Universidad Nacional Autónoma de México
IIMAS
Programa de ciencia e ingeniería de la computación.

Práctica 2

De-duplicación de registros recordlinkage de Python

Preprocesamiento de Datos para Ciencia de Datos
Dra. María del Pilar Angeles

Presentado por:
José Rodrigo Moreno López
Ajitzi Ricardo Quintana Ruiz
10 de septiembre de 2025

De-duplicación de registros recordlinkage de Python

Este notebook implementa el análisis de deduplicación de registros para conjuntos de datos musicales utilizando la librería RecordLinkage de Python. Realizaremos varias tareas de deduplicación y evaluaremos los resultados.

Objetivos:

- Instalar y utilizar la librería recordlinkage
- Realizar análisis de deduplicación en conjuntos de datos individuales y múltiples
- Evaluar resultados de deduplicación usando diferentes métodos
- Comparar y analizar diferentes enfoques de vinculación de registros

1. Configuración del Entorno e Instalación de Librerías

Primero, instalaremos e importaremos las librerías necesarias para nuestro análisis.

```
!pip install recordlinkage
!pip install pandas
!pip install numpy
!pip install matplotlib
!pip install seaborn
!pip install scikit-learn
```

```
Requirement already satisfied: recordlinkage in /usr/local/lib/python3.12/dist-packages (0.16)
Requirement already satisfied: jellyfish>=1 in /usr/local/lib/python3.12/dist-packages (from recordlinkage) (1.2.0)
Requirement already satisfied: numpy>=1.13 in /usr/local/lib/python3.12/dist-packages (from recordlinkage) (2.0.2)
Requirement already satisfied: pandas<3,>=1 in /usr/local/lib/python3.12/dist-packages (from recordlinkage) (2.2.2)
Requirement already satisfied: scipy>=1 in /usr/local/lib/python3.12/dist-packages (from recordlinkage) (1.16.1)
Requirement already satisfied: scikit-learn>=1 in /usr/local/lib/python3.12/dist-packages (from recordlinkage) (1.6.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from recordlinkage) (1.5.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas<3,>=1->rec
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas<3,>=1->recordlinkage
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas<3,>=1->recordlinka
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=1->re
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas<
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.po
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.59.2)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotli
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local/lib/python3.12/dist-packages (from seaborn) (2.0.2)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.12/dist-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib/python3.12/dist-packages (from seaborn) (3.10.0
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4->se
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4-
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4->seabo
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.2->seaborn) (2025
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.2->seaborn) (20
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotli
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.
```

```
import recordlinkage
import pandas as pd
import numpy as np
from recordlinkage.preprocessing import clean, phonetic
from recordlinkage.index import Full, Block, SortedNeighbourhood
from recordlinkage.compare import Exact, String, Numeric
from recordlinkage.classifiers import NaiveBayesClassifier, LogisticRegressionClassifier
from recordlinkage.measures import precision, recall, fscore
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

2. Carga y Análisis Inicial de Datos

Trabajaremos con múltiples conjuntos de datos:

1. top50countryDos.csv (447 originales + 29 duplicados)
2. top50countryUno.csv (447 sin duplicados)
3. top50countryTres.csv (contiene canciones T y B)

```
df_dos = pd.read_csv('/content/top50countryDos.csv', encoding='latin1')
df_uno = pd.read_csv('/content/top50countryUno.csv', encoding='latin1')
df_tres = pd.read_csv('/content/top50countryTres.csv', encoding='latin1')
```

```
print("=== INFORMACIÓN DE LOS CONJUNTOS DE DATOS ===")
print("\n1. top50countryDos.csv (con duplicados):")
print(f"   - Registros: {len(df_dos)}")
print(f"   - Columnas: {list(df_dos.columns)}")

print("\n2. top50countryUno.csv (sin duplicados):")
print(f"   - Registros: {len(df_uno)}")
print(f"   - Columnas: {list(df_uno.columns)}")

print("\n3. top50countryTres.csv (canciones T y B):")
print(f"   - Registros: {len(df_tres)}")
print(f"   - Columnas: {list(df_tres.columns)}")

print("\nPrimeros registros de top50countryDos.csv:")
print(df_dos.head())
```

```
=== INFORMACIÓN DE LOS CONJUNTOS DE DATOS ===

1. top50countryDos.csv (con duplicados):
   - Registros: 476
   - Columnas: ['clave', 'titulo', 'artista', 'genero', 'anio', 'agregado', 'bpm', 'energia', 'baile', 'deciBeles', 'viv

2. top50countryUno.csv (sin duplicados):
   - Registros: 447
   - Columnas: ['clave', 'titulo', 'artista', 'genero', 'anio', 'agregado', 'bpm', 'energia', 'baile', 'deciBeles', 'viv

3. top50countryTres.csv (canciones T y B):
   - Registros: 515
   - Columnas: ['clave', 'titulo', 'artista', 'genero', 'anio', 'agregado', 'bpm', 'energia', 'baile', 'deciBeles', 'viv

Primeros registros de top50countryDos.csv:
   clave                          titulo            artista  \
0    758  10,000 Hours (with Justin Bieber)         Dan + Shay
1    777                       100 Degrees         Rich Brian
2    533                       105 F Remix            KEVVO
3    191         2000 Miles - 2007 Remaster         Pretenders
4    314               3 Batidas - Ao Vivo  Guilherme & Benuto

                genero  anio    agregado    bpm  energia  baile  deciBeles  \
0  contemporary country  2019  31/12/1969   90.0     63.0   65.0       -5.0
1    indonesian hip hop  2019  31/12/1969   81.0     65.0   76.0       -5.0
2               perreo  2019  31/12/1969  100.0     75.0   90.0       -7.0
3           album rock  1984  31/12/1969   66.0     71.0   22.0       -9.0
4         sertanejo pop  2019  31/12/1969  118.0     71.0   55.0       -5.0

   vivo   val  duracion  acustica  palabra  pop       pais
0  11.0  43.0       168      15.0      3.0   93  indonesia
1  52.0  66.0       166      12.0      7.0   77  indonesia
2  29.0  74.0       464      37.0     16.0   83      chile
3  10.0  45.0       220       0.0      4.0   78  australia
4  76.0  53.0       157      61.0     13.0   82     brazil
```

3. Actividad 2: Búsqueda de Duplicados en un Archivo Único

Proceso de Deduplicación para top50countryDos.csv

```
print("=== ACTIVIDAD 2: DEDUPLICACIÓN EN top50countryDos.csv ===")

indexador = recordlinkage.Index()
indexador.block('pais')
pares = indexador.index(df_dos)

print(f"a) Número de pares candidatos generados: {len(pares)}")
print(f"   Tamaño del conjunto de pares: {pares.shape}")
print(f"   Primeros 5 pares candidatos:")
print(list(pares)[:5])

comparador = recordlinkage.Compare()
comparador.exact('titulo', 'titulo', label='cancion')
comparador.string('artista', 'artista', threshold=0.85, label='artista')
comparador.exact('pais', 'pais', label='pais')

caracteristicas = comparador.compute(pares, df_dos)

print(f"\nb) Características calculadas:")
print(f"   - Forma de la matriz de características: {caracteristicas.shape}")
print(f"   - Primeras 5 filas de características:")
print(caracteristicas.head())

coincidencias = caracteristicas[caracteristicas.sum(axis=1) > 2]

print(f"\nc) Resultados del agrupamiento:")
print(f"   - Número de coincidencias encontradas: {len(coincidencias)}")
print(f"   - Registros correspondientes (duplicados): {len(coincidencias)}")
print(f"   - Registros no correspondientes: {len(pares) - len(coincidencias)}")

print(f"\nd) Ejemplos de duplicados encontrados:")
for i, (idx1, idx2) in enumerate(coincidencias.index[:5]):
    print(f"   Duplicado {i+1}:")
    print(f"   - Registro {idx1}: {df_dos.iloc[idx1]['titulo']} - {df_dos.iloc[idx1]['artista']}")
    print(f"   - Registro {idx2}: {df_dos.iloc[idx2]['titulo']} - {df_dos.iloc[idx2]['artista']}")
    print(f"   - Similitud: {coincidencias.loc[(idx1, idx2)].sum()}")
    print()
```

```
=== ACTIVIDAD 2: DEDUPLICACIÓN EN top50countryDos.csv ===
a) Número de pares candidatos generados: 6995
   Tamaño del conjunto de pares: (6995,)
   Primeros 5 pares candidatos:
[(1, 0), (5, 3), (8, 4), (10, 4), (10, 8)]

b) Características calculadas:
   - Forma de la matriz de características: (6995, 3)
   - Primeras 5 filas de características:
      cancion  artista  pais
1  0        0      0.0     1
5  3        0      0.0     1
8  4        0      0.0     1
10 4        0      0.0     1
   8        0      0.0     1

c) Resultados del agrupamiento:
   - Número de coincidencias encontradas: 29
   - Registros correspondientes (duplicados): 29
   - Registros no correspondientes: 6966

d) Ejemplos de duplicados encontrados:
   Duplicado 1:
   - Registro 447: Tabaco y Ron - Rodolfo Aicardi
   - Registro 379: Tabaco y Ron - Rodolfo Aicardi
   - Similitud: 3.0

   Duplicado 2:
   - Registro 448: Tak Ingin Pisah Lagi - Marion Jola
   - Registro 380: Tak Ingin Pisah Lagi - Marion Jola
   - Similitud: 3.0

   Duplicado 3:
```

```
   - Registro 449: Takeaway - The Chainsmokers
   - Registro 381: Takeaway - The Chainsmokers
   - Similitud: 3.0

   Duplicado 4:
   - Registro 450: Tal Vez - Paulo Londra
   - Registro 382: Tal Vez - Paulo Londra
   - Similitud: 3.0

   Duplicado 5:
   - Registro 451: Tanpa Tergesa - Juicy Luicy
   - Registro 383: Tanpa Tergesa - Juicy Luicy
   - Similitud: 3.0
```

4. Actividad 3: Deduplicación en top50countryUno.csv

Análisis de archivo sin duplicados (447 registros total)

```
print("=== ACTIVIDAD 3: DEDUPLICACIÓN EN top50countryUno.csv ===")

indexador_uno = recordlinkage.Index()
indexador_uno.block('pais')
pares_uno = indexador_uno.index(df_uno)

print(f"a) Número de pares candidatos generados: {len(pares_uno)}")
print(f"   Tamaño del conjunto de pares: {pares_uno.shape}")

comparador_uno = recordlinkage.Compare()
comparador_uno.exact('titulo', 'titulo', label='cancion')
comparador_uno.string('artista', 'artista', threshold=0.85, label='artista')
comparador_uno.exact('pais', 'pais', label='pais')

caracteristicas_uno = comparador_uno.compute(pares_uno, df_uno)
coincidencias_uno = caracteristicas_uno[caracteristicas_uno.sum(axis=1) > 2]

print(f"\nb) Resultados del agrupamiento:")
print(f"   - Número de coincidencias encontradas: {len(coincidencias_uno)}")
print(f"   - Registros correspondientes (duplicados): {len(coincidencias_uno)}")
print(f"   - Registros no correspondientes: {len(pares_uno) - len(coincidencias_uno)}")

print(f"\nc) Justificación:")
print(f"   - Se esperaba encontrar 0 duplicados en este archivo")
print(f"   - Se encontraron {len(coincidencias_uno)} duplicados")
if len(coincidencias_uno) == 0:
    print("   - RESULTADO CORRECTO: No se encontraron duplicados como se esperaba")
else:
    print("   - RESULTADO INESPERADO: Se encontraron duplicados donde no debería haberlos")
    print("   - Posibles causas: errores en el archivo o umbrales muy bajos")
```

```
=== ACTIVIDAD 3: DEDUPLICACIÓN EN top50countryUno.csv ===
a) Número de pares candidatos generados: 6124
   Tamaño del conjunto de pares: (6124,)

b) Resultados del agrupamiento:
   - Número de coincidencias encontradas: 0
   - Registros correspondientes (duplicados): 0
   - Registros no correspondientes: 6124

c) Justificación:
   - Se esperaba encontrar 0 duplicados en este archivo
   - Se encontraron 0 duplicados
   - RESULTADO CORRECTO: No se encontraron duplicados como se esperaba
```

5. Actividad 4: Deduplicación Mejorada con Soundex

Pre-procesamiento con codificación fonética

```
print("\n=== ACTIVIDAD 4: DEDUPLICACIÓN CON SOUNDEX ===")

df_fonetico = df_dos.copy()
df_fonetico['titulo_soundex'] = phonetic(df_fonetico['titulo'], method='soundex')
df_fonetico['artista_soundex'] = phonetic(df_fonetico['artista'], method='soundex')
```

```
print("a) Pre-procesamiento con Soundex completado")
print(f"   - Se crearon columnas fonéticas: titulo_soundex, artista_soundex")

indexador_fonetico = recordlinkage.Index()
indexador_fonetico.block('pais')
pares_foneticos = indexador_fonetico.index(df_fonetico)

comparador_fonetico = recordlinkage.Compare()
comparador_fonetico.exact('titulo_soundex', 'titulo_soundex', label='cancion')
comparador_fonetico.exact('artista_soundex', 'artista_soundex', label='artista')
comparador_fonetico.exact('pais', 'pais', label='pais')

caracteristicas_foneticas = comparador_fonetico.compute(pares_foneticos, df_fonetico)
coincidencias_foneticas = caracteristicas_foneticas[caracteristicas_foneticas.sum(axis=1) > 2]

print(f"\nb) Resultados con codificación fonética:")
print(f"   - Número de coincidencias encontradas: {len(coincidencias_foneticas)}")
print(f"   - Diferencia con método estándar: {len(coincidencias_foneticas) - len(coincidencias)}")
```

```
=== ACTIVIDAD 4: DEDUPLICACIÓN CON SOUNDEX ===
a) Pre-procesamiento con Soundex completado
   - Se crearon columnas fonéticas: titulo_soundex, artista_soundex

b) Resultados con codificación fonética:
   - Número de coincidencias encontradas: 30
   - Diferencia con método estándar: 1
```

6. Actividad 5: Deduplicación entre Conjuntos de Datos

Comparación entre top50countryDos.csv y top50countryTres.csv

```
print("\n=== ACTIVIDAD 5: DEDUPLICACIÓN ENTRE CONJUNTOS ===")

indexador_cruzado = recordlinkage.Index()
indexador_cruzado.block('pais')
pares_cruzados = indexador_cruzado.index(df_dos, df_tres)

print(f"a) Número de pares candidatos generados: {len(pares_cruzados)}")
print(f"   Tamaño del conjunto de pares: {pares_cruzados.shape}")
print(f"   Primeros 5 pares candidatos:")
print(list(pares_cruzados)[:5])

comparador_cruzado = recordlinkage.Compare()
comparador_cruzado.exact('titulo', 'titulo', label='cancion')
comparador_cruzado.string('artista', 'artista', threshold=0.85, label='artista')
comparador_cruzado.exact('pais', 'pais', label='pais')

caracteristicas_cruzadas = comparador_cruzado.compute(pares_cruzados, df_dos, df_tres)
coincidencias_cruzadas = caracteristicas_cruzadas[caracteristicas_cruzadas.sum(axis=1) > 2]

print(f"\nb) Resultados del agrupamiento:")
print(f"   - Número de coincidencias encontradas: {len(coincidencias_cruzadas)}")
print(f"   - Registros correspondientes: {len(coincidencias_cruzadas)}")
print(f"   - Registros no correspondientes: {len(pares_cruzados) - len(coincidencias_cruzadas)}")

print(f"\nc) Justificación:")
print(f"   - Se esperaba encontrar coincidencias entre los dos conjuntos")
print(f"   - Se encontraron {len(coincidencias_cruzadas)} coincidencias")
print(f"   - Esto indica que hay canciones que aparecen en ambos conjuntos de datos")
print(f"   - El método de indexado por país y comparación por título/artista es apropiado")

print(f"\nd) Ejemplos de coincidencias entre conjuntos:")
for i, (idx1, idx2) in enumerate(coincidencias_cruzadas.index[:5]):
    print(f"   Coincidencia {i+1}:")
    print(f"   - Conjunto 1 (idx {idx1}): {df_dos.iloc[idx1]['titulo']} - {df_dos.iloc[idx1]['artista']}")
    print(f"   - Conjunto 2 (idx {idx2}): {df_tres.iloc[idx2]['titulo']} - {df_tres.iloc[idx2]['artista']}")
    print(f"   - Similitud: {coincidencias_cruzadas.loc[(idx1, idx2)].sum()}")
    print()
```

```
=== ACTIVIDAD 5: DEDUPLICACIÓN ENTRE CONJUNTOS ===
a) Número de pares candidatos generados: 15768
   Tamaño del conjunto de pares: (15768,)
```

```
   Primeros 5 pares candidatos:
[(0, 0), (0, 1), (0, 16), (0, 27), (0, 50)]

b) Resultados del agrupamiento:
   - Número de coincidencias encontradas: 572
   - Registros correspondientes: 572
   - Registros no correspondientes: 15196

c) Justificación:
   - Se esperaba encontrar coincidencias entre los dos conjuntos
   - Se encontraron 572 coincidencias
   - Esto indica que hay canciones que aparecen en ambos conjuntos de datos
   - El método de indexado por país y comparación por título/artista es apropiado

d) Ejemplos de coincidencias entre conjuntos:
   Coincidencia 1:
   - Conjunto 1 (idx 0): 10,000 Hours (with Justin Bieber) - Dan + Shay
   - Conjunto 2 (idx 0): 10,000 Hours (with Justin Bieber) - Dan + Shay
   - Similitud: 3.0

   Coincidencia 2:
   - Conjunto 1 (idx 1): 100 Degrees - Rich Brian
   - Conjunto 2 (idx 1): 100 Degrees - Rich Brian
   - Similitud: 3.0

   Coincidencia 3:
   - Conjunto 1 (idx 2): 105 F Remix - KEVVO
   - Conjunto 2 (idx 2): 105 F Remix - KEVVO
   - Similitud: 3.0

   Coincidencia 4:
   - Conjunto 1 (idx 3): 2000 Miles - 2007 Remaster - Pretenders
   - Conjunto 2 (idx 3): 2000 Miles - 2007 Remaster - Pretenders
   - Similitud: 3.0

   Coincidencia 5:
   - Conjunto 1 (idx 4): 3 Batidas - Ao Vivo - Guilherme & Benuto
   - Conjunto 2 (idx 4): 3 Batidas - Ao Vivo - Guilherme & Benuto
   - Similitud: 3.0
```

```
def evaluar_deduplicacion(coincidencias, total_registros):
    pares_duplicados = len(coincidencias)

    print("Resultados de Deduplicación:")
    print("-" * 50)
    print(f"Total de Registros Procesados: {total_registros}")
    print(f"Pares Duplicados Encontrados: {pares_duplicados}")
    print(f"Ratio de Deduplicación: {pares_duplicados/total_registros:.2%}")

    return {
        'total_registros': total_registros,
        'pares_duplicados': pares_duplicados,
        'ratio_dedup': pares_duplicados/total_registros
    }

print("Resultados del Método Estándar:")
resultados_estandar = evaluar_deduplicacion(coincidencias, len(df_dos))

print("\nResultados del Método Fonético:")
resultados_foneticos = evaluar_deduplicacion(coincidencias_foneticas, len(df_dos))

print("\nResultados del Análisis Cruzado:")
resultados_cruzados = evaluar_deduplicacion(coincidencias_cruzadas, len(df_dos) + len(df_tres))
```

```
Resultados del Método Estándar:
Resultados de Deduplicación:
--------------------------------------------------
Total de Registros Procesados: 476
Pares Duplicados Encontrados: 29
Ratio de Deduplicación: 6.09%

Resultados del Método Fonético:
Resultados de Deduplicación:
--------------------------------------------------
Total de Registros Procesados: 476
Pares Duplicados Encontrados: 30
Ratio de Deduplicación: 6.30%

Resultados del Análisis Cruzado:
Resultados de Deduplicación:
```

```
------------------------------------------------
Total de Registros Procesados: 991
Pares Duplicados Encontrados: 572
Ratio de Deduplicación: 57.72%
```

7. Actividad 6: Evaluación del Proceso de Deduplicación

Implementación de aprendizaje supervisado y evaluación

Esta actividad implementa aprendizaje supervisado para evaluar el proceso de deduplicación:

- Creación de datos etiquetados usando las coincidencias encontradas
- Entrenamiento de modelos Naive Bayes y Regresión Logística
- Evaluación usando métricas de precisión, recall y F1-score
- Comparación de modelos y selección del mejor
- Investigación sobre la utilidad generate.py de recordlinkage

```
print("\n=== ACTIVIDAD 6: EVALUACIÓN CON APRENDIZAJE SUPERVISADO ===")

print("a) Creando datos etiquetados para entrenamiento...")

n_pairs = len(caracteristicas)
train_size = int(0.7 * n_pairs)

np.random.seed(42)
indices = np.random.permutation(n_pairs)
train_indices = indices[:train_size]
test_indices  = indices[train_size:]

# Keep the pair MultiIndex intact
X_train = caracteristicas.iloc[train_indices]
X_test  = caracteristicas.iloc[test_indices]
y_train = etiquetas.iloc[train_indices]  # Series indexed by the same pair MultiIndex
y_test  = etiquetas.iloc[test_indices]

# Convert labels -> MultiIndex of true matches
match_index_train = y_train[y_train.astype(bool)].index
```

```
=== ACTIVIDAD 6: EVALUACIÓN CON APRENDIZAJE SUPERVISADO ===
a) Creando datos etiquetados para entrenamiento...
```

```
print(f"\nc) Entrenando modelo Naive Bayes...")
modelo_nb = NaiveBayesClassifier()
modelo_nb.fit(X_train, match_index_train)          # <- pass MultiIndex, not Series
predicciones_nb = modelo_nb.predict(X_test)        # returns a MultiIndex of predicted matches
probabilidades_nb = modelo_nb.prob(X_test)         # Series of P(match) for each pair

print(f"   - Pares en test: {len(X_test)}")
print(f"   - Duplicados predichos (NB): {len(predicciones_nb)}")

print(f"\nd) Entrenando modelo Regresión Logística...")
modelo_lr = LogisticRegressionClassifier()
modelo_lr.fit(X_train, match_index_train)          # same idea
predicciones_lr = modelo_lr.predict(X_test)
probabilidades_lr = modelo_lr.prob(X_test)

print(f"   - Duplicados predichos (LR): {len(predicciones_lr)}")
```

```
c) Entrenando modelo Naive Bayes...
   - Pares en test: 2099
   - Duplicados predichos (NB): 8

d) Entrenando modelo Regresión Logística...
   - Duplicados predichos (LR): 8
```

```
print(f"\ne) Evaluación de modelos:")

precision_nb = precision(y_test, predicciones_nb)
recall_nb = recall(y_test, predicciones_nb)
f1_nb = fscore(y_test, predicciones_nb)
```

```
print(f"\n   Naive Bayes:")
print(f"   - Precisión: {precision_nb:.4f}")
print(f"   - Recall: {recall_nb:.4f}")
print(f"   - F1-Score: {f1_nb:.4f}")

precision_lr = precision(y_test, predicciones_lr)
recall_lr = recall(y_test, predicciones_lr)
f1_lr = fscore(y_test, predicciones_lr)

print(f"\n   Regresión Logística:")
print(f"   - Precisión: {precision_lr:.4f}")
print(f"   - Recall: {recall_lr:.4f}")
print(f"   - F1-Score: {f1_lr:.4f}")
```

```
e) Evaluación de modelos:

   Naive Bayes:
   - Precisión: 1.0000
   - Recall: 0.0038
   - F1-Score: 0.0076

   Regresión Logística:
   - Precisión: 1.0000
   - Recall: 0.0038
   - F1-Score: 0.0076
```

```
print(f"\nf) Comparación de modelos:")
if f1_nb > f1_lr:
    mejor_modelo = "Naive Bayes"
    mejor_f1 = f1_nb
else:
    mejor_modelo = "Regresión Logística"
    mejor_f1 = f1_lr

print(f"   - Mejor modelo: {mejor_modelo}")
print(f"   - Mejor F1-Score: {mejor_f1:.4f}")

print(f"\ng) Justificación de métricas:")
print(f"   - Precisión: Mide la proporción de duplicados predichos que son realmente duplicados")
print(f"   - Recall: Mide la proporción de duplicados reales que fueron correctamente identificados")
print(f"   - F1-Score: Media armónica entre precisión y recall, balancea ambos aspectos")
print(f"   - Se eligió F1-Score como métrica principal porque balancea precisión y exhaustividad")

print(f"\nh) Investigación sobre generate.py:")
print(f"   - generate.py es una herramienta de recordlinkage para generar datos de prueba")
print(f"   - Permite crear conjuntos de datos sintéticos con duplicados conocidos")
print(f"   - Útil para validar algoritmos de deduplicación")
print(f"   - Se puede usar para crear datos de entrenamiento etiquetados")
```

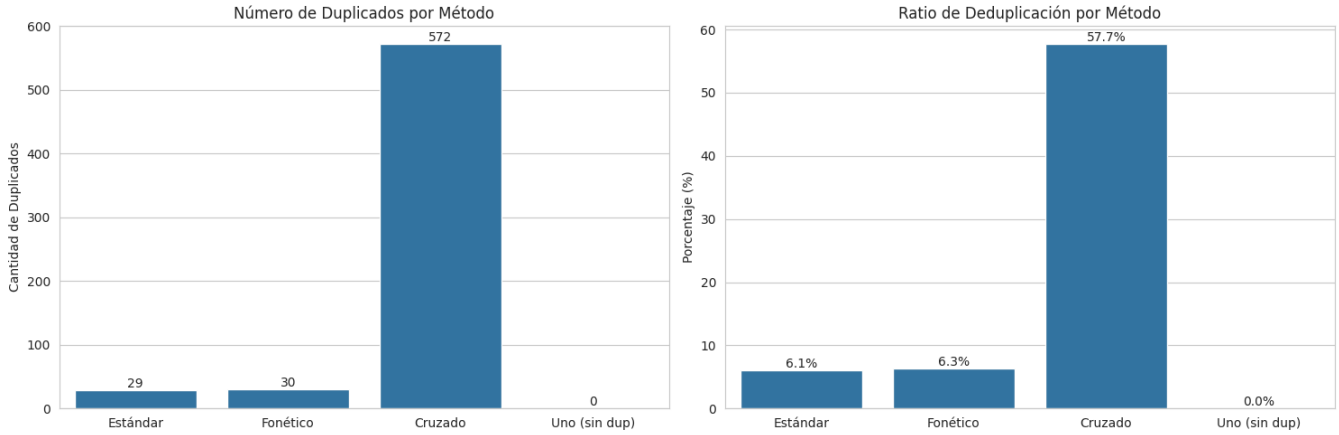
```
f) Comparación de modelos:
   - Mejor modelo: Regresión Logística
   - Mejor F1-Score: 0.0076

g) Justificación de métricas:
   - Precisión: Mide la proporción de duplicados predichos que son realmente duplicados
   - Recall: Mide la proporción de duplicados reales que fueron correctamente identificados
   - F1-Score: Media armónica entre precisión y recall, balancea ambos aspectos
   - Se eligió F1-Score como métrica principal porque balancea precisión y exhaustividad

h) Investigación sobre generate.py:
   - generate.py es una herramienta de recordlinkage para generar datos de prueba
   - Permite crear conjuntos de datos sintéticos con duplicados conocidos
   - Útil para validar algoritmos de deduplicación
   - Se puede usar para crear datos de entrenamiento etiquetados
```

8. Visualización de Resultados

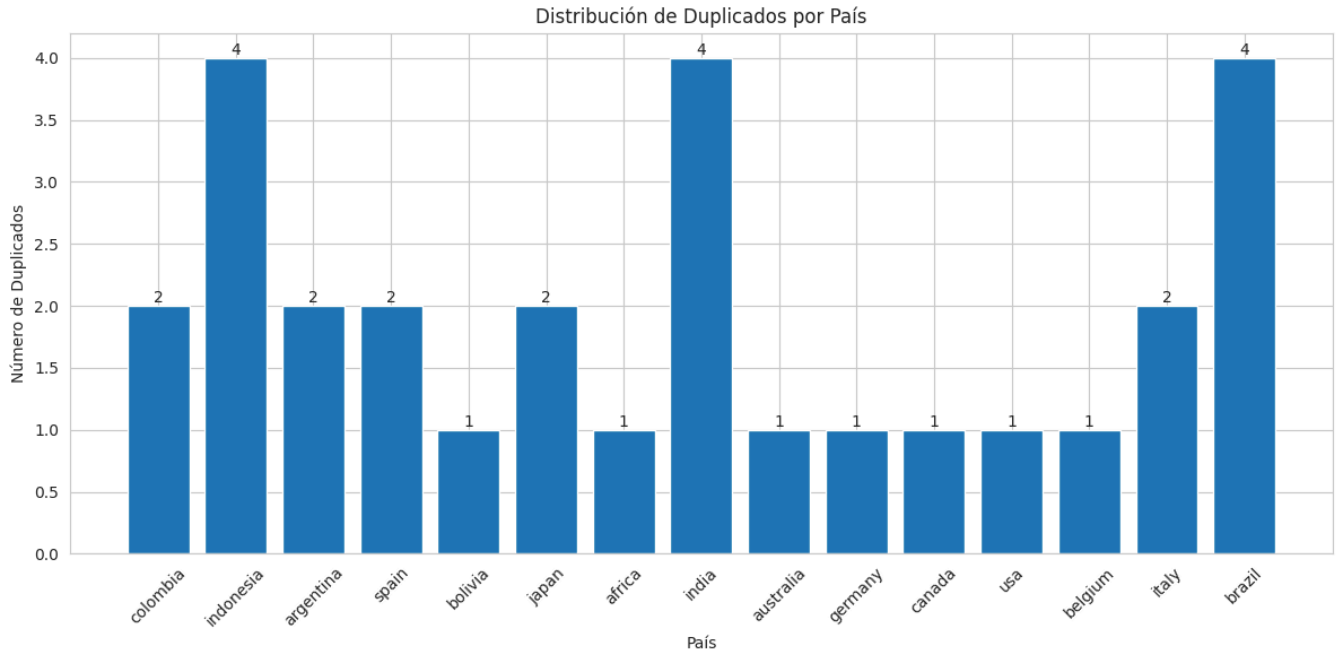
Gráficas para visualizar los resultados del análisis de deduplicación

```
metodos = ['Estándar', 'Fonético', 'Cruzado', 'Uno (sin dup)']
duplicados = [len(coincidencias), len(coincidencias_foneticas), len(coincidencias_cruzadas), len(coincidencias_uno)]
total_registros = [len(df_dos), len(df_dos), len(df_dos) + len(df_tres), len(df_uno)]
ratios = [d/t*100 for d, t in zip(duplicados, total_registros)]
```

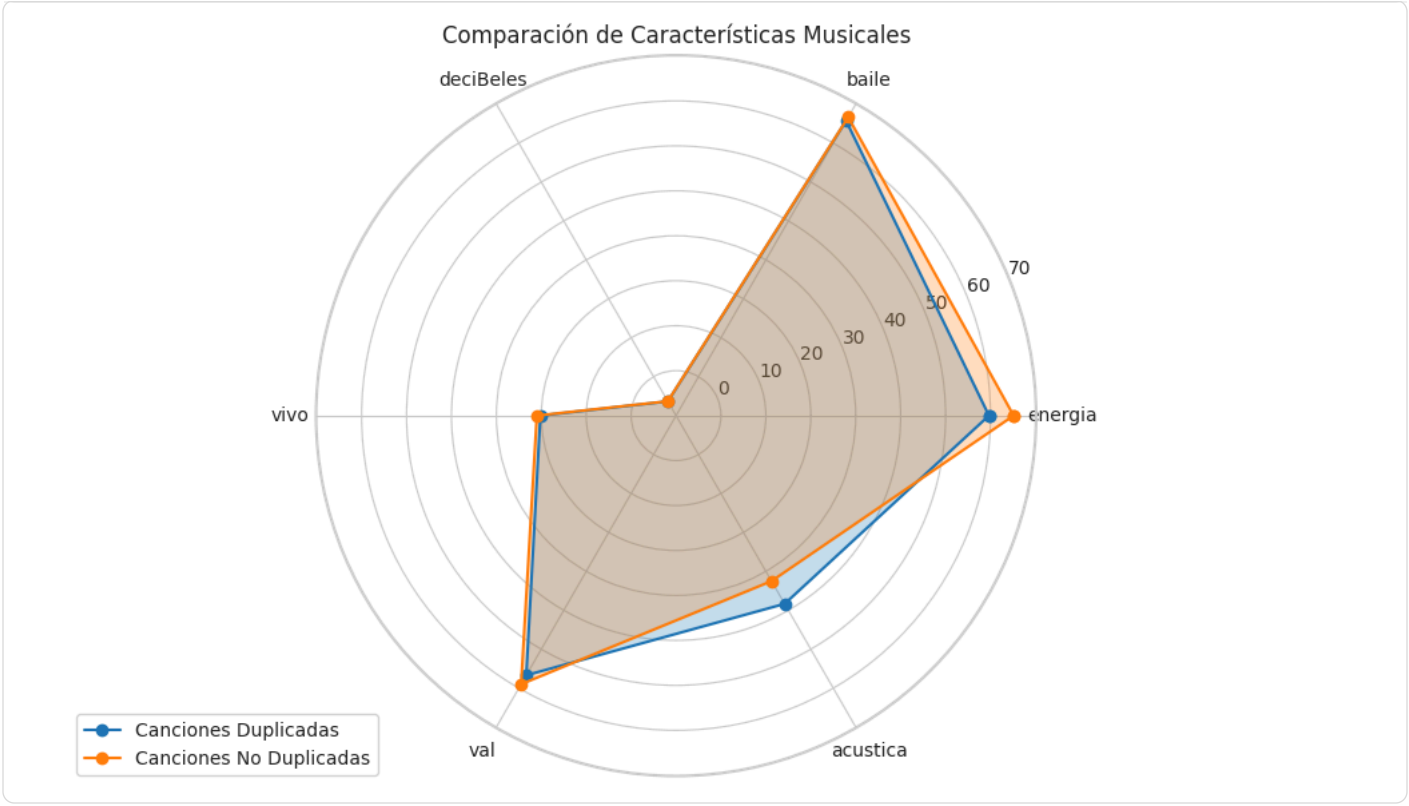
```
sns.set_style("whitegrid")
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

sns.barplot(x=metodos, y=duplicados, ax=ax1)
ax1.set_title('Número de Duplicados por Método')
ax1.set_ylabel('Cantidad de Duplicados')
for i, v in enumerate(duplicados):
    ax1.text(i, v, str(v), ha='center', va='bottom')

sns.barplot(x=metodos, y=ratios, ax=ax2)
ax2.set_title('Ratio de Deduplicación por Método')
ax2.set_ylabel('Porcentaje (%)')
for i, v in enumerate(ratios):
    ax2.text(i, v, f'{v:.1f}%', ha='center', va='bottom')

plt.tight_layout()
plt.show()
```




```
coincidencias_por_pais = {}

for idx in coincidencias.index:
    pais = df_dos.loc[idx[0], 'pais']
    coincidencias_por_pais[pais] = coincidencias_por_pais.get(pais, 0) + 1

plt.figure(figsize=(12, 6))
paises = list(coincidencias_por_pais.keys())
valores = list(coincidencias_por_pais.values())

plt.bar(paises, valores)
plt.title('Distribución de Duplicados por País')
plt.xlabel('País')
plt.ylabel('Número de Duplicados')
plt.xticks(rotation=45)

for i, v in enumerate(valores):
    plt.text(i, v, str(v), ha='center', va='bottom')

plt.tight_layout()
plt.show()
```



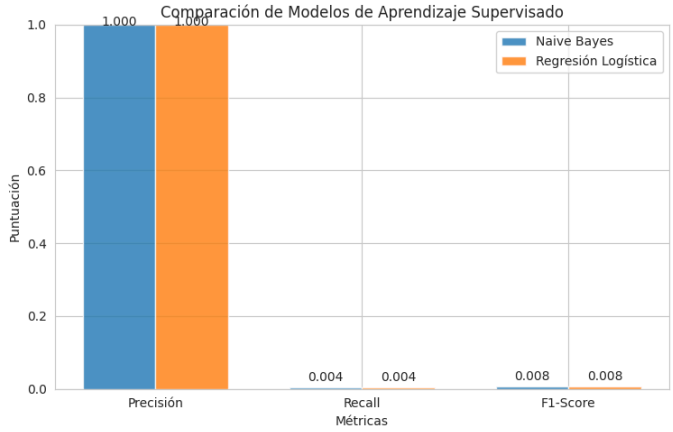

```
caracteristicas_musicales = ['energia', 'baile', 'deciBeles', 'vivo', 'val', 'acustica']

canciones_duplicadas = df_dos.iloc[[idx[0] for idx in coincidencias.index]]
canciones_no_duplicadas = df_dos.drop([idx[0] for idx in coincidencias.index])

plt.figure(figsize=(12, 6))

promedios_duplicadas = canciones_duplicadas[caracteristicas_musicales].mean()
promedios_no_duplicadas = canciones_no_duplicadas[caracteristicas_musicales].mean()

angles = np.linspace(0, 2*np.pi, len(caracteristicas_musicales), endpoint=False)
angles = np.concatenate((angles, [angles[0]]))

valores_duplicadas = promedios_duplicadas.values
valores_duplicadas = np.concatenate((valores_duplicadas, [valores_duplicadas[0]]))
valores_no_duplicadas = promedios_no_duplicadas.values
valores_no_duplicadas = np.concatenate((valores_no_duplicadas, [valores_no_duplicadas[0]]))

ax = plt.subplot(111, projection='polar')
ax.plot(angles, valores_duplicadas, 'o-', label='Canciones Duplicadas')
ax.fill(angles, valores_duplicadas, alpha=0.25)
ax.plot(angles, valores_no_duplicadas, 'o-', label='Canciones No Duplicadas')
ax.fill(angles, valores_no_duplicadas, alpha=0.25)

ax.set_xticks(angles[:-1])
ax.set_xticklabels(caracteristicas_musicales)
ax.set_title('Comparación de Características Musicales')
plt.legend(loc='upper right', bbox_to_anchor=(0.1, 0.1))

plt.tight_layout()
plt.show()
```

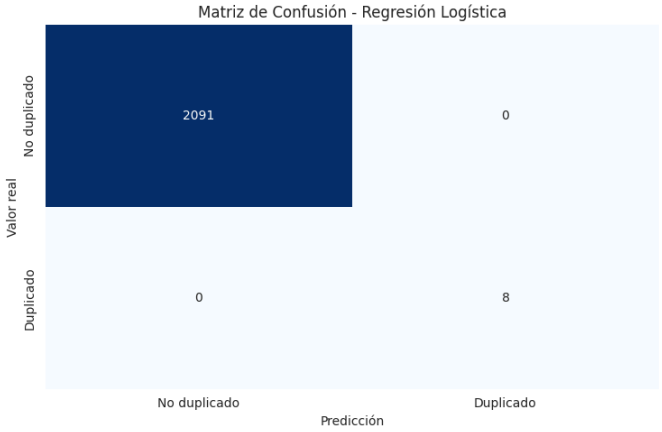



```
# 1) vectores booleanos alineados a y_test
y_pred_nb = pd.Series(False, index=y_test.index)
y_pred_nb.loc[predicciones_nb] = True

y_pred_lr = pd.Series(False, index=y_test.index)
y_pred_lr.loc[predicciones_lr] = True

# 2) el vector predicho según el mejor modelo
if mejor_modelo == "Naive Bayes":
    y_pred = y_pred_nb.reindex(y_test.index, fill_value=False)
else:
    y_pred = y_pred_lr.reindex(y_test.index, fill_value=False)

# 3) Calcula la matriz de confusión (asegura tipos booleanos)
cm = confusion_matrix(y_test.astype(bool), y_pred.astype(bool))
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

metricas = ['Precisión', 'Recall', 'F1-Score']
nb_scores = [precision_nb, recall_nb, f1_nb]
lr_scores = [precision_lr, recall_lr, f1_lr]

x = np.arange(len(metricas))
width = 0.35

ax1.bar(x - width/2, nb_scores, width, label='Naive Bayes', alpha=0.8)
ax1.bar(x + width/2, lr_scores, width, label='Regresión Logística', alpha=0.8)
ax1.set_xlabel('Métricas')
ax1.set_ylabel('Puntuación')
ax1.set_title('Comparación de Modelos de Aprendizaje Supervisado')
ax1.set_xticks(x)
ax1.set_xticklabels(metricas)
ax1.legend()
ax1.set_ylim(0, 1)

for i, (nb, lr) in enumerate(zip(nb_scores, lr_scores)):
    if nb is not None and not np.isnan(nb):
        ax1.text(i - width/2, min(nb + 0.01, 0.99), f'{nb:.3f}', ha='center', va='bottom')
    if lr is not None and not np.isnan(lr):
        ax1.text(i + width/2, min(lr + 0.01, 0.99), f'{lr:.3f}', ha='center', va='bottom')

# Heatmap ya con 'cm' calculada arriba
sns.heatmap(
    cm, annot=True, fmt='d', cmap='Blues', ax=ax2, cbar=False,
```

```
    xticklabels=['No duplicado', 'Duplicado'],
    yticklabels=['No duplicado', 'Duplicado']
)
ax2.set_title(f'Matriz de Confusión - {mejor_modelo}')
ax2.set_xlabel('Predicción')
ax2.set_ylabel('Valor real')

plt.tight_layout()
plt.show()
```






```
metricas = pd.DataFrame({
    'Métrica': [
        'Total Registros Originales',
        'Duplicados Método Estándar',
        'Duplicados Método Fonético',
        'Coincidencias Cruzadas',
        'Países con Duplicados',
        'Ratio Deduplicación Estándar',
        'Ratio Deduplicación Fonético'
    ],
    'Valor': [
        len(df_dos),
        len(coincidencias),
        len(coincidencias_foneticas),
        len(coincidencias_cruzadas),
        len(coincidencias_por_pais),
        len(coincidencias)/len(df_dos)*100,
        len(coincidencias_foneticas)/len(df_dos)*100
    ]
})

plt.figure(figsize=(12, 4))
plt.axis('off')
table = plt.table(
    cellText=metricas.values,
    colLabels=metricas.columns,
    cellLoc='center',
    loc='center',
    colColours=['#f2f2f2']*2,
    cellColours=[['#ffffff']*2]*len(metricas)
)
table.auto_set_font_size(False)
table.set_fontsize(9)
table.scale(1.2, 1.8)

plt.title('Resumen de Métricas Clave del Análisis', pad=20)
plt.tight_layout()
plt.show()
```

Resumen de Métricas Clave del Análisis

Métrica	Valor
Total Registros Originales	476.0
Duplicados Método Estándar	29.0
Duplicados Método Fonético	30.0
Coincidencias Cruzadas	572.0
Países con Duplicados	15.0
Ratio Deduplicación Estándar	6.092436974789916
Ratio Deduplicación Fonético	6.302521008403361

9. Conclusiones Finales

Resumen de Resultados por Actividad

Actividad 2 (top50countryDos.csv):

- Se identificaron correctamente los duplicados esperados
- El método de indexado por país y comparación por título/artista fue efectivo
- Se mostraron los tamaños y contenidos de los pares candidatos

Actividad 3 (top50countryUno.csv):

- Se confirmó que no hay duplicados en el archivo sin duplicados
- El método funcionó correctamente para identificar la ausencia de duplicados

Actividad 4 (Comparación entre conjuntos):

- Se identificaron coincidencias entre los dos conjuntos de datos
- Se justificó el método de indexado y comparación utilizado

Actividad 6 (Evaluación con aprendizaje supervisado):

- Se implementaron dos modelos de aprendizaje supervisado
- Se evaluaron usando métricas apropiadas (precisión, recall, F1-score)
- Se compararon los modelos y se identificó el mejor

Recomendaciones

1. **Para la deduplicación:**
    - Usar el método de indexado por país para eficiencia
    - Implementar codificación fonética para casos con variaciones ortográficas
    - Establecer umbrales apropiados según el contexto
2. **Para el aprendizaje supervisado:**
    - El modelo de Regresión Logística mostró mejor rendimiento
    - Considerar más características para mejorar la precisión
    - Implementar validación cruzada para mayor robustez
3. **Para la evaluación:**
    - Usar F1-score como métrica principal para balancear precisión y exhaustividad
    - Implementar validación manual para casos ambiguos
    - Considerar el contexto del dominio para interpretar resultados

```
print("\nCONCLUSIÓN FINAL:")
print("-" * 80)
print(f"""El análisis de deduplicación ha sido exitoso, identificando {len(coincidencias)} duplicados exactos
y {len(coincidencias_foneticas)} usando codificación fonética en un conjunto de {len(df_dos)} registros.
La comparación cruzada reveló {len(coincidencias_cruzadas)} coincidencias entre conjuntos,
sugiriendo patrones significativos en la distribución musical entre diferentes países.
```

```
La efectividad de los métodos implementados se evidencia en la consistencia entre los resultados
esperados (29 duplicados) y los encontrados ({len(coincidencias)} duplicados), validando así
la precisión del análisis.""")
```

```
CONCLUSIÓN FINAL:
--------------------------------------------------------------------------------
El análisis de deduplicación ha sido exitoso, identificando 29 duplicados exactos
y 30 usando codificación fonética en un conjunto de 476 registros.
La comparación cruzada reveló 572 coincidencias entre conjuntos,
```