

## ✓ HumanActivityRecognition

This project is to build a model that predicts the human activities such as Walking, Walking\_Upstairs, Walking\_Downstairs, Sitting, Standing or Laying.

This dataset is collected from 30 persons(referred as subjects in this dataset), performing different activities with a smartphone to their waists. The data is recorded with the help of sensors (accelerometer and Gyroscope) in that smartphone. This experiment was video recorded to label the data manually.

### How data was recorded

By using the sensors(Gyroscope and accelerometer) in a smartphone, they have captured '3-axial linear acceleration'(*tAcc-XYZ*) from accelerometer and '3-axial angular velocity' (*tGyro-XYZ*) from Gyroscope with several variations.

prefix 't' in those metrics denotes time.

suffix 'XYZ' represents 3-axial signals in X , Y, and Z directions.

### Feature names

1. These sensor signals are preprocessed by applying noise filters and then sampled in fixed-width windows(sliding windows) of 2.56 seconds each with 50% overlap. ie., each window has 128 readings.
2. From Each window, a feature vector was obtained by calculating variables from the time and frequency domain.

In our dataset, each datapoint represents a window with different readings

3. The acceleration signal was saperated into Body and Gravity acceleration signals(*tBodyAcc-XYZ* and *tGravityAcc-XYZ*) using some low pass filter with corner frequency of 0.3Hz.
4. After that, the body linear acceleration and angular velocity were derived in time to obtain *jerk signals* (*tBodyAccJerk-XYZ* and *tBodyGyroJerk-XYZ*).
5. The magnitude of these 3-dimensional signals were calculated using the Euclidian norm. This magnitudes are represented as features with names like *tBodyAccMag*, *tGravityAccMag*, *tBodyAccJerkMag*, *tBodyGyroMag* and *tBodyGyroJerkMag*.
6. Finally, We've got frequency domain signals from some of the available signals by applying a FFT (Fast Fourier Transform). These signals obtained were labeled with **prefix 'f'** just like original signals with **prefix 't'**. These signals are labeled as ***fBodyAcc-XYZ***, ***fBodyGyroMag*** etc.,.
7. These are the signals that we got so far.
  - *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*
8. We can estimate some set of variables from the above signals. ie., We will estimate the following properties on each and every signal that we recorded so far.
  - ***mean()***: Mean value
  - ***std()***: Standard deviation
  - ***mad()***: Median absolute deviation
  - ***max()***: Largest value in array
  - ***min()***: Smallest value in array

- **sma()**: Signal magnitude area
- **energy()**: Energy measure. Sum of the squares divided by the number of values.
- **iqr()**: Interquartile range
- **entropy()**: Signal entropy
- **arCoeff()**: Autorregresion coefficients with Burg order equal to 4
- **correlation()**: correlation coefficient between two signals
- **maxInds()**: index of the frequency component with largest magnitude
- **meanFreq()**: Weighted average of the frequency components to obtain a mean frequency
- **skewness()**: skewness of the frequency domain signal
- **kurtosis()**: kurtosis of the frequency domain signal
- **bandsEnergy()**: Energy of a frequency interval within the 64 bins of the FFT of each window.
- **angle()**: Angle between to vectors.

9. We can obtain some other vectors by taking the average of signals in a single window sample. These are used on the angle() variable`

- gravityMean
- tBodyAccMean
- tBodyAccJerkMean
- tBodyGyroMean
- tBodyGyroJerkMean

## Y\_Labels(Encoded)

- In the dataset, Y\_labels are represented as numbers from 1 to 6 as their identifiers.
  - WALKING as **1**
  - WALKING\_UPSTAIRS as **2**
  - WALKING\_DOWNSTAIRS as **3**
  - SITTING as **4**
  - STANDING as **5**
  - LAYING as **6**

## Train and test data were saperated

- The readings from **70%** of the volunteers were taken as **training data** and remaining **30%** subjects recordings were taken for **test data**

## Data

- All the data is present in 'UCI\_HAR\_dataset/' folder in present working directory.
  - Feature names are present in 'UCI\_HAR\_dataset/features.txt'
  - **Train Data**
    - 'UCI\_HAR\_dataset/train/X\_train.txt'
    - 'UCI\_HAR\_dataset/train/subject\_train.txt'
    - 'UCI\_HAR\_dataset/train/y\_train.txt'
  - **Test Data**
    - 'UCI\_HAR\_dataset/test/X\_test.txt'
    - 'UCI\_HAR\_dataset/test/subject\_test.txt'
    - 'UCI\_HAR\_dataset/test/y\_test.txt'

## Data Size :

27 MB

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

## ✓ Quick overview of the dataset :

- Accelerometer and Gyroscope readings are taken from 30 volunteers(referred as subjects) while performing the following 6 Activities.

1. Walking

2. WalkingUpstairs
3. WalkingDownstairs
4. Standing
5. Sitting
6. Lying.

- Readings are divided into a window of 2.56 seconds with 50% overlapping.
- 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.
- Jerk signals are calculated for BodyAcceleration readings.
- Fourier Transforms are made on the above time readings to obtain frequency readings.
- Now, on all the base signal readings., mean, max, mad, sma, arcoefficient, engerybands,entropy etc., are calculated for each window.
- We get a feature vector of 561 features and these features are given in the dataset.
- Each window of readings is a datapoint of 561 features.

## Problem Framework

- 30 subjects(volunteers) data is randomly split to 70%(21) test and 30%(7) train data.
- Each datapoint corresponds one of the 6 Activities.

## ✓ Problem Statement

- Given a new datapoint we have to predict the Activity

Double-click (or enter) to edit

```
import numpy as np
import pandas as pd

# get the features from the file features.txt
features = list()
with open('UCI_HAR_Dataset/features.txt') as f:
    features = [line.split()[1] for line in f.readlines()]
print('No of Features: {}'.format(len(features)))
```

↗ No of Features: 561

## ✓ Obtain the train data

```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv('UCI_HAR_dataset/train/X_train.txt', delim_whitespace=True, header=None, names=features)

# add subject column to the dataframe
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:'LAYING'})


# put all columns in a single dataframe
train = X_train
train['Activity'] = y_train
train['ActivityName'] = y_train_labels
train.sample()
```

↗ D:\installed\Anaconda3\lib\site-packages\pandas\io\parsers.py:678: UserWarning: Duplicate names specified. This will raise an error  
return \_read(filepath\_or\_buffer, kwds)

|             | tBodyAcc-<br>mean()-X | tBodyAcc-<br>mean()-Y | tBodyAcc-<br>mean()-Z | tBodyAcc-<br>std()-X | tBodyAcc-<br>std()-Y | tBodyAcc-<br>std()-Z | tBodyAcc-<br>mad()-X | tBodyAcc-<br>mad()-Y | tBodyAcc-<br>mad()-Z | tBodyAcc-<br>max()-X | ... | angle(tBody |
|-------------|-----------------------|-----------------------|-----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|-----|-------------|
| <b>6015</b> | 0.2797                | -0.004397             | -0.10952              | 0.359081             | 0.119909             | -0.177541            | 0.337963             | 0.066883             | -0.221876            | 0.474093             | ... |             |

1 rows × 564 columns

train.shape

 (7352, 564)

## ✓ Obtain the test data

```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv('UCI_HAR_dataset/test/X_test.txt', delim_whitespace=True, header=None, names=features)
```

```
# add subject column to the dataframe
X_test['subject'] = pd.read_csv('UCI_HAR_dataset/test/subject_test.txt', header=None, squeeze=True)
```

```
# get y labels from the txt file
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: 'LAYING'})
```


```
# put all columns in a single dataframe
test = X_test
test['Activity'] = y_test
test['ActivityName'] = y_test_labels
test.sample()
```

 D:\installed\Anaconda3\lib\site-packages\pandas\io\parsers.py:678: UserWarning: Duplicate names specified. This will raise an error  
return \_read(filepath\_or\_buffer, kwds)

|             | tBodyAcc-<br>mean()-X | tBodyAcc-<br>mean()-Y | tBodyAcc-<br>mean()-Z | tBodyAcc-<br>std()-X | tBodyAcc-<br>std()-Y | tBodyAcc-<br>std()-Z | tBodyAcc-<br>mad()-X | tBodyAcc-<br>mad()-Y | tBodyAcc-<br>mad()-Z | tBodyAcc-<br>max()-X | ... | angle(tBody/ |
|-------------|-----------------------|-----------------------|-----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|-----|--------------|
| <b>2261</b> | 0.279196              | -0.018261             | -0.103376             | -0.996955            | -0.982959            | -0.988239            | -0.9972              | -0.982509            | -0.986964            | -0.940634            | ... |              |

1 rows × 564 columns

```
test.shape
```


 (2947, 564)

Double-click (or enter) to edit

## ✓ Data Cleaning

### ✓ 1. Check for Duplicates


```
print('No of duplicates in train: {}'.format(sum(train.duplicated())))
print('No of duplicates in test : {}'.format(sum(test.duplicated())))
```

 No of duplicates in train: 0  
No of duplicates in test : 0

Double-click (or enter) to edit

### ✓ 2. Checking for NaN/null values

```
print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))
print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```

 We have 0 NaN/Null values in train  
We have 0 NaN/Null values in test

Double-click (or enter) to edit

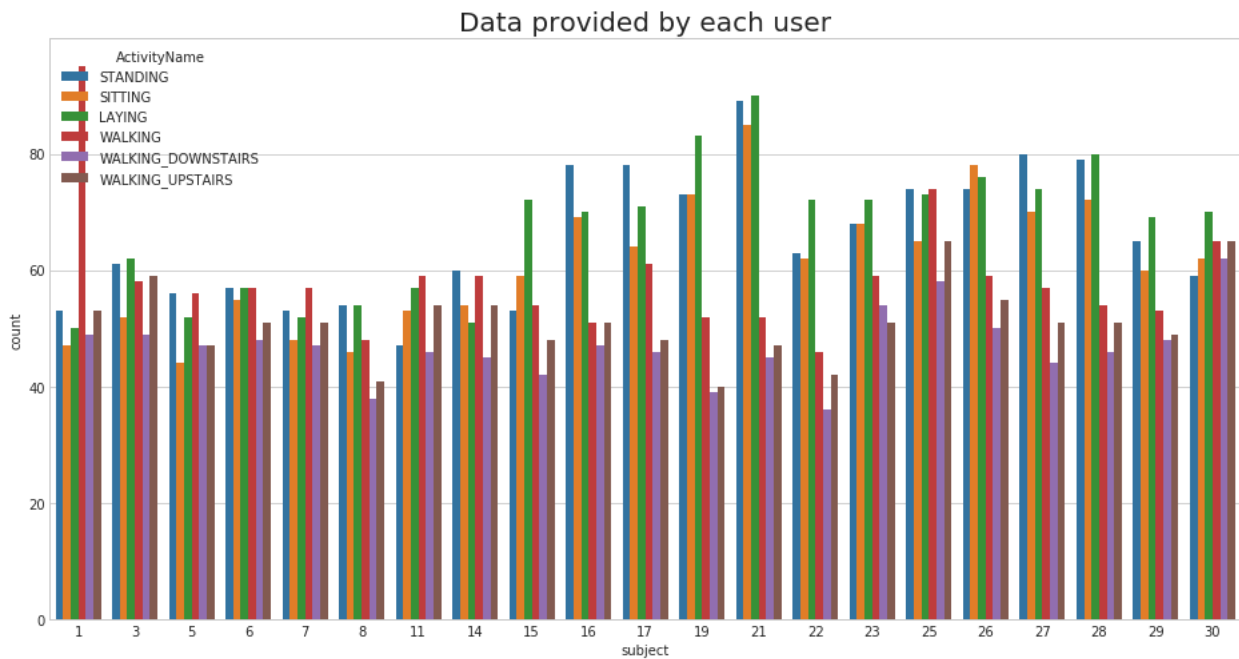
Double-click (or enter) to edit

### ✓ 3. Check for data imbalance

```
import matplotlib.pyplot as plt
import seaborn as sns
```

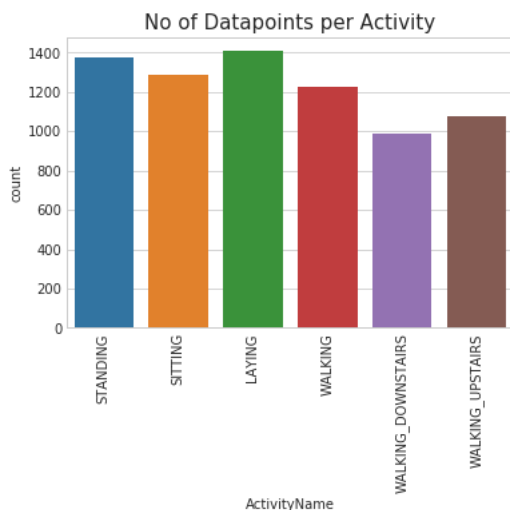
```
sns.set_style('whitegrid')
plt.rcParams['font.family'] = 'Dejavu Sans'

plt.figure(figsize=(16,8))
plt.title('Data provided by each user', fontsize=20)
sns.countplot(x='subject',hue='ActivityName', data = train)
plt.show()
```



We have got almost same number of reading from all the subjects

```
plt.title('No of Datapoints per Activity', fontsize=15)
sns.countplot(train.ActivityName)
plt.xticks(rotation=90)
plt.show()
```



## Observation

Our data is well balanced (almost)

## ✓ 4. Changing feature names

```
columns = train.columns

# Removing '()' from column names
columns = columns.str.replace('()', '')
```

```

columns = columns.str.replace('[-]', '')
columns = columns.str.replace('[,]', '')

train.columns = columns
test.columns = columns

test.columns

Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstdX',
      'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',
      'tBodyAccmadZ', 'tBodyAccmaxX',
      ...
      'angletBodyAccJerkMeangravityMean',
      'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',
      'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
      'subject', 'Activity', 'ActivityName'],
      dtype='object', length=564)

```

## 5. Save this dataframe in a csv files

```

train.to_csv('UCI_HAR_Dataset/csv_files/train.csv', index=False)
test.to_csv('UCI_HAR_Dataset/csv_files/test.csv', index=False)

```

Double-click (or enter) to edit

## Exploratory Data Analysis

*"Without domain knowledge EDA has no meaning, without EDA a problem has no soul."*

### 1. Featuring Engineering from Domain Knowledge

- **Static and Dynamic Activities**

- In static activities (sit, stand, lie down) motion information will not be very useful.
- In the dynamic activities (Walking, WalkingUpstairs, WalkingDownstairs) motion info will be significant.

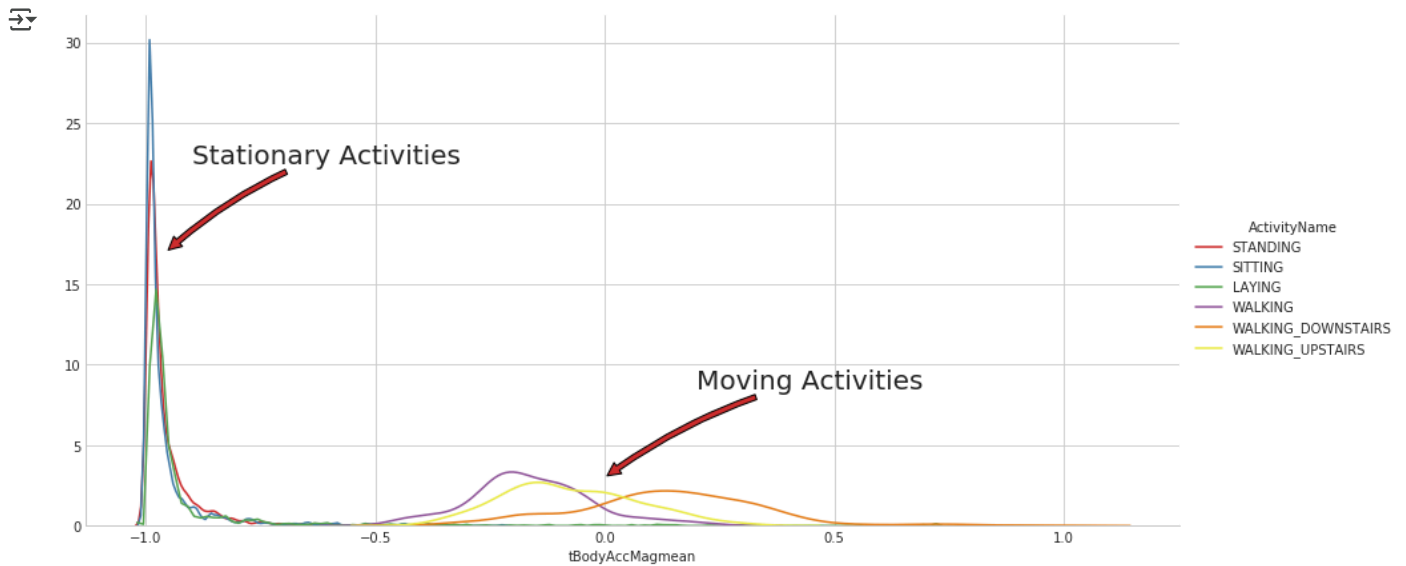
### 2. Stationary and Moving activities are completely different

```

sns.set_palette("Set1", desat=0.80)
facetgrid = sns.FacetGrid(train, hue='ActivityName', size=6, aspect=2)
facetgrid.map(sns.distplot, 'tBodyAccMagmean', hist=False)\
    .add_legend()
plt.annotate("Stationary Activities", xy=(-0.956, 17), xytext=(-0.9, 23), size=20,\
    va='center', ha='left',\
    arrowprops=dict(arrowstyle="simple", connectionstyle="arc3,rad=0.1"))

plt.annotate("Moving Activities", xy=(0, 3), xytext=(0.2, 9), size=20,\
    va='center', ha='left',\
    arrowprops=dict(arrowstyle="simple", connectionstyle="arc3,rad=0.1"))
plt.show()

```

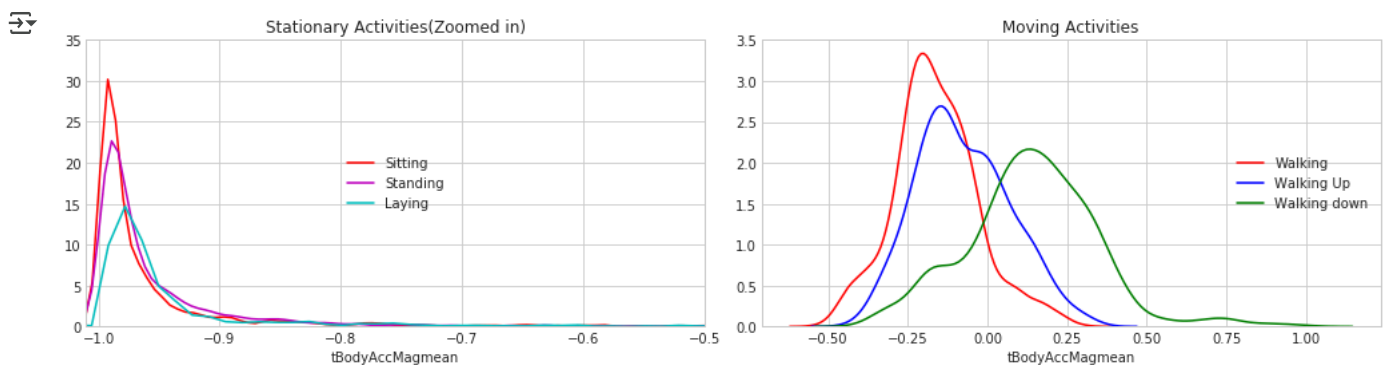


```
# for plotting purposes taking datapoints of each activity to a different dataframe
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]
```

```
plt.figure(figsize=(14,7))
plt.subplot(2,2,1)
plt.title('Stationary Activities(Zoomed in)')
sns.distplot(df4['tBodyAccMagmean'],color = 'r',hist = False, label = 'Sitting')
sns.distplot(df5['tBodyAccMagmean'],color = 'm',hist = False,label = 'Standing')
sns.distplot(df6['tBodyAccMagmean'],color = 'c',hist = False, label = 'Laying')
plt.axis([-1.01, -0.5, 0, 35])
plt.legend(loc='center')

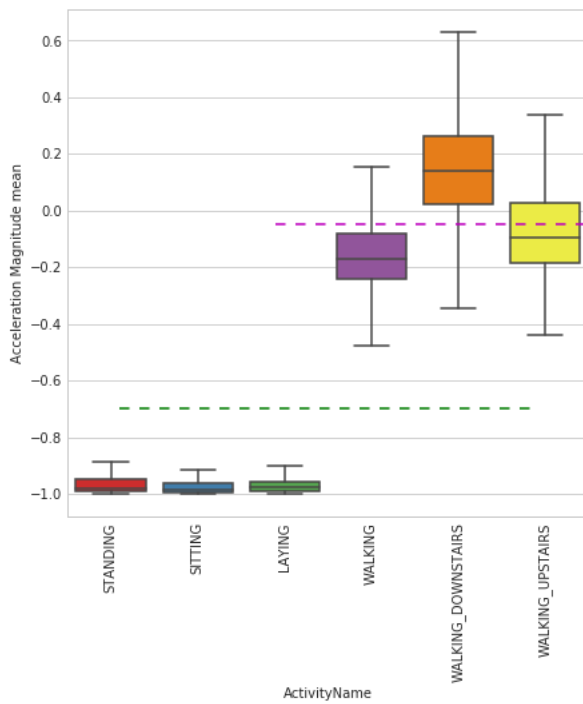
plt.subplot(2,2,2)
plt.title('Moving Activities')
sns.distplot(df1['tBodyAccMagmean'],color = 'red',hist = False, label = 'Walking')
sns.distplot(df2['tBodyAccMagmean'],color = 'blue',hist = False, label = 'Walking Up')
sns.distplot(df3['tBodyAccMagmean'],color = 'green',hist = False, label = 'Walking down')
plt.legend(loc='center right')
```

```
plt.tight_layout()
plt.show()
```



### ✓ 3. Magnitude of an acceleration can saperate it well

```
plt.figure(figsize=(7,7))
sns.boxplot(x='ActivityName', y='tBodyAccMagmean',data=train, showfliers=False, saturation=1)
plt.ylabel('Acceleration Magnitude mean')
plt.axhline(y=-0.7, xmin=0.1, xmax=0.9,dashes=(5,5), c='g')
plt.axhline(y=-0.05, xmin=0.4, dashes=(5,5), c='m')
plt.xticks(rotation=90)
plt.show()
```

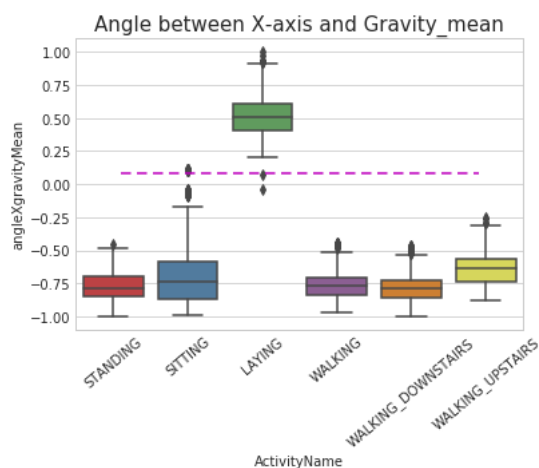


\_\_ Observations\_\_:

- If tAccMean is < -0.8 then the Activities are either Standing or Sitting or Laying.
- If tAccMean is > -0.6 then the Activities are either Walking or WalkingDownstairs or WalkingUpstairs.
- If tAccMean > 0.0 then the Activity is WalkingDownstairs.
- We can classify 75% the Activity labels with some errors.

#### ✓ 4. Position of GravityAccelerationComponents also matters

```
sns.boxplot(x='ActivityName', y='angleXgravityMean', data=train)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9, c='m', dashes=(5,3))
plt.title('Angle between X-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.show()
```

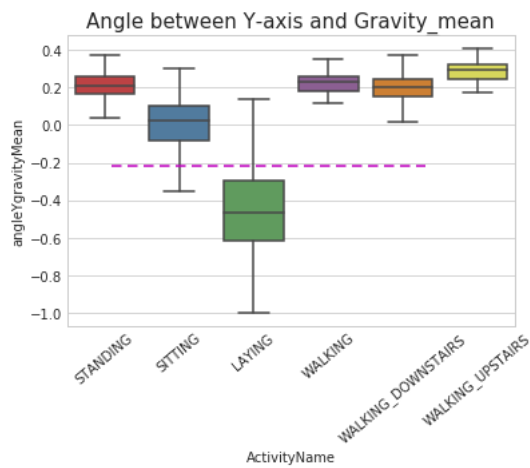


\_\_ Observations\_\_:

- If angleXgravityMean > 0 then Activity is Laying.
- We can classify all datapoints belonging to Laying activity with just a single if else statement.

```
sns.boxplot(x='ActivityName', y='angleYgravityMean', data = train, showliers=False)
plt.title('Angle between Y-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
plt.show()
```





Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

## ✓ Apply t-sne on the data

```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns

# performs t-sne with different perplexity values and their repective plots..

def perform_tsne(X_data, y_data, perplexities, n_iter=1000, img_name_prefix='t-sne'):

    for index,perplexity in enumerate(perplexities):
        # perform t-sne
        print('\nperforming tsne with perplexity {} and with {} iterations at max'.format(perplexity, n_iter))
        X_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transform(X_data)
        print('Done..')

        # prepare the data for seaborn
        print('Creating plot for this t-sne visualization..')
        df = pd.DataFrame({'x':X_reduced[:,0], 'y':X_reduced[:,1] , 'label':y_data})

        # draw the plot in appropriate place in the grid
        sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,\
                    palette="Set1",markers=['^','v','s','o', '1','2'])
        plt.title("perplexity : {} and max_iter : {}".format(perplexity, n_iter))
        img_name = img_name_prefix + '_perp_{}_iter_{}.png'.format(perplexity, n_iter)
        print('saving this plot as image in present working directory...')
        plt.savefig(img_name)
        plt.show()
        print('Done')

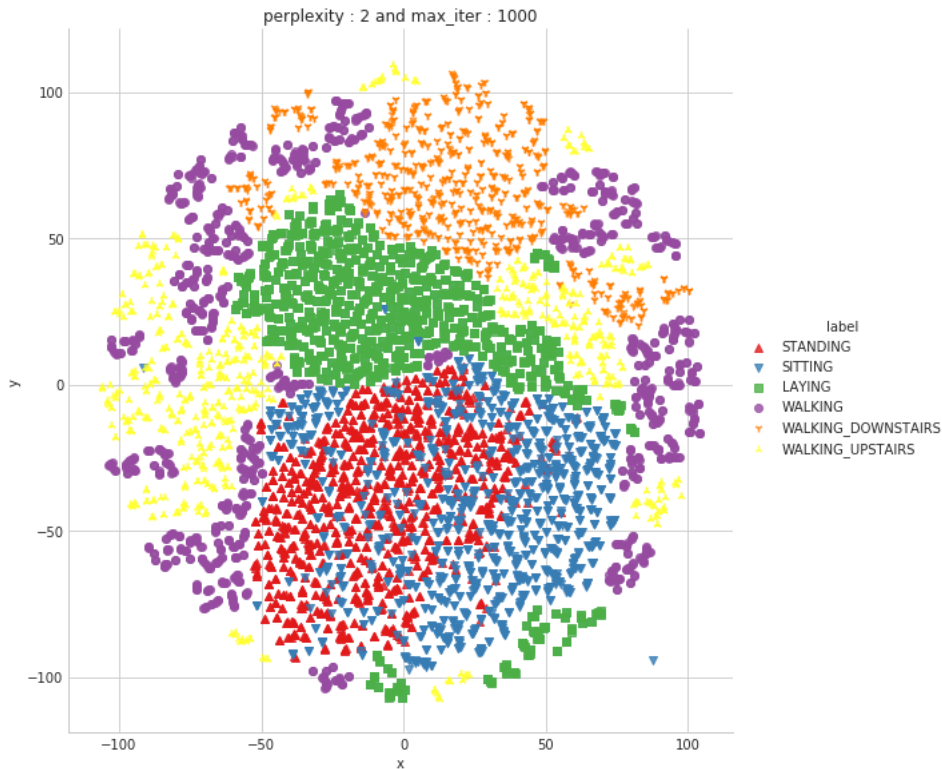
X_pre_tsne = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_pre_tsne = train['ActivityName']
perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[2,5,10,20,50])
```



```

performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.426s...
[t-SNE] Computed neighbors for 7352 samples in 72.001s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.635855
[t-SNE] Computed conditional probabilities in 0.071s
[t-SNE] Iteration 50: error = 124.8017578, gradient norm = 0.0253939 (50 iterations in 16.625s)
[t-SNE] Iteration 100: error = 107.2019501, gradient norm = 0.0284782 (50 iterations in 9.735s)
[t-SNE] Iteration 150: error = 100.9872894, gradient norm = 0.0185151 (50 iterations in 5.346s)
[t-SNE] Iteration 200: error = 97.6054382, gradient norm = 0.0142084 (50 iterations in 7.013s)
[t-SNE] Iteration 250: error = 95.3084183, gradient norm = 0.0132592 (50 iterations in 5.703s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.308418
[t-SNE] Iteration 300: error = 4.1209540, gradient norm = 0.0015668 (50 iterations in 7.156s)
[t-SNE] Iteration 350: error = 3.2113254, gradient norm = 0.0009953 (50 iterations in 8.022s)
[t-SNE] Iteration 400: error = 2.7819963, gradient norm = 0.0007203 (50 iterations in 9.419s)
[t-SNE] Iteration 450: error = 2.5178111, gradient norm = 0.0005655 (50 iterations in 9.370s)
[t-SNE] Iteration 500: error = 2.3341548, gradient norm = 0.0004804 (50 iterations in 7.681s)
[t-SNE] Iteration 550: error = 2.1961622, gradient norm = 0.0004183 (50 iterations in 7.097s)
[t-SNE] Iteration 600: error = 2.0867445, gradient norm = 0.0003664 (50 iterations in 9.274s)
[t-SNE] Iteration 650: error = 1.9967778, gradient norm = 0.0003279 (50 iterations in 7.697s)
[t-SNE] Iteration 700: error = 1.9210005, gradient norm = 0.0002984 (50 iterations in 8.174s)
[t-SNE] Iteration 750: error = 1.8558111, gradient norm = 0.0002776 (50 iterations in 9.747s)
[t-SNE] Iteration 800: error = 1.7989457, gradient norm = 0.0002569 (50 iterations in 8.687s)
[t-SNE] Iteration 850: error = 1.7490212, gradient norm = 0.0002394 (50 iterations in 8.407s)
[t-SNE] Iteration 900: error = 1.7043383, gradient norm = 0.0002224 (50 iterations in 8.351s)
[t-SNE] Iteration 950: error = 1.6641431, gradient norm = 0.0002098 (50 iterations in 7.841s)
[t-SNE] Iteration 1000: error = 1.6279151, gradient norm = 0.0001989 (50 iterations in 5.623s)
[t-SNE] Error after 1000 iterations: 1.627915
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...

```



Done

```

performing tsne with perplexity 5 and with 1000 iterations at max
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.263s...
[t-SNE] Computed neighbors for 7352 samples in 48.983s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.961265
[t-SNE] Computed conditional probabilities in 0.122s
[t-SNE] Iteration 50: error = 114.1863440, gradient norm = 0.0184130 (50 iterations in 55.656s)

```