ANNA UNIVERSITY

THANTHAI PERIYAR

GOVERNMENT INSTITUTE OF TECHNOLOGY

VELLORE-632 002

MASTER OF COMPUTER APPLICATIONS

MC4311– MACHINE LEARNING

Name: ______________________________

Reg. No: ______________________________

THANTHAI PERIYAR GOVERNMENT INSTITUTE OF TECHNOLOGY

VELLORE-632 002

MASTER OF COMPUTER APPLICATIONS

MC4311 –MACHINE LEARNING LABORATORY

2023 – 2025

Certified that this is a bonafide record of work done by …...with Reg. no ....in this department during the academic year of 2024 – 2025.

Staff Incharge **Head of the Department**

Date:

Submitted for M.C.A Degree Practical Examination (III Semester) held on

….. at TPGIT Bagayam, Vellore – 2.

Internal Examiner **External Examiner**

INDEX

STRUCTURE DATA

PROGRAM:

In [1]:
```
import pandas as pd
```

In [8]:
```
s1=pd.Series([2201,2202,2203,2204,2205,2206,2207,2208,2209,2210])
s2=pd.Series(["Shibu","Vicky","Akash","Surya","Nandhu","Vinzz","Lechu","Preethi",
              "Raghul","Mathesh"])
s3=pd.Series(["MCA","CSC","IT","MCA","IT","CSC","BCA","AI","BCA","BSC"])
s4=pd.Series["02/09/2001","12/08/2002","23/04/2001","28/05/2000","30/02/2002",
"26/07/2000","20/05/2002","19/08/2001","28/01/2001","01/09/2000"])
s5=pd.Series([23,22,23,24,22,24,22,23,23,24])
s6=pd.Series(["Male","Male","Male","Male","Male","Female","Female",
              "Female","Male","Male"])
s7=pd.Series(["shibu@gmail.com","vicky7@gmail.com","akash12@gmail.com","surya7@gmail.com,
             "nandhu19@gmail.com","vinzz11@gmail.com","lechu05@gmail.com",
             "preethi8@gmail.com","raghul@gmail.com","mathesh20@gmail.com"])
s8=pd.Series([8479782752,92378451052,8752305137,9462357103,8967453210,9087635812,
              9845671230,8765423912,8654320852,9543267539])
Student=pd.DataFrame({"REGNO":s1,"NAME":s2,"DEPT":s3,"DOB":s4,"AGE":s5,"GENDER":s6,
                      "EMAIL-ID":s7,"PH.NO":s8})
```

In [9]:
```
Student
```

OUTPUT:

Out[9]:

	REGNO	NAME	DEPT	DOB	AGE	GENDER	EMAIL-ID	PH.NO
0	1890	Shibu	MCA	02/09/2001	23	Male	shibu@gmail.com	8479782752
1	1891	Vicky	CSC	12/08/2002	22	Male	vicky7@gmail.com	92378451052
2	1892	Akash	IT	23/04/2001	23	Male	akash12@gmail.com	8752305137
3	1893	Surya	MCA	28/05/2000	24	Male	surya7@gmail.com	9462357103
4	1894	Nandhu	IT	30/02/2002	22	Male	nandhu19@gmail.com	8967453210
5	1895	Vinzz	CSC	26/07/2000	24	Female	vinzz11@gmail.com	9087635812
6	1896	Lechu	BCA	20/05/2002	22	Female	lechu05@gmail.com	9845671230
7	1897	Preethi	AI	19/08/2001	23	Female	preethi8@gmail.com	8765423912
8	1898	Raghul	BCA	28/01/2001	23	Male	raghul@gmail.com	8654320852
9	1899	Mathesh	BSC	01/09/2000	24	Male	mathesh20@gmail.com	9543267539

DATA PRE-PROCESSING

PROGRAM:

In [62]:
```
Import numpy as nm
Import pandas as pd
```

In [63]:
```
dataset=pd.read_csv('Data1.csv')
```

In [64]:
```
Dataset
```

Out[64]:

	Country	Age	Salary	Purchased
0	France	23.0	72000.0	No
1	Spain	14.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In[65]:
```
x=dataset.iloc[:,:-1].values
```

In[66]:
```
x
```

Out[66]:
```
array([['France',23.0,72000.0],
       ['Spain',14.0,48000.0],
    ['Germany',30.0,54000.0],
    ['Spain',38.0,61000.0],
    ['Germany',40.0,nan],
    ['France',35.0,58000.0],
    ['Spain',nan,52000.0],
    ['France',48.0,79000.0],
    ['Germany',50.0,83000.0],
    ['France',37.0,67000.0]],dtype=object)
```

In[67]:
```
y=dataset.iloc[:,3].values
```

In[68]:
```
y
```

Out[68]:
```
array(['No','Yes','No','No','Yes','Yes','No','Yes','No','Yes'],   dtype=object)
```

In[69]:
```
From sklearn.impute import SimpleImputer
```

In[70]:
```
imputer=SimpleImputer(missing_values=nm.nan,strategy='mean')
```

In[71]:
```
imputer1=imputer.fit(x[:,1:3])
```

In[72]:
```
x[:,1:3]=imputer.transform(x[:,1:3])
x
```

Out[73]:
```
array([['France',23.0,72000.0],
       ['Spain',14.0,48000.0],
       ['Germany',30.0,54000.0],
       ['Spain',38.0,61000.0],
       ['Germany',40.0,63777.77777777778],
       ['France',35.0,58000.0],
       ['Spain',35.0,52000.0],
       ['France',48.0,79000.0],
       ['Germany',50.0,83000.0],
       ['France',37.0,67000.0]],dtype=object)
```

In[74]:
```
From sklearn.preprocessing import LabelEncoder
```

In[75]:
```
labelencoder_x=LabelEncoder()
```

In[76]:
```
x[:,0]=labelencoder_x.fit_transform(x[:,0])
```

In[77]:
```
x
```

Out[77]:
```
array([[0, 23.0, 72000.0],
       [2, 14.0, 48000.0],
       [1, 30.0, 54000.0],
       [2, 38.0, 61000.0],
       [1, 40.0, 63777.77777777778],
       [0, 35.0, 58000.0],
       [2, 35.0, 52000.0],
       [0, 48.0, 79000.0],
       [1, 50.0, 83000.0],
       [0, 37.0, 67000.0]],dtype=object)
```

In[78]:
```
From sklearn.compose import ColumnTransformer
```

In[79]:
```
From sklearn.preprocessing import OneHotEncoder
```

In[82]:
```
ct=ColumnTransformer([("Country",OneHotEncoder(),[0])],remainder="passthrough")
```

In[83]:
```
ct=ColumnTransformer([("Country",OneHotEncoder(),[0])],remainder="passthrough")
```

In[84]:
```
x=ct.fit_transform(x)
```

In[85]:
```
x=x[:,0:]
```

In[86]:
```
x
```

Out[86]:
```
array([[1.0, 0.0, 0.0, 23.0, 72000.0],
       [0.0, 0.0, 1.0, 14.0, 48000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 35.0, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]],dtype=object)
```

In[87]:
```
label_encoder_y=LabelEncoder()
```

In[88]:
```
y=label_encoder_y.fit_transform(y)
```

In[89]:
```
From sklearn.model_selection import train_test_split
```

In[90]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

In[91]:
```
x_train
```

Out[91]:
```
array([[0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 37.0, 67000.0],
       [0.0, 0.0, 1.0, 14.0, 48000.0],
       [0.0, 0.0, 1.0, 35.0, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [1.0, 0.0, 0.0, 23.0, 72000.0],
       [1.0, 0.0, 0.0, 35.0, 58000.0]],dtype=object)
```

In[92]:
```
x_test
```

Out[92]:
```
array([[0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0]],dtype=object)
```

In[93]:
```
y_train
```

Out[93]:
```
array([1,1,1,0,1,0,0,1])
```

In[94]:
```
y_test
```

Out[94]:
```
array([0,0])
```

In[97]:
```
From sklearn.preprocessing import StandardScaler
```

In[98]:
```
st_x=StandardScaler()
```

In[99]:
```
x_train=st_x.fit_transform(x_train)
```

In[100]:
```
x=test=st_x.transform(x_test)
```

In[103]:
```
print("Feature Scaling")
print("x_train \n",x_train)
print("x_test \n",x_test)
```

OUTPUT:

```
FeatureScaling
x_train

 [[-1.          2.64575131-0.774596670.633165070.12381479]
  [1.          -0.37796447 -0.77459667  0.32924584  0.46175632]
 [-1.          -0.37796447  1.29099445 -2.00080164 -1.53093341]
 [-1.          -0.37796447  1.29099445  0.12663301 -1.11141978]
  [1.          -0.37796447 -0.77459667  1.44361637  1.7202972]
 [-1.          -0.37796447  1.29099445  0.43055225 -0.16751412]
  [1.          -0.37796447 -0.77459667 -1.08904393  0.98614835]
  [1.          -0.37796447 -0.77459667  0.12663301 -0.48214934]]

x_test
 [[0.01.00.030.054000.0]
 [0.01.00.050.083000.0]]
```

FEATURE SUBSET SELECTION

PROGRAM:

In [1]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
data="https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-d
features=['preg','plas','pres','skin','test','mass','pedi','age','class']
df=pd.read_csv(data,names=features)
```

In [5]:
```
df.head()
```

Out[5]:

	preg	plas	Pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [7]:
```
df.shape
```

Out[7]: (768, 9)

In [9]:
```
data=df.values
x=data[:,0:8]
y=data[:,8]
```

In [11]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [13]:
```
chi_best=SelectKBest(score_func=chi2,k=4)
k_best=chi_best.fit(x,y)

np.set_printoptions(precision=3)
print(k_best.scores_)

k_features=k_best.transform(x)
print(k_features[0:5,:])
```

```
[ 111.52  1411.887   17.605   53.108 2175.565  127.669    5.393  181.304]
[[148.     0.    33.6   50. ]
 [ 85.     0.    26.6   31. ]
 [183.     0.    23.3   32. ]
 [ 89.    94.    28.1   21. ]
 [137.   168.    43.1   33. ]]
```

In [15]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
```

```
In [17]: import warnings
         warnings.filterwarnings('ignore')

In [19]: model_lr=LogisticRegression()
         recur_fe=RFE(model_lr)
         Feature=recur_fe.fit(x,y)
         print("Number of Features: %d" % (Feature.n_features_))
         print("Selected Features are: %s" % (Feature.support_))
         print("Feature Ranking is as follows: %s" % (Feature.ranking_))

Number of Features: 4
Selected Features are: [ True  True False False False  True  True False]
Feature Ranking is as follows: [1 1 3 4 5 1 1 2]

In [21]: from sklearn.linear_model import Ridge
         ridge_reg=Ridge(alpha=1.0)
         ridge_reg.fit(x,y)

Out[21]: ▾ Ridge
         Ridge()

In [23]: def  print_coefs(coef,names=None,sort=False):
             if names==None:
                 names=["x%s" % x for x in range(len(coef))]
             lst=zip(coef,names)
             if sort:
                 lst  =sorted(lst,key=lambda  x:-np.abs(x[0]))
             return " + ".join("%s * %s" %(round(coefs,3),name) for coefs,name in lst)

In [25]: print("Ridge  model:",print_coefs(ridge_reg.coef_))
```

OUTPUT:

```
Ridge model: 0.021 * x0 + 0.006 * x1 + -0.002 * x2 + 0.0 * x3 + -0.0 * x4 + 0.013
* x5 + 0.145 * x6 + 0.003 * x7
```

PERFORMANCE MATRIX

PROGRAM:

In [3]:
```
import pandas as pd
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

In [4]:
```
url = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']dataframe = pd.read_csv(url, names=names)
array = dataframe.values
```

In [5]:
```
X = array[:,0:8]
Y = array[:,8]
kfold = model_selection.KFold(n_splits=10,random_state=7,shuffle=True)model =
LogisticRegression(solver='liblinear')
```

In [6]:
```
print("Classification Accuracy")
results =
model_selection.cross_val_score(model,X,Y,cv=kfold,scoring='accuracy')
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
```

```
Classification
Accuracy: 0.771 (0.051)
```

In [7]:
```
print("Log Loss")
results   =   model_selection.cross_val_score(model,X,Y,cv=kfold,scoring='neg_log_loss')
print("Logloss: %.3f (%.3f)" % (results.mean(), results.std()))
```

```
Log Loss
Logloss: -0.494 (0.042)
```

In [8]:
```
print('Area Under ROC Drive')
results   =   model_selection.cross_val_score(model,X,Y,cv=kfold,scoring='roc_auc')
print("AUC: %.3f (%.3f)" % (results.mean(), results.std()))
```

```
Area Under ROC Drive
AUC: 0.826 (0.050)
```

In [9]:
```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

In [10]:
```
X = array[:,0:8]
Y = array[:,8]
X_train,X_test,Y_train,Y_test=model_selection.train_test_split(X,Y,test_siz
e
=0.33,random_state=7)
```

In [11]:
```
model =
LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
```

In [12]:

```
print('Confusion Matrix')
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

```
Confusion Matrix[[141  21]
                 [ 41  51]]
```

In [13]:

```
from sklearn.metrics import
classification_report print('Classification
Report')
report = classification_report(Y_test,
predicted) print(report)
```

OUTPUT:

```
Classification Report
                Precision     recall      f1-score   support

0.0             0.77          0.87        0.82       162
1.0             0.71          0.55        0.62       92

accuracy                                  0.76       254
macro avg       0.74          0.71        0.72       254
weighted avg    0.75          0.76        0.75       254
```

NAVIE BAYES CLASSIFICATION - GAUSSIANNB

PROGRAM:

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics  import  accuracy_score,confusion_matrix,classification_report
```

In [4]:
```
raw_data=pd.read_csv("titanic.csv")
raw_data.head()
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [5]:
```
raw_data.describe(include='all')
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Braund, Mr. Owen Harris	male	NaN	NaN	NaN	347082	NaN	B96 B98	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

In [9]:
```
print(raw_data.columns)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
       'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [10]
```
raw_data.columns = raw_data.columns.str.strip()
```

In [11
```
columns_to_drop = ['PassengerId', 'Name', 'SibSp', 'Parch', 'Ticket',
'Cabin', 'Embmbarked']
missing_columns = [col for col in columns_to_drop if col not in raw_data.columns]
print("Missing columns:", missing_columns)
```

In [12]:
```python
data = raw_data.drop(columns=['PassengerId', 'Name', 'SibSp', 'Parch', 'Ticket',
'Cabin', 'Embarked'], errors='ignore')
```

In [13]:
```python
data.head()
```

Out[13]:

	Survived	Pclass	Sex	Age	Fare
0	0	3	male	22.0	7.2500
1	1	1	female	38.0	71.2833
2	1	3	female	26.0	7.9250
3	1	1	female	35.0	53.1000
4	0		male	35.0	8.0500

In [15]:
```python
mv=data.isnull()
```

Out [15]:
```
Survived      0
Pclass        0
Sex           0
Age         177
Fare          0
dtype: int64
```

In [16]:
```python
data_no_mv=data.dropna(axis=0)
```

In [17]:
```python
data_no_mv.describe(include='all')
```

Out[17]:

	Survived	Pclass	Sex	Age	Fare
count	714.000000	714.000000	714	714.000000	714.000000
unique	NaN	NaN	2	NaN	NaN
top	NaN	NaN	male	NaN	NaN
freq	NaN	NaN	453	NaN	NaN
mean	0.406162	2.236695	NaN	29.699118	34.694514
std	0.491460	0.838250	NaN	14.526497	52.918930
min	0.000000	1.000000	NaN	0.420000	0.000000
25%	0.000000	1.000000	NaN	20.125000	8.050000
50%	0.000000	2.000000	NaN	28.000000	15.741700
75%	1.000000	3.000000	NaN	38.000000	33.375000
max	1.000000	3.000000	NaN	80.000000	512.329200

In [18]:
```python
data_with_dummies=pd.get_dummies(data_no_mv,drop_first=True)
```

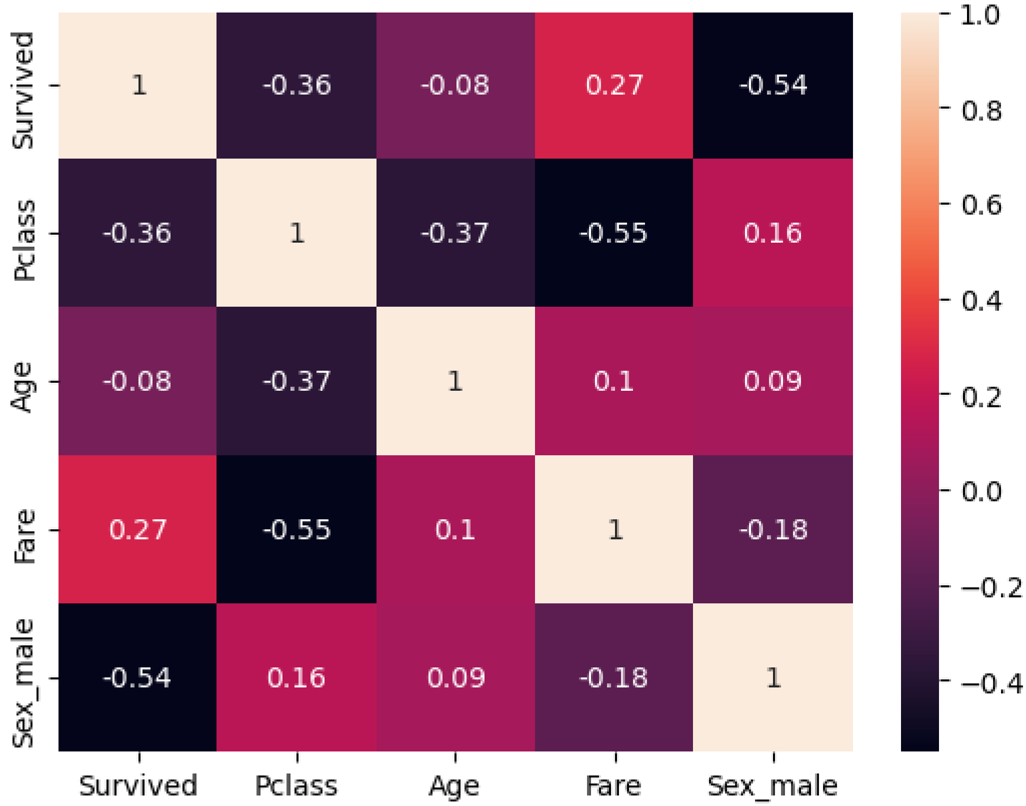
In [19]:
```python
data_with_dummies.head()
```

Out[19]:

	Survived	Pclass	Age	Fare	Sex_male
0	0	3	22.0	7.2500	True
1	1	1	38.0	71.2833	False
2	1	3	26.0	7.9250	False
3	1	1	35.0	53.1000	False
4	0	3	35.0	8.0500	True

In [20]:
```
corr_matrix=data_with_dummies.corr().round(2)
```

In [21]:
```
sns.heatmap(data=corr_matrix,annot=True)
plt.show()
```

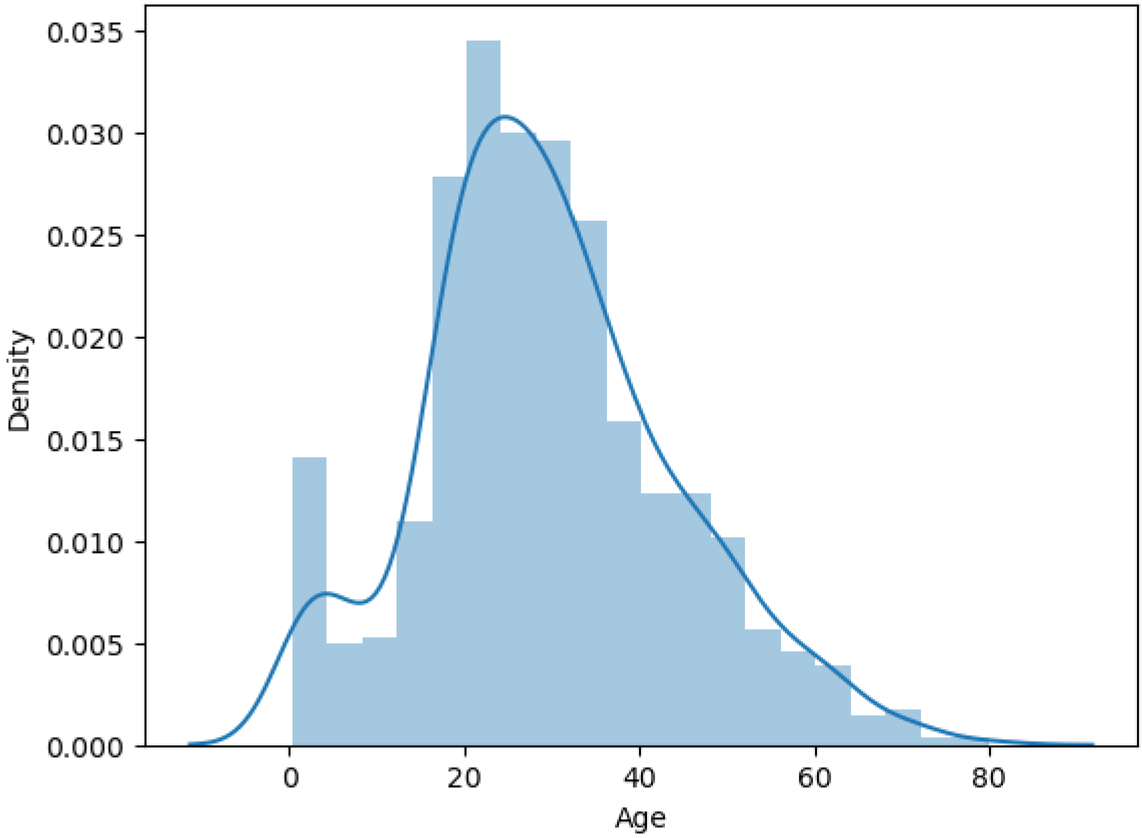


In [22]:
```
data_no_multicollinearity=data_with_dummies.drop('Fare',axis=1)
data_no_multicollinearity.head()
```

Out[22]:

	Survived	Pclass	Age	Sex_male
0	0	3	22.0	True
1	1	1	38.0	False
2	1	3	26.0	False
3	1	1	35.0	False
4	0	3	35.0	True

In [23]:
```
sns.distplot(data_no_multicollinearity['Age'])
plt.show()
```



In [24]:
```
features=data_no_multicollinearity.drop('Survived',axis=1)
label=data_no_multicollinearity['Survived']
```

In [26]:
```
X_train,X_test,y_train,y_test=train_test_split(features,label,test_size=0.2,random_
```

In [27]:
```
clf=GaussianNB()
clf.fit(X_train,y_train)
```

Out[27]:

GaussianNB

GaussianNB()

In [28]:
```
pred=clf.predict(X_test)
```

In [29]:
```
acc=accuracy_score(y_test,pred)
acc
```

Out[29]: 0.7692307692307693

In [30]:
```
matrix=pd.DataFrame(confusion_matrix(y_test,pred),columns=['Predicted  0', 'Predicted 1'])
```

In [31]:
```
matrix
```

OUTPUT:

Out [31]:

	Predicted 0	Predicted 1
0	73	14
1	19	37

In [32]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.84      0.82        87
           1       0.73      0.66      0.69        56

    accuracy                           0.77       143
   macro avg       0.76      0.75      0.75       143
weighted avg       0.77      0.77      0.77       143
```

NAVIE BAYES CLASSIFICATION- MULTINOMINALNB

PROGRAM:

In [1]:
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
```

In [2]:
```
data = {
'text': [
'Free money now',
'Call me now',
'Win a lottery',
'Important information regarding your account',
'Meeting at noon',
'Your invoice is attached',
'Lowest price in the market',
'Can you send me the report?'
],
'label': [
'spam',
'spam',
'spam',
'ham',
'ham',
'ham',
'spam',
'ham'
]
}
```

In [3]:
```
df = pd.DataFrame(data)
```

In [4]:
```
X = df['text']
y = df['label']
```

In [5]:
```
vectorizer = CountVectorizer()
X_vectorized = vectorizer.fit_transform(X)
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y, test_size=0.3,
```

In [7]:
```
model = MultinomialNB()
model.fit(X_train, y_train)
```

In [8]:
```
y_pred = model.predict(X_test)
```

In [9]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the Naïve Bayes classifier: {accuracy * 100:.2f}%')
```

```
Accuracy of the Naïve Bayes classifier: 66.67%
```

In [10]:
```
test_data = [
"Congratulations! You have won a gift card",
"Can we reschedule our meeting?",
"Get paid for taking surveys",
"Your appointment is confirmed",
"you have won a lottery",
"Congratulations! You've unlocked access to an exclusive AI course!",
"Can we schedule a call to discuss your AI project?",
"Get paid to test machine learning models—sign up now!",
"Your AI consultation is confirmed—see you soon!",
"You’ve been selected for a free machine learning webinar!",
]
```

In [11]:
```
test_vectorized = vectorizer.transform(test_data)
```

In [12]:
```
spam_predictions = model.predict(test_vectorized)
```

In [14]:
```
for text, prediction in zip(test_data, spam_predictions):
    print(f'Text: "{text}" | Prediction: {prediction}')
```

OUTPUT:

```
Text: "Congratulations! You have won a gift card" | Prediction: ham
Text: "Can we reschedule our meeting?" | Prediction: ham
Text: "Get paid for taking surveys" | Prediction: ham
Text: "Your appointment is confirmed" | Prediction:
ham Text: "you have won a lottery" | Prediction: ham
Text: "Congratulations! You've unlocked access to an exclusive AI course!" | Predict
ion: ham
Text: "Can we schedule a call to discuss your AI project?" | Prediction: ham
Text: "Get paid to test machine learning models—sign up now!" | Prediction:
ham Text: "Your AI consultation is confirmed—see you soon!" | Prediction: ham
Text: "You’ve been selected for a free machine learning webinar!" | Prediction:
ham
```

BAYESIAN NETWORK

PROGRAM:

In [3]:
```
import numpy as np
import csv
import pandas as pd
#python -m pip install pgmpy in jupyter terminal or anaconda terminal
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
```

In [5]:
```
#read Cleveland Heart Disease data
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
```

In [7]:
```
#display the data
print('Few examples from the dataset are given below')
print(heartDisease.head())
```

```
Few examples from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

In [21]:
```
#Model Bayesian Network
Model=BayesianNetwork([('age','trestbps'),('age','fbs'), ('sex','trestbps'),
('exang','trestbps'),('trestbps','target'),('fbs','target'),
('target','restecg'), ('target','thalach'),('target','chol')])
```

In [23]:
```
#Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
Model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
```

```
Learning CPD using Maximum likelihood estimators
```

In [25]:
```
# Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(Model)
```

```
Inferencing with Bayesian Network:
```

In [29]:
```
#computing the Probability of HeartDisease given Age
print('\n 1. Probability of HeartDisease given Age=29')
q=HeartDisease_infer.query(variables=['target'],evidence ={'age':29})
print(q)
```

```
 1. Probability of HeartDisease given Age=29
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.3872 |
+-----------+---------------+
| target(1) |        0.6128 |
+-----------+---------------+
```

In [31]:

```
#computing the Probability of HeartDisease given cholesterol
print('\n 2. Probability of HeartDisease given cholesterol=160')
q=HeartDisease_infer.query(variables=['target'],evidence ={'chol':160})
print(q)
```

```
 2. Probability of HeartDisease given cholesterol=160
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.0000 |
+-----------+---------------+
| target(1) |        1.0000 |
+-----------+---------------+
```

EM ALGORITHM

PROGRAM:

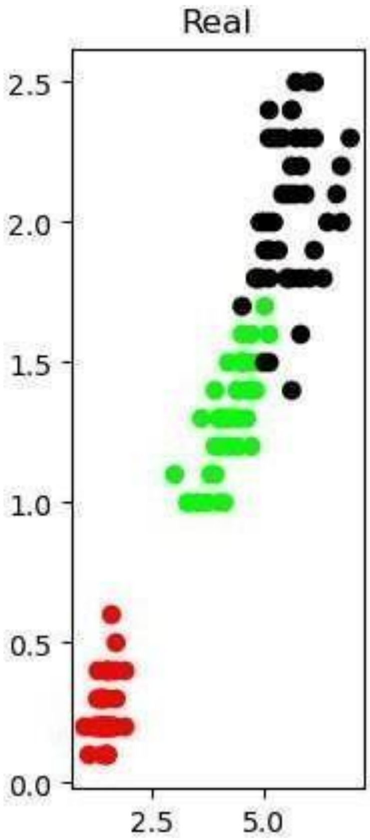
In [3]:
```
names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']
dataset = pd.read_csv("8-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}
y = [label[c] for c in dataset.iloc[:, -1]]
```

In [5]:
```
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
```

```
<Figure size 1400x700 with 0 Axes>
```

In [7]:
```
# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])
```

Out[7]:
```
<matplotlib.collections.PathCollection at 0x24ae3703f50>
```

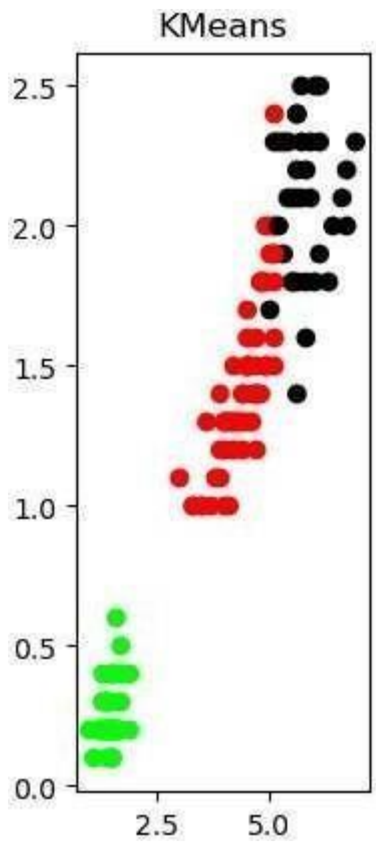


In [11]:
```
# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
```

```
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])
```

```
C:\Users\surya\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[11]: <matplotlib.collections.PathCollection at 0x24ae87b1fd0>



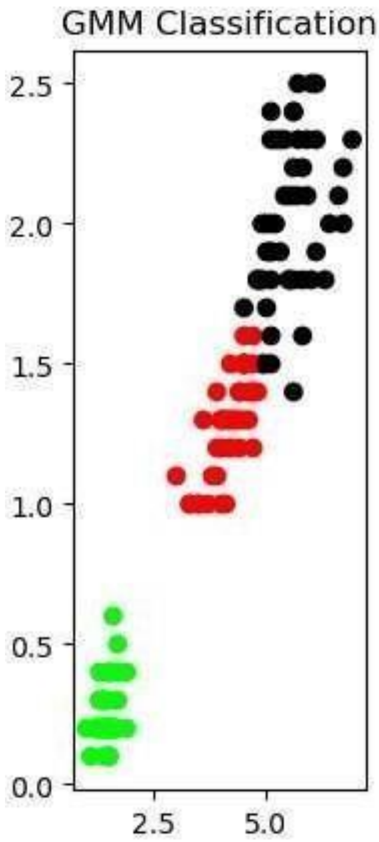
In [13]:
```
print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_)
```

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:
 [[ 0 50  0]
 [47  0  3]
 [14  0 36]]
```

In [15]:
```
# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])
```

```
C:\Users\surya\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[15]: `<matplotlib.collections.PathCollection at 0x24ae88eb110>`



In [17]:

```
print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

```
The accuracy score of EM:  0.3333333333333333
The Confusion matrix of EM:
  [[ 0 50  0]
 [45  0  5]
 [ 0  0 50]]
```

LOGISTIC REGRESSION – SPAM DETECTION

PROGRAM:

In [2]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,
accuracy_score,classification_report
# Load the dataset
data = pd.read_csv('spam.csv',
encoding='latin-1') # Display the first few
rows of the dataset print(data.head())
# Data preprocessing
# The dataset has columns 'v1' for labels and 'v2' for messages
data = data[['v1', 'v2']]
data.columns = ['label',
'message']
# Convert labels to binary: spam=1, ham=0
data['label'] = data['label'].map({'spam': 1, 'ham': 0})
# Split the dataset into features and target variable
X = data['message'] # Features
y = data['label'] # Target variable
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=20)
# Convert text data to numerical data using CountVectorizer
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
# Create a Logistic Regression model
model = LogisticRegression()
# Train the model
model.fit(X_train_vectorized,
y_train) # Make predictions
y_pred = model.predict(X_test_vectorized)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test,
y_pred) class_report =
classification_report(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification
Report:') print(class_report)
```

OUTPUT

```
      v1                                    v2 Unnamed: 2  \
0    ham  Go until jurong point, crazy.. Available only ...        NaN
1    ham      Ok lar... Joking wif u oni...                        NaN
2   spam  Free entry in 2 a wkly comp to win FA Cup fina...        NaN

3    ham  U dun say so early hor... U c already then say...        NaN
4    ham  Nah I don't think he goes to usf, he lives aro...        NaN
```

```
Accuracy: 0.98

Confusion Matrix:
[[ 968       2]
[ 21       124]]

Classi
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       970
           1       0.98      0.86      0.92       145

    accuracy                           0.98      1115
   macro avg       0.98      0.93      0.95      1115
weighted avg       0.98      0.98      0.98      1115
```

LOGISTIC REGRESSION - DIABETES

PROGRAM:

In [6]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
# Load the dataset
data = pd.read_csv('diabetes.csv')
# Display the first few rows of the dataset
print(data.head())
# Features and target variable
X = data.drop('Outcome', axis=1) # Features
y = data['Outcome'] # Target variable
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Create a Logistic Regression model
model = LogisticRegression()
# Train the model
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(class_report)
```

OUTPUT

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
Accuracy: 0.75


Confusion Matrix:
 [[87 14]
 [24 29]]


Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       101

           1       0.67      0.55      0.60        53

    accuracy                           0.75       154
   macro avg       0.73      0.70      0.71       154
weighted avg       0.75      0.75      0.75       154
```

```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```

ANN BACK PROPAGATION

PROGRAM:

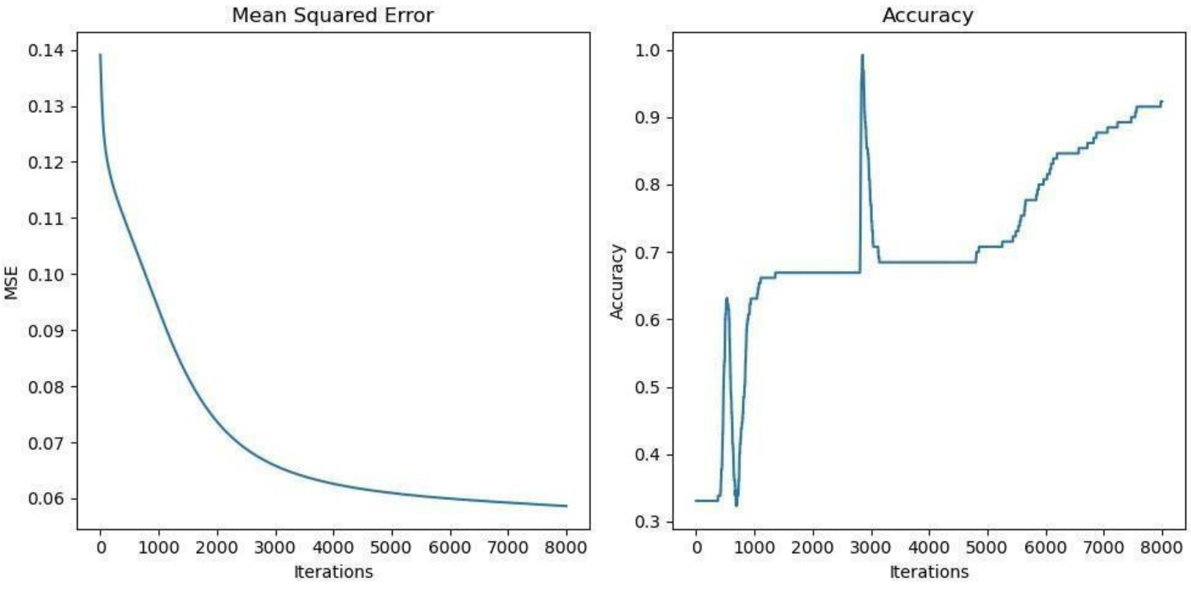
In [11]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load dataset
data = load_iris()

# Get features and target
X  =  data.datay
Y= data.target

# Prepare Dataset: Create dummy variables for class labels using get_dummies()
y = pd.get_dummies(y).values

# Split data into train and test data
x_train,x_test,y_train,y_test=train_test_split(X,  y,test_size=20,random_state=4)

# Initialize variables
learning_rate = 0.1
iterations=8000
N = y_train.size

# Number of input features
input_size = 4

# Number of hidden layers neurons
hidden_size = 2

# Number of neurons at the output layer
output_size = 3

results = pd.DataFrame(columns=["mse",  "accuracy"])

# Initialize weights
np.random.seed(10)

# Initializing weight for the hidden layer
W1  = np.random.normal(scale=0.5,  size=(input_size, hidden_size))

# Initializing weight for the output layer
W2 = np.random.normal(scale=0.5, size=(hidden_size, output_size))
```

```
# Helper functions
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred,y_true):
    return((y_pred-y_true)**2).sum()/(2*y_pred.size)

def accuracy(y_pred,y_true):
    acc=y_pred.argmax(axis=1)==y_true.argmax(axis=1)
    return acc.mean()

#Back propagation Neural Network
mse_list=[]
accuracy_list=[]

for itr in range(iterations):
    #Feed forward propagation
    #Hidden layer
    Z1=np.dot(x_train,W1)
    A1 = sigmoid(Z1)
    #Outputlayer
    Z2=np.dot(A1,2)
    A2 =sigmoid(Z2)

    #Calculating error
    Mse=mean_squared_error(A2,y_train)
    acc = accuracy(A2, y_train)
    mse_list.append(mse)
    accuracy_list.append(acc)

    #Error calculation
    error_output=A2-y_train
    dZ2 = error_output * A2 * (1-A2)
    error_hidden = np.dot(dZ2, W2.T)
    dZ1=error_hidden*A1*(1-A1)

    #Weight updates
    W2_update=np.dot(A1.T,dZ2)/N W1_update =
    np.dot(x_train.T,dZ1)/ N
    W2 = W2 - learning_rate * W2_update
    W1=W1-learning_rate*W1_update
```

```
#PlottingMeanSquaredErrorandAccuracy
plt.figure(figsize=(10,5)) plt.subplot(1, 2,
1) plt.plot(mse_list)
plt.title('Mean Squared Error')
plt.xlabel('Iterations')
plt.ylabel('MSE') plt.subplot(1,2,2)
plt.plot(accuracy_list)
plt.title('Accuracy')
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.tight_layout()
plt.show()
Z1 = np.dot(x_test, W1)
A1 = sigmoid(Z1)
Z2=np.dot(A1,W2)
A2=sigmoid(Z2)
test_acc=accuracy(A2,y_test)
print("TestAccuracy:{}".format(test_acc))
```

OUTPUT:



Test Accuracy: 0.9

K-NEAREST NEIGHBOR

PROGRAM:

In [1]:
```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In [3]:
```
data_set= pd.read_csv('user_data.csv')
```

In [5]:
```
x= data_set.iloc[:,
[2,3]].values y=
data_set.iloc[:, 4].values
```

In [7]:
```
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_state-0
```

In [9]:
```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [11]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

Out[11]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [13]:
```
y_pred= classifier.predict(x_test)
```

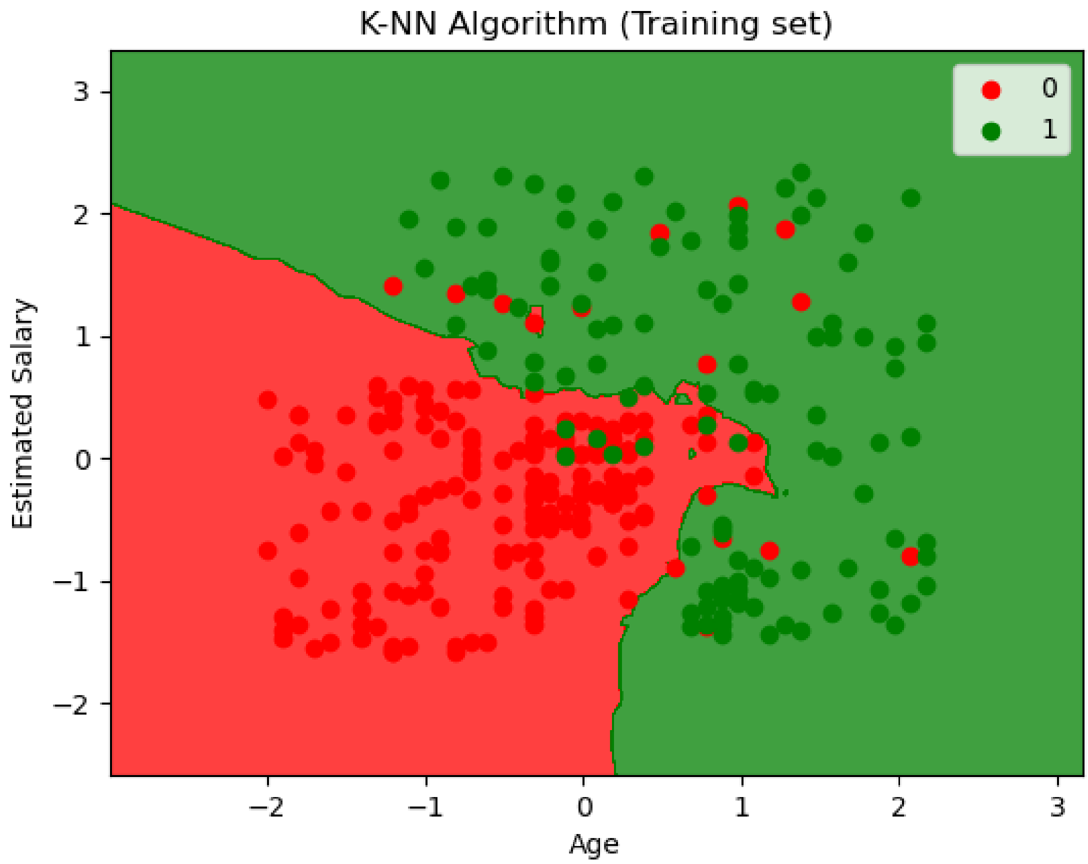
In [25]:
```
from sklearn.metrics import
confusion_matrix cm=
confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[64  4]
 [ 3 29]]
```

In [21]:

```
from      matplotlib.colors      import
ListedColormap  x_set,  y_set  =  x_train,
y_train
x1,  x2  =  nm.meshgrid(nm.arange(start  =  x_set[:,  0].min()  -  1,  stop  =
x_set[:, 0].max()+1,step=0.01),
nm.arange(start=x_set[:,1].min()-1,stop=x_set[:,1].max()+1,step=0.01)
mtp.contourf(x1,x2,classifier.predict(nm.array([x1.ravel(),x2.ravel()]).T).
reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red' ,'green' )))
mtp.xlim(x1.min(),x1.max())
mtp.ylim(x2.min(),x2.max())
for i, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set== j,0],x_set[y_set == j, 1],
c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

OUTPUT:


K-NN Algorithm (Training set)
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

DECISION TREE

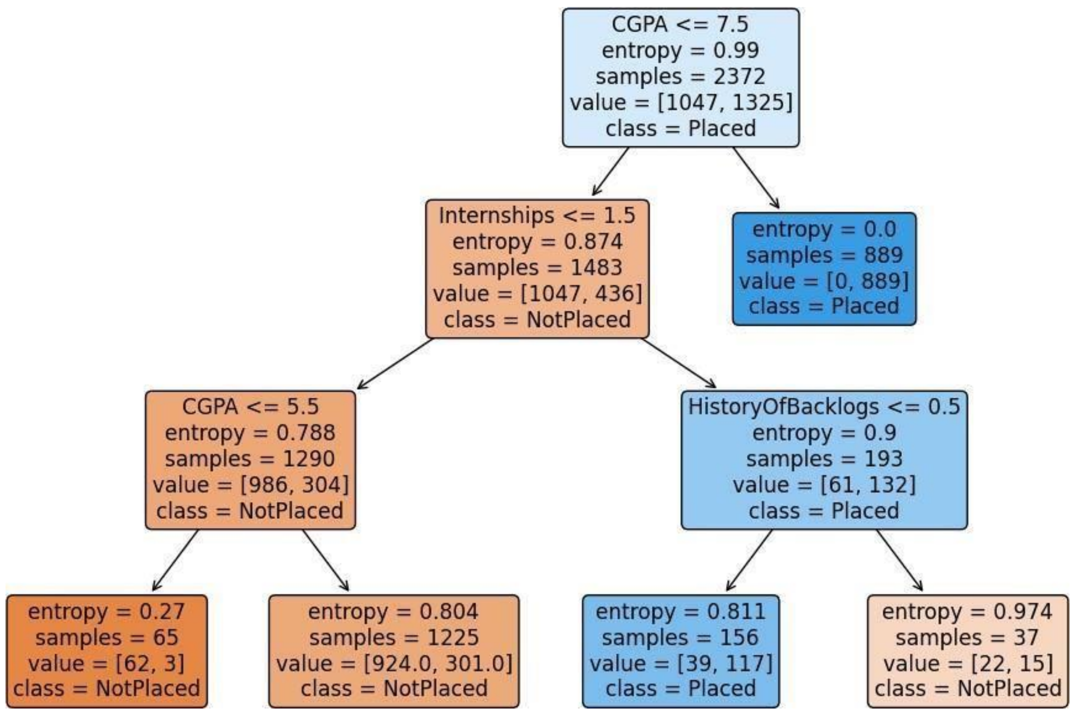
PROGRAM:

In [1]:

```
import pandas as pd
from  sklearn.tree  import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import tree
import matplotlib.pyplot as plt

data_set = pd.read_csv('collegePlace.csv') print("Data Set")
print(data_set.head())

# Features and target variable
#x = data_set.iloc[:, [3, 4, 6]].values
#y = data_set.iloc[:, 7].values

X  =  data_set[['Internships',  'CGPA',  'HistoryOfBacklogs']] # Feature columns
y  = data_set['PlacedOrNot'] # Target column

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize and train the decision tree classifier
clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Print test predictions and tree accuracy
print("Predictions on the test set:", y_pred)
print("Decision Tree Accuracy on the test set:", clf.score(X_test, y_test))

# Visualize the decision tree
plt.figure(figsize=(12,  8))
tree.plot_tree(clf, feature_names=X.columns, class_names=['NotPlaced','Placed'], fi
rounded=True)
#tree.plot_tree(clf,  filled=True)
plt.show()
```

OUTPUT:

```
Data Set
  Age Gender Stream  Internships CGPA Hostel \
0 22  Male   ECE          1         8    1
1 21  Female CSE          0         7    1
2 22  Female IT           1         6    0
3 21  Male   IT           0         8    0
4 22  Male   Mech         0         8    1

  HistoryOfBacklogs  PlacedOrNot
0           1          1
1           1          1
2           0          1
3           1          1
4           0          1
Predictions on the test set: [0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0
1 1 1 1 0 0
1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0 1 1 1 1
1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0
1 1 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 1 1 1 0 0 0 0 1
0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 1 0 0 1 0 1
0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0
0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 1 0 1
1 1 1 1 1 0 0 1 1 0 0 1 0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0
0 1 0 0 1 0 1 0 1 1 0 1 1 0 1 0 0 0 0 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0
0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0
1 1 0 0 0 0 1 0 0 1 0 1 0 1 1 1 0 1 0 1 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 1
0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1
1 1 1 1 1 0 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0
1 0 0 0 0 0 1 1 1 0 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1
1 1 1 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0
1 0]

Decision Tree Accuracy on the test set: 0.8451178451178452
```



SUPPORT VECTOR CLASSIFICATION

PROGRAM:

In [1]:

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC
```

In [2]:

```
#Load the dataset
cancer=load breast canser()
X=cancer.data[:, :2]
Y=cancer.targer
```

In [3]:

```
#Build the model
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
# Trained the model
svm.fit(X, y)
```

Out[3]:

```
SVC
SVC(gamma=0.5)
```

In [4]:

```
# Plot Decision Boundary
DecisionBoundaryDisplay.from_estimator(svm,X,
response_method="predict",
cmap=plt.cm.Spectral,
alpha=0.8,
xlabel=cancer.feature_names[0],
ylabel=cancer.feature_names[1],)
# Scatter plot
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolors="k")
plt.show()
```

OUTPUT;