Exercise: Residential Solar in NY

Summary

This exercises explores data on residential solar installations in New York by county. However, the main purpose is really to illustrate some key features of the `pandas` module.

Input Data

The input data is contained in **res_solar_by_county.csv**. It contains information on completed residential solar installations in New York State between 2000 and 2019. It has five fields: "county"; "projects", the number of installations in the county; "total_cost", the total cost of all the installations; "total_incentive", the total subsidy provided by New York State; and "total_kw", the total capacity built, in kilowatts. Note that "total_cost", "total_incentive" and "total_kw" are sums over all the projects in the corresponding county.

Deliverables

Please prepare a script called **solar.py** that carries out the steps described below.

Instructions

1. Import the `pandas` module as `pd`
2. By default, Pandas limits the number of rows it displays when a dataframe and or series is printed. That's convenient for very large datasets when no one would want to see every row but we're going to override it here. Set the maximum number of rows to print to `None` as shown below to remove the limit.

   ```
   pd.set_option('display.max_rows',None)
   ```

3. Use the `pd.read_csv()` to read the input file into a dataframe called `solar`.
4. Set the index of `solar` to be the county. Use the `inplace` keyword to modify `solar` rather than creating a new dataframe.
5. Print an appropriate heading and then print the list of columns for `solar`. The columns will be a Pandas Index object; you may want to use `list()` to turn that into a simple list, which will look a little clearer when printed.
6. Print an appropriate heading and then print the index for `solar`. As with the columns, you may want to clean up the output using `list()`.
7. Create a variable called `count` that is equal to the "projects" column of `solar`. It will be a Pandas series object, which is much like a dictionary with the counties as keys and the number of projects as the values.
8. Print an appropriate heading and then print `count`.
9. Now set up a list called `some_cny_counties` that contains "Onondaga", "Oswego", and "Wayne".
10. Print an appropriate heading and then print the project counts for the selected list via `count[some_cny_counties]`. Notice that it's a Series object since several index values (the counties) were given.
11. Now print an appropriate heading and the print the value of `count` for Albany county. Notice that the value is a scalar, not a Series, since only a single element was requested.
12. Now sort `count` from high to low using the `sort_values()` and `ascending=False`. Call the sorted series `high_to_low`.

13. Select the first five elements from `high_to_low` using `.iloc[0:5]` and call the result `top_five`. If the notation isn't clear recall that 0:5 in a subscript context is shorthand for a list of 5 values starting with 0 and ending with 4.
14. Print an appropriate heading and then print `top_five`.
15. Compute the mean values of all variables in `solar` by dividing it by `count` using the statement below:

    ```
    means = solar.div(count,axis='index')
    ```

    If you're new to Pandas, the result of this will probably seem almost miraculous: a single short line will create a new dataframe where all of the columns have been divided by the number of projects in the corresponding county.

    That will be true even if the rows of `count` and `solar` are not in the same order: Pandas will automatically match the indexes of `count` and `solar` during the calculation. If you aren't absolutely confident about this, try the same calculation using `high_to_low` instead of `count`. Even though the rows of `high_to_low` are in a different order from the rows of `solar`, the result will be exactly the same.

    The Tips section below explains what the `axis='index'` keyword does and why it's needed.
16. Print an appropriate heading and then print `means`.
17. Now we'll compare the means for all the counties to those for Onondaga. As a first step, use `.loc['Onondaga']` to pull the row for Onondaga out of `means`. Call the result `onondaga_row`.
18. Now create a variable called `relative` by dividing `means` by `onondaga_row` as shown:

    ```
    relative = means/onondaga_row
    ```

    The Tips section explains why no `axis` keyword is needed in this case.
19. Create a variable called `rel_incent` that is equal to the "total_incentive" column of `relative`.
20. Print a suitable heading and then print `rel_incent` sorted in ascending order using the `sort_values()` call.
21. There's no Markdown deliverable for this assignment but it's interesting to note that there are large differences in the mean incentive across counties. That's partially, but not completely, explained by differences in the average size of the projects.

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

- When carrying out an operation that involves two objects that aren't the same shape, such as a dataframe and a series, Pandas will automatically sweep the smaller object across the large one in a process known as "broadcasting". However, it has to deal with an inherent ambiguity: should the smaller object be treated like a column, and swept across the columns of the large object, or should it be treated like a row and swept down the larger object's rows.

  The `div(count,axis='index')` call resolves the ambiguity by telling Pandas to treat `count` as a column and line it up with the index of the dataframe. To have Pandas treat something as a row, as was the case with `onondaga_row`, the call would be `div(onondaga_row,axis='columns')`.

However, when given an expression that links two objects by a simple operator, such as `means/onondaga_row`, rather than a call like `div()`, Pandas will *assume* that it should use row-wise broadcasting. That's why `means/onondaga_row` worked as intended.

Be aware of this behavior: it's handy but if you forget and accidentally use something like `some_data_frame/some_column` you won't get what you expect and the result will usually be full of missing data.