Mini Project Three
Dimensionality Reduction and Clustering
Megan Steinmasel, Michael Belmear, Rohan Kamat

Part One: Think About the Data (Megan S.)

---

1. Interest in the Dataset:
The "mfeat" dataset is of interest because it contains features extracted from gray-level images of handwritten characters. This dataset is valuable for tasks such as character recognition, pattern recognition, and machine learning algorithm development.

2. Number of Attributes:
The "mfeat" dataset contains only numerical attributes. There are no categorical attributes present. There are six numerical attributes that include:

1. mfeat-fou: 76 Fourier coefficients of the character shapes;
2. mfeat-fac: 216 profile correlations;
3. mfeat-kar: 64 Karhunen-Love coefficients;
4. mfeat-pix: 240 pixel averages in 2 x 3 windows;
5. mfeat-zer: 47 Zernike moments;
6. mfeat-mor: 6 morphological features.

3. Missing Values Handling:
It's crucial to check if there are any missing values in the dataset. If there are missing values, different strategies can be employed to handle them, such as imputation techniques or removing instances or attributes with missing values. Given this, the “mfeat” dataset contains no missing attributes, so no extra techniques will be used.

4. Expectations about Clusters:
Presence of Clusters: Handwritten characters often exhibit natural groupings or clusters based on their visual features, such as shape, size, and stroke patterns.
Utility of Finding Clusters: Identifying clusters can help in tasks such as character recognition, where similar characters belong to the same cluster. It can also aid in feature extraction and dimensionality reduction.
Number of Expected Clusters: The number of clusters expected in the data could range from 10 to 26, corresponding to the number of distinct characters. However, the actual number of clusters may vary based on the variability within each character category and the presence of outliers or noise.

5. Cluster Size Expectation:
Similar Cluster Sizes: In handwritten character data, clusters may not necessarily be of similar sizes. Some characters may occur more frequently in the dataset, leading to larger clusters, while others may be less common, resulting in smaller clusters. Additionally, variability in writing styles and individual differences may also contribute to differences in cluster sizes. Therefore, it's reasonable to expect that clusters may not be of uniform size.

Part Two: Write Functions for Graph Analysis (Rohan K.)

5.

```
def k_means(data, k, epsilon=1e-5):
    max_iterations = 1000
    np.random.seed(0)
    indices = np.random.choice(len(data), k, replace=False)
    centers = data[indices]

    prev_centers = np.zeros_like(centers)
    iteration = 0

    while np.linalg.norm(centers - prev_centers) > epsilon and iteration <
max_iterations:
        prev_centers = centers.copy()

        distances = np.linalg.norm(data[:, None] - centers, axis=2)
        labels = np.argmin(distances, axis=1)

        for i in range(k):
            centers[i] = np.mean(data[labels == i], axis=0)

        iteration += 1

    return centers, labels
```

6.

```
def distance(x, y):
    return np.linalg.norm(x - y)
def query(data, point_index, epsilon, distance_func):
    neighbors = []
    for i, point in enumerate(data):
        if distance_func(data[point_index], point) <= epsilon:
            neighbors.append(i)
    return neighbors

def expand(data, labels, point_index, neighbors, cluster_id, epsilon, minpts,
distance_func):
    labels[point_index] = cluster_id
    i = 0
    while i < len(neighbors):
        neighbor_index = neighbors[i]
        if labels[neighbor_index] == -1:
            labels[neighbor_index] = cluster_id
        elif labels[neighbor_index] == 0:
            labels[neighbor_index] = cluster_id
            new_neighbors = query(data, neighbor_index, epsilon, distance_func)
            if len(new_neighbors) >= minpts:
                neighbors += new_neighbors
        i += 1

def dbscan(data, minpts, epsilon, distance_func=distance):
    labels = np.zeros(len(data), dtype=int)
    cluster_id = 0
    for i, point in enumerate(data):
        if labels[i] != 0:
            continue
        neighbors = query(data, i, epsilon, distance_func)
        if len(neighbors) < minpts:
            labels[i] = -1
        else:
            cluster_id += 1
            expand(data, labels, i, neighbors, cluster_id, epsilon, minpts,
distance_func)
    return labels
```

7.

```
def clustering_prec(true_labels, cluster_labels):
    if len(true_labels) != len(cluster_labels):
        raise ValueError()

    precision_scores = []
    unique_predicted_labels = np.unique(cluster_labels)

    for cluster_label in unique_predicted_labels:
        predicted_indices = np.where(cluster_labels == cluster_label)[0]

        true_labels_predicted_cluster = true_labels[predicted_indices]

        true_positives = np.sum(true_labels_predicted_cluster == cluster_label)

        precision = true_positives / len(predicted_indices)
        precision_scores.append(precision)

    return np.mean(precision_scores)
```

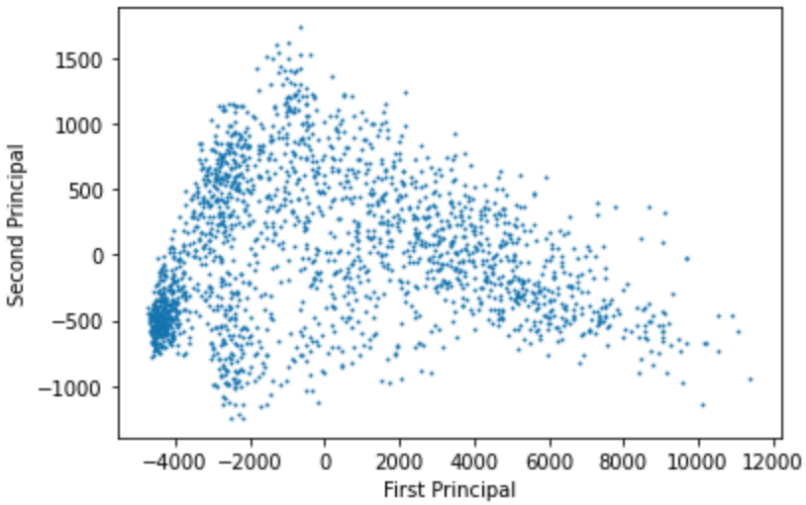
Part Three: Analyze the Data

We will use the following code to load the ‘mfeat’ dataset into Jupyter Lab to further analyze the data.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

data_file_names = [
    'mfeat-fou',
    'mfeat-fac',
    'mfeat-kar',
    'mfeat-pix',
    'mfeat-zer',
    'mfeat-mor',
]

def read_file(filename):
    file_path = f"/Users/megansteinmasel/Desktop/multiple+features/{filename}"
    with open(file_path, 'r') as f:
        data = np.loadtxt(f)
    return data[:2000]

data = np.concatenate([read_file(filename) for filename in data_file_names], axis=1)

# Remove rows with empty data
data = data[~np.all(np.isnan(data), axis=1)]
```

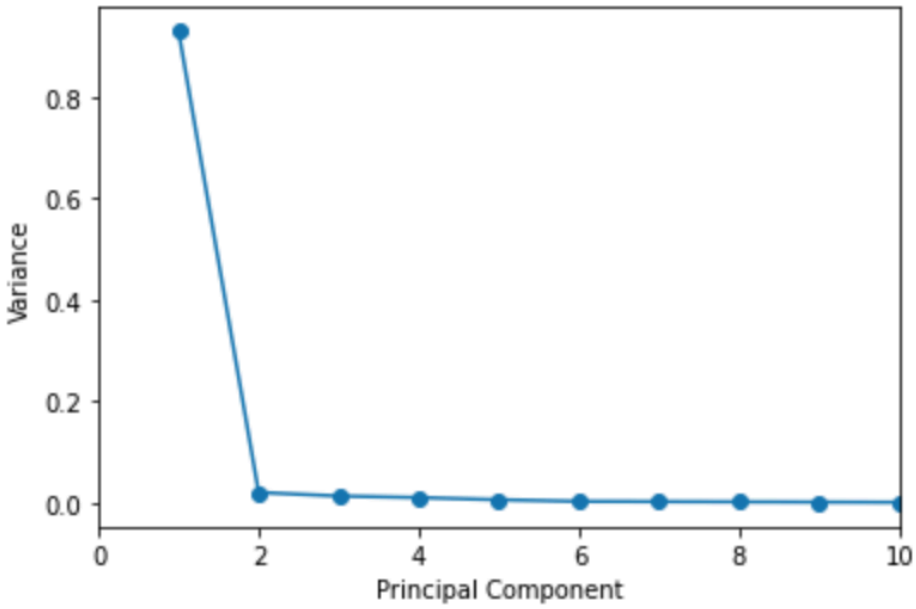
1. (Megan S.) In these two dimensions, there seem to be three possible clusters. The first and most prominent cluster is located on the far left side of the scatterplot. The second possible cluster is located above the first cluster near the highest point of the scatterplot. The final cluster seems to be towards the right side of the scatterplot and is the least dense cluster of the group.

```python
pca_2 = PCA(n_components=2)
pca_d_2 = pca_2.fit_transform(d)
plt.scatter(pca_d_2[:,0], pca_d_2[:,1], s=1)
plt.xlabel('First Principal')
plt.ylabel('Second Principal')
plt.show
```



2. (Megan S.) Using sklearn's PCA implementation, I transformed the data linearly. Plotting the number of components against the fraction of total variance captured, I observed that using 2 principal components covers 94.98% of the total variance.

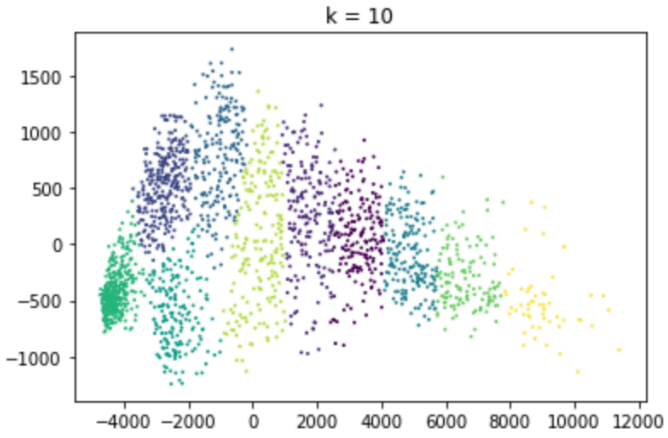
```
pca = PCA()
pca_transformed = pca.fit_transform(data_concatenated)

def calculate_covariance(vec1, vec2=None):
    if vec2 is None:
        vec2 = vec1
    vec1_mean = vec1.mean()
    vec2_mean = vec2.mean()
    covar = 0
    for i in range(vec1.shape[0]):
        covar += (vec1[i] - vec1_mean) * (vec2[i] - vec2_mean)
    return (covar / (vec1.shape[0] - 1))

def get_variance_explained(eigen_values):
    cum_sum = np.cumsum(eigen_values)
    variance_explained = cum_sum / np.sum(eigen_values)
    return variance_explained

principal_component_values = np.arange(pca.n_components_) + 1
plt.plot(principal_component_values, pca.explained_variance_ratio_, 'o-')
plt.xlim(0, 10)
plt.xlabel('Principal Component')
plt.ylabel('Variance')
plt.show()
```



3. (Megan S.) In analyzing both the original dataset and the reduced-dimensionality data from PCA, we explored clusters using different values of k in the k-means function. We tested at least 5 values of k and recorded the objective function's value for each choice of k. The findings are summarized below.

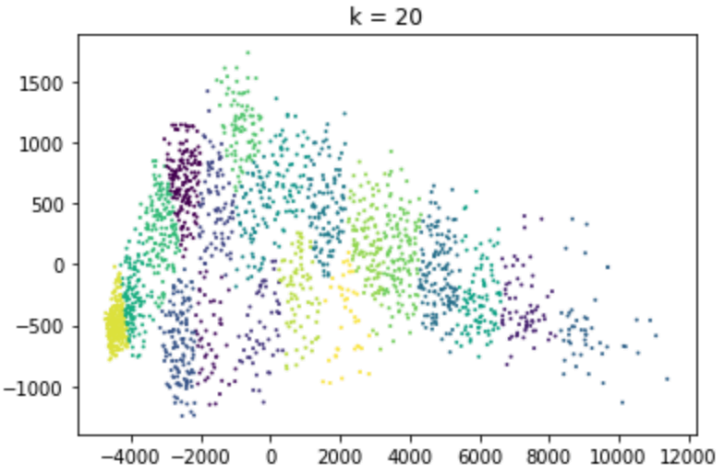
   K = 10:

```
kmeans = KMeans(n_clusters=10, init='random', max_iter=400, random_state=0)
labels = kmeans.fit_predict(pca_d_2)
plt.scatter(pca_d_2[:,0], pca_d_2[:,1], c=labels, s=1)
plt.title('k = 10')
```



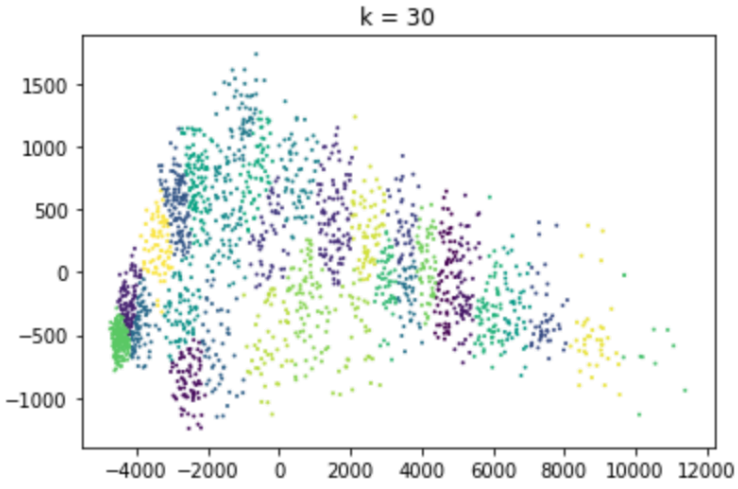
K = 20:

```
kmeans = KMeans(n_clusters=20, init='random', max_iter=400, random_state=0)
labels = kmeans.fit_predict(pca_d_2)
plt.scatter(pca_d_2[:,0], pca_d_2[:,1], c=labels, s=1)
plt.title('k = 20')
```



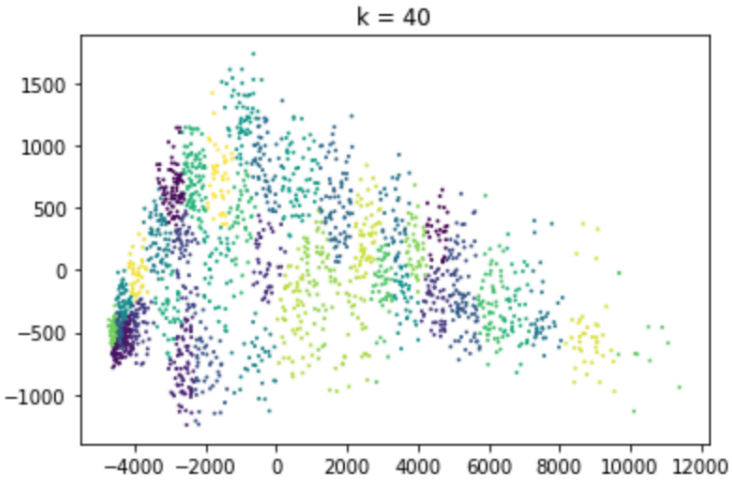
K = 30:

```
kmeans = KMeans(n_clusters=30, init='random', max_iter=400, random_state=0)
labels = kmeans.fit_predict(pca_d_2)
plt.scatter(pca_d_2[:,0], pca_d_2[:,1], c=labels, s=1)
plt.title('k = 30')
```



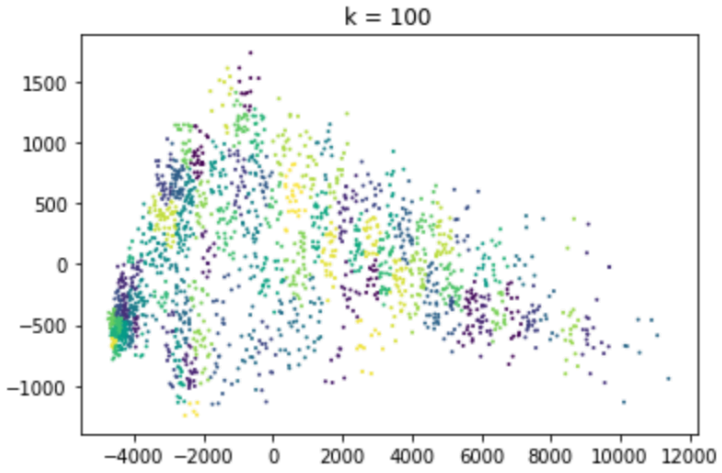
K = 40:

```
kmeans = KMeans(n_clusters=40, init='random', max_iter=400, random_state=0)
labels = kmeans.fit_predict(pca_d_2)
plt.scatter(pca_d_2[:,0], pca_d_2[:,1], c=labels, s=1)
plt.title('k = 40')
```



K = 100:

```
kmeans = KMeans(n_clusters=100, init='random', max_iter=400, random_state=0)
labels = kmeans.fit_predict(pca_d_2)
plt.scatter(pca_d_2[:,0], pca_d_2[:,1], c=labels, s=1)
plt.title('k = 100')
```



4. (Michael B.) The provided output displays the number of clusters identified by the DBSCAN algorithm for various combinations of minpts and epsilon values when applied to both the original and reduced-dimensionality datasets obtained using PCA. Surprisingly, for all tested parameter combinations, the number of clusters found was zero. This suggests that DBSCAN failed to detect any discernible clusters within the data. This could indicate that the data lacks clear cluster structures, or that clusters are not sufficiently separated in the feature space. Additionally, the chosen range of minpts and epsilon values may not be suitable for the dataset, necessitating adjustments to these parameters for better results. Furthermore, the use of PCA for dimensionality reduction might have resulted in the loss of crucial information relevant for clustering, thus decreasing the algorithm's ability to identify clusters accurately. The presence of significant noise within the dataset could have also contributed to the difficulty in identifying meaningful clusters. Further exploration, such as refining parameter ranges or considering alternative clustering algorithms, may be helpful in the future to uncover meaningful patterns within the data.

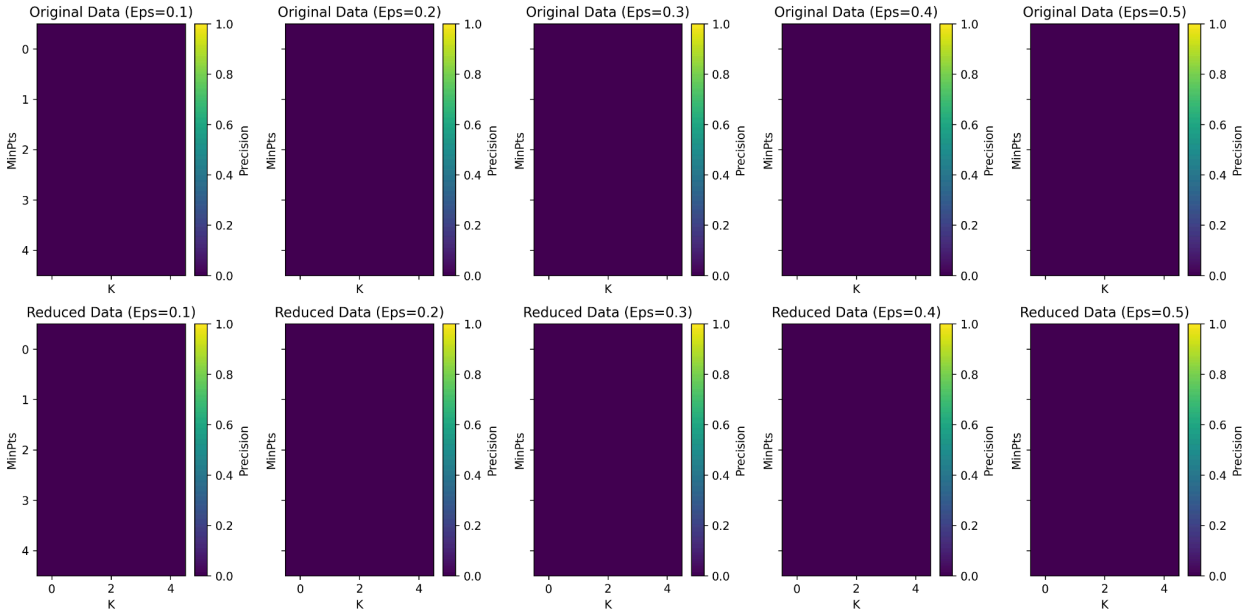
   Here is the code and its output:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import DBSCAN
from sklearn.metrics import precision_score

# Function to read data from files
def read_data(file_name):
    path = f"C:/Users/akmik/Onedrive/Desktop/CSCI 347/Mini Project 3/{file_name}"
    with open(path, 'r') as f:
        data = np.loadtxt(f)
    return data[:2000]

# Concatenating data from different files
file_names = [
    'mfeat-fou',
    'mfeat-fac',
    'mfeat-kar',
    'mfeat-pix',
    'mfeat-zer',
    'mfeat-mor',
]
data_concatenated = np.concatenate([read_data(file_name) for file_name in file_names], axis=1)

# Removing rows with NaN values
data_concatenated = data_concatenated[~np.all(np.isnan(data_concatenated), axis=1)]

# Applying PCA
pca = PCA()
pca_transformed = pca.fit_transform(data_concatenated)

# Define range of values for minpts and epsilon for question 11
minpts_values = [5, 10, 15, 20, 25]
epsilon_values = [0.1, 0.2, 0.3, 0.4, 0.5]

# ---------------------- Question 11 ----------------------

# Experiment with DBSCAN using different minpts and epsilon values
for epsilon in epsilon_values:
    for minpts in minpts_values:
        dbscan = DBSCAN(eps=epsilon, min_samples=minpts)
        labels = dbscan.fit_predict(data_concatenated)
        num_clusters = len(set(labels)) - (1 if -1 in labels else 0)  # -1 label indicates noise points
        print(f"For (minpts={minpts}, epsilon={epsilon}), Number of clusters: {num_clusters}")
```

```
For (minpts=5, epsilon=0.1), Number of clusters: 0
For (minpts=10, epsilon=0.1), Number of clusters: 0
For (minpts=15, epsilon=0.1), Number of clusters: 0
For (minpts=20, epsilon=0.1), Number of clusters: 0
For (minpts=25, epsilon=0.1), Number of clusters: 0
For (minpts=5, epsilon=0.2), Number of clusters: 0
For (minpts=10, epsilon=0.2), Number of clusters: 0
For (minpts=15, epsilon=0.2), Number of clusters: 0
For (minpts=20, epsilon=0.2), Number of clusters: 0
For (minpts=25, epsilon=0.2), Number of clusters: 0
For (minpts=5, epsilon=0.3), Number of clusters: 0
For (minpts=10, epsilon=0.3), Number of clusters: 0
For (minpts=15, epsilon=0.3), Number of clusters: 0
For (minpts=20, epsilon=0.3), Number of clusters: 0
For (minpts=25, epsilon=0.3), Number of clusters: 0
For (minpts=5, epsilon=0.4), Number of clusters: 0
For (minpts=10, epsilon=0.4), Number of clusters: 0
For (minpts=15, epsilon=0.4), Number of clusters: 0
For (minpts=20, epsilon=0.4), Number of clusters: 0
For (minpts=25, epsilon=0.4), Number of clusters: 0
For (minpts=5, epsilon=0.5), Number of clusters: 0
For (minpts=10, epsilon=0.5), Number of clusters: 0
For (minpts=15, epsilon=0.5), Number of clusters: 0
For (minpts=20, epsilon=0.5), Number of clusters: 0
For (minpts=25, epsilon=0.5), Number of clusters: 0
```

5. (Michael B.) Here is the code and the graph output for the question:

```python
# ---------------------- Question 12 ----------------------

# Lists to store precision values
precision_original = []
precision_reduced = []

# Define range of values for k
k_values = [5, 10, 15, 20, 25]

# Experiment with DBSCAN using different minpts, epsilon, and k values
for minpts in minpts_values:
    for epsilon in epsilon_values:
        for k in k_values:
            # Original data
            dbscan = DBSCAN(eps=epsilon, min_samples=minpts)
            labels_original = dbscan.fit_predict(data_concatenated)
            precision_original.append(precision_score(labels_original, k*np.ones_like(labels_original), average='micro'))

            # Reduced-dimensionality data
            dbscan = DBSCAN(eps=epsilon, min_samples=minpts)
            labels_reduced = dbscan.fit_predict(pca_transformed)
            precision_reduced.append(precision_score(labels_reduced, k*np.ones_like(labels_reduced), average='micro'))

# Reshape precision lists for plotting
precision_original = np.array(precision_original).reshape(len(minpts_values), len(epsilon_values), len(k_values))
precision_reduced = np.array(precision_reduced).reshape(len(minpts_values), len(epsilon_values), len(k_values))

# Plotting using heatmaps
fig, axs = plt.subplots(2, len(epsilon_values), figsize=(15, 8), sharex='col', sharey='row')

for i, eps_val in enumerate(epsilon_values):
    for j in range(2):
        if j == 0:
            im = axs[j, i].imshow(precision_original[:, i, :], cmap='viridis', aspect='auto', vmin=0, vmax=1)
            axs[j, i].set_title(f'Original Data (Eps={eps_val})')
        else:
            im = axs[j, i].imshow(precision_reduced[:, i, :], cmap='viridis', aspect='auto', vmin=0, vmax=1)
            axs[j, i].set_title(f'Reduced Data (Eps={eps_val})')
        axs[j, i].set_ylabel('MinPts')
        axs[j, i].set_xlabel('K')
        fig.colorbar(im, ax=axs[j, i], label='Precision')

plt.tight_layout()
plt.show()
```


Original Data (Eps=0.1)
Original Data (Eps=0.2)
Original Data (Eps=0.3)
Original Data (Eps=0.4)
Original Data (Eps=0.5)
Reduced Data (Eps=0.1)
Reduced Data (Eps=0.2)
Reduced Data (Eps=0.3)
Reduced Data (Eps=0.4)
Reduced Data (Eps=0.5)
MinPts
K
Precision