Govardhan Seshasayee

Human Activity from a smartphone data

Recognizing human activities from temporal streams of sensory data observations is a very important task on a wide variety of applications in context recognition. Human activities are hierarchical in nature, i.e. the complex activities can be decomposed to several simpler ones. Human activity recognition is the problem of classifying sequences of accelerometer data recorded by pre-installed sensors in smart phones into known well-defined movements to make it ready for predictive modelling.

Implementation:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
train = pd.read_csv("train.csv")
test = pd.read_csv('test.csv')
```

```
train.head()
```

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-kurtosis()	angle(tBody
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.710304	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	-0.861499	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.760104	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	-0.482845	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	-0.699205	

5 rows × 563 columns

```
print('Number of duplicates in train : ',sum(train.duplicated()))
print('Number of duplicates in test : ', sum(test.duplicated()))
```

```
Number of duplicates in train :  0
Number of duplicates in test :  0
```

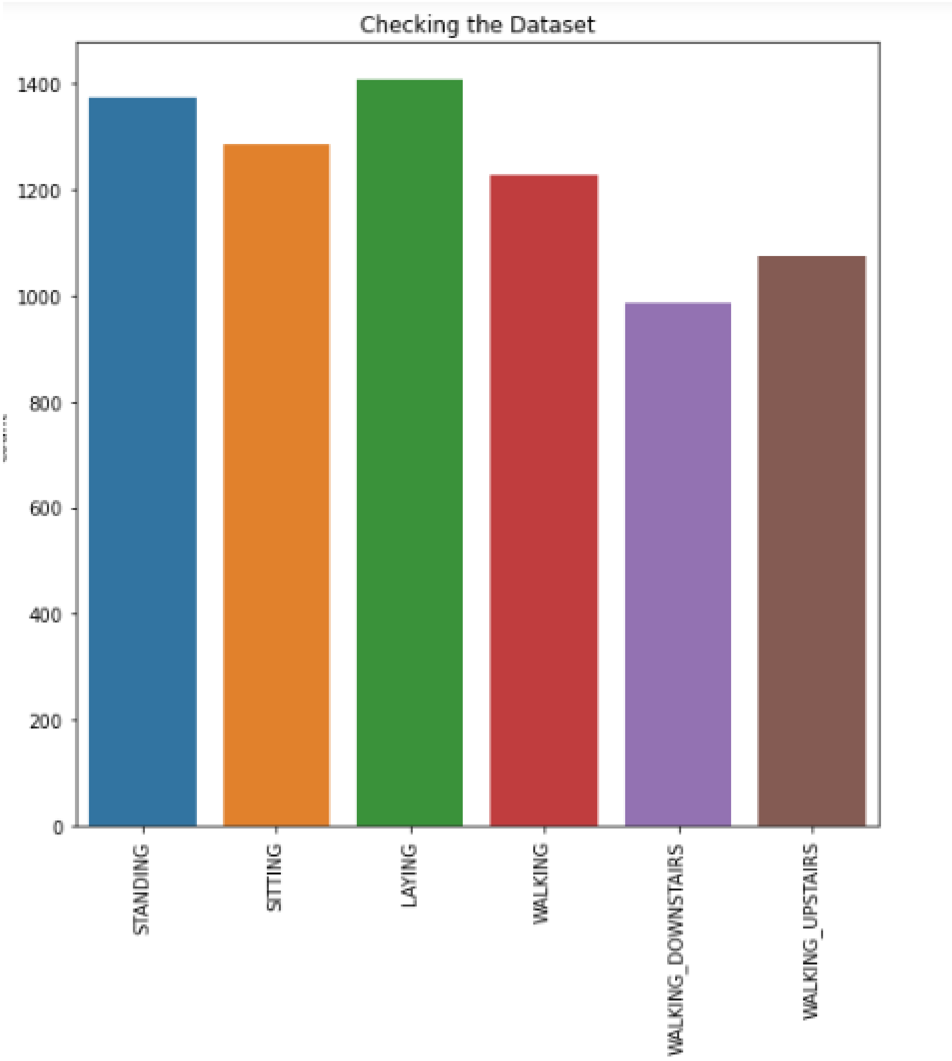

```
print('Number of duplicates in train : ',sum(train.duplicated()))
print('Number of duplicates in test : ', sum(test.duplicated()))
```

```
Number of duplicates in train :  0
Number of duplicates in test :  0
```


```
print('Total number of null values in train:',train.isna().values.sum())
print('Total number of null values in test:',test.isna().values.sum())
```

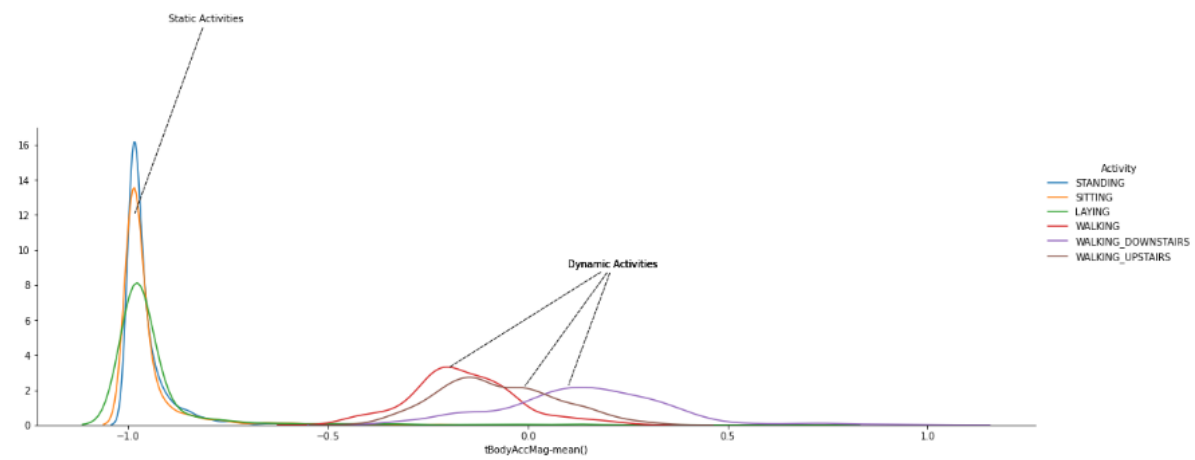
```
Total number of null values in train: 0
Total number of null values in test: 0
```

```
# Checking wether the classs are imbalanced or not
plt.figure(figsize =(8,8))
sns.countplot(train['Activity'])
plt.title('Checking the Dataset')
plt.xticks(rotation = 90)
plt.show()
```



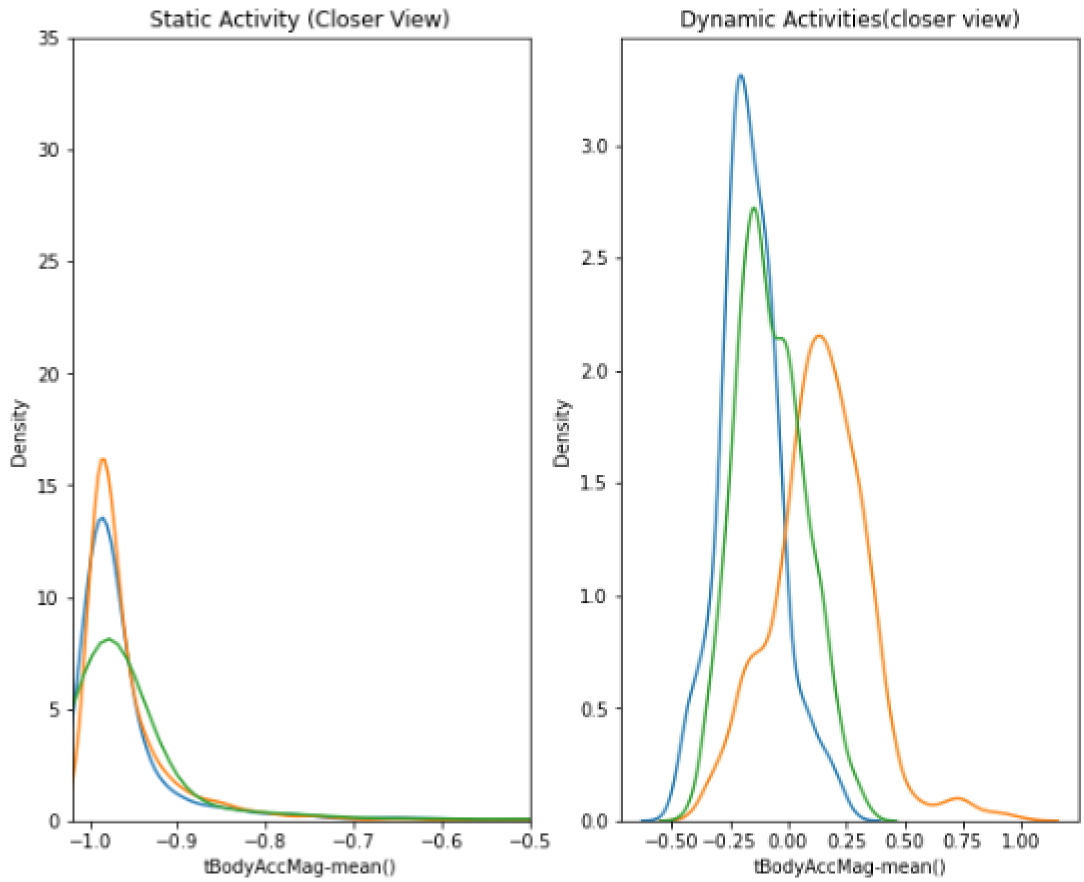

```
facetgrid = sns.FacetGrid(train, hue='Activity', height=5,aspect=3)
facetgrid.map(sns.distplot,'tBodyAccMag-mean()', hist=False).add_legend()
plt.annotate("Static Activities", xy=(-.996,21), xytext=(-0.9, 23),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
plt.annotate("Static Activities", xy=(-.990,26), xytext=(-0.9, 23),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
plt.annotate("Static Activities", xy=(-0.985,12), xytext=(-0.9, 23),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
plt.annotate("Dynamic Activities", xy=(-0.2,3.25), xytext=(0.1, 9),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
plt.annotate("Dynamic Activities", xy=(0.1,2.18), xytext=(0.1, 9),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
plt.annotate("Dynamic Activities", xy=(-0.01,2.15), xytext=(0.1, 9),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
```

```
Text(0.1, 9, 'Dynamic Activities')
```



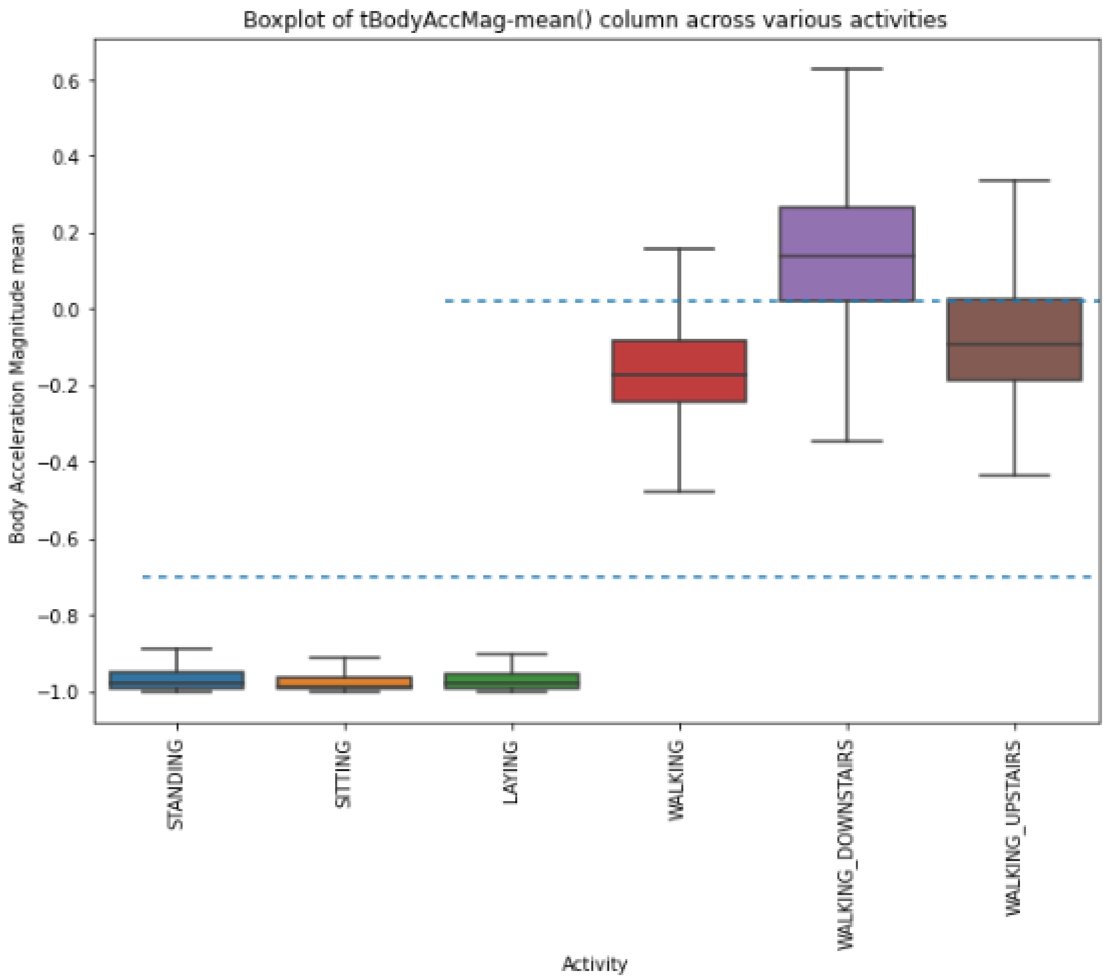
```python
plt.figure(figsize=(10,8))
plt.subplot(1,2,1)
plt.title("Static Activity (Closer View)")
sns.distplot(train[train['Activity']=='SITTING']['tBodyAccMag-mean()'],hist = False, label = 'Sitting')
sns.distplot(train[train['Activity']=='STANDING']['tBodyAccMag-mean()'],hist = False, label = 'STANDING')
sns.distplot(train[train['Activity']=='LAYING']['tBodyAccMag-mean()'],hist = False, label = 'LAYING')
plt.axis([-1.02, -0.5, 0, 35])
plt.subplot(1,2,2)
plt.title("Dynamic Activities(closer view)")
sns.distplot(train[train["Activity"]=="WALKING"]['tBodyAccMag-mean()'],hist = False, label = 'WALKING')
sns.distplot(train[train["Activity"]=="WALKING_DOWNSTAIRS"]['tBodyAccMag-mean()'],hist = False,label = 'WALKING_DOWNSTAIRS')
sns.distplot(train[train["Activity"]=="WALKING_UPSTAIRS"]['tBodyAccMag-mean()'],hist = False, label = 'WALKING_UPSTAIRS')
```

```
<AxesSubplot:title={'center':'Dynamic Activities(closer view)'}, xlabel='tBodyAccMag-mean()', ylabel='Density'>
```



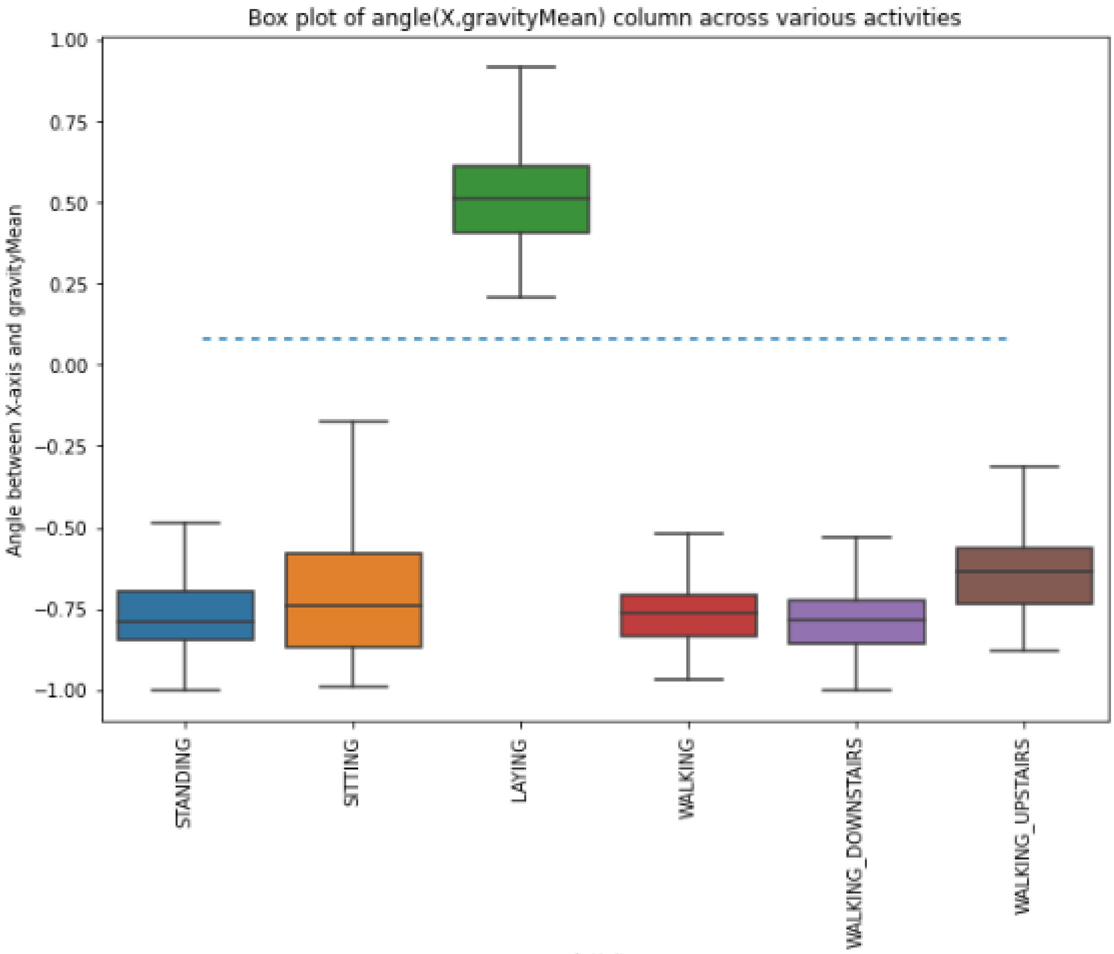

```python
plt.figure(figsize=(10,7))
sns.boxplot(x='Activity', y='tBodyAccMag-mean()',data=train, showfliers=False)
plt.ylabel('Body Acceleration Magnitude mean')
plt.title("Boxplot of tBodyAccMag-mean() column across various activities")
plt.axhline(y=-0.7, xmin=0.05,dashes=(3,3))
plt.axhline(y=0.020, xmin=0.35, dashes=(3,3))
plt.xticks(rotation=90)
```

```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'STANDING'),
  Text(1, 0, 'SITTING'),
  Text(2, 0, 'LAYING'),
  Text(3, 0, 'WALKING'),
  Text(4, 0, 'WALKING_DOWNSTAIRS'),
  Text(5, 0, 'WALKING_UPSTAIRS')])
```



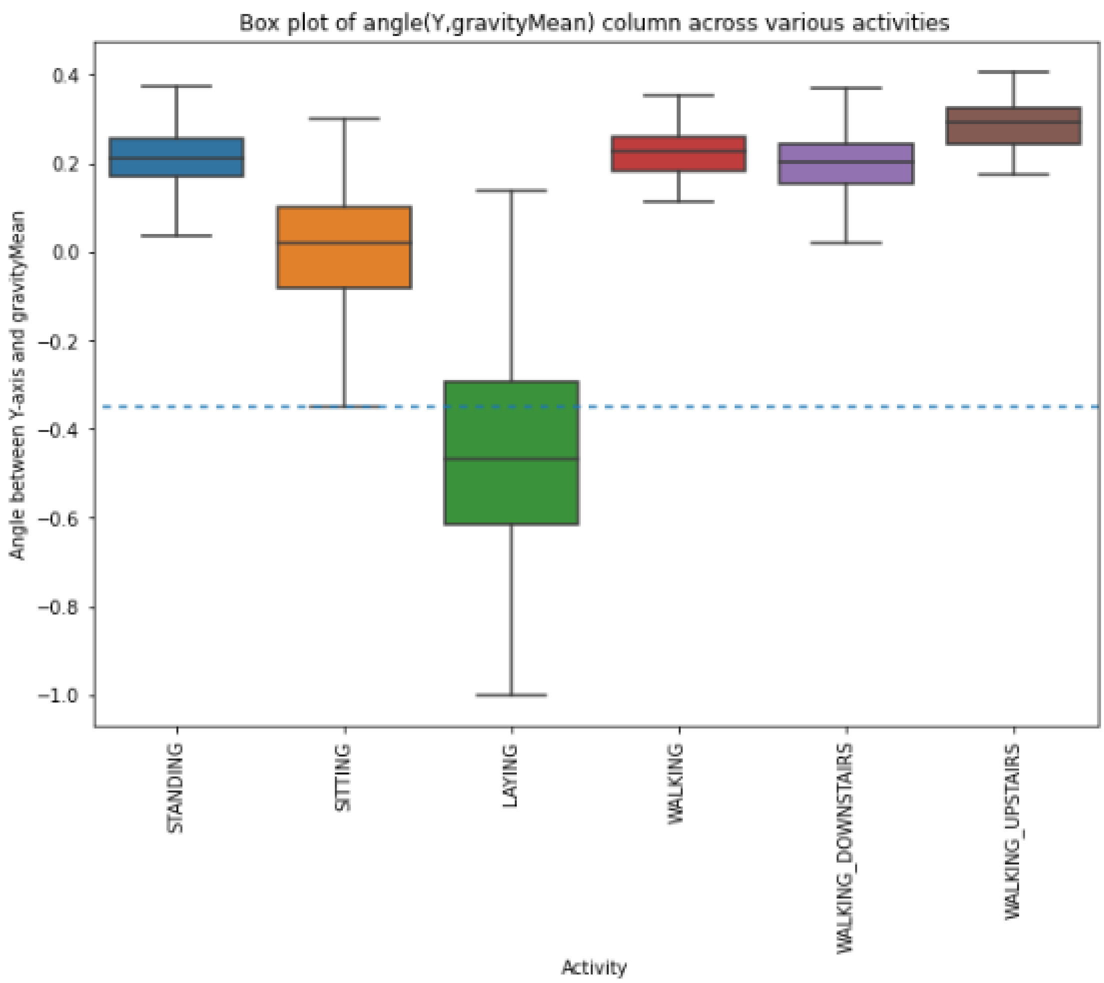

```
plt.figure(figsize=(10,7))
sns.boxplot(x='Activity', y='angle(X,gravityMean)', data=train, showfliers=False)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,dashes=(3,3))
plt.ylabel("Angle between X-axis and gravityMean")
plt.title('Box plot of angle(X,gravityMean) column across various activities')
plt.xticks(rotation = 90)
```

```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'STANDING'),
  Text(1, 0, 'SITTING'),
  Text(2, 0, 'LAYING'),
  Text(3, 0, 'WALKING'),
  Text(4, 0, 'WALKING_DOWNSTAIRS'),
  Text(5, 0, 'WALKING_UPSTAIRS')])
```




```
plt.figure(figsize=(10,7))
sns.boxplot(x='Activity', y='angle(Y,gravityMean)', data = train, showfliers=False)
plt.ylabel("Angle between Y-axis and gravityMean")
plt.title('Box plot of angle(Y,gravityMean) column across various activities')
plt.xticks(rotation = 90)
plt.axhline(y=-0.35, xmin=0.01, dashes=(3,3))
```

```
<matplotlib.lines.Line2D at 0x248abe860a0>
```


Box plot of angle(Y,gravityMean) column across various activities
Angle between Y-axis and gravityMean
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
STANDING
SITTING
LAYING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
Activity

```
from sklearn.manifold import TSNE
```

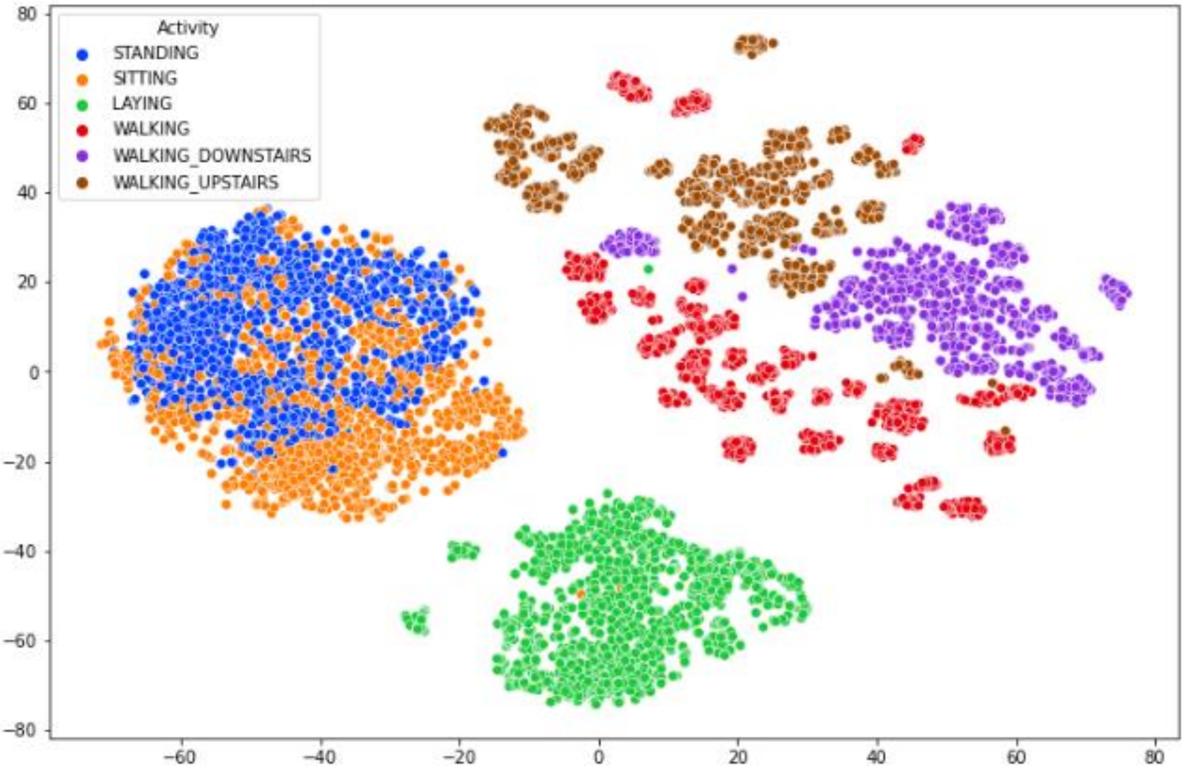
```
X_for_tsne = train.drop(['subject', 'Activity'], axis=1)
```

```
%time
tsne = TSNE(random_state = 32, n_components=2, verbose=1, perplexity=50, n_iter=1000).fit_transform(X_for_tsne)
```

```
Wall time: 0 ns
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.009s...
[t-SNE] Computed neighbors for 7352 samples in 3.461s...
[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] KL divergence after 250 iterations with early exaggeration: 74.122711
[t-SNE] KL divergence after 1000 iterations: 1.282488
```

```
plt.figure(figsize=(12,8))
sns.scatterplot(x =tsne[:, 0], y = tsne[:, 1], hue = train["Activity"],palette="bright")
```

```
<AxesSubplot:>
```



```
y_train = train.Activity
X_train = train.drop(['subject', 'Activity'], axis=1)
y_test = test.Activity
X_test = test.drop(['subject', 'Activity'], axis=1)
print('Training data size : ', X_train.shape)
print('Test data size : ', X_test.shape)
```

```
Training data size :  (7352, 561)
Test data size :  (2947, 561)
```

```
y_train
```

```
0               STANDING
1               STANDING
2               STANDING
3               STANDING
4               STANDING
                  ...
7347    WALKING_UPSTAIRS
7348    WALKING_UPSTAIRS
7349    WALKING_UPSTAIRS
7350    WALKING_UPSTAIRS
7351    WALKING_UPSTAIRS
Name: Activity, Length: 7352, dtype: object
```

Logistic regression model with Hyperparameter tuning and cross validation

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score,classification_report
```

```
parameters = {'C':np.arange(10,61,10), 'penalty':['l2','l1']}
lr_classifier = LogisticRegression()
lr_classifier_rs = RandomizedSearchCV(lr_classifier,param_distributions=parameters,cv=5,random_state=42)
lr_classifier_rs.fit(X_train, y_train)
y_pred = lr_classifier_rs.predict(X_test)
```

```
y_pred
```

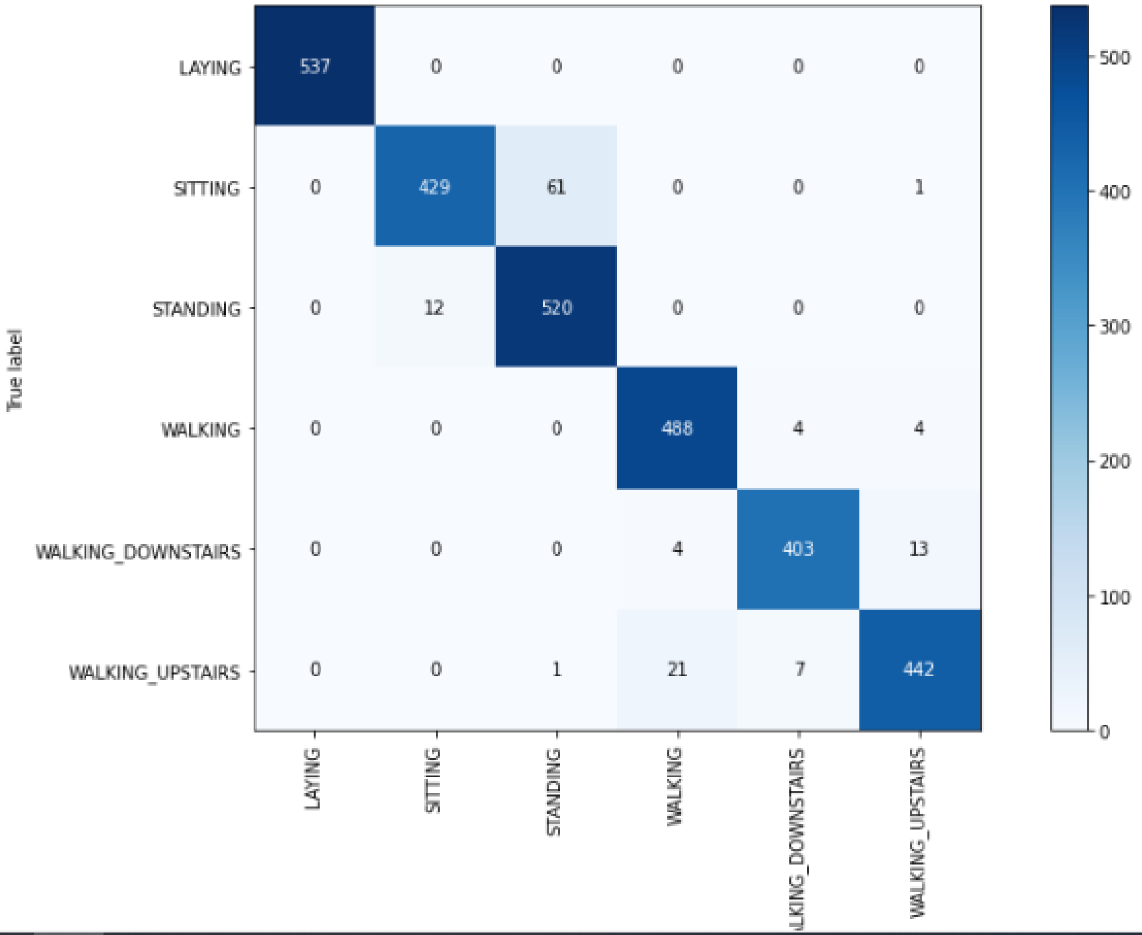
```
array(['STANDING', 'STANDING', 'STANDING', ..., 'WALKING_UPSTAIRS',
       'WALKING_UPSTAIRS', 'WALKING_UPSTAIRS'], dtype=object)
```

```
lr_accuracy = accuracy_score(y_true=y_test, y_pred=y_pred)
print("Accuracy using Logistic Regression : ", lr_accuracy)
```

```
Accuracy using Logistic Regression :  0.9565659993213438
```

```
# function to plot confusion matrix
def plot_confusion_matrix(cm,lables):
    fig, ax = plt.subplots(figsize=(12,8)) # for plotting confusion matrix as image
    im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    ax.figure.colorbar(im, ax=ax)
    ax.set(xticks=np.arange(cm.shape[1]),
    yticks=np.arange(cm.shape[0]),
    xticklabels=lables, yticklabels=lables,
    ylabel='True label',
    xlabel='Predicted label')
    plt.xticks(rotation = 90)
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, int(cm[i, j]),ha="center", va="center",color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
```

```
cm = confusion_matrix(y_test.values,y_pred)
plot_confusion_matrix(cm, np.unique(y_pred))  # plotting confusion matrix
```




```
#function to get best random search attributes
def get_best_randomsearch_results(model):
    print("Best estimator : ", model.best_estimator_)
    print("Best set of parameters : ", model.best_params_)
    print("Best score : ", model.best_score_)
```

```
# getting best random search attributes
get_best_randomsearch_results(lr_classifier_rs)
```

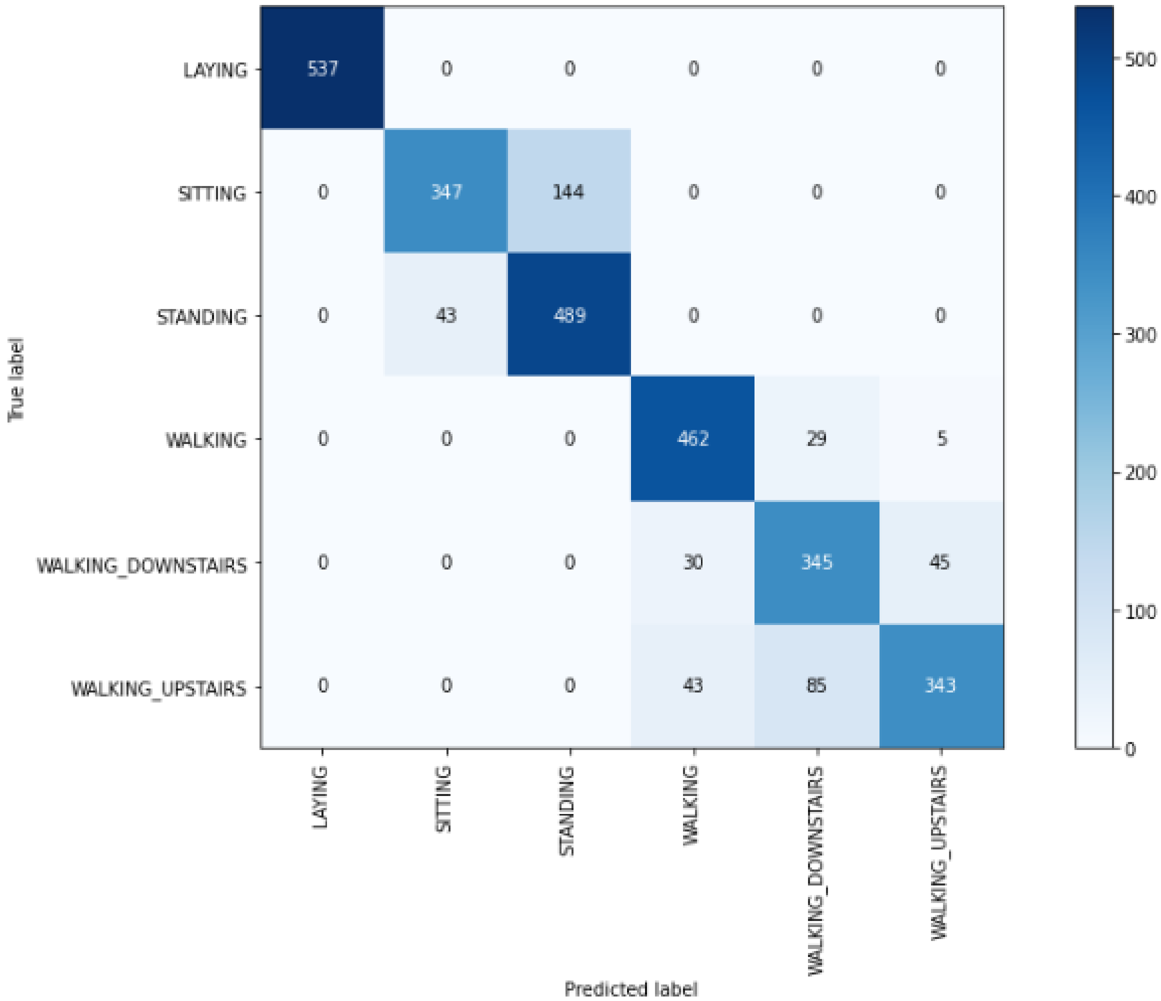
```
Best estimator :  LogisticRegression(C=60)
Best set of parameters :  {'penalty': 'l2', 'C': 60}
Best score :  0.9352626053820577
```

```
from sklearn.tree import DecisionTreeClassifier
parameters = {'max_depth':np.arange(2,10,2)}
dt_classifier = DecisionTreeClassifier()
dt_classifier_rs = RandomizedSearchCV(dt_classifier,param_distributions=parameters,random_state = 42)
dt_classifier_rs.fit(X_train, y_train)
y_pred = dt_classifier_rs.predict(X_test)
```

```
dt_accuracy = accuracy_score(y_true=y_test, y_pred=y_pred)
print("Accuracy using Decision tree : ", dt_accuracy)
```

```
Accuracy using Decision tree :  0.8561248727519511
```

```
cm = confusion_matrix(y_test.values,y_pred)
plot_confusion_matrix(cm, np.unique(y_pred)) # plotting confusion matrix
```




```
# getting best random search attributes
get_best_randomsearch_results(dt_classifier_rs)
```

```
Best estimator :  DecisionTreeClassifier(max_depth=6)
Best set of parameters :  {'max_depth': 6}
Best score :  0.8501108505944867
```

```
from hmmlearn import hmm
model = hmm.GaussianHMM(n_components=3, covariance_type="full", n_iter=100)
```

```
model.fit(X_train)
```

```
GaussianHMM(covariance_type='full', n_components=3, n_iter=100)
```

```
y_pred_hmm = model.predict(X_test)
```

```
np.unique(y_pred_hmm)
```

```
array([0, 1, 2], dtype=int64)
```