Trabajo Práctico
Inferencia Estadística y Reconocimiento de Patrones

Docente: *María Florencia Statti*

Estudiantes: *La Grutta, Dario*
Arias, Valentín Santino

Ciclo Lectivo: *2025, 2do Cuatrimestre*

Índice

1. Método Bootstrap

El método Bootstrap puede ser utilizado para evaluar la incertidumbre de una estimación muestral, y como herramienta para la construcción de intervalos de confianza robustos.

1.1 Explicación del procedimiento

El método bootstrap consiste en realizar determinada cantidad de veces un mismo procedimiento. Dado un estadístico y una muestra de tamaño n, procede a:

1. Hacer un remuestreo (con reposición) de tamaño n, que llamaremos muestra bootstrap
2. Calcular el estadístico en dicha muestra
3. Guardar el valor calculado

El espíritu del método bootstrap es intentar emular el proceso de recolección de nuevos datos de forma artificial, con el objetivo de estimar la variabilidad de un estadístico.

1.2 Selección de una muestra representativa

Para comenzar, vamos a tomar el total de nuestro dataset como la "población", de la cual procederemos a extraer una muestra de tamaño n=70 para aplicar el método.

Para lograrlo, utilizamos el método de selección aleatorio simple, provisto por la librería pandas.

```
# Muestra aleatoria
sample = data_wine.sample(n=70, random_state=42)
```

Además, realizamos algunas verificaciones para corroborar que la muestra cuenta con características similares a la población.

Primero, verificamos que la proporción de cada categoría presente en la columna 'quality' sea similar tanto en la población como en la muestra aleatoria, además de priorizar que contenga al menos una observación de cada clase, para abarcar la mayor proporción de información posible (aunque esto signifique sobrerrepresentar algunas clases por sobre otras).

POBLACIÓN

quality	
5	0.425891
6	0.398999
7	0.124453
4	0.033146
8	0.011257
3	0.006254

MUESTRA

quality	
5	0.485714
6	0.342857
7	0.128571
3	0.014286
8	0.014286
4	0.014286

Luego, verificamos que la media y el desvío de cada variable numérica se asemeje lo más posible desde la muestra a la población. Condición que, como observamos, parecería cumplirse.

```
# Media y desvío de cada variable numérica del dataset original
data_wine.describe().loc[['mean', 'std'], :]
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.636023
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.807569

```
# Media y desvío de cada variable numérica de la muestra
sample.describe().loc[['mean', 'std'], :]
```

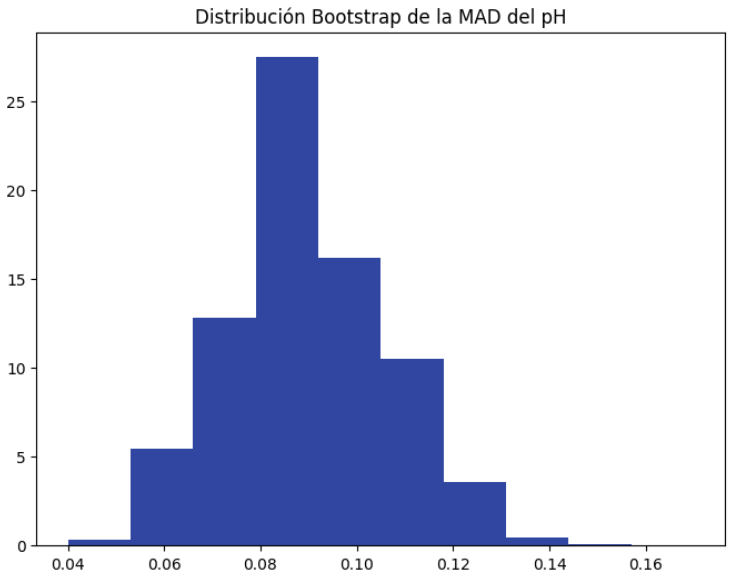
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
mean	8.185714	0.514786	0.263857	2.364286	0.084229	12.700000	42.057143	0.996638	3.317571	0.665571	10.260714	5.600000
std	1.749475	0.161863	0.184189	1.128265	0.044419	7.313577	29.387903	0.001832	0.145964	0.167017	0.967549	0.840979

1.3 Aplicación del método

Para nuestro caso, decidimos realizar el procedimiento con el número de iteraciones B=5000.

```
# Método Bootstrap
B = 5000
n = len(sample)
bs_mad = np.empty(B)

np.random.seed(42)
for i in range(B):
    bs_sample =  sample.sample(n=n, replace=True)
    bs_mad[i] = MAD(bs_sample['pH'])
```

Una vez aplicado, graficamos los resultados con un histograma.



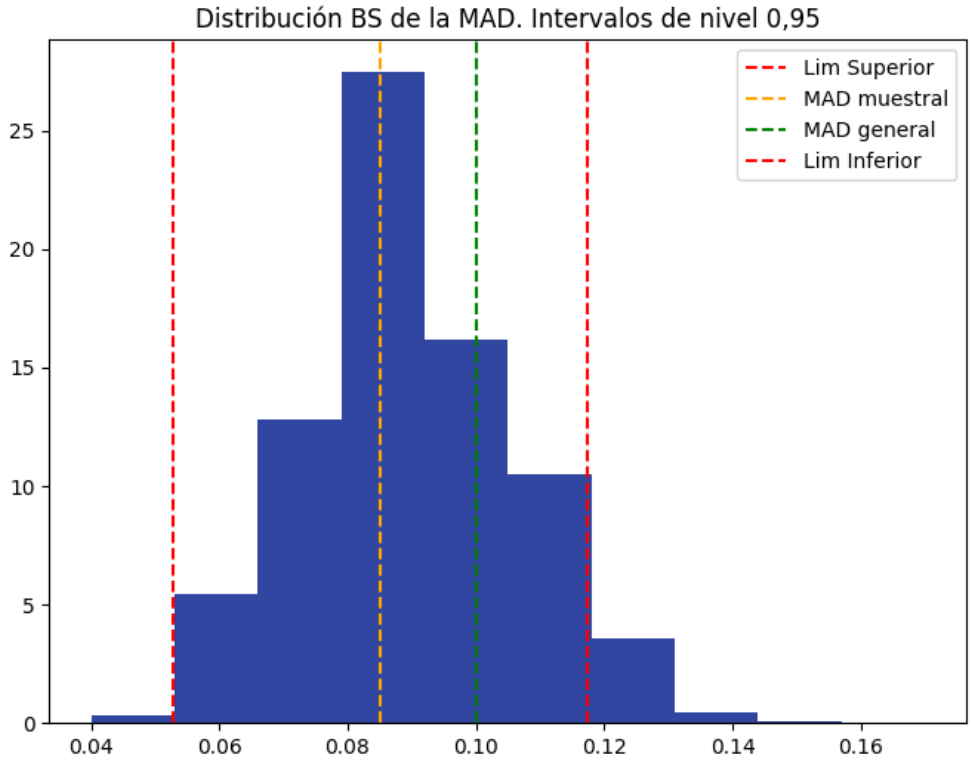
Este histograma sirve como una aproximación o estimación de la función de densidad de probabilidad del estadístico, por lo tanto, debido a que tiene una forma suficientemente acampanada, planteamos la construcción del intervalo de confianza de nivel 0,95 como:

$$MAD_{70} \pm Z_{0,025} \cdot se$$

```
# MAD de los datos de la muestra
mad = MAD(sample['pH'])

# Standard Error
se = np.std(bs_mad, ddof=1)

# Intervalo
bott = mad - 1.96 * se
top = mad + 1.96 * se
```

```
Intervalo bootstrap para la MAD de nivel 0,95:
 [0.0527 : 0.1173]
```

Para una mejor comprensión del intervalo calculado, lo graficamos junto al histograma



Además, decidimos calcular y mostrar dos medidas que sirven para cuantificar la exactitud y la precisión del estimador: el sesgo y el error estándar (ya calculado durante la construcción del intervalo)

```
Error Estándar = 0.01649
Sesgo = -0.01078
```

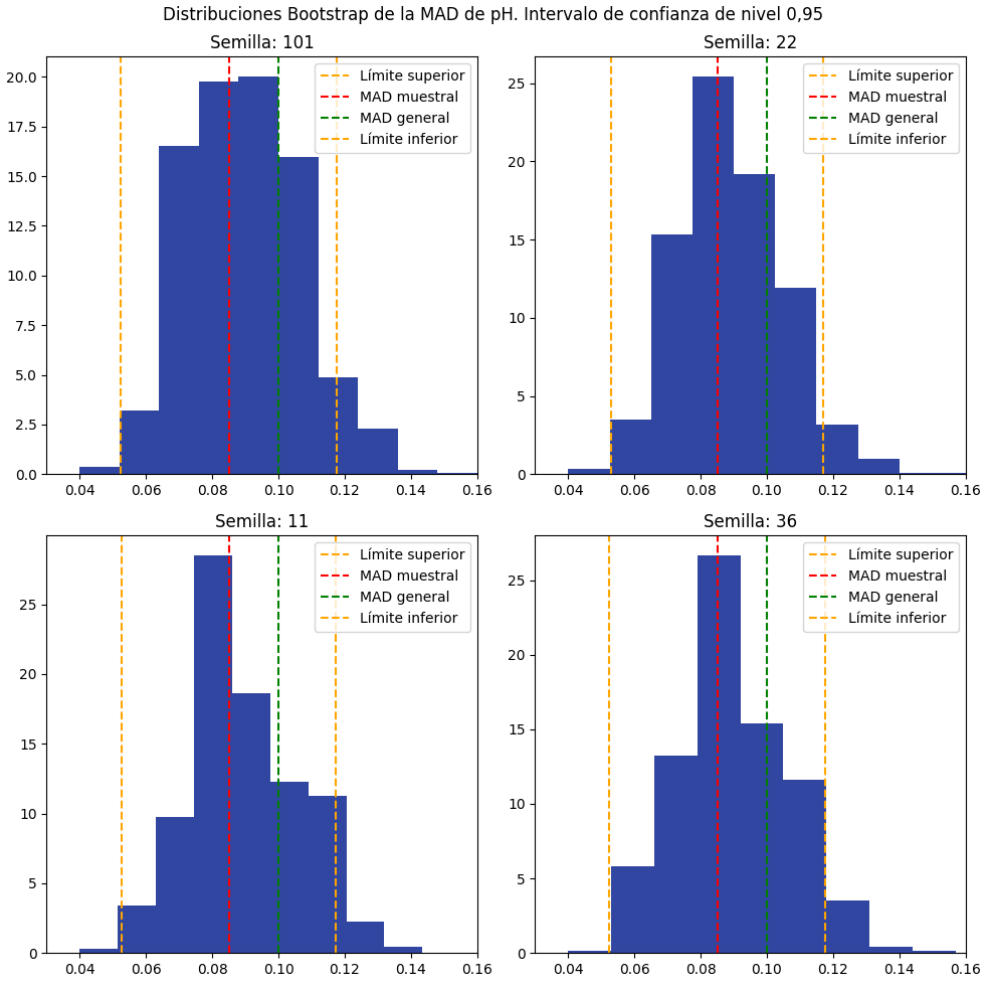
Analizando los resultados, podemos concluir en que la estimación por intervalos del parámetro MAD bajo el enfoque bootstrap, obtuvo buenos resultados, debido a que:

- El intervalo estimado contiene al parámetro de interés
- El error estándar es relativamente pequeño (precisión del estimador alta)
- El sesgo resulta ser incluso menor a una unidad del error estándar (exactitud también alta)

1.4 Repetición del experimento

A razón de verificar la fiabilidad de los resultados arrojados por el experimento realizado (para cubrirnos de algún posible inconveniente relacionado a cuestiones computacionales, como inestabilidad), decidimos hacer cuatro repeticiones del mismo experimento, partiendo de la misma muestra de tamaño n=70.

Para cada repetición, se llevó a cabo el método bootstrap (partiendo de una semilla diferente), luego, se calculó el sesgo y el error estándar, y se construyó el intervalo de confianza de nivel 0,95. Los resultados fueron:



Como es posible observar, no existen diferencias significativas en ninguno de los intervalos de confianza calculados. Simplemente se logra observar pequeñas diferencias en la forma de las campanas dibujadas por los histogramas.

```
Semilla 101
  Error estándar = 0.01661
  Sesgo = -0.01078

Semilla 22
  Error estándar = 0.01632
  Sesgo = -0.01085

Semilla 11
  Error estándar = 0.01652
  Sesgo = -0.01060

Semilla 36
  Error estándar = 0.01672
  Sesgo = -0.01053
```

Por último, analizando las métricas, logramos corroborar que, efectivamente, los resultados obtenidos en la primer instancia, así como las conclusiones propuestas, constan de fiabilidad.

2. Análisis de Componentes Principales

Antes de comenzar, introducimos brevemente el aprendizaje automático no supervisado:

El aprendizaje no supervisado es implementado cuando no se cuenta con una variable respuesta, ya que lo que se busca es poder decir algo del conjunto de datos a analizar. Sin embargo, se debe tener en cuenta que no siempre será posible establecer y hallar relaciones.

Cuando la dimensión de los datos con los que debemos trabajar supera las tres dimensiones, imposibilitando la opción de visualizar el conjunto en su totalidad, aparecen dificultades, problemas de complejidad y (en ocasiones, cuando las dimensiones son muy numerosas) de costo computacional. A partir de esta problemática surge la necesidad de implementar técnicas de reducción de dimensiones, que permitan extraer información relevante del conjunto de datos.

El análisis de componentes principales brinda la posibilidad de satisfacer esa necesidad.

Este modelo se basa en generar combinaciones lineales de las covariables originales (componentes principales), que expliquen la mayor proporción de varianza posible.

Debemos destacar que, en este apartado, la información es varianza. Si bien la varianza alta de un estadístico es mala para los datos, en este caso para saber cuánta información aporta cada variable es necesaria su alta varianza.

El análisis de componentes principales nos permitirá (mediante la reducción de la dimensión de los datos), visualizar información importante y realizar un preprocesamiento de los datos antes de aplicar otras técnicas de aprendizaje automático (usualmente, de aprendizaje supervisado).

2.1 Preprocesamiento de Datos

En primer lugar, para comenzar a trabajar el análisis de las componentes principales, debemos hacer un tratamiento de datos faltantes y corrección de unidades. Para ello, implementamos el siguiente fragmento de código y generamos un renombramiento de las columnas para nuestro conjunto de datos.

```
#Reemplazo de ceros

df = df.fillna(0)

#Pasaje de miligramos a gramos
for i in ['Na (mg)', 'Ca (mg)', 'Fe (mg)']:
    df[i] = df[i] / 1000

#Renombramos las columnas
df.rename(columns = {
    'Na (mg)': 'Na (gr)',
    'Ca (mg)': 'Ca (gr)',
    'Fe (mg)': 'Fe (gr)',
    }, inplace = True)
```

2.2 Análisis y Elección de Componentes Principales

El primer paso antes de comenzar un análisis y elección de componentes es estandarizar las variables. Este requerimiento es sumamente importante ya que, al analizar la varianza de estas, necesitamos mantener una escala equitativa de medición. De otro modo, si mantenemos sus escalas originales, las componentes principales darán mayor peso a las variables con valores grandes en magnitud, aunque no sean las que brindan la mayor información.

La forma en la que logramos adaptar nuestro conjunto de datos es la siguiente:

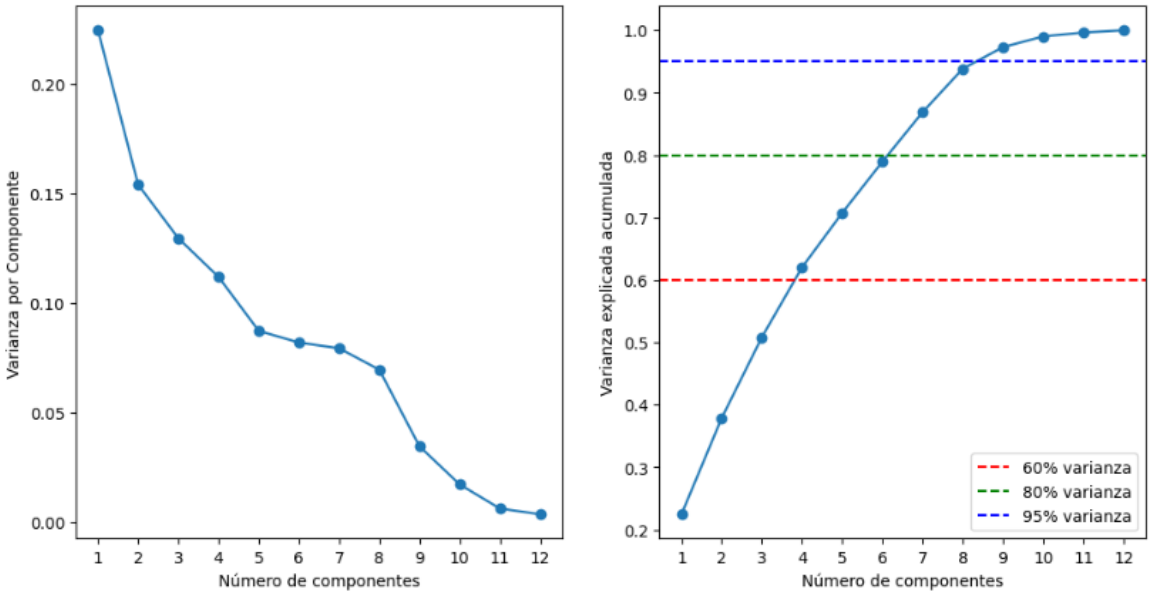
```
ss = StandardScaler()
X = df.drop('Alimento', axis=1)
y = df['Alimento']

X_scaled = ss.fit_transform(X)
dfStandard = pd.DataFrame(X_scaled, columns = X.columns)
```

En segundo lugar, analizamos la acumulación de varianza lograda según la cantidad de componentes contempladas. Para este apartado, buscaremos identificar el margen de varianza que explica cada componente principal seleccionada.

Para dicho análisis, generamos una identificación de varianza individual, por cada componente principal y el acumulado de ellas a medida que aumenta la cantidad de estas. De esa manera, obtenemos el siguiente resultado:

	Varianza	Acumulada
1	0.22456	0.22456
2	0.15404	0.37860
3	0.12959	0.50819
4	0.11193	0.62013
5	0.08731	0.70743
6	0.08200	0.78943
7	0.07936	0.86879
8	0.06960	0.93839
9	0.03459	0.97298
10	0.01721	0.99018
11	0.00622	0.99640
12	0.00360	1.00000

Gracias a la información recolectada anteriormente, procedemos a realizar gráficos representativos de ellos, que refuercen de manera visual, y nos ayuden a tomar una decisión sobre la cantidad de componentes óptima a elegir.



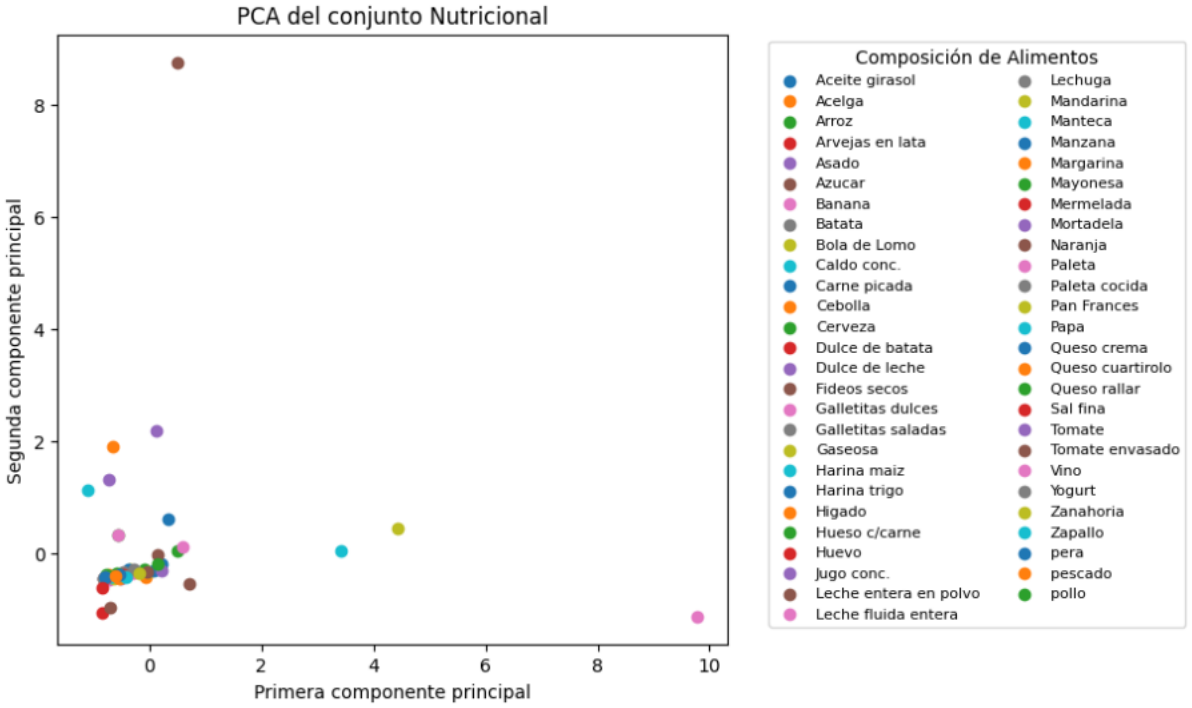
Analizando los gráficos, logramos identificar tres momentos donde se percibe la mayor caída en la cantidad de varianza explicada. Primordialmente, tenemos una gran disminución representada entre los componentes uno y dos. Notamos que la caída de varianza explicada es abrupta con relación a los demás puntos. Sin embargo, no logra ser una cantidad de componentes adecuada, debido a su baja representación acumulada. Luego, entre las componentes cuatro y cinco, aun así, la varianza acumulada en esa cantidad de componentes no logra ser de un valor adecuado a considerar.

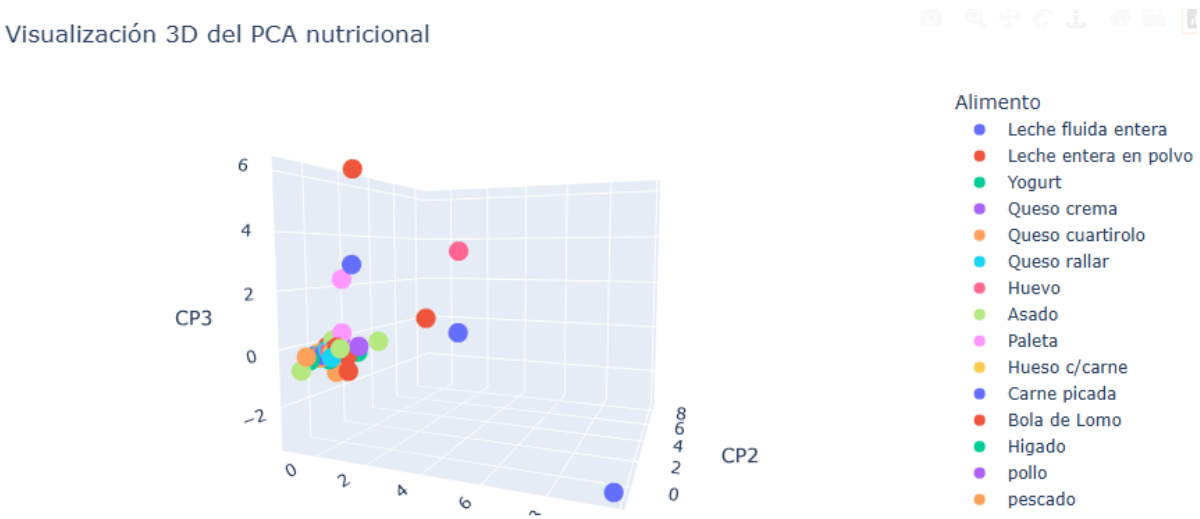
Debido a eso, identificamos que se produce una nueva caída en la cantidad de varianza explicada entre el número de componentes ocho y nueve.

Por lo visto en clase, y como se indica en la bibliografía de la materia ("An Introduction to Statistical Learning: With application in Python"), priorizamos la cantidad de varianza acumulada superior al 80% e inferior al 95% garantizando una explicación de varianza alta evitando el sobreajuste. Por lo tanto, decidimos elegir la cantidad de componentes n_components=8.

Sin embargo, la representación que será posible realizar en dos y tres dimensiones, alcanza una explicación de la varianza relativamente baja para nuestro set de datos, implementando solo dos y tres componentes principales respectivamente.

A continuación, implementamos el modelo de 'Análisis de Componentes Principales' (ACP) que nos ayudará a transformar los datos y generar observaciones.





Visualizando la gráfica realizada anteriormente, observamos que no existen agrupamientos marcados que puedan ser determinados por las primeras tres componentes principales. De acuerdo con la fundamentación realizada anteriormente, el análisis de un gráfico de componentes principales que explique menos del 60% de la varianza carece de fiabilidad. Por lo tanto, su poca interpretabilidad tampoco nos dice nada que deba ser tomado a cuento para nuestro análisis.

3. Clustering

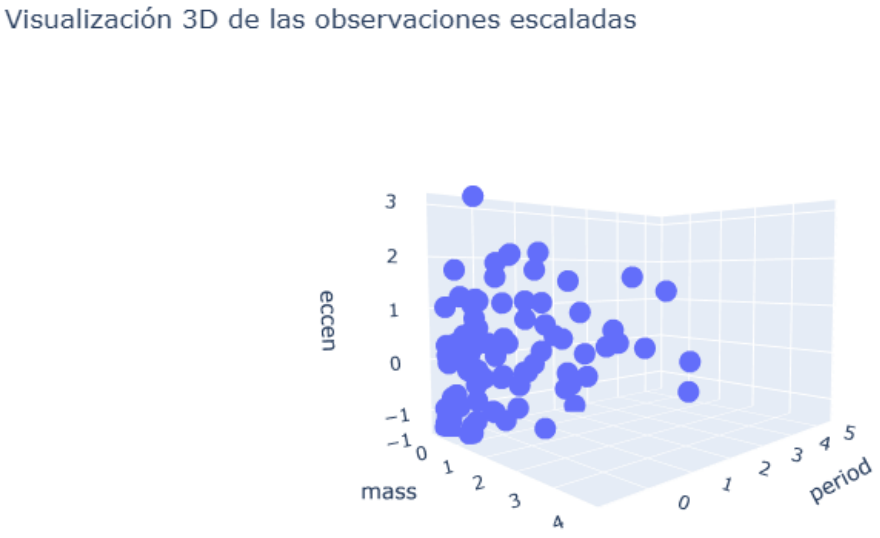
Los métodos de agrupamiento tienen como objetivo identificar posibles segmentaciones dentro de un conjunto de datos. Al lograr una agrupación, las observaciones que integran cada grupo resultan similares entre sí, mientras que se diferencian de las observaciones pertenecientes a otros grupos.

3.1 Preprocesamiento de Datos

Para lograr una óptima representación de las observaciones, es necesaria una estandarización de las escalas, al igual que sucedía en componentes principales antes de comenzar a trabajar.

```
ss = StandardScaler()
df_standard = pd.DataFrame(ss.fit_transform(df), columns=df.columns)
```

Una vez estandarizado nuestro conjunto de datos, graficamos las observaciones con las covariables escaladas.



Ahora sí, podemos proceder a la implementación de los métodos.

3.2 K-Means

K-Means es uno de los métodos disponibles para generar agrupamientos. Este algoritmo de clustering utiliza ‘centroides’ y un número de grupos definido por ‘k’, a través de los cuales asigna cada observación a un único grupo, evitando superposiciones e intersecciones entre ellos

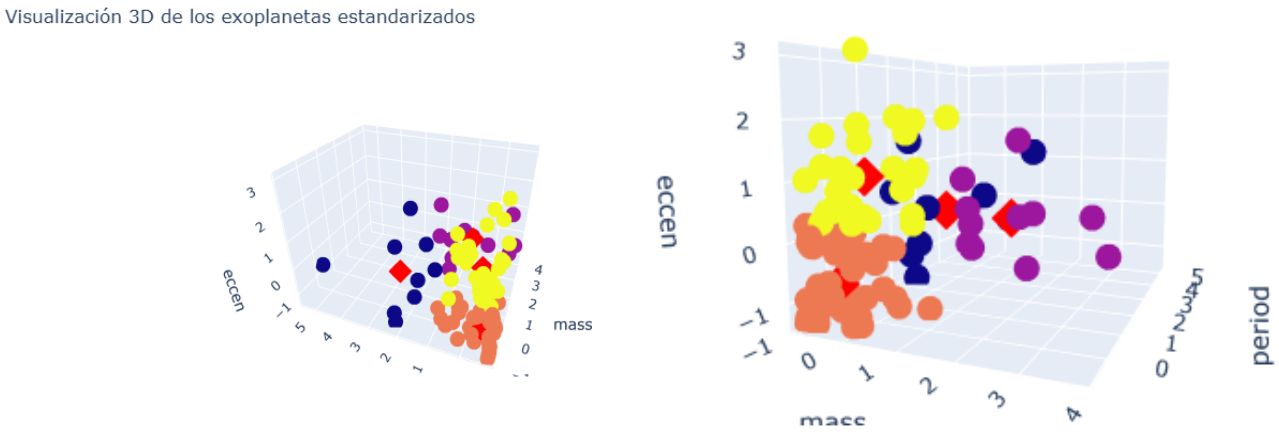
3.2.1 Primera implementación y repetición

Aplicando los datos previamente escalados con un modelo de kmeans, donde se implementa una cantidad de cuatro centroides (k = 4) con una elección aleatoria (“random”), obtendremos un mínimo de la función objetivo igual a 110.85 aproximadamente.

```
kmeans = KMeans(n_clusters=4, init = "random", n_init=10)
kmeans.fit(df_standard)
centroides=kmeans.cluster_centers_
etiquetas=kmeans.labels_
print(f"El valor mínimo de la función objetivo es: {kmeans.inertia_}")
```

```
El valor mínimo de la función objetivo es: 110.85687183618836
```

La visualización de los clusters resultantes es:

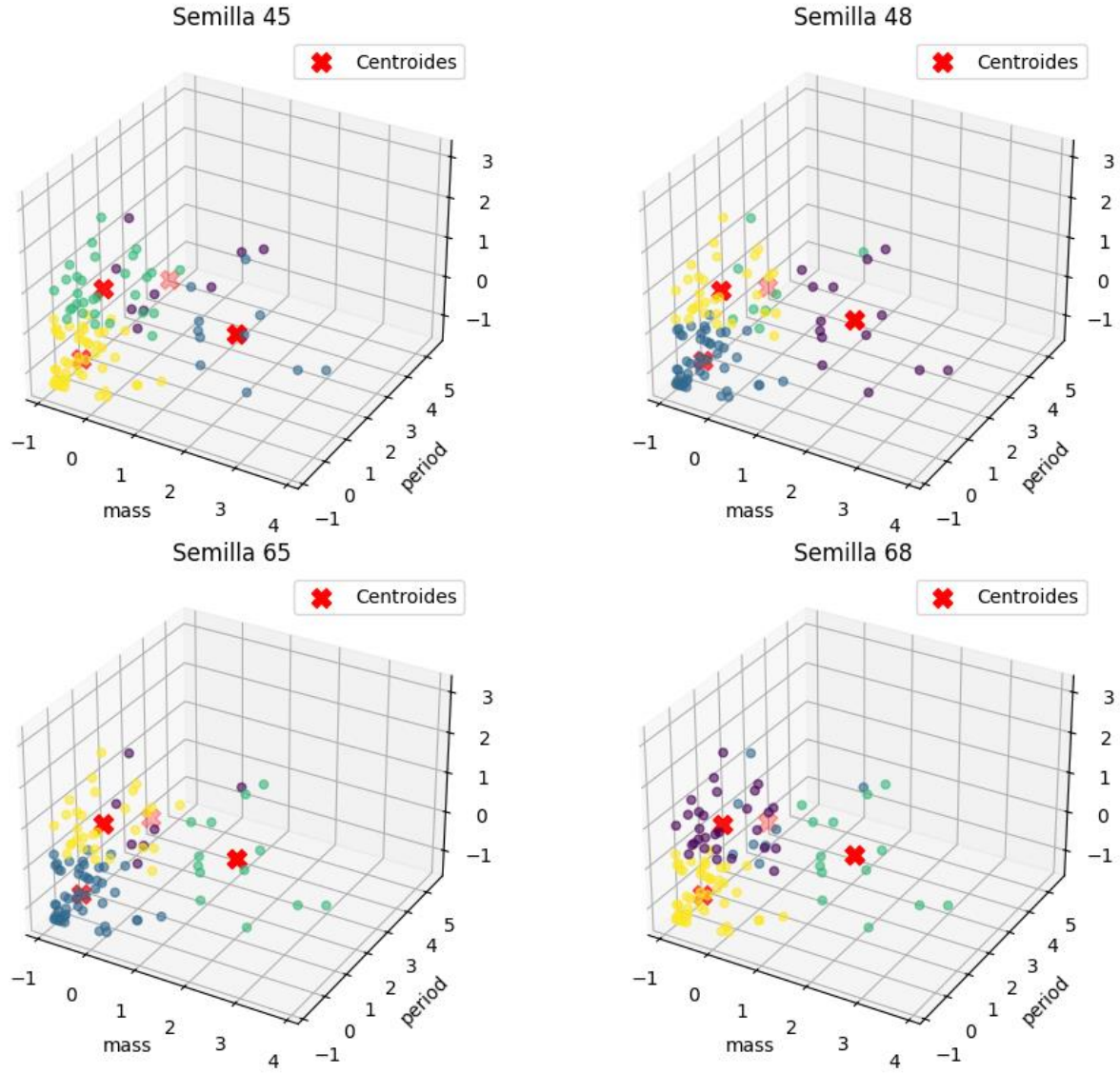


Después de una primera presentación, repetiremos el clúster generado, variando las semillas para lograr ampliar las posibilidades y ver que tan aproximados o distanciados están los modelos generados.

```python
np.random.seed(0)
seeds = list(np.random.randint(0, 100, size = 4))
centroides_rep = []
etiquetas_rep = []
for seed in seeds:
    kmeans = KMeans(n_clusters=4, init = "random", n_init=10, random_state=seed)
    kmeans.fit(df_standard)
    centroides_rep.append(kmeans.cluster_centers_)
    etiquetas_rep.append(kmeans.labels_)
    print(f"\nPara semilla = {seed}, el valor mínimo de la función objetivo es: {kmeans.inertia_:.4f}")
```

```
Para semilla = 44, el valor mínimo de la función objetivo es: 110.8569

Para semilla = 47, el valor mínimo de la función objetivo es: 110.6429

Para semilla = 64, el valor mínimo de la función objetivo es: 110.3288

Para semilla = 67, el valor mínimo de la función objetivo es: 110.6061
```

Habiendo realizado dicho experimento, observamos que los valores hallados para la función objetivo en varios casos fueron similares y/o aproximados. A continuación, también se muestran los grupos conformados en la gráfica correspondiente.



3.2.2 Selección de Centroides

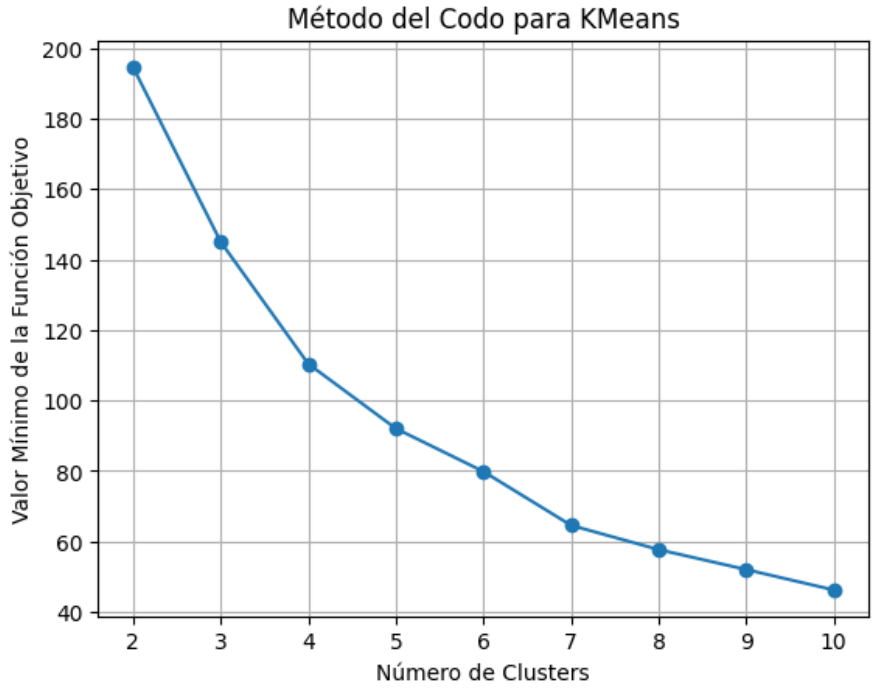
El algoritmo de KMeans, dispone de diferentes opciones para seleccionar los centroides. Lo que realizamos fue analizar el rendimiento para cada implementación, observando el mínimo de la función objetivo resultante.

Dentro de las opciones, contamos con la posibilidad de una elección totalmente aleatoria de los centroides, como se determina mediante init='random'. Por otra parte, también existe la implementación más compleja del algoritmo que es 'k-means++' donde solo el primer centroide es seleccionado al azar y la decisión de los demás dependerá de maximizar las distancias entre ellos. Por último, tenemos la opción de elegir centroides propios, lo cual resulta de utilidad si poseemos conocimiento previo del conjunto de datos. Para nuestro caso, esta última implementación la realizamos con una selección aleatoria del dataset implementando 'sample' y comparando si funciona mejor o peor esa selección en comparación con el método 'random'.

	random	k-means++	propios
2	194.592600	194.592600	194.867700
3	145.014300	145.014300	153.617800
4	110.606100	110.379100	110.999000
5	92.012900	92.012900	93.293100
6	78.357000	79.805000	85.291600
7	64.909600	64.531800	77.701700
8	57.997100	57.641200	68.401200
9	52.406500	51.974000	65.441900
10	48.593600	46.184600	60.900000
Promedio	93.832189	93.570611	101.168222

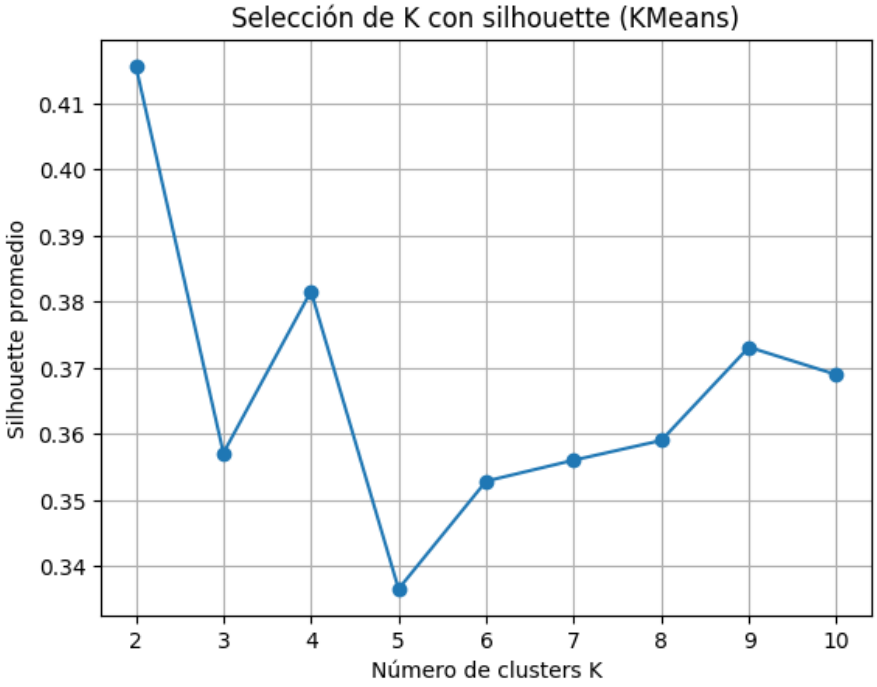
Como resultado final, obtuvimos que el criterio de selección que mejor se desempeñó fue 'k-means++', ya que, el promedio del valor mínimo obtenido (para dicho criterio) de la función objetivo para cada k, presenta el valor más bajo entre los tres algoritmos analizados. Esta es la idea principal detrás de K-Means, lograr un buen agrupamiento el cual logre minimizar la variación intragrupo entre las observaciones que componen a cada grupo en cuestión.

3.2.3 Método del Codo y Silhouette



Al intentar implementar el método del codo, podemos observar que el gráfico denota una caída sostenida del valor mínimo de la función objetivo, como es de esperarse. Sin embargo, no se logra visualizar un quiebre contundente con posterior planicie que permitiera identificar un número de clusters acertado.

Procedemos a implementar también el método Silhouette a continuación, para poder obtener más información y poder tomar una decisión.



Silhouette nos indica una cohesión positiva para un número de cluster igual a 2, 4 y 9.

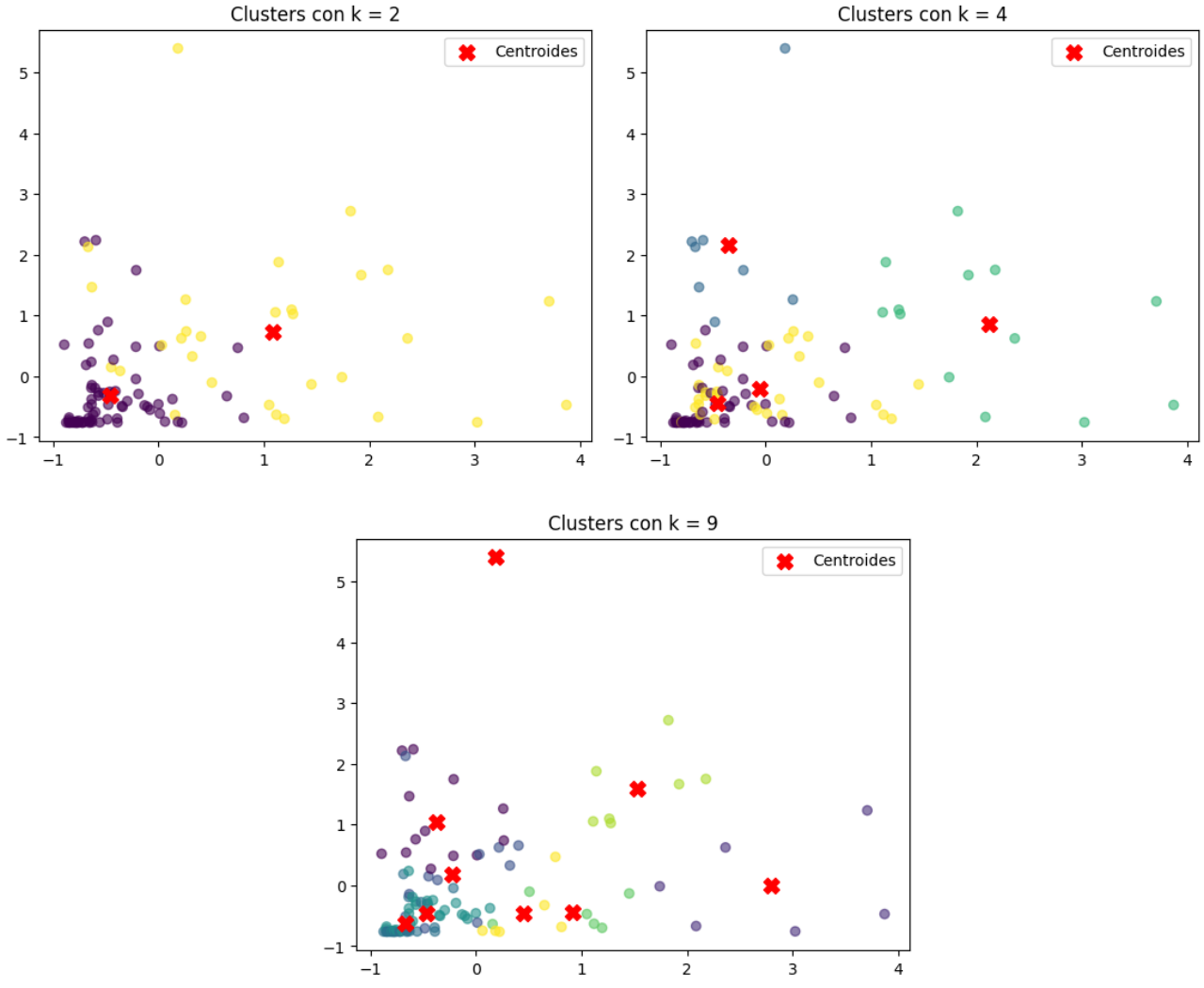
3.2.4 Conclusión

Con k=2 se obtiene una separación clara y compacta sin tener en cuenta posibles subgrupos.

Trabajando con k=4 se mejora la estructura interna de separación y su explicación en la variabilidad interna, además no sobreajustamos demasiado el modelo ni agregamos complejidad.

Por último, El tercer valor más alto conseguido es para k=9. Sin embargo, este valor es descartado por aportar complejidad innecesaria con información dificil de interpretar y comunicar en sus gráficos.

Para respaldar nuestra conclusión, procedemos a graficar las elecciones y descarte frente a la cantidad de Clusters



3.3 K-Medoids

K-Medoids es otro método de clustering que, al igual que K-Means, parte de un número de grupos definido por k. Sin embargo, en lugar de utilizar centroides calculados a partir del promedio de los puntos, selecciona como representante de cada grupo una observación real llamada medoide, que minimiza la distancia total respecto al resto de los elementos del cluster.

Este método resulta una alternativa más robusta a comparación de K-Means (frente a outliers y ruido), ya que toma observaciones reales, en vez de realizar un promedio. Como desventaja, cabe aclarar que puede tener un alto costo computacional, especialmente con datasets grandes.

3.3.1 Método de PAM y asignación de Medoides

Primero, ejecutamos el método K-Medoides con k=4, con el comando *pam* de la librería cluster.

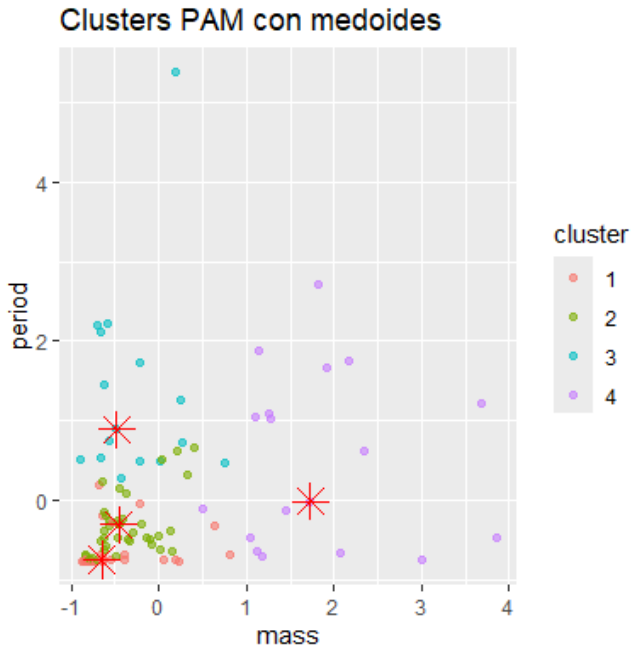
```
#Aplicación de PAM
resultado <- pam(df_std, k=4)
```

Luego, observamos los medoides obtenidos:

	mass	period	eccen
25	-0.6513304	-0.74606166	-1.1924879
48	-0.4557101	-0.30378404	0.3719123
44	-0.4801627	0.89037928	-0.3865847
93	1.7314328	-0.01523463	0.6089427

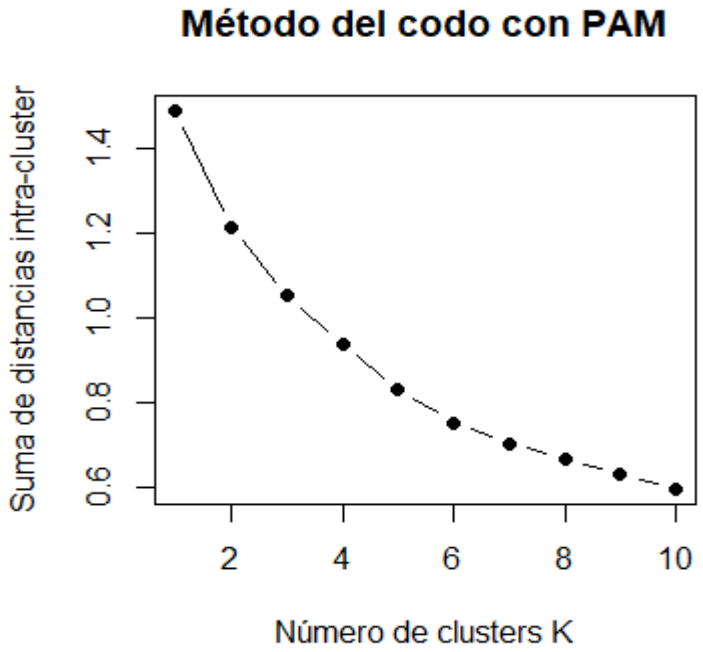
Recordemos que cada uno corresponde a una observación real del conjunto de datos.

Y, por último, graficamos el agrupamiento realizado



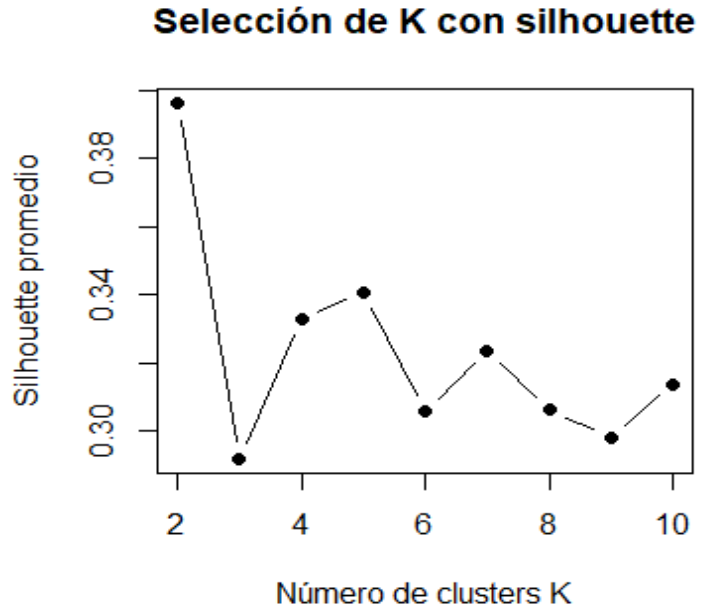
3.3.2 Método del Codo y Silhouette

Con el objetivo de poder decidir la cantidad de clusters óptima, implementamos, primero el método del codo:



Analizando el gráfico, observamos que nos encontramos en la misma situación que con el modelo de K-Means. Este método no nos sirve para tomar una decisión clara y fundamentada sobre la cantidad de clusters óptima para el modelo.

Procedemos a implementar el método Silhouette:



El gráfico indica que k=2 obtiene la mayor puntuación de silhouette, cercano a 0.4, indicando una separación más clara entre grupos y la mayor coherencia interna. A partir de k=3, los valores disminuyen de manera significativa y se mantienen alrededor de 0.30–0.34, sin mejoras sustanciales.

3.3.3 Conclusión

El valor óptimo de K es 2, ya que maximiza la métrica de silhouette y sugiere una estructura natural más clara en los datos. Aumentar el número de clusters no mejora la calidad del agrupamiento y puede introducir subgrupos artificiales sin adicionar un aporte real.

A continuación, el gráfico de la agrupación con el valor óptimo: