Show Code

1.IMPORTING ALL THE REQUIRED PACKAGES.

```
All required packages are imported
```

2. LOADING TRAIN AND TEST DATA,COMBINING TRAIN AND TEST DATA

```
Train dataset has been loaded
```

```
Test dataset has been loaded
```

```
Train and test datasets are appended.
```

3.ANALISING THE DATA

3.1 Checking the dimension of the data

```
No. of Rows in the data:  10299
No. of Columns in the data:  563
```

3.2 Printing the sample of the data

Out[7]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyA mad(
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.9831
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.9827
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979

5 rows × 563 columns

3.3 Checking if the data contains NULL values

```
Data contains NULL values: False
```

3.4 Checking for DUPLICATED VALUES

```
Number of DUPLICATED values in the Data: 0
```

3.5 Checking the columns present in the data

```
Columns Present in the Data Set
Index(['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(tBodyAccMean,gravity)',
       'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'subject', 'Activit
y'],
      dtype='object', length=563)
```

3.6 Checking for Collinearity.

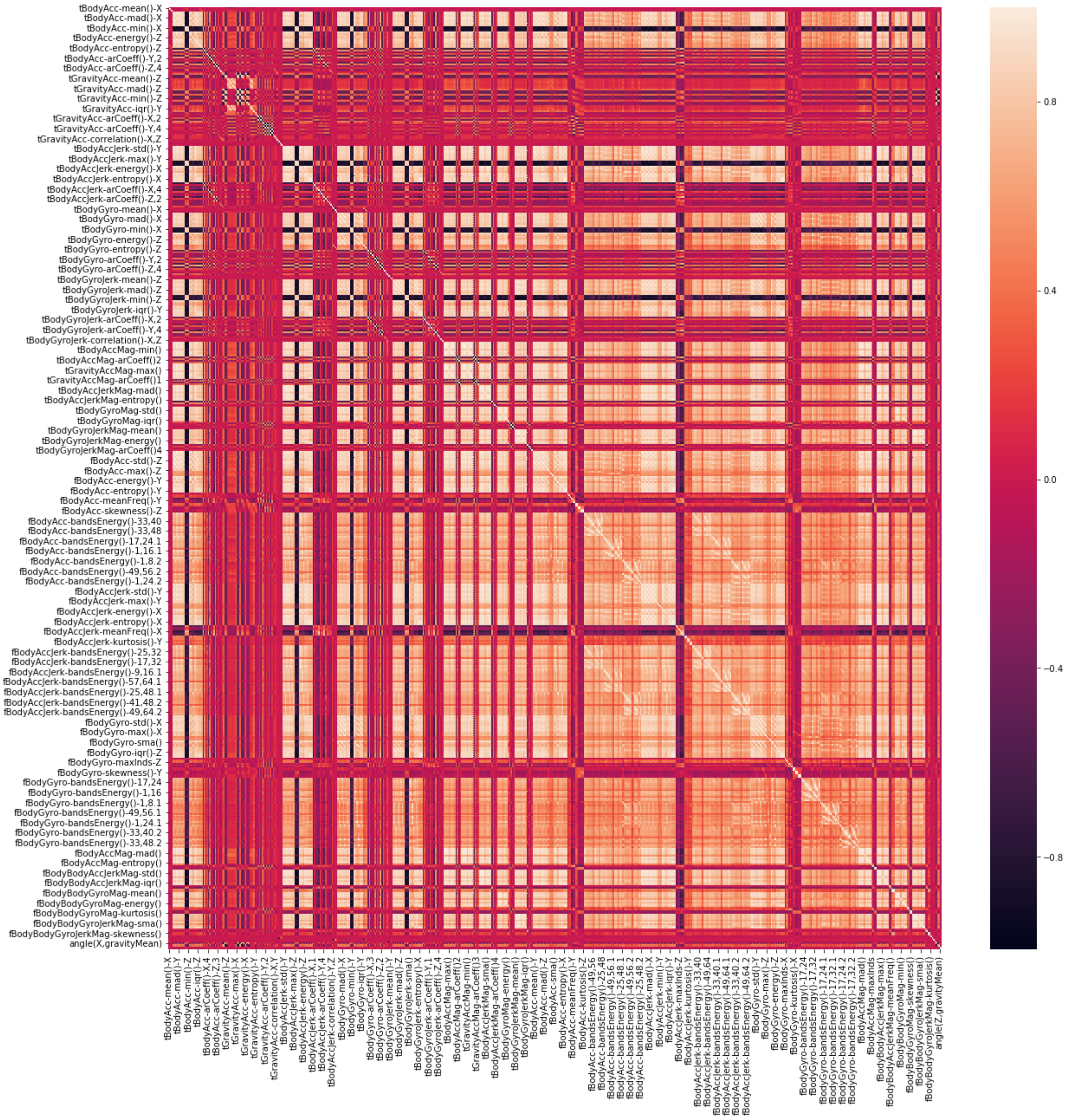
Out[11]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X
tBodyAcc-mean()-X	1.000000	0.128037	-0.230302	0.004590	-0.016785	-0.036071	0.010303
tBodyAcc-mean()-Y	0.128037	1.000000	-0.029882	-0.046352	-0.046996	-0.054153	-0.045247
tBodyAcc-mean()-Z	-0.230302	-0.029882	1.000000	-0.024185	-0.023745	-0.015632	-0.022872
tBodyAcc-std()-X	0.004590	-0.046352	-0.024185	1.000000	0.922525	0.861910	0.998662
tBodyAcc-std()-Y	-0.016785	-0.046996	-0.023745	0.922525	1.000000	0.888259	0.918561

5 rows × 562 columns

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x193e7866be0>
```

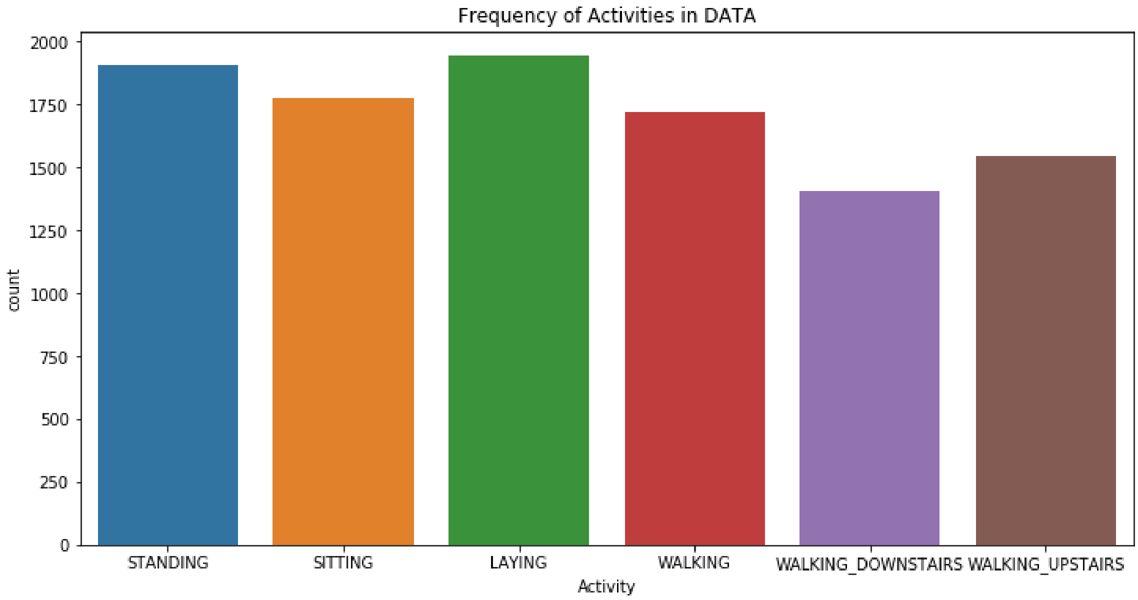


3.7 Checking the possible human activities that can be predicted.

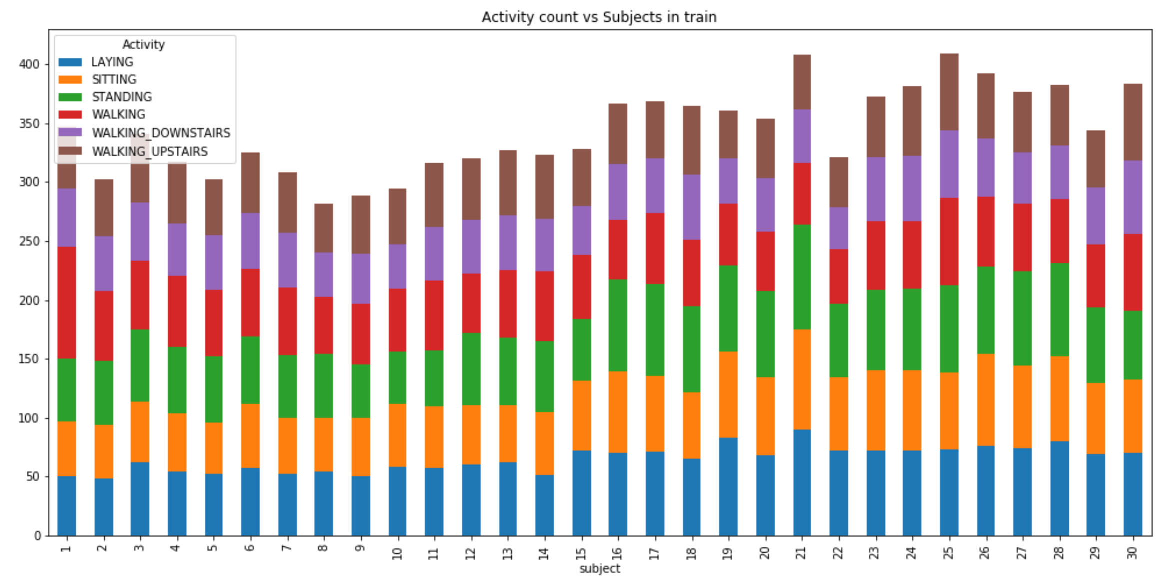
```
Human Activities that can be predicted using this dataset are:
STANDING
SITTING
LAYING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
```

4. DATA VISUALISATION

4.1 Frequency of each activity



4.2 Daily Routine i.e. time spent for each activity of the 30 candidates,whose data is collected.



5.DATA PREPROCESSING

The data contains no null values, no duplicate values. The data also have no categorial values in the features. But the data have high collinearity between the features and also the number of columns in the data is equal to 563 so to reduce the dimension,get rwed of collinearity and reduce the model fitting time we use Principal component analysis .

Out[17]:

	0	1	2	3	4	5	6	7	
0	14.516791	-7.040348	-0.848517	-1.311452	1.918762	-0.754556	-0.576247	0.192814	0.67
1	14.512251	-7.066135	-0.714288	-1.728410	0.957263	-0.503252	0.936521	0.066616	-0.86
2	14.521702	-7.002312	-0.434467	-2.051527	0.740811	0.106743	-0.095229	0.016855	-0.01
3	14.499971	-7.202094	0.218497	-2.058371	0.477533	-0.697812	0.861611	0.692418	-0.50
4	14.491649	-7.270128	0.422661	-2.277227	0.468014	-0.357469	0.282462	0.252537	-0.09

5 rows × 29 columns

After using PCA the size of the columns decreases. Checking the size.

```
No of rows in the dataset after using PCA: 10299
No of columns in the dataset after using PCA: 29
```

6. Model fitting

6.1 Model fitting using KNN

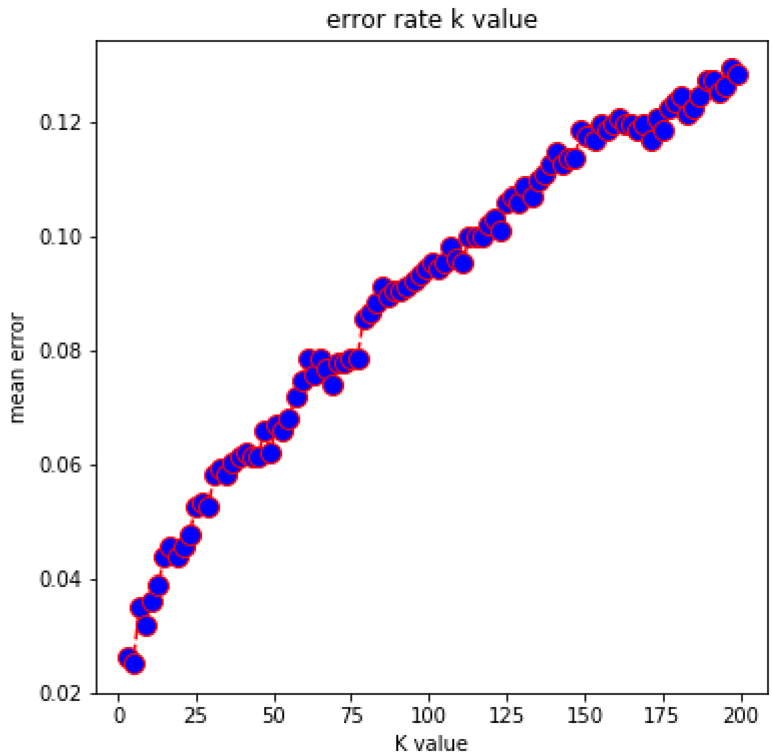
6.1.1 Finding the random state required for splitting data in training and testing data for KNeighborsClassifier

```
The Random State for splitting for KNeighborsClassifier: 161
```

6.1.2 Splitting the data into training and testing data.

```
Data has been splitted into x_train,x_test,y_train,y_test for KNeighborsCl
assifier
```

6.1.3 Finding the optimum value of nearest neighbours



6.1.4 Fitting the KNN model.

Out[42]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

6.1.5 Prediction of y using the Model.

```
The Model For KNeighboursClassifier is predicted
```

6.2 Model Fitting using DecisionTreeClassifier

6.2.1 Finding the random state required for splitting data in training and testing data for DecisionTreeClassifier

```
The Random State for splitting for DecisionTreeClassifier: 1194
```

6.2.2 Splitting the data into training and testing data.

```
Data has been splitted into x_train,x_test,y_train,y_test for DecisionTree
Classifier
```

6.2.3 Fitting the DecisionTreeClassifier.

Out[60]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=N
one,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=0.01, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=Non
e,
            splitter='best')
```

6.2.4 Prediction of y using the Model.

```
The Model For DecisionTreeClassifier is predicted
```

6.3 Model Fitting using RandomForestClassifier

6.3.1 Finding the random state required for splitting data in training and testing data for RandomForestClassifier

```
The Random State for splitting for RandomForestClassifier: 1633
```

```
The Random State for RandomForestClassifier parameter: 1511
```

6.3.2 Splitting the data into training and testing data.

```
Data has been splitted into x_train,x_test,y_train,y_test for RandomForest
Classifier
```

6.3.3 Fitting the RandomForestClassifier.

Out[51]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=0.01, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=1511, verbose=0,
            warm_start=False)
```

6.3.4 Prediction of y using the Model.

```
The Model For RandomForestClassifier is predicted
```

7.Determination of Accuracy

```
The Accuracy Score using KNeighborsClassifier is:  0.974757281553398

The Accuracy Score using DecisionTreeClassifier is:  0.8543689320388349

The Accuracy Score using RandomForestClassifier is:  0.9058252427184466
```

Graphical Representation of ACCURACY and EXECUTION Time for The Models Used.