Reading from More General Data Files

Leigh J. Little SUNY Brockport December 2, 2020

Contents

1 Introduction

This chapter discusses working with more general data files in Python. In previous lectures, we examined how to read data from a file in the case where the file consists only of a rectangular grid of numbers and how to read those values into individual lists or arrays. This is the easiest case, but this case does not occur as often as we would like.

Frequently, a data file will contain other information such as header lines that contain information such as where/when the data was gathered and headers indicating what the variables in the columns are. In addition, there may be situations where some of the data values are not present. We would like to be able to handle data files such as these.

It is important to note that there is no universal technique for reading data files. When writing a function to extract information from a data file, it is necessary to examine the data file itself to look for patterns in how the data and headers are arranged.

1.1 A Simple Example

An simple example of how a file of the type discussed above may look is given below:

```
Location ID:  45734
Date:  12/2/1945
Time:  16:45
---------------------------
Temp    Distance   Angle
 (C)         (m)   (deg)
---------------------------
57.6        1.60      35
58.3        1.87      39
...(more lines like this) ..   ## Important; the examples below assume that there are
                               ## no blank lines at the end of the files.
```

If the goal is to extract the data into individual columns, we can use a process that is more or less identical to the one given in the previous discussion. The only changes are that we need to include some read statements to account for the headers. We also need to consider that the first two columns are floats and the last column is an integer.

The example data file has 7 header lines. We need to read these lines, but we don't need to do anything with them. Once these are read, we can start inserting the values into the lists. Download the file `data_g1.dat` from the course web page. This contains the data in the example above.

```
T = []    # Initialize list for temp, distance, angle
D = []
A = []
with open('data_g1.dat','r') as fnum:
     i = 0
     for currline in fnum:
```

```
            t = currline.strip().split()  # strip leading/trailing spaces
                                          # and split into list elements
            i = i + 1;
            if i > 7:                     # start inserting with line 8
                 T.append(float(t[0]))
                 D.append(float(t[1]))
                 A.append(int(t[2]))
print(T)
print(D)
print(A)
```

The above code will give the output

```
[57.6, 58.3]
[1.6, 1.87]
[35, 39]
```

This correctly extracts the temperature into T, the distance into D and the angle into A.

2 Columns with Missing Data

It is not unusual to encounter data files where one or more values in the set is indeterminate. Generally when a value in a table is meant to be interpreted as indeterminate, one of three indicators will be used:

- Indicator Value - in this case, the field where the number is unavailable is filled with a numeric indicator value. This may be -999, NaN or some other special value that cannot occur within the range of legitimate values in the set.
- Character Indicator Value - in this case, the field where the number is unavailable is filled with some character value. A star, a short sequence of stars or a dash are the most common characters used. The sequence N/A is becoming more common.
- Missing Value - in this case, the field where the number is unavailable is simply left blank. This is the most difficult case to deal with.

An additional consideration when reading files with missing data is how should such situations be handled? Consider the example:

```
Location ID:  45734
Date:  12/2/1945
Time:  16:45
--------------------------
Temp    Distance   Angle
 (C)         (m)   (deg)
--------------------------
57.6        1.60      35
58.3        1.87      39
60.2        -999      43
```

There is a missing distance in the last row. Should that entire row be discarded, or should the -999 value be left in the set? As a general rule, if no additional information is known it is best to err on the side of caution and retain the data in some way. It is expensive to obtain real data and discarding adjacent, legitimate values may be considered a waste of resources. This leaves the decision about how to handle such values with the programmer who will be calling the function.

2.1 Indicator Value

This is the easiest case to deal with. Even though there is a placeholder value, it still behaves like a number. With respect to the process of reading in the values, nothing about the example in the previous section would need to be changed.

In the event that it is desired to not include an entire row with an indicator value, the code from the previous section can be modified to:

```
# Numeric indicator value (-999 in this example)
# Discard rows where indicator occurs
T = []    # Initialize list for temp, distance, angle
D = []
A = []
with open('data_g1.dat','r') as fnum:
     i = 0
     for currline in fnum:
          t = currline.strip().split()  # strip leading/trailing spaces
                                        # and split into list elements
          i = i + 1;
          if i > 7:                 # start inserting with line 8
               if int(t[0]) != -999 and int(t[1]) != -999 and int(t[2]) != -999:
                  T.append(float(t[0]))
                  D.append(float(t[1]))
                  A.append(int(t[2]))
print(T)
print(D)
print(A)
```

The only change is an `if` statement that checks if all values are different from -999. If all values are not equal to -999, then the current data is appended to the arrays. Otherwise, the line is skipped and not appended.

If there were many columns of data to read, then it would be better to handle this `if` statement using a process similar to the one below

```
found = 0
for j in range(len(t)):    # Search through t looking for indicator
    if t[j] == '-999':
       found = 1           # Set found = 1 if indicator is present
if found != 1:
   (statements to append data)
```

2.2 Character Indicator Value

This case is slightly more complicated than the previous example, but not much. As before, you need to decide if you will exclude the entire row of data. You also need to decide what to do with the character indicator value. Because the other elements in the list will be numerical, it is not recommended to keep the indicator as a character. This may cause problems with computational routines that may subsequently be applied to the array. Here, you have two main options

- Replace the character with a numerical indicator value. The potential drawback to this approach is that you need to ensure that the value you choose as an indicator is not a feasible value for the data. For example, if your data were recorded temperatures in the range -30C to 50C, you would not want to use 0 as your indicator value.
- Replace the character with NaN. This has some advantages over a numeric value. In this case, there is no need to guess as to what value to choose, so it is a more universal option. In addition, plotting routines in Python (and Matlab) will automatically ignore NaN values when generating plots. The drawback to using NaN is that NaN is treated a floating-point value, so this can't be used with integer arrays.

The code to handle this case (assuming we want to keep the row with the indeterminate data) is close to the code we just examined. The main change is that in addition to checking if the indicator occurs, we also need to keep track of where it occurs so that the correct array value will be changed.

```
# Character indicator value (* in this example)
# Keep these values and replace * by NaN for floats and -999 for integers
T = []    # Initialize list for temp, distance, angle
D = []
```

```
A = []
with open('data_g1.dat','r') as fnum:
     i = 0
     for currline in fnum:
          t = currline.strip().split()  # strip leading/trailing spaces
                                        # and split into list elements
          i = i + 1
          if i > 7:                     # start inserting with line 8
               found = 0
               ifound = -1              # Location where indicator found
               for j in range(len(t)):  # Loop over list to see if any
                                        # elements are *
                  if t[j] == '*':
                     found = 1          # Found a star
                     ifound = j         # It is in position ifound in t.
               if found == 1:           # If a star was found, adjust the element of t.
                  if ifound == 0 or ifound == 1:
                     t[ifound] = 'nan'
                  elif ifound == 2:
                     t[ifound] = '-999'
               T.append(float(t[0]))
               D.append(float(t[1]))
               A.append(int(t[2]))
print(T)
print(D)
print(A)
```

2.3 Empty Fields

The third common option for dealing with indeterminate data is to just leave that particular field blank. This is the most difficult case to process. To see why consider a data file that contains 3 columns

```
 1  2  3
 4  5  6
 7     9
10 11 12
```

In the example codes above, when the second line is read, the value of `t` will be

```
t = ['1','2','3']
```

However, when the third line is read, the value of `t` will be

```
t = ['7','9']
```

Because the length of `t` is two, we can tell that there is a value that is missing. What we can't easily determine is which value is missing. Is the missing value in column 0, column 1 or column 2?

To answer this is helps to again look at the data file for any pattern that you can use as an indicator. Look closely at the file `data_g2.dat`. In this example, each column is right justified. This means that if the value in a field is a legitimate numerical value, the last column of the field will be a digit.

We can thus search for a space where there should not be one as the indicator that there is a missing value. Before beginning, you need to go into the data file and manually count where the last digit in each field is located.

```
# Missing value
# Keep these values and replace by NaN for floats and -999 for integers
T = []    # Initialize list for temp, distance, angle
D = []
A = []
digpos = [2, 15, 23]     # List with positions of last digit in fields
```

```
                          # Remember to start numbering positions with 0 and not 1.
     with open('data_g2.dat','r') as fnum:
          i = 0
          for currline in fnum:
               twhole = currline             # Keep whole string for missing search
               t = currline.strip().split()  # strip leading/trailing spaces
                                             # and split into list element
               i = i + 1
               imiss = -1
               if i > 7:                                 # start inserting with line 8
                    for j in range(len(digpos)):         # Check for missing values
                                                         # We need to do this on the whole
                                                         # string, before .strip().split()

                        if twhole[digpos[j]] == ' ':     # Look for a space where there should
                                                         # not be one.
                             imiss = j                   # Missing data is in column j
# Need to insert missing value into correct place
                    if imiss != -1:
                      tnew = []                          # These lines create the tnew list
                      for j in range(3):                 # consisting of 3 spaces
                          tnew.append(' ')
                      for j in range(3):                 # Loop over tnew.  Insert either element of t
                          if j > imiss:                  # or the missing value into tnew depending on
                             tnew[j] = t[j-1]            # value of imiss
                          elif j == imiss:
                             if imiss == 0 or imiss == 1:
                                tnew[j] = 'nan'
                             elif imiss == 2:
                                tnew[j] = '-999'
                          elif j < imiss:
                             tnew[j] = t[j]
                      t = tnew
                    T.append(float(t[0]))
                    D.append(float(t[1]))
                    A.append(int(t[2]))
print(T)
print(D)
print(A)
```

This code gives the output

```
[1.0, 4.0, 7.0, 10.0]
[2.0, 5.0, nan, 11.0]
[3, 6, 9, 12]
```

In the event you just want to discard the rows with missing values, the code becomes much simpler:

```
# Missing value
# Discard rows with missing values
T = []    # Initialize list for temp, distance, angle
D = []
A = []
digpos = [2, 15, 23]     # List with positions of last digit in fields
                         # Remember to start numbering positions with 0 and not 1.
with open('data_g2.dat','r') as fnum:
     i = 0
     for currline in fnum:
          twhole = currline             # Keep whole string for missing search
```

```
                t = currline.strip().split()  # strip leading/trailing spaces
                                              # and split into list element
                i = i + 1
                imiss = -1
                if i > 7:                                 # start inserting with line 8
                     for j in range(len(digpos)):         # Check for missing values
                                                          # We need to do this on the whole
                                                          # string, before .strip().split()

                          if twhole[digpos[j]] == ' ':    # Look for a space where there should
                                                          # not be one.
                               imiss = j                  # Missing data is column j
# Insert if no values missing.
                     if imiss == -1:   # If no missing data, append to lists.  Nothing happens
                                       # if there is missing data.
                        T.append(float(t[0]))
                        D.append(float(t[1]))
                        A.append(int(t[2]))
print(T)
print(D)
print(A)
```

Note that we have only covered the most basic situations here. Some additional complications that can occur would be the case where a line has several missing values or a file that has a mix of missing values and indicator values.