In-Class Exercises: Solutions

Instructor copy – do not distribute before the exercise session.

Data Science with Python – BSc Course

Exercise 1: K-Means by Hand — Solution

Iteration 1 – Assignment

Initial centroids: $\mu_1 = (1, 1)$, $\mu_2 = (8, 8)$.

Point	(x, y)	$d(\text{point}, \mu_1)$	$d(\text{point}, \mu_2)$	**Assign to**
A	$(1, 1)$	$\sqrt{(1-1)^2 + (1-1)^2} = 0.00$	$\sqrt{(1-8)^2 + (1-8)^2} = 9.90$	C1
B	$(2, 1)$	$\sqrt{(2-1)^2 + (1-1)^2} = 1.00$	$\sqrt{(2-8)^2 + (1-8)^2} = 9.22$	C1
C	$(1.5, 2)$	$\sqrt{(1.5-1)^2 + (2-1)^2} = 1.12$	$\sqrt{(1.5-8)^2 + (2-8)^2} = 8.85$	C1
D	$(7, 8)$	$\sqrt{(7-1)^2 + (8-1)^2} = 9.22$	$\sqrt{(7-8)^2 + (8-8)^2} = 1.00$	C2
E	$(8, 7)$	$\sqrt{(8-1)^2 + (7-1)^2} = 9.22$	$\sqrt{(8-8)^2 + (7-8)^2} = 1.00$	C2
F	$(8, 9)$	$\sqrt{(8-1)^2 + (9-1)^2} = 10.63$	$\sqrt{(8-8)^2 + (9-8)^2} = 1.00$	C2

Cluster 1: {A, B, C} **Cluster 2:** {D, E, F}

Iteration 1 – Update centroids

$$\mu_1^{\text{new}} = \left(\frac{1+2+1.5}{3},\ \frac{1+1+2}{3}\right) = (1.50,\ 1.33)$$

$$\mu_2^{\text{new}} = \left(\frac{7+8+8}{3},\ \frac{8+7+9}{3}\right) = (7.67,\ 8.00)$$

Iteration 2 – Assignment

New centroids: $\mu_1 = (1.50, 1.33)$, $\mu_2 = (7.67, 8.00)$.

Point	(x, y)	$d(\text{point}, \mu_1)$	$d(\text{point}, \mu_2)$	**Assign to**
A	$(1, 1)$	$\sqrt{0.25 + 0.11} = 0.60$	$\sqrt{44.49 + 49.00} = 9.67$	C1
B	$(2, 1)$	$\sqrt{0.25 + 0.11} = 0.60$	$\sqrt{32.15 + 49.00} = 9.01$	C1
C	$(1.5, 2)$	$\sqrt{0.00 + 0.45} = 0.67$	$\sqrt{38.07 + 36.00} = 8.60$	C1
D	$(7, 8)$	$\sqrt{30.25 + 44.49} = 8.65$	$\sqrt{0.45 + 0.00} = 0.67$	C2
E	$(8, 7)$	$\sqrt{42.25 + 32.15} = 8.63$	$\sqrt{0.11 + 1.00} = 1.05$	C2
F	$(8, 9)$	$\sqrt{42.25 + 58.83} = 10.05$	$\sqrt{0.11 + 1.00} = 1.05$	C2

No points change cluster. The algorithm has **converged**.

Key takeaway

(e) When no points change clusters, the centroids will also remain the same, so further iterations produce identical results. K-Means always converges, though it may converge to a local (not global) minimum of the objective function J.

Final result:

- Cluster 1: {A, B, C} with centroid $(1.50, 1.33)$
- Cluster 2: {D, E, F} with centroid $(7.67, 8.00)$
- Converged after 2 iterations

Exercise 2: Read the Dendrogram — Solution

Answers depend on the specific dendrogram from slide 15. The model answers below assume the Ward-linkage dendrogram shown in the lecture.

(a) Cutting at $h = 3$: The horizontal line at height 3 intersects multiple vertical branches. Count the intersections to get the number of clusters. Typically this yields 3–4 clusters for the lecture dendrogram.

(b) Cutting at $h = 6$: Fewer intersections, typically 2 clusters. A higher cut produces fewer, larger clusters.

(c) The first merge (lowest height) occurs between the two most similar items. A low merge height means those items are very close in the distance metric — they are nearly identical.

(d) The dendrogram shows all possible numbers of clusters simultaneously. You look for a large "gap" in merge heights — a tall vertical line before the next merge. Cutting just below that gap gives a natural number of clusters. This is analogous to the elbow method but visual and more informative.

General rule

Number of clusters = number of vertical lines crossed by the horizontal cut line.

Exercise 3: PCA – Keep or Drop? — Solution

(a) Explained variance ratios

Total variance $= 4.2 + 1.8 + 0.5 + 0.3 + 0.2 = 7.0$.

Component	PC1	PC2	PC3	PC4	PC5
Eigenvalue	4.2	1.8	0.5	0.3	0.2
Expl. Variance (%)	60.0	25.7	7.1	4.3	2.9
Cumulative (%)	60.0	85.7	92.9	97.1	100.0

(b) Kaiser rule

Keep components with eigenvalue > 1:

- PC1: $4.2 > 1$ — keep
- PC2: $1.8 > 1$ — keep
- PC3: $0.5 < 1$ — drop
- PC4: $0.3 < 1$ — drop
- PC5: $0.2 < 1$ — drop

Kaiser recommends 2 components (85.7% cumulative variance).

(c) Elbow rule

The eigenvalues drop sharply from 4.2 to 1.8 (large drop), then from 1.8 to 0.5 (another large drop), then flatten (0.5, 0.3, 0.2). The **elbow is at PC2 or PC3**. Most practitioners would read this as 2 components, with 3 as a reasonable alternative.

(d) 90% rule

- 2 components: 85.7% — below 90%
- 3 components: 92.9% — above 90%

The 90% rule requires 3 components.

(e) Summary

Rule	Components	Cumulative Variance
Kaiser (eigenvalue > 1)	2	85.7%
Elbow (scree plot bend)	2–3	85.7–92.9%
90% cumulative variance	3	92.9%

Recommendation: 2 components for quick visualization and exploration (captures 85.7%). 3 components if the application demands higher fidelity (92.9%). In practice, the Kaiser and elbow rules give the strongest signal here: the first 2 PCs capture the bulk of the variance, and PC3 adds only 7.1% more.

Exercise 4: Which Method? — Solution

	Description	Best Method	Reasoning
A	10K customers, 5 features, want exactly 4 segments	**K-Means**	Known K, moderate size, spherical clusters likely sufficient
B	200 stocks, want a hierarchy of relationships	**Hierarchical**	Need a tree/dendrogram; dataset small enough for $O(n^2)$
C	Transactions with unknown fraud	**DBSCAN**	Fraud = rare outliers = noise points; DBSCAN labels them automatically
D	50 features, reduce to 3 for visualization	**PCA**	Linear dim. reduction; fast, interpretable, preserves variance

Bonus: Why K-Means fails for fraud detection

K-Means assigns *every* point to a cluster — it has no concept of "noise" or "outlier." Fraudulent transactions (rare, unusual patterns) would be absorbed into the nearest normal cluster rather than being flagged. DBSCAN explicitly labels low-density points as noise, making it a natural fit for anomaly detection.

Alternative answers worth full credit

- Dataset A: GMM is also acceptable (provides soft assignments, works well with 5 features).
- Dataset C: Isolation Forest or one-class SVM are also valid anomaly detection methods, though they were not listed.
- Dataset D: t-SNE could be used for visualization, but PCA is preferred when you need the reduced features as input to downstream models (t-SNE is non-parametric and cannot transform new data).

Exercise 5: Build a Pipeline — Solution

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler        # (1)
from sklearn.decomposition import PCA                   # (2)
from sklearn.cluster import KMeans                      # (3)
import numpy as np

# Synthetic data: 500 samples, 10 features
np.random.seed(42)
X = np.random.randn(500, 10)

# Build the pipeline
pipe = Pipeline([
    ('scaler',  StandardScaler()),                      # (4)
    ('pca',     PCA(n_components=3)),                   # (5)
    ('cluster', KMeans(n_clusters=4, random_state=42)),# (6)
])

# Fit and get cluster labels
labels = pipe.fit_predict(X)                            # (7)

print("Cluster sizes:", np.bincount(labels))
print("PCA explained variance:",
      pipe.named_steps['pca'].explained_variance_ratio_)
```

Blank answers

Blank	Answer
(1)	`StandardScaler`
(2)	`PCA`
(3)	`KMeans`
(4)	`StandardScaler()`
(5)	`PCA(n_components=3)`
(6)	`KMeans(n_clusters=4, random_state=42)`
(7)	`fit_predict`

(b) Why scaler first?

PCA finds directions of maximum variance. If features have different scales (e.g., income in thousands vs. age in decades), the high-magnitude feature dominates the first principal component regardless of its actual importance. StandardScaler ensures all features contribute equally. K-Means uses Euclidean distance, which is also scale-sensitive.

(c) Data leakage

If you fit the scaler outside the pipeline on the full dataset, the scaler learns the mean and standard deviation from *all* data, including the validation fold. When `cross_val_score` then evaluates on the validation fold, the scaler has already "seen" those samples. This is **data leakage**: the model indirectly uses validation data during training, leading to optimistically biased performance estimates.

Inside a Pipeline, `cross_val_score` correctly fits the scaler only on the training fold of each split.

Exercise 6: Cluster Real Stocks — Solution

(a) Cluster profiles

Running the code produces three clusters with approximately these mean values (exact numbers depend on the random seed):

Cluster	Annual Return	Volatility	Market Cap (B)
0 (Stable / Blue Chips)	≈ 0.05	≈ 0.10	≈ 100
1 (Growth)	≈ 0.25	≈ 0.25	≈ 30
2 (Speculative)	≈ 0.50	≈ 0.45	≈ 5

Descriptive names:

- Cluster 0: "Blue Chips" — low return, low risk, large cap
- Cluster 1: "Growth Stocks" — moderate return and risk, mid-cap
- Cluster 2: "Speculative Bets" — high return, high risk, micro-cap

(b) Converting centroids back to original scale

`km.cluster_centers_` are in standardized (z-score) space. To convert back:

```
# Inverse transform: original = z * std + mean
centroids_original = scaler.inverse_transform(
    km.cluster_centers_
)
print(pd.DataFrame(
    centroids_original,
    columns=stocks.columns[:3]
).round(3))
```

The formula for each feature is: $x_{\text{original}} = x_{\text{scaled}} \times \sigma + \mu$, where σ and μ are the standard deviation and mean learned by the scaler.

(c) Without StandardScaler

Market cap ranges from roughly 2 to 130, while return and volatility range from roughly 0.01 to 0.60. Without scaling, `market_cap_B` has variance orders of magnitude larger than the other features. K-Means would cluster almost entirely by market cap, ignoring the risk-return profile. The result would be "big companies" vs. "small companies" rather than a meaningful risk segmentation.

Rule: Always scale features before distance-based algorithms (K-Means, DBSCAN, hierarchical clustering) unless all features already share the same unit and scale.