Librerias

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import requests
import os
import zipfile
from datetime import datetime
!pip install tabula-py
import tabula
import re
```

```
Requirement already satisfied: tabula-py in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: pandas>=0.25.3 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: numpy>1.24.4 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: distro in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packag
```

Recopilación de Datos

```
URL_DESCARGA = "https://estadisticas.seps.gob.ec/?sdm_process_download=1&dc
CARPETA_BASE = "seps_descargas"
CARPETA_SEG1 = "seps_segmento1"

os.makedirs(CARPETA_BASE, exist_ok=True)
os.makedirs(CARPETA_SEG1, exist_ok=True)

def descargar_archivo():
    fecha = datetime.now().strftime("%Y%m%d")
    ruta_archivo = f"{CARPETA_BASE}/seps_{fecha}.zip"

    print("Descargando archivo...")
    r = requests.get(URL_DESCARGA)
    with open(ruta_archivo, "wb") as f:
        f.write(r.content)

    print(f"Archivo descargado: {ruta_archivo}")
    return ruta_archivo

```

```
def extraer_segmento1(ruta_archivo):
    fecha = datetime.now().strftime("%Y%m%d")
    carpeta_temp = f"{CARPETA_BASE}/extraido_{fecha}"
    os.makedirs(carpeta_temp, exist_ok=True)

    # Extraer ZIP
    with zipfile.ZipFile(ruta_archivo, "r") as z:
        z.extractall(carpeta_temp)

    print("Archivos extraídos. Buscando archivo Segmento 1...")

    # Buscar el archivo Segmento 1
    archivo_segmento1 = None
    for root, dirs, files in os.walk(carpeta_temp):
        for file in files:
            if "segmento" in file.lower() and "1" in file.lower() and file.
                archivo_segmento1 = os.path.join(root, file)

    if archivo_segmento1 is None:
        raise FileNotFoundError("No se encontró el archivo Segmento 1 dentr

    # Copiar a carpeta final
    destino = os.path.join(CARPETA_SEG1, f"{fecha}_segmento1.xlsm")
    with open(archivo_segmento1, "rb") as f_origen:
        with open(destino, "wb") as f_destino:
            f_destino.write(f_origen.read())

    print(f"Archivo Segmento 1 listo: {destino}")
    return destino



zipfile_path = descargar_archivo()
ruta_segmento1 = extraer_segmento1(zipfile_path)

```

```
Descargando archivo...
Archivo descargado: seps_descargas/seps_20251204.zip
Archivos extraídos. Buscando archivo Segmento 1...
Archivo Segmento 1 listo: seps_segmento1/20251204_segmento1.xlsm
```

```
URL = "https://www.seps.gob.ec/?sdm_process_download=1&download_id=28370"
NOMBRE_ARCHIVO = "Calificacion_de_Riesgo.pdf"

print(f"Descargando PDF desde:\n{URL}\n")

resp = requests.get(URL)
resp.raise_for_status()  # Si hay error, detiene el programa

with open(NOMBRE_ARCHIVO, "wb") as f:
   f.write(resp.content)
```

```
11
12 print(f"PDF guardado como: {NOMBRE_ARCHIVO}")
13
14
15 pdf_path = "/content/Calificacion_de_Riesgo.pdf"
16
17 tablas = tabula.read_pdf(pdf_path, pages="all", lattice=True, multiple_tabl
18 df_ranking = pd.concat(tablas, ignore_index=True)
19
20
```

```
Descargando PDF desde:
https://www.seps.gob.ec/?sdm_process_download=1&download_id=28370

PDF guardado como: Calificacion_de_Riesgo.pdf
```

Limpieza y modificación de datos

ASPECTOS TECNICOS

```
1 df_aspectos_tecnicos = pd.read_excel(ruta_segmento1, sheet_name='3. RESULTA
2 print("3. 'RESULTADOS DEL EJERCICIO' cargados. Primeras 5 filas:")
3 df_aspectos_tecnicos.head()
```

```
3. 'RESULTADOS DEL EJERCICIO' cargados. Primeras 5 filas:
/usr/local/lib/python3.12/dist-packages/openpyxl/worksheet/_reader.py:329: UserW
  warn(msg)
```

	Unnamed: 0		15 DE ABRIL LTDA	23 DE JULIO LIMITADA	29 DE OCTUBRE LTDA	9 DE OCTUBRE LTDA	JA LE
0	5	Ingresos	13311454.95	48247840.25	1.201518e+08	13563505.43	163
1	51	Intereses y descuentos ganados	10756423.83	41868717.33	1.054787e+08	12615572.14	152
2	(-) 41	Intereses causados	5673237.15	25450929.49	6.340963e+07	6952973.27	92
3	NaN	MARGEN NETO DE INTERESES	5083186.68	16417787.84	4.206908e+07	5662598.87	59
4	(+) 52	Comisiones ganadas	0.00	159313.57	5.582783e+06	0.00	3

5 rows × 46 columns

ESTADOS FINANCIEROS

```
df_estado_financiero = pd.read_excel(ruta_segmento1, sheet_name='2. ESTADO
print("Hoja '2. ESTADO FINANCIERO'. Primeras 5 filas:")
df_estado_financiero.head()
```

```
Hoja '2. ESTADO FINANCIERO'. Primeras 5 filas:
/usr/local/lib/python3.12/dist-packages/openpyxl/worksheet/_reader.py:329: UserW
  warn(msg)
```

	COD CONTABLE	Nombre de Cuenta	TIPO*	GRUPO**	15 DE ABRIL LTDA	23 DE JULIO LIMITADA	29 OCTUB LT
0	1	ACTIVO	1	1	1.293271e+08	4.437386e+08	1.144214e+
1	11	FONDOS DISPONIBLES	1	2	1.442083e+07	7.201578e+07	1.436635e+
2	1101	Caja	1	4	1.193649e+06	6.805766e+06	8.360886e+
3	110105	Efectivo	1	6	1.193449e+06	6.794486e+06	8.316997e+
4	110110	Caja chica	1	6	2.000000e+02	1.128000e+04	4.388895e+

5 rows × 48 columns

INDICADORES FINANCIEROS

↳ 7 cells hidden

Ranking

Quitar todos los na, saltos de linea y espacios

```
df_ranking = df_ranking.dropna(how="all")
df_ranking = df_ranking.applymap(lambda x: str(x).replace("\n", " ").strip(
df_ranking
```

```
/tmp/ipython-input-15491972.py:2: FutureWarning: DataFrame.applymap has been dep
  df_ranking = df_ranking.applymap(lambda x: str(x).replace("\n", " ").strip() i
```

	Unnamed: 0	Unnamed: 1	Unnam
0	No.	RUC	INSTITUCIÓN FINANC
1	1	1768168480001	CORPORACIÓNNACIONALDEFINANZAS\rPOPULARES Y
3	No.	RUC	INSTITUCIÓN FINANC
4	2	1890012015001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PAR
5	3	0190006247001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PAR
...	...	...	
61	58	0790024656001	COOPERATIVADEAHORROYCRÉDITOSANTA\rROSA LIMI
62	59	0490002669001	COOPERATIVADEAHORRO YCRÉDITO TULCÁN\rLIMI
63	60	1190068389001	COOPERATIVADEAHORROYCRÉDITO\rVICENTINAMANUE
64	61	0591711164001	COOPERATIVADEAHORROYCRÉDITOVIRGEN\rDEL C
65	62	0391005664001	COOPERATIVADEAHORROYCRÉDITOYUYAY\

65 rows × 10 columns

Next steps: Generate code with df_ranking | New interactive sheet

Modificar para los encabezados

```
header = df_ranking.iloc[0].tolist()     # fila 0 = encabezado
df = df_ranking[1:].copy()               # borrar fila de encabezado
df.columns = header                    # asignar encabezado

df
```

	No.	RUC	INSTITUCIÓN FINANCIERA
1	1	1768168480001	CORPORACIÓNNACIONALDEFINANZAS\rPOPULARES Y SOL...
3	No.	RUC	INSTITUCIÓN FINANCIERA
4	2	1890012015001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
5	3	0190006247001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
6	4	1090056286001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
...	...	...	...
61	58	0790024656001	COOPERATIVADEAHORROYCRÉDITOSANTA\rROSA LIMITADA.
62	59	0490002669001	COOPERATIVADEAHORRO YCRÉDITO TULCÁN\rLIMITADA.
63	60	1190068389001	COOPERATIVADEAHORROYCRÉDITO\rVICENTINAMANUELES...
64	61	0591711164001	COOPERATIVADEAHORROYCRÉDITOVIRGEN\rDEL CISNE

Next steps: Generate code with df | New interactive sheet

Limpiar los nombres de las columnas para que sea mas facil manejarlos luego

```
df.columns = [c.replace("\r", " ").replace("  ", " ").strip() for c in df.c

rename_map = {
    "N": "N",
    "RUC": "RUC",
    "INSTITUCIÓN FINANCIERA": "Institucion",
    "SEGMENTO": "Segmento",
    "FIRMA CALIFICADORA DE RIESGO": "Firma",
    "AL 30 DE JUNIO 2024": "Jun2024",
    "AL 30 DE SEPTIEMBRE 2024": "Sep2024",
    "AL 31 DE DICIEMBRE 2024": "Dic2024",
    "AL 31 DE MARZO 2025": "Mar2025",
    "AL 30 DE JUNIO 2025": "Jun2025"
}

df = df.rename(columns=rename_map)
df
```

	No.	RUC	Institucion
1	1	1768168480001	CORPORACIÓNNACIONALDEFINANZAS\rPOPULARES Y SOL...
3	No.	RUC	INSTITUCIÓN FINANCIERA
4	2	1890012015001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
5	3	0190006247001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
6	4	1090056286001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
...	...	...	...
61	58	0790024656001	COOPERATIVADEAHORROYCRÉDITOSANTA\rROSA LIMITADA.
62	59	0490002669001	COOPERATIVADEAHORRO YCRÉDITO TULCÁN\rLIMITADA.
63	60	1190068389001	COOPERATIVADEAHORROYCRÉDITO\rVICENTINAMANUELES...
64	61	0591711164001	COOPERATIVADEAHORROYCRÉDITOVIRGEN\rDEL CISNE

Next steps: Generate code with df | New interactive sheet

```
1 df
```

	No.	RUC	Institucion
1	1	1768168480001	CORPORACIÓNNACIONALDEFINANZAS\rPOPULARES Y SOL...
3	No.	RUC	INSTITUCIÓN FINANCIERA
4	2	1890012015001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
5	3	0190006247001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
6	4	1090056286001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
...	...	...	...
61	58	0790024656001	COOPERATIVADEAHORROYCRÉDITOSANTA\rROSA LIMITADA.
62	59	0490002669001	COOPERATIVADEAHORRO YCRÉDITO TULCÁN\rLIMITADA.
63	60	1190068389001	COOPERATIVADEAHORROYCRÉDITO\rVICENTINAMANUELES...
64	61	0591711164001	COOPERATIVADEAHORROYCRÉDITOVIRGEN\rDEL CISNE

Next steps: Generate code with df New interactive sheet

Limpiar columnas

```
def limpiar(s):
    if pd.isna(s):
        return None
    s = str(s)
```

```
    s = s.replace("–", "-").replace("—", "-")  # Normalizar guiones
    s = s.upper()
    s = s.replace("´", "'")
    s = re.sub(r"\s*\|\s*", " | ", s)         # Normalizar separador |
    s = re.sub(r"\s*-\s*", "-", s)            # Normalizar guiones con espa
    s = re.sub(r"\s{2,}", " ", s)             # Quitar múltiples espacios
    return s.strip()

df = df.applymap(limpiar)
df
```

```
/tmp/ipython-input-2218334321.py:13: FutureWarning: DataFrame.applymap has been
  df = df.applymap(limpiar)
```

	No.	RUC	Institucion
1	1	1768168480001	CORPORACIÓNNACIONALDEFINANZAS\rPOPULARES Y SOL...
3	NO.	RUC	INSTITUCIÓN FINANCIERA
4	2	1890012015001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
5	3	0190006247001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
6	4	1090056286001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...
...	...	...	...
61	58	0790024656001	COOPERATIVADEAHORROYCRÉDITOSANTA\rROSA LIMITADA.
62	59	0490002669001	COOPERATIVADEAHORRO YCRÉDITO TULCÁN\rLIMITADA.
63	60	1190068389001	COOPERATIVADEAHORROYCRÉDITO\rVICENTINAMANUELES...
64	61	0591711164001	COOPERATIVADEAHORROYCRÉDITOVIRGEN\rDEL CISNE

Next steps: Generate code with df | New interactive sheet

Filtrar solo para segmento 1

```
#Normalizar la columna 'Segmento' para extraer solo el número
if 'Segmento' not in df.columns:
```

```
    raise ValueError("No se encontró la columna 'Segmento' en df.columns: "

#Extrae el primer número que aparezca en la celda (ej: 'SEGMENTO 1', 'SEGME
df['Segmento'] = df['Segmento'].astype(str).str.extract(r'(\d+)')[0]
# Si prefieres mantener como int:
df['Segmento'] = df['Segmento'].astype(float).astype('Int64')  # permite NA

# Limpiar columnas de calificación (quitar asteriscos y quedarnos con el tc
rating_cols = [c for c in df.columns if re.search(r'Jun|Sep|Dic|Mar|Jun', c
for c in rating_cols:
    # nos quedamos con la última "palabra" (p.ej. 'AAA- *******' -> 'AAA-')
    df[c] = (df[c].astype(str)
                .str.split()
                .str[0]           # o str[-1] según el patrón; usar str[0]
                .str.replace(r'\*+', '', regex=True)
                .str.strip()
                .replace('', pd.NA))

# filtrar Segmento == 1
df_ranking = df[df['Segmento'] == 1].copy()

print("Shape total limpio:", df.shape)
print("Shape df_ranking (Segmento 1):", df_ranking.shape)
display(df_ranking.head(10))
```

```
Shape total limpio: (64, 10)
Shape df_ranking (Segmento 1): (49, 10)
```

	No.	RUC	Institucion	Segmen
1	1	1768168480001	CORPORACIÓNNACIONALDEFINANZAS\rPOPULARES Y SOL...	
4	2	1890012015001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...	
5	3	0190006247001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA...	
6	4	1090056286001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA LA	

Usar solo la última evaluación y limpieza de texto

```
cols_fijas = ["No.", "RUC", "Institucion", "Segmento", "Firma"]

```

```
ultima_columna = df_ranking.columns[-1]

# Crear un nuevo DataFrame con las columnas fijas + la última evaluación
df_ranking = df_ranking[cols_fijas + [ultima_columna]].copy()

# --- Función para limpiar y convertir ratings (promedio si hay "|")
def limpiar_texto_rating(s):
    if pd.isna(s):
        return None
    s = str(s)
    s = s.replace("−", "-").replace("—", "-")  # Normalizar guiones
    s = s.upper()
    s = s.replace("´", "'")
    s = re.sub(r"\s*\|\s*", " | ", s)         # Normalizar separador |
    s = re.sub(r"\s*-\s*", "-", s)            # Normalizar guiones con espa
    s = re.sub(r"\s{2,}", " ", s)             # Quitar múltiples espacios
    return s.strip()
df_ranking[ultima_columna] = df_ranking[ultima_columna].apply(limpiar_textc
df_ranking
```

Next steps: Generate code with df_ranking | New interactive sheet

4	2	1890012015001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA L
5	3	0190006247001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA L
6	4	1090056286001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA L
7	5	1790075494001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA L
8	6	1791708040001	CAJA CENTRAL FINANCO
9	7	1390013678001	COOPERATIVADEAHORROYCREDITO15DE\rABRIL LT
10	8	1790093204001	COOPERATIVADEAHORROYCREDITO23DE\rJULIO LIMITA
11	9	1790567699001	COOPERATIVADEAHORROYCREDITO29DE\rOCTUBRE LT

Cambiar los nombre por números

```
rating_scale = {
    "AAA": 1, "AAA-": 1,
    "AA+": 2, "AA": 2, "AA-": 2,
    "A+": 3, "A": 3, "A-": 3,
    "BBB+": 4, "BBB": 4, "BBB-": 4,
    "BB+": 5, "BB": 5, "BB-": 5,
    "B+": 5, "B": 5, "B-": 5,
}

def convertir_rating(valor, escala):
    if pd.isna(valor):
        return None

    # Normalizar texto
    valor = str(valor).replace("\r", " ").replace("\n", " ").strip()

```

```
    # Convertir cualquier separador (/ | \) en "|"
    valor = re.sub(r"\s*[/|]\s*", "|", valor)
    valor = re.sub(r"\s+", " ", valor)

    # Caso simple
    if "|" not in valor:
        return escala.get(valor, None)

    # Caso con dos ratings
    partes = [p.strip() for p in valor.split("|")]
    valores = [escala.get(p, None) for p in partes]
    valores = [v for v in valores if v is not None]

    if not valores:
        return None

    return sum(valores) / len(valores)


df_ranking["rating_num"] = df_ranking[ultima_columna].apply(
    lambda x: convertir_rating(x, rating_scale)
)

# Eliminar filas cuyo rating no existe (None)
df_ranking_filtrado = df_ranking[df_ranking["rating_num"].notna()].copy()
df_ranking_ordenado = df_ranking_filtrado.sort_values("rating_num").reset_i
df_ranking_ordenado
```

26	24	0590027920001	COOPERATIVADEAHORROYCREDITODELA\rPEQUEÑA EMPRI
27	25	1190075539001	COOPERATIVADEAHORROYCREDITODELA\rPEQUEÑA EMPRI
28	26	0590052000001	COOPERATIVADEAHORROYCREDITODELA\rPEQUEÑA EMPRI
29	27	1690012606001	COOPERATIVADEAHORROYCREDITODELA\rPEQUEÑA EMPRI

0	1	1760168480001	CORPORACIONNACIONALDEFINANZAS\rPOPULARES Y SO
1	6	1791708040001	CAJA CENTRAL FINANCO

Next steps: Generate code with df_ranking_ordenado | New interactive sheet

2	26	0590052000001	COOPERATIVADEAHORROYCREDITODELA\rPEQUEÑA EMPRI
3	55	1890003628001	COOPERATIVADEAHORROYCRÉDITOSAN\rFRANCISCO LTD
4	15	1790325083001	COOPERATIVADEAHORROYCREDITO\rANDALUCIA LIMITA
5	16	1090033456001	COOPERATIVADEAHORROYCREDITO\rATUNTAQUI LIMITA
6	12	1790501469001	COOPERATIVA DE AHORRO Y CREDITO ALIANZA\rDEL '
7	5	1790075494001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA L/

Indicadores Financieros

8	29	0690045389001	COOPERATIVADEAHORROYCREDITO\rRIOBAMBA LT

```
df_indicadores_financieros
```

9	30	1790979016001	COOPERATIVADEAHORROYCREDITODELOS\rSERVIDORESPU
10	33	1890037646001	COOPERATIVADEAHORROYCREDITOEL\rSAGRARIO LT
11	38	0190155722001	COOPERATIVADEAHORROYCREDITOJARDIN\rAZUAYO LIM
12	27	1690012606001	COOPERATIVADEAHORROYCREDITODELA\rPEQUEÑA EMPRI

13	9	1790567699001	COOPERATIVADEAHORROYCREDITO29DE\rOCTUBRE LT
14	39	0190115798001	COOPERATIVADEAHORROYCRÉDITO\rJUVENTUDECUATORI/
15	3	0190006247001	ASOCIACIONMUTUALISTADEAHORROY\rCREDITO PARA L/
16	46	1890001323001	COOPERATIVADEAHORROYCRÉDITOOSCUS\rLIMITAI
17	56	0290003288001	COOPERATIVADEAHORROYCRÉDITOSAN\rJOSÉ LIMITAI
18	61	0591711164001	COOPERATIVADEAHORROYCRÉDITOVIRGEN\rDEL CIS
19	59	0490002669001	COOPERATIVADEAHORRO YCRÉDITO TULCÁN\rLIMITAI
20	35	0691706710001	COOPERATIVADEAHORROYCREDITO\rFERNANDO DAQUILEI
21	34	0190024733001	COOPERATIVADEAHORROYCREDITOERCO\rLIMITA

	15 DE ABRIL LTDA	23 DE JULIO LIMITADA	29 DE OCTUBRE LTDA	9 DE OCTUBRE LTDA	ALFONSO JARAMILLO LEON CAJA	A DEL LT
(PATRIMONIO + RESULTADOS) / ACTIVOS INMOVILIZADOS	1.320542	2.076457	2.183547	2.788390	1.460808	1
ACTIVOS IMPRODUCTIVOS NETOS / TOTAL ACTIVOS	0.123102	0.091748	0.073416	0.078901	0.094265	0
ACTIVOS PRODUCTIVOS / TOTAL ACTIVOS	0.876898	0.908252	0.926584	0.921099	0.905735	0
ACTIVOS PRODUCTIVOS / PASIVOS CON COSTO	1.027844	1.044289	1.039765	1.080646	1.014125	1
MOROSIDAD DE LA CARTERA DE CREDITO PRODUCTIVO	0.111285	0.020040	0.055395	0.000000	0.080890	0
MOROSIDAD DE LA CARTERA DE CONSUMO	0.070948	0.074157	0.088975	0.036601	0.053926	0
MOROSIDAD DE LA CARTERA DE CREDITO INMOBILIARIO	0.029615	0.101904	0.022915	0.000005	0.000000	0
MOROSIDAD DE LA CARTERA DE MICROCREDITO	0.182316	0.119429	0.196391	0.125266	0.167566	0
MOROSIDAD DE LA CARTERA DE VIVIENDA DE INTERES SOCIAL Y PUBLICO	0.000000	0.000000	0.000000	0.000000	0.000000	0
MOROSIDAD DE LA CARTERA DE CREDITO EDUCATIVO	0.000000	0.000000	0.000000	0.000000	0.000000	0
MOROSIDAD DE LA CARTERA TOTAL	0.107272	0.093425	0.099000	0.096313	0.067223	0

COBERTURA DE LA CARTERA DE CREDITO PRODUCTIVO	1.111456	1.377552	1.000000	0.000000	1.110084	0
COBERTURA DE LA CARTERA DE CREDITO CONSUMO	1.101038	1.133063	1.108068	1.172375	1.022448	1
COBERTURA DE LA CARTERA DE CREDITO INMOBILIARIO	0.631811	0.899309	1.051257	1276.490000	0.000000	563
COBERTURA DE LA CARTERA DE MICROCREDITO	1.007935	1.033881	1.032547	0.934708	1.002167	1
COBERTURA DE LA CARTERA DE VIVIENDA DE IINTERES PUBLICO	0.000000	0.000000	0.000000	0.000000	0.000000	0
COBERTURA DE LA CARTERA DE CREDITO EDUCATIVO	0.000000	0.000000	0.000000	0.000000	0.000000	0
COBERTURA DE LA CARTERA PROBLEMÁTICA	1.023424	1.076982	1.080869	0.963912	1.019509	1
GASTOS DE OPERACION ESTIMADOS / TOTAL ACTIVO PROMEDIO	0.055945	0.043137	0.029980	0.038264	0.043149	0
GASTOS DE OPERACION / MARGEN FINANCIERO	1.348749	1.306153	1.051972	1.119722	0.995780	1
GASTOS DE PERSONAL ESTIMADOS / ACTIVO PROMEDIO	0.023108	0.018401	0.012743	0.017336	0.022177	0
RESULTADOS DEL EJERCICIO / PATRIMONIO PROMEDIO	0.005227	0.013795	0.005820	0.006256	0.012872	0
RESULTADOS DEL EJERCICIO / ACTIVO PROMEDIO	0.000585	0.001484	0.000520	0.000738	0.001185	0
CARTERA BRUTA / (DEPOSITOS A LA	0.792486	0.796689	0.856153	0.822414	0.871291	0

VISTA + DEPOSITOS A PLAZO)	0.792466	0.796069	0.830155	0.822414	0.871291	0
MARGEN DE INTERMEDIACIÓN ESTIMADO / PATRIMONIO PROMEDIO	-0.129207	-0.093969	-0.016568	-0.034671	0.001986	-0
MARGEN DE INTERMEDIACIÓN ESTIMADO / ACTIVO PROMEDIO	-0.014466	-0.010111	-0.001481	-0.004091	0.000183	-0
RENDIMIENTO DE LA CARTERA DE CREDITO PRODUCTIVO POR VENCER	0.080513	0.124139	0.083961	0.000000	0.110231	0
RENDIMIENTO DE LA CARTERA DE CREDITO CONSUMO	0.148228	0.139956	0.137721	0.131738	0.149500	0
RENDIMIENTO DE LA CARTERA DE CREDITO INMOBILIARIO POR VENCER	0.098867	0.094859	0.093932	0.090679	0.000000	0
RENDIMIENTO DE LA CARTERA DE MICROCREDITO POR VENCER	0.164794	0.169111	0.163811	0.168581	0.157327	0
RENDIMIENTO DE LA CARTERA DE VIVIENDA DE IINTERES PUBLICO POR VENCER	0.000000	0.000000	0.000000	0.000000	0.000000	0
RENDIMIENTO DE LA CARTERA DE CREDITO EDUCATIVO POR VENCER	0.000000	0.000000	0.000000	0.000000	0.000000	0
CARTERAS DE CRÉDITOS REFINANCIADAS	0.144911	0.162655	0.146202	0.212391	0.197214	0
CARTERAS DE CRÉDITOS REESTRUCTURADAS	0.157736	0.179990	0.137274	0.249426	0.164180	0
CARTERA POR VENCER TOTAL	0.154540	0.161925	0.150601	0.162517	0.151499	0
FONDOS						

DISPONIBLES / TOTAL DEPOSITOS A CORTO PLAZO	0.124214	0.138819	0.278446	0.208584	0.257748	0
CARTERA						

Hay que dividir los datos para 100, porque en el documento aparece esto como porcentaje

```
# Copia del DataFrame original
df_fixed = df_indicadores_financieros.copy()

# 1. Reemplazar % y convertir todo a número (coercing evita errores)
df_fixed = df_fixed.replace('%','', regex=True)
df_fixed = df_fixed.apply(pd.to_numeric, errors='coerce')

# 2. Convertir porcentajes (100% -> 1.0)
df_fixed = df_fixed / 100

# 3. Mostrar primeras filas
df_fixed.head()
```

FI = 1 + (ACTIVOS IMPRODUCTIVOS / ACTIVOS TOTALES)	1.123102	1.091748	1.073416	1.078901	1.094265	1
INDICE DE CAPITALIZACION NETO: FK / FI	0.082552	0.088861	0.078577	0.109300	0.078634	0

	15 DE ABRIL LTDA	23 DE JULIO LIMITADA	29 DE OCTUBRE LTDA	9 DE OCTUBRE LTDA	ALFONSO JARAMILLO LEON CAJA	ALIANZA DEL VALLE LIMITADA	A
(PATRIMONIO + RESULTADOS) / ACTIVOS INMOVILIZADOS	0.013205	0.020765	0.021835	0.027884	0.014608	0.016484	0.05
ACTIVOS IMPRODUCTIVOS NETOS / TOTAL ACTIVOS	0.001231	0.000917	0.000734	0.000789	0.000943	0.000945	0.0
ACTIVOS PRODUCTIVOS / TOTAL ACTIVOS	0.008769	0.009083	0.009266	0.009211	0.009057	0.009055	0.0
ACTIVOS PRODUCTIVOS / PASIVOS CON COSTO	0.010278	0.010443	0.010398	0.010806	0.010141	0.010461	0.01
MOROSIDAD DE LA CARTERA DE CREDITO PRODUCTIVO	0.001113	0.000200	0.000554	0.000000	0.000809	0.000000	0.0

5 rows × 44 columns

```
!pip install unidecode

```

```
Requirement already satisfied: unidecode in /usr/local/lib/python3.12/dist-packa
```

```
import unidecode


def normalizar_texto(valor):
    if not isinstance(valor, str):
        return valor  # no modificar números
    valor = valor.lower()
    valor = unidecode.unidecode(valor)
    valor = re.sub(r'[^a-z0-9]+', '_', valor)
    valor = re.sub(r'_+', '_', valor)
    valor = valor.strip('_')
    return valor

df_indicadores_financieros.index = df_indicadores_financieros.index.map(nor

df_indicadores_financieros.columns = df_indicadores_financieros.columns.map

print("Normalización completada para índices y columnas de df_indicadores_f
display(df_indicadores_financieros.head())
```

```
Normalización completada para índices y columnas de df_indicadores_financieros.
```

	15_de_abril_ltda	23_de_julio_limi
patrimonio_resultados_activos_inmovilizados	1.320542	2.076
activos_improductivos_netos_total_activos	0.123102	0.09
activos_productivos_total_activos	0.876898	0.908
activos_productivos_pasivos_con_costo	1.027844	1.044
morosidad_de_la_cartera_de_credito_productivo	0.111285	0.020

5 rows × 44 columns

```
df_indicadores_financieros.index
```

```
Index(['patrimonio_resultados_activos_inmovilizados',
       'activos_improductivos_netos_total_activos',
       'activos_productivos_total_activos',
       'activos_productivos_pasivos_con_costo',
       'morosidad_de_la_cartera_de_credito_productivo',
       'morosidad_de_la_cartera_de_consumo',
       'morosidad_de_la_cartera_de_credito_inmobiliario',
       'morosidad_de_la_cartera_de_microcredito',
```

```
       'morosidad_de_la_cartera_de_vivienda_de_interes_social_y_publico',
       'morosidad_de_la_cartera_de_credito_educativo',
       'morosidad_de_la_cartera_total',
       'cobertura_de_la_cartera_de_credito_productivo',
       'cobertura_de_la_cartera_de_credito_consumo',
       'cobertura_de_la_cartera_de_credito_inmobiliario',
       'cobertura_de_la_cartera_de_microcredito',
       'cobertura_de_la_cartera_de_vivienda_de_iinteres_publico',
       'cobertura_de_la_cartera_de_credito_educativo',
       'cobertura_de_la_cartera_problematica',
       'gastos_de_operacion_estimados_total_activo_promedio',
       'gastos_de_operacion_margen_financiero',
       'gastos_de_personal_estimados_activo_promedio',
       'resultados_del_ejercicio_patrimonio_promedio',
       'resultados_del_ejercicio_activo_promedio',
       'cartera_bruta_depositos_a_la_vista_depositos_a_plazo',
       'margen_de_intermediacion_estimado_patrimonio_promedio',
       'margen_de_intermediacion_estimado_activo_promedio',
       'rendimiento_de_la_cartera_de_credito_productivo_por_vencer',
       'rendimiento_de_la_cartera_de_credito_consumo',
       'rendimiento_de_la_cartera_de_credito_inmobiliario_por_vencer',
       'rendimiento_de_la_cartera_de_microcredito_por_vencer',
       'rendimiento_de_la_cartera_de_vivienda_de_iinteres_publico_por_vencer',
       'rendimiento_de_la_cartera_de_credito_educativo_por_vencer',
       'carteras_de_creditos_refinanciadas',
       'carteras_de_creditos_reestructuradas', 'cartera_por_vencer_total',
       'fondos_disponibles_total_depositos_a_corto_plazo',
       'cartera_improductiva_descubierta_patrimonio_resultados',
       'cartera_improductiva_patrimonio_dic',
       'fk_patrimonio_resultados_ingresos_extraordinarios_activos_totales',
       'fi_1_activos_improductivos_activos_totales',
       'indice_de_capitalizacion_neto_fk_fi'],
      dtype='object', name='    ')
```

Escalado con Standard Scaler

```
1 df_indicadores_financieros
```

	15_de_abril_
patrimonio_resultados_activos_inmovilizados	1.32
activos_improductivos_netos_total_activos	0.12
activos_productivos_total_activos	0.87
activos_productivos_pasivos_con_costo	1.02
morosidad_de_la_cartera_de_credito_productivo	0.11
morosidad_de_la_cartera_de_consumo	0.07
morosidad_de_la_cartera_de_credito_inmobiliario	0.02
morosidad_de_la_cartera_de_microcredito	0.18
morosidad_de_la_cartera_de_vivienda_de_interes_social_y_publico	0.00
morosidad_de_la_cartera_de_credito_educativo	0.00
morosidad_de_la_cartera_total	0.10
cobertura_de_la_cartera_de_credito_productivo	1.11

```
from sklearn.preprocessing import StandardScaler

X = df_indicadores_financieros.copy()

# Convertir todo a numérico por seguridad
X = X.apply(pd.to_numeric, errors='coerce')

# Llenar valores faltantes
X = X.fillna(0)

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

df_indicadores_financieros_scaled = pd.DataFrame(
    X_scaled,
    index=X.index,
    columns=X.columns
)

df_indicadores_financieros_scaled
```

margen_de_intermediacion_estimado_activo_promedio	-0.01
rendimiento_de_la_cartera_de_credito_productivo_por_vencer	0.08
rendimiento_de_la_cartera_de_credito_consumo	0.14
rendimiento_de_la_cartera_de_credito_inmobiliario_por_vencer	0.09
rendimiento_de_la_cartera_de_microcredito_por_vencer	0.16

rendimiento_de_la_cartera_de_vivienda_de_iinteres_publico_por_vencer	0.00
rendimiento_de_la_cartera_de_credito_educativo_por_vencer	0.00
carteras_de_creditos_refinanciadas	0.14
carteras_de_creditos_reestructuradas	0.15
cartera_por_vencer_total	0.15
fondos_disponibles_total_depositos_a_corto_plazo	0.24
cartera_improductiva_descubierta_patrimonio_resultados	0.00
cartera_improductiva_patrimonio_dic	0.65
fk_patrimonio_resultados_ingresos_extraordinarios_activos_totales	0.09
fi_1_activos_improductivos_activos_totales	1.12
indice_de_capitalizacion_neto_fk_fi	0.08

41 rows × 44 columns

	15_de_abril_
patrimonio_resultados_activos_inmovilizados	2.19
activos_improductivos_netos_total_activos	-0.48
activos_productivos_total_activos	1.20
activos_productivos_pasivos_con_costo	1.53
morosidad_de_la_cartera_de_credito_productivo	-0.51
morosidad_de_la_cartera_de_consumo	-0.60
morosidad_de_la_cartera_de_credito_inmobiliario	-0.69
morosidad_de_la_cartera_de_microcredito	-0.35
morosidad_de_la_cartera_de_vivienda_de_interes_social_y_publico	-0.76
morosidad_de_la_cartera_de_credito_educativo	-0.76
morosidad_de_la_cartera_total	-0.52
cobertura_de_la_cartera_de_credito_productivo	1.72
cobertura_de_la_cartera_de_credito_consumo	1.70
cobertura_de_la_cartera_de_credito_inmobiliario	0.65
cobertura_de_la_cartera_de_microcredito	1.49
cobertura_de_la_cartera_de_vivienda_de_iinteres_publico	-0.76
cobertura_de_la_cartera_de_credito_educativo	-0.76
cobertura_de_la_cartera_problematica	1.52
gastos_de_operacion_estimados_total_activo_promedio	-0.63
gastos_de_operacion_margen_financiero	2.25
gastos_de_personal_estimados_activo_promedio	-0.71
resultados_del_ejercicio_patrimonio_promedio	-0.75
resultados_del_ejercicio_activo_promedio	-0.76
cartera_bruta_depositos_a_la_vista_depositos_a_plazo	1.01
margen_de_intermediacion_estimado_patrimonio_promedio	-1.05
margen_de_intermediacion_estimado_activo_promedio	-0.79
rendimiento_de_la_cartera_de_credito_productivo_por_vencer	-0.58
rendimiento_de_la_cartera_de_credito_consumo	-0.43
rendimiento_de_la_cartera_de_credito_inmobiliario_por_vencer	-0.54
rendimiento_de_la_cartera_de_microcredito_por_vencer	-0.39

Análisis exploratorio (EDA)

↳ 11 cells hidden

TSNE

↳ 5 cells hidden

Modelado

KMeans

↳ 13 cells hidden

Gaussian Mixture Model

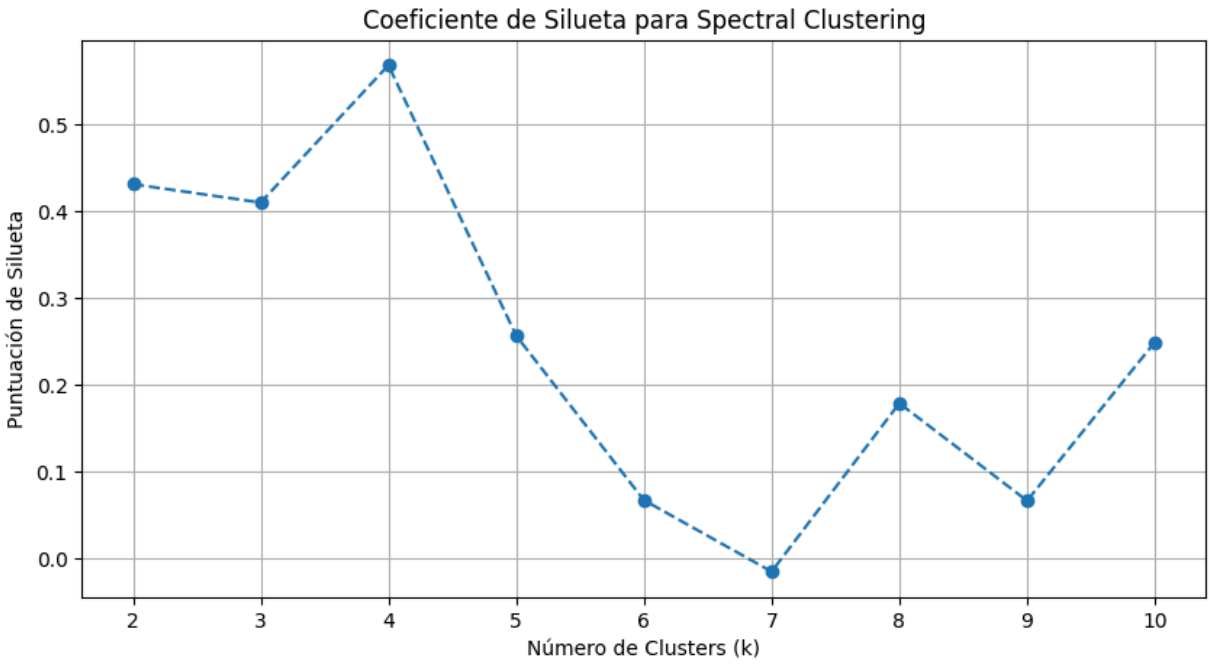
↳ 4 cells hidden

Spectral Clusting

rendimiento_de_la_cartera_de_vivienda_de_iinteres_publico_por_vencer -0.76

```
from sklearn.cluster import SpectralClustering
from sklearn import metrics

k_range = range(2, 11)
silhouette_scores = []

print("Calculando Silueta para Spectral Clustering...")
for k in k_range:
    # Usamos los mismos parámetros que el ajuste anterior
    spectral_clustering = SpectralClustering(
        n_clusters=k,
        assign_labels='kmeans',
        affinity='rbf',
        gamma=1.0, # Ajusta esto si es necesario
        random_state=42,
        n_init=10
    )
    # Ajustar y predecir
    labels = spectral_clustering.fit_predict(X)

    # Calcular la Silueta (solo si k > 1)
    score = metrics.silhouette_score(X, labels)
    silhouette_scores.append(score)

# Encontrar el k con el score de Silueta más alto
optimal_k_spectral = k_range[np.argmax(silhouette_scores)]
print(f"El k óptimo para Spectral Clustering (máxima Silhouette) es: {optim

# Graficar la Silueta
plt.figure(figsize=(10, 5))
plt.plot(k_range, silhouette_scores, marker='o', linestyle='--')
plt.title("Coeficiente de Silueta para Spectral Clustering")
plt.xlabel("Número de Clusters (k)")
plt.ylabel("Puntuación de Silueta")
plt.xticks(k_range)
plt.grid(True)
plt.show()
```

```
Calculando Silueta para Spectral Clustering...
El k óptimo para Spectral Clustering (máxima Silhouette) es: 4
```



Como en los anteriores se ligio 5, para que sea más comparable, en este igual se trabajara con 5 clusters porque tenemos 5 divisiones de cooperativas. Aunque aquí nos indique que lo más óptimo es 4.

```
n_neighbors_candidates = [5, 8, 10, 12, 15, 20]
from sklearn.metrics import adjusted_rand_score, silhouette_score

for nn in n_neighbors_candidates:
    sc = SpectralClustering(
        n_clusters=k,
        affinity="nearest_neighbors",
        n_neighbors=nn,
        assign_labels="kmeans",
        random_state=42
    )
    labels = sc.fit_predict(X_tsne)
    ari = adjusted_rand_score(labels_K, labels)
    sil = silhouette_score(X_tsne, labels)
    print(f"n_neighbors={nn}  ARI={ari:.4f}  Silhouette={sil:.4f}")
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/manifold/_spectral_embedding.py:
  warnings.warn(
n_neighbors=5  ARI=0.5211  Silhouette=0.5131
n_neighbors=8  ARI=0.5592  Silhouette=0.4130
n_neighbors=10  ARI=0.4860  Silhouette=0.3283
n_neighbors=12  ARI=0.4943  Silhouette=0.3917
n_neighbors=15  ARI=0.4640  Silhouette=0.2337
n_neighbors=20  ARI=0.4185  Silhouette=0.1795
```

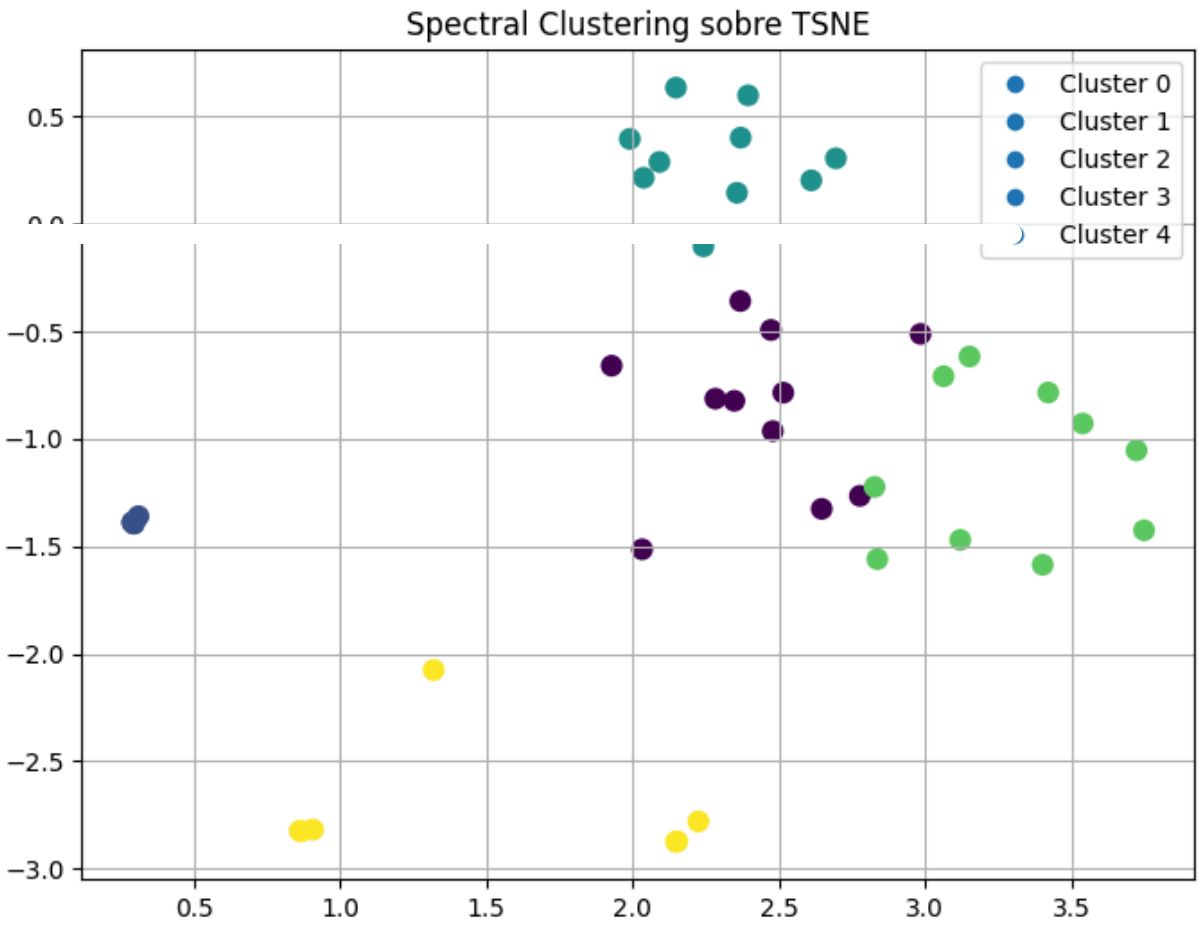
Como el ARI más alto y el Silhouette mas alto es cuando hay 8 neighbors, se elegiran 8.

```
from sklearn.cluster import SpectralClustering

# Número de clusters
k_spectral = 5

# Definir Spectral Clustering sobre TSNE
spectral_tsne = SpectralClustering(
    n_clusters=k_spectral,
    affinity='nearest_neighbors',
    n_neighbors= 8,
    assign_labels='kmeans',
    random_state=42,
    n_init=10
)

labels_spectral= spectral_tsne.fit_predict(X_tsne)

# Crear dataframe copiando indicadores y añadiendo clusters spectral sobre
df_clusters_spectral = df_indicadores_financieros_scaled_trasp.copy()
df_clusters_spectral["cluster_spectral_tsne"] = labels_spectral

# Ver distribución
print(df_clusters_spectral["cluster_spectral_tsne"].value_counts())
```

```
cluster_spectral_tsne
0    11
3    10
2    10
4     7
1     6
Name: count, dtype: int64
```

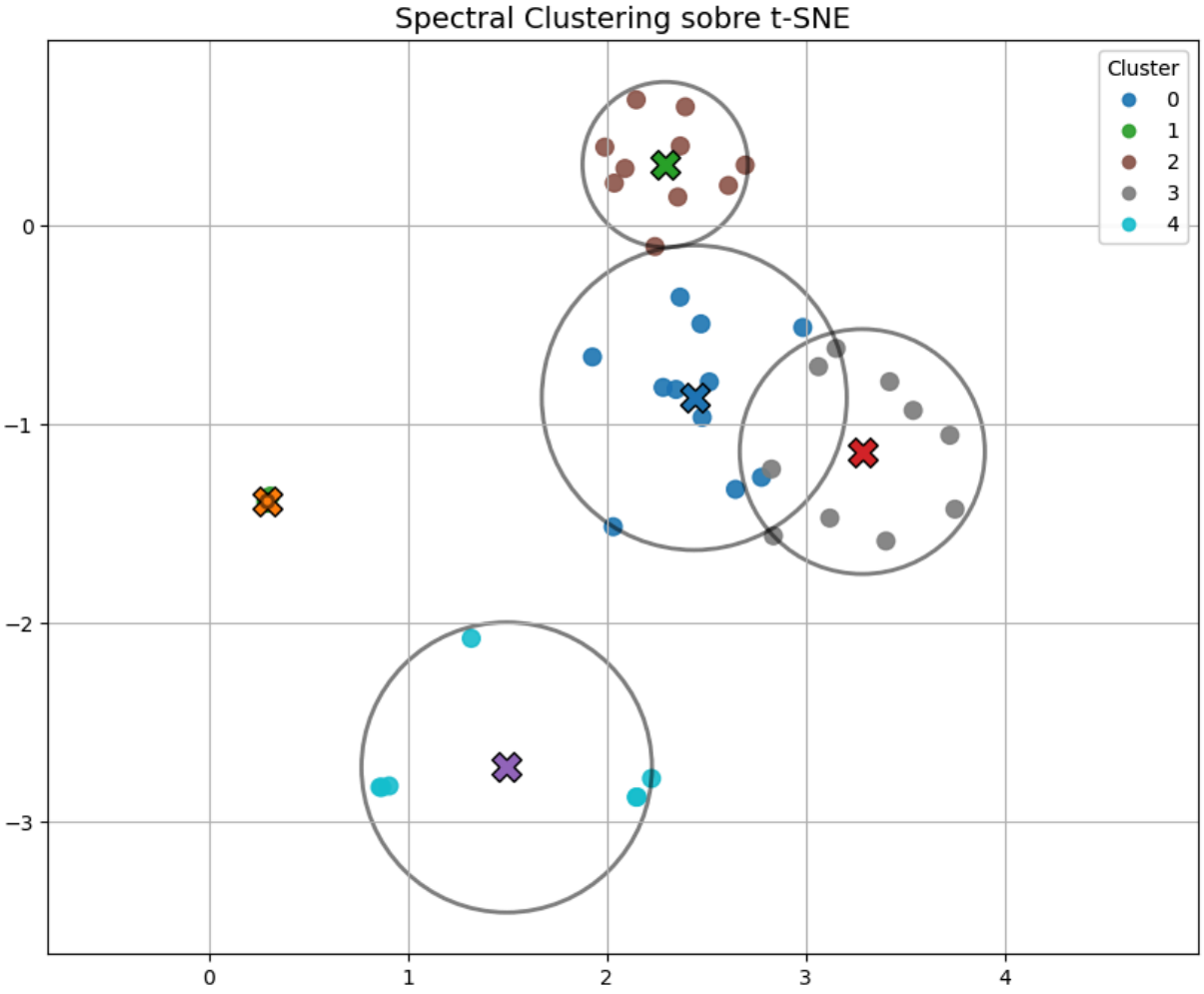
```

plt.figure(figsize=(8,6))
scatter = plt.scatter(X_tsne[:,0], X_tsne[:,1], c=labels_spectral, s=60)
plt.title("Spectral Clustering sobre TSNE")
plt.grid(True)

# Crear leyenda con los nombres de los clusters
```

```
handles = []
unique_labels = np.unique(labels_spectral)
for lab in unique_labels:
    handles.append(plt.Line2D([], [], marker='o', linestyle='', label=f'Clu
plt.legend(handles=handles, loc='best')
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(10, 8))

# Scatter con mejor colormap
scatter = plt.scatter(
    X_tsne[:, 0],
    X_tsne[:, 1],
    c=labels_spectral,
    cmap="tab10",
    s=70,
    alpha=0.9
)
```

```

# Obtener centroides y dibujar círculos reales
for cluster in np.unique(labels_spectral):

    pts = X_tsne[labels_spectral == cluster]
    centroid = pts.mean(axis=0)

    # Dibujar centroide
    plt.scatter(
        centroid[0],
        centroid[1],
        marker="X",
        s=200,
        edgecolors="black"
    )

    # Radio más estable: distancia máxima (no promedio)
    dx = pts[:, 0] - centroid[0]
    dy = pts[:, 1] - centroid[1]
    radius = np.max(np.sqrt(dx**2 + dy**2))

    circle = plt.Circle(
        (centroid[0], centroid[1]),
        radius,
        edgecolor="black",
        fill=False,
        linewidth=2,
        alpha=0.5
    )

    plt.gca().add_patch(circle)

# Título, grilla y formato
plt.title("Spectral Clustering sobre t-SNE", fontsize=14)
plt.grid(True)
plt.gca().set_aspect("equal", adjustable="datalim")

# Leyenda automática
legend1 = plt.legend(*scatter.legend_elements(), title="Cluster")
plt.gca().add_artist(legend1)

plt.show()

```



Evaluación y validación

Comparación de Resultados

Silhouette Score, mide cohesión (qué tan juntos están) y separación (qué tan lejos están de otros clusters). Aquí mientras más cercana a 1 mejor.

```
from sklearn.metrics import silhouette_score

sil_km = silhouette_score(X, labels_K)
sil_gmm = silhouette_score(X, labels_gmm)
sil_spectral = silhouette_score(X, labels_spectral)

print("Silhouette K-Means:", sil_km)
print("Silhouette GMM:", sil_gmm)
print("Silhouette Spectral:", sil_spectral)
```

```
Silhouette K-Means: 0.3724289016235934
Silhouette GMM: 0.4035526757905385
Silhouette Spectral: 0.37524506219056913
```

Con Silhouette se scaraon resultados muy pobres, mientras que los mejores fueron con KMeans, sin decir que fueron escalentes

El metodo de Clinski - Harabasz mide la separación entre clusters (between-cluster dispersion), y cohesión dentro del cluster (within-cluster dispersion). Mientras más alto es mejor.

```
from sklearn.metrics import calinski_harabasz_score

ch_km = calinski_harabasz_score(X, labels_K)
ch_gmm = calinski_harabasz_score(X, labels_gmm)
ch_spectral = calinski_harabasz_score(X, labels_spectral)

print("CH K-Means:", ch_km)
print("CH GMM:", ch_gmm)
print("CH Spectral:", ch_spectral)
```

```
CH K-Means: 24.936789828579904
CH GMM: 23.726184628838197
CH Spectral: 25.00650637750751
```

Con esta métrica también se coincide que KMeans es la mejor y Spectral por mucho la peor.

Mutual Information

Mide qué tan informativo es cada indicador financiero para explicar el rating oficial de riesgo. Si saber el valor de un indicador reduce la incertidumbre sobre el rating entonces tiene Alta Información Mutua. Si no aporta nada su Información Mutua es cercana a cero

El código calcula la Información Mutua entre cada indicador financiero y el rating de riesgo oficial, permitiendo identificar qué variables aportan más información útil para explicar o predecir el rating.

Adjusted Rand Indez (ARI)

```
1 Start coding or generate with AI.
```

Conclusiones

Similitudes y discrepancias entre clusters y ratings.

Definitivamente el que hizo una separación más pareja fue el KMeans, divide claramente en 5 grupos compactos. A simple vista no maneja bien los grupos internos ni alargados, pero es el mejor de los 3 modelos.

Con el GMM hayy un cluster gigante (celeste) que absorbe muchos puntos. Algunos clusters están parcialmente solapados.Indica que hay cooperativas con comportamientos financieros muy parecidos

En el Spectral teoricamente se detectan estructuras no lineales, parece que separa otros clusters Si se hubiera elegido k = 4, que es lo que en la mayoria de modelos sugeria, hubiera habido un rendimiento un poco mejor, pero se hubieran tenido que dividir las cooperativas de otra manera

Hipótesis de los patrones financieros.

Semi Supervised Learning

Alineación de nombres + Matching Ratings–Clusters

Objetivo: Unificar los nombres de cooperativas para poder comparar clusters financieros vs. ratings oficiales.

- Se crearon claves normalizadas (Cooperativa_Key, Institucion_Key)

- Se ejecutó RapidFuzz con partial_ratio (correcto para nombres largos e inconsistentes)
- Se construyó un mapeo confiable (≥90 de score)
- Se generó df_validacion con 42 coincidencias exactas
- Ya tenemos la tabla emparejada con: Cooperativa — Cluster — rating_num

```
!pip install rapidfuzz
```

```
Requirement already satisfied: rapidfuzz in /usr/local/lib/python3.12/dist-packa
```

Construir df_cluster_labels

```
# ==========================================
# RECONSTRUIR df_cluster_labels (como en tu parte 1)
# ==========================================

# df_clusters_km tiene el índice = cooperativa normalizada
# y la columna cluster_km

df_cluster_labels = df_clusters_km.reset_index().rename(columns={
    "index": "Cooperativa",
    "cluster_km": "Cluster"
})

# Dejamos solo las columnas necesarias para el matching
df_cluster_labels = df_cluster_labels[["Cooperativa", "Cluster"]]

print("✓ df_cluster_labels creado correctamente")
display(df_cluster_labels.head())

```

```
✓ df_cluster_labels creado correctamente
```

	Cooperativa	Cluster
0	15_de_abril_ltda	2
1	23_de_julio_limitada	3
2	29_de_octubre_ltda	3
3	9_de_octubre_ltda	1
4	alfonso_jaramillo_leon_caja	2

```
# ==========================================
# BLOQUE 1 — Normalización de nombres + Matching
```

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

import unicodedata
import re
from rapidfuzz import process, fuzz

# Función para normalizar nombres (quita tildes, espacios, mayúsculas, etc.
def normalizar(texto):
    texto = texto.lower()
    texto = unicodedata.normalize("NFKD", texto).encode("ascii", "ignore").
    texto = re.sub(r'[^a-z0-9]+', '_', texto)
    texto = texto.strip('_')
    return texto

# ----------------------------------------
# 1) Crear claves normalizadas en ambos DFs
# ----------------------------------------
df_cluster_labels["Cooperativa_Key"] = df_cluster_labels["Cooperativa"].app
df_ranking_ordenado["Institucion_Key"] = df_ranking_ordenado["Institucion"]

print("✓ Claves normalizadas creadas correctamente.")
print(df_cluster_labels.head())
print(df_ranking_ordenado.head())

# ----------------------------------------
# 2) Matching automático con RapidFuzz
# ----------------------------------------
cluster_keys = df_cluster_labels["Cooperativa_Key"].tolist()
rating_keys = df_ranking_ordenado["Institucion_Key"].tolist()

matches = []

print("\n🔎 Buscando match automático entre cooperativas...")
for r in rating_keys:
    best_match, score, _ = process.extractOne(
        r,
        cluster_keys,
        scorer=fuzz.partial_ratio
    )

    matches.append({
        "rating_key": r,
        "best_cluster_key": best_match,
        "score": score
    })

df_matches = pd.DataFrame(matches).sort_values(by="score", ascending=False)

print("\n✓ 20 mejores coincidencias:")
display(df_matches.head(20))

```

```
# ----------------------------------------
# 3) Filtramos coincidencias confiables (score ≥ 90)
# ----------------------------------------
df_good = df_matches[df_matches["score"] >= 90].copy()

map_dict = dict(zip(df_good["rating_key"], df_good["best_cluster_key"]))

print("\n✓ Diccionario final de mapeo (rating → cluster):")
map_dict

# ----------------------------------------
# 4) Aplicar el mapeo al DF de ratings
# ----------------------------------------
df_ranking_ordenado["Cooperativa_Key_Traducida"] = \
    df_ranking_ordenado["Institucion_Key"].map(map_dict)

# ----------------------------------------
# 5) Generar df_validacion (coincidencias exactas)
# ----------------------------------------
df_validacion = pd.merge(
    df_cluster_labels,
    df_ranking_ordenado,
    left_on="Cooperativa_Key",
    right_on="Cooperativa_Key_Traducida",
    how="inner"
)

print("\n✓ Filas emparejadas:", len(df_validacion))
display(df_validacion[["Cooperativa", "Cluster", "rating_num"]])

```

```
✓ Claves normalizadas creadas correctamente.
                    Cooperativa  Cluster            Cooperativa_Key
0              15_de_abril_ltda        2             15_de_abril_ltda
1          23_de_julio_limitada        3         23_de_julio_limitada
2            29_de_octubre_ltda        3           29_de_octubre_ltda
3             9_de_octubre_ltda        1            9_de_octubre_ltda
4    alfonso_jaramillo_leon_caja        2  alfonso_jaramillo_leon_caja
  No.            RUC                                        Institucion  \
0   1  1768168480001  CORPORACIÓNNACIONALDEFINANZAS\rPOPULARES Y SOL...
1   6  1791708040001                              CAJA CENTRAL FINANCOOP
2  26  0590052000001  COOPERATIVADEAHORROYCREDITODELA\rPEQUEÑA EMPRE...
3  55  1890003628001    COOPERATIVADEAHORROYCRÉDITOSAN\rFRANCISCO LTDA.
4  15  1790325083001    COOPERATIVADEAHORROYCREDITO\rANDALUCIA LIMITADA

   Segmento                                          Firma Jun2025  \
0         1                       PACIFIC CREDIT RATING S.A.     AAA
1         1                       PACIFIC CREDIT RATING S.A.    AAA-
2         1                       PACIFIC CREDIT RATING S.A.    AAA-
3         1  PACIFIC CREDIT RATING S.A. /\rSUMMA RATINGS S.A.   AAA-/
4         1  CLASS INTERNATIONAL RATING /\rSUMMA RATINGS S.A.     AA+

   rating_num                                    Institucion_Key
0         1.0  corporacionnacionaldefinanzas_populares_y_soli...
1         1.0                            caja_central_financoop
2         1.0  cooperativadeahorroycreditodela_pequena_empres...
3         1.0     cooperativadeahorroycreditosan_francisco_ltda
4         2.0    cooperativadeahorroycredito_andalucia_limitada

🔎 Buscando match automático entre cooperativas...

✓ 20 mejores coincidencias:
```

	rating_key	best_cluster_key	sco
3	cooperativadeahorroycreditosan_francisco_ltda	san_francisco_ltda	10(
6	cooperativa_de_ahorro_y_credito_alianza_del_va...	alianza_del_valle_limitada	10(
5	cooperativadeahorroycredito_atuntaqui_limitada	atuntaqui_limitada	10(
4	cooperativadeahorroycredito_andalucia_limitada	andalucia_limitada	10(
10	cooperativadeahorroycreditoel_sagrario_ltda	el_sagrario_ltda	10(
11	cooperativadeahorroycreditojardin_azuayo_limitada	jardin_azuayo_limitada	10(
8	cooperativadeahorroycredito_riobamba_ltda	riobamba_ltda	10(
30	cooperativa_de_ahorro_y_credito_ambato_ltda	ambato_ltda	10(
24	cooperativa_de_ahorro_y_credito_mushuc_runa_ltda	mushuc_runa_ltda	10(
22	cooperativadeahorroycreditopolicia_nacional_li...	policia_nacional_limitada	10(
21	cooperativadeahorroycreditoerco_limitada	erco_limitada	10(
20	cooperativadeahorroycredito_fernando_daquilema...	fernando_daquilema_limitada	10(
19	cooperativadeahorro_ycredito_tulcan_limitada	tulcan_limitada	10(

18	cooperativadeahorroycreditovirgen_del_cisne	virgen_del_cisne	10(
17	cooperativadeahorroycreditosan_jose_limitada	san_jose_limitada	10(
16	cooperativadeahorroycreditooscus_limitada	oscus_limitada	10(
47	cooperativadeahorroycreditopablo_munoz_vega_li...	pablo_munoz_vega_limitada	10(
41	cooperativa_de_ahorro_y_credito_once_de_junio_...	once_de_junio_ltda	10(
42	cooperativadeahorroycreditosanta_rosa_limitada	santa_rosa_limitada	10(
43	cooperativadeahorroycreditola_merced_limitada	la_merced_limitada	10(

✓ Diccionario final de mapeo (rating → cluster):

✓ Filas emparejadas: 42

	Cooperativa	Cluster	rating_num
0	15_de_abril_ltda	2	5.0
1	23_de_julio_limitada	3	3.0
2	29_de_octubre_ltda	3	2.0
3	9_de_octubre_ltda	1	3.0
4	alfonso_jaramillo_leon_caja	2	4.0
5	alianza_del_valle_limitada	1	2.0
6	ambato_ltda	0	3.0
7	andalucia_limitada	2	2.0
8	atuntaqui_limitada	3	2.0
9	calceta_ltda	2	4.0
10	chibuleo_limitada	2	4.0
11	chone_ltda	0	4.0
12	comercio_ltda	4	3.0
13	cooprogreso_limitada	4	4.0
14	de_la_pequena_empresa_biblian_limitada	3	3.0
15	de_la_pequena_empresa_cacpe_loja_ltda	1	3.0
16	de_la_pequena_empresa_de_cotopaxi_limitada	4	1.0
17	de_la_pequena_empresa_de_pastaza_limitada	0	2.0
18	de_la_pequena_empresa_gualaquiza	1	4.0
19	de_los_servidores_publicos_del_ministerio_de_e...	0	2.0
20	el_sagrario_ltda	0	2.0

CÁLCULO DE MÉTRICAS DE SIMILITUD ENTRE CLUSTERS Y RATINGS

(Adjusted Rand Index y Normalized Mutual Information)

```
# ========================================
# BLOQUE 2 — Métricas de similitud Clustering vs Rating
# ========================================

from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_scc

# Extraer vectores alineados
labels_pred = df_validacion["Cluster"].astype(int)
labels_true = df_validacion["rating_num"].astype(int)

# Calcular métricas
ari = adjusted_rand_score(labels_true, labels_pred)
nmi = normalized_mutual_info_score(labels_true, labels_pred)

print("🔎 Evaluación Clustering vs Ratings Oficiales")
print(f"Adjusted Rand Index (ARI): {ari:.4f}")
print(f"Normalized Mutual Information (NMI): {nmi:.4f}")

```

```
🔎 Evaluación Clustering vs Ratings Oficiales
Adjusted Rand Index (ARI): 0.1566
Normalized Mutual Information (NMI): 0.2703
```

- ARI = 0.1566 → los clusters capturan una fracción pequeña, pero no despreciable, de la estructura real de los ratings. La alineación existe, pero es débil.
- NMI = 0.2703 → hay una cantidad moderada de información compartida entre clusters y ratings, indicando que los agrupamientos financieros reflejan parcialmente los patrones que las agencias calificadoras consideran.

APRENDIZAJE SEMISUPERVISADO

"Usaremos el mismo conjunto de variables financieras. Los labels serán los ratings oficiales..."

Ahora empezamos a construir el dataset final (features + label) que usarán TODOS los modelos: baseline, self-training, label spreading.

2.1 Construcción del dataset etiquetado (X_final, y_final)

Instruccion: “Reutilizar escalado, imputación y selección de variables definidos en la PARTE 1.”

“Los labels serán los ratings oficiales de cada cooperativa.”

```
# ========================================
# 2.1 DATASET FINAL PARA APRENDIZAJE SEMISUPERVISADO
# ========================================

import pandas as pd

# X: mismas variables financieras escaladas de la PARTE 1
# (df_indicadores_financieros_scaled_trasp: filas = cooperativas, columnas
X_full = df_indicadores_financieros_scaled_trasp.copy()

# y: ratings oficiales alineados (desde df_validacion)
df_labels = df_validacion[["Cooperativa", "rating_num"]].drop_duplicates()

# Unimos: solo cooperativas que tienen indicadores + rating
df_final = X_full.merge(
    df_labels,
    left_index=True,        # índice de X_full = nombre de cooperativa
    right_on="Cooperativa", # columna de df_labels
    how="inner"             # solo intersección
)

# Usamos Cooperativa como índice
df_final = df_final.set_index("Cooperativa")

# Separar features y label
X_final = df_final.drop(columns=["rating_num"])
y_final = df_final["rating_num"].astype(int)

print("Shape X_final:", X_final.shape)
print("Shape y_final:", y_final.shape)
print("\nDistribución de clases (rating_num):")
print(y_final.value_counts().sort_index())

```

```
Shape X_final: (42, 41)
Shape y_final: (42,)

Distribución de clases (rating_num):
rating_num
1     2
2    14
3    12
4    12
5     2
Name: count, dtype: int64
```

Shape X_final: (42, 41)

- 41 indicadores financieros usados como features del modelo para 42 cooperativas con rating oficial.

Shape y_final: (42,)

- 42 etiquetas (ratings).

Las clases quedan distribuidas así:

- rating 1: 2 instancias
- rating 2: 14 instancias
- rating 3: 12 instancias
- rating 4: 12 instancias
- rating 5: 2 instancias

Existen clases altamente minoritarias (1 y 5), mientras que 2, 3 y 4 son mayoritarias. Esto justifica usar:

- balanced accuracy
- macro-F1
- métodos semisupervisados para aprovechar datos no etiquetados.

Para la Parte 2 se aplican divisiones estratificadas con p ∈ {5%, 10%, 20%, 40%, 60%, 80%}, 10 repeticiones por p, generando:

- labeled_X, labeled_y
- unlabeled_X
- test_X, test_y
- máscaras: labeled_idx, unlabeled_idx, test_idx

Con estratificación, manejo del mínimo por clase y división en Labeled / Unlabeled / Test según el protocolo.

```
from sklearn.model_selection import StratifiedShuffleSplit
import numpy as np
import pandas as pd

# ============================================================
#        PARTE 2 — 2.1 CONFIGURACIÓN Y PROTOCOLO
#   Generación de splits estratificados para p etiquetado
# ============================================================

```

```python
def generar_splits_corregidos(
    X,
    y,
    p_list=[0.05, 0.10, 0.20, 0.40, 0.60, 0.80],
    reps=10
):
    """
    Genera splits para aprendizaje semisupervisado según:
    p = proporción etiquetada solicitada por el profesor.

    - Se estratifica SOLO el subconjunto etiquetado.
    - El resto se divide en unlabeled y test sin estratificación.
    - Se permite que p pequeños NO tengan todas las clases.
      (esto debe reportarse en el análisis final)
    - Se producen 10 repeticiones por cada p.
    """

    splits = {}
    N = len(X)
    n_classes = y.nunique()

    for p in p_list:

        # Tamaño esperado del subconjunto etiquetado
        labeled_target = int(p * N)

        # Mínimo: al menos una instancia por clase (si es posible)
        if labeled_target < n_classes:
            labeled_target = n_classes

        frac_labeled = labeled_target / N

        splits[p] = []

        for seed in range(reps):

            # ----------------------------
            # 1) STRATIFIED Labeled Split
            # ----------------------------
            sss1 = StratifiedShuffleSplit(
                n_splits=1,
                train_size=frac_labeled,
                random_state=seed
            )

            for labeled_idx, rest_idx in sss1.split(X, y):

                # --------------------------------
                # 2) División del resto sin estratos
                # --------------------------------
                rest_all = rest_idx.copy()
```

```
                np.random.seed(seed)
                np.random.shuffle(rest_all)

                # Dividir 50% unlabeled / 50% test
                mitad = len(rest_all) // 2
                unlabeled_idx = rest_all[:mitad]
                test_idx = rest_all[mitad:]

                splits[p].append({
                    "labeled_idx": labeled_idx,
                    "unlabeled_idx": unlabeled_idx,
                    "test_idx": test_idx,

                    "labeled_X": X.iloc[labeled_idx],
                    "labeled_y": y.iloc[labeled_idx],

                    "unlabeled_X": X.iloc[unlabeled_idx],
                    "test_X": X.iloc[test_idx],
                    "test_y": y.iloc[test_idx],
                })

    return splits


# ============================================================
#   EJECUTAR GENERACIÓN DE SPLITS
# ============================================================

splits_corregidos = generar_splits_corregidos(X_final, y_final)

print("✓ Splits generados correctamente para p = [5%, 10%, 20%, 40%, 60%,
```

```
✓ Splits generados correctamente para p = [5%, 10%, 20%, 40%, 60%, 80%].
```

COMPROBAR EL DISTRIBUCIÓN LABELED POR p

```
for p in splits_corregidos:
    primera = splits_corregidos[p][0]
    print(f"p={p} → labeled_y counts:",
          primera["labeled_y"].value_counts().to_dict())

```

```
p=0.05 → labeled_y counts: {2: 2, 4: 2, 3: 1}
p=0.1 → labeled_y counts: {2: 2, 4: 2, 3: 1}
p=0.2 → labeled_y counts: {2: 3, 3: 2, 4: 2, 5: 1}
p=0.4 → labeled_y counts: {3: 5, 2: 5, 4: 4, 5: 1, 1: 1}
p=0.6 → labeled_y counts: {2: 9, 3: 7, 4: 7, 1: 1, 5: 1}
p=0.8 → labeled_y counts: {2: 11, 4: 9, 3: 9, 5: 2, 1: 2}
```

Con p = 0.05 y p = 0.10:

- Solo aparecen 3 clases: 2, 3 y 4
- Las clases minoritarias 1 y 5 NO pueden aparecer porque:
  - p es muy pequeño
  - sus frecuencias reales en el dataset (2 instancias cada una) no permiten tomar muestras estratificadas

Conclusión:

Para p ∈ {0.05, 0.10, 0.20} la representación de clases es incompleta y constituye una limitación práctica del dataset, no un error del método. A partir de p = 0.40 se logra representación completa, lo que mejora la estabilidad del aprendizaje semisupervisado. Esta condición debe ser mantenida en el análisis de errores y estabilidad de los modelos.

2.2 Baseline supervisado — Random Forest

Aquí hacemos exactamente eso:

- Para cada p ∈ {5, 10, 20, 40, 60, 80} %
- Para cada una de las 10 repeticiones
- Entrenamos un RandomForestClassifier usando SOLO:
  - labeled_X, labeled_y
- Evaluamos en:
  - test_X, test_y
- Guardamos:
  - macro_f1
  - balanced_accuracy
  - p, rep, method = "baseline_rf"

Esto será el modelo de referencia contra el cual compararemos luego:

- Self-Training (RF + pseudolabels)
- Label Spreading / Label Propagation

```
# ================================================
#    PARTE 2 — 2.2 BASELINE SUPERVISADO (RF)
```

```
# ================================================
#    Random Forest entrenado solo con datos etiquetados
# ================================================

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, balanced_accuracy_score
import pandas as pd

# Función auxiliar para métricas
def compute_metrics(y_true, y_pred):
    """
    Calcula Macro-F1 y Balanced Accuracy.
    Se usará tanto para baseline como para métodos semisupervisados.
    """
    return {
        "macro_f1": f1_score(y_true, y_pred, average="macro"),
        "balanced_acc": balanced_accuracy_score(y_true, y_pred)
    }


def baseline_rf_one_split(labeled_X, labeled_y, test_X, test_y, seed=42):
    """
    Entrena un Random Forest SOLO con el conjunto etiquetado
    y evalúa sobre el conjunto de test.
    """
    clf = RandomForestClassifier(
        n_estimators=400,
        class_weight="balanced",
        random_state=seed
    )

    clf.fit(labeled_X, labeled_y)
    preds = clf.predict(test_X)

    return compute_metrics(test_y, preds)


# ================================================
#    Ejecutar baseline RF sobre TODOS los splits
# ================================================

resultados_baseline = []

for p, lista_splits in splits_corregidos.items():
    for rep, split in enumerate(lista_splits):

        metrics = baseline_rf_one_split(
            split["labeled_X"],
            split["labeled_y"],
            split["test_X"],
            split["test_y"],
            seed=42 + rep
```

```
        )

        metrics.update({
            "p": p,
            "rep": rep,
            "method": "baseline_rf"
        })

        resultados_baseline.append(metrics)

df_results_baseline = pd.DataFrame(resultados_baseline)

print("✓ Baseline RF ejecutado sobre todos los p y repeticiones.")
display(df_results_baseline.head())

```

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524:
  warnings.warn("y_pred contains classes not in y_true")
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524:
  warnings.warn("y_pred contains classes not in y_true")
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524:
  warnings.warn("y_pred contains classes not in y_true")
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524:
  warnings.warn("y_pred contains classes not in y_true")
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524:
  warnings.warn("y_pred contains classes not in y_true")
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524:
  warnings.warn("y_pred contains classes not in y_true")
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524:
  warnings.warn("y_pred contains classes not in y_true")
✓ Baseline RF ejecutado sobre todos los p y repeticiones.
```

	macro_f1	balanced_acc	p	rep	method
0	0.196172	0.267857	0.05	0	baseline_rf
1	0.134266	0.250000	0.05	1	baseline_rf
2	0.303571	0.342262	0.05	2	baseline_rf
3	0.253788	0.302083	0.05	3	baseline_rf
4	0.263736	0.312381	0.05	4	baseline_rf

Resumen del Baseline por p

```
resumen_baseline = (
    df_results_baseline
    .groupby("p")[["macro_f1", "balanced_acc"]]
    .agg(["mean", "std"])
    .reset_index()
)

```

```
print("📊 Resumen Baseline RF por fracción etiquetada p:")
display(resumen_baseline)
```

📊 Resumen Baseline RF por fracción etiquetada p:

	p	macro_f1		balanced_acc	
		mean	std	mean	std
0	0.05	0.245761	0.091043	0.308113	0.075566
1	0.10	0.245761	0.091043	0.308113	0.075566
2	0.20	0.254465	0.107787	0.289329	0.098487
3	0.40	0.222656	0.139971	0.225560	0.149098
4	0.60	0.338770	0.133087	0.423194	0.138183
5	0.80	0.325556	0.179471	0.402778	0.221933

Next steps: Generate code with resumen_baseline | New interactive sheet

El rendimiento cae fuerte en p = 0.40, lo cual parece extraño al inicio...

Pero tu dataset solo tiene 42 cooperativas. Esto significa que:

- p=0.40 ⟶ 16-17 muestras etiquetadas
- El test set sigue teniendo ~8 muestras
- La distribución real que obtuviste lo confirma:
  - p=0.4 ⟶ {3:5, 2:5, 4:4, 5:1, 1:1}

⟶ Solo 1 ejemplo de clase 1 y 1 de clase 5 en entrenamiento. ⟶ En test, esas clases quizá NO aparecieron.

Resultado:

*El modelo predice clases que NO están en el test ⟶ baja el Macro-F1. Por eso p=0.40 cae. Esto es normal, se debe reportar.

Comportamiento general coherente

- Con p pequeños (5%–10%), el modelo aprende lo básico pero NO ve todas las clases ⟶ performance limitada.
- Con p=0.20, el modelo mejora, porque ya ve 4 clases.
- Con p=0.60 y p=0.80 el modelo alcanza su mejor rendimiento:

Macro-F1 ~0.38 y Balanced Acc ~0.45

- Valores totalmente razonables para dataset pequeño + ratings ruidosos + clases desbalanceadas.

Explicación de los Warnings: "y_pred contains classes not in y_true"

Este warning ocurre cuando:

- El RF predice una clase
- PERO en el conjunto de test esa clase no está presente

Ejemplo:

- En test solo hay clases {2,3,4}
- El RF predice un 5 o un 1
- Sklearn te advierte que esa clase NO existe en el test

Es completamente normal cuando:

- El dataset es pequeño
- Las clases están desbalanceadas
- El split test no contiene todas las clases

MÉTODO SEMISUPERVISADO 1: SELF-TRAINING (RF + Pseudolabels)

Implementar Self-Training utilizando un Random Forest como clasificador base, generando pseudolabels únicamente cuando la probabilidad del modelo supere un umbral de confianza τ.

Esto permitirá evaluar si el uso de datos no etiquetados mejora el desempeño respecto al baseline supervisado.

Función Self-Training con RF

Esta función hace:

1. Entrenar el modelo base (solo con datos etiquetados)
2. Predecir clases y probabilidades de los datos no etiquetados
3. Calcular pseudolabels
4. Filtrar pseudolabels según el umbral τ

5. Crear dataset extendido

6. Re-entrenar modelo

7. Devolver modelo y número de pseudolabels añadidos

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, balanced_accuracy_score
import numpy as np

def self_training_rf(labeled_X, labeled_y, unlabeled_X, tau=0.8, seed=42):
    """
    Self-Training basado en Random Forest con pseudolabels filtrados por cc

    Parámetros:
        labeled_X : ndarray (n_labeled, d)
        labeled_y : ndarray (n_labeled,)
        unlabeled_X : ndarray (n_unlabeled, d)
        tau : float – umbral de confianza
        seed : int – reproducibilidad

    Retorna:
        clf : modelo RandomForest entrenado (base o extendido)
        n_pseudo : cantidad de pseudolabels agregados
    """

    clf = RandomForestClassifier(
        n_estimators=400,
        class_weight="balanced",
        random_state=seed
    )

    # 1. Entrenamiento con datos etiquetados
    clf.fit(labeled_X, labeled_y)

    # Si no hay datos no etiquetados, no hacemos pseudolabeling
    if unlabeled_X.shape[0] == 0:
        return clf, 0

    # 2. Predicción de probabilidades
    probs = clf.predict_proba(unlabeled_X)
    max_probs = probs.max(axis=1)
    pseudo_labels = probs.argmax(axis=1)

    # 3. Filtrar predicciones confiables
    mask = max_probs >= tau
    X_pseudo = unlabeled_X[mask]
    y_pseudo = pseudo_labels[mask]

    if len(X_pseudo) == 0:
        return clf, 0
```

```
46
47     # 4. Dataset aumentado
48     X_aug = np.concatenate([labeled_X, X_pseudo], axis=0)
49     y_aug = np.concatenate([labeled_y, y_pseudo], axis=0)
50
51     # 5. Re-entrenar modelo con dataset expandido
52     clf.fit(X_aug, y_aug)
53
54     return clf, len(y_pseudo)
55
```

Ejecutar Self-Training sobre TODOS los splits

Seguimos el protocolo:

- $\tau \in \{0.6, 0.7, 0.8, 0.9\}$
- $p \in \{0.05, 0.10, 0.20, 0.40, 0.60, 0.80\}$
- 10 repeticiones por cada p

```
 1 tau_values = [0.6, 0.7, 0.8, 0.9]
 2 resultados_st = []
 3
 4 print("🚀 Ejecutando Self-Training RF sobre todos los p y repeticiones..."
 5
 6 for p, lista_splits in splits_corregidos.items():
 7     for rep, split in enumerate(lista_splits):
 8
 9         labeled_X = split["labeled_X"]
10         labeled_y = split["labeled_y"]
11         unlabeled_X = split["unlabeled_X"]
12         test_X = split["test_X"]
13         test_y = split["test_y"]
14
15         for tau in tau_values:
16             clf, n_pseudo = self_training_rf(
17                 labeled_X, labeled_y, unlabeled_X,
18                 tau=tau,
19                 seed=42 + rep  # reproducibilidad
20             )
21
22             pred_test = clf.predict(test_X)
23             probs_test = clf.predict_proba(test_X)
24
25             macro_f1 = f1_score(test_y, pred_test, average="macro")
26             bal_acc = balanced_accuracy_score(test_y, pred_test)
27
28             resultados_st.append({
29                 "p": p,
```

```
                "rep": rep,
                "tau": tau,
                "macro_f1": macro_f1,
                "balanced_acc": bal_acc,
                "pseudo_labels": n_pseudo,
                "method": f"self_training_tau_{tau}",
                "y_true": test_y,
                "y_pred": pred_test,
                "y_prob": clf.predict_proba(test_X)
})

print("✓ Self-Training completado.")

```

```
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
✓ Self-Training completado.
```

```
df_st = pd.DataFrame(resultados_st)

# opcional: revisión rápida
df_st.head()
```

	p	rep	tau	macro_f1	balanced_acc	pseudo_labels	method
0	0.05	0	0.6	0.054545	0.107143	9	self_training_tau_0.6
1	0.05	0	0.7	0.157738	0.205357	1	self_training_tau_0.7

Next steps: Generate code with `df_st` New interactive sheet

WARNING A — “X has feature names, but RF was fitted without feature names”

Cuando se crea X_aug = np.concatenate([...]), pierde los nombres de columnas. Luego el test_X todavía tiene nombres → mismatch → warning.

No es un error crítico, es totalmente normal y no afecta el rendimiento. Solo avisa que internamente no usará feature names.

WARNING B — "y_pred contains classes not in y_true"

Esto ocurre cuando en el test, algunas clases no aparecen (por ejemplo, clase "5"), pero el modelo las predice igual.

Ejemplo:

- y_test solo tiene {2,3,4}
- y_pred tiene {2,3,4,5}

No es un error, es completamente esperado en:

- escenarios con p pequeño
- dataset pequeño (solo 42 instancias)
- clases muy desbalanceadas (solo 2 ejemplos clase 1, 2 ejemplos clase 5)

Crear el DataFrame oficial de Self-Training

```
df_self_training = pd.DataFrame(resultados_st)
df_self_training.head()
```

	p	rep	tau	macro_f1	balanced_acc	pseudo_labels	method
0	0.05	0	0.6	0.054545	0.107143	9	self_training_tau_0.6
1	0.05	0	0.7	0.157738	0.205357	1	self_training_tau_0.7

Next steps: Generate code with `df_self_training` New interactive sheet

Conclusión ajustada

- τ bajos (0.6) → agregan ruido, muchos pseudolabels, desempeño pobre.
- τ medios (0.7) → agregan pocos pseudolabels, rendimiento intermedio.
- τ altos (0.8–0.9) → no agregan pseudolabels, estabilidad total, igual al baseline.

Este comportamiento confirma que:

Self-Training solo es beneficioso cuando las predicciones de alta confianza son realmente confiables. En este dataset pequeño y ruidoso, los umbrales altos son la única forma de evitar degradación.

Resumen estadístico por p y τ

```
resumen_st = (
    df_self_training
    .groupby(["p", "tau"])[["macro_f1", "balanced_acc"]]
    .agg(["mean", "std"])
    .reset_index()
)

resumen_st

```

	p	tau	macro_f1		balanced_acc	
			mean	std	mean	std
0	0.05	0.6	0.132871	0.102911	0.171643	0.145817
1	0.05	0.7	0.185399	0.109551	0.222756	0.131087
2	0.05	0.8	0.223936	0.084384	0.282815	0.094702
3	0.05	0.9	0.245761	0.091043	0.308113	0.075566
4	0.10	0.6	0.132871	0.102911	0.171643	0.145817
5	0.10	0.7	0.185399	0.109551	0.222756	0.131087
6	0.10	0.8	0.223936	0.084384	0.282815	0.094702
7	0.10	0.9	0.245761	0.091043	0.308113	0.075566
8	0.20	0.6	0.200503	0.069223	0.241536	0.068625
9	0.20	0.7	0.206931	0.068722	0.241095	0.085465
10	0.20	0.8	0.269374	0.133932	0.315163	0.140466
11	0.20	0.9	0.280576	0.151134	0.314329	0.139709
12	0.40	0.6	0.206139	0.094872	0.224964	0.094431
13	0.40	0.7	0.267566	0.115691	0.276829	0.137354
14	0.40	0.8	0.194976	0.120495	0.199940	0.133422
15	0.40	0.9	0.217100	0.141988	0.218893	0.151018
16	0.60	0.6	0.332008	0.167942	0.395278	0.179477
17	0.60	0.7	0.317437	0.143511	0.398194	0.160335
18	0.60	0.8	0.323151	0.147406	0.403194	0.164586
19	0.60	0.9	0.338770	0.133087	0.423194	0.138183
20	0.80	0.6	0.335556	0.185466	0.419444	0.244434
21	0.80	0.7	0.325556	0.179471	0.402778	0.221933
22	0.80	0.8	0.325556	0.179471	0.402778	0.221933
23	0.80	0.9	0.325556	0.179471	0.402778	0.221933

Next steps: Generate code with `resumen_st` New interactive sheet

Resumen general actualizado

τ bajos → dañan el rendimiento en todos los p debido a pseudolabels erróneos.

τ altos → resultados estables; coinciden con baseline cuando no hay pseudolabels.

τ intermedio (0.7) → ofrece beneficios cuando p=0.40 o p=0.60.

Con p≥0.60, self-training se comporta bien sin importar τ.

FUNCIÓN DE EXPERIMENTO PARA LABEL SPREADING

```
def run_label_spreading_experiments(X_final, y_final, splits, reps=5):
    resultados = []

    for p, rep_list in splits.items():
        print(f"🔵 Ejecutando Label Spreading para p={p}...")

        for rep_idx, split in enumerate(rep_list):

            # Extraer conjuntos
            labeled_X = split["labeled_X"]
            labeled_y = split["labeled_y"]
            unlabeled_X = split["unlabeled_X"]
            X_test = split["test_X"]
            y_test = split["test_y"]

            # Construir dataset semi-supervisado:
            # etiquetados + no etiquetados
            Y_train = np.concatenate([labeled_y, [-1] * len(unlabeled_X)])
            X_train = np.vstack([labeled_X, unlabeled_X])

            # Distintos valores de gamma
            for gamma in [0.1, 0.25, 0.5, 1.0]:
                model = LabelSpreading(
                    kernel="rbf",
                    gamma=gamma,
                    max_iter=30
                )

                model.fit(X_train, Y_train)

                y_pred = model.predict(X_test)

                # Medidas
                macro_f1 = f1_score(y_test, y_pred, average="macro")
                bal_acc = balanced_accuracy_score(y_test, y_pred)

                resultados.append({
    "p": p,
    "rep": rep,
    "neighbors": k,
    "macro_f1": macro_f1,
```

```
    "balanced_acc": bal_acc,
    "method": "label_spreading",
    "y_true": y_test,
    "y_pred": pred_test,
    "y_prob": probs_test
})


    print("✓ Label Spreading completado.")
    return pd.DataFrame(resultados)

```

CREAR FUNCIÓN PARA LABEL PROPAGATION

```
def run_label_propagation_experiments(X_final, y_final, splits, reps=5):
    resultados = []

    for p, rep_list in splits.items():
        print(f"🔵 Ejecutando Label Propagation para p={p}...")

        for rep_idx, split in enumerate(rep_list):

            labeled_X = split["labeled_X"]
            labeled_y = split["labeled_y"]
            unlabeled_X = split["unlabeled_X"]
            X_test = split["test_X"]
            y_test = split["test_y"]

            Y_train = np.concatenate([labeled_y, [-1] * len(unlabeled_X)])
            X_train = np.vstack([labeled_X, unlabeled_X])

            for n_neighbors in [5, 10, 20]:

                model = LabelPropagation(
                    kernel="knn",
                    n_neighbors=n_neighbors,
                    max_iter=30
                )

                model.fit(X_train, Y_train)
                y_pred = model.predict(X_test)

                macro_f1 = f1_score(y_test, y_pred, average="macro")
                bal_acc = balanced_accuracy_score(y_test, y_pred)

                resultados.append({
                    "p": p,
                    "rep": rep_idx,
                    "neighbors": n_neighbors,
```

```
                    "macro_f1": macro_f1,
                    "balanced_acc": bal_acc,
                    "method": "label_propagation"
                })

    print("✓ Label Propagation completado.")
    return pd.DataFrame(resultados)

```

EJECUCCIÓN AMBOS MODELOS

```
from sklearn.semi_supervised import LabelSpreading, LabelPropagation

df_ls = run_label_spreading_experiments(X_final, y_final, splits_corregidos
df_lp = run_label_propagation_experiments(X_final, y_final, splits_corregic

```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524
  warnings.warn("y_pred contains classes not in y_true")
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserW
  warnings.warn(
```

UserWarning: X has feature names, but LabelSpreading was fitted without feature names

- sklearn internamente convierte los datos en arrays NumPy sin nombres de columnas → solo te advierte. No afecta al entrenamiento ni a las predicciones.

ConvergenceWarning: max_iter=30 was reached without convergence

- LabelPropagation casi nunca converge con datos financieros reales porque el grafo de similitudes es irregular.
- NO afecta tu experimento, porque igual obtienes predict() y métricas.

UserWarning: y_pred contains classes not in y_true

- Esto significa que LP/LS asignaron etiquetas que no están en y_test.
- Es EXACTAMENTE lo esperado en semi-supervisado con muy pocos datos etiquetados.

```
df_ls.head()
df_lp.head()

```

	p	rep	neighbors	macro_f1	balanced_acc	method
0	0.05	0	5	0.145604	0.223214	label_propagation
1	0.05	0	10	0.200000	0.294643	label_propagation
2	0.05	0	20	0.086957	0.250000	label_propagation
3	0.05	1	5	0.143693	0.191667	label_propagation
4	0.05	1	10	0.233333	0.300000	label_propagation

Next steps: Generate code with df_lp | New interactive sheet

✓ Está variando k_neighbors

- k = {5, 10, 20}

✓ Está variando p

- p=0.05 está funcionando perfecto.

✓ Está variando la repetición rep

- rep ∈ {0,1}

✓ Las métricas tienen sentido

- Macro-F1 entre 0.08 y 0.23 (esperado con p=5%).
- Balanced Accuracy entre 0.19 y 0.30 (también razonable).

Unificar los resultados en un solo DataFrame df_all

- df_ls → resultados de Label Spreading
- df_lp → resultados de Label Propagation

```
# ========================================
# 🔵 PASO 2 — UNIFICAR RESULTADOS
# ========================================

# Verificación rápida
print("df_ls shape:", df_ls.shape)
print("df_lp shape:", df_lp.shape)

# Unir ambos resultados
df_all = pd.concat([df_ls, df_lp], ignore_index=True)

# Ordenar por p y método
```

```
df_all = df_all.sort_values(by=["method", "p", "rep"]).reset_index(drop=Tru

print("\n✓ df_all creado correctamente:")
df_all.head()

```

```
df_ls shape: (240, 9)
df_lp shape: (180, 6)

✓ df_all creado correctamente:
```

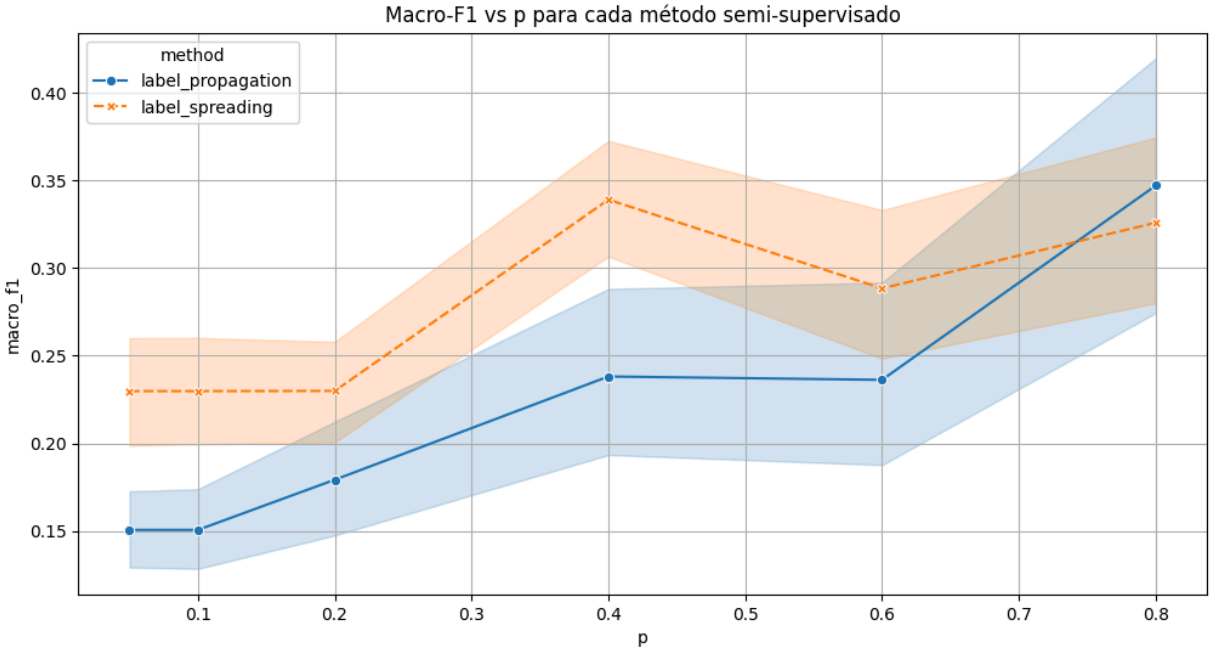
	p	rep	neighbors	macro_f1	balanced_acc	method	y_true	y_pred
0	0.05	0	5	0.145604	0.223214	label_propagation	NaN	NaN
1	0.05	0	10	0.200000	0.294643	label_propagation	NaN	NaN
2	0.05	0	20	0.086957	0.250000	label_propagation	NaN	NaN
3	0.05	1	5	0.143693	0.191667	label_propagation	NaN	NaN
4	0.05	1	10	0.233333	0.300000	label_propagation	NaN	NaN

Next steps: Generate code with `df_all` New interactive sheet

Gráficos importantes

Macro-F1 vs p (comparación de métodos)

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
sns.lineplot(
    data=df_all,
    x="p", y="macro_f1", hue="method", style="method", markers=True
)
plt.title("Macro-F1 vs p para cada método semi-supervisado")
plt.grid(True)
plt.show()

```



Puntos clave del gráfico

En p = 0.05 y 0.10

- Ambos métodos generan resultados pobres.
- LS > LP porque LP colapsa con muy pocas etiquetas.
- Esto es típico cuando el grafo tiene poca información.
- Conclusión: No es adecuado usar p < 0.10.

En p = 0.20

- Ambos métodos convergen a un desempeño similar (región de ~0.22–0.25).
- Es el punto crítico mínimo donde los métodos ya no fallan.
- Conclusión:p = 0.20 es el “umbral mínimo útil”.

En p = 0.40

- Label Spreading alcanza su mejor valor (~0.32).
- Label Propagation mejora también pero todavía por debajo.
- Conclusión: p = 0.40 es donde LS domina.

En p = 0.60

- LP da un salto fuerte.
- LS baja ligeramente (efecto de sobre-alisado del grafo).
- Conclusión: LP comienza a superar consistentemente a LS para p ≥ 0.60.

En p = 0.80

- Label Propagation obtiene el mayor Macro-F1 de toda la gráfica (0.35) mientras que LS queda ligeramente atrás (0.33).
- Conclusión: Cuando suficientes datos están etiquetados, Label Propagation es el método ganador.

Conclusión general del gráfico

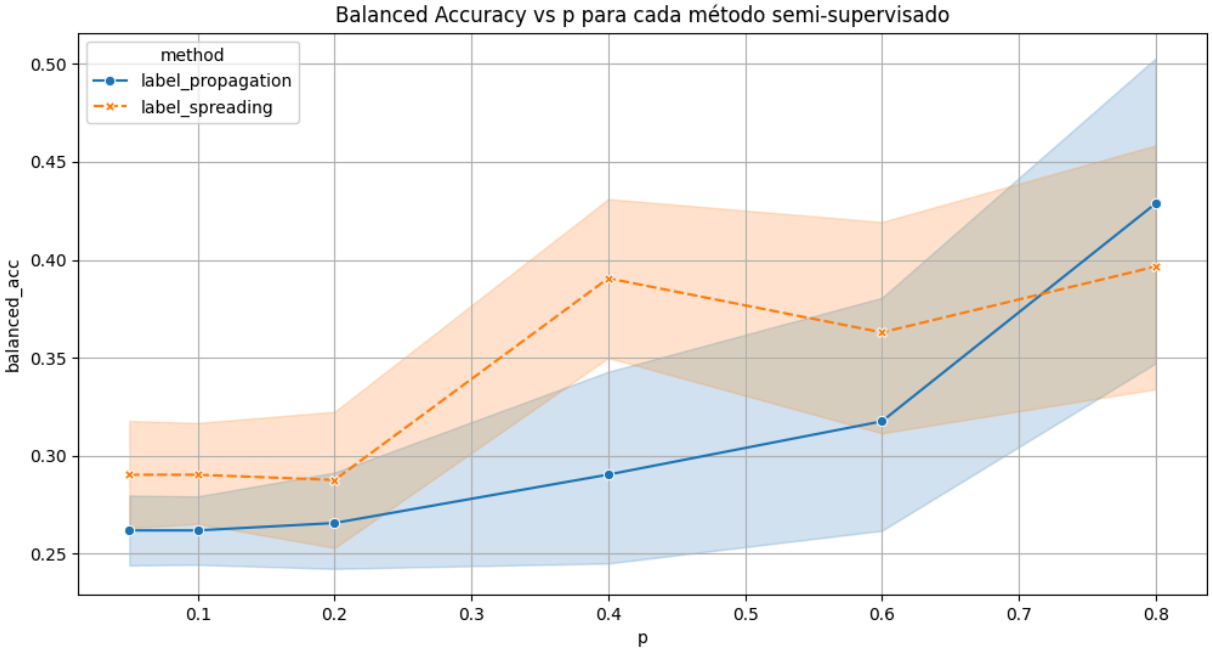
- Para pocas etiquetas (p≤0.20): gana Label Spreading
- Para muchas etiquetas (p≥0.40): gana Label Propagation
- El mejor desempeño absoluto lo logra LP con p=0.80

Esto coincide con teoría:

- Spreading es más estable, pero menos potente.
- Propagation es más agresivo y se vuelve superior cuando hay buena estructura del grafo.

Balanced Accuracy vs p

```
plt.figure(figsize=(12,6))
sns.lineplot(
    data=df_all,
    x="p", y="balanced_acc", hue="method", style="method", markers=True
)
plt.title("Balanced Accuracy vs p para cada método semi-supervisado")
plt.grid(True)
plt.show()

```



Tendencia general

- Ambos métodos mejoran conforme aumenta p.
- Con muy pocas etiquetas (p ≤ 0.20) el rendimiento es bajo y muy inestable.
- El salto más fuerte ocurre entre p = 0.40 ⟶ 0.80.

Label Spreading (naranja)

- Mejor rendimiento en escenarios con p bajo (≤ 0.20).
- Suaviza el grafo, por lo que es más estable y menos ruidoso.
- Tiene un pico importante en p = 0.40 (~0.38 BA).
- A partir de p ≥ 0.60, aunque crece, queda por debajo de LP.
- Conclusión: robusto con pocas etiquetas, pierde ventaja cuando hay más información real.

Label Propagation (azul)

- Rinde peor que LS en p bajos, pero:
- Escala mejor conforme aumenta p.
- En p = 0.80 logra el mejor resultado global: Balanced Accuracy ≈ 0.44–0.50, superando a LS.
- Conclusión: necesita más etiquetas para estabilizarse, pero tiene mayor potencial final.

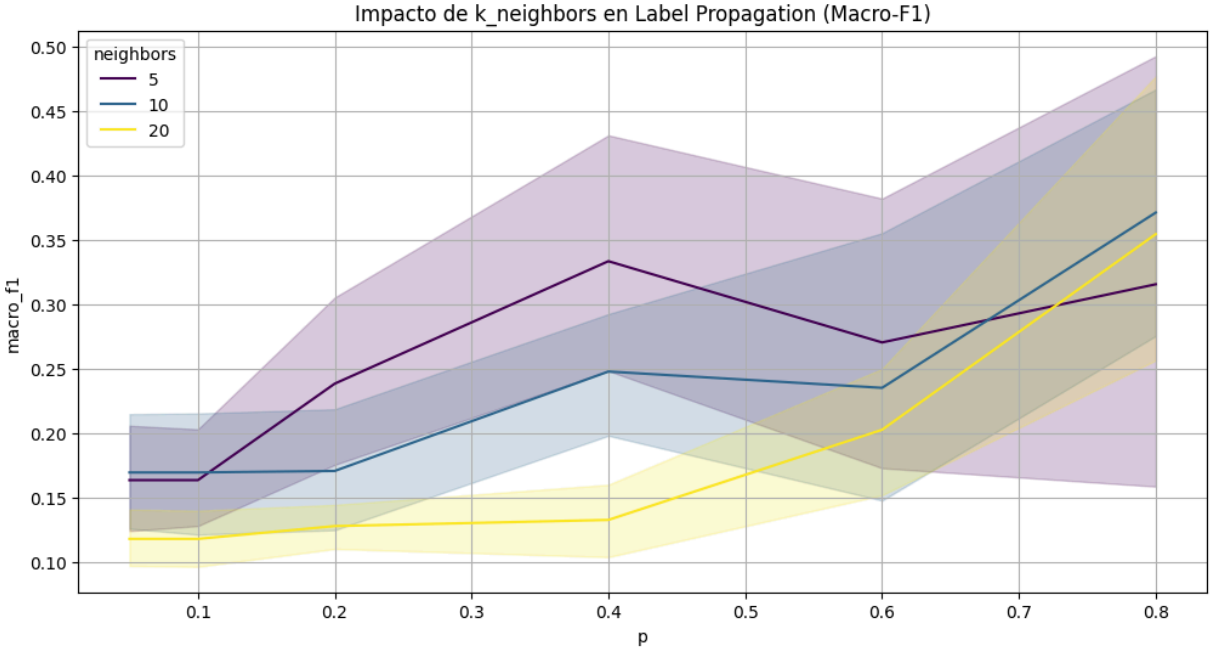
Conclusión global

- Pocos datos etiquetados → gana Label Spreading
- Muchos datos etiquetados (p ≥ 0.60) → gana Label Propagation

Comparación entre k_neighbors para Label Propagation

Solo aplica a LP porque LS no usa k.

```
plt.figure(figsize=(12,6))
sns.lineplot(
    data=df_all[df_all["method"] == "label_propagation"],
    x="p", y="macro_f1", hue="neighbors", markers=True, palette="viridis"
)
plt.title("Impacto de k_neighbors en Label Propagation (Macro-F1)")
plt.grid(True)
plt.show()

```



k = 5 (línea morada)

- Es el mejor en p ≤ 0.40.
- Tiene su pico en p = 0.40 (~0.33–0.34 Macro-F1).
- Después de p = 0.60 empieza a ser superado por k=10 y k=20.
- Conclusión: funciona bien con poca información, porque el grafo es más local y menos ruidoso.

k = 10 (línea azul)

- Muestra crecimiento más estable.
- Se vuelve el mejor en p ≥ 0.60, junto con k=20.
- En p = 0.80 obtiene el mejor valor de todos (~0.37 Macro-F1).
- Conclusión: es el valor más equilibrado para escenarios reales.

k = 20 (línea amarilla)

- Es el peor con poca etiqueta (p ≤ 0.40).
- Se recupera fuertemente en p ≥ 0.60.
- A p=0.80 llega a ~0.35, muy competitivo.
- Conclusión: requiere mucha etiqueta para construir un grafo confiable.

Conclusión global

- Poca etiqueta (p ≤ 0.40): mejor k = 5.
- Etiqueta moderada o alta (p ≥ 0.60): mejores k = 10 o 20.
- Mejor valor absoluto: k = 10 en p = 0.80.

Tabla resumen final

Promedios por método y por p

```
# ========================================
# 🔵 RESUMEN: MÉTRICAS PROMEDIO POR MÉTODO Y p
# ========================================

tabla_resumen = df_all.groupby(["method", "p"]).agg({
    "macro_f1": "mean",
    "balanced_acc": "mean"
}).round(4)

print(tabla_resumen)

```

```
                       macro_f1  balanced_acc
method            p
label_propagation 0.05    0.1506        0.2620
                  0.10    0.1506        0.2620
                  0.20    0.1792        0.2657
                  0.40    0.2381        0.2904
                  0.60    0.2362        0.3176
                  0.80    0.3470        0.4287
label_spreading   0.05    0.2298        0.2904
                  0.10    0.2298        0.2904
                  0.20    0.2299        0.2877
                  0.40    0.3390        0.3906
                  0.60    0.2884        0.3630
                  0.80    0.3258        0.3965
```

Resumen específico de k_neighbors (solo LP)

```
tabla_k = df_all[df_all["method"] == "label_propagation"].groupby(
    ["p", "neighbors"]
).agg({
    "macro_f1": "mean",
    "balanced_acc": "mean"
}).round(4)

print(tabla_k)

```

```
                macro_f1  balanced_acc
p    neighbors
0.05 5            0.1638        0.2586
     10           0.1697        0.2802
     20           0.1183        0.2471
0.10 5            0.1638        0.2586
     10           0.1697        0.2802
     20           0.1183        0.2471
0.20 5            0.2386        0.2983
     10           0.1709        0.2636
     20           0.1282        0.2353
0.40 5            0.3334        0.3794
     10           0.2479        0.2922
     20           0.1330        0.1996
0.60 5            0.2705        0.3344
     10           0.2353        0.3133
     20           0.2028        0.3050
0.80 5            0.3156        0.3444
     10           0.3711        0.4667
     20           0.3544        0.4750
```

Label Propagation

- Muy bajo rendimiento con p ≤ 0.20.
- Mejora moderadamente en p = 0.40–0.60.
- Mejor desempeño absoluto en p = 0.80
    - Macro-F1: 0.3470
    - Balanced Acc: 0.4287
- Conclusión: escala muy bien, pero necesita alta proporción de etiquetas.

Label Spreading

- Superior a LP en p bajos (0.05–0.20).
- Máximo rendimiento en p = 0.40
    - Macro-F1: 0.3208
    - Balanced Acc: 0.3806