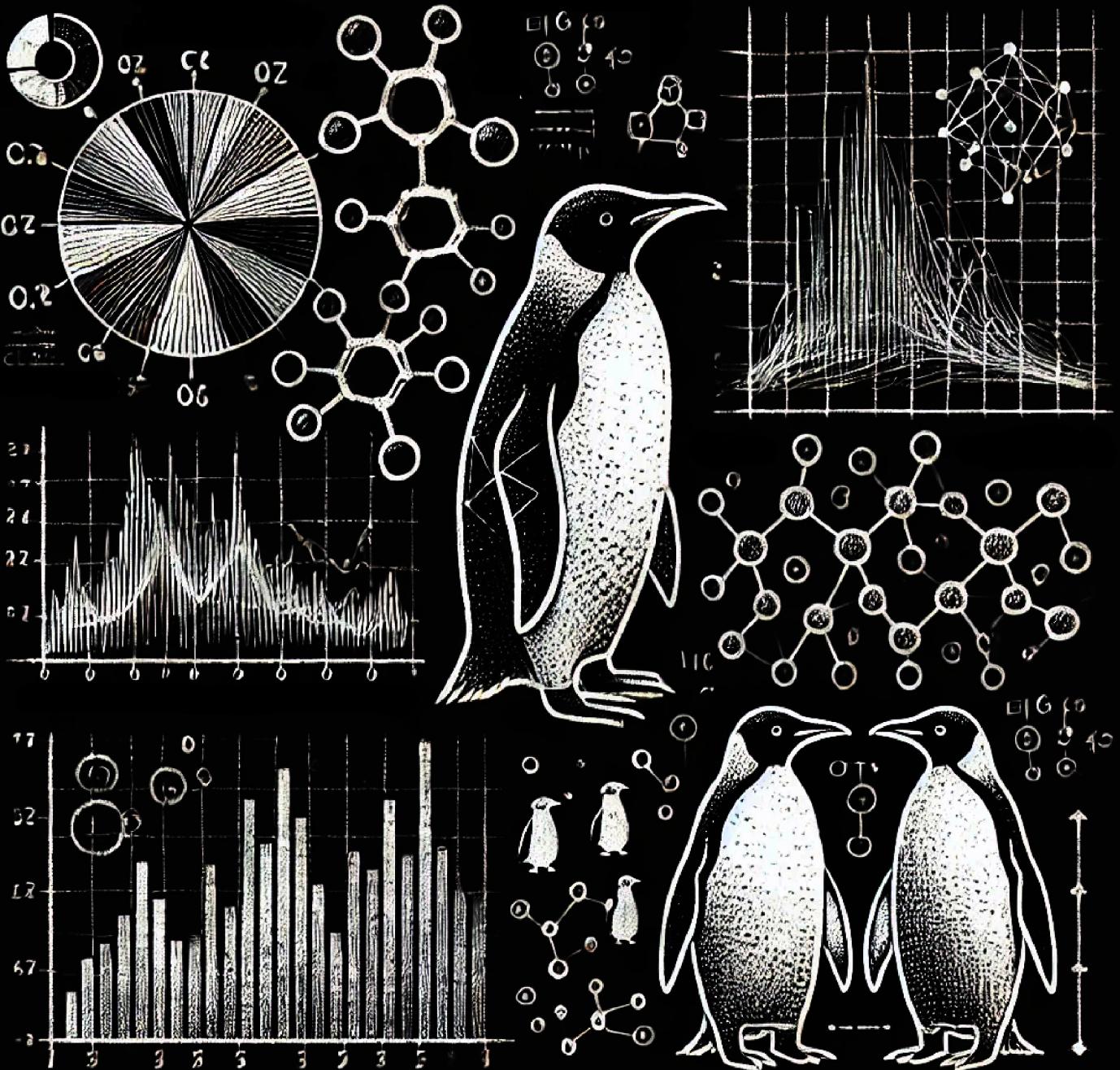


ACP y Clustering

Práctica de el módulo de Minería de Datos y Modelización Predictiva

Laura Rodríguez Ropero



ACP y Clustering

Práctica de el módulo de Minería de Datos y
Modelización Predictiva

por

Laura Rodríguez Ropero

06/03/2025

Índice

1.	Cálculo de la matriz de correlaciones y su representación gráfica	1
2.	Análisis de Componentes Principales (PCA)	3
3.	PCA con el número adecuado de componentes	4
4.	Agrupamiento jerárquico y determinación del número de grupos	7
5.	Agrupamiento K-Means y selección del número óptimo de grupos	12
6.	Validación del agrupamiento	14
7.	Comparación entre agrupamiento jerárquico y K-Means	17
8.	Interpretación de los grupos	18
9.	Conclusiones	19

Profesor: Pablo Arcadio
Facultad: Facultad de Estudios Estadísticos
Universidad: Universidad Complutense de Madrid
Máster: Big Data, Data Science e Inteligencia Artificial



Carga y exploración del dataset

En esta sección se presenta la primera parte del proyecto, dedicada a la carga y exploración del conjunto de datos de *Penguins*. El objetivo principal es familiarizarnos con las características de los datos y obtener una primera idea de su estructura.

0.1. Carga del conjunto de datos

Primero es necesario importar todas las librerías y funciones que se van a usar para realizar la tarea.

```
1 import pandas as pd # Manipulación y análisis de datos tabulares (filas y columnas).
2 import numpy as np # Operaciones numéricas y matriciales.
3 import seaborn as sns # Visualización estadística de datos.
4 import matplotlib.pyplot as plt # Creación de gráficos y visualizaciones.
5 from sklearn.decomposition import PCA # Implementación del PCA
6 from sklearn.preprocessing import StandardScaler # Estandarización de datos
7 from FuncionesMineria2 import (plot_varianza_explicada, plot_cos2_heatmap, plot_corr_cos,
    plot_cos2_bars, plot_contribuciones_proporcionales, plot_pca_scatter,
    plot_pca_scatter_with_vectors, plot_pca_scatter_with_categories)

1 data = sns.load_dataset("penguins").dropna() #Eliminamos los missings al importar
```

0.2. Estadísticas descriptivas básicas

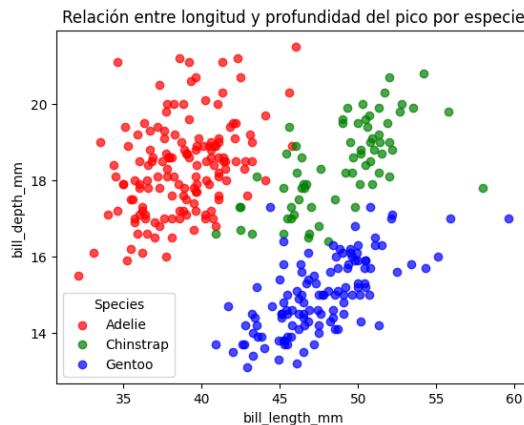
```
1 print(data.describe())
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
count	333.000000	333.000000	333.000000	333.000000
mean	43.992793	17.164865	200.966967	4207.057057
std	5.468668	1.969235	14.015765	805.215802
min	32.100000	13.100000	172.000000	2700.000000
25%	39.500000	15.600000	190.000000	3550.000000
50%	44.500000	17.300000	197.000000	4050.000000
75%	48.600000	18.700000	213.000000	4775.000000
max	59.600000	21.500000	231.000000	6300.000000

0.3. Más visualizaciones

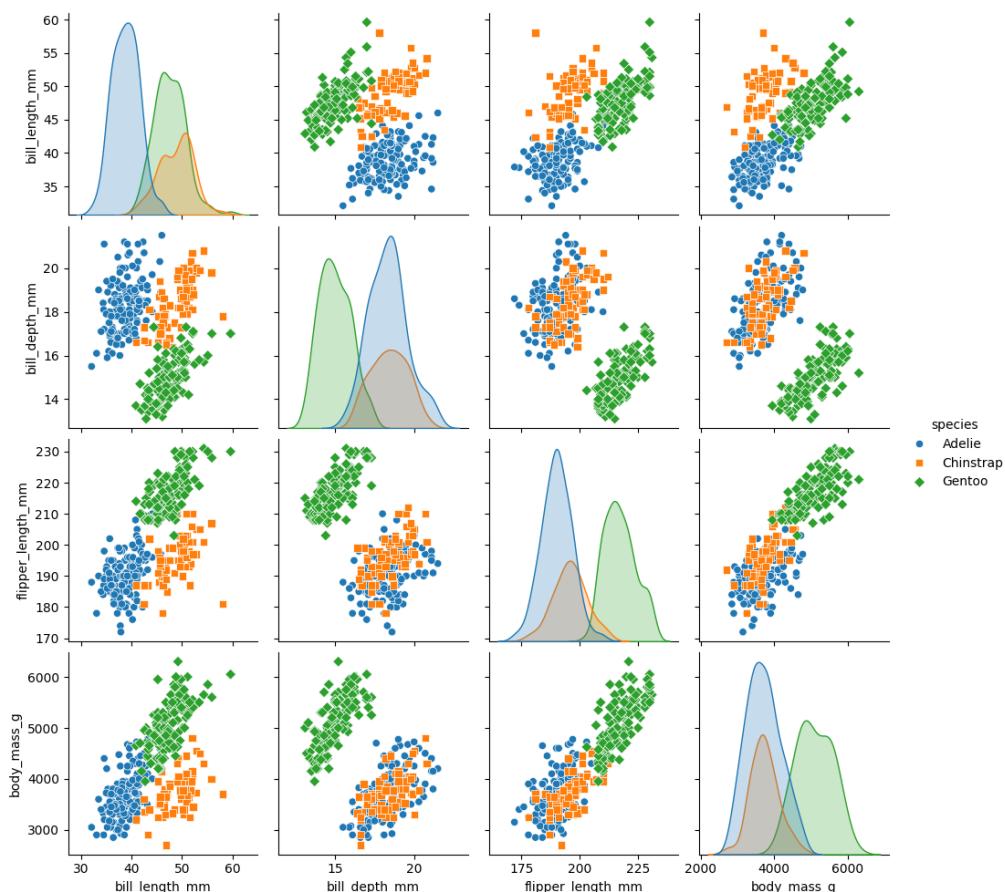
A continuación, se presentan algunas visualizaciones adicionales que permiten explorar de forma más detallada la distribución y la relación entre las variables del conjunto de datos de pingüinos de seaborn.

```
1 unique_species = data['species'].unique()
2 colors = ['red', 'green', 'blue']
3 fig, ax = plt.subplots()
4 for sp, color in zip(unique_species, colors):
5     subset = data[data['species'] == sp]
6     ax.scatter(
7         subset['bill_length_mm'],
8         subset['bill_depth_mm'],
9         c=color,
10        label=sp,
11        alpha=0.7
12    )
13 ax.set_xlabel('bill_length_mm')
14 ax.set_ylabel('bill_depth_mm')
15 ax.set_title('Relación entre longitud y profundidad del pico por especie')
16 ax.legend(title='Species')
17 plt.show()
```



En esta gráfica, se utiliza la librería matplotlib para dibujar un diagrama de dispersión entre bill_length_mm y bill_depth_mm. Se observa que existe cierta superposición entre las nubes de puntos, pero también hay zonas en las que las especies se distinguen con mayor claridad, lo que podría ser relevante para tareas de clasificación o análisis posterior.

```
1 sns.pairplot(data, hue="species", diag_kind="kde", markers=["o", "s", "D"])
2 plt.show()
```



Por otro lado, esta otra gráfica muestra diagramas de dispersión para todas las combinaciones de variables cuantitativas, así como distribuciones univariantes (histogramas o densidades en la diagonal). Con ello, es posible identificar patrones, correlaciones y posibles agrupaciones de los datos. Estas visualizaciones proporcionan un panorama más completo sobre cómo se relacionan la longitud y profundidad del pico, la longitud de las aletas, y el peso corporal de los pingüinos, variables que serán de utilidad para los análisis posteriores. En particular, la diferenciación visual entre especies servirá para orientar técnicas de clasificación o reducción de dimensionalidad, así como para verificar hipótesis acerca de la variabilidad morfológica en las distintas especies de pingüinos.

1

Cálculo de la matriz de correlaciones y su representación gráfica

En primer lugar, se filtran las columnas para conservar únicamente aquellas con datos de tipo entero o flotante, ya que el enfoque se centra en el análisis de las variables cuantitativas. A continuación, se eliminan del conjunto de datos las columnas que contienen información categórica para simplificar el análisis.

Con el subconjunto de datos numéricos, se calculan varios estadísticos descriptivos: mínimos, máximos, percentiles (25, 50 y 75), media, desviación estándar, varianza y el coeficiente de variación de cada variable.

```
1 numericas = data.select_dtypes(include=['int', 'int32', 'int64','float', 'float32', 'float64']) .columns
2 categoricas = [variable for variable in list(data) if variable not in numericas]
3 data_num = data.drop(columns= categoricas)
4 variables = list(data_num.columns)
5
6 estadisticos = pd.DataFrame({
7     'Mínimo': data_num[variables].min(),
8     'Percentil 25': data_num[variables].quantile(0.25),
9     'Mediana': data_num[variables].median(),
10    'Percentil 75': data_num[variables].quantile(0.75),
11    'Media': data_num[variables].mean(),
12    'Máximo': data_num[variables].max(),
13    'Desviación Estándar': data_num[variables].std(),
14    'Varianza': data_num[variables].var(),
15    'Coeficiente de Variación': (data_num[variables].std() / data_num[variables].mean()),
16    'Datos Perdidos': data_num[variables].isna().sum()
17 })
18 display(estadisticos)
```

	Mínimo	Percentil 25	Mediana	Percentil 75	Media
bill_length_mm	32.1	39.5	44.5	48.6	43.992793
bill_depth_mm	13.1	15.6	17.3	18.7	17.164865
flipper_length_mm	172.0	190.0	197.0	213.0	200.966967
body_mass_g	2700.0	3550.0	4050.0	4775.0	4207.057057

	Máximo	Desv. Est.	Varianza	Coef. Var.	Datos Perdidos
bill_length_mm	59.6	5.468668	29.906333	0.124308	0
bill_depth_mm	21.5	1.969235	3.877888	0.114725	0
flipper_length_mm	231.0	14.015765	196.441677	0.069742	0
body_mass_g	6300.0	805.215802	648372.487699	0.191396	0

A continuación, se lleva a cabo el proceso de estandarización de los datos y el posterior cálculo y visualización de la matriz de correlaciones.

```

1 data_std = pd.DataFrame(
2     StandardScaler().fit_transform(data_num),
3     columns=['{}_z'.format(variable) for variable in variables],
4     index=data_num.index
5 )
6 data_std.head()

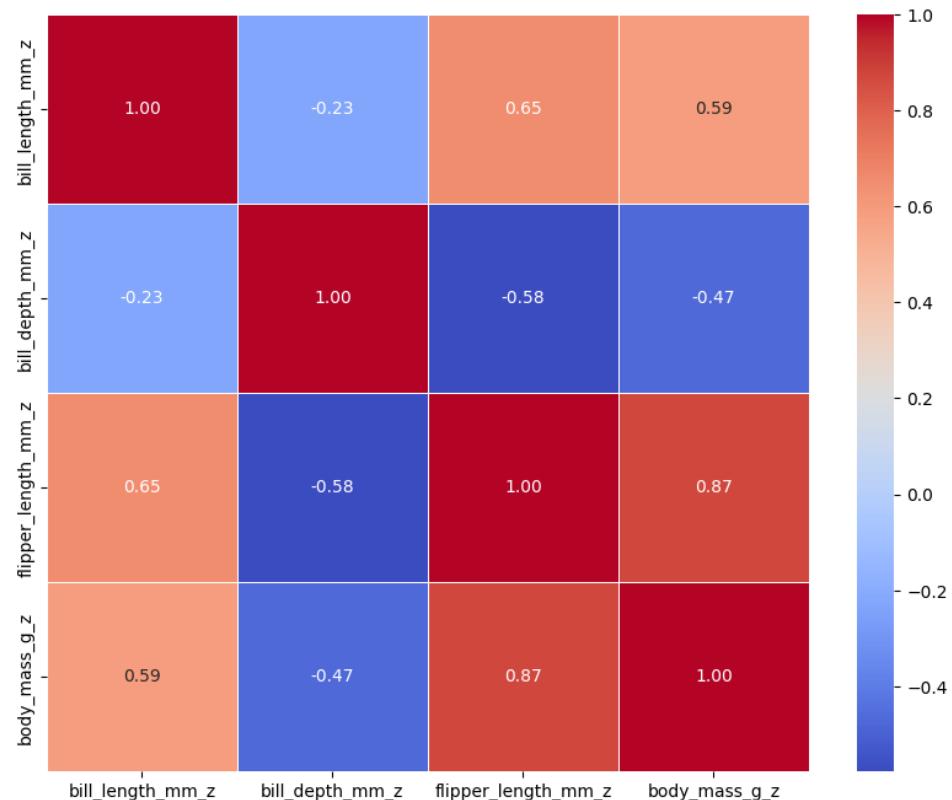
```

Índice	bill_length_mm_z	bill_depth_mm_z	flipper_length_mm_z	body_mass_g_z
0	-0.896042	0.780732	-1.426752	-0.568475
1	-0.822788	0.119584	-1.069474	-0.506286
2	-0.676280	0.424729	-0.426373	-1.190361
4	-1.335566	1.085877	-0.569284	-0.941606
5	-0.859415	1.747026	-0.783651	-0.692852

```

1 R = data_std.corr()
2 plt.figure(figsize=(10, 8))
3 sns.heatmap(R, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
4 plt.show()

```



Observando la matriz:

bill_length_mm_z presenta una correlación moderada a alta con flipper_length_mm_z (0.65) y con body_mass_g_z (0.59). Esto sugiere que, en pingüinos, a mayor longitud del pico tienden a tener mayor longitud de aletas y mayor masa corporal. bill_depth_mm_z tiene correlaciones negativas con flipper_length_mm_z (-0.58) y con body_mass_g_z (-0.47), lo que indica que un pico más profundo se asocia, en general, a aletas más cortas y a un peso corporal menor, al menos dentro de este conjunto de datos. flipper_length_mm_z y bill_depth_mm_z muestran la correlación inversa más alta (-0.58), mientras que flipper_length_mm_z y body_mass_g_z muestran la correlación directa más alta (0.87), lo que implica que los pingüinos con aletas más largas suelen ser también los de mayor masa corporal.

2

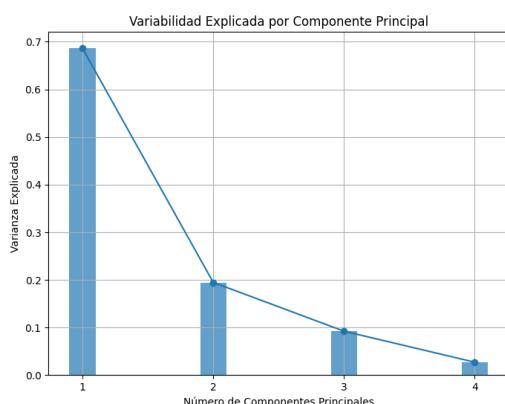
Análisis de Componentes Principales (PCA)

En esta sección se realiza el Análisis de Componentes Principales (PCA) con un máximo de cuatro componentes, tomando como base la matriz de correlaciones de las variables estandarizadas. A partir de ello, se obtienen los autovalores (indicativos de la varianza asociada a cada componente), la proporción de varianza explicada y la varianza acumulada.

```
1 pca = PCA(n_components=4)
2 fit = pca.fit(data_std)
3 autovalores = fit.explained_variance_
4 var_explicada = fit.explained_variance_ratio_
5 var_acumulada = np.cumsum(var_explicada)
6 datos = {'Autovalores': autovalores, 'Variabilidad_Explicada': var_explicada, 'Variabilidad_Acumulada': var_acumulada}
7 tabla = pd.DataFrame(datos, index=['Componente_{}'.format(i) for i in range(1, fit.n_components_+1)])
8 print(tabla)
```

Componente	Autovalores	Var. Explicada	Var. Acumulada
1	2.753625	0.686339	0.686339
2	0.780461	0.194529	0.880868
3	0.369753	0.092161	0.973029
4	0.108210	0.026971	1.000000

```
1 resultados_pca = pd.DataFrame(fit.transform(data_std), columns=['Componente_{}'.format(i) for i in range(1, fit.n_components_+1)], index=data_std.index)
2 plot_varianza_explicada(var_explicada, fit.n_components_)
```



Atendiendo a la varianza explicada y a la varianza acumulada, se observa que las dos primeras componentes recogen cerca del 88% de la variabilidad. A menudo se elige el número de componentes que cubra al menos entre un 80% y un 90% de la varianza para un buen equilibrio entre simplicidad y fidelidad. Por ello, dos componentes podrían considerarse suficientes para capturar la mayor parte de la información en los datos de pingüinos, si bien se podría llegar a un 97% con tres componentes si se desea un nivel más alto de detalle.

3

PCA con el número adecuado de componentes

En esta sección se ha llevado a cabo un PCA con solo dos componentes principales (CP1 y CP2) para simplificar la interpretación y, al mismo tiempo, capturar la mayor parte de la variabilidad de los datos de pingüinos. A grandes rasgos, se ha hecho lo siguiente:

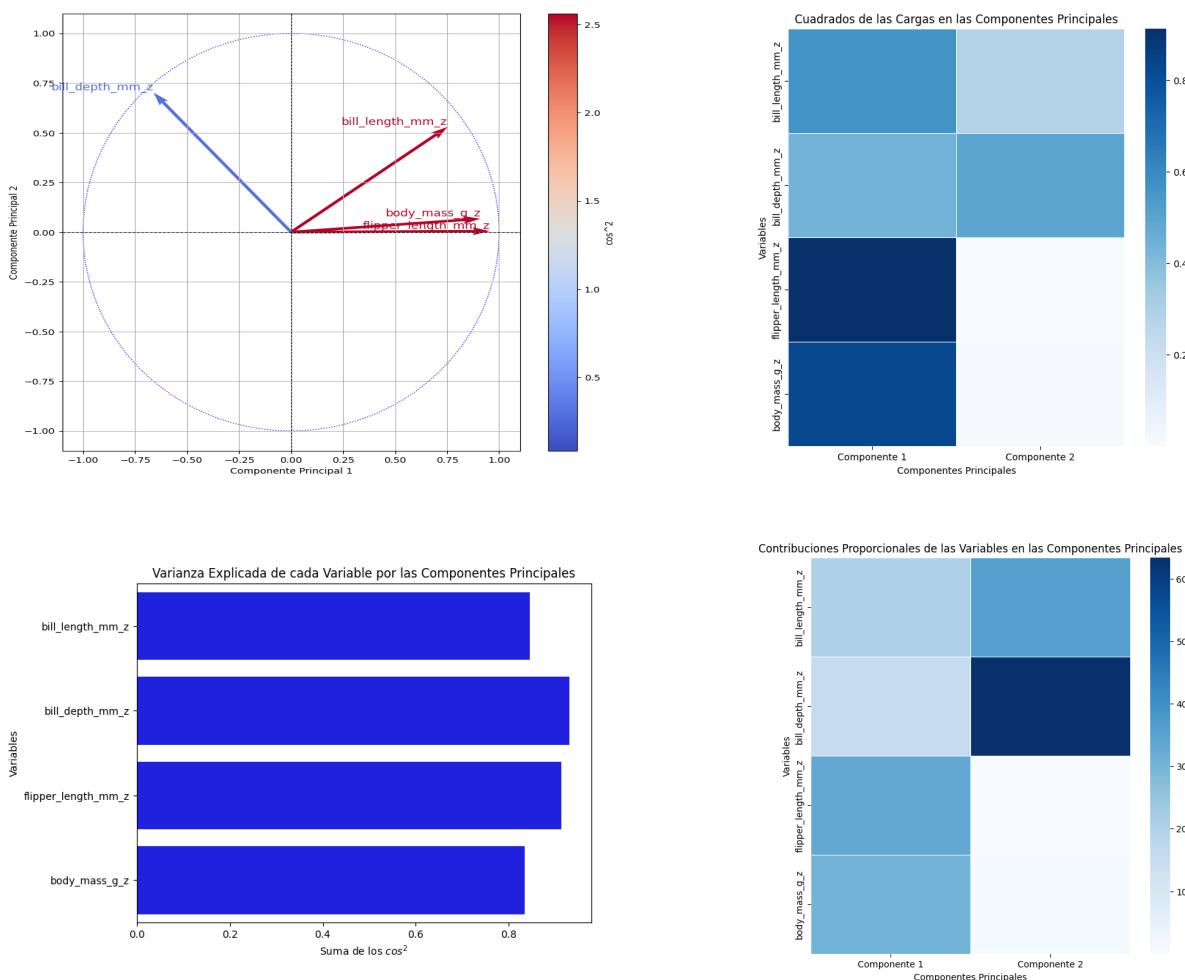
Se crea un objeto PCA (`n_components=2`) que se ajusta (`fit`) a los datos estandarizados. Se obtienen los autovectores, que indican la contribución de cada variable original a cada componente, y los autovalores, que reflejan la varianza explicada por cada uno de esos componentes. Mediante `transform`, se proyectan las observaciones en el nuevo espacio (CP1, CP2), generando un DataFrame donde cada fila conserva el índice de la observación original pero se describe solo por sus nuevas coordenadas (las dos componentes principales).

```
1 pca = PCA(n_components=2)
2 fit = pca.fit(data_std)
3 autovalores = fit.explained_variance_
4 autovectores = pd.DataFrame(pca.components_.T, columns = ['A\u00f3utovector_{}'.format(i) for i in
    range(1, fit.n_components_+1)], index = ['{}_z'.format(variable) for variable in
    variables])
5 resultados_pca = pd.DataFrame(fit.transform(data_std), columns=['Componente_{}'.format(i) for
    i in range(1, fit.n_components_+1)], index=datos_estandarizados.index)
```

Se calcula la correlación entre las variables originales y los componentes principales, con el fin de entender qué tan fuertemente se relaciona cada variable con cada componente. Se dibujan gráficos donde se representan las variables originales en el plano (CP1, CP2). Se genera la matriz de `cos2`, que es simplemente el cuadrado de estas correlaciones, y se visualiza en forma de mapa de calor, para identificar qué variables tienen mayor aportación o “contribución” a cada componente.

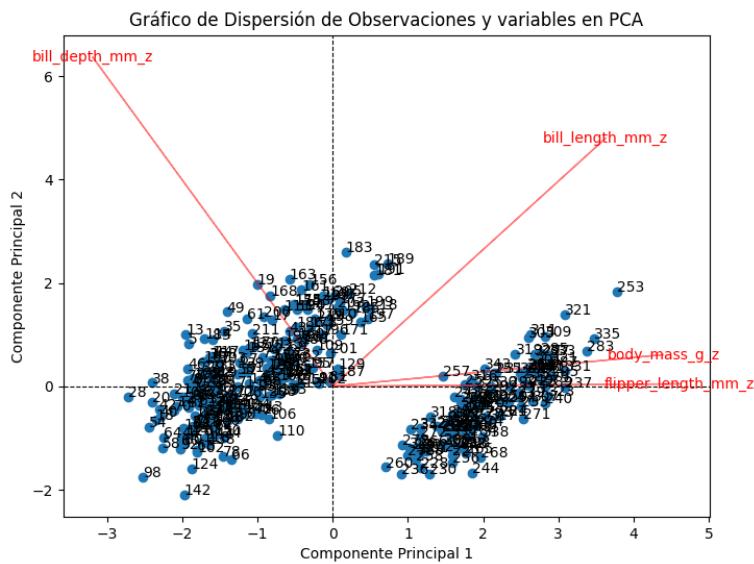
```
1 datos_z_cp = pd.concat([data_std, resultados_pca], axis=1)
2 variables_cp = datos_z_cp.columns
3 correlacion = pd.DataFrame(np.corrcoef(data_std.T, resultados_pca.T),
    index = variables_cp, columns = variables_cp)
4 n_variables = fit.n_features_in_
5 correlaciones_data_con_cp = correlacion.iloc[:fit.n_features_in_, fit.n_features_in_:]
6 plot_corr_cos(fit.n_components, correlaciones_data_con_cp)
7 cos2 = correlaciones_data_con_cp **2
8 plot_cos2_heatmap(cos2)

1 plot_cos2_bars(cos2)
2 contribuciones_proporcionales = plot_contribuciones_proporcionales(cos2,autovalores,fit.
    n_components)
```



Adicionalmente, se muestra un diagrama de dispersión con vectores que indican la dirección y magnitud de cada variable original dentro del espacio de componentes, así como las observaciones (pingüinos) en esos mismos ejes.

```
1 plot_pca_scatter(pca, data_std, fit.n_components)
2 plot_pca_scatter_with_vectors(pca, data_std, fit.n_components, fit.components_)
```



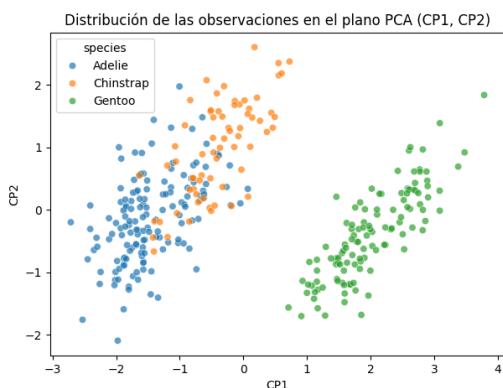
- a) Representación de cada componente en términos de las características físicas de los pingüinos.
- **Primera componente (PC1):** Representa principalmente la variabilidad en el tamaño del pingüino, pues tiene cargas altas en la longitud de la aleta y el peso corporal.
 - **Segunda componente (PC2):** Distingue principalmente la forma del pico, con una alta carga en la profundidad del pico y longitud del pico en sentidos opuestos.

- b) Distribución de las observaciones por especie en el plano PCA (CP1, CP2)

```

1 data['CP1'] = resultados_pca['Componente_U1']
2 data['CP2'] = resultados_pca['Componente_U2']
3 plt.figure(figsize=(7,5))
4 sns.scatterplot(data=data, x='CP1', y='CP2', hue='species', alpha=0.7)
5 plt.show()

```



En el eje de la Componente 1 (CP1), la especie Gentoo aparece con valores más altos, lo que sugiere que presenta mayores rasgos físicos (por ejemplo, longitud de aleta o masa corporal) asociados positivamente a este componente, mientras que Adelie se ubica en la zona negativa, indicando valores menores en esas mismas variables. Chinstrap queda en una posición intermedia en CP1. En cuanto a la Componente 2 (CP2), se aprecia que Chinstrap tiende a alcanzar valores más elevados que Adelie y Gentoo, lo cual refleja una variación adicional (quizá relacionada con la forma del pico) que distingue más a esta especie del resto en el segundo eje.

- c) El índice físico es una forma de resumir en un solo número distintas características físicas de los pingüinos. Para construirlo, se realiza una combinación lineal de las variables estandarizadas con las cargas del componente principal que se quiera utilizar. A grandes rasgos, si contamos con cuatro variables estandarizadas $\{z_1, z_2, z_3, z_4\}$ y los correspondientes pesos o cargas del primer componente principal $\{w_1, w_2, w_3, w_4\}$, el índice de cada pingüino se calcula de la siguiente forma:

$$\text{Índice_Físico} = w_1 \cdot z_1 + w_2 \cdot z_2 + w_3 \cdot z_3 + w_4 \cdot z_4.$$

Este valor resumirá, en un único indicador, la mayor parte de la variabilidad que expresan las variables físicas, de acuerdo con el criterio marcado por el componente principal que estemos utilizando (en este caso el primero, por explicar la mayor proporción de varianza).

```

1 primer_componente = pca.components_[0]
2 df['Indice_Físico'] = datos_estandarizados.drop(columns=['CP1', 'CP2']).values.dot(
    primer_componente)
3 indice_por_especie = df.groupby('species')['Indice_Físico'].mean()
4 print(indice_por_especie)

```

Valor medio del índice físico por especie: Adelie -1.459721, Chinstrap -0.388600, Gentoo 2.012976

Podemos extraer las siguientes conclusiones:

- **Gentoo** presenta un valor medio positivo y alto (+2.012976), lo que sugiere que, en las variables físicas que más inciden positivamente en el primer componente (por ejemplo, masa corporal o longitud de aleta), esta especie suele tener valores altos en relación con las otras.
- **Adelie** tiene el valor medio más bajo (-1.459721), indicando que esta especie muestra valores más reducidos en esas mismas características físicas principales, en comparación con la media del conjunto.
- **Chinstrap** aparece con un valor intermedio (-0.388600), lo que la ubica entre los otros dos grupos en cuanto al conjunto de características físicas consideradas.

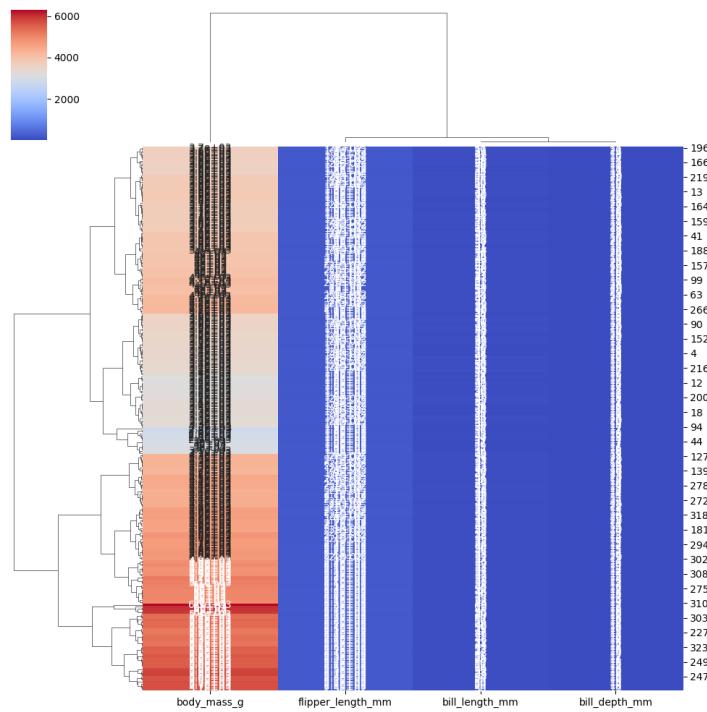
4

Agrupamiento jerárquico y determinación del número de grupos

Como primer paso, se importan los módulos necesarios.

```
1 from scipy.spatial import distance
2 import scipy.cluster.hierarchy as sch
3 from sklearn.cluster import KMeans, DBSCAN
4 from sklearn.metrics import silhouette_samples, silhouette_score
```

Ahora, se pretente hacer un primer agrupamiento jeráquico, nos ayudamos de la función `clustermap` de Seaborn para generar un mapa de calor junto con un dendograma.



Si nos fijamos en el dendrograma de la izquierda, muestra cómo las observaciones (cada pingüino) se van uniendo progresivamente en función de su similitud en las variables analizadas. Visiblemente, hay varios puntos en los que se produce un salto mayor de la distancia, lo que sugiere la presencia de 3 grandes grupos (o al menos dos) cuando hacemos un “corte” adecuado del dendrograma.

Se utiliza la función `cdist` para calcular, de forma pareada, la distancia euclídea entre todas las observaciones del conjunto de datos.

```
1 distance_matrix = distance.cdist(data_num, data_num, 'euclidean')
2 distance_small = distance_matrix[:5, :5]
3 distance_small = pd.DataFrame(distance_small, index=data_num.index[:5], columns=data_num.
        index[:5])
4 distance_small_rounded = distance_small.round(2)
5 print(distance_small_rounded)
```

	0	1	2	4	5
0	0.00	50.27	500.20	300.25	100.42
1	50.27	0.00	550.07	350.09	150.09
2	500.20	550.07	0.00	200.05	400.04
4	300.25	350.09	200.05	0.00	200.04
5	100.42	150.09	400.04	200.04	0.00

Se convierte la matriz completa en un DataFrame, y se genera un mapa de calor, en el que cada celda se colorea en función de la distancia entre cada par de observaciones.

```
1 plt.figure(figsize=(8, 6))
2 data_distance = pd.DataFrame(distance_matrix, index = data_num.index, columns = data_num.
        index)
3 sns.heatmap(data_distance, annot=False, cmap="YlGnBu", fmt=".1f")
4 plt.show()
```

Luego se ejecuta el agrupamiento jerárquico, se extrae la matriz de enlace (linkage), se determina un orden basado en ese dendrograma y, por último, se vuelve a representar el conjunto de datos con las filas y columnas reordenadas según la estructura de clusters identificada.

```
1 linkage = sns.clustermap(data_distance, cmap="YlGnBu", fmt=".1f", annot=False, method='
        average').dendrogram_row.linkage
2 order = pd.DataFrame(linkage, columns=['cluster_1', 'cluster_2', 'distance', 'new_count']).
        index
3 reordered_data = data_num.reindex(index=order, columns=order)
4 sns.heatmap(reordered_data, cmap="YlGnBu", fmt=".1f", cbar=False)
5 plt.show()
```

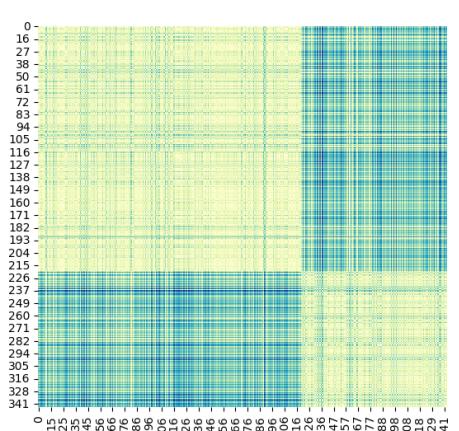


Figure 4.1: Matriz de distancias euclídeas

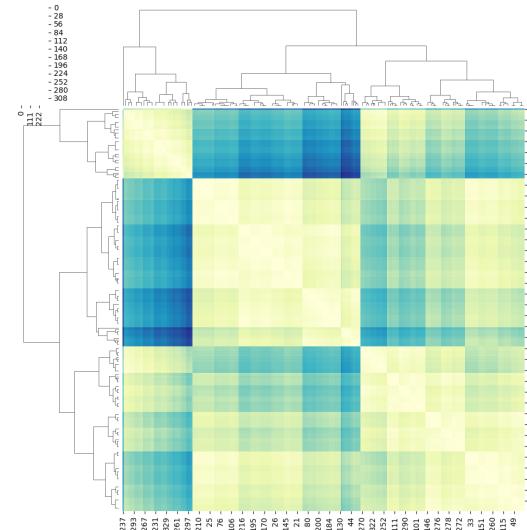


Figure 4.2: Matriz de enlace (linkage)

La figura 4.1 (matriz de distancias) muestra bloques bien delimitados que indican similitudes internas en determinadas secciones. La figura 4.2 (matriz de enlace, con dendrograma) confirma esa segmentación en torno a tres clústeres principales, demostrando que los datos se agrupan de forma clara y consistente.

Ahora, se estandarizan las columnas, se calcula la matriz de distancias euclídeas sobre los datos normalizados y se representa un heatmap. Esto permite comparar cómo cambian las distancias una vez que cada variable se ha escalado, evitando que un rango numérico mayor distorsione el análisis.

```

1 distance_std = distance.cdist(data_std, data_std, "euclidean")
2
3 plt.figure(figsize=(8, 6))
4 data_std_distance = pd.DataFrame(distance_std, index = data_std.index, columns = data_num.
5     index)
5 sns.heatmap(data_std_distance, annot=False, cmap="YlGnBu", fmt=".1f")
6 plt.show()

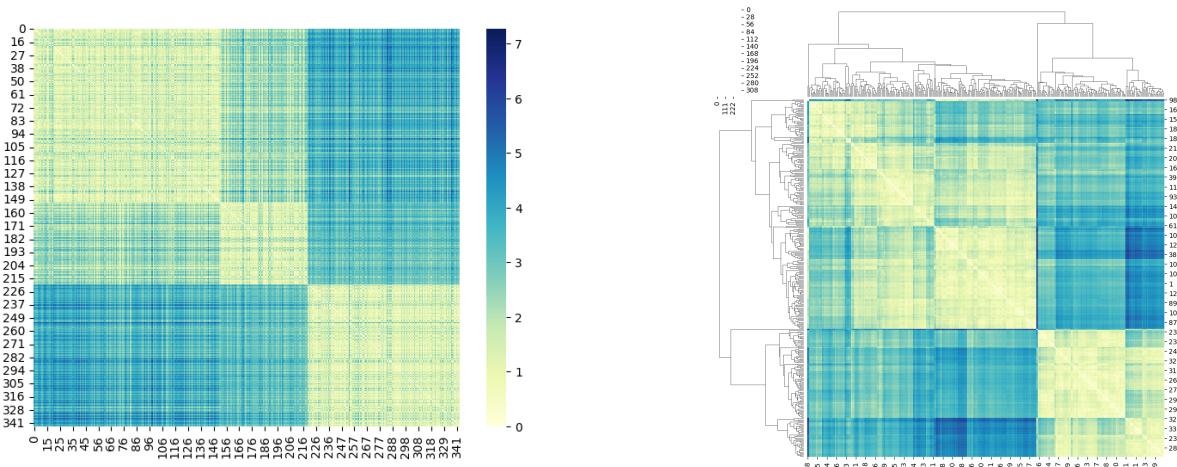
```

Se obtiene la matriz de enlace tras estandarizar la matriz de distancias, se ordenan las muestras según el dendrograma y, por último, se visualizan los datos reordenados en un heatmap. Esto refuerza la separación en clusters, ahora controlando las diferencias de escala entre variables.

```

1 linkage = sns.clustermap(data_std_distance, cmap="YlGnBu", fmt=".1f", annot=False, method='
    average').dendrogram_row.linkage
2 order = pd.DataFrame(linkage, columns=['cluster_1', 'cluster_2', 'distance', 'new_count']).
    index
3 reordered_data = data_num.reindex(index=order, columns=order)
4 sns.heatmap(reordered_data, cmap="YlGnBu", fmt=".1f", cbar=False)
5 plt.show()

```

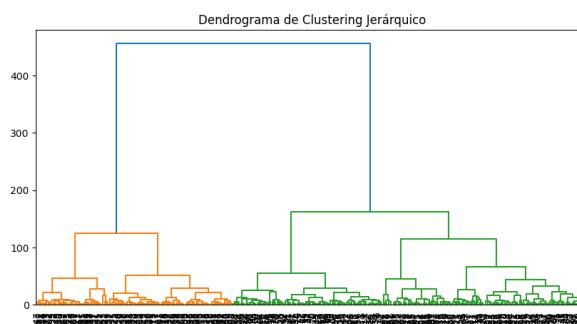


Por último, se genera la matriz de enlace con el método de Ward, y se construye un dendrograma. Así se observa cómo se agrupan jerárquicamente las observaciones tras la estandarización de los datos.

```

1 linkage_matrix = sch.linkage(data_std_distance, method='ward')
2 plt.figure(figsize=(10, 5))
3 dendrogram = sch.dendrogram(linkage_matrix, labels=data_std.index, leaf_font_size=9,
    leaf_rotation=90)
4 plt.title("Dendrograma de Clustering Jerárquico")
5 plt.show()

```



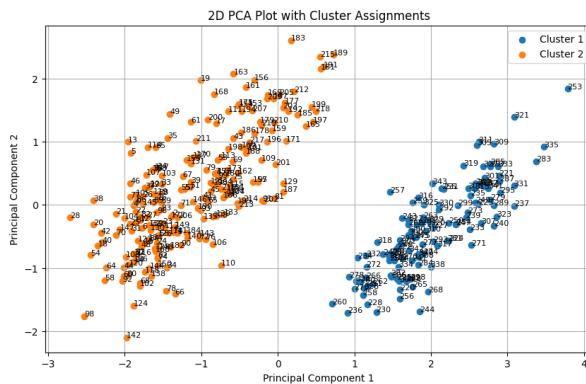
Observando la estructura del dendrograma podemos hacernos una idea de cual podría ser el número más adecuado de cluster. Por ejemplo, en nuestro caso podría ser dos. En ese caso podemos guardar los 2 cluster que hemos elegido para más tarde poder caracterizar cada uno.

```
1 num_clusters = 2
2 cluster_assignments = sch.fcluster(linkage_matrix, num_clusters, criterion='maxclust')
3 datos['Cluster2'] = cluster_assignments
```

Luego, para visualizar mejor la separación, se realiza un PCA a dos componentes y se dibuja un diagrama de dispersión, coloreando cada punto según su pertenencia al clúster. Este enfoque confirma cómo se agrupan las observaciones en un plano bidimensional, destacando la coherencia de los grupos elegidos.

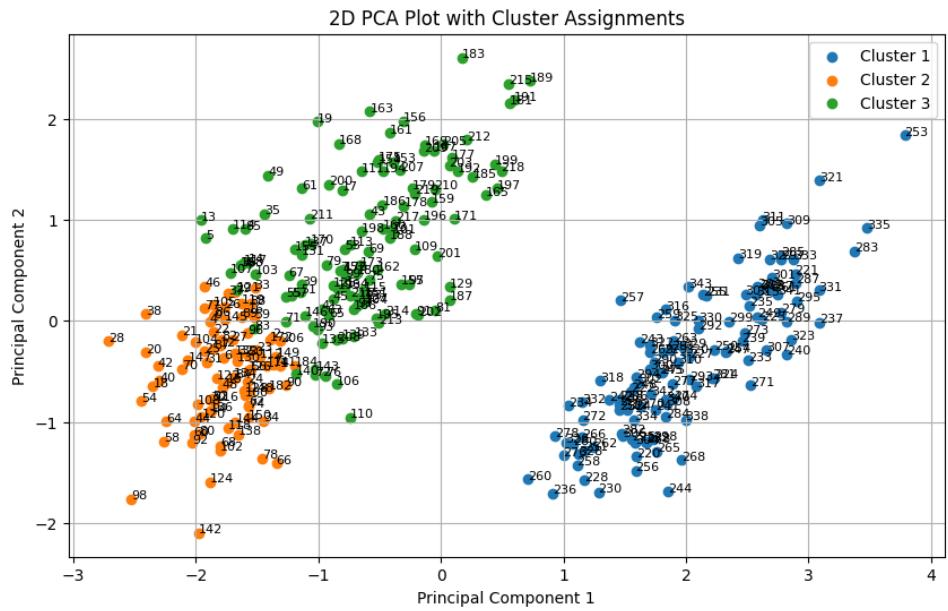
```
1 pca = PCA(n_components=2)
2 principal_components = pca.fit_transform(data_std)
3 data_pca = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
4 plt.figure(figsize=(10, 6))

5
6 for cluster in np.unique(cluster_assignments):
7     plt.scatter(data_pca.loc[cluster_assignments == cluster, 'PC1'],
8                 data_pca.loc[cluster_assignments == cluster, 'PC2'],
9                 label=f'Cluster_{cluster}')
10 for i, row in data_pca.iterrows():
11     plt.text(row['PC1'], row['PC2'], str(data_std.index[i]), fontsize=8)
12 plt.title("2D PCA Plot with Cluster Assignments")
13 plt.xlabel("Principal Component 1")
14 plt.ylabel("Principal Component 2")
15 plt.legend()
16 plt.grid()
17 plt.show()
```



Por probar también, se eligen tres clústeres a partir del dendrograma, se añaden al DataFrame y se realiza un PCA a dos dimensiones para visualizar la agrupación en un diagrama de dispersión.

```
1 num_clusters = 3
2 cluster_assignments = sch.fcluster(linkage_matrix, num_clusters, criterion='maxclust')
3 datos['Cluster3'] = cluster_assignments
4
5 pca = PCA(n_components=2)
6 principal_components = pca.fit_transform(data_std)
7 data_pca = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
8 plt.figure(figsize=(10, 6))
9 for cluster in np.unique(cluster_assignments):
10     plt.scatter(data_pca.loc[cluster_assignments == cluster, 'PC1'],
11                 data_pca.loc[cluster_assignments == cluster, 'PC2'],
12                 label=f'Cluster_{cluster}')
13 for i, row in data_pca.iterrows():
14     plt.text(row['PC1'], row['PC2'], str(data_num.index[i]), fontsize=8)
15 plt.legend()
16 plt.grid()
17 plt.show()
```



5

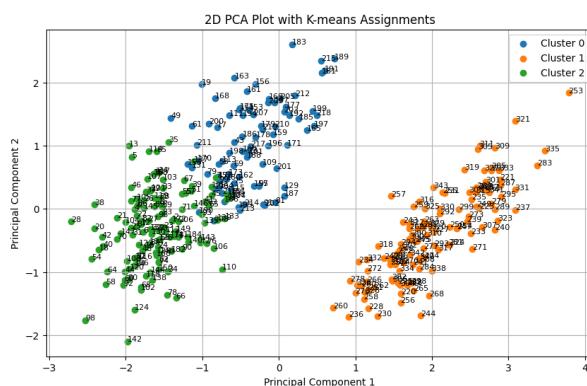
Agrupamiento K-Means y selección del número óptimo de grupos

En esta sección se aplica el algoritmo de K-Means para agrupar los datos en diferentes clústeres. K-Means es un método ampliamente utilizado por su simplicidad y eficiencia a la hora de descubrir patrones y subgrupos en grandes volúmenes de información. Se explorarán varios valores de k y se recurrirá a técnicas como el método del codo para estimar el número óptimo de grupos. De esta forma, se pretende encontrar una partición que refleje de manera equilibrada la estructura subyacente de los datos.

Empezamos ajustando el modelo de K-means con k=3 (clústeres) sobre los datos estandarizados y se obtienen las etiquetas de clúster para cada observación. Luego, se proyectan las muestras en dos componentes principales (PCA) y se genera un diagrama de dispersión, coloreando cada punto según su asignación al clúster. Esto permite comparar visualmente la segmentación de los datos en el plano bidimensional y comprobar la separación entre los tres grupos.

```

1 k = 3
2 kmeans = KMeans(n_clusters=k, random_state=0)
3 kmeans.fit(data_std)
4 kmeans_cluster_labels = kmeans.labels_
5 plt.figure(figsize=(10, 6))
6 for cluster in np.unique(kmeans_cluster_labels):
7     plt.scatter(data_pca.loc[kmeans_cluster_labels == cluster, 'PC1'],
8                 data_pca.loc[kmeans_cluster_labels == cluster, 'PC2'],
9                 label=f'Cluster {cluster}')
10 for i, row in data_pca.iterrows():
11     plt.text(row['PC1'], row['PC2'], str(data.index[i]), fontsize=8)
12 plt.legend()
13 plt.grid()
14 plt.show()
```



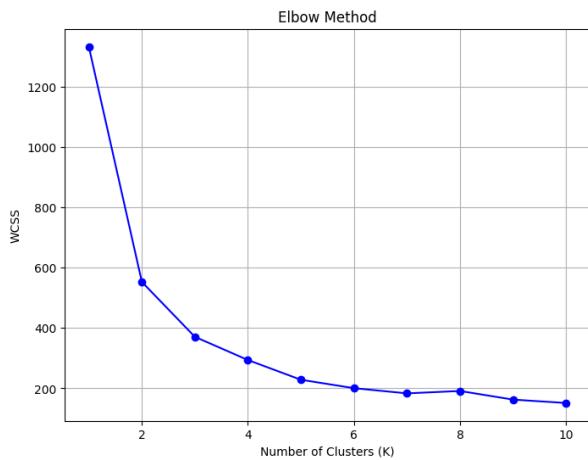
Ahora, calculamos la Within-Cluster Sum of Squares (WCSS) para diferentes valores de K (del 1 al 10) y se representa en un gráfico. El objetivo es localizar el “codo” o punto donde añadir más clusters apenas reduce el WCSS, ayudando a elegir el número óptimo de grupos en K-means.

```

1 wcss = []
2 for k in range(1, 11):
3     kmeans = KMeans(n_clusters=k, random_state=0)
4     kmeans.fit(data_std)
5     wcss.append(kmeans.inertia_)

6
7 plt.figure(figsize=(8, 6))
8 plt.plot(range(1, 11), wcss, marker='o', linestyle='-', color='b')
9 plt.title('Elbow Method')
10 plt.xlabel('Number of Clusters (K)')
11 plt.ylabel('WCSS')
12 plt.grid(True)
13 plt.show()

```



Se puede apreciar que el WCSS deja de reducirse drásticamente a partir de $k = 3$, y coincide con lo hallado por el clúster jerárquico. Con un cuarto clúster la mejora sería muy poca.

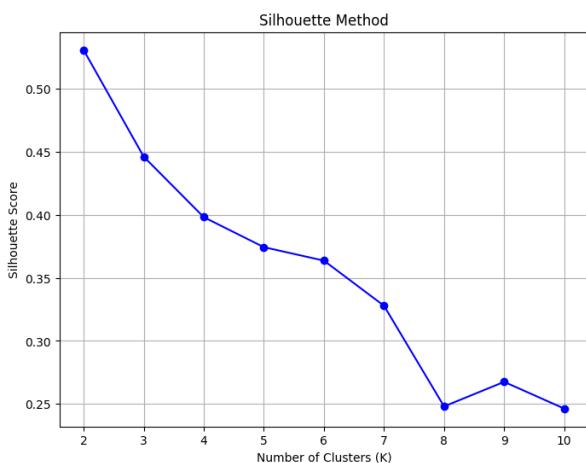
6

Validación del agrupamiento

En esta sección se profundiza en la validación de los resultados obtenidos tras el proceso de agrupamiento, recurriendo a métricas específicas como la puntuación de silueta. El objetivo es valorar hasta qué punto los clústeres formados son coherentes internamente (elementos similares agrupados) y están bien separados entre sí (clústeres distintos). A través de estas medidas se puede determinar la efectividad del algoritmo de clustering para capturar la estructura inherente de los datos y, en última instancia, reforzar la confianza en la segmentación propuesta.

Primero, calculamos la puntuación de silueta para distintos valores de K (del 2 al 10) en el algoritmo de K-means. La gráfica resultante muestra cómo cambia dicha puntuación con cada número de clústeres, ayudando a identificar en qué punto se maximiza la calidad de la separación entre los grupos (máximo de la curva)

```
1 silhouette_scores = []
2 for k in range(2, 11):
3     kmeans = KMeans(n_clusters=k, random_state=0)
4     kmeans.fit(data_std)
5     labels = kmeans.labels_
6     silhouette_avg = silhouette_score(data_std, labels)
7     silhouette_scores.append(silhouette_avg)
8
9 plt.figure(figsize=(8, 6))
10 plt.plot(range(2, 11), silhouette_scores, marker='o', linestyle='--', color='b')
11 plt.title('Silhouette Method')
12 plt.xlabel('Number of Clusters (K)')
13 plt.ylabel('Silhouette Score')
14 plt.grid(True)
15 plt.show()
```



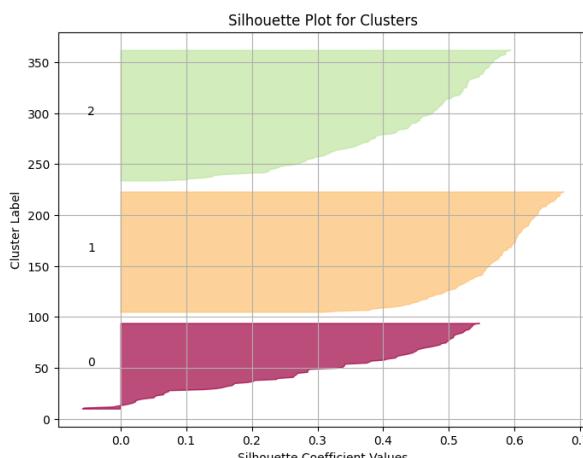
Aunque el pico de la silueta se alcanza con K=2, optar por 3 clústeres también arroja un valor de silueta razonablemente alto y permite una segmentación más detallada. De este modo se capturan matrices adicionales en los datos, facilitando la identificación de un tercer grupo con características distintivas que podría ser relevante según el contexto o la interpretación de los resultados.

Se entrena K-means con k=3 sobre los datos estandarizados y se calcula la silueta a nivel individual con silhouette_samples, quedando en silhouette_values la medida que indica, para cada observación, cuán cerca está de su propio clúster en comparación con otros clústeres.

```

1 kmeans = KMeans(n_clusters=3, random_state=0)
2 kmeans.fit(data_std)
3 labels = kmeans.labels_
4 silhouette_values = silhouette_samples(data_std, labels)
5
6 plt.figure(figsize=(8, 6))
7 y_lower = 10
8 for i in range(3):
9     ith_cluster_silhouette_values = silhouette_values[labels == i]
10    ith_cluster_silhouette_values.sort()
11    size_cluster_i = ith_cluster_silhouette_values.shape[0]
12    y_upper = y_lower + size_cluster_i
13    color = plt.cm.get_cmap("Spectral")(float(i) / 3)
14    plt.fill_betweenx(np.arange(y_lower, y_upper),
15                      0, ith_cluster_silhouette_values,
16                      facecolor=color, edgecolor=color, alpha=0.7)
17    plt.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
18    y_lower = y_upper + 10
19 plt.title("Silhouette Plot for Clusters")
20 plt.xlabel("Silhouette Coefficient Values")
21 plt.ylabel("Cluster Label")
22 plt.grid(True)
23 plt.show()

```



Sin embargo, en el diagrama de silueta por clúster, se observa que, si bien hay cierta variabilidad en los valores para cada clúster, la mayor parte de las observaciones presenta valores positivos y relativamente altos, lo cual sugiere una buena cohesión interna. Algunos puntos quedan con silueta más baja (e incluso cercana a cero), reflejando que, para esos casos concretos, la asignación de clúster no es tan clara. Aun así, en conjunto, la estructura de clústeres elegida muestra una segmentación razonablemente diferenciada.

Para terminar, vamos a dedicar la última parte de esta sección a la caracterización de los clústeres. Se agregan las etiquetas de clúster al DataFrame, se reordenan las filas por la columna "label" y, finalmente, se calcula la media de cada grupo. De ese modo, cluster_centroids (en datos estandarizados) y cluster_centroids_orig (en datos originales) recogen, para cada clúster, el valor medio de sus variables, sirviendo como "centro" o "perfil" representativo de cada grupo.

```

1 data_std['label'] = labels
2 data_std_sort = data_std.sort_values(by="label")
3 data_std = data_std.set_index(data_num.index)

1 cluster_centroids = data_std_sort.groupby('label').mean()
2 cluster_centroids.round(2)
3 \begin{lstlisting}[language=python]

```

Table 6.1: Centroides de los datos estandarizados por clúster

label	bill_length_mm_z	bill_depth_mm_z	flipper_length_mm_z	body_mass_g_z
0	0.67	0.81	-0.29	-0.38
1	0.65	-1.10	1.16	1.10
2	-1.05	0.49	-0.88	-0.76

```
1 data_num['label'] = labels
2 data_sort = data_num.sort_values(by="label")
3 cluster_centroids_orig = data_sort.groupby('label').mean()
4 cluster_centroids_orig.round(2)
```

Table 6.2: Centroides de los datos originales por clúster

label	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
0	47.66	18.75	196.92	3898.24
1	47.57	15.00	217.24	5092.44
2	38.28	18.12	188.63	3593.80

7

Comparación entre agrupamiento jerárquico y K-Means

Ambos métodos identificaron patrones claros en los datos, sugiriendo la existencia de dos o tres grupos principales. El agrupamiento jerárquico (con método de Ward y distancia euclídea) permitió visualizar cómo se fusionan progresivamente las observaciones a lo largo del dendrograma, ayudando a decidir el número de clústeres tras inspeccionar los saltos más grandes en las distancias de enlace. Por otro lado, K-Means impone la partición con un número de clústeres predefinido y, mediante métricas como el método del codo o la silueta, se confirmó que dos o tres grupos maximizan la cohesión interna y la separación entre grupos. En muchos casos, las asignaciones coinciden, aunque con ligeras variaciones en ciertos puntos donde los métodos difieren en sus criterios de partición (jerárquico se basa en distancias de enlace; K-Means minimiza la varianza intra-clúster).

8

Interpretación de los grupos

Los grupos obtenidos se asocian con los rasgos morfológicos de los pingüinos (por ejemplo, masa corporal, longitud y profundidad del pico, longitud de aleta). En términos de especies:

- Un primer grupo tiende a incluir individuos con menor masa corporal y aletas algo más cortas, rasgos característicos de pingüinos Adelie.
- El segundo agrupa pingüinos con medidas intermedias (picos de distinta profundidad y pesos no tan extremos), coincidiendo en gran parte con Chinstrap.
- El tercero engloba los ejemplares de mayor tamaño y peso, rasgos que suelen caracterizar a Gentoo.

Así, en la mayoría de los casos, cada clúster refleja una combinación de atributos físicos que coincide razonablemente con la clasificación conocida de las especies, si bien puede haber individuos atípicos que no encajan perfectamente en estas categorías.

9

Conclusiones

En conjunto, tanto el agrupamiento jerárquico como K-Means han identificado grupos bien diferenciados al analizar las variables morfológicas de los pingüinos. Se ha comprobado que 2 o 3 clústeres explican de forma adecuada la variabilidad observada, capturando la separación natural existente entre Adelie, Gentoo y Chinstrap. Sin embargo, se presentan algunos retos o limitaciones:

- Outliers o valores extremos pueden distorsionar la formación de clústeres.
- La escala de las variables requiere estandarización para evitar que una variable con mayor rango numérico domine el análisis.
- Es posible que algunos individuos no encajen perfectamente en un único clúster, sobre todo si sus medidas se solapan con las de otras especies.

Pese a ello, la combinación de distintas métricas de validación (método del codo, silueta) y la proyección en PCA corroboran la solidez de estos grupos y ofrecen un panorama claro de la estructura subyacente del conjunto de datos.