Py Notes

Applications and examples...

Published - 2016-04-02
Dan Patterson

Part I Summarize and Classify Observations

This example shows how to go from individual observations to a summary of the results. The example is loosely based on a real world case.

To begin, you are tasked to collect data, organize and present it in a coherent fashion. The choice of how to collect and organize is often as important as what you collect. Reclassification may be part of the process. In this example, the results of a origin-destination travel time study to a location will be presented.

The data, d, represent the results of a travel time study within a Township (A) and its various villages (A_ to Z_) to a location (Hosp). The last column are the travel times to the destination. Each row represents one person's observed travel time.

```
d = [(u'A', u'A_', u'Hosp',  6),
     (u'A', u'A_', u'Hosp', 25),
     (u'A', u'A_', u'Hosp', 21),
     (u'A', u'A_', u'Hosp', 22),
     (u'A', u'A_', u'Hosp', 22),
     (u'A', u'B_', u'Hosp', 51),
     (u'A', u'B_', u'Hosp', 18),
     (u'A', u'B_', u'Hosp',  4),
     (u'A', u'C_', u'Hosp', 55),
     (u'A', u'C_', u'Hosp', 40)]
```

The u in front of the text values simply indicate that the data are using a Unicode encoding. Data encoding allows for different character sets/alphabets to be used. In this study, the encoding is note relevant.

The task is to present the data in various formats and to summarize it.

To begin, the data needs to be reclassified into travel time categories so that the average travel times from the towns to the location can be derived. The categories that will be used are

limits = 0, 5, 10, 15, 20, 25, 30 and 60

These numbers represent the lower limits of the travel times with the exception of the last value which closes the possible classification range. Hence, the category 10 represents the travel times from 10 - < 15 units, inclusive of 10 but exclusive of 15. This is standard statistical practice.

The data will be analyzed using NumPy since it facilitates this type of array format.

A spreadsheet could have also been used. The data can be given a structure and an array produced. Several variables are defined to enable reuse of code snippets. They are defined as follows:

```
d_names=['A','B','C']          # used in 2 places
d_type = ['U20','U20','U20']   # unicode encoding
dt0= list(zip(d_names,d_type)) # see above
dt1 = dt0 + [('Time','<i4')]   # data array's dtype
#
# ---- dtype for full array, bins, bin classes and unique
bins = [0, 5, 10, 15, 20, 25, 30, 60]
binclass = [('t'+str(i),'<i4') for i in bins[:-1]] # skip last (60)
```

The d_names variable is a list used to denote the column headings. Each of those columns uses a 20 character unicode string data type (d_type). The data types and the field names for those three columns are combined together. In python, the zip function makes this possible by combining the pairing of column name and datatype. The use of 'list' is python 3.x requirement and is used to convert a zip object into a list object. The last column is different so its column heading and data type are added to the previously constructed data type. The result is a final dtype expressed below:

```
dt1 = [ ('A','U20'), ('B','U20'),
        ('C','U20'), ('Time',np.int32) ]
```

The array can be constructed from the format structure and the unique values.

```
a = np.array(data, dtype=dt1)
uni = np.unique(a[d_names])
```

The array, a, was created using 'd' and the specified dtype. The unique values were similarly derived and the data are ready for processing.

The whole process can be described as a sequence of steps:

- Organize the data and construct into array form.
- Find the unique values in the data, in this case, the first three columns, (A, B and C). These columns represent the facility (C) being visited by people in the villages (B) within the township (A). By constructing the data and query in this fashion, we could broaden the coding to include other townships, their villages and/or the type of facility.
- Once the unique observations are obtained, extract the final piece of information, the time data. An obvious tip ... don't include a field that is unique to begin with, such as an ID field or anything with a sequential continuous pattern.
- The time data can then be classified.

This can be expressed as:

```
vals = a[a[d_names] == u ]['Time'].tolist()
```

which looks pretty onerous, so stepwise, the above line means:

```
vals = a[                   Extract from the array, a,
          a[d_names] == u   all the rows that are equal to each
        ]                   unique condition.
        ['Time']            Extract the time information from the row.
        .tolist()           Convert the data to facilitate summary.
```

The complete code is shown below.

```
# ---- Begin by constructing dtype by parts... useful later on
#      dt0 = [('A','U20'),('B','U20'),('C','U20')]
d_names=['A','B','C']          # used in 2 places
d_type = ['U20','U20','U20']   # unicode encoding
dt0= list(zip(d_names,d_type)) # see above
dt1 = dt0 + [('Time','<i4')]   # data array's dtype
#
# ---- dtype for full array, bins, bin classes and unique
bins = [0, 5, 10, 15, 20, 25, 30, 60]
binclass = [('t'+str(i),'<i4') for i in bins[:-1]] # skip last (60)
#
a = np.array(data, dtype=dt1)
uni = np.unique(a[d_names])
#
out =[]
for u in uni:
    vals = a[a[d_names] == u ]['Time'].tolist()
    hist, edges = np.histogram(vals,bins)
    result = tuple(u) + tuple(hist)
    out.append(result)
dt2 = dt0 + binclass
out_array = np.array(out,dtype=dt2)
print("\nFinal array\n{!s:}".format(out_array))
```

And the results are:

```
Final array
[(u'A', u'A_', u'Hosp', 0, 1, 0, 0, 3, 1, 0)
 (u'A', u'B_', u'Hosp', 1, 0, 0, 1, 0, 0, 1)
 (u'A', u'C_', u'Hosp', 0, 0, 0, 0, 0, 0, 2)]
```

with the columns: **Township, Village and Facility, then times of 0, 5, 10, 15, 20, 25, 30-60**

Part II Reclassification from Groupings

Now sometimes, you are stuck with data that are already grouped and you want to view the set in a different light. The previous example left off at the point where travel times to a facility were being aggregated from individual observations into travel time groupings. These groupings could have been further summarized like the data shown in the table to the right.

```
d =[('A','A_','Hosp',60,411),
    ('A','B_','Hosp',20,  6),
    ('A','B_','Hosp',25,148),
    ('A','C_','Hosp',60, 51),
    ('A','D_','Hosp',60,133),
    ('A','E_','Hosp',20, 25),
    ('A','F_','Hosp',15, 38),
    ('A','F_','Hosp',20,102),
    ('A','S_','Hosp',30, 15),
    ('A','S_','Hosp',60, 27),
    ('A','T_','Hosp',30, 32),
    ('A','T_','Hosp',60, 36),
    ('A','U_','Hosp',25,  3),
    ('A','U_','Hosp',30, 59),
    ('A','V_','Hosp',60, 32),
    ('A','W_','Hosp',20, 84),
    ('A','W_','Hosp',25, 17),
    ('A','X_','Hosp',20, 50),
    ('A','X_','Hosp',25,235),
    ('A','Y_','Hosp',20, 15),
    ('A','Y_','Hosp',25,101),
```

The columns now are:

Township, Village, Facility, Time and People

The first row indicates that 411 people travelled to the facility and they fell into the 30-60 unit category. The next two rows, indicate 6 people were within the 15-20 unit time category while 148 were with 20-25 time units. Both groups came from the same village, that bucolic hamlet of B_, affectionately know as Bunderscore.

These data, although nicely summarized, do mask the temporal pattern shown in the previous example. The task at hand now is how to unravel the pattern so that it is similar to that derived from the previous groupings. The process is essentially the same as that followed in the previous example.

- Generate and array from the data, conserving input data types.
- Determine the unique data groupings using Township, Village and Facility as the keys.
- Use the time classes as previously outlined and unravel the original data into its new form.

The results are as follows and the column headings still:

Township, Village and Facility, then times of 0, 5, 10, 15, 20, 25, 30-60

```
Output array...
[(u'A', u'A_', u'Hosp', 0, 0, 0, 0, 0, 0, 411)
 (u'A', u'B_', u'Hosp', 0, 0, 0, 6, 148, 0, 0)
 (u'A', u'C_', u'Hosp', 0, 0, 0, 0, 0, 0, 51)
 (u'A', u'D_', u'Hosp', 0, 0, 0, 0, 0, 0, 133)
 (u'A', u'E_', u'Hosp', 0, 0, 0, 25, 0, 0, 0)
 (u'A', u'F_', u'Hosp', 0, 0, 38, 102, 0, 0, 0)
 (u'A', u'S_', u'Hosp', 0, 0, 0, 0, 0, 15, 27)
 (u'A', u'T_', u'Hosp', 0, 0, 0, 0, 0, 32, 36)
 (u'A', u'U_', u'Hosp', 0, 0, 0, 0, 3, 59, 0)
 (u'A', u'V_', u'Hosp', 0, 0, 0, 0, 0, 0, 32)
 (u'A', u'W_', u'Hosp', 0, 0, 0, 84, 17, 0, 0)
 (u'A', u'X_', u'Hosp', 0, 0, 0, 50, 235, 0, 0)
 (u'A', u'Y_', u'Hosp', 0, 0, 0, 15, 101, 0, 0)
 (u'A', u'Z_', u'Hosp', 0, 2, 11, 0, 0, 0, 0)]
```

The process is still the same. The code is essentially the same, with some minor variations introduced to facilitate understanding.

```
dtv = [('A','U20'),('B','U20'),('C','U20'),
       ('Time',np.int32), ('People',np.int32)]
v = np.array(d,dtype=dtv)         # create an array with dtv
uni = np.unique(v[['A','B','C']]) # get the unique on first 3 cols
shp = (len(uni),)                 # number of unique rows
# construct the output arrays
dt = [('A','U20'),('B','U20'),('C','U20'),('t05',np.int32),
      ('t10',np.int32), ('t15',np.int32),('t20',np.int32),
      ('t25',np.int32), ('t30',np.int32),('t60',np.int32)]
a = np.zeros(shp, dtype=dt)
a['A'] = uni['A']   # just assign the first 3 columns
a['B'] = uni['B']   # the values from the unique test
a['C'] = uni['C']   # everything is sorted!!!

for i in range(len(uni)): # show time... need to vectorize
    r = v[v[['A','B','C']] == uni[i] ] # pull out the rows that match
    for j in range(len(r)):
        col = 't'+ str(r[j]['Time'])
        num = r[j]['People']
        a[i][col] = num
print("\nOutput array...\n{!s:}".format(a))
```

A fairly simple and reusable process. Graphing and further aggregation would be the next steps.

Appendix I Full Script with some additions

```
# coding: utf-8
"""
Script:   observation_summary_demo_2016_04_02.py
Author:   Dan.Patterson@carleton.ca
Modified: 2016-04-02
Purpose:  Demonstrate the use of np.unique and np.histogram for data
    summary and presentation.
Reference:  https://geonet.esri.com/thread/174845

"""
import numpy as np
np.set_printoptions(edgeitems=3,linewidth=80,precision=2, suppress=True,threshold=20)

d =  load data here

#County, Town, Facility, Time, People
dt = [('A','U20'),('B','U20'),('C','U20'),('5',np.int32),('10',np.int32),('15',np.int32),
      ('20',np.int32),('25',np.int32), ('30',np.int32),('60',np.int32)]

dtv = [('A','U20'),('B','U20'),('C','U20'), ('Time',np.int32), ('People',np.int32)]
v = np.array(d,dtype=dtv)         # create an array with dtv
uni = np.unique(v[['A','B','C']]) # get the unique on first 3 cols
shp = (len(uni),)                 # number of unique rows
# construct the output arrays
dt = [('A','U20'),('B','U20'),('C','U20'),('t05',np.int32),
      ('t10',np.int32), ('t15',np.int32),('t20',np.int32),
      ('t25',np.int32), ('t30',np.int32),('t60',np.int32)]
a = np.zeros(shp, dtype=dt)
a['A'] = uni['A']   # just assign the first 3 columns
a['B'] = uni['B']   # the values from the unique test
a['C'] = uni['C']   # everything is sorted!!!

for i in range(len(uni)): # show time... need to vectorize
    r = v[v[['A','B','C']] == uni[i] ] # pull out the rows that match
    for j in range(len(r)):
        col = 't'+ str(r[j]['Time'])
        num = r[j]['People']
        a[i][col] = num
print("\nOutput array...\n{!s:}".format(a))
```

```
# demo   TOWN, SETTLEMENT, NAME, TIME
data = load data here
# ---- Begin by constructing dtype by parts... useful later on
#      dt0 = [('A','U20'),('B','U20'),('C','U20')]
d_names=['A','B','C']          # used in 2 places
d_type = ['U20','U20','U20']   # unicode encoding
dt0= list(zip(d_names,d_type)) # see above
dt1 = dt0 + [('Time','<i4')]   # data array's dtype
#
# ---- dtype for full array, bins, bin classes and unique
bins = [0, 5, 10, 15, 20, 25, 30, 60]
bin_dt = [('t'+str(i),'<i4') for i in bins[:-1]] # skip last (60)
stats_dt = [('Mean','<f8'),('Min','<f8'),('Max','<f8')]
#
#
a = np.array(data, dtype=dt1)
uni = np.unique(a[d_names])
#
out =[]
for u in uni:
    vals = a[a[d_names] == u ]['Time'] #.tolist()
    hist, edges = np.histogram(vals.tolist(),bins)
    stats = (vals.mean(), vals.min(), vals.max())
    result = tuple(u) + tuple(hist) + stats
    out.append(result)
# ---- combine the dtypes, produce and print the array
dt2 = dt0 + bin_dt + stats_dt
out_array = np.array(out,dtype=dt2)
print("final array\n{!s:}\nfields {}".format(out_array, out_array.dtype.names))
```