Final Project: PCA and Clustering

- **Full Name** = Sudarshan Naicker
- **UCID** = 30162797 ***

The purpose of this assignment is to practice using PCA and clustering techniques on a given dataset.

For this assignment, in addition to your .ipynb file, please also attach a PDF file. To generate this PDF file, you can use the print function (located under the "File" within Jupyter Notebook). Name this file ENGG444_Final_Project_yourUCID.pdf (this name is similar to your main .ipynb file). We will evaluate your assignment based on the two files and you need to provide both.

Question	Point(s)
1. Principle Component Analysis (PCA)	
1.1	3
1.2	2
1.3	2
1.4	3
1.5	6
1.6	2
2. Pipeline and Modeling	
2.1	3
2.2	2
2.3	2
2.4	3
3. Bonus Question	**2**
Total	28

Data

The data on this page pertains to a study on wheat kernels, specifically focusing on the geometrical properties of kernels from three different wheat varieties: Kama, Rosa, and Canadian. Here's a summary of the key points:

- **Dataset Characteristics**: The data is multivariate and real-valued, used for classification and clustering tasks in biology.
- **Measurement Technique**: A soft X-ray technique was employed for high-quality visualization of the internal kernel structure, which is non-destructive and cost-effective compared to other methods.

- **Geometric Parameters**: Seven parameters were measured for each kernel: area (A), perimeter (P), compactness (C = 4*pi*A/P^2), length, width, asymmetry coefficient, and length of kernel groove.
- **Research Purpose**: The dataset facilitates the analysis of features in X-ray images of wheat kernels and can be applied to various statistical and machine learning tasks.

This dataset was collected for an experiment conducted at the Institute of Agrophysics of the Polish Academy of Sciences in Lublin and has been cited in several research papers for its application in feature analysis and classification algorithms.

```
# Download the zip file using wget
!wget -N "https://archive.ics.uci.edu/static/public/236/seeds.zip"

# Unzip wine.data from the downloaded zip file
#!unzip -o seeds.zip seeds_dataset.txt

# Remove the downloaded zip file after extraction
#!rm -r seeds.zip
```

```
--2024-04-09 15:26:40--
https://archive.ics.uci.edu/static/public/236/seeds.zip
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|
128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: 'seeds.zip'

     0K .........
47.7M=0s

Last-modified header missing -- time-stamps turned off.
2024-04-09 15:26:41 (47.7 MB/s) - 'seeds.zip' saved [9432]
```

```
# 1. area A,
# 2. perimeter P,
# 3. compactness C = 4*pi*A/P^2,
# 4. length of kernel,
# 5. width of kernel,
# 6. asymmetry coefficient
# 7. length of kernel groove.

# https://archive.ics.uci.edu/dataset/236/seeds

import pandas as pd

data = pd.read_csv('seeds_dataset.txt', sep = '\s+', header = None)
data.columns = ['Area', 'Perimeter', 'Compactness',
                'Length of Kernel', 'Width of Kernel',
                'Asymmetry Coefficient', 'Length of Kernel Groove',
'Type']
display(data)

      Area  Perimeter  Compactness  Length of Kernel  Width of Kernel
\
0    15.26      14.84       0.8710             5.763            3.312

1    14.88      14.57       0.8811             5.554            3.333

2    14.29      14.09       0.9050             5.291            3.337

3    13.84      13.94       0.8955             5.324            3.379

4    16.14      14.99       0.9034             5.658            3.562

..     ...        ...          ...               ...              ...

205  12.19      13.20       0.8783             5.137            2.981

206  11.23      12.88       0.8511             5.140            2.795

207  13.20      13.66       0.8883             5.236            3.232

208  11.84      13.21       0.8521             5.175            2.836

209  12.30      13.34       0.8684             5.243            2.974


     Asymmetry Coefficient  Length of Kernel Groove  Type
0                    2.221                    5.220     1
1                    1.018                    4.956     1
2                    2.699                    4.825     1
3                    2.259                    4.805     1
4                    1.355                    5.175     1
..                     ...                      ...   ...
205                  3.631                    4.870     3
206                  4.325                    5.003     3
207                  8.315                    5.056     3
208                  3.598                    5.044     3
209                  5.637                    5.063     3

[210 rows x 8 columns]
```

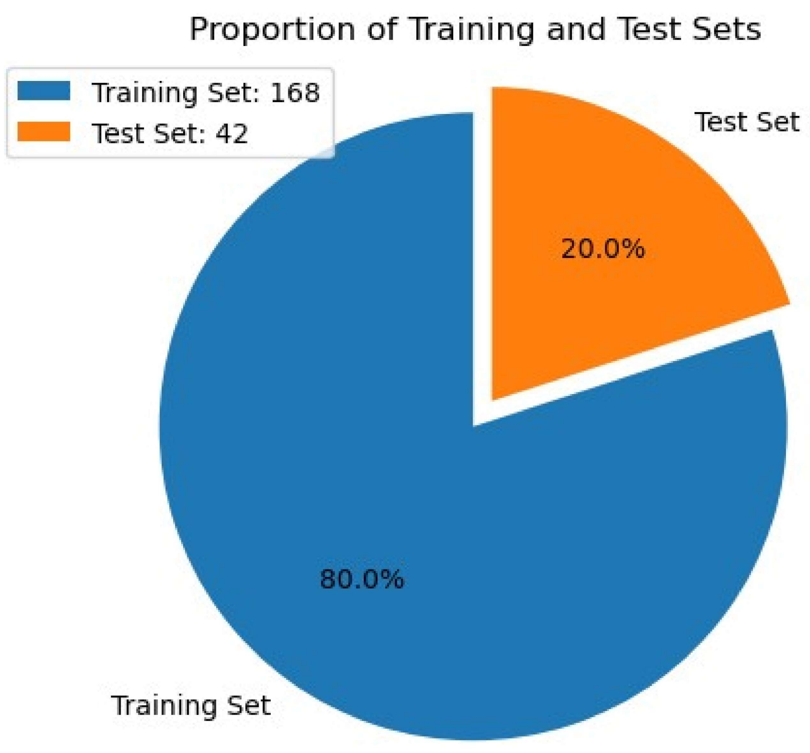
1. Principle Component Analysis (PCA)

1.1 Preprocessing (3 Points)

- **Split the data into X and y** (0.5 Point)
  - Assign the features to `X` and the target variable to `y`.
- **Stratified Split of X and y into Train and Test Sets** (0.5 Point)

    - Utilize stratification to ensure representative distribution of classes while splitting.
- **Plot Train and Test Proportions in a Pie Chart** (2 Points)
    - The pie chart should include:
        - Labels indicating 'Training Set' and 'Test Set'.
        - A title for the chart.
        - Proportion percentages for the Training and Test sets displayed on each slice of the pie.
        - The number of entries within the Training and Test sets shown below the corresponding percentage.

```python
# 1.1
# Add necessary code here.
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Split the data into X and y
X = data.drop('Type', axis=1)
y = data['Type']

# Split X and y into Train and Test Sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)


# Plot Train and Test Proportions in a Pie Chart
sizes = [len(y_train), len(y_test)]
labels = ['Training Set', 'Test Set']

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=(0, 0.1), labels=labels, autopct='%1.1f%%',
startangle=90)
ax1.axis('equal')


plt.title('Proportion of Training and Test Sets')
plt.legend(labels=[f'Training Set: {sizes[0]}', f'Test Set:
{sizes[1]}'], loc="upper left")
plt.show()
```



Answer:

- **1.1** ....

1.2 Scaling the Data (2 Points)

To ensure that our preprocessing pipeline optimizes the performance of our machine learning model, we need to scale the data appropriately.

- **Selecting an Appropriate Scaler**:
  - Explain your choice of scaler for the dataset. (1 Points)
  - Justify your decision based on the characteristics of the data and the requirements of the algorithm being used. (1 Points)

Answer:

- **1.2** ....
- I have chosen Standard Scaler, it is a technique for transforming numerical data to have a mean of zero and a standard deviation of one. This scaling ensures that all features are scaled to a standard range, which is crucial for models sensitive to feature scales. it is calculate by using the formula: `z = (x - μ) / σ` where x is original data, μ is mean of the data and σ is the standard deviation of the data.
- The dataset contains features like area, perimeter, and kernel length that likely have different ranges and units of measurement. Scaling ensures these features are on a

similar scale, preventing any single feature from disproportionately influencing the model's predictions.

1.3 Model Selection and Justification (2 Points)

- **Choose an Appropriate Machine Learning Model**:
  - Identify the model that you believe is most suitable for the dataset.
  - Provide a justification for your choice based on the dataset's characteristics.

```
# 1.3
# Add necessary code here.
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)

# Fit the model to the training data
rf_classifier.fit(X_train, y_train)

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

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of the Random Forest Classifier on the test set:
{accuracy:.2f}")
```

```
Accuracy of the Random Forest Classifier on the test set: 0.90
```

Answer:

- **1.3** ....

- I have chosen `Random Forest Classifier` it is an ensemble learning method that operates by constructing a collection of decision trees and outputting the result by either averaging (Regression) or majority voting (Classification) of all trees. The trees are averaged out and make it perform way better than an individual tree, it can handle non-linear features well and requires minimal preprocessing hence I have chosen RF.

1.4 Hyperparameter Optimization with Grid Search (3 Points)

- **Set Up the Grid Search**:
  - Construct a pipeline that incorporates the selected scaler from part 1.2 to standardize the data.
  - Execute a grid search within this pipeline to identify the best hyperparameter settings for your chosen model.

  - Provide a broad and varied range of hyperparameter values to ensure a thorough search.

```
# 1.4
# Add necessary code here.
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV


# Pipeline with StandardScaler and RF
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('rf', rf_classifier)
])

param_grid = {
    'rf__n_estimators': [100, 300],
    'rf__max_depth': [5, 10, 15],
    'rf__min_samples_leaf': [1],
    'rf__min_samples_split': [5],
}

grid_search = GridSearchCV(pipeline, param_grid, cv=5,
scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train)

# Outputs of the best parameters and the best score
print("\nBest parameters found:", grid_search.best_params_)
print("Best cross-validation score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

Best parameters found: {'rf__max_depth': 10, 'rf__min_samples_leaf':
1, 'rf__min_samples_split': 5, 'rf__n_estimators': 300}
Best cross-validation score: 0.9171122994652408
```

Answer:

- **1.4** ....

1.5 Dimensionality Reduction and Model Optimization (6 Points)

- **Dimensionality Reduction Choice** (2 Points):
  - Choose between PCA and t-SNE for reducing the dataset to two dimensions.
  - Justify your selection based on the characteristics of the seeds dataset.
- **Implement Dimensionality Reduction** (2 Points):
  - Apply the chosen dimensionality reduction technique to the seeds dataset.
  - Reduce the dataset to two dimensions as required.
- **Model Optimization on Reduced Data** (2 Points):
  - Redo the grid search from part 1.4 using the two-dimensional data.

    - Compare the model's performance with the original higher-dimensional data.

```
# 1.5
# Add necessary code here.
from sklearn.decomposition import PCA

# Applying PCA to reduce the dataset to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Split the reduced data into train and test sets
X_train_pca, X_test_pca, y_train_pca, y_test_pca =
train_test_split(X_pca, y, test_size=0.2, stratify=y, random_state=42)

# Redo GridSearch
grid_search_pca = GridSearchCV(pipeline, param_grid, cv=5,
scoring='accuracy', verbose=1)
grid_search_pca.fit(X_train_pca, y_train_pca)

# Outputs of the best parameters and  best score after pca
print("\nBest parameters (PCA reduced data):",
grid_search_pca.best_params_)
print("Best cross-validation score (PCA reduced data):",
grid_search_pca.best_score_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

Best parameters (PCA reduced data): {'rf__max_depth': 10,
'rf__min_samples_leaf': 1, 'rf__min_samples_split': 5,
'rf__n_estimators': 100}
Best cross-validation score (PCA reduced data): 0.8985739750445634
```

Answer:

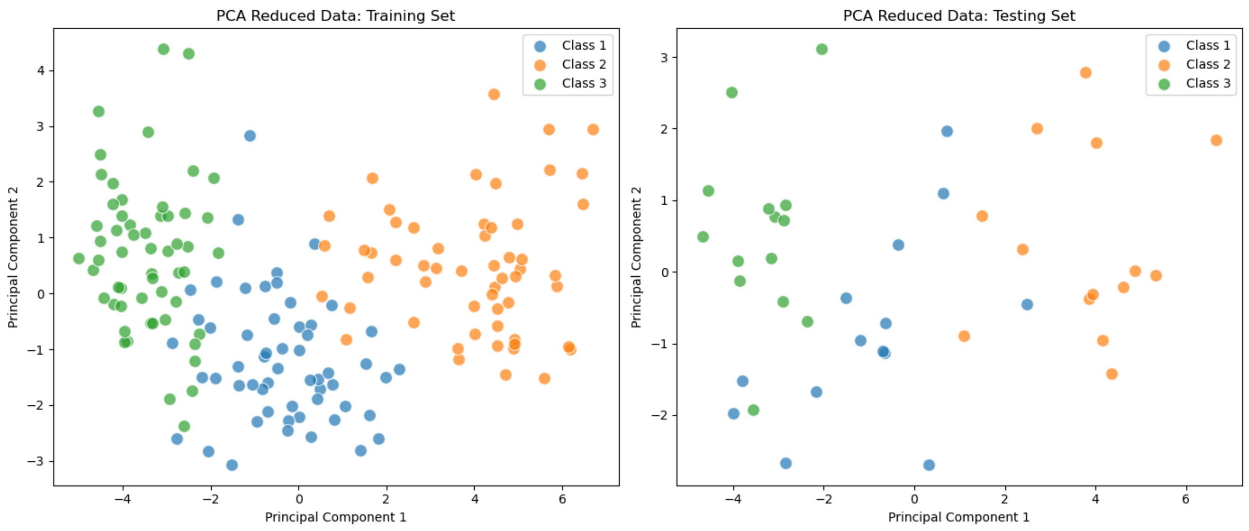
- **1.5** ....

1.6 Visualizing Reduced Dimensionality Data (2 Points)

- **Create a 2D Scatter Plot for Training and Testing Sets**:
    - Generate 1-row-two-column subplots for scatter plots for the two-dimensional training and testing data obtained from part 1.5.
    - Clearly label the x-axis and y-axis for both plots.
    - Include a legend in each plot that distinctly represents the distribution of the three classes (you can use different shapes and colors to represent different classes).

```
# 1.6
# Add necessary code here.
import matplotlib.pyplot as plt
import numpy as np
```

```
# Scatter plot for Training Data
fig, ax = plt.subplots(1, 2, figsize=(14, 6))  # 1-row-2-column
subplot

# Loop through each class for the training set
for class_value in np.unique(y_train_pca):
    ix = np.where(y_train_pca == class_value)  # Find rows of this
class
    ax[0].scatter(X_train_pca[ix, 0], X_train_pca[ix, 1],
label=f"Class {class_value}", alpha=0.7, edgecolors='w', s=100)

ax[0].set_title('PCA Reduced Data: Training Set')
ax[0].set_xlabel('Principal Component 1')
ax[0].set_ylabel('Principal Component 2')
ax[0].legend()

# Loop through each class for the testing set
for class_value in np.unique(y_test_pca):
    ix = np.where(y_test_pca == class_value)  # Find rows of this
class
    ax[1].scatter(X_test_pca[ix, 0], X_test_pca[ix, 1], label=f"Class
{class_value}", alpha=0.7, edgecolors='w', s=100)

ax[1].set_title('PCA Reduced Data: Testing Set')
ax[1].set_xlabel('Principal Component 1')
ax[1].set_ylabel('Principal Component 2')
ax[1].legend()

plt.tight_layout()  # Adjust layout to not overlap subplots
plt.show()
```



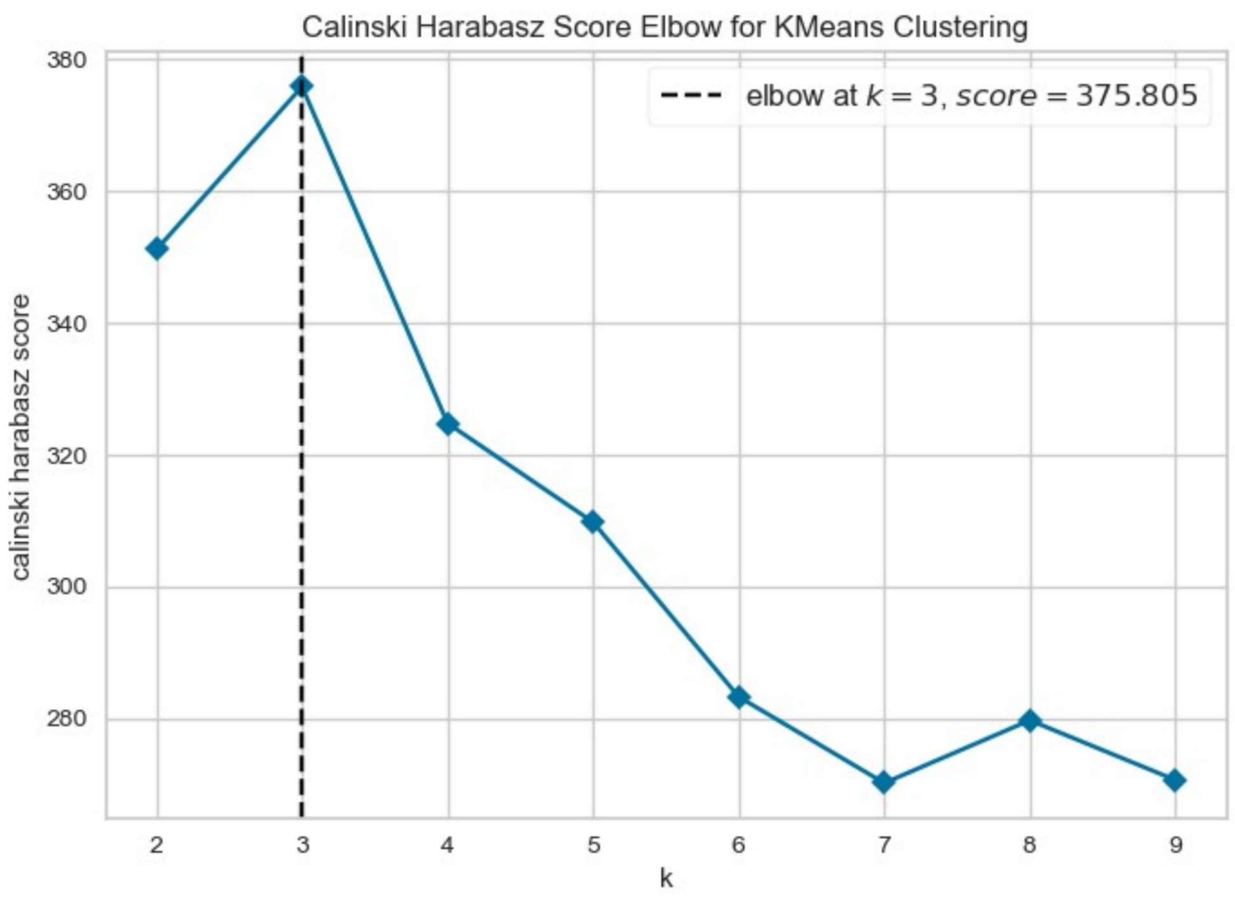
2. Clustering and Visualization of the Seeds Dataset

2.1 Create a Pipeline for Scaling and K-Means Clustering (3 Points)

- Construct a pipeline that includes a scaler and the K-Means clustering algorithm.
- Use the `KelbowVisualizer` with `metric='calinski_harabasz'` from Yellowbrick to determine the optimal number of clusters, `k`.
- Explain the results of the `KelbowVisualizer`.

```
# 2.1
# Add necessary code here.
import os
os.environ['OMP_NUM_THREADS'] = '1'
print("OMP_NUM_THREADS:", os.getenv("OMP_NUM_THREADS"))

from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer

pipeline_2 = Pipeline([
    ('scaler', StandardScaler()),
    ('kmeans', KMeans(n_init='auto', random_state=42))
])

# Initialize the KElbowVisualizer with the KMeans model and the
calinski_harabasz metric
visualizer = KElbowVisualizer(pipeline_2.named_steps['kmeans'],
k=(2,10), metric='calinski_harabasz', timings=False)

# Fit the data to the visualizer
visualizer.fit(X)
visualizer.show()
```

```
OMP_NUM_THREADS: 1

c:\Users\SUDARSHAN\anaconda3\envs\ensf-ml\Lib\site-packages\sklearn\
cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory
leak on Windows with MKL, when there are less chunks than available
threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\SUDARSHAN\anaconda3\envs\ensf-ml\Lib\site-packages\sklearn\
cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory
leak on Windows with MKL, when there are less chunks than available
threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\SUDARSHAN\anaconda3\envs\ensf-ml\Lib\site-packages\sklearn\
cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory
leak on Windows with MKL, when there are less chunks than available
threads. You can avoid it by setting the environment variable
```



```
<Axes: title={'center': 'Calinski Harabasz Score Elbow for KMeans
Clustering'}, xlabel='k', ylabel='calinski harabasz score'>
```

Answer:

- **2.1** ....

The graph shows the highest Calinski-Harabasz score of 375.8 when k=3. Higher score means the clusters are dense and well-seperated i.e., at k=3 the three clusters provide a potentially optimal clustering. Any higher value of k results in lower score meaning that the algorithm is not able to assign the data points to the clusters well enough.

2.2 Label the Data Using the Optimal Number of Clusters (2 Points)

- Label the training data using the pipeline that includes both the scaler and K-Means with the optimal `k` found in part 2.1.

```
# 2.2
# Add necessary code here.

# Adjust the 'kmeans' step in the pipeline to the optimal number of
clusters (k=3)
pipeline_2.set_params(kmeans__n_clusters=3)
```

```
# Fit the pipeline to the training data to label it
pipeline_2.fit(X_train)

# Print the cluster labels for the training data
cluster_labels = pipeline_2.named_steps['kmeans'].labels_
print(cluster_labels[:10])
```

```
[0 2 2 0 0 1 2 1 1 1]

c:\Users\SUDARSHAN\anaconda3\envs\ensf-ml\Lib\site-packages\sklearn\
cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory
leak on Windows with MKL, when there are less chunks than available
threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

2.3 Dimensionality Reduction Using PCA (2 Points)

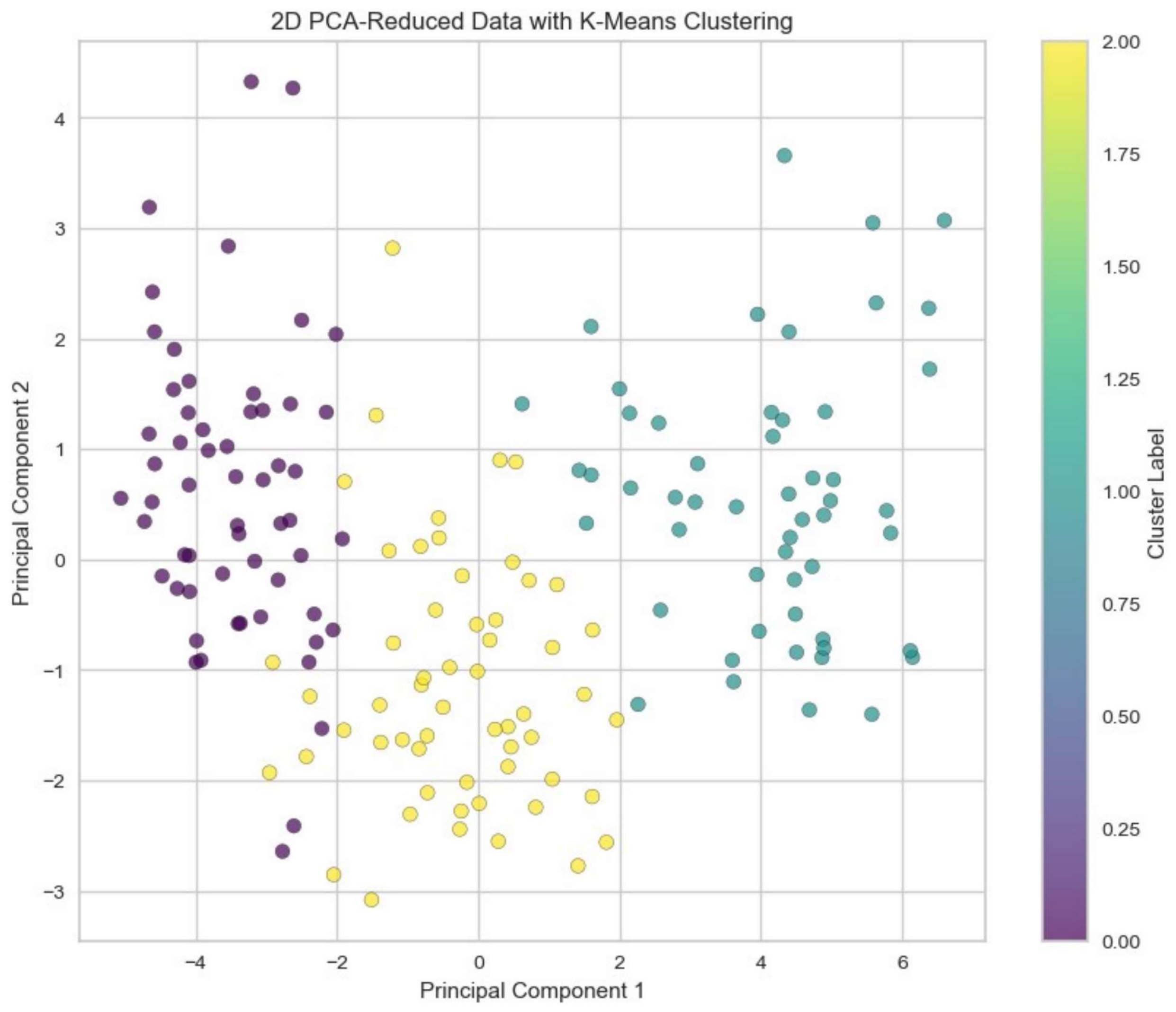
- Apply PCA to reduce the dimensionality of the dataset to 2D.

```
# 2.3
# Add necessary code here.
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train)
```

2.4 Plot the 2D Data with Cluster Labels (3 Points)

- Create a 2D scatter plot of the PCA-reduced data.
- Color the points using the labels obtained from K-Means clustering.

```
# 2.4
# Add necessary code here.
plt.figure(figsize=(10, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels,
cmap='viridis', edgecolor='k', alpha=0.7, s=50)

# Adding titles and labels
plt.title('2D PCA-Reduced Data with K-Means Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

# Adding a color bar to show the cluster labels
plt.colorbar(label='Cluster Label')

# Show the plot
plt.show()
```



Bonus Question: Interpretation of Clustering Results (2 Points)

- **Analyze and Interpret the Clustering Outcome**:
  - Based on the 2D PCA plot with K-Means clustering labels from part 2.4, provide an interpretation of the clustering results.
  - Discuss any patterns or insights observed from the plot, considering the distribution and overlap of clusters.

Answer:

- **Bonus Question** …………………

The clusters are reasonably well-separated, suggesting that the PCA transformation has retained significant structure from the higher-dimensional space, allowing for clear cluster formation.

- The yellow cluster is mainly found in the lower-middle part of the plot. It has a more compact spread compared to the other clusters.
- The purple cluster is primarily in the lower-left quadrant, it is mostly condensed but a few plots overlap the yellow clusters this may be because there may be certain points in the data that are not clearly distinguishable.
- The blue cluster occupies the upper right quadrant and is more dispersed than the other two, meaning greater variability within this cluster.