\
"""
Decision Trees & Random Forests
Task 5 — Heart Disease Classification
Author: [Your Name]
Date: [Today's Date]

This script performs:
- Data loading and basic cleaning
- Train/test split and optional scaling
- Decision Tree training with depth tuning (GridSearchCV)
- Random Forest training and comparison
- Cross-validation evaluation
- Visualization: decision tree plot, feature importances, accuracy comparison
- Saves outputs as PNG files and a cleaned dataset as CSV for reproducibility

Usage:
1. Download the Heart Disease dataset (CSV) and save as `heart.csv` in the same folder.
2. Run: python decision_tree_random_forest.py
"""

```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, StratifiedKFold
```

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import os

# --- Configuration ---
DATA_PATH = "heart.csv"  # expected dataset filename
RANDOM_STATE = 42
OUTPUT_DIR = "outputs"
os.makedirs(OUTPUT_DIR, exist_ok=True)

def load_and_clean(path):
    df = pd.read_csv(path)
    # Basic cleaning: drop completely empty columns, and rows with too many missing values
    df = df.dropna(axis=1, how='all')
    # If any missing values, fill numerical with median and categorical with mode
    for col in df.columns:
        if df[col].isnull().sum() > 0:
            if df[col].dtype in [np.float64, np.int64]:
                df[col].fillna(df[col].median(), inplace=True)
            else:
                df[col].fillna(df[col].mode()[0], inplace=True)
    return df

def prepare_features(df):
    # Assume the target column is named 'target' or 'heart_disease' or 'Target' etc.
    target_candidates = ['target', 'Target', 'heart_disease', 'HeartDisease', 'label', 'condition']
    target_col = None
```

```
    for t in target_candidates:
        if t in df.columns:
            target_col = t
            break
    if target_col is None:
        # fallback: if last column is binary, use it
        last_col = df.columns[-1]
        if set(df[last_col].unique()) <= set([0,1]) or df[last_col].nunique() <= 3:
            target_col = last_col
        else:
            raise ValueError("Target column not found. Please ensure your CSV has a binary target column named 'target' or similar.")

    X = df.drop(columns=[target_col])
    y = df[target_col].astype(int)

    # Convert categorical columns with low cardinality to dummies
    cat_cols = [c for c in X.columns if X[c].dtype == object or X[c].nunique() <= 10]
    X = pd.get_dummies(X, columns=cat_cols, drop_first=True)

    return X, y, target_col

def scale_if_needed(X_train, X_test):
    scaler = StandardScaler()
    numeric_cols = X_train.select_dtypes(include=[np.number]).columns.tolist()
    if numeric_cols:
        X_train[numeric_cols] = scaler.fit_transform(X_train[numeric_cols])
        X_test[numeric_cols] = scaler.transform(X_test[numeric_cols])
    return X_train, X_test
```

```
def train_decision_tree(X_train, y_train, X_val, y_val):
    dt = DecisionTreeClassifier(random_state=RANDOM_STATE)
    param_grid = {'max_depth': [3, 5, 7, 9, None], 'min_samples_split': [2, 5, 10]}
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
    grid = GridSearchCV(dt, param_grid, cv=cv, scoring='accuracy', n_jobs=-1)
    grid.fit(X_train, y_train)
    best = grid.best_estimator_
    val_preds = best.predict(X_val)
    val_acc = accuracy_score(y_val, val_preds)
    return best, grid.best_params_, val_acc

def train_random_forest(X_train, y_train, X_val, y_val):
    rf = RandomForestClassifier(random_state=RANDOM_STATE, n_jobs=-1)
    param_grid = {'n_estimators': [100, 200], 'max_depth': [5, 10, None], 'max_features': ['sqrt', None]}
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
    grid = GridSearchCV(rf, param_grid, cv=cv, scoring='accuracy', n_jobs=-1)
    grid.fit(X_train, y_train)
    best = grid.best_estimator_
    val_preds = best.predict(X_val)
    val_acc = accuracy_score(y_val, val_preds)
    return best, grid.best_params_, val_acc

def cross_validate_model(model, X, y):
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
    scores = cross_val_score(model, X, y, cv=cv, scoring='accuracy', n_jobs=-1)
    return scores

def plot_and_save_tree(model, feature_names, filename):
```

```
    plt.figure(figsize=(18,10))
    plot_tree(model, feature_names=feature_names, class_names=['0','1'], filled=True, proportion=True, rounded=True, fontsize=8)
    plt.title("Decision Tree")
    plt.tight_layout()
    plt.savefig(filename)
    plt.close()

def plot_feature_importances(importances, feature_names, filename, top_n=20):
    fi = pd.Series(importances, index=feature_names).sort_values(ascending=False).head(top_n)
    plt.figure(figsize=(10,6))
    sns.barplot(x=fi.values, y=fi.index)
    plt.title("Feature Importances")
    plt.xlabel("Importance")
    plt.tight_layout()
    plt.savefig(filename)
    plt.close()

def main():
    print("Loading dataset...")
    df = load_and_clean(DATA_PATH)
    print("Dataset shape:", df.shape)

    X, y, target_col = prepare_features(df)
    print("Using target column:", target_col)
    # save cleaned version for reproducibility
    cleaned_path = os.path.join(OUTPUT_DIR, "cleaned_heart.csv")
    df.to_csv(cleaned_path, index=False)
    print("Saved cleaned dataset to:", cleaned_path)
```

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

    X_train, X_test = scale_if_needed(X_train.copy(), X_test.copy())


    print("Training Decision Tree with GridSearchCV...")

    dt_model, dt_params, dt_acc = train_decision_tree(X_train, y_train, X_test, y_test)

    print("Best Decision Tree params:", dt_params)

    print("Decision Tree validation accuracy:", dt_acc)


    print("Training Random Forest with GridSearchCV...")

    rf_model, rf_params, rf_acc = train_random_forest(X_train, y_train, X_test, y_test)

    print("Best Random Forest params:", rf_params)

    print("Random Forest validation accuracy:", rf_acc)


    # Cross-validate best models on full training data

    print("Cross-validating Decision Tree...")

    dt_cv_scores = cross_validate_model(dt_model, X, y)

    print("Decision Tree CV accuracy: mean=%.4f std=%.4f" % (dt_cv_scores.mean(), dt_cv_scores.std()))


    print("Cross-validating Random Forest...")

    rf_cv_scores = cross_validate_model(rf_model, X, y)

    print("Random Forest CV accuracy: mean=%.4f std=%.4f" % (rf_cv_scores.mean(), rf_cv_scores.std()))


    # Feature importances (from RF)

    feature_names = X.columns.tolist()

    plot_feature_importances(rf_model.feature_importances_, feature_names,
os.path.join(OUTPUT_DIR, "feature_importances.png"))
```

```
# Decision tree plot (pruned/best)
try:
    plot_and_save_tree(dt_model, feature_names, os.path.join(OUTPUT_DIR, "decision_tree.png"))
except Exception as e:
    print("Could not plot tree due to:", e)

# Accuracy comparison plot
plt.figure(figsize=(6,4))
models = ['Decision Tree', 'Random Forest']
accs = [dt_acc, rf_acc]
sns.barplot(x=models, y=accs)
plt.ylim(0,1)
plt.ylabel("Validation Accuracy")
plt.title("Model Accuracy Comparison")
plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, "model_accuracy_comparison.png"))
plt.close()

# Confusion matrix for Random Forest on test set
y_pred = rf_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Random Forest - Confusion Matrix (Test)")
plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, "rf_confusion_matrix.png"))
plt.close()
```

```
    # Save models' basic info to a report
    report_path = os.path.join(OUTPUT_DIR, "report.txt")
    with open(report_path, "w") as f:
        f.write("Decision Tree best params: %s\n" % str(dt_params))
        f.write("Decision Tree validation accuracy: %.4f\n" % dt_acc)
        f.write("Decision Tree CV mean: %.4f std: %.4f\n" % (dt_cv_scores.mean(), dt_cv_scores.std()))
        f.write("\nRandom Forest best params: %s\n" % str(rf_params))
        f.write("Random Forest validation accuracy: %.4f\n" % rf_acc)
        f.write("Random Forest CV mean: %.4f std: %.4f\n" % (rf_cv_scores.mean(), rf_cv_scores.std()))
        f.write("\nTop 20 feature importances (Random Forest):\n")
        importances = sorted(zip(feature_names, rf_model.feature_importances_), key=lambda x: x[1],
reverse=True)[:20]
        for feat, imp in importances:
            f.write("%s: %.6f\n" % (feat, imp))

    print("All outputs saved to the 'outputs' directory.")

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