Actividad 5.2 Componentes Principales

Cynthia Cristal Quijas Flores - A01655996

Realizar una regresión lineal múltiple utilizando todas las variables (sin interacciones ni términos de orden superior).

Utilizar gdpp como la variable de respuesta.

Incluir la interpretación de los p-value, VIF, supuestos, residuales, etc...

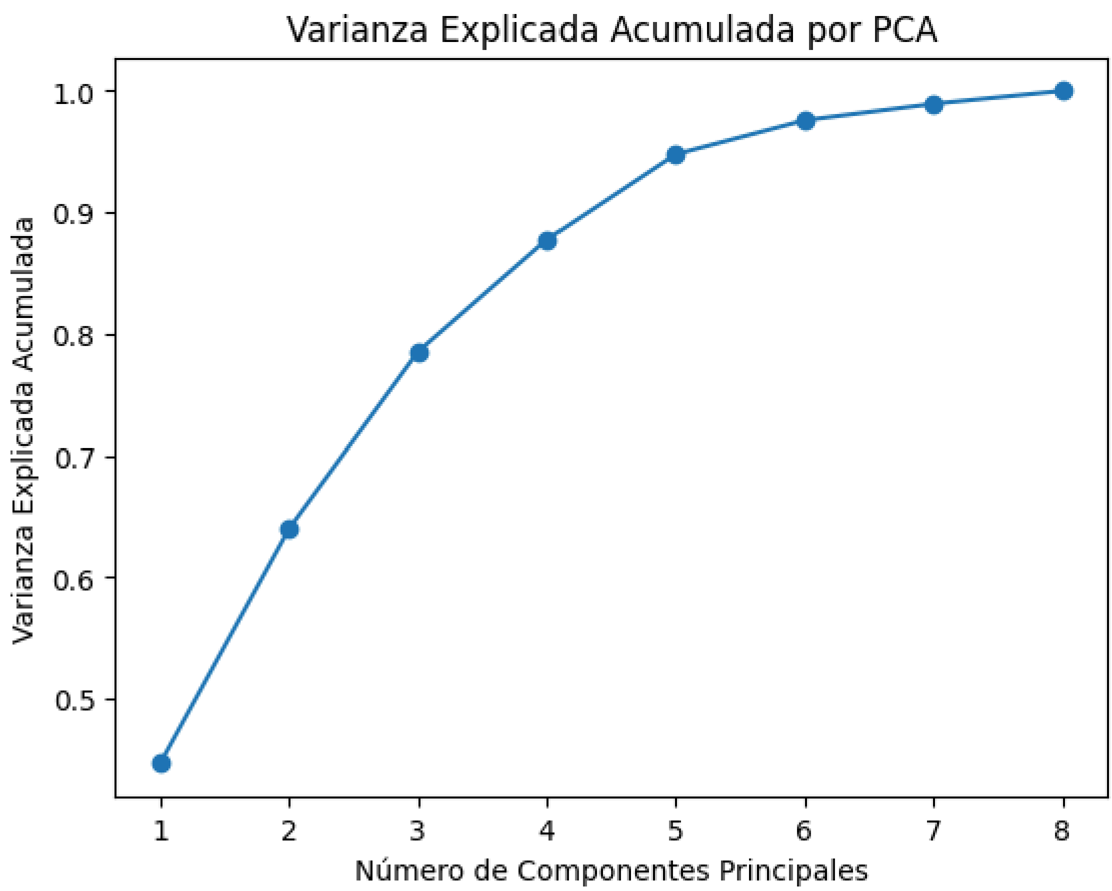
In [7]:
```
import pandas as pd

data = pd.read_csv("Country-data.csv")
data.head(3)
```

Out[7]:

	country	child_mort	exports	health	imports	income	inflation	life_expec	tota
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	

In [8]:
```
from sklearn.preprocessing import StandardScaler

# Selección de variables predictoras y escalado
X = data.drop(columns=['country', 'gdpp'])
y = data['gdpp']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [9]:
```
# PCA

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

# Aplicar PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Varianza explicada acumulada
explained_variance = np.cumsum(pca.explained_variance_ratio_)
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o')
plt.xlabel('Número de Componentes Principales')
plt.ylabel('Varianza Explicada Acumulada')
plt.title('Varianza Explicada Acumulada por PCA')
plt.show()
```



Se puede observar que dependiendo el número de componentes que se elijan es el equivalente al porcentaje de variabilidad que se puede explicar de los datos. En este caso en específico, con cinco componentes principales se cubre más del 90% de la varianza. Esto sugiere que la mayor parte de la información puede ser capturada con los primeros cinco componentes, lo que podría ayudar a reducir la dimensionalidad del conjunto de datos sin perder mucha información. A partir de este componente realmente ya no convendría tomar más componentes porque elevaría la complejidad de algún modelo y ya no hay tanta diferencia de varianza explicada entre el componente cinco y los que siguen.

In [13]:
```
# Regresión lineal múltiple

import statsmodels.api as sm
import numpy as np

# Añadir constante para la regresión
X_with_const = sm.add_constant(X)

# Ajuste del modelo
model = sm.OLS(y, X_with_const).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   gdpp   R-squared:                       0.866
Model:                            OLS   Adj. R-squared:                  0.859
Method:                 Least Squares   F-statistic:                     127.7
Date:                Fri, 25 Oct 2024   Prob (F-statistic):           6.13e-65
Time:                        17:14:42   Log-Likelihood:                -1707.9
No. Observations:                 167   AIC:                             3434.
Df Residuals:                     158   BIC:                             3462.
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -4.193e+04   1.11e+04     -3.768      0.000   -6.39e+04      -2e+04
child_mort    66.5667     35.522      1.874      0.063      -3.593     136.727
exports       28.4864     43.211      0.659      0.511     -56.859     113.831
health      1548.7889    227.184      6.817      0.000    1100.079    1997.499
imports      -28.1190     42.516     -0.661      0.509    -112.093      55.855
income         0.7856      0.044     17.990      0.000       0.699       0.872
inflation   -100.4978     56.673     -1.773      0.078    -212.432      11.437
life_expec   388.9480    142.967      2.721      0.007     106.574     671.322
total_fer    615.0903    679.917      0.905      0.367    -727.809    1957.990
==============================================================================
Omnibus:                       53.684   Durbin-Watson:                   1.914
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              287.333
Skew:                           1.040   Prob(JB):                     4.04e-63
Kurtosis:                       9.080   Cond. No.                     5.39e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.39e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

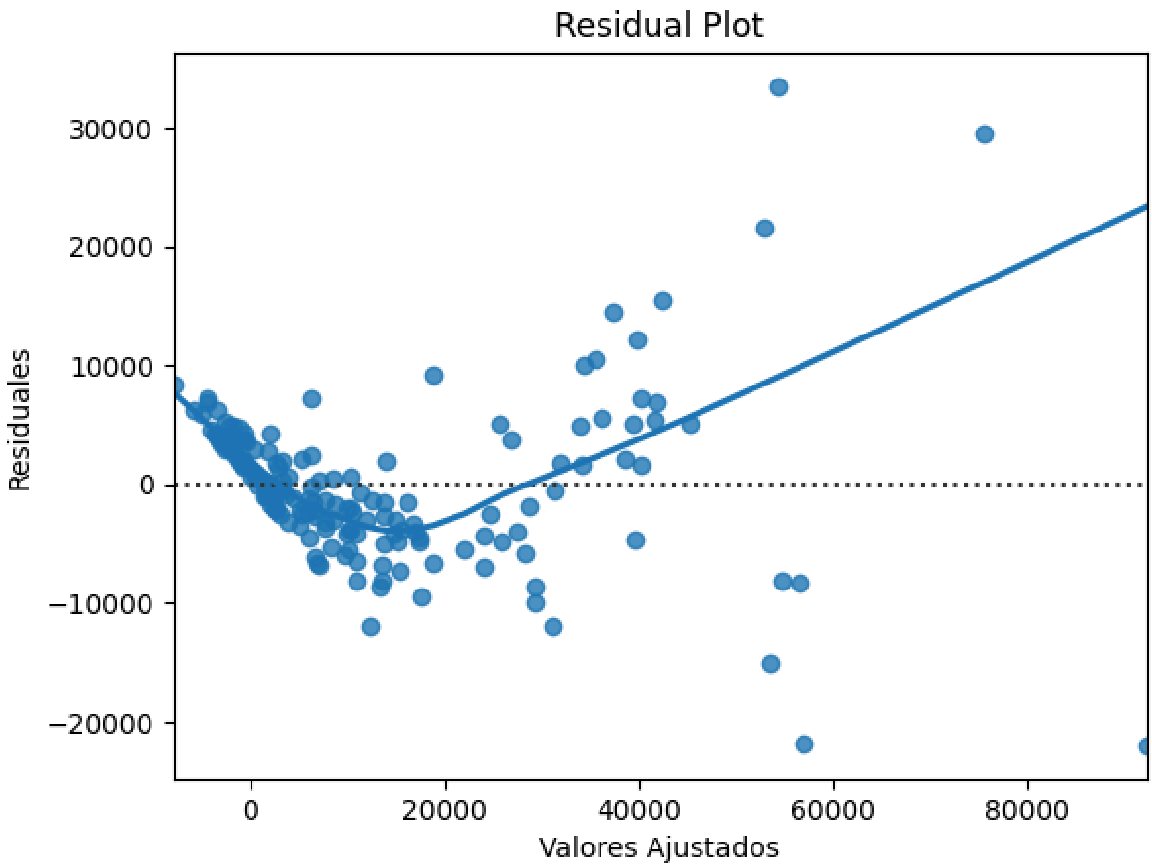
R-cuadrada (R^2): 0.866, lo que indica que el 86.6% de la variabilidad en gdpp (variable de respuesta) es explicada por el conjunto de variables predictoras (child_mort, exports, health, etc.).

Considerando una variable significativa al 95% de nivel de confianza, las variables que entran por tener un valor p > 0.05 son health, income y life expectancy

In [16]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Calcular VIF
vif_data = pd.DataFrame()
vif_data['feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(len(X.c
print(vif_data)

# Graficar residuos para ver homocedasticidad
import seaborn as sns

sns.residplot(x=model.fittedvalues, y=model.resid, lowess=True)
plt.xlabel('Valores Ajustados')
plt.ylabel('Residuales')
```

```
plt.title('Residual Plot')
plt.show()
```

```
     feature        VIF
0  child_mort   8.060881
1     exports  16.008933
2      health   9.832297
3     imports  17.078102
4      income   4.282023
5   inflation   1.942327
6  life_expec  20.033757
7   total_fer  17.652689
```



El VIF mide la multicolinealidad entre las variables independientes. Un VIF superior a 10 suele considerarse alto. life_expec (20.03), total_fer (17.65), imports (17.08) y exports (16.00) tienen valores de VIF altos, lo que sugiere que hay multicolinealidad.

Verificación de supuestos

Los residuos no parecen estar distribuidos aleatoriamente alrededor de 0, lo que sugiere que el modelo no cumple completamente con el supuesto de linealidad. Esto indica que puede haber una relación no lineal entre algunas de las variables predictoras y la variable dependiente (gdpp).

Parece haber una mayor dispersión de los residuos a medida que aumenta el valor ajustado, lo que podría indicar problemas de heterocedasticidad (varianza no constante de los errores).

Aplicar la técnica de componentes principales a los datos para reducir la dimensionalidad de las variables predictoras.

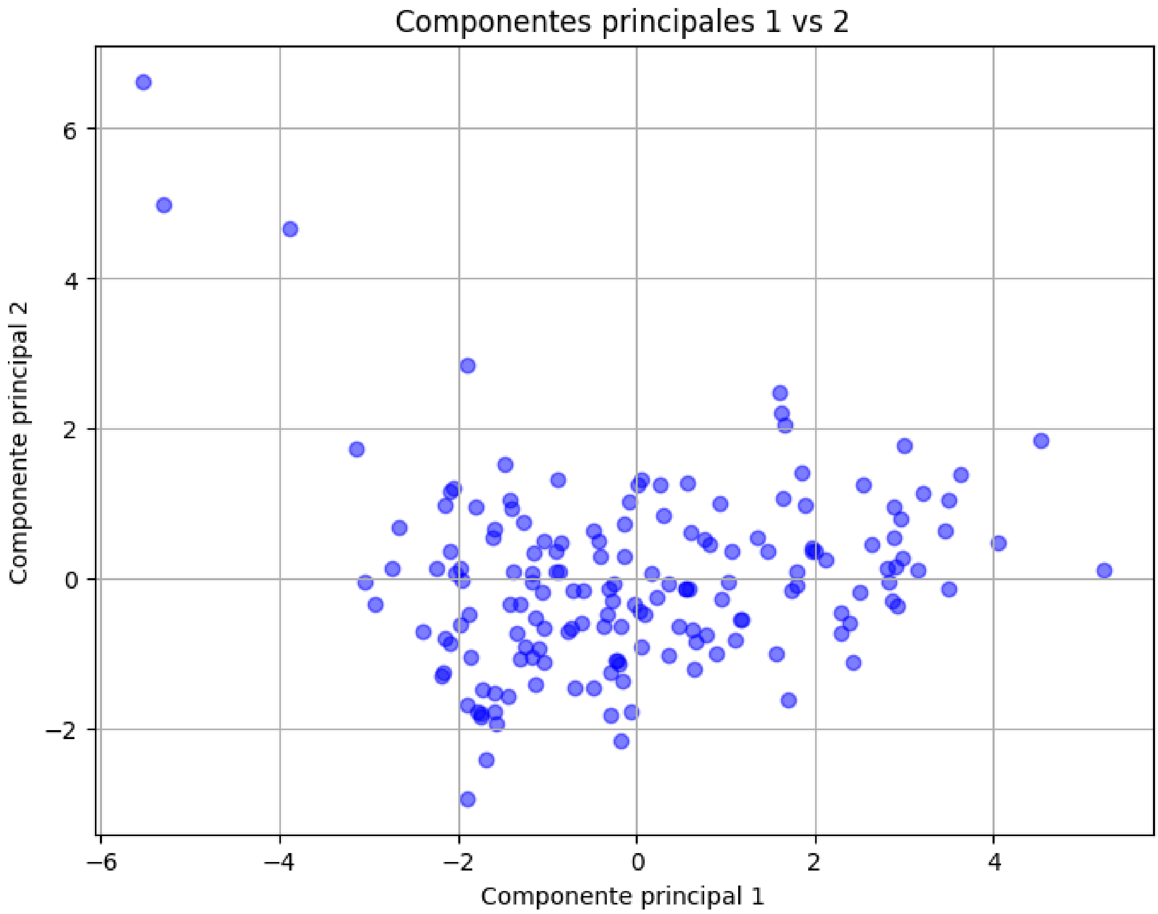
Incluir la explicación del procedimiento y la interpretación de los resultados, valores y vecotres propios, direcciones de los componentes y si existen o no agrupaciones de los datos.

In [23]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Aplicar PCA con cinco componentes
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)

# Varianza explicada por los cinco componentes
varianza_explicada = pca.explained_variance_ratio_
varianza_acumulada = varianza_explicada.cumsum()

# Mostrar la varianza explicada y acumulada
for i, var_exp in enumerate(varianza_explicada, 1):
    print(f'Componente {i}: {var_exp:.4f} de varianza explicada')

print(f'Varianza acumulada por los 5 componentes: {varianza_acumulada[-1]:.4f}')
```

```
Componente 1: 0.4468 de varianza explicada
Componente 2: 0.1930 de varianza explicada
Componente 3: 0.1454 de varianza explicada
Componente 4: 0.0923 de varianza explicada
Componente 5: 0.0703 de varianza explicada
Varianza acumulada por los 5 componentes: 0.9479
```

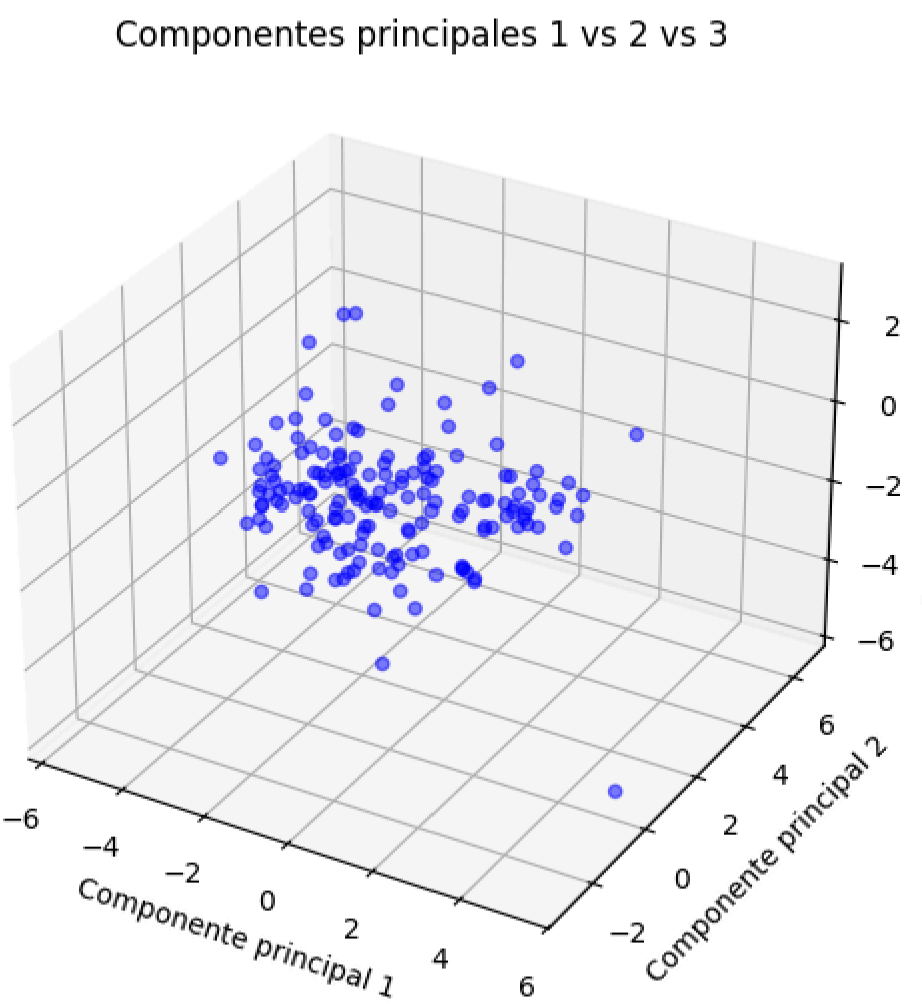
Se elegieron cinco componentes debido a la gráfica "Varianza Explicada Acumulada por PCA", en donde se ve con más detalle cuánta varianza explican los diferentes números de componentes.

In [24]:
```
# Graficar los primeros dos componentes principales
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c='blue', alpha=0.5)
plt.title('Componentes principales 1 vs 2')
plt.xlabel('Componente principal 1')
plt.ylabel('Componente principal 2')
plt.grid(True)
plt.show()
```



In [25]:

```
from mpl_toolkits.mplot3d import Axes3D

# Gráfico 3D con los primeros tres componentes
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c='blue', alpha=0.5)
ax.set_title('Componentes principales 1 vs 2 vs 3')
ax.set_xlabel('Componente principal 1')
ax.set_ylabel('Componente principal 2')
ax.set_zlabel('Componente principal 3')
plt.show()
```



In [26]:
```
# Mostrar los vectores propios (componentes principales)
print("Componentes principales (vectores propios):")
print(pca.components_)
```

```
Componentes principales (vectores propios):
[[ 0.47287988 -0.30839609 -0.14456816 -0.19464001 -0.38678706  0.22047498
  -0.46419134  0.45695156]
 [ 0.21412406  0.60837422 -0.24160817  0.66113128  0.03120652  0.00577075
  -0.23734341  0.17670197]
 [ 0.09998804 -0.14603735  0.64740271  0.28525732 -0.24777586 -0.6157768
  -0.15808191  0.05108475]
 [ 0.11518655  0.10150821  0.68015594  0.05636071  0.31502867  0.62129198
   0.00385699  0.15930427]
 [ 0.29716975  0.05751086 -0.05895877 -0.31536808  0.72825615 -0.41786462
  -0.09136627  0.30353554]]
```

Con cinco componentes principales, estaremos reteniendo la mayoría de la varianza del conjunto de datos, permitiendo reducir la dimensionalidad sin perder demasiada información.

Obtener las ecuaciones de Transformación Lineal de cada componente en función de las variables más importantes.

In [27]:
```
# Definir los nombres de las variables predictoras originales
variables = ['child_mort', 'exports', 'health', 'imports', 'income', 'inflation'
```

```
# Obtener los vectores propios de cada componente
componentes = pca.components_

# Mostrar las ecuaciones de transformación lineal para cada componente
for i in range(5):  # Para los primeros 5 componentes
    print(f"Componente {i + 1}:")
    ecuacion = " + ".join([f"{componentes[i, j]:.4f} * {variables[j]}" for j in
    print(f"Componente {i + 1} = {ecuacion}\n")
```

```
Componente 1:
Componente 1 = 0.4729 * child_mort + -0.3084 * exports + -0.1446 * health + -0.19
46 * imports + -0.3868 * income + 0.2205 * inflation + -0.4642 * life_expec + 0.4
570 * total_fer

Componente 2:
Componente 2 = 0.2141 * child_mort + 0.6084 * exports + -0.2416 * health + 0.6611
* imports + 0.0312 * income + 0.0058 * inflation + -0.2373 * life_expec + 0.1767
* total_fer

Componente 3:
Componente 3 = 0.1000 * child_mort + -0.1460 * exports + 0.6474 * health + 0.2853
* imports + -0.2478 * income + -0.6158 * inflation + -0.1581 * life_expec + 0.051
1 * total_fer

Componente 4:
Componente 4 = 0.1152 * child_mort + 0.1015 * exports + 0.6802 * health + 0.0564
* imports + 0.3150 * income + 0.6213 * inflation + 0.0039 * life_expec + 0.1593 *
total_fer

Componente 5:
Componente 5 = 0.2972 * child_mort + 0.0575 * exports + -0.0590 * health + -0.315
4 * imports + 0.7283 * income + -0.4179 * inflation + -0.0914 * life_expec + 0.30
35 * total_fer
```

Dependiendo del signo que tiene el coeficiente que acompaña a cada variable se indica si la misma tiene un peso negativo o positivo.

Dar un nombre a cada componente principal con base en las variables que lo conforman.

In [29]:
```
# Importante: Los nombres de las variables originales
variables = ['child_mort', 'exports', 'health', 'imports', 'income', 'inflation'

# Mostrar las contribuciones de cada variable a cada componente
for i in range(5):  # Para los primeros 5 componentes
    print(f"Componente {i + 1}:")
    for j in range(len(variables)):
        print(f"{variables[j]}: {componentes[i, j]:.4f}")
    print("\n")
```

```
Componente 1:
child_mort: 0.4729
exports: -0.3084
health: -0.1446
imports: -0.1946
income: -0.3868
inflation: 0.2205
life_expec: -0.4642
total_fer: 0.4570


Componente 2:
child_mort: 0.2141
exports: 0.6084
health: -0.2416
imports: 0.6611
income: 0.0312
inflation: 0.0058
life_expec: -0.2373
total_fer: 0.1767


Componente 3:
child_mort: 0.1000
exports: -0.1460
health: 0.6474
imports: 0.2853
income: -0.2478
inflation: -0.6158
life_expec: -0.1581
total_fer: 0.0511


Componente 4:
child_mort: 0.1152
exports: 0.1015
health: 0.6802
imports: 0.0564
income: 0.3150
inflation: 0.6213
life_expec: 0.0039
total_fer: 0.1593


Componente 5:
child_mort: 0.2972
exports: 0.0575
health: -0.0590
imports: -0.3154
income: 0.7283
inflation: -0.4179
life_expec: -0.0914
total_fer: 0.3035
```

Una manera en la que se pueden asignar nombres a los componentes sería de acuerdo con los pesos que tienen las variables predictoras, poniendo los dos más altos por ejemplo.

Realizar nuevamente la regresión con los componentes principales seleccionados.

Reliazar la interpretación adecuada.

In [30]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Realizar regresión lineal utilizando los cinco componentes principales
modelo = LinearRegression()
modelo.fit(X_pca, y)  # 'y' es la variable dependiente

# Predicciones
y_pred = modelo.predict(X_pca)

# Métricas de rendimiento
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# Resultados
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R-squared (R2): {r2:.4f}")
```

```
Mean Squared Error (MSE): 46412465.7068
R-squared (R2): 0.8610
```

In [31]:
```
# Imprimir los coeficientes y el intercepto del modelo
print("Intercepto del modelo:", modelo.intercept_)
for i, coef in enumerate(modelo.coef_, 1):
    print(f"Coeficiente del Componente {i}: {coef:.4f}")
```

```
Intercepto del modelo: 12964.155688622754
Coeficiente del Componente 1: -6705.6797
Coeficiente del Componente 2: -616.1469
Coeficiente del Componente 3: -880.3942
Coeficiente del Componente 4: 7493.7030
Coeficiente del Componente 5: 12209.2686
```

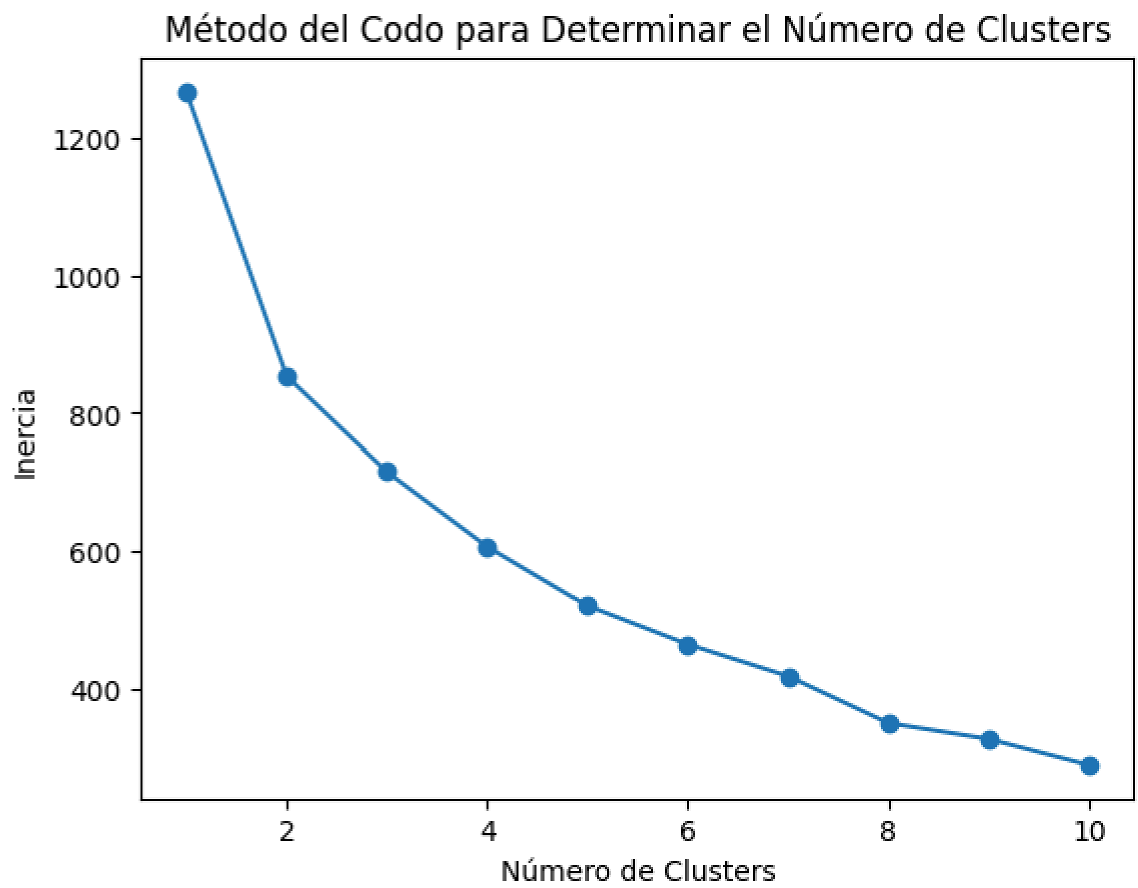
Comparando la R^2, en el modelo que se obtuvo sin aplicar la técnica de los componentes principales, el resultado fue de 0.866, en cambio, cuando se aplicó la técnica y se realizó un análisis del número de componentes que se iban a considerar, la métrica bajó unas centésimas, con un resultado de 0.861. Tal vez sería conveniente en futuros desarrollos volver a repetir el proceso de PCA, pero con seis componentes principales para ver si los resultados mejoran un poco. No obstante, ambos modelos muestran resultados favorables para el caso de estudio.

Realizar un análisis de conglomerados (clusters) utilizando los componentes principales y presentar una visualización de los países en cada uno de los grupos

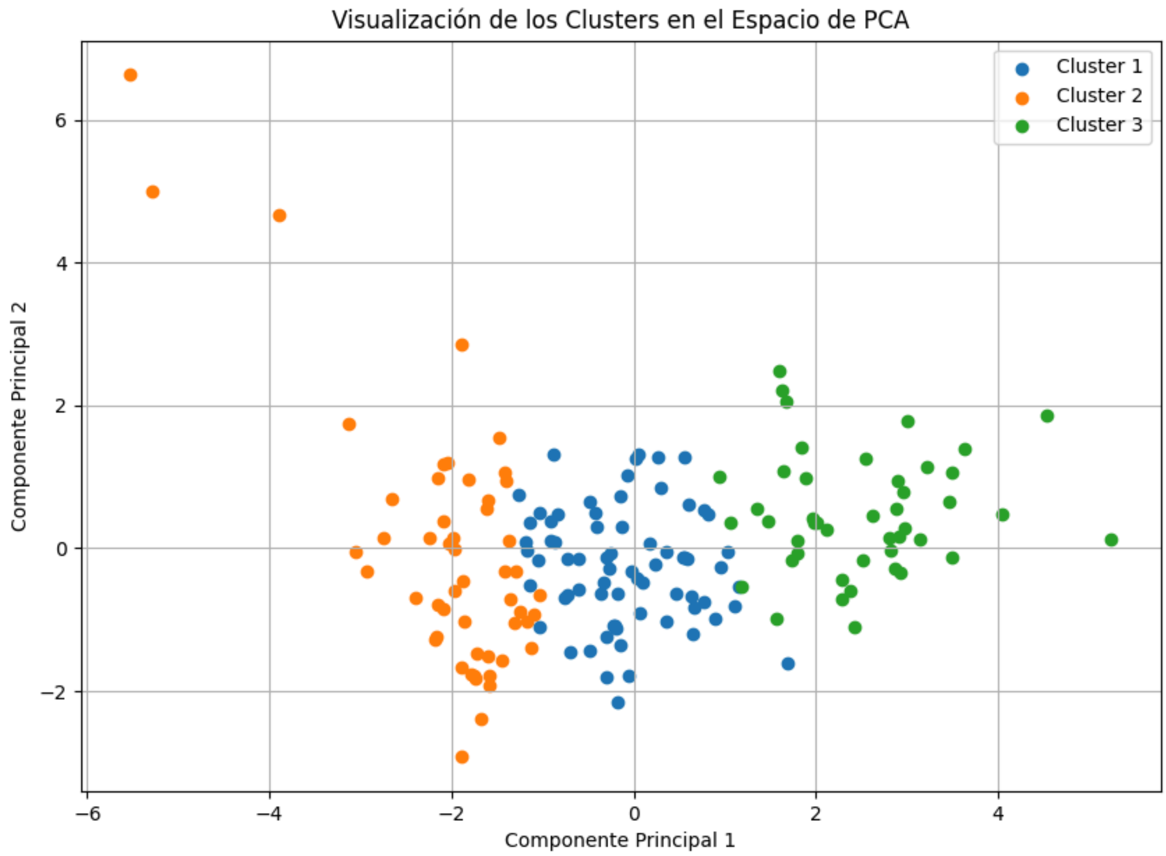
In [32]:
```
from sklearn.cluster import KMeans
```

```python
# Determinar el número óptimo de clusters utilizando el método del codo
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X_pca)
    inertia.append(kmeans.inertia_)

# Graficar el método del codo
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Número de Clusters')
plt.ylabel('Inercia')
plt.title('Método del Codo para Determinar el Número de Clusters')
plt.show()

# Elegir el número de clusters (ejemplo: 3)
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
clusters = kmeans.fit_predict(X_pca)

# Añadir la información del cluster al DataFrame original
data['Cluster'] = clusters

# Visualización de los clusters en el espacio de los primeros dos componentes pr
plt.figure(figsize=(10, 7))
for i in range(n_clusters):
    plt.scatter(X_pca[clusters == i, 0], X_pca[clusters == i, 1], label=f'Cluste

plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Visualización de los Clusters en el Espacio de PCA')
plt.legend()
plt.grid()
plt.show()

# Visualizar los países en cada cluster
for i in range(n_clusters):
    print(f"\nCluster {i + 1}:")
    print(data[data['Cluster'] == i]['country'].values)
```

```
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\joblib\externals\loky\backen
d\context.py:136: UserWarning: Could not find the number of physical cores for th
e following reason:
found 0 physical cores < 1
Returning the number of logical cores instead. You can silence this warning by se
tting LOKY_MAX_CPU_COUNT to the number of cores you want to use.
  warnings.warn(
  File "c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\joblib\externals\lok
y\backend\context.py", line 282, in _count_physical_cores
    raise ValueError(f"found {cpu_count_physical} physical cores < 1")
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
```




```
c:\Users\crist\miniconda3\envs\SAM\Lib\site-packages\sklearn\cluster\_kmeans.py:1
429: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



```
Cluster 1:
['Albania' 'Algeria' 'Antigua and Barbuda' 'Argentina' 'Armenia'
 'Azerbaijan' 'Bangladesh' 'Barbados' 'Belarus' 'Belize' 'Bhutan'
 'Bolivia' 'Botswana' 'Brazil' 'Bulgaria' 'Cambodia' 'Cape Verde' 'Chile'
 'China' 'Colombia' 'Dominican Republic' 'Ecuador' 'Egypt' 'El Salvador'
 'Fiji' 'Georgia' 'Grenada' 'Guatemala' 'Guyana' 'India' 'Indonesia'
 'Iran' 'Iraq' 'Jamaica' 'Jordan' 'Kazakhstan' 'Kyrgyz Republic' 'Libya'
 'Macedonia, FYR' 'Mauritius' 'Micronesia, Fed. Sts.' 'Moldova' 'Mongolia'
 'Morocco' 'Myanmar' 'Nepal' 'Oman' 'Paraguay' 'Peru' 'Philippines'
 'Romania' 'Russia' 'Samoa' 'Saudi Arabia' 'Solomon Islands' 'Sri Lanka'
 'St. Vincent and the Grenadines' 'Suriname' 'Tajikistan' 'Thailand'
 'Tonga' 'Tunisia' 'Turkey' 'Turkmenistan' 'Ukraine' 'Uruguay'
 'Uzbekistan' 'Vanuatu' 'Venezuela' 'Vietnam']

Cluster 2:
['Australia' 'Austria' 'Bahamas' 'Bahrain' 'Belgium'
 'Bosnia and Herzegovina' 'Brunei' 'Canada' 'Costa Rica' 'Croatia'
 'Cyprus' 'Czech Republic' 'Denmark' 'Estonia' 'Finland' 'France'
 'Germany' 'Greece' 'Hungary' 'Iceland' 'Ireland' 'Israel' 'Italy' 'Japan'
 'Kuwait' 'Latvia' 'Lebanon' 'Lithuania' 'Luxembourg' 'Malaysia'
 'Maldives' 'Malta' 'Montenegro' 'Netherlands' 'New Zealand' 'Norway'
 'Panama' 'Poland' 'Portugal' 'Qatar' 'Serbia' 'Seychelles' 'Singapore'
 'Slovak Republic' 'Slovenia' 'South Korea' 'Spain' 'Sweden' 'Switzerland'
 'United Arab Emirates' 'United Kingdom' 'United States']

Cluster 3:
['Afghanistan' 'Angola' 'Benin' 'Burkina Faso' 'Burundi' 'Cameroon'
 'Central African Republic' 'Chad' 'Comoros' 'Congo, Dem. Rep.'
 'Congo, Rep.' "Cote d'Ivoire" 'Equatorial Guinea' 'Eritrea' 'Gabon'
 'Gambia' 'Ghana' 'Guinea' 'Guinea-Bissau' 'Haiti' 'Kenya' 'Kiribati'
 'Lao' 'Lesotho' 'Liberia' 'Madagascar' 'Malawi' 'Mali' 'Mauritania'
 'Mozambique' 'Namibia' 'Niger' 'Nigeria' 'Pakistan' 'Rwanda' 'Senegal'
 'Sierra Leone' 'South Africa' 'Sudan' 'Tanzania' 'Timor-Leste' 'Togo'
 'Uganda' 'Yemen' 'Zambia']
```