K-Means Clustering

Objetivos

- Utilizar K-means para clusterizar.

Introducción

Hay muchos modelos para el clustering, en esta notebook presentaremos uno de los más sencillos: **K-means**. K-means es muy útil si necesita descubir rápidamente información de datos no etiquetados. Aquí aprenderemos a utilizar K-means para la segmentación de clientes.

Algunas aplicaciones de K-means en el mundo real:

- Segmentación de clientes
- Entender lo que los visitantes de una web intentan lograr
- Reconocimiento de patrones
- Machine Learning
- Compresión de datos

En esta notebook veremos 2 ejemplos:

- K-means en un dataset generado aleatoriamente
- K-means para la segmentación de clientes

Tabla de contenido

Importando librerías

Importamos las librerías necesarias

In [1]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs
%matplotlib inline
```

```
C:\Users\marco\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143: FutureWar
ning: The sklearn.datasets.samples_generator module is  deprecated in version 0.22 and
will be removed in version 0.24. The corresponding classes / functions should instead
be imported from sklearn.datasets. Anything that cannot be imported from sklearn.datas
ets is now part of the private API.
  warnings.warn(message, FutureWarning)
```

k-Means en un dataset generado aleatoriamente

Crearemos nuestro propio dataset para este laboratorio.

Primero debemos establecer una semilla aleatoria. Utilizaremos la función **random.seed()** de numpy, donde la estableceremos en 0.

In [2]:
```
np.random.seed(0)
```

Luego realizaremos clusters de puntos aleatorios utilizando la clase make_blobs. La clase make_blobs puede tomar muchas entradas, aquí usaremos algunas específicas.

Entrada

- **n_samples**: El número total de puntos igualmente divididos entre clusters.
  - El valor será: 5000
- **centers**: El número de centros a generar, o las ubicaciones fijas de los centros
  - El valor será: [[4, 4], [-2, -1], [2, -3],[1,1]]
- **cluster_std**: La desviación estándar de los clusters
  - El valor será: 0.9

Salida

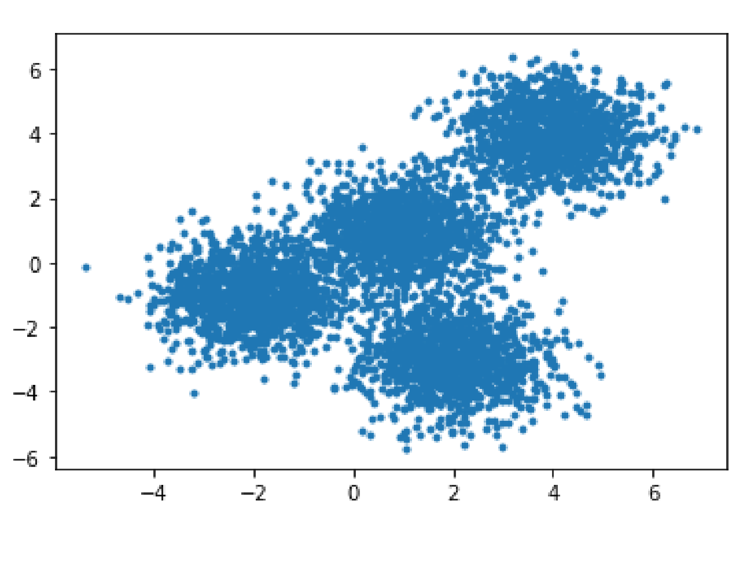
- **X**: Arreglo de tamaño [n_samples, n_features]. (Matriz de características)
  - Las muestras generadas.
- **y**: Arreglo de la forma [n_samples]. (Vector respuesta)
  - Las etiquetas enteras para las membresías a clusters de cada muestra.

In [3]:
```
X, y = make_blobs(n_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1, 1]], cluster_
```

Desplegamos un gráfico de dispersión para los datos generados aleatoriamente.

In [4]:
```
plt.scatter(X[:, 0], X[:, 1], marker='.')
```

Out[4]:
```
<matplotlib.collections.PathCollection at 0x26f6952f1f0>
```



Estableciendo K-Means

La clase K-means tiene muchos parámetros, aquí los siguientes 3:

- **init**: Método de inicialización de los centroides
  - El valor será: "k-means++"
  - k-means++: Selecciona centros inicilaes para los clusters de forma inteligente de modo de acelerar la convergencia
- **n_clusters**: El número de clusters a formar así como el número de centroides a generar.
  - El valor será: 4 (ya que tenemos 4 centros)
- **n_init**: El número de veces que el algoritmo de K-means es ejecutado con diferentes semillas de centroides. Los resultados finales serán la mejor salida de n_init ejecuciones consecutivas en términos de inercia.
  - El valor será: 12

Inicializamos K-means con estos parámetros, donde la salida es llamada **k_means**.

In [5]:
```
k_means = KMeans(init = "k-means++", n_clusters = 4, n_init = 12)
```

Ajustamos el modelo KMeans con la matriz de características creada arriba **X**

In [6]:
```
k_means.fit(X)
```

Out[6]:
```
KMeans(n_clusters=4, n_init=12)
```

Tomemos las etiquetas para cada punto en el modelo usando el atributo **.labels_** de KMeans y guardémoslo como **k_means_labels**.

In [7]:
```
k_means_labels = k_means.labels_
k_means_labels
```

Out[7]:
```
array([0, 2, 2, ..., 1, 0, 0])
```

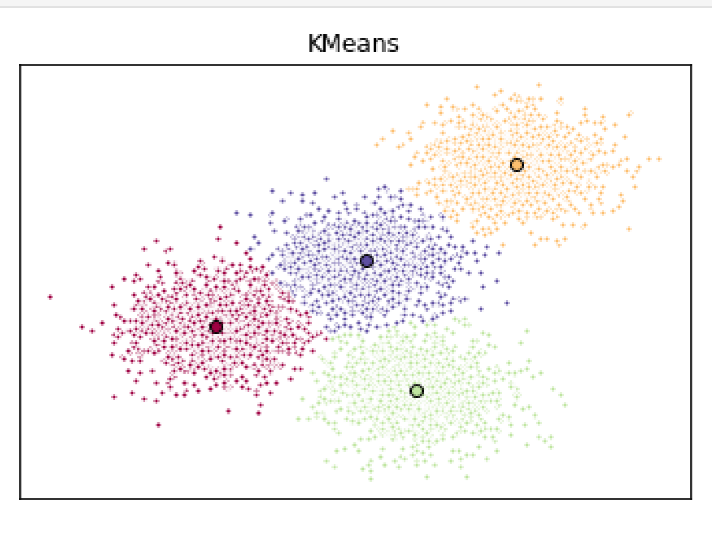
Obtendremos las coordenadas de los centros de los clusters usando **.cluster*centers*** de KMeans y guardémoslo como **k_means_cluster_centers**

In [8]:
```
k_means_cluster_centers = k_means.cluster_centers_
k_means_cluster_centers
```

Out[8]:
```
array([[-2.03375169, -0.99827293],
       [ 3.97334234,  3.98758687],
       [ 1.99876902, -3.01796355],
       [ 0.96959198,  0.98543802]])
```

Creando la visualización

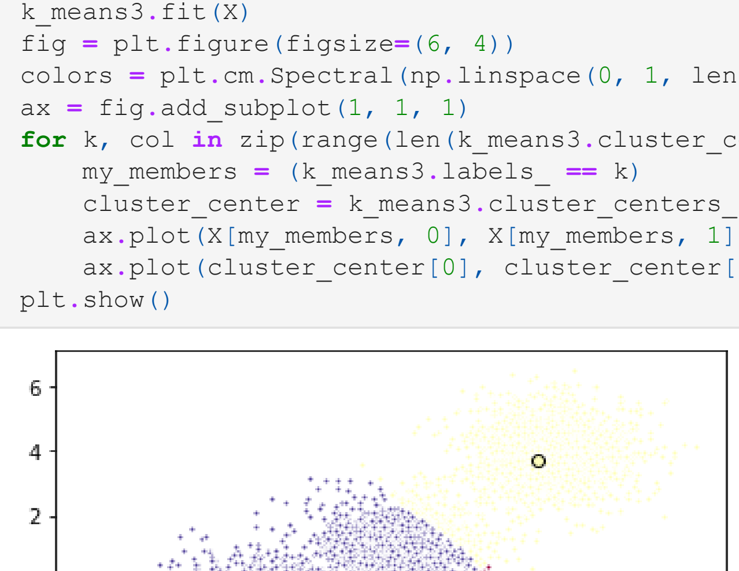
In [9]:
```
# Inicializamos el gráfico con las dimensiones especificadas
fig = plt.figure(figsize=(6, 4))

# Colors usa un color map, que produce un arreglo de colores basado en el número de
# etiquetas que hay. Utilizamos set(k_means_labels) para obtener etiquetas únicas
colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k_means_labels))))

# Creamos un gráfico
ax = fig.add_subplot(1, 1, 1)



# Loop for que grafica los puntos de datos y centroides.
# k tiene un rango 0-3, que coincidirá copn los posibles clusters en que se encuentre
for k, col in zip(range(len([[4,4], [-2, -1], [2, -3], [1, 1]])), colors):

    # Creamos una lista de todos los puntos de datos, donde los puntos que están en
    # en el cluster (ejemplo cluster 0) son etiquetados como true, de otro modo son
    # etiquetados como false
    my_members = (k_means_labels == k)

    # Definimos los centroides
    cluster_center = k_means_cluster_centers[k]

    # Graficamos los puntos de datos con el color col
    ax.plot(X[my_members, 0], X[my_members, 1], 'w', markerfacecolor=col, marker='.')

    # Graficamos los centroides con el color especificado, pero con un outline más os
    ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,  markered

# Título
ax.set_title('KMeans')

#
ax.set_xticks(())

#
ax.set_yticks(())

# Graficamos
plt.show()
```



Práctica

Intente clusterizar el dataset de arriba en 3 clusters.

In [10]:
```
k_means3 = KMeans(init = "k-means++", n_clusters = 3, n_init = 12)
k_means3.fit(X)
fig = plt.figure(figsize=(6, 4))
colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k_means3.labels_))))
ax = fig.add_subplot(1, 1, 1)
for k, col in zip(range(len(k_means3.cluster_centers_)), colors):
    my_members = (k_means3.labels_ == k)
    cluster_center = k_means3.cluster_centers_[k]
    ax.plot(X[my_members, 0], X[my_members, 1], 'w', markerfacecolor=col, marker='.')
    ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,  markered
plt.show()
```



Segmentación de clientes con K-Means

Imagine que tiene un dataset de clientes y necesita realizar segmentación acorde a datos históricos. La segmentación de clientes es la práctica de particionar una base de clientes en grupos de individuos que tengan características similares. Esto permite por ejemplo asignar recursos de marketing específicos. Un grupo, por ejemplo, puede contener clientes de alto perfil y bajo riesgo que es más probable compren productos o contraten servicios. Una tarea del negocio puede ser retener estos clientes. Otro grupo podrían ser las organizaciones no benéficas y así sucesivamente.

In [12]:
```
#!wget -O Cust_Segmentation.csv https://cf-courses-data.s3.us.cloud-object-storage.app

import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
filename = 'Cust_Segmentation.csv'
urllib.request.urlretrieve(url, filename)
```

Out[12]:
```
('Cust_Segmentation.csv', <http.client.HTTPMessage at 0x26f69cb9310>)
```

Cargando los datos desde el CSV

In [13]:
```
import pandas as pd
cust_df = pd.read_csv("Cust_Segmentation.csv")
cust_df.head()
```

Out[13]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Address	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	NBA001	6.3
1	2	47	1	26	100	4.582	8.218	0.0	NBA021	12.8
2	3	33	2	10	57	6.111	5.802	1.0	NBA013	20.9
3	4	29	2	4	19	0.681	0.516	0.0	NBA009	6.3
4	5	47	1	31	253	9.308	8.908	0.0	NBA008	7.2

Pre-procesamiento</h2

En el dataset, **Address** es una variable categórica. El algoritmo k-means no es aplicable directamente a variables categóricas porque la función de distancia euclideana no es significativa para variables discretas, así que eliminémosla.

In [14]:
```
df = cust_df.drop('Address', axis=1)
df.head()
```

Out[14]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	6.3
1	2	47	1	26	100	4.582	8.218	0.0	12.8
2	3	33	2	10	57	6.111	5.802	1.0	20.9
3	4	29	2	4	19	0.681	0.516	0.0	6.3
4	5	47	1	31	253	9.308	8.908	0.0	7.2

Normalizando sobre la desviación estándar

Normalicemos el dataset. Por qué normalizamos? La normalización es un método estadístico que ayuda a los algoritmos matemáticos a interpretar características con diferentes magnitudes y distribuciones de forma igualitaria. Usamos **StandardScaler()** para normalizar nuestro dataset.

In [15]:
```
from sklearn.preprocessing import StandardScaler
X = df.values[:,1:]
X = np.nan_to_num(X)
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet
```

Out[15]:
```
array([[ 0.74291541,  0.31212243, -0.37878978, ..., -0.59048916,
        -0.52379654, -0.57652509],
       [ 1.48949049, -0.76634938,  2.5737211 , ...,  1.51296181,
        -0.52379654,  0.39138677],
       [-0.25251804,  0.31212243,  0.2117124 , ...,  0.80170393,
         1.90913822,  1.59755385],
       ...,
       [-1.24795149,  2.46906604, -1.26454304, ...,  0.03863257,
         1.90913822,  3.45892281],
       [-0.37694723, -0.76634938,  0.50696349, ..., -0.70147601,
        -0.52379654, -1.08281745],
       [ 2.1116364 , -0.76634938,  1.09746566, ...,  0.16463355,
        -0.52379654, -0.2340332 ]])
```

Modelando

En nuestro ejemplo (si no tuviésemos acceso al algoritmo k-means) sería lo mismo que adivinar que cada grupo de clientes tiene cierta edad, ingresos, educación, etc, con múltiples test y experimentos. De todos modos, el clustering k-means puede facilitar esto.

Apliquemos K-means a nuestro dataset y miremos las etiquetas de clusters.

In [16]:
```
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

```
[1 0 1 1 2 0 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 1 1
 1 1 0 1 0 1 2 1 0 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0 1 0 0 1 1 0 1 1 1 0 0 0 1
 1 1 1 1 0 1 0 0 2 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1
 1 1 1 1 1 1 0 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1
 1 1 1 1 0 1 1 0 1 0 1 1 0 2 1 0 1 1 1 1 1 1 2 0 1 1 1 1 0 1 1 0 0 1 0 1 0
 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 2 0 1 1 1 1 1 1 1 1 0 1 1 1 1
 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 0 0 1 1 1 1 1 1
 1 1 1 0 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1
 1 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 2 1 1 1 0 1 0 0 0 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 2
 1 1 1 1 1 1 0 1 1 1 2 1 1 1 1 0 1 2 1 1 1 1 0 1 0 0 0 1 1 0 0 1 1 1 1 1 1
 1 0 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1
 1 0 0 1 1 1 1 1 1 1 1 1 1 1 2 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1 1 2 1 2 1
 1 2 1 1 1 1 1 1 1 1 1 0 1 0 1 1 2 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0
 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0
 0 1 1 0 1 0 1 1 0 1 0 1 1 2 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 0 0 1 1
 0 1 1 1 0 1 2 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1
 1 1 0 1 1 0 1 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1
 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 0 0 1 1 0 1 1 1 1 1 0 0
 1 1 1 1 1 1 1 0 1 1 1 1 1 1 2 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0]
```

Conocimientos

Asignamos las etiquetas a cada fila en el dataframe.

In [17]:
```
df["Clus_km"] = labels
df.head(5)
```

Out[17]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio	Clus_km
0	1	41	2	6	19	0.124	1.073	0.0	6.3	1
1	2	47	1	26	100	4.582	8.218	0.0	12.8	0
2	3	33	2	10	57	6.111	5.802	1.0	20.9	1
3	4	29	2	4	19	0.681	0.516	0.0	6.3	1
4	5	47	1	31	253	9.308	8.908	0.0	7.2	2

Podemos cheqeuar los valores de los centroides promediando las características en cada cluster.

In [18]:
```
df.groupby('Clus_km').mean()
```

Out[18]:

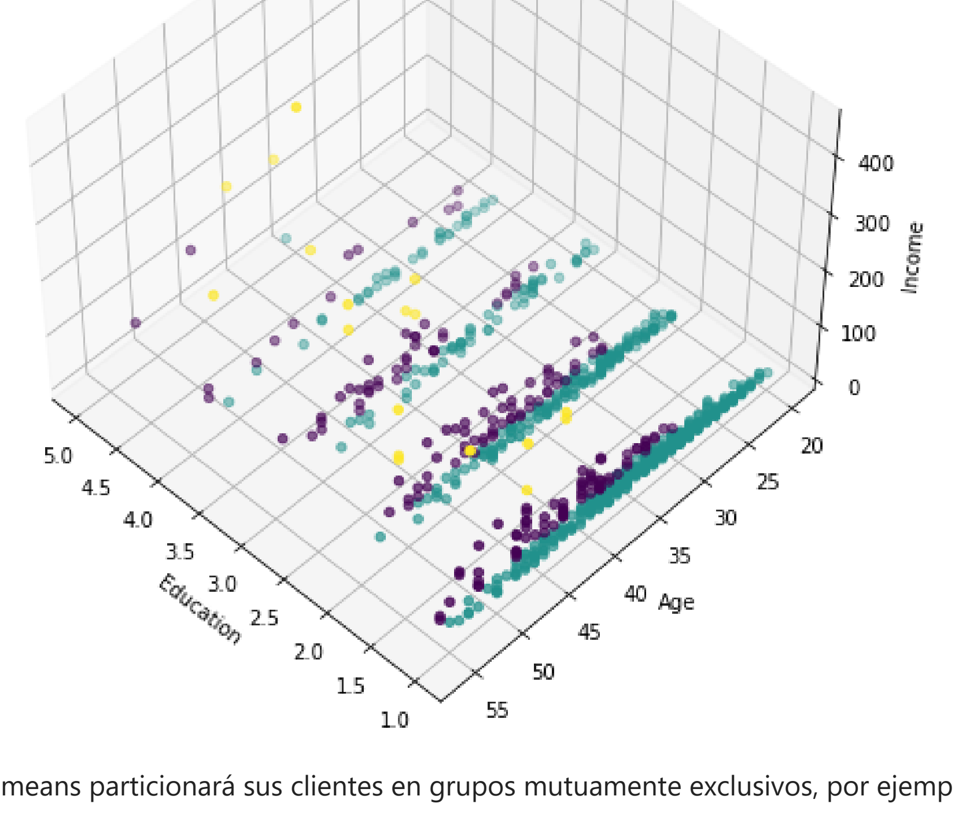
	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeR
Clus_km									
0	403.780220	41.368132	1.961538	15.252747	84.076923	3.114412	5.770352	0.172414	10.725
1	432.006154	32.967692	1.613846	6.389231	31.204615	1.032711	2.108345	0.284658	10.095
2	410.166667	45.388889	2.666667	19.555556	227.166667	5.678444	10.907167	0.285714	7.322

Miremos la distribución de los clientes basado en su edad e ingreso:

In [19]:
```
area = np.pi * ( X[:, 1])**2
plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(np.float), alpha=0.5)
plt.xlabel('Age', fontsize=18)
plt.ylabel('Income', fontsize=16)

plt.show()
```

In [20]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize=(8, 6))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

plt.cla()
# plt.ylabel('Age', fontsize=18)
# plt.xlabel('Income', fontsize=16)
# plt.zlabel('Education', fontsize=16)
ax.set_xlabel('Education')
ax.set_ylabel('Age')
ax.set_zlabel('Income')

ax.scatter(X[:, 1], X[:, 0], X[:, 3], c= labels.astype(np.float))
```

Out[20]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x26f6bc73f70>
```



k-means particionará sus clientes en grupos mutuamente exclusivos, por ejemplo, en 3 clusters. Los clientes en cada cluster son similares entre sí demográficamente. Ahora podemos crear un perfil para cada grupo, considerando las características comunes de cada cluster. Por ejemplo, los clusters pueden ser:

- Afluente, educado y anciano
- Mediana edad y mediano ingreso
- Joven y de bajos ingresos