Importing Packages & Set Up Data Layout

Preprocessing Information for the Given Data.

A high-pass filter with a 30 Hz cut-off frequency and a power line notch filter (50 Hz) were used. All recordings are artifact-free EEG segments of 60 seconds duration. At the stage of data preprocessing, the Independent Component Analysis (ICA) was used to eliminate the artifacts (eyes, muscle, and cardiac overlapping of the cardiac pulsation). The arithmetic task was the serial subtraction of two numbers. Each trial started with the communication orally 4-digit (minuend) and 2-digit (subtrahend) numbers (e.g. 3141 and 42).

In [1]:

```
# Let's load some packages we need (pip install mne)
import mne
import mne.viz
from mne.datasets import eegbci
from mne.io import concatenate_raws, read_raw_edf
from mne.channels import make_standard_montage
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
# ! pip install mne

# Read raw data files where each file contains a run
files = ['../../datasets/HW2Datasets/Subject06_1.edf', '../../datasets/HW2Datasets/Subject06_2.edf', '../../data

# Read the raw EDF files into an array
raws = [read_raw_edf(f, preload=True) for f in files]

# Loop through the array and make the following changes to the raw files
for raw in raws:

    # Rename the raw channels
    raw.rename_channels({'EEG F3':'F3', 'EEG F4':'F4',
                         'EEG Fp1':'Fp1', 'EEG Fp2':'Fp2', 'EEG F7':'F7', 'EEG F8':'F8',
                         'EEG T3':'T3', 'EEG T4':'T4', 'EEG C3':'C3', 'EEG C4':'C4',
                         'EEG T5':'T5', 'EEG T6':'T6', 'EEG P3':'P3', 'EEG P4':'P4',
                         'EEG O1':'O1', 'EEG O2':'O2', 'EEG Fz':'Fz', 'EEG Cz':'Cz',
                         'EEG Pz':'Pz', 'EEG A2-A1':'A2', 'ECG ECG':'ECG'})


    # Set channel types
    raw.set_channel_types({'ECG':'ecg'})

    # Define the channel locations
    raw.set_montage(mne.channels.make_standard_montage('standard_1020'))

    # Print Raw Channel Names for double checking
    print(raw.ch_names)

# Rename the raws with more insightfull names
subject6_background = raws[0] # Subject 6 background raw
subject6_task = raws[1] # Subject 6 task raw
subject7_background = raws[2] # Subject 7 background raw
subject7_task = raws[3] # Subject 7 task raw

# Function to segment data into non-overlapping windows of length 300 samples
def segment_data(raw, window_size=300):
    data = raw.get_data()  # Get the raw data
    n_channels, n_samples = data.shape # get dimensions
    print("Data Shape Before:", n_channels, n_samples) # display dimensions for understanding
    n_windows = n_samples // window_size  # Number of windows

    # Reshape data into (n_channels, n_windows, window_size)
    segmented_data = data[:, :n_windows * window_size].reshape(n_channels, n_windows, window_size)
    print("Data Shape After:", n_channels, n_windows, window_size) # display shape after reshaping

    return segmented_data # return the segmented data


# Segment each raw file into windows
subject6_background_segments = segment_data(subject6_background)
subject6_task_segments = segment_data(subject6_task)
subject7_background_segments = segment_data(subject7_background)
subject7_task_segments = segment_data(subject7_task)

# Create labels: 0 for background, 1 for task
subject6_background_labels = np.zeros(subject6_background_segments.shape[1])
subject6_task_labels = np.ones(subject6_task_segments.shape[1])
subject7_background_labels = np.zeros(subject7_background_segments.shape[1])
subject7_task_labels = np.ones(subject7_task_segments.shape[1])
```

```python
# Concatenate data for both subjects
X = np.concatenate([subject6_background_segments, subject6_task_segments,
                    subject7_background_segments, subject7_task_segments], axis=1)

# Concatenate labels for both subjects
y = np.concatenate([subject6_background_labels, subject6_task_labels,
                    subject7_background_labels, subject7_task_labels])

# X shape will be (n_channels, total_windows * window_size), and y will be the labels for each window
print("Shape of segmented data:", X.shape) # See the dimensions of X
print("Shape of labels:", y.shape) # See the dimensions of y
```

```
Extracting EDF parameters from /home/joshua/Desktop/MainFolder/OuClasses/2024 Fall/Neural-Data-Science/datasets/
HW2Datasets/Subject06_1.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 90999  =      0.000 ...   181.998 secs...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 90999  =      0.000 ...   181.998 secs...
Extracting EDF parameters from /home/joshua/Desktop/MainFolder/OuClasses/2024 Fall/Neural-Data-Science/datasets/
HW2Datasets/Subject06_2.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 30999  =      0.000 ...    61.998 secs...
Extracting EDF parameters from /home/joshua/Desktop/MainFolder/OuClasses/2024 Fall/Neural-Data-Science/datasets/
HW2Datasets/Subject07_1.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 90999  =      0.000 ...   181.998 secs...
Extracting EDF parameters from /home/joshua/Desktop/MainFolder/OuClasses/2024 Fall/Neural-Data-Science/datasets/
HW2Datasets/Subject07_2.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 30999  =      0.000 ...    61.998 secs...
['Fp1', 'Fp2', 'F3', 'F4', 'F7', 'F8', 'T3', 'T4', 'C3', 'C4', 'T5', 'T6', 'P3', 'P4', 'O1', 'O2', 'Fz', 'Cz', '
Pz', 'A2', 'ECG']
['Fp1', 'Fp2', 'F3', 'F4', 'F7', 'F8', 'T3', 'T4', 'C3', 'C4', 'T5', 'T6', 'P3', 'P4', 'O1', 'O2', 'Fz', 'Cz', '
Pz', 'A2', 'ECG']
['Fp1', 'Fp2', 'F3', 'F4', 'F7', 'F8', 'T3', 'T4', 'C3', 'C4', 'T5', 'T6', 'P3', 'P4', 'O1', 'O2', 'Fz', 'Cz', '
Pz', 'A2', 'ECG']
['Fp1', 'Fp2', 'F3', 'F4', 'F7', 'F8', 'T3', 'T4', 'C3', 'C4', 'T5', 'T6', 'P3', 'P4', 'O1', 'O2', 'Fz', 'Cz', '
Pz', 'A2', 'ECG']
Data Shape Before: 21 91000
Data Shape After: 21 303 300
Data Shape Before: 21 31000
Data Shape After: 21 103 300
Data Shape Before: 21 91000
Data Shape After: 21 303 300
Data Shape Before: 21 31000
Data Shape After: 21 103 300
Shape of segmented data: (21, 812, 300)
Shape of labels: (812,)
```

Q4)

Repeat the analysis in (Q3) using a different machine learning algorithm of your choice (other than logistic regression), and discuss how your results have changed.

So there are alot of ML algorithms to choose from random forest to boosted decision trees to k nearest neighbors. However I will be using Neural Networks mostly because out of all the models I have tested and messed around with (including hyperparameters), this one preformed the best and most importantly, did better than the logistic regression model on all metrics.

In [2]:

```python
# Import ML Libraries
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score, f1_score

# Import library for k-folds
from sklearn.model_selection import StratifiedKFold

# Import library for fast fourier transform
```

```
from scipy.fft import fft

# Import library for equal data distribution
from imblearn.over_sampling import SMOTE

# Import library for normalization
from sklearn.preprocessing import StandardScaler

# Make FFT function
def apply_fft(X, n_fft=300):
    # X is of shape (n_samples, n_channels, n_points_per_window)
    X_fft = np.abs(fft(X, n=n_fft, axis=2))  # FFT along the last axis (window axis)
    return X_fft[:, :, :n_fft//2]  # Take only the positive frequencies (half of the spectrum)

# Apply FFT to the data (shape will still be (n_samples, n_channels, n_features_per_channel))
X_fft = apply_fft(X)

# Reshape the data for model training (n_samples, n_features)
X_reshaped = X_fft.reshape(X_fft.shape[1], -1)  # (n_windows, n_channels * window_size)

# Apply SMOTE after transformation
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_reshaped, y)

# Initialize the Neural Network model
model = MLPClassifier(hidden_layer_sizes=(5, 10, 5), max_iter=1000, random_state=42, solver='sgd', learning_rate
# learning_rate_init=0.0001

# Create k-folds where k=5
skf = StratifiedKFold(n_splits=5)
folds = skf.split(X_resampled, y_resampled) # make different folds for X and y

train_idxs=[] # store training indexes
test_idxs=[] # store test indexes
total_y_test = [] # store all y test values
total_y_pred = [] # store all y pred values

# Loop through all folds
for i, fold in enumerate(folds):
    train_idx, test_idx = fold # Grab indexes from fold
    train_idxs.append(train_idx) # append training indexes to the training list
    test_idxs.append(test_idx) # append testing indexes to the testing list

accuracy_arr = []
balanced_accuracy_arr = []
f1_score_arr = []

# Loop through the 5 folds made previously
for i in range(5):
    X_train = X_resampled[train_idxs[i][:]] # Load in the training X values from index i
    y_train = y_resampled[train_idxs[i][:]] # Load in the training y values from index i
    X_test = X_resampled[test_idxs[i][:]] # Load in the testing X values from index i
    y_test = y_resampled[test_idxs[i][:]] # Load in the testing y values from index i

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.fit_transform(X_test)

    # Fit the model
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Extend total y test array
    total_y_test.extend(y_test)

    # Extend total y pred array
    total_y_pred.extend(y_pred)

    # Print out the current fold we are iterating over
    print("Examining fold %i" % (i + 1))

    # Evaluate the model using accuracy
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_arr.append(accuracy) # Append accuracy to array

    # Evaluate the model using balanced accuracy
    balanced_accuracy = balanced_accuracy_score(y_test, y_pred)
    balanced_accuracy_arr.append(balanced_accuracy) # Append balanced accuracy to array

    # Evaluate the model using f1 score
    f1 = f1_score(y_test, y_pred)
```

```python
    f1_score_arr.append(f1) # Append f1 score to array

    # Print accuracy
    print(f"Accuracy: {accuracy*100:.2f}%")

    # Print balanced accuracy
    print(f"Balanced Accuracy: {balanced_accuracy*100:.2f}%")

    # Print f1 score
    print(f"F1 Score: {f1*100:.2f}%\n")

print(f"Average Accuracy Score: {np.sum(accuracy_arr)*100/5:.2f}%")
print(f"Average Balanced Accuracy Score: {np.sum(balanced_accuracy_arr)*100/5:.2f}%")
print(f"Average F1 Score: {np.sum(f1_score_arr)*100/5:.2f}%")
```

```
Examining fold 1
Accuracy: 69.96%
Balanced Accuracy: 69.93%
F1 Score: 67.26%

Examining fold 2
Accuracy: 82.72%
Balanced Accuracy: 82.68%
F1 Score: 84.09%

Examining fold 3
Accuracy: 90.91%
Balanced Accuracy: 90.91%
F1 Score: 91.54%

Examining fold 4
Accuracy: 90.08%
Balanced Accuracy: 90.08%
F1 Score: 90.84%

Examining fold 5
Accuracy: 84.71%
Balanced Accuracy: 84.71%
F1 Score: 85.71%

Average Accuracy Score: 83.68%
Average Balanced Accuracy Score: 83.66%
Average F1 Score: 83.89%
```

As you can see above the F1-Score, Balanced Accuracy, and Accuracy metrics all increase by around 3% compared to the logistic regression model. **HOWEVER** there is a huge caveate to this. Firstly the Neural Network Model here is way more complex. This is because the hidden layers add hidden interpretation of the data, which is way harder to explain and attempting to would be nothing more than an educated guess about what patterns the model is noticing.

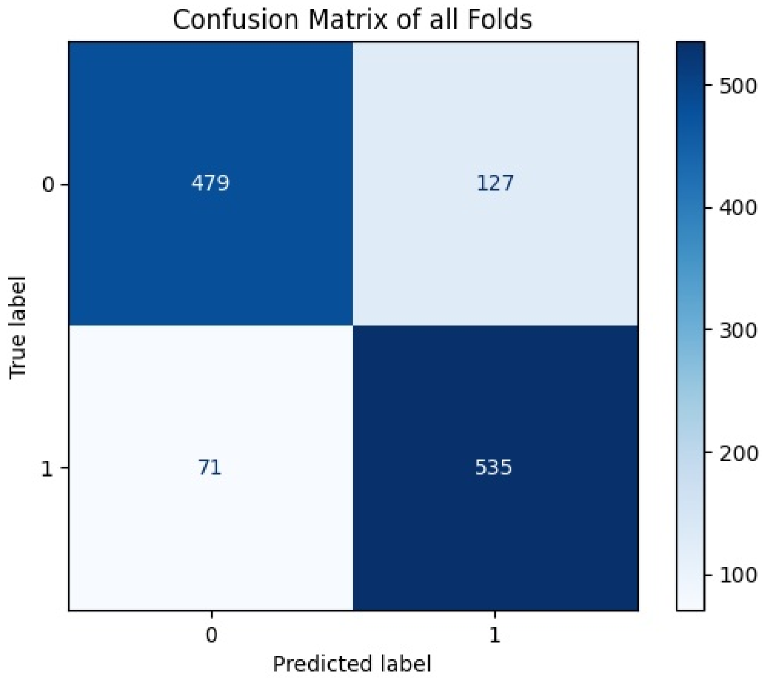
Secondly if you look at the models hyperparameters, you can see that the model is assigned to random_state=42, this is because I want the output to beconsistant, but also because other states preform **really bad** I'm talking like 55% accuracy type bad and it just so happend that this model preformed the best and even better than the logistic regression model. However, I'm aware that I could have overfitted the model to the k-folds testing (which is better than no k-folds) and so I increase the number of folds and included random shuffling. Even with this change the Neural Network Model Prefomed good and better than the logistic regression model by around 2% on average.

In [3]:
```python
# Import library for confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate the confusion matrix
cm = confusion_matrix(total_y_test, total_y_pred)

# Construct the matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap=plt.cm.Blues)  # plot the matrix
plt.title("Confusion Matrix of all Folds") # title matrix
plt.show() # show matrix

# Get the confusion matrix values
tn, fp, fn, tp = cm.ravel()
print(f"True Negatives (TN): {tn}") # Show the Number of True Negatives
print(f"False Positives (FP): {fp}") # Show the Number of False Positives
print(f"False Negatives (FN): {fn}") # Show the Number of False Negatives
print(f"True Positives (TP): {tp}") # Show the Number of True Positives
```




```
True Negatives (TN): 479
False Positives (FP): 127
False Negatives (FN): 71
True Positives (TP): 535
```

Above you can see the improvement in the Number of True Positives and True Negatives in comparision to the logistic regression model.

So would I use this model over the logistic regression model? It would depend. Mostly because the neural net model could also be overfit to subject 6 and 7's brain waves. Maybe not, but to be honest I wouldn't bet my money that this model is always better in having higher metrics overall due to a lack of testing (more subjects being in the data would make a neural network like this one more reliable for results). Overall I would say the logistic regression model would be more reliable (in terms of generalizing to other people and understanding) and doesn't give much worse results.