```
import pandas as pd
import numpy as np


import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.compose import ColumnTransformer


!wget -O titanic_train.csv https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv
```

```
--2023-06-25 06:29:42--  https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 60302 (59K) [text/plain]
Saving to: ‘titanic_train.csv’

titanic_train.csv   100%[===================>]  58.89K  --.-KB/s    in 0.008s

2023-06-25 06:29:42 (7.30 MB/s) - ‘titanic_train.csv’ saved [60302/60302]
```

```
df=pd.read_csv('titanic_train.csv',usecols=['Age','Fare','Survived'])


df.head()
```

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

```
df.dropna(inplace=True)


df.shape
```

```
(714, 3)
```

```
x=df.iloc[:,1:]
y=df.iloc[:,0]


x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)


y_train.head()
```

```
328    1
73     0
253    0
719    0
666    0
Name: Survived, dtype: int64
```

```
clf=DecisionTreeClassifier()


clf.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred=clf.predict(x_test)


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

```
0.6293706293706294
```

```
kbin_age=KBinsDiscretizer(n_bins=25,encode='onehot',strategy="quantile")
kbin_fare=KBinsDiscretizer(n_bins=25,encode='onehot',strategy="quantile")
```

```
trf=ColumnTransformer([('first',kbin_age,[0]),('second',kbin_fare,[0])])
```

```
x_train_trf=trf.fit_transform(x_train)
x_test_trf=trf.transform(x_test)
```

```
trf.named_transformers_['first'].bin_edges_
```

```
array([array([ 0.42,  3.  ,  8.6 , 16.  , 18.  , 19.  , 20.9 , 22.  , 23.  ,
              24.  , 25.  , 27.  , 28.  , 29.  , 30.  , 32.  , 33.  , 35.  ,
              36.2 , 39.  , 42.  , 45.  , 48.  , 51.4 , 59.2 , 80.  ])      ],
      dtype=object)
```

```
output=pd.DataFrame({'age':x_train['Age'],'age_trf':x_train_trf[:,0],'fare':x_train['Fare'],'fare_trf':x_train_trf[:,1]})
```

```
output['age_labels']=pd.cut(x=x_train['Age'],bins=trf.named_transformers_['first'].bin_edges_[0].tolist())
output['fare_labels']=pd.cut(x=x_train['Fare'],bins=trf.named_transformers_['second'].bin_edges_[0].tolist())
```

```
output.sample(5)
```

	age	age_trf	fare	fare_trf	age_labels	fare_labels
607	27.0		30.5000		(25.0, 27.0]	(30.0, 32.0]
103	33.0		8.6542		(32.0, 33.0]	(8.6, 16.0]
462	47.0		38.5000		(45.0, 48.0]	(36.2, 39.0]
765	51.0		77.9583		(48.0, 51.4]	(59.2, 80.0]
869	4.0		11.1333	(0, 0)\t1.0	(3.0, 8.6]	(8.6, 16.0]

```
clf=DecisionTreeClassifier()
clf.fit(x_train_trf,y_train)
y_pred2=clf.predict(x_test_trf)
```

```
accuracy_score(y_test,y_pred2)
```

```
0.6013986013986014
```

Write a classification program for implementingSVM using MNIST dataset

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

```
digits = datasets.load_digits()
```

```
X_train, X_test, y_train, y_test=train_test_split(digits.data, digits.target, test_size=0.2, random_state=42)
```

```
svm_classifier = SVC()
```

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

```
▾ SVC
SVC()
```

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

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9861111111111112
```

Write a classification program for implementing Naïve Bayes algorithm using iris dataset

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


iris=load_iris()
X_train, X_test, y_train, y_test=train_test_split(iris.data,iris.target,test_size=0.2,random_state=42)


nb_classifier = GaussianNB()


nb_classifier.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

Write a classification program for implementing decision tree using pima-indians-diabetes datset

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score


diabetes = load_diabetes()
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.target, test_size=0.2, random_state=42)
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)
y_pred = dt_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.0
```

Write a classification program for implementing kNN

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


iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.33, random_state=42)
knn_classifier = KNeighborsClassifier(n_neighbors=10)
knn_classifier.fit(X_train, y_train)
y_pred = knn_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.98
```

Write a clustering program for implementing k Means , k-medoids and Hierarchical Clustering using Wisconsin Breast Cancer Dataset

```
!pip install scikit-learn-extra
```

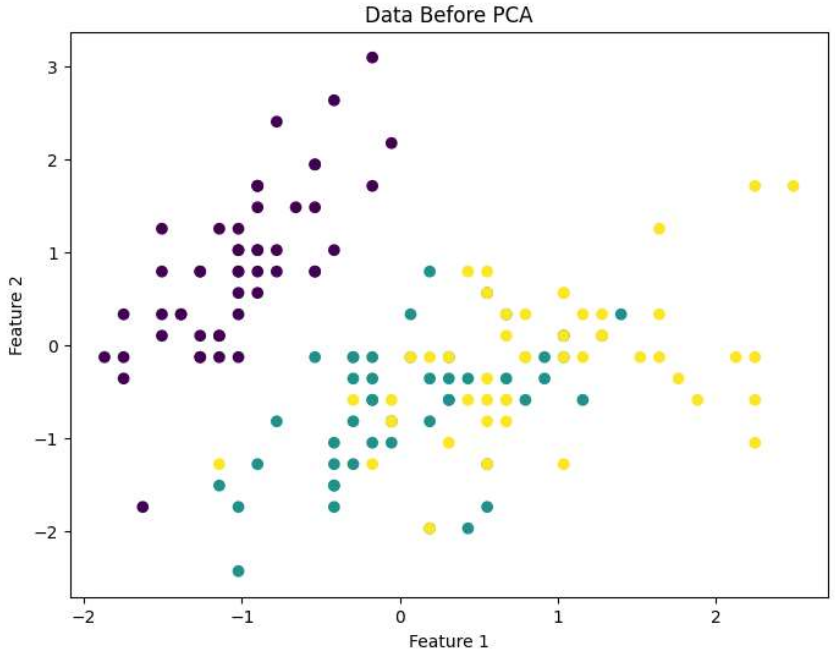
```
Collecting scikit-learn-extra
  Downloading scikit_learn_extra-0.3.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 23.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.22.4)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.10.1)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-learn-ex
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-l
Installing collected packages: scikit-learn-extra
Successfully installed scikit-learn-extra-0.3.0
```

```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn_extra.cluster import KMedoids



cancer = load_breast_cancer()
# Standardize the dataset
scaler = StandardScaler()
X = scaler.fit_transform(cancer.data)
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(X)
# Perform k-medoids clustering
kmedoids = KMedoids(n_clusters=2, random_state=42)
kmedoids_labels = kmedoids.fit_predict(X)
# Perform hierarchical clustering
hierarchical = AgglomerativeClustering(n_clusters=2)
hierarchical_labels = hierarchical.fit_predict(X)
print("K-Means Clustering:")
print(kmeans_labels)
print("K-Medoids Clustering:")
print(kmedoids_labels)
print("Hierarchical Clustering:")
print(hierarchical_labels)
```

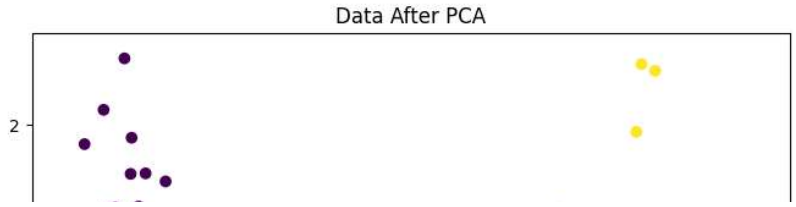
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fr
  warnings.warn(
K-Means Clustering:
[0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 0 1
 1 0 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1
 1 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1
 1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 1 1 1 1 1
 1 1 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0
 0 0 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1
 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 0 1 0 0 1 1 1 1 0 1 1 0 1 1 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 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 0 0 0 0 0 0 1]
K-Medoids Clustering:
[0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 0 1
 1 0 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1
 1 0 1 1 1 1 0 0 0 1 0 0 1 1 1 1 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1
 1 1 1 1 0 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 1 1 0 0 0 1 1
 1 0 1 1 1 1 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 1 0 0 0
 0 0 1 0 1 0 0 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1
 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 0 1 0 0 1 1 1 1 0 1 1 0 1 1 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 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 0 0 0 0 0 0 1]
Hierarchical Clustering:
[0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 0 1 1 0 1 0 0 0 1
 1 0 1 0 0 1 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1
 1 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1
 1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 0 0 1 1 1
 1 0 1 1 1 0 1 1 0 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 0 1 1
 1 1 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0
 0 0 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 0 1 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

Write a program to implement PCA

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt


iris = load_iris()


scaler = StandardScaler()
X = scaler.fit_transform(iris.data)


pca = PCA(n_components=2)  # Specify the number of components to keep
X_pca = pca.fit_transform(X)


# Visualize the data before PCA
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=iris.target)
plt.title("Data Before PCA")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```




```
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=iris.target)
plt.title("Data After PCA")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



Write a program to evaluate Classification Model using different Evaluation Metrics

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
```

```
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.33, random_state=42)

knn_classifier = KNeighborsClassifier(n_neighbors=10)
knn_classifier.fit(X_train, y_train)
y_pred = knn_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.98
Precision: 0.98125
Recall: 0.98
F1 Score: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```