In [1]:
```
# Step 0: Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score
```

In [2]:
```
# Step 1: Load dataset
df = pd.read_csv("Mall_Customers.csv")
df.head()
```

Out[2]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [3]:
```
# Quick structure checks
print(df.shape)
print(df.info())
print(df.isnull().sum())
df.describe(include='all')
```

```
(200, 5)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
None
CustomerID                0
Genre                     0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

Out[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200	200.000000	200.000000	200.000000
unique	NaN	2	NaN	NaN	NaN
top	NaN	Female	NaN	NaN	NaN
freq	NaN	112	NaN	NaN	NaN
mean	100.500000	NaN	38.850000	60.560000	50.200000
std	57.879185	NaN	13.969007	26.264721	25.823522
min	1.000000	NaN	18.000000	15.000000	1.000000
25%	50.750000	NaN	28.750000	41.500000	34.750000
50%	100.500000	NaN	36.000000	61.500000	50.000000
75%	150.250000	NaN	49.000000	78.000000	73.000000
max	200.000000	NaN	70.000000	137.000000	99.000000

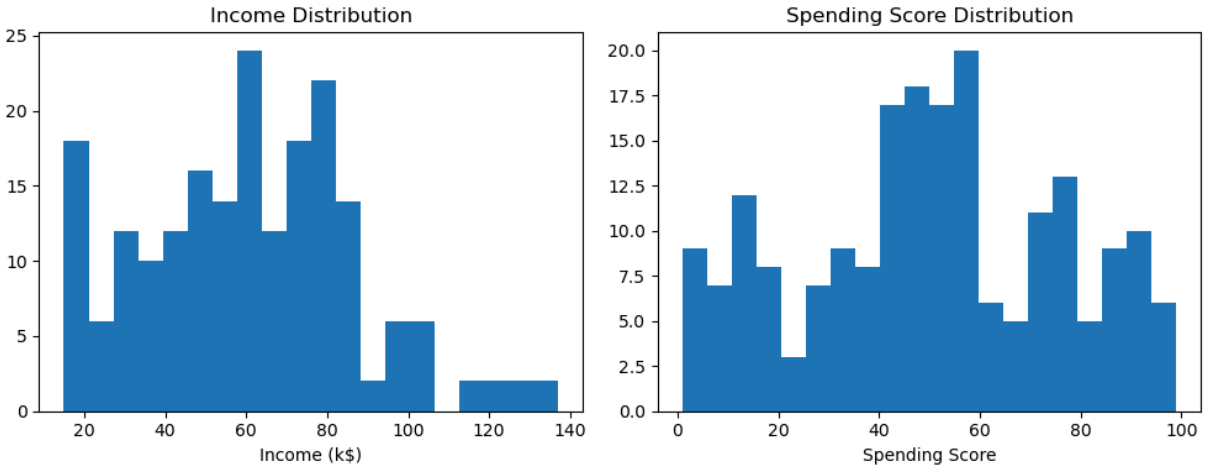
In [4]:
```
#  Rename the two features we will cluster on
rename_map = {
    'Annual Income (k$)': 'Income',
    'Spending Score (1-100)': 'SpendingScore'
}
df = df.rename(columns=rename_map)

# Keep a safe copy of original
raw_df = df.copy()

# Optional: drop duplicates if any
df = df.drop_duplicates()

# Sanity check for required columns
assert {'Income','SpendingScore'}.issubset(df.columns), "Missing required columns."
df[['Income','SpendingScore']].head()
```

Out[4]:

	Income	SpendingScore
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40

In [6]:
```
#Basic exploration plots
plt.figure(figsize=(5,4))
plt.scatter(df['Income'], df['SpendingScore'])
plt.xlabel("Annual Income (k$)")
```

```
plt.ylabel("Spending Score (1-100)")
plt.title("Income vs Spending Score")
plt.show()

# Histograms
fig, axes = plt.subplots(1, 2, figsize=(10,4))
axes[0].hist(df['Income'], bins=20)
axes[0].set_title("Income Distribution")
axes[0].set_xlabel("Income (k$)")

axes[1].hist(df['SpendingScore'], bins=20)
axes[1].set_title("Spending Score Distribution")
axes[1].set_xlabel("Spending Score")
plt.tight_layout()
plt.show()
```



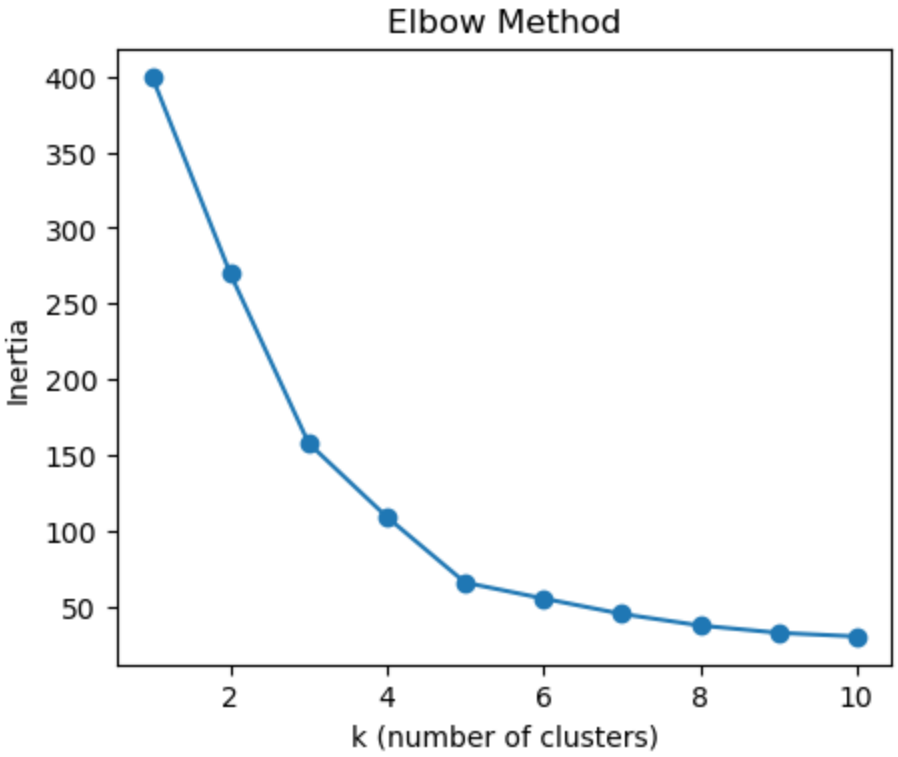
In [7]:
```
# Step 4: Select features and scale
X = df[['Income','SpendingScore']].values
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_scaled[:5]
```

Out[7]:
```
array([[-1.73899919, -0.43480148],
       [-1.73899919,  1.19570407],
       [-1.70082976, -1.71591298],
       [-1.70082976,  1.04041783],
       [-1.66266033, -0.39597992]])
```

In [8]:
```
# Step 5A: Elbow (inertia) method
inertias = []
K = range(1, 11)
for k in K:
    km = KMeans(n_clusters=k, n_init=10, random_state=42)
    km.fit(X_scaled)
    inertias.append(km.inertia_)

plt.figure(figsize=(5,4))
plt.plot(list(K), inertias, marker='o')
plt.xlabel("k (number of clusters)")
plt.ylabel("Inertia")
plt.title("Elbow Method")
plt.show()
```

```
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```



In [9]:
```
# Step 5B: Silhouette scores (k >= 2)
sil_scores = {}
for k in range(2, 11):
    km = KMeans(n_clusters=k, n_init=10, random_state=42)
    labels = km.fit_predict(X_scaled)
    sil = silhouette_score(X_scaled, labels)
    sil_scores[k] = sil

sil_scores
```

```
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```

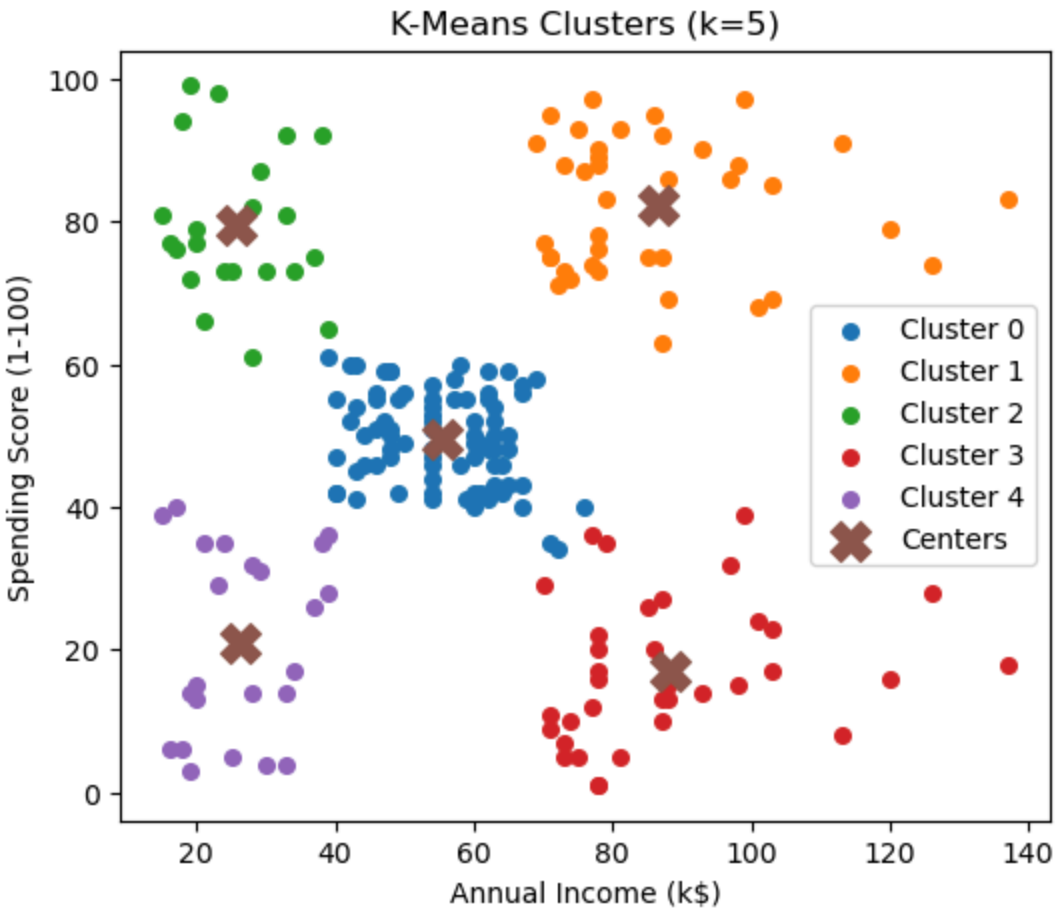
```
Out[9]: {2: np.float64(0.3212707813918878),
         3: np.float64(0.46658474419000145),
         4: np.float64(0.4939069237513199),
         5: np.float64(0.5546571631111091),
         6: np.float64(0.5398800926790663),
         7: np.float64(0.5281492781108291),
         8: np.float64(0.4552147906587443),
         9: np.float64(0.4570853966942764),
         10: np.float64(0.4431713026508046)}
```

In [10]:
```
# Step 6: Pick k based on your plots; default to 5 if unsure
k_opt = 5  # change if your elbow/silhouette suggests otherwise

kmeans = KMeans(n_clusters=k_opt, n_init=10, random_state=42)
cluster_labels = kmeans.fit_predict(X_scaled)

# Add labels back to dataframe
df_km = df.copy()
df_km['Cluster'] = cluster_labels

# Cluster centers (in original units)
centers_scaled = kmeans.cluster_centers_
centers_original = scaler.inverse_transform(centers_scaled)
centers_original
```

```
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[10]:
```
array([[55.2962963 , 49.51851852],
       [86.53846154, 82.12820513],
       [25.72727273, 79.36363636],
       [88.2       , 17.11428571],
       [26.30434783, 20.91304348]])
```

In [11]:
```
# Step 7A: Plot in ORIGINAL units with centers
plt.figure(figsize=(6,5))
for c in range(k_opt):
    mask = df_km['Cluster'] == c
    plt.scatter(df_km.loc[mask, 'Income'], df_km.loc[mask, 'SpendingScore'], label=

plt.scatter(centers_original[:,0], centers_original[:,1], marker='X', s=200, label=
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.title(f"K-Means Clusters (k={k_opt})")
plt.legend()
plt.show()

# Step 7B: Silhouette score for your chosen k
km_sil = silhouette_score(X_scaled, cluster_labels)
km_sil
```



Out[11]: np.float64(0.5546571631111091)

In [16]:
```
# Step 8: Cluster summary
summary = df_km.groupby('Cluster').agg(
    customers=('Cluster', 'size'),
    avg_income=('Income', 'mean'),
    median_income=('Income', 'median'),
    avg_spend=('SpendingScore', 'mean'),
    median_spend=('SpendingScore', 'median')
).sort_index()

summary
```

Out[16]:

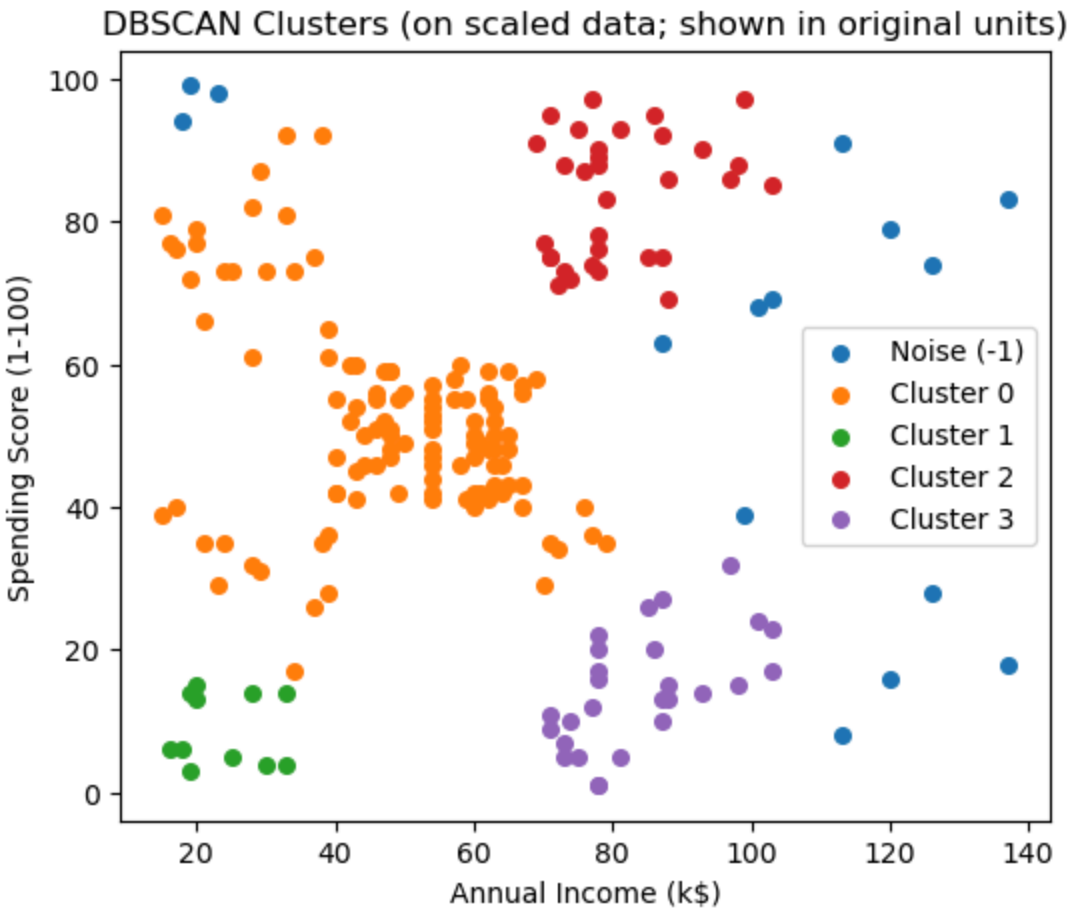
Cluster	customers	avg_income	median_income	avg_spend	median_spend
0	81	55.296296	54.0	49.518519	50.0
1	39	86.538462	79.0	82.128205	83.0
2	22	25.727273	24.5	79.363636	77.0
3	35	88.200000	85.0	17.114286	16.0
4	23	26.304348	25.0	20.913043	17.0

In [17]:
```
# Save labeled results
df_km.to_csv("segmented_customers_kmeans.csv", index=False)
summary.to_csv("cluster_summary_kmeans.csv")
```

In [18]:
```
#  Quick sweep to see #clusters vs eps
eps_list = [0.2, 0.3, 0.4, 0.5, 0.6]
for eps in eps_list:
    db = DBSCAN(eps=eps, min_samples=5)
    labels = db.fit_predict(X_scaled)
    n_noise = (labels == -1).sum()
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    print(f"eps={eps:.1f}: clusters={n_clusters}, noise={n_noise}")
```

```
eps=0.2: clusters=7, noise=77
eps=0.3: clusters=7, noise=35
eps=0.4: clusters=4, noise=15
eps=0.5: clusters=2, noise=8
eps=0.6: clusters=1, noise=5
```

In [19]:
```
# Fit DBSCAN with a chosen eps
# Pick an eps from the sweep that yields a sensible #clusters (e.g., 2–6) and not t
db = DBSCAN(eps=0.4, min_samples=5)  # adjust eps based on the sweep above
db_labels = db.fit_predict(X_scaled)

df_db = df.copy()
df_db['DBSCAN_Cluster'] = db_labels

# Plot
plt.figure(figsize=(6,5))
for label in sorted(set(db_labels)):
    mask = (df_db['DBSCAN_Cluster'] == label)
    name = f"Cluster {label}" if label != -1 else "Noise (-1)"
    plt.scatter(df_db.loc[mask, 'Income'], df_db.loc[mask, 'SpendingScore'], s=30,

plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.title("DBSCAN Clusters (on scaled data; shown in original units)")
plt.legend()
plt.show()

# Cluster-wise average spending (exclude noise)
db_summary = df_db[df_db['DBSCAN_Cluster'] != -1].groupby('DBSCAN_Cluster').agg(
    customers=('DBSCAN_Cluster','size'),
    avg_income=('Income','mean'),
    avg_spend=('SpendingScore','mean')
).sort_index()

db_summary
```



Out[19]:

DBSCAN_Cluster	customers	avg_income	avg_spend
0	115	48.304348	51.730435
1	11	23.727273	8.909091
2	32	80.875000	83.625000
3	27	83.925926	14.444444

In [25]:
```
#Testing system
import pandas as pd
# Number of customers in each cluster
print(df_km['Cluster'].value_counts())
```

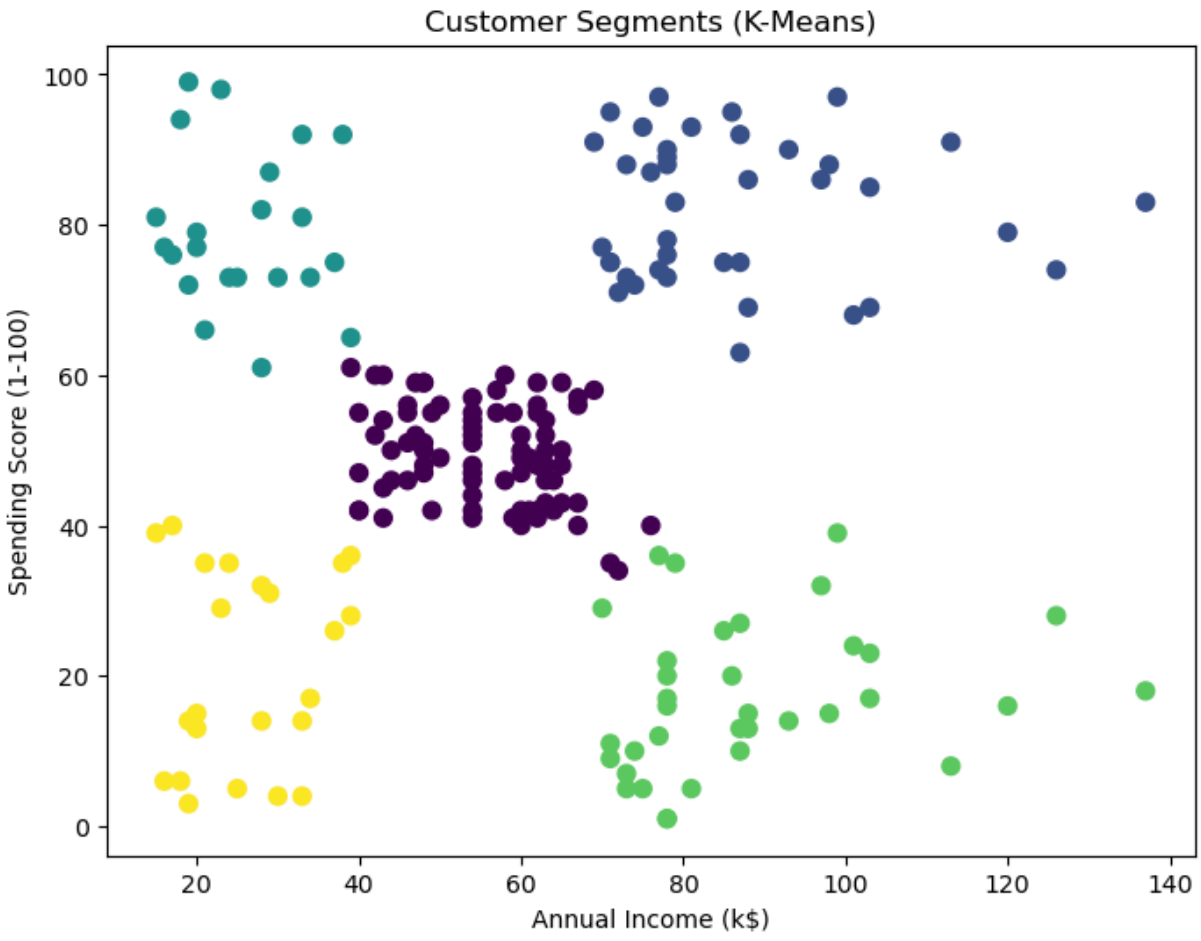
```
Cluster
0    81
1    39
3    35
4    23
2    22
Name: count, dtype: int64
```

In [28]:
```
import pandas as pd
```

```
# Number of customers in each cluster
print(df_db['DBSCAN_Cluster'].value_counts())
```

```
DBSCAN_Cluster
 0    115
 2     32
 3     27
-1     15
 1     11
Name: count, dtype: int64
```

In [36]:
```
#Visualize cluster 2D plot
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
plt.scatter(df_km['Income'], df_km['SpendingScore'],
            c=df_km['Cluster'], cmap='viridis', s=50)
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Customer Segments (K-Means)')
plt.show()
```



In [40]:
```
avg_spending = df_km.groupby('Cluster')['SpendingScore'].mean()
print(avg_spending)
```

```
Cluster
0    49.518519
1    82.128205
2    79.363636
3    17.114286
4    20.913043
Name: SpendingScore, dtype: float64
```

In [41]:
```
import pandas as pd

# Load Mall Customers dataset
df = pd.read_csv("Mall_Customers.csv")
print(df.head())
```

```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

In [42]:
```
X = df[['Annual Income (k$)', 'Spending Score (1-100)']]
print(X.head())
```

```
   Annual Income (k$)  Spending Score (1-100)
0                  15                      39
1                  15                      81
2                  16                       6
3                  16                      77
4                  17                      40
```

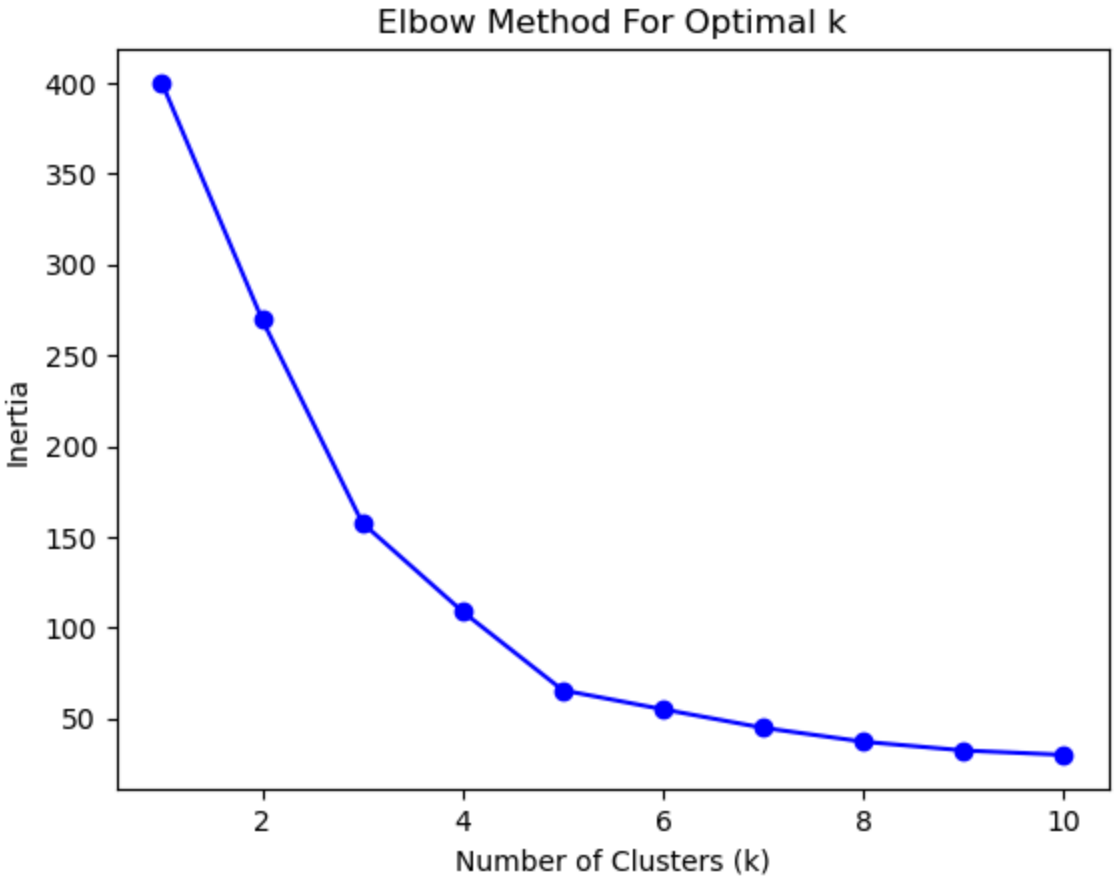
In [43]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print(X_scaled[:5])  # check first 5 scaled values
```

```
[[-1.73899919 -0.43480148]
 [-1.73899919  1.19570407]
 [-1.70082976 -1.71591298]
 [-1.70082976  1.04041783]
 [-1.66266033 -0.39597992]]
```

In [47]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

inertia = []
K = range(1, 11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot elbow curve
plt.plot(K, inertia, 'bo-')
```

```
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```

```
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```



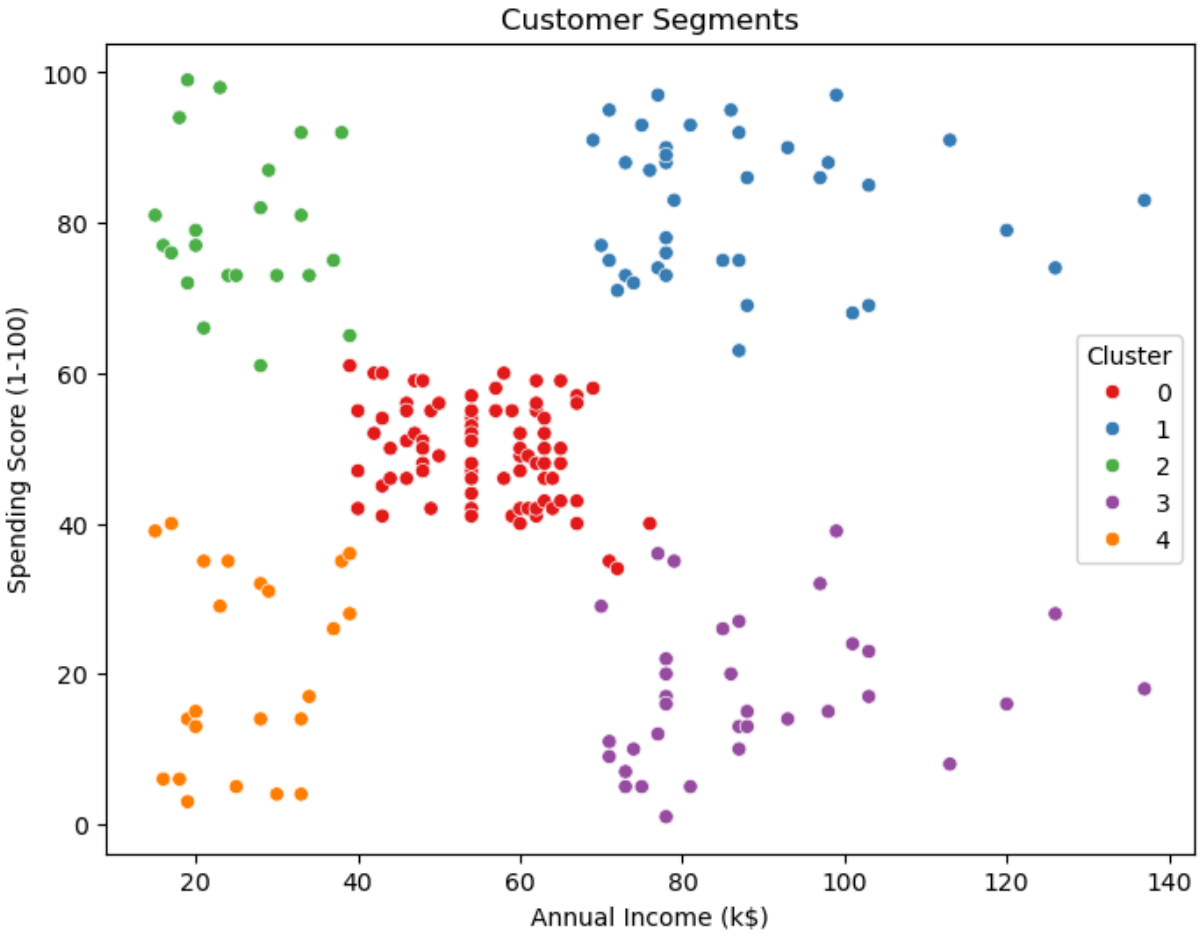
In [45]:
```
kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)

print(df.head())
```

```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)  \
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40

   Cluster
0        4
1        2
2        4
3        2
4        4
```

```
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```

In [46]:
```
import seaborn as sns

plt.figure(figsize=(8,6))
sns.scatterplot(x='Annual Income (k$)',
```

```
                y='Spending Score (1-100)',
                hue='Cluster',
                palette='Set1',
                data=df)
plt.title("Customer Segments")
plt.show()
```



In [48]:
```
kmeans = KMeans(n_clusters=5, random_state=42)
df['Cluster'] = kmeans.fit_predict(df[['Annual Income (k$)', 'Spending Score (1-100

# Cluster Size Output
cluster_counts = df['Cluster'].value_counts().sort_index()
print("Number of customers in each cluster:")
print(cluster_counts)
```

```
Number of customers in each cluster:
Cluster
0    81
1    39
2    22
3    35
4    23
Name: count, dtype: int64
```

```
C:\Users\aaa\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarnin
g: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```

In [ ]: