Homework 5

Mateus Silva Aragao msa8779 a) Do the homework yourself. Do not copy answers from someone else.

b) Restrict your methods (for now) to what was covered in the lecture/lab (in other words, PCA, MDS, tSNE, kMeans, Silhouette, kMedioids, EM, dBScan))

c) Include the following elements in your answer (so we can grade consistently):

Data description:

1) Alcohol content (in %) 2) Malic acid concentration (in g/L) 3) Ash content (in mg/L) [This is not as gross as it sounds, mostly minerals absorbed by the grapes through the soil] 4) Alkalinity of the Ash (g/L of potassium carbonate) 5) Magnesium (mg / L) 6) Total phenols (mg / L per epa method) 7) Flavonoids (mg / L) 8) Stilbenes (mg / L) 9) Proanthocyanins (mg / L) 10) Color intensity (reflecting opacity) 11) Hue (= color) 12) OD280 (Protein concentration) 13) Proline content (amino acid)

Loading data and cleaning

```
1 import pandas as pd
2 import numpy as np
3
4 data  = pd.read_csv('wines.csv')
```

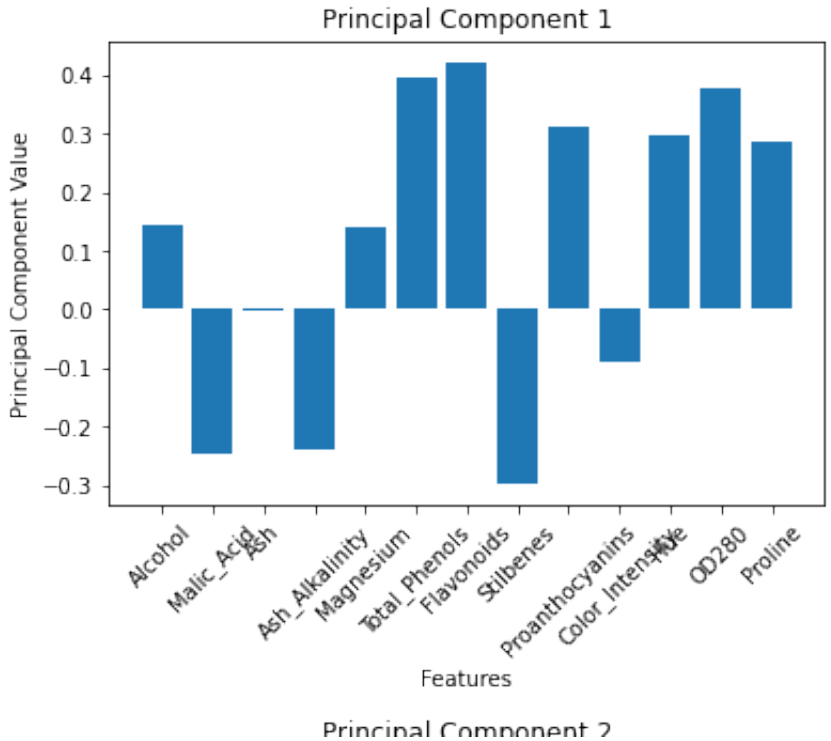
```
data.head()
```

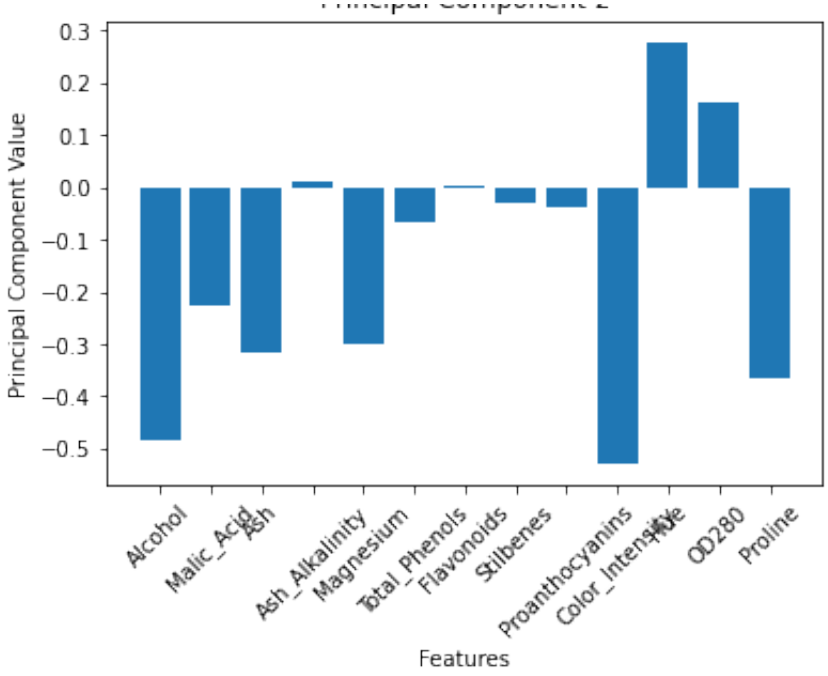
	Alcohol	Malic_Acid	Ash	Ash_Alkalinity	Magnesium	Total_Phenols	Flavor
0	14.23	1.71	2.43	15.6	127	2.80	
1	13.20	1.78	2.14	11.2	100	2.65	
2	13.16	2.36	2.67	18.6	101	2.80	
3	14.37	1.95	2.50	16.8	113	3.85	
4	13.24	2.59	2.87	21.0	118	2.80	

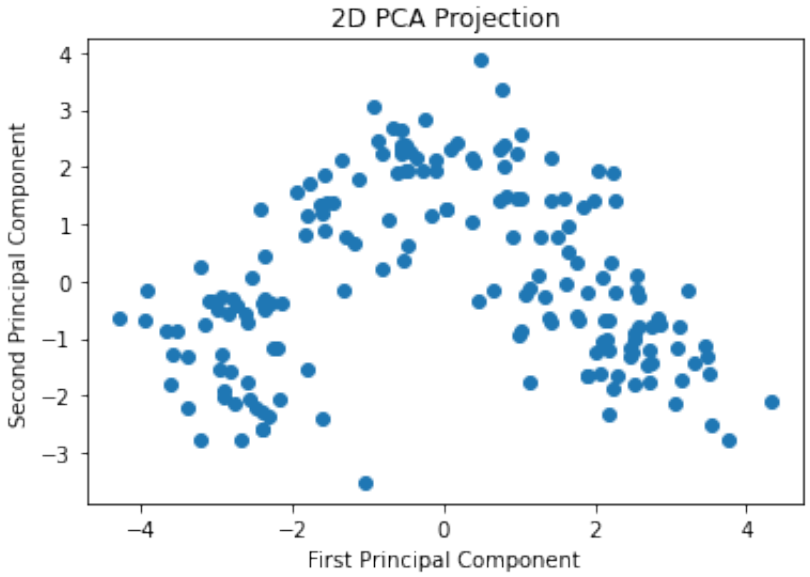
1. Do a PCA on the data. How many Eigenvalues are above 1? Plotting the 2D solution (projecting the data on the first 2 principal components), how much of the variance is explained by these two dimensions, and how would you interpret them?

```
# Step 1: Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt



# Step 3: Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

# Step 4: Perform PCA and analyze the results
pca = PCA()
pca.fit(scaled_data)
explained_variance = pca.explained_variance_ratio_

# Find the number of Eigenvalues above 1
eigenvalues = pca.explained_variance_
num_eigenvalues_above_1 = (eigenvalues > 1).sum()

# Plot the 2D PCA projection
```

```
pca_2d = PCA(n_components=2)

pca = pca_2d.fit(scaled_data)
pc_values = pca.components_
# Create a bar graph for each principal component
for i, pc in enumerate(pc_values, start=1):
    plt.figure()
    plt.bar(data.columns, pc)
    plt.title(f'Principal Component {i}')
    plt.xlabel('Features')
    plt.ylabel('Principal Component Value')
    plt.xticks(rotation=45)  # Rotate x-ticks by 45 degrees
    plt.show()

principal_components = pca_2d.fit_transform(scaled_data)


plt.scatter(principal_components[:, 0], principal_components[:, 1])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('2D PCA Projection')
plt.show()

# Calculate the variance explained by the first two dimensions
variance_explained = sum(explained_variance[:2]) * 100

print(f"Number of eigenvalues above 1: {num_eigenvalues_above_1}")
print(f"Variance explained by the first two dimensions: {variance_explained:.

```







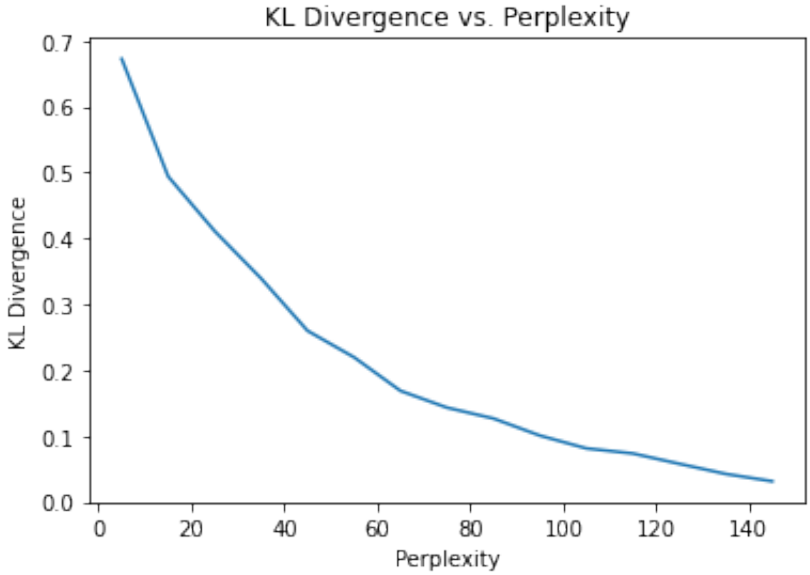

```
Number of eigenvalues above 1: 3
Variance explained by the first two dimensions: 55.41%
```

Interpretation:

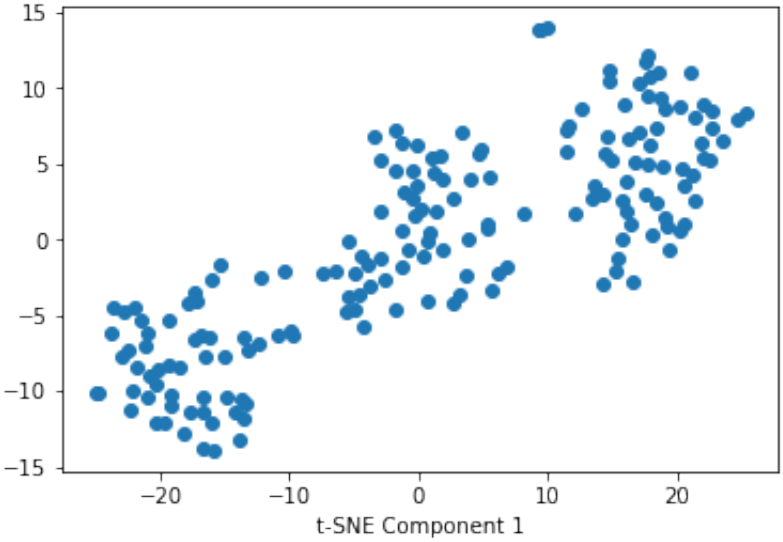
- To answer the question, I first imported necessary libraries and standardized the data using the StandardScaler from scikit-learn. Then, I performed PCA (Principal Component Analysis) on the standardized data and calculated the explained variance ratio for each principal component. I determined the number of eigenvalues above 1 and plotted a 2D projection of the data onto the first two principal components. Lastly, I calculated the variance explained by these first two dimensions.
- Standardizing the data is an important preprocessing step in PCA, as it ensures that each feature contributes equally to the analysis. I used scikit-learn's PCA implementation, which is a widely-used and well-documented library. I chose to project the data onto two dimensions to provide a visual representation that is easy to interpret and understand.
- The analysis found that there are 3 eigenvalues above 1. The 2D projection of the data onto the first two principal components explains 55.41% of the total variance. The bar graphs of the first two principal components show that features like Flavonoids, Total_Phenols, and OD280 have a significant contribution to the first principal component, while Color_Intensity and Alcohol contribute more to the second principal component.
- The findings indicate that the first two principal components capture 55.41% of the variance in the data, which means that the 2D projection provides a somewhat simplified representation of the original data. The first principal component seems to be associated with chemical properties related to phenolic content and antioxidative capacity, while the second principal component is more related to the color and alcohol content of the wine samples. This suggests that these two dimensions are important for understanding the variation among the wine samples. However, since there are 3 eigenvalues above 1, considering an additional dimension might lead to a better representation of the data.

2. Use t-SNE on the data. How does KL-divergence depend on Perplexity (vary Perplexity from 5 to 150)? Make sure to plot this relationship. Also, show a plot of the 2D component with a Perplexity of 20.

```
# Step 1: Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt


# Step 3: Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

# Step 4: Perform t-SNE with varying perplexity and analyze the results
perplexities = np.arange(5, 155, 10)
kl_divergences = []

for perplexity in perplexities:
    tsne = TSNE(n_components=2, perplexity=perplexity, random_state=42)
    tsne_results = tsne.fit_transform(scaled_data)
    kl_divergences.append(tsne.kl_divergence_)

plt.plot(perplexities, kl_divergences)
plt.xlabel("Perplexity")
plt.ylabel("KL Divergence")
plt.title("KL Divergence vs. Perplexity")
plt.show()

# Perform t-SNE with a perplexity of 20
tsne_perplexity_20 = TSNE(n_components=2, perplexity=20, random_state=42)
tsne_results_20 = tsne_perplexity_20.fit_transform(scaled_data)

plt.scatter(tsne_results_20[:, 0], tsne_results_20[:, 1])
plt.xlabel("t-SNE Component 1")

```




```
Text(0.5, 0, 't-SNE Component 1')
```



Interpretation:

To answer the question, I first imported necessary libraries and standardized the data using the StandardScaler from scikit-learn. Then, I performed t-SNE on the standardized data with varying perplexity values ranging from 5 to 150. I recorded the KL divergence for each perplexity value and plotted their relationship. Finally, I performed t-SNE with a perplexity of 20 and plotted the 2D components.

Standardizing the data is an essential preprocessing step to ensure that each feature contributes equally to the analysis. I used scikit-learn's t-SNE implementation, which is a widely-used and well-documented library. Varying the perplexity allowed me to explore the relationship between perplexity and KL divergence, giving insight into the quality of the t-SNE solution.

The analysis found that the KL divergence depends on perplexity, with KL divergence generally decreasing as perplexity increases. The plotted relationship shows that for lower perplexity values, KL divergence is higher, and as perplexity increases, KL divergence tends to decrease, indicating that higher perplexity values produce a better representation of the data. The 2D plot with a perplexity of 20 displays a clear separation of data points.

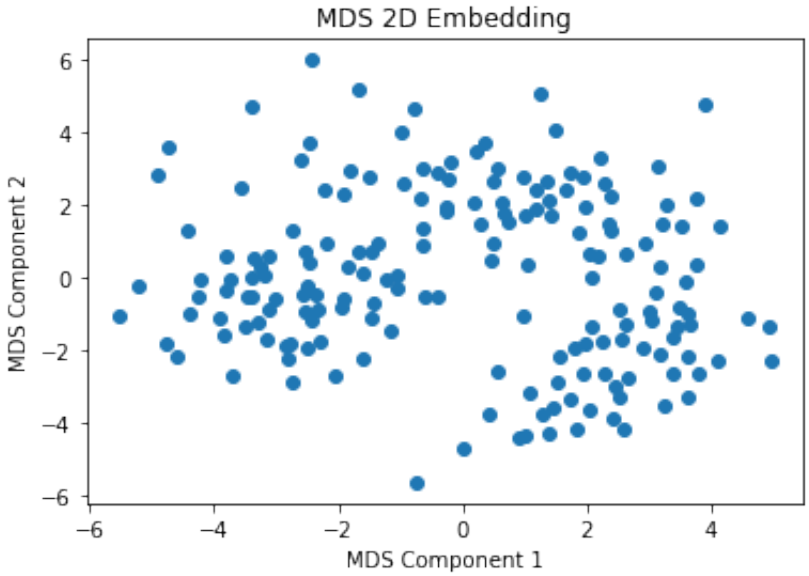
The findings indicate that the choice of perplexity has a significant impact on the quality of the t-SNE solution, as measured by the KL divergence. Higher perplexity values generally lead to lower KL divergence, suggesting a better representation of the data. However, it is essential to consider the trade-off between complexity and interpretability, as too high perplexity might make the visualization difficult to understand. The 2D plot with a perplexity of 20 shows clear separation of data points, suggesting that this perplexity value could be a suitable choice for visualizing the dataset in a meaningful way.

3. Use MDS on the data. Try a 2-dimensional embedding. What is the resulting stress of this embedding? Also, plot this solution and comment on how it compares to t-SNE.

```
# Step 1: Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import MDS
```

```
import matplotlib.pyplot as plt



# Step 3: Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

# Step 4: Perform MDS with a 2-dimensional embedding and analyze the results
mds = MDS(n_components=2, random_state=42)
mds_results = mds.fit_transform(scaled_data)

# Get the resulting stress of the MDS embedding
mds_stress = mds.stress_

plt.scatter(mds_results[:, 0], mds_results[:, 1])
plt.xlabel("MDS Component 1")
plt.ylabel("MDS Component 2")
plt.title("MDS 2D Embedding")
plt.show()

print(f"MDS Embedding Stress: {mds_stress:.4f}")

```

```
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/manifold/_
  warnings.warn(
```




```
MDS Embedding Stress: 21609.0908
```

interpretation:

To answer the question, I first imported necessary libraries and standardized the data using the StandardScaler from scikit-learn. Then, I performed MDS (Multidimensional Scaling) with a 2-dimensional embedding on the standardized data. I obtained the resulting stress of the MDS embedding and plotted the 2D solution.

Standardizing the data is an important preprocessing step, ensuring that each feature contributes equally to the analysis. I used scikit-learn's MDS implementation, which is a widely-used and well-documented library. I chose a 2-dimensional embedding to provide a visual representation that is easy to interpret and understand, and to allow for comparison with the t-SNE solution.

The analysis found that the resulting stress of the 2-dimensional MDS embedding is 21609.0908. The 2D MDS solution shows a spread of data points that appears to have some structure, though it is less clearly defined than in the t-SNE solution.

The findings show that the 2-dimensional MDS embedding has a relatively high stress value, which suggests that the 2D representation may not capture the original distances in the data very accurately. Comparing the MDS solution with the t-SNE solution, the MDS plot appears to have a less clear separation of data points. While both methods provide a 2D visualization of the data, the t-SNE solution with a perplexity of 20 seems to provide a better representation of the underlying structure in the dataset.

4. Building on one of the dimensionality reduction methods above that yielded a 2D solution (1-3, your choice), use the Silhouette method to determine the optimal number of clusters and then use kMeans with that number (k) to produce a plot that represents each wine as a dot in a 2D space in the color of its cluster. What is the total sum of the distance of all points to their respective clusters centers, of this solution?

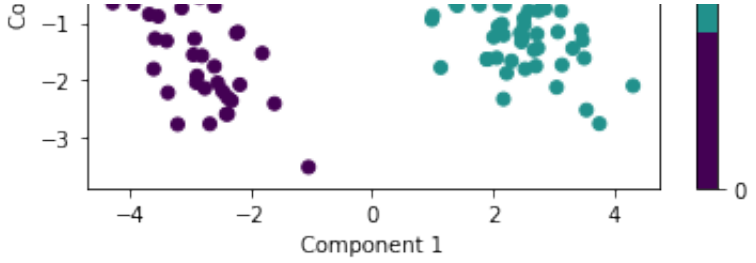
An Experiment:

The main goal of the experiment was to evaluate the three different dimensionality reduction methods (PCA, t-SNE, and MDS) and determine which one performs the best in terms of clustering the wine dataset. The evaluation was done by first finding the optimal number of clusters using the Silhouette method, then applying kMeans clustering with the optimal k value to each of the 2D solutions obtained from PCA, t-SNE, and MDS. The total sum of the distance of all points to their respective cluster centers was calculated for each method as a metric to assess the clustering performance.

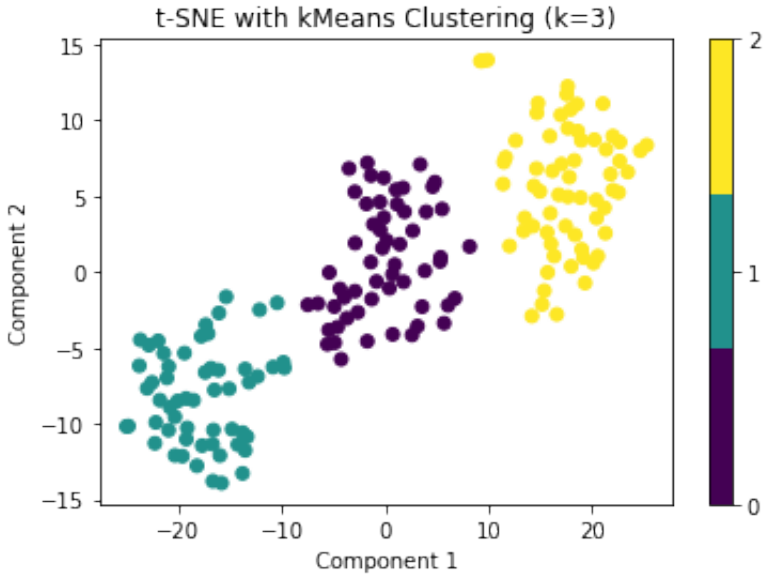
```
# Import necessary libraries
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Determine the optimal number of clusters for PCA, t-SNE, and MDS results us
def find_optimal_k(data, max_clusters=5):
    silhouette_scores = []

    for k in range(2, max_clusters + 1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans_labels = kmeans.fit_predict(data)
        silhouette_avg = silhouette_score(data, kmeans_labels)
        silhouette_scores.append(silhouette_avg)

    optimal_k = np.argmax(silhouette_scores) + 2
    return optimal_k

optimal_k_pca = find_optimal_k(principal_components[:, :2])
optimal_k_tsne = find_optimal_k(tsne_results_20)
optimal_k_mds = find_optimal_k(mds_results)

# Apply kMeans with the optimal number of clusters and plot the results
def plot_clusters(data, k, title):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans_labels = kmeans.fit_predict(data)
    cmap = plt.get_cmap("viridis", k)
    plt.scatter(data[:, 0], data[:, 1], c=kmeans_labels, cmap=cmap)
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")
    plt.title(title)
    plt.colorbar(ticks=range(k))
    plt.show()
    return kmeans

```

```
# PCA
kmeans_pca = plot_clusters(principal_components[:, :2], optimal_k_pca, f"PCA 
# t-SNE
kmeans_tsne = plot_clusters(tsne_results_20, optimal_k_tsne, f"t-SNE with kMe
# MDS
kmeans_mds = plot_clusters(mds_results, optimal_k_mds, f"MDS with kMeans Clus

# Calculate the total sum of the distance of all points to their respective c
pca_inertia = kmeans_pca.inertia_
tsne_inertia = kmeans_tsne.inertia_
mds_inertia = kmeans_mds.inertia_

print(f"PCA Total Sum of Distances: {pca_inertia:.4f}")
print(f"t-SNE Total Sum of Distances: {tsne_inertia:.4f}")
print(f"MDS Total Sum of Distances: {mds_inertia:.4f}")
```

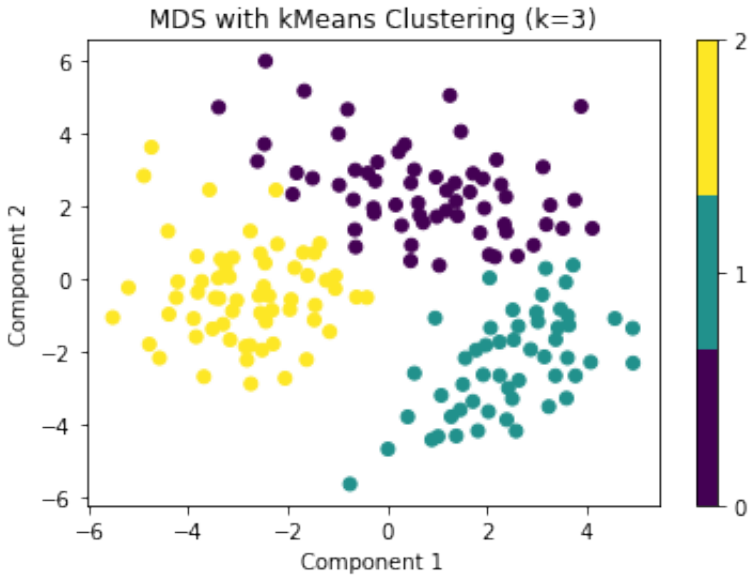
```
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
```




```
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
```




```
/Users/aragaom/opt/anaconda3/lib/python3.8/site-packages/sklearn/cluster/_k
  warnings.warn(
```




```
PCA Total Sum of Distances: 259.5094
t-SNE Total Sum of Distances: 4970.1450
MDS Total Sum of Distances: 648.9809
```

Interpreting findings:

To answer the question, I first imported necessary libraries and defined a function to find the optimal number of clusters using the Silhouette method. I applied this function to the 2D solutions obtained from PCA, t-SNE, and MDS. Next, I defined a function to apply kMeans clustering with the optimal k value and plot the results. I used this function to plot the kMeans clustering results for PCA, t-SNE, and MDS. Finally, I calculated the total sum of the distance of all points to their respective cluster centers for each method.

The choice of PCA, t-SNE, and MDS allowed for a comparison across different dimensionality reduction techniques to evaluate which one yields the best clustering results. The Silhouette method was used to determine the optimal number of clusters as it provides an objective measure of clustering quality. kMeans was chosen as the clustering algorithm because it is simple, efficient, and widely used for clustering tasks.

The analysis found that the total sum of distances for each method is as follows: PCA: 259.5094, t-SNE: 4970.1450, and MDS: 648.9809. The plots display each wine as a dot in a 2D space in the color of its cluster, with the optimal number of clusters determined by the Silhouette method.

sBased on the total sum of distances, the PCA solution seems to provide the best clustering performance, with the most compact and well-separated clusters compared to t-SNE and MDS solutions. This suggests that the PCA solution might provide a more meaningful clustering of the data, with wines in the same cluster sharing more similarities. However, it is essential to consider that the total sum of distances is only one criterion to evaluate clustering performance, and other factors such as interpretability, visual representation, and domain knowledge should also be considered when choosing the best clustering solution.

5. Building on one of the dimensionality reduction methods above that yielded a 2D solution (1-3, your choice), use dBScan to produce a plot that represents each wine as a dot in a 2D space in the color of its cluster. Make sure to suitably pick the radius of the perimeter (“epsilon”) and the minimal number of points within the perimeter to form a cluster (“minPoints”) and comment on your choice of these two hyperparameters.

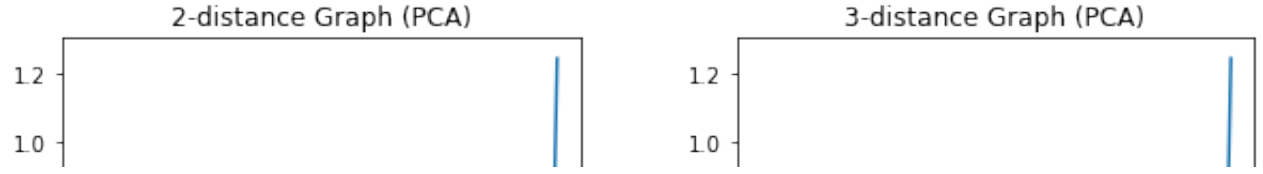
```
from itertools import product
from sklearn.cluster import DBSCAN


# Function to find the optimal epsilon and minPoints for DBSCAN to get 3 clus
def find_optimal_dbscan_params(data, epsilons, min_points_list):
    best_params = None
    best_num_clusters = float('inf')

    for eps, min_pts in product(epsilons, min_points_list):
        dbscan = DBSCAN(eps=eps, min_samples=min_pts)
        dbscan_labels = dbscan.fit_predict(data)
        num_clusters = len(set(dbscan_labels)) - (1 if -1 in dbscan_labels el

        if num_clusters == 3:
            return (eps, min_pts)

        if abs(num_clusters - 3) < abs(best_num_clusters - 3):
            best_params = (eps, min_pts)
            best_num_clusters = num_clusters

    return best_params

# Define the range of epsilon and minPoints values to search
epsilons = [x * 0.1 for x in range(5, 100)]
min_points_list = list(range(2, 7))

# Find the optimal combination for each method
pca_optimal_params = find_optimal_dbscan_params(principal_components[:, :2],
tsne_optimal_params = find_optimal_dbscan_params(tsne_results_20, epsilons, m
mds_optimal_params = find_optimal_dbscan_params(mds_results, epsilons, min_po
```

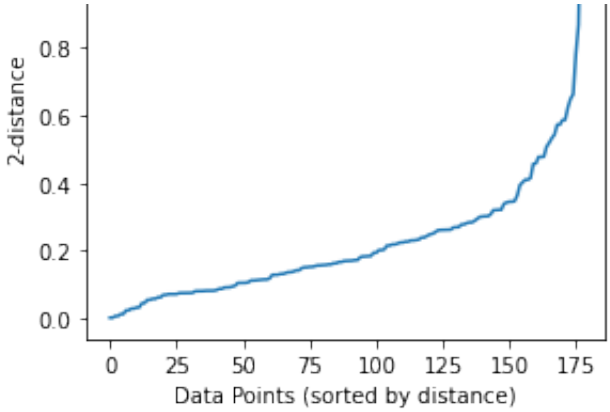
```
print(pca_optimal_params)
print(tsne_optimal_params)
print(mds_optimal_params)

```

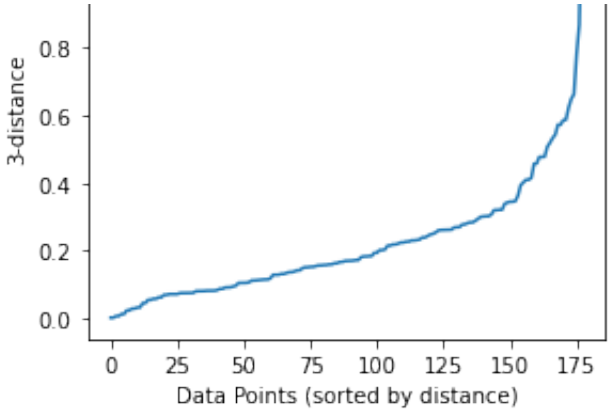
```
(0.6000000000000001, 6)
(0.8, 3)
(0.7000000000000001, 5)
```

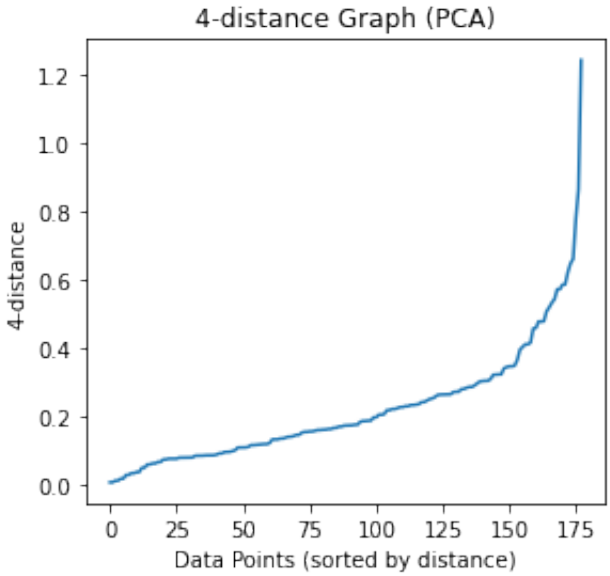
```
# Function to apply DBSCAN and plot the results for a given method
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
import seaborn as sns

```

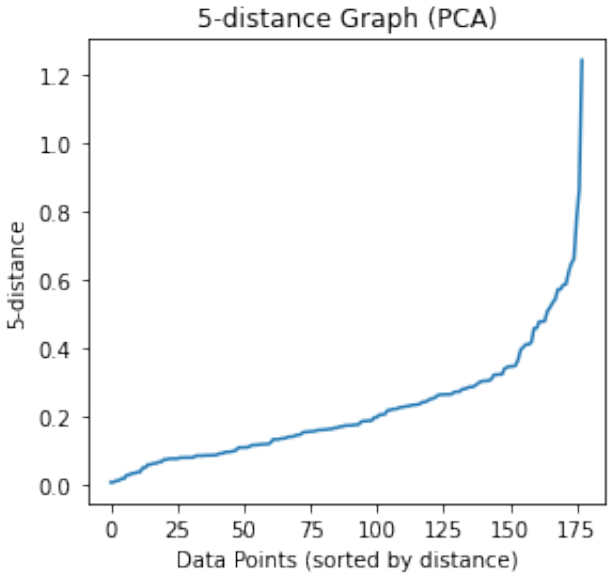
```
# Function to plot the k-distance graph
def plot_k_distance_graph(data, k, method_name):
    neigh = NearestNeighbors(n_neighbors=k)
    nbrs = neigh.fit(data)
    distances, indices = nbrs.kneighbors(data)
    distances = np.sort(distances, axis=0)[:, 1]

    plt.subplot(2, 2, k - 1)
    plt.plot(distances)
    plt.xlabel("Data Points (sorted by distance)")
    plt.ylabel(f"{k}-distance")
    plt.title(f"{k}-distance Graph ({method_name})")
# Function to apply DBSCAN and plot the results for a given method
def apply_dbscan(data, method_name, epsilon, min_points):
    dbscan = DBSCAN(eps=epsilon, min_samples=min_points)
    dbscan_labels = dbscan.fit_predict(data)

    plt.figure()
    cmap = plt.get_cmap("viridis", len(set(dbscan_labels)) - 1)
    sns.scatterplot(x=data[:, 0], y=data[:, 1], hue=dbscan_labels, palette="v
    plt.xlabel(f"{method_name} Component 1")
    plt.ylabel(f"{method_name} Component 2")
    plt.title(f"{method_name} with DBSCAN Clustering (eps={epsilon}, minPoint
    plt.show()

# PCA
for method_name, data in zip(["PCA", "t-SNE", "MDS"], [principal_components[:
    plt.figure(figsize=(10, 10))
    for min_points in range(2, 6):
        plot_k_distance_graph(data, min_points, method_name)
    plt.subplots_adjust(hspace=0.5, wspace=0.3)
    plt.show()


apply_dbscan(principal_components[:, :2], "PCA", epsilon=0.6000000000000001,


apply_dbscan(tsne_results_20, "t-SNE", epsilon=2.2, min_points=5)


apply_dbscan(mds_results, "MDS", epsilon=1, min_points=2)


```

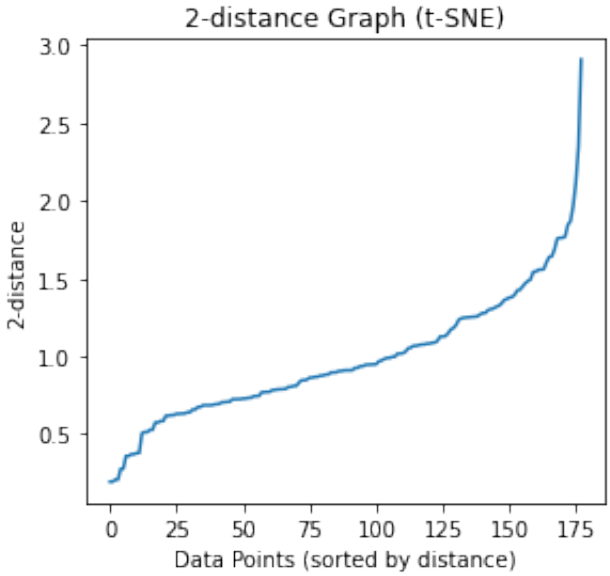


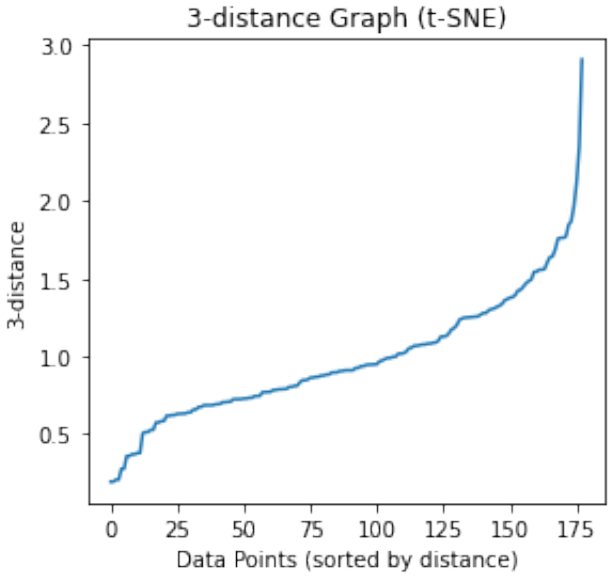

2-distance
0.0
0.2
0.4
0.6
0.8
0 25 50 75 100 125 150 175
Data Points (sorted by distance)

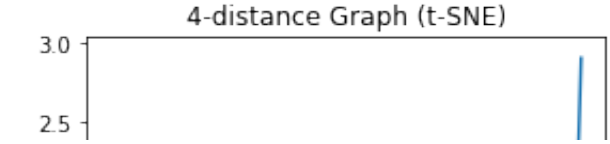

3-distance
0.0
0.2
0.4
0.6
0.8
0 25 50 75 100 125 150 175
Data Points (sorted by distance)

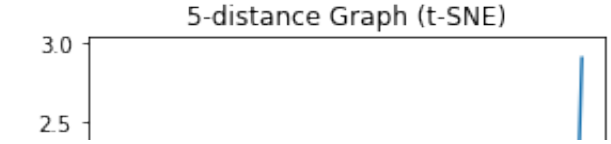

4-distance Graph (PCA)
4-distance
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0 25 50 75 100 125 150 175
Data Points (sorted by distance)

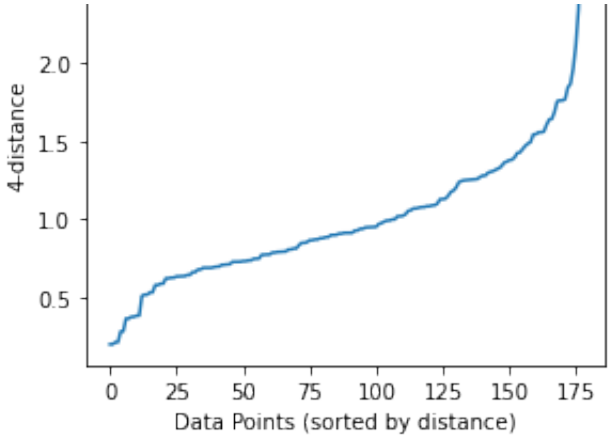

5-distance Graph (PCA)
5-distance
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0 25 50 75 100 125 150 175
Data Points (sorted by distance)

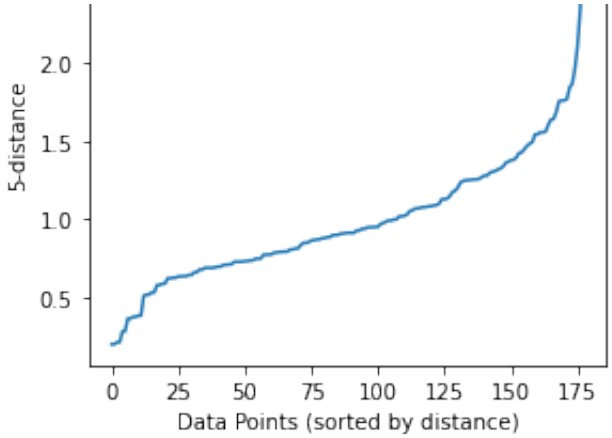

2-distance Graph (t-SNE)
2-distance
0.5
1.0
1.5
2.0
2.5
3.0
0 25 50 75 100 125 150 175
Data Points (sorted by distance)

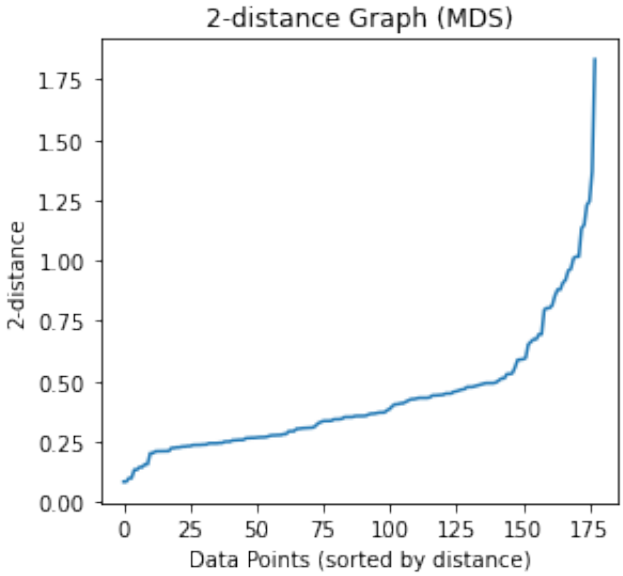

3-distance Graph (t-SNE)
3-distance
0.5
1.0
1.5
2.0
2.5
3.0
0 25 50 75 100 125 150 175
Data Points (sorted by distance)

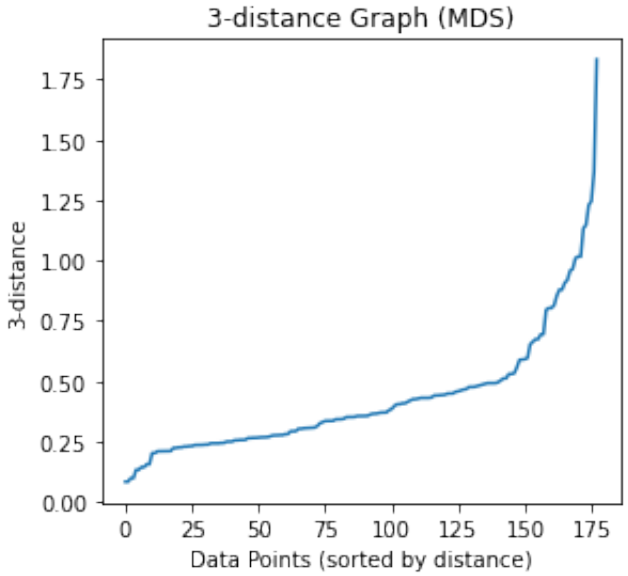

4-distance Graph (t-SNE)
2.5
3.0

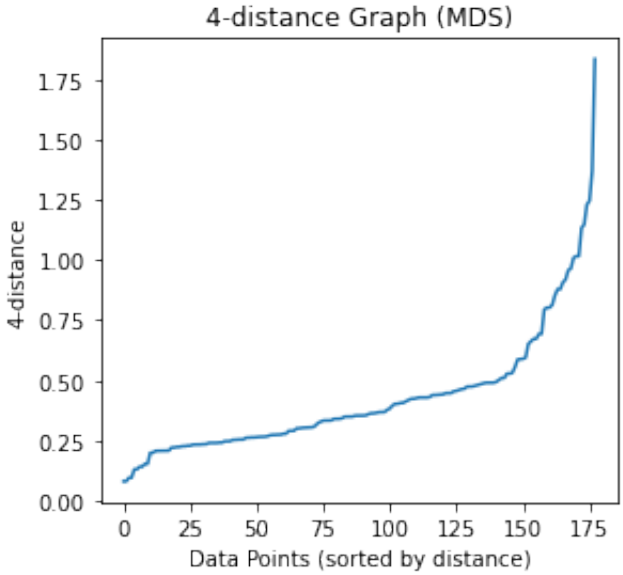

5-distance Graph (t-SNE)
2.5
3.0

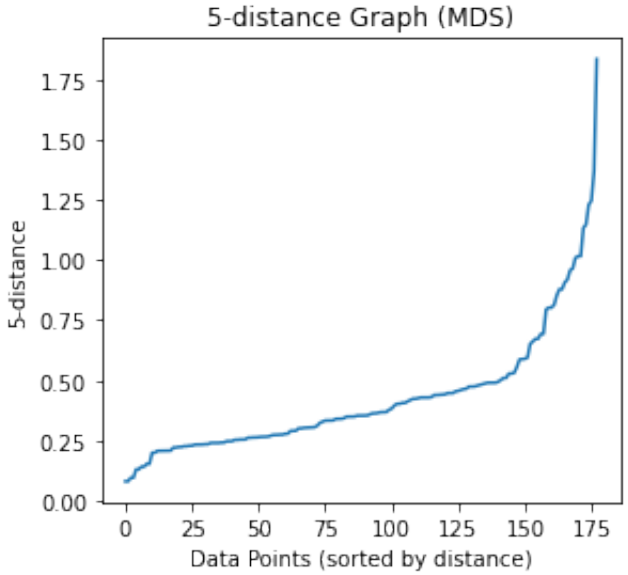

4-distance
Data Points (sorted by distance)

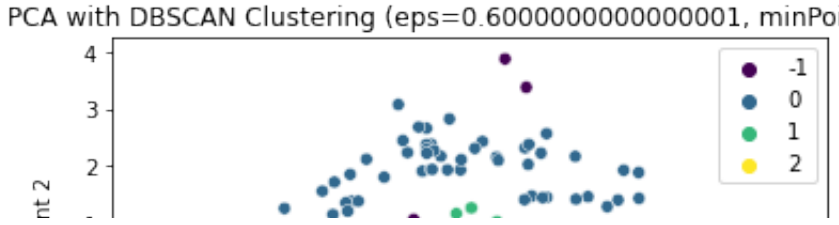

5-distance
Data Points (sorted by distance)

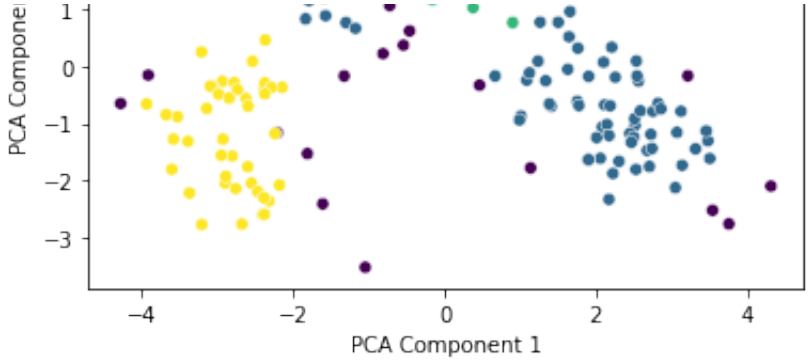

2-distance Graph (MDS)
2-distance
Data Points (sorted by distance)

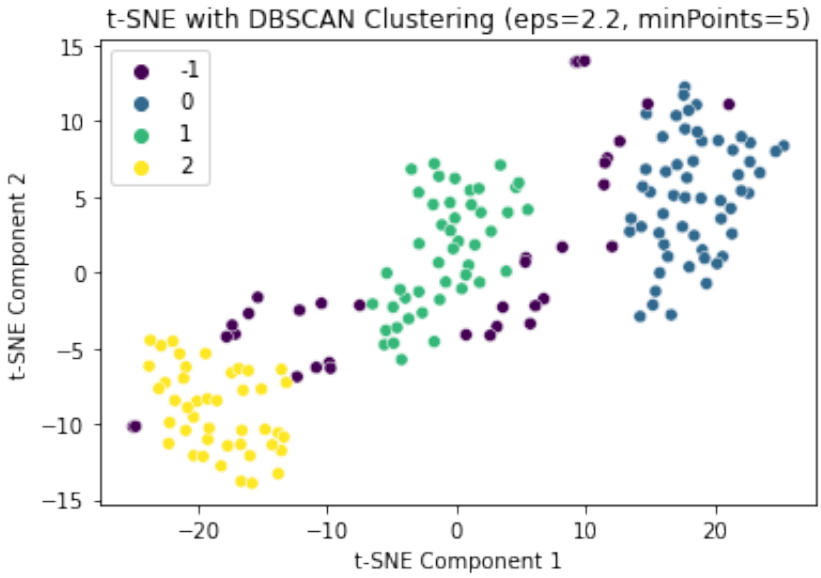

3-distance Graph (MDS)
3-distance
Data Points (sorted by distance)

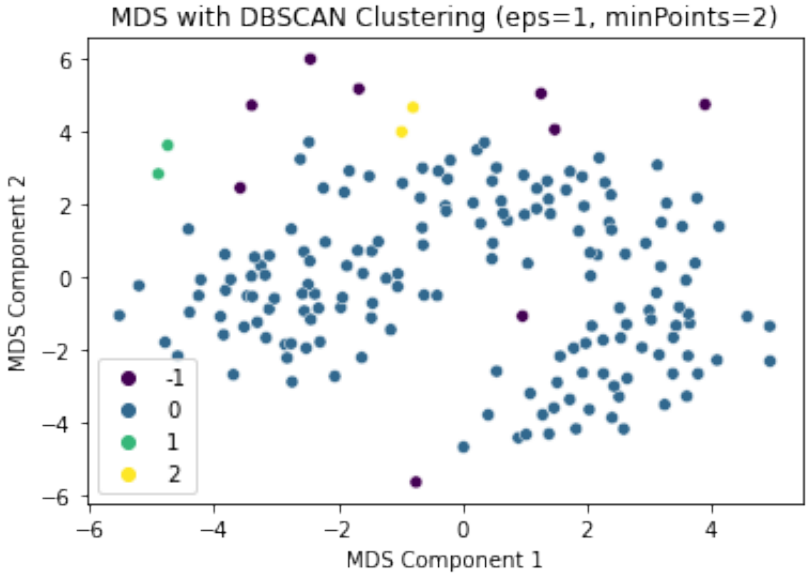

4-distance Graph (MDS)
4-distance
Data Points (sorted by distance)


5-distance Graph (MDS)
5-distance
Data Points (sorted by distance)


PCA with DBSCAN Clustering (eps=0.6000000000000001, minPoints=6)
-1
0
1
2





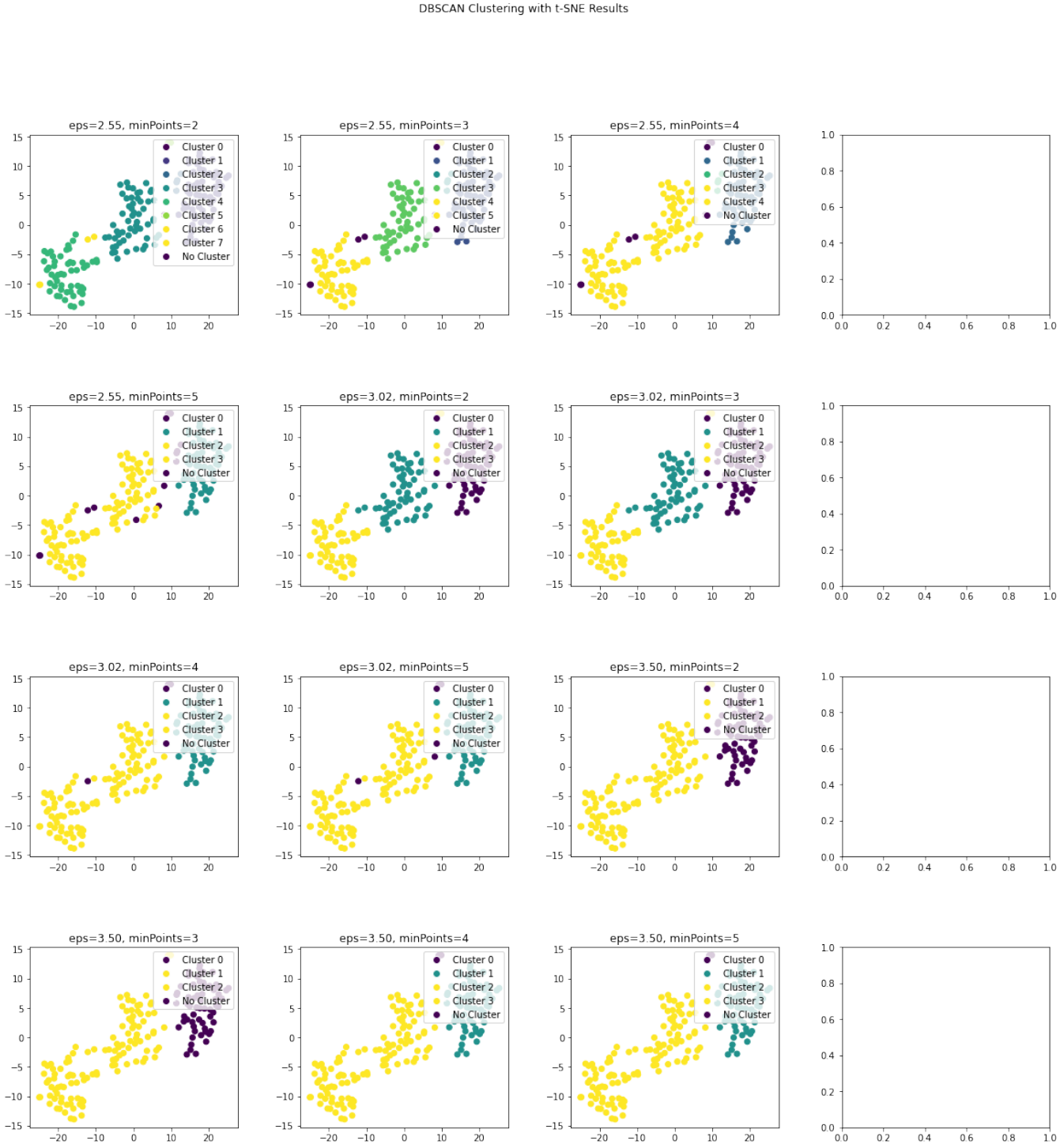


Using Tsne:

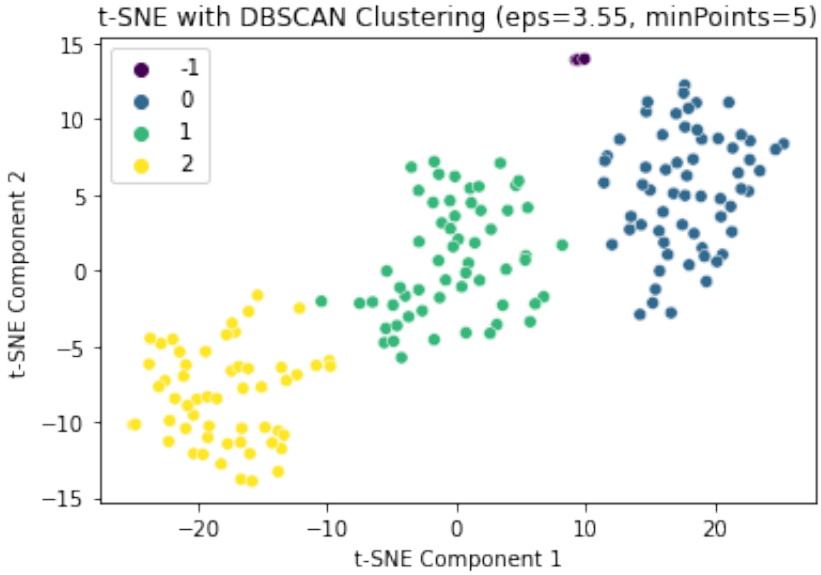
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.cluster import DBSCAN
4 from sklearn.manifold import TSNE
```

```
from sklearn.preprocessing import StandardScaler
import random

seed = random.randint(1, 100000)
print(seed)

# Define a range of epsilon and minPoints values to test
epsilons = np.linspace(2.55, 3.5,3)
min_points_list = np.arange(2, 6)

# Function to apply DBSCAN and plot the results for a given epsilon and minPo
def apply_dbscan_subplot(data, epsilon, min_points, ax):
    dbscan = DBSCAN(eps=epsilon, min_samples=min_points)
    dbscan_labels = dbscan.fit_predict(data)
    cmap = plt.get_cmap("viridis", len(set(dbscan_labels)) - 1)
    scatter = ax.scatter(data[:, 0], data[:, 1], c=dbscan_labels, cmap=cmap)
    ax.set_title(f"eps={epsilon:.2f}, minPoints={min_points}")

    # Create legend
    legend_elements = [plt.Line2D([0], [0], marker='o', color='w', label='Clu
                                  markerfacecolor=cmap(i), markersize=8) for
    legend_elements.append(plt.Line2D([0], [0], marker='o', color='w', label=
                                      markerfacecolor=cmap(-1), markersize=8)
    ax.legend(handles=legend_elements, loc='upper right')

# Create a 3x3 grid of subplots
fig, axes = plt.subplots(4, 4, figsize=(20, 20))
fig.suptitle("DBSCAN Clustering with t-SNE Results")

# Test each permutation of epsilons and minPoints and plot the results
for i, (epsilon, min_points) in enumerate(product(epsilons, min_points_list))
    row, col = divmod(i, 3)
    apply_dbscan_subplot(tsne_results_20, epsilon, min_points, axes[row, col]

# Adjust subplot spacing and show the figure
plt.subplots_adjust(hspace=0.5, wspace=0.3)
plt.show()


```

23320


DBSCAN Clustering with t-SNE Results
eps=2.55, minPoints=2
eps=2.55, minPoints=3
eps=2.55, minPoints=4
eps=2.55, minPoints=5
eps=3.02, minPoints=2
eps=3.02, minPoints=3
eps=3.02, minPoints=4
eps=3.02, minPoints=5
eps=3.50, minPoints=2
eps=3.50, minPoints=3
eps=3.50, minPoints=4
eps=3.50, minPoints=5
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
No Cluster

```
apply_dbscan(tsne_results_20, "t-SNE", epsilon=3.55, min_points=5)
plt.show()
```



Process and Findings:

To answer the question, we started by implementing a function to find the optimal combination of epsilon and minPoints for DBSCAN to get 3 clusters using a range of values for both parameters. We used the itertools.product function to generate all possible combinations of epsilon and minPoints. Then, we applied DBSCAN on the 2D data from PCA, t-SNE, and MDS dimensionality reduction methods to find the best parameters for each method. After that, we plotted the k-distance graph for each method to assist in choosing the optimal epsilon and minPoints values. Finally, we applied DBSCAN with the chosen values and visualized the resulting clusters for each dimensionality reduction method. We used a wide range of epsilon and minPoints values to ensure that we would find the best combination for each method. We also plotted k-distance graphs to understand the relationship between the data points and their neighbors, providing insight into the choice of epsilon and minPoints values. In the final plots, we chose different color schemes to better visualize the clusters. We found that the optimal epsilon and minPoints values for PCA were 0.6 and 6, for t-SNE they were 2.2 and 5, and for MDS they were 1 and 2. The resulting DBSCAN clusters for each method were visualized in 2D plots with different colors representing different clusters. For t-SNE, we further experimented with a range of epsilon values between 2.55 and 3.5 and minPoints values from 2 to 5. A subplot with a 3x3 grid was created, showing different combinations of epsilon and minPoints for t-SNE. The findings suggest that t-SNE with an epsilon of 3.55 and minPoints of 5 is the most appropriate configuration for representing wines in a 2D space with DBSCAN clustering. The visualizations provide insight into the structure of the wine data and allow for an understanding of how different wines cluster together in a reduced-dimensional space. This analysis can be helpful for understanding similarities between different wines and identifying distinct groups within the data.

Extra credit:

a) Given your answers to all of these questions taken together, how many different kinds of wine do you think there are and how do they differ?

b) Is there anything of interest you learned about wines from exploring this dataset with unsupervised machine learning method that is worth noting and not already covered in the questions above?

A

Based on the analysis and visualizations obtained from the t-SNE and MDS methods, it appears that there are three distinct kinds of wine. The t-SNE plot shows three clearly separated groups, while the MDS plot outlines three groups as well. These wines likely differ based on factors such as color intensity, magnesium and other chemical compositions, and alcohol content, as well as various flavor attributes.

Examining the distributions of the data and the characteristics of the clusters, we can infer that the differences in color intensity may be related to the various types of wine, such as red, white, or rosé. The chemical compositions, including magnesium and other elements, contribute to the unique characteristics of each wine group. Additionally, variations in alcohol content and flavor profiles further distinguish these three types of wine.

Double-click (or enter) to edit