hw04_master

October 3, 2019

1 Homework 4: Functions, Histograms, and Groups

Reading:

- Visualizing Numerical Distributions
- Functions and Tables

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 4 is due Thursday, 9/26 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, 9/25 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.

[ ]:
```
# 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 plt
plt.style.use('fivethirtyeight')

from client.api.notebook import Notebook
ok = Notebook('hw04.ok')
```

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!

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

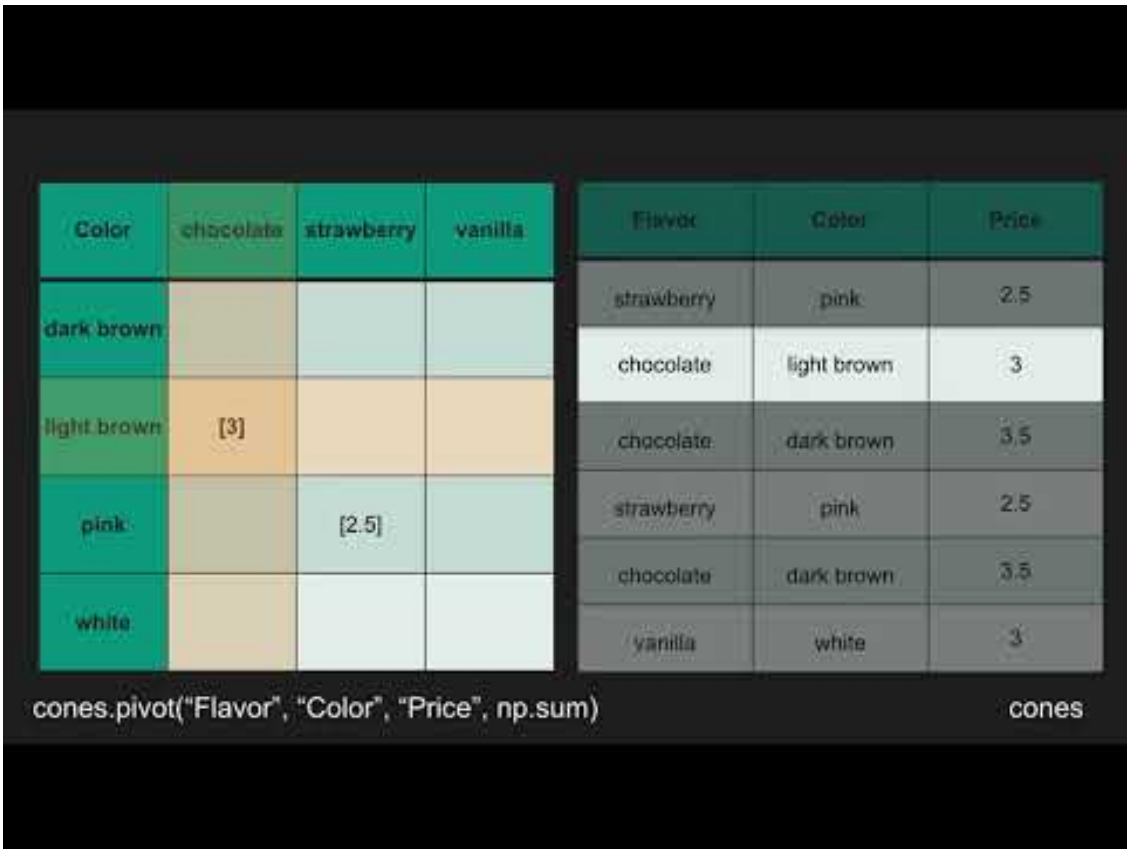
1.1 1. Causes of Death by Year

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.

Run the cell below to view a demo on how you can use pivot on a table.

[4]:
```
from IPython.display import YouTubeVideo
YouTubeVideo("4WzXo8eKLAg")
```

[4]:



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.

[5]:
```
!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
```

[5]:
```
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 decode the abbreviations.

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

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

The dataset is missing data on certain causes of death, such as homicide and hypertensive renal disease, for certain years. It looks like those causes of death are relatively rare, so for some purposes it makes sense to remove 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. Let's clean up our data. First, remove rows with HOM, HYP, and NEP as the cause of death 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 abbreviation of the disease, and rename the column that contains the full description to 'Cause of Death'. Assign the resulting table to the name cleaned_causes.

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

BEGIN QUESTION
name: q1_1

[7]:
```
""" # BEGIN PROMPT
cleaned_causes = ...
cleaned_causes
"""; # END PROMPT
# BEGIN SOLUTION NO PROMPT
no_hom_hyp_nep = causes.where('Cause of Death', are.not_equal_to("HOM"))\
                    .where('Cause of Death', are.not_equal_to("HYP"))\
                    .where('Cause of Death', are.not_equal_to("NEP"))
with_abbreviations = no_hom_hyp_nep.join('Cause of Death', abbreviations)
cleaned_causes = with_abbreviations.drop('Cause of Death').relabel('Cause of␣
↪Death (Full Description)', 'Cause of Death')
cleaned_causes
# END SOLUTION
```

[7]:
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

… (251538 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 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.

BEGIN QUESTION
name: q1_2

[12]:
```
cleaned_causes_by_year = cleaned_causes.pivot('Cause of Death', 'Year',␣
↪'Count', sum) #SOLUTION
cleaned_causes_by_year.show(15)
```

<IPython.core.display.HTML object>

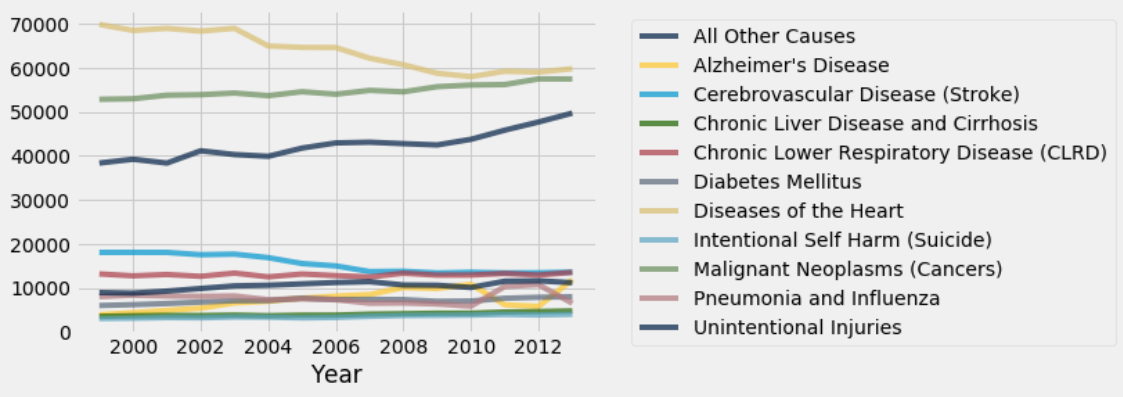
Question 3. Make a plot of all the causes of death by year, using the cleaned_causes_by_year table. There should be a single plot with one line per cause of death.

Hint: Use the Table method plot. If you pass only a single argument, a line will

be made for each of the other columns.

BEGIN QUESTION
name: q1_3
manual: true

[14]:
```
cleaned_causes_by_year.plot("Year") #SOLUTION
```



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.

[15]:
```
def percents(array_x):
    return np.round( (array_x/sum(array_x))*100, 2)

# We are making the labels an array; you are not expected to know the asarray␣
 ↪function.
labels = np.asarray(cleaned_causes_by_year.labels)
distributions = Table().with_columns(labels.item(0), cleaned_causes_by_year.
 ↪column(0),
                                     labels.item(1),␣
 ↪percents(cleaned_causes_by_year.column(1)),
                                     labels.item(2),␣
 ↪percents(cleaned_causes_by_year.column(2)),
                                     labels.item(3),␣
 ↪percents(cleaned_causes_by_year.column(3)),
                                     labels.item(4),␣
 ↪percents(cleaned_causes_by_year.column(4)),
                                     labels.item(5),␣
 ↪percents(cleaned_causes_by_year.column(5)),
```

```
                                    labels.item(6),␣
↪percents(cleaned_causes_by_year.column(6)),
                                    labels.item(7),␣
↪percents(cleaned_causes_by_year.column(7)),
                                    labels.item(8),␣
↪percents(cleaned_causes_by_year.column(8)),
                                    labels.item(9),␣
↪percents(cleaned_causes_by_year.column(9)),
                                    labels.item(10),␣
↪percents(cleaned_causes_by_year.column(10)),
                                    labels.item(11),␣
↪percents(cleaned_causes_by_year.column(11)))
distributions.show()
```

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

Question 4. What is the sum (roughly) of each of the columns (except the Year column) in the table above? Why does this make sense?

BEGIN QUESTION
name: q1_4
manual: true

SOLUTION: The columns should sum up approximately to 100 percent. This makes sense because each column is its own distribution; for each disease, each cell represents what percentage of the deaths in the years 1999-2013 happened in the corresponding year. Together, for a disease, 100 percent of the deaths occur in this period, so everything should add up to 100.

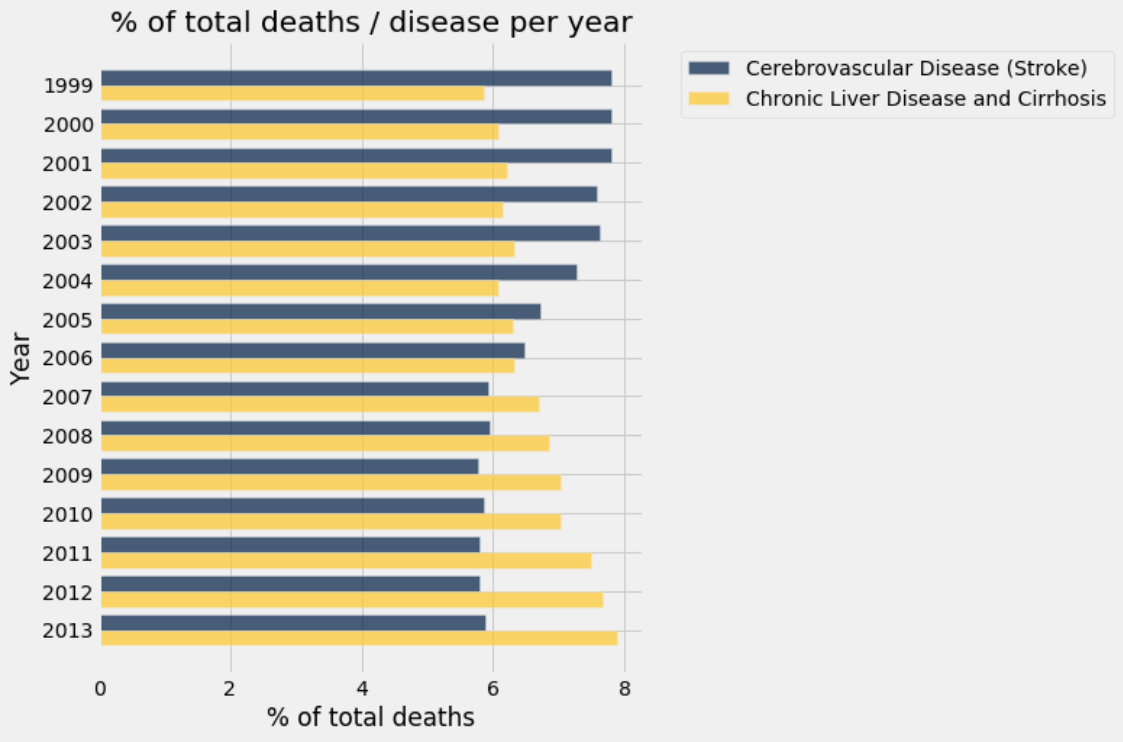
Question 5: Over the years 1999-2013, we suspect that most stroke-related deaths happened in earlier years, while most Chronic Liver Disease-related deaths occured in more recent years. Draw a horizontal bar chart to display the percent of total deaths related to "Cerebrovascular Disease (Stroke)" and "Chronic Liver Disease and Cirrhosis" over the time period.

Hint: Use the Table method barh. If you pass through a single column label, it creates bar charts of the other columns.

BEGIN QUESTION
name: q1_5
manual: true

[16]:
```
distributions.select("Year", "Cerebrovascular Disease (Stroke)", "Chronic Liver␣
↪Disease and Cirrhosis").barh("Year") #SOLUTION

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



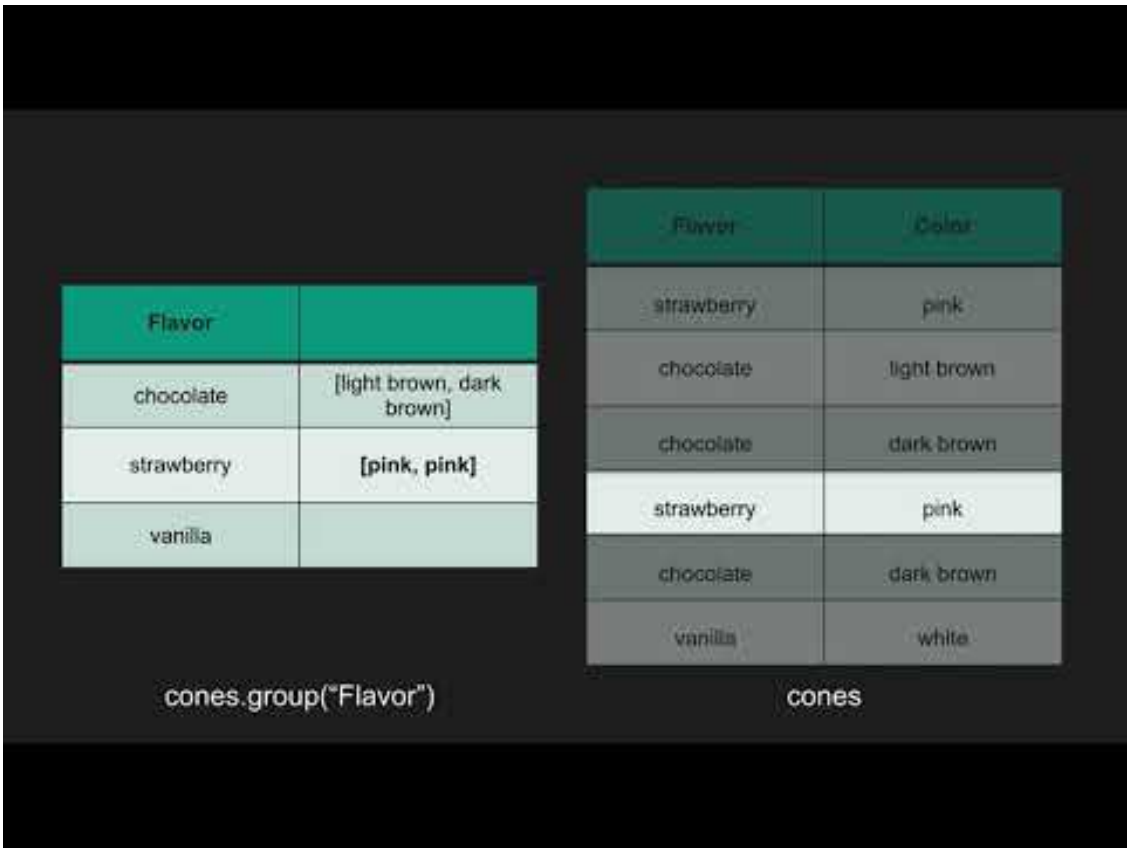
1.2 2. Faculty salaries

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

Run the cell below to view a demo on how you can use group on a table.

[17]:
```
from IPython.display import YouTubeVideo
YouTubeVideo("HLoYTCUP0fc")
```

[17]:



In the next cell, we load a dataset created by the Daily Cal which contains Berkeley faculty, their departments, their positions, and their gross salaries in 2015.

[18]:
```
raw_profs = Table.read_table("faculty.csv").where("year", are.equal_to(2015)).
 ↪drop("year", "title")
profs = raw_profs.relabeled("title_category", "position")
profs
```

[18]:
```
name            | department                      | position            |
gross_salary
CYNTHIA ABAN    | South & Southeast Asian Studies | lecturer            | 64450
PIETER ABBEEL   | Computer Science                | associate professor | 184998
SALLY ABEL      | Law                             | lecturer            | 3466
ELIZABETH ABEL  | English                         | professor           | 138775
DOR ABRAHAMSON  | Education                       | associate professor | 100300
KATHRYN ABRAMS  | Law                             | professor           | 319693
BARBARA ABRAMS  | Public Health                   | professor           | 191162
SARAH ACCOMAZZO | Social Welfare                  | lecturer            | 14779
CHARISMA ACEY   | City and Regional Planning      | assistant professor | 101567
DAVID ACKERLY   | Biology                         | professor           | 182288
```

… (2049 rows omitted)

We want to use this table to generate arrays with the names of each professor in each department.

Question 1. Set prof_names to a table with two columns. The first column should be called department and have the name of every department once, and the second column should be called faculty and contain an *array* of the names of all faculty members in that department.

Hint: Think about how group works: it collects values into an array and then applies a function to that array. We have defined two functions below for you, and you will need to use one of them in your call to group.

BEGIN QUESTION
name: q2_1

[19]:
```
# Pick one of the two functions defined below in your call to group.
def identity(array):
    '''Returns the array that is passed through'''
    return array

def first(array):
    '''Returns the first item'''
    return array.item(0)

# Make a call to group using one of the functions above when you define␣
↪prof_names
prof_names = profs.drop("position", "gross_salary").group("department",␣
↪identity).relabeled("name identity", 'faculty') # SOLUTION
prof_names
```

[19]:

department	faculty
African American Studies	['AYA DE LEON' 'CHIYUMA ELLIOTT' 'NIKKI JONES' 'DAVID KY …
Agricultural and Resource Economics and Policy	['MAXIMILIAN AUFFHAMMER' 'CHARLES GIBBONS' 'JEFFREY PERL …
Anthroplogy	['SABRINA AGARWAL' 'STANLEY BRANDES' 'CHARLES BRIGGS' ' …
Architecture	['MARK ANDERSON' 'JACOB ATHERTON' 'WILLIAM ATWOOD' 'R.GA …
Art History	['DILIANA ANGELOVA' 'PATRICIA BERGER' 'JULIA BRYAN-WILSO …
Art Practice	['ALLAN DESOUZA' 'AIDA GAMEZ' 'RANDY HUSSONG' 'JENNIFER …
Astronomy	['GIBOR BASRI' 'STEVEN BECKWITH' 'LEO BLITZ' 'EUGENE CHI …
Bioengineering	['ADAM ARKIN' 'IRINA CONBOY'

```
'STEVEN CONOLLY' 'JOHN DUEB …
Biology                                          | ['DAVID ACKERLY' 'HILLEL
ADESNIK' 'KELLY AGNEW' 'DORIS B …
Buddhist Studies                                 | ['JANN RONIS']
… (61 rows omitted)
```

Question 2. At the moment, the name column is sorted by last name. Would the arrays you generated in the previous part be the same if we had sorted by first name instead before generating them? Two arrays are the **same** if they contain the same number of elements and the elements located at corresponding indexes in the two arrays are identical. Explain your answer.

BEGIN QUESTION
name: q2_2
manual: true

SOLUTION: If the order of the names in a department changes after we sort by first name, then that will also change the array we get back. That is because group does a sequential search of the table (from top to bottom) and collects the values in the array in the order in which they appear.

Question 3. Set biggest_range_dept to the name of the department with the largest salary range, where range is defined as the **difference between the highest salary and the lowest salary in the department**.

Hint: First you'll need to define a new function salary_range which takes in an array of salaries and returns the range of salaries in that array. Then, set department_ranges to a table containing the names and salary ranges of each department.

BEGIN QUESTION
name: q2_3
manual: false

[32]:
```
# Define salary_range first
def salary_range(salaries): #SOLUTION
    return max(salaries) - min(salaries) # SOLUTION

department_ranges = profs.drop("name", "position").group("department",␣
 ↪salary_range).relabeled("gross_salary salary_range", "spread") # SOLUTION
biggest_range_dept = department_ranges.sort("spread", descending = True).
 ↪column("department").item(0) # SOLUTION
biggest_range_dept
```

[32]:
```
'Economics'
```

1.3 3. 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!

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

[ ]:
```
# 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) <= 10]
print("Finished running all tests.")
```