Trabalho 2 - Clusterizando com *Minisom* e *K-Means*

Pedro Miguel Pinto Botelho, Ricardo Augusto Silva Bonfim,
Rômulo José Pereira Da Costa Junior

[1]Instituto de Computação – Universidade Federal do Amazonas(UFAM)
Av. Gen. Rodrigo Octávio,6200, Coroado I,Setor Norte do Campus–69080–900

{pedro.botelho, ricardo.bonfim, romulo.junior}@icomp.ufam.edu.br

Introdução

A clusterização é uma técnica que permite organização de dados similares em grupos distintos. Tanto o *MiniSOM* quanto o algoritmo *K-means* são amplamente utilizados para essa finalidade. O *MiniSOM*, uma variação da Rede Neural de Kohonen, projeta dados em um espaço de baixa dimensionalidade, onde neurônios representam clusters potenciais ajustados durante o treinamento. Enquanto isso, o *K-means* divide o conjunto de dados em K *clusters*, recalculando iterativamente centróides até a convergência. A escolha entre eles depende da natureza do problema e dos requisitos do conjunto de dados, com o *MiniSOM* adequado para alta dimensionalidade e preservação de topologia, e o *K-means* mais simples e eficiente para conjuntos de dados grandes e bem definidos.

1. Considerações iniciais

Decidimos em remover as colunas: “id”, pois a mesma é usada no csv apenas para identificação, e como ela não é necessária para fazer o rastreio dos dados, ela não é útil; além de todas as demais (com exceção das colunas “*Country*”, “*Raised*”, “ESG”, “E”, “S”,“G”, que são as de interesse). Além disso, vale ressaltar que fizemos diversos testes (incluindo retirando as colunas de interesse por completo, apenas algumas de interesse, etc), e essa configuração foi a que trouxe melhores resultados.

2. Clusterização com *MiniSom*

Para começar a Clusterização com *MiniSom*, tivemos que fazer alguns ajustes no código

2.1. Preparação de dados e execução do treinamento

Iniciamos fazendo uma cópia do *dataframe* original (X) e selecionamos apenas as colunas de interesse:

```
X = df.copy(deep = True)
X = X[["Raised", "E", "S", "G","ESG","Country"]]
X
```

	Raised	E	S	G	ESG	Country
0	5986183.0	0.008	0.002	0.000	0.010	Lithuania
1	0.0	0.010	0.006	0.002	0.018	Netherlands
2	3500000.0	0.010	0.000	0.004	0.014	Austria
3	0.0	0.006	0.000	0.002	0.008	Hong Kong SAR, China
4	0.0	0.000	0.011	0.006	0.017	United States
...	...	...	...	...	...	...
6536	0.0	0.000	0.000	0.000	0.000	United States
6537	0.0	0.000	0.000	0.000	0.000	Colombia
6538	0.0	0.000	0.000	0.000	0.000	Spain
6539	0.0	0.000	0.000	0.000	0.000	Belarus
6540	0.0	0.000	0.000	0.000	0.000	Thailand

6541 rows x 6 columns

Figura 1. Selecionando apenas as colunas de interesse

Depois, separamos o *dataframe* nas colunas numéricas e não numéricas (com *Country* sendo a única não numérica).

```
[ ]  XsemC = X.drop(['Country'], axis=1)
     country = X['Country']
```

Figura 2. Separando as colunas numéricas e colunas não numéricas

Após isso, normalizamos os dados numéricos com uma escala de 0 a 1. Não é estritamente necessário que os dados estejam normalizados para funcionar, mas é interessante que estejam, principalmente quando as *features* tem *ranges* muito diferentes.

```
sc = MinMaxScaler(feature_range=(0,1)).set_output(transform="pandas")
sc.fit(XsemC)
XsemC=sc.transform(XsemC)
type(XsemC)
XsemC
```

	Raised	E	S	G	ESG
0	0.005986	0.100	0.033333	0.000000	0.166667
1	0.000000	0.125	0.100000	0.033333	0.300000
2	0.003500	0.125	0.000000	0.066667	0.233333
3	0.000000	0.075	0.000000	0.033333	0.133333
4	0.000000	0.000	0.183333	0.100000	0.283333
...	...	...	...	...	...
6536	0.000000	0.000	0.000000	0.000000	0.000000
6537	0.000000	0.000	0.000000	0.000000	0.000000
6538	0.000000	0.000	0.000000	0.000000	0.000000
6539	0.000000	0.000	0.000000	0.000000	0.000000
6540	0.000000	0.000	0.000000	0.000000	0.000000

6541 rows × 5 columns

Figura 3. Separando as colunas numéricas e colunas não numéricas

Após a normalização das *features*, podemos incluir novamente a coluna *country*.

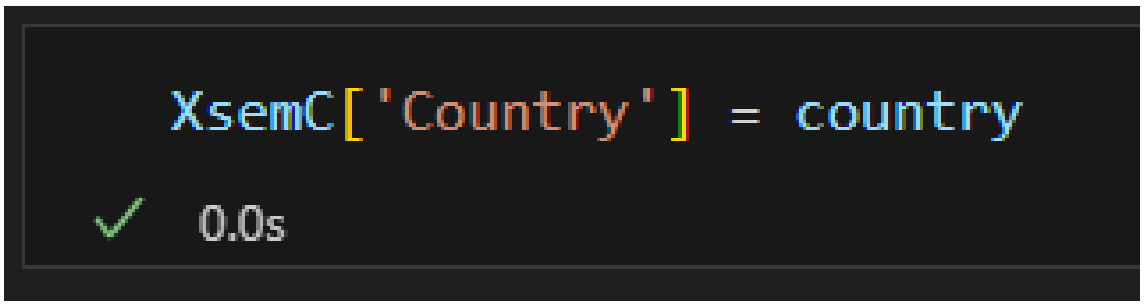
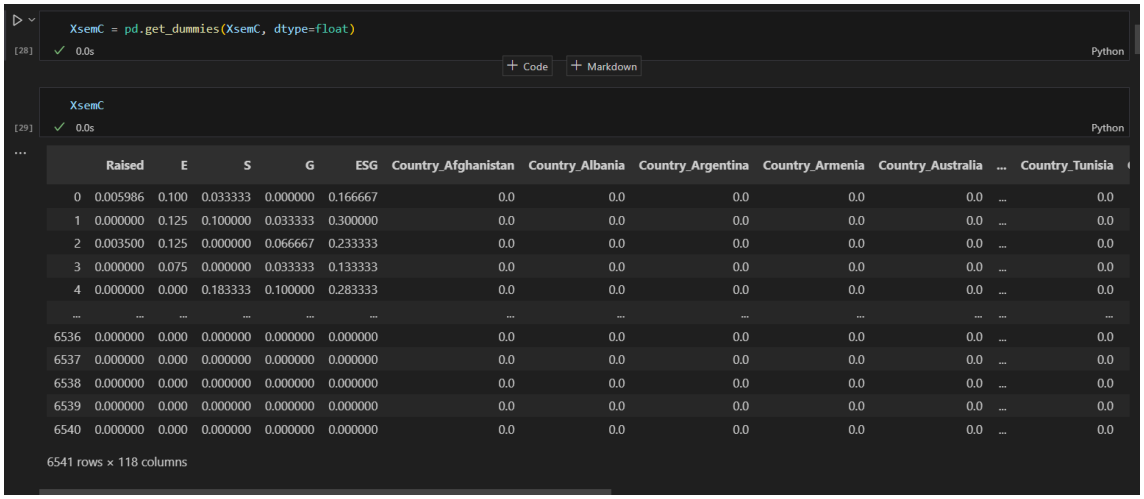


Figura 4. Incluindo novamente a coluna *country*

Depois de incluirmos a coluna com dados não numéricos, usamos a função *get_dummies* do pandas para incluir esses dados de texto como algo numérico para nosso método conseguir aprender sobre.



```
XsemC = pd.get_dummies(XsemC, dtype=float)
```

0.0s — Python

+ Code + Markdown

```
XsemC
```

0.0s — Python

	Raised	E	S	G	ESG	Country_Afghanistan	Country_Albania	Country_Argentina	Country_Armenia	Country_Australia	...	Country_Tunisia
0	0.005986	0.100	0.033333	0.000000	0.166667	0.0	0.0	0.0	0.0	0.0	...	0.0
1	0.000000	0.125	0.100000	0.033333	0.300000	0.0	0.0	0.0	0.0	0.0	...	0.0
2	0.003500	0.125	0.000000	0.066667	0.233333	0.0	0.0	0.0	0.0	0.0	...	0.0
3	0.000000	0.075	0.000000	0.033333	0.133333	0.0	0.0	0.0	0.0	0.0	...	0.0
4	0.000000	0.000	0.183333	0.100000	0.283333	0.0	0.0	0.0	0.0	0.0	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
6536	0.000000	0.000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0
6537	0.000000	0.000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0
6538	0.000000	0.000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0
6539	0.000000	0.000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0
6540	0.000000	0.000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0

6541 rows × 118 columns

Figura 5. *get_dummies* usado nos dados

após usarmos método *get_dummies*, temos um dataset pronto para ser usado. Portanto, o próximo passo é rodar o método de aprendizagem. Primeiro, rodamos com um tamanho de mapa grande para poder salientar os *clusters* de dados que temos.

```
#data = X.values
data = XsemC.values
som_shape = (118, 118)
som = MiniSom(som_shape[0], som_shape[1], 118, sigma=1.5, learning_rate=.5,
              neighborhood_function='gaussian', random_seed=0)
#som.train_batch(data, 500, verbose=True)

som.pca_weights_init(data)
som.train(data, 1000, verbose=True)
```

22.3s

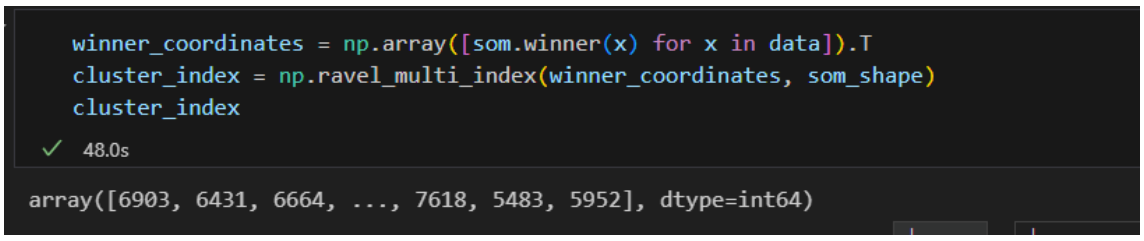
```
c:\Users\ricar\AppData\Local\Programs\Python\Python312\Lib\site-packages\minisom.py:384: ComplexWarning:

Casting complex values to real discards the imaginary part

 [ 1000 / 1000 ] 100% - 0:00:00 left
 quantization error: 0.09888939635517786
```

Figura 6. Rodando *MiniSom* para o dataset, e usando 1000 epochs pois temos uma complexidade alta

Após terminar de treinar, *clusterizamos* os dados.



```
winner_coordinates = np.array([som.winner(x) for x in data]).T
cluster_index = np.ravel_multi_index(winner_coordinates, som_shape)
cluster_index
```

48.0s

```
array([6903, 6431, 6664, ..., 7618, 5483, 5952], dtype=int64)
```

Figura 7. Enter Caption

E plotamos para podermos visualizar os *clusters* de dados. Detalhe é que devido a alta complexidade da grade (118x118), temos diversos *clusters* presentes (cada cor diferente é para ser um *cluster*, mas vendo pelo gráfico fica evidente que temos 4 *clusters* principais de dados.

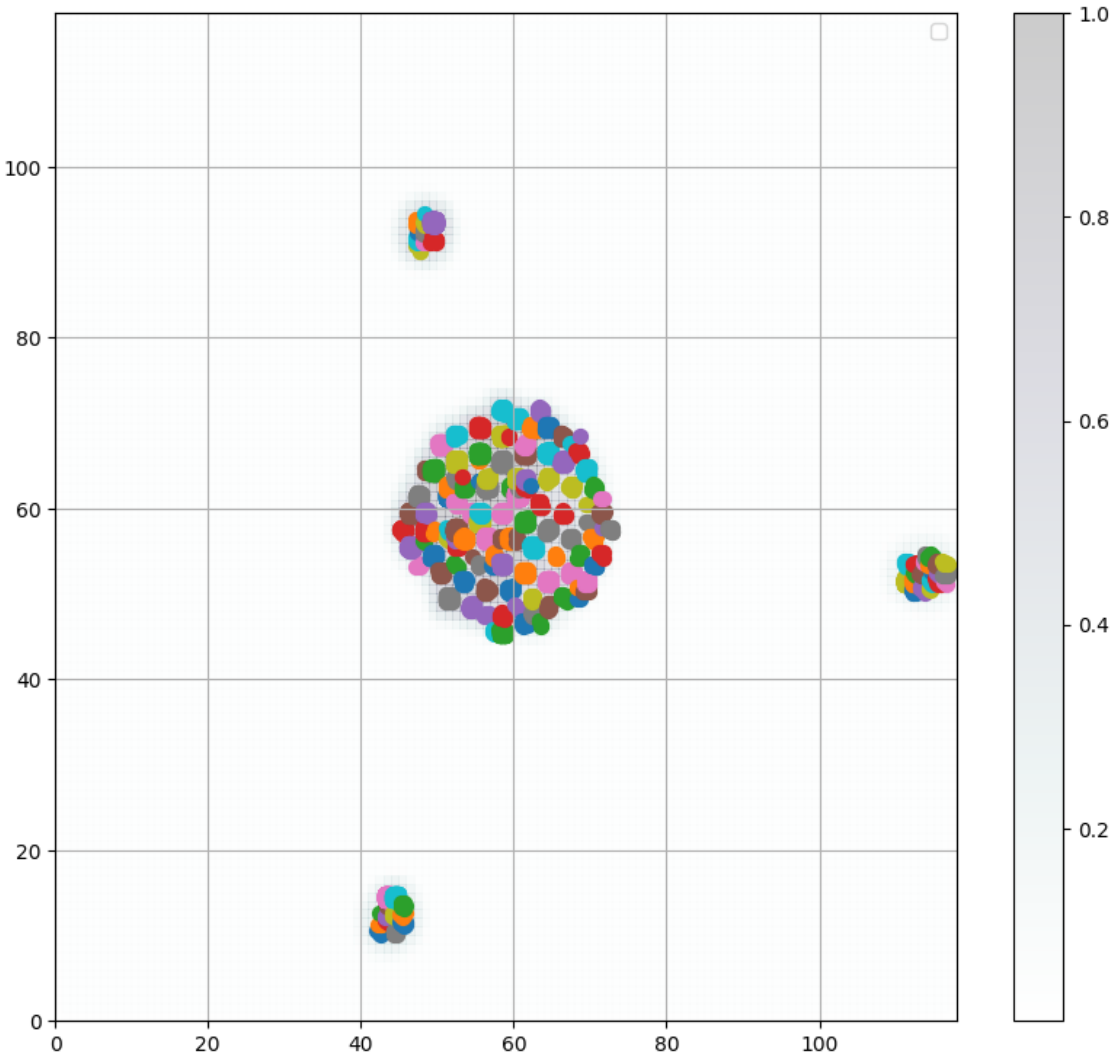


Figura 8. clusterização de dados com 118 *features* e grade 118x118

2.2. Análises alternativas

Além disso, também fizemos um teste não normalizando a coluna *Raised*, que também será comentado na análise gráfica.

3. Clusterização com *K-Means*

Para o método do *K-Means*, seguimos exatamente os passos do *Visualizing High Dimensional Clusters*. Iremos explicar a seguir:

Iniciamos fazendo uma cópia do *dataframe* original (X) e selecionamos apenas as colunas de interesse:

Tirando uma cópia do arquivo para manipulação

```
[5]  X = df.copy(deep = True)
     X = X[['Country','Raised', 'ESG', 'E', 'S','G']]
     X
```

	Country	Raised	ESG	E	S	G
0	Lithuania	5986183.0	0.010	0.008	0.002	0.000
1	Netherlands	0.0	0.018	0.010	0.006	0.002
2	Austria	3500000.0	0.014	0.010	0.000	0.004
3	Hong Kong SAR, China	0.0	0.008	0.006	0.000	0.002
4	United States	0.0	0.017	0.000	0.011	0.006
...	...	...	...	...	...	...

Figura 9. Selecionando apenas as colunas de interesse

Verificamos por valores nulos, como não houve confirmação na saída, estamos bem:

Verificando nulos (nada na saída = ok)

```
for col, hasNull in X.isnull().any().items():
  if(hasNull):
    print(f'coluna{col}:  {hasNull}')
```

Figura 10. Verificando valores nulos

Utilizamos a função *get_dummies()* para converter as colunas não numéricas para valores 0 e 1. A função funciona da seguinte forma: uma coluna nova é gerada para cada valor não

numérico em cada coluna não numérica, de forma que caso o valor original do atributo equivale à coluna gerada, a célula é marcada como 1, caso contrário, é marcada como 0. No nosso caso, temos apenas "*Country*" como uma coluna não numérica, dessa forma, uma coluna para cada país na coluna "*Country*" será gerada:

Usando o método get_dummies do pandas para converter colunas não numéricas.

```
[8] #We use pandas's 'get_dummies()' method
    X = pd.get_dummies(X)
```

```
X.head()
```

	Raised	ESG	E	S	G	Country_Afghanistan	Country_Albania	Country_Argentina	Country_Armenia	Country_Australia	...
0	5986183.0	0.010	0.008	0.002	0.000	0	0	0	0	0	...
1	0.0	0.018	0.010	0.006	0.002	0	0	0	0	0	...
2	3500000.0	0.014	0.010	0.000	0.004	0	0	0	0	0	...
3	0.0	0.008	0.006	0.000	0.002	0	0	0	0	0	...
4	0.0	0.017	0.000	0.011	0.006	0	0	0	0	0	...

5 rows × 118 columns

Figura 11. Usando a função *get_dummies*

Após isso, o autor separa manualmente as colunas numéricas originais (guardadas em numer) do *dataframe* das colunas não numéricas convertidas (guardadas em cater). Aqui fizemos um pequeno script para fazer isso automaticamente:

No site, o autor separa manualmente. Aqui, fiz um pequeno código

```
numer = []
cater = []

for col in X.columns:
  if(X[col].dtype == np.int64 or X[col].dtype == np.float64):
    numer.append(col)
  else:
    cater.append(col)

print('numericas originais:',numer)
print('nao numericas convertidas:', cater)
```

```
numericas originais: ['Raised', 'ESG', 'E', 'S', 'G']
nao numericas convertidas: ['Country_Afghanistan', 'Country_Albania',
```

Figura 12. Separando as colunas

Com as colunas separadas, separamos os dados em dois *dataframes*:

```
# gerando um dataframe apenas com as colu
numer = X.loc[:, numer].copy(deep = True)
numer
```

	Raised	ESG	E	S	G
0	5986183.0	0.010	0.008	0.002	0.000
1	0.0	0.018	0.010	0.006	0.002
2	3500000.0	0.014	0.010	0.000	0.004
3	0.0	0.008	0.006	0.000	0.002
4	0.0	0.017	0.000	0.011	0.006
...	...	...	...	...	...

Figura 13. Separando em *dataframes*

```
# gerando um dataframe apenas com as colunas não numericas convertidas
cater = X.loc[:, cater].copy(deep = True)
cater
```

	Country_Afghanistan	Country_Albania	Country_Argentina	Country_Armenia	Country_Australia
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
...	...	...	...	...	...
6536	0	0	0	0	0
6537	0	0	0	0	0
6538	0	0	0	0	0
6539	0	0	0	0	0
6540	0	0	0	0	0

6541 rows × 113 columns

Figura 14. Separando em *dataframes*

Agora, com os *dataframes* separados, vamos fazer a escala para os dados numéricos originais (que estão no *dataframe* numer):

```
#Initialize our scaler
scaler = StandardScaler()

# Salvando os nomes originais das colunas para renomeação depois, assim como o autor fez
nomesColOriginais = numer.columns

# fazendo a escala
numer = pd.DataFrame(scaler.fit_transform(numer))

#renaming cols that has been scaled
numer.columns = [f"{nomesColOriginais[i]}_scaled" for i in range(numer.shape[1])]
numer
```

	Raised_scaled	ESG_scaled	E_scaled	S_scaled	G_scaled
0	0.086857	3.092590	4.880242	0.769739	-0.229437
1	-0.187503	5.809621	6.126549	2.782878	1.249132
2	-0.027090	4.451105	6.126549	-0.236831	2.727700
3	-0.187503	2.413333	3.633935	-0.236831	1.249132
4	-0.187503	5.469992	-0.104986	5.299302	4.206269
...	...	...	...	...	...

Figura 15. Fazendo a escala

Após fazer a escala, reunimos os dados novamente em um único *dataframe*:

```
# Unindo os dataframes numer e cater, mas agora com uma escala aplicada em numer
X = pd.concat([cater, numer], axis=1, join='inner')
X.head()
```

Figura 16. Reunindo os dados

Agora, iremos aplicar o método do *K-Means* para agrupar os dados em *clusters* (4 pois esse foi o número encontrado no *MiniSom*):

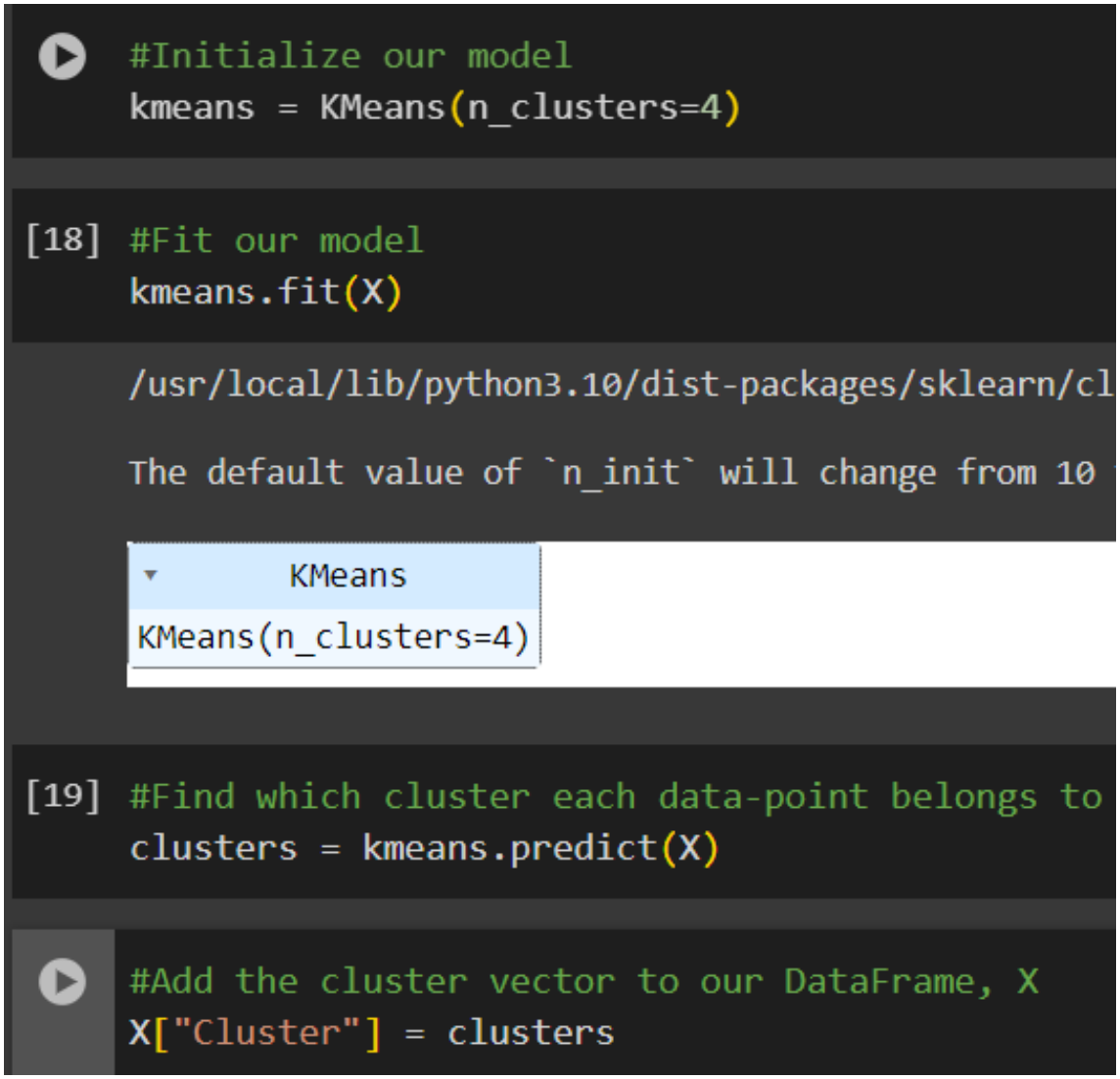


Figura 17. Utilizando o PCA

Agora, temos um *dataframe* com os *clusters gerados*, porém precisamos reduzir a dimensionalidade dos dados para conseguir gerar uma visualização, então, utilizamos o PCA (*Principal Component Analysis*) para isso:

```
#plotX = pd.DataFrame(np.array(X.sample(5000))) # original do tutorial
plotX = X

#Rename plotX's columns since it was briefly converted to an np.array above
plotX.columns = X.columns
```

initialize our PCA models

```
[22] #PCA with two principal components
pca_2d = PCA(n_components=2)
```

build our new DataFrames

```
[23] #This DataFrame contains the two principal components that will be used
#for the 2-D visualization mentioned above
PCs_2d = pd.DataFrame(pca_2d.fit_transform(plotX.drop(["Cluster"], axis=1)))
```

Rename the columns of these newly created DataFrames

```
[24] PCs_2d.columns = ["PC1_2d", "PC2_2d"]
```

We concatenate these newly created DataFrames to plotX so that they can be used by plotX as columns

```
[25] plotX = pd.concat([plotX,PCs_2d], axis=1, join='inner')
```

Figura 18. Gerando os *clusters*

Após isso, fazemos apenas alguns passos para mostrar o gráfico e gerar o resultado.

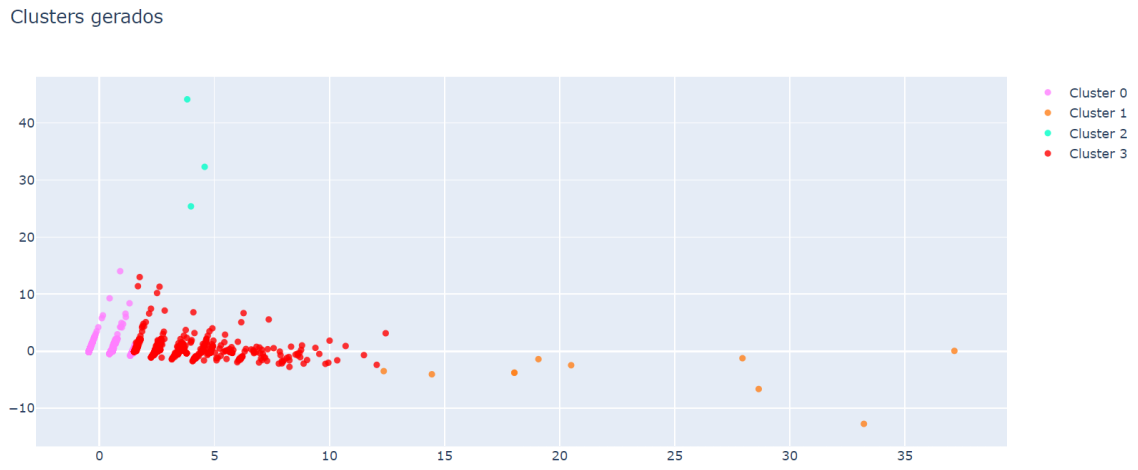


Figura 19. Gerando os *clusters*

4. Gráficos Explicativos e Possíveis Correlações Encontradas

Devido cada método ter agrupado os *clusters* de forma diferente, separamos a análise por método para facilitar o entendimento.

4.1. Explicação sobre o *MiniSom*

Para encontrar uma regra lógica, geramos gráficos como forma de visualizar as relações entre as variáveis de interesse. Vale notar que ao rodar novamente, os *clusters* não se agrupavam exatamente da mesma maneira, mesmo que o código não tivesse mudado.

4.1.1. Primeira análise

Primeiramente geramos gráficos para verificar quais e quantas *startups* de cada país estavam presentes nos *clusters*

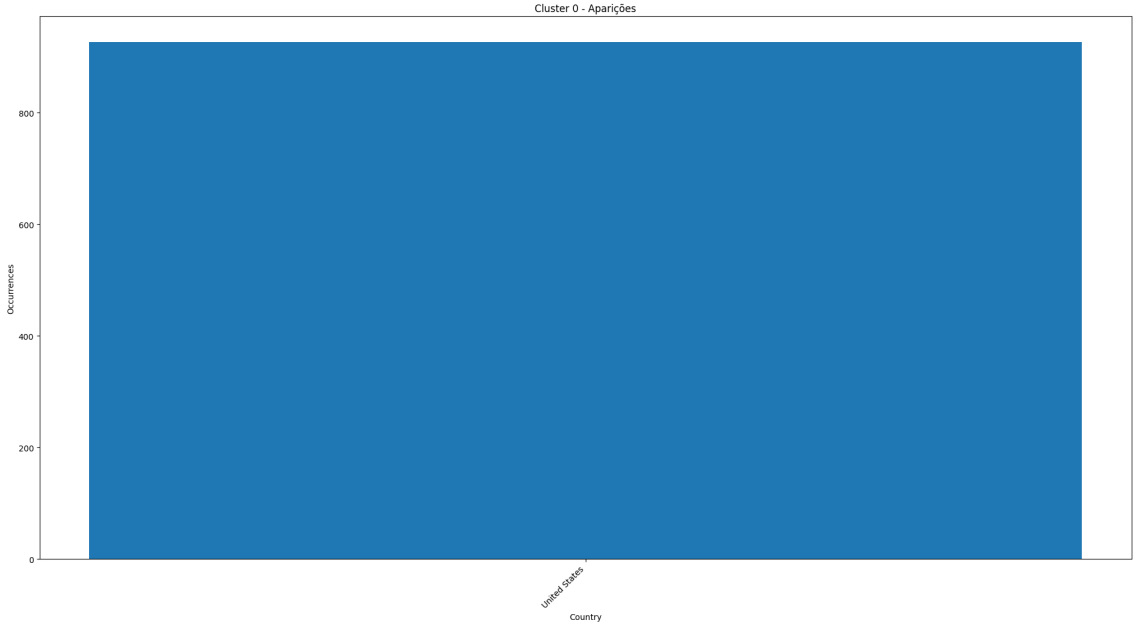


Figura 20. *Cluster* 0

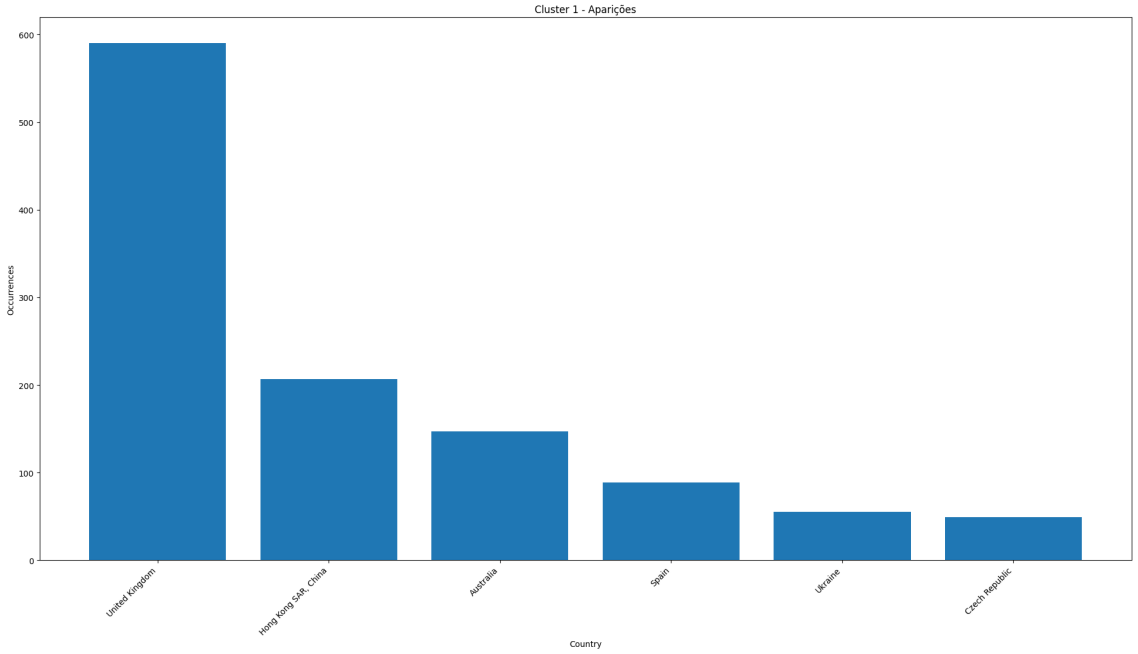


Figura 21. *Cluster* 1

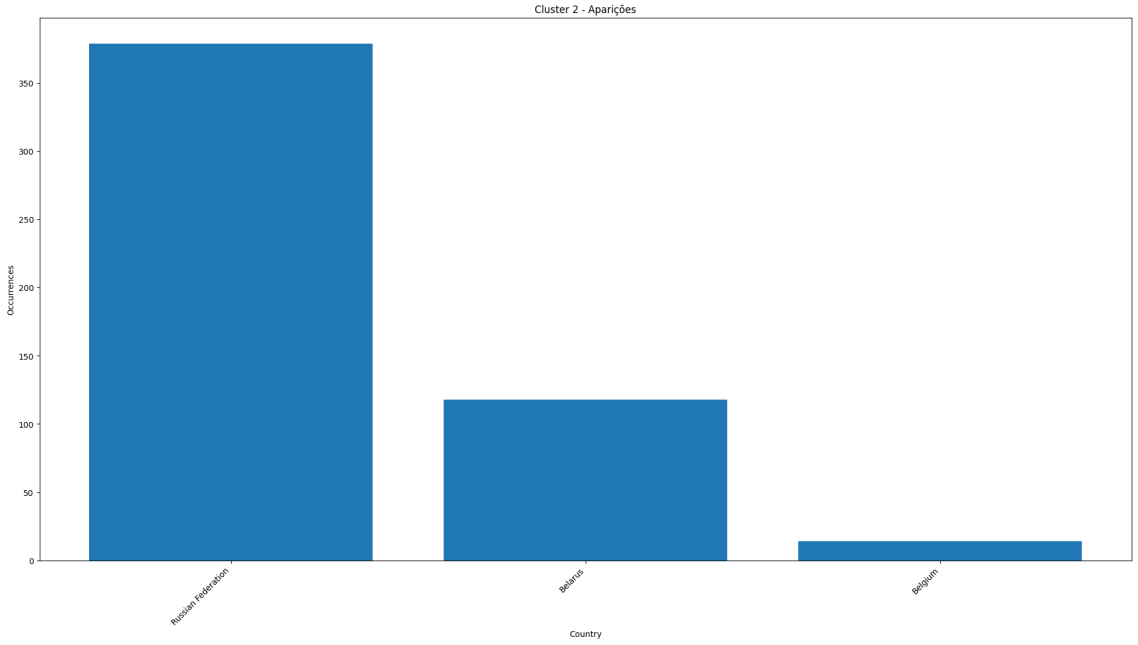


Figura 22. *Cluster* 2

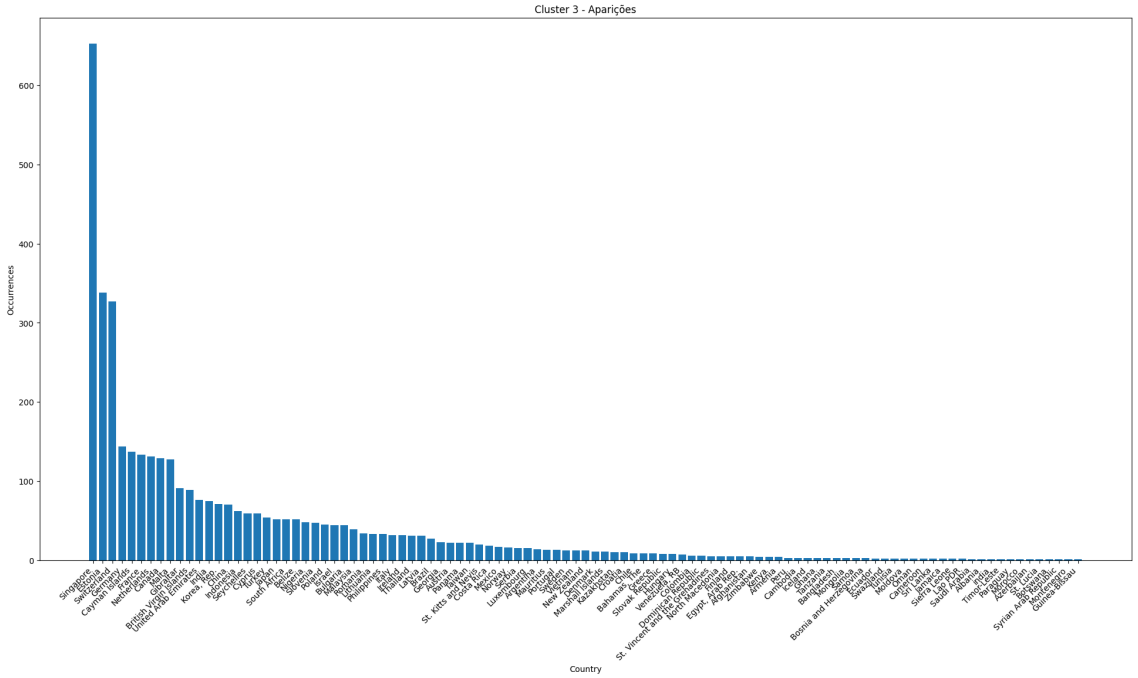


Figura 23. *Cluster* 3

Analisando os gráficos, pudemos observar que no *cluster* 0 estão localizadas as empresas dos Estados Unidos apenas. Porém, olhando para os outros gráficos, não vemos uma outra relação aparente. Ao analizar os valores dos montantes totais, percebemos o seguinte padrão.

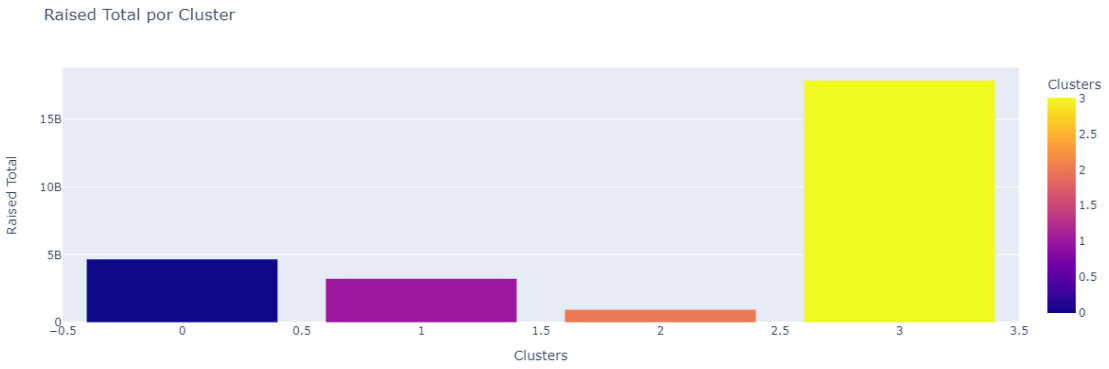


Figura 24. Relações entre *clusters* e *raised* total

O *cluster* 3 tem o maior "*raised*" total, seguido pelo *cluster* 0, depois o 1, e depois o 2. O que é interessante pois fizemos a seguinte análise:

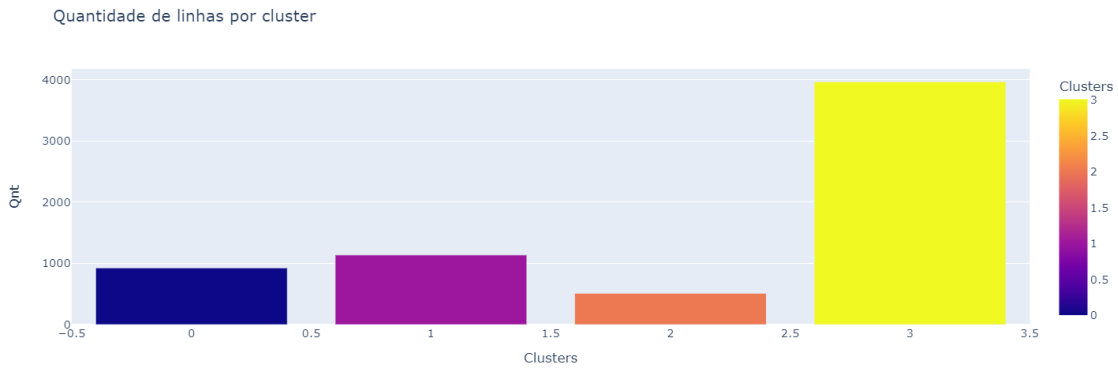


Figura 25. Quantidade de linhas por *cluster*

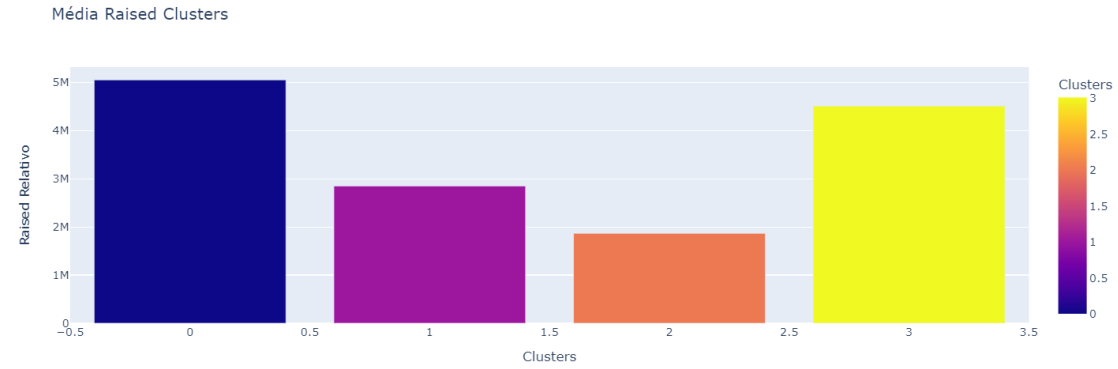


Figura 26. Media raised por *cluster*

Podemos ver que devido ao *cluster* 3 ter o maior número de linhas, o montante médio não é o maior. Vemos então que em média, o *cluster* 0, tem maior montante, seguido pelo *cluster* 3, depois o *cluster* 1, e por fim o *cluster* 2.

Vemos que, devido ao processo que o método *get_dummies* faz, as *features* de nome de país se tornaram as mais relevantes

4.1.2. Segunda análise

A segunda análise feita foi não normalizando a coluna *raised*. O primeiro ponto a se notar é que o erro de quantização sobe muito:

```
#data = X.values
data = XsemC.values
som_shape = (100, 100)
som = MiniSom(som_shape[0], som_shape[1], 118, sigma=1.5, learning_rate=.5,
              neighborhood_function='gaussian', random_seed=0)
#som.train_batch(data, 500, verbose=True)

som.pca_weights_init(data)
som.train(data, 10000, verbose=True)
```

[49] ✓ 2m 7.2s

```
[ 10000 / 10000 ] 100% - 0:00:00 left
quantization error: 236265.54293629268
```

Figura 27. Erro altíssimo ao rodar sem normalizar os dados

Ao partir para o gráfico de *clusters*, verificamos que 3 *clusters* se formaram.

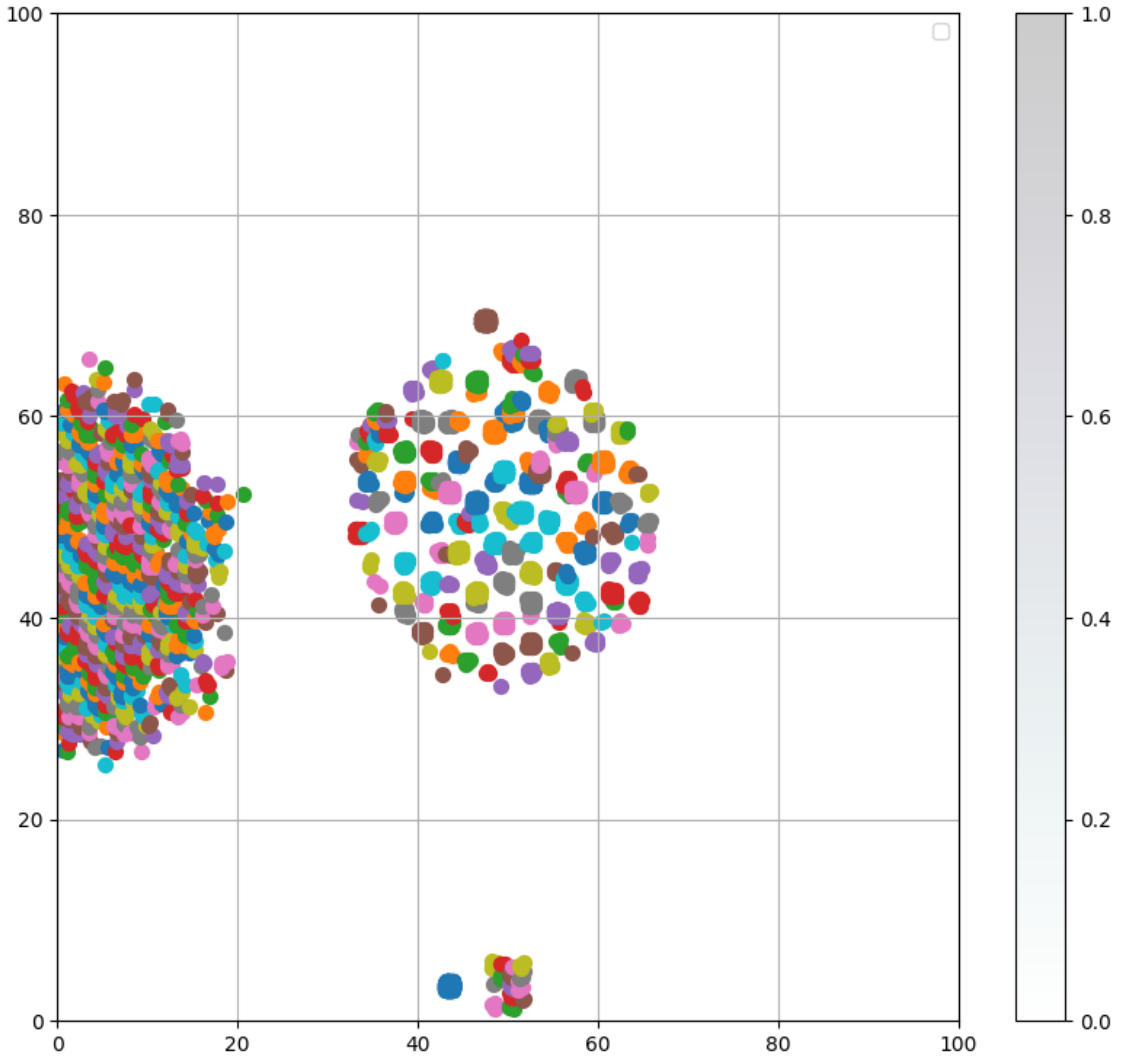


Figura 28. *Clusters* formados após segunda análise

E tivemos esses gráficos de análise sobre esses 3 *clusters*:

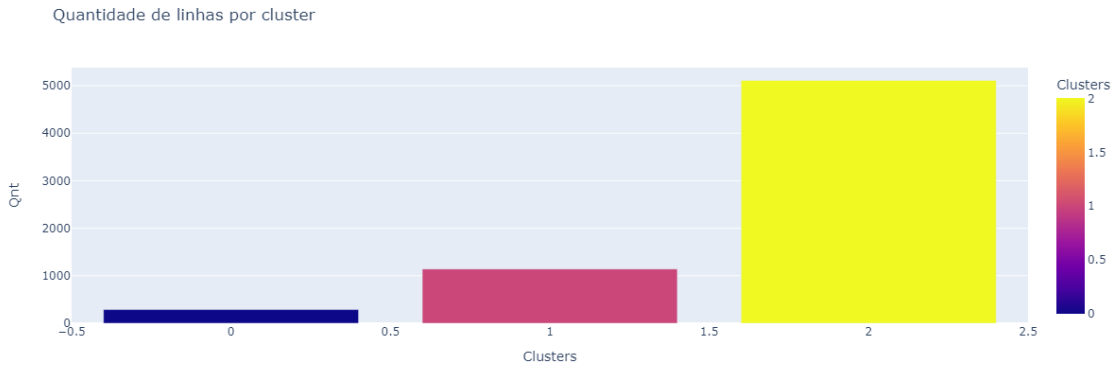


Figura 29. Quantidade de linhas por *cluster* 2ª análise

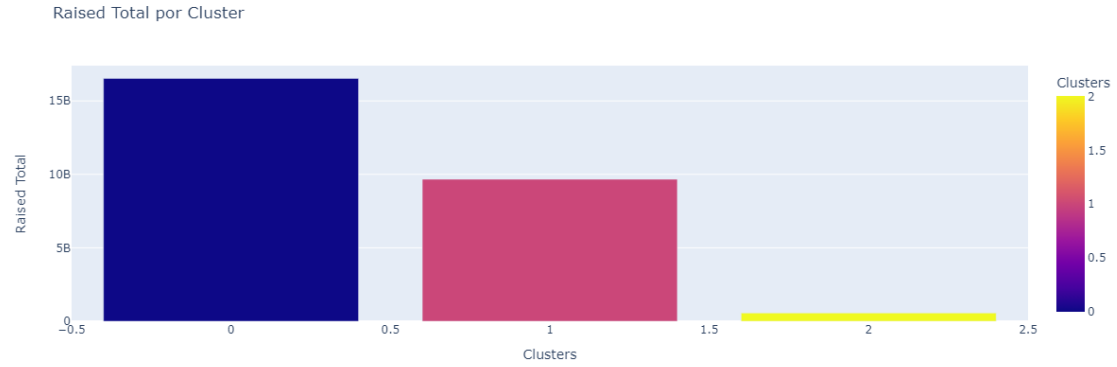


Figura 30. Montante total por *cluster* 2ª análise

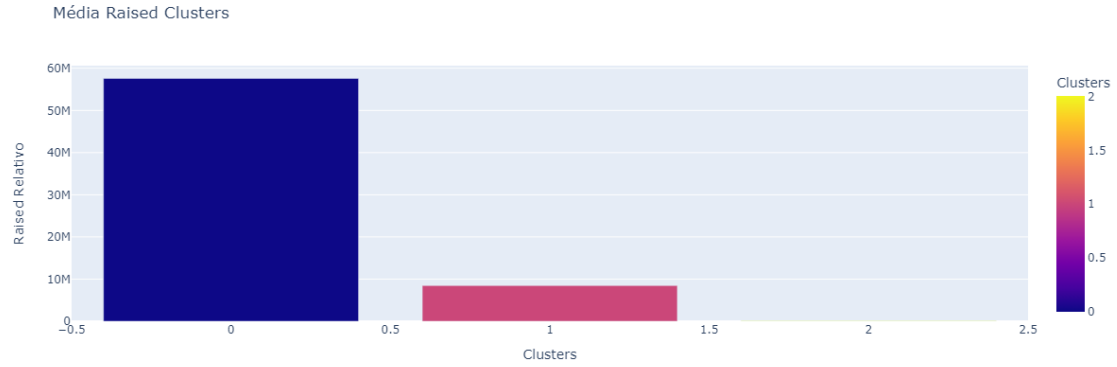


Figura 31. Montante médio por *cluster* 2ª análise

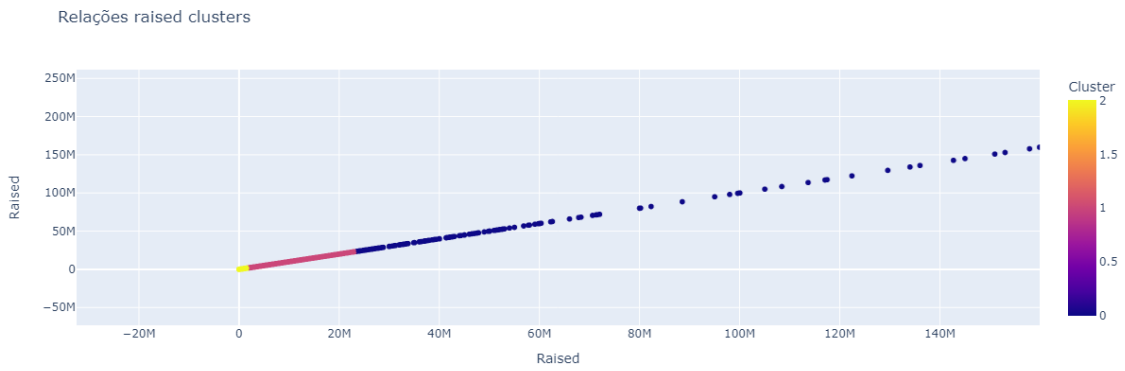


Figura 32. Conforme raised aumenta, *cluster* muda

Primeira análise que podemos fazer sobre os gráficos apresentados, na figura 30, vemos que o *cluster* 0 tem maior montante total, e maior montante médio, algo que não aconteceu na primeira análise (consultar figuras 24, e 26).

E conforme continuamos nossa análise, percebemos que esse agrupamente se deu com um *range* de *raised*, conforme visto na figura 32. Então temos que:

- O *cluster* **2** possui as linhas com *Raised* entre [0,1987132]
- O *cluster* **1** possui as linhas com *Raised* entre [2000000,23046800]
- O *cluster* **0** possui as linhas com *Raised* entre [23211600,1000000000]

4.2. Explicacão sobre o K-Means

Para o resultado do*K-Means*, geramos alguns gráficos e estatísticas para procurar uma regra lógica. OBS: a numeração dos *clusters* pode mudar de acordo com a execução do código, dessa forma, nos referimos aos *clusters* pela numeração encontrada na nossa execução.

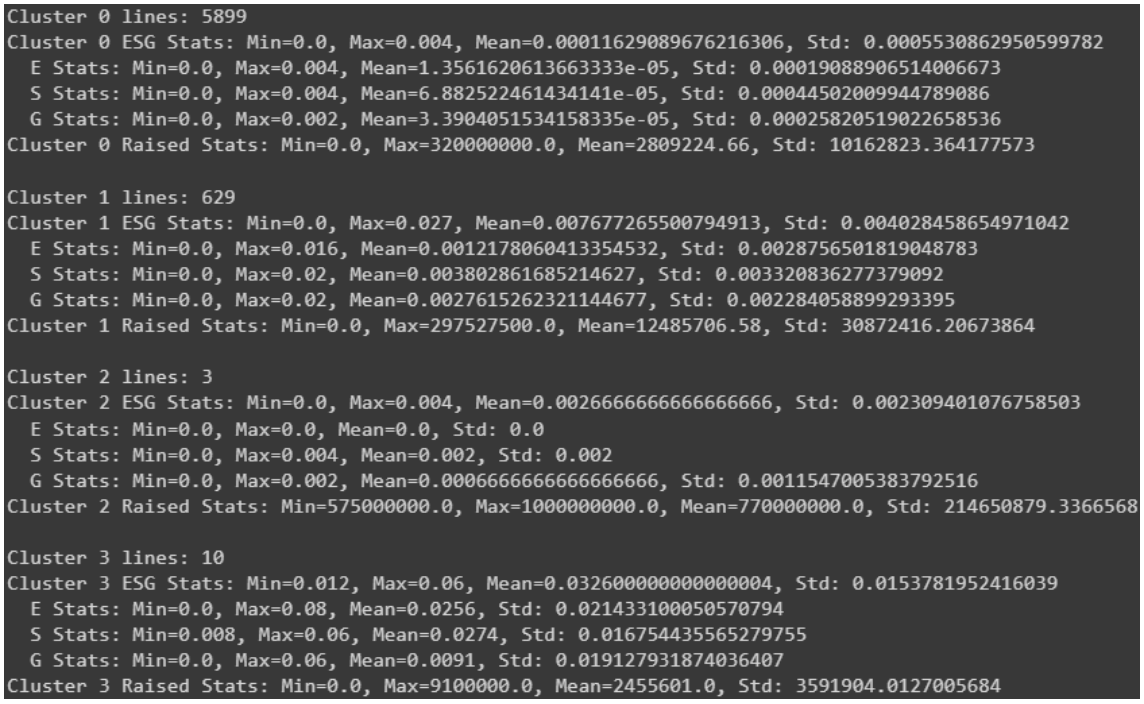


Figura 33. Estatísticas sobre o *Raised* e Países no K-Means

Na imagem acima podemos chegar nas seguintes conclusões:

- O *cluster 0* possui o menor ESG médio, e um *Raised* médio relativamente baixo;
- O *cluster 1* possui um número de statups com valores variados de ESG e *Raised*. Portanto, possui um ESG médio bem mediano e um valor *Raised* médiano também. Ou seja, o *cluster* 1 é bem mediano em geral;
- O *cluster* 2 possui valores de *Raised* fora da curva (muito mais altos que os outros *clusters*). Com relação ao ESG, o *cluster* possui o menor valor de ESG médio;
- O *cluster* 3 posui os menores valores de *Raised* em relação aos demais, ou seja, um *Raised* médio relativamente baixo. Com relação ao ESG, ele possui um ESG médio bem alto.

4.2.1. Com relação ao *k-means* (baseado na informação ESG)

Então, para confirmar nossa hipótese, geramos um gráfico de ESG x País. No gráfico abaixo é possível ter as seguintes conclusões:

- O *cluster* 0 possui as startups com os menores valores de ESG.
- O *cluster* 1 possui o maior número de *startups* com valores ESG na média;
- O *cluster* 2 possui o menor número de *startups* (ao ponto de quase não aparecer no gráfico, mas pelas estatísticas, acreditamos que esteja por trás dos valores do *cluster* 0)
- O *cluster* 3 possui as *startups* com os maiores valores de ESG;

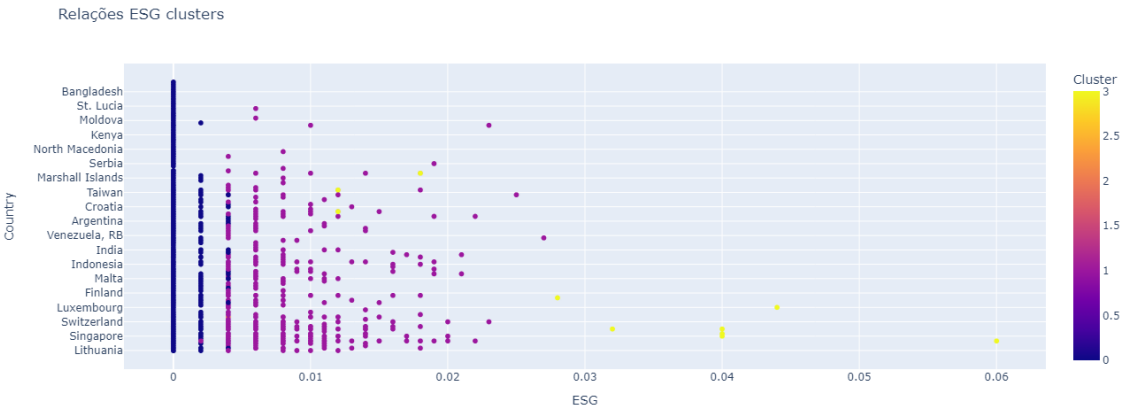


Figura 34. Estatísticas sobre o ESG e Países

4.2.2. Com relação ao *k-means* (baseado na informação *Raised*)

Fizemos também um gráfico de *Raised* x País, e observamos os seguintes pontos:

- O *cluster* 0 possui as *startups* com os menores valores de *Raised*.
- O *cluster* 1 possui valores nem tão baixos quanto o *cluster* 0, nem tão altos quanto os outros *clusters*, ou seja, estão na média;
- O *cluster* 2 possui as *startups* com os maiores *Raised* (fora da curva);
- O *cluster* 3 possui o menor número de *startups* (ao ponto de quase não aparecer no gráfico)

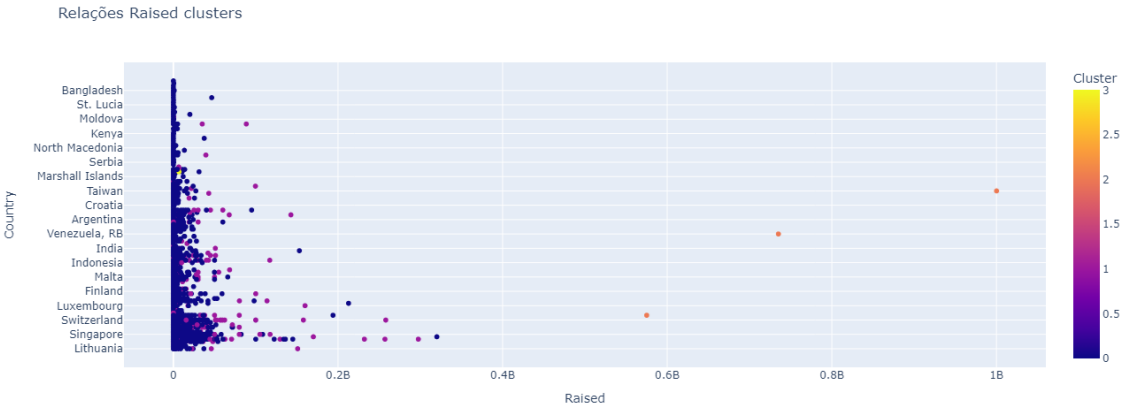


Figura 35. Estatísticas sobre o *Raised* e Países

OBS:Não identificamos a existência da influência de um país na escolha de um *cluster*.

5. Comparação entre *MiniSom* e *K-Means*

Portanto, ao utilizar o *Minisom*, obtivemos duas regras lógicas: uma baseada no nome do país, juntamente com o valor de *Raised*; e a outra baseada unicamente no valor de *Raised*, de forma que os agrupamentos foram feios baseado em intervalos.

Entretanto, ao utilizar o *K-Means*, chegamos a conclusão de que os *clusters* são formados com base nos valores ESG e *Raised*. Valores extremos de um dos campos tendem a dividir os elementos em um *cluster*, mas a combinação dos dois é que gera o resultado.

Chegamos a conclusão de que as regras lógicas dos *clusters* são:

- *cluster* 0 -> valores ESG e *Raised* muito baixos;
- *cluster* 1 -> valores ESG e *Raised* intermediários/baixos, mas principalmente levando em conta o ESG;
- *cluster* 2 -> principalmente o valor de *Raised* muito alto;
- *cluster* 3 -> Principalmente valor de ESG muito alto.