Lab 7 – Data Preprocessing II

Submission: lab07.ipynb, pushed to your Git repo on master.
Points: **15**
Due: Monday, February 10, 2pm

Objectives

- Munge this!
- Cleaning messy data by integrating other data
- Work on different techniques to assess similar variables

Introduction

You already started the first part of this lab in Lab 6, in which you downloaded a year of hourly weather observations. If not, be sure to complete the first part of the lab before you begin.

Preparing for your lab

Create a **lab07.ipynb** file. Create your header cell, then your first cell to set up your imports:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Work through this lab, and enter the answers to questions that are scattered throughout this lab.

As usual, pay attention to the [P] vs. [M] indicator. All [P] questions should be answered by writing the Python code that outputs the answer. Your code quality should be striving for smart, efficient, well-designed, well-documented code. For code that is asking for specific answers, you should be writing Python code that outputs the answer that is wanted, and nothing more.

In your last lab, you created a **data** directory, and downloaded a dataset from *The Pennsylvania State Climatologist* at http://climate.psu.edu/data/ida representing Williamsport, PA (KIPT). The last lab focused quite a bit on preliminary steps to clean the data. You also had your first real exposure to dealing with times and dates with data, and understood first-hand why it's so important to be sure you set the proper type of every variable in your data. You should have printed out summary statistics for every variable, which should have included the occurrence of missing (e.g. NaN, null) values.

There are two important observations to make here:

1) You had missing data
2) The data that you do have may be questionable, as it is quite noisy at times.

How will we create a complete dataset?

Your task

The scenario for this lab is as follows. You have some missing data, and you've decided to estimate the data from nearby stations. This is a **data integration** challenge. KIPT is a pretty reliable station, but there are missing values, and a few other peculiarities.

Exercises

1) [P] Create a Python function called `process_FAA_hourly_data` that takes a filename (with path) as a string, and returns a completely processed pandas data frame, ready for analysis. It should do everything that the previous lab did to clean the file, including
    a. converting all numeric variables to their simplest numeric types
    b. converting the date/time stamp (first variable) to a pandas `DatetimeIndex`, which becomes the actual index for the data frame. (It should drop the date time variable after moving it to become the index.)
    c. If you did not do this in the last lab, make sure that the `DatetimeIndex` is localized to a specific timezone! This is very important! What time zone? Did you notice the header? **The time stamp is in GMT**, so be sure to localize the index accordingly. HOW? After you set up the index, you can do:

    ```
    df.index = df.index.tz_localize(tz='GMT')
    ```

    NOTE: The last exercise in the previous lab had you eliminate a year from the data for the very last problem. Do NOT do that here! We'll explore that again later.

2) [P] Use your new function to read in the KIPT data file you downloaded in the last lab. Store your data frame as `df_kipt`. Output the results of `info()` and `describe()` to confirm you read it in correctly.

3) [P] In the last lab, you assessed the number of missing dates in your data, under the assumption that every hour should have an observation. For now, we'll leave the fact that there are completely missing hourly observations from the weather station. This time, report the number of missing values in each variable of `df_kipt` from the data you have. (HINT: One way is to use the `isna()` method.)

4) [M] Which variables seem to have the most consistent, complete observations? Which are missing the most? Are they really "missing", or are they observations where an event did not occur? Discuss.

5) [P/M] Let's pay attention to "`Average Temp (F)`". Are there hours of the day are most likely to have missing values? Report the frequency over each *hour* that has missing "`Average Temp (F)`" values. Be sure to report the LOCAL times according to the time zone "`US/Eastern`". Output the hours in order of the most frequently missing to least. Then, as a comment, just interpret what you see. Do you see a pattern? Do missing temps tend to happen at a certain time of day?

    (HINT: This might be challenging. First, as always, select the subset of your data matching your criteria. Then, for these data, look at the `index`. Date / time data types have LOTS of attributes themselves… such as `hour`. What do you get if you count these values?)

6) [P/M] Repeat the previous exercise, but this time, assess the same variable for the day of the week. (NOTE: Be sure to note what a 0 is. In pandas, a 0 for day of the week is a Monday! (See https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DatetimeIndex.dayofweek.html)

Next, let's suppose your supervisor suggests that you grab similar data from a nearby station. The aim will eventually be to use these nearby stations to fill in our data.

7) [P] Read in the file **`FAA_PA_stations.csv`** provided on Moodle. It's not actually a comma separated file, but a *tab* separated file. Store the data frame as `stations`. Show `stations.info()` after you read in the data.

8) [P] As usual, you must always assess your missing data, if any. Are there any observations (rows) in `stations` that have missing data? Output them, and eliminate them from your data. Be sure to `reset_index(drop=True)` to reset the index in case any observations are dropped. Output `stations.info()` again.

9) [P] Examine the data frame of `stations` by showing the first few observations using `stations.head(10)` In particular, pay close attention to the variables `Lat` and `Lon`. These represent the precise latitude and longitude geolocation for the weather station.

10) Create a new variable in `stations` called `"distKIPT"` that stores the distance of every station in PA to Williamsport (KIPT). Use a standard Euclidean distance calculation (over latitude and longitude) to compute the distance between the stations. As a reminder, Euclidean distance between two points defined by (x_1, y_1) and (x_2, y_2) is:

$$dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

11) [P] Output the top 10 stations that are closest to KIPT. (The closest one should be to itself!) The stations should be listed in order of increasing distance from KIPT.

12) [P] Using your results, go back to the PSU climate website (http://climate.met.psu.edu/data/ida/), and download the *faa_hourly* data for the THREE closest stations that have hourly data available in the same date range as the data you downloaded from KIPT (i.e. 2000-01-01 → 2019-12-31). (HINT: You may need to skip a station because it does not have data available in this range.) Copy the data into your `data` folder. Then, read in each data file into its own data frame using your function. You should have four data frames: `df_kipt`, and three other data frames representing the three closest stations. Show the result of `info()` on your three new data frames.

    (HINT: KSEG, KUNV, KCXY)

You're going to repair our KIPT data, but how? Since we have data from nearby stations with independent readings, let's see how many of our missing data can be filled in from a nearby station. Again, for simplicity, we're only going to focus on **average temperature** (column 1).

As an advanced CS elective, you should be going for efficient, clean code. It need not be perfect, but it should be well-thought out and documented so you can readily show it to prospective employers, and also use it later! Remember - you are going to start doing more advanced analyses very soon! So, when your code or your output is not obvious or intuitive, **write markdown after your output is generated that explains your code, and interprets your results!**

13) [P] Create a new data frame called `df_ave_temps` that contains the average temperature from all four stations. Name the variables with the four-letter station identifier (e.g. `"KIPT"`). The index should have a COMPLETE hourly date range from the start date `"20000101 00:00:00 GMT"` to finish date `"20191231 23:00:00 GMT"`. The results should be a complete dataset with an observation for every hour. If hourly observations are missing from the station you are copying from, then a NaN value should be stored for that entry. You will use these data for the remainder of this exercise.

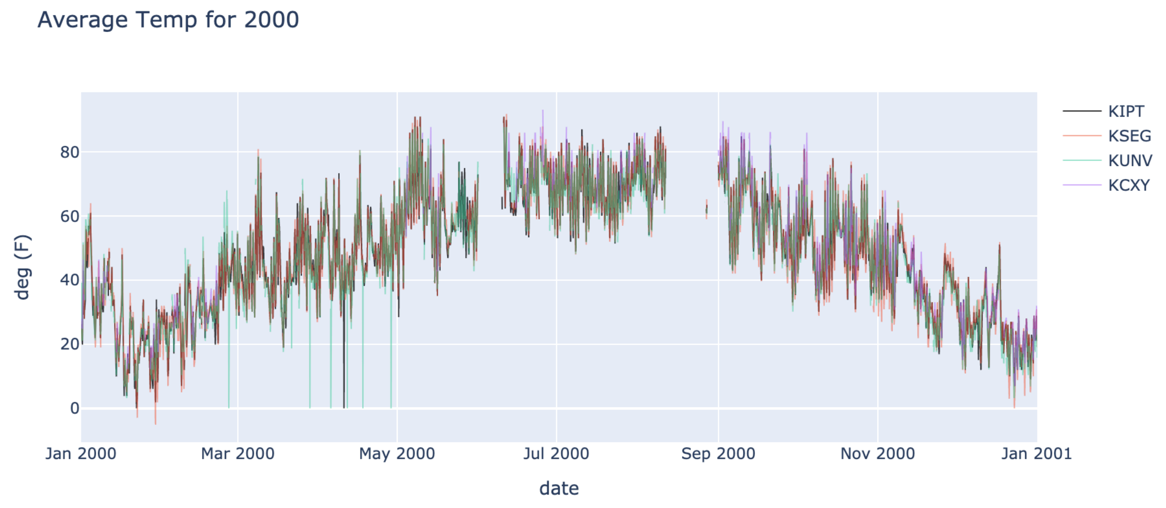
Show `df_ave_temps.info()`

14) [P] Report the number of observations in `df_ave_temps` that have missing temps for each location. You should output a `DataFrame` or `Series` that shows four values.

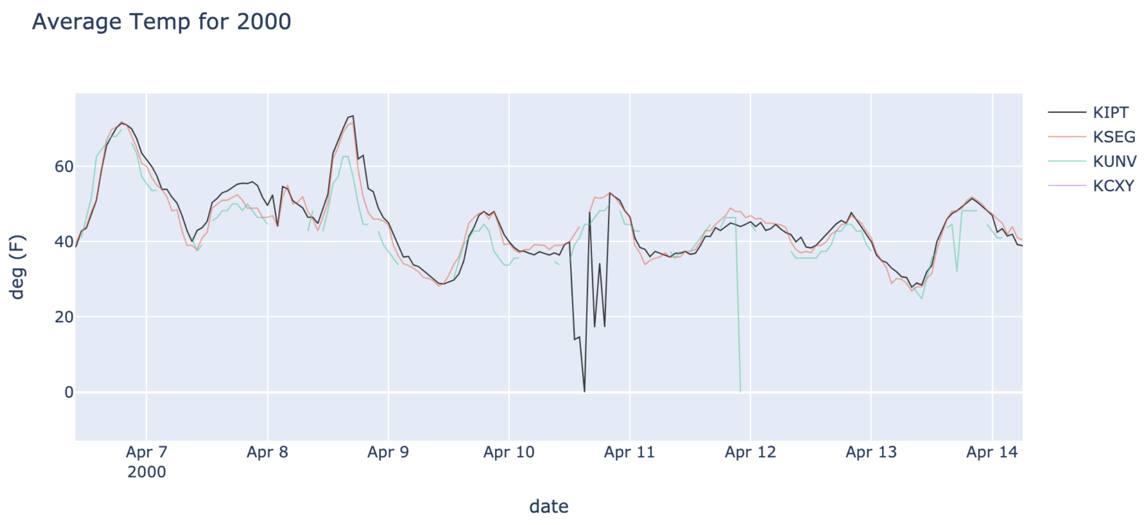
15) [P] Report the number of missing data in KIPT that have at least one alternative station with an existing value. You should output a statement like, "`There are XXXX out of XXXX values in KIPT that can be restored from other locations.`" Also, show the first 10 observations of these data that meet this criteria using `head(10)`.

16) [P] Remember that exercise in the previous lab that gathered the number of missing data by year? Report the number of missing data in KIPT by year that CANNOT be restored from any of the other stations. What year is standing out as the least likely to be successfully restored?

17) [P] It still looks like one year in particular is pretty bad. Confirm this **visually** by creating a line plot that plots all four stations for that one year, with each station a different color. Make sure KIPT stands out in some way. Only show the data for that one year you answered in the previous exercise. Interpret your results. In particular, do you see any other problems from any stations? (This might be a good time to use an interactive data visualization tool such as Plotly!) Label your plot (e.g. title, axis, legend)



With the proper interactive data visualization tool, you can easily search through your data for problems. Zooming in on problems can help you develop the additional logic you need in order to clean your data. For example:



The above graph shows an interesting problem on April 10, 2000. It's not a missing value, but it's a problem, and **these problems can have an impact on your data you use for EDA, summary statistics, and modeling.**

18) [P] Looking at your plot of 2000 over all stations should reveal that KUNV is problematic at 6 different times. Report these observations, but report them from your full KUNV dataframe. Show only those observations.

19) [M] How could you algorithmically detect those problems? Keep in mind that simply saying to turn 0.0 into NaN is not an acceptable solution. 0.0 may very well be a real value!

20) [P] Now, write the code to generate line plot(s) for *all* of KIPT visually, and only KIPT. Look for peculiarities, usually indicated by a sudden change that is outside of what would be considered normal, or an extreme temperature reading that would be impossible to observe in reality. Then, document your findings of areas that you think may be problematic, if any.

21) [P] Compute a new Series that represents a running delta temperature between adjacent average temperature readings for KIPT. Then, plot the distribution of these data using whatever visualization you think characterizes this distribution best. (HINT: It's a series of observations over a single numeric variable. What type of plot can reveal the distribution of these data?)

22) [P] Perhaps it's more important to select the station that has the most similar values. Write a function called `compare_station` that takes two `Series` objects of numeric data, and computes the sum of the absolute value of the difference between each pair of numbers in both Series. You should only sum the values that have valid values for both entries. Return the average of these absolute differences.

    Then, call `compare_station` on KIPT and each of the new station, but pass only the average temp vector from each station using your `df_ave_temps`

23) [P] As we learned in class, you could compute a *correlation coefficient* between columns of data to determine similarity. Compute the correlation coefficient between the average temp of KIPT, and each of the other stations you downloaded. They should all be very close to 1, but not quite. What does this technique suggest which station is most similar?

24) [M] Interpret what you have observed so far. Which station is most similar? How would this affect your approach to cleaning your data? Are there other things you might do to clean your data?

25) [P] Create a new attribute called `KIPT_GOOD` in your `df_ave_temps` data frame that keeps all of the original average temp data, but takes the readings from the closest station with available data to replace in the NA values. Be sure to replace the data from the best representative first, then the second best. Ignore the third. When you perform data cleaning, NEVER DELETE YOUR ORIGINAL DATA! Either store it, or just create a separate attribute of cleaned data, or create a separate data frame. AND, be sure to print out what you are doing. Be sure to include a before and after view to indicate how many values you fixed. For example:

```
Starting with 3067 missing values in KIPT
Copying 1505 values from KSEG
Next, copying 329 values from KUNV
We still have 1233 missing values in KIPT
```

26) [P] How many missing values left in `KIPT_GOOD` are "singletons", i.e. a missing value surrounded by two good observations?

27) [P] Go through the variable `KIPT_GOOD`, and convert all singletons to an average of the surrounding observations. For example `[…, 2, Nan, 5, …]` would be filled in with `(2+5)/2 = 3.5`. Then, report the number of values that are still missing in `KIPT_GOOD`.

28) [P] Eliminate that first year of data from `df_ave_temps`. There are too many missing values in these data to make it worthwhile.

29) [P] Generate a final report of the total number of missing values in `df_ave_temps.KIPT_GOOD` by year.

30) [P] Finally, create some good, clean line plots of `KIPT_GOOD`. Create at least three plots using different averaging times. One should be the raw data. Create one by month. Then, create one by year. Be sure they are labeled.

Congratulations! Between this lab and the previous lab, you gained a taste of what life is like as a data munger!

Deliverables

Commit and push `lab07.ipynb`. Be sure you have every cell run, and output generated. All plots should have plt.show() as the last command to ensure the plot is generated and viewable on Gitlab. Verify that your file is pushed properly on Gitlab. Be sure each question is properly annotated.