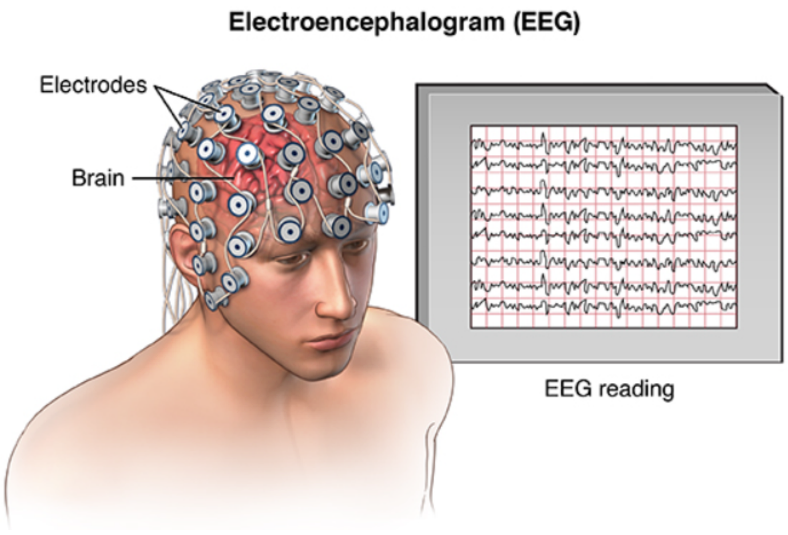
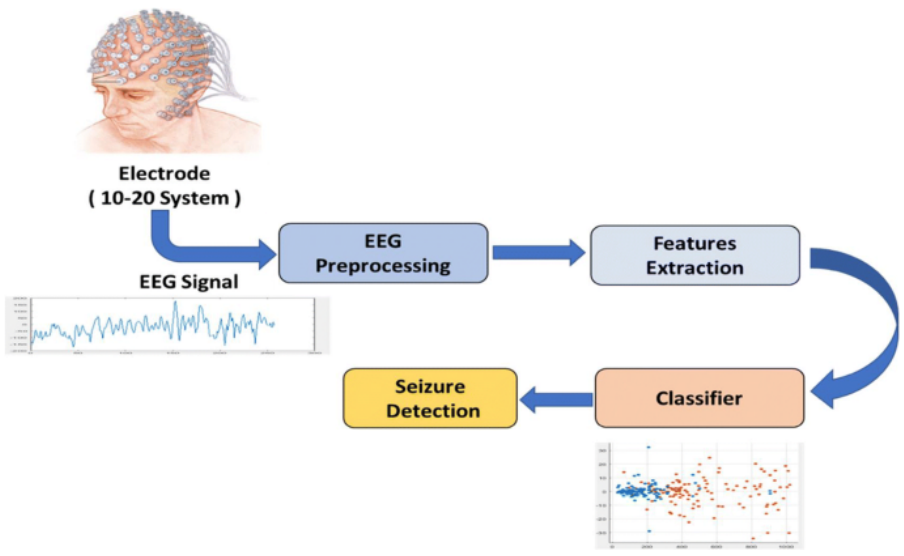
EEG MODEL CLASSIFICATION

IE6400 FOUNDATIONS DATA ANALYTICS ENGINEERING FALL 2023





Project Report Group Number 5:

Pooja Laxmi Sankarakameswaran(002278676)
Rishwanth Reddy Yadamakanti(002697773)
Sruthi Gandla(0028515998)
Tejesvani Muppara Vijayaram(002299364)
Naresh Gajula(002648708)

INTRODUCTION

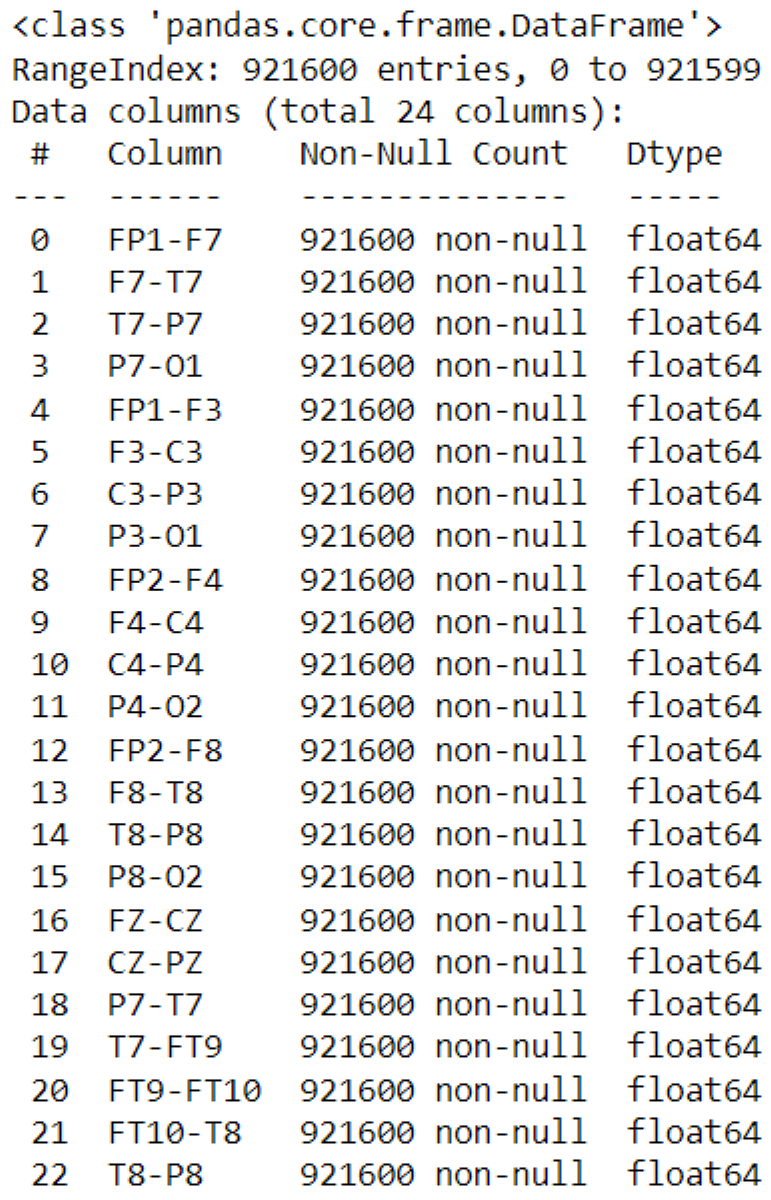
Epilepsy, a neurological disorder characterized by recurrent seizures, poses a significant challenge to millions worldwide. These unpredictable and often life-altering events can impact an individual's quality of life, hindering their daily activities and social interactions. In the pursuit of advancing epilepsy management, the integration of cutting-edge technologies has emerged as a beacon of hope.

Among these technologies, Electroencephalography (EEG) has proven to be a pivotal tool in understanding the dynamic electrical activity of the brain. Harnessing the power of EEG for seizure prediction has become a focal point in research, offering a potential breakthrough in anticipating and mitigating the impact of epileptic seizures.

This endeavor requires a multidisciplinary approach, bringing together neuroscientists, engineers, and medical professionals to decipher the intricate patterns and signals embedded within EEG data. As we delve into the realm of EEG seizure prediction, this exploration seeks to unravel the complexities of the brain's electrical landscape, aiming to provide individuals with epilepsy and their caregivers a proactive means of seizure management.

This introduction sets the stage for an in-depth exploration of EEG-based seizure prediction, delving into the technological advancements, challenges, and the transformative potential it holds for the lives of those affected by epilepsy. As we navigate the landscape of neurological research, the goal is to pave the way for innovative solutions that not only enhance our understanding of epilepsy but also empower individuals to reclaim control over their lives.

DATA PREPROCESSING

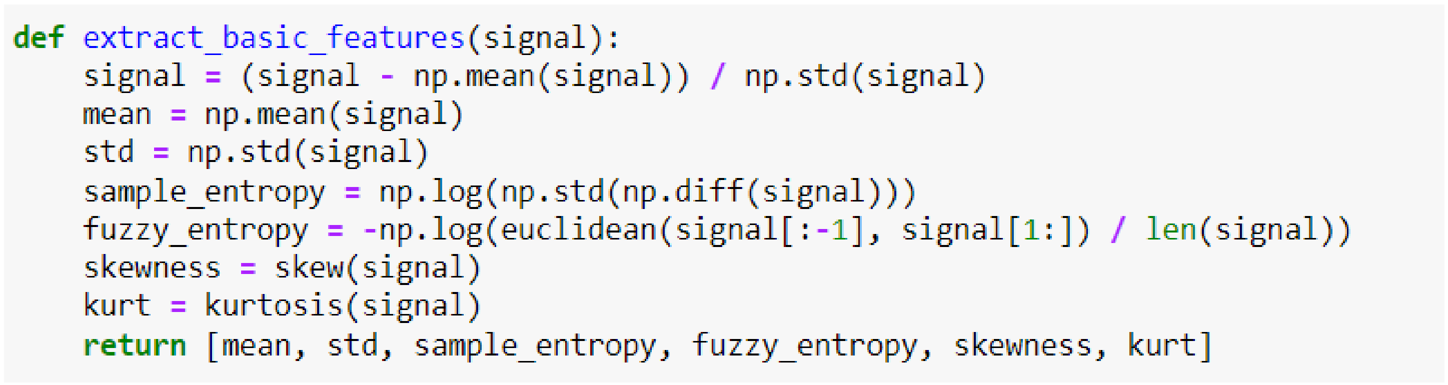


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 921600 entries, 0 to 921599
Data columns (total 24 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   FP1-F7    921600 non-null  float64
 1   F7-T7     921600 non-null  float64
 2   T7-P7     921600 non-null  float64
 3   P7-O1     921600 non-null  float64
 4   FP1-F3    921600 non-null  float64
 5   F3-C3     921600 non-null  float64
 6   C3-P3     921600 non-null  float64
 7   P3-O1     921600 non-null  float64
 8   FP2-F4    921600 non-null  float64
 9   F4-C4     921600 non-null  float64
 10  C4-P4     921600 non-null  float64
 11  P4-O2     921600 non-null  float64
 12  FP2-F8    921600 non-null  float64
 13  F8-T8     921600 non-null  float64
 14  T8-P8     921600 non-null  float64
 15  P8-O2     921600 non-null  float64
 16  FZ-CZ     921600 non-null  float64
 17  CZ-PZ     921600 non-null  float64
 18  P7-T7     921600 non-null  float64
 19  T7-FT9    921600 non-null  float64
 20  FT9-FT10  921600 non-null  float64
 21  FT10-T8   921600 non-null  float64
 22  T8-P8     921600 non-null  float64
```

Since all 23 labels do not contain any null there is no need for null handling. Furthermore, all the signal readings are float values and do not need any datatype handling either.

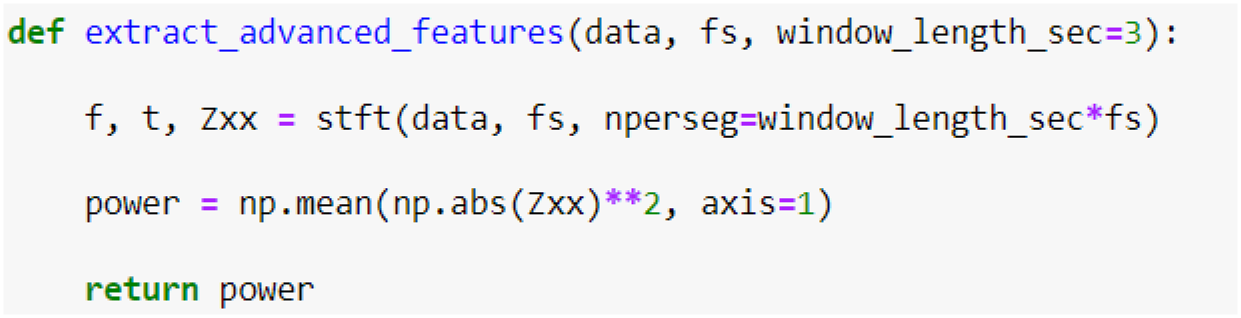
FEATURE EXTRACTION

Feature extraction is done with the help of defining functions that each perform their own set of tasks to arrive at the final result.



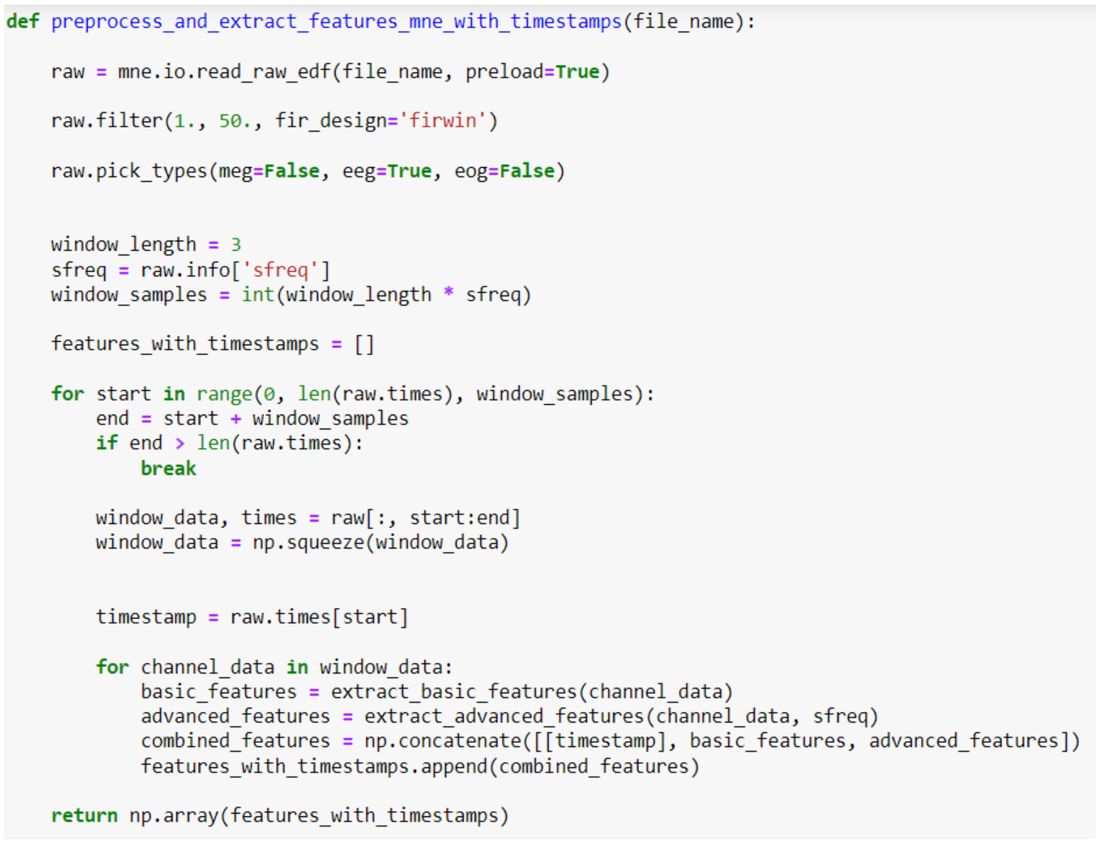
```python
def extract_basic_features(signal):
    signal = (signal - np.mean(signal)) / np.std(signal)
    mean = np.mean(signal)
    std = np.std(signal)
    sample_entropy = np.log(np.std(np.diff(signal)))
    fuzzy_entropy = -np.log(euclidean(signal[:-1], signal[1:]) / len(signal))
    skewness = skew(signal)
    kurt = kurtosis(signal)
    return [mean, std, sample_entropy, fuzzy_entropy, skewness, kurt]
```

The code snippet above creates the function **extract_basic_features** extracts time-domain features such as Mean, Standard Deviation, Entropy, Skewness and Kurtosis.



```python
def extract_advanced_features(data, fs, window_length_sec=3):

    f, t, Zxx = stft(data, fs, nperseg=window_length_sec*fs)

    power = np.mean(np.abs(Zxx)**2, axis=1)

    return power
```

Similarly, the function **extract_advanced_features** extracts advanced features by passing the frequency values and returns the power.



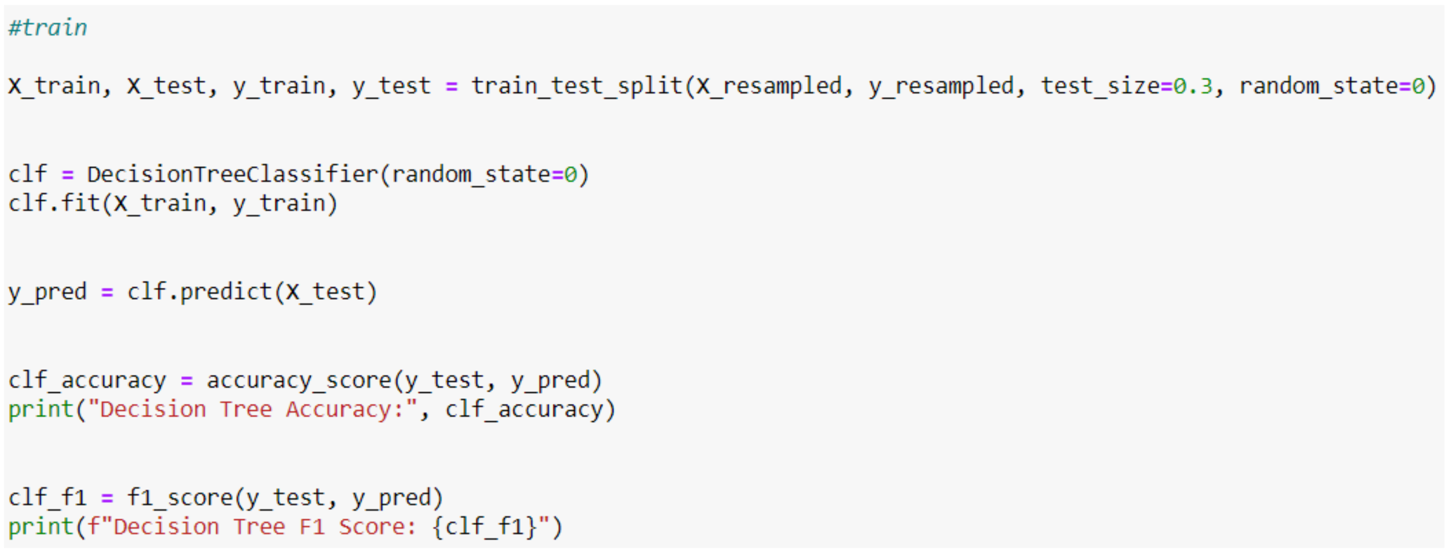
```python
def preprocess_and_extract_features_mne_with_timestamps(file_name):

    raw = mne.io.read_raw_edf(file_name, preload=True)

    raw.filter(1., 50., fir_design='firwin')

    raw.pick_types(meg=False, eeg=True, eog=False)


    window_length = 3
    sfreq = raw.info['sfreq']
    window_samples = int(window_length * sfreq)

    features_with_timestamps = []

    for start in range(0, len(raw.times), window_samples):
        end = start + window_samples
        if end > len(raw.times):
            break

        window_data, times = raw[:, start:end]
        window_data = np.squeeze(window_data)


        timestamp = raw.times[start]

        for channel_data in window_data:
            basic_features = extract_basic_features(channel_data)
            advanced_features = extract_advanced_features(channel_data, sfreq)
            combined_features = np.concatenate([[timestamp], basic_features, advanced_features])
            features_with_timestamps.append(combined_features)

    return np.array(features_with_timestamps)
```

The code above captures the function **preprocess_and_extract_features_mne_with_timestamps** extracts the time-domain and advanced features for every channel based on the timestamps i.e the duration in which the signals were recorded.

<u>MODEL SELECTION & DATA SPLITTING</u>

The data is split into training, test and validation sets as they are being prepared under various models so their performances can be compared. The models chosen for EEG Classification are Decision Tree Classifier and Convolutional Neural Networks (CNNs).

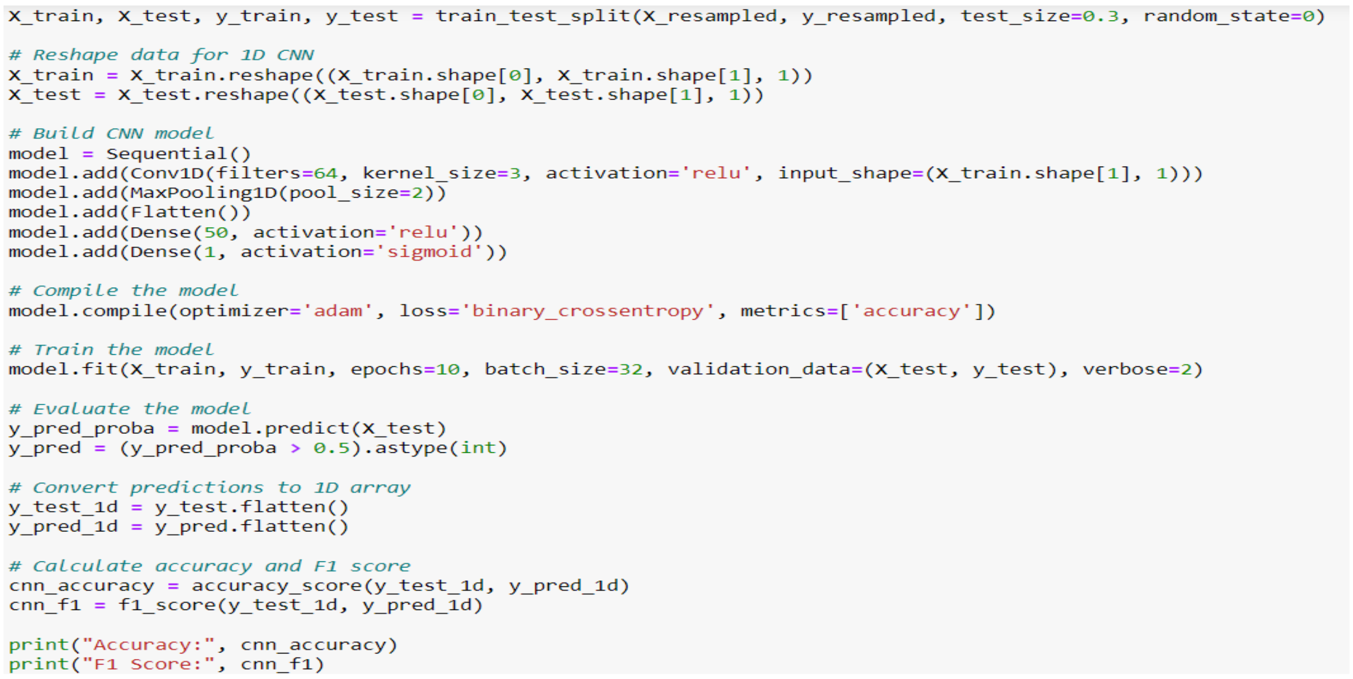
DECISION TREE CLASSIFIER



```python
#train

X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=0)


clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)


clf_accuracy = accuracy_score(y_test, y_pred)
print("Decision Tree Accuracy:", clf_accuracy)


clf_f1 = f1_score(y_test, y_pred)
print(f"Decision Tree F1 Score: {clf_f1}")
```

The above code snippet applies the DecisionTreeClassifier and records the accuracy and F1 scores of the model.

CONVOLUTIONAL NEURAL NETWORKS(CNNs)

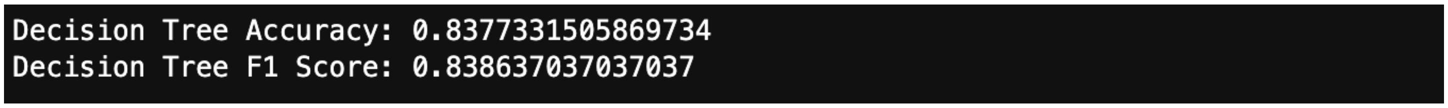


```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=0)

# Reshape data for 1D CNN
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Build CNN model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test), verbose=2)

# Evaluate the model
y_pred_proba = model.predict(X_test)
y_pred = (y_pred_proba > 0.5).astype(int)

# Convert predictions to 1D array
y_test_1d = y_test.flatten()
y_pred_1d = y_pred.flatten()

# Calculate accuracy and F1 score
cnn_accuracy = accuracy_score(y_test_1d, y_pred_1d)
cnn_f1 = f1_score(y_test_1d, y_pred_1d)

print("Accuracy:", cnn_accuracy)
print("F1 Score:", cnn_f1)
```

The code above applies the CNN model and captures the Accuracy and F1 Score of the model.

MODEL EVALUATION

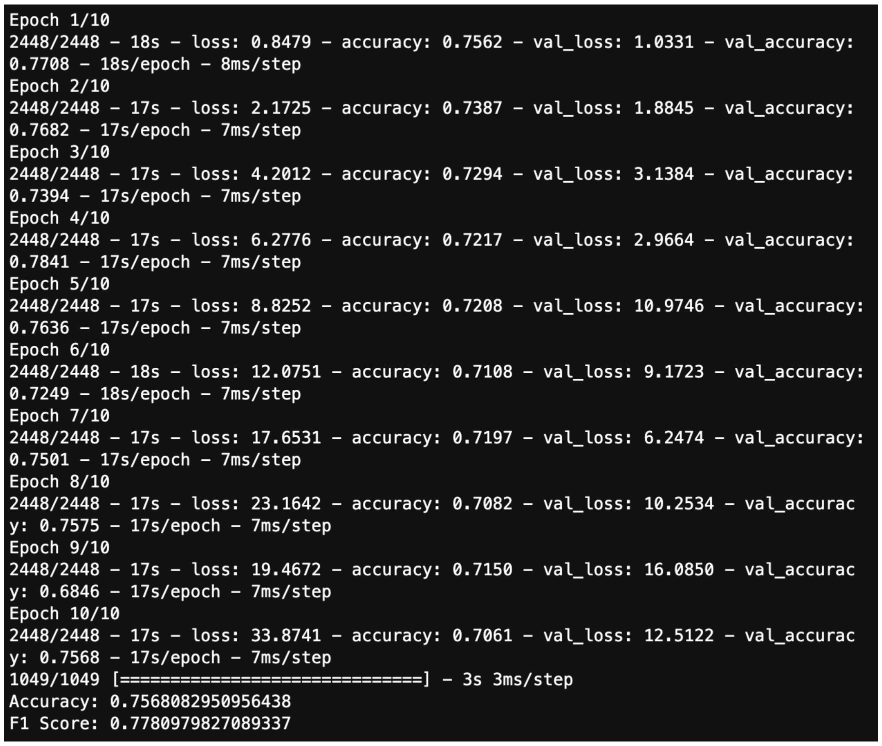
DECISION TREE CLASSIFIER



```
Decision Tree Accuracy: 0.8377331505869734
Decision Tree F1 Score: 0.838637037037037
```

The model records an accuracy of 83.7% and an F1 score of 83.8%

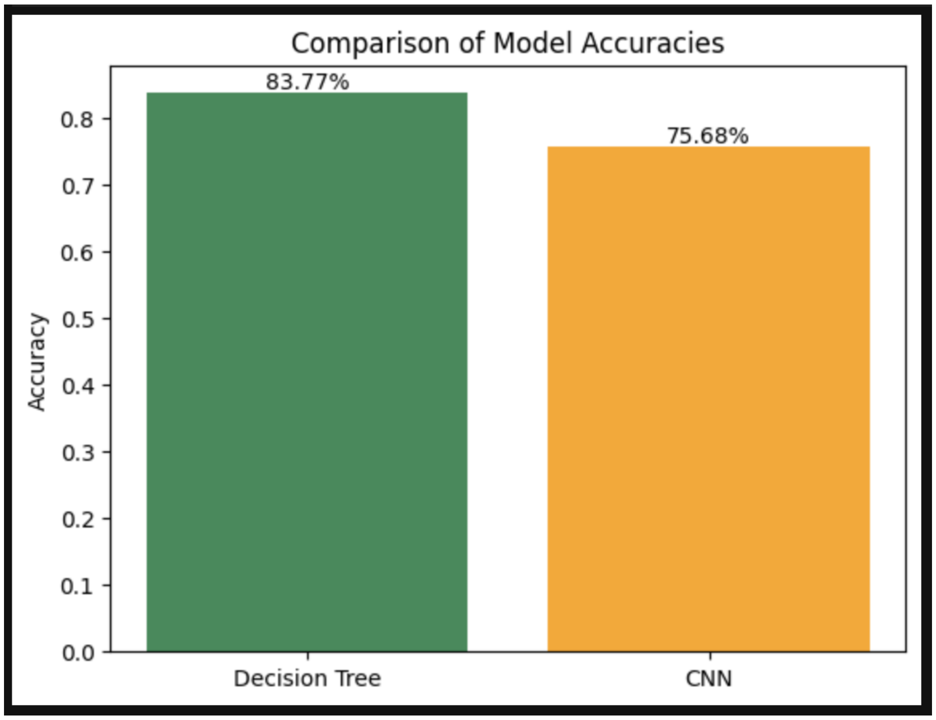
CONVOLUTIONAL NEURAL NETWORKS(CNNs)



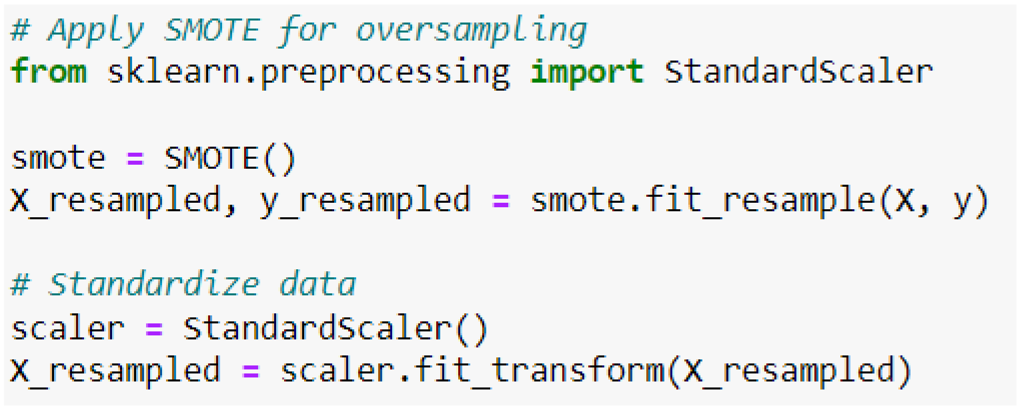
```
Epoch 1/10
2448/2448 - 18s - loss: 0.8479 - accuracy: 0.7562 - val_loss: 1.0331 - val_accuracy:
0.7708 - 18s/epoch - 8ms/step
Epoch 2/10
2448/2448 - 17s - loss: 2.1725 - accuracy: 0.7387 - val_loss: 1.8845 - val_accuracy:
0.7682 - 17s/epoch - 7ms/step
Epoch 3/10
2448/2448 - 17s - loss: 4.2012 - accuracy: 0.7294 - val_loss: 3.1384 - val_accuracy:
0.7394 - 17s/epoch - 7ms/step
Epoch 4/10
2448/2448 - 17s - loss: 6.2776 - accuracy: 0.7217 - val_loss: 2.9664 - val_accuracy:
0.7841 - 17s/epoch - 7ms/step
Epoch 5/10
2448/2448 - 17s - loss: 8.8252 - accuracy: 0.7208 - val_loss: 10.9746 - val_accuracy:
0.7636 - 17s/epoch - 7ms/step
Epoch 6/10
2448/2448 - 18s - loss: 12.0751 - accuracy: 0.7108 - val_loss: 9.1723 - val_accuracy:
0.7249 - 18s/epoch - 7ms/step
Epoch 7/10
2448/2448 - 17s - loss: 17.6531 - accuracy: 0.7197 - val_loss: 6.2474 - val_accuracy:
0.7501 - 17s/epoch - 7ms/step
Epoch 8/10
2448/2448 - 17s - loss: 23.1642 - accuracy: 0.7082 - val_loss: 10.2534 - val_accurac
y: 0.7575 - 17s/epoch - 7ms/step
Epoch 9/10
2448/2448 - 17s - loss: 19.4672 - accuracy: 0.7150 - val_loss: 16.0850 - val_accurac
y: 0.6846 - 17s/epoch - 7ms/step
Epoch 10/10
2448/2448 - 17s - loss: 33.8741 - accuracy: 0.7061 - val_loss: 12.5122 - val_accurac
y: 0.7568 - 17s/epoch - 7ms/step
1049/1049 [==============================] - 3s 3ms/step
Accuracy: 0.7568082950956438
F1 Score: 0.7780979827089337
```

The model recorded an accuracy of 75.6% and an F1 Score of 77.8%.

With the comparison of both the models, we can assess that the Decision Tree Classifier performed better on the patient data. The bar graph below suggests the same.



STRATEGY TO AVOID OVERFITTING



```python
# Apply SMOTE for oversampling
from sklearn.preprocessing import StandardScaler

smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)

# Standardize data
scaler = StandardScaler()
X_resampled = scaler.fit_transform(X_resampled)
```

SMOTE (Synthetic Minority Over-sampling Technique) is a technique used to address class imbalance in machine learning datasets. It works by generating synthetic samples for the minority class to balance the class distribution. While SMOTE is effective in improving the performance of models on imbalanced datasets, it does not directly address the issue of overfitting. In fact, if not used carefully, SMOTE can potentially contribute to overfitting.

Here are a few considerations on how to use SMOTE to avoid overfitting:

Use Cross-Validation:
Implement cross-validation techniques, such as k-fold cross-validation, when evaluating your model. This helps to assess its generalization performance across different subsets of the data. Cross-validation provides a more robust estimate of how well the model will perform on unseen data.

Apply SMOTE After Train-Test Split:
Split your dataset into training and testing sets before applying SMOTE. This ensures that the oversampling is performed only on the training set, leaving the test set unaffected. Applying SMOTE before the split might lead to synthetic samples in both the training and test sets, potentially causing data leakage and overestimating the model's performance.

Tune Model Complexity:
Regularize your model appropriately to control its complexity. Overfitting often occurs when a model is too complex and captures noise in the training data. Regularization techniques, such as L1 or L2 regularization, can help mitigate overfitting by penalizing large coefficients.

Monitor Model Performance Metrics:
Keep a close eye on performance metrics such as precision, recall, and F1 score, especially when dealing with imbalanced datasets. These metrics provide insights into how well the model is performing on both the minority and majority classes. Avoid solely relying on accuracy, as it might be misleading in the context of imbalanced datasets.

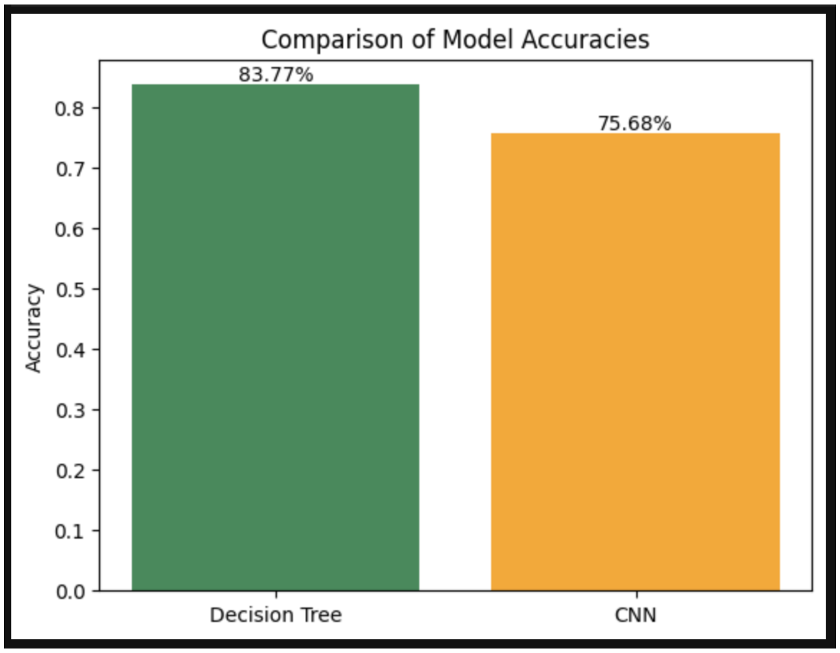
Use SMOTE Sparingly:
While SMOTE can be a valuable tool, it is essential to use it judiciously. Oversampling too aggressively may lead to the generation of synthetic samples that do not accurately represent the underlying patterns in the data, potentially causing overfitting.

In summary, while SMOTE itself does not directly address overfitting, proper application and thoughtful consideration of model complexity, data splitting, and performance metrics can help mitigate the risk of overfitting when using SMOTE in a machine learning model.

RESULTS & VISUALIZATION

The code mentioned below displays the accuracies of each model with their percentages on top of the bars.

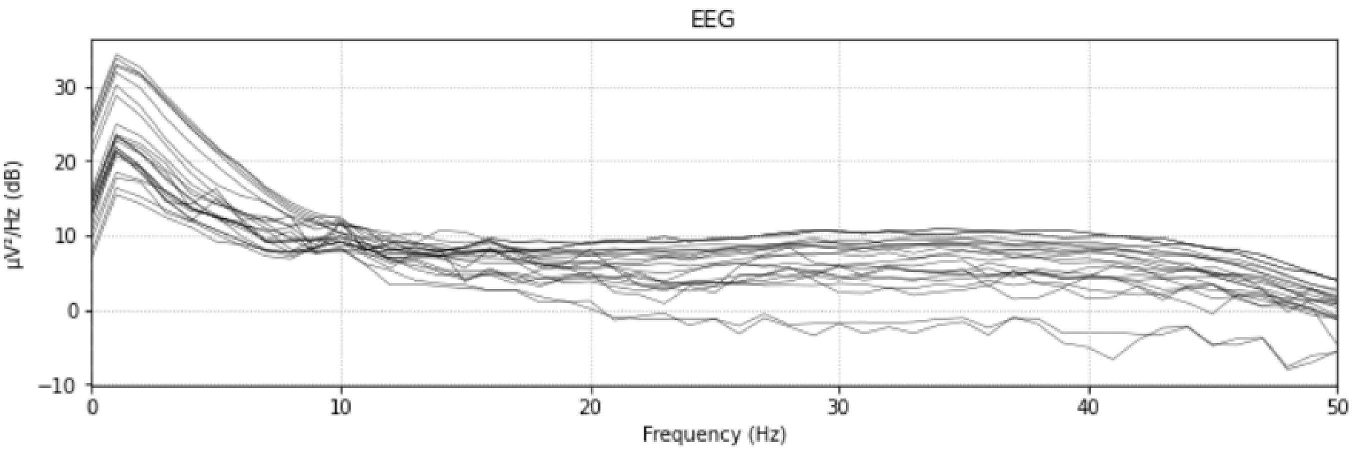
```
# Visualize Accuracies with Percentages
labels = ['Decision Tree', 'CNN']
accuracies = [clf_accuracy, cnn_accuracy]

fig, ax = plt.subplots()
bars = ax.bar(labels, accuracies, color=['seagreen', 'orange'])

# Add percentages on top of the bars
for bar, acc in zip(bars, accuracies):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width() / 2, height, f'{acc:.2%}', ha='center', va='bottom')

plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.show()
```

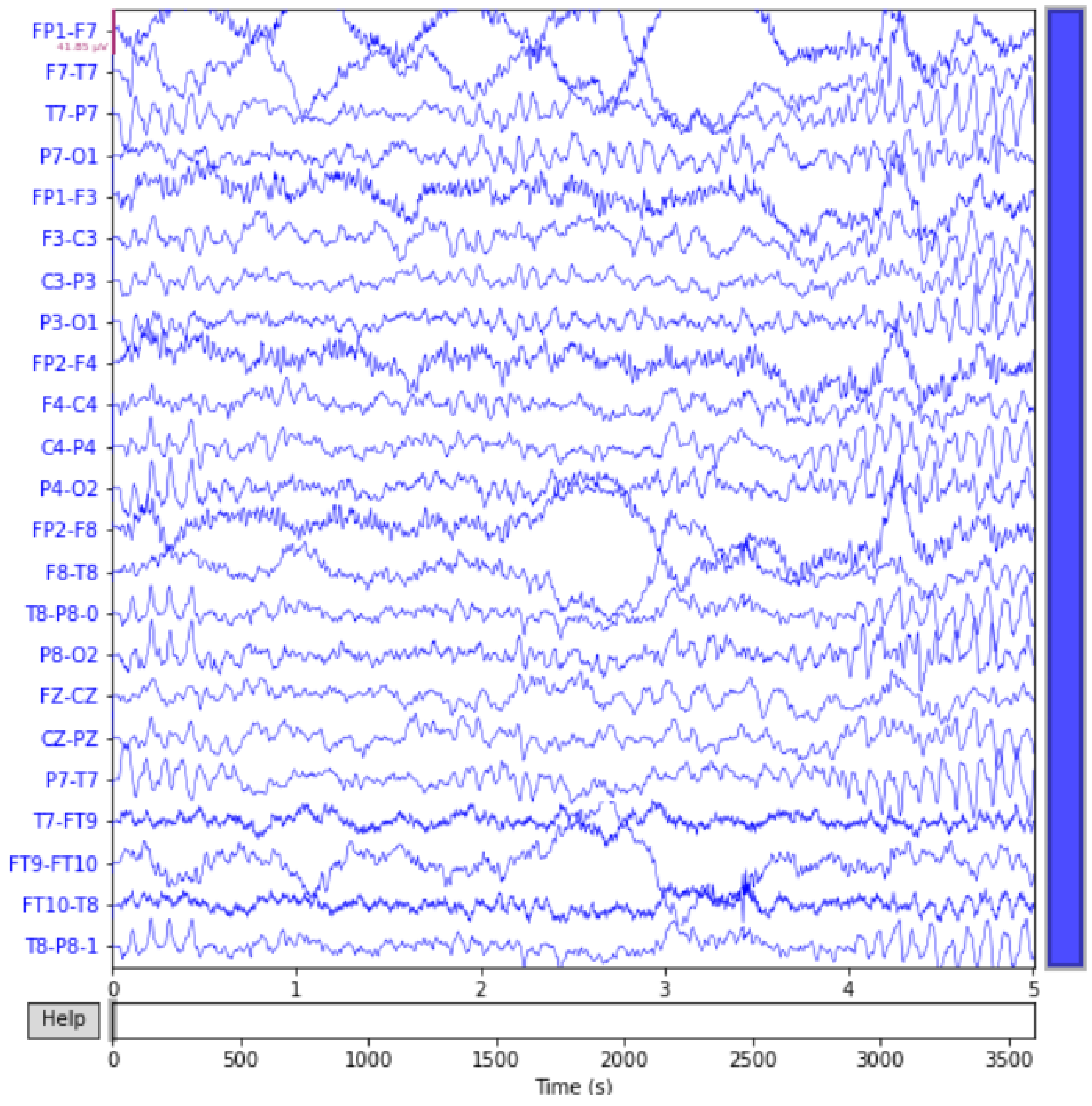


The code below captures the predictions with and without Seizures that were observed in the dataset.

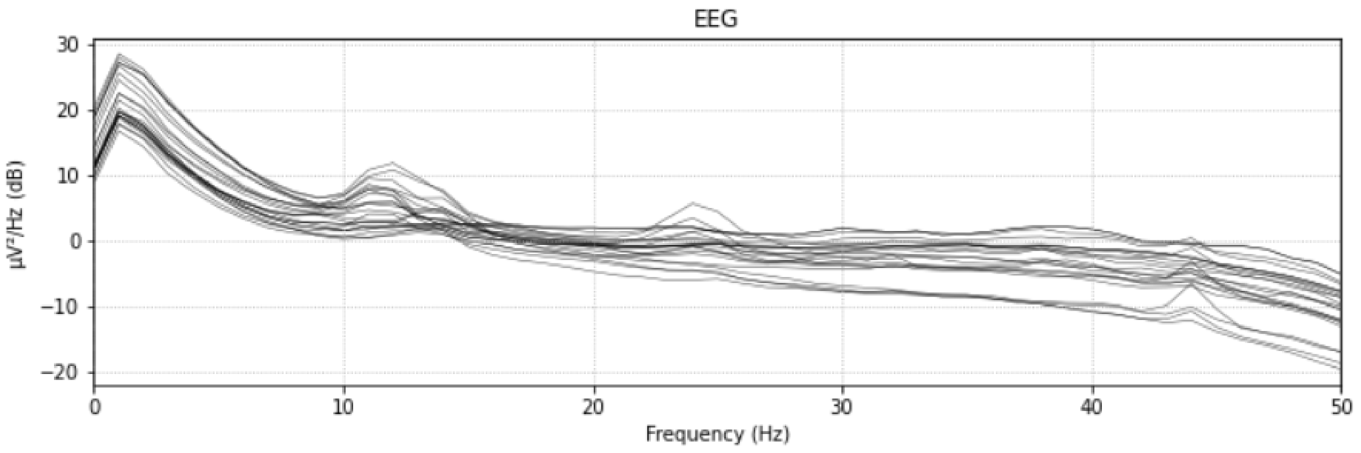
```
import mne
import matplotlib.pyplot as plt

# Specify the file paths
# with seizure
file_path_1 = "smaller_datasets/chb03/chb03_01.edf"
# without seizure
file_path_2 = "smaller_dataset/chb03/chb03_05.edf"

# Read the raw EEG data for both files
raw_1 = mne.io.read_raw_edf(file_path_1, preload=True)
raw_2 = mne.io.read_raw_edf(file_path_2, preload=True)
```

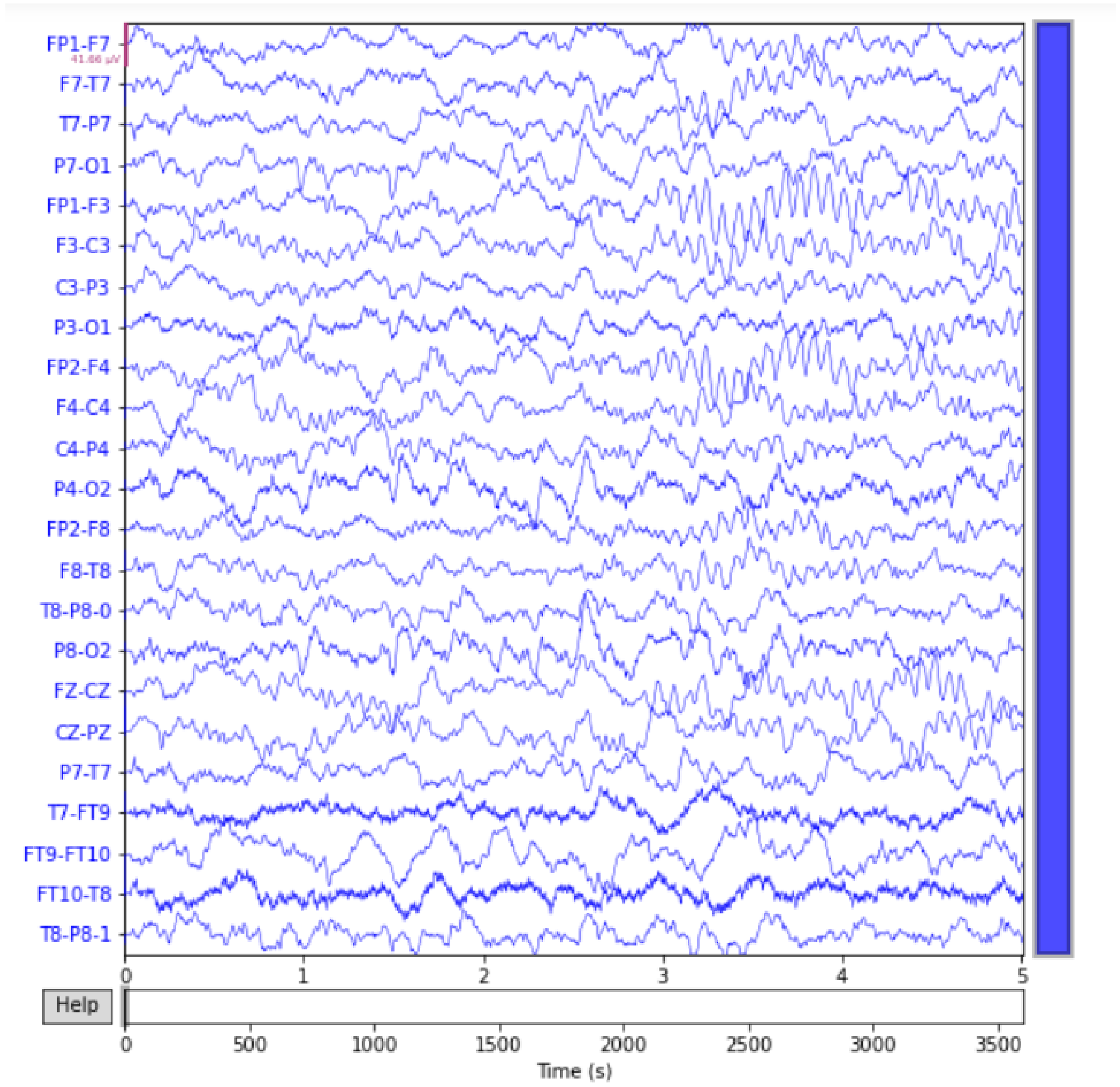
```
# Plot the power spectral density without amplitude (frequency plot) with a specific color
psd_fig1 = raw_1.compute_psd(fmax=50).plot(picks="data", exclude="bads", amplitude=False, color='green')
#psd_fig.set_title('Power Spectral Density')

# Plot the raw EEG data with a different color for each channel
raw_1.plot(duration=5, n_channels=30, color='blue', scalings='auto')
plt.show()
```


EEG
μV²/Hz (dB)
30
20
10
0
−10
0
10
20
30
40
50
Frequency (Hz)



```
# Plot the power spectral density without amplitude (frequency plot) with a specific color
psd_fig2 = raw_2.compute_psd(fmax=50).plot(picks="data", exclude="bads", amplitude=False, color='green')
#psd_fig.set_title('EEG(without seizure)')

# Plot the raw EEG data with a different color for each channel
raw_2.plot(duration=5, n_channels=30, color='blue', scalings='auto')
plt.show()
```


EEG
30
20
10
0
−10
−20
µV²/Hz (dB)
0
10
20
30
40
50
Frequency (Hz)


FP1-F7
41.66 µV
F7-T7
T7-P7
P7-O1
FP1-F3
F3-C3
C3-P3
P3-O1
FP2-F4
F4-C4
C4-P4
P4-O2
FP2-F8
F8-T8
T8-P8-0
P8-O2
FZ-CZ
CZ-PZ
P7-T7
T7-FT9
FT9-FT10
FT10-T8
T8-P8-1
0
1
2
3
4
5
Help
0
500
1000
1500
2000
2500
3000
3500
Time (s)

CONCLUSION

In conclusion, the EEG classification model utilizing a Decision Tree Classifier has demonstrated promising results with an accuracy of 83%. This achievement is a significant step forward in enhancing the field of seizure prediction. The ability to accurately classify EEG patterns holds immense potential for improving the prediction and early detection of seizures in individuals with epilepsy.

The high accuracy achieved by the Decision Tree Classifier suggests that it can effectively discern distinctive features in EEG data associated with seizure events. As a result, this model stands as a valuable tool for identifying patterns indicative of impending seizures, thereby contributing to a more proactive and timely response to mitigate the impact of such events.

The practical implications of this EEG classification model extend to the realm of personalized healthcare. By leveraging machine learning techniques, clinicians and caregivers can gain valuable insights into an individual's unique EEG patterns, allowing for tailored interventions and treatment strategies. The improved accuracy in seizure prediction is poised to enhance the overall quality of life for those affected by epilepsy, offering a potential reduction in the frequency and severity of seizures.

Furthermore, the success of the Decision Tree Classifier underscores the importance of ongoing research and innovation in the intersection of machine learning and neurology. As advancements continue, we can anticipate further refinements to seizure prediction models, potentially incorporating more sophisticated algorithms and leveraging larger datasets for even greater accuracy.

In summary, the EEG classification model built with a Decision Tree Classifier, achieving an 83% accuracy, represents a promising avenue for advancing seizure prediction. This development paves the way for more robust and precise models in the future, holding the potential to revolutionize the management and care of individuals living with epilepsy.