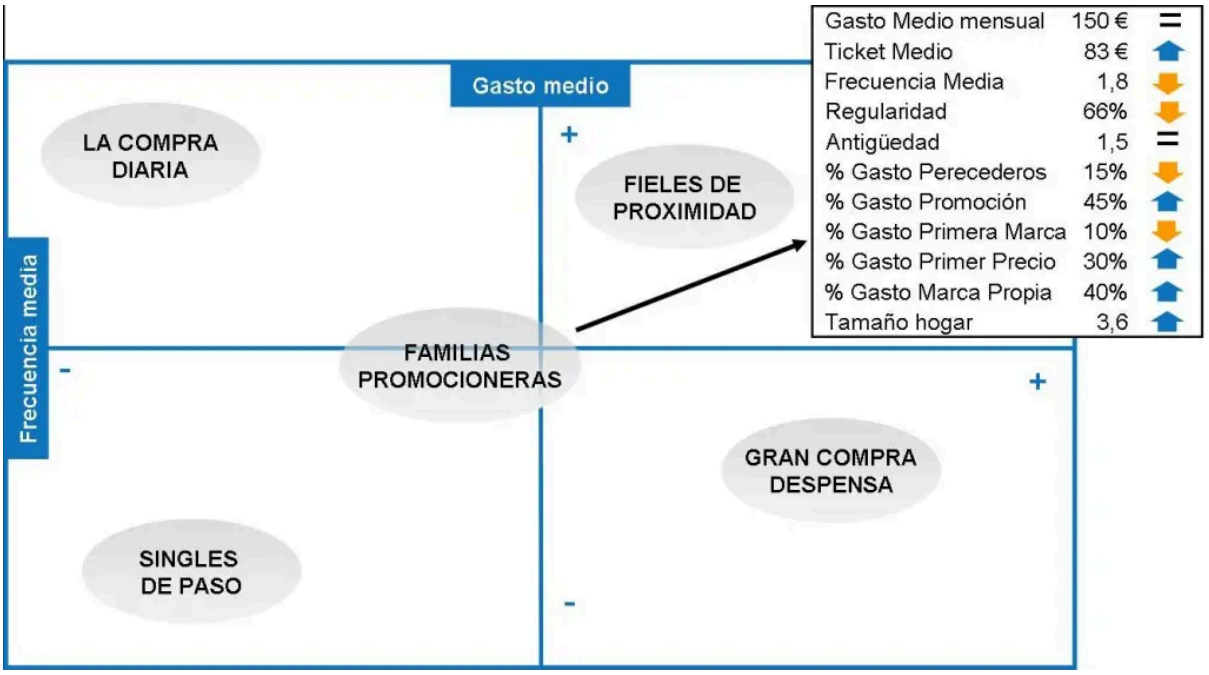
Segmentación de clientes según datos bancarios



Adrián Yared Armas de la Nuez

Contenido

1. Objetivo

El objeto de esta actividad en aplicar técnicas de clustering para obtener una serie de cluster que pueden ser utilizados para futuras campañas de Marketing.

Ahora bien, a menudo en la vida real podremos encontrarnos con conjuntos de datos relacionados con la adquisición/compra de productos y servicios, para a continuación realizar un estudio de clustering con el objeto de ver/estudiar una posible relación entre perfiles/segmentos/clusters de clientes y la adquisición de los productos/servicios.

Para esta actividad se ofrecen dos datasets, de los que has de elegir SÓLO uno de ellos según tu criterio personal.

Inicialmente se utilizarían para problemas/casos de uso de aprendizaje supervisado (Clasificación).

2. Enlaces requeridos

Título: Bank Marketing
Url: https://archive.ics.uci.edu/dataset/222/bank+marketing
Descripción: Los datos están relacionados con campañas de marketing directo (llamadas telefónicas) de una institución bancaria portuguesa. El objetivo de la clasificación es predecir si
el cliente suscribirá un depósito a plazo (variable y)

Título: Statlog (German Credit Data)
Url: https://archive.ics.uci.edu/dataset/144/statlog+german+credit+data
Descripción: Este conjunto de datos clasifica a las personas descritas según un conjunto de atributos como riesgos crediticios buenos o malos. Viene en dos formatos (uno todo numérico).

Ejemplo de referencia a considerar, por si ves algún mecanismo/utilidad/herramienta que desees incorporar a tu solución.:
• Título: Clustering for Effective Marketing Strategy
Url: https://www.kaggle.com/code/caesarmario/clustering-for-effective-marketingstrategy

3. Explicación y mejoras del código

3.1 Carga de los datos

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
# Importación de los datos
#repositorio =
'https://raw.githubusercontent.com/SalvadorBR/SNS_2324_ACT_4_1/main/ban
k%2Bmarketing/bank/bank-full.csv'
repositorio =
'https://raw.githubusercontent.com/SalvadorBR/SNS_2324_ACT_4_1/main/ban
k%2Bmarketing/bank/bank.csv'


df_clients_ori = pd.read_csv(repositorio, sep=';')
columns =df_clients_ori.columns.values
```

Este bloque carga el dataset bank.csv utilizando pandas y muestra la estructura de los datos y almacena las columnas para futuras manipulaciones o análisis.

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	c
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul	329	
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	may	153	
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug	151	
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb	129	
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr	345	

4521 rows × 17 columns

3.2 Muestra columnas

Este código muestra las columnas previamente guardadas

```
columns
```

```
array(['age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration',
       'campaign', 'pdays', 'previous', 'poutcome', 'y'], dtype=object)
```

3.3 Análisis de los datos

Conteo de los valores únicos por columna en el DataFrame, proporcionando información sobre la diversidad de datos, útil para identificar variables categóricas, numéricas y posibles problemas de datos.

```
for columna in df_clients_ori.columns:
    num_valores_unicos = df_clients_ori[columna].nunique()
    print(f'Número de valores únicos en la columna {columna}:
{num_valores_unicos}')
```

```
Número de valores únicos en la columna age: 67
Número de valores únicos en la columna job: 12
Número de valores únicos en la columna marital: 3
Número de valores únicos en la columna education: 4
Número de valores únicos en la columna default: 2
Número de valores únicos en la columna balance: 2353
Número de valores únicos en la columna housing: 2
Número de valores únicos en la columna loan: 2
Número de valores únicos en la columna contact: 3
Número de valores únicos en la columna day: 31
Número de valores únicos en la columna month: 12
Número de valores únicos en la columna duration: 875
Número de valores únicos en la columna campaign: 32
Número de valores únicos en la columna pdays: 292
Número de valores únicos en la columna previous: 24
Número de valores únicos en la columna poutcome: 4
Número de valores únicos en la columna y: 2
```

3.4 Limpieza de datos

Este paso reduce la dimensionalidad del conjunto de datos, eliminando columnas irrelevantes para el análisis de clustering, lo que mejora la eficiencia y enfoque del modelo.

```
# Elimino las columnas que considero no son interesantes para realizar
el clustering
# ['age', 'job', 'marital', 'education', 'default', 'balance',
'housing', 'loan', 'contact','day', 'month', 'duration','campaign',
'pdays', 'previous', 'poutcome', 'y']
```

```
# No voy a continuar con las siguientes columnas: 'day', 'month',
'default', 'previous', 'pdays'
columns_selected = ['age', 'job', 'marital', 'education', 'balance',
'housing', 'loan', 'contact',  'duration','campaign', 'poutcome', 'y']

df_clients = df_clients_ori[columns_selected]
df_clients
```

	age	job	marital	education	balance	housing	loan	contact	duration	campaign	poutcome	y
0	30	unemployed	married	primary	1787	no	no	cellular	79	1	unknown	no
1	33	services	married	secondary	4789	yes	yes	cellular	220	1	failure	no
2	35	management	single	tertiary	1350	yes	no	cellular	185	1	failure	no
3	30	management	married	tertiary	1476	yes	yes	unknown	199	4	unknown	no
4	59	blue-collar	married	secondary	0	yes	no	unknown	226	1	unknown	no
...	...	...	...	...	...	...	...	...	...	...	...	...
4516	33	services	married	secondary	-333	yes	no	cellular	329	5	unknown	no
4517	57	self-employed	married	tertiary	-3313	yes	yes	unknown	153	1	unknown	no
4518	57	technician	married	secondary	295	no	no	cellular	151	11	unknown	no
4519	28	blue-collar	married	secondary	1137	no	no	cellular	129	4	other	no
4520	44	entrepreneur	single	tertiary	1136	yes	yes	cellular	345	2	other	no

4521 rows × 12 columns

Como podemos observar se han eliminado 5 columnas.

3.5 Variables categóricas a numéricas

```
# Conversión de variables catgóricas a numéricas
le = LabelEncoder()

pd.options.mode.copy_on_write = True # Para que no muestre el warning

df_clients['job'] = le.fit_transform(df_clients['job'])
df_clients['marital'] = le.fit_transform(df_clients['marital'])
df_clients['education'] = le.fit_transform(df_clients['education'])
#df_clients['default'] = le.fit_transform(df_clients['default'])
df_clients['housing'] = le.fit_transform(df_clients['housing'])
df_clients['loan'] = le.fit_transform(df_clients['loan'])
df_clients['contact'] = le.fit_transform(df_clients['contact'])
```

```
df_clients['poutcome'] = le.fit_transform(df_clients['poutcome'])
#error
df_clients['y'] = le.fit_transform(df_clients['y'])
```

Convierte variables categóricas a valores numéricos utilizando LabelEncoder para cada columna especificada en el DataFrame df_clients, permitiendo su procesamiento en el modelo.

Como podemos observar en el resultado, ahora todas las variables son numéricas para un procesamiento y cruce entre las variables más sencilla:

	age	job	marital	education	balance	housing	loan	contact	duration	campaign	poutcome	y
0	30	10	1	0	1787	0	0	0	79	1	3	0
1	33	7	1	1	4789	1	1	0	220	1	0	0
2	35	4	2	2	1350	1	0	0	185	1	0	0
3	30	4	1	2	1476	1	1	2	199	4	3	0
4	59	1	1	1	0	1	0	2	226	1	3	0
...	...	...	...	...	...	...	...	...	...	...	...	...
4516	33	7	1	1	-333	1	0	0	329	5	3	0
4517	57	6	1	2	-3313	1	1	2	153	1	3	0
4518	57	9	1	1	295	0	0	0	151	11	3	0
4519	28	1	1	1	1137	0	0	0	129	4	1	0
4520	44	2	2	2	1136	1	1	0	345	2	1	0

4521 rows × 12 columns

3.6 Transformación de los datos

```
from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()
clients_scaled = min_max_scaler.fit_transform(df_clients)
clients_scaled
```

El código utiliza MinMaxScaler para normalizar las variables.
Escala los valores de cada columna a un rango entre 0 y 1, según el valor mínimo y máximo de cada característica y pone el resultado es un array de valores escalados.

```
array([[0.16176471, 0.90909091, 0.5       , ..., 0.        , 1.        ,
        0.        ],
       [0.20588235, 0.63636364, 0.5       , ..., 0.        , 0.        ,
        0.        ],
       [0.23529412, 0.36363636, 1.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.55882353, 0.81818182, 0.5       , ..., 0.20408163, 1.        ,
        0.        ],
       [0.13235294, 0.09090909, 0.5       , ..., 0.06122449, 0.33333333,
        0.        ],
       [0.36764706, 0.18181818, 1.        , ..., 0.02040816, 0.33333333,
        0.        ]])
```

3.7 Reducción dimensional PCA (Análisis de Componentes Principales)

```
from sklearn.decomposition import PCA
import numpy as np

# Reducimos la dimensionalidad de los datos (a dos dimensiones)
pca = PCA(n_components = 2)
X_pca = pca.fit_transform(clients_scaled)
# Mostramos el porcentaje de varianza explicada por cada uno de los
componentes seleccionados.
print(pca.explained_variance_ratio_)

# Visualizar la "importancia" de cada variable original del problema en
las nuevas dimensiones
pd.DataFrame(np.matrix.transpose(pca.components_), columns=['PC-1',
'PC-2'], index=df_clients.columns)
```

Este código reduce la dimensionalidad de los datos a dos componentes principales usando PCA, mostrando el porcentaje de varianza explicada y cómo cada variable contribuye a las nuevas dimensiones generadas.

```
[0.26445274 0.18531217]
```

	PC-1	PC-2
age	-0.040490	0.055506
job	-0.114728	0.022419
marital	-0.050828	-0.061621
education	-0.081631	-0.036904
balance	-0.003900	0.001125
housing	0.806814	-0.537082
loan	0.029657	-0.002243
contact	0.547678	0.715328
duration	-0.004461	-0.008826
campaign	0.001596	0.007281
poutcome	0.064045	0.427425
y	-0.141534	-0.090980

La salida que compartes muestra la importancia de cada variable original en los dos componentes principales (PC-1 y PC-2) generados por PCA.
Cada número en la tabla indica la "carga" o contribución de cada variable original a un componente específico, por ejemplo:

La variable age tiene un valor negativo en PC-1 (-0.040490) y un valor positivo en PC-2 (0.055506), lo que indica que influye de manera opuesta en ambas dimensiones.

housing tiene una carga alta en PC-1 (0.806814) y negativa en PC-2 (-0.537082), sugiriendo que esta variable tiene una fuerte correlación con la primera dimensión, pero de forma inversa con la segunda.

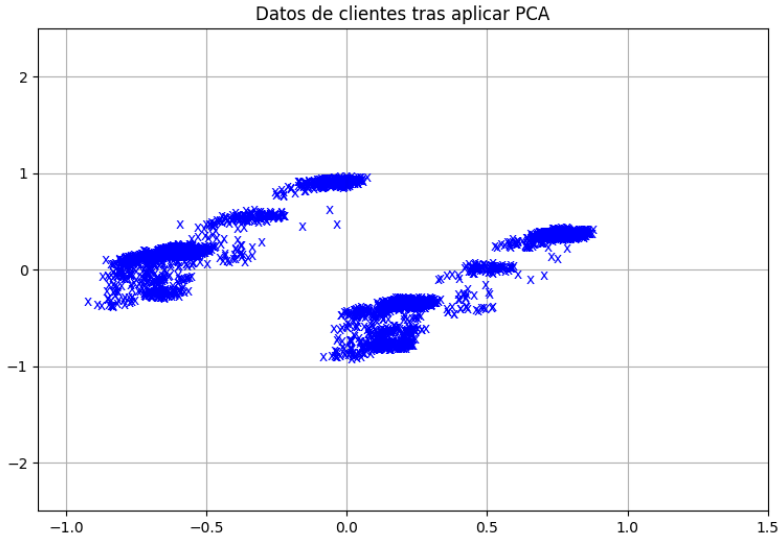
Los valores más cercanos a 0 indican que esa variable tiene poca o ninguna influencia en ese componente principal.

3.8 Graficado del PCA

```
# Visualizar el dataset utilizando las dos dimensiones obtenidas en el
PCA
import matplotlib.pyplot as plt

plt.figure(figsize=(9, 6))
for i in range(len(X_pca)):
```

```
    plt.text(X_pca[i][0], X_pca[i][1], 'x', color="b")
plt.xlim(-1.1, 1.5)
plt.ylim(-2.5, 2.5)
plt.title("Datos de clientes tras aplicar PCA")
plt.grid()
plt.show()
```

Este código grafica los datos transformados por PCA en dos dimensiones, mostrando cada punto con una 'x' azul en un espacio bidimensional para visualización.



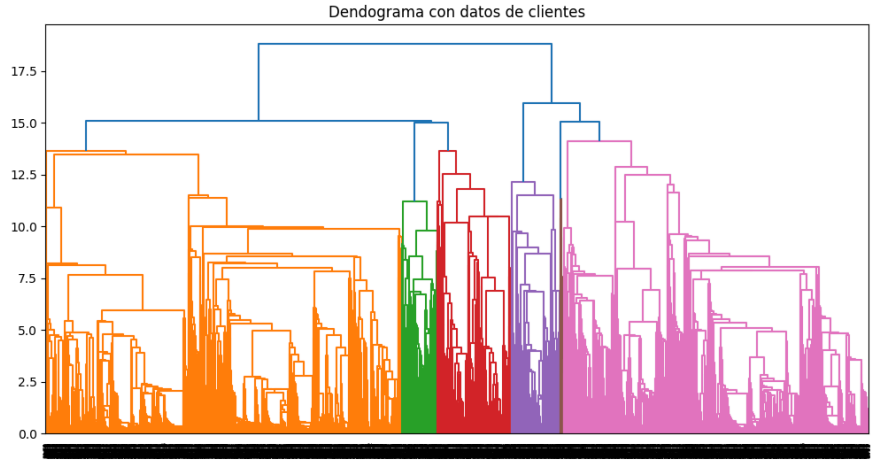
esta gráfica muestra datos de clientes proyectados en dos dimensiones, revelando agrupaciones que podrían indicar distintos perfiles o segmentos de clientes.

3.9 Distancia entre clientes

```
from sklearn.metrics import pairwise_distances
import numpy as np
import matplotlib.pyplot as plt
from scipy import cluster

# 1. Calcular la matriz de distancias
D = pairwise_distances(clients_scaled, metric='euclidean')
avD = np.average(D)
print("Distancia Media\t {:6.2f}".format(avD))

# 2. Construir dendrograma
plt.figure(figsize=(12, 6))
```

```
clusters = cluster.hierarchy.linkage(D, method='single')  # Puedes
cambiar a 'ward' o 'complete'
cluster.hierarchy.dendrogram(clusters, color_threshold=15)
plt.title("Dendograma con datos de clientes")
plt.show()
```

El código mide qué tan parecidos son los clientes y dibuja un gráfico (dendrograma) que muestra cómo se pueden agrupar según sus similitudes.



Como podemos observar el dendrograma resultante muestra cómo se agrupan los clientes según su similitud. Las líneas conectan a los clientes o grupos similares, y mientras más alto se hace el enlace, menos parecidos son. Los colores indican diferentes grupos formados al cortar el árbol en un cierto nivel. En este caso, se visualizan varios clústeres de clientes potenciales, útiles para segmentación.

3.10 Agrupación de clientes

```
# 3. Obtenemos el grupo al que pertenece cada observación
threshold = 15 # ad-hoc
labels = cluster.hierarchy.fcluster(clusters, threshold ,
criterion='distance')

# ¿Cuántos grupos hay? Contamos el número de "labels" distintas en el
vector
print("Número de clusters {}".format(len(set(labels))))
labels
```

Tras realizar el dendrograma, se asigna cada cliente a un grupo y un umbral de distancia, luego cuenta cuántos grupos distintos se formaron.

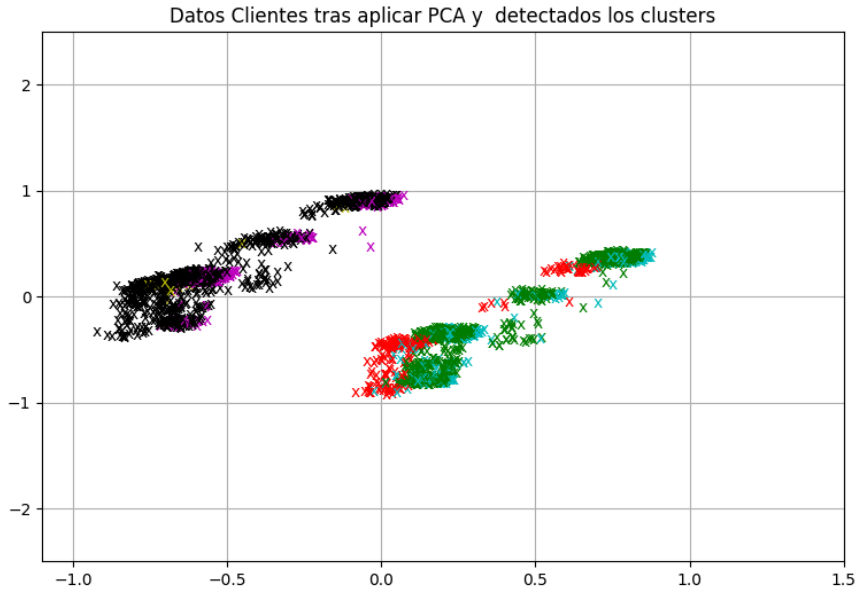
```
Número de clusters 6
array([6, 3, 1, ..., 6, 6, 3], dtype=int32)
```

En este caso se crearon 6 grupos o clústers.

3.11 Graficado PCA con los nuevos clústers

```
colores = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
colores = np.hstack([colores] * 20)

plt.figure(figsize=(9, 6))
for i in range(len(X_pca)):
    plt.text(X_pca[i][0], X_pca[i][1], 'x', color=colores[labels[i]])
plt.xlim(-1.1, 1.5)
plt.ylim(-2.5, 2.5)
plt.grid()
plt.title("Datos Clientes tras aplicar PCA y  detectados los clusters")
plt.show()
```

Este código grafica los clientes en 2D tras aplicar PCA, coloreando cada punto según el grupo (cluster) al que pertenece, para visualizar la segmentación de forma clara.



Como podemos observar, se ven claramente los diferentes grupos de clientes que previamente eran únicamente distinguibles en dos nubes azules.

3.12 Descarga en excel

Descarga en excel añadiendo el número de cluster al que pertenece:

```
import os

# Verificar que el directorio de destino existe
output_dir = '/mnt/data'
os.makedirs(output_dir, exist_ok=True)

# Ruta completa al archivo de salida
output_file = os.path.join(output_dir, 'clients_with_clusters.xlsx')

# Exportar el DataFrame con los clusters a Excel
df_clients.to_excel(output_file, index=False)

output_file
```

```
'/mnt/data/clients_with_clusters.xlsx'
```

1	age	job	marital	education	balance	housing	loan	contact	duration	campaign	poutcome	y	Cluster
2	0,705502	0,091899	0,375649	0,381939	0,492892	0,654242	0,866901	0,476133	0,015971	0,284768	0,851601	0,244728	1
3	0,800195	0,339732	0,36065	0,612744	0,965525	0,160503	0,110019	0,608884	0,737295	0,837791	0,031367	0,801016	3
4	0,768479	0,541927	0,844742	0,691741	0,611447	0,200893	0,117254	0,281259	0,073933	0,86249	0,354019	0,544587	1
5	0,752623	0,505321	0,457689	0,117679	0,370761	0,657304	0,170875	0,077355	0,493612	0,593156	0,109956	0,376799	5
6	0,918982	0,169692	0,263821	0,100124	0,950495	0,925967	0,549201	0,906113	0,310912	0,349842	0,517666	0,010931	6
7	0,883265	0,663798	0,84801	0,586148	0,945617	0,688727	0,722429	0,706934	0,807113	0,239958	0,712534	0,44949	5
8	0,06384	0,838554	0,128086	0,319698	0,768203	0,585216	0,159853	0,509018	0,219788	0,71912	0,242489	0,245731	1
9	0,540111	0,37286	0,644537	0,885311	0,146396	0,564503	0,218463	0,90921	0,81419	0,276208	0,182964	0,8676	2
10	0,187147	0,894426	0,781091	0,633746	0,800323	0,15021	0,200509	0,569273	0,087298	0,904925	0,531174	0,97053	2
11	0,109217	0,835252	0,017649	0,664378	0,318819	0,467715	0,941433	0,24078	0,273723	0,044139	0,803628	0,781374	2
12	0,482847	0,565665	0,888279	0,004729	0,593294	0,783363	0,709939	0,611293	0,708671	0,741156	0,876281	0,603708	6
13	0,63643	0,944236	0,767781	0,787313	0,186179	0,154737	0,125403	0,750209	0,562823	0,645412	0,833938	0,668811	1
14	0,749958	0,95908	0,38247	0,582649	0,627336	0,878616	0,813857	0,52506	0,466085	0,274082	0,622879	0,788978	3
15	0,269241	0,81297	0,193588	0,097738	0,077334	0,270314	0,121682	0,137978	0,533432	0,929878	0,641584	0,395099	1
16	0,751665	0,434332	0,14147	0,211898	0,378422	0,478203	0,497093	0,474484	0,517076	0,743954	0,135942	0,758866	1
17	0,524902	0,810572	0,062991	0,682226	0,987649	0,848215	0,430921	0,612668	0,252154	0,629857	0,134647	0,800624	1
18	0,844301	0,305419	0,087298	0,684256	0,594845	0,47477	0,872491	0,952733	0,157278	0,655468	0,499263	0,392976	6
19	0,527808	0,951539	0,234379	0,056538	0,15141	0,989059	0,92736	0,209265	0,981462	0,592018	0,855733	0,68879	6
20	0,848442	0,661439	0,553188	0,39171	0,711854	0,165607	0,923359	0,743663	0,520603	0,050474	0,637457	0,775765	2

4. Mejora de la actividad y correcciones

He planteado dos apartados que podrían mejorar la actividad, así como corregí un campo que estaba erróneo en la actividad.

4.1 Corrección del error

En el grupo de código:

```
# Conversión de variables catgóricas a numéricas
le = LabelEncoder()

pd.options.mode.copy_on_write = True # Para que no muestre el warning

df_clients['job'] = le.fit_transform(df_clients['job'])
df_clients['marital'] = le.fit_transform(df_clients['marital'])
df_clients['education'] = le.fit_transform(df_clients['education'])
#df_clients['default'] = le.fit_transform(df_clients['default'])
df_clients['housing'] = le.fit_transform(df_clients['housing'])
df_clients['loan'] = le.fit_transform(df_clients['loan'])
df_clients['contact'] = le.fit_transform(df_clients['contact'])
df_clients['poutcome'] = le.fit_transform(df_clients['poutcome'])
df_clients['y'] = le.fit_transform(df_clients['y'])
```

Había un error con la línea putoutcome:

```
df_clients['poutcome'] = le.fit_transform(df_clients['poutcome'])
```

Ya que previamente estaba codificando putoutcome con los valores de martial:

```
df_clients['poutcome'] = le.fit_transform(df_clients['marital'])
```

4.2 Calidad del clustering

Consideré interesante añadir una manera de medir la calidad del clustering para tener una mejor conclusión sobre la utilidad y fiabilidad de lo obtenido.

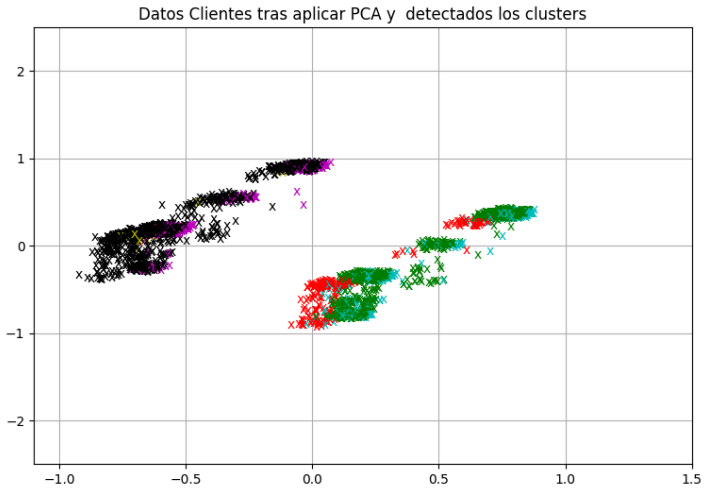
Este código normaliza los datos, aplica clustering jerárquico y evalúa la calidad de los grupos usando la puntuación de silueta, que mide qué tan bien están separados los clusters.

```
from sklearn.metrics import silhouette_score
from sklearn import preprocessing
from scipy import cluster

# Normalización
scaler = preprocessing.MinMaxScaler()
clients_scaled = scaler.fit_transform(df_clients)
```

```
# Clustering jerárquico
linkage_matrix = cluster.hierarchy.linkage(clients_scaled,
method='ward')
labels = cluster.hierarchy.fcluster(linkage_matrix, t=15,
criterion='distance')

# Silhouette
silhouette = silhouette_score(clients_scaled, labels)
print(f"Puntuación de silueta: {silhouette:.2f}")
```

```
Puntuación de silueta: 0.32
```

La puntuación de silueta de 0.32 indica que los clusters tienen una separación moderada, pero no muy clara. Por lo que pueden haber muchos valores cercanos a los bordes entre los clúster como podemos observar en la imagen de la nube previa:



4.3 Análisis de los clústers obtenidos

También consideré una mejora asignar a cada cliente su grupo (cluster) y luego calcula el promedio de cada variable por cluster, lo que permite entender las características típicas de cada grupo. Así puedes comparar, por ejemplo, si un grupo gasta más, es más joven, etc.

Todo esto provee un mayor nivel de uso de los datos y calidad de tratamiento de estos.

```
df_clients['Cluster'] = labels
print(df_clients.groupby('Cluster').mean())
```

Cluster	age	job	marital	education	balance	housing
1	45.031802	5.212014	1.190813	1.360424	1788.915194	0.000000
2	39.470588	4.000000	1.151261	1.273109	1313.974790	0.924370
3	45.798942	4.701058	1.097884	1.195767	1793.669312	0.000000
4	42.169291	5.043307	1.213583	1.376969	1654.561024	0.000000
5	43.910931	4.319838	1.056680	1.153846	934.287449	0.000000
6	39.204489	4.039900	1.097257	1.117207	922.476309	0.950125
7	39.860165	3.903643	1.101058	1.090482	1359.856639	1.000000
8	39.436314	4.032520	1.192412	1.216802	1577.615176	1.000000
9	39.018970	4.224932	1.154472	1.243902	1238.138211	1.000000

Cluster	loan	contact	duration	campaign	poutcome	y
1	0.000000	0.236749	477.332155	2.159011	2.374558	1.0
2	0.180672	0.415966	642.411765	2.394958	2.281513	1.0
3	0.000000	1.759259	192.806878	2.804233	2.986772	0.0
4	0.000000	0.024606	224.421260	2.977362	2.588583	0.0
5	1.000000	0.473684	230.680162	2.935223	2.995951	0.0
6	1.000000	0.765586	227.588529	2.932668	2.413965	0.0
7	0.000000	1.917744	237.163337	2.988249	2.997650	0.0
8	0.000000	0.078591	228.092141	2.113821	0.346883	0.0
9	0.000000	0.010840	230.710027	2.899729	2.993225	0.0

Los clusters muestran diferencias en edad, balance bancario, ocupación, y respuesta a campañas y podemos extraer datos como que los clústers 1 y 2 tienen clientes más jóvenes, financieramente estables y propensos a aceptar ofertas, mientras que los demás muestran menos interés y menor aceptación.

5. Conclusiones

En el análisis de los datos de clientes bancarios, se aplicó un proceso de clustering jerárquico para identificar grupos. Utilizando PCA para reducir la dimensionalidad, se identificaron 6 clústeres con características distintivas en variables como edad, balance y comportamiento con respecto a préstamos. La calidad del clustering, evaluada mediante el índice de silueta, mostró una puntuación moderada de 0.32, lo que indica que los clústeres son razonablemente coherentes pero podrían mejorarse. Este proceso permite aprender sobre los patrones de comportamiento de los clientes, lo que podría ser útil para personalizar estrategias de marketing y optimizar recursos.

6. Github y Colab