TUIA

MINERÍA DE DATOS

TRABAJO PRÁCTICO Nº 1

Integrantes:
López, Eugenio
Pistelli, Pablo

Octubre 2023

Objetivo

El objetivo de este trabajo práctico es integrar los conocimientos adquiridos en las unidades 2 (reducción de dimensionalidad) y 3 (clustering) en un problema real asociado a los cultivos.

Carga del dataset y análisis exploratorio de datos

```
1 ruta_al_csv = '/content/Crop_recommendation.csv'
2 df = pd.read_csv(ruta_al_csv)
```

```
1 df.head()
```

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice

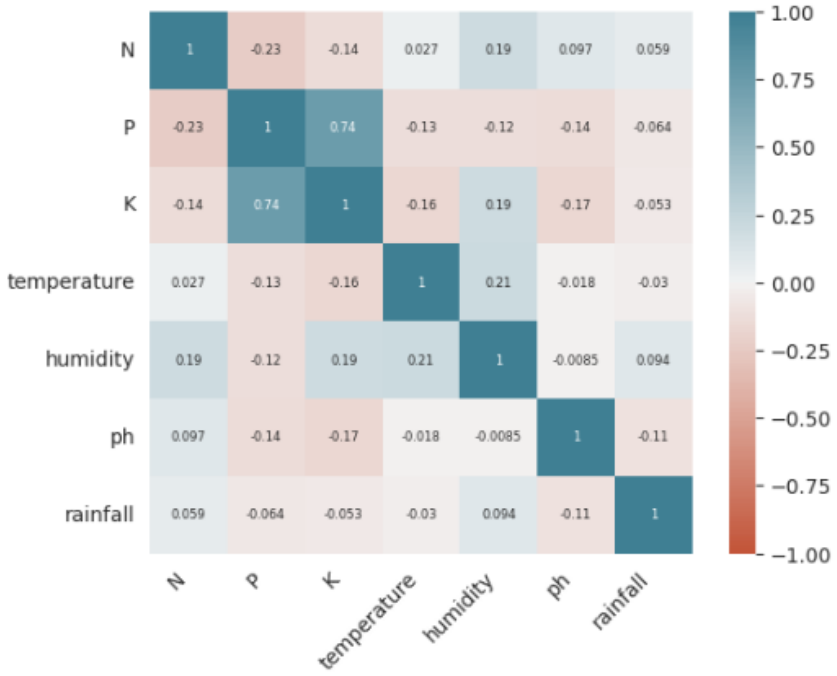
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   N            2200 non-null   int64  
 1   P            2200 non-null   int64  
 2   K            2200 non-null   int64  
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object 
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

Notamos que 'label' es una variable categórica. Los demás atributos son numéricos.

Además observamos que no hay valores nulos en el DataFrame.

Matriz de correlación



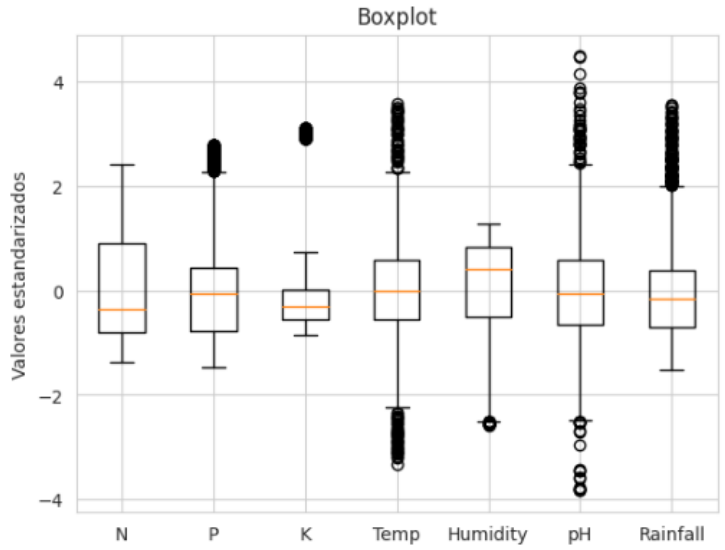
A partir de la matriz de correlación, se puede notar que los atributos 'P' y 'K' son aquellos que más se correlacionan, con un valor de 0.74. Los demás atributos no poseen una correlación tan marcada.

Estandarización de datos

Realizamos la estandarización de los datos basada en la desviación estándar

```
df_sub = df.drop(['label'], axis=1)
df_sub_std = (df_sub-df_sub.mean())/df_sub.std()
```

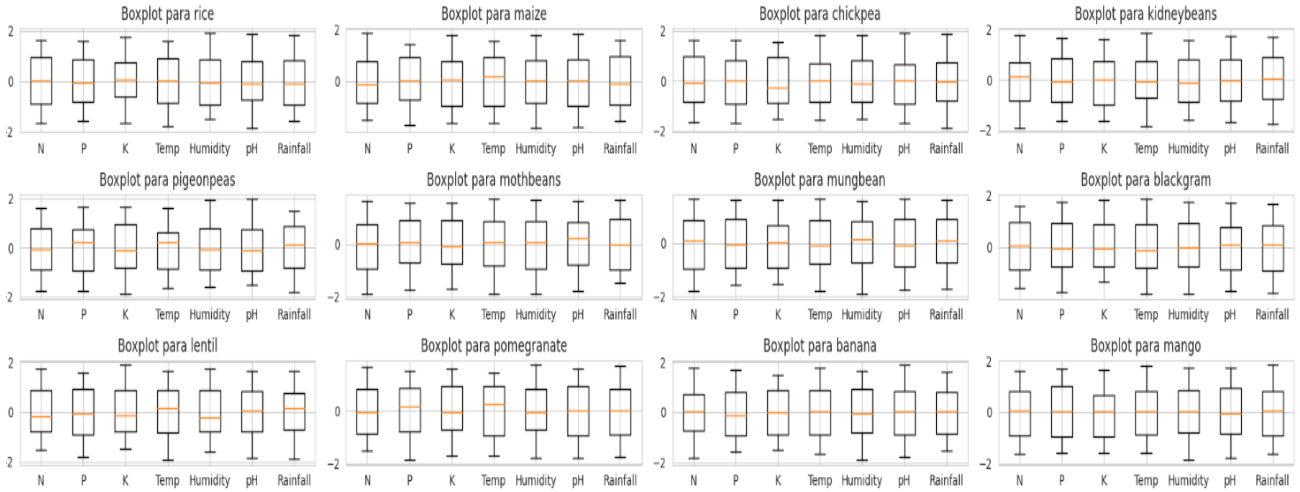
Boxplot DataFrame completo

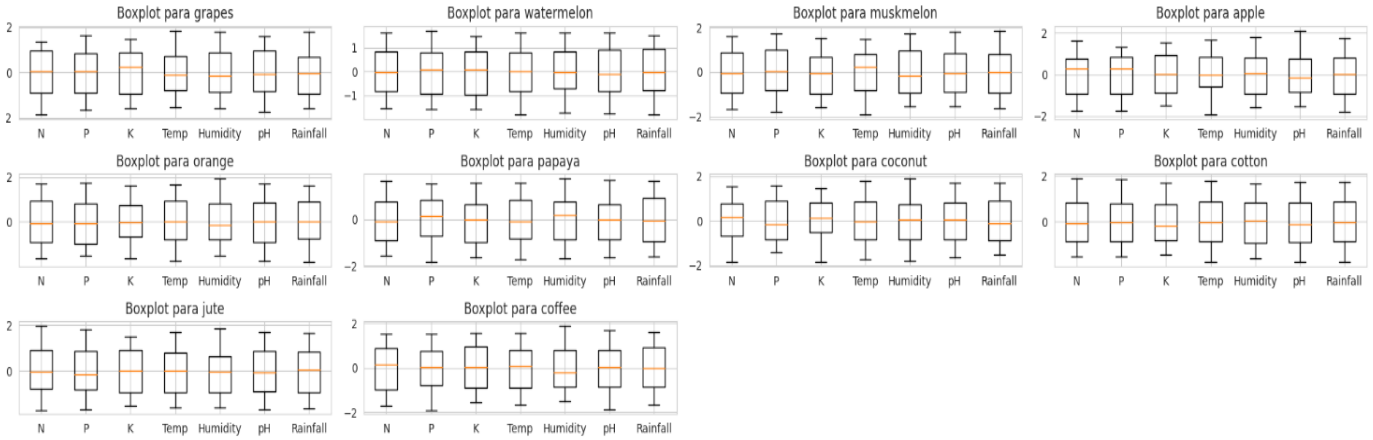


Notamos que aparecen una serie de outliers en algunas características. Pensamos que puede deberse a que están todos los cultivos juntos en el mismo gráfico y algunos se pueden comportar de manera distinta a otros, por eso pueden visualizarse esos outliers.

Gráfico boxplot para cada label

Lo que buscamos es ver si existen outliers para cada label por separado. De esta forma, si existieran, analizaremos un criterio para removerlos. De lo contrario continuaremos sin realizar modificaciones.





Observando los boxplots, no se han visualizado outliers para cada label por separado, por lo tanto decidimos continuar sin realizar modificaciones.

PCA

```python
# Obtener todas las componentes principales
pca = PCA(n_components=df_sub.shape[1])

pca_features = pca.fit_transform(df_sub_std)

# PC dataframe
pca_df = pd.DataFrame(
    data=pca_features,
    columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7'])
pca_df['label'] = df['label']
```

```python
pca_df.head()
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	label
0	-0.582737	-0.844394	1.373031	-1.613762	0.308154	-0.095975	-0.025234	rice
1	-0.474527	-0.784716	1.251893	-1.792355	1.107493	-0.532134	-0.280479	rice
2	-0.633924	-0.694365	1.179064	-1.817692	2.522690	-0.538428	-0.105943	rice
3	-1.047682	-1.087411	1.393035	-0.982177	1.448452	-0.656779	0.275209	rice
4	-0.873059	-0.658523	1.455354	-2.334481	1.959188	-0.317952	0.052728	rice

Eigenvectors

	Eigenvalues	Proporción de variancia explicada	Proporción acumulado de variancia explicada
0	1.931218	0.275888	0.275888
1	1.293910	0.184844	0.460733
2	1.076509	0.153787	0.614520
3	1.022891	0.146127	0.760647
4	0.805928	0.115133	0.875780
5	0.676562	0.096652	0.972431
6	0.192981	0.027569	1.000000

Gráfico de varianza acumulada

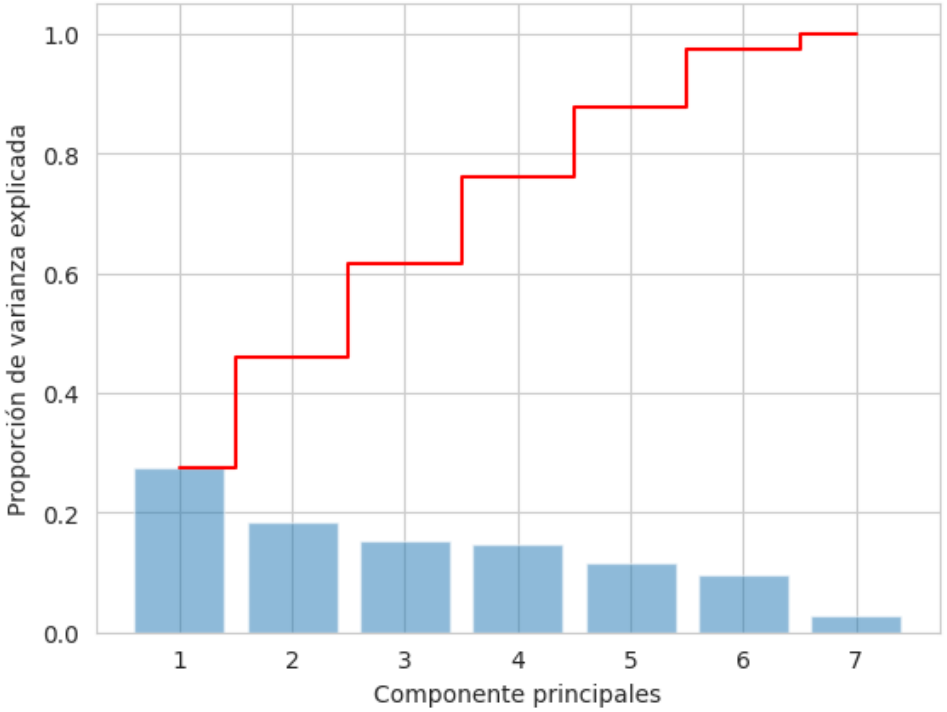
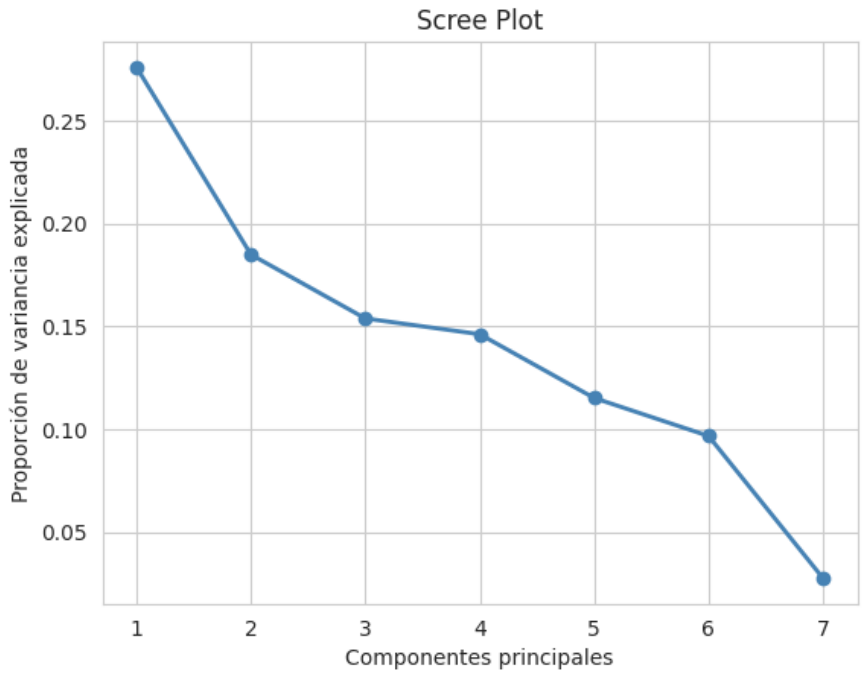


Gráfico del codo

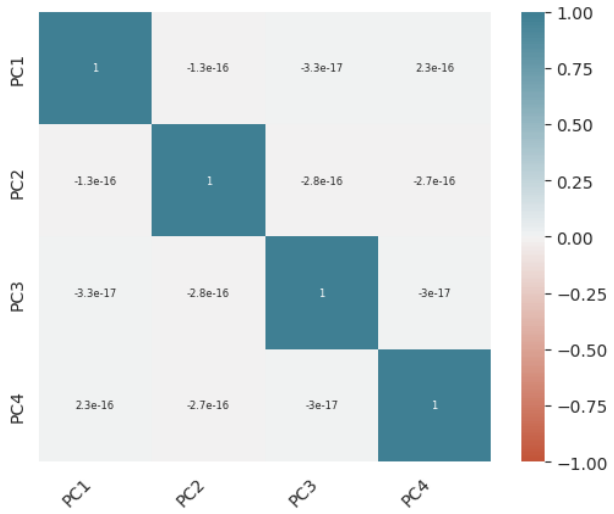


Criterios de selección de componentes principales:

- Proporción de variancia acumulada (~75% -80%)
- Criterio de Kaiser (eigenvalues > 1)
- Gráfico del codo (Scree)

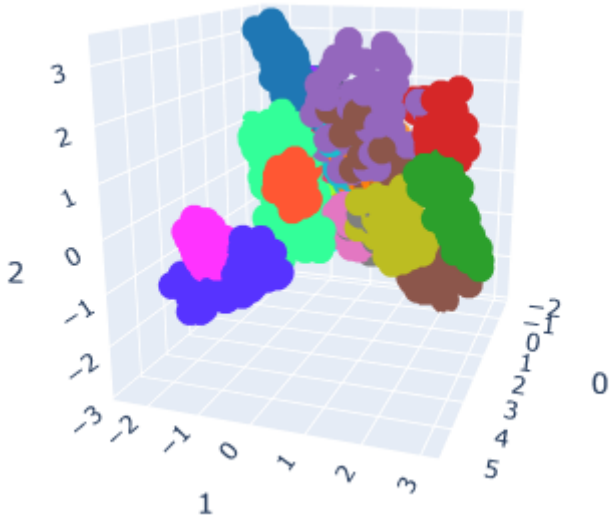
Al observar el gráfico del codo se logra visualizar un quiebre en el PC4. Además, considerando el criterio de Kaiser y la proporción de variancia acumulada definimos seleccionar PC1 - PC2 - PC3 - PC4, ya que son aquellos con eigenvalues > 1 y además acumulan el 76% de la variancia.

Matriz de correlación entre PC seleccionados



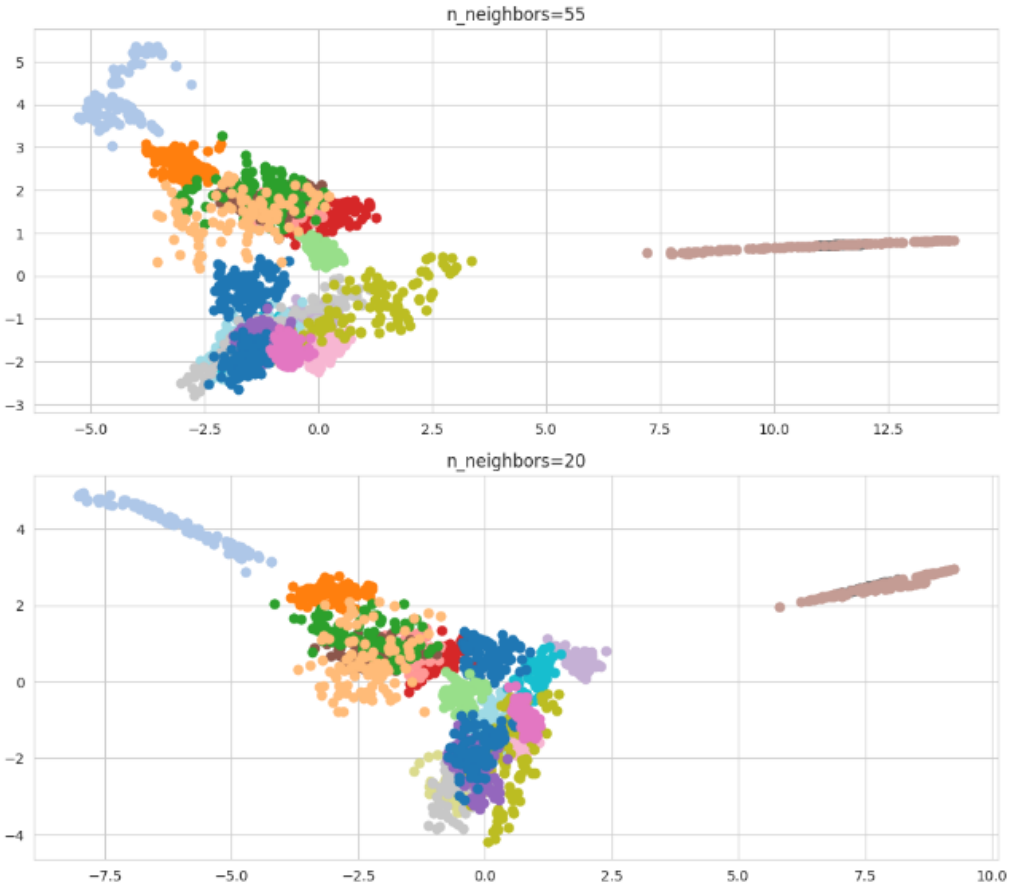
Vemos que la matriz de correlación entre las PC seleccionadas es una matriz diagonal, por lo tanto podemos decir que no tienen ninguna relación las PC entre ellas.

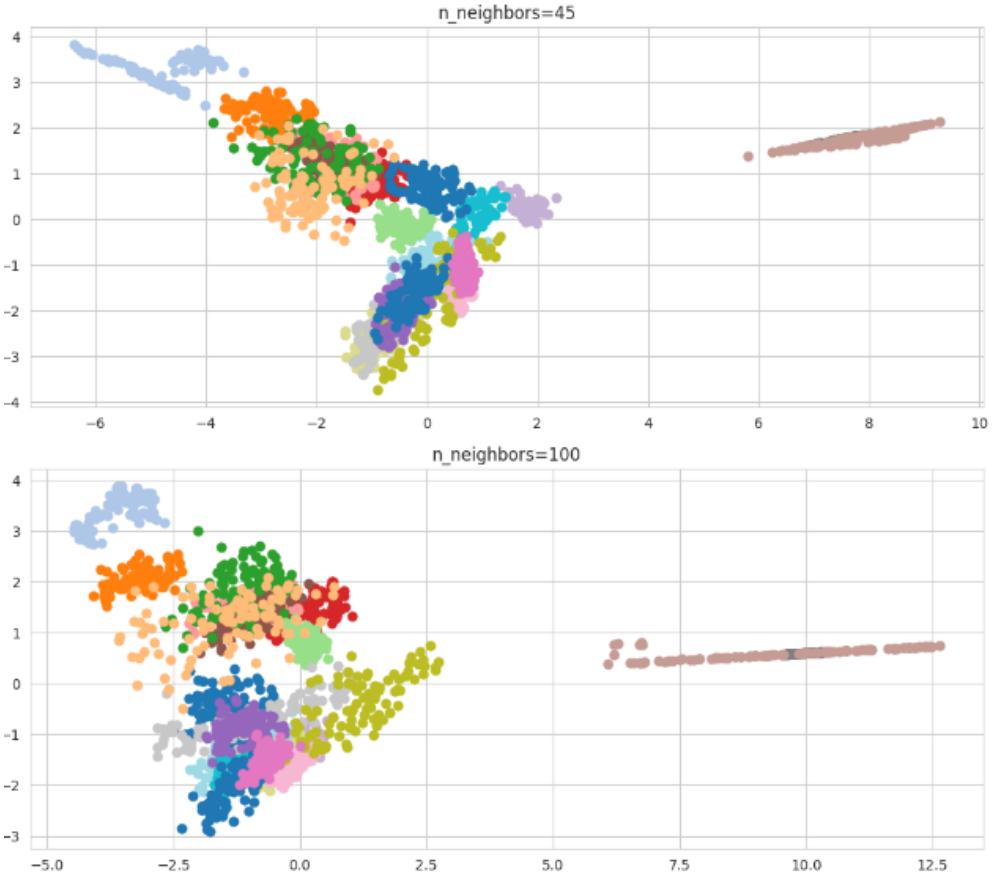
Scatter plots



Isomap

Analizamos isomap para distintos valores de n_neighbors y así chequear los diferentes resultados.


n_neighbors=55
n_neighbors=20


n_neighbors=45
n_neighbors=100

Al modificar el valor de n_neighbors se modifica la forma en que se construye el grafo de vecinos cercanos que se utiliza para calcular las distancias geodésicas. Por lo tanto, afecta la representación de las dimensiones reducidas que Isomap produce.

- Disminuir n_neighbors puede llevar a una representación en dimensiones reducidas que es más sensible a los detalles locales y al ruido en los datos, pero puede ser más rápida de calcular computacionalmente.
- Al aumentar n_neighbors se observa que los puntos están más dispersos, funcionará mejor para capturar estructuras globales en los datos y por lo tanto es más robusto frente al ruido. El costo computacional es mayor.

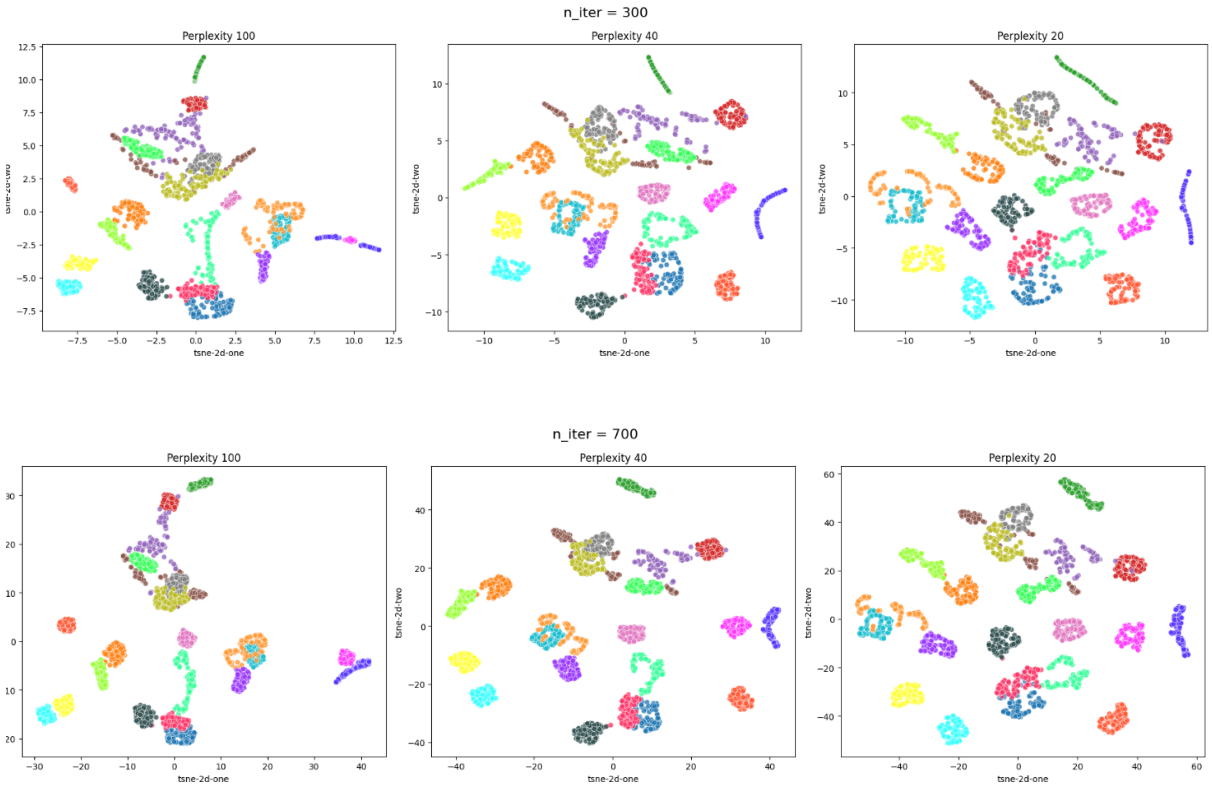
Un valor óptimo sería n_neighbors=55

Utilizamos n_components=2 ya que graficamos en 2D.

---

t-SNE

Analizamos t-SNE para distintos valores de perplejidad y de número de iteraciones y así chequear los diferentes resultados.



Notamos que en el algoritmo t-SNE, cambiar los valores de n_iter y perplexity puede tener un impacto significativo en los resultados y en el rendimiento del algoritmo:

n_iters:

- Aumentar el número de iteraciones puede ayudar a mejorar la calidad de la representación final ya que permite que el algoritmo tenga más tiempo para encontrar una representación óptima. Sin embargo, aumenta bastante el tiempo de cálculo.
- Disminuir n_iter hará que el algoritmo se ejecute más rápidamente, pero puede resultar en representaciones subóptimas si no se ejecuta el tiempo suficiente para converger. Puede ser útil si necesitas resultados rápidos pero estás dispuesto a aceptar una ligera pérdida en la calidad de la representación.

perplexity:

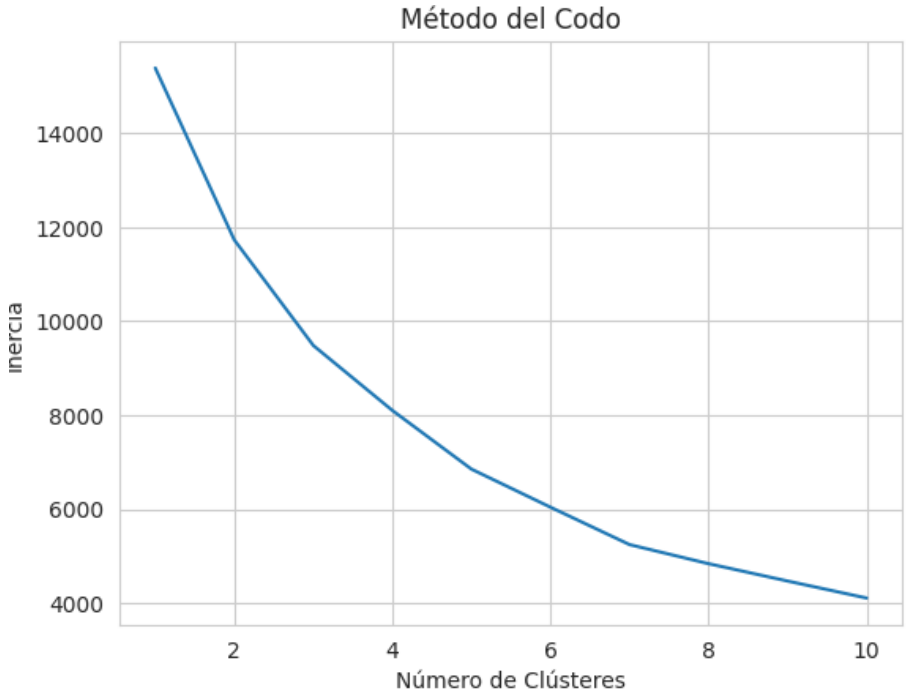
- Aumentar perplexity hace que cada punto considere más vecinos cercanos, por lo tanto, aumenta la influencia de los puntos distantes en el cálculo de las similitudes y puede hacer que el algoritmo ignore estructuras locales.
- Disminuir perplexity hace que cada punto considere menos vecinos cercanos y se centre más en las estructuras locales. Esto puede ser útil para resaltar detalles finos en los datos, pero también puede hacer que el algoritmo sea más sensible al ruido.

A fin de cuentas, decidimos optar por n_ter=700 y perplexity=40.

Con estos parámetros se obtuvieron los mejores resultados.

K-Means

Cálculo del diagrama de codo



El objetivo es identificar un punto en el gráfico donde la disminución en la suma de las distancias intraclúster (inercia) comienza a disminuir de manera significativamente más lenta. Este punto se denomina "codo" y sugiere el número óptimo de clústeres para el conjunto de datos.

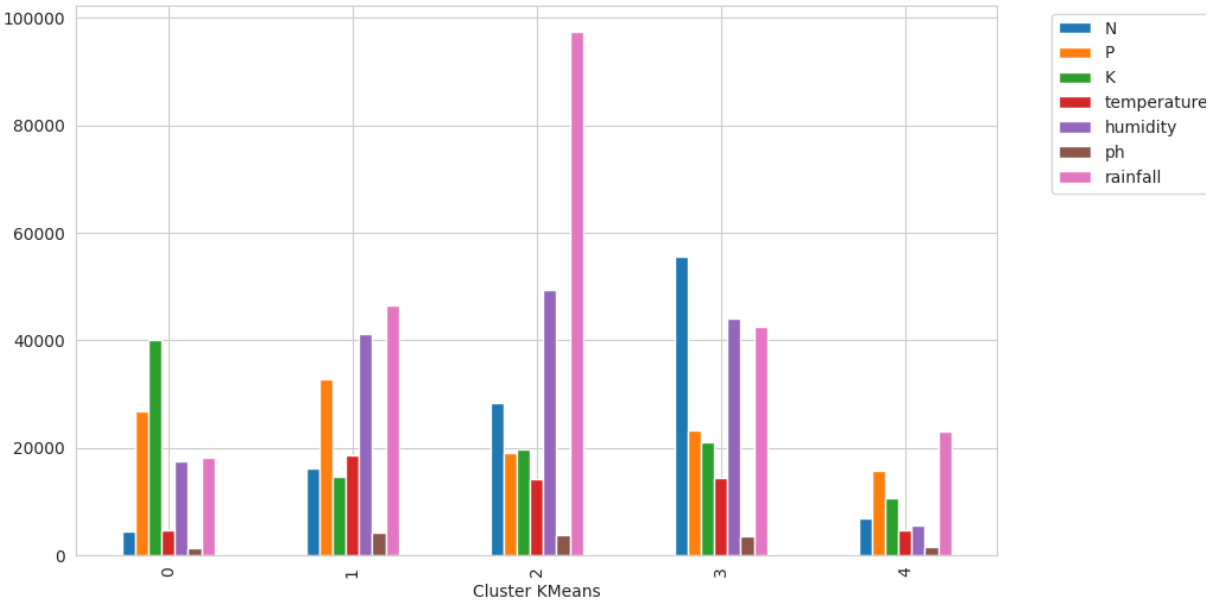
En este caso, no se visualiza una disminución tan marcada pero notamos que a partir del cluster 5 se observa un leve declive en la curva.

```
# Modelo con n_clusters=5
kmeans = KMeans(n_clusters=5)
kmeans.fit(X_scaled) #Entrenamos el modelo

# El metodo labels_ nos da a que cluster corresponde cada observacion
df_cluster = df_sub.copy()
df_cluster['Cluster KMeans'] = kmeans.labels_
```

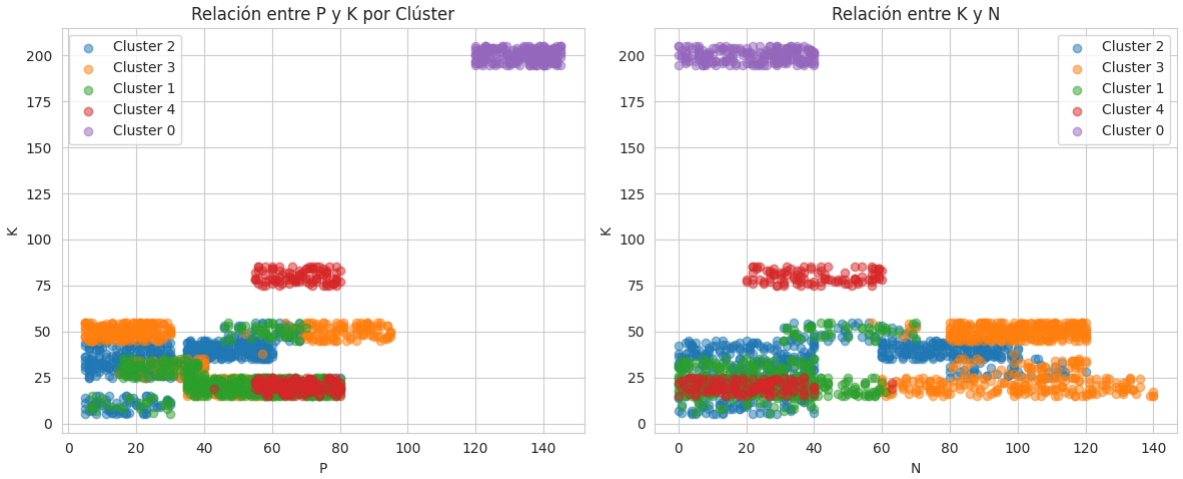
```
1 df_cluster.head()
```

	N	P	K	temperature	humidity	ph	rainfall	Cluster KMeans
0	90	42	43	20.879744	82.002744	6.502985	202.935536	2
1	85	58	41	21.770462	80.319644	7.038096	226.655537	2
2	60	55	44	23.004459	82.320763	7.840207	263.964248	2
3	74	35	40	26.491096	80.158363	6.980401	242.864034	2
4	78	42	42	20.130175	81.604873	7.628473	262.717340	2

Observamos cuán influyente es cada variable en nuestros clusters.



Por ejemplo, vemos que en el cluster 2 tenemos la mayor cantidad de observaciones de 'rainfall' y en el cluster 3 la mayor cantidad de 'N'. Por otro lado, en el cluster 4 tenemos la menor cantidad de observaciones.



Observando relaciones entre algunas variables, por ejemplo vemos que en el cluster 2 tenemos muy bajas cantidades de P y K y en el cluster 0 tenemos las más altas de estos dos, pero muy bajas de N.
El cluster 3 presenta bajas cantidades de K pero muy altas de N.

Para darnos una idea de manera gráfica sobre cómo quedaron conformados los clusters y debido a que tenemos muchas características y no es posible hacer un gráfico que las represente a todas, realizamos el análisis de componentes principales PCA.
Si se armaron grupos definidos deberíamos poder visualizar grupos separados a ojo y así confirmar que KMeans categorizó bien y no quedaron grupos superpuestos.

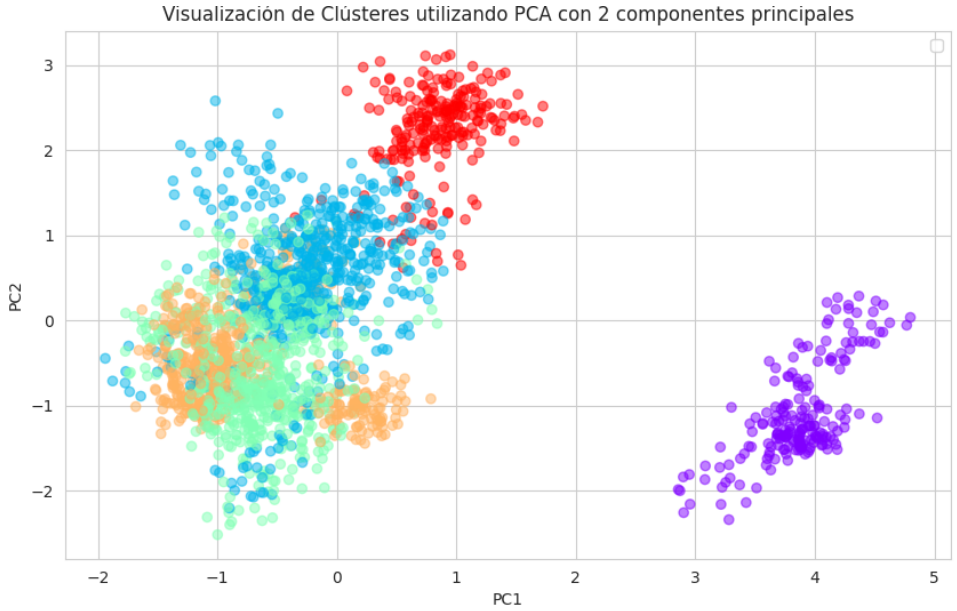
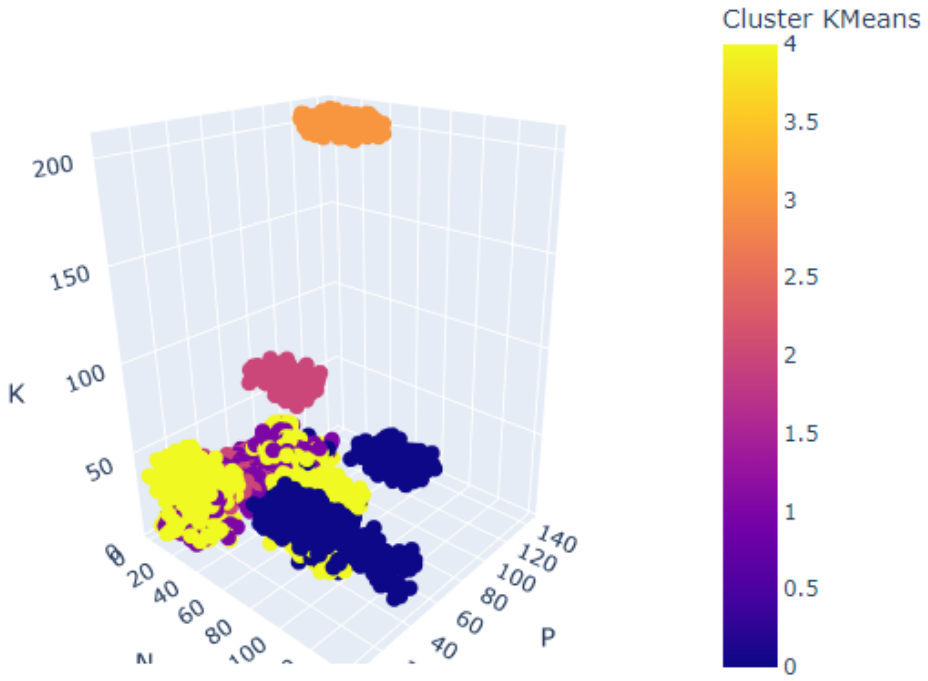


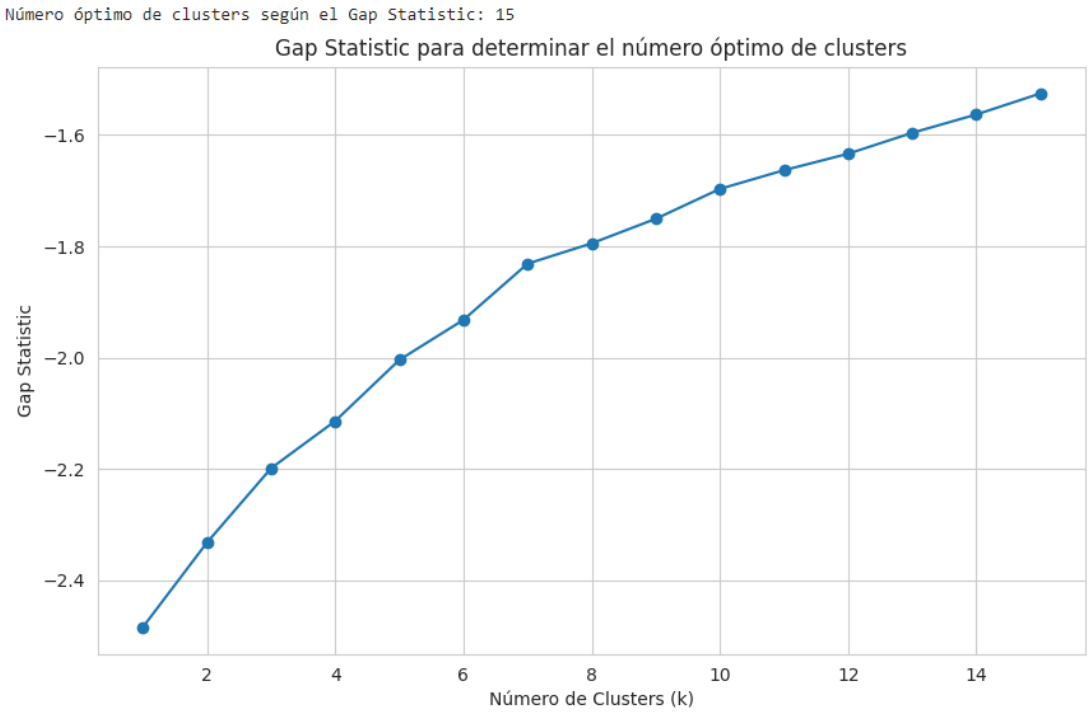
Gráfico 3D utilizando los atributos N, P y K



Notamos que hay algunos grupos definidos, como el cluster 3.
Muchos otros están superpuestos, por lo tanto KMeans no pudo realizar un agrupamiento o categorización satisfactoria de los datos o elegimos mal el número de clusters.

Gap Statistic

Utilizamos GAP para conocer el número óptimo de clusters para el modelo



Utilizando OptimalK de la librería gap_statistic

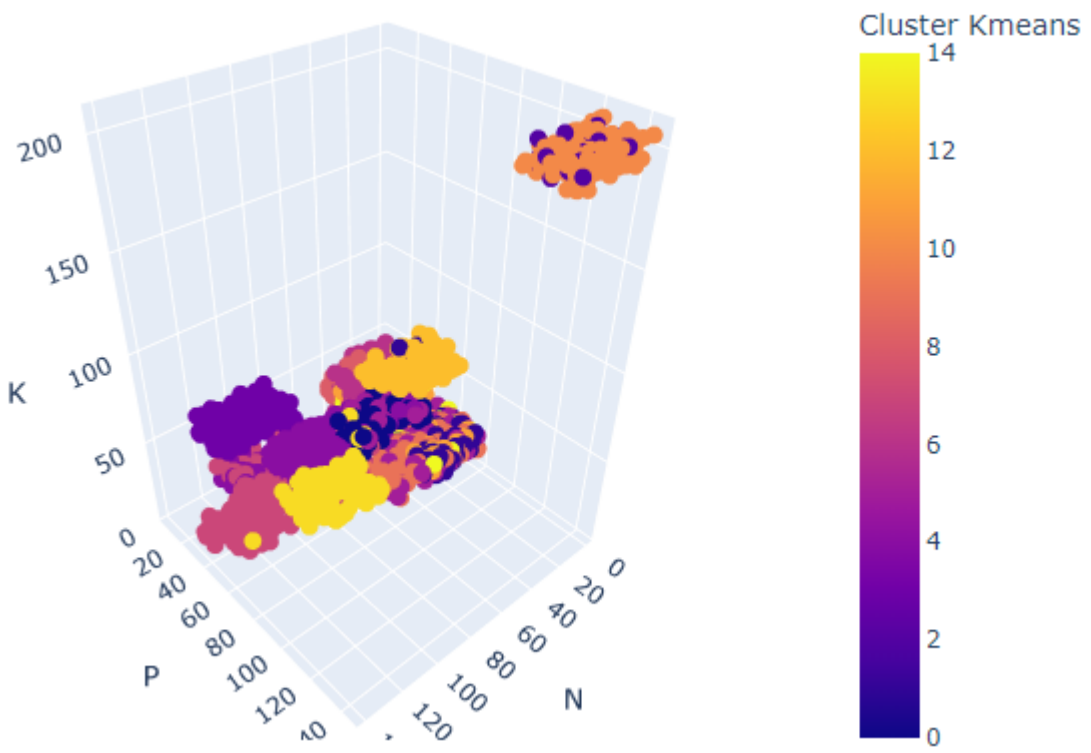
```
from gap_statistic import OptimalK
gs_obj = OptimalK(n_jobs=1, n_iter= 100)
n_clusters = gs_obj(X_scaled, n_refs=50, cluster_array=np.arange(1, 15))
print('Optimal number of clusters: ', n_clusters)
```

```
Optimal number of clusters:  14
```

Obtuvimos, mediante el uso de GAP, el número óptimo de clusters, que estará entre 14 y 15

Creamos el modelo para n_clusters=15

```
# Modelo con n_clusters=15
kmeans_15 = KMeans(n_clusters=15)
kmeans_15.fit(X_scaled) #Entrenamos el modelo

# El metodo labels_ nos da a que cluster corresponde cada observacion
df_cluster_15 = df_sub.copy()
df_cluster_15['Cluster Kmeans'] = kmeans_15.labels_
```

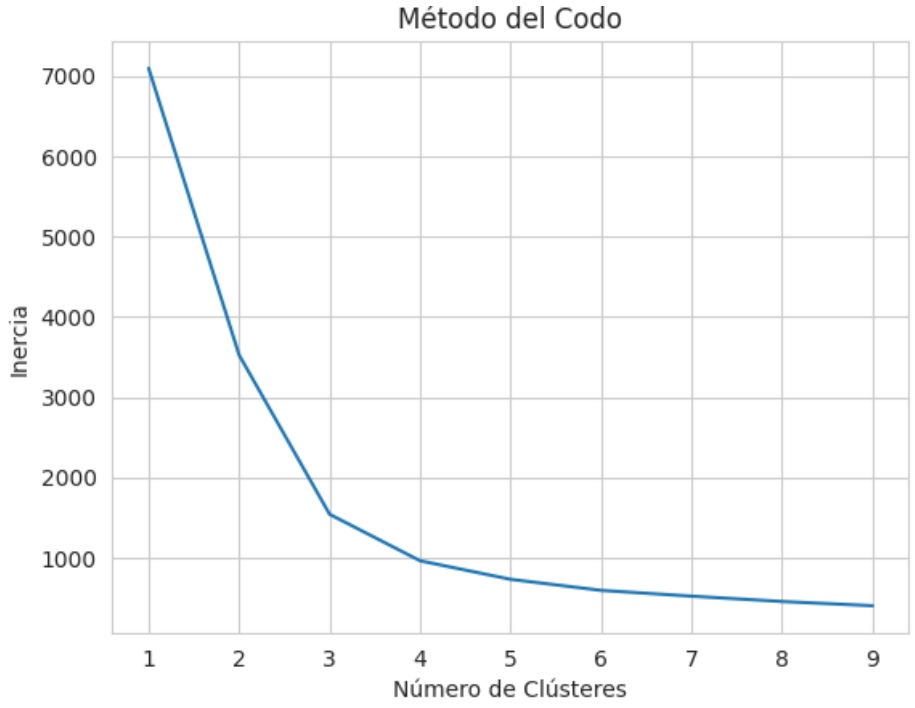


Aún así, construyendo el modelo con 15 clusters, el resultado de KMeans a la hora de categorizar los datos fue bastante malo.

Pensamos que esto puede deberse a que los datos están desbalanceados y a que puede haber características que están haciendo más fuerte ese desbalance.
Para saciar esto, procedemos a realizar PCA para reducir dimensionalidad y aplicar el modelo nuevamente pero ahora a las componentes principales.

Aplicando PCA

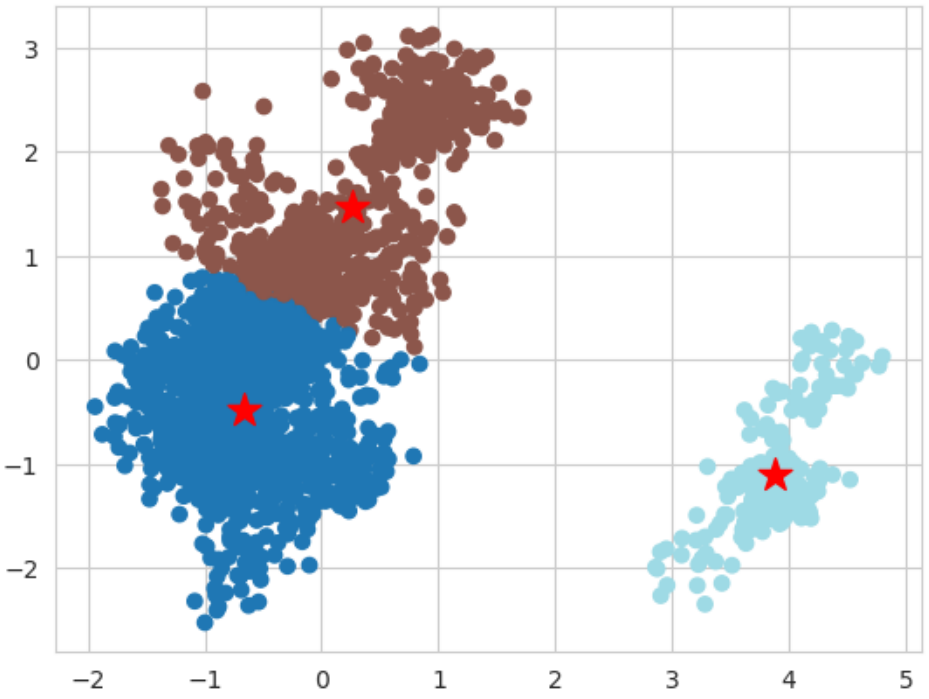
```
# Modelo para 2 componentes
pca_kmeans = PCA(n_components=2)
componentes_principales2 = pca_kmeans.fit_transform(df_sub_std)
```



A partir del método del codo visualizamos que en el cluster número 3 ocurre un quiebre pronunciado.

```
kmeans_pca = KMeans(n_clusters=3)
kmeans_pca.fit(componentes_principales2)
```

Visualizamos el clustering con sus centroides:



```
1 # Modelo para 3 componentes
2 pca_kmeans_3d = PCA(n_components=3)
3 componentes_principales3 = pca_kmeans_3d.fit_transform(df_sub_std)
```

Gráfico 3D para 3 clusters

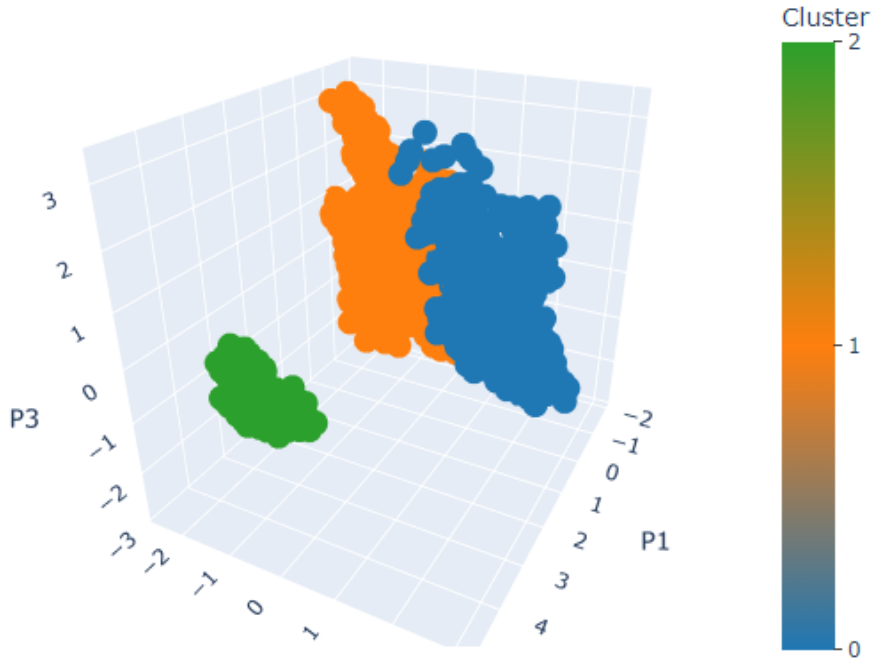
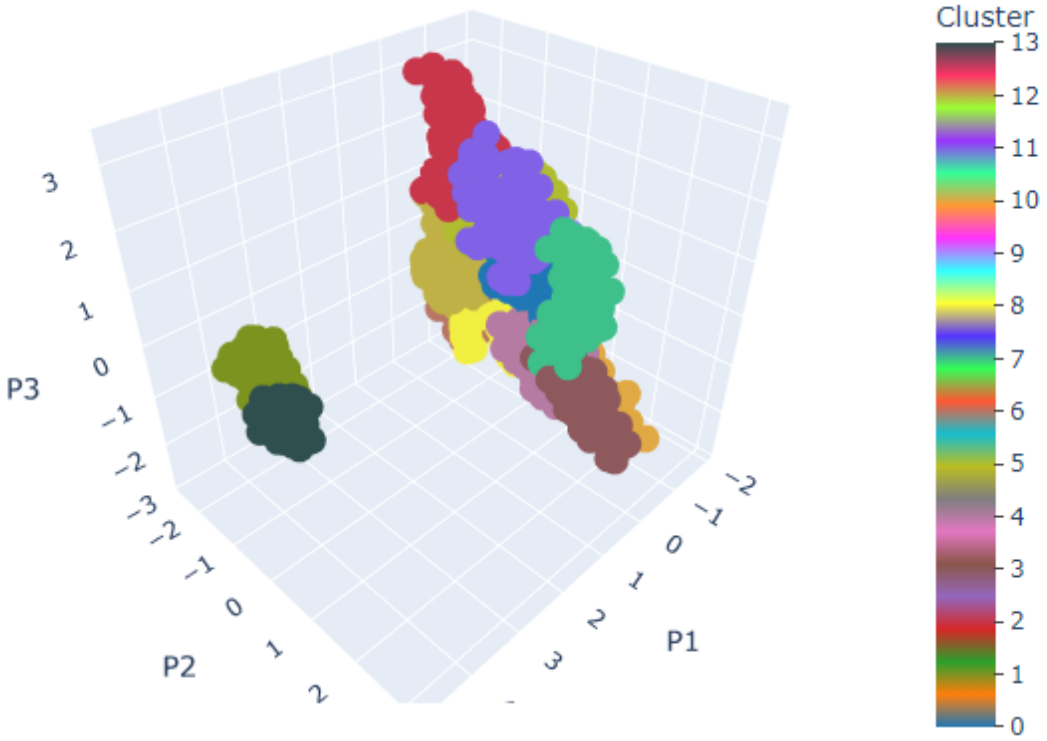


Gráfico 3D para 14 clusters

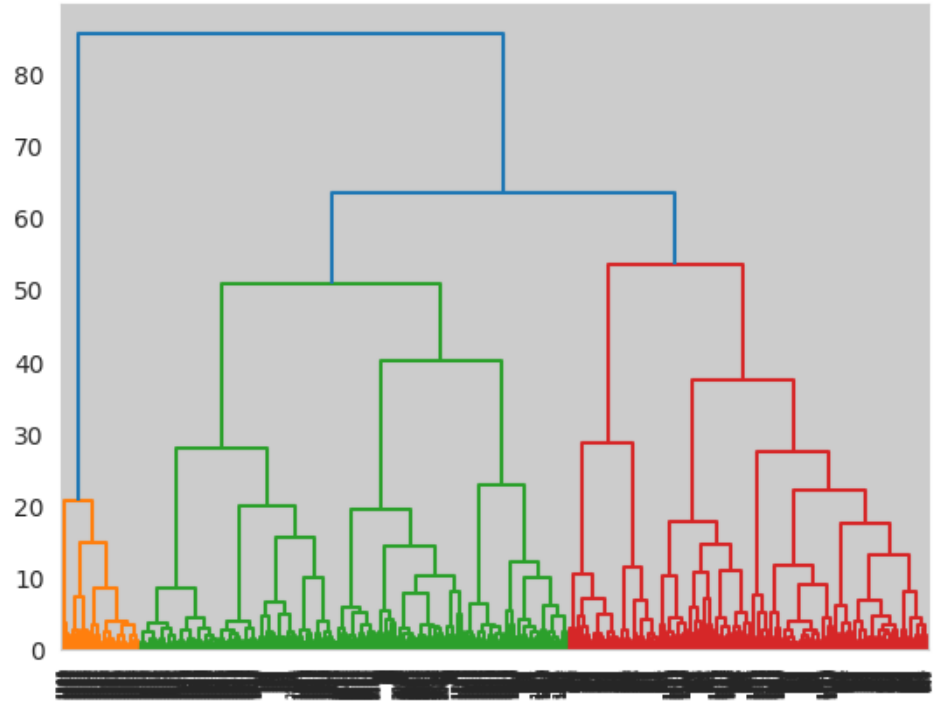


Aplicando PCA y luego realizando K-Means a esas componentes principales obtenidas pudimos observar como los datos se han podido categorizar de una manera mucho mas fuerte.
El método del codo nos arrojó un óptimo de 3 clusters, donde visualizamos que se agruparon bien los datos en esos 3 clusters, sin superponerse casi en su totalidad.
Decidimos aplicar K-Means con 14 clusters a las componentes principales, ya que era el número óptimo de clusters que arrojaba GAP anteriormente. En este caso pudimos ver que también se han categorizado muy bien.
Claramente, utilizando PCA previo a KMeans, se pudo realizar una mucho mejor categorización de los datos.

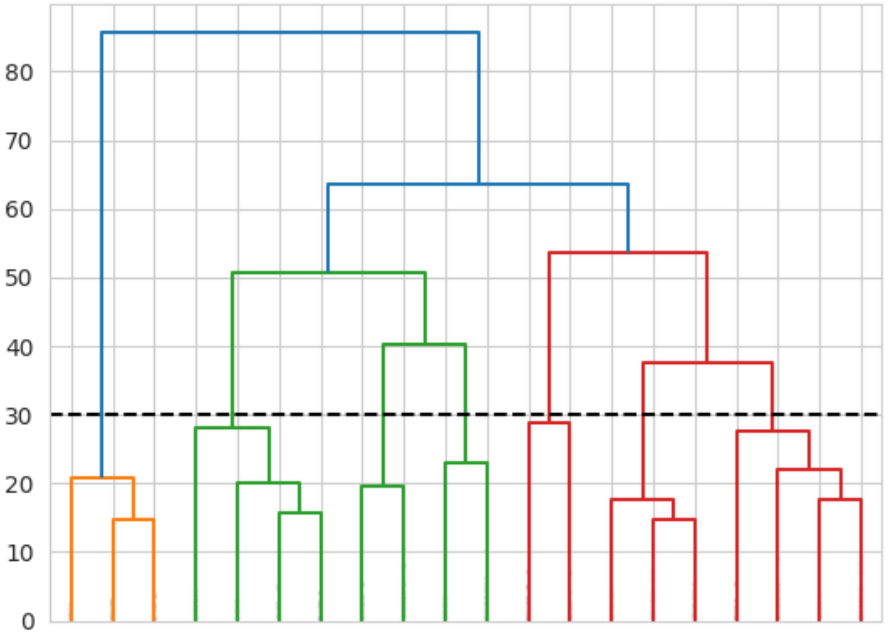
Clustering Jerárquico

```
# Armamos la matriz de enlace con los datos normalizados
Z = linkage(X_scaled, "ward")
```

```
# Creamos el dendrograma
dendrogram(Z)
plt.show()
```

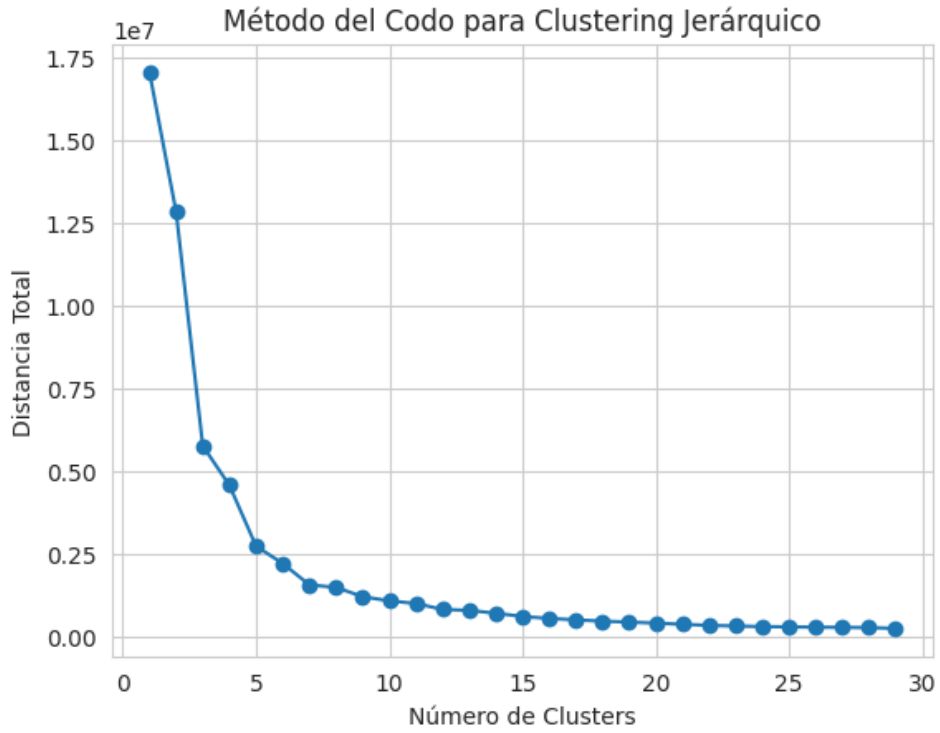


Analizamos un dendrograma un poco más limpio, truncándolo en los últimos 20 grupos, sabiendo que no vamos a definir más de 20 clusters



Definimos cortar el dendrograma en y=30, por lo que vamos a calcular un total de 7 clusters.

Método del codo para clustering jerárquico



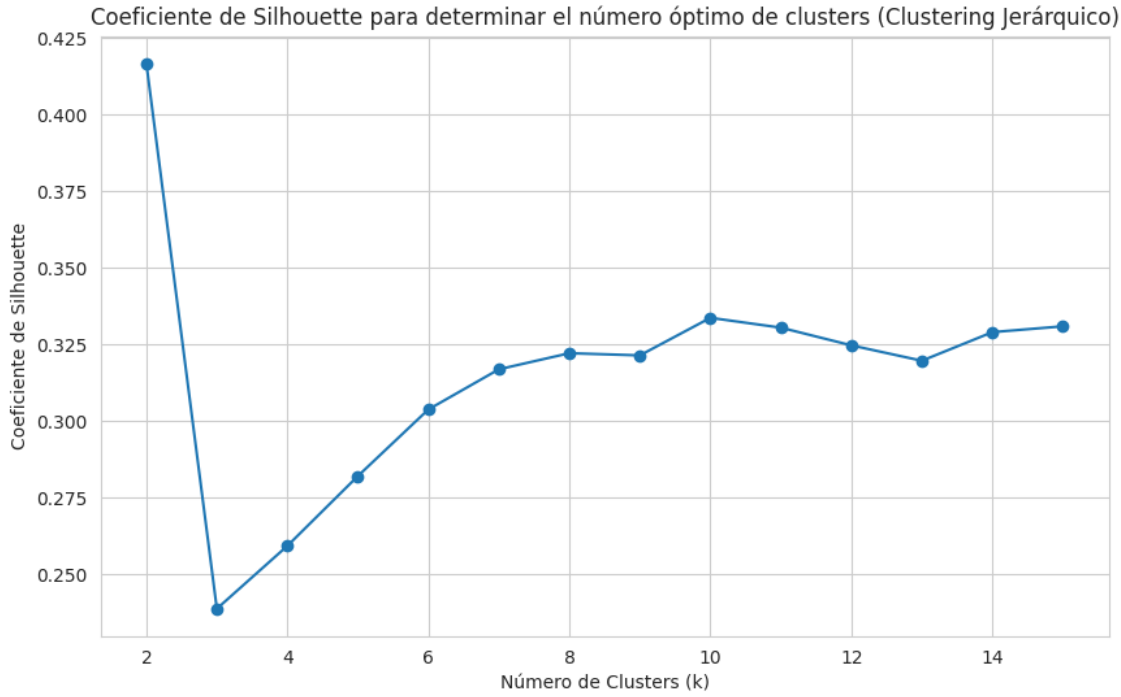

```
n_clusters = 7
clustering = AgglomerativeClustering(n_clusters=n_clusters)

cluster_assignments = clustering.fit_predict(X_scaled)

df_cluster['Cluster Jer'] = cluster_assignments

df_cluster.head()
```

	N	P	K	temperature	humidity	ph	rainfall	Cluster KMeans	Cluster Jer
0	90	42	43	20.879744	82.002744	6.502985	202.935536	2	5
1	85	58	41	21.770462	80.319644	7.038096	226.655537	2	5
2	60	55	44	23.004459	82.320763	7.840207	263.964248	2	5
3	74	35	40	26.491096	80.158363	6.980401	242.864034	2	5
4	78	42	42	20.130175	81.604873	7.628473	262.717340	2	5

A partir del dendrograma y del método del codo aplicado a clustering jerárquico, decidimos que el óptimo de clusters es 7.

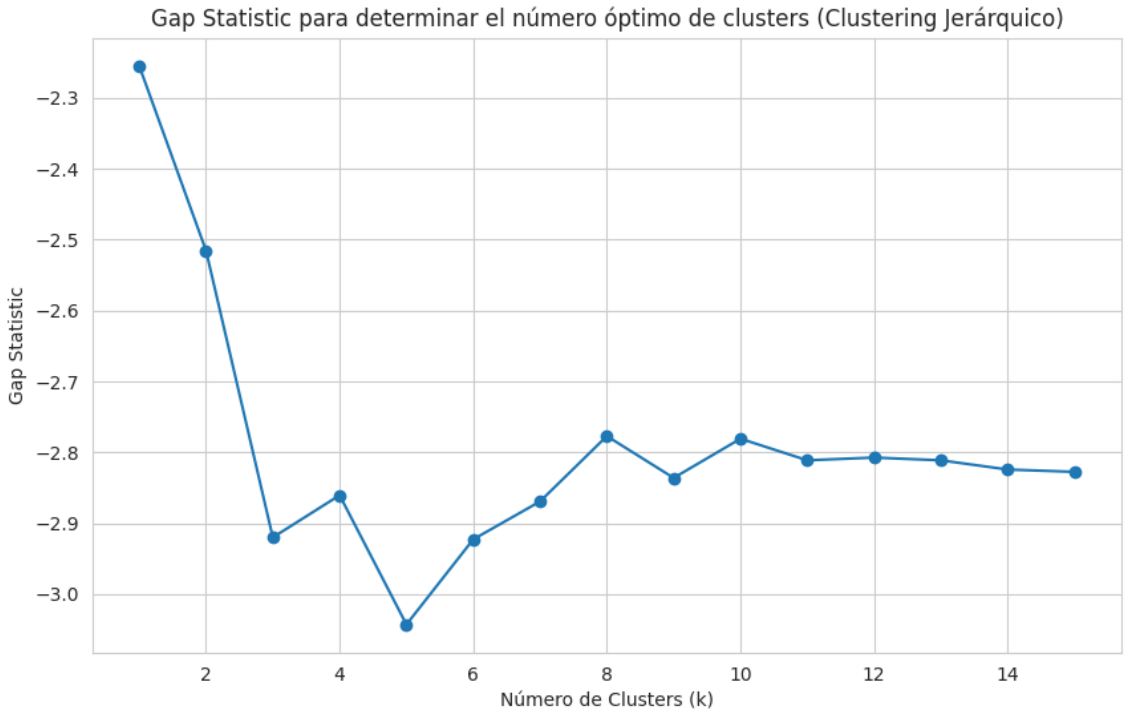
Notamos a simple vista que se guarda una relación entre los clusters de KMeans y los clusters jerárquicos calculados recién.

Silhouette



Utilizando el score de Silhouette, notamos que el score comienza a aumentar a partir del cluster 3 y optar por entre 8 y 10 clusters sería lo correcto.

Gap Statistic



Aplicando GAP, también notamos que 7 clusters sería una cantidad óptima de clusters para utilizar.

CONCLUSIONES

En este trabajo práctico hemos realizado por un lado, técnicas de reducción de dimensionalidad y por otro hemos aplicado algoritmos de agrupación o clustering.

Dentro de las técnicas de reducción de dimensionalidad, aplicamos:

- **Análisis de componentes principales (PCA)** en donde realizamos una extracción de características que utilizamos para analizar y visualizar los datos. El objetivo del PCA es reducir la dimensionalidad de un conjunto de datos creando un nuevo conjunto de variables (componentes principales) que expliquen lo mejor posible la variación de los datos. Definimos seleccionar 4 componentes principales.
- **Isomap** en el cual conseguimos caracterizar las vecindades presentes en la variedad de datos y capturamos la distribución real de los mismos. Notamos que al modificar el valor de n_neighbors cambia la forma en que se construye el grafo de vecinos cercanos que se utiliza para calcular las distancias geodésicas. Por lo tanto, afecta la representación de las dimensiones reducidas que Isomap produce.
- **t-SNE**: utilizamos esta técncia principalmente para la exploración y visualización de datos de alta dimensión. Notamos que en este algoritmo, cambiar los valores de iteraciones y perplejidad entre sus parámetros, puede tener un impacto significativo en los resultados y en el rendimiento del algoritmo.

Dentro de las técnicas de clustering, aplicamos:

- **K-Means** donde se realiza una partición del conjunto de observaciones en k grupos, en el que cada observación pertenece al grupo cuya distancia es menor. Aplicamos el método del codo y GAP statistic para obtener el número óptimo de clusters. Aún así, el modelo no arrojó buenos resultados de agrupamiento en el set de datos, por lo que nos decidimos a realizar previamente el análisis de PCA y luego aplicar K-Means a las componentes principales. Allí obtuvimos una buena categorización de los datos ya que con PCA pudimos eliminar información redundante de los datos y reducir las variables ruidosas que nos perjudicaban.
- **Clustering Jerárquico** donde construimos una jerarquía de clústeres en el conjunto de datos. A partir del dendrograma, y del método del codo obtuvimos un primer indicio de un buen número de clusters a utilizar. Luego aplicando las técnicas de Silhoutte y de GAP pudimos quedarnos con el número óptimos de clusters, que no se diferenció de lo que habíamos fijado con las técnicas anteriores.