```
# face_recognition_pipeline.py

import os
import shutil
import numpy as np
import pandas as pd
from deepface import DeepFace
from sklearn.model_selection import train_test_split

# --- Configuration ---
# Update these paths to match your local file structure.
# For example, on Windows:
# BASE_DIR = "C:\\Users\\YourUsername\\Data"
# EXCEL_PATH = "C:\\Users\\YourUsername\\Data\\dataset.xlsx"
BASE_DIR = "/content/drive/MyDrive/Data"
EXCEL_PATH = "/content/drive/MyDrive/Data/dataset.xlsx"
OUT_ROOT = "/content/face_data"

# --- Main Functions ---

def build_dataset_df(base_dir, excel_path=None):
    """
    Builds a DataFrame of image paths and identities from a directory.

    Args:
        base_dir (str): The root directory containing identity subfolders.
        excel_path (str, optional): Path to a metadata Excel file.

    Returns:
        pd.DataFrame: A DataFrame with 'path' and 'identity' columns.
    """
    if excel_path and os.path.exists(excel_path):
        try:
            df_meta = pd.read_excel(excel_path)
            df_meta.columns = [c.strip().lower() for c in df_meta.columns]
            df_meta["name"] = df_meta["name"].astype(str).str.strip()
            print("Metadata loaded successfully.")
        except FileNotFoundError:
```

```
            print(f"Metadata file not found at {excel_path}.")
            df_meta = None
    else:
        df_meta = None

    def is_img(f):
        return f.lower().endswith((".jpg", ".jpeg", ".png", ".webp"))

    rows = []
    if not os.path.isdir(base_dir):
        print(f"Base directory not found at {base_dir}. Cannot build dataset table.")
        return pd.DataFrame(rows)

    for folder in os.listdir(base_dir):
        fpath = os.path.join(base_dir, folder)
        if not os.path.isdir(fpath):
            continue
        for fn in os.listdir(fpath):
            if not is_img(fn):
                continue
            rows.append({"path": os.path.join(fpath, fn), "identity": folder})

    df = pd×DataFrame(rows)
    print(f"Identities: {df['identity'].nunique()} | Images: {len(df)}")
    return df

def split_and_copy_data(df, out_root):
    """
    Splits the DataFrame into training and validation sets and copies files.

    Args:
        df (pd.DataFrame): The main dataset DataFrame.
        out_root (str): The root directory for the train/val split.
    """
    if df.empty:
        print("Dataset is empty. Skipping split and copy.")
        return None, None
```

```
    try:
        train_df, val_df = train_test_split(df, test_size=0.2, stratify=df["identity"],
random_state=42)
    except ValueError as e:
        print(f"Could not perform stratified split. Error: {e}")
        print("Falling back to a non-stratified split.")
        train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)

    def copy_split(split_df, root):
        if split_df.empty:
            print(f"Warning: The DataFrame for {root} is empty. No files to copy.")
            return
        for _, r in split_df.iterrows():
            dst_dir = os.path.join(root, r["identity"])
            os.makedirs(dst_dir, exist_ok=True)
            shutil.copy2(r["path"], os.path.join(dst_dir, os.path.basename(r["path"])))

    TRAIN_DIR = os.path.join(out_root, "train")
    VAL_DIR = os.path.join(out_root, "val")

    for d in (TRAIN_DIR, VAL_DIR):
        if os.path.exists(d):
            shutil.rmtree(d)
        os.makedirs(d)

    copy_split(train_df, TRAIN_DIR)
    copy_split(val_df, VAL_DIR)

    print(f"Train IDs: {len(os.listdir(TRAIN_DIR))} | Val IDs: {len(os.listdir(VAL_DIR))}")
    return TRAIN_DIR, VAL_DIR

def build_facebank(root, model="Facenet", detector="mtcnn"):
    """
    Creates a facebank of average embeddings (centroids) for each identity.
    """
    facebank, idents = {}, []
    if not os.path.isdir(root):
        print(f"Facebank directory not found at {root}. Cannot build facebank.")
```

```
        return facebank, idents

    for ident in sorted(os.listdir(root)):
        person_dir = os.path.join(root, ident)
        if not os.path.isdir(person_dir):
            continue

        try:
            reps = DeepFace.represent(
                img_path=[os.path.join(person_dir, f) for f in os.listdir(person_dir)],
                model_name=model,
                detector_backend=detector,
                enforce_detection=False
            )

            if not reps:
                print(f"Warning: No faces found for identity '{ident}'. Skipping.")
                continue

            if isinstance(reps, dict): reps = [reps]
            embs = [np.array(r["embedding"], dtype="float32") for r in reps if "embedding" in r]
            if not embs:
                print(f"Warning: No valid embeddings found for identity '{ident}'. Skipping.")
                continue

            centroid = np×mean(embs, axis=0)
            centroid = centroid / (np.linalg.norm(centroid) + 1e-8)

            facebank[ident] = centroid
            idents.append(ident)
        except Exception as e:
            print(f"Error processing identity '{ident}': {e}")

    return facebank, idents

def evaluate_model(val_dir, facebank, idents):
    """
    Evaluates the face recognition system on the validation set.
```

```
"""
def cosine(a, b):
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b) + 1e-8)

total, correct = 0, 0
if not os.path.isdir(val_dir):
    print(f"Validation directory not found at {val_dir}. Skipping evaluation.")
    return

for ident in os.listdir(val_dir):
    person_dir = os.path.join(val_dir, ident)
    if not os.path.isdir(person_dir):
        continue

    for fn in os.listdir(person_dir):
        img_path = os.path.join(person_dir, fn)
        try:
            rep = DeepFace×represent(
                img_path=img_path,
                model_name="Facenet",
                detector_backend="mtcnn",
                enforce_detection=False
            )

            if not rep or "embedding" not in rep[0]:
                continue

            emb = np×array(rep[0]["embedding"], dtype="float32")
            emb = emb / (np.linalg.norm(emb) + 1e-8)

            if not idents:
                print("No identities enrolled in the facebank. Cannot evaluate.")
                break

            sims = np.array([cosine(emb, facebank[idt]) for idt in idents])
            pred = idents[np.argmax(sims)]

            total += 1
```

```
            correct += int(pred == ident)
        except Exception as e:
            print(f"Error processing image {img_path}: {e}")

    acc = correct / max(total, 1)
    print(f"Validation Accuracy: {acc:.4f} ({correct}/{total})")

def verify_face(username, image_path, facebank, threshold=0.4):
    """
    Verifies if a face in an image matches an enrolled user.
    """
    def cosine(a, b):
        return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b) + 1e-8)

    if username not in facebank:
        return {"decision": "DENIED", "reason": "user_not_enrolled"}

    try:
        rep = DeepFace×represent(
            img_path=image_path,
            model_name="Facenet",
            detector_backend="mtcnn",
            enforce_detection=False
        )

        if not rep or "embedding" not in rep[0]:
            return {"decision": "DENIED", "reason": "no_embedding"}

        emb = np×array(rep[0]["embedding"], dtype="float32")
        emb = emb / (np×linalg×norm(emb) + 1e-8)

        score = cosine(emb, facebank[username])
        decision = "GRANTED" if score >= threshold else "DENIED"

        return {"username": username, "score": round(score, 4), "decision": decision}
    except Exception as e:
        print(f"Error during verification: {e}")
        return {"decision": "DENIED", "reason": "processing_error"}
```

```
# --- Execution ---

def main():
    """Runs the entire face recognition pipeline."""
    # 1. Ensure libraries are installed (uncomment if needed)
    # !pip install deepface==0.0.79 pillow==9.5.0

    print("--- Starting Face Recognition Pipeline ---")

    # 2. Build dataset DataFrame and prepare directories
    df = build_dataset_df(BASE_DIR, EXCEL_PATH)
    if df.empty:
        print("No data found. Exiting.")
        return

    os.makedirs(OUT_ROOT, exist_ok=True)
    train_dir, val_dir = split_and_copy_data(df, OUT_ROOT)

    if not train_dir or not val_dir:
        print("Data split failed. Exiting.")
        return

    # 3. Build the facebank from the training set
    print("\n--- Building Facebank ---")
    facebank, idents = build_facebank(train_dir)
    print(f"Enrolled identities: {len(idents)}")

    if not facebank:
        print("Facebank is empty. Cannot proceed with evaluation or verification.")
        return

    # 4. Evaluate on the validation set
    print("\n--- Evaluating on Validation Set ---")
    evaluate_model(val_dir, facebank, idents)

    # 5. Run a verification example
    print("\n--- Running Verification Example ---")
```

```
    if not val_df.empty:
        test_identity = val_df["identity"].iloc[0]
        test_img = val_df["path"].iloc[0]
        print(f"Verifying identity: '{test_identity}' with image: '{os.path.basename(test_img)}'")
        result = verify_face(test_identity, test_img, facebank)
        print(result)
    else:
        print("Validation DataFrame is empty. Cannot run verification example.")

if __name__ == "__main__":
    main()
```