

# Development of a Digital Biomarker using a Neural Network Model to Differentiate Between Long and Short EEG Blinks

**Author:** Hsaine El-Ali

**Date:** 08 Jan 2024

# Contents

1. Abstract:.....	3
2. Introduction.....	3
The Emergence of EEG in Biomedical Applications .....	3
The Challenge of Blink Detection in EEG.....	3
Project Objective .....	3
3. Data Preprocessing .....	4
Initial Data Handling .....	4
Noise Filtering .....	4
4. Feature Extraction .....	5
Time-Domain Features.....	5
Frequency-Domain Features .....	5
5. Model Development.....	6
Introduction .....	6
Data Preparation for Training.....	6
Model Description and Training .....	7
Performance on Test Data.....	8
Results and Analysis.....	8
6. Decision Tree Classifier .....	8
7. Conclusion.....	10
Achievements and Implications .....	10
Advancements in EEG Analysis.....	10

## **1. Abstract:**

This report documents the process and outcomes of a project aimed at distinguishing between long and short blinks using EEG data. The project involved data preprocessing, feature extraction, model building, and evaluation. The final model, a neural network, demonstrated high accuracy on new, unseen data, indicating its potential for real-world applications.

## **2. Introduction**

### *The Emergence of EEG in Biomedical Applications*

Electroencephalography (EEG) is a cornerstone in the field of biomedical research and diagnostics. This non-invasive technique captures electrical activity of the brain, offering invaluable insights into neurological functions and disorders. Its applications span from clinical diagnostics, particularly in epilepsy and sleep disorders, to advanced research in cognitive neuroscience and brain-computer interfaces.

EEG data, characterized by its complex and dynamic nature, encodes rich information about brain activity. The analysis of these signals provides a window into the brain's functioning, allowing researchers and clinicians to interpret various cognitive states and neural responses.

### *The Challenge of Blink Detection in EEG*

Among the myriad of phenomena observable through EEG, the distinction between different types of eye blinks stands out as a subtle yet significant challenge. Eye blinks, particularly long and short blinks, are not only common artifacts in EEG recordings but also carry potential diagnostic value. Long blinks, for instance, might be associated with certain neurological conditions or fatigue levels.

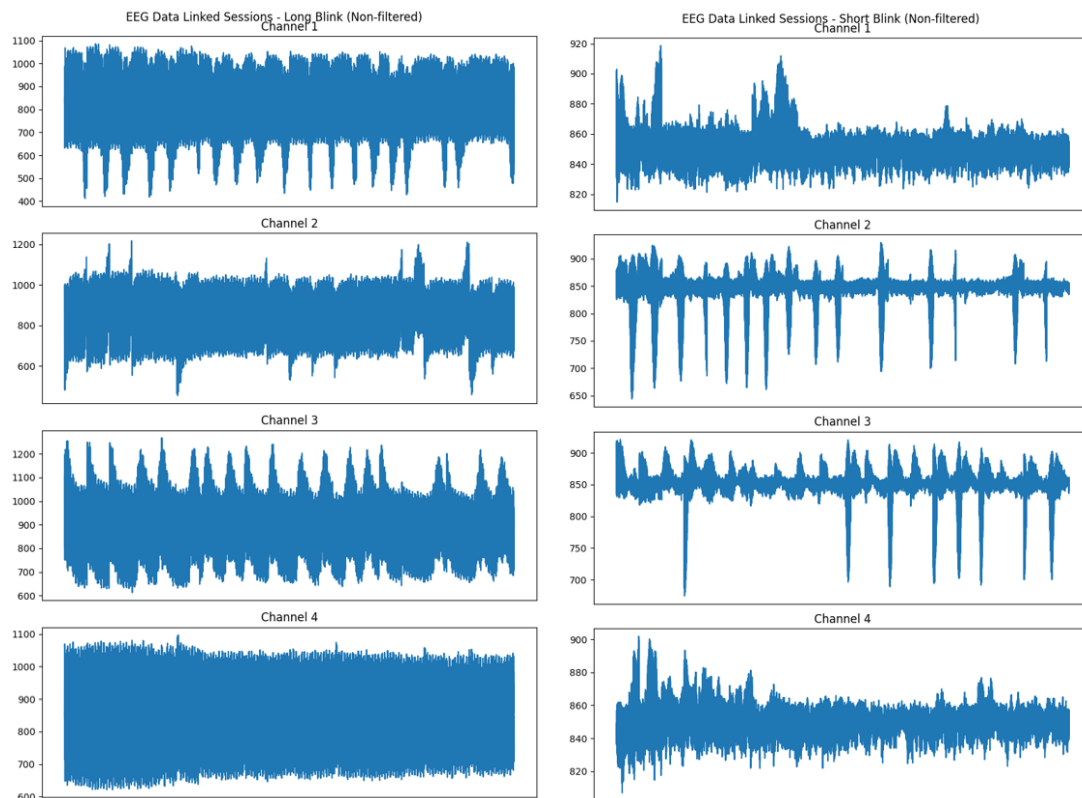
### *Project Objective*

This project aims to develop a computational model based on a digital biomarker extracted from EEG data, to differentiate effectively between long and short blink.

### 3. Data Preprocessing

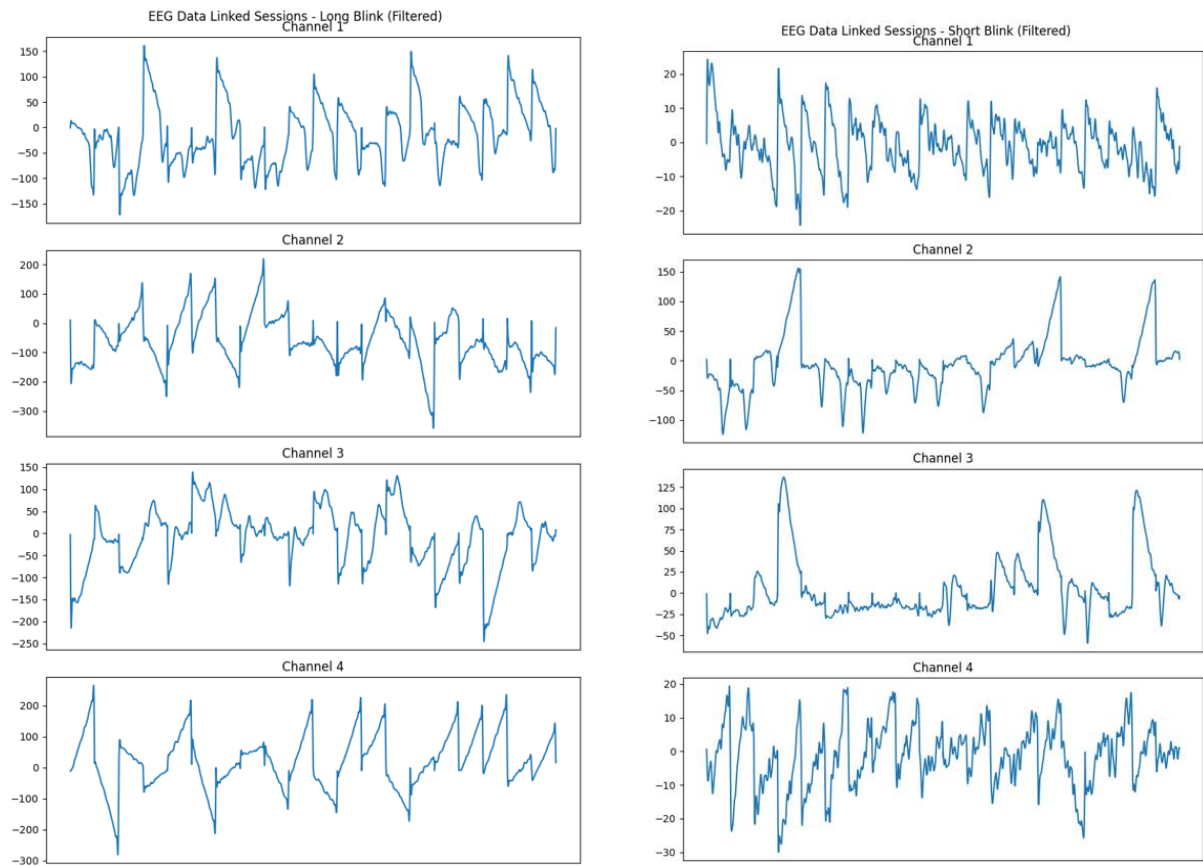
#### *Initial Data Handling*

The EEG data provided comprised two primary datasets: long blinks and short blinks. Each dataset was a collection of EEG signals across multiple channels. The initial step involved flattening the raw data and segmenting it into sessions, each containing a sequence of 510 data points. This segmentation was vital in order to properly display each channel.



#### *Noise Filtering*

A critical preprocessing step was the application of a bandpass filter. With a lower frequency of 0.1 Hz and a higher frequency of 5.0 Hz, this filter was instrumental in removing noise and irrelevant frequency components from the EEG signals. It ensured the isolation of frequencies most relevant to blink activities, enhancing signal quality for subsequent analysis.



## 4. Feature Extraction

### *Time-Domain Features*

In the time domain, three features were extracted from each session:

1. **Peak-to-Peak Amplitude:** This provided insights into the signal's overall range, crucial for distinguishing blink types.
2. **Mean:** It offered a measure of the central tendency of the EEG signal values.
3. **Variance:** This feature represented the spread of the EEG signal values.

### *Frequency-Domain Features*

Frequency-domain analysis was performed using Welch's method to compute the Power Spectral Density (PSD). From the PSD, two features were derived:

1. **Dominant Frequency:** The frequency with the maximum power in the PSD.
2. **Bandwidth:** The range of frequencies where the signal's power was above half of its maximum.

All extracted features per channel were summed up and an average feature function was computed.

```
Average Features for Long Blink:
Channel_1:
Peak-to-Peak Amplitude: 129.8616612248363
Mean: -20.849668217062895
Variance: 1575.8526414529533
Dominant Frequency: 0.9765625 Hz
Bandwidth: 0.09765625 Hz
```

```
Channel_2:
Peak-to-Peak Amplitude: 172.43904458745726
Mean: -41.5251531327259
Variance: 3249.658251927242
Dominant Frequency: 0.9765625 Hz
Bandwidth: 0.01953125 Hz
```

```
Channel_3:
Peak-to-Peak Amplitude: 101.12089006557608
Mean: 10.307606354403505
Variance: 1077.112039549957
Dominant Frequency: 0.9765625 Hz
Bandwidth: 0.0 Hz
```

```
Channel_4:
Peak-to-Peak Amplitude: 143.8210520027745
Mean: 8.219421664701237
Variance: 2170.2322483996836
Dominant Frequency: 1.015625 Hz
Bandwidth: 0.13671875 Hz
```

```
Average Features for Short Blink:
Channel_1:
Peak-to-Peak Amplitude: 20.194121433649826
Mean: -0.036523084256816975
Variance: 36.634402539262716
Dominant Frequency: 1.2109375 Hz
Bandwidth: 0.5078125 Hz
```

```
Channel_2:
Peak-to-Peak Amplitude: 56.417390102846696
Mean: 0.39942976345353015
Variance: 465.24677498959295
Dominant Frequency: 1.09375 Hz
Bandwidth: 0.390625 Hz
```

```
Channel_3:
Peak-to-Peak Amplitude: 47.081388256270046
Mean: 5.707890401306884
Variance: 268.7592012436157
Dominant Frequency: 1.09375 Hz
Bandwidth: 0.48828125 Hz
```

```
Channel_4:
Peak-to-Peak Amplitude: 18.024944784568888
Mean: -0.791987561083572
Variance: 31.07352839223029
Dominant Frequency: 1.34765625 Hz
Bandwidth: 0.9765625 Hz
```

## 5. Model Development

### *Introduction*

In this project, we developed a machine learning model to differentiate between long and short blinks in electroencephalogram (EEG) data. The approach involved data preprocessing, feature extraction, and the use of a neural network model. Key to the model's success was ensuring that the training and testing data were handled correctly to avoid label leakage and ensure an unbiased evaluation of the model's performance.

### *Data Preparation for Training*

#### 1. Merging and Labeling Data:

- We combined features from both long and short blink data for each of the four EEG channels (**Channel\_1** to **Channel\_4**). This was done using **np.vstack** to vertically stack the features of each channel type, ensuring data from both blink types were included.

- Labels were created with **0** representing short blinks and **1** for long blinks. These labels corresponded to the stacked feature data of each blink type.

## 2. Splitting the Dataset:

- The combined feature set and labels were divided into training and testing datasets using **train\_test\_split** with a 70/30 split. This ensured that 30% of the data was held out for testing the model's performance on unseen data.
- The labels for the training set were converted to one-hot encoding format using **to\_categorical**, a necessary step for classification in neural networks.

### *Model Description and Training*

## 3. Building the Neural Network:

- A Sequential model was constructed with dense layers. The input layer had 64 neurons, and subsequent layers (including hidden layers) were added with 32 neurons each.
- To mitigate overfitting, dropout layers with a dropout rate of 0.5 were introduced. These layers randomly set input units to 0 at each step during training, which helps prevent overfitting.
- The final layer was a dense layer with 2 neurons corresponding to our two classes (short and long blinks), using a 'softmax' activation function for classification.
- The model was compiled with the 'adam' optimizer and 'categorical\_crossentropy' as the loss function.

## 4. Cross-Validation and Training:

- Stratified K-Fold cross-validation with 5 splits was employed to validate the model's effectiveness. This approach ensured that each fold was a good representative of the whole dataset.
- For each fold, the model was trained and validated, allowing us to observe the variation in model performance across different subsets of the training data.

## 5. Model Evaluation and Saving:

- After training, the model's performance was evaluated on the test set. The accuracy, confusion matrix, and a classification report were generated, providing insights into the model's predictive capabilities on unseen data.
- Finally, the trained model was saved to a specified path for future use in making predictions on new data.

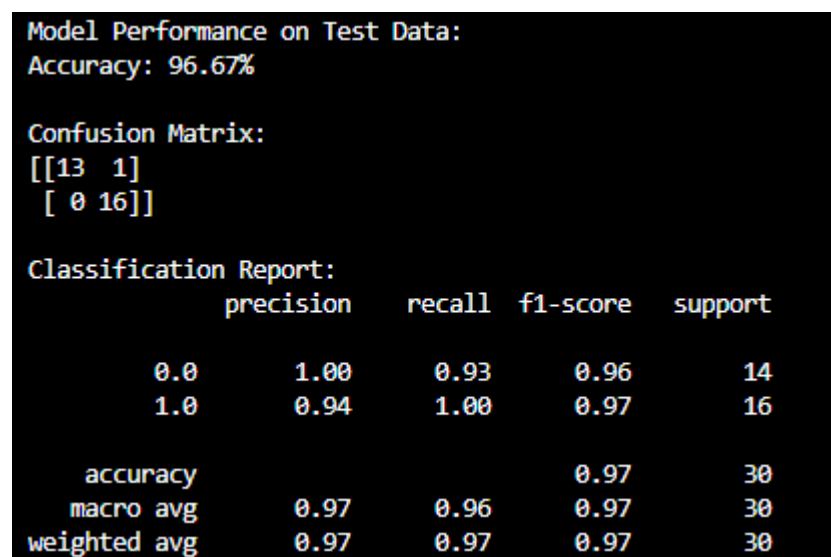
## 6. Model Evaluation and Results

### *Performance on Test Data*

The model was evaluated on a separate test set, accounting for 30% of the total data. This approach ensured unbiased evaluation on unseen data.

### *Results and Analysis*

The model achieved a remarkable 96.67% accuracy, with the confusion matrix showing only one misclassification in the test set. The classification report indicated high precision, recall, and F1-scores for both classes, demonstrating the model's capability to effectively distinguish



```
Model Performance on Test Data:
Accuracy: 96.67%

Confusion Matrix:
[[13  1]
 [ 0 16]]

Classification Report:
              precision    recall  f1-score   support

     0.0         1.00      0.93      0.96         14
     1.0         0.94      1.00      0.97         16

 accuracy          0.97      0.96      0.97         30
 macro avg         0.97      0.96      0.97         30
weighted avg         0.97      0.97      0.97         30
```

## 6. Decision Tree Classifier

A Decision Tree Classifier was implemented using Scikit-learn's **DecisionTreeClassifier** class. The classifier was trained on a set of labeled features extracted from the EEG data. The features for both long and short blinks were combined and used as input variables, while the labels were encoded as binary outputs (0 for short blinks, 1 for long blinks).



The training process involved splitting the data into training and test sets, with a 70-30 split ratio, ensuring that 30% of the data was reserved for model evaluation. The classifier was then fitted to the training data, learning to associate the input features with the correct blink type.

## Model Evaluation

The performance of the trained Decision Tree Classifier was evaluated using the reserved test set. The evaluation metrics included:

- Accuracy: The proportion of correctly predicted instances out of all predictions.
- Confusion Matrix: A table used to describe the performance of a classification model on a set of test data for which the true values are known. It allows visualization of the model's ability to correctly or incorrectly classify instances.
- Classification Report: A summary of the precision, recall, and F1-score for each class.

The model achieved an accuracy of 96.67% on the test data, indicating a high level of predictive performance. The confusion matrix and classification report provided further insight into the model's classification capabilities for each blink type.

Model Performance on Test Data:					
Accuracy: 96.67%					
Confusion Matrix:					
[[17  0]					
[ 1 12]]					
Classification Report:					
	precision	recall	f1-score	support	
0.0	0.94	1.00	0.97	17	
1.0	1.00	0.92	0.96	13	
accuracy			0.97	30	
macro avg	0.97	0.96	0.97	30	
weighted avg	0.97	0.97	0.97	30	

## 7. Conclusion

### *Achievements and Implications*

This project successfully demonstrated the feasibility of using machine learning algorithms to differentiate between long and short blinks in EEG data. Through meticulous data preprocessing, feature extraction, and the development of a robust neural network model, we have paved the way for more accurate EEG analysis in both clinical and research settings.

### *Advancements in EEG Analysis*

The ability to distinguish between various types of blinks represents a significant advancement in EEG data interpretation. This capability enhances the accuracy of neurological assessments and potentially aids in diagnosing specific conditions.

## 8. References

1. Niedermeyer, E., & da Silva, F. L. (Eds.). (2005). *Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins.
2. Teplan, M. (2002). *Fundamentals of EEG measurement*. Measurement Science Review, 2(2), 1-11.
3. Sanei, S., & Chambers, J. A. (2013). *EEG Signal Processing*. John Wiley & Sons.
4. Shambroom, J. R., Fábregas, S. E., & Johnstone, J. (2012). *Validation of an automated wireless system to monitor sleep in healthy adults*. Journal of Sleep Research, 21(2), 221-230.
5. Fatourechi, M., Bashashati, A., Ward, R. K., & Birch, G. E. (2007). *EMG and EOG artifacts in brain computer interface systems: A survey*. Clinical Neurophysiology, 118(3), 480-494.
6. Croft, R. J., & Barry, R. J. (2000). *Removal of ocular artifact from the EEG: A review*. Neurophysiologie Clinique/Clinical Neurophysiology, 30(1), 5-19.
7. Schlögl, A., Keinrath, C., Zimmermann, D., Scherer, R., Leeb, R., & Pfurtscheller, G. (2007). *A fully automated correction method of EOG artifacts in EEG recordings*. Clinical Neurophysiology, 118(1), 98-104.
8. Lindsley, D. B. (1939). *A longitudinal study of the occipital alpha rhythm in normal children: Frequency and amplitude standards*. Journal of Genetic Psychology, 55(1), 197-213.

## 9. Appendix

*Code:*

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate
import ast # For safely evaluating the string
import mne
from scipy.signal import find_peaks
from scipy.signal import welch
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils import to_categorical
from keras.callbacks import History
from sklearn.model_selection import StratifiedKFold
from keras.regularizers import l2

plt.ion() # Turn on interactive mode

# Load the CSV files
long_blink_data = pd.read_csv('C:/dev/digitalbiomarkers/EEG-
data/LongBlink.csv')
short_blink_data = pd.read_csv('C:/dev/digitalbiomarkers/EEG-
data/ShortBlink.csv')

# Display data tables
print("\nTable for LongBlink.csv:")
print(tabulate(long_blink_data.head(), headers='keys', tablefmt='grid'))

print("\nTable for ShortBlink.csv:")
print(tabulate(short_blink_data.head(), headers='keys', tablefmt='grid'))

# Convert string representations to actual lists
long_blink_data['data'] = long_blink_data['data'].apply(ast.literal_eval)
short_blink_data['data'] = short_blink_data['data'].apply(ast.literal_eval)

# Flatten the data
long_blink_values = [value for sublist in long_blink_data['data'].tolist() for
value in sublist]
short_blink_values = [value for sublist in short_blink_data['data'].tolist()
for value in sublist]

# Chunk the data
```

```

long_blink_sessions = [long_blink_values[i:i + 510] for i in range(0,
len(long_blink_values), 510)]
short_blink_sessions = [short_blink_values[i:i + 510] for i in range(0,
len(short_blink_values), 510)]

# Convert to arrays
long_blink_array = np.array(long_blink_sessions)
short_blink_array = np.array(short_blink_sessions)

# Apply bandpass filter
def apply_bandpass_filter(data, l_freq=0.1, h_freq=5.0):
    sfreq = 215.0
    data_filtered = mne.filter.filter_data(data, sfreq, l_freq, h_freq,
method='iir', verbose=False)
    return data_filtered

long_blink_array_filtered = apply_bandpass_filter(long_blink_array)
short_blink_array_filtered = apply_bandpass_filter(short_blink_array)

# Plot sessions function
def plot_linked_sessions(data_array, title):
    fig, axs = plt.subplots(4, 1, figsize=(8, 12))
    for channel_idx in range(4):
        blink_data = np.concatenate([data_array[session_idx * 4 + channel_idx]
for session_idx in range(20)])
        axs[channel_idx].plot(blink_data)
        axs[channel_idx].set_title(f"Channel {channel_idx + 1}")
        axs[channel_idx].set_xticks([])
    plt.tight_layout()
    plt.subplots_adjust(top=0.95)
    fig.suptitle(title)
    plt.draw()

# Plot the data
plot_linked_sessions(long_blink_array, "EEG Data Linked Sessions - Long Blink
(Non-filtered)")
plot_linked_sessions(short_blink_array, "EEG Data Linked Sessions - Short
Blink (Non-filtered)")
plot_linked_sessions(long_blink_array_filtered, "EEG Data Linked Sessions -
Long Blink (Filtered)")
plot_linked_sessions(short_blink_array_filtered, "EEG Data Linked Sessions -
Short Blink (Filtered)")

plt.show(block=True)

# Feature Extraction function

```

```

def extract_features_per_channel(data_array):
    # Initialize feature lists
    features_per_channel = {f"Channel_{i+1}": [] for i in range(4)}

    for session_idx in range(len(data_array) // 4): # 4 channels per session
        for channel_idx in range(4):
            session = data_array[session_idx * 4 + channel_idx]

            # Time domain features

            # 1. Peak-to-peak amplitude
            positive_peaks, _ = find_peaks(session)
            negative_peaks, _ = find_peaks(-session)
            if positive_peaks.size and negative_peaks.size:
                max_peak = max(session[positive_peaks])
                min_peak = min(session[negative_peaks])
                peak_to_peak_amplitude = max_peak - min_peak
            else:
                peak_to_peak_amplitude = 0

            # 2. Mean
            mean_val = np.mean(session)

            # 3. Variance
            variance_val = np.var(session)

            # Frequency domain features

            # Compute the Power Spectral Density
            f, Pxx = welch(session, fs=250.0, nperseg=256)

            # 4. Dominant frequency
            dominant_frequency = f[np.argmax(Pxx)]

            # 5. Bandwidth
            half_power = np.max(Pxx) / 2
            indices = np.where(Pxx > half_power)
            first_index = indices[0][0]
            last_index = indices[0][-1]
            bandwidth = f[last_index] - f[first_index]

            # Append features to the channel's feature list
            features_per_channel[f"Channel_{channel_idx+1}"].append([peak_to_p
eak_amplitude, mean_val, variance_val, dominant_frequency, bandwidth])

    # Convert lists to arrays
    for key in features_per_channel:
        features_per_channel[key] = np.array(features_per_channel[key])

```

```

    return features_per_channel

long_blink_features_per_channel =
extract_features_per_channel(long_blink_array_filtered)
short_blink_features_per_channel =
extract_features_per_channel(short_blink_array_filtered)

# Compute average features function
def compute_average_features_per_channel(features_dict):
    average_features = {}
    for key in features_dict:
        avg_peak_to_peak_amplitude = np.mean(features_dict[key][:, 0])
        avg_mean_val = np.mean(features_dict[key][:, 1])
        avg_variance_val = np.mean(features_dict[key][:, 2])
        avg_dominant_frequency = np.mean(features_dict[key][:, 3])
        avg_bandwidth = np.mean(features_dict[key][:, 4])

        average_features[key] = [avg_peak_to_peak_amplitude, avg_mean_val,
avg_variance_val, avg_dominant_frequency, avg_bandwidth]

    return average_features

# Calculate average features for long and short blinks
long_blink_avg_features_per_channel =
compute_average_features_per_channel(long_blink_features_per_channel)
short_blink_avg_features_per_channel =
compute_average_features_per_channel(short_blink_features_per_channel)

# Print results
print("\nAverage Features for Long Blink:")
for key in long_blink_avg_features_per_channel:
    print(f"{key}:")
    print(f"Peak-to-Peak Amplitude:
{long_blink_avg_features_per_channel[key][0]}")
    print(f"Mean: {long_blink_avg_features_per_channel[key][1]}")
    print(f"Variance: {long_blink_avg_features_per_channel[key][2]}")
    print(f"Dominant Frequency: {long_blink_avg_features_per_channel[key][3]}
Hz")
    print(f"Bandwidth: {long_blink_avg_features_per_channel[key][4]} Hz\n")

print("Average Features for Short Blink:")
for key in short_blink_avg_features_per_channel:
    print(f"{key}:")
    print(f"Peak-to-Peak Amplitude:
{short_blink_avg_features_per_channel[key][0]}")
    print(f"Mean: {short_blink_avg_features_per_channel[key][1]}")
    print(f"Variance: {short_blink_avg_features_per_channel[key][2]}")

```

```

    print(f"Dominant Frequency: {short_blink_avg_features_per_channel[key][3]} Hz")
    print(f"Bandwidth: {short_blink_avg_features_per_channel[key][4]} Hz\n")

# Preparing Dataset

# 1. Merge data from both blink types and label them
all_features = []
for channel in ["Channel_1", "Channel_2", "Channel_3", "Channel_4"]:
    all_features_channel =
np.vstack([short_blink_features_per_channel[channel],
long_blink_features_per_channel[channel]])
    all_features.append(all_features_channel)

all_features = np.hstack(all_features) # Combining features from all channels

labels_short =
np.zeros(short_blink_features_per_channel["Channel_1"].shape[0])
labels_long = np.ones(long_blink_features_per_channel["Channel_1"].shape[0])
all_labels = np.hstack([labels_short, labels_long])

# 2. Split the dataset
X_train, X_test, y_train, y_test = train_test_split(all_features, all_labels,
test_size=0.3)

# Convert labels to one-hot encoding for training
y_train_categorical = to_categorical(y_train)

# 3. Build the Neural Network with modifications
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Cross-validation
kfold = StratifiedKFold(n_splits=5, shuffle=True)
cv_scores = []

for train, val in kfold.split(X_train, y_train):
    y_train_k_categorical = to_categorical(y_train[train])
    history: History = model.fit(X_train[train], y_train_k_categorical,
epochs=50, batch_size=10, validation_data=(X_train[val],
to_categorical(y_train[val])))

```



```

    scores = model.evaluate(X_train[val], to_categorical(y_train[val]),
verbose=0)
    cv_scores.append(scores[1] * 100)
    print(f"Fold Accuracy: {scores[1]*100:.2f}%")

print(f"Mean Accuracy: {np.mean(cv_scores):.2f}%, Standard Deviation:
{np.std(cv_scores):.2f}%")

# 4. Plot the learning curve
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='lower right')
plt.show(block=True)

# 5. Use the trained model to predict on the test set
y_pred_probabilities = model.predict(X_test)
y_pred = np.argmax(y_pred_probabilities, axis=1)

# Compare the predictions with the true labels of the test set
accuracy = accuracy_score(y_test, y_pred)

# Print results
print("\nModel Performance on Test Data:")
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Save the model
model_save_path = 'C:/dev/digitalbiomarkers/blink_model.keras'
model.save(model_save_path)
print("Model saved at", model_save_path)

```

## DECISION TREE CALISSIFIER CODE

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate
import ast # For safely evaluating the string
import mne
from scipy.signal import find_peaks
from scipy.signal import welch
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils import to_categorical
from keras.callbacks import History
from sklearn.model_selection import StratifiedKFold
from keras.regularizers import l2
from sklearn.tree import DecisionTreeClassifier

plt.ion() # Turn on interactive mode

# Load the CSV files
long_blink_data = pd.read_csv('C:/dev/digitalbiomarkers/EEG-
data/LongBlink.csv')
short_blink_data = pd.read_csv('C:/dev/digitalbiomarkers/EEG-
data/ShortBlink.csv')

# Display data tables
print("\nTable for LongBlink.csv:")
print(tabulate(long_blink_data.head(), headers='keys', tablefmt='grid'))

print("\nTable for ShortBlink.csv:")
print(tabulate(short_blink_data.head(), headers='keys', tablefmt='grid'))

# Convert string representations to actual lists
long_blink_data['data'] = long_blink_data['data'].apply(ast.literal_eval)
short_blink_data['data'] = short_blink_data['data'].apply(ast.literal_eval)

# Flatten the data
long_blink_values = [value for sublist in long_blink_data['data'].tolist() for
value in sublist]
short_blink_values = [value for sublist in short_blink_data['data'].tolist()
for value in sublist]

# Chunk the data
```

```

long_blink_sessions = [long_blink_values[i:i + 510] for i in range(0,
len(long_blink_values), 510)]
short_blink_sessions = [short_blink_values[i:i + 510] for i in range(0,
len(short_blink_values), 510)]

# Convert to arrays
long_blink_array = np.array(long_blink_sessions)
short_blink_array = np.array(short_blink_sessions)

# Apply bandpass filter
def apply_bandpass_filter(data, l_freq=0.1, h_freq=5.0):
    sfreq = 215.0
    data_filtered = mne.filter.filter_data(data, sfreq, l_freq, h_freq,
method='iir', verbose=False)
    return data_filtered

long_blink_array_filtered = apply_bandpass_filter(long_blink_array)
short_blink_array_filtered = apply_bandpass_filter(short_blink_array)

# Plot sessions function
def plot_linked_sessions(data_array, title):
    fig, axs = plt.subplots(4, 1, figsize=(8, 12))
    for channel_idx in range(4):
        blink_data = np.concatenate([data_array[session_idx * 4 + channel_idx]
for session_idx in range(20)])
        axs[channel_idx].plot(blink_data)
        axs[channel_idx].set_title(f"Channel {channel_idx + 1}")
        axs[channel_idx].set_xticks([])
    plt.tight_layout()
    plt.subplots_adjust(top=0.95)
    fig.suptitle(title)
    plt.draw()

# Plot the data
plot_linked_sessions(long_blink_array, "EEG Data Linked Sessions - Long Blink
(Non-filtered)")
plot_linked_sessions(short_blink_array, "EEG Data Linked Sessions - Short
Blink (Non-filtered)")
plot_linked_sessions(long_blink_array_filtered, "EEG Data Linked Sessions -
Long Blink (Filtered)")
plot_linked_sessions(short_blink_array_filtered, "EEG Data Linked Sessions -
Short Blink (Filtered)")

plt.show(block=True)

# Feature Extraction function

```

```

def extract_features_per_channel(data_array):
    # Initialize feature lists
    features_per_channel = {f"Channel_{i+1}": [] for i in range(4)}

    for session_idx in range(len(data_array) // 4): # 4 channels per session
        for channel_idx in range(4):
            session = data_array[session_idx * 4 + channel_idx]

            # Time domain features

            # 1. Peak-to-peak amplitude
            positive_peaks, _ = find_peaks(session)
            negative_peaks, _ = find_peaks(-session)
            if positive_peaks.size and negative_peaks.size:
                max_peak = max(session[positive_peaks])
                min_peak = min(session[negative_peaks])
                peak_to_peak_amplitude = max_peak - min_peak
            else:
                peak_to_peak_amplitude = 0

            # 2. Mean
            mean_val = np.mean(session)

            # 3. Variance
            variance_val = np.var(session)

            # Frequency domain features

            # Compute the Power Spectral Density
            f, Pxx = welch(session, fs=250.0, nperseg=256)

            # 4. Dominant frequency
            dominant_frequency = f[np.argmax(Pxx)]

            # 5. Bandwidth
            half_power = np.max(Pxx) / 2
            indices = np.where(Pxx > half_power)
            first_index = indices[0][0]
            last_index = indices[0][-1]
            bandwidth = f[last_index] - f[first_index]

            # Append features to the channel's feature list
            features_per_channel[f"Channel_{channel_idx+1}"].append([peak_to_p
eak_amplitude, mean_val, variance_val, dominant_frequency, bandwidth])

    # Convert lists to arrays
    for key in features_per_channel:
        features_per_channel[key] = np.array(features_per_channel[key])

```

```

    return features_per_channel

long_blink_features_per_channel =
extract_features_per_channel(long_blink_array_filtered)
short_blink_features_per_channel =
extract_features_per_channel(short_blink_array_filtered)

# Compute average features function
def compute_average_features_per_channel(features_dict):
    average_features = {}
    for key in features_dict:
        avg_peak_to_peak_amplitude = np.mean(features_dict[key][:, 0])
        avg_mean_val = np.mean(features_dict[key][:, 1])
        avg_variance_val = np.mean(features_dict[key][:, 2])
        avg_dominant_frequency = np.mean(features_dict[key][:, 3])
        avg_bandwidth = np.mean(features_dict[key][:, 4])

        average_features[key] = [avg_peak_to_peak_amplitude, avg_mean_val,
avg_variance_val, avg_dominant_frequency, avg_bandwidth]

    return average_features

# Calculate average features for long and short blinks
long_blink_avg_features_per_channel =
compute_average_features_per_channel(long_blink_features_per_channel)
short_blink_avg_features_per_channel =
compute_average_features_per_channel(short_blink_features_per_channel)

# Print results
print("\nAverage Features for Long Blink:")
for key in long_blink_avg_features_per_channel:
    print(f"{key}:")
    print(f"Peak-to-Peak Amplitude:
{long_blink_avg_features_per_channel[key][0]}")
    print(f"Mean: {long_blink_avg_features_per_channel[key][1]}")
    print(f"Variance: {long_blink_avg_features_per_channel[key][2]}")
    print(f"Dominant Frequency: {long_blink_avg_features_per_channel[key][3]}
Hz")
    print(f"Bandwidth: {long_blink_avg_features_per_channel[key][4]} Hz\n")

print("Average Features for Short Blink:")
for key in short_blink_avg_features_per_channel:
    print(f"{key}:")
    print(f"Peak-to-Peak Amplitude:
{short_blink_avg_features_per_channel[key][0]}")
    print(f"Mean: {short_blink_avg_features_per_channel[key][1]}")
    print(f"Variance: {short_blink_avg_features_per_channel[key][2]}")

```

```

    print(f"Dominant Frequency: {short_blink_avg_features_per_channel[key][3]} Hz")
    print(f"Bandwidth: {short_blink_avg_features_per_channel[key][4]} Hz\n")

# Preparing Dataset

# 1. Merge data from both blink types and label them
all_features = []
for channel in ["Channel_1", "Channel_2", "Channel_3", "Channel_4"]:
    all_features_channel =
np.vstack([short_blink_features_per_channel[channel],
long_blink_features_per_channel[channel]])
    all_features.append(all_features_channel)

all_features = np.hstack(all_features) # Combining features from all channels

labels_short =
np.zeros(short_blink_features_per_channel["Channel_1"].shape[0])
labels_long = np.ones(long_blink_features_per_channel["Channel_1"].shape[0])
all_labels = np.hstack([labels_short, labels_long])

# 2. Split the dataset
X_train, X_test, y_train, y_test = train_test_split(all_features, all_labels,
test_size=0.3)

# Convert labels to one-hot encoding for training
# y_train_categorical = to_categorical(y_train)

# Build and train the Decision Tree model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Use the trained model to predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print results
print("\nModel Performance on Test Data:")
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")

```

```
print(conf_matrix)
print("\nClassification Report:")
print(classification_rep)

# Save the model
# DecisionTree models are typically saved using joblib or pickle
import joblib
model_save_path = 'C:/dev/digitalbiomarkers/decision_tree_model.joblib'
joblib.dump(model, model_save_path)
print("Model saved at", model_save_path)
```