Modelos ResNET para determinar la progresión de la Dementia

Daniella Guerra & Tawny Torres

2026-05-11

Table of contents

1 MRI - Deep learning solutions fot dementia Disease prediction

De acuerdo con la organización mundial de la salud la dementia es un término que engloba varias enfermedades que afectan la memoria, el pensamiento y la capacidad para realizar actividades cotidianas. Esta enfermedad empeora con el tiempo, relacionada a condiciones que destruyen las células nerviosas y dañan el cerebro, lo que conduce a al deterioro de funciones cognitivas. La detección de esta destrucción resulta relevante para la salud de los pacientes e influye en las mejoras a su calidad de vida.

Las imágenes de resonancias magnéticas (RM) tienen un papel en el diagnóstico de la demencia, consiste en permitir el estudio de las estructuras cerebrales y cómo cambian con el tiempo. En este sentido, los cambios en el hipocampo, así como en las regiones frontal y parietal, son marcadores evidentes del progreso de la enfermedad hacia la demencia.

La Iniciativa de Neuroimagen de la Enfermedad de Alzheimer (ADNI) es la colección más importante y extensa de información relacionada con el Alzheimer, que contiene diferentes tipos de imágenes, información genética, información demográfica, pruebas cognitivas y biomarcadores en líquido cefalorraquídeo.

Se propone técnicas de Deep Learning y Convolutional Neural Networks, para analizar si []

2 ETL -IMAGES

```
# [Config] Librerías
import pandas as pd
import os, re
import nibabel as nib
from datetime import datetime
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
import matplotlib as plt
import statsmodels
from statsmodels.stats.proportion import proportions_ztest
import os
import numpy as np
import SimpleITK as sitk
from pathlib import Path
```

2.1 *Imágenes descargadas*

Leer atributos de las imágenes

```
# CARACTERISTICAS DE LAS IMAGENES DESCARGADAS

# [Config] Rutas
output_dir  = r"C:\Users\usuario\MRI\IMAGES_NII"

# [] Guardar atributos de las imágenes
records = []

for root, dirs, files in os.walk(output_dir):
    for file in files:
```

```
        if file.endswith(".nii"):
            full_path = os.path.normpath(os.path.join(root, file))

            #  Extraer ID del sujeto
            match_id = re.search(r'(\d{3}_S_\d{4})', full_path)
            sujeto_id = match_id.group(1) if match_id else None

            #  Extraer fecha del estudio (YYYY-MM-DD)
            match_fecha = re.search(r'(\d{4}-\d{2}-\d{2})', full_path)
            fecha = match_fecha.group(1) if match_fecha else None

            #  Extraer ID de imagen (ej. I######)
            match_img = re.search(r'(I\d+)', full_path)
            imagen_id = match_img.group(1) if match_img else None

            # Leer metadatos del NIfTI
            try:
                img = nib.load(full_path)
                data = img.get_fdata()

                header = img.header
                shape = img.shape
                voxel_size = header.get_zooms()
                voxel_volume = np.prod(voxel_size)
                total_volume = voxel_volume*np.prod(shape)
                datatype = str(header.get_data_dtype())
                mean_intensity = np.mean(data)
                std_intensity = np.std(data)
                orientation = nib.aff2axcodes(img.affine)
                units = header.get_xyzt_units()

                records.append([
                    sujeto_id, fecha, imagen_id,
                    shape, voxel_size, datatype,
                    voxel_volume, total_volume,
                    mean_intensity, std_intensity,
                    orientation, units, full_path
                ])
            except Exception as e:
                print(f"  Error leyendo {file}: {e}")

df_images = pd.DataFrame(records, columns=[
```

```
    "sujeto_id", "fecha_imagen", "imagen_id",
    "shape", "voxel_size", "datatype",
    "voxel_volume_mm3", "total_volume",
    "mean_intensity", "std_intensity",
    "orientation", "units","ruta"
])

# Convertir fecha a tipo datetime
df_images["fecha_imagen"] = pd.to_datetime(df_images["fecha_imagen"], errors="coerce")

print(f"En total {len(df_images)} imágenes fueron cargadas correctamente.")
```

```
En total 220 imágenes fueron cargadas correctamente.
```

```
df_images.to_csv('../Data/ADNI_Images.csv', index=False)
```

Combinar con datos de descarga

```
# Archivos sobre imágenes descargadas
# ------------------------------------

descargas = pd.read_csv('../Data/ADNI_Descargas.csv', sep=';')
descargas.rename(columns={'Subject':'sujeto_id', 'Acq Date': 'fecha_imagen', 'Image Data ID'

# Vistas válidas
descargas = descargas[
    (descargas['Visit'] != 'sc') &
    (descargas['Visit'] != 'bl') &
    (descargas['Visit'] != 'nv')
]

# Correción de fechas
descargas['fecha_imagen'] = pd.to_datetime(
    descargas['fecha_imagen'],
    errors='coerce'
)

# Eliminar imágenes no usadas
descargas = descargas.drop(columns=['Group','Modality', 'Type', 'Format', 'Downloaded', 'Des

print(
```

```
    "Sobre las imágenes descargadas: "
    f"\nImágenes: {len(descargas)}"
    f"\nPacientes únicos: {descargas['sujeto_id'].nunique()}"
    f"\nVistras únicas: {descargas['Visit'].nunique()}, {descargas['Visit'].unique()}"
    f"\nAtributos: {descargas.columns}"
    )

#Image Data ID;Subject;Group;Sex;Age;Visit;Modality;Description;Type;Acq Date;Format;Downloa
```

```
Sobre las imágenes descargadas:
Imágenes: 246
Pacientes únicos: 54
Vistras únicas: 7, ['m18' 'm36' 'm06' 'm12' 'm24' 'm48' 'm60']
Atributos: Index(['imagen_id', 'sujeto_id', 'Sex', 'Age', 'Visit', 'fecha_imagen'], dtype='ob
```

```
adni_images = pd.merge(
    df_images,
    descargas,
    on=['imagen_id', 'sujeto_id', 'fecha_imagen'],
    how='left'  # o 'left', 'outer', etc.
)
print(
    "Sobre los Imagenes + Pacientes: "
    f"\nImágenes: {len(adni_images)}"
    f"\nPacientes únicos: {adni_images['sujeto_id'].nunique()}"
    f"\nVistras únicas: {adni_images['Visit'].nunique()}, {adni_images['Visit'].unique()}"
    f"\nAtributos: {adni_images.columns}"
    )
```

```
Sobre los Imagenes + Pacientes:
Imágenes: 220
Pacientes únicos: 51
Vistras únicas: 7, ['m18' 'm36' 'm06' 'm24' 'm48' 'm12' 'm60']
Atributos: Index(['sujeto_id', 'fecha_imagen', 'imagen_id', 'shape', 'voxel_size',
       'datatype', 'voxel_volume_mm3', 'total_volume', 'mean_intensity',
       'std_intensity', 'orientation', 'units', 'ruta', 'Sex', 'Age', 'Visit'],
      dtype='object')
```

Combinar con datos del paciente BioMarcadores

```
# Archivos ADNI pacientes
adni = pd.read_csv('../Data/ADNI_pacients.csv', sep=';')
adni.rename(columns = {'VISCODE':'Visit'}, inplace=True)
adni = adni.drop(columns=['AGE', 'PTGENDER', 'EXAMDATE'])

print(
    "Sobre los atributos de los pacientes: "
    f"\nImágenes: {len(adni)}"
    f"\nPacientes únicos: {adni['sujeto_id'].nunique()}"
    f"\nVistras únicas: {adni['Visit'].nunique()}, {adni['Visit'].unique()}"
    f"\nAtributos: {adni.columns}"
    )
```

```
Sobre los atributos de los pacientes:
Imágenes: 228
Pacientes únicos: 51
Vistras únicas: 6, ['m06' 'm12' 'm18' 'm24' 'm30' 'm36']
Atributos: Index(['sujeto_id', 'Visit', 'DX', 'PTEDUCAT', 'APOE4', 'CDRSB', 'MMSE',
       'ADAS13', 'FAQ', 'RAVLT_immediate', 'RAVLT_learning',
       'RAVLT_forgetting', 'DIGITSCOR', 'TRABSCOR', 'Ventricles',
       'Hippocampus', 'WholeBrain', 'Entorhinal', 'Fusiform', 'MidTemp',
       'ICV'],
      dtype='object')
```

```
out = pd.merge(
    adni_images,
    adni,
    on=['sujeto_id', 'Visit'],
    how='left'  # o 'left', 'outer', etc.
)
print(
    "Sobre los Imagenes + Pacientes: "
    f"\nImágenes: {len(out)}"
    f"\nPacientes únicos: {out['sujeto_id'].nunique()}"
    f"\nVistras únicas: {out['Visit'].nunique()}, {out['Visit'].unique()}"
    f"\nAtributos: {out.columns}"
    )
```

```
Sobre los Imagenes + Pacientes:
Imágenes: 220
Pacientes únicos: 51
```

```
Vistras únicas: 7, ['m18' 'm36' 'm06' 'm24' 'm48' 'm12' 'm60']
Atributos: Index(['sujeto_id', 'fecha_imagen', 'imagen_id', 'shape', 'voxel_size',
       'datatype', 'voxel_volume_mm3', 'total_volume', 'mean_intensity',
       'std_intensity', 'orientation', 'units', 'ruta', 'Sex', 'Age', 'Visit',
       'DX', 'PTEDUCAT', 'APOE4', 'CDRSB', 'MMSE', 'ADAS13', 'FAQ',
       'RAVLT_immediate', 'RAVLT_learning', 'RAVLT_forgetting', 'DIGITSCOR',
       'TRABSCOR', 'Ventricles', 'Hippocampus', 'WholeBrain', 'Entorhinal',
       'Fusiform', 'MidTemp', 'ICV'],
      dtype='object')
```

```
out.to_csv('../Data/ANDI_out.csv', index=False)
```

2.2 *Imputación*

Imputación de acuerdo al sujeto

```
# IMPUTACIÓN
df_dx = out.copy()

# DIAGNÓSTIVCO
# ------------------------------
df_dx = df_dx.sort_values(["sujeto_id", "Visit"])
mapping_order = {"CN": 0, "MCI": 1, "Dementia": 2}
df_dx["DX_num"] = df_dx["DX"].map(mapping_order)
sujetos_sin_dx = (
    df_dx.groupby("sujeto_id")["DX"]
    .apply(lambda x: x.isna().all())
)
df_dx.loc[df_dx["sujeto_id"].isin(sujetos_sin_dx[sujetos_sin_dx].index), "DX_num"] = mapping
df_dx["DX_num"] = df_dx.groupby("sujeto_id")["DX_num"].ffill()
df_dx["DX_num"] = df_dx.groupby("sujeto_id")["DX_num"].bfill()
reverse_mapping = {v: k for k, v in mapping_order.items()}
df_dx["DX_imputed"] = df_dx["DX_num"].map(reverse_mapping)
df_dx = df_dx.drop(columns=["DX_num"])

# NIVEL EDUCATIVO
# ------------------------------
mediana_global = df_dx["PTEDUCAT"].median()
mediana_sujeto = df_dx.groupby("sujeto_id")["PTEDUCAT"].transform("median")
# 3. Asignar:
```

```
# - si el sujeto tiene medianas válidas, usarla
# - si el sujeto no tiene ningún dato, usar la mediana global
df_dx["PTEDUCAT_imputed"] = mediana_sujeto.fillna(mediana_sujeto)
df_dx["Educat"] = mediana_sujeto.fillna(mediana_global)

# SEXO - VISITA - TARFET
# ------------------------------
df_dx["Sexo"] = df_dx["Sex"].map({"M":0, "F":1})
df_dx["Visita"] =  df_dx['Visit'].str.extract(r'(\d+)').astype(int)
map_dx = { "MCI":0, "Dementia":1}
df_dx["is_dementia"] = df_dx["DX_imputed"].map(map_dx)

# LABEL de progresion
label_por_sujeto = (
    df_dx
    .sort_values(['sujeto_id', 'Visita'])
    .groupby('sujeto_id')['is_dementia']
    .apply(lambda x: 1 if (x.diff() == 1).any() or (x.iloc[0] == 1) else 0)
)
df_dx['label'] = df_dx['sujeto_id'].map(label_por_sujeto)
```

Normalización de BioMarcadores

```
# VARIABLE VOLUMÉTRICAS
cognitivas = [
    "APOE4", #APOE4 es una variante genética de la apolipoproteína E
    "CDRSB", #"Suma de cajas del Clinical Dementia Rating (CDR); mide la severidad de la deme
    "MMSE", #"Mini-Mental State Examination; evaluación global del estado cognitivo (máx. 30
    "ADAS13", #"Alzheimer's Disease Assessment Scale - 13 ítems; mide deterioro cognitivo en
    "FAQ", #"Functional Activities Questionnaire; evalúa la capacidad funcional en actividade
    "RAVLT_immediate", # "Puntuación inmediata en la prueba verbal de aprendizaje (Rey Audito
    "RAVLT_learning", # "Puntuación de aprendizaje acumulado en RAVLT; mide retención verbal
    "RAVLT_forgetting", # "Índice de olvido en RAVLT; diferencia entre aprendizaje y recuerdo
    "DIGITSCOR", #"Digit Span Score; mide memoria de trabajo y atención mediante secuencias r
    "TRABSCOR", # "Trail Making Test B Score; evalúa función ejecutiva y flexibilidad cogniti
]
volumen = [
    "Ventricles", # "Volumen de los ventrículos cerebrales; puede indicar atrofia cerebral."
    "Hippocampus", # "Volumen del hipocampo; clave en memoria y afectado en Alzheimer.",
    "WholeBrain", # "Volumen total del cerebro; útil para evaluar atrofia global.",
    "Entorhinal", # "Volumen de la corteza entorrinal; región afectada tempranamente en Alzhe
    "Fusiform", #"Volumen del giro fusiforme; relacionado con reconocimiento visual.",
```

```
    "MidTemp", # "Volumen del lóbulo temporal medio; implicado en memoria y procesamiento auc
    "ICV", #"Volumen intracraneal total; usado para normalizar medidas volumétricas."
]
```

```
# NORMALIZAR
def imputar_y_normalizar(df, variables, nombre_grupo):

    print('-'*50)
    # Filtrar solo variables numéricas válidas
    variables_num = [v for v in variables if v in df.columns and df[v].dtype.kind in "iufc"]
    print(f"Variables numéricas para imputación ({nombre_grupo}):\n{variables_num}")

    # Subset de datos
    datos = df[variables_num].copy()

    # Imputación multivariada
    # Modelo bayesiano iterativo para predecir valores faltantes en función de las demás vari
    imputer = IterativeImputer(random_state=42, max_iter=20, sample_posterior=True)
    datos_imputados = imputer.fit_transform(datos)

    # Convertir a DataFrame imputado
    df_imputado = pd.DataFrame(datos_imputados, columns=variables_num, index=df.index)

    # Reemplazar en el DataFrame original
    for v in variables_num:
        df[v] = df_imputado[v]

    print("  Imputación completada.")

    # Normalización z-score
    scaler = StandardScaler()
    df_std = pd.DataFrame(
        scaler.fit_transform(df[variables_num]),
        columns=[v + "_std" for v in variables_num],
        index=df.index
    )

    # Concatenar al DataFrame original
    df = pd.concat([df, df_std], axis=1)

    return df
```

```
print('Imputación')
df_dx = imputar_y_normalizar(df_dx, cognitivas, "cognitivas")
df_dx = imputar_y_normalizar(df_dx, volumen, "volumen")
```

```
Imputación
--------------------------------------------------
Variables numéricas para imputación (cognitivas):
['APOE4', 'CDRSB', 'MMSE', 'ADAS13', 'FAQ', 'RAVLT_immediate', 'RAVLT_learning', 'RAVLT_forge
 Imputación completada.
--------------------------------------------------
Variables numéricas para imputación (volumen):
['Ventricles', 'Hippocampus', 'WholeBrain', 'Entorhinal', 'Fusiform', 'MidTemp', 'ICV']
 Imputación completada.
```

Datos Tabulares para el modelo

```
columnas = [
    'sujeto_id','label','is_dementia', 'Visita', 'Age', 'Sexo', 'Educat',
    'APOE4_std', 'CDRSB_std','MMSE_std', 'ADAS13_std', 'FAQ_std',
    'RAVLT_immediate_std','RAVLT_learning_std', 'RAVLT_forgetting_std',
    'DIGITSCOR_std', 'TRABSCOR_std', 'Ventricles_std', 'Hippocampus_std',
    'WholeBrain_std', 'Entorhinal_std', 'Fusiform_std', 'MidTemp_std', 'ICV_std'
]

df_tab = df_dx[columnas]
```

```
df_tab.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 220 entries, 47 to 199
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   sujeto_id             220 non-null    object
 1   label                 220 non-null    int64
 2   is_dementia           220 non-null    int64
 3   Visita                220 non-null    int64
 4   Age                   220 non-null    int64
 5   Sexo                  220 non-null    int64
 6   Educat                220 non-null    float64
 7   APOE4_std             220 non-null    float64
```

```
 8   CDRSB_std              220 non-null    float64
 9   MMSE_std               220 non-null    float64
 10  ADAS13_std             220 non-null    float64
 11  FAQ_std                220 non-null    float64
 12  RAVLT_immediate_std    220 non-null    float64
 13  RAVLT_learning_std     220 non-null    float64
 14  RAVLT_forgetting_std   220 non-null    float64
 15  DIGITSCOR_std          220 non-null    float64
 16  TRABSCOR_std           220 non-null    float64
 17  Ventricles_std         220 non-null    float64
 18  Hippocampus_std        220 non-null    float64
 19  WholeBrain_std         220 non-null    float64
 20  Entorhinal_std         220 non-null    float64
 21  Fusiform_std           220 non-null    float64
 22  MidTemp_std            220 non-null    float64
 23  ICV_std                220 non-null    float64
dtypes: float64(18), int64(5), object(1)
memory usage: 43.0+ KB
```

```
df_tab.to_csv('../Data/TABULAR.csv', index=False)
```

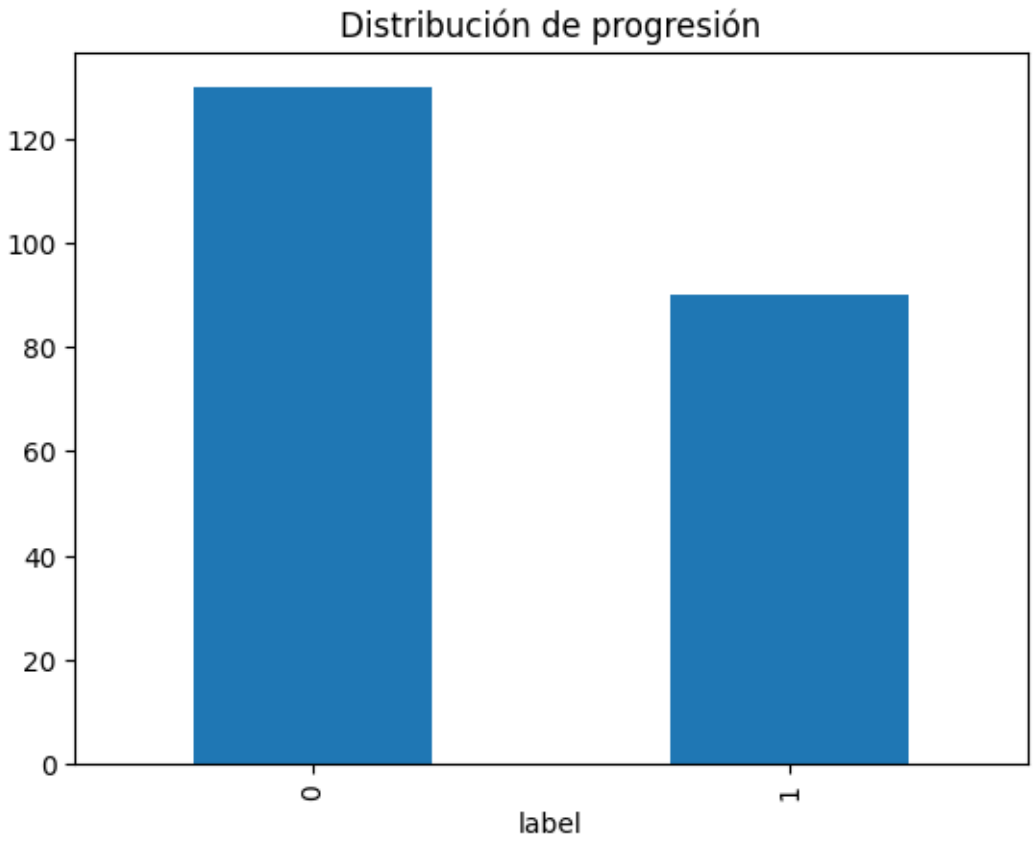
Distribución de clases: Progresión de casos

```
# [] Balance de clases
print("\n Distribución de la variable objetivo:")
display(df_tab["label"].value_counts(normalize=True).mul(100).round(2).to_frame())
df_tab["label"].value_counts().plot(kind="bar", title="Distribución de progresión")

# [Test] Z-test para comparar proporciones
counts = [df_tab["label"].value_counts()[1],
          df_tab["label"].value_counts()[0]]
nobs = [sum(counts), sum(counts)]
stat, pval = proportions_ztest(counts, nobs)
print(f"\nPrueba Z-test: Z = {stat:.2f}, p-value = {pval:.4f}")
```

```
 Distribución de la variable objetivo:
```

label	proportion
0	59.09
1	40.91

```
Prueba Z-test: Z = -3.81, p-value = 0.0001
```



2.3 *Estandarización de las imágenes*

Las imágenes fueron procesadas siguiendo un pipeline estandarizado orientado a:

- Reorientación al sistema anatómico RAS.
- Resampleo a una resolución isotrópica de 1.0 mm.

- Redimensionamiento a un shape uniforme de 160×192×192 voxeles.
- Normalización de intensidades mediante Z-score.
- Exportación a formato NumPy (.npy) para uso en modelos 3D.

```
# Medidas más comunes
display(out['orientation'].value_counts().to_frame())
display(out['shape'].value_counts().to_frame().head(3))
display(out['voxel_size'].value_counts().to_frame().head(3))
```

orientation	count
(R, A, S)	192
(P, S, R)	28

shape	count
(166, 256, 256)	79
(160, 192, 192)	77
(180, 256, 256)	25

voxel_size	count
(1.2, 0.9375, 0.9375)	97
(1.2, 1.25, 1.25)	75
(0.9375, 0.9375, 1.2)	27

```
import shutil

def preprocess_mri_folder(
        BASE_DIR  = Path(r"C:\Users\usuario\MRI\IMAGES_NII"),
        output_folder=r"C:\Users\usuario\MRI\IMAGES_npy",
        target_shape=(160,192,192),
        target_spacing=(1.0,1.0,1.0)
    ):

    if os.path.exists(output_folder):
```

```
        print("  Limpiando carpeta de salida...")
        shutil.rmtree(output_folder)
    os.makedirs(output_folder, exist_ok=True)

    # Encuentra TODOS los .nii y .nii.gz dentro de todas las carpetas
    nii_files = list(BASE_DIR.rglob("*.nii")) + list(BASE_DIR.rglob("*.nii.gz"))

    print(f"Total imágenes encontradas: {len(nii_files)}")

    for fname in tqdm(nii_files, desc="Procesando MRI", unit="img"):
        #print(f"\nProcesando: {fname}")

        # ----------------------
        # 1) Cargar imagen y orientar a RAS
        # ----------------------
        img = sitk.ReadImage(os.path.join(BASE_DIR, fname))
        img = sitk.DICOMOrient(img, "RAS")

        # ----------------------
        # 2) Resamplear a voxel 1.0 mm
        # ----------------------
        original_spacing = img.GetSpacing()
        original_size = img.GetSize()

        new_size = [
            int(round(original_size[i] * (original_spacing[i] / target_spacing[i])))
            for i in range(3)
        ]

        resampler = sitk.ResampleImageFilter()
        resampler.SetInterpolator(sitk.sitkLinear)
        resampler.SetOutputSpacing(target_spacing)
        resampler.SetSize(new_size)
        resampler.SetOutputDirection(img.GetDirection())
        resampler.SetOutputOrigin(img.GetOrigin())

        img_resampled = resampler.Execute(img)

        # Convertir a numpy
        arr = sitk.GetArrayFromImage(img_resampled)  # (D,H,W)

        # ----------------------
```

```
        # 3) Ajustar tamaño a (160,192,192)
        #    - Crop o pad automático
        # ----------------------
        def resize_to_shape(volume, target):
            out = np.zeros(target, dtype=volume.dtype)

            # cálculo de offsets
            d, h, w = volume.shape
            td, th, tw = target

            # límites
            d0 = max((td - d)//2, 0); d1 = d0 + min(d, td)
            h0 = max((th - h)//2, 0); h1 = h0 + min(h, th)
            w0 = max((tw - w)//2, 0); w1 = w0 + min(w, tw)

            vd0 = max((d - td)//2, 0); vd1 = vd0 + min(d, td)
            vh0 = max((h - th)//2, 0); vh1 = vh0 + min(h, th)
            vw0 = max((w - tw)//2, 0); vw1 = vw0 + min(w, tw)

            out[d0:d1, h0:h1, w0:w1] = volume[vd0:vd1, vh0:vh1, vw0:vw1]
            return out

        arr = resize_to_shape(arr, target_shape)

        # ----------------------
        # 4) Normalizar intensidad (z-score)
        # ----------------------
        arr = arr.astype(np.float32)
        m = np.mean(arr)
        s = np.std(arr) + 1e-6
        arr = (arr - m) / s

        # ----------------------
        # 5) Guardar como .npy
        # ----------------------

        subject_id = fname.parents[3].name
        image_id = fname.parents[0].name

        out_name = f"{subject_id}_{image_id}.npy"
        out_path = os.path.join(output_folder, out_name)
```

```
        np.save(out_path, arr)

        #print(f"  Guardado: {out_path}")

    print("\n  Procesamiento completado.")


# Ejecutar
preprocess_mri_folder()
```

```
  Limpiando carpeta de salida...
Total imágenes encontradas: 220

Procesando MRI: 100%|      | 220/220 [00:46<00:00,  4.70img/s]


  Procesamiento completado.
```

3 Análisis Exploratorio de Datos

Las imágenes utilizadas en este estudio provienen de la iniciativa **Alzheimer's Disease Neuroimaging Initiative (ADNI)**, un repositorio internacional de neuroimágenes longitudinales. Se seleccionaron exclusivamente imágenes estructurales T1-ponderadas (MRI) correspondientes a sujetos con diagnóstico deterioro cognitivo leve (MCI) y demencia tipo Alzheimer (AD).

El dataset tabular proviene de los datos clínicos y neuropsicológicos asociados a las mismas visitas de imagen, conteniendo información sociodemográfica, genética, diagnóstica y volumétrica (FreeSurfer).

3.1 DATOS TABULARES

```
# [Config] Rutas
df_tab = pd.read_csv("../Data/TABULAR.csv")
df_dx = pd.read_csv('../Data/ANDI_out.csv')
```

```
# Caracteristicas
print("CARACTERÍSTICAS\n")
print("======================"
      f"\n  Imágenes: {len(df_tab)}"
      f"\n  Cantidad de pacientes: {df_dx['sujeto_id'].nunique()}"
      f"\n  Vistas dispobibles: {df_dx['Visit'].nunique()}, en los meses {df_tab['Visita'].un
      "\n======================"
)

# Progeso de la enfermedad
labels_por_sujeto = df_tab.groupby("sujeto_id")["label"].first()
tabla_1 = pd.DataFrame({
    "conteo": labels_por_sujeto.value_counts(),
    "porcentaje": labels_por_sujeto.value_counts(normalize=True).mul(100).round(2)
})
tabla_1.index = tabla_1.index.map({0: "No progreso", 1: "Progreso"})
display(tabla_1.style.set_caption("Distribución del progreso de la enfermedad"))
```

```python
# Imágnes
tabla_2 = pd.DataFrame({
    "conteo": df_tab["is_dementia"].value_counts(),
    "porcentaje": df_tab["is_dementia"].value_counts(normalize=True).mul(100).round(2)
})
tabla_2.index = tabla_2.index.map({0: "MCI", 1: "Dementia"})
display(tabla_2.style.set_caption("Distribución de imágenes con Dementia"))

# Edad
print("======================"
      f"\n#  Edad de los pacientes:"
      f"\nMín: {df_tab['Age'].min()} | Máx: {df_tab['Age'].max()} | Mean: {df_tab['Age'].mean
      "\n======================"
)

# Sexo
sex = df_tab.groupby("sujeto_id")["Sexo"].first()
tabla_3 = pd.DataFrame({
    "conteo": sex.value_counts(),
    "porcentaje": sex.value_counts(normalize=True).mul(100).round(2)
})
tabla_3.index = tabla_3.index.map({0: "Masculino", 1: "Femenino"})
display(tabla_3.style.set_caption("Distribución del sexo"))

# Nivel educativo
educat = {
    10: "10 años (básica incompleta)",
    12: "12 años (secundaria completa)",
    13: "13 años (secundaria + 1 año)",
    14: "14 años (preuniversitario)",
    16: "16 años (licenciatura)",
    17: "17 años (posgrado parcial)",
    18: "18 años (maestría)",
    19: "19 años (posgrado incompleto)",
    20: "20 años (doctorado/profesional)"
}
tabla_4 = pd.DataFrame({
    "conteo": df_tab["Educat"].value_counts(),
    "porcentaje": df_tab["Educat"].value_counts(normalize=True).mul(100).round(2)
})
tabla_4.index = tabla_4.index.map(educat)
display(tabla_4.style.set_caption("Distribución de nivel educativo"))
```

```
CARACTERÍSTICAS

======================
  Imágenes: 220
   Cantidad de pacientes: 51
  Vistas dispobibles: 7, en los meses [ 6 12 18 24 36 48 60]
======================
```

Table 3.1: Distribución del progreso de la enfermedad

label	conteo	porcentaje
No progreso	31	60.780000
Progreso	20	39.220000

Table 3.2: Distribución de imágenes con Dementia

is_dementia	conteo	porcentaje
MCI	158	71.820000
Dementia	62	28.180000

```
======================
#  Edad de los pacientes:
Mín: 61 | Máx: 91 | Mean: 75.79
======================
```

Table 3.3: Distribución del sexo

Sexo	conteo	porcentaje
Masculino	38	74.510000
Femenino	13	25.490000

Table 3.4: Distribución de nivel educativo

Educat	conteo	porcentaje
16 años (licenciatura)	62	28.180000
18 años (maestría)	53	24.090000
20 años (doctorado/profesional)	31	14.090000
19 años (posgrado incompleto)	19	8.640000
12 años (secundaria completa)	17	7.730000
14 años (preuniversitario)	14	6.360000
10 años (básica incompleta)	13	5.910000
17 años (posgrado parcial)	6	2.730000
13 años (secundaria + 1 año)	5	2.270000

Variables Volumétricas

- APOE4 es una variante genética de la apolipoproteína E “CDRSB”, #“Suma de cajas del Clinical Dementia Rating (CDR); mide la severidad de la demencia.”,/ “MMSE”, #“Mini-Mental State Examination; evaluación global del estado cognitivo (máx. 30 puntos).”,/ “ADAS13”, #“Alzheimer’s Disease Assessment Scale – 13 ítems; mide deterioro cognitivo en Alzheimer.”, “FAQ”, #“Functional Activities Questionnaire; evalúa la capacidad funcional en actividades diarias.”, “RAVLT_immediate”, # “Puntuación inmediata en la prueba verbal de aprendizaje (Rey Auditory Verbal Learning Test).”, “RAVLT_learning”, # “Puntuación de aprendizaje acumulado en RAVLT; mide retención verbal.”, “RAVLT_forgetting”, # “Índice de olvido en RAVLT; diferencia entre aprendizaje y recuerdo tardío.”, “DIGITSCOR”, #“Digit Span Score; mide memoria de trabajo y atención mediante secuencias numéricas.”, “TRABSCOR”, # “Trail Making Test B Score; evalúa función ejecutiva y flexibilidad cognitiva.”,] volumen = [“Ventricles”, # “Volumen de los ventrículos cerebrales; puede indicar atrofia cerebral.”, “Hippocampus”, # “Volumen del hipocampo; clave en memoria y afectado en Alzheimer.”, “WholeBrain”, # “Volumen total del cerebro; útil para evaluar atrofia global.”, “Entorhinal”, # “Volumen de la corteza entorrinal; región afectada tempranamente en Alzheimer.”, “Fusiform”, #“Volumen del giro fusiforme; relacionado con reconocimiento visual.”, “MidTemp”, # “Volumen del lóbulo temporal medio; implicado en memoria y procesamiento auditivo.”, “ICV”, #“Volumen intracraneal total; usado para normalizar medidas volumétricas.”]

```
df_tab.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220 entries, 0 to 219
Data columns (total 24 columns):
```

```
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   sujeto_id             220 non-null    object
 1   label                 220 non-null    int64
 2   is_dementia           220 non-null    int64
 3   Visita                220 non-null    int64
 4   Age                   220 non-null    int64
 5   Sexo                  220 non-null    int64
 6   Educat                220 non-null    float64
 7   APOE4_std             220 non-null    float64
 8   CDRSB_std             220 non-null    float64
 9   MMSE_std              220 non-null    float64
 10  ADAS13_std            220 non-null    float64
 11  FAQ_std               220 non-null    float64
 12  RAVLT_immediate_std   220 non-null    float64
 13  RAVLT_learning_std    220 non-null    float64
 14  RAVLT_forgetting_std  220 non-null    float64
 15  DIGITSCOR_std         220 non-null    float64
 16  TRABSCOR_std          220 non-null    float64
 17  Ventricles_std        220 non-null    float64
 18  Hippocampus_std       220 non-null    float64
 19  WholeBrain_std        220 non-null    float64
 20  Entorhinal_std        220 non-null    float64
 21  Fusiform_std          220 non-null    float64
 22  MidTemp_std           220 non-null    float64
 23  ICV_std               220 non-null    float64
dtypes: float64(18), int64(5), object(1)
memory usage: 41.4+ KB
```

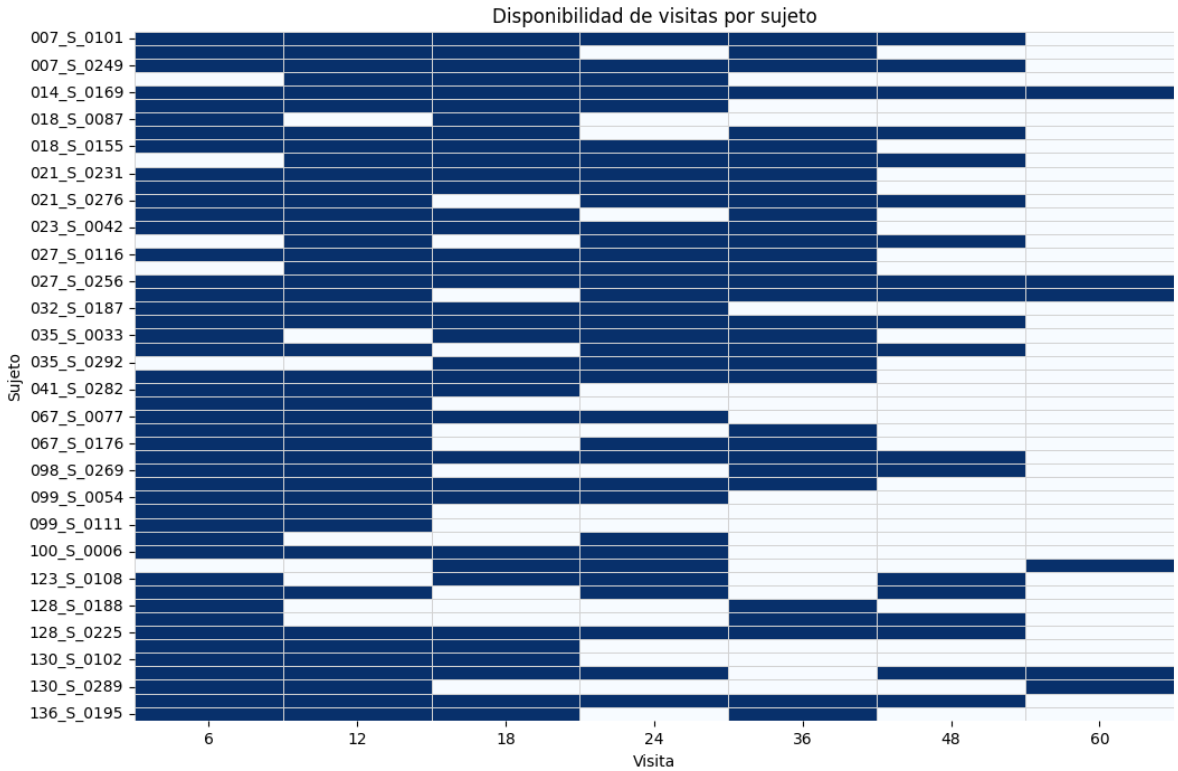
```
df_tab.columns
```

```
Index(['sujeto_id', 'label', 'is_dementia', 'Visita', 'Age', 'Sexo', 'Educat',
       'APOE4_std', 'CDRSB_std', 'MMSE_std', 'ADAS13_std', 'FAQ_std',
       'RAVLT_immediate_std', 'RAVLT_learning_std', 'RAVLT_forgetting_std',
       'DIGITSCOR_std', 'TRABSCOR_std', 'Ventricles_std', 'Hippocampus_std',
       'WholeBrain_std', 'Entorhinal_std', 'Fusiform_std', 'MidTemp_std',
       'ICV_std'],
      dtype='object')
```

3.1.1 Diagnóstico

- Disponibilidad de visitas por sujeto

```
# Crear tabla de presencia de visitas por sujeto
# Asumiendo df tiene columnas: subject_id, Visita
df_visitas = df_tab.copy()
df_visitas['Presente'] = 1  # Marca cada fila como "visita existente"

tabla_visitas = df_visitas.pivot_table(
    index='sujeto_id',
    columns='Visita',
    values='Presente',
    aggfunc='last',
    fill_value=0
)
# Graficar heatmap
plt.figure(figsize=(12,8))
sns.heatmap(tabla_visitas, cmap="Blues", cbar=False, linewidths=0.5, linecolor='lightgray')
plt.title("Disponibilidad de visitas por sujeto")
plt.xlabel("Visita")
plt.ylabel("Sujeto")
plt.show()
```

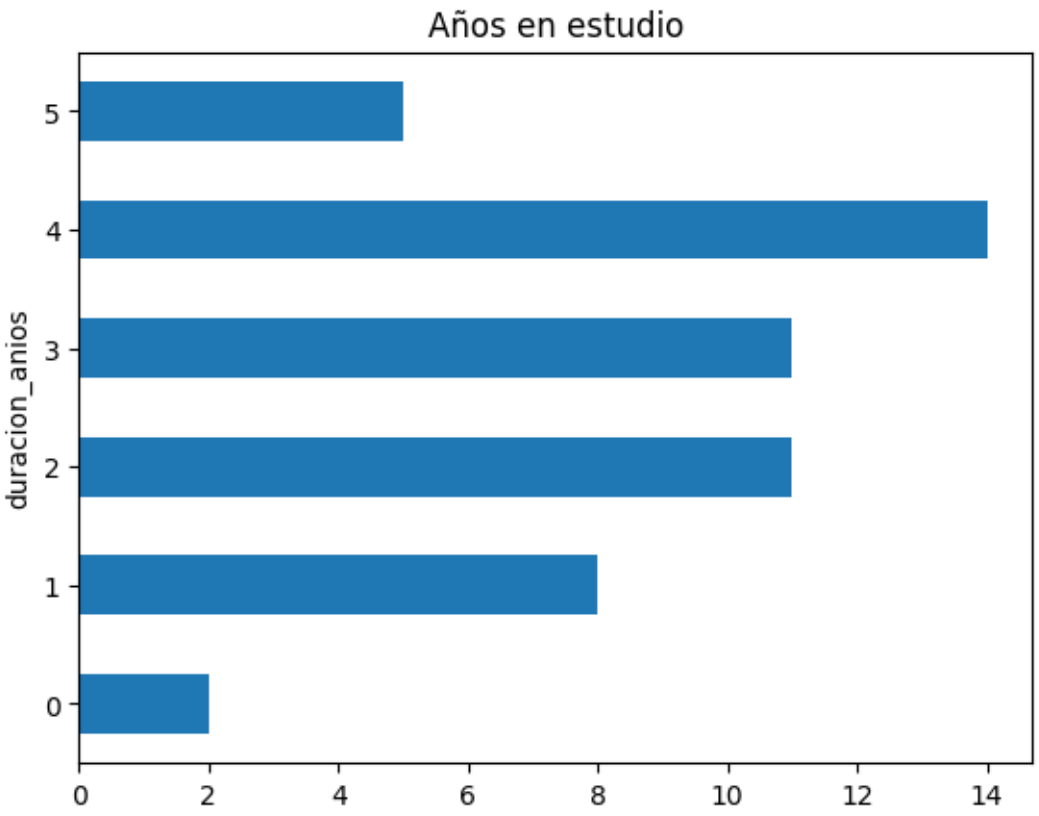


- Años en estudio

```
df_dx["fecha_imagen"] = pd.to_datetime(df_dx["fecha_imagen"])
df_dx["año"] = df_dx["fecha_imagen"].dt.year
df_duracion = df_dx.sort_values(["sujeto_id", "fecha_imagen"]).groupby("sujeto_id").agg(
    año_inicio=("año", "first"),
    año_final=("año", "last")
).reset_index()

df_duracion["duracion_anios"] = df_duracion["año_final"] - df_duracion["año_inicio"]
display(df_duracion["duracion_anios"].value_counts().sort_index().to_frame())

df_duracion["duracion_anios"].value_counts().sort_index().plot.barh()
plt.title("Años en estudio")
```

duracion_anios	count
0	2
1	8
2	11
3	11
4	14
5	5

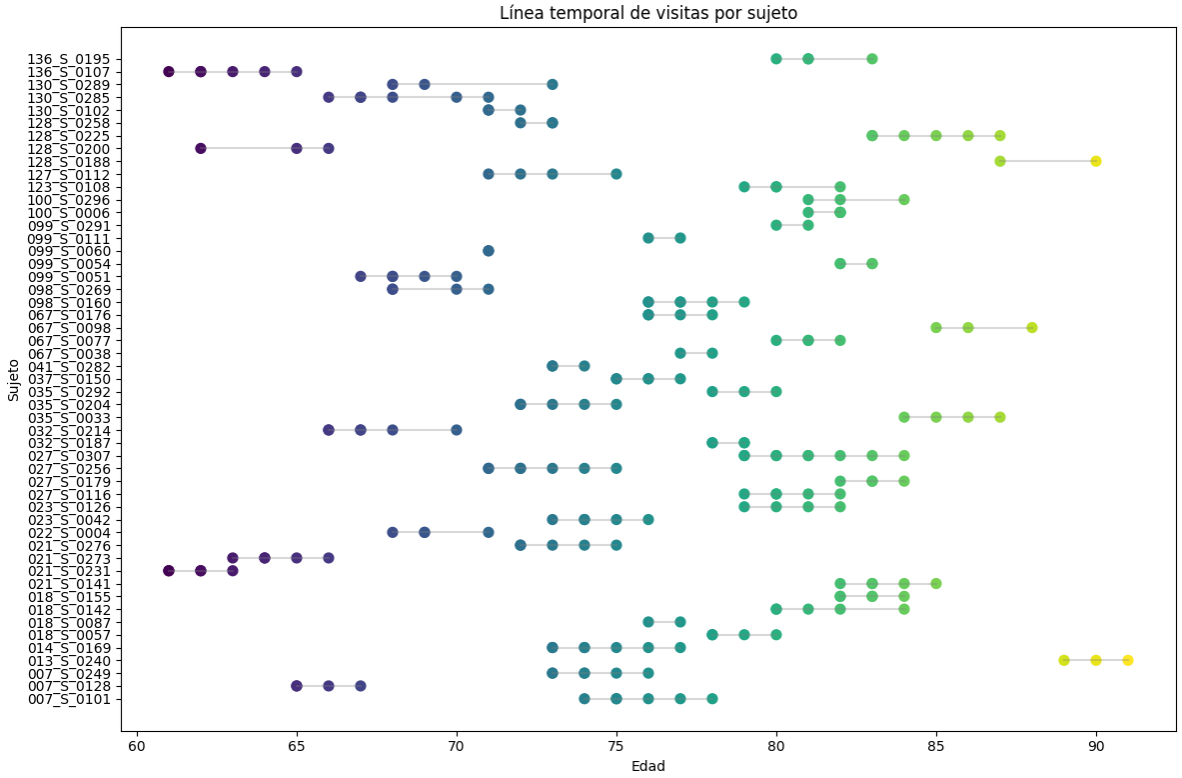
```
Text(0.5, 1.0, 'Años en estudio')
```



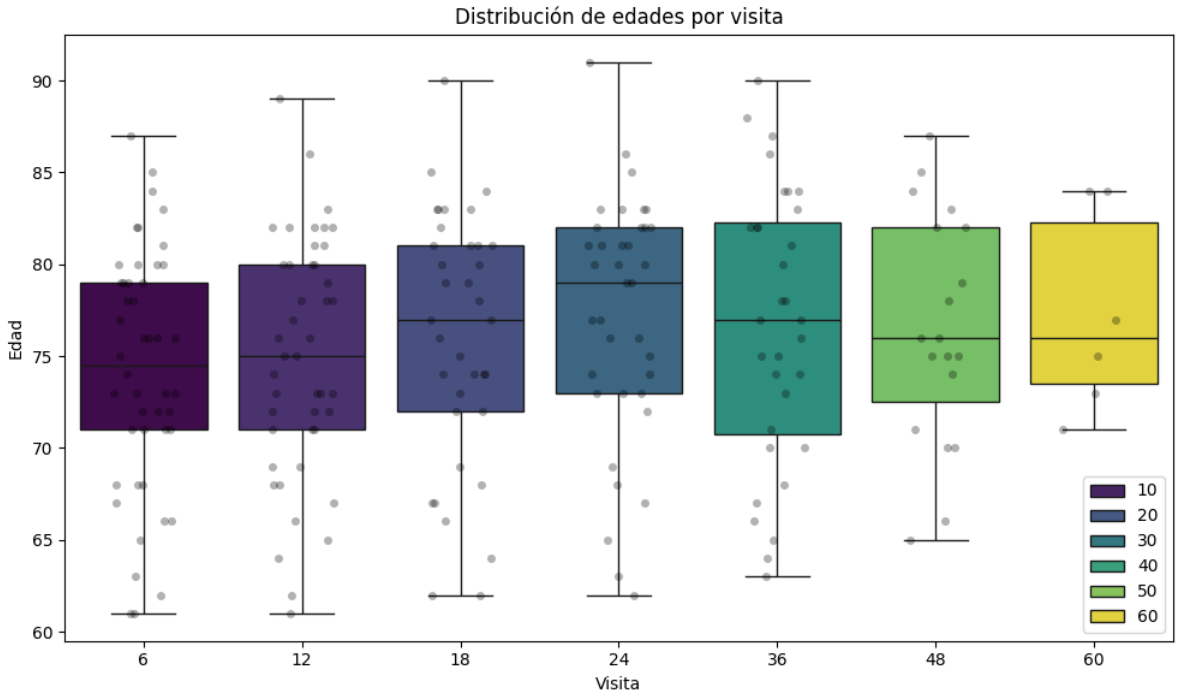
- Distribución de edad por visita

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

plt.figure(figsize=(12,8))

# Colormap continuo
colormap = plt.get_cmap("viridis")

# Normalizamos edades entre 0 y 1 para el colormap
ages = df_tab['Age']
norm = plt.Normalize(vmin=ages.min(), vmax=ages.max())

for sid, grupo in df_tab.groupby('sujeto_id'):
    color_values = colormap(norm(grupo['Age']))  # color por edad
    plt.scatter(grupo['Age'], [sid]*len(grupo), c=color_values, s=50)  # puntos
    plt.plot(grupo['Age'], [sid]*len(grupo), color='gray', alpha=0.3)     # línea base
```

```
plt.xlabel("Edad")
plt.ylabel("Sujeto")
plt.title("Línea temporal de visitas por sujeto")
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(10,6))
sns.boxplot(data=df_tab, x='Visita', y='Age', palette='viridis', hue='Visita')
sns.stripplot(data=df_tab, x='Visita', y='Age', color='black', alpha=0.3, jitter=0.2)  # opc

plt.title("Distribución de edades por visita")
plt.xlabel("Visita")
plt.ylabel("Edad")
plt.tight_layout()
plt.show()
```




```
df_tab.columns
```

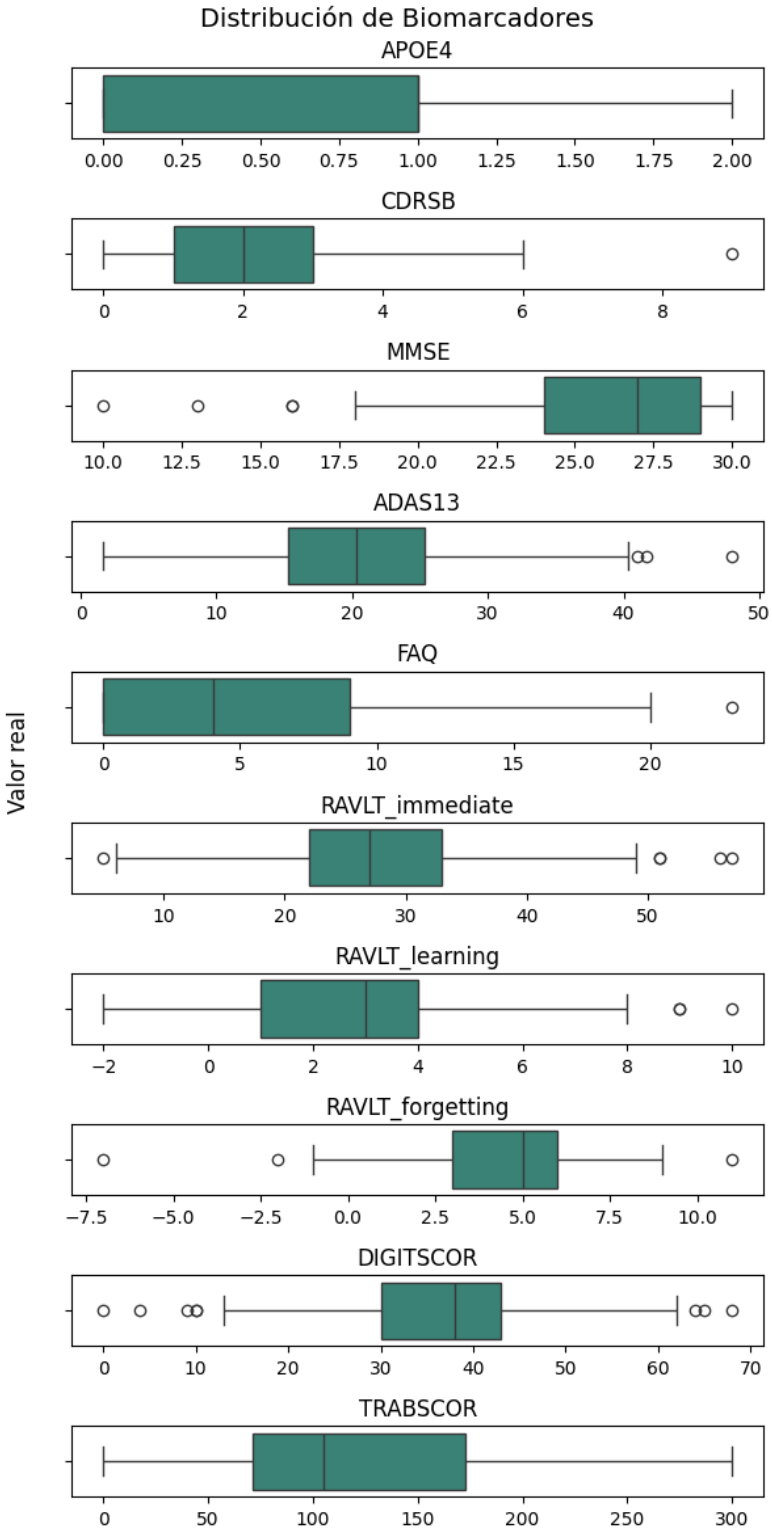
```
Index(['sujeto_id', 'label', 'is_dementia', 'Visita', 'Age', 'Sexo', 'Educat',
       'APOE4_std', 'CDRSB_std', 'MMSE_std', 'ADAS13_std', 'FAQ_std',
       'RAVLT_immediate_std', 'RAVLT_learning_std', 'RAVLT_forgetting_std',
       'DIGITSCOR_std', 'TRABSCOR_std', 'Ventricles_std', 'Hippocampus_std',
       'WholeBrain_std', 'Entorhinal_std', 'Fusiform_std', 'MidTemp_std',
       'ICV_std'],
      dtype='object')
```

3.1.2 Biomarcadores

```
biomarcadores = ['APOE4', 'CDRSB', 'MMSE', 'ADAS13', 'FAQ',
       'RAVLT_immediate', 'RAVLT_learning', 'RAVLT_forgetting', 'DIGITSCOR',
       'TRABSCOR']
```

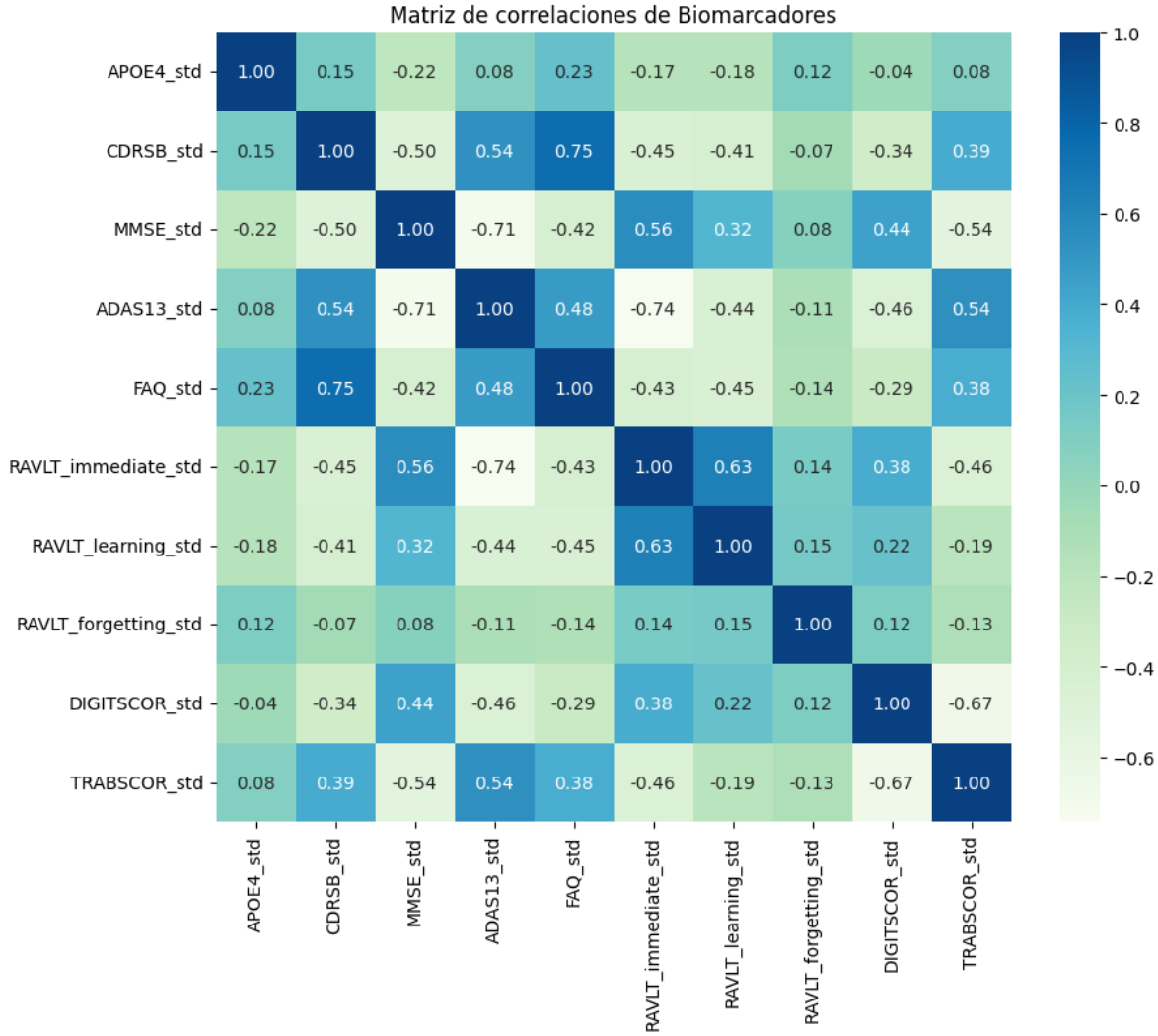
```
plt.figure(figsize=(6, 12))

for i, biom in enumerate(biomarcadores, 1):
```

```
    plt.subplot(len(biomarcadores), 1, i)
    sns.boxplot(x=df_dx[biom], color='#2E8E7E')
    plt.title(biom)
    plt.xlabel("")
plt.suptitle("Distribución de Biomarcadores", fontsize=14)
plt.figtext(0.02, 0.5, "Valor real", ha='center', va='center', rotation='vertical', fontsize=
plt.tight_layout(rect=[0.05, 0, 1, 1])  # deja espacio para el ylabel global
plt.show()
plt.show()
```


Distribución de Biomarcadores
APOE4
CDRSB
MMSE
ADAS13
FAQ
RAVLT_immediate
RAVLT_learning
RAVLT_forgetting
DIGITSCOR
TRABSCOR
Valor real

```
bio_std = ['APOE4_std', 'CDRSB_std', 'MMSE_std', 'ADAS13_std', 'FAQ_std',
       'RAVLT_immediate_std', 'RAVLT_learning_std', 'RAVLT_forgetting_std',
       'DIGITSCOR_std', 'TRABSCOR_std']
```

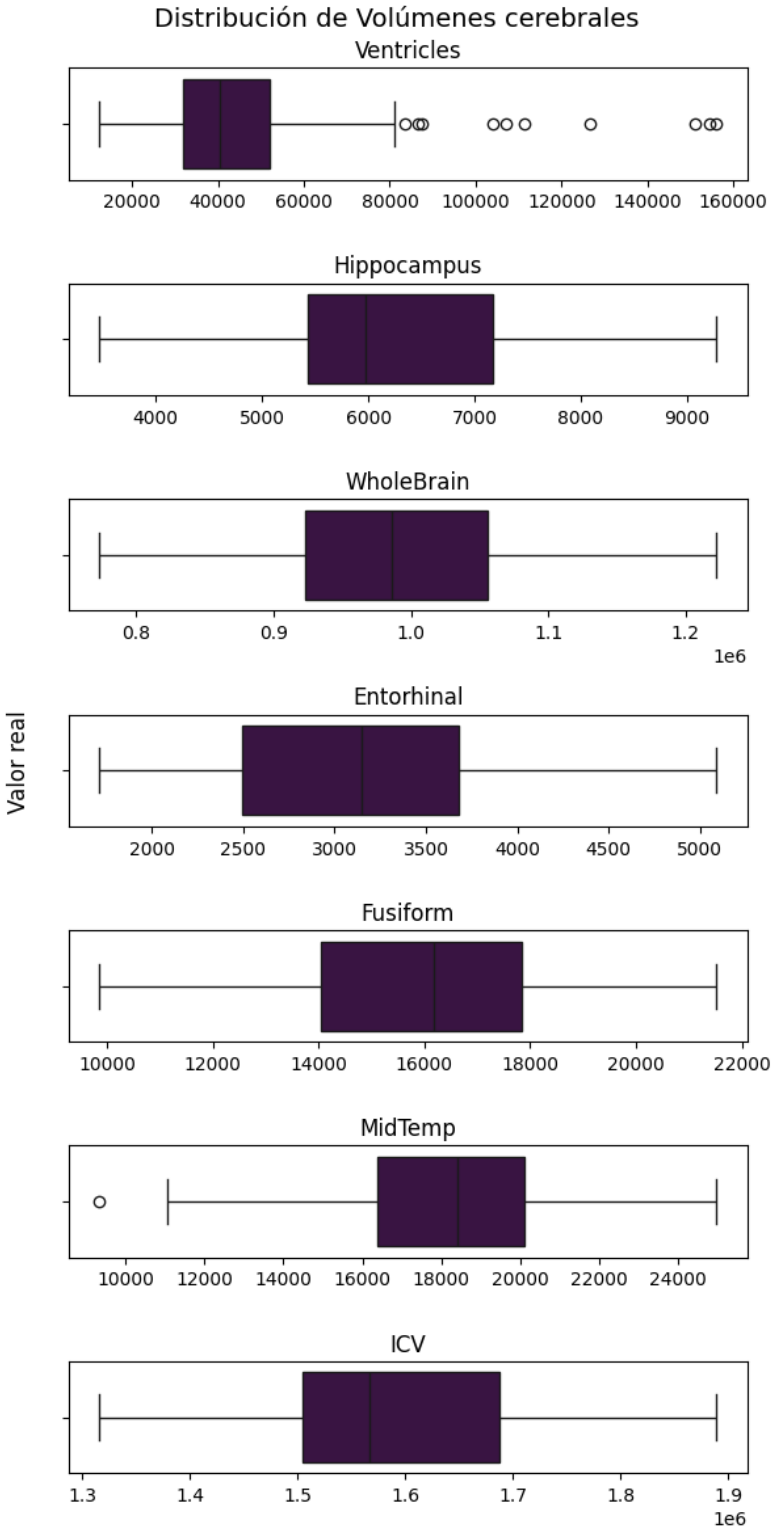
```
plt.figure(figsize=(10,8))
sns.heatmap(df_tab[bio_std].corr(), annot=True, fmt=".2f", cmap="GnBu")
plt.title("Matriz de correlaciones de Biomarcadores")
plt.show()
```



3.1.3 Volúmenes cerbrales

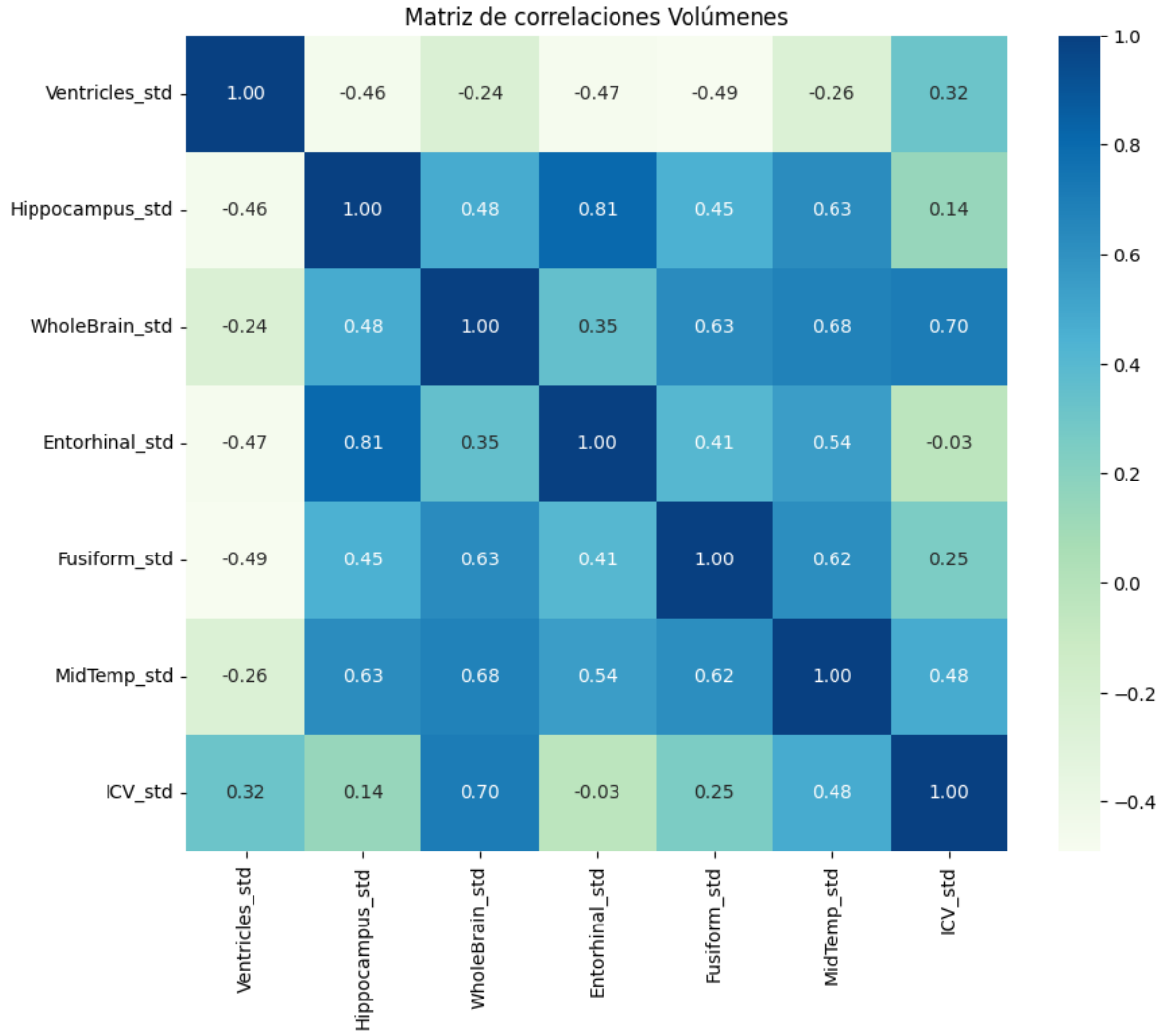
```
volumen = ['Ventricles', 'Hippocampus', 'WholeBrain', 'Entorhinal',
       'Fusiform', 'MidTemp', 'ICV']
```

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

for i, biom in enumerate(volumen, 1):
    plt.subplot(len(volumen), 1, i)
    sns.boxplot(x=df_dx[biom], color='#3E0C4A')
    plt.title(biom)
    plt.xlabel("")
plt.suptitle("Distribución de Volúmenes cerebrales", fontsize=14)
plt.figtext(0.02, 0.5, "Valor real", ha='center', va='center', rotation='vertical', fontsize=
plt.tight_layout(rect=[0.05, 0, 1, 1])  # deja espacio para el ylabel global
plt.show()
plt.show()
```


Distribución de Volúmenes cerebrales
Ventricles
20000
40000
60000
80000
100000
120000
140000
160000
Hippocampus
4000
5000
6000
7000
8000
9000
WholeBrain
0.8
0.9
1.0
1.1
1.2
1e6
Entorhinal
Valor real
2000
2500
3000
3500
4000
4500
5000
Fusiform
10000
12000
14000
16000
18000
20000
22000
MidTemp
10000
12000
14000
16000
18000
20000
22000
24000
ICV
1.3
1.4
1.5
1.6
1.7
1.8
1.9
1e6

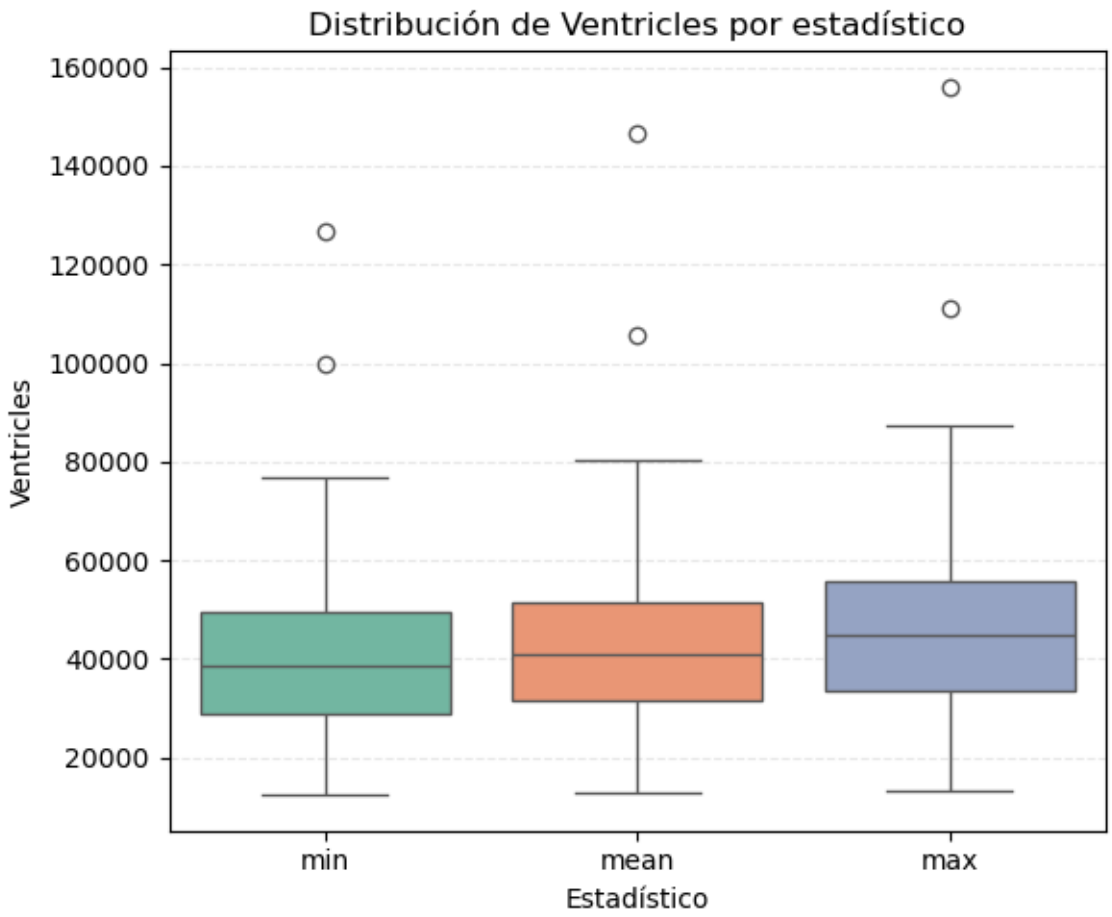
```
volumn_std =[
    'Ventricles_std', 'Hippocampus_std',
       'WholeBrain_std', 'Entorhinal_std', 'Fusiform_std', 'MidTemp_std',
       'ICV_std'
]
```

```
plt.figure(figsize=(10,8))
sns.heatmap(df_tab[volumn_std].corr(), annot=True, fmt=".2f", cmap="GnBu")
plt.title("Matriz de correlaciones Volúmenes")
plt.show()
```

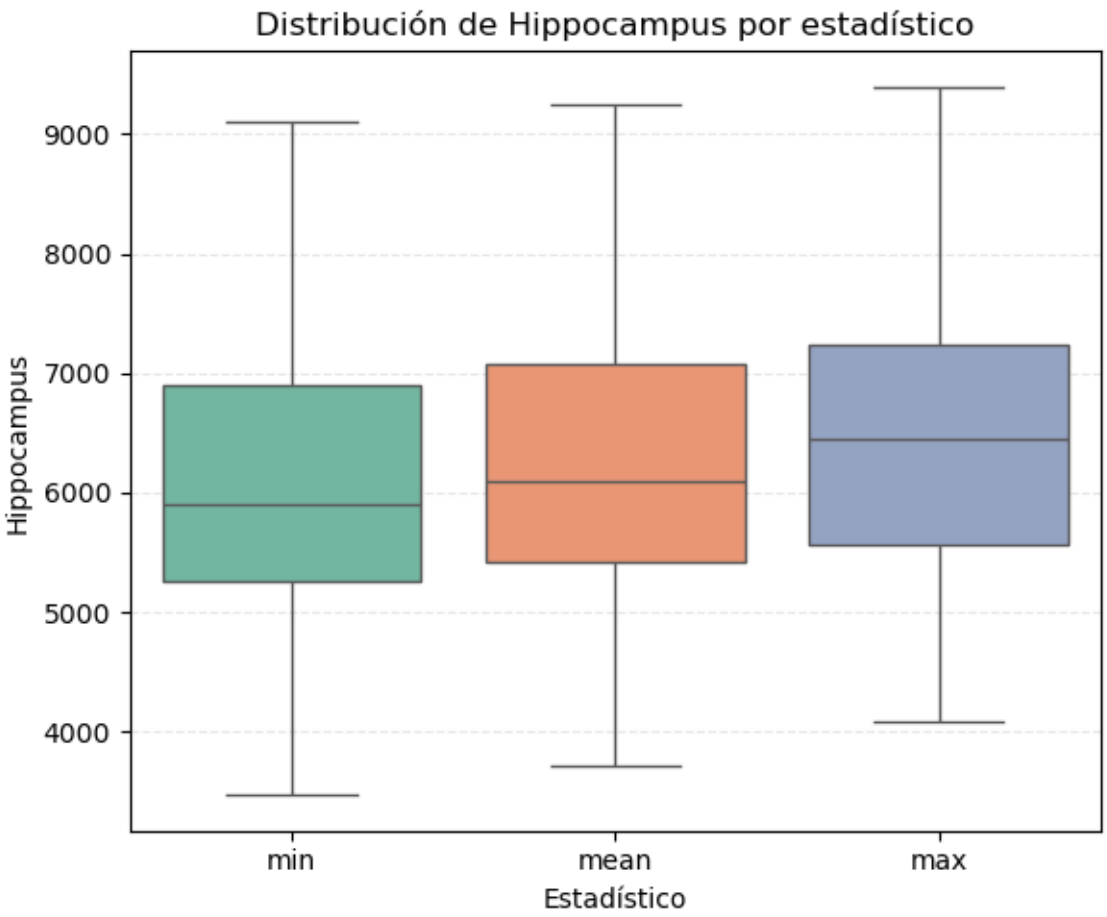


```python
# Aplanar columnas
info_volumen.columns = ['sujeto_id'] + [f"{var}_{stat}" for var, stat in info_volumen.columns

# Convertir a formato largo
info_long = pd.melt(
    info_volumen,
    id_vars="sujeto_id",
    var_name="variable_estadistica",
    value_name="valor"
)

# Separar nombre de variable y tipo de estadístico
info_long[["variable", "estadistica"]] = info_long["variable_estadistica"].str.rsplit("_", n=
```

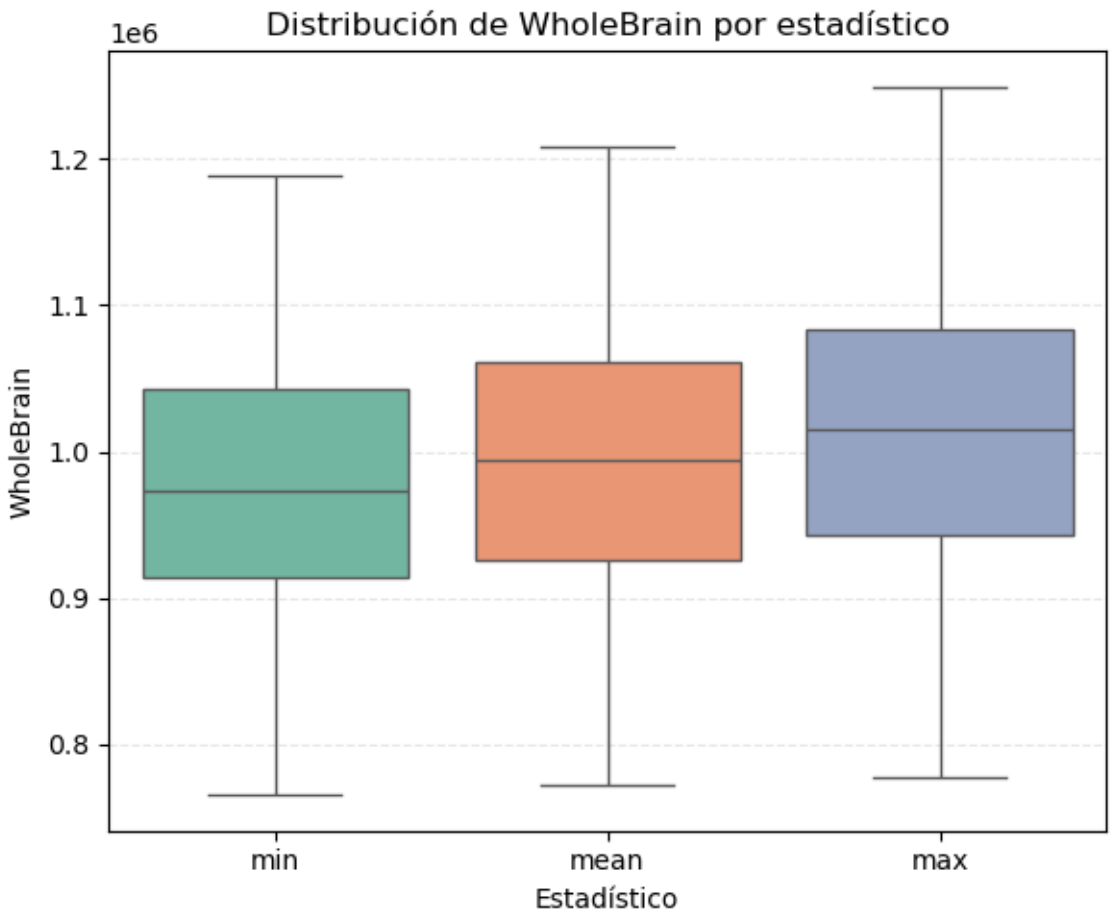
```
C:\Users\Hp\AppData\Local\Temp\ipykernel_34204\1002387156.py:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig

  sns.boxplot(
```



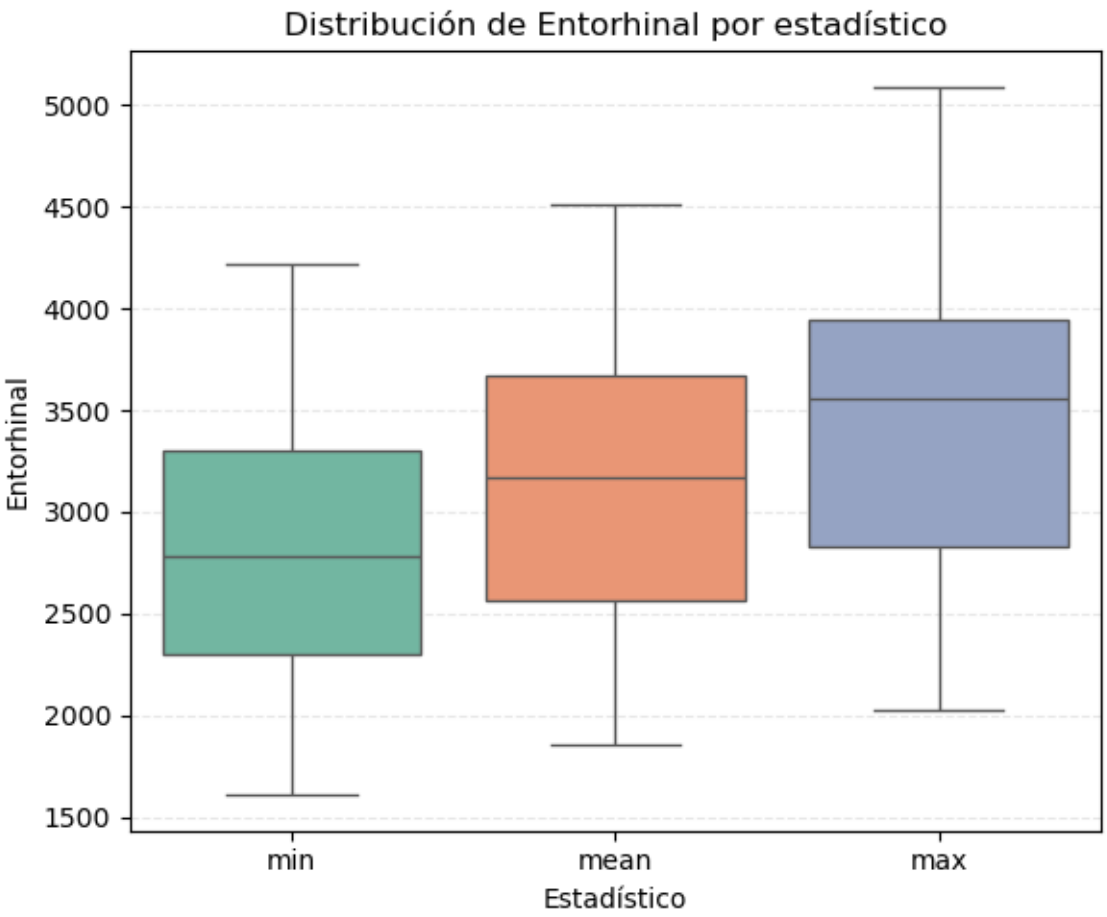

```
C:\Users\Hp\AppData\Local\Temp\ipykernel_34204\1002387156.py:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig

  sns.boxplot(
```



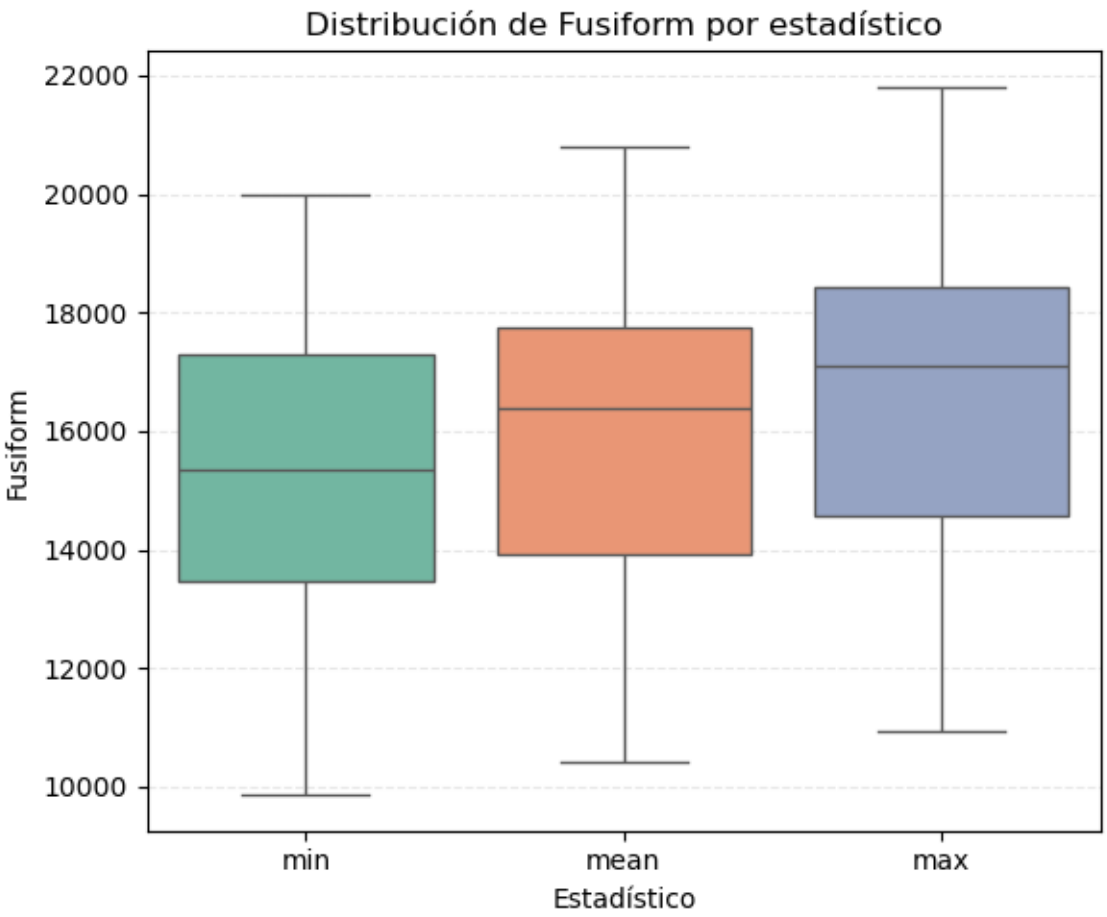

```
C:\Users\Hp\AppData\Local\Temp\ipykernel_34204\1002387156.py:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig

  sns.boxplot(
```



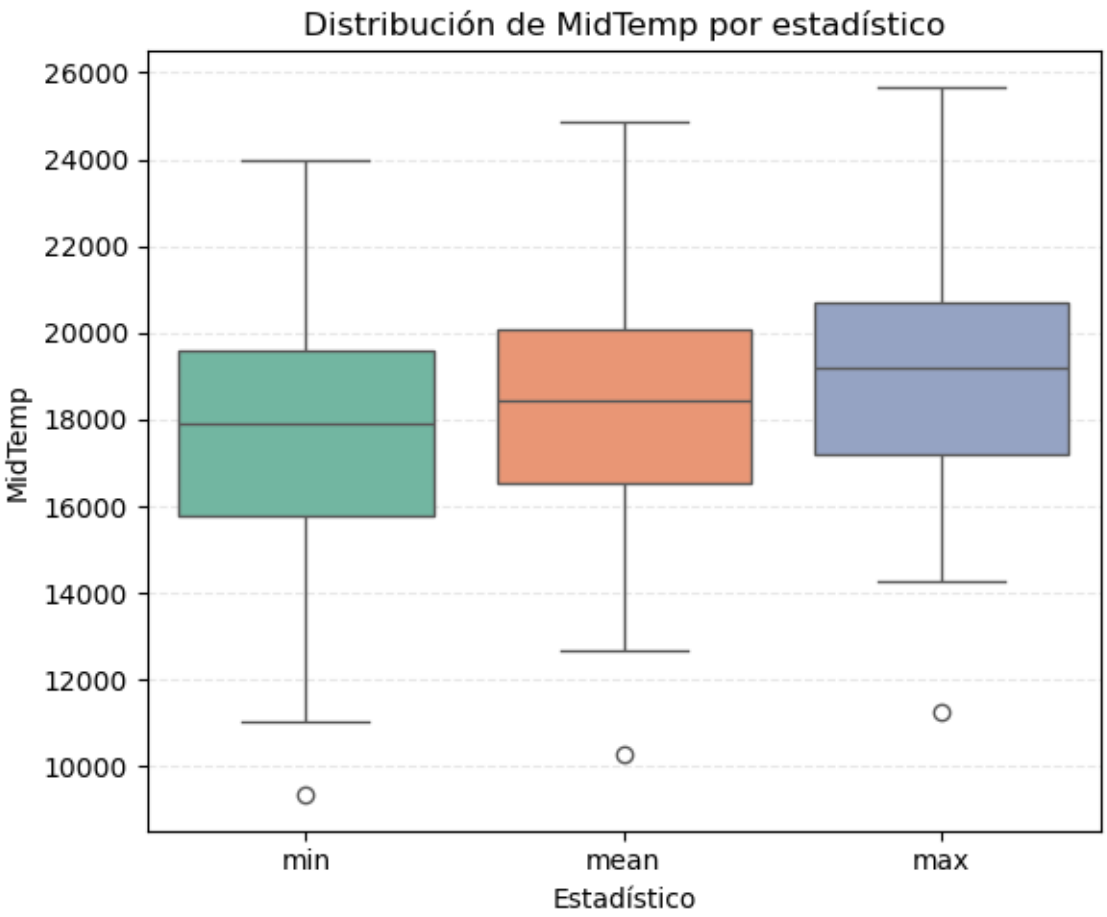

```
C:\Users\Hp\AppData\Local\Temp\ipykernel_34204\1002387156.py:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig

  sns.boxplot(
```



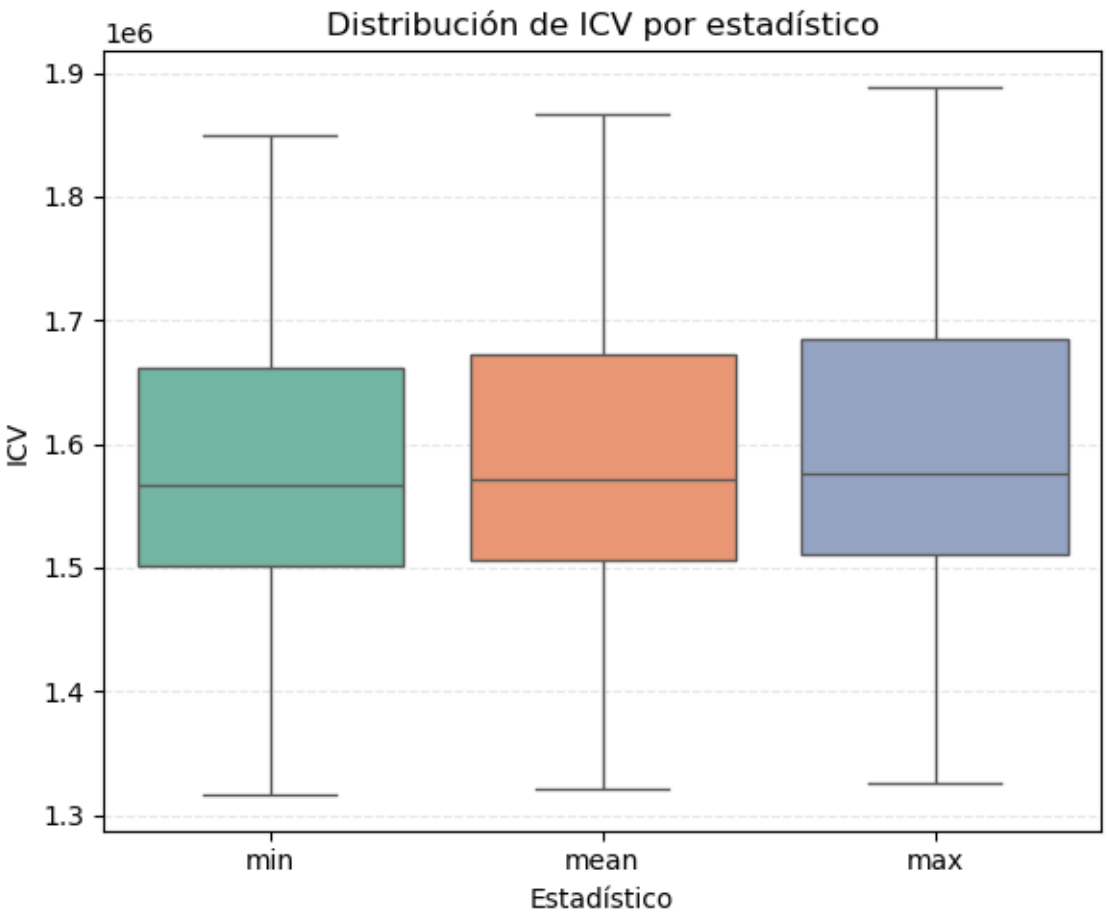

```
C:\Users\Hp\AppData\Local\Temp\ipykernel_34204\1002387156.py:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig

  sns.boxplot(
```




```
C:\Users\Hp\AppData\Local\Temp\ipykernel_34204\1002387156.py:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig

  sns.boxplot(
```




```
C:\Users\Hp\AppData\Local\Temp\ipykernel_34204\1002387156.py:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig

  sns.boxplot(
```




```
# Agrupar por variable y estadístico
tabla_resumen = info_long.groupby(["variable", "estadistica"])["valor"].agg(["min", "mean", "
# Ordenar por variable y tipo de estadístico
tabla_resumen = tabla_resumen.sort_values(["variable", "estadistica"])

# Mostrar
tabla_resumen
```

	variable	estadistica	min	mean	max
0	Entorhinal	max	2023.00	3481.71	5090.0
1	Entorhinal	mean	1853.00	3157.49	4511.2
2	Entorhinal	min	1608.00	2832.86	4215.0
3	Fusiform	max	10946.00	16742.86	21808.0
4	Fusiform	mean	10424.33	16061.54	20798.8

	variable	estadistica	min	mean	max
5	Fusiform	min	9860.00	15365.92	19972.0
6	Hippocampus	max	4087.00	6431.73	9396.0
7	Hippocampus	mean	3717.67	6205.67	9251.5
8	Hippocampus	min	3471.00	5997.59	9102.0
9	ICV	max	1325720.00	1595814.31	1888900.0
10	ICV	mean	1320936.67	1585118.96	1867338.0
11	ICV	min	1315940.00	1575091.76	1850260.0
12	MidTemp	max	11241.00	18991.55	25678.0
13	MidTemp	mean	10291.00	18319.15	24870.0
14	MidTemp	min	9341.00	17627.04	23970.0
15	Ventricles	max	13209.00	48518.61	156066.0
16	Ventricles	mean	12704.00	45054.87	146665.4
17	Ventricles	min	12346.00	41663.00	126585.0
18	WholeBrain	max	777166.00	1012856.61	1249300.0
19	WholeBrain	mean	771940.33	994606.37	1208020.0
20	WholeBrain	min	765255.00	976903.06	1188380.0

Las variables de volumen derivadas de imágenes de resonancia magnética permiten cuantificar la estructura anatómica del cerebro y detectar cambios asociados con la atrofia y el deterioro cognitivo.

En general, los **volúmenes medios** observados muestran una distribución coherente con las variaciones esperadas entre sujetos y posibles etapas de deterioro. El **volumen intracraneal total** (**ICV**) presenta valores promedio cercanos a **1.59 millones de mm³**, siendo una medida anatómica estable que se utiliza para normalizar otras variables de volumen.

El **volumen total cerebral** (**WholeBrain**) muestra valores promedio alrededor del **millón de mm³**, mientras que los **ventrículos** (**Ventricles**) presentan una alta variabilidad (media 47.000 mm^3), lo cual es característico de la expansión ventricular asociada a procesos de atrofia cortical.

Regiones críticas vinculadas con la memoria, como el **hipocampo** (**Hippocampus**) y la **corteza entorrinal** (**Entorhinal**), presentan volúmenes medios de aproximadamente **6.200 mm³** y **3.100 mm³**, respectivamente. La reducción en estas áreas se considera uno de los primeros indicadores estructurales del Alzheimer.

Por otro lado, estructuras del lóbulo temporal, como el **giro fusiforme** (**Fusiform**) y el **lóbulo temporal medio** (**MidTemp**), presentan volúmenes intermedios (entre **15.000 y 18.000 mm³**), reflejando su participación en procesos de reconocimiento visual y auditivo.

En conjunto, los resultados muestran una tendencia esperada de atrofia progresiva, principalmente en regiones temporales y de memoria, mientras que **ICV** y **WholeBrain** actúan como referencias anatómicas estables para la comparación entre sujetos y sesiones.

3.2 6. Evolución del Diagnostico por Visita

```
tabla_dx = df_dx.pivot_table(
    index="sujeto_id",
    columns="Visit",
    values="DX",
    aggfunc="first"  # Asume que hay una sola entrada por sujeto-visita
).fillna("-")
tabla_dx
```

Visit sujeto_id	m06	m12	m18	m24	m36
007_S_0101	MCI	MCI	MCI	Dementia	Dementia
007_S_0128	MCI	MCI	Dementia	-	-
007_S_0249	MCI	Dementia	Dementia	Dementia	Dementia
013_S_0240	-	MCI	Dementia	-	-
014_S_0169	MCI	MCI	MCI	MCI	MCI
018_S_0057	MCI	MCI	Dementia	Dementia	-
018_S_0087	MCI	-	MCI	-	-
018_S_0142	MCI	MCI	MCI	-	-
018_S_0155	MCI	MCI	MCI	MCI	-
021_S_0141	-	Dementia	Dementia	Dementia	-
021_S_0231	MCI	MCI	MCI	Dementia	Dementia
021_S_0273	MCI	MCI	MCI	MCI	-
021_S_0276	MCI	MCI	-	MCI	MCI
022_S_0004	MCI	MCI	MCI	-	-
023_S_0042	MCI	Dementia	Dementia	Dementia	Dementia
023_S_0126	-	MCI	-	MCI	Dementia
027_S_0116	MCI	MCI	MCI	MCI	MCI
027_S_0179	-	MCI	Dementia	-	-
027_S_0256	Dementia	Dementia	Dementia	Dementia	Dementia
027_S_0307	MCI	MCI	-	MCI	MCI
032_S_0187	MCI	MCI	MCI	Dementia	-
032_S_0214	MCI	MCI	MCI	Dementia	Dementia
035_S_0033	MCI	-	MCI	MCI	MCI
035_S_0204	MCI	Dementia	-	Dementia	-
035_S_0292	-	-	MCI	-	-
037_S_0150	MCI	MCI	MCI	MCI	-
041_S_0282	MCI	MCI	MCI	-	-
067_S_0038	MCI	MCI	-	-	-

Visit sujeto_id	m06	m12	m18	m24	m36
067_S_0077	MCI	Dementia	Dementia	Dementia	-
067_S_0098	MCI	MCI	-	-	-
067_S_0176	MCI	MCI	-	MCI	-
098_S_0160	MCI	MCI	MCI	MCI	-
098_S_0269	Dementia	Dementia	-	-	-
099_S_0051	MCI	MCI	MCI	MCI	-
099_S_0054	MCI	Dementia	Dementia	Dementia	-
099_S_0060	MCI	MCI	-	-	-
099_S_0111	MCI	Dementia	-	-	-
099_S_0291	MCI	-	-	-	-
100_S_0006	MCI	MCI	MCI	MCI	-
100_S_0296	-	-	MCI	MCI	-
123_S_0108	MCI	-	Dementia	Dementia	-
128_S_0188	MCI	-	-	-	-
128_S_0200	MCI	-	-	-	-
128_S_0225	MCI	MCI	MCI	MCI	MCI
128_S_0258	MCI	MCI	MCI	-	-
130_S_0102	MCI	MCI	MCI	-	-
130_S_0285	MCI	MCI	MCI	MCI	-
130_S_0289	MCI	MCI	-	-	-
136_S_0107	MCI	MCI	MCI	MCI	MCI
136_S_0195	MCI	Dementia	Dementia	-	-

3.2.1 Diagnostico Inicial vrs Final

```
df_a = df_dx.sort_values(["sujeto_id", "Visit"]).groupby("sujeto_id").agg(
    dx_inicio=("DX", "first"),
    dx_final=("DX", "last")
).reset_index()
tabla_transicion = df_a.groupby(["dx_inicio", "dx_final"]).size().unstack(fill_value=0)
tabla_transicion
```

dx_final dx_inicio	Dementia	MCI
Dementia	3	0
MCI	17	30

El diagnóstico inicial muestra que todos los participantes comenzaron el estudio con **deterioro cognitivo leve (MCI)**. Sin embargo, al finalizar el seguimiento, **22 de ellos progresaron a demencia**, mientras que **29 se mantuvieron en la misma condición**. Esto refleja una tendencia esperada en la evolución clínica del MCI, donde una proporción significativa de pacientes presenta un empeoramiento cognitivo a lo largo del tiempo, mientras que otros permanecen estables.
