COMP 364 FALL 2017: COMPUTER TOOLS FOR LIFE SCIENCES ASSIGNMENT #5

DUE: THURSDAY DECEMBER 7, 2017 11:59:59 PM

This assignment has two major parts.

In part one, you will write a Python module to handle and extract data from Comma Separated Value (CSV) files. A CSV file, as described in class, is a widely used format for storing data in a tabular form. Typically, data records are represented as rows, and their attributes/features are separated by a comma (forming a set of columns). See the tester file below for an example.

In part two, you will use your CSV module from part one to generate visualizations from a dataset of your choosing and then interpret the results.

NOTE: For this assignment, a Jupyter notebook submission is not valid. To better simulate a more "standard" Python development environment, we strongly recommend you use a simple text editor such as Sublime and the Python command line environment (e.g., `python mymodule.py`). Please see "What to Submit" at the end of the assignment outline for information about MyCourses submissions.

1. Part I: Parsing (60 points)

Write a module called `mycsv.py` for parsing CSV files. For this module creation task, you may not use any third party Python modules or packages, or the `csv` module from Python's standard library.

The module will contain two class definitions:

(1) `MyCSV`
(2) `MyData`

1.1. MyCSV (**35 points**)

The `MyCSV` class will implement the following two methods:

(1) An initializer method that takes as input:

- `self` as a positional argument
- `delimiter = ","`, and `header = True` as keyword arguments

The `delimiter` value is a string that is used to separate column entries. In CSV files, `delimiter` is typically set to a comma (','), but we will support any string or whitespace characters. The `header` argument indicates whether the first row of the file should be used as column labels. The initializer uses these arguments to set the class attributes `delimiter` and `header`.

(2) An instance method called `parse` with the following positional arguments:

- `self` -a `MyCSV` object
- `path` - a string representing the path to the file to be parsed

The `parse` function returns a new `MyData` object containing data read from the provided file path. Before returning, it will set a few attributes of the newly created `MyData` object:

- `columns`: a list of column names in the data file. If `header` is `True`, the column names are the values in the first row of the file. Otherwise, column names are strings of the form `col_0`, `col_1`, ..., `col_N-1` for `N` columns.
- An attribute for each data column. The name of an attribute is the same as the column it represents. An attributes value is a list of values belonging to its column, i.e. one entry in the list per row belonging to the column. (Hint: the built-in function `setattr` will be useful)

1.2. MyData (**25 points**)

The `MyData` class will only implement **one** instance method:

- `iterrows`

`iterrows` accepts a single positional argument, `self`, that is an instance of the `MyData` class. The `iterrows` method creates an iterator over the rows of the `MyData` object. At every iteration of the iterator, `iterrows` yields a row from the table as a `namedtuple` object whose names correspond to the column names. (Hint: the built-in function `getattr` will be useful)

For example, consider the following sample file, `file.csv`:

```
name,gpa,id
bob,4.0,2220330
alice,3.8,2232219
adam,2.5,2449291
eve,3.1,2411293
```

The expected behaviour of your `mycsv` module would be the following:

```
>>> from mycsv import  *
>>> csv_path = "/path/to/csv/file.csv"
>>> parser = MyCSV(delimiter=",", header=True)
>>> c = parser.parse(csv_path)
>>> c.name
['bob', 'alice', 'adam', 'eve']
>>> c.columns
['name', 'gpa', 'id']
>>> for row in c.iterrows():
...     print(row.gpa, row.id)
'4.0', '2220330'
'3.8', '2232219'
'2.5', '2449291'
'3.1', '2411293'
```

1.3. **Please Note.**

Some considerations when developing your module:

- You may assume that input files have the correct number of entries in each column
- You may store all the data as strings in memory
- We are providing a tester module `testcsv.py` and a sample csv file to parse `test.csv`. Using these files is strongly encouraged for debugging. **Although a successful implementation on the sample file does not guarantee full marks**. These provisions are simply there to help debugging. Full marks are given to submissions that work as specified by the instructions.
- Please use informative variable names, docstrings, and comments. Avoid copy-pasting code from the Internet, from classmates, and from other parts in your own code files.

2. **Part II: Visualization (40 points)**

Visit Kaggle, an online database of interesting datasets: `https://www.kaggle.com/datasets`.

Choose one of the available datasets in CSV format that interests you and download it. Try to use a different dataset than your friends.

Using the module you wrote in part one to parse data stored within a CSV file, create three **well labeled** `Matplotlib` plots that demonstrate interesting trends in the data. These plots can be in a style of your choosing (scatter, line, histogram, bar, heatmap, etc.). Save the code you write to generate the plots in a `.py` file, called "plot.py".

Next, save your figures and include them in a PDF file. Provide a description of the dataset, your plots, and how they illustrate the trend (one page long at most). Make sure to include a link to the dataset on Kaggle.

Save the PDF and include it in your submission. Your PDF file should have the following name convention: "COMP364_F17_Lastname_Firstname.pdf"

3. **What to submit**

Please include the following in a `.zip` or `.tar.gz` compressed file.

(1) `mycsv.py`
(2) the CSV from Kaggle that you chose to parse
(3) your plotting code in the `plot.py` file.
(4) your report as a `.pdf` file. (Do not submit `.doc`, `.docx`, or any other formats)