Final Project: PCA and Clustering

- **Full Name** =
- **UCID** =

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
```

```
--2024-04-10 05:40:43--  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... c
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘seeds.zip’

seeds.zip               [ <=>                ]   9.21K  --.-KB/s    in 0s

Last-modified header missing -- time-stamps turned off.
2024-04-10 05:40:43 (96.4 MB/s) - ‘seeds.zip’ saved [9432]

Archive:  seeds.zip
 extracting: seeds_dataset.txt
```

```
# 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	Asymmetry Coefficient	Length of Kernel Groove	Type
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1
...	...	...	...	...	...	...	...	...
205	12.19	13.20	0.8783	5.137	2.981	3.631	4.870	3
206	11.23	12.88	0.8511	5.140	2.795	4.325	5.003	3
207	13.20	13.66	0.8883	5.236	3.232	8.315	5.056	3
208	11.84	13.21	0.8521	5.175	2.836	3.598	5.044	3
209	12.30	13.34	0.8684	5.243	2.974	5.637	5.063	3

Next steps: Generate code with `data` View recommended plots

1. Principle Component Analysis (PCA)

1.1 Preprocessing (3 Points)

- **Split the data into X and y** (0.5 Point)

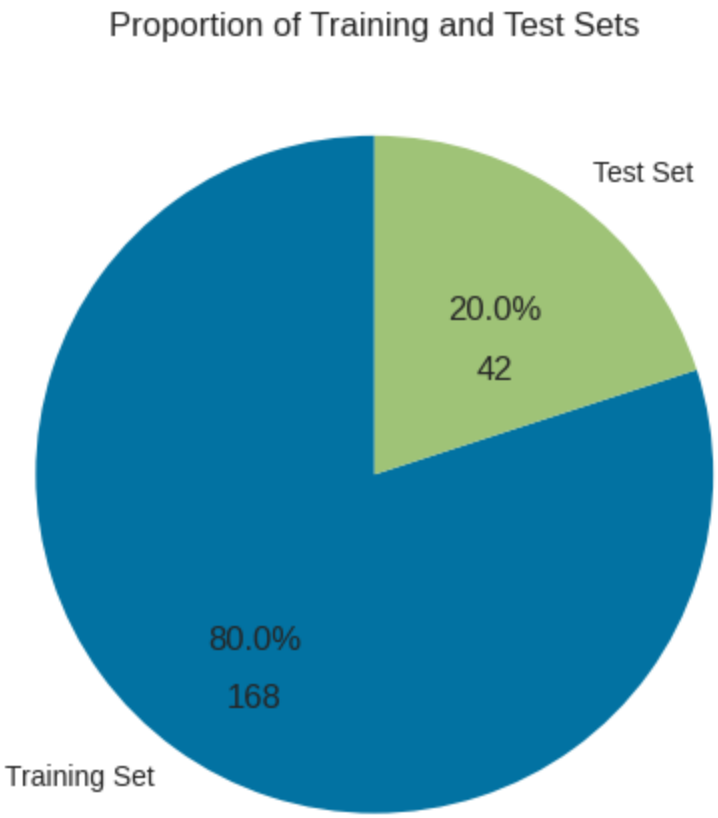
- Assign the features to `X` and the target variable to `y`.

- **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.

```
def plotTestTrain(X_train, X_test):
  # Calculate the number of entries in the training and test sets
  train_count = len(X_train)
  test_count = len(X_test)

  # Calculate the percentages for the training and test sets
  train_percent = train_count / (train_count + test_count) * 100
  test_percent = test_count / (train_count + test_count) * 100

  # Plotting the pie chart
  fig, ax = plt.subplots()
  wedges, _, labels = ax.pie([train_count, test_count], labels=['Training Set', 'Tes

  # Title for the chart
  ax.set_title('Proportion of Training and Test Sets')

  # Add the actual count below each percentage
  for label, count in zip(labels, [train_count, test_count]):
      ax.text(label.get_position()[0], label.get_position()[1]-0.2, f'{count}', ha='

  plt.show()
```

```
# 1.1
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
#Split the data into features and labels.
X = data.drop(columns=['Type'])
y = data['Type']
#Stratifed split X and y into training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Plot the pie chart
plotTestTrain(X_train, X_test)
```



Answer:

- **1.1**
  - 168 entries are assigned to training.
  - 42 entries are assigned to testing.

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)

```
from sklearn.preprocessing import StandardScaler
```

Answer:

- **1.2** Standard scaler maintains shape and distribution of data since scales mean to 0 & variance to 1. Looking at the first and last few data points, the data is revelatently similar per charateristic. With a lack of outliers, standard scaler will perform well enough for requirements of RandomForestClassifier.

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
from sklearn.ensemble import RandomForestClassifier
```

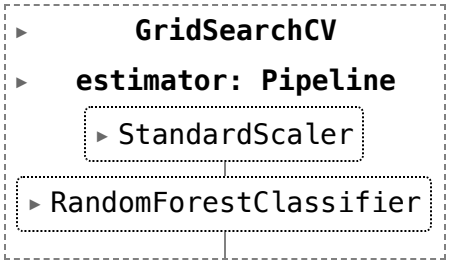
Answer:

- **1.3** Random forest is good at classifying multiple features, it is also robust to overfitting. It will match well with the dataset due to the high feature count.

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.

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV


# 1.4
# Construct a pipeline that incorporates the selected scaler to standardize the data
RFC_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier())
])

# Provide a broad and varied range of hyperparameter values to ensure a thorough sea
RFC_param_grid = {
    'classifier__n_estimators': [10, 50, 75, 100],
    'classifier__max_depth': [25, 50, 75],
    'classifier__min_samples_split': [2, 5, 10]
}

# Execute a grid search within this pipeline to identify the best hyperparameter set
grid_search = GridSearchCV(RFC_pipeline, param_grid=RFC_param_grid, cv=5)
grid_search.fit(X_train, y_train)
```




```
# Display the best settings, and validate the model performance.
print('Best hyperparameters:', grid_search.best_params_)
print('Test set accuracy:', grid_search.score(X_test, y_test))
```

```
Best hyperparameters: {'classifier__max_depth': 25, 'classifier__min_samples_spl
Test set accuracy: 0.9285714285714286
```

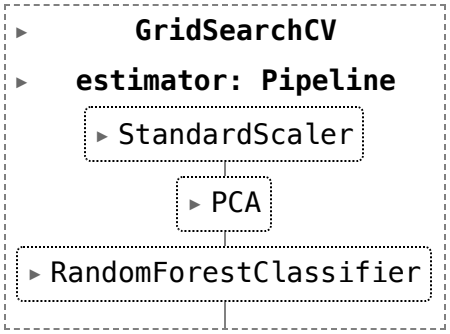
Answer:

- **1.4**
  - The Random Forest Classifier achieved an accuracy of 83.33% on the test set with the best hyperparameters: max_depth=75, min_samples_split=5, and n_estimators=10.

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
from sklearn.decomposition import PCA

# Apply PCA to the Reduce the dimensionality of the data.
pca = PCA(n_components=2)
pca.fit(X_train)

# Pipeline again, but with PCA as well.
RFC_pipeline_pca = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', pca),
    ('classifier', RandomForestClassifier())
])

# Perform a grid search to find best hyperparameters.
grid_search_pca = GridSearchCV(RFC_pipeline_pca, param_grid=RFC_param_grid, cv=5)
grid_search_pca.fit(X_train, y_train)
```




```
# Display the best settings, and validate the model performance.
print('Best hyperparameters with PCA:', grid_search_pca.best_params_)
print('Test set accuracy with PCA:', grid_search_pca.score(X_test, y_test))
```

```
Best hyperparameters with PCA: {'classifier__max_depth': 50, 'classifier__min_sa
Test set accuracy with PCA: 0.9761904761904762
```

```
# Compare the model's performance, display in 3 decimal places.
print(f'Test set accuracy, Original: {grid_search.score(X_test, y_test):.03f}')
print(f'Test set accuracy with PCA: {grid_search_pca.score(X_test, y_test):.03f}')
```

```
Test set accuracy, Original: 0.929
Test set accuracy with PCA: 0.976
```

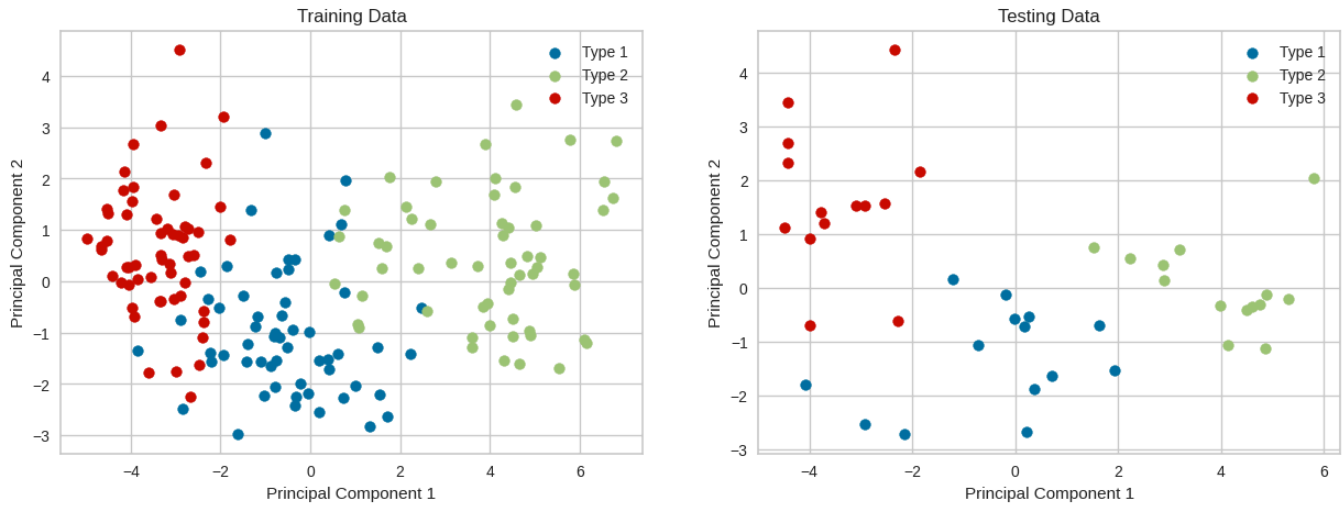
Answer:

- **1.5** PCA is good at maximizing variance in data to cluster the data into groups that are human understandable. Improvement from the original to pca.

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
# Create a 2D Scatter Plot for Training and Testing Sets
# Generate 1-row-two-column subplots for scatter plots for the two-dimensional train
fig, axs = plt.subplots(1, 2, figsize=(15, 5))

for i, (X_data, y_data, title) in enumerate([(X_train, y_train, 'Training'), (X_test
    # Transform the data using PCA
    X_data_pca = pca.transform(X_data)

    # Plot the data
    for label in y.unique():
        axs[i].scatter(X_data_pca[y_data == label, 0], X_data_pca[y_data == label, 1

    axs[i].set_title(f'{title} Data')
    axs[i].set_xlabel('Principal Component 1')
    axs[i].set_ylabel('Principal Component 2')
    axs[i].legend()

plt.show()
```



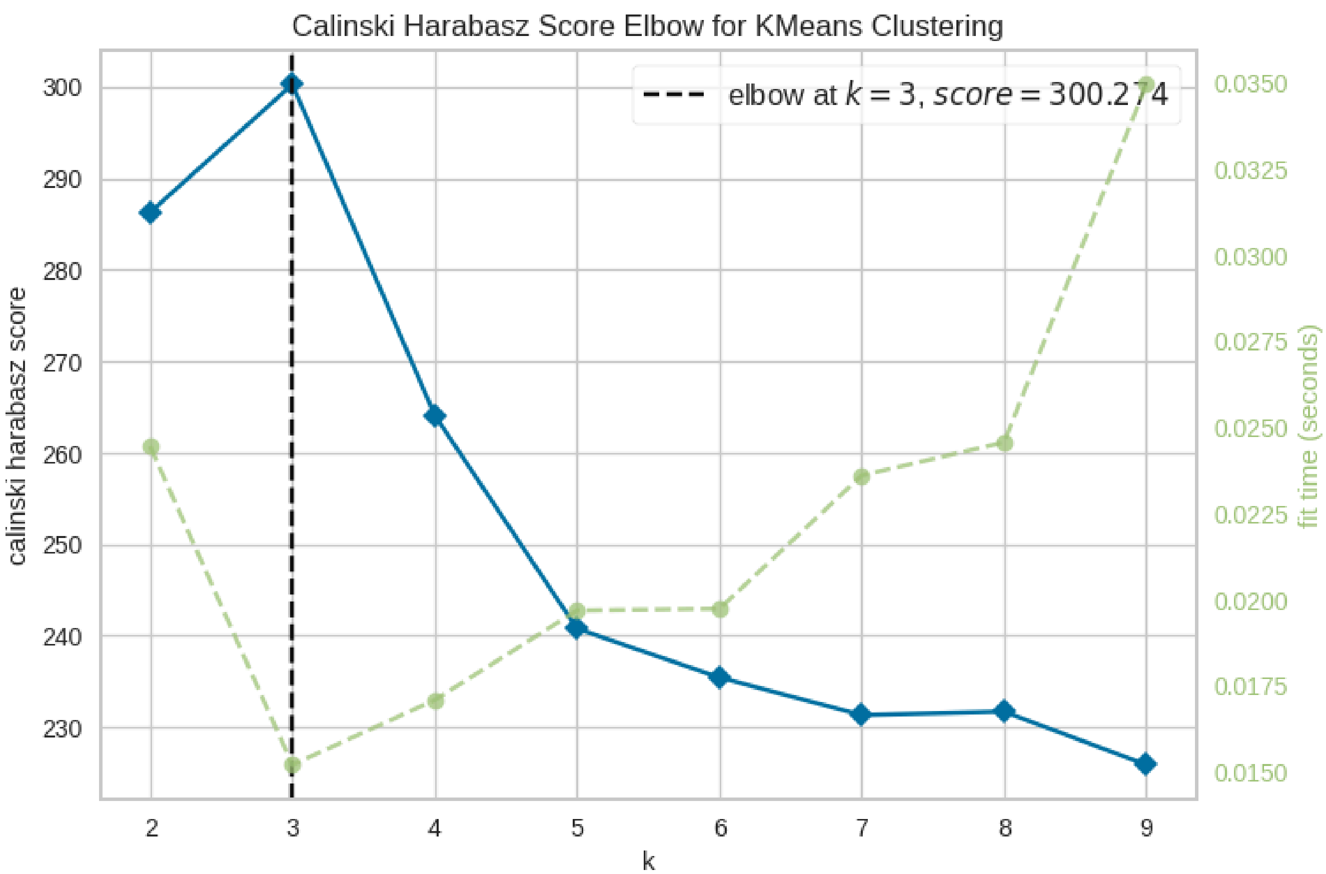
2. Clustering and Visualization of the Seeds Dataset

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`.

```
# import k-means and kelbow_visualizer
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer


# 2.1
# Construct a pipeline that includes standard scaler and the K-Means clustering algo
kmeans_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('kmeans', KMeans())
])

# Use the `KelbowVisualizer` with `metric='calinski_harabasz'` from Yellowbrick to d
visualizer = KElbowVisualizer(kmeans_pipeline.named_steps['kmeans'], k=(2, 10), metr
visualizer.fit(X_train)
visualizer.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
```




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

Answer:

- **2.1** Maximun curvature of the calinski harabasz is reached at k = 3. Hence, 3 is the optimal number of clusters for this dataset.

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
# Label the training data using the pipeline that includes both the scaler and K-Mea
kmeans_pipeline.set_params(kmeans__n_clusters=visualizer.elbow_value_)
kmeans_pipeline.fit(X_train)
y_train_kmeans = kmeans_pipeline.predict(X_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
```

2.3 Dimensionality Reduction Using PCA (2 Points)

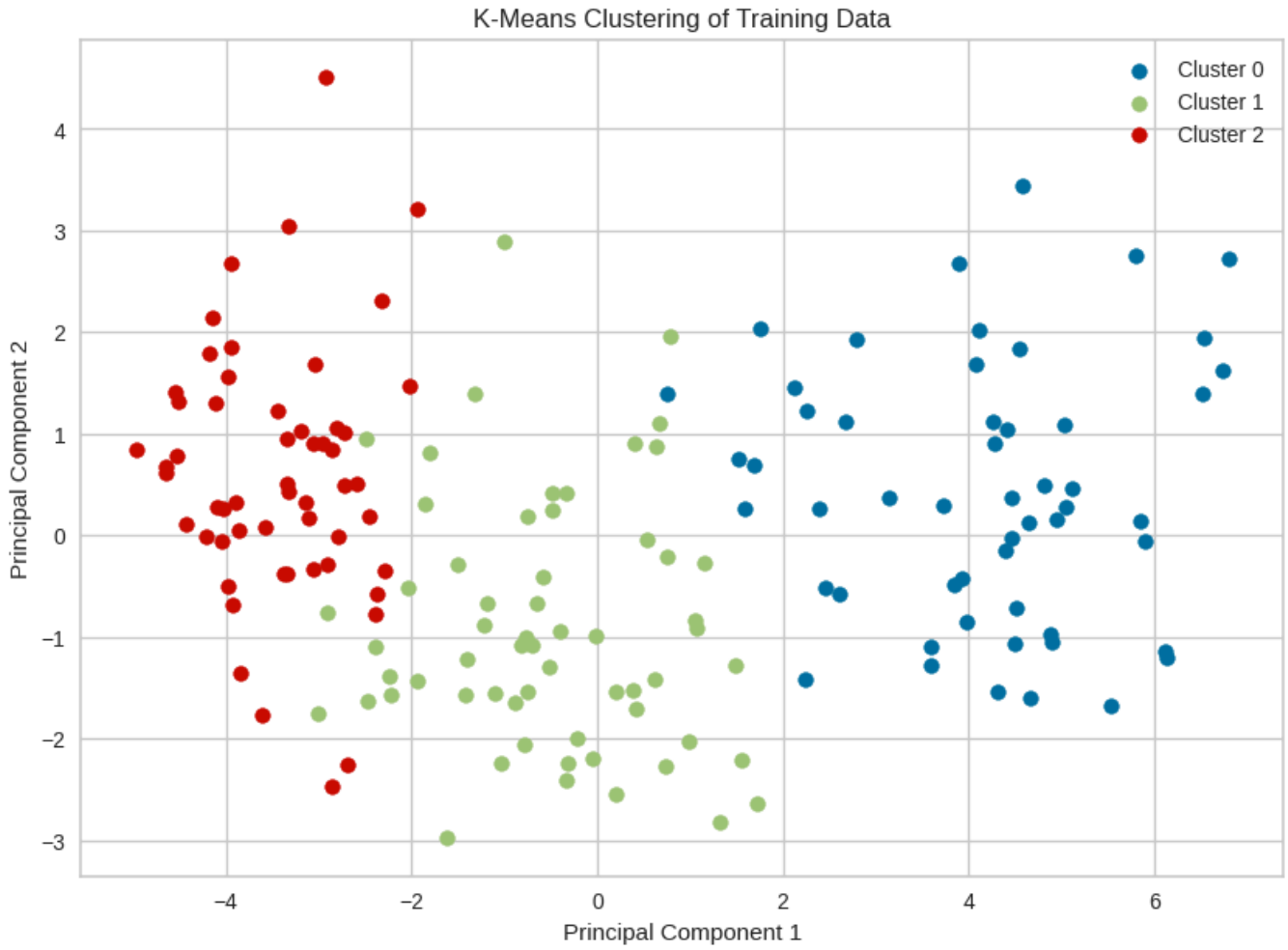
- Apply PCA to reduce the dimensionality of the dataset to 2D.

```
# 2.3
# Apply PCA to reduce the dimensionality of the dataset to 2D.
pca = PCA(n_components=2)
pca.fit(X_train)
X_train_pca = pca.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
# Create a 2D scatter plot of the PCA-reduced data.
plt.figure(figsize=(10, 7))
for label in range(visualizer.elbow_value_):
    plt.scatter(X_train_pca[y_train_kmeans == label, 0], X_train_pca[y_train_kmeans

plt.title('K-Means Clustering of Training Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
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**

- There is distinct separation of clusters based on principal component 1, whereas principal component 2 does not define the data's label. This shows that PCA has done a good job at maximizing variance; There is likely a distinct difference between the variance ratio of principal components 1 & 2.
-

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Double-click (or enter) to edit

Double-click (or enter) to edit