hw05

March 28, 2020

1 Homework 5: Pivot Tables and Iteration

Reading: * Cross-Classifying by more than one variable * Bike Sharing * Randomness

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

Homework 5 is due Thursday, 2/20, 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.

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

[23]:
```python
# Don't change this cell; just run it.

import numpy as np
from datascience import *

# These lines do some fancy plotting magic.
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)
```

1.1 1. Causes of Death by Year

This exercise is designed to give you practice using the Table method `pivot`. Here is a link to the Python reference page in case you need a quick refresher.

We'll be looking at a dataset from the California Department of Public Health that records the cause of death, as recorded on a death certificate, for everyone who died in California from 1999 to 2013. The data is in the file `causes_of_death.csv.zip`. Each row records the number of deaths by a specific cause in one year in one ZIP code.

To make the file smaller, we've compressed it; run the next cell to unzip and load it.

[24]:
```python
!unzip -o causes_of_death.csv.zip
causes = Table.read_table('causes_of_death.csv')
```

```
causes
```

```
Archive:  causes_of_death.csv.zip
  inflating: causes_of_death.csv
```

[24]:
```
Year | ZIP Code | Cause of Death | Count | Location
1999 | 90002    | SUI            | 1     | (33.94969, -118.246213)
1999 | 90005    | HOM            | 1     | (34.058508, -118.301197)
1999 | 90006    | ALZ            | 1     | (34.049323, -118.291687)
1999 | 90007    | ALZ            | 1     | (34.029442, -118.287095)
1999 | 90009    | DIA            | 1     | (33.9452, -118.3832)
1999 | 90009    | LIV            | 1     | (33.9452, -118.3832)
1999 | 90009    | OTH            | 1     | (33.9452, -118.3832)
1999 | 90010    | STK            | 1     | (34.060633, -118.302664)
1999 | 90010    | CLD            | 1     | (34.060633, -118.302664)
1999 | 90010    | DIA            | 1     | (34.060633, -118.302664)
... (320142 rows omitted)
```

The causes of death in the data are abbreviated. We've provided a table called `abbreviations.csv` to translate the abbreviations.

[25]:
```
abbreviations = Table.read_table('abbreviations.csv')
abbreviations.show()
```

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

The dataset is missing data on certain causes of death for certain years. It looks like those causes of death are relatively rare, so for some purposes it makes sense to drop them from consideration. Of course, we'll have to keep in mind that we're no longer looking at a comprehensive report on all deaths in California.

Question 1.1. Let's clean up our data. First, filter out the HOM, HYP, and NEP rows from the table for the reasons described above. Next, join together the abbreviations table and our causes of death table so that we have a more detailed description of each disease in each row. Lastly, drop the column which contains the acronym of the disease, and rename the column with the full description 'Cause of Death'. Assign the variable `cleaned_causes` to the resulting table.

Hint: You should expect this to take more than one line. Use many lines and many intermediate tables to complete this question.

[26]:
```
filter_rows=('HOM', 'HYP', 'NEP')
for i in filter_rows:
    cleaned_causes= abbreviations.where ('Cause of Death', are.not_containing␣
 ↪(i))

cleaned_causes= causes.join('Cause of Death', cleaned_causes ,'Cause of Death')

cleaned_causes= cleaned_causes.drop ('Cause of Death').relabeled ('Cause of␣
 ↪Death (Full Description)', 'Cause of Death')
cleaned_causes
```

[26]:
```
Year | ZIP Code | Count | Location                 | Cause of Death
1999 | 90006    | 1     | (34.049323, -118.291687) | Alzheimer's Disease
1999 | 90007    | 1     | (34.029442, -118.287095) | Alzheimer's Disease
1999 | 90012    | 1     | (34.061396, -118.238479) | Alzheimer's Disease
1999 | 90015    | 1     | (34.043439, -118.271613) | Alzheimer's Disease
1999 | 90017    | 1     | (34.055864, -118.266582) | Alzheimer's Disease
1999 | 90020    | 1     | (34.066535, -118.302211) | Alzheimer's Disease
1999 | 90031    | 1     | (34.078349, -118.211279) | Alzheimer's Disease
1999 | 90033    | 1     | (34.048676, -118.208442) | Alzheimer's Disease
1999 | 90042    | 1     | (34.114527, -118.192902) | Alzheimer's Disease
1999 | 90044    | 1     | (33.955089, -118.290119) | Alzheimer's Disease
... (297274 rows omitted)
```

We're going to examine the changes in causes of death over time. To make a plot of those numbers, we need to have a table with one row per year, and the information about all the causes of death for each year.

Question 1.2. Create a table with one row for each year and a column for each kind of death, where each cell contains the number of deaths by that cause in that year. Call the table `cleaned_causes_by_year`.

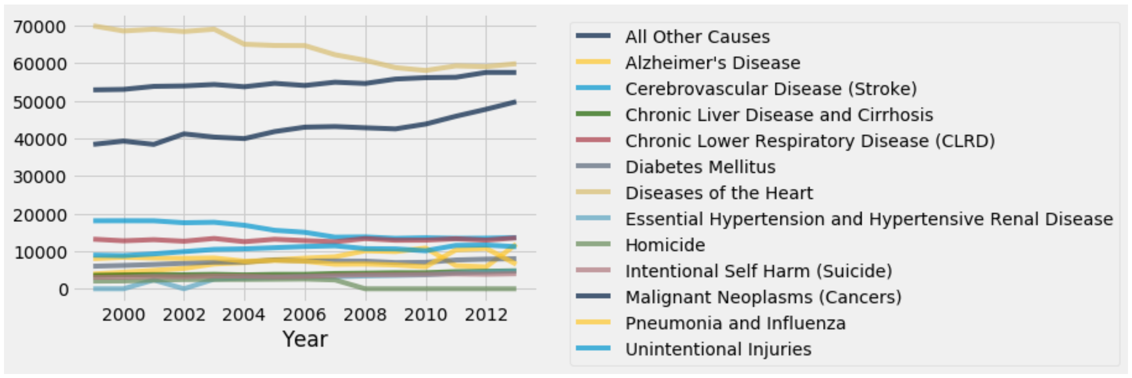
[35]:
```
cleaned_causes_by_year = cleaned_causes.pivot('Cause of Death', 'Year',␣
↪values='Count',collect= sum)
cleaned_causes_by_year.show(15)
```

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

Question 1.3. Make a plot of all the causes of death by year, using your cleaned-up version of the dataset. There should be a single plot with one line per cause of death.

Hint: Use the Table method `plot`. If you pass only a single argument, a line will be made for each of the other columns.

[37]:
```
cleaned_causes_by_year.plot('Year')
```



After seeing the plot above, we would now like to examine the distributions of diseases over the years using percentages. Below, we have assigned `distributions` to a table with all of the same columns, but the raw counts in the cells are replaced by the percentage of the the total number of deaths by a particular disease that happened in that specific year.

Try to understand the code below.

[43]:
```
def percents(array_x):
    return np.round( (array_x/sum(array_x))*100, 2)

labels = cleaned_causes_by_year.labels
distributions = Table().with_columns(labels[0], cleaned_causes_by_year.
 ↪column(0),
                                     labels[1], percents(cleaned_causes_by_year.
 ↪column(1)),
                                     labels[2], percents(cleaned_causes_by_year.
 ↪column(2)),
                                     labels[3], percents(cleaned_causes_by_year.
 ↪column(3)),
                                     labels[4], percents(cleaned_causes_by_year.
 ↪column(4)),
                                     labels[5], percents(cleaned_causes_by_year.
 ↪column(5)),
                                     labels[6], percents(cleaned_causes_by_year.
 ↪column(6)),
                                     labels[7], percents(cleaned_causes_by_year.
 ↪column(7)),
                                     labels[8], percents(cleaned_causes_by_year.
 ↪column(8)),
                                     labels[9], percents(cleaned_causes_by_year.
 ↪column(9)),
                                     labels[10],␣
 ↪percents(cleaned_causes_by_year.column(10)),
                                     labels[11],␣
 ↪percents(cleaned_causes_by_year.column(11)))
distributions.show()
```

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

Question 1.4. What is the sum (roughly) of each of the columns (excluding the Year column) in the table above? Why does this make sense?

They all roughly add up to 100 because each cause of death over the period represents 100 percent of the deaths since the total value of deaths is split among the years.

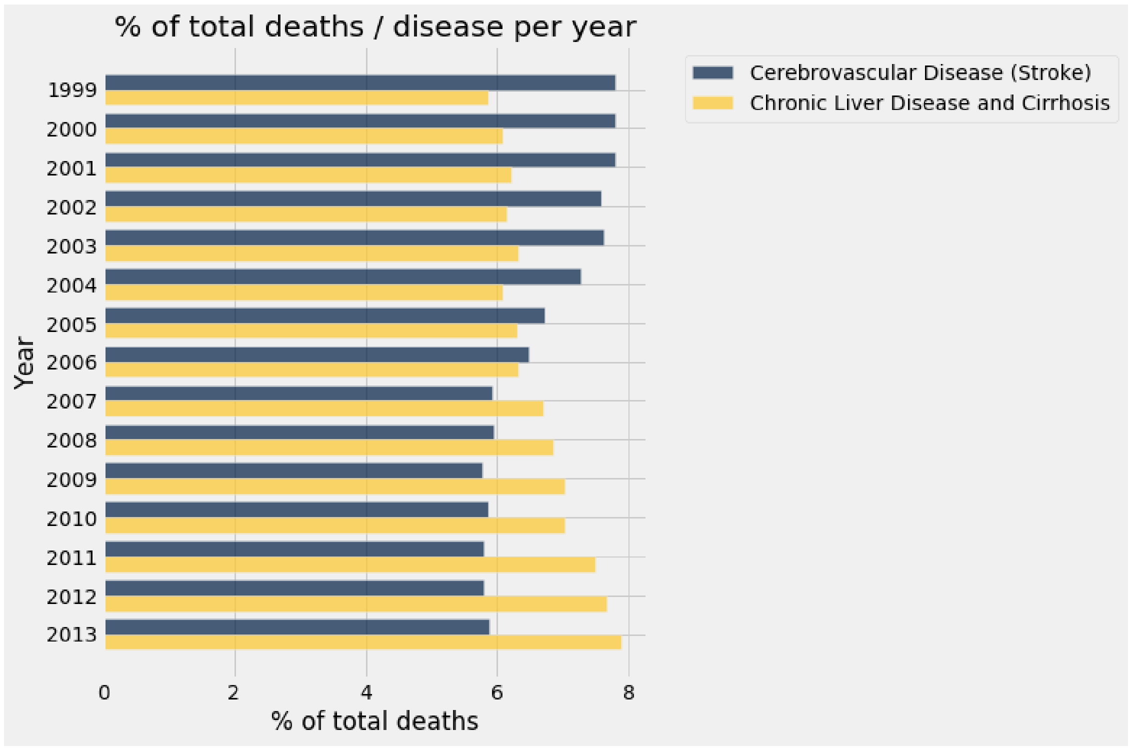
Question 1.5: We suspect that the larger percentage of stroke-related deaths over the years 1999-2013 happened in the earlier years, while the larger percentage of deaths related to Chronic Liver Disease over this time period occured in the most recent years. Draw a bar chart to display both of the distributions of these diseases over the time period.

Hint: The relevant column labels are "Cerebrovascular Disease (Stroke)" and "Chronic Liver Disease and Cirrhosis"

[47]:
```
distributions.select ('Year','Cerebrovascular Disease (Stroke)', 'Chronic Liver␣
 ↪Disease and Cirrhosis'). barh ('Year')

# Don't change the code below this comment.
plt.title("% of total deaths / disease per year")
plt.xlabel("% of total deaths")
```

[47]:
```
Text(0.5, 0, '% of total deaths')
```



1.2 (Optional) Unrolling Loops

The rest of this homework is optional. Do it for your own practice, but it will not be incorporated into the final grading!

"Unrolling" a `for` loop means to manually write out all the code that it executes. The result is code that does the same thing as the loop, but without the structure of the loop. For example, for the following loop:

```
for num in np.arange(3):
    print("The number is", num)
```

The unrolled version would look like this:

```
print("The number is", 0)
print("The number is", 1)
print("The number is", 2)
```

Unrolling a `for` loop is a great way to understand what the loop is doing during each step. In this exercise, you'll practice unrolling `for` loops.

In each question below, write code that does the same thing as the given code, but with any `for` loops unrolled. It's a good idea to run both your answer and the original code to verify that they do the same thing. (Of course, if the code does something random, you'll get a different random outcome than the original code!)

First, run the cell below to load data that will be used in a few questions. It's a table with 52 rows, one for each type of card in a deck of playing cards. A playing card has a "suit" ("", "", "", or "") and a "rank" (2 through 10, J, Q, K, or A). There are 4 suits and 13 ranks, so there are $4 \times 13 = 52$ different cards.

[ ]:
```
deck = Table.read_table("deck.csv")
deck
```

Optional Question 1. Unroll the code below.

[ ]:
```
# This table will hold the cards in a randomly-drawn hand of
# 5 cards.  We simulate cards being drawn as follows: We draw
# a card at random from the deck, make a copy of it, put the
# copy in our hand, and put the card back in the deck.  That
# means we might draw the same card multiple times, which is
# different from a normal draw in most card games.
hand = Table().with_columns("Rank", make_array(), "Suit", make_array())
for suit in np.arange(5):
    card = deck.row(np.random.randint(deck.num_rows))
    hand = hand.with_row(card)
hand
```

[ ]:
```
hand = Table().with_columns("Rank", make_array(), "Suit", make_array())
...
```

Optional Question 2. Unroll the code below.

[ ]:
```
for joke_iteration in np.arange(4):
    print("Knock, knock.")
    print("Who's there?")
    print("Banana.")
    print("Banana who?")
print("Knock, knock.")
print("Who's there?")
print("Orange.")
print("Orange who?")
print("Orange you glad I didn't say banana?")
```

[ ]:
```
...
```

Optional Question 3. Unroll the code below.

Hint: `np.random.randint` returns a random integer between 0 (inclusive) and the value that's passed in (exclusive).

[ ]:
```
# This table will hold the cards in a randomly-drawn hand of
# 4 cards.  The cards are drawn as follows: For each of the
# 4 suits, we draw a random card of that suit and put it into
# our hand.  The cards within a suit are drawn uniformly at
# random, meaning each card of the suit has an equal chance of
# being drawn.
hand_of_4 = Table().with_columns("Rank", make_array(), "Suit", make_array())
for suit in make_array("", "", "", ""):
    cards_of_suit = deck.where("Suit", are.equal_to(suit))
    card = cards_of_suit.row(np.random.randint(cards_of_suit.num_rows))
    hand_of_4 = hand_of_4.with_row(card)
hand_of_4
```

[ ]:
```
...
```

1.3 3. Submission

Once you're finished, submit your assignment as a .ipynb (Jupyter Notebook) and .pdf (download as .html, then print to save as a .pdf) on the class Canvas site.