INPUT 1

```
# Importing basic python built-in libraries
import numpy as np
import pandas as pd
```

INPUT 2

```
# Obtain the train and test data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
print(train.shape, test.shape)
```

OUTPUT 2

```
(7352, 564) (2947, 564)
```

INPUT 3

```
train.head(3)
```

OUTPUT 3

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	...	subject	Activity	ActivityName
0	0.288585	-0.020294	-0.132905	...	1	5	STANDING
1	0.278419	-0.016411	-0.123520	...	1	5	STANDING
2	0.279653	-0.019467	-0.113462	...	1	5	STANDING

[3 rows x 564 columns]

INPUT 4

```
# get X_train and y_train from csv files
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train.ActivityName

# get X_test and y_test from test csv file
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test.ActivityName

print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
```

OUTPUT 4

```
X_train and y_train : ((7352, 561),(7352,))
```

INPUT 5

```
print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
```

OUTPUT 5

```
X_test  and y_test  : ((2947, 561),(2947,))
```

INPUT 6

```
# Labels that are useful in plotting confusion matrix
labels=['LAYING',
        'SITTING',
        'STANDING',
        'WALKING',
        'WALKING_DOWNSTAIRS',
        'WALKING_UPSTAIRS']

# Function to plot the confusion matrix
import itertools
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import sklearn.metrics as metrics

plt.rcParams["font.family"] = 'DejaVu Sans'


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

INPUT 7

```
# Generic function to run any model specified
from datetime import datetime
def perform_model(model, X_train, y_train, X_test, y_test, class_labels,
cm_normalize=True, \
                 print_cm=True, cm_cmap=plt.cm.Greens):

    # to store results at various phases
    results = dict()

    # time at which model starts training
    train_start_time = datetime.now()
    print('training the model..')
    model.fit(X_train, y_train)
    print('Done \n \n')
    train_end_time = datetime.now()
    results['training_time'] =  train_end_time - train_start_time
    print('training_time(HH:MM:SS.ms) - {}\n\n'.format(results['training_time']))


    # predict test data
    print('Predicting test data')
    test_start_time = datetime.now()
    y_pred = model.predict(X_test)
    test_end_time = datetime.now()
    print('Done \n \n')
    results['testing_time'] = test_end_time - test_start_time
    print('testing time(HH:MM:SS:ms) - {}\n\n'.format(results['testing_time']))
    results['predicted'] = y_pred


    # calculate overall accuracty of the model
    accuracy = metrics.accuracy_score(y_true=y_test, y_pred=y_pred)
    # store accuracy in results
    results['accuracy'] = accuracy
    print('---------------------')
    print('|      Accuracy      |')
    print('---------------------')
    print('\n    {}\n\n'.format(accuracy))


    # confusion matrix
    cm = metrics.confusion_matrix(y_test, y_pred)
    results['confusion_matrix'] = cm
    if print_cm:
        print('--------------------')
        print('| Confusion Matrix |')
        print('--------------------')
        print('\n {}'.format(cm))

    # plot confusin matrix
    plt.figure(figsize=(5.3,5.3))
    plt.grid(b=False)
    plot_confusion_matrix(cm, classes=class_labels, normalize=True,
title='Normalized confusion    matrix', cmap = cm_cmap)
    plt.show()

    # get classification report
    print('\n\n')
    print('-------------------------')
    print('| Classification Report |')
    print('-------------------------')
    classification_report = metrics.classification_report(y_test, y_pred)
    # store report in results
    results['classification_report'] = classification_report
    print(classification_report)

    # add the trained  model to the results
    results['model'] = model

    return results
```

INPUT 8

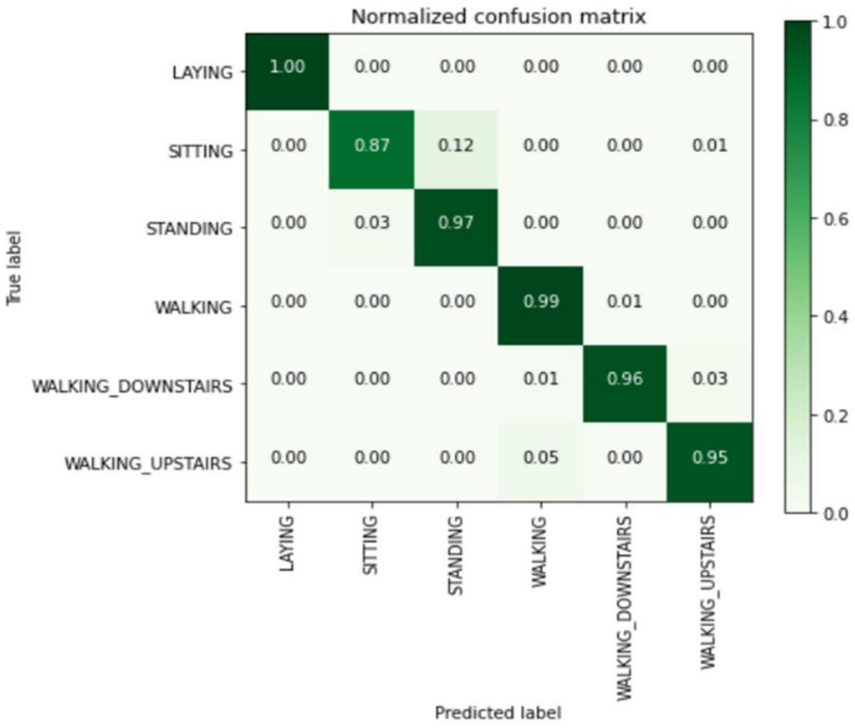
```
# Method to print the gridsearch Attributes
def print_grid_search_attributes(model):

    # Estimator that gave highest score among all the estimators formed in
    GridSearch
    print('--------------------------')
    print('|      Best Estimator     |')
    print('--------------------------')
    print('\n\t{}\n'.format(model.best_estimator_))

    # parameters that gave best results while performing grid search
    print('--------------------------')
    print('|     Best parameters     |')
    print('--------------------------')
    print('\tParameters of best estimator : \n\n\t{}\n'.format(model.best_params_))

    #  number of cross validation splits
    print('---------------------------------')
    print('|   No of CrossValidation sets   |')
    print('--------------------------------')
    print('\n\tTotal number of cross validation sets: {}\n'.format(model.n_splits_))

# Average cross validated score of the best estimator, from the Grid Search
    print('--------------------------')
    print('|        Best Score       |')
    print('--------------------------')
    print('\n\tAverage Cross Validate scores of best estimator :
\n\n\t{}\n'.format(model.best_score_))
```

INPUT 9

```
# importing sklearn for machine learning algorithms
from sklearn import metrics
from sklearn.model_selection import GridSearchCV

# LOGISTIC REGRESSION (ALGORITHM 1)
from sklearn import linear_model

# start Grid search
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
log_reg = linear_model.LogisticRegression()
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1,
n_jobs=-1)
log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X_test,
y_test, class_labels=labels)
```

OUTPUT 9

```
training the model..
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Done


training_time(HH:MM:SS.ms) - 0:00:29.117835


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.006376


---------------------
|      Accuracy      |
---------------------

    0.9579233118425518


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 428  60   0   0   3]
 [  0  16 516   0   0   0]
 [  0   0   0 492   3   1]
 [  0   0   0   4 403  13]
 [  0   0   0  23   1 447]]
```




```
-------------------------
| Classification Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.96      0.87      0.92       491
          STANDING       0.90      0.97      0.93       532
           WALKING       0.95      0.99      0.97       496
WALKING_DOWNSTAIRS       0.99      0.96      0.97       420
  WALKING_UPSTAIRS       0.96      0.95      0.96       471

          accuracy                           0.96      2947
         macro avg       0.96      0.96      0.96      2947
      weighted avg       0.96      0.96      0.96      2947
```

INPUT 10

```
# plotting confusion matrix
plt.figure(figsize=(5.3,5.3))
```

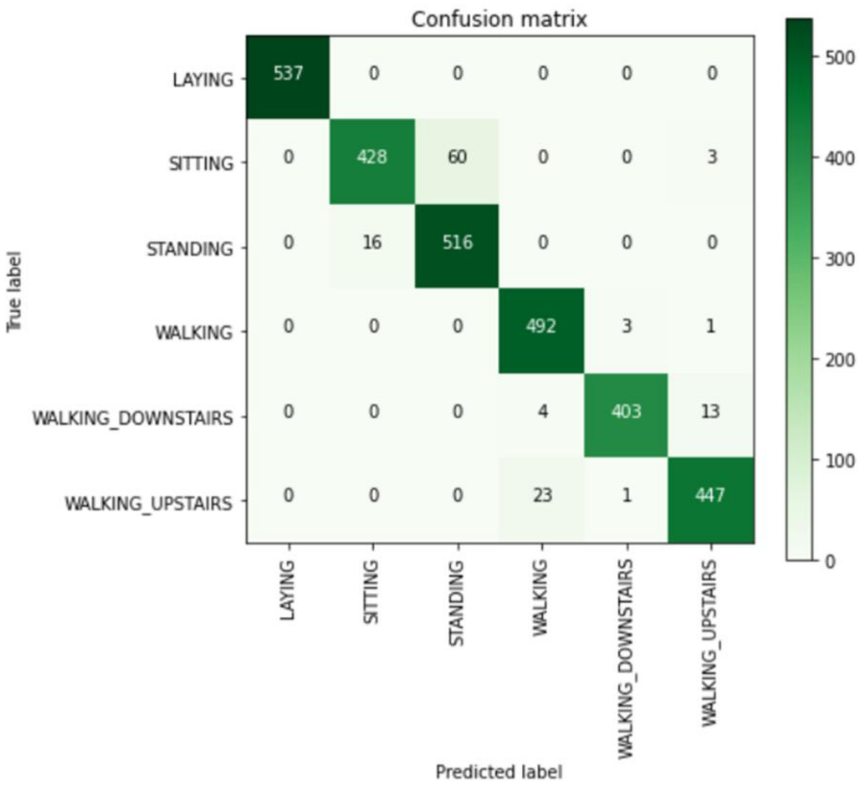
OUTPUT 10

```
<Figure size 530x530 with 0 Axes>
```

INPUT 11

```
plt.grid(b=False)
plot_confusion_matrix(log_reg_grid_results['confusion_matrix'], classes=labels, cmap=plt.cm.Greens, )
plt.show()
```

OUTPUT 11



INPUT 12

```
# observe the attributes of the model
print_grid_search_attributes(log_reg_grid_results['model'])
```

OUTPUT 12

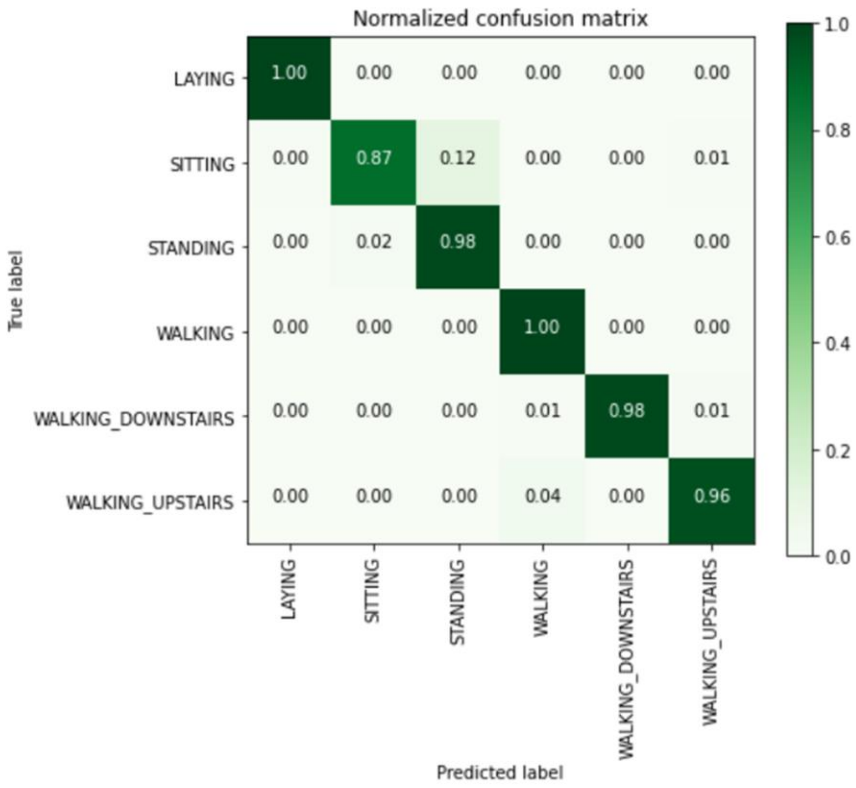
```
--------------------------
|      Best Estimator     |
--------------------------


          LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                intercept_scaling=1, l1_ratio=None, max_iter=100,
                multi_class='auto', n_jobs=None, penalty='l2',
                random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
          Parameters of best estimator :

          {'C': 1, 'penalty': 'l2'}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

          Total number of cross validation sets: 3

--------------------------
|       Best Score        |
--------------------------

          Average Cross Validate scores of best estimator :

          0.9374335617559958
```

INPUT 13

```
# SUPPORT VECTOR MACHINE (ALGORITHM 2)
from sklearn.svm import LinearSVC

# start Grid search
parameters = {'C':[0.125, 0.5, 1, 2, 8, 16]}
lr_svc = LinearSVC(tol=0.00005)
lr_svc_grid = GridSearchCV(lr_svc, param_grid=parameters, n_jobs=-1, verbose=1)
lr_svc_grid_results = perform_model(lr_svc_grid, X_train, y_train, X_test, y_test,
class_labels=labels)
```

OUTPUT 13

```
training the model..
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Done


training_time(HH:MM:SS.ms) - 0:01:37.599530


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.006488


---------------------
|      Accuracy      |
---------------------

    0.9664065151001018


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  2 427  59   0   0   3]
 [  0   9 522   1   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0   3 412   5]
 [  0   0   0  17   0 454]]
```




```
-------------------------
| Classification Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.98      0.87      0.92       491
          STANDING       0.90      0.98      0.94       532
           WALKING       0.96      1.00      0.98       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.96      0.97       471

          accuracy                           0.97      2947
         macro avg       0.97      0.97      0.97      2947
      weighted avg       0.97      0.97      0.97      2947
```

INPUT 14

```
# plotting confusion matrix
plt.figure(figsize=(5.3,5.3))
```

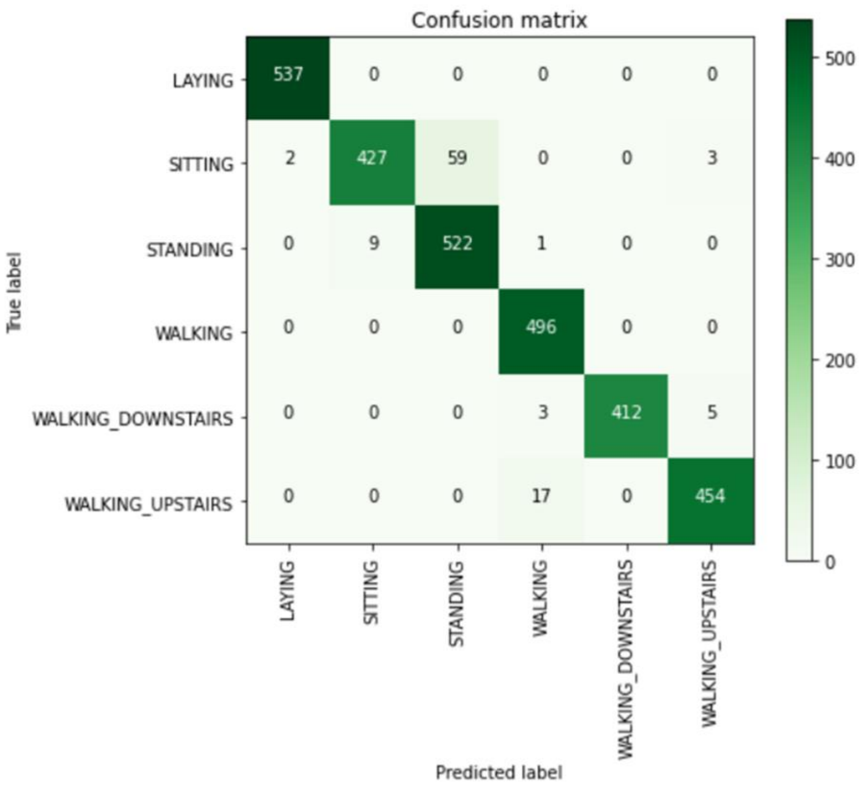
OUTPUT 14

```
<Figure size 530x530 with 0 Axes>
```

INPUT 15

```
plt.grid(b=False)
plot_confusion_matrix(lr_svc_grid_results['confusion_matrix'], classes=labels,
cmap=plt.cm.Greens, )
plt.show()
```

OUTPUT 15



INPUT 16

```
# observe the attributes of the model
print_grid_search_attributes(lr_svc_grid_results['model'])
```

OUTPUT 16

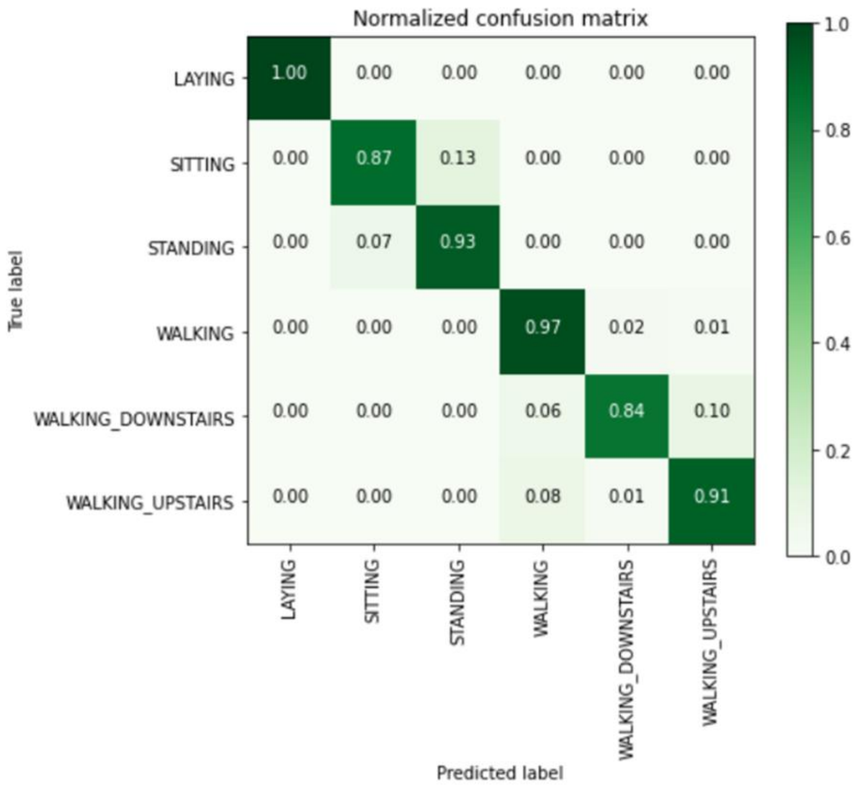
```
--------------------------
|      Best Estimator     |
--------------------------

         LinearSVC(C=0.5, class_weight=None, dual=True, fit_intercept=True,
        intercept_scaling=1, loss='squared_hinge', max_iter=1000,
        multi_class='ovr', penalty='l2', random_state=None, tol=5e-05,
        verbose=0)

--------------------------
|     Best parameters     |
--------------------------
         Parameters of best estimator :

         {'C': 0.5}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

         Total number of cross validation sets: 5

--------------------------
|       Best Score        |
--------------------------

         Average Cross Validate scores of best estimator :

         0.9420644015594
```

INPUT 17

```
# RANDOM FOREST (ALGORITHM 3)
from sklearn.ensemble import RandomForestClassifier

# start Grid search
params = {'n_estimators': np.arange(10,201,20), 'max_depth':np.arange(3,15,2)}
rfc = RandomForestClassifier()
rfc_grid = GridSearchCV(rfc, param_grid=params, n_jobs=-1)
rfc_grid_results = perform_model(rfc_grid, X_train, y_train, X_test, y_test,
class_labels=labels)
```

OUTPUT 17

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:25:49.011720


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.043691


---------------------
|      Accuracy      |
---------------------

    0.9239904988123515


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 429  62   0   0   0]
 [  0  38 494   0   0   0]
 [  0   0   0 481  10   5]
 [  0   0   0  26 354  40]
 [  0   0   0  36   7 428]]
```




```
-------------------------
| Classification Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.92      0.87      0.90       491
          STANDING       0.89      0.93      0.91       532
           WALKING       0.89      0.97      0.93       496
WALKING_DOWNSTAIRS       0.95      0.84      0.90       420
  WALKING_UPSTAIRS       0.90      0.91      0.91       471

          accuracy                           0.92      2947
         macro avg       0.93      0.92      0.92      2947
      weighted avg       0.93      0.92      0.92      2947
```

INPUT 18

```
# plotting confusion matrix
plt.figure(figsize=(5.3,5.3))
```

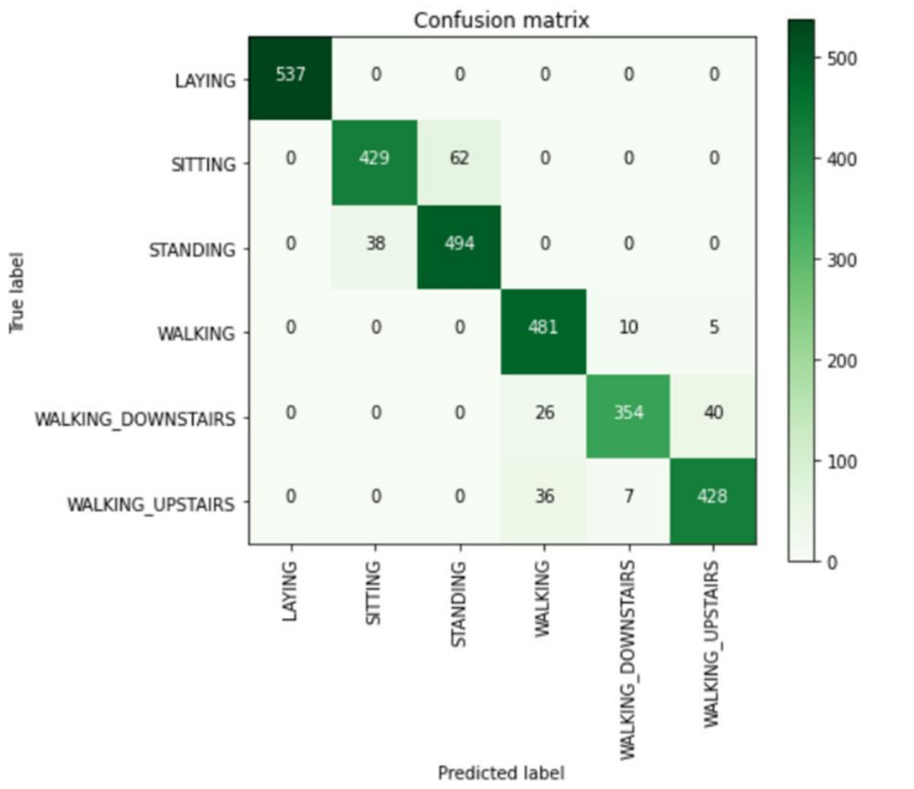
OUTPUT 18

```
<Figure size 530x530 with 0 Axes>
```

INPUT 19

```
plt.grid(b=False)
plot_confusion_matrix(rfc_grid_results['confusion_matrix'], classes=labels,
cmap=plt.cm.Greens, )
plt.show()
```

OUTPUT 19



INPUT 20

```
# observe the attributes of the model
print_grid_search_attributes(rfc_grid_results['model'])
```

OUTPUT 20

```
--------------------------
|      Best Estimator     |
--------------------------

        RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                criterion='gini', max_depth=11, max_features='auto',
                max_leaf_nodes=None, max_samples=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, n_estimators=110,
                n_jobs=None, oob_score=False, random_state=None,
                verbose=0, warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'max_depth': 11, 'n_estimators': 110}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total number of cross validation sets: 5

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.9208435189167442
```

INPUT 21

```
# Comparing all models

import matplotlib.ticker as mticker

labels = ['Logistic Regression', 'Support Vector Machine', 'Random Forest']

accuracy_whole = [(log_reg_grid_results['accuracy'] * 100),
(lr_svc_grid_results['accuracy'] * 100), (rfc_grid_results['accuracy'] * 100)]
accuracy_2f = ["%.2f" % member for member in accuracy_whole]
accuracy = [(float(accu_x)) for accu_x in accuracy_2f]

error_whole = [(100-(log_reg_grid_results['accuracy'] * 100)), (100-
(lr_svc_grid_results['accuracy'] * 100)), (100-(rfc_grid_results['accuracy'] * 100))]
error_2f = ["%.2f" % member for member in error_whole]
error = [(float(err_x)) for err_x in error_2f]

# the label locations
x = np.arange(len(labels))
# the width of the bars
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, accuracy, width, label='Accuracy')
rects2 = ax.bar(x + width/2, error, width, label='Misclassification Error')


# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Machine Learning Algorithms')
ax.set_ylabel('Accuracy and Misclassification Error (%)')
ax.set_title('Model Comparison')
ax.set_xticks(x)
ax.set_ylim(0,110)
ax.yaxis.set_major_locator(mticker.MaxNLocator(nbins=12, prune='upper'))
ax.set_xticklabels(labels,fontsize=8.8)
ax.legend(loc="upper left",bbox_to_anchor=(1,1))

# Attach a text label above each bar in *rects*, displaying its height
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    # 3 points vertical offset
                    xytext=(0, 3),
                    textcoords="offset points"
                    ha='center', va='bottom')


autolabel(rects1)
autolabel(rects2)

fig.tight_layout()

plt.show()
```

OUTPUT 21