Youth Science

SCIENCE FAIR
WHERE SCIENCE MEETS INDUSTRY

Final Report

NeuroData: Model for Alzheimer's Diagnosis

Fernando Patricio Gutiérrez González

Norma Estefanía Murillo Guajardo

Enrique Uriel Aguilar Flores

Sofía González Reyna

Mario Alberto Landa Flores

21/05/2024

Monterrey, Nuevo León , México

Abstract

This project presents a machine learning model designed to aid healthcare professionals in the diagnostic process of Alzheimer's disease. The model analyzes electroencephalograms (EEGs) and classifies them to determine whether the EEG belongs to a person with Alzheimer's, a healthy individual, or someone with another condition. We developed a MATLAB code utilizing the EEGLAB library to preprocess the EEG data. Subsequently, we implemented a Python code that loads the data, extracts valuable features from the EEGs, and ultimately develops a neural network model to categorize the EEGs. This project aims to streamline the diagnosis of Alzheimer's disease, potentially reducing the need for extensive medical testing. Our results indicate that the model can effectively distinguish between different types of EEGs, offering a promising tool for early and accurate diagnosis of Alzheimer's, thus facilitating timely intervention and treatment. This model highlights the potential of integrating machine learning techniques with medical diagnostics to enhance the efficiency and accuracy of disease identification.

Background/Introduction

Alzheimer’s disease is a constantly damaging neurodegenerative disorder that affects elderly people and causes symptoms such as memory loss, behavioral changes, and cognitive decline. Alzheimer's disease patients progressively develop dementia, another neurocognitive disorder, for every 3 or 4 out of 5 patients. The pathology of Alzheimer's includes an increase in the levels of amyloid-beta plaques and neurofibrillary tangles composed of tau proteins; such increments lead to neuronal damage and brain atrophy (Brody H., 2011).

Electroencephalogram (EEG) tests are performed on patients who require a diagnosis of the current health of the encephalic body. Such tests are non-invasive and measure electrical activity in the brain where electrodes are placed on the scalp of the patients. Electroencephalograms capture brain wave patterns and obtain real-time data on neural oscillations. The electrical activity recorded reflects the synchronous firing of neurons; different frequency bands are associated with the recorded waves, and named: Delta, theta, alpha, beta, and gamma. EEGs are widely used in clinical practice to diagnose conditions like epilepsy, sleep disorders, and encephalopathies.

EEG (Electroencephalogram) frequency bands constitute specific ranges within the brain's electrical activity, each correlated with distinct states of neural function and cognitive processes. These bands are delineated based on the oscillation rate or wave frequency per second, quantified in Hertz (Hz).

Delta waves, ranging from 0.5 to 4 Hz, represent the slowest oscillatory activity in the brain. Predominantly observed during the deepest stages of sleep, known as slow-wave sleep, delta waves are also prominent in infants and young children. In adults, the presence of elevated delta wave activity while awake can be indicative of severe brain injuries, underlying pathologies, or other significant neurological anomalies.

Theta waves, oscillating between 4 and 8 Hz, are marginally faster than delta waves and are typically associated with light sleep, drowsiness, meditative states, and profound relaxation. Enhanced theta activity can often be observed in individuals with attention deficits, learning disabilities, and various forms of brain dysfunction, highlighting its diagnostic relevance in these contexts.

Alpha waves, with frequencies ranging from 8 to 13 Hz, are characterized by moderate speed and amplitude. These waves are most pronounced when an individual is awake but in a relaxed state, such as during quiet rest or meditation with closed eyes. The presence of alpha waves diminishes with mental exertion and concentrated effort, serving as a marker for relaxation. Abnormalities in alpha wave patterns can signify underlying brain disorders, making their analysis crucial for diagnostic purposes.

Beta waves, spanning frequencies from 13 to 30 Hz, are rapid, low-amplitude waves associated with active cognitive processes. These waves are evident during periods of intense mental activity, problem-solving, anxiety, and engagement in complex tasks. An overabundance of beta activity is frequently linked to stress and certain psychiatric conditions, whereas a reduction in beta wave activity can be observed in instances of brain injury or neurological disease.

Gamma waves, the fastest brain waves with frequencies ranging from 30 to 100 Hz, possess the smallest amplitude and are implicated in higher cognitive functions. These include perception, problem-solving, and consciousness. Gamma waves are believed to play a pivotal role in the integration of sensory information and the synchronization of different brain regions. Aberrations in gamma activity have been associated with cognitive impairments, schizophrenia, and other significant neurological disorders, underscoring their importance in the realm of neurophysiological research.

EEGs provide insights into brain functions and neural network integrity. What has been practical to help our project, is that we have considered that patients with Alzheimer's often exhibit distinct EEG patterns, such as a reduced power in higher frequency bands (alpha and beta) and increased power in lower frequency bands (delta and theta).

Engineering Fundamentals

Neural networks are machine learning models based on how the neurons of human brains work. They consist of interconnected nodes known as neurons, organized in layers. Like other machine learning models, neural networks are designed to learn patterns and make predictions based on data, making them interesting tools to take into account for tasks like classification.

The fundamental parts of a neural network model are neurons and layers:

Neurons receive inputs, process them, and pass the outputs to the next layer. Meanwhile, there are 3 types of layers, the input layer which receives the initial data, the hidden layers which perform computations and feature extraction, and the output layer which produces the final results. Then there are connections and weights, in which each neuron is connected to one another in between layers and has an associated weight that adjusts during training.

This type of model works after training them, first, the forward propagation occurs, in which the data passes through the input layer, hidden layers, and output layer, and basically, the neurons process the data. Then comes the loss function step, in which the model measures the difference between the predicted output and the actual output. For the third step, the model performs a backpropagation, in which the neural network adjusts the weights to minimize the loss function. Finally, it optimizes the algorithms by updating the weights during training.

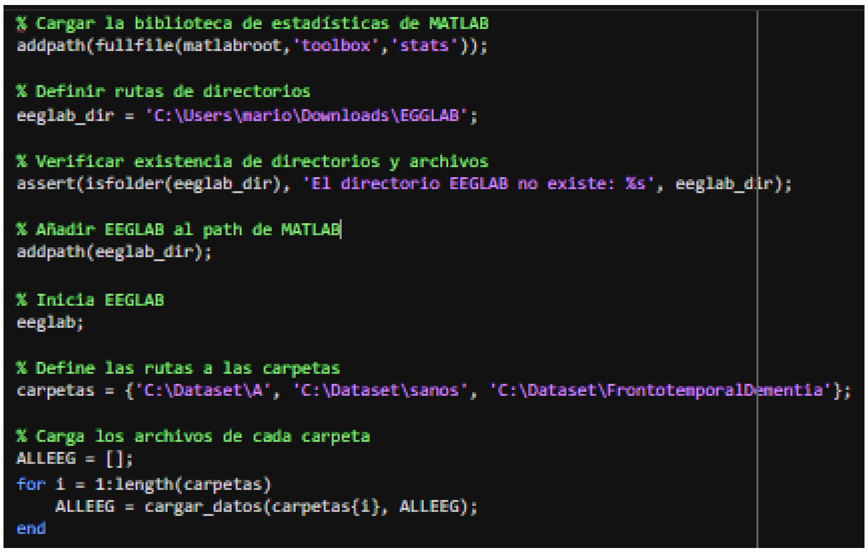
Since EEGs, measure electrical activity in the brain, the model analyzed frequency bands from delta, theta, alpha, beta, and gamma waves, and techniques such as FFT or Fast Fourier Transform and the Wavelet Transform were employed to decompose the EEG signals into separate frequencies in a filtering process. The specific type of model used was the Convolutional Neural Network method because this technology is suited for spatial data analysis such as EEG data by treating the brain signals as multi-dimensional arrays, which lets them capture local dependencies and hierarchical patterns. The model also utilized a Signal Processing Technique, that preprocessed the data by filtering the noise from the EEG signals, normalizing the data, and segmenting it into relevant 'cells' of data, and then extracting features from it.

Our objective to predict Alzheimer's using EEG data through neural networks is grounded in the theoretical and practical frameworks described previously, and the integration of this model with a rigorous signal processing technique provides a robust foundation for achieving high accuracy while diagnosing, hoping it also improves patient intervention and outcomes.

Prototype Design

EEG DATA PREPROCESSING

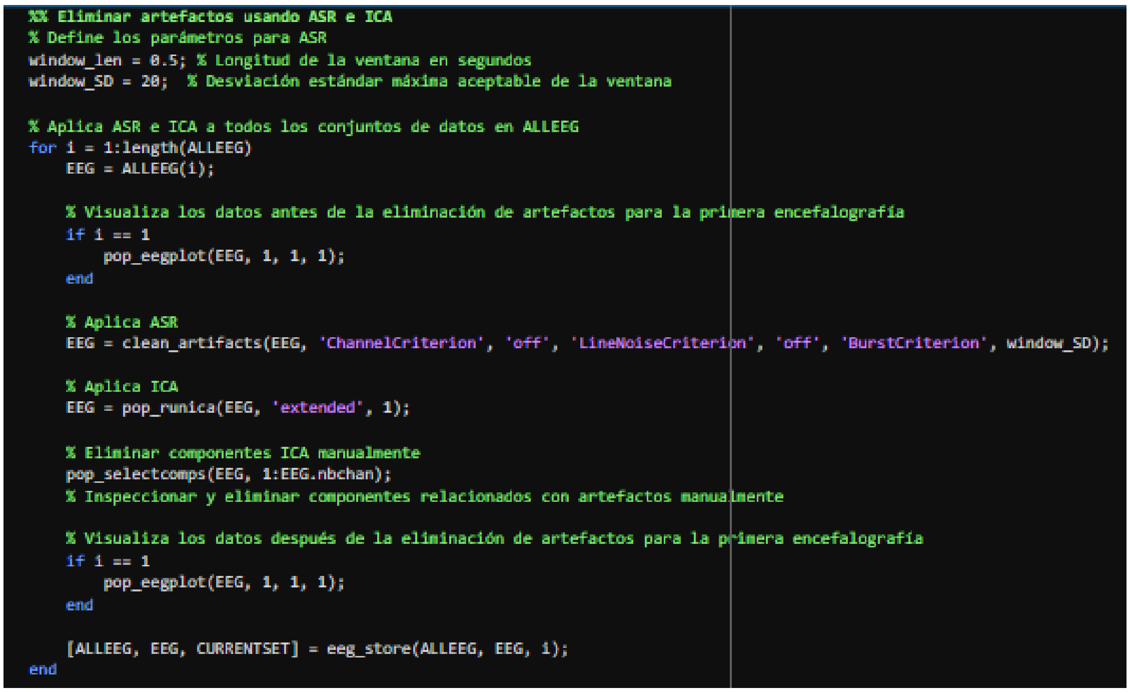
The initial phase of code development was conducted in MATLAB, where the addpath library was used as a starting point. This library was utilized to point to the directory containing the electroencephalograms. The directory was then verified, and the EEGs were added to Matlab with the function '*cargar_datos*' within the variable '*ALLEEG*', giving the start to the next section of code (Image 1).



```matlab
% Cargar la biblioteca de estadísticas de MATLAB
addpath(fullfile(matlabroot,'toolbox','stats'));

% Definir rutas de directorios
eeglab_dir = 'C:\Users\mario\Downloads\EGGLAB';

% Verificar existencia de directorios y archivos
assert(isfolder(eeglab_dir), 'El directorio EEGLAB no existe: %s', eeglab_dir);

% Añadir EEGLAB al path de MATLAB
addpath(eeglab_dir);

% Inicia EEGLAB
eeglab;

% Define las rutas a las carpetas
carpetas = {'C:\Dataset\A', 'C:\Dataset\sanos', 'C:\Dataset\FrontotemporalDementia'};

% Carga los archivos de cada carpeta
ALLEEG = [];
for i = 1:length(carpetas)
    ALLEEG = cargar_datos(carpetas{i}, ALLEEG);
end
```

Image 1. Adds EEG files

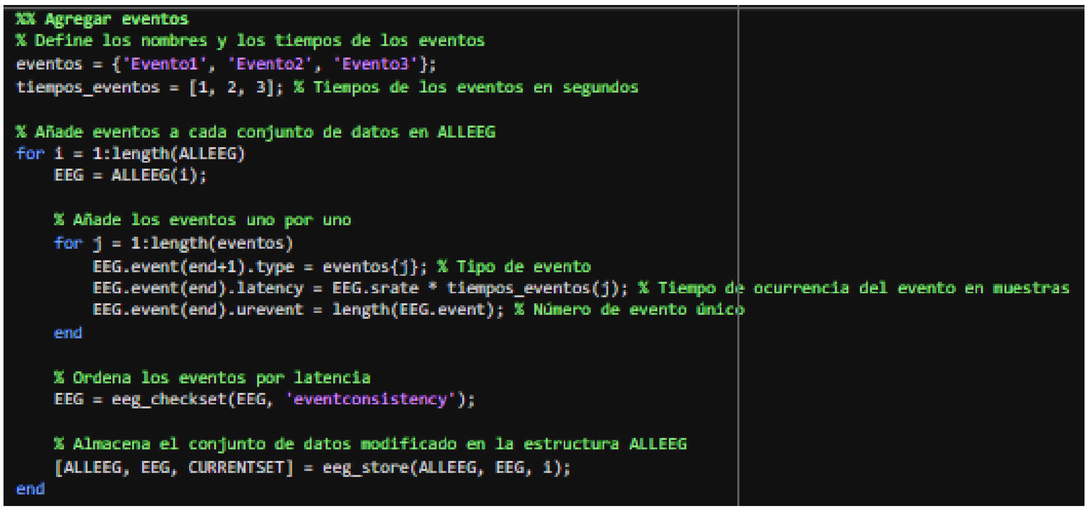
In the next section of the code, the EEG begins to be preprocessed. First, the data unclean is shown as unprocessed, then ASR (Artifact Subspace Reconstruction)and ICA (Independent Component Analysis) are implemented, which take a small window of 0.5 seconds of data and analyze any data that is out of the ordinary and erase them. After cleaning the data, the data are visualized again with the function '*pop_eegplot*', which returns an interactive plot of the cleaned data.



```
%% Eliminar artefactos usando ASR e ICA
% Define los parámetros para ASR
window_len = 0.5; % Longitud de la ventana en segundos
window_SD = 20;  % Desviación estándar máxima aceptable de la ventana

% Aplica ASR e ICA a todos los conjuntos de datos en ALLEEG
for i = 1:length(ALLEEG)
    EEG = ALLEEG(i);

    % Visualiza los datos antes de la eliminación de artefactos para la primera encefalografía
    if i == 1
        pop_eegplot(EEG, 1, 1, 1);
    end

    % Aplica ASR
    EEG = clean_artifacts(EEG, 'ChannelCriterion', 'off', 'LineNoiseCriterion', 'off', 'BurstCriterion', window_SD);

    % Aplica ICA
    EEG = pop_runica(EEG, 'extended', 1);

    % Eliminar componentes ICA manualmente
    pop_selectcomps(EEG, 1:EEG.nbchan);
    % Inspeccionar y eliminar componentes relacionados con artefactos manualmente

    % Visualiza los datos después de la eliminación de artefactos para la primera encefalografía
    if i == 1
        pop_eegplot(EEG, 1, 1, 1);
    end

    [ALLEEG, EEG, CURRENTSET] = eeg_store(ALLEEG, EEG, i);
end
```

Image 2. Eliminates impertinent data.

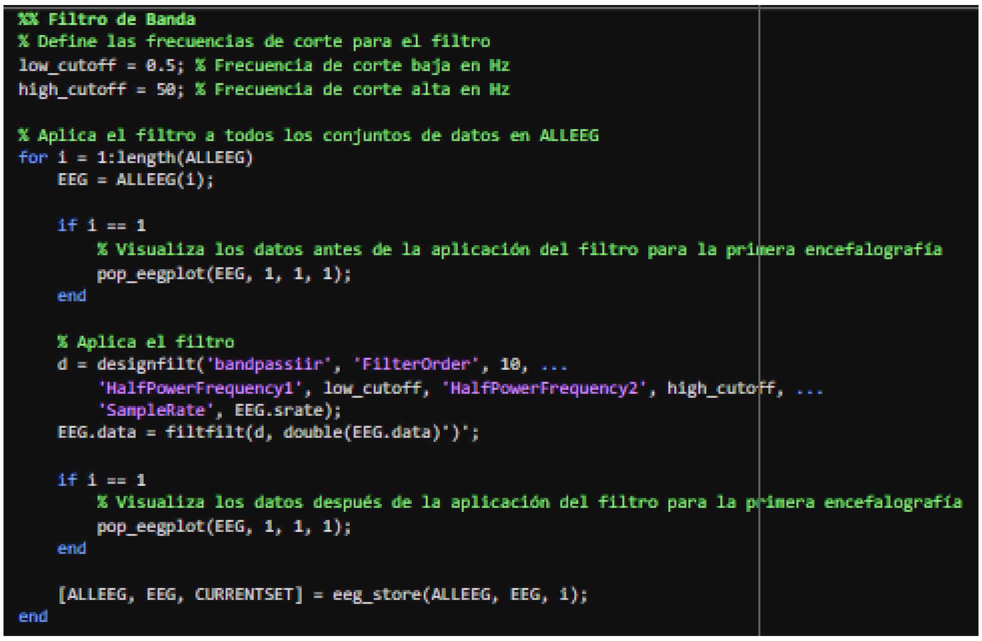
Afterward, the data is analyzed, and in the first, second, and third seconds of the data events 1, 2, and 3 are added respectively. The data is looped one by one, and the results of each data are stored into variables such as '*type*' which stores the event, and '*latency*' which stores samples that are obtained by multiplying the event in seconds by the frequency of the sample, and '*urevent*' which is a unique identification number. Furthermore, the data is organized by latency by means of the function '*eeg_checkset*'. The data is once again added to the variable '*ALLEEG*' (Image 3).



```
%% Agregar eventos
% Define los nombres y los tiempos de los eventos
eventos = {'Evento1', 'Evento2', 'Evento3'};
tiempos_eventos = [1, 2, 3]; % Tiempos de los eventos en segundos

% Añade eventos a cada conjunto de datos en ALLEEG
for i = 1:length(ALLEEG)
    EEG = ALLEEG(i);

    % Añade los eventos uno por uno
    for j = 1:length(eventos)
        EEG.event(end+1).type = eventos{j}; % Tipo de evento
        EEG.event(end).latency = EEG.srate * tiempos_eventos(j); % Tiempo de ocurrencia del evento en muestras
        EEG.event(end).urevent = length(EEG.event); % Número de evento único
    end

    % Ordena los eventos por latencia
    EEG = eeg_checkset(EEG, 'eventconsistency');

    % Almacena el conjunto de datos modificado en la estructura ALLEEG
    [ALLEEG, EEG, CURRENTSET] = eeg_store(ALLEEG, EEG, i);
end
```

Image 3. Divides and organizes data

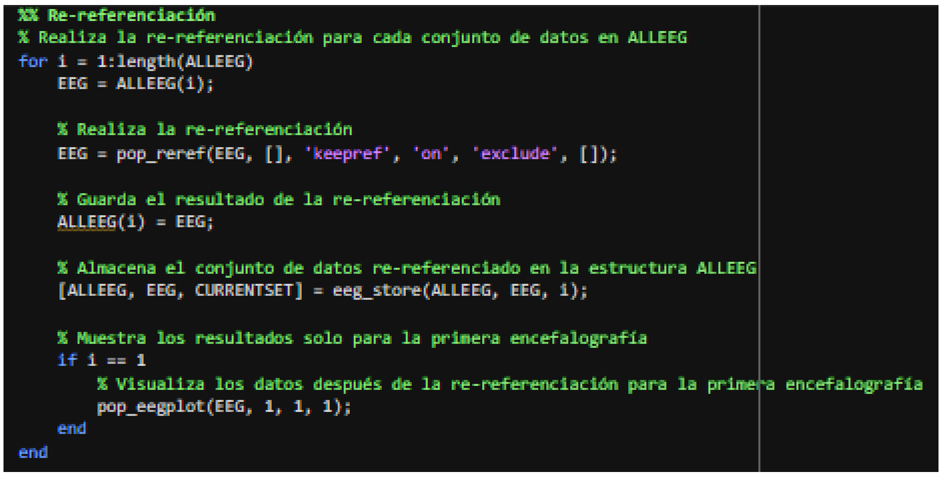
Subsequently, a range is added to the variables '*low_cutoff*' and '*high_cutoff*', with the intention of then looping through the data and eliminating any data that is not within the specified range, thereby preserving only the essential data (Image 4).



```
%% Filtro de Banda
% Define las frecuencias de corte para el filtro
low_cutoff = 0.5; % Frecuencia de corte baja en Hz
high_cutoff = 50; % Frecuencia de corte alta en Hz

% Aplica el filtro a todos los conjuntos de datos en ALLEEG
for i = 1:length(ALLEEG)
    EEG = ALLEEG(i);

    if i == 1
        % Visualiza los datos antes de la aplicación del filtro para la primera encefalografía
        pop_eegplot(EEG, 1, 1, 1);
    end

    % Aplica el filtro
    d = designfilt('bandpassiir', 'FilterOrder', 10, ...
        'HalfPowerFrequency1', low_cutoff, 'HalfPowerFrequency2', high_cutoff, ...
        'SampleRate', EEG.srate);
    EEG.data = filtfilt(d, double(EEG.data)')';

    if i == 1
        % Visualiza los datos después de la aplicación del filtro para la primera encefalografía
        pop_eegplot(EEG, 1, 1, 1);
    end

    [ALLEEG, EEG, CURRENTSET] = eeg_store(ALLEEG, EEG, i);
end
```

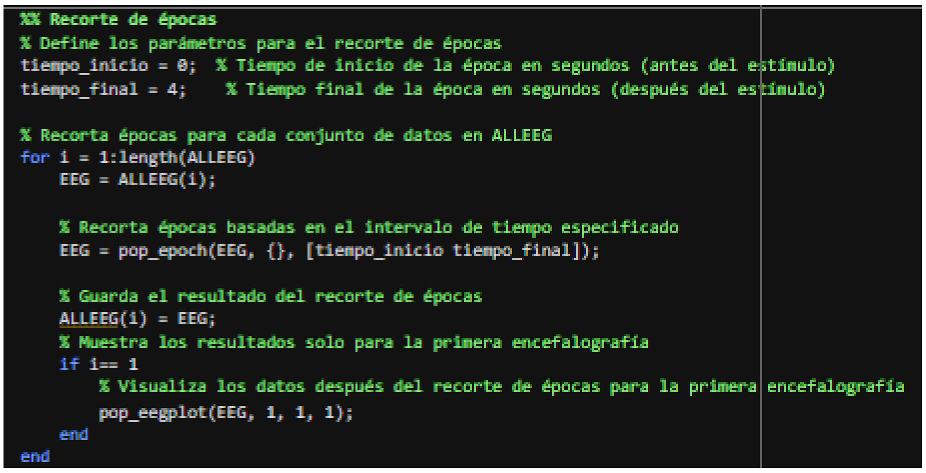
Image 4. Applies ranges to the data.

The next section of the code calculates the average of the signal in all channels and subtracts it from each channel individually. The result is then stored in the variable '*ALLEEG*'. Subsequently, a graph depicting the outcomes is presented (Image 5). The data are then divided into four-second intervals (Image 6), and an average is obtained. This average is then stored in the variable '*ALLEEG*' (Image 7).



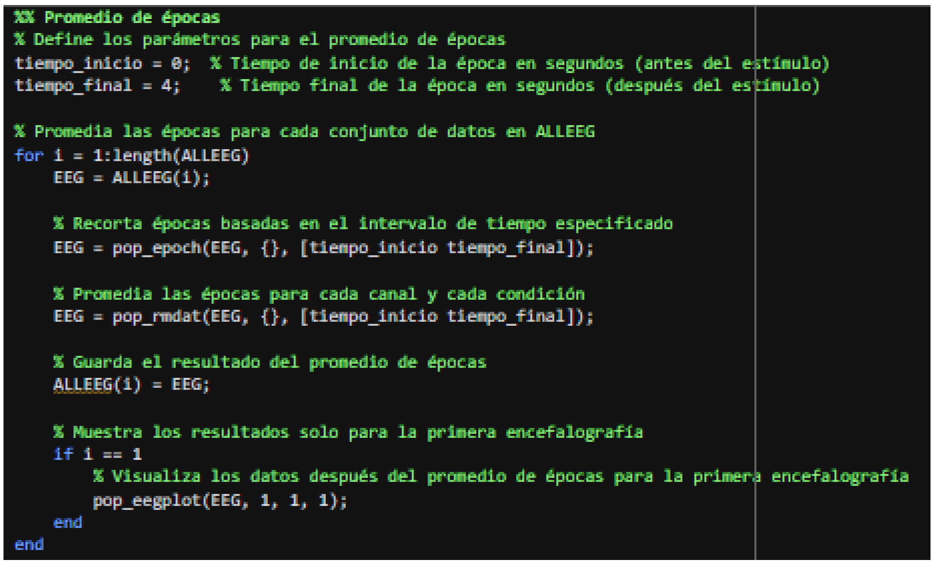
```
%% Re-referenciación
% Realiza la re-referenciación para cada conjunto de datos en ALLEEG
for i = 1:length(ALLEEG)
    EEG = ALLEEG(i);

    % Realiza la re-referenciación
    EEG = pop_reref(EEG, [], 'keepref', 'on', 'exclude', []);

    % Guarda el resultado de la re-referenciación
    ALLEEG(i) = EEG;

    % Almacena el conjunto de datos re-referenciado en la estructura ALLEEG
    [ALLEEG, EEG, CURRENTSET] = eeg_store(ALLEEG, EEG, i);

    % Muestra los resultados solo para la primera encefalografía
    if i == 1
        % Visualiza los datos después de la re-referenciación para la primera encefalografía
        pop_eegplot(EEG, 1, 1, 1);
    end
end
```

Image 5. Obtains average and subtracts it from the data.



```
%% Recorte de épocas
% Define los parámetros para el recorte de épocas
tiempo_inicio = 0;  % Tiempo de inicio de la época en segundos (antes del estímulo)
tiempo_final = 4;    % Tiempo final de la época en segundos (después del estímulo)

% Recorta épocas para cada conjunto de datos en ALLEEG
for i = 1:length(ALLEEG)
    EEG = ALLEEG(i);

    % Recorta épocas basadas en el intervalo de tiempo especificado
    EEG = pop_epoch(EEG, {}, [tiempo_inicio tiempo_final]);

    % Guarda el resultado del recorte de épocas
    ALLEEG(i) = EEG;
    % Muestra los resultados solo para la primera encefalografía
    if i== 1
        % Visualiza los datos después del recorte de épocas para la primera encefalografía
        pop_eegplot(EEG, 1, 1, 1);
    end
end
```

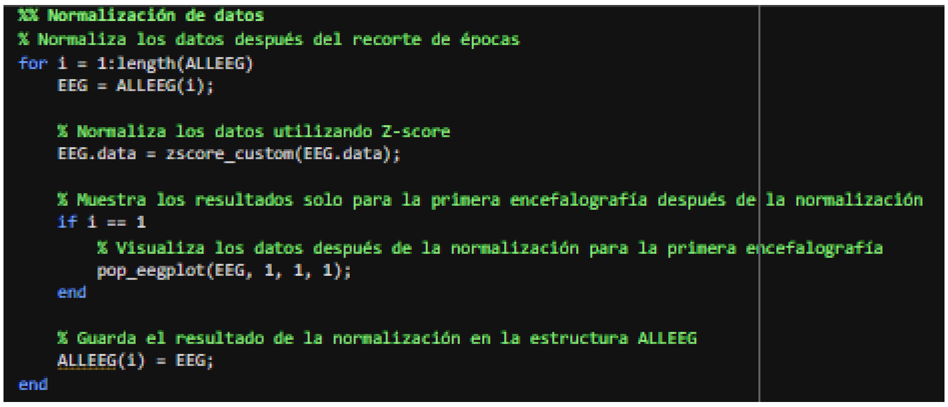
Image 6. Divide data into four-second intervals.



```
%% Promedio de épocas
% Define los parámetros para el promedio de épocas
tiempo_inicio = 0;  % Tiempo de inicio de la época en segundos (antes del estímulo)
tiempo_final = 4;    % Tiempo final de la época en segundos (después del estímulo)

% Promedia las épocas para cada conjunto de datos en ALLEEG
for i = 1:length(ALLEEG)
    EEG = ALLEEG(i);

    % Recorta épocas basadas en el intervalo de tiempo especificado
    EEG = pop_epoch(EEG, {}, [tiempo_inicio tiempo_final]);

    % Promedia las épocas para cada canal y cada condición
    EEG = pop_rmdat(EEG, {}, [tiempo_inicio tiempo_final]);

    % Guarda el resultado del promedio de épocas
    ALLEEG(i) = EEG;

    % Muestra los resultados solo para la primera encefalografía
    if i == 1
        % Visualiza los datos después del promedio de épocas para la primera encefalografía
        pop_eegplot(EEG, 1, 1, 1);
    end
end
```

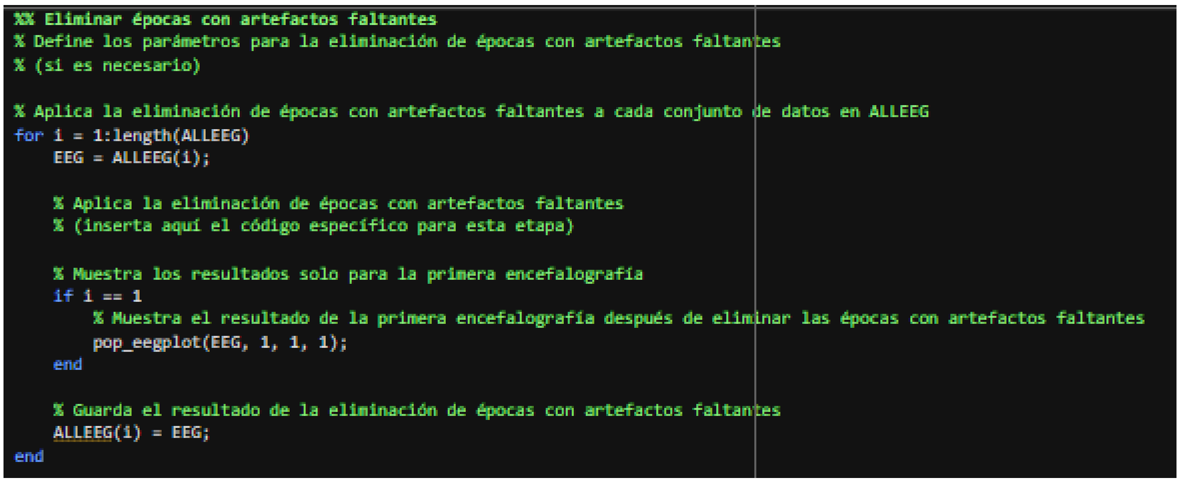
Image 7. Obtains average and stores it.

This next block of code normalizes the EEG data for each data set in '*ALLEEG*' using the Z-score. This procedure standardizes the data so that each channel has a mean of 0 and a standard deviation of 1. After normalization, the code displays the first data set to verify the process and saves the normalized data back into the '*ALLEEG*' structure (Image 8). Subsequently, for each EEG data set in '*ALLEEG*', the code deletes incomplete electrode data using a spherical method. After eliminating the missing data, it displays the data from the initial set to verify its accuracy and then stores the revised data back into the '*ALLEEG*' variable (Image 9).



```
%% Normalización de datos
% Normaliza los datos después del recorte de épocas
for i = 1:length(ALLEEG)
    EEG = ALLEEG(i);

    % Normaliza los datos utilizando Z-score
    EEG.data = zscore_custom(EEG.data);

    % Muestra los resultados solo para la primera encefalografía después de la normalización
    if i == 1
        % Visualiza los datos después de la normalización para la primera encefalografía
        pop_eegplot(EEG, 1, 1, 1);
    end

    % Guarda el resultado de la normalización en la estructura ALLEEG
    ALLEEG(i) = EEG;
end
```

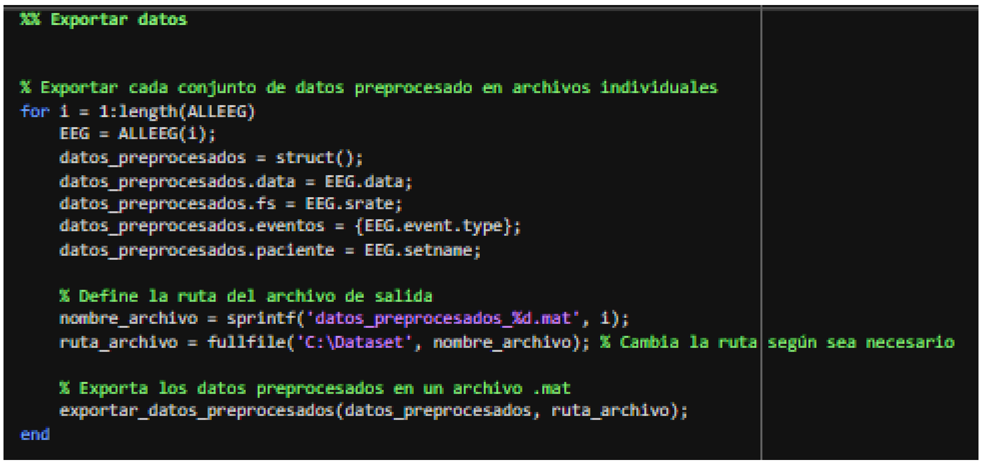
Image 8. Normalizes data.



```
%% Eliminar épocas con artefactos faltantes
% Define los parámetros para la eliminación de épocas con artefactos faltantes
% (si es necesario)

% Aplica la eliminación de épocas con artefactos faltantes a cada conjunto de datos en ALLEEG
for i = 1:length(ALLEEG)
    EEG = ALLEEG(i);

    % Aplica la eliminación de épocas con artefactos faltantes
    % (inserta aquí el código específico para esta etapa)

    % Muestra los resultados solo para la primera encefalografía
    if i == 1
        % Muestra el resultado de la primera encefalografía después de eliminar las épocas con artefactos faltantes
        pop_eegplot(EEG, 1, 1, 1);
    end

    % Guarda el resultado de la eliminación de épocas con artefactos faltantes
    ALLEEG(i) = EEG;
end
```

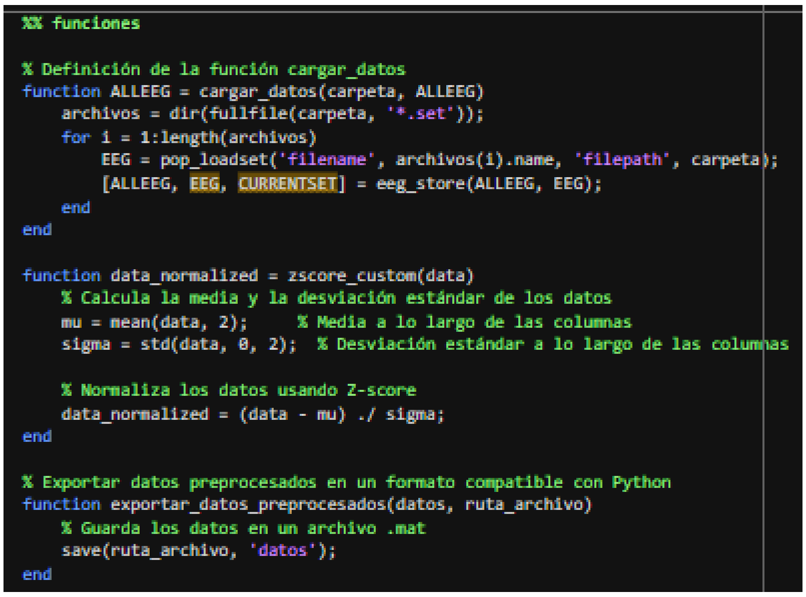
Image 9. Eliminates incomplete data

The final section of code goes through each set of preprocessed EEG data, arranges the information into a structured format, establishes a path to the output file, and then stores the data in individual .mat files for subsequent utilization (Image 10). And invokes all methods to ensure their correct execution. (Image 11).



```
%% Exportar datos

% Exportar cada conjunto de datos preprocesado en archivos individuales
for i = 1:length(ALLEEG)
    EEG = ALLEEG(i);
    datos_preprocesados = struct();
    datos_preprocesados.data = EEG.data;
    datos_preprocesados.fs = EEG.srate;
    datos_preprocesados.eventos = {EEG.event.type};
    datos_preprocesados.paciente = EEG.setname;

    % Define la ruta del archivo de salida
    nombre_archivo = sprintf('datos_preprocesados_%d.mat', i);
    ruta_archivo = fullfile('C:\Dataset', nombre_archivo); % Cambia la ruta según sea necesario

    % Exporta los datos preprocesados en un archivo .mat
    exportar_datos_preprocesados(datos_preprocesados, ruta_archivo);
end
```

Image 10. Exports data.



```
%% funciones

% Definición de la función cargar_datos
function ALLEEG = cargar_datos(carpeta, ALLEEG)
    archivos = dir(fullfile(carpeta, '*.set'));
    for i = 1:length(archivos)
        EEG = pop_loadset('filename', archivos(i).name, 'filepath', carpeta);
        [ALLEEG, EEG, CURRENTSET] = eeg_store(ALLEEG, EEG);
    end
end

function data_normalized = zscore_custom(data)
    % Calcula la media y la desviación estándar de los datos
    mu = mean(data, 2);     % Media a lo largo de las columnas
    sigma = std(data, 0, 2);  % Desviación estándar a lo largo de las columnas

    % Normaliza los datos usando Z-score
    data_normalized = (data - mu) ./ sigma;
end

% Exportar datos preprocesados en un formato compatible con Python
function exportar_datos_preprocesados(datos, ruta_archivo)
    % Guarda los datos en un archivo .mat
    save(ruta_archivo, 'datos');
end
```

Image 11. Function calling.

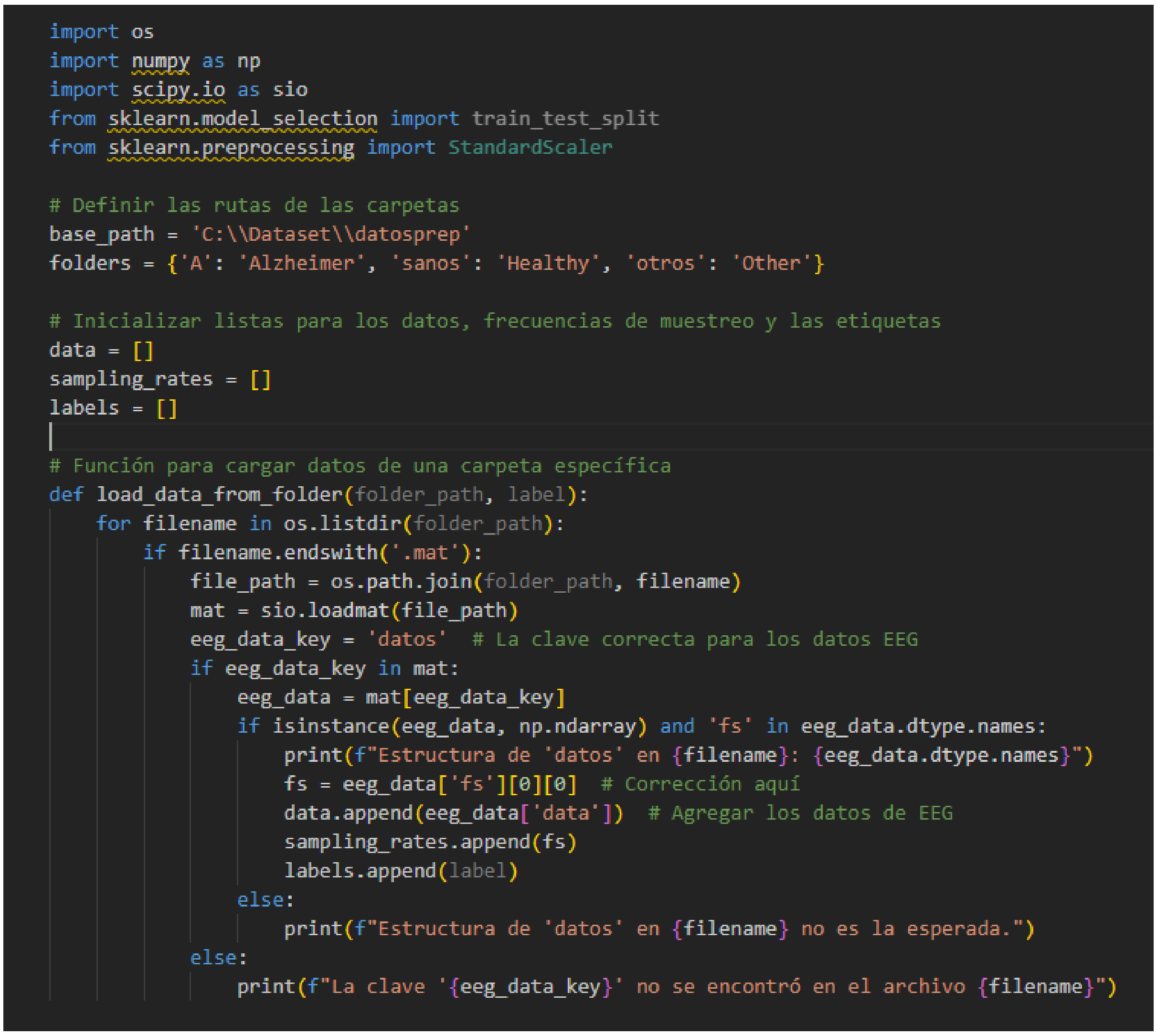
DATA PROCESSING

The last phase of code development was conducted in Python and was divided into cells, thus it will be explained as such.

DATA LOADING

The first cell processes EEG data stored in .mat files from the previous phase. The libraries used were ‘os’ for interacting with the operating system, ‘numpy’ for handling arrays and numerical data, ‘scipy.io’ for reading the MATLAB (.mat) files, and ‘sklearn’ for machine learning tasks, although it wasn’t used yet.

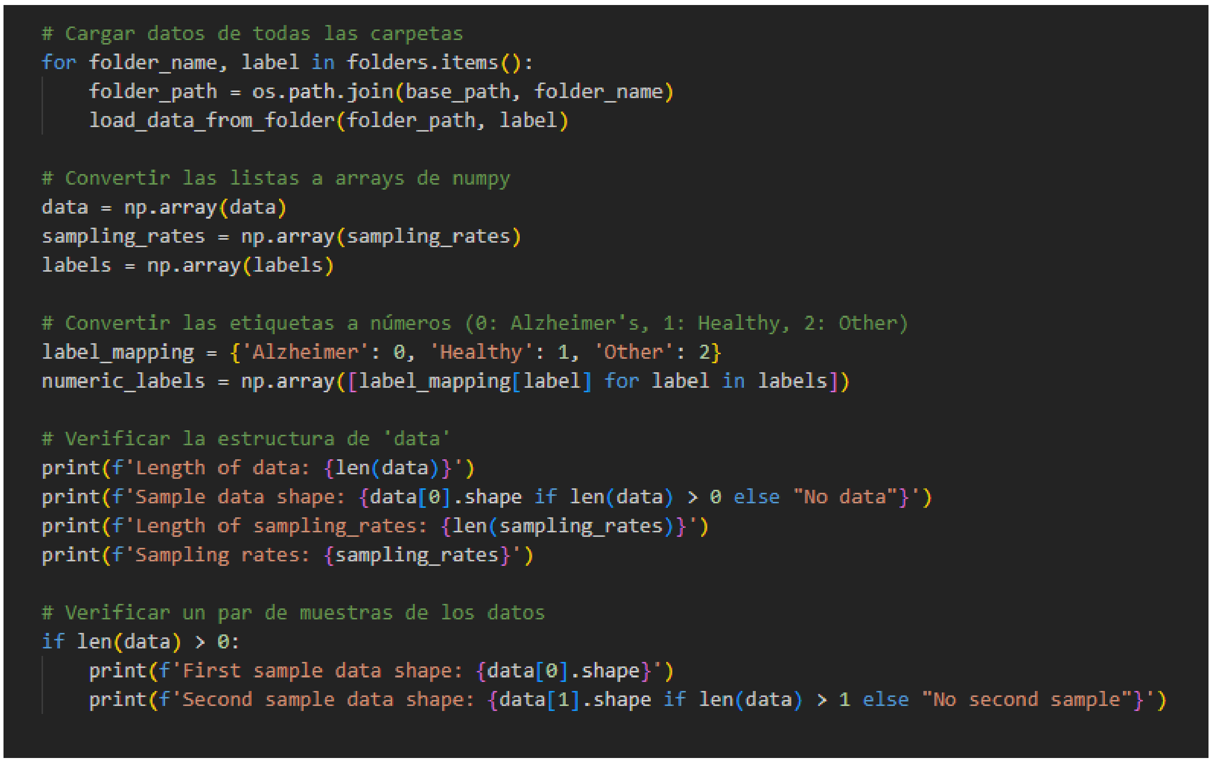
The code first sets up the paths where the data will be stored with ‘base_path’ and then categorizes it in the labels for EEG data from ‘Alzheimer’, ‘Healthy’, and ‘Other’ patients. Then it creates empty lists to store the EEG data, the sampling rates, and the labels.



```python
import os
import numpy as np
import scipy.io as sio
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Definir las rutas de las carpetas
base_path = 'C:\\Dataset\\datosprep'
folders = {'A': 'Alzheimer', 'sanos': 'Healthy', 'otros': 'Other'}

# Inicializar listas para los datos, frecuencias de muestreo y las etiquetas
data = []
sampling_rates = []
labels = []

# Función para cargar datos de una carpeta específica
def load_data_from_folder(folder_path, label):
    for filename in os.listdir(folder_path):
        if filename.endswith('.mat'):
            file_path = os.path.join(folder_path, filename)
            mat = sio.loadmat(file_path)
            eeg_data_key = 'datos'  # La clave correcta para los datos EEG
            if eeg_data_key in mat:
                eeg_data = mat[eeg_data_key]
                if isinstance(eeg_data, np.ndarray) and 'fs' in eeg_data.dtype.names:
                    print(f"Estructura de 'datos' en {filename}: {eeg_data.dtype.names}")
                    fs = eeg_data['fs'][0][0]  # Corrección aquí
                    data.append(eeg_data['data'])  # Agregar los datos de EEG
                    sampling_rates.append(fs)
                    labels.append(label)
                else:
                    print(f"Estructura de 'datos' en {filename} no es la esperada.")
            else:
                print(f"La clave '{eeg_data_key}' no se encontró en el archivo {filename}")
```

Image 12. Libraries, path settlement, and label categorization.

After all these preparations are made, the model loads the data, locating it in the labeled folders, and generates a loop to read the data from each folder with 'load_data_from_folder'. Once all the data is collected, the lists convert into numpy arrays, and the labels convert into numerical values. Finally, the cell prints some information to verify that the data-loading process was successful.



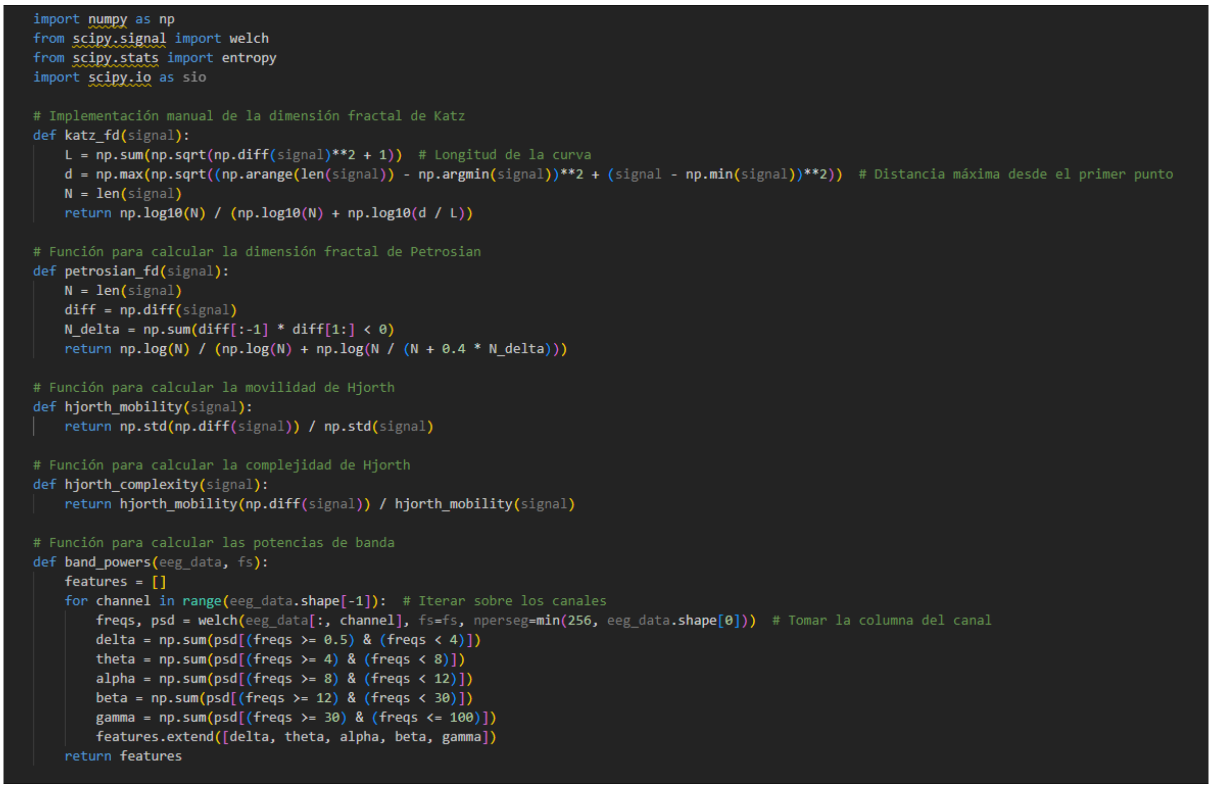
```python
# Cargar datos de todas las carpetas
for folder_name, label in folders.items():
    folder_path = os.path.join(base_path, folder_name)
    load_data_from_folder(folder_path, label)

# Convertir las listas a arrays de numpy
data = np.array(data)
sampling_rates = np.array(sampling_rates)
labels = np.array(labels)

# Convertir las etiquetas a números (0: Alzheimer's, 1: Healthy, 2: Other)
label_mapping = {'Alzheimer': 0, 'Healthy': 1, 'Other': 2}
numeric_labels = np.array([label_mapping[label] for label in labels])

# Verificar la estructura de 'data'
print(f'Length of data: {len(data)}')
print(f'Sample data shape: {data[0].shape if len(data) > 0 else "No data"}')
print(f'Length of sampling_rates: {len(sampling_rates)}')
print(f'Sampling rates: {sampling_rates}')

# Verificar un par de muestras de los datos
if len(data) > 0:
    print(f'First sample data shape: {data[0].shape}')
    print(f'Second sample data shape: {data[1].shape if len(data) > 1 else "No second sample"}')
```

Image 13. Data loading, value conversions, and data verification.

DATA EXTRACTION

The next cell extracts the loaded data and calculates various features for each sample and then organizes them into a consistent format for their further analysis. The libraries used were 'numpy' again for handling arrays and numerical operations, 'scipy.io' also again for loading the MATLAB files, 'scipy.signal' for signal processing tasks and 'scipy.stats' for statistical calculations.

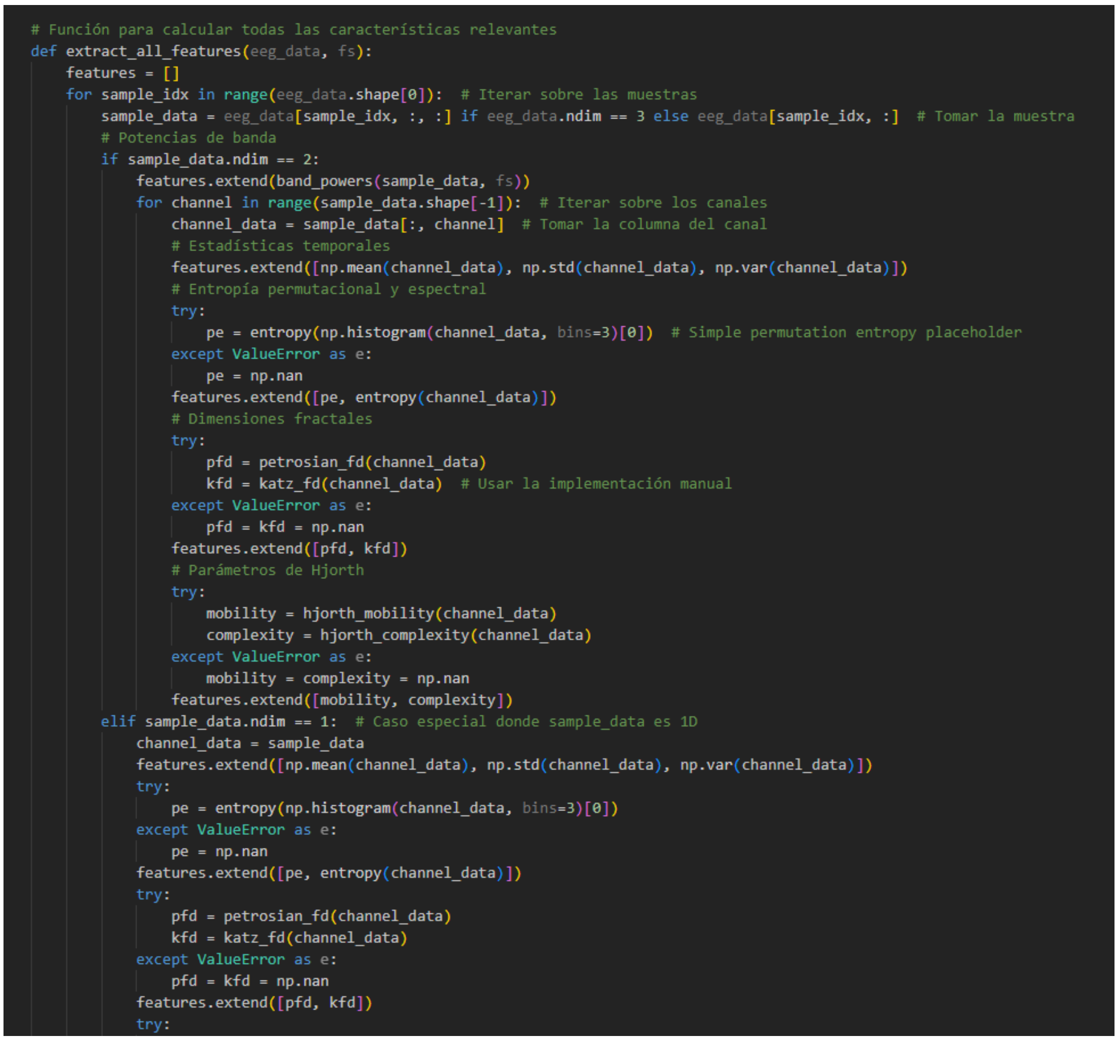
The code first defines both Katz's fractal dimension ('katz_fd') and Petrosian's fractal dimension ('petrosian_fd'), which both measure the complexity of a time series. Then, it defines the Hjorth parameters functions for mobility ('hjorth_mobility') and complexity ('hjorth_complexity'), the first indicates the signal's frequency content and the latest measures the signal's structural complexity. Later the code calculates the band powers from deltha, theta, alpha, beta, and gamma frequencies with 'band_powers'.



```python
import numpy as np
from scipy.signal import welch
from scipy.stats import entropy
import scipy.io as sio

# Implementación manual de la dimensión fractal de Katz
def katz_fd(signal):
    L = np.sum(np.sqrt(np.diff(signal)**2 + 1))  # Longitud de la curva
    d = np.max(np.sqrt((np.arange(len(signal)) - np.argmin(signal))**2 + (signal - np.min(signal))**2))  # Distancia máxima desde el primer punto
    N = len(signal)
    return np.log10(N) / (np.log10(N) + np.log10(d / L))

# Función para calcular la dimensión fractal de Petrosian
def petrosian_fd(signal):
    N = len(signal)
    diff = np.diff(signal)
    N_delta = np.sum(diff[:-1] * diff[1:] < 0)
    return np.log(N) / (np.log(N) + np.log(N / (N + 0.4 * N_delta)))

# Función para calcular la movilidad de Hjorth
def hjorth_mobility(signal):
    return np.std(np.diff(signal)) / np.std(signal)

# Función para calcular la complejidad de Hjorth
def hjorth_complexity(signal):
    return hjorth_mobility(np.diff(signal)) / hjorth_mobility(signal)

# Función para calcular las potencias de banda
def band_powers(eeg_data, fs):
    features = []
    for channel in range(eeg_data.shape[-1]):  # Iterar sobre los canales
        freqs, psd = welch(eeg_data[:, channel], fs=fs, nperseg=min(256, eeg_data.shape[0]))  # Tomar la columna del canal
        delta = np.sum(psd[(freqs >= 0.5) & (freqs < 4)])
        theta = np.sum(psd[(freqs >= 4) & (freqs < 8)])
        alpha = np.sum(psd[(freqs >= 8) & (freqs < 12)])
        beta = np.sum(psd[(freqs >= 12) & (freqs < 30)])
        gamma = np.sum(psd[(freqs >= 30) & (freqs <= 100)])
        features.extend([delta, theta, alpha, beta, gamma])
    return features
```

Image 14. Libraries, functions implementation, band powers extraction.

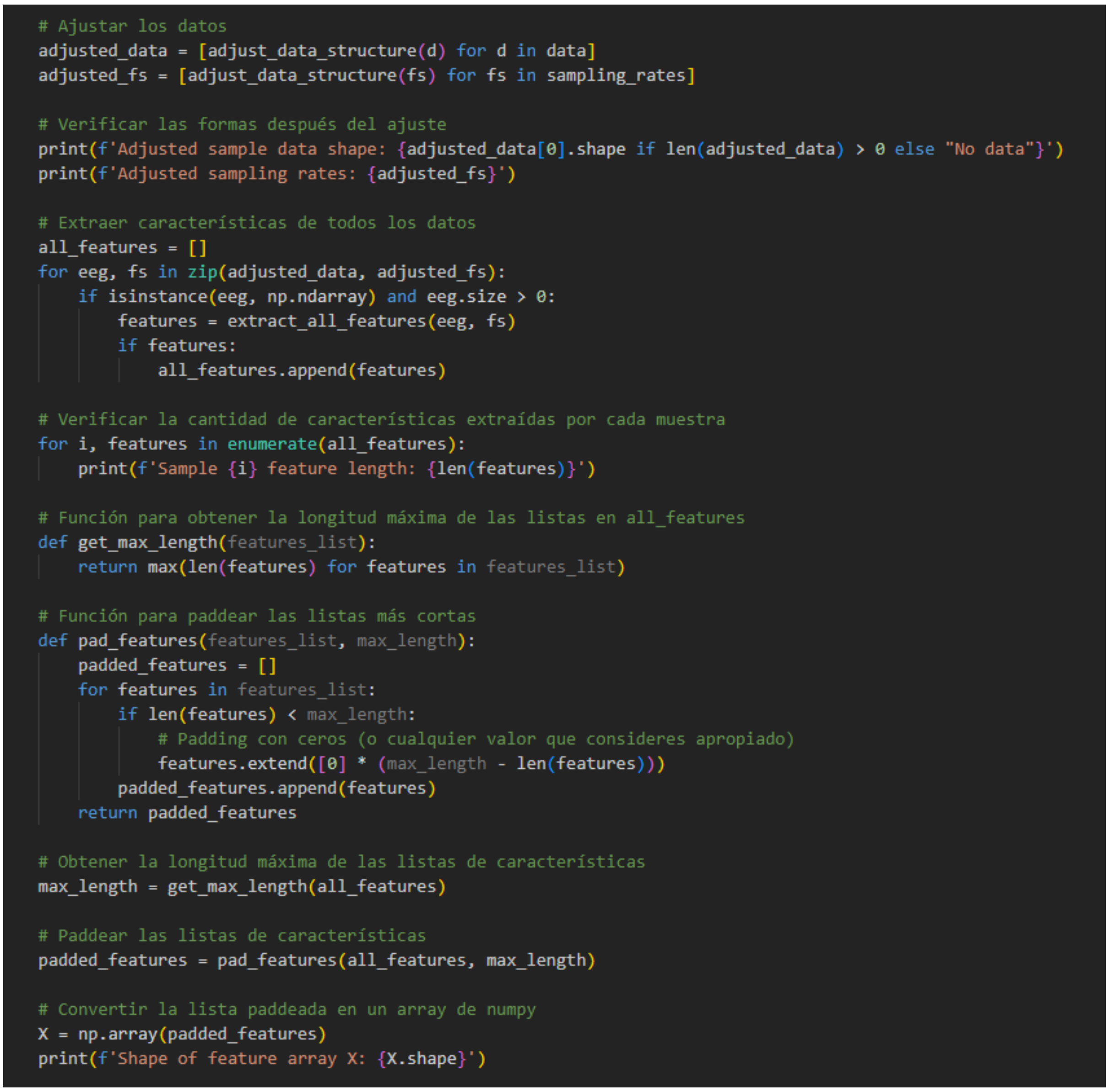
Once all the functions are defined and the band powers are extracted, the code combines all the features along other statistical measures with ‘extract_all_features’, ensures that the data is in the correct format with ‘adjust_data_structure’, and finally iterates through each sample and calculates all the features with ‘extract_all_features’.



```python
# Función para calcular todas las características relevantes
def extract_all_features(eeg_data, fs):
    features = []
    for sample_idx in range(eeg_data.shape[0]):  # Iterar sobre las muestras
        sample_data = eeg_data[sample_idx, :, :] if eeg_data.ndim == 3 else eeg_data[sample_idx, :]  # Tomar la muestra
        # Potencias de banda
        if sample_data.ndim == 2:
            features.extend(band_powers(sample_data, fs))
            for channel in range(sample_data.shape[-1]):  # Iterar sobre los canales
                channel_data = sample_data[:, channel]  # Tomar la columna del canal
                # Estadísticas temporales
                features.extend([np.mean(channel_data), np.std(channel_data), np.var(channel_data)])
                # Entropía permutacional y espectral
                try:
                    pe = entropy(np.histogram(channel_data, bins=3)[0])  # Simple permutation entropy placeholder
                except ValueError as e:
                    pe = np.nan
                features.extend([pe, entropy(channel_data)])
                # Dimensiones fractales
                try:
                    pfd = petrosian_fd(channel_data)
                    kfd = katz_fd(channel_data)  # Usar la implementación manual
                except ValueError as e:
                    pfd = kfd = np.nan
                features.extend([pfd, kfd])
                # Parámetros de Hjorth
                try:
                    mobility = hjorth_mobility(channel_data)
                    complexity = hjorth_complexity(channel_data)
                except ValueError as e:
                    mobility = complexity = np.nan
                features.extend([mobility, complexity])
        elif sample_data.ndim == 1:  # Caso especial donde sample_data es 1D
            channel_data = sample_data
            features.extend([np.mean(channel_data), np.std(channel_data), np.var(channel_data)])
            try:
                pe = entropy(np.histogram(channel_data, bins=3)[0])
            except ValueError as e:
                pe = np.nan
            features.extend([pe, entropy(channel_data)])
            try:
                pfd = petrosian_fd(channel_data)
                kfd = katz_fd(channel_data)
            except ValueError as e:
                pfd = kfd = np.nan
            features.extend([pfd, kfd])
            try:
```

Image 15. Function to extract all relevant features.

After the features are extracted, the code finds the maximum length of the feature lists with ‘get_max_length’ and ensures all of these are of the same length by padding with zeros with ‘pad_features’. To end this cell’s code, the data is converted to numpy arrays.

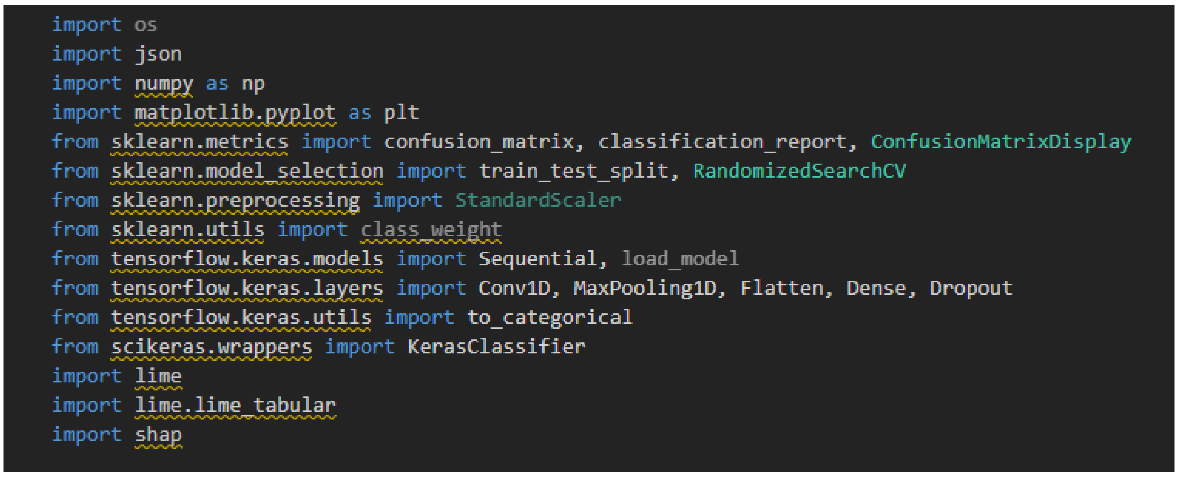


```python
# Ajustar los datos
adjusted_data = [adjust_data_structure(d) for d in data]
adjusted_fs = [adjust_data_structure(fs) for fs in sampling_rates]

# Verificar las formas después del ajuste
print(f'Adjusted sample data shape: {adjusted_data[0].shape if len(adjusted_data) > 0 else "No data"}')
print(f'Adjusted sampling rates: {adjusted_fs}')

# Extraer características de todos los datos
all_features = []
for eeg, fs in zip(adjusted_data, adjusted_fs):
    if isinstance(eeg, np.ndarray) and eeg.size > 0:
        features = extract_all_features(eeg, fs)
        if features:
            all_features.append(features)

# Verificar la cantidad de características extraídas por cada muestra
for i, features in enumerate(all_features):
    print(f'Sample {i} feature length: {len(features)}')

# Función para obtener la longitud máxima de las listas en all_features
def get_max_length(features_list):
    return max(len(features) for features in features_list)

# Función para paddear las listas más cortas
def pad_features(features_list, max_length):
    padded_features = []
    for features in features_list:
        if len(features) < max_length:
            # Padding con ceros (o cualquier valor que consideres apropiado)
            features.extend([0] * (max_length - len(features)))
        padded_features.append(features)
    return padded_features

# Obtener la longitud máxima de las listas de características
max_length = get_max_length(all_features)

# Paddear las listas de características
padded_features = pad_features(all_features, max_length)

# Convertir la lista paddeada en un array de numpy
X = np.array(padded_features)
print(f'Shape of feature array X: {X.shape}')
```

Image 16. Data adjustment, feature extraction iteration, data conversion to arrays.

MODEL TRAINING AND DATA EVALUATION

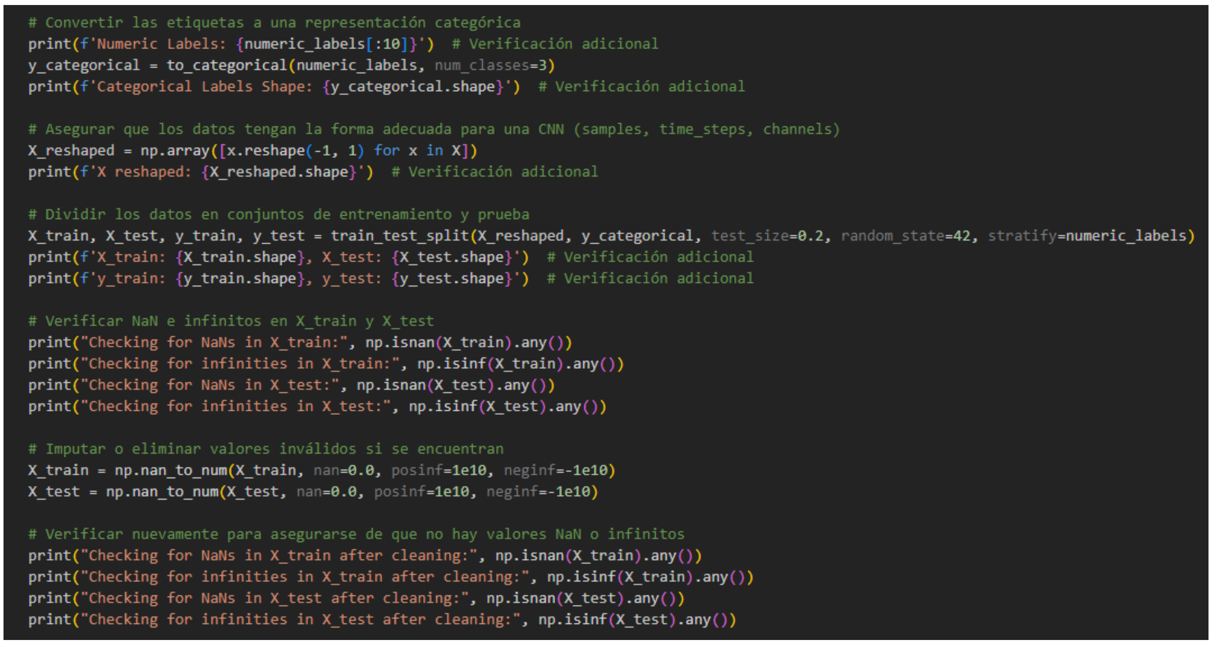
The last cell trains and evaluates the Convolutional Neural Network model for classifying EEG data into the categories, and thus, predicts Alzheimer's disease. The libraries used were 'os' again to handle directory and file operations, 'json' to save training history as a JSON file, 'numpy' once more for numerical operations and handling arrays, 'matplotlib.pyplot' for plotting graphs, 'sklearn' for data preprocessing, splitting, evaluation and hyperparameter tuning, 'tensorflow.keras' for building and training the neural network, and 'lime' and 'shap', which were for explaining model predictions but not used in this cell.



```python
import os
import json
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.utils import class_weight
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from scikeras.wrappers import KerasClassifier
import lime
import lime.lime_tabular
import shap
```

Image 17. Libraries.

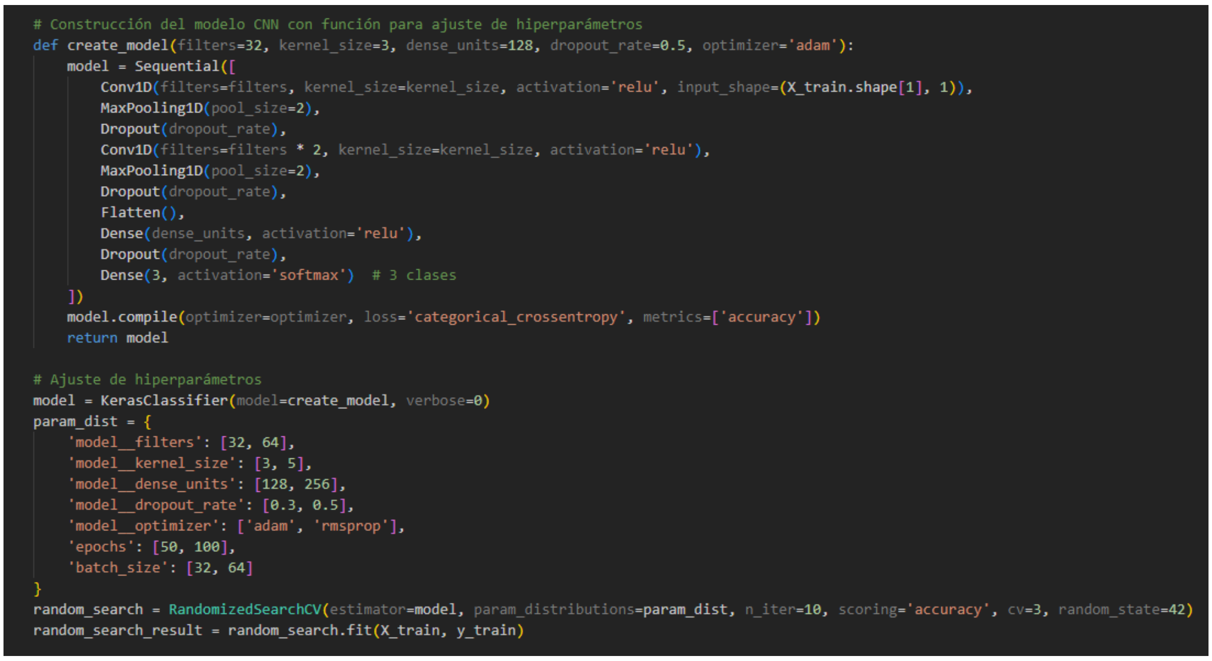
First the code prepares the labels by converting the numerical labels (converted during the previous cell) into a format suitable for neural network training using 'to_categorical'. Then it reshapes the data with 'x_reshaped' to include a time step and a single channel. Then the data is split into training and testing sets. It then checks the Not a Number values and infinite values in the training testing sets and replaces them with valid numbers.



```python
# Convertir las etiquetas a una representación categórica
print(f'Numeric Labels: {numeric_labels[:10]}')  # Verificación adicional
y_categorical = to_categorical(numeric_labels, num_classes=3)
print(f'Categorical Labels Shape: {y_categorical.shape}')  # Verificación adicional

# Asegurar que los datos tengan la forma adecuada para una CNN (samples, time_steps, channels)
X_reshaped = np.array([x.reshape(-1, 1) for x in X])
print(f'X reshaped: {X_reshaped.shape}')  # Verificación adicional

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y_categorical, test_size=0.2, random_state=42, stratify=numeric_labels)
print(f'X_train: {X_train.shape}, X_test: {X_test.shape}')  # Verificación adicional
print(f'y_train: {y_train.shape}, y_test: {y_test.shape}')  # Verificación adicional

# Verificar NaN e infinitos en X_train y X_test
print("Checking for NaNs in X_train:", np.isnan(X_train).any())
print("Checking for infinities in X_train:", np.isinf(X_train).any())
print("Checking for NaNs in X_test:", np.isnan(X_test).any())
print("Checking for infinities in X_test:", np.isinf(X_test).any())

# Imputar o eliminar valores inválidos si se encuentran
X_train = np.nan_to_num(X_train, nan=0.0, posinf=1e10, neginf=-1e10)
X_test = np.nan_to_num(X_test, nan=0.0, posinf=1e10, neginf=-1e10)

# Verificar nuevamente para asegurarse de que no hay valores NaN o infinitos
print("Checking for NaNs in X_train after cleaning:", np.isnan(X_train).any())
print("Checking for infinities in X_train after cleaning:", np.isinf(X_train).any())
print("Checking for NaNs in X_test after cleaning:", np.isnan(X_test).any())
print("Checking for infinities in X_test after cleaning:", np.isinf(X_test).any())
```

Image 18. Label conversion, data reshaping and splitting, NaN and infinites verification.

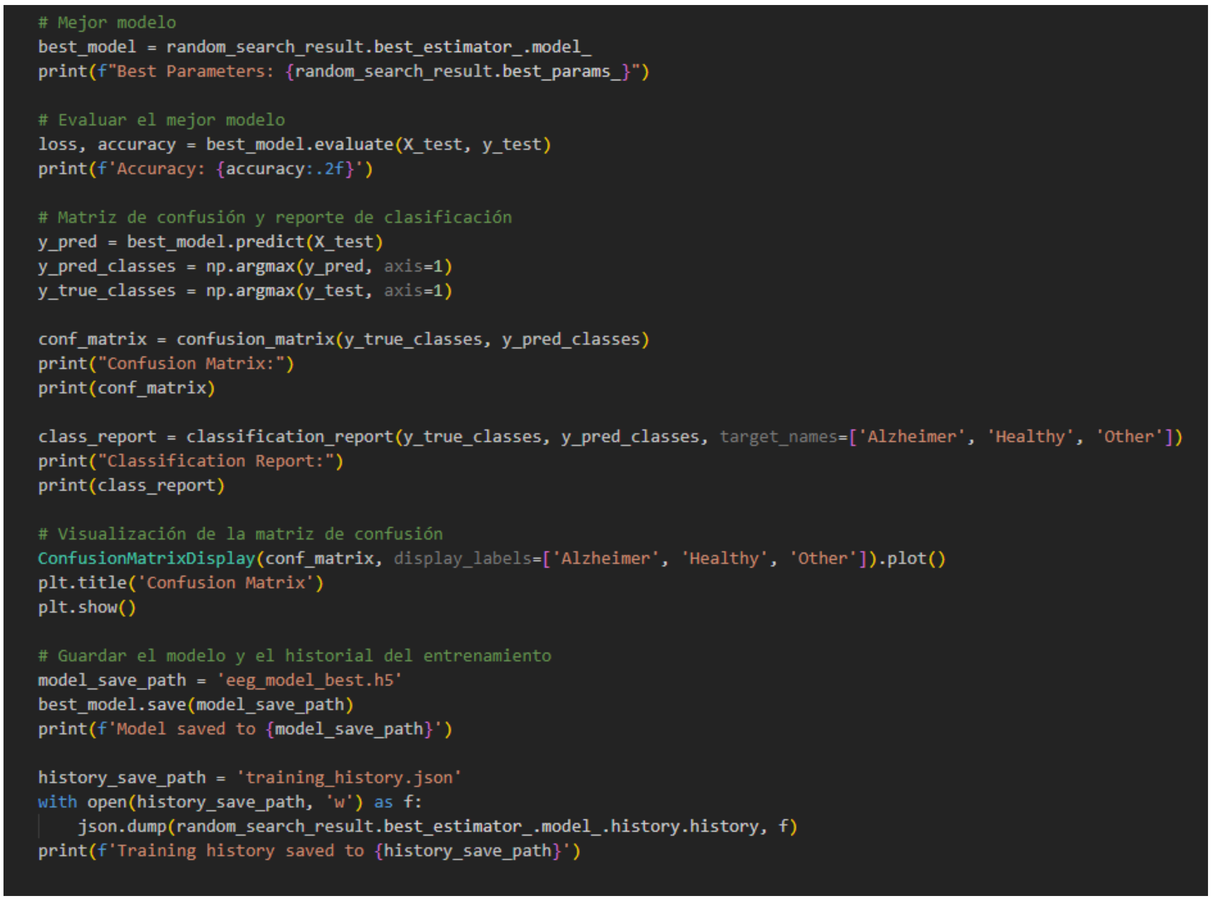
After all these preparations are done, the model defines a function to create the CNN model with adjustable parameters, and with the function 'RandomizedSearchCV' to find the best hyperparameters for the CNN model.



```python
# Construcción del modelo CNN con función para ajuste de hiperparámetros
def create_model(filters=32, kernel_size=3, dense_units=128, dropout_rate=0.5, optimizer='adam'):
    model = Sequential([
        Conv1D(filters=filters, kernel_size=kernel_size, activation='relu', input_shape=(X_train.shape[1], 1)),
        MaxPooling1D(pool_size=2),
        Dropout(dropout_rate),
        Conv1D(filters=filters * 2, kernel_size=kernel_size, activation='relu'),
        MaxPooling1D(pool_size=2),
        Dropout(dropout_rate),
        Flatten(),
        Dense(dense_units, activation='relu'),
        Dropout(dropout_rate),
        Dense(3, activation='softmax')  # 3 clases
    ])
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Ajuste de hiperparámetros
model = KerasClassifier(model=create_model, verbose=0)
param_dist = {
    'model__filters': [32, 64],
    'model__kernel_size': [3, 5],
    'model__dense_units': [128, 256],
    'model__dropout_rate': [0.3, 0.5],
    'model__optimizer': ['adam', 'rmsprop'],
    'epochs': [50, 100],
    'batch_size': [32, 64]
}
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_iter=10, scoring='accuracy', cv=3, random_state=42)
random_search_result = random_search.fit(X_train, y_train)
```

Image 19. Convolutional Neural Network model building with hyperparameters.

It evaluates the best model found during the tuning on the test set and calculates and prints the confusion matrix and classification report for the test set predictions.



```
# Mejor modelo
best_model = random_search_result.best_estimator_.model_
print(f"Best Parameters: {random_search_result.best_params_}")

# Evaluar el mejor modelo
loss, accuracy = best_model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy:.2f}')

# Matriz de confusión y reporte de clasificación
y_pred = best_model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

conf_matrix = confusion_matrix(y_true_classes, y_pred_classes)
print("Confusion Matrix:")
print(conf_matrix)

class_report = classification_report(y_true_classes, y_pred_classes, target_names=['Alzheimer', 'Healthy', 'Other'])
print("Classification Report:")
print(class_report)

# Visualización de la matriz de confusión
ConfusionMatrixDisplay(conf_matrix, display_labels=['Alzheimer', 'Healthy', 'Other']).plot()
plt.title('Confusion Matrix')
plt.show()

# Guardar el modelo y el historial del entrenamiento
model_save_path = 'eeg_model_best.h5'
best_model.save(model_save_path)
print(f'Model saved to {model_save_path}')

history_save_path = 'training_history.json'
with open(history_save_path, 'w') as f:
    json.dump(random_search_result.best_estimator_.model_.history.history, f)
print(f'Training history saved to {history_save_path}')
```

Image 20. Best model evaluation, confusion matrix and classification report.

With the use of the library Matplotlib, the model visualizes the confusion matrix and lastly saves the trained model and its training history to files for future use. In summary, this last cell inputs the data, creates the CNN model, trains with hyperparameter tuning, evaluates, and saves the model and training history.

Implemented Methodology

The research project adopts a quantitative research design that aims to analyze EEG data in order to discern patterns indicative of mental health tendencies. Quantitative research involves the systematic collection and analysis of numerical data with the aim of uncovering fundamental relationships or trends. In this study, MATLAB serves as the primary computational tool for data preprocessing, and Python is used for the processing. An environment where '*Jupyter Notebook*' is available is needed. MATLAB facilitates preprocessing of EEG signals, encompassing tasks such as artifact removal, filtering, epoch, and normalization. Subsequently, Python is utilized for more advanced analysis, feature extraction, classification, and predictive modeling. Together, these instruments allow for a quantitative understanding of EEG data in the context of mental health research.

Instruments and technologies play a pivotal role in the project's execution, with MATLAB serving as the primary instrument for data preprocessing and Python as the environment for advanced analysis. MATLABs functionalities are utilized for preprocessing tasks, ensuring the quality and integrity of the EEG data. Python, however, is a flexible and interactive platform for data exploration, analysis, and visualization. Through the integration of these tools, the project utilizes sophisticated methodologies to extract meaningful insights from EEG data. By utilizing a combination of preprocessing techniques and advanced algorithms, the research aims to uncover subtle yet significant patterns associated with mental health conditions such as Alzheimer, epilepsy, dementia, among others, thereby facilitating the advancement of diagnostic approaches in the field.

The research follows a systematic process from data collection to analysis, emphasizing rigorous methodologies and robust practices. The EEG data is initially gathered from renounced databases. The subsequent preprocessing steps in MATLAB improve the quality of the EEG signals and extract relevant features relevant to mental health analysis. Python is then used to apply a range of algorithms and models to the pre-processed data, using techniques such as machine learning and statistical analysis to discern patterns indicative of mental health tendencies. The results undergo meticulous interpretation and validation to ascertain the reliability and efficacy of the methodology, providing valuable insights into the potential application of EEG-based diagnostics in mental health assessment.

Adaptive Resilience

Throughout the development of this project, several significant challenges emerged, each impacting the project's progress and requiring adaptive strategies to overcome.

Firstly, the electroencephalograms (EEGs) available online were in various formats, necessitating the development of multiple scripts to handle these different formats. This issue required additional time and resources to ensure compatibility and accurate data preprocessing across all formats.

Another major challenge was the limited availability of references and documentation on using the EEGLAB library in MATLAB. Addressing errors and issues in the code required extensive research and problem-solving, compounded by the scarcity of experts in EEGLAB. The few available experts, primarily busy medical professionals, were inaccessible, further complicating troubleshooting efforts.

Additionally, the preprocessing of data in MATLAB followed by analysis in Python was a time-intensive process. Both preprocessing and analysis involved complex procedures, and each attempt to run the code was lengthy. Consequently, debugging and verifying the corrections made to the code was a prolonged process, delaying overall progress.

Moreover, the project faced limitations due to an insufficient amount of data to develop a highly accurate model. This constraint highlighted the need for more comprehensive datasets to enhance the model's precision.

Engineering Innovation

An automated and precise analysis of EEGs can be extremely beneficial for diagnosing Alzheimer's disease. Its medical contribution has an unmeasured impact on today's healthcare system but a few researchers have proved that, in the future, the use of artificial intelligence and machine learning techniques will be exponential. Diagnosing neurological disorders has been a difficult task for physicians although, it seems promising that applying signal-processing training for automated systems could derive results that would make better clinical decisions.

The process has involved the collection of EEG data from patients diagnosed with Alzheimer's, healthy individuals, and other neurological disorders. Massive access to data is the main concern to progress with machine learning processes since the data forms the basis for training the AI model. According to the Alzheimer's Association (2019), the population of Alzheimer's disease registered patients in the United States is 5.8 million Americans, of which 5.6 million people are age 65 and older. Thus, it is essential to understand the early detection of the disease, by contributing to a fast method using machine learning.

Financial Feasibility

The estimate cost that this project will have in the case of develop it completely, can be divided in the next sections, the initial expense, operational costs, and other relevant outlays:

Costs

- Initial expense
  - Develop: This includes primary software and hardware development, machine learning model training, and initial data collection. In the case of this project, the approximate cost is in between $10,000 and $50,000 dollars annually
  - Licensing: for the necessary software licenses and subscription for AI tools it is estimated that the cost will be approximately $10,000 dollars a month, given the fact that it is a project that uses deep learning algorithms,
  - Personnel: The salary for an ML consultant usually reaches $5,000 to $7,000 dollars per project. However, a development team in Latin America costs $12 dollars per hour per member, so if it is estimated that the project takes 3 months in the initial phase it would be 3,200, so the price paid to each member would be 36,000.
- Operational costs:
  - Updates: Regular updates and maintenance of the ML model and system infrastructure, it is estimated an annual cost of $100,000.
  - Date: For acquiring new data for continuous model improvement, it is estimated at $50,000 annually.

  - Personnel: Taking into consideration that the team that will give the maintenance will work 160 hours per person annually, that means the estimate cost of the personnel will be $1,920 dollars
- Other relevant outlays:
  - Training and support: To provide training to end users, as well as taking into account customer service, it is estimated that $50,000 should be invested annually
  - Marketing and dissemination: To make the machine learning system known to healthcare providers and potential clients, an investment of $50.00 per year is estimated.

Projected Benefits

- Income streams:

Subscription: Offer access to private hospitals and clinics through the use of the prediction system, offering different subscriptions based on the different features that will be offered:

- Basic: Includes access to standard predictions and basic system functionality.
- Premium: Offers advanced features such as detailed analytics, priority support, and real-time updates.

It is estimated that, with these subscription models, a significant annual income can be generated. Subscriptions enable recurring, predictable revenue, making it easier to financially plan and invest in ongoing system improvements.

- Estimated income per basic subscription: Approximately $500,000 annually.

- Estimated income per premium subscription: Approximately $1,000,000 annually.

Partnerships: Establish partnerships with pharmaceutical companies and research institutions for exclusive use of the prediction model, projecting revenues of $300,000 per year.

In this project, there is solid financial viability, since it is supported by several factors. The first is the growing demand in the market for diagnostic tools for the healthcare area, particularly for the detection of Alzheimer's as the world's population ages and the incidence of Alzheimer's increases. In addition, this machine learning system can be scaled to handle larger volumes of data and more users, which allows growth in both national and international markets, as technology has presented advances in diagnostic accuracy, increasing reliability. Finally, there are partnership opportunities with research institutions that can generate additional revenue streams and integrate broader medical solutions.

Potential Financial Risks and Mitigation Strategies

Some risks were detected in the project, such as high investment, which can be mitigated by ensuring financing through grants, risk capital and strategic partnerships to distribute the financial burden. Another risk found would be the change in medical and data privacy regulations. To do this, a team specialized in this type of regulation must be in place and that guarantees compliance with all of these, however, this would be long-term.

Outcome Presentation

The product that was obtained from the work done previously was a classification report, which shows the performance of the machine learning model that classifies those EEGs into different categories according to their characteristics to create a deductive model that differs between a patient with Alzheimer's and a healthy one

For the initial expectations, ideal performance by the machine learning model was planned, however, as can be seen in Image 1, the general precision of the model is low.

The next steps for this model are obtaining more EEGs that help the accuracy of the model, as well as the creation of an interactive interface that helps the visualization of the results for users, given the current visualization of the results seen in Image 21.

Image 21. Confusion Matrix

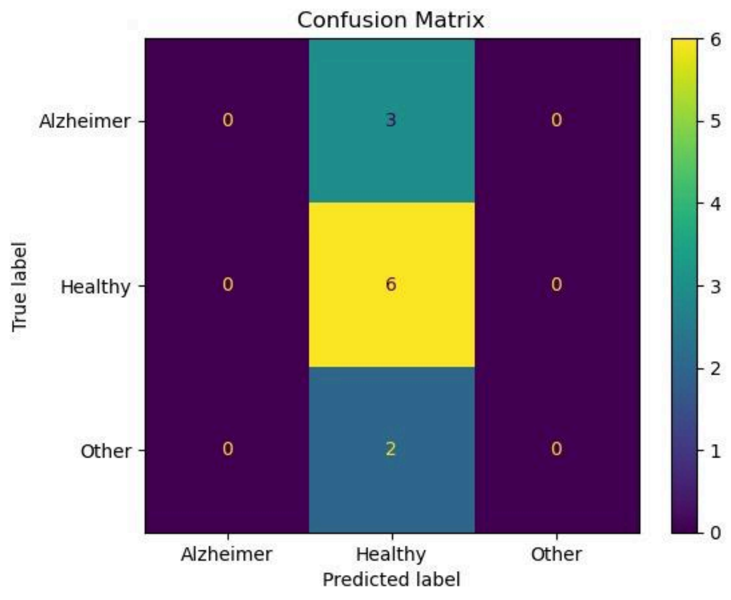
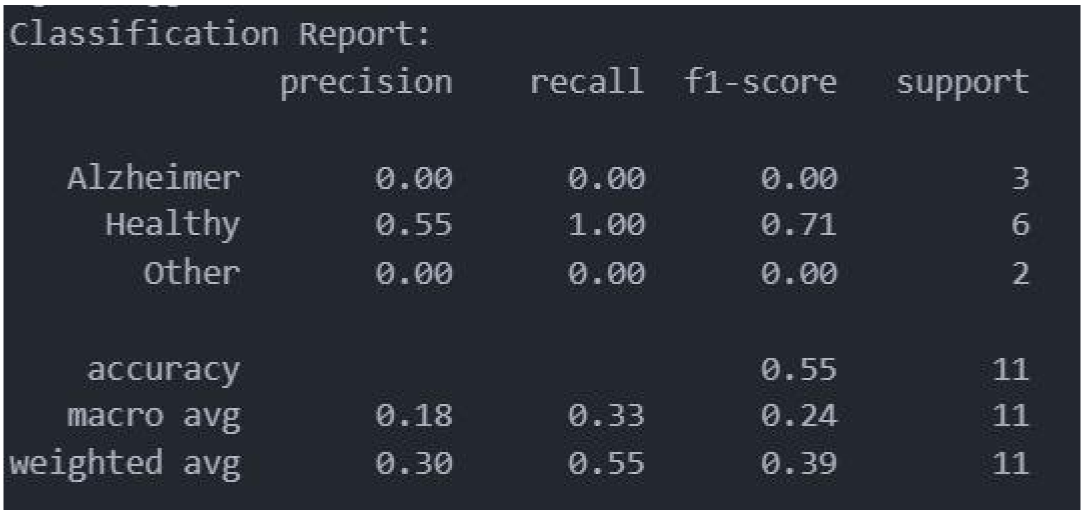


Image 21. Graphic visualization of the results

Results Analysis

The results obtained in this project are shown in the following image.

Image 22. Result table



```
Classification Report:
              precision    recall  f1-score   support

   Alzheimer       0.00      0.00      0.00         3
     Healthy       0.55      1.00      0.71         6
       Other       0.00      0.00      0.00         2

    accuracy                           0.55        11
   macro avg       0.18      0.33      0.24        11
weighted avg       0.30      0.55      0.39        11
```

Image 22. Shows the results table of the Python code that performs the predictive model

In this image, we see the precision column, which tells us about the proportion of positive predictions that are actually correct, so a low-value model frequently labels healthy individuals as having Alzheimer's, as is shown in image 22. In addition, we can also observe the recall column which speaks of the proportion of current positive cases that are correctly identified by the model, which implies that if the value is low, as is shown, the model misses many individuals with Alzheimer's.

The f1-socre column shows the model's performance, where you can see if the model struggles with false positives and negatives in case the value is low. Finally, the support column shows how many encephalograms per category are loaded.

With this we can see that the predictive model still does not work in the expected or adequate way, however this is because there is not a large amount of EEG data with Alzheimer's with which the model can be trained to detect this disease, that is why little accuracy is observed.

References

Alzheimer's Association. (2019, March 1). Alzheimer's disease facts and figures. *Alzheimer's & dementia*, 15(3), 330. https://doi.org/10.1016/j.jalz.2019.01.010

Brody, H. (2011, July 13). Alzheimer's disease. Nature, 475(7355), S1-S1 https://doi.org/10.1038/475S1a

Coop, R. (2021, May 20). What is the Cost to Deploy and Maintain a Machine Learning Model? PhData. https://www.phdata.io/blog/what-is-the-cost-to-deploy-and-maintain-a-machine-learning-model/

Future Processing. (2024, March 27). AI pricing: how much does AI cost in 2024? | Future Processing. Technology & Software Development Blog | Future Processing. https://www.future-processing.com/blog/ai-pricing-is-ai-expensive/

ITRex. (2023, October 3). Machine Learning Costs: Price Factors and Real-World Estimates. Hackernoon.com. https://hackernoon.com/machine-learning-costs-price-factors-and-real-world-estimates

KADENA TATE. (2023, September 5). Maximizing Growth with Subscription Business Models in Healthcare. KADENA TATE. https://www.kadenatate.com/blog/maximizing-growth-with-subscription-business-models-in-healthcare