# Experimental Work

Our idea is to use sensor data coming from a smartphone to recognize human activities.
For that we used the accelerometer and gyroscope sensors from a smartphone.

## Extracting the data

To extract the accelerometer and gyroscope's data, we have used an android app that is accessible to everyone in Play Store called '**Physics Toolbox Suite**'. This app gives us the data in .CSV file format from the moment that we start recording to the moment we stop it (.csv files from this phase of extracting data will be given in the attachments to this file).

- ### Accelerometer

  The linear accelerometer measures acceleration in a straight-line in three different dimensions.
  Linear acceleration changes whenever the mobile device speeds up, slows down or changes direction. When the mobile device is at rest with respect to the surface of the earth, it reads acceleration values of (0,0,0).

  Linear acceleration is derived from the accelerometer, but also uses the gyroscope and the magnetometer to negate the effects of the earth's gravitational field on the sensor.
  The accelerometer contains either a cantilever or comb-like inertial massed attached to springs with one dimension.

  *Sensor Collection Rate:* 125*Hz*

- ### Gyroscope

  The gyroscope measures rotational velocity around three axes (x,y,z).

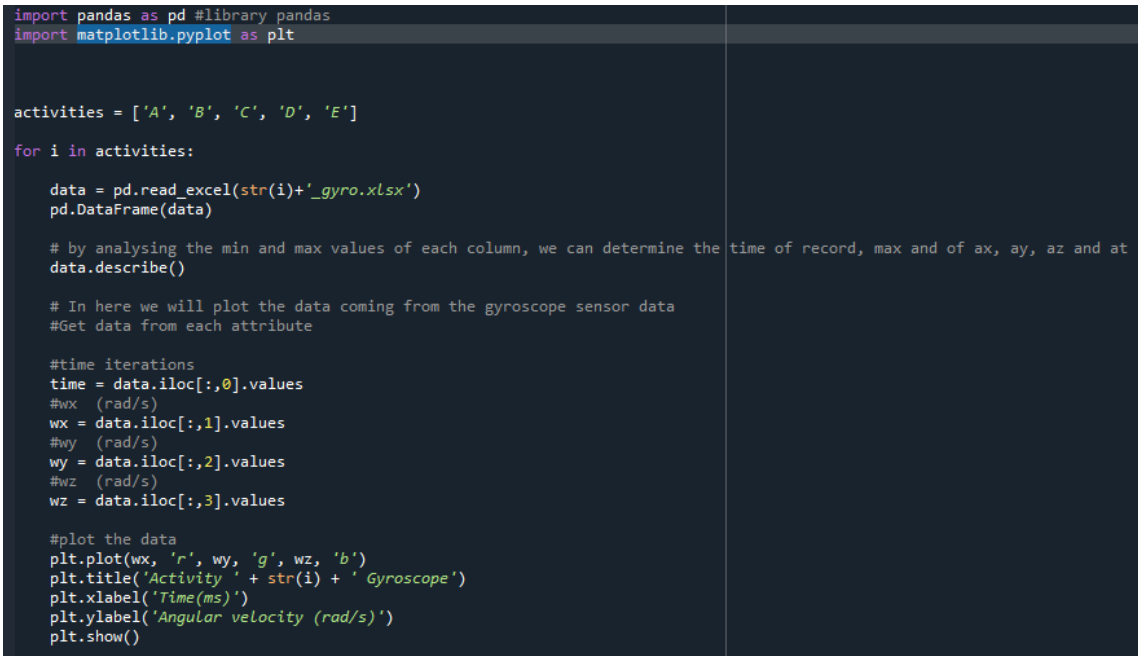
  The gyroscope is often composed of one or a paired set of comb-like inertial masses that are free to move along a plane. A small electrical current causes the frames of the comb-like structures to resonate along a single dimension. When the mobile device rotates, the comb-like structure begins to move out of the single direction of resonant vibration. This is due to Coriolis effect, often attributed to the Coriolis force ( a false force) that results from different frames of reference.
  As the mobile device spins, the comb-like mass moves about in the presence of a capacitor plates between the comb-like structures, varying the electrical potential across the plates. The changes in electric potential due to the difference in direction of the vibration compared to a non-rotating system is interpreted as rotational velocity.

  *Sensor Collection Rate:* 500 *Hz*

## Plots of the raw data from sensors

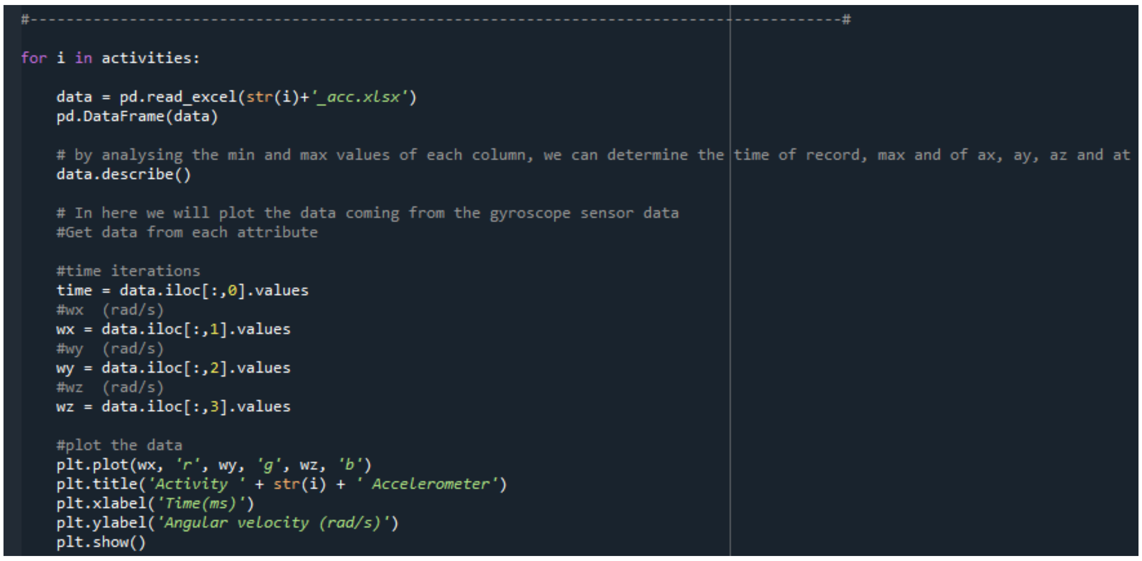
To plot the raw data extracted from the accelerometer and gyroscope, we used a python library "*matplotlib.pyplot*" and have implemented a small script to generate the plot for all activities. The activities are name A, B, C, D and E (the description of the activities is in the READ_ME.txt file).

The script to generate the plot of the raw data is the following:



```python
import pandas as pd #library pandas
import matplotlib.pyplot as plt


activities = ['A', 'B', 'C', 'D', 'E']

for i in activities:

    data = pd.read_excel(str(i)+'_gyro.xlsx')
    pd.DataFrame(data)

    # by analysing the min and max values of each column, we can determine the time of record, max and of ax, ay, az and at
    data.describe()

    # In here we will plot the data coming from the gyroscope sensor data
    #Get data from each attribute

    #time iterations
    time = data.iloc[:,0].values
    #wx  (rad/s)
    wx = data.iloc[:,1].values
    #wy  (rad/s)
    wy = data.iloc[:,2].values
    #wz  (rad/s)
    wz = data.iloc[:,3].values

    #plot the data
    plt.plot(wx, 'r', wy, 'g', wz, 'b')
    plt.title('Activity ' + str(i) + ' Gyroscope')
    plt.xlabel('Time(ms)')
    plt.ylabel('Angular velocity (rad/s)')
    plt.show()
```

*Figure 1- Script part to plot raw data from gyroscope*



```python
#---------------------------------------------------------------------------------------#

for i in activities:

    data = pd.read_excel(str(i)+'_acc.xlsx')
    pd.DataFrame(data)

    # by analysing the min and max values of each column, we can determine the time of record, max and of ax, ay, az and at
    data.describe()

    # In here we will plot the data coming from the gyroscope sensor data
    #Get data from each attribute

    #time iterations
    time = data.iloc[:,0].values
    #wx  (rad/s)
    wx = data.iloc[:,1].values
    #wy  (rad/s)
    wy = data.iloc[:,2].values
    #wz  (rad/s)
    wz = data.iloc[:,3].values

    #plot the data
    plt.plot(wx, 'r', wy, 'g', wz, 'b')
    plt.title('Activity ' + str(i) + ' Accelerometer')
    plt.xlabel('Time(ms)')
    plt.ylabel('Angular velocity (rad/s)')
    plt.show()
```

*Figure 2- Script part to plot raw data from accelerometer*

The script that generates the plots is also in the attachments for analysis named ***plotsAccGyro.py***.

The plots of raw data from gyroscope are:

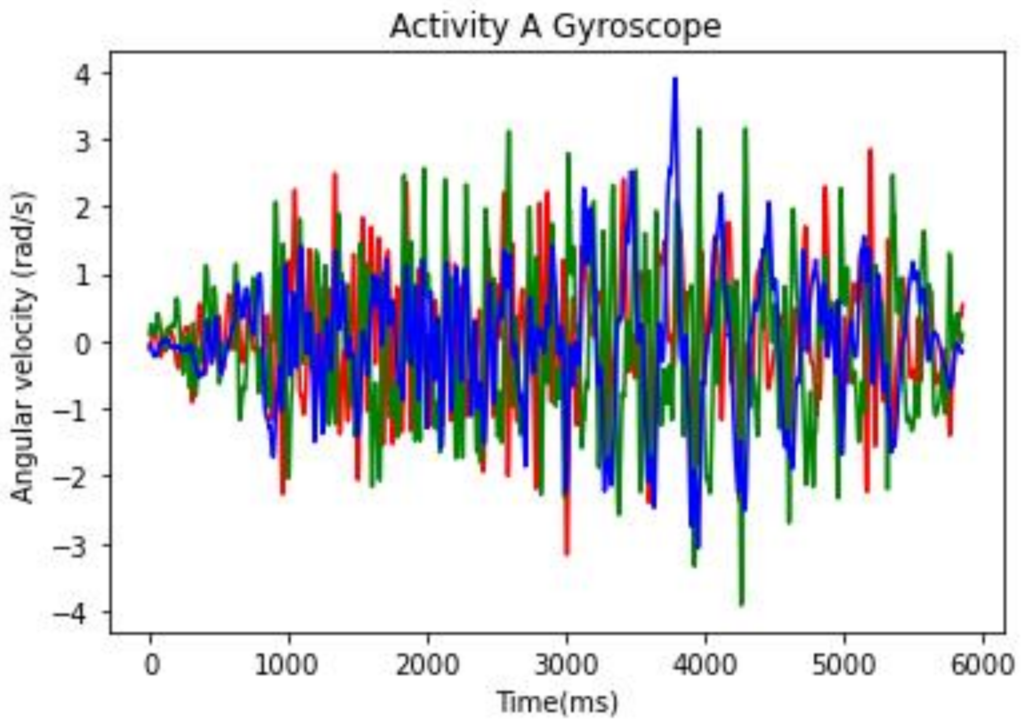


*Figure 3- Plot of Activity A (Gyroscope)*

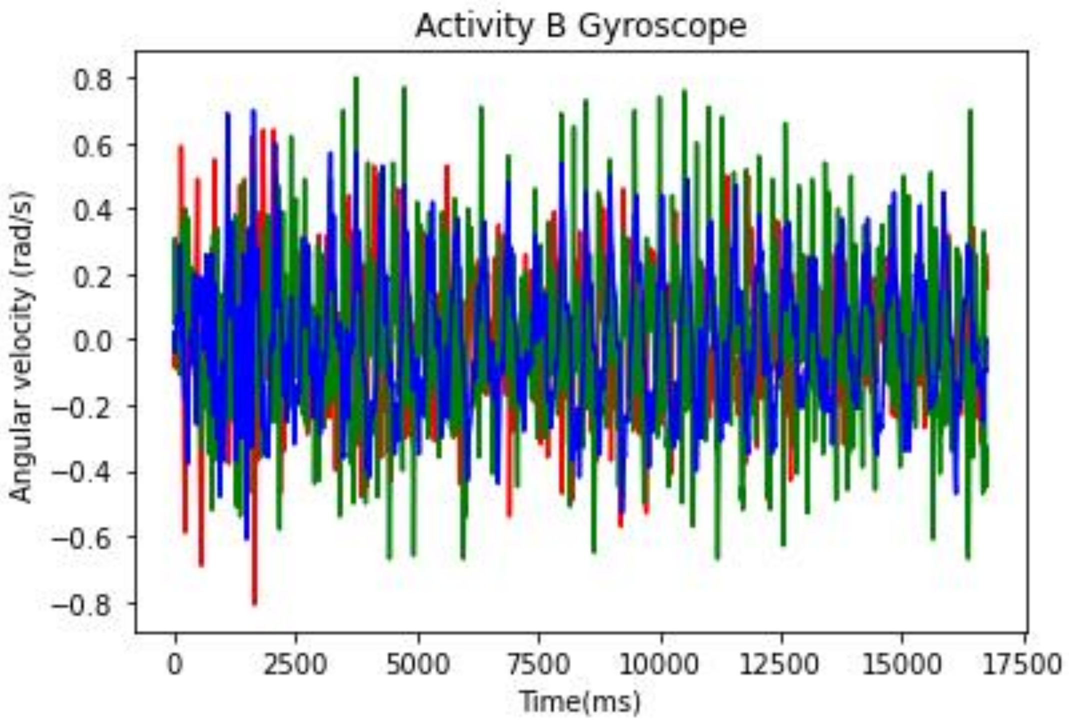


*Figure 4- Plot of Activity B (Gyroscope)*

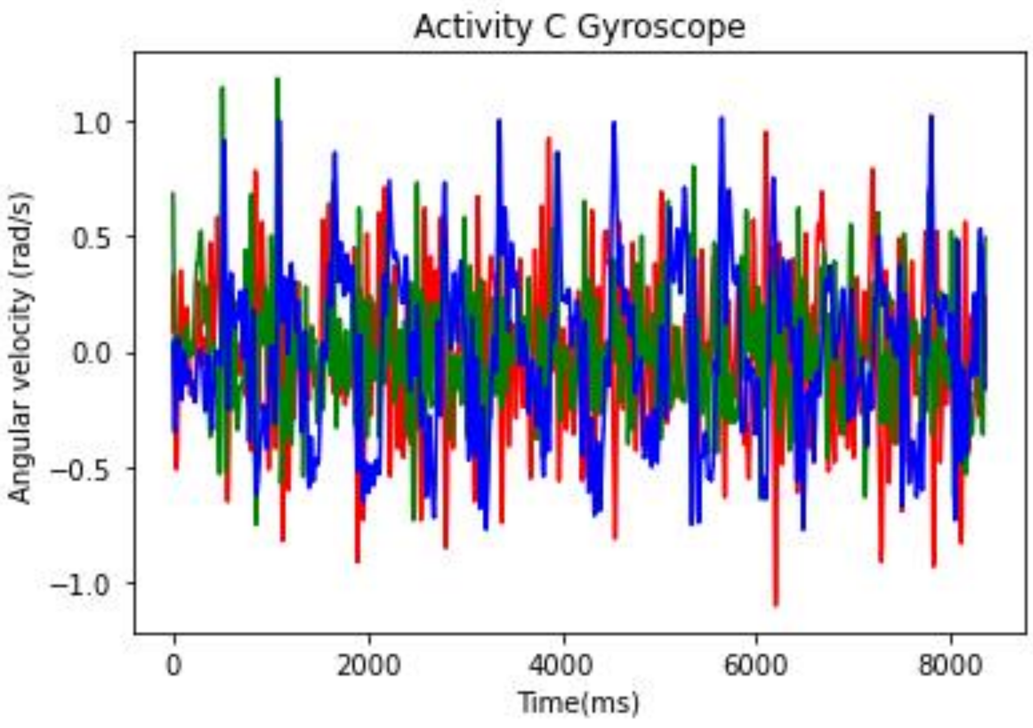


*Figure 5- Plot of Activity C (Gyroscope)*

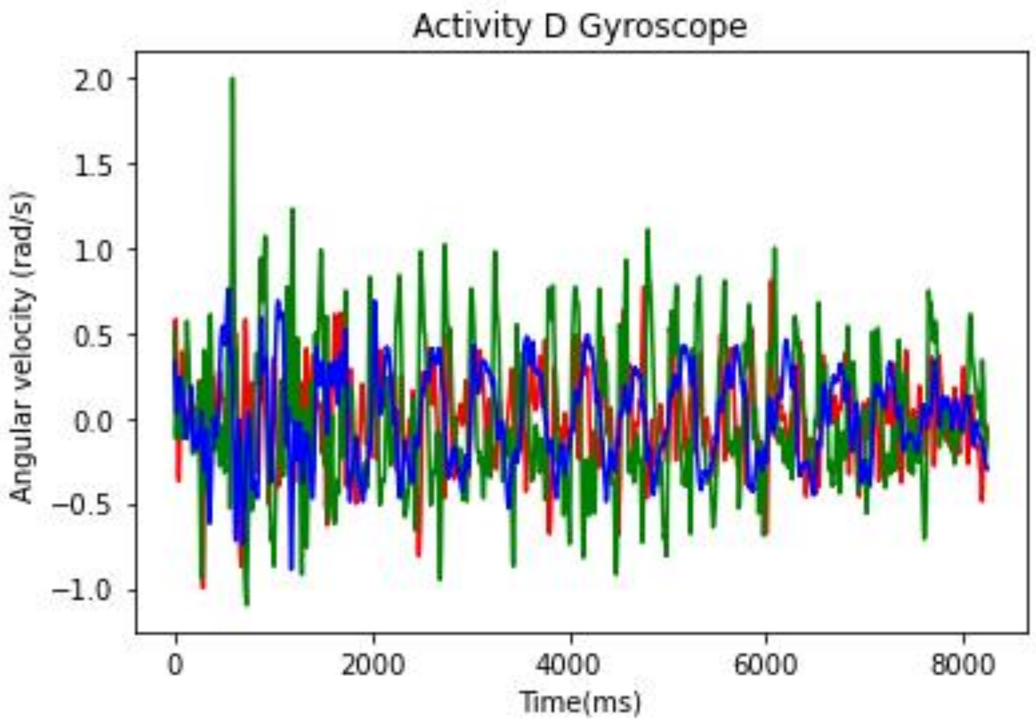


*Figure 6 - Plot of Activity D (Gyroscope)*

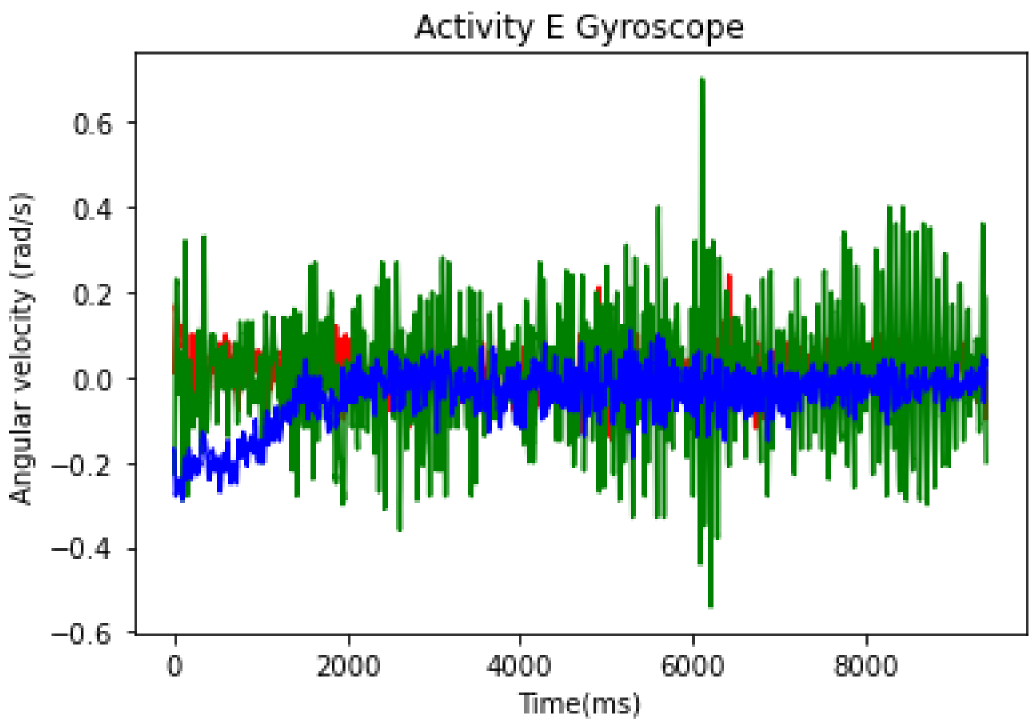


*Figure 7 - Plot of Activity E (Gyroscope)*

The plots of raw data from accelerometer are:

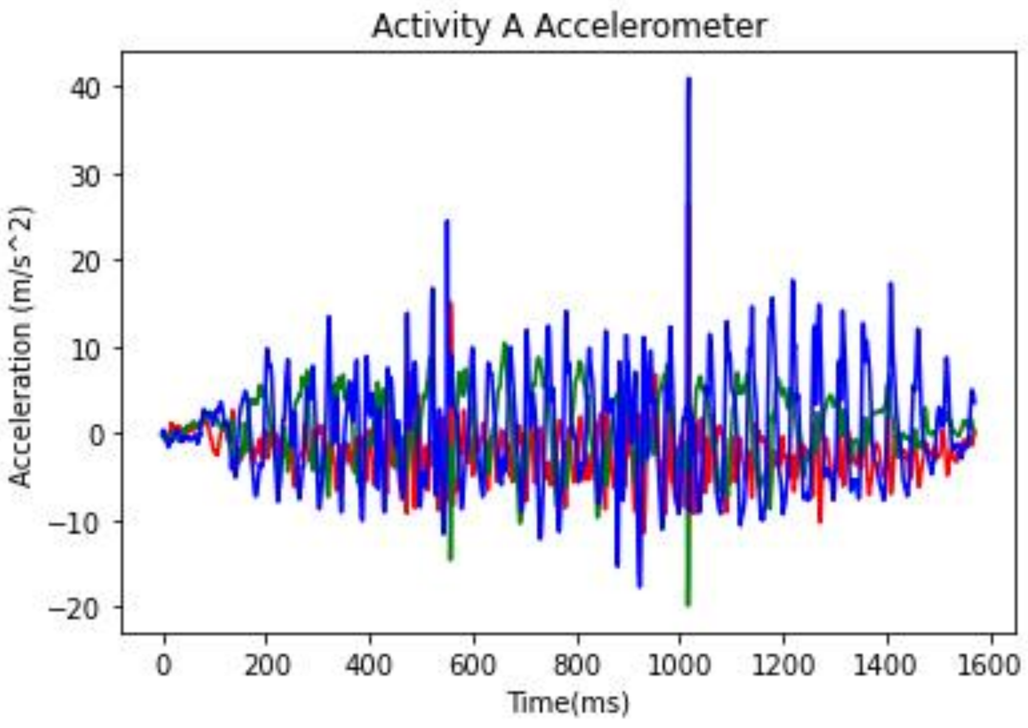


*Figure 8 - Plot of Activity A (Accelerometer)*

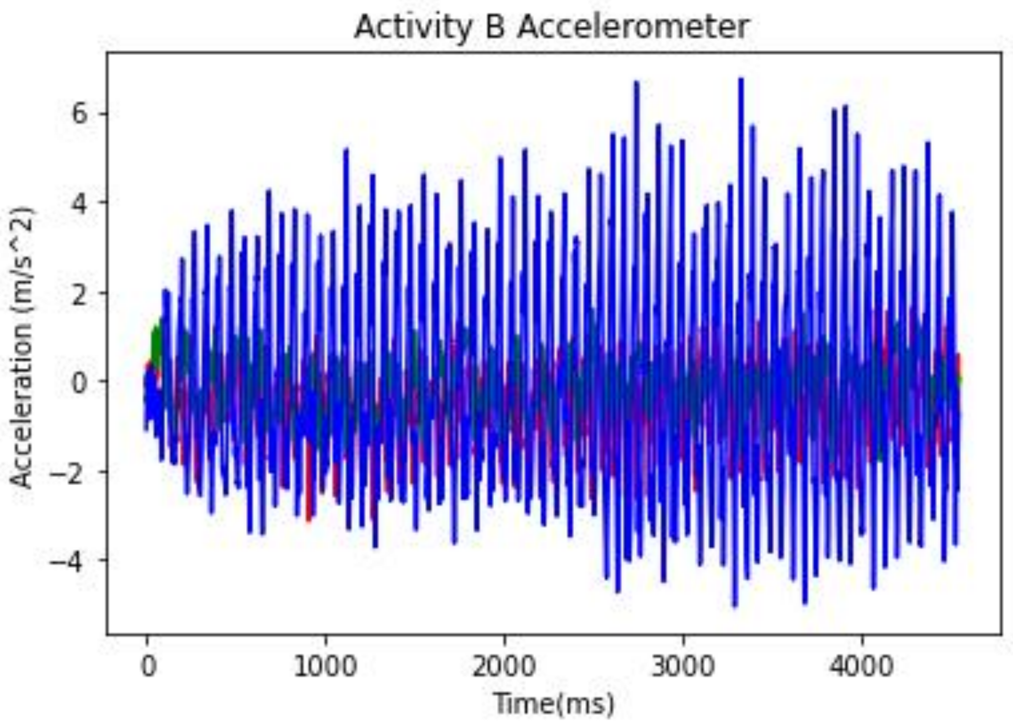


*Figure 9 - Plot of Activity B (Accelerometer)*

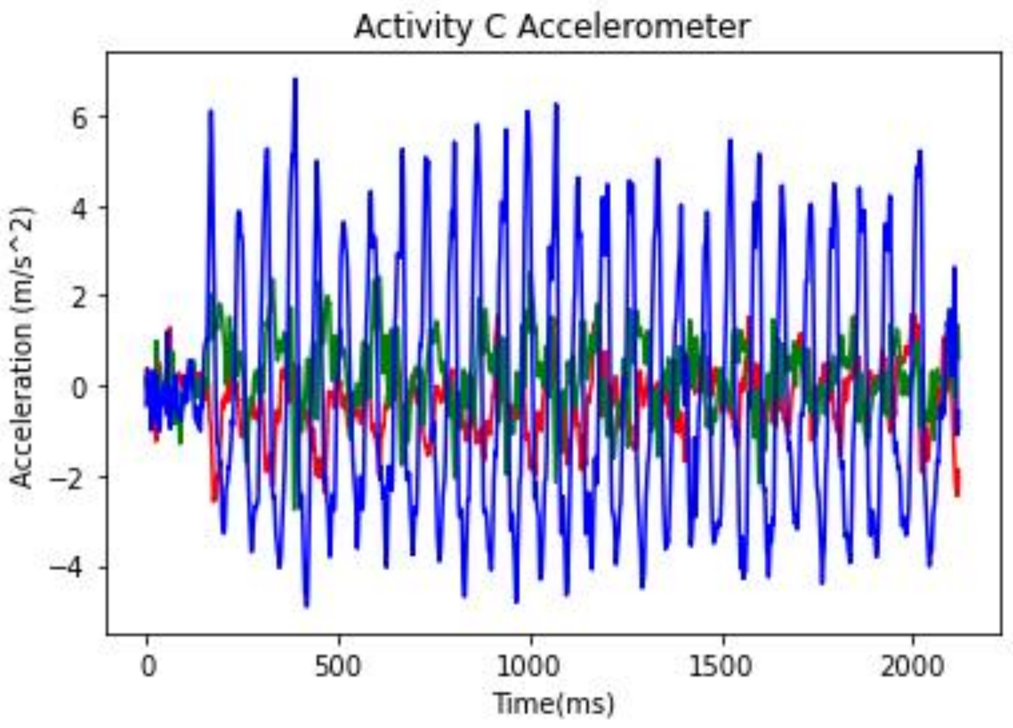


*Figure 10 - Plot of Activity C (Accelerometer)*

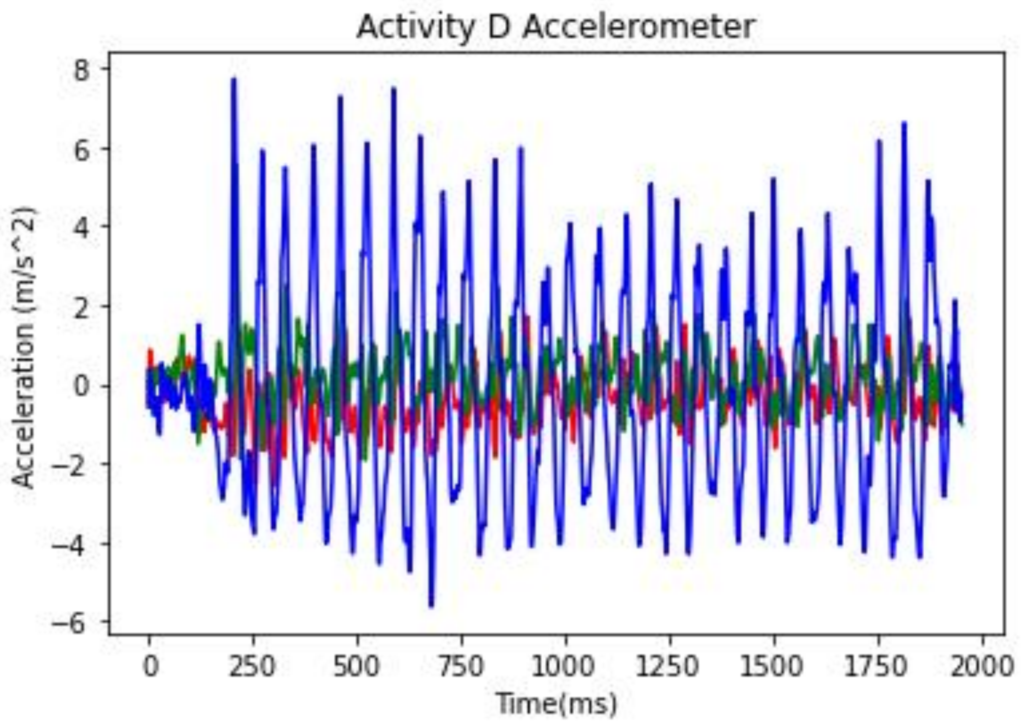


*Figure 11 - Plot of Activity D (Accelerometer)*

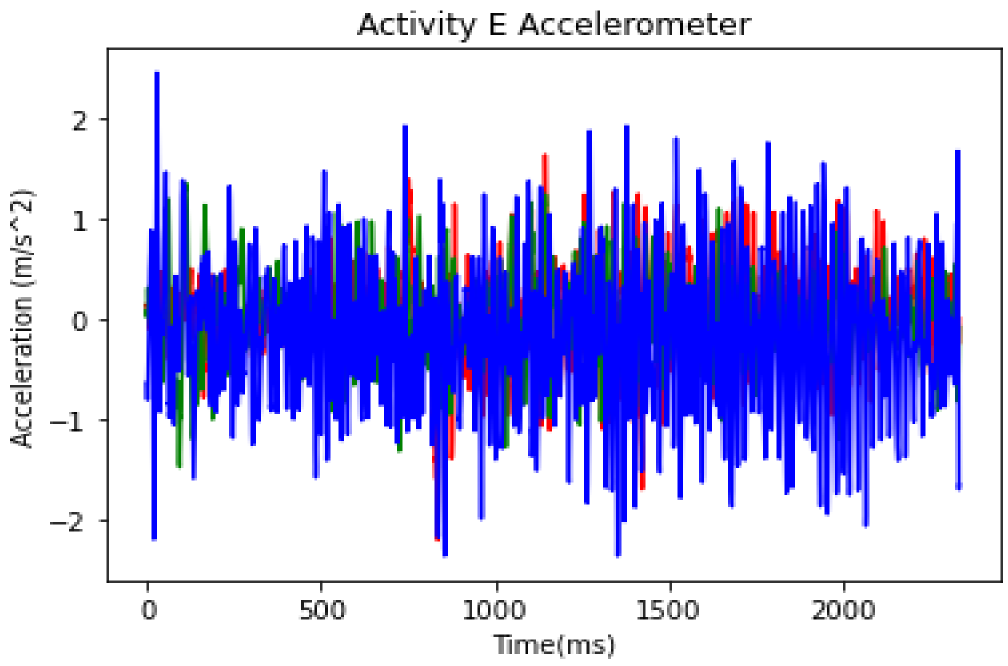


*Figure 12 - Plot of Activity E (Accelerometer)*

## Applying Machine Learning

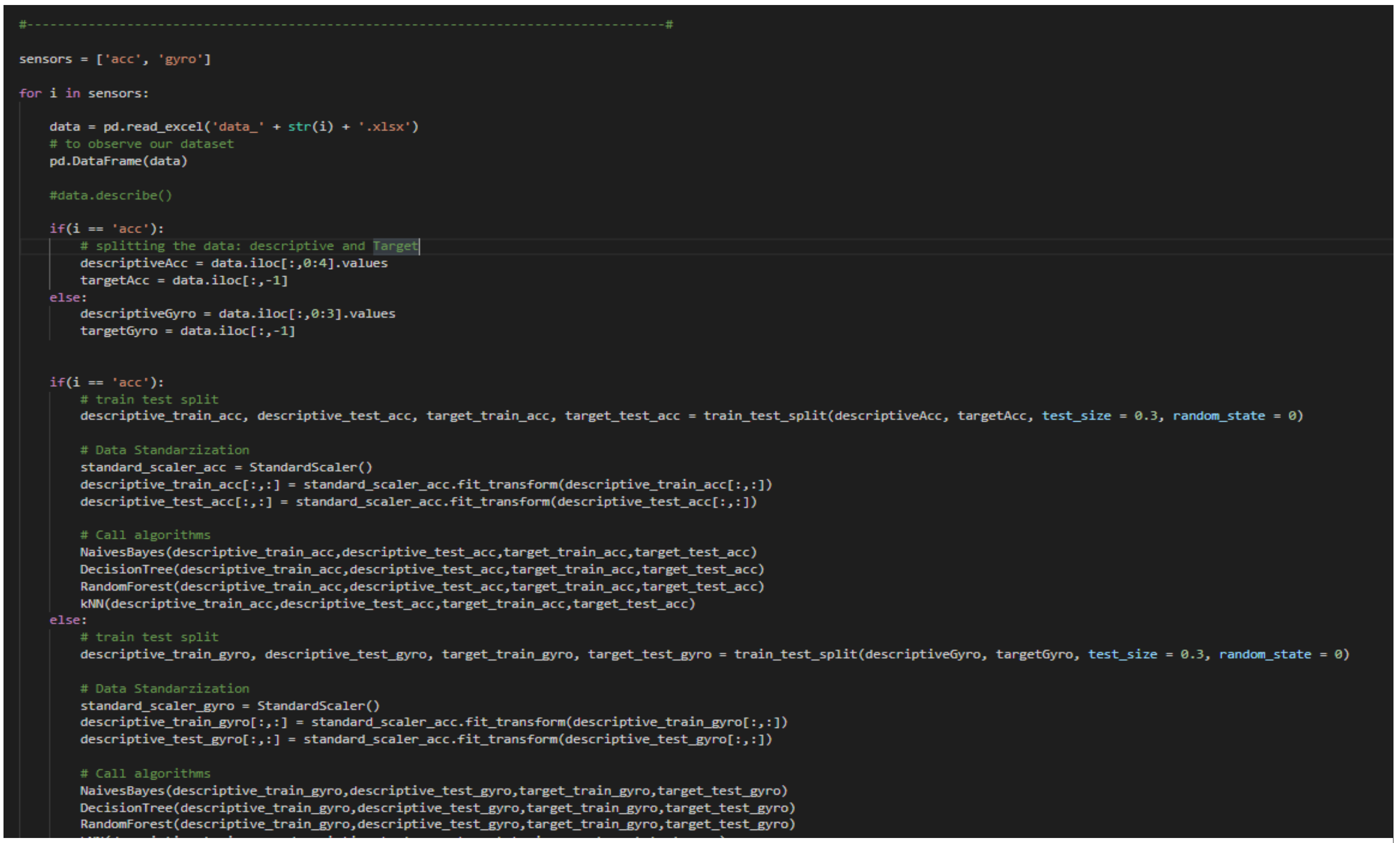
To apply machine learning to our extracted data, we had to prepare our files from "***rawData***" folder to start our analysis.

For each file (each activity for each sensor) we have deleted the 'Time' attribute (it is not relevant to the analysis that we needed to do), added a column for the corresponding activity and renamed the files in the following way: ***data_X_Y***, where X = [A…E] and Y = ['acc', 'gyro']. These files can be found the in folder named "***Data Preparation***".

After this preparation of our data, we combined all the data from accelerometer in one file (***data_acc.xlxs***) and the data from the gyroscope in another file (***data_gyro.xlxs***).
In order to shuffle all the data in each file, we used the function RAND() from excel to obtain a more realistic data set to apply machine learning algorithms. These files can be found in folder named "***Data Set***".

- ### Methods to apply Machine Learning

  In order to apply various algorithms of machine learning to our data sets, we have developed a script in python that reads the dataset, splits the data (descriptive and target), trains the test split, standardizes the data and call the various machine algorithms that we applied, depending on the sensor we want to evaluate

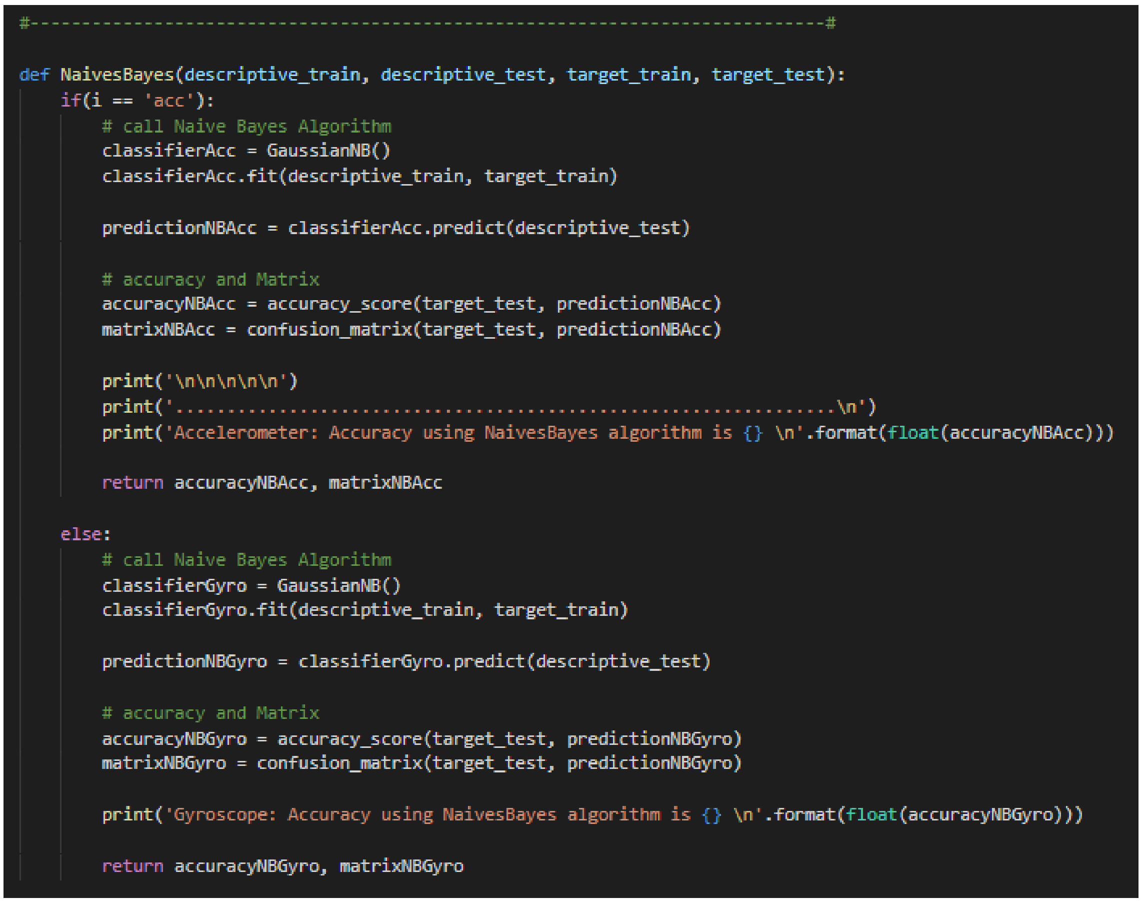


```
#---------------------------------------------------------------------------------#

sensors = ['acc', 'gyro']

for i in sensors:

    data = pd.read_excel('data_' + str(i) + '.xlsx')
    # to observe our dataset
    pd.DataFrame(data)

    #data.describe()

    if(i == 'acc'):
        # splitting the data: descriptive and Target
        descriptiveAcc = data.iloc[:,0:4].values
        targetAcc = data.iloc[:,-1]
    else:
        descriptiveGyro = data.iloc[:,0:3].values
        targetGyro = data.iloc[:,-1]


    if(i == 'acc'):
        # train test split
        descriptive_train_acc, descriptive_test_acc, target_train_acc, target_test_acc = train_test_split(descriptiveAcc, targetAcc, test_size = 0.3, random_state = 0)

        # Data Standarzization
        standard_scaler_acc = StandardScaler()
        descriptive_train_acc[:,:] = standard_scaler_acc.fit_transform(descriptive_train_acc[:,:])
        descriptive_test_acc[:,:] = standard_scaler_acc.fit_transform(descriptive_test_acc[:,:])

        # Call algorithms
        NaivesBayes(descriptive_train_acc,descriptive_test_acc,target_train_acc,target_test_acc)
        DecisionTree(descriptive_train_acc,descriptive_test_acc,target_train_acc,target_test_acc)
        RandomForest(descriptive_train_acc,descriptive_test_acc,target_train_acc,target_test_acc)
        kNN(descriptive_train_acc,descriptive_test_acc,target_train_acc,target_test_acc)
    else:
        # train test split
        descriptive_train_gyro, descriptive_test_gyro, target_train_gyro, target_test_gyro = train_test_split(descriptiveGyro, targetGyro, test_size = 0.3, random_state = 0)

        # Data Standarzization
        standard_scaler_gyro = StandardScaler()
        descriptive_train_gyro[:,:] = standard_scaler_acc.fit_transform(descriptive_train_gyro[:,:])
        descriptive_test_gyro[:,:] = standard_scaler_acc.fit_transform(descriptive_test_gyro[:,:])

        # Call algorithms
        NaivesBayes(descriptive_train_gyro,descriptive_test_gyro,target_train_gyro,target_test_gyro)
        DecisionTree(descriptive_train_gyro,descriptive_test_gyro,target_train_gyro,target_test_gyro)
        RandomForest(descriptive_train_gyro,descriptive_test_gyro,target_train_gyro,target_test_gyro)
```

*Figure 13- Main function of appliedMl.py script*

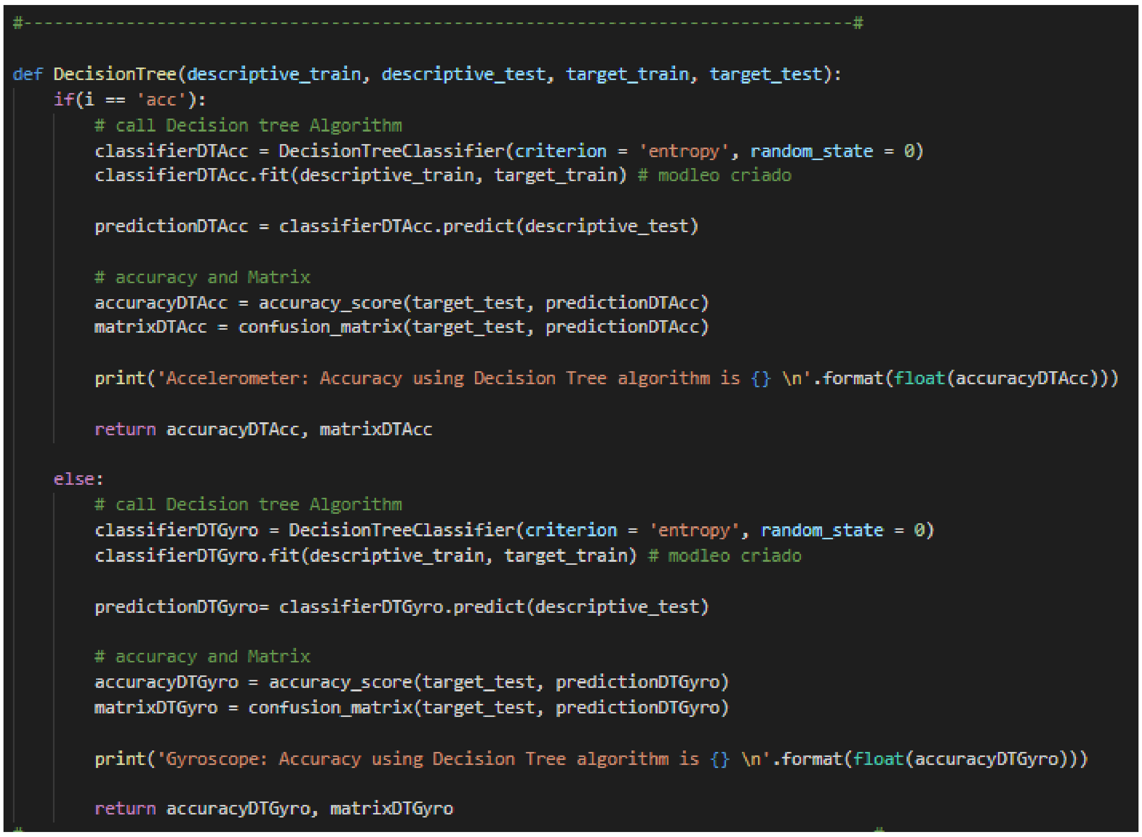
(accelerometer or gyroscope). The script is named ***appliedML.py*** and can be found in "***DataSet***". The code that implements these features is presented below:

To test the Machine Learning Model, we use 30% of our dataset, like is shown in the picture above.
The algorithms that we have chosen to use to analyse our dataset are: Naives Bayes, Decision Tree, Random Forest and kNN. For each algorithm, we have implemented a function that return the accuracy values and confusion matrix. But the values that we really want is the accuracy values. These values will be the ones that will helps us analyse our results.
The implementation of the functions for each algorithm is presented below:



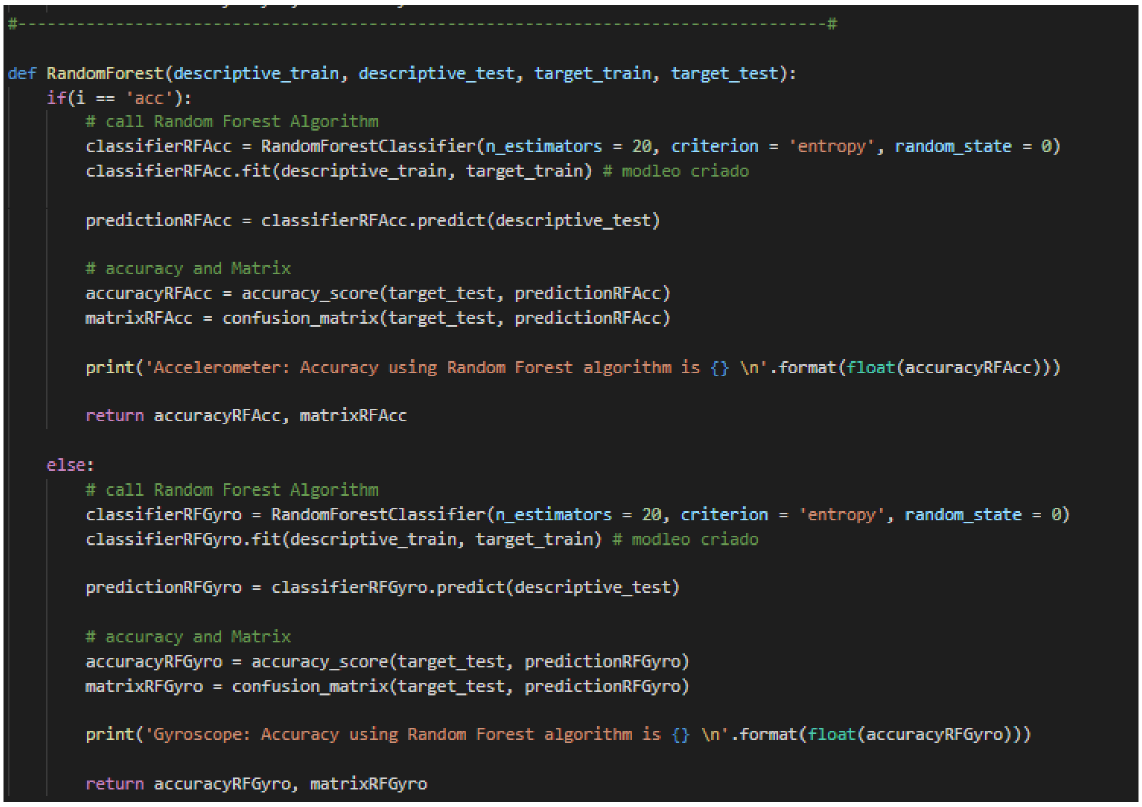
```
#----------------------------------------------------------------------------#

def NaivesBayes(descriptive_train, descriptive_test, target_train, target_test):
    if(i == 'acc'):
        # call Naive Bayes Algorithm
        classifierAcc = GaussianNB()
        classifierAcc.fit(descriptive_train, target_train)

        predictionNBAcc = classifierAcc.predict(descriptive_test)

        # accuracy and Matrix
        accuracyNBAcc = accuracy_score(target_test, predictionNBAcc)
        matrixNBAcc = confusion_matrix(target_test, predictionNBAcc)

        print('\n\n\n\n\n')
        print('.................................................................\n')
        print('Accelerometer: Accuracy using NaivesBayes algorithm is {} \n'.format(float(accuracyNBAcc)))

        return accuracyNBAcc, matrixNBAcc

    else:
        # call Naive Bayes Algorithm
        classifierGyro = GaussianNB()
        classifierGyro.fit(descriptive_train, target_train)

        predictionNBGyro = classifierGyro.predict(descriptive_test)

        # accuracy and Matrix
        accuracyNBGyro = accuracy_score(target_test, predictionNBGyro)
        matrixNBGyro = confusion_matrix(target_test, predictionNBGyro)

        print('Gyroscope: Accuracy using NaivesBayes algorithm is {} \n'.format(float(accuracyNBGyro)))

        return accuracyNBGyro, matrixNBGyro
```

*Figure 14 - NaivesBayes function in appliedML.py*



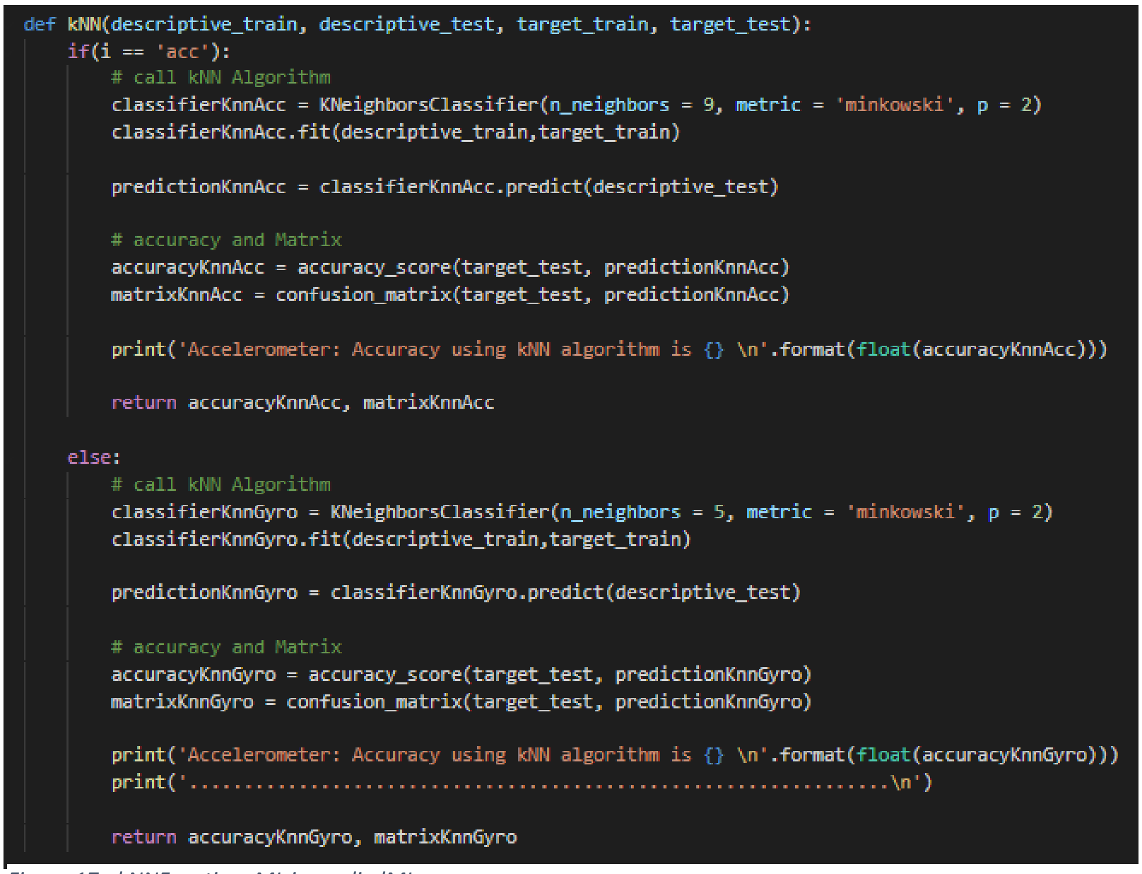
```
#-------------------------------------------------------------------------------#

def DecisionTree(descriptive_train, descriptive_test, target_train, target_test):
    if(i == 'acc'):
        # call Decision tree Algorithm
        classifierDTAcc = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
        classifierDTAcc.fit(descriptive_train, target_train) # modleo criado

        predictionDTAcc = classifierDTAcc.predict(descriptive_test)

        # accuracy and Matrix
        accuracyDTAcc = accuracy_score(target_test, predictionDTAcc)
        matrixDTAcc = confusion_matrix(target_test, predictionDTAcc)

        print('Accelerometer: Accuracy using Decision Tree algorithm is {} \n'.format(float(accuracyDTAcc)))

        return accuracyDTAcc, matrixDTAcc

    else:
        # call Decision tree Algorithm
        classifierDTGyro = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
        classifierDTGyro.fit(descriptive_train, target_train) # modleo criado

        predictionDTGyro= classifierDTGyro.predict(descriptive_test)

        # accuracy and Matrix
        accuracyDTGyro = accuracy_score(target_test, predictionDTGyro)
        matrixDTGyro = confusion_matrix(target_test, predictionDTGyro)

        print('Gyroscope: Accuracy using Decision Tree algorithm is {} \n'.format(float(accuracyDTGyro)))

        return accuracyDTGyro, matrixDTGyro
```

*Figure 15- DecisionTree function in appliedML.py*



```
#-------------------------------------------------------------------------------#

def RandomForest(descriptive_train, descriptive_test, target_train, target_test):
    if(i == 'acc'):
        # call Random Forest Algorithm
        classifierRFAcc = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', random_state = 0)
        classifierRFAcc.fit(descriptive_train, target_train) # modleo criado

        predictionRFAcc = classifierRFAcc.predict(descriptive_test)

        # accuracy and Matrix
        accuracyRFAcc = accuracy_score(target_test, predictionRFAcc)
        matrixRFAcc = confusion_matrix(target_test, predictionRFAcc)

        print('Accelerometer: Accuracy using Random Forest algorithm is {} \n'.format(float(accuracyRFAcc)))

        return accuracyRFAcc, matrixRFAcc

    else:
        # call Random Forest Algorithm
        classifierRFGyro = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', random_state = 0)
        classifierRFGyro.fit(descriptive_train, target_train) # modleo criado

        predictionRFGyro = classifierRFGyro.predict(descriptive_test)

        # accuracy and Matrix
        accuracyRFGyro = accuracy_score(target_test, predictionRFGyro)
        matrixRFGyro = confusion_matrix(target_test, predictionRFGyro)

        print('Gyroscope: Accuracy using Random Forest algorithm is {} \n'.format(float(accuracyRFGyro)))

        return accuracyRFGyro, matrixRFGyro
```

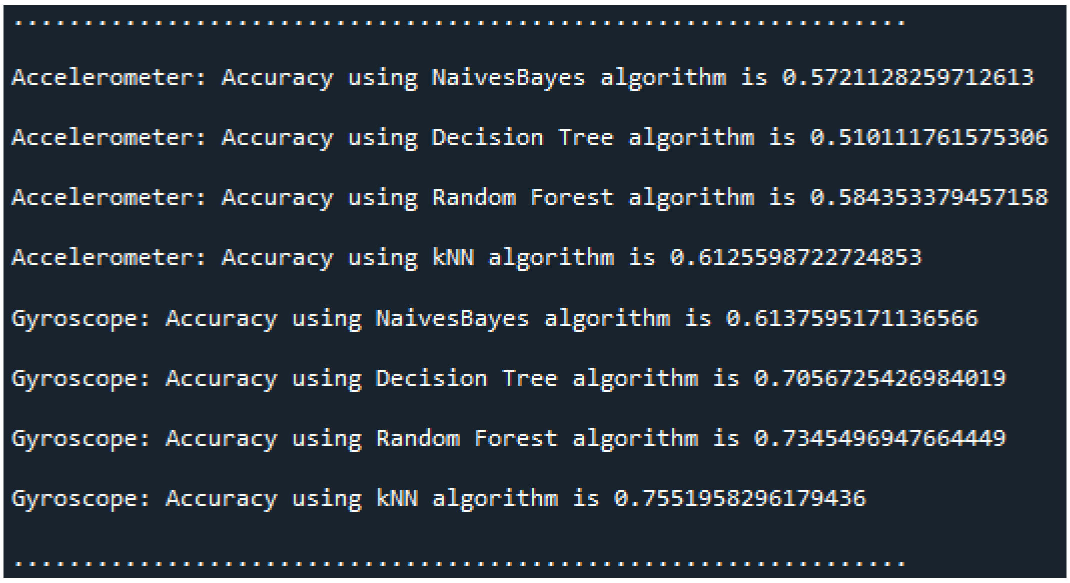
*Figure 16 - RandomForest function in appliedML.py*



```
def kNN(descriptive_train, descriptive_test, target_train, target_test):
    if(i == 'acc'):
        # call kNN Algorithm
        classifierKnnAcc = KNeighborsClassifier(n_neighbors = 9, metric = 'minkowski', p = 2)
        classifierKnnAcc.fit(descriptive_train,target_train)

        predictionKnnAcc = classifierKnnAcc.predict(descriptive_test)

        # accuracy and Matrix
        accuracyKnnAcc = accuracy_score(target_test, predictionKnnAcc)
        matrixKnnAcc = confusion_matrix(target_test, predictionKnnAcc)

        print('Accelerometer: Accuracy using kNN algorithm is {} \n'.format(float(accuracyKnnAcc)))

        return accuracyKnnAcc, matrixKnnAcc

    else:
        # call kNN Algorithm
        classifierKnnGyro = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
        classifierKnnGyro.fit(descriptive_train,target_train)

        predictionKnnGyro = classifierKnnGyro.predict(descriptive_test)

        # accuracy and Matrix
        accuracyKnnGyro = accuracy_score(target_test, predictionKnnGyro)
        matrixKnnGyro = confusion_matrix(target_test, predictionKnnGyro)

        print('Accelerometer: Accuracy using kNN algorithm is {} \n'.format(float(accuracyKnnGyro)))
        print('................................................................\n')

        return accuracyKnnGyro, matrixKnnGyro
```

*Figure 17 - kNNFucntion_ML in appliedML.py*

The results that this script (***appliedML.py***) returns are the values of accuracy for each sensor and for each algorithm, like shown below:



```
................................................................

Accelerometer: Accuracy using NaivesBayes algorithm is 0.5721128259712613

Accelerometer: Accuracy using Decision Tree algorithm is 0.510111761575306

Accelerometer: Accuracy using Random Forest algorithm is 0.584353379457158

Accelerometer: Accuracy using kNN algorithm is 0.6125598722724853

Gyroscope: Accuracy using NaivesBayes algorithm is 0.6137595171136566

Gyroscope: Accuracy using Decision Tree algorithm is 0.7056725426984019

Gyroscope: Accuracy using Random Forest algorithm is 0.7345496947664449

Gyroscope: Accuracy using kNN algorithm is 0.7551958296179436

................................................................
```

*Figure 18 - Final results (Accuracy)*

## Results and Conclusions

The results coming from ***appliedML.py*** are the following:

| **Algorithm\|Sensor** | **Accelerometer** | **Gyroscope** |
|---|---|---|
| **Naives Bayes** | 0.5721128259712613 | 0.6137595171136566 |
| **Decision Tree** | 0.510111761575306 | 0.7056725426984019 |
| **Random Forest** | 0.584353379457158 | 0.7345496947664449 |
| **kNN** | 0.6125598722724853 | 0.7551958296179436 |

*Table 1 - Results from our experimental work*

The conclusions that we take of our experimental work, based on the accuracy levels of the machine learning algorithms that we applied, is that the kNN algorithm gives us the best results, i.e., since the values of accuracy are higher than the other algorithms, it means that using this algorithm our prediction is more correct than using the others.
Using the kNN algorithm, the prediction in the **accelerometer** is ***61.26%*** correct and for the **gyroscope** is ***75.52%*** correct.