Exercise: Distributional Analysis Using Pandas

Summary

This exercise uses Pandas to do the ETR calculation from the earlier assignment.

Input Data

Files **households.csv** and **quantities.csv** are the CSV files from the previous distributional analysis. As you'll probably recall, there are 1000 households in the analysis and households.csv has their attributes. It has five columns, `id`, `type`, `inc`, `a` and `b`, and there is one row for each household. The second file, quantities.csv, has three columns: `id`, `qd1`, and `qd2` and again there is one row for each household.

Deliverables

A script called **etr.py** that carries out the calculations described below.

Instructions

1. Import pandas as pd.
2. Define a function called `print_groups()` that takes two arguments, a dataframe of ETRs called `hh`, and a list of variables to be used to group the data called `group_vars`. 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.
    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 `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.
6. As in the earlier exercise, create a variable called `pd1` that is equal to `53.35` and one called `pd2` equal to `55.27`. Then create a variable called `dp` that is equal to `pd2 - pd1`.
7. 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. Also, this would work correctly no matter what order `hh` and `q` were in because Pandas will use the indexes to match up the income and quantity variables.
8. Now aggregate and print the ETRs by quintile alone by setting `med_q` to the result of calling `print_groups()` with arguments `hh` and `['quint']`.

9. Next, aggregate and print the results by type alone by setting `med_t` to the result of calling `print_groups()` with arguments `hh` and `['type']`.
10. Finally, aggregate and print the results by both type and quintile by setting `med_b` to the result of calling `print_groups()` with arguments `hh` and `['type', 'quint']`.
11. 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.
12. 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.
13. Print an appropriate heading and then list the medians for the 5th quintile by using the `.xs()` method with arguments 5 and `level="quint"`.
14. 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"`.
15. Print `etr_lowest`. You'll see that it's a series with `"type"` as its index.
16. 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.
17. Print `etr_change`.

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 calculation 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.