NOAA & OpenWeatherMap weather data analysis

Tianye Zhao
CS504 Supplementary Project

Goals

1. Load station and temperature data from publicly available text file from NOAA;
2. Integrate missing data, smooth, and plot temperature data;
3. Compute daily records at given location in specific year;
4. Compare warmest year with coldest year of one location.

The steps for processing the data are as follows, here are only main objectives on what to be done in analysis. Clear comments to explain the necessary procedure are in source code.

Load station data

1. Download file over FTP (http://www.ncdc.noaa.gov);
2. Parse space-separated text file into Python dict.

After using *readlines* and slicing to select few items in the stations list, we know a station code in each line followed by geographical location (longitude and latitude), sea level height in meters and by station name. Some stations are tagged as GSN. That's the GCOS Surface Network. This project concentrates on these because only weather data of stations which marked as GSN can be downloaded from NOAA FTP.

Load temperature data

1. Parse fixed-field text file using *np.genfromtxt*;
2. Use ranges of numpy *datatime* objects.

The format of this file is rather uninspiring. It begins with the station code, followed by the date, year, and month, by the name of a weather observable, such as temperature maximum, or temperature minimum, and by a bunch of numbers that represent the observable over the days of the month. The format is described in detail in the readme.txt file in the directory under NOAA FTP.

Integrate missing data

1. Use numpy boolean mask indexing;
2. Interpolate one-dimensional arrays with *numpy.interp*.

After grab the minimum and maximum temperature and plot them, it's noticeable that the -999.9 value associated with missing observations. It's better to change them to more representative format such as *numpy.nan*. When take mean of the series, it will get *nan* because the mean of actual numbers with a *nan* value is not a number. A better way to fill in those missing temperatures is replacing missing values by interpolating the values of the neighbors.

Smooth data

1. Smooth data with running mean;
2. Show plots with *matplotlib*.

The temperature series are still noisy as it's basically a big block of pixels. To see trends on longer time scales, it needs to average out short-term fluctuations which is smoothing short-term oscillations by averaging other nearby values.

Compute daily records

1. Combine boolean masks with logical operators;
2. Compute *max* and *min* across array rows;
3. Plot shaded area with *matplotlib*.

Here it plots temperature data for a selected year, and the historical records which is the most extreme temperatures achieved on that day of the year across all available years.

Compute extreme records

1. Find the year with the highest mean *TMAX*;
2. Find the year with the lowest mean *TMIN*;
3. Plot *TMAX* and *TMIN* throughout those years in same plot.

OpenWeatherMap API

This part uses OpenWeatherMap API to collect weather data from main cities in Canada. First, get OpenWeatherMap API key. Run query for Sherbrooke for example, it returns a JSON string for 5 days / 3 hours forecast. The dictionary contains the humidity, the pressure, the temperature, the name of city, the country of city, and the date time which is the time of the data of the weather information was forecasted (in Unix time stamp format).