Python Demonstration: Reading and Writing CSV files

What are CSV files?

A CSV is a comma-separated values file, which allows data to be saved in a tabular format. CSVs look like a garden-variety spreadsheet but with a .csv extension.

We'll be learning the basics of iterating through a CSV file to create dictionaries and collect summary statistics.
First, let's import the CSV module, which will assist us in reading in our CSV file.
Using some iPython magic, let's set the floating point precision for printing to 2.
Now let's read in our mpg.csv using csv.DictReader and convert it to a list of dictionaries.

Reading and Writing CSV files

Let's import our datafile mpg.csv, which contains fuel economy data for 234 cars.

- mpg : miles per gallon
- class : car classification
- cty : city mpg
- cyl : # of cylinders
- displ : engine displacement in liters
- drv : f = front-wheel drive, r = rear wheel drive, 4 = 4wd
- fl : fuel (e = ethanol E85, d = diesel, r = regular, p = premium, c = CNG)
- hwy : highway mpg
- manufacturer : automobile manufacturer
- model : model of car
- trans : type of transmission
- year : model year

In [1]:
```
import csv

%precision 2

with open('mpg.csv') as csvfile:
    mpg = list(csv.DictReader(csvfile))

mpg[:3] # The first three dictionaries in our list.
```

Let's look at the first three elements of our list.
We can see that the dictionaries in the list have the column names of the CSV as keys, and data for each specific car are the values.

```
Out[1]: [OrderedDict([('', '1'),
                      ('manufacturer', 'audi'),
                      ('model', 'a4'),
                      ('displ', '1.8'),
                      ('year', '1999'),
                      ('cyl', '4'),
                      ('trans', 'auto(15)'),
                      ('drv', 'f'),
                      ('cty', '18'),
                      ('hwy', '29'),
                      ('fl', 'p'),
                      ('class', 'compact')]),
              OrderedDict([('', '2'),
                      ('manufacturer', 'audi'),
                      ('model', 'a4'),
                      ('displ', '1.8'),
                      ('year', '1999'),
                      ('cyl', '4'),
                      ('trans', 'manual(m5)'),
                      ('drv', 'f'),
                      ('cty', '21'),
                      ('hwy', '29'),
                      ('fl', 'p'),
                      ('class', 'compact')]),
              OrderedDict([('', '3'),
                      ('manufacturer', 'audi'),
                      ('model', 'a4'),
                      ('displ', '2'),
                      ('year', '2008'),
                      ('cyl', '4'),
                      ('trans', 'manual(m6)'),
                      ('drv', 'f'),
                      ('cty', '20'),
                      ('hwy', '31'),
                      ('fl', 'p'),
                      ('class', 'compact')])]
```

The length of our list is 234, meaning we have a dictionary for each of the 234 cars in the CSV file.

```
In [ ]: len(mpg)
```

We can look at what the column names of the CSV are by using the key method.

keys gives us the column names of our csv

```
In [2]: mpg[0].keys()
Out[2]: odict_keys(['', 'manufacturer', 'model', 'displ', 'year', 'cyl', 'trans', 'drv', 'cty', 'hwy', 'fl', 'class'])
```

Suppose we want to find the average city MPG across all cars in our CSV file.

We sum the city MPG entry across all the dictionaries in our list and divide by the length of the list.

Because the type for all the values in our dictionary are strings, we need to convert to float to perform mathematical operations.

This is how to find the average cty fuel economy across all cars. All values in the dictionaries are strings, so we need to convert to float.

```
In [4]: sum(float(d['cty']) for d in mpg) / len(mpg)
Out[4]: 3945.00
```

Similarly, we can find the average highway MPG across all the cars in our CSV file. And it makes sense that the average highway fuel economy is higher than in the city.

Similarly this is how to find the average hwy fuel economy across all cars.

```
In [ ]: sum(float(d['hwy']) for d in mpg) / len(mpg)
```

Now, let's look at a more complex example.
Say we want to know what the average city MPG is grouped by the number of cylinders a car has.

Creating a set of the values in the cylinder entry of the dictionaries will give us the unique levels for a number of cylinders.
We see that we have cars in our dataset with 4, 5, 6, and 8 cylinders.

Use set to return the unique values for the number of cylinders the cars in our dataset have.

```
In [5]: cylinders = set(d['cyl'] for d in mpg)
        cylinders
Out[5]: {'4', '5', '6', '8'}
```

First, we'll create an empty list where we'll store our calculations.
Next, let's iterate over all the cylinder levels. And then we'll iterate over all the dictionaries.

Here's a more complex example where we are grouping the cars by number of cylinder, and finding the average cty mpg for each group.

```
In [ ]: CtyMpgByCyl = []

for c in cylinders: # iterate over all the cylinder levels
    summpg = 0
    cyltypecount = 0
    for d in mpg: # iterate over all dictionaries
        if d['cyl'] == c: # if the cylinder level type matches,
            summpg += float(d['cty']) # add the cty mpg
            cyltypecount += 1 # increment the count
    CtyMpgByCyl.append((c, summpg / cyltypecount)) # append the tuple ('cylinder', 'avg mpg')

CtyMpgByCyl.sort(key=lambda x: x[0])
CtyMpgByCyl
```

If the cylinder level for the dictionary we're on matches the cylinder level we're calculating the average for, we add the mpg to our summpg variable and increment the count.
After going through all the dictionaries, we perform the average MPG calculation and append it to our list.

To make things clearer, I'm going to sort the list from lowest number of cylinders to highest.
And we can see that the city fuel economy appears to be decreasing as the number of cylinders increases.

Let's look at one more similar example.

Suppose we're interested in finding the average highway MPG for the different vehicle classes.
Looking at the different vehicle classes, we have 2seater, compact, midsize, minivan, pickup, subcompact, and SUV.
Similar to the last example, we iterate over all the vehicle classes, then iterate over all the dictionaries.
If the vehicle class for the dictionary matches the vehicle class we're computing the average highway MPG for, we add the value to our total, and increment the count.
Then we perform the average calculation and append it to our list.
This time, let's sort our list from lowest MPG to highest.
Looks like the pickup had the worst fuel economy and the compact had the best.

Use set to return the unique values for the class types in our dataset.

In [ ]:
```
vehicleclass = set(d['class'] for d in mpg) # what are the class types
vehicleclass
```

And here's an example of how to find the average hwy mpg for each class of vehicle in our dataset.

In [ ]:
```
HwyMpgByClass = []

for t in vehicleclass: # iterate over all the vehicle classes
    summpg = 0
    vclasscount = 0
    for d in mpg: # iterate over all dictionaries
        if d['class'] == t: # if the cylinder amount type matches,
            summpg += float(d['hwy']) # add the hwy mpg
            vclasscount += 1 # increment the count
    HwyMpgByClass.append((t, summpg / vclasscount)) # append the tuple ('class', 'avg mpg')

HwyMpgByClass.sort(key=lambda x: x[1])
HwyMpgByClass
```

So that was a look into how to summarize data through iteration.