OPERATION RESEARCH

The criteria for the pre-selection are:

> • Viscosity of the neat resin should be less than 1 Pa·s (1000 cps) to enable VARIM processing. • Tg (glass transition temperature) of the neat resin should preferable be higher than 50-60°C. • The process temperature should be lower than 230°C, so that low cost accessories can be used. • The resin should have a long pot life. • Cost of the resin should be affordable. • Availability of basic knowledge and technology.

Algorithm

1. Elimination Search according to constraints d
2. Obtain the decision matrix and relative importance matrix a
3. Normalize the decision matrix using the method from TOPSIS rij=xij/Σ(xij)^0.5
4. Normalize the relative importance matrix using the Geometric Mean from AHP Method GMi=(πaij)^(1/n) by using AHP
5. Eigen value calculation and multiplying decision matrix by weights to obtain normalized weighted matrix
6. Obtain the best and worst solution from weighted normalized matrix using TOPSIS
7. Obtain best soln using calculating the Euclidean distance and then ranking them according to RSI score.RSI=S_minus/S_plus+S_minus

In [1]:
```
#Using the suitable libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
%matplotlib inline
```

In [2]:
```
#Loading the Excel sheet containing the Decision matrix
df=pd.read_excel("")
df
```

Out[2]:

	Attributes Material	Viscosity (Pa s)	Processing temperature(Celsius)\n	Cost(€/kg)	Availability
0	Polymethylmethacrylate (PMMA)	0.10	140	4.0	0.75
1	Polyamide 6 (PA 6)	0.01	150	2.5	1.00
2	Polyamide 12 (PA 12)	0.05	220	20.0	0.75
3	Polyethylene Terephtalate (PET)	0.95	280	3.0	0.50
4	Thermoplastic Polyurethane (TPU)	0.80	300	10.0	0.75
5	Polybutylene Terephtalate (PBT)	0.02	200	9.0	1.00
6	Polycarbonate (PC)	0.70	250	5.0	0.25
7	Polyether Ketone (PEK)	0.10	350	60.0	0.25

	Attributes Material	Viscosity (Pa s)	Processing temperature(Celsius)\n	Cost(€/kg)	Availability
8	Polyphthalamide (PPA)	1.00	250	5.5	0.75

In [3]:
```
d=df.drop(columns=df.columns[0])
```

In [4]:
```
d
```

Out[4]:

	Viscosity (Pa s)	Processing temperature(Celsius)\n	Cost(€/kg)	Availability
0	0.10	140	4.0	0.75
1	0.01	150	2.5	1.00
2	0.05	220	20.0	0.75
3	0.95	280	3.0	0.50
4	0.80	300	10.0	0.75
5	0.02	200	9.0	1.00
6	0.70	250	5.0	0.25
7	0.10	350	60.0	0.25
8	1.00	250	5.5	0.75

In [5]:
```
#Normalizing the decision matrix
def den(i,j,d):
    sum_=0
    for i in range(0,d.shape[0]):
        sum_+=(d[i][j])**2
    return sum_
d=np.array(d)
r=np.zeros((d.shape[0],d.shape[1]))
for i in range(0,d.shape[0]):
    for j in range(0,d.shape[1]):
        r[i][j]=d[i][j]/(den(i,j,d))**0.5
```

In [6]:
```
r
```

Out[6]:
```
array([[0.05720828, 0.18939682, 0.06123126, 0.34874292],
       [0.00572083, 0.20292517, 0.03826954, 0.46499055],
       [0.02860414, 0.29762358, 0.30615632, 0.34874292],
       [0.54347862, 0.37879365, 0.04592345, 0.23249528],
       [0.4576662 , 0.40585034, 0.15307816, 0.34874292],
       [0.01144166, 0.27056689, 0.13777034, 0.46499055],
       [0.40045793, 0.33820861, 0.07653908, 0.11624764],
       [0.05720828, 0.47349206, 0.91846896, 0.11624764],
       [0.57208275, 0.33820861, 0.08419299, 0.34874292]])
```

In [7]:
```
#Loading the Excel sheet containing the realtive importance matrix
df1=pd.read_excel("")
df1
```

Out[7]:

	Unnamed: 0	Viscosity	Processing temp	cost	availability
0	Viscosity	1	1	1	1
1	Processing temp	1	1	1	1

	Unnamed: 0	Viscosity	Processing temp	cost	availability
2	cost	1	1	1	1
3	availability	1	1	1	1

In [8]:
```
df1.drop(columns=['Unnamed: 0'],inplace=True)
```

In [9]:
```
df1
```

Out[9]:

	Viscosity	Processing temp	cost	availability
0	1	1	1	1
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1

In [10]:
```
#To remove the bias, incorporating the AHP method for calcualtion of weights
def gm(df1):
    gmi=np.ones((df1.shape[0]))
    for i in range(0,df1.shape[0]):
        for j in range(0,df1.shape[1]):
            gmi[i]*=df1[i][j]
        gmi[i]=gmi[i]**(1/df1.shape[1])
    return gmi

df1_np=np.array(df1)
gmi=gm(df1_np)
w=np.zeros(gmi.shape[0])
for i in range(0,gmi.shape[0]):
    w[i]=gmi[i]/sum(gmi)

N2=np.transpose(w)
N3=np.dot(df1_np,N2)
N4=N3/N2
N2,N3,N4,gmi,df1_np,w
```

Out[10]:
```
(array([0.25, 0.25, 0.25, 0.25]),
 array([1., 1., 1., 1.]),
 array([4., 4., 4., 4.]),
 array([1., 1., 1., 1.]),
 array([[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]], dtype=int64),
 array([0.25, 0.25, 0.25, 0.25]))
```

In [11]:
```
#Calculating the eigen values using the relative importance matrix
eigen,righteigen=np.linalg.eig(df1_np)
```

In [12]:
```
eigen,righteigen
```

Out[12]:
```
(array([0.00000000e+00, 4.00000000e+00, 0.00000000e+00, 2.80731443e-32]),
 array([[-0.8660254 , -0.5       , -0.8660254 , -0.64641535],
        [ 0.28867513, -0.5       ,  0.28867513, -0.32788993],
        [ 0.28867513, -0.5       ,  0.28867513,  0.48715264],
        [ 0.28867513, -0.5       ,  0.28867513,  0.48715264]]))
```

In [13]:
```
lambdamax=max(eigen)
lambdamax
```

Out[13]: 4.0

In [14]:
```
#Calculating the consistency ratio to ensure best solution. CR=CI/RI. CR<=1
#Random index values corresponding to different matrix sizes in array ri
ri=np.array([0.00,0.00,0.58,0.90,1.12,1.24,1.32,1.41,1.45,1.49])
ci=(lambdamax-df1_np.shape[0])/(df1_np.shape[0]-1)
consistency_ratio=ci/ri[df1_np.shape[0]-1]
consistency_ratio
```

Out[14]: 0.0

In [15]:
```
#Obtaining the weighted normalized matrix
V=r*w
V
```

Out[15]:
```
array([[0.01430207, 0.04734921, 0.01530782, 0.08718573],
       [0.00143021, 0.05073129, 0.00956739, 0.11624764],
       [0.00715103, 0.07440589, 0.07653908, 0.08718573],
       [0.13586965, 0.09469841, 0.01148086, 0.05812382],
       [0.11441655, 0.10146258, 0.03826954, 0.08718573],
       [0.00286041, 0.06764172, 0.03444259, 0.11624764],
       [0.10011448, 0.08455215, 0.01913477, 0.02906191],
       [0.01430207, 0.11837301, 0.22961724, 0.02906191],
       [0.14302069, 0.08455215, 0.02104825, 0.08718573]])
```

In [16]:
```
#Obtaining the best and worst system from matrix V
V_plus=np.zeros(V.shape[1])
V_minus=np.zeros(V.shape[1])
for j in range(0,V.shape[1]):
    V_plus[j]=np.max(V[:,[j]])
    V_minus[j]=np.min(V[:,[j]])
V_plus,V_minus
```

Out[16]:
```
(array([0.14302069, 0.11837301, 0.22961724, 0.11624764]),
 array([0.00143021, 0.04734921, 0.00956739, 0.02906191]))
```

In [17]:
```
#Caluculating the Euclidean distance from best and worst system of each entry in V.
#RSI=S_minus/S_plus+S_minus
S_plus=np.zeros(V.shape[0])
S_minus=np.zeros(V.shape[0])
RSI=np.zeros(V.shape[0])
for i in range(0,V.shape[0]):
    for j in range(0,V.shape[1]):
        S_plus[i]+=(V[i][j]-V_plus[j])**2
        S_minus[i]+=(V[i][j]-V_minus[j])**2
    S_plus[i]=S_plus[i]**0.5
    S_minus[i]=S_minus[i]**0.5
    RSI[i]=S_minus[i]/(S_plus[i]+S_minus[i])
```

In [18]:
```
RSI
```

Out[18]:
```
array([0.18613542, 0.24404589, 0.30531042, 0.39046714, 0.41802793,
       0.27449964, 0.31129606, 0.5983321 , 0.42544464])
```

In [19]:
```
#Sorting the items according to the RSI Score
sortedRSI=np.sort(RSI)[::-1]
sortedRSI_index=np.zeros(RSI.shape[0])
for i in range(0,RSI.shape[0]):
    sortedRSI_index[i]=int(np.where(RSI==sortedRSI[i])[0]+1)
sortedRSI_index=sortedRSI_index.astype(int)
RSI,sortedRSI,sortedRSI_index
```

Out[19]:
```
(array([0.18613542, 0.24404589, 0.30531042, 0.39046714, 0.41802793,
        0.27449964, 0.31129606, 0.5983321 , 0.42544464]),
```

```
array([0.5983321 , 0.42544464, 0.41802793, 0.39046714, 0.31129606,
       0.30531042, 0.27449964, 0.24404589, 0.18613542]),
array([8, 9, 5, 4, 7, 3, 6, 2, 1]))
```

In [20]:

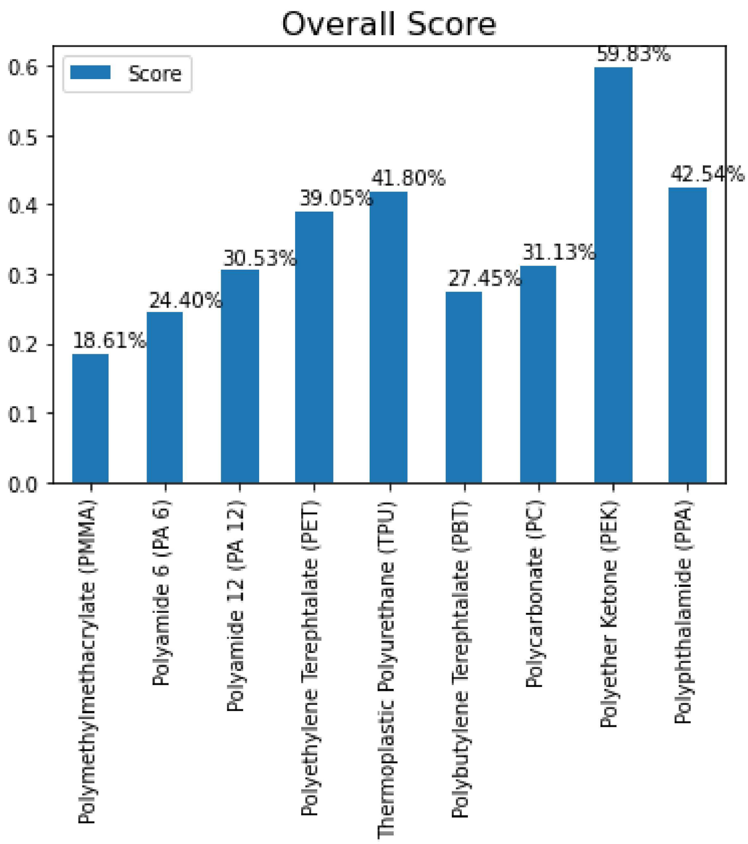
```
#Obtaining the list of items in best to worst order and obtaining the bar graph to s
names=list(df.columns)
items=list(df[names[0]])
for i in range(0,sortedRSI_index.shape[0]):
     print(items[sortedRSI_index[i]-1])
df_final = {names[0]:items, 'Score': RSI}
df_final= pd.DataFrame(data=df_final)
ax=df_final.plot(kind='bar')
ax.set_title("Overall Score",fontsize=16)
for p in ax.patches:
    ax.annotate("{:.2%}".format(p.get_height()),
                xy=(p.get_x()+0.02, p.get_height()+0.01))

ax.set_xticklabels(items)
```

```
Polyether Ketone (PEK)
Polyphthalamide (PPA)
Thermoplastic Polyurethane (TPU)
Polyethylene Terephtalate (PET)
Polycarbonate (PC)
Polyamide 12 (PA 12)
Polybutylene Terephtalate (PBT)
Polyamide 6 (PA 6)
Polymethylmethacrylate (PMMA)
```

Out[20]:

```
[Text(0, 0, 'Polymethylmethacrylate (PMMA)'),
 Text(1, 0, 'Polyamide 6 (PA 6)'),
 Text(2, 0, 'Polyamide 12 (PA 12)'),
 Text(3, 0, 'Polyethylene Terephtalate (PET)'),
 Text(4, 0, 'Thermoplastic Polyurethane (TPU)'),
 Text(5, 0, 'Polybutylene Terephtalate (PBT)'),
 Text(6, 0, 'Polycarbonate (PC)'),
 Text(7, 0, 'Polyether Ketone (PEK)'),
 Text(8, 0, 'Polyphthalamide (PPA)')]
```



In [21]:

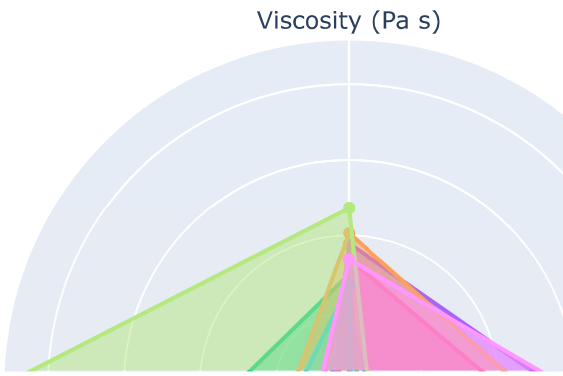
```
#For obtaining the spider diagram also known as the Radar plot
```

```
list_name = names
attributes= items
fig = go.Figure()

for i in range(0,V.shape[0]):
    fig.add_trace(go.Scatterpolar(
          r=V[i],
          theta=list_name,
          fill='toself',
          name=attributes[i]
    ))

fig.update_layout(
  polar=dict(
    radialaxis=dict(
      visible=True,
      range=[np.min(V),np.max(V)]
    )),
  showlegend=True
)

fig.show()
```



In [ ]: