```
# 1. Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score
```

```
# 2. Load the California Housing dataset
cal_housing = fetch_california_housing(as_frame=True)

# Features (X) and target (y)
X = cal_housing.data       # features
y = cal_housing.target     # median house value (for interpretation only)

# Combine into one DataFrame for easier exploration
df = cal_housing.frame     # includes both X and y
df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Next steps: Generate code with df | New interactive sheet

```
# 3. Basic data exploration
print("Shape of dataset:", df.shape)
print("\nColumns:\n", df.columns)
print("\nInfo:")
print(df.info())
print("\nSummary statistics:")
print(df.describe())

# 4. Check for missing values
print("\nMissing values per column:")
print(df.isna().sum())
```

```
Shape of dataset: (20640, 9)

Columns:
 Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude', 'MedHouseVal'],
      dtype='object')

Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
None

Summary statistics:
```

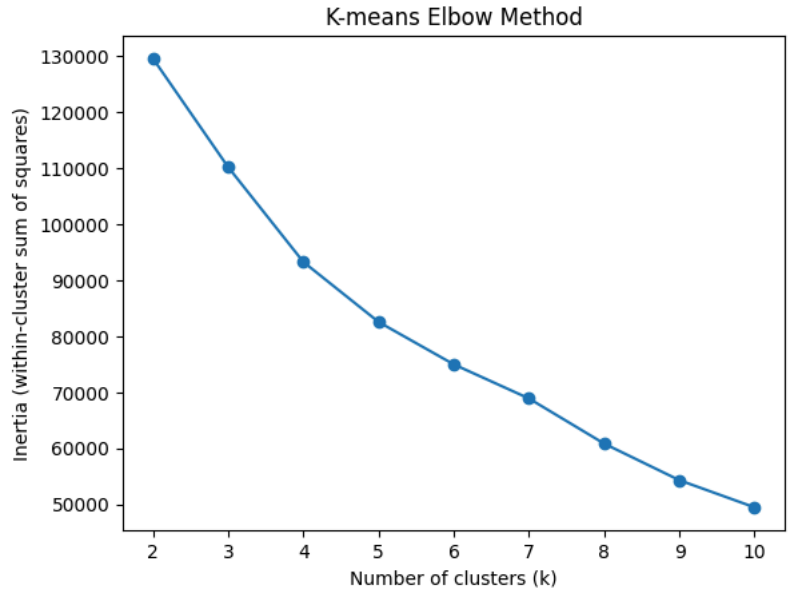
```
             MedInc      HouseAge      AveRooms     AveBedrms    Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.870671     28.639486      5.429000      1.096675   1425.476744
std        1.899822     12.585558      2.474173      0.473911   1132.462122
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.563400     18.000000      4.440716      1.006079    787.000000
50%        3.534800     29.000000      5.229129      1.048780   1166.000000
75%        4.743250     37.000000      6.052381      1.099526   1725.000000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

           AveOccup      Latitude     Longitude   MedHouseVal
count  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704      2.068558
std       10.386050      2.135952      2.003532      1.153956
min        0.692308     32.540000   -124.350000      0.149990
25%        2.429741     33.930000   -121.800000      1.196000
50%        2.818116     34.260000   -118.490000      1.797000
75%        3.282261     37.710000   -118.010000      2.647250
max     1243.333333     41.950000   -114.310000      5.000010

Missing values per column:
MedInc         0
HouseAge       0
AveRooms       0
AveBedrms      0
Population     0
AveOccup       0
Latitude       0
Longitude      0
MedHouseVal    0
dtype: int64
```

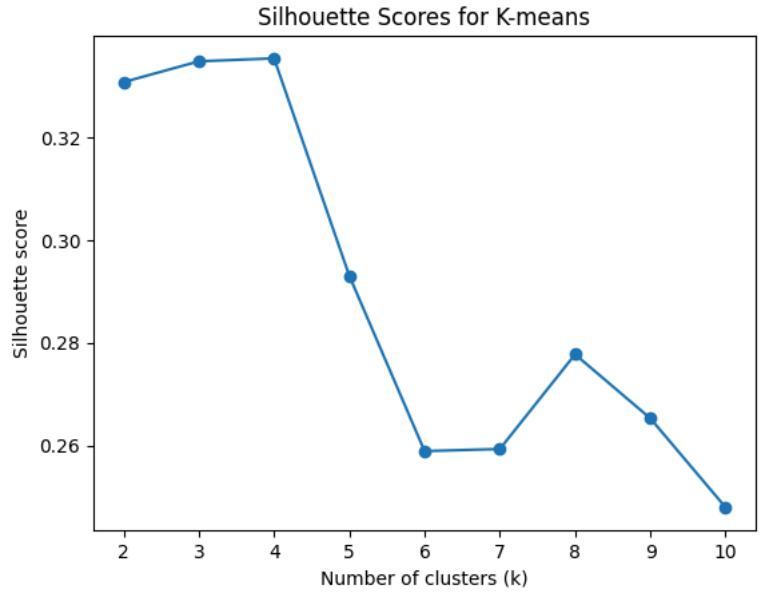
```
# 5. Prepare features for clustering (exclude target)
feature_cols = cal_housing.feature_names   # list of feature names
X_features = df[feature_cols].copy()

# 6. Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_features)

print("Scaled feature shape:", X_scaled.shape)
```

```
Scaled feature shape: (20640, 8)
```

```
# 7. Determine optimal number of clusters for K-means
inertias = []
sil_scores = []
K_range = range(2, 11)   # Try k from 2 to 10

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X_scaled)
    inertias.append(kmeans.inertia_)
    sil = silhouette_score(X_scaled, labels)
    sil_scores.append(sil)
    print(f"k={k}: inertia={kmeans.inertia_:.2f}, silhouette={sil:.4f}")

# 8. Plot Elbow curve (inertia)
plt.figure()
plt.plot(list(K_range), inertias, marker='o')
plt.xlabel("Number of clusters (k)")
plt.ylabel("Inertia (within-cluster sum of squares)")
plt.title("K-means Elbow Method")
plt.show()

# 9. Plot Silhouette scores
plt.figure()
plt.plot(list(K_range), sil_scores, marker='o')
plt.xlabel("Number of clusters (k)")
plt.ylabel("Silhouette score")
plt.title("Silhouette Scores for K-means")
plt.show()

# 10. Choose best k as the one with the highest silhouette score
best_k = K_range[np.argmax(sil_scores)]
print("Best k based on silhouette score:", best_k)
```

```
k=2: inertia=129613.19, silhouette=0.3308
k=3: inertia=110213.28, silhouette=0.3349
k=4: inertia=93278.18, silhouette=0.3355
k=5: inertia=82558.81, silhouette=0.2931
k=6: inertia=74984.96, silhouette=0.2589
k=7: inertia=68890.42, silhouette=0.2593
k=8: inertia=60832.39, silhouette=0.2777
k=9: inertia=54301.17, silhouette=0.2653
k=10: inertia=49454.93, silhouette=0.2479
```






```
Best k based on silhouette score: 4
```

```
#11. Fit K-means with best_k
from sklearn.cluster import KMeans
kmeans_final = KMeans(n_clusters=best_k, random_state=42, n_init=10)
kmeans_labels = kmeans_final.fit_predict(X_scaled)

# Attach K-means cluster labels to the original DataFrame
df['kmeans_cluster'] = kmeans_labels

# 12. Inspect average features per cluster (and average target)
cluster_profile_kmeans = df.groupby('kmeans_cluster')[feature_cols + ['MedHouseVal']].mean()
cluster_profile_kmeans
```

kmeans_cluster	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	3.804135	29.027008	5.503493	1.081950	1286.128086	2.824753	37.950009	-121.730083	1.973423
1	3.921187	28.418182	5.207149	1.071848	1532.163469	3.056250	33.942571	-118.008752	2.140146
2	6.669400	42.333333	5.795482	1.094628	6063.333333	781.836386	38.016667	-121.063333	1.850000
3	3.373379	18.733333	32.149384	6.741434	291.360000	2.497688	37.710133	-119.408267	1.628373

Next steps: Generate code with cluster_profile_kmeans | New interactive sheet

```
# 13. Apply Agglomerative Clustering using the same number of clusters (best_k)
agg = AgglomerativeClustering(n_clusters=best_k, linkage='ward')
agg_labels = agg.fit_predict(X_scaled)

df['agg_cluster'] = agg_labels

# 14. Evaluate Agglomerative using silhouette score
agg_silhouette = silhouette_score(X_scaled, agg_labels)
print(f"Agglomerative Clustering silhouette score (k={best_k}): {agg_silhouette:.4f}")

# 15. Compare with K-means silhouette
kmeans_silhouette = silhouette_score(X_scaled, kmeans_labels)
print(f"K-means silhouette score (k={best_k}): {kmeans_silhouette:.4f}")

# 16. Cluster profile for Agglomerative
cluster_profile_agg = df.groupby('agg_cluster')[feature_cols + ['MedHouseVal']].mean()
cluster_profile_agg
```

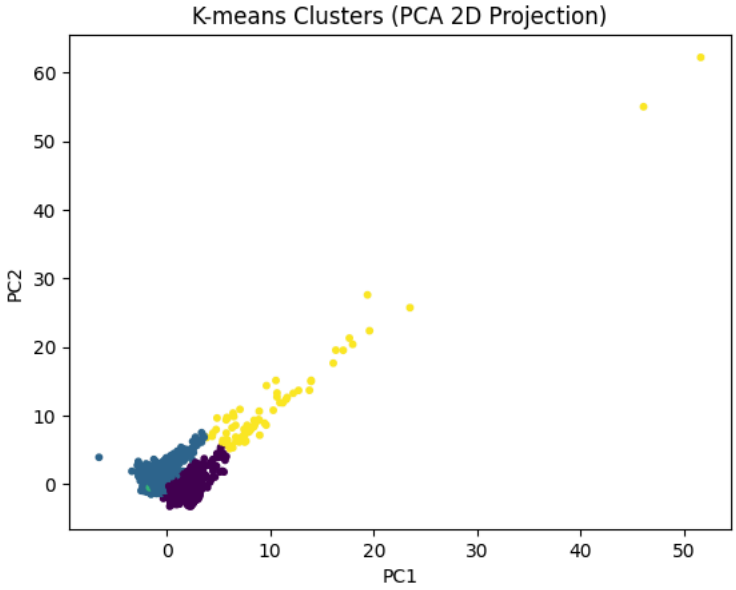
```
Agglomerative Clustering silhouette score (k=4): 0.3232
K-means silhouette score (k=4): 0.3355
```

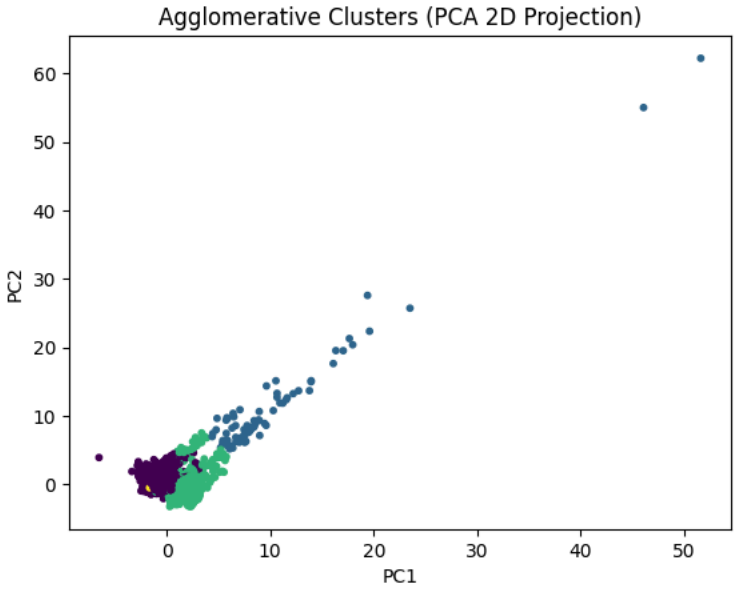
agg_cluster	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	3.935994	28.192973	5.217788	1.065418	1537.071667	3.081337	34.164741	-118.197095	2.101582
1	3.364109	18.786667	32.165032	6.735987	284.813333	2.502471	37.751467	-119.439333	1.634013
2	3.764711	29.481158	5.522474	1.094074	1247.206155	2.753612	38.076698	-121.878005	2.017374
3	6.669400	42.333333	5.795482	1.094628	6063.333333	781.836386	38.016667	-121.063333	1.850000

Next steps: Generate code with cluster_profile_agg | New interactive sheet

```
# 17. PCA to 2 dimensions for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("Explained variance ratios:", pca.explained_variance_ratio_)
print("Total variance explained by first 2 components:",
      pca.explained_variance_ratio_.sum())

# 18. Plot K-means clusters in PCA space
plt.figure()
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, s=10)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("K-means Clusters (PCA 2D Projection)")
plt.show()

# 19. Plot Agglomerative clusters in PCA space
plt.figure()
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=agg_labels, s=10)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Agglomerative Clusters (PCA 2D Projection)")
plt.show()
```

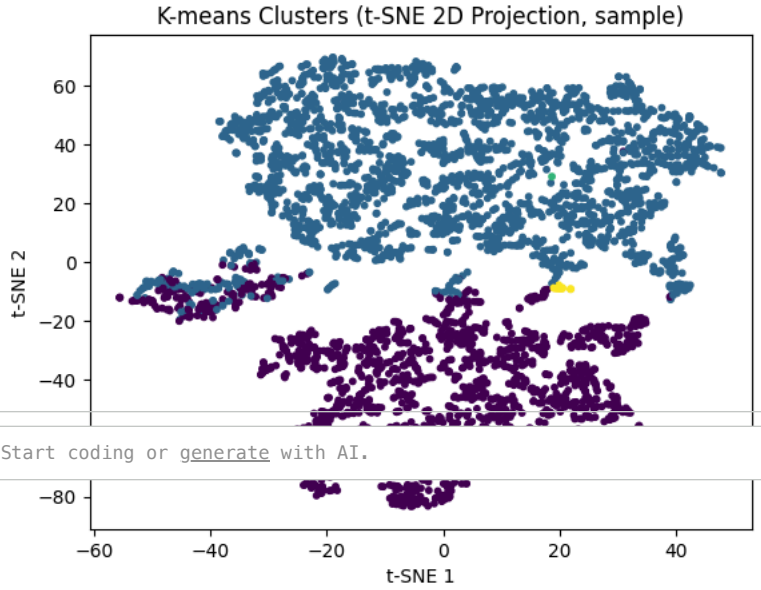
```
Explained variance ratios: [0.25336868 0.23516245]
Total variance explained by first 2 components: 0.4885311266922721
```






```
# 20. t-SNE for a non-linear 2D visualization (can be slow on large data)
# (Optionally, you can sample a subset for speed)
sample_frac = 0.2   # 20% of data
sample_idx = np.random.choice(len(X_scaled), size=int(len(X_scaled)*sample_frac), replace=False)
X_sample = X_scaled[sample_idx]
kmeans_sample_labels = kmeans_labels[sample_idx]

tsne = TSNE(n_components=2, perplexity=30, n_iter=1000, random_state=42)
X_tsne = tsne.fit_transform(X_sample)

plt.figure()
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=kmeans_sample_labels, s=10)
plt.xlabel("t-SNE 1")
plt.ylabel("t-SNE 2")
plt.title("K-means Clusters (t-SNE 2D Projection, sample)")
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/manifold/_t_sne.py:1164: FutureWarning: 'n_iter' was renamed to 'max_
  warnings.warn(
```




```
Start coding or generate with AI.
```