ignore-over-max

July 23, 2020

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

[2]:
```
targetdir = '../../data/processed/'
filelist = sorted(os.listdir(targetdir))
```

[3]:
```
filelist
```

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

[4]:
```
df = pd.DataFrame()
```

[5]:
```
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)
```

[6]:
```
df
```

[6]:

	1.data	2.data	3.data	4.data	5.data	6.data	7.data
0	1.00	1.00	1.00	1.00	0.73	1.00	1.00
1	1.06	1.16	0.00	0.00	0.73	0.00	0.00
2	1.12	1.22	0.00	0.00	0.73	0.00	0.00
3	0.99	0.83	0.91	0.83	0.63	0.91	1.00
4	0.99	0.83	0.91	0.83	0.63	0.00	0.00
..	…	…	…	…	…	…	…
724	1.06	0.87	0.00	0.00	0.65	0.88	1.01
725	1.11	0.87	0.00	0.00	0.65	0.00	0.00
726	1.00	0.74	1.00	1.00	0.54	0.50	1.00
727	1.00	0.74	1.00	1.00	0.54	0.50	1.00
728	1.00	0.74	1.00	1.00	0.54	0.50	1.00

[729 rows x 7 columns]

[7]:
```
# Replace values >1 with NaN
df.mask(df > 1, np.NaN, inplace=True)
```

[8]:
```
# Count NaN values
df.isna().sum()
```

[8]:
```
1.data    376
2.data    198
3.data     54
4.data     54
5.data      0
6.data     41
7.data    168
dtype: int64
```

Now we do have invalid values in our data so we will drop them from the dataframe.

[9]:
```
# Ignore invalid values by dropping them from the dataframe
df = df.dropna()
```

[10]:
```
df
```

[10]:

	1.data	2.data	3.data	4.data	5.data	6.data	7.data
0	1.00	1.00	1.00	1.00	0.73	1.00	1.00
3	0.99	0.83	0.91	0.83	0.63	0.91	1.00
4	0.99	0.83	0.91	0.83	0.63	0.00	0.00
5	0.99	0.83	0.91	0.83	0.63	0.00	0.00
6	0.97	0.71	0.81	0.71	0.53	0.49	1.00
..	…	…	…	…	…	…	…
718	0.99	0.73	0.97	0.98	0.53	0.50	0.99
719	0.99	0.73	0.97	0.98	0.53	0.50	0.99
726	1.00	0.74	1.00	1.00	0.54	0.50	1.00
727	1.00	0.74	1.00	1.00	0.54	0.50	1.00
728	1.00	0.74	1.00	1.00	0.54	0.50	1.00

[302 rows x 7 columns]

[11]:
```
# Creating ranked dataframe
ranked_df = pd.DataFrame()
stats_df = pd.DataFrame()
```

The tao variable should be the length of the dataframe.

[12]:
```
# Creating scenario quantity variable
tao = len(df)
tao
```

[12]:
```
302
```

[13]:
```
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()

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

    # Calculate R2k
    if tao != ek:
        R2k = (Sk - ek)/(tao - ek)
    elif Rk1 == 1:
        R2k = 0

    R2k = round(R2k,3)
```

```
    # Printing results
    print("WWTP", wwtp,"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 = 0.953 Maximum = 1.0 Minimum = 0.0 Amplitude (max-min)(%) = 100.0
WWTP 1 R1k = 0.219 | Sk = 287.89 | R2k = 0.94
WWTP 2 Mean = 0.794 Maximum = 1.0 Minimum = 0.7 Amplitude (max-min)(%) = 30.0
WWTP 2 R1k = 0.06 | Sk = 239.91 | R2k = 0.781
WWTP 3 Mean = 0.874 Maximum = 1.0 Minimum = 0.0 Amplitude (max-min)(%) = 100.0
WWTP 3 R1k = 0.179 | Sk = 263.82 | R2k = 0.846
WWTP 4 Mean = 0.832 Maximum = 1.0 Minimum = 0.0 Amplitude (max-min)(%) = 100.0
WWTP 4 R1k = 0.179 | Sk = 251.13 | R2k = 0.795
WWTP 5 Mean = 0.587 Maximum = 0.74 Minimum = 0.51 Amplitude (max-min)(%) = 23.0
WWTP 5 R1k = 0.0 | Sk = 177.34 | R2k = 0.587
WWTP 6 Mean = 0.573 Maximum = 1.0 Minimum = 0.0 Amplitude (max-min)(%) = 100.0
WWTP 6 R1k = 0.06 | Sk = 173.17 | R2k = 0.546
WWTP 7 Mean = 0.709 Maximum = 1.0 Minimum = 0.0 Amplitude (max-min)(%) = 100.0
WWTP 7 R1k = 0.358 | Sk = 214.2 | R2k = 0.547
```

[14]:
```
# Reorder columns to be usable as a results table
ranked_df = ranked_df.reindex(columns=['WWTP',R1k_col, R2k_col])
```

[15]:
```
ranked_df
```

[15]:

	WWTP	R^{1}	R^{2}
0	1	0.219	0.940
1	2	0.060	0.781
2	3	0.179	0.846
3	4	0.179	0.795
4	5	0.000	0.587
5	6	0.060	0.546
6	7	0.358	0.547

[16]:
```
# Save rankings dataframe as csv file
ranked_df.to_csv("../../results/ignore-over-max/ranking.csv",index=False)
```

[17]:
```
# 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)
```

[18]:
```
# 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)
```

[19]:
```
# 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)
```

[20]:
```
# 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)
```

[21]:
```
# Reorder columns
stats_df = stats_df.reindex(columns=["WWTP", "Mean", "Maximum", "Minimum",␣
↪amp_str])
```

[22]:
```
stats_df
```

[22]:

	WWTP	Mean	Maximum	Minimum	Amplitude (max-min)(%)
0	1	0.953	1.000	0.000	100.000
1	2	0.794	1.000	0.700	30.000
2	3	0.874	1.000	0.000	100.000
3	4	0.832	1.000	0.000	100.000
4	5	0.587	0.740	0.510	23.000
5	6	0.573	1.000	0.000	100.000
6	7	0.709	1.000	0.000	100.000
7	Mean	0.760	0.963	0.173	79.000
8	SD	0.133	0.091	0.278	33.257

[23]:
```
# Save statistics dataframe as csv file
stats_df.to_csv("../../results/ignore-over-max/statistics.csv",index=False)
```