hw03

March 28, 2020

1 Homework 3: Table Manipulation and Visualization

Reading: * Visualization

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests. Each time you start your server, you will need to execute this cell again to load the tests.

Homework 3 is due Thursday, 2/6 at 11:59pm. Start early so that you can come to office hours if you're stuck. Check the website for the office hours schedule. Late work will not be accepted as per the policies of this course.

[9]:
```
# Don't change this cell; just run it.

import numpy as np
from datascience import *

# These lines do some fancy plotting magic.\n",
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')
```

1.1 1. Differences between Universities

Question 1.1. Suppose you're choosing a university to attend, and you'd like to *quantify* how *dissimilar* any two universities are. You rate each university you're considering on several numerical traits. You decide on a very detailed list of 1000 traits, and you measure all of them! Some examples:

- The cost to attend (per year).
- The average Yelp review of nearby Thai restaurants.
- The USA Today ranking of the Medical school.
- The USA Today ranking of the Engineering school.

You decide that the dissimilarity between two universities is the *total* of the differences in their traits. That is, the dissimilarity is:

- the **sum** of
- the absolute values of

- the 1000 differences in their trait values.

In the next cell, we've loaded arrays containing the 1000 trait values for Stanford and Berkeley. Compute the dissimilarity (according to the above technique) between Stanford and Berkeley. Call your answer `dissimilarity`. Use a single line of code to compute the answer.

Note: The data we're using aren't real – we made them up for this exercise, except for the cost-of-attendance numbers, which were found online.

[10]:
```
stanford = Table.read_table("stanford.csv").column("Trait value")
berkeley = Table.read_table("berkeley.csv").column("Trait value")

dissimilarity = sum (abs(stanford-berkeley))
dissimilarity
```

[10]:
```
14060.558701067917
```

Question 1.2. Why do we sum up the absolute values of the differences in trait values, rather than just summing up the differences?

Write your answer here, replacing this text.

Weighing the traits After computing dissimilarities between several schools, you notice a problem with your method: the scale of the traits matters a lot.

Since schools cost tens of thousands of dollars to attend, the cost-to-attend trait is always a much bigger *number* than most other traits. That makes it affect the dissimilarity a lot more than other traits. Two schools that differ in cost-to-attend by $900, but are otherwise identical, get a dissimilarity of 900. But two schools that differ in graduation rate by 0.9 (a huge difference!), but are otherwise identical, get a dissimilarity of only 0.9.

One way to fix this problem is to assign different "weights" to different traits. For example, we could fix the problem above by multiplying the difference in the cost-to-attend traits by .001, so that a difference of $900 in the attendance cost results in a dissimilarity of $900 $\times$ .001, or 0.9.

Here's a revised method that does that for every trait:

1. For each trait, subtract the two schools' trait values.
2. Then take the absolute value of that difference.
3. Now multiply that absolute value by a trait-specific number, like .001 or 2.
4. Now, sum the 1000 resulting numbers.

Question 1.3. Suppose you've already decided on a weight for each trait. These are loaded into an array called `weights` in the cell below. `weights.item(0)` is the weight for the first trait, `weights.item(1)` is the weight for the second trait, and so on. Use the revised method to compute a revised dissimilarity between Berkeley and Stanford.

Hint: Using array arithmetic, your answer should be almost as short as in question 1.

[11]:
```
weights = Table.read_table("weights.csv").column("Weight")

revised_dissimilarity = sum(abs(stanford-berkeley)*weights)
revised_dissimilarity
```

[11]:
```
505.98313211458805
```

1.2 2. Unemployment

The Federal Reserve Bank of St. Louis publishes data about jobs in the US. Below, we've loaded data on unemployment in the United States. There are many ways of defining unemployment, and our dataset includes two notions of the unemployment rate:

1. Among people who are able to work and are looking for a full-time job, the percentage who can't find a job. This is called the Non-Employment Index, or NEI.
2. Among people who are able to work and are looking for a full-time job, the percentage who can't find any job *or* are only working at a part-time job. The latter group is called "Part-Time for Economic Reasons", so the acronym for this index is NEI-PTER. (Economists are great at marketing.)

The source of the data is here.

Question 2.1. The data are in a CSV file called `unemployment.csv`. Load that file into a table called `unemployment`.

[12]:
```
unemployment = Table.read_table ('unemployment.csv')
unemployment
```

[12]:
```
Date       | NEI     | NEI-PTER
1994-01-01 | 10.0974 | 11.172
1994-04-01 | 9.6239  | 10.7883
1994-07-01 | 9.3276  | 10.4831
1994-10-01 | 9.1071  | 10.2361
1995-01-01 | 8.9693  | 10.1832
1995-04-01 | 9.0314  | 10.1071
1995-07-01 | 8.9802  | 10.1084
1995-10-01 | 8.9932  | 10.1046
1996-01-01 | 9.0002  | 10.0531
1996-04-01 | 8.9038  | 9.9782
... (80 rows omitted)
```

Question 2.2. Sort the data in descending order by NEI, naming the sorted table `by_nei`. Create another table called `by_nei_pter` that's sorted in descending order by NEI-PTER instead.

[13]:
```
by_nei = unemployment.sort ('NEI')
by_nei_pter = unemployment.sort ('NEI-PTER', descending = True)
```

Question 2.3. Use `take` to make a table containing the data for the 10 quarters when NEI was greatest. Call that table `greatest_nei`.

[14]:
```
greatest_nei = by_nei.take(np.arange (10))
greatest_nei
```

[14]:
```
Date       | NEI    | NEI-PTER
2000-01-01 | 7.6128 | 8.3379
2000-04-01 | 7.6754 | 8.4199
2000-10-01 | 7.6769 | 8.4192
2000-07-01 | 7.6915 | 8.4458
2001-01-01 | 7.757  | 8.4969
1999-10-01 | 7.7832 | 8.541
1999-01-01 | 7.8933 | 8.7141
```

```
1999-07-01 | 7.8956 | 8.7204
1998-04-01 | 7.9992 | 8.8923
2001-04-01 | 8.0353 | 8.804
```

Question 2.4. It's believed that many people became PTER (recall: "Part-Time for Economic Reasons") in the "Great Recession" of 2008-2009. NEI-PTER is the percentage of people who are unemployed (and counted in the NEI) plus the percentage of people who are PTER. Compute an array containing the percentage of people who were PTER in each quarter. (The first element of the array should correspond to the first row of `unemployment`, and so on.)

Note: Use the original `unemployment` table for this.

[15]:
```
pter = unemployment.column ('NEI-PTER') - unemployment.column ('NEI')
pter
```

[15]:
```
array([1.0746, 1.1644, 1.1555, 1.129 , 1.2139, 1.0757, 1.1282, 1.1114,
       1.0529, 1.0744, 1.1004, 1.0747, 1.0705, 1.0455, 1.008 , 0.9734,
       0.9753, 0.8931, 0.9451, 0.8367, 0.8208, 0.8105, 0.8248, 0.7578,
       0.7251, 0.7445, 0.7543, 0.7423, 0.7399, 0.7687, 0.8418, 0.9923,
       0.9181, 0.9629, 0.9703, 0.9575, 1.0333, 1.0781, 1.0675, 1.0354,
       1.0601, 1.01  , 1.0042, 1.0368, 0.9704, 0.923 , 0.9759, 0.93  ,
       0.889 , 0.821 , 0.9409, 0.955 , 0.898 , 0.8948, 0.9523, 0.9579,
       1.0149, 1.0762, 1.2873, 1.4335, 1.7446, 1.8415, 1.9315, 1.8859,
       1.8257, 1.9067, 1.8376, 1.8473, 1.7993, 1.8061, 1.7651, 1.7927,
       1.7286, 1.6387, 1.6808, 1.6805, 1.6629, 1.6253, 1.6477, 1.6298,
       1.4796, 1.5131, 1.4866, 1.4345, 1.3675, 1.3097, 1.2319, 1.1735,
       1.1844, 1.1746])
```

Question 2.5. Add `pter` as a column to `unemployment` (named "PTER") and sort the resulting table by that column in descending order. Call the table `by_pter`.

Try to do this with a single line of code, if you can.

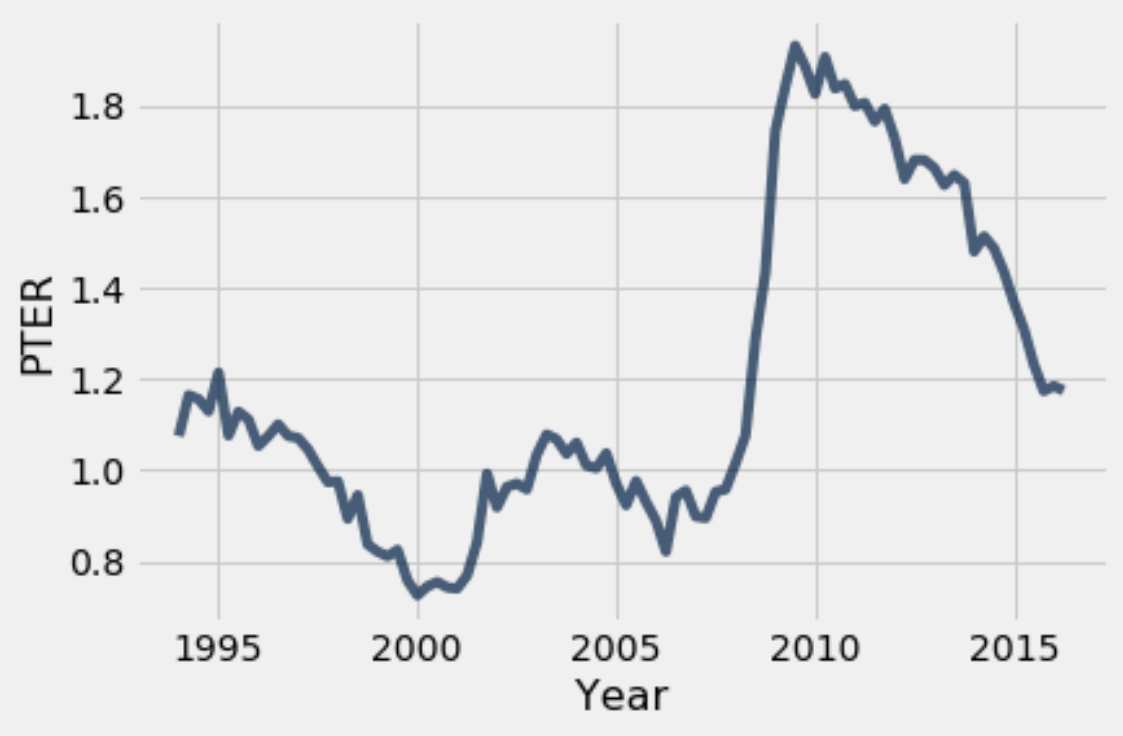
[16]:
```
by_pter = unemployment. with_column ('PTER', pter).sort('PTER', descending =␣
 ↪True)
by_pter
```

[16]:
```
Date       | NEI     | NEI-PTER | PTER
2009-07-01 | 10.8089 | 12.7404  | 1.9315
2010-04-01 | 10.6597 | 12.5664  | 1.9067
2009-10-01 | 10.9698 | 12.8557  | 1.8859
2010-10-01 | 10.5856 | 12.4329  | 1.8473
2009-04-01 | 10.7082 | 12.5497  | 1.8415
2010-07-01 | 10.5521 | 12.3897  | 1.8376
2010-01-01 | 10.9054 | 12.7311  | 1.8257
2011-04-01 | 10.4409 | 12.247   | 1.8061
2011-01-01 | 10.5024 | 12.3017  | 1.7993
2011-10-01 | 10.3287 | 12.1214  | 1.7927
... (80 rows omitted)
```

Question 2.6. Create a line plot of the PTER over time. To do this, first add the `year` array and the `pter` array to the `unemployment` table; label these columns "Year" and "PTER", respectively. Then, generate a line plot using one of the table methods you've learned in class. Assign this new

table to `pter_over_time`.

[17]:
```
year = 1994 + np.arange(by_pter.num_rows)/4
pter_over_time = unemployment.with_columns ("Year",year, "PTER", pter)
pter_over_time.plot ("Year", "PTER")
```



Question 2.7. Were PTER rates high during or directly after the Great Recession (that is to say, were PTER rates particularly high in the years 2008 through 2011)? Assign highPTER to `True` if you think PTER rates were high in this period, and `False` if you think they weren't.

[18]:
```
highPTER = True
```

1.3 3. Birth Rates

The following table gives census-based population estimates for each state on both July 1, 2015 and July 1, 2016. The last four columns describe the components of the estimated change in population during this time interval. **For all questions below, assume that the word "states" refers to all 52 rows including Puerto Rico & the District of Columbia.**

The data was taken from here.

If you want to read more about the different column descriptions, go here! As of February 2017, no descriptions were posted for 2010 - 2016.

[19]:
```
# Don't change this cell; just run it.
pop = Table.read_table('nst-est2016-alldata.csv').where('SUMLEV', 40).
 ↪select([1, 4, 12, 13, 27, 34, 62, 69])
```