In [1]:
```
# Question 1:

# Part 1:

import numpy as np

# Create a 5x5 array with random integer values between a specified range (e.g., 0 and
random_array = np.random.randint(0, 20, size=(5, 5))

# Find the minimum and maximum values in the array
min_value = np.min(random_array)
max_value = np.max(random_array)

# Print the array and its minimum and maximum values
print("Random 5x5 Array:")
print(random_array)
print()
print("Minimum Value:", min_value)
print("Maximum Value:", max_value)
```

```
Random 5x5 Array:
[[ 2 18 17 18  5]
 [17 18  7 16 15]
 [ 5 13  3 19  9]
 [ 6  6 13  7 13]
 [ 4 18  4 16 16]]

Minimum Value: 2
Maximum Value: 19
```

In [2]:
```
# Question 1:

# Part 2:

import numpy as np

# Create a random 10x4 array with integers between 0 and 19
random_array = np.random.randint(0, 20, size=(10, 4))

# Extract the first five rows and store them in a variable
first_five_rows = random_array[:5, :]

# Print the original array and the first five rows
print("Random 10x4 Array:")
print(random_array)
print("\nFirst Five Rows:")
print(first_five_rows)
```

```
Random 10x4 Array:
[[14  9  3 12]
 [12 11 13 15]
 [ 3  4 12 19]
 [ 7 10 17 12]
 [ 3 18  0 15]
 [17  9  5 13]
 [18  3 16  3]
 [ 5 17 16 13]
 [ 6  4  6  6]
 [12  2  6  8]]

First Five Rows:
[[14  9  3 12]
 [12 11 13 15]
 [ 3  4 12 19]
 [ 7 10 17 12]
 [ 3 18  0 15]]
```

In [3]:

```
# Question 1:

# Part 3:

import numpy as np

# Create a random vector of size 10 with random integers (adjust the range as needed)
random_vector = np.random.randint(0, 20, size=10)

# Sort the vector in ascending order
sorted_vector = np.sort(random_vector)

# Print the original vector and the sorted vector
print("Random Vector:")
print(random_vector)
print()
print("Sorted Vector:")
print(sorted_vector)
```

```
Random Vector:
[18 11 12  7  2 19  9  9 16  0]

Sorted Vector:
[ 0  2  7  9  9 11 12 16 18 19]
```

In [4]:

```
# Question 1:

# Part 4:

import numpy as np

# Generate a random one-dimensional array of a specified size
array_size = 20
arr = np.random.randint(1, 20, size=array_size)  # Generates random integers between 1

# Find the most frequent value
unique_values, counts = np.unique(arr, return_counts=True)
most_frequent_value = unique_values[np.argmax(counts)]

# Print the generated array and the most frequent value
```

```
print("Generated Array:", arr)
print("Most Frequent Value:", most_frequent_value)
```

```
Generated Array: [ 3 12  7  1 15  3 15 10 10  6 10  5 13  7 10  5 16 12 11 14]
Most Frequent Value: 10
```

In [5]:

```
# Question 2:

import numpy as np
from scipy.linalg import solve

# Define the coefficients matrix (left-hand side)
coefficients = np.array([[1.02, 1.04], [1, 1]])

# Define the constants vector (right-hand side)
constants = np.array([10250, 10000])

# Solve for x and y using scipy.linalg.solve
initial_investments = solve(coefficients, constants)

# Extract the values of x and y
x = initial_investments[0]
y = initial_investments[1]

print("Bob initially invested $%.2f in the first mutual fund and $%.2f in the second m
```

```
Bob initially invested $7500.00 in the first mutual fund and $2500.00 in the second m
utual fund.
```

In the above code, we're dealing with the problem, where the initial investment is $10000 in two mutual funds (let's assume x and y.

Per the problem, after one year, return is $10,250 after +2% yield in x and +4% yield in y. So the equations we're solving for are:

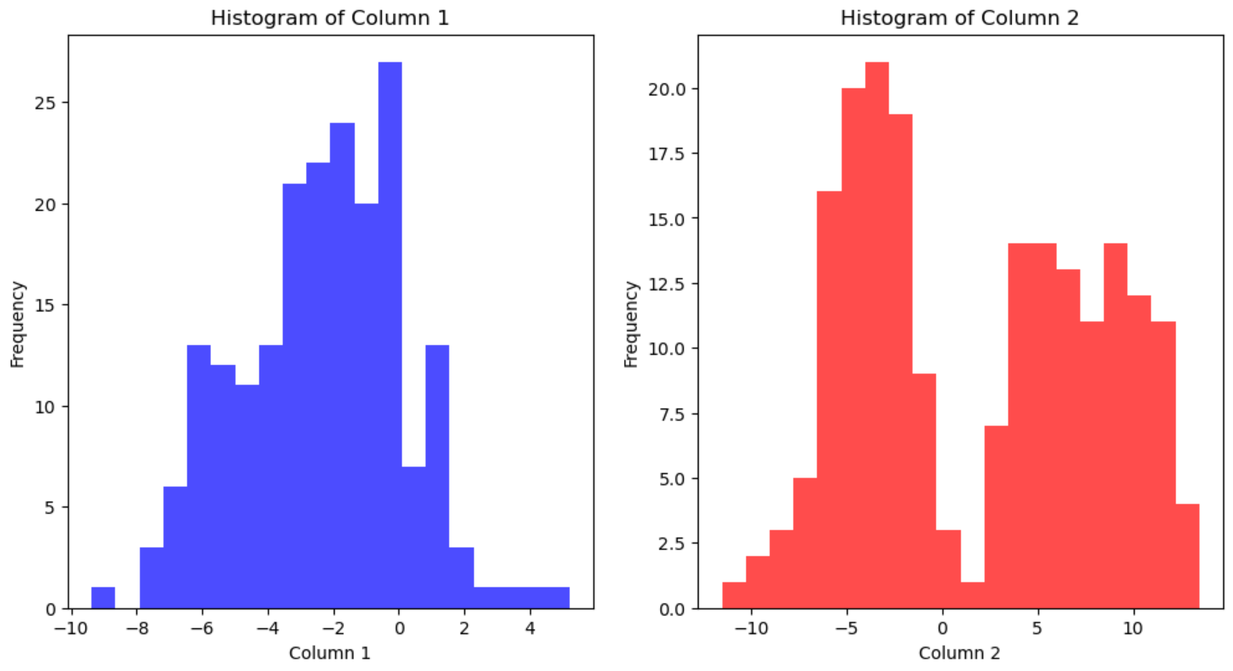
$x + y = 10000$

$1.02x + 1.04y = 10250$

In [6]:

```
# Question 3:

# Part 1: Create a histogram for the first and second column respectively

# Import necessary libraries
import os  # Operating system utilities
import pandas as pd  # Data manipulation and analysis
import numpy as np  # Numerical operations
import matplotlib.pyplot as plt  # Data visualization
from sklearn.cluster import KMeans  # K-means clustering algorithm
from scipy.cluster.vq import kmeans, vq  # Vector quantization functions
from scipy.spatial.distance import cdist  # Pairwise distance computation
import seaborn as sns  # Data visualization library
import warnings


warnings.filterwarnings('ignore')

# Set OMP_NUM_THREADS environment variable to avoid memory leak (Windows with MKL)
os.environ["OMP_NUM_THREADS"] = "1"
```

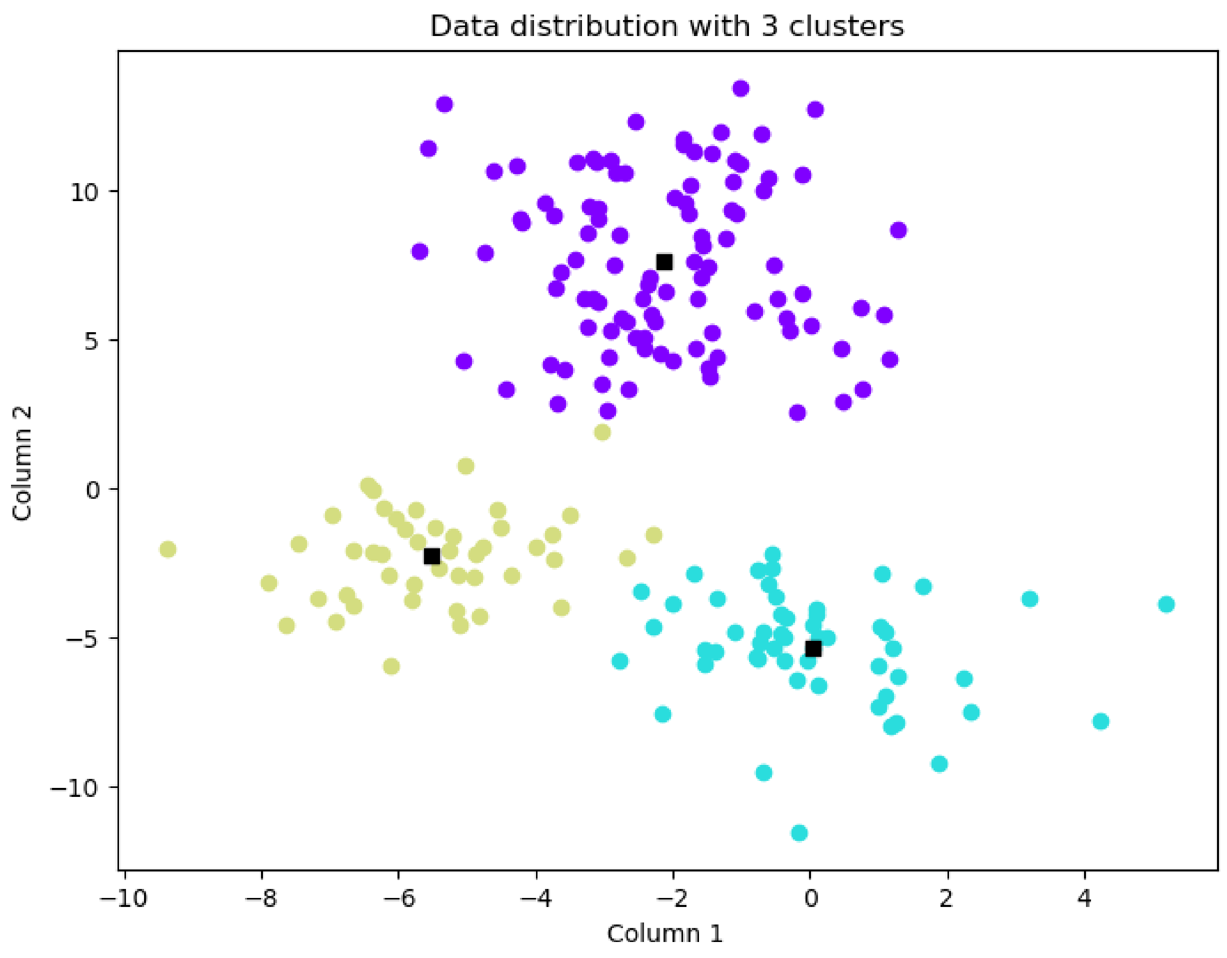
```
# Load data from the CSV file
df = pd.read_csv('Assignment_table.csv', header=None)

# Create histograms for the first and second columns
plt.figure(figsize=(12, 6))  # Create a figure with a specified size
plt.subplot(1, 2, 1)  # Create the first subplot in a 1x2 grid
plt.hist(df[0], bins=20, color='blue', alpha=0.7)  # Create a histogram for Column 1
plt.title("Histogram of Column 1")  # Set the title for the first subplot
plt.xlabel("Column 1")  # Set the x-axis label
plt.ylabel("Frequency")  # Set the y-axis label

plt.subplot(1, 2, 2)  # Create the second subplot in a 1x2 grid
plt.hist(df[1], bins=20, color='red', alpha=0.7)  # Create a histogram for Column 2
plt.title("Histogram of Column 2")  # Set the title for the second subplot
plt.xlabel("Column 2")  # Set the x-axis label
plt.ylabel("Frequency")  # Set the y-axis label

plt.show()  # Display the figure with the subplots and histograms
```

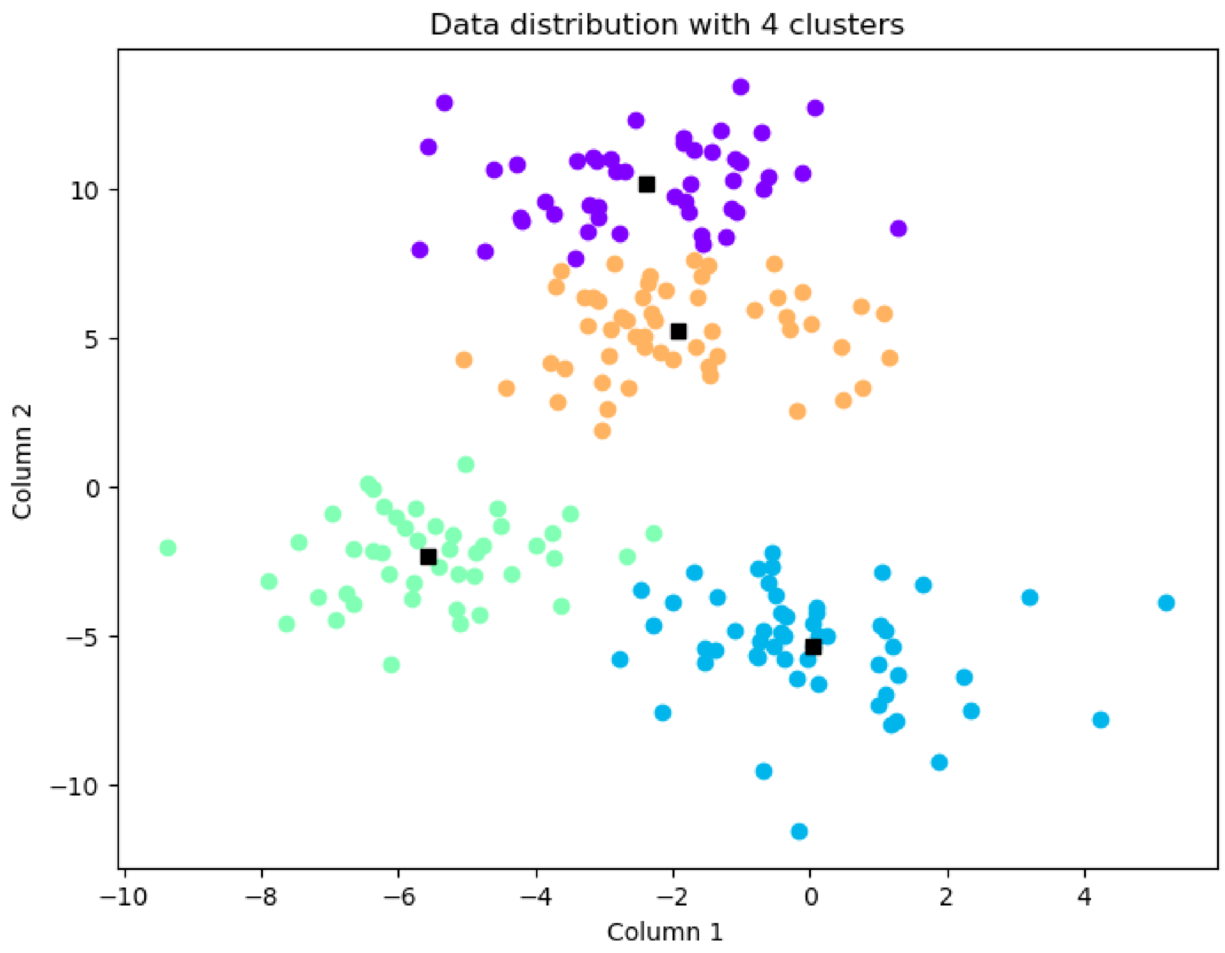


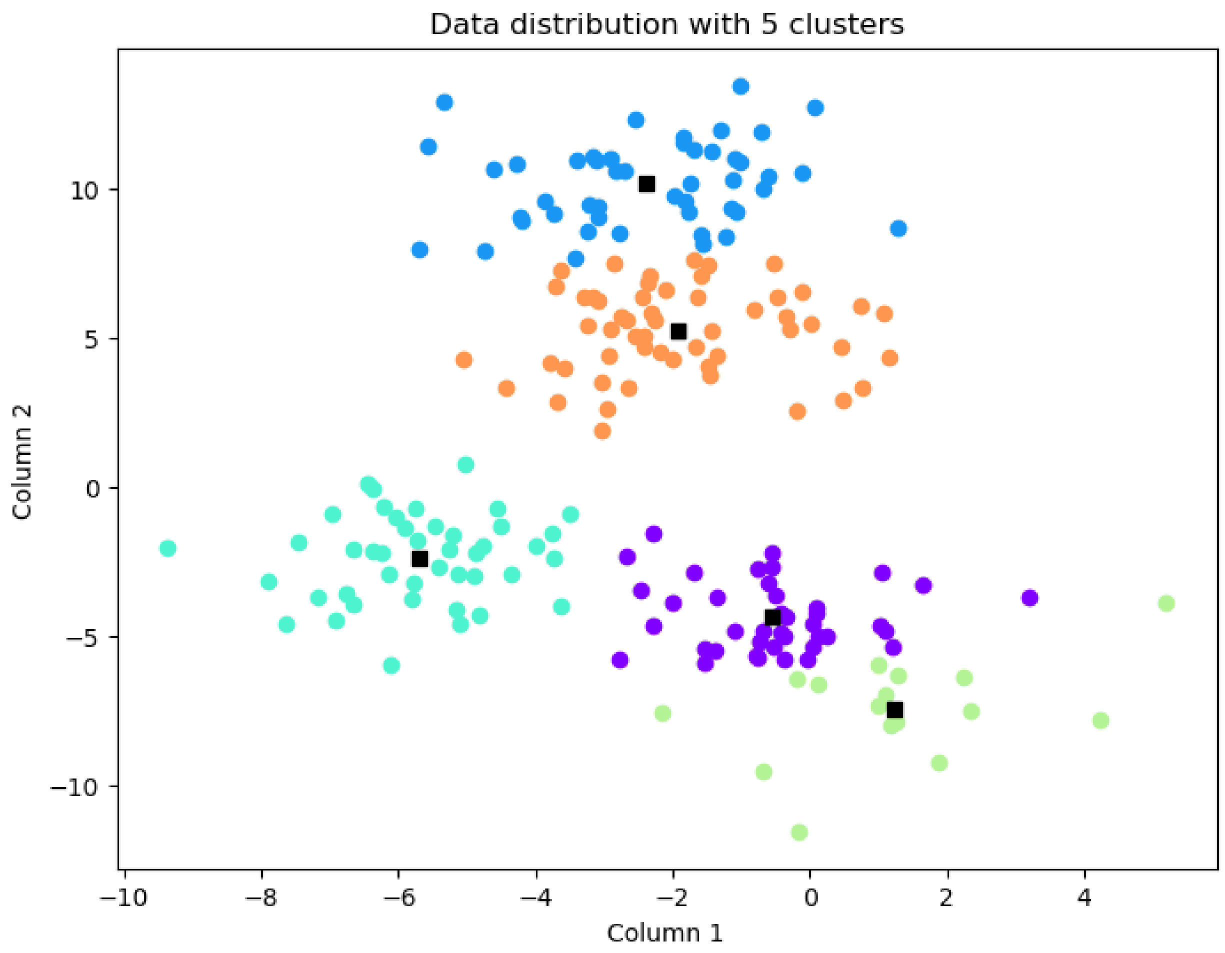
In [7]:

```
# Question 3:

# Part 2: Conduct a clustering algorithm on these two columns. Try out 3, 4, 5 cluster

# Perform K-means clustering with 3, 4, and 5 clusters
x = df.values  # Extract the values from the DataFrame
num_clusters = [3, 4, 5]  # List of cluster numbers to try

for n_clusters in num_clusters:
    # Create a KMeans clustering model with the specified number of clusters
    kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=0)

    # Fit the model and assign each data point to a cluster
    identified_clusters = kmeans.fit_predict(x)

    # Create a copy of the original DataFrame with an additional 'Clusters' column
    data_with_clusters = df.copy()
    data_with_clusters['Clusters'] = identified_clusters
```

```
# Visualize the data with circular markers for data points and square markers for
plt.figure(figsize=(8, 6))  # Create a new figure for the plot
for i in range(n_clusters):
    # Scatter plot data points of the current cluster with rainbow colors
    plt.scatter(
        data_with_clusters[data_with_clusters['Clusters'] == i][0],
        data_with_clusters[data_with_clusters['Clusters'] == i][1],
        c=[plt.cm.rainbow(i / n_clusters)],  # Use rainbow color map for data poin
        marker='o',  # Use circular markers for data points
    )

# Scatter plot cluster centroids in black with square markers
plt.scatter(
    kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1],
    c='black',  # Set marker color for centroids to black
    marker='s',  # Use square markers for centroids
)

plt.title(f'Data distribution with {n_clusters} clusters')  # Set the plot title
plt.xlabel("Column 1")  # Label the x-axis
plt.ylabel("Column 2")  # Label the y-axis
```

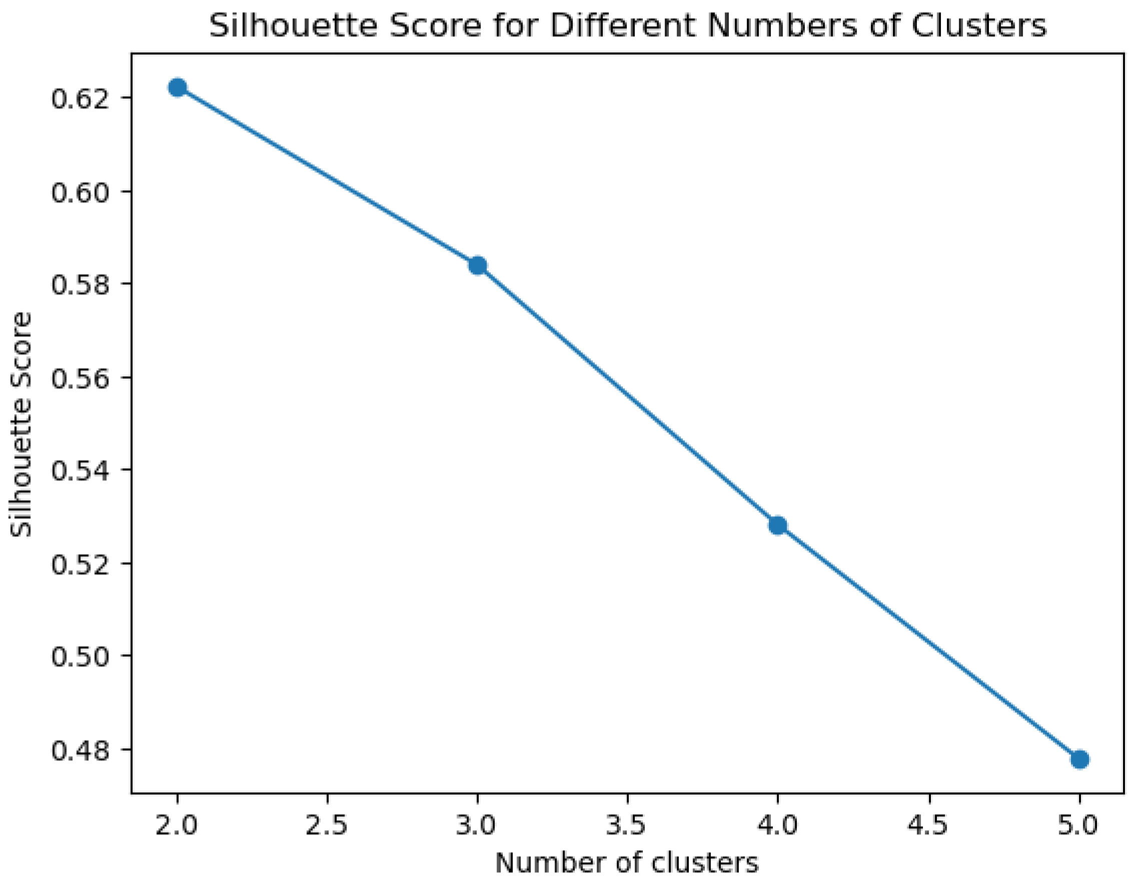



Data distribution with 4 clusters
Column 2
Column 1
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4


Data distribution with 5 clusters
Column 2
Column 1
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4

1. Question - What do you think is the right number of clusters for this dataset?
2. Answer - In order to figure out the right number of clusters, there are many methods, but one method I've chosen below is to calcucalte the the Silhouette Score, which measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation).

In [8]:
```
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Define the range of cluster numbers to consider
num_clusters = range(2, 6)  # Define a range of cluster numbers to evaluate (2 to 5 cl

silhouette_scores = []  # Initialize an empty list to store silhouette scores for each

for n_clusters in num_clusters:
    kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=0)  # Create a KMea
    cluster_labels = kmeans.fit_predict(x)  # Fit the model and get cluster labels for
    silhouette_avg = silhouette_score(x, cluster_labels)  # Calculate the silhouette s
    silhouette_scores.append(silhouette_avg)  # Append the silhouette score to the lis

# Plot the Silhouette scores
plt.plot(num_clusters, silhouette_scores, marker='o')  # Create a line plot of silhoue
plt.xlabel('Number of clusters')  # Label for the x-axis
plt.ylabel('Silhouette Score')  # Label for the y-axis
plt.title('Silhouette Score for Different Numbers of Clusters')  # Set the plot title
plt.show()  # Display the plot
```



Once we calculate the silhouette score for different numbers of clusters, we choose the one with the highest score. As a result, the right number of clusters for this dataset is 3. Sincewe were asked to do comparison between 3, 4, and 5, the answer is 3.

```
In [ ]:
```