HAR_EDA

March 16, 2019

1 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.

1.1 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.

1.1.1 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 obtianed by calculating variables from the time and frequency domain. > In our dataset, each datapoint represents a window with different readings

3. The accelertion signal was saperated into Body and Gravity acceleration signals(***tBodyAcc-XYZ*** and ***tGravityAcc-XYZ***) using some low pass filter with corner frequecy of 0.3Hz.

4. After that, the body linear acceleration and angular velocity were derived in time to obtian *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 esitmate some set of variables from the above signals. ie., We will estimate the following properties on each and every signal that we recoreded 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

1.1.2 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**

1.2 Train and test data were saperated

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

1.3 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'

1.4 Data Size :

27 MB

2 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.

2.1 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.

2.2 Problem Statement

- Given a new datapoint we have to predict the Activity

```
In [1]: 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
```

```
In [3]: data=pd.read_csv('UCI_HAR_dataset/train/X_train.txt', delim_whitespace=True, header=No
        data.shape
```

```
Out[3]: (7352, 561)
```

In [17]:

Out[17]: (564,)

In [ ]:

In [ ]:

In [5]:
```
data.head(4)
```

Out[5]:
```
   tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBodyAcc-std()-X  \
0           0.288585          -0.020294          -0.132905         -0.995279
1           0.278419          -0.016411          -0.123520         -0.998245
2           0.279653          -0.019467          -0.113462         -0.995380
3           0.279174          -0.026201          -0.123283         -0.996091

   tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAcc-mad()-Y  \
0         -0.983111         -0.913526         -0.995112         -0.983185
1         -0.975300         -0.960322         -0.998807         -0.974914
2         -0.967187         -0.978944         -0.996520         -0.963668
3         -0.983403         -0.990675         -0.997099         -0.982750

   tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  fBodyBodyGyroJerkMag-meanFreq()  \
0         -0.923527         -0.934724  ...                        -0.074323
1         -0.957686         -0.943068  ...                         0.158075
2         -0.977469         -0.938692  ...                         0.414503
3         -0.989302         -0.938692  ...                         0.404573

   fBodyBodyGyroJerkMag-skewness()  fBodyBodyGyroJerkMag-kurtosis()  \
0                        -0.298676                        -0.710304
1                        -0.595051                        -0.861499
2                        -0.390748                        -0.760104
3                        -0.117290                        -0.482845

   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

   angle(tBodyGyroMean,gravityMean)  angle(tBodyGyroJerkMean,gravityMean)  \
0                         -0.464761                             -0.018446
1                         -0.732626                              0.703511
2                          0.100699                              0.808529
3                          0.640011                             -0.485366

   angle(X,gravityMean)  angle(Y,gravityMean)  angle(Z,gravityMean)
0             -0.841247              0.179941             -0.058627
1             -0.844788              0.180289             -0.054317
```

```
2              -0.848933              0.180637              -0.049118
3              -0.848649              0.181935              -0.047663

[4 rows x 561 columns]
```

2.3 Obtain the train data

In [18]:
```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv('UCI_HAR_dataset/train/X_train.txt', delim_whitespace=True, head

# add subject column to the dataframe
X_train['subject'] = pd.read_csv('UCI_HAR_dataset/train/subject_train.txt', header=Nor

y_train = pd.read_csv('UCI_HAR_dataset/train/y_train.txt', names=['Activity'], squeeze
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()
```

Out[18]:
```
     tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  \
784            0.18082          -0.002281          -0.072974

     tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  \
784         -0.249202           0.10667          -0.49802         -0.265213

     tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  \
784           0.05835         -0.455868         -0.169506  ...

     angle(tBodyAccMean,gravity)  angle(tBodyAccJerkMean),gravityMean)  \
784                     0.653324                              -0.87969

     angle(tBodyGyroMean,gravityMean)  angle(tBodyGyroJerkMean,gravityMean)  \
784                          0.946132                             -0.790122

     angle(X,gravityMean)  angle(Y,gravityMean)  angle(Z,gravityMean)  \
784             -0.936894              0.139184              0.033872

     subject  Activity  ActivityName
784        5         1       WALKING

[1 rows x 564 columns]
```

In [3]:
```
train.shape
```

Out[3]:
```
(7352, 564)
```

2.4 Obtain the test data

In [19]:
```
# 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=

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

# get y labels from the txt file
y_test = pd.read_csv('UCI_HAR_dataset/test/y_test.txt', names=['Activity'], squeeze=T
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()
```

Out[19]:
```
      tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  \
1273           0.223884          -0.016198          -0.110829

      tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  \
1273         -0.966023         -0.970969         -0.995167         -0.969574

      tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  \
1273          -0.97103         -0.994497         -0.918352  ...

      angle(tBodyAccMean,gravity)  angle(tBodyAccJerkMean),gravityMean)  \
1273                     0.016828                              0.500849

      angle(tBodyGyroMean,gravityMean)  angle(tBodyGyroJerkMean,gravityMean)  \
1273                          0.259041                              0.219003

      angle(X,gravityMean)  angle(Y,gravityMean)  angle(Z,gravityMean)  \
1273              0.602713             -0.413909             -0.596575

      subject  Activity  ActivityName
1273       12         6        LAYING

[1 rows x 564 columns]
```

In [20]:
```
test.shape
```

Out[20]:
```
(2947, 564)
```

3 Data Cleaning

3.1 1. Check for Duplicates

```
In [21]: 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
```

3.2 2. Checking for NaN/null values

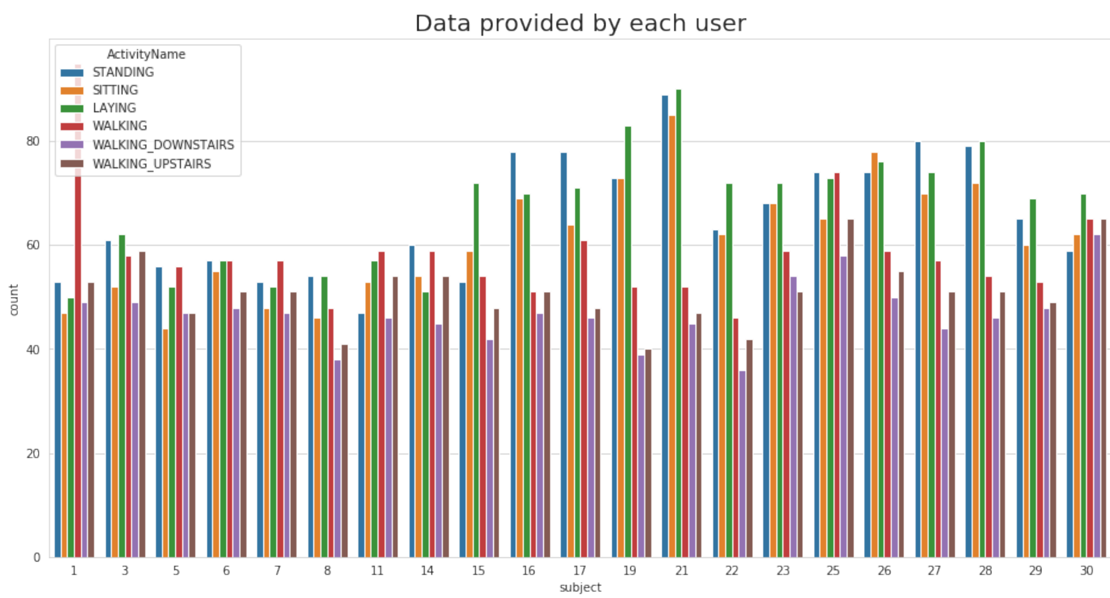
```
In [22]: print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))
         print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```

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

3.3 3. Check for data imbalance

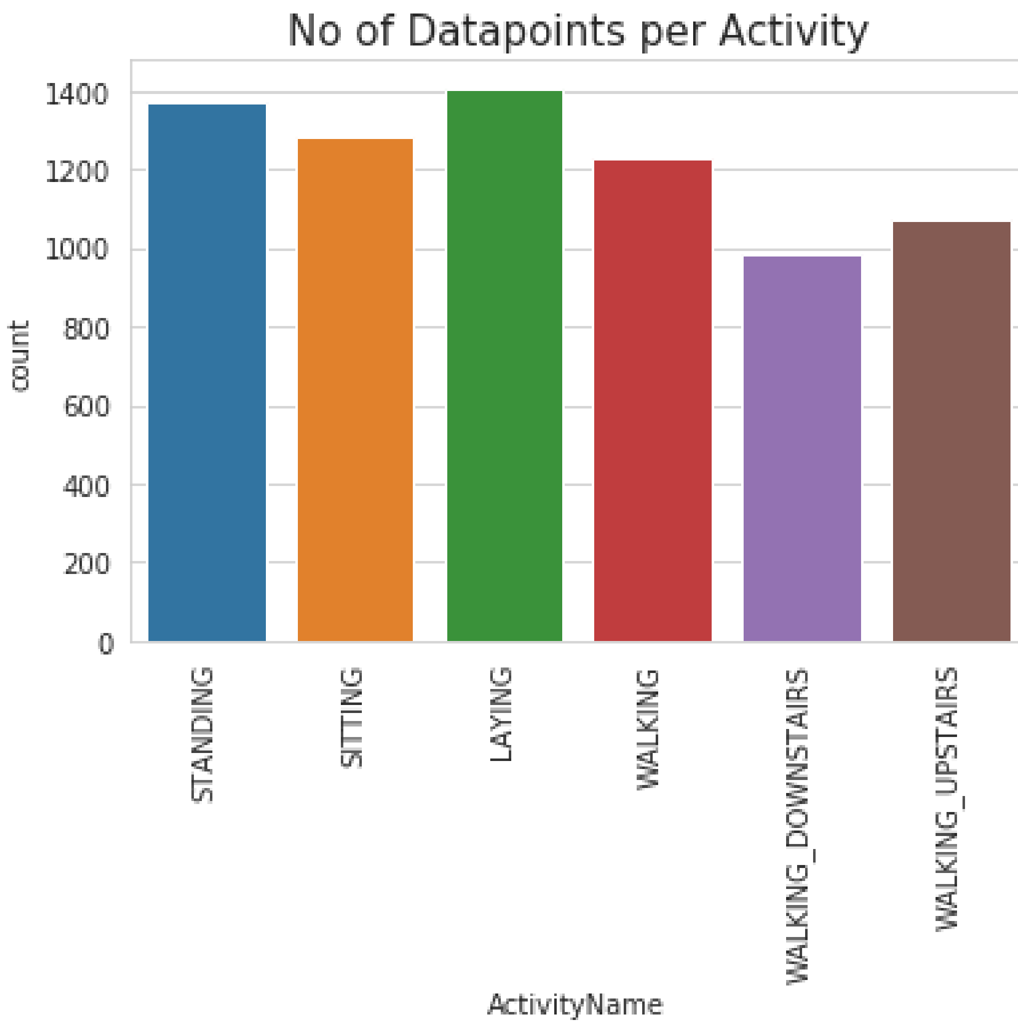
```
In [23]: import matplotlib.pyplot as plt
         import seaborn as sns

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

```
In [24]: 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

```
In [25]: plt.title('No of Datapoints per Activity', fontsize=15)
         sns.countplot(train.ActivityName)
         plt.xticks(rotation=90)
         plt.show()
```



3.3.1 Observation

Our data is well balanced (almost)

3.4 4. Changing feature names

```
In [26]: 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
```

```
Out[26]: Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstdX',
                'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',
                'tBodyAccmadZ', 'tBodyAccmaxX',
                ...
                'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',
                'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',
                'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
                'subject', 'Activity', 'ActivityName'],
               dtype='object', length=564)
```

3.5 5. Save this dataframe in a csv files

```
In [27]: train.to_csv('UCI_HAR_Dataset/csv_files/train.csv', index=False)
         test.to_csv('UCI_HAR_Dataset/csv_files/test.csv', index=False)
```

4 Exploratory Data Analysis

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

4.0.1 1. Featuring Engineering from Domain Knowledge

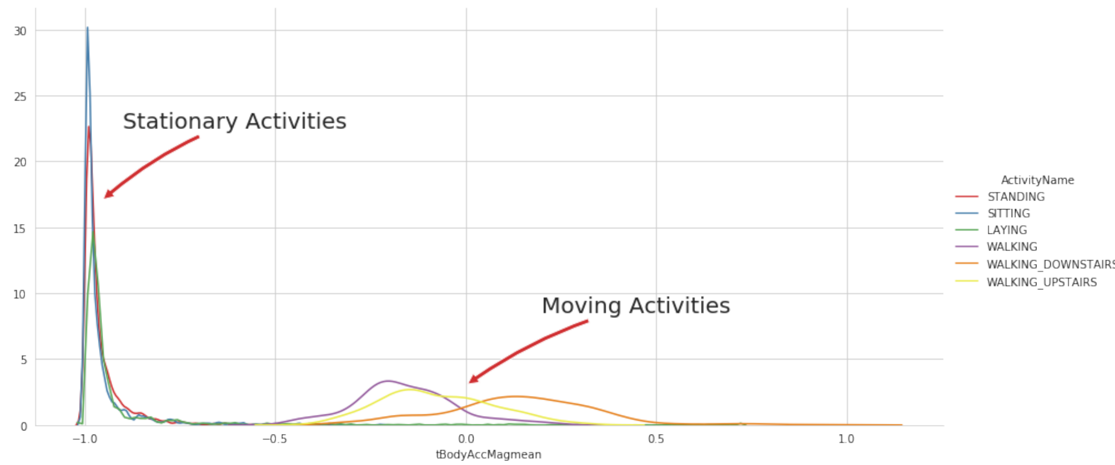
- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

4.0.2 2. Stationary and Moving activities are completely different

```
In [28]: 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',\
```

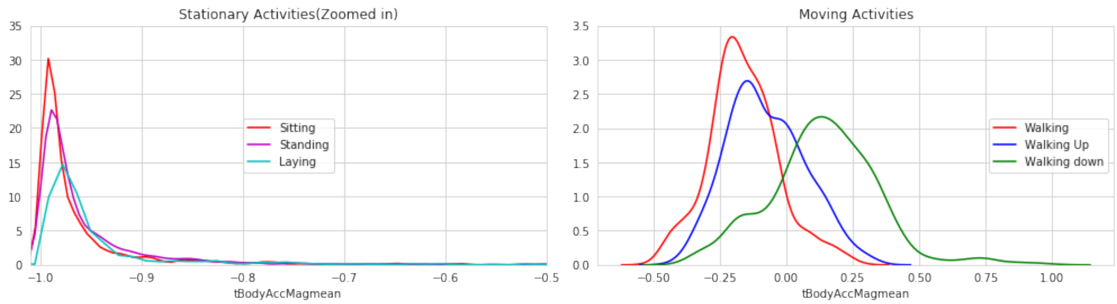
```
                    arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))

        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()
```

```
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\seaborn\axisgrid.py:230
  warnings.warn(msg, UserWarning)
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\scipy\stats\stats.py:171
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



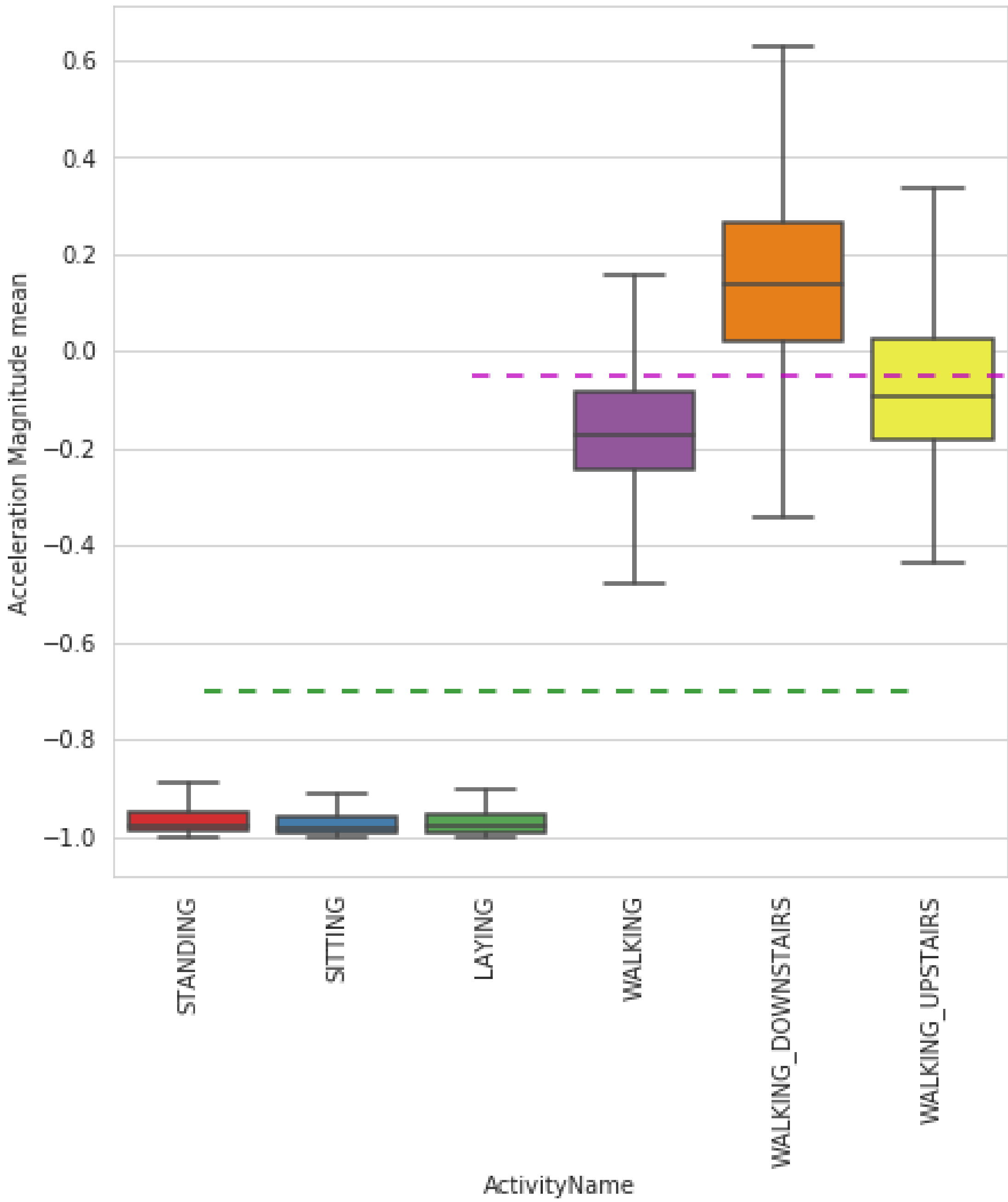
In [29]:
```
# for plotting purposes taking datapoints of each activity to a different dataframe
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]

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 dow
plt.legend(loc='center right')


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



4.0.3 3. Magnitude of an acceleration can saperate it well

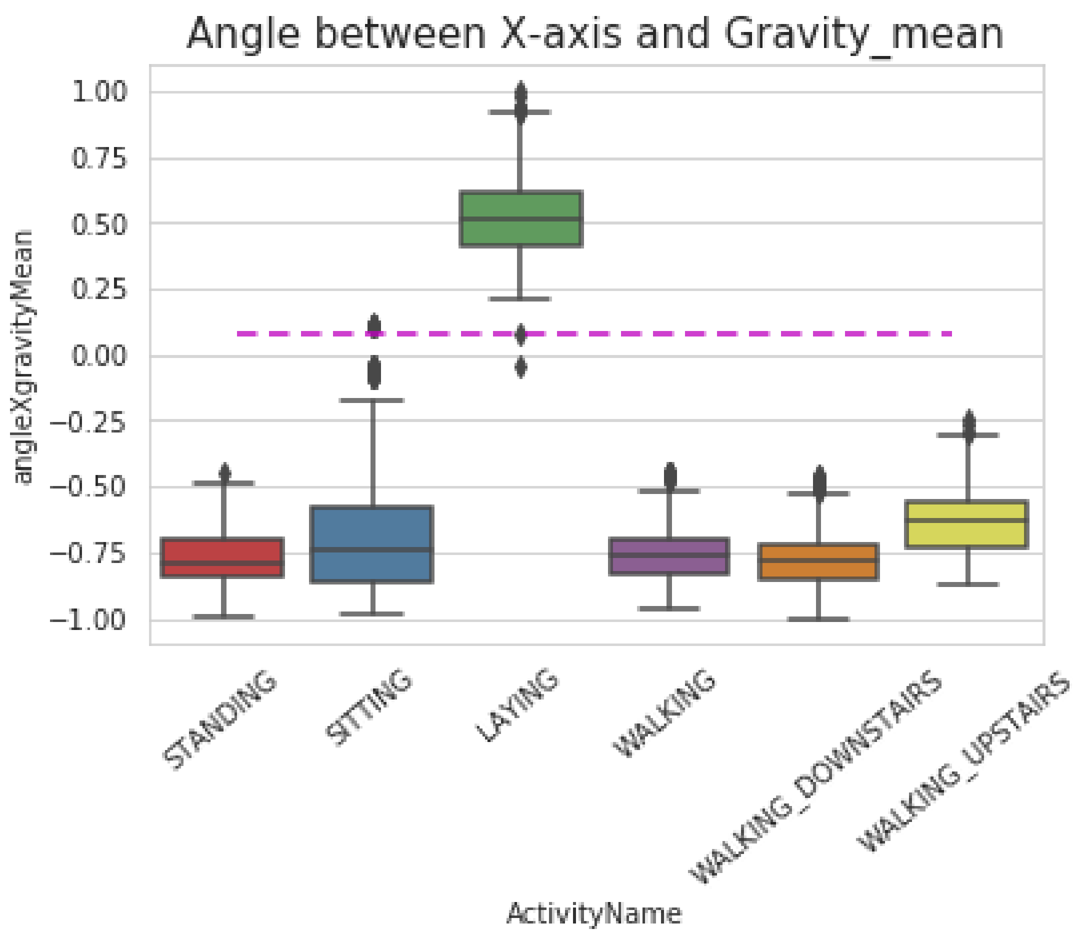
```
In [30]: plt.figure(figsize=(7,7))
         sns.boxplot(x='ActivityName', y='tBodyAccMagmean',data=train, showfliers=False, satura
         plt.ylabel('Acceleration Magnitude mean')
         plt.axhline(y=-0.7, xmin=0.1, xmax=0.9,dashes=(5,5), c='g')
         plt.axhline(y=-0.05, xmin=0.4, dashes=(5,5), 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 Acitivity labels with some errors.

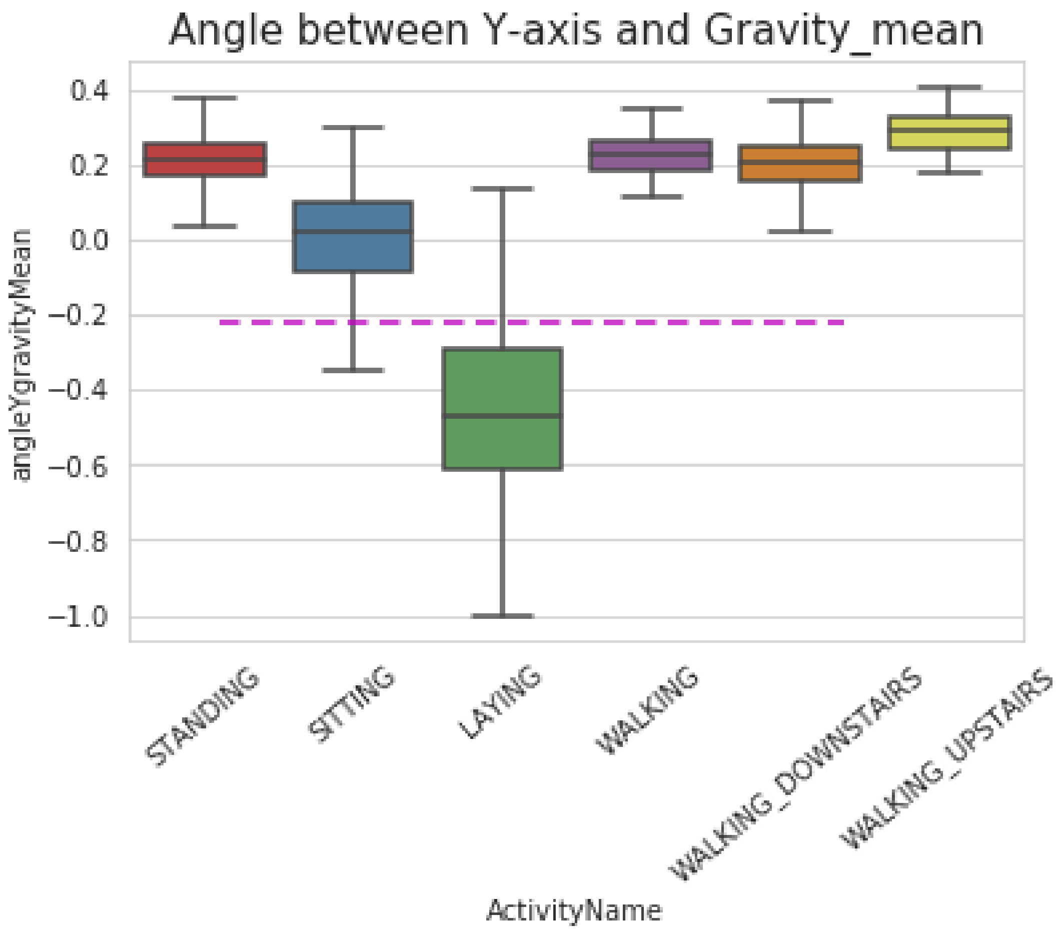
4.0.4 4. Position of GravityAccelerationComponants also matters

In [31]:
```
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 angleX,gravityMean > 0 then Activity is Laying. * We can classify all datapoints belonging to Laying activity with just a single if else statement.

In [32]:
```
sns.boxplot(x='ActivityName', y='angleYgravityMean', data = train, showfliers=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()
```



5 Apply t-sne on the data

In [33]:
```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns
```

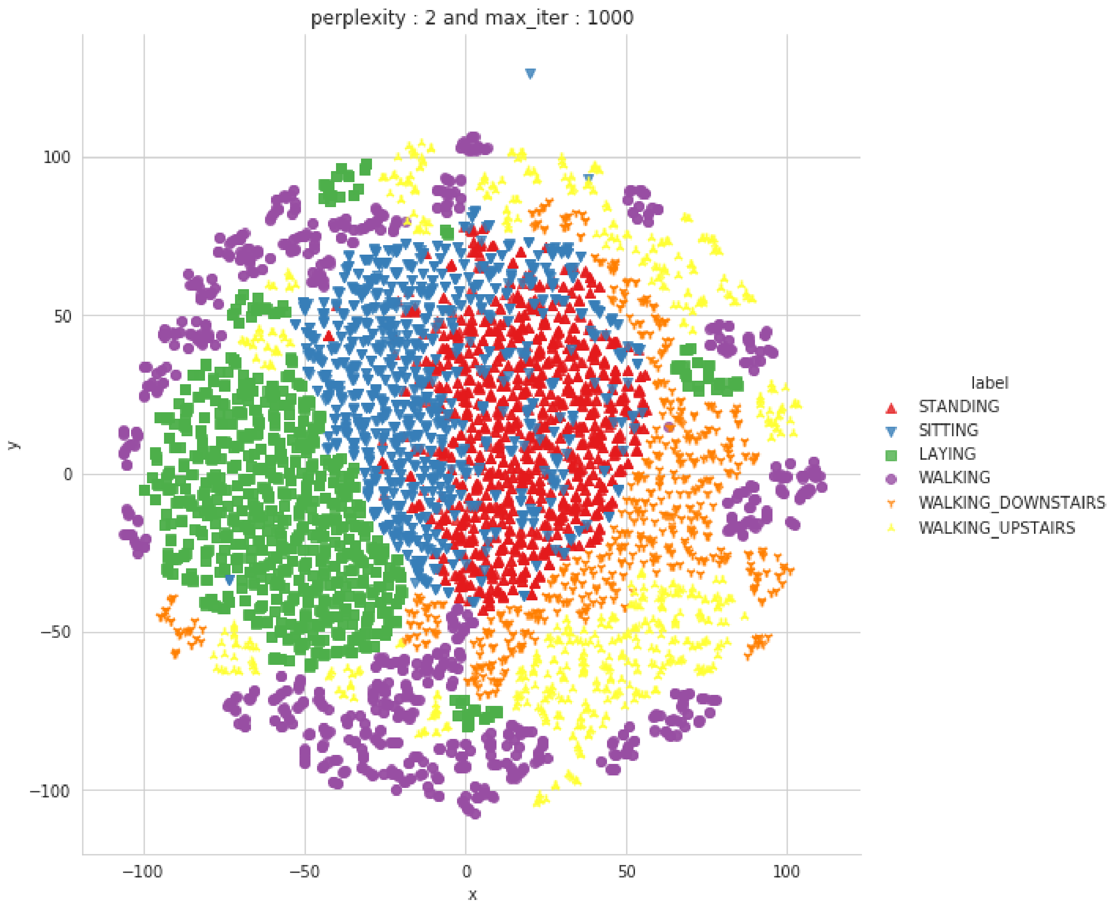
In [34]:
```
# performs t-sne with different perplexity values and their repective plots..

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'.fo
        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')
```

```
In [35]: 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 0.683s...
[t-SNE] Computed neighbors for 7352 samples in 34.777s...
[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 1.428s
[t-SNE] Iteration 50: error = 124.7093124, gradient norm = 0.0298898 (50 iterations in 13.299s
[t-SNE] Iteration 100: error = 107.0696716, gradient norm = 0.0300952 (50 iterations in 3.791s
[t-SNE] Iteration 150: error = 100.7640762, gradient norm = 0.0188750 (50 iterations in 2.655s
[t-SNE] Iteration 200: error = 97.4186554, gradient norm = 0.0156425 (50 iterations in 2.487s)
[t-SNE] Iteration 250: error = 95.1398087, gradient norm = 0.0141706 (50 iterations in 2.363s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.139809
[t-SNE] Iteration 300: error = 4.1169271, gradient norm = 0.0015617 (50 iterations in 2.227s)
[t-SNE] Iteration 350: error = 3.2087421, gradient norm = 0.0010064 (50 iterations in 1.988s)
[t-SNE] Iteration 400: error = 2.7794907, gradient norm = 0.0007180 (50 iterations in 2.052s)
[t-SNE] Iteration 450: error = 2.5153465, gradient norm = 0.0005748 (50 iterations in 2.074s)
[t-SNE] Iteration 500: error = 2.3325195, gradient norm = 0.0004788 (50 iterations in 2.102s)
[t-SNE] Iteration 550: error = 2.1948435, gradient norm = 0.0004139 (50 iterations in 2.167s)
[t-SNE] Iteration 600: error = 2.0854611, gradient norm = 0.0003656 (50 iterations in 2.160s)
[t-SNE] Iteration 650: error = 1.9956354, gradient norm = 0.0003303 (50 iterations in 2.141s)
[t-SNE] Iteration 700: error = 1.9197433, gradient norm = 0.0003057 (50 iterations in 2.157s)
```

```
[t-SNE] Iteration 750: error = 1.8549173, gradient norm = 0.0002774 (50 iterations in 2.239s)
[t-SNE] Iteration 800: error = 1.7982968, gradient norm = 0.0002593 (50 iterations in 2.324s)
[t-SNE] Iteration 850: error = 1.7484928, gradient norm = 0.0002401 (50 iterations in 2.253s)
[t-SNE] Iteration 900: error = 1.7037975, gradient norm = 0.0002273 (50 iterations in 2.260s)
[t-SNE] Iteration 950: error = 1.6636121, gradient norm = 0.0002126 (50 iterations in 2.274s)
[t-SNE] Iteration 1000: error = 1.6271712, gradient norm = 0.0002006 (50 iterations in 2.403s)
[t-SNE] KL divergence after 1000 iterations: 1.627171
Done..
Creating plot for this t-sne visualization..


c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:5
  warnings.warn(msg, UserWarning)


saving this plot as image in present working directory...
```

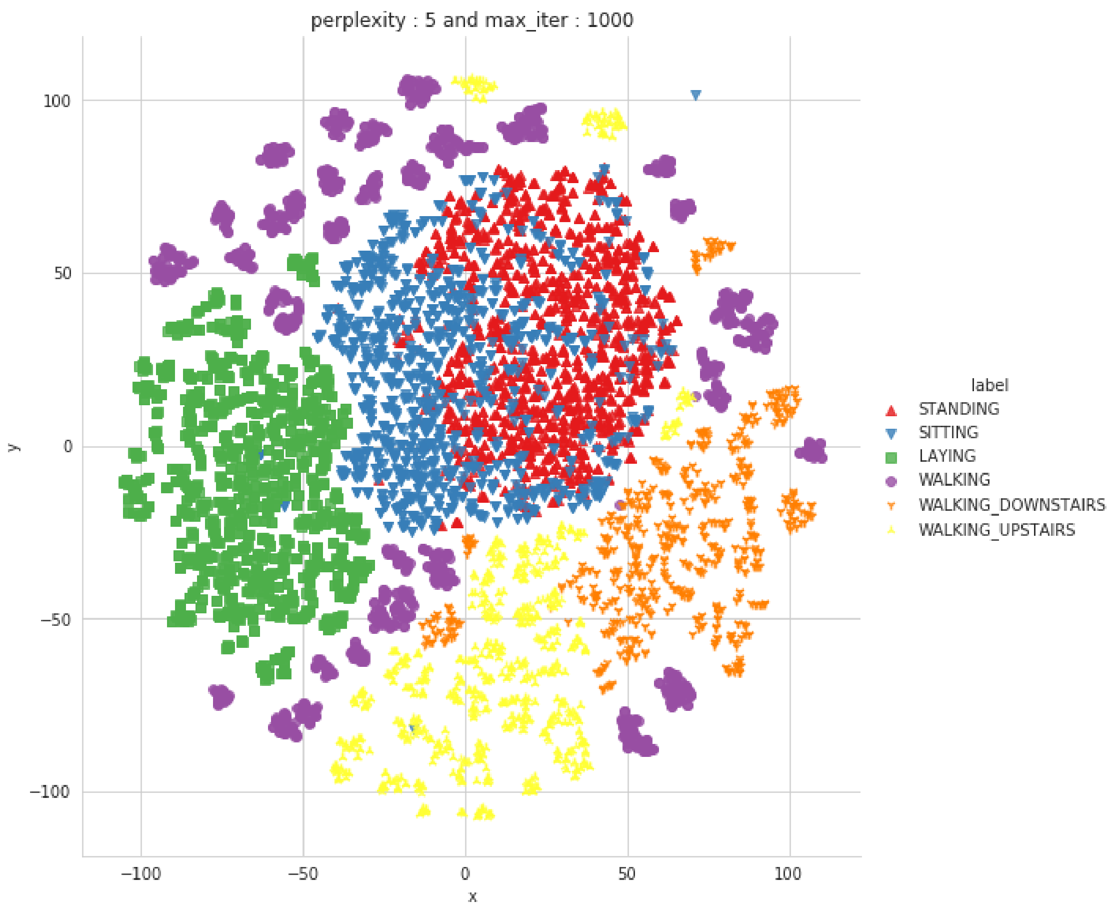



```
Done
```

```
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 36.633s...
[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 0.046s
[t-SNE] Iteration 50: error = 114.0588455, gradient norm = 0.0204286 (50 iterations in 13.314s
[t-SNE] Iteration 100: error = 97.3685379, gradient norm = 0.0153800 (50 iterations in 2.791s)
[t-SNE] Iteration 150: error = 93.0875320, gradient norm = 0.0081664 (50 iterations in 2.172s)
[t-SNE] Iteration 200: error = 91.0965881, gradient norm = 0.0063155 (50 iterations in 2.129s)
[t-SNE] Iteration 250: error = 89.9250259, gradient norm = 0.0051821 (50 iterations in 2.221s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 89.925026
[t-SNE] Iteration 300: error = 3.5678382, gradient norm = 0.0014579 (50 iterations in 2.052s)
[t-SNE] Iteration 350: error = 2.8112514, gradient norm = 0.0007469 (50 iterations in 2.115s)
[t-SNE] Iteration 400: error = 2.4308279, gradient norm = 0.0005256 (50 iterations in 2.145s)
[t-SNE] Iteration 450: error = 2.2140884, gradient norm = 0.0004070 (50 iterations in 2.114s)
[t-SNE] Iteration 500: error = 2.0699861, gradient norm = 0.0003323 (50 iterations in 2.123s)
[t-SNE] Iteration 550: error = 1.9651834, gradient norm = 0.0002803 (50 iterations in 2.247s)
[t-SNE] Iteration 600: error = 1.8841531, gradient norm = 0.0002474 (50 iterations in 2.164s)
[t-SNE] Iteration 650: error = 1.8188921, gradient norm = 0.0002234 (50 iterations in 2.180s)
[t-SNE] Iteration 700: error = 1.7650652, gradient norm = 0.0002000 (50 iterations in 2.126s)
[t-SNE] Iteration 750: error = 1.7196234, gradient norm = 0.0001805 (50 iterations in 2.120s)
[t-SNE] Iteration 800: error = 1.6806608, gradient norm = 0.0001651 (50 iterations in 2.147s)
[t-SNE] Iteration 850: error = 1.6467364, gradient norm = 0.0001556 (50 iterations in 2.127s)
[t-SNE] Iteration 900: error = 1.6169364, gradient norm = 0.0001427 (50 iterations in 2.120s)
[t-SNE] Iteration 950: error = 1.5905139, gradient norm = 0.0001342 (50 iterations in 2.135s)
[t-SNE] Iteration 1000: error = 1.5668000, gradient norm = 0.0001274 (50 iterations in 2.129s)
[t-SNE] KL divergence after 1000 iterations: 1.566800
Done..
Creating plot for this t-sne visualization..
```

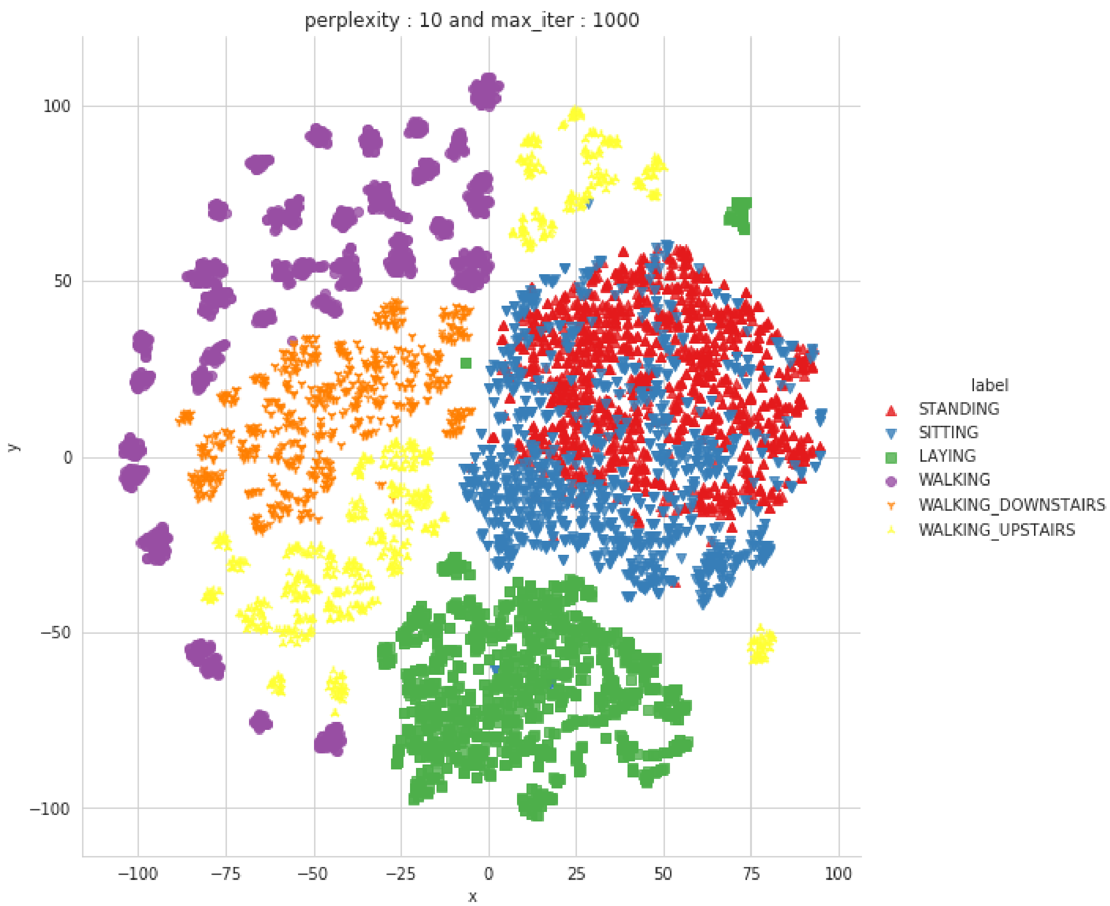
```
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:5
  warnings.warn(msg, UserWarning)
```

```
saving this plot as image in present working directory...
```




```
Done

performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.215s...
[t-SNE] Computed neighbors for 7352 samples in 89.354s...
[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: 1.133828
[t-SNE] Computed conditional probabilities in 0.087s
[t-SNE] Iteration 50: error = 105.8209076, gradient norm = 0.0197871 (50 iterations in 4.039s)
[t-SNE] Iteration 100: error = 90.2392960, gradient norm = 0.0115153 (50 iterations in 2.773s)
[t-SNE] Iteration 150: error = 87.2182007, gradient norm = 0.0052304 (50 iterations in 2.423s)
```

```
[t-SNE] Iteration 200: error = 86.0040054, gradient norm = 0.0040429 (50 iterations in 2.377s)
[t-SNE] Iteration 250: error = 85.3215866, gradient norm = 0.0039597 (50 iterations in 2.371s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.321587
[t-SNE] Iteration 300: error = 3.1310625, gradient norm = 0.0013930 (50 iterations in 2.296s)
[t-SNE] Iteration 350: error = 2.4868283, gradient norm = 0.0006485 (50 iterations in 2.242s)
[t-SNE] Iteration 400: error = 2.1667283, gradient norm = 0.0004216 (50 iterations in 2.494s)
[t-SNE] Iteration 450: error = 1.9824675, gradient norm = 0.0003127 (50 iterations in 2.663s)
[t-SNE] Iteration 500: error = 1.8641733, gradient norm = 0.0002521 (50 iterations in 2.747s)
[t-SNE] Iteration 550: error = 1.7808340, gradient norm = 0.0002116 (50 iterations in 2.904s)
[t-SNE] Iteration 600: error = 1.7183496, gradient norm = 0.0001820 (50 iterations in 2.919s)
[t-SNE] Iteration 650: error = 1.6696091, gradient norm = 0.0001593 (50 iterations in 2.555s)
[t-SNE] Iteration 700: error = 1.6300225, gradient norm = 0.0001441 (50 iterations in 2.391s)
[t-SNE] Iteration 750: error = 1.5975467, gradient norm = 0.0001287 (50 iterations in 2.382s)
[t-SNE] Iteration 800: error = 1.5702302, gradient norm = 0.0001163 (50 iterations in 2.332s)
[t-SNE] Iteration 850: error = 1.5469478, gradient norm = 0.0001075 (50 iterations in 2.326s)
[t-SNE] Iteration 900: error = 1.5268013, gradient norm = 0.0001012 (50 iterations in 2.338s)
[t-SNE] Iteration 950: error = 1.5091782, gradient norm = 0.0000946 (50 iterations in 2.453s)
[t-SNE] Iteration 1000: error = 1.4939352, gradient norm = 0.0000908 (50 iterations in 2.356s)
[t-SNE] KL divergence after 1000 iterations: 1.493935
Done..
Creating plot for this t-sne visualization..


c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:5
  warnings.warn(msg, UserWarning)


saving this plot as image in present working directory...
```

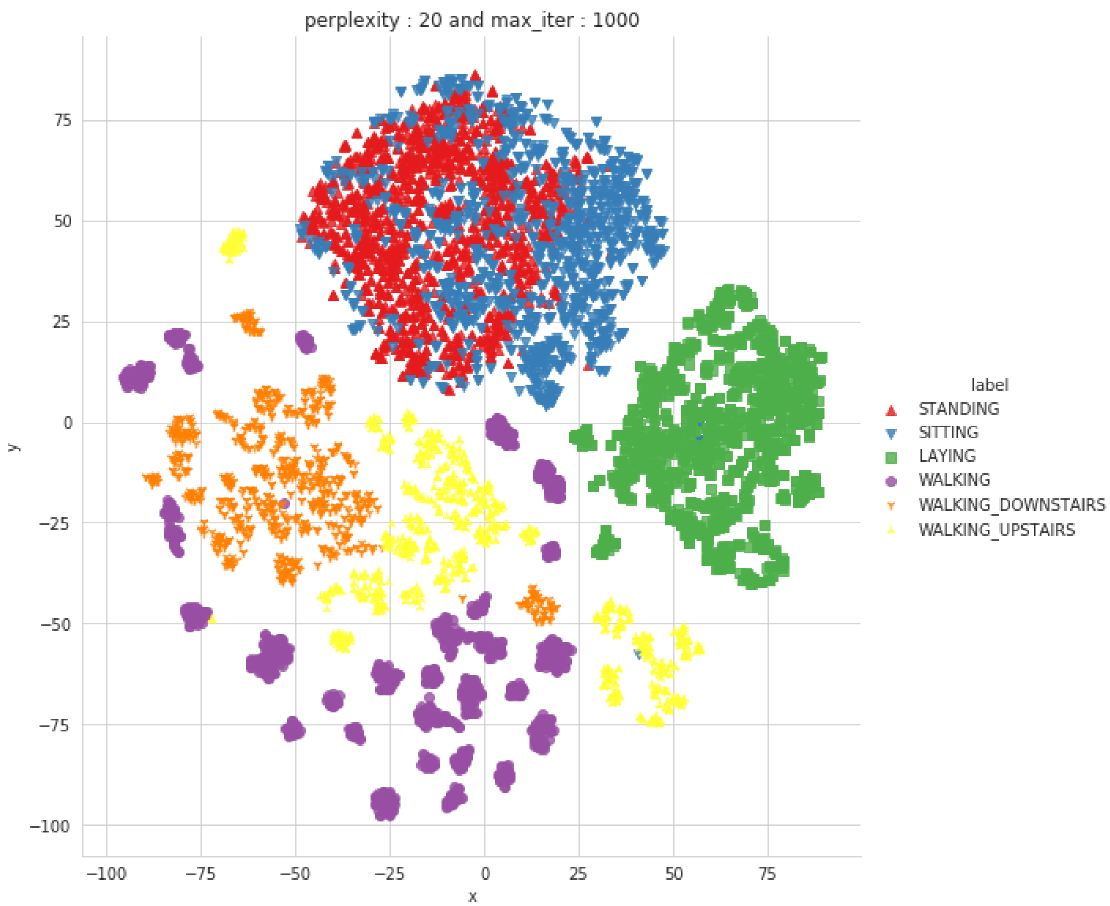



```
Done

performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.245s...
[t-SNE] Computed neighbors for 7352 samples in 38.396s...
[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: 1.274335
[t-SNE] Computed conditional probabilities in 0.166s
[t-SNE] Iteration 50: error = 97.6708450, gradient norm = 0.0176280 (50 iterations in 5.906s)
[t-SNE] Iteration 100: error = 84.3800354, gradient norm = 0.0063340 (50 iterations in 2.821s)
[t-SNE] Iteration 150: error = 82.0177536, gradient norm = 0.0036729 (50 iterations in 2.578s)
```

```
[t-SNE] Iteration 200: error = 81.2173843, gradient norm = 0.0026996 (50 iterations in 2.484s)
[t-SNE] Iteration 250: error = 80.8109741, gradient norm = 0.0019187 (50 iterations in 2.448s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.810974
[t-SNE] Iteration 300: error = 2.6923869, gradient norm = 0.0012901 (50 iterations in 2.383s)
[t-SNE] Iteration 350: error = 2.1612105, gradient norm = 0.0005702 (50 iterations in 2.332s)
[t-SNE] Iteration 400: error = 1.9129137, gradient norm = 0.0003478 (50 iterations in 2.335s)
[t-SNE] Iteration 450: error = 1.7666686, gradient norm = 0.0002471 (50 iterations in 2.380s)
[t-SNE] Iteration 500: error = 1.6727108, gradient norm = 0.0001924 (50 iterations in 2.451s)
[t-SNE] Iteration 550: error = 1.6085465, gradient norm = 0.0001583 (50 iterations in 2.481s)
[t-SNE] Iteration 600: error = 1.5619932, gradient norm = 0.0001363 (50 iterations in 2.459s)
[t-SNE] Iteration 650: error = 1.5270989, gradient norm = 0.0001192 (50 iterations in 2.401s)
[t-SNE] Iteration 700: error = 1.5000280, gradient norm = 0.0001065 (50 iterations in 2.506s)
[t-SNE] Iteration 750: error = 1.4784101, gradient norm = 0.0000985 (50 iterations in 2.378s)
[t-SNE] Iteration 800: error = 1.4609820, gradient norm = 0.0000901 (50 iterations in 2.383s)
[t-SNE] Iteration 850: error = 1.4464401, gradient norm = 0.0000824 (50 iterations in 2.381s)
[t-SNE] Iteration 900: error = 1.4338733, gradient norm = 0.0000782 (50 iterations in 2.418s)
[t-SNE] Iteration 950: error = 1.4233242, gradient norm = 0.0000755 (50 iterations in 2.533s)
[t-SNE] Iteration 1000: error = 1.4146587, gradient norm = 0.0000724 (50 iterations in 2.396s)
[t-SNE] KL divergence after 1000 iterations: 1.414659
Done..
Creating plot for this t-sne visualization..
```

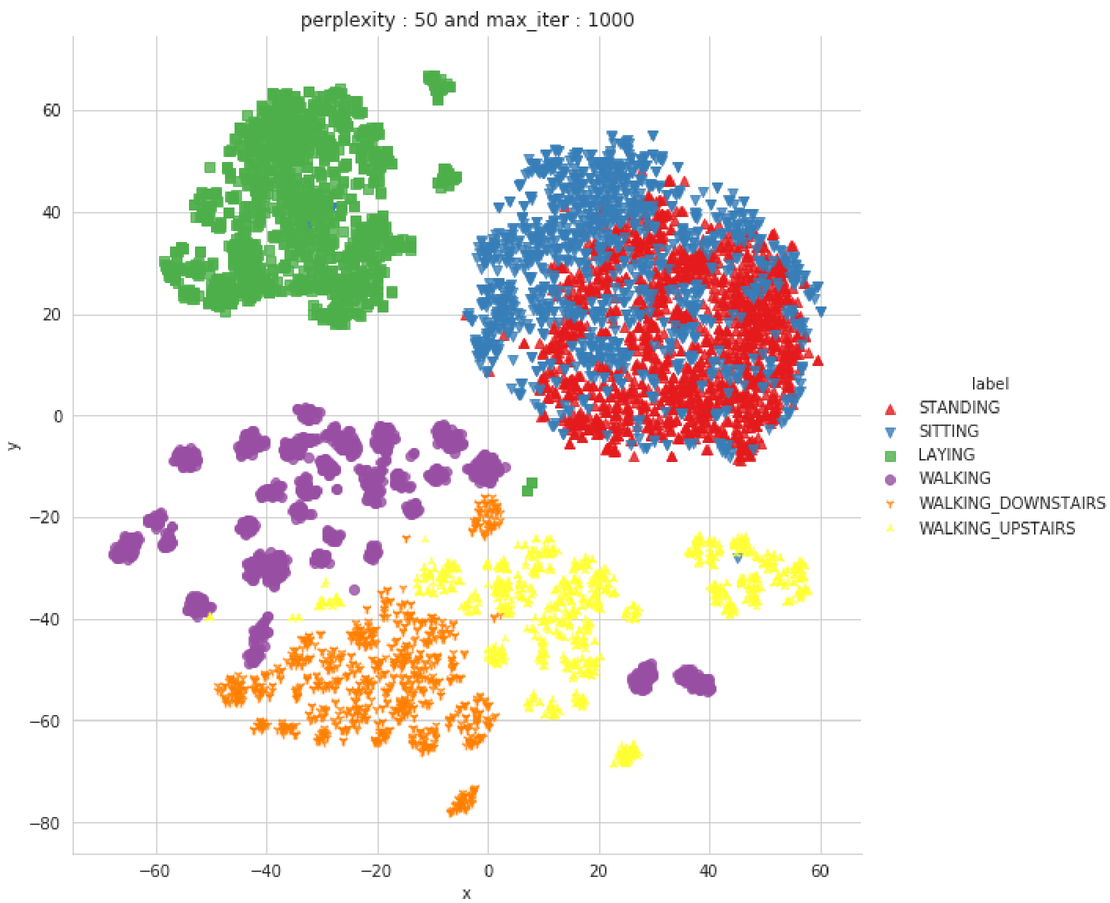
```
c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:5
  warnings.warn(msg, UserWarning)
```

```
saving this plot as image in present working directory...
```




```
Done

performing tsne with perplexity 50 and with 1000 iterations at max
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.211s...
[t-SNE] Computed neighbors for 7352 samples in 38.269s...
[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: 1.437672
[t-SNE] Computed conditional probabilities in 0.388s
[t-SNE] Iteration 50: error = 85.2197418, gradient norm = 0.0320484 (50 iterations in 7.086s)
[t-SNE] Iteration 100: error = 75.8540802, gradient norm = 0.0045513 (50 iterations in 6.271s)
[t-SNE] Iteration 150: error = 74.7807770, gradient norm = 0.0034706 (50 iterations in 4.688s)
```

```
[t-SNE] Iteration 200: error = 74.3307114, gradient norm = 0.0015979 (50 iterations in 4.536s)
[t-SNE] Iteration 250: error = 74.1124420, gradient norm = 0.0012009 (50 iterations in 4.448s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.112442
[t-SNE] Iteration 300: error = 2.1579430, gradient norm = 0.0011955 (50 iterations in 3.603s)
[t-SNE] Iteration 350: error = 1.7620761, gradient norm = 0.0004855 (50 iterations in 3.159s)
[t-SNE] Iteration 400: error = 1.5937347, gradient norm = 0.0002800 (50 iterations in 3.098s)
[t-SNE] Iteration 450: error = 1.4997300, gradient norm = 0.0001932 (50 iterations in 3.114s)
[t-SNE] Iteration 500: error = 1.4395635, gradient norm = 0.0001440 (50 iterations in 3.210s)
[t-SNE] Iteration 550: error = 1.3983837, gradient norm = 0.0001140 (50 iterations in 3.108s)
[t-SNE] Iteration 600: error = 1.3690649, gradient norm = 0.0000945 (50 iterations in 3.110s)
[t-SNE] Iteration 650: error = 1.3477201, gradient norm = 0.0000853 (50 iterations in 3.097s)
[t-SNE] Iteration 700: error = 1.3325582, gradient norm = 0.0000795 (50 iterations in 3.138s)
[t-SNE] Iteration 750: error = 1.3215077, gradient norm = 0.0000702 (50 iterations in 3.131s)
[t-SNE] Iteration 800: error = 1.3128009, gradient norm = 0.0000643 (50 iterations in 3.177s)
[t-SNE] Iteration 850: error = 1.3058068, gradient norm = 0.0000602 (50 iterations in 3.181s)
[t-SNE] Iteration 900: error = 1.3001055, gradient norm = 0.0000568 (50 iterations in 3.169s)
[t-SNE] Iteration 950: error = 1.2951843, gradient norm = 0.0000535 (50 iterations in 3.187s)
[t-SNE] Iteration 1000: error = 1.2907714, gradient norm = 0.0000531 (50 iterations in 3.184s)
[t-SNE] KL divergence after 1000 iterations: 1.290771
Done..
Creating plot for this t-sne visualization..


c:\users\dell\appdata\local\programs\python\python36\lib\site-packages\seaborn\regression.py:54
  warnings.warn(msg, UserWarning)


saving this plot as image in present working directory...
```


perplexity : 50 and max_iter : 1000
label
STANDING
SITTING
LAYING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
x
y

Done