Ejercicio 1 — Transcripción de audios y extracción de entidades (NER)

En este ejercicio vamos a resolver el problema en dos etapas: primero convertimos los audios en español a texto mediante **Whisper** y, después, aplicamos **NER** con el modelo preentrenado `MMG/xlm-roberta-large-ner-spanish` para detectar entidades como personas (PER), organizaciones (ORG), ubicaciones (LOC), fechas (DATE), etc. La salida solicitada por el enunciado será un **JSON por audio** y un **JSON global** con el identificador del archivo, la transcripción y las entidades agrupadas por tipo.

Pasos a seguir

1. Preparación del entorno (Colab + Drive + rutas)
2. Verificación del entorno (Python y GPU)
3. Instalación de dependencias (Whisper y Transformers)
4. Prueba piloto con un audio
5. Procesamiento por lotes y guardado de resultados (JSON)
6. Validación rápida y resumen

1.Rutas, estructura y README

Trabajamos habitualmente en **Google Drive** para guardar y reanudar fácilmente el ejercicio (`USE_DRIVE=True`).
Para la **entrega**, usamos **rutas relativas** con `USE_DRIVE=False`, de modo que el cuaderno sea **portable** y se pueda **re-ejecutar** sin depender de Drive. El cuaderno es **reproducible**: al volver a ejecutarlo, transcribe de nuevo y regenera los JSON/CSV.

Estructura creada+:

- `notebooks/`, `src/`, `scripts/` (aquí dejaremos el **.py** exportado)
- `data/audio/` (aquí se copian los **audios**; no van en el ZIP)
- `outputs/json/`, `outputs/csv/`, `outputs/figures/`
- `outputs/docs/` (aquí guardaremos el **PDF** del informe)
- `outputs/logs/`

El README indica dónde colocar **datos**, **.py** y **PDF**, y qué salidas se generan.

```
#Rutas + estructura + README
from pathlib import Path

USE_DRIVE = True  #True (monta Drive)
                  #False (rutas relativas, sin Drive)

if USE_DRIVE:
    try:
        from google.colab import drive  # solo si estamos en Colab
        drive.mount('/content/drive')
    except Exception as e:
        print("Aviso: no se pudo montar Drive (¿no estás en Colab?).", e)
    ROOT = Path("/content/drive/MyDrive/MASTER BIG DATA/md2_2025")
    EJ1  = ROOT / "ej1_transcripcion"
else:
```

```
    here = Path(".").resolve()
    EJ1  = here if (here.name == "ej1_transcripcion") else (here / "ej1_transcripcion")

PATHS = {
    "root":        str(EJ1.parent),
    "ej1":         str(EJ1),
    "nb":          str(EJ1 / "notebooks"),
    "src":         str(EJ1 / "src"),
    "scripts":     str(EJ1 / "scripts"),                # .py exportado
    "data_audio":  str(EJ1 / "data" / "audio"),
    "out_json":    str(EJ1 / "outputs" / "json"),
    "out_csv":     str(EJ1 / "outputs" / "csv"),
    "out_figs":    str(EJ1 / "outputs" / "figures"),
    "out_docs":    str(EJ1 / "outputs" / "docs"),       # PDF del informe
    "out_logs":    str(EJ1 / "outputs" / "logs"),
}

def ensure_dir(p: str): Path(p).mkdir(parents=True, exist_ok=True)
for k in ("nb","src","scripts","data_audio","out_json","out_csv","out_figs","out_docs","out_l
    ensure_dir(PATHS[k])

# README
readme = Path(PATHS["ej1"]) / "README.md"
if not readme.exists():
    readme.write_text("""# Ejercicio 1 — Transcripción + NER (ES)

**Objetivo.** Transcribimos audios en español con Whisper y extraemos entidades (PER/ORG/LOC/
**Notebook principal:** `notebooks/ej1_transcripcion.ipynb`

## Datos (no incluidos en el ZIP)
Copiar los audios en: `data/audio/` (formatos: wav, mp3, m4a, flac, ogg).

## Salidas
- `outputs/json/`  → JSON por audio + `resultados_global.json`
- `outputs/csv/`   → resúmenes tabulares (si se generan)
- `outputs/figures/` → figuras opcionales (si se generan)

## Entrega
- Exportar el notebook a **PDF** y guardarlo en `outputs/docs/`.
- Exportar opcionalmente a **.py** y colocarlo en `scripts/`.

## Requisitos (Colab o local)
- `ffmpeg`, `openai-whisper`, `transformers`, `torch` (y aceleración si hay GPU).
""", encoding="utf-8")
    print("README creado:", readme)
else:
    print("README ya existe: no se modifica.")

print("Rutas clave:")
for k in ("data_audio","out_json","out_csv","out_figs","out_docs","scripts"):
    print(f"  {k:>11} → {PATHS[k]}")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
README ya existe: no se modifica.
Rutas clave:
   data_audio → /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripcion/data/au
     out_json → /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripcion/outputs
      out_csv → /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripcion/outputs
     out_figs → /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripcion/outputs
     out_docs → /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripcion/outputs
      scripts → /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripcion/scripts
```

2.Verificación del entorno (Python y GPU)

Comprobamos la versión de Python y si disponemos de GPU. Esta información nos ayudará a decidir si ejecutamos los modelos en CPU o en GPU en los siguientes pasos.

```
# Inspección rápida del entorno

import platform
import torch

print("Python:", platform.python_version())
has_cuda = torch.cuda.is_available()
print("Torch CUDA disponible:", has_cuda)
if has_cuda:
    print("GPU:", torch.cuda.get_device_name(0))
else:
    print("Sin GPU.")
```

```
Python: 3.11.13
Torch CUDA disponible: True
GPU: Tesla T4
```

3.Instalación de dependencias

Instalamos las librerías necesarias para el ejercicio: **Whisper** (transcripción), **Transformers** (NER) y **ffmpeg** (necesario para procesar audio). Mantendremos las versiones provistas por Colab para maximizar la compatibilidad.

```
# Instalación de librerías necesarias
!pip -q install -U openai-whisper transformers accelerate datasets
!apt -q install -y ffmpeg
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.0/42.0 kB 2.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.3/11.3 MB 116.3 MB/s eta 0:00:00
Reading package lists...
Building dependency tree...
Reading state information...
ffmpeg is already the newest version (7:4.4.2-0ubuntu0.22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
```

```
### Comprobación de la instalación
import sys
import transformers, whisper

print("Python      :", platform.python_version())
print("Torch       :", torch.__version__)
print("Transformers:", transformers.__version__)
# confirmamos import y ruta de instalación
print("Whisper     : OK (", whisper.__file__, ")")
```

```
Python      : 3.11.13
Torch       : 2.6.0+cu124
Transformers: 4.55.1
Whisper     : OK ( /usr/local/lib/python3.11/dist-packages/whisper/__init__.py )
```

```
#Importación resto de librerias
# Estándar
```

```
import glob, json, csv
from datetime import datetime
from collections import Counter, defaultdict

# Terceros
from transformers import AutoTokenizer, AutoModelForTokenClassification, pipeline
```

4.Carga de audios de prueba

Listamos los audios disponibles en `ej1_transcripcion/data/audio/` y seleccionamos automáticamente el primero como muestra de trabajo. Este paso nos permite verificar que la carpeta de datos está correctamente configurada antes de continuar.

```
AUDIO_DIR = PATHS["data_audio"]

# Listamos extensiones comunes de audio por si en un futuro añadimos mas archivos
exts = ("*.wav", "*.mp3", "*.m4a", "*.flac", "*.ogg")
audio_paths = []
for ext in exts:
    audio_paths.extend(glob.glob(os.path.join(AUDIO_DIR, ext)))

audio_paths = sorted(audio_paths)

print("Carpeta de audio:", AUDIO_DIR)
print("Nº de archivos encontrados:", len(audio_paths))
for p in audio_paths[:10]:
    print(" -", os.path.basename(p))

# Seleccionamos un audio de ejemplo
AUDIO_SAMPLE = audio_paths[0] if audio_paths else None
print("Audio de ejemplo seleccionado:", AUDIO_SAMPLE)
```

```
Carpeta de audio: /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripcion/data/
Nº de archivos encontrados: 80
 - frase_01.wav
 - frase_02.wav
 - frase_03.wav
 - frase_04.wav
 - frase_05.wav
 - frase_06.wav
 - frase_07.wav
 - frase_08.wav
 - frase_09.wav
 - frase_10.wav
Audio de ejemplo seleccionado: /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transc
```

4.1Prueba piloto con un audio

En esta sección cargamos los modelos necesarios, transcribimos el audio de ejemplo y ejecutamos el reconocimiento de entidades (NER). Después agrupamos las entidades por tipo y construimos un diccionario con el formato de salida que utilizaremos más adelante (JSON), manteniéndolo de momento en memoria para validar el flujo.

Pasos:

1. Cargar Whisper (tamaño `base`) y el modelo NER `MMG/xlm-roberta-large-ner-spanish`.

2. Transcribir el archivo de audio seleccionado automáticamente.
3. Ejecutar NER sobre la transcripción.
4. Agrupar las entidades por tipo y mostrar un resumen (conteos por tipo y primeros ejemplos).

```
# 1) Dispositivo
device = "cuda" if torch.cuda.is_available() else "cpu"

# 2) Modelos
WHISPER_SIZE = "base"  # se podrá cambiar a 'small'/'medium' más adelante si lo necesitamos
NER_MODEL = "MMG/xlm-roberta-large-ner-spanish"

whisper_model = whisper.load_model(WHISPER_SIZE, device=device)

tokenizer = AutoTokenizer.from_pretrained(NER_MODEL)
ner_model = AutoModelForTokenClassification.from_pretrained(NER_MODEL)
ner_pipe = pipeline(
    "token-classification",
    model=ner_model,
    tokenizer=tokenizer,
    aggregation_strategy="simple",
    device=0 if device == "cuda" else -1
)

# 3) Audio de ejemplo (seleccionado en la sección anterior)
audio_path = AUDIO_SAMPLE
fname = os.path.basename(audio_path) if audio_path else None

# 4) Transcripción
transcripcion = ""
if audio_path:
    result = whisper_model.transcribe(audio_path, language="es", verbose=False)
    transcripcion = (result.get("text") or "").strip()

# 5) NER
entidades_por_tipo = {}
if transcripcion:
    ents = ner_pipe(transcripcion)
    for ent in ents:
        etype = ent.get("entity_group", "MISC")
        item = {
            "texto": ent.get("word", ""),
            "start": int(ent.get("start", 0)),
            "end": int(ent.get("end", 0)),
            "score": float(ent.get("score", 0.0)),
        }
        entidades_por_tipo.setdefault(etype, []).append(item)

# 6) Resultado piloto en memoria (formato final que usaremos para JSON)
resultado_piloto = {
    "archivo_audio": fname,
    "ruta_audio": audio_path,
    "transcripcion": transcripcion,
    "entidades": entidades_por_tipo,
    "modelo_whisper": WHISPER_SIZE,
    "modelo_ner": NER_MODEL,
    "timestamp": datetime.utcnow().isoformat() + "Z",
}

# 7) Resumen legible
print("Archivo:", resultado_piloto["archivo_audio"])
print("\nTranscripción (primeros 200 caracteres):")
```

```
print(resultado_piloto["transcripcion"][:200])

print("\nEntidades detectadas por tipo:")
for t, lista in resultado_piloto["entidades"].items():
    print(f" - {t}: {len(lista)}")

# 8) Muestra de las primeras 3 entidades de cada tipo (si existen)
for t, lista in resultado_piloto["entidades"].items():
    if not lista:
        continue
    print(f"\nEjemplos [{t}]:")
    for e in lista[:3]:
        print(f"  • {e['texto']} (score={e['score']:.3f})")
```

```
Device set to use cuda:0
100%|██████████| 405/405 [00:02<00:00, 181.94frames/s]
Archivo: frase_01.wav

Transcripción (primeros 200 caracteres):
Sofia viajó a Sevilla para asistir a un Congreso de Medicina.

Entidades detectadas por tipo:
 - PER: 1
 - LOC: 1
 - MISC: 1

Ejemplos [PER]:
  • Sofia (score=0.983)

Ejemplos [LOC]:
  • Sevilla (score=0.999)

Ejemplos [MISC]:
  • Congreso de Medicina (score=0.979)
```

5.Procesamiento por lotes y guardado de resultados (JSON)

Aplicamos el flujo validado a todos los audios de `ej1_transcripcion/data/audio/`. Para cada archivo generamos un **JSON individual** con: `archivo_audio`, `ruta_audio`, `transcripcion`, `entidades` (agrupadas por tipo), `modelo_whisper`, `modelo_ner` y `timestamp`. Además, consolidamos todo en un **JSON global**. Los guardados se realizan en `ej1_transcripcion/outputs/json/`.

```
# --- Utilidades ---
def transcribir_audio(path_audio, model, language="es"):
    """Transcribe un audio con Whisper y devuelve el texto (str)."""
    out = model.transcribe(path_audio, language=language, verbose=False)
    return (out.get("text") or "").strip()

def extraer_entidades(texto, ner_pipeline):
    """Ejecuta NER y agrupa por tipo. Devuelve dict {tipo: [..]}."""
    if not texto:
        return {}
    raw = ner_pipeline(texto)
    entidades = {}
    for ent in raw:
        etype = ent.get("entity_group", "MISC")
        item = {
            "texto": ent.get("word", ""),
            "start": int(ent.get("start", 0)),
            "end": int(ent.get("end", 0)),
```

```
            "score": float(ent.get("score", 0.0)),
        }
        entidades.setdefault(etype, []).append(item)
    return entidades

# --- Garantizar modelos en memoria (por si la sesión se reinició) ---
try:
    whisper_model
    ner_pipe
    WHISPER_SIZE
    NER_MODEL
except NameError:
    import torch, whisper
    from transformers import AutoTokenizer, AutoModelForTokenClassification, pipeline
    device = "cuda" if torch.cuda.is_available() else "cpu"
    WHISPER_SIZE = "base"
    NER_MODEL = "MMG/xlm-roberta-large-ner-spanish"
    whisper_model = whisper.load_model(WHISPER_SIZE, device=device)
    tok = AutoTokenizer.from_pretrained(NER_MODEL)
    ner_model = AutoModelForTokenClassification.from_pretrained(NER_MODEL)
    ner_pipe = pipeline(
        "token-classification",
        model=ner_model,
        tokenizer=tok,
        aggregation_strategy="simple",
        device=0 if device == "cuda" else -1
    )

# --- Localizar audios y preparar salida ---
AUDIO_DIR = PATHS["data_audio"]
OUT_JSON_DIR = PATHS["out_json"]
os.makedirs(OUT_JSON_DIR, exist_ok=True)

exts = ("*.wav", "*.mp3", "*.m4a", "*.flac", "*.ogg")
audio_paths = []
for ext in exts:
    audio_paths.extend(glob.glob(os.path.join(AUDIO_DIR, ext)))
audio_paths = sorted(audio_paths)

resultados = []
contador_tipos = Counter()
procesados = 0
fallidos = 0

# --- Bucle principal ---
for apath in audio_paths:
    fname = os.path.basename(apath)
    try:
        # 1) Transcripción
        texto = transcribir_audio(apath, whisper_model, language="es")

        # 2) NER
        entidades = extraer_entidades(texto, ner_pipe)

        # 3) Registro y guardado JSON individual
        registro = {
            "archivo_audio": fname,
            "ruta_audio": apath,
            "transcripcion": texto,
            "entidades": entidades,
            "modelo_whisper": WHISPER_SIZE,
            "modelo_ner": NER_MODEL,
```

```
            "timestamp": datetime.utcnow().isoformat() + "Z",
        }
        resultados.append(registro)

        # Acumulados
        for t, lst in entidades.items():
            contador_tipos[t] += len(lst)

        # JSON por audio
        out_path = os.path.join(OUT_JSON_DIR, f"{fname}.json")
        with open(out_path, "w", encoding="utf-8") as f:
            json.dump(registro, f, ensure_ascii=False, indent=2)

        procesados += 1

    except Exception as e:
        fallidos += 1
        # Guardamos un JSON de error para trazabilidad
        registro_err = {
            "archivo_audio": fname,
            "ruta_audio": apath,
            "error": str(e),
            "modelo_whisper": WHISPER_SIZE,
            "modelo_ner": NER_MODEL,
            "timestamp": datetime.utcnow().isoformat() + "Z",
        }
        resultados.append(registro_err)
        out_path = os.path.join(OUT_JSON_DIR, f"{fname}.error.json")
        with open(out_path, "w", encoding="utf-8") as f:
            json.dump(registro_err, f, ensure_ascii=False, indent=2)

# --- JSON global ---
GLOBAL_JSON = os.path.join(OUT_JSON_DIR, "resultados_global.json")
with open(GLOBAL_JSON, "w", encoding="utf-8") as f:
    json.dump(resultados, f, ensure_ascii=False, indent=2)

# --- Resumen en consola ---
print("Audios totales     :", len(audio_paths))
print("Procesados OK      :", procesados)
print("Con error          :", fallidos)
print("Salida (JSON por audio):", OUT_JSON_DIR)
print("JSON global        :", GLOBAL_JSON)
print("\nEntidades por tipo (acumuladas):")
for t, c in sorted(contador_tipos.items(), key=lambda x: (-x[1], x[0])):
    print(f" - {t}: {c}")
```

```
100%|██████████| 406/406 [00:00<00:00, 2015.35frames/s]
100%|██████████| 410/410 [00:00<00:00, 1662.17frames/s]
100%|██████████| 449/449 [00:00<00:00, 1424.53frames/s]
100%|██████████| 303/303 [00:00<00:00, 1673.23frames/s]
100%|██████████| 386/386 [00:00<00:00, 1977.30frames/s]
100%|██████████| 417/417 [00:00<00:00, 1651.54frames/s]
100%|██████████| 426/426 [00:00<00:00, 1998.30frames/s]
100%|██████████| 411/411 [00:00<00:00, 2105.55frames/s]
100%|██████████| 435/435 [00:00<00:00, 2366.57frames/s]
100%|██████████| 452/452 [00:00<00:00, 1607.54frames/s]
100%|██████████| 425/425 [00:00<00:00, 1236.21frames/s]
100%|██████████| 348/348 [00:00<00:00, 1913.66frames/s]
100%|██████████| 422/422 [00:00<00:00, 1891.85frames/s]
100%|██████████| 391/391 [00:00<00:00, 1783.21frames/s]
100%|██████████| 360/360 [00:00<00:00, 2116.85frames/s]
100%|██████████| 376/376 [00:00<00:00, 2124.96frames/s]
100%|██████████| 355/355 [00:00<00:00, 2163.42frames/s]
100%|██████████| 397/397 [00:00<00:00, 2012.23frames/s]
100%|██████████| 425/425 [00:00<00:00, 1925.84frames/s]
100%|██████████| 310/310 [00:00<00:00, 1133.91frames/s]
100%|██████████| 376/376 [00:00<00:00, 1467.40frames/s]
100%|██████████| 415/415 [00:00<00:00, 2120.87frames/s]
100%|██████████| 424/424 [00:00<00:00, 1959.59frames/s]
100%|██████████| 340/340 [00:00<00:00, 2042.83frames/s]
100%|██████████| 400/400 [00:00<00:00, 2023.55frames/s]
100%|██████████| 429/429 [00:00<00:00, 2568.12frames/s]
100%|██████████| 481/481 [00:00<00:00, 2131.96frames/s]
100%|██████████| 415/415 [00:00<00:00, 1769.56frames/s]
Audios totales     : 80
Procesados OK      : 80
Con error          : 0
Salida (JSON por audio): /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripci
JSON global        : /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripcion/c

Entidades por tipo (acumuladas):
 - LOC: 77
 - PER: 32
 - MISC: 23
 - ORG: 23
```

6.Validación y resumen de resultados

En este apartado revisamos de forma sistemática la calidad del procesamiento por lotes. Para ello:

- Leemos el `resultados_global.json` generado en la fase anterior.
- Calculamos un resumen por archivo: longitud de la transcripción y número de entidades por tipo (PER, ORG, LOC, MISC) y total.
- Generamos un recuento global de las entidades más frecuentes por tipo, con su frecuencia y una confianza media aproximada.
- Guardamos dos ficheros de apoyo en `outputs/csv/`: `resumen_transcripciones.csv` (una fila por audio) y `entidades_top.csv` (frecuencias por entidad y tipo).

Estos resúmenes nos permiten detectar rápidamente audios con transcripción vacía o anómala, así como validar que el etiquetado de entidades es coherente con el contenido.

```
# Resumen de calidad y guardado de CSV auxiliares.
OUT_JSON_DIR = PATHS["out_json"]
OUT_CSV_DIR  = PATHS["out_csv"]
os.makedirs(OUT_CSV_DIR, exist_ok=True)

GLOBAL_JSON = os.path.join(OUT_JSON_DIR, "resultados_global.json")
```

```
# 1) Cargar resultados
with open(GLOBAL_JSON, "r", encoding="utf-8") as f:
    resultados = json.load(f)

# 2) Construir resumen por archivo
resumen_rows = []
vacios = 0
tipos_conocidos = ("PER", "ORG", "LOC", "MISC")

for r in resultados:
    fname = r.get("archivo_audio", "")
    texto = (r.get("transcripcion") or "").strip()
    ents  = r.get("entidades") or {}
    if texto == "":
        vacios += 1

    # Conteos por tipo
    counts = {t: len(ents.get(t, [])) for t in tipos_conocidos}
    total  = sum(counts.values())

    resumen_rows.append({
        "archivo_audio": fname,
        "longitud_transcripcion": len(texto),
        "n_PER": counts["PER"],
        "n_ORG": counts["ORG"],
        "n_LOC": counts["LOC"],
        "n_MISC": counts["MISC"],
        "n_total": total,
    })

# Guardar resumen por archivo
resumen_csv = os.path.join(OUT_CSV_DIR, "resumen_transcripciones.csv")
with open(resumen_csv, "w", encoding="utf-8", newline="") as f:
    writer = csv.DictWriter(
        f,
        fieldnames=[
            "archivo_audio", "longitud_transcripcion",
            "n_PER", "n_ORG", "n_LOC", "n_MISC", "n_total"
        ],
    )
    writer.writeheader()
    writer.writerows(resumen_rows)

# 3) Frecuencias de entidades por tipo y texto + confianza media
freq = defaultdict(lambda: {"count": 0, "score_sum": 0.0})
for r in resultados:
    ents = r.get("entidades") or {}
    for t, lista in ents.items():
        for e in lista:
            clave = (t, e.get("texto", ""))
            freq[clave]["count"] += 1
            freq[clave]["score_sum"] += float(e.get("score", 0.0))

# Preparar filas ordenadas por tipo y frecuencia
entidades_rows = []
for (t, texto), info in freq.items():
    avg_score = info["score_sum"] / max(info["count"], 1)
    entidades_rows.append({
        "tipo": t,
        "texto": texto,
        "frecuencia": info["count"],
```

```
        "confianza_media": round(avg_score, 3),
    })

# Orden: tipo asc, frecuencia desc, texto asc
entidades_rows.sort(key=lambda x: (x["tipo"], -x["frecuencia"], x["texto"]))

# Guardar CSV de entidades top
entidades_csv = os.path.join(OUT_CSV_DIR, "entidades_top.csv")
with open(entidades_csv, "w", encoding="utf-8", newline="") as f:
    writer = csv.DictWriter(
        f,
        fieldnames=["tipo", "texto", "frecuencia", "confianza_media"],
    )
    writer.writeheader()
    writer.writerows(entidades_rows)

# 4) Informe corto en consola
total = len(resumen_rows)
print("Audios procesados  :", total)
print("Transcripciones vacías:", vacios)
print("CSV (resumen por archivo):", resumen_csv)
print("CSV (entidades top):      ", entidades_csv)

# Mostrar un avance pequeño (primeras 5 filas del resumen)
print("\nPrimeras 5 filas del resumen:")
for row in resumen_rows[:5]:
    print(row)
```

```
Audios procesados  : 80
Transcripciones vacías: 0
CSV (resumen por archivo): /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripc
CSV (entidades top):       /content/drive/MyDrive/MASTER BIG DATA/md2_2025/ej1_transcripc

Primeras 5 filas del resumen:
{'archivo_audio': 'frase_01.wav', 'longitud_transcripcion': 61, 'n_PER': 1, 'n_ORG': 0, '
{'archivo_audio': 'frase_02.wav', 'longitud_transcripcion': 53, 'n_PER': 1, 'n_ORG': 0, '
{'archivo_audio': 'frase_03.wav', 'longitud_transcripcion': 56, 'n_PER': 0, 'n_ORG': 1, '
{'archivo_audio': 'frase_04.wav', 'longitud_transcripcion': 54, 'n_PER': 0, 'n_ORG': 0, '
{'archivo_audio': 'frase_05.wav', 'longitud_transcripcion': 63, 'n_PER': 0, 'n_ORG': 0, '
```

7.Conclusiones

Hemos transcrito todos los audios con Whisper y aplicado NER en español con `MMG/xlm-roberta-large-ner-spanish`. Generamos un **JSON por audio** y un **JSON global** con el nombre del archivo, la transcripción y las entidades agrupadas por tipo, tal y como exige el enunciado. La validación final confirma que no hubo errores y que las transcripciones no quedaron vacías. Además, guardamos dos **CSV de apoyo**:

- `resumen_transcripciones.csv`: recoge, para cada audio, la longitud de la transcripción y el número de entidades detectadas por tipo (PER, ORG, LOC, MISC) y el total. Nos sirve para revisar rápidamente posibles casos anómalos.
- `entidades_top.csv`: lista cada entidad detectada (por texto y tipo), su **frecuencia** y la **confianza media**. Nos ayuda a entender qué entidades aparecen más y con qué seguridad.

Como mejoras futuras, podríamos evaluar tamaños superiores de Whisper para ganar precisión, revisar entidades etiquetadas como MISC y normalizar nombres propios para análisis posteriores.

8.Despliegue con Gradio

En este apartado montamos una **mini app web** con **Gradio** para probar el flujo de **transcripción** (Whisper) y **reconocimiento de entidades** (XLM-R).
La interfaz permite subir un audio (o grabarlo con el micro), lanzar la transcripción y visualizar una tabla con las entidades detectadas.

Objetivos:

- Facilitar la **demostración** del modelo a no técnicos.
- Generar **evidencias** (JSON por ejecución) sin reentrenar ni alterar el pipeline.

Metodología y banderas:

- Respetamos `USE_DRIVE` para rutas.
- Añadimos `SAVE_RESULTS` para **guardar** o no las salidas de la app (por defecto, activado).
- Si los modelos (`whisper_model`, `ner_pipe`) están ya en memoria, **se reutilizan**; en caso contrario, se **cargan** con la misma configuración del cuaderno.

Salidas (si `SAVE_RESULTS=True`):

- Un JSON por ejecución en `outputs/json/gradio_runs/` con transcripción, entidades y metadatos.
- Guardamos las capturas de pantalla en `outputs/figures/`

> Preparamos el despliegue con Gradio de forma autónoma, de modo que puede ejecutarse sin necesidad de rehacer todos los pasos previos pero manteniendo la coherencia con los modelos utilizados en el ejercicio. Está optimizado para su ejecución en Google Colab. De esta forma se genera automáticamente un enlace público temporal (`.gradio.live`) accesible desde cualquier navegador. En un entorno local (Jupyter Notebook), bastaría con instalar Gradio y ejecutar las mismas celdas para abrir la interfaz en `http://127.0.0.1:7860`.

```
# Flag para controlar si lanzamos realmente el despliegue o solo cargamos resultados ya genera
RUN_DEPLOY = True
```

```
# === Configuración para el despliegue con Gradio ===
from pathlib import Path
import json, os
from datetime import datetime

# Directorios
OUT_JSON_DIR = Path(PATHS["out_json"])
GRADIO_RUNS  = OUT_JSON_DIR / "gradio_runs"
GRADIO_RUNS.mkdir(parents=True, exist_ok=True)

# Seleccionamos una muestra real para el modo entregable
candidatos = sorted(GRADIO_RUNS.glob("*.json"))
if len(candidatos) == 0:
    ejemplo = {
        "archivo_audio": "ejemplo.wav",
        "transcripcion": "Buenos días, esto es un ejemplo de transcripción.",
        "entidades": {"LOC": [{"texto": "Madrid", "score": 0.98}]},
        "modelo_whisper": "base",
        "modelo_ner": "MMG/xlm-roberta-large-ner-spanish",
        "timestamp": datetime.utcnow().isoformat()+"Z",
    }
    sample_path = GRADIO_RUNS / "sample_gradio.json"
```

```
    with open(sample_path, "w", encoding="utf-8") as f:
        json.dump(ejemplo, f, ensure_ascii=False, indent=2)
    print("ℹ️ No se encontraron JSON previos. Creado ejemplo sintético en:", sample_path)
else:
    src = candidatos[0]
    sample_path = GRADIO_RUNS / "sample_gradio.json"
    with open(src, "r", encoding="utf-8") as f:
        data = json.load(f)
    ejemplo = {
        "archivo_audio": data.get("archivo_audio", os.path.basename(data.get("ruta_audio","mue
        "transcripcion": data.get("transcripcion", data.get("texto","")),
        "entidades": data.get("entidades", {}),
        "modelo_whisper": "base",
        "modelo_ner": "MMG/xlm-roberta-large-ner-spanish",
        "timestamp": datetime.utcnow().isoformat()+"Z",
    }
    with open(sample_path, "w", encoding="utf-8") as f:
        json.dump(ejemplo, f, ensure_ascii=False, indent=2)
    print("✅ Muestra real preparada para el modo entregable:", sample_path)
```

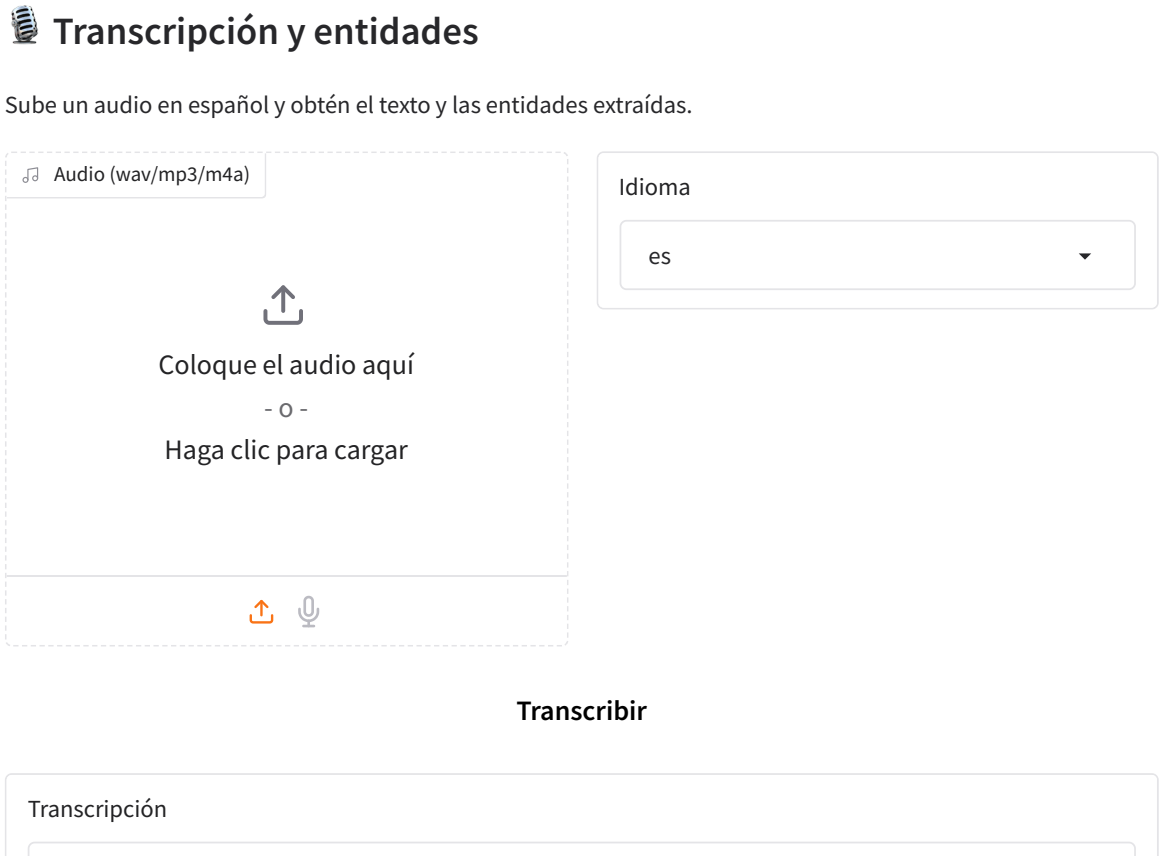
```
✅ Muestra real preparada para el modo entregable: /content/drive/MyDrive/MASTER BIG DATA
```

```
# === 8.B) Despliegue con Gradio (limpio, integrado con PATHS/GRADIO_RUNS) ===
# Usa RUN_DEPLOY:
#   - True  → instala condicionalmente, carga modelos y lanza Gradio (en Colab: share=True)
#   - False → modo entregable: muestra la muestra real 'sample_gradio.json' (sin instalar nada

# Comprobación suave por si RUN_DEPLOY no existe:
if "RUN_DEPLOY" not in globals():
    RUN_DEPLOY = False

if RUN_DEPLOY:
    # --- Instalación condicional de dependencias pesadas ---
    try:
        import whisper
    except ImportError:
        %pip -q install -U openai-whisper
        import whisper

    try:
        import gradio as gr
    except ImportError:
        %pip -q install -U gradio
        import gradio as gr

    try:
        import transformers
        from transformers import AutoTokenizer, AutoModelForTokenClassification, pipeline
    except ImportError:
        %pip -q install -U transformers accelerate
        from transformers import AutoTokenizer, AutoModelForTokenClassification, pipeline

    import shutil
    if not shutil.which("ffmpeg"):
        !apt -q install -y ffmpeg

    # --- Imports específicos para el despliegue ---
    import torch
    import pandas as pd

    # --- Config coherente con pipeline (si existen, reusa; si no, default) ---
    device       = "cuda" if torch.cuda.is_available() else "cpu"
```

```
WHISPER_SIZE = globals().get("WHISPER_SIZE", "base")
NER_MODEL    = globals().get("NER_MODEL", "MMG/xlm-roberta-large-ner-spanish")

# --- Carga perezosa de modelos (si ya estaban en memoria, se reutilizan) ---
if "whisper_model" not in globals():
    whisper_model = whisper.load_model(WHISPER_SIZE, device=device)

if "ner_pipe" not in globals():
    tokenizer = AutoTokenizer.from_pretrained(NER_MODEL)
    ner_model = AutoModelForTokenClassification.from_pretrained(NER_MODEL)
    ner_pipe  = pipeline(
        "token-classification",
        model=ner_model,
        tokenizer=tokenizer,
        aggregation_strategy="simple",
        device=0 if device == "cuda" else -1
    )

# --- Auxiliar: convertir entidades a DataFrame legible ---
def _ents_to_df(ents):
    if not ents:
        return pd.DataFrame(columns=["entity_group","word","score","start","end"])
    rows = []
    for e in ents:
        rows.append({
            "entity_group": e.get("entity_group",""),
            "word": e.get("word",""),
            "score": float(e.get("score",0.0)),
            "start": e.get("start", None),
            "end": e.get("end", None),
        })
    return pd.DataFrame(rows)[["entity_group","word","score","start","end"]]

# --- Lógica de la app: transcribe y hace NER; guarda JSON por ejecución ---
def transcribir_y_ner(audio_path: str, language: str = "es"):
    if not audio_path or not os.path.exists(audio_path):
        return "⚠️ No se recibió audio.", pd.DataFrame(columns=["entity_group","word","sco

    # 1) Transcripción
    result = whisper_model.transcribe(audio_path, language=language, verbose=False)
    texto  = (result.get("text") or "").strip()

    # 2) NER
    ents   = ner_pipe(texto) if texto else []
    df     = _ents_to_df(ents)

    # 3) Guardado JSON de esta ejecución en outputs/json/gradio_runs
    payload = {
        "timestamp_utc": datetime.utcnow().isoformat()+"Z",
        "audio_file": os.path.basename(audio_path),
        "language": language,
        "whisper_size": WHISPER_SIZE,
        "ner_model": NER_MODEL,
        "transcripcion": texto,
        "entidades": df.to_dict(orient="records"),
    }
    run_name = f"gradio_run_{datetime.utcnow().strftime('%Y%m%dT%H%M%SZ')}.json"
    with open((Path(PATHS["out_json"]) / "gradio_runs" / run_name), "w", encoding="utf-8")
        json.dump(payload, f, ensure_ascii=False, indent=2)

    return texto, df

# --- Interfaz Gradio ---
```

```
    with gr.Blocks(title="Transcripción + NER (Whisper + XLM-R)") as demo:
        gr.Markdown("## 🎙️ Transcripción y entidades\nSube un audio en español y obtén el text
        with gr.Row():
            audio_in = gr.Audio(sources=["upload","microphone"], type="filepath", label="Audio
            lang_in  = gr.Dropdown(choices=["es","en","fr","de","it","pt"], value="es", label='
        btn     = gr.Button("Transcribir")
        txt_out = gr.Textbox(label="Transcripción", lines=8)
        ents_df = gr.Dataframe(label="Entidades (NER)", interactive=False)

        btn.click(transcribir_y_ner, inputs=[audio_in, lang_in], outputs=[txt_out, ents_df])

    # En Colab, share=True para URL pública temporal (.gradio.live)
    demo.launch(share=True)

else:
    # === MODO ENTREGABLE: muestra sample_gradio.json creado en 8.A (sin instalar nada) ===
    sample_path = Path(PATHS["out_json"]) / "gradio_runs" / "sample_gradio.json"
    if sample_path.exists():
        with open(sample_path, "r", encoding="utf-8") as f:
            data = json.load(f)
        print("📄 Ejemplo real (modo entregable):")
        print(json.dumps(data, ensure_ascii=False, indent=2))
    else:
        print("⚠️ No se encontró sample_gradio.json en", sample_path.parent)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://124c5cf105481c87e6.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `grad
```



Empieza a programar o a crear código con IA.