06

Mãos à obra!

Vamos tentar fazer um agrupamento um pouquinho diferente do primeiro. Neste caso, vamos criar um modelo `K-Means` e falar para ele agrupar os dados em 20 grupos (um para cada gênero). Com o modelo criado, vamos treiná-lo passando os `generos_escalados` :

```
modelo = KMeans(n_clusters=20)

modelo.fit(generos_escalados)
```

Como antes, vamos criar um data frame a partir do centróides do grupo:

```
grupos = pd.DataFrame(modelo.cluster_centers_,
            columns=generos.columns)
```

E plotar um gráfico para cada centróide. Dessa vez, vamos rotacionar (`rot`) os rótulos do eixo `x` para ficar mas legível:

```
grupos.transpose().plot.bar(subplots=True,
                figsize=(25, 50),
                sharex=False,
                rot=0)
```

Como antes, vamos pegar algum grupo e fazer uma filtragem no data frame de filmes para ver como foram agrupados:

```
grupo = 2

filtro = modelo.labels_ == grupo

dados_dos_filmes[filtro].sample(10)
```

Aparentemente, os dados foram agrupados de uma maneira que faz sentido. Mas então, quantos grupos devemos usar?

Vamos criar a função que recebe o número de clusters e os dados e retorna o número de clusters e o erro (`inertia_`) daquele modelo:

```
def kmeans(numero_de_clusters, generos):
  modelo = KMeans(n_clusters=numero_de_clusters)
  modelo.fit(generos)
  return [numero_de_clusters, modelo.inertia_]
```

Vamos rodar essa função começando a agrupar em um único grupo e ir agrupando até 40 grupos. Para isso, vamos usar uma compressão de lista do Python:

```
resultado = [kmeans(numero_de_grupos, generos_escalados) for numero_de_grupos in range(1, 41)]
```

Lembrando que podemos ver essa lista colocando a variável como último conteúdo da célula.

Para facilitar o trabalho, vamos transformar essa variável em um data frame para facilitar sua manipulação:

```
resultado = pd.DataFrame(resultado,
            columns=['grupos', 'inertia'])
```

Agora, basta plotarmos um gráfico da coluna `inertia` do data frame. Lembrando que devemos passar a coluna `grupos` como parâmetro dos rótulos do eixo `x`:

```
resultado.inertia.plot(xticks=resultado.grupos)
```

Podemos ver que o ponto de quebra no gráfico foi próximo ao número `17`, logo, esse é o número de clusters que otimiza nosso modelo. Podemos rodar um novo modelo com 17 grupos e mostrar seus centróides:

```
modelo = KMeans(n_clusters=17)
modelo.fit(generos_escalados)

grupos = pd.DataFrame(modelo.cluster_centers_,
            columns=generos.columns)

grupos.transpose().plot.bar(subplots=True,
               figsize=(25, 50),
               sharex=False,
               rot=0)
```

Podemos também realizar um filtro por algum grupo para ver se fazem sentido:

```
grupo = 16

filtro = modelo.labels_ == grupo

dados_dos_filmes[filtro].sample(10)
```

Veremos que o agrupamento parece fazer muito sentido com o que esperamos.