ITNPD2 Assignment 2016: A JSON Controlled Data Summariser in Python

Introduction

Data comes in many formats and in varying degrees of quality. The first steps when considering a new file of data for processing are to establish facts about its format, the special characters it uses such as separators and string enclosures, whether or not it has a header row or footer, and so on. Once these things have been established, you might want to summarise the data or identify problems such as missing values.

This practical involves writing a Python program that uses JSON to describe the contents of a file and as a form of command language for manipulating data. JSON will be used both to allow a user to control the program (there will be no GUI) and as the format in which results are generated.

Let"s call the program PyProc. It should take one parameter when run, the name of the JSON formatted file that contains the details of what it should do. For example:

```
PyProc param.json
```

where the file param.json contains:

```
{"inputFilePath":"C:/data/data.csv",
 "metaFile":"C:/data/metadata.json",
 "format":"tabular",
 "hasheader":true,
 "separator":"," }
```

should make the program run and open the file data.csv and write the meta data it calculates to metadata.json. This file also specifies that data.csv has a header row and uses a comma separator. Those parameters are not needed for JSON format and are optional for tabular format where, if they are missing, the program must guess the most likely value.

Generating Metadata

Your program should read a file and generate metadata about that file. The file format details are one example of metadata. Other required metadata fields are described below. The output from your program should be a JSON object containing the metadata extracted from a given file. The JSON object should contain an array of JSON objects, one for every field (or variable) defined in the file, plus some other fields describing the file as a whole.

The JSON output about a file should describe the following about it:

- Filename
- Format (JSON or tabular)
    - If tabular, specify character for field separator
    - If tabular, specify whether first row contains field names or not
- Number of entries (rows if tabular, objects if JSON)
- Number of fields (columns if tabular, different fields if JSON)
- For each field:
    - Name
    - Type (numeric or string)
        - For numeric data: max, min
        - For string data: Number of different values the field takes

Here is an example:

```
{
"filename":"C:/data/data.csv",
"format":{"type":"tabular","sep":","},

"headerrow":1,
"numentries":100,
"numfields":2,
"fields:[{"name":"ID","type":"numeric","min":0,"max":100},
          {"name":"Name","type":"string","uniquevals":30}
        ]
}
```

If the input file is in CSV format, use the first row as column headings or, if there are no column headings, use default values "variable1", "variable2", etc. If the file is in JSON format, you must build the list of keys from the objects in the data. Only list the top level keys (i.e. ignore keys in embedded objects).

Format Detection

Your program should accept data in JSON or tabular format. It should be able to infer the format of the file from the file extension (.JSON, .txt, .csv), but also allow the input JSON parameter file to specify things about the format where known. For tabular files (.csv, .txt, etc.), the program should attempt to guess the following:

- Does the first row contain field headings? If all the entries on the top row are strings, but subsequent rows contain numbers, then your program can infer that the top row is a header (but state this as an assumption).
- What is the field separator character? You may assume that .csv files use a comma and .txt files use a tab (\t).

For JSON format files, your program should accept a file where every row contains a single JSON object. That will save you the bother of parsing multi-line JSON objects.

The results of the guesses above should be written to the output file in JSON format, as in the example above. To override a guess, the user should be able to edit the resulting JSON and re-run the program with the new format in the JSON parameter file.

Type Detection

Your program should detect the most likely type of each field, classifying each as either `numeric` or `string`. Use a regular expression to define the possible formats for a number (remember the possibility of scientific notation: 2.304e+3, for example and negative numbers such as -4.5). It is enough to identify the type of each variable based on the first record in a tabular file. For JSON, you need to scan the whole file to be sure of finding every variable, but you can still make your decision about type based on the first occurrence of each variable name. When the data is in JSON format, you can work on the top level fields only – do not process embedded objects.

The default (if parameters do not specify otherwise) should be to write all fields of all records in the same format as the input file, including those with missing values. Consequently, all parameters are optional except the output file name. If that is missing, nothing is written.

Counting Values and Max and Min

Your JSON output file should list the maximum and minimum value for numeric fields and the number of unique values string fields can take. Check the type of each variable and decide which calculation to make. To count the number of unique values a field takes, store its possible values in a Python `set` and then find its length.

Hints and Help

- Use the json library to read and write JSON format. You"ll find `json.load()` and `json.dump()` useful.
- Windows filename has \ backslashes in them. You may choose to deal with them or, more easily, insist that the forward slash / is used in the parameter file. This will still work fine.
- Remember that readline() keeps the \n at the end of lines. Remove it using `.rstrip()`.
- For tabular data, you can keep the field names, types etc. in an array indexed by the column number in the file, but there is no ordering in JSON, so you need to index by the field names.
- Before you start to code, draw out a design of the steps the program will take. For example, it should identify the file format first, then decide whether there is a header row or not, then count the columns, etc.

Target Output

The two files you have been given to process, `carsdata.csv` and `carsdata.json` represent the same data in two different formats. The result of processing the csv should look like this:

```
{"format": "tabular",
 "fields":
   [{"Type": "string", "Name": "Manufacturer", "uniquevals": 3},
    {"Type": "string", "Name": "Colour", "uniquevals": 5},
    {"Type": "string", "Name": "Size", "uniquevals": 10},
    {"max": "44201.0", "Type": "numeric", "Name": "Mileage", "min":
     "33872.0"},
   {"max": "2732.0", "Type": "numeric", "Name": "Price", "min":
    "1745.0"}
   ],
"numrows": 100, "header": 1, "separator": ",", "numfields": 5,
"infile": "H:/Big Data/Lectures/Python/Assignment/cars.csv"}
```

The results of processing the json file should be similar, except that the format should say `json` and there will be no values for `header` or `separator`. Don"t worry about the layout – your output can be all on one line. You could list the actual unique values if you wish.

What to Submit

Write your program in Python. Get the data and parameter files from the course web site and run your program on them. You should submit your program code (a single file called `PyProc.py`) and the results of running it on the data and parameter files from the assignment page of the course web site. Do this by zipping them up and emailing them to jrw@cs.stir.ac.uk. Please also include your name and student number in the comments at the start of `PyProc.py`). Make sure your code is well commented throughout and also comment each function with docstring. 10% of the marks are for good comments. Please also submit a printed copy of your code and output on the two files. Submit this via the pigeon holes.

How marks will be awarded

You will gain a pass mark if your program is able to read a JSON parameter file that specifies everything about the data file (format, header row, etc.) and use it to load and process a data file, writing the results in the same format to an output file. Above that, extra marks are awarded for discovering the format and type of variables automatically, converting between one input format and another output format, and calculating the metadata statistics described above.

Late submission & Plagiarism

It is possible for the co-ordinator to vary the dates for a student. If you must submit your work late (perhaps for medical reasons) then this can be arranged, but it must be discussed with the co-ordinator as soon as possible after the problem becomes known. Assessed coursework submitted late will be accepted up to seven calendar days after the submission date (or expiry of any agreed extension) but the grade will be lowered by three marks per day or part thereof. After seven days the piece of work will be deemed a non-submission, and will result in a fail grade for the module as a whole.

Work which is submitted for assessment must be your own work. All students should note that the University has a formal policy on plagiarism which can be found at http://www.quality.stir.ac.uk/ac-policy/assessment.php. The assignment counts for 50% of the total course mark. The deadline for submission is Friday 11th of November at 4pm.