Demo

July 30, 2021

1 An instruction on how to use this tool

1.1 Data requirements

- Only csv and excel files are supported
- Make sure that the data include columns named 'waterelevel' and 'time'
- Make sure that the time includes year, month and date

1.2 Environment requirements

- Python libraries:Numpy, pandas and matplotlib are needed

1.3 Instruction

- First running the file
- Input the path of data
- check whether the first fitering is satisfying. Input 1 if you don't want second fiteration
- check the output statistics and plot. The statistics are saved as two excel files

[5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import timedelta
from pandas.plotting import register_matplotlib_converters
from datetime import datetime

def readtemplate(path):
    filename = path[-4:]
    if filename == '.csv':
        data = pd.read_csv(path, nrows=14000)
    elif filename == 'xlsx':
        data = pd.read_excel(path, nrows=14000)
    return data


def cal_sd(df):
    drop_vec = []
    coverage_time = 4  # refer to 4 time span of the time unit related to the␣
 ↪data
```

```
    downtemp = 0  # record whether decreasing happened
    uptemp = 0  # record whether increasing happened
    smooth_count = 0
    for i in range(len(df['waterlevel']) - 1):
        currvalue = df.at[i, 'waterlevel']
        gradient = df.at[i + 1, 'waterlevel'] - currvalue
        if gradient > 0:
            if uptemp == 0:
                startpoint = currvalue
                uptemp = 1
            elif smooth_count > coverage_time:
                startpoint = currvalue
                smooth_count = 0
            else:
                smooth_count = 0
        elif gradient == 0 and uptemp == 1:
            smooth_count += 1
        elif gradient < 0 and uptemp == 1:
            top = currvalue
            drop_vec.append(top - startpoint)
            uptemp = 0
            smooth_count = 0
    sd = np.sqrt(np.std(drop_vec))
    return sd


'''
df:    dataset
x1:    /             (tolerance for the max time interval in one single␣
↪incrasing/decreasing. If the time is beyond the tolerance, we replace the␣
↪old one with the new one)
x2:       /           (tolerance for the max time interval between one␣
↪hydropeak.)
x3:                   (tolerance for the max drop between one hydropeak)
x5:                (tolerance for the max gradient to be detected)
'''


def filterhydro(df, sd):
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.plot(df['time'], df['waterlevel'], 'k--', label='Before')
    global time1, time2, bottom, top, uptemp, downtemp, gradient1, gradient2,␣
↪timestart, timeend, valuestart, valueend, smooth_count
    # Initialize the data
    cov_time = 4  # refers to 4 interval of time related to the data
    downtemp = 0  # record whether decreasing happened
```

```python
    uptemp = 0  # record whether increasing happened
    start_val, top, end_val = 0, 0, 0
    time1, time2 = 0, 0
    smooth_count = 0
    lock = 0
    fit_count = 0
    # filterfunction
    for i in range(len(df['waterlevel'])-1):
        currtime = i
        currvalue = df.at[i, 'waterlevel']
        # set the gradient
        gradient = df.at[i+1, 'waterlevel'] - currvalue
        #
        # When an increasing gradient is detected, we want to find whether there is decreasing gradient nearby. We record the time interval and waterlevel difference
        if uptemp != 1 or downtemp != 1:
            if gradient > 0:
                if uptemp == 0:
                    time1 = currtime
                    start_val = currvalue
                    uptemp = 1
                elif smooth_count > cov_time:
                    time1 = currtime
                    start_val = currvalue
                    smooth_count = 0
                else:
                    smooth_count = 0
            elif gradient == 0 and uptemp == 1:
                smooth_count += 1
            elif gradient < 0 and uptemp == 1:
                top = currvalue
                time2 = i + 1
                end_val = df.at[i + 1, 'waterlevel']
                downtemp = 1
        # match
        elif downtemp == 1 and uptemp == 1:
            if gradient > 0:
                leftdrop = top - start_val
                rightdrop = top - end_val
                if leftdrop <= sd and rightdrop <= sd:
                    fillnum = (end_val - start_val) / (time2 - time1)
                    for j in range(time1, time2 + 1):
                        df.at[j, 'waterlevel'] = (
                            j - time1) * fillnum + start_val
                    uptemp = 0
                    downtmp = 0
```

```
                    lock = 0
                    fit_count += 1
                elif rightdrop >= sd and leftdrop < sd:
                    for j in range(time1, time2 + 1):
                        if df.at[j, 'waterlevel'] >= start_val:
                            df.at[j, 'waterlevel'] = start_val
                        else:
                            break
                    downtemp = 0
                    uptemp = 0
                    lock = 0
                    fit_count += 1
                elif leftdrop >= sd and rightdrop < sd:
                    for j in range(time2, time1 - 1, -1):
                        if df.at[j, 'waterlevel'] >= end_val:
                            df.at[j, 'waterlevel'] = end_val
                        else:
                            break

                    downtemp = 0
                    fit_count += 1
                    time1 = currtime
                    start_val = currvalue
                    uptemp = 1
                lock = 0
                smooth_count = 0
                time1 = currtime
                start_val = currvalue
                uptemp = 1
                downtemp = 0
            elif gradient == 0:
                smooth_count += 1  # record the times of no change
            elif gradient < 0:
                if smooth_count > cov_time:
                    lock = 1  # lock the record of time and value if there are more than 4 times with no change
                if lock != 1:
                    time2 = currtime + 1
                    end_val = df.at[i + 1, 'waterlevel']
                    smooth_count = 0
    print('We have filtered', fit_count, 'flowing within the sd')
    ax.set_title('The waterlevel before and after filtering')
    ax.set_xlabel('time')
    ax.set_ylabel('water level')
    ax.plot(df['time'], df['waterlevel'], label='After')
    ax.legend(loc='best')
    plt.show()
```

```
#    return df


def filterhydro_plot(df, sd):
    ctn = 0
    while True:
        filterhydro(df, sd)
        ctn = input('Does the filter look plausible? If yes, input 1:\n')
        if ctn == '1':
            break

def datetime_preprocess(df):
    df['time'] = pd.to_datetime(df['time'])
    df1 = df.set_index('time')
    df1['num'] = 1
    day = df1.to_period('D')
    date_num = df1.resample('D').sum()
    date_list = list(x.strftime('%d-%m-%Y') for x in date_num.index)
    date_num = list(date_num['num'])
    return date_num, date_list

def searchhydro(df, sd):
    global time1, time2, bottom, top, uptemp, downtemp, timestart, timeend,␣
↪valuestart, valueend, smooth_count
    # Initialize the data
    cov_time = 4  # refers to 4 interval of time related to the data
    hydropeak_num = 0
    downtemp = 0  # record whether decreasing happened
    uptemp = 0  # record whether increasing happened
    bottom, top = 0, 0
    time1, time2 = 0, 0
    timestart = []
    timeend = []
    valuestart = []
    valueend = []
    smooth_count = 0
    rightdrop = []
    leftdrop = []
    rightslope = []
    leftslope = []
    peak_duration = []
    peak_duration_count = 0
    smooth_count = 0

    #import the time series
    date_num, date_list = datetime_preprocess(df)
    last_hydronum = 0
```

```
    last_day = 0
    day_count = 0
    hydronum_daily = {'number':[],'date':[]}
    for i in range(len(df['waterlevel'])-1):
        #append the hydronum
        if i-last_day == date_num[day_count]-1:
            hydronum_daily['number'].append(hydropeak_num-last_hydronum)
            hydronum_daily['date'].append(date_list[day_count])
            last_hydronum = hydropeak_num
            last_day = i
            if day_count < len(date_list)-1:
                day_count += 1
        elif i == len(df['waterlevel'])-1:
            hydronum_daily['number'].append(hydropeak_num-last_hydronum)
            hydronum_daily['date'].append(date_list[day_count])
            last_hydronum = hydropeak_num
        #start searching hydronum
        currtime = i
        currvalue = df.at[i, 'waterlevel']
        # set the gradient
        gradient = df.at[i+1, 'waterlevel'] - currvalue
        #
        # When an increasing gradient is detected, we want to find whether␣
↪there is decreasing gradient nearby. We record the time interval and␣
↪waterlevel difference
        if uptemp != 1 or downtemp != 1:
            if gradient > 0:
                if uptemp == 0:
                    time1 = currtime
                    start_val = currvalue
                    uptemp = 1
                    peak_duration_count = 0
                elif smooth_count > cov_time and uptemp == 1:
                    time1 = currtime
                    start_val = currvalue
                    smooth_count = 0
                    peak_duration_count = 0
                else:
                    smooth_count = 0
                    peak_duration_count = 0
            elif gradient == 0 and uptemp == 1:
                smooth_count += 1
                peak_duration_count += 1
            elif gradient < 0 and uptemp == 1:
                top = currvalue
                toptime = i
                time2 = currtime + 1
```

```python
                end_val = df.at[i + 1, 'waterlevel']
                downtemp = 1
                smooth_count = 0
                peak_duration_count = 0
            # match

        elif downtemp == 1 and uptemp == 1:
            if gradient > 0:
                hydropeak_num += 1
                start_time = df.at[time1, 'time']
                end_time = df.at[time2, 'time']
                top_time = df.at[toptime, 'time']
                timestart.append(start_time)
                timeend.append(end_time)
                valuestart.append(start_val)
                valueend.append(end_val)
                rttime = to_integer(top_time - start_time)
                lttime = to_integer(end_time - top_time)
                rightdrop.append(top - start_val)
                leftdrop.append(top - end_val)
                rightslope.append((top - end_val)/rttime)
                leftslope.append((top - end_val)/lttime)
                peak_duration.append(peak_duration_count)
                downtemp = 0
                time1 = currtime
                start_val = currvalue
                smooth_count = 0
            elif gradient == 0:
                smooth_count += 1
            elif gradient < 0:
                if smooth_count > cov_time:
                    hydropeak_num += 1
                    start_time = df.at[time1, 'time']
                    end_time = df.at[time2, 'time']
                    top_time = df.at[toptime, 'time']
                    timestart.append(start_time)
                    timeend.append(end_time)
                    valuestart.append(start_val)
                    valueend.append(end_val)
                    rttime = to_integer(top_time - start_time)
                    lttime = to_integer(end_time - top_time)
                    rightdrop.append(top - start_val)
                    leftdrop.append(top - end_val)
                    rightslope.append((top - end_val)/rttime)
                    leftslope.append((top - end_val)/lttime)
                    peak_duration.append(peak_duration_count)
                    uptemp = 0
```

```
                downtemp = 0
            else:
                end_val = df.at[i + 1, 'waterlevel']
                time2 = currtime + 1
                smooth_count = 0
    ␣
 ↪data_tocsv(leftdrop,rightdrop,timestart,timeend,rightslope,leftslope,peak_duration_count)
    daily_hydronum_tocsv(hydronum_daily)
    print('The number of hydropeak is', hydropeak_num)
    print('The sd is', sd)
    plothydro(df)

def to_integer(datetime):
    return int(datetime.total_seconds()/60)

def daily_hydronum_tocsv(hydronum_daily):
    hydronum_daily = pd.DataFrame(hydronum_daily)
    hydronum_daily.to_csv('Daily_hydronum.csv')

def␣
 ↪data_tocsv(leftdrop,rightdrop,timestart,timeend,rightslope,leftslope,peak_duration_count):
 ↪
    AllData = {
        'leftdrop': leftdrop,
        'rightdrop': rightdrop,
        'starting_time':timestart,
        'ending_time':timeend,
        'rightslope':rightslope,
        'leftslope':leftslope,
        'peak_duration':peak_duration_count
    }
    AllData = pd.DataFrame(AllData,columns=['leftdrop',
                                         'rightdrop',
                                         'starting_time',
                                        'ending_time',
                                        'rightslope',
                                        'leftslope',
                                         'peak_duration'
                                        ])
    AllData.to_csv('Output.csv')

def plothydro(df):
    plt.title('list')
    plt.xlabel('time')
    plt.ylabel('water level')
    plt.plot(df['time'], df['waterlevel'])
    plt.scatter(timestart, valuestart, c='g')
```

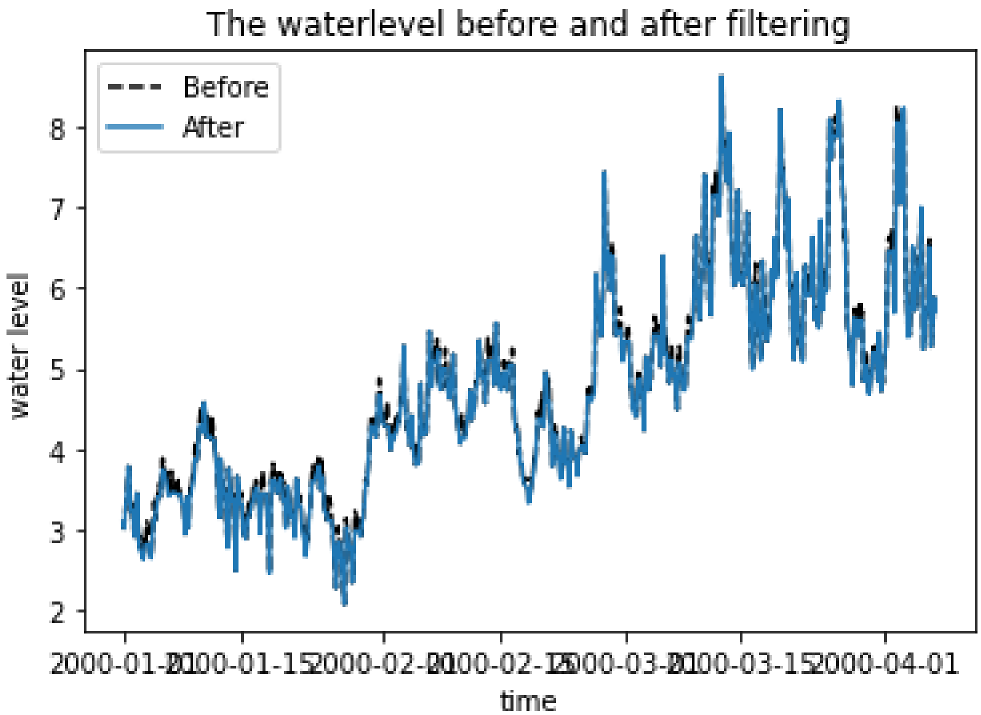
```
    plt.scatter(timeend, valueend, c='r')
    plt.show()

def main():
    register_matplotlib_converters()
    print('start')
    path = input('Please input the data path\n')
    #
    while True:
        filename = path[-4:]
        if filename != '.csv' and filename != 'xlsx':
            path = input('please input a excel or csv file\n')
        else:
            try:
                df = readtemplate(path)
                break
            except:
                path = input(
                    'Cannot find the file or there is something wrong. Please␣
 ↪input again\n')
    try:
        sd = 0.5 * cal_sd(df)
        filterhydro_plot(df, sd)
        searchhydro(df, sd)
    except KeyError:
        print('Please ensure there are columns named‘time’ ‘waterlevel’\n')
    print('finished')
```

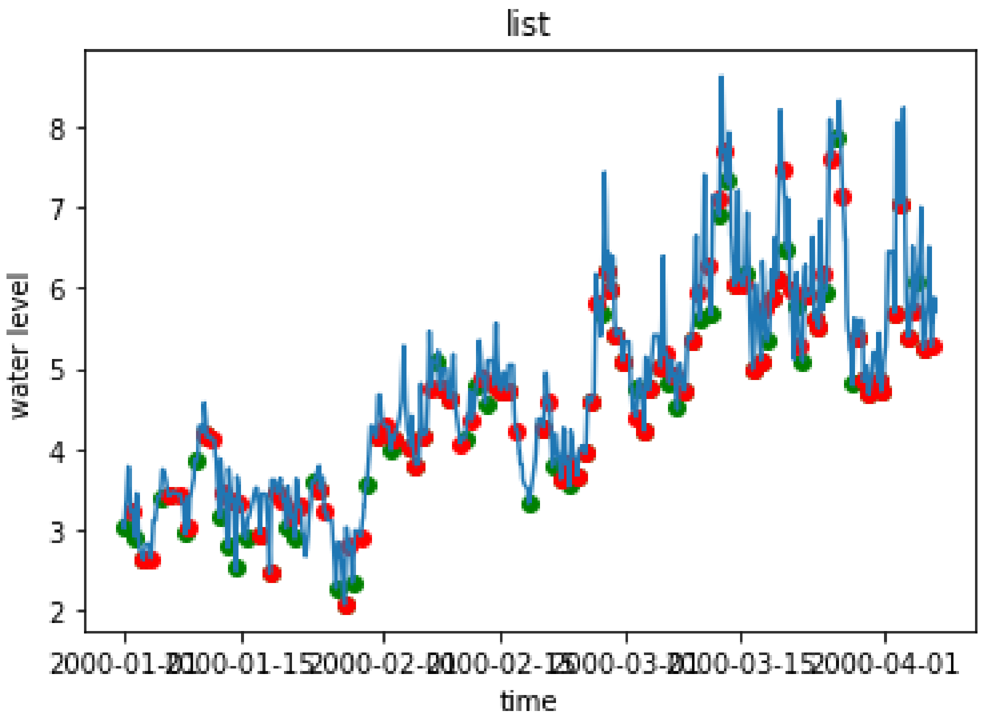
1.4 Running the file

[6]:
```
if __name__ == "__main__":
    main()
```

```
start
Please input the data path
../dataset/2009_Altered.xlsx
We have filtered 196 flowing within the sd
```




```
Does the filter look plausible? If yes, input 1:
1
The number of hydropeak is 91
The sd is 0.33278067741195316
```




```
finished
```

From the plot, we can clearly see the starting and ending point of the peak. The recorded statistics are saved as two files called 'Daily_hydronum.csv' and 'Output'