TOPSIS Ranking

May 28, 2021

1 TOPSIS Ranking

[1]:
```
import numpy as np                # for linear algebra
import pandas as pd               # for tabular output
from scipy.stats import rankdata # for ranking the candidates
```

1.1 Step 0 - Obtaining and processing the data

The data from the Excel sheet is saved into CSV files and stored in the `data` folder at the root of the project. The criteria, their rankings, the players' scores based on the mentioned criteria are stored in Numpy arrays and processed for the next step.

Note that an attribute can be beneficial attribute (in which case, we will want to maximize it's contribution) or a cost attribute (which we will need to minimize). We call the set of beneficial attributes J_1 and that of cost attributes $J_2 = J_1^C$.

[2]:
```
bowlers_data = {
    'weights': '../data/bowling_criteria.csv',
    'scores': '../data/bowlers.csv',
}
batsmen_data = {
    'weights': '../data/batting_criteria.csv',
    'scores': '../data/batsmen.csv',
}
data = batsmen_data
```

[3]:
```
attributes_data = pd.read_csv(data['weights'])
attributes_data
```

[3]:

	Name	Ranking	Ideally
0	SR	1	Higher
1	Avg	2	Higher
2	Runs	3	Higher
3	Inn	4	Higher
4	NO	5	Higher
5	6s	6	Higher
6	4s	7	Higher
7	100s	8	Higher

```
8    50s        9  Higher
9    Mat       10  Higher
10    HS       11  Higher
11    BF       12  Higher
```

[4]:
```
benefit_attributes = set()
attributes = []
ranks = []
n = 0

for i, row in attributes_data.iterrows():
    attributes.append(row['Name'])
    ranks.append(float(row['Ranking']))
    n += 1

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

ranks = np.array(ranks)
```

[5]:
```
weights = 2 * (n + 1 - ranks) / (n * (n + 1))
pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[5]:
```
        Weight
SR    0.153846
Avg   0.141026
Runs  0.128205
Inn   0.115385
NO    0.102564
6s    0.089744
4s    0.076923
100s  0.064103
50s   0.051282
Mat   0.038462
HS    0.025641
BF    0.012821
```

[6]:
```
original_dataframe = pd.read_csv(data['scores'])
candidates = original_dataframe['Name'].to_numpy()
raw_data = pd.DataFrame(original_dataframe, columns=attributes).to_numpy()

dimensions = raw_data.shape
m = dimensions[0]
n = dimensions[1]

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

[6]:

	SR	Avg	Runs	Inn	NO	6s	4s	100s	50s	Mat	HS	BF
AB de Villiers	154.00	44.20	442.0	13.0	3.0	26.0	31.0	0.0	5.0	13.0	82.0	287.0
Andre Russel	204.81	56.67	510.0	13.0	4.0	52.0	31.0	0.0	4.0	14.0	80.0	249.0
Ben Stokes	124.24	20.50	123.0	9.0	3.0	4.0	8.0	0.0	0.0	9.0	46.0	99.0
Chris Gayle	153.60	40.83	490.0	13.0	1.0	34.0	45.0	0.0	4.0	13.0	99.0	319.0
Chris Lynn	139.65	31.15	405.0	13.0	0.0	22.0	41.0	0.0	4.0	13.0	82.0	290.0
David Warner	143.86	69.20	692.0	12.0	2.0	21.0	57.0	1.0	8.0	12.0	100.0	481.0
Faf Du Plessis	123.36	36.00	396.0	12.0	1.0	15.0	36.0	0.0	3.0	12.0	96.0	321.0
Jonny Bairstow	157.24	55.63	445.0	10.0	2.0	18.0	48.0	1.0	2.0	10.0	114.0	283.0
Jos Buttler	151.70	38.88	311.0	8.0	0.0	14.0	38.0	0.0	3.0	8.0	89.0	205.0
Kane Williamson	120.00	22.29	156.0	9.0	2.0	5.0	12.0	0.0	1.0	9.0	70.0	130.0
Kieron Pollard	156.74	34.88	279.0	14.0	6.0	22.0	14.0	0.0	1.0	16.0	83.0	178.0
Marcus Stoinis	135.25	52.75	211.0	10.0	6.0	10.0	14.0	0.0	0.0	10.0	46.0	156.0
Moeen Ali	165.41	27.50	220.0	10.0	2.0	17.0	16.0	0.0	2.0	11.0	66.0	133.0
Quinton de Kock	132.91	35.27	529.0	16.0	1.0	25.0	45.0	0.0	4.0	16.0	81.0	398.0
Shane Watson	127.56	23.41	398.0	17.0	0.0	20.0	42.0	0.0	3.0	17.0	96.0	312.0
Steve Smith	116.00	39.88	319.0	10.0	2.0	4.0	30.0	0.0	3.0	12.0	73.0	275.0

1.2 Step 1 - Normalizing the ratings

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[7]:
```
divisors = np.empty(n)
for j in range(n):
    column = raw_data[:,j]
    divisors[j] = np.sqrt(column @ column)
```

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

[7]:

	SR	Avg	Runs	Inn	NO	6s
AB de Villiers	0.264163	0.266301	0.277322	0.269260	0.264135	0.287807
Andre Russel	0.351320	0.341432	0.319987	0.269260	0.352180	0.575614
Ben Stokes	0.213114	0.123511	0.077173	0.186411	0.264135	0.044278
Chris Gayle	0.263477	0.245997	0.307438	0.269260	0.088045	0.376363
Chris Lynn	0.239548	0.187676	0.254107	0.269260	0.000000	0.243529
David Warner	0.246770	0.416924	0.434178	0.248548	0.176090	0.232460
Faf Du Plessis	0.211605	0.216897	0.248460	0.248548	0.088045	0.166043
Jonny Bairstow	0.269721	0.335166	0.279204	0.207123	0.176090	0.199251
Jos Buttler	0.260218	0.234249	0.195129	0.165699	0.000000	0.154973
Kane Williamson	0.205841	0.134295	0.097878	0.186411	0.176090	0.055348
Kieron Pollard	0.268863	0.210149	0.175052	0.289973	0.528271	0.243529
Marcus Stoinis	0.232000	0.317815	0.132387	0.207123	0.528271	0.110695
Moeen Ali	0.283735	0.165685	0.138034	0.207123	0.176090	0.188182
Quinton de Kock	0.227987	0.212499	0.331908	0.331397	0.088045	0.276738
Shane Watson	0.218809	0.141043	0.249715	0.352110	0.000000	0.221390
Steve Smith	0.198980	0.240274	0.200149	0.207123	0.176090	0.044278

	4s	100s	50s	Mat	HS	BF
AB de Villiers	0.222189	0.000000	0.354441	0.260889	0.245830	0.259994
Andre Russel	0.222189	0.000000	0.283552	0.280957	0.239834	0.225570
Ben Stokes	0.057339	0.000000	0.000000	0.180615	0.137905	0.089684
Chris Gayle	0.322533	0.000000	0.283552	0.260889	0.296795	0.288983
Chris Lynn	0.293863	0.000000	0.283552	0.260889	0.245830	0.262712
David Warner	0.408542	0.707107	0.567105	0.240820	0.299792	0.435740
Faf Du Plessis	0.258026	0.000000	0.212664	0.240820	0.287801	0.290795
Jonny Bairstow	0.344035	0.707107	0.141776	0.200683	0.341763	0.256371
Jos Buttler	0.272361	0.000000	0.212664	0.160547	0.266815	0.185710
Kane Williamson	0.086009	0.000000	0.070888	0.180615	0.209855	0.117767
Kieron Pollard	0.100344	0.000000	0.070888	0.321094	0.248828	0.161251
Marcus Stoinis	0.100344	0.000000	0.000000	0.200683	0.137905	0.141321
Moeen Ali	0.114678	0.000000	0.141776	0.220752	0.197863	0.120485
Quinton de Kock	0.322533	0.000000	0.283552	0.321094	0.242832	0.360550
Shane Watson	0.301031	0.000000	0.212664	0.341162	0.287801	0.282642
Steve Smith	0.215022	0.000000	0.212664	0.240820	0.218848	0.249123

1.3 Step 2 - Calculating the Weighted Normalized Ratings

$$v_{ij} = w_j r_{ij}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[8]:
```
raw_data *= weights
pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[8]:

	SR	Avg	Runs	Inn	NO	6s
AB de Villiers	0.040640	0.037555	0.035554	0.031068	0.027091	0.025829
Andre Russel	0.054049	0.048151	0.041024	0.031068	0.036121	0.051658
Ben Stokes	0.032787	0.017418	0.009894	0.021509	0.027091	0.003974
Chris Gayle	0.040535	0.034692	0.039415	0.031068	0.009030	0.033776
Chris Lynn	0.036854	0.026467	0.032578	0.031068	0.000000	0.021855
David Warner	0.037965	0.058797	0.055664	0.028679	0.018061	0.020862
Faf Du Plessis	0.032555	0.030588	0.031854	0.028679	0.009030	0.014901
Jonny Bairstow	0.041496	0.047267	0.035795	0.023899	0.018061	0.017882
Jos Buttler	0.040034	0.033035	0.025017	0.019119	0.000000	0.013908
Kane Williamson	0.031668	0.018939	0.012549	0.021509	0.018061	0.004967
Kieron Pollard	0.041364	0.029636	0.022443	0.033458	0.054182	0.021855
Marcus Stoinis	0.035692	0.044820	0.016973	0.023899	0.054182	0.009934
Moeen Ali	0.043652	0.023366	0.017697	0.023899	0.018061	0.016888
Quinton de Kock	0.035075	0.029968	0.042552	0.038238	0.009030	0.024835
Shane Watson	0.033663	0.019891	0.032015	0.040628	0.000000	0.019868
Steve Smith	0.030612	0.033885	0.025660	0.023899	0.018061	0.003974

	4s	100s	50s	Mat	HS	BF
AB de Villiers	0.017091	0.000000	0.018176	0.010034	0.006303	0.003333
Andre Russel	0.017091	0.000000	0.014541	0.010806	0.006150	0.002892
Ben Stokes	0.004411	0.000000	0.000000	0.006947	0.003536	0.001150
Chris Gayle	0.024810	0.000000	0.014541	0.010034	0.007610	0.003705
Chris Lynn	0.022605	0.000000	0.014541	0.010034	0.006303	0.003368
David Warner	0.031426	0.045327	0.029082	0.009262	0.007687	0.005586
Faf Du Plessis	0.019848	0.000000	0.010906	0.009262	0.007380	0.003728
Jonny Bairstow	0.026464	0.045327	0.007271	0.007719	0.008763	0.003287
Jos Buttler	0.020951	0.000000	0.010906	0.006175	0.006841	0.002381
Kane Williamson	0.006616	0.000000	0.003635	0.006947	0.005381	0.001510
Kieron Pollard	0.007719	0.000000	0.003635	0.012350	0.006380	0.002067
Marcus Stoinis	0.007719	0.000000	0.000000	0.007719	0.003536	0.001812
Moeen Ali	0.008821	0.000000	0.007271	0.008490	0.005073	0.001545
Quinton de Kock	0.024810	0.000000	0.014541	0.012350	0.006226	0.004622
Shane Watson	0.023156	0.000000	0.010906	0.013122	0.007380	0.003624
Steve Smith	0.016540	0.000000	0.010906	0.009262	0.005611	0.003194

1.4 Step 3 - Identifying PIS (A^*) and NIS (A^-)

$$A^* = \{v_1^*, v_2^*, \ldots, v_n^*\}$$
$$A^- = \{v_1^-, v_2^-, \ldots, v_n^-\}$$

And we define

$$v_j^* = \max(v_{ij}), \text{ if} j \in J_1$$

$$v_j^* = \min(v_{ij}), \text{ if} j \in J_2$$
$$v_j^- = \min(v_{ij}), \text{ if} j \in J_1$$
$$v_j^- = \max(v_{ij}), \text{ if} j \in J_2$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[9]:
```
a_pos = np.zeros(n)
a_neg = np.zeros(n)
for j in range(n):
    column = raw_data[:,j]
    max_val = np.max(column)
    min_val = np.min(column)

    # See if we want to maximize benefit or minimize cost (for PIS)
    if j in benefit_attributes:
        a_pos[j] = max_val
        a_neg[j] = min_val
    else:
        a_pos[j] = min_val
        a_neg[j] = max_val

pd.DataFrame(data=[a_pos, a_neg], index=["$A^*$", "$A^-$"], columns=attributes)
```

[9]:

	SR	Avg	Runs	Inn	NO	6s	4s
A^*	0.054049	0.058797	0.055664	0.040628	0.054182	0.051658	0.031426
A^-	0.030612	0.017418	0.009894	0.019119	0.000000	0.003974	0.004411

	100s	50s	Mat	HS	BF
A^*	0.045327	0.029082	0.013122	0.008763	0.005586
A^-	0.000000	0.000000	0.006175	0.003536	0.001150

1.5 Step 4 and 5 - Calculating Separation Measures and Similarities to PIS

The separation or distance between the alternatives can be measured by the n-dimensional Euclidean distance. The separation from the PIS A^* and NIS A^- are S^* and S^- respectively.

$$S_i^* = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^*\right)^2}$$
$$S_i^- = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^-\right)^2}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

We also calculate

$$C_i^* = \frac{S_i^-}{S_i^* + S_i^-}, \text{ where } i = 1, 2, \ldots, m$$

[10]:
```
sp = np.zeros(m)
sn = np.zeros(m)
cs = np.zeros(m)

for i in range(m):
    diff_pos = raw_data[i] - a_pos
    diff_neg = raw_data[i] - a_neg
    sp[i] = np.sqrt(diff_pos @ diff_pos)
    sn[i] = np.sqrt(diff_neg @ diff_neg)
    cs[i] = sn[i] / (sp[i] + sn[i])

pd.DataFrame(data=zip(sp, sn, cs), index=candidates, columns=["$S^*$", "$S^-$",␣
↪"$C^*$"])
```

[10]:

	S^*	S^-	C^*
AB de Villiers	0.070196	0.055112	0.439813
Andre Russel	0.056888	0.081165	0.587927
Ben Stokes	0.106526	0.027294	0.203959
Chris Gayle	0.076169	0.055195	0.420169
Chris Lynn	0.090296	0.040840	0.311430
David Warner	0.051679	0.090778	0.637231
Faf Du Plessis	0.088862	0.036606	0.291756
Jonny Bairstow	0.062814	0.069648	0.525796
Jos Buttler	0.095810	0.032634	0.254072
Kane Williamson	0.105748	0.019120	0.153120
Kieron Pollard	0.079607	0.062885	0.441323
Marcus Stoinis	0.087082	0.061924	0.415581
Moeen Ali	0.093227	0.029365	0.239533
Quinton de Kock	0.080278	0.053022	0.397761
Shane Watson	0.094307	0.041946	0.307855
Steve Smith	0.092551	0.033953	0.268397

1.6 Step 6 - Ranking the candidates/alternatives

We choose the candidate with the maximum C^* or rank all the alternatives in descending order according to their C^* values. This process can also be done for the S^* and S^- values.

[11]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage[::-1]
```

[12]:
```
cs_order = rank_according_to(cs)
sp_order = rank_according_to(sp)
sn_order = rank_according_to(sn)

pd.DataFrame(data=zip(cs_order, sp_order[::-1], sn_order), index=range(1, m +␣
↪1),
             columns=["$C^*$", "$S^*$", "$S^-$"])
```

[12]:

	C^*	S^*	S^-
1	David Warner	David Warner	David Warner
2	Andre Russel	Andre Russel	Andre Russel
3	Jonny Bairstow	Jonny Bairstow	Jonny Bairstow
4	Kieron Pollard	AB de Villiers	Kieron Pollard
5	AB de Villiers	Chris Gayle	Marcus Stoinis
6	Chris Gayle	Kieron Pollard	Chris Gayle
7	Marcus Stoinis	Quinton de Kock	AB de Villiers
8	Quinton de Kock	Marcus Stoinis	Quinton de Kock
9	Chris Lynn	Faf Du Plessis	Shane Watson
10	Shane Watson	Chris Lynn	Chris Lynn
11	Faf Du Plessis	Steve Smith	Faf Du Plessis
12	Steve Smith	Moeen Ali	Steve Smith
13	Jos Buttler	Shane Watson	Jos Buttler
14	Moeen Ali	Jos Buttler	Moeen Ali
15	Ben Stokes	Kane Williamson	Ben Stokes
16	Kane Williamson	Ben Stokes	Kane Williamson

[13]:
```
print("The best candidate/alternative according to C* is " + cs_order[0])
print("The preferences in descending order are " + ", ".join(cs_order) + ".")
```

```
The best candidate/alternative according to C* is David Warner
The preferences in descending order are David Warner, Andre Russel, Jonny
Bairstow, Kieron Pollard, AB de Villiers, Chris Gayle, Marcus Stoinis, Quinton
de Kock, Chris Lynn, Shane Watson, Faf Du Plessis, Steve Smith, Jos Buttler,
Moeen Ali, Ben Stokes, Kane Williamson.
```

TOPSIS Ranking

May 28, 2021

1 TOPSIS Ranking

[1]:
```
import numpy as np                # for linear algebra
import pandas as pd               # for tabular output
from scipy.stats import rankdata # for ranking the candidates
```

1.1 Step 0 - Obtaining and processing the data

The data from the Excel sheet is saved into CSV files and stored in the `data` folder at the root of the project. The criteria, their rankings, the players' scores based on the mentioned criteria are stored in Numpy arrays and processed for the next step.

Note that an attribute can be beneficial attribute (in which case, we will want to maximize it's contribution) or a cost attribute (which we will need to minimize). We call the set of beneficial attributes J_1 and that of cost attributes $J_2 = J_1^C$.

[2]:
```
bowlers_data = {
    'weights': '../data/bowling_criteria.csv',
    'scores': '../data/bowlers.csv',
}
batsmen_data = {
    'weights': '../data/batting_criteria.csv',
    'scores': '../data/batsmen.csv',
}
data = bowlers_data
```

[3]:
```
attributes_data = pd.read_csv(data['weights'])
attributes_data
```

[3]:

	Name	Ranking	Ideally
0	SR	1	Lower
1	Econ	2	Lower
2	Avg	3	Lower
3	Wkts	4	Higher
4	Runs	5	Lower
5	Inns	6	Higher
6	TBB	7	Higher
7	4w	8	Higher

```
8   Mat        9  Higher
```

[4]:
```
benefit_attributes = set()
attributes = []
ranks = []
n = 0

for i, row in attributes_data.iterrows():
    attributes.append(row['Name'])
    ranks.append(float(row['Ranking']))
    n += 1

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

ranks = np.array(ranks)
```

[5]:
```
weights = 2 * (n + 1 - ranks) / (n * (n + 1))
pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[5]:

	Weight
SR	0.200000
Econ	0.177778
Avg	0.155556
Wkts	0.133333
Runs	0.111111
Inns	0.088889
TBB	0.066667
4w	0.044444
Mat	0.022222

[6]:
```
original_dataframe = pd.read_csv(data['scores'])
candidates = original_dataframe['Name'].to_numpy()
raw_data = pd.DataFrame(original_dataframe, columns=attributes).to_numpy()

dimensions = raw_data.shape
m = dimensions[0]
n = dimensions[1]

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

[6]:

	SR	Econ	Avg	Wkts	Runs	Inns	TBB	4w	Mat
Andre Russell	16.45	9.51	26.09	11.0	287.0	12.0	181.0	0.0	14.0
Ben Stokes	16.83	11.23	31.50	6.0	189.0	6.0	101.0	0.0	9.0
Chris Morris	15.23	9.27	23.54	13.0	306.0	9.0	198.0	0.0	9.0
Dwayne Bravo	22.45	8.02	30.00	11.0	330.0	12.0	247.0	0.0	12.0
Imran Tahir	14.85	6.70	16.58	26.0	431.0	17.0	386.0	2.0	17.0

Jofra Archer	23.45	6.77	26.45	11.0	291.0	11.0	258.0	0.0	11.0
Kagiso Rabada	11.28	7.83	14.72	25.0	368.0	12.0	282.0	2.0	12.0
Keemo Paul	18.11	8.72	26.33	9.0	237.0	8.0	163.0	0.0	8.0
Lasith Malinga	16.81	9.77	27.38	16.0	438.0	12.0	269.0	2.0	12.0
Moeen Ali	25.00	6.76	28.17	6.0	169.0	9.0	150.0	0.0	11.0
Mohammad Nabi	21.88	6.65	24.25	8.0	194.0	8.0	175.0	1.0	8.0
Rashid Khan	21.18	6.28	22.18	17.0	377.0	15.0	360.0	0.0	15.0
Sam Curran	19.80	9.79	32.30	10.0	323.0	9.0	198.0	1.0	9.0
Sunil Narine	26.60	7.83	34.70	10.0	347.0	12.0	266.0	0.0	12.0
Trent Boult	22.80	8.58	32.60	5.0	163.0	5.0	114.0	0.0	5.0

1.2 Step 1 - Normalizing the ratings

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[7]:
```
divisors = np.empty(n)
for j in range(n):
    column = raw_data[:,j]
    divisors[j] = np.sqrt(column @ column)

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

[7]:

	SR	Econ	Avg	Wkts	Runs	Inns	\
Andre Russell	0.212905	0.293421	0.249384	0.207142	0.239655	0.283870	
Ben Stokes	0.217824	0.346490	0.301096	0.112987	0.157822	0.141935	
Chris Morris	0.197115	0.286016	0.225010	0.244804	0.255521	0.212902	
Dwayne Bravo	0.290561	0.247449	0.286758	0.207142	0.275561	0.283870	
Imran Tahir	0.192197	0.206721	0.158482	0.489608	0.359900	0.402149	
Jofra Archer	0.303503	0.208881	0.252825	0.207142	0.242995	0.260214	
Kagiso Rabada	0.145992	0.241586	0.140703	0.470777	0.307293	0.283870	
Keemo Paul	0.234390	0.269046	0.251678	0.169480	0.197903	0.189246	
Lasith Malinga	0.217565	0.301443	0.261715	0.301297	0.365745	0.283870	
Moeen Ali	0.323564	0.208573	0.269266	0.112987	0.141121	0.212902	
Mohammad Nabi	0.283184	0.205179	0.231796	0.150649	0.161997	0.189246	
Rashid Khan	0.274124	0.193763	0.212010	0.320129	0.314808	0.354837	
Sam Curran	0.256263	0.302060	0.308743	0.188311	0.269716	0.212902	
Sunil Narine	0.344272	0.241586	0.331684	0.188311	0.289757	0.283870	
Trent Boult	0.295091	0.264727	0.311610	0.094155	0.136111	0.118279	

	TBB	4w	Mat
Andre Russell	0.197151	0.000000	0.319173
Ben Stokes	0.110012	0.000000	0.205182
Chris Morris	0.215668	0.000000	0.205182

Dwayne Bravo	0.269040	0.000000	0.273576
Imran Tahir	0.420443	0.534522	0.387567
Jofra Archer	0.281021	0.000000	0.250778
Kagiso Rabada	0.307163	0.534522	0.273576
Keemo Paul	0.177545	0.000000	0.182384
Lasith Malinga	0.293003	0.534522	0.273576
Moeen Ali	0.163385	0.000000	0.250778
Mohammad Nabi	0.190615	0.267261	0.182384
Rashid Khan	0.392123	0.000000	0.341971
Sam Curran	0.215668	0.267261	0.205182
Sunil Narine	0.289735	0.000000	0.273576
Trent Boult	0.124172	0.000000	0.113990

1.3 Step 2 - Calculating the Weighted Normalized Ratings

$$v_{ij} = w_j r_{ij}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[8]:
```
raw_data *= weights
pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[8]:

	SR	Econ	Avg	Wkts	Runs	Inns
Andre Russell	0.042581	0.052164	0.038793	0.027619	0.026628	0.025233
Ben Stokes	0.043565	0.061598	0.046837	0.015065	0.017536	0.012616
Chris Morris	0.039423	0.050847	0.035001	0.032641	0.028391	0.018925
Dwayne Bravo	0.058112	0.043991	0.044607	0.027619	0.030618	0.025233
Imran Tahir	0.038439	0.036750	0.024653	0.065281	0.039989	0.035747
Jofra Archer	0.060701	0.037134	0.039328	0.027619	0.026999	0.023130
Kagiso Rabada	0.029198	0.042949	0.021887	0.062770	0.034144	0.025233
Keemo Paul	0.046878	0.047830	0.039150	0.022597	0.021989	0.016822
Lasith Malinga	0.043513	0.053590	0.040711	0.040173	0.040638	0.025233
Moeen Ali	0.064713	0.037080	0.041886	0.015065	0.015680	0.018925
Mohammad Nabi	0.056637	0.036476	0.036057	0.020086	0.018000	0.016822
Rashid Khan	0.054825	0.034447	0.032979	0.042684	0.034979	0.031541
Sam Curran	0.051253	0.053700	0.048027	0.025108	0.029968	0.018925
Sunil Narine	0.068854	0.042949	0.051595	0.025108	0.032195	0.025233
Trent Boult	0.059018	0.047063	0.048473	0.012554	0.015123	0.010514

	TBB	4w	Mat
Andre Russell	0.013143	0.000000	0.007093
Ben Stokes	0.007334	0.000000	0.004560
Chris Morris	0.014378	0.000000	0.004560
Dwayne Bravo	0.017936	0.000000	0.006079
Imran Tahir	0.028030	0.023757	0.008613
Jofra Archer	0.018735	0.000000	0.005573
Kagiso Rabada	0.020478	0.023757	0.006079

Keemo Paul	0.011836	0.000000	0.004053
Lasith Malinga	0.019534	0.023757	0.006079
Moeen Ali	0.010892	0.000000	0.005573
Mohammad Nabi	0.012708	0.011878	0.004053
Rashid Khan	0.026142	0.000000	0.007599
Sam Curran	0.014378	0.011878	0.004560
Sunil Narine	0.019316	0.000000	0.006079
Trent Boult	0.008278	0.000000	0.002533

1.4 Step 3 - Identifying PIS (A^*) and NIS (A^-)

$$A^* = \{v_1^*, v_2^*, \ldots, v_n^*\}$$
$$A^- = \{v_1^-, v_2^-, \ldots, v_n^-\}$$

And we define

$$v_j^* = \max(v_{ij}), \text{ if} j \in J_1$$
$$v_j^* = \min(v_{ij}), \text{ if} j \in J_2$$
$$v_j^- = \min(v_{ij}), \text{ if} j \in J_1$$
$$v_j^- = \max(v_{ij}), \text{ if} j \in J_2$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[9]:
```
a_pos = np.zeros(n)
a_neg = np.zeros(n)
for j in range(n):
    column = raw_data[:,j]
    max_val = np.max(column)
    min_val = np.min(column)

    # See if we want to maximize benefit or minimize cost (for PIS)
    if j in benefit_attributes:
        a_pos[j] = max_val
        a_neg[j] = min_val
    else:
        a_pos[j] = min_val
        a_neg[j] = max_val

pd.DataFrame(data=[a_pos, a_neg], index=["$A^*$", "$A^-$"], columns=attributes)
```

[9]:
	SR	Econ	Avg	Wkts	Runs	Inns	TBB	\
A^*	0.029198	0.034447	0.021887	0.065281	0.015123	0.035747	0.028030	
A^-	0.068854	0.061598	0.051595	0.012554	0.040638	0.010514	0.007334	

	4w	Mat
A^*	0.023757	0.008613

A^-	0.000000	0.002533

1.5 Step 4 and 5 - Calculating Separation Measures and Similarities to PIS

The separation or distance between the alternatives can be measured by the n-dimensional Euclidean distance. The separation from the PIS A^* and NIS A^- are S^* and S^- respectively.

$$S_i^* = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^*\right)^2}$$

$$S_i^- = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^-\right)^2}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

We also calculate

$$C_i^* = \frac{S_i^-}{S_i^* + S_i^-}, \text{ where } i = 1, 2, \ldots, m$$

[10]:
```
sp = np.zeros(m)
sn = np.zeros(m)
cs = np.zeros(m)

for i in range(m):
    diff_pos = raw_data[i] - a_pos
    diff_neg = raw_data[i] - a_neg
    sp[i] = np.sqrt(diff_pos @ diff_pos)
    sn[i] = np.sqrt(diff_neg @ diff_neg)
    cs[i] = sn[i] / (sp[i] + sn[i])

pd.DataFrame(data=zip(sp, sn, cs), index=candidates, columns=["$S^*$", "$S^-$",␣
 ↪"$C^*$"])
```

[10]:

	S^*	S^-	C^*
Andre Russell	0.056819	0.040468	0.415961
Ben Stokes	0.075085	0.034796	0.316672
Chris Morris	0.053264	0.043989	0.452315
Dwayne Bravo	0.062330	0.033812	0.351688
Imran Tahir	0.026770	0.081947	0.753762
Jofra Archer	0.060683	0.039074	0.391692
Kagiso Rabada	0.024786	0.079581	0.762512
Keemo Paul	0.062163	0.036585	0.370485
Lasith Malinga	0.048952	0.050299	0.506789
Moeen Ali	0.073078	0.037874	0.341355
Mohammad Nabi	0.061346	0.042463	0.409050

```
Rashid Khan      0.047658  0.055154  0.536456
Sam Curran       0.063257  0.030364  0.324327
Sunil Narine     0.072042  0.030814  0.299582
Trent Boult      0.078508  0.031140  0.284000
```

1.6 Step 6 - Ranking the candidates/alternatives

We choose the candidate with the maximum C^* or rank all the alternatives in descending order according to their C^* values. This process can also be done for the S^* and S^- values.

[11]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage[::-1]
```

[12]:
```
cs_order = rank_according_to(cs)
sp_order = rank_according_to(sp)
sn_order = rank_according_to(sn)

pd.DataFrame(data=zip(cs_order, sp_order[::-1], sn_order), index=range(1, m +␣
↪1),
             columns=["$C^*$", "$S^*$", "$S^-$"])
```

[12]:

	C^*	S^*	S^-
1	Kagiso Rabada	Kagiso Rabada	Imran Tahir
2	Imran Tahir	Imran Tahir	Kagiso Rabada
3	Rashid Khan	Rashid Khan	Rashid Khan
4	Lasith Malinga	Lasith Malinga	Lasith Malinga
5	Chris Morris	Chris Morris	Chris Morris
6	Andre Russell	Andre Russell	Mohammad Nabi
7	Mohammad Nabi	Jofra Archer	Andre Russell
8	Jofra Archer	Mohammad Nabi	Jofra Archer
9	Keemo Paul	Keemo Paul	Moeen Ali
10	Dwayne Bravo	Dwayne Bravo	Keemo Paul
11	Moeen Ali	Sam Curran	Ben Stokes
12	Sam Curran	Sunil Narine	Dwayne Bravo
13	Ben Stokes	Moeen Ali	Trent Boult
14	Sunil Narine	Ben Stokes	Sunil Narine
15	Trent Boult	Trent Boult	Sam Curran

[13]:
```
print("The best candidate/alternative according to C* is " + cs_order[0])
print("The preferences in descending order are " + ", ".join(cs_order) + ".")
```

```
The best candidate/alternative according to C* is Kagiso Rabada
The preferences in descending order are Kagiso Rabada, Imran Tahir, Rashid Khan,
Lasith Malinga, Chris Morris, Andre Russell, Mohammad Nabi, Jofra Archer, Keemo
```

Paul, Dwayne Bravo, Moeen Ali, Sam Curran, Ben Stokes, Sunil Narine, Trent Boult.

NR TOPSIS Ranking

August 4, 2021

1 TOPSIS Ranking

[1]:
```
import numpy as np                # for linear algebra
import pandas as pd               # for tabular output
from scipy.stats import rankdata # for ranking the candidates
```

1.1 Step 0 - Obtaining and processing the data

The data from the Excel sheet is saved into CSV files and stored in the `data` folder at the root of the project. The criteria, their rankings, the players' scores based on the mentioned criteria are stored in Numpy arrays and processed for the next step.

Note that an attribute can be beneficial attribute (in which case, we will want to maximize it's contribution) or a cost attribute (which we will need to minimize). We call the set of beneficial attributes J_1 and that of cost attributes $J_2 = J_1^C$.

[2]:
```
bowlers_data = {
    'weights': '../data/bowling_criteria.csv',
    'scores': '../data/bowlers.csv',
}
batsmen_data = {
    'weights': '../data/batting_criteria.csv',
    'scores': '../data/batsmen.csv',
}
data = batsmen_data
```

[3]:
```
attributes_data = pd.read_csv(data['weights'])
attributes_data
```

[3]:

	Name	Ranking	Ideally
0	SR	1	Higher
1	Avg	2	Higher
2	Runs	3	Higher
3	Inn	4	Higher
4	NO	5	Higher
5	6s	6	Higher
6	4s	7	Higher
7	100s	8	Higher

```
8    50s        9  Higher
9    Mat       10  Higher
10    HS       11  Higher
11    BF       12  Higher
```

[4]:
```
benefit_attributes = set()
attributes = []
ranks = []
n = 0

for i, row in attributes_data.iterrows():
    attributes.append(row['Name'])
    ranks.append(float(row['Ranking']))
    n += 1

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

ranks = np.array(ranks)
```

[5]:
```
weights = 2 * (n + 1 - ranks) / (n * (n + 1))
pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[5]:
```
        Weight
SR    0.153846
Avg   0.141026
Runs  0.128205
Inn   0.115385
NO    0.102564
6s    0.089744
4s    0.076923
100s  0.064103
50s   0.051282
Mat   0.038462
HS    0.025641
BF    0.012821
```

[6]:
```
original_dataframe = pd.read_csv(data['scores'])
candidates = original_dataframe['Name'].to_numpy()
raw_data = pd.DataFrame(original_dataframe, columns=attributes).to_numpy()

dimensions = raw_data.shape
m = dimensions[0]
n = dimensions[1]

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

[6]:

	SR	Avg	Runs	Inn	NO	6s	4s	100s	50s	Mat	HS	BF
AB de Villiers	154.00	44.20	442.0	13.0	3.0	26.0	31.0	0.0	5.0	13.0	82.0	287.0
Andre Russel	204.81	56.67	510.0	13.0	4.0	52.0	31.0	0.0	4.0	14.0	80.0	249.0
Ben Stokes	124.24	20.50	123.0	9.0	3.0	4.0	8.0	0.0	0.0	9.0	46.0	99.0
Chris Gayle	153.60	40.83	490.0	13.0	1.0	34.0	45.0	0.0	4.0	13.0	99.0	319.0
Chris Lynn	139.65	31.15	405.0	13.0	0.0	22.0	41.0	0.0	4.0	13.0	82.0	290.0
David Warner	143.86	69.20	692.0	12.0	2.0	21.0	57.0	1.0	8.0	12.0	100.0	481.0
Faf Du Plessis	123.36	36.00	396.0	12.0	1.0	15.0	36.0	0.0	3.0	12.0	96.0	321.0
Jonny Bairstow	157.24	55.63	445.0	10.0	2.0	18.0	48.0	1.0	2.0	10.0	114.0	283.0
Jos Buttler	151.70	38.88	311.0	8.0	0.0	14.0	38.0	0.0	3.0	8.0	89.0	205.0
Kane Williamson	120.00	22.29	156.0	9.0	2.0	5.0	12.0	0.0	1.0	9.0	70.0	130.0
Kieron Pollard	156.74	34.88	279.0	14.0	6.0	22.0	14.0	0.0	1.0	16.0	83.0	178.0
Marcus Stoinis	135.25	52.75	211.0	10.0	6.0	10.0	14.0	0.0	0.0	10.0	46.0	156.0
Moeen Ali	165.41	27.50	220.0	10.0	2.0	17.0	16.0	0.0	2.0	11.0	66.0	133.0
Quinton de Kock	132.91	35.27	529.0	16.0	1.0	25.0	45.0	0.0	4.0	16.0	81.0	398.0
Shane Watson	127.56	23.41	398.0	17.0	0.0	20.0	42.0	0.0	3.0	17.0	96.0	312.0
Steve Smith	116.00	39.88	319.0	10.0	2.0	4.0	30.0	0.0	3.0	12.0	73.0	275.0

1.2 Step 1 - Normalizing the ratings

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[7]:
```
for j in range(n):
    column = raw_data[:,j]
    min_val = np.min(column)
    max_val = np.max(column)
    denom = max_val - min_val
```

```
    if j in benefit_attributes:
        raw_data[:,j] = (raw_data[:,j] - min_val) / denom
    else:
        raw_data[:,j] = (max_val - raw_data[:,j]) / denom

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

[7]:

	SR	Avg	Runs	Inn	NO	6s	4s	100s	50s	Mat	HS	BF
AB de Villiers	0.427880	0.486653	0.560633	0.555556	0.500000	0.458333	0.469388	0.0	0.625	0.555556	0.529412	0.492147
Andre Russel	1.000000	0.742710	0.680141	0.555556	0.666667	1.000000	0.469388	0.0	0.500	0.666667	0.500000	0.392670
Ben Stokes	0.092782	0.000000	0.000000	0.111111	0.500000	0.000000	0.000000	0.0	0.000	0.111111	0.000000	0.000000
Chris Gayle	0.423376	0.417454	0.644991	0.555556	0.166667	0.625000	0.755102	0.0	0.500	0.555556	0.779412	0.575916
Chris Lynn	0.266299	0.218686	0.495606	0.555556	0.000000	0.375000	0.673469	0.0	0.500	0.555556	0.529412	0.500000
David Warner	0.313703	1.000000	1.000000	0.444444	0.333333	0.354167	1.000000	1.0	1.000	0.444444	0.794118	1.000000
Faf Du Plessis	0.082874	0.318275	0.479789	0.444444	0.166667	0.229167	0.571429	0.0	0.375	0.444444	0.735294	0.581152
Jonny Bairstow	0.464362	0.721355	0.565905	0.222222	0.333333	0.291667	0.816327	1.0	0.250	0.222222	1.000000	0.481675
Jos Buttler	0.401982	0.377413	0.330404	0.000000	0.000000	0.208333	0.612245	0.0	0.375	0.000000	0.632353	0.277487
Kane Williamson	0.045040	0.036756	0.057996	0.111111	0.333333	0.020833	0.081633	0.0	0.125	0.111111	0.352941	0.081152
Kieron Pollard	0.458732	0.295277	0.274165	0.666667	1.000000	0.375000	0.122449	0.0	0.125	0.888889	0.544118	0.206806
Marcus Stoinis	0.216755	0.662218	0.154657	0.222222	1.000000	0.125000	0.122449	0.0	0.000	0.222222	0.000000	0.149215
Moeen Ali	0.556356	0.143737	0.170475	0.222222	0.333333	0.270833	0.163265	0.0	0.250	0.333333	0.294118	0.089005
Quinton de Kock	0.190406	0.303285	0.713533	0.888889	0.166667	0.437500	0.755102	0.0	0.500	0.888889	0.514706	0.782723
Shane Watson	0.130166	0.059754	0.483304	1.000000	0.000000	0.333333	0.693878	0.0	0.375	1.000000	0.735294	0.557592
Steve Smith	0.000000	0.397947	0.344464	0.222222	0.333333	0.000000	0.448980	0.0	0.375	0.444444	0.397059	0.460733

1.3 Step 2 - Calculating the Weighted Normalized Ratings

$$v_{ij} = w_j r_{ij}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[8]:
```
raw_data *= weights
pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[8]:

	SR	Avg	Runs	Inn	NO	6s
AB de Villiers	0.065828	0.068631	0.071876	0.064103	0.051282	0.041132
Andre Russel	0.153846	0.104741	0.087198	0.064103	0.068376	0.089744
Ben Stokes	0.014274	0.000000	0.000000	0.012821	0.051282	0.000000
Chris Gayle	0.065135	0.058872	0.082691	0.064103	0.017094	0.056090
Chris Lynn	0.040969	0.030840	0.063539	0.064103	0.000000	0.033654
David Warner	0.048262	0.141026	0.128205	0.051282	0.034188	0.031784
Faf Du Plessis	0.012750	0.044885	0.061511	0.051282	0.017094	0.020566
Jonny Bairstow	0.071440	0.101730	0.072552	0.025641	0.034188	0.026175
Jos Buttler	0.061843	0.053225	0.042360	0.000000	0.000000	0.018697
Kane Williamson	0.006929	0.005183	0.007435	0.012821	0.034188	0.001870
Kieron Pollard	0.070574	0.041642	0.035149	0.076923	0.102564	0.033654
Marcus Stoinis	0.033347	0.093390	0.019828	0.025641	0.102564	0.011218
Moeen Ali	0.085593	0.020271	0.021856	0.025641	0.034188	0.024306
Quinton de Kock	0.029293	0.042771	0.091479	0.102564	0.017094	0.039263
Shane Watson	0.020025	0.008427	0.061962	0.115385	0.000000	0.029915
Steve Smith	0.000000	0.056121	0.044162	0.025641	0.034188	0.000000

	4s	100s	50s	Mat	HS	BF
AB de Villiers	0.036107	0.000000	0.032051	0.021368	0.013575	0.006310
Andre Russel	0.036107	0.000000	0.025641	0.025641	0.012821	0.005034
Ben Stokes	0.000000	0.000000	0.000000	0.004274	0.000000	0.000000
Chris Gayle	0.058085	0.000000	0.025641	0.021368	0.019985	0.007384
Chris Lynn	0.051805	0.000000	0.025641	0.021368	0.013575	0.006410
David Warner	0.076923	0.064103	0.051282	0.017094	0.020362	0.012821
Faf Du Plessis	0.043956	0.000000	0.019231	0.017094	0.018854	0.007451
Jonny Bairstow	0.062794	0.064103	0.012821	0.008547	0.025641	0.006175
Jos Buttler	0.047096	0.000000	0.019231	0.000000	0.016214	0.003558
Kane Williamson	0.006279	0.000000	0.006410	0.004274	0.009050	0.001040
Kieron Pollard	0.009419	0.000000	0.006410	0.034188	0.013952	0.002651
Marcus Stoinis	0.009419	0.000000	0.000000	0.008547	0.000000	0.001913
Moeen Ali	0.012559	0.000000	0.012821	0.012821	0.007541	0.001141
Quinton de Kock	0.058085	0.000000	0.025641	0.034188	0.013198	0.010035
Shane Watson	0.053375	0.000000	0.019231	0.038462	0.018854	0.007149
Steve Smith	0.034537	0.000000	0.019231	0.017094	0.010181	0.005907

1.4 Step 3 - Identifying PIS (A^*) and NIS (A^-)

$$A^* = \{w_1, w_2, \ldots, w_n\}$$

$$A^- = \{0, 0, \ldots, 0\}$$

[9]:
```
a_pos = np.copy(weights)
a_neg = np.zeros(n)
```

```
pd.DataFrame(data=[a_pos, a_neg], index=["$A^*$", "$A^-$"], columns=attributes)
```

[9]:

	SR	Avg	Runs	Inn	NO	6s	4s	\
A^*	0.153846	0.141026	0.128205	0.115385	0.102564	0.089744	0.076923	
A^-	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

	100s	50s	Mat	HS	BF
A^*	0.064103	0.051282	0.038462	0.025641	0.012821
A^-	0.000000	0.000000	0.000000	0.000000	0.000000

1.5 Step 4 and 5 - Calculating Separation Measures and Similarities to PIS

The separation or distance between the alternatives can be measured by the n-dimensional Euclidean distance. The separation from the PIS A^* and NIS A^- are S^* and S^- respectively.

$$S_i^* = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^*\right)^2}$$

$$S_i^- = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^-\right)^2}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

We also calculate

$$C_i^* = \frac{S_i^-}{S_i^* + S_i^-}, \text{ where } i = 1, 2, \ldots, m$$

[10]:
```
sp = np.zeros(m)
sn = np.zeros(m)
cs = np.zeros(m)

for i in range(m):
    diff_pos = raw_data[i] - a_pos
    diff_neg = raw_data[i] - a_neg
    sp[i] = np.sqrt(diff_pos @ diff_pos)
    sn[i] = np.sqrt(diff_neg @ diff_neg)
    cs[i] = sn[i] / (sp[i] + sn[i])

pd.DataFrame(data=zip(sp, sn, cs), index=candidates, columns=["$S^*$", "$S^-$",␣
↪"$C^*$"])
```

[10]:

	S^*	S^-	C^*
AB de Villiers	0.174382	0.160163	0.478748
Andre Russel	0.116700	0.248776	0.680691
Ben Stokes	0.302746	0.054920	0.153552

```
Chris Gayle       0.182300  0.164425  0.474222
Chris Lynn        0.226352  0.126221  0.357999
David Warner      0.154195  0.238726  0.607567
Faf Du Plessis    0.237810  0.110955  0.318137
Jonny Bairstow    0.175557  0.179480  0.505525
Jos Buttler       0.245161  0.108004  0.305817
Kane Williamson   0.300392  0.040603  0.119072
Kieron Pollard    0.202568  0.164377  0.447960
Marcus Stoinis    0.236055  0.147292  0.384226
Moeen Ali         0.241691  0.105719  0.304306
Quinton de Kock   0.204539  0.169953  0.453822
Shane Watson      0.244431  0.153702  0.386057
Steve Smith       0.257682  0.094443  0.268208
```

1.6 Step 6 - Ranking the candidates/alternatives

We choose the candidate with the maximum C^* or rank all the alternatives in descending order according to their C^* values. This process can also be done for the S^* and S^- values.

[11]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage[::-1]
```

[12]:
```
cs_order = rank_according_to(cs)
sp_order = rank_according_to(sp)
sn_order = rank_according_to(sn)

pd.DataFrame(data=zip(cs_order, sp_order[::-1], sn_order), index=range(1, m +␣
↪1),
             columns=["$C^*$", "$S^*$", "$S^-$"])
```

[12]:

	C^*	S^*	S^-
1	Andre Russel	Andre Russel	Andre Russel
2	David Warner	David Warner	David Warner
3	Jonny Bairstow	AB de Villiers	Jonny Bairstow
4	AB de Villiers	Jonny Bairstow	Quinton de Kock
5	Chris Gayle	Chris Gayle	Chris Gayle
6	Quinton de Kock	Kieron Pollard	Kieron Pollard
7	Kieron Pollard	Quinton de Kock	AB de Villiers
8	Shane Watson	Chris Lynn	Shane Watson
9	Marcus Stoinis	Marcus Stoinis	Marcus Stoinis
10	Chris Lynn	Faf Du Plessis	Chris Lynn
11	Faf Du Plessis	Moeen Ali	Faf Du Plessis
12	Jos Buttler	Shane Watson	Jos Buttler
13	Moeen Ali	Jos Buttler	Moeen Ali

```
14      Steve Smith      Steve Smith      Steve Smith
15       Ben Stokes  Kane Williamson       Ben Stokes
16  Kane Williamson       Ben Stokes  Kane Williamson
```

[13]:
```
print("The best candidate/alternative according to C* is " + cs_order[0])
print("The preferences in descending order are " + ", ".join(cs_order) + ".")
```

```
The best candidate/alternative according to C* is Andre Russel
The preferences in descending order are Andre Russel, David Warner, Jonny
Bairstow, AB de Villiers, Chris Gayle, Quinton de Kock, Kieron Pollard, Shane
Watson, Marcus Stoinis, Chris Lynn, Faf Du Plessis, Jos Buttler, Moeen Ali,
Steve Smith, Ben Stokes, Kane Williamson.
```

NR TOPSIS Ranking

August 9, 2021

1 TOPSIS Ranking

[1]:
```
import numpy as np                # for linear algebra
import pandas as pd               # for tabular output
from scipy.stats import rankdata # for ranking the candidates
```

1.1 Step 0 - Obtaining and processing the data

The data from the Excel sheet is saved into CSV files and stored in the `data` folder at the root of the project. The criteria, their rankings, the players' scores based on the mentioned criteria are stored in Numpy arrays and processed for the next step.

Note that an attribute can be beneficial attribute (in which case, we will want to maximize it's contribution) or a cost attribute (which we will need to minimize). We call the set of beneficial attributes J_1 and that of cost attributes $J_2 = J_1^C$.

[2]:
```
bowlers_data = {
    'weights': '../data/bowling_criteria.csv',
    'scores': '../data/bowlers.csv',
}
batsmen_data = {
    'weights': '../data/batting_criteria.csv',
    'scores': '../data/batsmen.csv',
}
data = bowlers_data
```

[3]:
```
attributes_data = pd.read_csv(data['weights'])
attributes_data
```

[3]:

	Name	Ranking	Ideally
0	SR	1	Lower
1	Econ	2	Lower
2	Avg	3	Lower
3	Wkts	4	Higher
4	Runs	5	Lower
5	Inns	6	Higher
6	TBB	7	Higher
7	4w	8	Higher

```
8   Mat        9  Higher
```

[4]:
```
benefit_attributes = set()
attributes = []
ranks = []
n = 0

for i, row in attributes_data.iterrows():
    attributes.append(row['Name'])
    ranks.append(float(row['Ranking']))
    n += 1

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

ranks = np.array(ranks)
```

[5]:
```
weights = 2 * (n + 1 - ranks) / (n * (n + 1))
pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[5]:

	Weight
SR	0.200000
Econ	0.177778
Avg	0.155556
Wkts	0.133333
Runs	0.111111
Inns	0.088889
TBB	0.066667
4w	0.044444
Mat	0.022222

[6]:
```
original_dataframe = pd.read_csv(data['scores'])
candidates = original_dataframe['Name'].to_numpy()
raw_data = pd.DataFrame(original_dataframe, columns=attributes).to_numpy()

dimensions = raw_data.shape
m = dimensions[0]
n = dimensions[1]

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

[6]:

	SR	Econ	Avg	Wkts	Runs	Inns	TBB	4w	Mat
Andre Russell	16.45	9.51	26.09	11.0	287.0	12.0	181.0	0.0	14.0
Ben Stokes	16.83	11.23	31.50	6.0	189.0	6.0	101.0	0.0	9.0
Chris Morris	15.23	9.27	23.54	13.0	306.0	9.0	198.0	0.0	9.0
Dwayne Bravo	22.45	8.02	30.00	11.0	330.0	12.0	247.0	0.0	12.0
Imran Tahir	14.85	6.70	16.58	26.0	431.0	17.0	386.0	2.0	17.0

Jofra Archer	23.45	6.77	26.45	11.0	291.0	11.0	258.0	0.0	11.0
Kagiso Rabada	11.28	7.83	14.72	25.0	368.0	12.0	282.0	2.0	12.0
Keemo Paul	18.11	8.72	26.33	9.0	237.0	8.0	163.0	0.0	8.0
Lasith Malinga	16.81	9.77	27.38	16.0	438.0	12.0	269.0	2.0	12.0
Moeen Ali	25.00	6.76	28.17	6.0	169.0	9.0	150.0	0.0	11.0
Mohammad Nabi	21.88	6.65	24.25	8.0	194.0	8.0	175.0	1.0	8.0
Rashid Khan	21.18	6.28	22.18	17.0	377.0	15.0	360.0	0.0	15.0
Sam Curran	19.80	9.79	32.30	10.0	323.0	9.0	198.0	1.0	9.0
Sunil Narine	26.60	7.83	34.70	10.0	347.0	12.0	266.0	0.0	12.0
Trent Boult	22.80	8.58	32.60	5.0	163.0	5.0	114.0	0.0	5.0

1.2 Step 1 - Normalizing the ratings

$$r_{ij} = \begin{cases} \frac{x_{ij} - min_j}{max_j - min_j} & \text{if } j \in J_1 \\ \frac{max_j - x_{ij}}{max_j - min_j} & \text{if } j \in J_2 \end{cases}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[7]:
```
for j in range(n):
    column = raw_data[:,j]
    min_val = np.min(column)
    max_val = np.max(column)
    denom = max_val - min_val
    if j in benefit_attributes:
        raw_data[:,j] = (raw_data[:,j] - min_val) / denom
    else:
        raw_data[:,j] = (max_val - raw_data[:,j]) / denom

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

[7]:

	SR	Econ	Avg	Wkts	Runs	Inns
Andre Russell	0.662533	0.347475	0.430931	0.285714	0.549091	0.583333
Ben Stokes	0.637728	0.000000	0.160160	0.047619	0.905455	0.083333
Chris Morris	0.742167	0.395960	0.558559	0.380952	0.480000	0.333333
Dwayne Bravo	0.270888	0.648485	0.235235	0.285714	0.392727	0.583333
Imran Tahir	0.766971	0.915152	0.906907	1.000000	0.025455	1.000000
Jofra Archer	0.205614	0.901010	0.412913	0.285714	0.534545	0.500000
Kagiso Rabada	1.000000	0.686869	1.000000	0.952381	0.254545	0.583333
Keemo Paul	0.554178	0.507071	0.418919	0.190476	0.730909	0.250000
Lasith Malinga	0.639034	0.294949	0.366366	0.523810	0.000000	0.583333
Moeen Ali	0.104439	0.903030	0.326827	0.047619	0.978182	0.333333
Mohammad Nabi	0.308094	0.925253	0.523023	0.142857	0.887273	0.250000
Rashid Khan	0.353786	1.000000	0.626627	0.571429	0.221818	0.833333
Sam Curran	0.443864	0.290909	0.120120	0.238095	0.418182	0.333333
Sunil Narine	0.000000	0.686869	0.000000	0.238095	0.330909	0.583333
Trent Boult	0.248042	0.535354	0.105105	0.000000	1.000000	0.000000

	TBB	4w	Mat
Andre Russell	0.280702	0.0	0.750000
Ben Stokes	0.000000	0.0	0.333333
Chris Morris	0.340351	0.0	0.333333
Dwayne Bravo	0.512281	0.0	0.583333
Imran Tahir	1.000000	1.0	1.000000
Jofra Archer	0.550877	0.0	0.500000
Kagiso Rabada	0.635088	1.0	0.583333
Keemo Paul	0.217544	0.0	0.250000
Lasith Malinga	0.589474	1.0	0.583333
Moeen Ali	0.171930	0.0	0.500000
Mohammad Nabi	0.259649	0.5	0.250000
Rashid Khan	0.908772	0.0	0.833333
Sam Curran	0.340351	0.5	0.333333
Sunil Narine	0.578947	0.0	0.583333
Trent Boult	0.045614	0.0	0.000000

1.3 Step 2 - Calculating the Weighted Normalized Ratings

$$v_{ij} = w_j r_{ij}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[8]:
```
raw_data *= weights
pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[8]:

	SR	Econ	Avg	Wkts	Runs	Inns	\
Andre Russell	0.132507	0.061773	0.067034	0.038095	0.061010	0.051852	
Ben Stokes	0.127546	0.000000	0.024914	0.006349	0.100606	0.007407	
Chris Morris	0.148433	0.070393	0.086887	0.050794	0.053333	0.029630	
Dwayne Bravo	0.054178	0.115286	0.036592	0.038095	0.043636	0.051852	
Imran Tahir	0.153394	0.162694	0.141074	0.133333	0.002828	0.088889	
Jofra Archer	0.041123	0.160180	0.064231	0.038095	0.059394	0.044444	
Kagiso Rabada	0.200000	0.122110	0.155556	0.126984	0.028283	0.051852	
Keemo Paul	0.110836	0.090146	0.065165	0.025397	0.081212	0.022222	
Lasith Malinga	0.127807	0.052435	0.056990	0.069841	0.000000	0.051852	
Moeen Ali	0.020888	0.160539	0.050840	0.006349	0.108687	0.029630	
Mohammad Nabi	0.061619	0.164489	0.081359	0.019048	0.098586	0.022222	
Rashid Khan	0.070757	0.177778	0.097475	0.076190	0.024646	0.074074	
Sam Curran	0.088773	0.051717	0.018685	0.031746	0.046465	0.029630	
Sunil Narine	0.000000	0.122110	0.000000	0.031746	0.036768	0.051852	
Trent Boult	0.049608	0.095174	0.016350	0.000000	0.111111	0.000000	

	TBB	4w	Mat
Andre Russell	0.018713	0.000000	0.016667
Ben Stokes	0.000000	0.000000	0.007407
Chris Morris	0.022690	0.000000	0.007407
Dwayne Bravo	0.034152	0.000000	0.012963

```
Imran Tahir      0.066667  0.044444  0.022222
Jofra Archer     0.036725  0.000000  0.011111
Kagiso Rabada    0.042339  0.044444  0.012963
Keemo Paul       0.014503  0.000000  0.005556
Lasith Malinga   0.039298  0.044444  0.012963
Moeen Ali        0.011462  0.000000  0.011111
Mohammad Nabi    0.017310  0.022222  0.005556
Rashid Khan      0.060585  0.000000  0.018519
Sam Curran       0.022690  0.022222  0.007407
Sunil Narine     0.038596  0.000000  0.012963
Trent Boult      0.003041  0.000000  0.000000
```

1.4 Step 3 - Identifying PIS (A^*) and NIS (A^-)

$$A^* = \{w_1, w_2, \ldots, w_n\}$$
$$A^- = \{0, 0, \ldots, 0\}$$

[9]:
```
a_pos = np.copy(weights)
a_neg = np.zeros(n)

pd.DataFrame(data=[a_pos, a_neg], index=["$A^*$", "$A^-$"], columns=attributes)
```

[9]:
```
          SR      Econ       Avg      Wkts      Runs      Inns       TBB  \
$A^*$  0.2  0.177778  0.155556  0.133333  0.111111  0.088889  0.066667
$A^-$  0.0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

                4w       Mat
$A^*$  0.044444  0.022222
$A^-$  0.000000  0.000000
```

1.5 Step 4 and 5 - Calculating Separation Measures and Similarities to PIS

The separation or distance between the alternatives can be measured by the n-dimensional Euclidean distance. The separation from the PIS A^* and NIS A^- are S^* and S^- respectively.

$$S_i^* = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^*\right)^2}$$
$$S_i^- = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^-\right)^2}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

We also calculate

$$C_i^* = \frac{S_i^-}{S_i^* + S_i^-}, \text{ where } i = 1, 2, \ldots, m$$

[10]:
```
sp = np.zeros(m)
sn = np.zeros(m)
cs = np.zeros(m)

for i in range(m):
    diff_pos = raw_data[i] - a_pos
    diff_neg = raw_data[i] - a_neg
    sp[i] = np.sqrt(diff_pos @ diff_pos)
    sn[i] = np.sqrt(diff_neg @ diff_neg)
    cs[i] = sn[i] / (sp[i] + sn[i])

pd.DataFrame(data=zip(sp, sn, cs), index=candidates, columns=["$S^*$", "$S^-$",␣
↪"$C^*$"])
```

[10]:

	S^*	S^-	C^*
Andre Russell	0.207621	0.185358	0.471674
Ben Stokes	0.288852	0.164804	0.363279
Chris Morris	0.191566	0.203492	0.515094
Dwayne Bravo	0.239656	0.157935	0.397230
Imran Tahir	0.119727	0.320159	0.727823
Jofra Archer	0.224961	0.199751	0.470322
Kagiso Rabada	0.109768	0.320403	0.744827
Keemo Paul	0.213923	0.180645	0.457831
Lasith Malinga	0.221858	0.183265	0.452369
Moeen Ali	0.261022	0.204400	0.439172
Mohammad Nabi	0.213779	0.221073	0.508387
Rashid Khan	0.181835	0.249050	0.577996
Sam Curran	0.260104	0.126541	0.327279
Sunil Narine	0.295574	0.147027	0.332188
Trent Boult	0.284633	0.155375	0.353119

1.6 Step 6 - Ranking the candidates/alternatives

We choose the candidate with the maximum C^* or rank all the alternatives in descending order according to their C^* values. This process can also be done for the S^* and S^- values.

[11]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage[::-1]
```

[12]:
```
cs_order = rank_according_to(cs)
sp_order = rank_according_to(sp)
sn_order = rank_according_to(sn)

pd.DataFrame(data=zip(cs_order, sp_order[::-1], sn_order), index=range(1, m +␣
↪1),
```

```
                columns=["$C^*$", "$S^*$", "$S^-$"])
```

[12]:

	C^*	S^*	S^-
1	Kagiso Rabada	Kagiso Rabada	Kagiso Rabada
2	Imran Tahir	Imran Tahir	Imran Tahir
3	Rashid Khan	Rashid Khan	Rashid Khan
4	Chris Morris	Chris Morris	Mohammad Nabi
5	Mohammad Nabi	Andre Russell	Moeen Ali
6	Andre Russell	Mohammad Nabi	Chris Morris
7	Jofra Archer	Keemo Paul	Jofra Archer
8	Keemo Paul	Lasith Malinga	Andre Russell
9	Lasith Malinga	Jofra Archer	Lasith Malinga
10	Moeen Ali	Dwayne Bravo	Keemo Paul
11	Dwayne Bravo	Sam Curran	Ben Stokes
12	Ben Stokes	Moeen Ali	Dwayne Bravo
13	Trent Boult	Trent Boult	Trent Boult
14	Sunil Narine	Ben Stokes	Sunil Narine
15	Sam Curran	Sunil Narine	Sam Curran

[13]:
```
print("The best candidate/alternative according to C* is " + cs_order[0])
print("The preferences in descending order are " + ", ".join(cs_order) + ".")
```

```
The best candidate/alternative according to C* is Kagiso Rabada
The preferences in descending order are Kagiso Rabada, Imran Tahir, Rashid Khan,
Chris Morris, Mohammad Nabi, Andre Russell, Jofra Archer, Keemo Paul, Lasith
Malinga, Moeen Ali, Dwayne Bravo, Ben Stokes, Trent Boult, Sunil Narine, Sam
Curran.
```

TODIM Ranking

May 28, 2021

1 TODIM Ranking

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

1.1 Step 0 - Obtaining and preprocessing data

[2]:
```
bowlers_data = {
    'weights': '../data/bowling_criteria.csv',
    'scores': '../data/bowlers.csv',
}
batsmen_data = {
    'weights': '../data/batting_criteria.csv',
    'scores': '../data/batsmen.csv',
}
data = batsmen_data
```

[3]:
```
attributes_data = pd.read_csv(data['weights'])
attributes_data
```

[3]:

	Name	Ranking	Ideally
0	SR	1	Higher
1	Avg	2	Higher
2	Runs	3	Higher
3	Inn	4	Higher
4	NO	5	Higher
5	6s	6	Higher
6	4s	7	Higher
7	100s	8	Higher
8	50s	9	Higher
9	Mat	10	Higher
10	HS	11	Higher
11	BF	12	Higher

[4]:
```
benefit_attributes = set()
attributes = []
ranks = []
n = 0

for i, row in attributes_data.iterrows():
    attributes.append(row['Name'])
    ranks.append(float(row['Ranking']))
    n += 1

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

ranks = np.array(ranks)
```

[5]:
```
weights = 2 * (n + 1 - ranks) / (n * (n + 1))
pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[5]:

	Weight
SR	0.153846
Avg	0.141026
Runs	0.128205
Inn	0.115385
NO	0.102564
6s	0.089744
4s	0.076923
100s	0.064103
50s	0.051282
Mat	0.038462
HS	0.025641
BF	0.012821

[6]:
```
original_dataframe = pd.read_csv(data['scores'])
candidates = original_dataframe['Name'].to_numpy()
raw_data = pd.DataFrame(original_dataframe, columns=attributes).to_numpy()

dimensions = raw_data.shape
m = dimensions[0]
n = dimensions[1]

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

[6]:

	SR	Avg	Runs	Inn	NO	6s	4s	100s	50s	Mat
AB de Villiers	154.00	44.20	442.0	13.0	3.0	26.0	31.0	0.0	5.0	13.0
Andre Russel	204.81	56.67	510.0	13.0	4.0	52.0	31.0	0.0	4.0	14.0
Ben Stokes	124.24	20.50	123.0	9.0	3.0	4.0	8.0	0.0	0.0	9.0
Chris Gayle	153.60	40.83	490.0	13.0	1.0	34.0	45.0	0.0	4.0	13.0

Chris Lynn	139.65	31.15	405.0	13.0	0.0	22.0	41.0	0.0	4.0	13.0
David Warner	143.86	69.20	692.0	12.0	2.0	21.0	57.0	1.0	8.0	12.0
Faf Du Plessis	123.36	36.00	396.0	12.0	1.0	15.0	36.0	0.0	3.0	12.0
Jonny Bairstow	157.24	55.63	445.0	10.0	2.0	18.0	48.0	1.0	2.0	10.0
Jos Buttler	151.70	38.88	311.0	8.0	0.0	14.0	38.0	0.0	3.0	8.0
Kane Williamson	120.00	22.29	156.0	9.0	2.0	5.0	12.0	0.0	1.0	9.0
Kieron Pollard	156.74	34.88	279.0	14.0	6.0	22.0	14.0	0.0	1.0	16.0
Marcus Stoinis	135.25	52.75	211.0	10.0	6.0	10.0	14.0	0.0	0.0	10.0
Moeen Ali	165.41	27.50	220.0	10.0	2.0	17.0	16.0	0.0	2.0	11.0
Quinton de Kock	132.91	35.27	529.0	16.0	1.0	25.0	45.0	0.0	4.0	16.0
Shane Watson	127.56	23.41	398.0	17.0	0.0	20.0	42.0	0.0	3.0	17.0
Steve Smith	116.00	39.88	319.0	10.0	2.0	4.0	30.0	0.0	3.0	12.0

	HS	BF
AB de Villiers	82.0	287.0
Andre Russel	80.0	249.0
Ben Stokes	46.0	99.0
Chris Gayle	99.0	319.0
Chris Lynn	82.0	290.0
David Warner	100.0	481.0
Faf Du Plessis	96.0	321.0
Jonny Bairstow	114.0	283.0
Jos Buttler	89.0	205.0
Kane Williamson	70.0	130.0
Kieron Pollard	83.0	178.0
Marcus Stoinis	46.0	156.0
Moeen Ali	66.0	133.0
Quinton de Kock	81.0	398.0
Shane Watson	96.0	312.0
Steve Smith	73.0	275.0

1.2 Step 1 - Normalizing the Ratings And Weights

$$P_{ij} = \begin{cases} \frac{x_{ij}}{\sum_{k=1}^{m} x_{kj}} & \text{if } j \in J_1 \\ \frac{\frac{1}{x_{ij}}}{\sum_{k=1}^{m} \frac{1}{x_{kj}}} & \text{if } j \in J_2 \end{cases}$$

$$w_{rc} = \frac{w_c}{w_r}$$

and $w_r = \max\{w_c | c = 1, 2, \ldots, n\}$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[7]:
```
for j in range(n):
    column = raw_data[:,j]
    if j in benefit_attributes:
        raw_data[:,j] /= sum(column)
```

```
    else:
        column = 1 / column
        raw_data[:,j] = column / sum(column)

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

[7]:

	SR	Avg	Runs	Inn	NO	6s
AB de Villiers	0.066773	0.070266	0.074587	0.068783	0.085714	0.084142
Andre Russel	0.088803	0.090090	0.086061	0.068783	0.114286	0.168285
Ben Stokes	0.053869	0.032589	0.020756	0.047619	0.085714	0.012945
Chris Gayle	0.066599	0.064908	0.082686	0.068783	0.028571	0.110032
Chris Lynn	0.060551	0.049520	0.068343	0.068783	0.000000	0.071197
David Warner	0.062376	0.110009	0.116774	0.063492	0.057143	0.067961
Faf Du Plessis	0.053488	0.057230	0.066824	0.063492	0.028571	0.048544
Jonny Bairstow	0.068178	0.088436	0.075093	0.052910	0.057143	0.058252
Jos Buttler	0.065775	0.061808	0.052481	0.042328	0.000000	0.045307
Kane Williamson	0.052031	0.035435	0.026325	0.047619	0.057143	0.016181
Kieron Pollard	0.067961	0.055450	0.047081	0.074074	0.171429	0.071197
Marcus Stoinis	0.058643	0.083858	0.035606	0.052910	0.171429	0.032362
Moeen Ali	0.071720	0.043717	0.037125	0.052910	0.057143	0.055016
Quinton de Kock	0.057628	0.056070	0.089268	0.084656	0.028571	0.080906
Shane Watson	0.055309	0.037215	0.067162	0.089947	0.000000	0.064725
Steve Smith	0.050296	0.063398	0.053831	0.052910	0.057143	0.012945

	4s	100s	50s	Mat	HS	BF
AB de Villiers	0.061024	0.0	0.106383	0.066667	0.062932	0.069728
Andre Russel	0.061024	0.0	0.085106	0.071795	0.061397	0.060496
Ben Stokes	0.015748	0.0	0.000000	0.046154	0.035303	0.024052
Chris Gayle	0.088583	0.0	0.085106	0.066667	0.075979	0.077502
Chris Lynn	0.080709	0.0	0.085106	0.066667	0.062932	0.070457
David Warner	0.112205	0.5	0.170213	0.061538	0.076746	0.116861
Faf Du Plessis	0.070866	0.0	0.063830	0.061538	0.073676	0.077988
Jonny Bairstow	0.094488	0.5	0.042553	0.051282	0.087490	0.068756
Jos Buttler	0.074803	0.0	0.063830	0.041026	0.068304	0.049806
Kane Williamson	0.023622	0.0	0.021277	0.046154	0.053722	0.031584
Kieron Pollard	0.027559	0.0	0.021277	0.082051	0.063699	0.043246
Marcus Stoinis	0.027559	0.0	0.000000	0.051282	0.035303	0.037901
Moeen Ali	0.031496	0.0	0.042553	0.056410	0.050652	0.032313
Quinton de Kock	0.088583	0.0	0.085106	0.082051	0.062164	0.096696
Shane Watson	0.082677	0.0	0.063830	0.087179	0.073676	0.075802
Steve Smith	0.059055	0.0	0.063830	0.061538	0.056025	0.066812

[8]:
```
max_weight = max(weights)
weights /= max_weight

pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[8]:

	Weight
SR	1.000000
Avg	0.916667
Runs	0.833333
Inn	0.750000
NO	0.666667
6s	0.583333
4s	0.500000
100s	0.416667
50s	0.333333
Mat	0.250000
HS	0.166667
BF	0.083333

1.3 Step 2 - Calculating Dominance Degrees

For the contribution of each criteria, we have:

$$\Phi_c\left(A_i, A_j\right) = \begin{cases} \sqrt{\frac{(P_{ic}-P_{jc})w_{rc}}{\sum_{c=1}^{n} w_{rc}}} & \text{if } P_{ic} - P_{jc} > 0 \\ 0 & \text{if } P_{ic} - P_{jc} = 0 \\ -\frac{1}{\theta}\sqrt{\frac{\left(\sum_{c=1}^{n} w_{rc}\right)(P_{jc}-P_{ic})}{w_{rc}}} & \text{if } P_{ic} - P_{jc} < 0 \end{cases}$$

Combining all contributions, we get the dominance degrees:

$$\delta\left(A_i, A_j\right) = \sum_{c=1}^{n} \Phi_c\left(A_i, A_j\right)$$

Here $c = 1, 2, \ldots, n$, $i, j = 1, 2, \ldots, m$.

[9]:
```
# The loss attenuation factor
theta = 1.0
```

[10]:
```
phi = np.zeros((n, m, m))

weight_sum = sum(weights)

for c in range(n):
    for i in range(m):
        for j in range(m):
            pic = raw_data[i,c]
            pjc = raw_data[j,c]
            val = 0
            if pic > pjc:
                val = math.sqrt((pic - pjc) * weights[c] / weight_sum)
            if pic < pjc:
```

```
                val = -1.0 / theta * math.sqrt(weight_sum * (pjc - pic) /␣
↪weights[c])
            phi[c, i, j] = val
```

[11]:
```
delta = np.zeros((m, m))
for i in range(m):
    for j in range(m):
        delta[i,j] = sum(phi[:,i,j])

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

[11]:

	AB de Villiers	Andre Russel	Ben Stokes	Chris Gayle
AB de Villiers	0.000000	-2.863567	0.541653	-2.736826
Andre Russel	-1.432863	0.000000	0.680447	-2.185471
Ben Stokes	-8.636770	-9.667204	0.000000	-9.146171
Chris Gayle	-1.464281	-2.969557	-0.175313	0.000000
Chris Lynn	-2.701454	-3.741018	-0.417572	-3.823306
David Warner	-1.209961	-2.478412	0.286893	-0.915043
Faf Du Plessis	-3.656001	-4.732884	-0.356786	-3.705243
Jonny Bairstow	-3.131000	-4.366829	0.157937	-3.398367
Jos Buttler	-5.721169	-6.457348	-1.096931	-6.463471
Kane Williamson	-8.286399	-8.948843	-0.484118	-8.413792
Kieron Pollard	-4.391405	-5.261568	0.525750	-5.592233
Marcus Stoinis	-7.120193	-7.819790	0.373489	-7.790279
Moeen Ali	-7.067200	-8.058174	-0.172304	-7.215517
Quinton de Kock	-2.139380	-2.687265	-0.192804	-1.683666
Shane Watson	-3.144248	-4.090498	-0.432750	-3.807919
Steve Smith	-5.171623	-5.638931	-0.338728	-5.684441

	Chris Lynn	David Warner	Faf Du Plessis	Jonny Bairstow
AB de Villiers	-0.470107	-8.323082	-1.469548	-4.718239
Andre Russel	-1.227112	-8.352768	-1.751344	-4.861276
Ben Stokes	-8.182355	-13.270051	-7.802967	-10.830438
Chris Gayle	0.285547	-8.043642	0.118903	-4.553716
Chris Lynn	0.000000	-9.426578	-1.984101	-5.766008
David Warner	-0.167186	0.000000	0.639631	-0.486275
Faf Du Plessis	-2.293111	-9.522781	0.000000	-5.872369
Jonny Bairstow	-2.207302	-6.057709	-1.841390	0.000000
Jos Buttler	-4.292833	-11.468555	-4.064842	-8.271405
Kane Williamson	-7.306546	-12.686198	-7.081497	-10.064896
Kieron Pollard	-3.562238	-9.797233	-4.159653	-7.506049
Marcus Stoinis	-6.826716	-12.139600	-6.384429	-9.027501
Moeen Ali	-6.017898	-11.455425	-5.522270	-7.842208
Quinton de Kock	-0.034919	-8.317030	-0.465049	-5.084213
Shane Watson	-1.373950	-9.392831	-1.139287	-5.636553
Steve Smith	-4.242396	-10.580745	-3.466620	-6.795844

	Jos Buttler	Kane Williamson	Kieron Pollard	Marcus Stoinis	\
AB de Villiers	-0.478668	0.563287	-1.747245	-0.811863	
Andre Russel	-0.406260	0.668864	-1.363234	-0.227213	
Ben Stokes	-6.242437	-3.047484	-7.635036	-4.509206	
Chris Gayle	0.409858	0.014354	-1.793310	-1.094065	
Chris Lynn	-0.686636	-0.283309	-2.544914	-1.401158	
David Warner	0.574301	0.791753	-1.890787	-0.340003	
Faf Du Plessis	-0.466721	-0.110308	-2.814779	-1.472636	
Jonny Bairstow	-0.088623	0.652716	-2.297592	-0.518570	
Jos Buttler	0.000000	-0.965622	-3.323164	-2.234822	
Kane Williamson	-5.310323	0.000000	-6.188112	-3.995256	
Kieron Pollard	-2.951892	0.470782	0.000000	-0.161458	
Marcus Stoinis	-4.712187	-1.131119	-4.804666	0.000000	
Moeen Ali	-3.900915	-0.055282	-4.854388	-2.047408	
Quinton de Kock	-0.542137	-0.004426	-1.384756	-1.273152	
Shane Watson	-0.422441	-0.299606	-1.951912	-1.619388	
Steve Smith	-1.879711	0.002998	-3.729563	-1.917626	

	Moeen Ali	Quinton de Kock	Shane Watson	Steve Smith
AB de Villiers	0.263601	-3.177333	-2.701426	0.402402
Andre Russel	0.585822	-3.176525	-2.930901	-0.184972
Ben Stokes	-5.279731	-9.286885	-8.464639	-6.335553
Chris Gayle	-0.283686	-2.311074	-0.825192	-0.137100
Chris Lynn	-0.675130	-4.204480	-2.472064	-0.754139
David Warner	0.431890	-0.987474	-0.655316	0.684438
Faf Du Plessis	-0.849702	-4.643568	-2.159345	-0.519110
Jonny Bairstow	-0.049678	-4.253256	-2.720731	-0.668040
Jos Buttler	-1.985314	-6.235162	-4.644032	-2.984853
Kane Williamson	-3.472313	-8.463717	-7.646521	-5.268632
Kieron Pollard	-0.692945	-5.152669	-4.855014	-3.049238
Marcus Stoinis	-2.986853	-7.949185	-7.297779	-4.810067
Moeen Ali	0.000000	-7.394533	-6.366026	-4.321665
Quinton de Kock	-0.407927	0.000000	-0.962690	-0.375253
Shane Watson	-0.952716	-3.997976	0.000000	-0.868611
Steve Smith	-0.833003	-6.062567	-4.729501	0.000000

1.4 Step 3 - Calculate ratings from the normalised dominance degree values

$$\zeta_i = \frac{\sum_{j=1}^{m} \delta\left(A_i, A_j\right) - \delta_{\min}}{\delta_{\max} - \delta_{\min}}$$

where

$$\delta_{\min} = \min_i \sum_{j=1}^{m} \delta\left(A_i, A_j\right)$$

$$\delta_{\max} = \max_i \sum_{j=1}^{m} \delta\left(A_i, A_j\right)$$

and $i, j = 1, 2, \ldots, m$

[12]:
```
delta_sums = np.zeros(m)
for i in range(m):
    delta_sums[i] = sum(delta[i,:])
pd.DataFrame(data=delta_sums,index=candidates,columns=['Sum'])
```

[12]:

	Sum
AB de Villiers	-27.726961
Andre Russel	-26.164804
Ben Stokes	-118.336927
Chris Gayle	-22.822275
Chris Lynn	-40.881867
David Warner	-5.721549
Faf Du Plessis	-43.175344
Jonny Bairstow	-30.788434
Jos Buttler	-70.209524
Kane Williamson	-103.617163
Kieron Pollard	-56.137062
Marcus Stoinis	-90.426875
Moeen Ali	-82.291213
Quinton de Kock	-25.554667
Shane Watson	-39.130688
Steve Smith	-61.068302

[13]:
```
delta_min = min(delta_sums)
delta_max = max(delta_sums)
pd.DataFrame(data=[delta_min, delta_max], columns=['Value'], index=['Minimum',␣
↪'Maximum'])
```

[13]:

	Value
Minimum	-118.336927
Maximum	-5.721549

[14]:
```
ratings = (delta_sums - delta_min) / (delta_max - delta_min)
pd.DataFrame(data=ratings, index=candidates, columns=['Rating'])
```

[14]:

	Rating
AB de Villiers	0.804597
Andre Russel	0.818468
Ben Stokes	0.000000
Chris Gayle	0.848149
Chris Lynn	0.687784
David Warner	1.000000

```
Faf Du Plessis   0.667418
Jonny Bairstow   0.777412
Jos Buttler      0.427361
Kane Williamson  0.130708
Kieron Pollard   0.552321
Marcus Stoinis   0.247835
Moeen Ali        0.320078
Quinton de Kock  0.823886
Shane Watson     0.703334
Steve Smith      0.508533
```

1.5 Step 4 - Create ranking based on the calculated ζ_i values

[15]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage[::-1]
```

[16]:
```
result = rank_according_to(ratings)
pd.DataFrame(data=result, index=range(1, m + 1), columns=['Name'])
```

[16]:

	Name
1	David Warner
2	Chris Gayle
3	Quinton de Kock
4	Andre Russel
5	AB de Villiers
6	Jonny Bairstow
7	Shane Watson
8	Chris Lynn
9	Faf Du Plessis
10	Kieron Pollard
11	Steve Smith
12	Jos Buttler
13	Moeen Ali
14	Marcus Stoinis
15	Kane Williamson
16	Ben Stokes

[17]:
```
print("The best candidate/alternative according to C* is " + result[0])
print("The preferences in descending order are " + ", ".join(result) + ".")
```

```
The best candidate/alternative according to C* is David Warner
The preferences in descending order are David Warner, Chris Gayle, Quinton de
Kock, Andre Russel, AB de Villiers, Jonny Bairstow, Shane Watson, Chris Lynn,
Faf Du Plessis, Kieron Pollard, Steve Smith, Jos Buttler, Moeen Ali, Marcus
```

Stoinis, Kane Williamson, Ben Stokes.

TODIM Ranking

May 28, 2021

1 TODIM Ranking

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

1.1 Step 0 - Obtaining and preprocessing data

[2]:
```
bowlers_data = {
    'weights': '../data/bowling_criteria.csv',
    'scores': '../data/bowlers.csv',
}
batsmen_data = {
    'weights': '../data/batting_criteria.csv',
    'scores': '../data/batsmen.csv',
}
data = bowlers_data
```

[3]:
```
attributes_data = pd.read_csv(data['weights'])
attributes_data
```

[3]:

	Name	Ranking	Ideally
0	SR	1	Lower
1	Econ	2	Lower
2	Avg	3	Lower
3	Wkts	4	Higher
4	Runs	5	Lower
5	Inns	6	Higher
6	TBB	7	Higher
7	4w	8	Higher
8	Mat	9	Higher

[4]:
```
benefit_attributes = set()
attributes = []
ranks = []
n = 0
```

```
for i, row in attributes_data.iterrows():
    attributes.append(row['Name'])
    ranks.append(float(row['Ranking']))
    n += 1

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

ranks = np.array(ranks)
```

[5]:
```
weights = 2 * (n + 1 - ranks) / (n * (n + 1))
pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[5]:

	Weight
SR	0.200000
Econ	0.177778
Avg	0.155556
Wkts	0.133333
Runs	0.111111
Inns	0.088889
TBB	0.066667
4w	0.044444
Mat	0.022222

[6]:
```
original_dataframe = pd.read_csv(data['scores'])
candidates = original_dataframe['Name'].to_numpy()
raw_data = pd.DataFrame(original_dataframe, columns=attributes).to_numpy()

dimensions = raw_data.shape
m = dimensions[0]
n = dimensions[1]

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

[6]:

	SR	Econ	Avg	Wkts	Runs	Inns	TBB	4w	Mat
Andre Russell	16.45	9.51	26.09	11.0	287.0	12.0	181.0	0.0	14.0
Ben Stokes	16.83	11.23	31.50	6.0	189.0	6.0	101.0	0.0	9.0
Chris Morris	15.23	9.27	23.54	13.0	306.0	9.0	198.0	0.0	9.0
Dwayne Bravo	22.45	8.02	30.00	11.0	330.0	12.0	247.0	0.0	12.0
Imran Tahir	14.85	6.70	16.58	26.0	431.0	17.0	386.0	2.0	17.0
Jofra Archer	23.45	6.77	26.45	11.0	291.0	11.0	258.0	0.0	11.0
Kagiso Rabada	11.28	7.83	14.72	25.0	368.0	12.0	282.0	2.0	12.0
Keemo Paul	18.11	8.72	26.33	9.0	237.0	8.0	163.0	0.0	8.0
Lasith Malinga	16.81	9.77	27.38	16.0	438.0	12.0	269.0	2.0	12.0
Moeen Ali	25.00	6.76	28.17	6.0	169.0	9.0	150.0	0.0	11.0
Mohammad Nabi	21.88	6.65	24.25	8.0	194.0	8.0	175.0	1.0	8.0

Rashid Khan	21.18	6.28	22.18	17.0	377.0	15.0	360.0	0.0	15.0
Sam Curran	19.80	9.79	32.30	10.0	323.0	9.0	198.0	1.0	9.0
Sunil Narine	26.60	7.83	34.70	10.0	347.0	12.0	266.0	0.0	12.0
Trent Boult	22.80	8.58	32.60	5.0	163.0	5.0	114.0	0.0	5.0

1.2 Step 1 - Normalizing the Ratings And Weights

$$P_{ij} = \begin{cases} \frac{x_{ij}}{\sum_{k=1}^{m} x_{kj}} & \text{if } j \in J_1 \\ \frac{\frac{1}{x_{ij}}}{\sum_{k=1}^{m} \frac{1}{x_{kj}}} & \text{if } j \in J_2 \end{cases}$$

$$w_{rc} = \frac{w_c}{w_r}$$

and $w_r = \max\{w_c | c = 1, 2, \ldots, n\}$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[7]:
```
for j in range(n):
    column = raw_data[:,j]
    if j in benefit_attributes:
        raw_data[:,j] /= sum(column)
    else:
        column = 1 / column
        raw_data[:,j] = column / sum(column)

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

[7]:

	SR	Econ	Avg	Wkts	Runs	Inns	\
Andre Russell	0.075177	0.056158	0.064018	0.059783	0.062346	0.076433	
Ben Stokes	0.073479	0.047557	0.053023	0.032609	0.094673	0.038217	
Chris Morris	0.081199	0.057612	0.070953	0.070652	0.058474	0.057325	
Dwayne Bravo	0.055085	0.066592	0.055674	0.059783	0.054222	0.076433	
Imran Tahir	0.083277	0.079712	0.100737	0.141304	0.041515	0.108280	
Jofra Archer	0.052736	0.078887	0.063146	0.059783	0.061489	0.070064	
Kagiso Rabada	0.109633	0.068208	0.113466	0.135870	0.048623	0.076433	
Keemo Paul	0.068286	0.061246	0.063434	0.048913	0.075499	0.050955	
Lasith Malinga	0.073567	0.054664	0.061002	0.086957	0.040852	0.076433	
Moeen Ali	0.049466	0.079004	0.059291	0.032609	0.105877	0.057325	
Mohammad Nabi	0.056520	0.080311	0.068875	0.043478	0.092233	0.050955	
Rashid Khan	0.058388	0.085043	0.075303	0.092391	0.047462	0.095541	
Sam Curran	0.062457	0.054552	0.051710	0.054348	0.055397	0.057325	
Sunil Narine	0.046491	0.068208	0.048133	0.054348	0.051565	0.076433	
Trent Boult	0.054239	0.062246	0.051234	0.027174	0.109774	0.031847	

	TBB	4w	Mat
Andre Russell	0.054062	0.000	0.085366
Ben Stokes	0.030167	0.000	0.054878

```
Chris Morris     0.059140  0.000  0.054878
Dwayne Bravo     0.073775  0.000  0.073171
Imran Tahir      0.115293  0.250  0.103659
Jofra Archer     0.077061  0.000  0.067073
Kagiso Rabada    0.084229  0.250  0.073171
Keemo Paul       0.048686  0.000  0.048780
Lasith Malinga   0.080346  0.250  0.073171
Moeen Ali        0.044803  0.000  0.067073
Mohammad Nabi    0.052270  0.125  0.048780
Rashid Khan      0.107527  0.000  0.091463
Sam Curran       0.059140  0.125  0.054878
Sunil Narine     0.079450  0.000  0.073171
Trent Boult      0.034050  0.000  0.030488
```

[8]:
```
max_weight = max(weights)
weights /= max_weight

pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[8]:
```
        Weight
SR    1.000000
Econ  0.888889
Avg   0.777778
Wkts  0.666667
Runs  0.555556
Inns  0.444444
TBB   0.333333
4w    0.222222
Mat   0.111111
```

1.3 Step 2 - Calculating Dominance Degrees

For the contribution of each criteria, we have:

$$\Phi_c\left(A_i, A_j\right) = \begin{cases} \sqrt{\frac{(P_{ic}-P_{jc})w_{rc}}{\sum_{c=1}^{n} w_{rc}}} & \text{if } P_{ic} - P_{jc} > 0 \\ 0 & \text{if } P_{ic} - P_{jc} = 0 \\ -\frac{1}{\theta}\sqrt{\frac{\left(\sum_{c=1}^{n} w_{rc}\right)(P_{jc}-P_{ic})}{w_{rc}}} & \text{if } P_{ic} - P_{jc} < 0 \end{cases}$$

Combining all contributions, we get the dominance degrees:

$$\delta\left(A_i, A_j\right) = \sum_{c=1}^{n} \Phi_c\left(A_i, A_j\right)$$

Here $c = 1, 2, \ldots, n$, $i, j = 1, 2, \ldots, m$.

[9]:
```
# The loss attenuation factor
theta = 1.0
```

[10]:
```
phi = np.zeros((n, m, m))

weight_sum = sum(weights)

for c in range(n):
    for i in range(m):
        for j in range(m):
            pic = raw_data[i,c]
            pjc = raw_data[j,c]
            val = 0
            if pic > pjc:
                val = math.sqrt((pic - pjc) * weights[c] / weight_sum)
            if pic < pjc:
                val = -1.0 / theta * math.sqrt(weight_sum * (pjc - pic) /␣
 ↪weights[c])
            phi[c, i, j] = val
```

[11]:
```
delta = np.zeros((m, m))
for i in range(m):
    for j in range(m):
        delta[i,j] = sum(phi[:,i,j])

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

[11]:

	Andre Russell	Ben Stokes	Chris Morris	Dwayne Bravo	\
Andre Russell	0.000000	-0.256091	-0.948618	-0.640115	
Ben Stokes	-3.395140	0.000000	-2.367408	-3.153278	
Chris Morris	-1.681511	-0.280010	0.000000	-1.883396	
Dwayne Bravo	-1.480425	-0.635606	-1.023334	0.000000	
Imran Tahir	0.094481	-0.072683	0.124330	0.210453	
Jofra Archer	-1.569885	-0.568460	-0.722302	-0.775823	
Kagiso Rabada	-0.624253	-0.039962	0.203299	0.224265	
Keemo Paul	-2.566535	-0.895393	-1.996320	-2.520664	
Lasith Malinga	-1.293810	-0.294205	-0.724292	-0.329792	
Moeen Ali	-2.594644	-0.116315	-1.519322	-2.119307	
Mohammad Nabi	-2.412364	-0.652604	-1.831497	-2.249504	
Rashid Khan	-0.363674	-0.525040	-0.359261	0.066327	
Sam Curran	-2.622509	-0.672382	-1.230612	-2.336888	
Sunil Narine	-1.865025	-0.917318	-1.257069	-0.747582	
Trent Boult	-3.826860	-1.826479	-3.386515	-3.672372	

	Imran Tahir	Jofra Archer	Kagiso Rabada	Keemo Paul	\
Andre Russell	-6.620828	-0.812562	-4.983520	-0.333476	
Ben Stokes	-7.897602	-3.179277	-7.033048	-1.701473	

Chris Morris	-7.104803	-2.000113	-6.166372	-0.333710
Dwayne Bravo	-6.860415	-0.902784	-4.725282	-0.737200
Imran Tahir	0.000000	0.089699	-0.705105	0.024531
Jofra Archer	-6.753051	0.000000	-5.267292	-0.478002
Kagiso Rabada	-2.763542	-0.126805	0.000000	0.055154
Keemo Paul	-7.602526	-2.521952	-6.657522	0.000000
Lasith Malinga	-4.310707	-0.637133	-2.392892	-0.549940
Moeen Ali	-7.248964	-1.736133	-6.023291	-0.902745
Mohammad Nabi	-6.613377	-2.124177	-5.719613	-0.224017
Rashid Khan	-5.138686	-0.063239	-3.890964	-0.384254
Sam Curran	-6.862080	-2.596379	-5.808854	-0.901349
Sunil Narine	-6.913739	-1.185152	-4.613248	-0.929599
Trent Boult	-8.314198	-3.728651	-7.495802	-2.713294

	Lasith Malinga	Moeen Ali	Mohammad Nabi	Rashid Khan	\
Andre Russell	-3.329825	-0.739164	-2.546245	-2.951349	
Ben Stokes	-5.810701	-2.542547	-3.579048	-4.543259	
Chris Morris	-4.510778	-1.516449	-2.380317	-3.652038	
Dwayne Bravo	-3.541567	-0.908027	-2.760399	-3.355487	
Imran Tahir	0.410632	-0.197184	-0.230257	-0.012023	
Jofra Archer	-4.027618	-0.467397	-2.473359	-3.347820	
Kagiso Rabada	0.350569	-0.435901	-0.398507	-1.898623	
Keemo Paul	-5.224812	-1.864451	-2.736070	-4.146433	
Lasith Malinga	0.000000	-0.757209	-0.961624	-3.011334	
Moeen Ali	-5.029017	0.000000	-2.736111	-3.978705	
Mohammad Nabi	-4.593938	-1.299065	0.000000	-3.928230	
Rashid Khan	-2.368579	-0.364650	-1.996668	0.000000	
Sam Curran	-4.571475	-1.796194	-1.159252	-3.995724	
Sunil Narine	-3.554167	-1.180616	-2.980542	-3.502250	
Trent Boult	-6.406009	-2.904881	-4.638627	-5.064456	

	Sam Curran	Sunil Narine	Trent Boult
Andre Russell	-1.719993	-0.674008	-0.528739
Ben Stokes	-3.274775	-2.997130	-0.744730
Chris Morris	-1.472675	-1.949796	-0.524199
Dwayne Bravo	-1.781258	-0.267294	-0.428851
Imran Tahir	0.208900	0.261912	-0.138995
Jofra Archer	-1.651956	-0.793466	-0.442198
Kagiso Rabada	0.278570	0.277966	-0.128276
Keemo Paul	-2.907840	-2.495736	-0.387368
Lasith Malinga	-0.032746	-0.289125	-0.548951
Moeen Ali	-2.607814	-1.924619	-0.121989
Mohammad Nabi	-1.387018	-2.217326	-0.049529
Rashid Khan	-1.737908	0.152185	-0.319922
Sam Curran	0.000000	-2.024779	-0.611952
Sunil Narine	-2.149481	0.000000	-0.820291
Trent Boult	-4.468290	-3.412196	0.000000

1.4 Step 3 - Calculate ratings from the normalised dominance degree values

$$\zeta_i = \frac{\sum_{j=1}^{m} \delta\left(A_i, A_j\right) - \delta_{\min}}{\delta_{\max} - \delta_{\min}}$$

where

$$\delta_{\min} = \min_i \sum_{j=1}^{m} \delta\left(A_i, A_j\right)$$

$$\delta_{\max} = \max_i \sum_{j=1}^{m} \delta\left(A_i, A_j\right)$$

and $i, j = 1, 2, \ldots, m$

[12]:
```
delta_sums = np.zeros(m)
for i in range(m):
    delta_sums[i] = sum(delta[i,:])
pd.DataFrame(data=delta_sums,index=candidates,columns=['Sum'])
```

[12]:

	Sum
Andre Russell	-27.084533
Ben Stokes	-52.219415
Chris Morris	-35.456166
Dwayne Bravo	-29.407928
Imran Tahir	0.068690
Jofra Archer	-29.338628
Kagiso Rabada	-5.026045
Keemo Paul	-44.523623
Lasith Malinga	-16.133760
Moeen Ali	-38.658977
Mohammad Nabi	-35.302259
Rashid Khan	-17.294333
Sam Curran	-37.190432
Sunil Narine	-32.616078
Trent Boult	-61.858631

[13]:
```
delta_min = min(delta_sums)
delta_max = max(delta_sums)
pd.DataFrame(data=[delta_min, delta_max], columns=['Value'], index=['Minimum',␣
↪'Maximum'])
```

[13]:

	Value
Minimum	-61.858631
Maximum	0.068690

[14]:
```
ratings = (delta_sums - delta_min) / (delta_max - delta_min)
pd.DataFrame(data=ratings, index=candidates, columns=['Rating'])
```

[14]:

	Rating
Andre Russell	0.561531
Ben Stokes	0.155654
Chris Morris	0.426346
Dwayne Bravo	0.524013
Imran Tahir	1.000000
Jofra Archer	0.525132
Kagiso Rabada	0.917730
Keemo Paul	0.279925
Lasith Malinga	0.738363
Moeen Ali	0.374627
Mohammad Nabi	0.428831
Rashid Khan	0.719623
Sam Curran	0.398341
Sunil Narine	0.472208
Trent Boult	0.000000

1.5 Step 4 - Create ranking based on the calculated ζ_i values

[15]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage[::-1]
```

[16]:
```
result = rank_according_to(ratings)
pd.DataFrame(data=result, index=range(1, m + 1), columns=['Name'])
```

[16]:

	Name
1	Imran Tahir
2	Kagiso Rabada
3	Lasith Malinga
4	Rashid Khan
5	Andre Russell
6	Jofra Archer
7	Dwayne Bravo
8	Sunil Narine
9	Mohammad Nabi
10	Chris Morris
11	Sam Curran
12	Moeen Ali
13	Keemo Paul
14	Ben Stokes
15	Trent Boult

[17]:
```
print("The best candidate/alternative according to C* is " + result[0])
print("The preferences in descending order are " + ", ".join(result) + ".")
```

```
The best candidate/alternative according to C* is Imran Tahir
The preferences in descending order are Imran Tahir, Kagiso Rabada, Lasith
Malinga, Rashid Khan, Andre Russell, Jofra Archer, Dwayne Bravo, Sunil Narine,
Mohammad Nabi, Chris Morris, Sam Curran, Moeen Ali, Keemo Paul, Ben Stokes,
Trent Boult.
```