Graded Assignment 1 - Data Ingestion June 2022

This assignment will be marked out 100, and is split into two tasks, accounting for 65% and 35% of the total marks respectively.

Data

You are provided with four data files that contain daily values of outdoor temperature, indoor temperature + humidity, barometric pressure, and rainfall covering a year.

The files are in the CSV format and can be found in the weather-data.zip linked on the Engage page of this Graded Assignment.

Submission format

This assignment is **not** autograded. However, we ask you to follow the submission template *ApAi_GA1_Template.ipynb* when preparing your work to aid manual marking and minimise the chance of violating assignment specifications resulting in loss of points. The template can also be found linked on the Engage page of this Graded Assignment.

Task 1 (65%): CSV-parser

Your first task is to write a Python program to parse data files written in the CSV format. Note that such a parser ingests data from a given CSV file recognising and properly handling data types and stores the data in an appropriate Python data structure.

For this task you **must not** use any existing CSV reading or parsing code. This includes (but is not limited to) the csv-parsers from NumPy, Pandas, ast and the inbuilt csv module of Python itself (please also refer to **Library use restrictions** on the Engage page of this Graded Assignment). Instead, you are asked to code the parsing capability from scratch using Python's file IO operations.

The objective for Task 1 is to demonstrate you understand the low-level functionality involved in reading a CSV file, including an understanding of the format itself. These insights can be extremely useful in trying to understand the origins of data ingestion errors in this context.

You can read more about IO handling in Python here:

https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

This file handling functionality in Python is easy to use and hence the difficulty of the task is not Python-related. The challenge is in devising an approach to read raw text (be it line-by-line, character-by-character, or otherwise) and to convert it into an internally useable form by correctly handling different data types represented in the ingested text. In addition, as discussed in the course material, raw text data can be cumbersome to process due to the possible input irregularities and CSV format variations. Hence the **generalisation ability** of your parser will be also tested using a set of additional hidden CSV files. Furthermore, your

implementation will be tested for robustness in its **error handling ability** when faced with files violating the CSV format in some way. The following sections detail the testing process.

CSV-parser testing: provided and hidden files

Step 1:

First of all, your code should correctly parse the CSV files provided in weather-data.zip.

Step 2:

Next, you need to consider what modifications are necessary to make your CSV-parser (more) universally applicable to other files **that follow the CSV format**, considering both **input data** and **format variations** within the scope of the CSV definition:

https://datatracker.ietf.org/doc/html/rfc4180.

Please note that the CSV-parser is not allowed any additional parameters except the CSV file name (you can assume the file will be in the same folder as your code). However, you can assume the character set encoding of the hidden files to be the same as that of the provided files (for example, please note that LF, not CRLF, is used to denote line breaks in these files).

The **generalisation ability** of your CSV-parser will be tested on two hidden files:

- **Hidden file 1 - strings:** this file is designed to test your parser's handling of strings and related input irregularities;
- **Hidden file 2 - dates:** this file is designed to test handling of date formats.

Please note that, in addition to the specified input inconsistencies, the hidden files can have a different structure or variation of the CSV-format compared to the provided files.

Step 3:

Finally, you need to make sure your CSV-parser can handle files with some CSV format errors gracefully i.e. without throwing unhandled exceptions and, where possible, returning a sensible partial result of any input portion with valid data. Informative messages with detected format violations as part of your CSV-parser output are encouraged!

This **error handling ability** will be tested on the third hidden file:

- **Hidden file 3 - malformed:** the file contains some valid data but also fields violating the CSV format. Once again, the CSV format specification can be found here:

  https://datatracker.ietf.org/doc/html/rfc4180.

  Please concentrate on exception detection and handling of format violations that seem the most likely to occur.

If the input file contains invalid data as in the malformed file, your CSV-parser should not generate any unhandled exceptions. You should aim for recovery of any valid data from such a file to provide a sensible partial result, but the parser is not expected to correctly parse any invalid data (i.e. input violating format specification).

To summarise, your CSV-parser implementation will be assessed according to the following criteria:

Assessment Criteria for Task 1	**Marks Awarded**
Is the CSV parser implementation wrapped in a function that takes only a filename as input and returns data? And does it not make use of any existing CSV reading and parsing library for file handling? *Note: passing this section is required for subsequent marks*	10
Does the program correctly parse the **provided CSV files** into a suitable Python data structure recognising and properly handling data types?	10
Does the program correctly parse the **first hidden CSV** file into a suitable Python data structure recognising and properly handling data types? This file will in particular test how well your code can handle a variety of different **strings**.	15
Does the program correctly parse the **second hidden CSV** file into a suitable Python data structure recognising and properly handling data types? This file will in particular test how well your code can handle manually inputted **dates**.	15
Does the program give a sensible partial result for the hidden **malformed CSV file**, which contains some valid data, without throwing unhandled exceptions? Please note that correct **informative messages** on detected csv format violations in the file generated by the parser will also be credited under this criterion.	15

Task 2 (35%): Data Wrangling

As your second task, you will explore how your CSV-parser would be used in the processes of data ingestion and wrangling. Along with some more code, you will need to provide in-depth analysis to illustrate the wrangling process in the jupyter notebook with your submission.

First, using your implementation from Task 1 to read the data, write another function which computes the minimum, maximum, mean, and standard deviation for each component of the weather data and report these statistics in the jupyter notebook. You may use NumPy, Pandas, or the inbuilt Python math library to calculate these values.

Then, using Python or another editor of your choice (e.g. Excel), modify one of the CSV files to include some plausible incorrect data which is not malformed but of the kind that you may wish to investigate during the data wrangling process. An easy example would be to introduce one or more outliers into the data. Describe the modification and explain your rationale for introducing it in the markdown cell of the jupyter notebook with your submission.

Finally, use the same function you wrote earlier to re-calculate the summary statistics on the modified CSV file, and report the new results against the old ones. Comment on the differences, and whether the statistics alone would be sufficient to identify the incorrect data. Is there any additional statistical analysis that may be helpful instead / in addition?

Please note: your notebook submission for Task 2 must be a self-contained report with all the supporting statistics pre-generated to guide the discussions in the markdown cells. At the same time, all the relevant code must be there if the marker wants to re-run it for validation. Please follow the submission template provided in *ApAi_GA1_Template.ipynb*.

Assessment Criteria for Task 2	**Marks Awarded**
Does the program correctly output the minimum, maximum, mean, and standard deviation for every component in the provided CSV files?	5
Has one of the CSV files been modified to include some plausible incorrect data, and does the rationale in the jupyter notebook support the choice?	5
Does the program correctly output the summary statistics for the new CSV file?	5
Does the jupyter notebook present a discussion of the differences between the statistics, and give a convincing analysis as to whether they would be sufficient to identify the incorrect data?	20