

clip-max-ignore-zeros-custom

August 2, 2020

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

```
[2]: source = "6a"
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	1.0	0.87	1.00	1.00	0.63	1.00	1.0
1	1.0	0.87	-1.00	-1.00	0.63	-1.00	1.0
2	1.0	0.87	-1.00	-1.00	0.63	-1.00	1.0
3	1.0	0.80	0.98	0.92	0.59	0.53	1.0
4	1.0	0.80	0.98	0.92	0.59	0.53	1.0

```
[5]: # Clip values > 1 with 1 and ignore non-positive values
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    162
     2.data     0
     3.data   342
     4.data   324
     5.data     0
     6.data   297
     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	1.0	0.87	1.00	1.00	0.63	1.00	1.0
3	1.0	0.80	0.98	0.92	0.59	0.53	1.0
4	1.0	0.80	0.98	0.92	0.59	0.53	1.0
6	1.0	0.74	0.95	0.85	0.53	0.50	1.0
7	1.0	0.74	0.95	0.85	0.53	0.50	1.0
..
718	1.0	0.86	0.99	1.00	0.63	1.00	1.0
719	1.0	0.86	0.99	1.00	0.63	1.00	1.0
726	1.0	0.86	1.00	1.00	0.63	0.86	1.0
727	1.0	0.86	1.00	1.00	0.63	1.00	1.0
728	1.0	0.86	1.00	1.00	0.63	1.00	1.0

```
[387 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]: 387
```

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':\u2192wwtp},ignore_index=True)

```

```

WWTP 1 Mean = 1.0 Maximum = 1.0 Minimum = 1.0 Amplitude (max-min)(%) = 0.0
ek = 387
WWTP 1 | ek = 387 | R1k = 1.0 | Sk = 387.0 | R2k = 0
WWTP 2 Mean = 0.81 Maximum = 0.88 Minimum = 0.72 Amplitude (max-min)(%) = 16.0
ek = 0
WWTP 2 | ek = 0 | R1k = 0.0 | Sk = 313.44 | R2k = 0.81
WWTP 3 Mean = 0.979 Maximum = 1.0 Minimum = 0.93 Amplitude (max-min)(%) = 7.0
ek = 108
WWTP 3 | ek = 108 | R1k = 0.279 | Sk = 378.75 | R2k = 0.97
WWTP 4 Mean = 0.94 Maximum = 1.0 Minimum = 0.84 Amplitude (max-min)(%) = 16.0
ek = 126
WWTP 4 | ek = 126 | R1k = 0.326 | Sk = 363.88 | R2k = 0.911
WWTP 5 Mean = 0.597 Maximum = 0.66 Minimum = 0.52 Amplitude (max-min)(%) = 14.0
ek = 0
WWTP 5 | ek = 0 | R1k = 0.0 | Sk = 231.13 | R2k = 0.597
WWTP 6 Mean = 0.696 Maximum = 1.0 Minimum = 0.49 Amplitude (max-min)(%) = 51.0
ek = 84
WWTP 6 | ek = 84 | R1k = 0.217 | Sk = 269.5 | R2k = 0.612
WWTP 7 Mean = 1.0 Maximum = 1.0 Minimum = 1.0 Amplitude (max-min)(%) = 0.0
ek = 387
WWTP 7 | ek = 387 | R1k = 1.0 | Sk = 387.0 | R2k = 0

```

```

[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  R1  R2
0     1  1.000  0.000
1     2  0.000  0.810
2     3  0.279  0.970
3     4  0.326  0.911
4     5  0.000  0.597

```

5	6	0.217	0.612
6	7	1.000	0.000

```
[16]: import os

# define the name of the directory to be created
path = "../../results/" + source + "/"

try:
    os.mkdir(path)
except OSError:
    print ("Creation of the directory %s failed" % path)
else:
    print ("Successfully created the directory %s " % path)
```

Creation of the directory ../../results/6a/ failed

```
[19]: 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

2 Calculate Descriptive Statistics

```
[ ]: # 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)
```

```
[ ]: # 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)
```

```
[ ]: # 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)
```

```
[ ]: # 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)
```

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

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

```
[ ]: stats_df
```

```
[ ]: # Save statistics dataframe as csv file
```

```
stats_df.to_csv(path + "statistics.csv",index=False)
```

```
[ ]: # Convert Jupyter Notebook to PDF LaTeX file
```

```
!jupyter-nbconvert --to pdf "clip-max-ignore-zeros-custom.ipynb" --output-dir ".  
↪./../results/6a/"
```

```
[ ]: !jupyter nbconvert --to <output format> <input notebook> -h
```

```
[ ]:
```