Kevin Paulovici

Bellevue University

DSC 540 Data Preparation

Final Project

For the final project, I chose to use the <https://openweathermap.org/api>. I initially selected this because I thought it would be interesting to integrate the API calls (i.e., data gathering) into some GUI. However, due to time constraints this did not makes its way into this project. I’ll touch more on that in the future work discussion. This summary will discuss the process used to collect data, transformations and data cleaning methods, the relevance of this, and future work.

Before starting any programming, it is a good idea to get familiar with the API functions and data gathering possibilities. From here, I simplified the process by setting a few assumptions such as limiting the scope to US zip codes and imperial units. Ideally, this could accept various countries or cities, and provide an option for the units. Since this project requires a zip code, I allowed the user to enter a value of their choice with some error checking to verify a valid value is provided. This could be changed to multiple values or a static value depending on the needs. The next step is to build the url based on the zip code, and some other assumptions, like units, and the API key. Using the requests library and our url, we can retrieve our data in a json format. From here, we’d have to investigate the returned data and determine what values are to be include.

Once we know and understand what data we want, we can begin parsing it to a new dictionary and perform transformations/cleaning as we see fit. The two transformations/cleaning steps I performed include correcting the date/time stamp and fixing the headers. The date/time stamp provide is in a Unix/Linux format and is useless to the user. Using the datetime library this value can be converted into a useable time and the date can be added as a new value to our dictionary. For a Pythonic method of setting this dictionary, the keys used were based on the json keys. This is not ideal since they are less descriptive. By transforming the headers we can result in a new dictionary with cleaner key values (See Table 1 for example output).

The value of creating something to collect weather data really comes when you expand the scope of the data collected. If you want data for dozens of areas, you can set up a list and continuously perform the actions described above and output everything neatly to a single file. This would be idea for someone tracking weather over a long period of time. Additionally, this could be run daily (e.g., using cron) without the user having to do anything after the initial setup.

As I mentioned, for simplicity, the scope of this project only takes a single input. To expand on this, the user could create a list of values to request data for and set an automated routine to run on a schedule. Alternatively, this project could be implemented using a GUI depending on the user’s needs. The GUI could accept lists of data to request or allow the user set the output parameters. A sample GUI is provided in Figure 1.

Table 1 – Example Output

|  |
| --- |
| City,Schenectady  Country,US  Country\_id,5782  Sunrise,11:31:13  Sunset,22:45:31  Pressure\_(hPa),1020.0  Temperature\_Feels\_like\_(F),12.02  Temperature\_(F),26.15  Min\_Temperature\_(F),21.0  Max\_Temperature\_(F),30.0  Humidity\_(%),53.0  Weather\_Description,scattered clouds  Wind\_Speed\_(miles/hr),14.99  Latitude,42.88  Longitude,-73.99 |

Figure 1 – Example GUI for Weather App

