Overview

This project implements a **Decision Tree Classifier** to predict weather types (Sunny, Rainy, Snowy, Cloudy) using the `weather_classification_data.csv` dataset. It includes data cleaning, model training, evaluation, visualizations, and an interactive user interface for activity recommendations based on predicted weather types.

Objective

- **Primary Goal**: Predict the weather type based on features such as Temperature, Humidity, Wind Speed, Precipitation (%), UV Index, Visibility (km), Cloud Cover, Season, and Location.
- **Secondary Goal**: Recommend activities based on the predicted weather type (e.g., "Go to the beach" for Sunny, "Stay indoors" for Rainy).
- **Dataset**: `weather_classification_data.csv` with 10 features and a target column (`Weather Type`).

Dependencies

The script uses the following Python libraries:

- `numpy` : For numerical operations and array manipulations.
- `pandas` : For data loading, cleaning, and preprocessing.
- `sklearn.model_selection.train_test_split` : To split the dataset into training and test sets.
- `matplotlib.pyplot` : For generating and saving visualizations.
- `os` : To create directories for saving output diagrams.

Data Processing

Steps

1. **Load Dataset**:

- The dataset `weather_classification_data.csv` is loaded using `pandas.read_csv`.
- Columns are renamed for clarity: `Temperature`, `Humidity`, `Wind Speed`, `Precipitation (%)`, `Cloud Cover`, `Atmospheric Pressure`, `UV Index`, `Season`, `Visibility (km)`, `Location`, `Weather Type`.

2. **Clean Data**:
   - The `Atmospheric Pressure` column is dropped as it is not used in the model.
   - A subset of the first 100 rows is selected for simplicity and faster computation.
3. **Feature and Target Separation**:
   - Features (`weather_classification_data_x`): All columns except `Weather Type`.
   - Target (`weather_classification_data_y`): `Weather Type` column.
4. **Encode Categorical Data**:
   - Categorical features (`Cloud Cover`, `Location`, `Season`) are converted to numerical dummy variables using `pd.get_dummies`.
   - The target variable (`Weather Type`) is encoded into numerical values (e.g., Sunny=0, Rainy=1, Snowy=2, Cloudy=3) using `astype('category').cat.codes`.
5. **Split Dataset**:
   - The dataset is split into training (80%) and test (20%) sets using `train_test_split` with a random state of 41 for reproducibility.

Decision Tree Algorithm

Core Guidelines

The Decision Tree Classifier is implemented from scratch with the following components:

1. Node Class

- Represents a node in the decision tree.
- Attributes:
  - `feature_index`: The index of the feature used for splitting.
  - `threshold`: The threshold value for the split.
  - `left` and `right`: Child nodes (left for values ≤ threshold, right for values > threshold).
  - `info_gain`: Information gain from the split.
  - `value`: Leaf node value (class label) if the node is a leaf.

2. DecisionTreeClassifier Class

- **Initialization**:
  - `min_samples_split=3` : Minimum number of samples required to split a node.
  - `max_depth=3` : Maximum depth of the tree to prevent overfitting.
  - `root` : The root node of the tree (initially None).
- **Gini Index** (`gini_index`):
  - Calculates the Gini impurity for a set of labels.
  - Formula: (\text{Gini} = 1 - \sum_{i=1}^{n} p_i^2), where (p_i) is the probability of class (i).
  - Used to measure the impurity of a node.
- **Split Function** (`split`):
  - Splits the dataset into left and right subsets based on a feature and threshold.
  - Left subset: Samples where the feature value is ≤ threshold.
  - Right subset: Samples where the feature value is > threshold.
- **Information Gain** (`information_gain`):
  - Calculates the information gain of a split using Gini impurity.
  - Formula: (\text{Info Gain} = \text{Gini(parent)} - (\text{weight}*{\text{left}} \cdot \text{Gini(left)} + \text{weight}*{\text{right}} \cdot \text{Gini(right)})).
  - `weight_left` and `weight_right` are the proportions of samples in the left and right subsets.
- **Best Split** (`get_best_split`):
  - Iterates over all features and their unique values as thresholds.
  - Computes the information gain for each potential split.
  - Returns the split with the highest information gain, including the feature index, threshold, and resulting datasets.
- **Tree Building** (`build_tree`):
  - Recursively builds the decision tree.
  - Stops if:
    - The number of samples is less than `min_samples_split` .
    - The current depth exceeds `max_depth` .
    - No positive information gain is achieved.
  - If stopping criteria are met, creates a leaf node with the majority class (`np.bincount(y).argmax()`).
  - Otherwise, splits the dataset and recursively builds left and right subtrees.
- **Fit** (`fit`):
  - Combines features and labels into a single dataset.
  - Initiates tree building starting from the root.
- **Predict** (`predict`):

- Makes predictions for a set of samples by traversing the tree for each sample.
- Calls `make_prediction` for each sample.
- **Make Prediction** (`make_prediction`):
  - Traverses the tree for a single sample.
  - If at a leaf node, returns the node's value (class label).
  - Otherwise, compares the feature value at the current node to the threshold and recursively traverses the left or right subtree.

3. **Training and Evaluation**

- The classifier is trained on the training set using `classifier.fit(X_train, y_train)`.
- Predictions are made on the test set using `classifier.predict(X_test)`.
- Accuracy is calculated as the mean of correct predictions: `np.mean(y_pred == y_test)`.

Output Diagrams

The script generates three visualizations to analyze the data and model performance, saved in the `output/figures/` directory.

1. Training Set Visualization

- **File**: `output/figures/training_temperature_humidity_scatter.png`
- **Type**: Scatter Plot
- **Description**:
  - Plots Temperature (x-axis) vs. Humidity (y-axis) for the training set.
  - Points are colored by encoded Weather Type (e.g., 0=Sunny, 1=Rainy, 2=Snowy, 3=Cloudy) using the `viridis` colormap.
  - Includes a colorbar labeled "Weather Type (Encoded)".
  - Helps visualize how the training data is distributed across two key features, which the decision tree uses for splitting.

2. Test Set Visualization

- **File**: `output/figures/test_actual_vs_predicted.png`
- **Type**: Line Graph
- **Description**:
  - Compares actual vs. predicted Weather Types for the test set.
  - X-axis: Test sample index.

- Y-axis: Encoded Weather Type.
- Two lines:
  - Actual Weather Type (blue, with circle markers).
  - Predicted Weather Type (orange, with 'x' markers).
- Includes a legend to distinguish between actual and predicted values.
- Illustrates the model's performance on unseen data by showing where predictions match or diverge from actual labels.

3. Feature Distribution Visualization

- **File**: `output/figures/temperature_distribution.png`
- **Type**: Line Graph
- **Description**:
  - Plots the Temperature trend across the dataset (first 100 samples).
  - X-axis: Sample index.
  - Y-axis: Temperature (°C).
  - Line is colored blue and labeled "Temperature".
  - Includes a legend.
  - Relevant for understanding patterns in Temperature, a key feature the decision tree may use for splits.

Activity Recommendation

Mapping

- **Weather Type Mapping**:
  - Encoded values are mapped to weather types: {0: "Sunny", 1: "Rainy", 2: "Snowy", 3: "Cloudy"}.
- **Activity Mapping**:
  - Each weather type is associated with a list of activities:
    - Sunny: "Go to the beach", "Have a picnic", "Hike".
    - Rainy: "Stay indoors", "Watch a movie", "Read a book".
    - Snowy: "Build a snowman", "Go skiing", "Stay warm indoors".
    - Cloudy: "Visit a museum", "Go for a walk", "Do indoor crafts".

User Interaction

- **Input**:
  - Users are prompted to input weather conditions: Temperature (°C), Humidity (%), Wind Speed (km/h), Precipitation (%), UV Index, Visibility (km), Cloud Cover (clear/partly cloudy/overcast), and Location (e.g., coastal, inland).
- **Processing**:
  - Input is converted into a feature vector matching the training data format using `pd.get_dummies` and aligned with training features.
- **Prediction**:
  - The model predicts the weather type based on the user input.
- **Output**:
  - The predicted weather type and suggested activities are printed.
  - Users can choose to try again or exit the loop.

Execution Flow

1. **Data Processing**: Load, clean, and preprocess the dataset.
2. **Model Training**: Train the Decision Tree Classifier on the training set.
3. **Evaluation**: Predict on the test set and compute accuracy.
4. **Visualizations**: Generate and save the three plots described above.
5. **User Interaction**: Prompt for user input, predict the weather type, and recommend activities in a loop until the user exits.

Notes

- **Dataset**: Assumes `weather_classification_data.csv` is in the same directory as the script.
- **Directory Creation**: Uses `os.makedirs('output/figures', exist_ok=True)` to ensure the output directory exists for saving plots.
- **Accuracy**: The model achieves an accuracy of around 0.65 (65%) on the test set, indicating moderate performance that could be improved with hyperparameter tuning or more data.
- **Extensibility**: The script can be extended with cross-validation, additional visualizations (e.g., confusion matrix), or feature importance analysis.