In [48]:

```
# Data Handling
import numpy as np
import pandas as pd
pd.set_option("display.max_columns", None)

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use("ggplot")

# Preprocessing
from sklearn.model_selection import train_test_split as tts

# Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier

# Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Time
import time

# Warnings
import warnings
warnings.filterwarnings("ignore")
```

In [49]:

```
df2=pd.read_csv("D:/Amrita_University/S5/Machine Learning/Project/dataset2_ML.csv")
```

In [50]:

```
df2.head(10)
```

Out[50]:

	label	acc_x_0	acc_x_1	acc_x_2	acc_x_3	acc_x_4	acc_x_5	acc_x_6	acc_x_7	acc_x_8	acc_x_9	acc_x_10	acc_x_11	acc_x_12	acc_x_13	acc_
0	idle	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.
1	motion	0.1042	0.2377	0.2494	0.1576	0.0721	0.0231	-0.1946	-0.1585	-0.1443	-0.0637	0.0402	0.0603	0.0679	0.0631	0.
2	step	-0.0193	-0.0174	-0.0053	0.0000	0.0000	0.0000	0.0000	0.0000	-0.0010	-0.0062	0.0000	-0.0069	-0.0197	-0.0200	-0.

	label	acc_x_0	acc_x_1	acc_x_2	acc_x_3	acc_x_4	acc_x_5	acc_x_6	acc_x_7	acc_x_8	acc_x_9	acc_x_10	acc_x_11	acc_x_12	acc_x_13	acc_
3	idle	0.0300	0.0300	0.0300	0.0300	0.0300	0.0300	0.0300	0.0300	0.0300	0.0300	0.0300	0.0300	0.0300	0.0300	0.
4	fall	0.2156	0.2289	0.2470	0.2472	0.1967	0.0283	0.0563	0.0814	0.0620	0.1771	0.1613	0.0235	0.0218	0.0078	0.
5	idle	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.
6	idle	0.0200	0.0200	0.0200	0.0200	0.0200	0.0200	0.0200	0.0200	0.0200	0.0200	0.0200	0.0200	0.0200	0.0200	0.
7	fall	-0.3952	-0.4326	-0.4414	-0.4800	-0.4800	-0.4665	-0.4516	-0.4311	-0.3904	-0.3667	-0.3544	-0.3445	-0.2958	-0.1829	-0.
8	motion	0.0200	0.0200	0.0164	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0128	0.0159	0.0190	0.0157	0.0199	0.
9	fall	-0.0321	-0.0054	0.0000	-0.0130	-0.0732	-0.0900	-0.1505	-0.2072	-0.2760	-0.2557	-0.2298	-0.3495	-0.4116	-0.4200	-0.

In [51]:

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Display basic information about the dataset
print("Dataset Info Before Cleaning:")
print(df2.info())

# Identify numeric and non-numeric columns
numeric_columns = df2.select_dtypes(include=['number']).columns
non_numeric_columns = df2.select_dtypes(exclude=['number']).columns

# Handling missing values for numeric columns
numeric_imputer = SimpleImputer(strategy='mean')
df2[numeric_columns] = numeric_imputer.fit_transform(df2[numeric_columns])

# Handling missing values for non-numeric columns
non_numeric_imputer = SimpleImputer(strategy='most_frequent')
df2[non_numeric_columns] = non_numeric_imputer.fit_transform(df2[non_numeric_columns])

# Handling duplicates
df2_no_duplicates = df2.drop_duplicates()

# Display information after handling missing values and duplicates
print("\nDataset Info After Cleaning:")
print(df2_no_duplicates.info())

# Handling outliers (optional)
```

```
# You can use various techniques to detect and handle outliers based on your dataset
# Here, we use Z-score to identify and remove outliers from numeric columns
z_scores = (df2_no_duplicates[numeric_columns] - df2_no_duplicates[numeric_columns].mean()) / df2_no_duplicates[numeric_c
df2_no_outliers = df2_no_duplicates[(z_scores.abs() < 3).all(axis=1)]

# Display information after handling outliers
print("\nDataset Info After Handling Outliers:")
print(df2_no_outliers.info())

# Scale numerical features using StandardScaler
scaler = StandardScaler()
df2_scaled = pd.DataFrame(scaler.fit_transform(df2_no_outliers[numeric_columns]), columns=numeric_columns)

# Now df1_scaled is the cleaned and scaled DataFrame that you can use for further analysis or modeling
```

```
Dataset Info Before Cleaning:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 496 entries, 0 to 495
Columns: 961 entries, label to gy_z_159
dtypes: float64(960), object(1)
memory usage: 3.6+ MB
None

Dataset Info After Cleaning:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 493 entries, 0 to 494
Columns: 961 entries, label to gy_z_159
dtypes: float64(960), object(1)
memory usage: 3.6+ MB
None

Dataset Info After Handling Outliers:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 352 entries, 0 to 492
Columns: 961 entries, label to gy_z_159
dtypes: float64(960), object(1)
memory usage: 2.6+ MB
None
```

In [52]:

```
df2.isnull().sum()
```

Out[52]:

```
label       0
acc_x_0     0
acc_x_1     0
acc_x_2     0
acc_x_3     0
           ..
```

```
gy_z_155    0
gy_z_156    0
gy_z_157    0
gy_z_158    0
gy_z_159    0
Length: 961, dtype: int64
```

In [53]:

```
# Separate into independent and dependent variables (target).
X = df2.drop("label", axis = 1)
y = df2["label"]
```

In [54]:

```
## We divide into training and test set.

# We define the random seed for reproducibility.
SEED = 123
X_train,X_test,y_train,y_test = tts(X,y,
                                    test_size = 0.2,
                                    shuffle = True,
                                    random_state = SEED,
                                    stratify = y)
print(f"X train: {X_train.shape}")
print(f"X test: {X_test.shape}")
```

```
X train: (396, 960)
X test: (100, 960)
```

In [55]:

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.utils import shuffle
from sklearn.model_selection import GridSearchCV
import time
import numpy as np



SEED = 42  # Set a seed for reproducibility

# Create Extra Trees model
et = ExtraTreesClassifier(bootstrap=True, random_state=SEED)

# Shuffle the training data
X_train, y_train = shuffle(X_train, y_train, random_state=SEED)
```

```
start = time.time()

# Use cross-validation to tune hyperparameters
param_grid = {}  # Add hyperparameter grid for Extra Trees if needed
grid_search = GridSearchCV(et, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
et_best = grid_search.best_estimator_

# Fit Extra Trees model
et_best.fit(X_train, y_train)
end = time.time()
print(f"* Extra Trees => Total training time = {end - start} seconds")

# Make predictions on training and test sets
y_pred_train_et = et_best.predict(X_train)
y_pred_test_et = et_best.predict(X_test)

# Calculate metrics
accuracy_train_et = accuracy_score(np.array(y_train), y_pred_train_et)
accuracy_test_et = accuracy_score(np.array(y_test), y_pred_test_et)

cf_matrix_train_et = confusion_matrix(np.array(y_train), y_pred_train_et)
cf_matrix_test_et = confusion_matrix(np.array(y_test), y_pred_test_et)

clf_report_train_et = classification_report(np.array(y_train), y_pred_train_et)
clf_report_test_et = classification_report(np.array(y_test), y_pred_test_et)
```

```
* Extra Trees => Total training time = 5.121779203414917 seconds
```

In [56]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.utils import shuffle
from sklearn.model_selection import GridSearchCV
import time
import numpy as np



SEED = 42  # Set a seed for reproducibility

# Create Random Forest model
rf = RandomForestClassifier(random_state=SEED)

# Shuffle the training data
X_train, y_train = shuffle(X_train, y_train, random_state=SEED)
```

```
start = time.time()

# Use cross-validation to tune hyperparameters
param_grid = {}  # Add hyperparameter grid for Random Forest if needed
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
rf_best = grid_search.best_estimator_

# Fit Random Forest model
rf_best.fit(X_train, y_train)
end = time.time()
print(f"* Random Forest => Total training time = {end - start} seconds")

# Make predictions on training and test sets
y_pred_train_rf = rf_best.predict(X_train)
y_pred_test_rf = rf_best.predict(X_test)

# Calculate metrics
accuracy_train_rf = accuracy_score(np.array(y_train), y_pred_train_rf)
accuracy_test_rf = accuracy_score(np.array(y_test), y_pred_test_rf)

cf_matrix_train_rf = confusion_matrix(np.array(y_train), y_pred_train_rf)
cf_matrix_test_rf = confusion_matrix(np.array(y_test), y_pred_test_rf)

clf_report_train_rf = classification_report(np.array(y_train), y_pred_train_rf)
clf_report_test_rf = classification_report(np.array(y_test), y_pred_test_rf)
```

```
* Random Forest => Total training time = 3.4000792503356934 seconds
```

In [57]:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.utils import shuffle
from sklearn.model_selection import GridSearchCV
import time
import numpy as np



SEED = 42  # Set a seed for reproducibility

# Create SVM model
svm = SVC(random_state=SEED)

# Shuffle the training data
```

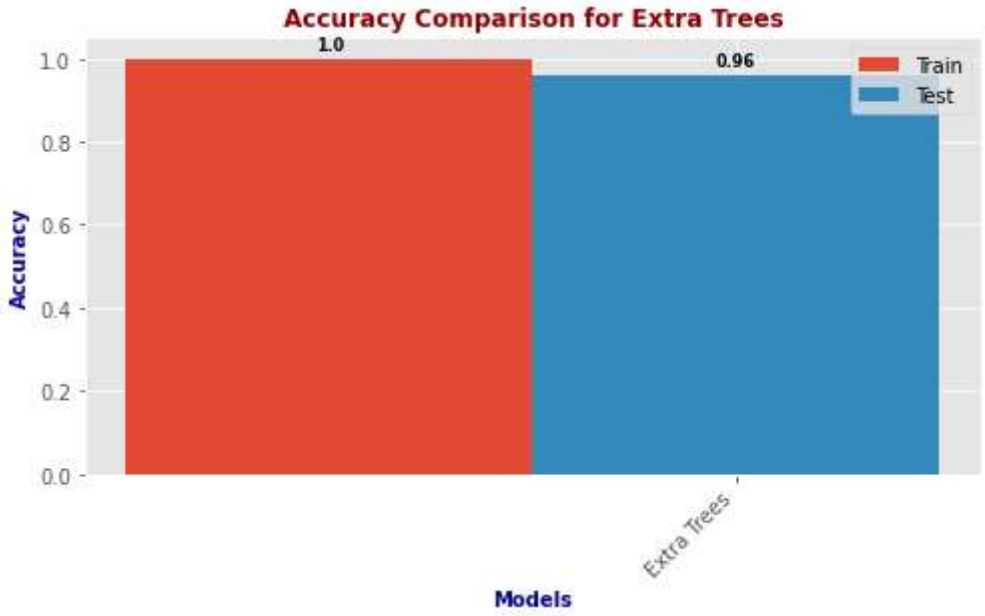
```
X_train, y_train = shuffle(X_train, y_train, random_state=SEED)

start = time.time()

# Use cross-validation to tune hyperparameters
param_grid = {}  # Add hyperparameter grid for SVM if needed
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
svm_best = grid_search.best_estimator_

# Fit SVM model
svm_best.fit(X_train, y_train)
end = time.time()
print(f"* SVM => Total training time = {end - start} seconds")

# Make predictions on training and test sets
y_pred_train_svm = svm_best.predict(X_train)
y_pred_test_svm = svm_best.predict(X_test)

# Calculate metrics
accuracy_train_svm = accuracy_score(np.array(y_train), y_pred_train_svm)
accuracy_test_svm = accuracy_score(np.array(y_test), y_pred_test_svm)

cf_matrix_train_svm = confusion_matrix(np.array(y_train), y_pred_train_svm)
cf_matrix_test_svm = confusion_matrix(np.array(y_test), y_pred_test_svm)

clf_report_train_svm = classification_report(np.array(y_train), y_pred_train_svm)
clf_report_test_svm = classification_report(np.array(y_test), y_pred_test_svm)
```

```
* SVM => Total training time = 2.015699863433838 seconds
```

In [58]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def plot_accuracy(accuracy_train: dict, accuracy_test: dict):
    df_accuracy_train = pd.DataFrame.from_dict(accuracy_train, orient='index').rename(columns={0: 'Train'})
    df_accuracy_test = pd.DataFrame.from_dict(accuracy_test, orient='index').rename(columns={0: 'Test'})
    df_accuracy = pd.merge(df_accuracy_train, df_accuracy_test, left_index=True, right_index=True)
    df_accuracy = df_accuracy.sort_values(['Train', 'Test'], ascending=False)

    fig, ax = plt.subplots(figsize=(8, 4))
    n = len(df_accuracy.index)
    x = np.arange(n)
    width = 0.3
```

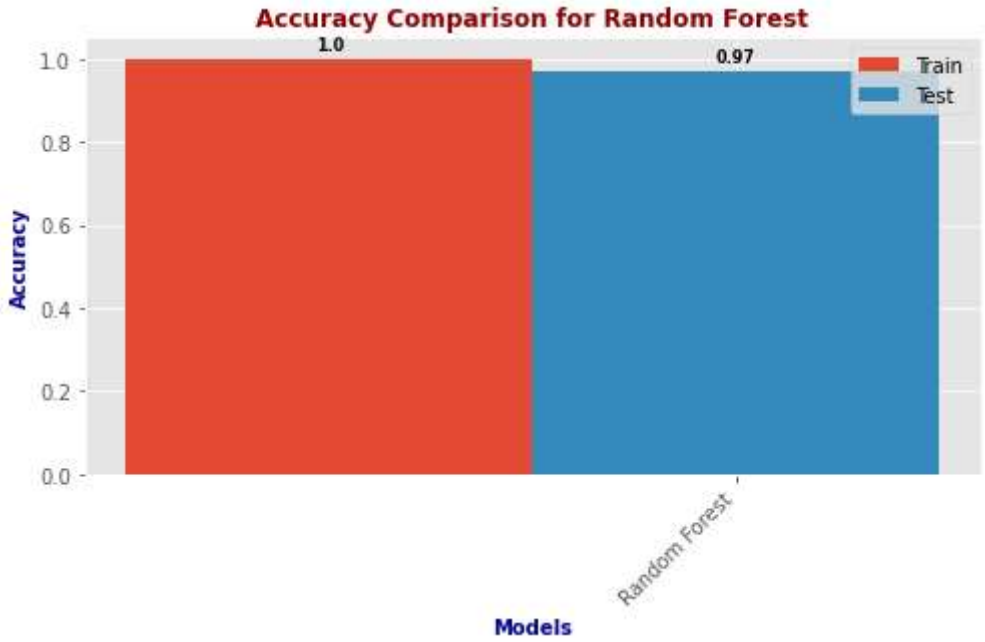
```
    rects1 = ax.bar(x=x - width, height=df_accuracy.iloc[:, 0], width=width)
    rects2 = ax.bar(x=x, height=df_accuracy.iloc[:, 1], width=width)
    ax.set_xticks(x)
    ax.set_xticklabels(df_accuracy.index.to_list(), rotation=45, ha="right")
    ax.set_xlabel('Models', fontsize=10, fontweight='bold', color='darkblue')
    ax.set_ylabel('Accuracy', fontsize=10, fontweight='bold', color='darkblue')
    ax.set_title('Accuracy Comparison for Extra Trees', fontsize=12, fontweight='bold', color='darkred')

    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.annotate(text=f'{round(height, 5)}',
                        xy=(rect.get_x() + rect.get_width() / 2, height),
                        xytext=(0, 3),
                        textcoords='offset points',
                        ha='center',
                        va='bottom',
                        size=8,
                        weight='bold',
                        color='black')

    autolabel(rects1)
    autolabel(rects2)
    ax.legend(["Train", "Test"])
    plt.show()

# Call the function to plot accuracy metrics for Extra Trees
plot_accuracy({'Extra Trees': accuracy_train_et}, {'Extra Trees': accuracy_test_et})
```



In [59]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def plot_accuracy(accuracy_train: dict, accuracy_test: dict):
    df_accuracy_train = pd.DataFrame.from_dict(accuracy_train, orient='index').rename(columns={0: 'Train'})
    df_accuracy_test = pd.DataFrame.from_dict(accuracy_test, orient='index').rename(columns={0: 'Test'})
    df_accuracy = pd.merge(df_accuracy_train, df_accuracy_test, left_index=True, right_index=True)
    df_accuracy = df_accuracy.sort_values(['Train', 'Test'], ascending=False)

    fig, ax = plt.subplots(figsize=(8, 4))
    n = len(df_accuracy.index)
    x = np.arange(n)
    width = 0.3

    rects1 = ax.bar(x=x - width, height=df_accuracy.iloc[:, 0], width=width)
    rects2 = ax.bar(x=x, height=df_accuracy.iloc[:, 1], width=width)
    ax.set_xticks(x)
    ax.set_xticklabels(df_accuracy.index.to_list(), rotation=45, ha="right")
    ax.set_xlabel('Models', fontsize=10, fontweight='bold', color='darkblue')
    ax.set_ylabel('Accuracy', fontsize=10, fontweight='bold', color='darkblue')
    ax.set_title('Accuracy Comparison for Random Forest', fontsize=12, fontweight='bold', color='darkred')

    def autolabel(rects):
```

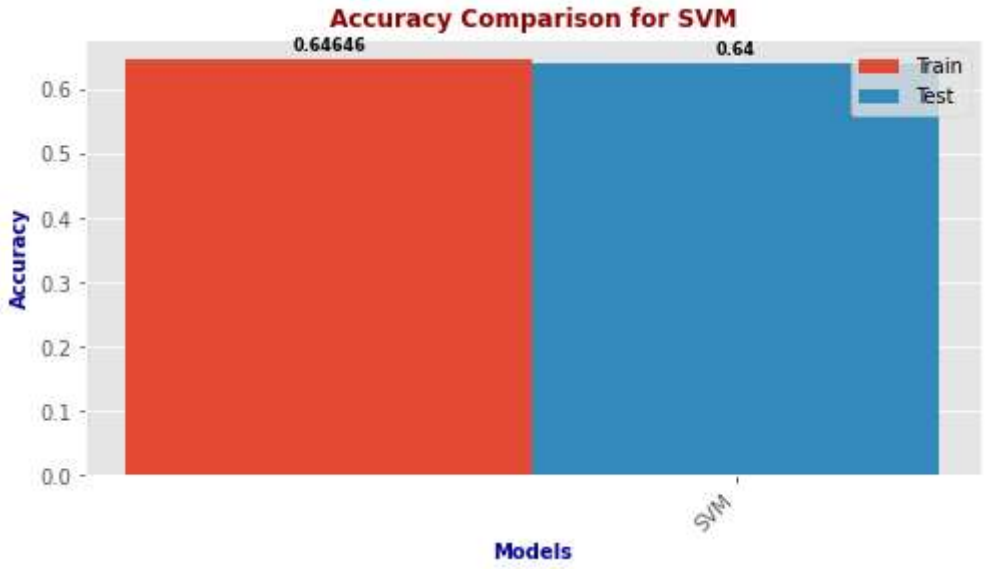
```
        for rect in rects:
            height = rect.get_height()
            ax.annotate(text=f'{round(height, 5)}',
                        xy=(rect.get_x() + rect.get_width() / 2, height),
                        xytext=(0, 3),
                        textcoords='offset points',
                        ha='center',
                        va='bottom',
                        size=8,
                        weight='bold',
                        color='black')

    autolabel(rects1)
    autolabel(rects2)
    ax.legend(["Train", "Test"])
    plt.show()

# Call the function to plot accuracy metrics for Random Forest
plot_accuracy({'Random Forest': accuracy_train_rf}, {'Random Forest': accuracy_test_rf})
```



In [60]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```python
def plot_accuracy(accuracy_train: dict, accuracy_test: dict):
    df_accuracy_train = pd.DataFrame.from_dict(accuracy_train, orient='index').rename(columns={0: 'Train'})
    df_accuracy_test = pd.DataFrame.from_dict(accuracy_test, orient='index').rename(columns={0: 'Test'})
    df_accuracy = pd.merge(df_accuracy_train, df_accuracy_test, left_index=True, right_index=True)
    df_accuracy = df_accuracy.sort_values(['Train', 'Test'], ascending=False)

    fig, ax = plt.subplots(figsize=(8, 4))
    n = len(df_accuracy.index)
    x = np.arange(n)
    width = 0.3

    rects1 = ax.bar(x=x - width, height=df_accuracy.iloc[:, 0], width=width)
    rects2 = ax.bar(x=x, height=df_accuracy.iloc[:, 1], width=width)
    ax.set_xticks(x)
    ax.set_xticklabels(df_accuracy.index.to_list(), rotation=45, ha="right")
    ax.set_xlabel('Models', fontsize=10, fontweight='bold', color='darkblue')
    ax.set_ylabel('Accuracy', fontsize=10, fontweight='bold', color='darkblue')
    ax.set_title('Accuracy Comparison for SVM', fontsize=12, fontweight='bold', color='darkred')

    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.annotate(text=f'{round(height, 5)}',
                        xy=(rect.get_x() + rect.get_width() / 2, height),
                        xytext=(0, 3),
                        textcoords='offset points',
                        ha='center',
                        va='bottom',
                        size=8,
                        weight='bold',
                        color='black')

    autolabel(rects1)
    autolabel(rects2)
    ax.legend(["Train", "Test"])
    plt.show()

# Call the function to plot accuracy metrics for SVM
plot_accuracy({'SVM': accuracy_train_svm}, {'SVM': accuracy_test_svm})
```



In [61]:

```
import numpy as np
import time
from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import GridSearchCV

# Function for enhanced feature engineering
def enhance_features(data):
    # Add more sophisticated feature engineering
    # Example: data['new_feature'] = np.log1p(data['feature1'] * data['feature2'])
    return data

# Load and preprocess your dataset
# Assuming X_train, X_test, y_train, y_test are already defined

# Enhance features
X_train = enhance_features(X_train)
X_test = enhance_features(X_test)

SEED = 42  # Set a seed for reproducibility

# Update ExtraTreesClassifier and RandomForestClassifier hyperparameter grids
et_param_grid = {
    'n_estimators': [100, 200, 300],
```

```
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    # Add more hyperparameters as needed
}

rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    # Add more hyperparameters as needed
}

svm_param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    # Add more hyperparameters as needed
}

et = ExtraTreesClassifier(bootstrap=True, random_state=SEED)
rf = RandomForestClassifier(random_state=SEED)
svm = SVC(random_state=SEED)

MODELS = {'Extra_Trees': et, 'Random_Forest': rf, 'SVM': svm}

# We store the accuracy metric
accuracy_train = {}
accuracy_test = {}

# We store the confusion matrix
cf_matrix_train = {}
cf_matrix_test = {}

# We store the classification report
clf_report_train = {}
clf_report_test = {}

for model_name, model in MODELS.items():
    start = time.time()

    # Use cross-validation to tune hyperparameters
    if model_name == 'Extra_Trees':
        param_grid = et_param_grid
    elif model_name == 'Random_Forest':
        param_grid = rf_param_grid
    else:  # SVM
```

```
        param_grid = svm_param_grid

    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
    grid_search.fit(X_train, y_train)
    model = grid_search.best_estimator_

    # Fit model
    model.fit(X_train, y_train)
    end = time.time()
    print(f"* {model_name} => Total training time = {end - start}")

    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    # Accuracy score
    accuracy_train[model_name] = accuracy_score(np.array(y_train), y_pred_train)
    accuracy_test[model_name] = accuracy_score(np.array(y_test), y_pred_test)

    # Confusion Matrix
    cf_matrix_train[model_name] = confusion_matrix(np.array(y_train), y_pred_train)
    cf_matrix_test[model_name] = confusion_matrix(np.array(y_test), y_pred_test)

    # Classification Report
    clf_report_train[model_name] = classification_report(np.array(y_train), y_pred_train)
    clf_report_test[model_name] = classification_report(np.array(y_test), y_pred_test)

# Print the results
# print("Final Results:")
# print(f"Accuracy on Train Set: {accuracy_train}")
# print(f"Accuracy on Test Set: {accuracy_test}")

# print("\n")
# print(f"Classification report:\n {clf_report_test}")
# print("\n")
# print(f"Confusion Matrix:\n {cf_matrix_test}")

# Print the results
print("Final Results:")
print("\nAccuracy on Train Set:")
for model_name, acc in accuracy_train.items():
    print(f"{model_name}: {acc:.4f}")

print("\nAccuracy on Test Set:")
for model_name, acc in accuracy_test.items():
    print(f"{model_name}: {acc:.4f}")
```

```python
print("\nClassification Report:")
for model_name, report in clf_report_test.items():
    print(f"\n{model_name}:\n{report}")

print("\nConfusion Matrix:")
for model_name, matrix in cf_matrix_test.items():
    print(f"\n{model_name}:\n{matrix}")
```

```
* Extra_Trees => Total training time = 7.258159637451172
* Random_Forest => Total training time = 10.654806852340698
* SVM => Total training time = 1.156442403793335
Final Results:

Accuracy on Train Set:
Extra_Trees: 1.0000
Random_Forest: 1.0000
SVM: 0.6970

Accuracy on Test Set:
Extra_Trees: 0.9700
Random_Forest: 0.9500
SVM: 0.6200

Classification Report:

Extra_Trees:
              precision    recall  f1-score   support

        fall       1.00      1.00      1.00        27
        idle       1.00      1.00      1.00        24
      motion       1.00      0.88      0.93        24
        step       0.89      1.00      0.94        25

    accuracy                           0.97       100
   macro avg       0.97      0.97      0.97       100
weighted avg       0.97      0.97      0.97       100


Random_Forest:
              precision    recall  f1-score   support

        fall       0.96      0.96      0.96        27
        idle       1.00      1.00      1.00        24
      motion       0.95      0.83      0.89        24
        step       0.89      1.00      0.94        25

    accuracy                           0.95       100
```

```
   macro avg       0.95      0.95      0.95       100
weighted avg       0.95      0.95      0.95       100


SVM:
              precision    recall  f1-score   support

        fall       0.90      1.00      0.95        27
        idle       0.34      0.50      0.41        24
      motion       1.00      0.50      0.67        24
        step       0.48      0.44      0.46        25

    accuracy                           0.62       100
   macro avg       0.68      0.61      0.62       100
weighted avg       0.68      0.62      0.63       100


Confusion Matrix:

Extra_Trees:
[[27  0  0  0]
 [ 0 24  0  0]
 [ 0  0 21  3]
 [ 0  0  0 25]]

Random_Forest:
[[26  0  1  0]
 [ 0 24  0  0]
 [ 1  0 20  3]
 [ 0  0  0 25]]

SVM:
[[27  0  0  0]
 [ 0 12  0 12]
 [ 3  9 12  0]
 [ 0 14  0 11]]
```

In [62]:

```
# MODELS = {'Extra_Trees': et, 'Random_Forest': rf, 'SVM': svm}

# # We store the accuracy metric
# accuracy_train = {}
# accuracy_test = {}

# # We store the confusion matrix
# cf_matrix_train = {}
# cf_matrix_test = {}
```

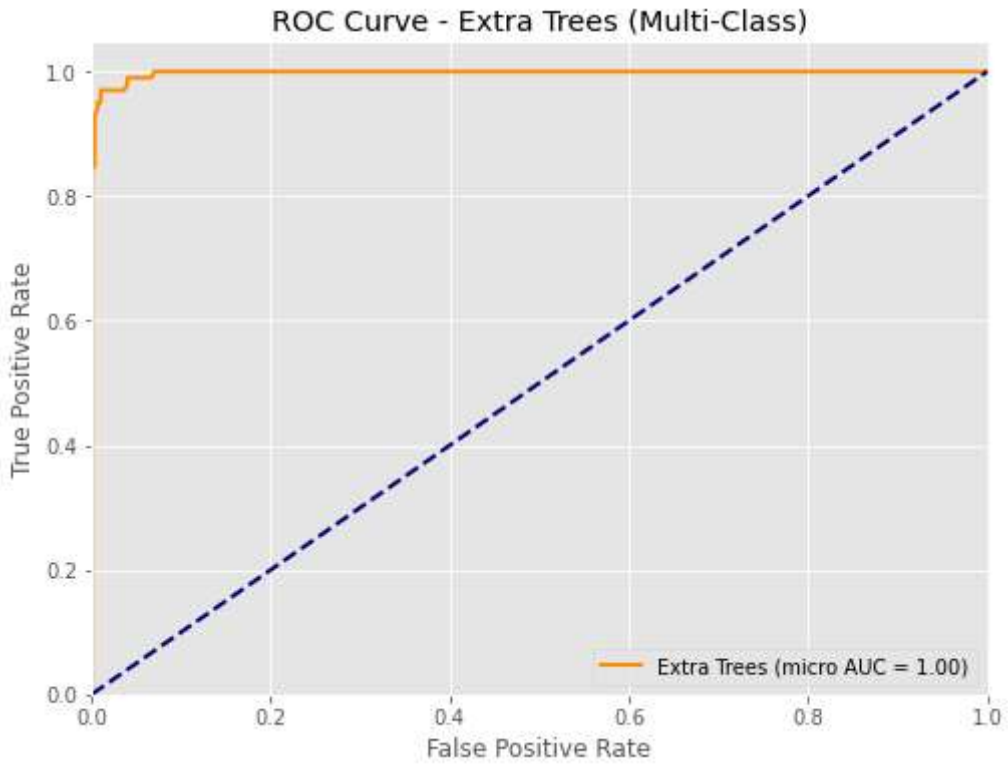
```
# # We store the classification report
# clf_report_train = {}
# clf_report_test = {}

# for model_name, model in MODELS.items():
#     start = time.time()

#     # Use cross-validation to tune hyperparameters
#     if model_name == 'Extra_Trees':
#         param_grid = et_param_grid
#     elif model_name == 'Random_Forest':
#         param_grid = rf_param_grid
#     else:  # SVM
#         param_grid = svm_param_grid

#     grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
#     grid_search.fit(X_train, y_train)
#     model = grid_search.best_estimator_

#     # Fit model
#     model.fit(X_train, y_train)
#     end = time.time()
#     print(f"* {model_name} => Total training time = {end - start}")

#     y_pred_train = model.predict(X_train)
#     y_pred_test = model.predict(X_test)

#     # Accuracy score
#     accuracy_train[model_name] = accuracy_score(np.array(y_train), y_pred_train)
#     accuracy_test[model_name] = accuracy_score(np.array(y_test), y_pred_test)

#     # Confusion Matrix
#     cf_matrix_train[model_name] = confusion_matrix(np.array(y_train), y_pred_train)
#     cf_matrix_test[model_name] = confusion_matrix(np.array(y_test), y_pred_test)

#     # Classification Report
#     clf_report_train[model_name] = classification_report(np.array(y_train), y_pred_train)
#     clf_report_test[model_name] = classification_report(np.array(y_test), y_pred_test)

#     # Store the trained model in a variable with a specific name
#     if model_name == 'Extra_Trees':
#         model_et = model
#     elif model_name == 'Random_Forest':
#         model_rf = model
```

```
#     else:  # SVM
#         model_svm = model
```

In [63]:

```
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score

# Function for enhanced feature engineering
def enhance_features(data):
    # Add more sophisticated feature engineering
    # Example: data['new_feature'] = np.log1p(data['feature1'] * data['feature2'])
    return data

# Load and preprocess your dataset
# Assuming X_train, X_test, y_train, y_test are already defined

# Enhance features
X_train = enhance_features(X_train)
X_test = enhance_features(X_test)

SEED = 42  # Set a seed for reproducibility

# Update ExtraTreesClassifier hyperparameter grid
et_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    # Add more hyperparameters as needed
}

et = ExtraTreesClassifier(bootstrap=True, random_state=SEED)

# Use cross-validation to tune hyperparameters
grid_search = GridSearchCV(et, et_param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
et = grid_search.best_estimator_

# Fit model
et.fit(X_train, y_train)
```

```python
# Convert multi-class labels to binary format
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# ROC curve for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(np.unique(y_test))):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], et.predict_proba(X_test)[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Micro-average ROC curve
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), et.predict_proba(X_test).ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr["micro"], tpr["micro"], color='darkorange', lw=2, label=f'Extra Trees (micro AUC = {roc_auc["micro"]:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Extra Trees (Multi-Class)')
plt.legend(loc="lower right")
plt.show()
```

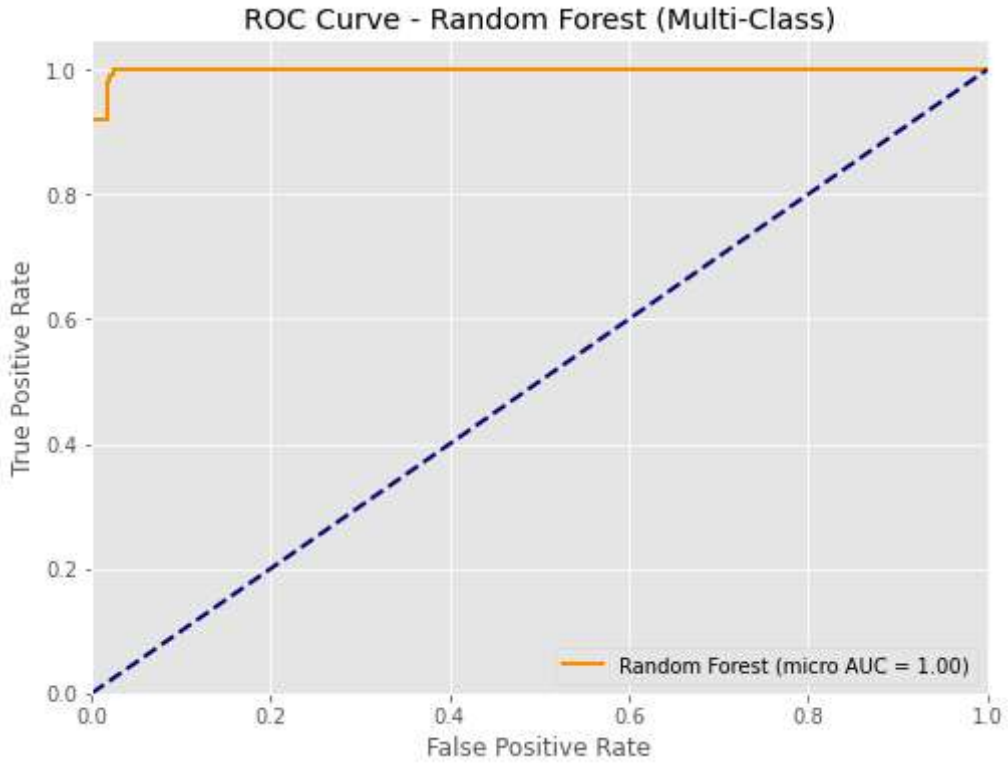


In [64]:

```
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score

# Function for enhanced feature engineering
def enhance_features(data):
    # Add more sophisticated feature engineering
    # Example: data['new_feature'] = np.log1p(data['feature1'] * data['feature2'])
    return data

# Load and preprocess your dataset
# Assuming X_train, X_test, y_train, y_test are already defined

# Enhance features
```

```
X_train = enhance_features(X_train)
X_test = enhance_features(X_test)

SEED = 42  # Set a seed for reproducibility

# Update RandomForestClassifier hyperparameter grid
rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    # Add more hyperparameters as needed
}

rf = RandomForestClassifier(random_state=SEED)

# Use cross-validation to tune hyperparameters
grid_search = GridSearchCV(rf, rf_param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
rf = grid_search.best_estimator_

# Fit model
rf.fit(X_train, y_train)

# Convert multi-class labels to binary format
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# ROC curve for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(np.unique(y_test))):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], rf.predict_proba(X_test)[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Micro-average ROC curve
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), rf.predict_proba(X_test).ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr["micro"], tpr["micro"], color='darkorange', lw=2, label=f'Random Forest (micro AUC = {roc_auc["micro"]:.2f})
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest (Multi-Class)')
plt.legend(loc="lower right")
plt.show()
```

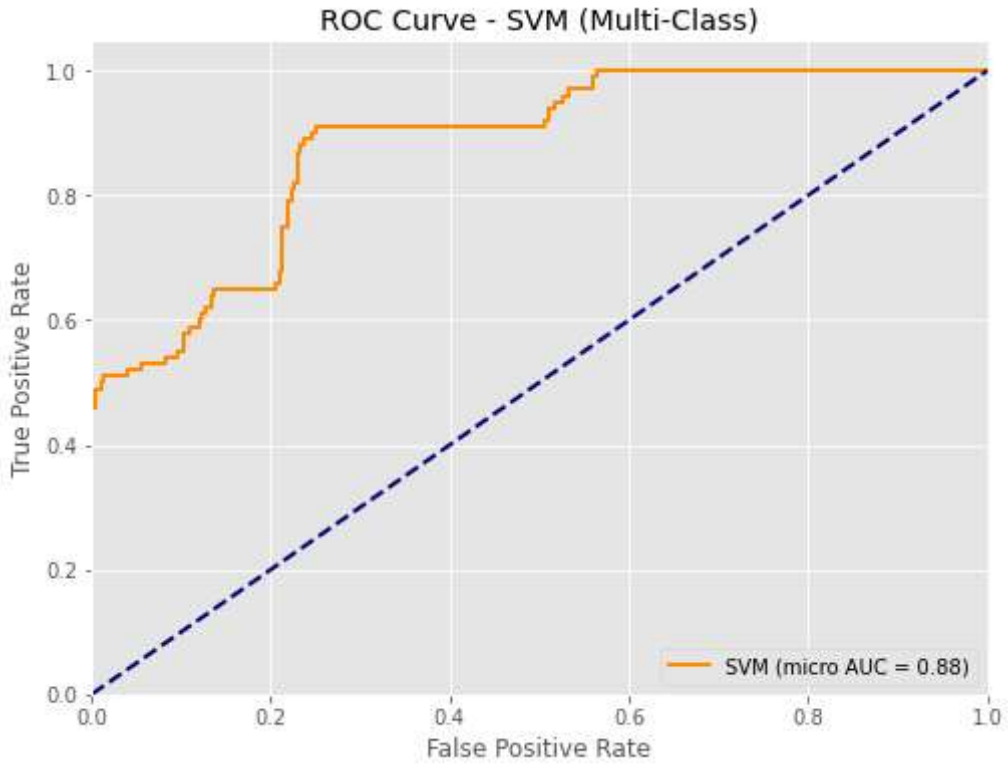


In [65]:

```
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score

# Function for enhanced feature engineering
def enhance_features(data):
    # Add more sophisticated feature engineering
    # Example: data['new_feature'] = np.log1p(data['feature1'] * data['feature2'])
    return data
```

```
# Load and preprocess your dataset
# Assuming X_train, X_test, y_train, y_test are already defined

# Enhance features
X_train = enhance_features(X_train)
X_test = enhance_features(X_test)

SEED = 42  # Set a seed for reproducibility

# Update SVM hyperparameter grid
svm_param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    # Add more hyperparameters as needed
}

svm = SVC(random_state=SEED)

# Use cross-validation to tune hyperparameters
grid_search = GridSearchCV(svm, svm_param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
svm = grid_search.best_estimator_

# Fit model
svm.fit(X_train, y_train)

# Convert multi-class labels to binary format
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# ROC curve for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(np.unique(y_test))):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], svm.decision_function(X_test)[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Micro-average ROC curve
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), svm.decision_function(X_test).ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curves
plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr["micro"], tpr["micro"], color='darkorange', lw=2, label=f'SVM (micro AUC = {roc_auc["micro"]:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - SVM (Multi-Class)')
plt.legend(loc="lower right")
plt.show()
```



In [66]:

```
from sklearn.metrics import recall_score, precision_score, f1_score, matthews_corrcoef

# Assuming y_true and y_pred are your true and predicted labels for Extra Trees
# Modify as needed based on your actual variable names
y_true_et = y_test  # Change this based on your actual variable names
y_pred_et = model_et.predict(X_test)  # Change this based on your actual variable names

# Calculate metrics for Extra Trees
recall_et = recall_score(y_true_et, y_pred_et, average='weighted')
precision_et = precision_score(y_true_et, y_pred_et, average='weighted')
```

```
f1_et = f1_score(y_true_et, y_pred_et, average='weighted')
mcc_et = matthews_corrcoef(y_true_et, y_pred_et)

# Print the metrics for Extra Trees
print(f"Recall for Extra Trees: {recall_et}")
print(f"Precision for Extra Trees: {precision_et}")
print(f"F1 Score for Extra Trees: {f1_et}")
print(f"MCC for Extra Trees: {mcc_et}")
```

```
Recall for Extra Trees: 0.97
Precision for Extra Trees: 0.9732142857142857
F1 Score for Extra Trees: 0.9698490566037736
MCC for Extra Trees: 0.9611078921196086
```

In [67]:

```
y_true_rf = y_test  # Change this based on your actual variable names
y_pred_rf = model_rf.predict(X_test)  # Change this based on your actual variable names

# Calculate metrics for Random Forest
recall_rf = recall_score(y_true_rf, y_pred_rf, average='weighted')
precision_rf = precision_score(y_true_rf, y_pred_rf, average='weighted')
f1_rf = f1_score(y_true_rf, y_pred_rf, average='weighted')
mcc_rf = matthews_corrcoef(y_true_rf, y_pred_rf)

# Print the metrics for Random Forest
print(f"Recall for Random Forest: {recall_rf}")
print(f"Precision for Random Forest: {precision_rf}")
print(f"F1 Score for Random Forest: {f1_rf}")
print(f"MCC for Random Forest: {mcc_rf}")
```

```
Recall for Random Forest: 0.95
Precision for Random Forest: 0.9517857142857143
F1 Score for Random Forest: 0.949182389937107
MCC for Random Forest: 0.934377037103071
```

In [68]:

```
y_true_svm = y_test  # Change this based on your actual variable names
y_pred_svm = model_svm.predict(X_test)  # Change this based on your actual variable names

# Calculate metrics for SVM
recall_svm = recall_score(y_true_svm, y_pred_svm, average='weighted')
precision_svm = precision_score(y_true_svm, y_pred_svm, average='weighted')
f1_svm = f1_score(y_true_svm, y_pred_svm, average='weighted')
mcc_svm = matthews_corrcoef(y_true_svm, y_pred_svm)

# Print the metrics for SVM
print(f"Recall for SVM: {recall_svm}")
```

```
print(f"Precision for SVM: {precision_svm}")
print(f"F1 Score for SVM: {f1_svm}")
print(f"MCC for SVM: {mcc_svm}")
```

```
Recall for SVM: 0.62
Precision for SVM: 0.6848509316770186
F1 Score for SVM: 0.6279999256616117
MCC for SVM: 0.5018682652510785
```

In [69]:

```
# from sklearn.model_selection import train_test_split
# from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
# from sklearn.svm import SVC
# from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# # Split your data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# # Random Forest
# rf_model = RandomForestClassifier()
# rf_model.fit(X_train, y_train)
# y_pred_rf = rf_model.predict(X_test)

# print("Random Forest Metrics:")
# print("Accuracy:", accuracy_score(y_test, y_pred_rf))
# print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
# print("Classification Report:\n", classification_report(y_test, y_pred_rf))

# # Extra Trees
# et_model = ExtraTreesClassifier()
# et_model.fit(X_train, y_train)
# y_pred_et = et_model.predict(X_test)

# print("\nExtra Trees Metrics:")
# print("Accuracy:", accuracy_score(y_test, y_pred_et))
# print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_et))
# print("Classification Report:\n", classification_report(y_test, y_pred_et))

# # SVM
# svm_model = SVC()
# svm_model.fit(X_train, y_train)
# y_pred_svm = svm_model.predict(X_test)

# print("\nSVM Metrics:")
# print("Accuracy:", accuracy_score(y_test, y_pred_svm))
```

```
# print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
# print("Classification Report:\n", classification_report(y_test, y_pred_svm))
```

In [70]:

```
# from sklearn.model_selection import train_test_split
# from sklearn.ensemble import RandomForestClassifier
# from sklearn.multiclass import OneVsRestClassifier
# from sklearn.preprocessing import label_binarize
# from sklearn.metrics import roc_curve, auc
# import matplotlib.pyplot as plt
# import numpy as np

# # Split your data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# # Create a multiclass classifier using One-vs-Rest strategy
# rf_model = OneVsRestClassifier(RandomForestClassifier())
# y_score = rf_model.fit(X_train, y_train).predict_proba(X_test)

# # Binarize the labels
# y_test_bin = label_binarize(y_test, classes=np.unique(y))

# # Plot ROC curve for each class
# plt.figure(figsize=(8, 6))

# for i in range(len(np.unique(y))):
#     fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
#     roc_auc = auc(fpr, tpr)
#     plt.plot(fpr, tpr, label=f'Class {i+1} (AUC = {roc_auc:.2f})')

# plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Guessing')
# plt.xlabel('False Positive Rate (FPR)')
# plt.ylabel('True Positive Rate (TPR)')
# plt.title('ROC Curve for Random Forest (Multiclass)')
# plt.legend(loc='lower right')
# plt.show()
```

In [71]:

```
# from sklearn.model_selection import train_test_split
# from sklearn.ensemble import ExtraTreesClassifier
# from sklearn.multiclass import OneVsRestClassifier
# from sklearn.preprocessing import label_binarize
# from sklearn.metrics import roc_curve, auc
# import matplotlib.pyplot as plt
# import numpy as np
```

```
# # Split your data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# # Create a multiclass classifier using One-vs-Rest strategy
# et_model = OneVsRestClassifier(ExtraTreesClassifier())
# y_score_et = et_model.fit(X_train, y_train).predict_proba(X_test)

# # Binarize the labels
# y_test_bin_et = label_binarize(y_test, classes=np.unique(y))

# # Plot ROC curve for each class
# plt.figure(figsize=(8, 6))

# for i in range(len(np.unique(y))):
#     fpr, tpr, _ = roc_curve(y_test_bin_et[:, i], y_score_et[:, i])
#     roc_auc = auc(fpr, tpr)
#     plt.plot(fpr, tpr, label=f'Class {i+1} (AUC = {roc_auc:.2f})')

# plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Guessing')
# plt.xlabel('False Positive Rate (FPR)')
# plt.ylabel('True Positive Rate (TPR)')
# plt.title('ROC Curve for Extra Trees (Multiclass)')
# plt.legend(loc='lower right')
# plt.show()
```

In [72]:

```
# from sklearn.model_selection import train_test_split
# from sklearn.svm import SVC
# from sklearn.multiclass import OneVsRestClassifier
# from sklearn.preprocessing import label_binarize
# from sklearn.metrics import roc_curve, auc
# import matplotlib.pyplot as plt
# import numpy as np

# # Split your data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# # Create a multiclass classifier using One-vs-Rest strategy
# svm_model = OneVsRestClassifier(SVC(probability=True))
# y_score_svm = svm_model.fit(X_train, y_train).predict_proba(X_test)

# # Binarize the labels
# y_test_bin_svm = label_binarize(y_test, classes=np.unique(y))
```

```
# # Plot ROC curve for each class
# plt.figure(figsize=(8, 6))

# for i in range(len(np.unique(y))):
#     fpr, tpr, _ = roc_curve(y_test_bin_svm[:, i], y_score_svm[:, i])
#     roc_auc = auc(fpr, tpr)
#     plt.plot(fpr, tpr, label=f'Class {i+1} (AUC = {roc_auc:.2f})')

# plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Guessing')
# plt.xlabel('False Positive Rate (FPR)')
# plt.ylabel('True Positive Rate (TPR)')
# plt.title('ROC Curve for SVM (Multiclass)')
# plt.legend(loc='lower right')
# plt.show()
```

In [ ]: