```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as sch
from scipy.spatial.distance import pdist, squareform
from sklearn.decomposition import PCA
```

```
# Configuración para visualizaciones
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("viridis")
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 12
```

```
# prompt: carga de datos desde google drive

from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Specify the file path in your Google Drive
file_path = '/content/drive/MyDrive/BBDD_ParaInciar_MakDigital (1).xlsx'  # Replace 'your_file.csv' with the actual file name and pa
```

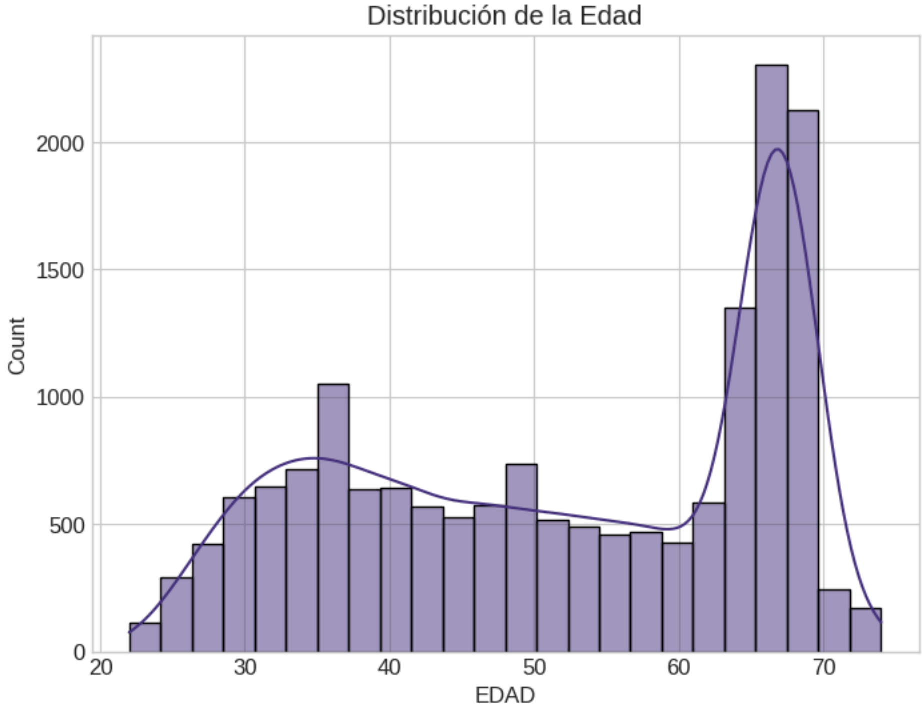
```
df = pd.read_excel(file_path)
df.columns

```

```
/usr/local/lib/python3.11/dist-packages/openpyxl/worksheet/_reader.py:329: UserWarning: Unknown extension is not supported and will
  warn(msg)
Index(['IDENTIFICACION', 'NOMBRE', 'PERFILRIESGOENDEUDAMIENTO', 'EDAD', 'sexo',
       'estado_civil', 'MAXIMA_TARJETA', 'MAXIMO_CONSUMO', 'PAIS_DOM_CAL_DAT',
       'PROV_DOM_CAL_DAT', 'CIUDAD_DOM_CAL_DAT', 'DIR_DOM_CAL_DAT',
       'TEL_DOM_1_CAL_DAT', 'DIR_TRAB_1_CAL_DAT', 'TEL_TRA_1_CAL_DAT',
       'CELULAR_1', 'CELULAR_BAN', 'CORREO_BAN'],
      dtype='object')
```

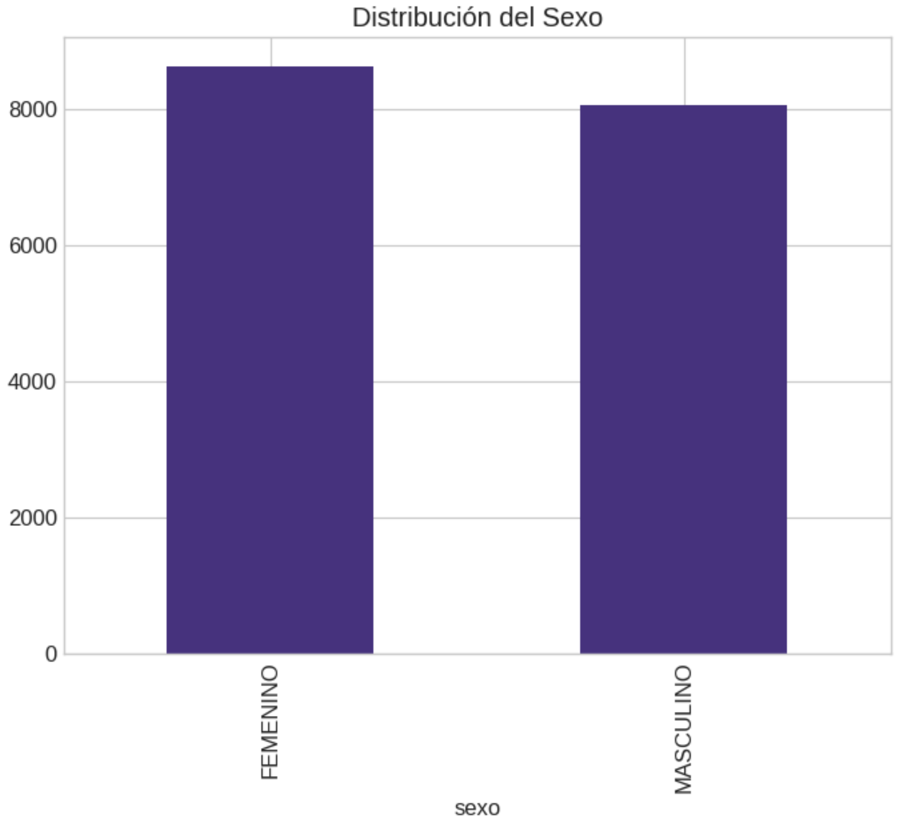
```
# Análisis exploratorio de variables seleccionadas

# Convertir 'EDAD' a numérica (si no lo está ya) y manejar valores no numéricos
df['EDAD'] = pd.to_numeric(df['EDAD'], errors='coerce')

# Análisis de la variable 'EDAD'
print("Análisis de la variable 'EDAD':")
print(df['EDAD'].describe())
plt.figure(figsize=(8, 6))
sns.histplot(df['EDAD'], kde=True)
plt.title('Distribución de la Edad')
plt.show()

# Análisis de la variable 'sexo'
print("\nAnálisis de la variable 'sexo':")
print(df['sexo'].value_counts())
plt.figure(figsize=(8, 6))
df['sexo'].value_counts().plot(kind='bar')
plt.title('Distribución del Sexo')
plt.show()

# Análisis de la variable 'estado_civil'
print("\nAnálisis de la variable 'estado_civil':")
print(df['estado_civil'].value_counts())
plt.figure(figsize=(8, 6))
df['estado_civil'].value_counts().plot(kind='bar')
plt.title('Distribución del Estado Civil')
plt.show()

# Análisis de la variable 'MAXIMA_TARJETA'
print("\nAnálisis de la variable 'MAXIMA_TARJETA':")
print(df['MAXIMA_TARJETA'].describe())
plt.figure(figsize=(8, 6))
sns.histplot(df['MAXIMA_TARJETA'], kde=True)
```

```
plt.title('Distribución del Máximo de Tarjeta')
plt.show()


# Análisis de la variable 'MAXIMO_CONSUMO'
print("\nAnálisis de la variable 'MAXIMO_CONSUMO':")
print(df['MAXIMO_CONSUMO'].describe())
plt.figure(figsize=(8, 6))
sns.histplot(df['MAXIMO_CONSUMO'], kde=True)
plt.title('Distribución del Máximo Consumo')
plt.show()

```

```
Análisis de la variable 'EDAD':
count    16682.000000
mean        51.917636
std         14.487055
min         22.000000
25%         38.000000
50%         54.000000
75%         66.000000
max         74.000000
Name: EDAD, dtype: float64
```

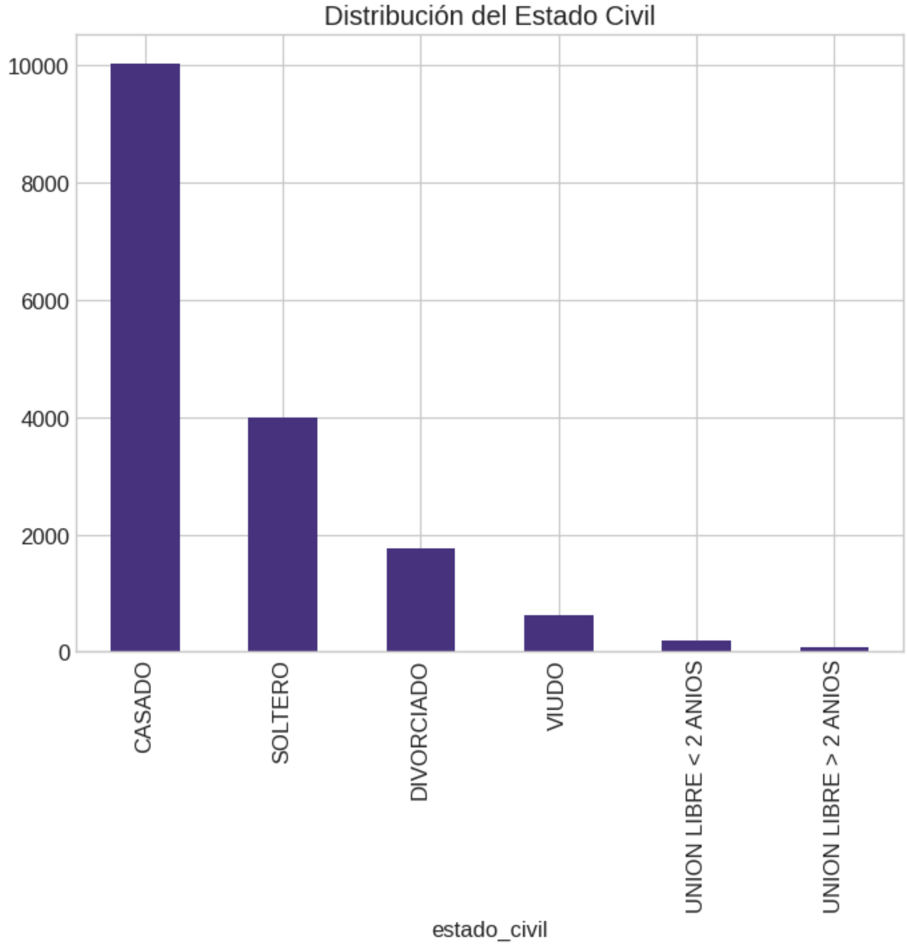



```
Análisis de la variable 'sexo':
sexo
FEMENINO     8624
MASCULINO    8058
Name: count, dtype: int64
```

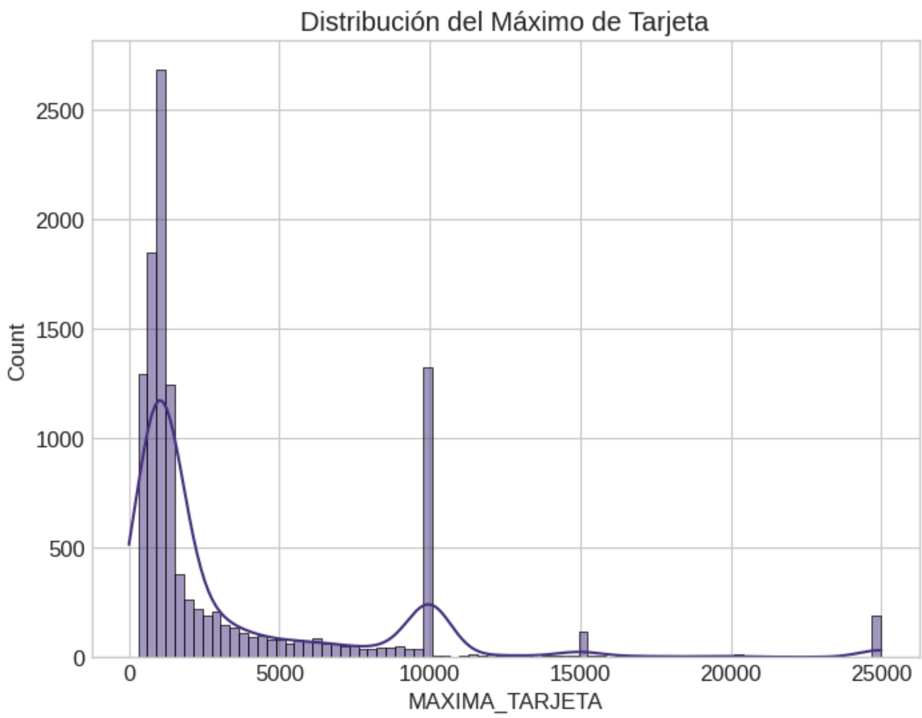



```
Análisis de la variable 'estado_civil':
estado_civil
CASADO                 10019
SOLTERO                 4005
DIVORCIADO              1754
```

```
VIUDO                      632
UNION LIBRE < 2 ANIOS      200
UNION LIBRE > 2 ANIOS       72
Name: count, dtype: int64
```

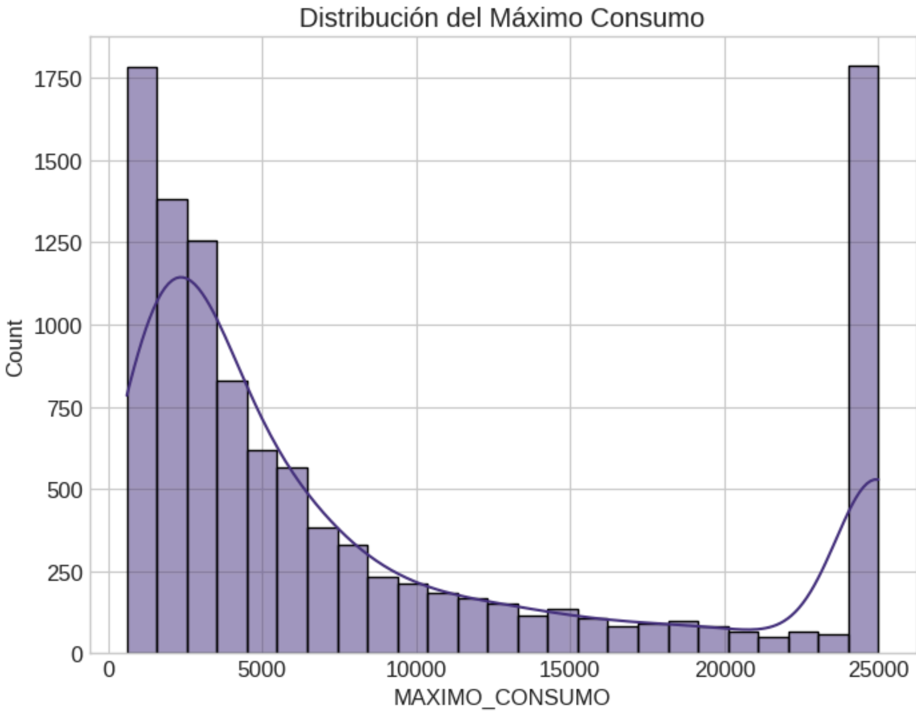



```
Análisis de la variable 'MAXIMA_TARJETA':
count    11859.000000
mean      3606.358040
std       4725.992054
min          0.000000
25%        900.000000
50%       1300.000000
75%       4400.000000
max      25000.000000
Name: MAXIMA_TARJETA, dtype: float64
```

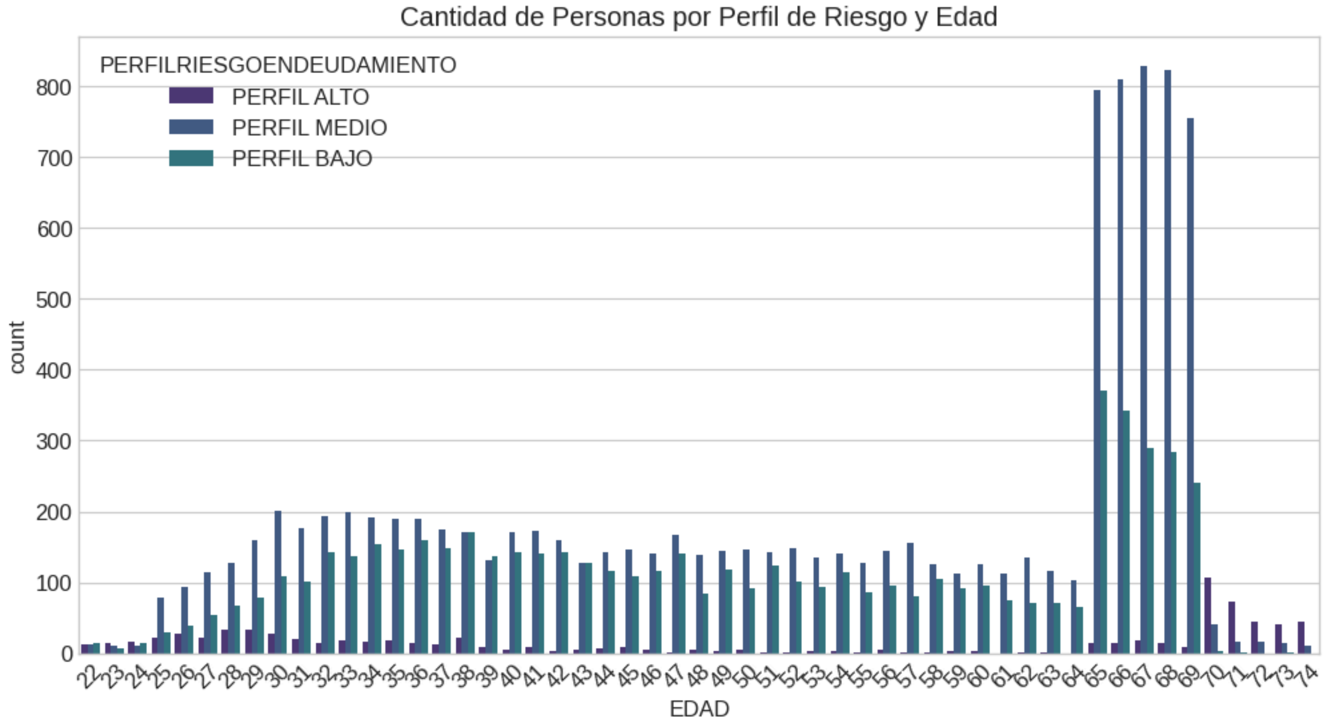



```
Análisis de la variable 'MAXIMO_CONSUMO':
count    10838.000000
```

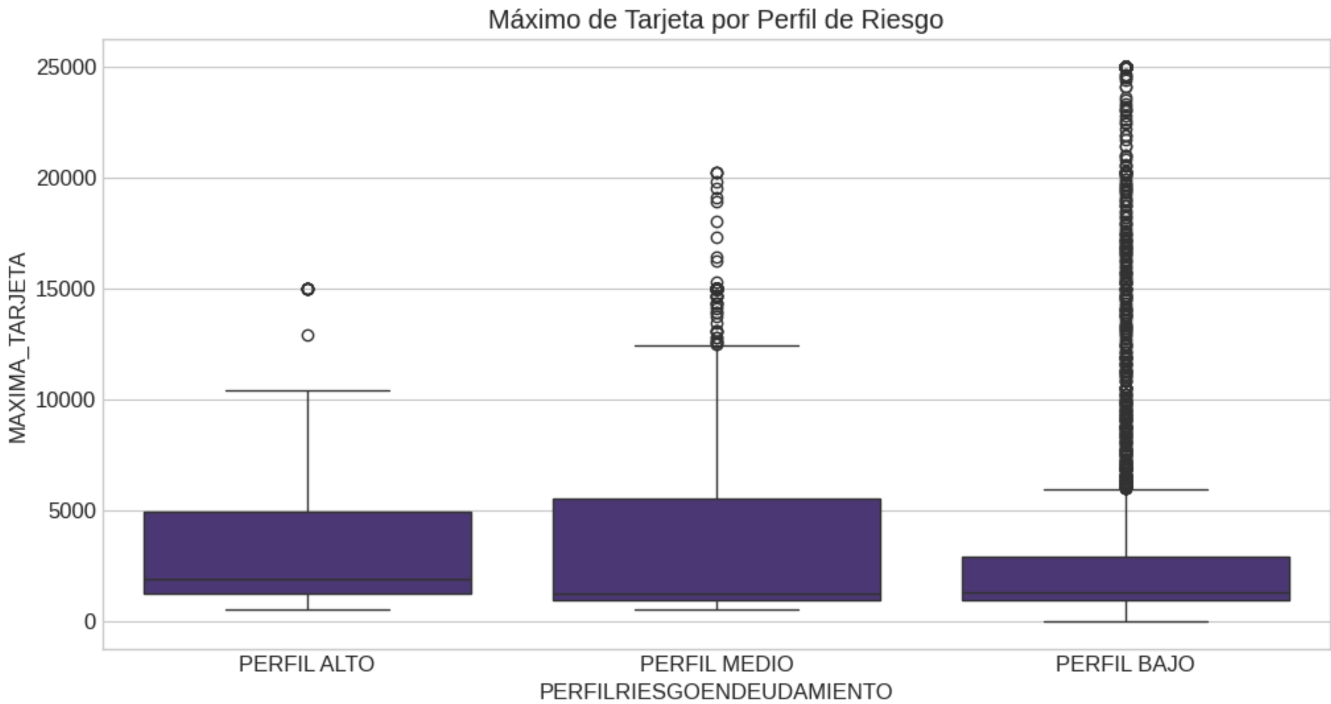
```
mean      8783.327182
std       8632.709979
min        600.000000
25%       2400.000000
50%       4800.000000
75%      13500.000000
max      25000.000000
Name: MAXIMO_CONSUMO, dtype: float64
```



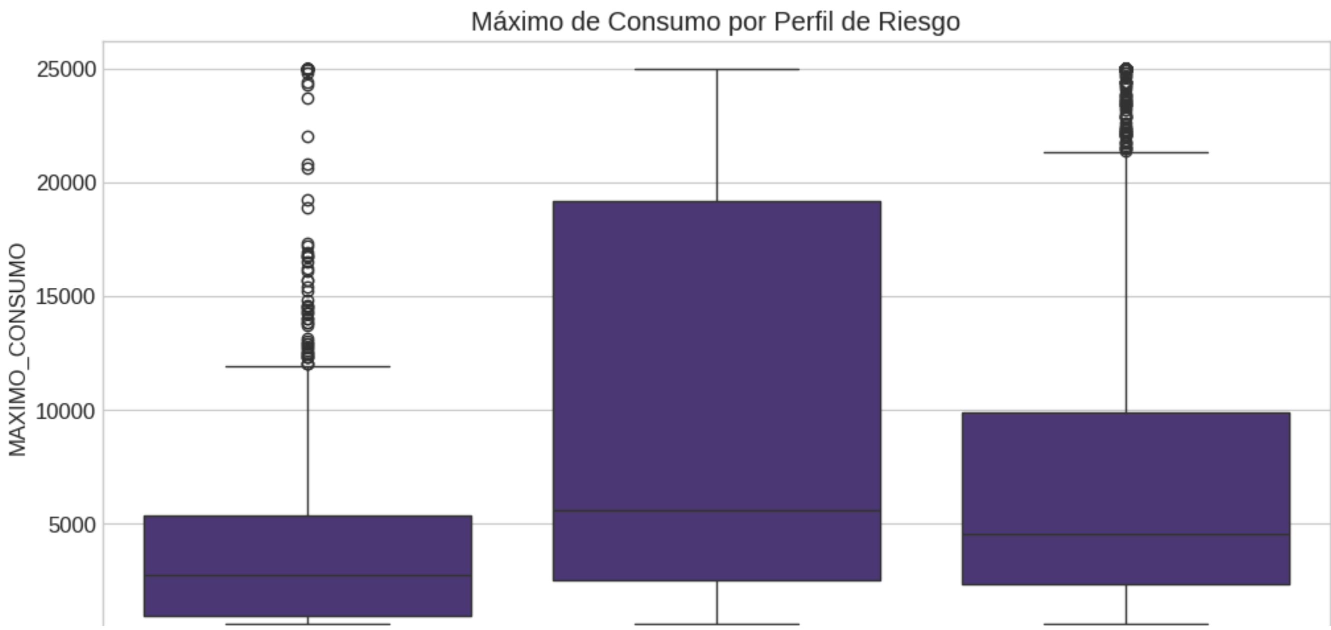
```python
# Análisis de 'PERFILRIESGOENDEUDAMIENTO' vs. otras variables

# 1. Cantidad de personas por perfil de riesgo y edad
plt.figure(figsize=(12, 6))
sns.countplot(x='EDAD', hue='PERFILRIESGOENDEUDAMIENTO', data=df)
plt.title('Cantidad de Personas por Perfil de Riesgo y Edad')
plt.xticks(rotation=45)
plt.show()

# 2. Máximo de tarjeta por perfil de riesgo
plt.figure(figsize=(12, 6))
sns.boxplot(x='PERFILRIESGOENDEUDAMIENTO', y='MAXIMA_TARJETA', data=df)
plt.title('Máximo de Tarjeta por Perfil de Riesgo')
plt.show()

# 3. Máximo de consumo por perfil de riesgo
plt.figure(figsize=(12, 6))
sns.boxplot(x='PERFILRIESGOENDEUDAMIENTO', y='MAXIMO_CONSUMO', data=df)
plt.title('Máximo de Consumo por Perfil de Riesgo')
plt.show()


# 4. Estado civil por perfil de riesgo
plt.figure(figsize=(12, 6))
sns.countplot(x='estado_civil', hue='PERFILRIESGOENDEUDAMIENTO', data=df)
plt.title('Estado Civil por Perfil de Riesgo')
plt.xticks(rotation=45)
plt.show()

# Análisis adicionales (opcional)
# Puedes crear gráficos más específicos o combinar variables para un análisis más profundo:

# Ejemplo: Distribución de la edad para cada perfil de riesgo
for perfil in df['PERFILRIESGOENDEUDAMIENTO'].unique():
    plt.figure(figsize=(8, 6))
    sns.histplot(df[df['PERFILRIESGOENDEUDAMIENTO'] == perfil]['EDAD'], kde=True, label=perfil)
    plt.title(f'Distribución de Edad para el perfil: {perfil}')
    plt.legend()
    plt.show()

```



Cantidad de Personas por Perfil de Riesgo y Edad
PERFILRIESGOENDEUDAMIENTO
PERFIL ALTO
PERFIL MEDIO
PERFIL BAJO
count
EDAD

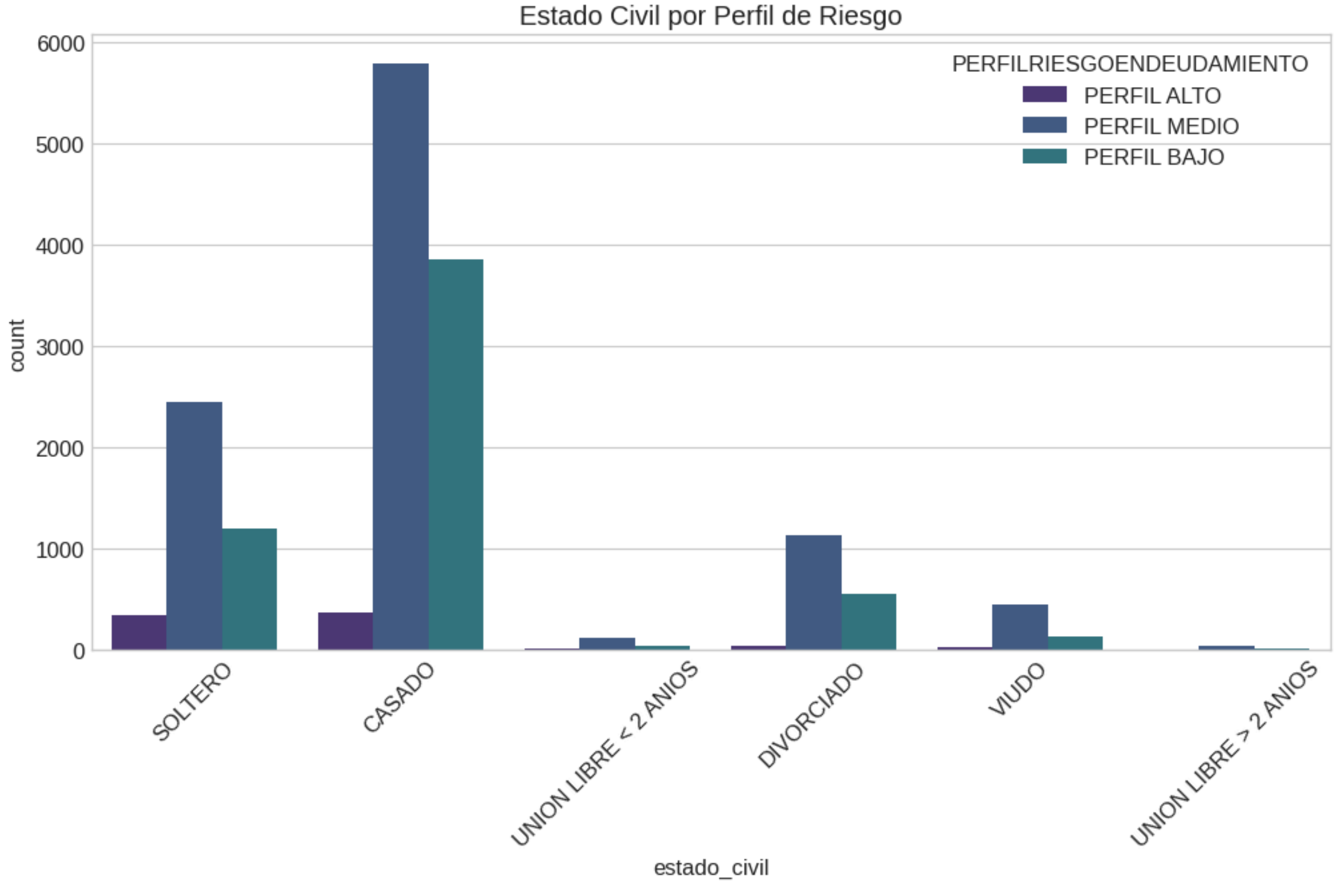


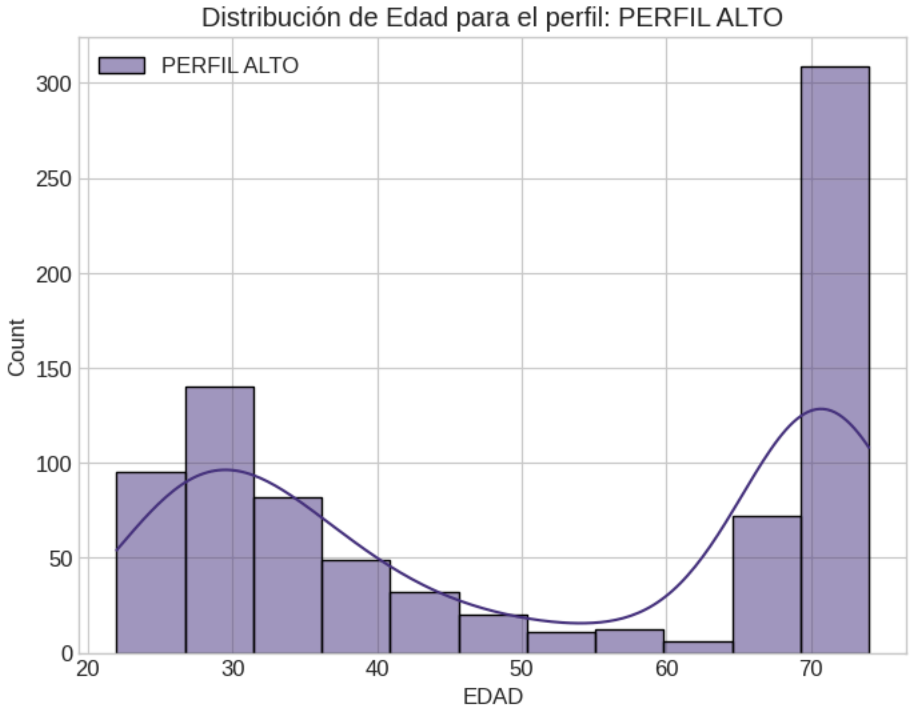
Máximo de Tarjeta por Perfil de Riesgo
MAXIMA_TARJETA
PERFIL ALTO
PERFIL MEDIO
PERFIL BAJO
PERFILRIESGOENDEUDAMIENTO

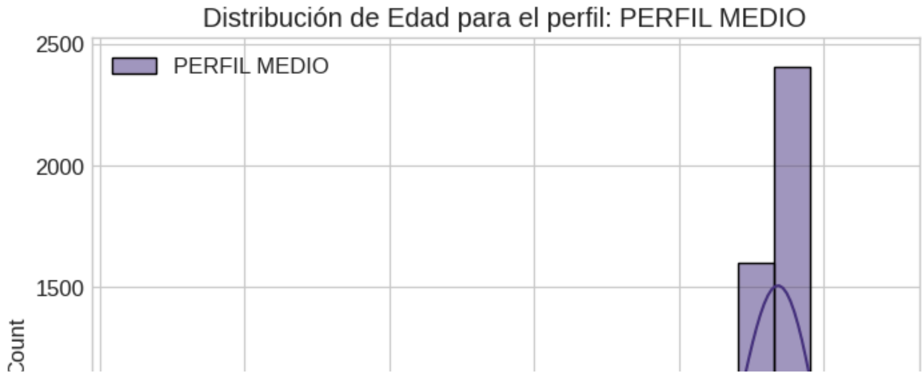


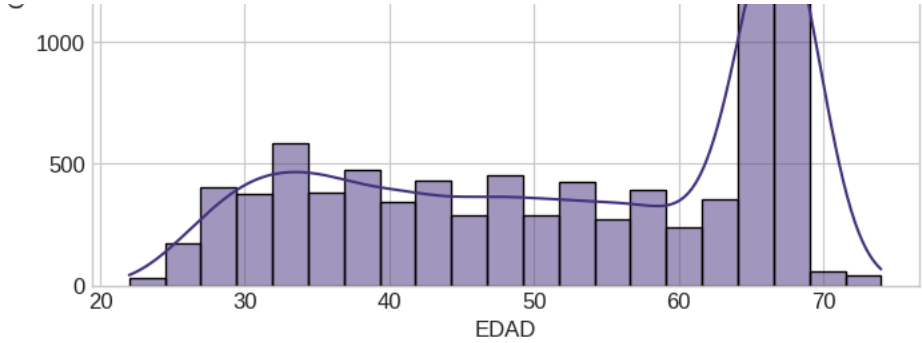
Máximo de Consumo por Perfil de Riesgo
MAXIMO_CONSUMO

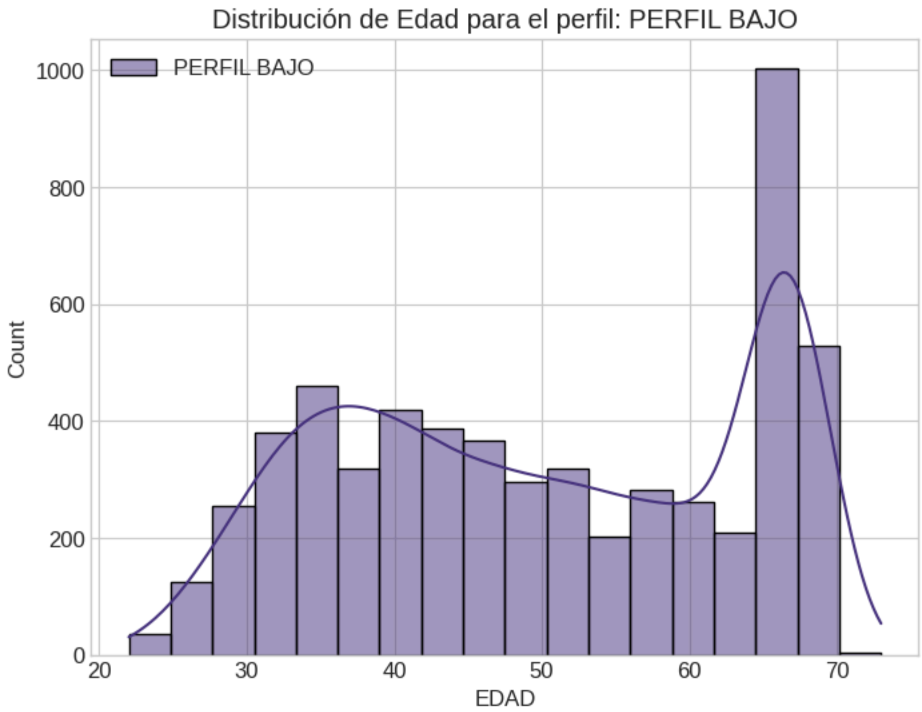

0
PERFIL ALTO
PERFIL MEDIO
PERFIL BAJO
PERFILRIESGOENDEUDAMIENTO


Estado Civil por Perfil de Riesgo
PERFILRIESGOENDEUDAMIENTO
PERFIL ALTO
PERFIL MEDIO
PERFIL BAJO
count
0
1000
2000
3000
4000
5000
6000
SOLTERO
CASADO
UNION LIBRE < 2 ANIOS
DIVORCIADO
VIUDO
UNION LIBRE > 2 ANIOS
estado_civil

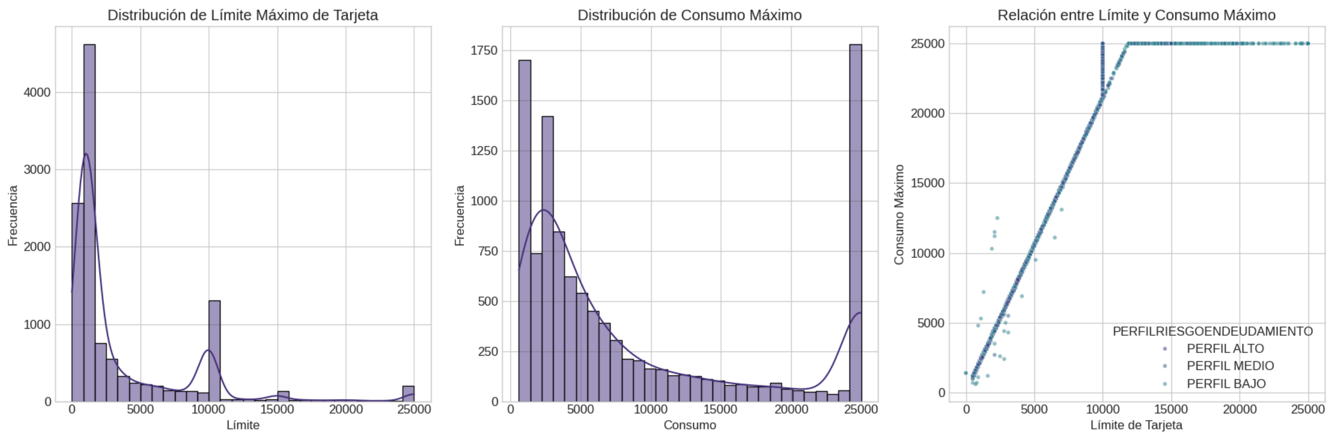

Distribución de Edad para el perfil: PERFIL ALTO
PERFIL ALTO
Count
0
50
100
150
200
250
300
20
30
40
50
60
70
EDAD


Distribución de Edad para el perfil: PERFIL MEDIO
PERFIL MEDIO
1500
2000
2500


1000
500
0
20
30
40
50
60
70
EDAD

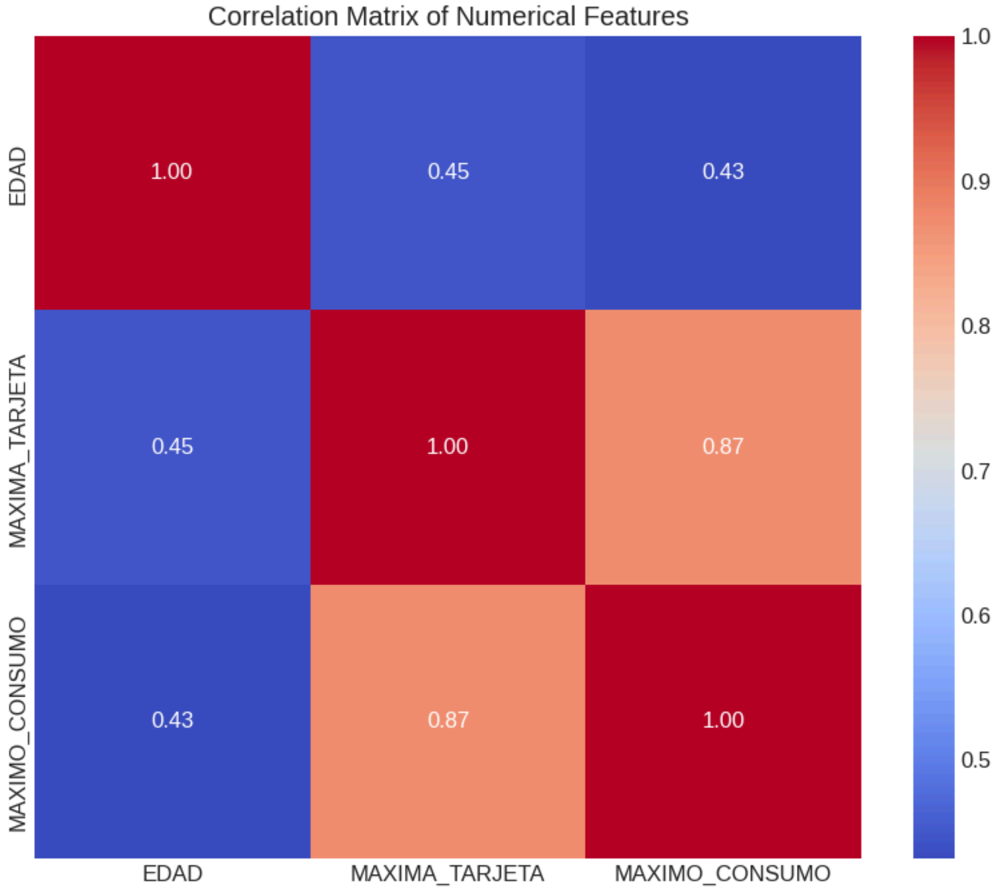

Distribución de Edad para el perfil: PERFIL BAJO
PERFIL BAJO
Count
1000
800
600
400
200
0
20
30
40
50
60
70
EDAD

```python
# 3.2 Análisis de variables financieras
plt.figure(figsize=(18, 6))

# Distribución del límite de tarjeta
plt.subplot(131)
sns.histplot(df['MAXIMA_TARJETA'], kde=True, bins=30)
plt.title('Distribución de Límite Máximo de Tarjeta')
plt.xlabel('Límite')
plt.ylabel('Frecuencia')

# Distribución del consumo máximo
plt.subplot(132)
sns.histplot(df['MAXIMO_CONSUMO'], kde=True, bins=30)
plt.title('Distribución de Consumo Máximo')
plt.xlabel('Consumo')
plt.ylabel('Frecuencia')

# Relación entre límite y consumo
plt.subplot(133)
sns.scatterplot(x='MAXIMA_TARJETA', y='MAXIMO_CONSUMO', data=df,
                hue='PERFILRIESGOENDEUDAMIENTO', alpha=0.5, s=15)
plt.title('Relación entre Límite y Consumo Máximo')
plt.xlabel('Límite de Tarjeta')
plt.ylabel('Consumo Máximo')

plt.tight_layout()
plt.savefig('financieros.png')
plt.close()
from IPython.display import Image
display(Image('financieros.png'))
```




```python
# Assuming 'df' is your DataFrame from the previous code

# Select numerical columns for correlation analysis
numerical_cols = ['EDAD', 'MAXIMA_TARJETA', 'MAXIMO_CONSUMO']
numerical_df = df[numerical_cols]

# Calculate the correlation matrix
correlation_matrix = numerical_df.corr()

# Display the correlation matrix
print(correlation_matrix)

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()

```

```
                     EDAD  MAXIMA_TARJETA  MAXIMO_CONSUMO
EDAD             1.000000        0.445320        0.431907
MAXIMA_TARJETA   0.445320        1.000000        0.870249
MAXIMO_CONSUMO   0.431907        0.870249        1.000000
```




```python
# prompt: remover del dataset las siguientes columnas: 'DIR_DOM_CAL_DAT', 'DIR_TRAB_1_CAL_DAT', 'NOMBRE'

# Remove specified columns
df = df.drop(columns=['DIR_DOM_CAL_DAT', 'DIR_TRAB_1_CAL_DAT', 'NOMBRE'], errors='ignore')

```

```python
# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()

# Print the categorical columns
print("Categorical columns:")
categorical_cols

```

```
Categorical columns:
['PERFILRIESGOENDEUDAMIENTO',
 'sexo',
 'estado_civil',
 'PAIS_DOM_CAL_DAT',
 'PROV_DOM_CAL_DAT',
 'CIUDAD_DOM_CAL_DAT',
 'CORREO_BAN']
```

```python
encoding_maps = {}  # Diccionario para almacenar las bibliotecas de valores
for col in categorical_cols:
    if col != 'IDENTIFICACION':  # No transformar la columna 'IDENTIFICACION'
        unique_values = df[col].unique()
        encoding_map = {val: i for i, val in enumerate(unique_values)}  # Asignar números consecutivos
        df[col] = df[col].map(encoding_map)
        encoding_maps[col] = encoding_map
# Mostrar las bibliotecas de valores y el DataFrame transformado
print("\nBibliotecas de valores:")
for col, map in encoding_maps.items():
    print(f"{col}: {map}")
print("\nDataFrame transformado:")
# No usar decimales en la salida del DataFrame
with pd.option_context('display.float_format', '{:.0f}'.format):
    print(df.head())

```

```
Bibliotecas de valores:
PERFILRIESGOENDEUDAMIENTO: {'PERFIL ALTO': 0, 'PERFIL MEDIO': 1, 'PERFIL BAJO': 2}
sexo: {'MASCULINO': 0, 'FEMENINO': 1}
estado_civil: {'SOLTERO': 0, 'CASADO': 1, 'UNION LIBRE < 2 ANIOS': 2, 'DIVORCIADO': 3, 'VIUDO': 4, 'UNION LIBRE > 2 ANIOS': 5}
PAIS_DOM_CAL_DAT: {'ECUADOR': 0}
PROV_DOM_CAL_DAT: {'PICHINCHA': 0}
CIUDAD_DOM_CAL_DAT: {'QUITO': 0}
CORREO_BAN: {'oliver_leo1994@hotmail,es': 0, 'rcristofer_dtb@hotmail,com': 1, 'andresquevedoandy@live.com.mx': 2, 'leojhongrefa@gma

DataFrame transformado:
   IDENTIFICACION  PERFILRIESGOENDEUDAMIENTO  EDAD  sexo  estado_civil  \
0      2350359549                          0    24     0             0
1      2300650476                          0    23     0             0
2      2300107253                          1    26     0             1
3      2200176531                          0    25     0             0
4      2200046163                          2    28     0             0

   MAXIMA_TARJETA  MAXIMO_CONSUMO  PAIS_DOM_CAL_DAT  PROV_DOM_CAL_DAT  \
0             NaN            2900                 0                 0
1             NaN             800                 0                 0
2            1000             NaN                 0                 0
3             NaN             800                 0                 0
4            1000             NaN                 0                 0

   CIUDAD_DOM_CAL_DAT  TEL_DOM_1_CAL_DAT  TEL_TRA_1_CAL_DAT  CELULAR_1  \
0                   0           22420481                NaN  960605351
1                   0           23447377                NaN  967428769
2                   0           22441940           23971000  988676891
3                   0                NaN                NaN  998645219
4                   0                NaN                NaN  939070056

   CELULAR_BAN  CORREO_BAN
0    993931984           0
1    997398980           1
2    992510700           2
3    985945741           3
4    939983650           4
```

```
# Manejar valores faltantes
for column in df.columns:
    df[column].fillna(df[column].mode()[0], inplace=True)
```

```
  df[column].fillna(df[column].mode()[0], inplace=True)
<ipython-input-217-df6bc2e8bee9>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett:

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  df[column].fillna(df[column].mode()[0], inplace=True)
<ipython-input-217-df6bc2e8bee9>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett:

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  df[column].fillna(df[column].mode()[0], inplace=True)
<ipython-input-217-df6bc2e8bee9>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett:

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  df[column].fillna(df[column].mode()[0], inplace=True)
<ipython-input-217-df6bc2e8bee9>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett:

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  df[column].fillna(df[column].mode()[0], inplace=True)
<ipython-input-217-df6bc2e8bee9>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett:

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  df[column].fillna(df[column].mode()[0], inplace=True)
<ipython-input-217-df6bc2e8bee9>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett:
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  df[column].fillna(df[column].mode()[0], inplace=True)
<ipython-input-217-df6bc2e8bee9>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chainec
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setti

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  df[column].fillna(df[column].mode()[0], inplace=True)
```

```
pip install pandas scikit-learn gower
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: gower in /usr/local/lib/python3.11/dist-packages (0.1.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

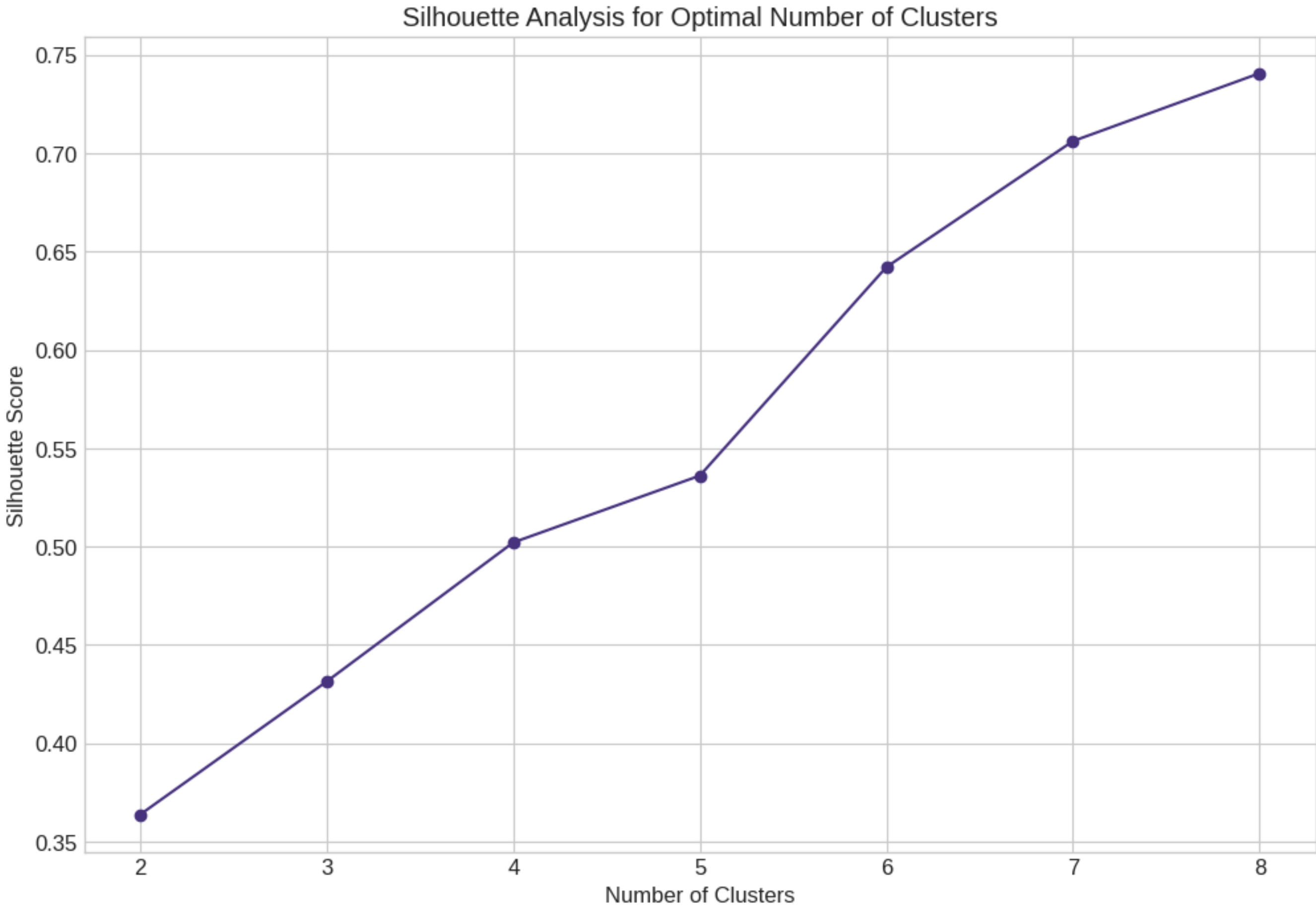
```
# Separar variables numéricas y categóricas
numerical_features = df.select_dtypes(include=np.number).columns.tolist()
categorical_features = df.select_dtypes(exclude=np.number).columns.tolist()

# Transformación de variables categóricas (One-Hot Encoding)
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded_categorical = encoder.fit_transform(df[categorical_features])
encoded_categorical_df = pd.DataFrame(encoded_categorical, index=df.index)

# Get feature names from encoder
encoded_feature_names = encoder.get_feature_names_out(categorical_features)
encoded_categorical_df.columns = encoded_feature_names # Assign feature names to columns

df = pd.concat([df[numerical_features], encoded_categorical_df], axis=1)

# Replace infinite values with NaN
df.replace([np.inf, -np.inf], np.nan, inplace=True)

# Impute NaN values (if any) with the mean of the column
for col in numerical_features:
    df[col].fillna(df[col].mean(), inplace=True)

# Ensure all column names are strings before scaling
df.columns = df.columns.astype(str)

# Escalar variables numéricas
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

```
<ipython-input-219-3883bf016583>:21: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained a
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df[col].fillna(df[col].mean(), inplace=True)
```

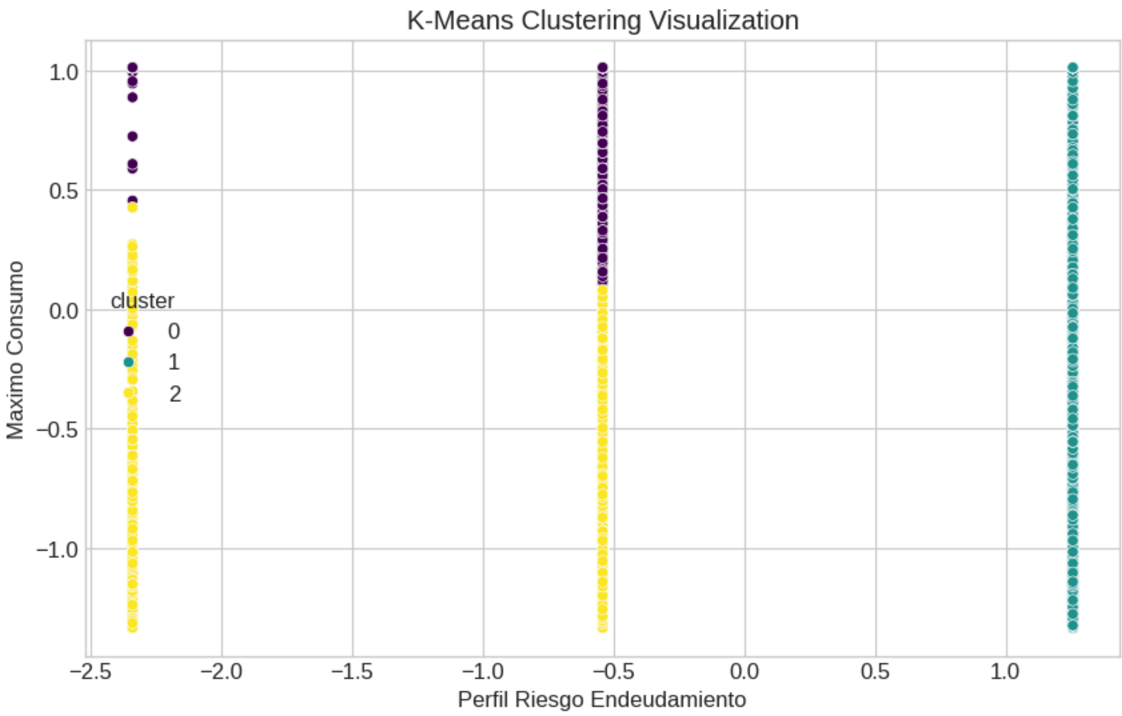
```
# prompt: realizar algoritmo k-,means para poder definir los clusters más logicos según el perfil de endeudamiento  y maximo de cons

# Define features for clustering
features_to_cluster = ['PERFILRIESGOENDEUDAMIENTO', 'MAXIMA_TARJETA', 'MAXIMO_CONSUMO']

# Filter DataFrame for selected features
X = df[features_to_cluster]

# Determine optimal number of clusters using Silhouette analysis
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
silhouette_scores = []

for n_clusters in range_n_clusters:
```

```
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(X)
    silhouette_avg = silhouette_score(X, cluster_labels)
    silhouette_scores.append(silhouette_avg)
    print(f"For n_clusters = {n_clusters}, the average silhouette_score is: {silhouette_avg}")

# Plot Silhouette scores
plt.plot(range_n_clusters, silhouette_scores, marker='o')
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Analysis for Optimal Number of Clusters")
plt.show()

# Perform K-Means clustering with the optimal number of clusters (determined visually from the plot above)
optimal_n_clusters = 3 # Replace with the visually determined optimal k from the plot
kmeans = KMeans(n_clusters=optimal_n_clusters, random_state=42)
df['cluster'] = kmeans.fit_predict(X)

# Analyze and visualize the clusters
# Example: Calculate cluster means for each feature
cluster_means = df.groupby('cluster')[features_to_cluster].mean()
print("\nCluster Means:")
print(cluster_means)

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='PERFILRIESGOENDEUDAMIENTO', y='MAXIMO_CONSUMO', hue='cluster', palette='viridis')
plt.title('K-Means Clustering Visualization')
plt.xlabel('Perfil Riesgo Endeudamiento') # Updated x-axis label
plt.ylabel('Maximo Consumo')
plt.show()
```

```
For n_clusters = 2, the average silhouette_score is: 0.3637257445503861
For n_clusters = 3, the average silhouette_score is: 0.43140390816397345
For n_clusters = 4, the average silhouette_score is: 0.5021076585191356
For n_clusters = 5, the average silhouette_score is: 0.5360789095668325
For n_clusters = 6, the average silhouette_score is: 0.642139587966258
For n_clusters = 7, the average silhouette_score is: 0.7061838400023218
For n_clusters = 8, the average silhouette_score is: 0.7406880606947717
```

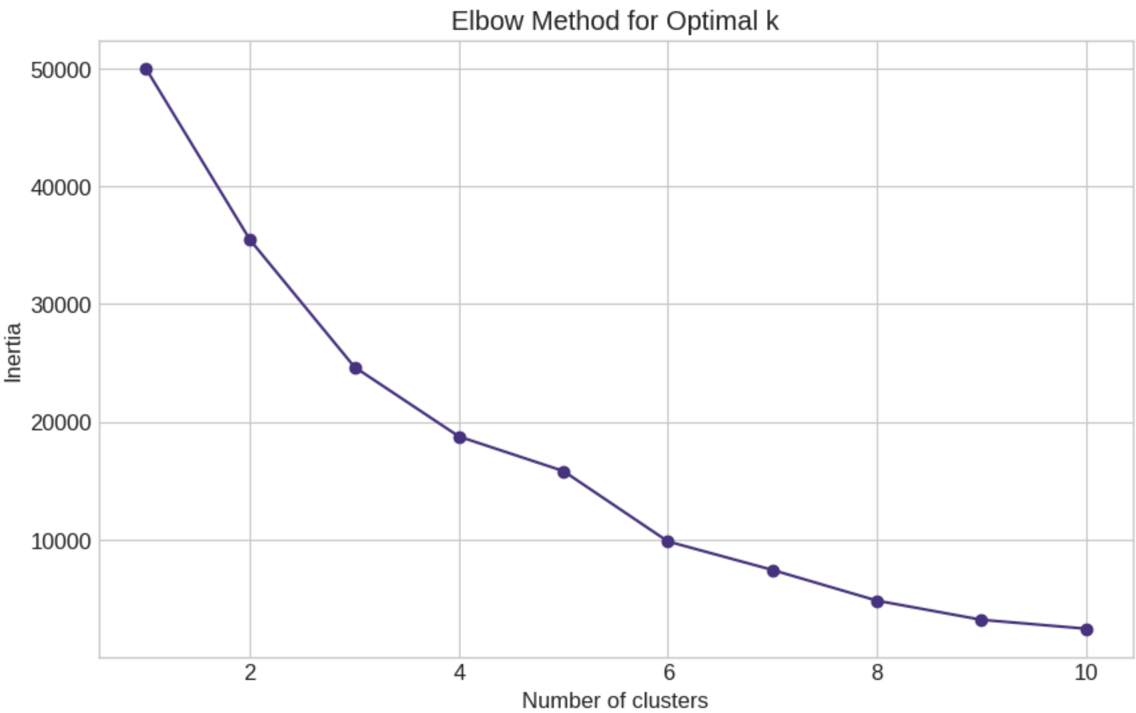



```
Cluster Means:
         PERFILRIESGOENDEUDAMIENTO  MAXIMA_TARJETA  MAXIMO_CONSUMO
cluster
0                        -0.557605       -0.314335        0.968812
1                         1.256783        0.185494       -0.094374
2                        -0.810493        0.136323       -0.961597
```

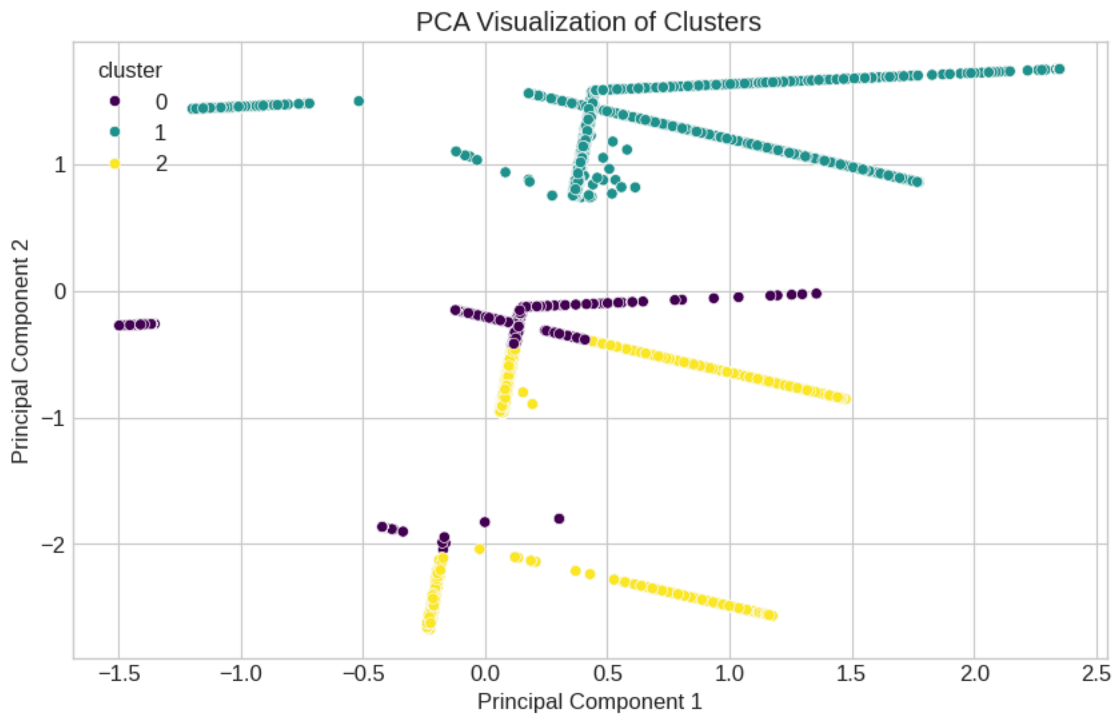



```
1 # prompt: grafico de codo
2
```

```
# Elbow Method for optimal k
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()

```




```
# Apply PCA
pca = PCA(n_components=2)  # Reduce to 2 principal components
df_pca = pca.fit_transform(X)

# Create a DataFrame for the PCA results
df_pca = pd.DataFrame(data=df_pca, columns=['PC1', 'PC2'])
df_pca['cluster'] = df['cluster']  # Add the cluster labels

# Visualize the PCA results
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_pca, x='PC1', y='PC2', hue='cluster', palette='viridis')
plt.title('PCA Visualization of Clusters')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# Explained Variance Ratio
explained_variance_ratio = pca.explained_variance_ratio_
print(f"Explained Variance Ratio: {explained_variance_ratio}")
print(f"Total Explained Variance: {np.sum(explained_variance_ratio)}")

```



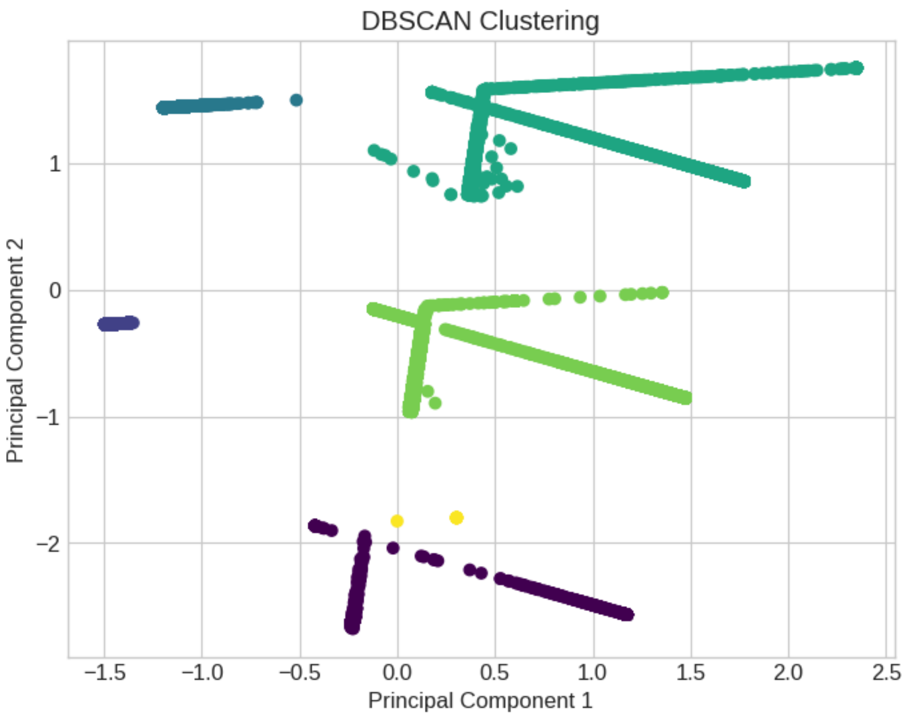

```
Explained Variance Ratio: [0.42833476 0.33762586]
Total Explained Variance: 0.7659606166129003
```

```
# Identify outliers using IQR method
def find_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    return outliers

outlier_data = {}
for col in features_to_cluster:
    outliers = find_outliers_iqr(df[col])
    outlier_data[col] = outliers

# Create a DataFrame for outliers
outliers_df = pd.DataFrame(outlier_data)

# Download the outliers to a CSV file
outliers_df.to_csv('outliers.csv', index=False)
files.download('outliers.csv')

```

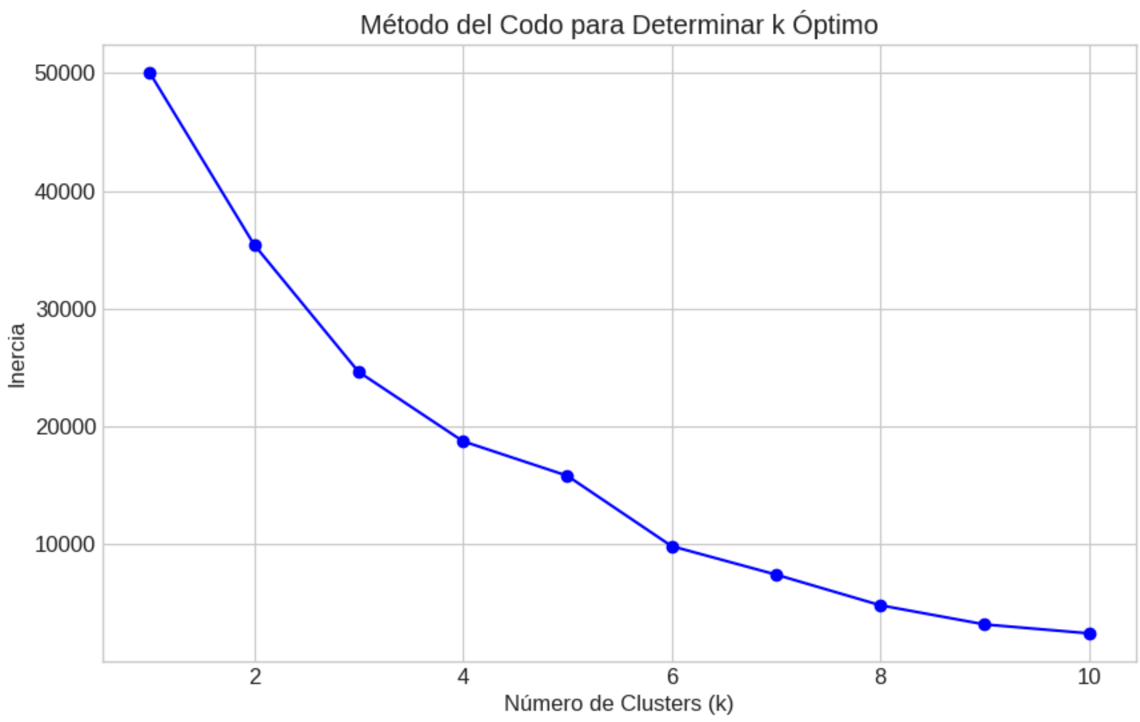
```
from sklearn.cluster import DBSCAN
# Suponiendo que 'df' es tu DataFrame y 'X' contiene las características numéricas para el clustering
# Por ejemplo:
X = df[['PERFILRIESGOENDEUDAMIENTO', 'MAXIMA_TARJETA', 'MAXIMO_CONSUMO']]

# Escalar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Aplicar DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)  # Ajustar eps y min_samples según tus datos
clusters = dbscan.fit_predict(X_scaled)

# Agregar los clusters al DataFrame
df['cluster'] = clusters

# Analizar los resultados
print(df['cluster'].value_counts())
# Visualizar los clusters (ejemplo con las primeras dos componentes principales)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis')
```

```
plt.title('DBSCAN Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

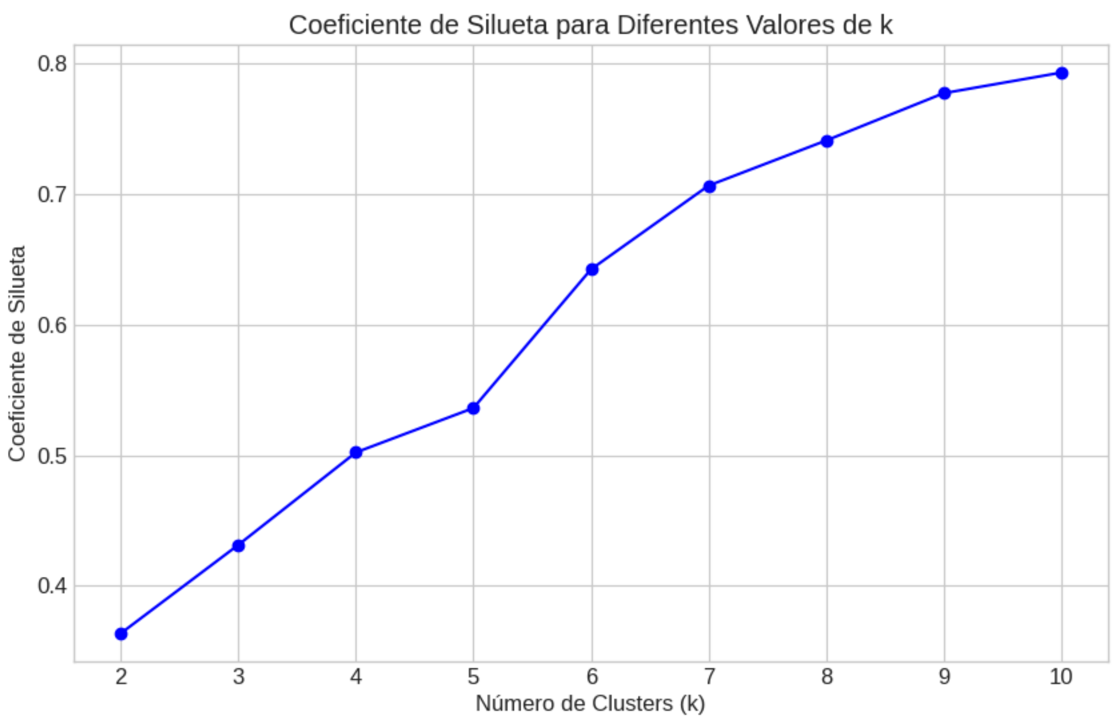
```
cluster
4    6146
3    3864
1    3863
2    1981
0     819
5       9
Name: count, dtype: int64
```




```
# 7. Exploración de patrones mediante clustering preliminar
# Seleccionamos las variables más relevantes para el clustering
X = df[['PERFILRIESGOENDEUDAMIENTO', 'MAXIMA_TARJETA', 'MAXIMO_CONSUMO']]

# Preprocesamiento: escalado de variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 7.1 Determinación del número óptimo de clusters con el método del codo
inertias = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(K_range, inertias, 'bo-')
plt.title('Método del Codo para Determinar k Óptimo')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Inercia')
plt.grid(True)
plt.savefig('metodo_codo.png')
plt.close()
from IPython.display import Image
display(Image('metodo_codo.png'))
```




```
# 7.2 Evaluación con el coeficiente de silueta
silhouette_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(X_scaled)
    silhouette_avg = silhouette_score(X_scaled, cluster_labels)
    silhouette_scores.append(silhouette_avg)

plt.figure(figsize=(10, 6))
plt.plot(range(2, 11), silhouette_scores, 'bo-')
plt.title('Coeficiente de Silueta para Diferentes Valores de k')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Coeficiente de Silueta')
plt.grid(True)
plt.savefig('silueta.png')
plt.close()
from IPython.display import Image
display(Image('silueta.png'))
```




```
optimal_k = 3 #Reemplaza con el valor optimo de k obtenido de las graficas
print(f"El número óptimo de clusters es: {optimal_k}")

```

```
El número óptimo de clusters es: 3
```

```
# 7.3 Aplicación preliminar de K-means con k=3 (valor asumido óptimo)
from sklearn.cluster import KMeans # Make sure KMeans is imported
from sklearn.preprocessing import StandardScaler

# Select features for clustering
X = df[['PERFILRIESGOENDEUDAMIENTO', 'MAXIMA_TARJETA', 'MAXIMO_CONSUMO']]

# Preprocessing: scaling features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # Define X_scaled here before using it

kmeans = KMeans(n_clusters=5, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled)
```

```
# Visualización de los clusters en 2D usando PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(12, 10))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['cluster'], cmap='viridis', alpha=0.6, s=30)
plt.colorbar(scatter, label='Cluster')
plt.title('Visualización de Clusters Mediante PCA')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.savefig('clusters_pca.png')
plt.close()
from IPython.display import Image
display(Image('clusters_pca.png'))
```