In [1]:
```
#=========================================================================
# ENSEMBLE LEARNING PROJECT
# Human Activity Recognition with Smartphones (Kaggle)
#=========================================================================

# ------- Runtime knobs -------
FAST_SEARCH = True   # True = faster hyperparam search; set False for

# 1. Importing all the important libraries
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, StratifiedKFold,
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

from sklearn.linear_model import Perceptron, RidgeClassifier, Logistic
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, VotingClassifier,
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_r

from sklearn.base import clone
from scipy.stats import loguniform, randint as sp_randint
from joblib import dump
```

In [2]:
```
# ------------------------ 1.1 Load dataset ------------------------
df = pd.read_csv("train.csv")
print("\n== Loading the Datasets ==")
print("\nDataset Shape(rows, columns): ", df.shape)
print("\nFirst 5 rows: \n", df.head(5))
print("\nColumn names: \n", df.columns.tolist())
```

```
== Loading the Datasets ==

Dataset Shape(rows, columns):  (7352, 563)

First 5 rows:
    tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBodyAcc-s
td()-X  \
0           0.288585          -0.020294          -0.132905         -0.9
95279
1           0.278419          -0.016411          -0.123520         -0.9
98245
```

```
2           0.279653          -0.019467          -0.113462         -0.9
95380
3           0.279174          -0.026201          -0.123283         -0.9
96091
4           0.276629          -0.016570          -0.115362         -0.9
98139

   tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAcc-mad()
-Y  \
0         -0.983111         -0.913526         -0.995112         -0.9831
85
1         -0.975300         -0.960322         -0.998807         -0.9749
14
2         -0.967187         -0.978944         -0.996520         -0.9636
68
3         -0.983403         -0.990675         -0.997099         -0.9827
50
4         -0.980817         -0.990482         -0.998321         -0.9796
72

   tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  fBodyBodyGyroJerkMag-kurtos
is()  \
0         -0.923527         -0.934724  ...                       -0.71
0304
1         -0.957686         -0.943068  ...                       -0.86
1499
2         -0.977469         -0.938692  ...                       -0.76
0104
3         -0.989302         -0.938692  ...                       -0.48
2845
4         -0.990441         -0.942469  ...                       -0.69
9205

   angle(tBodyAccMean,gravity)  angle(tBodyAccJerkMean),gravityMean)  \
0                    -0.112754                             0.030400
1                     0.053477                            -0.007435
2                    -0.118559                             0.177899
3                    -0.036788                            -0.012892
4                     0.123320                             0.122542

   angle(tBodyGyroMean,gravityMean)  angle(tBodyGyroJerkMean,gravityMea
n)  \
0                         -0.464761                          -0.0184
46
1                         -0.732626                           0.7035
11
2                          0.100699                           0.8085
29
3                          0.640011                          -0.4853
66
4                          0.693578                          -0.6159
71
```

```
   angle(X,gravityMean)  angle(Y,gravityMean)  angle(Z,gravityMean)  su
bject  \
0             -0.841247              0.179941             -0.058627
1
1             -0.844788              0.180289             -0.054317
1
2             -0.848933              0.180637             -0.049118
1
3             -0.848649              0.181935             -0.047663
1
4             -0.847865              0.185151             -0.043892
1

   Activity
0  STANDING
1  STANDING
2  STANDING
3  STANDING
4  STANDING

[5 rows x 563 columns]

Column names:
 ['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z', 'tBody
Acc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z', 'tBodyAcc-mad()-
X', 'tBodyAcc-mad()-Y', 'tBodyAcc-mad()-Z', 'tBodyAcc-max()-X', 'tBodyA
cc-max()-Y', 'tBodyAcc-max()-Z', 'tBodyAcc-min()-X', 'tBodyAcc-min()-
Y', 'tBodyAcc-min()-Z', 'tBodyAcc-sma()', 'tBodyAcc-energy()-X', 'tBody
Acc-energy()-Y', 'tBodyAcc-energy()-Z', 'tBodyAcc-iqr()-X', 'tBodyAcc-i
qr()-Y', 'tBodyAcc-iqr()-Z', 'tBodyAcc-entropy()-X', 'tBodyAcc-entrop
y()-Y', 'tBodyAcc-entropy()-Z', 'tBodyAcc-arCoeff()-X,1', 'tBodyAcc-arC
oeff()-X,2', 'tBodyAcc-arCoeff()-X,3', 'tBodyAcc-arCoeff()-X,4', 'tBody
Acc-arCoeff()-Y,1', 'tBodyAcc-arCoeff()-Y,2', 'tBodyAcc-arCoeff()-Y,3',
'tBodyAcc-arCoeff()-Y,4', 'tBodyAcc-arCoeff()-Z,1', 'tBodyAcc-arCoeff()
-Z,2', 'tBodyAcc-arCoeff()-Z,3', 'tBodyAcc-arCoeff()-Z,4', 'tBodyAcc-co
rrelation()-X,Y', 'tBodyAcc-correlation()-X,Z', 'tBodyAcc-correlation()
-Y,Z', 'tGravityAcc-mean()-X', 'tGravityAcc-mean()-Y', 'tGravityAcc-mea
n()-Z', 'tGravityAcc-std()-X', 'tGravityAcc-std()-Y', 'tGravityAcc-st
d()-Z', 'tGravityAcc-mad()-X', 'tGravityAcc-mad()-Y', 'tGravityAcc-ma
d()-Z', 'tGravityAcc-max()-X', 'tGravityAcc-max()-Y', 'tGravityAcc-ma
x()-Z', 'tGravityAcc-min()-X', 'tGravityAcc-min()-Y', 'tGravityAcc-mi
n()-Z', 'tGravityAcc-sma()', 'tGravityAcc-energy()-X', 'tGravityAcc-ene
rgy()-Y', 'tGravityAcc-energy()-Z', 'tGravityAcc-iqr()-X', 'tGravityAcc
-iqr()-Y', 'tGravityAcc-iqr()-Z', 'tGravityAcc-entropy()-X', 'tGravityA
cc-entropy()-Y', 'tGravityAcc-entropy()-Z', 'tGravityAcc-arCoeff()-X,
1', 'tGravityAcc-arCoeff()-X,2', 'tGravityAcc-arCoeff()-X,3', 'tGravity
Acc-arCoeff()-X,4', 'tGravityAcc-arCoeff()-Y,1', 'tGravityAcc-arCoeff()
-Y,2', 'tGravityAcc-arCoeff()-Y,3', 'tGravityAcc-arCoeff()-Y,4', 'tGrav
ityAcc-arCoeff()-Z,1', 'tGravityAcc-arCoeff()-Z,2', 'tGravityAcc-arCoef
f()-Z,3', 'tGravityAcc-arCoeff()-Z,4', 'tGravityAcc-correlation()-X,Y',
'tGravityAcc-correlation()-X,Z', 'tGravityAcc-correlation()-Y,Z', 'tBod
```

```
yAccJerk-mean()-X', 'tBodyAccJerk-mean()-Y', 'tBodyAccJerk-mean()-Z', '
tBodyAccJerk-std()-X', 'tBodyAccJerk-std()-Y', 'tBodyAccJerk-std()-Z',
'tBodyAccJerk-mad()-X', 'tBodyAccJerk-mad()-Y', 'tBodyAccJerk-mad()-Z',
'tBodyAccJerk-max()-X', 'tBodyAccJerk-max()-Y', 'tBodyAccJerk-max()-Z',
'tBodyAccJerk-min()-X', 'tBodyAccJerk-min()-Y', 'tBodyAccJerk-min()-Z',
'tBodyAccJerk-sma()', 'tBodyAccJerk-energy()-X', 'tBodyAccJerk-energy()
-Y', 'tBodyAccJerk-energy()-Z', 'tBodyAccJerk-iqr()-X', 'tBodyAccJerk-i
qr()-Y', 'tBodyAccJerk-iqr()-Z', 'tBodyAccJerk-entropy()-X', 'tBodyAccJ
erk-entropy()-Y', 'tBodyAccJerk-entropy()-Z', 'tBodyAccJerk-arCoeff()-
X,1', 'tBodyAccJerk-arCoeff()-X,2', 'tBodyAccJerk-arCoeff()-X,3', 'tBod
yAccJerk-arCoeff()-X,4', 'tBodyAccJerk-arCoeff()-Y,1', 'tBodyAccJerk-ar
Coeff()-Y,2', 'tBodyAccJerk-arCoeff()-Y,3', 'tBodyAccJerk-arCoeff()-Y,
4', 'tBodyAccJerk-arCoeff()-Z,1', 'tBodyAccJerk-arCoeff()-Z,2', 'tBodyA
ccJerk-arCoeff()-Z,3', 'tBodyAccJerk-arCoeff()-Z,4', 'tBodyAccJerk-corr
elation()-X,Y', 'tBodyAccJerk-correlation()-X,Z', 'tBodyAccJerk-correla
tion()-Y,Z', 'tBodyGyro-mean()-X', 'tBodyGyro-mean()-Y', 'tBodyGyro-mea
n()-Z', 'tBodyGyro-std()-X', 'tBodyGyro-std()-Y', 'tBodyGyro-std()-Z',
'tBodyGyro-mad()-X', 'tBodyGyro-mad()-Y', 'tBodyGyro-mad()-Z', 'tBodyGy
ro-max()-X', 'tBodyGyro-max()-Y', 'tBodyGyro-max()-Z', 'tBodyGyro-min()
-X', 'tBodyGyro-min()-Y', 'tBodyGyro-min()-Z', 'tBodyGyro-sma()', 'tBod
yGyro-energy()-X', 'tBodyGyro-energy()-Y', 'tBodyGyro-energy()-Z', 'tBo
dyGyro-iqr()-X', 'tBodyGyro-iqr()-Y', 'tBodyGyro-iqr()-Z', 'tBodyGyro-e
ntropy()-X', 'tBodyGyro-entropy()-Y', 'tBodyGyro-entropy()-Z', 'tBodyGy
ro-arCoeff()-X,1', 'tBodyGyro-arCoeff()-X,2', 'tBodyGyro-arCoeff()-X,
3', 'tBodyGyro-arCoeff()-X,4', 'tBodyGyro-arCoeff()-Y,1', 'tBodyGyro-ar
Coeff()-Y,2', 'tBodyGyro-arCoeff()-Y,3', 'tBodyGyro-arCoeff()-Y,4', 'tB
odyGyro-arCoeff()-Z,1', 'tBodyGyro-arCoeff()-Z,2', 'tBodyGyro-arCoeff()
-Z,3', 'tBodyGyro-arCoeff()-Z,4', 'tBodyGyro-correlation()-X,Y', 'tBody
Gyro-correlation()-X,Z', 'tBodyGyro-correlation()-Y,Z', 'tBodyGyroJerk-
mean()-X', 'tBodyGyroJerk-mean()-Y', 'tBodyGyroJerk-mean()-Z', 'tBodyGy
roJerk-std()-X', 'tBodyGyroJerk-std()-Y', 'tBodyGyroJerk-std()-Z', 'tBo
dyGyroJerk-mad()-X', 'tBodyGyroJerk-mad()-Y', 'tBodyGyroJerk-mad()-Z',
'tBodyGyroJerk-max()-X', 'tBodyGyroJerk-max()-Y', 'tBodyGyroJerk-max()-
Z', 'tBodyGyroJerk-min()-X', 'tBodyGyroJerk-min()-Y', 'tBodyGyroJerk-mi
n()-Z', 'tBodyGyroJerk-sma()', 'tBodyGyroJerk-energy()-X', 'tBodyGyroJe
rk-energy()-Y', 'tBodyGyroJerk-energy()-Z', 'tBodyGyroJerk-iqr()-X', 't
BodyGyroJerk-iqr()-Y', 'tBodyGyroJerk-iqr()-Z', 'tBodyGyroJerk-entrop
y()-X', 'tBodyGyroJerk-entropy()-Y', 'tBodyGyroJerk-entropy()-Z', 'tBod
yGyroJerk-arCoeff()-X,1', 'tBodyGyroJerk-arCoeff()-X,2', 'tBodyGyroJerk
-arCoeff()-X,3', 'tBodyGyroJerk-arCoeff()-X,4', 'tBodyGyroJerk-arCoef
f()-Y,1', 'tBodyGyroJerk-arCoeff()-Y,2', 'tBodyGyroJerk-arCoeff()-Y,3',
'tBodyGyroJerk-arCoeff()-Y,4', 'tBodyGyroJerk-arCoeff()-Z,1', 'tBodyGyr
oJerk-arCoeff()-Z,2', 'tBodyGyroJerk-arCoeff()-Z,3', 'tBodyGyroJerk-arC
oeff()-Z,4', 'tBodyGyroJerk-correlation()-X,Y', 'tBodyGyroJerk-correlat
ion()-X,Z', 'tBodyGyroJerk-correlation()-Y,Z', 'tBodyAccMag-mean()', 't
BodyAccMag-std()', 'tBodyAccMag-mad()', 'tBodyAccMag-max()', 'tBodyAccM
ag-min()', 'tBodyAccMag-sma()', 'tBodyAccMag-energy()', 'tBodyAccMag-iq
r()', 'tBodyAccMag-entropy()', 'tBodyAccMag-arCoeff()1', 'tBodyAccMag-a
rCoeff()2', 'tBodyAccMag-arCoeff()3', 'tBodyAccMag-arCoeff()4', 'tGravi
tyAccMag-mean()', 'tGravityAccMag-std()', 'tGravityAccMag-mad()', 'tGra
vityAccMag-max()', 'tGravityAccMag-min()', 'tGravityAccMag-sma()', 'tGr
avityAccMag-energy()', 'tGravityAccMag-iqr()', 'tGravityAccMag-entrop
```

y()', 'tGravityAccMag-arCoeff()1', 'tGravityAccMag-arCoeff()2', 'tGravityAccMag-arCoeff()3', 'tGravityAccMag-arCoeff()4', 'tBodyAccJerkMag-mean()', 'tBodyAccJerkMag-std()', 'tBodyAccJerkMag-mad()', 'tBodyAccJerkMag-max()', 'tBodyAccJerkMag-min()', 'tBodyAccJerkMag-sma()', 'tBodyAccJerkMag-energy()', 'tBodyAccJerkMag-iqr()', 'tBodyAccJerkMag-entropy()', 'tBodyAccJerkMag-arCoeff()1', 'tBodyAccJerkMag-arCoeff()2', 'tBodyAccJerkMag-arCoeff()3', 'tBodyAccJerkMag-arCoeff()4', 'tBodyGyroMag-mean()', 'tBodyGyroMag-std()', 'tBodyGyroMag-mad()', 'tBodyGyroMag-max()', 'tBodyGyroMag-min()', 'tBodyGyroMag-sma()', 'tBodyGyroMag-energy()', 'tBodyGyroMag-iqr()', 'tBodyGyroMag-entropy()', 'tBodyGyroMag-arCoeff()1', 'tBodyGyroMag-arCoeff()2', 'tBodyGyroMag-arCoeff()3', 'tBodyGyroMag-arCoeff()4', 'tBodyGyroJerkMag-mean()', 'tBodyGyroJerkMag-std()', 'tBodyGyroJerkMag-mad()', 'tBodyGyroJerkMag-max()', 'tBodyGyroJerkMag-min()', 'tBodyGyroJerkMag-sma()', 'tBodyGyroJerkMag-energy()', 'tBodyGyroJerkMag-iqr()', 'tBodyGyroJerkMag-entropy()', 'tBodyGyroJerkMag-arCoeff()1', 'tBodyGyroJerkMag-arCoeff()2', 'tBodyGyroJerkMag-arCoeff()3', 'tBodyGyroJerkMag-arCoeff()4', 'fBodyAcc-mean()-X', 'fBodyAcc-mean()-Y', 'fBodyAcc-mean()-Z', 'fBodyAcc-std()-X', 'fBodyAcc-std()-Y', 'fBodyAcc-std()-Z', 'fBodyAcc-mad()-X', 'fBodyAcc-mad()-Y', 'fBodyAcc-mad()-Z', 'fBodyAcc-max()-X', 'fBodyAcc-max()-Y', 'fBodyAcc-max()-Z', 'fBodyAcc-min()-X', 'fBodyAcc-min()-Y', 'fBodyAcc-min()-Z', 'fBodyAcc-sma()', 'fBodyAcc-energy()-X', 'fBodyAcc-energy()-Y', 'fBodyAcc-energy()-Z', 'fBodyAcc-iqr()-X', 'fBodyAcc-iqr()-Y', 'fBodyAcc-iqr()-Z', 'fBodyAcc-entropy()-X', 'fBodyAcc-entropy()-Y', 'fBodyAcc-entropy()-Z', 'fBodyAcc-maxInds-X', 'fBodyAcc-maxInds-Y', 'fBodyAcc-maxInds-Z', 'fBodyAcc-meanFreq()-X', 'fBodyAcc-meanFreq()-Y', 'fBodyAcc-meanFreq()-Z', 'fBodyAcc-skewness()-X', 'fBodyAcc-kurtosis()-X', 'fBodyAcc-skewness()-Y', 'fBodyAcc-kurtosis()-Y', 'fBodyAcc-skewness()-Z', 'fBodyAcc-kurtosis()-Z', 'fBodyAcc-bandsEnergy()-1,8', 'fBodyAcc-bandsEnergy()-9,16', 'fBodyAcc-bandsEnergy()-17,24', 'fBodyAcc-bandsEnergy()-25,32', 'fBodyAcc-bandsEnergy()-33,40', 'fBodyAcc-bandsEnergy()-41,48', 'fBodyAcc-bandsEnergy()-49,56', 'fBodyAcc-bandsEnergy()-57,64', 'fBodyAcc-bandsEnergy()-1,16', 'fBodyAcc-bandsEnergy()-17,32', 'fBodyAcc-bandsEnergy()-33,48', 'fBodyAcc-bandsEnergy()-49,64', 'fBodyAcc-bandsEnergy()-1,24', 'fBodyAcc-bandsEnergy()-25,48', 'fBodyAcc-bandsEnergy()-1,8.1', 'fBodyAcc-bandsEnergy()-9,16.1', 'fBodyAcc-bandsEnergy()-17,24.1', 'fBodyAcc-bandsEnergy()-25,32.1', 'fBodyAcc-bandsEnergy()-33,40.1', 'fBodyAcc-bandsEnergy()-41,48.1', 'fBodyAcc-bandsEnergy()-49,56.1', 'fBodyAcc-bandsEnergy()-57,64.1', 'fBodyAcc-bandsEnergy()-1,16.1', 'fBodyAcc-bandsEnergy()-17,32.1', 'fBodyAcc-bandsEnergy()-33,48.1', 'fBodyAcc-bandsEnergy()-49,64.1', 'fBodyAcc-bandsEnergy()-1,24.1', 'fBodyAcc-bandsEnergy()-25,48.1', 'fBodyAcc-bandsEnergy()-1,8.2', 'fBodyAcc-bandsEnergy()-9,16.2', 'fBodyAcc-bandsEnergy()-17,24.2', 'fBodyAcc-bandsEnergy()-25,32.2', 'fBodyAcc-bandsEnergy()-33,40.2', 'fBodyAcc-bandsEnergy()-41,48.2', 'fBodyAcc-bandsEnergy()-49,56.2', 'fBodyAcc-bandsEnergy()-57,64.2', 'fBodyAcc-bandsEnergy()-1,16.2', 'fBodyAcc-bandsEnergy()-17,32.2', 'fBodyAcc-bandsEnergy()-33,48.2', 'fBodyAcc-bandsEnergy()-49,64.2', 'fBodyAcc-bandsEnergy()-1,24.2', 'fBodyAcc-bandsEnergy()-25,48.2', 'fBodyAccJerk-mean()-X', 'fBodyAccJerk-mean()-Y', 'fBodyAccJerk-mean()-Z', 'fBodyAccJerk-std()-X', 'fBodyAccJerk-std()-Y', 'fBodyAccJerk-std()-Z', 'fBodyAccJerk-mad()-X', 'fBodyAccJerk-mad()-Y', 'fBodyAccJerk-mad()-Z', 'fBodyAccJerk-max()-X', 'fBodyAccJerk-max()-Y', 'fBodyAccJerk-max()-Z', 'fBodyAccJerk-min()-X', 'fBodyAccJerk-min()-Y',

'fBodyAccJerk-min()-Z', 'fBodyAccJerk-sma()', 'fBodyAccJerk-energy()-
X', 'fBodyAccJerk-energy()-Y', 'fBodyAccJerk-energy()-Z', 'fBodyAccJerk
-iqr()-X', 'fBodyAccJerk-iqr()-Y', 'fBodyAccJerk-iqr()-Z', 'fBodyAccJer
k-entropy()-X', 'fBodyAccJerk-entropy()-Y', 'fBodyAccJerk-entropy()-Z',
'fBodyAccJerk-maxInds-X', 'fBodyAccJerk-maxInds-Y', 'fBodyAccJerk-maxIn
ds-Z', 'fBodyAccJerk-meanFreq()-X', 'fBodyAccJerk-meanFreq()-Y', 'fBody
AccJerk-meanFreq()-Z', 'fBodyAccJerk-skewness()-X', 'fBodyAccJerk-kurto
sis()-X', 'fBodyAccJerk-skewness()-Y', 'fBodyAccJerk-kurtosis()-Y', 'fB
odyAccJerk-skewness()-Z', 'fBodyAccJerk-kurtosis()-Z', 'fBodyAccJerk-ba
ndsEnergy()-1,8', 'fBodyAccJerk-bandsEnergy()-9,16', 'fBodyAccJerk-band
sEnergy()-17,24', 'fBodyAccJerk-bandsEnergy()-25,32', 'fBodyAccJerk-ban
dsEnergy()-33,40', 'fBodyAccJerk-bandsEnergy()-41,48', 'fBodyAccJerk-ba
ndsEnergy()-49,56', 'fBodyAccJerk-bandsEnergy()-57,64', 'fBodyAccJerk-b
andsEnergy()-1,16', 'fBodyAccJerk-bandsEnergy()-17,32', 'fBodyAccJerk-b
andsEnergy()-33,48', 'fBodyAccJerk-bandsEnergy()-49,64', 'fBodyAccJerk-
bandsEnergy()-1,24', 'fBodyAccJerk-bandsEnergy()-25,48', 'fBodyAccJerk-
bandsEnergy()-1,8.1', 'fBodyAccJerk-bandsEnergy()-9,16.1', 'fBodyAccJer
k-bandsEnergy()-17,24.1', 'fBodyAccJerk-bandsEnergy()-25,32.1', 'fBodyA
ccJerk-bandsEnergy()-33,40.1', 'fBodyAccJerk-bandsEnergy()-41,48.1', 'f
BodyAccJerk-bandsEnergy()-49,56.1', 'fBodyAccJerk-bandsEnergy()-57,64.
1', 'fBodyAccJerk-bandsEnergy()-1,16.1', 'fBodyAccJerk-bandsEnergy()-1
7,32.1', 'fBodyAccJerk-bandsEnergy()-33,48.1', 'fBodyAccJerk-bandsEnerg
y()-49,64.1', 'fBodyAccJerk-bandsEnergy()-1,24.1', 'fBodyAccJerk-bandsE
nergy()-25,48.1', 'fBodyAccJerk-bandsEnergy()-1,8.2', 'fBodyAccJerk-ban
dsEnergy()-9,16.2', 'fBodyAccJerk-bandsEnergy()-17,24.2', 'fBodyAccJerk
-bandsEnergy()-25,32.2', 'fBodyAccJerk-bandsEnergy()-33,40.2', 'fBodyAc
cJerk-bandsEnergy()-41,48.2', 'fBodyAccJerk-bandsEnergy()-49,56.2', 'fB
odyAccJerk-bandsEnergy()-57,64.2', 'fBodyAccJerk-bandsEnergy()-1,16.2',
'fBodyAccJerk-bandsEnergy()-17,32.2', 'fBodyAccJerk-bandsEnergy()-33,4
8.2', 'fBodyAccJerk-bandsEnergy()-49,64.2', 'fBodyAccJerk-bandsEnergy()
-1,24.2', 'fBodyAccJerk-bandsEnergy()-25,48.2', 'fBodyGyro-mean()-X', '
fBodyGyro-mean()-Y', 'fBodyGyro-mean()-Z', 'fBodyGyro-std()-X', 'fBodyG
yro-std()-Y', 'fBodyGyro-std()-Z', 'fBodyGyro-mad()-X', 'fBodyGyro-ma
d()-Y', 'fBodyGyro-mad()-Z', 'fBodyGyro-max()-X', 'fBodyGyro-max()-Y',
'fBodyGyro-max()-Z', 'fBodyGyro-min()-X', 'fBodyGyro-min()-Y', 'fBodyGy
ro-min()-Z', 'fBodyGyro-sma()', 'fBodyGyro-energy()-X', 'fBodyGyro-ener
gy()-Y', 'fBodyGyro-energy()-Z', 'fBodyGyro-iqr()-X', 'fBodyGyro-iqr()-
Y', 'fBodyGyro-iqr()-Z', 'fBodyGyro-entropy()-X', 'fBodyGyro-entropy()-
Y', 'fBodyGyro-entropy()-Z', 'fBodyGyro-maxInds-X', 'fBodyGyro-maxInds-
Y', 'fBodyGyro-maxInds-Z', 'fBodyGyro-meanFreq()-X', 'fBodyGyro-meanFre
q()-Y', 'fBodyGyro-meanFreq()-Z', 'fBodyGyro-skewness()-X', 'fBodyGyro-
kurtosis()-X', 'fBodyGyro-skewness()-Y', 'fBodyGyro-kurtosis()-Y', 'fBo
dyGyro-skewness()-Z', 'fBodyGyro-kurtosis()-Z', 'fBodyGyro-bandsEnerg
y()-1,8', 'fBodyGyro-bandsEnergy()-9,16', 'fBodyGyro-bandsEnergy()-17,2
4', 'fBodyGyro-bandsEnergy()-25,32', 'fBodyGyro-bandsEnergy()-33,40', '
fBodyGyro-bandsEnergy()-41,48', 'fBodyGyro-bandsEnergy()-49,56', 'fBody
Gyro-bandsEnergy()-57,64', 'fBodyGyro-bandsEnergy()-1,16', 'fBodyGyro-b
andsEnergy()-17,32', 'fBodyGyro-bandsEnergy()-33,48', 'fBodyGyro-bandsE
nergy()-49,64', 'fBodyGyro-bandsEnergy()-1,24', 'fBodyGyro-bandsEnerg
y()-25,48', 'fBodyGyro-bandsEnergy()-1,8.1', 'fBodyGyro-bandsEnergy()-
9,16.1', 'fBodyGyro-bandsEnergy()-17,24.1', 'fBodyGyro-bandsEnergy()-2
5,32.1', 'fBodyGyro-bandsEnergy()-33,40.1', 'fBodyGyro-bandsEnergy()-4

```
1,48.1', 'fBodyGyro-bandsEnergy()-49,56.1', 'fBodyGyro-bandsEnergy()-5
7,64.1', 'fBodyGyro-bandsEnergy()-1,16.1', 'fBodyGyro-bandsEnergy()-17,
32.1', 'fBodyGyro-bandsEnergy()-33,48.1', 'fBodyGyro-bandsEnergy()-49,6
4.1', 'fBodyGyro-bandsEnergy()-1,24.1', 'fBodyGyro-bandsEnergy()-25,48.
1', 'fBodyGyro-bandsEnergy()-1,8.2', 'fBodyGyro-bandsEnergy()-9,16.2',
'fBodyGyro-bandsEnergy()-17,24.2', 'fBodyGyro-bandsEnergy()-25,32.2', '
fBodyGyro-bandsEnergy()-33,40.2', 'fBodyGyro-bandsEnergy()-41,48.2', 'f
BodyGyro-bandsEnergy()-49,56.2', 'fBodyGyro-bandsEnergy()-57,64.2', 'fB
odyGyro-bandsEnergy()-1,16.2', 'fBodyGyro-bandsEnergy()-17,32.2', 'fBod
yGyro-bandsEnergy()-33,48.2', 'fBodyGyro-bandsEnergy()-49,64.2', 'fBody
Gyro-bandsEnergy()-1,24.2', 'fBodyGyro-bandsEnergy()-25,48.2', 'fBodyAc
cMag-mean()', 'fBodyAccMag-std()', 'fBodyAccMag-mad()', 'fBodyAccMag-ma
x()', 'fBodyAccMag-min()', 'fBodyAccMag-sma()', 'fBodyAccMag-energy()',
'fBodyAccMag-iqr()', 'fBodyAccMag-entropy()', 'fBodyAccMag-maxInds', 'f
BodyAccMag-meanFreq()', 'fBodyAccMag-skewness()', 'fBodyAccMag-kurtosi
s()', 'fBodyBodyAccJerkMag-mean()', 'fBodyBodyAccJerkMag-std()', 'fBody
BodyAccJerkMag-mad()', 'fBodyBodyAccJerkMag-max()', 'fBodyBodyAccJerkMa
g-min()', 'fBodyBodyAccJerkMag-sma()', 'fBodyBodyAccJerkMag-energy()',
'fBodyBodyAccJerkMag-iqr()', 'fBodyBodyAccJerkMag-entropy()', 'fBodyBod
yAccJerkMag-maxInds', 'fBodyBodyAccJerkMag-meanFreq()', 'fBodyBodyAccJe
rkMag-skewness()', 'fBodyBodyAccJerkMag-kurtosis()', 'fBodyBodyGyroMag-
mean()', 'fBodyBodyGyroMag-std()', 'fBodyBodyGyroMag-mad()', 'fBodyBody
GyroMag-max()', 'fBodyBodyGyroMag-min()', 'fBodyBodyGyroMag-sma()', 'fB
odyBodyGyroMag-energy()', 'fBodyBodyGyroMag-iqr()', 'fBodyBodyGyroMag-e
ntropy()', 'fBodyBodyGyroMag-maxInds', 'fBodyBodyGyroMag-meanFreq()', '
fBodyBodyGyroMag-skewness()', 'fBodyBodyGyroMag-kurtosis()', 'fBodyBody
GyroJerkMag-mean()', 'fBodyBodyGyroJerkMag-std()', 'fBodyBodyGyroJerkMa
g-mad()', 'fBodyBodyGyroJerkMag-max()', 'fBodyBodyGyroJerkMag-min()', '
fBodyBodyGyroJerkMag-sma()', 'fBodyBodyGyroJerkMag-energy()', 'fBodyBod
yGyroJerkMag-iqr()', 'fBodyBodyGyroJerkMag-entropy()', 'fBodyBodyGyroJe
rkMag-maxInds', 'fBodyBodyGyroJerkMag-meanFreq()', 'fBodyBodyGyroJerkMa
g-skewness()', 'fBodyBodyGyroJerkMag-kurtosis()', 'angle(tBodyAccMean,g
ravity)', 'angle(tBodyAccJerkMean),gravityMean)', 'angle(tBodyGyroMean,
gravityMean)', 'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravity
Mean)', 'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'subject', 'Act
ivity']
```

In [3]:

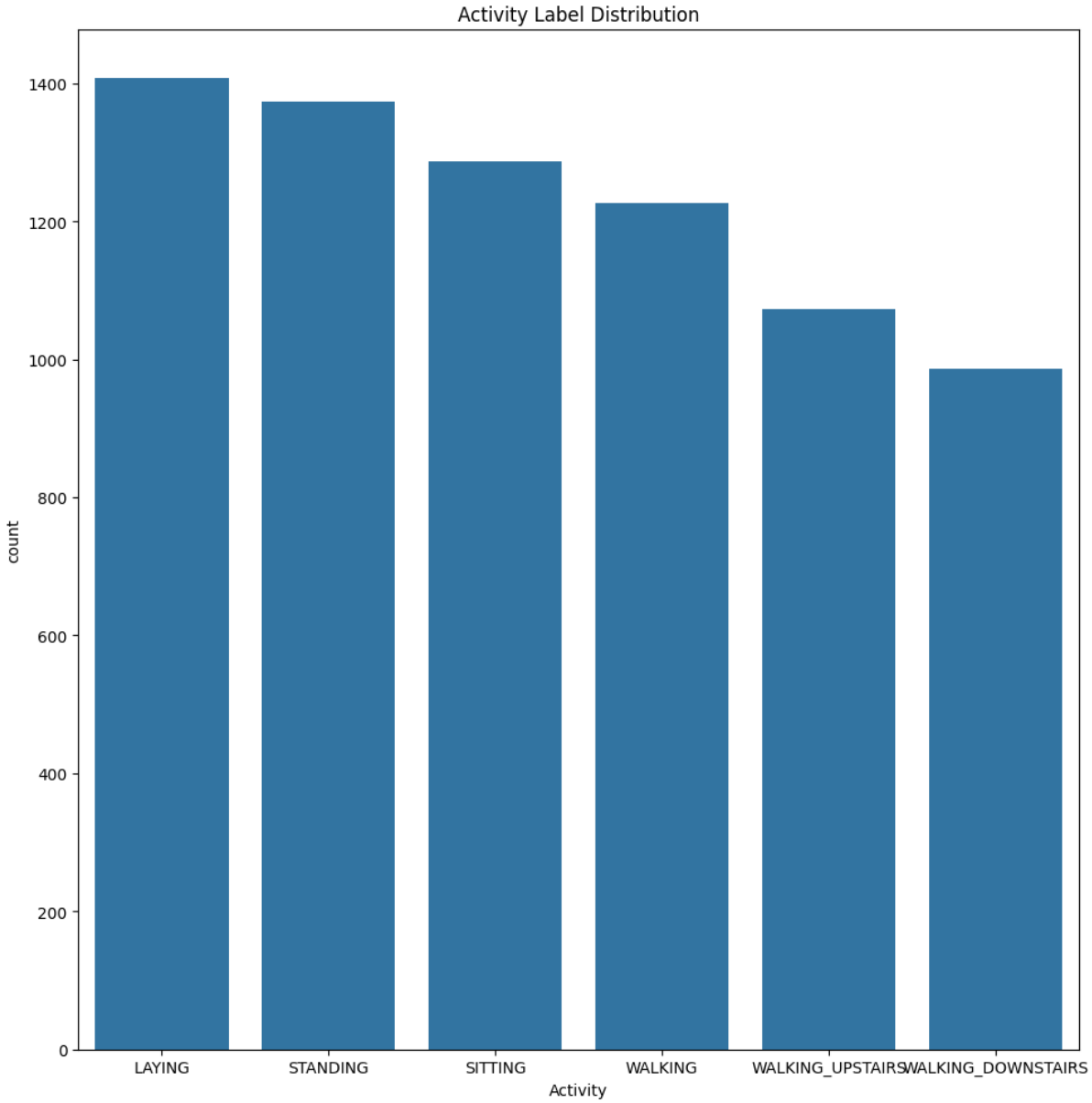
```
# ------------------------ 2. Data Exploration ----------------------
print("\n== Basic info == ")
print(df.info())

total_missing = df.isna().sum().sum()
print("\nTotal number of missing values in the entire dataset: ", tota

missing_per_column = df.isna().sum()
print("\nMissing values per column (only columns with missing values):
print(missing_per_column[missing_per_column > 0])

print("\n== Numerical Summary Statistics for the first 10 features ==="
print(df.describe().T.head(10))

print("\n== Activity Label Distribution ==")
activity_counts = df["Activity"].value_counts()
```

```
print(activity_counts)

plt.figure(figsize=(10, 10))
sns.countplot(x="Activity", data=df, order=activity_counts.index)
plt.title("Activity Label Distribution")
plt.tight_layout()
plt.show()

print("\n== Number of samples per subject ==")
subject_count = df["subject"].value_counts().sort_index()
print(subject_count.head(10))

plt.figure(figsize=(10, 10))
subject_count.plot(kind="bar")
plt.title("Number of Samples per Subject")
plt.xlabel("Subject ID")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# 2.5 Look for Outliers and key features
features_to_inspect = ["tBodyAcc-mean()-X", "tBodyAcc-mean()-Y", "tBod
print("\n== Example Features to inspect ==")
print(features_to_inspect)

fig, axes = plt.subplots(2, 2, figsize=(8, 8))
for ax, col in zip(axes.ravel(), features_to_inspect):
    ax.hist(df[col], bins=30)
    ax.set_title(col)
fig.suptitle("Histograms of selected features", y=1.02)
fig.tight_layout()
plt.show()

fig, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(data=df[features_to_inspect], ax=ax)
ax.set_title("Boxplot of selected features (outlier inspection)")
ax.set_ylabel("Standardized measurement units")
ax.tick_params(axis='x', rotation=20)
fig.tight_layout()
plt.show()

corr_subset = df[features_to_inspect].corr()
print("\n== Correlation Matrix for the selected features==")
print(corr_subset)

fig, ax = plt.subplots(figsize=(8, 8))
sns.heatmap(corr_subset, annot=True, fmt=".2f", cmap="coolwarm", squar
ax.set_title("Correlation HeatMap for selected features")
fig.tight_layout()
plt.show()
```

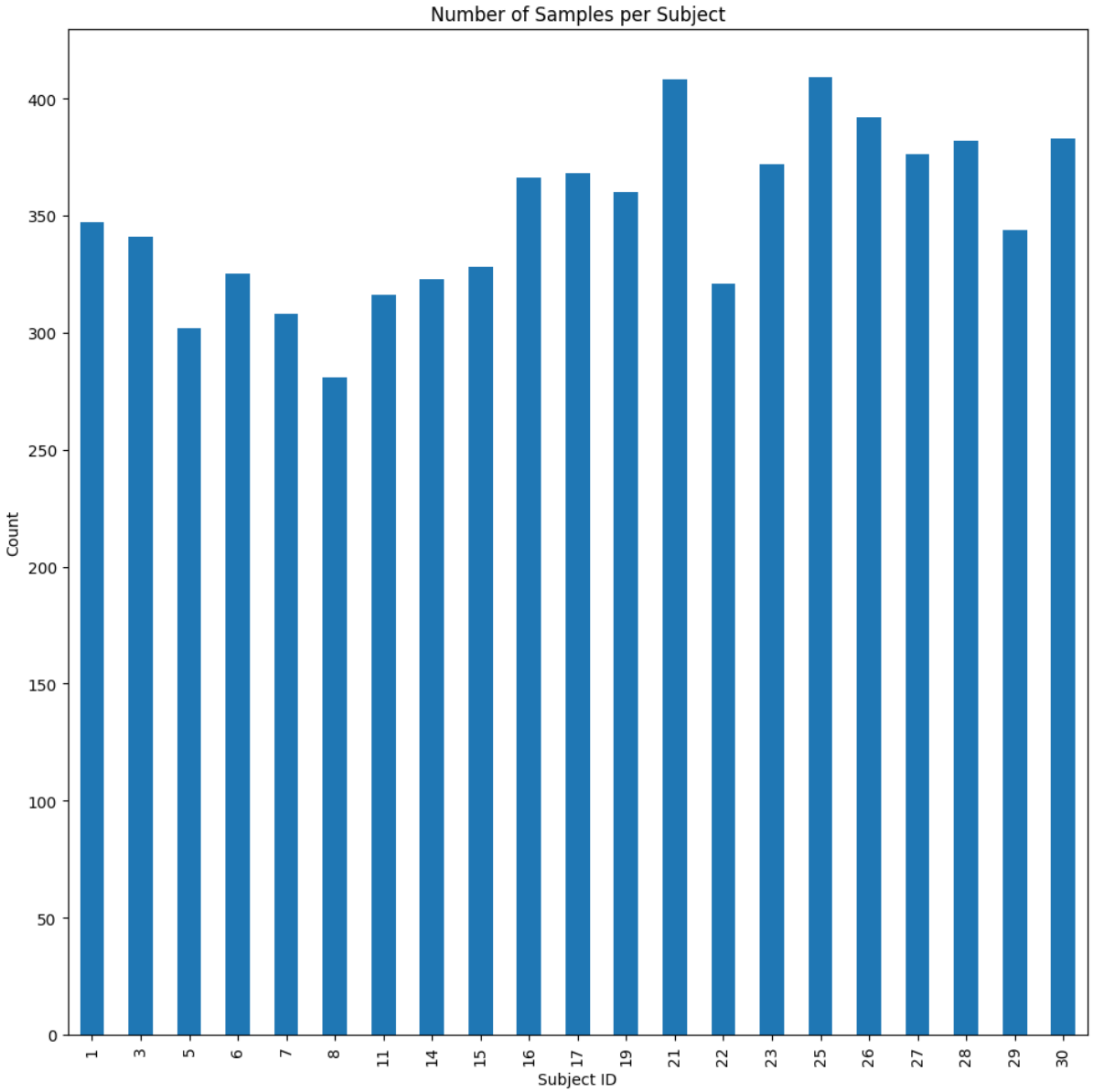
```
== Basic info ==
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 7352 entries, 0 to 7351
Columns: 563 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), int64(1), object(1)
memory usage: 31.6+ MB
None

Total number of missing values in the entire dataset:  0

Missing values per column (only columns with missing values):
Series([], dtype: int64)

== Numerical Summary Statistics for the first 10 features ==
                     count      mean       std       min       25%
50%  \
tBodyAcc-mean()-X  7352.0  0.274488  0.070261 -1.000000  0.262975  0.27
7193
tBodyAcc-mean()-Y  7352.0 -0.017695  0.040811 -1.000000 -0.024863 -0.01
7219
tBodyAcc-mean()-Z  7352.0 -0.109141  0.056635 -1.000000 -0.120993 -0.10
8676
tBodyAcc-std()-X   7352.0 -0.605438  0.448734 -1.000000 -0.992754 -0.94
6196
tBodyAcc-std()-Y   7352.0 -0.510938  0.502645 -0.999873 -0.978129 -0.85
1897
tBodyAcc-std()-Z   7352.0 -0.604754  0.418687 -1.000000 -0.980233 -0.85
9365
tBodyAcc-mad()-X   7352.0 -0.630512  0.424073 -1.000000 -0.993591 -0.95
0709
tBodyAcc-mad()-Y   7352.0 -0.526907  0.485942 -1.000000 -0.978162 -0.85
7328
tBodyAcc-mad()-Z   7352.0 -0.606150  0.414122 -1.000000 -0.980251 -0.85
7143
tBodyAcc-max()-X   7352.0 -0.468604  0.544547 -1.000000 -0.936219 -0.88
1637

                         75%       max
tBodyAcc-mean()-X  0.288461  1.000000
tBodyAcc-mean()-Y -0.010783  1.000000
tBodyAcc-mean()-Z -0.097794  1.000000
tBodyAcc-std()-X  -0.242813  1.000000
tBodyAcc-std()-Y  -0.034231  0.916238
tBodyAcc-std()-Z  -0.262415  1.000000
tBodyAcc-mad()-X  -0.292680  1.000000
tBodyAcc-mad()-Y  -0.066701  0.967664
tBodyAcc-mad()-Z  -0.265671  1.000000
tBodyAcc-max()-X  -0.017129  1.000000

== Activity Label Distribution ==
Activity
LAYING                1407
STANDING              1374
SITTING               1286
```

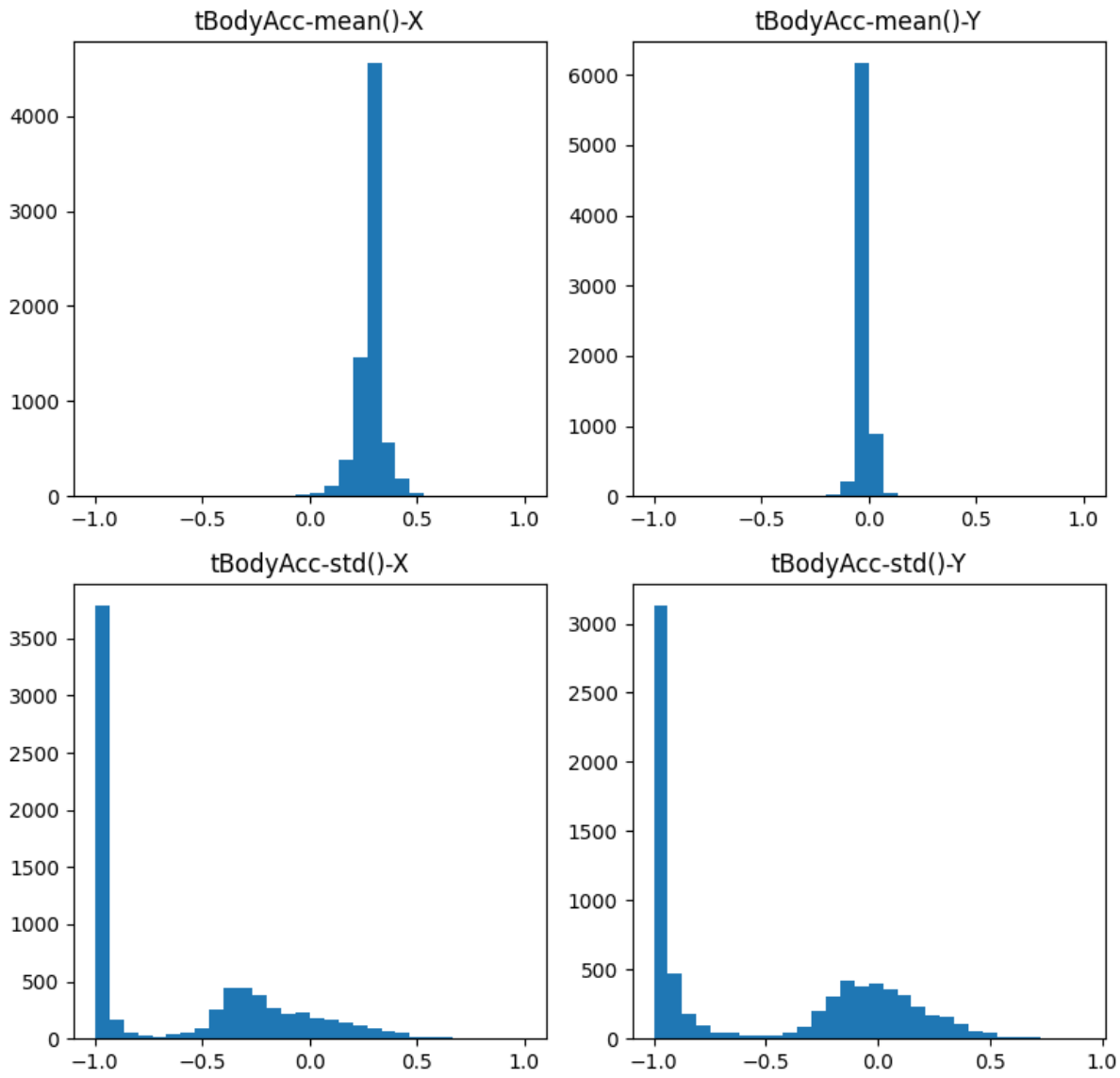
```
WALKING               1226
WALKING_UPSTAIRS      1073
WALKING_DOWNSTAIRS     986
Name: count, dtype: int64
```

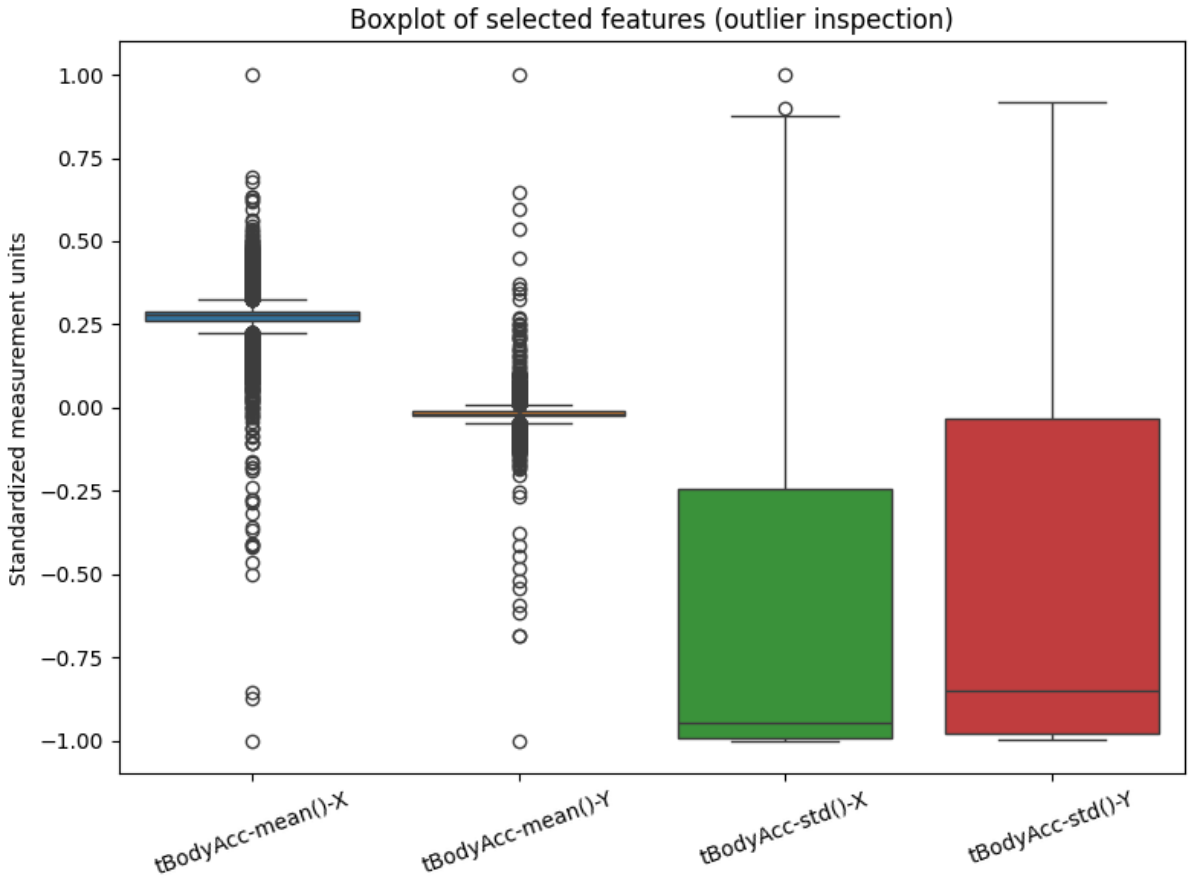



```
== Number of samples per subject ==
subject
1     347
3     341
5     302
6     325
7     308
8     281
11    316
14    323
15    328
16    366
Name: count, dtype: int64
```

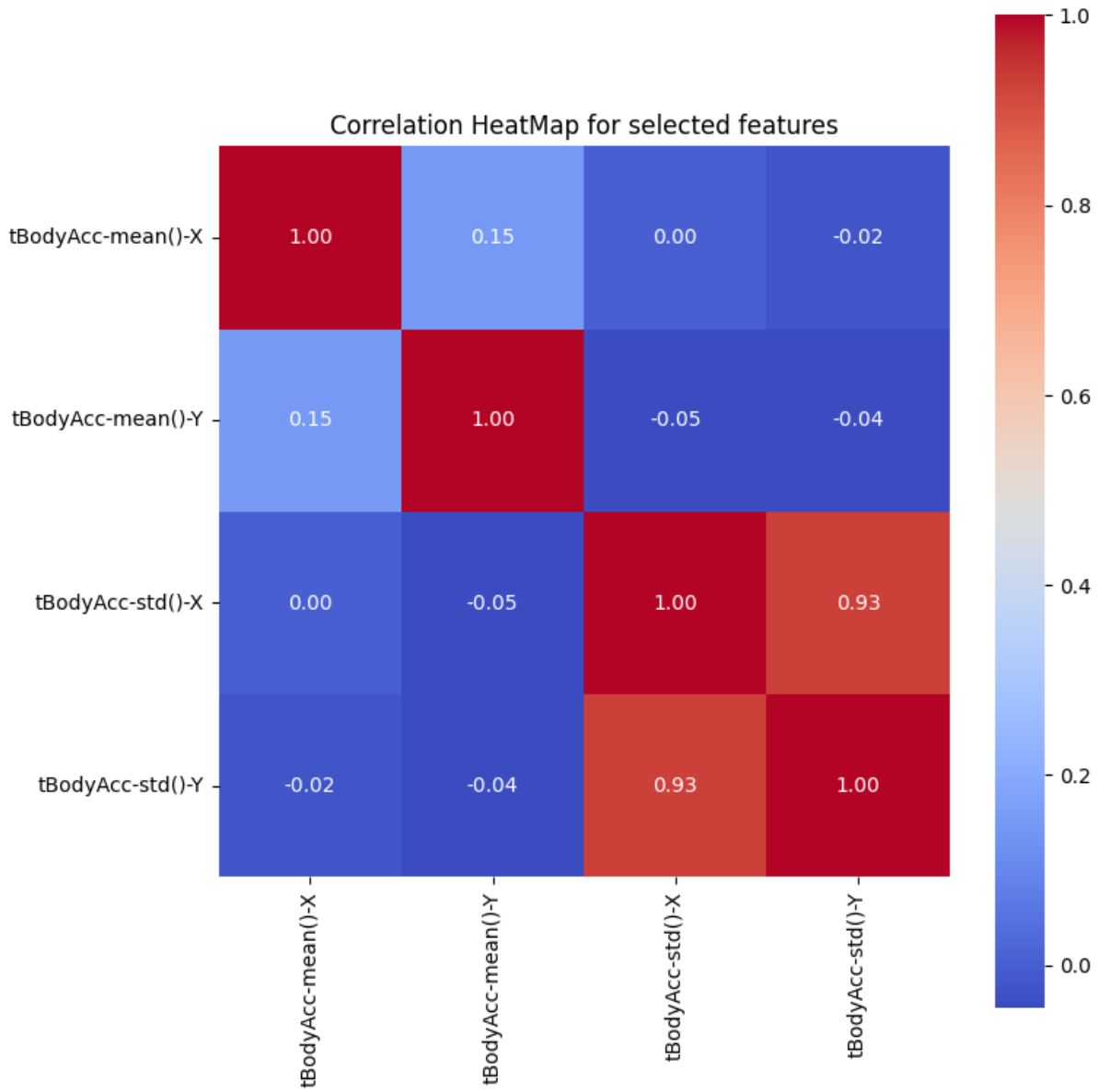



```
== Example Features to inspect ==
['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-std()-X', 'tBodyAc
c-std()-Y']
```


Histograms of selected features
tBodyAcc-mean()-X
tBodyAcc-mean()-Y
tBodyAcc-std()-X
tBodyAcc-std()-Y




```
== Correlation Matrix for the selected features==
                  tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-std()
-X  \
tBodyAcc-mean()-X           1.000000           0.148061          0.0006
19
tBodyAcc-mean()-Y           0.148061           1.000000         -0.0451
60
tBodyAcc-std()-X            0.000619          -0.045160          1.0000
00
tBodyAcc-std()-Y           -0.021903          -0.044920          0.9274
61

                  tBodyAcc-std()-Y
tBodyAcc-mean()-X         -0.021903
tBodyAcc-mean()-Y         -0.044920
tBodyAcc-std()-X           0.927461
tBodyAcc-std()-Y           1.000000
```



In [4]:

```
# ------------------------ 3. Prepare Features / Split -------------
# Drop ID-like column to avoid leakage
X = df.drop(columns=['Activity', 'subject'])
y = df['Activity']
num_features = X.columns.tolist()

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
print("\n== Train/Test Sizes==")
print(f"X_train: {X_train.shape}, y_train: {y_train.shape}")

# Preprocessing
scaler = ColumnTransformer(transformers=[("num", StandardScaler(), num

def scaled_pipeline(estimator):
    return Pipeline(steps=[("scaler", scaler), ("estimator", estimator
```

```
def passthrough_pipeline(estimator):
    return Pipeline(steps=[("clf", estimator)])
```

```
== Train/Test Sizes==
X_train: (5881, 561), y_train: (5881,)
```

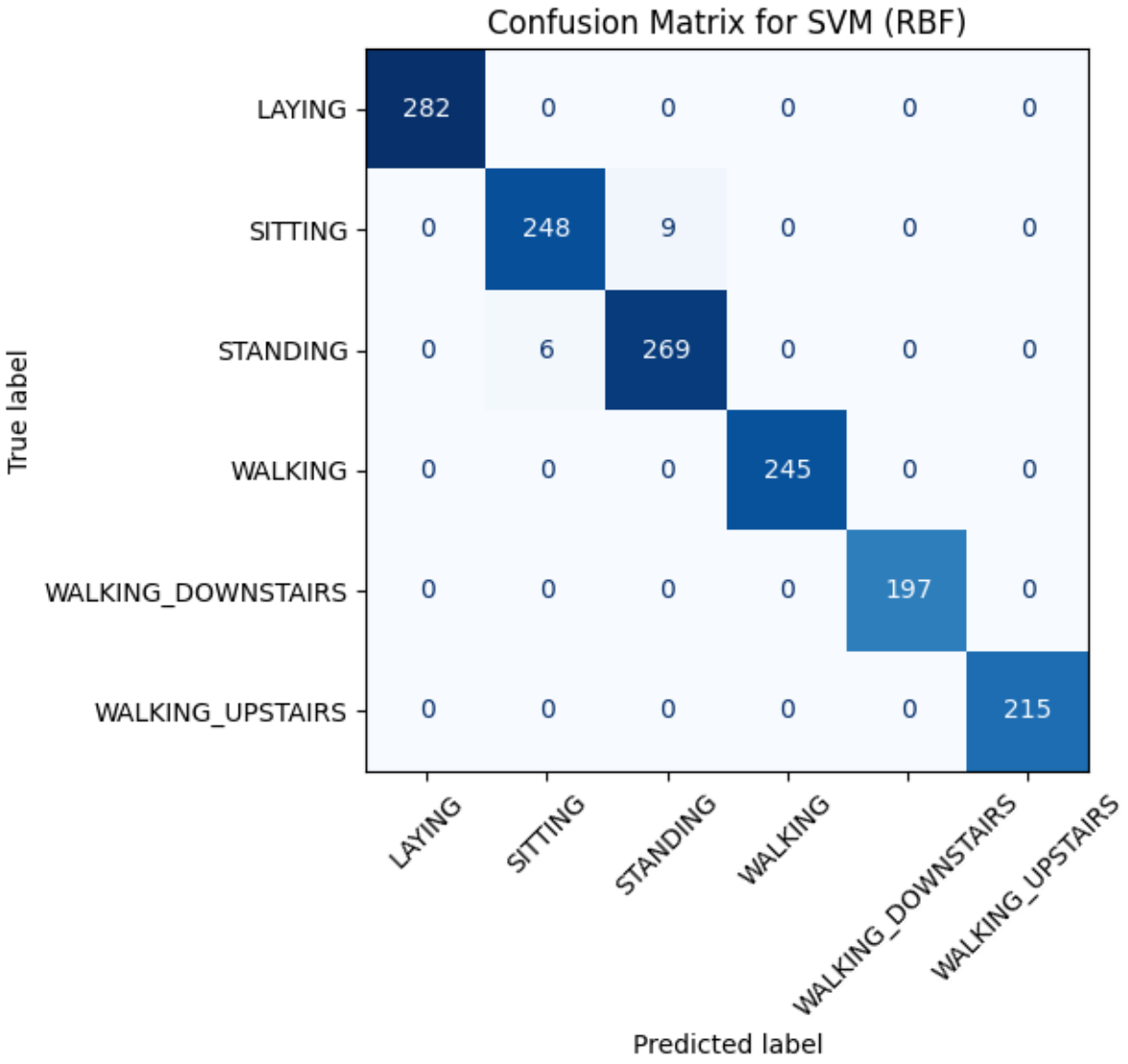
In [5]:
```
# ------------------------ 3.4 Models ------------------------
models = {
    "Perceptron": scaled_pipeline(Perceptron(random_state=42)),
    "RidgeClassifier (Adaline-like)": scaled_pipeline(RidgeClassifier(
    "Logistic Regression": scaled_pipeline(LogisticRegression(max_iter
    "SVM (RBF)": scaled_pipeline(SVC(kernel="rbf", C=10, gamma="scale"
    "KNN (k=15)": scaled_pipeline(KNeighborsClassifier(n_neighbors=15,
    "DecisionTree": passthrough_pipeline(DecisionTreeClassifier(max_de
    "RandomForest": passthrough_pipeline(RandomForestClassifier(n_esti
}
```

In [6]:
```
# 1. Importing all the important libraries
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, StratifiedKFold,
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

from sklearn.linear_model import Perceptron, RidgeClassifier, Logistic
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, VotingClassifier,
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_r

from sklearn.base import clone
from scipy.stats import loguniform, randint as sp_randint
from joblib import dump
```

In [7]:
```
# ------------------------ 3.5 Train & Evaluate --------------------
results = []
reports = {}

for name, pipe in models.items():
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    f1m = f1_score(y_test, y_pred, average="macro")
    results.append({"model": name, "accuracy": acc, "f1_macro": f1m})
```

```
    reports[name] = classification_report(y_test, y_pred, digits=3)
    print(f"\n== {name} ==")
    print(f"Accuracy: {acc:.4f} | f1_macro: {f1m:.4f}")

results_df = pd.DataFrame(results).sort_values(by=["accuracy", "f1_mac
print("\n == Model Leaderboard (Test Set) ==")
print(results_df.to_string(index=False))

best_name = results_df.iloc[0]["model"]
print(f"\n== Best Model: {best_name} ==")
print(reports[best_name])

best_pipe = models[best_name]
y_pred_best = best_pipe.predict(X_test)
fig, ax = plt.subplots(figsize=(8, 6))
ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred_best, xticks_rotation=45, cmap="Blues", colorbar=Fa
)
ax.set_title(f"Confusion Matrix for {best_name}")
fig.tight_layout()
plt.show()
```

```
== Perceptron ==
Accuracy: 0.9810 | f1_macro: 0.9818

== RidgeClassifier (Adaline-like) ==
Accuracy: 0.9810 | f1_macro: 0.9822

== Logistic Regression ==
Accuracy: 0.9857 | f1_macro: 0.9868

== SVM (RBF) ==
Accuracy: 0.9898 | f1_macro: 0.9906

== KNN (k=15) ==
Accuracy: 0.9565 | f1_macro: 0.9578

== DecisionTree ==
Accuracy: 0.9388 | f1_macro: 0.9380

== RandomForest ==
Accuracy: 0.9878 | f1_macro: 0.9879

 == Model Leaderboard (Test Set) ==
                         model  accuracy  f1_macro
                     SVM (RBF)  0.989803  0.990587
                  RandomForest  0.987763  0.987936
            Logistic Regression  0.985724  0.986825
RidgeClassifier (Adaline-like)  0.980965  0.982162
                    Perceptron  0.980965  0.981771
                    KNN (k=15)  0.956492  0.957820
                  DecisionTree  0.938817  0.937979

== Best Model: SVM (RBF) ==
                    precision    recall  f1-score   support

            LAYING      1.000     1.000     1.000       282
           SITTING      0.976     0.965     0.971       257
          STANDING      0.968     0.978     0.973       275
           WALKING      1.000     1.000     1.000       245
WALKING_DOWNSTAIRS      1.000     1.000     1.000       197
  WALKING_UPSTAIRS      1.000     1.000     1.000       215

          accuracy                          0.990      1471
         macro avg      0.991     0.991     0.991      1471
      weighted avg      0.990     0.990     0.990      1471
```



In [8]:

```
# ------------------------- 4. Hyperparameter Search -----------------
# ------- Runtime knobs -------
FAST_SEARCH = True   # True = faster hyperparam search; set False for

cv_splits = 3 if FAST_SEARCH else 5
cv = StratifiedKFold(n_splits=cv_splits, shuffle=True, random_state=42

def eval_and_log(name, estimator, Xtr=X_train, ytr=y_train, xte=X_test
    estimator = estimator.fit(Xtr, ytr)
    yhat = estimator.predict(xte)
    acc = accuracy_score(yte, yhat)
    f1m = f1_score(yte, yhat, average="macro")
    print(f"\n== {name} ==")
    print(f"Accuracy: {acc:.4f} | f1_macro: {f1m:.4f}")
    return {"model": name, "estimator": estimator, "accuracy": acc, "f

# Parameter spaces (target the inner estimator in each pipeline)
search_specs = {
    "SVM (RBF)": {
        "pipe": models["SVM (RBF)"],
```

```
        "params": {
            "estimator__C": loguniform(1e-1, 1e2),   # 0.1–100
            "estimator__gamma": ["scale", "auto"]
        },
        "n_iter": 15 if FAST_SEARCH else 25
    },
    "RandomForest": {
        "pipe": models["RandomForest"],
        "params": {
            "clf__n_estimators": sp_randint(150, 401),  # 150–400 tree
            "clf__max_depth": [None, 10, 20, 30],
            "clf__min_samples_split": sp_randint(2, 11),
            "clf__min_samples_leaf": sp_randint(1, 5),
            "clf__max_features": ["sqrt", "log2", None]
        },
        "n_iter": 15 if FAST_SEARCH else 30
    },
    "Logistic Regression": {
        "pipe": models["Logistic Regression"],
        "params": {
            "estimator__C": loguniform(1e-2, 1e2),
            "estimator__penalty": ["l2"],
            "estimator__solver": ["lbfgs", "saga"]
        },
        "n_iter": 15 if FAST_SEARCH else 25
    },
    "KNN (k=15)": {
        "pipe": models["KNN (k=15)"],
        "params": {
            "estimator__n_neighbors": sp_randint(5, 31),
            "estimator__weights": ["uniform", "distance"],
            "estimator__p": [1, 2]
        },
        "n_iter": 15 if FAST_SEARCH else 25
    },
}

print("\n == Hyperparameter search (RandomizedSearchCV) ==")
tuned_models = {}
for name, spec in search_specs.items():
    search = RandomizedSearchCV(
        estimator=spec["pipe"],
        param_distributions=spec["params"],
        n_iter=spec["n_iter"],
        cv=cv,
        scoring="f1_macro",
        random_state=42,
        n_jobs=-1,
        verbose=1,
        refit=True,
        error_score="raise"
    )
```

```
    search.fit(X_train, y_train)
    print(f"\n == Best params for {name}: {search.best_params_}")
    print(f"CV best f1_macro: {search.best_score_:.4f}")
    tuned_models[f"{name} (tuned)"] = search.best_estimator_

# Evaluate tuned models on the test set and extend the leaderboard
more_results = []
for nm, est in tuned_models.items():
    more_results.append(eval_and_log(nm, est))

if more_results:
    more_df = pd.DataFrame([{k: v for k, v in r.items() if k != "estim
    results_df = (
        pd.concat([results_df, more_df[["model", "accuracy", "f1_macro
          .sort_values(by=["accuracy", "f1_macro"], ascending=False)
    )
    print("\n == Updated Leaderboard (incl. tuned) ==")
    print(results_df.to_string(index=False))
```

```
 == Hyperparameter search (RandomizedSearchCV) ==
Fitting 3 folds for each of 15 candidates, totalling 45 fits

 == Best params for SVM (RBF): {'estimator__C': np.float64(31.428808908
401084), 'estimator__gamma': 'auto'}
CV best f1_macro: 0.9830
Fitting 3 folds for each of 15 candidates, totalling 45 fits

 == Best params for RandomForest: {'clf__max_depth': 30, 'clf__max_feat
ures': 'log2', 'clf__min_samples_leaf': 2, 'clf__min_samples_split': 3,
'clf__n_estimators': 341}
CV best f1_macro: 0.9757
Fitting 3 folds for each of 15 candidates, totalling 45 fits

 == Best params for Logistic Regression: {'estimator__C': np.float64(0.
5342937261279778), 'estimator__penalty': 'l2', 'estimator__solver': 'lb
fgs'}
CV best f1_macro: 0.9812
Fitting 3 folds for each of 15 candidates, totalling 45 fits

 == Best params for KNN (k=15): {'estimator__n_neighbors': 16, 'estimat
or__p': 1, 'estimator__weights': 'uniform'}
CV best f1_macro: 0.9626

== SVM (RBF) (tuned) ==
Accuracy: 0.9898 | f1_macro: 0.9906

== RandomForest (tuned) ==
Accuracy: 0.9850 | f1_macro: 0.9853

== Logistic Regression (tuned) ==
Accuracy: 0.9857 | f1_macro: 0.9868

== KNN (k=15) (tuned) ==
Accuracy: 0.9708 | f1_macro: 0.9721

 == Updated Leaderboard (incl. tuned) ==
                          model  accuracy  f1_macro
             SVM (RBF) (tuned)  0.989803  0.990592
                     SVM (RBF)  0.989803  0.990587
                  RandomForest  0.987763  0.987936
           Logistic Regression  0.985724  0.986825
   Logistic Regression (tuned)  0.985724  0.986825
          RandomForest (tuned)  0.985044  0.985288
RidgeClassifier (Adaline-like)  0.980965  0.982162
                    Perceptron  0.980965  0.981771
            KNN (k=15) (tuned)  0.970768  0.972094
                    KNN (k=15)  0.956492  0.957820
                  DecisionTree  0.938817  0.937979
```

In [9]:

```python
# ------------------------- 5. True Ensembles: Voting + Stacking -----
def ensure_svc_probability(pipe):
    p = clone(pipe)
```

```
    if "estimator" in p.named_steps and isinstance(p.named_steps["esti
        if not p.named_steps["estimator"].probability:
            p.set_params(estimator__probability=True)
    return p

svm_base = tuned_models.get("SVM (RBF) (tuned)", models["SVM (RBF)"])
rf_base  = tuned_models.get("RandomForest (tuned)", models["RandomFore
lr_base  = tuned_models.get("Logistic Regression (tuned)", models["Log
knn_base = tuned_models.get("KNN (k=15) (tuned)", models["KNN (k=15)"]

svm_soft = ensure_svc_probability(svm_base)

voting_hard = VotingClassifier(
    estimators=[
        ("perc", models["Perceptron"]),
        ("ridge", models["RidgeClassifier (Adaline-like)"]),
        ("lr",   lr_base),
        ("svm",  svm_base),
        ("dt",   models["DecisionTree"]),
        ("rf",   rf_base),
        ("knn",  knn_base),
    ],
    voting="hard",
    n_jobs=-1
)

voting_soft = VotingClassifier(
    estimators=[
        ("lr",  lr_base),
        ("svm", svm_soft),
        ("rf",  rf_base),
        ("knn", knn_base),
    ],
    voting="soft",
    n_jobs=-1
)

stacking = StackingClassifier(
    estimators=[
        ("lr",  lr_base),
        ("svm", svm_soft),
        ("rf",  rf_base),
        ("knn", knn_base),
    ],
    final_estimator=LogisticRegression(max_iter=5000, random_state=42)
    stack_method="predict_proba",
    cv=cv,
    n_jobs=-1,
    passthrough=False
)

ensemble_models = {
```

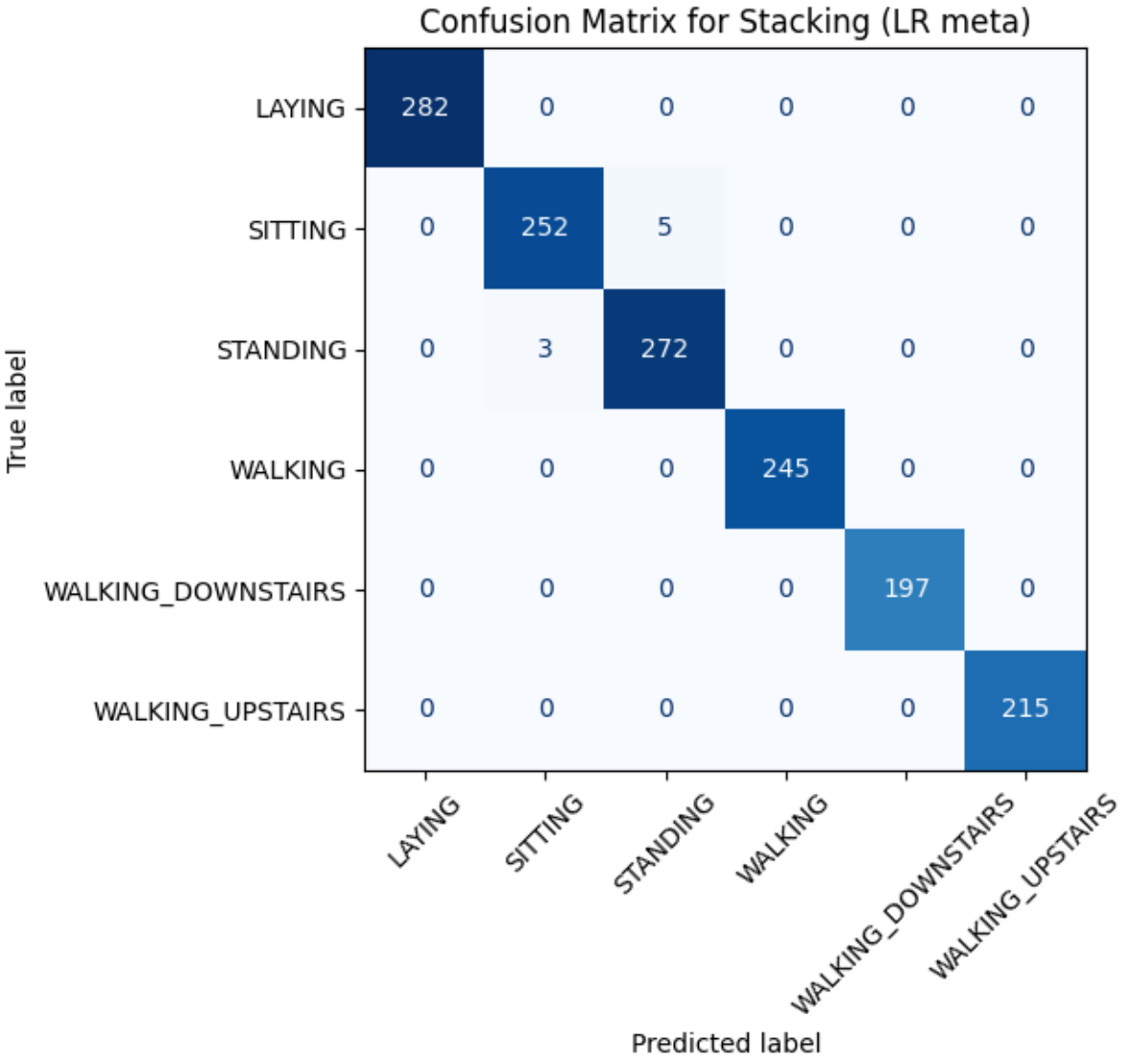
```
    "Voting (hard)": voting_hard,
    "Voting (soft)": voting_soft,
    "Stacking (LR meta)": stacking,
}

print("\n== Ensembles ==")
ensemble_results = []
for nm, est in ensemble_models.items():
    ensemble_results.append(eval_and_log(nm, est))
```

```
== Ensembles ==

== Voting (hard) ==
Accuracy: 0.9905 | f1_macro: 0.9912

== Voting (soft) ==
Accuracy: 0.9918 | f1_macro: 0.9925

== Stacking (LR meta) ==
Accuracy: 0.9946 | f1_macro: 0.9950
```

In [10]:

```
# ------------------------ 6. Consolidate and choose global best ----
all_results = []
for r in results:
    all_results.append({"model": r["model"], "accuracy": r["accuracy"]
for r in more_results:
    all_results.append({"model": r["model"], "accuracy": r["accuracy"]
for r in ensemble_results:
    all_results.append({"model": r["model"], "accuracy": r["accuracy"]

all_df = pd.DataFrame(all_results).sort_values(by=["accuracy", "f1_mac
print("\n == GRAND LEADERBOARD ==")
print(all_df.to_string(index=False))

best_overall_name = all_df.iloc[0]["model"]
print(f"\n== Best Overall: {best_overall_name} ==")

name_to_est = {**{k: v for k, v in models.items()}, **tuned_models, **
best_overall_est = name_to_est[best_overall_name]
best_overall_est.fit(X_train, y_train)

fig, ax = plt.subplots(figsize=(8, 6))
ConfusionMatrixDisplay.from_predictions(
    y_test, best_overall_est.predict(X_test), xticks_rotation=45, cmap
)
ax.set_title(f"Confusion Matrix for {best_overall_name}")
fig.tight_layout()
plt.show()
```

```
== GRAND LEADERBOARD ==
                          model  accuracy  f1_macro
            Stacking (LR meta)  0.994562  0.994980
                 Voting (soft)  0.991842  0.992473
                 Voting (hard)  0.990483  0.991216
              SVM (RBF) (tuned)  0.989803  0.990592
                       SVM (RBF)  0.989803  0.990587
                   RandomForest  0.987763  0.987936
             Logistic Regression  0.985724  0.986825
    Logistic Regression (tuned)  0.985724  0.986825
           RandomForest (tuned)  0.985044  0.985288
RidgeClassifier (Adaline-like)  0.980965  0.982162
                     Perceptron  0.980965  0.981771
              KNN (k=15) (tuned)  0.970768  0.972094
                     KNN (k=15)  0.956492  0.957820
                   DecisionTree  0.938817  0.937979

== Best Overall: Stacking (LR meta) ==
```



In [11]:
```
# ------------------------- 7. Persist the final pipeline & inference
MODEL_PATH = "best_har_model.joblib"
```

```
FEATURE_LIST_PATH = "feature_names.txt"

dump(best_overall_est, MODEL_PATH)
with open(FEATURE_LIST_PATH, "w") as f:
    for c in num_features:
        f.write(c + "\n")

print(f"\nSaved model to: {MODEL_PATH}")
print(f"Saved feature list to: {FEATURE_LIST_PATH}")

def predict_activities(new_data, model_path=MODEL_PATH, feature_list_p
    """
    new_data: pandas DataFrame (same feature space) OR path to CSV wit
    Returns: numpy array of predicted activity labels.
    """
    from joblib import load
    if isinstance(new_data, str):
        new_df = pd.read_csv(new_data)
    else:
        new_df = new_data.copy()

    with open(feature_list_path, "r") as f:
        feats = [ln.strip() for ln in f if ln.strip()]

    missing = [c for c in feats if c not in new_df.columns]
    if missing:
        raise ValueError(f"Missing required columns: {missing[:10]}{'.

    new_df = new_df.reindex(columns=feats)
    model = load(model_path)
    return model.predict(new_df)
```

```
Saved model to: best_har_model.joblib
Saved feature list to: feature_names.txt
```

In [12]:

```
# Bagging & Boosting add-ons (separate from your existing `models`)
from sklearn.ensemble import (
    BaggingClassifier, ExtraTreesClassifier, AdaBoostClassifier, HistG
)
from sklearn.tree import DecisionTreeClassifier

# A separate dict so you don't have to touch your original `models`
bag_boost_models = {
    # Bagging family
    "Bagging (DT)": passthrough_pipeline(
        BaggingClassifier(
            estimator=DecisionTreeClassifier(random_state=42),
            n_estimators=200, bootstrap=True, n_jobs=-1, random_state=
        )
    ),
    "ExtraTrees": passthrough_pipeline(
        ExtraTreesClassifier(n_estimators=400, n_jobs=-1, random_state
    ),
```

```
    # Boosting family
    "AdaBoost": passthrough_pipeline(
        AdaBoostClassifier(
            estimator=DecisionTreeClassifier(max_depth=2, random_state
            n_estimators=300, learning_rate=0.5, random_state=42
        )
    ),
    "HistGradientBoosting": passthrough_pipeline(
        HistGradientBoostingClassifier(
            max_iter=300, learning_rate=0.1, random_state=42
        )
    ),
}
```

In [13]:

```
bb_results = []
bb_reports = {}

for name, pipe in bag_boost_models.items():
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    f1m = f1_score(y_test, y_pred, average="macro")
    bb_results.append({"model": name, "accuracy": acc, "f1_macro": f1m
    bb_reports[name] = classification_report(y_test, y_pred, digits=3)
    print(f"\n== {name} (baseline) ==")
    print(f"Accuracy: {acc:.4f} | f1_macro: {f1m:.4f}")

bb_baseline_df = pd.DataFrame(bb_results).sort_values(
    by=["accuracy","f1_macro"], ascending=False
)
print("\n== Bagging/Boosting Baseline Leaderboard ==")
print(bb_baseline_df.to_string(index=False))

# If you already have results_df from earlier, show a merged view for
if 'results_df' in globals():
    merged = pd.concat([results_df, bb_baseline_df], ignore_index=True
              .sort_values(by=["accuracy","f1_macro"], ascending=False
    print("\n== Combined Leaderboard (incl. new baselines) ==")
    print(merged.to_string(index=False))
```

```
== Bagging (DT) (baseline) ==
Accuracy: 0.9721 | f1_macro: 0.9722

== ExtraTrees (baseline) ==
Accuracy: 0.9932 | f1_macro: 0.9937

== AdaBoost (baseline) ==
Accuracy: 0.9864 | f1_macro: 0.9869

== HistGradientBoosting (baseline) ==
Accuracy: 0.9939 | f1_macro: 0.9941

== Bagging/Boosting Baseline Leaderboard ==
                model  accuracy  f1_macro
HistGradientBoosting  0.993882  0.994067
          ExtraTrees  0.993202  0.993722
            AdaBoost  0.986404  0.986879
        Bagging (DT)  0.972128  0.972211

== Combined Leaderboard (incl. new baselines) ==
                         model  accuracy  f1_macro
          HistGradientBoosting  0.993882  0.994067
                     ExtraTrees  0.993202  0.993722
            SVM (RBF) (tuned)  0.989803  0.990592
                     SVM (RBF)  0.989803  0.990587
                  RandomForest  0.987763  0.987936
                      AdaBoost  0.986404  0.986879
           Logistic Regression  0.985724  0.986825
   Logistic Regression (tuned)  0.985724  0.986825
          RandomForest (tuned)  0.985044  0.985288
RidgeClassifier (Adaline-like)  0.980965  0.982162
                    Perceptron  0.980965  0.981771
                  Bagging (DT)  0.972128  0.972211
           KNN (k=15) (tuned)  0.970768  0.972094
                     KNN (k=15)  0.956492  0.957820
                  DecisionTree  0.938817  0.937979
```

In [14]:

```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform, randint as sp_randint

# Smaller n_iter to keep runtime reasonable; increase later if you wan
QUICK = True
iters = (12 if QUICK else 30)

bb_search_specs = {
    "ExtraTrees": {
        "pipe": bag_boost_models["ExtraTrees"],
        "params": {
            "clf__n_estimators": sp_randint(200, 601),
            "clf__max_depth": [None, 10, 20, 30],
            "clf__max_features": ["sqrt", "log2", None],
            "clf__min_samples_split": sp_randint(2, 11),
            "clf__min_samples_leaf": sp_randint(1, 5),
```

```
        },
        "n_iter": iters
    },
    "Bagging (DT)": {
        "pipe": bag_boost_models["Bagging (DT)"],
        "params": {
            "clf__n_estimators": sp_randint(100, 401),
            "clf__bootstrap": [True, False],
            "clf__estimator__max_depth": [None, 5, 10, 20],
            "clf__estimator__min_samples_split": sp_randint(2, 11),
            "clf__estimator__min_samples_leaf": sp_randint(1, 5),
        },
        "n_iter": iters
    },
    "AdaBoost": {
        "pipe": bag_boost_models["AdaBoost"],
        "params": {
            "clf__n_estimators": sp_randint(100, 601),
            "clf__learning_rate": loguniform(1e-2, 1.0),
            "clf__estimator__max_depth": [1, 2, 3],
        },
        "n_iter": iters
    },
    "HistGradientBoosting": {
        "pipe": bag_boost_models["HistGradientBoosting"],
        "params": {
            "clf__learning_rate": loguniform(1e-2, 3e-1),
            "clf__max_iter": sp_randint(150, 401),
            "clf__max_leaf_nodes": sp_randint(15, 65),
            "clf__max_depth": [None, 6, 8, 10],
            "clf__min_samples_leaf": sp_randint(10, 60),
            "clf__l2_regularization": loguniform(1e-4, 1.0),
        },
        "n_iter": (iters + 6 if QUICK else 40)
    },
}

bb_tuned_models = {}
for name, spec in bb_search_specs.items():
    print(f"\n== RandomizedSearchCV for {name} ==")
    search = RandomizedSearchCV(
        estimator=spec["pipe"],
        param_distributions=spec["params"],
        n_iter=spec["n_iter"],
        cv=cv,
        scoring="f1_macro",
        random_state=42,
        n_jobs=-1,
        verbose=1,
        refit=True,
        error_score="raise"
    )
```

```
    search.fit(X_train, y_train)
    print(f"Best params for {name}: {search.best_params_}")
    print(f"CV best f1_macro: {search.best_score_:.4f}")
    bb_tuned_models[f"{name} (tuned)"] = search.best_estimator_

# Evaluate tuned ones
bb_more_results = [eval_and_log(nm, est) for nm, est in bb_tuned_model
bb_more_df = pd.DataFrame(
    [{k: v for k, v in r.items() if k != "estimator"} for r in bb_more
).sort_values(by=["accuracy", "f1_macro"], ascending=False)

print("\n== Bagging/Boosting Tuned Leaderboard ==")
print(bb_more_df.to_string(index=False))

# Optional: merge with previous overall leaderboard if available
if 'results_df' in globals():
    combined_df = pd.concat([results_df, bb_baseline_df, bb_more_df],
                    .sort_values(by=["accuracy","f1_macro"], ascending
    print("\n== Updated GRAND LEADERBOARD (incl. bagging/boosting) =="
    print(combined_df.to_string(index=False))
```

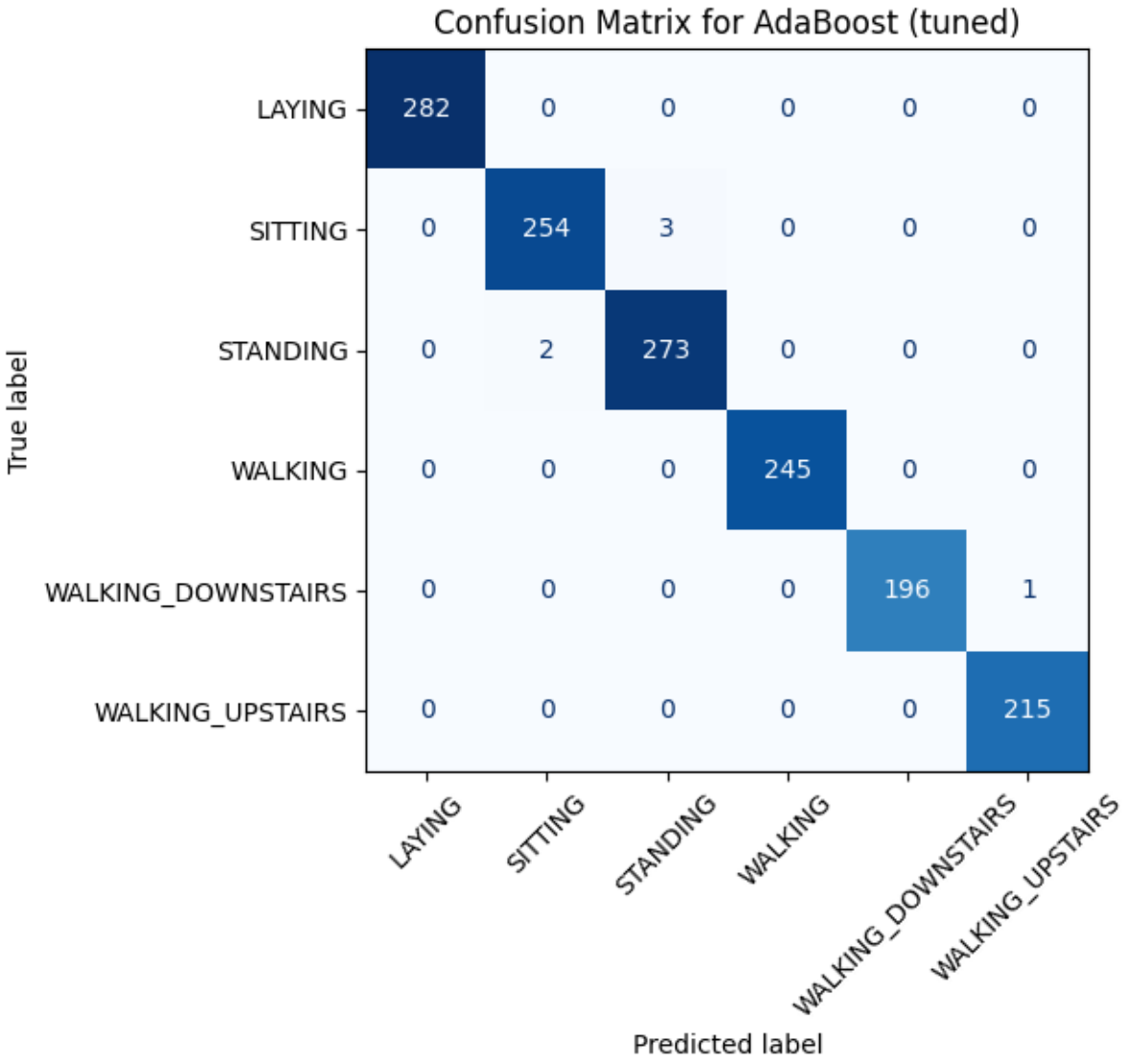
```
== RandomizedSearchCV for ExtraTrees ==
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best params for ExtraTrees: {'clf__max_depth': 30, 'clf__max_features':
'sqrt', 'clf__min_samples_leaf': 1, 'clf__min_samples_split': 4, 'clf__
n_estimators': 563}
CV best f1_macro: 0.9824

== RandomizedSearchCV for Bagging (DT) ==
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best params for Bagging (DT): {'clf__bootstrap': True, 'clf__estimator_
_max_depth': None, 'clf__estimator__min_samples_leaf': 3, 'clf__estimat
or__min_samples_split': 3, 'clf__n_estimators': 188}
CV best f1_macro: 0.9626

== RandomizedSearchCV for AdaBoost ==
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best params for AdaBoost: {'clf__estimator__max_depth': 3, 'clf__learni
ng_rate': np.float64(0.39181943471417396), 'clf__n_estimators': 370}
CV best f1_macro: 0.9910

== RandomizedSearchCV for HistGradientBoosting ==
Fitting 3 folds for each of 18 candidates, totalling 54 fits
Best params for HistGradientBoosting: {'clf__l2_regularization': np.flo
at64(0.12604664585649453), 'clf__learning_rate': np.float64(0.244204608
44911424), 'clf__max_depth': 6, 'clf__max_iter': 377, 'clf__max_leaf_no
des': 28, 'clf__min_samples_leaf': 40}
CV best f1_macro: 0.9917

== ExtraTrees (tuned) ==
Accuracy: 0.9925 | f1_macro: 0.9931

== Bagging (DT) (tuned) ==
```

```
Accuracy: 0.9714 | f1_macro: 0.9712

== AdaBoost (tuned) ==
Accuracy: 0.9959 | f1_macro: 0.9961

== HistGradientBoosting (tuned) ==
Accuracy: 0.9925 | f1_macro: 0.9929

== Bagging/Boosting Tuned Leaderboard ==
                        model  accuracy  f1_macro
           AdaBoost (tuned)  0.995921  0.996052
         ExtraTrees (tuned)  0.992522  0.993093
HistGradientBoosting (tuned)  0.992522  0.992915
       Bagging (DT) (tuned)  0.971448  0.971197

== Updated GRAND LEADERBOARD (incl. bagging/boosting) ==
                          model  accuracy  f1_macro
              AdaBoost (tuned)  0.995921  0.996052
         HistGradientBoosting  0.993882  0.994067
                    ExtraTrees  0.993202  0.993722
           ExtraTrees (tuned)  0.992522  0.993093
  HistGradientBoosting (tuned)  0.992522  0.992915
           SVM (RBF) (tuned)  0.989803  0.990592
                     SVM (RBF)  0.989803  0.990587
                  RandomForest  0.987763  0.987936
                      AdaBoost  0.986404  0.986879
           Logistic Regression  0.985724  0.986825
   Logistic Regression (tuned)  0.985724  0.986825
         RandomForest (tuned)  0.985044  0.985288
RidgeClassifier (Adaline-like)  0.980965  0.982162
                    Perceptron  0.980965  0.981771
                  Bagging (DT)  0.972128  0.972211
         Bagging (DT) (tuned)  0.971448  0.971197
           KNN (k=15) (tuned)  0.970768  0.972094
                     KNN (k=15)  0.956492  0.957820
                  DecisionTree  0.938817  0.937979
```

In [15]:

```python
# Pick best among the new/tuned models and show confusion
if 'bb_more_df' in globals() and not bb_more_df.empty:
    best_bb_name = bb_more_df.iloc[0]["model"]
    print(f"\n== Best Bagging/Boosting Model: {best_bb_name} ==")
    best_bb_est = {**bag_boost_models, **bb_tuned_models}[best_bb_name
    from sklearn.metrics import ConfusionMatrixDisplay
    fig, ax = plt.subplots(figsize=(8,6))
    ConfusionMatrixDisplay.from_predictions(
        y_test, best_bb_est.predict(X_test),
        xticks_rotation=45, cmap="Blues", colorbar=False, ax=ax
    )
    ax.set_title(f"Confusion Matrix for {best_bb_name}")
    fig.tight_layout()
    plt.show()
```

```
== Best Bagging/Boosting Model: AdaBoost (tuned) ==
```


Confusion Matrix for AdaBoost (tuned)
True label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
282 0 0 0 0 0
0 254 3 0 0 0
0 2 273 0 0 0
0 0 0 245 0 0
0 0 0 0 196 1
0 0 0 0 0 215
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
Predicted label