ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

📖 CLUSTERING TECHNIQUES: K-MEANS, HIERARCHICAL CLUSTERING, DBSCAN

📌 CHAPTER 1: INTRODUCTION TO CLUSTERING

1.1 What is Clustering?

Clustering is an **unsupervised machine learning technique** used to group similar data points together. It helps in:

- Identifying hidden patterns in data.
- Market segmentation and customer behavior analysis.
- Anomaly detection in cybersecurity and fraud detection.

1.2 Why is Clustering Important?

Clustering is widely used in various industries:

- **Marketing** – Customer segmentation for targeted advertising.
- **Healthcare** – Disease classification based on symptoms.
- **E-commerce** – Recommending products based on browsing behavior.

- **Image Processing** – Object detection in images.

1.3 Types of Clustering Techniques

- **K-Means Clustering** – A centroid-based clustering method.
- **Hierarchical Clustering** – A tree-like clustering method.
- **DBSCAN (Density-Based Spatial Clustering)** – A density-based clustering method.

📌 CHAPTER 2: K-MEANS CLUSTERING

2.1 Understanding K-Means

K-Means is one of the most popular clustering techniques. It divides data into **K clusters,** where each cluster has a centroid that represents the group.

How K-Means Works:

1. Select **K** (number of clusters).
2. Initialize **K** centroids randomly.
3. Assign each data point to the nearest centroid.
4. Update centroids based on cluster assignments.
5. Repeat until centroids no longer change.

2.2 Example of K-Means Clustering

Imagine a retail store analyzing **customer spending habits**.
The algorithm clusters customers into **high spenders, medium spenders, and low spenders**.

2.3 Choosing the Optimal K Value

The **Elbow Method** helps determine the best number of clusters by plotting the **inertia** (sum of squared distances to the nearest centroid).

```
from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import numpy as np


# Sample data

X = np.random.rand(100, 2)


# Find the optimal K using Elbow Method

wcss = []

for k in range(1, 11):

    kmeans = KMeans(n_clusters=k, random_state=42)

    kmeans.fit(X)

    wcss.append(kmeans.inertia_)


# Plot Elbow Method

plt.plot(range(1, 11), wcss, marker='o')

plt.xlabel("Number of Clusters (K)")

plt.ylabel("WCSS (Within Cluster Sum of Squares)")

plt.title("Elbow Method for Optimal K")
```

plt.show()

- **Optimal K** is the point where the **graph bends like an elbow**.

📌 CHAPTER 3: HIERARCHICAL CLUSTERING

3.1 What is Hierarchical Clustering?

Hierarchical clustering creates a **tree-like structure (dendrogram)** to group data points based on similarity.

- **Agglomerative Clustering (Bottom-Up Approach)** – Each data point starts as its own cluster and merges step by step.
- **Divisive Clustering (Top-Down Approach)** – Starts with all data points in one cluster and splits recursively.

3.2 How Hierarchical Clustering Works

1. Compute the **distance matrix** between all points.
2. Merge the two closest clusters.
3. Repeat until all points belong to one cluster.
4. Cut the dendrogram at the desired number of clusters.

3.3 Visualizing Hierarchical Clustering (Dendrogram)

```
import scipy.cluster.hierarchy as sch

import matplotlib.pyplot as plt

import numpy as np


# Generate random data
```

```
X = np.random.rand(20, 2)


# Create a dendrogram
plt.figure(figsize=(8, 5))
sch.dendrogram(sch.linkage(X, method='ward'))
plt.title("Dendrogram of Hierarchical Clustering")
plt.xlabel("Data Points")
plt.ylabel("Euclidean Distance")
plt.show()
```

- **Cutting the dendrogram** at different levels determines the **number of clusters**.

📌 CHAPTER 4: DBSCAN (DENSITY-BASED CLUSTERING)

4.1 What is DBSCAN?

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a **density-based clustering algorithm** that groups data points based on **dense regions** and marks outliers.

- **Core points** – Have a minimum number of neighbors.
- **Border points** – Near core points but have fewer neighbors.
- **Outliers (Noise points)** – Do not belong to any cluster.

4.2 How DBSCAN Works

1. Select **ε (epsilon)** (neighborhood radius) and **MinPts** (minimum points required to form a cluster).
2. Identify **core points** with enough neighbors.
3. Expand clusters from core points.
4. Mark **outliers** that do not belong to any cluster.

4.3 Implementing DBSCAN

```
from sklearn.cluster import DBSCAN
import numpy as np
import matplotlib.pyplot as plt

# Generate random data
X = np.random.rand(50, 2)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.2, min_samples=3)
clusters = dbscan.fit_predict(X)

# Plot clusters
plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='rainbow', marker='o')
plt.title("DBSCAN Clustering")
plt.xlabel("Feature 1")
```

```
plt.ylabel("Feature 2")

plt.show()
```

- DBSCAN is **great for finding clusters of varying shapes** and detecting **outliers**.

📌 CHAPTER 5: COMPARISON OF CLUSTERING TECHNIQUES

Feature	**K-Means**	**Hierarchical Clustering**	**DBSCAN**
Cluster Shape	Circular (Spherical)	Tree-based	Any Shape
Works with Noise?	No	No	Yes
Scalability	Fast	Slow for large data	Efficient
Handles Outliers?	No	No	Yes
Best For	Large datasets	Hierarchical data	Complex, noisy data

- **K-Means** – Best for large structured datasets.
- **Hierarchical** – Best for **small datasets** and tree-like structures.
- **DBSCAN** – Best for **density-based clustering** and outlier detection.

📌 CHAPTER 6: EXERCISES & ASSIGNMENTS

6.1 Multiple Choice Questions

1. **Which clustering method requires choosing the number of clusters (K)?**
   (a) K-Means ✅
   (b) DBSCAN
   (c) Hierarchical
   (d) None

2. **Which clustering method can detect outliers?**
   (a) K-Means
   (b) Hierarchical
   (c) DBSCAN ✅
   (d) None

3. **Which technique builds a dendrogram?**
   (a) K-Means
   (b) Hierarchical ✅
   (c) DBSCAN
   (d) Logistic Regression

6.2 Practical Assignments

- **Task 1:** Perform **K-Means Clustering** on a dataset (e.g., customer segmentation).
- **Task 2:** Create a **dendrogram using Hierarchical Clustering** and interpret the results.
- **Task 3:** Apply **DBSCAN on a noisy dataset** and detect outliers.

📌 CHAPTER 7: SUMMARY

- **K-Means Clustering** – Uses centroids to create **K clusters**.
- **Hierarchical Clustering** – Builds a **dendrogram** for clustering.
- **DBSCAN** – Groups points based on **density** and detects outliers.
- Each technique is **best suited** for different types of data and clustering needs.

🌟 CONCLUSION: THE FUTURE OF CLUSTERING

Clustering is a key technique in **AI, big data, and business analytics**. Advanced techniques like **fuzzy clustering and deep learning-based clustering** are shaping the future of AI-driven insights.

☑ Dimensionality Reduction: Principal Component Analysis (PCA), t-SNE

📌 Chapter 1: Introduction to Dimensionality Reduction

1.1 What is Dimensionality Reduction?

Dimensionality reduction is a machine learning technique used to **reduce the number of features (dimensions) in a dataset while retaining important information**. It helps in handling high-dimensional data by removing redundant or less informative features.

1.2 Why is Dimensionality Reduction Important?

- **Reduces computational complexity** and improves model efficiency.
- Helps in **visualizing high-dimensional data** in 2D or 3D.
- **Removes multicollinearity,** leading to better generalization.
- **Prevents overfitting** by reducing noise in the dataset.

1.3 Types of Dimensionality Reduction

There are two major types:

1. **Feature Selection**: Selects the most relevant features (e.g., correlation analysis, LASSO regression).

2. **Feature Extraction**: Transforms data into a lower-dimensional space while preserving its structure. (e.g., **Principal Component Analysis (PCA), t-SNE**).

This study material focuses on two key techniques:

- **Principal Component Analysis (PCA)**
- **t-Distributed Stochastic Neighbor Embedding (t-SNE)**

📌 Chapter 2: Principal Component Analysis (PCA)

2.1 What is PCA?

Principal Component Analysis (PCA) is an **unsupervised learning algorithm** that reduces the dimensionality of a dataset by transforming correlated features into a set of **uncorrelated principal components**.

PCA **preserves variance,** meaning it keeps as much information as possible while reducing dimensions.

2.2 How PCA Works?

1. **Standardize the dataset**: Normalize features to ensure equal contribution.
2. **Compute the covariance matrix**: Measures relationships between features.
3. **Calculate eigenvalues & eigenvectors**: Used to find principal components.
4. **Select top K principal components**: Keep components that capture most of the variance.

5. **Transform data** into a new lower-dimensional space.

2.3 PCA Example

Imagine a dataset with **100 features**. PCA can **reduce it to 2 or 3 features** while retaining most of the information.

2.4 When to Use PCA?

✅ When dealing with **high-dimensional datasets** (e.g., images, genomics).
✅ When features are **correlated**, leading to redundancy.
✅ When **reducing computational complexity** is necessary.

2.5 Advantages of PCA

✓ Reduces overfitting and improves model performance.
✓ Speeds up training and prediction time.
✓ Helps in data visualization (converting high-dimensional data into 2D or 3D).

2.6 Disadvantages of PCA

✗ Loses interpretability, as transformed features are not directly understandable.
✗ Assumes **linear relationships** between variables, which may not always be true.

2.7 Implementing PCA in Python

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA
```

```
from sklearn.preprocessing import StandardScaler


# Load dataset (example: Iris dataset)
from sklearn.datasets import load_iris
data = load_iris()
X = data.data  # Features


# Standardize the dataset
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


# Apply PCA
pca = PCA(n_components=2)  # Reduce to 2 components
X_pca = pca.fit_transform(X_scaled)


# Plot PCA-transformed data
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=data.target, cmap='viridis',
edgecolor='k')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA on Iris Dataset")
```

```
plt.colorbar()

plt.show()
```

📌 Chapter 3: t-Distributed Stochastic Neighbor Embedding (t-SNE)

3.1 What is t-SNE?

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a **non-linear dimensionality reduction technique** used for **visualizing high-dimensional data** in 2D or 3D.

Unlike PCA, which focuses on **variance**, t-SNE **preserves local similarities**, meaning similar points remain close together.

3.2 How t-SNE Works?

1. **Computes pairwise similarities** in high-dimensional space.
2. **Maps points** into a lower-dimensional space (2D or 3D).
3. **Minimizes Kullback-Leibler (KL) divergence**, ensuring that similar points remain close.

3.3 When to Use t-SNE?

✅ When visualizing **complex, high-dimensional data** (e.g., image datasets).
✅ When PCA fails to capture **non-linear relationships**.
✅ When clustering similar data points is required.

3.4 Advantages of t-SNE

✓ **Better than PCA for visualization** because it captures **local structures**.
✓ Works well with **non-linear data**.

3.5 Disadvantages of t-SNE

✕ **Computationally expensive,** especially for large datasets.
✕ **Not suitable for feature extraction,** only for visualization.
✕ **Non-deterministic** (results vary for different runs unless seed is fixed).

3.6 Implementing t-SNE in Python

```
from sklearn.manifold import TSNE


# Apply t-SNE
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)


# Plot t-SNE-transformed data
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=data.target, cmap='viridis', edgecolor='k')
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.title("t-SNE Visualization of Iris Dataset")
plt.colorbar()
plt.show()
```

📌 Chapter 4: PCA vs. t-SNE

Feature	PCA	t-SNE
Type	Linear	Non-Linear
Goal	Reduce dimensionality while preserving variance	Preserve local relationships
Best For	Feature extraction & speed improvement	Data visualization
Computational Cost	Low	High
Interpretability	Components are linear combinations of original features	Not easily interpretable

📌 Chapter 5: Case Study

5.1 Case Study: Handwritten Digit Classification

Problem:

We have images of handwritten digits (0-9) from the **MNIST dataset** (28x28 pixels, 784 features). The goal is to **reduce dimensions and visualize clusters**.

Solution Using PCA & t-SNE

```
from sklearn.datasets import fetch_openml

import seaborn as sns
```

```
# Load MNIST dataset
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target.astype(int)

# Standardize data
X_scaled = StandardScaler().fit_transform(X)

# Apply PCA (reduce to 50 components before t-SNE for efficiency)
pca = PCA(n_components=50)
X_pca = pca.fit_transform(X_scaled)

# Apply t-SNE
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
X_tsne = tsne.fit_transform(X_pca)

# Plot results
plt.figure(figsize=(8,6))
sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=y,
palette="tab10", s=10)
plt.title("t-SNE Visualization of MNIST Digits")
```

```
plt.show()
```

Key Observations

- **t-SNE reveals clusters for each digit,** showing its effectiveness.
- **PCA helps speed up t-SNE** by reducing initial dimensions.

📌 Chapter 6: Summary & Conclusion

6.1 Key Takeaways

✅ **PCA** reduces dimensionality by capturing variance.
✅ **t-SNE** is best for **visualizing** high-dimensional data.
✅ Use **PCA when feature extraction is needed,** and **t-SNE for visualization**.

6.2 Next Steps

- **Try other techniques** like **UMAP** (faster alternative to t-SNE).
- **Apply PCA in real-world applications** like **image compression, finance, and bioinformatics**.
- **Use deep learning-based methods** like **autoencoders** for dimensionality reduction.

🕮 Anomaly Detection in Machine Learning

📌 CHAPTER 1: INTRODUCTION TO ANOMALY DETECTION

1.1 What is Anomaly Detection?

Anomaly detection is a machine learning technique used to **identify rare, unusual, or suspicious data points** in a dataset. These anomalies, also known as outliers, deviate from the normal behavior of data and may indicate **fraud, system faults, or cyber threats**.

1.2 Why is Anomaly Detection Important?

Anomaly detection is widely used in various fields: ✓ **Cybersecurity** – Detecting malware, network intrusions, and fraud.
✓ **Finance** – Identifying fraudulent transactions in banking.
✓ **Healthcare** – Detecting abnormal medical conditions in patient data.
✓ **Manufacturing** – Predicting equipment failure through sensor data.

1.3 Types of Anomalies

- **Point Anomalies** – A single instance is significantly different from others (e.g., fraud in bank transactions).
- **Contextual Anomalies** – Anomalies depend on the context (e.g., temperature spikes at midnight).
- **Collective Anomalies** – A group of related anomalies form an abnormal pattern (e.g., DDoS attack).

📌 CHAPTER 2: METHODS OF ANOMALY DETECTION

2.1 Supervised vs. Unsupervised Anomaly Detection

✓ Supervised Learning:

- Requires labeled datasets with normal and abnormal instances.
- Example algorithms: Logistic Regression, Decision Trees, Neural Networks.

✓ Unsupervised Learning:

- Does not require labeled data and finds anomalies by analyzing data distribution.
- Example algorithms: Isolation Forest, Autoencoders, DBSCAN.

✓ Semi-Supervised Learning:

- Trains on normal data only and detects deviations as anomalies.

2.2 Statistical Methods for Anomaly Detection

🔹 **Z-Score Analysis:** Detects anomalies by measuring how far data deviates from the mean.

🔹 **IQR (Interquartile Range):** Identifies outliers based on statistical range.

2.3 Machine Learning Methods

✓ **Isolation Forest (IForest)** – Efficient algorithm that isolates anomalies by randomly partitioning data.

✓ **Autoencoders (Neural Networks)** – Detect anomalies by reconstructing input data with deep learning.

✓ **One-Class SVM** – Learns a boundary for normal data and classifies anomalies as deviations.

📌 CHAPTER 3: STATISTICAL ANOMALY DETECTION

3.1 Z-Score for Anomaly Detection

Z-score measures how many standard deviations a data point is from the mean.

Formula:

$$Z = \frac{X - \mu}{\sigma}$$

where:

- X = Data point
- μ = Mean
- σ = Standard deviation

A **Z-score > 3 or < -3** indicates an anomaly.

3.2 Interquartile Range (IQR) Method

IQR helps detect outliers using quartiles:

Formula:

$$\text{IQR} = Q3 - Q1$$

where:

- $Q1$ = 25th percentile
- $Q3$ = 75th percentile
- Data points beyond $Q1 - 1.5 \times IQR$ or $Q3 + 1.5 \times IQR$ are anomalies.

3.3 Implementing Statistical Methods in Python

```
import numpy as np

import pandas as pd



# Sample dataset

data = np.array([10, 12, 15, 14, 18, 21, 100, 23, 25, 30])  # 100 is an anomaly



# Compute Z-score

mean = np.mean(data)

std = np.std(data)

z_scores = (data - mean) / std

outliers = data[np.abs(z_scores) > 3]

print("Z-score Anomalies:", outliers)
```

```
# Compute IQR
Q1, Q3 = np.percentile(data, [25, 75])
IQR = Q3 - Q1
outliers = data[(data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))]
print("IQR Anomalies:", outliers)
```

📌 CHAPTER 4: MACHINE LEARNING APPROACHES FOR ANOMALY DETECTION

4.1 Isolation Forest

Isolation Forest identifies anomalies by **randomly partitioning data** into isolation trees.

```
from sklearn.ensemble import IsolationForest

# Sample dataset
data = [[10], [12], [15], [14], [18], [21], [100], [23], [25], [30]]

# Train model
iso_forest = IsolationForest(contamination=0.1)  # 10% expected anomalies
iso_forest.fit(data)
```

```
# Predict anomalies (-1 = anomaly, 1 = normal)
predictions = iso_forest.predict(data)
print("Isolation Forest Predictions:", predictions)
```

4.2 Autoencoder for Deep Learning-Based Anomaly Detection

Autoencoders are **neural networks** that learn to reconstruct normal data and detect deviations.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Sample dataset
X_train = np.random.normal(size=(100, 10))  # Normal data
X_test = np.random.normal(size=(10, 10))  # Test data with normal distribution
X_test[0] = X_test[0] * 10  # Inject an anomaly

# Build autoencoder
model = Sequential([
    Dense(10, activation='relu', input_shape=(10,)),
    Dense(5, activation='relu'),
    Dense(10, activation='sigmoid')
])
```

```
# Compile and train
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, X_train, epochs=50, verbose=0)

# Compute reconstruction error
reconstruction_error = np.mean(np.abs(model.predict(X_test) - X_test), axis=1)
anomalies = reconstruction_error > np.percentile(reconstruction_error, 95)
print("Autoencoder Anomalies:", anomalies)
```

📌 CHAPTER 5: COMPARISON OF ANOMALY DETECTION METHODS

Method	Type	Best For	Limitations
Z-Score	Statistical	Small datasets	Fails with skewed data
IQR	Statistical	Simple datasets	Fails for large datasets
Isolation Forest	ML-based	Large datasets	Assumes anomalies are few
Autoencoder	Deep Learning	Complex data	Requires lots of data

5.1 Choosing the Right Anomaly Detection Method

✓ Use **statistical methods** for small datasets.
✓ Use **Isolation Forest** for structured, large datasets.
✓ Use **Autoencoders** for deep learning applications.

📌 CHAPTER 6: EXERCISES & ASSIGNMENTS

6.1 Multiple Choice Questions

1. **Which method is best for detecting anomalies in a large dataset?**
☐ (a) Z-Score
☐ (b) Isolation Forest ✅
☐ (c) IQR

2. **What is the main goal of anomaly detection?**
☐ (a) Finding average values
☐ (b) Identifying outliers ✅
☐ (c) Sorting data

3. **Which anomaly type depends on time-based context?**
☐ (a) Point Anomalies
☐ (b) Contextual Anomalies ✅
☐ (c) Collective Anomalies

6.2 Practical Assignments

📌 **Task 1:** Apply Z-score and IQR methods on a real-world dataset.
📌 **Task 2:** Train an Isolation Forest model to detect fraud in transaction data.
📌 **Task 3:** Use an Autoencoder to identify anomalies in sensor data.

📌 CHAPTER 7: SUMMARY

✅ **Anomaly Detection** helps identify unusual patterns in data.
✅ **Statistical methods** (Z-score, IQR) work for small datasets.
✅ **Machine learning models** (Isolation Forest, Autoencoders) are useful for large datasets.
✅ **Choosing the right method** depends on dataset size and complexity.

🌟 CONCLUSION: THE FUTURE OF ANOMALY DETECTION

With advancements in **AI and deep learning**, anomaly detection is becoming more **automated, accurate, and scalable**. Future applications will enhance **cybersecurity, fraud prevention, and predictive maintenance**.

📖 Association Rule Learning (Market Basket Analysis)

📌 CHAPTER 1: INTRODUCTION TO ASSOCIATION RULE LEARNING

1.1 What is Association Rule Learning?

Association Rule Learning is a machine learning technique used to discover interesting relationships or patterns between items in large datasets. It is widely used in Market Basket Analysis, where retailers analyze customer purchase behavior.

🔹 **Example:** If customers frequently buy *bread* and *butter* together, the store can offer a discount on butter for customers who buy bread.

1.2 Why is Association Rule Learning Important?

Association rule learning helps businesses: ✓ Understand customer buying behavior
✓ Optimize product placement
✓ Personalize marketing and recommendations
✓ Improve sales through bundling strategies

1.3 Key Concepts in Association Rule Learning

Concept	Definition
Itemset	A group of items purchased together (e.g., {Milk, Bread})
Support	Frequency of an itemset appearing in transactions
Confidence	Probability that item B is bought when item A is bought
Lift	Strength of association between items (greater than 1 indicates strong relation)

Example:

- If 50% of transactions contain *Milk and Bread* ⟶ Support = 0.50
- If 80% of *Milk buyers* also buy *Bread* ⟶ Confidence = 0.80
- If *Bread* is bought 30% of the time regardless, but when *Milk* is bought, it's 80% ⟶ Lift = **0.80 / 0.30 = 2.67** (Strong association)

📌 CHAPTER 2: APRIORI ALGORITHM

2.1 What is the Apriori Algorithm?

The **Apriori algorithm** is a widely used algorithm for finding frequent itemsets and generating association rules. It works by: ✓ Identifying **frequent itemsets**

✓ Generating **association rules**

✓ Filtering rules based on **support, confidence, and lift**

2.2 Steps in the Apriori Algorithm

1. **Set a minimum support threshold**
2. **Find frequent itemsets that meet the support threshold**
3. **Generate association rules from frequent itemsets**
4. **Filter rules based on confidence and lift**

2.3 Example of the Apriori Algorithm

Sample Transactions

Transaction ID	Items Purchased
1	Milk, Bread, Butter
2	Milk, Bread
3	Bread, Butter
4	Milk, Butter
5	Milk, Bread, Butter

Step 1: Identify Frequent Itemsets

- **Support for {Milk, Bread} = 4/5 = 0.80**
- **Support for {Milk, Butter} = 3/5 = 0.60**
- **Support for {Bread, Butter} = 3/5 = 0.60**

Step 2: Generate Association Rules

- {Milk} ⟶ {Bread} (Confidence = 4/4 = 1.0, Lift = 1.33)
- {Bread} ⟶ {Butter} (Confidence = 3/4 = 0.75, Lift = 1.5)

Step 3: Apply Thresholds

If **minimum confidence = 0.75,** only {Bread} ⟶ {Butter} is selected.

2.4 Implementing Apriori in Python

```
import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules


# Sample data

data = {'Milk': [1,1,0,1,1], 'Bread': [1,1,1,0,1], 'Butter': [1,0,1,1,1]}

df = pd.DataFrame(data)


# Apply Apriori Algorithm

frequent_itemsets = apriori(df, min_support=0.5,
use_colnames=True)

rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.7)


# Display results

print(rules[['antecedents', 'consequents', 'support', 'confidence',
'lift']])
```

📌 CHAPTER 3: ECLAT ALGORITHM

3.1 What is the ECLAT Algorithm?

ECLAT (**E**quivalence **CL**ass **Cl**ustering and **A**ugmentation) is another method for finding frequent itemsets. Unlike Apriori, it: ✓ Uses **depth-first search** instead of breadth-first
✓ Is **faster** for large datasets
✓ Finds **frequent itemsets** but does not generate rules

3.2 Steps in ECLAT

1. **Convert data into a vertical format (item → transactions list)**
2. **Find frequent itemsets based on support threshold**
3. **Generate association rules (if needed)**

3.3 Implementing ECLAT in Python

```
from mlxtend.frequent_patterns import fpgrowth


# Using the same dataset as before

frequent_itemsets = fpgrowth(df, min_support=0.5,
use_colnames=True)


print(frequent_itemsets)
```

📌 CHAPTER 4: COMPARING APRIORI & ECLAT

Feature	**Apriori**	**ECLAT**
Method	Breadth-first search	Depth-first search

Generates Rules?	Yes	No
Speed	Slower on large datasets	Faster
Best for	Small datasets	Large datasets

📌 CHAPTER 5: APPLICATIONS OF MARKET BASKET ANALYSIS

✓ **Retail:** Identifying frequently bought items together
✓ **E-commerce:** Recommending products based on browsing history
✓ **Healthcare:** Finding common medicine prescriptions
✓ **Banking:** Identifying fraudulent transaction patterns

📌 CHAPTER 6: EXERCISES & ASSIGNMENTS

6.1 Multiple Choice Questions

1. **Which metric measures the frequency of an itemset?**
- ☐ (a) Confidence
- ☐ (b) Lift
- ☐ (c) Support ✅
- ☐ (d) Precision

2. **Which algorithm uses depth-first search?**
- ☐ (a) Apriori
- ☐ (b) ECLAT ✅
- ☐ (c) K-Means
- ☐ (d) Decision Tree

3. **Which metric tells us how strongly two items are associated?**

- ☐ (a) Support
- ☐ (b) Confidence
- ☐ (c) Lift ✅
- ☐ (d) Recall

6.2 Practical Assignments

📌 **Task 1:** Use Apriori on a grocery dataset and identify frequent itemsets.

📌 **Task 2:** Compare the results of Apriori and ECLAT on the same dataset.

📌 CHAPTER 7: SUMMARY

✅ **Association Rule Learning** helps uncover relationships between items.

✅ **Apriori** is good for generating association rules but can be slow.

✅ **ECLAT** is faster and better for large datasets.

✅ **Market Basket Analysis** is used in **retail, healthcare, and banking**.

🌟 CONCLUSION: THE FUTURE OF MARKET BASKET ANALYSIS

With the rise of **AI-powered recommendation systems,** Market Basket Analysis is evolving. Modern **deep learning** models can now predict purchase behavior more accurately than ever!

📖 Real-World Applications of Unsupervised Learning

📌 CHAPTER 1: INTRODUCTION TO UNSUPERVISED LEARNING

1.1 What is Unsupervised Learning?

Unsupervised Learning is a type of machine learning where algorithms analyze and group data without labeled outputs. Unlike supervised learning, the model **identifies hidden patterns and structures** in the dataset without prior knowledge of the target variable.

1.2 Why is Unsupervised Learning Important?

✓ Helps discover patterns in **unlabeled data**.
✓ Used for **clustering, anomaly detection, and dimensionality reduction**.
✓ Essential for **exploratory data analysis (EDA)**.
✓ Helps businesses make **data-driven decisions** without requiring labeled data.

1.3 Types of Unsupervised Learning

- **Clustering** – Grouping similar data points together.
- **Dimensionality Reduction** – Reducing large datasets while retaining key information.
- **Anomaly Detection** – Identifying rare and unusual patterns.
- **Association Rule Learning** – Finding relationships between variables (e.g., market basket analysis).

📌 CHAPTER 2: REAL-WORLD APPLICATIONS OF UNSUPERVISED LEARNING

2.1 Customer Segmentation (Marketing & Retail)

📌 How it Works:

- Businesses use clustering algorithms like **K-Means** to group customers based on purchasing behavior, demographics, and interests.
- This helps in creating **personalized marketing campaigns**.

📌 Example:

- **E-commerce platforms (Amazon, Flipkart, Shopify)** use customer segmentation for targeted ads.
- **Banks and financial institutions** use it to tailor credit card offers.

📌 Common Algorithms:

✓ K-Means Clustering
✓ Hierarchical Clustering

2.2 Fraud Detection (Finance & Banking)

📌 How it Works:

- Anomaly detection techniques like **DBSCAN and Isolation Forest** identify unusual spending patterns that could indicate fraud.
- Used in **credit card transactions** and **insurance claims**.

📌 Example:

- **Banks (JPMorgan, HSBC, PayPal)** detect suspicious activities and prevent fraudulent transactions.
- **Cryptocurrency platforms** detect abnormal trading behaviors.

📌 **Common Algorithms:**

✓ One-Class SVM

✓ DBSCAN (Density-Based Clustering)

2.3 Anomaly Detection in Cybersecurity

📌 **How it Works:**

- Security systems use unsupervised learning to **detect malware, network intrusions, and security threats**.
- The model **flags abnormal activities** that deviate from normal user behavior.

📌 **Example:**

- **SIEM systems (Splunk, IBM QRadar)** use unsupervised learning for security monitoring.
- **Google and Microsoft** use it to identify suspicious login attempts.

📌 **Common Algorithms:**

✓ Isolation Forest

✓ Autoencoders

2.4 Recommendation Systems (Entertainment & E-commerce)

📌 **How it Works:**

- Algorithms **identify patterns in user behavior** to recommend relevant content or products.
- Collaborative filtering groups users with similar interests.

📌 **Example:**

- **Netflix, YouTube, and Spotify** recommend movies, videos, and songs.
- **Amazon and eBay** suggest products based on purchase history.

📌 **Common Algorithms:**

✓ K-Means Clustering

✓ Matrix Factorization

2.5 Image Compression & Feature Extraction (Computer Vision)

📌 **How it Works:**

- **Dimensionality reduction techniques** like **PCA (Principal Component Analysis)** reduce image size while preserving details.
- Used for **facial recognition, object detection, and medical imaging**.

📌 **Example:**

- **Facebook and Apple** use PCA for facial recognition.
- **MRI and CT scans** use unsupervised learning for image enhancement.

📌 **Common Algorithms:**

✓ Principal Component Analysis (PCA)

✓ t-SNE (t-Distributed Stochastic Neighbor Embedding)

2.6 Topic Modeling (Natural Language Processing - NLP)

📌 **How it Works:**

- **Latent Dirichlet Allocation (LDA)** finds hidden topics in large text documents.
- Used for **news classification, chatbot training, and document clustering**.

📌 **Example:**

- **Google News, Medium, and Twitter** use topic modeling to categorize articles and tweets.
- **Legal and research firms** use it to summarize long documents.

📌 **Common Algorithms:**

✓ LDA (Latent Dirichlet Allocation)

✓ NMF (Non-Negative Matrix Factorization)

📌 CHAPTER 3: IMPLEMENTING UNSUPERVISED LEARNING USING SCIKIT-LEARN

3.1 Implementing K-Means Clustering for Customer Segmentation

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
```

```
from sklearn.preprocessing import StandardScaler


# Load sample customer data
data = {'Annual_Income': [15, 18, 21, 35, 40, 55, 60, 75, 90, 120],
      'Spending_Score': [39, 81, 6, 77, 40, 76, 6, 94, 28, 98]}
df = pd.DataFrame(data)


# Standardize data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)


# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(df_scaled)


# Visualize clusters
plt.scatter(df['Annual_Income'], df['Spending_Score'],
c=df['Cluster'], cmap='viridis')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.title('Customer Segmentation using K-Means')
plt.show()
```

📌 CHAPTER 4: EXERCISES & ASSIGNMENTS

4.1 Multiple Choice Questions

1. **Which of the following is NOT an application of unsupervised learning?**

- ☐ (a) Customer segmentation
- ☐ (b) Predicting stock prices ✅
- ☐ (c) Anomaly detection
- ☐ (d) Image compression

2. **Which algorithm is commonly used for fraud detection?**

- ☐ (a) K-Means
- ☐ (b) Isolation Forest ✅
- ☐ (c) Linear Regression
- ☐ (d) Decision Trees

3. **Which method is used for dimensionality reduction?**

- ☐ (a) DBSCAN
- ☐ (b) PCA ✅
- ☐ (c) K-Means
- ☐ (d) Logistic Regression

4.2 Practical Assignments

📌 **Task 1:** Implement K-Means clustering on a **retail customer dataset**.

📌 **Task 2:** Use **PCA** to reduce the dimensionality of an image dataset and visualize the results.

📌 CHAPTER 5: SUMMARY

✅ **Unsupervised Learning** helps find patterns in **unlabeled data**.

✅ **K-Means Clustering** is widely used for **customer segmentation**.

✅ **Anomaly Detection** is useful for **fraud prevention and cybersecurity**.
✅ **Dimensionality Reduction (PCA)** is used in **image compression and feature extraction**.
✅ **Recommendation Systems** improve user experiences in **entertainment and e-commerce**.

🌟 Conclusion: The Future of Unsupervised Learning

Unsupervised learning is transforming industries by **automating insights, detecting fraud, and improving recommendation systems**. Future advancements in **AI and deep learning** will further enhance these applications.

📌 ✨ ASSIGNMENT 1:

🎯 PERFORM CUSTOMER SEGMENTATION USING K-MEANS CLUSTERING.

📌✨ Assignment Solution 1: Perform Customer Segmentation using K-Means Clustering

🎯 Objective

The goal of this assignment is to use **K-Means Clustering** to segment customers based on their purchasing behavior. This is a commonly used approach in marketing to **identify different customer groups**, improve targeting, and personalize services.

🛠️ Step 1: Import Necessary Libraries

We will use Python and **Scikit-Learn** for implementing K-Means clustering.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler
```

📂 Step 2: Load the Dataset

We will use a **Customer Spending Dataset** which contains information like **Annual Income, Spending Score, and Age**.

```
# Load dataset (Replace 'customer_data.csv' with your actual dataset)
```

```
df = pd.read_csv("customer_data.csv")


# Display first few rows

df.head()
```

Example Dataset Structure

CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
1	25	30	65
2	32	50	40
3	45	85	20

🔍 Step 3: Exploratory Data Analysis (EDA)

3.1 Check Dataset Information

```
# Get dataset info

df.info()


# Check for missing values

df.isnull().sum()
```

✓ Action Plan:

- If missing values exist**, fill them** using the mean or drop them.

3.2 Visualizing Customer Spending vs Income

```
# Scatter plot for spending score vs annual income

plt.figure(figsize=(8,5))
```

```
sns.scatterplot(x=df['Annual Income (k$)'], y=df['Spending Score (1-100)'], color='blue')

plt.title("Customer Spending vs Annual Income")

plt.xlabel("Annual Income (k$)")

plt.ylabel("Spending Score")

plt.show()
```

📊 Step 4: Feature Selection and Scaling

K-Means clustering works best when the **data is standardized**.

4.1 Selecting Features for Clustering

We use **Annual Income and Spending Score** for segmentation.

```
# Selecting features

X = df[['Annual Income (k$)', 'Spending Score (1-100)']]
```

4.2 Scaling the Data

Since K-Means is distance-based, we need to **scale the data**.

```
# Standardizing the data

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

📉 Step 5: Finding the Optimal Number of Clusters (K)

We use the **Elbow Method** to determine the best **K value**.

```
# Find optimal K using Elbow Method

wcss = []
```

```
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# Plot Elbow Method
plt.figure(figsize=(8,5))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("WCSS (Within Cluster Sum of Squares)")
plt.title("Elbow Method to Determine Optimal K")
plt.show()
```

✓ **Interpretation:**

- Choose the **K value where the WCSS curve forms an elbow** (usually around **3 to 5 clusters**).

📌 Step 6: Build and Train the K-Means Model

We train the K-Means model using the **optimal K** found in Step 5.

```
# Train K-Means model with chosen K (e.g., 4)
kmeans = KMeans(n_clusters=4, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)
```

📊 Step 7: Visualizing the Customer Segments

```
# Scatter plot of clustered customers
plt.figure(figsize=(8,5))
sns.scatterplot(x=df['Annual Income (k$)'], y=df['Spending Score (1-100)'], hue=df['Cluster'], palette='viridis')
plt.title("Customer Segmentation Based on Spending & Income")
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score")
plt.legend(title="Cluster")
plt.show()
```

✓ Insight:

- Each **color represents a unique customer group** with different spending behavior.

📏 Step 8: Analyzing the Customer Segments

Each cluster represents a **distinct customer type**:

Cluster	Description
0	**Low income, low spending** – Budget-conscious shoppers
1	**High income, high spending** – Luxury buyers
2	**Moderate income, moderate spending** – Balanced consumers
3	**Low income, high spending** – Impulse buyers

✓ **Business Impact:**

- **Targeted advertising** – Offer promotions to impulse buyers.
- **Loyalty programs** – Engage high spenders with special offers.
- **New product strategies** – Customize marketing campaigns for each segment.

📌 Step 9: Conclusion

✅ Key Takeaways

- We **loaded and explored** the dataset.
- We **used K-Means clustering** for **customer segmentation**.
- We determined the **optimal K using the Elbow Method**.
- We visualized **customer groups** and interpreted their spending behavior.

🎯 Next Steps

- Try **adding more features** like **age, location, and purchase history**.
- Experiment with **Hierarchical Clustering** for comparison.
- Use **DBSCAN** for detecting **outliers or rare spending patterns**.

📊 Summary Table

Step	Task
1	Import necessary libraries

2	Load and explore dataset
3	Perform EDA and visualize spending patterns
4	Select relevant features and scale the data
5	Determine the optimal number of clusters using the Elbow Method
6	Train the K-Means clustering model
7	Visualize and analyze customer segments
8	Interpret business insights from clusters

📌 ✨ ASSIGNMENT 2: 🎯 APPLY PCA FOR FEATURE REDUCTION ON A DATASET.

📌 ✨ Assignment Solution 2: Apply PCA for Feature Reduction on a Dataset

🎯 Objective

The goal of this assignment is to **apply Principal Component Analysis (PCA) to reduce the number of features in a dataset** while retaining as much variance as possible. PCA is commonly used to **speed up machine learning models, reduce overfitting,** and **improve interpretability**.

🛠️ Step 1: Install and Import Necessary Libraries

We will use the following libraries:

- **pandas**: For data handling
- **numpy**: For numerical operations
- **matplotlib & seaborn**: For visualization
- **sklearn**: For PCA implementation

🔹 Install Dependencies (if not installed)

```
!pip install pandas numpy matplotlib seaborn scikit-learn
```

🔹 Import Required Libraries

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

📥 Step 2: Load and Explore the Dataset

For this assignment, we will use the **Iris dataset**, which contains **4 features** describing flowers and a target column with species labels.

🔹 Load the Dataset

```
from sklearn.datasets import load_iris

# Load dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target variable (species)

# Convert to DataFrame for better readability
df = pd.DataFrame(X, columns=iris.feature_names)
df['species'] = y  # Adding target column

# Display first 5 rows
print(df.head())
```

🔹 Check Dataset Information

```
# Check dataset shape and column types

print(df.info())


# Summary statistics

print(df.describe())
```

🖌️ Step 3: Standardize the Data

PCA is affected by feature scaling, so we must standardize the dataset using **StandardScaler** to ensure all features have equal importance.

```
# Extract features (excluding target)

X_features = df.drop(columns=['species'])


# Standardize the dataset

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X_features)


# Convert back to DataFrame

df_scaled = pd.DataFrame(X_scaled, columns=iris.feature_names)


# Display first few rows after scaling

print(df_scaled.head())
```

📊 Step 4: Apply PCA

Now, let's apply PCA and check how much variance each component captures.

🔹 Compute PCA

```
# Apply PCA

pca = PCA()

X_pca = pca.fit_transform(X_scaled)



# Convert to DataFrame

df_pca = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(len(X_features.columns))])



# Display first few rows

print(df_pca.head())
```

📈 Step 5: Determine the Optimal Number of Components

To determine how many principal components to keep, we plot the **explained variance ratio**.

🔹 Explained Variance Plot

```
# Get explained variance ratio

explained_variance = pca.explained_variance_ratio_
```

```
# Plot cumulative explained variance
plt.figure(figsize=(8,5))
plt.plot(range(1, len(explained_variance)+1),
np.cumsum(explained_variance), marker='o', linestyle='--')
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Explained Variance by PCA Components")
plt.grid()
plt.show()
```

Interpretation:

- Choose the number of components where the curve **flattens out** (e.g., **95% variance is explained by fewer components**).
- In most cases, **2 components are sufficient**.

📝 Step 6: Reduce Dimensions Using PCA

Since the plot shows that **two components** explain most of the variance, we reduce the dataset to **2 principal components**.

```
# Apply PCA with 2 components
pca_2 = PCA(n_components=2)
X_pca_2 = pca_2.fit_transform(X_scaled)


# Convert to DataFrame
df_pca_2 = pd.DataFrame(X_pca_2, columns=['PC1', 'PC2'])
```

```
df_pca_2['species'] = y  # Add species column


# Display first few rows
print(df_pca_2.head())
```

📌 Step 7: Visualize the PCA-Reduced Data

Now, let's plot the data using the two principal components.

```
# Plot PCA components
plt.figure(figsize=(8,6))
sns.scatterplot(x=df_pca_2['PC1'], y=df_pca_2['PC2'],
hue=df_pca_2['species'], palette='viridis')
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization of Iris Dataset")
plt.legend(title="Species")
plt.grid()
plt.show()
```

Interpretation:

- The species are now **distinguishable** in 2D.
- PCA has reduced **4 features** into **2 principal components** while retaining most of the variance.

🤖 Step 8: Train a Model Using Reduced Features

Now, let's train a **Logistic Regression model** using the **PCA-transformed features** and compare its performance with the original dataset.

```
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


# Split the dataset into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X_pca_2, y, test_size=0.2, random_state=42)


# Train a logistic regression model

model = LogisticRegression()

model.fit(X_train, y_train)


# Predictions

y_pred = model.predict(X_test)


# Accuracy score

print(f"Model Accuracy with PCA: {accuracy_score(y_test, y_pred):.2f}")
```

Results:

- If the accuracy **remains high**, PCA has successfully **reduced dimensionality** without losing much information.

- If accuracy drops significantly, **more components may be needed**.

✅ Final Summary

Step	Description
Step 1	Install and import necessary libraries
Step 2	Load and explore the dataset
Step 3	Standardize the dataset
Step 4	Apply PCA and transform data
Step 5	Determine the optimal number of components
Step 6	Reduce dimensions and retain key features
Step 7	Visualize the transformed data
Step 8	Train a classification model on reduced features

🌟 Conclusion

- **PCA effectively reduces dimensionality** while preserving most of the variance.
- **Visualization improves** when reducing features to 2D.
- **Machine learning models remain effective** even with fewer dimensions.
- **Choosing the right number of components is crucial** for balancing performance and efficiency.

📌 Next Steps

- **Apply PCA to a different dataset** (e.g., handwritten digit recognition).
- **Experiment with more PCA components** to see their effect on model accuracy.
- **Use PCA before clustering algorithms** (e.g., k-means) to improve efficiency.