In [42]:
```
# Import necessary libraries
from glob import glob
import os
import mne
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, precision_score, recall_scor
from scipy import stats
import numpy as np
from tqdm import tqdm_notebook
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GroupKFold,GridSearchCV,cross_val_scor
from tensorflow.keras.layers import Conv1D,BatchNormalization,LeakyReLU,Ma
from tensorflow.keras.models import Sequential
from tensorflow.keras.backend import clear_session
from sklearn.model_selection import GroupKFold,LeaveOneGroupOut
from sklearn.preprocessing import StandardScaler
```

In [43]:
```
# Read all files ending with .edf and .edf.seizures
all_files_path = glob(r"D:\Documents\Prof_Docs\FDA\Projects\Project 3\chb-
                 glob(r"D:\Documents\Prof_Docs\FDA\Projects\Project 3\chb-


print("Total files:", len(all_files_path))
```

```
Total files: 23
```

In [44]:
```
all_files_path[0]
```

Out[44]:
```
'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_03.edf'
```

In [45]:

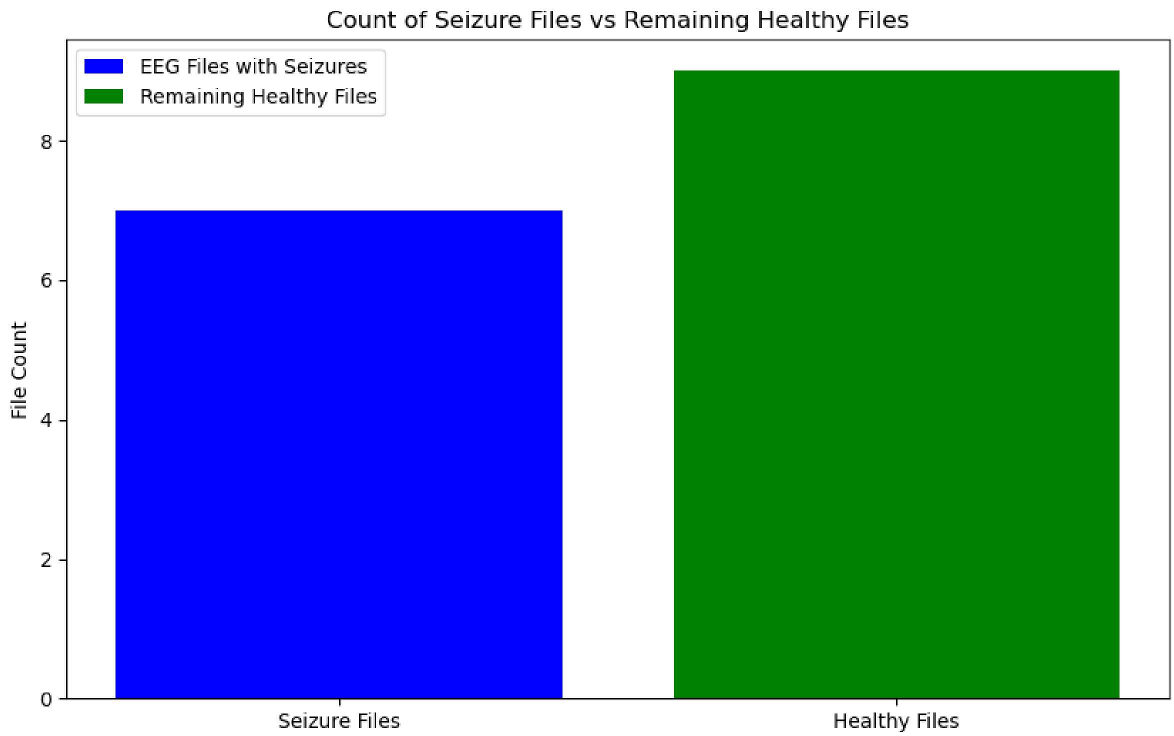
```
# Segregate files with seizures and their corresponding EEG data files
files_with_seizures = []
eeg_files_with_seizures = []

# Create a copy of the all_files_path list
all_file_path = all_files_path.copy()

for seizure_file_path in all_files_path:
    if '.edf.seizures' in seizure_file_path:
        # This is a seizure data file, so add it to the list
        files_with_seizures.append(seizure_file_path)

        # Extract the corresponding EEG data file name
        eeg_file_name = os.path.basename(seizure_file_path).replace('.edf.
        eeg_file_path = next((path for path in all_files_path if eeg_file_

        if eeg_file_path:
            # Remove the corresponding EEG data file from the copy
            all_file_path.remove(eeg_file_path)
            eeg_files_with_seizures.append(eeg_file_path)

# Remove files with '.edf.seizures' extension from the remaining files
healthy_file_path = [path for path in all_file_path if '.edf.seizures' not

# Print or use the segregated file paths
print("EEG files with seizures corresponding to seizure files:", len(eeg_f
print(eeg_files_with_seizures)
```

```
EEG files with seizures corresponding to seizure files: 7
['D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-datab
ase-1.0.0\\chb01\\chb01_03.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Pr
oject 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_04.edf', 'D:\\Documen
ts\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\ch
b01\\chb01_15.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-
mit-scalp-eeg-database-1.0.0\\chb01\\chb01_16.edf', 'D:\\Documents\\Prof_Docs
\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_1
8.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-ee
g-database-1.0.0\\chb01\\chb01_21.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Proje
cts\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_26.edf']
```

In [46]:
```
print("Remaining files without seizures:", len(healthy_file_path))
print(healthy_file_path)
```

```
Remaining files without seizures: 9
['D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_17.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_19.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_20.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_22.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_23.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_24.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_25.edf',
'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_27.edf', 'D:\\Documents\\Prof_Docs\\FDA\\Projects\\Project 3\\chb-mit-scalp-eeg-database-1.0.0\\chb01\\chb01_29.edf']
```

In [47]:
```
# Count of the files
seizure_files_count = len(files_with_seizures)
healthy_files_count = len(healthy_file_path)

# Visualization
plt.figure(figsize=(10, 6))

# Plotting the count of corresponding EEG files using a bar plot
plt.bar([0], [seizure_files_count], color='blue', label='EEG Files with Se

# Plotting the count of remaining healthy files using a bar plot
plt.bar([1], [healthy_files_count], color='green', label='Remaining Health

plt.xticks([0, 1], ['Seizure Files', 'Healthy Files'])
plt.ylabel('File Count')
plt.title('Count of Seizure Files vs Remaining Healthy Files')
plt.legend()
plt.show()
```

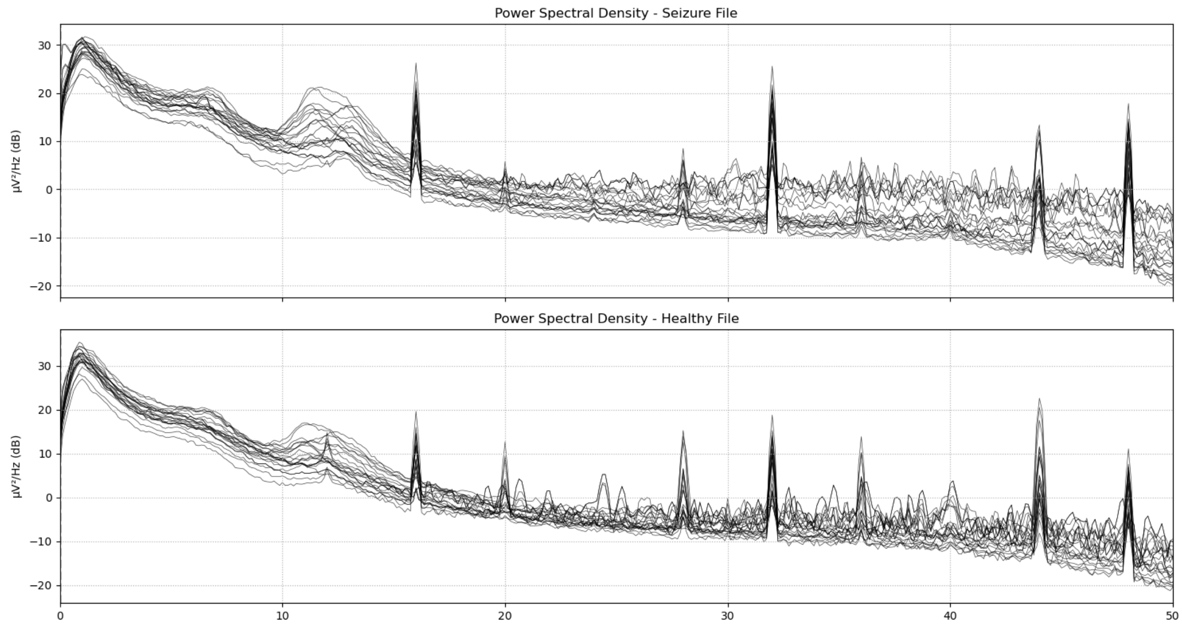


In [48]:

```python
# Taking the first file from each path
seizure_file = eeg_files_with_seizures[0]
healthy_file = healthy_file_path[0]

# Reading EEG data from seizure file
seizure_raw = mne.io.read_raw_edf(seizure_file, preload=True)

# Reading EEG data from healthy file
healthy_raw = mne.io.read_raw_edf(healthy_file, preload=True)

# Plotting EEG data for one seizure file
fig, ax = plt.subplots(2, 1, figsize=(15, 8), sharex=True)
seizure_raw.plot_psd(ax=ax[0], fmin=0, fmax=50, show=False)
ax[0].set_title('Power Spectral Density - Seizure File')

# Plotting EEG data for one healthy file
healthy_raw.plot_psd(ax=ax[1], fmin=0, fmax=50, show=False)
ax[1].set_title('Power Spectral Density - Healthy File')

plt.tight_layout()
plt.show()
```

```
Extracting EDF parameters from D:\Documents\Prof_Docs\FDA\Projects\Project 3
\chb-mit-scalp-eeg-database-1.0.0\chb01\chb01_03.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 921599  =      0.000 ...  3599.996 secs...

C:\Users\shrey\AppData\Local\Temp\ipykernel_24812\117461896.py:6: RuntimeWarn
ing: Channel names are not unique, found duplicates for: {'T8-P8'}. Applying
running numbers for duplicates.
  seizure_raw = mne.io.read_raw_edf(seizure_file, preload=True)

Extracting EDF parameters from D:\Documents\Prof_Docs\FDA\Projects\Project 3
\chb-mit-scalp-eeg-database-1.0.0\chb01\chb01_17.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 921599  =      0.000 ...  3599.996 secs...

C:\Users\shrey\AppData\Local\Temp\ipykernel_24812\117461896.py:9: RuntimeWarn
ing: Channel names are not unique, found duplicates for: {'T8-P8'}. Applying
running numbers for duplicates.
  healthy_raw = mne.io.read_raw_edf(healthy_file, preload=True)

NOTE: plot_psd() is a legacy function. New code should use .compute_psd().plo
t().
Effective window size : 8.000 (s)
NOTE: plot_psd() is a legacy function. New code should use .compute_psd().plo
t().
Effective window size : 8.000 (s)
```

```
C:\Users\shrey\AppData\Local\Temp\ipykernel_24812\117461896.py:13: RuntimeWar
ning: Channel locations not available. Disabling spatial colors.
  seizure_raw.plot_psd(ax=ax[0], fmin=0, fmax=50, show=False)
C:\Users\shrey\AppData\Local\Temp\ipykernel_24812\117461896.py:17: RuntimeWar
ning: Channel locations not available. Disabling spatial colors.
  healthy_raw.plot_psd(ax=ax[1], fmin=0, fmax=50, show=False)
```



In [49]:
```
# Define a function to read and preprocess EEG data from a given file
def read_data(file_path):
    # Read raw EEG data from the specified file and preload it into memory
    datax = mne.io.read_raw_edf(file_path, preload=True)

    # Print the shape of the original EEG data
    print("Original data shape:", datax.get_data().shape)

    # Set EEG reference to average reference and apply bandpass filtering
    datax.set_eeg_reference()
    datax.filter(l_freq=1, h_freq=45)

    # Print the shape of the processed EEG data
    print("Processed data shape:", datax.get_data().shape)

    # Create fixed-length epochs from the preprocessed data with a specifi
    epochs = mne.make_fixed_length_epochs(datax, duration=25, overlap=0)

    # Print the shape of the epochs data before loading
    print("Epochs data shape before loading:", epochs.get_data().shape)

    # Load the data into memory for all epochs, drop bad epochs, and print
    epochs.load_data()
    epochs.drop_bad()
    print("Number of epochs created:", len(epochs))

    # Get the data from the epochs and print the final shape
    epochs_data = epochs.get_data()
    print("Final epochs data shape:", epochs_data.shape)

    # Return the preprocessed and segmented EEG data
    return epochs_data
```

In [50]:
```
# Checking if the function works as expected
datax=read_data(healthy_file_path[0])
```

```
Extracting EDF parameters from D:\Documents\Prof_Docs\FDA\Projects\Project 3
\chb-mit-scalp-eeg-database-1.0.0\chb01\chb01_17.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 921599  =      0.000 ...  3599.996 secs...

C:\Users\shrey\AppData\Local\Temp\ipykernel_24812\1607341177.py:4: RuntimeWar
ning: Channel names are not unique, found duplicates for: {'T8-P8'}. Applying
running numbers for duplicates.
  datax = mne.io.read_raw_edf(file_path, preload=True)

Original data shape: (23, 921600)
EEG channel type selected for re-referencing
Applying average reference.
Applying a custom ('EEG',) reference.
Filtering raw data in 1 contiguous segment
Setting up band-pass filter from 1 - 45 Hz

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal bandpass filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 1.00
- Lower transition bandwidth: 1.00 Hz (-6 dB cutoff frequency: 0.50 Hz)
- Upper passband edge: 45.00 Hz
- Upper transition bandwidth: 11.25 Hz (-6 dB cutoff frequency: 50.62 Hz)
- Filter length: 845 samples (3.301 s)


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.
0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.
0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.
0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:    0.
0s

Processed data shape: (23, 921600)
Not setting metadata
144 matching events found
No baseline correction applied
0 projection items activated
Using data from preloaded Raw for 144 events and 6400 original time points
...
0 bad epochs dropped

[Parallel(n_jobs=1)]: Done  23 out of  23 | elapsed:    0.6s finished
```

```
Epochs data shape before loading: (144, 23, 6400)
Using data from preloaded Raw for 144 events and 6400 original time points
...
Number of epochs created: 144
Final epochs data shape: (144, 23, 6400)

C:\Users\shrey\AppData\Local\Temp\ipykernel_24812\1607341177.py:28: FutureWar
ning: The current default of copy=False will change to copy=True in 1.7. Set
the value of copy explicitly to avoid this warning
  epochs_data = epochs.get_data()
```

In [51]:

```
# Getting the shape to have an idea for our input shape
datax.shape
```

Out[51]:

```
(144, 23, 6400)
```

In [52]:

```
# Filtering all the files
control_epochs_array=[read_data(subject) for subject in healthy_file_path]
patients_epochs_array=[read_data(subject) for subject in eeg_files_with_se
```

```
Extracting EDF parameters from D:\Documents\Prof_Docs\FDA\Projects\Project
3\chb-mit-scalp-eeg-database-1.0.0\chb01\chb01_17.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 921599  =      0.000 ...  3599.996 secs...

C:\Users\shrey\AppData\Local\Temp\ipykernel_24812\1607341177.py:4: Runtime
Warning: Channel names are not unique, found duplicates for: {'T8-P8'}. Ap
plying running numbers for duplicates.
  datax = mne.io.read_raw_edf(file_path, preload=True)

Original data shape: (23, 921600)
EEG channel type selected for re-referencing
Applying average reference.
Applying a custom ('EEG',) reference.
Filtering raw data in 1 contiguous segment
Setting up band-pass filter from 1 - 45 Hz

FIR filter parameters
```

In [53]:

```
# getting the length of the files for reference
control_epochs_labels=[len(i)*[0] for i in control_epochs_array]
patients_epochs_labels=[len(i)*[1] for i in patients_epochs_array]
print(len(control_epochs_labels),len(patients_epochs_labels))
```

```
9 7
```

In [54]:

```
# Storing all files
data_list=control_epochs_array+patients_epochs_array
label_list=control_epochs_labels+patients_epochs_labels
print(len(data_list),len(label_list))
```

```
16 16
```

In [55]:

```
# Grouping the files
groups_list=[[i]*len(j) for i, j in enumerate(data_list)]
```

In [56]:

```
# Stacking the data
data_array=np.vstack(data_list)
label_array=np.hstack(label_list)
group_array=np.hstack(groups_list)
print(data_array.shape,label_array.shape,group_array.shape)
```

```
(2095, 23, 6400) (2095,) (2095,)
```

In [57]:
```
# Function to calculate the mean along the last axis of the input data
def mean(data):
    return np.mean(data, axis=-1)

# Function to calculate the standard deviation along the last axis of the
def std(data):
    return np.std(data, axis=-1)

# Function to calculate the peak-to-peak range along the last axis of the
def ptp(data):
    return np.ptp(data, axis=-1)

# Function to calculate the variance along the last axis of the input data
def var(data):
    return np.var(data, axis=-1)

# Function to find the minimum value along the last axis of the input data
def minim(data):
    return np.min(data, axis=-1)

# Function to find the maximum value along the last axis of the input data
def maxim(data):
    return np.max(data, axis=-1)

# Function to find the index of the minimum value along the last axis of t
def argminim(data):
    return np.argmin(data, axis=-1)

# Function to find the index of the maximum value along the last axis of t
def argmaxim(data):
    return np.argmax(data, axis=-1)

# Function to calculate the mean square along the last axis of the input d
def mean_square(data):
    return np.mean(data**2, axis=-1)

# Function to calculate the root mean square along the last axis of the in
def rms(data):
    return np.sqrt(np.mean(data**2, axis=-1))

# Function to calculate the absolute differences between consecutive eleme
def abs_diffs_signal(data):
    return np.sum(np.abs(np.diff(data, axis=-1)), axis=-1)

# Function to calculate the skewness along the last axis of the input data
def skewness(data):
    return stats.skew(data, axis=-1)

# Function to calculate the kurtosis along the last axis of the input data
def kurtosis(data):
    return stats.kurtosis(data, axis=-1)

# Function to concatenate a set of statistical features along the last axi
def concatenate_features(data):
    return np.concatenate((mean(data), std(data), ptp(data), var(data), mi
                           argminim(data), argmaxim(data), mean_square(dat
```

```
                          abs_diffs_signal(data), skewness(data), kurtosi
```

In [58]:

```
# Initialize an empty list to store computed features
features = []

# Iterate through each data array in the 'data_array' list
for data in tqdm_notebook(data_array):
    # Compute and append the concatenated features for the current data ar
    features.append(concatenate_features(data))

# Convert the list of features into a NumPy array
features = np.array(features)

# Print the shape of the resulting features array
print("Shape of the features array:", features.shape)
```

```
C:\Users\shrey\AppData\Local\Temp\ipykernel_24812\4194004714.py:5: TqdmDeprec
ationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for data in tqdm_notebook(data_array):
```

100% 2095/2095 [00:25<00:00, 75.00it/s]

```
Shape of the features array: (2095, 299)
```

In [59]:

```
# Initialize a Logistic Regression classifier with a specified maximum num
clf = LogisticRegression(max_iter=1000)

# Initialize a GroupKFold cross-validator with 5 splits
gkf = GroupKFold(n_splits=5)

# Define the parameter grid for hyperparameter tuning
param_grid = {'classifier__C': [0.01, 0.05, 0.1, 0.5, 1, 2, 3, 4, 5, 8, 10

# Create a pipeline with a standard scaler and the logistic regression cla
pipe = Pipeline([('scaler', StandardScaler()), ('classifier', clf)])

# Initialize GridSearchCV with the pipeline, parameter grid, GroupKFold, c
gscv = GridSearchCV(pipe, param_grid, cv=gkf, n_jobs=16)

# Fit the GridSearchCV to the features, labels, and group information
gscv.fit(features, label_array, groups=group_array)
```

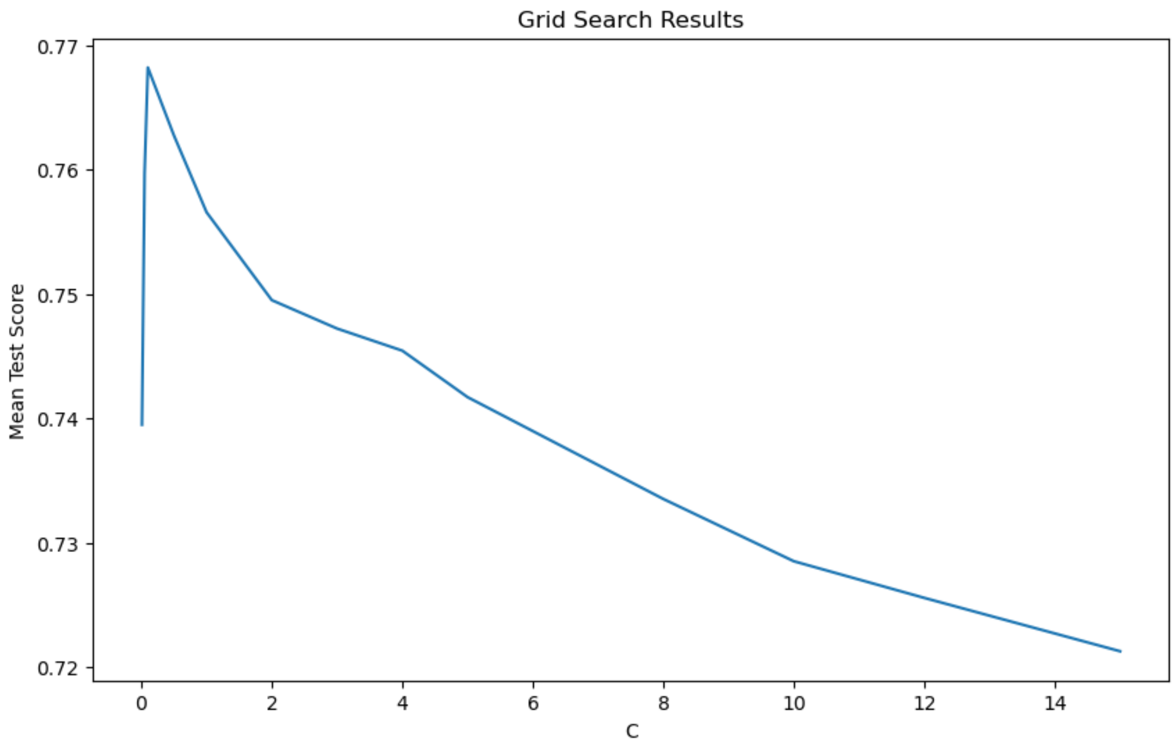
Out[59]:

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ StandardScaler
  ▸ LogisticRegression
```

In [60]:
```python
# Getting the best score
gscv.best_score_
```

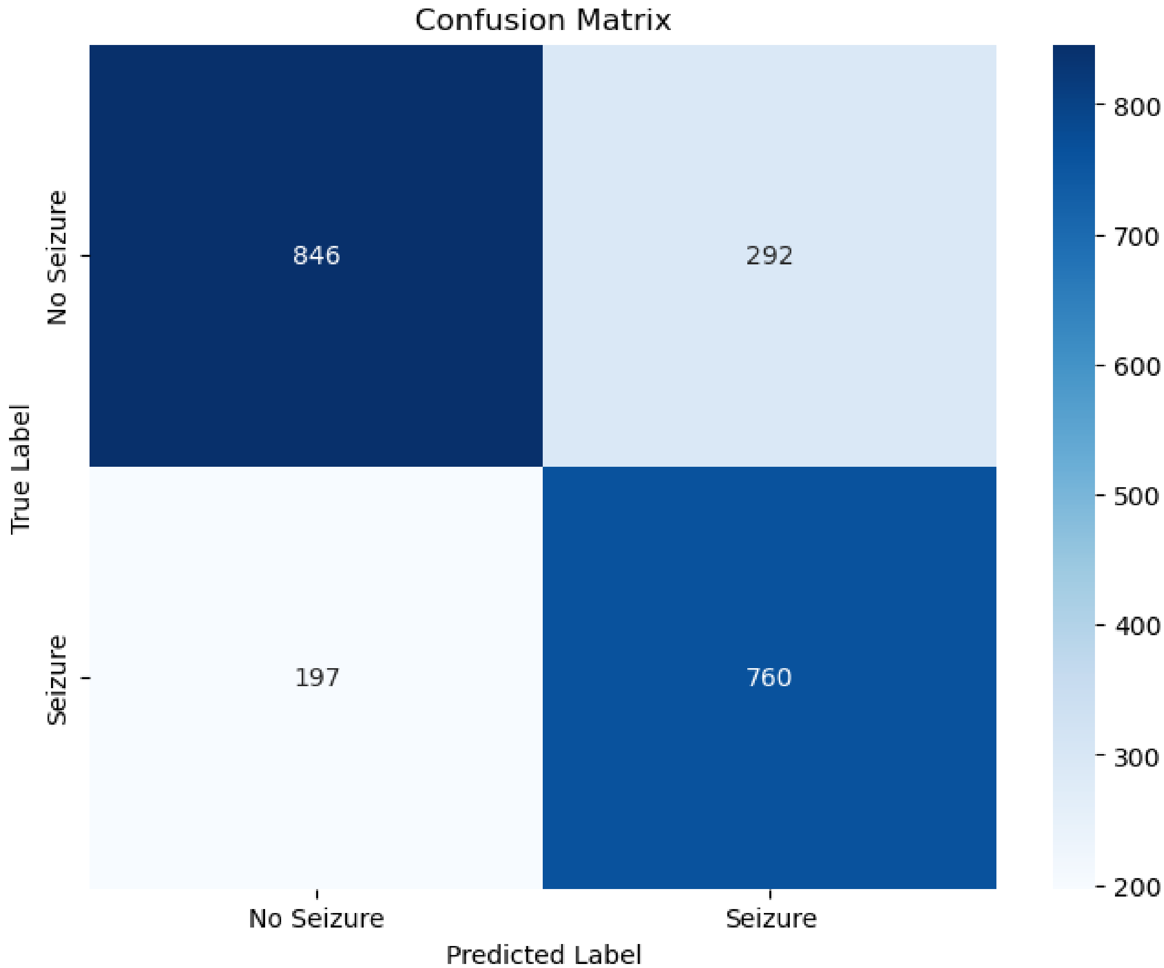
Out[60]: 0.7681923293852229

In [61]:
```python
# Extract grid search results
results_df = pd.DataFrame(gscv.cv_results_)

# Plot mean test scores across different values of C
plt.figure(figsize=(10, 6))
sns.lineplot(x='param_classifier__C', y='mean_test_score', data=results_df
plt.title('Grid Search Results')
plt.xlabel('C')
plt.ylabel('Mean Test Score')
plt.show()
```



In [62]:
```
# Get the best estimator from the grid search cross-validation
best_clf = gscv.best_estimator_

# Use cross_val_predict to obtain predicted labels with cross-validation
y_pred = cross_val_predict(best_clf, features, label_array, groups=group_a

# Compute the confusion matrix
cm = confusion_matrix(label_array, y_pred)

# Calculate precision, recall, and F1 score
precision = precision_score(label_array, y_pred)
recall = recall_score(label_array, y_pred)
f1 = f1_score(label_array, y_pred)

# Print the confusion matrix and evaluation metrics
print("Confusion Matrix:")
print(cm)
print("\nPrecision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# Plot a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Seizur
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
Confusion Matrix:
[[846 292]
 [197 760]]

Precision: 0.7224334600760456
Recall: 0.7941483803552769
F1 Score: 0.7565953210552514
```


Confusion Matrix
No Seizure
Seizure
True Label
846
292
197
760
No Seizure
Seizure
Predicted Label
800
700
600
500
400
300
200

In [63]:
```
# Function to define a CNN model
def cnnmodel():
    # Clear any previous TensorFlow sessions and models
    clear_session()

    # Create a Sequential model
    model = Sequential()

    # Convolutional Layer 1
    model.add(Conv1D(filters=3, kernel_size=3, strides=1, input_shape=(640
    model.add(BatchNormalization())
    model.add(LeakyReLU())
    model.add(MaxPool1D(pool_size=2, strides=2))  # 2

    # Convolutional Layer 2
    model.add(Conv1D(filters=3, kernel_size=3, strides=1))  # 3
    model.add(LeakyReLU())
    model.add(MaxPool1D(pool_size=2, strides=2))  # 4
    model.add(Dropout(0.3))

    # Convolutional Layer 3
    model.add(Conv1D(filters=3, kernel_size=3, strides=1))  # 5
    model.add(LeakyReLU())
    model.add(AveragePooling1D(pool_size=2, strides=2))  # 6
    model.add(Dropout(0.3))

    # Convolutional Layer 4
    model.add(Conv1D(filters=3, kernel_size=3, strides=1))  # 7
    model.add(LeakyReLU())
    model.add(AveragePooling1D(pool_size=2, strides=2))  # 8

    # Convolutional Layer 5
    model.add(Conv1D(filters=3, kernel_size=3, strides=1))  # 9
    model.add(LeakyReLU())

    # Global Average Pooling Layer
    model.add(GlobalAveragePooling1D())  # 10

    # Output Layer
    model.add(Dense(1, activation='sigmoid'))  # 11

    # Compile the model with Adam optimizer and binary crossentropy loss
    model.compile('adam', loss='binary_crossentropy', metrics=['accuracy']

    return model

# Create an instance of the CNN model
model = cnnmodel()

# Display the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d (Conv1D)             (None, 6398, 3)           210       
                                                                 
 batch_normalization (Batch  (None, 6398, 3)           12        
 Normalization)                                                  
                                                                 
 leaky_re_lu (LeakyReLU)     (None, 6398, 3)           0         
                                                                 
 max_pooling1d (MaxPooling1  (None, 3199, 3)           0         
 D)                                                              
                                                                 
 conv1d_1 (Conv1D)           (None, 3197, 3)           30        
                                                                 
 leaky_re_lu_1 (LeakyReLU)   (None, 3197, 3)           0         
                                                                 
 max_pooling1d_1 (MaxPoolin  (None, 1598, 3)           0         
 g1D)                                                            
                                                                 
 dropout (Dropout)           (None, 1598, 3)           0         
                                                                 
 conv1d_2 (Conv1D)           (None, 1596, 3)           30        
                                                                 
 leaky_re_lu_2 (LeakyReLU)   (None, 1596, 3)           0         
                                                                 
 average_pooling1d (Average  (None, 798, 3)            0         
 Pooling1D)                                                      
                                                                 
 dropout_1 (Dropout)         (None, 798, 3)            0         
                                                                 
 conv1d_3 (Conv1D)           (None, 796, 3)            30        
                                                                 
 leaky_re_lu_3 (LeakyReLU)   (None, 796, 3)            0         
                                                                 
 average_pooling1d_1 (Avera  (None, 398, 3)            0         
 gePooling1D)                                                    
                                                                 
 conv1d_4 (Conv1D)           (None, 396, 3)            30        
                                                                 
 leaky_re_lu_4 (LeakyReLU)   (None, 396, 3)            0         
                                                                 
 global_average_pooling1d (  (None, 3)                 0         
 GlobalAveragePooling1D)                                         
                                                                 
 dense (Dense)               (None, 1)                 4         
                                                                 
=================================================================
Total params: 346 (1.35 KB)
Trainable params: 340 (1.33 KB)
Non-trainable params: 6 (24.00 Byte)
_________________________________________________________________
```

In [67]:
```
# Initialize GroupKFold with the default number of splits
gkf = GroupKFold()

# Initialize lists to store training and validation accuracies for each fo
train_accuracies = []
val_accuracies = []

# Initialize variables to track the best training and validation accuracie
best_train_accuracy = 0.0
best_val_accuracy = 0.0

# Iterate through the training and validation splits defined by GroupKFold
for train_index, val_index in gkf.split(data_array, label_array, groups=gr
    # Split data into training and validation sets
    train_features, train_labels = data_array[train_index], label_array[tr
    val_features, val_labels = data_array[val_index], label_array[val_inde

    # Transpose the features if necessary
    train_features = train_features.transpose(0, 2, 1)
    val_features = val_features.transpose(0, 2, 1)

    # Standardize features using the same scaler for both training and val
    scaler = StandardScaler()
    train_features = scaler.fit_transform(train_features.reshape(-1, train
    val_features = scaler.transform(val_features.reshape(-1, val_features.

    # Create and train the CNN model
    model = cnnmodel()
    history = model.fit(train_features, train_labels, epochs=20, batch_siz

    # Append training and validation accuracy values to the lists
    train_accuracies.append(history.history['accuracy'])
    val_accuracies.append(history.history['val_accuracy'])

    # Evaluate accuracies for the current fold
    current_train_accuracy = history.history['accuracy'][-1]
    current_val_accuracy = history.history['val_accuracy'][-1]

    # Update the best training accuracy if the current training accuracy i
    if current_train_accuracy > best_train_accuracy:
        best_train_accuracy = current_train_accuracy

    # Update the best validation accuracy if the current validation accura
    if current_val_accuracy > best_val_accuracy:
        best_val_accuracy = current_val_accuracy

# Print the best training and validation accuracies
print("Best Training Accuracy:", best_train_accuracy)
print("Best Validation Accuracy:", best_val_accuracy)
```

```
14/14 [==============================] - 1s 99ms/step - loss: 0.6078 - acc
uracy: 0.5740 - val_loss: 0.6475 - val_accuracy: 0.4148
Epoch 8/20
14/14 [==============================] - 1s 104ms/step - loss: 0.5777 - ac
curacy: 0.6059 - val_loss: 0.6036 - val_accuracy: 0.7975
Epoch 9/20
14/14 [==============================] - 1s 100ms/step - loss: 0.5490 - ac
curacy: 0.7675 - val_loss: 0.5590 - val_accuracy: 0.8494
Epoch 10/20
14/14 [==============================] - 1s 99ms/step - loss: 0.5161 - acc
uracy: 0.7864 - val_loss: 0.5211 - val_accuracy: 0.8469
Epoch 11/20
14/14 [==============================] - 1s 100ms/step - loss: 0.4818 - ac
curacy: 0.8047 - val_loss: 0.4929 - val_accuracy: 0.8296
Epoch 12/20
14/14 [==============================] - 1s 101ms/step - loss: 0.4612 - ac
curacy: 0.8024 - val_loss: 0.4716 - val_accuracy: 0.8247
Epoch 13/20
14/14 [==============================] - 1s 102ms/step - loss: 0.4412 - ac
curacy: 0.8101 - val_loss: 0.4678 - val_accuracy: 0.8543
```

In [68]:

```
# Print the best training and validation accuracies
print("Best Training Accuracy:", best_train_accuracy)
print("Best Validation Accuracy:", best_val_accuracy)
```

```
Best Training Accuracy: 0.8384615182876587
Best Validation Accuracy: 0.8604061007499695
```

In [69]:

```
# Plotting the training and validation accuracy over epochs
plt.figure(figsize=(10, 6))

for i in range(len(train_accuracies)):
    plt.plot(train_accuracies[i], label=f'Training Fold {i+1}')

for i in range(len(val_accuracies)):
    plt.plot(val_accuracies[i], label=f'Validation Fold {i+1}', linestyle=

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy Over Epochs')
plt.legend()
plt.show()
```