
            rowLookup[i] = False
            slashCodeLookup[slashCode[i][col]] = False
            backslashCodeLookup[backslashCode[i][col]] = False

    return False

def solveNQueens():
    board = [[0 for i in range(N)]
```

```
            for j in range(N)]

    # helper matrices
    slashCode = [[0 for i in range(N)]
                 for j in range(N)]
    backslashCode = [[0 for i in range(N)]
                     for j in range(N)]

    # arrays to tell us which rows are occupied
    rowLookup = [False] * N

    # keep two arrays to tell us
    # which diagonals are occupied
    x = 2 * N - 1
    slashCodeLookup = [False] * x
    backslashCodeLookup = [False] * x

    # initialize helper matrices
    for rr in range(N):
        for cc in range(N):
            slashCode[rr][cc] = rr + cc
            # DIAGONAL CONDITION
            backslashCode[rr][cc] = rr - cc + 7

    if(solveNQueensUtil(board, 0, slashCode, backslashCode,
                        rowLookup, slashCodeLookup,
                        backslashCodeLookup) == False):
        print("Solution does not exist")
        return False

    # solution found
    printSolution(board)
    return True

# Driver Code
solveNQueens()
```

OUTPUT:

```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0

True
```

Name-Ayesha Shaikh
Class-TE C Rollno-COTC62

```
import random

# Define some responses for the chatbot
responses = {
    "hi": ["Hello!", "Hi there!", "Hey!"],
    "how are you": ["I'm doing well, thank you!", "I'm fine, thanks for
asking.", "Pretty good!"],
    "default": ["I'm sorry, I didn't understand what you said.", "Can you please
rephrase that?", "I'm not sure what you mean."],
    "which mobile do you want to purchase":["Redmi"],
    "mobile cost":["20000"],
    "warranty period":["1 year"]
}

# Define a function to respond to user input
def chatbot_response(user_input):
    if user_input.lower() in responses:
        return random.choice(responses[user_input.lower()])
    else:
        return random.choice(responses["default"])

# Chat with the user
print("Hi, I'm a simple chatbot. What can I help you with today?")
while True:
    user_input = input()
    if user_input.lower() == "bye":
        print("Goodbye!")
        break
    else:
        print(chatbot_response(user_input))
```

OUTPUT:

```
Hi, I'm a simple chatbot. What can I help you with today?
"how are you"
Pretty good!
"which mobile do you want to purchase"
Redmi
"mobile cost"
20000
"warranty period"
1 year
"bye"
Goodbye!
```

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearch
CV
from sklearn.metrics import accuracy_score, classification_repor
t, confusion_matrix
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [3]:
```
data = pd.read_csv('Hr.csv')
```

In [4]:
```
data.shape
```

Out[4]:
```
(1200, 28)
```

In [5]:
```
data.columns
```

Out[5]:
```
Index([u'EmpNumber', u'Age', u'Gender', u'EducationBackground',
       u'MaritalStatus', u'EmpDepartment', u'EmpJobRole',
       u'BusinessTravelFrequency', u'DistanceFromHome', u'EmpEdu
cationLevel',
       u'EmpEnvironmentSatisfaction', u'EmpHourlyRate', u'EmpJob
Involvement',
       u'EmpJobLevel', u'EmpJobSatisfaction', u'NumCompaniesWork
ed',
       u'OverTime', u'EmpLastSalaryHikePercent',
       u'EmpRelationshipSatisfaction', u'TotalWorkExperienceInYe
ars',
       u'TrainingTimesLastYear', u'EmpWorkLifeBalance',
       u'ExperienceYearsAtThisCompany', u'ExperienceYearsInCurre
ntRole',
       u'YearsSinceLastPromotion', u'YearsWithCurrManager', u'At
trition',
       u'PerformanceRating'],
      dtype='object')
```

In [6]:
```
data.head()
```

Out[6]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartme
0	E1001000	32	Male	Marketing	Single	Sales
1	E1001006	47	Male	Marketing	Single	Sales
2	E1001007	40	Male	Life Sciences	Married	Sales
3	E1001009	41	Male	Human Resources	Divorced	Human Resources
4	E1001010	60	Male	Marketing	Single	Sales

5 rows × 28 columns

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 28 columns):
EmpNumber                          1200 non-null object
Age                                1200 non-null int64
Gender                             1200 non-null object
EducationBackground                1200 non-null object
MaritalStatus                      1200 non-null object
EmpDepartment                      1200 non-null object
EmpJobRole                         1200 non-null object
BusinessTravelFrequency            1200 non-null object
DistanceFromHome                   1200 non-null int64
EmpEducationLevel                  1200 non-null int64
EmpEnvironmentSatisfaction         1200 non-null int64
EmpHourlyRate                      1200 non-null int64
EmpJobInvolvement                  1200 non-null int64
EmpJobLevel                        1200 non-null int64
EmpJobSatisfaction                 1200 non-null int64
NumCompaniesWorked                 1200 non-null int64
OverTime                           1200 non-null object
EmpLastSalaryHikePercent           1200 non-null int64
EmpRelationshipSatisfaction        1200 non-null int64
TotalWorkExperienceInYears         1200 non-null int64
TrainingTimesLastYear              1200 non-null int64
EmpWorkLifeBalance                 1200 non-null int64
ExperienceYearsAtThisCompany       1200 non-null int64
ExperienceYearsInCurrentRole       1200 non-null int64
YearsSinceLastPromotion            1200 non-null int64
YearsWithCurrManager               1200 non-null int64
Attrition                          1200 non-null object
PerformanceRating                  1200 non-null int64
dtypes: int64(19), object(9)
memory usage: 262.6+ KB
```

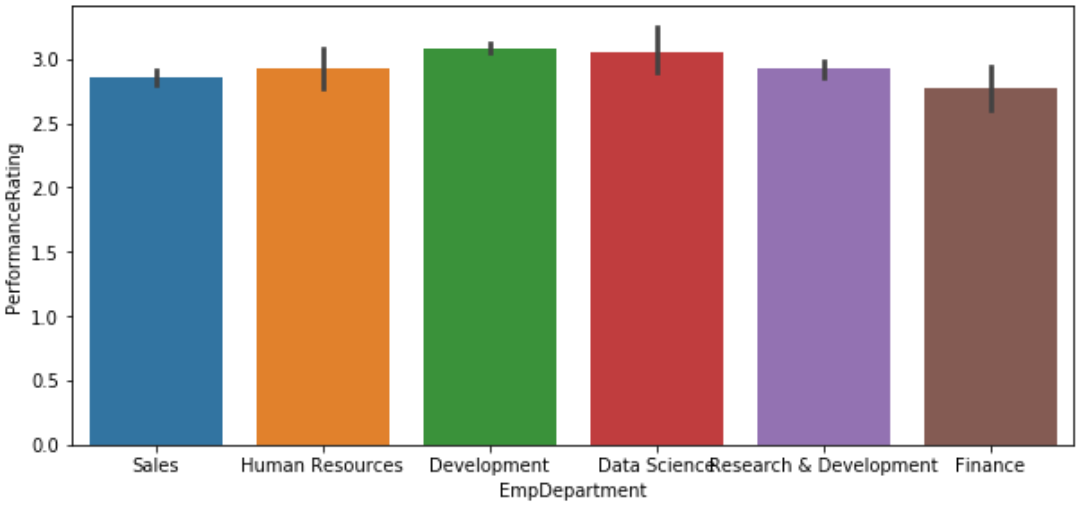
In [8]:
```
dept = data.iloc[:,[5,27]].copy()
dept_per = dept.copy()
```

In [9]:
```
dept_per.groupby(by='EmpDepartment')['PerformanceRating'].mean()
```

Out[9]:
```
EmpDepartment
Data Science                3.050000
Development                 3.085873
Finance                     2.775510
Human Resources             2.925926
Research & Development      2.921283
Sales                       2.860590
Name: PerformanceRating, dtype: float64
```

In [10]:
```
plt.figure(figsize=(10,4.5))
sns.barplot(dept_per['EmpDepartment'],dept_per['PerformanceRatin
g'])
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f078c105a50>
```

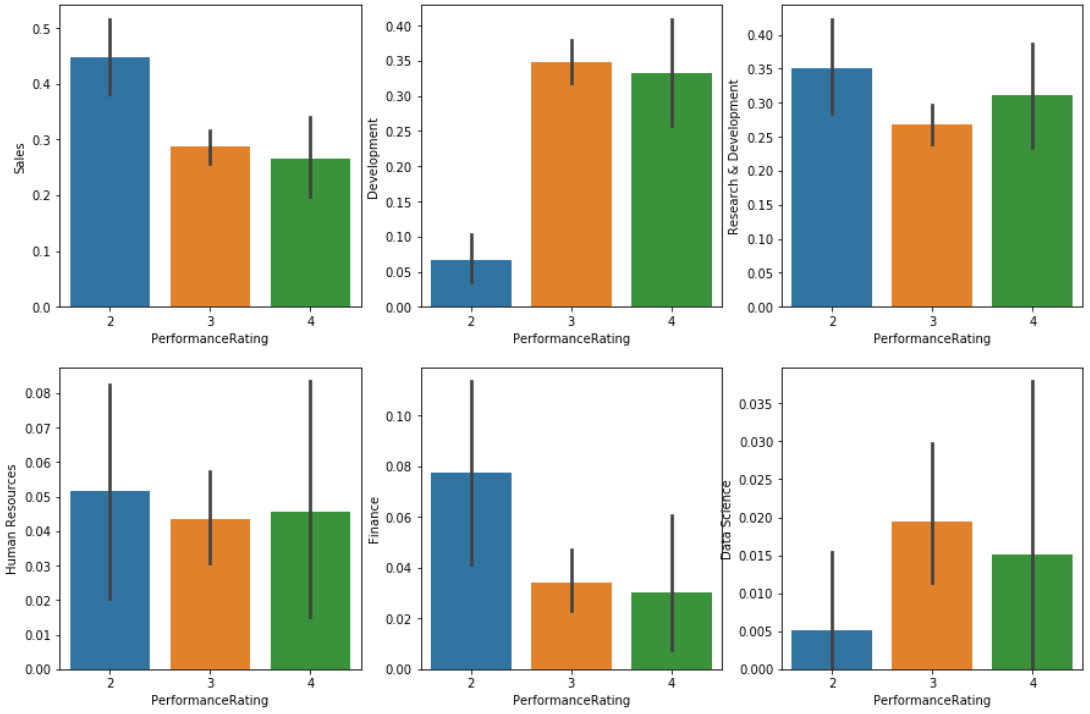


In [11]:
```
dept_per.groupby(by='EmpDepartment')['PerformanceRating'].value_
counts()
```

Out[11]:
```
EmpDepartment           PerformanceRating
Data Science            3                    17
                        4                     2
                        2                     1
Development             3                   304
                        4                    44
                        2                    13
Finance                 3                    30
                        2                    15
                        4                     4
Human Resources         3                    38
                        2                    10
                        4                     6
Research & Development  3                   234
                        2                    68
                        4                    41
Sales                   3                   251
                        2                    87
                        4                    35
Name: PerformanceRating, dtype: int64
```

In [12]:
```
department = pd.get_dummies(dept_per['EmpDepartment'])
performance = pd.DataFrame(dept_per['PerformanceRating'])
dept_rating = pd.concat([department,performance],axis=1)
```

In [14]:
```
plt.figure(figsize=(15,10))
plt.subplot(2,3,1)
sns.barplot(dept_rating['PerformanceRating'],dept_rating['Sales
'])
plt.subplot(2,3,2)
sns.barplot(dept_rating['PerformanceRating'],dept_rating['Develo
pment'])
plt.subplot(2,3,3)
sns.barplot(dept_rating['PerformanceRating'],dept_rating['Resear
ch & Development'])
plt.subplot(2,3,4)
sns.barplot(dept_rating['PerformanceRating'],dept_rating['Human
Resources'])
plt.subplot(2,3,5)
sns.barplot(dept_rating['PerformanceRating'],dept_rating['Financ
e'])
plt.subplot(2,3,6)
sns.barplot(dept_rating['PerformanceRating'],dept_rating['Data S
cience'])
plt.show()
```



In [16]:
```
#Data Processing
enc = LabelEncoder()
for i in (2,3,4,5,6,7,16,26):
    data.iloc[:,i] = enc.fit_transform(data.iloc[:,i])
data.head()
```

Out[16]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartme
0	E1001000	32	1	2	2	5
1	E1001006	47	1	2	2	5
2	E1001007	40	1	1	1	5
3	E1001009	41	1	0	0	3
4	E1001010	60	1	2	2	5

5 rows × 28 columns

In [17]:
```
data.corr()
```

Out[17]:

	Age	Gender	EducationBackground	Ma
Age	1.000000	-0.040107	-0.055905	-0.(
Gender	-0.040107	1.000000	0.009922	-0.(
EducationBackground	-0.055905	0.009922	1.000000	-0.(
MaritalStatus	-0.098368	-0.042169	-0.001097	1.0
EmpDepartment	-0.000104	-0.010925	-0.026874	0.0
EmpJobRole	-0.037665	0.011332	-0.012325	0.0
BusinessTravelFrequency	0.040579	-0.043608	0.012382	0.0
DistanceFromHome	0.020937	-0.001507	-0.013919	-0.(
EmpEducationLevel	0.207313	-0.022960	-0.047978	0.0
EmpEnvironmentSatisfaction	0.013814	0.000033	0.045028	-0.(
EmpHourlyRate	0.062867	0.002218	-0.030234	-0.(
EmpJobInvolvement	0.027216	0.010949	-0.025505	-0.(
EmpJobLevel	0.509139	-0.050685	-0.056338	-0.(
EmpJobSatisfaction	-0.002436	0.024680	-0.030977	0.0
NumCompaniesWorked	0.284408	-0.036675	-0.032879	-0.(
OverTime	0.051910	-0.038410	0.007046	-0.(
EmpLastSalaryHikePercent	-0.006105	-0.005319	-0.009788	0.0
EmpRelationshipSatisfaction	0.049749	0.030707	0.005652	0.0
TotalWorkExperienceInYears	0.680886	-0.061055	-0.027929	-0.(
TrainingTimesLastYear	-0.016053	-0.057654	0.051596	0.0
EmpWorkLifeBalance	-0.019563	0.015793	0.022890	0.0
ExperienceYearsAtThisCompany	0.318852	-0.030392	-0.009887	-0.(
ExperienceYearsInCurrentRole	0.217163	-0.031823	-0.003215	-0.(
YearsSinceLastPromotion	0.228199	-0.021575	0.014277	-0.(
YearsWithCurrManager	0.205098	-0.036643	0.002767	-0.(
Attrition	-0.189317	0.035758	0.027161	0.1
PerformanceRating	-0.040164	-0.001780	0.005607	0.0

27 rows × 27 columns

In [18]:
```
data.drop(['EmpNumber'],inplace=True,axis=1)
```

In [19]:
```
data.head()
```

Out[19]:

	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRo
0	32	1	2	2	5	13
1	47	1	2	2	5	13
2	40	1	1	1	5	13
3	41	1	0	0	3	8
4	60	1	2	2	5	13

5 rows × 27 columns

In [40]:
```
y=data.PerformanceRating
X=data.iloc[:,[4,5,9,16,20,21,22,23,24]]
X.head()
```

Out[40]:

	EmpDepartment	EmpJobRole	EmpEnvironmentSatisfaction	EmpLastSalaryHik
0	5	13	4	12
1	5	13	4	12
2	5	13	4	21
3	3	8	2	15
4	5	13	1	14

In [41]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=
0.3,random_state=10)
```

In [42]:
```
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [43]:
```
X_train.shape
```

Out[43]: (840, 9)

In [44]:
```
X_test.shape
```

Out[44]: (360, 9)

In [46]:
```
from sklearn.ensemble import RandomForestClassifier
classifier_rfg=RandomForestClassifier(random_state=33,n_estimato
rs=23)
parameters=[{'min_samples_split':[2,3,4,5],'criterion':['gini','
entropy'],'min_samples_leaf':[1,2,3],'min_samples_split':[2,3,4,
5]}]
model_gridrf=GridSearchCV(estimator=classifier_rfg, param_grid=p
arameters, scoring='accuracy')
model_gridrf.fit(X_train,y_train)
```

Out[46]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_we
ight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=
None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=23, n_job
s=1,
            oob_score=False, random_state=33, verbose=0, warm_st
art=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'min_samples_split': [2, 3, 4, 5], 'criterio
n': ['gini', 'entropy'], 'min_samples_leaf': [1, 2, 3]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='
warn',
       scoring='accuracy', verbose=0)
```

In [47]:
```
model_gridrf.best_params_
```

Out[47]:
```
{'criterion': 'entropy', 'min_samples_leaf': 2, 'min_samples_spl
it': 2}
```

In [48]:
```
y_predict_rf = model_gridrf.predict(X_test)
```

In [49]:
```
print(accuracy_score(y_test,y_predict_rf))
print(classification_report(y_test,y_predict_rf))
```

```
0.930555555556
             precision    recall  f1-score   support

          2       0.92      0.89      0.90        63
          3       0.94      0.97      0.96       264
          4       0.83      0.73      0.77        33

avg / total       0.93      0.93      0.93       360
```

In [50]:
```
confusion_matrix(y_test,y_predict_rf)
```

Out[50]:
```
array([[ 56,   7,   0],
       [  4, 255,   5],
       [  1,   8,  24]])
```