clip-max-ignore-zeros-custom

August 13, 2020

[1]:
```
# rank the obtained results using the *.log files
import os
import pandas as pd
import numpy as np
```

[2]:
```
source = "10"
targetdir = '../../data/' + source + "/"
filelist = sorted(os.listdir(targetdir))
```

[3]:
```
filelist
```

[3]:
```
['1.data', '2.data', '3.data', '4.data', '5.data', '6.data', '7.data']
```

[4]:
```
# Create dataframe from files
df = pd.DataFrame()

for file in filelist:
    filename = targetdir+file
    col_name = [file]
    temp_df = pd.read_csv(filename,names=col_name)
    df = pd.concat([df, temp_df], axis=1)

# Look at the data
df.head()
```

[4]:

	1.data	2.data	3.data	4.data	5.data	6.data	7.data
0	0.7981	0.9064	1.0000	1.0000	0.9103	1.0000	0.6492
1	0.7981	0.9064	1.0001	-1.0000	0.9103	-1.0000	0.6492
2	0.7981	0.9064	1.0002	-1.0000	0.9103	-1.0000	0.6492
3	0.6604	0.7501	0.8219	0.8276	0.7533	0.5203	0.5373
4	0.6604	0.7501	0.8219	0.8276	0.7533	0.5203	0.5373

[5]:
```
# Clip values > 1 with 1 and ignore 0s
df.mask(df > 1, 1, inplace=True)
df.mask(df <= 0, np.NaN, inplace=True)
```

[6]:
```
# Count NaN values
df.isna().sum()
```

[6]:
```
1.data      0
2.data      0
3.data      0
4.data    324
5.data      0
6.data    144
7.data      0
dtype: int64
```

[7]:
```
# Ignore invalid values by dropping them from the dataframe
df = df.dropna()
```

[8]:
```
df
```

[8]:

	1.data	2.data	3.data	4.data	5.data	6.data	7.data
0	0.7981	0.9064	1.0000	1.0000	0.9103	1.0000	0.6492
3	0.6604	0.7501	0.8219	0.8276	0.7533	0.5203	0.5373
4	0.6604	0.7501	0.8219	0.8276	0.7533	0.5203	0.5373
5	0.6604	0.7501	0.8219	0.8276	0.7533	0.5203	0.5373
6	0.5633	0.6398	0.6880	0.7058	0.6425	0.4438	0.4582
..	…	…	…	…	…	…	…
718	0.6209	0.6609	0.7854	0.9846	0.6495	0.4468	0.4640
719	0.6209	0.6609	0.7854	0.9846	0.6495	0.4468	0.4640
726	0.6306	0.6712	0.7977	1.0000	0.6597	0.4538	0.4713
727	0.6306	0.6712	0.7977	1.0000	0.6597	0.4538	0.4713
728	0.6306	0.6712	0.7977	1.0000	0.6597	0.4538	0.4713

[405 rows x 7 columns]

[9]:
```
# Save processed dataframe as csv file
df.to_csv("../../data/processed/" + source +".csv",index=False)
```

[10]:
```
# Creating ranked dataframe
ranked_df = pd.DataFrame()
stats_df = pd.DataFrame()
```

[11]:
```
# Creating scenario quantity variable
tao = len(df)
tao
```

[11]:
```
405
```

0.1 Ranking of WWTP

[12]:
```
for column in df:
    wwtp = column[0]

    # TODO: get original (pre-analysis) value
    # pending

    # calculate mean
    avg_eff = round(df[column].mean(),3)

    # calculate max
    max_eff = round(df[column].max(),3)

    # calculate min
    min_eff = round(df[column].min(),3)

    # calculate amplitude
    amplitude = round((max_eff - min_eff)*100,2)

    amp_str = "Amplitude (max-min)(%)"

    # print stats results
    print("WWTP", wwtp,"Mean =",avg_eff,"Maximum =",max_eff,"Minimum␣
↪=",min_eff, amp_str,"=",amplitude)
    stats_df = stats_df.append({ 'WWTP': wwtp, "Mean": avg_eff, "Maximum" :␣
↪max_eff, "Minimum": min_eff, amp_str: amplitude},ignore_index=True)


    # TODO: Populate statistics dataframe using pd.df.append

    # Calculating Sk sum of factors
    Sk = round(df[column].sum(),3)

    # Calculating ek sum of factors of 1 (or above if errors in calculation)
    ek = df[column] >= 1
    ek = ek.sum()
    print("ek =",ek)

    # Calculating R1k ek/tao
    R1k = round(ek/tao,3)

    # Calculate R2k
    if tao != ek:
        R2k = (Sk - ek)/(tao - ek)
    elif R1k == 1:
        R2k = 0
```

```
    R2k = round(R2k,3)

    # Printing results
    print("WWTP", wwtp,"| ek =",ek,"| R1k =",R1k, "| Sk =",Sk, "| R2k =",R2k)

    # Populate ranking dataframe using pd.df.append
    # Using unicode to name columns with super and subscripts
    R1k_col = 'R\u00B9\u2096\u2080'
    R2k_col = 'R\u00B2\u2096\u2080'
    ranked_df = ranked_df.append({ R2k_col:R2k, R1k_col: R1k,'WWTP':␣
↪wwtp},ignore_index=True)
```

```
WWTP 1 Mean = 0.649 Maximum = 0.823 Minimum = 0.552 Amplitude (max-min)(%) =
27.1
ek = 0
WWTP 1 | ek = 0 | R1k = 0.0 | Sk = 263.004 | R2k = 0.649
WWTP 2 Mean = 0.724 Maximum = 0.935 Minimum = 0.63 Amplitude (max-min)(%) = 30.5
ek = 0
WWTP 2 | ek = 0 | R1k = 0.0 | Sk = 293.386 | R2k = 0.724
WWTP 3 Mean = 0.805 Maximum = 1.0 Minimum = 0.65 Amplitude (max-min)(%) = 35.0
ek = 30
WWTP 3 | ek = 30 | R1k = 0.074 | Sk = 325.958 | R2k = 0.789
WWTP 4 Mean = 0.875 Maximum = 1.0 Minimum = 0.684 Amplitude (max-min)(%) = 31.6
ek = 108
WWTP 4 | ek = 108 | R1k = 0.267 | Sk = 354.209 | R2k = 0.829
WWTP 5 Mean = 0.72 Maximum = 0.939 Minimum = 0.622 Amplitude (max-min)(%) = 31.7
ek = 0
WWTP 5 | ek = 0 | R1k = 0.0 | Sk = 291.73 | R2k = 0.72
WWTP 6 Mean = 0.525 Maximum = 1.0 Minimum = 0.433 Amplitude (max-min)(%) = 56.7
ek = 18
WWTP 6 | ek = 18 | R1k = 0.044 | Sk = 212.57 | R2k = 0.503
WWTP 7 Mean = 0.515 Maximum = 0.67 Minimum = 0.446 Amplitude (max-min)(%) = 22.4
ek = 0
WWTP 7 | ek = 0 | R1k = 0.0 | Sk = 208.421 | R2k = 0.515
```

[13]:
```
# Reorder columns to be usable as a results table
ranked_df = ranked_df.reindex(columns=['WWTP',R1k_col, R2k_col])
```

[14]:
```
ranked_df
```

[14]:

	WWTP	R^1	R^2
0	1	0.000	0.649
1	2	0.000	0.724
2	3	0.074	0.789
3	4	0.267	0.829
4	5	0.000	0.720

```
5    6  0.044  0.503
6    7  0.000  0.515
```

[15]:
```
import os

path = "../../results/" + source + "/"

# Save rankings dataframe as csv file

try:
    ranked_df.to_csv(path + "ranking.csv",index=False)
    print("Save succesful")
except:
    print("Creating folder and saving")
    os.mkdir(path)
    ranked_df.to_csv(path + "ranking.csv",index=False)
```

```
Save succesful
```

0.2 Calculate Descriptive Statistics

[16]:
```
# Calculate the mean of every column
mean_mean = round(stats_df.Mean.mean(),3)
mean_max = round(stats_df.Maximum.mean(),3)
mean_min = round(stats_df.Minimum.mean(),3)
mean_amp = round(stats_df[amp_str].mean(),3)
```

[17]:
```
# Add means to stats dataframe
stats_df = stats_df.append({ 'WWTP': "Mean", "Mean" : mean_mean, "Maximum" :␣
↪mean_max,
                            "Minimum" : mean_min, amp_str :␣
↪mean_amp},ignore_index=True)
```

[18]:
```
# Calculate the standard deviation of every column
sd_mean = round(stats_df.Mean.std(),3)
sd_max = round(stats_df.Maximum.std(),3)
sd_min = round(stats_df.Minimum.std(),3)
sd_amp = round(stats_df[amp_str].std(),3)
```

[19]:
```
# Add means to stats dataframe
stats_df = stats_df.append({ 'WWTP': "SD", "Mean" : sd_mean, "Maximum" : sd_max,
                            "Minimum" : sd_min, amp_str :␣
↪sd_amp},ignore_index=True)
```

[20]:
```
# Reorder columns
```

```
stats_df = stats_df.reindex(columns=["WWTP", "Mean", "Maximum", "Minimum",␣
↪amp_str])
```

[21]:
```
stats_df
```

[21]:

	WWTP	Mean	Maximum	Minimum	Amplitude (max-min)(%)
0	1	0.649	0.823	0.552	27.100
1	2	0.724	0.935	0.630	30.500
2	3	0.805	1.000	0.650	35.000
3	4	0.875	1.000	0.684	31.600
4	5	0.720	0.939	0.622	31.700
5	6	0.525	1.000	0.433	56.700
6	7	0.515	0.670	0.446	22.400
7	Mean	0.688	0.910	0.574	33.571
8	SD	0.125	0.114	0.093	10.145

[22]:
```
# Save statistics dataframe as csv file
stats_df.to_csv(path + "statistics.csv",index=False)
```

[23]:
```
# Convert Jupyter Notebook to PDF LaTeX file
!jupyter-nbconvert --to pdf "clip-max-ignore-zeros-custom.ipynb" --output-dir ".
↪./../results/10/"
```

```
[NbConvertApp] Converting notebook clip-max-ignore-zeros-custom.ipynb to pdf
[NbConvertApp] Writing 45170 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 47339 bytes to ../../results/10/clip-max-ignore-zeros-
custom.pdf
```

[ ]:
```

```