Exercise: Joining Weather Data onto Electricity Usage

Summary

This exercise explores how electricity use depends on the season (month) and daily weather conditions (hourly temperature). It demonstrates an outer join on two datasets using three join keys. It also shows how duplicate records can be handled and how box plots can be generated.

Input Data

There are two input files: **use.csv**, which contains hourly electricity usage for 2014 for the household in Austin, Texas, in the exercise earlier in the semester; and **weather.csv**, which contains data on the weather in Austin that year.

Deliverables

A script called **join.py** that joins the two datasets, writes out the combined data as **join.csv**, and plots two figures, **by_temp.png** and **by_month.png**.

Instructions

1. Import `pandas` as `pd` and `matplotlib.pyplot` as `plt`.
2. Set the default resolution for figures to `300` DPI.
3. Set `weather` to the result of using `pd.read_csv()` to read `"weather.csv"`.
4. Create a dictionary called `fix_name` for streamlining one of the names in the weather file. It should have one key, `"Temperature (F)"`, and the key's value should be `"degrees"`. Don't overlook the space in the original name.
5. Rename the temperature variable by setting `weather` to the result of calling the `.rename()` method on `weather` with the keyword argument `columns=fix_name`. Be sure to check `weather.columns` before going on to make sure the change was successful.
6. Now look for records with duplicated timestamps. Set `is_dup` to the result of calling the `.duplicated()` method on `weather` with two arguments: `subset="Local Hour"` and `keep=False`. The result will be a series of true and false values indicating whether there is another record with the same timestamp. The `keep=False` argument causes all records with identical timestamps to be considered duplicates. Without it, Pandas only considers the second and subsequent records as duplicates: that is, it does not consider the first record with a repeated timestamp to be a duplicate.
7. Select the duplicated records by setting `dups` to `weather[ is_dup ]`.
8. Print `dups`.
9. Now filter out the duplicated records by setting `weather` to the result of calling the `.drop_duplicates()` method on `weather` with the argument `subset="Local Hour"`.
10. We'll check that there's now only one record for each of the problematic hours. Set `fixed` to the result of calling the `.isin()` method on the `"Local Hour"` column of `weather`. As the argument for `.isin()` use the `"Local Hour"` column of `dups`. The outcome will be a series of true and false values indicating whether each record in `weather` has a timestamp that matches any of the timestamps in `dups`.
11. Now print `weather[ fixed ]`. If all has gone well, it should show one record for each of the problematic timestamps. The records should be the first of each set of duplicates.

12. Now we'll split the timestamp into pieces to allow the records to be joined to the usage data. Set `date` to the value of calling the Pandas function `pd.to_datetime()` on the `"Local Hour"` column of `weather`. The result will be a series in the internal datetime format used by Pandas.
13. Print the `"Local Hour"` column of `weather` and then, in a second statement, print `date`. Notice that the dates are the same even though the format is different.
14. Set column `"month"` in `weather` to `date.dt.month`, which is the month part of each date. Note that `.month` is an attribute of the date, not a function, so no parentheses are used after it. The same is true for the next few attributes below.
15. Set column `"day"` in `weather` to `date.dt.day`.
16. Set column `"hour"` in `weather` to `date.dt.hour`.
17. Set column `dow` in `weather` to `date.dt.dayofweek`. This will be the day of the week, where 0 indicates Monday and 6 indicates Sunday. It would be useful in a regression because electricity use usually varies with the day of the week.
18. Next, read in the usage data by setting `use` to the result of using `pd.read_csv()` to read `"use.csv"`.
19. Create a list called `join_keys` that consists of the three strings that together identify the hour of the year: `"month"`, `"day"` and `"hour"`.
20. Now merge the two datasets using a one-to-one outer join. Set `merged` equal to the result of calling the `.merge()` method on `use` with the following arguments: `weather`, `on=join_keys`, `how="outer"`, `validate="1:1"` and `indicator=True`.
21. Print the result of calling the `.value_counts()` method on the `"_merge"` column of `merged`. It will show the number of records that were in both datasets and the number that were only in the left dataset (`use`) or only in the right dataset (`weather`). In this exercise, expect that there will be some records that are not in both datasets. We'll leave all the records in but those with missing data won't show up in the plots later on.
22. Now create a temperature bin variable that rounds the temperature to the nearest ten degrees. Set the `"tbin"` column of `merged` to the result of calling `.round(-1)` on the `"degrees"` column of `merged`. The -1 tells `.round()` to round to one place to the *left* of the decimal point: that is, to the tens place.
23. Check the results by printing the result of calling `.value_counts()` on the `"tbin"` column of `merged`. It should produce a small table with counts of records in the 80s, 70s, and so on.
24. Save the results by calling `.to_csv()` on `merged` with arguments `"join.csv"` and `index=False`. The `index` argument omits the index, which in this case is just the row number of data.
25. Now start a new figure and create an empty set of axes by setting the tuple `fig1, ax1` to the result of calling `plt.subplots()`.
26. Now draw box plots for electricity usage in each temperature bin. Call the `.boxplot()` method on `merged` with the following four arguments: `"usage"` (the Y variable), `by="tbin"` (the X variable), `ax=ax1` (put the graph on the `ax1` axes), `grid=False` (turn off some unnecessary grid lines), and `showfliers=False` (turn off drawing of outliers). Please note that this and the remaining commands for drawing this figure are all pure method calls and don't generate any variables. That is, they should be called like this: `name.method()` and *not* like this: `var = name.method()`.
27. Call the `.suptitle()` method on `fig1` with the argument `"Usage by Temperature"` to set the figure's title.

28. Call the `.set_title()` method on `ax1` with argument `None` to turn off an extra title for the axes that is generated by default.
29. Call the `.set_ylabel()` method on `ax1` with argument `"kW"` to set the label for the Y axis.
30. Call the `.set_xlabel()` method on `ax1` with argument `"Temperature Bin"` to set the X axis label.
31. Call `.tight_layout()` on the figure as usual and then use `.savefig()` to save the figure as `"by_temp.png"`.
32. Now create a similar box plot of usage by month. Repeat the steps above starting with `plt.subplots()` but use `"month"` as the by-variable in the box plot and adjust the `.suptitle()` and `.set_xlabel()` calls accordingly. Tighten the layout and then save the file as `"by_month.png"`.

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.