Programming Assignment 1

Student: Muzammilkhon Muradullaev
Course: Data Mining / Fall 2025
Dataset (Images): Kaggle – Solar Panel Images (classes: Bird-drop, Snow-Covered, Dusty, Clean)
Link: https://www.kaggle.com/datasets/pythonafroz/solar-panel-images

In [144…

```
%matplotlib inline

from pathlib import Path
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from skimage import io, color, filters, exposure, feature
from skimage.color import rgb2gray
from sklearn.decomposition import PCA
from sklearn.metrics import pairwise_distances
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectori
plt.rcParams['figure.figsize'] = (6,4)
DATA_DIR = Path('/Users/muradullaev03/Downloads/Faulty_solar_panel/')
CLASSES = ['Bird-drop', 'Snow-Covered', 'Dusty', 'Clean']
IMG_EXTS = {'.jpg', '.jpeg'}
```

In [146…

```
def load_image_gray(path: Path):
    img = io.imread(path)
    if img.ndim == 2:
        gray = img.astype(np.float32)
        if gray.max() > 1.01:
            gray = gray / 255.0
    else:
        if img.shape[-1] == 4:
            img = img[..., :3]
        gray = rgb2gray(img).astype(np.float32)
    return np.clip(gray, 0.0, 1.0)

def angle(dx, dy):
    return np.mod(np.arctan2(dy, dx), np.pi)

def grad_orientation_hist(gray: np.ndarray, nbins: int = 36):
    dx = filters.sobel_h(gray)
    dy = filters.sobel_v(gray)
    theta = angle(dx, dy)
    counts, centers = exposure.histogram(theta, nbins=nbins)
    return counts.astype(np.float32), centers

def hog_descriptor_and_vis(gray: np.ndarray):
    hog_vec, hog_img = feature.hog(
        gray,
        orientations=9,
        pixels_per_cell=(16, 16),
        cells_per_block=(2, 2),
        block_norm='L2-Hys',
        visualize=True
    )
    return hog_vec, hog_img

def list_images_by_class(data_dir: Path, classes):
    mapping = {}
    for cls in classes:
        cls_dir = data_dir / cls
        files = []
        if cls_dir.exists():
            for p in cls_dir.rglob('*'):
                if p.suffix.lower() in IMG_EXTS and p.is_file():
                    files.append(p)
        mapping[cls] = sorted(files)
    return mapping

images_by_class = list_images_by_class(DATA_DIR, CLASSES)
for cls, files in images_by_class.items():
    print(f'{cls:14s}: {len(files)} images')
```

```
Bird-drop     : 207 images
Snow-Covered  : 123 images
Dusty         : 190 images
Clean         : 192 images
```

In [112…

```
def pick_one_per_class(images_by_class):
    picks = {}
    for cls, files in images_by_class.items():
        if not files:
            raise RuntimeError(f'No images in class {cls}. Put images und
        picks[cls] = files[0]
    return picks

picks = pick_one_per_class(images_by_class)
picks
```

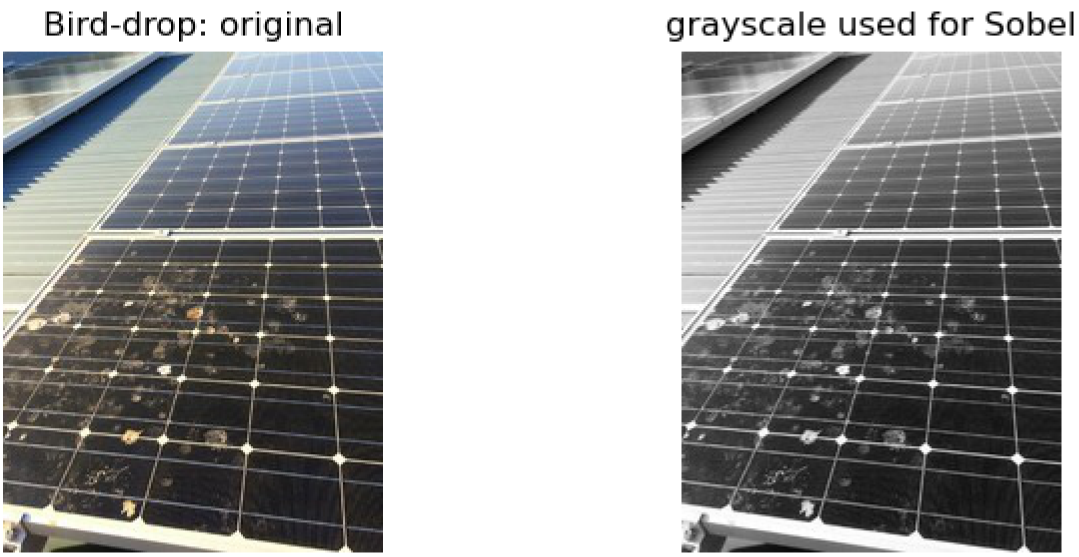
Out[112…

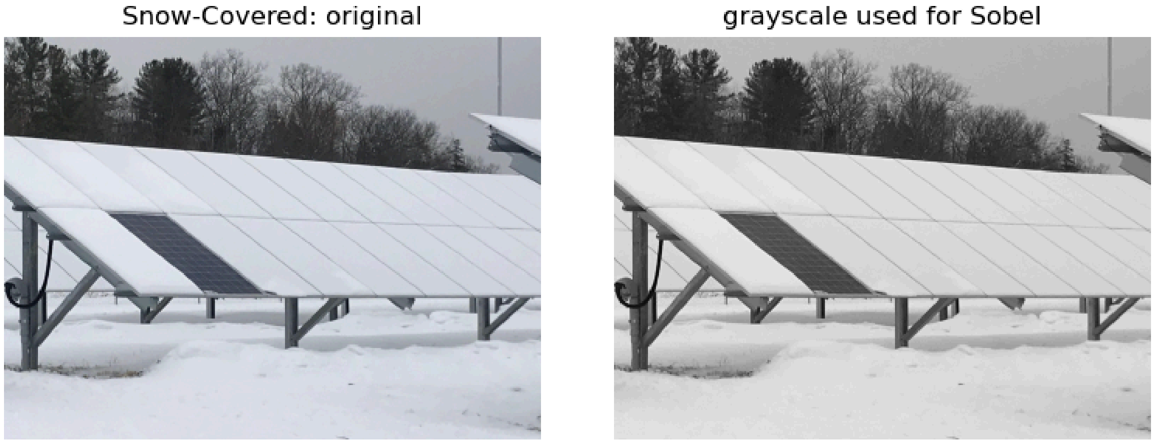
```
{'Bird-drop': PosixPath('/Users/muradullaev03/Downloads/Faulty_solar_pan
el/Bird-drop/Bird (1).jpeg'),
 'Snow-Covered': PosixPath('/Users/muradullaev03/Downloads/Faulty_solar_
panel/Snow-Covered/Snow (1).jpg'),
 'Dusty': PosixPath('/Users/muradullaev03/Downloads/Faulty_solar_panel/D
usty/Dust (1).jpg'),
 'Clean': PosixPath('/Users/muradullaev03/Downloads/Faulty_solar_panel/C
lean/Clean (1).jpeg')}
```

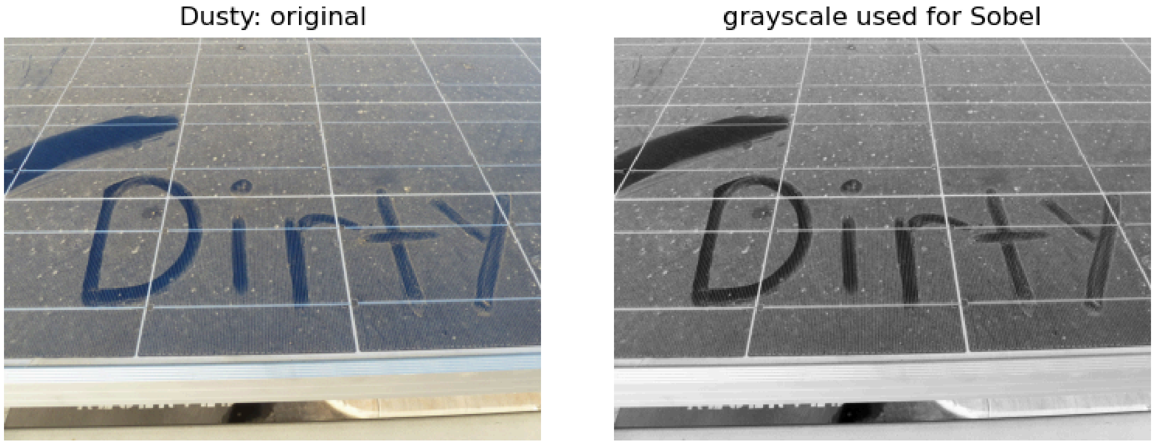
2(b)Grayscale demonstration

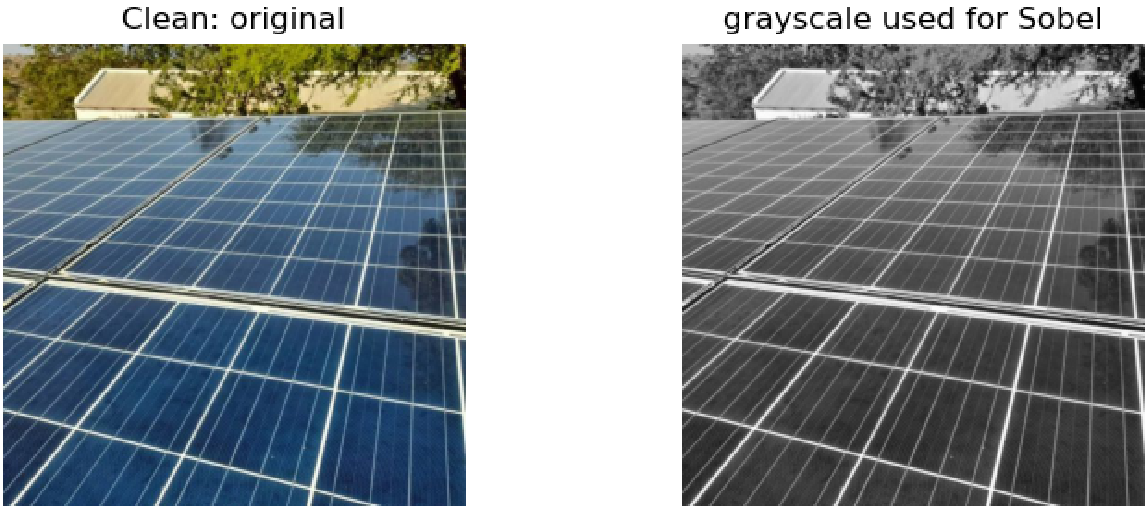
In [114…

```
for cls, p in picks.items():
    img = io.imread(p)
    if img.ndim == 3 and img.shape[-1] in (3,4):
        if img.shape[-1] == 4:
            img = img[..., :3]
        gray_demo = rgb2gray(img).astype('float32')
    else:
        gray_demo = img.astype('float32')
        if gray_demo.max() > 1.01:
            gray_demo = gray_demo/255.0
    fig, axs = plt.subplots(1,2, figsize=(8,3))
    axs[0].imshow(img if img.ndim==3 else gray_demo, cmap='gray')
    axs[0].set_title(f'{cls}: original'); axs[0].axis('off')
    axs[1].imshow(gray_demo, cmap='gray')
    axs[1].set_title('grayscale used for Sobel'); axs[1].axis('off')
    plt.tight_layout(); plt.show()
```







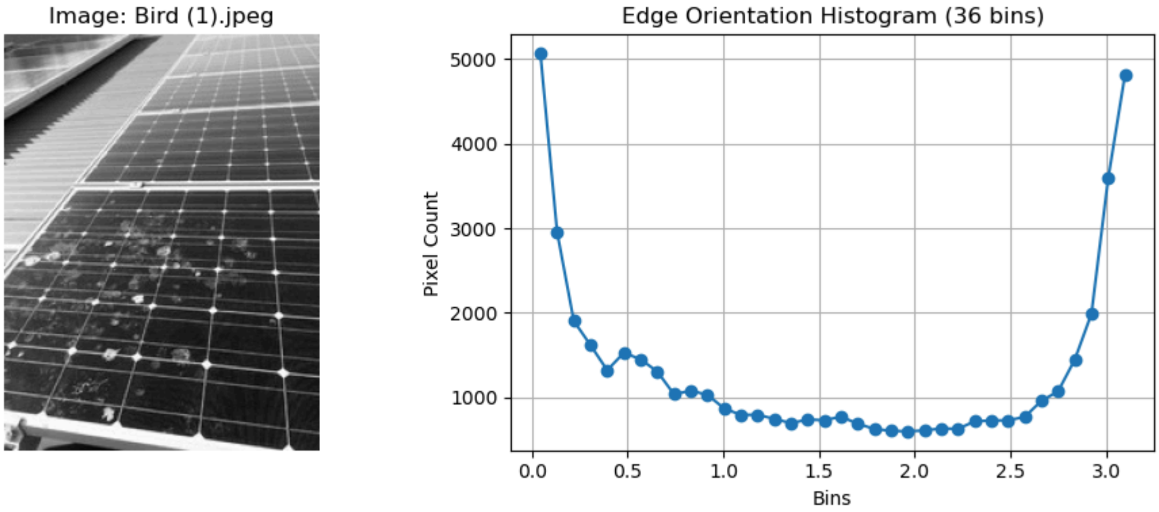


2(e) Edge-orientation histograms

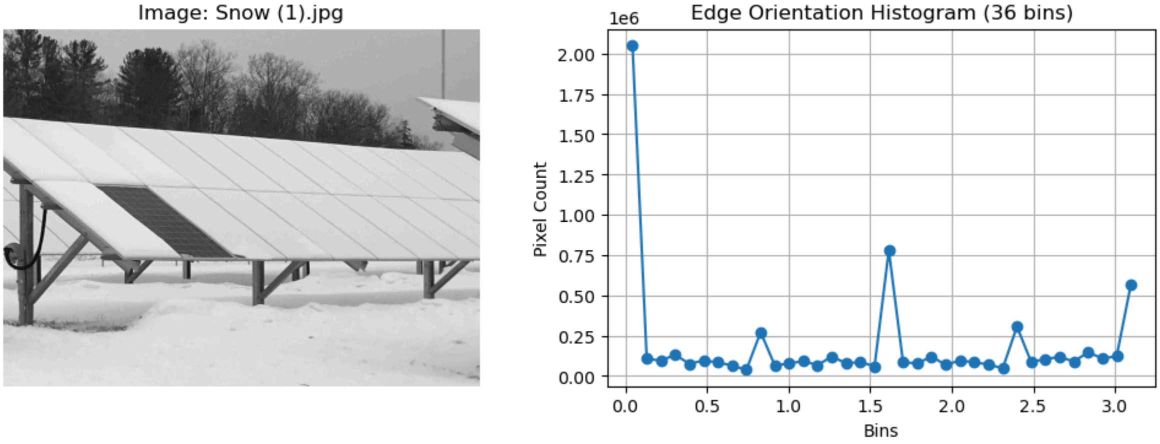
In [116…

```
def plot_image_and_hist(img_path):
    gray = load_image_gray(img_path)
    hist36, centers = grad_orientation_hist(gray, nbins=36)

    fig = plt.figure(figsize=(10,4))
    ax1 = plt.subplot(1,2,1)
    ax1.imshow(gray, cmap='gray'); ax1.set_title(f'Image: {img_path.name}

    ax2 = plt.subplot(1,2,2)
    ax2.plot(centers, hist36, marker='o')
    ax2.set_title('Edge Orientation Histogram (36 bins)')
    ax2.set_xlabel('Bins')
    ax2.set_ylabel('Pixel Count')
    ax2.grid(True)
    plt.tight_layout(); plt.show()
    return hist36, centers

hist_by_class = {}
centers_ref = None
for cls in CLASSES:
    print(f'=== {cls} ===')
    h, centers = plot_image_and_hist(picks[cls])
    hist_by_class[cls] = h.astype(float)
    if centers_ref is None:
        centers_ref = centers
    else:
        assert np.allclose(centers_ref, centers), "bin centers mismatch"

hist_by_class_norm = {cls: (h/(h.sum()+1e-8)) for cls, h in hist_by_class
```

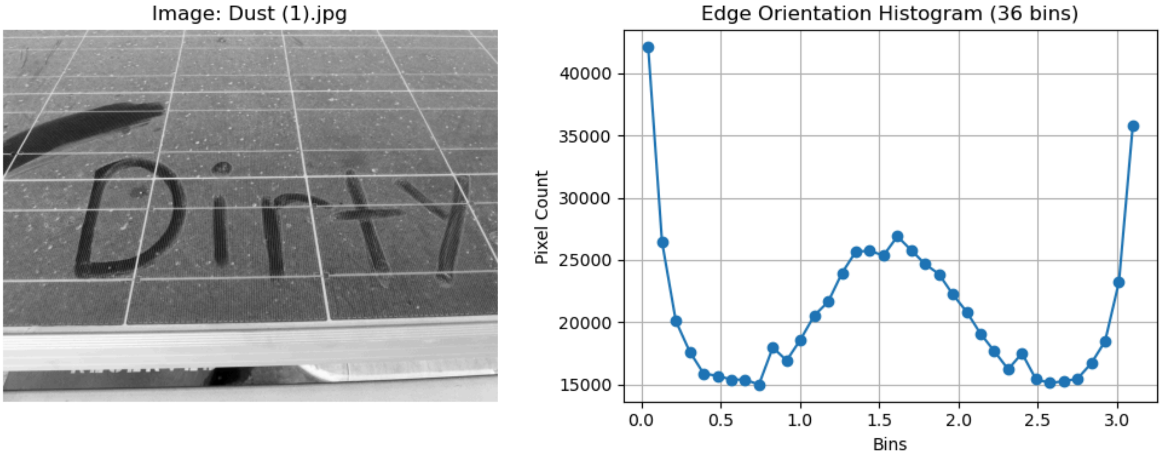
```
=== Bird-drop ===
```



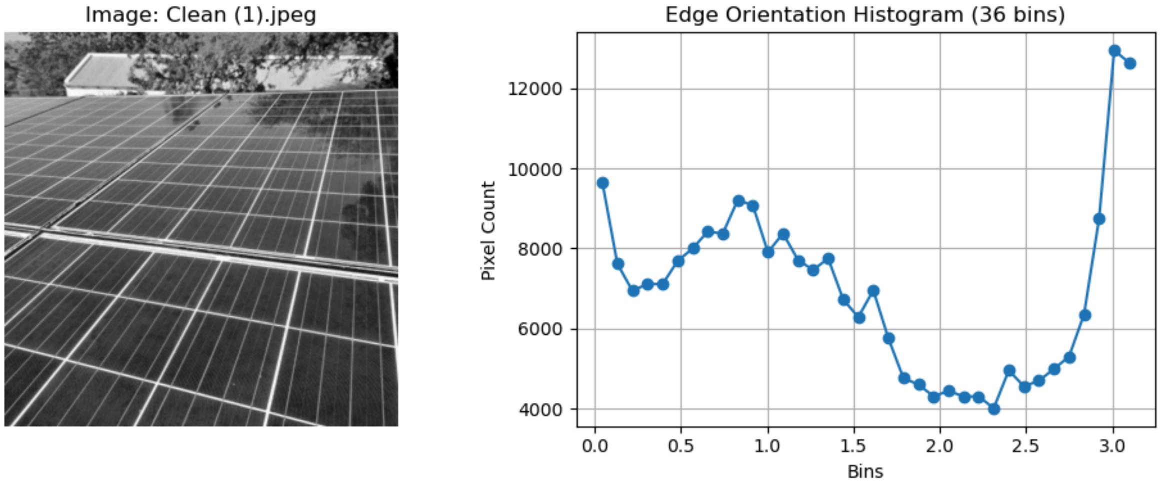
```
=== Snow-Covered ===
```



```
=== Dusty ===
```



```
=== Clean ===
```



2(f)Compare two histograms

In [118…

```
A, B = 'Bird-drop', 'Snow-Covered'

hA = hist_by_class_norm[A].reshape(1,-1)
hB = hist_by_class_norm[B].reshape(1,-1)

l2 = pairwise_distances(hA, hB, metric='euclidean')[0,0]
l1 = pairwise_distances(hA, hB, metric='manhattan')[0,0]
cos = pairwise_distances(hA, hB, metric='cosine')[0,0]

print(f'Comparing "{A}" vs "{B}":')
print(f'  Euclidean (L2):  {l2:.6f}')
print(f'  Manhattan (L1):  {l1:.6f}')
print(f'  Cosine distance: {cos:.6f}')
```

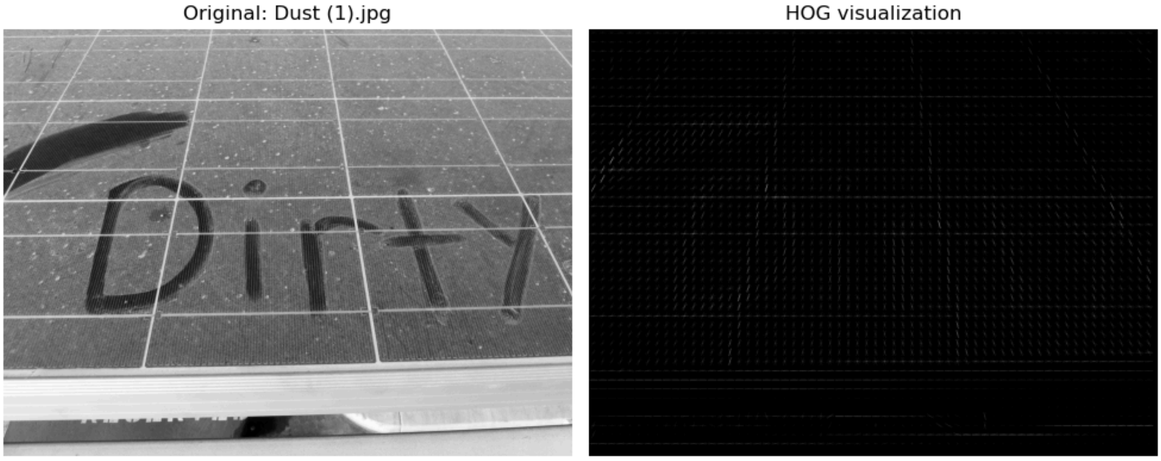
```
Comparing "Bird-drop" vs "Snow-Covered":
  Euclidean (L2):  0.243867
  Manhattan (L1):  0.707471
  Cosine distance: 0.274620
```

3(a)HOG descriptor + visualization

In [120…

```
hog_cls = 'Dusty'
hog_img_path = picks[hog_cls]

gray = load_image_gray(hog_img_path)
hog_vec, hog_vis = hog_descriptor_and_vis(gray)
print(f'HOG vector length: {len(hog_vec)}')

plt.figure(figsize=(10,4))
plt.subplot(1,2,1); plt.imshow(gray, cmap='gray'); plt.title(f'Original:
plt.subplot(1,2,2); plt.imshow(hog_vis, cmap='gray'); plt.title('HOG visu
plt.tight_layout(); plt.show()
```

```
HOG vector length: 98820
```

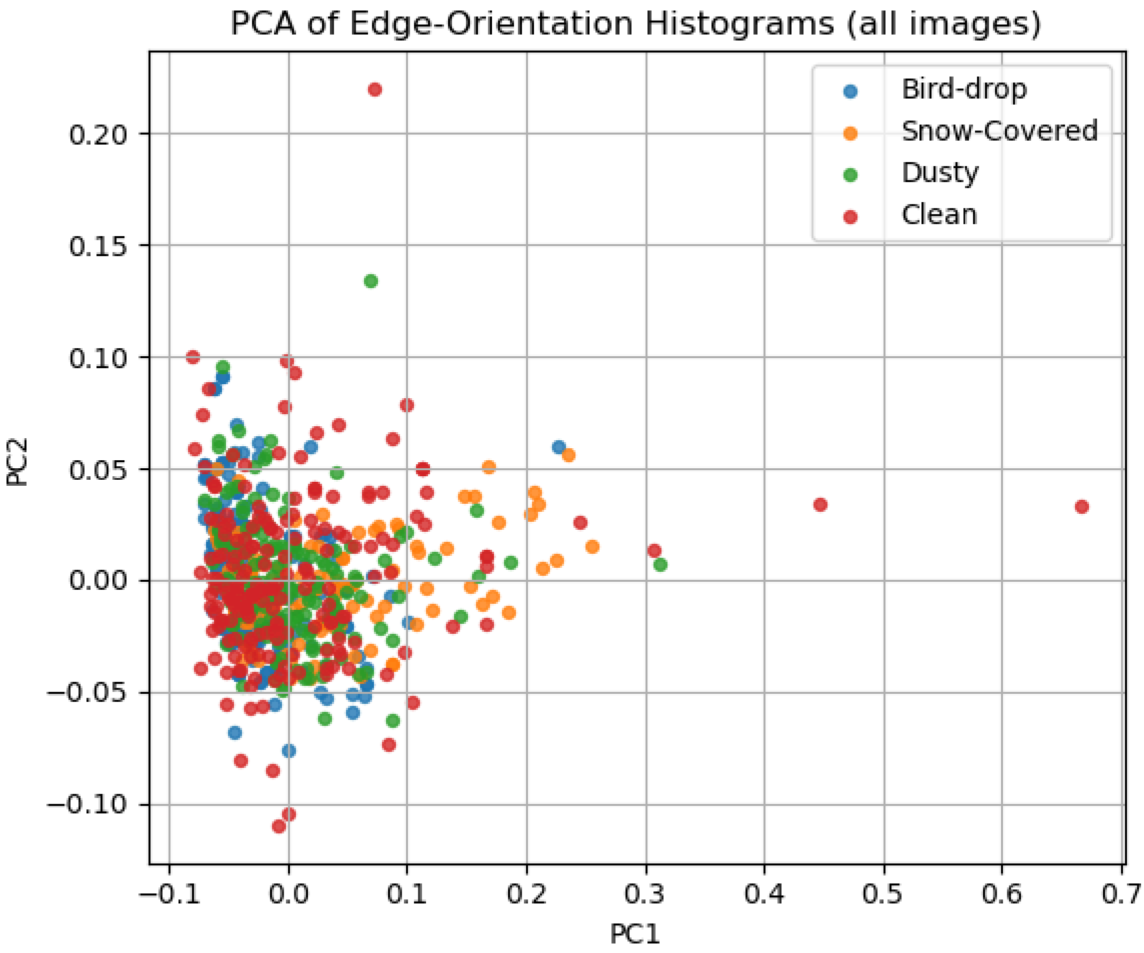


4(a–d)PCA on all images + scatter

In [122…

```
X, y, paths = [], [], []
nbins = 36
for cls, files in images_by_class.items():
    for p in files:
        try:
            g = load_image_gray(p)
            h, _ = grad_orientation_hist(g, nbins=nbins)
            h = h.astype(float); h = h/(h.sum()+1e-8)
            X.append(h); y.append(cls); paths.append(p)
        except Exception as e:
            print(f'[WARN] skip {p}: {e}')

if len(X) == 0:
    raise RuntimeError("No features built.")

X = np.vstack(X); y = np.array(y)
print("Feature matrix shape:", X.shape)
print("Classes:", {c: int((y==c).sum()) for c in CLASSES})

pca = PCA(n_components=2, random_state=42)
X2 = pca.fit_transform(X)
print("Explained variance ratio (PC1, PC2):", pca.explained_variance_rati
print("Total explained:", float(pca.explained_variance_ratio_.sum()))

plt.figure(figsize=(6,5))
for cls in CLASSES:
    m = (y == cls)
    plt.scatter(X2[m,0], X2[m,1], s=18, label=cls, alpha=0.8)
plt.xlabel('PC1'); plt.ylabel('PC2'); plt.title('PCA of Edge-Orientation
plt.legend(); plt.grid(True); plt.tight_layout(); plt.show()
```

```
Feature matrix shape: (713, 36)
Classes: {'Bird-drop': 207, 'Snow-Covered': 123, 'Dusty': 190, 'Clean': 19
3}
Explained variance ratio (PC1, PC2): [0.55413587 0.12717299]
Total explained: 0.6813088581600473
```



Answer for 4(d): Visual separability

Number of fully visually separable (non-overlapping) classes: **0**.

Observation: clusters show partial overlap; *Snow-Covered* tends to shift along PC1, *Clean* has a higher mean on PC2, while *Dusty* is near the centre and *Bird-drop* shifts left on PC1.

Text Dataset (Tweets)

Using the training set only. Default parameters for `CountVectorizer` and `TfidfVectorizer`.

In [125…

```
import json, pandas as pd
from pathlib import Path

DATA_TEXT = Path('/Users/muradullaev03/Downloads/Faulty_solar_panel/train

def load_json_or_jsonl(path: Path):
    with open(path, 'r', encoding='utf-8') as f:
        txt = f.read().strip()
    try:
        obj = json.loads(txt)
        if isinstance(obj, list):
            return obj
        elif isinstance(obj, dict):
            return [obj]
    except json.JSONDecodeError:
        pass
    rows = []
    with open(path, 'r', encoding='utf-8') as f:
        for line in f:
            line = line.strip()
            if line:
                rows.append(json.loads(line))
    return rows

rows = load_json_or_jsonl(DATA_TEXT)
df_text = pd.DataFrame(rows)
print("Raw df_text shape:", df_text.shape)
print("Columns:", list(df_text.columns)[:20])

emotions = ['anger','anticipation','disgust','fear','joy','love',
            'optimism','pessimism','sadness','surprise','trust']

if 'Tweet' in df_text.columns and set(emotions).issubset(df_text.columns)
    pos_counts = df_text[emotions].sum(axis=1)
    df_single = df_text[pos_counts == 1].copy()
    df_single['label'] = df_single[emotions].idxmax(axis=1)
    df_single['text']  = df_single['Tweet']
    df_text = df_single[['text','label']].reset_index(drop=True)
elif {'text','label'}.issubset(df_text.columns):
    df_text = df_text[['text','label']].reset_index(drop=True)
else:
    raise ValueError("Cannot identify format")

print("After adapter:", df_text.shape)
df_text.head()
```

```
Raw df_text shape: (3000, 13)
Columns: ['ID', 'Tweet', 'anger', 'anticipation', 'disgust', 'fear', 'jo
y', 'love', 'optimism', 'pessimism', 'sadness', 'surprise', 'trust']
After adapter: (428, 2)
```

Out[125…

	text	label
0	Tears and eyes can dry but I won't, I'm burnin...	anger
1	@JustinRow10 madden is the reason why controll...	disgust
2	10 page script due Friday for class. Who said ...	fear
3	@HillaryClinton Yeah U gotta make them HEEL. I...	anticipation
4	When you saw a t-shirt with the phrase 'My min...	sadness

In [126…

```
chosen = ['joy', 'sadness', 'anger', 'fear']

if not df_text.empty:
    df4 = df_text[df_text['label'].isin(chosen)].copy()
    df4['label'] = df4['label'].astype('category')
    print(df4['label'].value_counts())
    print("Filtered shape:", df4.shape)
else:
    df4 = pd.DataFrame(columns=['text','label'])
```

```
label
joy        127
fear        72
sadness     70
anger       50
Name: count, dtype: int64
Filtered shape: (319, 2)
```

Selected 4 classes (text): joy, sadness, anger, fear.

In [128…

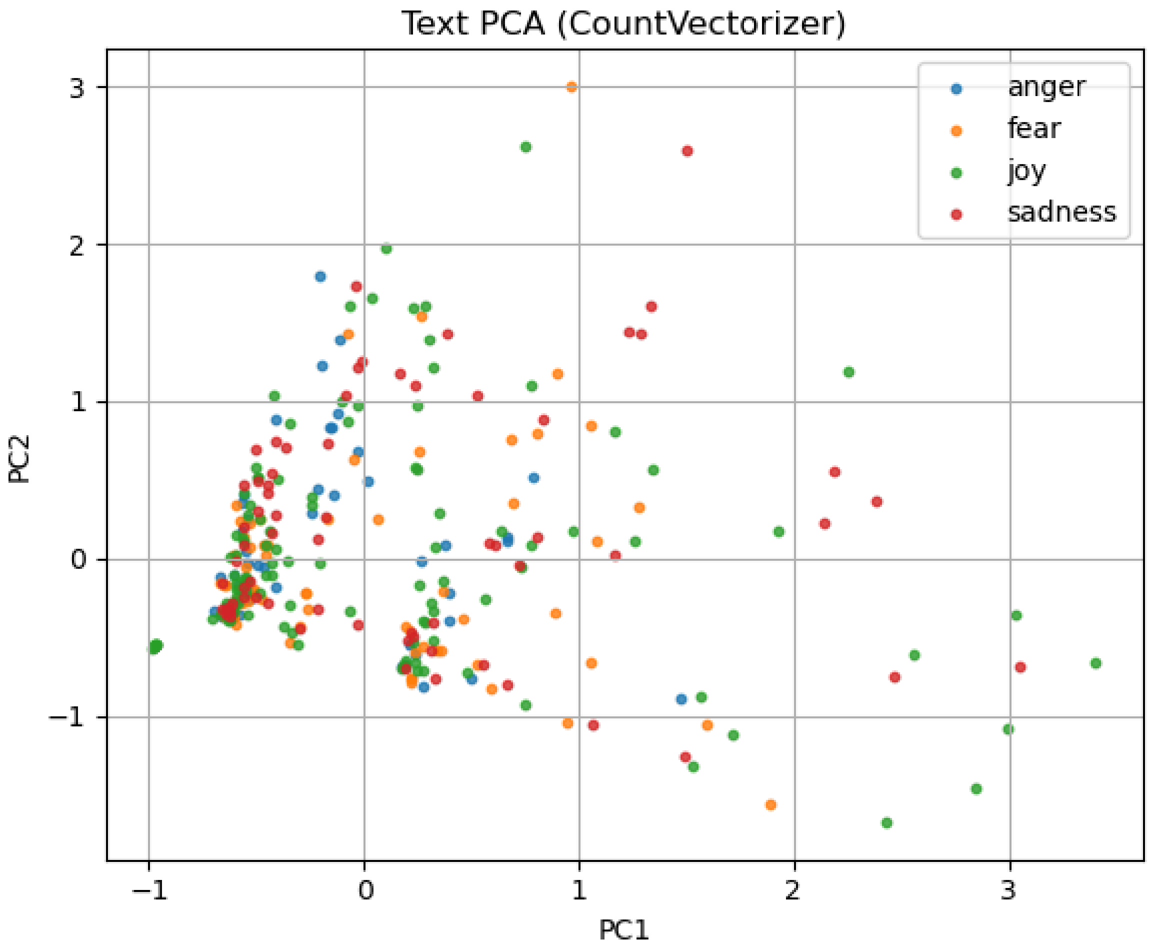
```
if not df4.empty:
    count_vec = CountVectorizer()
    tfidf_vec = TfidfVectorizer()

    X_count = count_vec.fit_transform(df4['text'])
    X_tfidf = tfidf_vec.fit_transform(df4['text'])
    y_text = df4['label'].values

    print("Count shape:", X_count.shape)
    print("TFIDF shape:", X_tfidf.shape)
```

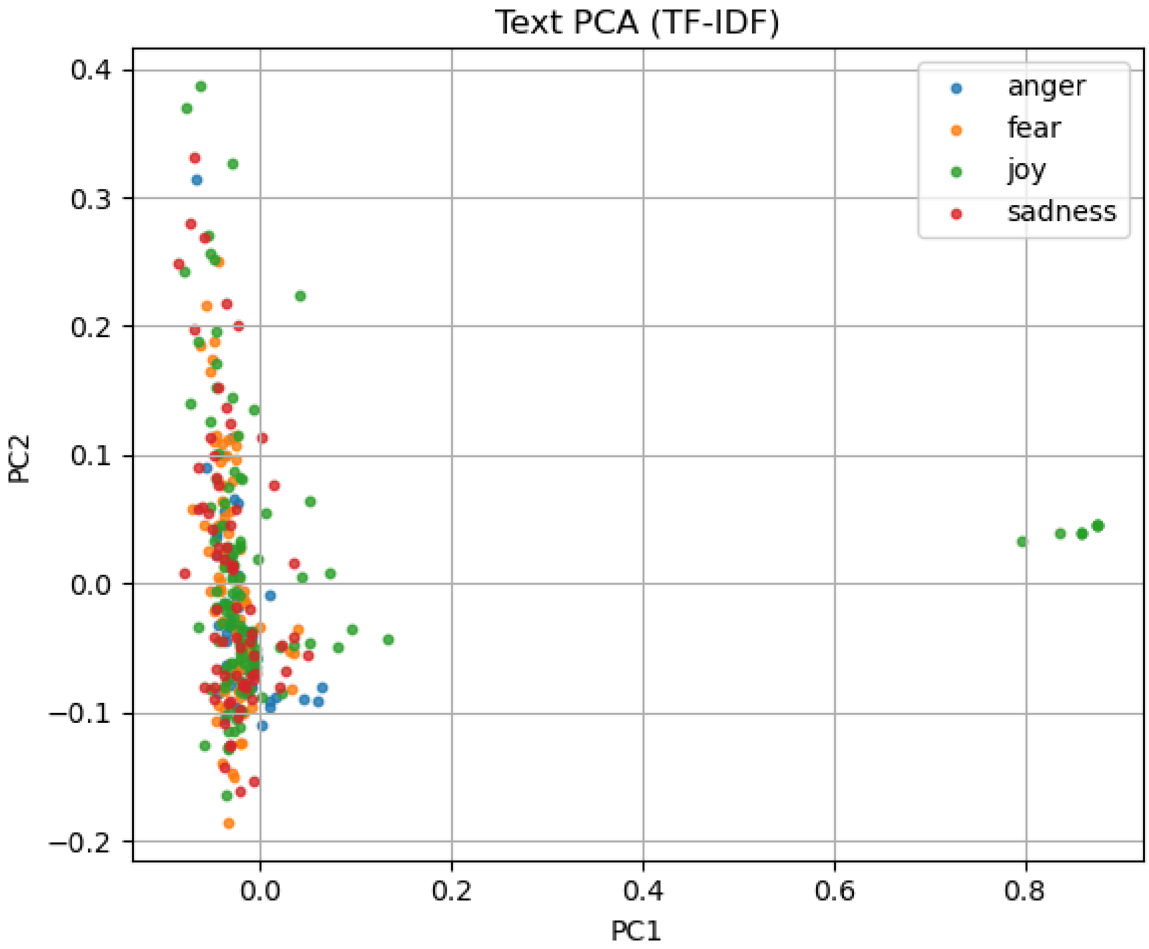
```
Count shape: (319, 1907)
TFIDF shape: (319, 1907)
```

Vector dimensions: Count = (N, D1); TF-IDF = (N, D2) — as printed above.

In [130…

```
from sklearn.decomposition import PCA

if 'X_count' in globals():
    Xc = X_count.astype('float32').toarray()
    Xt = X_tfidf.astype('float32').toarray()

    pca_c = PCA(n_components=2, random_state=42).fit_transform(Xc)
    pca_t = PCA(n_components=2, random_state=42).fit_transform(Xt)

    def scatter_pca(X2, labels, title):
        plt.figure(figsize=(6,5))
        labs = pd.Series(labels)
        for cls in sorted(labs.unique()):
            m = (labs == cls).values
            plt.scatter(X2[m,0], X2[m,1], s=10, label=cls, alpha=0.8)
        plt.xlabel('PC1'); plt.ylabel('PC2'); plt.title(title)
        plt.legend(); plt.grid(True); plt.tight_layout(); plt.show()

    scatter_pca(pca_c, y_text, 'Text PCA (CountVectorizer)')
    scatter_pca(pca_t, y_text, 'Text PCA (TF-IDF)')
```


Text PCA (CountVectorizer)
PC1
PC2
−1
0
1
2
3
−1
0
1
2
3
anger
fear
joy
sadness


Text PCA (TF-IDF)
PC1
PC2
0.0
0.2
0.4
0.6
0.8
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
anger
fear
joy
sadness

Visual separability (text, both plots):

Selected classes: joy, sadness, anger, fear.

On both PCA plots, visually separable classes: (fill based on your plots, or 'none').

Strongest overlaps: (state the pairs that overlap the most).

(Next steps if desired: text cleaning, stopwords tuning, n-grams, lemmatisation, min_df/max_df.)

In [ ]:

```

```