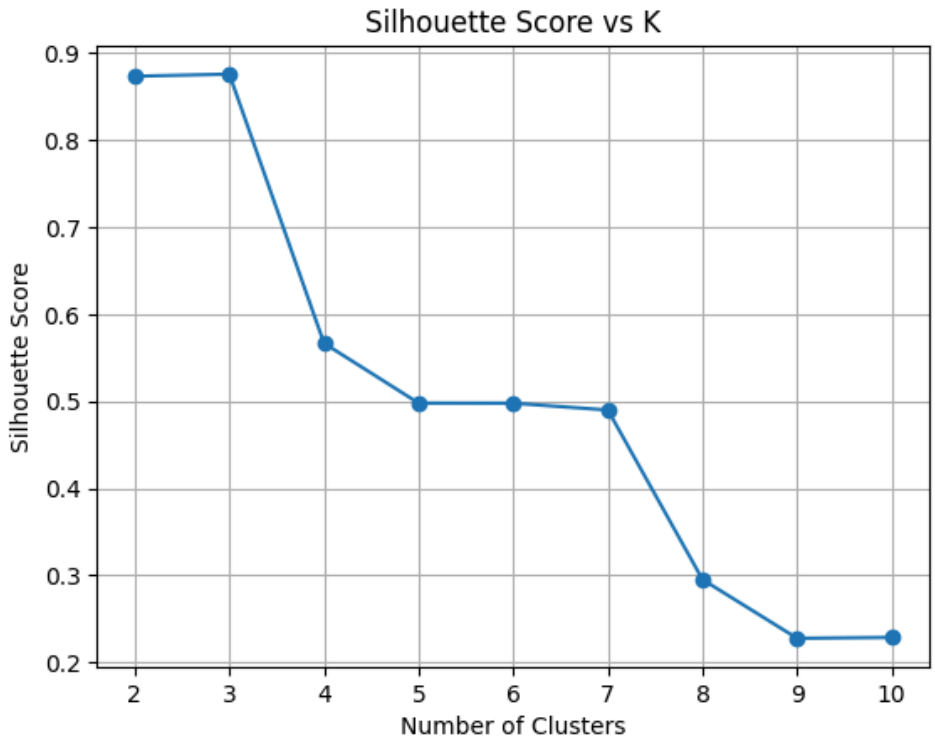
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from matplotlib.patches import Circle

# Step 1: Load and parse 'Serial Data' column
df = pd.read_csv("Serial_data_On_metal_table.csv")
serial_data_numeric = df['Serial Data'].apply(lambda x: list(map(int, x.split(','))))
X = np.array(serial_data_numeric.tolist())

# Step 2: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA (retain 95% variance)
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)
print("Reduced shape after PCA:", X_pca.shape)

# Step 4: Identify important features
importance_scores = np.abs(pca.components_).sum(axis=0)
important_feature_indices = np.argsort(importance_scores)[::-1]
print("Top 5 important feature indices:", important_feature_indices[:5])

# Step 5: Tune K using silhouette scores
silhouette_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_pca)
    score = silhouette_score(X_pca, labels)
    silhouette_scores.append(score)

# Plot silhouette scores
plt.plot(K_range, silhouette_scores, marker='o')
plt.title("Silhouette Score vs K")
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.grid(True)
plt.show()

# Step 6: Final clustering with best K
optimal_k = K_range[np.argmax(silhouette_scores)]
print("Optimal number of clusters:", optimal_k)

kmeans_final = KMeans(n_clusters=optimal_k, random_state=42)
final_labels = kmeans_final.fit_predict(X_pca)
df['Cluster'] = final_labels

# Step 7: Visualize in 2D PCA space
pca_2d = PCA(n_components=2)
X_2d = pca_2d.fit_transform(X_scaled)

# Add PCA results and Grid number to a new DataFrame
viz_df = pd.DataFrame({
    'PC1': X_2d[:, 0],
    'PC2': X_2d[:, 1],
    'Grid': df['Grid No'],
    'Cluster': df['Cluster']
```

```
})

# Plot with each grid in a different color
plt.figure(figsize=(10, 7))
unique_grids = viz_df['Grid'].unique()
colors = plt.cm.get_cmap("tab10", len(unique_grids))

for i, grid_label in enumerate(unique_grids):
    subset = viz_df[viz_df['Grid'] == grid_label]
    plt.scatter(subset['PC1'], subset['PC2'], label=grid_label, s=20, color=colors(i))

# Add cluster centroids and circular boundaries
ax = plt.gca()
for cluster_label in viz_df['Cluster'].unique():
    cluster_data = viz_df[viz_df['Cluster'] == cluster_label][['PC1', 'PC2']].values

    # Centroid
    centroid = cluster_data.mean(axis=0)
    plt.scatter(centroid[0], centroid[1], color='black', marker='x', s=100, linewidths=2)

    # Circle radius: max distance from centroid
    distances = np.linalg.norm(cluster_data - centroid, axis=1)
    radius = distances.max()

    # Add circle
    circle = Circle(centroid, radius, color='gray', fill=False, linestyle='--', linewidth=1.5)
    ax.add_patch(circle)

plt.title('Grid-wise Visualization using PCA with Clusters (Circles + Centroids)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Grid NO')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Reduced shape after PCA: (3208, 15)
Top 5 important feature indices: [249 230 250 240 245]
```




```
Optimal number of clusters: 3
<ipython-input-1-f766e34118be>:70: MatplotlibDeprecationWarning: The get_cmap function was deprecated i
  colors = plt.cm.get_cmap("tab10", len(unique_grids))
```