6th semester BCA Machine learning lab manual

2)data exploration and preprocessing in ML

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

In [69]:

```
df = pd.read_csv("Lab1.csv")
df.head()
```

Out[69]:

	country	age	salary	purchased
0	France	NaN	7200	no
1	Spain	27.0	4800	yes
2	Germany	30.0	5400	yes
3	UK	49.0	98000	no

In [70]:

```
df.tail()
```

Out[70]:

	country	age	salary	purchased
0	France	NaN	7200	no
1	Spain	27.0	4800	yes
2	Germany	30.0	5400	yes
3	UK	49.0	98000	no

In [71]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
```

```
---  ------     --------------  -----
 0   country    4 non-null      object
 1   age        3 non-null      float64
 2   salary     4 non-null      int64
 3   purchased  4 non-null      object
dtypes: float64(1), int64(1), object(2)
memory usage: 256.0+ bytes
```

In [72]:

```
df.describe()
```

Out[72]:

	age	salary
count	3.000000	4.000000
mean	35.333333	28850.000000
std	11.930353	46111.278447
min	27.000000	4800.000000
25%	28.500000	5250.000000
50%	30.000000	6300.000000
75%	39.500000	29900.000000
max	49.000000	98000.000000

In [73]:

```
df.isnull().sum()
```

Out[73]:

```
country      0
age          1
salary       0
purchased    0
dtype: int64
```

In [74]:

```
df['age'].fillna(df['age'].mean(), inplace = True)
df['salary'].fillna(df['salary'].mean(), inplace=True)
```

In [75]:

```
df.isnull().sum()
```

Out[75]:

```
country      0
age          0
salary       0
purchased    0
dtype: int64
```

In [76]:

```
from sklearn.impute import SimpleImputer
x = df.iloc[:,:-1].values
x
```

Out[76]:

```
array([['France', 35.333333333333336, 7200],
       ['Spain', 27.0, 4800],
       ['Germany', 30.0, 5400],
       ['UK', 49.0, 98000]], dtype=object)
```

In [77]:

```
y = df.iloc[:,3: ].values
y
```

Out[77]:

```
array([['no'],
       ['yes'],
       ['yes'],
       ['no']], dtype=object)
```

In [78]:

```
imp = SimpleImputer(missing_values =np.nan, strategy = "mean")
x[:, 1:3] = imp.fit_transform(x[:, 1:3])
x
```

Out[78]:

```
array([['France', 35.333333333333336, 7200.0],
       ['Spain', 27.0, 4800.0],
       ['Germany', 30.0, 5400.0],
       ['UK', 49.0, 98000.0]], dtype=object)
```

In [79]:

```
from sklearn.preprocessing import LabelEncoder
```

In [80]:

```
le = LabelEncoder()
h = le.fit_transform(x[:,0])
h
```

Out[80]:

```
array([0, 2, 1, 3])
```

In [81]:

```
y = le.fit_transform(y)
y
```

```
C:\Users\25LAB-2BCA\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

Out[81]:

```
array([0, 1, 1, 0])
```

In [82]:

```
from sklearn.utils import column_or_1d
y = column_or_1d(y, warn = True)
y
```

Out[82]:

```
array([0, 1, 1, 0])
```

In [83]:

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

In [84]:

```
transform = ColumnTransformer([('norm1', OneHotEncoder(), [0])], remainder ="passthrough")
x = transform.fit_transform(x)
```

x

Out[84]:

```
array([[1.0, 0.0, 0.0, 0.0, 35.333333333333336, 7200.0],
       [0.0, 0.0, 1.0, 0.0, 27.0, 4800.0],
       [0.0, 1.0, 0.0, 0.0, 30.0, 5400.0],
       [0.0, 0.0, 0.0, 1.0, 49.0, 98000.0]], dtype=object)
```

In [85]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

In [86]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [87]:

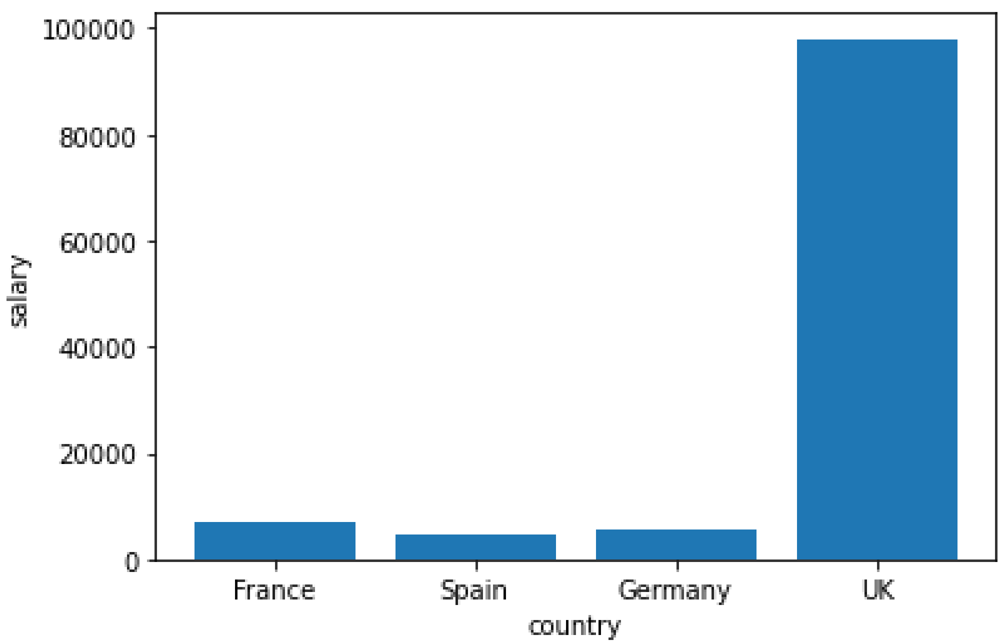
```
x_train[:,4:6] = sc.fit_transform(x_train[:, 4:6])
x_train
```

Out[87]:

```
array([[0.0, 0.0, 0.0, 1.0, 1.3109359202840398, 1.4138527953056175],
       [0.0, 0.0, 1.0, 0.0, -1.1149081191200718, -0.7345887349522664],
       [1.0, 0.0, 0.0, 0.0, -0.1960278011639687, -0.679264060353351]],
      dtype=object)
```

In [91]:

```
plt.bar(df['country '],df['salary'])
plt.xlabel('country')
plt.ylabel('salary')
plt.show()
```

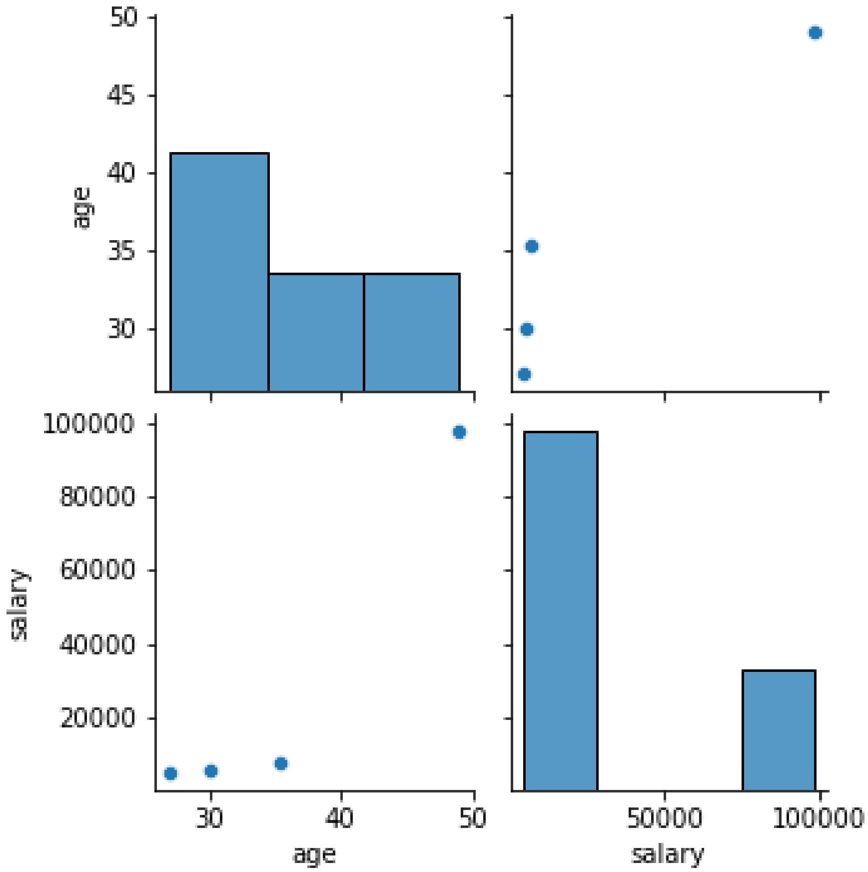


In [92]:

```
import seaborn as sns
sns.pairplot(df)
```

Out[92]:

<seaborn.axisgrid.PairGrid at 0x18b7dac21f0>



3) evaluate classifier using performance measures

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import
accuracy_score,precision_score,recall_score,f1_score,roc_auc_score,confusio
n_matrix,classification_report
iris=load_iris()
x=iris.data
y=iris.target
```

In [2]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_sta
te=42)
log_reg=LogisticRegression(max_iter=1000)
```

In [3]:

```
log_reg.fit(x_train,y_train)
y_pred=log_reg.predict(x_test)
accuracy=round(accuracy_score(y_test,y_pred)*100,2)
print("Accuracy:",accuracy)
```

```
Accuracy: 100.0
```

In [4]:

```
precision=precision_score(y_test,y_pred,average="weighted")
print("Precision:",precision)
```

```
Precision: 1.0
```

In [5]:

```
recall=recall_score(y_test,y_pred,average="weighted")
print("Recall:",recall)
```

```
Recall: 1.0
```

In [6]:

```
f1=f1_score(y_test,y_pred,average="weighted")
print("f1score:",f1)
```

```
f1score: 1.0
```

In [7]:

```
roc_auc=roc_auc_score(y_test,log_reg.predict_proba(x_test),multi_class='ovr
')
print("ROC AUC score:",roc_auc)
```

```
ROC AUC score: 1.0
```

In [8]:

```
conf_matrix=confusion_matrix(y_test,y_pred)
print("Confusion matrix:",conf_matrix)
```

```
Confusion matrix: [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

In [9]:

```
result1=classification_report(y_test,y_pred)
print("Classification report:",result1)
```

```
Classification report:               precision    recall  f1-score   suppor
t

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
4)k nearest neighbour classification using python
import pandas as pd
import numpy as np
df=pd.read_csv("heart.csv")
df.head()
```

Out[1]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	60	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	35	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
3	55	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	56	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [2]:

```
df.describe()
```

Out[2]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
count	289.000000	289.000000	289.000000	289.000000	289.000000	289.000000	289.000000	289.000000	289.000000	289.000000	289.000000	289.000000	289.000000	289.000000
mean	54.010381	0.678201	1.020761	131.377163	247.961938	0.145329	0.515571	150.231834	0.318339	1.007612	1.418685	0.712803	2.314879	0.570934
std	9.132316	0.467977	1.027192	17.518432	51.596208	0.353043	0.514309	22.899650	0.466640	1.133491	0.613333	1.022596	0.596128	0.495801
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.000000	0.000000	0.000000	120.000000	212.000000	0.000000	0.000000	136.000000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	54.000000	1.000000	1.000000	130.000000	243.000000	0.000000	1.000000	154.000000	0.000000	0.600000	1.000000	0.000000	2.000000	1.000000
75%	60.000000	1.000000	2.000000	140.000000	276.000000	0.000000	1.000000	168.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 289 entries, 0 to 288
```

```
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       289 non-null    int64  
 1   sex       289 non-null    int64  
 2   cp        289 non-null    int64  
 3   trtbps    289 non-null    int64  
 4   chol      289 non-null    int64  
 5   fbs       289 non-null    int64  
 6   restecg   289 non-null    int64  
 7   thalachh  289 non-null    int64  
 8   exng      289 non-null    int64  
 9   oldpeak   289 non-null    float64
 10  slp       289 non-null    int64  
 11  caa       289 non-null    int64  
 12  thall     289 non-null    int64  
 13  output    289 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 31.7 KB
```

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

In [5]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [6]:

```
y=df.iloc[:,13]
x=df.iloc[:,:-1]
```

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=0.2,random_sta
te=0)
```

In [8]:

```
x_train_scaled=scaler.fit_transform(x_train)
x_test_scaled=scaler.fit_transform(x_test)
```

In [9]:

```
from sklearn.neighbors import KNeighborsClassifier
lr=KNeighborsClassifier(n_neighbors=5)
lr.fit(x_train_scaled,y_train)
```

```
y_pred=lr.predict(x_test_scaled)
```

In [10]:

```
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score
result=confusion_matrix(y_test,y_pred)
print("Confusion Matrix: ")
print(result)
```

```
Confusion Matrix:
[[19  8]
 [ 5 26]]
```

In [11]:

```
result1=classification_report(y_test,y_pred)
print("Classification Report :",)
print(result1)
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.70      0.75        27
           1       0.76      0.84      0.80        31

    accuracy                           0.78        58
   macro avg       0.78      0.77      0.77        58
weighted avg       0.78      0.78      0.77        58
```

In [12]:

```
result2=round(accuracy_score(y_test,y_pred)*100,2)
print("Acccuracy: ",result2)
```

```
Acccuracy:  77.59
```

```
5) dimensionality reduction using pca
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

In [3]:

```
iris = load_iris()
x = iris.data
y = iris.target
print(x)
print(y)
s = x.shape
print(s)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
```

```
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
(150, 4)
```

In [4]:

```
target_names = iris.target_names
print(target_names)
```

```
['setosa' 'versicolor' 'virginica']
```

In [5]:

```
scaler =StandardScaler()
x_scaled =scaler.fit_transform(x)
print(x_scaled)
```

```
[[-9.00681170e-01  1.01900435e+00 -1.34022653e+00 -1.31544430e+00]
 [-1.14301691e+00 -1.31979479e-01 -1.34022653e+00 -1.31544430e+00]
 [-1.38535265e+00  3.28414053e-01 -1.39706395e+00 -1.31544430e+00]
 [-1.50652052e+00  9.82172869e-02 -1.28338910e+00 -1.31544430e+00]
 [-1.02184904e+00  1.24920112e+00 -1.34022653e+00 -1.31544430e+00]
 [-5.37177559e-01  1.93979142e+00 -1.16971425e+00 -1.05217993e+00]
```

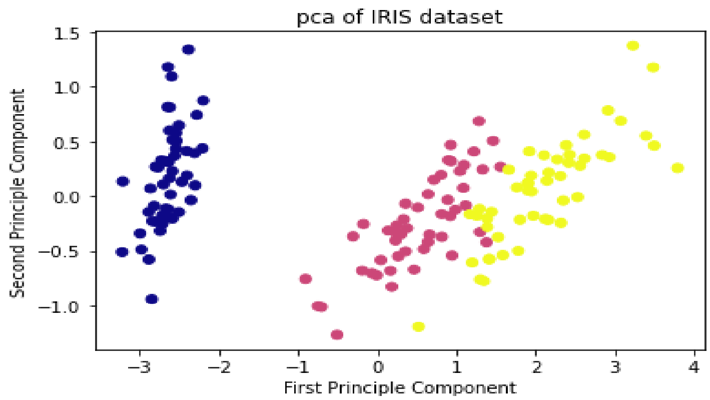
In [6]:

```
pca = PCA(n_components = 2)
x_pca = pca.fit_transform(x)
print(x_pca)
print(x_pca.shape)
```

```
[[-2.68412563  0.31939725]
 [-2.71414169 -0.17700123]
 [-2.88899057 -0.14494943]
 [-2.74534286 -0.31829898]
 [-2.72871654  0.32675451]
 [-2.28085963  0.74133045]
 [-2.82053775 -0.08946138]
(150, 2)
```

In [7]:

```
plt.scatter(x_pca[:, 0], x_pca[:, 1], c=y, cmap='plasma')
plt.xlabel('First Principle Component')
plt.ylabel('Second Principle Component')
plt.title('pca of IRIS dataset')
plt.show()
```



6) regression analysis using linear regression

```
#linear regression
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.datasets import load_iris
```

In [3]:

```
iris = load_iris()
```

In [4]:

```
X = iris.data
```

In [5]:

```
y = iris.target
```

In [6]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7,
test_size = 0.3, random_state=100)
```

In [14]:

```
X_train
```

Out[14]:

```
[array([[4.6, 3.4, 1.4, 0.3],
        [5. , 3. , 1.6, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [5.8, 2.6, 4. , 1.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.1, 3.3, 1.7, 0.5],
        [5. , 3.2, 1.2, 0.2],
from sklearn.linear_model
import LinearRegression
lr = LinearRegression()
```

In [18]:

```
lr.fit(X_train,y_train)
```

Out[18]:

```
LinearRegression()
```

In [19]:

```
y_pred = lr.predict(X_test)
```

In [20]:

```
mse = mean_squared_error(y_test, y_pred)
```

In [21]:

```
y_pred = lr.predict(X_test)
```

In [22]:

```
mse = mean_squared_error(y_test, y_pred)
print(mse)
```

```
0.035170909783059694
```

r2 = r2_score(y_test, y_pred) print(r2)

In [23]:

```
mae = mean_absolute_error(y_test, y_pred)
print(mae)
```

```
0.13682917742580056
```

7) classification using logistic regression

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import
accuracy_score,precision_score,recall_score,f1_score,roc_auc_score,confusio
n_matrix,classification_report
iris=load_iris()
x=iris.data
y=iris.target
```

In [2]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_sta
te=42)
log_reg=LogisticRegression(max_iter=1000)
```

In [3]:

```
log_reg.fit(x_train,y_train)
y_pred=log_reg.predict(x_test)
accuracy=round(accuracy_score(y_test,y_pred)*100,2)
print("Accuracy:",accuracy)
```

```
Accuracy: 100.0
```

In [4]:

```
precision=precision_score(y_test,y_pred,average="weighted")
print("Precision:",precision)
```

```
Precision: 1.0
```

In [5]:

```
recall=recall_score(y_test,y_pred,average="weighted")
print("Recall:",recall)
```

```
Recall: 1.0
```

In [6]:

```
f1=f1_score(y_test,y_pred,average="weighted")
print("f1score:",f1)
```

```
f1score: 1.0
```

In [7]:

```
roc_auc=roc_auc_score(y_test,log_reg.predict_proba(x_test),multi_class='ovr
')
```

```
print("ROC AUC score:",roc_auc)
```

```
ROC AUC score: 1.0
```

In [8]:

```
Confusion _matrix=confusion_matrix(y_test,y_pred)
print("Confusion matrix:",conf_matrix)
```

```
Confusion matrix: [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

In [9]:

```
result1=classification_report(y_test,y_pred)
print("Classification report:",result1)
```

```
Classification report:               precision    recall  f1-score   suppor
t

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

8) DECISION TREE

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [2]:

```
from sklearn.datasets import load_iris
iris = load_iris()
x = iris.data
y = iris.target
```

In [3]:

```
dtc = DecisionTreeClassifier()
```

In [6]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=1)
dtc.fit(x_train, y_train)
```

Out[6]:

```
DecisionTreeClassifier()
```

In [7]:

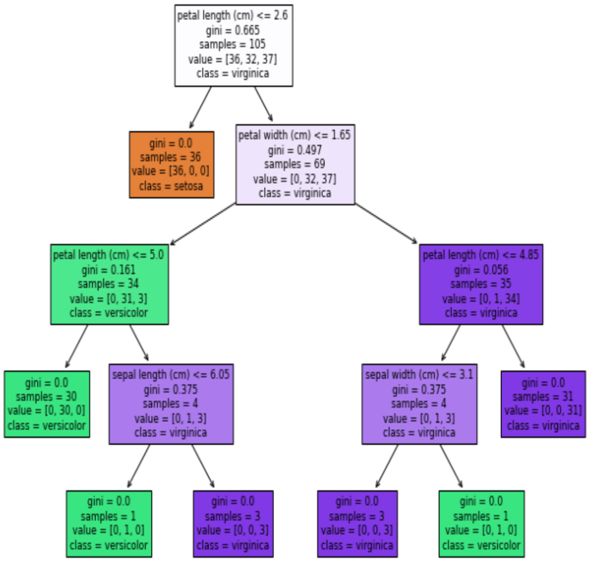
```
iris.target_names
```

Out[7]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [10]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
plot_tree(dtc, feature_names = iris.feature_names, class_names =
list(iris.target_names), filled=True)
plt.show()
```



9) SVM support vector machine

```
#classification using SVM
import pandas as pd
import numpy as np
dataset = pd.read_csv('titanic2.csv')
print(dataset.shape)
```

```
(891, 12)
```

In [2]:

```
dataset.head()
```

Out[2]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
0	1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	1
2	3	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	1

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	1
4	5	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	0

In [3]:

```
from sklearn.impute import SimpleImputer
dataset.isnull().sum()
```

Out[3]:

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
Survived         0
dtype: int64
```

In [4]:

```
dataset = dataset.drop(columns=['Name', 'Cabin', 'Embarked'])
```

In [5]:

```
dataset.head()
```

Out[5]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Survived
0	1	3	male	22.0	1	0	A/5 21171	7.2500	0
1	2	1	female	38.0	1	0	PC 17599	71.2833	1
2	3	3	female	26.0	0	0	STON/O2. 3101282	7.9250	1
3	4	1	female	35.0	1	0	113803	53.1000	1

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Survived
4	5	3	male	35.0	0	0	373450	8.0500	0

In [6]:

```
dataset.isnull().sum()
```

Out[6]:

```
PassengerId      0
Pclass           0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Survived         0
dtype: int64
```

In [7]:

```
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
x = dataset.iloc[:,:-1].values

y = dataset.iloc[:,8].values
x
y
```

Out[7]:

```
array([0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

In [8]:

```
x[:,3:4]=imp.fit_transform(x[:,3:4])
```

In [9]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
x[:,6] = le.fit_transform(x[:,6])
x[:,2] = le.fit_transform(x[:,2])
x
```

Out[9]:

```
array([[1, 3, 1, ..., 0, 523, 7.25],
       [2, 1, 0, ..., 0, 596, 71.2833],
       [3, 3, 0, ..., 0, 669, 7.925],
       ...,
       [889, 3, 0, ..., 2, 675, 23.45],
       [890, 1, 1, ..., 0, 8, 30.0],
       [891, 3, 1, ..., 0, 466, 7.75]], dtype=object)
```

In [10]:

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
transform = ColumnTransformer([('norm1',
OneHotEncoder(),[2])],remainder='passthrough')

x =transform.fit_transform(x)
print(x)
```

```
[[0.0 1.0 1 ... 0 523 7.25]
 [1.0 0.0 2 ... 0 596 71.2833]
 [1.0 0.0 3 ... 0 669 7.925]
 ...
 [1.0 0.0 889 ... 2 675 23.45]
 [0.0 1.0 890 ... 0 8 30.0]
 [0.0 1.0 891 ... 0 466 7.75]]
```

In [11]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,
random_state=0)
```

In [12]:

```
from sklearn import svm
lr = svm.SVC()
lr.fit(x_train,y_train)
y_pred = lr.predict(x_test)
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix : ")
print(result)
```

```
Confusion Matrix :
[[102   8]
 [ 50  19]]
```

In [ ]:

```
10)NAVIE BAYES
```

```
# classification techniques - Naive Bayes
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

In [2]:

```
from sklearn.metrics import accuracy_score
df = pd.read_csv("heart.csv")
df.head()
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

y = df.iloc[:,13]
x = df.iloc[:,:-1]

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,
random_state=0)

x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

In [7]:

```python
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train_scaled, y_train)
y_pred = gnb.predict(x_test_scaled)
```

In [8]:

```python
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
result2 = round(accuracy_score(y_test, y_pred)*100,2)
print("Accuracy : ",result2)
```

```
Accuracy :  79.31
```

In [9]:

```python
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix : ")
print(result)
```

```
Confusion Matrix :
[[32 11]
 [ 7 37]]
```

In [10]:

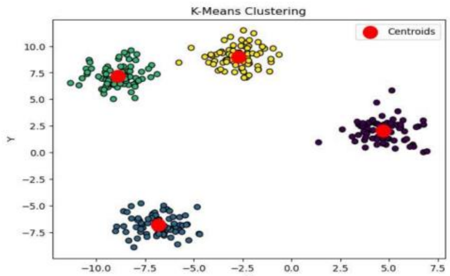
```python
result1 = classification_report(y_t
                                est,y_pred)
print("Classification Report : ")
print(result1)
```

```
Classification Report :
              precision    recall  f1-score   support

           0       0.82      0.74      0.78        43
           1       0.77      0.84      0.80        44

    accuracy                           0.79        87
   macro avg       0.80      0.79      0.79        87
weighted avg       0.80      0.79      0.79        87
```

11) K_MEANS CLUSTERING

```python
import matplotlib.pyplot as plt
 from sklearn.datasets
import make_blobs
from sklearn.cluster
import KMeans
# Generating synthetic data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=1.0, random_state=42)
# Initialize K-Means with the number of clusters
kmeans = KMeans(n_clusters=4)
# Fit the K-Means model to the data
kmeans.fit(X)
# Predict cluster
labels cluster_labels = kmeans.predict(X)
# Visualize the clusters
plt.figure(figsize=(7,5))
 plt.scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis', edgecolors='k')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
marker='o', s=200, color='red', label='Centroids')
plt.title('K-Means Clustering')
```

```
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```



12) baaging

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# Load the Iris dataset
iris=load_iris()
X=iris.data
y=iris.target
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

```
from sklearn.ensemble import RandomForestClassifier

dt= RandomForestClassifier(n_estimators= 10, criterion="entropy")

dt.fit(X_train,y_train)

y_pred=dt.predict(X_test)

from sklearn.metrics import classification_report,
confusion_matrix,accuracy_score

result1 = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(result1)

result2 = classification_report(y_test, y_pred)

print("Classification Report:",)

print (result2)

result3 = accuracy_score(y_test,y_pred)

print("Accuracy:",result3)
```

13. Ensemble method- Boosting

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris=load_iris()
X=iris.data
y=iris.target



# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

from sklearn.ensemble import GradientBoostingClassifier
gb=GradientBoostingClassifier()

gb.fit(x_train,y_train)



y_pred=gb.predict(x_test)



from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
```

```
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)


result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
```

14. Ensemble method- Stacking

```
#Stacking
from sklearn.datasets import load_iris
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


# Load the Iris dataset
iris=load_iris()
X=iris.data
y=iris.target
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)


# Define base learners
base_learners = [
        ('decision_tree', DecisionTreeClassifier(max_depth=1)),
    ('lr', LogisticRegression())  ]


# Define the meta-learner
meta_learner = SVC(probability=True, random_state=42)


# Initialize the Stacking Classifier with the base learners and the meta-learner
stack_clf = StackingClassifier(estimators=base_learners,
final_estimator=meta_learner)


# Train the stacking classifier
stack_clf.fit(X_train, y_train)
# Make predictions on the test set
```

```
y_pred = stack_clf.predict(X_test)



# Evaluate and print the accuracy of the model
print("Stacking Model Accuracy:", accuracy_score(y_test, y_pred))



from sklearn.metrics import classification_report, confusion_matrix,
Result2 = confusion_matrix(y_test, y_pred)




print("Confusion Matrix:")
print(result2)

Result3 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result3)
```