CoCoSo Ranking

March 19, 2022

1 CoCoSo Ranking

[1]:
```
import math                       # For sqrt and other stuff
import numpy as np                # For linear algebra
import pandas as pd               # For tabular output
from scipy.stats import rankdata # For ranking the candidates based on score
```

2 Step 0 - Obtaining and preprocessing the data

[2]:
```
attributes_data = pd.read_csv('../data/criteria.csv')
attributes_data
```

[2]:

	Indicator	Name	Unit	\
0	C1	The average wage	US Dollar	
1	C2	The employment rate	% of the working age population	
2	C3	Income inequality	ratio	
3	C4	Labor force	Thousand persons	
4	C5	Poverty gap	Ratio	
5	C6	Poverty rate	Ratio	
6	C7	Working hours	Hours/worker	
7	C8	Women in politics	Percentage	
8	C9	Population density	Ratio	
9	C10	Adult education level	% of 25-64 year-old	
10	C11	Spending on tertiary education	% of education spending	
11	C12	International student mobility	% of students enrolled	
12	C13	Tertiary graduation rate	% of the same level	
13	C14	Social spending	% of GDP	

	Ideally	Rank
0	Higher	11
1	Higher	7
2	Lower	14
3	Higher	1
4	Lower	10
5	Lower	9
6	Higher	8

```
7   Higher     5
8    Lower     2
9   Higher     6
10  Higher     4
11  Higher     3
12  Higher    13
13  Higher    12
```

[3]:
```
benefit_attributes = set()
attributes = []
rankings = []
n = 0

for i, row in attributes_data.iterrows():
    attributes.append(row['Indicator'])
    rankings.append(row['Rank'])
    n += 1

    if row['Ideally'] == 'Higher':
        benefit_attributes.add(i)
```

[4]:
```
# rankings = np.array(rankings)
# weights = 2 * (n + 1 - rankings) / (n * (n + 1))
weights = [
    0.0393,
    0.0498,
    0.0358,
    0.1819,
    0.0406,
    0.0433,
    0.0457,
    0.0779,
    0.1748,
    0.0620,
    0.0789,
    0.0940,
    0.0370,
    0.0389
]

pd.DataFrame(zip(attributes, weights), columns=['Attribute', 'Weight'])
```

[4]:
```
   Attribute  Weight
0         C1  0.0393
1         C2  0.0498
2         C3  0.0358
3         C4  0.1819
```

```
4          C5  0.0406
5          C6  0.0433
6          C7  0.0457
7          C8  0.0779
8          C9  0.1748
9         C10  0.0620
10        C11  0.0789
11        C12  0.0940
12        C13  0.0370
13        C14  0.0389
```

[5]:
```
print(f'The sum of the weights is {sum(weights):0.2f}')
```

```
The sum of the weights is 1.00
```

[6]:
```
original_dataframe = pd.read_csv('../data/alternatives.csv').T

updated_dataframe = original_dataframe.drop(original_dataframe.index[0])

candidates = np.array(updated_dataframe.index)
raw_data = updated_dataframe.to_numpy()

[m, n] = updated_dataframe.shape

pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[6]:

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
CA	53198.17	64.73	0.31	20199.55	0.3	0.12	1670.0	51.7	4.0	57.88	49.052	12.917	54.4	20.89
FR	46480.62	66.02	0.29	29682.22	0.25	0.08	1505.0	52.9	122.0	36.89	77.838	10.201	54.31	31.68
DE	53637.8	76.09	0.28	43769.63	0.25	0.1	1386.1	33.3	237.0	29.06	82.723	8.373	49.33	24.76
IT	39189.37	59.07	0.33	25941.4	0.4	0.13	1717.8	27.8	205.0	19.32	61.715	5.311	56.07	25.36
JP	38617.47	77.95	0.33	68863.34	0.33	0.15	1644.0	15.8	347.0	51.92	32.416	4.265	36.87	23.51
UK	47226.09	75.61	0.35	33964.07	0.34	0.11	1538.0	30.8	275.0	45.78	24.991	17.918	54.47	24.49
USA	65835.58	62.56	0.39	163538.7	0.38	0.17	1779.0	16.7	36.0	47.43	35.205	5.18	55.41	30.02

3 Step 1 - Normalizing the Ratings and Weights

[7]:
```
max_vals = np.amax(raw_data, axis=0)
min_vals = np.amin(raw_data, axis=0)

for j in range(n):
    column = raw_data[:,j]
    denominator = max_vals[j] - min_vals[j]

    if j in benefit_attributes:
        raw_data[:,j] = (raw_data[:,j] - min_vals[j]) / denominator
    else:
        raw_data[:,j] = (max_vals[j] - raw_data[:,j]) / denominator

pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[7]:

	C1	C2	C3	C4	C5	C6	C7	\
CA	0.535698	0.299788	0.727273	0.0	0.666667	0.555556	0.722576	
FR	0.288894	0.368114	0.909091	0.066155	1.0	1.0	0.302622	
DE	0.551851	0.901483	1.0	0.164436	1.0	0.777778	0.0	
IT	0.021012	0.0	0.545455	0.040058	0.0	0.444444	0.844235	
JP	0.0	1.0	0.545455	0.339501	0.466667	0.222222	0.656401	
UK	0.316283	0.876059	0.363636	0.096028	0.4	0.666667	0.386612	
USA	1.0	0.184852	0.0	1.0	0.133333	0.0	1.0	

	C8	C9	C10	C11	C12	C13	C14
CA	0.967655	1.0	1.0	0.416771	0.633707	0.913021	0.0
FR	1.0	0.655977	0.455654	0.915385	0.434776	0.908333	1.0
DE	0.471698	0.3207	0.252593	1.0	0.300886	0.648958	0.358665
IT	0.32345	0.413994	0.0	0.636112	0.076613	1.0	0.414272
JP	0.0	0.0	0.845436	0.128612	0.0	0.0	0.242817
UK	0.404313	0.209913	0.686203	0.0	1.0	0.916667	0.333642
USA	0.024259	0.906706	0.728994	0.176921	0.067018	0.965625	0.846154

[8]:
```
sum_values_full = np.zeros((m, n))
pow_values_full = np.zeros((m, n))

for i in range(m):
    sum_values_full[i, :] = weights * raw_data[i, :]
    pow_values_full[i, :] = weights ** raw_data[i, :]
```

[9]:
```
pd.DataFrame(data=sum_values_full, index=candidates, columns=attributes)
```

[9]:

	C1	C2	C3	C4	C5	C6	C7	\
CA	0.021053	0.014929	0.026036	0.000000	0.027067	0.024056	0.033022	
FR	0.011354	0.018332	0.032545	0.012034	0.040600	0.043300	0.013830	
DE	0.021688	0.044894	0.035800	0.029911	0.040600	0.033678	0.000000	

```
IT   0.000826  0.000000  0.019527  0.007287  0.000000  0.019244  0.038582
JP   0.000000  0.049800  0.019527  0.061755  0.018947  0.009622  0.029998
UK   0.012430  0.043628  0.013018  0.017467  0.016240  0.028867  0.017668
USA  0.039300  0.009206  0.000000  0.181900  0.005413  0.000000  0.045700

           C8        C9        C10       C11       C12       C13       C14
CA   0.075380  0.174800  0.062000  0.032883  0.059568  0.033782  0.000000
FR   0.077900  0.114665  0.028251  0.072224  0.040869  0.033608  0.038900
DE   0.036745  0.056058  0.015661  0.078900  0.028283  0.024011  0.013952
IT   0.025197  0.072366  0.000000  0.050189  0.007202  0.037000  0.016115
JP   0.000000  0.000000  0.052417  0.010147  0.000000  0.000000  0.009446
UK   0.031496  0.036693  0.042545  0.000000  0.094000  0.033917  0.012979
USA  0.001890  0.158492  0.045198  0.013959  0.006300  0.035728  0.032915
```

[10]:
```
pd.DataFrame(data=pow_values_full, index=candidates, columns=attributes)
```

[10]:
```
           C1        C2        C3        C4        C5        C6        C7  \
CA   0.176611  0.406860  0.088773  1.000000  0.118127  0.174781  0.107569
FR   0.392580  0.331460  0.048456  0.893375  0.040600  0.043300  0.393063
DE   0.167616  0.066923  0.035800  0.755597  0.040600  0.086994  1.000000
IT   0.934256  1.000000  0.162633  0.934008  1.000000  0.247739  0.073902
JP   1.000000  0.049800  0.162633  0.560677  0.224205  0.497734  0.131937
UK   0.359278  0.072226  0.297947  0.849031  0.277594  0.123308  0.303324
USA  0.039300  0.574355  1.000000  0.181900  0.652334  1.000000  0.045700

           C8        C9        C10       C11       C12       C13       C14
CA   0.084604  0.174800  0.062000  0.347004  0.223493  0.049288  1.000000
FR   0.077900  0.318511  0.281675  0.097814  0.357718  0.050055  0.038900
DE   0.300013  0.571588  0.495412  0.078900  0.490940  0.117712  0.312078
IT   0.437993  0.485754  1.000000  0.198801  0.834311  0.037000  0.260528
JP   1.000000  1.000000  0.095289  0.721359  1.000000  1.000000  0.454585
UK   0.356315  0.693426  0.148366  1.000000  0.094000  0.048699  0.338491
USA  0.939962  0.205688  0.131723  0.638072  0.853455  0.041440  0.064103
```

[11]:
```
sum_values = np.sum(sum_values_full, axis=1)
pow_values = np.sum(pow_values_full, axis=1)
```

[12]:
```
pd.DataFrame(data=sum_values, index=candidates, columns=['Sum-Weighted CS'])
```

[12]:
```
     Sum-Weighted CS
CA          0.584576
FR          0.578411
DE          0.460181
IT          0.293535
JP          0.261659
UK          0.400947
USA         0.576001
```

[13]:
```
pd.DataFrame(data=pow_values, index=candidates, columns=['power-Weighted CS'])
```

[13]:

	power-Weighted CS
CA	4.013909
FR	3.365408
DE	4.520173
IT	7.606925
JP	7.898220
UK	4.962006
USA	6.368032

[14]:
```
ma_denom = sum(sum_values) + sum(pow_values)

m_a = (sum_values + pow_values) / ma_denom

pd.DataFrame(data=m_a, index=candidates, columns=['$M_a$'])
```

[14]:

	M_a
CA	0.109775
FR	0.094147
DE	0.118891
IT	0.188600
JP	0.194793
UK	0.128025
USA	0.165768

[15]:
```
min_sum = min(sum_values)
min_pow = min(pow_values)

m_b = sum_values / min_sum + pow_values / min_pow

pd.DataFrame(data=m_b, index=candidates, columns=['$M_b$'])
```

[15]:

	M_b
CA	3.426812
FR	3.210553
DE	3.101835
IT	3.382149
JP	3.346883
UK	3.006740
USA	4.093543

[16]:
```
lambda_ = 0.5
```

[17]:
```
max_sum = max(sum_values)
max_pow = max(pow_values)

one_minus_lambda = 1 - lambda_
```

```
mc_denom = lambda_ * max_sum + one_minus_lambda * max_pow

m_c = (lambda_ * sum_values + one_minus_lambda * pow_values) / mc_denom

pd.DataFrame(data=m_c, index=candidates, columns=['$M_c$'])
```

[17]:

	M_c
CA	0.542096
FR	0.464920
DE	0.587112
IT	0.931351
JP	0.961933
UK	0.632215
USA	0.818602

[18]:
```
one_third = 1.0 / 3.0
m_vals = (m_a * m_b * m_c) ** one_third + one_third * (m_a + m_b + m_c)

pd.DataFrame(data=m_vals, index=candidates, columns=['$M$'])
```

[18]:

	M
CA	1.948165
FR	1.776442
DE	1.869757
IT	2.341352
JP	2.357162
UK	1.879996
USA	2.514675

[21]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage
```

[22]:
```
result = rank_according_to(m_vals)
pd.DataFrame(data=result, index=range(1, m + 1), columns=['Name'])
```

[22]:

	Name
1	FR
2	DE
3	UK
4	CA
5	IT
6	JP
7	USA

[ ]: