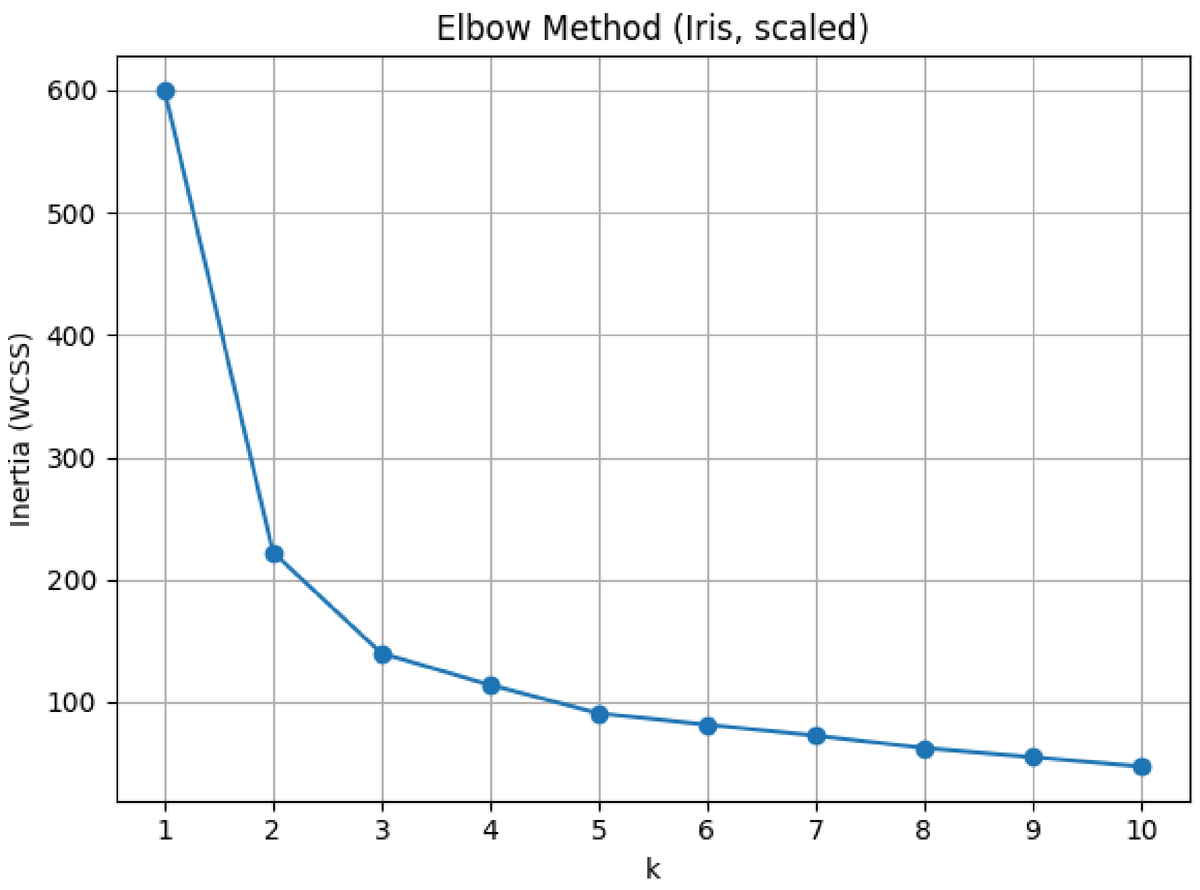
In [116…
```
"""
K-Means and Agglomerative Clustering on the Iris dataset

K-Means:  Uses scaling, elbow, and a 2D scatter on two features

Agglomerative:  Uses scaling, linkage, dendrogram and a 2D scatter on two fe

"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# --------------------------------
# 1) Load data
# --------------------------------
iris = load_iris()
X = iris.data                                 # shape (150, 4)
feature_names = iris.feature_names            # 4 feature names

df = pd.DataFrame(X, columns=feature_names)

print("Samples:", X.shape[0], "| Features:", X.shape[1])
print("Feature names:", feature_names)

# --------------------------------
# 2) Scale features
# --------------------------------
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
Samples: 150 | Features: 4
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (c
m)', 'petal width (cm)']
```

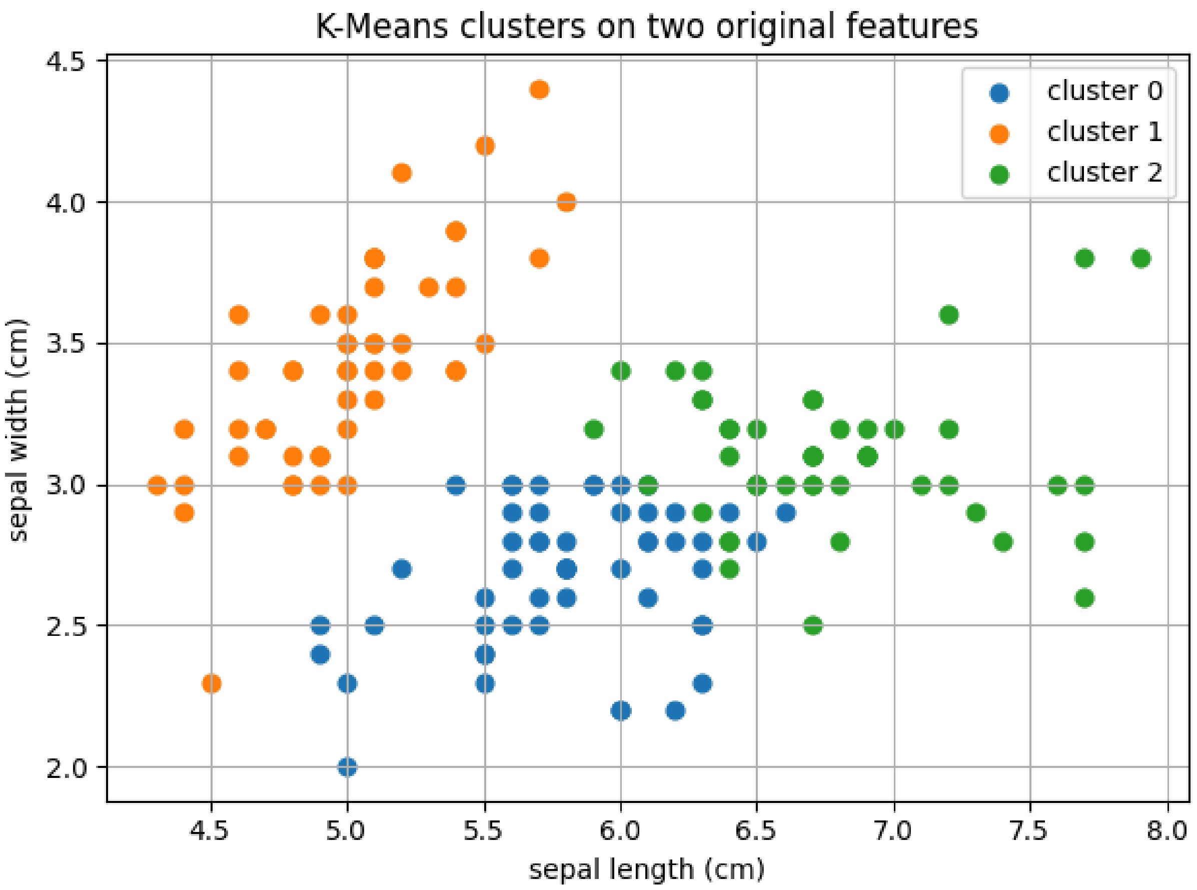
In [117…
```
############################################
## K Means Clustering
############################################
```

In [120…
```
# --------------------------------
# 3) Elbow method (find k)
# --------------------------------
ks = range(1, 11)
inertias = []
for k in ks:
    km = KMeans(n_clusters=k, init="k-means++", n_init=10,
                random_state=42)
    km.fit(X_scaled)
    inertias.append(km.inertia_)

plt.figure()
```

```python
plt.plot(ks, inertias, marker="o")
plt.title("Elbow Method (Iris, scaled)")
plt.xlabel("k")
plt.ylabel("Inertia (WCSS)")
plt.xticks(list(ks))
plt.grid()
plt.tight_layout()
plt.show()

# Choose k (for Iris, k=3 is natural)
k = 3

# --------------------------------
# 4) Fit final K-Means (k=3)
# -------------------------------
kmeans = KMeans(n_clusters=k, init="k-means++", n_init = 10,
                random_state=10)
labels = kmeans.fit_predict(X_scaled)

print("\nCluster sizes:", np.bincount(labels))
print("Final inertia (WCSS):", kmeans.inertia_)
print("Centroids in scaled space:\n", kmeans.cluster_centers_)

# --------------------------------
# 5) Simple 2D scatter (pick any two features)
#    just plot two original features
# --------------------------------
# Choose two feature indices to plot
i, j = 0, 1  # sepal length vs sepal width often shows structure

plt.figure()
for c in range(k):
    mask = (labels == c)
    plt.scatter(X[mask, i], X[mask, j], label=f"cluster {c}", s=40)
plt.xlabel(feature_names[i])
plt.ylabel(feature_names[j])
plt.title("K-Means clusters on two original features")
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()

# --------------------------------
# 6) Predict new samples
# --------------------------------
new_samples = np.array([
    [5.0, 3.5, 1.3, 0.3],  # Setosa-like
    [6.0, 2.7, 5.1, 1.6],  # Versicolor/Virginica-like
])
new_scaled = scaler.transform(new_samples)
pred = kmeans.predict(new_scaled)
print("\nPredicted clusters for new samples:", pred)
```



```
Cluster sizes: [53 50 47]
Final inertia (WCSS): 139.82049635974982
Centroids in scaled space:
 [[-0.05021989 -0.88337647  0.34773781  0.2815273 ]
 [-1.01457897  0.85326268 -1.30498732 -1.25489349]
 [ 1.13597027  0.08842168  0.99615451  1.01752612]]
```



```
Predicted clusters for new samples: [1 0]
```

In [ ]:

```

```

In [106…

```
############################################
## Hierarchical Clustering: Agglomerative
############################################
```

In [107…

```
# ============================================================
# WHAT IS SCIPY?
# ============================================================

# SciPy = Scientific Python
# It's a library built ON TOP of NumPy for scientific computing

# Think of it this way:
# NumPy      --> Basic math, arrays, linear algebra
# SciPy      --> Advanced scientific tools (statistics, optimization, etc.)
# Scikit-Learn --> Machine learning (built on NumPy)

from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.cluster import AgglomerativeClustering

# ============================================================
# WHAT IS fcluster?
# ============================================================

# fcluster = "Form clusters"
```

```
#
# It's a SciPy function that CUTS a hierarchical tree
# to get final cluster assignments for each sample

# Think of it like:
# 1. linkage() builds a TREE of how clusters merge
# 2. dendrogram() VISUALIZES that tree
# 3. fcluster() CUTS the tree to get final clusters
```

In [108…

```
# ================================================
# 2) AGGLOMERATIVE CLUSTERING (Bottom-up)
# ================================================
print("\n" + "="*60)
print("AGGLOMERATIVE CLUSTERING (Bottom-up)")
print("="*60)
```

```
============================================================
AGGLOMERATIVE CLUSTERING (Bottom-up)
============================================================
```

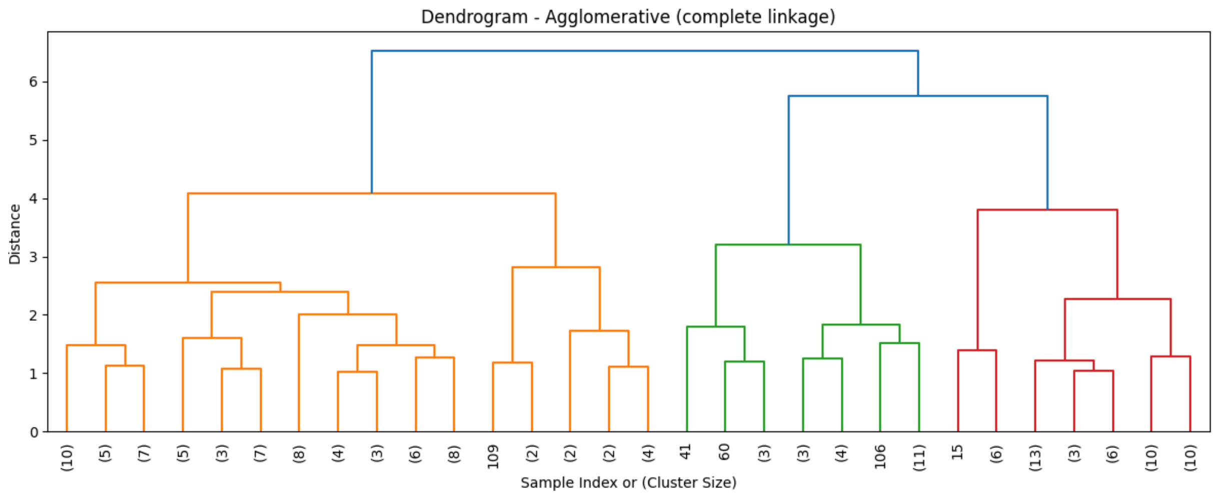
In [109…

```
# Different linkage methods
linkage_methods = ["complete", "average", "single"]
```

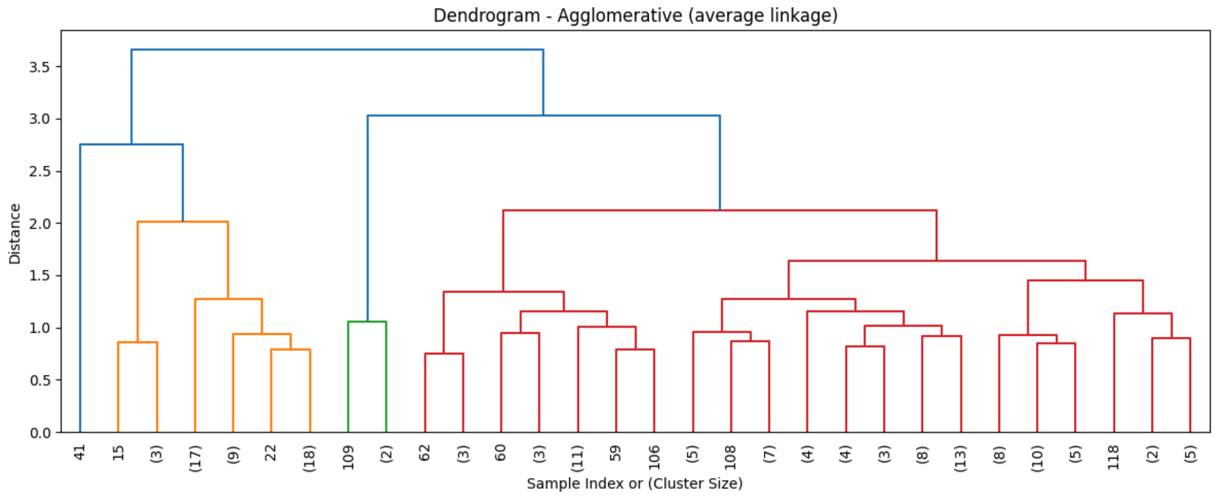
In [110…

```
for method in linkage_methods:
    print(f"\n--- Linkage Method: {method} ---")

    # Compute linkage matrix
    Z = linkage(X_scaled, method=method)

    # Create dendrogram
    plt.figure(figsize=(12, 5))
    dendrogram(Z, truncate_mode='lastp', p=30,
               leaf_rotation=90., leaf_font_size=10.)
    plt.title(f"Dendrogram - Agglomerative ({method} linkage)")
    plt.xlabel("Sample Index or (Cluster Size)")
    plt.ylabel("Distance")
    plt.tight_layout()
    plt.show()

    # Cluster with k=3
    agg_clustering = AgglomerativeClustering(n_clusters=3, linkage=method)
    agg_labels = agg_clustering.fit_predict(X_scaled)

    print(f"Cluster sizes: {np.bincount(agg_labels)}")
```

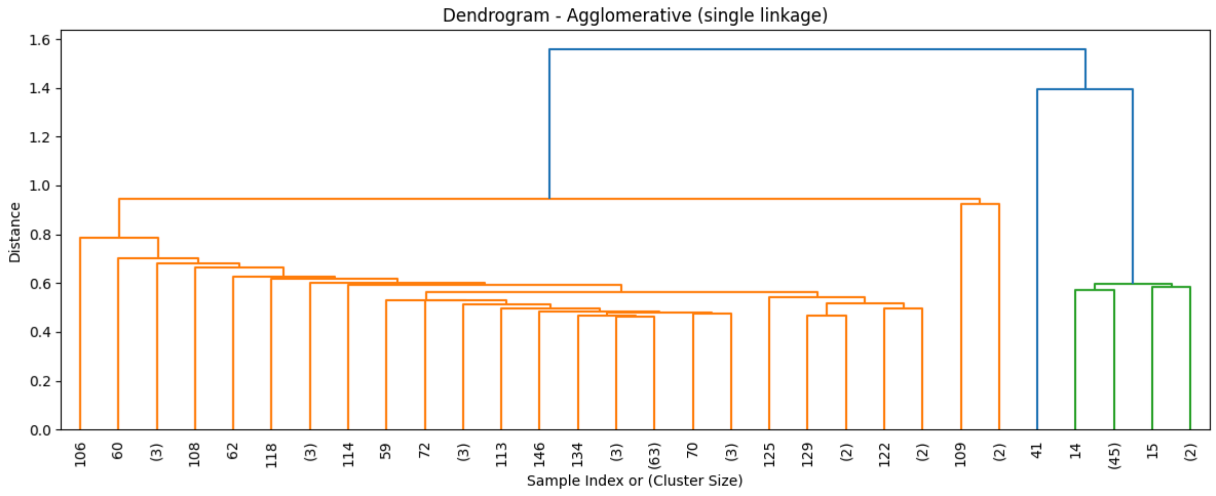
```
--- Linkage Method: complete ---
```



```
Cluster sizes: [77 49 24]

--- Linkage Method: average ---
```



```
Cluster sizes: [50 97  3]

--- Linkage Method: single ---
```

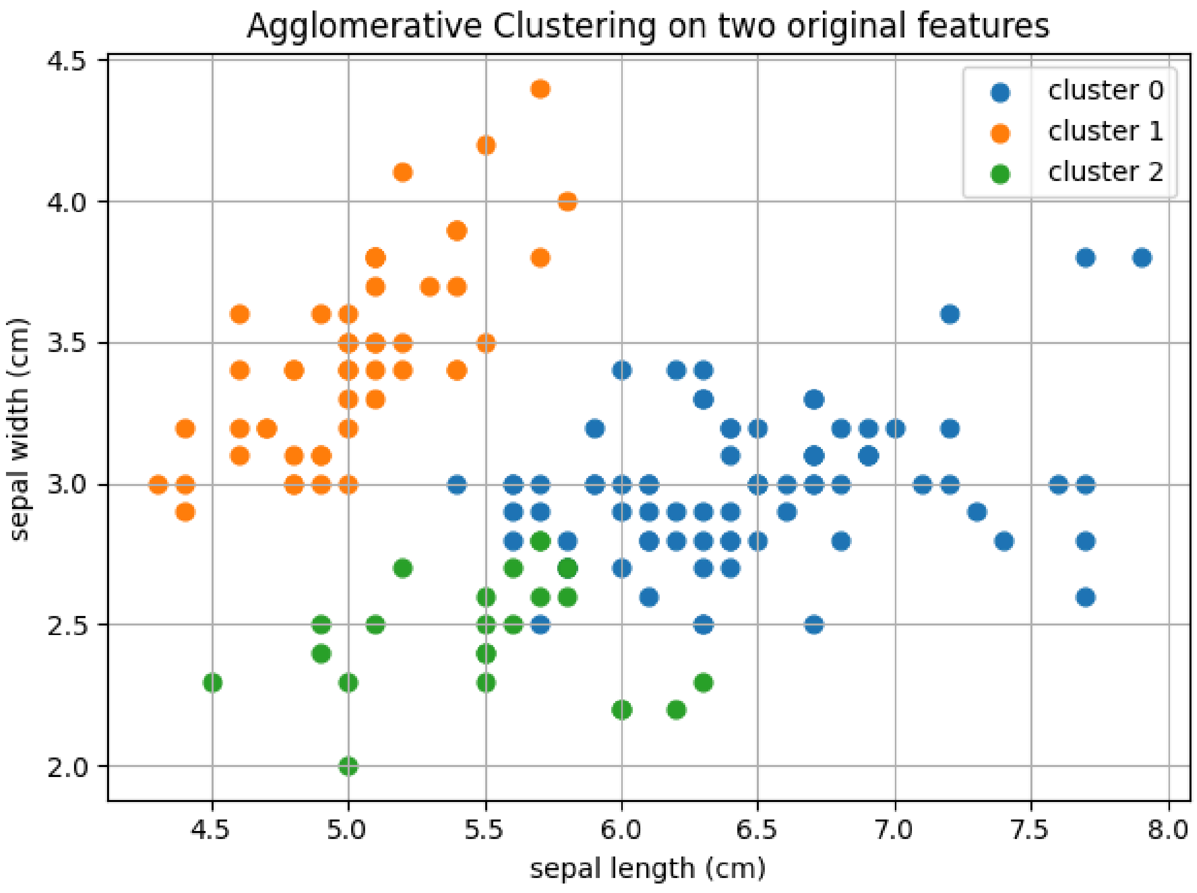


```
Cluster sizes: [100   1  49]
```

In [111…

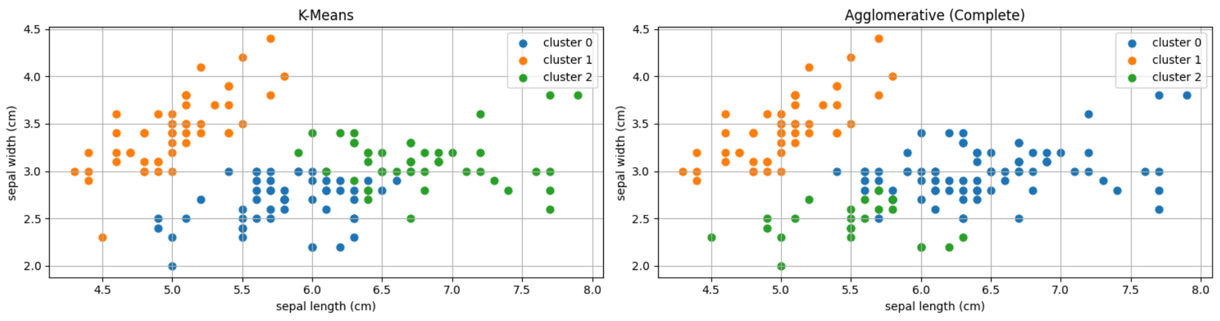
```
# ================================================
# 3) Agglomerative with Complete linkage
# ================================================
print("\n" + "="*60)
print("DETAILED AGGLOMERATIVE (Complete Linkage, k=3)")
print("="*60)

agg_complete = AgglomerativeClustering(n_clusters=3, linkage="complete")
agg_labels = agg_complete.fit_predict(X_scaled)

# # It COUNTS how many times each number appears in an array
print(f"Cluster sizes: {np.bincount(agg_labels)}")

# 2D scatter plot for agglomerative
i, j = 0, 1 # sepal length vs sepal width often shows structure

plt.figure()
for c in range(3):
    mask = (agg_labels == c)
    plt.scatter(X[mask, i], X[mask, j], label=f"cluster {c}", s=40)
plt.xlabel(feature_names[i])
plt.ylabel(feature_names[j])
plt.title("Agglomerative Clustering on two original features")
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```

```
============================================================
DETAILED AGGLOMERATIVE (Complete Linkage, k=3)
============================================================
Cluster sizes: [77 49 24]
```



```
In [122…  # ==================================================
          # 4) Compare K-Means and Agglomerative
          # ==================================================
          print("\n" + "="*60)
          print("COMPARISON: K-Means vs Agglomerative")
          print("="*60)

          from sklearn.cluster import KMeans

          # K-Means
          kmeans = KMeans(n_clusters=3, init="k-means++", n_init=10,
                          random_state=42)
          kmeans_labels = kmeans.fit_predict(X_scaled)

          # Agglomerative (Compelte)
          agg_labels_complete = AgglomerativeClustering(n_clusters=3,
                          linkage="complete").fit_predict(X_scaled)



          # Plot comparison
          fig, axes = plt.subplots(1, 2, figsize=(15, 4))

          methods = ["K-Means", "Agglomerative (Complete)"]
          labels_list = [kmeans_labels, agg_labels_complete]

          for idx, (ax, method, labels) in enumerate(zip(axes,methods,labels_list)):
              for c in range(3):
                  mask = (labels == c)
                  ax.scatter(X[mask, i], X[mask, j], label=f"cluster {c}", s=40)
              ax.set_xlabel(feature_names[i])
              ax.set_ylabel(feature_names[j])
              ax.set_title(f"{method}")
              ax.grid()
              ax.legend()

          plt.tight_layout()
          plt.show()
```

```
============================================================
COMPARISON: K-Means vs Agglomerative
============================================================
```



```
In [123…  # ================================================================
          # 5) Predict new samples (Agglomerative doesn't support predict)
          # ================================================================
          print("\n" + "="*60)
print("SUMMARY")
print("="*60)
print("\nNote: Agglomerative & Divisive are distance-based hierarchical meth
print("- K-Means: Centroid-based, iterative")
print("- Agglomerative: Bottom-up, builds hierarchy from individual points")
print("- Divisive: Most ML libraries focus on agglomerative instead")
print("\nFor new sample prediction, K-Means is more suitable.")
print("For hierarchy interpretation, use dendrograms from hierarchical metho
```

```
============================================================
SUMMARY
============================================================

Note: Agglomerative & Divisive are distance-based hierarchical methods
- K-Means: Centroid-based, iterative
- Agglomerative: Bottom-up, builds hierarchy from individual points
- Divisive: Most ML libraries focus on agglomerative instead

For new sample prediction, K-Means is more suitable.
For hierarchy interpretation, use dendrograms from hierarchical methods.
```

In [ ]:

In [ ]: