Universidad Nacional Autónoma de México

Facultad de Ingeniería

PRÁCTICA 9

CLUSTERING JERÁRQUICO & PARTICIONAL

Minería de Datos

Profesor:

Dr. Molero Castillo Guillermo Gilberto

Grupo 1

Alumna:

Monroy Velázquez Alejandra Sarahí

No. Cuenta: 314000417

OBJETIVO

Obtener grupos de pacientes con características similares, diagnosticadas con un tumor de mama, a través de clustering jerárquico y particional.

DESARROLLO

El conjunto de datos corresponde a estudios clínicos a partir de imágenes digitalizadas de pacientes con cáncer de mama de Wisconsin (WDBC, Wisconsin Diagnostic Breast Cancer). La fuente de datos incluye:

Variable	Descripción	Tipo
ID number	Identifica al paciente	Discreto
Diagnosis	Diagnostico (M=maligno, B=benigno)	Booleano
Radius	Media de las distancias del centro y puntos del perímetro	Continuo
Texture	Desviación estándar de la escala de grises	Continuo
Perimeter	Valor del perímetro del cáncer de mama	Continuo
Area	Valor del área del cáncer de mama	Continuo
Smoothness	Variación de la longitud del radio	Continuo
Compactness	Perímetro ^ 2 /Area - 1	Continuo
Concavity	Caída o gravedad de las curvas de nivel	Continuo
Concave points	Número de sectores de contorno cóncavo	Continuo
Symmetry	Simetría de la imagen	Continuo
Fractal dimension	"Aproximación de frontera" - 1	Continuo

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()
```

```
Elegir archivos  OriginalWDBC.txt
• OriginalWDBC.txt(text/plain) - 45410 bytes, last modified: 30/9/2021 - 100% done
Saving OriginalWDBC.txt to OriginalWDBC.txt
{'OriginalWDBC.txt': b'Identificador\tDiagnosis\tRadius\tTexture\tPerimeter\tArea\tSmoothi
```

```
[3] BCancer = pd.read_table("OriginalWDBC.txt")
    BCancer
```

	Identificador	Diagnosis	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension
0	P-842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871
1	P-842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667
2	P-84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999
3	P-84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744
4	P-84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883
...	...	...	...	...	...	...	...	...	...	...	...	...
564	P-926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623
565	P-926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533
566	P-926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648
567	P-927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016
568	P-92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884

569 rows × 12 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 flotante.

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Identificador     569 non-null    object
 1   Diagnosis         569 non-null    object
 2   Radius            569 non-null    float64
 3   Texture           569 non-null    float64
 4   Perimeter         569 non-null    float64
 5   Area              569 non-null    float64
 6   Smoothness        569 non-null    float64
 7   Compactness       569 non-null    float64
 8   Concavity         569 non-null    float64
 9   ConcavePoints     569 non-null    float64
 10  Symmetry          569 non-null    float64
 11  FractalDimension  569 non-null    float64
dtypes: float64(10), object(2)
memory usage: 53.5+ 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())
```

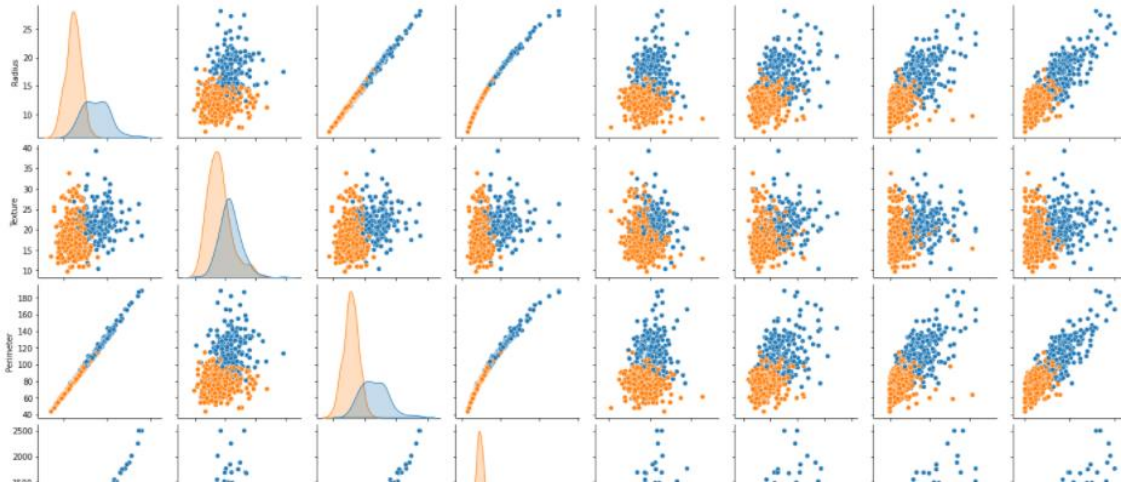
```
comprar
0    135
1     67
dtype: int64
```

Selección de Características

a) *Evaluación visual*

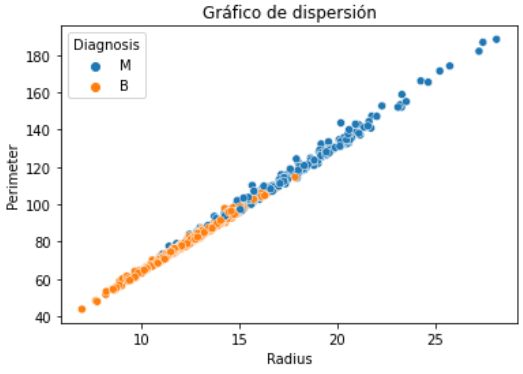
Como parte del proceso de selección de características realizamos una evaluación visual para observar las gráficas de los pares de variables e identificar las posibles correlaciones entre las mismas:

```
[5]  sns.pairplot(BCancer, hue="Diagnosis")
     plt.show()
```

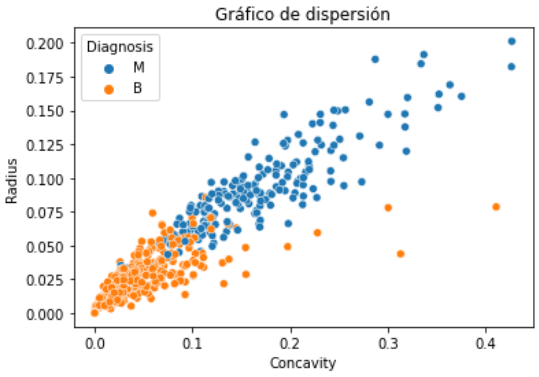


Así como también podemos graficar solamente un par de variables para observarla de una mejor manera y con mayor detalle, en este caso graficamos solo *Radius* vs Perimeter y *Cocavity* vs *Radius*:

```
[6]  sns.scatterplot(x='Radius', y ='Perimeter', data=BCancer, hue='Diagnosis')
     plt.title('Gráfico de dispersión')
     plt.xlabel('Radius')
     plt.ylabel('Perimeter')
     plt.show()
```



```
[7] sns.scatterplot(x='Concavity', y ='ConcavePoints', data=BCancer, hue='Diagnosis')
    plt.title('Gráfico de dispersión')
    plt.xlabel('Concavity')
    plt.ylabel('Radius')
    plt.show()
```



b) *Análisis de Componentes Principales*

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, después borramos las columnas que no necesitamos, en este caso Identificador y Diagnosis los cuales son nominales. Calculamos media y desviación estándar de cada variable. Por último, normalizamos y guardamos en una nueva variable llamada MNormalizada.

```
[9] from sklearn.preprocessing import StandardScaler
    from sklearn.decomposition import PCA
```

```
[12] normalizar = StandardScaler()                                    #Se instancia el objeto StandarScaler
     nuevaMatriz = BCancer.drop(columns=["Identificador","Diagnosis"])     #Se quitan las variables no necesarias (nominales)
     normalizar.fit(nuevaMatriz)                                      #Se calcula la media y desviación para cada variable
     MNormalizada = normalizar.transform(nuevaMatriz)
```

```
[13] pd.DataFrame(MNormalizada, columns = nuevaMatriz.columns)
```

	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392	-0.868652
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	0.939685	-0.398008
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383	4.910919
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.009560	-0.562450

Una vez que los datos han sido normalizados, calculamos la matriz de covarianzas o correlaciones y calculamos los componentes (eigen-vectores) y la varianza (eigen-valores), y los imprimimos.

```
[15] pca = PCA(n_components=None)           # Se instancia el objeto PCA
     pca.fit(MNormalizada)                  # Se obtiene los componentes
     print(pca.components_)
```

```
[[ 3.63937928e-01  1.54451129e-01  3.76044342e-01  3.64085847e-01
   2.32480530e-01  3.64442059e-01  3.95748488e-01  4.18038400e-01
   2.15237970e-01  7.18374352e-02]
 [-3.13929073e-01 -1.47180910e-01 -2.84657885e-01 -3.04841714e-01
   4.01962323e-01  2.66013147e-01  1.04285969e-01  7.18360466e-03
   3.68300910e-01  5.71767700e-01]
 [-1.24427590e-01  9.51056591e-01 -1.14083595e-01 -1.23377856e-01
  -1.66532470e-01  5.82778620e-02  4.11464835e-02 -6.85538259e-02
   3.67236467e-02  1.13583953e-01]
 [ 2.95588570e-02  8.91608121e-03  1.34580681e-02  1.34426810e-02
  -1.07802034e-01 -1.85700414e-01 -1.66653518e-01 -7.29839511e-02
   8.92998475e-01 -3.49331792e-01]
 [-3.10670238e-02 -2.19922759e-01 -5.94508289e-03 -1.93412233e-02
  -8.43745291e-01  2.40182964e-01  3.12533253e-01 -9.18019959e-03
   1.12888066e-01  2.64878075e-01]
 [-2.64180151e-01 -3.22065675e-02 -2.37819464e-01 -3.31707451e-01
   6.22253741e-02  5.27109684e-03  6.01467157e-01  2.65613396e-01
  -6.19570070e-02 -5.67918995e-01]
 [-4.41883879e-02  2.05574807e-02 -8.33692247e-02  2.61187967e-01
   1.12919772e-02 -8.03804838e-01  3.67136295e-01  1.41313069e-01
   4.79020066e-02  3.45213591e-01]
 [ 8.48340616e-02 -7.12679476e-03  8.92588808e-02  1.44609745e-01
   1.70503132e-01  6.39801435e-02  4.49573310e-01 -8.50918764e-01
   1.64556026e-02 -6.52594660e-02]
 [-4.74425304e-01 -4.21262951e-03 -3.80167210e-01  7.47347358e-01
  -5.84738672e-03  2.18732406e-01 -8.11706695e-02  2.20246512e-02
  -9.06784987e-03 -1.29667490e-01]
 [-6.69071489e-01  2.49782581e-04  7.40490534e-01 -3.23589581e-02
   3.69040560e-03 -5.27527799e-02 -1.03668029e-02 -3.74754732e-03
   1.46694726e-03  7.05734783e-03]]
```

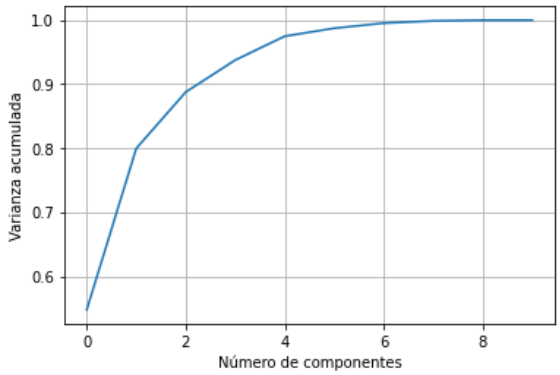
Calculamos el porcentaje de relevancia, es decir, entre el 75% y 90% de varianza total; si tomamos 4 componentes obtenemos el 93%, por lo que lo correcto será tomar 3 componentes ya que nos da un 88% y este porcentaje si entra dentro del rango requerido.

```
[17] Varianza = pca.explained_variance_ratio_
     print('Proporción de varianza:', Varianza)
     print('Varianza acumulada:', sum(Varianza[0:3]))
```

```
Proporción de varianza: [5.47858799e-01 2.51871359e-01 8.80615179e-02 4.99009435e-02
 3.72539192e-02 1.24141748e-02 8.00853111e-03 3.48897932e-03
 1.11354606e-03 2.82305886e-05]
Varianza acumulada: 0.8877916754778117
```

Luego graficamos la varianza acumulada de los componentes, para poder identificar con mayor facilidad el grupo de componentes con mayor varianza:

```
[18] # Se grafica la varianza acumulada en las nuevas dimensiones
     plt.plot(np.cumsum(pca.explained_variance_ratio_))
     plt.xlabel('Número de componentes')
     plt.ylabel('Varianza acumulada')
     plt.grid()
     plt.show()
```



Como último paso examinamos la proporción de relevancia o las cargas. Se revisan los valores absolutos de los componentes principales seleccionados. Como cuanto mayor sea el valor absoluto, más importante es esa variable en el componente principal, en este caso, identificamos las cargas mayores al 37%:

```
[22] CargasComponentes = pd.DataFrame(abs(pca.components_), columns=nuevaMatriz.columns)
     CargasComponentes
```

	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension
0	0.363938	0.154451	0.376044	0.364086	0.232481	0.364442	0.395748	0.418038	0.215238	0.071837
1	0.313929	0.147181	0.284658	0.304842	0.401962	0.266013	0.104286	0.007184	0.368301	0.571768
2	0.124428	0.951057	0.114084	0.123378	0.166532	0.058278	0.041146	0.068554	0.036724	0.113584
3	0.029559	0.008916	0.013458	0.013443	0.107802	0.185700	0.166654	0.072984	0.892998	0.349332
4	0.031067	0.219923	0.005945	0.019341	0.843745	0.240183	0.312533	0.009180	0.112888	0.264878
5	0.264180	0.032207	0.237819	0.331707	0.062225	0.005271	0.601467	0.265613	0.061957	0.567919
6	0.044188	0.020557	0.083369	0.261188	0.011292	0.803805	0.367136	0.141313	0.047902	0.345214
7	0.084834	0.007127	0.089259	0.144610	0.170503	0.063980	0.449573	0.850919	0.016456	0.065259
8	0.474425	0.004213	0.380167	0.747347	0.005847	0.218732	0.081171	0.022025	0.009068	0.129667
9	0.669071	0.000250	0.740491	0.032359	0.003690	0.052753	0.010367	0.003748	0.001467	0.007057

Las variables restantes que no tuvieron cargas mayores al porcentaje esperado se eliminan del *dataframe*, en este caso las variables son *Radius*, *Area*, *Compactness*, *Symmetry*, *Diagnosis* e *Identificador*.

```
[24] DatosCancer = BCancer.drop(columns = ["Identificador", "Diagnosis","Radius","Area","Compactness","Symmetry"])
     DatosCancer
```

	Texture	Perimeter	Smoothness	Concavity	ConcavePoints	FractalDimension
0	10.38	122.80	0.11840	0.30010	0.14710	0.07871
1	17.77	132.90	0.08474	0.08690	0.07017	0.05667
2	21.25	130.00	0.10960	0.19740	0.12790	0.05999
3	20.38	77.58	0.14250	0.24140	0.10520	0.09744
4	14.34	135.10	0.10030	0.19800	0.10430	0.05883
...	...	...	...	...	...	...
564	22.39	142.00	0.11100	0.24390	0.13890	0.05623
565	28.25	131.20	0.09780	0.14400	0.09791	0.05533
566	28.08	108.30	0.08455	0.09251	0.05302	0.05648
567	29.33	140.10	0.11780	0.35140	0.15200	0.07016
568	24.54	47.92	0.05263	0.00000	0.00000	0.05884

569 rows × 6 columns

Estandarización de los datos

Nuevamente estandarizamos los datos, pero ahora con nuestra nueva matriz *DatosCancer*:

```
[26] from sklearn.preprocessing import StandardScaler, MinMaxScaler
     estandarizar = StandardScaler()                          #Se instancia el objeto StandardScaler o MinMaxScaler
     MEstandarizada = estandarizar.fit_transform(DatosCancer)   #Se escalan los datos
     pd.DataFrame(MEstandarizada)
```

	0	1	2	3	4	5
0	-2.073335	1.269934	1.568466	2.652874	2.532475	2.255747
1	-0.353632	1.685955	-0.826962	-0.023846	0.548144	-0.868652
2	0.456187	1.566503	0.942210	1.363478	2.037231	-0.398008
3	0.253732	-0.592687	3.283553	1.915897	1.451707	4.910919
4	-1.151816	1.776573	0.280372	1.371011	1.428493	-0.562450
...	...	...	...	...	...	...
564	0.721473	2.060786	1.041842	1.947285	2.320965	-0.931027
565	2.085134	1.615931	0.102458	0.693043	1.263669	-1.058611
566	2.045574	0.672676	-0.840484	0.046588	0.105777	-0.895587
567	2.336457	1.982524	1.525767	3.296944	2.658866	1.043695
568	1.221792	-1.814389	-3.112085	-1.114873	-1.261820	-0.561032

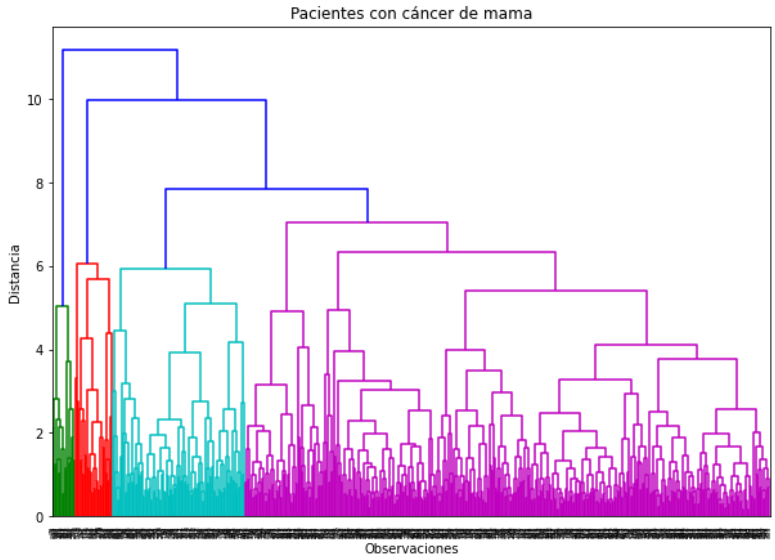
569 rows × 6 columns

Aplicación del algoritmo: Ascendente Jerárquico

a) *Creación del árbol*

Importamos las bibliotecas de clustering jerárquico para crear el árbol, las cuales son *scipy* y *sklearn Y graficamos:*

```
[27] #Se importan las bibliotecas de clustering jerárquico para crear el árbol
     import scipy.cluster.hierarchy as shc
     from sklearn.cluster import AgglomerativeClustering
     plt.figure(figsize=(10,7))
     plt.title("Pacientes con cáncer de mama")
     plt.xlabel("Observaciones")
     plt.ylabel("Distancia")
     Arbol = shc.dendrogram(shc.linkage(MEstandarizada, method = "complete", metric = "euclidean"))
     #plt.axhline(y=7, color = "orange", linestyle="--")
     #Probar con otras mediciones de distancia (euclidean,chebyshev, cityblock)
```



b) *Etiquetas en los clústeres*

Creamos las etiquetas de los elementos en los clústeres:

```
[28] #Se crean las etiquetas de los elementos en los clústeres
     MJerarquico = AgglomerativeClustering(n_clusters=4, linkage="complete", affinity="euclidean")
     MJerarquico.fit_predict(MEstandarizada)
     MJerarquico.labels_
```

```
array([2, 1, 1, 2, 1, 2, 1, 0, 2, 2, 0, 0, 1, 0, 2, 1, 0, 2, 1, 0, 0, 0,
       2, 1, 0, 2, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 2, 0, 1, 1,
       0, 0, 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 3, 3, 0, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 3, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0,
       2, 0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 3, 2, 0, 0, 0, 0, 0, 0, 1, 0, 3, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       2, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 2, 3, 1, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 3, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2,
       3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 3, 0, 0,
       0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 3,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 2, 2,
       0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1, 1, 0, 3, 0])
```

E imprimimos los datos, ahora con una nueva columna *clusterH* que nos indica a que clúster pertenece ese registro:

```
[29] BCancer["clusterH"] = MJerarquico.labels_
     BCancer
```

	Identificador	Diagnosis	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension	clusterH
0	P-842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	2
1	P-842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	1
2	P-84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	1
3	P-84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	2
4	P-84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	P-926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	1
565	P-926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	1
566	P-926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	0
567	P-927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	3
568	P-92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	0

569 rows × 13 columns

Utilizamos la función *groupby()* y *count()* para observar cuantos elementos tiene cada clúster. El clúster 0 tiene 416, el 1 105, el 2 tiene 30, etc.

```
[30] #Cantidad de elementos en los clusters
     BCancer.groupby(["clusterH"])["clusterH"].count()
```

```
clusterH
0    416
1    105
2     30
3     18
Name: clusterH, 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:

```
[31] BCancer[BCancer.clusterH == 0]
```

	Identificador	Diagnosis	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension	clusterH
7	P-84458202	M	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	0.2196	0.07451	0
10	P-845636	M	16.02	23.24	102.70	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	0.05697	0
11	P-84610002	M	15.78	17.89	103.60	781.0	0.09710	0.12920	0.09954	0.06606	0.1842	0.06082	0
13	P-846381	M	15.85	23.95	103.70	782.7	0.08401	0.10020	0.09938	0.05364	0.1847	0.05338	0
16	P-848406	M	14.68	20.13	94.74	684.5	0.09867	0.07200	0.07395	0.05259	0.1586	0.05922	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
559	P-925291	B	11.51	23.93	74.52	403.5	0.09261	0.10210	0.11120	0.04105	0.1388	0.06570	0
560	P-925292	B	14.05	27.15	91.38	600.4	0.09929	0.11260	0.04462	0.04304	0.1537	0.06171	0
561	P-925311	B	11.20	29.37	70.67	386.0	0.07449	0.03558	0.00000	0.00000	0.1060	0.05502	0
566	P-926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	0
568	P-92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	0

416 rows × 13 columns

c) *Obtención de los centroides*

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

```
[32] CentroidesH = BCancer.groupby(["clusterH"])["Texture", "Perimeter", "Smoothness" ,"Concavity", "ConcavePoints", "FractalDimension"].mean()
     CentroidesH
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple c
  """Entry point for launching an IPython kernel.
```

clusterH	Texture	Perimeter	Smoothness	Concavity	ConcavePoints	FractalDimension
0	18.309087	82.359375	0.092786	0.051333	0.030737	0.061853
1	22.648286	121.482857	0.100994	0.164879	0.092342	0.060987
2	18.750667	86.129333	0.117116	0.204140	0.082217	0.079053
3	23.257778	151.627778	0.117339	0.318656	0.160328	0.068105

Interpretación

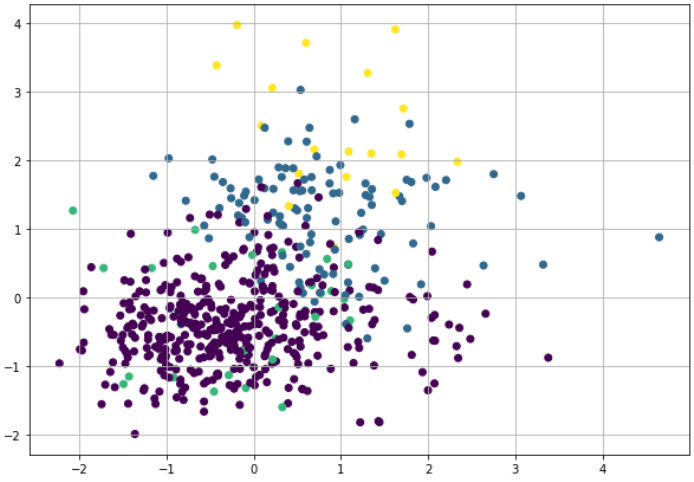
A la tabla que sacamos hay que analizarla y deducir una interpretación:

- **Clúster 0:** Conformado por 416 pacientes, con un perímetro promedio de tumor de 82 pixeles y una desviación estándar de textura de 18 pixeles. Aparentemente es un tumor cuya suavidad alcanza 0.09 pixeles, una concavidad de 0.05 pixeles, y 0.03 pixeles de puntos cóncavos y una dimensión fractal promedio de 0.06 pixeles.
- **Clúster 1:** Conformado por 105 pacientes, con un perímetro promedio de tumor de 121 pixeles y una desviación estándar de textura de 22 pixeles. Aparentemente es un tumor cuya suavidad alcanza 0.10 pixeles, una concavidad de 0.16 pixeles, y 0.09 pixeles de puntos cóncavos y una dimensión fractal promedio de 0.06 pixeles.
- **Clúster 2:** Conformado por 30 pacientes, con un perímetro promedio de tumor de 86 pixeles y una desviación estándar de textura de 18 pixeles. Aparentemente es un tumor cuya suavidad alcanza 0.11 pixeles, una concavidad de 0.20 pixeles, y 0.08 pixeles de puntos cóncavos y una dimensión fractal promedio de 0.07 pixeles.
- **Clúster 3:** Conformado por 18 pacientes, con un perímetro promedio de tumor de 151 pixeles y una desviación estándar de textura de 23 pixeles. Aparentemente es un tumor cuya suavidad

alcanza 0.11 pixeles, una concavidad de 0.31 pixeles, y 0.16 pixeles de puntos cóncavos y una dimensión fractal promedio de 0.06 pixeles.

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

```
[33] plt.figure(figsize=(10,7))
     plt.scatter(MEstandarizada[:,0], MEstandarizada[:,1], c=MJerarquico.labels_)
     plt.grid()
     plt.show()
```



Aplicación del algoritmo: K-means

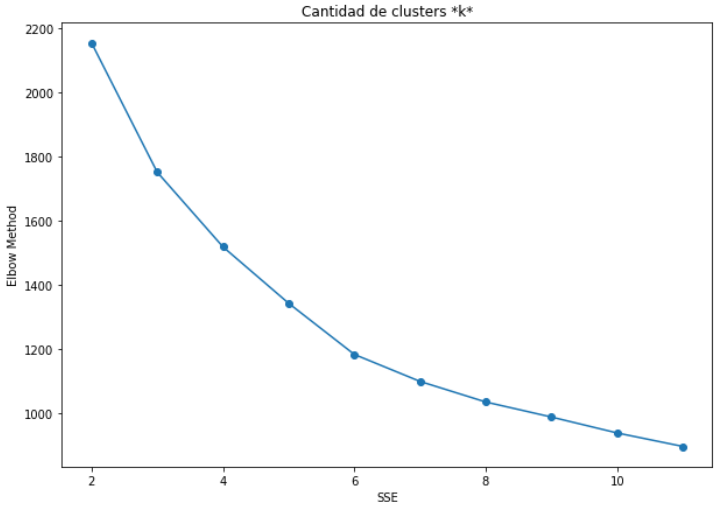
a) *Método del codo*

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:

```
[34] #Se importan las bibliotecas de clustering jerárquico para crear el árbol
     from sklearn.cluster import KMeans
     from sklearn.metrics import pairwise_distances_argmin_min
```

```
[35] #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.title("Cantidad de clusters *k*")
     plt.xlabel("SSE")
     plt.ylabel("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,

```
[36] !pip install kneed
```

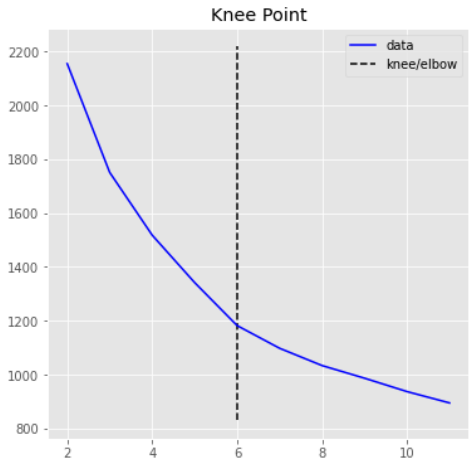
```
Collecting kneed
  Downloading kneed-0.7.0-py2.py3-none-any.whl (9.4 kB)
Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.7/dist-packages (from kneed) (1.19.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from kneed) (3.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from kneed) (1.4.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->kneed) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib->kneed) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->kneed) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->kneed) (1.3.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from cycler>=0.10->matplotlib->kneed) (1.15.0)
Installing collected packages: kneed
Successfully installed kneed-0.7.0
```

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

```
6
```

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

```
[38] plt.style.use("ggplot")
     kl.plot_knee()
```



b) *Etiquetas en los clústeres*

Creamos las etiquetas de los elementos en los clústeres:

```
[39] #Se crean las etiquetas de los elementos en los clústeres
     MParticional = KMeans(n_clusters=6, random_state=0).fit(MEstandarizada)
     MParticional.predict(MEstandarizada)
     MParticional.labels_
```

```
array([5, 0, 1, 5, 0, 5, 0, 5, 5, 5, 4, 0, 5, 0, 5, 5, 0, 5, 0, 2, 3, 3,
       5, 0, 0, 5, 5, 0, 0, 0, 1, 5, 1, 0, 0, 0, 0, 2, 4, 3, 4, 3, 0, 3,
       4, 0, 2, 5, 3, 4, 4, 2, 2, 0, 4, 2, 0, 5, 2, 3, 3, 3, 5, 2, 5, 5,
       3, 2, 5, 2, 0, 3, 0, 3, 2, 0, 3, 5, 1, 2, 3, 3, 1, 1, 2, 0, 0, 0,
       4, 3, 4, 0, 2, 2, 0, 0, 3, 3, 2, 3, 4, 3, 4, 3, 3, 5, 3, 2, 1, 4,
       3, 3, 5, 3, 3, 4, 3, 5, 5, 0, 2, 0, 1, 3, 2, 2, 4, 0, 5, 1, 3, 0,
       0, 2, 0, 4, 3, 2, 5, 3, 2, 0, 3, 2, 2, 3, 5, 2, 3, 2, 3, 3, 5, 2,
       2, 2, 0, 2, 2, 2, 3, 0, 1, 3, 0, 2, 2, 0, 0, 2, 2, 2, 5, 3, 2, 3,
       3, 0, 4, 2, 1, 1, 0, 2, 4, 2, 0, 2, 2, 2, 5, 4, 2, 4, 5, 2, 5, 0,
       0, 0, 2, 0, 1, 5, 3, 2, 3, 0, 3, 2, 0, 2, 1, 0, 0, 3, 2, 2, 0, 0,
       2, 3, 3, 0, 2, 2, 3, 2, 4, 5, 5, 4, 4, 0, 2, 4, 1, 0, 4, 0, 2, 2,
       3, 4, 0, 3, 2, 2, 4, 3, 1, 2, 1, 0, 0, 3, 0, 5, 5, 0, 0, 4, 0, 2,
       0, 0, 3, 4, 2, 3, 2, 3, 1, 2, 0, 3, 2, 0, 2, 2, 0, 2, 0, 5, 2, 2,
       4, 2, 4, 2, 5, 2, 3, 2, 2, 2, 2, 3, 2, 3, 1, 2, 1, 3, 2, 4, 2, 2,
       2, 2, 2, 2, 2, 2, 3, 2, 2, 0, 5, 2, 3, 0, 3, 1, 2, 3, 2, 2, 0, 5,
       0, 3, 3, 2, 2, 0, 3, 0, 3, 1, 3, 3, 3, 0, 3, 3, 2, 2, 2, 3, 2, 5,
       1, 0, 2, 2, 3, 2, 2, 3, 2, 4, 2, 0, 2, 0, 0, 2, 0, 1, 0, 2, 1, 0,
       2, 3, 5, 4, 2, 5, 3, 2, 4, 3, 2, 4, 2, 2, 3, 0, 3, 3, 5, 1, 3, 2,
       3, 2, 2, 2, 1, 2, 2, 2, 2, 3, 2, 4, 0, 2, 2, 3, 4, 4, 4, 4, 3, 5,
       2, 3, 2, 5, 3, 2, 3, 4, 3, 4, 2, 2, 5, 3, 1, 0, 2, 3, 2, 2, 2, 2,
       3, 0, 2, 2, 0, 3, 0, 2, 2, 0, 4, 0, 4, 3, 2, 4, 4, 4, 4, 4, 0, 1,
       4, 2, 2, 4, 4, 2, 5, 3, 2, 4, 2, 4, 3, 2, 4, 2, 3, 5, 2, 2, 3, 2,
       3, 3, 2, 1, 3, 2, 4, 2, 0, 2, 4, 0, 3, 2, 0, 1, 3, 5, 3, 0, 5, 5,
       3, 3, 2, 5, 2, 2, 5, 2, 2, 3, 0, 0, 3, 3, 3, 1, 2, 3, 3, 3, 3, 2,
       3, 3, 3, 3, 2, 0, 3, 1, 0, 3, 4, 4, 3, 4, 4, 4, 3, 4, 2, 3, 2, 4,
       4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 5, 1, 1, 0, 4, 1, 4],
      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:

```
[40] BCancer["clusterP"] = MParticional.labels_
     BCancer
```

	Identificador	Diagnosis	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension	clusterH	clusterP
0	P-842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	2	5
1	P-842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	1	0
2	P-84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	1	1
3	P-84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	2	5
4	P-84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	P-926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	1	1
565	P-926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	1	0
566	P-926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	0	4
567	P-927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	3	1
568	P-92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	0	4

569 rows × 14 columns

Utilizamos la función *groupby()* y *count()* para observar cuantos elementos tiene cada clúster. El clúster 0 tiene 103, el 1 36, el 2 tiene 169, el 3 tiene 127, etc.

```
[41] #Cantidad de elementos en los clusters
     BCancer.groupby(["clusterP"])["clusterP"].count()
```

```
clusterP
0    103
1     36
2    169
3    127
4     77
5     57
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:

```
[42] BCancer[BCancer.clusterP == 0]
```

	Identificador	Diagnosis	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension	clusterH	clusterP
1	P-842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	1	0
4	P-84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	1	0
6	P-844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	0.1794	0.05742	1	0
11	P-84610002	M	15.78	17.89	103.60	781.0	0.09710	0.12920	0.09954	0.06606	0.1842	0.06082	0	0
13	P-846381	M	15.85	23.95	103.70	782.7	0.08401	0.10020	0.09938	0.05364	0.1847	0.05338	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
516	P-916799	M	18.31	20.58	120.80	1052.0	0.10680	0.12480	0.15690	0.09451	0.1860	0.05941	1	0
517	P-916838	M	19.89	20.26	130.50	1214.0	0.10370	0.13100	0.14110	0.09431	0.1802	0.06188	1	0
533	P-91930402	M	20.47	20.67	134.70	1299.0	0.09156	0.13130	0.15230	0.10150	0.2166	0.05419	1	0
536	P-91979701	M	14.27	22.55	93.77	629.8	0.10380	0.11540	0.14630	0.06139	0.1926	0.05982	1	0
565	P-926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	1	0

103 rows × 14 columns

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

```
[43] CentroidesP = BCancer.groupby(["clusterP"])["Texture", "Perimeter", "Smoothness" ,"Concavity", "ConcavePoints", "FractalDimension"].mean()
     CentroidesP
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple
  """Entry point for launching an IPython kernel.
```

clusterP	Texture	Perimeter	Smoothness	Concavity	ConcavePoints	FractalDimension
0	21.780485	118.559903	0.096619	0.132804	0.079392	0.058170
1	22.361389	145.452778	0.111778	0.270875	0.142322	0.064372
2	16.247870	81.291420	0.085740	0.033679	0.021541	0.059116
3	17.145984	74.945354	0.105395	0.055820	0.032692	0.067427
4	24.289610	80.890000	0.084377	0.044193	0.023581	0.060271
5	19.889123	94.694211	0.113700	0.191454	0.086418	0.074183

Interpretación

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

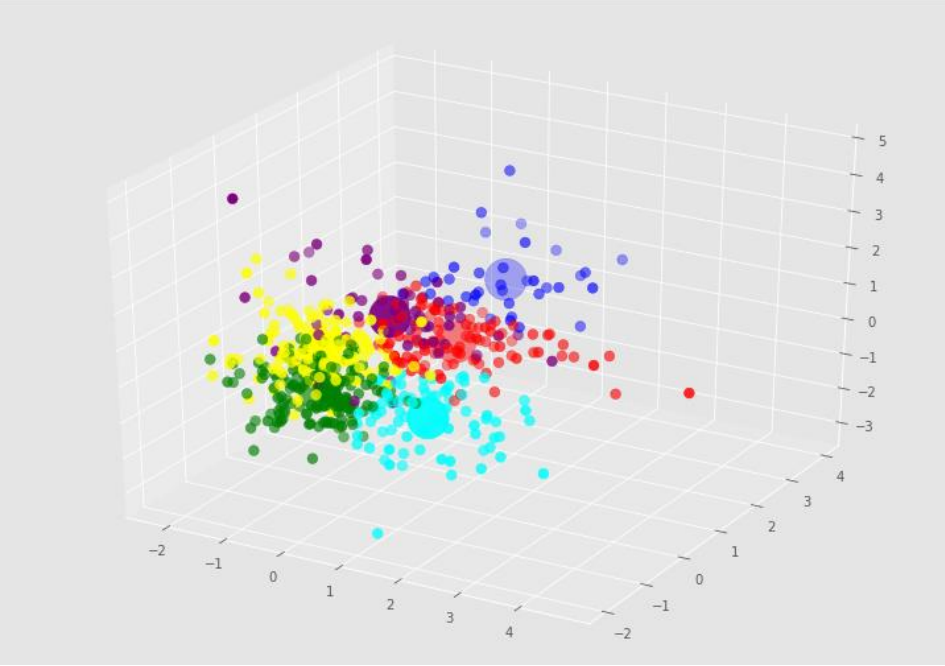
- **Clúster 0:** Conformado por 103 pacientes, con un perímetro promedio de tumor de 118 pixeles y una desviación estándar de textura de 21 pixeles. Aparentemente es un tumor cuya suavidad alcanza 0.09 pixeles, una concavidad de 0.13 pixeles, y 0.07 pixeles de puntos cóncavos y una dimensión fractal promedio de 0.05 pixeles.
- **Clúster 1:** Conformado por 36 pacientes, con un perímetro promedio de tumor de 145 pixeles y una desviación estándar de textura de 22 pixeles. Aparentemente es un tumor cuya suavidad alcanza 0.11 pixeles, una concavidad de 0.27 pixeles, y 0.14 pixeles de puntos cóncavos y una dimensión fractal promedio de 0.06 pixeles.
- **Clúster 2:** Conformado por 169 pacientes, con un perímetro promedio de tumor de 81 pixeles y una desviación estándar de textura de 16 pixeles. Aparentemente es un tumor cuya suavidad alcanza 0.08 pixeles, una concavidad de 0.03 pixeles, y 0.02 pixeles de puntos cóncavos y una dimensión fractal promedio de 0.05 pixeles.
- **Clúster 3:** Conformado por 127 pacientes, con un perímetro promedio de tumor de 74 pixeles y una desviación estándar de textura de 17 pixeles. Aparentemente es un tumor cuya suavidad alcanza 0.10 pixeles, una concavidad de 0.05 pixeles, y 0.03 pixeles de puntos cóncavos y una dimensión fractal promedio de 0.06 pixeles.
- **Clúster 4:** Conformado por 77 pacientes, con un perímetro promedio de tumor de 80 pixeles y una desviación estándar de textura de 24 pixeles. Aparentemente es un tumor cuya suavidad alcanza 0.08 pixeles, una concavidad de 0.04 pixeles, y 0.02 pixeles de puntos cóncavos y una dimensión fractal promedio de 0.06 pixeles.
- **Clúster 5:** Conformado por 57 pacientes, con un perímetro promedio de tumor de 94 pixeles y una desviación estándar de textura de 19 pixeles. Aparentemente es un tumor cuya suavidad alcanza 0.11 pixeles, una concavidad de 0.19 pixeles, y 0.08 pixeles

de puntos cóncavos y una dimensión fractal promedio de 0.07 pixeles.

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

```
[44] #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", "cyan", "purple"]
     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

En esta sesión pusimos en práctica los dos tipos de clustering vistos anteriormente, aunque se podría pensar que podríamos tener resultados iguales con ambos procedimientos, se demostró que cada clustering arroja resultados

diferentes, ya que por ejemplo en el clustering jerárquico obtuvimos 4 clústeres, mientras que en el particional fueron 6, además los centroides de ambos fueron diferentes. Esto quiere decir que es importante elegir con sumo cuidado que procedimiento elegir a la hora de ponerlo en práctica ya que podríamos obtener resultados distintos a los esperados.