```
#!/usr/bin/env python3
"""
Task 6: K-Nearest Neighbors (KNN) Classification
Dataset: Iris (local iris.csv)
Requirements satisfied:
- Normalize features
- Try different K values
- Evaluate with accuracy + confusion matrix
- Visualize decision boundaries (2D using petal_length vs petal_width)
"""
import os
import argparse
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

RANDOM_STATE = 42
OUTPUT_DIR = "outputs"
DATA_PATH = "iris.csv"

def load_data(path: str) -> pd.DataFrame:
    df = pd.read_csv(path)
    # Try to standardize column names
    df.columns = [c.strip().lower().replace(" (cm)","").replace(" ","_") for c in df.columns]
```

```python
    # If iris from sklearn, expected columns:
    # sepal length (cm), sepal width (cm), petal length (cm), petal width (cm), species
    # After normalization: sepal_length, sepal_width, petal_length, petal_width, species
    return df


def split_features_labels(df: pd.DataFrame):
    X = df.drop(columns=[df.columns[-1]]).values
    y = df.iloc[:, -1].values
    return X, y


def tune_k(X_train, y_train, X_val, y_val, k_max=30):
    ks = list(range(1, k_max + 1))
    accs = []
    for k in ks:
        clf = KNeighborsClassifier(n_neighbors=k)
        clf.fit(X_train, y_train)
        preds = clf.predict(X_val)
        accs.append(accuracy_score(y_val, preds))
    best_k = ks[int(np.argmax(accs))]
    return best_k, ks, accs


def plot_accuracy_vs_k(ks, accs, path):
    plt.figure()
    plt.plot(ks, accs, marker='o')
    plt.xlabel("K (neighbors)")
    plt.ylabel("Accuracy")
    plt.title("Accuracy vs K")
    plt.grid(True, linewidth=0.4, linestyle='--')
    plt.tight_layout()
```

```python
    plt.savefig(path, dpi=200)
    plt.close()

def plot_confusion_matrix(cm, class_names, path):
    fig, ax = plt.subplots(figsize=(5.5, 4.5))
    im = ax.imshow(cm, interpolation='nearest')
    ax.set_title("Confusion Matrix")
    ax.set_xlabel("Predicted")
    ax.set_ylabel("Actual")
    ax.set_xticks(np.arange(len(class_names)))
    ax.set_yticks(np.arange(len(class_names)))
    ax.set_xticklabels(class_names, rotation=45, ha="right")
    ax.set_yticklabels(class_names)
    # annotate
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, cm[i, j], ha="center", va="center")
    fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
    plt.tight_layout()
    plt.savefig(path, dpi=200)
    plt.close()

def plot_decision_boundary(X2, y, best_k, path):
    """
    X2: two-feature matrix (petal_length, petal_width)
    y: labels
    """
    # Standardize
    scaler = StandardScaler()
```

```
    X2s = scaler.fit_transform(X2)

    # Train KNN with best_k
    clf = KNeighborsClassifier(n_neighbors=best_k)
    clf.fit(X2s, y)

    # Mesh grid
    x_min, x_max = X2s[:, 0].min() - 1, X2s[:, 0].max() + 1
    y_min, y_max = X2s[:, 1].min() - 1, X2s[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 400),
                         np.linspace(y_min, y_max, 400))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    # Plot
    plt.figure(figsize=(6,5))
    plt.contourf(xx, yy, Z, alpha=0.35)
    plt.scatter(X2s[:, 0], X2s[:, 1], c=pd.factorize(y)[0], edgecolor='k')
    plt.title(f"Decision Boundary (K={best_k}) on Petal features")
    plt.xlabel("petal_length (standardized)")
    plt.ylabel("petal_width (standardized)")
    plt.tight_layout()
    plt.savefig(path, dpi=200)
    plt.close()

def main(args):
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    df = load_data(args.data)

    # Split
```

```
X, y = split_features_labels(df)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=RANDOM_STATE, stratify=y
)

# Normalize
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

# Tune K on the test split (simple hold-out for the task requirement)
best_k, ks, accs = tune_k(X_train_s, y_train, X_test_s, y_test, k_max=args.kmax)

# Train with best K
model = KNeighborsClassifier(n_neighbors=best_k)
model.fit(X_train_s, y_train)
preds = model.predict(X_test_s)

# Metrics
acc = accuracy_score(y_test, preds)
cm = confusion_matrix(y_test, preds)
report = classification_report(y_test, preds)

# Plots
plot_accuracy_vs_k(ks, accs, os.path.join(OUTPUT_DIR, "accuracy_vs_k.png"))
plot_confusion_matrix(cm, sorted(np.unique(y)), os.path.join(OUTPUT_DIR, "confusion_matrix.png"))

# Decision boundary on two features (petal_length, petal_width)
# Try common Iris names; fall back by position
```

```
    cols = [c for c in df.columns if c != df.columns[-1]]
    col_map = {c:c for c in cols}
    # standard column hints
    for c in cols:
        if "petal_length" in c or "petal length" in c:
            col_map["petal_length"] = c
        if "petal_width" in c or "petal width" in c:
            col_map["petal_width"] = c
    c1 = col_map.get("petal_length", cols[-2])
    c2 = col_map.get("petal_width", cols[-1])
    X2 = df[[c1, c2]].values
    plot_decision_boundary(X2, df.iloc[:, -1].values, best_k, os.path.join(OUTPUT_DIR,
"decision_boundary.png"))

    # Save report
    with open(os.path.join(OUTPUT_DIR, "report.txt"), "w", encoding="utf-8") as f:
        f.write(f"Best K: {best_k}\n")
        f.write(f"Test Accuracy: {acc:.4f}\n\n")
        f.write(report)

    print(f"Best K: {best_k}")
    print(f"Test Accuracy: {acc:.4f}")
    print(report)
    print("Artifacts saved to 'outputs/'")


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--data", type=str, default=DATA_PATH, help="Path to CSV dataset (last column
= label)")
```

```
parser.add_argument("--kmax", type=int, default=30, help="Max K to try")
main(parser.parse_args())
```