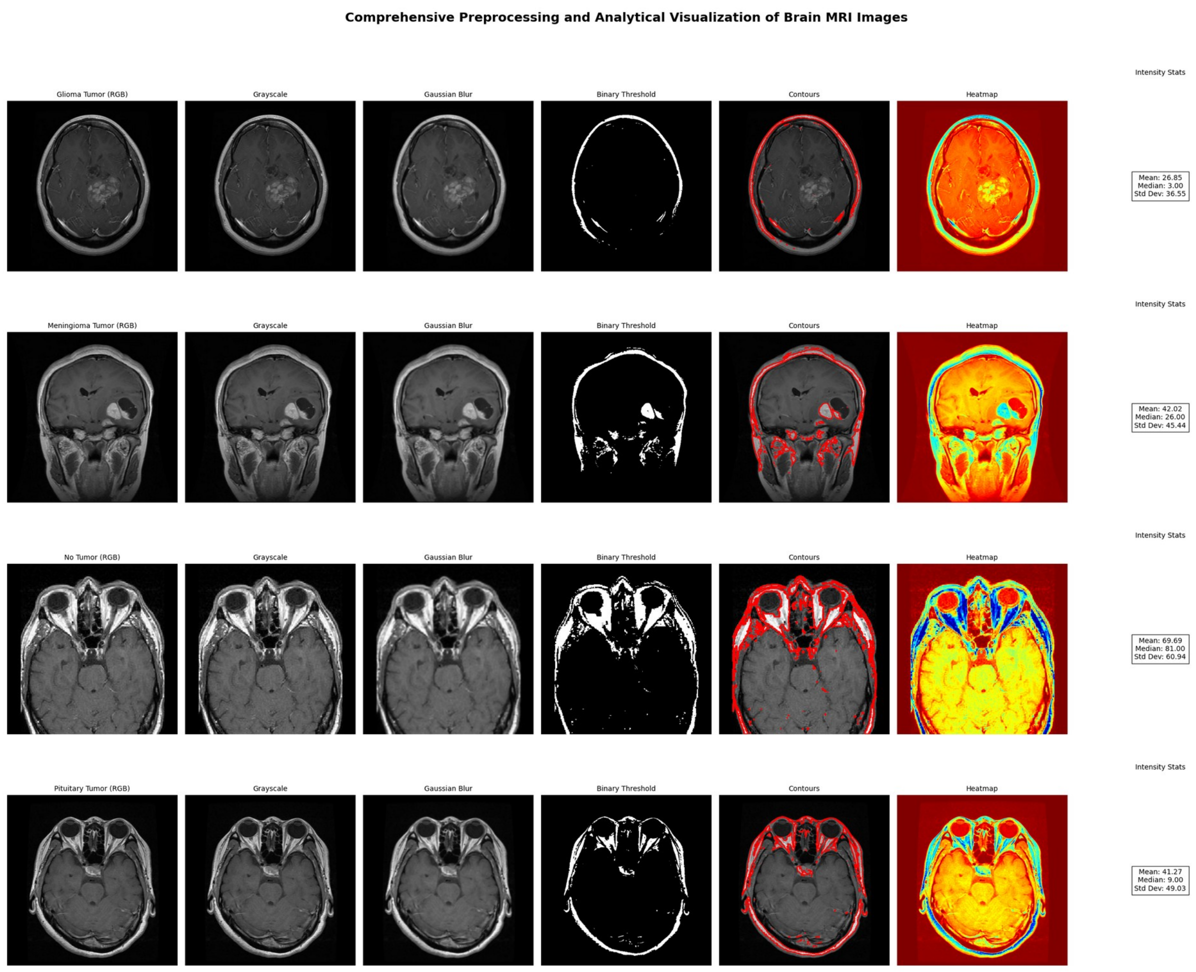
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# List of image paths and their corresponding labels
images = [
    (r'DATASETS\TRAIL_DATA\TRAINING\glioma_tumor\gg (1).jpg', 'Glioma
Tumor'),
    (r'DATASETS\TRAIL_DATA\TRAINING\meningioma_tumor\m (15).jpg',
'Meningioma Tumor'),
    (r'DATASETS\TRAIL_DATA\TRAINING\no_tumor\no_tumor.jpg', 'No
Tumor'),
    (r'DATASETS\TRAIL_DATA\TRAINING\pituitary_tumor\p (1).jpg',
'Pituitary Tumor')
]

# Preprocess and visualize the images
fig, axes = plt.subplots(len(images), 7, figsize=(25, 20))
fig.suptitle("Comprehensive Preprocessing and Analytical Visualization
of Brain MRI Images", fontsize=18, fontweight='bold')

for i, (image_path, label) in enumerate(images):
    # Read the image
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to
RGB
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert to
Grayscale

    # Gaussian Blur
    img_blur = cv2.GaussianBlur(img_gray, (5, 5), 0)

    # Binary Thresholding
    _, img_thresh = cv2.threshold(img_gray, 127, 255,
cv2.THRESH_BINARY)

    # Contour Detection
    contours, _ = cv2.findContours(img_thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    img_contours = cv2.drawContours(img_rgb.copy(), contours, -1,
(255, 0, 0), 2)

    # Pixel Intensity Statistics
    mean_intensity = np.mean(img_gray)
    median_intensity = np.median(img_gray)
    std_intensity = np.std(img_gray)

    # Heatmap
    img_heatmap = cv2.applyColorMap(img_gray, cv2.COLORMAP_JET)
    # Plot Original RGB Image
    axes[i, 0].imshow(img_rgb)
    axes[i, 0].set_title(f"{label} (RGB)", fontsize=10)
    axes[i, 0].axis('off')

    # Plot Grayscale Image
    axes[i, 1].imshow(img_gray, cmap='gray')
    axes[i, 1].set_title("Grayscale", fontsize=10)
    axes[i, 1].axis('off')

    # Plot Gaussian Blurred Image
    axes[i, 2].imshow(img_blur, cmap='gray')
    axes[i, 2].set_title("Gaussian Blur", fontsize=10)
    axes[i, 2].axis('off')

    # Plot Thresholded Image
    axes[i, 3].imshow(img_thresh, cmap='gray')
    axes[i, 3].set_title("Binary Threshold", fontsize=10)
    axes[i, 3].axis('off')

    # Plot Contour Image
    axes[i, 4].imshow(img_contours)
    axes[i, 4].set_title("Contours", fontsize=10)
    axes[i, 4].axis('off')

    # Plot Heatmap
    axes[i, 5].imshow(img_heatmap)
    axes[i, 5].set_title("Heatmap", fontsize=10)
    axes[i, 5].axis('off')

    # Display Pixel Intensity Statistics
    axes[i, 6].text(0.5, 0.5,
                    f"Mean: {mean_intensity:.2f}\n"
                    f"Median: {median_intensity:.2f}\n"
                    f"Std Dev: {std_intensity:.2f}",
                    fontsize=10, ha='center', va='center',
bbox=dict(facecolor='white', alpha=0.8))
    axes[i, 6].set_title("Intensity Stats", fontsize=10)
    axes[i, 6].axis('off')

plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout for title
plt.show()
```



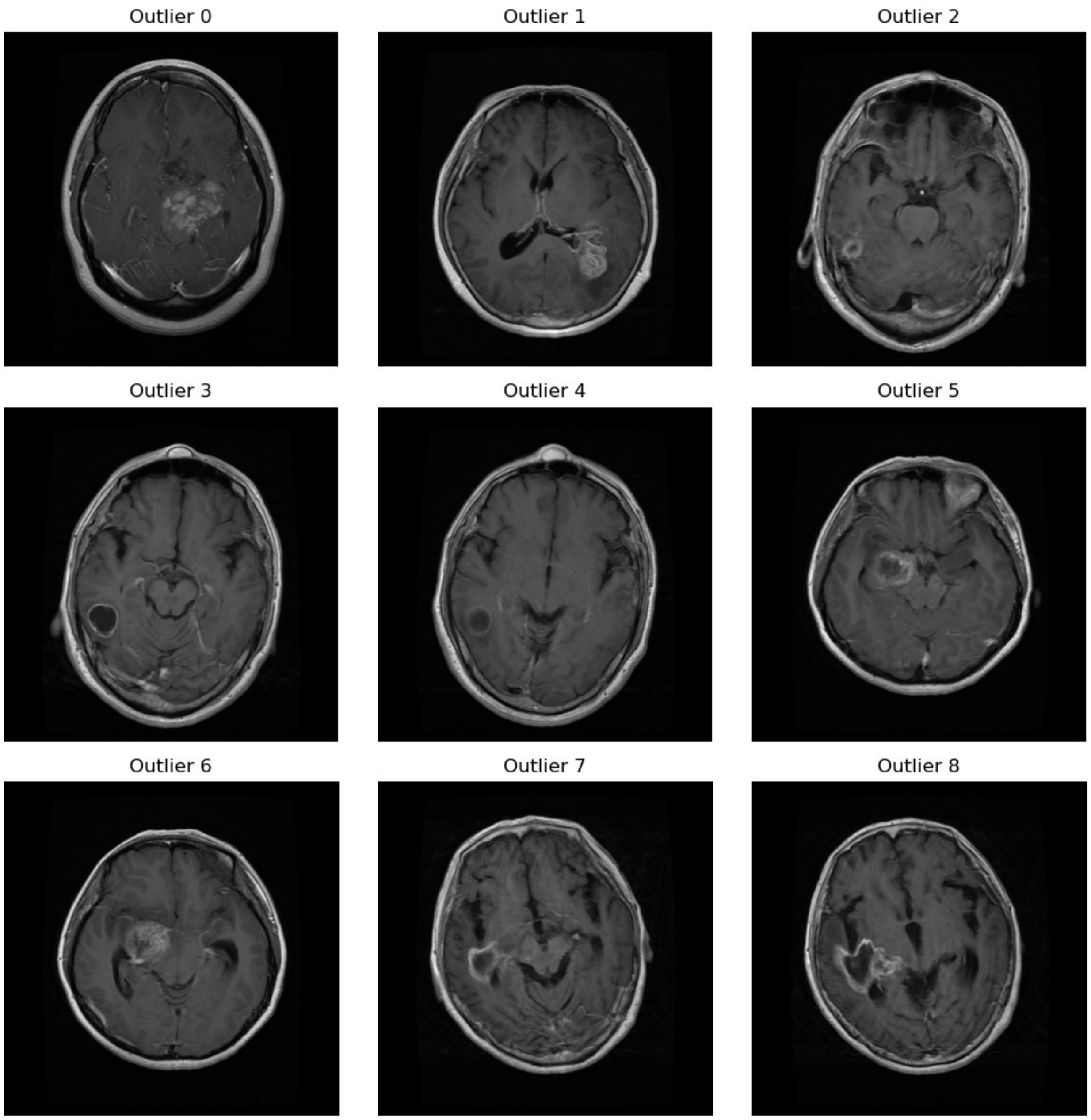
```python
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score
from scipy.cluster.hierarchy import dendrogram, linkage

# Define the paths
base_dir = r'DATASETS\TRAIL_DATA\TRAINING'
categories = ['glioma_tumor', 'meningioma_tumor', 'no_tumor',
'pituitary_tumor']

# Load images
image_data = []
labels = []

for label, category in enumerate(categories):
    folder_path = os.path.join(base_dir, category)
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        # Read and resize the image
        img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)  # Convert
to grayscale
        img_resized = cv2.resize(img, (64, 64))  # Resize to a
standard size
        image_data.append(img_resized.flatten())  # Flatten the image
to a vector
        labels.append(label)

# Convert to numpy arrays
image_data = np.array(image_data)
labels = np.array(labels)

print("Loaded image data shape:", image_data.shape)
```

```
Loaded image data shape: (84, 4096)
```

```
def filter_outliers(image_data, min_contour_area=500,
max_contour_area=10000):
    """
    Filters images based on contour properties (area).

    Parameters:
    - image_data: List of image arrays.
    - min_contour_area: Minimum area for a valid contour.
    - max_contour_area: Maximum area for a valid contour.

    Returns:
    - filtered_images: List of valid images (without outliers).
    - outlier_indices: Indices of images identified as outliers.
    """
    filtered_images = []
    outlier_indices = []

    for idx, img in enumerate(image_data):
        # Convert to grayscale (if not already) and threshold
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) if len(img.shape)
== 3 else img
        _, thresh = cv2.threshold(gray, 50, 255, cv2.THRESH_BINARY)

        # Find contours
        contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        # Calculate contour properties (area)
        areas = [cv2.contourArea(cnt) for cnt in contours]
        if areas and (min_contour_area <= max(areas) <=
max_contour_area):
            filtered_images.append(img)
        else:
            outlier_indices.append(idx)

    return filtered_images, outlier_indices

# Apply the filter
image_data_ = [cv2.imread(os.path.join(base_dir, cat, fname)) for cat
in categories for fname in os.listdir(os.path.join(base_dir, cat))]
filtered_images, outlier_indices = filter_outliers(image_data_)

print(f"Total images: {len(image_data_)}")
print(f"Filtered images (after removing outliers):
{len(filtered_images)}")
print(f"Outlier indices: {outlier_indices}")
```

```
Total images: 84
Filtered images (after removing outliers): 0
Outlier indices: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83]
```

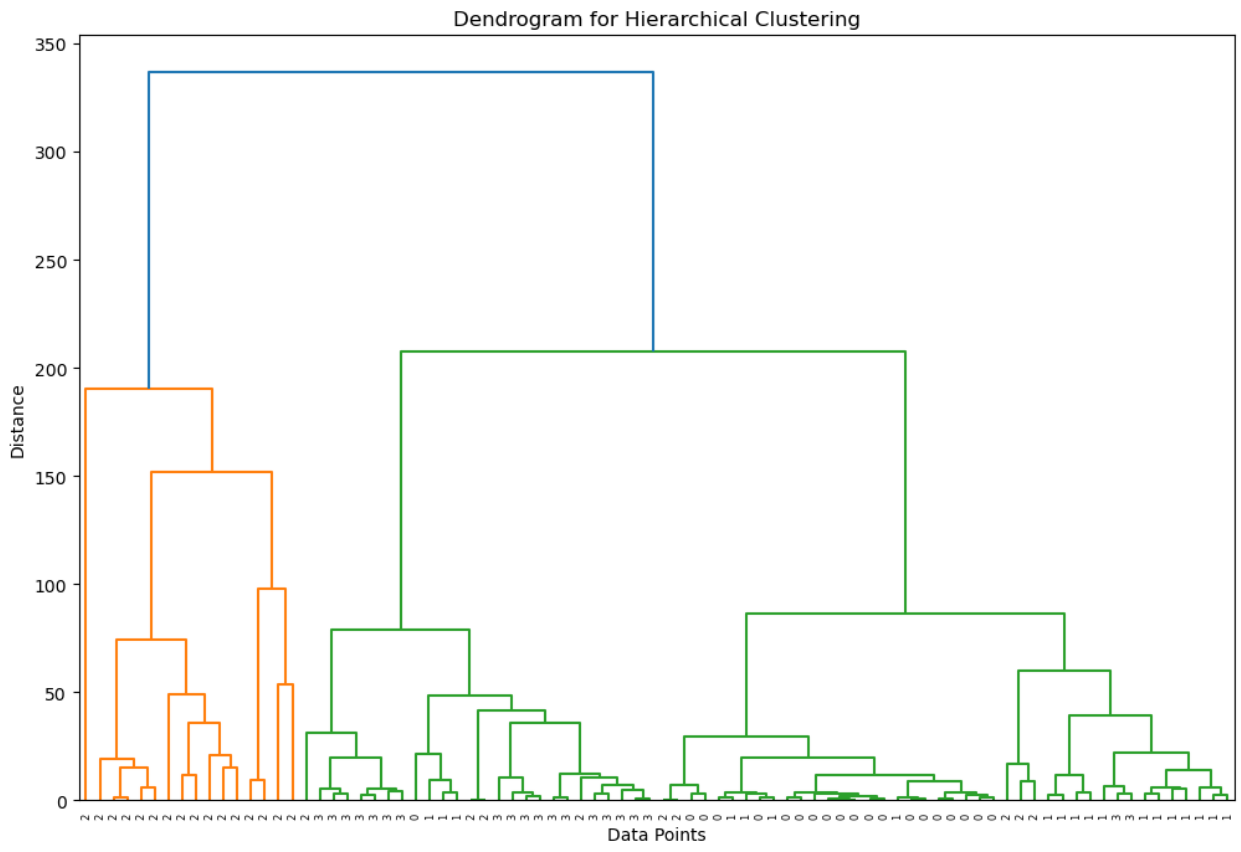
```
# Visualize outliers for reference
plt.figure(figsize=(10, 10))
for i, idx in enumerate(outlier_indices[:9]):  # Show first 9 outliers
    plt.subplot(3, 3, i + 1)
    plt.imshow(image_data_[idx], cmap='gray')
    plt.title(f"Outlier {idx}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```



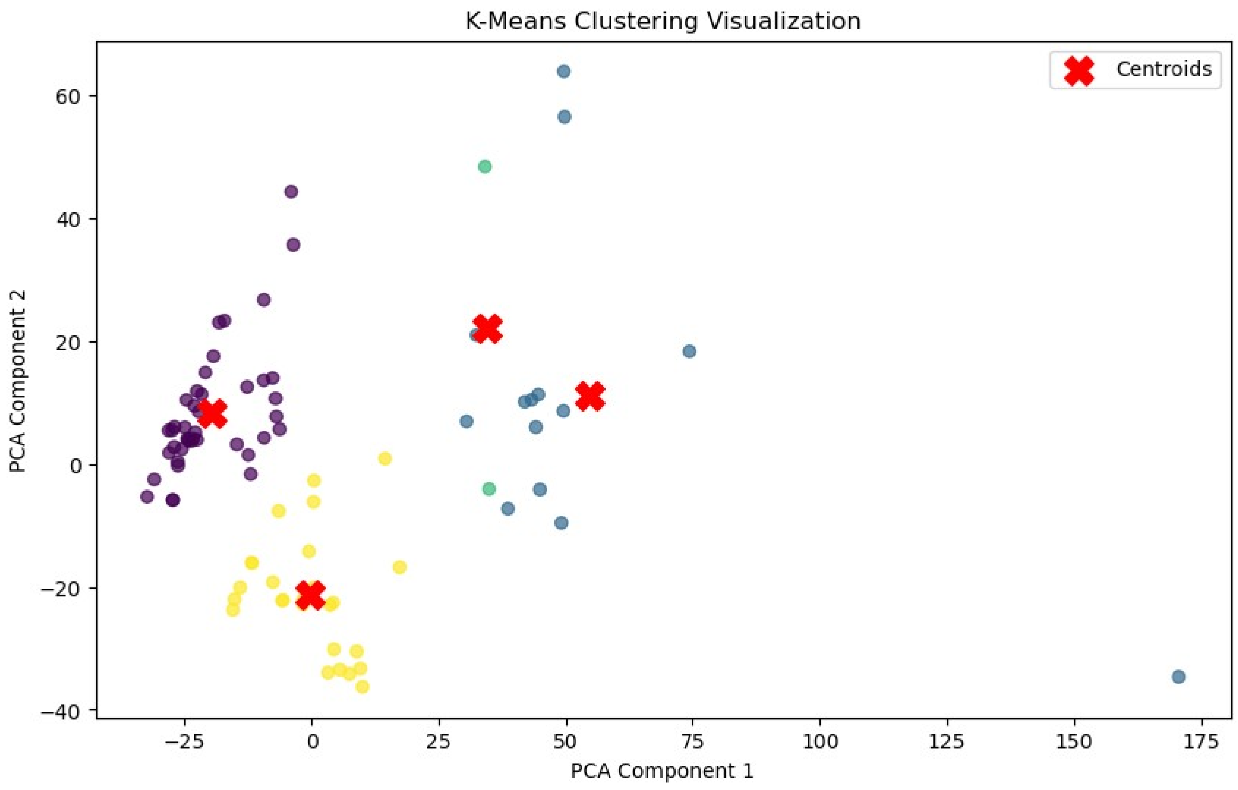
```python
# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(image_data)

# Optional: Dimensionality Reduction using PCA
pca = PCA(n_components=3)
data_pca = pca.fit_transform(data_scaled)

print("Data shape after PCA:", data_pca.shape)
```

```
Data shape after PCA: (84, 3)
```

```python
# Perform Hierarchical Clustering
linkage_matrix = linkage(data_pca, method='ward')
plt.figure(figsize=(12, 8))
dendrogram(linkage_matrix, truncate_mode='level', p=10, labels=labels)
plt.title("Dendrogram for Hierarchical Clustering")
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.show()
```



```python
# Perform K-Means Clustering
kmeans = KMeans(n_clusters=4, random_state=42)
clusters_kmeans = kmeans.fit_predict(data_pca)

# Visualize Clusters in 2D
plt.figure(figsize=(10, 6))
plt.scatter(data_pca[:, 0], data_pca[:, 1], c=clusters_kmeans,
cmap='viridis', alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1], s=200, c='red', marker='X', label='Centroids')
plt.title("K-Means Clustering Visualization")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend()
plt.show()

c:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1429: 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(
```



```
# Evaluate Clustering
silhouette_kmeans = silhouette_score(data_pca, clusters_kmeans)
davies_kmeans = davies_bouldin_score(data_pca, clusters_kmeans)

print(f"Silhouette Score (K-Means): {silhouette_kmeans:.2f}")
print(f"Davies-Bouldin Index (K-Means): {davies_kmeans:.2f}")

# Hierarchical Clustering Evaluation
hierarchical = AgglomerativeClustering(n_clusters=4)
clusters_hierarchical = hierarchical.fit_predict(data_pca)
silhouette_hierarchical = silhouette_score(data_pca,
clusters_hierarchical)
davies_hierarchical = davies_bouldin_score(data_pca,
clusters_hierarchical)
```

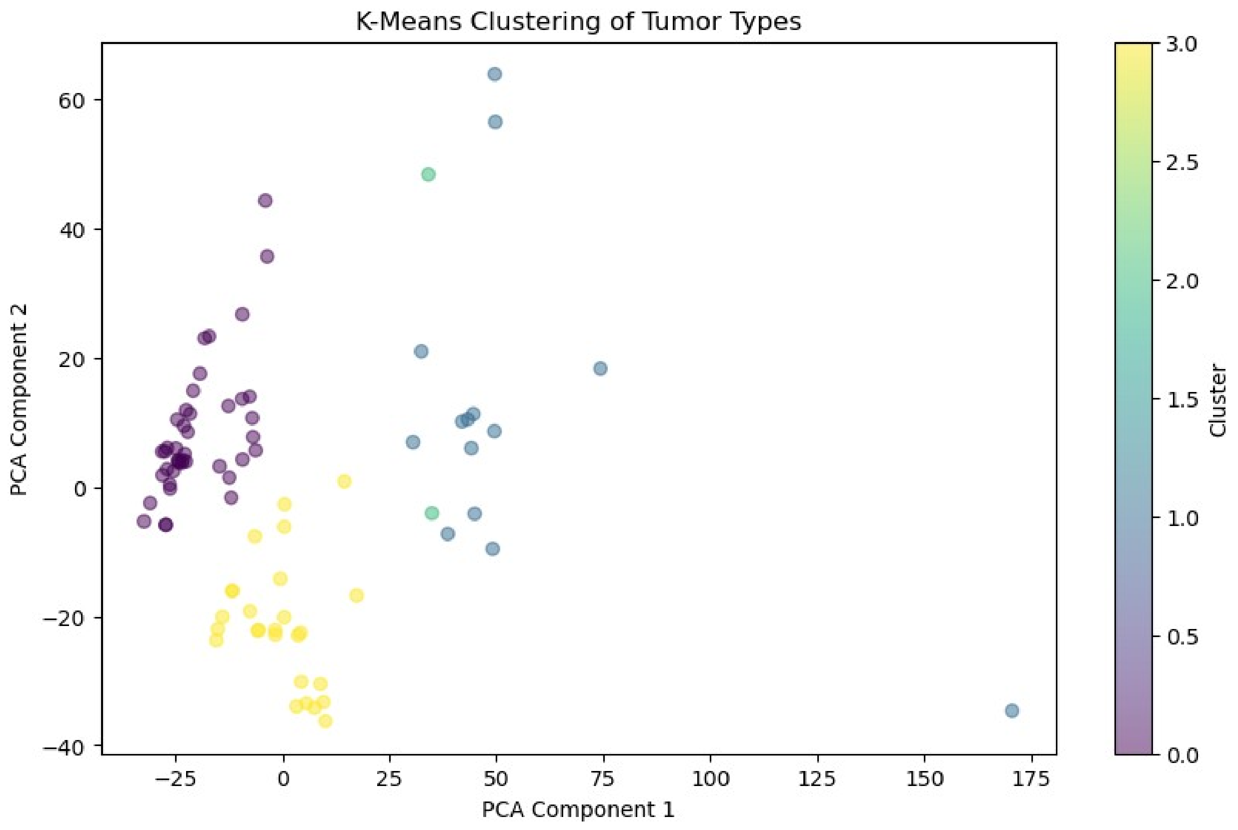
```
print(f"Silhouette Score (Hierarchical):
{silhouette_hierarchical:.2f}")
print(f"Davies-Bouldin Index (Hierarchical):
{davies_hierarchical:.2f}")
```

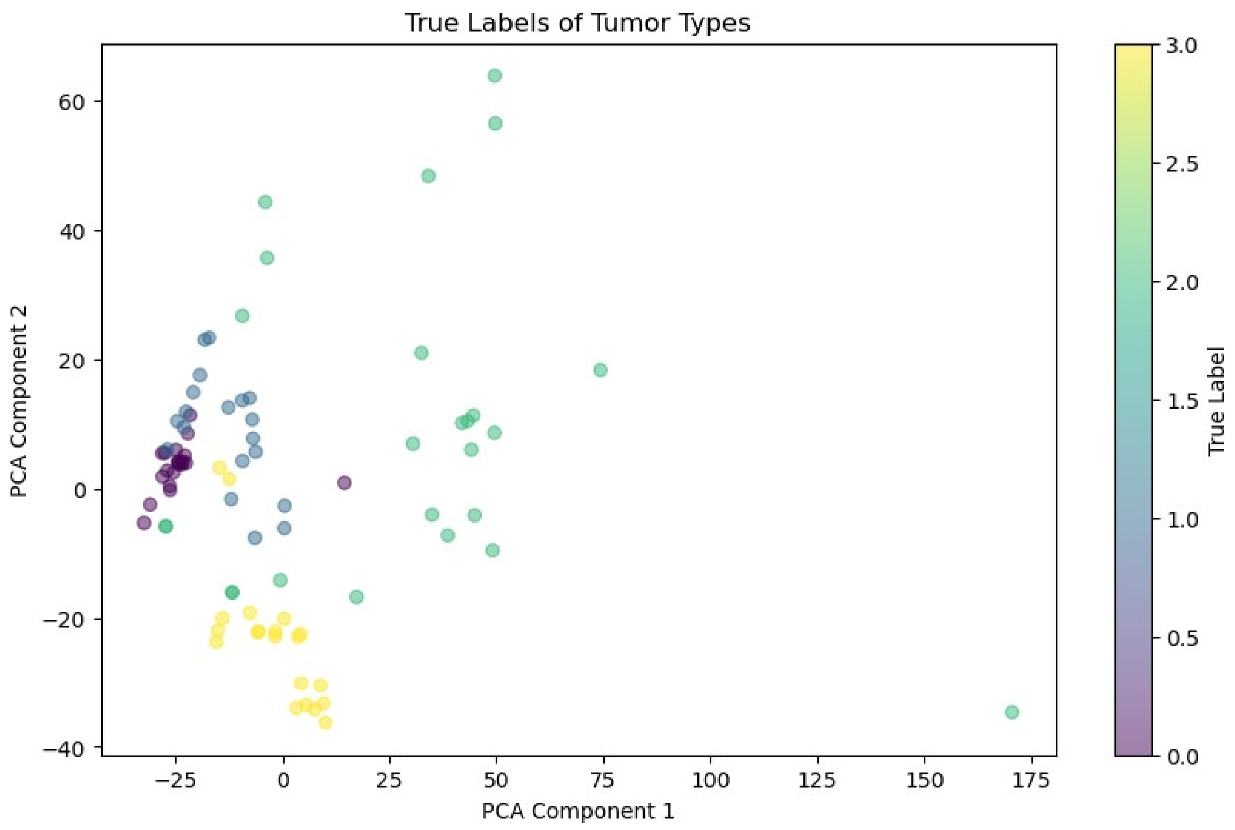
```
Silhouette Score (K-Means): 0.48
Davies-Bouldin Index (K-Means): 0.76
Silhouette Score (Hierarchical): 0.48
Davies-Bouldin Index (Hierarchical): 0.67
```

```
# Assign cluster labels to a DataFrame for comparison
import pandas as pd
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Create a DataFrame with true labels and predicted clusters
results_df = pd.DataFrame({
    'True_Label': labels,  # True tumor type labels
    'KMeans_Cluster': clusters_kmeans,
    'Hierarchical_Cluster': clusters_hierarchical
})

# Map label numbers to their respective tumor names
label_mapping = {0: "glioma_tumor", 1: "meningioma_tumor", 2:
"no_tumor", 3: "pituitary_tumor"}
results_df['Label_Name'] = results_df['True_Label'].map(label_mapping)

print(results_df.tail(20))  # View sample results
```

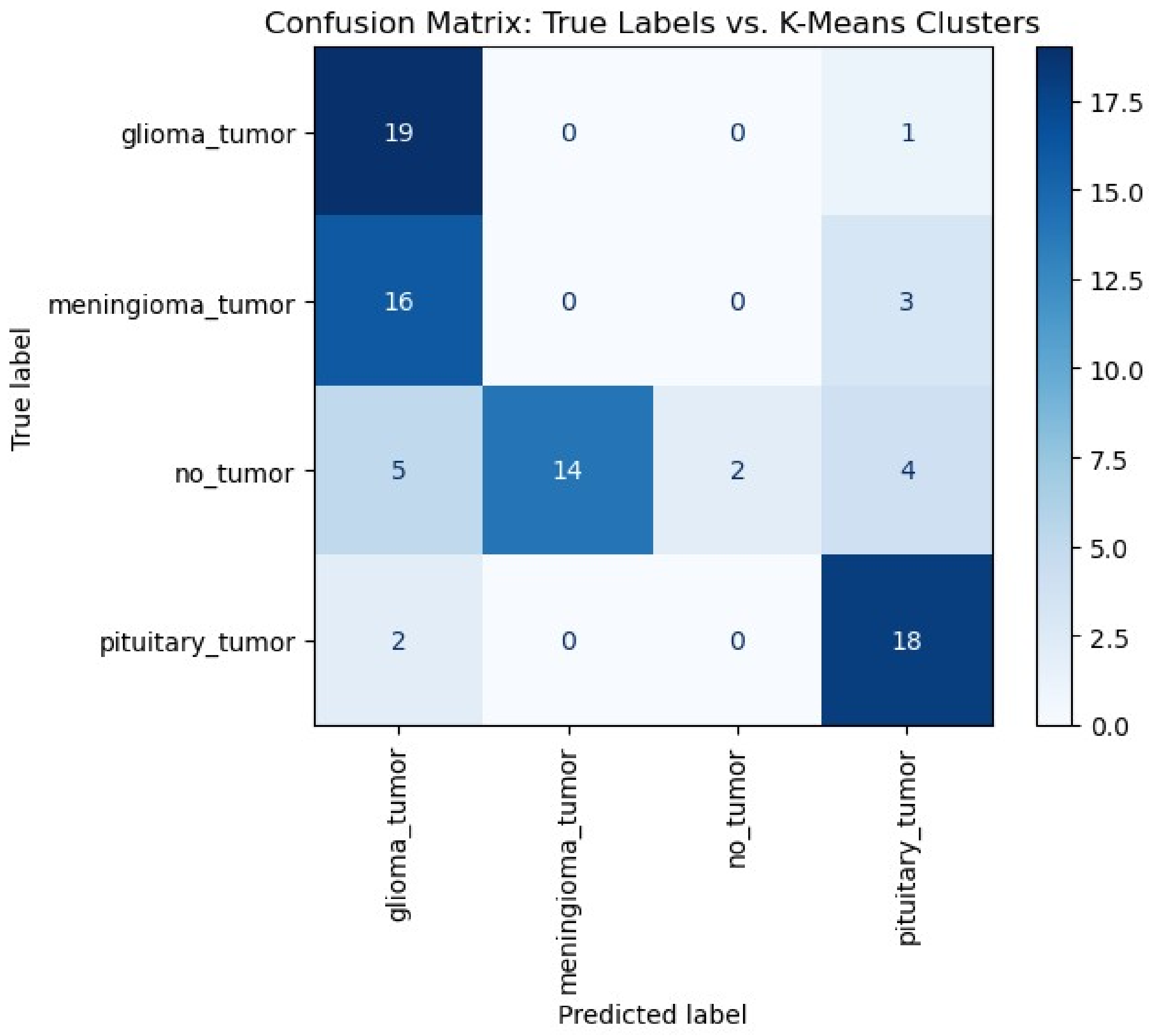
```
    True_Label  KMeans_Cluster  Hierarchical_Cluster       Label_Name
64           3               0                     1  pituitary_tumor
65           3               3                     2  pituitary_tumor
66           3               3                     2  pituitary_tumor
67           3               3                     2  pituitary_tumor
68           3               3                     2  pituitary_tumor
69           3               3                     2  pituitary_tumor
70           3               3                     2  pituitary_tumor
71           3               3                     2  pituitary_tumor
72           3               3                     2  pituitary_tumor
73           3               3                     2  pituitary_tumor
74           3               3                     2  pituitary_tumor
75           3               3                     2  pituitary_tumor
76           3               0                     1  pituitary_tumor
77           3               3                     2  pituitary_tumor
78           3               3                     2  pituitary_tumor
79           3               3                     2  pituitary_tumor
80           3               3                     2  pituitary_tumor
81           3               3                     2  pituitary_tumor
82           3               3                     2  pituitary_tumor
83           3               3                     2  pituitary_tumor
```

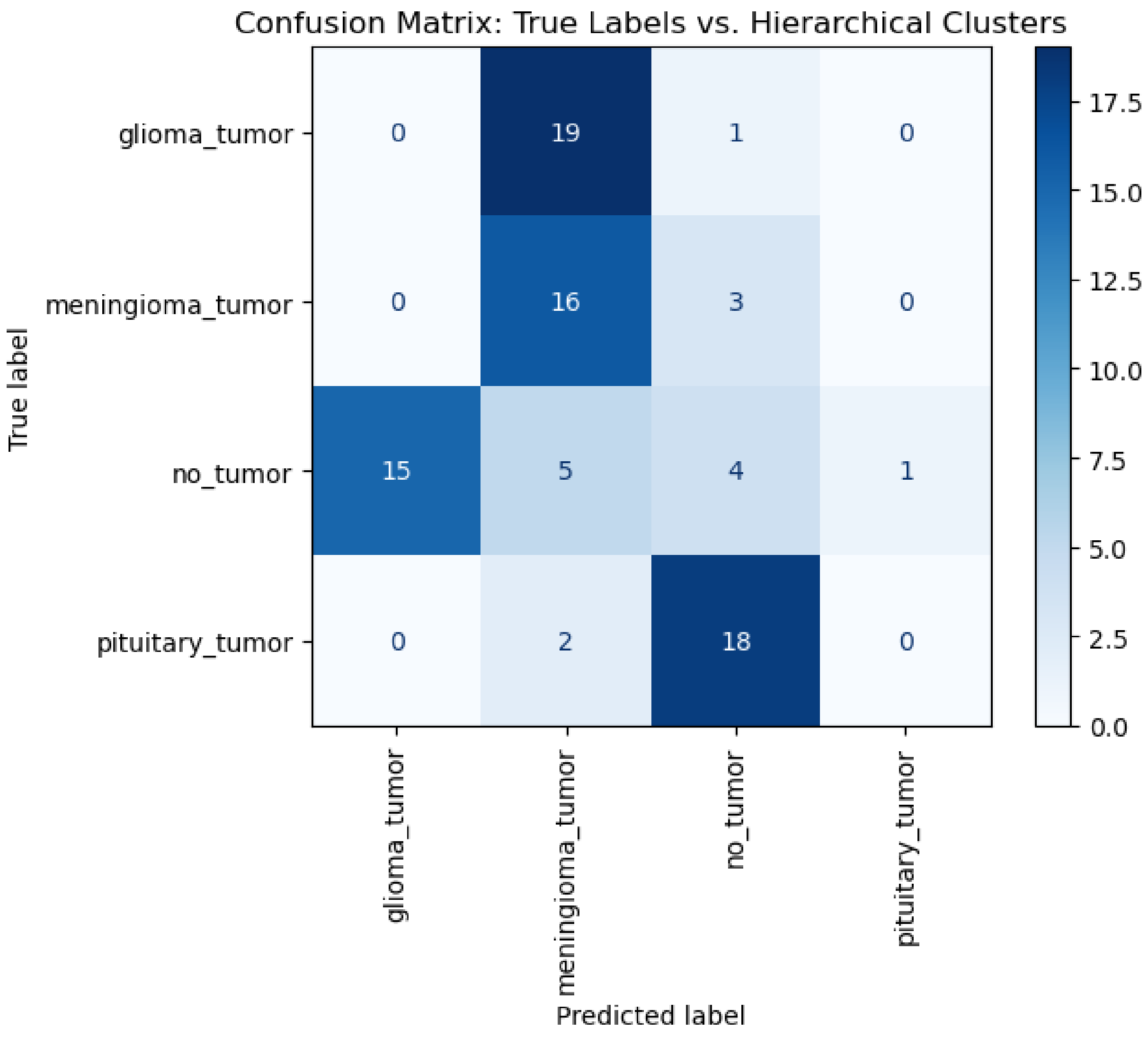
```python
# Visualize K-Means Clustering results
plt.figure(figsize=(10, 6))
plt.scatter(data_pca[:, 0], data_pca[:, 1], c=clusters_kmeans,
cmap='viridis', alpha=0.5, label='K-Means Clusters')
plt.title("K-Means Clustering of Tumor Types")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.colorbar(label="Cluster")
plt.show()

# Visualize True Labels
plt.figure(figsize=(10, 6))
plt.scatter(data_pca[:, 0], data_pca[:, 1], c=labels, cmap='viridis',
alpha=0.5, label='True Labels')
plt.title("True Labels of Tumor Types")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.colorbar(label="True Label")
plt.show()
```





```
# Confusion Matrix for K-Means
cm_kmeans = confusion_matrix(labels, clusters_kmeans)
disp_kmeans = ConfusionMatrixDisplay(confusion_matrix=cm_kmeans,
display_labels=list(label_mapping.values()))
disp_kmeans.plot(cmap='Blues', xticks_rotation='vertical')
plt.title("Confusion Matrix: True Labels vs. K-Means Clusters")
plt.show()

# Confusion Matrix for Hierarchical Clustering
cm_hierarchical = confusion_matrix(labels, clusters_hierarchical)
disp_hierarchical =
ConfusionMatrixDisplay(confusion_matrix=cm_hierarchical,
display_labels=list(label_mapping.values()))
disp_hierarchical.plot(cmap='Blues', xticks_rotation='vertical')
plt.title("Confusion Matrix: True Labels vs. Hierarchical Clusters")
plt.show()
```


Confusion Matrix: True Labels vs. K-Means Clusters
True label
Predicted label
glioma_tumor
meningioma_tumor
no_tumor
pituitary_tumor
19
0
0
1
16
0
0
3
5
14
2
4
2
0
0
18
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0



Confusion Matrix: True Labels vs. Hierarchical Clusters
True label
Predicted label
glioma_tumor
meningioma_tumor
no_tumor
pituitary_tumor
0 19 1 0
0 16 3 0
15 5 4 1
0 2 18 0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5