Data Mining Project

(- Under the supervision of **DR. Bhaskar Biswas**)

Human Activity Recognition

- Neha Kumari (18074012)
- Kumar Shivam Ranjan (18075031)
- Madhav Bansal (18075036)

Project Overview

- **Human Activity Recognition**, is the problem of predicting what kind of activity a person is performing based on a signals detected by smartphone sensors on their waist.

- Two types of sensors present in smartphones are:

    - Accelerometer

    - Gyroscope

- **Accelerometer** measures acceleration and **Gyroscope** measures angular velocity.

How the data was prepared?

- 30 volunteers referred to as **subjects** performed the experiment for data collection wearing smartphones sensors on their waist.
- The two smartphone sensors captured the **3 axial linear acceleration** as well as the **3 axial angular velocity** of the subject.
- The sensor signals were sampled in fixed-width sliding windows of **2.56** sec and 50% overlap (**128** readings/window).
- The data were recorded at the constant frequency of **50Hz** (50 data points were recorded each second)

Additional Information

- 561 feature vector were engineered from each time window of 2.56 second with both time and frequency domain variables
- Features are normalized and bounded within [-1,1]
- The gyroscopic data is measured in radian/sec
- The units used for the accelerations are 'g' (9.8 m/s2)
- Accelerometer readings are divided into gravity acceleration and body acceleration readings, which has x,y and z components each.
- Gyroscope readings are the measure of angular velocities which has x,y and z components.
- Fourier Transforms are made on the above time readings to obtain frequency readings.
- Now, on all the base signal readings, mean, std,max, skewness, kurtosis, etc are calculated for each window.

Quick Overview of the Dataset

1. Data is downloaded from following source:
   Human Activity Recognition Using Smartphones Data Set
2. Feature names are present in **UCI_HAR_dataset/features.txt**
3. **Train Data**
   a) UCI_HAR_dataset/train/X_train.txt
   b) UCI_HAR_dataset/train/subject_train.txt
   c) UCI_HAR_dataset/train/y_train.txt
4. **Test Data**
   a) UCI_HAR_dataset/test/X_test.txt
   b) UCI_HAR_dataset/test/subject_test.txt
   c) UCI_HAR_dataset/test/y_test.txt

1. UCI_HAR_dataset/ features_info.txt

Shows information about the variables used on the feature vector

```
These signals were used to estimate variables of the feature vector for each pattern:
'-XYZ' is used to denote 3-axial signals in the X, Y and Z directions.

tBodyAcc-XYZ
tGravityAcc-XYZ
tBodyAccJerk-XYZ
tBodyGyro-XYZ
tBodyGyroJerk-XYZ
tBodyAccMag
tGravityAccMag
tBodyAccJerkMag
tBodyGyroMag
tBodyGyroJerkMag
fBodyAcc-XYZ
fBodyAccJerk-XYZ
fBodyGyro-XYZ
fBodyAccMag
fBodyAccJerkMag
fBodyGyroMag
fBodyGyroJerkMag
```

2. UCI_HAR_dataset/train/subject_train.txt

(Each row identifies the subject who performed the activity for each window sample. Its range is from 1 to 30)

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

3. **UCI_HAR_dataset/train/Inertial Signals/total_acc_x_train.txt**
   (The acceleration signal from the smartphone accelerometer X axis in standard gravity units 'g'. Every row shows a 128 element vector. The same description applies for the **total_acc_x_train.txt** and **total_acc_z_train.txt** files for the Y and Z axis)

```
1  1.0128170e+000 1.0228330e+000 1.0220280e+000 1.0178770e+000 1.0236800e+000 1.0169740e+000 1.0177460e+000 1.0192630e+000 1.0164170e+000
   1.0207450e+000 1.0186430e+000 1.0195210e+000 1.0202600e+000 1.0180410e+000 1.0208290e+000 1.0186440e+000 1.0193980e+000 1.0203990e+000
   1.0192220e+000 1.0220930e+000 1.0204330e+000 1.0205340e+000 1.0215030e+000 1.0199310e+000 1.0204800e+000 1.0189450e+000 1.0192380e+000
   1.0199890e+000 1.0189170e+000 1.0197620e+000 1.0190210e+000 1.0178870e+000 1.0181360e+000 1.0195430e+000 1.0202420e+000 1.0187570e+000
   1.0195340e+000 1.0198620e+000 1.0190600e+000 1.0207170e+000 1.0210550e+000 1.0201780e+000 1.0181080e+000 1.0147760e+000 1.0153740e+000
   1.0184290e+000 1.0198950e+000 1.0186470e+000 1.0163870e+000 1.0170530e+000 1.0195720e+000 1.0210970e+000 1.0194880e+000 1.0172180e+000
   1.0198760e+000 1.0220220e+000 1.0205740e+000 1.0215880e+000 1.0222980e+000 1.0193690e+000 1.0169800e+000 1.0167740e+000 1.0160790e+000
   1.0152920e+000 1.0188510e+000 1.0223800e+000 1.0207810e+000 1.0202180e+000 1.0213440e+000 1.0205220e+000 1.0197900e+000 1.0192160e+000
   1.0183070e+000 1.0179960e+000 1.0179320e+000 1.0181210e+000 1.0183050e+000 1.0184580e+000 1.0182010e+000 1.0171290e+000 1.0178140e+000
   1.0188000e+000 1.0176010e+000 1.0179700e+000 1.0184890e+000 1.0177870e+000 1.0191670e+000 1.0197890e+000 1.0194620e+000 1.0204330e+000
   1.0211890e+000 1.0219030e+000 1.0219360e+000 1.0205500e+000 1.0188780e+000 1.0185480e+000 1.0173890e+000 1.0150210e+000 1.0193100e+000
   1.0246060e+000 1.0218630e+000 1.0202010e+000 1.0205730e+000 1.0187290e+000 1.0193600e+000 1.0199540e+000 1.0189690e+000 1.0196330e+000
   1.0195530e+000 1.0191790e+000 1.0196950e+000 1.0191450e+000 1.0185160e+000 1.0179260e+000 1.0177800e+000 1.0189170e+000 1.0206060e+000
   1.0225830e+000 1.0209810e+000 1.0180650e+000 1.0196380e+000 1.0200170e+000 1.0187660e+000 1.0198150e+000 1.0192900e+000 1.0184450e+000
   1.0193720e+000 1.0211710e+000
2  1.0188510e+000 1.0223800e+000 1.0207810e+000 1.0202180e+000 1.0213440e+000 1.0205220e+000 1.0197900e+000 1.0192160e+000 1.0183070e+000
   1.0179960e+000 1.0179320e+000 1.0181210e+000 1.0183050e+000 1.0184580e+000 1.0182010e+000 1.0171290e+000 1.0178140e+000 1.0188000e+000
   1.0176010e+000 1.0179700e+000 1.0184890e+000 1.0177870e+000 1.0191670e+000 1.0197890e+000 1.0194620e+000 1.0204330e+000 1.0211890e+000
   1.0219030e+000 1.0219360e+000 1.0205500e+000 1.0188780e+000 1.0185480e+000 1.0173890e+000 1.0150210e+000 1.0193100e+000 1.0246060e+000
   1.0218630e+000 1.0202010e+000 1.0205730e+000 1.0187290e+000 1.0193600e+000 1.0199540e+000 1.0189690e+000 1.0196330e+000 1.0195530e+000
   1.0191790e+000 1.0196950e+000 1.0191450e+000 1.0185160e+000 1.0179260e+000 1.0177800e+000 1.0189170e+000 1.0206060e+000 1.0225830e+000
   1.0209810e+000 1.0180650e+000 1.0196380e+000 1.0200170e+000 1.0187660e+000 1.0198150e+000 1.0192900e+000 1.0184450e+000 1.0193720e+000
   1.0211710e+000 1.0231270e+000 1.0218820e+000 1.0191780e+000 1.0158610e+000 1.0128930e+000 1.0164510e+000 1.0203310e+000 1.0202660e+000
   1.0217590e+000 1.0186490e+000 1.0131170e+000 1.0161670e+000 1.0189770e+000 1.0166530e+000 1.0177820e+000 1.0205280e+000 1.0218770e+000
   1.0220960e+000 1.0207310e+000 1.0207610e+000 1.0204050e+000 1.0202130e+000 1.0216750e+000 1.0199890e+000 1.0179970e+000 1.0173910e+000
   1.0179940e+000 1.0216610e+000 1.0223480e+000 1.0203360e+000 1.0190660e+000 1.0188820e+000 1.0200130e+000 1.0182620e+000 1.0174740e+000
   1.0189540e+000 1.0196280e+000 1.0227940e+000 1.0242380e+000 1.0227830e+000 1.0204870e+000 1.0181460e+000 1.0197790e+000 1.0199220e+000
   1.0187090e+000 1.0199800e+000 1.0193010e+000 1.0192430e+000 1.0201690e+000 1.0208920e+000 1.0227710e+000 1.0215530e+000 1.0198110e+000
```

4. UCI_HAR_dataset/activity_labels.txt

Links the class labels with their activity name

```
1 WALKING
2 WALKING_UPSTAIRS
3 WALKING_DOWNSTAIRS
4 SITTING
5 STANDING
6 LAYING
```

Problem Statement

Predict one of the following six activities that a Smartphone user is performing at that 2.56 Seconds time window by using either 561 feature data or raw features of 128 reading.

1. Walking
2. Walking Upstairs
3. Walking Downstairs
4. Sitting
5. Standing
6. Laying

Deep learning model

- Accelerometer data was divided into body acceleration and total acceleration (body = total - gravitational force)
- The readings from 70% of the volunteers were taken as training data and 30% volunteers records were taken for test data.
- We can use LSTM (long short term memory) model of the Recurrent Neural Network (RNN) to recognize various activities of humans like standing, climbing upstairs and downstairs etc.

Why LSTM?

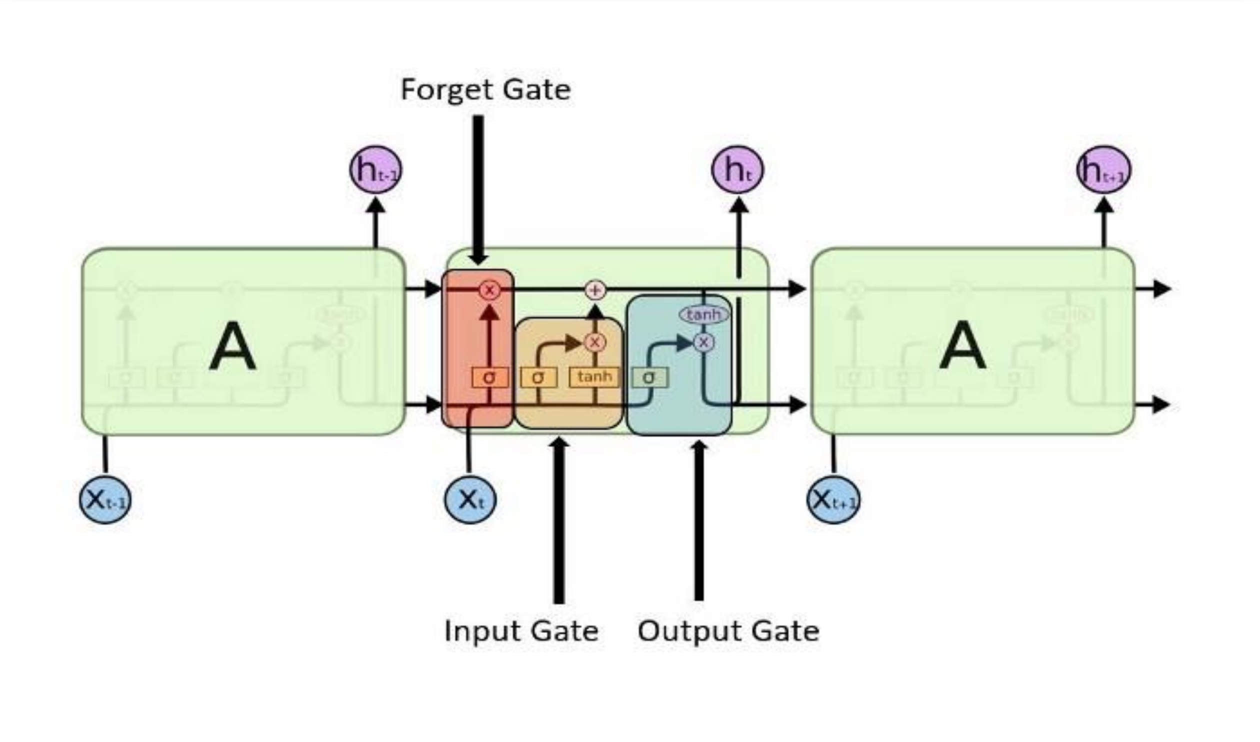
- **LSTM model** is a type of recurrent neural network.It helps us to look at recent information to perform the present task
- LSTM works well on time-series data.
- This model is used as this helps in remembering values over arbitrary intervals.

Long Short Term Memory Networks

- RNNs work better for time series data. Out project data has time window of 2.56 second. In this window, particular volunteer is performing movements like standing ,sitting etc and we are capturing their linear acceleration and angular velocity . RNNs usually handle such time series data very well.
- RNNs definitely solve the problem of long term dependencies by using the previous output of the neuron to predict the output of the current neuron but they do suffer serious problems when gap between relevant information and where it is needed become predominantly large.
- Moreover,They do suffer from
  i) Vanishing Gradient Problem
  ii) Exploding Gradient Problem

That is when LSTMs came into picture.

LSTM architecture that will solve HAR problem



Machine learning Model(Logistic Regression)

- In Deep learning Model , we used raw features of 128 readings to predict the activity.The Dataset used there was basically a 3D matrix of(7352x128x9).
- The Same Dataset cannot be used to train the machine learning model as RNNs are designed to work with time series data not machine learning.
- This is why 561 features were engineered from raw 128 accelerometer and gyroscope signal readings so that we can use them in our machine learning model.
- We will apply classical Machine Learning models on these 561 sized domain expert engineered features.
- As we know that LSTM works well on time-series data, so,we will apply LSTM of Recurrent Neural Networks on 128 sized raw readings that we obtained from accelerometer and gyroscope signals.

Importing all the libraries needed for the LSTM model

In [3]:
```
import numpy as np       #numpy
import pandas as pd      #pandas
import matplotlib.pyplot as plt   #matplotlib
import seaborn as sns             #seaborn
import tensorflow as tf           #tensorflow
from keras.models import Sequential    #keras
from keras.layers import LSTM          #keras layers
from keras.layers.core import Dense, Dropout  #keras layers core
from keras.layers.normalization import BatchNormalization  #keras layers normalisation
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
import matplotlib.pyplot as plt
```

In [4]:
```
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}
# Data directory
DATADIR = 'UCI_HAR_Dataset'

labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_UPSTAIRS']
```

Obtaining the train data

In [5]:
```
X_train = pd.read_csv('UCI_HAR_Dataset/train/X_train.txt', delim_whitespace=True, header=None)


X_train['subject'] = pd.read_csv('UCI_HAR_Dataset/train/subject_train.txt', header=None, squeeze=True)

y_train = pd.read_csv('UCI_HAR_Dataset/train/y_train.txt', names=['Activity'], squeeze=True)
y_train_labels = y_train.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',4:'SITTING', 5:'STANDING',6

train = X_train
train['Activity'] = y_train
train['ActivityName'] = y_train_labels
train.head()
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	554	555	556	557
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.112754	0.030400	-0.464761	-0.018446
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.053477	-0.007435	-0.732626	0.703511
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.118559	0.177899	0.100699	0.808529
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	-0.036788	-0.012892	0.640011	-0.485366
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.123320	0.122542	0.693578	-0.615971

5 rows × 564 columns

In [6]:
```
print(train.shape,train.size)
```

```
(7352, 564) 4146528
```

Obtaining the test data

In [7]:

```
X_test = pd.read_csv('UCI_HAR_Dataset/test/X_test.txt', delim_whitespace=True, header=None)


X_test['subject'] = pd.read_csv('UCI_HAR_Dataset/test/subject_test.txt', header=None, squeeze=True)


y_test = pd.read_csv('UCI_HAR_Dataset/test/y_test.txt', names=['Activity'], squeeze=True)
y_test_labels = y_test.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS', 4:'SITTING', 5:'STANDING',6:


test = X_test
test['Activity'] = y_test
test['ActivityName'] = y_test_labels
test.head()
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	554	555	556	557
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.925249	-0.674302	-0.894088	...	0.006462	0.162920	-0.825886	0.271151
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.968401	-0.945823	-0.894088	...	-0.083495	0.017500	-0.434375	0.920593
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.970735	-0.963483	-0.939260	...	-0.034956	0.202302	0.064103	0.145068
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.974471	-0.968897	-0.938610	...	-0.017067	0.154438	0.340134	0.296407
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.965953	-0.977346	-0.938610	...	-0.002223	-0.040046	0.736715	-0.118545

5 rows × 564 columns

In [8]:

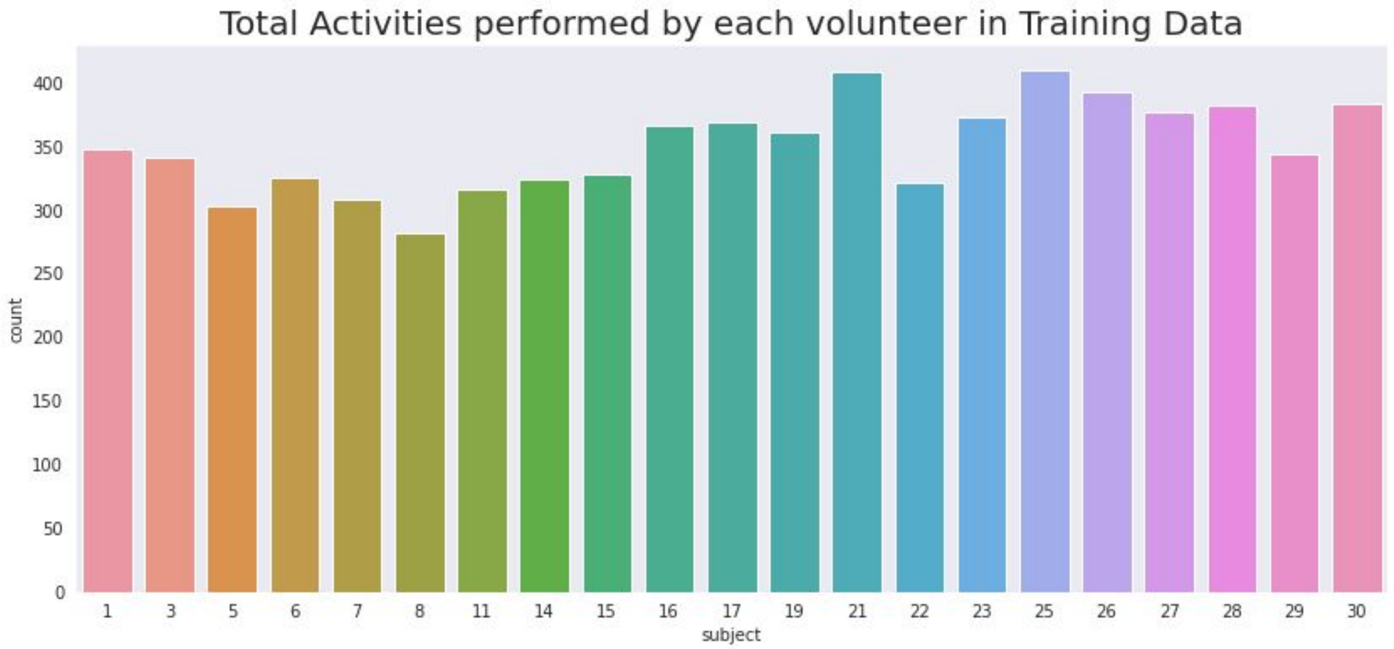
```
print(test.shape,test.size)
```

```
(2947, 564) 1662108
```

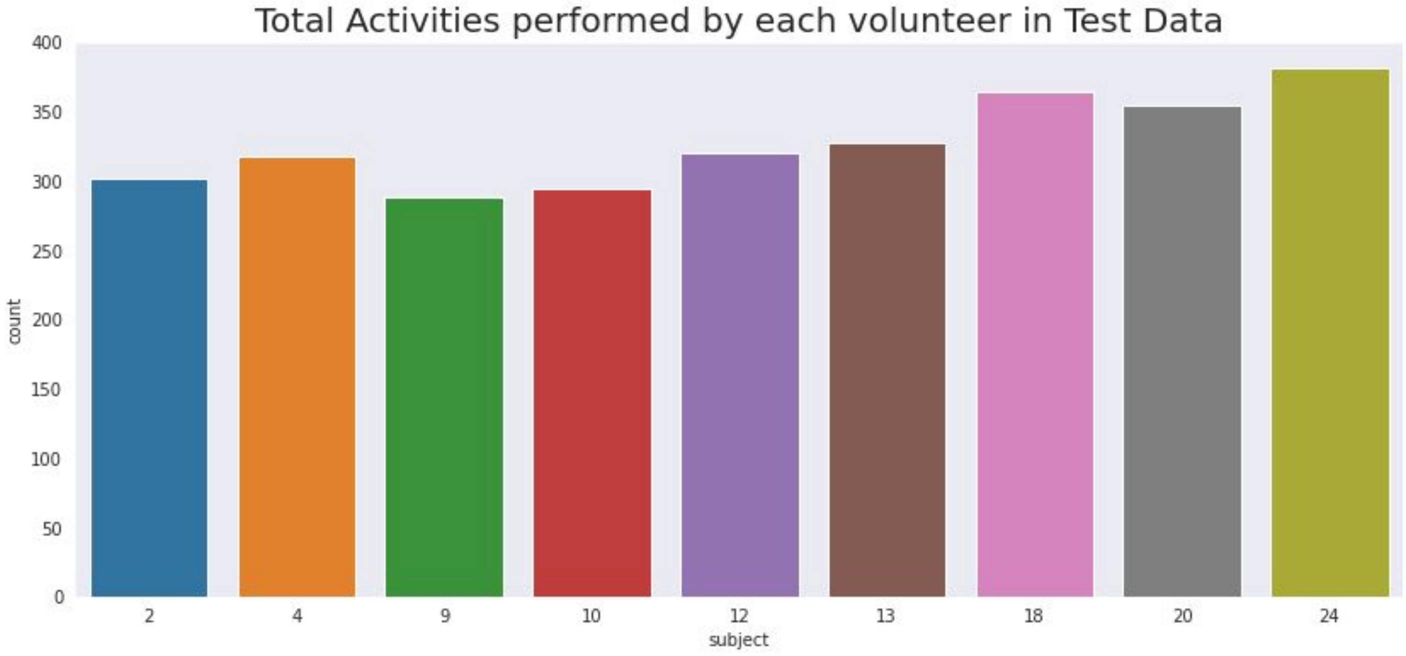
1. Visualising the Dataset

In [12]:
```
sns.set_style('dark')
plt.rcParams['lines.linewidth'] = 1
plt.rcParams['font.family'] = 'DejaVu Sans'
```

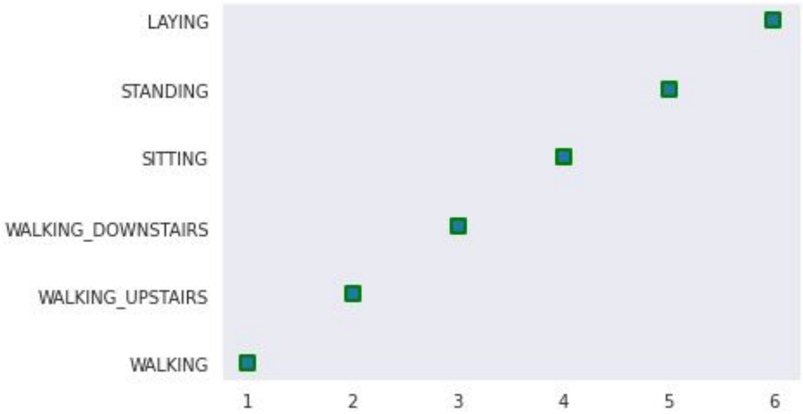
In [13]:
```
plt.figure(figsize=(14,6))
plt.title("Total Activities performed by each volunteer in Training Data",fontsize=20)
sns.countplot(x=train.subject)
plt.show()
```



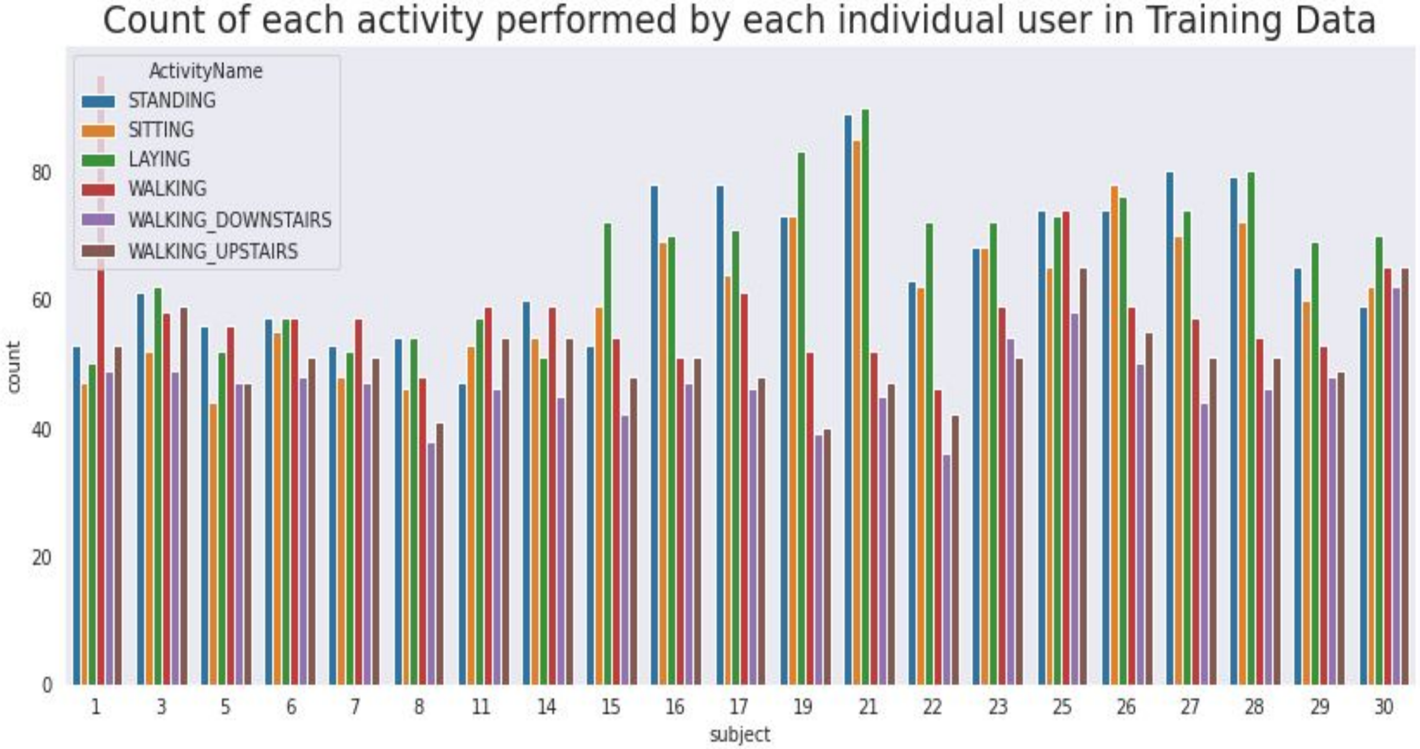
In [14]:
```
plt.figure(figsize=(14,6))
plt.title("Total Activities performed by each volunteer in Test Data",fontsize=20)
sns.countplot(x=test.subject)
plt.show()
```



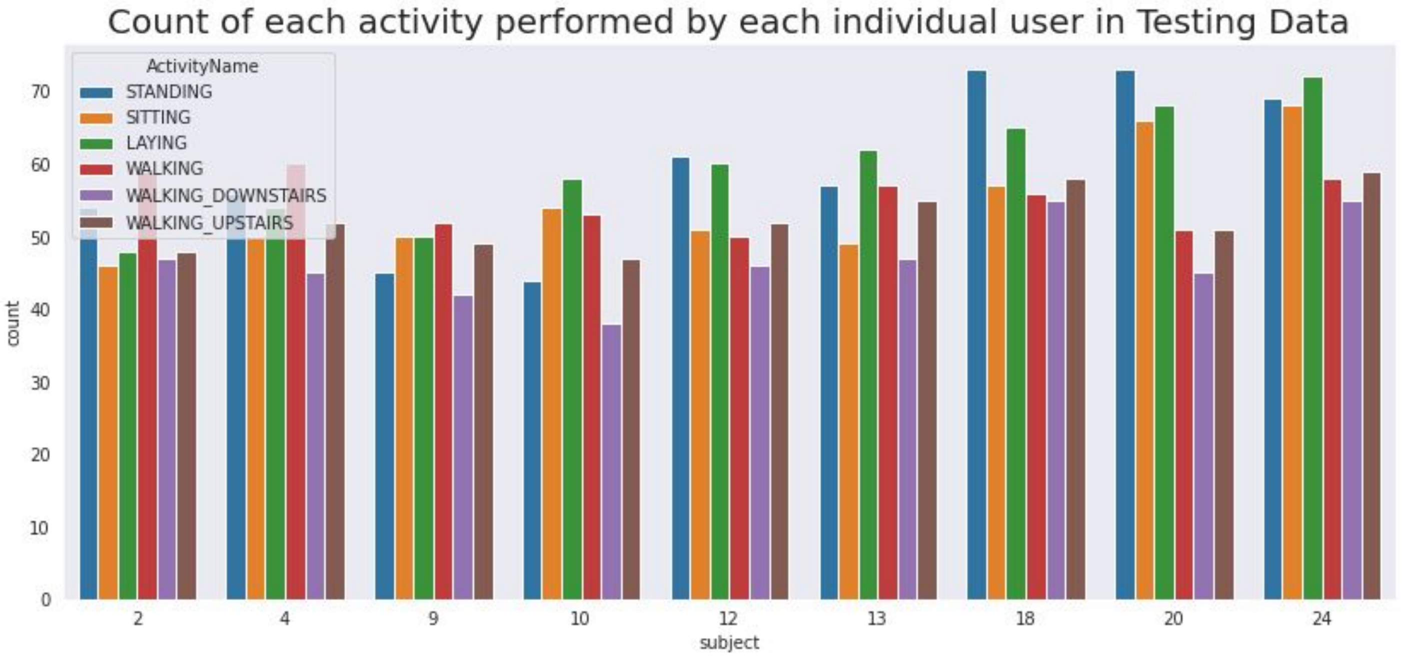
In [15]:
```
y_train_labels = {1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',4:'SITTING', 5:'STANDING',6:'LAYING'}
lists=sorted(y_train_labels.items())
x, y = zip(*lists)
plt.scatter(x, y,linewidths = 2,marker ="s",edgecolor ="green", s = 50)
plt.show()
```



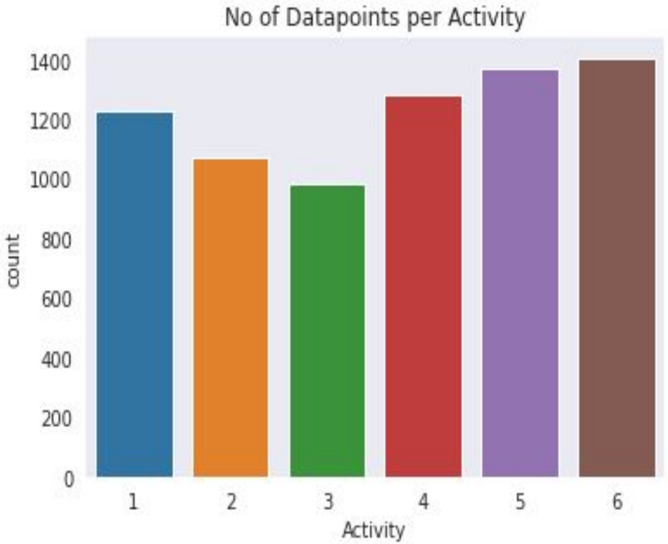
In [16]:
```
plt.figure(figsize=(14,6))
plt.title('Count of each activity performed by each individual user in Training Data', fontsize=20)
sns.countplot(x='subject',hue='ActivityName', data = train)
plt.show()
```



In [17]:
```
plt.figure(figsize=(14,6))
plt.title('Count of each activity performed by each individual user in Testing Data', fontsize=20)
sns.countplot(x='subject',hue='ActivityName', data = test)
plt.show()
```



In [18]:
```
plt.title('No of Datapoints per Activity')
sns.countplot(x=train.Activity)
plt.xticks(rotation=0)
plt.show()
```



LSTM model for HAR

In [19]:
```
np.random.seed(42)
import tensorflow as tf
tf.random.set_seed(42)
```

In [20]:
```
# Configuring a session
session_conf = tf.compat.v1.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [21]:
```
# Import Keras
from tensorflow.compat.v1.keras import backend as K
sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [22]:
```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
from keras.layers.normalization import BatchNormalization
```

In [24]:
```python
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]
```

In [25]:
```python
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

def generate_x_data(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(_read_csv(filename).values)
    return np.transpose(signals_data, (1, 2, 0))


def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [26]:
```python
def generate_y_data(subset):

    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).values
```

In [27]:
```
def load_data():

    X_train, X_test = generate_x_data('train'), generate_x_data('test')
    y_train, y_test = generate_y_data('train'), generate_y_data('test')

    return X_train, X_test, y_train, y_test
```

In [29]:
```
X_train, X_test, Y_train, Y_test = load_data()
```

In [30]:
```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
print(n_classes)
```

```
128
9
7352
6
```

In [31]:
```
epochs = 30
batch_size = 32
n_hidden = 128
pv = 0.25
```

In [33]:
```
model = Sequential()

model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
model.add(BatchNormalization())

model.add(Dropout(pv))

model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 128)               70656
_________________________________________________________________
batch_normalization (BatchNo (None, 128)               512
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 6)                 774
=================================================================
Total params: 71,942
Trainable params: 71,686
Non-trainable params: 256
```

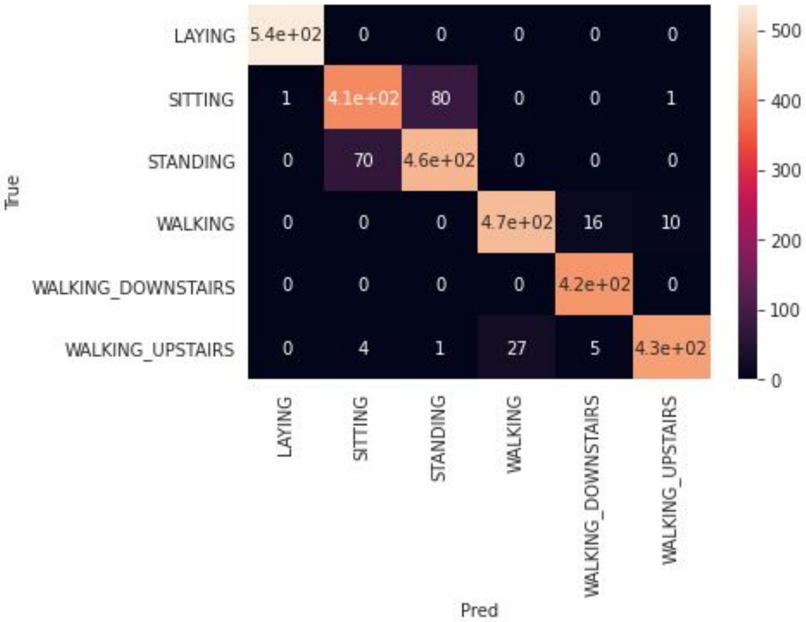
In [34]:
```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',metrics=['accuracy'])
```

In [35]:
```
model.fit(X_train,Y_train,batch_size=batch_size,validation_data=(X_test, Y_test),epochs=epochs)
```

```
Epoch 1/30
230/230 [==============================] - 17s 72ms/step - loss: 0.9775 - accuracy: 0.5635 - val_loss: 0.9544 - va
l_accuracy: 0.5836
Epoch 2/30
230/230 [==============================] - 16s 71ms/step - loss: 0.7032 - accuracy: 0.6602 - val_loss: 0.7071 - va
l_accuracy: 0.6566
Epoch 3/30
230/230 [==============================] - 17s 73ms/step - loss: 0.6357 - accuracy: 0.7076 - val_loss: 0.7523 - va
l_accuracy: 0.6203
Epoch 4/30
230/230 [==============================] - 18s 76ms/step - loss: 0.4320 - accuracy: 0.8298 - val_loss: 0.3244 - va
l_accuracy: 0.8894
Epoch 5/30
230/230 [==============================] - 18s 78ms/step - loss: 0.2150 - accuracy: 0.9214 - val_loss: 0.3481 - va
l_accuracy: 0.8823
Epoch 6/30
230/230 [==============================] - 18s 76ms/step - loss: 0.1855 - accuracy: 0.9320 - val_loss: 0.3101 - va
l_accuracy: 0.9006
Epoch 7/30
230/230 [==============================] - 18s 77ms/step - loss: 0.1877 - accuracy: 0.9328 - val_loss: 0.3312 - va
l_accuracy: 0.8887
Epoch 8/30
230/230 [==============================] - 18s 77ms/step - loss: 0.1598 - accuracy: 0.9377 - val_loss: 0.3471 - va
l_accuracy: 0.9019
Epoch 9/30
230/230 [==============================] - 18s 77ms/step - loss: 0.1444 - accuracy: 0.9377 - val_loss: 0.5204 - va
l_accuracy: 0.9087
Epoch 10/30
230/230 [==============================] - 18s 78ms/step - loss: 0.1532 - accuracy: 0.9421 - val_loss: 0.3719 - va
l_accuracy: 0.8911
Epoch 11/30
230/230 [==============================] - 18s 79ms/step - loss: 0.1623 - accuracy: 0.9429 - val_loss: 0.2438 - va
```

In [45]:
```
result=confusion_matrix(Y_test, model.predict(X_test))
sn.heatmap(result, annot=True)
```

Out[45]: <AxesSubplot:xlabel='Pred', ylabel='True'>



```
In [38]: score = model.evaluate(X_test, Y_test)

93/93 [==============================] - 2s 18ms/step - loss: 0.3387 - accuracy: 0.9270

In [39]: score

Out[39]: [0.33865147829055786, 0.9270444512367249]
```

- With a simple semi tuned stacked LSTM architecture we got 92.7% accuracy and a loss of 0.33

Machine learning model(Logistic Regression)

In [40]:
```
print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
y_test.sample()
y_test
```

```
X_train and y_train : ((7352, 128, 9),(7352,))
X_test  and y_test  : ((2947, 128, 9),(2947,))
```

Out[40]:
```
0       5
1       5
2       5
3       5
4       5
       ..
2942    2
2943    2
2944    2
2945    2
2946    2
Name: Activity, Length: 2947, dtype: int64
```

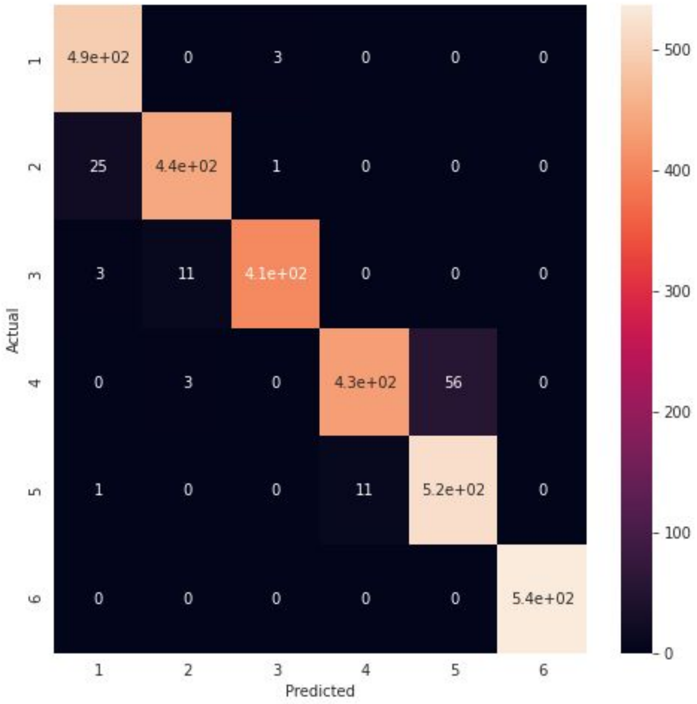
In [41]:
```

# get X_train and y_train from csv files
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train.Activity
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test.Activity
print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
print(y_train.head())
```

```
X_train and y_train : ((7352, 561),(7352,))
X_test  and y_test  : ((2947, 561),(2947,))
```

In [56]:
```
logistic_regression= LogisticRegression(solver='lbfgs',max_iter=100000)
result=logistic_regression.fit(X_train,y_train)
y_pred=logistic_regression.predict(X_test)
```

In [57]:
```
plt.figure(figsize=(8,8))
plt.grid(b=False)
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
```

Out[57]: <AxesSubplot:xlabel='Predicted', ylabel='Actual'>



```
In [47]: print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
         plt.show()

Accuracy:  0.9613165931455717
```

Final Comments:
LSTM Model Accuracy: 92.7%
Machine learning Model Accuracy: 96.13%