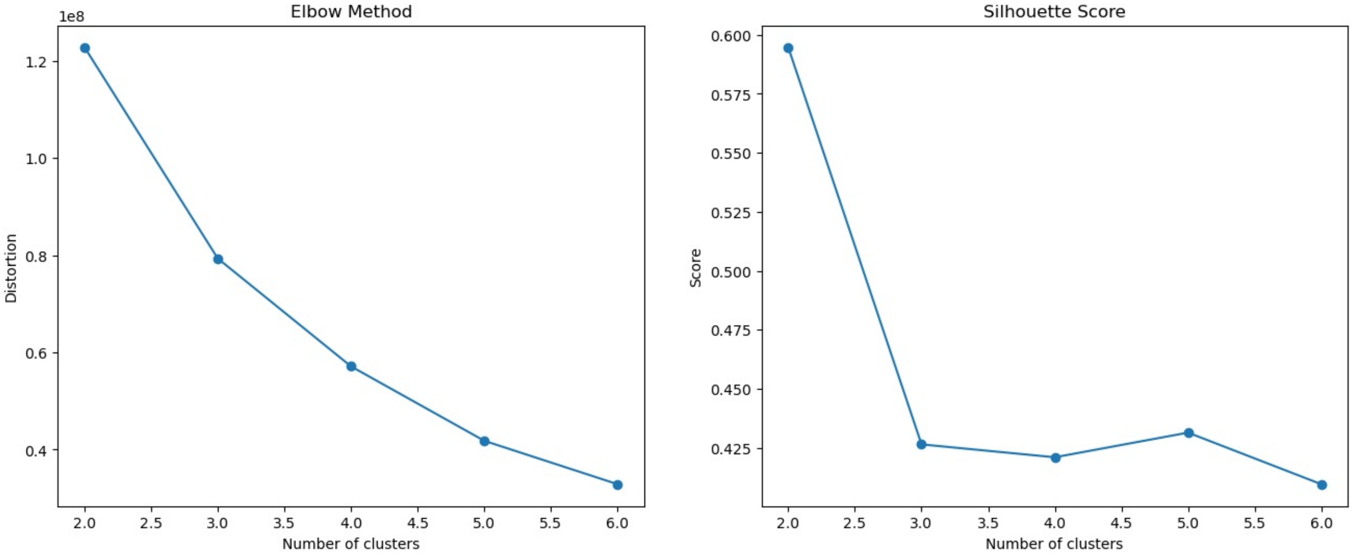
In [53]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, f1_score, recall_score
from sklearn.metrics import silhouette_score
from time import time
from sklearn.preprocessing import StandardScaler
from sklearn.mixture import GaussianMixture
```

1) K-Means

In [8]:
```python
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))


class KMeansOptimized:
    def __init__(self, K=5, max_iters=100, plot_steps=False):
        self.K = K
        self.max_iters = max_iters
        self.plot_steps = plot_steps
        self.clusters = [[] for _ in range(self.K)]
        self.centroids = []

    def predict(self, X):
        self.X = X
        self.n_samples, self.n_features = X.shape

        # Initialize centroids
        self.centroids = self._initialize_centroids()

        for _ in range(self.max_iters):
            # Assigning samples to closest centroids
            self.clusters = self._create_clusters(self.centroids)

            # Calculating new centroids..
            centroids_old = self.centroids
            self.centroids = self._get_centroids(self.clusters)

            if self._is_converged(centroids_old, self.centroids):
                break

        return self._get_cluster_labels(self.clusters)

    def _initialize_centroids(self):
        centroids = []
        centroids.append(self.X[np.random.choice(self.n_samples)])
        for _ in range(1, self.K):
            distances = np.array([min([euclidean_distance(x, c) for c in centroids]) for x in self.X])
            next_centroid = self.X[np.argmax(distances)]
            centroids.append(next_centroid)
        return np.array(centroids)

    def _create_clusters(self, centroids):
        clusters = [[] for _ in range(self.K)]
        for idx, sample in enumerate(self.X):
            centroid_idx = self._closest_centroid(sample, centroids)
            clusters[centroid_idx].append(idx)
        return clusters

    def _closest_centroid(self, sample, centroids):
        distances = [euclidean_distance(sample, point) for point in centroids]
        return np.argmin(distances)

    def _get_centroids(self, clusters):
        return np.array([np.mean(self.X[cluster], axis=0) for cluster in clusters])

    def _is_converged(self, centroids_old, centroids):
        distances = [euclidean_distance(centroids_old[i], centroids[i]) for i in range(self.K)]
        return sum(distances) == 0

    def _get_cluster_labels(self, clusters):
        labels = np.empty(self.n_samples)
        for cluster_idx, cluster in enumerate(clusters):
            for sample_idx in cluster:
                labels[sample_idx] = cluster_idx
        return labels
```

In [62]:
```python
# Evaluate optimal K
def evaluate_optimal_k(X, max_k=10):
```

```
    distortions = []
    silhouette_scores = []
    for k in range(2, max_k + 1):
        kmeans = KMeansOptimized(K=k, max_iters=100)
        labels = kmeans.predict(X)
        distortions.append(sum([euclidean_distance(X[idx], kmeans.centroids[cluster_idx]) ** 2
                                for cluster_idx, cluster in enumerate(kmeans.clusters) for idx in cluster]))
        silhouette_scores.append(silhouette_score(X, labels))


    fig, ax = plt.subplots(1, 2, figsize=(16, 6))
    ax[0].plot(range(2, max_k + 1), distortions, marker='o')
    ax[0].set_title('Elbow Method')
    ax[0].set_xlabel('Number of clusters')
    ax[0].set_ylabel('Distortion')

    ax[1].plot(range(2, max_k + 1), silhouette_scores, marker='o')
    ax[1].set_title('Silhouette Score')
    ax[1].set_xlabel('Number of clusters')
    ax[1].set_ylabel('Score')

    plt.show()


df = pd.read_csv('HTRU_2.csv')
X = df.values

start_time = time()
evaluate_optimal_k(X, max_k=6)
print(f"Runtime for evaluating optimal K: {time() - start_time:.2f} seconds")

    # Example clustering with best K
best_k = 4  # Choose based on evaluation
kmeans = KMeansOptimized(K=best_k, max_iters=8, plot_steps=False)
y_pred = kmeans.predict(X)
```




```
Runtime for evaluating optimal K: 615.68 seconds
```

In [ ]:

```
distortion_custom = sum([
    euclidean_distance(X[idx], kmeans_custom.centroids[cluster_idx]) ** 2
    for cluster_idx, cluster in enumerate(kmeans_custom.clusters)
    for idx in cluster
])

runtime_custom, kmeans_custom.centroids, distortion_custom
```

Sklearn Kmeans

In [60]:

```
from sklearn.cluster import KMeans

# Run scikit-learn's KMeans implementation
start_time = time()
kmeans_sklearn = KMeans(n_clusters=4, max_iter=8, random_state=0, init='k-means++')
kmeans_sklearn.fit(X)
runtime_sklearn = time() - start_time

# Extract results from scikit-learn implementation
centroids_sklearn = kmeans_sklearn.cluster_centers_
distortion_sklearn = kmeans_sklearn.inertia_  # Sum of squared distances to closest centroid

# Output scikit-learn results
```

```
runtime_sklearn, centroids_sklearn, distortion_sklearn
```

```
C:\Users\tegbe\anaconda3\Lib\site-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(
```

Out[60]:
```
(0.988760232925415,
 array([[1.15964967e+02, 4.68764368e+01, 2.26292669e-01, 4.11710880e-01,
         2.65023505e+00, 1.77565360e+01, 9.03515739e+00, 9.86655294e+01],
        [1.14012822e+02, 4.64126667e+01, 2.51602041e-01, 4.94391969e-01,
         1.32035260e+00, 1.22676788e+01, 1.38726130e+01, 2.37439809e+02],
        [1.01128804e+02, 4.60003613e+01, 1.02992707e+00, 4.77714743e+00,
         3.57606883e+01, 4.88868210e+01, 3.32384360e+00, 1.65258454e+01],
        [1.15564444e+02, 4.75141510e+01, 2.16842603e-01, 3.79980279e-01,
         6.46129545e-01, 9.33803610e+00, 2.17424191e+01, 5.50353765e+02]]),
 57189270.795420825)
```

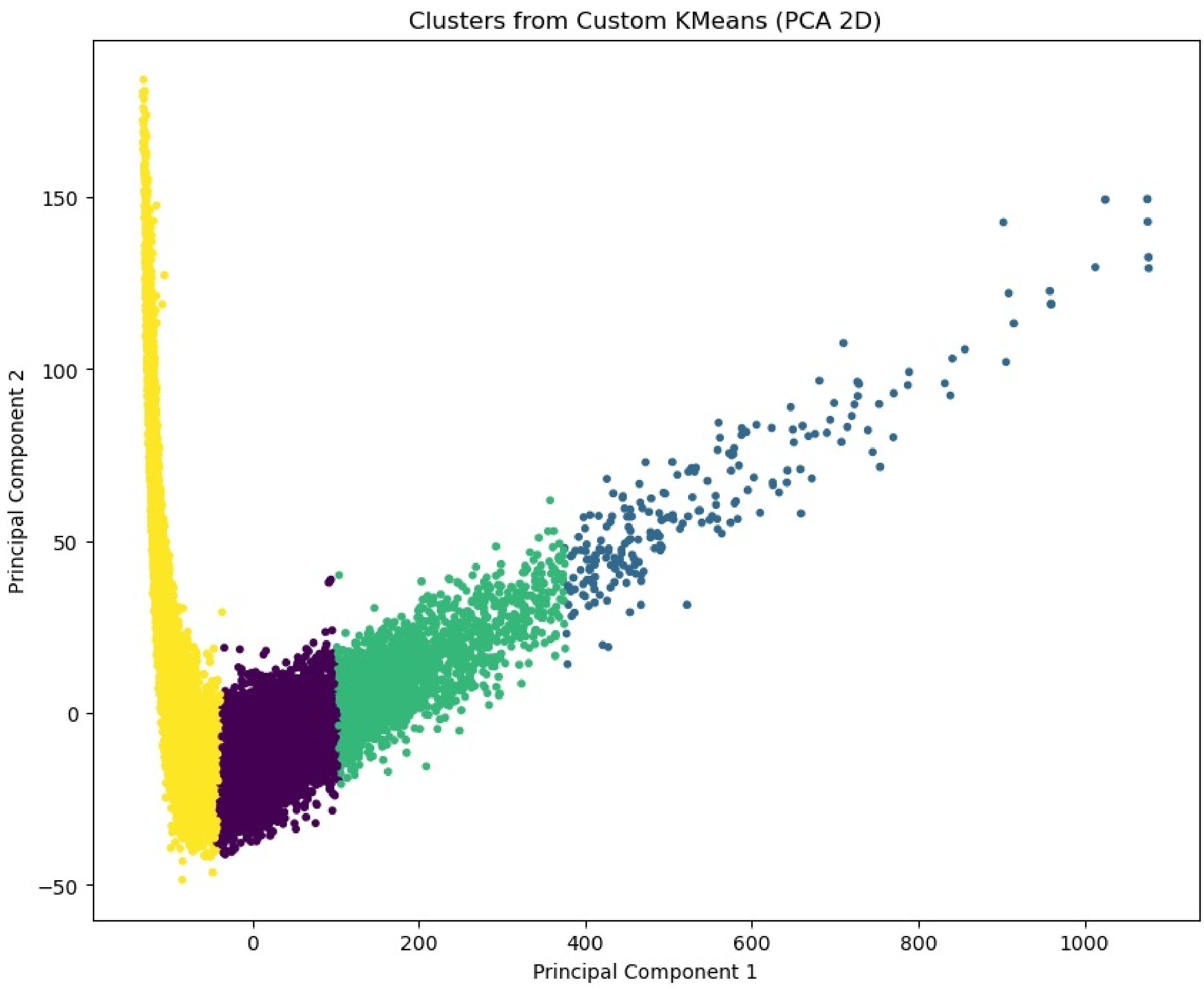
In [ ]:

In [ ]:

Using PCA to represent the data

In [64]:
```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def perform_pca(X, n_components=2):
    # Center the data
    X_mean = np.mean(X, axis=0)
    X_centered = X - X_mean

    U, S, Vt = np.linalg.svd(X_centered, full_matrices=False) # Singular Value Decomposition
    X_pca = np.dot(X_centered, Vt[:n_components].T) # Projecting the data onto principal components
    return X_pca

# dataset (2D)
X_pca_2d = perform_pca(X, n_components=2)
```

In [67]:
```python
def plot_clusters_2d(X_pca, labels, title="2D PCA Cluster Representation"):
    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis', s=10)
    plt.title(title)
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
    plt.show()

plot_clusters_2d(X_pca_2d, y_pred, title="Clusters from Custom KMeans (PCA 2D)") #Using our Kmeans implementati
```



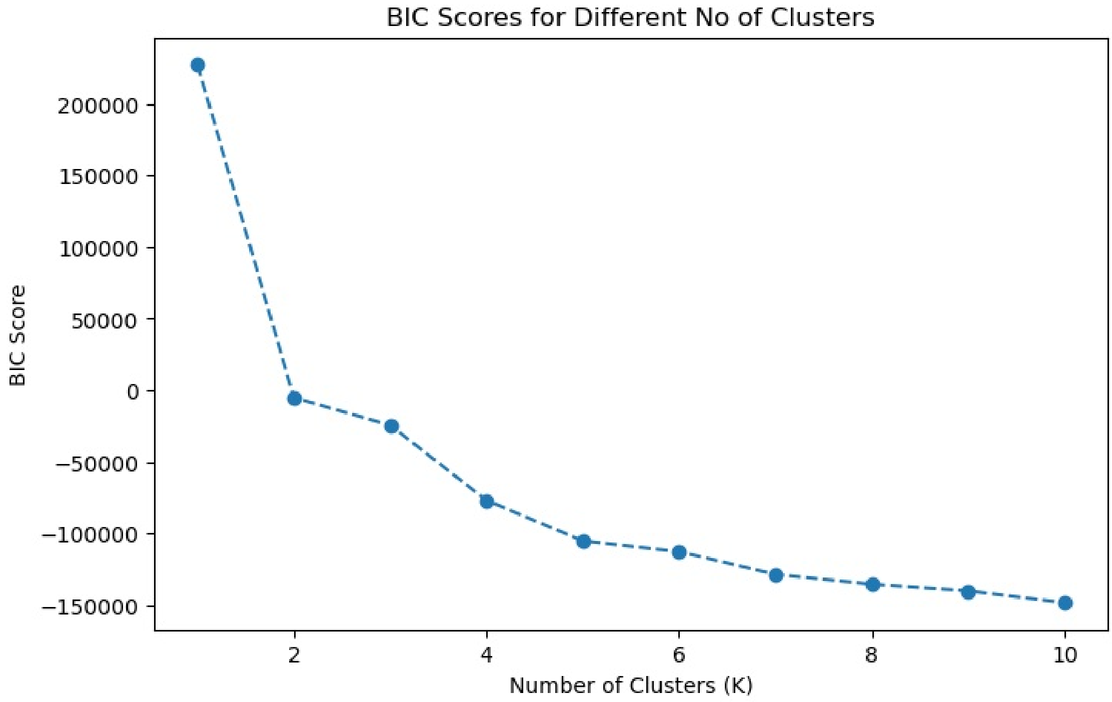
2.) Gaussian Mixtures

In [ ]:

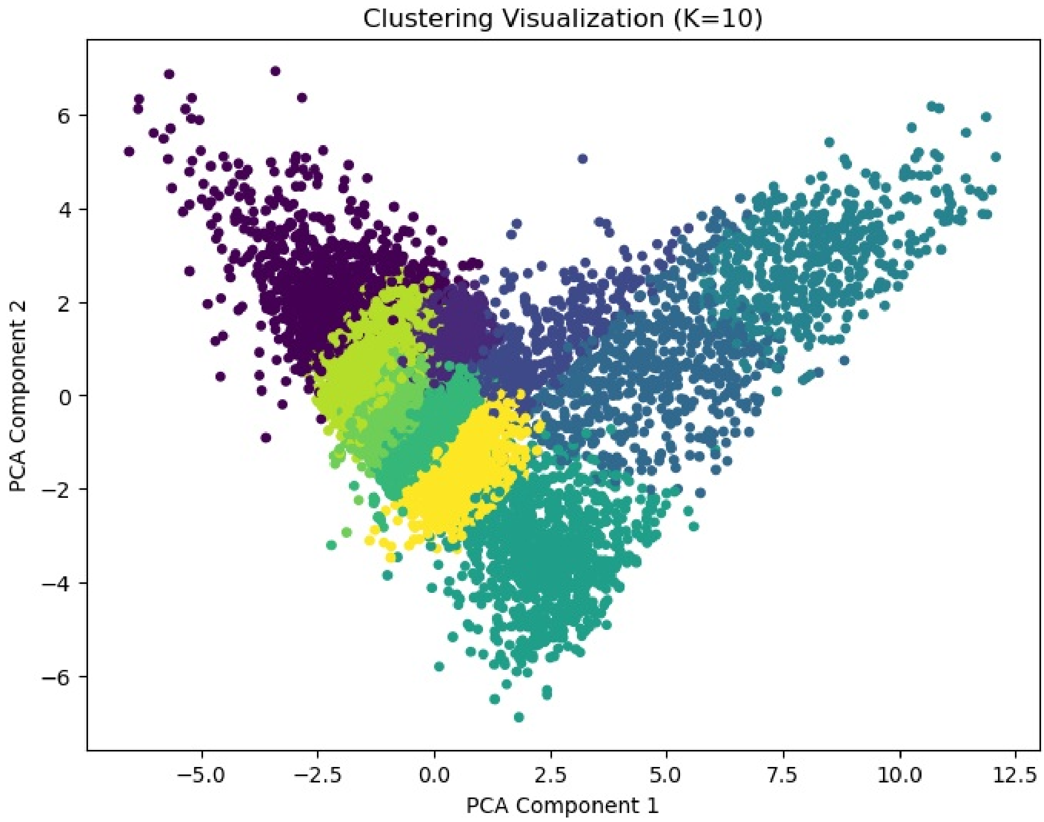
In [54]:
```
file_path = 'HTRU_2.csv'
data = pd.read_csv(file_path, header=None)

# need to set the first row as headers and remove it from data
data.columns = data.iloc[0]
data = data[1:].reset_index(drop=True)

# Converting columns to numeric and dropping rows with NaNs...
data = data.apply(pd.to_numeric, errors='coerce').dropna().reset_index(drop=True)
```

In [68]:
```
scaler = StandardScaler()
data_normalized = scaler.fit_transform(data)

K_range = range(1, 11)

# fitting Gaussian mixtures for each K and getting BIC
bic_scores = []
for K in K_range:
    gmm = GaussianMixture(n_components=K, random_state=42)
    gmm.fit(data_normalized)
    bic_scores.append(gmm.bic(data_normalized))

# Plot BIC scores to get the most optimal number of clusters
plt.figure(figsize=(8, 5))
plt.plot(K_range, bic_scores, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('BIC Score')
plt.title('BIC Scores for Different No of Clusters')
plt.show()

# Select the optimal K (lowest BIC)
optimal_K = K_range[np.argmin(bic_scores)]
print(f"Optimal number of clusters (K): {optimal_K}")

# Fitting Gaussian mixtures with optimal K
gmm = GaussianMixture(n_components=optimal_K, random_state=42)
```

```
gmm.fit(data_normalized)

# Predicting soft cluster assignments (probabilities)
soft_clusters = gmm.predict_proba(data_normalized)

# Predict hard cluster
hard_clusters = gmm.predict(data_normalized)

# Visualize the clustering results using PCA
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_normalized)

plt.figure(figsize=(8, 6))
plt.scatter(data_pca[:, 0], data_pca[:, 1], c=hard_clusters, cmap='viridis', s=15)
#plt.colorbar(label='Cluster')
plt.title(f'Clustering Visualization (K={optimal_K})')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```




```
Optimal number of clusters (K): 10
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js