Multiple-criteria decision analysis

In [1]:
```
from urllib import request
from bs4 import BeautifulSoup
import re
import statistics
import urllib
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import time
import math
from tabula import read_pdf
from matplotlib.colors import ListedColormap
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png', 'pdf')
import networkx as nx
from tabulate import tabulate
from IPython.display import Image, display
from pymcdm import methods as mcdm_methods
from pymcdm import weights as mcdm_weights
from pymcdm import normalizations as norm
from pymcdm import correlations as corr
from pymcdm.helpers import rankdata, rrankdata
```

In [2]:
```
os.getcwd()
```

Out[2]:
```
'D:\\mcdm'
```

In [3]:
```
ls
```

```
 Volume in drive D is Data
 Volume Serial Number is 125B-8C1E

 Directory of D:\mcdm

18-Sep-22  10:47 AM    <DIR>          .
25-Aug-22  07:16 PM    <DIR>          .ipynb_checkpoints
18-Sep-22  10:44 AM               821 am.txt
17-Sep-22  11:56 AM            10,370 final.xlsx
18-Sep-22  10:47 AM               803 final1
25-Aug-22  07:11 PM            99,738 matrix_image.jpeg
08-Sep-22  10:13 PM         1,360,338 mcdm_models - Jupyter Notebook.pdf
18-Sep-22  10:46 AM           385,709 mcdm_models.ipynb
28-Aug-22  04:41 PM             1,402 weights.csv
25-Aug-22  07:11 PM            66,825 weights_image.jpeg
               8 File(s)      1,926,006 bytes
               2 Dir(s)  988,194,381,824 bytes free
```

In [4]:
```
1 display(Image("matrix_image.jpeg"))
```

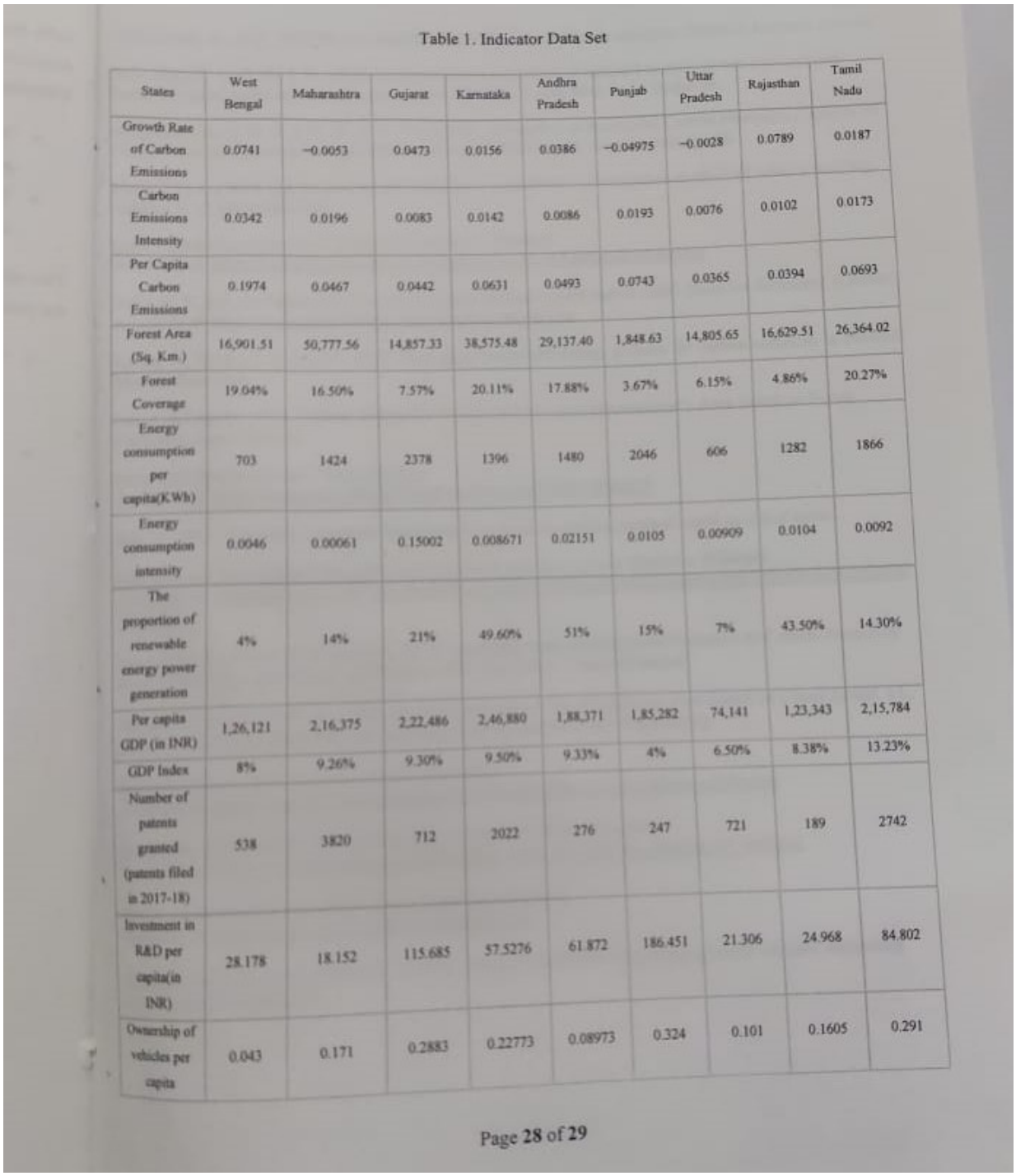


Table 1. Indicator Data Set

States	West Bengal	Maharashtra	Gujarat	Karnataka	Andhra Pradesh	Punjab	Uttar Pradesh	Rajasthan	Tamil Nadu
Growth Rate of Carbon Emissions	0.0741	−0.0053	0.0473	0.0156	0.0386	−0.04975	−0.0028	0.0789	0.0187
Carbon Emissions Intensity	0.0342	0.0196	0.0083	0.0142	0.0086	0.0193	0.0076	0.0102	0.0173
Per Capita Carbon Emissions	0.1974	0.0467	0.0442	0.0631	0.0493	0.0743	0.0365	0.0394	0.0693
Forest Area (Sq. Km.)	16,901.51	50,777.56	14,857.33	38,575.48	29,137.40	1,848.63	14,805.65	16,629.51	26,364.02
Forest Coverage	19.04%	16.50%	7.57%	20.11%	17.88%	3.67%	6.15%	4.86%	20.27%
Energy consumption per capita(KWh)	703	1424	2378	1396	1480	2046	606	1282	1866
Energy consumption intensity	0.0046	0.00061	0.15002	0.008671	0.02151	0.0105	0.00909	0.0104	0.0092
The proportion of renewable energy power generation	4%	14%	21%	49.60%	51%	15%	7%	43.50%	14.30%
Per capita GDP (in INR)	1,26,121	2,16,375	2,22,486	2,46,880	1,88,371	1,85,282	74,141	1,23,343	2,15,784
GDP Index	8%	9.26%	9.30%	9.50%	9.33%	4%	6.50%	8.38%	13.23%
Number of patents granted (patents filed in 2017-18)	538	3820	712	2022	276	247	721	189	2742
Investment in R&D per capita(in INR)	28.178	18.152	115.685	57.5276	61.872	186.451	21.306	24.968	84.802
Ownership of vehicles per capita	0.043	0.171	0.2883	0.22773	0.08973	0.324	0.101	0.1605	0.291

Page 28 of 29

description of the data set

In [5]:
```
1 df = pd.read_excel(r"D:\mcdm\final.xlsx")
```

In [6]:
```
1 df1 = pd.read_csv(r"D:\mcdm\weights.csv")
```

In [7]:
```
1 df1
```

Out[7]:

	INDICATORS	STATES	WEST BENGAL	MAHARASHTRA	GUJARAT	KARNATAKA	AND PRAD
0	A1	Growth rate of carbon emissions	5.0000	8.0000	14.0000	8.0000	7.
1	A2	Carbon emissions intensity	0.0342	0.0196	0.0083	0.0142	0.
2	A3	Per capita carbon emissions	0.1974	0.0467	0.0442	0.0631	0.
3	A4	Forest area(sq. km)	16901.5100	50777.5600	14875.3300	38575.4800	29137.
4	A5	Forest coverage rate	0.1904	0.1650	0.0757	0.2011	0.
5	A6	Energy consumptopn per capita(KWh)	703.0000	1424.0000	2378.0000	1396.0000	1480.
6	A7	Energy consumptoion intensity	0.0046	0.0006	0.1500	0.0087	0.
7	A8	The proportion of renewable energy power gener...	0.0400	0.1400	0.2100	0.4960	0.
8	A9	per capita GDP(in INR)	126121.0000	216375.0000	222486.0000	246880.0000	188371.
9	A10	GDP index	0.0800	0.0926	0.0930	0.0950	0.
10	A11	Number of patents granted(patents filled in 20...	538.0000	3820.0000	712.0000	2022.0000	276.
11	A12	Investment in R&D per capita(in INR)	28.1780	18.1520	115.6850	57.5276	61.
12	A13	Ownership of vehicles per capita	0.0430	0.1710	0.2883	0.2277	0.

In [10]:
```
df = df.T
df.drop(index=df.index[0],
        axis=0,
        inplace=True)
df
```

Out[10]:

	W.B	Mah	Guj	Kar	A.P.	Pun	U.P.	Rajasthan	T.N.
1	49.5	27.5	48.3	42.7	92.6	35	63.2	50.2	31.3
2	.197.04	0.0467	0.0442	0.0631	0.0493	0.0743	0.0365	0.0394	0.0693
3	16,901.51	50,777.56	14857.33	38575.48	29137.40	1848.63	14,805.65	16629.51	26,364.02
4	19.04	16.5	7.57	20.19	17.88	3.67	6.15	4.86	20.27
5	703	1424	2378	1396	1480	2046	606	1282	1866
6	0.0046	0.00061	0.15002	0.008671	0.02151	0.0105	0.00909	0.0104	0.0092
7	12.33	32.73	44.75	65.57	39.78	33.66	15.92	60.65	52.11
8	121463	202130	213936	223175	168480	151491	65431	116492	213396
9	7.5	6.2	13.1	8.4	4.8	3.83	8.9	5.1	9.1
10	538	3820	712	2022	276	247	721	189	2742
11	278.41	221.77	789.52	373.48	543.49	551.44	471.02	192.56	688.34
12	31.9	50.45	44.45	41.12	39.13	37.5	22.3	24.9	42.54
13	31.3	62.6	72	70.2	49.8	97.5	56.6	74.6	70.4

In [11]:
```
df.shape
```

Out[11]:
```
(13, 9)
```

In [12]:
```
weights = list(df1["subjective_weights"])
```

In [13]:
```
weights.insert(12,0.0358)
print(weights)
```

```
[0.0327, 0.0412, 0.0426, 0.0902, 0.0419, 0.0336, 0.040999999999999995, 0.0886,
0.0325, 0.0334, 0.1007, 0.0704, 0.0358, 0.0377]
```

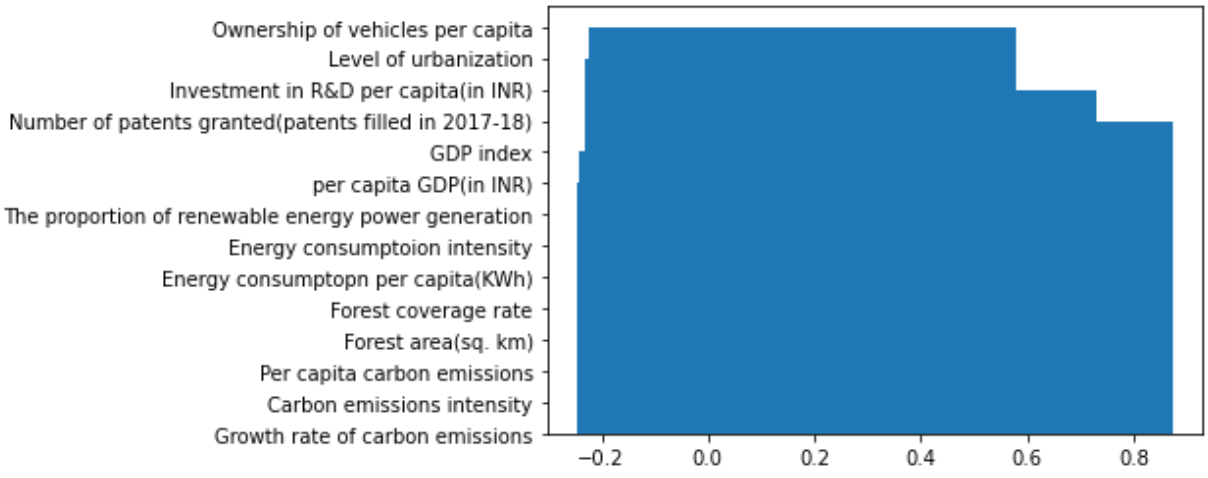
In [14]:
```
sum =0
for i in weights:
    t = i**2
    sum = sum + t
root = sum**0.5
for i in range(len(weights)):
    weights[i] = weights[i]/root
print(weights)
subjective_weights = weights
```

```
[0.15397069032917832, 0.19399365264716045, 0.2005856699701222, 0.42471425895082
215, 0.1972896613086413, 0.1582084157510823, 0.1930519358867373, 0.417180524867
43723, 0.1530289735687552, 0.15726669899065918, 0.47415438887303535, 0.33148429
96689344, 0.16856730011573648, 0.17751360933975602]
```

In [15]:
```
states = list(df1["STATES"])
states.insert(12,"Level of urbanization")
print(states)
```

```
['Growth rate of carbon emissions', 'Carbon emissions intensity', 'Per capita c
arbon emissions', 'Forest area(sq. km)', 'Forest coverage rate', 'Energy consum
ptopn per capita(KWh)', 'Energy consumptoion intensity', 'The proportion of ren
ewable energy power generation', 'per capita GDP(in INR)', 'GDP index', 'Number
of patents granted(patents filled in 2017-18)', 'Investment in R&D per capita(i
n INR)', 'Level of urbanization', 'Ownership of vehicles per capita']
```

In [16]:
```
x = weights
y = states
plt.bar(x,y)
plt.show()
```



setting types based on the Criteria types profit = +1 and cost = -1

In [17]:
```
types = np.array([-1,-1,-1,+1,+1,-1,-1,+1,+1,+1,+1,+1,-1,-1])
```

In [42]:
```
df.to_csv("final1")
```

In [43]:
```
df = pd.read_csv("am.txt")
df
```

Out[43]:

	Unnamed: 0	W.B	Mah	Guj	Kar	A.P	Pun
0	0	5.0000	8.00000	14.00000	8.000000	7.00000	19.0000
1	1	49.5000	27.50000	48.30000	42.700000	92.60000	35.0000
2	2	197.0400	0.04670	0.04420	0.063100	0.04930	0.0743
3	3	16901.5100	50777.56000	14857.33000	38575.480000	29137.40000	1848.6300
4	4	19.0400	16.50000	7.57000	20.190000	17.88000	3.6700
5	5	703.0000	1424.00000	2378.00000	1396.000000	1480.00000	2046.0000
6	6	0.0046	0.00061	0.15002	0.008671	0.02151	0.0105
7	7	12.3300	32.73000	44.75000	65.570000	39.78000	33.6600
8	8	121463.0000	202130.00000	213936.00000	223175.000000	168480.00000	151491.0000
9	9	7.5000	6.20000	13.10000	8.400000	4.80000	3.8300
10	10	538.0000	3820.00000	712.00000	2022.000000	276.00000	247.0000
11	11	278.4100	221.77000	789.52000	373.480000	543.49000	551.4400
12	12	31.9000	50.45000	44.45000	41.120000	39.13000	37.5000
13	13	31.3000	62.60000	72.00000	70.200000	49.80000	97.5000

final matrix

In [44]:
```
matrix = df[df.columns[1::]].to_numpy()
matrix
matrix = matrix.astype(float)
```

In [45]:
```
matrix = matrix.T
```

In [46]:
```
df.columns
```

Out[46]:
```
Index(['Unnamed: 0', 'W.B', 'Mah', 'Guj', 'Kar', 'A.P', 'Pun', 'U.P',
       'Rajasthan', 'T.N'],
      dtype='object')
```

normalization of the matrix

In [47]:
```
normalization_methods = [
    norm.minmax_normalization,
    norm.max_normalization,
    norm.sum_normalization,
    norm.vector_normalization,
    norm.logaritmic_normalization
]
norm_list = []
for method in normalization_methods:
    nmatrix = norm.normalize_matrix(matrix, method, types)
    norm_list.append(nmatrix)
```

In [54]:
```
norm_list[3]
```

Out[54]:
```
array([[0.16874355, 0.04023754, 0.03053517, 0.08051594, 0.16427955,
        0.19243148, 0.08829305, 0.01823154, 0.07888563, 0.10322581,
        0.04763591, 0.04704491, 0.32862625],
       [0.10546472, 0.0702104 , 0.12907158, 0.24189571, 0.14236411,
        0.09499953, 0.67691338, 0.06381039, 0.13533732, 0.11948387,
        0.33823269, 0.03030588, 0.08263701],
       [0.06026556, 0.16579805, 0.136372  , 0.07086356, 0.06531493,
        0.05688786, 0.00270765, 0.09571559, 0.1391596 , 0.12      ,
        0.06304232, 0.19314324, 0.04901467],
       [0.10546472, 0.09691013, 0.09552524, 0.18376707, 0.17351165,
        0.09690496, 0.04668368, 0.2260711 , 0.15441745, 0.12258065,
        0.17903311, 0.09604588, 0.06205941],
       [0.12053111, 0.16001439, 0.12226456, 0.13880565, 0.15427092,
        0.09140495, 0.01889061, 0.23245214, 0.11782149, 0.1203871 ,
        0.02443775, 0.10329912, 0.15753544],
       [0.14061963, 0.07130175, 0.08112574, 0.00880656, 0.03166523,
        0.06611893, 0.03868076, 0.06836828, 0.1158894 , 0.0516129 ,
        0.02426067, 0.31129145, 0.04361398],
       [0.03835081, 0.18106892, 0.16514089, 0.07053161, 0.05176877,
        0.22323322, 0.04463165, 0.0319052 , 0.0463734 , 0.08387097,
        0.06383921, 0.03557168, 0.13991018],
       [0.04963046, 0.1349141 , 0.15298585, 0.07922017, 0.0419327 ,
        0.1055221 , 0.03905269, 0.198268  , 0.07714806, 0.10812903,
        0.01673455, 0.04171567, 0.08804317],
       [0.21092944, 0.07954473, 0.08697897, 0.12559373, 0.17489215,
        0.07249696, 0.04414652, 0.06517776, 0.13496766, 0.17070968,
        0.24278378, 0.14158217, 0.04855989]])
```

In [ ]:
```

```

Calculating the objective_weights using the standard deviation method

non normalized

In [48]:
```
objective_stdv0 = mcdm_weights.standard_deviation_weights(matrix)
objective_stdv0
```

Out[48]:
```
array([9.20037475e-05, 2.76009274e-04, 9.18310800e-04, 2.05115756e-01,
       9.97193508e-05, 8.11060856e-03, 6.60539213e-07, 2.56324038e-04,
       7.63578231e-01, 3.98222591e-05, 1.82236223e-02, 2.90349276e-03,
       1.28671117e-04, 2.56768145e-04])
```

norm.minmax_normalization,

norm.max_normalization,

norm.sum_normalization,

norm.vector_normalization,

norm.logaritmic_normalization

In [49]:
```
objective_stdv1 = mcdm_weights.standard_deviation_weights(norm_list[0])
objective_stdv1
```

Out[49]:
```
array([0.07804365, 0.0647362 , 0.07117382, 0.06400853, 0.0917225 ,
       0.06988666, 0.06750306, 0.07351159, 0.07391034, 0.06559197,
       0.0766325 , 0.07426429, 0.06979225, 0.05922263])
```

In [50]:
```
objective_stdv2 = mcdm_weights.standard_deviation_weights(norm_list[1])
objective_stdv2
```

Out[50]:
```
array([0.07950775, 0.05666819, 0.08860576, 0.0767987 , 0.0935303 ,
       0.06484377, 0.08370976, 0.07432087, 0.0650481 , 0.0577938 ,
       0.09069806, 0.06991724, 0.04848937, 0.05006834])
```

In [51]:
```
objective_stdv3 = mcdm_weights.standard_deviation_weights(norm_list[2])
objective_stdv3
```

Out[51]:
```
array([0.07118403, 0.04152524, 0.05517877, 0.07529295, 0.06616019,
       0.06204167, 0.22849562, 0.05524271, 0.03985939, 0.04584234,
       0.12462011, 0.05442989, 0.03535316, 0.04477394])
```

In [52]:
```
objective_stdv4 = mcdm_weights.standard_deviation_weights(norm_list[3])
objective_stdv4
```

Out[52]:
```
array([0.07184938, 0.05400845, 0.14319839, 0.07748235, 0.07019613,
       0.05313678, 0.13253183, 0.06060421, 0.04549767, 0.05158571,
       0.1064112 , 0.05985176, 0.03455021, 0.03909592])
```

In [53]:
```
objective_stdv5 = mcdm_weights.standard_deviation_weights(norm_list[4])
objective_stdv5
```

Out[53]:
```
array([0.02545093, 0.00925847, 0.1285619 , 0.07465679, 0.22163115,
       0.00608004, 0.02945866, 0.12366206, 0.02582013, 0.15112951,
       0.12627431, 0.06337073, 0.00725759, 0.00738774])
```

In [54]:
```
dict1 = {"non normalized":objective_stdv0,"minmax_normalization":objective_s
    "max_normalization":objective_stdv2,
    "sum_normalization":objective_stdv3,
    "vector_normalization":objective_stdv4,
    "logaritmic_normalization":objective_stdv5}
stdv_df = pd.DataFrame(dict1)
```

objective weights for standard deviation

In [55]:
```
1 stdv_df
```

Out[55]:

	non normalized	minmax_normalization	max_normalization	sum_normalization	vector_normalization
0	9.200375e-05	0.078044	0.079508	0.071184	0.071849
1	2.760093e-04	0.064736	0.056668	0.041525	0.054008
2	9.183108e-04	0.071174	0.088606	0.055179	0.143198
3	2.051158e-01	0.064009	0.076799	0.075293	0.077482
4	9.971935e-05	0.091723	0.093530	0.066160	0.070196
5	8.110609e-03	0.069887	0.064844	0.062042	0.053137
6	6.605392e-07	0.067503	0.083710	0.228496	0.132532
7	2.563240e-04	0.073512	0.074321	0.055243	0.060604
8	7.635782e-01	0.073910	0.065048	0.039859	0.045498
9	3.982226e-05	0.065592	0.057794	0.045842	0.051586
10	1.822362e-02	0.076632	0.090698	0.124620	0.106411
11	2.903493e-03	0.074264	0.069917	0.054430	0.059852
12	1.286711e-04	0.069792	0.048489	0.035353	0.034550
13	2.567681e-04	0.059223	0.050068	0.044774	0.039096

In [57]:
```
1 for i in stdv_df:
2     stdv_df[i] = (stdv_df[i]+weights)/2
```

In [58]:
```
1 stdv_df
```

Out[58]:

	non normalized	minmax_normalization	max_normalization	sum_normalization	vector_normalization
0	0.077031	0.116007	0.116739	0.112577	0.112910
1	0.097135	0.129365	0.125331	0.117759	0.124001
2	0.100752	0.135880	0.144596	0.127882	0.171892
3	0.314915	0.244361	0.250756	0.250004	0.251098
4	0.098695	0.144506	0.145410	0.131725	0.133743
5	0.083160	0.114048	0.111526	0.110125	0.105673
6	0.096526	0.130277	0.138381	0.210774	0.162792
7	0.208718	0.245346	0.245751	0.236212	0.238892
8	0.458304	0.113470	0.109039	0.096444	0.099263
9	0.078653	0.111429	0.107530	0.101555	0.104426
10	0.246189	0.275393	0.282426	0.299387	0.290283
11	0.167194	0.202874	0.200701	0.192957	0.195668
12	0.084348	0.119180	0.108528	0.101960	0.101559
13	0.088885	0.118368	0.113791	0.111144	0.108305

Calculating the objective_weights using the entropy method

In [59]:
```
1 h = 1/math.log(9)
2 h
```

Out[59]:
```
0.45511961331341866
```

non normalized

In [60]:
```
1 objective_entropy0 = mcdm_weights.entropy_weights(matrix)
```

In [61]:
```
1 objective_entropy0
```

Out[61]:
```
array([0.03171869, 0.01453517, 0.47765957, 0.04123877, 0.03312549,
       0.01624145, 0.20522134, 0.02281071, 0.01191184, 0.01391356,
       0.09639667, 0.02073328, 0.0062934 , 0.00820006])
```

normalization - minmax normalization

In [62]:
```
1 objective_entropy1 = mcdm_weights.entropy_weights(norm_list[0])
```

In [63]:
```
1 objective_entropy1
```

Out[63]:
```
array([0.07142857, 0.07142857, 0.07142857, 0.07142857, 0.07142857,
       0.07142857, 0.07142857, 0.07142857, 0.07142857, 0.07142857,
       0.07142857, 0.07142857, 0.07142857, 0.07142857])
```

normalization - max normalization

In [64]:
```
1 objective_entropy2 = mcdm_weights.entropy_weights(norm_list[1])
```

In [65]:
```
1 objective_entropy2
```

Out[65]:
```
array([0.13336501, 0.13336501, 0.13336501, 0.01141092, 0.00916595,
       0.13336501, 0.13336501, 0.00631181, 0.00329605, 0.00384993,
       0.02667331, 0.00573698, 0.13336501, 0.13336501])
```

normalization - sum normalization

In [66]:
```
1 objective_entropy3 = mcdm_weights.entropy_weights(norm_list[2])
```

In [67]:
```
1 objective_entropy3
```

Out[67]:
```
array([0.0587459 , 0.02095732, 0.05648031, 0.0725612 , 0.05828558,
       0.04116053, 0.36508986, 0.04013633, 0.02095935, 0.02448144,
       0.16961365, 0.03648101, 0.01401327, 0.02103425])
```

normalization - vector_normalization

In [68]:
```
1 objective_entropy4 = mcdm_weights.entropy_weights(norm_list[3])
```

In [69]:
```
1 objective_entropy4
```

Out[69]:
```
array([0.0187327 , 0.01163798, 0.08480304, 0.13535804, 0.10872783,
       0.01043164, 0.07594827, 0.07487162, 0.03909826, 0.04566848,
       0.31640288, 0.06805287, 0.00448869, 0.0057777 ])
```

normalization - logarithmic normalization

In [70]:
```
objective_entropy5 = mcdm_weights.entropy_weights(norm_list[4])
```

In [71]:
```
objective_entropy5
```

Out[71]:
```
array([4.81887842e-03, 6.39081017e-04, 1.10041907e-01, 4.36840910e-02,
       3.88220231e-01, 2.74670809e-04, 6.46511519e-03, 1.19636151e-01,
       5.01987815e-03, 1.72297665e-01, 1.17932624e-01, 3.01734670e-02,
       3.91210219e-04, 4.05030274e-04])
```

In [72]:
```
dict2 = {"non normalized":objective_entropy0,"minmax_normalization":objectiv
    "max_normalization":objective_entropy2,
    "sum_normalization":objective_entropy3,
    "vector_normalization":objective_entropy4,
    "logaritmic_normalization":objective_entropy5}
entropy_df = pd.DataFrame(dict2)
```

objective weights for entropy

In [73]:
```
 entropy_df
```

Out[73]:

	non normalized	minmax_normalization	max_normalization	sum_normalization	vector_normalization
0	0.031719	0.071429	0.133365	0.058746	0.018733
1	0.014535	0.071429	0.133365	0.020957	0.011638
2	0.477660	0.071429	0.133365	0.056480	0.084803
3	0.041239	0.071429	0.011411	0.072561	0.135358
4	0.033125	0.071429	0.009166	0.058286	0.108728
5	0.016241	0.071429	0.133365	0.041161	0.010432
6	0.205221	0.071429	0.133365	0.365090	0.075948
7	0.022811	0.071429	0.006312	0.040136	0.074872
8	0.011912	0.071429	0.003296	0.020959	0.039098
9	0.013914	0.071429	0.003850	0.024481	0.045668
10	0.096397	0.071429	0.026673	0.169614	0.316403
11	0.020733	0.071429	0.005737	0.036481	0.068053
12	0.006293	0.071429	0.133365	0.014013	0.004489
13	0.008200	0.071429	0.133365	0.021034	0.005778

In [75]:
```
for i in entropy_df:
    entropy_df[i] = (entropy_df[i]+weights)/2
```

combined weights using entropy

In [76]:
```
1 entropy_df
```

Out[76]:

	non normalized	minmax_normalization	max_normalization	sum_normalization	vector_normalization
0	0.092845	0.112700	0.143668	0.106358	0.086352
1	0.104264	0.132711	0.163679	0.107475	0.102816
2	0.339123	0.136007	0.166975	0.128533	0.142694
3	0.232977	0.248071	0.218063	0.248638	0.280036
4	0.115208	0.134359	0.103228	0.127788	0.153009
5	0.087225	0.114818	0.145787	0.099684	0.084320
6	0.199137	0.132240	0.163208	0.279071	0.134500
7	0.219996	0.244305	0.211746	0.228658	0.246026
8	0.082470	0.112229	0.078163	0.086994	0.096064
9	0.085590	0.114348	0.080558	0.090874	0.101468
10	0.285276	0.272791	0.250414	0.321884	0.395279
11	0.176109	0.201456	0.168611	0.183983	0.199769
12	0.087430	0.119998	0.150966	0.091290	0.086528
13	0.092857	0.124471	0.155439	0.099274	0.091646

In [77]:
```
1 print(dir(mcdm_weights))
```

```
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', '_fake_normalization', 'angle_weights',
'cilos_weights', 'correlation_matrix', 'critic_weights', 'entropy_weights', 'eq
ual_weights', 'gini_weights', 'idocriw_weights', 'linear_normalization', 'merec
_weights', 'minmax_normalization', 'normalize_matrix', 'np', 'null_space', 'pea
rson', 'standard_deviation_weights', 'sum_normalization']
```

Topsis method

In [93]:
```
1 col = list(df.columns)
2 col.remove('Unnamed: 0')
```

In [94]:
```
1 topsis = mcdm_methods.TOPSIS()
```

In [95]:
```
topsis_methods = {
    'minmax': mcdm_methods.TOPSIS(norm.minmax_normalization),
    'max': mcdm_methods.TOPSIS(norm.max_normalization),
    'sum': mcdm_methods.TOPSIS(norm.sum_normalization),
    'vector': mcdm_methods.TOPSIS(norm.vector_normalization),
    'log': mcdm_methods.TOPSIS(norm.logaritmic_normalization),
}
```

In [96]:
```
results = {}
for name, function in topsis_methods.items():
    results[name] = function(matrix, vec, types)
```

In [97]:
```
print(tabulate([[name, *np.round(pref, 2)] for name, pref in results.items()
               headers=['Method'] + [f'A{i+1}' for i in range(10)]))
```

```
Method      A1    A2    A3    A4    A5    A6    A7    A8    A9
--------  ----  ----  ----  ----  ----  ----  ----  ----  ----
minmax    0.36  0.64  0.45  0.65  0.45  0.34  0.37  0.41  0.68
max       0.34  0.69  0.42  0.66  0.45  0.33  0.36  0.4   0.68
sum       0.22  0.78  0.27  0.46  0.27  0.16  0.23  0.24  0.5
vector    0.34  0.76  0.41  0.67  0.46  0.39  0.43  0.43  0.71
log       0.47  0.67  0.55  0.73  0.48  0.27  0.39  0.4   0.75
```

In [98]:
```
top = tabulate([[name, *rankdata(pref, reverse=True)] for name, pref in resu
               headers=['Method'] + [f'A{i+1}' for i in range(10)])
```

In [99]:
```
print(col)
```

```
['W.B', 'Mah', 'Guj', 'Kar', 'A.P', 'Pun', 'U.P', 'Rajasthan', 'T.N']
```

In [100]:
```
norm_list.insert(0,matrix)
```

In [101]:
```
entropy_df["subjective"] = weights
```

In [102]:
```
vec = np.array(list((entropy_df["vector_normalization"]+entropy_df["subjecti
```

In [103]:
```
m = topsis(matrix, vec, types)
```

In [104]:
```
m = list(m)
```

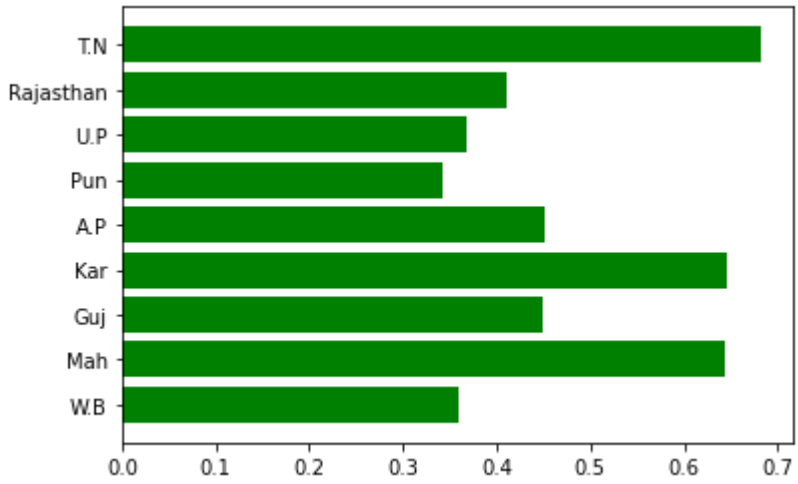
In [105]:
```
print(m)
```

```
[0.3586792108010278, 0.6429005650492241, 0.4486186695657274, 0.645737141353914
4, 0.4509100770400232, 0.342261727379166, 0.3674053780781845, 0.410971179132432
36, 0.6818350055960277]
```

In [106]:
```
1 print(col)
```

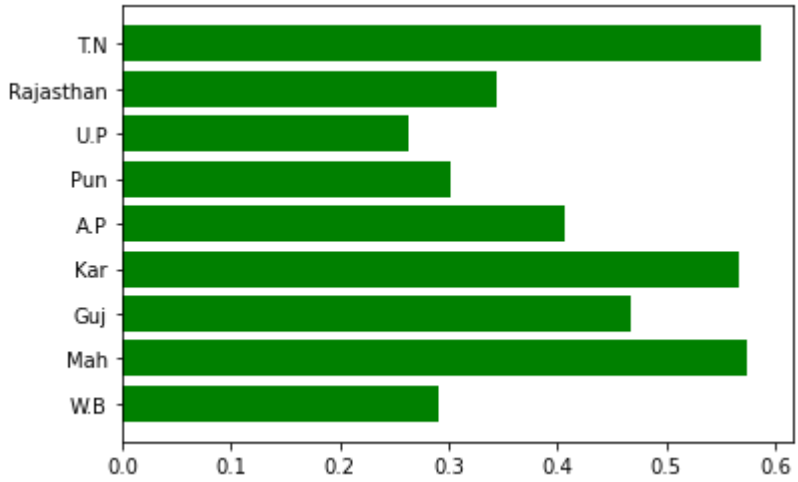
```
['W.B', 'Mah', 'Guj', 'Kar', 'A.P', 'Pun', 'U.P', 'Rajasthan', 'T.N']
```

In [107]:
```
1 plt.barh(col,m, color ='green')
2 plt.show()
```



In [108]:
```
1 g = list(topsis(norm_list[3],vec, types))
```

In [109]:
```
1 plt.barh(col,g, color ='green')
2 plt.show()
```



In [110]:
```
1 vikor = mcdm_methods.VIKOR()
2 v = vikor(matrix, vec, types)
```

In [111]:
```
vikor_methods = {
    'VIKOR': mcdm_methods.VIKOR(),
    'minmax': mcdm_methods.VIKOR(norm.minmax_normalization),
    'max': mcdm_methods.VIKOR(norm.max_normalization),
    'sum': mcdm_methods.VIKOR(norm.sum_normalization),
    'vector': mcdm_methods.VIKOR(norm.vector_normalization),
    'log': mcdm_methods.VIKOR(norm.logaritmic_normalization),
}
```

In [112]:
```
results = {}
for name, function in vikor_methods.items():
    results[name] = function(matrix, vec, types)
```

In [113]:
```
print(tabulate([[name, *np.round(pref, 2)] for name, pref in results.items()
                headers=['Method'] + [f'A{i+1}' for i in range(10)]))
```

Method	A1	A2	A3	A4	A5	A6	A7	A8	A9
VIKOR	0.82	0.2	0.7	0.11	0.76	0.99	0.79	0.89	0
minmax	0.82	0.2	0.7	0.11	0.76	0.99	0.79	0.89	0
max	0.82	0.2	0.7	0.11	0.76	0.99	0.79	0.89	0
sum	0.79	0.15	0.64	0.16	0.77	0.99	0.75	0.89	0.02
vector	0.82	0.2	0.7	0.11	0.76	0.99	0.79	0.89	0
log	0.53	0.3	0.33	0.06	0.65	0.93	0.55	0.9	0

In [114]:
```
copras = mcdm_methods.COPRAS()
cop = copras(matrix, vec, types)
```

ranks by vikor method

In [115]:
```
print(tabulate([[name, *rankdata(pref)] for name, pref in results.items()],
                headers=['Method'] + [f'A{i+1}' for i in range(10)]))
```

Method	A1	A2	A3	A4	A5	A6	A7	A8	A9
VIKOR	7	3	4	2	5	9	6	8	1
minmax	7	3	4	2	5	9	6	8	1
max	7	3	4	2	5	9	6	8	1
sum	7	2	4	3	6	9	5	8	1
vector	7	3	4	2	5	9	6	8	1
log	5	3	4	2	7	9	6	8	1

In [116]:
```
print(col)
```

```
['W.B', 'Mah', 'Guj', 'Kar', 'A.P', 'Pun', 'U.P', 'Rajasthan', 'T.N']
```

ranks by topsis method

In [117]:
```
print(top)
```

```
Method      A1    A2    A3    A4    A5    A6    A7    A8    A9
--------  ----  ----  ----  ----  ----  ----  ----  ----  ----
minmax       8     3     5     2     4     9     7     6     1
max          8     1     5     3     4     9     7     6     2
sum          8     1     4     3     5     9     7     6     2
vector       9     1     7     3     4     8     6     5     2
log          6     3     4     2     5     9     8     7     1
```

In [118]:
```
print(col)
```

```
['W.B', 'Mah', 'Guj', 'Kar', 'A.P', 'Pun', 'U.P', 'Rajasthan', 'T.N']
```

ranks by corpus method

In [119]:
```
print(tabulate([['Preference', *np.round(cop, 2)],
                ['Rank', *rrankdata(cop)]],
               headers=[''] + [f'A{i+1}' for i in range(10)]))
```

```
              A1    A2    A3    A4    A5    A6    A7    A8    A9
----------  ----  ----  ----  ----  ----  ----  ----  ----  ----
Preference  0.37     1  0.52  0.88  0.59  0.43  0.53  0.52  0.93
Rank        9        1  7     3     4     8     5     6     2
```

In [ ]:
```

```

In [ ]:
```

```