IS 452 Foundations Information Processing Fall 2017

Final Project

Nanzhu Liu

UIN: 655394878

Project Type

Programming project

Project Summary

The project is to estimate demographic data related to children age 5 and under. Two types of demographic data are included: Population of children age 5 and under, and Population of children under age 5 by race. The data is updated annually, and the estimation has to be done every year. I used to estimate the data by Excel and Access. It took me a lot of time. The structure is similar every year, so the process is repetitive. I decide to write a python file to estimate data.

Dataset

Two primary sources that I use are American Community Survey (ACS) and the Population Estimates Program(PEP). There are total 6 data files.

I used 2015 American Community Survey (ACS) 5-year Estimates, which is from https://factfinder.census.gov/faces/nav/jsf/pages/searchresults.xhtml?refresh=t

Table name	What is it?
ACS_15_5YR_B01001_with_ann.csv	Population
ACS_15_5YR_B01001A_with_ann.csv	Race: White Population
ACS_15_5YR_B01001B_with_ann.csv	Race: Black Population
ACS_15_5YR_B01001C_with_ann.csv	Race: American Indian and Alaska Native Population
ACS_15_5YR_B01001D_with_ann.csv	Race: Asian Population

Tables contain “metadata” are description files.

I used 2015 Population Estimates at National Vital Statistics System (NVSS) mentioned above, which is from https://www.cdc.gov/nchs/nvss/bridged_race/data_documentation.htm

Table name: pcen_v2015_y15_txt.txt

Methodology

The goal is to have number of children under age 1, age 1, age 2, age 3, age 4, and age5 by counties in Illinois and children in race: white, black, American Indian and Alaska Native and Asian age 1, age 1, age 2, age 3, age 4, and age5 by counties in Illinois. ACS contains varieties of data sets and has large data sets. They have total number of children age 5 and under, and population of children by race, but they

don't have the number of children under age 1, age 1, age 2, age 3, age4 and age5 separately. PEP contains less kinds of data set but has number of children across all ages. To make the data consistent, I use the data from ACS and make estimation based on PEP. I download population and population by race from ACS. I also collect data from PEP because this is more precise. The estimation is shown below:

$$Pop(acs,t) = Pop(pep, t)/Pop(pep) * Pop(acs) \quad (1)$$

In equation, Pop(acs,t) is the population from ACS at time t such as age1. Pop(PEP, t) is the population from PEP at time t such as age1. Pop(pep) is the total population children age 5 and under from PEP. Pop(acs) is the total population children age 5 and under from ACS.

The structure of Python code is divided into three parts. In the first part, I read data sets from PEP into Python and create a matrix that each number is calculated by Pop(pep, t)/Pop(pep). In the second part, I read population data sets from ACS, and estimate children population across birth to age5. The third part, I read all four population data by race, and I create a function to estimate children population by race across birth to age5. Last, I merge the population and population by four races together into a large file. I will give a detailed explanation on how I write the python code.

Narratives

Set up

I import pandas as pd and import csv

Part I

I collect and prepare population data by counties from PEP. The file is .txt file. The whole txt file contains population data across country from age 0 to 85. The txt file contains two columns. The first column tells the data information including series vintage, years, month, states FIPS code, counties FIPS code, ages and so on. The second column tells numbers. I need to select data that satisfy the following requirements: data is Illinois and age is from 0 to 5.

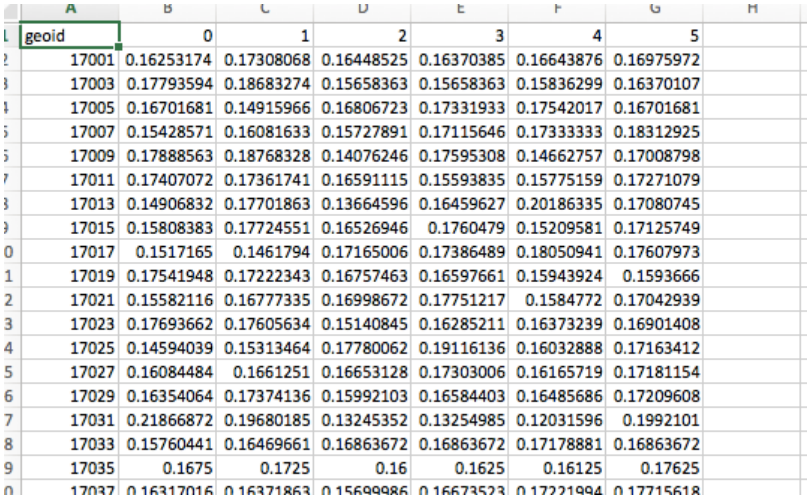
First, I use readlines() to read .txt. Second, I select IL data using split() and if condition. The delimiter of split() is tab. I use slicing to keep states FIPS code, counties FIPS code, ages from the first columns and delete other information. I select age from 0 to 9. There is a space before age if age is from 0 to 9. I use split() to separate the first column into two, and use if condition to select data with 2 lengths. The delimiter of split() is a space. If the age equals to 10 or above, there is no space, so the length is 1. Now, the first column represents "geoid". The second column represents age. I use join function to joined geoid, age with numbers. Next, I separate FIPS code, age and numbers from one long string into three strings by using .split(). I create an empty row named "row" and use for loop to append rows into the row list. I create an empty list "age0to5" to store age which equals or less than 5. Since the age in the row are strings. I use int() to covert string to number. I use for loop and if condition to get the row that meets the requirement. To save the result into "age0to5", I use append() function. I stored the filtered

data into a new csv file named as “pep.csv” in case I needed in the future. In addition, “pep.csv” is much smaller than the original file. The Figure 1 below shows the first several rows from “pep.csv”.

	A	B	C	D
1	17001	0	376	
2	17001	1	386	
3	17001	2	373	
4	17001	3	381	
5	17001	4	386	
6	17001	5	383	
7	17001	0	10	
8	17001	1	18	
9	17001	2	15	
10	17001	3	15	
11	17001	4	15	
12	17001	5	16	
13	17001	0	359	
14	17001	1	386	
15	17001	2	357	
16	17001	3	361	
17	17001	4	365	
18	17001	5	368	
19	17001	0	11	
20	17001	1	10	
21	17001	2	11	
22	17001	3	9	

Fig.1 pep file

I use pandas to read the “pep.csv” file I just create. The goal is to get the number of children across birth to age 5 by counties and the total number of children age 5 and under by counties. A geoid presents a county. Firstly, I have to group the data by geoid and age. For example, if rows have same geoid and age, the numbers which is the third column will be added up. I use groupby([0,1]) function to group geoid and age. 0 means the first column which is geoid. 1 means the second column which is age. To add number together I used sum(). Groupby() function creates an index. After I get the number summed up, I use reset_index() to remove the index because I don’t need index. I rename the column name by rename() function. I rename the number column as “number”. Secondly, I have to group the data by geoid only to get the total number of children age 5 and under. I use groupby(), sum(), and reset_index() functions again. I rename the columns by rename() function again. I rename the number column as “total”. Thirdly, I joined the two groupby results together by using merge() function. The common attribute is geoid. To get the percent, I divide the “number” column by the “total” column. Finally, I create a new data frame named as “matrix”. I add the geoid column, age column and percent column into the new data frame. I export “matrix” as a csv file. The Figure 2 below shows the first several rows from “matrix.csv”.



geoid	0	1	2	3	4	5
17001	0.16253174	0.17308068	0.16448525	0.16370385	0.16643876	0.16975972
17003	0.17793594	0.18683274	0.15658363	0.15658363	0.15836299	0.16370107
17005	0.16701681	0.14915966	0.16806723	0.17331933	0.17542017	0.16701681
17007	0.15428571	0.16081633	0.15727891	0.17115646	0.17333333	0.18312925
17009	0.17888563	0.18768328	0.14076246	0.17595308	0.14662757	0.17008798
17011	0.17407072	0.17361741	0.16591115	0.15593835	0.15775159	0.17271079
17013	0.14906832	0.17701863	0.13664596	0.16459627	0.20186335	0.17080745
17015	0.15808383	0.17724551	0.16526946	0.1760479	0.15209581	0.17125749
17017	0.1517165	0.1461794	0.17165006	0.17386489	0.18050941	0.17607973
17019	0.17541948	0.17222343	0.16757463	0.16597661	0.15943924	0.1593666
17021	0.15582116	0.16777335	0.16998672	0.17751217	0.1584772	0.17042939
17023	0.17693662	0.17605634	0.15140845	0.16285211	0.16373239	0.16901408
17025	0.14594039	0.15313464	0.17780062	0.19116136	0.16032888	0.17163412
17027	0.16084484	0.1661251	0.16653128	0.17303006	0.16165719	0.17181154
17029	0.16354064	0.17374136	0.15992103	0.16584403	0.16485686	0.17209608
17031	0.21866872	0.19680185	0.13245352	0.13254985	0.12031596	0.1992101
17033	0.15760441	0.16469661	0.16863672	0.16863672	0.17178881	0.16863672
17035	0.1675	0.1725	0.16	0.1625	0.16125	0.17625
17037	0.16317016	0.16371863	0.15699986	0.16673523	0.17221994	0.17715618

Fig. 2 Matrix

Part II

Firstly, I use pandas to read “matrix.csv” file I just created and save in “matrix”. The type of geoid is int. But I need to convert it into string because I will use it to merge with population file. The common attribute is geoid. They must be in the same type. The geoid in the population file is string, so I have to convert geoid from integer into string from “matrix”. I use apply(str) function to convert geoid from integer to string.

Secondly, I collected and prepared population data by counties from ACS. The table is “ACS_15_5YR_B01001_with_ann.csv”. I use pandas to read the file and save as “population”. I selecte four columns: geoid, region name, male under 5, and female under 5. The corresponding columns in “population” are “GEO.id”, “GEO.display-label”, “HD01_VD03” and “HD01_VD27”. I save them as “geoid”, “display”, “boy” and “girl”. I aggregate “boy” and “girl” to get the total number of children under 5 and saved as “total”. I combine “geoid”, “display” and “total” by using concat() function and save as “child”. I rename the header of total as “population” because it doesn’t have a header name. I also rename the header of “GEO.display-label” as “county” to make it shorter. In this file, there are many different regions. County is one of them. The geoid of county contains “0500000” which can separate county from other regions. I target on geoid column and I use contains() to get data in county region. child['GEO.id'].str.contains(region_code) is to decide whether the data in geoid column contains region code which is “0500000”. It will return True or False.
child[child['GEO.id'].str.contains(region_code)] can show the data which satisfy the requirement. Now the geoid column from “child” is longer than the geoid column from “matrix”. For example, geoid from “child” is 0500000US17001 and geoid from “matrix” is 17001. To make them exactly the same, I use str[9:15] function to get the last 5 digits.

Finally, I use merge() function to join “child” and “matrix” together by geoid and save as “detail_pop”. The joined table is shown below.

```
          GEO.id                      county population  geoid         0  \
0  0500000US17001      Adams County, Illinois   21252007  17001  0.162532
1  0500000US17003  Alexander County, Illinois     286219  17003  0.177936
2  0500000US17005        Bond County, Illinois     447410  17005  0.167017
3  0500000US17007       Boone County, Illinois   16941633  17007  0.154286
4  0500000US17009       Brown County, Illinois     134136  17009  0.178886

          1         2         3         4         5
0  0.173081  0.164485  0.163704  0.166439  0.169760
1  0.186833  0.156584  0.156584  0.158363  0.163701
2  0.149160  0.168067  0.173319  0.175420  0.167017
3  0.160816  0.157279  0.171156  0.173333  0.183129
4  0.187683  0.140762  0.175953  0.146628  0.170088

Process finished with exit code 0
```

Fig. 3 Joined table

To do calculation, columns have to be number type. The type of “population” is string. I use apply(int) to convert “population” from string to integer. To estimate children under age 1 from ACS, I multiple “population” with column 0. The result will be float. However, the number should be integer. I use astype(int) function to convert float to integer. Recall from the part I, column 0 is the percent of children under age 1 by total number of children age 5 and under from PEP. The calculation is the same to calculate age 1, age 2, age 3, age 4 and age 5. I use for loop to do the estimation. After all ages are estimated, the table looks like the figure 4 shown below.

```
          GEO.id                      county  population  geoid         0  \
0  0500000US17001      Adams County, Illinois    21252007  17001  0.162532
1  0500000US17003  Alexander County, Illinois      286219  17003  0.177936
2  0500000US17005        Bond County, Illinois      447410  17005  0.167017
3  0500000US17007       Boone County, Illinois    16941633  17007  0.154286
4  0500000US17009       Brown County, Illinois      134136  17009  0.178886

          1         2         3         4         5     pop0     pop1  \
0  0.173081  0.164485  0.163704  0.166439  0.169760  3454125  3678311
1  0.186833  0.156584  0.156584  0.158363  0.163701    50928    53475
2  0.149160  0.168067  0.173319  0.175420  0.167017    74724    66735
3  0.160816  0.157279  0.171156  0.173333  0.183129  2613851  2724491
4  0.187683  0.140762  0.175953  0.146628  0.170088    23995    25175

      pop2     pop3     pop4     pop5
0  3495641  3479035  3537157  3607734
1    44817    44817    45326    46854
2    75194    77544    78484    74724
3  2664561  2899669  2936549  3102508
4    18881    23601    19668    22814
```

Fig. 4 Estimated table

Now the matrix column is not useful. I delete them by using drop() function. The final “detail_pop” looks like the figure 5 shown below. Its name is “detail_pop”

```
          GEO.id                      county  population  geoid     pop0  \
0  0500000US17001      Adams County, Illinois    21252007  17001  3454125
1  0500000US17003  Alexander County, Illinois      286219  17003    50928
2  0500000US17005        Bond County, Illinois      447410  17005    74724
3  0500000US17007       Boone County, Illinois    16941633  17007  2613851
4  0500000US17009       Brown County, Illinois      134136  17009    23995

      pop1     pop2     pop3     pop4     pop5
0  3678311  3495641  3479035  3537157  3607734
1    53475    44817    44817    45326    46854
2    66735    75194    77544    78484    74724
3  2724491  2664561  2899669  2936549  3102508
4    25175    18881    23601    19668    22814
```

Fig. 5 Modified table

I saved it to a csv file named as “Population of Children Age 5 and Under.csv”.

Part III

The estimation of race data is exactly same as estimation of population. I create a function named as “estimationbyrace” to estimate the data using the same method as population. To use the function, three variables need to be defined “race”, “type” and “total”. “race” is the input file read by pandas. For example, race can be pd.read_csv('data/ACS_15_5YR_B01001A_with_ann.csv'). “type” is race type. For example, race = “white”. “total” is the column which male under 5 plus the column which female under 5.

I take race: white population as an example. When all three variables are defined, I call the function estimationbyrace(race, type, total). The function can return the final table. I name the final table as detail_white and saved it into a csv file.

After estimating four race files, I get detail_white, detail_black, detail_aian, and detail_asian. I delete geoid column and county column by a for loop, because they will appear multiple times if I merge them. I just need one geoid column and one county column. After that, I merge detail_pop and the four races data by a for loop and save as “all”. I export it into a csv file named “All.csv”.

The End

I get “All.csv” and separated estimated csv files. These are estimations for ACS data across all ages from birth to age 5 based on the PEP data.