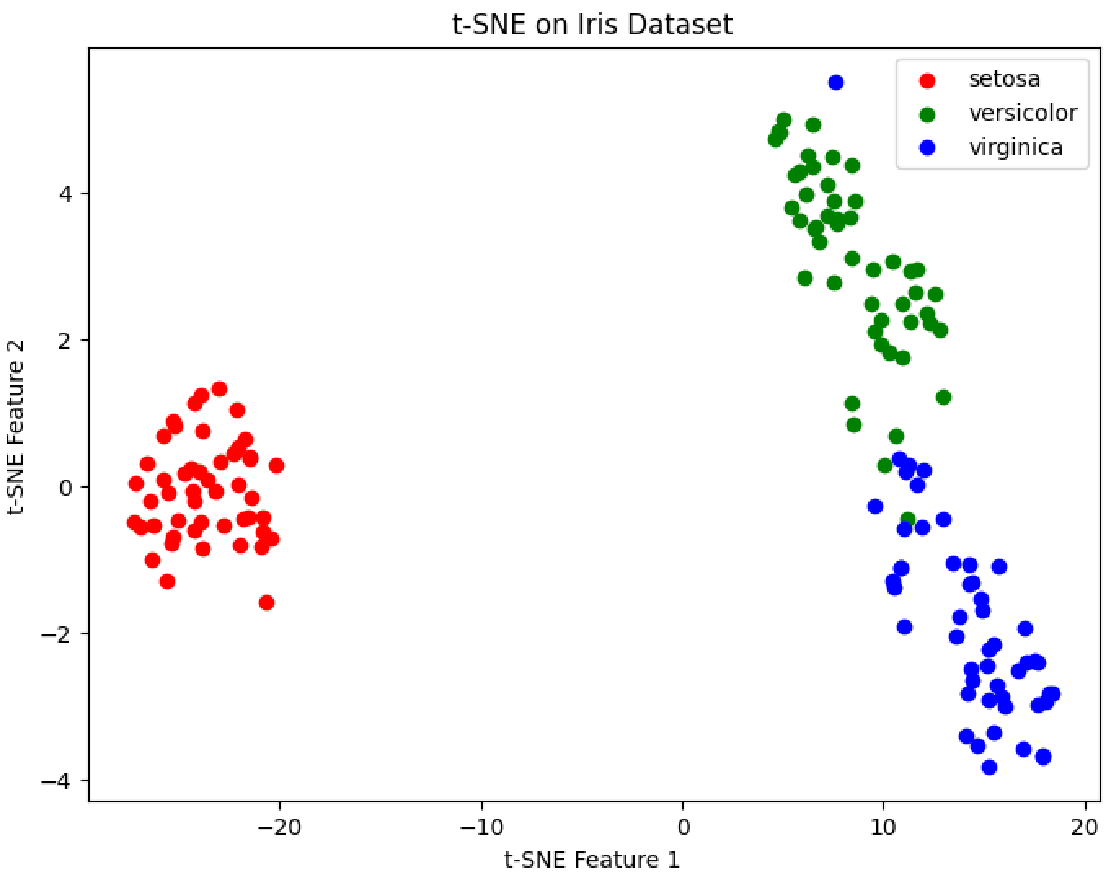
Assignment_4

April 26, 2025

0.1 1. Perform dimensionality reduction using scikit-learn's TSNE estimator on the Iris dataset, then graph the results.

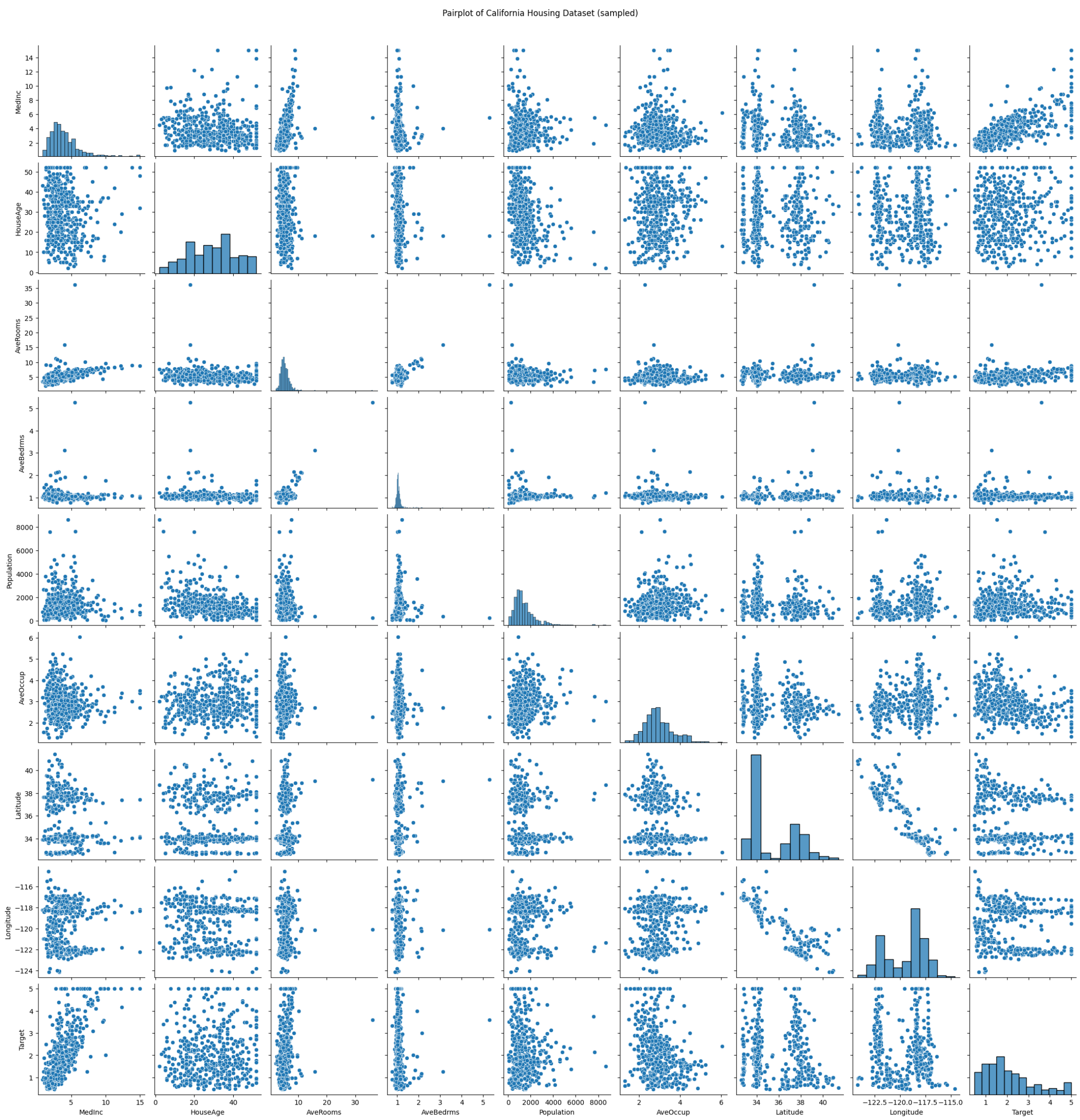
[1]:
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.manifold import TSNE

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

# Step 2: Perform t-SNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)

# Step 3: Plot the results
plt.figure(figsize=(8, 6))
for target, color, label in zip([0, 1, 2], ['r', 'g', 'b'], target_names):
    plt.scatter(X_tsne[y == target, 0], X_tsne[y == target, 1], c=color,␣
 ↪label=label)
plt.legend()
plt.title("t-SNE on Iris Dataset")
plt.xlabel("t-SNE Feature 1")
plt.ylabel("t-SNE Feature 2")
plt.show()
```



0.2 2. Create a Seaborn pairplot graph for the California Housing dataset. Try the Matplotlib features to panning and zoom in on the diagram. These are accessible via the icons in the Matplotlib window.

[2]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
import pandas as pd

# Step 1: Load the California Housing dataset
california = fetch_california_housing()
data = pd.DataFrame(california.data, columns=california.feature_names)
data['Target'] = california.target

# Step 2: Create a smaller sample (optional, because the dataset is large)
sampled_data = data.sample(500, random_state=42)  # sampling 500 points to make
 ↪it faster

# Step 3: Create a Seaborn pairplot
```

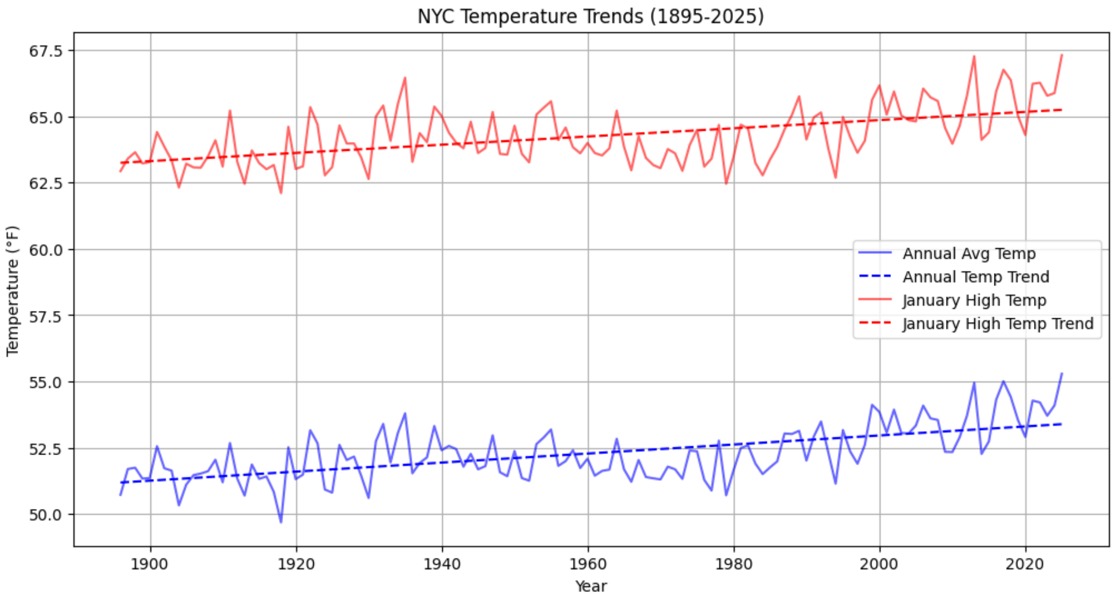
```
sns.pairplot(sampled_data)
plt.suptitle('Pairplot of California Housing Dataset (sampled)', y=1.02)

# Step 4: Show plot
plt.show()
```



3. Go to NOAA's Climate at a Glance page (Link) and download the available time series data for the average annual temperatures of New York City from 1895 to today (1895-2025). Implement simple linear regression using average annual temperature data. Also, show how does the temperature trend compare to the average January high temperatures?

[7]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import numpy as np

# Step 2a: Load the datasets (with column names)
annual_data = pd.read_csv('nyc_annual_temp.csv', skiprows=4, names=['Date',␣
↪'Value'])
january_data = pd.read_csv('nyc_january_high.csv', skiprows=4, names=['Date',␣
↪'Value'])

# Step 2b: Prepare the data
X_annual = (annual_data['Date'] // 100).values.reshape(-1, 1)  # Extract year
y_annual = annual_data['Value'].values

X_january = (january_data['Date'] // 100).values.reshape(-1, 1)
y_january = january_data['Value'].values

# Step 2c: Perform Linear Regression
model_annual = LinearRegression()
model_annual.fit(X_annual, y_annual)

model_january = LinearRegression()
model_january.fit(X_january, y_january)

# Step 2d: Predict values
y_pred_annual = model_annual.predict(X_annual)
y_pred_january = model_january.predict(X_january)

# Step 2e: Plotting
plt.figure(figsize=(12, 6))

# Annual Average Temperature
plt.plot(X_annual, y_annual, label='Annual Avg Temp', color='blue', alpha=0.6)
plt.plot(X_annual, y_pred_annual, label='Annual Temp Trend', color='blue',␣
↪linestyle='--')

# January High Temperature
plt.plot(X_january, y_january, label='January High Temp', color='red', alpha=0.
↪6)
plt.plot(X_january, y_pred_january, label='January High Temp Trend',␣
↪color='red', linestyle='--')

plt.title('NYC Temperature Trends (1895-2025)')
plt.xlabel('Year')
plt.ylabel('Temperature (°F)')
plt.legend()
```

```
plt.grid(True)
plt.show()
```



0.3 4. Load the Iris dataset from the scikit-learn library and perform classification on it with the k-nearest neighbors algorithm. Use a KNeighborsClassifier with the default k value. What is the prediction accuracy?

[8]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 2: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Step 3: Initialize KNN classifier (default k=5)
knn = KNeighborsClassifier()

# Step 4: Train the model
knn.fit(X_train, y_train)
```

```python
# Step 5: Make predictions
y_pred = knn.predict(X_test)

# Step 6: Calculate prediction accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Prediction accuracy: {accuracy:.2f}")
```

```
Prediction accuracy: 1.00
```

0.4 5. You are given a dataset of 2D points with their corresponding class labels. The dataset is as follows. A new point P with coordinates (3.0,3.0) needs to be classified using the KNN algorithm. Use the Euclidean distance to calculate the distance between points

Point ID	x	y	Class
A	2.0	3.0	0
B	1.0	1.0	0
C	4.0	4.0	1
D	5.0	2.0	1

[9]:

```python
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

# Step 1: Create the dataset
X = np.array([
    [2.0, 3.0],  # A
    [1.0, 1.0],  # B
    [4.0, 4.0],  # C
    [5.0, 2.0]   # D
])

y = np.array([0, 0, 1, 1])  # Classes

# Step 2: New point to classify
P = np.array([[3.0, 3.0]])

# Step 3: Create KNN model
# First, for k=1
knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(X, y)
prediction_k1 = knn1.predict(P)
print(f"Prediction with k=1: Class {prediction_k1[0]}")

# Now for k=3
knn3 = KNeighborsClassifier(n_neighbors=3)
knn3.fit(X, y)
```

```
prediction_k3 = knn3.predict(P)
print(f"Prediction with k=3: Class {prediction_k3[0]}")
```

```
Prediction with k=1: Class 0
Prediction with k=3: Class 1
```

0.5 6. A teacher wants to classify students as "Pass" or "Fail" based on their performance in three exams. The dataset includes three features: A new student has the following scores:

- Exam 1 Score: 72
- Exam 2 Score: 78

Exam 1 Score	Exam 2 Score	Exam 3 Score	Class (Pass/Fail)
85	90	88	Pass
70	75	80	Pass
60	65	70	Fail
50	55	58	Fail
95	92	96	Pass
45	50	48	Fail

- Exam 3 Score: 75
  Classify this student using the K-Nearest Neighbors (KNN) algorithm with k = 3.

[10]:
```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

# Step 1: Create the dataset
X = np.array([
    [85, 90, 70],
    [60, 50, 95],
    [45, 75, 65],
    [55, 92, 50],
    [88, 80, 70],
    [58, 96, 48]
])

y = np.array(['Pass', 'Pass', 'Fail', 'Fail', 'Pass', 'Fail'])  # Classes

# Step 2: New student data
new_student = np.array([[72, 78, 75]])

# Step 3: Create and train KNN model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X, y)

# Step 4: Predict the class
prediction = knn.predict(new_student)

print(f"The new student is predicted to: {prediction[0]}")
```

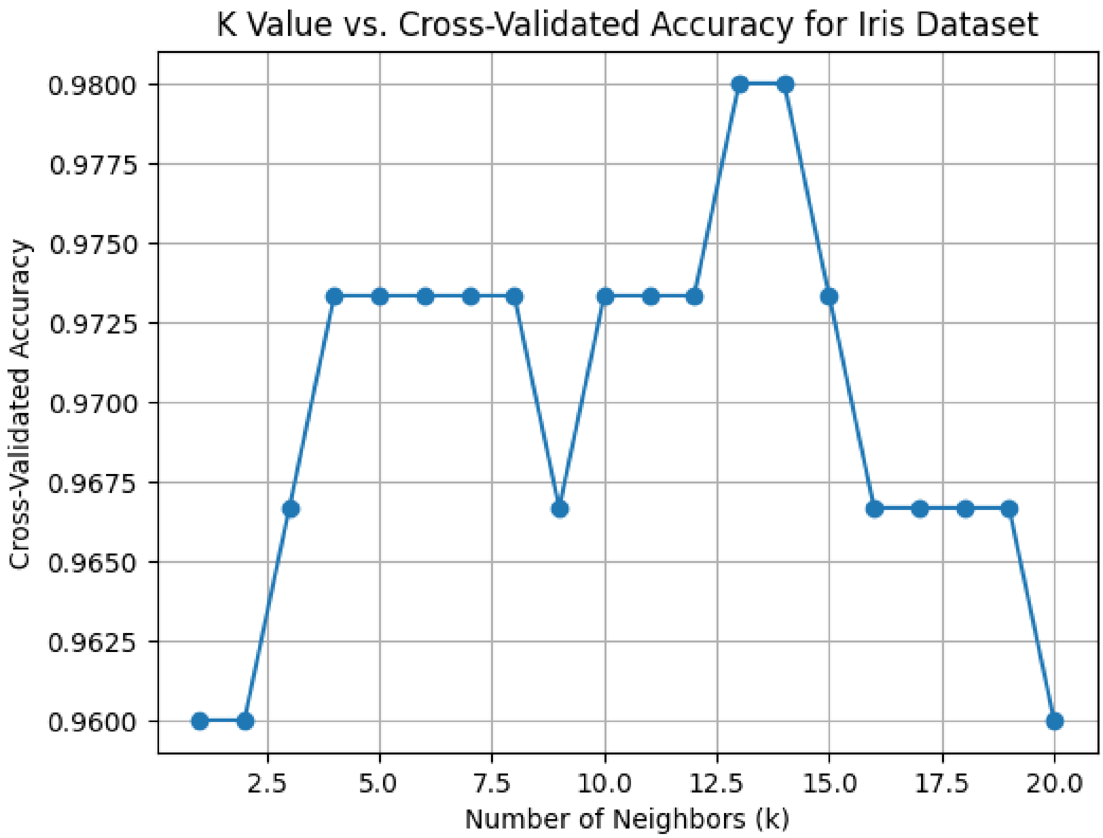
```
The new student is predicted to: Pass
```

0.6 7. Using scikit-learn's KFold class and the cross val score function, determine the optimal value for k to classify the Iris dataset using a KNeighborsClassifier.

[11]:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score, KFold
from sklearn.neighbors import KNeighborsClassifier

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 2: Set up KFold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)  # 5-fold cross-validation

# Step 3: Try different k values
k_values = range(1, 21)  # k from 1 to 20
cv_scores = []  # to store the average accuracy for each k

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=kf, scoring='accuracy')
    cv_scores.append(scores.mean())

# Step 4: Find the best k
optimal_k = k_values[np.argmax(cv_scores)]
print(f"The optimal value of k is: {optimal_k}")

# Step 5: Plot k vs accuracy
plt.plot(k_values, cv_scores, marker='o')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Cross-Validated Accuracy')
plt.title('K Value vs. Cross-Validated Accuracy for Iris Dataset')
plt.grid(True)
plt.show()
```

```
The optimal value of k is: 13
```

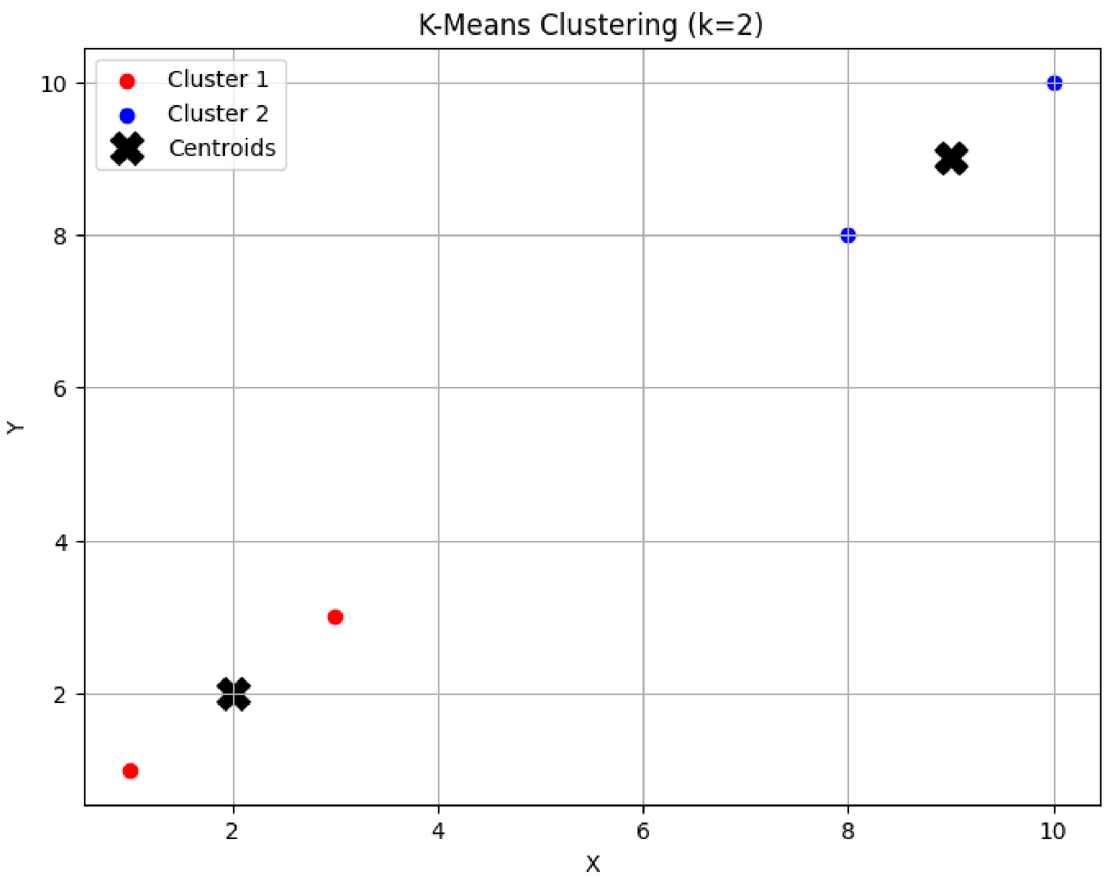


0.7 8. Write a Python script to perform K-Means clustering on the following dataset: Dataset:

{(1,1),(2,2),(3,3),(8,8),(9,9),(10,10)} Use k=2 and visualize the clusters.

[12]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Step 1: Create the dataset
X = np.array([
    [1, 1],
    [2, 2],
    [3, 3],
    [8, 8],
    [9, 9],
    [10, 10]
])

# Step 2: Create and fit the KMeans model
```

```python
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X)

# Step 3: Get the cluster labels
labels = kmeans.labels_

# Step 4: Get the cluster centers
centroids = kmeans.cluster_centers_

# Step 5: Visualize the clusters
plt.figure(figsize=(8, 6))
colors = ['red', 'blue']

for i in range(2):
    cluster_points = X[labels == i]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], c=colors[i],␣
 ↪label=f'Cluster {i+1}')

# Plot centroids
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, marker='X', c='black',␣
 ↪label='Centroids')

plt.title('K-Means Clustering (k=2)')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()
```

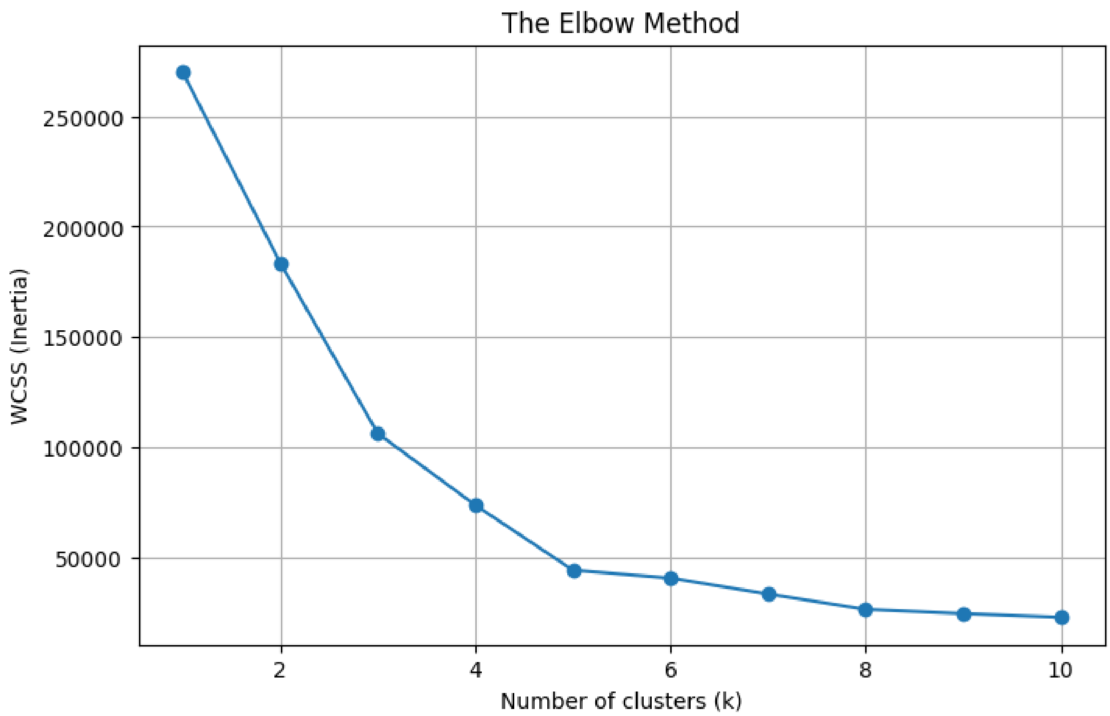


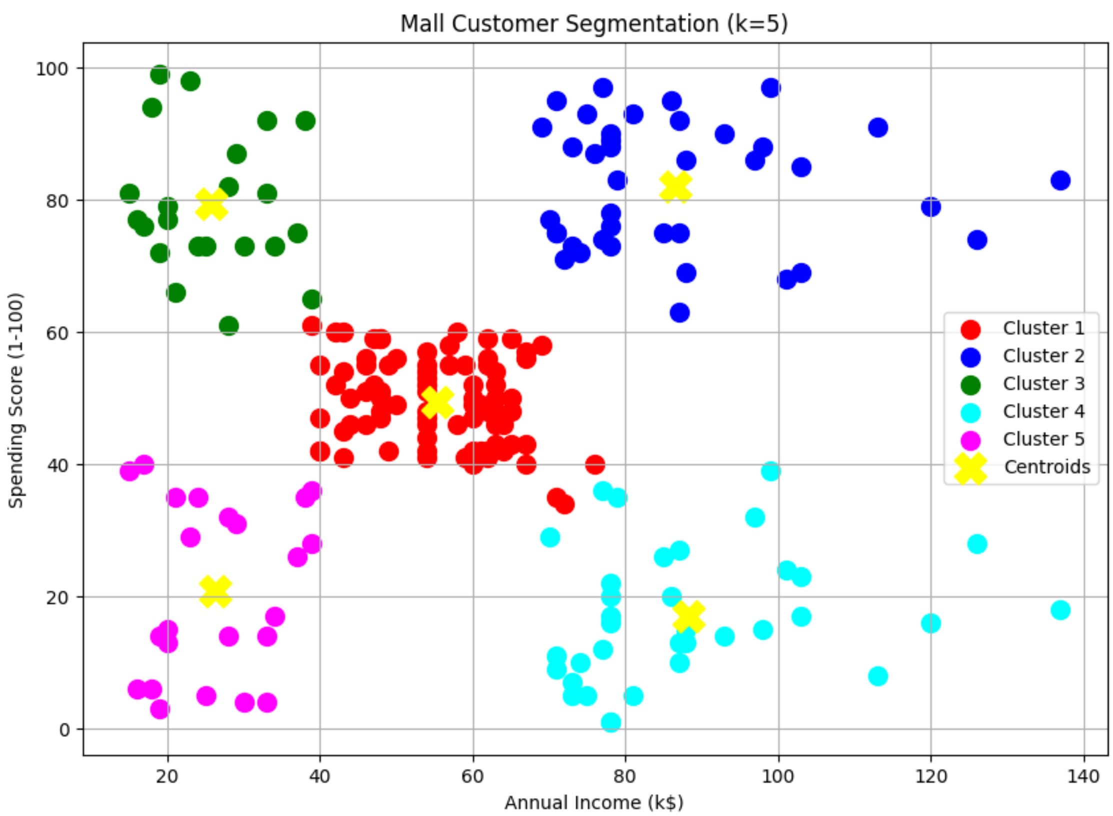
0.8 9. Write a Python script to perform K-Means clustering on the following dataset: Mall Customer Segmentation. Use k = 5 (also, determine optimal k via the Elbow Method) and visualize the clusters to identify customer segments.

Expected Output: * Scatter plot showing clusters (e.g., "High Income-Low Spenders," "Moderate Income-Moderate Spenders"). * Insights for targeted marketing strategies.

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

# Step 1: Load the dataset
# You can adjust the filename if you have it differently
mall_data = pd.read_csv('Mall_Customers.csv')

# Step 2: Select features for clustering
# Typically "Annual Income" and "Spending Score"
```

```python
X = mall_data[['Annual Income (k$)', 'Spending Score (1-100)']].values

# Step 3: Find the optimal number of clusters using the Elbow Method
wcss = []  # Within-cluster sum of squares
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters (k)')
plt.ylabel('WCSS (Inertia)')
plt.grid(True)
plt.show()

# Step 4: From Elbow graph, assume k=5 is good and perform clustering
kmeans = KMeans(n_clusters=5, random_state=42)
y_kmeans = kmeans.fit_predict(X)

# Step 5: Visualize the clusters
plt.figure(figsize=(10, 7))

colors = ['red', 'blue', 'green', 'cyan', 'magenta']
for i in range(5):
    plt.scatter(X[y_kmeans == i, 0], X[y_kmeans == i, 1],
                s=100, c=colors[i], label=f'Cluster {i+1}')

# Plot the centroids
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=300, c='yellow', marker='X', label='Centroids')

plt.title('Mall Customer Segmentation (k=5)')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.grid(True)
plt.show()
```


The Elbow Method
WCSS (Inertia)
250000
200000
150000
100000
50000
2
4
6
8
10
Number of clusters (k)


Mall Customer Segmentation (k=5)
Spending Score (1-100)
100
80
60
40
20
0
20
40
60
80
100
120
140
Annual Income (k$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids

0.9 10. Perform the following tasks using the pandas Series object:

- Create a Series from the list [7, 11, 13, 17].
- Create a Series with five elements where each element is 100.0.
- Create a Series with 20 elements that are all random numbers in the range 0 to 100. Use the describe method to produce the Series' basic descriptive statistics.
- Create a Series called temperatures with the following floating-point values: 98.6, 98.9, 100.2, and 97.9. Use the index keyword argument to specify the custom indices 'Julie', 'Charlie', 'Sam', and 'Andrea'.
- Form a dictionary from the names and values in Part (d), then use it to initialize a Series.

0.9.1 a)

[15]:
```
import pandas as pd

# (a)
series_a = pd.Series([7, 11, 13, 17])
series_a
```

[15]:
```
0     7
1    11
2    13
3    17
dtype: int64
```

0.9.2 b)

[16]:
```
# (b)
series_b = pd.Series([100.0] * 5)
series_b
```

[16]:
```
0    100.0
1    100.0
2    100.0
3    100.0
4    100.0
dtype: float64
```

0.9.3 c)

[17]:
```
import numpy as np

# (c)
series_c = pd.Series(np.random.randint(0, 101, size=20))
print("\nSeries (c):\n", series_c)
```

```
# Descriptive statistics
print("\nDescriptive Statistics:\n", series_c.describe())
```

```
Series (c):
 0     99
1     27
2     74
3      0
4     20
5     94
6     40
7     86
8     31
9     12
10    55
11    53
12    84
13    41
14    42
15    35
16    67
17    56
18    16
19    51
dtype: int32

Descriptive Statistics:
 count    20.000000
mean     49.150000
std      28.163573
min       0.000000
25%      30.000000
50%      46.500000
75%      68.750000
max      99.000000
dtype: float64
```

0.9.4 d)

[18]:
```
# (d)
temperatures = pd.Series(
    [98.6, 98.9, 100.2, 97.9],
    index=['Julie', 'Charlie', 'Sam', 'Andrea']
)
print("\nSeries (d) - Temperatures:\n", temperatures)
```

```
Series (d) - Temperatures:
 Julie       98.6
Charlie     98.9
Sam        100.2
Andrea      97.9
dtype: float64
```

0.9.5 e)

[19]:
```
# (e)
temp_dict = {
    'Julie': 98.6,
    'Charlie': 98.9,
    'Sam': 100.2,
    'Andrea': 97.9
}

series_e = pd.Series(temp_dict)
print("\nSeries (e) - From Dictionary:\n", series_e)
```

```
Series (e) - From Dictionary:
 Julie       98.6
Charlie     98.9
Sam        100.2
Andrea      97.9
dtype: float64
```

[ ]:
```
```