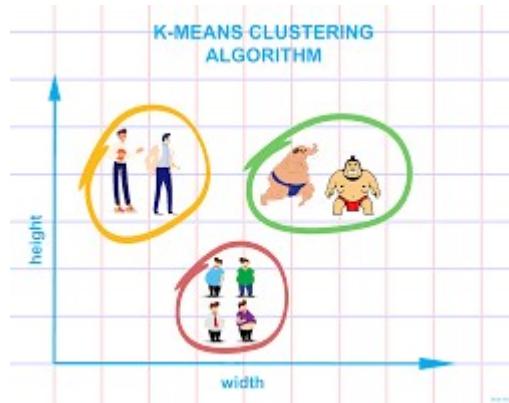


K-means Clustering



Contenidos

1. [Introducción](#)
 - 1.1 [Qué es K-means](#)
 - 1.2 [Aplicaciones](#)

1. TEORÍA

1. [¿Cómo funciona?](#)
 - 3.1 [Paso 1 : Elección del valor de K](#)
 - 3.2 [El método del codo \(Elbow Method\)](#)

2. Importar librerías

- 4.1 [Numpy](#)
- 4.2 [Pandas](#)
- 4.3 [Matplotlib](#)
- 4.4 [Seaborn](#)

1. Datos

- 5.1 [Análisis de datos exploratorios](#)
- 5.2 [Variable\(s\) categórica en numeros](#)

1. Normalización estadística

1. método del codo

1. K-Means con diferentes clústeres

[1. Plot](#)[Go to Top](#)

1.- Introducción



1.1- Que es K-means



- El agrupamiento "K-means" es un método de cuantificación de vectores, originalmente a partir del procesamiento de señales, que tiene como objetivo dividir n observaciones en k grupos en los que cada observación pertenece al grupo con la media más cercana (centros de grupo o centroide de grupo), sirviendo como un prototipo de el racimo. Esto da como resultado una partición del espacio de datos en celdas de Voronoi. El agrupamiento de k-medias minimiza las varianzas dentro del grupo (distancias euclidianas cuadradas), pero no distancias euclidianas regulares, que sería el problema de Weber más difícil: la media optimiza los errores cuadrados, mientras que solo la mediana geométrica minimiza las distancias euclidianas. Por ejemplo, se pueden encontrar mejores soluciones euclidianas usando K-mean y k-medoids. El algoritmo K-Means funciona de forma iterativa para asignar cada punto de datos a uno de los K grupos según las características que se proporcionan. Los puntos de datos se agrupan según la similitud de características.

Para saber mas ver: [K-means](#)

1.2.- Aplicaciones



K-means es bastante fácil de aplicar incluso a grandes conjuntos de datos. Se ha utilizado con éxito en la segmentación de mercado, visión por computadora y astronomía, entre muchos otros. A menudo se utiliza como paso de "preprocesamiento" para otros algoritmos, por ejemplo, para encontrar una configuración inicial. Algunos usos son por ejemplo:

- Cuantización vectorial
- Análisis de conglomerados
- Aprendizaje de funciones

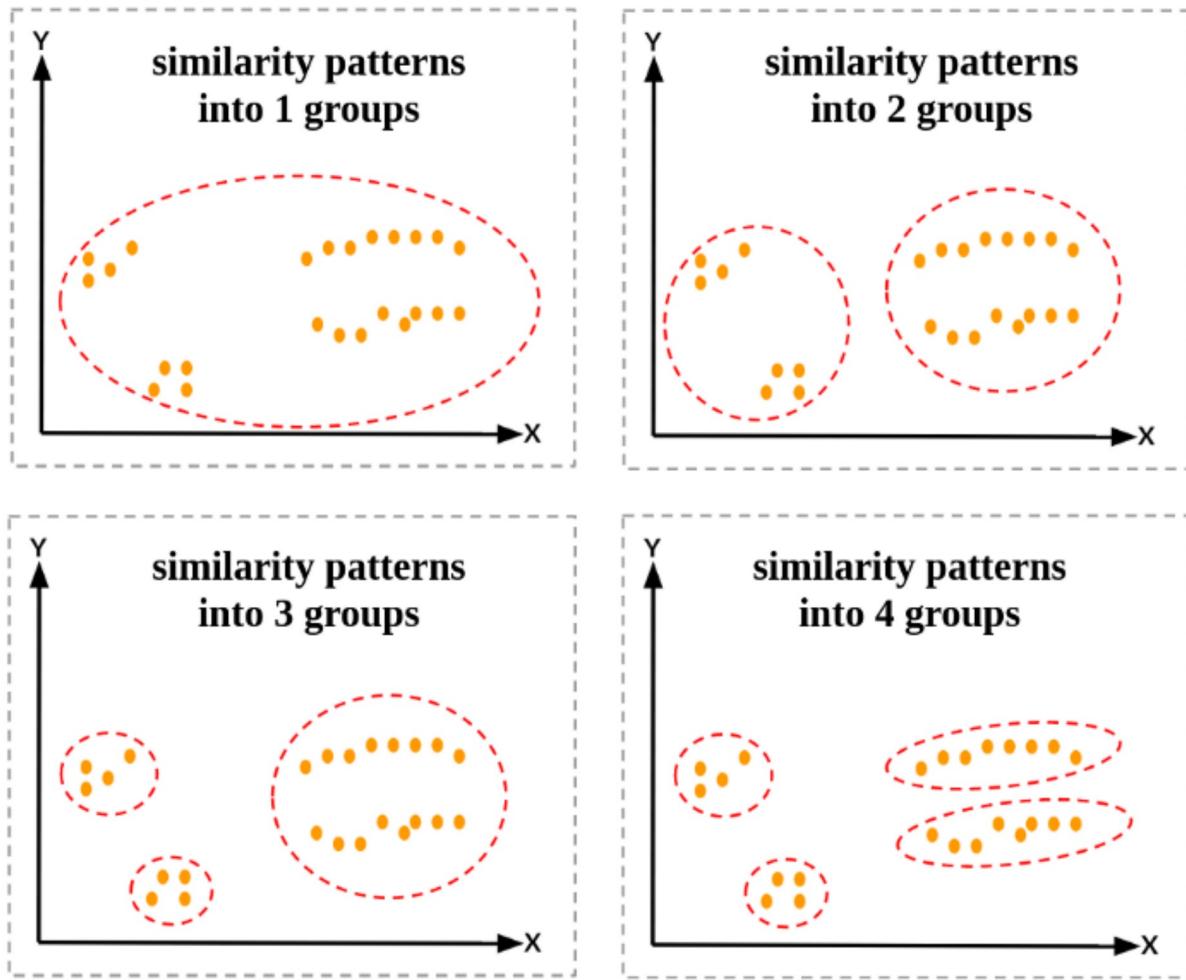
Para saber mas ver: [K-means Aplicaciones](#)

2.- TEORÍA

[Go to Top](#)

La agrupación en clústeres de K-means es una solución fácil para partitionar un conjunto(s) de datos en K grupos distintos. Para realizar un K-means, primero debemos especificar el número de grupos K, entonces el algoritmo asignará a cada dato a una K agrupación.

La agrupación de K-medias se puede representar en forma de diagrama de la siguiente manera:



K-means agrupa los datos tratando de separar muestras en N grupos de igual varianza, minimizando un criterio conocido como inercia o suma de cuadrados dentro del grupo. El algoritmo de K-means tiene como objetivo elegir un centroide que minimice la inercia, o el criterio de suma de cuadrados dentro del clúster.

$$\sum_{i=0}^n \min_{\mu_j \in C} \|x_i - \mu_j\|^2$$

[Go to Top](#)

3.- ¿Como funciona? 🤔

3.1.- Paso 1 : Elección del valor de K 💡

K-Means depende de encontrar un número de grupos y etiquetas de datos para un valor predefinido de K. Para encontrar el número de grupos en los datos, necesitamos ejecutar el algoritmo de agrupamiento de K-Means para diferentes valores de K y comparar los resultados. Por tanto, el rendimiento del algoritmo K-Means depende del valor de K. Debemos elegir el valor

óptimo de K que nos dé el mejor rendimiento. Hay diferentes técnicas disponibles para encontrar el valor óptimo de K. La técnica más común es el método del codo.

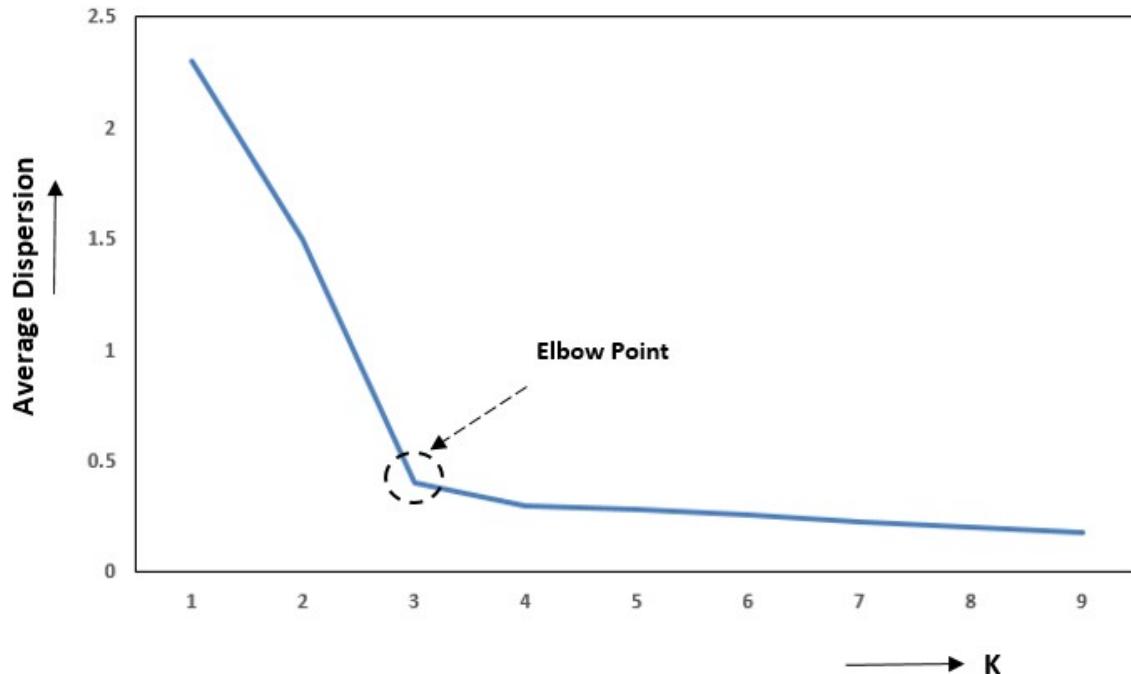
3.2.- El método del codo (Elbow Method)

Este método utiliza los valores de la inercia obtenidos tras aplicar el K-means a diferente número de Clusters (desde 1 a N Clusters), siendo la inercia la suma de las distancias al cuadrado de cada objeto del Cluster a su centroide:

$$\text{inercia} = \sum_{i=0}^n \min_{\mu_j \in C} \|x_i - \mu_j\|^2$$

Una vez obtenidos los valores de la inercia tras aplicar el K-means de 1 a N Clusters, representamos en una gráfica lineal la inercia respecto del número de Clusters. En esta gráfica se debería de apreciar un cambio brusco en la evolución de la inercia, teniendo la línea representada una forma similar a la de un brazo y su codo. El punto en el que se observa ese cambio brusco en la inercia nos dirá el número óptimo de Clusters a seleccionar para ese data set; o dicho de otra manera: el punto que representaría al codo del brazo será el número óptimo de Clusters para ese data set.

Elbow Method for selection of optimal “K” clusters



Es posible que al aplicar este método para ciertos conjunto de datos no se aprecie "el codo" o incluso se observen dos o más codos (o cambios bruscos en la evolución de la inercia). En ese caso habría que estudiar más en detalle o con otras técnicas el número óptimo de Clusters a seleccionar. Dada la finalidad didáctica de estos ejemplos, se aprecia muy bien "el codo" en la

[Go to Top](#)

4.- Importar librerías

```
In [1]: import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

4.1.- numpy

Instalación

```
conda install -c anaconda -y numpy
```

Descripción

Numpy es una biblioteca para el lenguaje de programación Python que (entre otras cosas) brinda soporte para grandes matrices multidimensionales.

¿Por qué es eso importante? Usando NumPy, podemos expresar imágenes como matrices multidimensionales.

Representar imágenes como matrices NumPy no solo es computacional y eficiente en el uso de recursos, sino que muchas otras bibliotecas de procesamiento de imágenes y aprendizaje automático también usan representaciones de matrices NumPy. Además, al usar las funciones matemáticas de alto nivel incorporadas de NumPy, podemos realizar rápidamente análisis numéricos en una imagen.

Link

[Numpy](#)

4.2.- pandas

Instalación

```
pip install pandas
```

o

```
conda install -c anaconda pandas
```

Descripción

pandas es una herramienta de manipulación y análisis de datos de código abierto rápida, potente, flexible y fácil de usar, construido sobre el lenguaje de programación Python.

Link

[pandas](#)

4.3.- Matplotlib

Instalación

```
| conda install -c conda-forge -y matplotlib
```

Descripción

Matplotlib es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arreglos en el lenguaje de programación Python y su extensión matemática NumPy.

Por medio del comando mágico **%matplotlib inline** se indica que los despliegues gráficos de la biblioteca matplotlib se despliegan en el cuaderno Jupyter directamente.

Link

[Matplotlib](#)

4.4.- Seaborn

Instalación

```
| pip install seaborn
```

o

```
| conda install -c anaconda seaborn
```

Descripción

Seaborn es una biblioteca de visualización de datos de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.

Link

[seaborn](#)

5.- Datos

[Go to Top](#)

```
In [2]: path_data = "./Data/Mall_Customers.csv"  
df = pd.read_csv(path_data)
```

5.1.- Análisis de datos exploratorios

Ver lo largo y ancho de los datos

```
In [3]: df.shape
```

```
Out[3]: (200, 5)
```

Ver un resumen conciso del DataFrame. Ver resumen del conjunto de datos

```
In [4]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 200 entries, 0 to 199  
Data columns (total 5 columns):  
 #   Column           Non-Null Count  Dtype     
---  --     
 0   CustomerID      200 non-null    int64    
 1   Gender          200 non-null    object    
 2   Age             200 non-null    int64    
 3   Annual Income (k$) 200 non-null    int64    
 4   Spending Score (1-100) 200 non-null    int64  
dtypes: int64(4), object(1)  
memory usage: 7.9+ KB
```

Detecta valores perdidos.

```
In [5]: df.isnull().sum()
```

```
Out[5]: CustomerID      0  
Gender          0  
Age            0  
Annual Income (k$) 0  
Spending Score (1-100) 0  
dtype: int64
```

Imprime las primeras filas.

```
In [6]: df.head()
```

	CustomerID	Gender	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Genera estadísticas descriptivas.

```
In [7]: df.describe()
```

Out[7]:

	CustomerID	Age	Annual Income (k\$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

Evaluar la importancia de las variables categóricas y ID

In [8]: `df['CustomerID'].unique()`

Out[8]: `array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200], dtype=int64)`

In [9]: `len(df['CustomerID'].unique())`

Out[9]: 200

Contamos con 200 etiquetas de identificación únicas para la variable **CustomerID** en un conjunto de datos de 200 registros, esto quiere decir que existe un identificador único para cada uno de los registros, por lo tanto no es una variable que otorgue valor valor y la borraremos

In [10]: `df['Gender'].unique()`

Out[10]: `array(['Male', 'Female'], dtype=object)`

In [11]: `len(df['Gender'].unique())`

Out[11]: 2

Contamos con 2 categorías de etiquetas en la variable **Gender**.

Borramos **CustomerID** del Data Frame

```
In [12]: df.drop(['CustomerID'], axis=1, inplace=True)
```

Repetimos el análisis para saber que tenemos

```
In [13]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Gender          200 non-null    object  
 1   Age             200 non-null    int64  
 2   Annual Income (k$) 200 non-null    int64  
 3   Spending Score (1-100) 200 non-null    int64  
dtypes: int64(3), object(1)
memory usage: 6.4+ KB
```

```
In [14]: df.head()
```

```
Out[14]:   Gender  Age  Annual Income (k$)  Spending Score (1-100)
0       Male    19              15                  39
1       Male    21              15                  81
2     Female    20              16                  6
3     Female    23              16                 77
4     Female    31              17                 40
```

```
In [15]: df.describe()
```

```
Out[15]:      Age  Annual Income (k$)  Spending Score (1-100)
count  200.000000        200.000000        200.000000
mean   38.850000        60.560000        50.200000
std    13.969007        26.264721        25.823522
min    18.000000        15.000000        1.000000
25%   28.750000        41.500000        34.750000
50%   36.000000        61.500000        50.000000
75%   49.000000        78.000000        73.000000
max   70.000000        137.000000       99.000000
```

Podemos ver que hay 1 columna no numérica **Gender** en el conjunto de datos. Lo convertiremos en equivalentes enteros.

5.2.- variable(s) categórica en numeros

1
2
3
4

Declarar df final

```
In [16]: X = df
```

Convertir variable categórica en numeros

```
In [17]: from sklearn.preprocessing import LabelEncoder  
  
le = LabelEncoder()  
  
X['Gender'] = le.fit_transform(X['Gender'])
```

AED Rapido

```
In [18]: X.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 200 entries, 0 to 199  
Data columns (total 4 columns):  
 #   Column           Non-Null Count  Dtype    
---  --    
 0   Gender          200 non-null    int32   
 1   Age             200 non-null    int64   
 2   Annual Income (k$) 200 non-null    int64   
 3   Spending Score (1-100) 200 non-null    int64  
dtypes: int32(1), int64(3)  
memory usage: 5.6 KB
```

```
In [19]: X.head()
```

```
Out[19]: Gender  Age  Annual Income (k$)  Spending Score (1-100)
```

0	1	19	15	39
1	1	21	15	81
2	0	20	16	6
3	0	23	16	77
4	0	31	17	40

[Go to Top](#)

6.- normalización estadística



```
In [20]: cols = X.columns
```

```
In [21]: from sklearn.preprocessing import MinMaxScaler  
  
ms = MinMaxScaler()  
  
X = ms.fit_transform(X)
```

```
In [22]: X = pd.DataFrame(X, columns=[cols])
```

```
In [23]: X.head()
```

Out[23]:

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)
0	1.0	0.019231	0.000000	0.387755
1	1.0	0.057692	0.000000	0.816327
2	0.0	0.038462	0.008197	0.051020
3	0.0	0.096154	0.008197	0.775510
4	0.0	0.250000	0.016393	0.397959

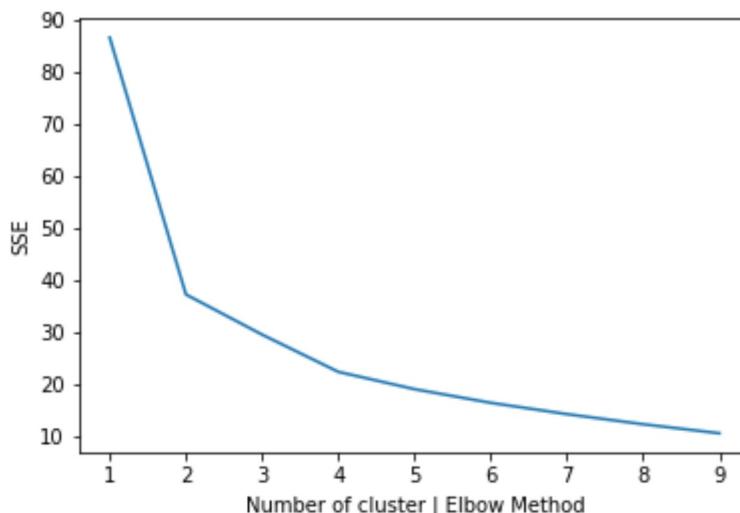
[Go to Top](#)

7.- Utilice el método del codo para encontrar el K-número óptimo para los datos



In [24]:

```
from sklearn.cluster import KMeans  
  
sse = {}  
  
for k in range(1, 10):  
    kmeans = KMeans(n_clusters=k, max_iter=1000)  
    kmeans.fit(X)  
    #print(data["clusters"])  
    sse[k] = kmeans.inertia_ # Inertia: Sum of distances of samples to their  
  
plt.figure()  
plt.plot(list(sse.keys()), list(sse.values()))  
plt.xlabel("Number of cluster | Elbow Method")  
plt.ylabel("SSE")  
plt.show()
```



[Go to Top](#)

8.- K-Means con diferentes clústeres



```
n_clusters = 2
```

```
In [25]: kmeans = KMeans(n_clusters=2, random_state=0)  
kmeans.fit(X)  
  
# check how many of the samples were correctly labeled  
labels = kmeans.labels_
```

```
In [26]: kmeans.inertia_
```

```
Out[26]: 37.27227241211877
```

```
In [27]: n_clusters = 3
```

```
In [28]: kmeans = KMeans(n_clusters=3, random_state=0)  
kmeans.fit_predict(X)  
  
# check how many of the samples were correctly labeled  
labels = kmeans.labels_
```

```
In [29]: kmeans.inertia_
```

```
Out[29]: 29.552857611943868
```

```
In [30]: n_clusters = 4
```

```
In [31]: kmeans = KMeans(n_clusters=4, random_state=0)  
kmeans.fit_predict(X)  
  
# check how many of the samples were correctly labeled  
labels = kmeans.labels_
```

```
In [32]: kmeans.inertia_
```

```
Out[32]: 22.39096001192855
```

```
In [33]: centers_on_PCs = kmeans.cluster_centers_
```

[Go to Top](#)

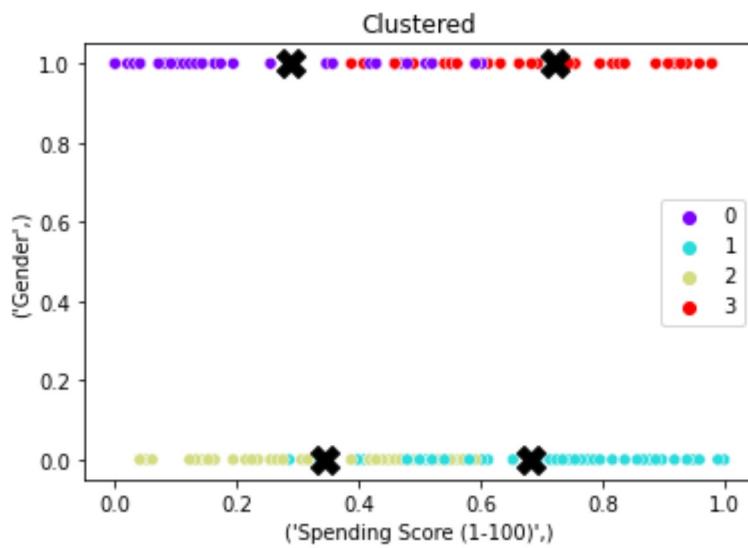
9.- Plot

```
In [34]: ### palette = sns.color_palette('bright', len(set(labels)))  
sns.scatterplot(X.iloc[:,3], X.iloc[:, 0], hue=labels, legend='full', palette=palette)  
plt.scatter(x=centers_on_PCs[:,3],y=centers_on_PCs[:,0],s=200,c="k",marker="x")  
plt.title('Clustered')
```

```
C:\Users\Equipo\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
```

arning: Pass the following variables as keyword args: `x, y`. From version 0.12, the only valid positional argument will be `'data'`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

Out[34]: Text(0.5, 1.0, 'Clustered')



Si te sirvio, **COMPARTE!** 😊

- Adios

Mis redes:

- [GIT](#)
- [linkedin](#)