Objetivo del Análisis y Enfoque General

El objetivo de este análisis es conocer qué tanto influyen diferentes variables como el nivel educativo, el ingreso, el género y la posesión de productos bancarios, entre otros, en el nivel de satisfacción de los clientes bancarios. La satisfacción se mide en una escala del 1 al 4, donde 1 significa “muy insatisfecho” y 4 “muy satisfecho”, y es la variable que queremos explicar. Para lograr esto, usamos tres modelos de clasificación en machine learning: Árbol de decisión, Bosque aleatorio y Red neuronal. Estos modelos ayudan a predecir el nivel de satisfacción de cada cliente según sus características y también a identificar qué variables son más importantes. Con estas técnicas, no solo podemos hacer mejores predicciones, sino también entender qué factores afectan más la satisfacción del cliente, lo que es útil para mejorar el servicio, mantener a los clientes fidelizados y mejora del servicio en el ámbito bancario.

Análisis Exploratorio de Datos (EDA)

Para comenzar con el análisis, lo primero fue limpiar los datos. Se eliminaron algunas columnas que no aportaban valor, como los identificadores (id, Unnamed: 0), ya que solo servían para registrar cada fila y no eran útiles para el análisis. Luego, se revisaron las variables de texto, como gender, que aparecía escrita de distintas formas (“female”, “male”). Para evitar posibles errores al aplicar los modelos de machine learning, se creó una nueva variable llamada gender_bin, que asigna un 0 si el cliente es mujer y un 1 si es hombre. Esto permite que los algoritmos, como los árboles de decisión o las redes neuronales, interpreten la información de manera más sencilla y eficiente.

También se transformaron variables importantes para el análisis. Por ejemplo, education fue tratada como una variable ordinal para respetar la jerarquía de niveles educativos, mientras que satisfaction, la variable que queremos predecir, se definió como categórica. Esta preparación fue clave para garantizar que los modelos pudieran trabajar correctamente con los datos.

Después, se clasificaron las variables según su tipo, lo que facilitó una mejor organización del conjunto de datos y ayudó a identificar cuáles serían útiles para el modelado. La clasificación fue la siguiente:

Variable objetivo:

- satisfaction: indica el nivel de satisfacción del cliente (de 1 a 4) y fue tratada como una variable categórica.

Variables numéricas continuas:

- age: edad del cliente.
- pincome: ingreso personal del cliente.

Variables binarias (valores 0 o 1, donde 1 indica presencia del producto o servicio):

- car, phone, fax, pc, savings1, savings2, creditcard1, creditcard2, atmcard1, atmcard2, cd1, cd2, specialchecking1, specialchecking2, auto.bill.payment1, auto.bill.payment2, personal.loans1, personal.loans2, mortgage1, mortgage2, installment.loan1, installment.loan2, investment.fund1, investment.fund2, commodities.fund1, commodities.fund2, annuities.fund1, annuities.fund2, car.insurance1, car.insurance2, home.insurance1, home.insurance2, life.insurance1, life.insurance2, y la nueva variable gender_bin.

Variable categórica ordinal:

- education: representa el nivel educativo del cliente, con un orden definido (de menor a mayor).

Variables descartadas:

- id, Unnamed: 0: eliminadas por ser identificadores sin valor predictivo.
- gender: reemplazada por su versión binaria gender_bin.

Además de esta organización, se realizaron validaciones lógicas para asegurar la calidad del dataset. Se revisaron casos como ingresos negativos, edades fuera de un rango realista (por ejemplo, menores de 0 o mayores de 100), o combinaciones ilógicas como tener fax sin teléfono, o tener coche siendo menores de edad. También se comprobó que no existieran datos ausentes. Estos controles garantizaron que los datos fueran coherentes antes de usarlos en los modelos predictivos.

Finalmente, se hizo una exploración visual para entender el comportamiento de las variables. Se usaron boxplots para observar cómo varían la edad y el ingreso según el nivel de satisfacción, y gráficos de barras para explorar relaciones entre la satisfacción y otras variables como el nivel educativo. Estas visualizaciones ayudaron a identificar patrones y posibles hipótesis, útiles para el desarrollo de los modelos de clasificación.

In [1]:
```
# ========================================
# 1. CARGA DE LIBRERÍAS Y DATOS
# ========================================
import pandas as pd
from pandas.api.types import CategoricalDtype
import seaborn as sns
import matplotlib.pyplot as plt
import gdown

# Descargar y cargar archivo
url = "https://drive.google.com/uc?id=1WDJQQf-Y7NSFUptLEIiHRAAOfJAM8sYj"
```

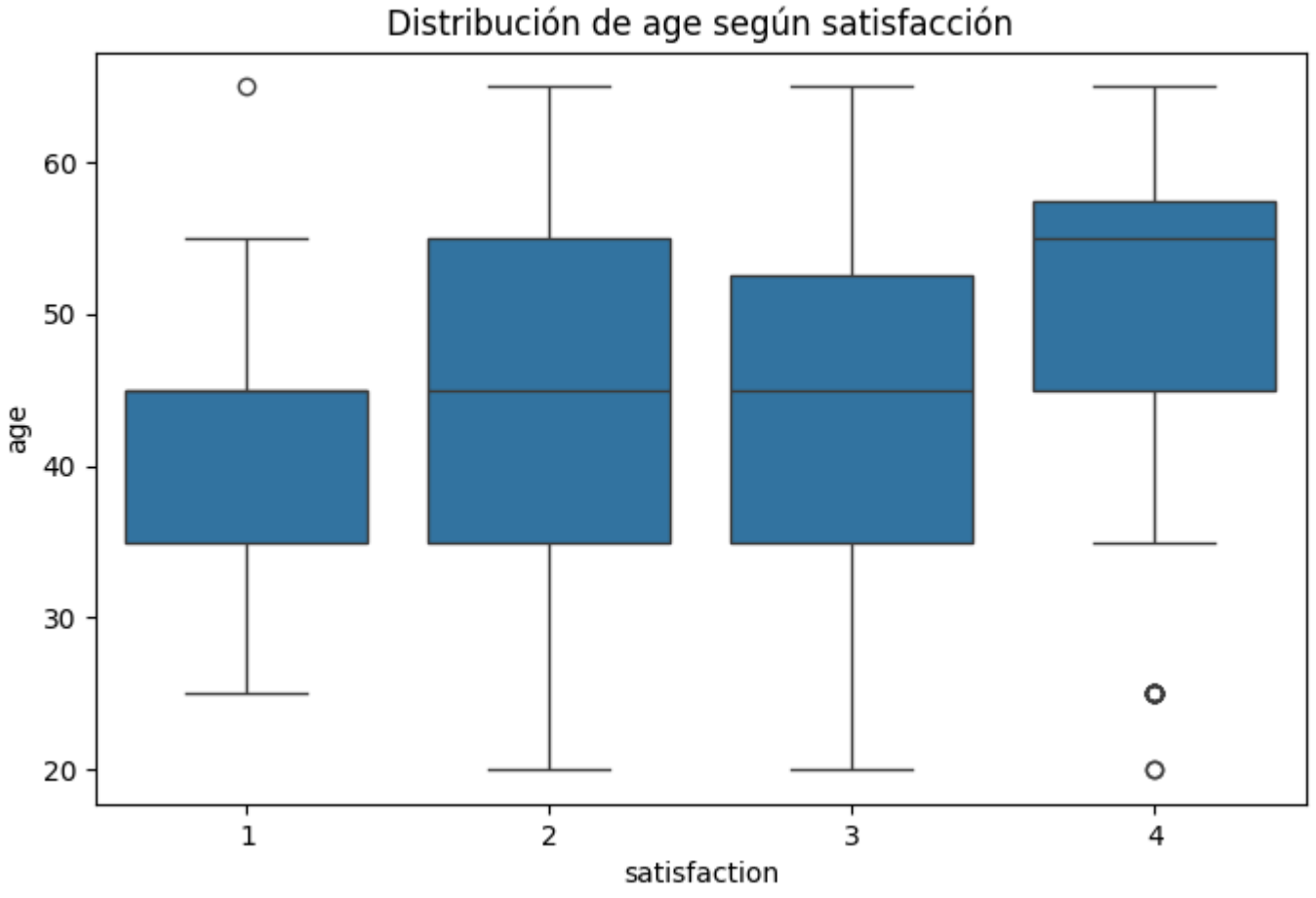
```python
output = "datos.csv"
gdown.download(url, output, quiet=False)
df = pd.read_csv(output)

# ======================================
# 2. LIMPIEZA Y PREPARACIÓN DE VARIABLES
# ======================================

# Eliminar columnas innecesarias (identificadores)
columnas_excluir = ['Unnamed: 0', 'id']
df_analisis = df.drop(columns=columnas_excluir)

# Normalizar texto en 'gender' y crear versión binaria
df_analisis['gender'] = df_analisis['gender'].str.strip().str.lower()
df_analisis['gender_bin'] = df_analisis['gender'].map({'female': 0, 'male': 1})

# Convertir 'education' en variable ordinal categórica
orden_education = CategoricalDtype(categories=[1, 2, 3, 4, 5], ordered=True)
df_analisis['education'] = df_analisis['education'].astype(orden_education)

# Convertir 'satisfaction' en variable categórica (target)
df_analisis['satisfaction'] = df_analisis['satisfaction'].astype('category')

# ======================================
# 3. CLASIFICACIÓN DE VARIABLES
# ======================================

# Variable objetivo (target)
target = 'satisfaction'  # Categórica

# Variables continuas (numéricas)
variables_continuas = ['age', 'pincome']

#  Variables binarias (valores 0 y 1)
variables_binarias = [
    'car', 'phone', 'fax', 'pc', 'savings1', 'savings2',
    'creditcard1', 'creditcard2', 'atmcard1', 'atmcard2',
    'cd1', 'cd2', 'specialchecking1', 'specialchecking2',
    'auto.bill.payment1', 'auto.bill.payment2', 'personal.loans1',
    'personal.loans2', 'mortgage1', 'mortgage2', 'installment.loan1',
    'installment.loan2', 'investment.fund1', 'investment.fund2',
    'commodities.fund1', 'commodities.fund2', 'annuities.fund1', 'annuities.fund2',
    'car.insurance1', 'car.insurance2', 'home.insurance1', 'home.insurance2',
    'life.insurance1', 'life.insurance2',
    'gender_bin'  # agregada manualmente
]

# Variables categóricas ordinales
variables_ordinales = ['education']

# Variables descartadas
variables_descartadas = ['Unnamed: 0', 'id', 'gender']  # ya no se usan

# ======================================
# 4. VARIABLES FINALES PARA MODELADO
# ======================================

# Variables predictoras finales que se usarán en los modelos
variables_modelo = variables_continuas + variables_binarias + variables_ordinales

print(f"Variable respuesta: {target}")
print(f"Variables numéricas: {variables_continuas}")
print(f"Variables binarias: {variables_binarias}")
print(f"Variables ordinales: {variables_ordinales}")
print(f"Variables eliminadas: {variables_descartadas}")
print(f"Variables finales para los modelos: {variables_modelo}")

#######################################################

# ======================================
# 5. VALIDACIONES LÓGICAS DE CALIDAD DE DATOS
# ======================================

# Valores imposibles o fuera de rango
print("\nEdades fuera de rango lógico (<0 o >100):")
print(df_analisis[(df_analisis['age'] < 0) | (df_analisis['age'] > 100)])

print("\nIngresos personales negativos:")
print(df_analisis[df_analisis['pincome'] < 0])

print("\nCasos con fax=1 pero phone=0 (inconsistencia):")
print(df_analisis[(df_analisis['fax'] == 1) & (df_analisis['phone'] == 0)])

print("\nCasos con coche pero edad < 18:")
print(df_analisis[(df_analisis['car'] == 1) & (df_analisis['age'] < 18)])
```

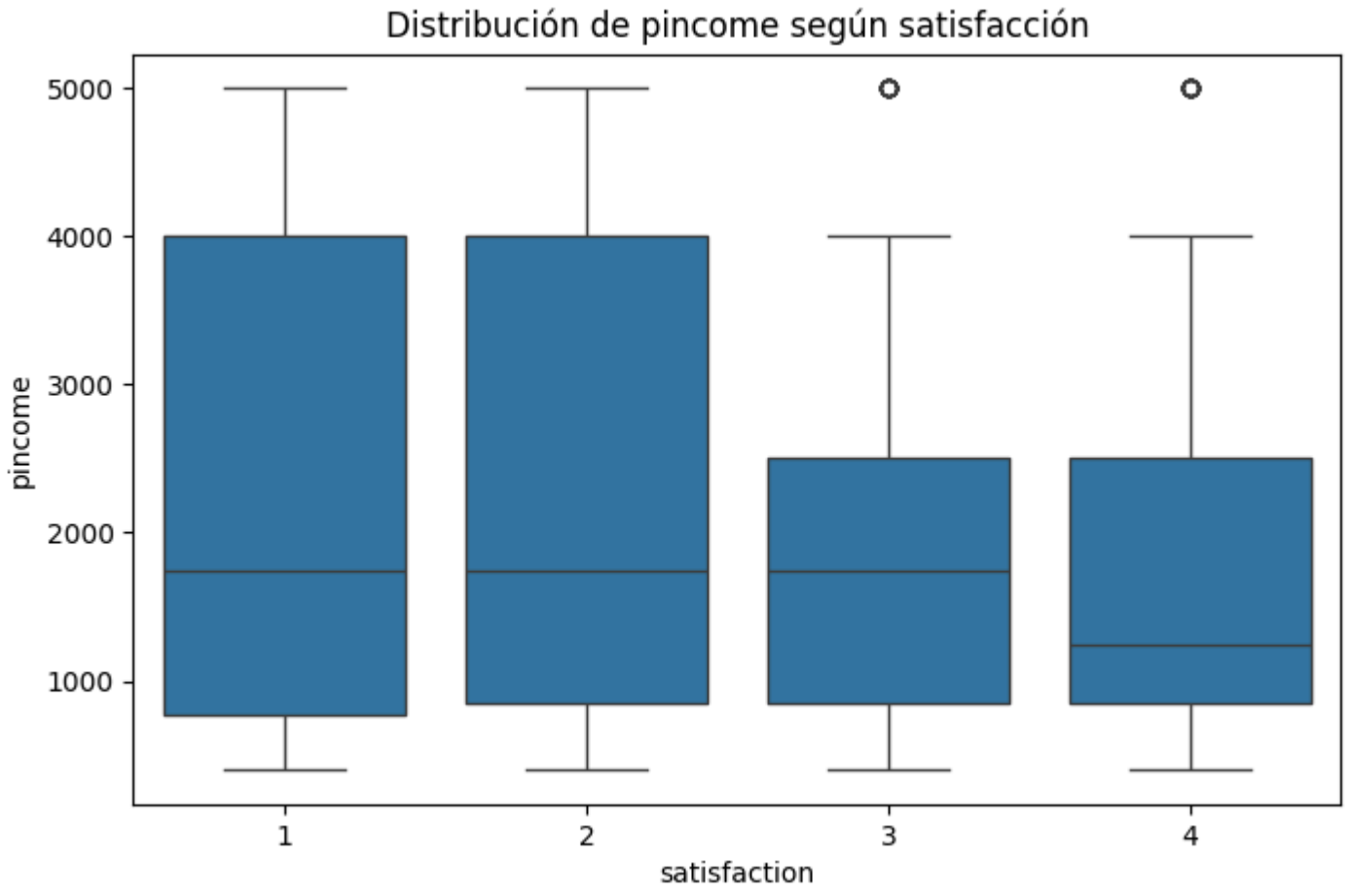
```python
# ======================================
# 6. ANÁLISIS EXPLORATORIO VISUAL
# ======================================

import seaborn as sns
import matplotlib.pyplot as plt

# Boxplots de variables continuas por nivel de satisfacción
for var in variables_continuas:
    plt.figure(figsize=(8, 5))
    sns.boxplot(x='satisfaction', y=var, data=df_analisis)
    plt.title(f'Distribución de {var} según satisfacción')
    plt.show()

# Gráficos de conteo de variables categóricas ordinales vs satisfacción
for var in variables_ordinales:
    plt.figure(figsize=(8, 5))
    sns.countplot(x=var, hue='satisfaction', data=df_analisis)
    plt.title(f'Conteo de satisfacción por nivel de {var}')
    plt.legend(title='Satisfacción')
    plt.show()

# Conteo de algunas variables binarias (por ejemplo, las primeras 5)
for var in variables_binarias[:4]:
    plt.figure(figsize=(6, 4))
    sns.countplot(x=var, data=df_analisis)
    plt.title(f'Conteo para {var}')
    plt.show()
```

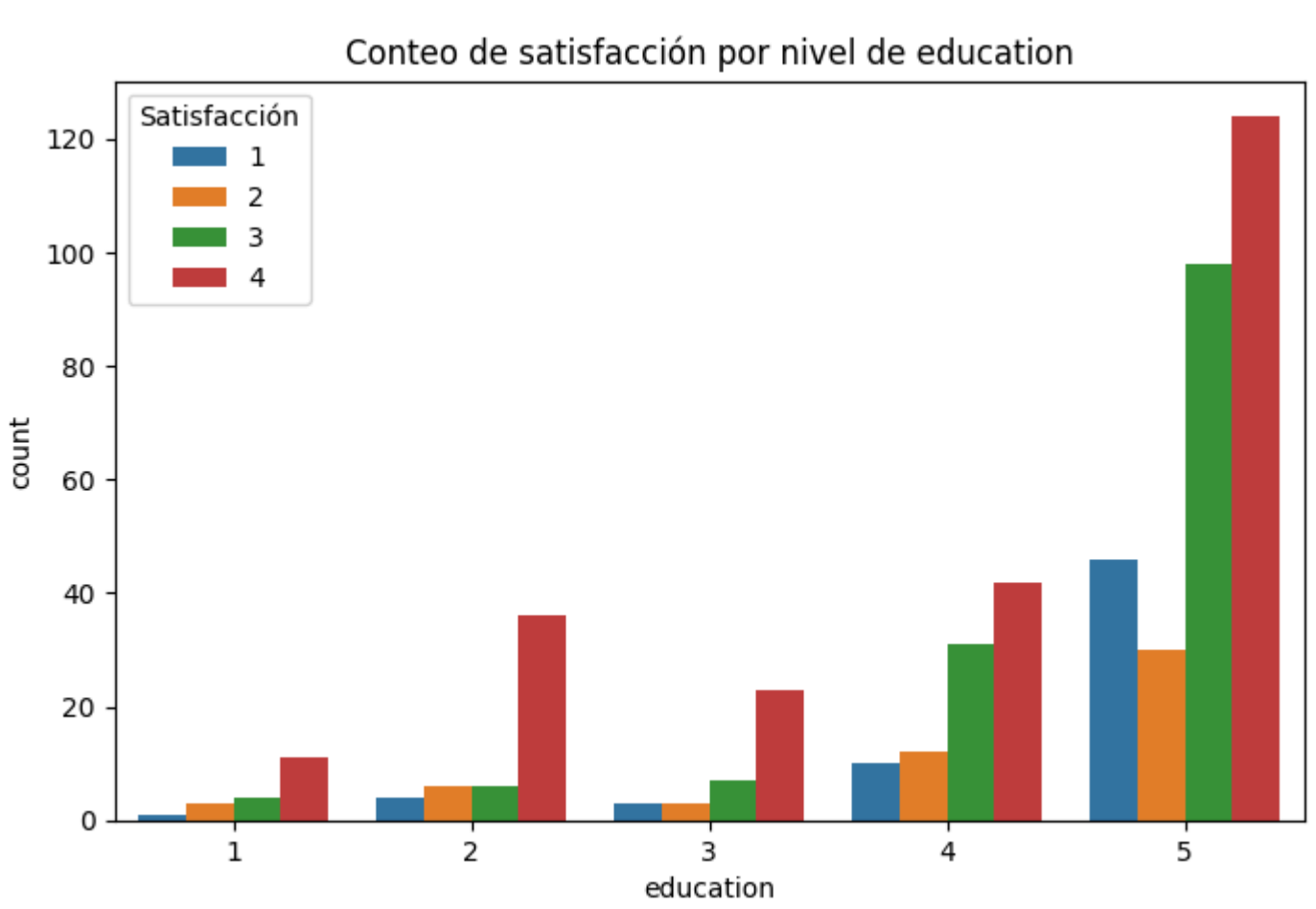
```python
## Conteo de datos de la variables satisfaccion

    plt.figure(figsize=(6, 4))
sns.countplot(x='satisfaction', data=df_analisis, order=sorted(df_analisis['satisfaction'].unique()))
plt.title('Conteo de satisfacción')
plt.xlabel('Nivel de satisfacción')
plt.ylabel('Frecuencia')
plt.show()
```

```
Downloading...
From: https://drive.google.com/uc?id=1WDJQQf-Y7NSFUptLEIiHRAAOfJAM8sYj
To: /content/datos.csv
100%|██████████| 50.0k/50.0k [00:00<00:00, 9.78MB/s]
```

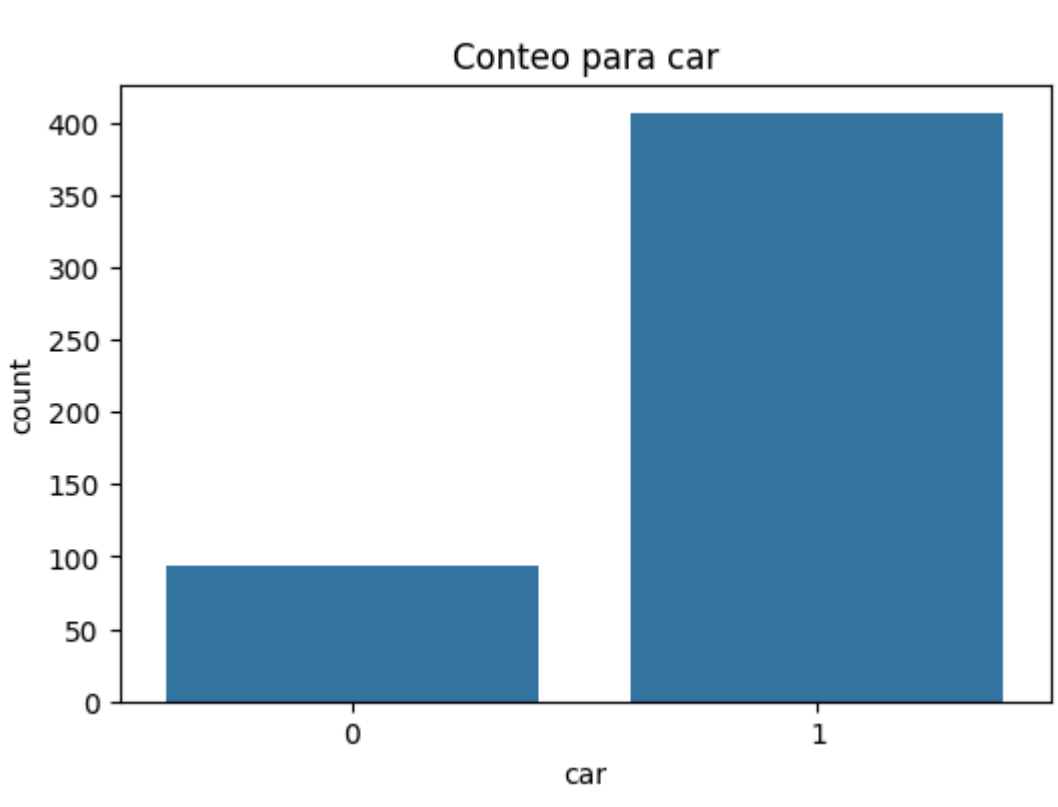
```
Variable respuesta: satisfaction
Variables numéricas: ['age', 'pincome']
Variables binarias: ['car', 'phone', 'fax', 'pc', 'savings1', 'savings2', 'creditcard1', 'creditcard2', 'atmcard1', 'atmcard2', 'cd1',
'cd2', 'specialchecking1', 'specialchecking2', 'auto.bill.payment1', 'auto.bill.payment2', 'personal.loans1', 'personal.loans2', 'mort
gage1', 'mortgage2', 'installment.loan1', 'installment.loan2', 'investment.fund1', 'investment.fund2', 'commodities.fund1', 'commoditi
es.fund2', 'annuities.fund1', 'annuities.fund2', 'car.insurance1', 'car.insurance2', 'home.insurance1', 'home.insurance2', 'life.insur
ance1', 'life.insurance2', 'gender_bin']
Variables ordinales: ['education']
Variables eliminadas: ['Unnamed: 0', 'id', 'gender']
Variables finales para los modelos: ['age', 'pincome', 'car', 'phone', 'fax', 'pc', 'savings1', 'savings2', 'creditcard1', 'creditcard
2', 'atmcard1', 'atmcard2', 'cd1', 'cd2', 'specialchecking1', 'specialchecking2', 'auto.bill.payment1', 'auto.bill.payment2', 'persona
l.loans1', 'personal.loans2', 'mortgage1', 'mortgage2', 'installment.loan1', 'installment.loan2', 'investment.fund1', 'investment.fund
2', 'commodities.fund1', 'commodities.fund2', 'annuities.fund1', 'annuities.fund2', 'car.insurance1', 'car.insurance2', 'home.insuranc
e1', 'home.insurance2', 'life.insurance1', 'life.insurance2', 'gender_bin', 'education']

Edades fuera de rango lógico (<0 o >100):
Empty DataFrame
Columns: [satisfaction, education, age, gender, car, phone, fax, pc, pincome, savings1, savings2, creditcard1, creditcard2, atmcard1,
atmcard2, cd1, cd2, specialchecking1, specialchecking2, auto.bill.payment1, auto.bill.payment2, personal.loans1, personal.loans2, mort
gage1, mortgage2, installment.loan1, installment.loan2, investment.fund1, investment.fund2, commodities.fund1, commodities.fund2, annu
ities.fund1, annuities.fund2, car.insurance1, car.insurance2, home.insurance1, home.insurance2, life.insurance1, life.insurance2, gend
er_bin]
Index: []

[0 rows x 40 columns]

Ingresos personales negativos:
Empty DataFrame
Columns: [satisfaction, education, age, gender, car, phone, fax, pc, pincome, savings1, savings2, creditcard1, creditcard2, atmcard1,
atmcard2, cd1, cd2, specialchecking1, specialchecking2, auto.bill.payment1, auto.bill.payment2, personal.loans1, personal.loans2, mort
gage1, mortgage2, installment.loan1, installment.loan2, investment.fund1, investment.fund2, commodities.fund1, commodities.fund2, annu
ities.fund1, annuities.fund2, car.insurance1, car.insurance2, home.insurance1, home.insurance2, life.insurance1, life.insurance2, gend
er_bin]
Index: []

[0 rows x 40 columns]

Casos con fax=1 pero phone=0 (inconsistencia):
Empty DataFrame
Columns: [satisfaction, education, age, gender, car, phone, fax, pc, pincome, savings1, savings2, creditcard1, creditcard2, atmcard1,
atmcard2, cd1, cd2, specialchecking1, specialchecking2, auto.bill.payment1, auto.bill.payment2, personal.loans1, personal.loans2, mort
gage1, mortgage2, installment.loan1, installment.loan2, investment.fund1, investment.fund2, commodities.fund1, commodities.fund2, annu
ities.fund1, annuities.fund2, car.insurance1, car.insurance2, home.insurance1, home.insurance2, life.insurance1, life.insurance2, gend
er_bin]
Index: []

[0 rows x 40 columns]

Casos con coche pero edad < 18:
Empty DataFrame
Columns: [satisfaction, education, age, gender, car, phone, fax, pc, pincome, savings1, savings2, creditcard1, creditcard2, atmcard1,
atmcard2, cd1, cd2, specialchecking1, specialchecking2, auto.bill.payment1, auto.bill.payment2, personal.loans1, personal.loans2, mort
gage1, mortgage2, installment.loan1, installment.loan2, investment.fund1, investment.fund2, commodities.fund1, commodities.fund2, annu
ities.fund1, annuities.fund2, car.insurance1, car.insurance2, home.insurance1, home.insurance2, life.insurance1, life.insurance2, gend
er_bin]
Index: []

[0 rows x 40 columns]
```



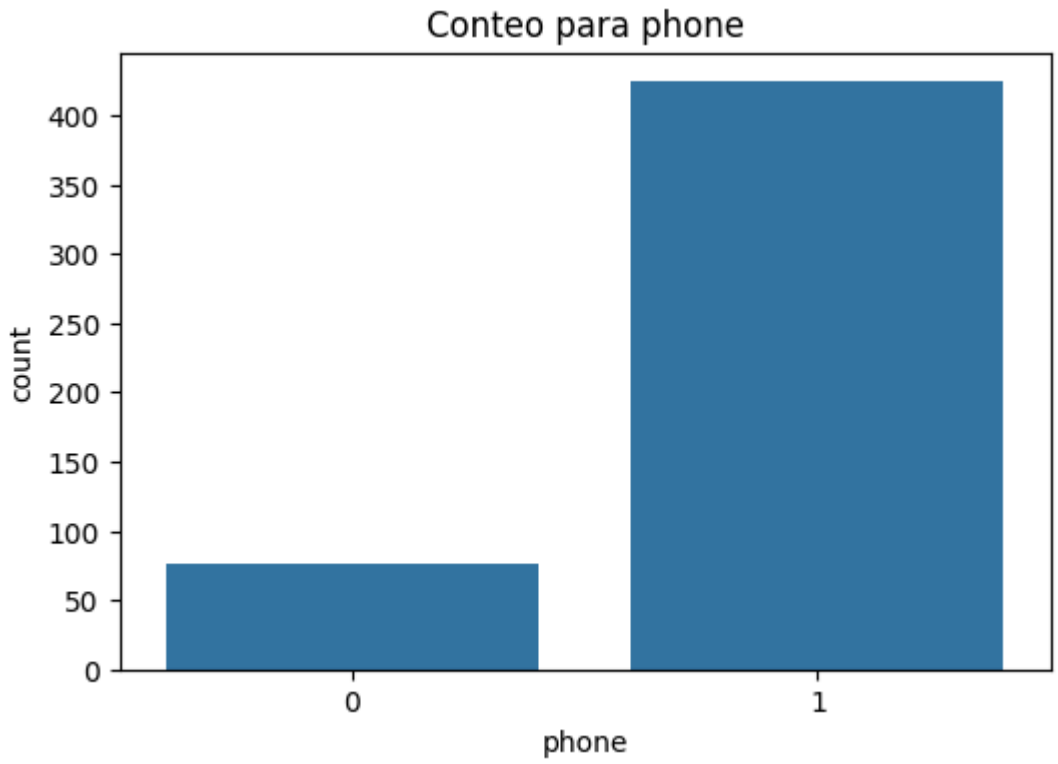




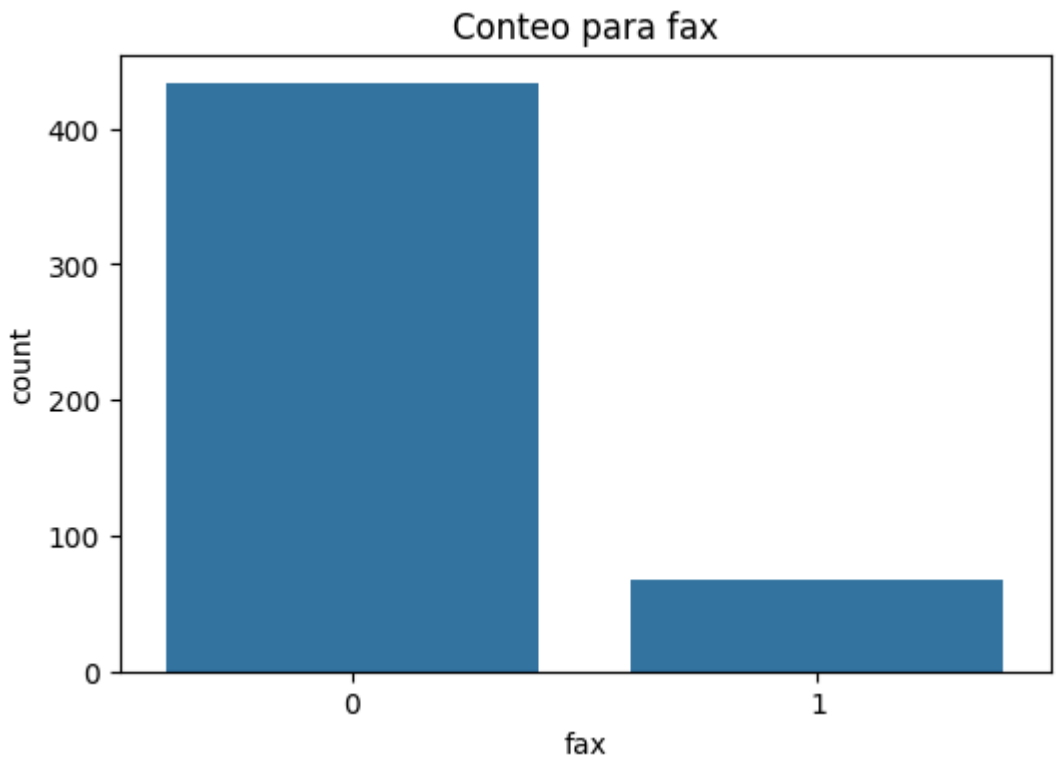
Conteo para car



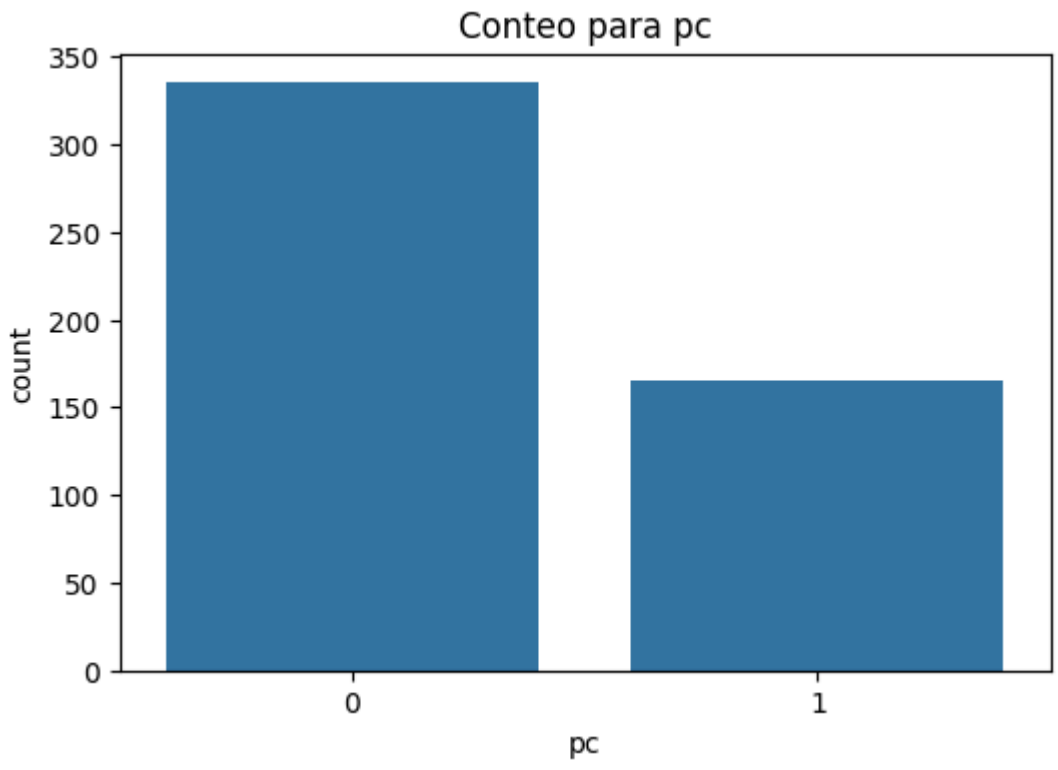
<Figure size 600x400 with 0 Axes>

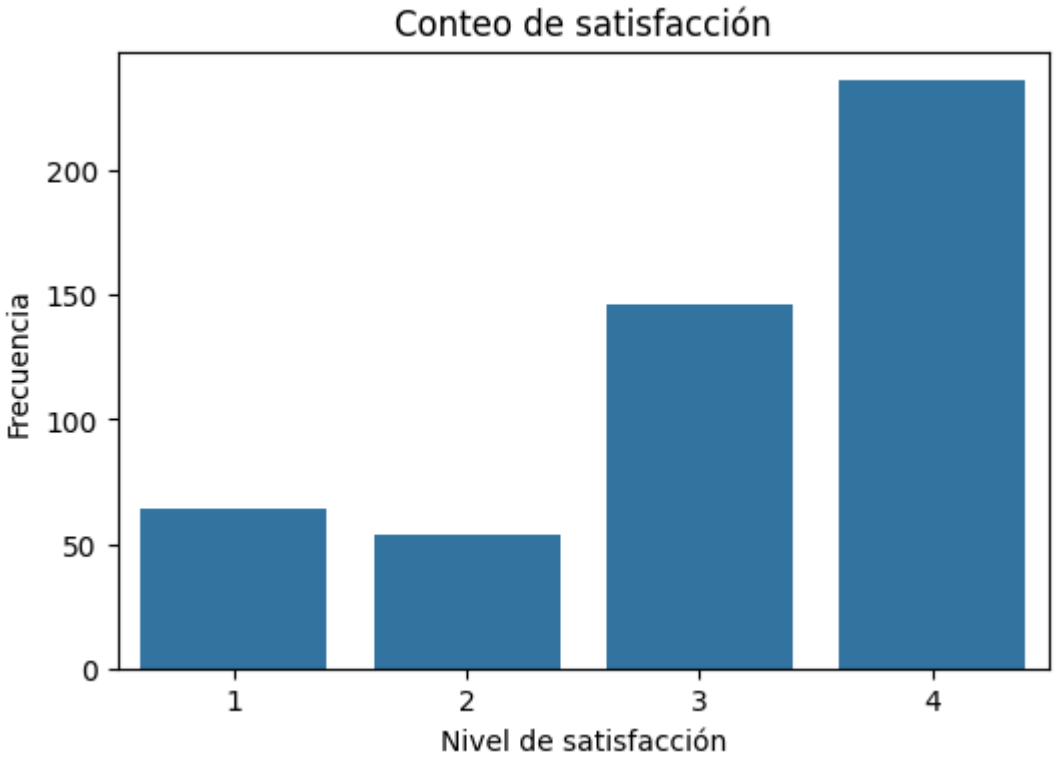


<Figure size 600x400 with 0 Axes>



<Figure size 600x400 with 0 Axes>





Luego de organizar las variables, se exploraron las relaciones entre ellas. Para las variables categóricas y binarias, se calculó el coeficiente de Cramér's V, que ayuda a identificar si hay asociaciones relevantes entre pares de variables. El resultado se visualizó con un mapa de calor, donde los colores más intensos señalaban mayor relación. Este análisis fue útil para notar algunas asociaciones esperadas, como entre productos financieros similares, pero en general no se encontraron relaciones excesivamente fuertes. En el caso de las variables numéricas como age e income, se usó la correlación de Pearson, y se observó que no tienen relación lineal entre sí, lo cual es positivo porque indica que ambas variables podrían aportar información distinta al modelo.

Después se revisó si algunas variables binarias tenían poca variación, es decir, si casi todos los clientes tenían el mismo valor. Esto puede ser un problema porque esas variables no ayudan a diferenciar entre niveles de satisfacción. Se encontró que variables como mortgage2 o installment.loan1 eran casos así porque más del 95% de los registros tenían el mismo valor, por lo que se eliminaron del análisis. También se evaluó si existían variables binarias muy parecidas entre sí (redundantes), pero no se detectaron combinaciones con una relación tan alta como para preocuparnos, así que la mayoría de estas variables se conservaron.

Con eso, se preparó el conjunto de datos final para los modelos. Se dejaron fuera las variables descartadas y se mantuvieron aquellas que realmente aportaban valor: variables numéricas, binarias filtradas, y la variable ordinal education. También se reemplazó la variable original de género por su versión binarizada. Finalmente, se dividieron los datos en dos conjuntos: uno para entrenar los modelos y otro para probarlos, asegurando que la proporción de niveles de satisfacción se mantuviera equilibrada en ambos. Esta preparación dejó todo listo para aplicar los algoritmos de clasificación y comenzar a analizar los resultados.

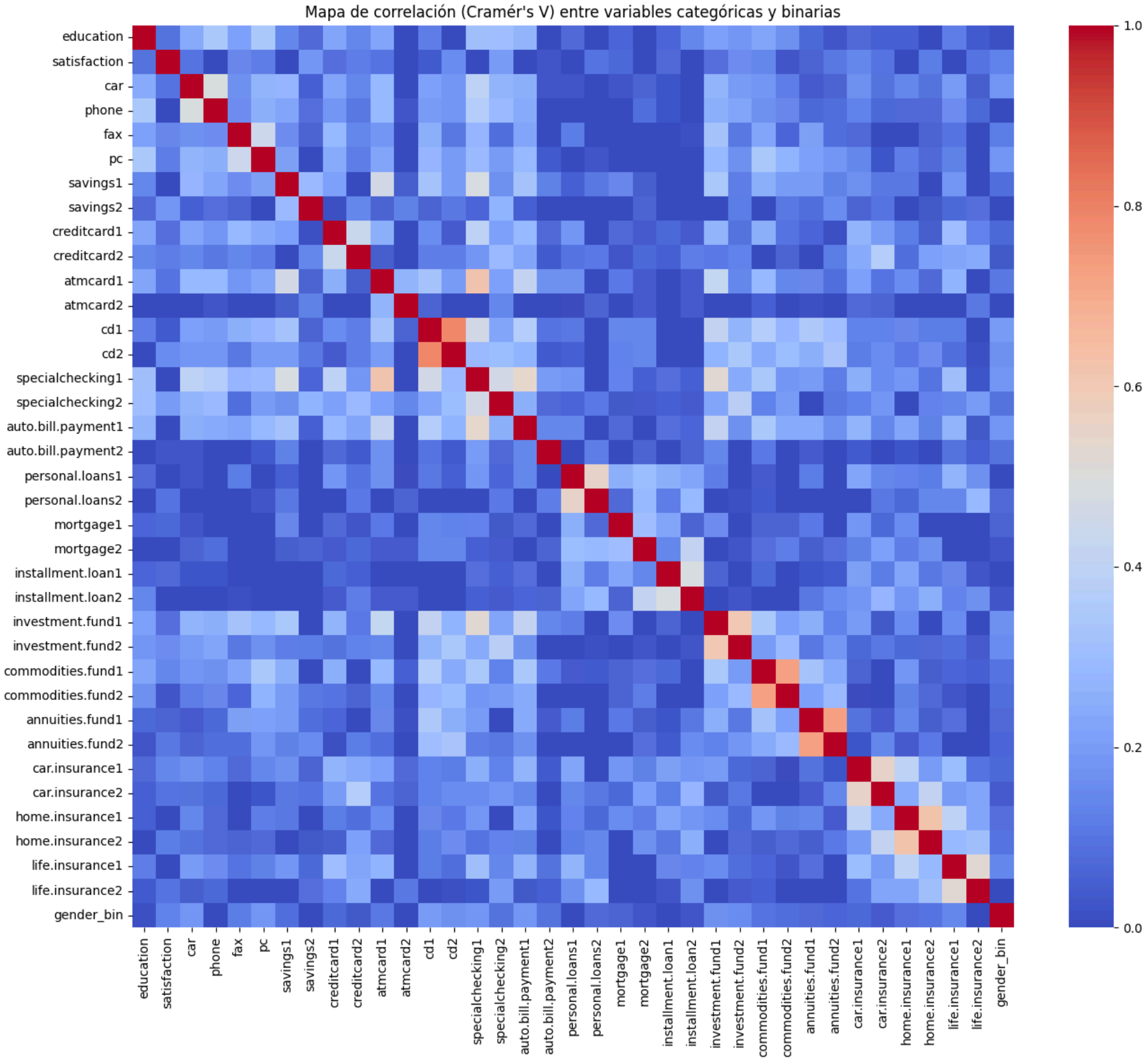
In [2]:
```
# ========================================
# LIBRERÍAS
# ========================================
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split

# ========================================
# FUNCIONES AUXILIARES
# ========================================

# Cálculo de Cramér's V
def cramers_v(x, y):
    confusion_matrix = pd.crosstab(x, y)
    chi2 = chi2_contingency(confusion_matrix, correction=False)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1)) / (n-1))
    rcorr = r - ((r-1)**2) / (n-1)
    kcorr = k - ((k-1)**2) / (n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

# ========================================
# MATRIZ DE CRAMÉR'S V (categorical + binarias)
# ========================================

# Variables categóricas y binarias
cat_vars = ['education', 'satisfaction'] + variables_binarias

# Matriz vacía
cramers_results = pd.DataFrame(index=cat_vars, columns=cat_vars)

# Cálculo de Cramér's V entre pares
for var1 in cat_vars:
    for var2 in cat_vars:
```

```
        try:
            cramers_results.loc[var1, var2] = cramers_v(df_analisis[var1], df_analisis[var2])
        except:
            cramers_results.loc[var1, var2] = np.nan

cramers_results = cramers_results.astype(float)

# Visualización
plt.figure(figsize=(14, 12))
sns.heatmap(cramers_results, annot=False, cmap='coolwarm', fmt=".2f")
plt.title("Mapa de correlación (Cramér's V) entre variables categóricas y binarias")
plt.tight_layout()
plt.show()


# Variables numéricas
num_vars = ['age', 'pincome']

# Matriz de correlación numérica (Pearson o Spearman)
num_corr = df_analisis[num_vars].corr(method='pearson')  # Cambia a 'spearman' si no hay linealidad

# Visualización
plt.figure(figsize=(6, 5))
sns.heatmap(num_corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Matriz de correlación entre variables numéricas")
plt.tight_layout()
```

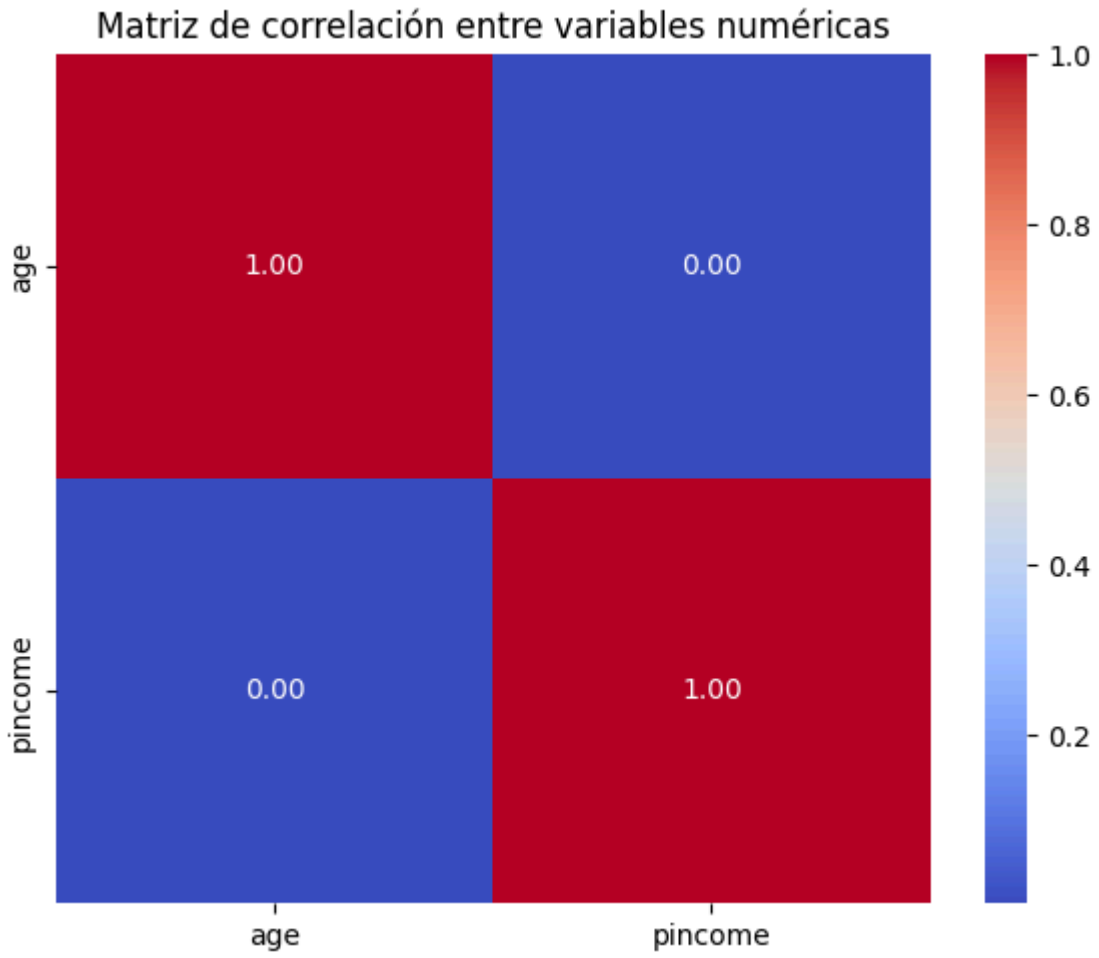
```
# ======================================
# FILTRO DE VARIABLES BINARIAS: VARIANZA CERCANA A CERO
# ======================================

# Detectar columnas binarias con varianza ≥ 95%
umbral_varianza = 0.95
cols_baja_varianza = [
    (col, df_analisis[col].value_counts(normalize=True).max())
    for col in variables_binarias
    if df_analisis[col].value_counts(normalize=True).max() >= umbral_varianza
]

if cols_baja_varianza:
    print("Columnas con varianza casi nula (≥ 95% del mismo valor):")
    for col, prop in cols_baja_varianza:
        print(f"  - {col}: proporción = {prop:.3f}")
else:
    print(" No se detectaron columnas binarias con varianza ≥ 0.95.\n")

# Variables con varianza casi nula a eliminar manualmente
cols_ruido = ['mortgage2', 'installment.loan1', 'installment.loan2']

# nuevo DataFrame sin esas variables
df_modelo = df_analisis.drop(columns=cols_ruido)

# Variables binarias finales conservadas
binarias_finales = [col for col in variables_binarias if col in df_modelo.columns]
print("\n Variables binarias conservadas tras limpieza:")
print(binarias_finales)

# ======================================
```

```
# DETECCIÓN DE BINARIAS REDUNDANTES (Cramér’s V ≥ 0.85)
# ======================================
redundantes = []
for i, col1 in enumerate(binarias_finales):
    for col2 in binarias_finales[i+1:]:
        v = cramers_v(df_modelo[col1], df_modelo[col2])
        if v >= 0.85:
            redundantes.append((col1, col2, round(v, 3)))

if redundantes:
    print("\n Pares de variables binarias con asociación muy alta (Cramér's V ≥ 0.85):")
    for c1, c2, v in redundantes:
        print(f"  - {c1} ↔ {c2}: V = {v}")
else:
    print("\n No se detectaron pares de variables binarias con Cramér's V ≥ 0.85.")

# ======================================
# SPLIT FINAL PARA MODELADO
# ======================================

# Variables predictoras finales
X = df_modelo.drop(columns=['satisfaction', 'gender'])  # gender ya reemplazada por gender_bin
y = df_modelo['satisfaction']

print("\n Columnas finales en X (predictoras):")
print(X.columns.tolist())

# División de datos en entrenamiento y prueba
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.30,
    random_state=40,
    stratify=y
)
```




```
Columnas con varianza casi nula (≥ 95% del mismo valor):
  - mortgage2: proporción = 0.986
  - installment.loan1: proporción = 0.958
  - installment.loan2: proporción = 0.986

 Variables binarias conservadas tras limpieza:
['car', 'phone', 'fax', 'pc', 'savings1', 'savings2', 'creditcard1', 'creditcard2', 'atmcard1', 'atmcard2', 'cd1', 'cd2', 'specialchec
king1', 'specialchecking2', 'auto.bill.payment1', 'auto.bill.payment2', 'personal.loans1', 'personal.loans2', 'mortgage1', 'investmen
t.fund1', 'investment.fund2', 'commodities.fund1', 'commodities.fund2', 'annuities.fund1', 'annuities.fund2', 'car.insurance1', 'car.i
nsurance2', 'home.insurance1', 'home.insurance2', 'life.insurance1', 'life.insurance2', 'gender_bin']

 No se detectaron pares de variables binarias con Cramér's V ≥ 0.85.

 Columnas finales en X (predictoras):
['education', 'age', 'car', 'phone', 'fax', 'pc', 'pincome', 'savings1', 'savings2', 'creditcard1', 'creditcard2', 'atmcard1', 'atmcar
d2', 'cd1', 'cd2', 'specialchecking1', 'specialchecking2', 'auto.bill.payment1', 'auto.bill.payment2', 'personal.loans1', 'personal.lo
ans2', 'mortgage1', 'investment.fund1', 'investment.fund2', 'commodities.fund1', 'commodities.fund2', 'annuities.fund1', 'annuities.fu
nd2', 'car.insurance1', 'car.insurance2', 'home.insurance1', 'home.insurance2', 'life.insurance1', 'life.insurance2', 'gender_bin']
```



Análisis de Modelos de Clasificación para Predecir la Satisfacción del Cliente

---

Modelos de Árboles de Decisión

El modelo de árbol de decisión básico presentó un rendimiento limitado, con una precisión de 0.42. Su mayor acierto fue en la clase 4, correspondiente a clientes muy satisfechos, con 39 predicciones correctas. Sin embargo, cometió muchos errores al clasificar las clases 1 y 2, asociadas a menor satisfacción. Esta tendencia se explica principalmente por el desbalance en los datos, ya que la mayoría de las observaciones pertenecían a la clase 4. Aunque se respetaron los tipos de variables (numéricas, ordinales y binarias), el modelo no logró captar adecuadamente los patrones de las clases menos representadas.

Al aplicar poda al árbol utilizando un valor de ccp_alpha optimizado por validación cruzada, la precisión general mejoró a 0.50. Este ajuste permitió una mejor clasificación de las clases 3 y 4, aunque la clase 2 continuó siendo mal identificada (0 aciertos). En este modelo, la variable edad destacó como uno de los predictores más relevantes, junto con algunas binarias como fax y fondos de inversión. Esto evidenció que ciertas variables numéricas y binarias específicas aportaban más valor que otras al momento de segmentar los datos.

Para mejorar la equidad entre clases, se entrenó un modelo con pesos balanceados (class_weight='balanced'), lo que redistribuyó el foco del modelo hacia las clases menos frecuentes. Si bien la precisión bajó a 0.35, se observó una mayor sensibilidad hacia la clase 2 y un reparto más uniforme de errores. No obstante, al introducir más variables con menor importancia, se generó cierto ruido que pudo afectar negativamente el rendimiento. Aun así, variables como ingreso, edad y educación mantuvieron su influencia.

El modelo final combinó tres estrategias: poda, balanceo de clases y uso exclusivo de variables con importancia significativa. Aunque la precisión global se mantuvo en 0.35, el modelo fue más claro, interpretativo y coherente. Conservó variables clave como ingreso, edad, educación y algunas binarias relevantes como auto.bill.payment2 y savings2, lo que permitió un enfoque más limpio y orientado a la comprensión del problema.

Más allá de las métricas generales, el análisis de las matrices de confusión reveló diferencias importantes entre modelos. El árbol básico, por ejemplo, confundió con frecuencia las clases 3 y 4, reflejando un sesgo hacia la clase mayoritaria. El árbol podado corrigió parcialmente este problema, aunque sacrificó completamente la detección de la clase 2. El modelo balanceado logró una distribución de errores más equitativa, mejorando la detección de clases menos frecuentes, aunque con una pérdida de precisión general. Estas diferencias muestran cómo el tratamiento del desbalance de clases influye directamente en la capacidad del modelo para representar de manera justa todas las categorías de satisfacción.

En cuanto al aporte de las variables, las numéricas como edad e ingreso fueron las más consistentes en todos los modelos, al permitir divisiones naturales que facilitan la segmentación. Por ejemplo, edad se posicionó frecuentemente como nodo raíz, indicando su fuerte relación con la variable respuesta. Las variables ordinales, como educación, resultaron también muy útiles cuando se respetó su jerarquía interna. Esto permitió al árbol generar divisiones más coherentes, mejorando su capacidad de generalización.

Las variables binarias tuvieron un comportamiento más variados. Algunas como fax, auto.bill.payment2 y savings2 mostraron utilidad y se mantuvieron en los modelos más depurados. Otras, sin embargo, fueron descartadas por tener baja varianza o escasa relación con la variable objetivo, lo cual contribuyó a reducir el ruido y mejorar la interpretabilidad del modelo. Una ventaja importante de los árboles de decisión es que pueden manejar sin problemas este tipo mixto de variables, sin necesidad de transformaciones adicionales.

En conclusión, la correcta clasificación y codificación de las variables desde el inicio fue fundamental para construir modelos más interpretables y efectivos. El uso combinado de poda, balanceo de clases y selección de variables importantes permitió desarrollar modelos más equilibrados, aún en presencia de datos desbalanceados. Aunque las métricas como accuracy o f1-score no alcanzaron niveles altos, el proceso permitió identificar factores

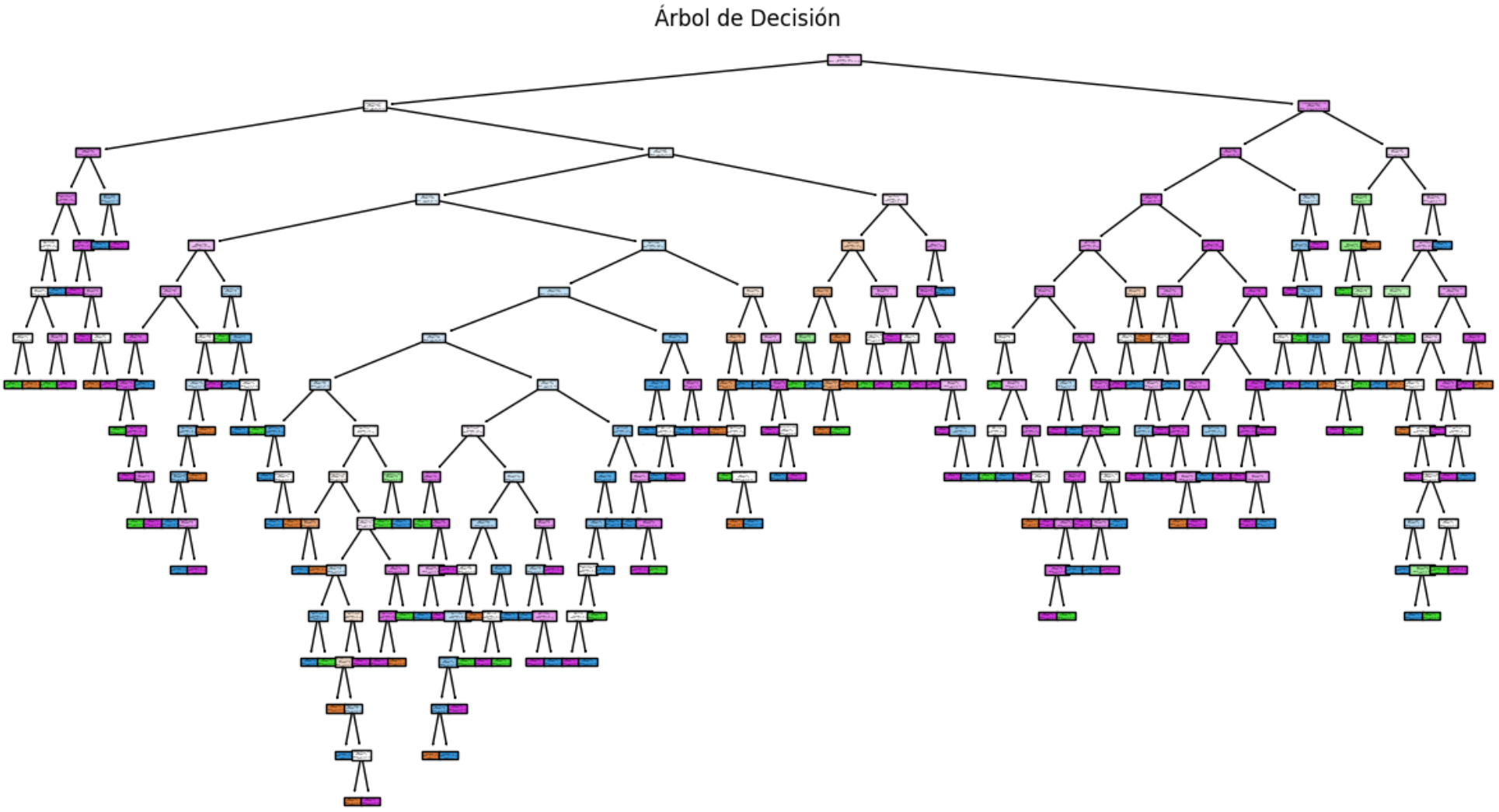
clave que influyen en la satisfacción de los clientes bancarios y establecer una base sólida para futuros modelos con algoritmos más robustos o ajustes más refinados.

In [3]:
```
# ================================================
# 1. Importación de librerías
# ================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay,
    classification_report, accuracy_score,
    f1_score, cohen_kappa_score
)

# ================================================
# 2. Preparación de datos
# ================================================
# Reutilizamos X_train, X_test, y_train, y_test, df_modelo, X, y
class_names = [str(cls) for cls in sorted(y.unique())]

# ================================================
# Función auxiliar para evaluar modelos
# ================================================
def evaluar_modelo(nombre, modelo, X_test, y_test, y_pred, cmap="Blues"):
    print(f"== {nombre} ==")
    print(classification_report(y_test, y_pred))
    print("Kappa:", cohen_kappa_score(y_test, y_pred))
    ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap=cmap)
    plt.title(f"Matriz de Confusión - {nombre}")
    plt.show()

# ================================================
# 3. Árbol de decisión básico
# ================================================
tree_model = DecisionTreeClassifier(random_state=40)
tree_model.fit(X_train, y_train)
y_pred_tree = tree_model.predict(X_test)

print(f'Accuracy árbol de decisión: {tree_model.score(X_test, y_test):.2f}')
plt.figure(figsize=(15, 8))
plot_tree(tree_model, filled=True, feature_names=X.columns, class_names=class_names)
plt.title("Árbol de Decisión")
plt.show()

evaluar_modelo("Árbol de Decisión", tree_model, X_test, y_test, y_pred_tree, cmap="Blues")

# ================================================
# 4. Árbol de decisión podado (búsqueda de ccp_alpha)
# ================================================
path = tree_model.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas

cv_scores = [
    cross_val_score(DecisionTreeClassifier(random_state=40, ccp_alpha=alpha),
                    X_train, y_train, cv=5).mean()
    for alpha in ccp_alphas
]

best_index = np.argmax(cv_scores)
best_ccp_alpha = ccp_alphas[best_index]

plt.figure(figsize=(8, 5))
plt.plot(ccp_alphas, cv_scores, marker='o', drawstyle="steps-post")
plt.xlabel("ccp_alpha")
plt.ylabel("Accuracy promedio CV")
plt.title("Selección de ccp_alpha con validación cruzada")
plt.grid(True)
plt.tight_layout()
plt.show()

print(f"Mejor ccp_alpha: {best_ccp_alpha:.5f} con accuracy CV: {cv_scores[best_index]:.4f}")

pruned_tree_model = DecisionTreeClassifier(random_state=40, ccp_alpha=best_ccp_alpha)
pruned_tree_model.fit(X_train, y_train)
y_pred_pruned = pruned_tree_model.predict(X_test)

plt.figure(figsize=(15, 8))
plot_tree(pruned_tree_model, filled=True, feature_names=X.columns, class_names=class_names)
plt.title(f"Árbol Podado (ccp_alpha={best_ccp_alpha:.5f})")
plt.show()
```

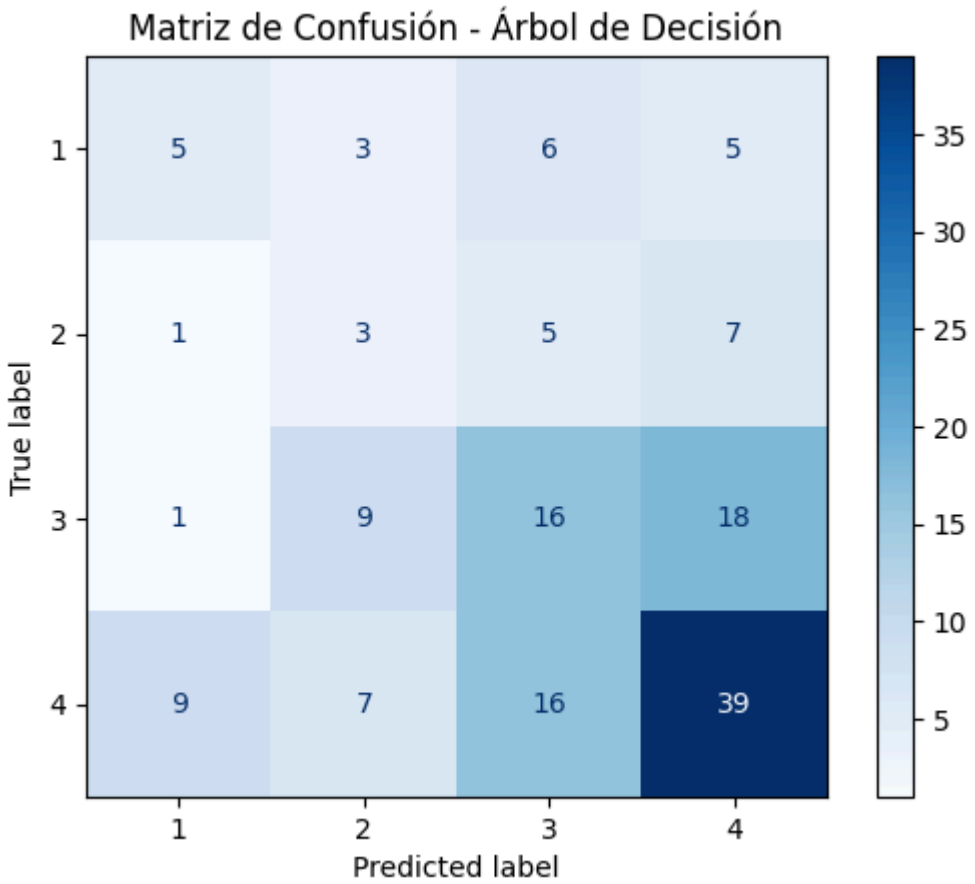
```python
evaluar_modelo("Árbol Podado", pruned_tree_model, X_test, y_test, y_pred_pruned, cmap="Purples")

# Importancia de variables - Árbol Podado
importances_tree = pruned_tree_model.feature_importances_
tree_importance_df = pd.DataFrame({
    'Variable': X.columns,
    'Importancia': importances_tree
}).sort_values(by='Importancia', ascending=False)

top_n = 20
plt.figure(figsize=(12, 6))
sns.barplot(data=tree_importance_df.head(top_n), x='Importancia', y='Variable')
plt.title(f"Top {top_n} Variables más Importantes - Árbol Podado")
plt.tight_layout()
plt.show()

# ============================================
# 5. Árbol de decisión balanceado
# ============================================
balanced_tree = DecisionTreeClassifier(random_state=40, class_weight='balanced')
balanced_tree.fit(X_train, y_train)
y_pred_bal = balanced_tree.predict(X_test)

evaluar_modelo("Árbol de Decisión Balanceado", balanced_tree, X_test, y_test, y_pred_bal, cmap="Greens")

# ============================================
# 6. Filtrar variables con importancia significativa (≥ 0.01)
# ============================================
importance_df = pd.DataFrame({
    'Variable': X.columns,
    'Importancia': balanced_tree.feature_importances_
}).sort_values(by='Importancia', ascending=False)

filtered_vars = importance_df[importance_df['Importancia'] >= 0.01]['Variable']
X_train_filtered = X_train[filtered_vars]
X_test_filtered = X_test[filtered_vars]

plt.figure(figsize=(10, 6))
sns.barplot(data=importance_df[importance_df['Importancia'] >= 0.01], x='Importancia', y='Variable')
plt.title("Variables con Importancia ≥ 0.01 - Árbol Balanceado")
plt.tight_layout()
plt.show()

# ============================================
# 7. Árbol podado con variables filtradas y pesos balanceados
# ============================================
tree_temp = DecisionTreeClassifier(random_state=40, class_weight='balanced')
tree_temp.fit(X_train_filtered, y_train)

path = tree_temp.cost_complexity_pruning_path(X_train_filtered, y_train)
ccp_alphas_f = path.ccp_alphas

cv_scores_f = [
    cross_val_score(DecisionTreeClassifier(random_state=40, ccp_alpha=alpha, class_weight='balanced'),
                    X_train_filtered, y_train, cv=5).mean()
    for alpha in ccp_alphas_f
]

best_index_f = np.argmax(cv_scores_f)
best_ccp_alpha_f = ccp_alphas_f[best_index_f]

final_model = DecisionTreeClassifier(random_state=40, ccp_alpha=best_ccp_alpha_f, class_weight='balanced')
final_model.fit(X_train_filtered, y_train)
y_pred_final = final_model.predict(X_test_filtered)

evaluar_modelo("Árbol Podado con Variables Importantes y Pesos Balanceados", final_model,
               X_test_filtered, y_test, y_pred_final, cmap="Oranges")


# ============================================
# Importancia de variables - Árbol Final Filtrado y Balanceado
# ============================================
final_importance_df = pd.DataFrame({
    'Variable': X_train_filtered.columns,
    'Importancia': final_model.feature_importances_
}).sort_values(by='Importancia', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(data=final_importance_df, x='Importancia', y='Variable')
plt.title("Importancia de Variables - Árbol Podado con Variables Importantes y Pesos Balanceados")
plt.tight_layout()
plt.show()



# ============================================
# 8. Almacenamiento de métricas clave
```

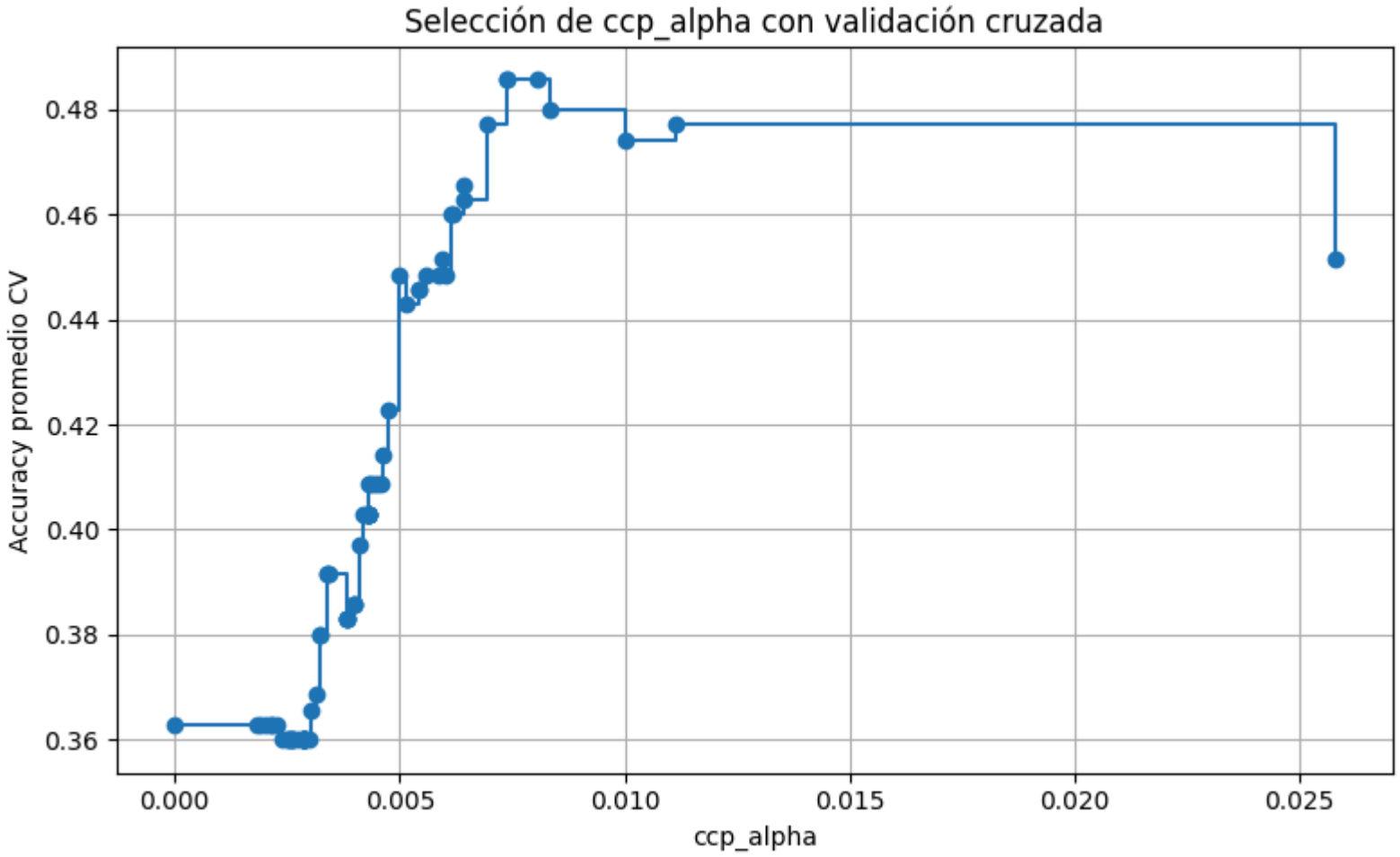
```python
# ============================================
from collections import OrderedDict

results = OrderedDict({
    "accuracy_arbol_basico": accuracy_score(y_test, y_pred_tree),
    "f1_arbol_basico": f1_score(y_test, y_pred_tree, average='weighted'),
    "kappa_arbol_basico": cohen_kappa_score(y_test, y_pred_tree),

    "accuracy_arbol_podado": accuracy_score(y_test, y_pred_pruned),
    "f1_arbol_podado": f1_score(y_test, y_pred_pruned, average='weighted'),
    "kappa_arbol_podado": cohen_kappa_score(y_test, y_pred_pruned),

    "accuracy_arbol_balanceado": accuracy_score(y_test, y_pred_bal),
    "f1_arbol_balanceado": f1_score(y_test, y_pred_bal, average='weighted'),
    "kappa_arbol_balanceado": cohen_kappa_score(y_test, y_pred_bal),

    "accuracy_arbol_importante_balanceado": accuracy_score(y_test, y_pred_final),
    "f1_arbol_importante_balanceado": f1_score(y_test, y_pred_final, average='weighted'),
    "kappa_arbol_importante_balanceado": cohen_kappa_score(y_test, y_pred_final)
})


print("\n Resultados almacenados:")
for metric, value in results.items():
    print(f"{metric}: {value:.4f}")
```

```
Accuracy árbol de decisión: 0.42
```

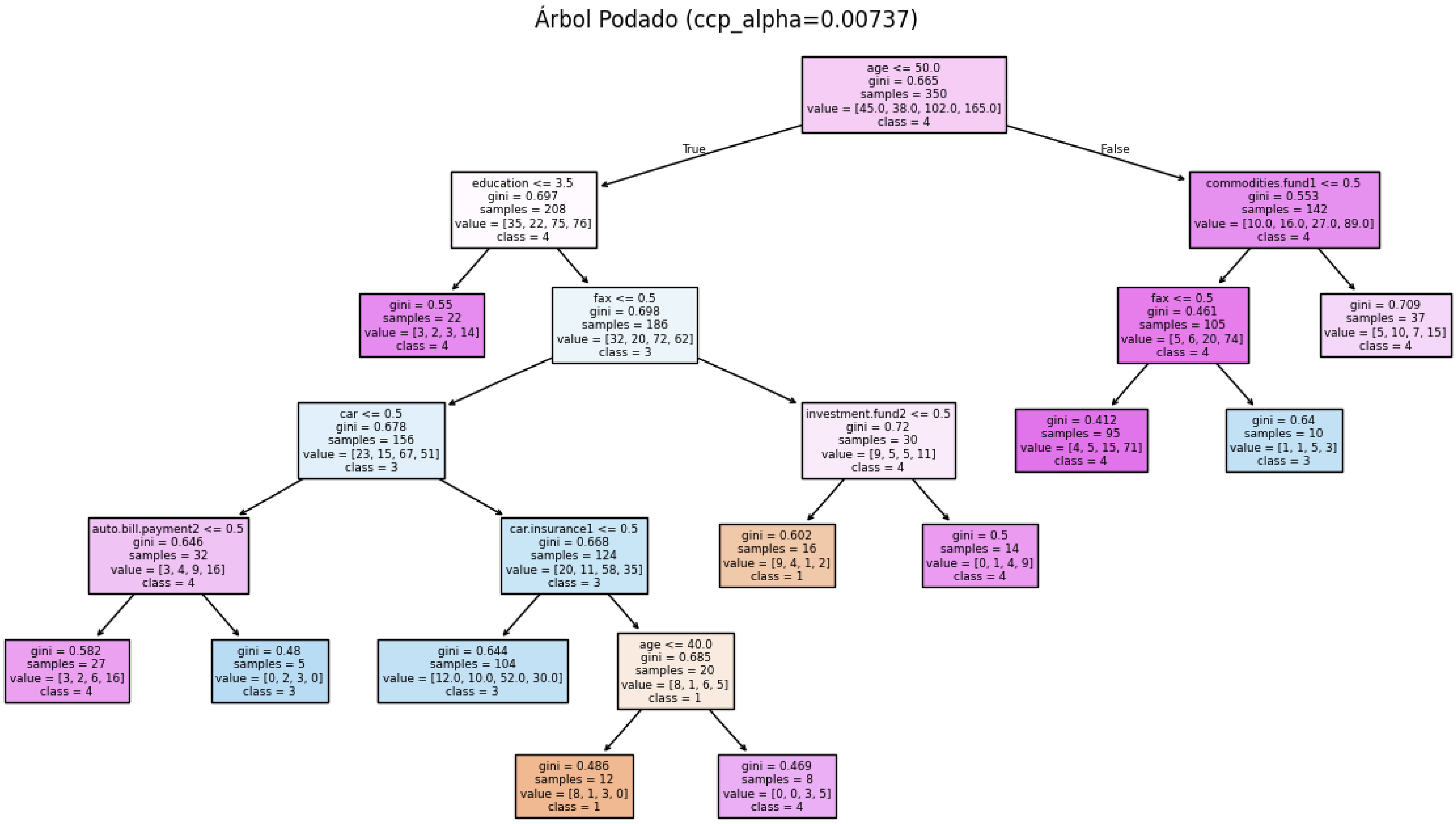



```
== Árbol de Decisión ==
              precision    recall  f1-score   support

           1       0.31      0.26      0.29        19
           2       0.14      0.19      0.16        16
           3       0.37      0.36      0.37        44
           4       0.57      0.55      0.56        71

    accuracy                           0.42       150
   macro avg       0.35      0.34      0.34       150
weighted avg       0.43      0.42      0.42       150

Kappa: 0.1330631767753937
```



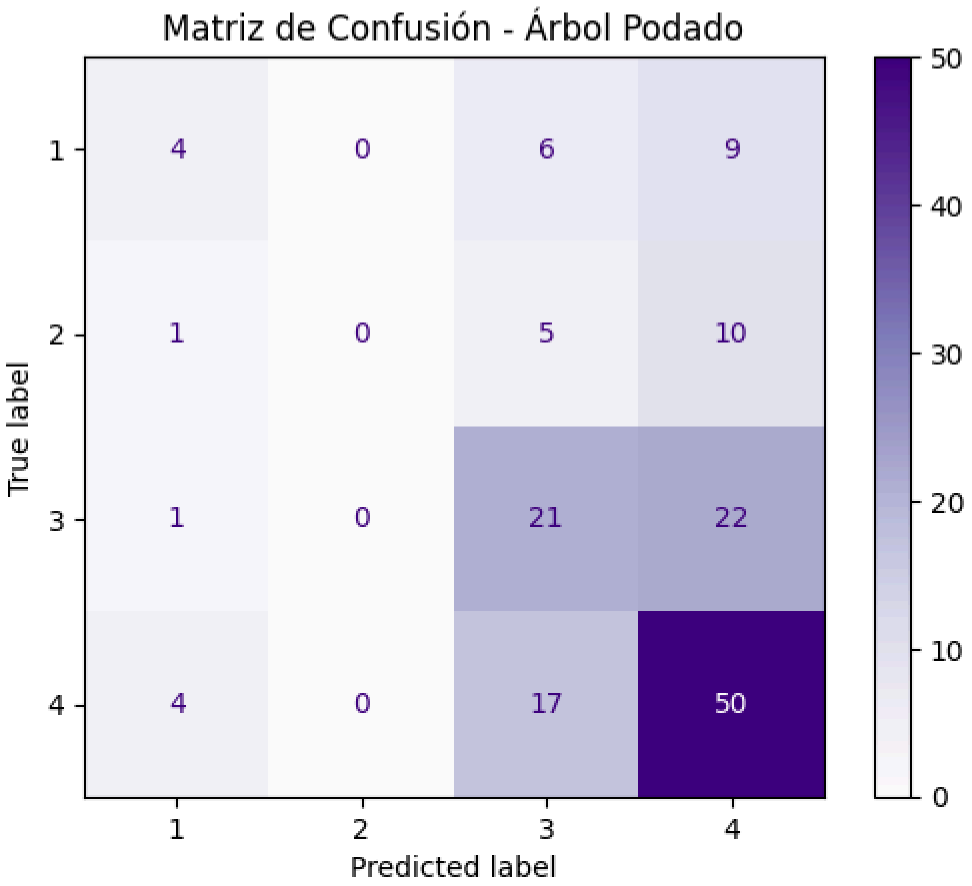


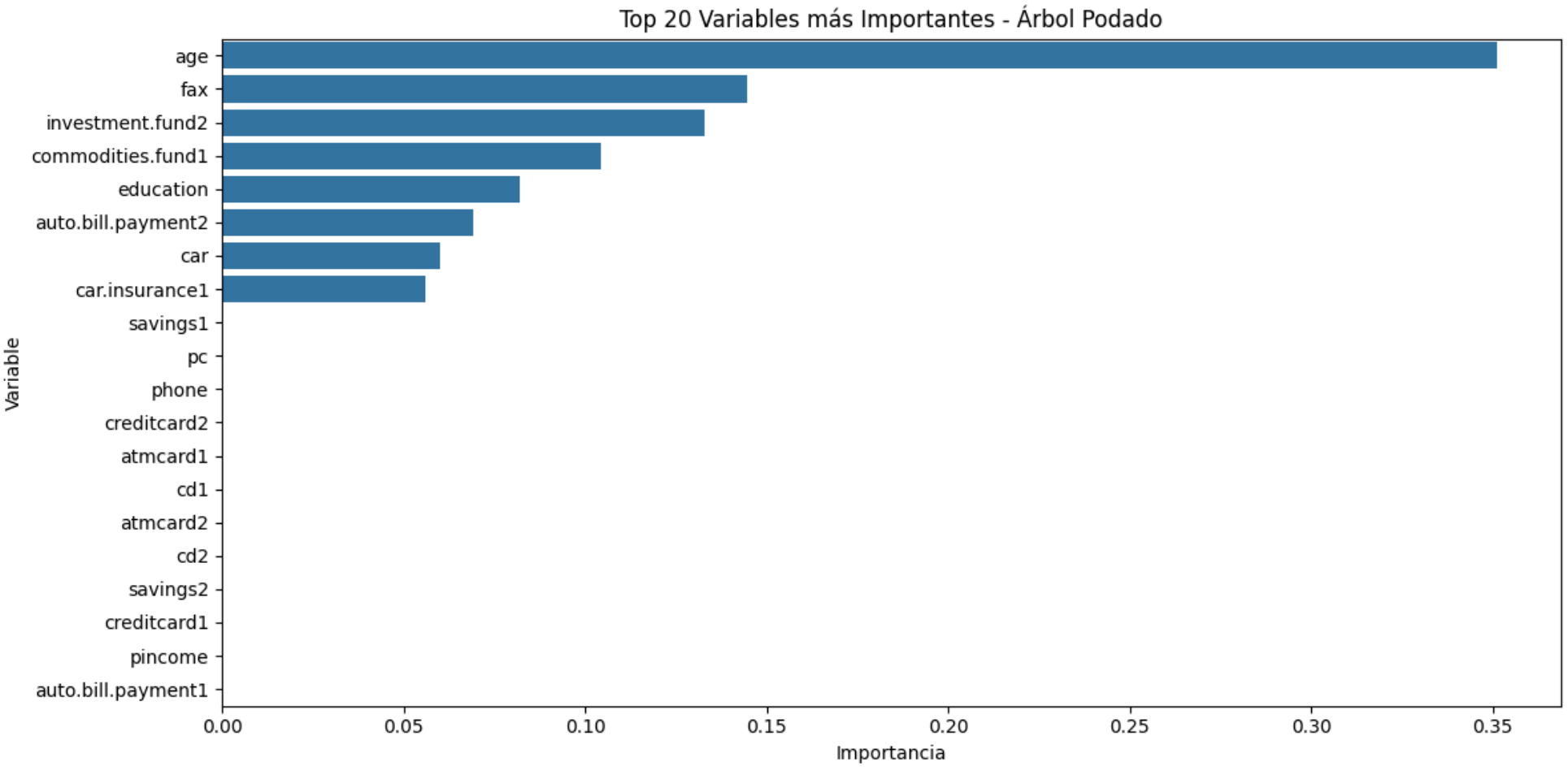

```
Mejor ccp_alpha: 0.00737 con accuracy CV: 0.4857
```




```
== Árbol Podado ==
              precision    recall  f1-score   support

           1       0.40      0.21      0.28        19
           2       0.00      0.00      0.00        16
           3       0.43      0.48      0.45        44
           4       0.55      0.70      0.62        71

    accuracy                           0.50       150
   macro avg       0.34      0.35      0.34       150
weighted avg       0.44      0.50      0.46       150

Kappa: 0.17841232746658875
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

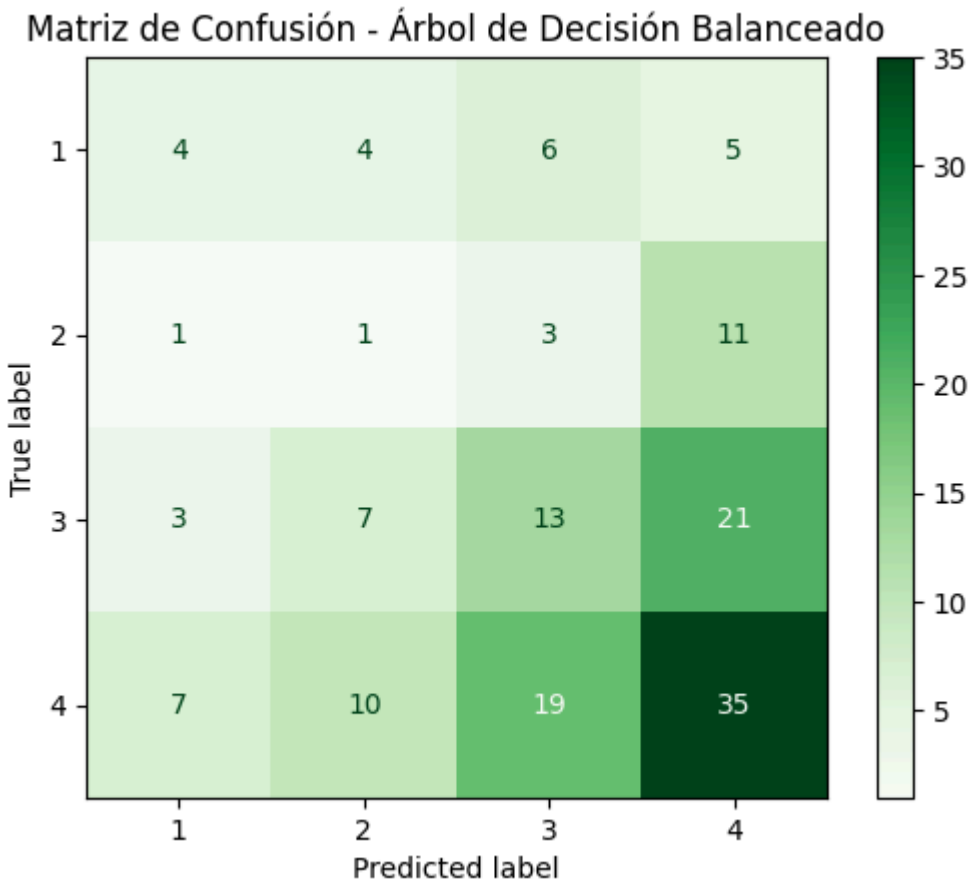


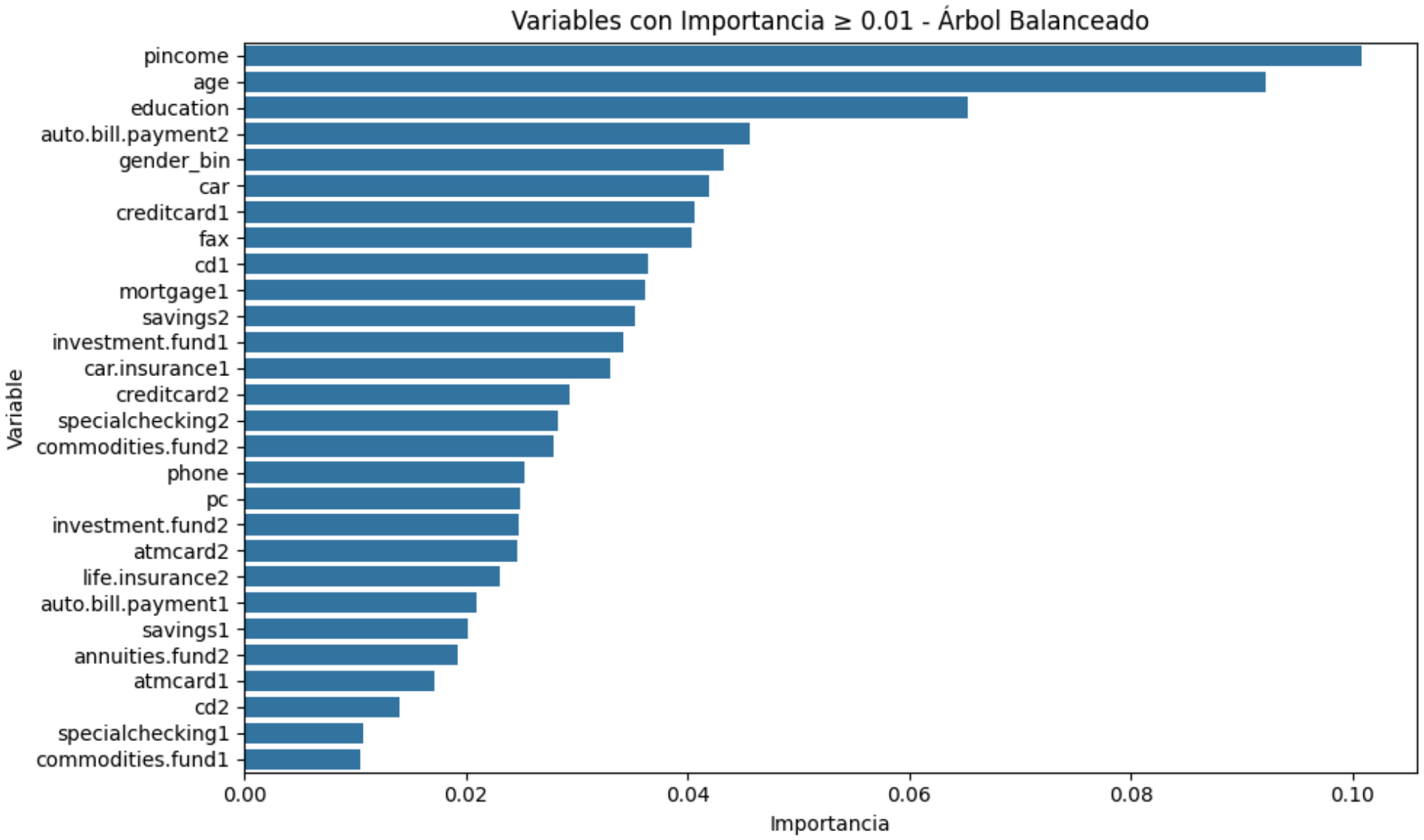


== Árbol de Decisión Balanceado ==

	precision	recall	f1-score	support
1	0.27	0.21	0.24	19
2	0.05	0.06	0.05	16
3	0.32	0.30	0.31	44
4	0.49	0.49	0.49	71
accuracy			0.35	150
macro avg	0.28	0.27	0.27	150
weighted avg	0.36	0.35	0.36	150

Kappa: 0.0265605138154813

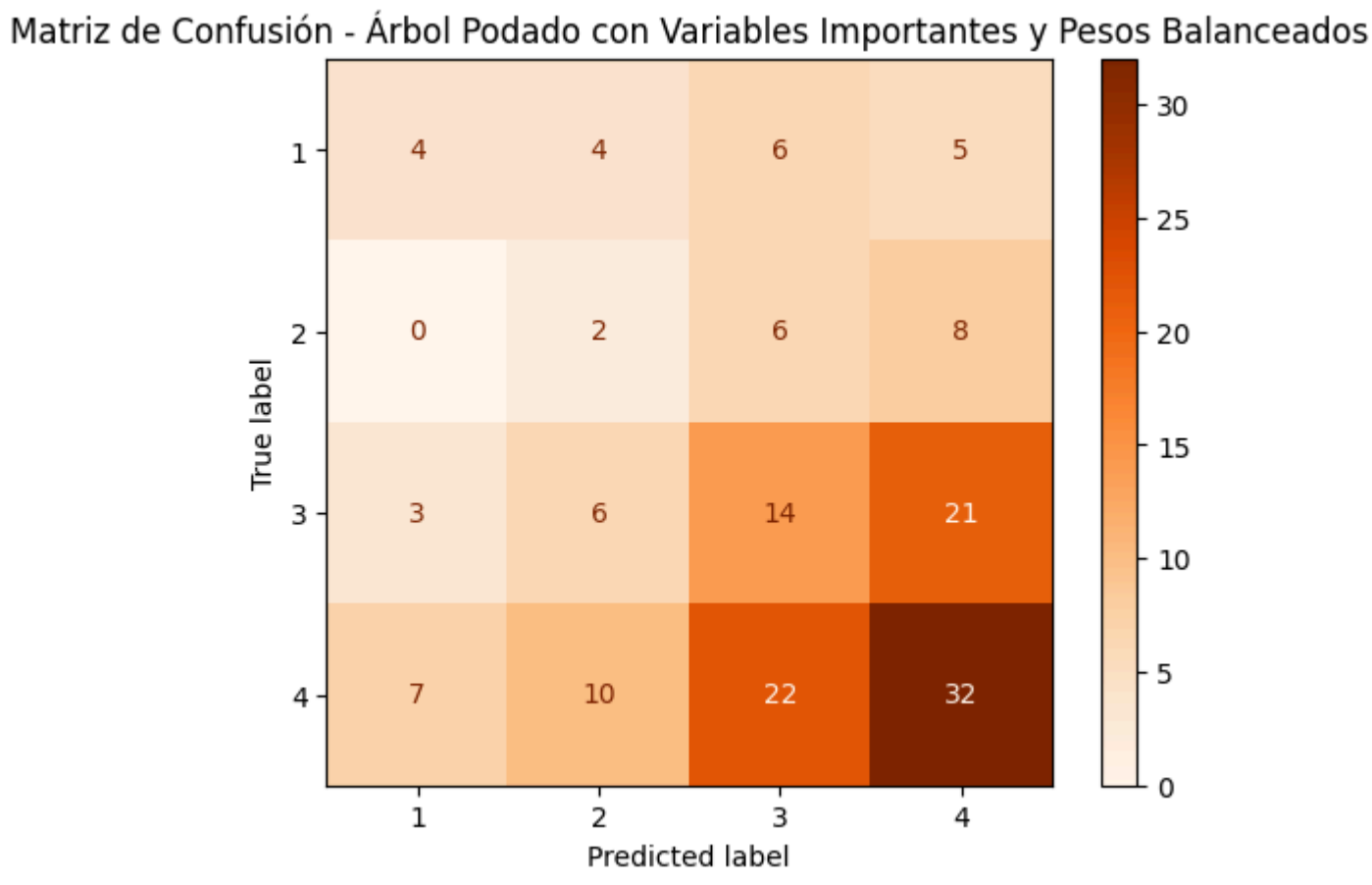


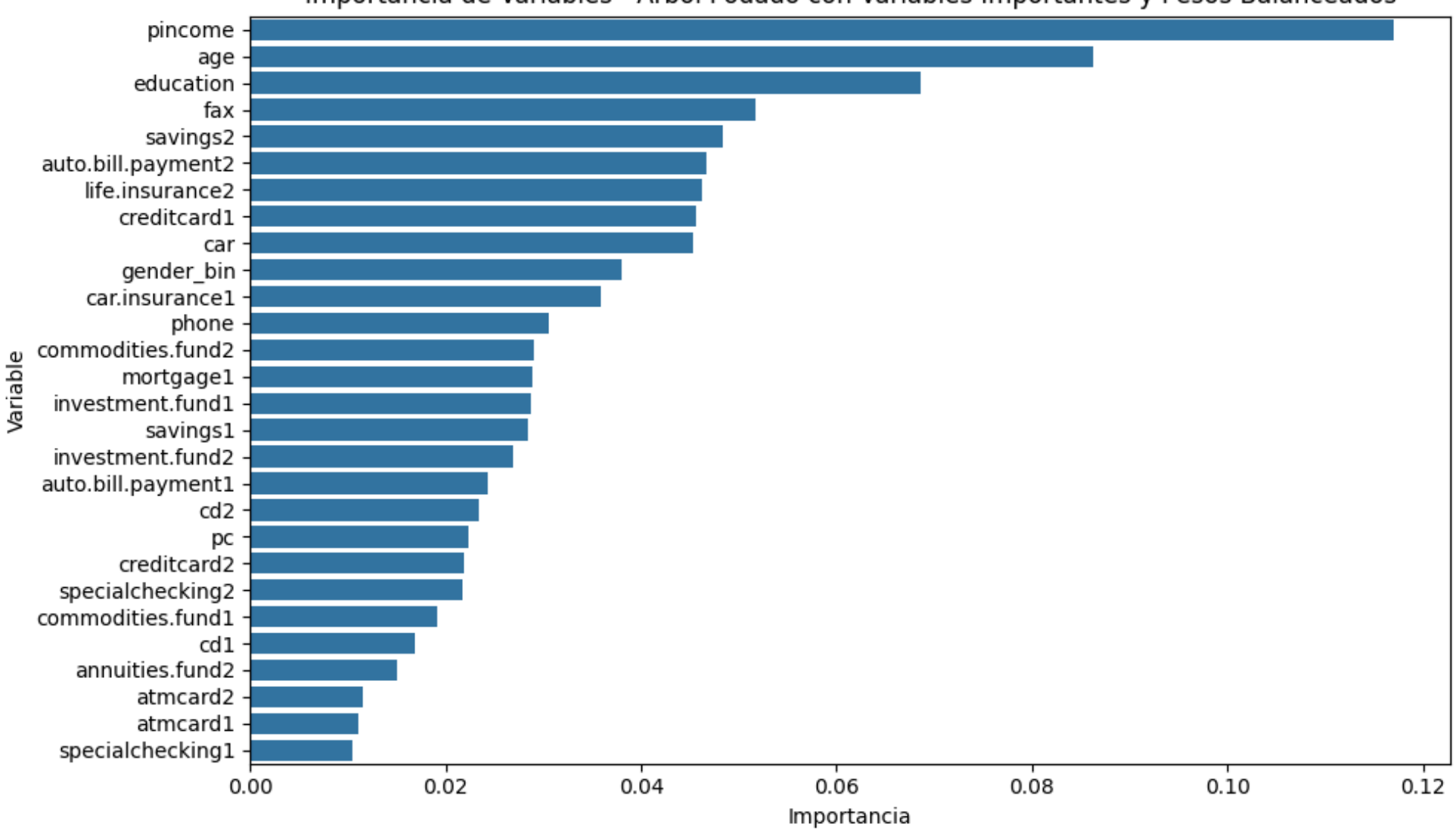


== Árbol Podado con Variables Importantes y Pesos Balanceados ==

	precision	recall	f1-score	support
1	0.29	0.21	0.24	19
2	0.09	0.12	0.11	16
3	0.29	0.32	0.30	44
4	0.48	0.45	0.47	71
accuracy			0.35	150
macro avg	0.29	0.28	0.28	150
weighted avg	0.36	0.35	0.35	150

Kappa: 0.02545743834526648






```
 Resultados almacenados:
accuracy_arbol_basico: 0.4200
f1_arbol_basico: 0.4246
kappa_arbol_basico: 0.1331
accuracy_arbol_podado: 0.5000
f1_arbol_podado: 0.4596
kappa_arbol_podado: 0.1784
accuracy_arbol_balanceado: 0.3533
f1_arbol_balanceado: 0.3568
kappa_arbol_balanceado: 0.0266
accuracy_arbol_importante_balanceado: 0.3467
f1_arbol_importante_balanceado: 0.3523
kappa_arbol_importante_balanceado: 0.0255
```

Modelo de Random Forest

El modelo básico de Random Forest alcanzó una precisión de 0.46, apenas superior al árbol de decisión original, aunque presentó limitaciones evidentes. Si bien logró predecir con mayor eficacia la clase 4 (clientes muy satisfechos), con 53 aciertos, tuvo serias dificultades con las clases 1, 2 y 3. En particular, la clase 2 fue mayormente confundida con la clase 4. Esto se refleja en la matriz de confusión normalizada, donde más del 70 % de los registros de las clases 2 y 3 fueron clasificados erróneamente. A pesar de que se respetaron adecuadamente los tipos de variables durante el entrenamiento, el desbalance entre clases siguió afectando la capacidad predictiva del modelo.

Cabe destacar que Random Forest es un modelo de tipo ensemble basado en la técnica de bagging . A diferencia de un solo árbol de decisión, este enfoque construye múltiples árboles sobre subconjuntos aleatorios del conjunto de datos y combina sus predicciones, reduciendo el sobreajuste y mejorando la estabilidad. No obstante, su rendimiento puede verse limitado si las clases están desbalanceadas o si el conjunto de variables incluye elementos poco relevantes.

Para mejorar el desempeño, se aplicó la técnica de GridSearchCV con el fin de ajustar los hiperparámetros. El proceso exploró distintas combinaciones de profundidad, número de árboles y cantidad de variables consideradas por división. La configuración óptima consistió en 300 árboles, profundidad máxima de 15 y 3 variables por nodo. Este modelo ajustado alcanzó una precisión de 0.49, con mejoras notables en la predicción de la clase 4 (61 aciertos) y un leve incremento en la precisión de la clase 3. Sin embargo, la clase 2 nuevamente no fue identificada correctamente, lo que evidenció que el ajuste de parámetros, aunque beneficioso, no resolvía por sí solo el problema del desbalance.

En una siguiente etapa, se entrenó un modelo con clases balanceadas, asignando pesos inversamente proporcionales a la frecuencia de cada clase. Esta variante mantuvo una precisión similar (0.47), pero logró una mejor distribución de las predicciones entre las clases minoritarias. La clase 1 mostró una leve mejora, mientras que las clases 2 y 3 continuaron siendo problemáticas. El análisis de importancia de variables reveló que age, pincome y education seguían siendo las más influyentes, destacando el valor de las variables numéricas y ordinales. Algunas variables binarias también cobraron relevancia, como gender_bin, savings2 y specialchecking2, posiblemente por su relación con los clientes más satisfechos.

Buscando mayor simplicidad y facil interpretación, se construyó una versión del modelo balanceado con filtrado de variables, eliminando aquellas cuya importancia fuera inferior a 0.01. Esta estrategia redujo el ruido y evitó que el modelo considerara variables irrelevantes. El resultado fue un modelo más compacto, centrado en las variables más influyentes, sin comprometer la precisión, que se mantuvo en 0.46. Se conservaron variables clave como age, pincome, education, además de algunas binarias útiles como savings2, auto.bill.payment1 y creditcard1. Esta selección facilitó el análisis posterior al trabajar con un conjunto de predictores más reducido y significativo.

En todos los modelos desarrollados se respetó cuidadosamente el tipo de cada variable, lo cual fue crucial para su correcto funcionamiento. Las variables numéricas demostraron ser las más consistentes en la predicción del nivel de satisfacción, mientras que education, tratada como ordinal, también ofreció un buen aporte, probablemente por su vínculo con el conocimiento financiero de los clientes. Las variables binarias mostraron un comportamiento más heterogéneo: algunas relacionadas con productos o servicios bancarios resultaron informativas, mientras que otras solo añadían complejidad sin contribuir al rendimiento.

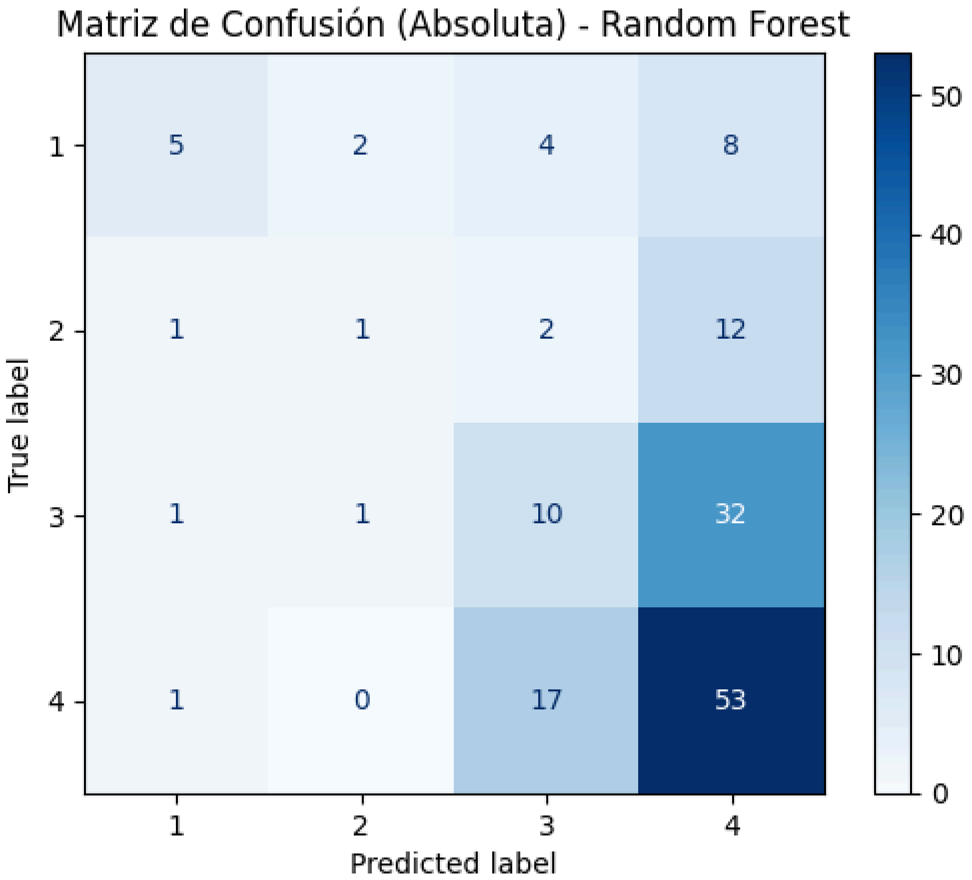
En conclusión, los modelos de Random Forest ofrecieron una mejora moderada respecto al árbol de decisión, especialmente tras aplicar técnicas como la optimización de hiperparámetros, el balanceo de clases y la selección de variables. Aunque la precisión general no superó el 50 %, el proceso permitió identificar con claridad qué variables aportan valor a la predicción del nivel de satisfacción del cliente bancario. La combinación de métodos ensemble, buen tratamiento de las variables y filtrado adecuado resultaron clave para construir modelos más robustos, interpretables y centrados en los factores verdaderamente relevantes del análisis.

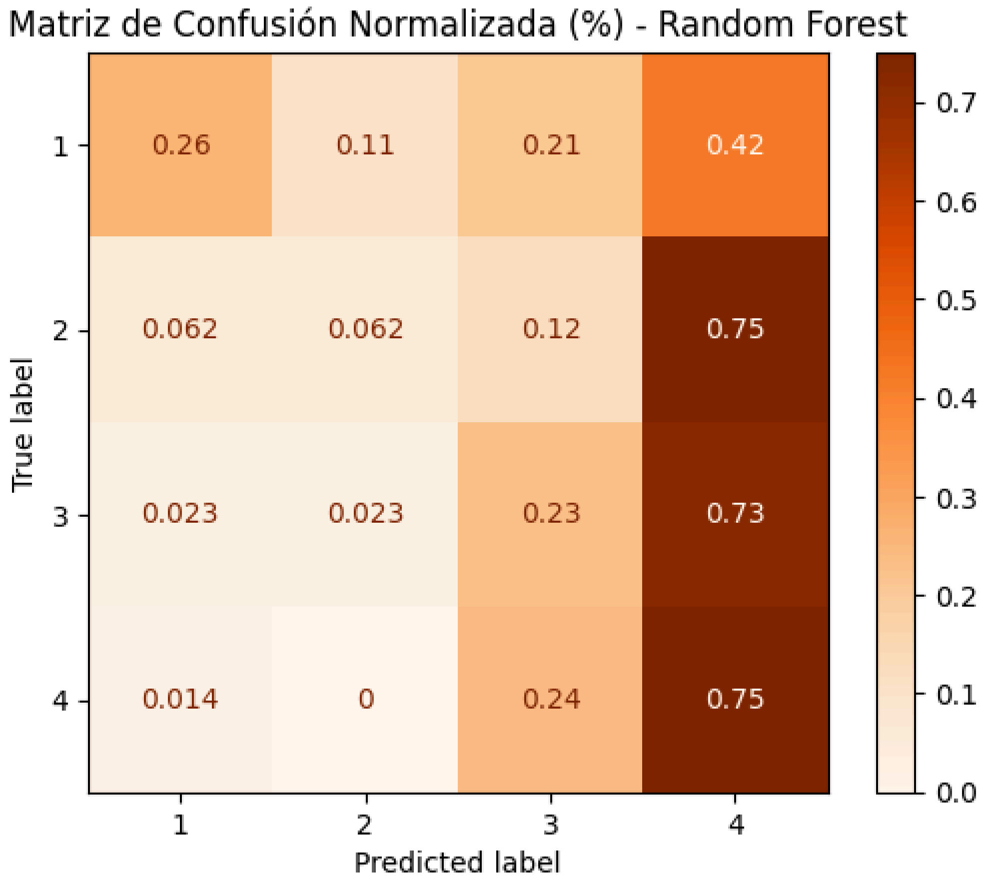
In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import (
    classification_report, ConfusionMatrixDisplay,
    accuracy_score, f1_score, cohen_kappa_score,
    precision_score, recall_score
)

# ============================================
# 5. Random Forest básico
# ============================================
rf_model = RandomForestClassifier(n_estimators=100, random_state=40)
rf_model.fit(X_train, y_train)

# Predicción en X_test
y_pred_rf = rf_model.predict(X_test)

print("== Random Forest ==")
print(classification_report(y_test, y_pred_rf))

# Matriz de confusión con conteo absoluto
ConfusionMatrixDisplay.from_predictions(y_test, y_pred_rf, cmap="Blues")
plt.title("Matriz de Confusión (Absoluta) - Random Forest")
plt.show()

# Matriz de confusión normalizada (porcentaje por fila)
ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred_rf,
    cmap="Oranges",
    normalize='true'
)
plt.title("Matriz de Confusión Normalizada (%) - Random Forest")
plt.show()

# ============================================
# 6. Búsqueda de hiperparámetros (GridSearchCV) para Random Forest
# ============================================
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_features': [3, 5, 7, 'sqrt'],
    'max_depth': [None, 10, 15],
    'min_samples_split': [2, 5],
}

grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=40),
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    n_jobs=-1,
    verbose=1
)
grid_search.fit(X_train, y_train)

print("Mejores hiperparámetros:", grid_search.best_params_)

best_model = grid_search.best_estimator_

# Predicción del mejor modelo sobre X_test
y_pred_best = best_model.predict(X_test)

print("== Mejor Modelo (Random Forest + GridSearch) ==")
print(classification_report(y_test, y_pred_best))

ConfusionMatrixDisplay.from_predictions(y_test, y_pred_best, cmap="Blues")
plt.title("Random Forest + GridSearchCV")
```

```python
plt.show()

# ============================================
# 7. Resumen de Métricas
# ============================================
metrics_summary = {}

def compute_metrics(y_true, y_pred, model_name):
    metrics_summary[model_name] = {
        'Accuracy': accuracy_score(y_true, y_pred),
        'Precision': precision_score(y_true, y_pred, average='weighted', zero_division=0),
        'Recall': recall_score(y_true, y_pred, average='weighted', zero_division=0),
        'F1-Score': f1_score(y_true, y_pred, average='weighted', zero_division=0),
        'Kappa': cohen_kappa_score(y_true, y_pred)
    }

compute_metrics(y_test, y_pred_tree, "Árbol de Decisión")
compute_metrics(y_test, y_pred_pruned, "Árbol Podado")
compute_metrics(y_test, y_pred_rf, "Random Forest")
compute_metrics(y_test, y_pred_best, "Random Forest + GridSearchCV")

metrics_df = pd.DataFrame(metrics_summary).T.round(4)

print("\n== Resumen de Métricas ==")
print(metrics_df)

# ============================================
# 8. Análisis de errores y importancia de variables (Random Forest)
# ============================================
errors = (y_pred_rf != y_test)
error_counts = pd.Series(y_test[errors]).value_counts().sort_index()

plt.figure(figsize=(8, 5))
error_counts.plot(kind='bar', color='tomato')
plt.title("Errores por Clase - Random Forest")
plt.xlabel("Clase Real")
plt.ylabel("Número de errores")
plt.xticks(rotation=45)
plt.grid(True, axis='y')
plt.tight_layout()
plt.show()

importances = best_model.feature_importances_
features = X.columns

feature_importance_df = pd.DataFrame({
    'Variable': features,
    'Importancia': importances
})

feature_importance_df = feature_importance_df[
    ~feature_importance_df['Variable'].str.lower().str.contains('id|unnamed')
]

feature_importance_df = feature_importance_df.sort_values(by='Importancia', ascending=False)

top_n = 20
plt.figure(figsize=(12, 6))
sns.barplot(
    data=feature_importance_df.head(top_n),
    x='Importancia',
    y='Variable',
    palette='crest'
)
plt.title(f"Top {top_n} Variables más Importantes - Random Forest")
plt.xlabel("Importancia")
plt.ylabel("Variable")
plt.tight_layout()
plt.show()

# ============================
# 9. Random Forest Balanceado
# ============================
rf_balanced = RandomForestClassifier(
    n_estimators=100,
    random_state=40,
    class_weight='balanced'
)
rf_balanced.fit(X_train, y_train)
y_pred_rf_bal = rf_balanced.predict(X_test)

print("== Random Forest con Pesos Balanceados ==")
print(classification_report(y_test, y_pred_rf_bal))

ConfusionMatrixDisplay.from_predictions(y_test, y_pred_rf_bal, cmap="Oranges")
plt.title("Matriz de Confusión - Random Forest Balanceado")
plt.show()
```

```python
# Importancia de variables
importances_bal = rf_balanced.feature_importances_
rf_bal_df = pd.DataFrame({
    'Variable': X.columns,
    'Importancia': importances_bal
}).sort_values(by='Importancia', ascending=False)

# Mostrar top 20
top_n = 20
plt.figure(figsize=(12, 6))
sns.barplot(data=rf_bal_df.head(top_n), x='Importancia', y='Variable', palette='flare')
plt.title(f"Top {top_n} Variables más Importantes - RF Balanceado")
plt.tight_layout()
plt.show()

# Mostrar variables con baja importancia (< 0.01)
print("\n Variables con baja importancia (< 0.01):")
print(rf_bal_df[rf_bal_df['Importancia'] < 0.01])


# ============================
# 10. RF Balanceado con Filtro
# ============================

# Filtrar variables significativas
variables_utiles_rf = rf_bal_df[rf_bal_df['Importancia'] >= 0.01]['Variable'].tolist()
X_reducido_rf = X[variables_utiles_rf]

# Nuevo split con variables filtradas
X_train_rf_red, X_test_rf_red, y_train_rf_red, y_test_rf_red = train_test_split(
    X_reducido_rf, y, test_size=0.3, random_state=40, stratify=y
)

# Entrenar Random Forest balanceado con datos reducidos
rf_filtrado = RandomForestClassifier(
    n_estimators=100,
    random_state=40,
    class_weight='balanced'
)
rf_filtrado.fit(X_train_rf_red, y_train_rf_red)
y_pred_rf_filtrado = rf_filtrado.predict(X_test_rf_red)

# Evaluación
print("== Random Forest Balanceado con Variables Filtradas ==")
print(classification_report(y_test_rf_red, y_pred_rf_filtrado))

ConfusionMatrixDisplay.from_predictions(y_test_rf_red, y_pred_rf_filtrado, cmap="Greens")
plt.title("Matriz de Confusión - RF Balanceado + Filtro")
plt.show()

# Nueva importancia de variables
importances_rf_filtrado = rf_filtrado.feature_importances_
rf_filtrado_df = pd.DataFrame({
    'Variable': X_reducido_rf.columns,
    'Importancia': importances_rf_filtrado
}).sort_values(by='Importancia', ascending=False)

plt.figure(figsize=(12, 6))
sns.barplot(data=rf_filtrado_df.head(top_n), x='Importancia', y='Variable', palette='crest')
plt.title(f"Top {top_n} Variables - RF Balanceado + Filtrado")
plt.tight_layout()
plt.show()


# ============================
# 11. Agregar métricas finales
# ============================
compute_metrics(y_test, y_pred_rf_bal, "RF Balanceado")
compute_metrics(y_test_rf_red, y_pred_rf_filtrado, "RF Balanceado + Filtro")

metrics_df = pd.DataFrame(metrics_summary).T.round(4)
print("\n== Resumen Completo de Métricas hasta ahora ==")
print(metrics_df)
```

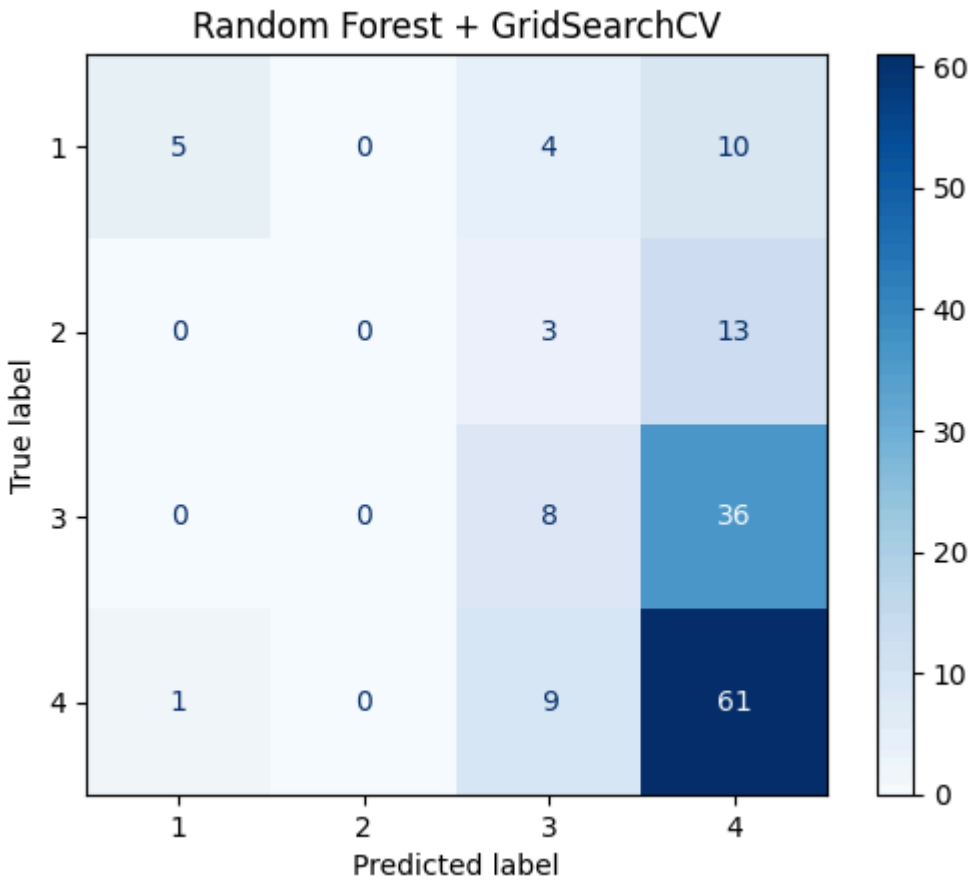
```
== Random Forest ==
              precision    recall  f1-score   support

           1       0.62      0.26      0.37        19
           2       0.25      0.06      0.10        16
           3       0.30      0.23      0.26        44
           4       0.50      0.75      0.60        71

    accuracy                           0.46       150
   macro avg       0.42      0.32      0.33       150
weighted avg       0.43      0.46      0.42       150
```



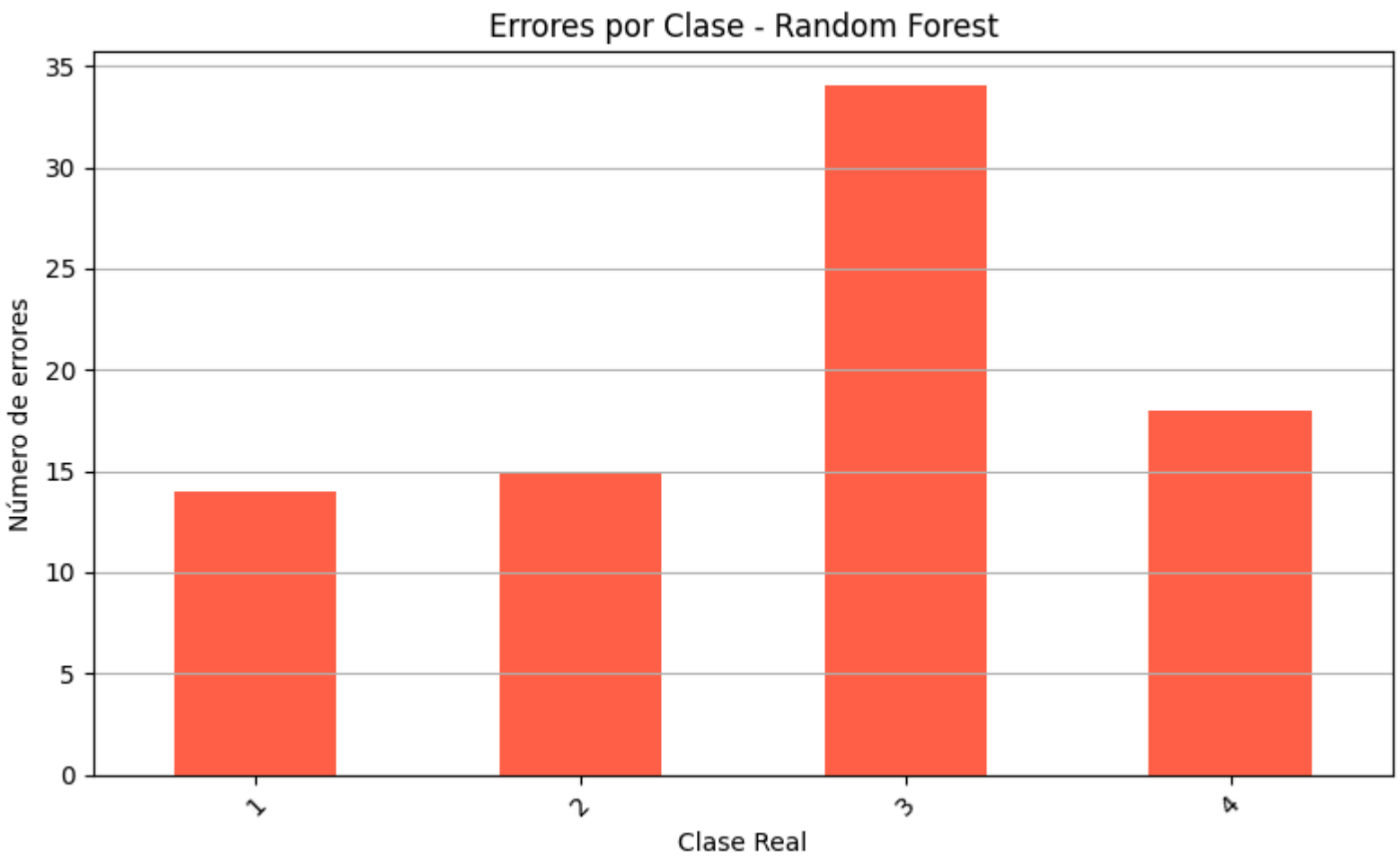



```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Mejores hiperparámetros: {'max_depth': 15, 'max_features': 3, 'min_samples_split': 5, 'n_estimators': 300}
== Mejor Modelo (Random Forest + GridSearch) ==
              precision    recall  f1-score   support

           1       0.83      0.26      0.40        19
           2       0.00      0.00      0.00        16
           3       0.33      0.18      0.24        44
           4       0.51      0.86      0.64        71

    accuracy                           0.49       150
   macro avg       0.42      0.33      0.32       150
weighted avg       0.44      0.49      0.42       150
```

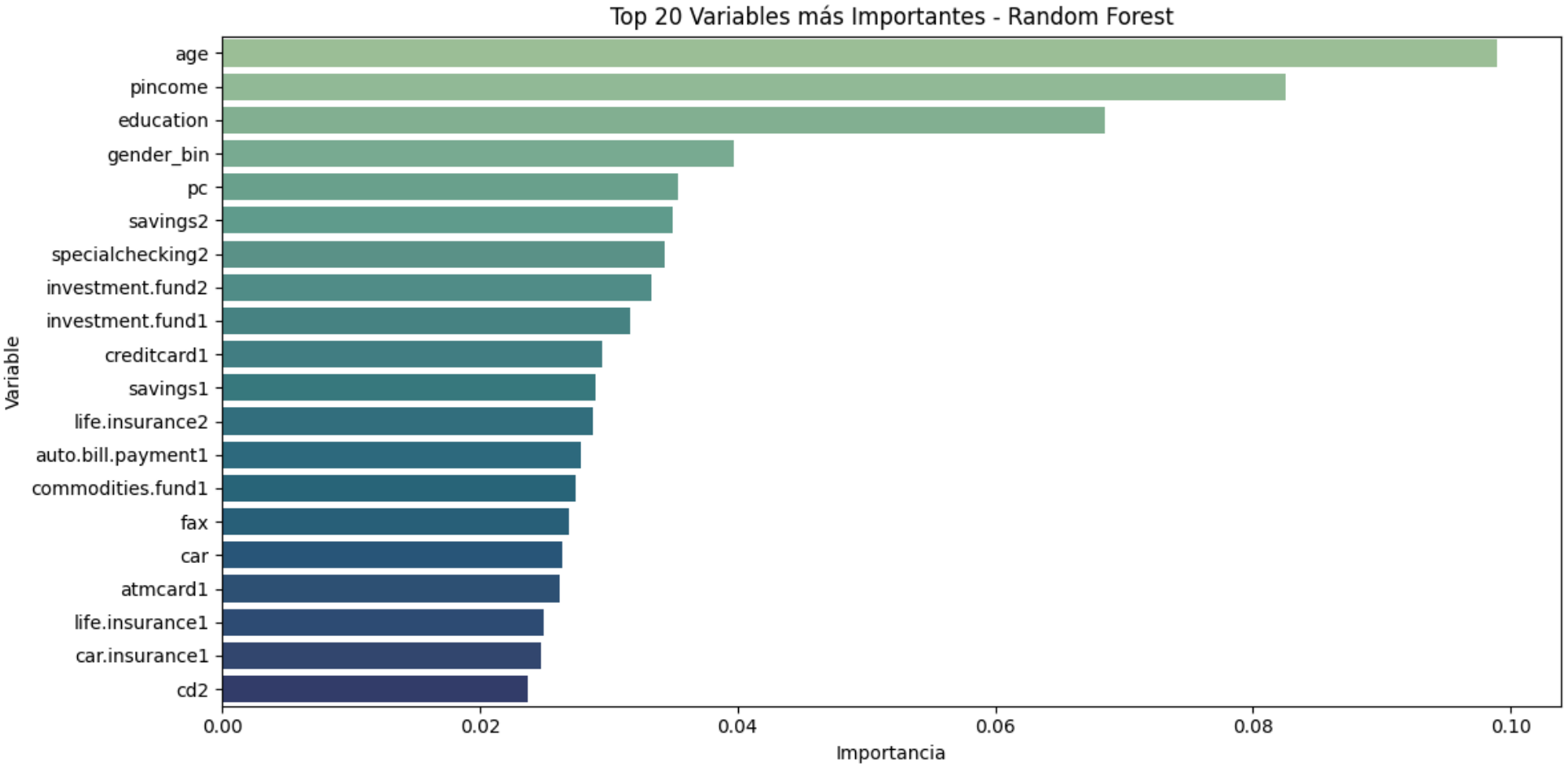
```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



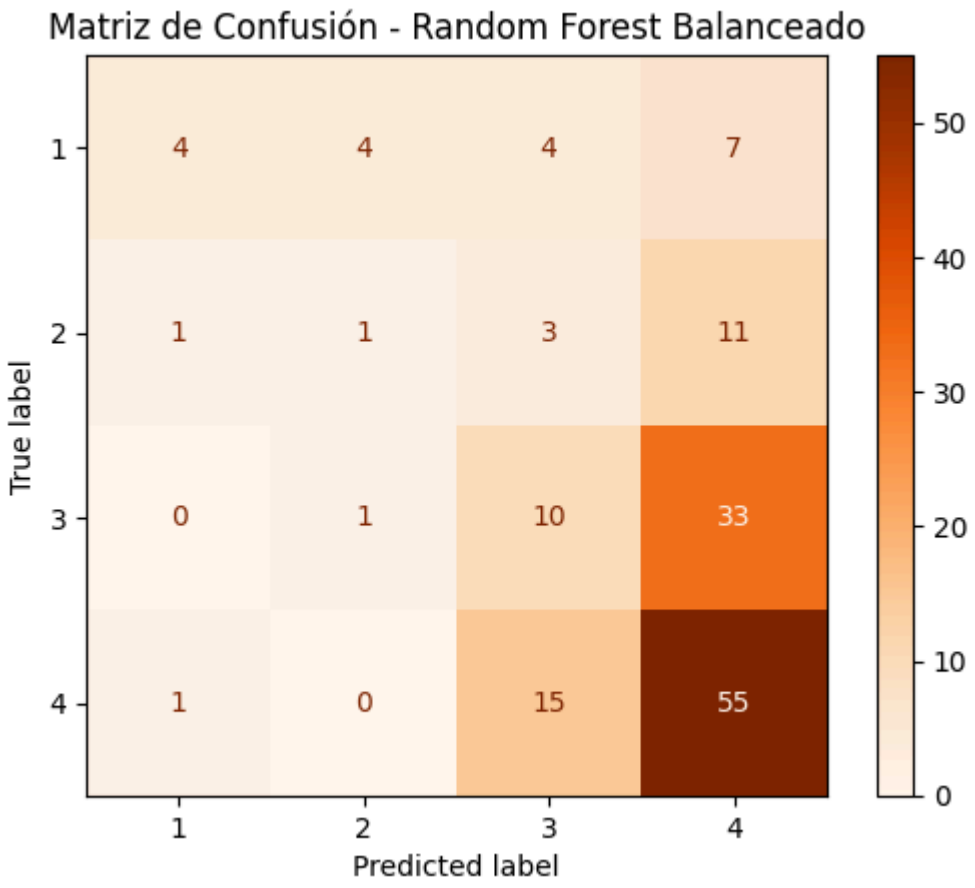

```
== Resumen de Métricas ==
                            Accuracy  Precision  Recall  F1-Score   Kappa
Árbol de Decisión             0.4200     0.4308  0.4200    0.4246  0.1331
Árbol Podado                  0.5000     0.4365  0.5000    0.4596  0.1784
Random Forest                 0.4600     0.4336  0.4600    0.4188  0.0917
Random Forest + GridSearchCV  0.4933     0.4439  0.4933    0.4220  0.1101
```



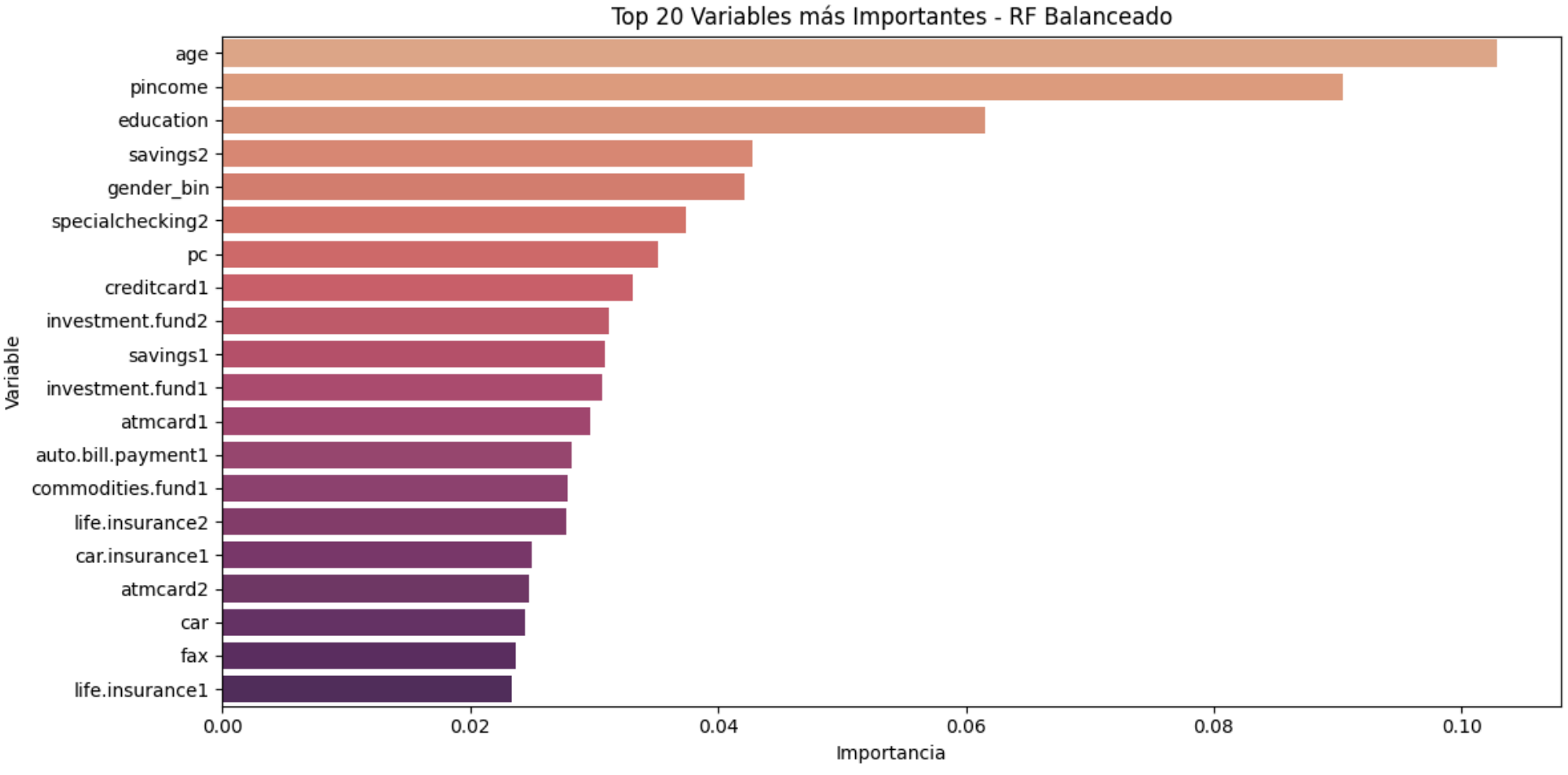

```
/tmp/ipython-input-4-3473542867.py:131: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `lege
nd=False` for the same effect.

  sns.barplot(
```



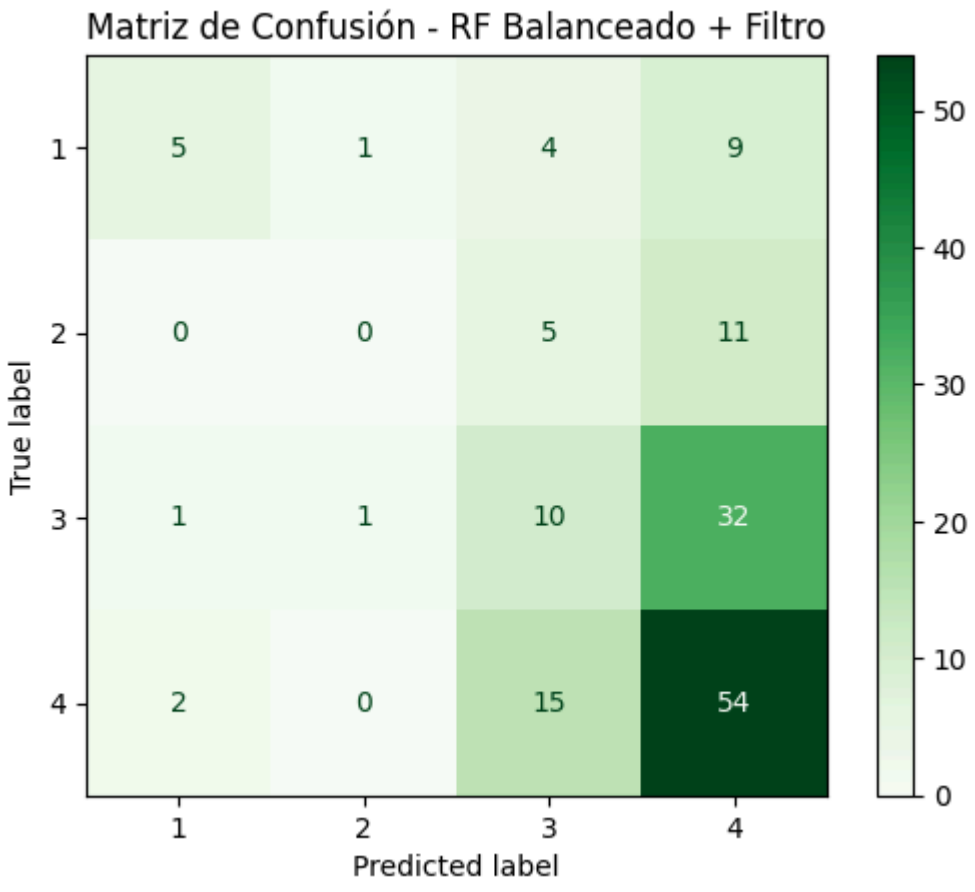

```
== Random Forest con Pesos Balanceados ==
              precision    recall  f1-score   support

           1       0.67      0.21      0.32        19
           2       0.17      0.06      0.09        16
           3       0.31      0.23      0.26        44
           4       0.52      0.77      0.62        71

    accuracy                           0.47       150
   macro avg       0.42      0.32      0.32       150
weighted avg       0.44      0.47      0.42       150
```



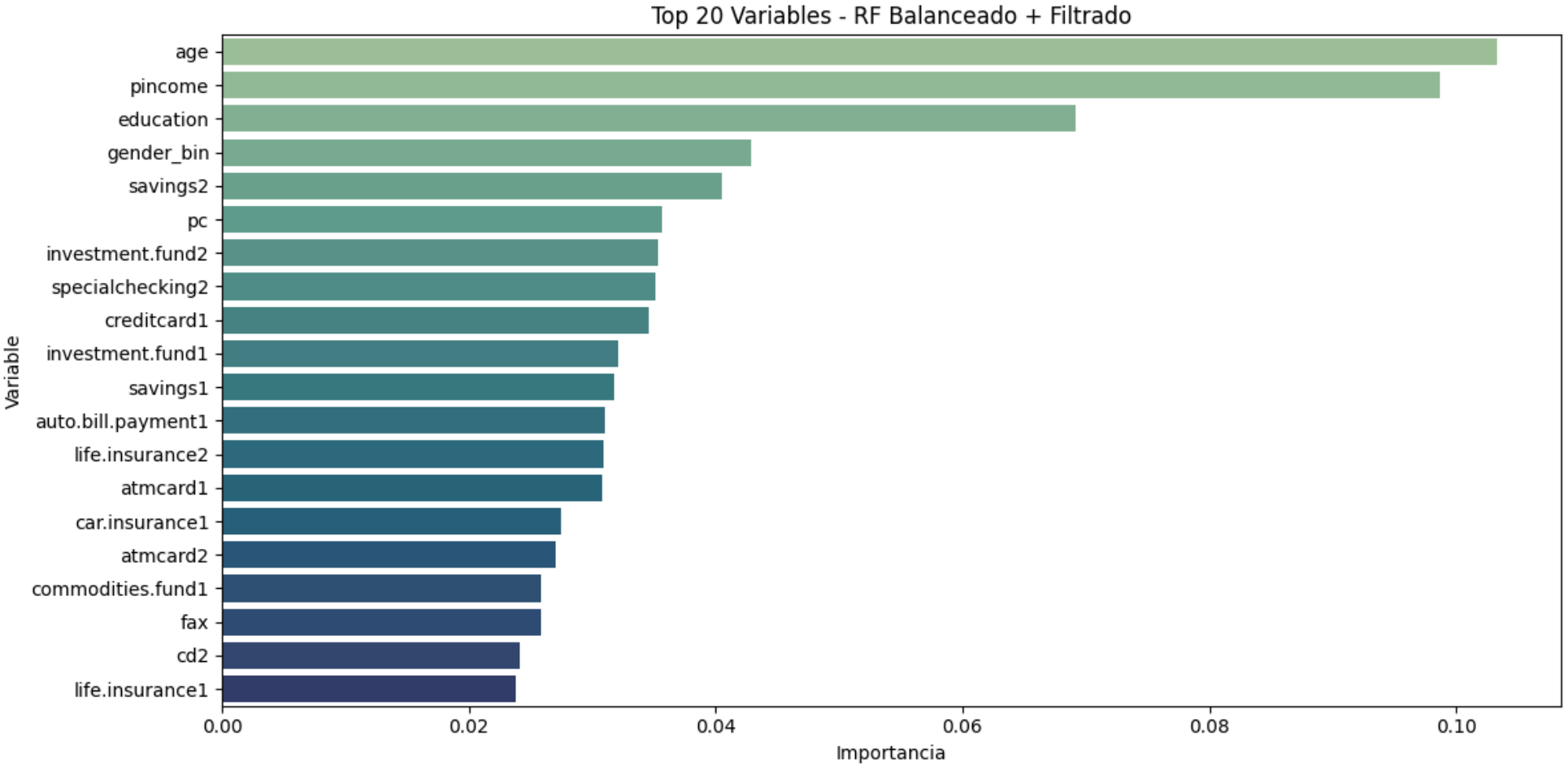

```
/tmp/ipython-input-4-3473542867.py:171: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `lege
nd=False` for the same effect.

  sns.barplot(data=rf_bal_df.head(top_n), x='Importancia', y='Variable', palette='flare')
```




```
 Variables con baja importancia (< 0.01):
          Variable  Importancia
20  personal.loans2     0.009084
19  personal.loans1     0.008975
29   car.insurance2     0.008869
27  annuities.fund2     0.008151
31  home.insurance2     0.006242
== Random Forest Balanceado con Variables Filtradas ==
              precision    recall  f1-score   support

           1       0.62      0.26      0.37        19
           2       0.00      0.00      0.00        16
           3       0.29      0.23      0.26        44
           4       0.51      0.76      0.61        71

    accuracy                           0.46       150
   macro avg       0.36      0.31      0.31       150
weighted avg       0.41      0.46      0.41       150
```




```
/tmp/ipython-input-4-3473542867.py:219: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `lege
nd=False` for the same effect.

  sns.barplot(data=rf_filtrado_df.head(top_n), x='Importancia', y='Variable', palette='crest')
```




```
== Resumen Completo de Métricas hasta ahora ==
                            Accuracy  Precision  Recall  F1-Score   Kappa
Árbol de Decisión             0.4200     0.4308  0.4200    0.4246  0.1331
Árbol Podado                  0.5000     0.4365  0.5000    0.4596  0.1784
Random Forest                 0.4600     0.4336  0.4600    0.4188  0.0917
Random Forest + GridSearchCV  0.4933     0.4439  0.4933    0.4220  0.1101
RF Balanceado                 0.4667     0.4395  0.4667    0.4216  0.1015
RF Balanceado + Filtro        0.4600     0.4066  0.4600    0.4109  0.0861
```

Red Neuronal

Por último se construyó un modelo de red neuronal multicapa (MLP) para predecir la variable de satisfacción, que tenía una distribución desigual entre las clases 1, 2, 3 y 4. Para mejorar la calidad del entrenamiento, se aplicó la técnica SMOTE, que genera nuevas observaciones sintéticas en las clases minoritarias. Esto permitió crear un conjunto de entrenamiento más balanceado, ayudando al modelo a no favorecer las clases más frecuentes y a tener un mejor desempeño en las categorías con menos datos.

Se utilizó GridSearchCV para buscar la mejor combinación de hiperparámetros del modelo, pero siempre respetando la regla de oro del aprendizaje automático: el conjunto de prueba no se toca hasta el final. En el grid se probaron cuatro configuraciones para el número de neuronas en la capa oculta y cinco valores del parámetro de regularización alpha, lo que dio lugar a 20 combinaciones. Con validación cruzada de 5 particiones, se realizaron 100 entrenamientos sobre los datos balanceados de entrenamiento (350 observaciones), sin utilizar información del conjunto de prueba.

Este enfoque permitió mantener el control sobre el sobreajuste. Un grid demasiado grande puede hacer que el modelo aprenda patrones que solo funcionan bien dentro del entrenamiento pero que no generalizan. En este caso, el número de combinaciones fue razonable, lo que ayudó a encontrar una configuración efectiva sin forzar el modelo. La mejor combinación resultó ser hidden_layer_sizes=(10,) y alpha=2.0.

Para las variables predictoras, se decidió utilizar las más importantes según el árbol decisión balanceado, el cual había identificado atributos como pincome, age, education y auto.bill.payment2 con una importancia significativa. Se aplicó un filtro de importancia mayor o igual a 0.01 para mantener solo aquellas variables que realmente aportaban al modelo. Esto redujo el ruido, mejoró el rendimiento y ayudó a evitar un modelo innecesariamente complejo.

La red neuronal se configuró con la función de activación logística (también conocida como sigmoide), siguiendo un enfoque similar al que usa el modelo nnet en R. Aunque la variable de respuesta tiene un orden (es ordinal), se trató como un problema de clasificación con varias categorías. Esto se debe a que el modelo MLPClassifier puede manejar bien este tipo de tareas, usando una capa de salida que calcula la probabilidad de cada clase sin necesidad de respetar un orden entre ellas.

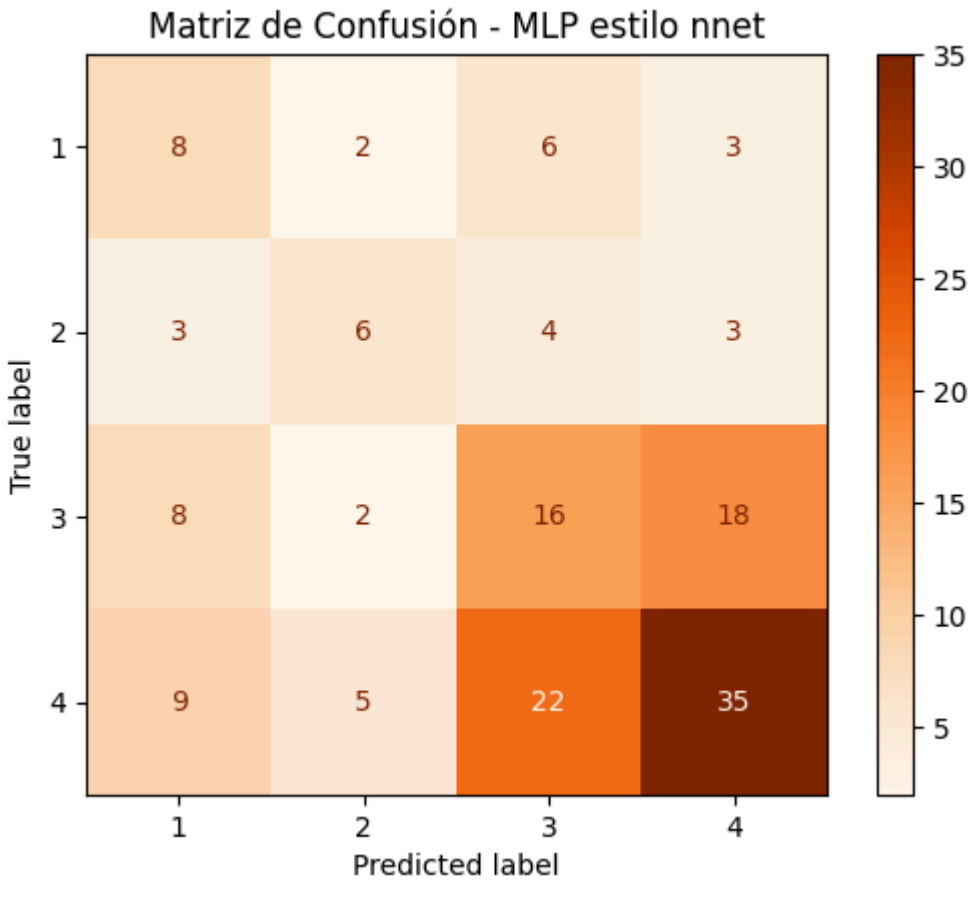
Los resultados finales fueron positivos. Se obtuvo un accuracy del 43%, un f1-score macro de 0.40 y un AUC macro de 0.642. Las curvas ROC mostraron un desempeño particularmente bueno para las clases 1 (AUC = 0.74) y 2 (AUC = 0.65), mientras que la clase 4 también alcanzó un AUC aceptable (0.62). La curva de aprendizaje mostró una separación moderada entre los rendimientos de entrenamiento y validación, sin grandes saltos, lo que indica que no hay señales claras de sobreajuste. En general, el modelo logró un buen equilibrio entre precisión, generalización y simplicidad, sobre todo teniendo en cuenta el tamaño limitado del dataset y la dificultad del problema.

In [5]:
```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV, learning_curve
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import (
    classification_report, confusion_matrix, ConfusionMatrixDisplay,
    roc_auc_score, roc_curve, auc
)
import matplotlib.pyplot as plt
```

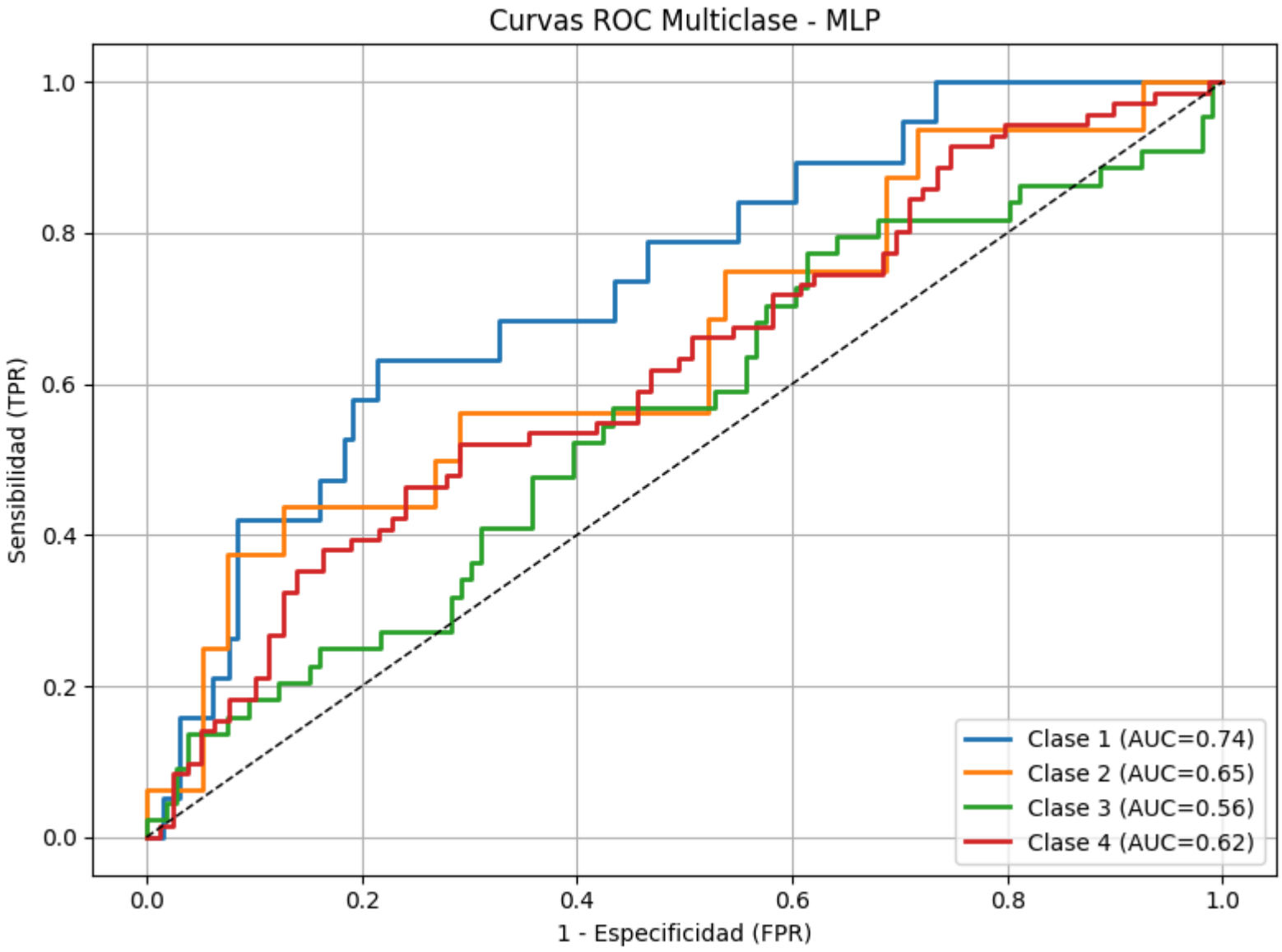
```python
import numpy as np
import pandas as pd

# 1. Variables importantes (según arbol decisión balanceado con importancia ≥ 0.01)
variables_importantes = [
    'pincome', 'age', 'education', 'auto.bill.payment2', 'gender_bin',
    'car', 'creditcard1', 'fax', 'cd1', 'mortgage1', 'savings2',
    'investment.fund1', 'car.insurance1', 'creditcard2', 'specialchecking2',
    'commodities.fund2', 'phone', 'pc', 'investment.fund2', 'atmcard2',
    'life.insurance2', 'auto.bill.payment1', 'savings1', 'annuities.fund2',
    'atmcard1', 'cd2', 'specialchecking1', 'commodities.fund1'
]

# 2. Subset del dataset
X_nn = X[variables_importantes].copy()
y_nn = y.copy()


# 3. División entrenamiento/prueba
X_train, X_test, y_train, y_test = train_test_split(
    X_nn, y_nn,
    test_size=0.30,
    stratify=y_nn,
    random_state=42
)

print("Tamaño entrenamiento:", X_train.shape[0])
print("Tamaño prueba:", X_test.shape[0])

# 4. Aplicar SMOTE al conjunto de entrenamiento
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

# 5. Pipeline con MLP estilo `nnet`
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('mlp', MLPClassifier(
        activation='logistic',
        solver='lbfgs',
        max_iter=2000,
        random_state=42
    ))
])

# 6. Grid de hiperparámetros
param_grid = {
    'mlp__hidden_layer_sizes': [(3,), (5,), (8,), (10,)],
    'mlp__alpha': [0.1, 0.5, 1.0, 2.0, 5.0]
}

# 7. GridSearchCV con validación cruzada
grid = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    scoring='accuracy',
    cv=StratifiedKFold(n_splits=5),
    n_jobs=-1,
    verbose=1
)

# 8. Entrenamiento
grid.fit(X_train_bal, y_train_bal)

# 9. Evaluación
y_pred = grid.predict(X_test)
y_proba = grid.predict_proba(X_test)

print("== Mejor combinación de hiperparámetros ==")
print(grid.best_params_)

print("\n== Classification Report ==")
print(classification_report(y_test, y_pred))

# 10. Matriz de confusión
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap="Oranges")
plt.title("Matriz de Confusión - MLP estilo nnet")
plt.show()

# 11. AUC macro
class_names = [str(cls) for cls in np.unique(y_train)]
y_test_bin = label_binarize(y_test, classes=np.unique(y_train))
roc_auc = roc_auc_score(y_test_bin, y_proba, average='macro', multi_class='ovr')
print("AUC macro:", round(roc_auc, 3))

# 12. Curvas ROC
fpr, tpr, roc_auc_dict = {}, {}, {}
```

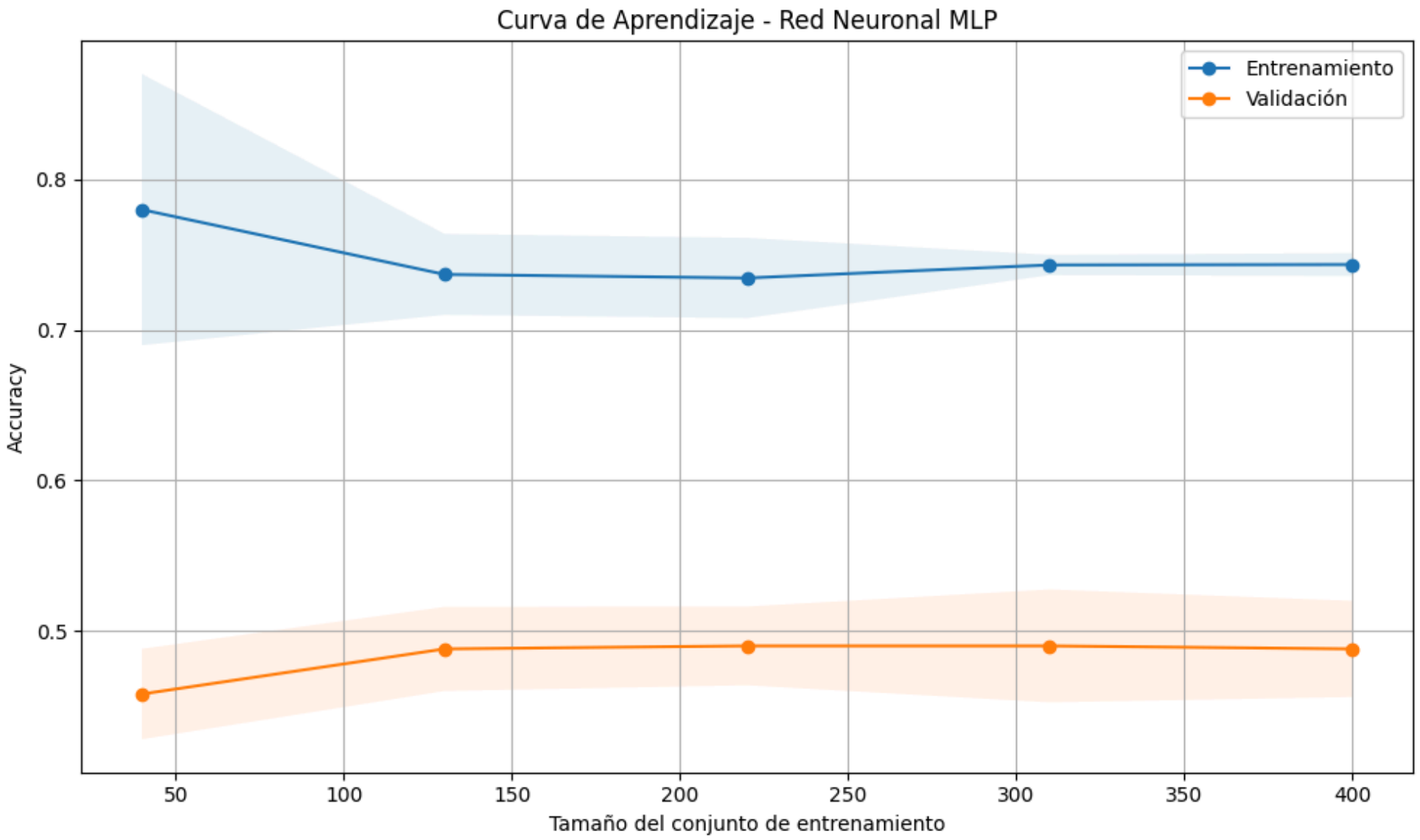
```python
for i in range(len(class_names)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_proba[:, i])
    roc_auc_dict[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(8, 6))
for i, color in zip(range(len(class_names)), plt.cm.tab10.colors):
    plt.plot(fpr[i], tpr[i], lw=2, label=f"Clase {class_names[i]} (AUC={roc_auc_dict[i]:.2f})")
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlabel('1 - Especificidad (FPR)')
plt.ylabel('Sensibilidad (TPR)')
plt.title('Curvas ROC Multiclase - MLP')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()

# 13. Curva de aprendizaje
best_model = grid.best_estimator_
train_sizes, train_scores, test_scores = learning_curve(
    estimator=best_model,
    X=X_nn, y=y_nn,
    train_sizes=np.linspace(0.1, 1.0, 5),
    cv=StratifiedKFold(n_splits=5),
    scoring='accuracy',
    n_jobs=-1
)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.figure(figsize=(10, 6))
plt.title("Curva de Aprendizaje - Red Neuronal MLP")
plt.xlabel("Tamaño del conjunto de entrenamiento")
plt.ylabel("Accuracy")
plt.grid(True)
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.1)
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.1)
plt.plot(train_sizes, train_mean, 'o-', label="Entrenamiento")
plt.plot(train_sizes, test_mean, 'o-', label="Validación")
plt.legend(loc="best")
plt.tight_layout()
plt.show()
```

```
Tamaño entrenamiento: 350
Tamaño prueba: 150
Fitting 5 folds for each of 20 candidates, totalling 100 fits
== Mejor combinación de hiperparámetros ==
{'mlp__alpha': 2.0, 'mlp__hidden_layer_sizes': (10,)}

== Classification Report ==
              precision    recall  f1-score   support

           1       0.29      0.42      0.34        19
           2       0.40      0.38      0.39        16
           3       0.33      0.36      0.35        44
           4       0.59      0.49      0.54        71

    accuracy                           0.43       150
   macro avg       0.40      0.41      0.40       150
weighted avg       0.46      0.43      0.44       150
```



AUC macro: 0.642





Conclusiones y comparacion final de modelos

La elección del mejor modelo para clasificar la satisfacción de los clientes no se realiza solo en base a una métrica como el accuracy. Es importante tener en cuenta qué si por ejemplo se necesita interpretar fácilmente los resultados, si se busca un rendimiento general equilibrado, o si es importante detectar bien todas las categorías, incluso las menos representadas. En este trabajo se compararon distintos modelos usando métricas como F1-score, Kappa y AUC, que ayudan a tener una visión más completa del desempeño.

SI queremos un modelo que permita explicar fácilmente las decisiones que toma, el árbol de decisión podado es una buena opción. Este modelo obtuvo el mejor accuracy y el mayor valor de Kappa, lo que indica una buena concordancia entre predicción y realidad. Además, su estructura es simple de visualizar y entender, lo cual es útil en contextos donde la transparencia del modelo es importante o presentaciones a áreas no técnicas.

En cambio, si lo más importante es mejorar el rendimiento general, especialmente cuando hay clases poco representadas, la red neuronal MLP entrenada con SMOTE destaca como una alternativa interesante. Este modelo alcanzó el mejor F1-score y mostró una capacidad razonable para distinguir entre todas las clases, como lo refleja su AUC de 0.64. Aunque su estructura es más difícil de interpretar, puede ser una buena elección cuando lo que se busca es un mayor equilibrio entre precisión y cobertura en todos los niveles de satisfacción.

In [6]:
```python
# Consolidación de todas las métricas
import pandas as pd
import numpy as np

# Diccionario base con métricas desde modelos anteriores
resumen_metricas = {
    "Árbol de Decisión": {
        "Accuracy": results["accuracy_arbol_basico"],
        "F1-Score": results["f1_arbol_basico"],
        "Kappa": results["kappa_arbol_basico"],
        "Precision": precision_score(y_test, y_pred_tree, average='weighted', zero_division=0),
        "Recall": recall_score(y_test, y_pred_tree, average='weighted', zero_division=0),
        "ROC-AUC": np.nan
    },
    "Árbol Podado": {
        "Accuracy": results["accuracy_arbol_podado"],
        "F1-Score": results["f1_arbol_podado"],
        "Kappa": results["kappa_arbol_podado"],
        "Precision": precision_score(y_test, y_pred_pruned, average='weighted', zero_division=0),
        "Recall": recall_score(y_test, y_pred_pruned, average='weighted', zero_division=0),
        "ROC-AUC": np.nan
    },
    "Árbol Balanceado": {
        "Accuracy": results["accuracy_arbol_balanceado"],
        "F1-Score": results["f1_arbol_balanceado"],
        "Kappa": results["kappa_arbol_balanceado"],
        "Precision": precision_score(y_test, y_pred_bal, average='weighted', zero_division=0),
        "Recall": recall_score(y_test, y_pred_bal, average='weighted', zero_division=0),
        "ROC-AUC": np.nan
    },
    "Árbol Balanceado + Filtro": {
```

```
        "Accuracy": results["accuracy_arbol_importante_balanceado"],
        "F1-Score": results["f1_arbol_importante_balanceado"],
        "Kappa": results["kappa_arbol_importante_balanceado"],
        "Precision": precision_score(y_test, y_pred_final, average='weighted', zero_division=0),
        "Recall": recall_score(y_test, y_pred_final, average='weighted', zero_division=0),
        "ROC-AUC": np.nan
    },
    "Random Forest": {
        "Accuracy": accuracy_score(y_test, y_pred_rf),
        "F1-Score": f1_score(y_test, y_pred_rf, average='weighted', zero_division=0),
        "Kappa": cohen_kappa_score(y_test, y_pred_rf),
        "Precision": precision_score(y_test, y_pred_rf, average='weighted', zero_division=0),
        "Recall": recall_score(y_test, y_pred_rf, average='weighted', zero_division=0),
        "ROC-AUC": np.nan
    },
    "Random Forest + GridSearch": {
        "Accuracy": accuracy_score(y_test, y_pred_best),
        "F1-Score": f1_score(y_test, y_pred_best, average='weighted', zero_division=0),
        "Kappa": cohen_kappa_score(y_test, y_pred_best),
        "Precision": precision_score(y_test, y_pred_best, average='weighted', zero_division=0),
        "Recall": recall_score(y_test, y_pred_best, average='weighted', zero_division=0),
        "ROC-AUC": np.nan
    },
    "RF Balanceado": {
        "Accuracy": accuracy_score(y_test, y_pred_rf_bal),
        "F1-Score": f1_score(y_test, y_pred_rf_bal, average='weighted', zero_division=0),
        "Kappa": cohen_kappa_score(y_test, y_pred_rf_bal),
        "Precision": precision_score(y_test, y_pred_rf_bal, average='weighted', zero_division=0),
        "Recall": recall_score(y_test, y_pred_rf_bal, average='weighted', zero_division=0),
        "ROC-AUC": np.nan
    },
    "RF Balanceado + Filtro": {
        "Accuracy": accuracy_score(y_test_rf_red, y_pred_rf_filtrado),
        "F1-Score": f1_score(y_test_rf_red, y_pred_rf_filtrado, average='weighted', zero_division=0),
        "Kappa": cohen_kappa_score(y_test_rf_red, y_pred_rf_filtrado),
        "Precision": precision_score(y_test_rf_red, y_pred_rf_filtrado, average='weighted', zero_division=0),
        "Recall": recall_score(y_test_rf_red, y_pred_rf_filtrado, average='weighted', zero_division=0),
        "ROC-AUC": np.nan
    },
    "Red Neuronal MLP + SMOTE": {
        "Accuracy": accuracy_score(y_test, y_pred),
        "F1-Score": f1_score(y_test, y_pred, average='weighted', zero_division=0),
        "Kappa": cohen_kappa_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred, average='weighted', zero_division=0),
        "Recall": recall_score(y_test, y_pred, average='weighted', zero_division=0),
        "ROC-AUC": roc_auc  # calculado previamente
    }
}

# Convertir a DataFrame
df_resumen_metricas = pd.DataFrame(resumen_metricas).T.round(4)

# Mostrar resumen final
print(" Comparación de Métricas de Todos los Modelos:")
display(df_resumen_metricas)
```

```
Comparación de Métricas de Todos los Modelos:
```

	Accuracy	F1-Score	Kappa	Precision	Recall	ROC-AUC
Árbol de Decisión	0.4200	0.4246	0.1331	0.3417	0.3333	NaN
Árbol Podado	0.5000	0.4596	0.1784	0.3910	0.4533	NaN
Árbol Balanceado	0.3533	0.3568	0.0266	0.3850	0.3867	NaN
Árbol Balanceado + Filtro	0.3467	0.3523	0.0255	0.3652	0.3600	NaN
Random Forest	0.4333	0.3916	0.0469	0.3973	0.4333	NaN
Random Forest + GridSearch	0.4333	0.3567	0.0047	0.3398	0.4333	NaN
RF Balanceado	0.4267	0.3776	0.0341	0.3747	0.4267	NaN
RF Balanceado + Filtro	0.4600	0.4109	0.0861	0.4066	0.4600	NaN
Red Neuronal MLP + SMOTE	0.4333	0.4413	0.1735	0.4574	0.4333	0.6423