K-means

K-Means es un algoritmo no supervisado de Clustering. Se utiliza cuando tenemos un montón de datos sin etiquetar. El objetivo de este algoritmo es el de encontrar “K” grupos (clusters) entre los datos crudos.

El algoritmo trabaja iterativamente para asignar a cada “punto” (las filas de nuestro conjunto de entrada forman una coordenada) uno de los “K” grupos basado en sus características. Son agrupados en base a la similitud de sus features (las columnas). Como resultado de ejecutar el algoritmo tendremos:

- Los “centroides” de cada grupo que serán unas “coordenadas” de cada uno de los K conjuntos que se utilizarán para poder etiquetar nuevas muestras.

- Etiquetas para el conjunto de datos de entrenamiento. Cada etiqueta perteneciente a uno de los K grupos formados.

Los grupos se van definiendo de manera “orgánica”, es decir que se va ajustando su posición en cada iteración del proceso, hasta que converge el algoritmo. Una vez hallados los centroids deberemos analizarlos para ver cuales son sus características únicas, frente a la de los otros grupos. Estos grupos son las etiquetas que genera el algoritmo.

El algoritmo de Clustering K-means es usado uno de los más usados para encontrar grupos ocultos, o sospechados en teoría sobre un conjunto de datos no etiquetado. Esto puede servir para confirmar o descartar alguna teoría que teníamos asumida de nuestros datos. Y también puede ayudarnos a descubrir relaciones asombrosas entre conjuntos de datos, que de manera manual, no hubiéramos reconocido. Una vez que el algoritmo ha ejecutado y obtenido las etiquetas, será fácil clasificar nuevos valores o muestras entre los grupos obtenidos.

Algunos casos de uso son:
- Segmentación por Comportamiento: relacionar el carrito de compras de un usuario, sus tiempos de acción e información del perfil.
- Categorización de Inventario: agrupar productos por actividad en sus ventas
- Detectar anomalías o actividades sospechosas: según el comportamiento en una web reconocer un troll o un bot de un usuario normal

Las “features” o características que utilizaremos como entradas para aplicar el algoritmo k-means deberán ser de valores numéricos, continuos en lo posible. En caso de valores categóricos (por ej. Hombre/Mujer o Ciencia Ficción, Terror, Novela,etc) se puede intentar pasarlo a valor numérico, pero no es recomendable pues no hay una “distancia real” como en el caso de géneros de película o libros.

Además es recomendable que los valores utilizados estén normalizados, manteniendo una misma escala. En algunos casos también funcionan mejor datos porcentuales en vez de

absolutos. No conviene utilizar features que estén correlacionados o que sean escalares de otros.

El algoritmo utiliza una proceso iterativo en el que se van ajustando los grupos para producir el resultado final. Para ejecutar el algoritmo deberemos pasar como entrada el conjunto de datos y un valor de K. El conjunto de datos serán las características o features para cada punto. Las posiciones iniciales de los K centroides serán asignadas de manera aleatoria de cualquier punto del conjunto de datos de entrada. Luego se itera en dos pasos:

1- Paso de Asignación de datos

En este paso, cada “fila” de nuestro conjunto de datos se asigna al centroide más cercano basado en la distancia cuadrada Euclideana. Se utiliza la siguiente fórmula (donde dist() es la distancia Euclideana standard):

$$\underset{c_i \in C}{\arg\min}\ dist(c_i,\ x)^2$$

2-Paso de actualización de Centroid

En este paso los centroides de cada grupo son recalculados. Esto se hace tomando una media de todos los puntos asignados en el paso anterior.

$$c_i = \frac{1}{|S_i|}\sum_{x_i \in S_i} x_i$$

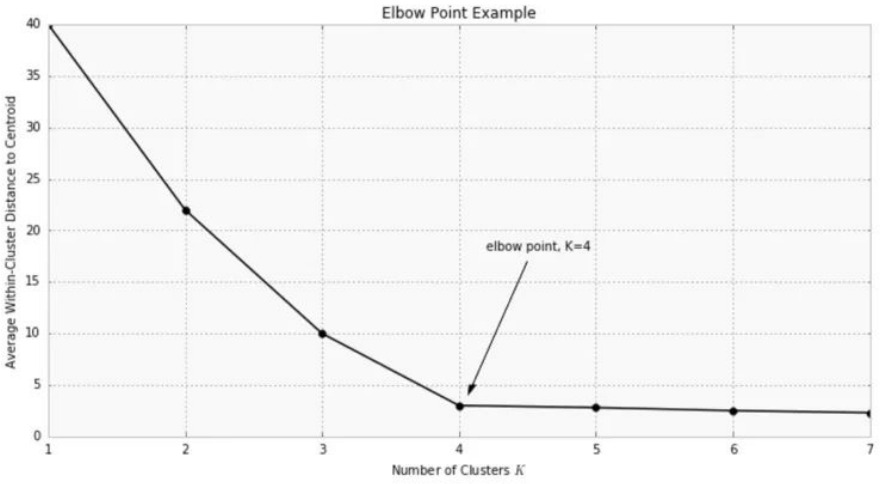
El algoritmo itera entre estos pasos hasta cumplir un criterio de detención:
* si no hay cambios en los puntos asignados a los grupos,
* o si la suma de las distancias se minimiza,
* o se alcanza un número máximo de iteraciones.

El algoritmo converge a un resultado que puede ser el óptimo local, por lo que será conveniente volver a ejecutar más de una vez con puntos iniciales aleatorios para confirmar si hay una salida mejor.

Este algoritmo funciona pre-seleccionando un valor de K. Para encontrar el número de clusters en los datos, deberemos ejecutar el algoritmo para un rango de valores K, ver los resultados y comparar características de los grupos obtenidos.

En general no hay un modo exacto de determinar el valor K, pero se puede estimar con aceptable precisión siguiendo la siguiente técnica:

Una de las métricas usada para comparar resultados es la distancia media entre los puntos de datos y su centroid. Como el valor de la media disminuirá a medida de aumentemos el valor de K, deberemos utilizar la distancia media al centroide en función de K y encontrar el “punto codo”, donde la tasa de descenso se “afila”. Aquí vemos una gráfica a modo de ejemplo:



Comenzaremos importando las librerías que nos asistirán para ejecutar el algoritmo y graficar.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min
from mpl_toolkits.mplot3d import Axes3D
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
```

Importamos el archivo csv y para simplificar, suponemos que el archivo se encuentra en el mismo directorio que el notebook y vemos los primeros 5 registros del archivo tabulados.

```
dataframe = pd.read_csv(r"analisis.csv")
dataframe.head()
```

	usuario	op	co	ex	ag	ne	wordcount	categoria
0	3gerardpique	34.297953	28.148819	41.948819	29.370315	9.841575	37.0945	7
1	aguerosergiokun	44.986842	20.525865	37.938947	24.279098	10.362406	78.7970	7
2	albertochicote	41.733854	13.745417	38.999896	34.645521	8.836979	49.2604	4
3	AlejandroSanz	40.377154	15.377462	52.337538	31.082154	5.032231	80.4538	2
4	alfredocasero1	36.664677	19.642258	48.530806	31.138871	7.305968	47.0645	4

También podemos ver una tabla de información estadística que nos provee Pandas dataframe:

```
dataframe.describe()
```

	op	co	ex	ag	ne	wordcount	categoria
count	140.000000	140.000000	140.000000	140.000000	140.000000	140.000000	140.000000
mean	44.414591	22.977135	40.764428	22.918528	8.000098	98.715484	4.050000
std	8.425723	5.816851	7.185246	7.657122	3.039248	44.714071	2.658839
min	30.020465	7.852756	18.693542	9.305985	1.030213	5.020800	1.000000
25%	38.206484	19.740299	36.095722	17.050993	6.086144	66.218475	2.000000
50%	44.507091	22.466718	41.457492	21.384554	7.839722	94.711400	3.500000
75%	49.365923	26.091606	45.197769	28.678867	9.758189	119.707925	7.000000
max	71.696129	49.637863	59.824844	40.583162	23.978462	217.183200	9.000000

El archivo contiene diferenciadas 9 categorías -actividades laborales- que son:

1. Actor/actriz
2. Cantante
3. Modelo
4. Tv, series
5. Radio
6. Tecnología
7. Deportes
8. Política
9. Escritor

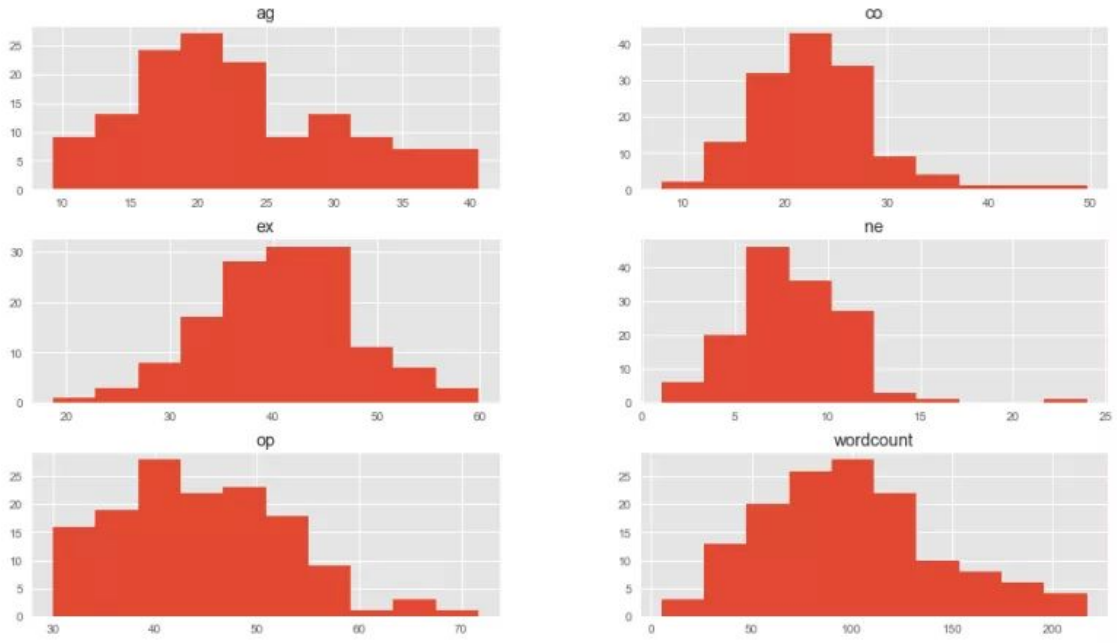
Para saber cuantos registros tenemos de cada uno hacemos:

```
print(dataframe.groupby('categoria').size())
```

```
categoria
1    27
2    34
3     9
4    19
5     4
6     8
7    17
8    16
9     6
dtype: int64
```

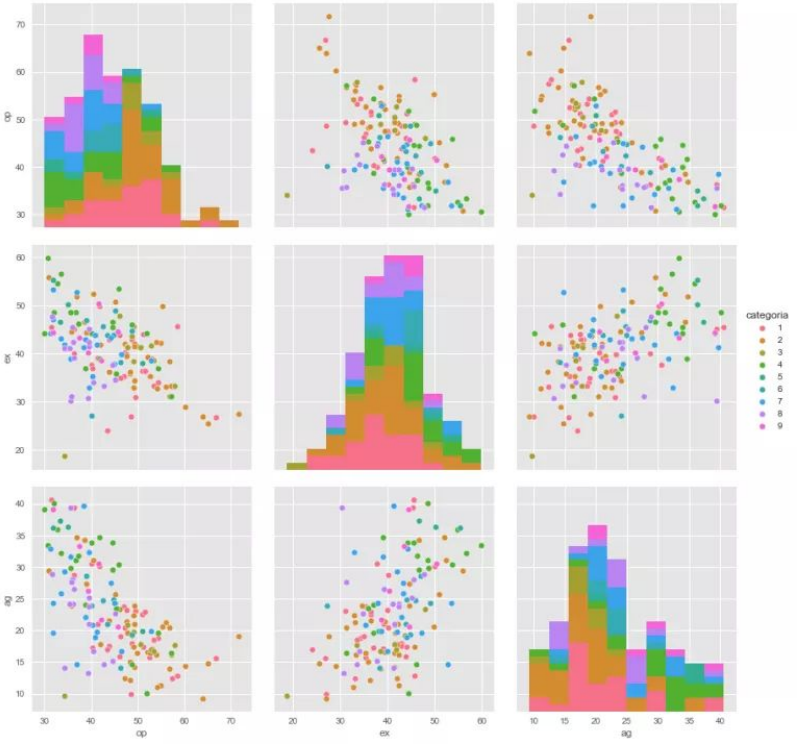
Como vemos tenemos 34 cantantes, 27 actores, 17 deportistas, 16 políticos,etc.
Veremos gráficamente nuestros datos para tener una idea de la dispersión de los mismos:

```
dataframe.drop(['categoria'],1).hist()
plt.show()
```



En este caso seleccionamos 3 dimensiones: op, ex y ag y las cruzamos para ver si nos dan alguna pista de su agrupación y la relación con sus categorías.

```
sb.pairplot(dataframe.dropna(),
hue='categoria',size=4,vars=["op","ex","ag"],kind='scatter')
```



Revisando la gráfica no pareciera que haya algún tipo de agrupación o correlación entre los usuarios y sus categorías.

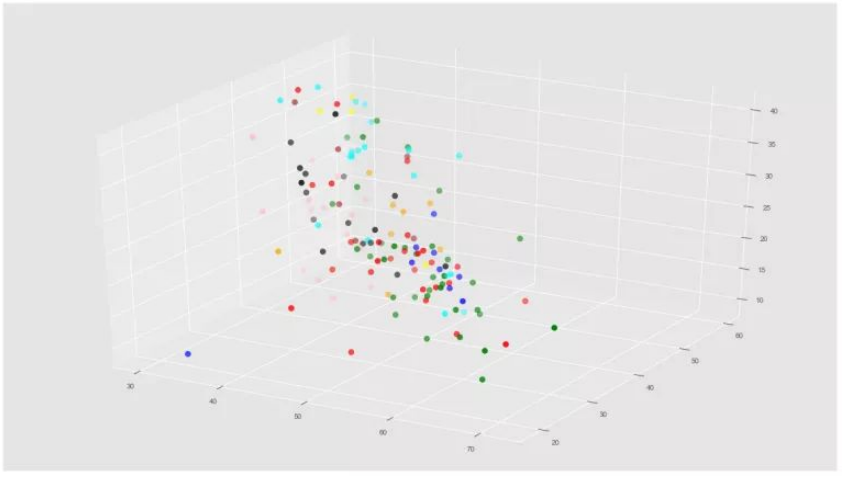
Concretamos la estructura de datos que utilizaremos para alimentar el algoritmo. Como se ve, sólo cargamos las columnas op, ex y ag en nuestra variable X.

```
X = np.array(dataframe[["op","ex","ag"]])
y = np.array(dataframe['categoria'])
X.shape
```

```
(140,3)
```

Ahora veremos una gráfica en 3D con 9 colores representando las categorías.

```
fig = plt.figure()
ax = Axes3D(fig)
colores=['blue','red','green','blue','cyan','yellow','orange',
'black','pink','brown','purple']
asignar=[]
for row in y:
    asignar.append(colores[row])
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=asignar,s=60)
```

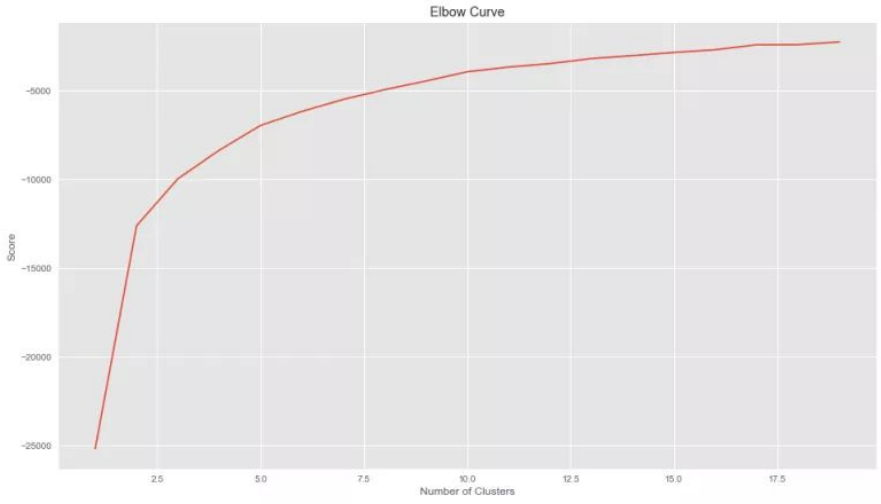


Veremos si con K-means, podemos “pintar” esta misma gráfica de otra manera, con clusters diferenciados.

Vamos a hallar el valor de K haciendo una gráfica e intentando hallar el “punto de codo” que comentábamos antes. Este es nuestro resultado:

```
Nc = range(1, 20)
kmeans = [KMeans(n_clusters=i) for i in Nc]
```

```
kmeans
score = [kmeans[i].fit(X).score(X) for i in
range(len(kmeans))]
score
plt.plot(Nc,score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```



Realmente la curva es bastante “suave”. Considero a 5 como un buen número para K. Según vuestro criterio podría ser otro.

Ejecutamos el algoritmo para 5 clusters y obtenemos las etiquetas y los centroids.

```
kmeans = KMeans(n_clusters=5).fit(X)
centroids = kmeans.cluster_centers_
print(centroids)
```

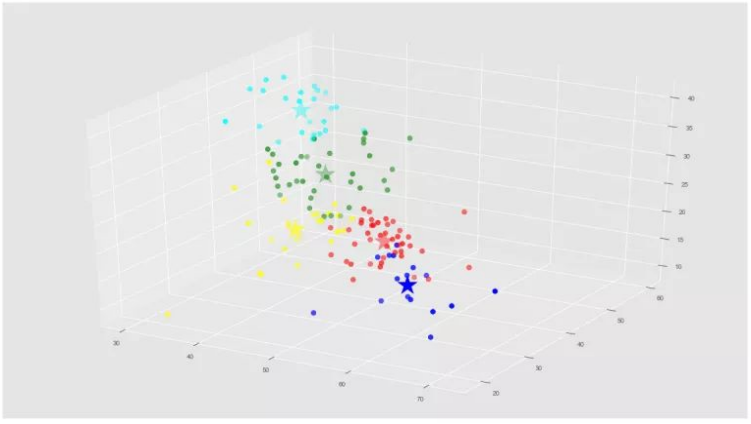
```
[[ 49.80086386  40.8972579   17.48224326]
 [ 39.63830586  44.75784737  25.86962057]
 [ 58.58657531  31.02839375  15.6120435 ]
 [ 34.5303535   48.01261321  35.01749504]
 [ 42.302263    33.65449587  20.812626  ]]
```

Ahora veremos esto en una gráfica 3D con colores para los grupos y veremos si se diferencian: (las estrellas marcan el centro de cada cluster)

```
# Predicting the clusters
labels = kmeans.predict(X)
# Getting the cluster centers
```

```
C = kmeans.cluster_centers_
colores=['red','green','blue','cyan','yellow']
asignar=[]
for row in labels:
    asignar.append(colores[row])

fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=asignar,s=60)
ax.scatter(C[:, 0], C[:, 1], C[:, 2], marker='*', c=colores,
s=1000)
```

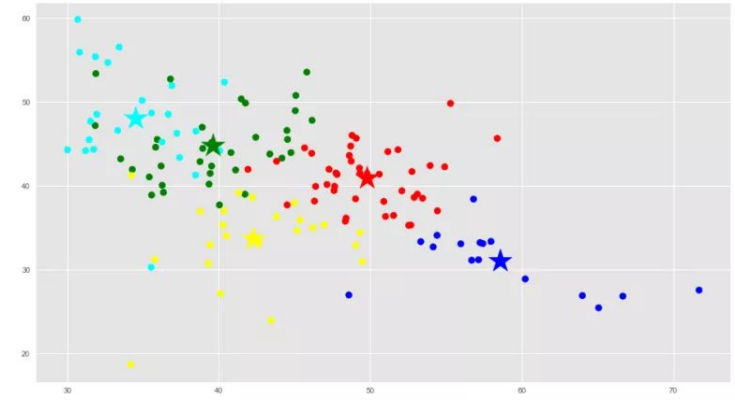


Aquí podemos ver que el Algoritmo de K-Means con K=5 ha agrupado a los 140 usuarios Twitter por su personalidad, teniendo en cuenta las 3 dimensiones que utilizamos: Openess, Extraversion y Agreeablenes. Pareciera que no hay necesariamente una relación en los grupos con sus actividades de Celebrity.

Haremos 3 gráficas en 2 dimensiones con las proyecciones a partir de nuestra gráfica 3D para que nos ayude a visualizar los grupos y su clasificación:

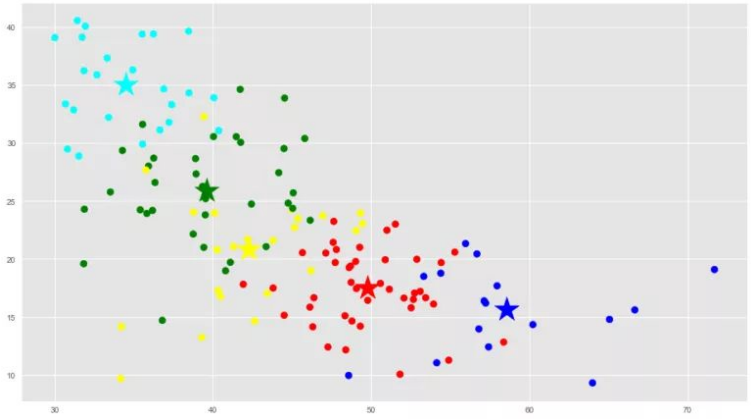
```
# Getting the values and plotting it
f1 = dataframe['op'].values
f2 = dataframe['ex'].values

plt.scatter(f1, f2, c=asignar, s=70)
plt.scatter(C[:, 0], C[:, 1], marker='*', c=colores, s=1000)
plt.show()
```



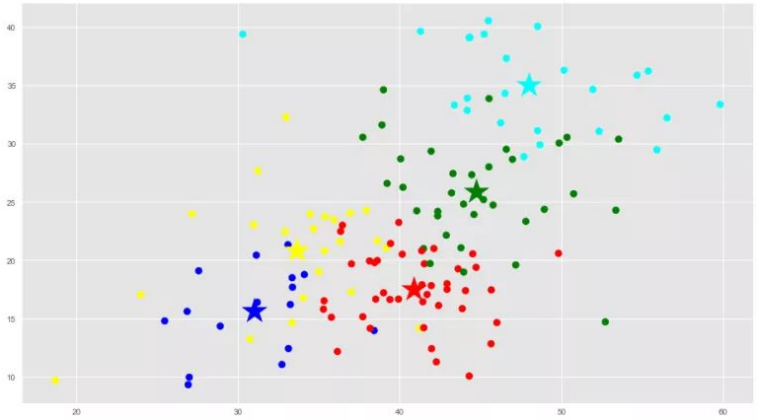
```
# Getting the values and plotting it
f1 = dataframe['op'].values
f2 = dataframe['ag'].values

plt.scatter(f1, f2, c=asignar, s=70)
plt.scatter(C[:, 0], C[:, 2], marker='*', c=colores, s=1000)
plt.show()
```



```
f1 = dataframe['ex'].values
f2 = dataframe['ag'].values

plt.scatter(f1, f2, c=asignar, s=70)
plt.scatter(C[:, 1], C[:, 2], marker='*', c=colores, s=1000)
plt.show()
```



En estas gráficas vemos que están bastante bien diferenciados los grupos.
Podemos ver cada uno de los clusters cuantos usuarios tiene:

```
copy =  pd.DataFrame()
copy['usuario']=dataframe['usuario'].values
```

```
copy['categoria']=dataframe['categoria'].values
copy['label'] = labels;
cantidadGrupo =  pd.DataFrame()
cantidadGrupo['color']=colores
cantidadGrupo['cantidad']=copy.groupby('label').size()
cantidadGrupo
```

	color	cantidad
0	red	42
1	green	33
2	blue	16
3	cyan	27
4	yellow	22

Y podemos ver la diversidad en rubros laborales de cada uno. Por ejemplo en el grupo 0 (rojo), vemos que hay de todas las actividades laborales aunque predominan de actividad 1 y 2 correspondiente a Actores y Cantantes con 11 y 15 famosos.

```
group_referrer_index = copy['label'] ==0
group_referrals = copy[group_referrer_index]

diversidadGrupo =  pd.DataFrame()
diversidadGrupo['categoria']=[0,1,2,3,4,5,6,7,8,9]
diversidadGrupo['cantidad']=group_referrals.groupby('categoria').size()
diversidadGrupo
```

	categoria	cantidad
0	0	NaN
1	1	11.0
2	2	15.0
3	3	6.0
4	4	3.0
5	5	1.0
6	6	2.0
7	7	2.0
8	8	1.0
9	9	1.0

De categoría 3 “modelos” hay 6 sobre un total de 9.

Buscaremos los usuarios que están más cerca a los centroides de cada grupo que podríamos decir que tienen los rasgos de personalidad característicos que representan a cada cluster:

```
#vemos el representante del grupo, el usuario cercano a su centroid
closest, _ = pairwise_distances_argmin_min(kmeans.cluster_centers_, X)
closest
```
array([21, 107, 82, 80, 91]) #posicion en el array de usuarios

```
users=dataframe['usuario'].values
for row in closest:
    print(users[row])
```

carmenelectra
*Pablo_Iglesias_*
JudgeJudy
JPVarsky
kobebryant

En los centros vemos que tenemos una modelo, un político, presentadora de Tv, locutor de Radio y un deportista.

Y finalmente podemos agrupar y etiquetar nuevos usuarios twitter con sus características y clasificarlos. Vemos el ejemplo con el usuario de David Guetta y nos devuelve que pertenece al grupo 1 (verde).

```
X_new = np.array([[45.92,57.74,15.66]]) #davidguetta

new_labels = kmeans.predict(X_new)
print(new_labels)
```

[1]

El algoritmo de K-means nos ayudará a crear clusters cuando tengamos grandes grupos de datos sin etiquetar, cuando queramos intentar descubrir nuevas relaciones entre features o para probar o declinar hipótesis que tengamos de nuestro negocio.

Atención: Puede haber casos en los que no existan grupos naturales, o clusters que contengan una verdadera razón de ser. Si bien K-means siempre nos brindará “k clusters”, quedará en nuestro criterio reconocer la utilidad de los mismos o bien revisar nuestras features y descartar las que no sirven o conseguir nuevas. También tener en cuenta que en este ejemplo estamos utilizando como medida de similitud entre features la distancia Euclideana pero podemos utilizar otras diversas funciones que podrían arrojar mejores resultados (como Manhattan, Lavenshtein, Mahalanobis, etc).