Universidad Nacional Autónoma de México

Facultad de Ingeniería

PRÁCTICA 8

CLUSTERING PARTICIONAL

(SEGMENTACIÓN DE CLIENTES)

Minería de Datos

Profesor:

Dr. Molero Castillo Guillermo Gilberto

Grupo 1

Alumna:

Monroy Velázquez Alejandra Sarahí

No. Cuenta: 314000417

OBJETIVO

Obtener clústeres de casos de usuarios, con características similares,evaluados para la adquisición de una casa a través de un crédito hipotecario con tasa fija 30 años.

DESARROLLO

El conjunto de datos corresponde a diversas variables relacionadas a las hipotecas. Este conjunto de datos incluye:

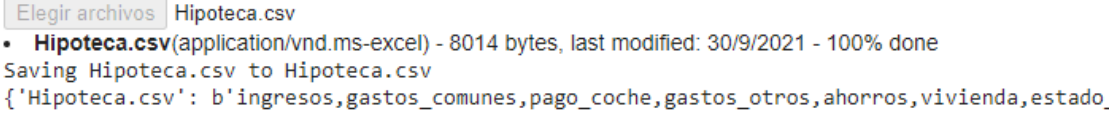
- ingresos: son ingresos mensuales de 1 o 2 personas, si están casados.
- gastos_comunes: son gastos mensuales de 1 o 2 personas, si están casados.
- pago_coche
- gastos_otros
- ahorros
- vivienda: valor de la vivienda.
- estado_civil: 0-soltero, 1-casado, 2-divorciado
- hijos: cantidad de hijos menores (no trabajan).
- trabajo: 0-sin trabajo, 1-autonomo, 2-asalariado, 3-empresario, 4-autonomos, 5-asalariados, 6-autonomo y asalariado, 7-empresario y autonomo, 8-empresarios o empresario y autónomo
- comprar: 0-alquilar, 1-comprar casa a través de crédito hipotecario con tasa fija a 30 años.

Primero comenzamos la importación de bibliotecas correspondientes que nos ayudarán para la realización del código, las cuales son *pandas* para la manipulación y análisis de datos, *numpy* para crear vectores y matrices, *matplotlib* para la generación de gráficas, así como *seaborn* para la visualización de datos. En esta práctica agregamos una biblioteca más, la cual es *scipy* que nos ayudara para el cálculo de distancias. Por último, la biblioteca files para subir el archivo *csv*.

Una vez importadas, el *dataframe* se lee y se despliega en pantalla:

```
[1]  import pandas as pd                # Para la manipulación y análisis de datos
     import numpy as np                 # Para crear vectores y matrices n dimensionales
     import matplotlib.pyplot as plt    # Para la generación de gráficas a partir de los datos
     import seaborn as sns              # Para la visualización de datos basado en matplotlib
     %matplotlib inline
```

```
[2]  from google.colab import files
     files.upload()
```



```
Hipoteca = pd.read_csv("Hipoteca.csv")
Hipoteca
```

	ingresos	gastos_comunes	pago_coche	gastos_otros	ahorros	vivienda	estado_civil	hijos	trabajo	comprar
0	6000	1000	0	600	50000	400000	0	2	2	1
1	6745	944	123	429	43240	636897	1	3	6	0
2	6455	1033	98	795	57463	321779	2	1	8	1
3	7098	1278	15	254	54506	660933	0	0	3	0
4	6167	863	223	520	41512	348932	0	0	3	1
...	...	...	...	...	...	...	...	...	...	...
197	3831	690	352	488	10723	363120	0	0	2	0
198	3961	1030	270	475	21880	280421	2	3	8	0
199	3184	955	276	684	35565	388025	1	3	8	0
200	3334	867	369	652	19985	376892	1	2	5	0
201	3988	1157	105	382	11980	257580	0	0	4	0

202 rows × 10 columns

Ahora que el *dataframe* está cargado, comenzamos con los pasos vistos en clase:

Para observar si existen datos faltantes utilizamos *info()*, por lo que observamos que todas las variables se encuentran integras, y además todas son de tipo entero.

```
[4]  Hipoteca.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202 entries, 0 to 201
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   ingresos        202 non-null    int64
 1   gastos_comunes  202 non-null    int64
 2   pago_coche      202 non-null    int64
 3   gastos_otros    202 non-null    int64
 4   ahorros         202 non-null    int64
 5   vivienda        202 non-null    int64
 6   estado_civil    202 non-null    int64
 7   hijos           202 non-null    int64
 8   trabajo         202 non-null    int64
 9   comprar         202 non-null    int64
dtypes: int64(10)
memory usage: 15.9 KB
```

Ahora observamos la variable *comprar* ya que esta representa un valor obtenido de un análisis preliminar. Lo que observamos de acuerdo con el resultado obtenido es que 135 personas podrían ser no acreedoras del crédito, mientras que 67 sí.

```
[5]  print(Hipoteca.groupby('comprar').size())
```

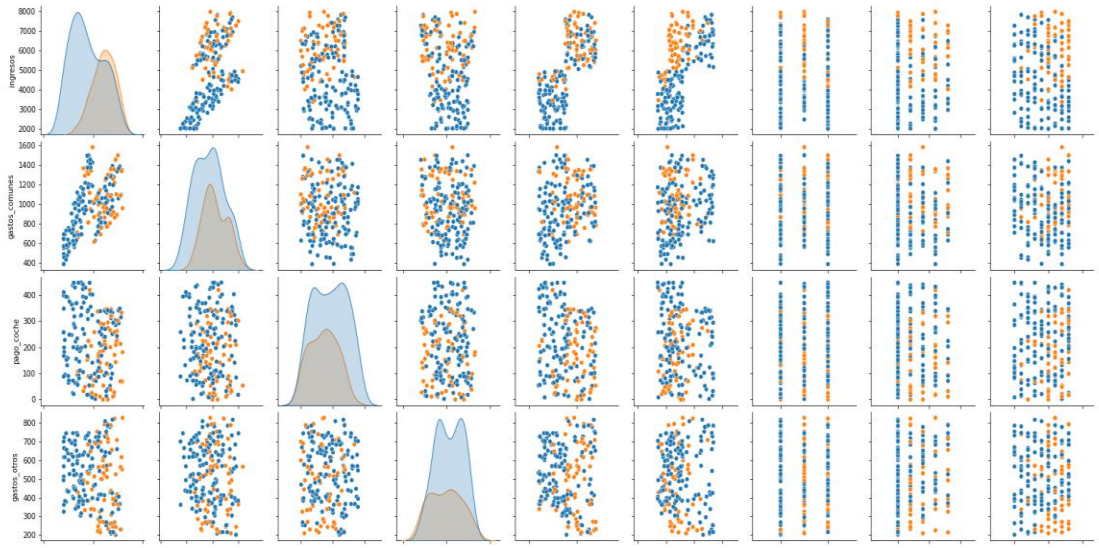
```
comprar
0    135
1     67
dtype: int64
```

Selección de Características

a) *Evaluación visual*

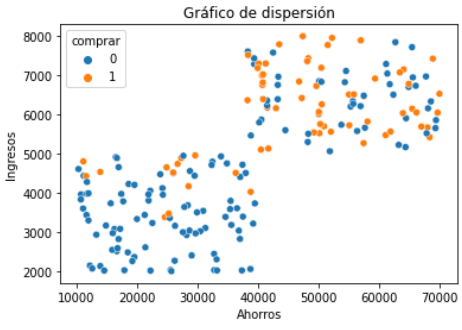
Como parte del proceso de selección de características realizamos una evaluación visual para observar las gráficas de los pares de variables e identificar las posibles correlaciones entre las mismas:

```
[7]  sns.pairplot(Hipoteca, hue='comprar')
     plt.show()
```



Así como también podemos graficar solamente un par de variables para observarla de una mejor manera y con mayor detalle, en este caso graficamos solo *ahorros* vs *ingresos*:

```
[9]  sns.scatterplot(x='ahorros', y ='ingresos', data=Hipoteca, hue='comprar')
     plt.title('Gráfico de dispersión')
     plt.xlabel('Ahorros')
     plt.ylabel('Ingresos')
     plt.show()
```



b) *Matriz de correlaciones*

Generamos la matriz de correlaciones, utilizando la correlación de Pearson:

```
[10] CorrHipoteca = Hipoteca.corr(method='pearson')
     CorrHipoteca
```

	ingresos	gastos_comunes	pago_coche	gastos_otros	ahorros	vivienda	estado_civil	hijos	trabajo	comprar
ingresos	1.000000	0.560211	-0.109780	-0.124105	0.712889	0.614721	-0.042556	-0.024483	-0.038852	0.467123
gastos_comunes	0.560211	1.000000	-0.054400	-0.099881	0.209414	0.204781	-0.057152	-0.072321	-0.079095	0.200191
pago_coche	-0.109780	-0.054400	1.000000	0.010602	-0.193299	-0.094631	0.052239	-0.044858	0.018946	-0.196468
gastos_otros	-0.124105	-0.099881	0.010602	1.000000	-0.064384	-0.054577	-0.020226	0.124845	0.047313	-0.110330
ahorros	0.712889	0.209414	-0.193299	-0.064384	1.000000	0.605836	-0.063039	0.001445	-0.023829	0.340778
vivienda	0.614721	0.204781	-0.094631	-0.054577	0.605836	1.000000	-0.113420	-0.141924	-0.211790	-0.146092
estado_civil	-0.042556	-0.057152	0.052239	-0.020226	-0.063039	-0.113420	1.000000	0.507609	0.589512	0.142799
hijos	-0.024483	-0.072321	-0.044858	0.124845	0.001445	-0.141924	0.507609	1.000000	0.699916	0.272883
trabajo	-0.038852	-0.079095	0.018946	0.047313	-0.023829	-0.211790	0.589512	0.699916	1.000000	0.341537
comprar	0.467123	0.200191	-0.196468	-0.110330	0.340778	-0.146092	0.142799	0.272883	0.341537	1.000000

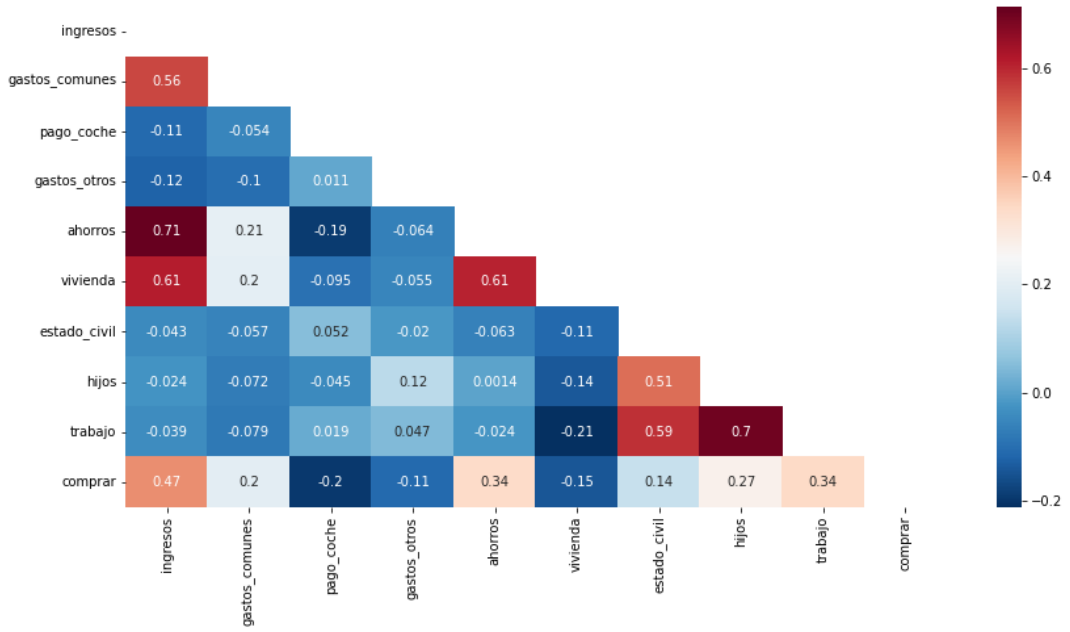
Para observar de una mejor forma las correlaciones con respecto a la variable *ingresos* , imprimimos y ordenamos las correlaciones de forma ascendente. Vemos que la variable *ahorros* tiene un coeficiente de correlación muy alto.

```
[11] print(CorrHipoteca['ingresos'].sort_values(ascending=False)[:10], '\n')   #Top 10 valores
```

```
ingresos          1.000000
ahorros           0.712889
vivienda          0.614721
gastos_comunes    0.560211
comprar           0.467123
hijos            -0.024483
trabajo          -0.038852
estado_civil     -0.042556
pago_coche       -0.109780
gastos_otros     -0.124105
Name: ingresos, dtype: float64
```

Generamos un mapa de calor para ubicar e identificar más fácilmente las correlaciones entre las variables:

```
plt.figure(figsize=(14,7))
MatrizInf = np.triu(CorrHipoteca)
sns.heatmap(CorrHipoteca, cmap='RdBu_r', annot=True, mask=MatrizInf)
plt.show()
```



A pesar de existir 2 correlaciones altas, entre 'ingresos' y 'ahorros' (0.71) y 'trabajo' e 'hijos' (0.69); éstas se tomarán en cuenta para obtener una segmentación que combine las variables mediante la similitud de los elementos. Se suprimirá la variable 'comprar' debido a que representa inherentemente un agrupamiento, y fue un campo calculado con base a un análisis hipotecario preliminar.

c) *Elección de variables*

Eliminamos la variable comprar y guardamos las variables restantes en *MatrizHipoteca*.

```
[13] MatrizHipoteca = np.array(Hipoteca[['ingresos', 'gastos_comunes', 'pago_coche', 'gastos_otros', 'ahorros', 'vivienda', 'estado_civil', 'hijos', 'trabajo']])
     pd.DataFrame(MatrizHipoteca)
     #MatrizHipoteca = Hipoteca.iloc[:, 0:9].values     #iloc para seleccionar filas y columnas según su posición
```

	0	1	2	3	4	5	6	7	8
0	6000	1000	0	600	50000	400000	0	2	2
1	6745	944	123	429	43240	636897	1	3	6
2	6455	1033	98	795	57463	321779	2	1	8
3	7098	1278	15	254	54506	660933	0	0	3
4	6167	863	223	520	41512	348932	0	0	3
...	...	...	...	...	...	...	...	...	...
197	3831	690	352	488	10723	363120	0	0	2
198	3961	1030	270	475	21880	280421	2	3	8
199	3184	955	276	684	35565	388025	1	3	8
200	3334	867	369	652	19985	376892	1	2	5
201	3988	1157	105	382	11980	257580	0	0	4

202 rows × 9 columns

Aplicación del algoritmo: K-means

Como primer paso estandarizamos o normalizamos el rango de las variables iniciales, para que cada una de ellas contribuya por igual al análisis. Para ello instanciamos el objeto StandardScaker, calculamos media y desviación estándar de cada variable y escalamos los datos. Guardamos en una nueva variable llamada *MEstandarizada*.

```
[14] from sklearn.preprocessing import StandardScaler, MinMaxScaler
     estandarizar = StandardScaler()                          # Se instancia el objeto StandardScaler o MinMaxScaler
     MEstandarizada = estandarizar.fit_transform(MatrizHipoteca)   # Se calculan la media y desviación y se escalan los datos
```

```
[15] pd.DataFrame(MEstandarizada)
```

	0	1	2	3	4	5	6	7	8
0	0.620129	0.104689	-1.698954	0.504359	0.649475	0.195910	-1.227088	0.562374	-0.984420
1	1.063927	-0.101625	-0.712042	-0.515401	0.259224	1.937370	-0.029640	1.295273	0.596915
2	0.891173	0.226266	-0.912634	1.667244	1.080309	-0.379102	1.167809	-0.170526	1.387582
3	1.274209	1.128886	-1.578599	-1.559015	0.909604	2.114062	-1.227088	-0.903426	-0.589086
4	0.719611	-0.400042	0.090326	0.027279	0.159468	-0.179497	-1.227088	-0.903426	-0.589086
...	...	...	...	...	...	...	...	...	...
197	-0.671949	-1.037402	1.125381	-0.163554	-1.617963	-0.075199	-1.227088	-0.903426	-0.984420
198	-0.594508	0.215214	0.467439	-0.241079	-0.973876	-0.683130	1.167809	1.295273	1.387582
199	-1.057368	-0.061099	0.515581	1.005294	-0.183849	0.107880	-0.029640	1.295273	1.387582
200	-0.968013	-0.385305	1.261783	0.814462	-1.083273	0.026040	-0.029640	0.562374	0.201581
201	-0.578424	0.683102	-0.856468	-0.795686	-1.545397	-0.851037	-1.227088	-0.903426	-0.193753

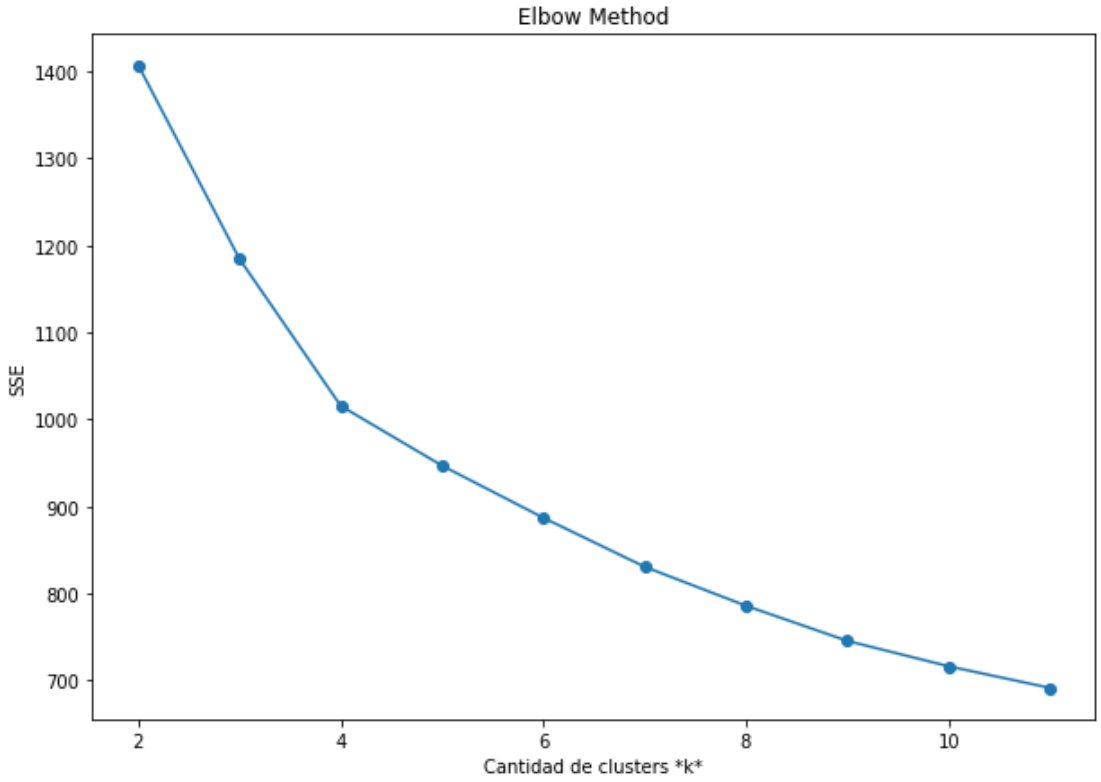
202 rows × 9 columns

Importamos la biblioteca *sklearn* para aplicar el método del codo, se realiza el algoritmo como a continuación se presenta y se genera una gráfica para identificar los clústeres:

```
[14] #Se importan las bibliotecas
     from sklearn.cluster import KMeans
     from sklearn.metrics import pairwise_distances_argmin_min #estimacion de las distancias
```

```
[15] #Definición de k clusters para K-means
     #Se utiliza random_state para inicializar el generador interno de números aleatorios
     SSE = []
     for i in range(2, 12):
         km = KMeans(n_clusters=i, random_state=0)
         km.fit(MEstandarizada)
         SSE.append(km.inertia_)

     #Se grafica SSE en función de k
     plt.figure(figsize=(10, 7))
     plt.plot(range(2, 12), SSE, marker='o')
     plt.xlabel('Cantidad de clusters *k*')
     plt.ylabel('SSE')
     plt.title('Elbow Method')
     plt.show()
```



En la práctica, puede que no exista un codo afilado (codo agudo) y, como método heurístico, ese "codo" no siempre puede identificarse sin ambigüedades.

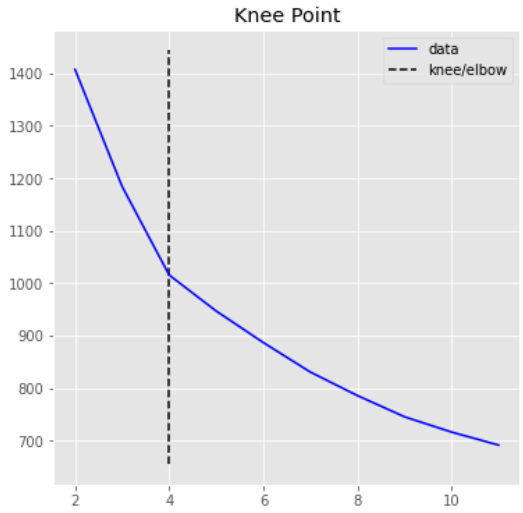
Lo siguiente será instalar la biblioteca *kneed* para ayudar a identificar de forma más exacta ese codo afilado,

```
[16] !pip install kneed
```

```
[17] from kneed import KneeLocator
     kl = KneeLocator(range(2, 12), SSE, curve="convex", direction="decreasing")
     kl.elbow
```

Y graficamos, lo que vemos será una línea vertical que nos indica en donde se encuentra el codo:

```
[18] plt.style.use('ggplot')
     kl.plot_knee()
```



Creamos las etiquetas de los elementos en los clústeres:

```
[19] #Se crean las etiquetas de los elementos en los clusters
     MParticional = KMeans(n_clusters=4, random_state=0).fit(MEstandarizada)
     MParticional.predict(MEstandarizada)
     MParticional.labels_
```

```
array([0, 2, 2, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 0, 2,
       0, 2, 0, 0, 2, 0, 0, 0, 2, 2, 0, 2, 0, 0, 2, 0, 2, 2, 2, 0, 2, 2, 2,
       0, 0, 3, 2, 2, 0, 1, 1, 1, 1, 3, 1, 3, 3, 3, 3, 1, 1, 3, 1, 3, 1,
       1, 3, 1, 3, 1, 1, 1, 1, 3, 1, 3, 1, 1, 3, 1, 0, 3, 3, 1, 1, 3, 1,
       1, 3, 3, 1, 1, 3, 3, 1, 3, 3, 1, 3, 1, 2, 0, 2, 2, 0, 0, 2, 0, 2,
       2, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 0,
       0, 0, 0, 0, 0, 2, 2, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 3, 1, 3,
       0, 3, 0, 1, 1, 3, 1, 1, 1, 1, 3, 1, 1, 3, 1, 1, 1, 3, 3, 1, 3, 1,
       3, 3, 1, 3, 1, 1, 1, 1, 3, 1, 3, 1, 0, 3, 1, 3, 3, 1, 1, 1, 3, 3,
       1, 1, 1, 3], dtype=int32)
```

E imprimimos los datos, ahora con una nueva columna *clusterH* que nos indica a que clúster pertenece ese registro, además de borrar la columna *comprar* la cual habíamos analizado en la elección de variables:

```
[20] Hipoteca = Hipoteca.drop(columns=['comprar'])
     Hipoteca['clusterP'] = MParticional.labels_
     Hipoteca
```

	ingresos	gastos_comunes	pago_coche	gastos_otros	ahorros	vivienda	estado_civil	hijos	trabajo	clusterP
0	6000	1000	0	600	50000	400000	0	2	2	0
1	6745	944	123	429	43240	636897	1	3	6	2
2	6455	1033	98	795	57463	321779	2	1	8	2
3	7098	1278	15	254	54506	660933	0	0	3	0
4	6167	863	223	520	41512	348932	0	0	3	0
...	...	...	...	...	...	...	...	...	...	...
197	3831	690	352	488	10723	363120	0	0	2	3
198	3961	1030	270	475	21880	280421	2	3	8	1
199	3184	955	276	684	35565	388025	1	3	8	1
200	3334	867	369	652	19985	376892	1	2	5	1
201	3988	1157	105	382	11980	257580	0	0	4	3

202 rows × 10 columns

Utilizamos la función *groupby()* y *count()* para observar cuantos elementos tiene cada clúster. El clúster 0 tiene 49, el 1 56, el 2 tiene 54, y el 3 tiene 43.

```
[21] #Cantidad de elementos en los clusters
     Hipoteca.groupby(['clusterP'])['clusterP'].count()
```

```
clusterP
0    49
1    56
2    54
3    43
Name: clusterP, dtype: int64
```

Para poder observar los elementos que conforman cada clúster hacemos uso de la siguiente línea de código, en el ejemplo estamos consultando los elementos del clúster 0:

```
[22] Hipoteca[Hipoteca.clusterP == 0]
```

34	7349	1102	221	402	48110	346403	0	0	3	0
36	7276	1455	285	384	61125	443008	0	0	0	0
40	6822	1296	81	786	50433	669054	0	0	0	0
44	6952	1043	251	319	43326	545659	0	0	3	0
45	5515	993	156	247	50121	286808	1	0	4	0
49	6959	1392	333	818	67714	571076	0	0	3	0
81	4927	1429	141	398	33863	279323	0	0	3	0
102	6332	1266	32	252	55641	665167	1	0	3	0
105	7420	1336	243	660	39328	655720	2	0	4	0
106	6205	1179	240	729	56904	661009	0	0	2	0
108	6813	1090	312	208	54305	635848	0	0	2	0
111	7595	1367	292	406	38286	608743	2	0	3	0
112	6355	953	135	740	38235	343128	0	0	1	0
113	5862	762	249	334	40529	443477	0	0	4	0

Ahora creamos una matriz donde podamos observar el promedio de cada variable en cada clúster, es decir, los centroides:

```
[23] CentroidesP = Hipoteca.groupby('clusterP').mean()
     CentroidesP
```

clusterP	ingresos	gastos_comunes	pago_coche	gastos_otros	ahorros	vivienda	estado_civil	hijos	trabajo
0	6358.959184	1117.306122	190.755102	465.653061	50687.081633	497262.265306	0.448980	0.061224	2.122449
1	3472.482143	905.607143	224.732143	536.589286	23957.642857	272010.535714	1.625000	2.250000	6.660714
2	6389.685185	998.851852	190.203704	524.148148	54899.722222	430860.092593	1.462963	2.222222	6.296296
3	3502.930233	857.209302	245.790698	533.627907	24129.139535	291900.953488	0.348837	0.000000	2.093023

Interpretación

A la tabla que sacamos hay que analizarla y deducir una interpretación:

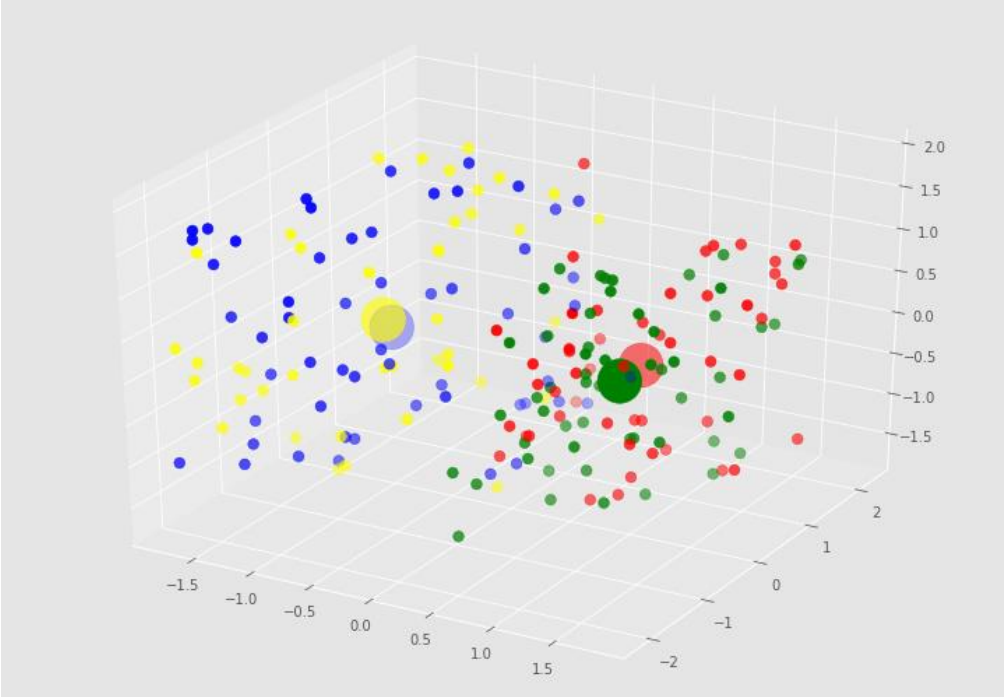
- **Clúster 0:** Conformado por 49 casos de una evaluación hipotecaría, con un ingreso promedio mensual de 6358 USD, con gastos comunes de 1117 USD, otros gastos de 465 USD y un pago mensual de coche de 190 USD. Estos gastos en promedio representan menos de la tercera parte del salario mensual (1772 USD). Por otro lado, este grupo de usuarios tienen un ahorro promedio de 50687 USD, y un valor promedio de vivienda (a comprar o hipotecar) de 497262 USD. Además, en su mayoría son solteros (0-soltero), casi sin hijos menores y tienen un tipo de trabajo, en su mayoría, asalariado (2-asalariado).
- **Clúster 1:** Conformado por 56 usuarios, con un ingreso promedio mensual de 3472 USD, con gastos comunes de 905 USD, otros gastos de 536 USD y un pago mensual de coche de 224 USD. Estos gastos en promedio (1665 USD) representan casi la mitad de salario mensual. Por otro lado, este grupo de usuarios tienen un ahorro promedio de 23957 USD, y un valor promedio de vivienda (a comprar o hipotecar) de 272010 USD. Además, en su mayoría son divorciados (2-divorciados), con 1 y 3 hijos y tienen un tipo de trabajo empresarios o empresario y autónomo (5-asalariado).
- **Clúster 2:** Conformado por 54 usuarios, con un ingreso promedio mensual de 6389 USD, con gastos comunes de 998 USD, otros gastos de 524 USD y un pago mensual de coche de 190 USD. Estos gastos en promedio (1712 USD) representan poco más de un cuarto de salario mensual. Por otro lado, este grupo de usuarios tienen un ahorro promedio de 54899 USD, y un valor promedio de vivienda (a comprar o hipotecar) de 430860 USD. Además, en su mayoría son casados (1-casado), con 1 hijo y tienen un tipo de trabajo asalariado (5-asalariado).
- **Clúster 3:** Conformado por 43 usuarios, con un ingreso promedio mensual de 3502 USD, con gastos comunes de 857 USD, otros gastos de 533 USD y un pago mensual de coche de 245 USD. Estos gastos en promedio representan casi la mitad del salario mensual (1635 USD). Por otro lado, este grupo de usuarios tienen un ahorro promedio de 24129 USD, y un valor promedio de vivienda (a

comprar o hipotecar) de 291900 USD. Además, en su mayoría son solteros (0-soltero), sin hijos y tienen un tipo de trabajo asalariado (2-asalariado).

Por último, generamos una gráfica que nos muestra la distribución de los clúster en el espacio, marcado con colores:

```
[24] # Gráfica de los elementos y los centros de los clusters
     from mpl_toolkits.mplot3d import Axes3D
     plt.rcParams['figure.figsize'] = (10, 7)
     plt.style.use('ggplot')
     colores=['red', 'blue', 'green', 'yellow']
     asignar=[]
     for row in MParticional.labels_:
         asignar.append(colores[row])

     fig = plt.figure()
     ax = Axes3D(fig)
     ax.scatter(MEstandarizada[:, 0],
                MEstandarizada[:, 1],
                MEstandarizada[:, 2], marker='o', c=asignar, s=60)
     ax.scatter(MParticional.cluster_centers_[:, 0],
                MParticional.cluster_centers_[:, 1],
                MParticional.cluster_centers_[:, 2], marker='o', c=colores, s=1000)
     plt.show()
```



CONCLUSIÓN

A diferencia de la practica numero siete, para realizar el clustering aplicamos un algoritmo diferente, mientras que en la otra fue jerárquico, en este fue particional, por lo que aplicamos el método del codo para generar el número de clústeres, en esencia los pasos son los mismos solo esta parte fue la que cambio. Como dijimos anteriormente hacer un análisis de clústeres nos ayuda a segmentar clientes y poder deducir y/o interpretar los datos a partir del calculo de los centroides en cada clúster.