Clustering & Principal Component Analysis

AERO 689: Machine Learning for Aerospace Engineers

Raktim Bhattacharya

Texas A&M University

Overview

Topics Covered

1. Classification vs Clustering
2. Clustering Algorithms
3. Cluster Validation
4. Principal Component Analysis (PCA)
5. PCA Applications
6. Integration with Clustering

Classification vs Clustering

The Fundamental Difference

Classification (Week 4): Supervised learning

- Requires labeled training data
- Predicts known categories
- Goal: Generalization to new examples
- Example: "Is this bearing fault type A, B, or C?"

Clustering (Week 5): Unsupervised learning

- No labels required
- Discovers hidden patterns
- Goal: Finding natural groupings
- Example: "What natural fault patterns exist?"

Mathematical Distinction

Classification: Learn $f : X \to Y$ where $Y = \{0, 1, \ldots, K-1\}$ is known

- Training data: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Labels y_i are given
- Optimize to minimize prediction error

Clustering: Find $f : X \to \{1, 2, \ldots, k\}$ discovered from data

- Training data: $x_1, x_2, \ldots, x_n$ (no labels!)
- Number of clusters k may be unknown
- Optimize to find natural groups

When to Use Each Method

Use Classification when:

- You have labeled examples
- Categories are well-defined (taxi, takeoff, cruise)
- Regulatory requirements demand specific fault detection
- Problem: Requires expensive labeling effort

Use Clustering when:

- No labeled data available
- Exploring new operational regimes
- Discovering anomalies never seen before
- Initial data exploration before labeling

Application Context

Classification Example (Week 4):

- Trained on known fault types (normal, fault_a, fault_b, fault_c)
- Predicts fault type on new sensor data
- Requires historical fault database

Clustering Example (Week 5):

- Discovers natural patterns from unlabeled data
- Finds new degradation modes without prior knowledge
- Explores operational envelope
- Creates labels for future classification

Key Insight: Clustering creates labels; Classification uses labels

Decision Framework

Do you have labeled data?

- **YES** $\rightarrow$ Classification
  - Logistic regression
  - Decision trees
  - SVM
  - Neural networks

Do you have labeled data?

- **NO** $\rightarrow$ Clustering
  - K-means
  - DBSCAN
  - Hierarchical
  - Gaussian Mixture Models

Validation Challenge: How do we know clustering is "correct" without ground truth?

Clustering Fundamentals

What is Clustering?

Goal: Partition data into groups (clusters) where:

- Points in same cluster are similar
- Points in different clusters are dissimilar

Mathematical Formulation:

Minimize within-cluster variance:

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

where $\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$ is cluster center

K-Means Algorithm

Lloyd's Algorithm:

1. **Initialize**: Randomly select k cluster centers $\mu_1, \ldots, \mu_k$
2. **Assignment**: Assign each point to nearest center

$$c_i = \arg\min_j \|x_i - \mu_j\|^2$$

3. **Update**: Recompute centers as mean of assigned points

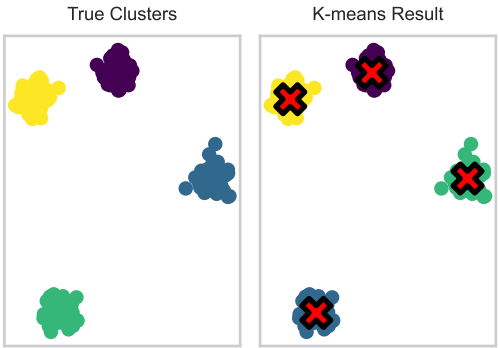
$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

4. **Repeat** steps 2-3 until convergence

Convergence: Guaranteed (objective function monotonically decreases)

Limitation: May converge to local minimum (solution: run multiple times)

K-Means Strength: Well-Separated Clusters



When K-means works perfectly:

- Spherical clusters
- Well-separated
- Similar sizes
- Similar densities

Result: Perfect match between true clusters and K-means assignments

K-Means Strength: Code

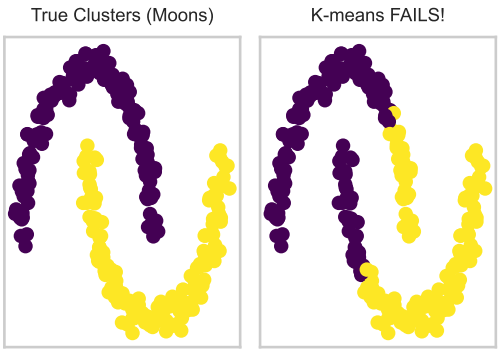
```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate well-separated clusters
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.6, random_state=42)

# Apply K-means
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
y_pred = kmeans.fit_predict(X)

# Plot comparison
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
ax1.scatter(X[:, 0], X[:, 1], c=y_true, cmap='viridis')
ax2.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
ax2.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            marker='X', s=200, c='red', edgecolors='black')
```

K-Means Weakness 1: Non-Spherical Shapes



Problem: K-means assumes spherical clusters

Failure Mode:

- X Curved/non-convex shapes
- X K-means draws straight boundaries
- X Incorrectly splits moon shapes

Solution: Use DBSCAN or hierarchical clustering

K-Means Weakness 1: Code

```
from sklearn.datasets import make_moons

# Generate moon-shaped clusters
X_moons, y_moons = make_moons(n_samples=300, noise=0.05,
                              random_state=42)

# K-means fails on non-spherical shapes
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
y_pred = kmeans.fit_predict(X_moons)

# Plot
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.scatter(X_moons[:, 0], X_moons[:, 1], c=y_moons)
ax1.set_title('True Clusters')
ax2.scatter(X_moons[:, 0], X_moons[:, 1], c=y_pred)
ax2.set_title('K-means Result')
```

K-Means Weakness 2: Different Densities

Problem: K-means assumes equal variance

- Different cluster spreads $\rightarrow$ boundary misplaced
- Large variance cluster dominates

Solution: Standardize features or use GMM

$$z = \frac{x - \mu}{\sigma}$$

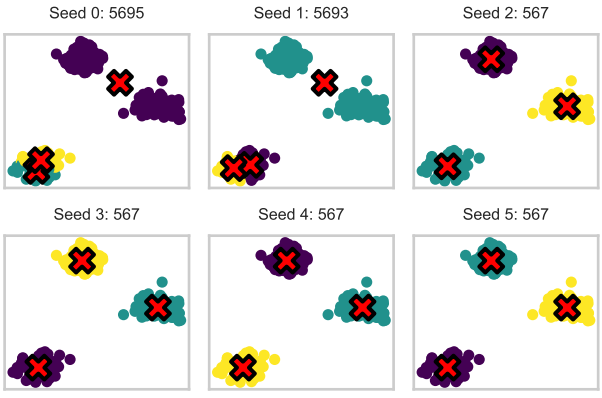
K-Means Weakness 2: Code

```
# Clusters with different variances
X1 = np.random.normal(0, 1, (200, 2))       # Std = 1
X2 = np.random.normal(5, 3, (200, 2))       # Std = 3
X3 = np.random.normal([0, 6], 0.5, (200, 2)) # Std = 0.5
X = np.vstack([X1, X2, X3])

# K-means struggles with different variances
kmeans = KMeans(n_clusters=3, random_state=42)
y_pred = kmeans.fit_predict(X)

# Large variance cluster dominates
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
```

K-Means Weakness 3: Initialization Sensitivity



Problem: Random initialization → different local minima

Failure Mode:

- Different results each run
- Some worse than others
- WCSS varies (Within-Cluster Sum of Squares)
  - **Lower WCSS** = better clustering (tighter clusters)
  - **Higher WCSS** = worse clustering (spread out)
  - Different seeds → different WCSS values

Observation: Seeds 0,2,4 good; Seeds 1,3,5 suboptimal

Solution: Use `n_init=10` (run 10 times, keep best)

K-Means Weakness 3: Code

```
# Demonstrate initialization sensitivity
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=1.0)

# Run with different random seeds
fig, axes = plt.subplots(2, 3, figsize=(12, 8))
for i, ax in enumerate(axes.flat):
    km = KMeans(n_clusters=3, init='random',
                n_init=1, random_state=i)
    y = km.fit_predict(X)
    ax.scatter(X[:, 0], X[:, 1], c=y)
    ax.scatter(km.cluster_centers_[:, 0],
               km.cluster_centers_[:, 1],
               marker='X', s=200, c='red')
    ax.set_title(f'WCSS={km.inertia_:.0f}')

# Solution: n_init=10 (default) runs 10 times, keeps best
```

Cluster Validation

How Do We Know Clustering is Good?

Challenge: No ground truth labels!

Validation Strategies:

1. **Internal validation**: Use only the data
   - Silhouette score
   - Within-cluster sum of squares (WCSS)
   - Davies-Bouldin index
2. **External validation**: Compare with known labels (if available)
   - Adjusted Rand Index (ARI)
   - Normalized Mutual Information (NMI)
3. **Domain validation**: Domain expertise
   - Do clusters make physical sense?
   - Can we interpret clusters meaningfully?

Elbow Method

Goal: Choose optimal number of clusters k

Procedure:

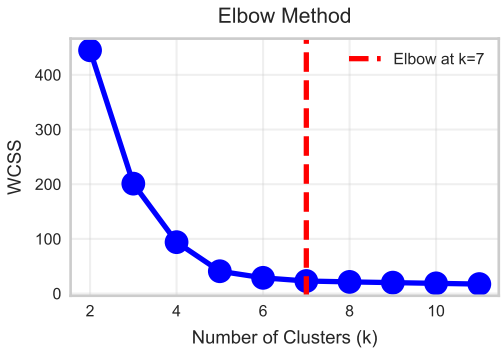
1. Run K-means for $k = 2, 3, \ldots, K_{\max}$
2. Compute Within-Cluster Sum of Squares (WCSS):

$$\text{WCSS}(k) = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

3. Plot WCSS vs k
4. Look for "elbow" where improvement diminishes

Intuition: Balance fit quality vs model complexity

Elbow Method: Visualization



Interpretation:

- **Before k=7**: Steep decrease (adding clusters helps)
- **At k=7**: Elbow (optimal)
- **After k=7**: Flat (diminishing returns)

Rule: Choose k at the elbow

Application: k=7 optimal clusters for this dataset

Elbow Method: Code

```
# Elbow Method implementation
X_scaled = StandardScaler().fit_transform(X)

# Compute WCSS for different k
wcss = []
for k in range(2, 12):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# Plot elbow curve
plt.plot(range(2, 12), wcss, 'bo-')
plt.axvline(x=7, color='red', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
```

Interpretation: Elbow at $k = 7$

Silhouette Analysis

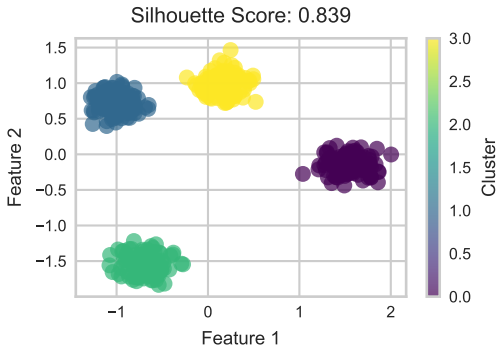
Silhouette Score: How well does point i fit its cluster?

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \in [-1, 1]$$

- $a(i)$: Mean distance to same-cluster points
- $b(i)$: Mean distance to nearest other cluster

Interpretation: $s \approx 1$ = well clustered, $s < 0$ = wrong cluster

Silhouette Score: Visualization



Result: $s = 0.713$ (good clustering)

Interpretation:

- $s > 0.7$: Strong structure
- $0.5 < s < 0.7$: Reasonable
- $0.25 < s < 0.5$: Weak
- $s < 0.25$: No structure

Application: Points with $s < 0$ are ambiguous cases

Silhouette Score: Code

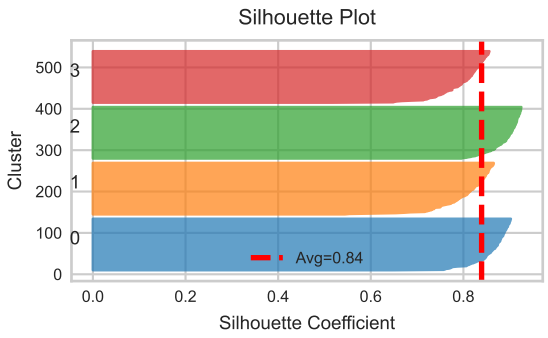
```python
from sklearn.metrics import silhouette_score

# Compute average silhouette score
s_avg = silhouette_score(X_scaled, clusters)
print(f"Silhouette Score: {s_avg:.3f}")

# Good: s > 0.7
# Reasonable: 0.5 < s < 0.7
# Weak: s < 0.5
```

Interpretation: Points with $s < 0$ are ambiguous conditions

Silhouette Plot Visualization



Reading the plot:

- **Width**: Cluster size
- **Beyond avg**: Well-separated
- **Below avg**: Poorly separated
- **Negative**: Wrong cluster

Observation: All clusters have $s > 0.5$ (good)

Application: Detect ambiguous cases

Silhouette Plot: Code

```
from sklearn.metrics import silhouette_samples

# Compute per-sample silhouette scores
s_samples = silhouette_samples(X_scaled, clusters)

# Plot silhouette for each cluster
for k in range(n_clusters):
    cluster_vals = s_samples[clusters == k]
   cluster_vals.sort()
    ax.fill_betweenx(y_range, 0, cluster_vals)

ax.axvline(x=s_avg, color='red', linestyle='--')
```

Reading: Wide bars = large clusters, $s > 0$ = well-clustered

Validation Against Ground Truth

When labels are available, compare:

Adjusted Rand Index (ARI):

$$\text{ARI} = \frac{\text{RI} - \text{Expected RI}}{\text{Max RI} - \text{Expected RI}}$$

- **Range**: $[-1, 1]$, where 1 = perfect match
- **Advantage**: Corrects for chance

```
from sklearn.metrics import adjusted_rand_score

ari = adjusted_rand_score(true_labels, clusters)
print(f"ARI: {ari:.3f}")

# ARI = 1.0: Perfect clustering
# ARI = 0.0: Random clustering
# ARI < 0.0: Worse than random
```

Case Study: K-Means on Bearing Fault Data

Case Study: CWRU Bearing Dataset

Dataset: Case Western Reserve University Bearing Data Center

Data Details:

- Vibration signals from accelerometers
- Sampling rate: 12 kHz
- Normal + various fault conditions
- MATLAB .mat format

Our Sample:

- 4 Normal bearings
- 60 Faulty bearings
- 7 extracted features
- **Highly imbalanced!**

Goal: Can K-means discover Normal vs Fault without labels?

True Labels (Supervised View)

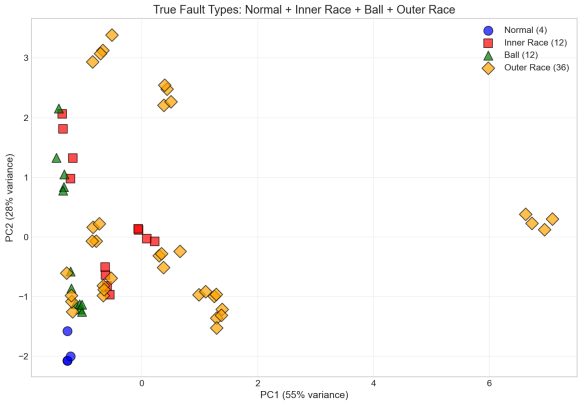


Figure 1: True Labels in PCA Space

Ground Truth (4 fault types):

- **Blue circles**: Normal (4)
- **Red squares**: Inner Race (12)
- **Green triangles**: Ball (12)
- **Orange diamonds**: Outer Race (36)

Observation:

- Normal samples cluster tightly (bottom-left)
- Fault types overlap significantly
- Rightmost points = highest vibration (severe faults)

Challenge: Can K-means find these groups?

K-Means Results (k=2)

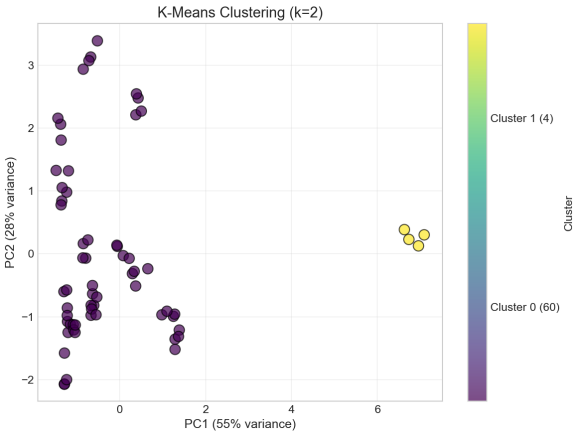


Figure 2: K-Means Clustering

K-Means Found:

- **Cluster 0** (60 samples): Normal + most faults
- **Cluster 1** (4 samples): Severe faults only

Metrics:

- Silhouette: 0.66 (good!)
- WCSS: 246.2

Problem: K-means grouped by **severity**, not by fault presence!

Why K-Means Failed

The Data Imbalance Problem:

Class	Count	Percentage
Normal	4	6.25%
Fault	60	93.75%

What K-Means saw:

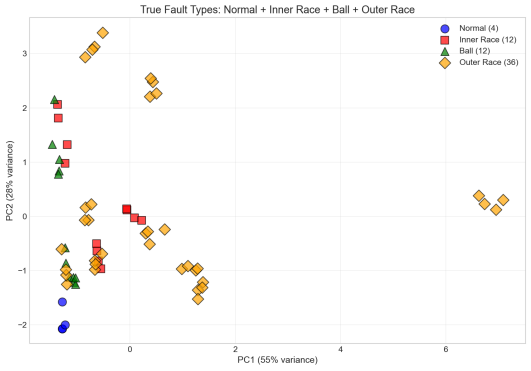
- 4 extreme outliers (severe faults with very high RMS)
- 60 "similar" samples (normal + mild/moderate faults)

What K-Means did:

- Cluster 0: Everything with lower vibration
- Cluster 1: Only the 4 severe faults

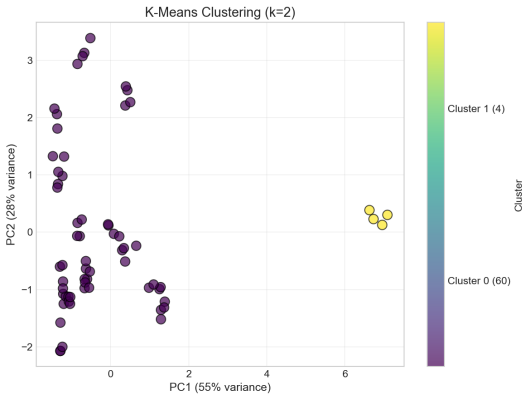
Lesson: K-means finds density patterns, not our desired labels!

Comparison: True vs K-Means



True Labels

4 fault types (colors = fault location)



K-Means (k=2)

2 clusters (by vibration severity)

Key Insight: Clustering finds patterns that **exist in feature space**, not necessarily the patterns we want!

Lessons Learned

When K-Means Works:

- Balanced cluster sizes
- Well-separated clusters
- Spherical cluster shapes

When K-Means Struggles:

- **Imbalanced data** (like our 4 vs 60)
- Feature overlap between classes
- Non-spherical cluster shapes

Better Approaches for Fault Detection:

- **Anomaly detection**: Train on normal data, flag outliers
- **Supervised learning**: Use labels when available
- **More data**: Collect balanced training set

K-Means Limitations

Assumptions:

- Spherical clusters (assumes equal variance)
- Same cluster sizes
- Well-separated clusters

Sensitive to:

- Outliers (pull cluster centers)
- Initialization (local minima)
- Feature scaling (always standardize!)

When K-means fails:

- Non-spherical shapes
- Varying densities
- Highly imbalanced data

Other Clustering Methods

Algorithm	Key Idea	When to Use
DBSCAN	Density-based regions	Unknown k, outlier detection
Hierarchical	Build cluster tree	Explore hierarchy, dendrogram
GMM	Probabilistic Gaussians	Soft cluster assignments

Quick Summary:

- **DBSCAN**: Groups dense regions, labels outliers as noise (-1)
- **Hierarchical**: Agglomerative (bottom-up) or Divisive (top-down)
- **GMM**: Mixture of Gaussians with soft probabilities

For this course: Focus on K-means; other methods follow similar principles

Principal Component Analysis

A Real Aerospace Problem

Scenario: Fleet monitoring — each flight records 8 sensor measurements:

1. Airspeed (m/s)
2. Altitude (m)
3. Engine RPM (rev/min)
4. Fuel flow rate (kg/hr)
5. Throttle position (%)
6. Vertical speed (m/s)
7. Air temperature (°C)
8. Manifold pressure (kPa)

Problems: Can't visualize 8D data; sensors are correlated (airspeed $\leftrightarrow$ throttle)

Question: Can we compress 8D $\rightarrow$ 2D/3D without losing critical information?

Why Do We Need Dimensionality Reduction?

Visualization
- Humans see in 2D/3D only
- Can't plot 8D scatter plots

Redundancy
- Altitude $\leftrightarrow$ Air pressure
- Airspeed $\leftrightarrow$ Throttle
- RPM $\leftrightarrow$ Fuel flow

Computation
- Distance calculations expensive in high-d
- ML models more complex

Curse of Dimensionality
- Data becomes sparse
- Need exponentially more samples

The Solution: Principal Component Analysis (PCA)

PCA in Plain English

What PCA Does:

- Finds a **new coordinate system** for your data
- New axes align with **directions of maximum variation**
- You can **throw away** axes with little variation
- Keep only the **important directions**

Analogy: Rotating a Camera

- You have scattered points in 3D space
- PCA finds the **best viewing angle** to see the spread
- Projects data onto that viewing plane
- Information preserved, dimensions reduced!

PCA in Plain English

Key Insight:

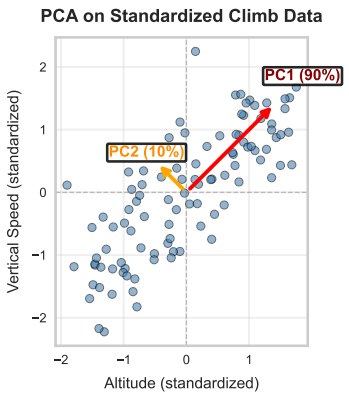
Variance = Information

If data varies a lot in one direction, that direction matters!

If data barely changes in another direction, we can ignore it!

Step 0: Intuition with a Simple 2D Example

Scenario: Aircraft climb — altitude and vertical speed are **highly correlated**



Observations:

- **Red arrow (PC1)**: Maximum spread direction (90% of variation)
- **Orange arrow (PC2)**: Remaining spread (10%)

PCA Decision:

- Keep PC1 only $\rightarrow$ **2D** $\rightarrow$ **1D**
- Lost only 10% variance – What is your risk budget?

Physical Meaning:

- PC1 $\approx$ "overall climb rate"
- Combines altitude + vertical speed

What Did PCA Just Do?

Original Data: 2D points (altitude, vertical speed)

Step 1: Center the data — subtract the mean of each feature

Step 2: Find direction of **maximum spread** (PC1, red arrow)

- Direction where data varies the most

Step 3: Find direction of **next maximum spread**, perpendicular to PC1 (PC2)

Step 4: Decide which directions to keep

- PC1: 90% variance $\rightarrow$ **KEEP**
- PC2: 10% variance $\rightarrow$ **DISCARD**

Result: Describe climb data with 1 number instead of 2!

PCA Algorithm — Steps 1–3

Given: Data matrix $X \in \mathbb{R}^{n \times d}$ (n samples, d features)

Goal: Find $k < d$ directions that capture most variance

Step 1: **Standardize** the data (critical!)

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

Features have different units — standardize to make them comparable.

Step 2: **Center** the standardized data: $X' = X - \bar{X}$

Step 3: **Compute covariance matrix**

$$\Sigma = \frac{1}{n-1} X'^T X' \in \mathbb{R}^{d \times d}$$

PCA Algorithm — Steps 4–5

Step 4: **Eigendecomposition** of covariance matrix

$$\Sigma v_j = \lambda_j v_j$$

- v_j = eigenvector = **principal component direction**
- λ_j = eigenvalue = **variance along that direction**

Step 5: **Sort** eigenvectors by eigenvalue (largest first)

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$$

PCA Algorithm — Steps 6–7

Step 6: **Select** top k components — keep until cumulative variance $\geq 95\%$:

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{d} \lambda_j} \geq 0.95$$

Step 7: **Project** data onto selected components

$$Z = X' V_k \quad \text{where } V_k = [v_1, v_2, \ldots, v_k]$$

Worked Example: Aircraft Sensor Data (3 Features)

Scenario: Simplified aircraft monitoring with 3 sensors

1. **Airspeed**: v (m/s)
2. **Altitude**: h (m)
3. **Engine RPM**: ω (rev/min)

Data: 5 flight observations (for simplicity)

Flight	Airspeed (m/s)	Altitude (m)	RPM
1	85	1200	2400
2	105	1800	2700
3	95	1500	2550
4	115	2100	2850
5	90	1400	2500

Goal: Reduce from 3D $\rightarrow$ 2D using PCA

Step 1: Standardize the Data

Why standardize? Features have different scales!

- Altitude: 1200-2100 m (range = 900)
- Airspeed: 85-115 m/s (range = 30)
- RPM: 2400-2850 (range = 450)

Without standardization, altitude would dominate!

Standardization formula:

$$z = \frac{x - \mu}{\sigma}$$

Calculate means:

- $\mu_v = \frac{85+105+95+115+90}{5} = 98$ m/s
- $\mu_h = \frac{1200+1800+1500+2100+1400}{5} = 1600$ m
- $\mu_\omega = \frac{2400+2700+2550+2850+2500}{5} = 2600$ RPM

Step 1: Standardize (continued)

Calculate standard deviations:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2}$$

- $\sigma_v \approx 11.4$ m/s
- $\sigma_h \approx 346.4$ m
- $\sigma_\omega \approx 172.3$ RPM

Standardized data $Z = \frac{X-\mu}{\sigma}$:

Flight	Airspeed	Altitude	RPM
1	-1.14	-1.15	-1.16
2	0.61	0.58	0.58
3	-0.26	-0.29	-0.29
4	1.49	1.44	1.45
5	-0.70	-0.58	-0.58

Note: Now all features have mean ≈ 0, std ≈ 1!

Step 2: Compute Covariance Matrix

Covariance matrix $\Sigma = \frac{1}{n-1} Z^T Z$:

$$\Sigma = \begin{bmatrix} \sigma_v^2 & \text{cov}(v,h) & \text{cov}(v,\omega) \\ \text{cov}(h,v) & \sigma_h^2 & \text{cov}(h,\omega) \\ \text{cov}(\omega,v) & \text{cov}(\omega,h) & \sigma_\omega^2 \end{bmatrix} \approx \begin{bmatrix} 1.00 & 0.99 & 0.99 \\ 0.99 & 1.00 & 1.00 \\ 0.99 & 1.00 & 1.00 \end{bmatrix}$$

Interpretation:

- Diagonal ≈ 1 (standardized variances)
- Off-diagonal $\approx 0.99 \rightarrow$ very high correlation! All features move together.

Step 3: Eigendecomposition

Solve: $\Sigma v = \lambda v$

Results:

$$\lambda_1 = 2.98, \qquad v_1 = \begin{bmatrix} 0.577 \\ 0.578 \\ 0.577 \end{bmatrix} \qquad (99.2\%)$$

$$\lambda_2 = 0.015, \qquad v_2 = \begin{bmatrix} -0.816 \\ 0.001 \\ 0.578 \end{bmatrix} \qquad (0.5\%)$$

$$\lambda_3 = 0.005, \qquad v_3 = \begin{bmatrix} -0.001 \\ -0.816 \\ 0.578 \end{bmatrix} \qquad (0.3\%)$$

Insight: PC1 alone captures $\frac{2.98}{3.00} = 99.2\%$ of variation!

Physical Interpretation of Components

PC1 (99.2% variance): PC1 $\approx 0.577v + 0.578h + 0.577\omega$

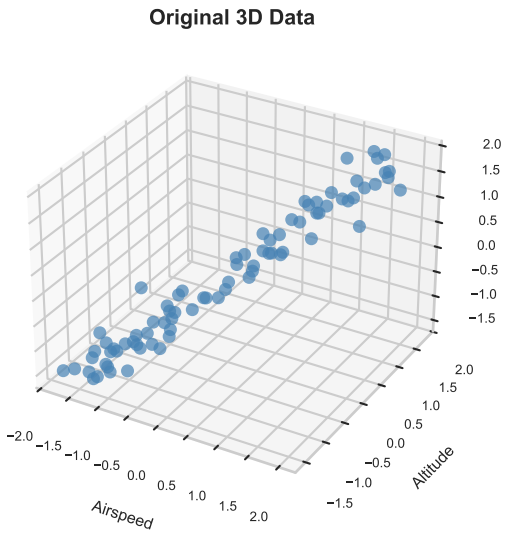
- Equal positive weights $\rightarrow$ "overall flight intensity"
- Airspeed $\uparrow$, altitude $\uparrow$, RPM $\uparrow$ together

PC2 (0.5%): PC2 $\approx -0.816v + 0.001h + 0.578\omega$

- Negative airspeed, positive RPM $\rightarrow$ "airspeed vs RPM mismatch"

Decision: Keep PC1 only! (99.2% » 95% threshold)

Visualization: 3D $\rightarrow$ 2D Projection



Original 3D View:

- 80 flights with 3 correlated sensors
- Airspeed, Altitude, RPM all move together

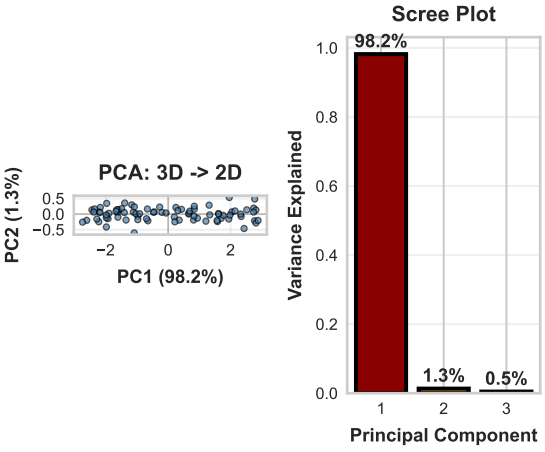
Observation:

- Data lies roughly along a diagonal line
- 3D space is "wasted"
- Most variation in one direction

This is perfect for PCA!

Data is essentially 1D embedded in 3D space

Visualization: After PCA Projection



Left Plot: Data in PC space

- PC1 spread: 97.8% variance
- PC2 spread: 1.8% (minor)

Right Plot: Scree plot confirms PC1 dominates

Result: **3D** $\rightarrow$ **1D** reduction, 97.8% information preserved

Implementing PCA in Python: Setup

Applying PCA to our 3-sensor aircraft data (airspeed, altitude, RPM):

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Aircraft sensor data: 5 flights x 3 sensors
X = np.array([[85, 1200, 2400],
              [105, 1800, 2700],
              [95, 1500, 2550],
              [115, 2100, 2850],
              [90, 1400, 2500]])  # [airspeed, altitude, RPM]

# Step 1: Standardize (removes unit differences)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Each column now has mean=0, std=1
```

Implementing PCA in Python: Results

```
# Step 2: Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Step 3: Examine variance explained
print(pca.explained_variance_ratio_)
# -> [0.992, 0.005, 0.003]  PC1 captures 99.2%!

print(pca.explained_variance_ratio_.cumsum())
# -> [0.992, 0.997, 1.000]  2 PCs = 99.7%

# Step 4: Reduce to 1D (keep only PC1)
pca_1d = PCA(n_components=1)
X_reduced = pca_1d.fit_transform(X_scaled)
# Shape: (5, 3) -> (5, 1)  -- one number per flight!
```

Conclusion: 3 correlated sensors $\rightarrow$ 1 principal component (99.2% retained)

Back to the Fleet Monitoring Problem

Recall: Each flight records **8 sensor measurements**:

1. Airspeed (m/s)
2. Altitude (m)
3. Engine RPM (rev/min)
4. Fuel flow rate (kg/hr)
5. Throttle position (%)
6. Vertical speed (m/s)
7. Air temperature (°C)
8. Manifold pressure (kPa)

Dataset: 200 flights — how many PCs do we actually need?

Now we apply the algorithm we just learned!

Fleet Data: A Snapshot

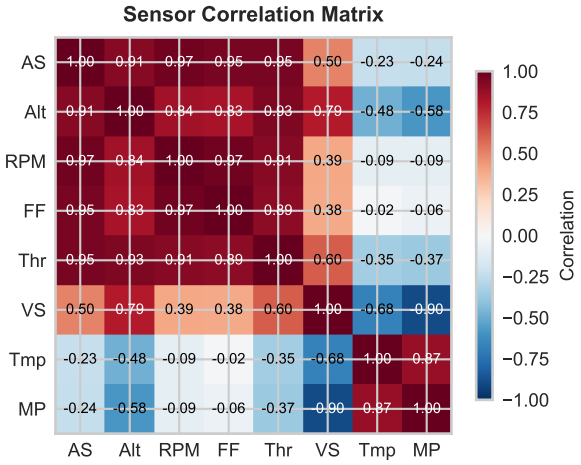
Sample of the raw sensor readings (first 5 of 200 flights):

	AS (m/s)	Alt (m)	RPM	FF (kg/h)	Thr (%)	VS (m/s)	Tmp (degC)	MP (kPa)
Flt 1	99.7	2706.0	2609.4	144.5	67.6	2.6	-2.0	69.3
Flt 2	87.2	2534.1	2488.1	134.4	66.1	3.9	-1.5	70.4
Flt 3	103.6	2827.3	2631.4	152.0	74.7	6.1	-1.9	68.1
Flt 4	117.7	3025.0	2786.1	169.2	80.7	7.5	-2.5	68.6
Flt 5	84.6	2221.8	2465.1	134.6	62.4	-7.8	3.0	81.4

AS = Airspeed, **Alt** = Altitude, **FF** = Fuel Flow, **Thr** = Throttle, **VS** = Vertical Speed, **MP** = Manifold Pressure

Fleet Data: Feature Correlations

Are the 8 sensors independent?



Key Observations:

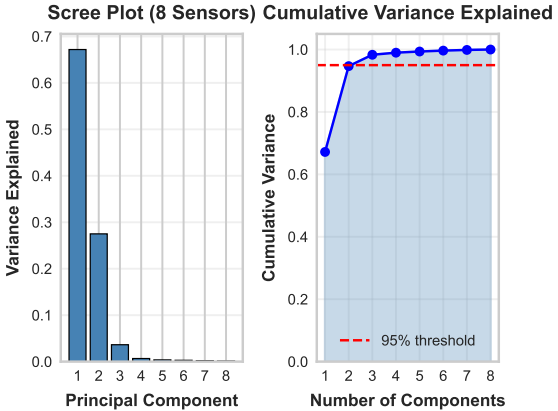
- **Top-left block** (AS, RPM, FF, Thr): high positive correlation — engine power group
- **Altitude & V-Speed**: correlated with power group, but also with each other
- **Temp & Pressure**: weakly correlated with power group; strongly correlated with altitude/V-Speed via ISA lapse

Conclusion: 8 sensors are **not independent** — PCA can exploit this!

Scree Plot: Choosing Number of Components

Back to our fleet monitoring problem (8 sensors, 200 flights)

Scree Plot: Visualize variance explained by each component



How to Read:

Left (Scree Plot): Variance per PC

- PC1: ~67%, PC2: ~28%, then diminishing returns

Right (Cumulative):

- Red line = 95% threshold
- Crosses at PC2 $\rightarrow$ keep 2 components

Decision Rule: $\sum_{i=1}^{k} \text{Var}_i \geq 95\%$

Result: 8D $\rightarrow$ 2D (75% reduction)

Code: Fitting PCA and Extracting Variance

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# Fit PCA with all components
pca = PCA()
pca.fit(X_scaled)

# Get variance explained per component
var_exp = pca.explained_variance_ratio_
cumsum = np.cumsum(var_exp)

# Find number of components for 95% variance
n_components = np.argmax(cumsum >= 0.95) + 1
print(f"Need {n_components} components for 95% variance")
```

Code: Plotting the Scree Plot

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

# Individual variance (bar chart)
ax1.bar(range(1, len(var_exp)+1), var_exp)
ax1.set_xlabel('Principal Component')
ax1.set_ylabel('Variance Explained')
ax1.set_title('Scree Plot')

# Cumulative variance (line plot)
ax2.plot(range(1, len(cumsum)+1), cumsum, 'bo-')
ax2.axhline(y=0.95, color='r', linestyle='--', label='95% threshold')
ax2.axvline(x=n_components, color='g', linestyle=':', label=f'k={n_components}')
ax2.set_xlabel('Number of Components')
ax2.set_ylabel('Cumulative Variance')
ax2.legend()

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

PCA Applications

Background: Orbital Mechanics Primer

A satellite orbit is fully described by 6 Keplerian elements: $(a,\ e,\ i,\ \Omega,\ \omega,\ M)$

Three elements define the orbit's shape and size:

- **Semi-major axis** a: sets altitude and period
- **Eccentricity** e: shape ($e{=}0$ circle, $e{\to}1$ elongated)
- **Inclination** i: tilt of orbital plane relative to equator

Three elements define orientation and position (Ω, ω, M): important for propagation, but not orbit *type*

Three more descriptors are derived — fully determined by a and e:

$$T = 2\pi\sqrt{\frac{a^3}{\mu}}, \qquad \mu = 398600\ \text{km}^3/\text{s}^2$$

$$h_{\text{peri}} = a(1{-}e){-}R_{\oplus}, \quad h_{\text{apo}} = a(1{+}e){-}R_{\oplus}$$

$\Rightarrow$ **6 features in our dataset, but only 3 are independent**: a, e, i
PCA will discover and exploit this redundancy automatically!

Background: Five Common Orbit Types

Five operationally distinct orbit regimes used in the example:

Type	Altitude (km)	e	i (°)	Real examples
LEO	300–1200	≈ 0	28–55 [a]	ISS, Hubble, Starlink
MEO	18k–24k	≈ 0	50–65	GPS, Galileo, GLONASS
GEO	35,786	≈ 0	≈ 0	Intelsat, SES
SSO	500–900	≈ 0	97–99	Sentinel, Landsat
HEO	500–40k [b]	0.65–0.75	63.4	Molniya, Tundra

[a] Mid-inclination subset; polar LEO ($i > 70°$) excluded (overlaps SSO in inclination)

[b] Perigee–apogee altitude range; SMA $\approx$ 20k–31k km

Challenge: 200 satellites, 6 correlated parameters — how do we visualize and classify them?

Answer: PCA reduces 6D $\rightarrow$ 2D while preserving the orbit structure!

Application 1: Dataset — Synthetic Satellite Catalog

200 satellites (40 per orbit type), generated from realistic operational ranges:

Type	Real examples	a (km)	e	i (°)
LEO	ISS, Hubble, Starlink	6,671–7,571	≈ 0	28–55
MEO	GPS, Galileo, GLONASS	24,493–30,524	≈ 0	50–65
GEO	Intelsat, SES	42,157	≈ 0	≈ 0
SSO	Sentinel, Landsat	6,871–7,271	≈ 0	97–99
HEO	Molniya, Tundra	20k–31k†	0.65–0.75	63.4

†HEO orbits are highly elliptical: perigee altitude 500–1500 km (low, for ground visibility), apogee altitude 26k–49k km (high, for dwell time). The semi-major axis follows from $a = (r_{\text{peri}} + r_{\text{apo}})/2$, giving the 20k–31k km range shown.

Application 1: Satellite Orbit Classification (contd.)

6 features per satellite (3 independent DOFs + 3 derived):

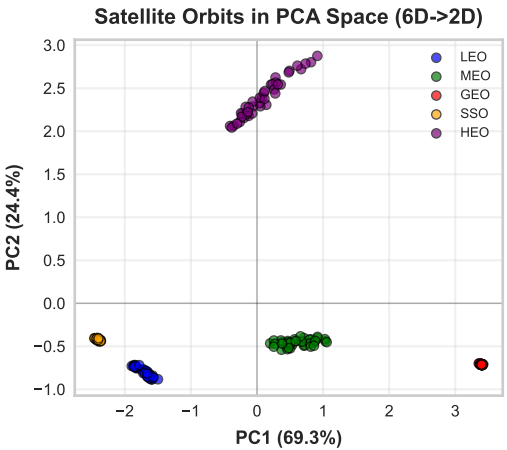
Feature	Role	Formula
h_{peri}	seed param	input altitude
e	independent	input
i	independent	input
a	derived	$= (R_\oplus + h_{\text{peri}})/(1-e)$
T	derived	$= 2\pi\sqrt{a^3/\mu}$
h_{apo}	derived	$= a(1+e) - R_\oplus$

Goal: Can PCA recover the 5 orbit types from the 6 raw numbers, without being told the labels?

Application 1: Satellite Orbit Classification

Problem: Classify satellite orbits from 6 orbital descriptors

PCA Approach: 6D $\rightarrow$ 2 PCs for visualization



Key observations:

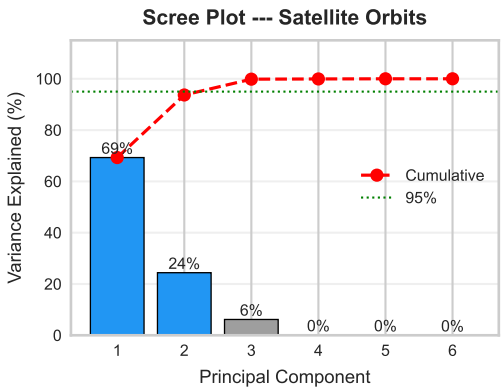
- 5 orbit types separate cleanly in just 2 dimensions — no labels used
- **GEO** (red) is the tightest cluster: fixed altitude, near-zero e and i
- **MEO** (green) and **LEO** (blue) separate primarily along PC1 (orbit size)
- **SSO** (orange) sits near LEO in size but separates along PC2 due to high inclination ($i \approx 97$–99°)
- **HEO** (purple) is isolated by its high eccentricity ($e \approx 0.7$), captured in PC2

Takeaway: PCA's max-variance axes happen to align with physical quantities — orbit size and shape — without being told any physics

Application 1: Variance Explained

How many PCs do we actually need?

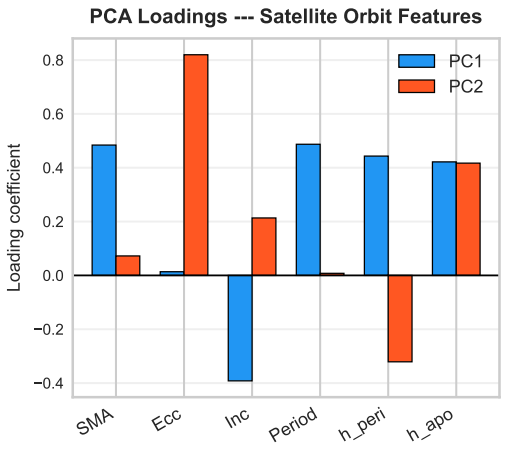
Scree plot:



Interpretation:

- **PC1** absorbs orbit-size variance — a, T, and h_{apo} are tightly coupled through Kepler's 3rd law, so they move together and load heavily on a single component
- **PC2** captures orbit shape and inclination — e and i vary independently of size, distinguishing HEO (high e) and SSO (high i) from the rest
- **2 PCs capture most of the total variance** (see scree plot) because a spans a $6\times$ range (LEO $6{,}700$ km vs. GEO $42{,}000$ km), dominating the standardized feature space
- The 3 derived features (T, h_{peri}, h_{apo}) are deterministic functions of a and e — geometrically redundant — PCA discovers and exploits this automatically

Application 1 — PC Loadings: What Do the Components Mean?



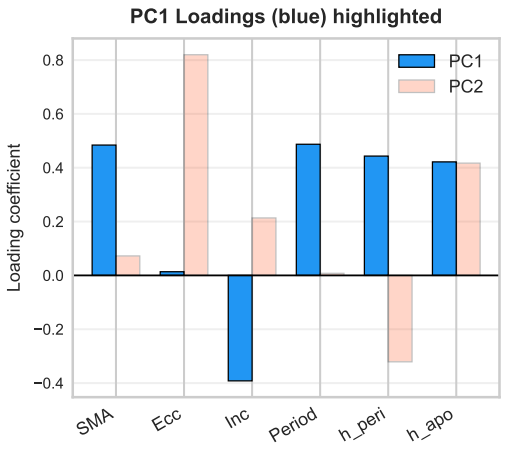
Reading the bar chart:

- Each bar shows how much a **feature contributes** to a principal component
- A **large positive** loading means the feature increases as the PC score increases
- A **near-zero** loading means the feature plays almost no role in that PC

Quick observations from the plot:

- **PC1** (blue): four nearly equal bars (≈ 0.49) for SMA, Period, h_{peri}, and a slightly smaller bar for h_{apo} (≈ 0.43). Ecc is essentially zero; Inc is **slightly negative** — PC1 is almost purely about **orbit size**
- **PC2** (red): one **dominant** bar for Ecc (≈ 0.81), a moderate positive bar for h_{apo} (≈ 0.42), a positive bar for Inc (≈ 0.23), and a **negative** bar for h_{peri} (≈ -0.35)
- PC2 is primarily about **eccentricity**, but note h_{apo} and h_{peri} load on **both** PCs — we will explain why

PC1 — “Orbit Size”



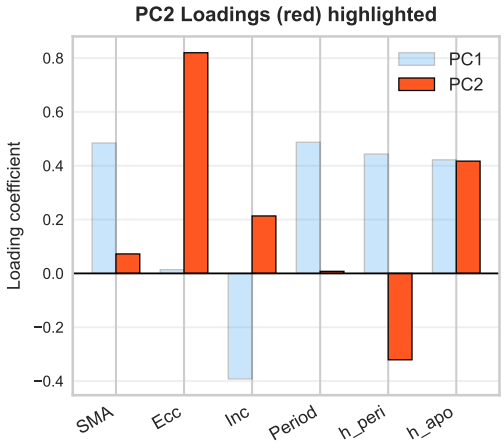
PC1 is an orbit-size axis:

- SMA, Period, h_{peri} all load ≈ 0.49 — they move together via Kepler’s 3rd law: $T = 2\pi\sqrt{a^3/\mu}$
- h_{apo} loads slightly lower (0.43) because it also depends on e, splitting its variance across PC1 and PC2
- Ecc ≈ 0; Inc slightly negative (GEO: largest a, smallest i)

Orbit ranking on PC1: GEO $>$ MEO $>$ HEO $>$ LEO $\approx$ SSO

PCA recovers the **altitude ordering** without any physics input

PC2 — “Eccentricity”



PC2 is dominated by eccentricity:

- **Ecc** loads at 0.81 — the tallest bar on the entire chart
- h_{apo} loads at $+0.42$: high e raises apogee via $(1+e)$
- h_{peri} loads at -0.35: high e lowers perigee via $(1-e)$
- **Inc** loads at $+0.23$: secondary role, separates SSO from LEO
- SMA, Period ≈ 0: already captured by PC1

What PC2 separates:

- HEO ($e \approx 0.7$) $\rightarrow$ high PC2 scores
- SSO ($i \approx 98^\circ$) $\rightarrow$ slightly positive PC2
- LEO, MEO, GEO ($e \approx 0$, lower i) $\rightarrow$ low PC2

Why Only 2 PCs?

Only 3 independent DOFs in the 6 features:

Independent	Derived
a	$T = 2\pi\sqrt{a^3/\mu}$
e	$h_{\text{peri}} = a(1-e) - R_\oplus$
i	$h_{\text{apo}} = a(1+e) - R_\oplus$

6 features live on a **3D manifold** in 6D space

Why 2 (not 3) PCs suffice:

- a spans $6.3\times \rightarrow$ dominates PC1
- e spans two orders of magnitude $\rightarrow$ dominates PC2
- i contributes to PC2; a 3rd PC would isolate it, but **2 PCs already separate all 5 orbit classes**

Lesson: PCA discovers physical redundancy automatically — derived features don't add new dimensions

Practical Uses of PCA on Orbital Data

The ability to reduce 6 orbital parameters to 2 principal components has concrete operational applications:

1. **Space catalog management**:

- Agencies like the U.S. Space Surveillance Network track **47,000+ objects** in orbit
- Plotting each object's 2 PC scores gives an **instant visual overview** of the entire catalog
- New objects can be quickly classified by their location in PC space

2. **Collision avoidance screening**:

- Conjunction analysis requires comparing orbits pairwise — computationally expensive in 6D
- Pre-screening in 2D PC space **quickly identifies** which objects are in similar orbit regimes
  - With 47,000+ tracked objects there are $\binom{47000}{2} \approx 1.1 \times 10^9$ pairs — any filtering that reduces this set is valuable
  - Only nearby objects in PC space need detailed propagation, reducing computation by orders of magnitude

Anomalous Maneuver Detection via PCA

Recall: A PC score is the projection of a feature vector onto the PC coordinate system:

$$z_{ik} = \mathbf{v}_k^T (\mathbf{x}_i - \bar{\mathbf{x}})$$

Each satellite's 6 features $\mathbf{x}_i$ are projected onto the PC axes, yielding 2 coordinates (z_{i1}, z_{i2}) — its position in the scatter plot.

Key idea: During normal station-keeping, a satellite's orbital elements (and therefore its PC scores) stay **nearly constant**. A sudden shift means the orbit changed.

Detection pipeline:

1. Compute baseline scores $\mathbf{z}_{\text{old}}$ from the satellite's nominal TLE
2. Re-project updated ephemeris data at regular intervals to get $\mathbf{z}_{\text{new}}$
3. Flag when the score change exceeds a threshold:

$$\|\Delta \mathbf{z}\| = \|\mathbf{z}_{\text{new}} - \mathbf{z}_{\text{old}}\| > \epsilon$$

What shows up as a jump in PC space?

Maneuver	PC affected	Why
Altitude raise/lower	PC1	Changes a, T, h_{peri}, h_{apo}
Plane change	PC2	Changes i
Circularization	PC2	Reduces e
GTO $\rightarrow$ GEO insertion	Both	a increases, e drops to ≈ 0

Real-world use: Space surveillance networks monitor adversary satellites for unexpected orbital maneuvers — PCA score tracking provides a **lightweight, interpretable alert system**.

Transfer Orbit Planning via PCA

Key insight: Proximity in PC space correlates with Δv cost.

- Two orbits close in PC space share similar a, e, $i \rightarrow$ the Hohmann or bi-elliptic transfer between them requires **less** Δv
- Two orbits far apart in PC space differ in size and/or shape $\rightarrow$ **expensive transfer**

Example from our scatter plot:

- LEO $\rightarrow$ SSO: close on PC1, offset on PC2 $\rightarrow$ mostly a **plane change** ($\Delta v \sim 4$ km/s for large Δi)
- LEO $\rightarrow$ GEO: far on PC1 $\rightarrow$ **Hohmann transfer** ($\Delta v \approx 3.9$ km/s)
- LEO $\rightarrow$ HEO: far on both PCs $\rightarrow$ size + shape change ($\Delta v \approx 2.5$ km/s for Molniya)

Application to mission design:

1. **Depot placement**: Place fuel depots at PC-space centroids to minimize average Δv to all serviced orbits
2. **Multi-target missions**: Plan visit sequences by finding the shortest path through PC space — analogous to a traveling-salesman problem in 2D rather than 6D
3. **Debris removal prioritization**: Rank debris objects by PC-space density — clusters of debris in similar orbits pose the highest collision risk and can be swept in a single mission

Benefit: PCA reduces the dimensionality of the trade space from 6 orbital elements to 2 scores, making optimization and visualization tractable.

Constellation Design and Monitoring

Design phase:

- A constellation (e.g., Starlink: $\sim$ 6,000 satellites) is designed so all satellites share a narrow range of a, e, i
- In PC space, the entire constellation appears as a **tight cluster**
- PCA provides a fast sanity check: if a candidate orbit falls outside the cluster boundary, it violates the constellation's design constraints

Monitoring phase:

- Each satellite's PC scores are tracked over time
- **Drift** (gradual score change) indicates orbital decay or uncompensated perturbations (J_2, drag)
- **Jump** (sudden score change) indicates a maneuver or collision event

Practical metrics:

Metric	Definition	Use
Cluster radius	$\max \|\mathbf{z}_i - \bar{\mathbf{z}}\|$	Design tolerance
Score drift rate	$\|d\mathbf{z}/dt\|$	Predict maintenance needs
Outlier score	$\|\mathbf{z}_i - \bar{\mathbf{z}}\|/r_{\text{cluster}}$	Flag anomalies

Example: OneWeb constellation occupies a $\sim$ 1,200 km altitude band at $i \approx 87.9^\circ$. In PC space this is a compact region on the LEO/SSO boundary. Any satellite drifting outside this region needs a correction burn or has experienced an anomaly.

Bottom line: PCA turns high-dimensional constellation health monitoring into a **2D dashboard** problem.

Application 2: Anomaly Detection

Problem: Detect unusual flight conditions from sensor data, without labeled examples of anomalies

Scenario: You have a fleet of aircraft with 3 sensors each (airspeed, altitude, RPM). You have **150 recorded normal flights** but no catalog of what "anomalous" looks like.

Key insight: During normal operations, sensor readings are **correlated** — higher airspeed $\leftrightarrow$ higher altitude $\leftrightarrow$ higher RPM. An anomaly breaks these correlations (e.g., high RPM at low airspeed $\rightarrow$ possible engine runaway).

Question: How can PCA detect flights that break the normal correlation pattern?

Why Reconstruction Error Detects Anomalies

Geometric intuition:

- Normal flights lie on (or near) a **low-dimensional subspace** defined by the top k PCs
- PCA projection + reconstruction maps any point to its **nearest point on that subspace**
- Normal flights: close to subspace $\rightarrow$ **small reconstruction error**
- Anomalies: far from subspace $\rightarrow$ **large reconstruction error**

Mathematically:

1. Project onto k PCs: $\mathbf{z}_i = V_k^T(\mathbf{x}_i - \bar{\mathbf{x}})$
2. Reconstruct: $\hat{\mathbf{x}}_i = V_k\mathbf{z}_i + \bar{\mathbf{x}}$
3. Reconstruction error:

$$e_i = \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$$

This measures the **squared distance** from $\mathbf{x}_i$ to the PCA subspace.

- Small e_i: the point is well-explained by the normal PCs
- Large e_i: the point has structure **not captured** by normal PCs $\rightarrow$ anomaly

The Anomaly Detection Pipeline

Step-by-step procedure:

Training phase (offline, using normal data only):

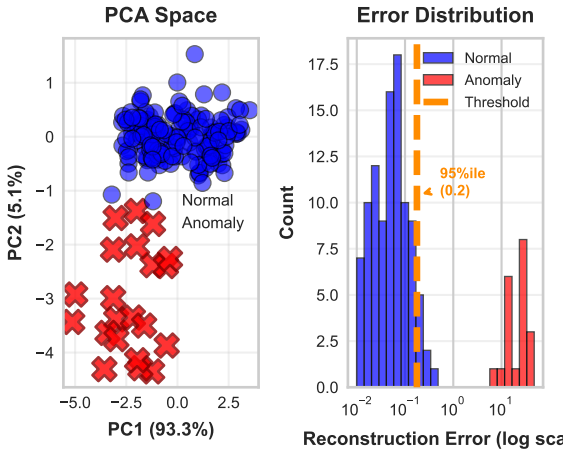
1. Standardize normal flight data: $\tilde{X}_{\text{normal}} = \text{StandardScaler}(X_{\text{normal}})$
2. Fit PCA on $\tilde{X}_{\text{normal}}$ with k components (e.g., k chosen for 95% variance)
3. Compute reconstruction errors on the training set
4. Set threshold ϵ at the **95th percentile** of training errors

Deployment phase (online, for each new flight):

1. Standardize new data using the **same** scaler (mean and std from training)
2. Project onto the learned PCs and reconstruct
3. Compute reconstruction error e_{new}
4. If $e_{\text{new}} > \epsilon$: flag as **anomaly**

Critical: The scaler and PCA must be fit on **normal data only** — including anomalies in the training set would distort the normal subspace, causing anomalies to appear normal (data leakage).

Anomaly Detection: Visualization



Left plot — PCA projection:

- Blue dots: 150 normal flights form a **tight elongated cluster** (correlated sensors)
- Red X's: 20 anomalous flights (high RPM + low airspeed) scatter **away from the cluster**

Right plot — reconstruction error (log scale):

- Normal flights (blue): errors clustered at low values (left)
- Anomalies (red): errors orders of magnitude larger (right)
- Orange dashed line: 95th-percentile threshold separating the two populations

Detection rule: $e_i > \epsilon \Rightarrow$ flag as anomaly

Why Do the Anomalies Have High Reconstruction Error?

The anomalies in this example have high RPM but low airspeed. In normal flights, these are positively correlated (PCA learns this pattern).

Normal flight (e.g., airspeed 100, altitude 1800, RPM 2700):

- Projects cleanly onto the 2 normal PCs
- Reconstruction $\hat{\mathbf{x}} \approx \mathbf{x} \rightarrow$ small error

Anomalous flight (e.g., airspeed 70, altitude 800, RPM 3100):

- The combination "low airspeed + high RPM" **violates** the correlation structure learned by PCA
- When PCA tries to reconstruct, it compromises: predicts moderate airspeed and moderate RPM
- The gap between actual and reconstructed values is large $\rightarrow$ high error

Why PCA Beats Simple Threshold Monitoring

What does this physically mean?

- High RPM at low airspeed could indicate:
  - Engine **over-revving** (governor malfunction)
  - **Propeller pitch** failure (blade not loading properly)
  - **Sensor malfunction** (RPM gauge stuck high)
- All of these are conditions worth investigating

Advantage over simple threshold monitoring: Checking "RPM > 3000" alone would miss anomalies where RPM is normal but the *combination* of sensors is unusual. PCA detects **multivariate** anomalies — unusual relationships, not just unusual values.

Anomaly Detection: Practical Considerations

Choosing the threshold ϵ:

Percentile	False positive rate	Sensitivity
90th	10% of normal flagged	Catches more anomalies
95th	5% of normal flagged	Balanced (common choice)
99th	1% of normal flagged	Only extreme anomalies

The right choice depends on the **cost of a missed anomaly vs. the cost of a false alarm**.

Choosing k (number of PCs):

- Too few PCs: normal data itself poorly reconstructed $\rightarrow$ high false positive rate
- Too many PCs: anomalies also well-reconstructed $\rightarrow$ high false negative rate
- Rule of thumb: keep enough PCs for 90–95% of normal variance

Aerospace applications: engine health monitoring (FADEC data), structural health monitoring (vibration sensors), avionics fault detection (redundant sensor cross-checks)

Application 3: Data Compression

Problem: Store 10,000 flights $\times$ 8 sensors $\times$ 3600 time steps = 288M values

PCA Solution (from fleet analysis: 2 PCs capture 95%):

1. Apply PCA: 8D $\rightarrow$ 2D
2. Store 2 PC scores per time step + small loading matrix V_k

Storage Comparison:

Method	Values / Flight	10,000 Flights
Original	$8 \times 3600 = 28{,}800$	288M values
PCA (k=2)	$2 \times 3600 = 7{,}200$	72M values
Savings	**75%**	**216M saved**

Reconstruct when needed: $\hat{X} = Z_k V_k^T + \bar{X}$

Trade-off: ~5% information loss vs 75% storage savings

Application 4: Noise Filtering

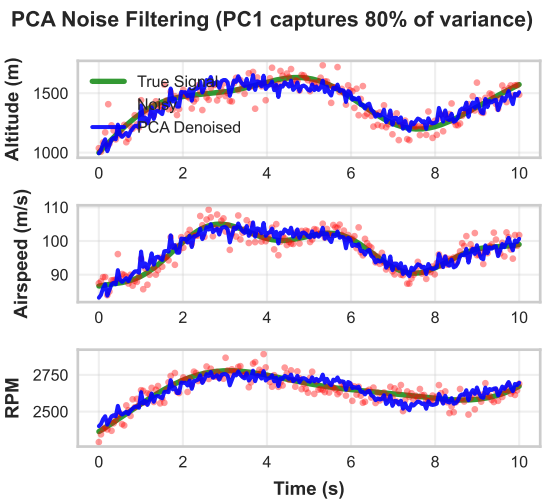
Insight: Signal lives in high-variance PCs; noise in low-variance PCs!

Procedure: PCA $\rightarrow$ keep top k PCs $\rightarrow$ discard the rest $\rightarrow$ reconstruct

$$X_{\text{noisy}} \xrightarrow{\text{PCA}} Z \xrightarrow{\text{Keep top } k} Z_k \xrightarrow{\text{Reconstruct}} X_{\text{denoised}}$$

Application: Clean vibration sensor data, remove electrical noise

Noise Filtering: Example



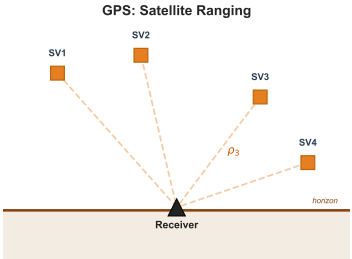
Three Plots (Altitude, Airspeed, RPM):

- **Green**: True (noise-free) signal
- **Red dots**: Noisy measurements
- **Blue**: PCA-denoised reconstruction

Why it works: All 3 sensors are driven by the **same flight profile** (perfectly correlated). PCA captures this shared structure in **PC1**. The noise is independent across sensors → it lands in PCs 2 and 3. Discarding those PCs removes the noise while preserving the signal.

Key point: PCA denoising exploits **cross-sensor correlation** — it would fail if the sensors were independent.

GPS Primer: How Does GPS Work?



Global Positioning System (GPS)
Each GPS satellite continuously broadcasts a radio signal containing:

1. **Timestamp** — when the signal was transmitted
2. **Satellite position** — precise ephemeris data

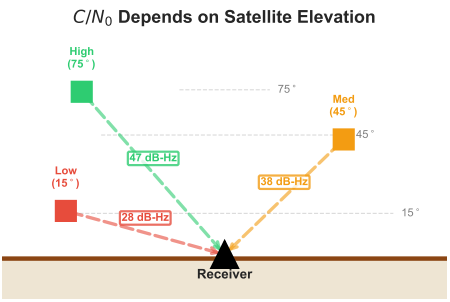
Positioning by trilateration:

- Receiver measures time of arrival of each satellite signal
- Signal travel time $\times$ speed of light $=$ **pseudorange** ρ
- With ≥ 4 satellites: solve for $(x, y, z, \Delta t_{\text{clk}})$

Key facts:

- Orbit altitude: $\sim 20{,}200$ km (MEO)
- 24–31 satellites in constellation
- Signal frequency: L1 $= 1575.42$ MHz
- Typical position accuracy: $\sim$ 3–5 m (civilian)

GPS Primer: Signal Strength — C/N_0



C/N_0: Carrier-to-Noise-Density Ratio
Measures GPS signal quality in **dB-Hz**:

$$C/N_0 = 10 \log_{10}\left(\frac{P_{\text{carrier}}}{N_0}\right)$$

where P_{carrier} is received signal power and N_0 is noise power spectral density.

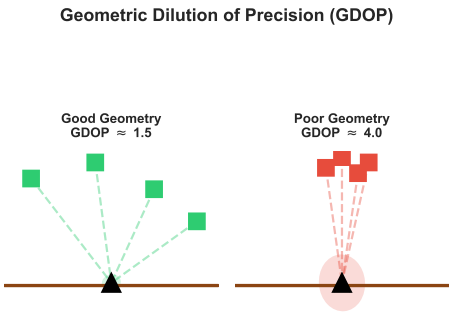
Why does C/N_0 vary across satellites?

Elevation	C/N_0	Reason
High ($> 60°$)	42–50	Short path, less atmosphere
Medium (30–60°)	35–42	Moderate attenuation
Low ($< 30°$)	25–35	Long path through atmosphere

Two useful statistics per epoch:

- C/N_0 spread: range of C/N_0 across tracked satellites (healthy: 10–20 dB)
- C/N_0 mean: average signal strength (healthy: 30–45 dB-Hz)

GPS Primer: Quality Metrics



Number of Visible Satellites

- Typically 7–12 visible from any point on Earth
- More satellites → lower GDOP, higher C/N_0 spread

Pseudorange Residuals

After computing position, the receiver checks how well each pseudorange fits:

$$r_i = \rho_i^{\text{meas}} - \rho_i^{\text{predicted}}$$

- Healthy: residuals are small, randomly distributed ($\sigma \approx 0.5$–2 m)
- Unhealthy: residuals are biased or large

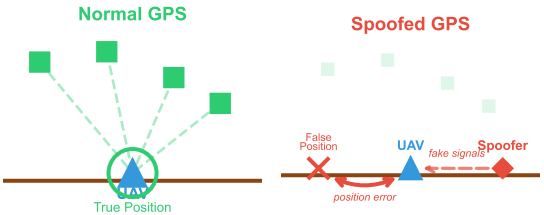
GDOP (Geometric Dilution of Precision)

Quantifies how satellite geometry amplifies measurement errors into position errors:

$$\sigma_{\text{position}} = \text{GDOP} \times \sigma_{\text{range}}$$

- Low GDOP (< 2.5): satellites spread across the sky → accurate fix
- High GDOP (> 4): satellites clustered → poor fix

GPS Primer: What is GPS Spoofing?



GPS Spoofing = broadcasting counterfeit GPS signals that a receiver accepts as authentic

How it works:

1. Attacker generates signals mimicking real GPS satellites
2. Broadcasts at higher power than authentic signals
3. Receiver locks onto fake signals
4. Computed position is wherever the **attacker chooses**

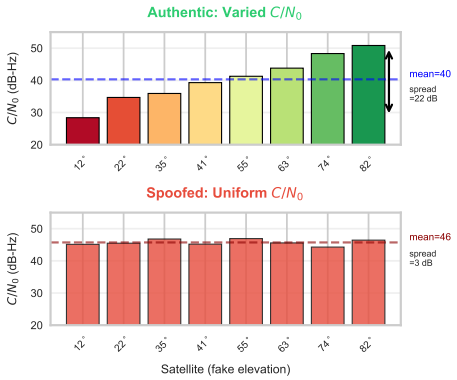
Real-world consequences:

- Military UAVs captured or redirected
- Commercial ships navigated to false positions
- Aircraft reported incorrect locations near conflict zones

Challenge: Civilian GPS has **no signal authentication** — the receiver cannot verify that a signal truly came from a satellite.

$\Rightarrow$ We need **indirect** detection methods.

GPS Spoofing: Why Does a Single-Antenna Spoofer Leave Fingerprints?



Key observation: These 5 features are correlated under authentic GPS but take on an unnatural combination under spoofing.

$\Rightarrow$ PCA trained on authentic data learns these correlations. Spoofed epochs violate them $\rightarrow$ high reconstruction error.

A real GPS constellation has satellites at many different elevations and directions. A spoofer uses one antenna on the ground.

Feature	Authentic	Spoofed
C/N_0 spread	10–20 dB (high + low elev.)	< 3 dB (one source)
C/N_0 mean	30–45 dB-Hz	42–48 dB-Hz (all strong)
Pseudorange residuals	Random, $\sigma \approx 0.5$–2 m	Near-zero (no multipath)
GDOP	1.5–4.0 (real geometry)	< 1.2 (fake perfect geom.)
Num. satellites	Varies (7–12)	Constant (spoofer's choice)

Application 5: GPS Spoofing Detection

Problem: A UAV navigates using GPS. An adversary broadcasts **fake GPS signals** to hijack the vehicle's position estimate. How do you detect spoofing without a second navigation system?

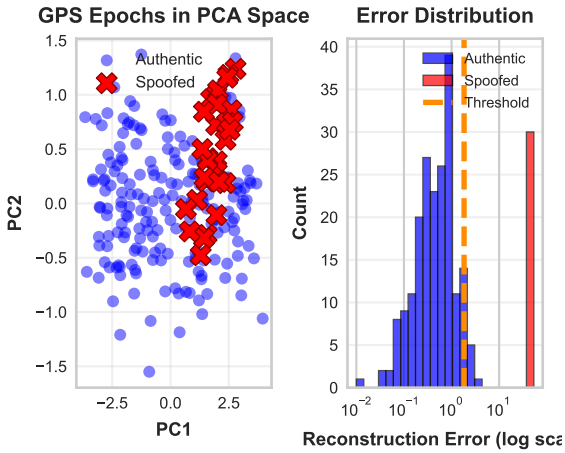
Key insight: Under authentic GPS, receiver observables are **physically constrained** and correlated:

- C/N_0 **spread** across tracked satellites varies naturally (satellites at different elevations)
- C/N_0 **mean** is moderate ($ $30–45 dB-Hz), reflecting the mix of high- and low-elevation satellites
- **Pseudorange residuals** are small and randomly distributed
- **GDOP** (geometric dilution of precision) matches the visible constellation geometry
- **Number of satellites** correlates with C/N_0 spread and GDOP

A spoofer transmits from **one antenna** $\rightarrow$ all "satellite" signals arrive with **suspiciously similar** C/N_0 (near-zero spread), **uniformly high** mean C/N_0 (no low-elevation satellites), and **artificially small** residuals (no real multipath). These break the normal correlation structure.

PCA approach: Fit PCA on features from authentic GPS epochs. During flight, compute reconstruction error per epoch. A spoofer distorts the feature correlations $\rightarrow$ high reconstruction error $\rightarrow$ flag as spoofed.

GPS Spoofing Detection: Example



Left — PCA projection of 5 GPS features:

- Blue: 200 authentic GPS epochs (natural correlation structure)
- Red X: 30 spoofed epochs (single-antenna spoofer)

Right — reconstruction error separates authentic from spoofed

Why spoofing breaks PCA correlations:

- Real GPS: high C/N_0 spread (satellites at different elevations), moderate mean C/N_0 (mix of strong and weak signals), realistic residuals
- Spoofed: near-zero C/N_0 spread (one antenna), uniformly high mean C/N_0 (no low-elevation satellites), tiny residuals (no multipath)

Operational use: Run PCA monitor onboard the UAV. If reconstruction error exceeds threshold for N consecutive epochs, switch to **INS-only** navigation and alert the operator.

Integration: PCA + Clustering

Why Combine PCA and Clustering?

Problem: Clustering high-dimensional data

Challenges:

1. **Curse of dimensionality**: distances become meaningless
2. **Computational cost**: $O(nkdi)$
3. **No visualization possible**
4. **Correlated features**: redundancy confuses algorithms

PCA Solutions:

1. **Reduce dimensions**: 8D $\rightarrow$ 3D
2. **Remove correlation**: PCs are orthogonal
3. **Enable visualization**: project to 2D/3D
4. **Denoise**: remove low-variance noise

Pipeline: Data $\rightarrow$ Standardize $\rightarrow$ PCA $\rightarrow$ Cluster $\rightarrow$ Interpret

Implementation: PCA + K-Means (1/2)

Steps 1–2: Preprocessing and dimensionality reduction

```python
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Step 1: Standardize (critical before PCA!)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 2: PCA -- keep enough components for 95% variance
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)
print(f"Reduced from {X.shape[1]}D to {X_pca.shape[1]}D")
print(f"Variance retained: {pca.explained_variance_ratio_.sum()*100:.1f}%")
```

- `n_components=0.95` automatically selects the minimum number of PCs that retain 95% of variance
- Always standardize **before** PCA — otherwise high-variance features dominate regardless of importance

Implementation: PCA + K-Means (2/2)

Steps 3–5: Clustering, evaluation, and visualization

```
# Step 3: K-Means on the reduced data
kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_pca)

# Step 4: Evaluate cluster quality
silhouette_avg = silhouette_score(X_pca, clusters)
print(f"Silhouette Score: {silhouette_avg:.3f}")

# Step 5: Visualize (works because X_pca is 2D after reduction)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis')
plt.xlabel('PC1'); plt.ylabel('PC2')
plt.title(f'K-Means clusters (Silhouette = {silhouette_avg:.3f})')
plt.colorbar(label='Cluster')
plt.show()
```

- Clustering is done in **PCA space**, not original space — distances are meaningful
- Visualization is possible only because PCA reduced to 2D; check `X_pca.shape[1]` first

When to Use PCA Before Clustering

Use PCA + Clustering when:
✓ High-dimensional data ($d > 10$)
✓ Features are correlated
✓ Need visualization
✓ Want faster computation
✓ Data contains noise

Skip PCA when:
× Data already low-dimensional ($d \leq 3$)
× Features are independent
× Every feature is critical (interpretability)
× Non-linear relationships (consider kernel PCA)

Best Practice: Always try both and compare silhouette scores!

Summary: PCA Key Concepts

What is PCA?

- New coordinate system aligned with data variance
- Projects data onto principal directions
- Reduces dimensions while preserving information

How to Apply?

1. Always standardize first
2. Fit PCA, examine scree plot
3. Choose k: cumulative variance $\geq 95\%$
4. Interpret loadings

Summary: PCA Applications

Four Applications:

1. **Visualization**: 8D $\rightarrow$ 2D/3D plots
2. **Dimensionality Reduction**: Fewer features for ML
3. **Noise Filtering**: Signal in high-variance PCs
4. **Data Compression**: Store fewer numbers

Integration:

- PCA + Classification: reduce features, then classify
- PCA + Clustering: reduce dimensions, then cluster
- PCA + Regression: remove collinearity

Supervised vs Unsupervised

The Complete ML Taxonomy

Method	Type	Input	Output	Example
Regression	Supervised	X, $y \in \mathbb{R}$	$\hat{y} \in \mathbb{R}$	Predict drag coeff.
Classification	Supervised	X, $y \in \{0, \ldots, K-1\}$	Class label	Identify fault type
Clustering	Unsupervised	X only	Cluster labels	Discover patterns
PCA	Unsupervised	X only	Reduced X'	Reduce features

When to Use Each Method

Key question: Do you have labels?

Yes $\rightarrow$ **Supervised**:

- Continuous target $\rightarrow$ **Regression**
- Categorical target $\rightarrow$ **Classification**

No $\rightarrow$ **Unsupervised**:

- Find groups $\rightarrow$ **Clustering**
- Reduce dimensions $\rightarrow$ **PCA**

Hybrid approaches:

- Use **clustering** to create pseudo-labels, then train a **classifier**
- Use **PCA** to reduce dimensions, then **cluster** or **classify**
- Use **PCA** to remove collinearity before **regression**