Data Mining

Exercise-4

Name: Chethan Kashyap Bangalore Muralidhara

Date: 23/11/2023

Completed Exercises: 1,2,3,4,5,6,7(All)

Solutions:

1. All the questions were solved in python and the code file is attached.

Question 1

In [8]:
```python
import pandas as pd
from sklearn.metrics.pairwise import euclidean_distances, manhattan_distances, cosine_distances

file_path = 'D:/MS/Data Mining/3/tcpc.csv'

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Drop the "DateTime" column
df = df.drop(columns=['DateTime'])

# Select the fifth case
target_case = df.iloc[4, :].values.reshape(1, -1)

# Exclude the target case from the DataFrame
df_excluded_target = df.drop(index=4)

# Euclidean distance
euclidean_dist = euclidean_distances(target_case, df_excluded_target).flatten()

# Manhattan distance
manhattan_dist = manhattan_distances(target_case, df_excluded_target).flatten()

# Cosine distance
cosine_dist = cosine_distances(target_case, df_excluded_target).flatten()

# Find the indices of the nearest neighbors
euclidean_nn_index = euclidean_dist.argmin()
manhattan_nn_index = manhattan_dist.argmin()
cosine_nn_index = cosine_dist.argmin()

# Display the results
print("Euclidean Nearest Neighbor (Index):", euclidean_nn_index)
print("Manhattan Nearest Neighbor (Index):", manhattan_nn_index)
print("Cosine Nearest Neighbor (Index):", cosine_nn_index)
```

```
Euclidean Nearest Neighbor (Index): 32717
Manhattan Nearest Neighbor (Index): 32717
Cosine Nearest Neighbor (Index): 1009
```

2.

Question 2

In [9]:
```python
import pandas as pd
from sklearn.metrics.pairwise import euclidean_distances, manhattan_distances, cosine_distances
from sklearn.preprocessing import StandardScaler

file_path = 'D:/MS/Data Mining/3/tcpc.csv'

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Drop the "DateTime" column
df = df.drop(columns=['DateTime'])

# Select the fifth case
target_case = df.iloc[4, :].values.reshape(1, -1)

# Standardize the data for Euclidean and Manhattan distances
scaler = StandardScaler()
df_standardized = scaler.fit_transform(df)

# Euclidean distance
euclidean_dist = euclidean_distances(target_case, df_standardized).flatten()

# Manhattan distance
manhattan_dist = manhattan_distances(target_case, df_standardized).flatten()

# Cosine distance
cosine_dist = cosine_distances(target_case, df_standardized).flatten()

# Find the indices of the nearest neighbors
euclidean_nn_index = euclidean_dist.argmin()
manhattan_nn_index = manhattan_dist.argmin()
cosine_nn_index = cosine_dist.argmin()

# Display the results
print("Euclidean Nearest Neighbor (Index):", euclidean_nn_index)
print("Manhattan Nearest Neighbor (Index):", manhattan_nn_index)
print("Cosine Nearest Neighbor (Index):", cosine_nn_index)
```

```
Euclidean Nearest Neighbor (Index): 29497
Manhattan Nearest Neighbor (Index): 31656
Cosine Nearest Neighbor (Index): 22148
```

3.

Question 3

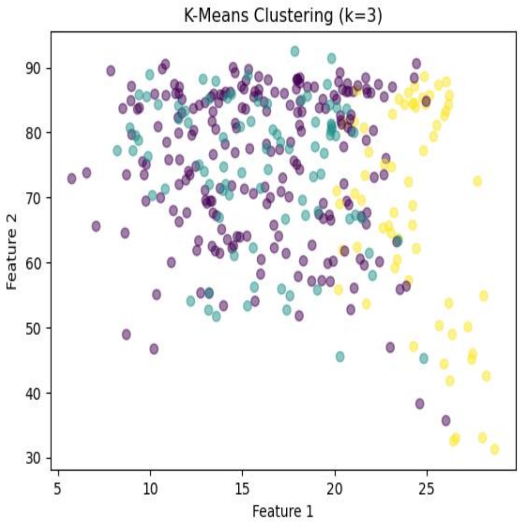
In [10]:
```
from sklearn.preprocessing import MinMaxScaler

# Select the fifth case
target_case = df.iloc[4, :].values.reshape(1, -1)

# Scale the data into the interval [0, 1]
scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df)

# Euclidean distance
euclidean_dist_scaled = euclidean_distances(target_case, df_scaled).flatten()

# Manhattan distance
manhattan_dist_scaled = manhattan_distances(target_case, df_scaled).flatten()

# Cosine distance
cosine_dist_scaled = cosine_distances(target_case, df_scaled).flatten()

# Find the indices of the nearest neighbors
euclidean_nn_index_scaled = euclidean_dist_scaled.argmin()
manhattan_nn_index_scaled = manhattan_dist_scaled.argmin()
cosine_nn_index_scaled = cosine_dist_scaled.argmin()

# Display the results
print("Scaled Euclidean Nearest Neighbor (Index):", euclidean_nn_index_scaled)
print("Scaled Manhattan Nearest Neighbor (Index):", manhattan_nn_index_scaled)
print("Scaled Cosine Nearest Neighbor (Index):", cosine_nn_index_scaled)
```

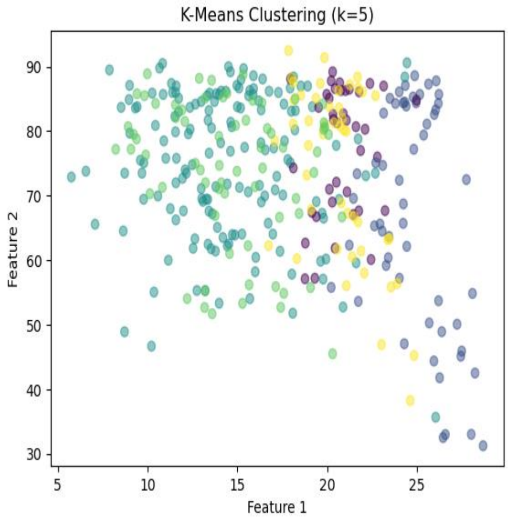
```
Scaled Euclidean Nearest Neighbor (Index): 29497
Scaled Manhattan Nearest Neighbor (Index): 33093
Scaled Cosine Nearest Neighbor (Index): 3420
```

4.

Question 4

In [13]:
```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Extract the values from the DataFrame
temperature_matrix = df.values

# Reshape the matrix to have each row represent a day and each column represent a feature
X = temperature_matrix.reshape(364, -1)

# Specify the number of clusters (k values)
k_values = [3, 5]

# Perform k-means clustering for each k value
for k in k_values:
    # Create a k-means instance
    kmeans = KMeans(n_clusters=k, random_state=42)

    # Fit the model
    kmeans.fit(X)

    # Get cluster assignments for each day
    labels = kmeans.labels_

    # Visualize the clustering results (plot the first two principal components)
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', alpha=0.5)
    plt.title(f'K-Means Clustering (k={k})')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()
```





5.

Question 5

In [26]:

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the data from "Iris.txt"
file_path = 'D:/MS/Data Mining/4/Iris.txt'
df = pd.read_csv(file_path, delimiter='\t', header=None, names=['index', 'feature1', 'feature2', 'feature3', 'feature4', 'class']

# Select the first 40 cases from each category
selected_cases = df.groupby('class').head(40)

# Extract features and labels
X_selected = selected_cases.iloc[:, 1:-1]  # Exclude the first and last columns
y_selected = selected_cases['class']

# Perform k-means clustering with k=3
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_selected)
selected_cases['cluster'] = kmeans.labels_

# Classify the remaining 10 cases using k-nearest neighbors
remaining_cases = df.groupby('class').tail(10)
X_remaining = remaining_cases.iloc[:, 1:-1]  # Exclude the first and last columns
y_remaining_true = remaining_cases['class']

# Use k-nearest neighbors for classification
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_selected, kmeans.labels_)  # Use cluster labels as target variable
y_remaining_pred = knn.predict(X_remaining)

# Display the classification results
classification_results = pd.DataFrame({'True Class': y_remaining_true, 'Predicted Class': y_remaining_pred})
print("Classification Results:")
print(classification_results)

# Calculate accuracy
accuracy = accuracy_score(y_remaining_true, y_remaining_pred)
print(f"\nAccuracy: {accuracy:.2%}")
```

```
Classification Results:
     True Class  Predicted Class
40            1                1
41            1                1
42            1                1
43            1                1
44            1                1
45            1                1
46            1                1
47            1                1
48            1                1
49            1                1
90            2                2
91            2                2
92            2                2
93            2                2
94            2                2
95            2                2
96            2                2
97            2                2
98            2                2
99            2                2
140           3                0
141           3                0
142           3                2
143           3                0
144           3                0
145           3                0
146           3                2
147           3                0
148           3                0
149           3                2

Accuracy: 66.67%
```

6.

Question 6

In [30]:
```
# Extract features and labels
X = df.iloc[:, 1:-1]  # Exclude the first and last columns
y = df['class']

# Identify highly correlated variables
correlation_matrix = X.corr()
max_corr = correlation_matrix.abs().max()
correlated_features = (correlation_matrix[correlation_matrix.isin(max_corr) & (correlation_matrix != 1)].stack().index.tolist())

# Replace highly correlated variables with their mean
for feature in correlated_features:
    feature1, feature2 = feature
    new_feature_name = f"{feature1}_{feature2}_mean"
    X[new_feature_name] = (X[feature1] + X[feature2]) / 2
    X = X.drop(columns=[feature1, feature2])

# Perform k-means clustering with k=3 on the reduced data
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Classify the remaining 10 cases using k-nearest neighbors
remaining_cases = df.groupby('class').tail(10)
X_remaining = remaining_cases.drop(columns=['index', 'class'])
y_remaining_true = remaining_cases['class']

# Use k-nearest neighbors for classification
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X, kmeans.labels_)  # Use cluster labels as target variable
y_remaining_pred = knn.predict(X_remaining)

# Display the classification results
classification_results = pd.DataFrame({'True Class': y_remaining_true, 'Predicted Class': y_remaining_pred})
print("Classification Results:")
print(classification_results)

# Calculate accuracy
accuracy = accuracy_score(y_remaining_true, y_remaining_pred)
print(f"\nAccuracy: {accuracy:.2%}")
```

```
Classification Results:
     True Class  Predicted Class
40            1                1
41            1                1
42            1                1
43            1                1
44            1                1
45            1                1
46            1                1
47            1                1
48            1                1
49            1                1
90            2                0
91            2                0
92            2                0
93            2                0
94            2                0
95            2                0
96            2                0
97            2                0
98            2                0
99            2                0
140           3                2
141           3                2
142           3                0
143           3                2
144           3                2
145           3                2
146           3                0
147           3                2
148           3                2
149           3                0

Accuracy: 33.33%
```

7.

Question 7

In [28]:
```
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.svm import SVC

# Extract features and labels
X = df.iloc[:, 1:-1]  # Exclude the first and last columns
y = df['class']

# Use Recursive Feature Elimination (RFE) with SVM to select features
svm = SVC(kernel="linear")
rfe = RFE(estimator=svm, n_features_to_select=2)
X_reduced = rfe.fit_transform(X, y)

# Perform k-means clustering with k=3 on the reduced data
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_reduced)

# Classify the remaining 10 cases using k-nearest neighbors
remaining_cases = df.groupby('class').tail(10)
X_remaining = remaining_cases.iloc[:, 1:-1]  # Exclude the first and last columns
X_remaining_reduced = rfe.transform(X_remaining)  # Transform the remaining cases to the reduced feature set
y_remaining_true = remaining_cases['class']

# Use k-nearest neighbors for classification
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_reduced, kmeans.labels_)  # Use cluster labels as target variable
y_remaining_pred = knn.predict(X_remaining_reduced)

# Display the classification results
classification_results = pd.DataFrame({'True Class': y_remaining_true, 'Predicted Class': y_remaining_pred})
print("Classification Results:")
print(classification_results)

# Calculate accuracy
accuracy = accuracy_score(y_remaining_true, y_remaining_pred)
print(f"\nAccuracy: {accuracy:.2%}")
```

```
Classification Results:
     True Class  Predicted Class
40            1                1
41            1                1
42            1                1
43            1                1
44            1                1
45            1                1
46            1                1
47            1                1
48            1                1
49            1                1
90            2                0
91            2                0
92            2                0
93            2                0
94            2                0
95            2                0
96            2                0
97            2                0
98            2                0
99            2                0
140           3                2
141           3                2
142           3                2
143           3                2
144           3                2
145           3                2
146           3                2
147           3                2
148           3                2
149           3                2

Accuracy: 33.33%
```

Another way to reduce the dimension of the Iris data is to use feature selection techniques. One popular method is to use a feature selection algorithm, such as Recursive Feature Elimination (RFE), which ranks features by their importance and recursively removes the least important features.In this example, RFE with a support vector machine is used to select the two most important features. The code then performs k-means clustering and k-nearest neighbors classification on the reduced data.