Aprendizado de Máquina
Aprendizado Supervisionado e Não Supervisionado
Validação de Agrupamento

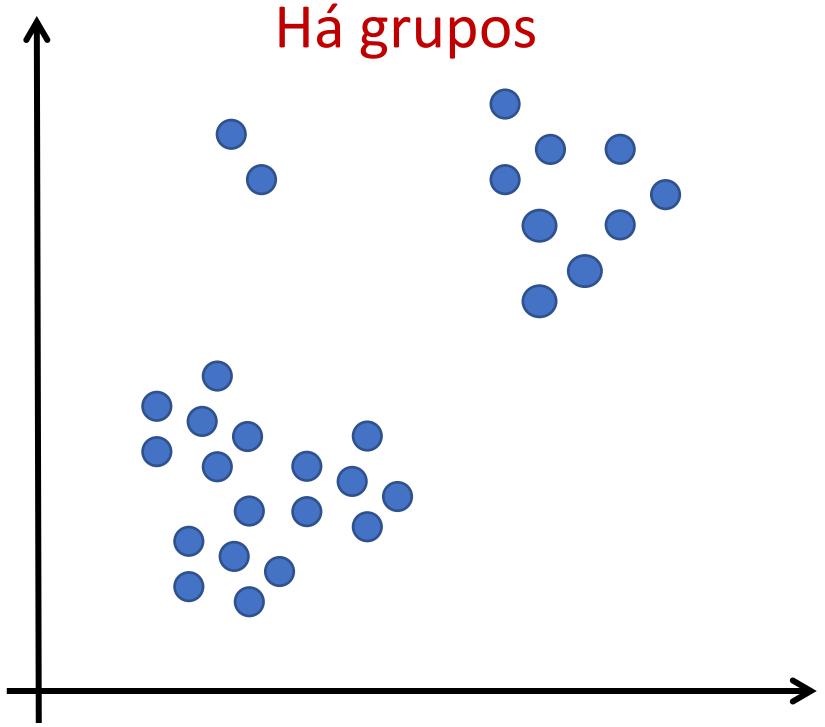
Inteligência Artificial – 2020/1

Validação de agrupamento

- A maioria dos algoritmos de agrupamento impõem uma estrutura de agrupamento ao conjunto de dados X .

Não há grupos

Há grupos



Validação de agrupamento

- A maioria dos algoritmos de agrupamento impõem uma estrutura de agrupamento ao conjunto de dados X .

- Entretanto, X pode não possuir uma estrutura de agrupamento.

  - Assim torna-se necessário fazer a avaliação dos resultados obtidos pelo agrupamento.

  - Validação de Clusters: tarefa que avalia quantitativamente os resultados de um algoritmo de agrupamento para verificar se os clusters são significativos

Abordagens para validação de agrupamento

- Critérios externos
  - Usa uma medida externa que mede o grau de correspondência entre um agrupamento C produzido por um algoritmo com uma partição $\mathcal{P}$ construída independentemente de C
  - Classes de objetos são conhecidas

- Critérios internos
  - Usa uma medida interna que avalia o agrupamento C produzido por um algoritmo com base nos dados e na matriz de proximidade
  - Geralmente avaliam se os grupos são compactos e bem separados

Abordagens para validação de agrupamento

- Critérios relativos:

- O agrupamento é avaliado por comparação com outras estruturas de agrupamento, resultantes da aplicação:

  - do mesmo algoritmo de agrupamento com diferentes parâmetros ou

  - de outros algoritmos de agrupamento

Índices de validação

- Um índice de validação é uma estatística pela qual a validade de um agrupamento é testada

- Índices podem ser:

- Externos – avaliam o agrupamento comparando a partição resultante de um algoritmo com uma partição já conhecida

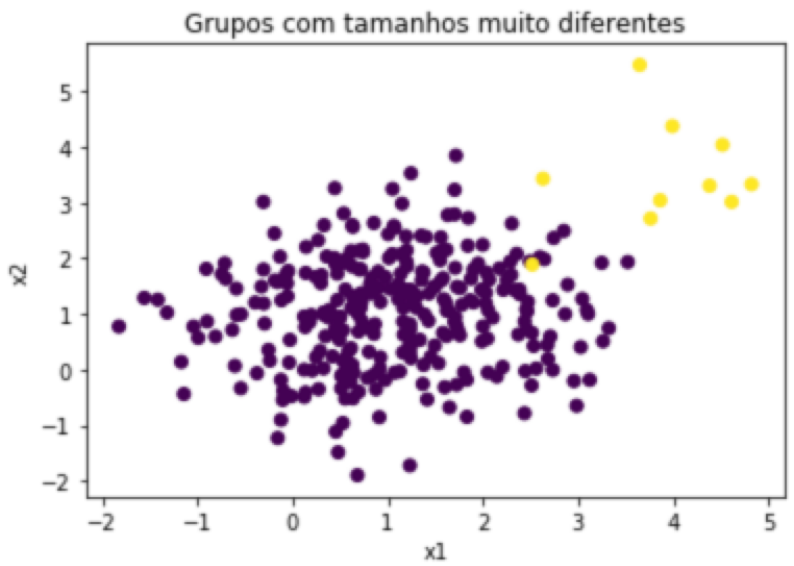
- Internos – avaliam o agrupamento com base apenas na matriz de dados ou na matriz de similaridade

- OBS- A validação relativa pode utilizar os dois tipos de índices

Índices de validação externos

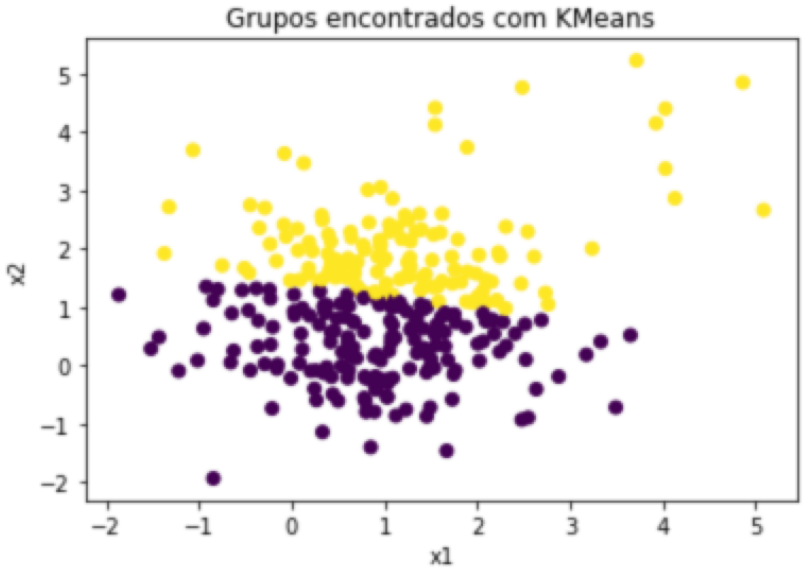
- Agrupamento obtido C = {C_1, C_2, ..., C_m}
- Agrupamento conhecido P = {P_1, P_2, ..., P_s},
  - O número de grupos em C não precisa ser igual ao número de grupos em P

Agrupamento conhecido P



?

Agrupamento obtido C



Índice Rand – validação externa

Mede a similaridade entre dois agrupamentos

$$R = \frac{a+d}{M}$$

- a: o número de pares de objetos de X em que os dois objetos pertencem ao mesmo grupo em C e em P
- d: o número de pares de objetos de X em que os dois objetos pertencem a grupos diferentes em C e em P

- M = todos os pares de objetos

- Mede a fração do número total de pares considerados “acertos”
- Valores entre 0 e 1
- Para atingir o valor máximo, é necessário que as partições tenham o mesmo número de grupos.

Índice Rand corrigido – validação externa

Ajusted Rand Index (ARI)

Ajustado para garantir um valor próximo de zero para agrupamentos aleatórios independente do número de clusters e instâncias e valor 1 para agrupamentos idênticos .

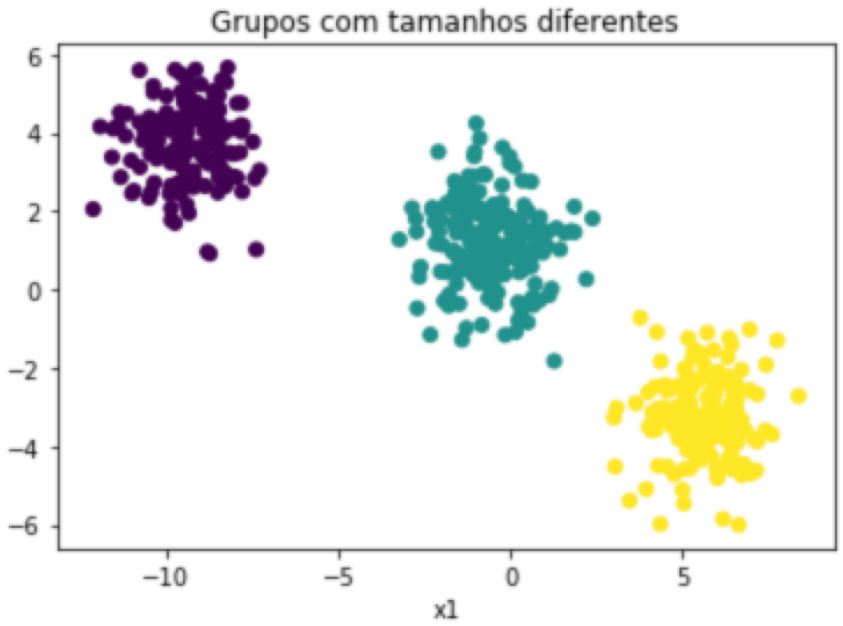
$$RC = \frac{R - E[R]}{\max(R) - E[R]}$$

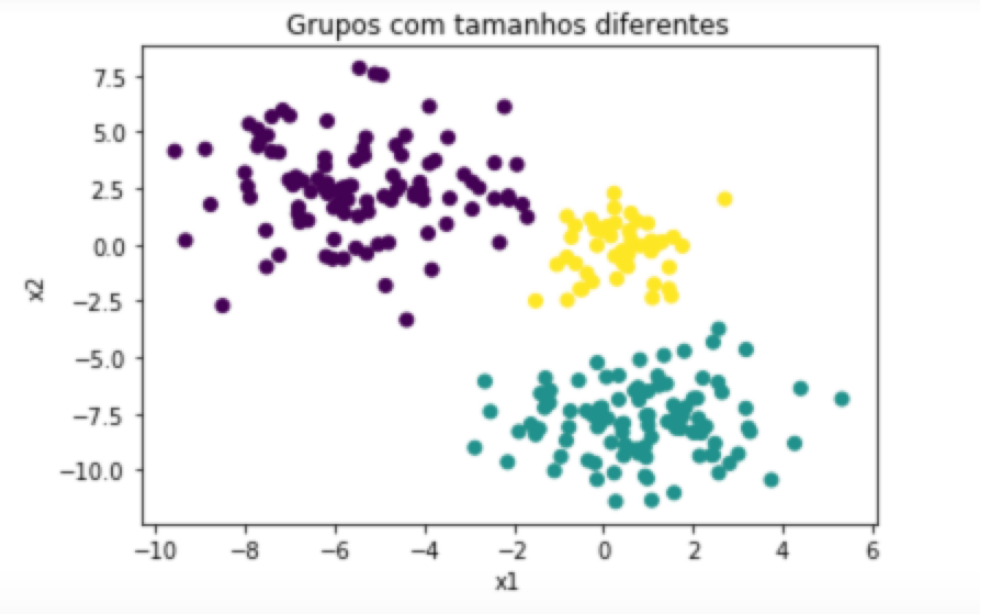
- Vantagens:
  - Agrupamentos aleatórios (grupos não válidos) tem um valor perto de zero independente do número de clusters ou de instâncias
  - Valores entre -1 e 1

Índices de validação internos

- A própria função de erro quadrático, otimizada no processo do Kmeans, pode ser usada como uma métrica de validação do agrupamento

$$err2(E,C) = \sum_{j=1}^{k}\sum_{i=1}^{n_j}\left\|x_i^{(j)} - c_j\right\|^2$$





Execução e validação de agrupamento em Python

K-Means em Python

- Importar a biblioteca e o objeto Kmeans

```
from sklearn.cluster import Kmeans
```

- Parâmetros
  - **n_clusters** número de clusters que o algoritmo vai gerar (padrão: 8)
  - **Init** modo como o algoritmo será inicializado:
    - ***k-means++***: método padrão, favorece a convergência.
    - ***random***: Inicializa os centróides de forma aleatória
    - ***ndarray***: Especifica um array de valores indicando qual seriam os centróides a serem usados para a inicialização.
  - **max_iter**: número máximo de iterações (padrão: 300)
  - **algorithm:** especifica a versão do algoritmo K-Means a ser utilizada. A versão clássica é executada através do valor **full**.

K-Means em Python

- Inicializar o K-means utilizando 3 clusters e método de inicialização random.

```
kmeans = KMeans(n_clusters = 3, init = 'random')
```

- Executar o método **fit()** para executar o algoritmo e agrupar os dados. O método fit() recebe como parâmetro os dados a serem agrupados

```
kmeans.fit(X)
```

- Exibir os centroides gerados pelo atributo **cluster_centers_**.

```
kmeans.cluster_ centers_
```

K-Means em Python

- Tabela de distâncias **fit_transform()**:
  - executa o K-means para agrupar os dados e retorna uma tabela de distâncias.

```
distance = kmeans.fit_transform(X)
distance
```

- Atributo **labels_** : retorna os labels para cada instância, ou seja, o número do cluster a que a instância de dados foi atribuída.

```
labels = kmeans.labels_ labels
```

Índice Rand corrigido em Python

- sklearn.metrics.**adjusted_rand_score**(*labels_true*, *labels_pred*)

- Parâmetros:
-
  - labels_true : int array, formato = [n_samples]
  - Rótulos dos grupos conhecidos usados como referência

  - labels_pred : array, formato = [n_samples]
  - Rótulos dos clusters obtidos no agrupamento

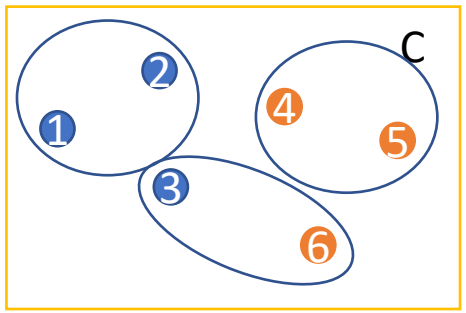
- Retorna:
  - ari : float
  - Índice de similaridade entre -1.0 and 1.0.

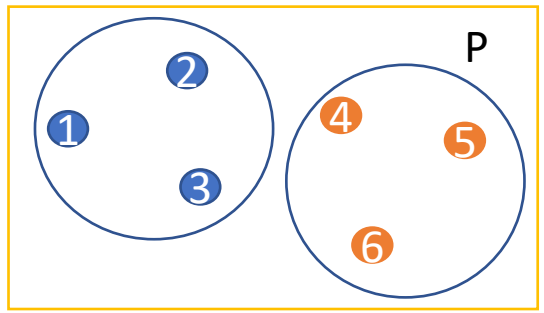
Índice Rand corrigido em Python

```
#Calcular o RC para o conjunto definido
```

```
from sklearn import metrics
labels_true = [0, 0, 0, 1, 1, 1]
labels_pred = [0, 0, 1, 1, 2, 2]
metrics.adjusted_rand_score(labels_true, labels_pred)
```

```
0.2424242424242246
```

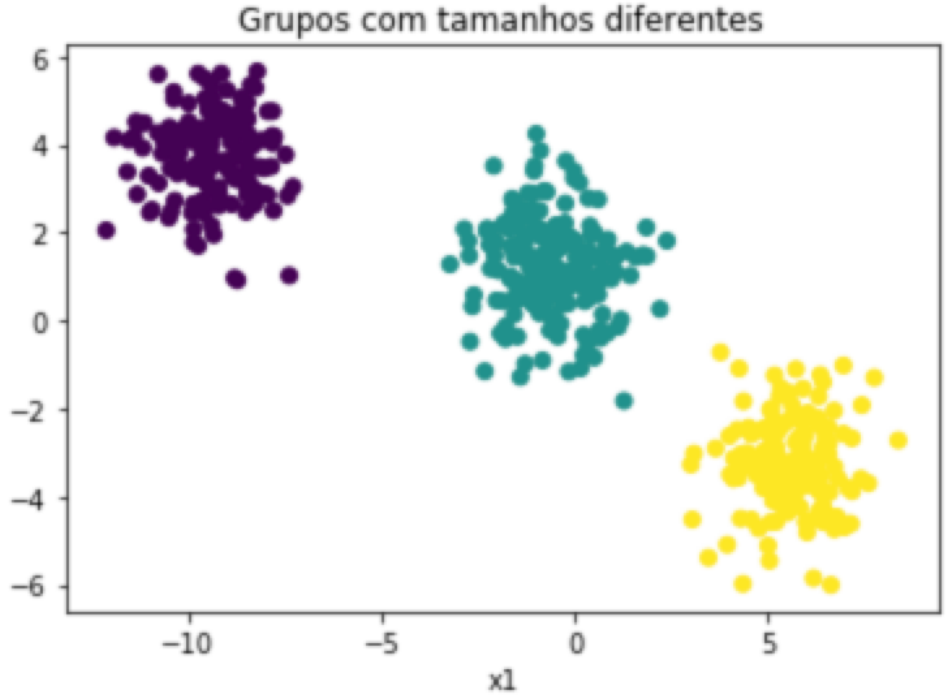




Índice Rand corrigido em Python

```
#Calcular o RC para o conjunto definido- gerado com make_blobs
```

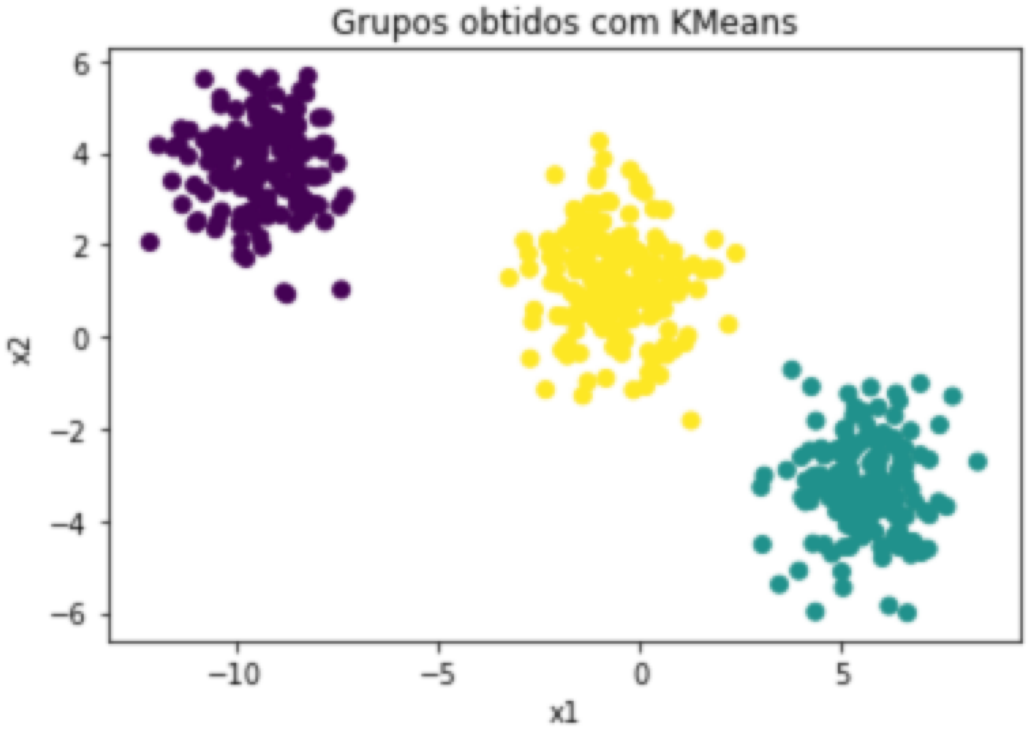
```
#gerando grupos com tamanhos diferentes e definindo o desvio padrão
X, y = make_blobs(n_samples=[150,200,150])
plt.scatter(X[:,0], X[:,1], c=y)
plt.title("Grupos com tamanhos diferentes")
plt.xlabel("x1")
plt.ylabel("x2")
```



Índice Rand corrigido em Python

```
#Agrupar com Kmeans
```

```
km = KMeans(n_clusters = 3)
km.fit(X)
plt.scatter(X[:,0], X[:,1], c=km.labels_)
plt.title("Grupos obtidos com KMeans")
plt.xlabel("x1")
plt.ylabel("x2")
```



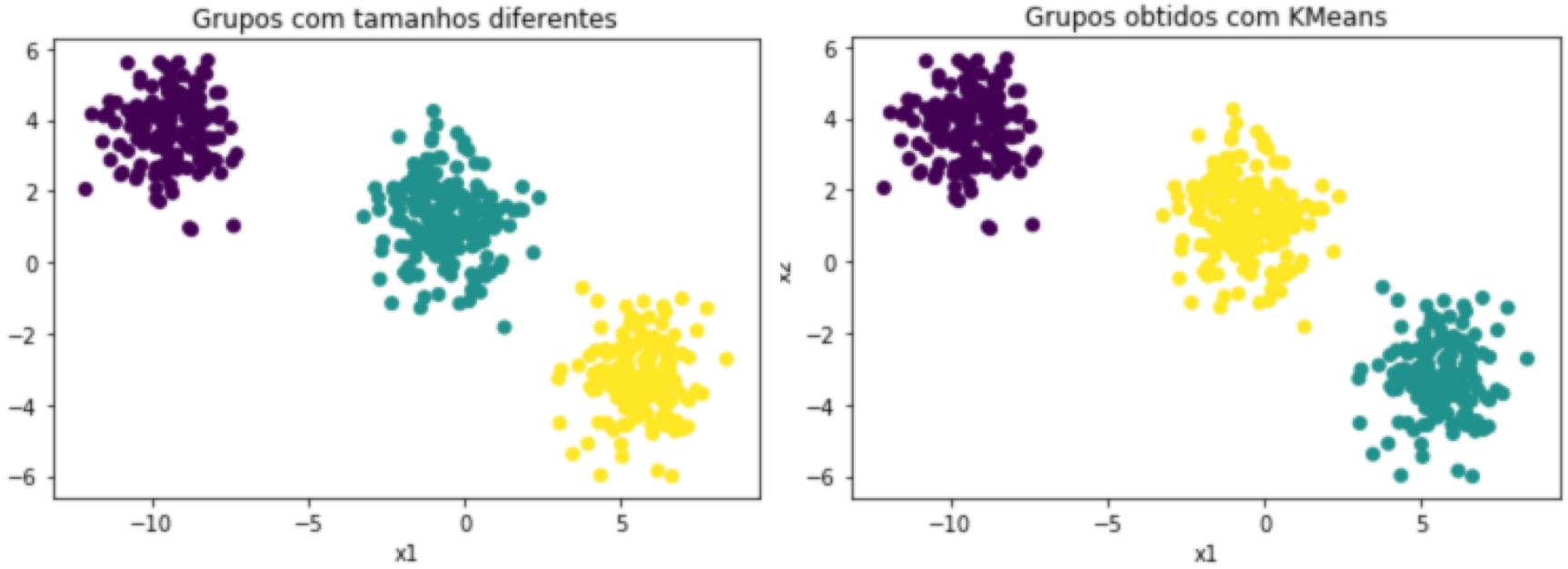
Índice Rand corrigido em Python

```
#Calcular RC
```

```
metrics.adjusted_rand_score(km.labels_,y)
```

```
1.0
```

Neste exemplo, os agrupamentos são iguais



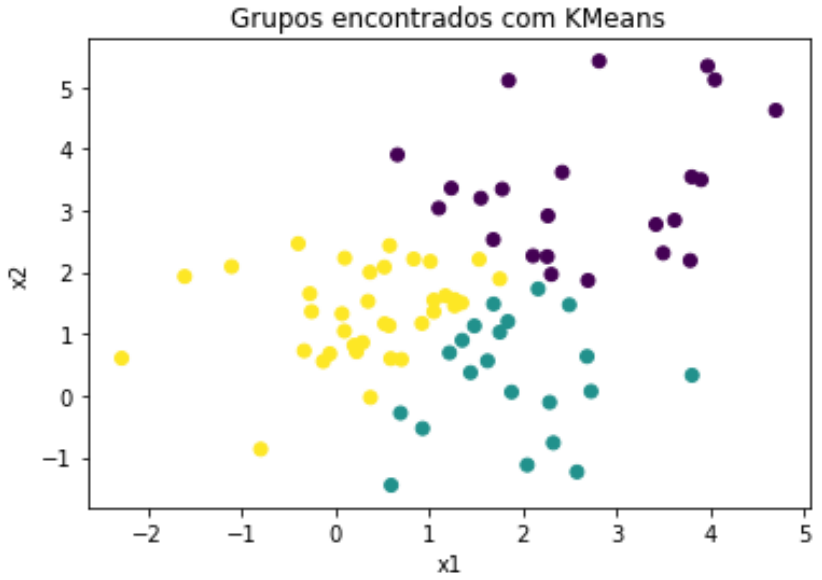
Índices de validação interno em Python

- A própria função de erro quadrático, otimizada no processo do Kmeans, pode ser usada como uma métrica de validação do agrupamento

- O algoritmo Kmeans em Python tem um atributo:
  - **inertia_** *:* float
  - Soma do quadrado das distâncias das instâncias para o centro de cluster mais próximo

```
# Executando KMeans
y_pred = Kmeans(n_clusters=3,init='random')
y_pred.fit(X)
y_pred.inertia_
```

```
127.4038470604296
```



Encontrar o melhor número de grupos para o algoritmo KMeans

- Método do “cotovelo”

  - Definir um intervalo de valores para o número de grupos [k_min, k_max]
  - Escolher um índice de validação (soma do quadrado das distâncias, silhueta, davis-bouldin,...)
  - Executar o algoritmo Kmeans para cada um dos valores de k (número de grupos)
  - Criar um gráfico com as coordenadas número de grupos “versus” índice
  - Encontrar o ponto do gráfico em que a melhora no valor do índice não é significativa (cotovelo)

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Importações
```

```
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.metrics as sm
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
```

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Usar a função read_csv da biblioteca pandas para ler o conjunto de dados
#Mostrar as primeiras cinco linhas com a função head()
```

```
data = pd.read_csv('Wholesale customers data.csv')
data.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Definir a lista de atributos categóricos e atributos contínuos
#Criar as estatísticas com a função describe
```

```
categorical_features = ['Channel', 'Region']
continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']
data[continuous_features].describe()
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Transformar atributos categóricos em binários
```

```
for col in categorical_features:
    dummies = pd.get_dummies(data[col], prefix=col)
    data = pd.concat([data, dummies], axis=1)
    data.drop(col, axis=1, inplace=True)
data.head()
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	Channel_1	Channel_2	Region_1	Region_2	Region_3
0	12669	9656	7561	214	2674	1338	0	1	0	0	1
1	7057	9810	9568	1762	3293	1776	0	1	0	0	1
2	6353	8808	7684	2405	3516	7844	0	1	0	0	1
3	13265	1196	4221	6404	507	1788	1	0	0	0	1
4	22615	5410	7198	3915	1777	5185	0	1	0	0	1

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Normalizar os atributos contínuos
```

```
#escalar os atributos contínuos
mms = MinMaxScaler()
mms.fit(data)
data_transformed = mms.transform(data)
```

```
# Fazer o agrupamento para cada quantidade de grupo no intervalo definido
Sum_of_squared_distances = []
K = range(1,15)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(data_transformed)
    Sum_of_squared_distances.append(km.inertia_)
```

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Construir o gráfico número de clusters X índice
```

```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Soma dos quadrados das distâncias')
plt.title('Método do Cotovelo para encontrar melhor valor de k')
plt.show()
```