hw03

February 20, 2019

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, 2/14 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. You will receive an early submission bonus point if you turn in your final submission by Wednesday, 2/13 at 11:59pm. Late work will not be accepted as per the policies of this course.

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. Moreover, please be sure to only put your written answers in the provided cells.

```
In [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')

        from client.api.notebook import Notebook
        ok = Notebook('hw03.ok')
```

```
=====================================================================
Assignment: Homework 3: Table Manipulation and Visualization
OK, version v1.12.5
=====================================================================
```

Before continuing the assignment, select "Save and Checkpoint" in the File menu and then execute the submit cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to okpy.org and flag the correct version. There will be another submit cell at the end of the assignment when you finish!

```
In [3]: _ = ok.submit()
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


Saving notebook... Saved 'hw03.ipynb'.
Submit... 100% complete
Submission successful for user: kayceepham@berkeley.edu
URL: https://okpy.org/cal/data8/sp19/hw03/submissions/NLxzWL
```

1.1 1. 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 1. The data are in a CSV file called `unemployment.csv`. Load that file into a table called `unemployment`.

```
In [4]: unemployment = Table.read_table("unemployment.csv")
        unemployment
```

```
Out[4]: 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)
```

```
In [5]: _ = ok.grade('q1_1')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 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.

```
In [6]: by_nei = unemployment.sort("NEI", descending = True)
        by_nei_pter = unemployment.sort("NEI-PTER", descending = True)
```

```
In [7]: by_nei
```

```
Out[7]: 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
        ... (80 rows omitted)
```

```
In [8]: _ = ok.grade('q1_2')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
```

```
[ooooooooook] 100.0% passed
```

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

`greatest_nei` should be sorted in descending order of `NEI` starting from the greatest value. Additionally, each row of `unemployment` represents a quarter.

```
In [9]: greatest_nei = by_nei.take((np.arange(0,9)))
        greatest_nei
```

```
Out[9]: 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
```

```
In [10]: _ = ok.grade('q1_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

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.

```
In [11]: pter = unemployment.column(2) - unemployment.column(1)
         pter
```

```
Out[11]: 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])
```

```
In [12]: _ = ok.grade('q1_4')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 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.

```
In [13]: by_pter = unemployment.with_column("PTER", pter).sort("PTER", descending = True)
         by_pter
```

```
Out[13]: 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)
```

```
In [14]: _ = ok.grade('q1_5')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
```

```
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

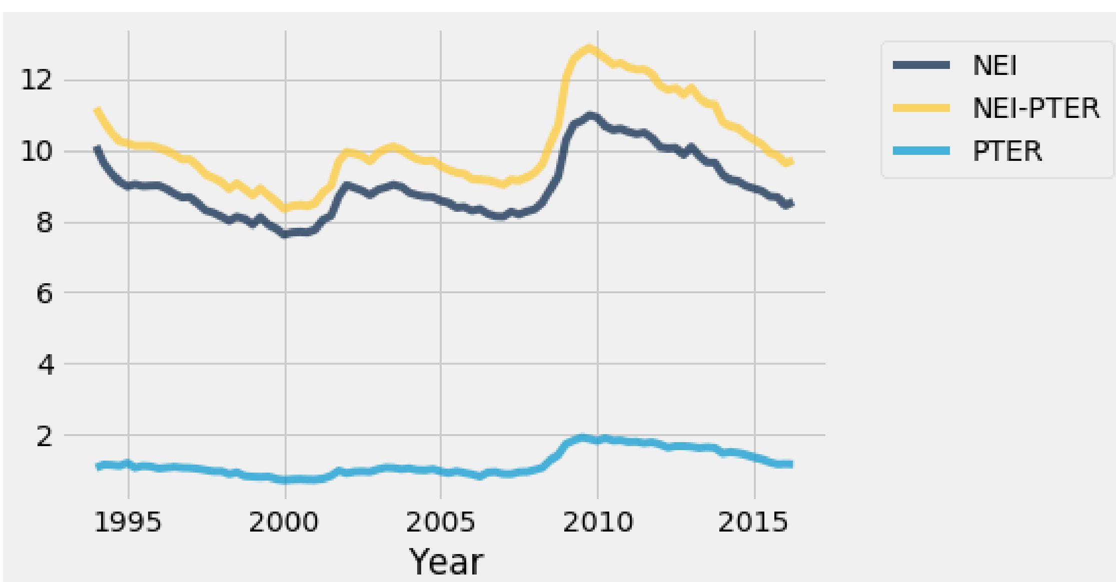
Question 6. Create a line plot of the PTER over time. To do this, first add the `year` array and the `pter` array to the `unemployment` table; label these columns `Year` and `PTER`, respectively. Then, generate a line plot using one of the table methods you've learned in class. Assign this new table to `pter_over_time`.

Note: Make sure that in your `pter_over_time` table, the `Year` column comes before the `PTER` column.

In [15]:
```
year = 1994 + np.arange(by_pter.num_rows)/4
pter_over_time = unemployment.with_column("Year",year,"PTER",pter)
pter_over_time.show
```

Out[15]:
```
<bound method Table.show of Date       | NEI     | NEI-PTER | Year    | PTER
1994-01-01 | 10.0974 | 11.172   | 1994    | 1.0746
1994-04-01 | 9.6239  | 10.7883  | 1994.25 | 1.1644
1994-07-01 | 9.3276  | 10.4831  | 1994.5  | 1.1555
1994-10-01 | 9.1071  | 10.2361  | 1994.75 | 1.129
1995-01-01 | 8.9693  | 10.1832  | 1995    | 1.2139
1995-04-01 | 9.0314  | 10.1071  | 1995.25 | 1.0757
1995-07-01 | 8.9802  | 10.1084  | 1995.5  | 1.1282
1995-10-01 | 8.9932  | 10.1046  | 1995.75 | 1.1114
1996-01-01 | 9.0002  | 10.0531  | 1996    | 1.0529
1996-04-01 | 8.9038  | 9.9782   | 1996.25 | 1.0744
... (80 rows omitted)>
```

In [16]:
```
pter_over_time.drop("Date").plot("Year")
```



```
In [17]: _ = ok.grade('q1_6')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

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)? Assign highPTER to `True` if you think PTER rates were high in this period, and `False` if you think they weren't.

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

```
In [19]: _ = ok.grade('q1_7')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

1.2 2. 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, click here!

```
In [20]: # Don't change this cell; just run it.
         pop = Table.read_table('nst-est2016-alldata.csv').where('SUMLEV', 40).select([1, 4, 1
         pop = pop.relabeled('POPESTIMATE2015', '2015').relabeled('POPESTIMATE2016', '2016')
         pop = pop.relabeled('BIRTHS2016', 'BIRTHS').relabeled('DEATHS2016', 'DEATHS')
         pop = pop.relabeled('NETMIG2016', 'MIGRATION').relabeled('RESIDUAL2016', 'OTHER')
         pop.set_format([2, 3, 4, 5, 6, 7], NumberFormatter(decimals=0)).show(5)
```

```
<IPython.core.display.HTML object>
```

Question 1. Assign `us_birth_rate` to the total US annual birth rate during this time interval. The annual birth rate for a year-long period is the total number of births in that period as a proportion of the population size at the start of the time period.

Hint: Which year corresponds to the start of the time period?

```
In [21]: state = pop.column("NAME")

In [22]: us_birth_rate = sum(pop.column("BIRTHS") / pop.column("2015")) / len(pop.column("NAME"
         us_birth_rate

Out[22]: 0.01232821678238599

In [23]: _ = ok.grade('q2_1')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 2. Assign `movers` to the number of states for which the **absolute value** of the **annual rate of migration** was higher than 1%. The annual rate of migration for a year-long period is the net number of migrations (in and out) as a proportion of the population size at the start of the period. The `MIGRATION` column contains estimated annual net migration counts by state.

```
In [24]: migration_rates = abs(pop.column("MIGRATION") / pop.column("2015"))
         pop_migration_rates = pop.with_column("MigRATE", migration_rates).where("MigRATE", are
         movers = len(pop_migration_rates.column("NAME"))
         movers

Out[24]: 9

In [25]: _ = ok.grade('q2_2')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 3. Assign `west_births` to the total number of births that occurred in region 4 (the Western US).

Hint: Make sure you double check the type of the values in the region column.

```
In [26]: west_births_array = pop.where("REGION", "4").column("BIRTHS")
         west_births = sum(west_births_array)
         west_births

Out[26]: 979657

In [27]: _ = ok.grade('q2_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 4. Assign `less_than_west_births` to the number of states that had a total population in 2016 that was smaller than the *total number of births in region 4 (the Western US)* during this time interval.

```
In [28]: less_than_west_births = len(pop.where("2016", are.below(west_births)).column("NAME"))
         less_than_west_births

Out[28]: 7

In [29]: _ = ok.grade('q2_4')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 5.

In the next question, you will be creating a visualization to understand the relationship between birth and death rates. The annual death rate for a year-long period is the total number of deaths in that period as a proportion of the population size at the start of the time period.

What visualization is most appropriate to see if there is an association between birth and death rates during a given time interval?

1. Line Graph
2. Scatter Plot
3. Bar Chart

Assign `visualization` below to the number corresponding to the correct visualization.

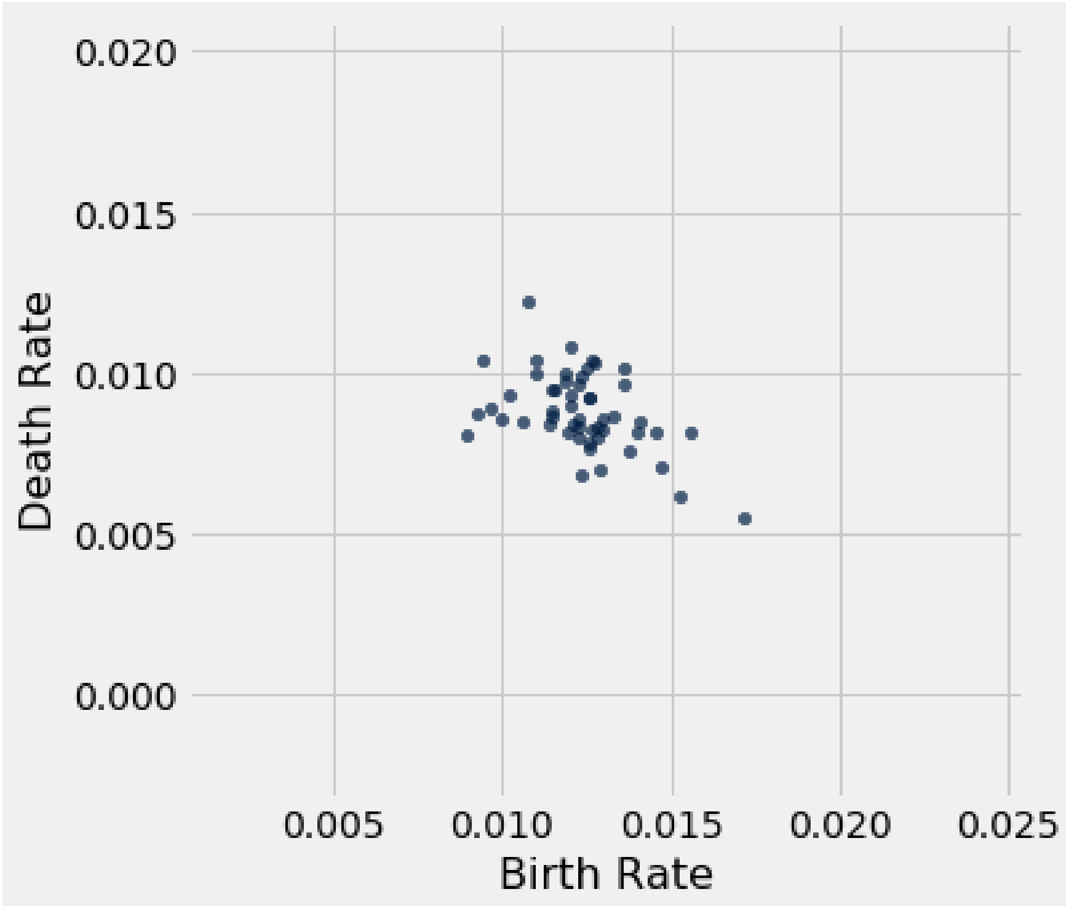
```
In [30]: visualization = 2

In [31]: _ = ok.grade('q2_5')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 6.

In the code cell below, create a visualization that will help us determine if there is an association between birth rate and death rate during this time interval. It may be helpful to create an intermediate table here.

```
In [32]: # Generate your chart in this cell
         birth_rate_array = pop.column("BIRTHS") / pop.column("2015")
         death_rate_array = pop.column("DEATHS") / pop.column("2015")
         main = Table().with_column("Birth Rate",birth_rate_array,"Death Rate",death_rate_array
         main.scatter("Birth Rate", "Death Rate")
```



Question 7. `True` or `False`: There is an association between birth rate and death rate during this time interval.

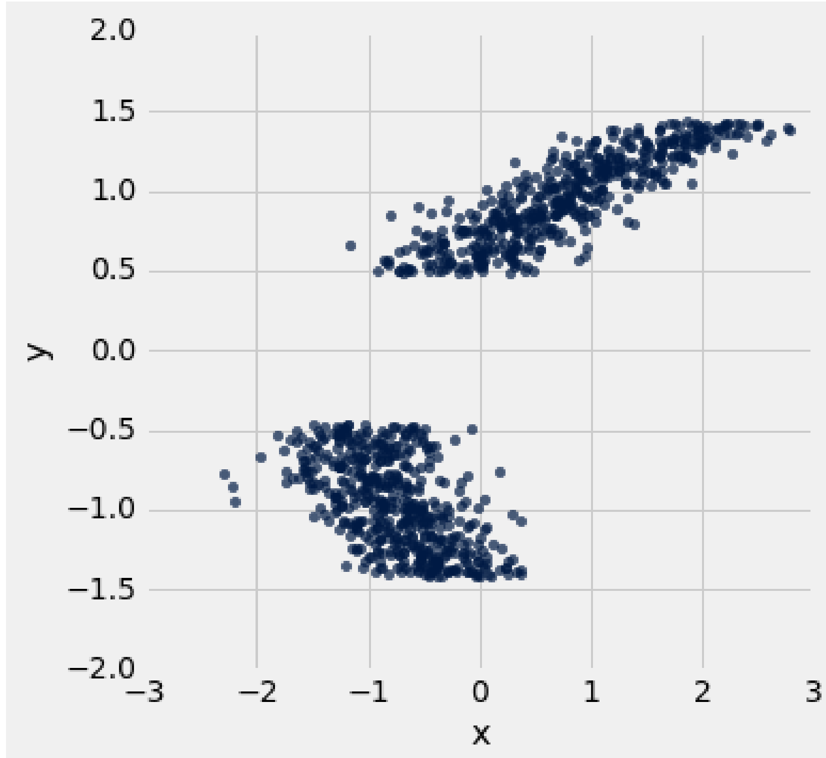
Assign `assoc` to `True` or `False` in the cell below.

```
In [33]: assoc = True
```

```
In [34]: _ = ok.grade('q2_7')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

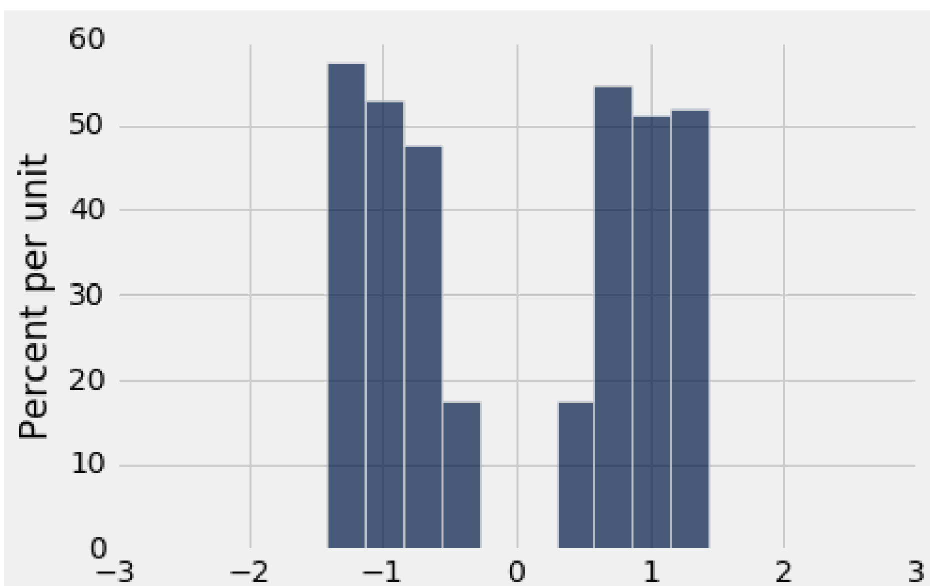
1.3 3. Marginal Histograms

Consider the following scatter plot: 

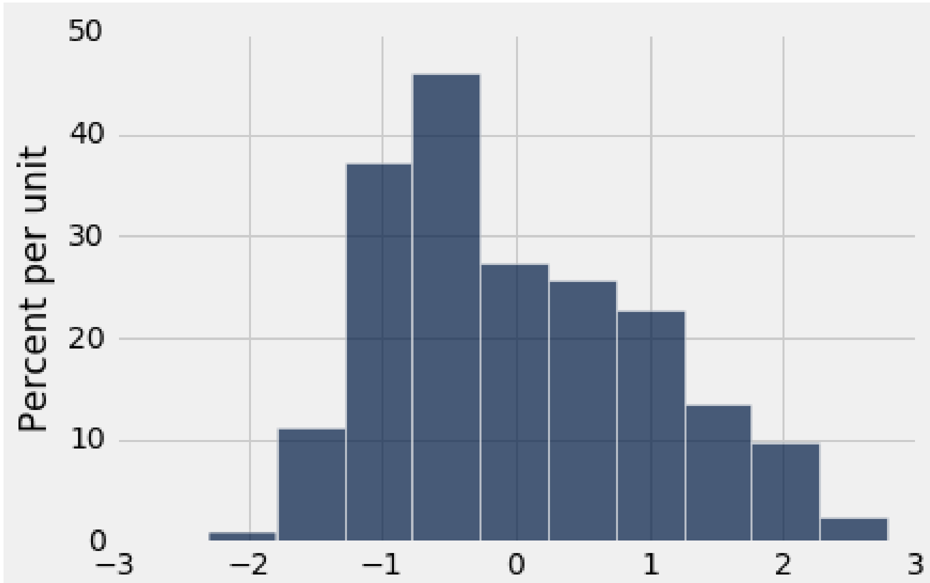
The axes of the plot represent values of two variables: x and y.
Suppose we have a table called `t` that has two columns in it:

- `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: 

His-

togram B: 

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

Histogram for Line 1: B

Explanation: This is because in the graph, x is most dense between -1 and 0, and the Histogram accurately depicts that. X has values spanning from -2.5 all the way to 3, without large gaps.

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

Histogram for Line 2: A

Explanation: In the graph, there are no plots for when y is in between -0.5 to 0.5, this the histogram is empty during that period. Y values only exist strictly in between -1.5 to -0.5 and 0.5 to 1.5.

1.4 4. Uber

Below we load tables containing 200,000 weekday Uber rides in the Manila, Philippines, and Boston, Massachusetts metropolitan areas from the Uber Movement project. The `sourceid` and `dstid` columns contain codes corresponding to start and end locations of each ride. The `hod` column contains codes corresponding to the hour of the day the ride took place. The `ride time` table contains the length of the ride, in minutes.

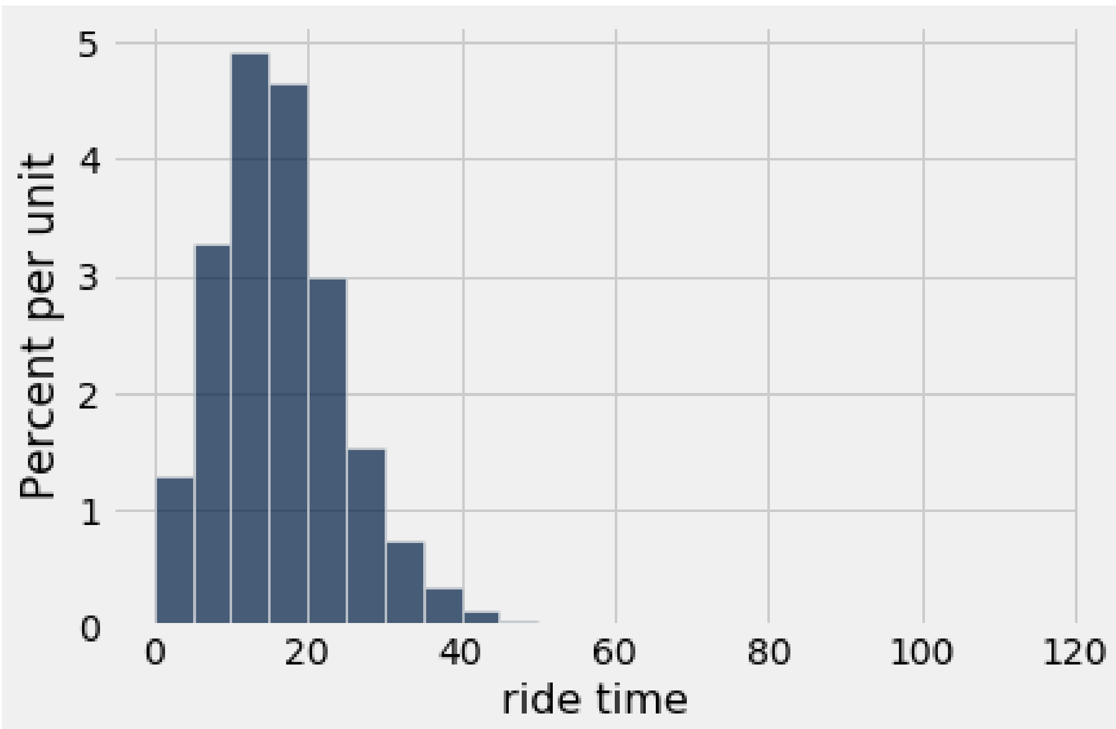
```
In [35]: boston = Table.read_table("boston.csv")
         manila = Table.read_table("manila.csv")
         print("Boston Table")
         boston.show(4)
         print("Manila Table")
         manila.show(4)
```

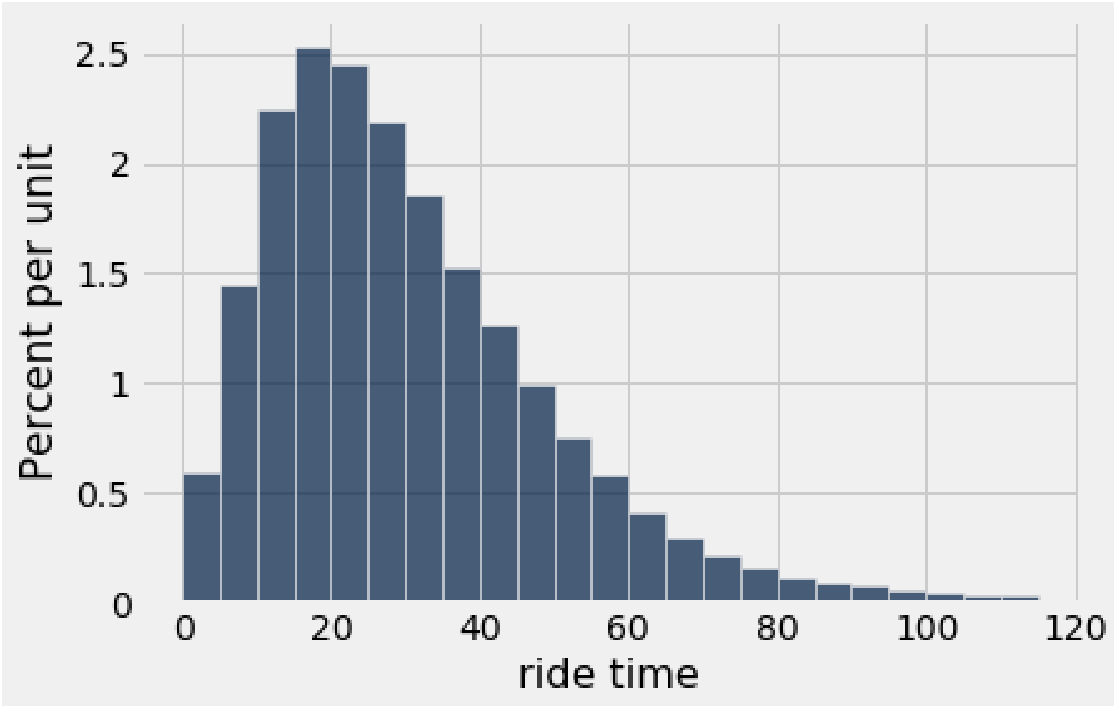
```
Boston Table


<IPython.core.display.HTML object>


Manila Table
```

<IPython.core.display.HTML object>

Question 1. Produce histograms of all ride times in Boston and in Manila, using the given bins. Please put the code for both of them in the following cells, and put the ride times for Boston first.

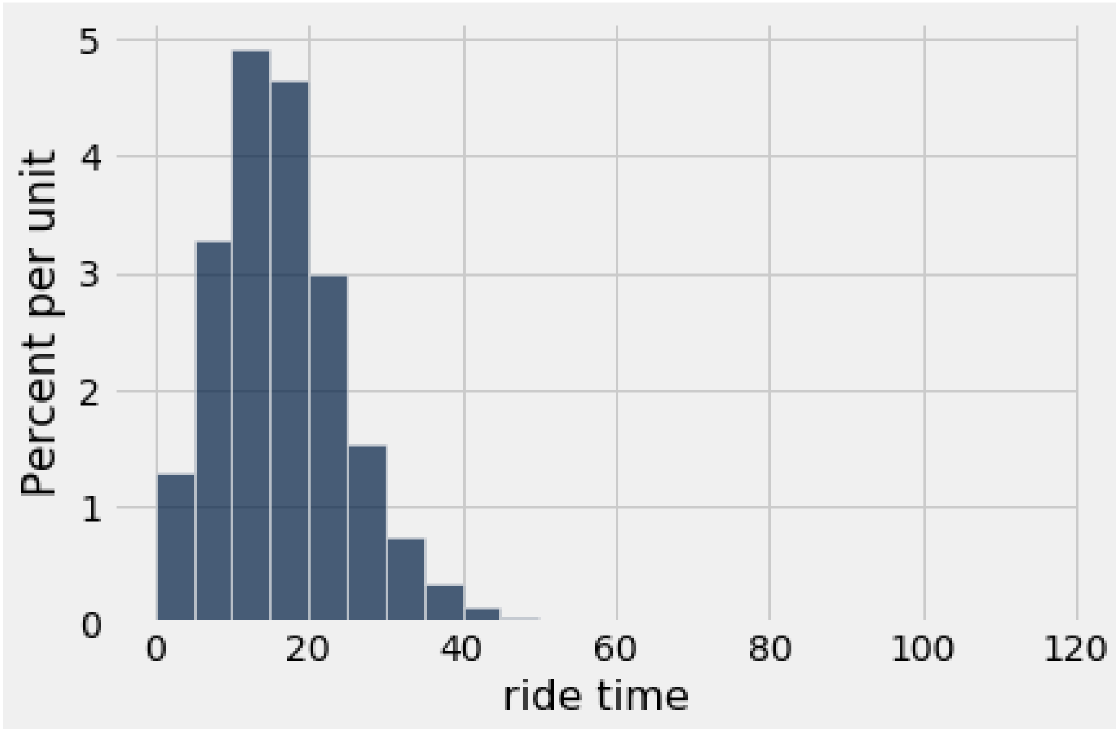
In [36]:
```
equal_bins = np.arange(0, 120, 5)
boston.hist("ride time", bins=equal_bins)
manila.hist("ride time", bins=equal_bins)
```

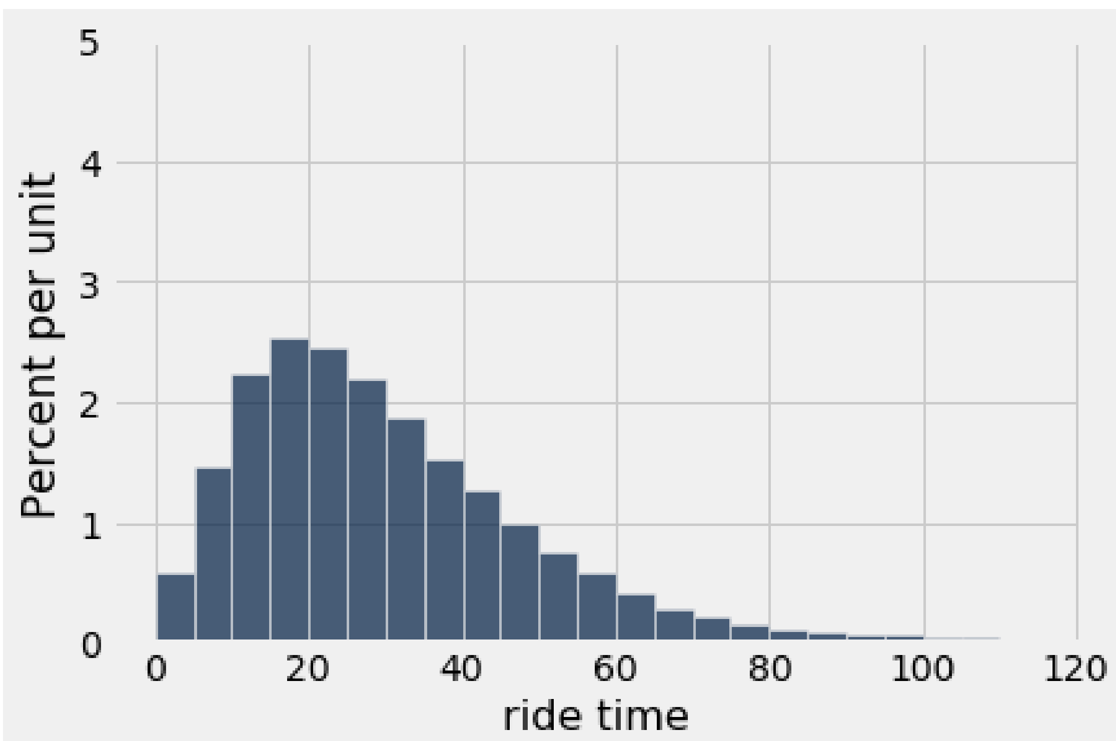




In [37]:
```
boston.hist("ride time", bins=equal_bins)
manila.hist("ride time", bins=equal_bins)

# Don't delete the following line!
plots.ylim(0, 0.05)
```

Out[37]:
```
(0, 0.05)
```


Percent per unit
ride time
0
1
2
3
4
5
0
20
40
60
80
100
120


Percent per unit
ride time
0
1
2
3
4
5
0
20
40
60
80
100
120

Question 2. Assign `boston_under_10` and `manila_under_10` to the percentage of rides that are less than 10 minutes in their respective metropolitan areas. Use the height variables provided below in order to compute the percentages. Your solution should consist of only mathematical operations and numbers.

```
In [38]: boston_under_5_height = 1.2
         manila_under_5_height = 0.6
         boston_5_to_under_10_height = 3.2
         manila_5_to_under_10_height = 1.4

         boston_under_10 = ((1.2 * 5) + (3.2 * 5))
         manila_under_10 = ((0.6 * 5) + (1.4 * 5))
```

```
In [39]: _ = ok.grade('q4_2')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 3. Let's take a closer look at the distribution of ride times in Manila. Assign `manila_median_bucket` to 1, 2, 3, or 4, where each number corresponds to a bucket which contains the median time.

1: 0-20 minutes
2: 20-40 minutes
3: 40-60 minutes
4: 60-80 minutes

The median of a sorted, nondecreasing list has half of the list elements to its left, and half to its right.

```
In [40]: manila_median_bucket = 2
         manila_median_bucket
```

```
Out[40]: 2
```

```
In [41]: _ = ok.grade('q4_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
```

```
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 4. Comment on the main difference between the two histograms. What might be causing this?

Hint: Try thinking about external factors that may be causing the difference!

Your Answer Here: The difference is that Boston caps at a much lower ride time in comparison to Manila, which stretches up to approximately 50 minutes longer. This is probably due to the fact that Boston is a very metropolitan area with a better developed road system, versus Manila, a more rural area with less efficient streets.

1.5 5. Submission

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to okpy.org and flag the correct version. To do so, go to the website, click on this assignment, and find the version you would like to have graded. There should be an option to flag that submission for grading!

```
In [42]: _ = ok.submit()
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


Saving notebook... Saved 'hw03.ipynb'.
Submit... 100% complete
Submission successful for user: kayceepham@berkeley.edu
URL: https://okpy.org/cal/data8/sp19/hw03/submissions/OM7A6g
```

```
In [43]: # For your convenience, you can run this cell to run all the tests at once!
         import os
         print("Running all tests...")
         _ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q') and len(q) <=
         print("Finished running all tests.")
```

```
Running all tests...
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
```

```
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests
```

```
---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed

Finished running all tests.
```