"Pollution in Graz"

Homework Assignment 2

November 7, 2018

Original data: https://luftdaten.info/

1 Tasks (20pt)

Download the assignment files[1] to your local system. The ZIP archive contains two files. The first file is the dataset (`sensors_graz.csv`) and the second is the Python file that you can test your code with (`ass_2_plot.py`). Make sure to **not** include these two files when you upload your assignment!

1.1 Command Line Parameters (3p)

Your program should be able to work with a variety of different command line parameters. Use the `argparse` package[2] to implement this functionality. Implement the following parameters:

`-i --input:` Path to the input file (the dataset)

- Data type: String
- Optional: No

`-o --output:` Path to the output file (the corrected dataset)

- Data type: String
- Optional: No

`-s --sensors:` Name of the desired sensors

- Data type: List of integers
- Optional: Yes
- Default value: `[ ]` (empty list)

`-r --resampling_delta:` Size of the time steps when resampling (in seconds)

- Data type: Integer
- Optional: Yes
- Default value: `600` (10 Minutes)

`-w --window_smoothing:` Size of the time window when smoothing (in seconds)

- Data type: Integer
- Optional: Yes
- Default value: `None`

[1] https://github.com/pkasper/info1-bm/raw/master/2018/assignments/assignment_2_files.zip
[2] https://docs.python.org/3.7/library/argparse.html

1.2 Basic I/O Checks (2p)

Check to make sure that the file related parameters are valid. Implement the following checks:

(i) Verify that the input data exists and is indeed a file.
(ii) Verify that the desired output path exists.
(iii) Verify that the output file ends in `.csv`.

Should one of the above tests not be passed provide an error message and end the program with `exit()`. Alternatively you could also raise an Exception[3].

1.3 Dataset Parsing and & Preparation(6p)

1.3.1 Parsing

The dataset consists of a precorrected CSV file. This means that eventual errors have been accounted for and removed. The international standard by which data is separated in this file format is the comma (`,`). CSV files are simple coded tabels (found in Excel and other similar programs that you may have used). In the first row you will find the labels for each column. This means that you have to read this row separately from the rest of the data.

In the dataset you will find the following header (columns):

(i) **`sensor_id` (integer)** Sensor Identifier.
(ii) **`sensor_type` (string)** Type of sensor. Always SDS011
(iii) **`location` (integer)** Location Identifier of the sensor.
(iv) **`lat` (float)** Latitude Coordinate
(v) **`lon` (float)** Longitude Coordinate.
(vi) **`timestamp` (numpy.datetime64)** Timestamp of the entry
(vii) **`P1` (float)** First sensor value (PM_{10})
(viii) **`P2` (float)** Second sensor value ($PM_{2.5}$)

Every row (after the header) contains the measured values of one sensor at one exact time point. Work your way row by row and save the values. The command line parameter `--sensors` is to be used as a filter. Skip the row of a certain sensor (ID) that you explicitly do not want to save. Should this parameter not be accounted for or in other words be the default value `[ ]`, then save all of the sensors.

Internal data structure. For this assignment there is no predefined internal data structure. You may however use the following template:

```
sensors = dict()

sensors[sensor_id] = dict()
sensors[sensor_id]['sensor_id'] =  #  Sensor ID
sensors[sensor_id]['lat'] =  # lat (latitude) Coordinate
sensors[sensor_id]['lon'] =  # lat (longitude) Coordinate
sensors[sensor_id]['sensor_type'] =  #  Type of Sensor (should always be SDS011)
sensors[sensor_id]['data'] = dict()  #  Dictionary for measured datapoints
sensors[sensor_id]['data']['timestamp'] = []  #  Timestamps
sensors[sensor_id]['data']['P1'] = []  #  Values in P1
sensors[sensor_id]['data']['P2'] = []  #  Values in P2
```

Check always to make sure that the sensor of the current row is already in your dictionary (in this case `sensors`). Otherwise you have to make a new entry for this sensor.

[3] https://docs.python.org/3/tutorial/errors.html

1.3.2 Checking and Preparing

When you have successfully parsed the dataset run a quick check. This check serves to make sure that for every sensor there has an exactly equal number of values for `timestamp`, `P1`, und `P2` saved. Should this not be the case then provide an appropriate error message and end the program.

For the next preparation step you have to verify that the order of the values for each sensor is chronologically correct. The `numpy.datetime64` data type can be sorted just like any integer. However, what in this case is necessary is the order of the indices and not the actual values (since P1 and P2 have to be sorted in the same order). You can use the following trick to get the desire indices:

1. Create a list with increasing numbers (e.g., via `range()`) with the same number of entries as the number of `timestamp` entries.
2. Sort this created list (via `sort()` or `sorted()`). This can be done by providing the `__getitem__` function of the `timestamp` list in the sort function. Then you receive the list of numbers sorted with regards to the timestamps, which in turn corresponds to the indices.
3. Reorder `timestamp`, `P1`, `P2` with regards to the indices. This can be achieved quite elegantly using list comprehensions.

Example for Creating Indices:

```
sort_order = sorted(<indices_list>, key=<timestamp_list>.__getitem__)
```

Hint: The trick for creating indices matches the functionality of `numpy.argsort()`[4]. You can use this function to make sure that you have programmed this step correctly.

1.4 Cleaning & Smoothing (7p)

Before starting this task create a copy of your data structure. To acccomplish this use `copy.deepcopy()`[5]. For the following tasks work with this copy. By making this copy you can compare the two data structures (the original and the cleaned one).

1.4.1 Removing Outliers

For each sensor loop over every list of values for `P1` and `P2`. While looping compare the current value with the previous and the next value. If the current value is larger than the sum of the previous and next value, then mark it as an outlier. For each outlier, calculate the mean of the preceding and succeeding values and replace the outlier with this new value.

1.4.2 Resampling

If you look at the values in `timestamp`, then you will see that the time intervals between the individual measurements are not constant and consistent between the different sensors. Therefore, resample all the sensors with the same time interval provided by the command line parameter `--resampling_delta`.

Calculate for each time window the median of the measured values (once for `P1` and once for `P2`). Start every time window at 00:00 for each day and place the first data entry for each sensor here. Overwrite the previous data with the new timestamps and values. Should there be no entries in a specific time window (e.g. due to sensors being offline), then save `None` as a value for this entry.

Hint: Should the median be two values, then calculate the mean of the larger and smaller neighbor. For example: The median for $[1, 2, 3, 4]$ is 2.5.

Tip: Read the documentation of the numpy datetime and timedelta datatypes [6]

[4] https://docs.scipy.org/doc/numpy-1.15.4/reference/generated/numpy.argsort.html?highlight=argsort#numpy.argsort
[5] https://docs.python.org/3.7/library/copy.html
[6] https://docs.scipy.org/doc/numpy-1.15.0/reference/arrays.datetime.html

1.4.3 Rolling Means

In order to detect trends and long term characteristics, calculate the mean for each rolling window individually for each sensor. The size of these windows is determined by the parameter `--window-smoothing`. For each entry consider all neighbors that are either more or less than ($\leq$) *window_smoothing*/2 separated from one another. Neighbors with `None` as values should be ignored. For each window, calculate the mean (arithmetic mean). If the parameter has not been used then you do not have to smooth the data!
Hint: The parameter `--window-smoothing` is defined in seconds. Therefore, you have to by yourself calculate how many values you should consider (division with the `--resampling_delta` Parameter)

Example Rolling Mean:
(In this example, the size of the window was precalculated.)

```
example_list = [2, 2, 4, 4, 6, 6, 8, 8]
window_size = 4 # half = 2

#1st window: [2, 2, 4, 4, 6] for index 2
#2nd window: [2, 4, 4, 6, 6] for index 3
#3rd window: [4, 4, 6, 6, 8] for index 4
#4th window: [4, 6, 6, 8, 8] for index 5

rolling_means = [3.6, 4.4, 5.6, 6.4]  #  we calculate the mean for each window
```

For each entry where your window would extend past the end of the list do not calculate the mean.
Hint: Since you changed all the sensor values to a static interval in the last step, you only have to divide the parameter with the interval to obtain the number of preceding and succeeding values that you should consider.

1.5 Saving & Testing (2p)

1.5.1 Saving

Save the cleaned and smoothed data in a new csv file. This should have the same structure as your input file. Additionally, the field `location` isn't needed anymore and should not be in the csv file! The file name and path should be defined by the command line parameter (`--output`).

1.5.2 Testing

Import the file `ass_2_plot.py`. Make sure that the file is saved in the same folder as your script. The import provides the function `plot_data(title, data_orig=None, data_smooth=None, filename=None)`. For `data_orig` and `data_smooth` the function expects a dictionary with the keys `P1` and `P2` that contain the sensor values and `timestamp` with the corresponding timestamps. (See `data` dictionary in the example data structure) `data_orig` provides the values before the all the cleaning steps (for this reason we copied all the values beforehand). `data_smooth` contains the values after all the cleaning and smoothing steps. When you provide the `filename` parameter, then the function saves an image in the desired location. You can compare this output to the reference output provided in the Wiki.

Create an image for each desired sensor (defined by `--sensors` command line parameter). The file name should match the name of the output file (`--output`) and have the suffix `_<sensor_id>`. For the file ending use `.png`

Output Example:
Test:

```
> assignment_2.py --input=sensors_graz.csv --output=assignment_2.csv --sensors 1503 1693
```

Created files:

- `assignment_2.csv`
- `assignment_2_1503.png`
- `assignment_2_1693.png`

Hint: The script expects, that you have the seaborn package installed. This is automatically provided with anaconda. Should you however not have it installed then use the command `pip install seaborn`.

Tipp: Compare your images with the reference in the Wiki[7].

1.6 Bonus: Rolling Window Edges (2p)

In Task 1.4.3 no mean values were calculated when the window extended beyond the end of our time interval. In the visualization we can see this clearly as sharp oscillations at the edges. Think of a method how you can also smooth the values at the edges (e.g., by shrinking the windows in the appropriate direction).

Change the implementation of the previous rolling windows accordingly.

Important: Mention in the comment header that you have attempted this bonus task!

2 Restrictions

- Do not add any bloat to your submission
- Use built-in functions where possible
- Only use/import explicitly allowed packages
- Do not use any command line arguments
  - You may use extra parameters. However, your program should also work without their use!

2.1 Allowed Packages

- `argparse`
- `copy`
- `os`
- `sys`
- `numpy`

- `ass_2_plot`

[7] https://palme.iicm.tugraz.at/wiki/Info1BM

3 File Headers

All Python source files have to contain a comment header with the following information:

- Author: – Your name
- MatNr: – Your matriculate number
- Description: – Purpose of the file
- Comments: – Any comments as to why if you deviate from the task or restrictions

Please copy and paste the example and change its contents to your data. (Depending on your PDF-viewer you may have to fix the whitespaces!)

Example Header:

```
########################################################################
# Author:      [FIRST NAME] [LAST NAME]
# MatNr:       [MAT NR]
# Description: [SHORTDESCRIPTION]
# Comments:    [ANY RELEVANT COMMENTS.
#              CAN BE MULTILINE]
########################################################################
```

4 Coding Standard

This lecture follows the official PEP 8 Standard[8]. it defines basic formalities as to how your code should look like. In particular, please look at the following aspects:

Language. Programming is done in English. This is to ensure someone else at the other end of the world can read your code. Please make sure all sources you submit are thus written in English. This covers both variable names and comments!

Spaces, not tabs. Indent your files with 4 spaces instead of tabs. PEP 8 forces this in Python 3 (whilst allowing more freedom in Python 2). Most editors have an option to indent with 4 spaces when you press the tab button!

Descriptive names. Use descriptive names for variables where possible! Whilst a simple i can be enough for a simple single loop code can become very messy really fast. If a variable has no purpose at all, use a single underscore (_) for its name. Should you be uncertain if a name is descriptive enough, simply as a comment describing the variable.

120 **characters line-limit** PEP 8 suggests a maximum line length of 79 characters. A different established limit proposes 120 characters. In this lecture, we thus use the wider limit to allow for more freedom. The maximum number of characters is 120 including indentations! Note that this is also applies to comments!

[8] https://www.python.org/dev/peps/pep-0008/

5 Automated Tests

Your code will be tested by an automated program.Please make sure your submission follows all the restrictions defined in this document. In particular concentrate on the predefined names for functions and variables!
If your submission fails any of the automated tests, we will look into your code to ensure the reason was not on our end and to evaluate which parts of your code are correct and awards points accordingly.

6 Submission

6.1 Deadline

4^{th} of December 2018 at 23:59:59.

Any submission handed in too late will be ignored with the obvious exception of emergencies. In case the submission system is globally unreachable at the deadline it will be pushed back for 24 hours. If for whatever reason you are unable to submit your work, contact your tutor *PRIOR* to the deadline.

6.2 Uploading your submission

Assignment submissions in this course will always be archives. You are allowed to use .zip or .tar.gz formats.
In addition to your Python source files, please also pack a *readme.txt*. The file is mandatory but its contents are optional. In this file please write down how much time you spent on the assignment and where you ran into issues. Your feedback allows for immediate adjustments to the lecture and tutor sessions.
Upload your work to the Palme website. Before you do please double-check the following aspects:

- File names and folder structure (see below)
- Comment headers in every source code file
- Coding standard upheld

6.3 Your submission file

```
assignment_2.zip (or assignment_2.tar.gz)
    assignment_2.py
    readme.txt
```

The input files (the dataset + `ass_2_plot.py`) should NOT be uploaded!