Práctiva de ACP y Clustering

Guillermo Díaz Aguado

In [271…
```
import pandas as pd  # Manipulación y análisis de datos tabulares (filas y colum
import numpy as np  # Operaciones numéricas y matriciales.
import seaborn as sns  # Visualización estadística de datos.
import matplotlib.pyplot as plt  # Creación de gráficos y visualizaciones.

# Matplotlib es una herramienta versátil para crear gráficos desde cero,
# mientras que Seaborn simplifica la creación de gráficos estadísticos.

from sklearn.decomposition import PCA  # Implementación del Análisis de Componen
from sklearn.preprocessing import StandardScaler  # Estandarización de datos par
```

Introducción

Descripción del conjunto de datos.

El conjunto de datos penguins de la librería seaborn de Python contiene la siguiente información sobre diferentes especies de pingüinos:

- species: Es la especie de pingüino. Hay tres especies en el conjunto de datos: Adelie, Chinstrap y Gentoo.
- island: Representa la isla donde se recopilaron los datos. Las islas son Biscoe, Dream y Torgersen.
- bill_length_mm: Longitud del pico en milímetros.
- bill_depth_mm: Profundidad del pico en milímetros.
- flipper_length_mm: Longitud de la aleta en milímetros.
- body_mass_g: Masa corporal del pingüino en gramos.
- sex: Género del pingüino, con las categorías Male (macho), Female (hembra) o NaN si la información no está disponible.

Tareas de carga y exploración de los datos:

- Carga el conjunto de datos de Palmer Penguins utilizando Python.

In [272…
```
# Cargamos el set de datos
df = sns.load_dataset("penguins")

df.head()
```

Out[272...

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0
3	Adelie	Torgersen	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0

- Muestra las estadísticas básicas descriptivas sobre el conjunto de datos.

In [273...

```
def muestra_estadist(df):
    variables = list(df.columns)


    estadisticos = pd.DataFrame({
        "Minimo": df[variables].min(),
        "Percentil 25": df[variables].quantile(0.25),
        "Mediana": df[variables].median(),
        "Percentil 75": df[variables].quantile(0.75),
        "Maximo":df[variables].max(),
        "Desviación estándar": df[variables].std(),
        "Varianza": df[variables].var(),
        "Coeficiente de variacion": df[variables].std()/df[variables].mean(),
        "Datos perdidos": df[variables].isna().sum()
    })
    return estadisticos

muestra_estadist(df.drop(columns=["species", "island", "sex"]))
```

Out[273...

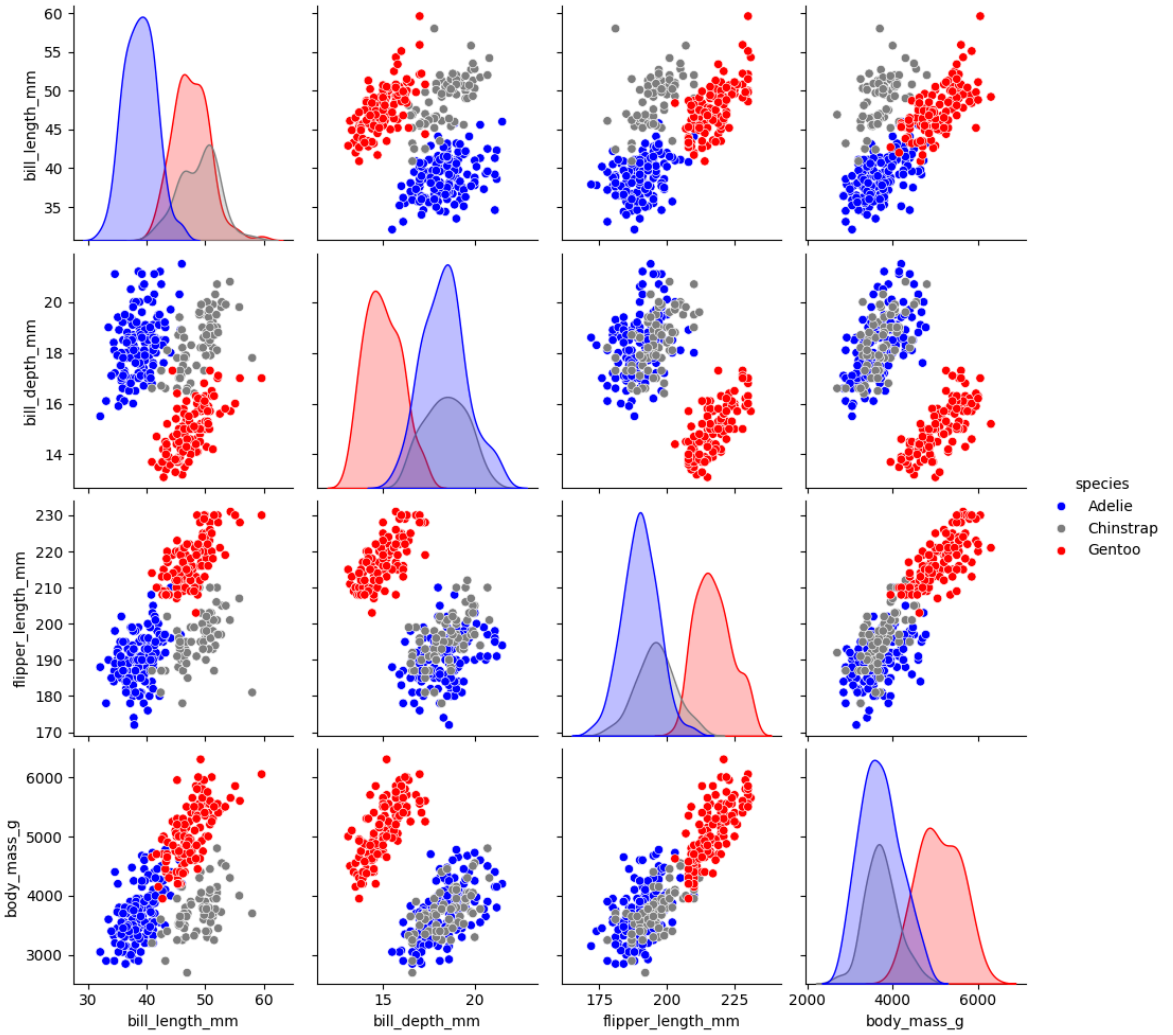
	Minimo	Percentil 25	Mediana	Percentil 75	Maximo	Desviación estándar	V
bill_length_mm	32.1	39.225	44.45	48.5	59.6	5.459584	29
bill_depth_mm	13.1	15.600	17.30	18.7	21.5	1.974793	3
flipper_length_mm	172.0	190.000	197.00	213.0	231.0	14.061714	197
body_mass_g	2700.0	3550.000	4050.00	4750.0	6300.0	801.954536	643131

In [274...

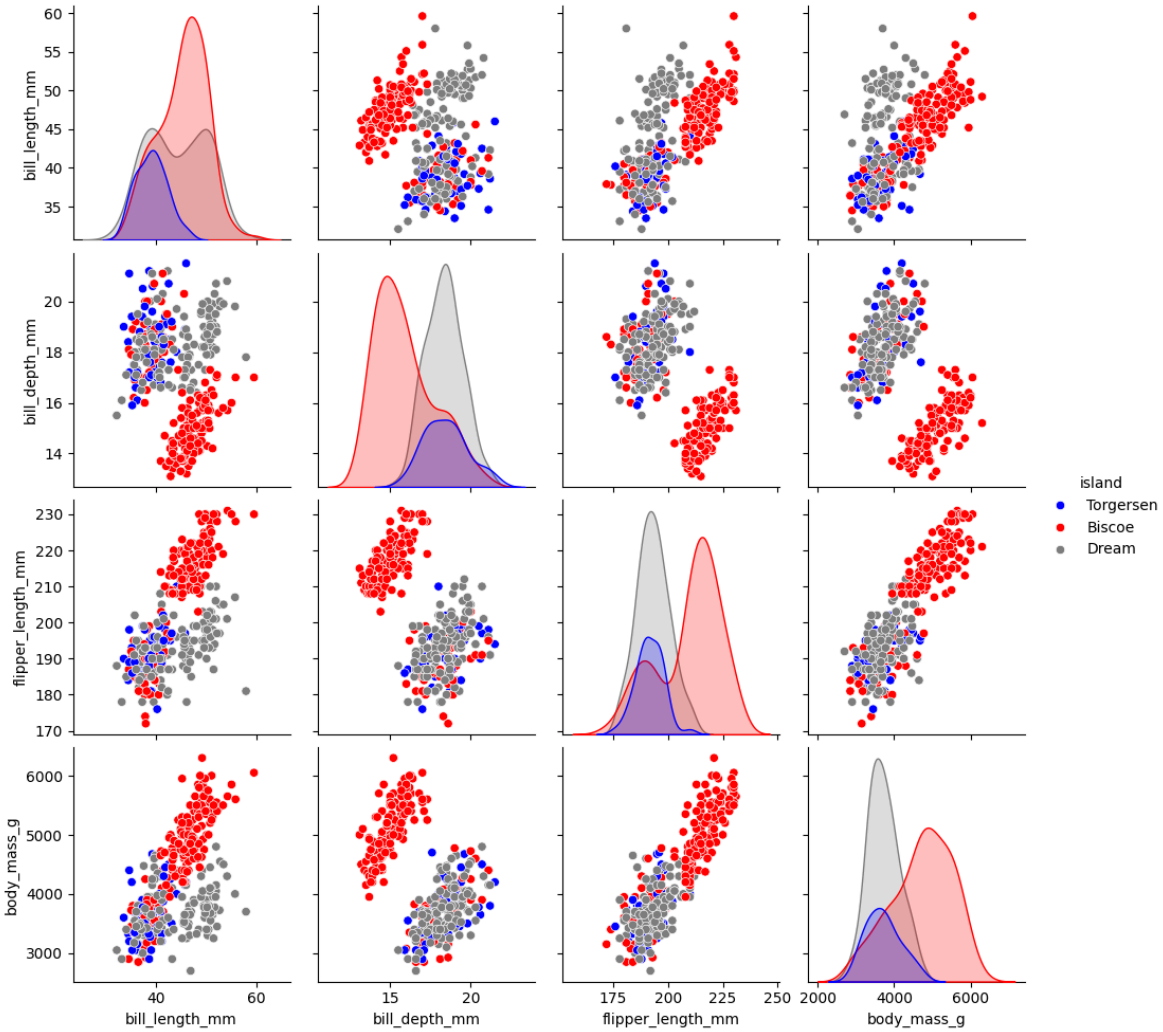
```
df = df.dropna(axis=0)
df_cat = df[["species", "island", "sex"]]
df_sin_cat = df.drop(columns=["species", "island", "sex"])
variables = list(df_sin_cat.columns)
```

- Realiza un visualizado de los datos mediante graficas de dispersión o cualquier otro método.

In [275…
```
colores_species = {"Adelie":"blue", "Chinstrap":"grey", "Gentoo":"red"}
sns.pairplot(df,hue="species", palette=colores_species)
plt.show()
```



In [276…
```
colores_islas = {"Torgersen":"blue", "Dream":"grey", "Biscoe":"red"}
sns.pairplot(df,hue="island", palette=colores_islas)
plt.show()
```



Una cosa que nos empieza a chirriar al ver estas graficas es la similitud en los registros que contienen "Gentoo" en la especie y "Biscoe" en la isla siguen una distribución muy parecida (he definido el color de ambas caracteristicas para que sean el mismo y se pueda distinguir mejor).

In [277…
```
num_gentoo = df['species'].value_counts().get('Gentoo', 0)
num_biscoe = df['island'].value_counts().get('Biscoe', 0)
num_gentoo_in_biscoe = ((df['species'] == 'Gentoo') & (df['island'] == 'Biscoe')

num_gentoo, num_gentoo_in_biscoe, num_biscoe
```

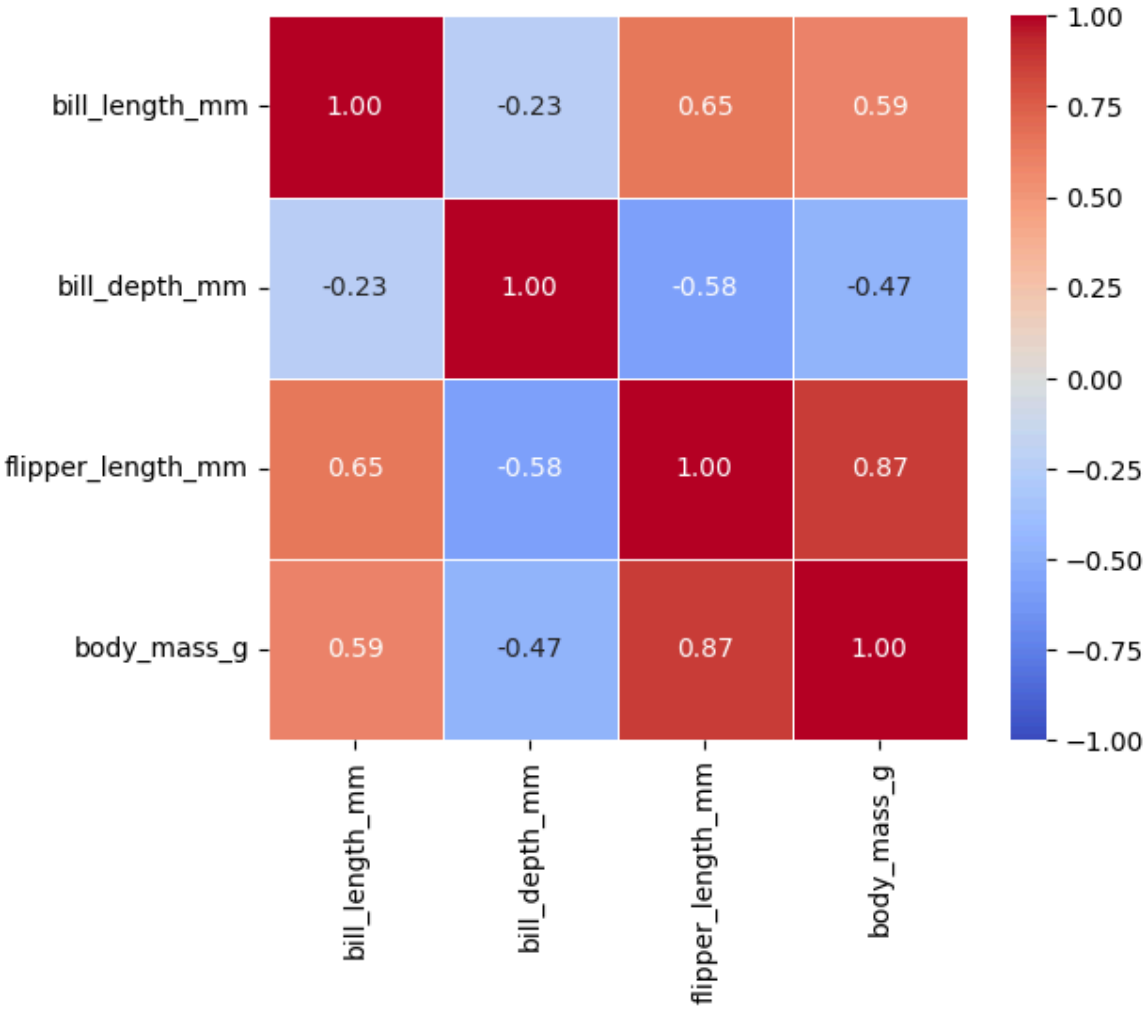
Out[277… (119, 119, 163)

Y como podemos ver en el código de antes, todos los registros con "Gentoo" de especie, proviene de la isla "Biscoe", aunque no todos los pigüinos en "Biscoe" son de la especie "Gentoo". Esto hará que tengan una relación muy estrecha estos valores.

1. Calcula la matriz de correlaciones y su representación gráfica.

In [278…
```
df = df_sin_cat
correlacion = df.corr()

# Crea una nueva figura de dimensiones 10x8 para el gráfico.
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(correlacion, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5,
plt.show()
```



Un breve resumen de lo que representa la gráfica anterior es:

- Las variables mas correlacionadas son "flipper_length_mm" con "body_mass_g", esto significa que cuanto mayor es la longitud de su aleta, más masa corporal tendrán, y viceversa.
- Las variables más correlacionadas de manera inversa son "flipper_length_mm" y "bill_depth_mm". Lo que indica que cuanto mayor el pico, menor será su aleta o al reves. Aunque está sea la variable mas correlacionada de forma inversa, tan solo es la tercera variable correlacionada.

Análisis de Componentes Principales ACP

2. Realiza un análisis de componentes principales (PCA) sobre la matriz de correlaciones, calculando un número adecuado de componentes

Empezaremos estandarizando los datos, ya que es siempre aconsejable para evitar que la magnitud influya de ninguna manera

In [279…
```
df_std = pd.DataFrame(StandardScaler().fit_transform(df))
print(df_std)
```

```
           0         1         2         3
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
3   -1.335566  1.085877 -0.569284 -0.941606
4   -0.859415  1.747026 -0.783651 -0.692852
..        ...       ...       ...       ...
328  0.587352 -1.762145  0.931283  0.892957
329  0.514098 -1.457000  1.002739  0.799674
330  1.173384 -0.744994  1.502928  1.919069
331  0.221082 -1.202712  0.788372  1.234995
332  1.081817 -0.541564  0.859828  1.483749

[333 rows x 4 columns]
```

Con los datos ya estandarizados realizaremos el cálculo de las componentes principales con la libreria de **scikit.learn**. En el primer ajuste usaremos como número de componentes el número de variables catégoricas que están presentes en nuestro DataFrame.

In [280…
```
num_variables = len(df_std.columns)

pca = PCA(n_components=num_variables)
fit = pca.fit(df_std)
```

Una vez ajustado, valoraremos los resultados.

In [281…
```
autovalores = fit.explained_variance_

# Obtener la varianza explicada como un porcentaje
var_explicada = fit.explained_variance_ratio_

# Calcular la varianza explicada acumulada
var_acumulada = np.cumsum(var_explicada)

# Crear un DataFrame de pandas con los datos anteriores y establecer índice
data = {'Autovalores': autovalores, 'Variabilidad Explicada': var_explicada,
        'Variabilidad Acumulada': var_acumulada}
tabla = pd.DataFrame(data, index=['Componente {}'.format(i) for i in range(1, fi
tabla
```

Out[281…

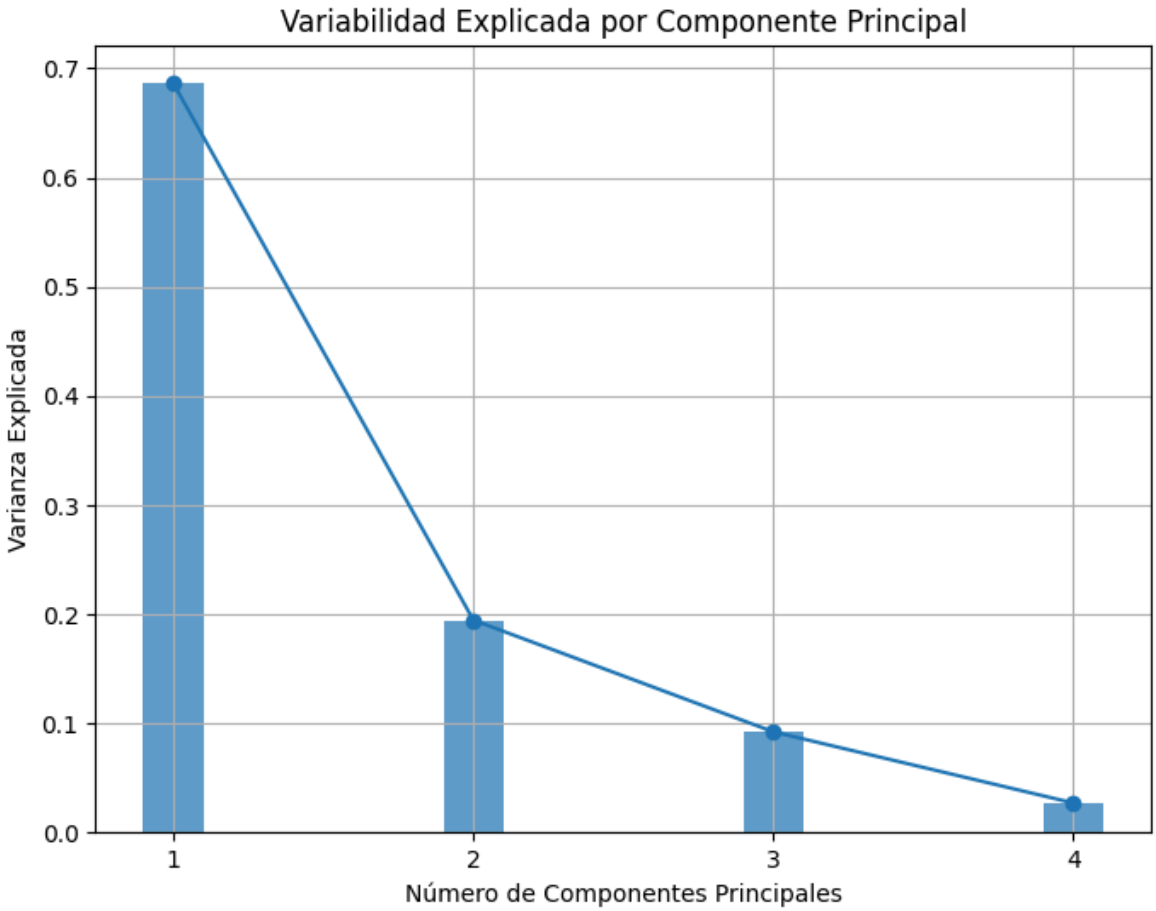
	Autovalores	Variabilidad Explicada	Variabilidad Acumulada
Componente 1	2.753625	0.686339	0.686339
Componente 2	0.780461	0.194529	0.880868
Componente 3	0.369753	0.092161	0.973029
Componente 4	0.108210	0.026971	1.000000

Como podemos ver, la primera componente contiene casi un 70% de la información de la variabilidad. En algunos casos, un 70% de la información podría ser suficiente pero no

tendría sentido usar solo una componente. Igualmente dibujaremos una gráfica para representar la prueba del codo y tener una mejor visualización de los datos.

In [282…

```
def plot_varianza_explicada(var_explicada, n_components):
    # Crear un rango de números de componentes principales de 1 a n_components
    num_componentes_range = np.arange(1, n_components + 1)

    # Crear una figura de dimensiones 8x6
    plt.figure(figsize=(8, 6))

    # Trazar la varianza explicada en función del número de componentes principa
    plt.plot(num_componentes_range, var_explicada, marker='o')

    # Etiquetas de los ejes x e y
    plt.xlabel('Número de Componentes Principales')
    plt.ylabel('Varianza Explicada')

    # Título del gráfico
    plt.title('Variabilidad Explicada por Componente Principal')

    # Establecer las marcas en el eje x para que coincidan con el número de comp
    plt.xticks(num_componentes_range)

    # Mostrar una cuadrícula en el gráfico
    plt.grid(True)

    # Agregar barras debajo de cada punto para representar el porcentaje de vari
    # - 'width': Ancho de las barras de la barra. En este caso, se establece en
    # - 'align': Alineación de las barras con respecto a los puntos en el eje x.
    #   'center' significa que las barras estarán centradas debajo de los puntos
    # - 'alpha': Transparencia de las barras. Un valor de 0.7 significa que las
    plt.bar(num_componentes_range, var_explicada, width=0.2, align='center', alp

    # Mostrar el gráfico
    plt.show()

plot_varianza_explicada(var_explicada, fit.n_components_)
```



Efectivamente es a partir de la segunda componente donde la pendiente deja de decrecer drásticamente, por ello un buen punto sería usar únicamente la componente principal 1, pero esto no tendría mucho sentido, ya que no seríamos capeces de mostrar toda la varianza solo con una componente. Es por ello que usaremos las dos primeras componentes, una otrogonal a la otra.

3. Realiza nuevamente el análisis de componentes principales sobre la matriz de correlaciones, esta vez indicando el número de componentes principales que hemos decidido retener.

Como hemos explicado en el apartado anterior, usaremos 2 componentes principales para representar la variabilidad.

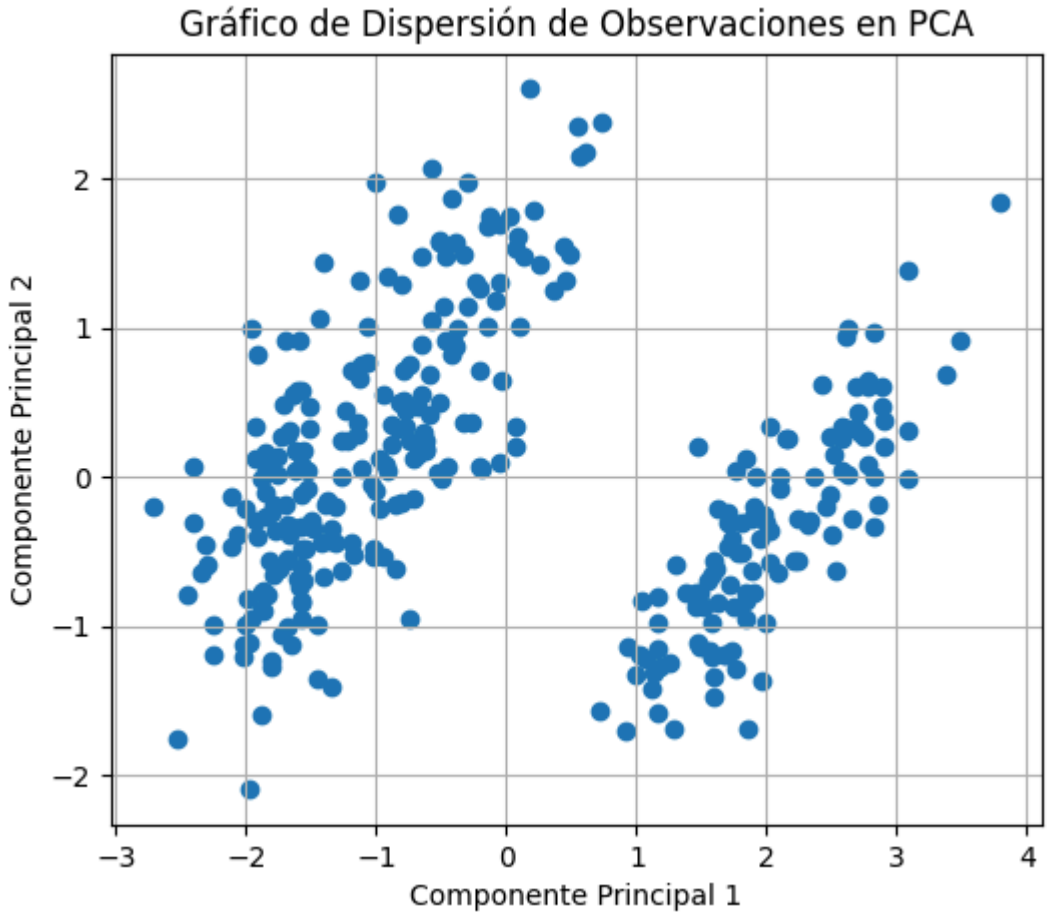
In [283…

```
n_comp = 2
pca = PCA(n_components=n_comp)
fit = pca.fit(df_std)


#Los autovectores:
autovectores = pd.DataFrame(pca.components_.T,
                            columns=[f"Autovector{i}" for i in range(1, fit.n_co
                            index = [f"{var}_z" for var in variables])

resultados_pca = pd.DataFrame(fit.transform(df_std),
                              columns = [f"Componente {i}" for i in range(1, fit
```

```
                                    index = df_std.index)

df_z_cp = pd.concat([df_std, resultados_pca], axis=1)
variables_cp = df_z_cp.columns
variables_cp
```

Out[283… `Index([0, 1, 2, 3, 'Componente 1', 'Componente 2'], dtype='object')`

In [284…
```
# Gráfico de dispersión de observaciones en los nuevos ejes
plt.figure(figsize=(6, 5))
plt.scatter(resultados_pca.iloc[:, 0], resultados_pca.iloc[:, 1])
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Gráfico de Dispersión de Observaciones en PCA')
plt.grid(True)

plt.show()
```

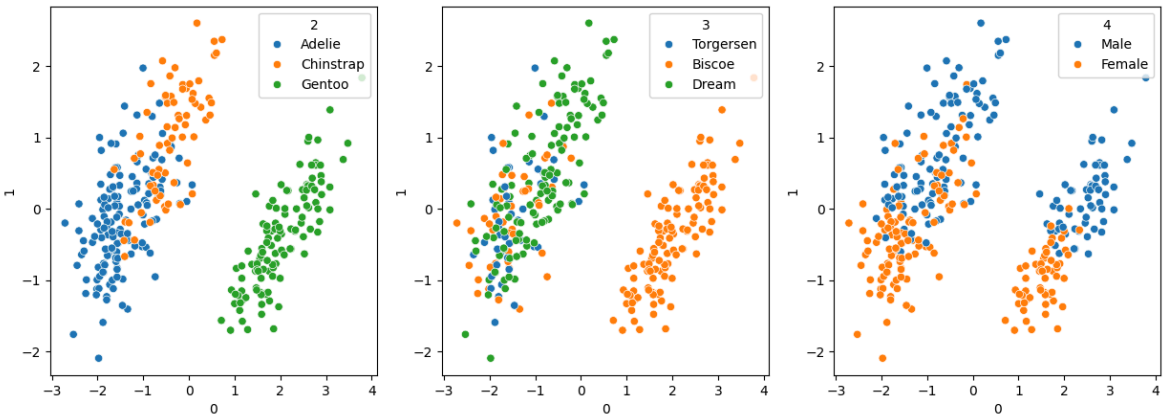


In [285…
```
df_cat = df_cat.reset_index(drop=True)
pca_cat = pd.concat((resultados_pca, df_cat), axis=1, ignore_index=True)
print(pca_cat)
```

```
            0         1       2          3       4
0   -1.853593  0.032069  Adelie  Torgersen    Male
1   -1.316254 -0.443527  Adelie  Torgersen  Female
2   -1.376605 -0.161230  Adelie  Torgersen  Female
3   -1.885288 -0.012351  Adelie  Torgersen  Female
4   -1.919981  0.817598  Adelie  Torgersen    Male
..        ...       ...     ...        ...     ...
328  1.997716 -0.976771  Gentoo     Biscoe  Female
329  1.832651 -0.784510  Gentoo     Biscoe  Female
330  2.751505  0.266556  Gentoo     Biscoe    Male
331  1.713854 -0.725875  Gentoo     Biscoe  Female
332  2.018537  0.336554  Gentoo     Biscoe    Male

[333 rows x 5 columns]
```

In [286…

```
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

sns.scatterplot(data=pca_cat, x=0, y=1, hue=2, ax=axes[0])
sns.scatterplot(data=pca_cat, x=0, y=1, hue=3, ax=axes[1])
sns.scatterplot(data=pca_cat, x=0, y=1, hue=4, ax=axes[2])
plt.show()
```



Como podemos ver en las tres gráficas anteriores podemos decir que:

1. En relación a la especie.
    - La especie Gentoo tiene los mayores valores en la componenete 1.
    - La especie Chinstrap es la que mayores valores tiene de la componente 2.
    - La especie Adelie es la que menos tiene de ambos componentes
2. En relación a la isla:
    - Poco se puede hablar solo que Gentoo pertenece a Biscoe
3. En relación al sexo:
    - La mayoria de las hembres de cada especie tiene menor componente 1 y 2 que los machos. Existe alguna hembra que destaca en su especie, pero se cuentan con las manos.

In [287…

```
# Calculo de las correalaciones entre las variables originales y las componentes
correlacion = pd.DataFrame(np.corrcoef(df_std.T, resultados_pca.T),
                           index=variables + list(resultados_pca.columns),
                           columns=variables + list(resultados_pca.columns))

correlaciones_df = correlacion.iloc[:fit.n_features_in_, fit.n_features_in_:]
```

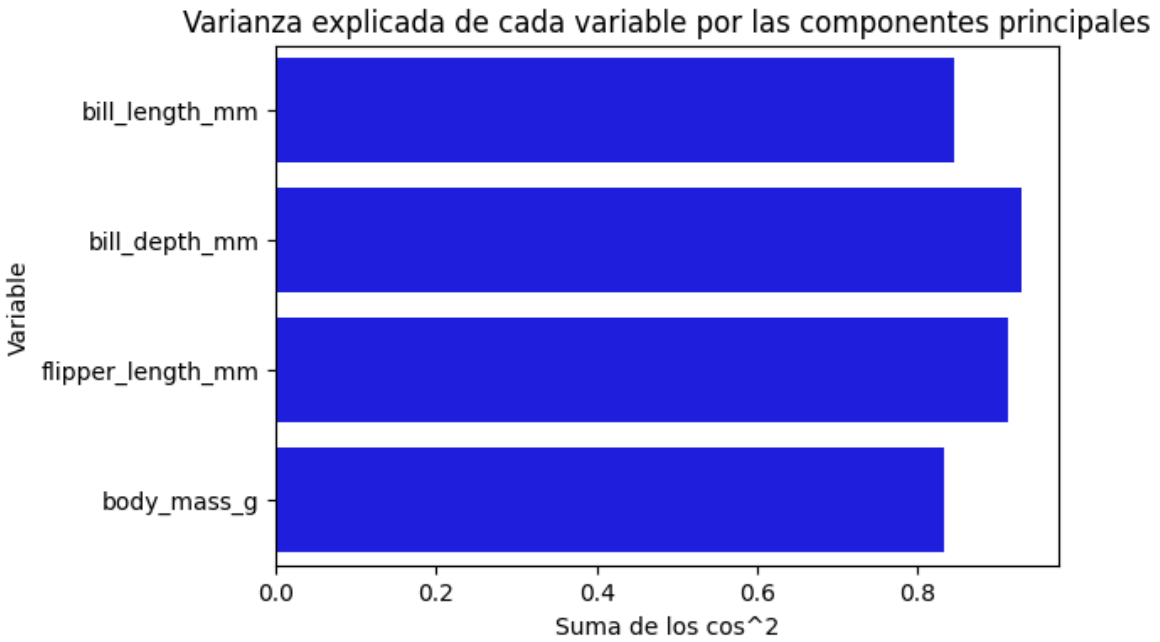
```
cos2 = correlaciones_df **2
cos2
```

Out[287…

	Componente 1	Componente 2
bill_length_mm	0.565247	0.280304
bill_depth_mm	0.437167	0.493237
flipper_length_mm	0.913454	0.000026
body_mass_g	0.829488	0.004549

In [288…

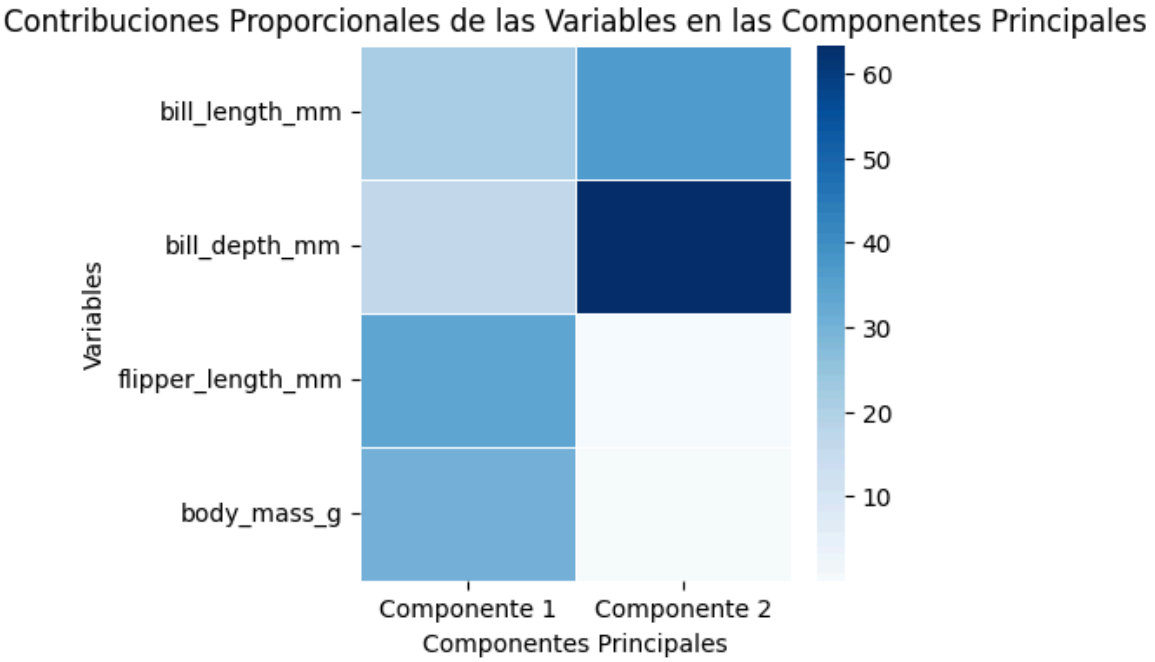
```
def plot_cos2_bars(cos2):
    plt.figure(figsize=(6,4))
    sns.barplot(x=cos2.sum(axis=1), y=cos2.index, color="blue")

    plt.xlabel("Suma de los cos^2")
    plt.ylabel("Variable")
    plt.title("Varianza explicada de cada variable por las componentes principal

    plt.show()

plot_cos2_bars(cos2)
```



In [289…

```
def plot_contribuciones_proporcionales(cos2, autovalores, n_components):
    contribuciones = cos2 * np.sqrt(autovalores)

    sumas_contribuciones = []
    for i in range(n_components):
        nombre_componente = f'Componente {i + 1}'
        suma_contribucion = np.sum(contribuciones[nombre_componente])
        sumas_contribuciones.append(suma_contribucion)

    contribuciones_proporcionales = contribuciones.div(sumas_contribuciones, axi

    plt.figure(figsize=(4,4))

    sns.heatmap(contribuciones_proporcionales, cmap='Blues', linewidths=0.5, ann
```

```
    plt.xlabel('Componentes Principales')
    plt.ylabel('Variables')

    plt.title('Contribuciones Proporcionales de las Variables en las Componentes

    plt.show()

    return contribuciones_proporcionales

autovalores = fit.explained_variance_
contribuciones_proporcionales = plot_contribuciones_proporcionales(cos2, autoval
```



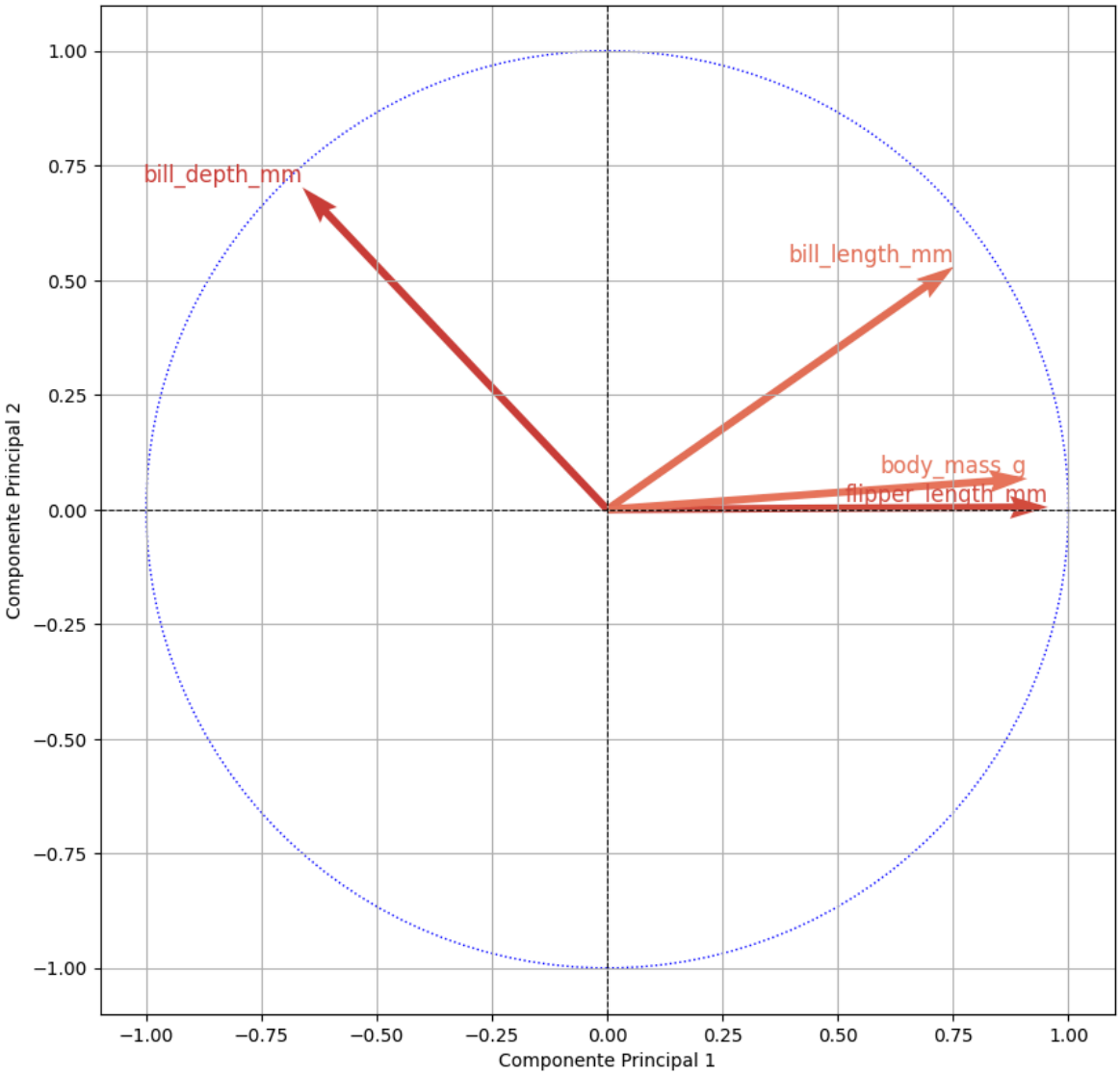
De esta gráfica se interpreta que la componente 1 está formada por todas las variables del DF, con mas peso las variables de "flipper_length_mm" y "body_mass_g". Mientras que la componente esta compuesta practicamente por "bill_depth_mm" y algo de contribución de "bill_length_mm"

In [290…

```
def plot_corr_cos(n_comp, correlaciones):
    cmap = plt.get_cmap("coolwarm")

    for i in range(n_comp):
        for j in range(i+1, n_comp):
            sum_cos2 = correlaciones.iloc[:,i] **2 + correlaciones.iloc[:,j]**2

            plt.figure(figsize=(10,10))
            circle = plt.Circle((0, 0), 1, fill=False, color="b", linestyle="dot
            plt.gca().add_patch(circle)

            for k, var_name in enumerate(correlaciones.index):
                x = correlaciones.iloc[k, i]
                y = correlaciones.iloc[k, j]

                color = cmap(sum_cos2[k])

                plt.quiver(0, 0, x, y, angles="xy", scale_units="xy", scale=1, c

                plt.text(x, y, var_name, color=color, fontsize=12, ha="right", v
```

```
            plt.axhline(0, color="black", linestyle="--", linewidth=0.8)
            plt.axvline(0, color="black", linestyle="--", linewidth=0.8)

            plt.xlabel(f'Componente Principal {i + 1}')
            plt.ylabel(f'Componente Principal {j + 1}')

            # Establecer los límites del gráfico
            plt.xlim(-1.1, 1.1)
            plt.ylim(-1.1, 1.1)

            # Mostrar el gráfico
            plt.grid()
            plt.show()


plot_corr_cos(fit.n_components_, correlaciones_df)
```

```
C:\Users\Usuario\AppData\Local\Temp\ipykernel_23844\4127350424.py:16: FutureWarni
ng: Series.__getitem__ treating keys as positions is deprecated. In a future vers
ion, integer keys will always be treated as labels (consistent with DataFrame beh
avior). To access a value by position, use `ser.iloc[pos]`
  color = cmap(sum_cos2[k])
```



En esta gráfica se puede apreciar mejor las contribuciones de cada variable a cada componente, y además puedo interpretar si estas contribuciones son positivas o

negativas:

Tanto "flipper_length_mm" como "body_mass_g" son lineas horizontales, por lo que impactarán poco en la componente 2, pero mucho en la componente 1. Además son positivas, con lo que si aumenta alguna de estas, aumentará su componente 1. En el caso de "bill_length_mm", impacta de la misma manera a la componente 1 como a la 2, incluso su impacto es positivo para ambas componentes. Finalmente para "bill_depth_mm, su contribución a las 2 componentes es el mismo en magnitud, pero para la componente 1, su impacto influirá de forma negativa

Tecnícas de Clustering

4. Determina el Número de Grupos.

Aplica el agrupamiento jerárquico al conjunto de datos y utiliza un dendrograma para sugerir un número razonable de grupos. Justifica tu elección del número de grupos. Realiza estos pasos tal y como has visto en clase y figura en los apuntes del tema.

Primero empezaremos estandarizando los datos, para que las magnitudes de estos no opaquen los cálculos de la distancia entre los registros, además que es un paso imprescindible a la hora de hacer clustering.

In [291…
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_std = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

df_std.head()
```

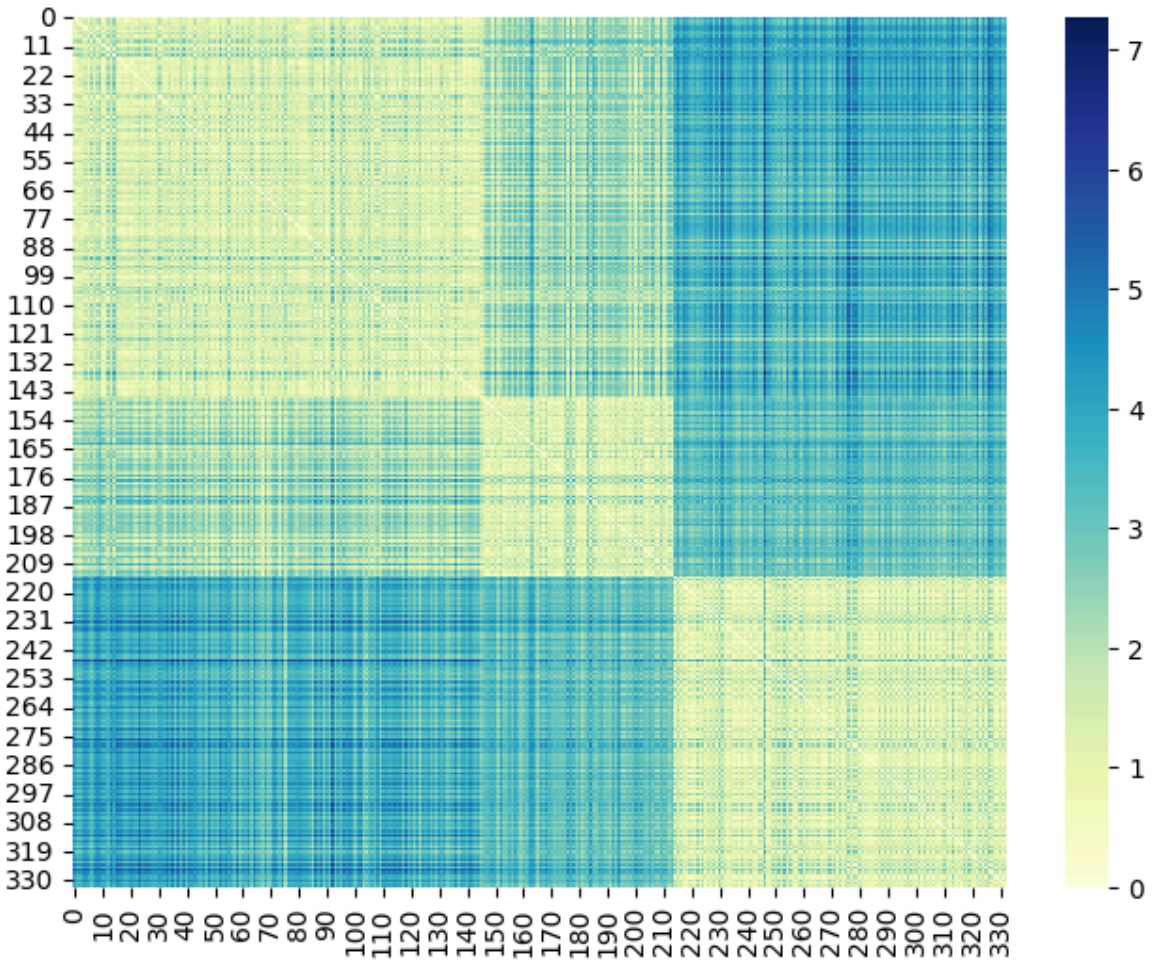
Out[291…

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
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
3	-1.335566	1.085877	-0.569284	-0.941606
4	-0.859415	1.747026	-0.783651	-0.692852

Siguiente paso vamos a realizar el calculo de las distancias de similitud (o diferencia entre estos registros). Usaremos la distancia euclidea para ver esta similitud, puesto que esta implementado de forma sencilla en python

In [292…
```
from scipy.spatial import distance

# Calculate the pairwise Euclidean distances
distance_matrix = pd.DataFrame(distance.cdist(df_std, df_std, 'euclidean'))

plt.figure(figsize=(8, 6))
```

```
sns.heatmap(distance_matrix, annot=False, cmap="YlGnBu", fmt=".1f")
plt.show()
```



Ya a partir de este gráfico empezamos a identificar los 3 posibles grupos en los cuadrados más claros (que indican cercanía entre esos registros) y lo externo a estos cuadrados claros, las bandas oscuras, nos indican la lejanía con otros grupos o clusters.

Generamos el dendrograma usando las distancias de Ward:

In [293…

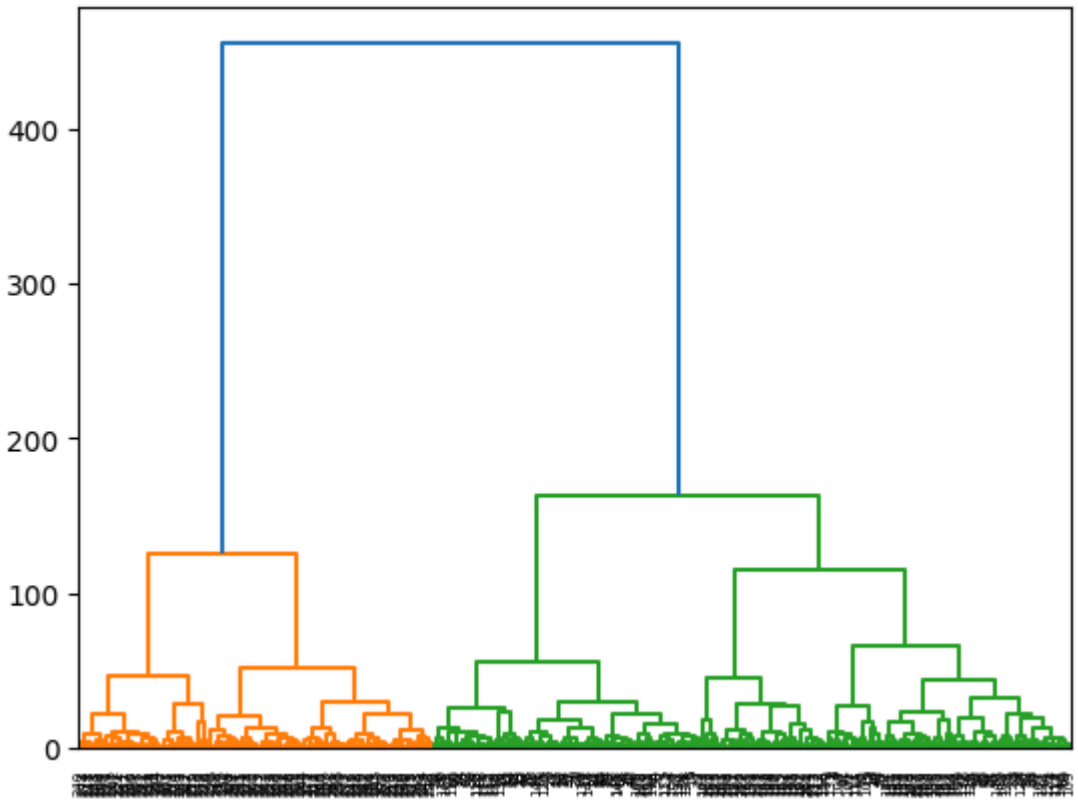
```
from scipy.cluster.hierarchy import ward

distance_std = distance.cdist(df_std, df_std, "euclidean")
distance_matrix = ward(df_std)
distance_matrix_df = pd.DataFrame(distance_matrix)

distance_matrix_df.head()
```

Out[293…

	0	1	2	3
0	220.0	304.0	0.109065	2.0
1	254.0	305.0	0.109065	2.0
2	31.0	101.0	0.126259	2.0
3	260.0	295.0	0.131750	2.0
4	70.0	137.0	0.133268	2.0

In [294…
```
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt

# Calculate the linkage matrix
df_std_distance = pd.DataFrame(distance_std, index=df_std.index, columns=df.inde
linkage_matrix = sch.linkage(df_std_distance, method='ward')

# Create the dendrogram
dendrogram = sch.dendrogram(linkage_matrix)

# Display the dendrogram
plt.show()
```

```
C:\Users\Usuario\AppData\Local\Temp\ipykernel_23844\1580049447.py:6: ClusterWarni
ng: The symmetric non-negative hollow observation matrix looks suspiciously like
an uncondensed distance matrix
  linkage_matrix = sch.linkage(df_std_distance, method='ward')
```



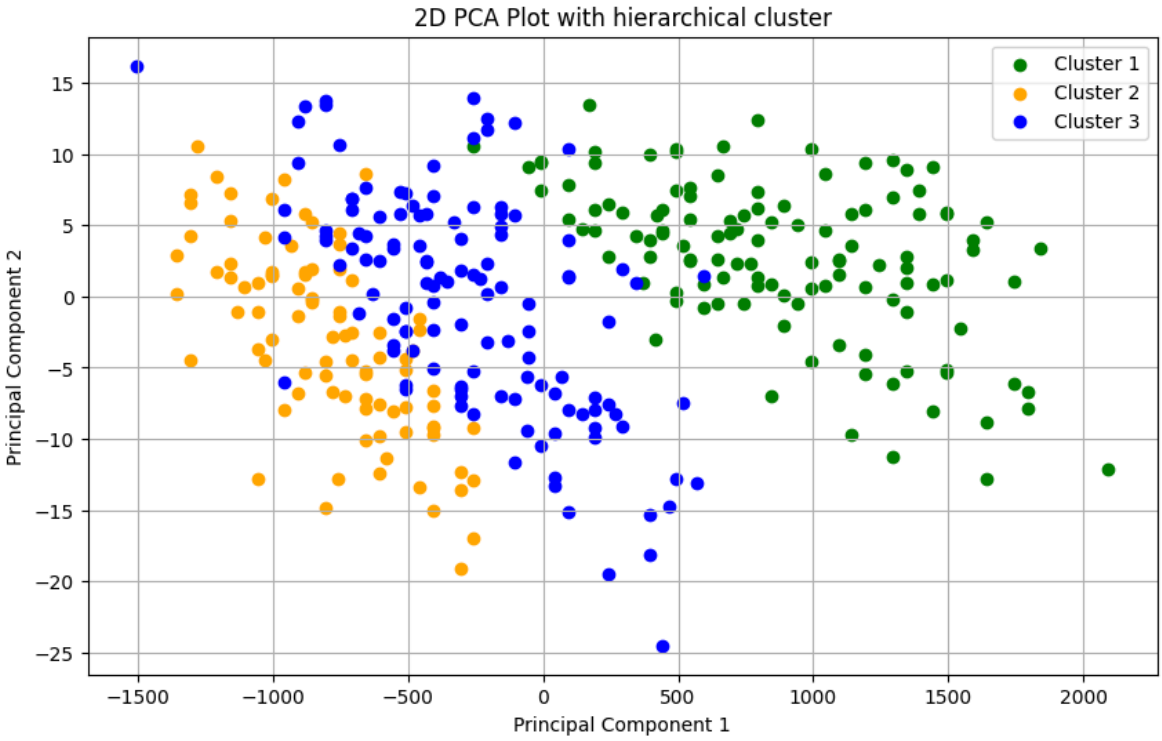
En esta gráfica, por colores divide nuestros pingüinos en dos clusters, pero realmente es posible ver que el árbol de en medio verde tiene bastante lejanía con la otra rama de este nodo. Por ello seccionaremos este árbol a la altura del 15 más o menos. Dandonos pues, 3 clusters.

In [295…
```
from scipy.cluster.hierarchy import fcluster

num_clusters = 3
cluster_assignments = fcluster(linkage_matrix, num_clusters, criterion='maxclust

df['Cluster4'] = cluster_assignments
print("Cluster Assignments:", cluster_assignments)
```

```
Cluster Assignments: [2 2 2 2 3 2 3 2 3 3 2 2 3 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 2 2 3 2 3
 2 3 2 3 2 2 3 2 3 2 3 2 3 2 2 2 3 2 3 2 3 2 3 2 3 3 3 2 3 3 2 2 3
 2 3 2 3 2 3 2 2 2 2 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 2 3 3 2 3 3 3 2 3 3 3 2
 3 2 2 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 3 2 2 3 2 2 3 2 2 2 2 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 2 3 3 3 3 3 3 3 2 3 2 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

In [296…

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Perform PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(df)

# Create a new DataFrame for the 2D principal components
df_pca = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])

# Create a scatter plot with colors for clusters
plt.figure(figsize=(10, 6))

colors = ["green", "orange", "blue"]
i=0
for cluster in np.unique(cluster_assignments):
    plt.scatter(df_pca.loc[cluster_assignments == cluster, 'PC1'],
                df_pca.loc[cluster_assignments == cluster, 'PC2'],
                color=colors[i],
                label=f'Cluster {cluster}')
    i+=1

plt.title("2D PCA Plot with hierarchical cluster")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.grid()
plt.show()
```



En este resultado final, donde proyectamos según componentes principales, podemos observar que más o menos se ha realizado un agrupamiento decente, donde tal vez se pueda ver que se solapan unos cuantos registros en los clusters 2 y 3, pero también hay que tener en cuenta que solo estamos visulizando 2 de 4 posibles dimensiones.

5. Agrupamiento K-Means.

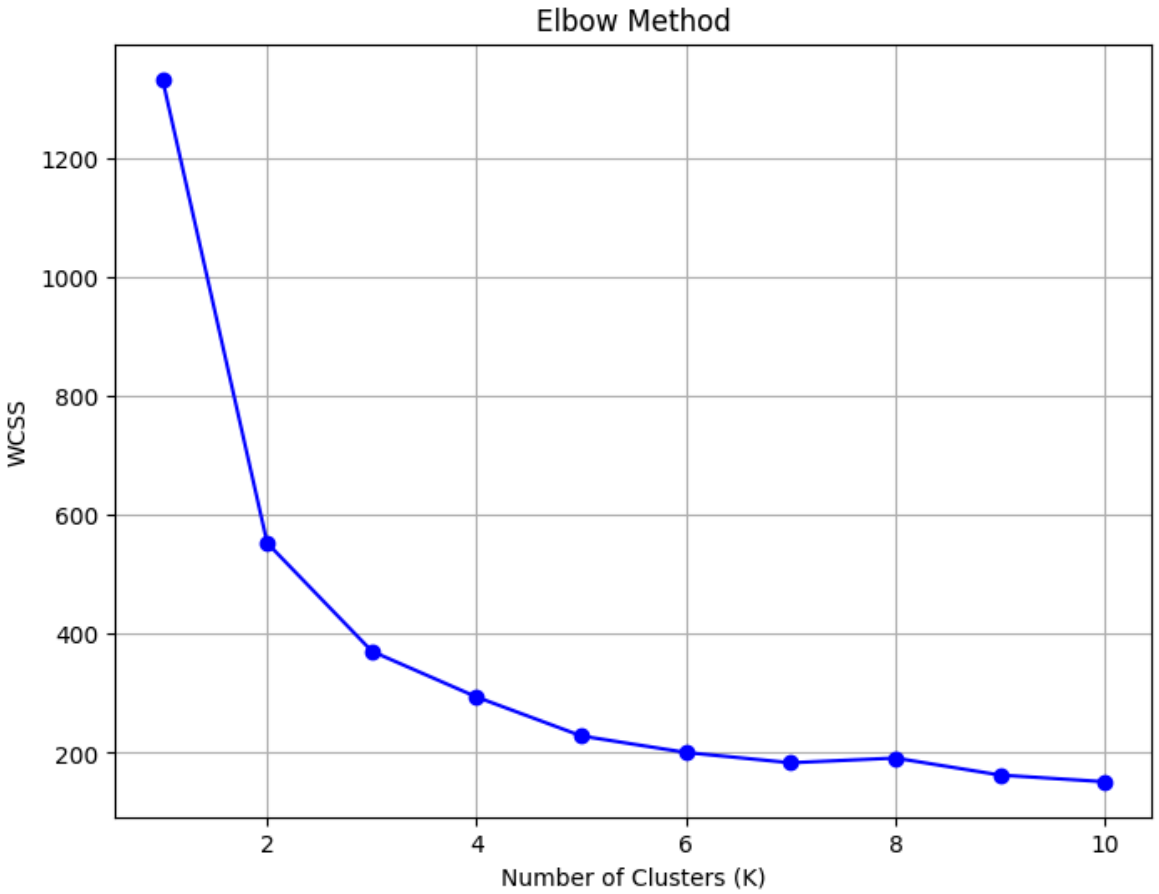
Implementa el algoritmo de agrupamiento k-means en el conjunto de datos. Experimenta con diferentes valores de k y utiliza métricas apropiadas (por ejemplo, método del codo) para determinar el número óptimo de grupos.

Primero probaremos con el metodo del codo para visulaizar el número de K que deberíamos usar.

In [297…

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Create an array to store the WCSS values for different values of K:
wcss = []

for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(df_std)
    wcss.append(kmeans.inertia_)

# Plot the WCSS values to find the elbow point
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='-', color='b')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
```

```
plt.grid(True)
plt.show()
```



Según la tabla anterior podemos ver que es a partir del tercer o el cuarto K donde la inclinación de la curva se estabiliza. Viendo, todo lo que hemos hecho hasta ahora vamos a usar K=3 para comprobar si nuestros calculos anteriores son correctos.

In [298…

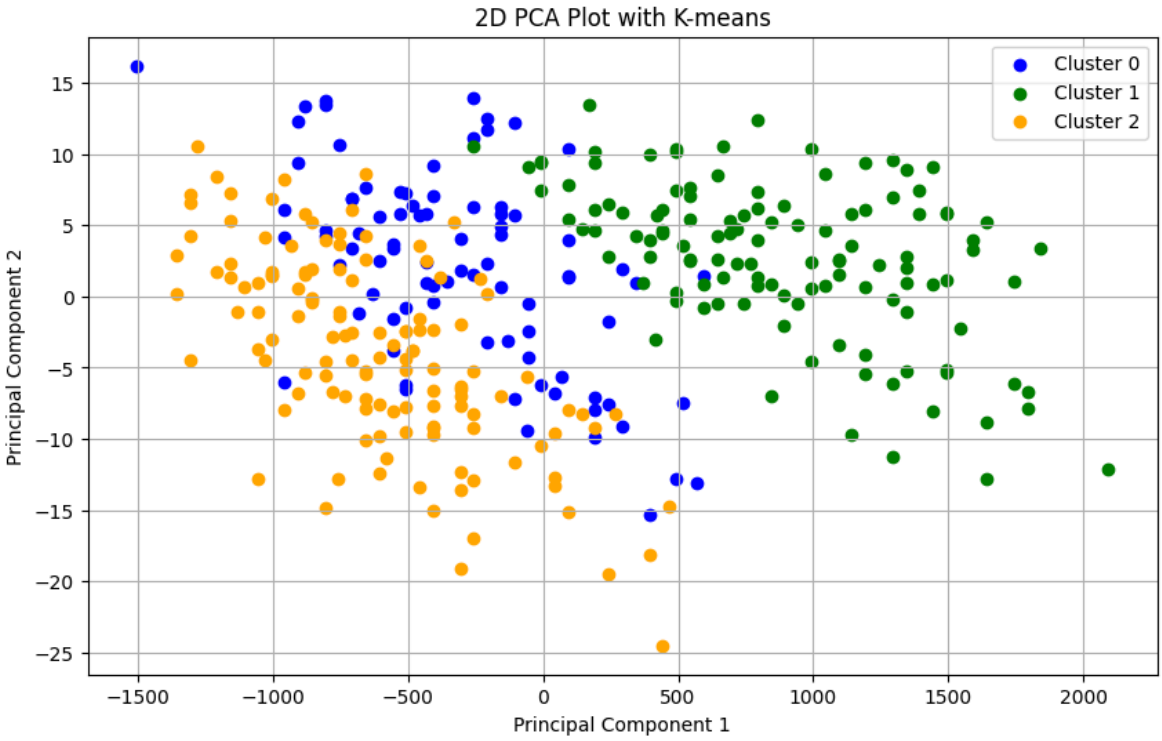
```
from sklearn.cluster import KMeans

# Set the number of clusters (k=4)
k = 3

# Initialize the KMeans model
kmeans = KMeans(n_clusters=k, random_state=0)

# Fit the KMeans model to your standardized data
kmeans.fit(df_std)

# Get the cluster labels for your data
kmeans_cluster_labels = kmeans.labels_
kmeans_cluster_labels
```

Out[298…

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

In [299…

```
# Perform PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(df)

# Create a new DataFrame for the 2D principal components
df_pca = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])

# Create a scatter plot with colors for clusters
plt.figure(figsize=(10, 6))

colors = ["blue", "green", "orange"]
i=0
for cluster in np.unique(kmeans_cluster_labels):
    plt.scatter(df_pca.loc[kmeans_cluster_labels == cluster, 'PC1'],
                df_pca.loc[kmeans_cluster_labels == cluster, 'PC2'],
                color=colors[i],
                label=f'Cluster {cluster}')
    i+=1

plt.title("2D PCA Plot with K-means")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.grid()
plt.show()
```



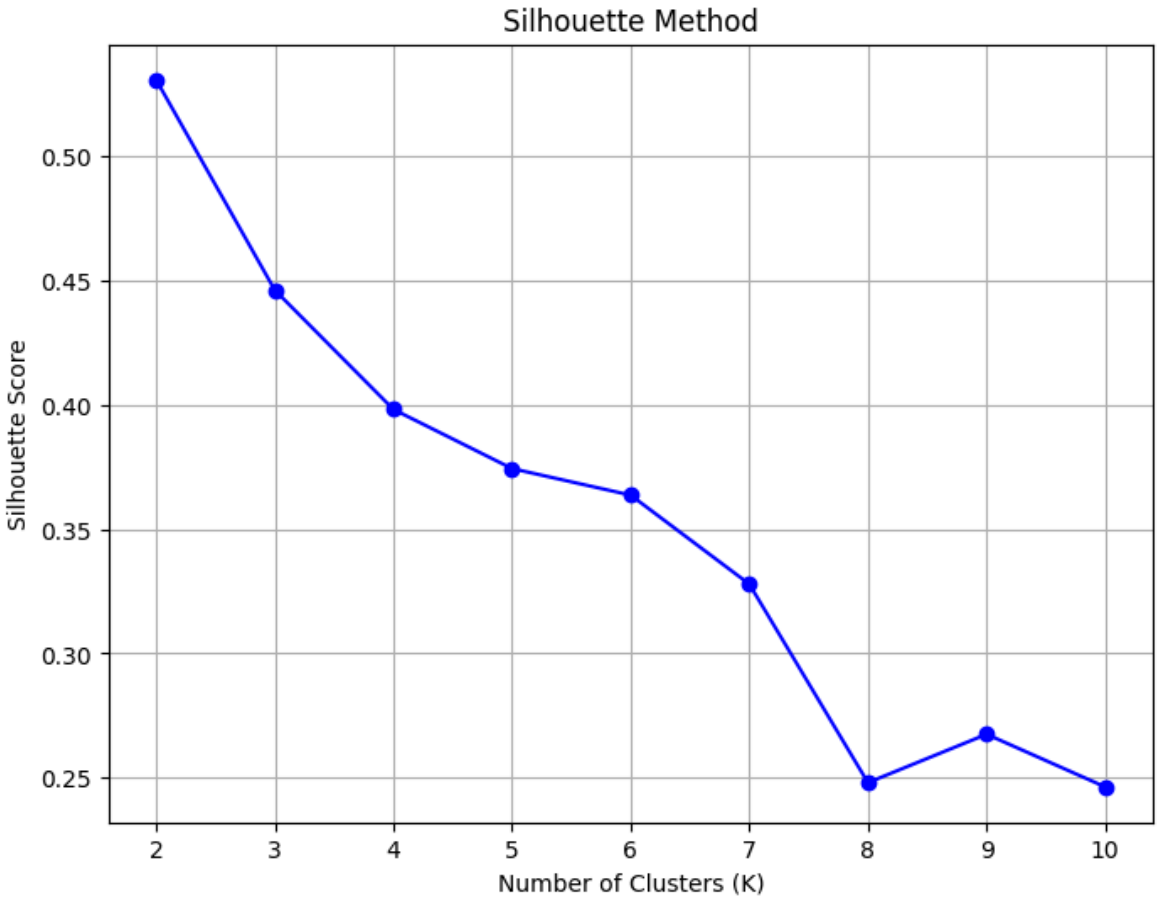
Igual que en el caso del agrupamiento jerarquico, realiza un clustering bastante aceptable, mirandolo desde 2D.

6. Validación del Agrupamiento.

Aplica métricas de validación del agrupamiento (por ejemplo, puntuación de silueta) para evaluar la calidad de los resultados del agrupamiento. Discute la efectividad del algoritmo de agrupamiento en capturar la estructura inherente de los datos tal y como se ha visto en clase.

In [300…

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Create an array to store silhouette scores for different values of K
silhouette_scores = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(df_std)
    labels = kmeans.labels_
    silhouette_avg = silhouette_score(df_std, labels)
    silhouette_scores.append(silhouette_avg)

plt.figure(figsize=(8, 6))
plt.plot(range(2, 11), silhouette_scores, marker='o', linestyle='-', color='b')
plt.title('Silhouette Method')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()
```

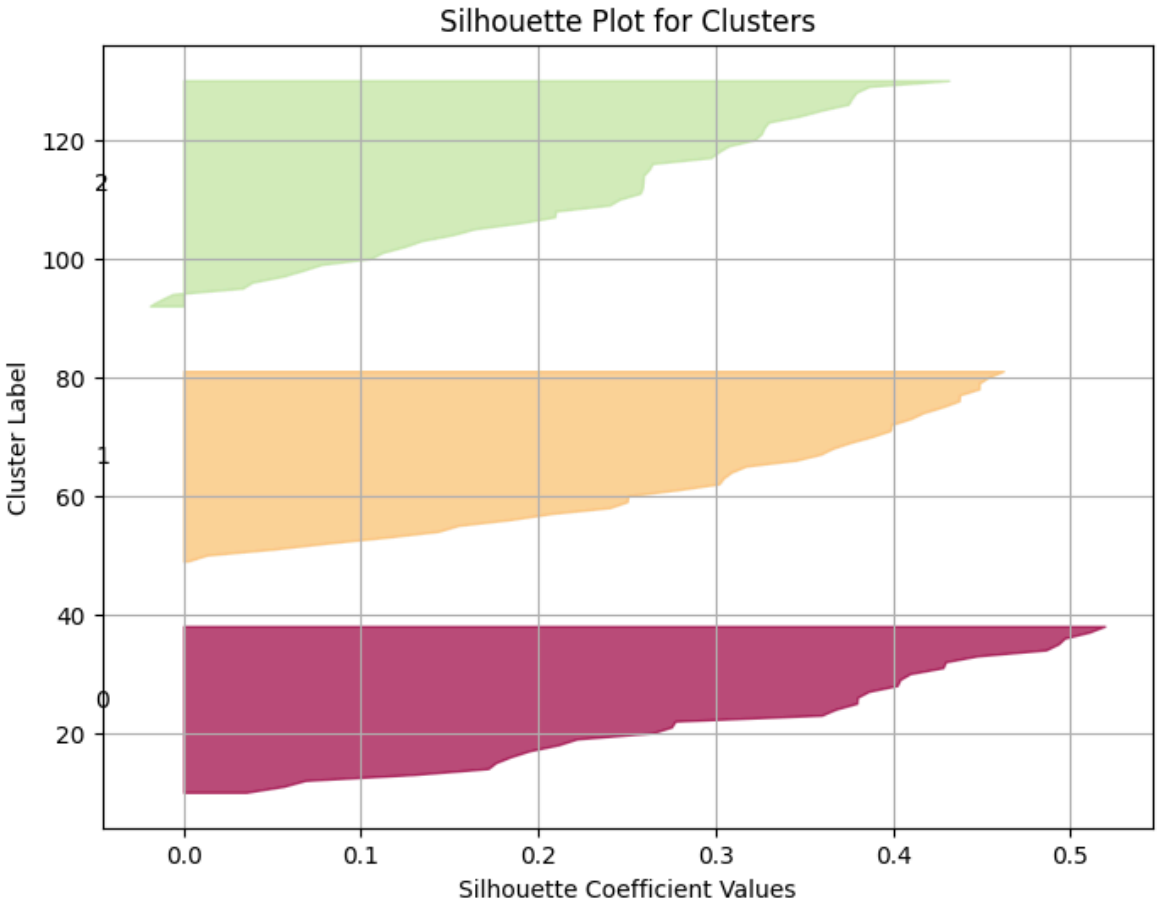


En esta gráfica de silueta la mejor solución posible es coger 2 grupos, ya que es el valor más alto, aunque tal vez al escoger solo 2 clusters se nos quede un agrupamiento demasiado grande, por ello hemos decidio usar el siguiente valor más alto: 3

In [301…

```
from sklearn.metrics import silhouette_samples

# Calculates silhouette scores for each cluster
silhouette_values = silhouette_samples(df_std, labels)

plt.figure(figsize=(8, 6))
y_lower = 10
k = 3
for i in range(k):
    ith_cluster_silhouette_values = silhouette_values[labels == i]
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    color = plt.cm.get_cmap("Spectral")(float(i) / k)
    plt.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)
    plt.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
    y_lower = y_upper + 10

plt.title("Silhouette Plot for Clusters")
plt.xlabel("Silhouette Coefficient Values")
plt.ylabel("Cluster Label")
plt.grid(True)
plt.show()
```

```
C:\Users\Usuario\AppData\Local\Temp\ipykernel_23844\3899455587.py:14: MatplotlibD
eprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and wil
l be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``ma
tplotlib.colormaps.get_cmap(obj)`` instead.
  color = plt.cm.get_cmap("Spectral")(float(i) / k)
```



Tal y como se observa en las puntuaciones de silueta para cada observación, donde mejor está definido el grupo es para el cluster naranja claro que tiendo a pensar que es el cluster 2(verde oscuro en la grafica de ACP) puesto que es el mejor agrupado. Lo deduzco por los siuientes puntos:

- No es el grupo con mas registros.
- Tiene bastantes registros con valores altos
- En la grafica de ACP se pueden observar varios registros bastantes alejados

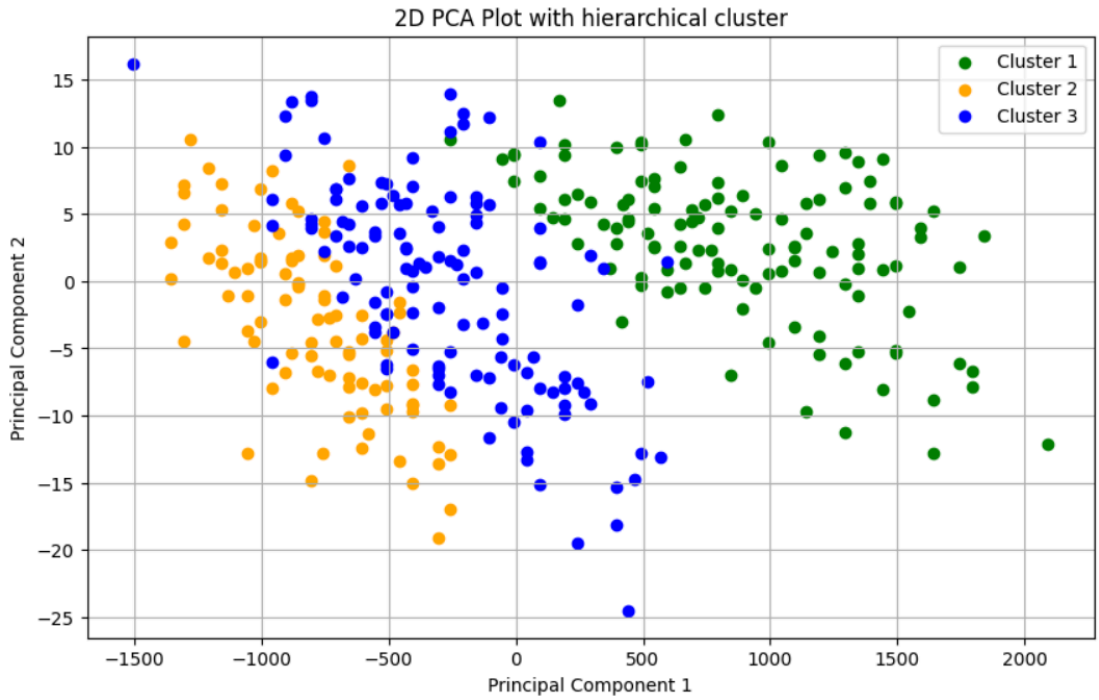
Luego el verde claro será el cluster 0 (naranja), mi pensamiento se basa en el razonamiento anterior, ya que ambas muestras son bastante parecidas. Finalmente el cluster granate será el cluster 1, ya que es el más disperso sin duda.

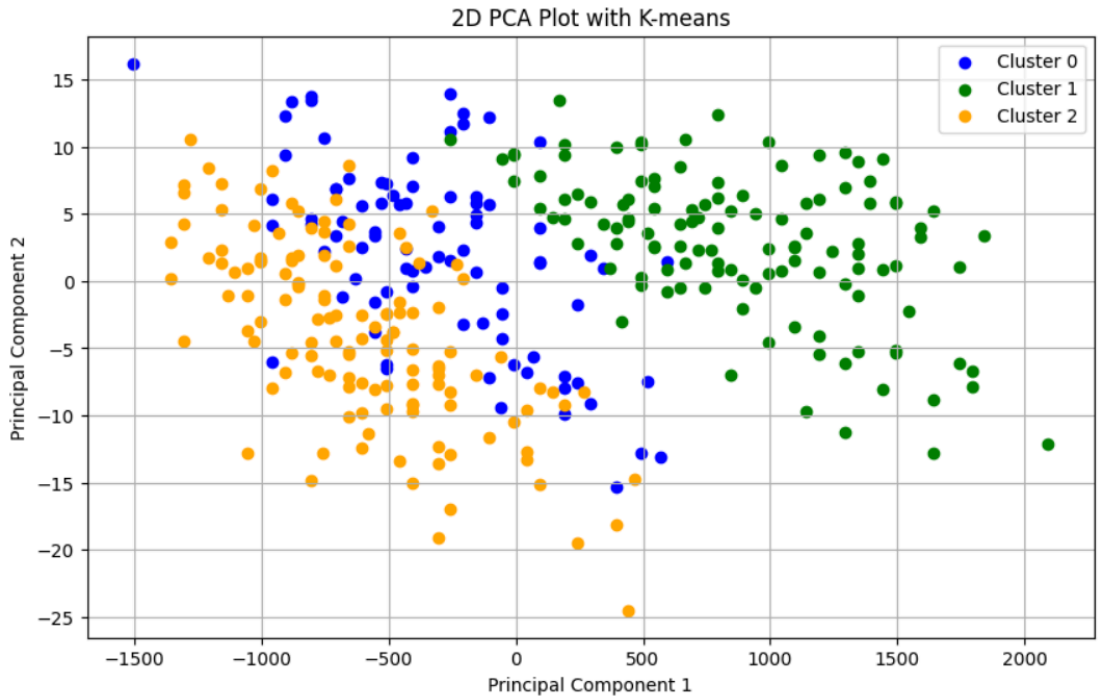
Todo este razonamiento lo podría haber hecho usando los DF, pero ¿Que gracia tendría hacerlo de esa forma? Creo que es más interesante entender las gráficas sin tener la chuletilla de los datos.

7. Jerárquico versus K-Means.

Compara y contrasta los resultados obtenidos del agrupamiento jerárquico y el agrupamiento k-means. Discute similitudes o diferencias en las asignaciones de los

grupos.





Ambos métodos de clustering, agrupamiento jerárquico y K-Means, han demostrado ser efectivos para clasificar la especie Gentoo, logrando una separación clara en la mayoría de los casos. Sin embargo, surgen diferencias notables en la forma en que los algoritmos manejan los otros grupos.

Problemas

- Agrupamiento Jerárquico: Tiende a dar mayor peso al cluster azul, haciéndolo predominante sobre el cluster amarillo. Esto indica que el criterio de unión de los clusters está favoreciendo ciertas características de los datos, lo que puede generar una segmentación desigual.

- K-Means: En este caso, el comportamiento es inverso: el cluster amarillo tiene más relevancia, mientras que el azul pierde fuerza. Esto sugiere que la inicialización de los centroides y la naturaleza de la partición basada en distancia euclidiana pueden estar afectando la distribución de los clusters.

8. Proporciona una interpretación de los grupos.

¿Qué representan los grupos identificados en el contexto de las especies de pingüinos? ¿Existen patrones o tendencias significativas?

Como he estado destacando en los puntos anteriores, y como se puede observar en varias gráficas: Los grupos representados son las especies de los pingüinos de los que tenemos registros (o tal vez la isla de procedencia de estos animales, ya que en cada isla suele predominar una especie).

En relación a las variables podemos afirmar que:

- Gentoo: Mayor tamaño, pico largo y plano, aletas largas, mayor masa corporal.
- Chinstrap: Pico más delgado, tamaño intermedio.
- Adelie: Pico corto y profundo, aletas más cortas, menor masa corporal.

9. Resume tus hallazgos y concluye el análisis. Discute limitaciones o desafíos encontrados durante el proceso de agrupamiento.

El análisis del dataset "penguins" ha permitido identificar patrones claros en las características físicas de las tres especies de pingüinos: Gentoo, Chinstrap y Adelie. A través de distintos métodos de agrupamiento, se ha evidenciado que cada especie presenta diferencias en variables como longitud del pico (bill_length_mm), profundidad del pico (bill_depth_mm), longitud de las aletas (flipper_length_mm) y masa corporal (body_mass_g), las cuales son claves para su clasificación.

Uno de los hallazgos más relevantes es que la especie Gentoo se agrupa de manera clara y consistente en ambos métodos de clustering. Esto se debe a que presenta valores extremos en la mayoría de las variables (picos largos, aletas grandes y mayor peso), lo que facilita su diferenciación. Sin embargo, las especies Chinstrap y Adelie presentan mayor similitud en sus características, lo que dificulta su separación precisa mediante técnicas de clustering.

Al comparar los dos métodos de agrupamiento, encontramos una discrepancia en la distribución de los clusters:

El agrupamiento jerárquico favorece la predominancia del cluster azul. K-Means, en cambio, favorece el cluster amarillo, lo que sugiere que la inicialización de centroides y la segmentación basada en distancias han influido en la separación de los grupos. Este comportamiento sugiere que ciertos factores, como la métrica de distancia utilizada, pueden estar afectando los resultados.

Conclusión Final El dataset "penguins" es ideal para explorar la clasificación y el clustering de especies, pero su correcta segmentación depende de la elección adecuada del preprocesamiento y del método de agrupamiento. Realmente se nota que este dataset se haya creado para realizar pruebas de estudio de aprendizaje no supervisado.

Un enfoque bien optimizado en la elección de métricas, selección de número de clusters puede mejorar significativamente la precisión de los agrupamientos.