Read_Write-Python3.6demo

January 29, 2019

1 # Working with tab-delimited data

1.0.1 Objective: Store all mean discharge measurements that are acceptable for publication

Start by downloading some USGS discharge data for a local stream gauge. Look at the file brazos_discharge_usgs.txt. This is a tab-delimited file downloaded from waterdata.usgs.gov - you can see the online version yourself at: https://waterdata.usgs.gov/nwis/dv?cb_00060=on&cb_00065=on&format=rdb&site_no=08096500&referred_module=sw&period=&begin_date=1898-10-01&end_date=2017-07-30.

In the following examples, data from a tab-delimited text file will be read in. A subset of the data (all measurements acceptable for publication) will be selected and written to an output file.

1.1 Example 1: Read and write data line-by-line

In this example data is read from and written to a file line-by-line.

Every programming language comes equipped with shortcuts used for reading in data, and it's easy to become dependent on specialized tools. The problem is that most tools aren't universal across languages, so it's important to have skills that will translate. String parsing and manipulation are essential programming skills that are often undervalued or ignored by beginners. "Parsing" means to search for a particular set of characters within a string, and "string manipulation" means altering or, more commonly, subsetting elements of a string. This example will perform parsing using "regular expressions". A regular expression is a powerful tool for string comparison, which is really useful when manipulating text files (https://docs.python.org/2/library/re.html).

Notice that the first 37 lines all begin with the character #. This is a header character − any line that begins with # is metadata or a comment about the site and the measurements collected and should not be read in as data. When reading in the data, test each line to see if it begins with the header character. If it does, skip this line. Use a regular expression to test for the header character.

One last thing to do before subsetting and writing data is to make sure not to store the column headers and corresponding format lines as data. The column headers are the first line of text after the header ends; the format line is directly below it. The format line specifies the format and length of each column entry (e.g., 5s means the column is populated with strings that are 5 characters long). These lines do not start with a header character, so another way is needed to exclude them. Each data line begins with an agency code in the first column. A quick scan through the data reveals that the agency code is USGS for every line. In general, agency codes will be four characters long, so determine how long the first column entry is, then move on. It is also possible

to write a regular expression for this, but it is a good idea to practice multiple methods of working with strings.

The next thing to do is subset only the desired data. For this example, these are the mean daily discharge. Lines 20−36 of the *.txt file specify the codes to interpret column headers. So, store data for mean daily discharge, which will have column header 135079_00060_0003, or column 8. Notice that each data column is followed by a column with the same header plus the string_cd. These columns store qualification codes that specify whether the data are acceptable for publication (quality controlled). All USGS data are provisional for some time after collection, during which it is subject to review and coded with a P. Data acceptable for publication are coded with an A. Read in all mean discharge measurements from column 8 if they are coded as acceptable for publication in column 9.

Each data line will be read in as a tab-delimited string. It is always important to know the delimiter for the data; data formatted as a *.txt file collected from waterdata.usgs.gov will always be tab-delimited. Note that on line 10 of the file, there is a link describing the output format, which is worth reading. Because the delimiter is what splits the data into columns, this facilitates selecting only the required data. Recall the goal to store the datetime (column 3) and those mean daily discharges acceptable for publication (columns 8 and 9). Use a "regular expression" to test for the delimiter, test for the qualification code, then store the data if it meets publication standards.

Implementation algorithm:

1. Open files for reading and writing (always write close statement at the same time)
2. Read in file line-by-line
3. Skip over any line that begins with "#"
4. Split non-header lines into a list of strings (re.split(delimiter,line))
5. If the value in column 9 is "A", write a string of data to be written to the output file.
6. Test the length of the first list entry to match the length of the agency code
7. Write the data string to the output file.

One last thing: Python is a zero-based language. That means that the first index of an object is always zero. For example, look at the following list:

list = ['A','B','C','D']

- The first item of **list** is 'A' and has index zero: list[0] = 'A'
- The second item of **list** is 'B' and has index one: list[1] = 'B'
- etc...

So, if we want to get data from columns 3, 8, and 9, that corresponds to indicies 2, 7, and 8.

```
In [1]: # Import packages - only need to do this once
        import re  # this package has all the tools we need to work with regular expressions

        # define variables
        re_str = '^#'  # regular expression pattern to perform string matches, for later
        filename = 'brazos_discharge_usgs.txt'  # file to read

        # read in, store data, and write file
        fout = 'line_by_line_out.txt'
```

```
f = open(filename,'r')  # open the file for reading
fout = open(fout,'w')
col_head_out = 'datetime\t135079_00060_00003\n'
fout.write(col_head_out)  # write column header to output file
for line in f.readlines():  # read the file line by line
    header = re.search(re_str,line)  # use a regular expression to identify lines that
    begin w/ the header character '#'
    if header is None:  # if a line doesn't begin with '#',
    the output will be the type None
        lst = re.split('\t',line)
        if len(lst[0]) == 4:
            site_no = lst[1]
            dt = lst[2]
            val = lst[7]
            cd = lst[8]
            if cd is 'A':
                string = dt + '\t' + val + '\n'  # string to write to file
                with delimiter and newline char
                fout.write(string)
f.close()  # close the file once we've read every line
fout.close()  # close output file once every line is written
```

1.2 Example 2: Reading and writing data with NumPy and Pandas

Python has two popular packages for data analysis: NumPy and Pandas. NumPy provides basic functionality and is an essential package to learn for data analysis. Pandas is built on NumPy and provides high-level functionality, plus it's incredibly slick.

This time, data will be read into an array structure using both NumPy and Pandas. The great thing about both of these packages is that reading and writing data can be done using a single line of code. The read/write functions accept multiple arguments to specify the filename, delimiter, and relevant file information. In this example, the USGS comment header rows should be skipped using the arguments `skip_header` (NumPy) or `skiprows` (Pandas) and specifying the number of lines to skip. To make data processing easier, only bring in the data from desired columns instead of the entire file with the `usecols` argument and specifying a list of column numbers. Remember that Python indexing is zero based. By default, both packages will try to guess the data type for each column and perform conversions while reading in the data. Each package example will use the same implementation algorithm, with different command syntax.

Implementation Algorithm

1. Read in data from datetime, mean discharge, and publication code columns. Skip the USGS header.
2. Find the indicies of the publication code column that have the value 'A'.
3. Subset the data read in by selecting only rows where the publication code is 'A' using the indicies obtained in step 2.
4. Write output file.

```
In [2]: # NUMPY EXAMPLE ------------------------------------------------------------------

        import numpy as np  # import numpy package using shortname np
        data_np = np.genfromtxt(filename,
                                dtype=None,
                                delimiter='\t',
                                skip_header=39,
                                usecols=(2,7,8))  # read in data to array
        data_np = data_np.reshape(len(data_np),1)  # reshape data into 2D array
        data_np  # display data

Out[2]: array([[(b'1898-10-01', 1400., b'A')],
               [(b'1898-10-02', 1100., b'A')],
               [(b'1898-10-03',  750., b'A')],
               ...,
               [(b'2017-07-28', 1460., b'P')],
               [(b'2017-07-29', 1470., b'P')],
               [(b'2017-07-30',  247., b'P')]],
              dtype=[('f0', 'S10'), ('f1', '<f8'), ('f2', 'S6')])

In [3]: # NumPy: select discharge data acceptable for publicaion

        cd = data_np['f2']  # mean discharge code column
        A_idx = np.where(cd == b'A')  # Python 3.6 fix...
```

```
        #A_idx = np.where(cd == 'A')  # find rows where cd is A (acceptable for publication)
        data_np_A = data_np[A_idx[0],:]  # create subset of data where mean discharge cd is A
        '''print(cd)
        print(A_idx)'''
        dt = data_np_A['f0']  # select datetime column from data subset
        discharge = data_np_A['f1']  # select mean discharge  column from data subset
        data_np_final = np.concatenate((dt,discharge),axis=1)  # concatenate selected
        columns into single array

        # write data to output file
        np.savetxt('numpy_out.txt',data_np_final,fmt='%s',delimiter='\t',header=col_head_out)

        data_np_final  # display final dataset
```

Out[3]:
```
array([[b'1898-10-01', b'1400.0'],
       [b'1898-10-02', b'1100.0'],
       [b'1898-10-03', b'750.0'],
       ...,
       [b'2017-03-18', b'1290.0'],
       [b'2017-03-19', b'724.0'],
       [b'2017-03-20', b'620.0']], dtype='|S32')
```

In [4]:
```
# PANDAS EXAMPLE ----------------------------------------------------------------------

import pandas as pd  # import pandas package using the shortname pd
data_pd = pd.read_csv(filename,
                      dtype=None,
                      sep='\t',
                      skiprows=[38],
                      header=37,
                      usecols=(2,7,8))  # read in data

data_pd.head()  # display first few rows of data
```

Out[4]:

	datetime	135079_00060_00003	135079_00060_00003_cd
0	1898-10-01	1400.0	A
1	1898-10-02	1100.0	A
2	1898-10-03	750.0	A
3	1898-10-04	680.0	A
4	1898-10-05	610.0	A

In [5]:
```
# select discharge data acceptable for publicaion
cd = data_pd.loc[:,'135079_00060_00003_cd']
A_idx = cd.index[cd == 'A']
data_pd_A = data_pd.loc[A_idx,['datetime','135079_00060_00003']]

# write data to output file
data_pd_A.to_csv('pandas_out.txt',sep='\t',header=True,index=False,encoding='ascii')
```

```
data_pd_A.head()
```

Out[5]:

	datetime	135079_00060_00003
0	1898-10-01	1400.0
1	1898-10-02	1100.0
2	1898-10-03	750.0
3	1898-10-04	680.0
4	1898-10-05	610.0