# 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 `weather` to the result of calling `pd.read_csv()` on `"weather.csv"`.
3. 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"`.
4. Rename the temperature variable by setting `weather` to the result of calling the `.rename()` method on `weather` with two arguments: `fix_name` and `axis="columns"`.
5. 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.
6. Select the duplicated records by setting `dups` to `weather[ is_dup ]`.
7. Print `dups`.
8. 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"`.
9. 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`.
10. 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.
11. 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.
12. 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.
13. Set column `"month"` in `weather` to `date.dt.month`, which is the month part of each date.

14. Set column "day" in weather to date.dt.day.
15. Set column "hour" in weather to date.dt.hour.
16. 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.
17. Next, read in the usage data by setting use to the result of calling pd.read_csv() on "use.csv".
18. Create a list called join_keys that consists of the three strings that together identify the hour of the year: "month", "day" and "hour".
19. 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="_merge".
20. 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.
21. Now create a temperature bin variable that rounds the temperature to the nearest ten degrees. Set the "tbin" column of weather to the result of calling .round(-1) on the "degrees" column of weather. The -1 tells .round() to round to one digit to the left of the decimal point: that is, to the tens place.
22. 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.
23. 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.
24. Now start a new figure and create an empty set of axes by setting the tuple fig, ax1 to the result of calling the Matplotlib function plt.subplots(). To avoid confusion, the call should look like this:

    ```
    fig, ax1 = plt.subplots()
    ```

    The variable fig will refer to the figure as a whole and variable ax1 will be a blank set of axes for use in the plot. The distinction between fig and ax1 is that in other situations a figure might include several panels, each with its own axes.
25. 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), and grid=False (turn off some unnecessary grid lines). 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().
26. Call the .suptitle() method on fig with the argument "Usage by Temperature" to set the figure's title.
27. Call the .set_title() method on ax1 with argument None to turn off an extra title for the axes that is generated by default.
28. Call the .set_ylabel() method on ax1 with argument "kW" to set the label for the Y axis.
29. Call the .set_xlabel() method on ax1 with argument "Temperature Bin" to set the X axis label.
30. Call the .savefig() on fig with arguments "by_temp.png" and dpi=300. Note that .savefig() is called on the figure (fig) rather than just the axes (ax1).
31. Now create a box plot of usage by month. Repeat the steps above but use "month" as the by-variable in the box plot and adjust the .suptitle() and .set_xlabel() calls accordingly. Save the file as "by_month.png" and use dpi=300 again. In setting up the plot, start with the call to plt.subplots()

and end with the call to `fig.savefig()`. Note that it's OK to reuse the names `fig` and `ax1` by putting them on the left again in the call to `plt.subplots()`.

**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.