In [2]:
```
import pandas as pd

# Load the datasets
boning_df = pd.read_csv('Boning.csv')
slicing_df = pd.read_csv('Slicing.csv')

# Check the exact column names in the datasets
print("Boning Dataset Columns:", boning_df.columns.tolist())
print("Slicing Dataset Columns:", slicing_df.columns.tolist())

# Define the columns to extract based on your student ID (assuming exact column nam
columns_to_extract = ['Right Forearm x', 'Right Forearm y', 'Right Forearm z',
                      'Left Forearm x', 'Left Forearm y', 'Left Forearm z']

# Extract the relevant columns and add the class label
boning_selected = boning_df[columns_to_extract].copy()
slicing_selected = slicing_df[columns_to_extract].copy()

# Add the frame column if it exists
if 'frame' in boning_df.columns:
    boning_selected['frame'] = boning_df['frame']
    slicing_selected['frame'] = slicing_df['frame']

boning_selected['class'] = 0  # Boning class label
slicing_selected['class'] = 1  # Slicing class label

# Combine the datasets
combined_df = pd.concat([boning_selected, slicing_selected], ignore_index=True)

# Save the combined dataset
combined_df.to_csv('combined_data.csv', index=False)

combined_df.head()
```

```
Boning Dataset Columns: ['Frame', 'L5 x', 'L5 y', 'L5 z', 'L3 x', 'L3 y', 'L3 z', 'T
12 x', 'T12 y', 'T12 z', 'T8 x', 'T8 y', 'T8 z', 'Neck x', 'Neck y', 'Neck z', 'Head
x', 'Head y', 'Head z', 'Right Shoulder x', 'Right Shoulder y', 'Right Shoulder z',
'Right Upper Arm x', 'Right Upper Arm y', 'Right Upper Arm z', 'Right Forearm x', 'R
ight Forearm y', 'Right Forearm z', 'Right Hand x', 'Right Hand y', 'Right Hand z',
'Left Shoulder x', 'Left Shoulder y', 'Left Shoulder z', 'Left Upper Arm x', 'Left U
pper Arm y', 'Left Upper Arm z', 'Left Forearm x', 'Left Forearm y', 'Left Forearm
z', 'Left Hand x', 'Left Hand y', 'Left Hand z', 'Right Upper Leg x', 'Right Upper L
eg y', 'Right Upper Leg z', 'Right Lower Leg x', 'Right Lower Leg y', 'Right Lower L
eg z', 'Right Foot x', 'Right Foot y', 'Right Foot z', 'Right Toe x', 'Right Toe y',
'Right Toe z', 'Left Upper Leg x', 'Left Upper Leg y', 'Left Upper Leg z', 'Left Low
er Leg x', 'Left Lower Leg y', 'Left Lower Leg z', 'Left Foot x', 'Left Foot y', 'Le
ft Foot z', 'Left Toe x', 'Left Toe y', 'Left Toe z']
Slicing Dataset Columns: ['Frame', 'L5 x', 'L5 y', 'L5 z', 'L3 x', 'L3 y', 'L3 z',
'T12 x', 'T12 y', 'T12 z', 'T8 x', 'T8 y', 'T8 z', 'Neck x', 'Neck y', 'Neck z', 'He
ad x', 'Head y', 'Head z', 'Right Shoulder x', 'Right Shoulder y', 'Right Shoulder
z', 'Right Upper Arm x', 'Right Upper Arm y', 'Right Upper Arm z', 'Right Forearm
x', 'Right Forearm y', 'Right Forearm z', 'Right Hand x', 'Right Hand y', 'Right Han
d z', 'Left Shoulder x', 'Left Shoulder y', 'Left Shoulder z', 'Left Upper Arm x',
'Left Upper Arm y', 'Left Upper Arm z', 'Left Forearm x', 'Left Forearm y', 'Left Fo
rearm z', 'Left Hand x', 'Left Hand y', 'Left Hand z', 'Right Upper Leg x', 'Right U
pper Leg y', 'Right Upper Leg z', 'Right Lower Leg x', 'Right Lower Leg y', 'Right L
ower Leg z', 'Right Foot x', 'Right Foot y', 'Right Foot z', 'Right Toe x', 'Right T
oe y', 'Right Toe z', 'Left Upper Leg x', 'Left Upper Leg y', 'Left Upper Leg z', 'L
eft Lower Leg x', 'Left Lower Leg y', 'Left Lower Leg z', 'Left Foot x', 'Left Foot
y', 'Left Foot z', 'Left Toe x', 'Left Toe y', 'Left Toe z']
```

Out[2]:

	Right Forearm x	Right Forearm y	Right Forearm z	Left Forearm x	Left Forearm y	Left Forearm z	class
0	-0.092450	0.527938	1.304532	0.464938	-0.063370	0.390884	0
1	0.644069	0.341520	-0.018086	-0.186032	-0.325616	0.413670	0
2	0.856214	0.374047	-0.075706	-0.444088	-0.447592	0.446798	0
3	1.025232	0.383944	-0.426156	-0.674285	-0.373350	0.634095	0
4	1.559554	-0.081100	0.120248	-1.087761	-0.365208	0.799243	0

In [3]:

```python
import numpy as np

# Define functions for RMS, Roll, and Pitch calculations
def calculate_rms(x, y):
    return np.sqrt(x**2 + y**2)

def calculate_rms_xyz(x, y, z):
    return np.sqrt(x**2 + y**2 + z**2)

def calculate_roll(accel_y, accel_x, accel_z):
    return np.degrees(np.arctan2(accel_y, np.sqrt(accel_x**2 + accel_z**2)))

def calculate_pitch(accel_x, accel_y, accel_z):
    return np.degrees(np.arctan2(accel_x, np.sqrt(accel_y**2 + accel_z**2)))

# Compute RMS, Roll, and Pitch for Right Forearm
combined_df['RMS_xy_right'] = calculate_rms(combined_df['Right Forearm x'], combine
```

```
combined_df['RMS_yz_right'] = calculate_rms(combined_df['Right Forearm y'], combine
combined_df['RMS_zx_right'] = calculate_rms(combined_df['Right Forearm z'], combine
combined_df['RMS_xyz_right'] = calculate_rms_xyz(combined_df['Right Forearm x'], co
combined_df['Roll_right'] = calculate_roll(combined_df['Right Forearm y'], combined
combined_df['Pitch_right'] = calculate_pitch(combined_df['Right Forearm x'], combin

# Compute RMS, Roll, and Pitch for Left Forearm
combined_df['RMS_xy_left'] = calculate_rms(combined_df['Left Forearm x'], combined_
combined_df['RMS_yz_left'] = calculate_rms(combined_df['Left Forearm y'], combined_
combined_df['RMS_zx_left'] = calculate_rms(combined_df['Left Forearm z'], combined_
combined_df['RMS_xyz_left'] = calculate_rms_xyz(combined_df['Left Forearm x'], comb
combined_df['Roll_left'] = calculate_roll(combined_df['Left Forearm y'], combined_d
combined_df['Pitch_left'] = calculate_pitch(combined_df['Left Forearm x'], combined

# Save the dataset with composite columns
combined_df.to_csv('combined_data_with_composites.csv', index=False)

combined_df.head()
```

Out[3]:

	Right Forearm x	Right Forearm y	Right Forearm z	Left Forearm x	Left Forearm y	Left Forearm z	class	RMS_xy_right	RM
0	-0.092450	0.527938	1.304532	0.464938	-0.063370	0.390884	0	0.535971	
1	0.644069	0.341520	-0.018086	-0.186032	-0.325616	0.413670	0	0.729014	
2	0.856214	0.374047	-0.075706	-0.444088	-0.447592	0.446798	0	0.934352	
3	1.025232	0.383944	-0.426156	-0.674285	-0.373350	0.634095	0	1.094767	
4	1.559554	-0.081100	0.120248	-1.087761	-0.365208	0.799243	0	1.561662	

In [4]:

```
from scipy.integrate import simps
from scipy.signal import find_peaks

# Define function to calculate AUC and count peaks
def calculate_auc(values):
    return simps(values)

def count_peaks(values):
    peaks, _ = find_peaks(values)
    return len(peaks)

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

# Iterate over the dataset in chunks of 60 rows (1 minute)
for i in range(0, len(combined_df), 60):
    chunk = combined_df.iloc[i:i+60]
    features = {}

    # Compute all statistical features for each relevant column
    for col in combined_df.columns[1:-1]:  # Exclude 'frame' and 'class'
        col_values = chunk[col].values
```

```
        features[f'{col}_mean'] = np.mean(col_values)
        features[f'{col}_std'] = np.std(col_values)
        features[f'{col}_min'] = np.min(col_values)
        features[f'{col}_max'] = np.max(col_values)
        features[f'{col}_auc'] = calculate_auc(col_values)
        features[f'{col}_peaks'] = count_peaks(col_values)

    # Extract the class value
    features['class'] = chunk['class'].iloc[0]

    features_list.append(features)

# Convert the list of features into a DataFrame
features_df = pd.DataFrame(features_list)

# Save the features dataset
features_df.to_csv('features_data.csv', index=False)

features_df.head()
```

Out[4]:

	Right Forearm y_mean	Right Forearm y_std	Right Forearm y_min	Right Forearm y_max	Right Forearm y_auc	Right Forearm y_peaks	Right Forearm z_mean	Right Forearm z_std	
0	0.036850	0.468068	-0.983582	1.083746	2.294041	14	0.062675	0.447047	·
1	0.025613	0.810618	-1.314524	2.608674	0.533953	15	0.226247	0.779904	·
2	-0.443062	3.721485	-18.794714	5.655015	-13.534683	12	0.106342	2.817171	·
3	-0.042345	2.420303	-5.874553	10.041036	-10.889918	18	0.048105	4.142846	-1
4	0.714606	8.156508	-22.986479	20.122030	52.647825	13	1.305862	12.777106	-3

5 rows × 103 columns

In [5]:

```
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Separate features and target
X = features_df.drop(columns=['class'])
y = features_df['class']

# Split data into training and testing sets (70/30 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Initialize models
svm_model = SVC(kernel='rbf', random_state=42)
sgd_model = SGDClassifier(random_state=42)
rf_model = RandomForestClassifier(random_state=42)
```

```
mlp_model = MLPClassifier(random_state=42)

# Train and evaluate SVM model
svm_model.fit(X_train, y_train)
svm_accuracy = accuracy_score(y_test, svm_model.predict(X_test))
svm_cv_scores = cross_val_score(svm_model, X, y, cv=10)
svm_cv_mean = svm_cv_scores.mean()

# Train and evaluate SGD model
sgd_model.fit(X_train, y_train)
sgd_accuracy = accuracy_score(y_test, sgd_model.predict(X_test))
sgd_cv_scores = cross_val_score(sgd_model, X, y, cv=10)
sgd_cv_mean = sgd_cv_scores.mean()

# Train and evaluate RandomForest model
rf_model.fit(X_train, y_train)
rf_accuracy = accuracy_score(y_test, rf_model.predict(X_test))
rf_cv_scores = cross_val_score(rf_model, X, y, cv=10)
rf_cv_mean = rf_cv_scores.mean()

# Train and evaluate MLP model
mlp_model.fit(X_train, y_train)
mlp_accuracy = accuracy_score(y_test, mlp_model.predict(X_test))
mlp_cv_scores = cross_val_score(mlp_model, X, y, cv=10)
mlp_cv_mean = mlp_cv_scores.mean()

# Compile results into a DataFrame
results_df = pd.DataFrame({
    'Model': ['SVM', 'SGD', 'Random Forest', 'MLP'],
    'Train-Test Split Accuracy': [svm_accuracy, sgd_accuracy, rf_accuracy, mlp_accu
    '10-Fold Cross-Validation Accuracy': [svm_cv_mean, sgd_cv_mean, rf_cv_mean, mlp
})

results_df
```

Out[5]:

	Model	Train-Test Split Accuracy	10-Fold Cross-Validation Accuracy
0	SVM	0.833795	0.852596
1	SGD	1.000000	0.998333
2	Random Forest	1.000000	1.000000
3	MLP	0.958449	0.971701

In [6]:

```
# Assuming results_df contains the accuracy and cross-validation scores for all mod

# Display the results to review
print("Model Performance Summary:")
print(results_df)

# Identify the model with the highest Train-Test Split Accuracy
best_train_test_model = results_df.iloc[results_df['Train-Test Split Accuracy'].idx

# Identify the model with the highest 10-Fold Cross-Validation Accuracy
best_cv_model = results_df.iloc[results_df['10-Fold Cross-Validation Accuracy'].idx
```

```
print("\nBest Model Based on Train-Test Split Accuracy:")
print(best_train_test_model)

print("\nBest Model Based on 10-Fold Cross-Validation Accuracy:")
print(best_cv_model)

# Depending on your criteria, you can choose which model to select
if best_cv_model['Model'] == best_train_test_model['Model']:
    print(f"\nFinal Selection: {best_cv_model['Model']} (Best overall based on both
else:
    print("\nFinal Selection Based on Cross-Validation (more reliable for generaliz
    print(best_cv_model['Model'])
```

```
Model Performance Summary:
           Model  Train-Test Split Accuracy  10-Fold Cross-Validation Accuracy
0            SVM                   0.833795                           0.852596
1            SGD                   1.000000                           0.998333
2  Random Forest                   1.000000                           1.000000
3            MLP                   0.958449                           0.971701

Best Model Based on Train-Test Split Accuracy:
Model                                    SGD
Train-Test Split Accuracy                1.0
10-Fold Cross-Validation Accuracy    0.998333
Name: 1, dtype: object

Best Model Based on 10-Fold Cross-Validation Accuracy:
Model                                Random Forest
Train-Test Split Accuracy                      1.0
10-Fold Cross-Validation Accuracy              1.0
Name: 2, dtype: object

Final Selection Based on Cross-Validation (more reliable for generalization):
Random Forest
```

In [ ]:

In [ ]: