Week 6 Assignment – AutoMPG Data Set

Overview

In this assignment, you will download a publicly available data set and develop some classes that will be useful in working with the data. The data set contains information about fuel efficiency for some different automobiles.

Please follow the steps outlined below.

Preparation

Review the links below that describe the AutoMPG data set that is part of the UCI Machine Learning Repository. This data set consists of 398 records each having nine attributes. Please download the data and save it in a file called **auto-mpg.data.txt**.

- https://archive.ics.uci.edu/ml/datasets/Auto+MPG
- https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data
- https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.names

Please create a new Python module in a file **autompg.py**, taking care to set it up to be useful both as a module and as a program. Also create a unittestt testcase skeleton in **test_autompg.py**.

Step 1: AutoMPG class

In **autompg.py** implement a class that represents these attributes that are available for each record in the data set:

- **make** – String. Automobile manufacturer. First token in the "car name" field of the data set.
- **model** – String. Automobile model. All the other tokens in the "car name" field of the data set except the first.
- **year** – Integer. Automobile year of manufacturer. This is the four-digit year that corresponds to the "model year" field of the data set.
- **mpg** – Floag. Miles per gallon. This is a floating point value that corresponds to the "mpg" field of the data set.

This class should implement the following methods:

- **__init__** – Constructor that takes four parameters after **self** and initializes the attributes described above.
- **__repr__** and **__str__** – Returns the string representation of the object. One of these can call the other one.

- **__eq__** – Implements equality comparison between two AutoMPG objects. Should use all four attributes and should only work between AutoMPG objects.
- **__lt__** – Implements less-than comparison between two AutoMPG objects. Should use all four attributes in the order described above (e.g., if the "make" is the same, then the model should be used).
- **__hash__** – Implement an appropriate hash function for these objects.

In **test_autompg.py** implement a test class and test cases that test all of the functionality above.

Step 2: AutoMPGData class

In **autompg.py** define the **AutoMPGData** class. This class represents the entire AutoMPG data set. It has a single attribute **data** that is a list of **AutoMPG** objects. One issue with the Auto MPG data set is that in the data file, most of the fields are separated by spaces, but the separator before the "car name" field is a TAB character. This makes it difficult to parse with Python's **csv** module, so there is a method described below (**_clean_data**) that takes care of this and saves the "cleaned" data file in **auto-mpg.clean.txt**.

This class should implement these methods:

- **__init__** – Constructor that takes no arguments. It should call the **_load_data** method described below.
- **__iter__** – Method to make the class iterable. It should return an iterator over the **data** list.
- **_load_data** – Method that will load the cleaned data file (**auto-mpg.clean.txt**) and instantiate AutoMPG objects and add them to the **data** attribute. See below for more details about a useful technique for parsing the data. This method should call **_clean_data** if the clean data file does not exist. (Hint: Use **os.path.exists** to check this.)
- **_clean_data** – Method that will read the original data file line by line (**auto-mpg.data.txt**) and use the **expandtabs** method available on **str** objects to convert the TAB character to spaces. Each line should be written to the "cleaned" file **auto-mpg.clean.txt**.

Parsing Notes

The main goal of the **_load_data** method is to read in the data file and instantiate objects of type **AutoMPG** using a few of the fields that are available. Here are some notes about how to accomplish this:

- Use the **csv** module to parse each line with space delimiters and ignore multiple sequential delimiters.
- This will give you a list of nine elements that correspond to the columns in the data file.
- Use **collections.namedtuple** to define a class called **Record** that has nine attributes that correspond to the nine data fields in the same order as in the file.
- Using tuple packing/unpacking, assign the list returned by the **csv** module for a row to create a **Record** object.

- Use the attributes of the **Record** object to pass the appropriate values to the constructor for **AutoMPG**.
- You will need to use **str.split** and **str.join** to pick out the make and model values from the data since they are contained within "" in the data file, so will be returned by the **csv** module as a single string value.

Implement a test class and test cases in **test_autompg.py** to ensure the functionality above.

Step 3: main

In **autompg.py** implement a main function that instantiates an **AutoMPGData** object and then uses the iterator protocol (e.g., **for a in AutoMPGData():**) to loop across the object, printing each AutoMPG object. For example, the output should look like this:

```
> python3 autompg.py
AutoMPG('chevrolet','chevelle malibu',1970,18.0)
AutoMPG('buick','skylark 320',1970,15.0)
AutoMPG('plymouth','satellite',1970,18.0)
AutoMPG('amc','rebel sst',1970,16.0)
AutoMPG('ford','torino',1970,17.0)
…
```

Upload

Please combine **autompg.py** and **test_autompg.py** into a single ZIP file.