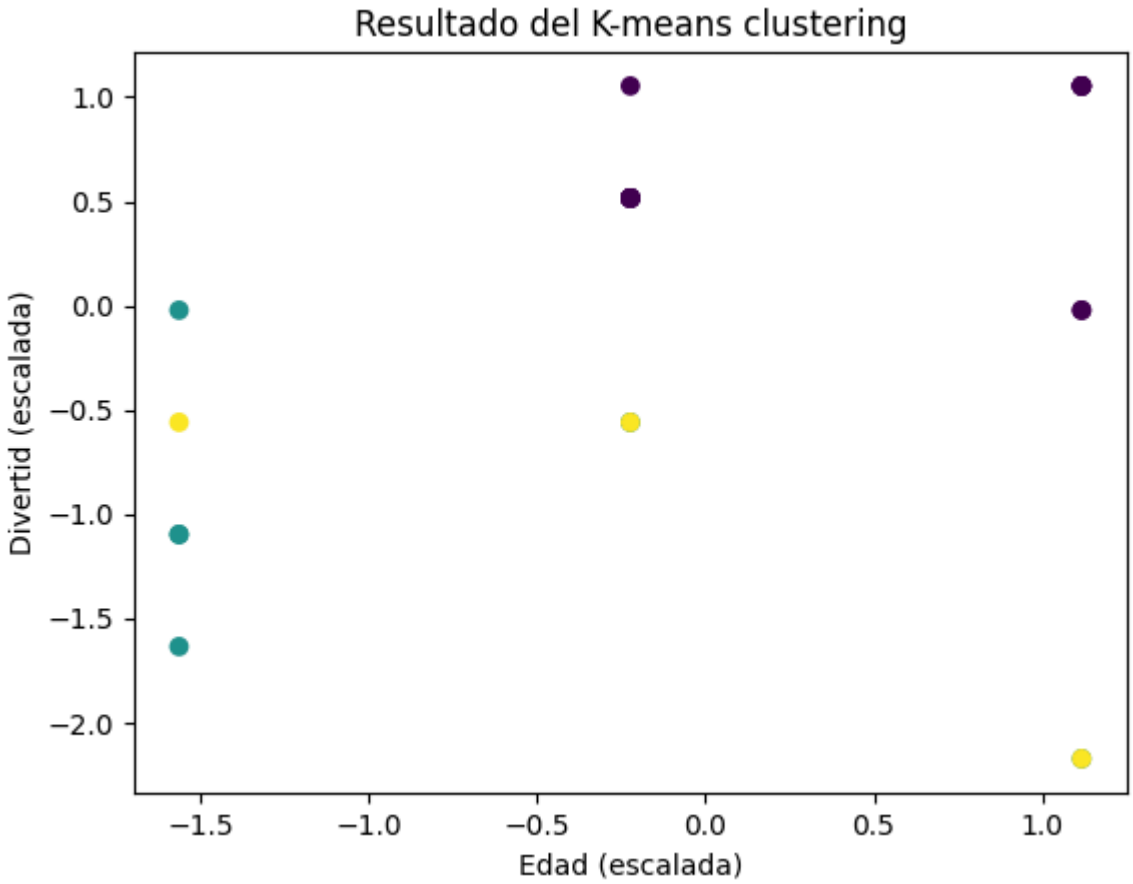
Resumen:

```
# Importar las bibliotecas necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Cargar el archivo CSV (este es tu archivo)
file_path = 'datos.csv'
df = pd.read_csv(file_path)

# Seleccionar las columnas relevantes para el análisis de conglomerados
data = df[['Edad', 'Divertid', 'Pidocomp', 'Aprendom', 'Excur', 'Quitatie', 'Nomeint', 'G

# Estandarización de los datos
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Aplicación del algoritmo K-means
kmeans = KMeans(n_clusters=3, random_state=42)  # Cambia n_clusters según tus necesidades
kmeans.fit(data_scaled)

# Asignación de etiquetas a cada punto de datos
labels = kmeans.labels_

# Agregar las etiquetas al DataFrame original
df['Cluster'] = labels

# Visualización de los conglomerados (solo se usarán las primeras dos variables para graf
plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels, cmap='viridis')
plt.title('Resultado del K-means clustering')
plt.xlabel('Edad (escalada)')
plt.ylabel('Divertid (escalada)')
plt.show()

# Mostrar centroides
centroids = kmeans.cluster_centers_
print('Centroides de los clusters:', centroids)

# Mostrar el DataFrame con los clusters asignados
print(df.head())
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarnin
  super()._check_params_vs_input(X, default_n_init=10)
```




```
Centroides de los clusters: [[ 4.02492236e-01  6.58888155e-01  6.04694625e-01  1.7024
  -6.76123404e-02  3.29292780e-01  4.09196604e-02  4.23390197e-01]
 [-8.94427191e-01 -1.09814693e+00 -1.10614870e+00 -7.73823233e-02
   1.69030851e-01  1.09764260e-01 -8.86592641e-01 -1.92450090e-01]
 [-2.23606798e-01 -1.09814693e+00 -8.11175716e-01 -6.96440909e-01
   1.48029737e-16 -1.86599242e+00  1.56858698e+00 -1.73205081e+00]]
   Caso  Sexo  Edad  Divertid  Pidocomp  Aprendom  Excur  Quitatie  Nomeint  \
0     2     1     9         6         7         3      3         4        2
1     3     1    10         7         6         3      3         4        3
2     4     0     9         6         7         3      4         4        3
3     5     0    10         7         6         4      4         5        3
4     6     1     9         7         6         3      4         3        3

   Gustovis  Unnamed: 10  Unnamed: 11  Cluster
0         1          NaN          NaN        0
1         1          NaN          NaN        0
2         1          NaN          NaN        0
3         1          NaN          NaN        0
4         0          NaN          NaN        0
```

Detalles:..

---

calcular la matriz de proximidades

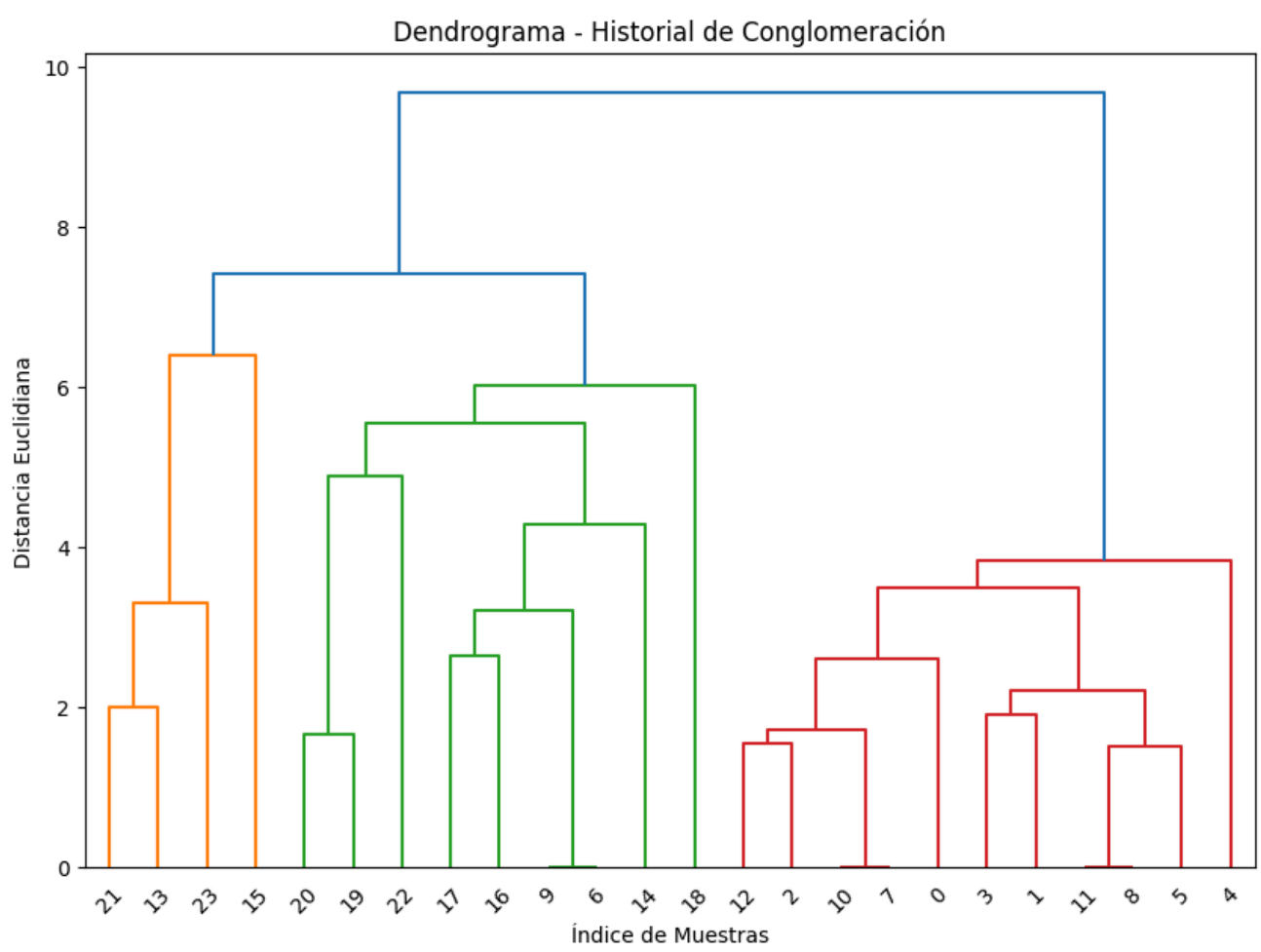
```
# Seleccionar las columnas relevantes para el análisis
data = df[['Edad', 'Divertid', 'Pidocomp', 'Aprendom', 'Excur', 'Quitatie', 'Nomeint', 'G

# Estandarización de los datos
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Calcular la matriz de distancias utilizando la distancia euclidiana
# pdist calcula las distancias por pares entre los puntos
distance_matrix = pdist(data_scaled, metric='euclidean')

# Convertir la matriz de distancias a una forma cuadrada (de proximidades)
proximity_matrix = squareform(distance_matrix)

# Convertir la matriz a un DataFrame para una visualización más clara
proximity_df = pd.DataFrame(proximity_matrix, index=df.index, columns=df.index)

# Mostrar las primeras filas de la matriz de proximidades
print(proximity_df.head())
```

```
          0         1         2         3         4         5         6   \
0  0.000000  2.261948  1.925524  2.956720  3.378351  3.128307  3.104132
1  2.261948  0.000000  1.861695  1.904150  3.145925  1.993445  2.069853
2  1.925524  1.861695  0.000000  2.243781  2.775898  2.465494  2.825799
3  2.956720  1.904150  2.243781  0.000000  3.864630  2.699592  3.373419
4  3.378351  3.145925  2.775898  3.864630  0.000000  2.888537  3.272806

          7         8         9   ...        14        15        16        17  \
0  2.441733  2.837492  3.104132  ...  5.358686  5.660881  3.798425  4.180238
1  2.080429  1.496290  2.069853  ...  4.431171  6.209372  3.444291  3.128307
2  1.501473  2.084160  2.825799  ...  4.194352  5.246393  3.851581  3.538771
3  2.042615  1.443252  3.373419  ...  4.378077  6.256832  3.977628  4.149043
4  2.927095  3.174679  3.272806  ...  4.283452  4.346035  4.496346  3.830197

         18        19        20        21        22        23
0  5.629356  4.079947  4.405619  5.320477  4.802789  6.381444
1  6.264193  4.273135  4.585106  4.666540  4.898556  5.561979
2  5.774093  3.872398  3.962555  4.386536  5.174401  5.806392
3  6.113954  4.859015  5.271268  5.503181  5.255629  6.442139
4  6.034065  4.577830  4.265327  2.899978  4.819339  4.819339

[5 rows x 24 columns]
```

Historial de conglomeración (dendrograma):

```
# Importar las bibliotecas necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
```

```
# Seleccionar las columnas relevantes para el análisis de conglomerados jerárquicos
data = df[['Edad', 'Divertid', 'Pidocomp', 'Aprendom', 'Excur', 'Quitatie', 'Nomeint', 'G

# Estandarización de los datos
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Calcular la matriz de enlaces utilizando el método 'ward' (puedes cambiarlo por otros c
linked = linkage(data_scaled, method='ward')

# Graficar el dendrograma
plt.figure(figsize=(10, 7))
dendrogram(linked,
           orientation='top',
           distance_sort='descending',
           show_leaf_counts=True)
plt.title('Dendrograma - Historial de Conglomeración')
plt.xlabel('Índice de Muestras')
plt.ylabel('Distancia Euclidiana')
plt.show()
```



¿Qué método de conglomeración es mejor?

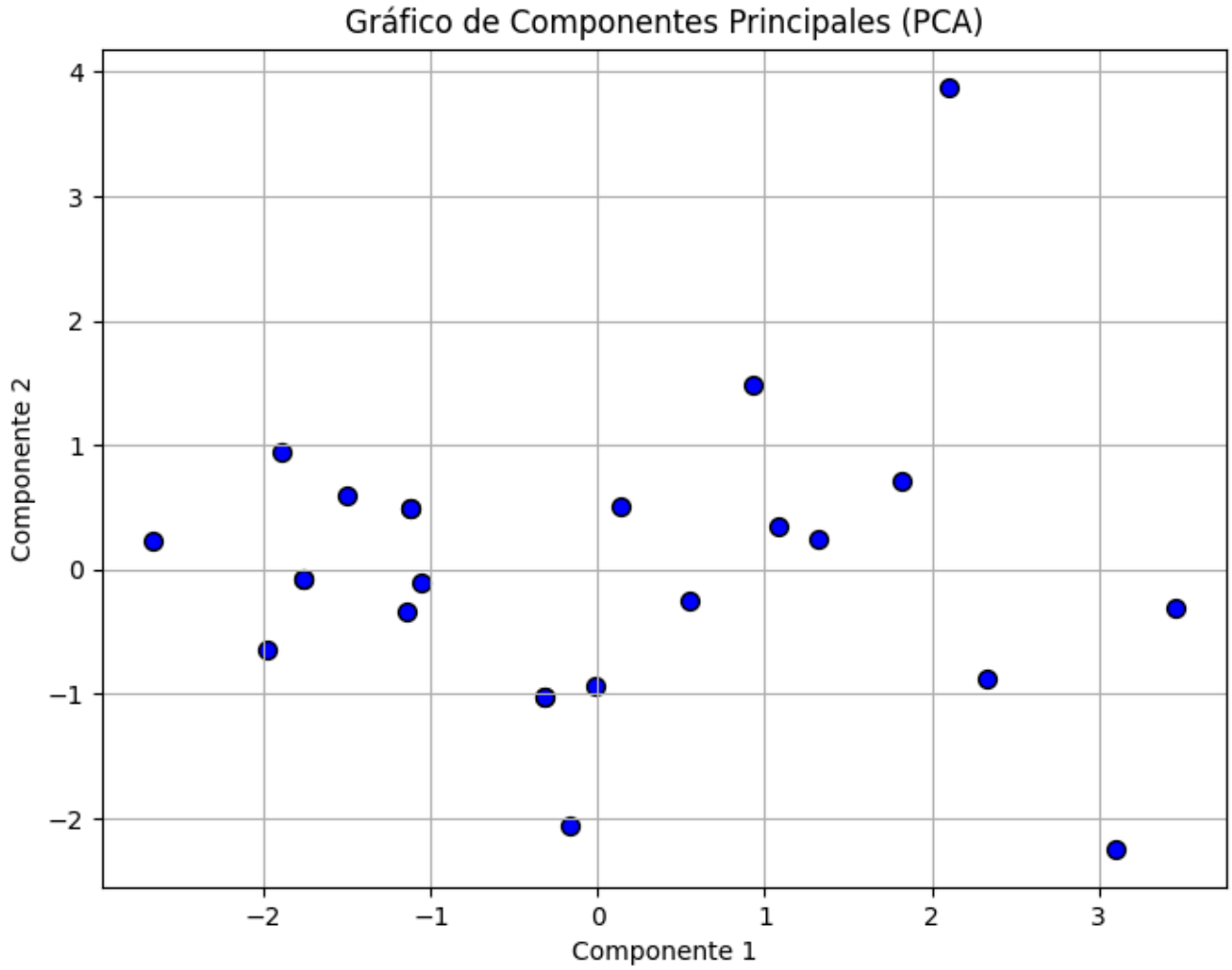
Ward's Method (ward):

Descripción: Minimiza la varianza total dentro de cada conglomerado. Es uno de los métodos más utilizados porque tiende a generar conglomerados más compactos y de tamaño similar.

Ventaja: Es útil cuando los conglomerados tienen forma esférica y cuando se busca minimizar la variabilidad dentro de los conglomerados. Cuándo usarlo: Si quieres que los conglomerados sean lo más compactos y homogéneos posible.

Gráfico de Componentes Principales (PCA)

```
# Importar las bibliotecas necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Seleccionar las columnas relevantes para el análisis
data = df[['Edad', 'Divertid', 'Pidocomp', 'Aprendom', 'Excur', 'Quitatie', 'Nomeint', 'G

# Estandarización de los datos
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Aplicar PCA para reducir a 2 componentes principales
pca = PCA(n_components=2)
pca_result = pca.fit_transform(data_scaled)

# Crear un DataFrame con los resultados de los componentes principales
df_pca = pd.DataFrame(pca_result, columns=['Componente 1', 'Componente 2'])

# Graficar los dos primeros componentes principales
plt.figure(figsize=(8, 6))
plt.scatter(df_pca['Componente 1'], df_pca['Componente 2'], c='blue', edgecolor='k', s=50
plt.title('Gráfico de Componentes Principales (PCA)')
plt.xlabel('Componente 1')
plt.ylabel('Componente 2')
plt.grid(True)
plt.show()

# Opcional: Mostrar la varianza explicada por cada componente
explained_variance = pca.explained_variance_ratio_
print(f'Varianza explicada por el Componente 1: {explained_variance[0]:.2f}')
print(f'Varianza explicada por el Componente 2: {explained_variance[1]:.2f}')
```

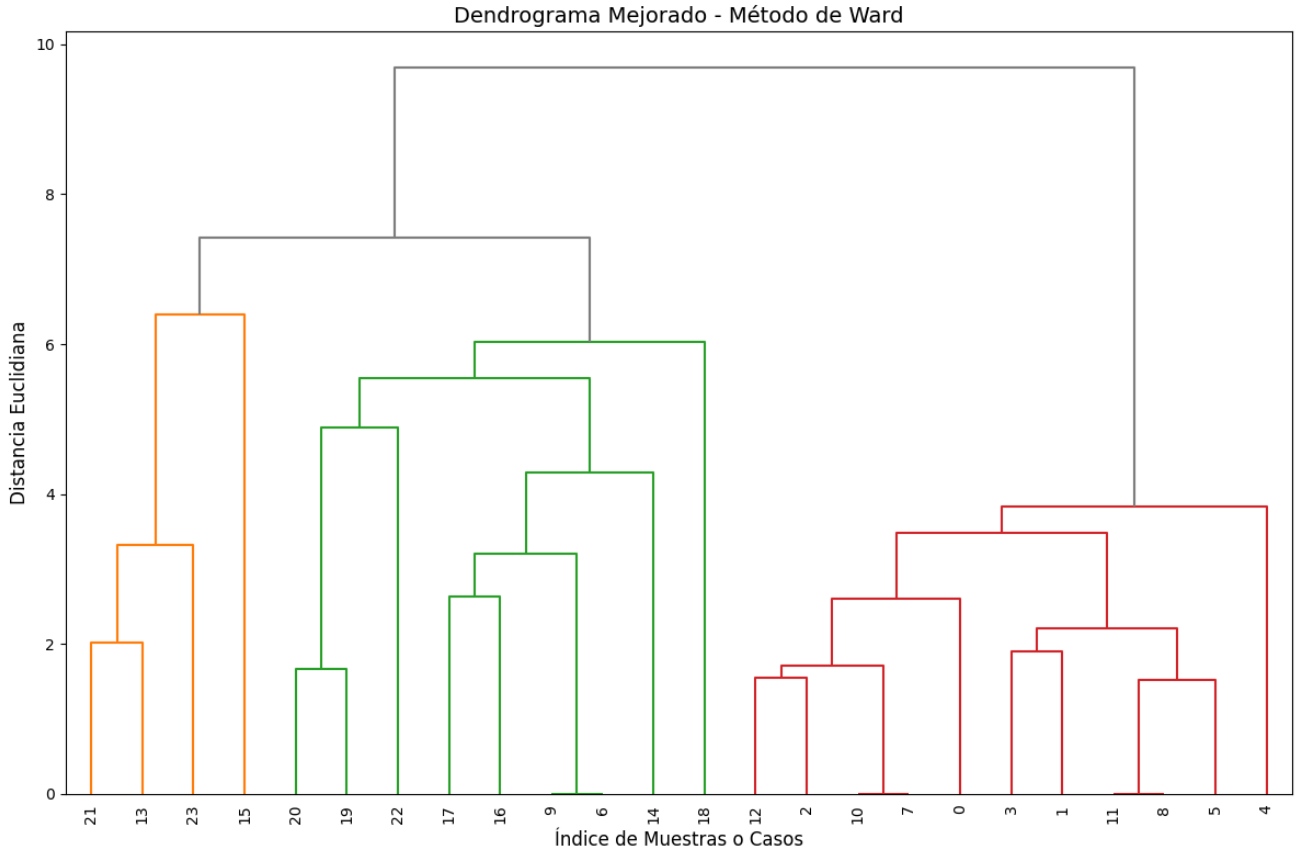



```
Varianza explicada por el Componente 1: 0.35
Varianza explicada por el Componente 2: 0.18
```

Dendrograma Mejorado - Método de Ward

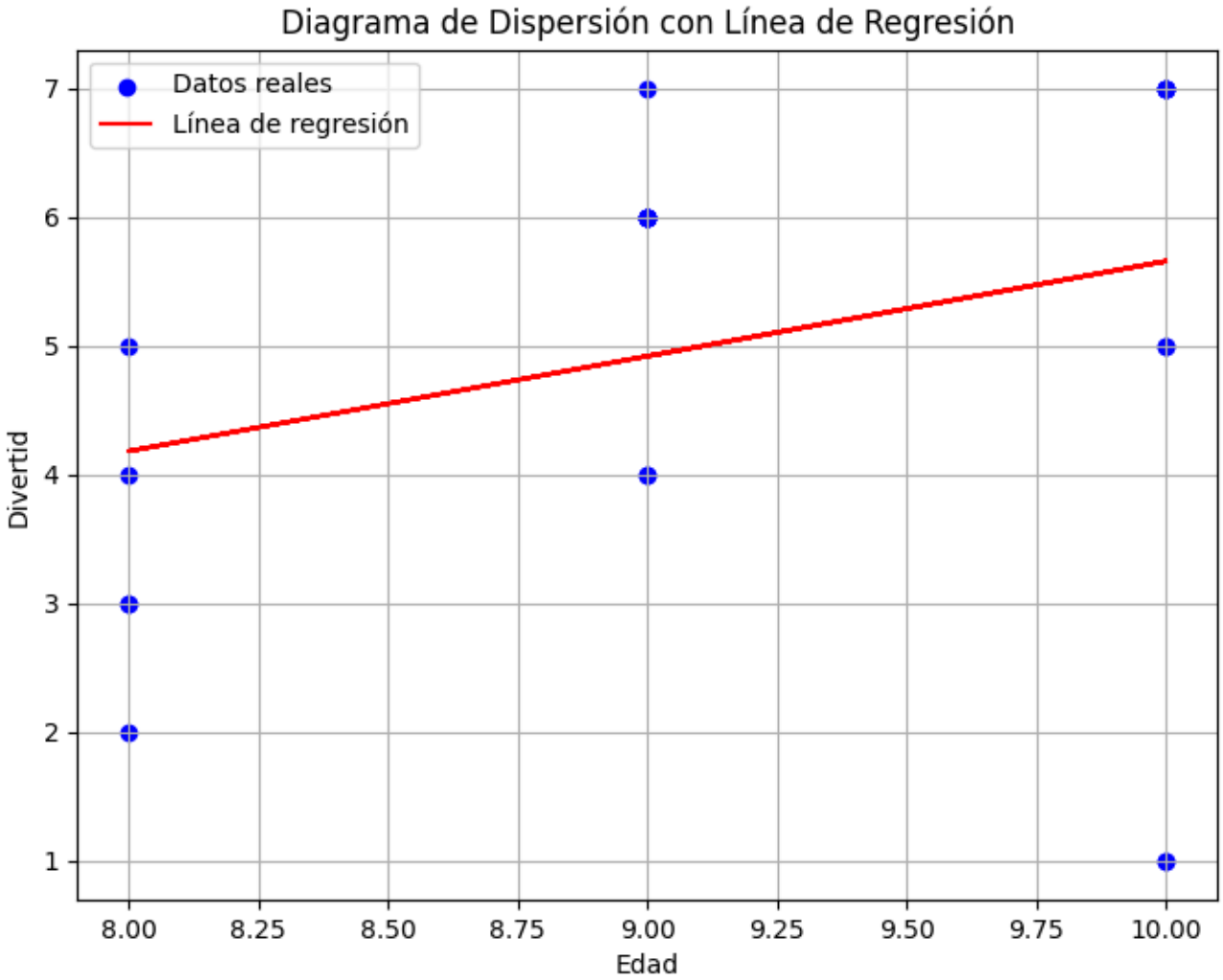
```
# Importar las bibliotecas necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Seleccionar las columnas relevantes para el análisis
data = df[['Edad', 'Divertid', 'Pidocomp', 'Aprendom', 'Excur', 'Quitatie', 'Nomeint', 'G

# Estandarización de los datos
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Aplicación del algoritmo jerárquico aglomerativo con el método de Ward
linked = linkage(data_scaled, method='ward')
```

```
# Tamaño de la figura para mejorar la visualización
plt.figure(figsize=(12, 8))

# Generar el dendrograma con etiquetas de hojas
dendrogram(linked,
           orientation='top',  # Cambia a 'left' o 'right' para otra orientación
           labels=df.index,  # Etiquetas para las hojas
           distance_sort='descending',  # Ordenar por distancia
           show_leaf_counts=True,  # Mostrar el número de puntos en cada conglomerado
           leaf_rotation=90,  # Rotación de etiquetas de las hojas
           leaf_font_size=10,  # Tamaño de la fuente de las etiquetas
           color_threshold=0.7 * max(linked[:, 2]),  # Umbral para colorear clusters
           above_threshold_color='gray'  # Color para las uniones que están por encima de
           )

# Título y etiquetas de los ejes
plt.title('Dendrograma Mejorado - Método de Ward', fontsize=14)
plt.xlabel('Índice de Muestras o Casos', fontsize=12)
plt.ylabel('Distancia Euclidiana', fontsize=12)

# Mostrar el dendrograma
plt.tight_layout()
plt.show()
```




```
# Importar las bibliotecas necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Seleccionar las columnas para el análisis de regresión
# Cambia 'columna_X' y 'columna_Y' por las columnas que quieras analizar
X = df[['Edad']]  # Variable independiente (predictora)
y = df['Divertid']  # Variable dependiente (a predecir)
```

```
# Crear y ajustar el modelo de regresión lineal
model = LinearRegression()
model.fit(X, y)

# Predecir valores de 'y' utilizando el modelo ajustado
y_pred = model.predict(X)

# Crear un diagrama de dispersión
plt.figure(figsize=(8, 6))
plt.scatter(X, y, color='blue', label='Datos reales')
plt.plot(X, y_pred, color='red', label='Línea de regresión')

# Añadir títulos y etiquetas
plt.title('Diagrama de Dispersión con Línea de Regresión')
plt.xlabel('Edad')  # Cambia según tu variable X
plt.ylabel('Divertid')  # Cambia según tu variable Y
plt.legend()

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

# Mostrar la pendiente y la intersección de la recta de regresión
print(f'Pendiente (coeficiente): {model.coef_[0]}')
print(f'Intersección: {model.intercept_}')
```




```
Pendiente (coeficiente): 0.7375000000000003
Intersección: -1.7187500000000018
```

ANOVA

```
# Importar las bibliotecas necesarias
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm


# Realizar ANOVA
# Cambia 'Divertid' por la variable dependiente y 'Sexo' por la variable independiente
model = ols('Divertid ~ C(Sexo)', data=df).fit()
anova_table = anova_lm(model, typ=2)

# Calcular eta cuadrado (tamaño del efecto)
anova_table['eta_sq'] = anova_table['sum_sq'] / sum(anova_table['sum_sq'])

# Calcular eta cuadrado parcial
```

```
anova_table['eta_sq_partial'] = anova_table['sum_sq'] / (anova_table['sum_sq'] + anova_ta

# Mostrar la tabla de ANOVA con el tamaño del efecto
print(anova_table)
```

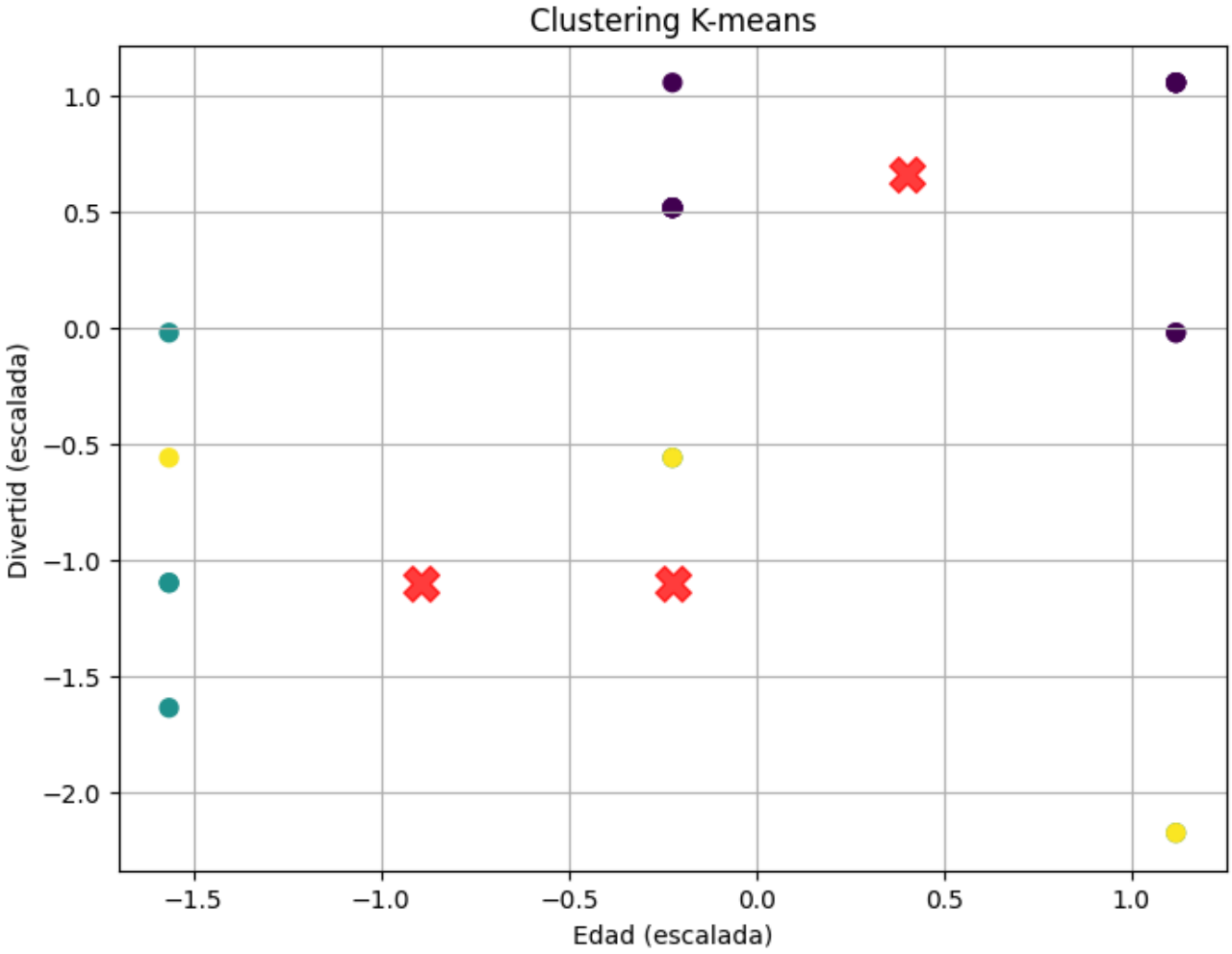
```
                sum_sq    df         F    PR(>F)    eta_sq  eta_sq_partial
C(Sexo)       0.344048   1.0  0.091619  0.764972  0.004147             1.0
Residual     82.614286  22.0       NaN       NaN  0.995853             1.0
```

Clustering K-means

```
# Importar las bibliotecas necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Seleccionar las columnas relevantes para el análisis de K-means
data = df[['Edad', 'Divertid', 'Pidocomp', 'Aprendom', 'Excur', 'Quitatie', 'Nomeint', 'G

# Estandarización de los datos (opcional pero recomendado para K-means)
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Definir el número de clusters (k)
k = 3  # Cambia este valor según tus necesidades

# Aplicar el algoritmo K-means
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(data_scaled)

# Asignación de etiquetas a cada punto de datos
labels = kmeans.labels_

# Agregar las etiquetas al DataFrame original
df['Cluster'] = labels

# Información de los centroides
centroids = kmeans.cluster_centers_

# Visualización de los resultados utilizando las primeras dos características
plt.figure(figsize=(8, 6))
plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels, cmap='viridis', s=50)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, alpha=0.75, marker='X')  #
plt.title('Clustering K-means')
plt.xlabel('Edad (escalada)')
plt.ylabel('Divertid (escalada)')
plt.grid(True)
plt.show()
```

```
# Mostrar el DataFrame con los clusters asignados
print(df.head())
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarnin
  super()._check_params_vs_input(X, default_n_init=10)
```




```
   Caso  Sexo  Edad  Divertid  Pidocomp  Aprendom  Excur  Quitatie  Nomeint  \
0     2     1     9         6         7         3      3         4        2
1     3     1    10         7         6         3      3         4        3
2     4     0     9         6         7         3      4         4        3
3     5     0    10         7         6         4      4         5        3
4     6     1     9         7         6         3      4         3        3

   Gustovis  Unnamed: 10  Unnamed: 11  Cluster
0         1          NaN          NaN        0
1         1          NaN          NaN        0
2         1          NaN          NaN        0
3         1          NaN          NaN        0
4         0          NaN          NaN        0
```