Objetivo.

Obtener grupos de deportistas con características similares, participantes en juegos olímpicos, a través de clustering jerárquico y particional.

Características.

Rregistros de las caracteristicas que identifican a deportistas de alto rendimiento los cuales participaron en diferentes deportes olímpicos a lo largo del siglo pasado y actual.

Los atributos de la fuente de datos son los siguientes.

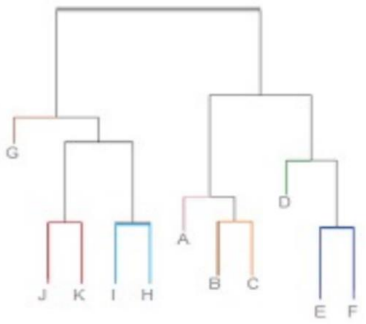
- ID: Identificador del atleta.
- Name: Nombre del atleta.
- Team: Pais del atleta.
- NOC: Comite olimpico al que pertenece el atleta.
- Games: Año y temporada de los juegos
- Age: Edad de los atletas.
- Height: Altura de los atletas.
- Weight: Peso de los atletas.
- Year: Año en el que se celebraron los juegos olímpicos.
- Medal: Medalla obtenida en los juegos.
- Sex: Sexo de los atletas.
    - M: Femenino
    - F: Masculino
- Season: Temporada en la que se celebraron los juegos olímpicos.
- Sport: Deporte que practica el atleta.
- Event: Nombre del evento en los juegos olímpicos

Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 31/03/2022
Practica No.7 Clustering Jerárquico y Particional

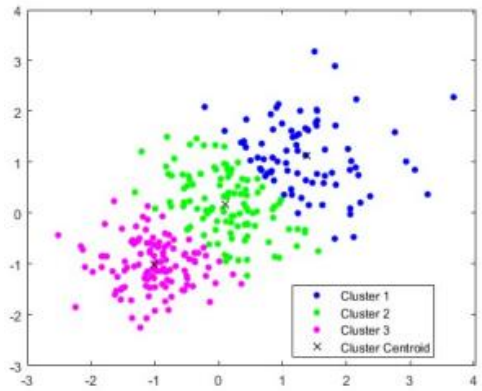
El clustering lo que busca es hacer una segmentación de los registros en el conjunto de datos, con esto se descubren una serie de patrones que normalmente estarían ocultos los cuales relacionan a cada uno de ellos consiguiendo agruparlos, sin embargo, no se ofrece una descripción clara y es tarea de nosotros darles una interpretación adecuada.

En particular el clustering *jerárquico* organiza los datos en una estructura de tipo árbol de forma recursiva de tal manera que en cada nivel del árbol se tiene cierto número de clústeres no definido e incrementa conforme bajamos en el árbol.



Por otro lado, el clustering *particional* organiza a los elementos determinando n centroides (Punto que ocupa la posición media en un clúster), donde n es un número que tenemos que definir nosotros, se usara el algoritmo de k-means el cual va a estar moviendo iterativamente los centroides tomando la media de cada clúster que se forme hasta un punto de convergencia donde ya no se muevan más.

Un punto importante de este algoritmo es que tenemos que definir el número de clústeres con los cuales queremos terminar y para ello es necesario utilizar otras herramientas como el método del codo, de la rodilla o el clustering jerárquico de la practica anterior los cuales nos sugieren un buen número por el cual podríamos iniciar con nuestro análisis.



Desarrollo.

Como primer paso tenemos la importación de las librerías necesarias para trabajar con el conjunto de datos de los pacientes a segmentar. `pandas` para la manipulación de los datos, `numpy` para el manejo de matrices, `matplotlib` y `seaborn` para la visualización de estos datos mediante gráficos de diferente tipo dependiendo del análisis, `StandardScaler` y `MinMaxScaler` para escalar o normalizar el conjunto de datos evitando sesgos en el análisis y aumentando también el rendimiento de los algoritmos.

Para el algoritmo de clustering jerárquico necesitaremos de `scipy` a la librería `hierarchy` para la implementación del modelo para el clustering jerárquico y además `AgglomerativeClustering` para etiquetar a cada registro con su respectivo cluster.

Por la parte del clustering particional necesitaremos además instalar la librería `kneed` para el método de la rodilla el cual nos da una aproximación adecuada de cual deberá de ser el número de clústeres para el algoritmo de k-means y por último la librería de `KMeans` que se encuentra dentro de los paquetes que ofrece `sklearn` para estos desarrollos.

Nuestro `DataFrame` leído con ayuda de `pandas` en un inicio se ve de la siguiente forma con 15 columnas y 271116 registros.

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
4	5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
271111	135569	Andrzej ya	M	29.0	179.0	89.0	Poland-1	POL	1976 Winter	1976	Winter	Innsbruck	Luge	Luge Mixed (Men)'s Doubles	NaN
271112	135570	Piotr ya	M	27.0	176.0	59.0	Poland	POL	2014 Winter	2014	Winter	Sochi	Ski Jumping	Ski Jumping Men's Large Hill, Individual	NaN
271113	135570	Piotr ya	M	27.0	176.0	59.0	Poland	POL	2014 Winter	2014	Winter	Sochi	Ski Jumping	Ski Jumping Men's Large Hill, Team	NaN

Como podemos apreciar la mayoría de los datos son valores nominales por lo cual me propuse evaluar cuales valía más la pena convertir a un valor numérico para considerarlo en el análisis. De esta manera considere que `Sex`, `Season` y `Medal` son los atributos nominales que podrían aportar al modelo por lo cual utilizando Pandas se mapearon los valores de femenino (0) y masculino (1) , se siguió una lógica similar para cambiar los valores para `Season` de Summer (0) y Winter (1) por ultimo las medallas se colocaron en 0 para los atletas que no ganaron medalla, 60 para las medallas de bronce, 80 para las medallas de plata y 100 para las medallas de oro teniendo que hacer un cast para estas últimas ya que por defecto los valores los cambia a unit8 y se necesitan en int64 para que `seaborn` los considere.

```
#Se crean valores nominales para las columnas Sex y Season ya que son binarios
Atletas_numericos = pd.get_dummies(Atletas, columns = ["Sex", "Season"],
drop_first=True)
Atletas_numericos.rename(columns = {"Sex_M": 'Sex', "Season_Winter": 'Season'},
inplace = True)
#Atletas_numericos
medals_ = {'Gold': 100, 'Silver': 80, 'Bronze': 60, 'NaN':0}
Atletas_numericos['Medal'] = Atletas_numericos['Medal'].map(medals_)              #
Cambiamos las medallas a un valor numerico
Atletas_numericos['Medal'] = Atletas_numericos['Medal'].fillna(0)                 #
Eliminamos los NaN de medallas
Atletas_numericos = Atletas_numericos.dropna()                                    #
Se eliminan todos los registros con NaN
Atletas = Atletas_numericos
Atletas
```

Las demás variables nominales se eliminaron ya que tenderían a tener muchos valores, estas variables fueron ID, Name, Team, NOC, Games, City, Event y Sport.

```
# Eliminando las columnas nominales innecesarias para el analisis
Atletas = Atletas.drop(columns = ['ID', 'Name', 'Team', 'NOC', 'Games', 'City',
'Event', 'Sport'])
Atletas = Atletas.astype({"Sex": int, "Season": int}, errors='raise')       # Se hace
un cast de unit8 a int
Atletas.info()
```

```
Int64Index: 206165 entries, 0 to 271115
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Age     206165 non-null  float64
 1   Height  206165 non-null  float64
 2   Weight  206165 non-null  float64
 3   Year    206165 non-null  int64
 4   Medal   206165 non-null  float64
 5   Sex     206165 non-null  int64
 6   Season  206165 non-null  int64
dtypes: float64(4), int64(3)
memory usage: 12.6 MB
```

Hasta ahora se tiene una estructura en el `dataframe` como de muestra en la imagen superior, sin embargo 271116 es un número muy grande de registros para el poder de computo con el que se cuenta por lo cual se opto por tomar una muestra representativa de 20, 000.

```
#Seleccion de una muestra del conjunto de datos
Atletas = Atletas.sample(n=1000, random_state=1)
```

El resultado de este primer acercamiento al conjunto de datos se muestra a continuación, se tiene un dataframe con 20,000 registros y 7 columnas las cuales aún se someterán a un análisis de reducción de variables para intentar aumentar el rendimiento del modelo.

	Age	Height	Weight	Year	Medal	Sex	Season
11474	29.0	188.0	72.0	1980	0.0	1	0
110653	20.0	167.0	66.0	1992	0.0	1	0
217501	27.0	175.0	70.0	1972	60.0	1	0
181056	21.0	184.0	74.0	1992	0.0	1	0
210835	25.0	175.0	74.0	1964	0.0	1	0
...	...	...	...	...	...	...	...
4956	25.0	184.0	82.0	2004	0.0	1	0
145158	22.0	173.0	73.0	2000	0.0	1	0
258748	26.0	173.0	72.0	1996	0.0	1	0
263059	23.0	187.0	89.0	1984	0.0	1	0
247833	26.0	172.0	69.0	1960	100.0	1	0

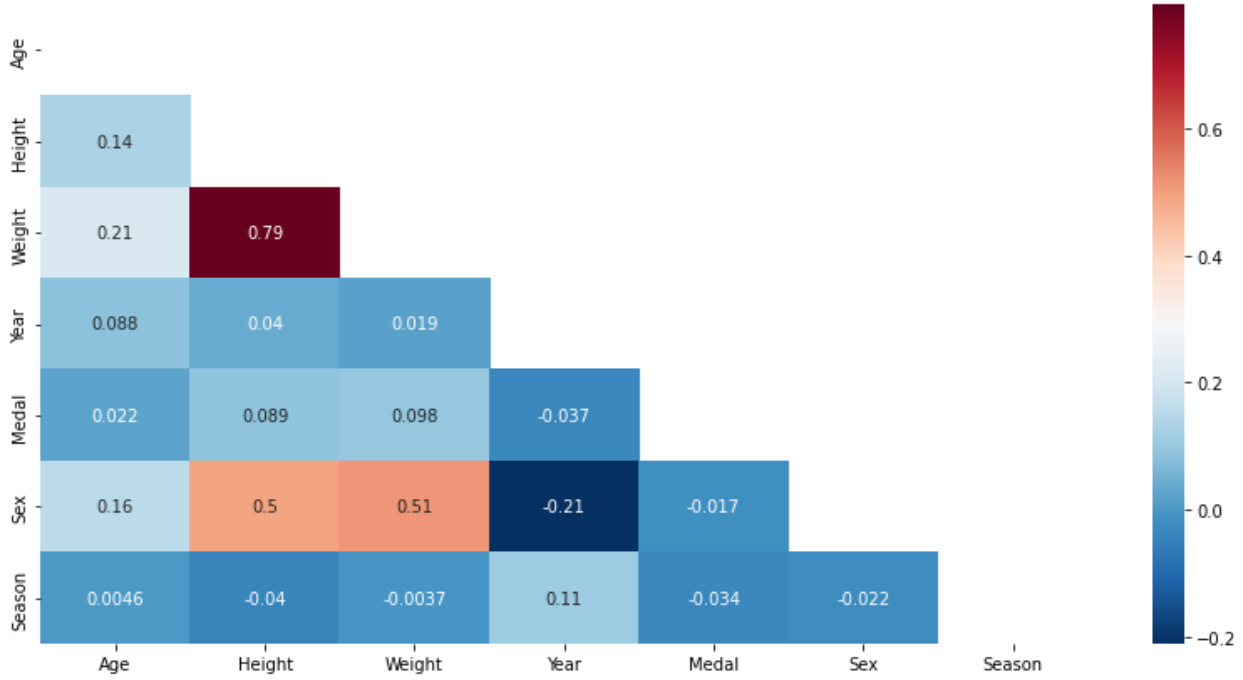
20000 rows × 7 columns

Para detectar este tipo de relación entre variables lo primero que se hizo fue trazar el grafico de dispersión de cada una de las características del modelo para observar a simple vista si es que hay candidatos a ser eliminados, este grafico lo podemos ver más claramente en la siguiente página.

En la matriz podemos ver que pudiera existir una relación fuerte de dependencia entre las variables `Weight` y `Height` lo cual tiene sentido ya que mientras más grande es una persona ésta pesa más.

Para asegurarnos de esta suposiciones se calculó la matriz de correlaciones con el método `corr` del `DataFrame`, con esta matriz nosotros podemos ordenar las columnas para que muestre la relación que existe para determinada variable con ayuda del método `sort_values` pasando la columna del `DataFrame` que queramos ordenar o bien podemos pasar esta matriz de correlaciones como parámetro a un mapa de calor el cual pinta de colores claros o fríos los índices de interés, esto se implementa con `seaborn` para observar más claro cuáles son los índices de correlación altos ya que al ser una matriz numérica de 7x7 es un poco complicado identificarlos y se tiende a cometer errores.

A continuación, se muestra el mapa de calor de la matriz de correlación de variables calculada con el método de `Pearson`.



Si bien existe una relación fuerte entre el peso (`Weight`) y la altura (`Height`) se optará por quedarnos con ambas ya que como estamos trabajando con deportistas resulta conveniente analizar estos dos aspectos.

Al final la selección de características para el modelo quedo de la siguiente manera.

- ID: Identificador del atleta. [SE ELIMINA]
- Name: Nombre del atleta. [SE ELIMINA]
- Team: Pais del atleta. [SE ELIMINA]
- NOC: Comite olimpico al que pertenece el atleta. [SE ELIMINA]
- Games: Año y temporada de los juegos [SE ELIMINA]
- Age: Edad de los atletas. [SE CONSERVA]
- Height: Altura de los atletas. [SE CONSERVA]
- Weight: Peso de los atletas. [SE CONSERVA]
- Year: Año en el que se celebraron los juegos olímpicos. [SE CONSERVA]
- Medal: Medalla obtenida en los juegos. [SE CONSERVA]
    - 0: No obtuvo medalla.
    - 60: Medalla de bronce.
    - 80: Medalla de plata.
    - 100: Medalla de oro.

- Sex: Sexo de los atletas. [SE CONSERVA]
  - M: Femenino.
  - F: Masculino.
- Season: Temporada en la que se celebraron los juegos olímpicos. [SE CONSERVA]
  - 0: Verano.
  - 1: invierno.
- Sport: Deporte que practica el atleta. [SE ELIMINA]
- Event: Nombre del evento en los juegos olímpicos. [SE ELIMINA]

Con la selección de características y la reducción de registros completa lo que restaría seria hacer un nuevo `DataFrame` con esas columnas (20000x7) para después hacer el proceso de estandarizar o normalizar los datos con el objetivo de que cada uno de éstos aporte la misma información a los modelos evitando sesgos y aumentando el rendimiento que se pudiera tener. Estos procesos se hacen con instancias de las clases `StandardScaler` (Estandarizar) y `MinMaxScaler` (Normalizar). A continuación, se muestran las matrices resultantes.

Estandarización. (Media = 0, Desviación Estándar = 1 y utilizando `StandardScaler.fit_transform`).

	0	1	2	3	4	5	6
0	0.697267	1.179677	0.082697	-0.475719	-0.409036	0.680641	-0.48385
1	-0.917006	-0.792156	-0.332545	0.116468	-0.409036	0.680641	-0.48385
2	0.338540	-0.040981	-0.055717	-0.870511	1.643157	0.680641	-0.48385
3	-0.737642	0.804090	0.221112	0.116468	-0.409036	0.680641	-0.48385
4	-0.020187	-0.040981	0.221112	-1.265302	-0.409036	0.680641	-0.48385
...	...	...	...	...	...	...	...
19995	-0.020187	0.804090	0.774769	0.708656	-0.409036	0.680641	-0.48385
19996	-0.558278	-0.228775	0.151904	0.511260	-0.409036	0.680641	-0.48385
19997	0.159176	-0.228775	0.082697	0.313864	-0.409036	0.680641	-0.48385
19998	-0.378915	1.085780	1.259219	-0.278323	-0.409036	0.680641	-0.48385
19999	0.159176	-0.322672	-0.124924	-1.462698	3.011287	0.680641	-0.48385

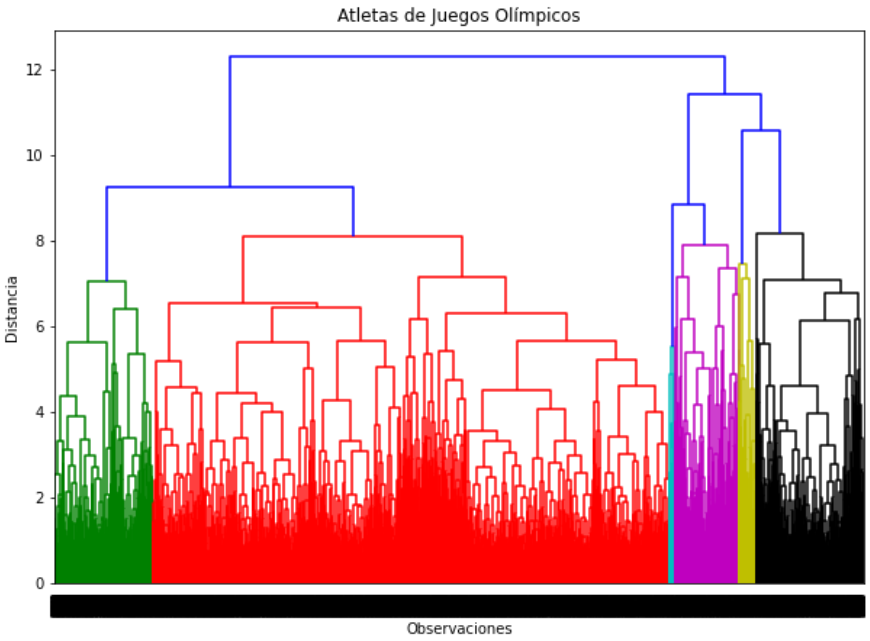
Normalización. (Escala de valores entre 0 y 1 utilizando `MinMaxScaler.fit_transform`).

	0	1	2	3	4	5	6
0	0.305085	0.616162	0.296296	0.700000	0.0	1.0	0.0
1	0.152542	0.404040	0.255892	0.800000	0.0	1.0	0.0
2	0.271186	0.484848	0.282828	0.633333	0.6	1.0	0.0
3	0.169492	0.575758	0.309764	0.800000	0.0	1.0	0.0
4	0.237288	0.484848	0.309764	0.566667	0.0	1.0	0.0
...	...	...	...	...	...	...	...
19995	0.237288	0.575758	0.363636	0.900000	0.0	1.0	0.0
19996	0.186441	0.464646	0.303030	0.866667	0.0	1.0	0.0
19997	0.254237	0.464646	0.296296	0.833333	0.0	1.0	0.0
19998	0.203390	0.606061	0.410774	0.733333	0.0	1.0	0.0
19999	0.254237	0.454545	0.276094	0.533333	1.0	1.0	0.0

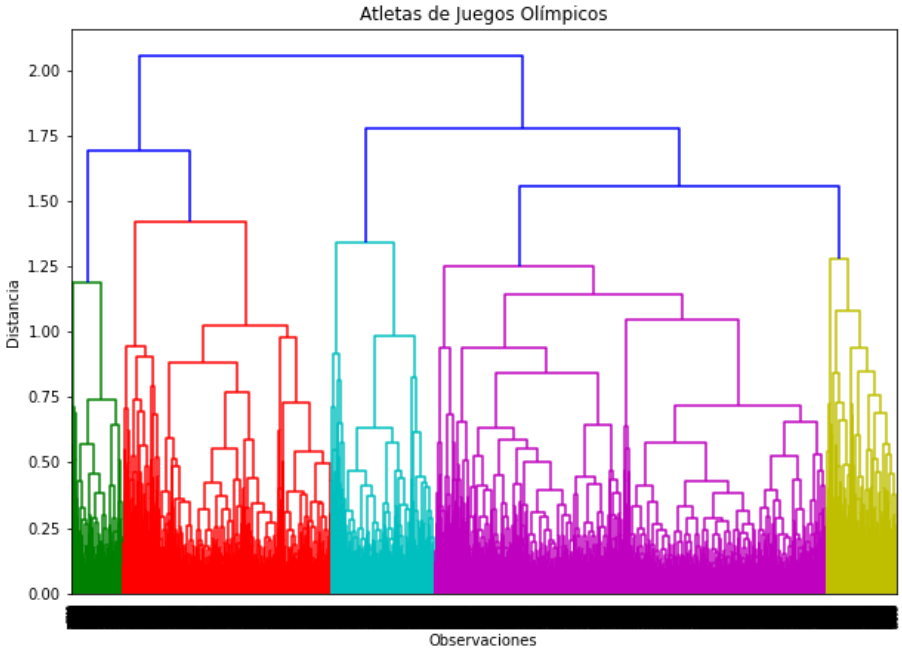
Con nuestros datos ya en forma podemos comenzar con la aplicación de los algoritmos iniciando con el clustering Jerárquico Particional el cual solamente necesita como entrada la matriz de datos con los registros que se van a segmentar y la métrica de distancias que queremos usar ayudándonos de la librería de `scipy` en su modulo `hierarchy` para trazar el `dendrograma` o grafico de árbol de la segmentación el cual se generó visualmente con ayuda de `matplotlib`. A continuación, se muestran los resultados de los clústeres obtenidos para las distancias euclidiana, chebyshev y Manhattan.

```
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
plt.figure(figsize=(10, 7))
plt.title("Atletas de Juegos Olímpicos")
plt.xlabel('Observaciones')
plt.ylabel('Distancia')
Arbol = shc.dendrogram(shc.linkage(MNormalizada, method='complete', metric='chebyshev'))
```

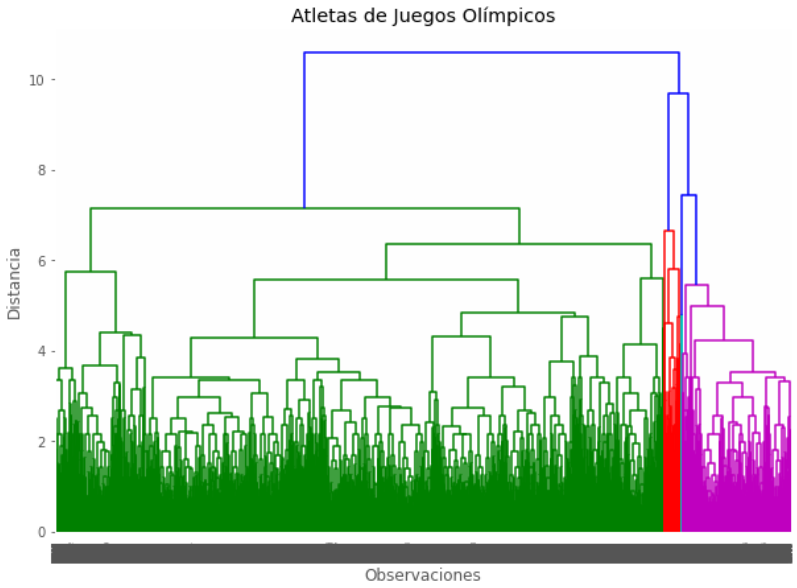
Métrica de distancia: `euclidean`, Matriz: `Estandarizada`, Resultado: `6 clústeres`.



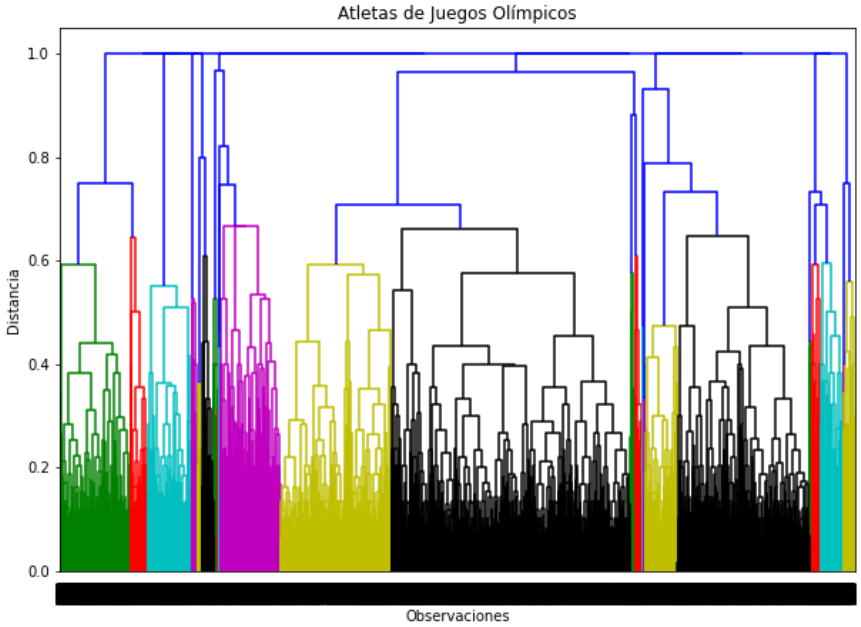
Métrica de distancia: `euclidean`, Matriz: `Normalizada`, Resultado: `5 clústeres`.



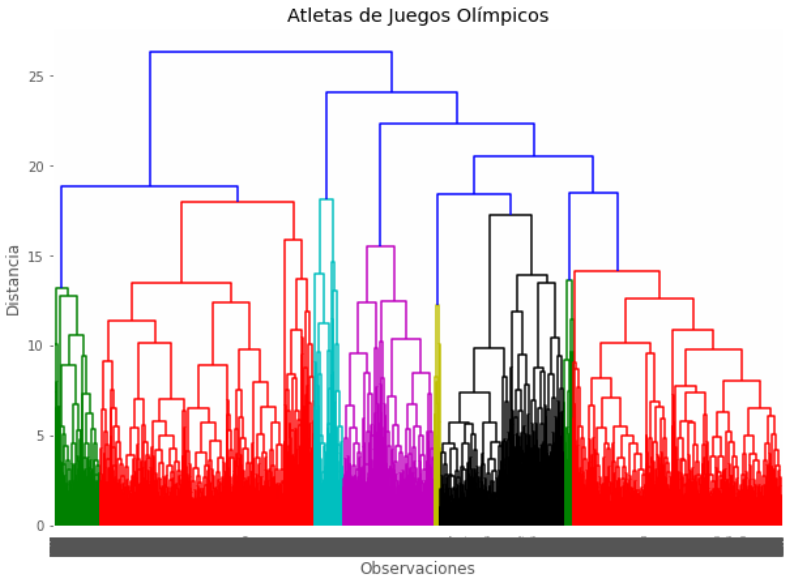
Métrica de distancia: `chebyshev`, Matriz: `Estandarizada`, Resultado: `4 clústeres`.



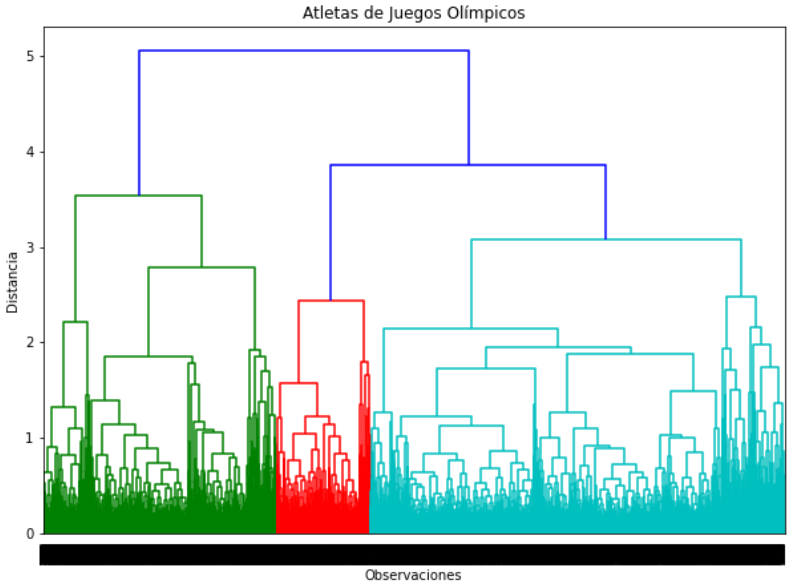
Métrica de distancia: `chebyshev`, Matriz: `Normalizada`, Resultado: `19 clústeres`.



Métrica de distancia: `cityblock`, Matriz: `Estandarizada`, Resultado: `8 clústeres`.



Métrica de distancia: `cityblock`, Matriz: `Normalizada`, Resultado: `3 clústeres`.



Con nuestros estos árboles podemos etiquetar a cada uno de los registros en el conjunto de datos ayudándonos de la clase `AgglomerativeClustering` y pasando como parámetros el número de clústeres, el método y la distancia que pasamos al hacer los dendrogramas, con esto se crea un objeto el cual lo podemos usar para la segmentación con su método `fit_predict` y pasando como parámetro la matriz de datos con la que se trabajara. Para esta parte del análisis se considerarán los clústeres obtenidos con la matriz estandarizada y la métrica euclidiana la cual nos da como resultado 6 clústeres. El primero de ellos con 2652 registros, el segundo con 1585, el tercero con 12773, el cuarto con 432, el quinto con 2435 y el sexto con 123 registros.

clusterH	
0	2652
1	1585
2	12773
3	432
4	2435
5	123

```
MJerarquico = AgglomerativeClustering(n_clusters=6, linkage='complete', affinity='euclidean')
MJerarquico.fit_predict(MEstandarizada)
MJerarquico.labels_
```

Con cada uno de los registros etiquetados se agrupa por clúster y luego se obtienen los centroides calculando la media para cada uno de los registros los cuales se muestran a continuación.

clusterH	Age	Height	Weight	Year	Medal	Sex	Season
0	26.215686	189.506787	92.305618	1999.095777	15.015083	0.991704	0.260935
1	35.200000	177.237224	76.307256	1989.716088	14.586751	0.863722	0.215142
2	23.950051	174.914429	68.901550	1989.028889	9.725202	0.697174	0.212166
3	24.481481	177.687500	72.480324	1922.717593	28.564815	0.974537	0.020833
4	22.373717	161.399179	53.262834	1994.613552	16.295688	0.090760	0.016838
5	48.495935	173.056911	75.398374	1984.813008	0.000000	0.991870	0.008130

Para cada uno de estos clústeres es pertinente hacer un análisis que describa de forma textual como es el comportamiento o las características que distinguen a cada uno de estos grupos.

Clúster 0.

Conformado por 2652 deportistas olímpicos con una edad promedio de 26 años, una altura de 189 m. y un peso de 92 kg. al momento de competir, la mayoría de ellos compitieron por el año 1999 en los juegos de verano, aunque son el grupo que más atletas de invierno contiene y prácticamente todos son de sexo masculino obteniendo un resultado bajo en cuanto a medallas.

Clúster 1.

Conformado por 1585 atletas olímpicos con una edad promedio de 35 años, una altura de 177 m. y un peso de 76 kg. al momento de competir, la mayoría de ellos compitieron por el año 1989 en los juegos de verano siendo de los más veteranos a día de hoy, gran parte del grupo son de sexo masculino y en general se obtuvo un resultado bajo en cuanto a medallas.

Clúster 2.

Conformado por 12773 atletas olímpicos siendo el grupo más grande atletas con una edad promedio de 24 años, una altura de 175 m. y un peso de 69 kg. al momento de competir, la mayoría de ellos compitieron por el año 1989 en los juegos de verano siendo de los más veteranos a día de hoy, gran parte de ellos son de sexo masculino sin embargo es el grupo que más mujeres contiene y en general se obtuvo un resultado bajo en cuanto a medallas.

Clúster 3.

Conformado por 432 deportistas olímpicos con una edad promedio de 24 años, una altura de 178 m. y un peso de 72 kg. al momento de competir, la mayoría de ellos compitieron por el año 1993 en los juegos de verano gran parte de ellos son de sexo masculino y en general es el grupo al cual se fue mejor en cuanto a medallas.

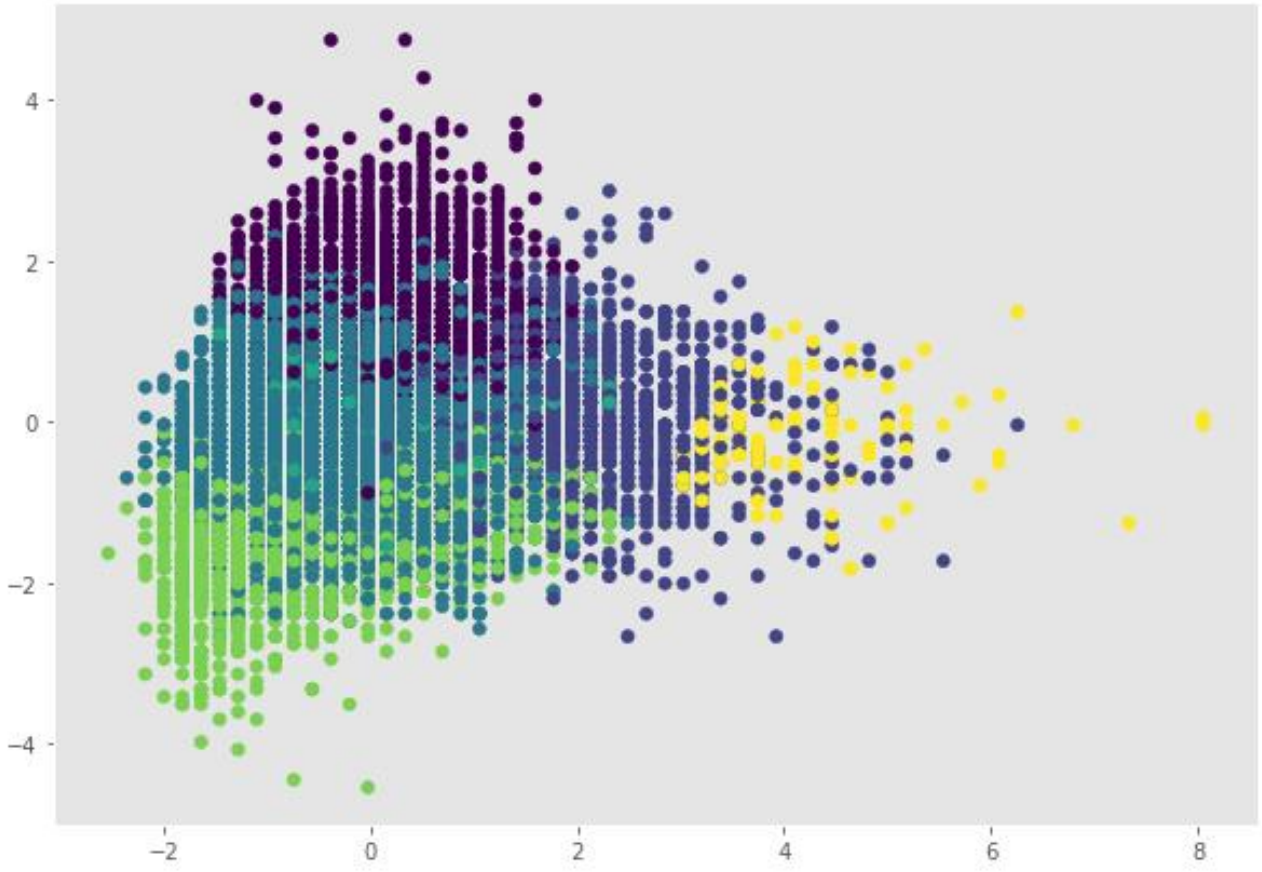
Clúster 4.

Conformado por 2435 deportistas olímpicos con una edad promedio de 22 años siendo el grupo más joven, una altura de 161 m. y un peso de 53 kg. al momento de competir, la mayoría de ellos compitieron por el año 1994 en los juegos de verano siendo de los más veteranos, gran parte de ellos son de sexo masculino y en general se obtuvo un resultado de los más bajos en cuanto a medallas.

Clúster 5.

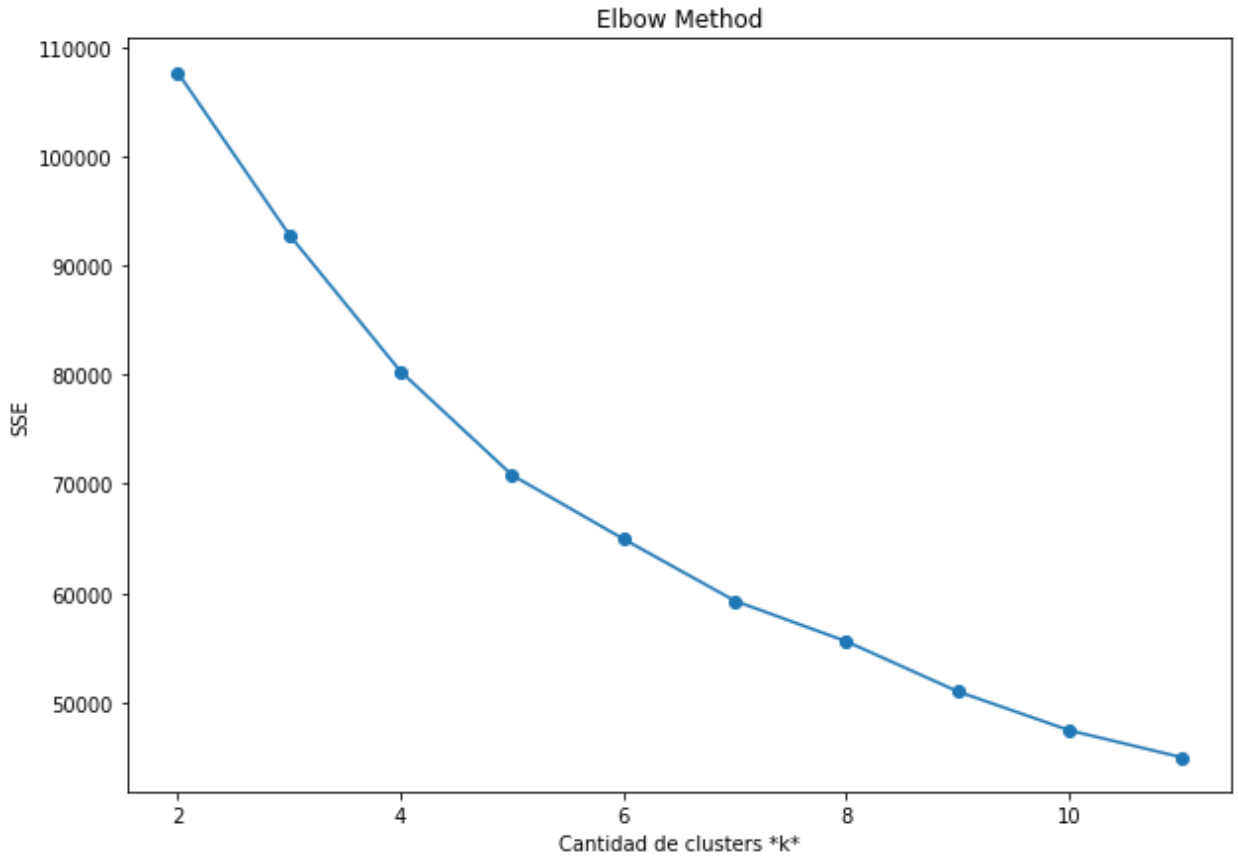
Conformado por 123 atletas olímpicos siendo el grupo más pequeño con una edad promedio de 48 años siendo el grupo más veterano, una altura de 173 m. y un peso de 75 kg. al momento de competir, la mayoría de ellos compitieron por el año 1985 en los juegos de verano siendo de los más veteranos también a día de hoy, prácticamente todos ellos son de sexo masculino y obteniendo el peor resultado ya que ninguno logro obtener una medalla.

Podemos además trazar un gráfico de dispersión para darnos una idea de como es que se distribuyen estos clústeres con ayuda de `matplotlib` y la matriz de datos estandarizada ya etiquetada para cada registro. Como nota aclaratoria el color que menos se ve es un verde fuerte, esto debido a que representa al grupo más pequeño de 123 registros.

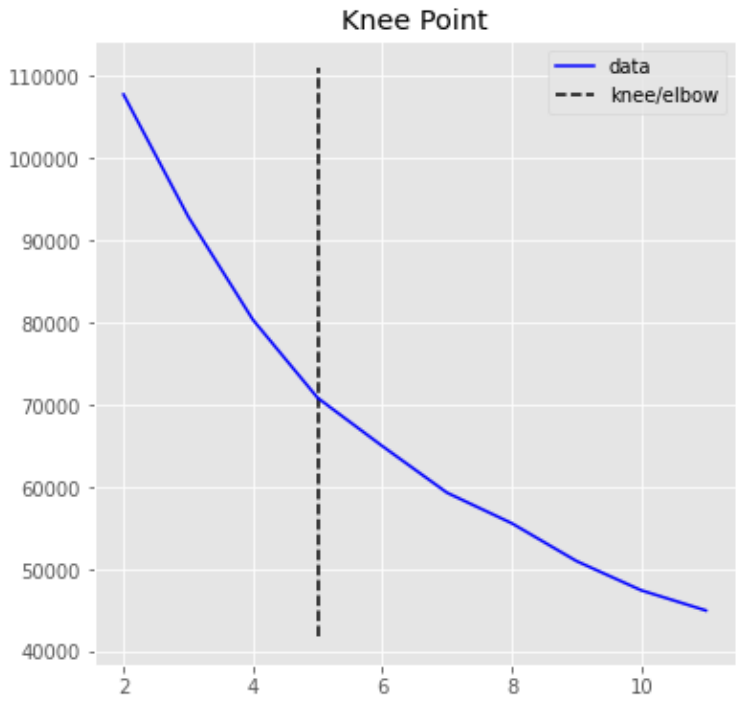


Comenzando con el clustering particional primero tenemos que establecer cual es el numero óptimo de clústeres a utilizar, podemos intuir por los arboles generados en el clustering jerárquico que este número pudiera estar entre 5 y 6 clústeres.

Para establecer bien este número es necesario ayudarnos en primera instancia del método del codo calculando la suma de las distancias al cuadrado desde cada uno de los registros a su centroide para cada una de las configuraciones, afortunadamente la clase de `KMeans` que nos proporciona la biblioteca de `sklearn` ya calcula estas sumas para la configuración de clústeres que se le da y basta con aplicar el modelo a la matriz de datos con el método `fit` para que estas sumas se dejen en el atributo `inertia_` el cual se ingresa a una lista ya que se hace para las configuraciones de 2 a 12 clústeres que es lo recomendado. Por ultimo se grafican estas sumas con ayuda de `matplotlib`.



Como podemos apreciar en esta ocasión el cambio abrupto de dirección que indicaría el número de clústeres adecuado no es tan claro pudiendo estar en 4, 5 o hasta 7 clústeres. En este caso el método de la rodilla del paquete de `Kneed` nos servirá para tener mayor seguridad de que el número de clústeres que se elija sea el adecuado. Basta con hacer un objeto de la clase `KneeLocator` el cual recibirá como parámetros el rango de clústeres, la lista de distancias para cada uno de esos clústeres, la forma de la curva (`convex`) y la dirección que tomara (`decreasing`) obteniendo que el numero adecuado de clústeres es de 5 y la grafica que se muestra a la derecha marcando este punto con una línea.



Ya con el número de clústeres definido, el siguiente paso seria aplicar el algoritmo con la clase de `KMeans` de `sklearn` para esta configuración, primero creando un objeto y luego calculándolo con el método `predict` para poder tener el atributo l`abels_` disponible el cual contiene una lista ordenada con las etiquetas de todos los clústeres del modelo, en esta ocasión como son 5 clústeres entonces se numerarán del 0 al 4. Esta es una lista que fácilmente se puede agregar como una columna más al `DataFrame` de la matriz de datos resultante de la reducción de características del modelo.

```
MParticional = KMeans(n_clusters=5, random_state=0).fit(MEstandarizada)
MParticional.predict(MEstandarizada)
MParticional.labels_
```

Con los registros etiquetados podemos agruparlos y contarlos con ayuda de pandas, de esta forma obtenemos que en el primer clúster se tienen 3452 elementos, en el segundo 4299, en el tercero 4289, en el cuarto 5470 y en el último 2490 registros de pacientes.

clusterP	
0	3452
1	4299
2	4289
3	5470
4	2490

Podemos además con ayuda de esta agrupación calcular los promedios por cada grupo para obtener a cada uno de sus centroides correspondientes obteniendo los siguientes resultados.

clusterP	Age	Height	Weight	Year	Medal	Sex	Season
0	25.042874	174.280417	70.139774	1994.362688	3.076477	0.656431	1.000000
1	23.315422	166.550128	57.963945	1995.197953	3.154222	0.003722	0.003722
2	26.918862	186.631383	85.917813	1997.382140	1.548146	0.967825	0.006995
3	24.971298	172.936563	67.488665	1977.634735	0.208410	0.999086	0.000000
4	25.510843	178.589960	75.151606	1986.533333	83.220884	0.710843	0.118876

Clúster 0.

Conformado por 3452 atletas olímpicos con una edad promedio de 25 años, una altura de 174 m. y un peso de 70 kg. al momento de competir, la mayoría de ellos compitieron por el año 1994 todos ellos en los juegos de invierno, la mayoría también son de sexo masculino, pero es uno de los grupos que más mujeres tiene y en general se obtuvo un resultado bajo en cuanto a medallas.

Clúster 1.

Conformado por 4299 atletas olímpicos con una edad promedio de 23 años siendo el grupo más joven, una altura de 167 m. y un peso de 58 kg. al momento de competir, la mayoría de ellos compitieron por el año 1995 en los juegos de verano prácticamente todos en el grupo son de sexo femenino y en general se obtuvo un resultado bajo en cuanto a medallas.

Clúster 2.

Conformado por 4289 atletas olímpicos con una edad promedio de 27 años siendo los más veteranos al competir, una altura de 187 m. y un peso de 86 kg. al momento de competir, la mayoría de ellos compitieron por el año 1997 en los juegos de verano, gran parte de ellos son de sexo masculino y en general se obtuvo un resultado bajo en cuanto a medallas.

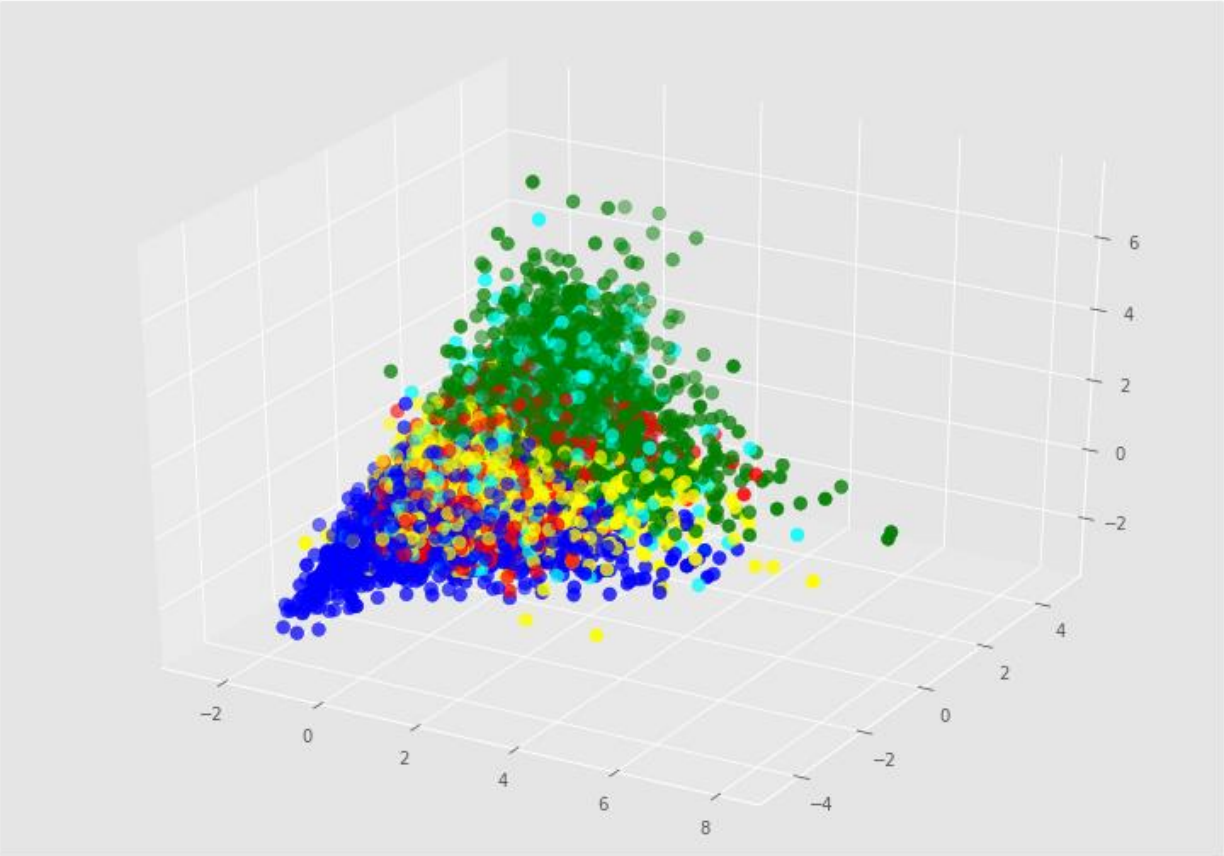
Clúster 3.

Conformado por 5470 deportistas olímpicos con una edad promedio de 25 años, una altura de 173 m. y un peso de 67 kg. al momento de competir, la mayoría de ellos compitieron por el año 1993 en los juegos de verano donde prácticamente todos ellos son de sexo masculino y en general se obtuvo un resultado bajo en cuanto a medallas.

Clúster 4.

Conformado por 2490 deportistas olímpicos con una edad promedio de entre 25 y 26 años siendo el grupo más joven, una altura de 179 m. y un peso de 75 kg. al momento de competir, la mayoría de ellos compitieron por el año 1986 en los juegos de verano siendo de los más veteranos a día de hoy, gran parte de ellos son de sexo masculino y en general se obtuvo el mejor resultado en cuanto a medallas donde el promedio cuenta con mínimo una medalla de plata.

El último paso del análisis es el generar una gráfica en la cual podemos ver un poco de la distribución de nuestros 5 clústeres de datos la cual se crea partiendo de la matriz de datos estandarizada y se ve de la siguiente forma.



Conclusiones.

En esta practica se lograron apreciar un poco mejor las diferencias entre el clustering particional y el jerárquico sobre todo en el numero de registros ya que originalmente se tenían 270, 000 numero que se redujo a poco mas de 200, 000 cuando se eliminaron todos los registros con mínimo un atributo desconocido e inmediatamente al intentar ejecutar el algoritmo Jerárquico se desbordo la memoria RAM de Google Colab por lo cual es importante primero considerar el poder de computo con el que se cuenta para ajustar una cantidad de registros adecuada.

Mientras que el algoritmo Jerárquico tomaba alrededor de 10 minutos para generar cualquier árbol, al algoritmo particional le tomaba segundos notándose la inestabilidad del algoritmo jerárquico aun cuando el numero de registros se redujo de 200, 000 a 20, 000, es decir el 10% de la población original. Esta reducción si bien puede ayudar al rendimiento no lo hace a tal punto de hacer viable el clustering jerárquico, por el contrario, hay que tener en cuenta que para poder utilizar la técnica particional se tiene que conocer cuál es el número de clústeres con el que queremos terminar, métrica que es muy importante ya que si se aplica mal podría llevar a tener sesgos en el análisis de los clústeres pero que requiere de un procesamiento extra para realizarse (Método del codo o la rodilla) aunque en la práctica este procesamiento tampoco resulto muy problemático tomando segundos en realizarse.

Otra opción que podría aumentar el rendimiento de nuestros modelos es hacer una reducción de variables estricta para el modelo eliminando características de los registros los cuales pudieran presentar redundancia en los datos.

Fuentes.

Datos: 120 years of Olympic history: athletes and results | Kaggle

sklearn.cluster.KMeans. (2022). Scikit-Learn. 2022, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

pandas documentation — pandas 1.4.1 documentation. (2022). Pandas. Recuperado 2022, de https://pandas.pydata.org/docs/index.html#