Lab 3: Tables

Welcome to lab 3! This week, we'll learn about *tables*, which let us work with multiple arrays of data about the same things. Tables are described in Chapter 6 of the text.

First, set up the imports by running the cell below.

In [1]:
```
import numpy as np
import pandas as pd
```

1. Introduction

For a collection of things in the world, an array is useful for describing a single attribute of each thing. For example, among the collection of US States, an array could describe the number of registered voters of each. Data frames extend this idea by describing multiple attributes for each element of a collection. You can think of an array as just one column in a spreadsheet, while a data frame is the entire spreadsheet.

Suppose we want to answer this question:

> When did world population cross 6 billion?

You could technically answer this question just from staring at the arrays, but it's a bit convoluted, since you would have to count the position where the population first crossed 6 billion, then find the corresponding element in the years array. In cases like these, it might be easier to put the data into a data frame, a 2-dimensional type of dataset, like a spreadsheet.

The expression below:

- Uses the data science library `pandas` , which is Python's most popular library for data structures and data analysis.
- Loads the spreadsheet `world_population.csv` and names it `population` .
- Displays the first five rows of `population` using `.head()` .

In [2]:
```
population = pd.read_csv("world_population.csv")
population.head()
```

Out[2]:

	Index	Position	Year	Population
0	0	Item #1	1950	2557628654
1	1	Item #2	1951	2594939877
2	2	Item #3	1952	2636772306
3	3	Item #4	1953	2682053389
4	4	Item #5	1954	2730228104

Now the data are all together in a single table! It's much easier to parse this data--if you need to know what the population was in 1959, for example, you can tell from a single glance. We'll revisit this table later.

2. Discovering Voter Files

The file `sample_voter_file.csv` contains a random sample of 100 people from the Ohio voter file. You could download the full data from https://www6.sos.state.oh.us/ords/f?p=111:1. This file contains the following fields:

- A unique identifier from the State of Ohio.
- The voter's full name.
- The voter's date of birth.
- The date the voter registered to vote.
- The voter's party registration (D = Democrat; R = Republican; missing = Independent).
- The voter's full address.
- Whether or not the voter voted in the 2012 general election (general11062012, where X = voted and missing = did not vote).
- Whether or not the voter voted in the 2014 general election (general11042014, where X = voted and missing = did not vote).

We are going to load this as a data frame called `vf`. Fix the below code to read in the file called `sample_voter_file.csv`.

In [3]:
```
vf = pd.read_csv("sample_voter_file.csv")
vf.head()
# Scroll to the right on the table to see more output.
```

Out[3]:

	sos_voterid	last_name	first_name	middle_name	suffix	date_of_birth	registration_date	p
0	OH0016289210	CROSS	DONNA	JEAN	NaN	1944-11-03	1970-09-29	
1	OH0016304706	CLARK	KENNETH	H	NaN	1954-05-27	2002-08-20	
2	OH0016289828	WELCH	VINIA	RUTH	NaN	1945-09-12	1973-10-09	
3	OH0016291257	LANG	BOBBIE	ANN	NaN	1932-11-26	1965-10-05	
4	OH0020938137	GILPIN	SCOTTY	E	NaN	1971-08-23	2010-01-05	

Analyzing datasets

With just a few lines of code, we can answer some interesting questions about the voter file.

If we want just the data on how many people voted in 2014, we can get an array that contains the data in that column. Note that `pandas` assigns the value of `NaN` to missing values. This just means that the cell of the spreadsheet has no value in it. In the case of the column `general11042014`, an individual is assigned "X" if they voted or missing otherwise.

In [4]:
```
voted_2014 = vf["general11042014"]
```

```
voted_2014
```

Out[4]:
```
0       X
1       X
2     NaN
3       X
4     NaN
     ...
95      X
96      X
97    NaN
98      X
99      X
Name: general11042014, Length: 100, dtype: object
```

Of these 100 people, how many voted in 2014? To do this, we could count the number of people who have in "X" in the above list. The below code gives one example of how we could do this.

In [5]:
```
voted_2014 = vf['general11042014'].value_counts()
voted_2014
```

Out[5]:
```
X    31
Name: general11042014, dtype: int64
```

To explain the syntax, we are counting how many records have each value in the column `general11042014` .

Question 1. How many people voted in 2012?

In [6]:
```
voted_2012 = vf['general11062012'].value_counts()
voted_2012
```

Out[6]:
```
X    57
Name: general11062012, dtype: int64
```

Having a bunch of X's and missing values isn't really helpful. Can we recode our variables so a voter receives a "1" if they voted and a "0" if they did not vote? In the below code, we are creating a new column called `voted_14` in our data frame called `vf` . We first use `np.where` to determine where `general11042014` is set as X. If a certain row is set as X, `voted_14` is given a 1. Otherwise, it is given a 0.

In [7]:
```
vf['voted_14'] = np.where(vf['general11042014'] == "X", 1, 0)
vf['voted_14'].value_counts()
```

Out[7]:
```
0    69
1    31
Name: voted_14, dtype: int64
```

Question 2. Can you create a variable called `voted_12` similar to `voted_14` ?

In [8]:
```
vf['voted_12'] = np.where(vf['general11062012'] == "X", 1, 0)
vf['voted_12'].value_counts()
```

Out[8]:
```
1    57
0    43
Name: voted_12, dtype: int64
```

Finding pieces of a dataset

Suppose you're interested in learning how voter turnout in 2014 differed by political party. We can create what is called a cross tab. A cross tab is simply a summary table of the data. In the below example, the columns summarize registered voters and the columns count how many voters of each party either did or did not vote in 2014.

In [9]:
```
pd.crosstab(vf.party_affiliation, vf.voted_12)
```

Out[9]:

voted_12	0	1
party_affiliation		
D	0	2
R	2	35

But something seems wrong about this summary table. It only contains `2 + 15 + 22 = 39` records. Doesn't our data frame have 100 records?

We are missing all the voters who are neither registered Democrats nor registered Republicans!

We can recode `party_affiliation` so missing data is set to I, for Independents. Then we can re-run our crosstab.

In [10]:
```
vf['party_affiliation'] = vf['party_affiliation'].fillna("I")
pd.crosstab(vf.party_affiliation, vf.voted_14)
```

Out[10]:

voted_14	0	1
party_affiliation		
D	0	2
I	54	7
R	15	22

Question 3. How many Democrats, Republicans, and Independents voted in 2012? Fill in the below code, replacing the ellipsis with **code** (do not manually enter the correct number, but use code to calculate this number) that counts how many of each type of voter voted in 2012?

HINT: I would suggest familiarizing yourselves with crosstabs and ".loc". See e.g., https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix. Remember, it is fine (expected!) that you will Google for help. That's how you learn to code.

In [11]:
```
vf['party_affiliation'] = vf['party_affiliation'].fillna("I")
pd.crosstab(vf.party_affiliation, vf.voted_12)
print("The number of Democrats who voted in 2012 is",  pd.crosstab(vf.party_affi
print("The number of Republicans who voted in 2012 is", pd.crosstab(vf.party_aff
print("The number of Independents who voted in 2012 is", pd.crosstab(vf.party_af
```

```
The number of Democrats who voted in 2012 is 2
The number of Republicans who voted in 2012 is 35
The number of Independents who voted in 2012 is 20
```

Other Pandas Operations

Pandas is an incredibly powerful library. You can learn a lot by reading short introductions like this excellent 10 Minutes to Pandas. Give it a read and return to it whenever you are stuck.

Here are just a few examples of what you can do with Pandas.

In [12]:

```
# Display the first 5 records using head() of every Independent who voted in 201
vf.query('voted_14 == 1 & party_affiliation == "I"').head()
```

Out[12]:

	sos_voterid	last_name	first_name	middle_name	suffix	date_of_birth	registration_date
3	OH0016291257	LANG	BOBBIE	ANN	NaN	1932-11-26	1965-10-05
42	OH0016306107	SILCOTT	CAROL	ANN	NaN	1965-08-13	1984-01-01
53	OH0024392537	DANIELS	RAYMOND	NaN	NaN	1942-05-06	2017-09-28
58	OH0016298379	SLOOP	RALPH	H	NaN	1932-07-05	1964-08-13
70	OH0016305109	KAIN	BEVERLY	ANN	NaN	1959-03-30	1983-08-31

In [13]:

```
# Rename party_affiliation to party_reg
vf = vf.rename(columns={'party_affiliation': 'party_reg'})
vf.head()
```

Out[13]:

	sos_voterid	last_name	first_name	middle_name	suffix	date_of_birth	registration_date	p
0	OH0016289210	CROSS	DONNA	JEAN	NaN	1944-11-03	1970-09-29	
1	OH0016304706	CLARK	KENNETH	H	NaN	1954-05-27	2002-08-20	
2	OH0016289828	WELCH	VINIA	RUTH	NaN	1945-09-12	1973-10-09	
3	OH0016291257	LANG	BOBBIE	ANN	NaN	1932-11-26	1965-10-05	
4	OH0020938137	GILPIN	SCOTTY	E	NaN	1971-08-23	2010-01-05	

Dates in Pandas

Dates are common in a lot of data science applications. But dates are also very annoying. As a human, we know that these are all equivalent:

- January 1, 2019
- 1 January 2019
- 1/1/2019
- 1/1/19

But even as a human, is 1/2/2019 referring to 2 January 2019 or is it 1 February 2019? What about 1/2/19? Is that referring to 2019, 1919, 1819, etc.? Dates can be a real pain!

Luckily, Pandas has a way to standardize all dates. The below code gives Pandas a particular date as a string and let's Pandas know how that string is formatted.

In [14]:
```
# Define dates as strings
date_str1 = 'Wednesday, June 6, 2018'
date_str2 = '06-06-2018'
date_str3 = '6/6/18'

# Define dates as datetime objects
date_dt1 = pd.to_datetime(date_str1, format='%A, %B %d, %Y')
date_dt2 = pd.to_datetime(date_str2, format='%m-%d-%Y')
date_dt3 = pd.to_datetime(date_str3, format='%m/%d/%y')

# Print converted dates
print(date_dt1)
print(date_dt2)
print(date_dt3)
```

```
2018-06-06 00:00:00
2018-06-06 00:00:00
2018-06-06 00:00:00
```

Question 4. Create a column called `age`.

In the first line, convert `date_of_birth` to a `datetime` object, like above. In the second line, take the difference between today's year and the year in which somebody was born to create a column called `age`. You might want to explore `.dt.year` here. You can find out how to get the current year here. In the third line, type `vf['age'].head()` to display the first five rows of this column.

In [15]:
```
vf['date_of_birth'] = pd.to_datetime(vf['date_of_birth'], format="%Y-%m-%d")
vf['age'] =  2021- vf['date_of_birth'].dt.year
vf['age'].head()
```

Out[15]:
```
0    77
1    67
2    76
3    89
4    50
Name: age, dtype: int64
```

Question 5. How many people are 50 or older?

Hint 1: See this to get the number of rows (length). *Hint 2*: Remember `query` from above!

In [16]:
```
len(vf.query('age >=50'))
```

Out[16]:
```
62
```

Question 6. What percent of people are 50 or older?

In [17]:
```
(len(vf.query('age> = 50')) / len(vf) )* (100)
```

Out[17]:
```
62.0
```

Question 7. Here's a challenge: Find the number of people who were born in *even* years.

Hint: The operator `%` computes the remainder when dividing by a number. So `5 % 2` is 1 and `6 % 2` is 0. A number is even if the remainder is 0 when you divide by 2.

Hint 2: `%` can be used on arrays, operating elementwise like `+` or `*`. So `make_array(5, 6, 7) % 2` is `array([1, 0, 1])`.

Hint 3: Create a column called "Year_Remainder" that's the remainder when each person's year of birth is divided by 2. Then create a new column called "Even_Year" that is set to 1 if "Year_Remainder" is 0 and is set to 0 if "Year_Remainder" is not 0. Remember `np.where` from above.

In [18]:
```
vf['Year_of_Birth'] = vf['date_of_birth'].dt.year
vf['Year_Remainder'] = vf['Year_of_Birth']%2
vf['even_year'] =  np.where(vf['Year_Remainder'] == 0, 1, 0)

# Do not change any of the below code.
# If anything breaks, make sure your variables are named the same as below.
# Use this .head() to check your work.
print(vf[['date_of_birth', 'Year_of_Birth', 'Year_Remainder', 'even_year']].head
# This will print your final answer.
print("The number of people born in even years is: ", len(vf.query('even_year ==
```

```
  date_of_birth  Year_of_Birth  Year_Remainder  even_year
0    1944-11-03           1944               0          1
1    1954-05-27           1954               0          1
2    1945-09-12           1945               1          0
3    1932-11-26           1932               0          1
4    1971-08-23           1971               1          0
The number of people born in even years is:  56
```

Congratulations!

You are done with the lab. Before you finish and submit, please fill out this brief evaluation:

- I spent around 3 hours on this lab,.
- This lab was just about the right difficulty.

To turn in your lab, you will need to submit a PDF through Canvas. You can download a notebook by opening it, turning Edit mode on, then navigating to File -> Download as -> PDF.

In [ ]: