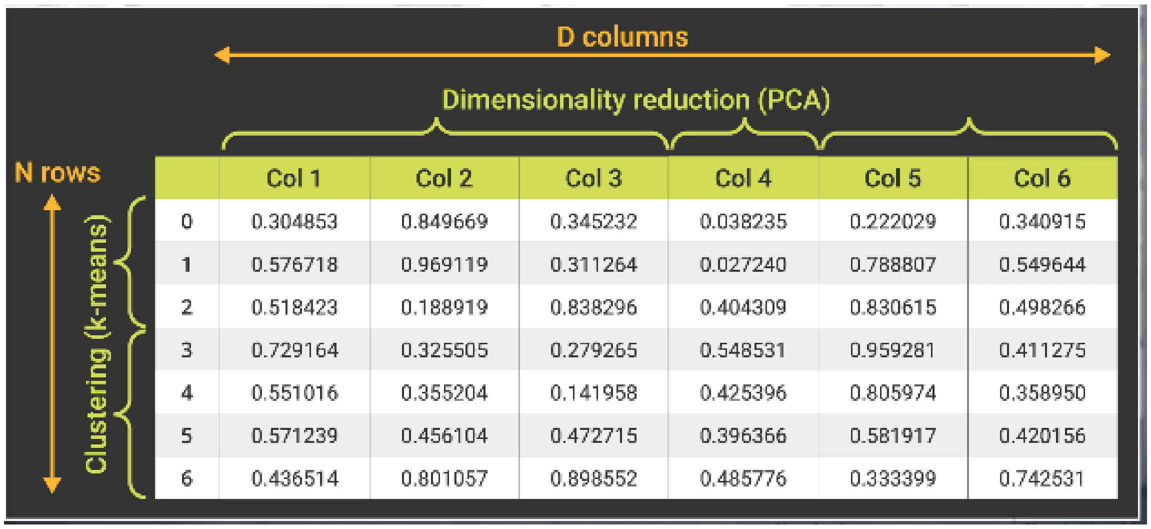
# 1 Module 6: Data Clustering & Principal Component Analysis

## Contents

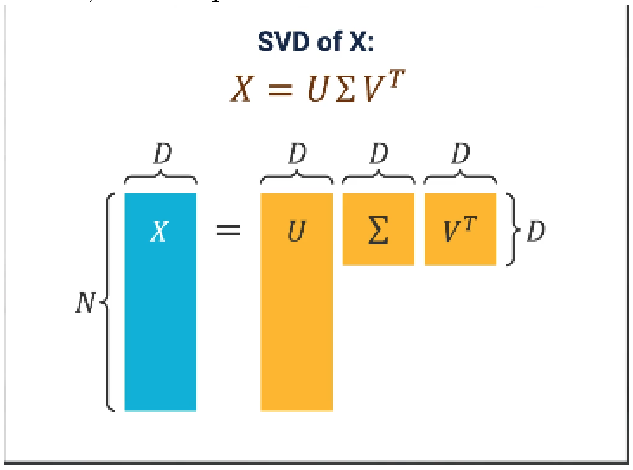
## 1.1 Unsupervised algorithms


D columns
Dimensionality reduction (PCA)
N rows
Clustering (k-means)

| | Col 1 | Col 2 | Col 3 | Col 4 | Col 5 | Col 6 |
|---|---|---|---|---|---|---|
| 0 | 0.304853 | 0.849669 | 0.345232 | 0.038235 | 0.222029 | 0.340915 |
| 1 | 0.576718 | 0.969119 | 0.311264 | 0.027240 | 0.788807 | 0.549644 |
| 2 | 0.518423 | 0.188919 | 0.838296 | 0.404309 | 0.830615 | 0.498266 |
| 3 | 0.729164 | 0.325505 | 0.279265 | 0.548531 | 0.959281 | 0.411275 |
| 4 | 0.551016 | 0.355204 | 0.141958 | 0.425396 | 0.805974 | 0.358950 |
| 5 | 0.571239 | 0.456104 | 0.472715 | 0.396366 | 0.581917 | 0.420156 |
| 6 | 0.436514 | 0.801057 | 0.898552 | 0.485776 | 0.333399 | 0.742531 |

- K-Means Algorithm for clustering
  - Looks for rows that are similar to each other
  - Creates groups out of similar rows
- Principal Components Analysis (PCA)
  - Looks for columns that are linear combinations of existing columns
  - Tries to capture the bulk of variation in the data

- e.g. a dataset with F° and C°: want to keep one to reduce dimensionality because they are already highly correlated
  F° has wider variation (40°) than C° (25°), therefore you would want F: larger spread means more-precise model (assuming both measurements have equal precision)
  That said, largest variance is the data projected onto the line-of-best-fit
- **Singular value decomposition** X: input data with rows N and columns (features) D
  $\rightarrow$ U is the same dimensions as X; $\sum$ is a diagonal matrix of singular values; $V^T$ is square DxD matrix



- Unsupervised learning: do not make use of output in training data
- **Curse of dimensionality**: amt of data needed to train a model increases exponentially with the number of inputs (columns) in order to cover the input space

## 1.2 Performing SVD

- For SVD, we need the mean of the data to be at the origin: Normalize data

$$X_{norm} = \frac{X - \mu}{\sigma}$$

  for $\mu$ and $\sigma$ the mean and std
- Then use SVD on normalized data

$$X_{norm} = U\Sigma V^T$$

U, s, Vt = scipy.linalg.svd(X, full_matrices = True)
$\Sigma$ = np.diag(s)
$V^T$ = V.T

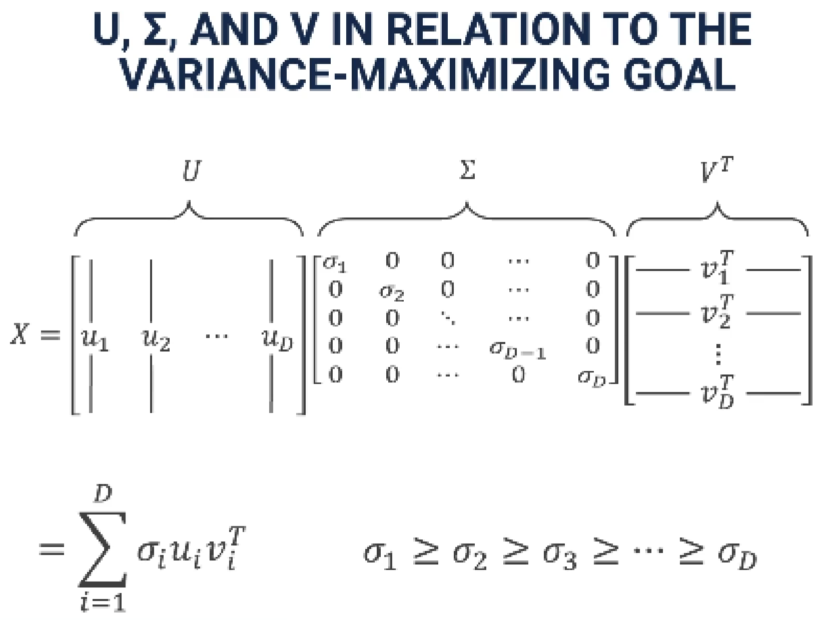
- Check that the decomposition is correct:
  U @ $\Sigma$ @ $Vt^T$ where @ = matrix multiplication in python
  X == U @ $\Sigma$ @ $V^T$ $\rightarrow$ np.allclose(X, U @ $\Sigma$ @ $V^T$)

- We can undo the original normalization (with no loss in precision) by inverting the formula

$$X = \mu + \sigma X_{norm}$$

  $\mu$ + $\sigma$ * pd.DataFrame(U @ $\Sigma$ @ $V^T$)

## 1.3 Connecting SVD to PCA

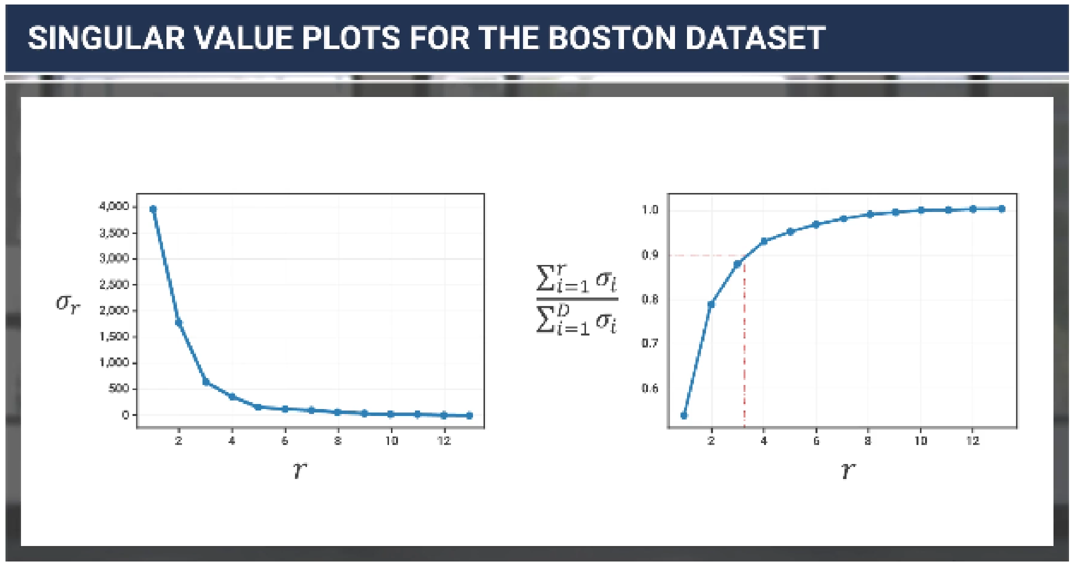

U, Σ, AND V IN RELATION TO THE VARIANCE-MAXIMIZING GOAL
$U$
$\Sigma$
$V^T$
$X = \underbrace{[u_1 \; u_2 \; \cdots \; u_D]}_{U} \; \underbrace{\mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{D-1}, \sigma_D)}_{\Sigma} \; \underbrace{[v_1^T; v_2^T; \vdots; v_D^T]}_{V^T}$
$= \sum_{i=1}^{D} \sigma_i u_i v_i^T$
$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_D$

- $\Sigma$ diagonal entries 1-D are the singular values (weighting) organized from largest to smallest ($\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_D$)

- $V^T$ are the optimal directions for projecting the data; each is a direction in $\mathbb{R}(D)$ (input space) and are each mutually orthogonal

- We can select the data by coarseness

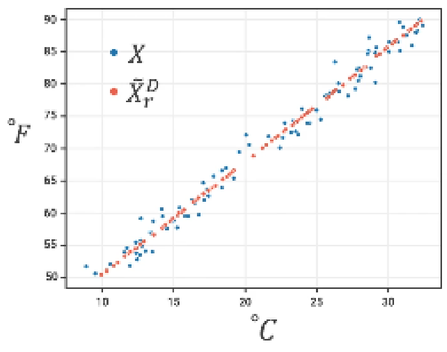
$$\tilde{X}_r^D = \sum_{i=1}^{r} \sigma_i u_i v_i^T \text{ for } u_i \in U \text{ and } v_i^T \in V^T$$

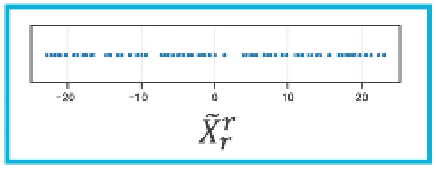

SINGULAR VALUE PLOTS FOR THE BOSTON DATASET

- We can select a bar for cumulative variance (e.g. 90%+ above) and project the data onto the lower-dimensional subspace

$$\tilde{X}_r^r = \tilde{X}_R^D V_r = U_r \Sigma_r$$

$$\tilde{X}_r^D = U_r \Sigma_r V_r^T \qquad \tilde{X}_r^r = \tilde{X}_R^D V_r = U_r \Sigma_r$$





Project $\tilde{X}_r^D$ to $\tilde{X}_r^r$ as shown above.

## 1.4 Performing PCA

- As before,
  $\mu$ = X.mean()
  $\sigma$ = X.std()
  $X_{norm} = \frac{X-\mu}{\sigma}$
  U, s, Vt = svd($X_{norm}$, full_matrices=True)
  $\Sigma$ = np.diag(s)
  $V^T$ = Vt.T

- Now we project data to 4 dimensions, for example

$$\tilde{X}_r^r = U_r \Sigma_r$$

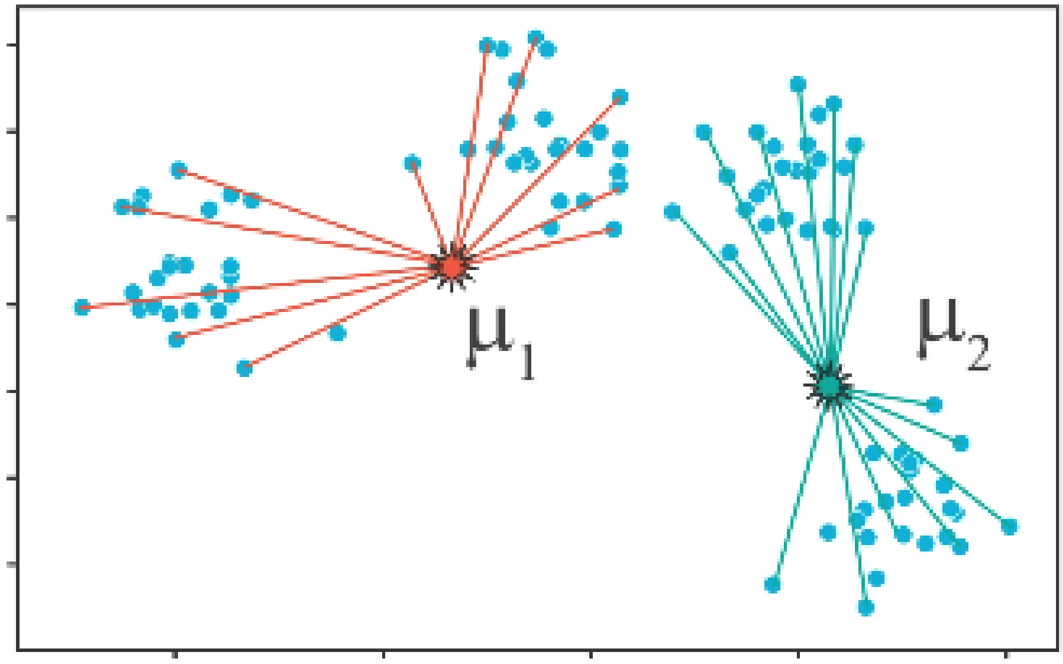
  $U_r$ = U[:,:4]
  $\Sigma_r$ = $\Sigma$[:r,:r] $\tilde{X}_r^r$ = pd.DataFrame($U_r$ @ $\Sigma_r$)

- What if we need to incorporate new data?: We can project down to the principal components we have already computed

$$\tilde{X}_r^r = \tilde{X}_r^D V_r$$

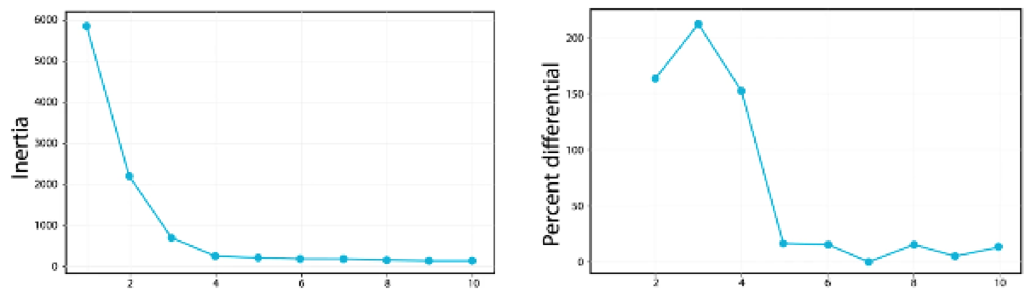
  First, normalize new data and get $\tilde{X}_r^D$; then,
  $\tilde{X}_r^r$.append( $\tilde{X}_r^r$ @ $V^T$[:,:r] )

### 1.5 Clustering and K-Means



- Centroids are the mean of a cluster ($\mu_1, \mu_2$); we try to minimize the inertia (sum of squared distance a.k.a. rotational inertia)
- Algorithm:
  - Select initial centroid placement (e.g. normal distribution)
  - Assign cluster points with nearest centroid
  - Calculate inertia (sum of the squared distance from the points to centroid)
  - Move centroids to the middle of their respective clusters
  - Iterate.
- We need to select an appropriate amount of centroids, k (between extremes k=1 with maximal inertia, and k=n for number of data points with zero inertia)

**SELECTING K: THE ELBOW METHOD**

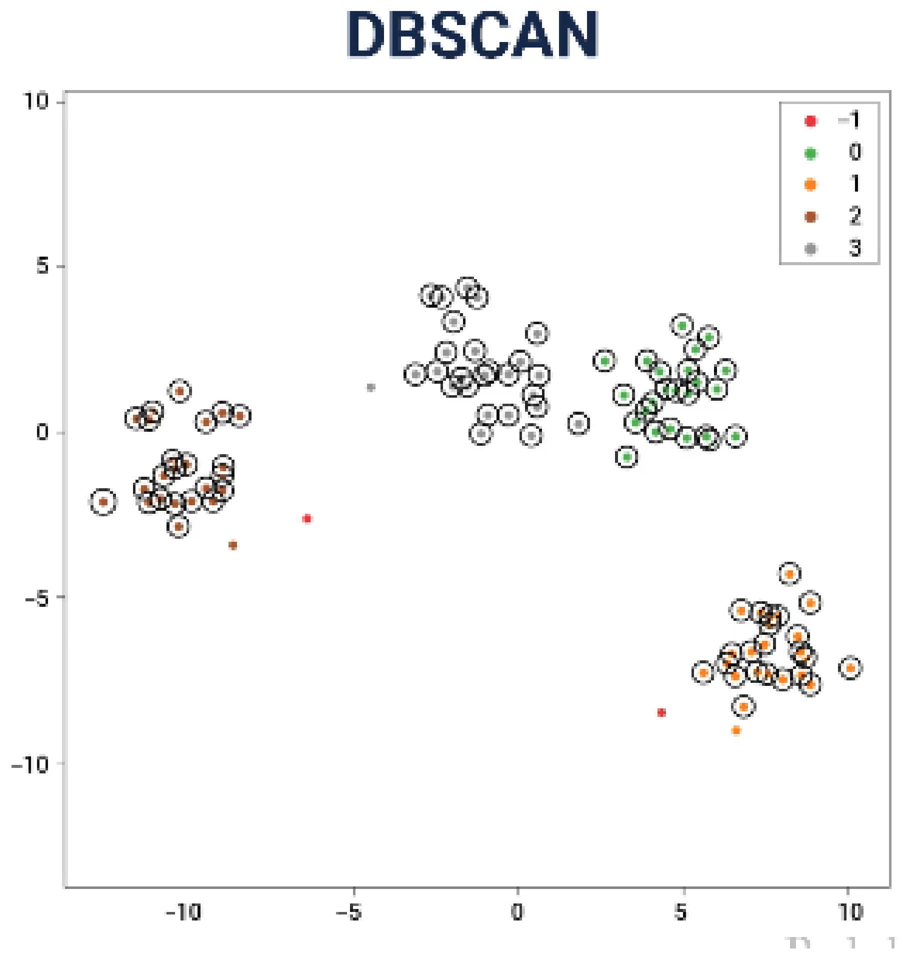


- Look to find a sharp change in either the Inertia or Percent Difference

## 1.6 Performing KMeans

- **Standard KMeans**
  km = sklearn.cluster.KMeans(n_clusters=5, init='random')
  km.fit(data)
  Label for each datapoint is returned as a label with km.labels_
- **KMeans++** improves initialization
  kmp = sklearn.cluster.KMeans(n_cluster=5, init='k-means++', verbose=1).fit(data)
  First centroid is chosen randomly and the remaining centroids try to space themselves evenly over the dataset
- **DBSCAN** Density-Based Spatial Clustering of Applications with Noise
  dbs = cluster.DBSCAN(eps=5, min_samples=3).fit(data)
  - eps = 'radius'
  - No cluster parameter: number of cluster arises naturally from algorithm (do not need to run ensembles; every run is roughly the same)
  - Curved boundaries to the cluster (KMeans is convex polygonal boundaries)

- Outliers are found which are labelled -1 and not included in clustering



- Algorithm:
  - For each data point, scan a radius of size epsilon – if min_samples points are contained, that point is designated a core point
  - Join all core points in one cluster
  - Points that are not core, but have at least one core point in their epsilon neighborhood, are added to that cluster (if it adjacent to TWO clusters, one is assigned randomly)
  - All others are outliers

- **Predict class** with kmeans_object.predict(new_data)
  Note that DBSCAN cannot predict automatically with sklearn