Exercise: Distributional Analysis Using Pandas

Summary

This exercise carries out a detailed distributional analysis of the tax in the earlier market equilbrium assignment.

Input Data

File **households.csv** is the earlier file giving data on 1000 households. As before, it has five columns, `id`, `type`, `inc`, `a` and `b`, and there is one row for each household. The `type` variable wasn't used before but it will be important in this exercise. It ranges from 1 to 4 and indicates the demographic type of the household. Here we'll think of it as indicating the region of the country where the household resides. In other studies it could be used to indicate other characteristics, such as the race, gender, or age of the household's head, the size of the household, whether the household lives in an urban or rural area, and so on. In those cases there would be many more than four types.

File **quantities.csv** gives the quantities demanded by each household under the base case and tax policy simulations. It has three columns, one for the household's ID and one each for the household's demands under the base and policy cases: `id`, `qd1` and `qd2`. In case you're curious, it was produced by running the earlier `ind_demand()` function for each equilibrium price and then writing out the output. However, you do not need to do that for this exercise: you should use the provided **quantities.csv** without recalculating it.

Deliverables

A script called **etr.py** that calculates the effective tax rate (ETR) for each household and then reports the median ETR for the groups indicated below. Then update **results.md** replacing the *TBD* placeholders with your answers to the questions in the file. Please try not to alter the other Markdown in the results.md file: it makes it easier to tell the difference between the questions and answers.

Instructions

To help make the structure of the analysis clearer, the instructions below have been divided into sections labeled A, B, etc. However, that's just for clarity: you should build a single script that does all of the steps.

A. Initial setup

1. Import `pandas as pd`.

2. Define a function called `print_groups()` that takes two arguments, a dataframe of household information called `hh`, and a list of variables called `group_vars` that will be used to group the data. The body of the function should do the following:

    1. Group the data by setting variable `grouped` to the result of calling the `.groupby()` method on `hh` using `group_vars` as the argument. If you get a FutureWarning on this line see the Tips section.

    2. Set variable `med` to the result of calling the `.median()` method on `grouped` column `"etr"`.

    3. Round the results to two digits by setting variable `med` to the result of calling the `.round()` method of `med` using 2 as the argument. If you want, you can combine this step with the one above by adding the call to `.round()` to the end of the statement.

    4. Print `med`.

    5. Return `med`.

3. Create a dataframe called `hh` by using `pd.read_csv()` to read `"households.csv"`. Use the keyword `index_col='id'` to set the index to the `id` field for each household.

4. Create a dataframe called `q` by using `pd.read_csv()` to read `"quantities.csv"`. Again use the keyword `index_col='id'` to set the index to the `id` field for each household.

5. Determine the income quintile of each household by setting `hh` column `"quint"` to the result of calling the Pandas function `pd.qcut()` on the household income data. The `pd.qcut()` function divides its input into bins and returns a series containing the bin number of each input record. The first argument to `qcut()` should be the column of incomes, `hh['inc']`, the second argument should be `5` to request quintiles, and the third should be `labels=[1,2,3,4,5]` to have the quintiles labeled 1-5. Please note that `pd.qcut()` is a standalone Pandas function like `pd.read_csv()`: it's not a series or dataframe method.

B. Computing the ETRs

1. Now create a variable called `pd1` that is equal to `53.35` and one called `pd2` equal to `55.27` (the equilibrium buyer prices from the market assignment). Then create a variable called `dp` that is equal to `pd2 - pd1` (the buyer burden of the tax).

2. Compute the ETRs by multiplying `100` times `dp` times the `qd2` column of `q` divided by the `inc` column of `hh`. Store the result in the `hh` dataframe as column `'etr'`. Notice that it's not necessary for the quantity and income data to be in the same dataframe because Pandas will use the indexes to match up the income and quantity variables. Also, this would work correctly even if the rows of the `hh` and `q` datasets were not in the same order.

C. Aggregating and printing the results

Next we'll explore the results by aggregating the ETR results in several ways: by income quintile, by household type, and by both.

1. Aggregate and print the ETRs by income quintile by setting `med_q` to the result of calling `print_groups()` with arguments `hh` and `['quint']`.

2. Next, aggregate and print the results by type by setting `med_t` to the result of calling `print_groups()` with arguments `hh` and `['type']`.

3. Finally, aggregate and print the results by both type and income quintile by setting `med_b` to the result of calling `print_groups()` with arguments `hh` and `['type', 'quint']`.

4. Print the index for `med_b`. Notice that it's a list of tuples with the type as the first element and the quintile as the second element.

5. Print an appropriate heading and then print the detailed medians for type `3` by using the `.xs()` method (short for cross-section) on `med_b` with arguments `3` and `level="type"`. The `level` keyword tells Pandas that the `3` is associated with the `"type"` part, or level, of the index.

6. Print an appropriate heading and then list the medians for the 5th quintile by using the `.xs()` method with arguments `5` and `level="quint"`.

D. Comparing within-group ETRs across income quintiles

1. Finally, to emphasize the power of the automatic alignment built into Pandas, we'll do a quick calculation showing how the ETR for each type changes moving up the income distribution. Start by setting `etr_lowest`

to the result of calling `.xs()` on `med_b` with arguments `1` and `level="quint"`.

2. Print `etr_lowest`. You'll see that it's a series with `"type"` as its index.

3. Now set variable `etr_change` to `med_b` minus `etr_lowest`. Pandas will automatically align the types, broadcast `etr_lowest` across all quintiles for each type (remember that `med_b` has two levels), and then do the subtraction. The result will be the difference in the ETR for each type and quintile from the ETR for quintile 1 of the same type. It's a quick way to see that the tax is progressive because the differences get progressively larger within each type.

4. Print `etr_change`.

E. Updating results.md

1. Now use the results to answer the questions in results.md.

Submitting

Once you're happy with everything and have committed all of the changes to your local repository, please push the changes to GitHub. At that point, you're done: you have submitted your answer.

Tips

- To convince yourself that the last calculation is working correctly, use a pocket calculator to check a couple of the numbers. This feature of Pandas (aligning and broadcasting across index levels) is very useful and avoids a lot of steps that would otherwise have to be done manually. Pandas is essentially doing a many-to-one join on `med_b` and `etr_change` before doing the subtraction. However, it's all automatic and the code is much cleaner than it would be otherwise.

- If you get a FutureWarning on the `.groupby()` call in the function it's OK to ignore it. It's warning you that an internal setting will be changed in future versions of Pandas but that particular setting doesn't make a difference in this script.