assignment2

October 15, 2019

1 Homework 2: Tables and Visualization

Please complete this notebook by filling in the cells provided.

For all problems that you must write our explanations and sentences for, you **must** provide your answer in the designated space. Moreover, throughout this homework and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on.

Directly sharing answers is not okay, but discussing problems with the instructor or with other students is encouraged. Refer to the syllabus page to learn more about how to learn cooperatively.

You should start early so that you have time to get help if you're stuck.

[2]:
```
# 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')
```

2 Tables

Question 1. Suppose you have 4 apples, 3 oranges, and 3 pineapples. (Perhaps you're using Python to solve a high school Algebra problem.) Create a table that contains this information. It should have two columns: "fruit name" and "count". Give it the name `fruits`.

Note: Use lower-case and singular words for the name of each fruit, like `"apple"`.

[3]:
```
fruits = Table().with_columns(
    'fruit name', make_array('apple','orange','pineapple'),
    'count', make_array(4,3,3)
)
fruits
```

[3]:
```
fruit name | count
apple      | 4
orange     | 3
pineapple  | 3
```

Question 2. The file `inventory.csv` contains information about the inventory at a fruit stand. Each row represents the contents of one box of fruit. Load it as a table named `inventory`.

[4]:
```
inventory = Table.read_table('inventory.csv')
inventory
```

[4]:
```
box ID | fruit name | count
53686  | kiwi       | 45
57181  | strawberry | 123
25274  | apple      | 20
48800  | orange     | 35
26187  | strawberry | 255
57930  | grape      | 517
52357  | strawberry | 102
43566  | peach      | 40
```

Question 3. Each box at the fruit stand contain a different fruit – true or false?

[5]:
```
# Set all_different to True if each box contains a different fruit
#or to False if multiple boxes contain the same fruit
all_different = False
all_different
```

[5]:
```
False
```

Question 4. The file `sales.csv` contains the number of fruit sold from each box last Saturday. It has an extra column called "price per fruit ($)" that's the price *per item of fruit* for fruit in that box. The rows are in the same order as the `inventory` table. Load these data into a table called `sales`.

[6]:
```
sales = Table.read_table('sales.csv')
sales
```

[6]:
```
box ID | fruit name | count sold | price per fruit ($)
53686  | kiwi       | 3          | 0.5
57181  | strawberry | 101        | 0.2
25274  | apple      | 0          | 0.8
48800  | orange     | 35         | 0.6
26187  | strawberry | 25         | 0.15
57930  | grape      | 355        | 0.06
52357  | strawberry | 102        | 0.25
43566  | peach      | 17         | 0.8
```

Question 5. How many fruits did the store sell in total on that day?

[7]:
```
total_fruits_sold = sales.column('count sold').sum()
total_fruits_sold
```

[7]: 638

Question 6. What was the store's total revenue (the total price of all fruits sold) on that day?

Hint: If you're stuck, think first about how you would compute the total revenue from just the grape sales.

[8]:
```
total_revenue = sum(sales.column(2) * sales.column(3))
total_revenue
```

[8]: 106.85

Question 7. Make a new table called `remaining_inventory`. It should have the same rows and columns as `inventory`, except that the amount of fruit sold from each box should be subtracted from that box's count, so that the "count" is the amount of fruit remaining after Saturday.

[9]:
```
remaining_inventory = inventory.with_column('count',
                                            inventory.column('count')-sales.
 ↪column('count sold'))
remaining_inventory
```

[9]:
```
box ID | fruit name | count
53686  | kiwi       | 42
57181  | strawberry | 22
25274  | apple      | 20
48800  | orange     | 0
26187  | strawberry | 230
57930  | grape      | 162
52357  | strawberry | 0
43566  | peach      | 23
```

3 Manipulating Tables & Visualization

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

[10]:
```
unemployment = Table.read_table('unemployment.csv')
unemployment
```

[10]:
```
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. Sort the data in decreasing order by NEI, naming the sorted table `by_nei`. Create another table called `by_nei_pter` that's sorted in decreasing order by NEI-PTER instead.

[11]:
```
by_nei = unemployment.sort('NEI',descending=True)
by_nei_pter = unemployment.sort('NEI-PTER',descending=True)
```

Question 3. Use `take` to make a table containing the data for the 10 quarters when NEI was greatest. Call that table `greatest_nei`.

[12]:
```
greatest_nei = by_nei.take(np.arange(10))
greatest_nei
```

[12]:
```
Date       | NEI     | NEI-PTER
2009-10-01 | 10.9698 | 12.8557
2010-01-01 | 10.9054 | 12.7311
2009-07-01 | 10.8089 | 12.7404
2009-04-01 | 10.7082 | 12.5497
2010-04-01 | 10.6597 | 12.5664
2010-10-01 | 10.5856 | 12.4329
2010-07-01 | 10.5521 | 12.3897
2011-01-01 | 10.5024 | 12.3017
2011-07-01 | 10.4856 | 12.2507
2011-04-01 | 10.4409 | 12.247
```

Question 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.

[14]:
```
pter = unemployment.column('NEI-PTER') - unemployment.column('NEI')
pter
```

[14]:
```
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 5. Add `pter` as a column to `unemployment` (named "PTER") and sort the resulting table by that column in decreasing order. Call the table `by_pter`.

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

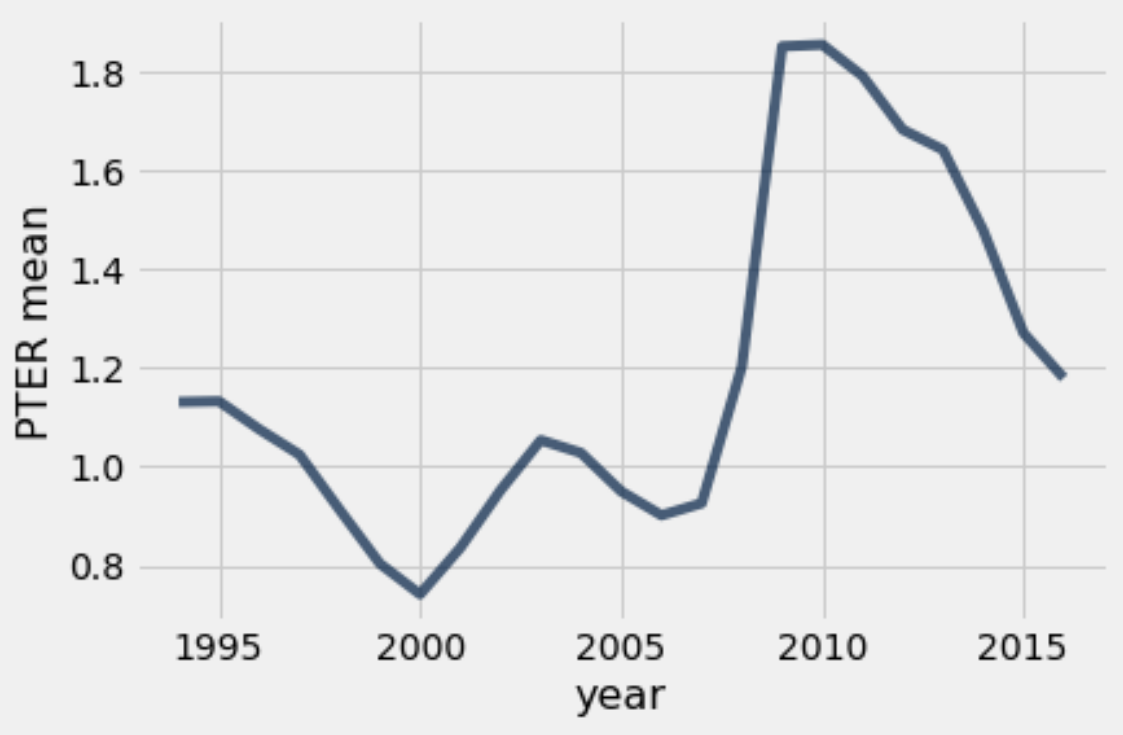
[16]:
```
by_pter = unemployment.with_column('PTER',pter).sort('PTER',descending=True)
by_pter
```

[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 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.

[24]:
```
# This was missing some instructions about how you would group by date. Below␣
↪is an example of this,
# but I won't take any points off if you didn't figure out how to plot by year␣
↪and just plotted by date.
```

```
year = np.array([int(d[:4]) for d in by_pter.column('Date')])
by_pter.with_column('year',year).group('year',np.mean).plot('year','PTER mean')
```

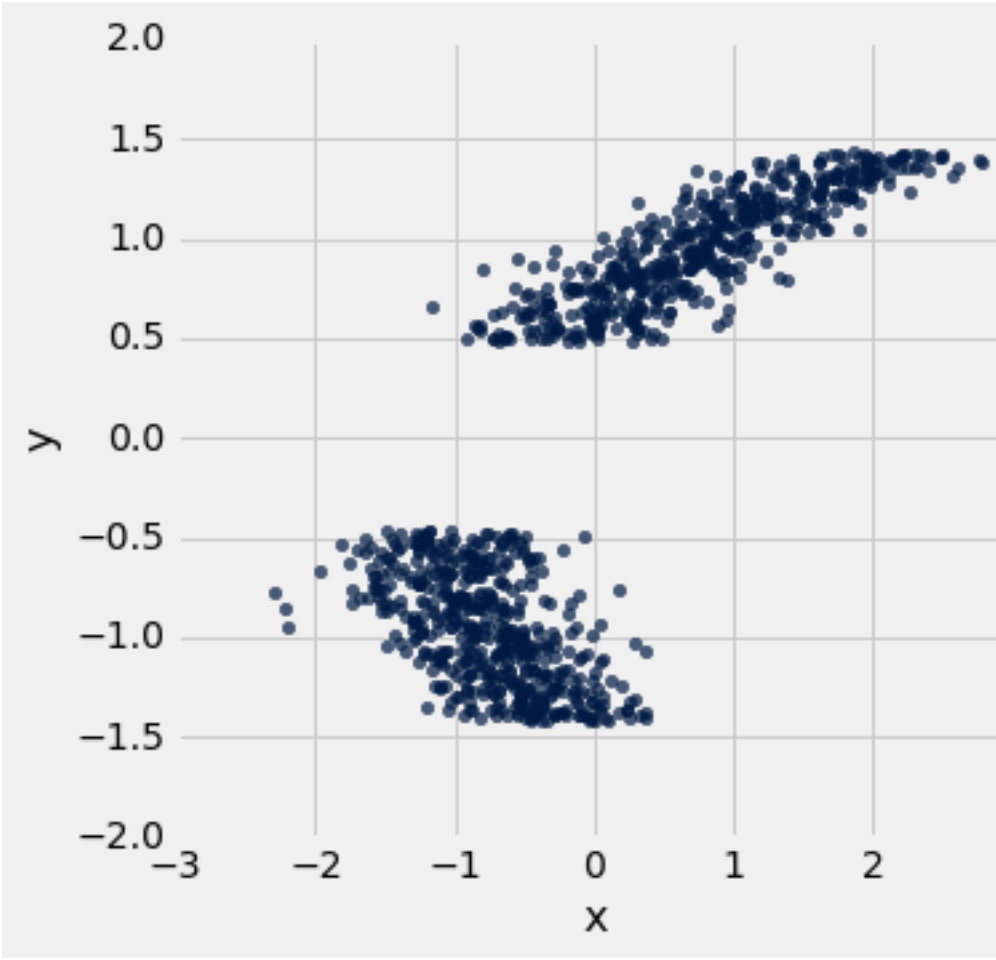


Question 7. Were PTER rates high during or directly after the Great Recession (that is to say, were PTER rates particularly high in the years 2008 through 2011 or so)? Assign highPTER to `True` if you think PTER rates were high in this period, and `False` if you think they weren't.

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

4 Marginal Histograms

Consider the following scatter plot:



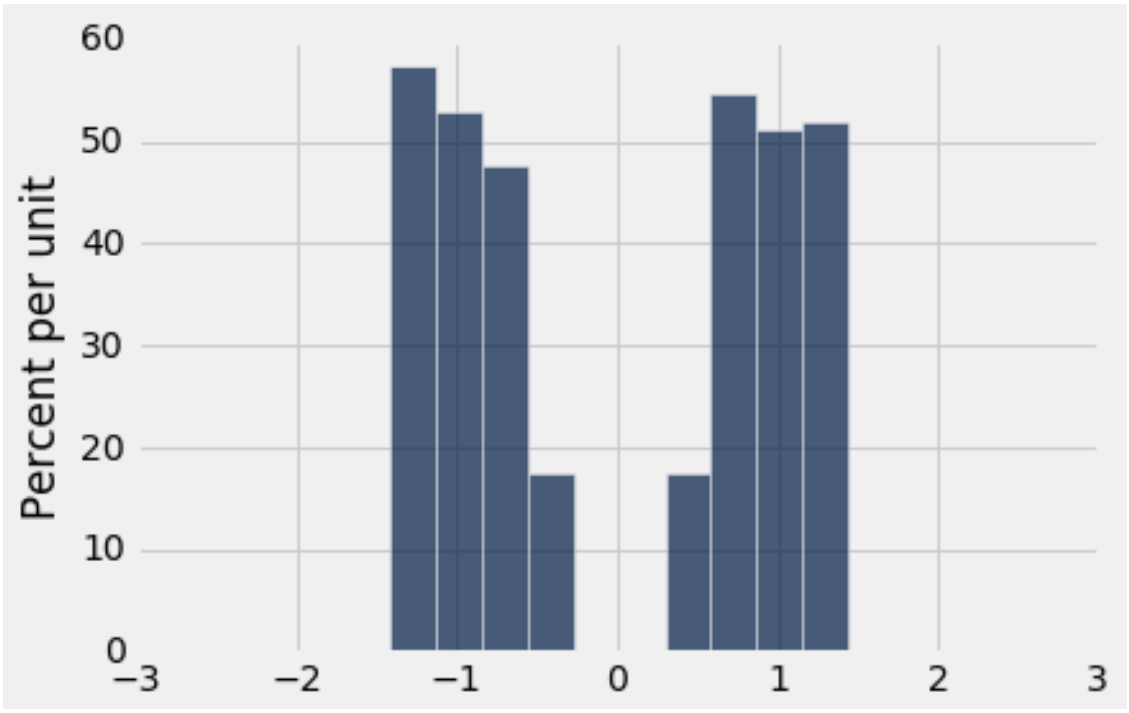
The axes of the plot represent values of two variables: x and y.

Suppose we have a table called `t` that has two columns in it:

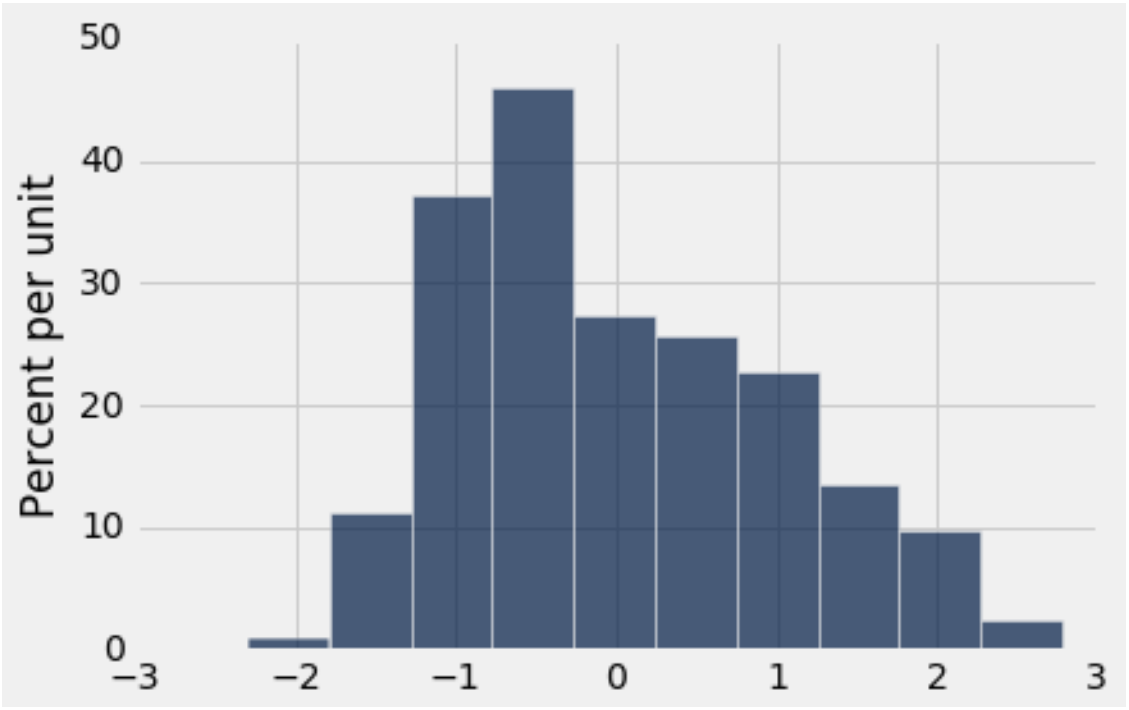
- `x`: a column containing the x-values of the points in the scatter plot
- `y`: a column containing the y-values of the points in the scatter plot

Question 1: Match each of the following histograms to the code that produced them. Explain your reasoning.

Histogram A:



Histogram B:



Line 1: `t.hist('x')`

Histogram for Line 1: Histogram B

Explanation: The values of X are spread across the values from around -2 to 3 without any "gaps" in the distribution.

Line 2: `t.hist('y')`

Histogram for Line 2: Histogram A

Explanation: If you were to flatten all of the data down onto the y-axis ('shift every point to the same x-value'), there would be a gap around 0, as seen in histogram A.

5 Submission

Download this IPython notebook and upload it to your git repository. Instructions for this can be found here.