untitled0-1

April 26, 2024

[109]:
```python
from IPython.display import Image

Image('/home/MLlogo.png')
```

[109]:

VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

[76]:
```python
from IPython.display import Image

Image('/home/ML.png')
```

[76]:

Name : Naman Rath

Registration Number : 21BIT0430

Course Name : Machine Learning

Faculty Name : Prof. Valarmathi B

Digital Assessment - 1

[108]:
```
## Machine Learning
## (BITE410L)
## Slot:A2+TA2
## Faculty Name:Prof. VALARMATHI B
## Digital Assignment 1
## Submitted By: Naman Rath (21BIT0430)
```

[77]:
```
from IPython.display import Image

Image('/home/ML2.png')
```

[77]:

Choose any dataset from UCI Repository / Kaggle / Other Repository and apply the following:

1. a data pre-processing technique
2. a classification & a clustering algorithms using python and document the results with appropriate screenshots.
3. data visualization techniques on a bench-marking dataset (**Basic Plots :** Line Graph, Scatter Plot, Bar Chart For Numerical Variable, Bar Chart For Categorical Variable, **Distribution Plots:** Boxplots and Histograms **Heat maps:** Visualizing Correlations and Missing Values) and document the results with appropriate screenshots.
4. Calculation of accuracy and error / Purity of cluster

[78]:
```
from IPython.display import Image

Image('/home/ML3.png')
```

[78]:

1. Dataset link, dataset description, sample data and data pre-processing technique (3 mark)

[79]:
```
# 1. Dataset link, dataset description, sample data and data pre-processing␣
↪technique
```

[80]:
```
#Sample Data
```

[59]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

[60]:

```
# Load the dataset
df = pd.read_csv('/home/kidney_disease.csv')
```

[61]:

```
df.head(20)
```

[61]:

```
    id   age     bp     sg   al   su       rbc        pc         pcc  \
0    0  48.0   80.0  1.020  1.0  0.0       NaN    normal  notpresent
1    1   7.0   50.0  1.020  4.0  0.0       NaN    normal  notpresent
2    2  62.0   80.0  1.010  2.0  3.0    normal    normal  notpresent
3    3  48.0   70.0  1.005  4.0  0.0    normal  abnormal     present
4    4  51.0   80.0  1.010  2.0  0.0    normal    normal  notpresent
5    5  60.0   90.0  1.015  3.0  0.0       NaN       NaN  notpresent
6    6  68.0   70.0  1.010  0.0  0.0       NaN    normal  notpresent
7    7  24.0    NaN  1.015  2.0  4.0    normal  abnormal  notpresent
8    8  52.0  100.0  1.015  3.0  0.0    normal  abnormal     present
9    9  53.0   90.0  1.020  2.0  0.0  abnormal  abnormal     present
10  10  50.0   60.0  1.010  2.0  4.0       NaN  abnormal     present
11  11  63.0   70.0  1.010  3.0  0.0  abnormal  abnormal     present
12  12  68.0   70.0  1.015  3.0  1.0       NaN    normal     present
13  13  68.0   70.0    NaN  NaN  NaN       NaN       NaN  notpresent
14  14  68.0   80.0  1.010  3.0  2.0    normal  abnormal     present
15  15  40.0   80.0  1.015  3.0  0.0       NaN    normal  notpresent
16  16  47.0   70.0  1.015  2.0  0.0       NaN    normal  notpresent
17  17  47.0   80.0    NaN  NaN  NaN       NaN       NaN  notpresent
18  18  60.0  100.0  1.025  0.0  3.0       NaN    normal  notpresent
19  19  62.0   60.0  1.015  1.0  0.0       NaN  abnormal     present

            ba  …  pcv     wc   rc  htn   dm  cad appet   pe  ane  \
0   notpresent  …   44   7800  5.2  yes  yes   no  good   no   no
1   notpresent  …   38   6000  NaN   no   no   no  good   no   no
2   notpresent  …   31   7500  NaN   no  yes   no  poor   no  yes
3   notpresent  …   32   6700  3.9  yes   no   no  poor  yes  yes
4   notpresent  …   35   7300  4.6   no   no   no  good   no   no
5   notpresent  …   39   7800  4.4  yes  yes   no  good  yes   no
6   notpresent  …   36    NaN  NaN   no   no   no  good   no   no
7   notpresent  …   44   6900    5   no  yes   no  good  yes   no
8   notpresent  …   33   9600  4.0  yes  yes   no  good   no  yes
9   notpresent  …   29  12100  3.7  yes  yes   no  poor   no  yes
10  notpresent  …   28    NaN  NaN  yes  yes   no  good   no  yes
11  notpresent  …   32   4500  3.8  yes  yes   no  poor  yes   no
12  notpresent  …   28  12200  3.4  yes  yes  yes  poor  yes   no
13  notpresent  …  NaN    NaN  NaN  yes  yes  yes  poor  yes   no
14     present  …   16  11000  2.6  yes  yes  yes  poor  yes   no
15  notpresent  …   24   3800  2.8  yes   no   no  good   no  yes
16  notpresent  …  NaN    NaN  NaN   no   no   no  good   no   no
```

```
17  notpresent  …  NaN    NaN  NaN  yes   no   no  poor   no   no
18  notpresent  …   37  11400  4.3  yes  yes  yes  good   no   no
19  notpresent  …   30   5300  3.7  yes   no  yes  good   no   no

   classification
0             ckd
1             ckd
2             ckd
3             ckd
4             ckd
5             ckd
6             ckd
7             ckd
8             ckd
9             ckd
10            ckd
11            ckd
12            ckd
13            ckd
14            ckd
15            ckd
16            ckd
17            ckd
18            ckd
19            ckd

[20 rows x 26 columns]
```

[62]:
```
df.describe()
```

[62]:
```
               id         age          bp          sg          al          su  \
count  400.000000  391.000000  388.000000  353.000000  354.000000  351.000000
mean   199.500000   51.483376   76.469072    1.017408    1.016949    0.450142
std    115.614301   17.169714   13.683637    0.005717    1.352679    1.099191
min      0.000000    2.000000   50.000000    1.005000    0.000000    0.000000
25%     99.750000   42.000000   70.000000    1.010000    0.000000    0.000000
50%    199.500000   55.000000   80.000000    1.020000    0.000000    0.000000
75%    299.250000   64.500000   80.000000    1.020000    2.000000    0.000000
max    399.000000   90.000000  180.000000    1.025000    5.000000    5.000000

              bgr          bu          sc         sod         pot        hemo
count  356.000000  381.000000  383.000000  313.000000  312.000000  348.000000
mean   148.036517   57.425722    3.072454  137.528754    4.627244   12.526437
std     79.281714   50.503006    5.741126   10.408752    3.193904    2.912587
min     22.000000    1.500000    0.400000    4.500000    2.500000    3.100000
25%     99.000000   27.000000    0.900000  135.000000    3.800000   10.300000
50%    121.000000   42.000000    1.300000  138.000000    4.400000   12.650000
```

```
75%    163.000000   66.000000    2.800000  142.000000    4.900000   15.000000
max    490.000000  391.000000   76.000000  163.000000   47.000000   17.800000
```

[63]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              400 non-null    int64
 1   age             391 non-null    float64
 2   bp              388 non-null    float64
 3   sg              353 non-null    float64
 4   al              354 non-null    float64
 5   su              351 non-null    float64
 6   rbc             248 non-null    object
 7   pc              335 non-null    object
 8   pcc             396 non-null    object
 9   ba              396 non-null    object
 10  bgr             356 non-null    float64
 11  bu              381 non-null    float64
 12  sc              383 non-null    float64
 13  sod             313 non-null    float64
 14  pot             312 non-null    float64
 15  hemo            348 non-null    float64
 16  pcv             330 non-null    object
 17  wc              295 non-null    object
 18  rc              270 non-null    object
 19  htn             398 non-null    object
 20  dm              398 non-null    object
 21  cad             398 non-null    object
 22  appet           399 non-null    object
 23  pe              399 non-null    object
 24  ane             399 non-null    object
 25  classification  400 non-null    object
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

[111]:
```
# for checking if value is null or not
df.isnull().sum()
```

[111]:
```
id                 0
age                9
bp                12
sg                47
al                46
```

```
su                 49
rbc               152
pc                 65
pcc                 4
ba                  4
bgr                44
bu                 19
sc                 17
sod                87
pot                88
hemo               52
pcv                70
wc                105
rc                130
htn                 2
dm                  2
cad                 2
appet               1
pe                  1
ane                 1
classification      0
dtype: int64
```

[81]:
```
# Dataset Name: Kidney Disease Prediction Dataset

# Dataset Link: https://www.kaggle.com/code/syedali110/
 ↪kidney-disease-prediction-98-accuracy/notebook

# Dataset Description: This dataset contains information about loan applicants.␣
 ↪The aim is to predict whether the loan will be approved or not based on␣
 ↪various features.
```

[112]:
```
from IPython.display import Image

Image('/home/imp1.png')
```

[112]:

Dataset Name: Kidney Disease Prediction Dataset

Dataset Link: https://www.kaggle.com/code/syedali110/kidney-disease-prediction-98-accuracy/notebook

Dataset Description:
The Kidney Disease Prediction dateset contains various medical features and the target variable indicating whether a patient has chronic kidney disease (CKD) or not.

Dataset Reference:
id: ID of the patient
age: Age of the patient (in years)
bp: Blood pressure of the patient (in mm/Hg)
sg: Specific gravity of urine
al: Albumin content in urine
su: Sugar content in urine
rbc: Red blood cells
pc: Pus cells
pcc: Pus cell clumps
ba: Bacteria
bgr: Blood glucose random (in mgs/dl)
bu: Blood urea (in mgs/dl)
sc: Serum creatinine (in mgs/dl)
sod: Sodium (in mEq/L)
pot: Potassium (in mEq/L)
hemo: Hemoglobin (in gms)
pcv: Packed cell volume
wc: White blood cell count (in cells/cumm)
rc: Red blood cell count (in millions/cmm)
htn: Hypertension
dm: Diabetes mellitus
cad: Coronary artery disease
appet: Appetite
pe: Pedal edema
ane: Anemia
classification: Target variable indicating whether a patient has chronic kidney disease (CKD) or not

[113]:
```python
from IPython.display import Image

Image('/home/imp2.png')
```

[113]:

Kidney Disease Prediction

Introduction:
Chronic kidney disease (CKD) is a condition where the kidneys gradually lose their function over time. Early detection and treatment can help prevent the progression of CKD and its complications. In this project, we aim to build a machine learning model to predict whether a patient has chronic kidney disease based on various medical features.

Data Preprocessing:

1. Handling Missing Values:
Missing values are filled using the mean for numerical features.
Categorical features are filled with the mode.

2. Handling Categorical Variables:
Categorical variables are encoded using Label Encoding.

3. Feature Scaling:
The features are normalized using StandardScaler.

Model Building:

1. Feature Selection:
Principal Component Analysis (PCA) is applied to reduce the dimensionality of the dataset.

2. Model Selection:
Logistic Regression, K-Nearest Neighbors (KNN), Decision Tree, Random Forest, and Naive Bayes classifiers are used for prediction.

Model Evaluation:

- **Evaluation Metrics:**

Classification Accuracy, Confusion Matrix, and Purity of Clusters are used to evaluate the performance of the models.

Conclusion:
The machine learning models trained on the Kidney Disease Prediction dateset showed promising results in predicting chronic kidney disease. Logistic Regression and Random Forest classifiers performed well with high accuracy and purity of clusters. These models can be valuable tools for early detection and prevention of chronic kidney disease.

[82]:
```
#Data Preprocesing
```

[1]:
```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
```

[3]:
```
# Load the dataset
df = pd.read_csv('/home/kidney_disease.csv')
```

[12]:
```python
# Handling missing values
imputer = SimpleImputer(strategy='mean')
df[['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo']] =␣
 ↪imputer.fit_transform(df[['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc',␣
 ↪'sod', 'pot', 'hemo']])

# Print values after handling missing values
print(df[['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot',␣
 ↪'hemo']].head())
```

```
    age         bp     sg   al   su    bgr    bu   sc         sod       pot  \
0  48.0  80.000000  1.020  1.0  0.0  121.0  36.0  1.2  137.528754  4.627244
3  48.0  70.000000  1.005  4.0  0.0  117.0  56.0  3.8  111.000000  2.500000
4  51.0  80.000000  1.010  2.0  0.0  106.0  26.0  1.4  137.528754  4.627244
5  60.0  90.000000  1.015  3.0  0.0   74.0  25.0  1.1  142.000000  3.200000
7  24.0  76.469072  1.015  2.0  4.0  410.0  31.0  1.1  137.528754  4.627244

   hemo
0  15.4
3  11.2
4  11.6
5  12.2
7  12.4
```

[16]:
```python
# Handling categorical variables
cat_columns = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe',␣
 ↪'ane', 'classification']
df[cat_columns] = df[cat_columns].fillna(df.mode().iloc[0])
df[cat_columns] = df[cat_columns].apply(LabelEncoder().fit_transform)

# Print the dataframe
print(df.head())
```

```
   id   age         bp     sg   al   su  rbc  pc  pcc  ba  …  pcv    wc  \
0   0  48.0  80.000000  1.020  1.0  0.0    1   1    0   0  …   44  7800
3   3  48.0  70.000000  1.005  4.0  0.0    1   0    1   0  …   32  6700
4   4  51.0  80.000000  1.010  2.0  0.0    1   1    0   0  …   35  7300
5   5  60.0  90.000000  1.015  3.0  0.0    1   1    0   0  …   39  7800
7   7  24.0  76.469072  1.015  2.0  4.0    1   0    0   0  …   44  6900

    rc  htn  dm  cad appet pe ane  classification
0  5.2    1   1    1     0  0   0               0
3  3.9    1   0    1     1  1   1               0
4  4.6    0   0    1     0  0   0               0
5  4.4    1   1    1     0  1   0               0
7    5    0   1    1     0  1   0               0
```

```
[5 rows x 26 columns]
```

[17]:
```
# Removing junk values
df.replace(to_replace={'\t?': np.nan, '\t43': 43, '\t?80': 80}, inplace=True)
df.dropna(inplace=True)

# Print the dataframe
print(df.head())
```

```
   id   age         bp     sg   al   su  rbc  pc  pcc  ba  ...  pcv    wc  \
0   0  48.0  80.000000  1.020  1.0  0.0    1   1    0   0  ...   44  7800
3   3  48.0  70.000000  1.005  4.0  0.0    1   0    1   0  ...   32  6700
4   4  51.0  80.000000  1.010  2.0  0.0    1   1    0   0  ...   35  7300
5   5  60.0  90.000000  1.015  3.0  0.0    1   1    0   0  ...   39  7800
7   7  24.0  76.469072  1.015  2.0  4.0    1   0    0   0  ...   44  6900

    rc  htn  dm  cad appet pe ane  classification
0  5.2    1   1    1     0  0   0               0
3  3.9    1   0    1     1  1   1               0
4  4.6    0   0    1     0  0   0               0
5  4.4    1   1    1     0  1   0               0
7    5    0   1    1     0  1   0               0

[5 rows x 26 columns]
```

[18]:
```
# Separating features and target variable
X = df.drop(columns=['id', 'classification'])
y = df['classification']

# Print the dataframe
print(df.head())
```

```
   id   age         bp     sg   al   su  rbc  pc  pcc  ba  ...  pcv    wc  \
0   0  48.0  80.000000  1.020  1.0  0.0    1   1    0   0  ...   44  7800
3   3  48.0  70.000000  1.005  4.0  0.0    1   0    1   0  ...   32  6700
4   4  51.0  80.000000  1.010  2.0  0.0    1   1    0   0  ...   35  7300
5   5  60.0  90.000000  1.015  3.0  0.0    1   1    0   0  ...   39  7800
7   7  24.0  76.469072  1.015  2.0  4.0    1   0    0   0  ...   44  6900

    rc  htn  dm  cad appet pe ane  classification
0  5.2    1   1    1     0  0   0               0
3  3.9    1   0    1     1  1   1               0
4  4.6    0   0    1     0  0   0               0
5  4.4    1   1    1     0  1   0               0
7    5    0   1    1     0  1   0               0

[5 rows x 26 columns]
```

```
[83]: # Normalization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
[9]: # Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)
```

```
[25]: # Applying PCA for training data
pca = PCA()
X_train_pca = pca.fit_transform(X_train)

# Principal components for training data
principal_components_train = pd.DataFrame(pca.components_, columns=X.columns)
print("Principal Components for Training Data:")
print(principal_components_train)

# Applying PCA for testing data
X_test_pca = pca.transform(X_test)

# Principal components for testing data
principal_components_test = pd.DataFrame(pca.components_, columns=X.columns)
print("\nPrincipal Components for Testing Data:")
print(principal_components_test)
```

```
Principal Components for Training Data:
         age        bp        sg        al        su       rbc        pc  \
0   0.118776  0.145140 -0.255064  0.261501  0.147642 -0.144823 -0.226898
1  -0.133778 -0.091580  0.127237 -0.062061 -0.396937  0.073086 -0.010553
2   0.102457  0.145211  0.048798 -0.138551  0.335084 -0.004279  0.280225
3  -0.082302  0.110488  0.113508 -0.034465  0.123553  0.127112 -0.001306
4  -0.331774 -0.009293 -0.056783  0.213329  0.130726 -0.285677 -0.326937
5   0.487176 -0.204123  0.085894  0.025255  0.056300  0.478525 -0.147072
6  -0.091559  0.772306  0.001622 -0.091433  0.068051  0.348202 -0.081415
7  -0.370325 -0.199485  0.071581  0.093528  0.248580  0.349749  0.167832
8  -0.352104 -0.250090  0.205031 -0.070519  0.221061  0.172743 -0.164498
9   0.291709 -0.131162  0.190275  0.077393 -0.013387 -0.371304 -0.053432
10  0.294288  0.308778  0.319608  0.104322  0.043636 -0.125370 -0.088043
11 -0.307095  0.196003  0.417788 -0.110362  0.042502 -0.240701 -0.097091
12  0.148071 -0.059214  0.196824 -0.171823  0.201341  0.110051  0.134313
13 -0.053404 -0.054486  0.266544  0.255261 -0.157242  0.056069  0.190412
14 -0.032633  0.019158  0.457872  0.176121 -0.128392  0.007534 -0.086326
15 -0.118841  0.073904 -0.371990  0.321976  0.018136  0.014569  0.165031
16  0.018620  0.055210 -0.089744 -0.344689 -0.256964 -0.084214 -0.288927
17  0.073697 -0.047206  0.160402  0.329002  0.186660  0.093303 -0.413259
18 -0.033457  0.095613  0.000396  0.267072  0.105499 -0.180792  0.332352
19 -0.039243  0.032401  0.124437 -0.153334 -0.236952 -0.125201  0.351966
```

```
20  0.123089  0.019492  0.116890  0.225273  0.237868 -0.098456  0.244388
21  0.022040 -0.093526 -0.049390 -0.448555  0.460378 -0.224083 -0.069053
22  0.030817  0.074948  0.068892 -0.022529 -0.071165  0.025173  0.099984
23  0.053044 -0.056783  0.069962 -0.079898  0.172739 -0.147511  0.086007

         pcc        ba       bgr  …      hemo       pcv        wc        rc  \
0   0.179953  0.114523  0.188908  … -0.297981 -0.297362  0.087756 -0.282186
1  -0.012883 -0.080097 -0.427787  … -0.085297 -0.089800 -0.211464 -0.085357
2  -0.320470 -0.221203  0.266565  …  0.107949  0.089618 -0.087547  0.075387
3  -0.171315 -0.216024  0.019175  … -0.051330 -0.073024 -0.198205 -0.069256
4   0.146388  0.454883  0.167183  …  0.107519  0.142288 -0.228847  0.182429
5   0.465358  0.001474  0.008793  …  0.071072  0.045816  0.025171 -0.017580
6   0.200454  0.050888  0.033679  … -0.005258 -0.020239 -0.020609  0.078408
7  -0.018918  0.158167  0.037866  … -0.029617 -0.018322  0.668382  0.021605
8   0.271176 -0.304048  0.266585  …  0.109926  0.142492 -0.361573  0.112552
9   0.053534 -0.243291  0.309853  …  0.040977  0.055911  0.264892  0.193371
10 -0.041374  0.090691 -0.152430  …  0.244997  0.186919  0.100052  0.278775
11  0.053071 -0.200404 -0.052661  … -0.116303 -0.130550  0.249033 -0.163452
12 -0.286259  0.487970  0.157126  … -0.040922 -0.042280 -0.216357 -0.152700
13 -0.132571  0.328496 -0.055978  …  0.086306  0.099653 -0.045973  0.093890
14  0.144920  0.039920 -0.095924  … -0.021497 -0.033287  0.039271  0.085588
15 -0.013466 -0.218785 -0.156967  … -0.016364  0.111987 -0.002635  0.419146
16 -0.087799  0.042930  0.104277  …  0.124374  0.175279  0.195488 -0.072371
17 -0.376813 -0.186268 -0.058520  … -0.121219 -0.099533 -0.074916 -0.215499
18  0.261666 -0.074839 -0.090001  …  0.362815  0.310014 -0.041973 -0.629592
19  0.276078  0.047189  0.348549  … -0.070646 -0.376964 -0.094407  0.048604
20  0.098053 -0.015422 -0.208475  … -0.127487 -0.339828 -0.134344  0.153718
21  0.169186  0.090481 -0.470950  …  0.092680 -0.042774  0.038394  0.041274
22  0.033625  0.001712  0.060143  … -0.547770  0.565247 -0.008452 -0.087909
23  0.146583  0.045210 -0.108573  … -0.532239  0.224992 -0.054487  0.096339

         htn        dm       cad     appet        pe       ane
0   0.276183  0.244659  0.121487  0.228418  0.202412  0.179672
1  -0.112377 -0.170815 -0.050880  0.025545  0.028525  0.170765
2   0.133414  0.214839  0.185071 -0.184434 -0.134944 -0.191811
3   0.050623  0.063915 -0.172714 -0.040208 -0.052434  0.259378
4  -0.191426 -0.126416  0.031751 -0.236566 -0.196265 -0.129857
5   0.034103 -0.153367  0.319837 -0.200035 -0.152015 -0.031404
6  -0.041558 -0.133267 -0.230037  0.019628 -0.177855  0.192669
7  -0.104220 -0.063998 -0.168022 -0.013953  0.158652  0.010228
8   0.035802  0.083297  0.027935  0.253264  0.320447  0.168255
9  -0.094348 -0.219969 -0.296988  0.050518 -0.272533  0.434374
10 -0.061263 -0.080634  0.039646  0.302317  0.495135 -0.213484
11 -0.009890 -0.177711  0.600896 -0.049937 -0.152104  0.006058
12 -0.026214 -0.218426  0.067863 -0.163911  0.135970  0.309863
13  0.430333  0.039144  0.192038  0.293785 -0.178646  0.298754
14  0.288693  0.501973 -0.270523 -0.435289 -0.039815 -0.089600
15  0.317776 -0.263310  0.201930 -0.317420  0.164431  0.209748
```

```
16  0.029504  0.122312  0.093959 -0.399916  0.412752  0.335502
17  0.148885 -0.378532 -0.190168 -0.156016  0.110036 -0.189460
18 -0.002632 -0.088392 -0.116312 -0.117852  0.080215  0.073203
19  0.245876 -0.350208 -0.146344 -0.122425  0.249535 -0.264389
20 -0.501172  0.194982  0.105674 -0.193356  0.188626  0.213511
21  0.331053 -0.067513 -0.152343  0.010151 -0.059169  0.028880
22 -0.076626  0.005955  0.019667 -0.018720  0.043396 -0.094112
23  0.077223 -0.063949 -0.111110 -0.000230  0.049063  0.022903

[24 rows x 24 columns]

Principal Components for Testing Data:
         age        bp        sg        al        su       rbc        pc  \
0   0.118776  0.145140 -0.255064  0.261501  0.147642 -0.144823 -0.226898
1  -0.133778 -0.091580  0.127237 -0.062061 -0.396937  0.073086 -0.010553
2   0.102457  0.145211  0.048798 -0.138551  0.335084 -0.004279  0.280225
3  -0.082302  0.110488  0.113508 -0.034465  0.123553  0.127112 -0.001306
4  -0.331774 -0.009293 -0.056783  0.213329  0.130726 -0.285677 -0.326937
5   0.487176 -0.204123  0.085894  0.025255  0.056300  0.478525 -0.147072
6  -0.091559  0.772306  0.001622 -0.091433  0.068051  0.348202 -0.081415
7  -0.370325 -0.199485  0.071581  0.093528  0.248580  0.349749  0.167832
8  -0.352104 -0.250090  0.205031 -0.070519  0.221061  0.172743 -0.164498
9   0.291709 -0.131162  0.190275  0.077393 -0.013387 -0.371304 -0.053432
10  0.294288  0.308778  0.319608  0.104322  0.043636 -0.125370 -0.088043
11 -0.307095  0.196003  0.417788 -0.110362  0.042502 -0.240701 -0.097091
12  0.148071 -0.059214  0.196824 -0.171823  0.201341  0.110051  0.134313
13 -0.053404 -0.054486  0.266544  0.255261 -0.157242  0.056069  0.190412
14 -0.032633  0.019158  0.457872  0.176121 -0.128392  0.007534 -0.086326
15 -0.118841  0.073904 -0.371990  0.321976  0.018136  0.014569  0.165031
16  0.018620  0.055210 -0.089744 -0.344689 -0.256964 -0.084214 -0.288927
17  0.073697 -0.047206  0.160402  0.329002  0.186660  0.093303 -0.413259
18 -0.033457  0.095613  0.000396  0.267072  0.105499 -0.180792  0.332352
19 -0.039243  0.032401  0.124437 -0.153334 -0.236952 -0.125201  0.351966
20  0.123089  0.019492  0.116890  0.225273  0.237868 -0.098456  0.244388
21  0.022040 -0.093526 -0.049390 -0.448555  0.460378 -0.224083 -0.069053
22  0.030817  0.074948  0.068892 -0.022529 -0.071165  0.025173  0.099984
23  0.053044 -0.056783  0.069962 -0.079898  0.172739 -0.147511  0.086007

         pcc        ba       bgr  …      hemo       pcv        wc        rc  \
0   0.179953  0.114523  0.188908  … -0.297981 -0.297362  0.087756 -0.282186
1  -0.012883 -0.080097 -0.427787  … -0.085297 -0.089800 -0.211464 -0.085357
2  -0.320470 -0.221203  0.266565  …  0.107949  0.089618 -0.087547  0.075387
3  -0.171315 -0.216024  0.019175  … -0.051330 -0.073024 -0.198205 -0.069256
4   0.146388  0.454883  0.167183  …  0.107519  0.142288 -0.228847  0.182429
5   0.465358  0.001474  0.008793  …  0.071072  0.045816  0.025171 -0.017580
6   0.200454  0.050888  0.033679  … -0.005258 -0.020239 -0.020609  0.078408
7  -0.018918  0.158167  0.037866  … -0.029617 -0.018322  0.668382  0.021605
8   0.271176 -0.304048  0.266585  …  0.109926  0.142492 -0.361573  0.112552
```

```
9   0.053534 -0.243291  0.309853  …  0.040977  0.055911  0.264892  0.193371
10 -0.041374  0.090691 -0.152430  …  0.244997  0.186919  0.100052  0.278775
11  0.053071 -0.200404 -0.052661  … -0.116303 -0.130550  0.249033 -0.163452
12 -0.286259  0.487970  0.157126  … -0.040922 -0.042280 -0.216357 -0.152700
13 -0.132571  0.328496 -0.055978  …  0.086306  0.099653 -0.045973  0.093890
14  0.144920  0.039920 -0.095924  … -0.021497 -0.033287  0.039271  0.085588
15 -0.013466 -0.218785 -0.156967  … -0.016364  0.111987 -0.002635  0.419146
16 -0.087799  0.042930  0.104277  …  0.124374  0.175279  0.195488 -0.072371
17 -0.376813 -0.186268 -0.058520  … -0.121219 -0.099533 -0.074916 -0.215499
18  0.261666 -0.074839 -0.090001  …  0.362815  0.310014 -0.041973 -0.629592
19  0.276078  0.047189  0.348549  … -0.070646 -0.376964 -0.094407  0.048604
20  0.098053 -0.015422 -0.208475  … -0.127487 -0.339828 -0.134344  0.153718
21  0.169186  0.090481 -0.470950  …  0.092680 -0.042774  0.038394  0.041274
22  0.033625  0.001712  0.060143  … -0.547770  0.565247 -0.008452 -0.087909
23  0.146583  0.045210 -0.108573  … -0.532239  0.224992 -0.054487  0.096339

         htn        dm       cad     appet        pe       ane
0   0.276183  0.244659  0.121487  0.228418  0.202412  0.179672
1  -0.112377 -0.170815 -0.050880  0.025545  0.028525  0.170765
2   0.133414  0.214839  0.185071 -0.184434 -0.134944 -0.191811
3   0.050623  0.063915 -0.172714 -0.040208 -0.052434  0.259378
4  -0.191426 -0.126416  0.031751 -0.236566 -0.196265 -0.129857
5   0.034103 -0.153367  0.319837 -0.200035 -0.152015 -0.031404
6  -0.041558 -0.133267 -0.230037  0.019628 -0.177855  0.192669
7  -0.104220 -0.063998 -0.168022 -0.013953  0.158652  0.010228
8   0.035802  0.083297  0.027935  0.253264  0.320447  0.168255
9  -0.094348 -0.219969 -0.296988  0.050518 -0.272533  0.434374
10 -0.061263 -0.080634  0.039646  0.302317  0.495135 -0.213484
11 -0.009890 -0.177711  0.600896 -0.049937 -0.152104  0.006058
12 -0.026214 -0.218426  0.067863 -0.163911  0.135970  0.309863
13  0.430333  0.039144  0.192038  0.293785 -0.178646  0.298754
14  0.288693  0.501973 -0.270523 -0.435289 -0.039815 -0.089600
15  0.317776 -0.263310  0.201930 -0.317420  0.164431  0.209748
16  0.029504  0.122312  0.093959 -0.399916  0.412752  0.335502
17  0.148885 -0.378532 -0.190168 -0.156016  0.110036 -0.189460
18 -0.002632 -0.088392 -0.116312 -0.117852  0.080215  0.073203
19  0.245876 -0.350208 -0.146344 -0.122425  0.249535 -0.264389
20 -0.501172  0.194982  0.105674 -0.193356  0.188626  0.213511
21  0.331053 -0.067513 -0.152343  0.010151 -0.059169  0.028880
22 -0.076626  0.005955  0.019667 -0.018720  0.043396 -0.094112
23  0.077223 -0.063949 -0.111110 -0.000230  0.049063  0.022903

[24 rows x 24 columns]
```

[11]:
```python
# Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
```

[26]:
```python
# Cumulative explained variance
cumulative_explained_variance = np.cumsum(explained_variance_ratio)
```

[27]:
```python
# Number of components to explain 95% variance
n_components = np.argmax(cumulative_explained_variance >= 0.95) + 1

print("Number of components to explain 95% variance:", n_components)
```

```
Number of components to explain 95% variance: 18
```

[86]:
```python
from IPython.display import Image

Image('/home/ML4.png')
```

[86]:

2. A classification & a clustering algorithms using python and document the results with appropriate screenshots and also explanation of classification & clustering algorithms (step by step procedure) (4 marks)

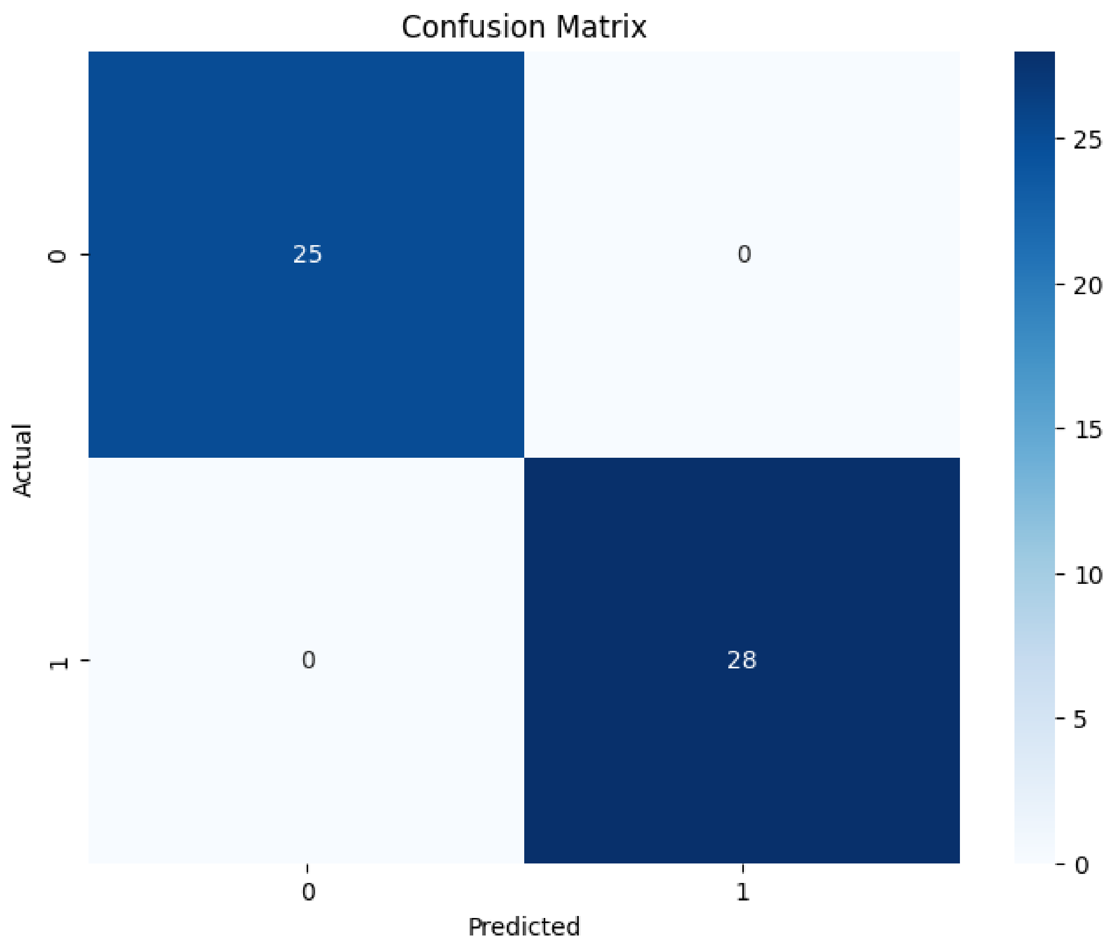
[ ]:
```python
# 2. A classification & a clustering algorithms using python and document the results with appropriate screenshots and also explanation of classification & clustering algorithms (step by step procedure)
```

[53]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.cluster import KMeans
from scipy.stats import mode

# Logistic Regression
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_pca, y_train)

# Predictions
y_pred = log_reg.predict(X_test_pca)

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

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

# Clustering using KMeans
```

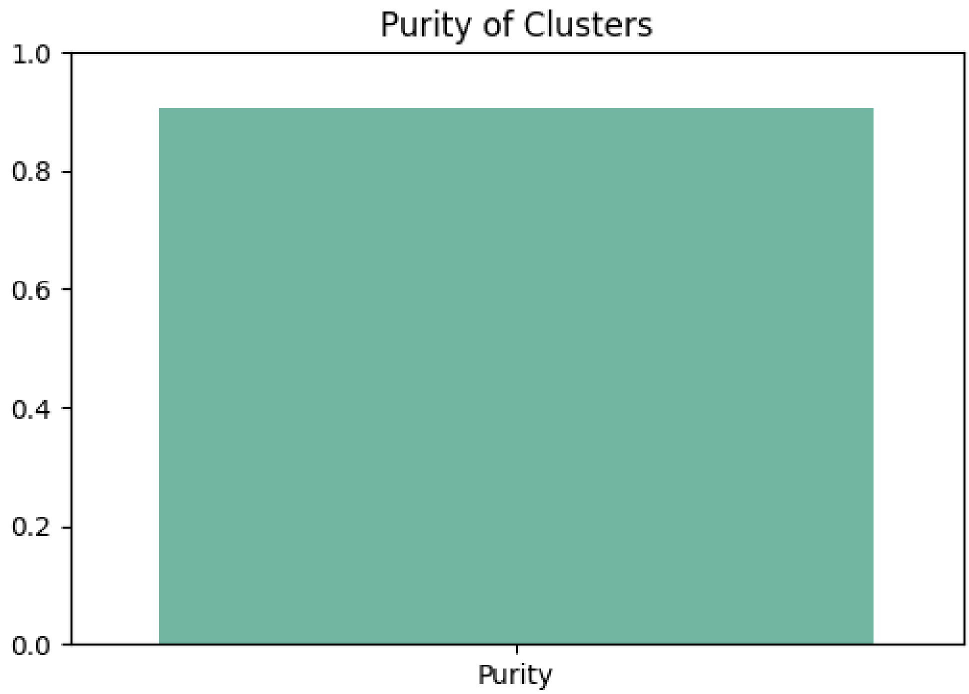
```
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans.fit(X_train_pca)

# Getting the cluster labels
cluster_labels = kmeans.labels_

# Purity of clusters
def purity_score(y_true, y_pred):
    majority = np.zeros_like(y_pred)
    for cluster in np.unique(y_pred):
        mask = (y_pred == cluster)
        majority[mask] = mode(y_true[mask])[0]
    return accuracy_score(y_true, majority)

purity = purity_score(y_train, cluster_labels)
print("\nPurity of Clusters:", purity)
```

```
Classification Accuracy: 1.0

Confusion Matrix:
[[25  0]
 [ 0 28]]

Purity of Clusters: 0.9052132701421801
```

[84]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Barplot for Purity of Clusters
plt.figure(figsize=(6, 4))
sns.barplot(x=['Purity'], y=[purity], palette='Set2')
plt.title('Purity of Clusters')
plt.ylim(0, 1)
plt.show()
```




```
<ipython-input-84-5d25b27718b0>:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=['Purity'], y=[purity], palette='Set2')
```



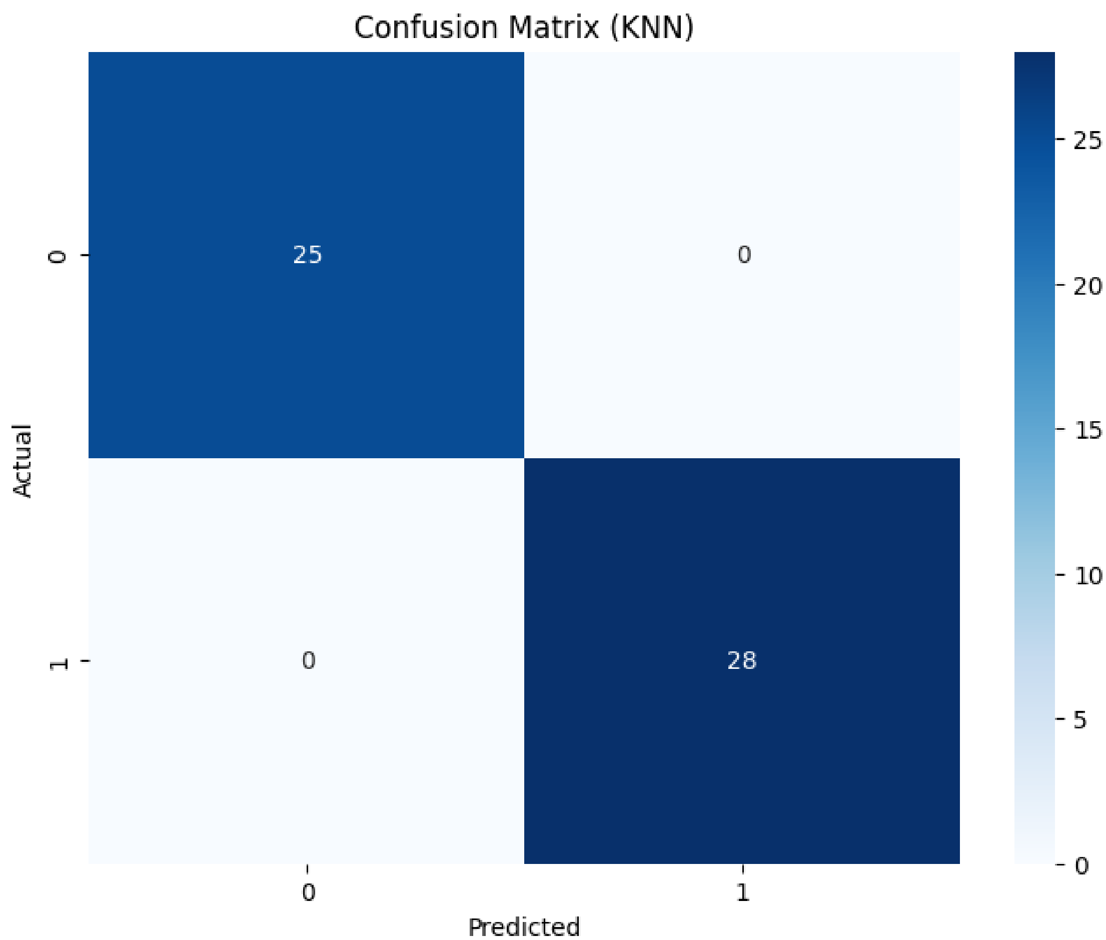
[47]:
```
from sklearn.neighbors import KNeighborsClassifier

# K-Nearest Neighbors
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_pca, y_train)

# Predictions
y_pred_knn = knn.predict(X_test_pca)

# Accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print("Classification Accuracy (KNN):", accuracy_knn)

# Confusion Matrix
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
print("\nConfusion Matrix (KNN):")
print(conf_matrix_knn)

# Purity of clusters
purity_knn = purity_score(y_train, kmeans.labels_)
print("\nPurity of Clusters (KNN):", purity_knn)
```

```
Classification Accuracy (KNN): 1.0
```

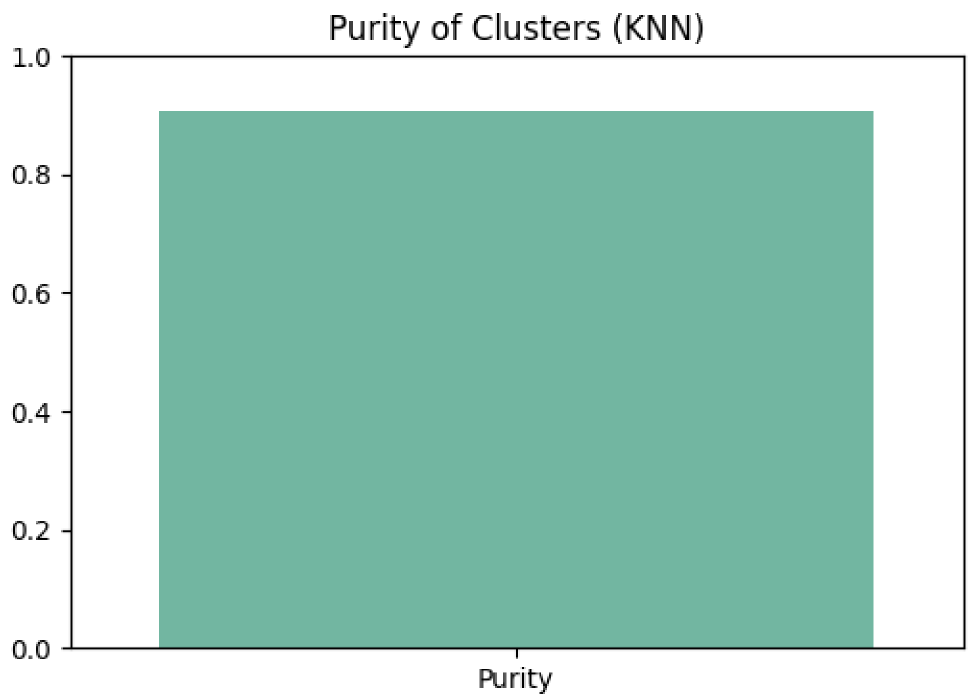
```
Confusion Matrix (KNN):
[[25  0]
 [ 0 28]]

Purity of Clusters (KNN): 0.9052132701421801
```

[48]:

```
# Visualizing Confusion Matrix (KNN)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_knn, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix (KNN)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Visualizing Purity of Clusters (KNN)
plt.figure(figsize=(6, 4))
sns.barplot(x=['Purity'], y=[purity_knn], palette='Set2')
plt.title('Purity of Clusters (KNN)')
plt.ylim(0, 1)
plt.show()
```



```
&lt;ipython-input-48-e6e287e2f234&gt;:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=['Purity'], y=[purity_knn], palette='Set2')
```



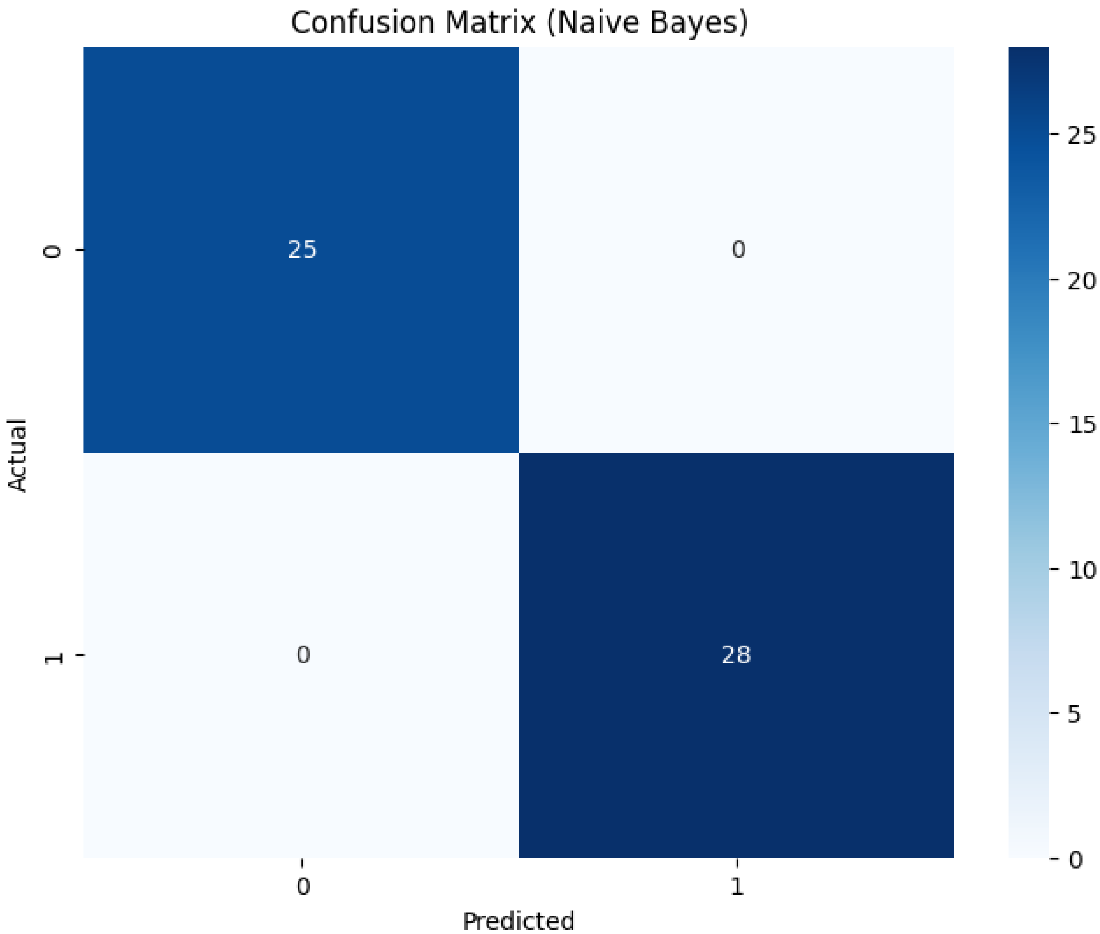
[50]:

```python
from sklearn.naive_bayes import GaussianNB

# Naive Bayes
nb = GaussianNB()
nb.fit(X_train_pca, y_train)

# Predictions
y_pred_nb = nb.predict(X_test_pca)

# Accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print("Classification Accuracy (Naive Bayes):", accuracy_nb)

# Confusion Matrix
```

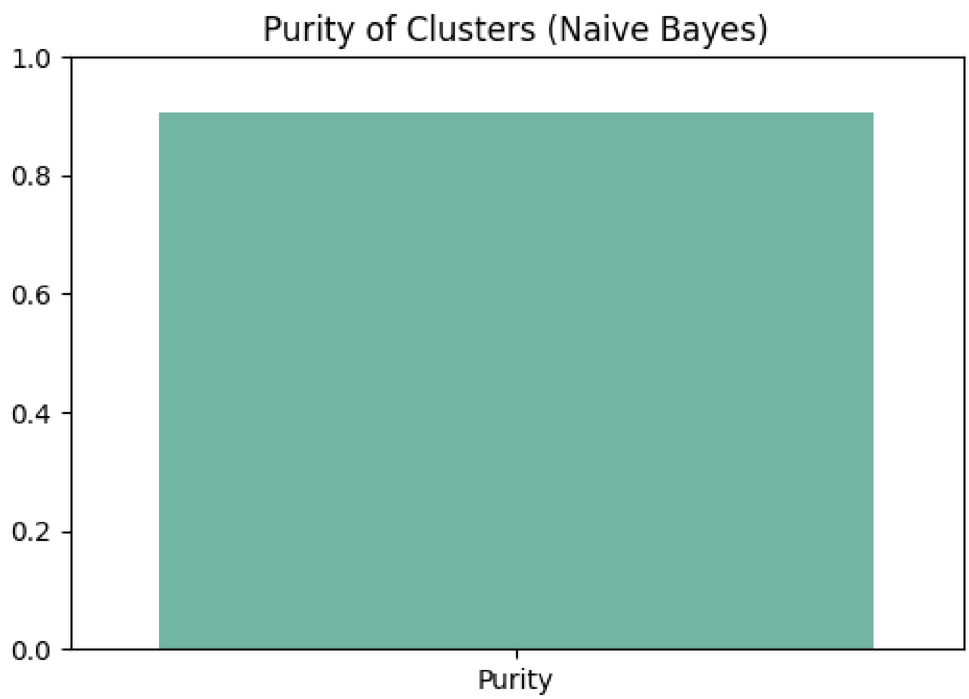
```
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)
print("\nConfusion Matrix (Naive Bayes):")
print(conf_matrix_nb)

# Purity of clusters
purity_nb = purity_score(y_train, kmeans.labels_)
print("\nPurity of Clusters (Naive Bayes):", purity_nb)
```

```
Classification Accuracy (Naive Bayes): 1.0

Confusion Matrix (Naive Bayes):
[[25  0]
 [ 0 28]]

Purity of Clusters (Naive Bayes): 0.9052132701421801
```

[54]:
```
# Visualizing Confusion Matrix (Naive Bayes)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_nb, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix (Naive Bayes)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Visualizing Purity of Clusters (Naive Bayes)
plt.figure(figsize=(6, 4))
sns.barplot(x=['Purity'], y=[purity_nb], palette='Set2')
plt.title('Purity of Clusters (Naive Bayes)')
plt.ylim(0, 1)
plt.show()
```




```
<ipython-input-54-043877ed10eb>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=['Purity'], y=[purity_nb], palette='Set2')
```



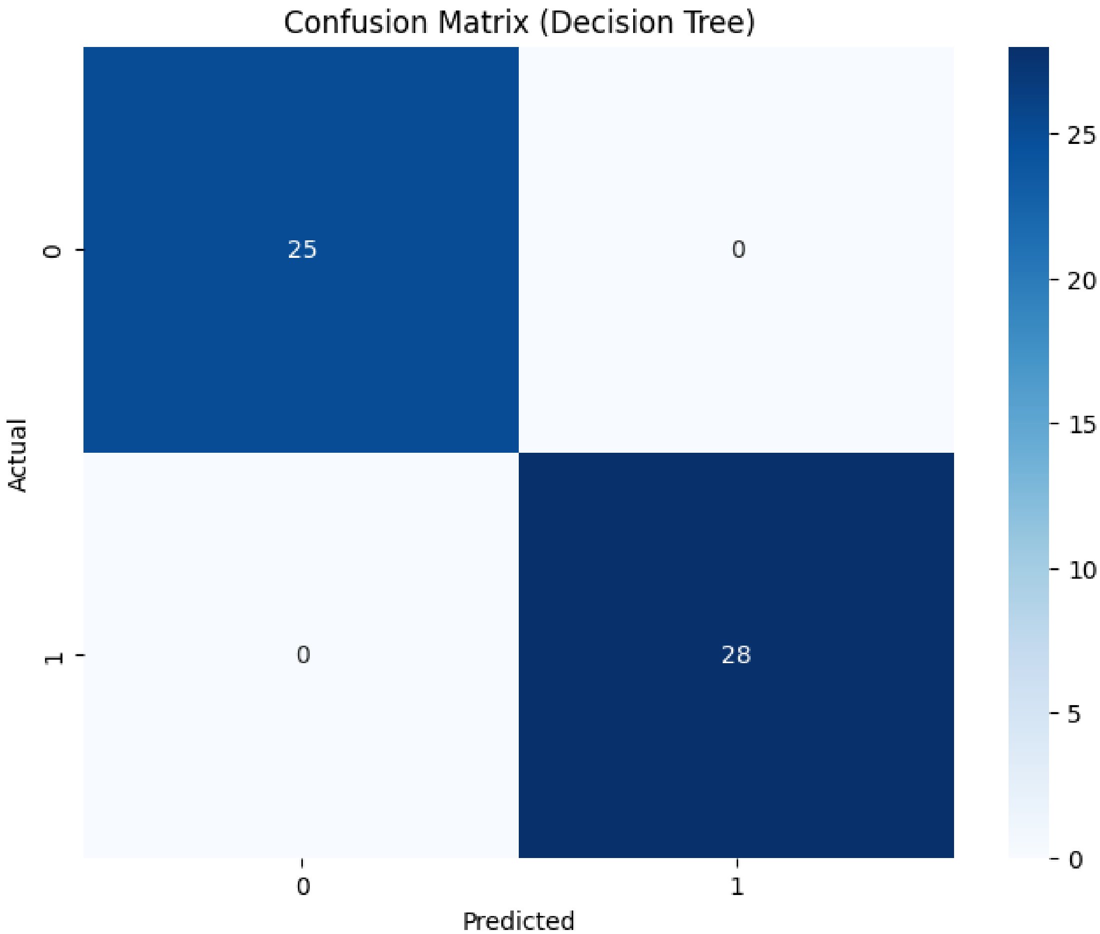
[55]:

```
from sklearn.tree import DecisionTreeClassifier

# Decision Tree
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train_pca, y_train)

# Predictions
y_pred_dt = decision_tree.predict(X_test_pca)

# Accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Classification Accuracy (Decision Tree):", accuracy_dt)

# Confusion Matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
print("\nConfusion Matrix (Decision Tree):")
print(conf_matrix_dt)

# Clustering using KMeans
kmeans_dt = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans_dt.fit(X_train_pca)

# Getting the cluster labels
cluster_labels_dt = kmeans_dt.labels_
```

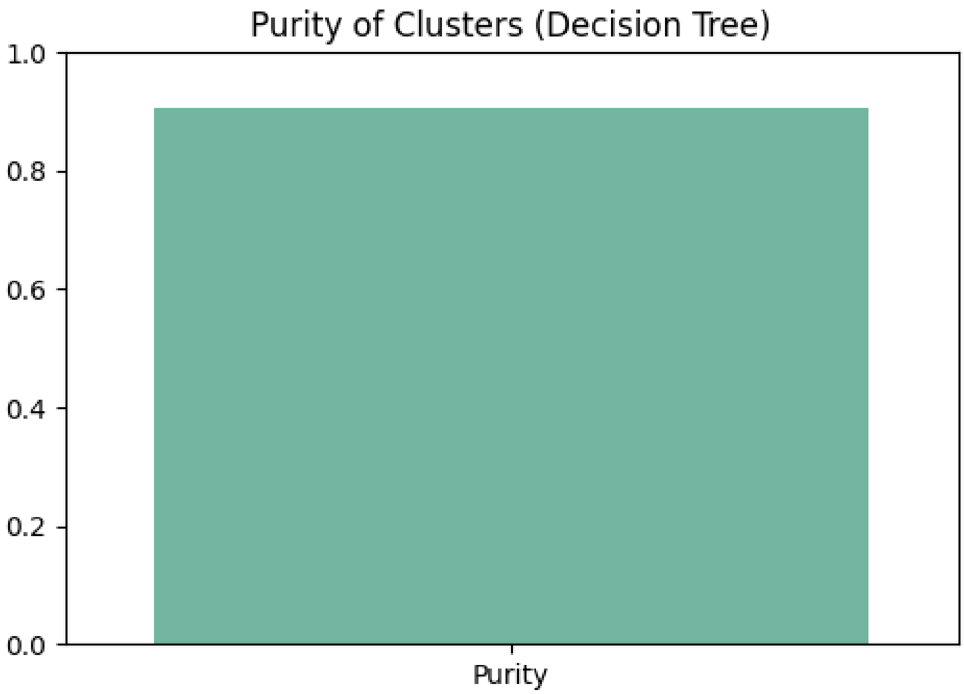
```
# Purity of clusters
purity_dt = purity_score(y_train, cluster_labels_dt)
print("\nPurity of Clusters (Decision Tree):", purity_dt)
```

```
Classification Accuracy (Decision Tree): 1.0

Confusion Matrix (Decision Tree):
[[25  0]
 [ 0 28]]

Purity of Clusters (Decision Tree): 0.9052132701421801
```

[56]:
```
# Visualizing Confusion Matrix (Decision Tree)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_dt, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix (Decision Tree)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Visualizing Purity of Clusters (Decision Tree)
plt.figure(figsize=(6, 4))
sns.barplot(x=['Purity'], y=[purity_dt], palette='Set2')
plt.title('Purity of Clusters (Decision Tree)')
plt.ylim(0, 1)
plt.show()
```




```
<ipython-input-56-3c5e7b261452>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=['Purity'], y=[purity_dt], palette='Set2')
```

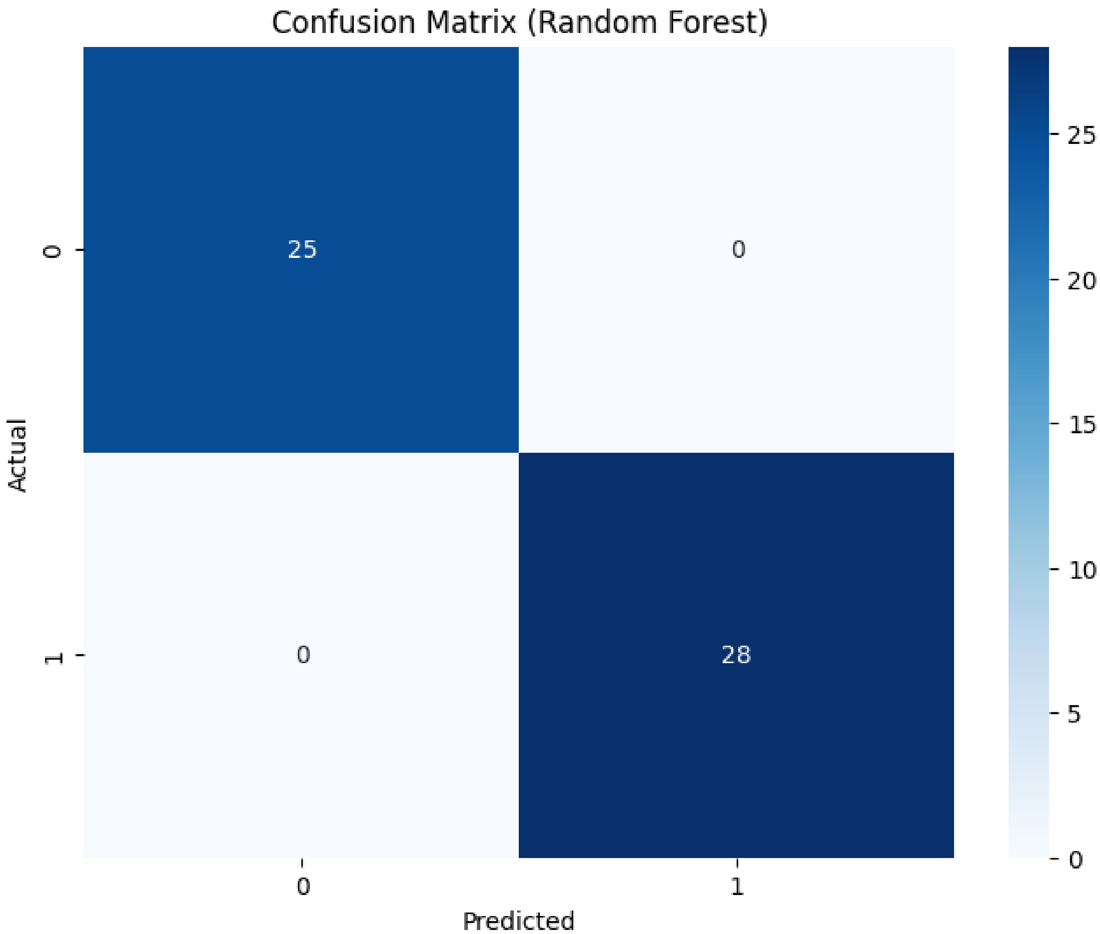


[57]:
```
from sklearn.ensemble import RandomForestClassifier

# Random Forest
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(X_train_pca, y_train)

# Predictions
y_pred_rf = random_forest.predict(X_test_pca)

# Accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Classification Accuracy (Random Forest):", accuracy_rf)

# Confusion Matrix
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print("\nConfusion Matrix (Random Forest):")
print(conf_matrix_rf)

# Clustering using KMeans
kmeans_rf = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans_rf.fit(X_train_pca)

# Getting the cluster labels
cluster_labels_rf = kmeans_rf.labels_
```

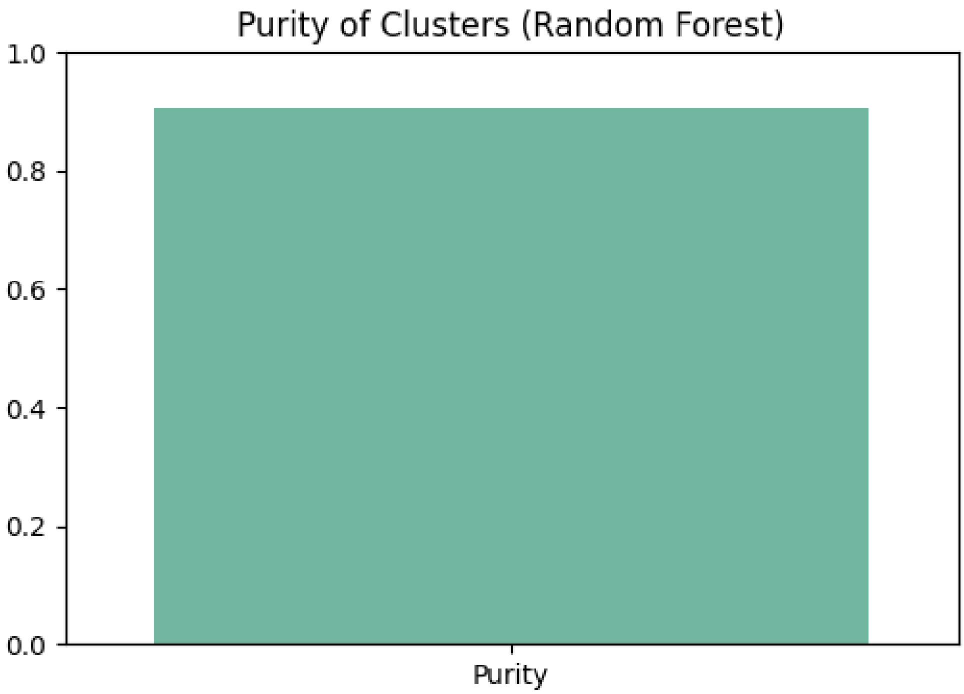
```
# Purity of clusters
purity_rf = purity_score(y_train, cluster_labels_rf)
print("\nPurity of Clusters (Random Forest):", purity_rf)
```

```
Classification Accuracy (Random Forest): 1.0

Confusion Matrix (Random Forest):
[[25  0]
 [ 0 28]]

Purity of Clusters (Random Forest): 0.9052132701421801
```

[58]:
```
# Visualizing Confusion Matrix (Random Forest)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix (Random Forest)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Visualizing Purity of Clusters (Random Forest)
plt.figure(figsize=(6, 4))
sns.barplot(x=['Purity'], y=[purity_rf], palette='Set2')
plt.title('Purity of Clusters (Random Forest)')
plt.ylim(0, 1)
plt.show()
```




```
<ipython-input-58-db0d9b61d25c>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=['Purity'], y=[purity_rf], palette='Set2')
```

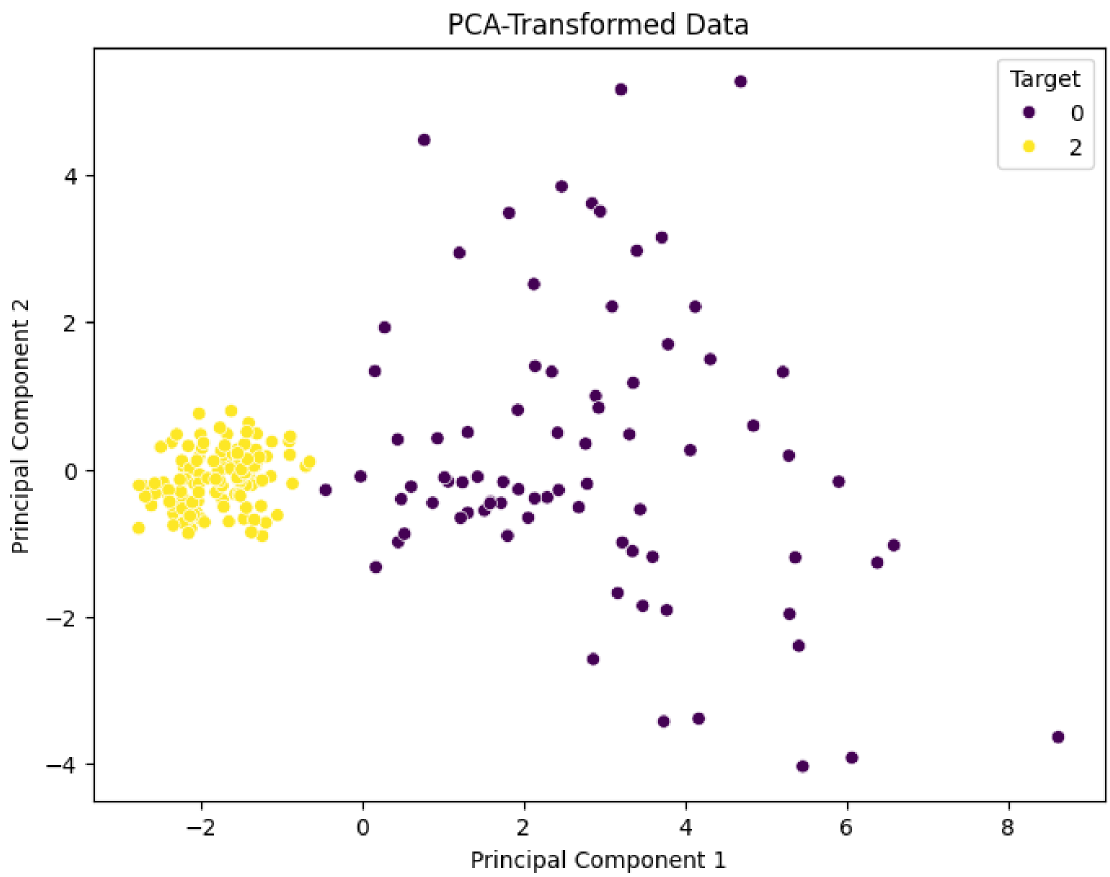


[ ]:
```
# Clustering Algorithm
```

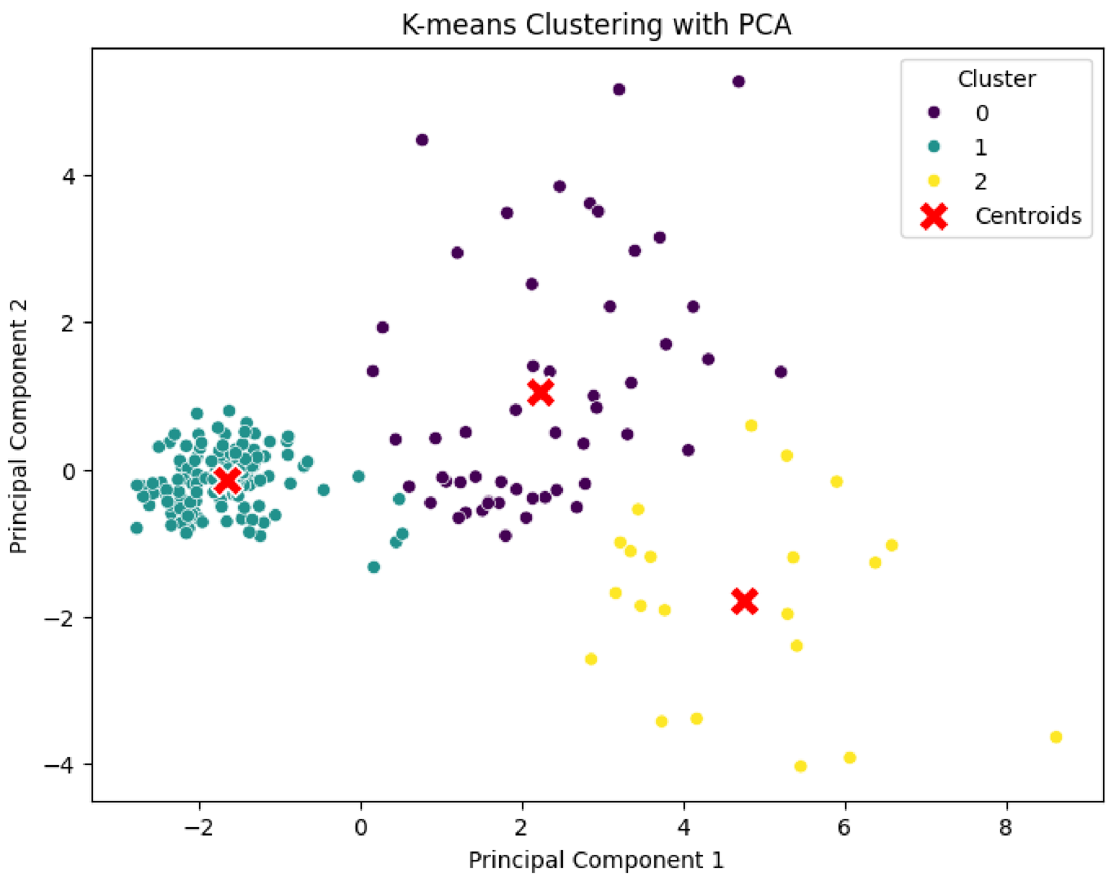
[30]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
from collections import Counter

# Load your dataset (replace 'data.csv' with your actual file path)
df = pd.read_csv('/home/kidney_disease.csv')

# Replace '\t?' with NaN for missing values
df.replace('\t?', np.nan, inplace=True)

# Convert non-numeric columns to numeric
cat_columns = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe',␣
 ↪'ane', 'classification']
df[cat_columns] = df[cat_columns].fillna(df.mode().iloc[0])
df[cat_columns] = df[cat_columns].apply(LabelEncoder().fit_transform)
```

```
# Drop rows with missing values (NaN)
df.dropna(inplace=True)

# Separate features (X) and target (y)
X = df.drop(columns=['classification'])  # Adjust 'classification' to your␣
 ↪target
y = df['classification']

# Scale the numeric features using StandardScaler
numeric_columns = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod',␣
 ↪'pot', 'hemo', 'pcv', 'wc', 'rc']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[numeric_columns])

# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)  # Specify the number of principal components
X_pca = pca.fit_transform(X_scaled)

# Visualize the PCA-transformed data
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=y, palette='viridis')
plt.title('PCA-Transformed Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Target')
plt.show()

# Determine the number of clusters (e.g., based on the visualization)
k = 3

# Apply K-means clustering
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X_pca)

# Get cluster labels
cluster_labels = kmeans.labels_

# Calculate and visualize cluster centroids
cluster_centers = kmeans.cluster_centers_
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=cluster_labels,␣
 ↪palette='viridis')
sns.scatterplot(x=cluster_centers[:, 0], y=cluster_centers[:, 1], color='red',␣
 ↪marker='X', s=200, label='Centroids')
plt.title('K-means Clustering with PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

```
plt.legend(title='Cluster')
plt.show()

# Calculate confusion matrix
conf_mat = confusion_matrix(y, cluster_labels)
print("Confusion Matrix:")
print(conf_mat)

# Calculate purity of clusters
def calculate_cluster_purity(labels_true, labels_pred):
    cluster_purity = {}
    for cluster in np.unique(labels_pred):
        labels_in_cluster = labels_true[labels_pred == cluster]
        most_common_label_count = Counter(labels_in_cluster).
 ↪most_common(1)[0][1]
        cluster_purity[cluster] = most_common_label_count / 
 ↪len(labels_in_cluster)
    overall_purity = sum(cluster_purity.values()) / len(cluster_purity)
    return cluster_purity, overall_purity

# Calculate purity for the clusters
cluster_purity, overall_purity = calculate_cluster_purity(y, cluster_labels)
print("\nCluster Purity:")
print(cluster_purity)
print("Overall Purity:", overall_purity)
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```




```
Confusion Matrix:
[[ 52   6  21]
 [  0   0   0]
 [  0 124   0]]

Cluster Purity:
{0: 1.0, 1: 0.9538461538461539, 2: 1.0}
Overall Purity: 0.9846153846153847
```

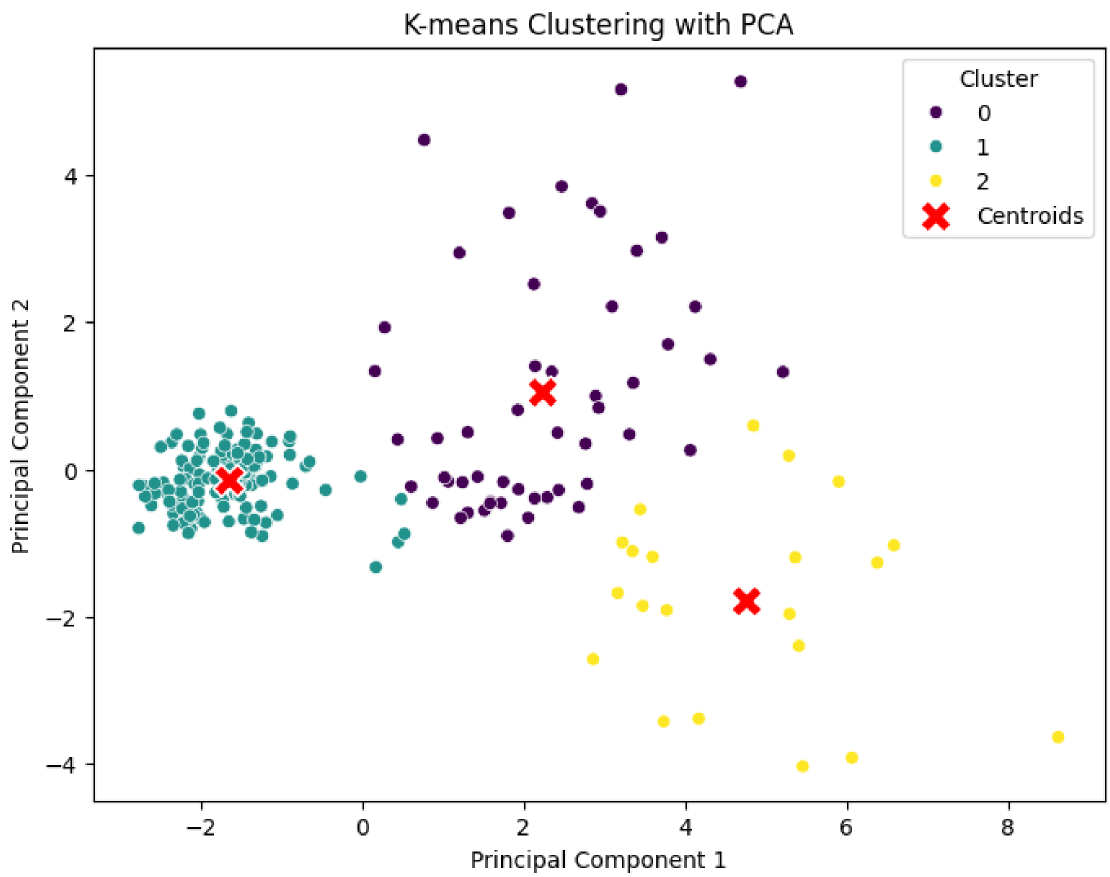
[33]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, confusion_matrix
from collections import Counter
```

```
# Load your dataset (replace 'data.csv' with your actual file path)
df = pd.read_csv('/home/kidney_disease.csv')

# Replace '\t?' with NaN for missing values
df.replace('\t?', np.nan, inplace=True)

# Convert non-numeric columns to numeric
cat_columns = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe',␣
 ↪'ane', 'classification']
df[cat_columns] = df[cat_columns].fillna(df.mode().iloc[0])
df[cat_columns] = df[cat_columns].apply(LabelEncoder().fit_transform)

# Drop rows with missing values (NaN)
df.dropna(inplace=True)

# Separate features (X) and target (y)
X = df.drop(columns=['classification'])  # Adjust 'classification' to your␣
 ↪target
y = df['classification']

# Scale the numeric features using StandardScaler
numeric_columns = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod',␣
 ↪'pot', 'hemo', 'pcv', 'wc', 'rc']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[numeric_columns])

# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)  # Specify the number of principal components
X_pca = pca.fit_transform(X_scaled)

# Determine the number of clusters (e.g., based on the visualization)
k = 3

# Apply K-means clustering
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X_pca)

# Get cluster labels
cluster_labels = kmeans.labels_

# Calculate and visualize cluster centroids
cluster_centers = kmeans.cluster_centers_
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=cluster_labels,␣
 ↪palette='viridis')
sns.scatterplot(x=cluster_centers[:, 0], y=cluster_centers[:, 1], color='red',␣
 ↪marker='X', s=200, label='Centroids')
```

```
plt.title('K-means Clustering with PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.show()

# Calculate confusion matrix
conf_mat = confusion_matrix(y, cluster_labels)
print("Confusion Matrix:")
print(conf_mat)

# Calculate accuracy and error rate
def calculate_accuracy_and_error_rate(labels_true, labels_pred):
    correct = np.sum(labels_true == labels_pred)
    accuracy = correct / len(labels_true)
    error_rate = 1 - accuracy
    return accuracy, error_rate

accuracy, error_rate = calculate_accuracy_and_error_rate(y, cluster_labels)
print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```




```
Confusion Matrix:
[[ 52   6  21]
 [  0   0   0]
 [  0 124   0]]
Accuracy: 0.2561576354679803
Error Rate: 0.7438423645320197
```

[87]:
```
from IPython.display import Image

Image('/home/ML5.png')
```

[87]:

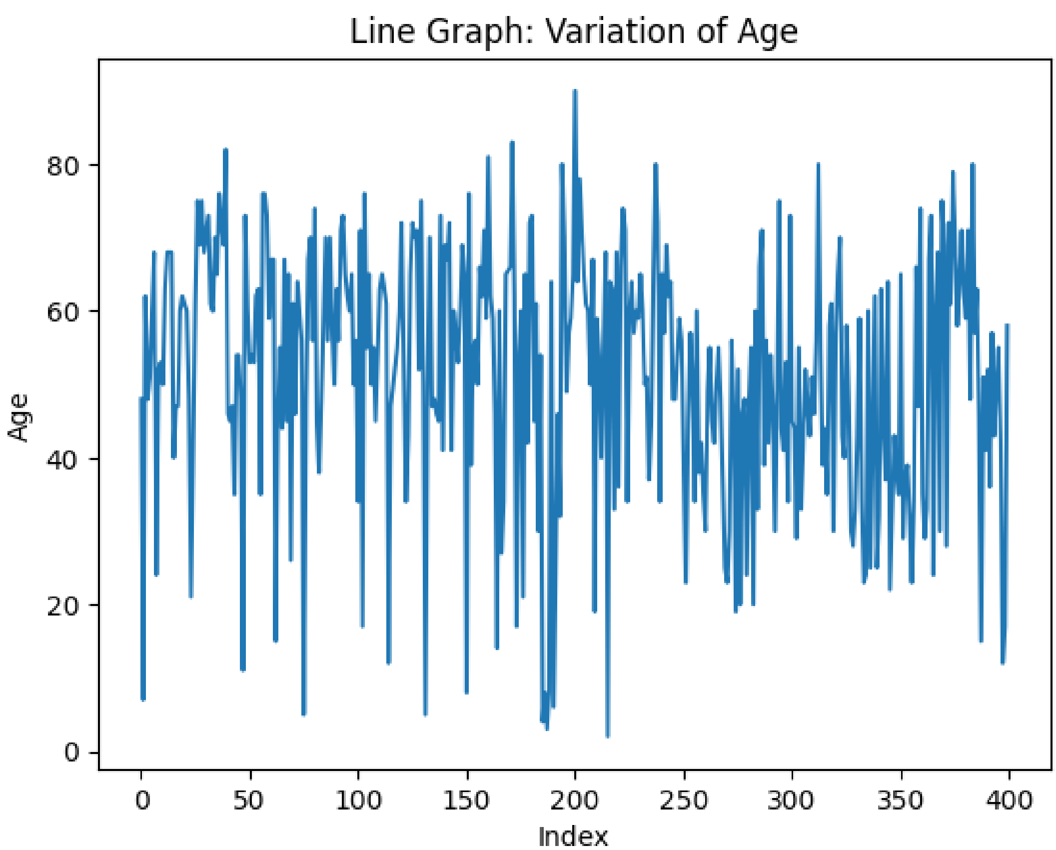
3. Data visualization techniques (2 marks)

4. Calculation of accuracy and error / Purity of cluster (1 mark)

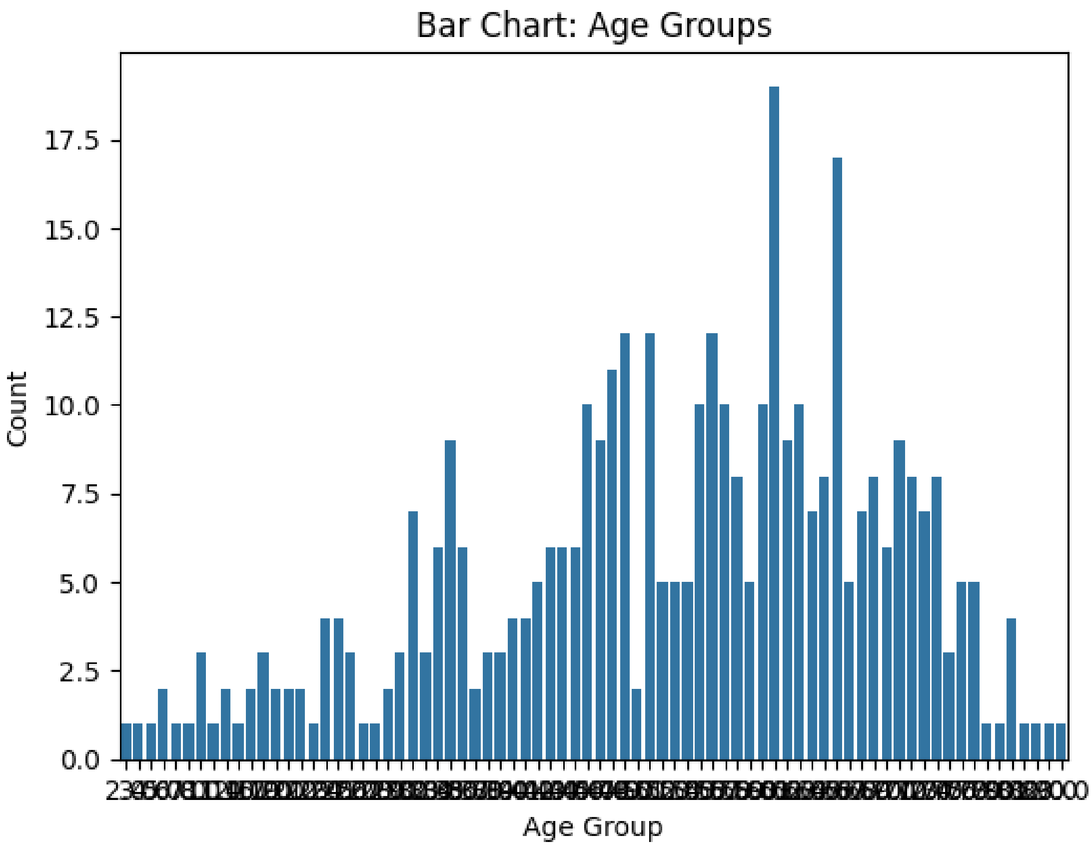
[88]:
```
# 3. Data visualization techniques
# 4. Calculation of accuracy and error / Purity of cluster
```

[89]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[90]:
```
# Line Graph
sns.lineplot(x=df.index, y='age', data=df)
plt.title('Line Graph: Variation of Age')
plt.xlabel('Index')
plt.ylabel('Age')
plt.show()
```



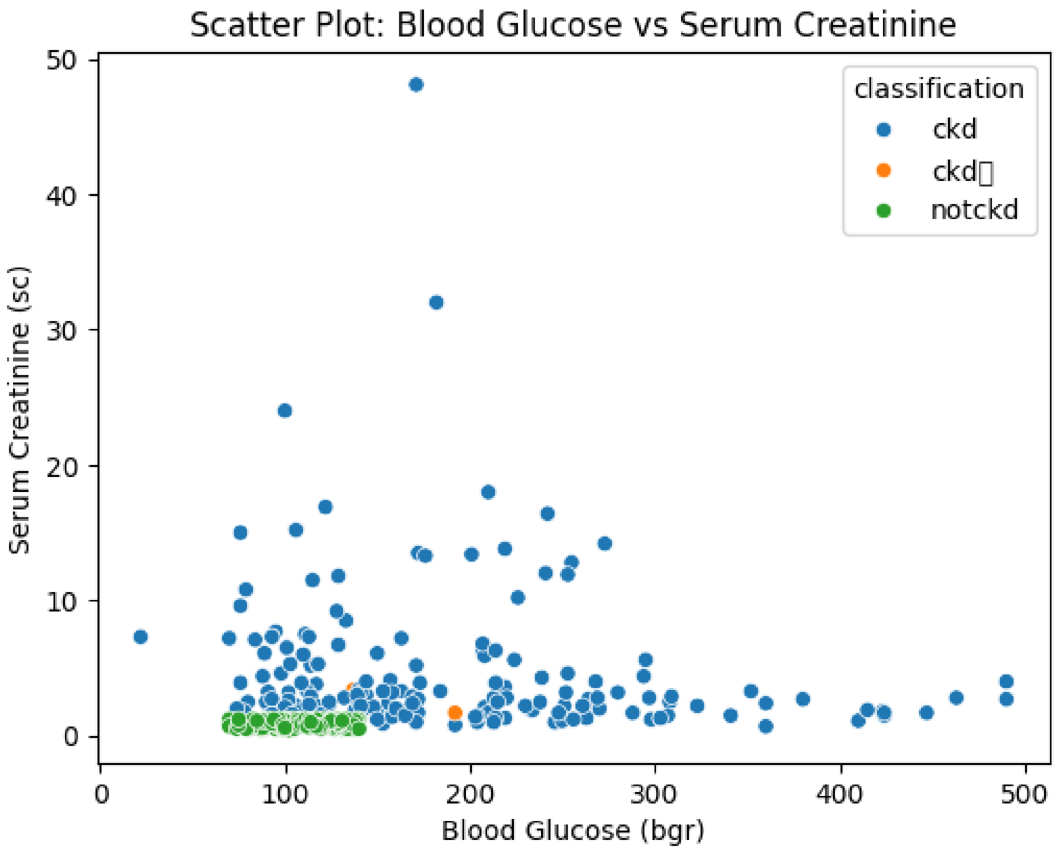
[91]:
```
# Bar Chart For Numerical Variable
sns.countplot(x='age', data=df)
plt.title('Bar Chart: Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.show()
```



[92]:

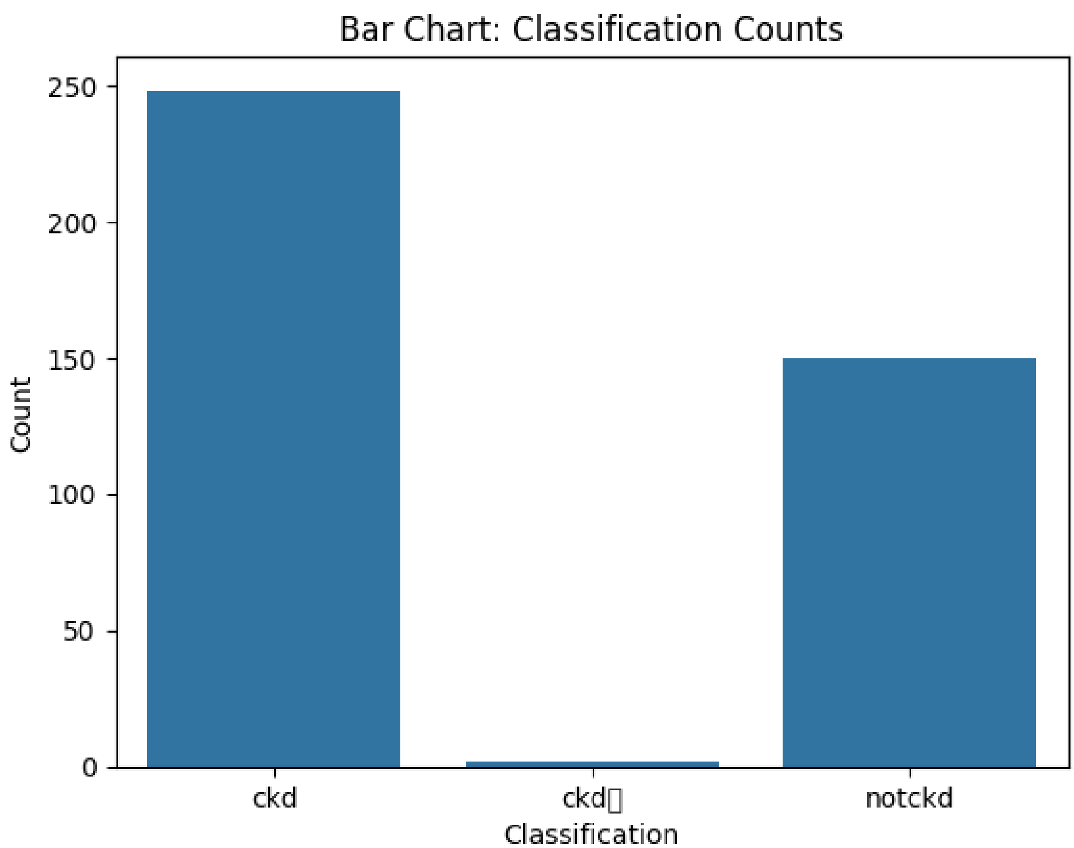
```
# Scatter Plot
sns.scatterplot(x='bgr', y='sc', data=df, hue='classification')
plt.title('Scatter Plot: Blood Glucose vs Serum Creatinine')
plt.xlabel('Blood Glucose (bgr)')
plt.ylabel('Serum Creatinine (sc)')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151:
UserWarning: Glyph 9 (   ) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```



[93]:
```
# Bar Chart For Categorical Variable
sns.countplot(x='classification', data=df)
plt.title('Bar Chart: Classification Counts')
plt.xlabel('Classification')
plt.ylabel('Count')
plt.show()
```

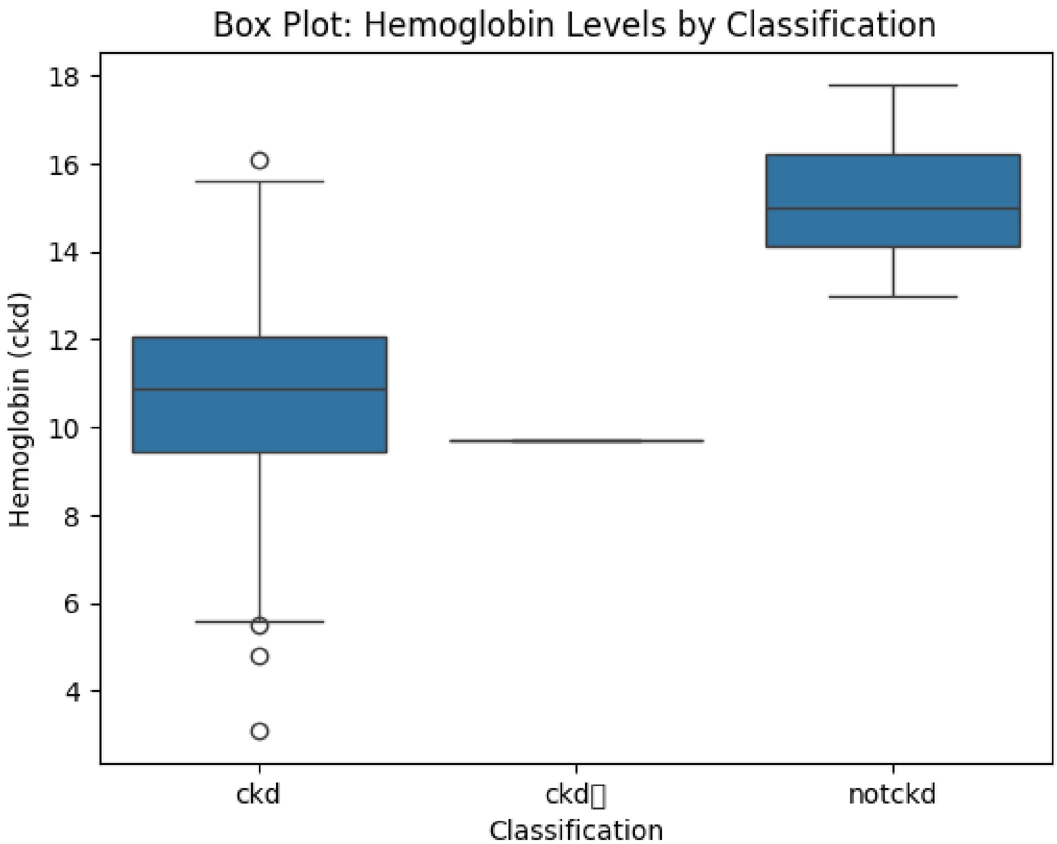
```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151:
UserWarning: Glyph 9 (	) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```



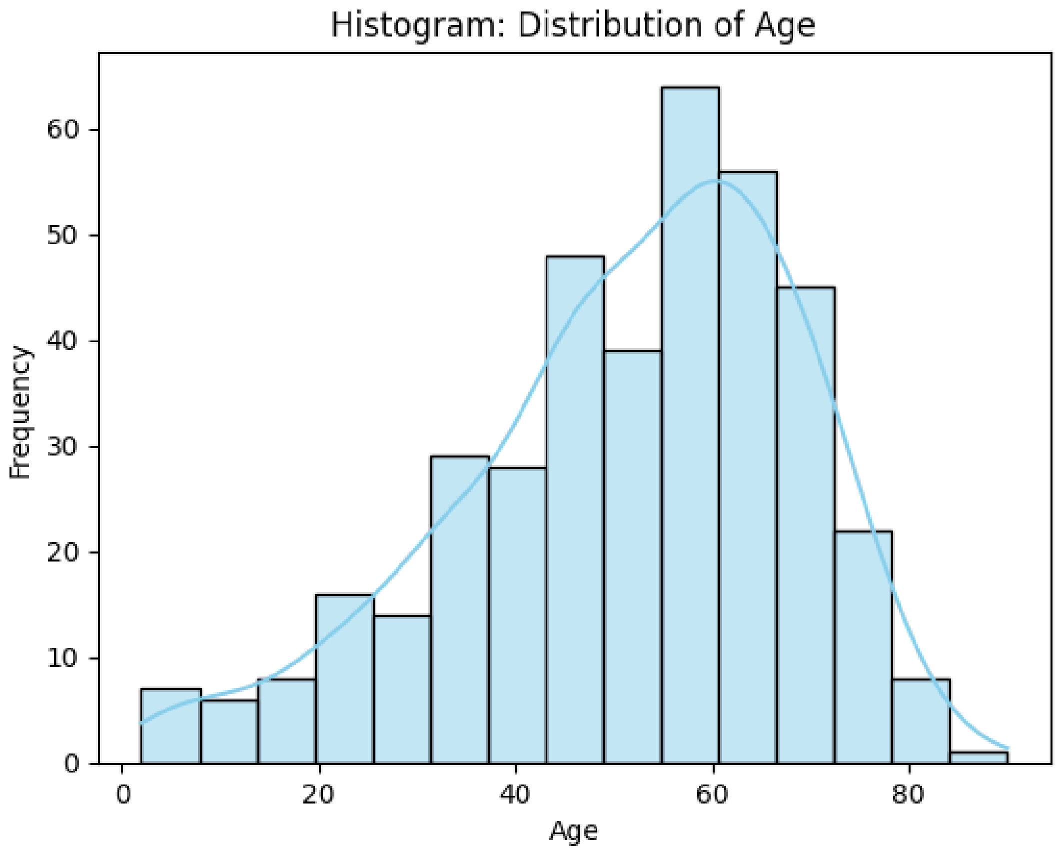
[94]:
```
#  Distribution Plots:
```

[95]:
```
# Boxplots
sns.boxplot(x='classification', y='hemo', data=df)
plt.title('Box Plot: Hemoglobin Levels by Classification')
plt.xlabel('Classification')
plt.ylabel('Hemoglobin (ckd)')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151:
UserWarning: Glyph 9 (	) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```



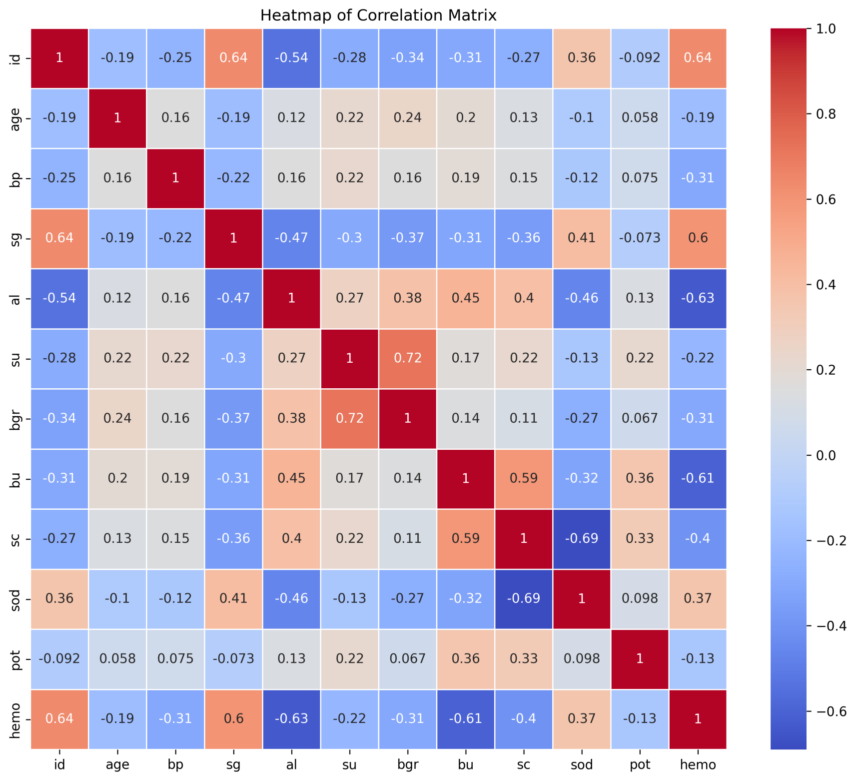
[96]:
```
# Histograms
sns.histplot(df['age'].dropna(), kde=True, color='skyblue')
plt.title('Histogram: Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



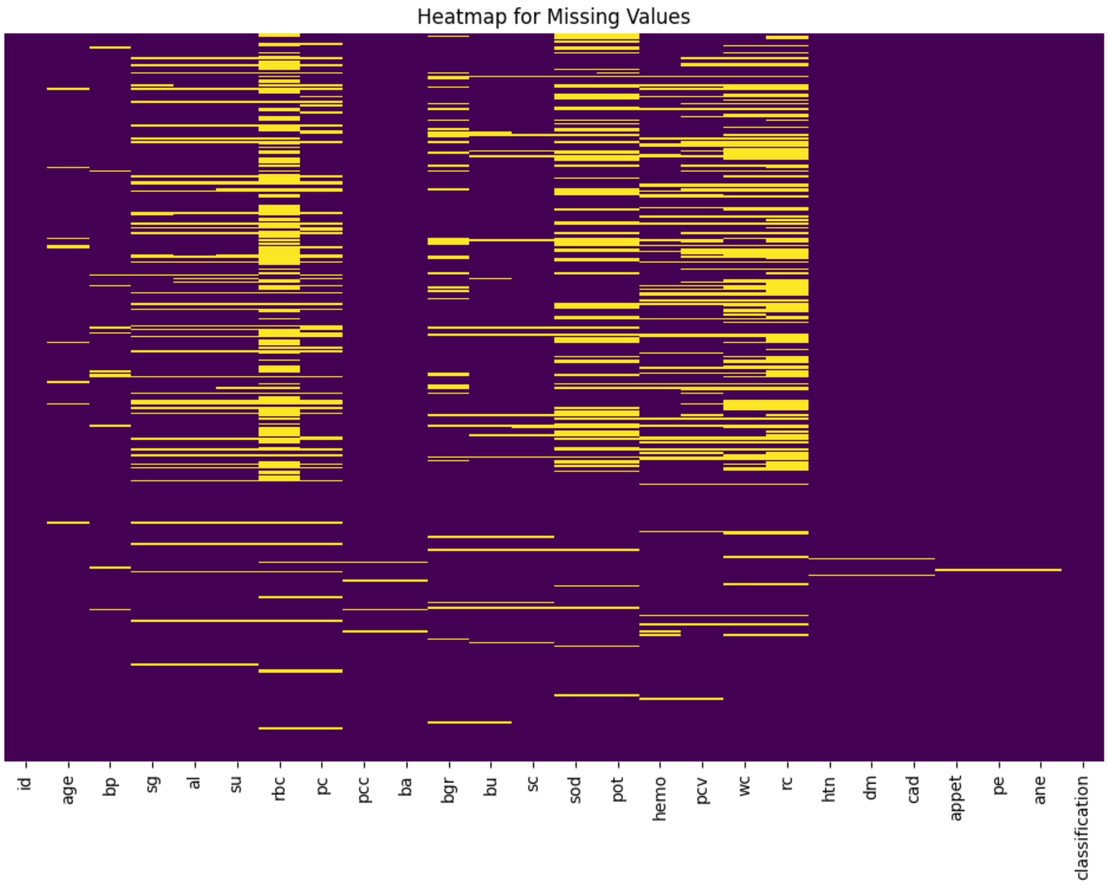
[97]:
```
# Heat maps
```

[101]:
```
# Selecting only numerical columns
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Heatmap for Correlations
plt.figure(figsize=(12, 10))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Heatmap of Correlation Matrix')
plt.savefig('correlation_heatmap.png', dpi=300)
plt.close()

# Display the correlation heatmap
Image('correlation_heatmap.png')
```

[101]:

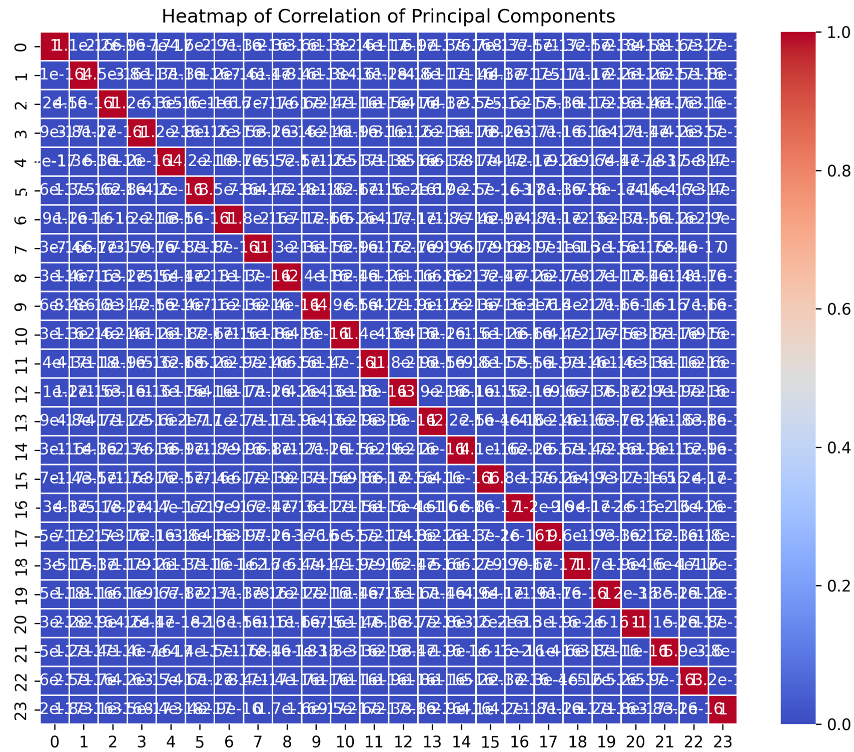


[114]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Heatmap for Missing Values
plt.figure(figsize=(12, 8))
sns.heatmap(df.isnull(), cmap='viridis', yticklabels=False, cbar=False)
plt.title('Heatmap for Missing Values')
plt.show()
```

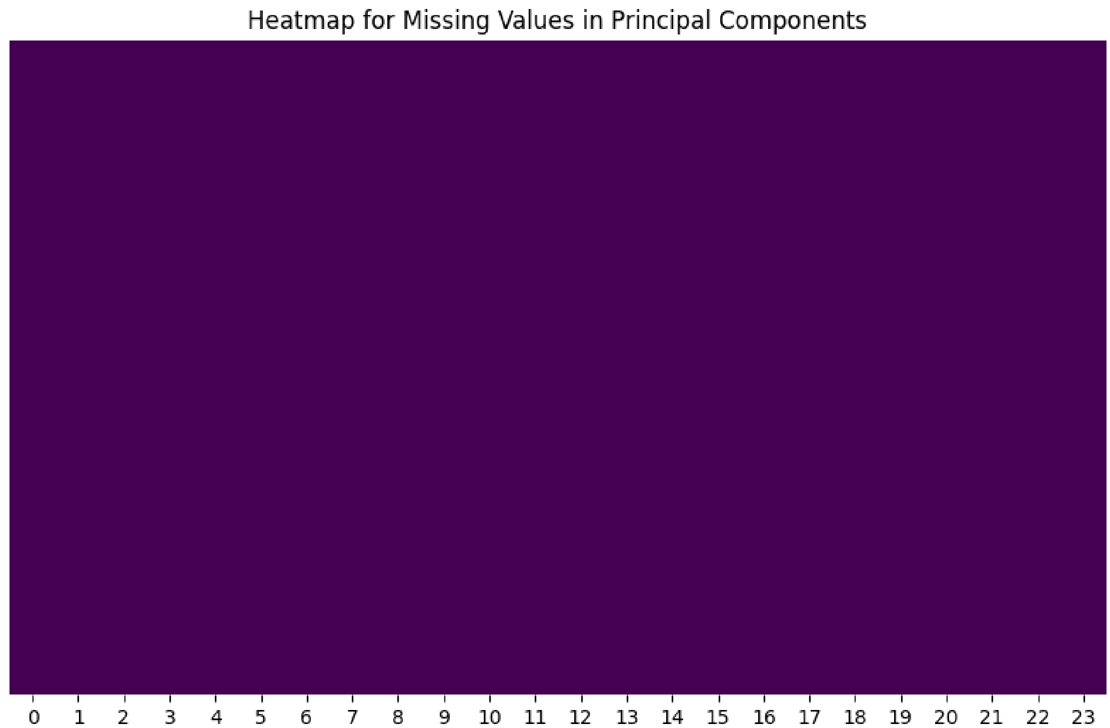


[106]:
```
from sklearn.decomposition import PCA

# Applying PCA
pca = PCA()
X_train_pca = pca.fit_transform(X_train)

# Calculate the correlation matrix for principal components
pc_corr = pd.DataFrame(X_train_pca).corr()

# Heatmap for Correlation of Principal Components
plt.figure(figsize=(10, 8))
sns.heatmap(pc_corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Heatmap of Correlation of Principal Components')
plt.savefig('pc_correlation_heatmap.png', dpi=300)
plt.close()

# Display the correlation heatmap for principal components
Image('pc_correlation_heatmap.png')
```

[106]:



[107]:
```python
# Heatmap for Missing Values in Principal Components
plt.figure(figsize=(10, 6))
sns.heatmap(pd.DataFrame(X_train_pca).isnull(), cmap='viridis',␣
 ↪yticklabels=False, cbar=False)
plt.title('Heatmap for Missing Values in Principal Components')
plt.show()
```


Heatmap for Missing Values in Principal Components
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23