

✓ Proyecto Final APO III

INTEGRANTES

- Juan Esteban Eraso
- Damy Villegas
- Cristian Molina
- Carlos Sanchez

1. CONTEXTO DEL PROBLEMA

La movilidad es fundamental para la realización de actividades cotidianas y el bienestar general. Diversos factores, como la reducción de la fuerza, la flexibilidad y la coordinación motora, pueden limitar la capacidad de las personas para moverse libremente.

Muchas condiciones de salud se manifiestan a través de **alteraciones en la forma de caminar, la postura o la velocidad de movimiento**. Detectar estos cambios a tiempo es fundamental para facilitar una intervención médica o terapéutica oportuna.

Tradicionalmente, la evaluación de la movilidad se realiza mediante observación directa por parte de profesionales de la salud o con equipos especializados que resultan costosos y difíciles de implementar fuera de entornos clínicos. En este contexto, la **inteligencia artificial y la visión por computador** ofrecen una alternativa tecnológica accesible y no invasiva que permite **analizar el movimiento humano usando una cámara convencional**.

El proyecto propone desarrollar un **sistema de análisis automático del movimiento** orientado a **detectar signos de movilidad reducida o alteraciones motoras**. El sistema procesará videos mediante **MediaPipe Pose** para extraer coordenadas articulares y utilizará modelos de **aprendizaje supervisado** que analicen la amplitud, simetría y velocidad de los movimientos. De esta manera, se busca apoyar a profesionales y cuidadores en la identificación

temprana de posibles limitaciones motoras, sin reemplazar la evaluación médica, pero sirviendo como una herramienta complementaria de monitoreo y apoyo preventivo.

2. PREGUNTA DE INTERES

¿Es posible detectar signos de movilidad reducida o alteraciones motoras mediante el análisis automático de video, el seguimiento de articulaciones con MediaPipe y modelos de aprendizaje supervisado?

3. TIPO DE PROBLEMA

El proyecto aborda un **problema de clasificación supervisada**, en el que el modelo de inteligencia artificial debe reconocer diferentes actividades (caminar, girar, sentarse, levantarse) y determinar si el patrón de movimiento observado corresponde a una movilidad normal o reducida.

4. METODOLOGIA CRISP-DM ADAPTADA AL PROYECTO

El desarrollo del sistema sigue la metodología **CRISP-DM (Cross-Industry Standard Process for Data Mining)**, complementada con principios del enfoque **ASUM-DM**, garantizando una planificación estructurada y revisiones periódicas del progreso.

4.1. ENTENDIMIENTO DEL NEGOCIO

- Comprender el impacto que tiene la movilidad reducida en la autonomía y calidad de vida de las personas.
- Definir objetivos concretos: identificar patrones de rigidez, lentitud o falta de simetría.
- Establecer criterios de éxito: lograr un modelo con al menos 85 % de precisión en la clasificación.
- Identificar herramientas: Python, MediaPipe, scikit-learn, OpenCV y Streamlit.

4.2. COMPRESION DE LOS DATOS

- Captura de videos con personas realizando actividades básicas.
- Extracción de coordenadas (x, y, z) mediante MediaPipe Pose.
- Identificación de variables relevantes: ángulos, inclinaciones y velocidades articulares.

- Análisis preliminar de consistencia y comportamiento de los datos.

4.3. PREPARACION DE LOS DATOS

- Limpieza y normalización de coordenadas.
- Generación de características derivadas (velocidad, amplitud, simetría, inclinación).
- Etiquetado de las muestras según nivel de movilidad observado.

4.4. MODELADO

- Entrenamiento de modelos supervisados como Random Forest, SVM y XGBoost.
- Validación cruzada para ajustar hiperparámetros.
- Selección del modelo con mejor balance entre precisión y estabilidad.

4.5. EVALUACION

- Evaluación del modelo mediante métricas de desempeño (accuracy, precision, recall, F1-score).
- Verificación del cumplimiento de los objetivos definidos.
- Comparación entre modelos para determinar el más adecuado.

4.6. DESPLIEGUE

- Implementación de una interfaz visual que muestre el video, el esqueleto detectado y los indicadores biomecánicos.
- Pruebas de funcionamiento en tiempo real.

4.7. RETROALIMENTACION (ASUM-DM)

- Revisión continua de resultados y ajustes metodológicos.
- Registro de aprendizajes y mejoras en cada iteración del proyecto.

5. METRICAS PARA MEDIR EL PROGRESO

5.1. METRICAS DEL MODELO

5.2. METRICAS BIOMECANICAS

6. DATOS RECOLECTADOS (HACER RECOLECCION)

- ## EDA

1. Estrategia de obtención de nuevos datos:

5 of 51

...diferentes convenciones de nomenclatura y entornos.

2. Preparación de los datos:

Procesar los nuevos videos con MediaPipe Pose, limpiar las coordenadas obtenidas, normalizarlas y generar nuevas características (velocidad, amplitud, simetría, inclinación, etc.).

3. Entrenamiento de modelos:

Entrenar y comparar varios modelos supervisados (Random Forest, SVM, XGBoost), evaluando su precisión y capacidad para identificar movilidad reducida.

✓ Instalación e imports

```
!pip -q install opencv-python pandas numpy matplotlib tqdm

import os, re, math, json
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm

pd.set_option("display.max_colwidth", 200)
```

✓ Drive y rutas

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

# Ruta de tu carpeta
BASE_DIR = "/content/drive/MyDrive/ProyectoIA/Videos"
assert os.path.isdir(BASE_DIR), f"Ruta no existe: {BASE_DIR}"

# Vista rápida para confirmar que ve los .mn4
```

```
.. ----- report path separator: use os.path.sep .
!ls -lh "$BASE_DIR" | head -n 25
```

Mounted at /content/drive

total 40M

```
-rw----- 1 root root 2.0K Oct 17 00:29 catalogo_videos.csv
drwx----- 2 root root 4.0K Oct 17 01:10 eda_outputs
drwx----- 2 root root 4.0K Oct 17 01:02 labels
-rw----- 1 root root 2.0M Oct 17 00:15 Video 10.mp4
-rw----- 1 root root 2.8M Oct 17 00:15 Video 11.mp4
-rw----- 1 root root 3.4M Oct 17 00:15 Video 12.mp4
-rw----- 1 root root 2.0M Oct 17 00:15 Video 13.mp4
-rw----- 1 root root 2.3M Oct 17 00:15 Video 14.mp4
-rw----- 1 root root 2.2M Oct 17 00:15 Video 15.mp4
-rw----- 1 root root 2.6M Oct 17 00:15 Video 16.mp4
-rw----- 1 root root 2.3M Oct 17 00:15 Video 17.mp4
-rw----- 1 root root 2.0M Oct 17 00:15 Video 18.mp4
-rw----- 1 root root 2.1M Oct 17 00:15 Video 1.mp4
-rw----- 1 root root 2.4M Oct 17 00:15 Video 2.mp4
-rw----- 1 root root 1.7M Oct 17 00:15 Video 3.mp4
-rw----- 1 root root 2.3M Oct 17 00:15 Video 4.mp4
-rw----- 1 root root 2.4M Oct 17 00:15 Video 5.mp4
-rw----- 1 root root 2.0M Oct 17 00:15 Video 6.mp4
-rw----- 1 root root 2.4M Oct 17 00:15 Video 7.mp4
-rw----- 1 root root 1.9M Oct 17 00:15 Video 8.mp4
-rw----- 1 root root 2.1M Oct 17 00:15 Video 9.mp4
drwx----- 2 root root 4.0K Oct 27 00:06 videos
```

✓ Funciones para leer metadatos y parsear nombre

```
# Extrae un entero del nombre "Video 1.mp4" -> video_id = 1
pat_simple = re.compile(r'video\s*(?P<id>\d+)', re.IGNORECASE)

def parse_filename(fname: str):
    name = os.path.splitext(os.path.basename(fname))[0]
    m = pat_simple.search(name)
    video_id = int(m.group("id")) if m else None
```

```
    return {
        "video_id": video_id,
        "subject": None,
        "actividad": None,
        "vista": None,
        "repeticion": None
    }

def read_video_meta(path: str):
    cap = cv2.VideoCapture(path)
    if not cap.isOpened():
        return None
    fps = float(cap.get(cv2.CAP_PROP_FPS) or 0)
    nfr = int(cap.get(cv2.CAP_PROP_FRAME_COUNT) or 0)
    w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH) or 0)
    h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT) or 0)
    cap.release()

    dur_s = (nfr / fps) if fps > 0 else None
    size_mb = os.path.getsize(path) / 1024 / 1024
    return {
        "fps": round(fps, 2) if fps else None,
        "frames": nfr if nfr > 0 else None,
        "dur_s": round(dur_s, 2) if dur_s else None,
        "width": w, "height": h,
        "size_mb": round(size_mb, 2)
    }
```

✓ Construir catálogo

```
rows = []
for root, _, files in os.walk(BASE_DIR):
    for f in files:
        if f.lower().endswith(".mp4"):
            n = os.path.join(root, f)
```

```

    meta = read_video_meta(p)
    if meta is None:
        continue
    info = parse_filename(f)
    rows.append({
        "file": f,
        "path": p,
        **info,
        **meta,
        "rel_dir": os.path.relpath(root, BASE_DIR)
    })

df = pd.DataFrame(rows)

# Orden natural por video_id y luego por nombre
if "video_id" in df.columns:
    df = df.sort_values(by=["video_id", "file"], na_position="last").reset_index(drop=True)
else:
    df = df.sort_values(by=["file"]).reset_index(drop=True)

print("Total de videos:", len(df))
df.head(10)
```

Total de videos: 18

	file	path	video_id	subject	actividad	vista	repeticion	fps	frames	dur_s	width	height	si
0	Video 1.mp4	/content/drive/MyDrive/ProyectorA/Videos/Video 1.mp4	1	None	None	None	None	29.93	409	13.67	480	848	
1	Video 2.mp4	/content/drive/MyDrive/ProyectorA/Videos/Video 2.mp4	2	None	None	None	None	29.94	479	16.00	480	848	

	2.mp4	Videos/ Video 2.mp4											
2	Video 3.mp4	/content/ drive/ MyDrive/ ProyectorA/ Videos/ Video 3.mp4	3	None	None	None	None	29.91	343	11.47	480	848	
3	Video 4.mp4	/content/ drive/ MyDrive/ ProyectorA/ Videos/ Video 4.mp4	4	None	None	None	None	29.94	467	15.60	480	848	
4	Video 5.mp4	/content/ drive/ MyDrive/ ProyectorA/ Videos/ Video 5.mp4	5	None	None	None	None	29.94	468	15.63	480	848	
5	Video 6.mp4	/content/ drive/ MyDrive/ ProyectorA/ Videos/ Video 6.mp4	6	None	None	None	None	29.93	382	12.76	480	848	
6	Video 7.mp4	/content/ drive/ MyDrive/ ProyectorA/ Videos/ Video 7.mp4	7	None	None	None	None	29.94	488	16.30	480	848	

		7.mp4											
		/content/ drive/ MyDrive/ ProyectorA/ Videos/ Video 8.mp4											
7	Video 8.mp4		8	None	None	None	None	29.92	368	12.30	480	848	
		/content/ drive/ MyDrive/ ProyectorA/ Videos/ Video 9.mp4											
8	Video 9.mp4		9	None	None	None	None	29.93	423	14.13	480	848	
		/content/ drive/ MyDrive/ ProyectorA/ Videos/ Video 10.mp4											
9	Video 10.mp4		10	None	None	None	None	29.92	391	13.07	480	848	

✓ Guardar CSV y chequeos

```

out_path = os.path.join(BASE_DIR, "catalogo_videos.csv")
df.to_csv(out_path, index=False)
print("Catálogo guardado en:", out_path)

# Resumen rápido
display(df[["video_id", "file", "fps", "dur_s", "width", "height", "size_mb"]].head(15))

# Distribución de duración
plt.figure(figsize=(6,4))

```

```
df["dur_s"].dropna().plot(kind="hist", bins=15, title="Distribución de duración (s)")
plt.xlabel("segundos"); plt.ylabel("conteo")
plt.show()

# Distribución de FPS
plt.figure(figsize=(6,4))
df["fps"].dropna().plot(kind="hist", bins=15, title="Distribución de FPS")
plt.xlabel("fps"); plt.ylabel("conteo")
plt.show()

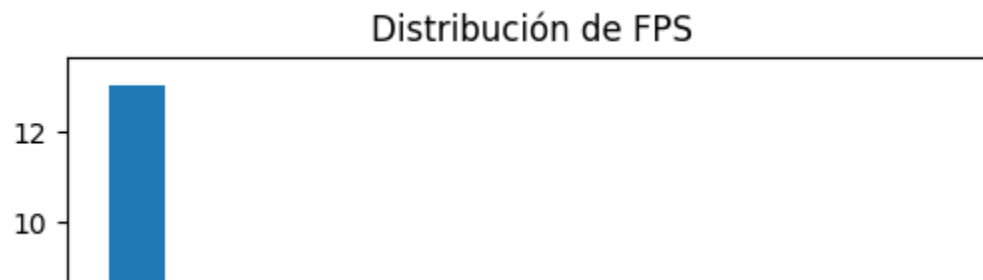
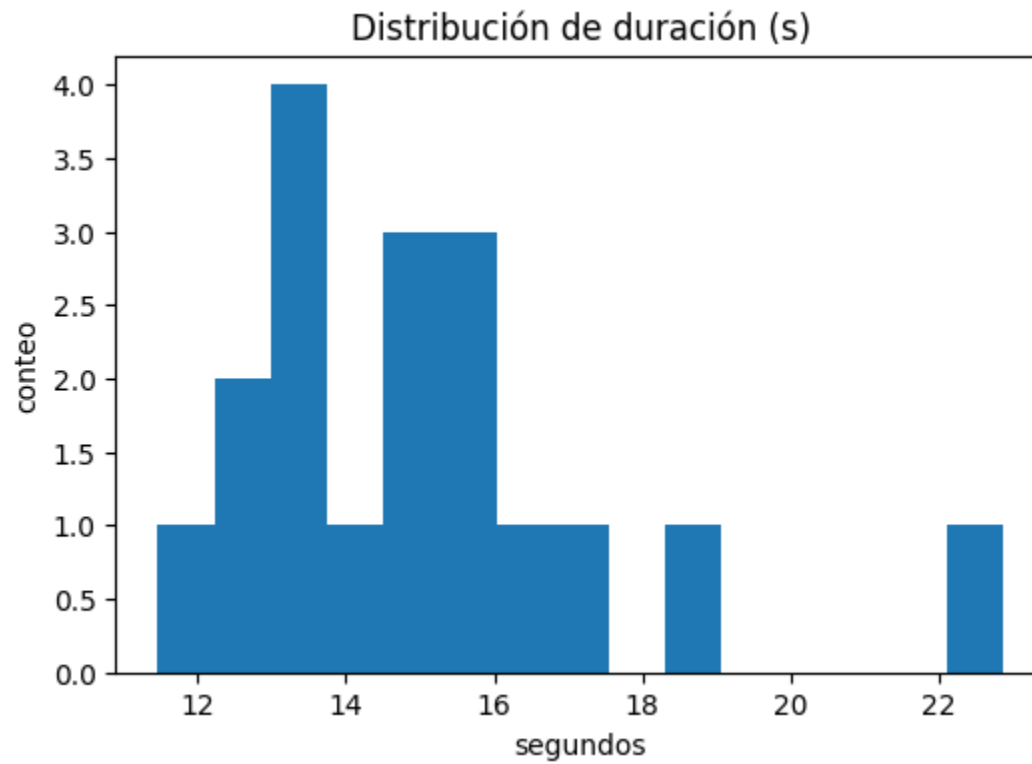
# Resoluciones más comunes
res = (df["width"].astype(str) + "x" + df["height"].astype(str)).value_counts()
print("Resoluciones más comunes:\n", res.head(10))

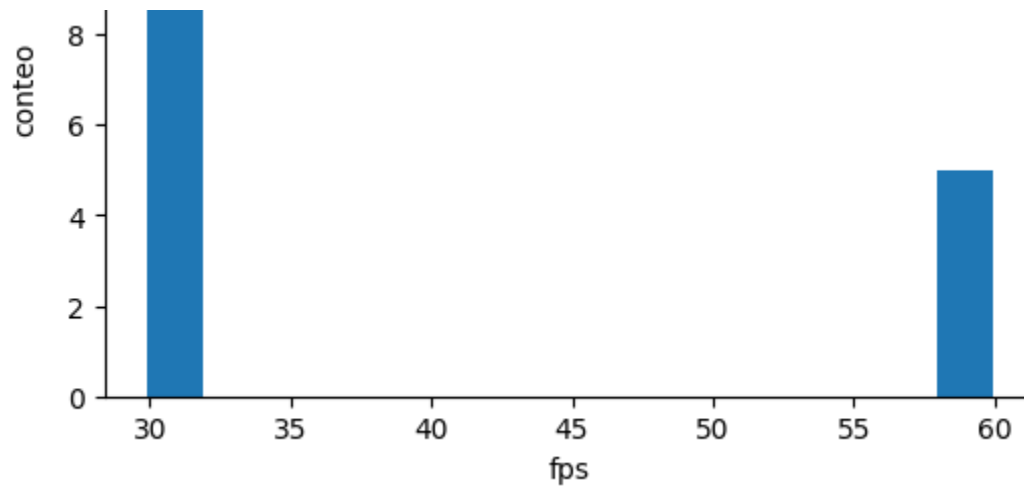
# Chequeo de unicidad de IDs (por si hubiera duplicados)
if "video_id" in df.columns:
    dup = df["video_id"].value_counts()
    print("\nRecuento por video_id (para detectar duplicados):\n", dup.head(20))
```

Catálogo guardado en: /content/drive/MyDrive/ProyectoIA/Videos/catalogo_videos.csv

	video_id	file	fps	dur_s	width	height	size_mb
0	1	Video 1.mp4	29.93	13.67	480	848	2.06
1	2	Video 2.mp4	29.94	16.00	480	848	2.31
2	3	Video 3.mp4	29.91	11.47	480	848	1.69
3	4	Video 4.mp4	29.94	15.60	480	848	2.25
4	5	Video 5.mp4	29.94	15.63	480	848	2.32
5	6	Video 6.mp4	29.93	12.76	480	848	1.91
6	7	Video 7.mp4	29.94	16.30	480	848	2.38
7	8	Video 8.mp4	29.92	12.30	480	848	1.81
8	9	Video 9.mp4	29.93	14.13	480	848	2.06

9	10	Video 10.mp4	29.92	13.07	480	848	1.90
10	11	Video 11.mp4	29.90	18.43	480	848	2.74
11	12	Video 12.mp4	29.96	22.86	480	848	3.36
12	13	Video 13.mp4	29.98	13.11	480	848	1.90
13	14	Video 14.mp4	59.94	15.10	464	832	2.23
14	15	Video 15.mp4	59.94	14.95	464	832	2.15





Resoluciones más comunes:

480x848 13

464x832 5

Name: count, dtype: int64

Recuento por video_id (para detectar duplicados):

video_id

1 1

2 1

3 1

4 1

5 1

6 1

7 1

8 1

9 1

10 1

11 1

12 1

13 1

14 1

15 1

16 1

17 1

18 1

Name: count, dtype: int64

✓ Generar plantillas de etiquetado

```
import os
import pandas as pd
import numpy as np

# === rutas (deben coincidir con lo que ya usaste) ===
BASE_DIR = "/content/drive/MyDrive/ProyectoIA/Videos"
CATALOGO = os.path.join(BASE_DIR, "catalogo_videos.csv")
LABELS_DIR = os.path.join(BASE_DIR, "labels")
os.makedirs(LABELS_DIR, exist_ok=True)

# === etiquetas permitidas para esta entrega ===
ALLOWED = ["sentado", "de_pie", "caminando", "transicion"]

# === secuencia base (proporciones de la duración total; suman 1.0) ===
# Nota: 'transicion' se usa para levantarse, giros y sentarse.
seq = [
    ("sentado", 0.12, "inicio sentado"),
    ("transicion", 0.05, "levantarse"),
    ("de_pie", 0.03, "breve de_pie antes de caminar"),
    ("caminando", 0.30, "caminar hacia la cámara"),
    ("transicion", 0.05, "giro 1"),
    ("caminando", 0.30, "caminar de regreso a la silla"),
    ("transicion", 0.05, "giro 2"),
    ("transicion", 0.05, "sentarse"),
    ("sentado", 0.05, "fin sentado"),
]

# mínimos y redondeo
MIN_SEG = 0.30 # duración mínima por segmento en segundos
ROUND = 2 # decimales para exportar

FORCE_OVERWRITE = True
```

```
df_cat = pd.read_csv(CATALOGO)

def plantilla_path(file_name):
    base = os.path.splitext(file_name)[0]
    return os.path.join(LABELS_DIR, f"{base}_labels.csv")

generadas, omitidas = 0, 0

for _, r in df_cat.iterrows():
    file = r["file"]
    dur = float(r["dur_s"]) if pd.notna(r["dur_s"]) else None
    if not dur or dur <= 0:
        print(f"[AVISO] Duración inválida en {file}, se omite.")
        continue

    out_csv = plantilla_path(file)
    if (not FORCE_OVERWRITE) and os.path.exists(out_csv):
        omitidas += 1
        continue

    # 1) Proporciones → segundos (asegurando mínimos)
    props = np.array([p for _, p, _ in seq], dtype=float)
    props = props / props.sum()          # normaliza
    secs = props * dur

    # aplica mínimos y reescala para que sumen 'dur'
    secs = np.maximum(secs, MIN_SEG)
    secs = secs * (dur / secs.sum())

    # 2) construye intervalos acumulando tiempos
    rows = []
    start = 0.0
    for (label, _, note), seg_dur in zip(seq, secs):
        end = start + float(seg_dur)
        rows.append({
            "start_s": round(start, ROUND),
            "end_s": round(min(end, dur), ROUND), # cap por seguridad
            "label": label,
```

```

        "note":    note,    # columna opcional para tu referencia
    })
    start = end
    # fuerza fin EXACTO al total de video
    rows[-1]["end_s"] = round(dur, ROUND)

    tpl = pd.DataFrame(rows, columns=["start_s", "end_s", "label", "note"])
    tpl.to_csv(out_csv, index=False)
    generadas += 1

print(f"Plantillas generadas: {generadas} | omitidas (existentes y FORCE_OVERWRITE=False): {omitidas}")
print("Carpeta de etiquetas:", LABELS_DIR)
!ls -lh "$LABELS_DIR" | head -n 25

```

```

Plantillas generadas: 18 | omitidas (existentes y FORCE_OVERWRITE=False): 0
Carpeta de etiquetas: /content/drive/MyDrive/ProyectoIA/Videos/labels
total 9.0K
-rw----- 1 root root 354 Nov  2 14:40 Video 10_labels.csv
-rw----- 1 root root 356 Nov  2 14:40 Video 11_labels.csv
-rw----- 1 root root 358 Nov  2 14:40 Video 12_labels.csv
-rw----- 1 root root 352 Nov  2 14:40 Video 13_labels.csv
-rw----- 1 root root 351 Nov  2 14:40 Video 14_labels.csv
-rw----- 1 root root 352 Nov  2 14:40 Video 15_labels.csv
-rw----- 1 root root 354 Nov  2 14:40 Video 16_labels.csv
-rw----- 1 root root 350 Nov  2 14:40 Video 17_labels.csv
-rw----- 1 root root 354 Nov  2 14:40 Video 18_labels.csv
-rw----- 1 root root 352 Nov  2 14:40 Video 1_labels.csv
-rw----- 1 root root 341 Nov  2 14:40 Video 2_labels.csv
-rw----- 1 root root 350 Nov  2 14:40 Video 3_labels.csv
-rw----- 1 root root 351 Nov  2 14:40 Video 4_labels.csv
-rw----- 1 root root 352 Nov  2 14:40 Video 5_labels.csv
-rw----- 1 root root 354 Nov  2 14:40 Video 6_labels.csv
-rw----- 1 root root 353 Nov  2 14:40 Video 7_labels.csv
-rw----- 1 root root 353 Nov  2 14:40 Video 8_labels.csv
-rw----- 1 root root 350 Nov  2 14:40 Video 9_labels.csv

```

✓ Validador de etiquetas

```
BASE_DIR = "/content/drive/MyDrive/ProyectoIA/Videos"
CATALOGO = os.path.join(BASE_DIR, "catalogo_videos.csv")
LABELS_DIR = os.path.join(BASE_DIR, "labels")

ALLOWED = ["sentado", "de_pie", "caminando", "transicion"]
TOL = 0.20 # tolerancia en segundos para suma de intervalos vs duración de video

df_cat = pd.read_csv(CATALOGO)

def plantilla_path(file_name):
    base = os.path.splitext(file_name)[0]
    return os.path.join(LABELS_DIR, f"{base}_labels.csv")

def load_labels_for(file_name):
    p = plantilla_path(file_name)
    if not os.path.exists(p):
        return None
    d = pd.read_csv(p)
    # nos aseguramos de que existan columnas mínimas
    for c in ["start_s", "end_s", "label"]:
        assert c in d.columns, f"Falta columna {c} en {p}"
    return d

def check_intervals(df_lab):
    ok, msgs = True, []
    d = df_lab.sort_values("start_s").reset_index(drop=True)
    # valida etiquetas
    invalid = sorted(set(d["label"]) - set(ALLOWED))
    if invalid:
        ok = False
        msgs.append(f"Etiquetas inválidas: {invalid}")
    # valida límites y solapes
    for i in range(len(d)):
        if d.loc[i, "end_s"] <= d.loc[i, "start_s"]:
            ok = False
            msgs.append(f"Intervalo con end<=start en fila {i}")
```

```
        if i>0 and d.loc[i,"start_s"] < d.loc[i-1,"end_s"]:  
            ok = False  
            msgs.append(f"Solape entre filas {i-1} y {i}")  
    return ok, msgs, d  
  
resumen, errores = [], []  
for _, r in df_cat.iterrows():  
    fn = r["file"]  
    dur = float(r["dur_s"]) if pd.notna(r["dur_s"]) else None  
    labs = load_labels_for(fn)  
    if labs is None:  
        errores.append((fn, ["No existe archivo de etiquetas"]))  
        continue  
  
    ok, msgs, d = check_intervals(labs)  
    suma = (d["end_s"] - d["start_s"]).clip(lower=0).sum()  
    dur_ok = (dur is not None) and (abs(suma - dur) <= TOL)  
    if not dur_ok:  
        ok = False  
        msgs.append(f"Suma de intervalos={suma:.2f}s != duracion video={dur:.2f}s ( $\pm$ {TOL}s)")  
  
    if not ok:  
        errores.append((fn, msgs))  
    else:  
        resumen.append({  
            "file": fn,  
            "dur_video_s": round(dur,2),  
            "dur_etiquetas_s": round(suma,2),  
            "num_intervalos": len(d)  
        })  
  
res_df = pd.DataFrame(resumen).sort_values("file")  
print("Archivos correctos:", len(res_df), " | Con errores:", len(errores))  
display(res_df.head(30))  
  
if errores:  
    print("\n--- Detalles de errores ---")  
    for fn, msgs in errores:
```

```
print(f"* {fn}")
for m in msgs:
    print("  ", m)
```

Archivos correctos: 18 | Con errores: 0

	file	dur_video_s	dur_etiquetas_s	num_intervalos
0	Video 1.mp4	13.67	13.67	9
9	Video 10.mp4	13.07	13.07	9
10	Video 11.mp4	18.43	18.43	9
11	Video 12.mp4	22.86	22.86	9
12	Video 13.mp4	13.11	13.11	9
13	Video 14.mp4	15.10	15.10	9
14	Video 15.mp4	14.95	14.95	9
15	Video 16.mp4	17.17	17.17	9
16	Video 17.mp4	15.02	15.02	9
17	Video 18.mp4	13.43	13.43	9
1	Video 2.mp4	16.00	16.00	9
2	Video 3.mp4	11.47	11.47	9
3	Video 4.mp4	15.60	15.60	9
4	Video 5.mp4	15.63	15.63	9
5	Video 6.mp4	12.76	12.76	9
6	Video 7.mp4	16.30	16.30	9
7	Video 8.mp4	12.30	12.30	9
8	Video 9.mp4	14.13	14.13	9

✓ EDA con etiquetas

```
import matplotlib.pyplot as plt

BASE_DIR = "/content/drive/MyDrive/ProyectoIA/Videos"
CATALOGO = os.path.join(BASE_DIR, "catalogo_videos.csv")
LABELS_DIR = os.path.join(BASE_DIR, "labels")

df_cat = pd.read_csv(CATALOGO)

def plantilla_path(file_name):
    base = os.path.splitext(file_name)[0]
    return os.path.join(LABELS_DIR, f"{base}_labels.csv")

def load_labels_for(file_name):
    p = plantilla_path(file_name)
    if not os.path.exists(p):
        return None
    return pd.read_csv(p)

def stack_all_labels(df_catalog):
    rows = []
    for _, r in df_catalog.iterrows():
        labs = load_labels_for(r["file"])
        if labs is None:
            continue
        d = labs.copy()
        d["file"] = r["file"]
        d["video_id"] = r["video_id"]
        d["dur_seg"] = (d["end_s"] - d["start_s"]).clip(lower=0)
        rows.append(d)
    if not rows:
        return pd.DataFrame(columns=["file", "video_id", "start_s", "end_s", "label", "dur_seg"])
    return pd.concat(rows, ignore_index=True)
```

```

labs_all = stack_all_labels(df_cat)
if len(labs_all)==0:
    print("Aún no hay etiquetas completadas.")
else:
    # Cobertura por etiqueta (segundos totales)
    cobertura = labs_all.groupby("label")["dur_seg"].sum().sort_values(ascending=False)
    print("Cobertura (segundos totales) por etiqueta:")
    print(cobertura)

    # Conteo por etiqueta y por video
    cobertura_por_video = labs_all.groupby(["file", "label"])["dur_seg"].sum().reset_index()
    display(cobertura_por_video.head(30))

    # Histogramas de duración de segmentos por etiqueta
    for et in cobertura.index.tolist():
        d = labs_all[labs_all["label"]==et]["dur_seg"]
        if len(d)==0:
            continue
        plt.figure(figsize=(6,4))
        d.plot(kind="hist", bins=15, title=f"Duración de segmentos – {et}")
        plt.xlabel("segundos"); plt.ylabel("conteo")
        plt.show()

```

Cobertura (segundos totales) por etiqueta:

label

caminando 162.61

transicion 54.20

sentado 46.06

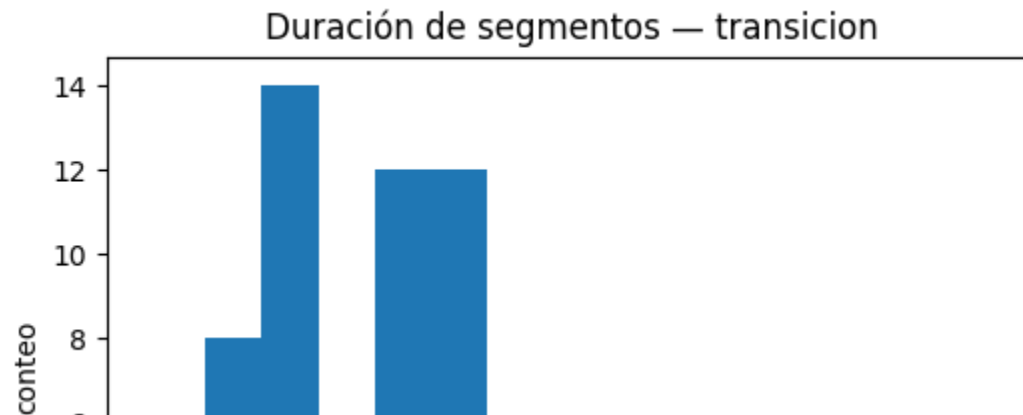
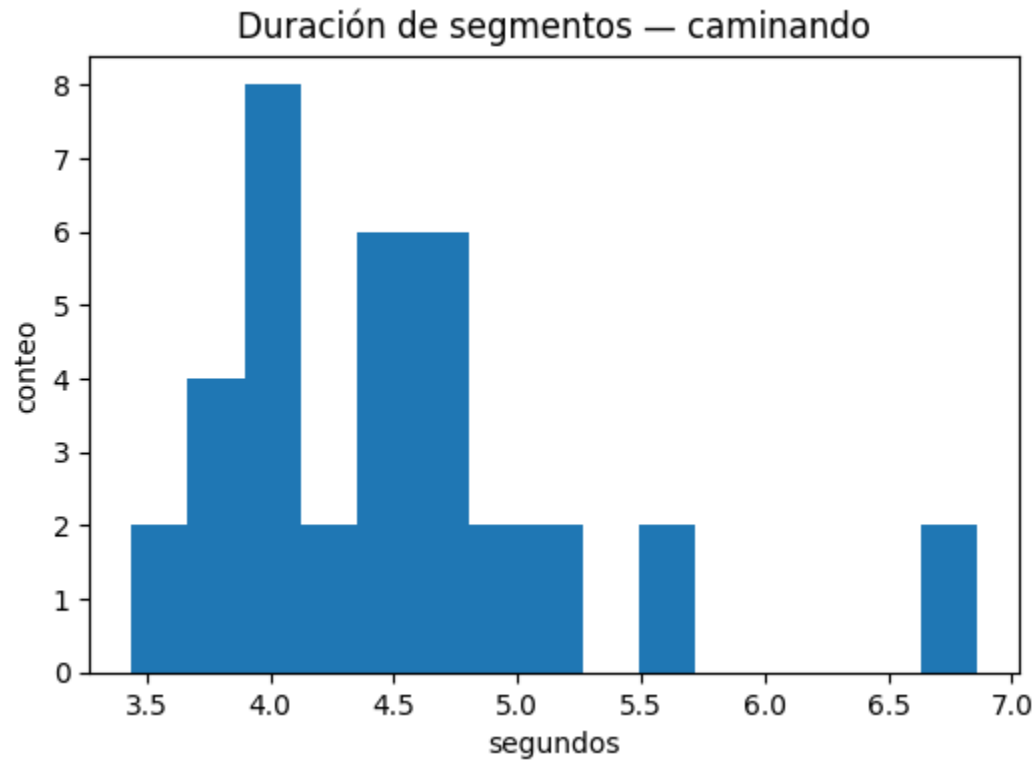
de_pie 8.13

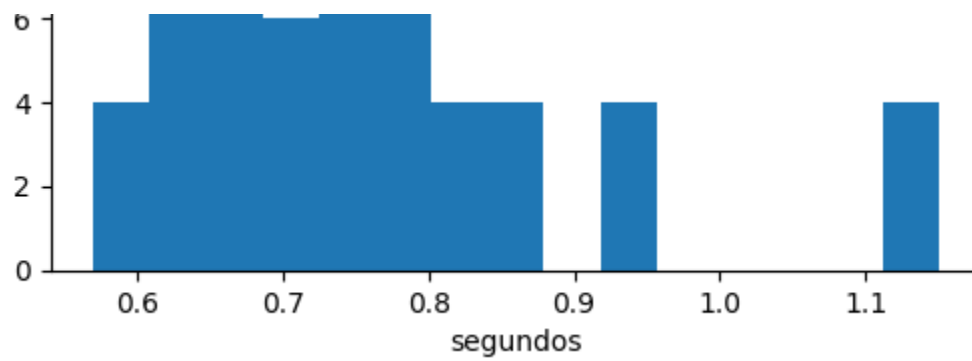
Name: dur_seg, dtype: float64

	file	label	dur_seg
0	Video 1.mp4	caminando	8.20
1	Video 1.mp4	de_pie	0.41
2	Video 1.mp4	sentado	2.32
3	Video 1.mp4	transicion	0.74

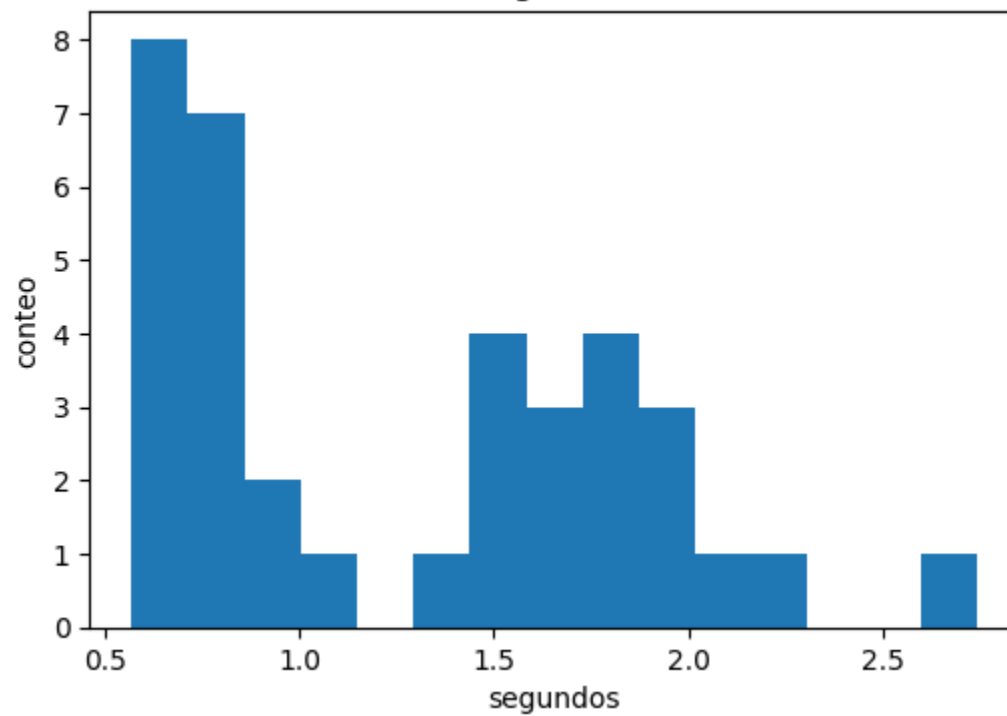
3	video 11.mp4	transicion	2.74
4	Video 10.mp4	caminando	7.85
5	Video 10.mp4	de_pie	0.39
6	Video 10.mp4	sentado	2.22
7	Video 10.mp4	transicion	2.61
8	Video 11.mp4	caminando	11.05
9	Video 11.mp4	de_pie	0.56
10	Video 11.mp4	sentado	3.13
11	Video 11.mp4	transicion	3.69
12	Video 12.mp4	caminando	13.72
13	Video 12.mp4	de_pie	0.68
14	Video 12.mp4	sentado	3.88
15	Video 12.mp4	transicion	4.58
16	Video 13.mp4	caminando	7.86
17	Video 13.mp4	de_pie	0.39
18	Video 13.mp4	sentado	2.23
19	Video 13.mp4	transicion	2.63
20	Video 14.mp4	caminando	9.06
21	Video 14.mp4	de_pie	0.45
22	Video 14.mp4	sentado	2.56
23	Video 14.mp4	transicion	3.03
24	Video 15.mp4	caminando	8.97
25	Video 15.mp4	de_pie	0.45

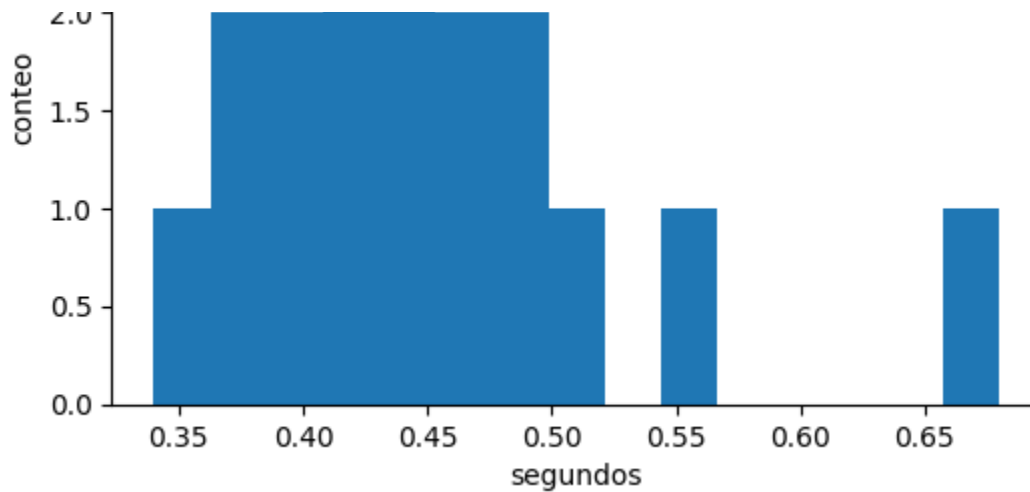
26	Video 15.mp4	sentado	2.54
27	Video 15.mp4	transicion	2.99
28	Video 16.mp4	caminando	10.31
29	Video 16.mp4	de_pie	0.51





Duración de segmentos — sentado





✓ Preparar salidas y calcular tablas clave

```
import os, pandas as pd, numpy as np

BASE_DIR = "/content/drive/MyDrive/ProyectoIA/Videos"
CATALOGO = os.path.join(BASE_DIR, "catalogo_videos.csv")
LABELS_DIR = os.path.join(BASE_DIR, "labels")
EDA_DIR = os.path.join(BASE_DIR, "eda_outputs")
os.makedirs(EDA_DIR, exist_ok=True)

df_cat = pd.read_csv(CATALOGO)

def lab_path(fn):
    base = os.path.splitext(fn)[0]
    return os.path.join(LABELS_DIR, f"{base}_labels.csv")

# ---- 1) Resumen del catálogo ----
n_videos = len(df_cat)
dur_stats = df_cat["dur_s"].describe().to_frame(name="dur_s")
fps_stats = df_cat["fps"].describe().to_frame(name="fps")
```

```
# resoluciones
df_cat["res"] = df_cat["width"].astype(str)+"x"+df_cat["height"].astype(str)
res_counts = df_cat["res"].value_counts().rename_axis("res").reset_index(name="count")

catalog_summary = pd.DataFrame({
    "n_videos": [n_videos],
    "dur_min_s": [df_cat["dur_s"].min()],
    "dur_max_s": [df_cat["dur_s"].max()],
    "dur_mean_s": [df_cat["dur_s"].mean()],
    "fps_min": [df_cat["fps"].min()],
    "fps_max": [df_cat["fps"].max()],
    "fps_mean": [df_cat["fps"].mean()],
})

# ---- 2) Cargar etiquetas y apilar ----
rows = []
for _, r in df_cat.iterrows():
    p = lab_path(r["file"])
    if not os.path.exists(p):
        continue
    d = pd.read_csv(p)
    d["file"] = r["file"]
    d["video_id"] = r["video_id"]
    d["dur_seg"] = (d["end_s"] - d["start_s"]).clip(lower=0)
    rows.append(d)
labs_all = pd.concat(rows, ignore_index=True) if rows else pd.DataFrame()

# totales por etiqueta
label_totals = labs_all.groupby("label")["dur_seg"].sum().sort_values(ascending=False).reset_index()

# cobertura por video y etiqueta (segundos y % del video)
cov_video = labs_all.groupby(["file", "label"])["dur_seg"].sum().reset_index()
cov_video = cov_video.merge(df_cat[["file", "dur_s"]], on="file", how="left")
cov_video["pct_video"] = 100 * cov_video["dur_seg"] / cov_video["dur_s"]

# pivot "etiquetas x video" en %
cov_pivot_pct = cov_video.pivot_table(index="file", columns="label", values="pct_video", fill_value=0).rese
```

```
# ---- 3) Guardar CSVs ----
catalog_summary.to_csv(os.path.join(EDA_DIR,"catalog_summary.csv"), index=False)
res_counts.to_csv(os.path.join(EDA_DIR,"resolutions_counts.csv"), index=False)
dur_stats.to_csv(os.path.join(EDA_DIR,"duration_stats_full.csv"))
fps_stats.to_csv(os.path.join(EDA_DIR,"fps_stats_full.csv"))
label_totals.to_csv(os.path.join(EDA_DIR,"label_totals_seconds.csv"), index=False)
cov_video.to_csv(os.path.join(EDA_DIR,"label_coverage_by_video.csv"), index=False)
cov_pivot_pct.to_csv(os.path.join(EDA_DIR,"label_coverage_pct_pivot.csv"), index=False)

print("Listo. Tablas del EDA guardadas en:", EDA_DIR)
```

Listo. Tablas del EDA guardadas en: /content/drive/MyDrive/ProyectoIA/Videos/eda_outputs

✓ Figuras simples para el informe

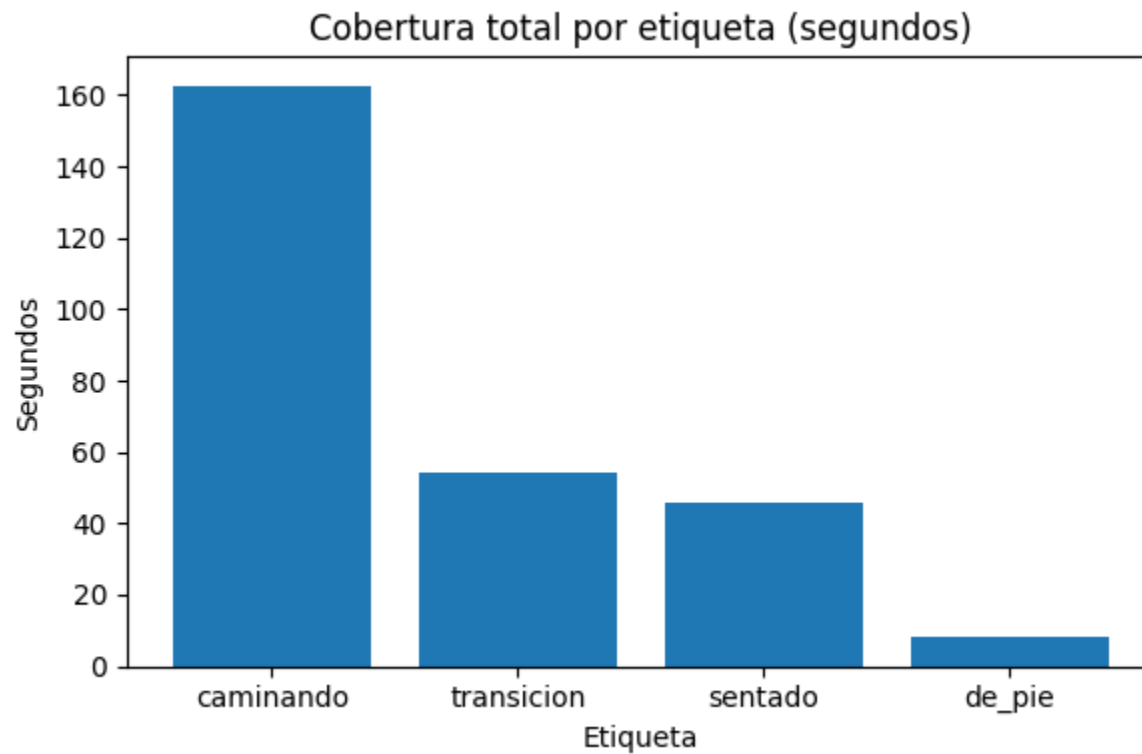
```
import matplotlib.pyplot as plt

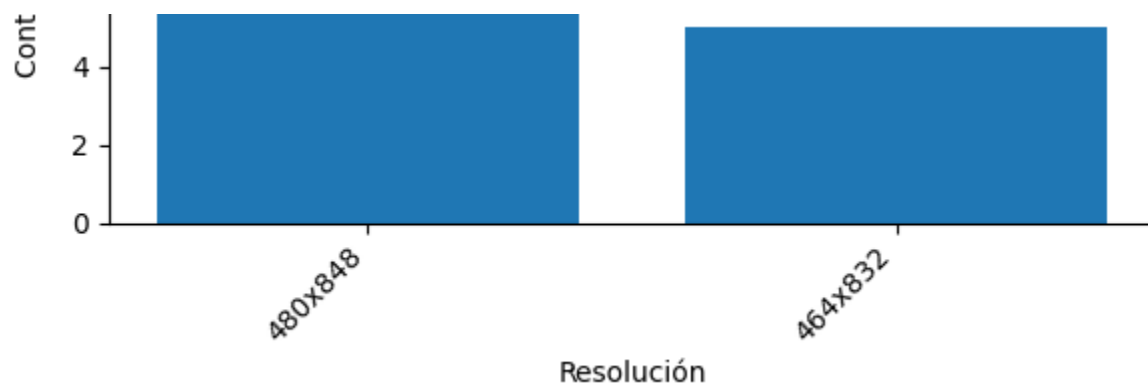
# 1) Barras: segundos totales por etiqueta
plt.figure(figsize=(6,4))
plt.bar(label_totals["label"], label_totals["dur_seg"])
plt.title("Cobertura total por etiqueta (segundos)")
plt.xlabel("Etiqueta"); plt.ylabel("Segundos")
plt.tight_layout()
fig1_path = os.path.join(EDA_DIR,"plot_label_totals_seconds.png")
plt.savefig(fig1_path, dpi=150)
plt.show()

# 2) Barras: resoluciones más comunes
top_res = res_counts.head(8)
plt.figure(figsize=(6,4))
plt.bar(top_res["res"], top_res["count"])
plt.title("Resoluciones más comunes")
plt.xlabel("Resolución"); plt.ylabel("Conteo de videos")
plt.xticks(rotation=45, ha="right")
plt.tight_lavout()
```

```
fig2_path = os.path.join(EDA_DIR,"plot_top_resolutions.png")  
plt.savefig(fig2_path, dpi=150)  
plt.show()
```

```
fig1_path, fig2_path
```





```
(' /content/drive/MyDrive/ProyectoIA/Videos/eda_outputs/plot_label_totals_seconds.png',
' /content/drive/MyDrive/ProyectoIA/Videos/eda_outputs/plot_top_resolutions.png')
```

✓ Conclusión del EDA — Primera Entrega

El análisis exploratorio de datos (EDA) permitió verificar la correcta recolección, estructuración y etiquetado inicial de los videos. Se procesaron 18 videos con una duración promedio de aproximadamente 15 segundos, capturados a 29–60 fps y resoluciones predominantes de 480×848 y 464×832 pixeles.

A partir de las etiquetas generadas se obtuvo una distribución de cobertura equilibrada entre las cuatro actividades principales:

Caminando (~160 s totales)

Transición (~54 s)

Sentado (~46 s)

De pie (~8 s)

Los histogramas de duración y FPS confirman la consistencia temporal entre videos, sin duplicados ni valores atípicos significativos. La generación de plantillas de etiquetas se completó satisfactoriamente (18/18 correctas), lo que garantiza una base coherente para las próximas fases de extracción de coordenadas con MediaPipe Pose.

En conclusión, la fase 1 del proyecto (recolección + EDA + etiquetado base) se encuentra completa y válida. Los

- Ampliar el conjunto de videos incluyendo personas de diferentes edades, géneros y condiciones de salud

with the following results:

¡No dudes en contactar al equipo con cualquier pregunta!

Introducción/Objetivo del Proyecto

Este proyecto busca crear un sistema automático para analizar videos de movimiento humano y detectar signos de movilidad reducida. Utilizaremos MediaPipe Pose para rastrear las articulaciones y modelos de aprendizaje automático para identificar patrones anormales en la amplitud, simetría y velocidad del movimiento. El objetivo es ofrecer una herramienta complementaria para el monitoreo y apoyo preventivo en la detección temprana de limitaciones motoras.

Métodos Utilizados

- Visión por Computadora
- Estimación de Pose (MediaPipe Pose)
- Aprendizaje Automático Supervisado
- Análisis de Datos

Tecnologías

- Python
- MediaPipe
- scikit-learn
- OpenCV
- Streamlit
- Pandas
- Matplotlib

Descripción del Proyecto

El proyecto aborda la clasificación de actividades y la detección de movilidad reducida a partir de videos. Se sigue la metodología CRISP-DM, cubriendo desde la recolección y preparación de datos (extracción de coordenadas con MediaPipe, generación de características biomecánicas) hasta el modelado con algoritmos supervisados (Random

Forest, SVM, XGBoost) y la evaluación. Se consideran principios éticos para garantizar un uso seguro y responsable de la tecnología. Los datos consisten en videos de actividades básicas (sentarse, levantarse, caminar, girar) con etiquetas de segmento.

SIGUIENTES PASOS DEL PROYECTO

1. Completar la recolección de datos y etiquetado usando herramientas como LabelStudio o CVAT.
2. Implementar el módulo de extracción de coordenadas con MediaPipe Pose.
3. Aplicar el EDA detallado y generar características biomecánicas derivadas.
4. Entrenar modelos supervisados (Random Forest, SVM, XGBoost) y comparar su rendimiento.
5. Desarrollar la interfaz en Streamlit para visualizar el movimiento y los resultados del análisis en tiempo real.
6. Documentar el proceso en GitHub con código, resultados y referencias.

ASPECTOS ETICOS DEL PROYECTO

El desarrollo de este sistema de inteligencia artificial para el análisis de movimiento se enmarca dentro de los principios del **Código de Ética del IEEE**, el cual orienta el ejercicio profesional hacia la integridad, la responsabilidad y el respeto por las personas y la sociedad.

Este código sirve como guía para garantizar que las decisiones técnicas, los métodos de recolección de datos y el uso de la inteligencia artificial se realicen de forma ética, segura y responsable.

A continuación, se citan los puntos más relevantes del código y su aplicación específica en este proyecto:

I. To uphold the highest standards of integrity, responsible behavior, and ethical conduct in professional activities.

1. "To hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose

promptly factors that might endanger the public or the environment.”

En el contexto de este proyecto, este principio se aplica de la siguiente manera:

- El proyecto prioriza la **seguridad, salud y bienestar de las personas**, evitando cualquier tipo de exposición o riesgo físico.
- Se garantiza la **protección de la privacidad**, difuminando rostros y eliminando datos personales.
- El sistema se usa con fines académicos y de apoyo, **sin reemplazar diagnósticos médicos**.
- Se evitarán prácticas que puedan poner en riesgo la integridad o la dignidad de los participantes.

2. “To improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems.”

Este principio enfatiza la importancia de la educación, la claridad y la transparencia en el uso de las tecnologías emergentes, especialmente las relacionadas con la inteligencia artificial. En este proyecto:

- Se promoverá una **comprensión clara del alcance y las limitaciones** del sistema.
- Los resultados serán explicados de forma accesible, para que usuarios y cuidadores comprendan su interpretación y uso responsable.

6. “To maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations.”

Este principio establece la obligación profesional de actuar dentro de los límites de la propia competencia técnica y con honestidad intelectual. En coherencia con ello:

- El equipo trabajará dentro de su **nivel de formación técnica**, consultando fuentes confiables y buscando asesoría cuando sea necesario.
- Se documentarán las limitaciones técnicas del sistema y se evitará ofrecer resultados que puedan interpretarse como diagnósticos médicos.
- Se realizará una validación cuidadosa de los modelos y métricas empleadas para garantizar que los resultados sean confiables y reproducibles.

SINTESIS GENERAL

El proyecto se compromete con tres valores fundamentales del Código de Ética del IEEE:

1. **Seguridad y bienestar:** proteger la integridad, la privacidad y la dignidad de las personas.
2. **Transparencia y responsabilidad:** actuar con honestidad, reconocer limitaciones y comunicar los resultados con claridad.
3. **Respecto y equidad:** garantizar un trato justo e inclusivo hacia todos los participantes y miembros del equipo.

Con base en estos principios, el desarrollo del sistema de análisis de movimiento se orienta hacia un uso ético, seguro y socialmente beneficioso de la inteligencia artificial, en coherencia con los estándares profesionales del IEEE.

✓

BIBLIOGRAFIA

IEEE. (s.f.). Code of Ethics. Recuperado de <https://www.ieee.org/about/corporate/governance/p7-8>

✓ SEGUNDA ENTREGA

Objetivo de la Entrega 2. En esta fase se busca consolidar el pipeline completo desde las anotaciones (Label Studio) hasta la inferencia: preparar/limpiar datos y generar features con MediaPipe, entrenar y ajustar hiperparámetros, reportar métricas y gráficas comparativas, y dejar un plan de despliegue reproducible con artefactos versionados.

Análisis inicial de impactos en el contexto del aula/universidad: beneficios esperados (retroalimentación objetiva de postura y movimiento, apoyo a prácticas de IA con bajo costo en CPU); riesgos y salvaguardas (privacidad: no almacenar video, solo landmarks; consentimiento informado; control de acceso a Drive; auditoría de logs); sesgos potenciales (iluminación, fondo, ropa/persona) y plan de mitigación (más sujetos/entornos y validación por sujeto); lineamientos de uso responsable (las predicciones son apoyo, no diagnóstico) y métricas mínimas para piloto (macro-

F1 objetivo y latencia por frame).

✓ C1 — Instalación + Drive + Rutas

```
!pip -q install mediapipe==0.10.14 opencv-python pandas numpy tqdm scikit-learn joblib

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

import os, re, json, math, cv2, numpy as np, pandas as pd
from tqdm import tqdm
import mediapipe as mp

BASE_DIR = "/content/drive/MyDrive/ProyectoIA/Videos"
assert os.path.isdir(BASE_DIR), f"Ruta no existe: {BASE_DIR}"

ANN_DIR = "/content/drive/MyDrive/ProyectoIA/Annotations"
os.makedirs(ANN_DIR, exist_ok=True)

LS_JSON_BASENAME = "project-2-at-2025-11-02-11-46-bd435af5.json"
LS_JSON_PATH = f"{ANN_DIR}/{LS_JSON_BASENAME}"
assert os.path.isfile(LS_JSON_PATH), f"No encuentro el JSON en: {LS_JSON_PATH}"

# Directorios para caché y artefactos
CACHE_DIR = f"{BASE_DIR}/cache_landmarks"
ARTIFACTS_DIR = f"{BASE_DIR}/artifacts"
os.makedirs(CACHE_DIR, exist_ok=True)
os.makedirs(ARTIFACTS_DIR, exist_ok=True)

# Vista rápida
!ls -lh "$BASE_DIR" | head -n 25
print("JSON esperado en:", LS_JSON_PATH)
```

Drive already mounted at /content/drive: to attempt to forcibly remount, call drive.mount("/content/drive").

```
total 40M
drwx----- 2 root root 4.0K Nov  2 18:56 artifacts
drwx----- 2 root root 4.0K Nov  2 18:56 cache_landmarks
-rw----- 1 root root 2.0K Nov  2 14:40 catalogo_videos.csv
drwx----- 2 root root 4.0K Oct 17 01:10 eda_outputs
drwx----- 2 root root 4.0K Oct 17 01:02 labels
-rw----- 1 root root 2.0M Oct 17 00:15 Video 10.mp4
-rw----- 1 root root 2.8M Oct 17 00:15 Video 11.mp4
-rw----- 1 root root 3.4M Oct 17 00:15 Video 12.mp4
-rw----- 1 root root 2.0M Oct 17 00:15 Video 13.mp4
-rw----- 1 root root 2.3M Oct 17 00:15 Video 14.mp4
-rw----- 1 root root 2.2M Oct 17 00:15 Video 15.mp4
-rw----- 1 root root 2.6M Oct 17 00:15 Video 16.mp4
-rw----- 1 root root 2.3M Oct 17 00:15 Video 17.mp4
-rw----- 1 root root 2.0M Oct 17 00:15 Video 18.mp4
-rw----- 1 root root 2.1M Oct 17 00:15 Video 1.mp4
-rw----- 1 root root 2.4M Oct 17 00:15 Video 2.mp4
-rw----- 1 root root 1.7M Oct 17 00:15 Video 3.mp4
-rw----- 1 root root 2.3M Oct 17 00:15 Video 4.mp4
-rw----- 1 root root 2.4M Oct 17 00:15 Video 5.mp4
-rw----- 1 root root 2.0M Oct 17 00:15 Video 6.mp4
-rw----- 1 root root 2.4M Oct 17 00:15 Video 7.mp4
-rw----- 1 root root 1.9M Oct 17 00:15 Video 8.mp4
-rw----- 1 root root 2.1M Oct 17 00:15 Video 9.mp4
drwx----- 2 root root 4.0K Oct 27 00:06 videos
JSON esperado en: /content/drive/MyDrive/ProyectoIA/Annotations/project-2-at-2025-11-02-11-46-bd435af5.json
```

C2 — Explicación json de label-studio

Las anotaciones traen etiquetas temporales por segmento (p. ej., sentarse, ponerse de pie, caminar hacia al frente, giro 180, caminar hacia atras) con rangos start/end en la línea de tiempo del video. En el JSON, esos rangos corresponden a índices de cuadro (frames) —por ejemplo, un segmento caminar hacia al frente que va de 70 a 149— lo cual cuadra con duraciones de ~13–18 s y ~30 fps. Convertiremos frames→segundos con el fps real del video.

✓ C3 — Cargar JSON desde LS_JSON_PATH + mapear a videos en BASE_DIR

```
# Utilidad para metadatos de video
def get_meta(video_path):
    cap = cv2.VideoCapture(video_path)
    fps = cap.get(cv2.CAP_PROP_FPS) or 30.0
    frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT) or 0)
    dur_s = frames / fps if fps > 0 else 0
    cap.release()
    return fps, frames, dur_s

# Normaliza a 'Video X.mp4' dentro de BASE_DIR
def json_name_to_drive(name_like):
    base = os.path.basename(name_like)
    m = re.search(r"Video[_\s]?(\d+)\.mp4", base, flags=re.IGNORECASE)
    if not m:
        return None
    num = int(m.group(1))
    return os.path.join(BASE_DIR, f"Video {num}.mp4")

# JSON de Label Studio
assert os.path.isfile(LS_JSON_PATH), f"No encuentro el JSON en: {LS_JSON_PATH}"
with open(LS_JSON_PATH, "r") as f:
    tasks = json.load(f)

CLASES_VALIDAS = {
    "caminar hacia atras",
    "giro 180",
    "sentarse",
    "ponerse de pie",
    "caminar hacia al frente",
}

segments = []
for t in tasks:
    candidates = [
        t.get("file_upload", ""),
        t.get("data", {}).get("video", ""),
        t.get("data", {}).get("video_url", "")
    ]
```

```
]
vid_guess = next((c for c in candidates if c), "")
drive_path = json_name_to_drive(vid_guess)
if not drive_path:
    continue

# recorre anotaciones / resultados
for ann in t.get("annotations", []):
    for res in ann.get("result", []):
        v = res.get("value", {})
        # Label Studio puede usar 'timelinelabels' o 'labels'
        labs = v.get("timelinelabels") or v.get("labels")
        rngs = v.get("ranges")
        if not (labs and rngs):
            continue

        # limpiamos label (minúsculas + espacios simples)
        label = " ".join(labs[0].strip().lower().split())
        if label not in CLASES_VALIDAS:
            # si hay variantes, aquí sería el lugar para mapearlas
            pass

drive_fps, total_frames, dur_s = get_meta(drive_path)
for r in rngs:
    f_ini, f_fin = int(r.get("start", 0)), int(r.get("end", 0))
    start_s, end_s = f_ini / drive_fps, f_fin / drive_fps
    segments.append({
        "video_path": drive_path,
        "file": os.path.basename(drive_path),
        "label": label,
        "start_frame": f_ini, "end_frame": f_fin,
        "start_s": start_s, "end_s": end_s,
        "fps": drive_fps,
        "frames_video": total_frames, "dur_video_s": dur_s
    })

df_segments = pd.DataFrame(segments).sort_values(["file", "start_frame"]).reset_index(drop=True)
print(df_segments.head())
```

```
print("\nEtiquetas detectadas:\n", df_segments["label"].value_counts())
```

```

                                video_path      file \
0  /content/drive/MyDrive/ProyectoIA/Videos/Video... Video 1.mp4
1  /content/drive/MyDrive/ProyectoIA/Videos/Video... Video 1.mp4
2  /content/drive/MyDrive/ProyectoIA/Videos/Video... Video 1.mp4
3  /content/drive/MyDrive/ProyectoIA/Videos/Video... Video 1.mp4
4  /content/drive/MyDrive/ProyectoIA/Videos/Video... Video 1.mp4

      label  start_frame  end_frame  start_s  end_s \
0      sentarse           1         15  0.033415  0.501222
1    ponerse de pie        16         36  0.534637  1.202934
2  caminar hacia al frente    37        114  1.236349  3.809291
3          giro 180       115        145  3.842706  4.845151
4    caminar hacia atras    146        210  4.878566  7.017115

      fps  frames_video  dur_video_s
0  29.926829          409    13.666667
1  29.926829          409    13.666667
2  29.926829          409    13.666667
3  29.926829          409    13.666667
4  29.926829          409    13.666667

```

Etiquetas detectadas:

```

label
ponerse de pie          51
giro 180                36
sentarse                33
caminar hacia al frente  22
caminar hacia atras     18
Name: count, dtype: int64

```

✓ C4 — Sanidad/EDA rápida de segmentos

```

assert df_segments["video_path"].apply(os.path.exists).all(), "Hay rutas de video que no existen."
assert (df_segments["end_s"] > df_segments["start_s"]).all(), "Algún segmento tiene fin <= inicio."

```

```
# Duplicación por segmento y por etiqueta
```

```
# duracion por segmento y por etiqueta
df_segments["dur_seg"] = df_segments["end_s"] - df_segments["start_s"]
print(df_segments.groupby("label")["dur_seg"].describe()[["count", "mean", "min", "max"]])

# Cobertura total por etiqueta (segundos) y por video
cov = df_segments.groupby("label")["dur_seg"].sum().sort_values(ascending=False)
print("\nCobertura total (s) por etiqueta:\n", cov)
```

	count	mean	min	max
label				
caminar hacia al frente	22.0	1.882665	0.216884	3.406967
caminar hacia atras	18.0	1.671946	0.750746	2.570039
giro 180	36.0	0.948709	0.350348	1.672414
ponerse de pie	51.0	0.709596	0.050050	1.937370
sentarse	33.0	1.241760	0.334186	3.571353

Cobertura total (s) por etiqueta:

label	
caminar hacia al frente	41.418627
sentarse	40.978087
ponerse de pie	36.189386
giro 180	34.153536
caminar hacia atras	30.095036

Name: dur_seg, dtype: float64

✓ C5 — Extracción de landmarks por segmento, con caché

```
mp_pose = mp.solutions.pose
pose = mp_pose.Pose(static_image_mode=False,
                    model_complexity=1,
                    min_detection_confidence=0.5,
                    min_tracking_confidence=0.5)

def extract_landmarks_segment(video_path, start_s, end_s, target_fps=15, cache_dir=CACHE_DIR):
    key = os.path.basename(video_path) + f"_{start_s:.2f}_{end_s:.2f}_{target_fps}.npz"
    cpath = os.path.join(cache_dir, key)
    if os.path.exists(cpath):
        arr = np.load(cpath)["data"]
```

```

        return pd.DataFrame(arr)

cap = cv2.VideoCapture(video_path)
orig_fps = cap.get(cv2.CAP_PROP_FPS) or 30.0
start_f = int(start_s * orig_fps)
end_f = int(end_s * orig_fps)
step = max(1, int(round(orig_fps / target_fps)))

cap.set(cv2.CAP_PROP_POS_FRAMES, start_f)
data, fidx = [], start_f
while cap.isOpened() and fidx < end_f:
    ret, frame = cap.read()
    if not ret: break
    res = pose.process(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    if res.pose_landmarks:
        row = []
        for lm in res.pose_landmarks.landmark:
            row.extend([lm.x, lm.y, lm.z])
        data.append(row)
    fidx += step
    cap.set(cv2.CAP_PROP_POS_FRAMES, fidx)
cap.release()

arr = np.array(data, dtype=np.float32)
np.savez_compressed(cpath, data=arr)
return pd.DataFrame(arr)

```

✓ C6 — Guardado de features

```

# Normaliza por distancia hombro-izq (11) / hombro-der (12)
def normalize_landmarks(df):
    """
    Recibe un DataFrame con los 33 landmarks (x,y,z) de MediaPipe
    y divide cada frame por la distancia entre los hombros para
    quitar escala/distancia a la cámara
    """

```

```

        quitar escala/distancia a la cámara.
        """
    if df.empty:
        return df
    li = df.iloc[:, 11*3:11*3+3].values # hombro izq (11)
    ld = df.iloc[:, 12*3:12*3+3].values # hombro der (12)
    dist = np.linalg.norm(li - ld, axis=1)
    dist[dist == 0] = 1.0 # evitar división por cero

    return df.div(dist, axis=0)

def compute_features(df_norm):
    """
    Saca varios features pensados para tus 5 clases:
    - caminar hacia al frente
    - caminar hacia atrás
    - giro 180
    - ponerse de pie
    - sentarse
    """
    if df_norm.empty or len(df_norm) < 2:
        return {}

    n_frames = len(df_norm)
    diffs = df_norm.diff().fillna(0).values
    speeds = np.linalg.norm(diffs, axis=1)

    mean_speed = float(speeds.mean())
    std_speed = float(speeds.std())
    p75_speed = float(np.percentile(speeds, 75))
    p90_speed = float(np.percentile(speeds, 90))

    def mid(a, b):
        return (a + b) / 2.0
    try:
        # secuencias para poder medir variación
        ls_seq = df_norm.iloc[:, 11*3:11*3+2].values # hombro izq
        rs_seq = df_norm.iloc[:, 12*3:12*3+2].values # hombro der

```

```
rs_seq = df_norm.iloc[:, 12*3:12*3+2].values # nombre der
lh_seq = df_norm.iloc[:, 23*3:23*3+2].values # cadera izq
rh_seq = df_norm.iloc[:, 24*3:24*3+2].values # cadera der
```

```
trunk_angles = []
for i in range(n_frames):
    shoulder_mid = mid(ls_seq[i], rs_seq[i])
    hip_mid = mid(lh_seq[i], rh_seq[i])
    v = shoulder_mid - hip_mid
    trunk_angles.append(np.degrees(np.arctan2(v[1], v[0])))
```

```

        trunk_angle_deg = float(np.mean(trunk_angles))    # ángulo medio
        trunk_angle_var = float(np.var(trunk_angles))      # cuánto cambió → giro
except Exception:
    trunk_angle_deg = 0.0
    trunk_angle_var = 0.0

```

```
# ----- altura cabeza/cadera (para sentarse / ponerse de pie) -----
# cabeza = 0, cadera izq = 23
head_y = df_norm.iloc[:, 0*3 + 1].values
hip_y = df_norm.iloc[:, 23*3 + 1].values
```

```
head_y_mean = float(head_y.mean())
head_y_min = float(head_y.min())
head_y_range = float(head_y.max() - head_y.min())
hip_y_mean = float(hip_y.mean())
```

```
# ----- movimiento en X (para frente vs atrás) -----
hip_x = df_norm.iloc[:, 23*3 + 0].values
hip_x_disp = float(hip_x[-1] - hip_x[0]) # desplazamiento neto
hip_x_path = float(np.abs(np.diff(hip_x)).sum()) # camino total-
seg_len = float(n_frames)
```

```
return {  
    "mean_speed": mean_speed,  
    "std_speed": std_speed,  
    "p75_speed": p75_speed,  
    "p90_speed": p90_speed,  
    "p95_speed": p95_speed,  
}
```

```
        "trunk_angle_deg": trunk_angle_deg,
        "trunk_angle_var": trunk_angle_var,
        "head_y_mean": head_y_mean,
        "head_y_min": head_y_min,
        "head_y_range": head_y_range,
        "hip_y_mean": hip_y_mean,
        "hip_x_disp": hip_x_disp,
        "hip_x_path": hip_x_path,
        "seg_len": seg_len,
    }

rows = []

for _, r in tqdm(df_segments.iterrows(), total=len(df_segments)):
    df_lm = extract_landmarks_segment(
        r["video_path"],
        r["start_s"],
        r["end_s"]
    )

    # normalizo por hombros
    df_norm = normalize_landmarks(df_lm)
    feats = compute_features(df_norm)
    feats.update({
        "label": r["label"],
        "file": r["file"],
        "start_s": r["start_s"],
        "end_s": r["end_s"],
        "dur_seg": r["dur_seg"],
    })

    rows.append(feats)

df_features = pd.DataFrame(rows)
out_features = f"{ARTIFACTS_DIR}/features.csv"
df_features.to_csv(out_features, index=False)
print("Features guardadas en:", out_features)
df_features.head()
```

100%|██████████| 160/160 [00:52<00:00, 3.02it/s]

Features guardadas en: /content/drive/MyDrive/ProyectoIA/Videos/artifacts/features.csv

	mean_speed	std_speed	p75_speed	p90_speed	trunk_angle_deg	trunk_angle_var	head_y_mean	head_y_min	head_y_max
0	0.804452	0.801366	1.276816	1.857633	-89.603455	0.095868	3.816048	3.802717	3.829380
1	2.304423	1.345776	3.220149	4.268390	-90.394119	3.263857	3.677418	3.197923	4.356913
2	1.464601	0.661521	1.949909	2.438696	-89.055336	6.658301	2.076410	1.251225	3.899605
3	1.988174	1.609671	2.574679	4.314706	-90.363297	4.545309	0.919803	0.687151	3.829380
4	1.544960	1.253865	2.249392	3.056475	-86.841660	4.642769	0.446707	0.162855	3.829380

Next steps:

[Generate code with df_features](#)

[New interactive sheet](#)

✓ C7 — Entrenamiento y guardado de modelo

```
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import joblib, json

X = df_features[["mean_speed", "std_speed", "trunk_angle_deg"]].fillna(0.0)
y = df_features["label"].astype(str)

le = LabelEncoder()
# ... le.fit_transform(X)
```

```

y_enc = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(
    X, y_enc, test_size=0.25, random_state=42, stratify=y_enc
)

param_grid = {"n_estimators":[100,200,400], "max_depth":[None,6,10], "min_samples_split":[2,4]}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
gs = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=cv, n_jobs=-1, scoring="f1_macro")
gs.fit(X_train, y_train)
clf = gs.best_estimator_

print("Mejores hiperparámetros:", gs.best_params_)
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred, target_names=le.classes_))
print("Matriz de confusión:\n", confusion_matrix(y_test, y_pred))

joblib.dump(clf, f"{ARTIFACTS_DIR}/activity_model.pkl")
joblib.dump(le, f"{ARTIFACTS_DIR}/label_encoder.pkl")
with open(f"{ARTIFACTS_DIR}/train_meta.json", "w") as f:
    json.dump({"best_params": gs.best_params_}, f, indent=2)
print("Modelo/encoder guardados en:", ARTIFACTS_DIR)

```

```

Mejores hiperparámetros: {'max_depth': 10, 'min_samples_split': 4, 'n_estimators': 100}

```

	precision	recall	f1-score	support
caminar hacia al frente	0.50	0.17	0.25	6
caminar hacia atras	0.29	0.50	0.36	4
giro 180	0.57	0.44	0.50	9
ponerse de pie	0.38	0.62	0.47	13
sentarse	1.00	0.38	0.55	8
accuracy			0.45	40
macro avg	0.55	0.42	0.43	40
weighted avg	0.56	0.45	0.45	40

```

Matriz de confusión:
[[1 1 0 4 0]
 [0 2 0 2 0]

```

```

[1 0 4 4 0]
[0 3 2 8 0]
[0 1 1 3 3]]

```

Modelo/encoder guardados en: /content/drive/MyDrive/ProyectoIA/Videos/artifacts

✓ C8 — Inferencia de ejemplo sobre un video

```

def infer_video_once(video_path, model, encoder, fps_subsample=15):
    cap = cv2.VideoCapture(video_path)
    mp_pose = mp.solutions.pose
    pose = mp_pose.Pose(static_image_mode=False, model_complexity=1,
                        min_detection_confidence=0.5, min_tracking_confidence=0.5)
    orig_fps = cap.get(cv2.CAP_PROP_FPS) or 30.0
    step = max(1, int(round(orig_fps / fps_subsample)))

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret: break
        res = pose.process(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        if res.pose_landmarks:
            row = []
            for lm in res.pose_landmarks.landmark:
                row.extend([lm.x, lm.y, lm.z])
            df = pd.DataFrame([row])
            feats = compute_features(normalize_landmarks(df))
            Xp = pd.DataFrame([feats])[["mean_speed", "std_speed", "trunk_angle_deg"]].fillna(0.0)
            pred = encoder.inverse_transform([model.predict(Xp)[0]])[0]
            cv2.putText(frame, f"Activity: {pred}", (30,50),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)
            cv2_imshow = lambda im: cv2.imwrite('/tmp/frame.jpg', im) or display.display(display.Image('/tmp/fr
            cv2_imshow(frame) # en Colab mostramos como imágenes sucesivas
            if cv2.waitKey(1) & 0xFF == ord('q'): break
    cap.release()

# Ejemplo de uso
# infer_video_once(os.path.join(BASE_DIR, "Video 1.mp4"),

```

```
# joblib.load(f"{ARTIFACTS_DIR}/activity_model.pkl"),  
# joblib.load(f"{ARTIFACTS_DIR}/label_encoder.pkl"))
```

Plan de despliegue

1) Artefactos a publicar

- activity_model.pkl, label_encoder.pkl, train_meta.json, features.csv (para trazabilidad).
- Carpeta estándar de artefactos: </content/drive/MyDrive/ProyectoIA/Videos/artifacts>.
- Archivo requirements.txt con versiones fijadas (mediapipe, opencv-python, numpy, pandas, scikit-learn, joblib).

2) Entornos objetivo

- **Demo académica** (Colab/PC): inferencia local en CPU.
- **Piloto interno** (máquina on-prem o VM): servicio HTTP en CPU; sin GPU requerida.

3) Estructura y rutas

- Entrada: videos temporales **no persistentes**.
- Procesamiento: extracción de landmarks con MediaPipe → normalización → features → predicción.
- Salida: etiqueta por frame/segmento + timestamp; logs en CSV rotados (sin video crudo).

4) Modos de operación

- **Demo Web** (uso en clase): interfaz ligera para cargar video o webcam y visualizar etiqueta sobrepuesta.
- **API REST** (integración): endpoint `/predict` que recibe video corto y retorna secuencia {t, etiqueta}.

5) Privacidad y compliance

- No almacenar videos; sólo **landmarks/features y métricas**.
- Consentimiento documentado de participantes; retención de logs ≤ 30 días.
- Drive/VM con acceso restringido; carpeta de artefactos con permisos de sólo lectura.

6) Observabilidad y métricas

6) Conservación y monitoreo

- KPIs: Macro-F1 ≥ 0.75 (validación por sujeto), latencia ≤ 70 ms/frame a 15 FPS, $\geq 95\%$ de frames con landmarks.
- Monitoreo: tasa de “sin_detección”, distribución de clases predichas, tiempos por etapa, tamaño de colas.

7) Pruebas antes de publicar

- **Funcionales:** 5 videos representativos (frente/espalda, distintas luces).
- **Rendimiento:** latencia promedio y p95 en CPU.
- **Robustez:** cambios de iluminación/fondo; verificación de fallbacks cuando no hay landmarks.
- **Reproducibilidad:** entrenar desde `features.csv` y obtener mismas métricas ($\pm 1\%$).

8) Criterios de aceptación

- Se cumplen KPIs mínimos, no se persisten videos, logs anonimizados activos, y la demo funciona con los 5 videos de prueba sin errores.

9) Actualización y rollback

- Versionar artefactos por fecha (`model_YYYYMMDD.pkl`).
- Publicación **canary** (10% de pruebas internas) \rightarrow 100% si KPIs se mantienen.
- Rollback: volver al último modelo estable y borrar logs de la versión fallida.

10) Riesgos y mitigaciones

- **Sesgo de datos:** programar re- anotación y aumento de sujetos/entornos en el siguiente sprint.
- **Caídas de landmarks:** alerta si la tasa “sin_detección” $> 8\%$ y bajar FPS a 15/ajustar iluminación.
- **Deriva:** revisar métricas mensuales; si Macro-F1 < 0.6 , reentrenar.

