Guide to the 2021 Python files

Hatta Misra 2021-11-08

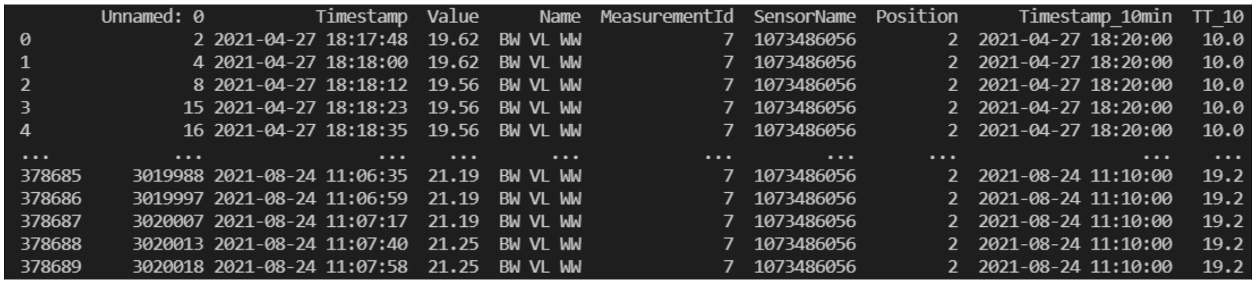
Introduction

The author of this guide, Hatta Misra, conducted a work experience period (Praktikum) from July to November 2021 with SigmaHeat GmbH. I worked to analyze data for the home heating system at the house of Jan Kirschnick's parents. I tried to find correlations between temperatures of the different parts of the system, and also correlations between system parameters and weather parameters (such as outside temperature). To do this, I used Python and its data science library, pandas, to analyze and visualize the data.

Pandas, the data science library for Python

The files in this project heavily rely on pandas, Python's data science library. Please check https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html for a quick introduction to Pandas.

Pandas relies on a type of data structure known as the DataFrame. An example DataFrame, created from reading a .csv file of data, is shown below. DataFrames have an index (the leftmost column in the image below), and data columns. Each column has a name, and an index can also be named. Indexes, however, sometimes behave differently from data columns. Columns can become indexes, and vice versa. For example, the DataFrame below can be modified so that the `Timestamp` column becomes an index, and drop the `Unnamed: 0` column. Then the DataFrame would be indexed by the `Timestamp` values, which is useful if, for example, we want to order the values by time.



```
        Unnamed: 0           Timestamp  Value      Name  MeasurementId  SensorName  Position      Timestamp_10min  TT_10
0                2 2021-04-27 18:17:48  19.62  BW VL WW              7  1073486056         2  2021-04-27 18:20:00   10.0
1                4 2021-04-27 18:18:00  19.62  BW VL WW              7  1073486056         2  2021-04-27 18:20:00   10.0
2                8 2021-04-27 18:18:12  19.56  BW VL WW              7  1073486056         2  2021-04-27 18:20:00   10.0
3               15 2021-04-27 18:18:23  19.56  BW VL WW              7  1073486056         2  2021-04-27 18:20:00   10.0
4               16 2021-04-27 18:18:35  19.56  BW VL WW              7  1073486056         2  2021-04-27 18:20:00   10.0
...            ...                 ...    ...       ...            ...         ...       ...                  ...    ...
378685     3019988 2021-08-24 11:06:35  21.19  BW VL WW              7  1073486056         2  2021-08-24 11:10:00   19.2
378686     3019997 2021-08-24 11:06:59  21.19  BW VL WW              7  1073486056         2  2021-08-24 11:10:00   19.2
378687     3020007 2021-08-24 11:07:17  21.19  BW VL WW              7  1073486056         2  2021-08-24 11:10:00   19.2
378688     3020013 2021-08-24 11:07:40  21.25  BW VL WW              7  1073486056         2  2021-08-24 11:10:00   19.2
378689     3020018 2021-08-24 11:07:58  21.25  BW VL WW              7  1073486056         2  2021-08-24 11:10:00   19.2
```

Figure 1: A DataFrame, printed out

Data collection and refining

Home heating system data

The home heating system data was provided to me in the form of .csv files.

Weather data

The weather data was collected from the Open Data Server of the Deutsche Wetterdienst (DWD) in the form of .txt files at https://opendata.dwd.de/. The relevant weather data for the project was the temperature measurements at Soltau station (recorded every 10 minutes with precision of 0.1 degree Celsius) and solar insolation for the nearest stations that recorded that data. The specific directory is https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/10_minutes/.

Data refining

The whole data is 400 MB, but most of the files only need a subset of that data (e.g. only the BW VL WW measurements). So, the `csvMaker` file outputs a file (and saves it in the proper spot) that contains only a subset of the data and merges the heating system data with the DWD data. Most files then read that smaller file and work with the smaller file instead of the massive raw data file.

Boiler operation analysis

We wanted to see how much the boiler is operating during certain conditions. Since we couldn't look at the boiler itself, we instead used temperature measurements as proxies. If one value (e.g. BW VL HZG) increased by at least X degrees Celsius within Y seconds, then we add that to "boiler uptime"; otherwise, we don't. Then the uptime ratio = uptime/total time. Multiple files do different things, all of them of some use: check the individual files in the relevant folder for more information.

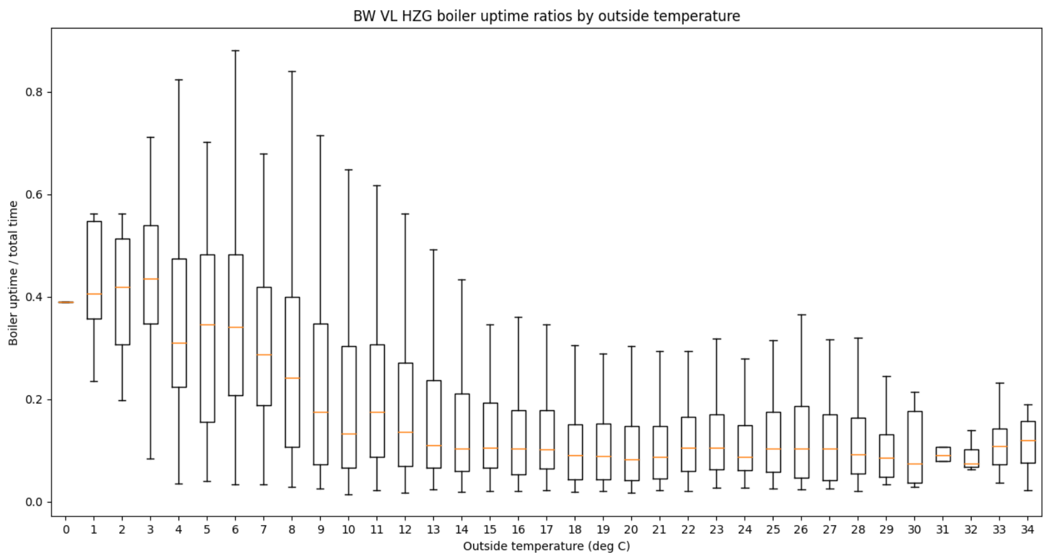


Figure 2: Boiler uptime ratios. This plot is for outside temperatures. Outside temp of 1 deg C, median boiler uptime of 0.4

Insolation analysis

We wanted to see whether measurements of solar insolation (how sunny it is) changed drastically from station to station. We found that it did: scatterplots showed that Soltau and Hannover had very different solar insolation measurements at the same time. This meant that Soltau solar insolation temperatures cannot be used for the home location, as the home will likely have different solar insolation values. So, if we want to see solar insolation, we need to have a meter put in place on location itself.

Plots

The `matplotlib.pyplot` and `seaborn` libraries (imported as `plt` and `sns` respectively) were used to make plots.

Box plots

A lot, and I mean *a lot*, of boxplots were created. Probably too many boxplots. See https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html for documentation.

Heatmaps

The heatmap files read a certain file and then create a heatmap organizing the values for each hour of the day and each day of the week.

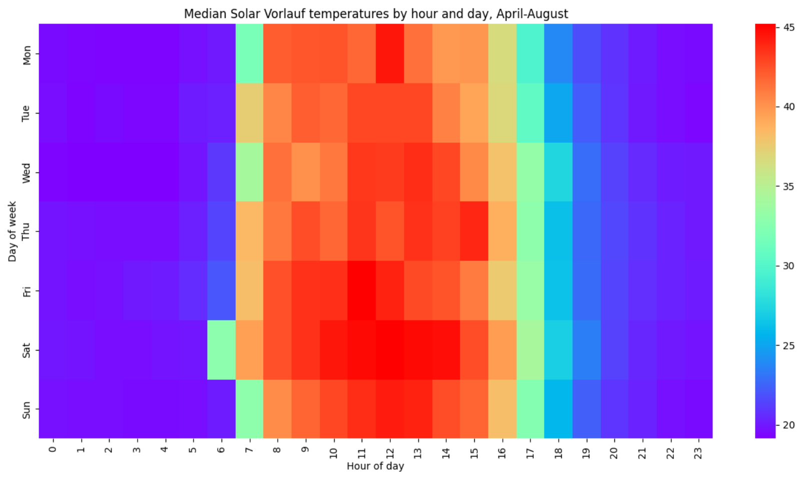


Figure 3: An example of a heatmap for Solar Vorlauf. Naturally, the values are higher during daylight hours.

Histograms

Histograms were created to give a preliminary view into the data. See https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html for more information.

Scatterplots

Multiple files create scatterplots using the `regplot` function of the `seaborn` module. See https://seaborn.pydata.org/generated/seaborn.regplot.html for documentation.

Time series

The `plt.plot` function plots y vs x as lines and/or markers. In our case, x (horizontal axis) is time and y (vertical axis) is temperature value. See https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html for documentation.

Temperature spike analysis

The temperature spike files all do the following things:

1. reads one or two files of temp parameters (e.g. BW VL WW) within certain hours of the day

2. divides the data into periods (e.g. 12 minutes)

3. checks each period, ignoring periods without valid values (e.g. ignores periods where BW VL WW does not exceed 50 degrees C)

4. then for each valid period, it takes values in and around that period and puts it into bins of a certain period of time (e.g. 30 second bins), and appends it to a list. In this step, the values might be filtered again, depending on the file.

5. That list of lists is then converted to a pandas DataFrame which is then used to create plots (mostly boxplots) that show the range of values (such as BW VL WW or VL HZG) relative to the time at which a value peaks (usually BW VL WW)

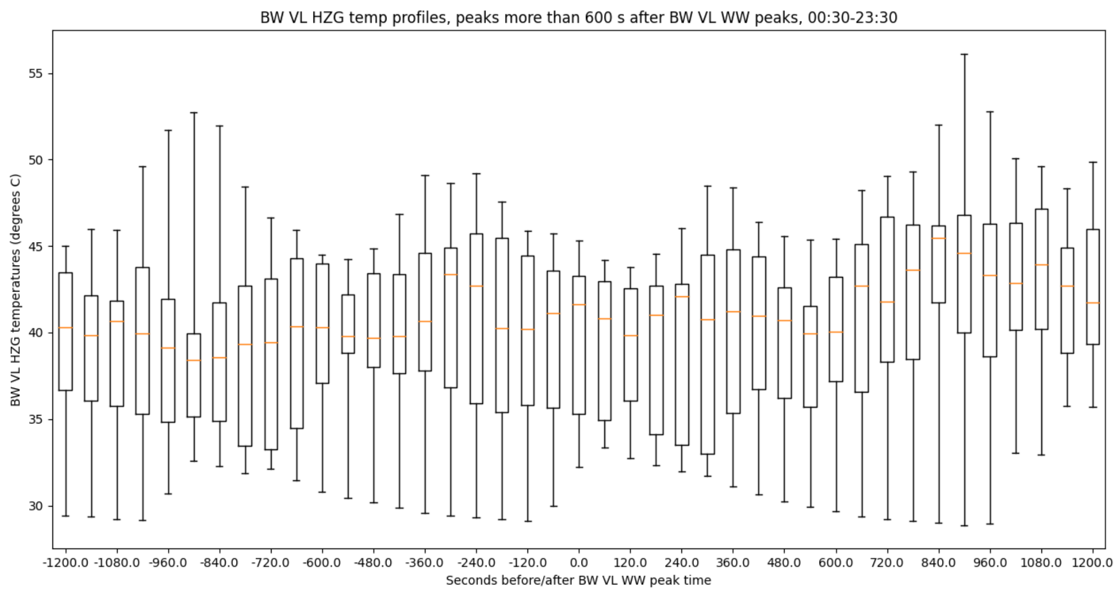


Figure 4: An example of the output of the temp spike analysis files.

Time series analysis

I tried to train a machine learning model to predict temperature values based on system parameters and other things like outside temperature, but I didn't know what I was doing and completely failed. You can see my failed efforts in the `time_series_analysis` folder. Hopefully you'll have better luck.

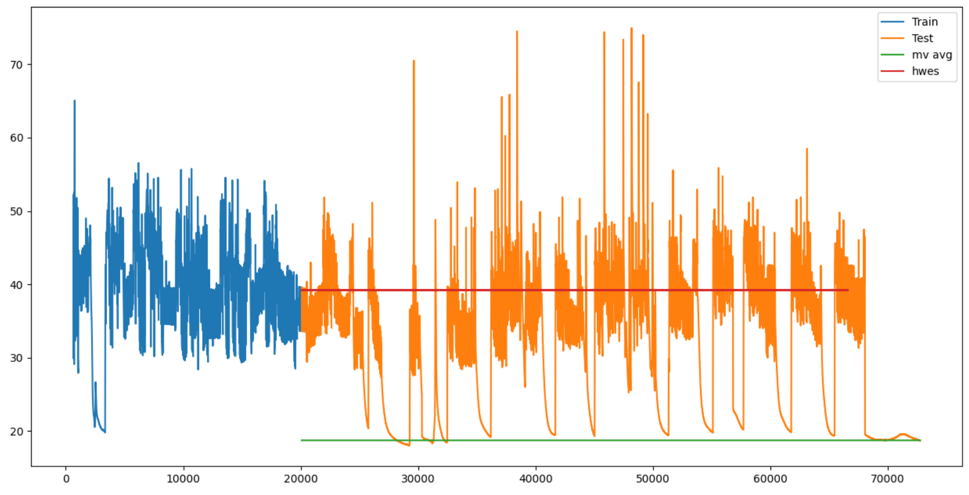


Figure 5: The green and red lines indicate failure.

Possible improvements

Many improvements could be made to the programs. As the focus of the project was to do as much analysis of the given data as possible, without concern for writing code that performs well or can be integrated easily into other applications, the code was written in a “quick and dirty” way that meant that bad practices were followed. Input values that were for filtering the data were embedded in the program itself, and so changing the data filter parameters would require changing that line in the code, which is poor practice. Performance is also poor, with the longer programs taking multiple minutes to run. This may be because I used many `for` loops in the code and similar non-vector solutions, even though pandas is designed to work with vectorization (similar to other languages like MATLAB).

If you need the performance to be better, or want to integrate the code into a larger system, I suggest that you modify the code in the following ways:

1. Have the input parameters for each file be stored in a .csv file, or similar, that the Python file then reads. This will make it much easier to reconfigure the input parameters, and the input parameters will be in the same place and thus the program design will be more clear.
2. Reconfigure the code so that instead of `for` loops, the code uses a more efficient solution like a list comprehension or the pandas `.apply` method.
3. Reconfigure the code so that portions of the code that are nearly (or completely) identical across multiple files (like the data filtering and resampling of the tempSpikes files) are instead written in one module that is shared across multiple files, so that modifications in functionality can be consistent and the code structure makes more sense, making refactoring easier. I’ve done this with the `merger.py` file, which is used as a module by several other files, but I should have done more of that.
4. Accessibility might be a concern. People who have poor sight will find it hard to look at all the charts and graphs. But if that is a concern, what you could do is just change the parts where you output a plot and create a sanitized and user-friendly table to show the data instead.
5. Most of the files create figures, but those figures have to be saved manually. I did not save files automatically because I did not want to overwrite anything. (Though now that I think about it, I could just autosave and use the current time in the filename, so I wouldn’t overwrite anything…) So, automatically saving figures is a potential extra feature.