PROJECT INCREMENT-2 REPORT

Python & Deep Learning

Smartphone Based Human Activity Recognition

TEAM:05

MEMBERS:

CHAKRADHAR CHINNAM

VENKATA SIVARAM PABOLU

ADITYA VELUGULA

Human Activity Recognition:

Problem of predicting what a person is doing based on a trace of their movement using sensors. Sensors are often located on the subject, such as a smartphone or vest, and often record accelerometer data in three dimensions (x, y, z).We are using Deep learning methods such as recurrent neural networks and one-dimensional convolutional neural networks(CNNs).

Activity Recognition Using Smartphones Dataset:

Dataset was made available and can be downloaded for free from the UCI Machine Learning Repository.

The activities recorded in the dataset :

1. Walking
2. Sitting
3. Laying
4. Standing
5. Walking Upstairs
6. Walking Downstairs

Methods used:

In this project, we are planning to use four algorithms namely Support Vector Classifier, Logistic Regression ,K Nearest Neighbors Classifier ,Random Forest Classifier.

Why SVM:

SVM model works well when there is clear margin of separation between classes and is relatively memory efficient. It is more effective in high dimensional spaces and in cases where number of dimensions is greater than the number of samples.

Why Logistic regression:

Logistic Regression performs well when the dataset is linearly separable and is efficient in high dimensional datasets as SVM model. It is easier to implement, interpret and very efficient to train.

Why KNN:

Robust with regard to the search space; for instance, classes don't have to be linearly separable. It is easier to implement as Logistic Regression. New data can be added seamlessly which will not impact the accuracy of the algorithm as no training is required before making predictions.

Why Random Forest:

Random Forest Classifier offer efficient estimates of the test error without incurring the cost of repeated model training associated with cross-validation. It works well with both categorical and continuous variables and can automatically handle missing values.

Download the Dataset:

Downloaded from the UCI Machine Learning repository.

In the downloaded folder there are “train” and “test” folders containing the split portions of data for modelling.

Loading the Data & Preprocessing Steps:

1. We developed the code for loading the dataset.

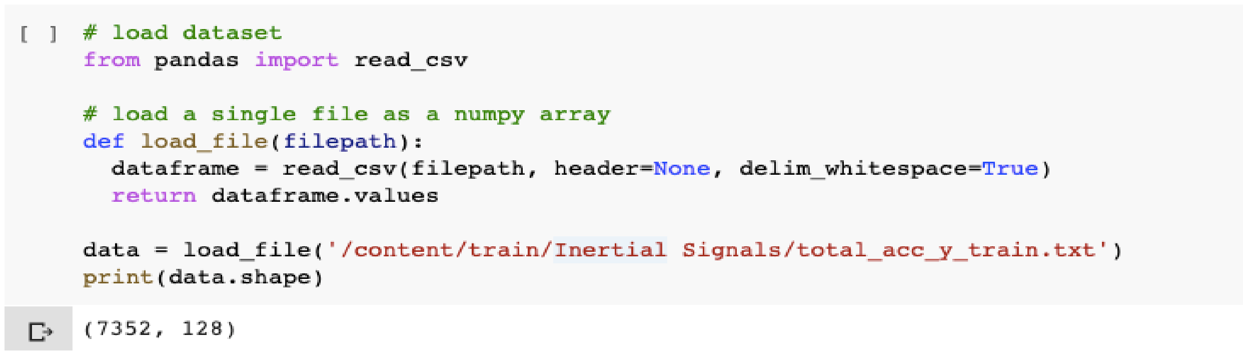
2.Initially as a trail we loaded a single data file to check how the dataset is functioning

3.read.csv() pandas function is used to load the data file and we specify that the file has no header and to separate columns using white space.

4.After loading each file as a NumPy array, all three arrays can be combined together. We use dstack() Numpy function to ensure that each array is combined so that the features get separated in the 3rd dimension.

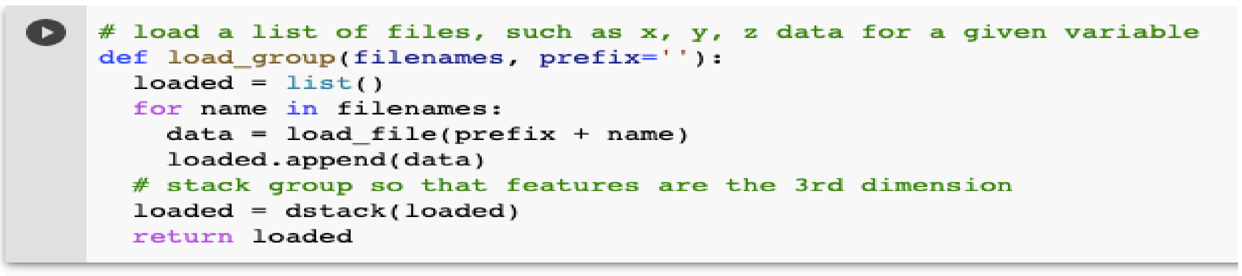
5.After loading the data we can perform training and testing on the data.

Loading Dataset for Analysing:

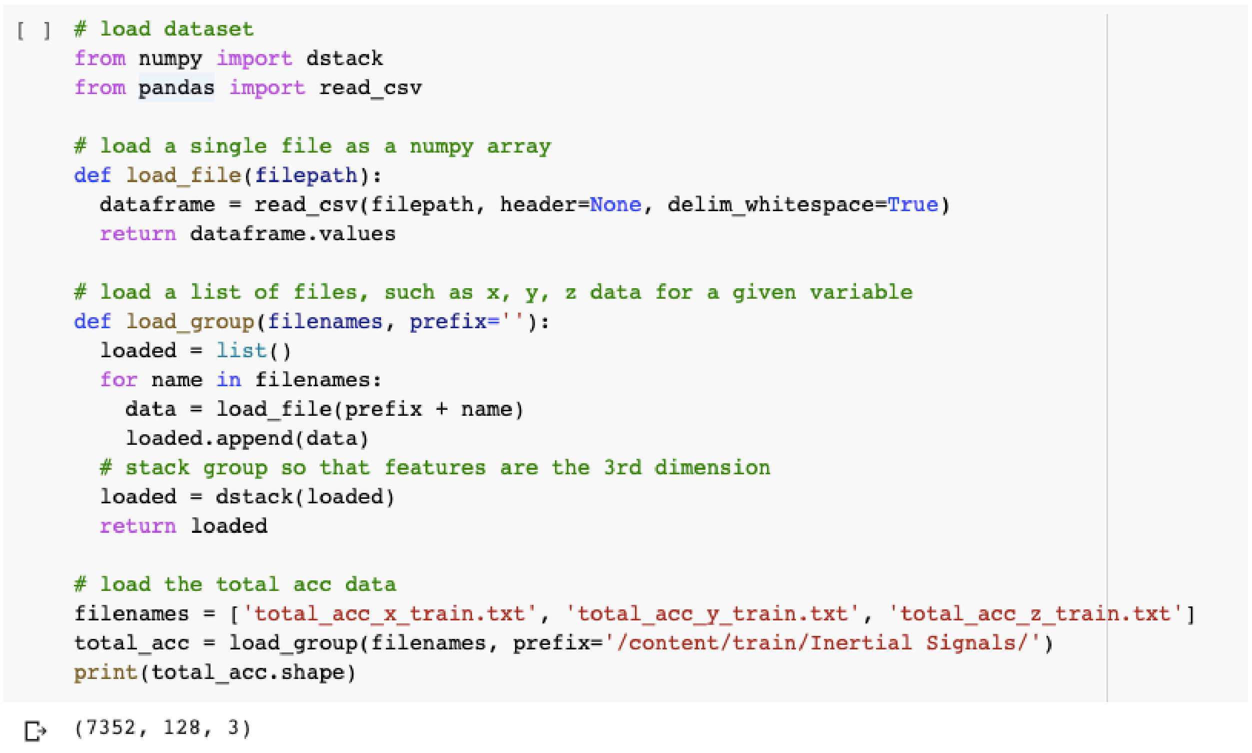


```
# load dataset
from pandas import read_csv

# load a single file as a numpy array
def load_file(filepath):
  dataframe = read_csv(filepath, header=None, delim_whitespace=True)
  return dataframe.values

data = load_file('/content/train/Inertial Signals/total_acc_y_train.txt')
print(data.shape)
```

```
(7352, 128)
```

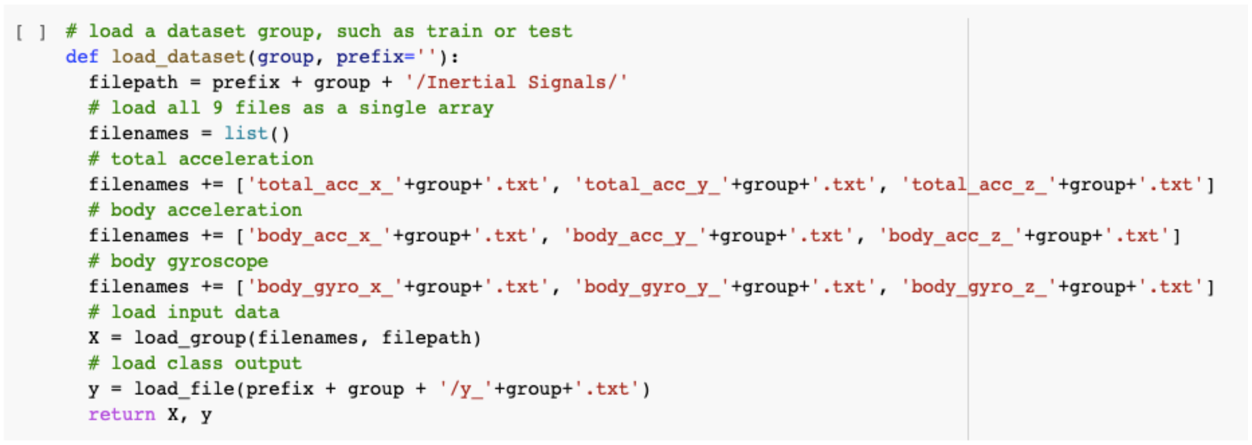


```
# load a list of files, such as x, y, z data for a given variable
def load_group(filenames, prefix=''):
  loaded = list()
  for name in filenames:
    data = load_file(prefix + name)
    loaded.append(data)
  # stack group so that features are the 3rd dimension
  loaded = dstack(loaded)
  return loaded
```



```
# load dataset
from numpy import dstack
from pandas import read_csv

# load a single file as a numpy array
def load_file(filepath):
  dataframe = read_csv(filepath, header=None, delim_whitespace=True)
  return dataframe.values

# load a list of files, such as x, y, z data for a given variable
def load_group(filenames, prefix=''):
  loaded = list()
  for name in filenames:
    data = load_file(prefix + name)
    loaded.append(data)
  # stack group so that features are the 3rd dimension
  loaded = dstack(loaded)
  return loaded

# load the total acc data
filenames = ['total_acc_x_train.txt', 'total_acc_y_train.txt', 'total_acc_z_train.txt']
total_acc = load_group(filenames, prefix='/content/train/Inertial Signals/')
print(total_acc.shape)
```

```
(7352, 128, 3)
```



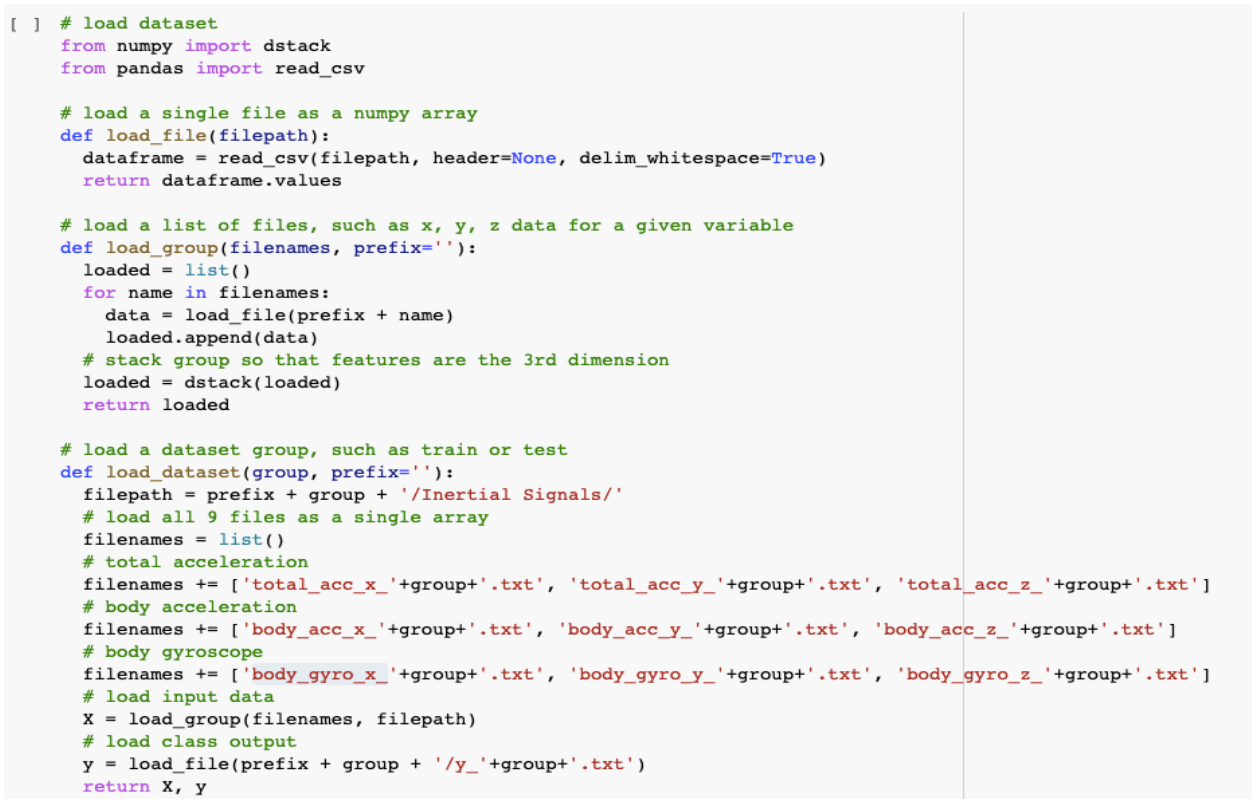
```python
# load a dataset group, such as train or test
def load_dataset(group, prefix=''):
  filepath = prefix + group + '/Inertial Signals/'
  # load all 9 files as a single array
  filenames = list()
  # total acceleration
  filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt', 'total_acc_z_'+group+'.txt']
  # body acceleration
  filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt', 'body_acc_z_'+group+'.txt']
  # body gyroscope
  filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt', 'body_gyro_z_'+group+'.txt']
  # load input data
  X = load_group(filenames, filepath)
  # load class output
  y = load_file(prefix + group + '/y_'+group+'.txt')
  return X, y
```

The “Inertial Signals” directory contains 9 files.

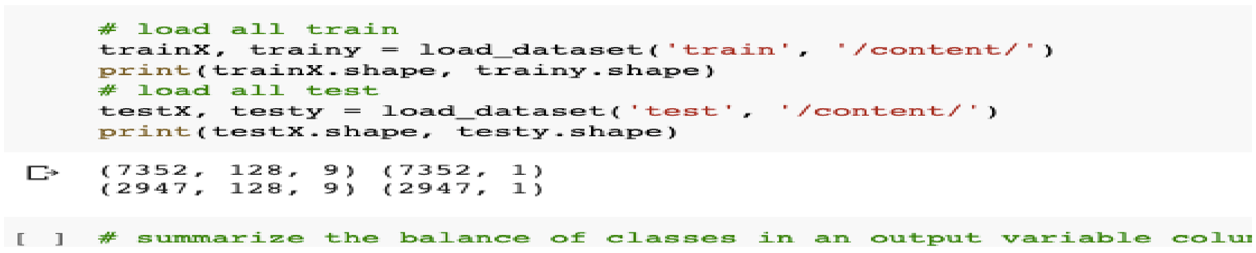
1.Gravitational acceleration data files for x, y and z
axes: total_acc_x_train.txt, total_acc_y_train.txt, total_acc_z_train.txt.

2.Body acceleration data files for x, y and z
axes: body_acc_x_train.txt, body_acc_y_train.txt, body_acc_z_train.txt.

3.Body gyroscope data files for x, y and z
axes: body_gyro_x_train.txt, body_gyro_y_train.txt, body_gyro_z_train.txt



```python
# load dataset
from numpy import dstack
from pandas import read_csv

# load a single file as a numpy array
def load_file(filepath):
  dataframe = read_csv(filepath, header=None, delim_whitespace=True)
  return dataframe.values

# load a list of files, such as x, y, z data for a given variable
def load_group(filenames, prefix=''):
  loaded = list()
  for name in filenames:
    data = load_file(prefix + name)
    loaded.append(data)
  # stack group so that features are the 3rd dimension
  loaded = dstack(loaded)
  return loaded

# load a dataset group, such as train or test
def load_dataset(group, prefix=''):
  filepath = prefix + group + '/Inertial Signals/'
  # load all 9 files as a single array
  filenames = list()
  # total acceleration
  filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt', 'total_acc_z_'+group+'.txt']
  # body acceleration
  filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt', 'body_acc_z_'+group+'.txt']
  # body gyroscope
  filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt', 'body_gyro_z_'+group+'.txt']
  # load input data
  X = load_group(filenames, filepath)
  # load class output
  y = load_file(prefix + group + '/y_'+group+'.txt')
  return X, y
```



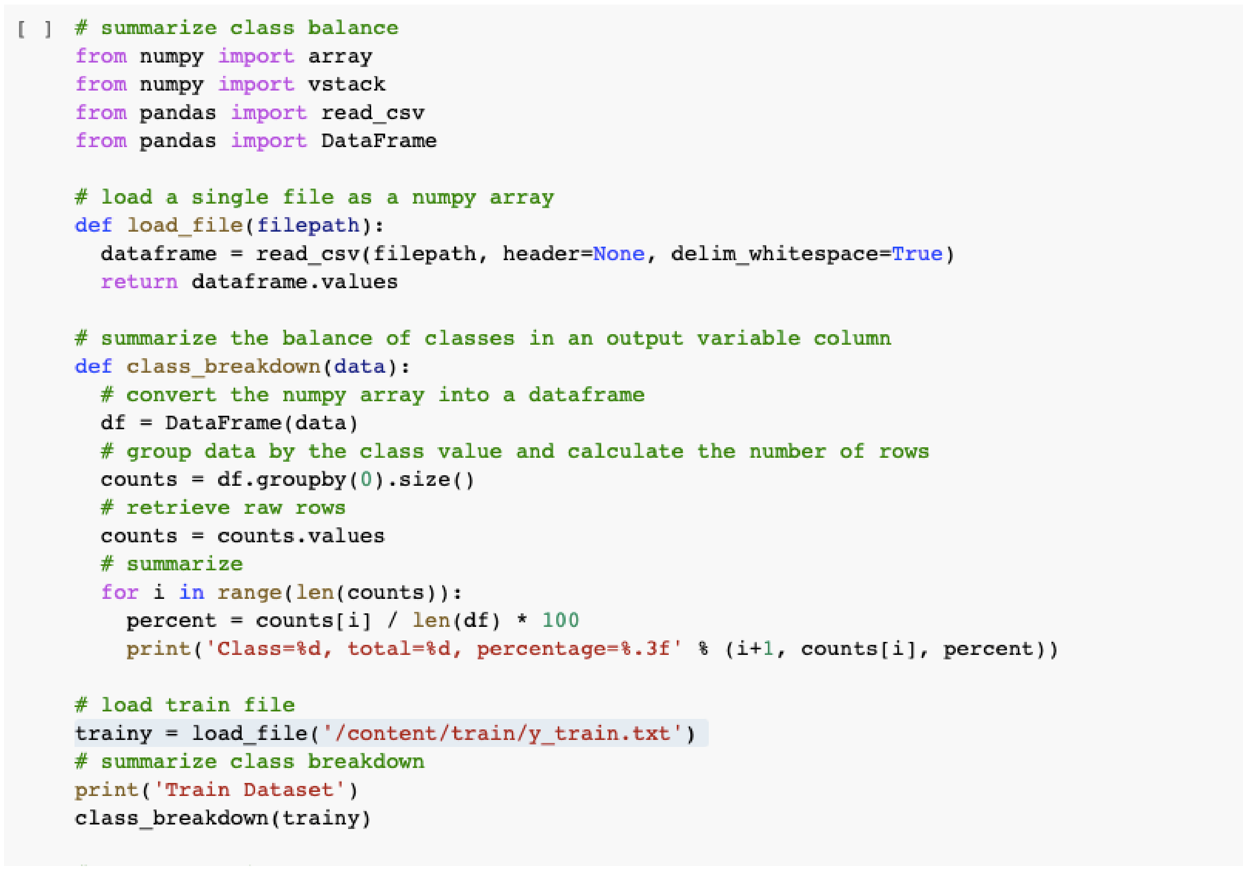
```
# load all train
trainX, trainy = load_dataset('train', '/content/')
print(trainX.shape, trainy.shape)
# load all test
testX, testy = load_dataset('test', '/content/')
print(testX.shape, testy.shape)
```

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
```

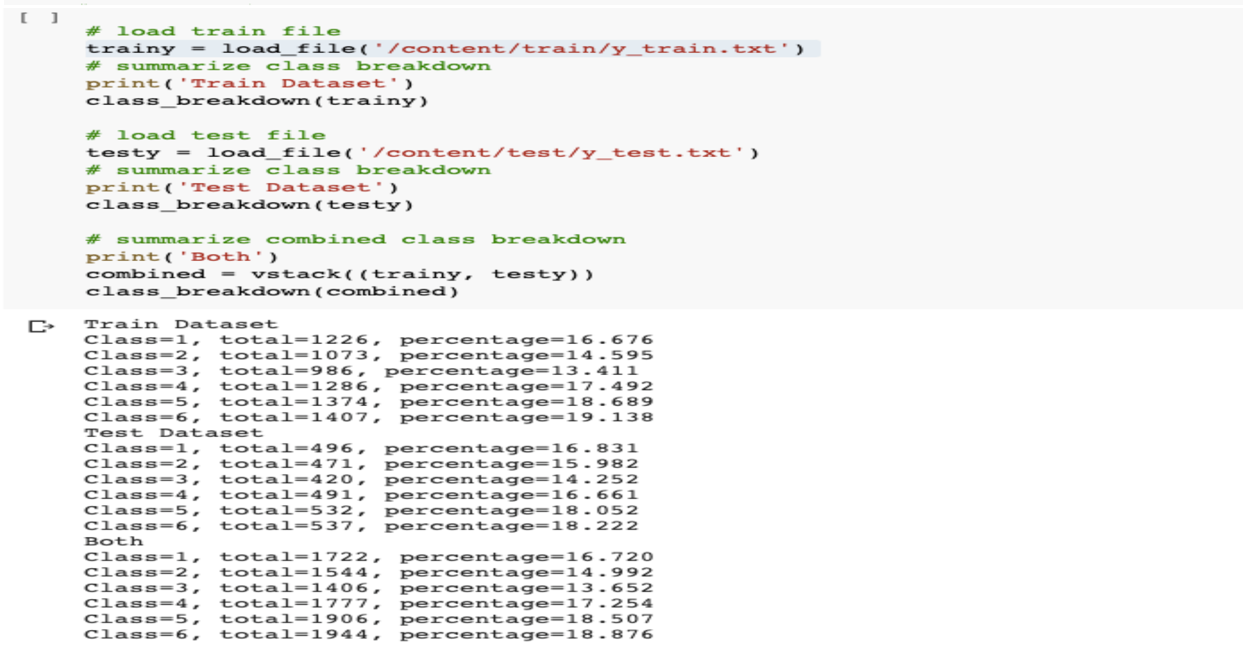
```
[ ] # summarize the balance of classes in an output variable colu
```

Balance of Activity Classes:

Data is to be verified or checked to maintain the class balance of each activity. We need to make sure that each of the 30 subjects performed each of the six activities. This helps the model to make sure that the data is loading correctly and interpreting the data.



```
[ ]  # summarize class balance
     from numpy import array
     from numpy import vstack
     from pandas import read_csv
     from pandas import DataFrame

     # load a single file as a numpy array
     def load_file(filepath):
       dataframe = read_csv(filepath, header=None, delim_whitespace=True)
       return dataframe.values

     # summarize the balance of classes in an output variable column
     def class_breakdown(data):
       # convert the numpy array into a dataframe
       df = DataFrame(data)
       # group data by the class value and calculate the number of rows
       counts = df.groupby(0).size()
       # retrieve raw rows
       counts = counts.values
       # summarize
       for i in range(len(counts)):
         percent = counts[i] / len(df) * 100
         print('Class=%d, total=%d, percentage=%.3f' % (i+1, counts[i], percent))

     # load train file
     trainy = load_file('/content/train/y_train.txt')
     # summarize class breakdown
     print('Train Dataset')
     class_breakdown(trainy)
```



```
[ ]  # load train file
     trainy = load_file('/content/train/y_train.txt')
     # summarize class breakdown
     print('Train Dataset')
     class_breakdown(trainy)

     # load test file
     testy = load_file('/content/test/y_test.txt')
     # summarize class breakdown
     print('Test Dataset')
     class_breakdown(testy)

     # summarize combined class breakdown
     print('Both')
     combined = vstack((trainy, testy))
     class_breakdown(combined)
```

```
Train Dataset
Class=1, total=1226, percentage=16.676
Class=2, total=1073, percentage=14.595
Class=3, total=986, percentage=13.411
Class=4, total=1286, percentage=17.492
Class=5, total=1374, percentage=18.689
Class=6, total=1407, percentage=19.138
Test Dataset
Class=1, total=496, percentage=16.831
Class=2, total=471, percentage=15.982
Class=3, total=420, percentage=14.252
Class=4, total=491, percentage=16.661
Class=5, total=532, percentage=18.052
Class=6, total=537, percentage=18.222
Both
Class=1, total=1722, percentage=16.720
Class=2, total=1544, percentage=14.992
Class=3, total=1406, percentage=13.652
Class=4, total=1777, percentage=17.254
Class=5, total=1906, percentage=18.507
Class=6, total=1944, percentage=18.876
```

Plot Time Series Data for One Subject

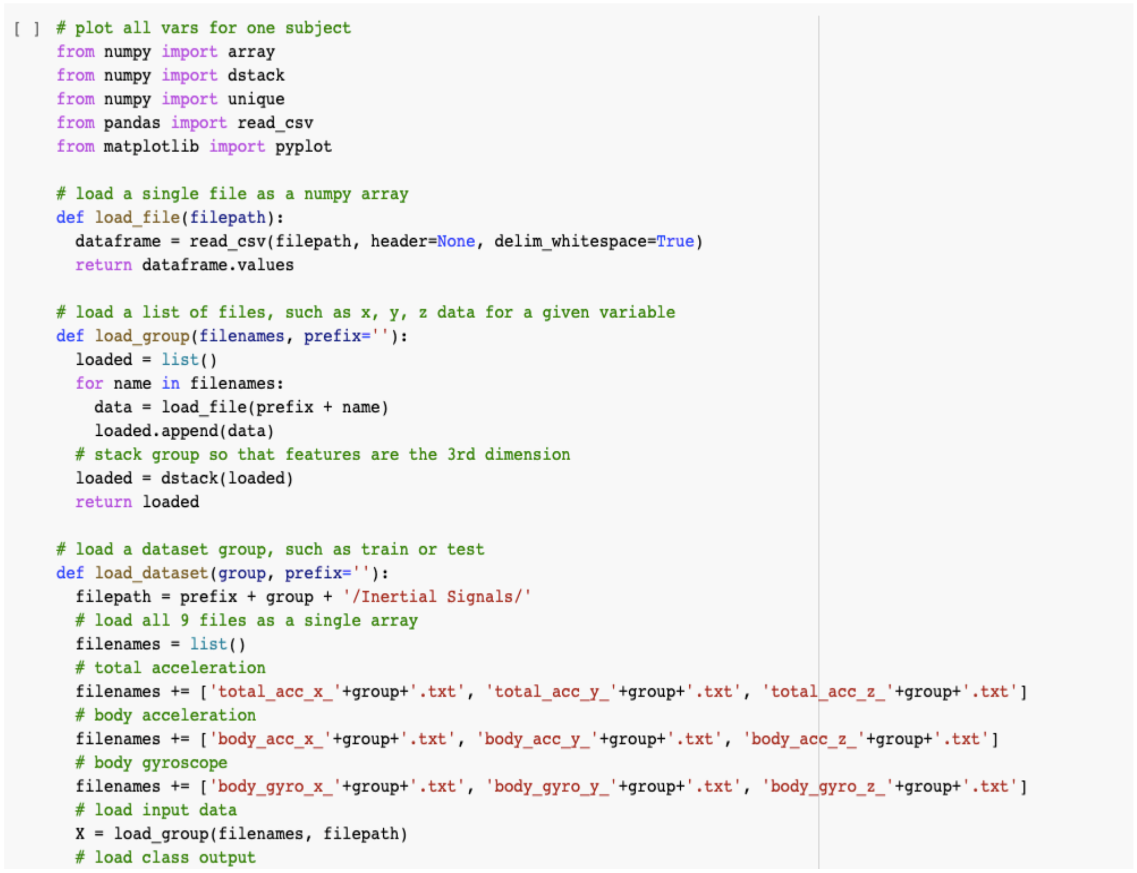
As in time series we need to import check on raw data.Raw data is in forms of windows of time series data per variable there will be overlap of data or repetition around 50%. To overcome this we need to remove the overlap.We can use the same for plotting another subject by changing its identifier number.

```python
# load data
trainX, trainy = load_dataset('train', '/content/')
```

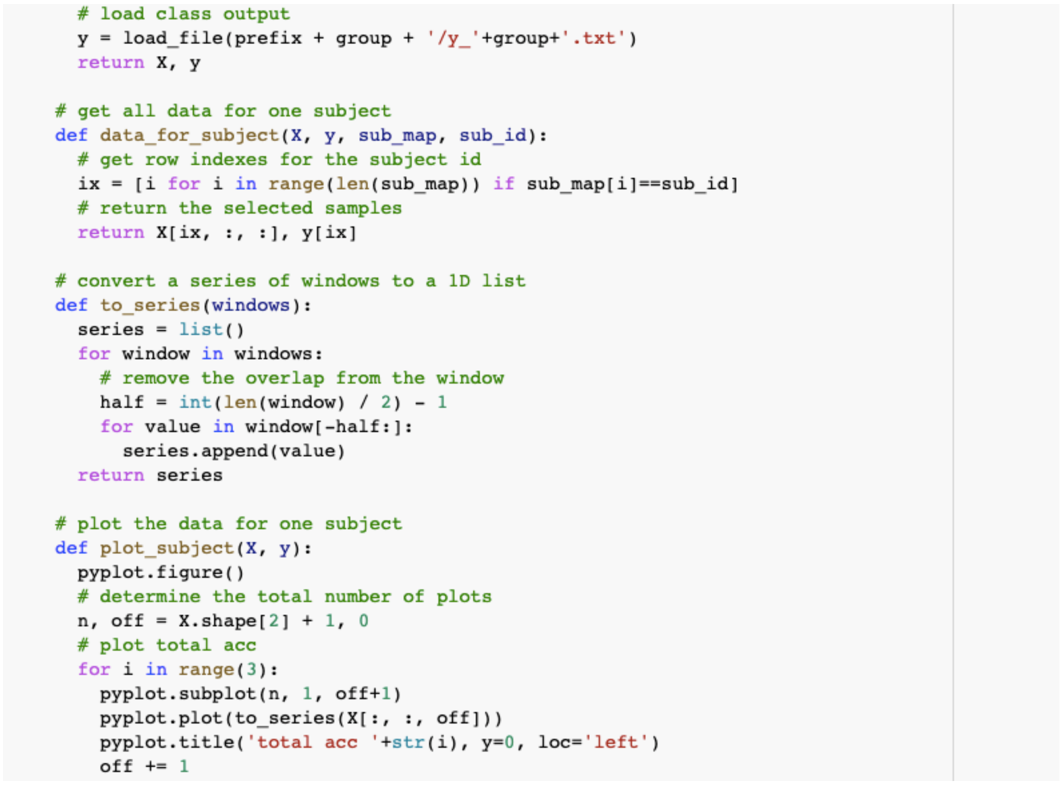
```python
# get all data for one subject
def data_for_subject(X, y, sub_map, sub_id):
  # get row indexes for the subject id
  ix = [i for i in range(len(sub_map)) if sub_map[i]==sub_id]
  # return the selected samples
  return X[ix, :, :], y[ix]
```

```python
# convert a series of windows to a 1D list
def to_series(windows):
  series = list()
  for window in windows:
    # remove the overlap from the window
    half = int(len(window) / 2) - 1
    for value in window[-half:]:
      series.append(value)
  return series
```

```python
# plot the data for one subject
def plot_subject(X, y):
  pyplot.figure()
  # determine the total number of plots
  n, off = X.shape[2] + 1, 0
  # plot total acc
  for i in range(3):
    pyplot.subplot(n, 1, off+1)
    pyplot.plot(to_series(X[:, :, off]))
    pyplot.title('total acc '+str(i), y=0, loc='left')
    off += 1
  # plot body acc
  for i in range(3):
    pyplot.subplot(n, 1, off+1)
    pyplot.plot(to_series(X[:, :, off]))
    pyplot.title('body acc '+str(i), y=0, loc='left')
    off += 1
  # plot body gyro
  for i in range(3):
    pyplot.subplot(n, 1, off+1)
    pyplot.plot(to_series(X[:, :, off]))
    pyplot.title('body gyro '+str(i), y=0, loc='left')
    off += 1
  # plot activities
  pyplot.subplot(n, 1, n)
  pyplot.plot(y)
  pyplot.title('activity', y=0, loc='left')
  pyplot.show()
```



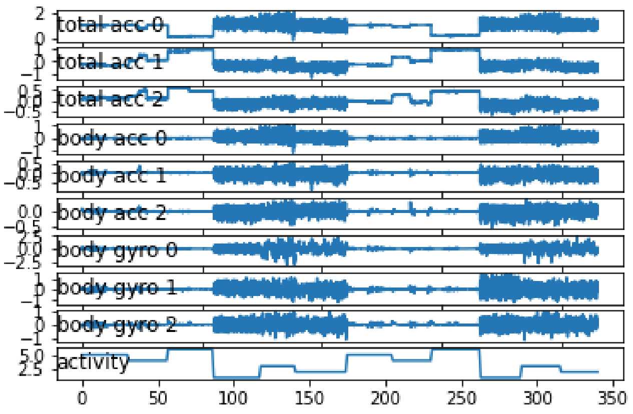
```
[ ] # plot all vars for one subject
    from numpy import array
    from numpy import dstack
    from numpy import unique
    from pandas import read_csv
    from matplotlib import pyplot

    # load a single file as a numpy array
    def load_file(filepath):
      dataframe = read_csv(filepath, header=None, delim_whitespace=True)
      return dataframe.values

    # load a list of files, such as x, y, z data for a given variable
    def load_group(filenames, prefix=''):
      loaded = list()
      for name in filenames:
        data = load_file(prefix + name)
        loaded.append(data)
      # stack group so that features are the 3rd dimension
      loaded = dstack(loaded)
      return loaded

    # load a dataset group, such as train or test
    def load_dataset(group, prefix=''):
      filepath = prefix + group + '/Inertial Signals/'
      # load all 9 files as a single array
      filenames = list()
      # total acceleration
      filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt', 'total_acc_z_'+group+'.txt']
      # body acceleration
      filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt', 'body_acc_z_'+group+'.txt']
      # body gyroscope
      filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt', 'body_gyro_z_'+group+'.txt']
      # load input data
      X = load_group(filenames, filepath)
      # load class output
```



```
  # load class output
  y = load_file(prefix + group + '/y_'+group+'.txt')
  return X, y

# get all data for one subject
def data_for_subject(X, y, sub_map, sub_id):
  # get row indexes for the subject id
  ix = [i for i in range(len(sub_map)) if sub_map[i]==sub_id]
  # return the selected samples
  return X[ix, :, :], y[ix]

# convert a series of windows to a 1D list
def to_series(windows):
  series = list()
  for window in windows:
    # remove the overlap from the window
    half = int(len(window) / 2) - 1
    for value in window[-half:]:
      series.append(value)
  return series

# plot the data for one subject
def plot_subject(X, y):
  pyplot.figure()
  # determine the total number of plots
  n, off = X.shape[2] + 1, 0
  # plot total acc
  for i in range(3):
    pyplot.subplot(n, 1, off+1)
    pyplot.plot(to_series(X[:, :, off]))
    pyplot.title('total acc '+str(i), y=0, loc='left')
    off += 1
  # plot total acc
  for i in range(3):
    pyplot.subplot(n, 1, off+1)
    pyplot.plot(to_series(X[:, :, off]))
    pyplot.title('total acc '+str(i), y=0, loc='left')
    off += 1
  # plot body acc
  for i in range(3):
    pyplot.subplot(n, 1, off+1)
    pyplot.plot(to_series(X[:, :, off]))
    pyplot.title('body acc '+str(i), y=0, loc='left')
    off += 1
  # plot body gyro
  for i in range(3):
    pyplot.subplot(n, 1, off+1)
    pyplot.plot(to_series(X[:, :, off]))
    pyplot.title('body gyro '+str(i), y=0, loc='left')
    off += 1
  # plot activities
  pyplot.subplot(n, 1, n)
  pyplot.plot(y)
  pyplot.title('activity', y=0, loc='left')
  pyplot.show()

# load data
trainX, trainy = load_dataset('train', '/content/')
# load mapping of rows to subjects
sub_map = load_file('/content/train/subject_train.txt')
train_subjects = unique(sub_map)
print(train_subjects)
# get the data for one subject
sub_id = train_subjects[0]
subX, suby = data_for_subject(trainX, trainy, sub_map, sub_id)
print(subX.shape, suby.shape)
```

```
[ ] # get the data for one subject
    sub_id = train_subjects[1]
    subX, suby = data_for_subject(trainX, trainy, sub_map, sub_id)
    print(subX.shape, suby.shape)
    # plot data for subject
    plot_subject(subX, suby)
```

```
(341, 128, 9) (341, 1)
```

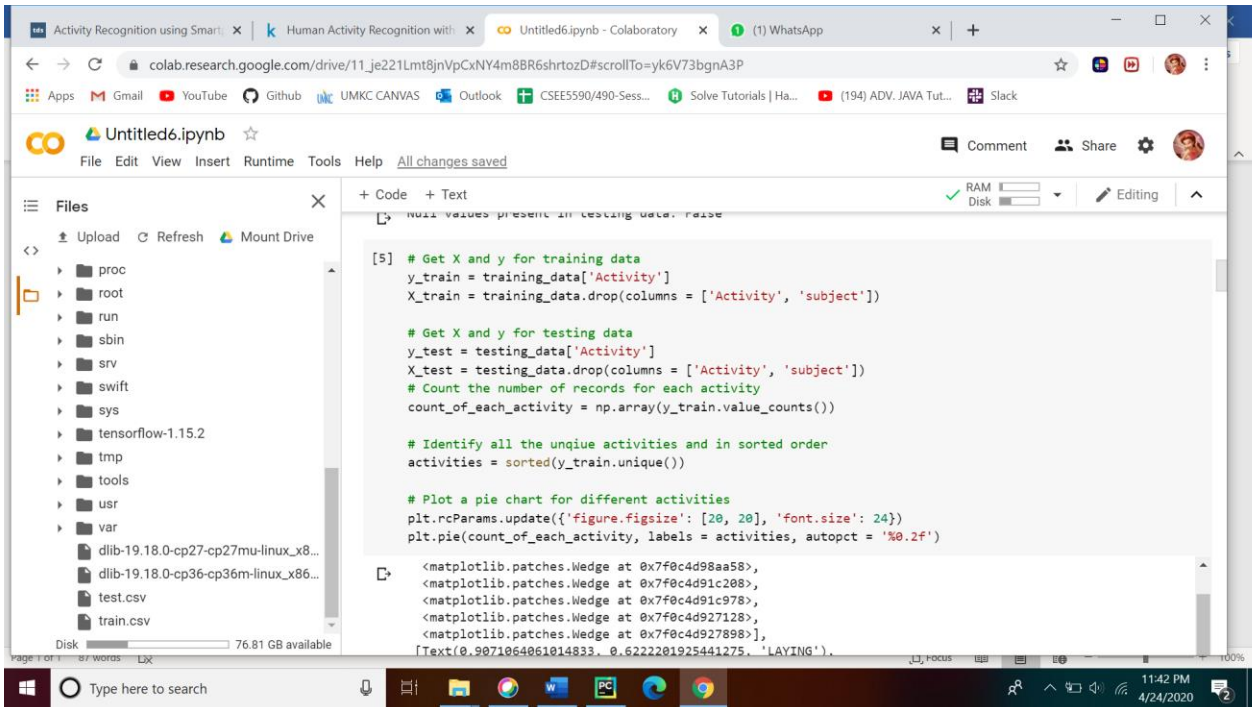


Visualization of dataset:

We should ensure that all classes are of approximately equal size to avoid biasing towards any classes when applying machine learning algorithms.

So, for that we have used pie char here. The first argument is the count for each activity, the second argument is the respective activity name denoted by labels and the third argument autopct computes percentages for each activity.

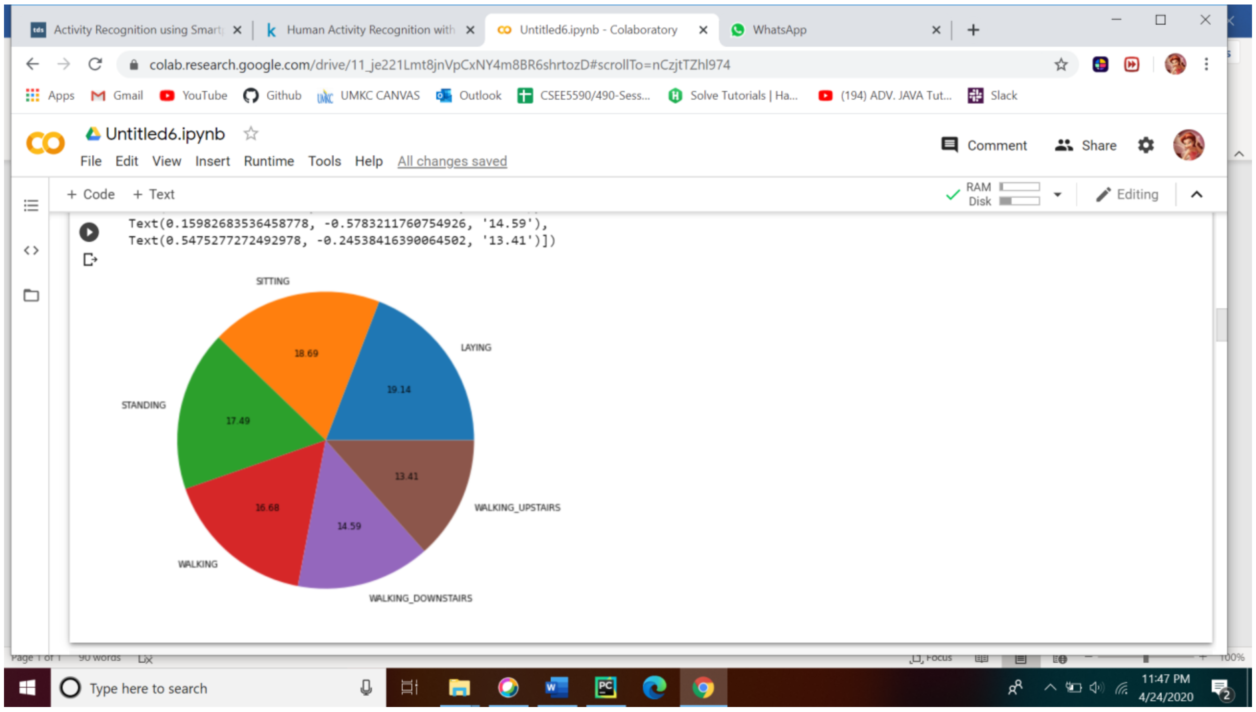
Code Snippet :



```
[5] # Get X and y for training data
    y_train = training_data['Activity']
    X_train = training_data.drop(columns = ['Activity', 'subject'])

    # Get X and y for testing data
    y_test = testing_data['Activity']
    X_test = testing_data.drop(columns = ['Activity', 'subject'])
    # Count the number of records for each activity
    count_of_each_activity = np.array(y_train.value_counts())

    # Identify all the unqiue activities and in sorted order
    activities = sorted(y_train.unique())

    # Plot a pie chart for different activities
    plt.rcParams.update({'figure.figsize': [20, 20], 'font.size': 24})
    plt.pie(count_of_each_activity, labels = activities, autopct = '%0.2f')
```

```
  <matplotlib.patches.Wedge at 0x7f0c4d98aa58>,
  <matplotlib.patches.Wedge at 0x7f0c4d91c208>,
  <matplotlib.patches.Wedge at 0x7f0c4d91c978>,
  <matplotlib.patches.Wedge at 0x7f0c4d927128>,
  <matplotlib.patches.Wedge at 0x7f0c4d927898>],
 [Text(0.9071064061014833, 0.6222201925441275, 'LAYING'),
```

Output:

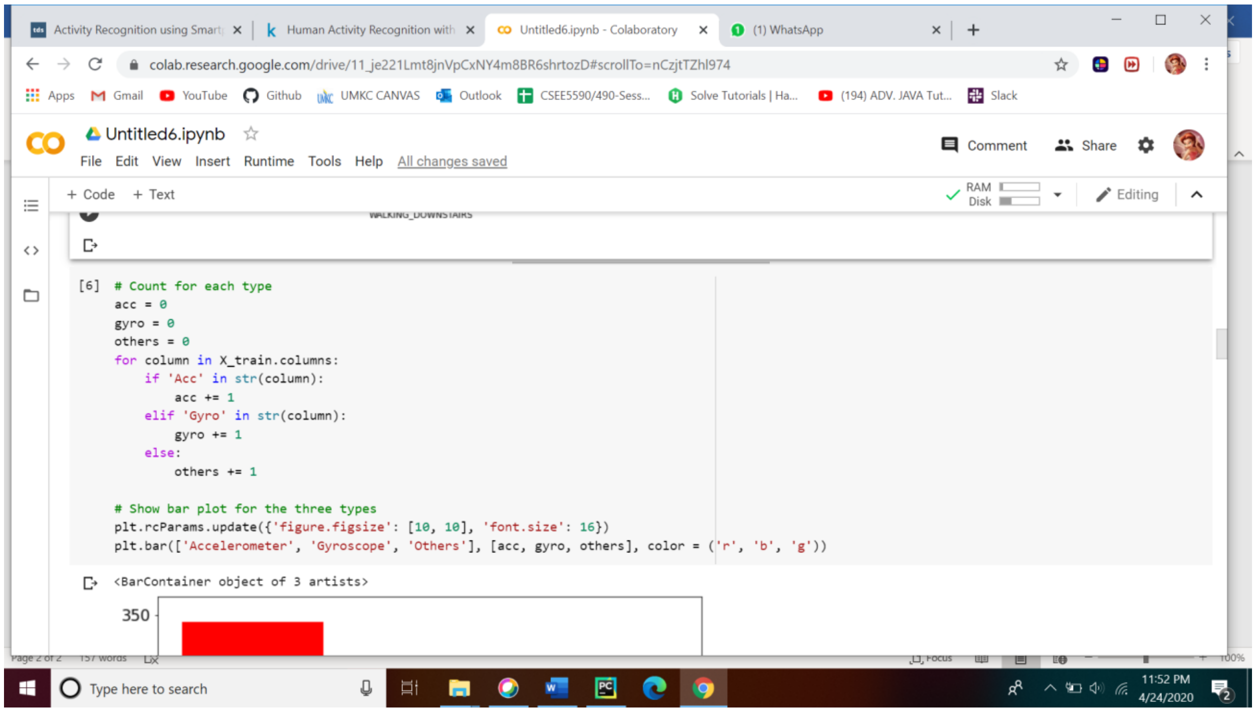


```
Text(0.15982683536458778, -0.5783211760754926, '14.59'),
Text(0.5475277272492978, -0.24538416390064502, '13.41')])
```

We can observe that the above pie chart and see that the share of each activity is almost equal. The dataset is very well formed and can be used directly.

Next, we can observe that the type of readings in the dataset. We have text Acc is referring to accelerometer reading ,Gyro is referring to gyroscope values or none of the two to refer to all others. Here,we have used bar plot using bar() method of pyplot subpackage. The first argument are the X axis labels, the second argument takes the array of Y axis values.

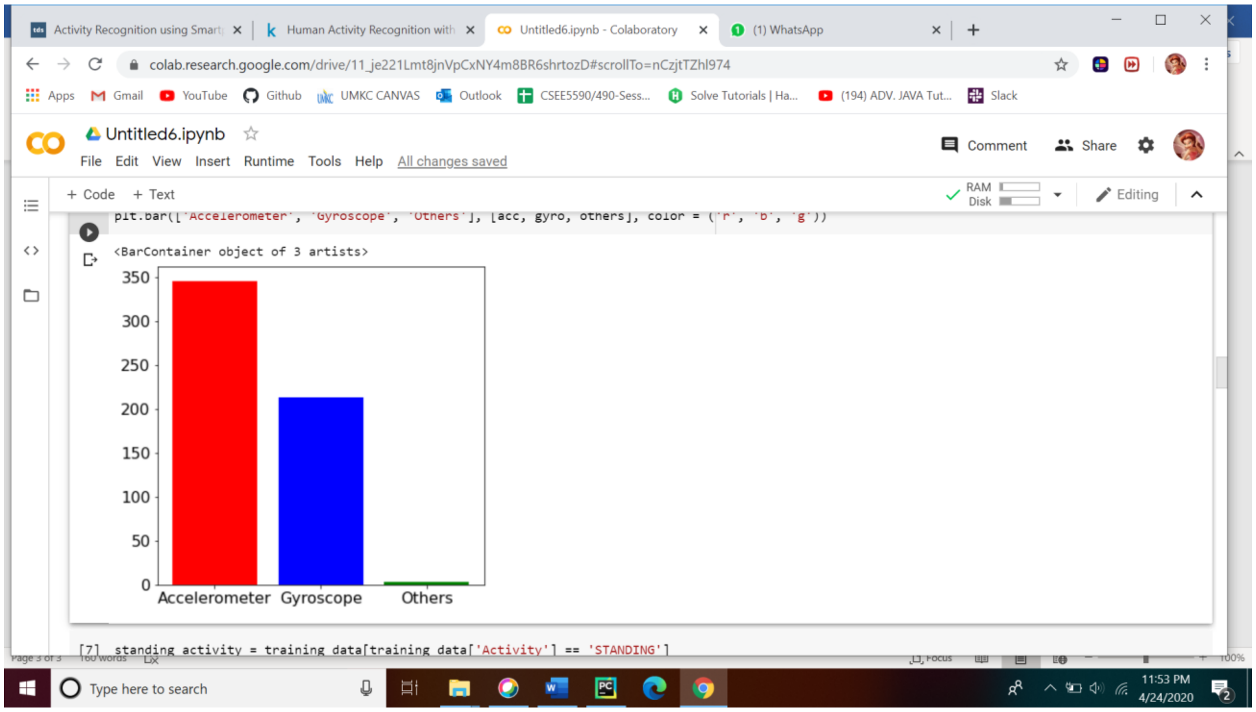
Code



```
[6] # Count for each type
    acc = 0
    gyro = 0
    others = 0
    for column in X_train.columns:
        if 'Acc' in str(column):
            acc += 1
        elif 'Gyro' in str(column):
            gyro += 1
        else:
            others += 1

    # Show bar plot for the three types
    plt.rcParams.update({'figure.figsize': [10, 10], 'font.size': 16})
    plt.bar(['Accelerometer', 'Gyroscope', 'Others'], [acc, gyro, others], color = ('r', 'b', 'g'))
```

```
<BarContainer object of 3 artists>
```

Output:



```
<BarContainer object of 3 artists>
```

```
[7] standing_activity = training_data[training_data['Activity'] == 'STANDING']
```

Inspecting a particular activity:

The dataset has activity records for 30 individuals and some individual performed the Standing activity.

The data collected is recorded at a stretch for each individual, especially for each activity. This means the records of any given activity will actually be in time series.

1. There are many features in each record, but we must not include all in one attempt as it might make the understanding of the data more difficult.

Standing Activity:

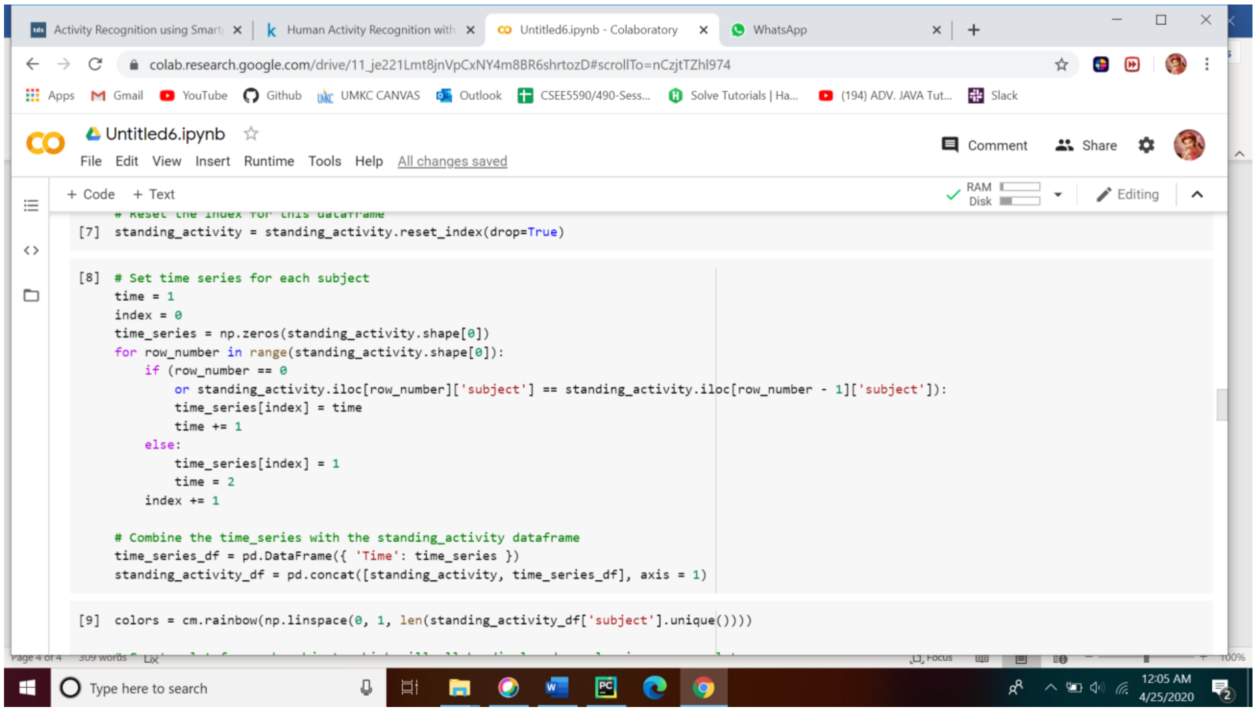
First we began by selecting all rows from the dataset that have the ‘Activity’ label as ‘STANDING’ and store it in standing_activity.

Code:

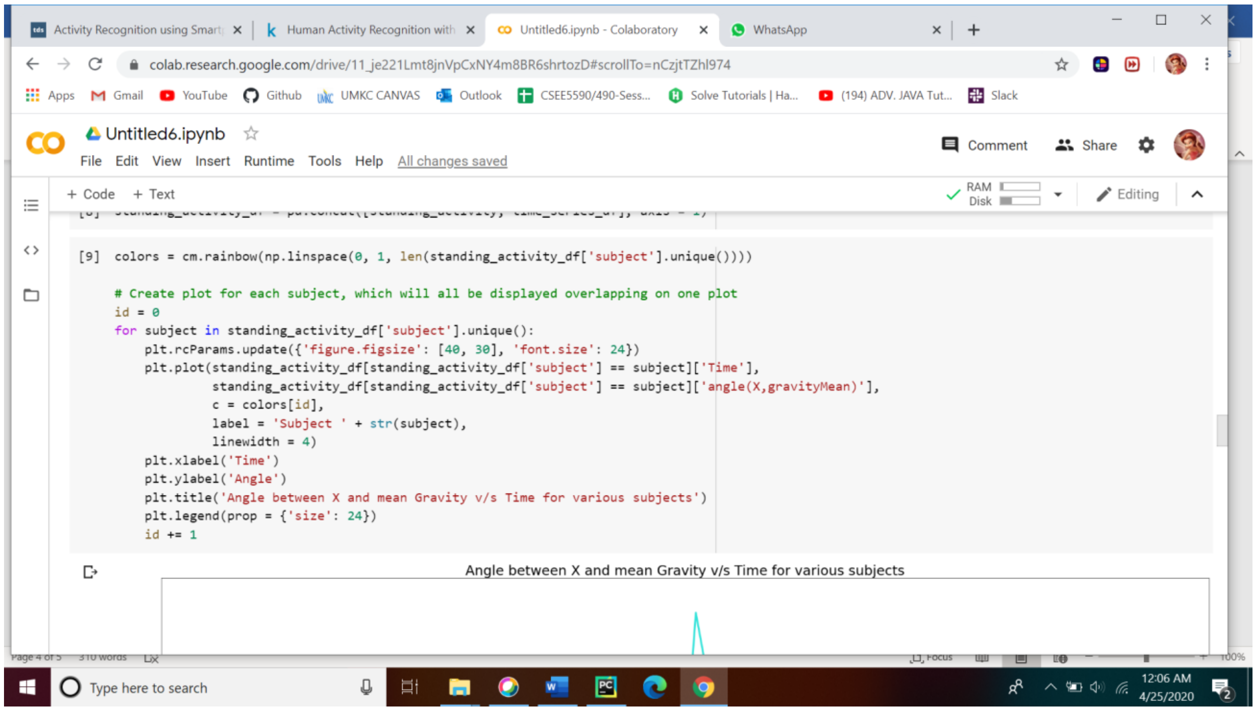
```
[7]  standing_activity = training_data[training_data['Activity'] == 'STANDING']
     # Reset the index for this dataframe
     standing_activity = standing_activity.reset_index(drop=True)
```

The data we collected is in continuous time series for each individual and was recorded at the same rate.

Code:

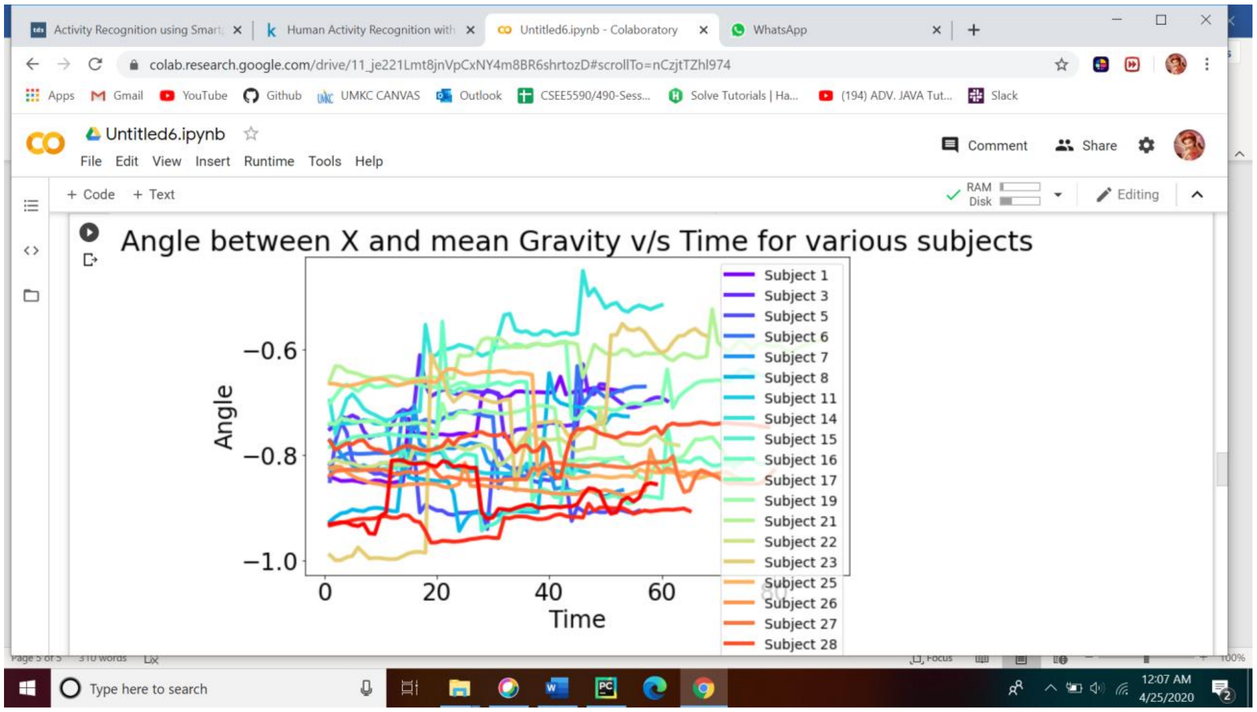


```
[7]  standing_activity = standing_activity.reset_index(drop=True)

[8]  # Set time series for each subject
     time = 1
     index = 0
     time_series = np.zeros(standing_activity.shape[0])
     for row_number in range(standing_activity.shape[0]):
         if (row_number == 0
             or standing_activity.iloc[row_number]['subject'] == standing_activity.iloc[row_number - 1]['subject']):
             time_series[index] = time
             time += 1
         else:
             time_series[index] = 1
             time = 2
         index += 1

     # Combine the time_series with the standing_activity dataframe
     time_series_df = pd.DataFrame({ 'Time': time_series })
     standing_activity_df = pd.concat([standing_activity, time_series_df], axis = 1)

[9]  colors = cm.rainbow(np.linspace(0, 1, len(standing_activity_df['subject'].unique())))
```



```
[9] colors = cm.rainbow(np.linspace(0, 1, len(standing_activity_df['subject'].unique())))

    # Create plot for each subject, which will all be displayed overlapping on one plot
    id = 0
    for subject in standing_activity_df['subject'].unique():
        plt.rcParams.update({'figure.figsize': [40, 30], 'font.size': 24})
        plt.plot(standing_activity_df[standing_activity_df['subject'] == subject]['Time'],
                 standing_activity_df[standing_activity_df['subject'] == subject]['angle(X,gravityMean)'],
                 c = colors[id],
                 label = 'Subject ' + str(subject),
                 linewidth = 4)
        plt.xlabel('Time')
        plt.ylabel('Angle')
        plt.title('Angle between X and mean Gravity v/s Time for various subjects')
        plt.legend(prop = {'size': 24})
        id += 1
```

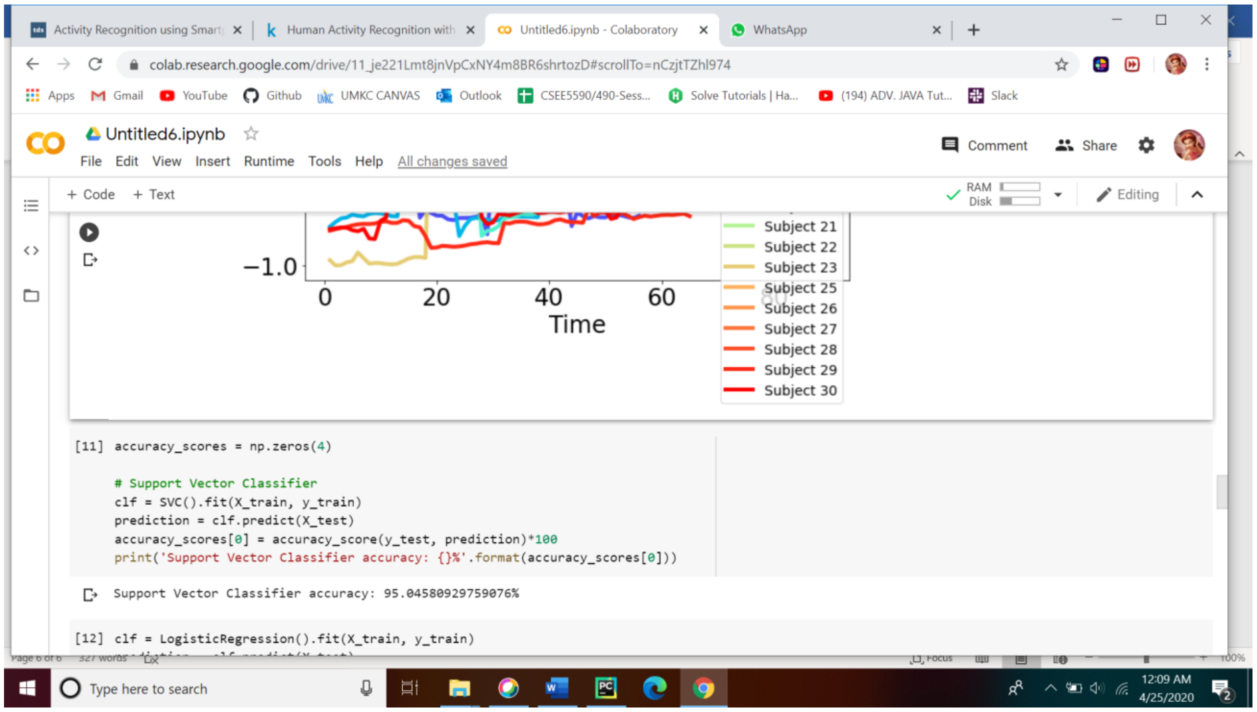
Output:



Classifying Activities:

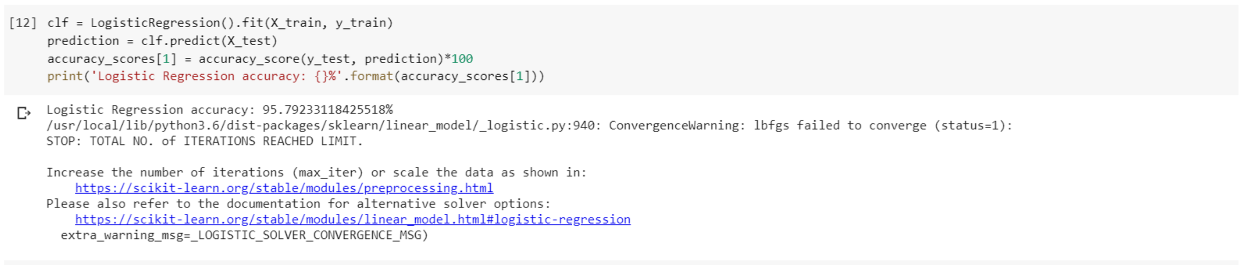
Now ,we can apply out machine learning algorithms and make classifications.

1.Support Vector Classifier



```
[11] accuracy_scores = np.zeros(4)

     # Support Vector Classifier
     clf = SVC().fit(X_train, y_train)
     prediction = clf.predict(X_test)
     accuracy_scores[0] = accuracy_score(y_test, prediction)*100
     print('Support Vector Classifier accuracy: {}%'.format(accuracy_scores[0]))

     Support Vector Classifier accuracy: 95.04580929759076%

[12] clf = LogisticRegression().fit(X_train, y_train)
```

2.Logistic Regression



```
[12] clf = LogisticRegression().fit(X_train, y_train)
     prediction = clf.predict(X_test)
     accuracy_scores[1] = accuracy_score(y_test, prediction)*100
     print('Logistic Regression accuracy: {}%'.format(accuracy_scores[1]))

     Logistic Regression accuracy: 95.79233118425518%
     /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
     STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

     Increase the number of iterations (max_iter) or scale the data as shown in:
         https://scikit-learn.org/stable/modules/preprocessing.html
     Please also refer to the documentation for alternative solver options:
         https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
       extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

And still we should implement Machine learning algorithms for K- nearest neighbor and Random Forest Classifier.

Challenges:

- At first we thought it’s much easier to find datasets online through Google Dataset Search, but there are too datasets many available . To pick the right data set is critical for this project to be successful.
- We must ensure that the dataset does not have any null values. Skimming the null values is important part of it. We use shape method to find the size of the dataset and used isnull().values.any() to check if any cell is empty. It results in the output as (rows, columns).
- One of the key things when working with data is to ensure that all classes are of approximately equal size. This is essential so that the machine learning algorithm is not biased towards any one class.

References:

• http://theprofessionalspoint.blogspot.com

•https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

• https://upcommons.upc.edu/handle/2117/20897

Work distribution:

Basically, to develop the code all there of us have worked together. Its always important to look from a different point of view when we got stuck at one point. So, we used to work on Google Collab for that as Its easy to hand over the work from one to another.

• Biweekly we used to connect over the Zoom to discuss the updates and challenges of the project.

But below are the strong points that we have focused as individual on top of overall project.

- Analysis of Data and Design– Aditya Velugula
- Application of Algorithms – Sivaram Pabolu
- Visualization and Results – Chakradhar Chinnam