Programming Assignment 4

Name: Muzammilkhon Muradullaev

In [22]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, c
all drive.mount("/content/drive", force_remount=True).
```

In [23]:
```
import os
import numpy as np
import torch

from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
from torchvision.models.feature_extraction import create_feature_extracto
```

In [24]:
```
DATA_DIR = "/content/drive/MyDrive/Faulty_solar_panel"
print(os.listdir(DATA_DIR))
```

```
['test.json', 'train.json', 'validation.json', '.DS_Store', 'Clean', 'Snow
-Covered', 'Dusty', 'venv', 'Bird-drop']
```

Dataset used (same as previous assignments)

I use the Faulty_solar_panel image dataset located at:

`/content/drive/MyDrive/Faulty_solar_panel`

It contains 4 classes (*Bird-drop, Clean, Dusty, Snow-Covered*).
This satisfies the assignment requirement to use the same dataset as previous assignments and matches K = 4 for clustering.

In [25]:
```
ALLOWED_EXT = {".jpg", ".jpeg", ".png", ".bmp"}

def is_valid_image(path: str) -> bool:
    path = path.lower()
    ext = os.path.splitext(path)[1]
    return ext in ALLOWED_EXT
```

In [26]:
```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

In [27]:
```
dataset = datasets.ImageFolder(
    root=DATA_DIR,
    transform=transform,
    is_valid_file=is_valid_image
)
```

```
print("Classes:", dataset.classes)
print("Total images:", len(dataset))
print("First file:", dataset.samples[0][0])
```

```
Classes: ['Bird-drop', 'Clean', 'Dusty', 'Snow-Covered', 'venv']
Total images: 778
First file: /content/drive/MyDrive/Faulty_solar_panel/Bird-drop/Bird (1).j
peg
```

In [28]:
```
loader = DataLoader(dataset, batch_size=16, shuffle=False)
```

In [29]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"

model = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
model.eval()

extractor = create_feature_extractor(
    model,
    return_nodes={"layer4": "features"}
).to(device)
```

In [30]:
```
@torch.no_grad()
def extract_features(loader):
    feats = []
    for imgs, _ in loader:
        imgs = imgs.to(device)
        out = extractor(imgs)["features"]   # [B, 512, 7, 7]
        out = out.mean(dim=(2, 3))           # GAP → [B, 512]
        feats.append(out.cpu().numpy())
    return np.vstack(feats)

X = extract_features(loader)
print("Feature matrix shape:", X.shape)
```

```
Feature matrix shape: (778, 512)
```

Feature extraction reference (required)

Feature extraction from ResNet18 follows the approach described here (required reference):
https://kozodoi.me/blog/20210527/extracting-features

I extracted activations from `layer4` (last convolutional block) and apply global average pooling to obtain a 512-D feature vector per image.

In [31]:
```
import numpy as np
y_true = np.array(dataset.targets)
print("y_true shape:", y_true.shape, "unique:", np.unique(y_true))
```

```
y_true shape: (778,) unique: [0 1 2 3 4]
```

In [32]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

X_scaled = StandardScaler().fit_transform(X)
X_2d = PCA(n_components=2, random_state=42).fit_transform(X_scaled)
```

```
print("X_2d shape:", X_2d.shape)
```

```
X_2d shape: (778, 2)
```

In [33]:
```
import pandas as pd
from sklearn.metrics import fowlkes_mallows_score, silhouette_score
from sklearn.cluster import (
    KMeans, BisectingKMeans, SpectralClustering, DBSCAN, AgglomerativeClu
)

results = []

def safe_silhouette(X2d, labels):
    uniq = set(labels)
    if len(uniq) < 2:
        return np.nan
    try:
        return silhouette_score(X2d, labels)
    except Exception:
        return np.nan

def evaluate(name, labels):
    return {
        "Method": name,
        "FMI": fowlkes_mallows_score(y_true, labels),
        "Silhouette": safe_silhouette(X_2d, labels),
        "Clusters": len(set(labels))
    }
```

In [34]:
```
labels = KMeans(n_clusters=4, init="random", n_init=10, random_state=42).
results.append(evaluate("KMeans (init=random)", labels))
```

In [35]:
```
labels = KMeans(n_clusters=4, init="k-means++", n_init=10, random_state=4
results.append(evaluate("KMeans (init=k-means++)", labels))
```

In [36]:
```
labels = BisectingKMeans(n_clusters=4, init="random", random_state=42).fi
results.append(evaluate("BisectingKMeans (init=random)", labels))
```

In [37]:
```
labels = SpectralClustering(n_clusters=4).fit_predict(X_2d)
results.append(evaluate("SpectralClustering (default params)", labels))
```

In [38]:
```
def dbscan_find_params(X2d, target_k=4):
    eps_grid = np.linspace(0.1, 5.0, 60)
    min_samples_grid = [3, 4, 5, 6, 8, 10]

    best = None  # (sil, eps, ms, labels, k)
    for ms in min_samples_grid:
        for eps in eps_grid:
            model = DBSCAN(eps=float(eps), min_samples=int(ms))
            labels = model.fit_predict(X2d)
            k = len(set(labels)) - (1 if -1 in labels else 0)  # without

            if k == target_k:

                mask = labels != -1
                if mask.sum() < 2 or len(set(labels[mask])) < 2:
                    sil = -np.inf
                else:
```

```
                    try:
                        sil = silhouette_score(X2d[mask], labels[mask])
                    except Exception:
                        sil = -np.inf

                cand = (sil, eps, ms, labels, k)
                if (best is None) or (cand[0] > best[0]):
                    best = cand
    return best

best_db = dbscan_find_params(X_2d, target_k=4)

if best_db is None:
    print("DBSCAN: cannot find the parameters which gives 4 clusters.")
else:
    sil, eps_used, ms_used, labels, k = best_db
    print(f"DBSCAN cannot find: eps={eps_used:.3f}, min_samples={ms_used}

    mask = labels != -1
    results.append({
        "Method": f"DBSCAN (eps={eps_used:.3f}, min_samples={ms_used})",
        "FMI": fowlkes_mallows_score(y_true[mask], labels[mask]),
        "Silhouette": safe_silhouette(X_2d[mask], labels[mask]),
        "Clusters": k
    })
```

```
DBSCAN cannot find: eps=0.266, min_samples=6, clusters=4
```

DBSCAN note (eps/min_samples + evaluation)

For DBSCAN, we search `eps` and `min_samples` to obtain exactly 4 clusters (excluding noise).
DBSCAN may label some points as noise (`-1`). For a fair comparison, FMI and Silhouette are computed on non-noise points only (`labels != -1`).

In [39]:
```
for link in ["single", "complete", "average", "ward"]:
    labels = AgglomerativeClustering(n_clusters=4, linkage=link).fit_pred
    results.append(evaluate(f"Agglomerative (linkage={link})", labels))
```

In [40]:
```
df = pd.DataFrame(results)

print("=== ALL RESULTS ===")
display(df)

print("=== Rank by FMI (best -> worst) ===")
display(df.sort_values("FMI", ascending=False).reset_index(drop=True))

print("=== Rank by Silhouette (best -> worst) ===")
display(df.sort_values("Silhouette", ascending=False).reset_index(drop=Tr
```

```
=== ALL RESULTS ===
```

	Method	FMI	Silhouette	Clusters
0	KMeans (init=random)	0.392029	0.391367	4
1	KMeans (init=k-means++)	0.392029	0.391367	4
2	BisectingKMeans (init=random)	0.376632	0.359498	4
3	SpectralClustering (default params)	0.506796	0.473370	4
4	DBSCAN (eps=0.266, min_samples=6)	0.446203	0.934064	4
5	Agglomerative (linkage=single)	0.473063	0.393752	4
6	Agglomerative (linkage=complete)	0.393774	0.325200	4
7	Agglomerative (linkage=average)	0.491869	0.428786	4
8	Agglomerative (linkage=ward)	0.346901	0.320865	4

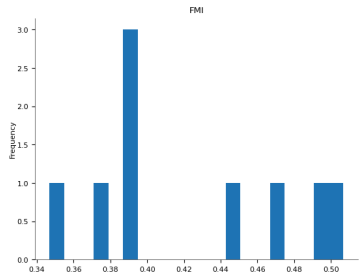
```
=== Rank by FMI (best -> worst) ===
```

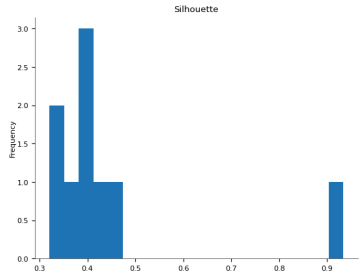
	Method	FMI	Silhouette	Clusters
0	SpectralClustering (default params)	0.506796	0.473370	4
1	Agglomerative (linkage=average)	0.491869	0.428786	4
2	Agglomerative (linkage=single)	0.473063	0.393752	4
3	DBSCAN (eps=0.266, min_samples=6)	0.446203	0.934064	4
4	Agglomerative (linkage=complete)	0.393774	0.325200	4
5	KMeans (init=random)	0.392029	0.391367	4
6	KMeans (init=k-means++)	0.392029	0.391367	4
7	BisectingKMeans (init=random)	0.376632	0.359498	4
8	Agglomerative (linkage=ward)	0.346901	0.320865	4

```
=== Rank by Silhouette (best -> worst) ===
```

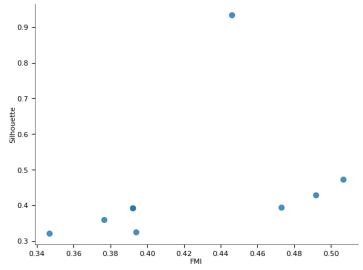
	Method	FMI	Silhouette	Clusters
0	DBSCAN (eps=0.266, min_samples=6)	0.446203	0.934064	4
1	SpectralClustering (default params)	0.506796	0.473370	4
2	Agglomerative (linkage=average)	0.491869	0.428786	4
3	Agglomerative (linkage=single)	0.473063	0.393752	4
4	KMeans (init=random)	0.392029	0.391367	4
5	KMeans (init=k-means++)	0.392029	0.391367	4
6	BisectingKMeans (init=random)	0.376632	0.359498	4
7	Agglomerative (linkage=complete)	0.393774	0.325200	4
8	Agglomerative (linkage=ward)	0.346901	0.320865	4

Distributions

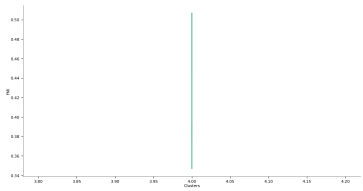


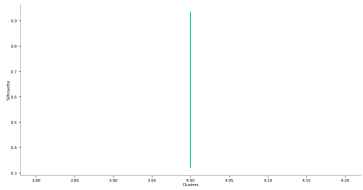


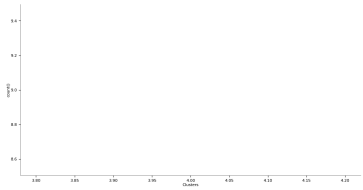
2-d distributions



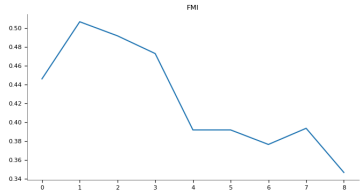
Time series

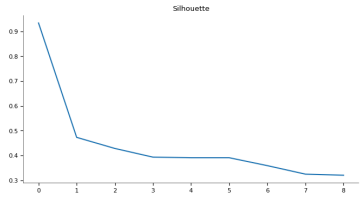






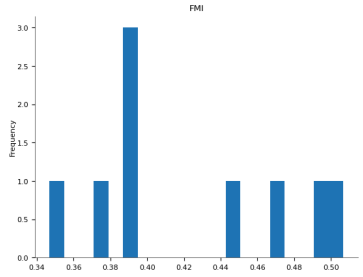
Values

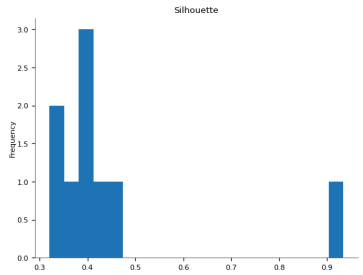




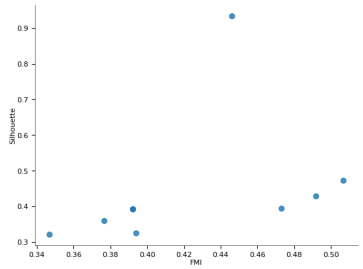

```
WARNING: Runtime no longer has a reference to this dataframe, please re-ru
n this cell and try again.
```

Distributions

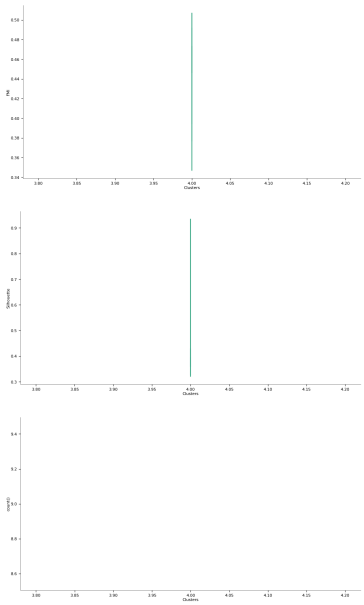




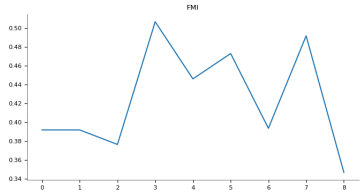
2-d distributions

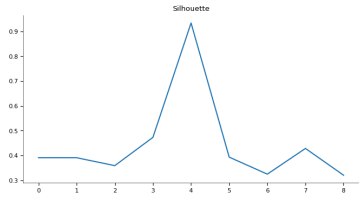


Time series

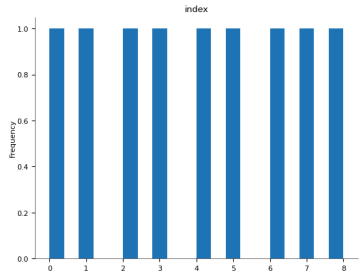


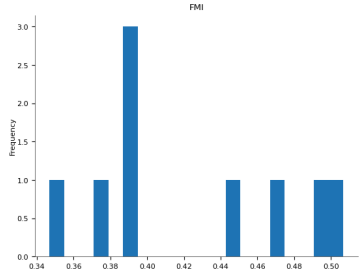
Values

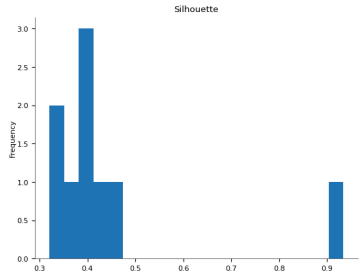




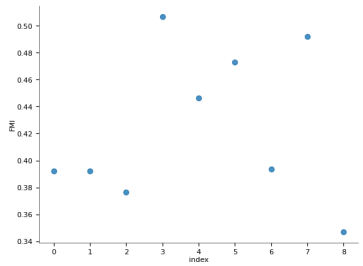
Distributions

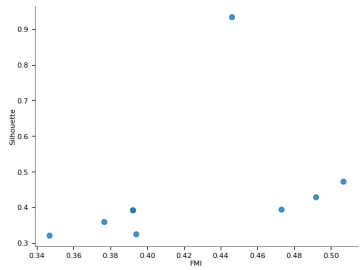




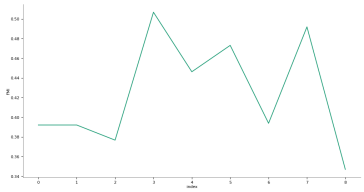


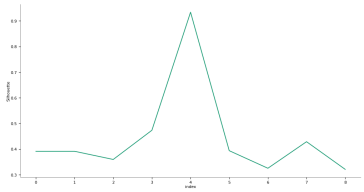
2-d distributions

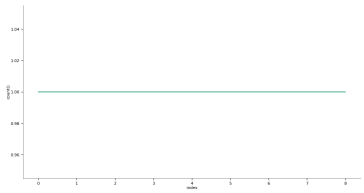


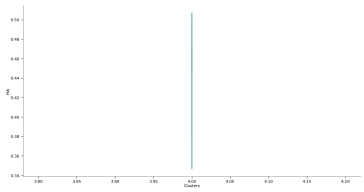


Time series

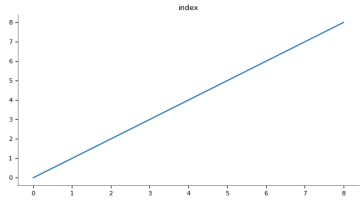


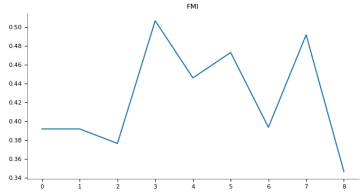


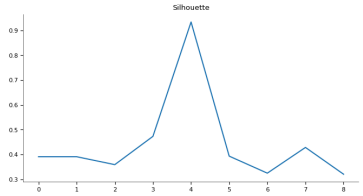




Values







In [40]: