CSCI 5015 Assignment 2 Updated
Due Feb 16, 11:59 PM.
50 points

Objective

Work with dictionaries and learn about Python's **`csv`** library.

Background

You've been given a data file named **Clarksville_weather_history.csv**. This file contains daily information gathered at Clarksville's main weather station from April 1, 2001 to December 13, 2017. The data was downloaded from NOAA's National Centers for Environmental Information, which "is responsible for preserving, monitoring, assessing, and providing public access to the Nation's treasure of climate and historical weather data and information."

The file has 10 columns

- STATION - weather station id
- NAME - weather station name
- DATE - date of the reading in the format YYYY-MM-DD (e.g. 2017-03-02 is March 2, 2017)
- PRCP - precipitation
- SNOW - snowfall
- SNWD - snow depth
- TAVG - average temperature
- TMAX - maximum daily temperature
- TMIN - minimum daily temperature

Reading the file

The data file is a Comma Separated Value (CSV) file, but notice it has some extra double quotes around all the values. We haven't seen that before. For example,

```
"USW00003894","CLARKSVILLE OUTLAW AIRPORT, TN US","2001-04-01",,,,"46","56","35"
```

The reason for this is that the NAME has a comma as part of the name. The quotes are telling the people and programs that use the file that commas inside the quotes are not field separators, but are actually part of the data. This means, if we read the data as I demonstrated in class and used **`split()`** on each line like this

```
for line in dataIn:
   line = line.strip()
   line = line.replace('"', '') # get rid of the pesky double quotes
   fields = line.split(',')
```

and then printed the line we just read, we would see this:

```
['USW00003894', 'CLARKSVILLE OUTLAW AIRPORT', ' TN US', '2001-04-01', '', '', '', '46', '56', '35']
```

Notice the NAME got split into two values, **`'CLARKSVILLE OUTLAW AIRPORT'`** and **`' TN US'`**. That is incorrect. The name should be one long value **`CLARKSVILLE OUTLAW AIRPORT, TN US`**. Using **`split()`** and removing the double quotes is exactly the wrong thing to do. So, how do we read this data?

I'm glad you asked. We *could* change the code we did in class to handle the quotes and split the line correctly. But, honestly, that's a pain. So, instead, we are going to use a module that comes with Python. The module is named **`csv`** and it contains a tool called **`DictReader`**.

`DictReader` does the same thing we did in class, that is, read a CSV file which has labels on the first row. After reading the first row, it turns each row into a **`dict`** value (actually, the value is a **`OrderedDict`**, which is the same a **`dict`**, at least for our purposes). The nice part is that **`DictReader`** handles splitting the fields correctly and handling CSV values with commas inside the double quotes.

When **`DictReader`** reads the file, it make all the values strings. That is fine for the NAME column, but not for columns like TAVG, which is a number. Another issue with the file is that many rows have fields that are missing values. When **`DictReader`** reads a row and finds a field with an empty value, it will add an an empty string to the **`dict`** for that field. To deal with both these issues, you can modify the data returned from reading a line with **`DictReader`** using an if-statement similar to the one below. You can do this for any field you need to work with. You can ignore values of fields you don't actually use.

```
        if len(row['TAVG']) > 0:
            # convert string to float number
            row['TAVG'] = float(row['TAVG'])
        else:
            # no value
            row['TAVG'] = None
```

In Python, **`None`** means "no value." It is similar to **`null`** in other languages. In later parts of the code, you will need to process the numbers, but you will also need to avoid the **`None`** value. If you don't skip the **`None`** value, it is very likely your code will crash. A simple if-statement will let you ignore them. For example, if we were looping through all the data, and we only wanted to print TAVG values that had a number, we could put this if-statement inside the loop:

```
      # only print the row if it has a value in it.
      if row['TAVG'] is not None:
         print(row['TAVG'])
```

Coding instructions

Create a Python file named **assignment2.py** and place the data file **Clarksville_weather_history.csv** in the same folder. Add a comment at the top of your file with your name, for example,

```
    # Written by John Nicholson
```

Your program should read the data file using **`DictReader`** and clean up the numbers as described above. You will need to research how to use **`DictReader`** through Google searches. Make sure you are looking for **Python 3** examples so that you see sample code written in the same version of Python as we are using. Once you know you have read the file correctly, compute the following values:

- The highest average temperature (TAVG) and the day it occurred (DATE)
- The lowest average temperature (TAVG) and the day it occurred (DATE)
- The highest maximum temperature (TMAX) and the day it occurred (DATE)
- The lowest maximum temperature (TMAX) and the day it occurred (DATE)
- The highest minimum temperature (TMIN) and the day it occurred (DATE)
- The lowest minimum temperature (TMIN) and the day it occurred (DATE)
- The highest precipitation (PRCP) and the day it occurred (DATE)
- The highest snow fall (SNOW) and the day it occurred (DATE)

UPDATE: If more than one day has the highest value, then pick the first or last one to print. If more than one day has the lowest value, then pick the first or last one to print.

UPDATE: As mentioned in the D2L News section, the data is noiser than expected. Do not include 0 or empty values in your calculations for the highest and lowest. Instead, for each value above, you should also output

- The number of values in the column (TMAX, TMIN, TAVG, PRCP, SNOW) that are empty
- The number of values in the column (TMAX, TMIN, TAVG, PRCP, SNOW) that are 0

Output your results to the file **results.txt**. The first line of the output should be a statement with your name, for example,

```
      Prepared by John Nicholson
```

Ensure that all data is labeled descriptively. It should be obvious what each item means.

Tips and help

The data file is big. You might want to create a similar, yet smaller, file to test you program on. This way you can place values in the small file that you know the answers for. Write your program and test it on that file. Once you are sure your program works on the small file, change it to work on the big file.

You can always come see me in my office or send me email. If you send me email, you should send your code as an attachment. Don't copy/paste your code into the message because that will make it harder for me to debug your code. Send your file as a **`.py`** file

Submission

When your program is correct, log into D2L and locate the Dropbox for assignment 1. Upload the two files

- assignment2.py

- results.txt

into the D2L dropbox.

Contact me if you have any problems.