2021

**HUMAN RECOGNITION WITH SMART PHONE DATASET**

Table of Contents

[HUMAN RECOGNITION WITH SMART PHONE DATASET 3](#_Toc90981333)

[Import Libraries 3](#_Toc90981334)

[Load the Dataset 5](#_Toc90981335)

[2. Data Visualization and Exploration 6](#_Toc90981336)

[1. Print 2 rows for sanity check to identify all feature present in the dataset and if the target matches with them 6](#_Toc90981337)

[2. Print the description and basic statistical details 7](#_Toc90981338)

[3. Print each class label count (Activity) and create a Pie Chart for each class 9](#_Toc90981339)

[4. Plot Activities by Subject/ Participant and Provide Appropiate Comments on Visualized Data 11](#_Toc90981340)

[Try exploring the Data and see what insights can be drawn from the datset 15](#_Toc90981341)

[COUNT PLOTS 15](#_Toc90981342)

[BAR PLOTS 17](#_Toc90981343)

[HISTOGRAM 18](#_Toc90981344)

[Data Preprocessing and Cleaning 19](#_Toc90981345)

[2. Use Min Max Normalization for Feature Transformation 22](#_Toc90981346)

[Python Code 22](#_Toc90981347)

[3. Do coorelational analysis on the dataset. Provide Visualization for the Same 23](#_Toc90981348)

[Data Preparation 25](#_Toc90981349)

[Do the final feature selection and extract them into column X and Class Label into collumn Y 25](#_Toc90981350)

[2. Split the Datasets into Training and Test Sets 26](#_Toc90981351)

[CASE 1 26](#_Toc90981352)

[CASE 2 26](#_Toc90981353)

[PART B MODEL BUILDING 26](#_Toc90981354)

# 

# HUMAN ACTIVITY RECOGNITION WITH SMART PHONES DATASET

**In this human activity recognition with smart phone dataset, we have 10299 rows and 563, Activity is our target variable while all the other are our independent variables**

# Import Libraries

First, we will import all the required libraries:

import pandas as pd

Pandas library is used for the data manipulation and analysis

import numpy as np

Numpy allows us to create an multidimensional array, it is useful for performing different mathematical operations in python

import seaborn as sns

Seaborn is a library in Python predominantly used for making statistical graphics.

import matplotlib.pyplot as plt

For data vaisulization we use matplotlib library

import missingno as msno

Missingno is a Python library that provides the ability to understand the distribution of missing values through informative visualizations

from sklearn import preprocessing

We do pre-processing to apply transformations to our data before feeding it to the algorithm. Here we are applying MinMax Scaler

from sklearn.model\_selection import train\_test\_split

Now I will do train and test split, to split my dataset into train and test set

from sklearn.linear\_model import LogisticRegression

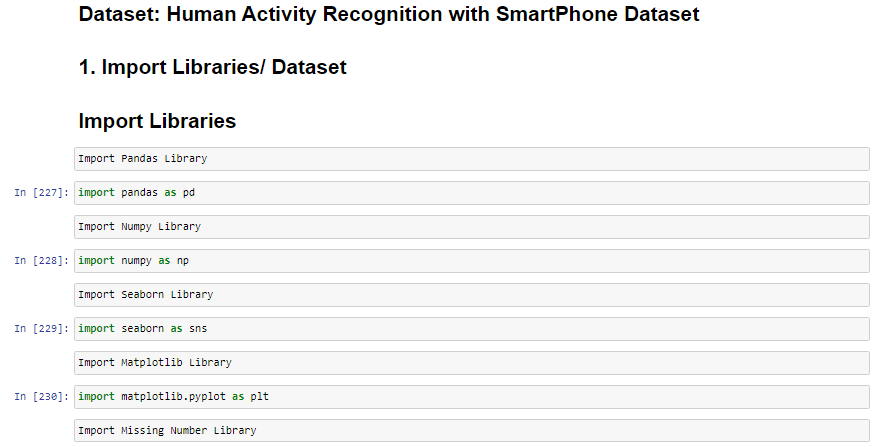
As we are implementing LogisticRegression, so I will be implementing LogisticRegression here.

from sklearn import metrics

To calculate accuracy, precision score, f1\_score and recall score

from sklearn.metrics import classification\_report

It will print the classification report, which contains details about accuracy, precision, f1\_score and recall score



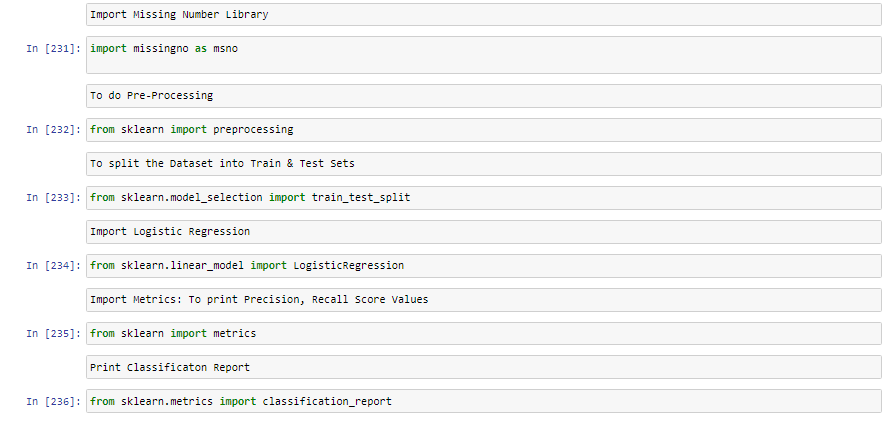


Figure Import Libraries Screen Shot of JupyterNotebook

# Load the Dataset

As my dataset is in the form of .csv file so I will load it using the following command

**pd.read\_csv(“”)**

**In below code I am checking the first few rows using the .head() and .tail() to check the last few rows**

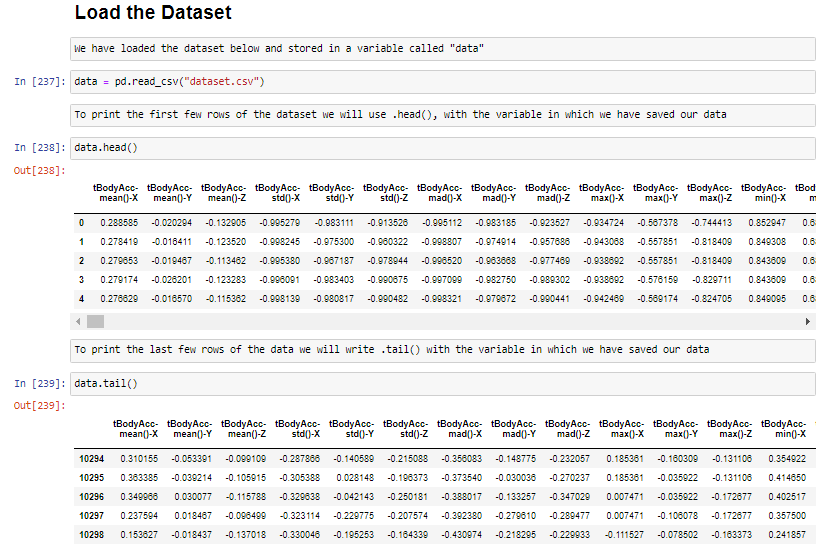
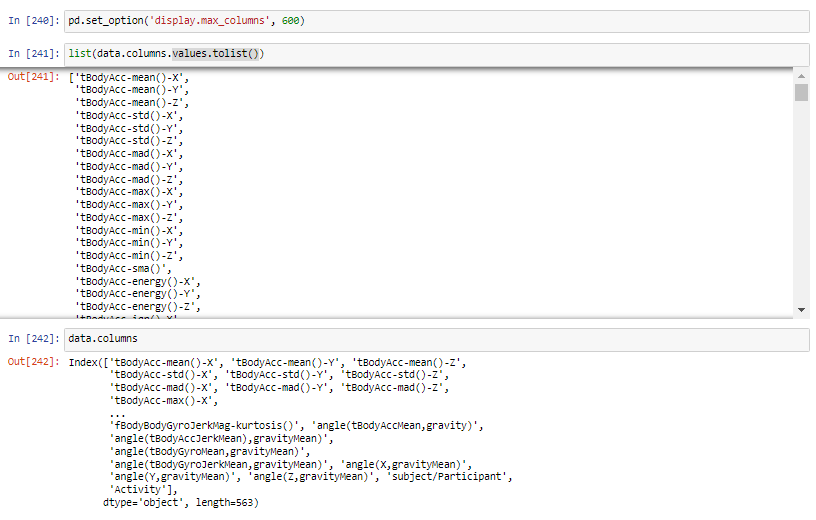


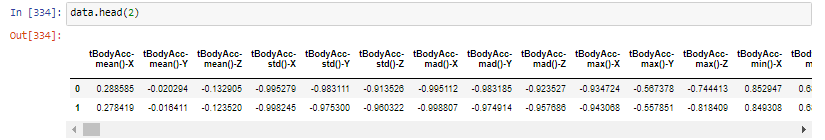
Figure Load the Dataset JupyterNotebook Image attached

# 2. Data Visualization and Exploration



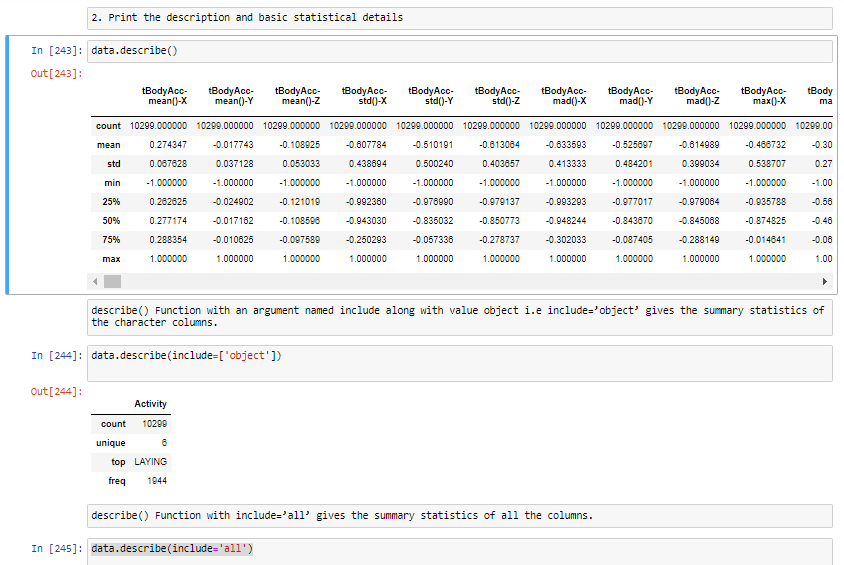
## Print 2 rows for sanity check to identify all feature present in the dataset and if the target matches with them

Using .head(2) we can print the first two rows for sanity check and it can be seen all features are present in the dataset and the target values matches as well.



# Print the description and basic statistical details

By writing .describe() with the variable name in which data is stored we can print the description



**I have performed basic statistical tests, using seaborn pairplot**

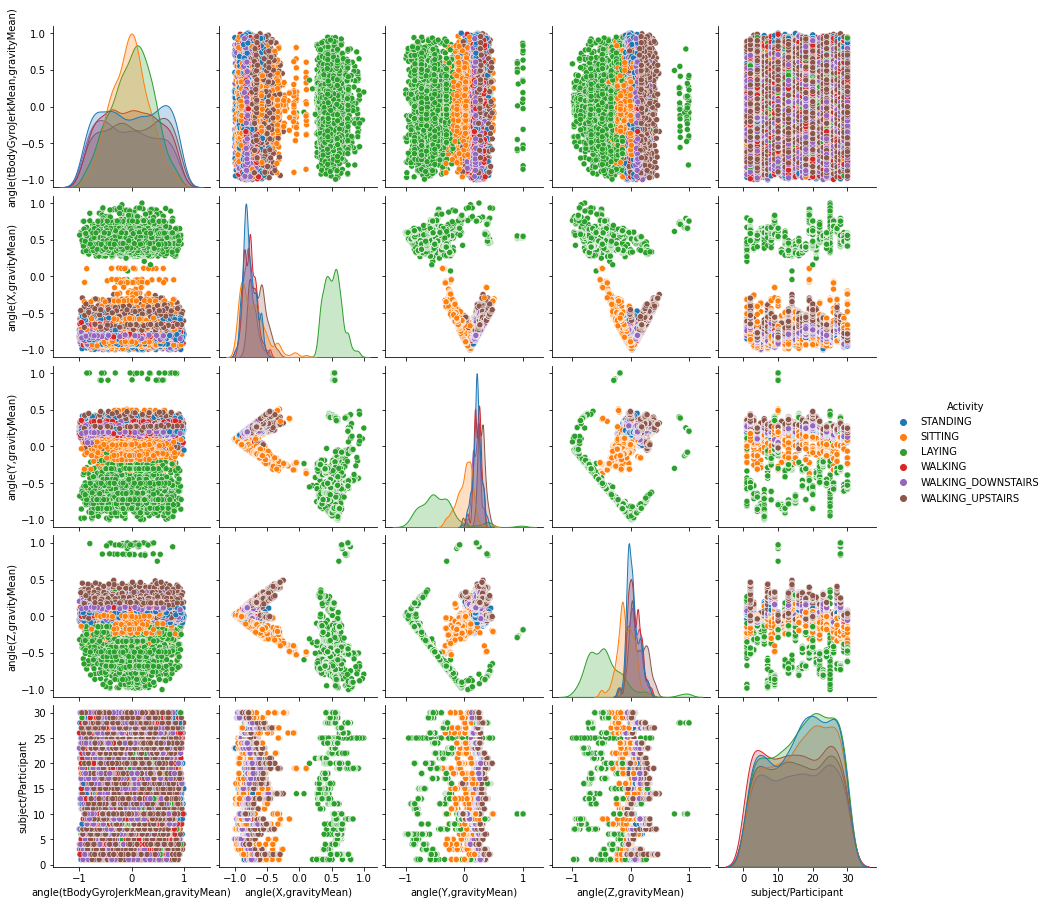
**Following is the python code used for this purpose**

plt.figure()

sns.pairplot(data, vars=['angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',

'angle(Y,gravityMean)', 'angle(Z,gravityMean)','subject/Participant'], hue = 'Activity')

plt.show()



## Print each class label count (Activity) and create a Pie Chart for each class

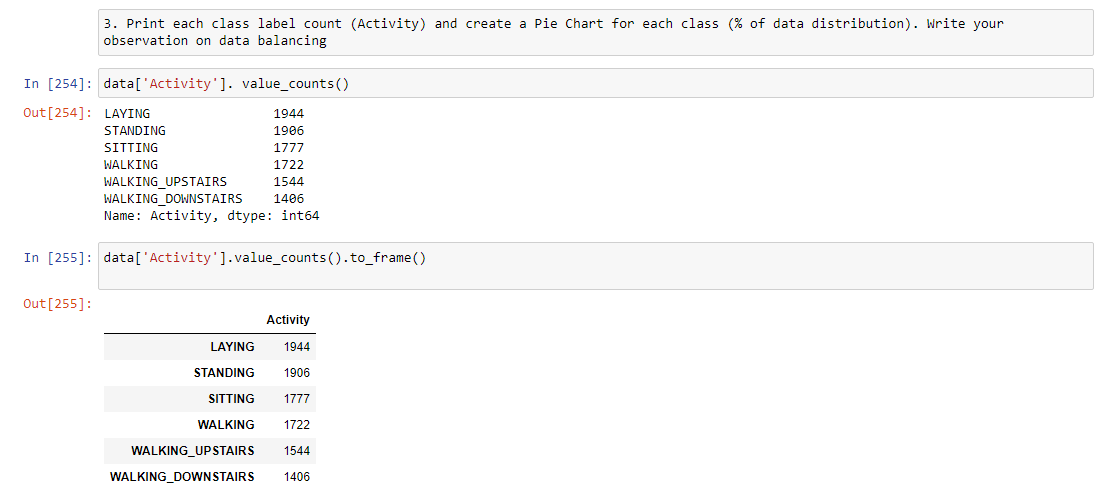
**Using**

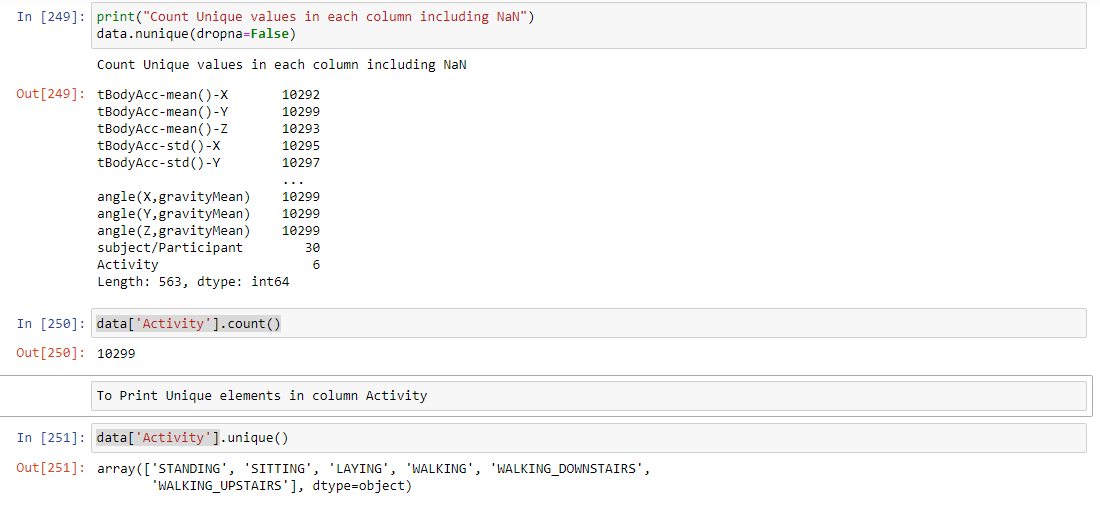
**.value\_counts()**

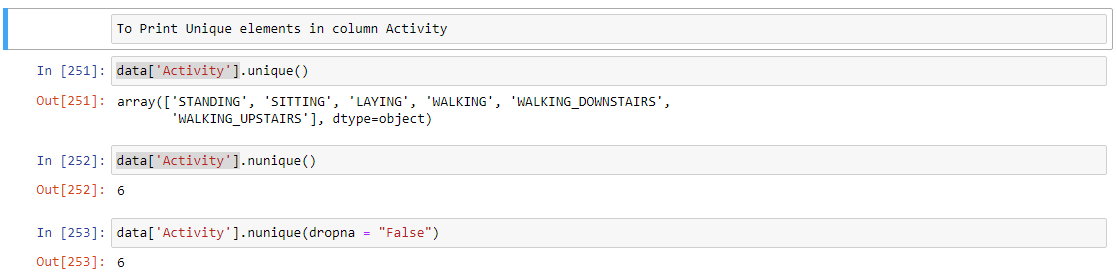
**.value\_counts().to\_frame()**

**.count()**

**We can print each class label count**

****

****

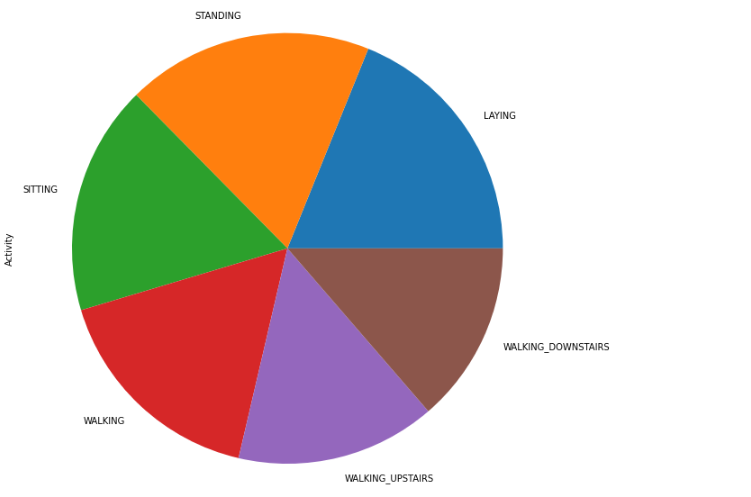
****

**CREATE A PIE CHART**

**We will use the following python code to create an PIE chart**

plt.figure(figsize=(11, 11))

data['Activity'].value\_counts().plot(kind='pie')



A balanced dataset is basically an dataset in which each of the target class isbeing represented by the same number of input samples. Balancing can be performed by using one of the following techniques:

oversampling.

undersampling.

class weight.

While, Imbalanced data typically refers to a classification problem where the number of observations per class is not equally distributed

# Plot Activities by Subject/ Participant and Provide Appropiate Comments on Visualized Data

In the image shown below we have created an bar plot in which we have ploted activities by participant we have set the fig size 15 by 15 so all the x labels and y labels can be incorporated in it and plus we have tried one new thing in this graph we have changed the font of x-label and y-label as well

fig\_dims = (15, 15)

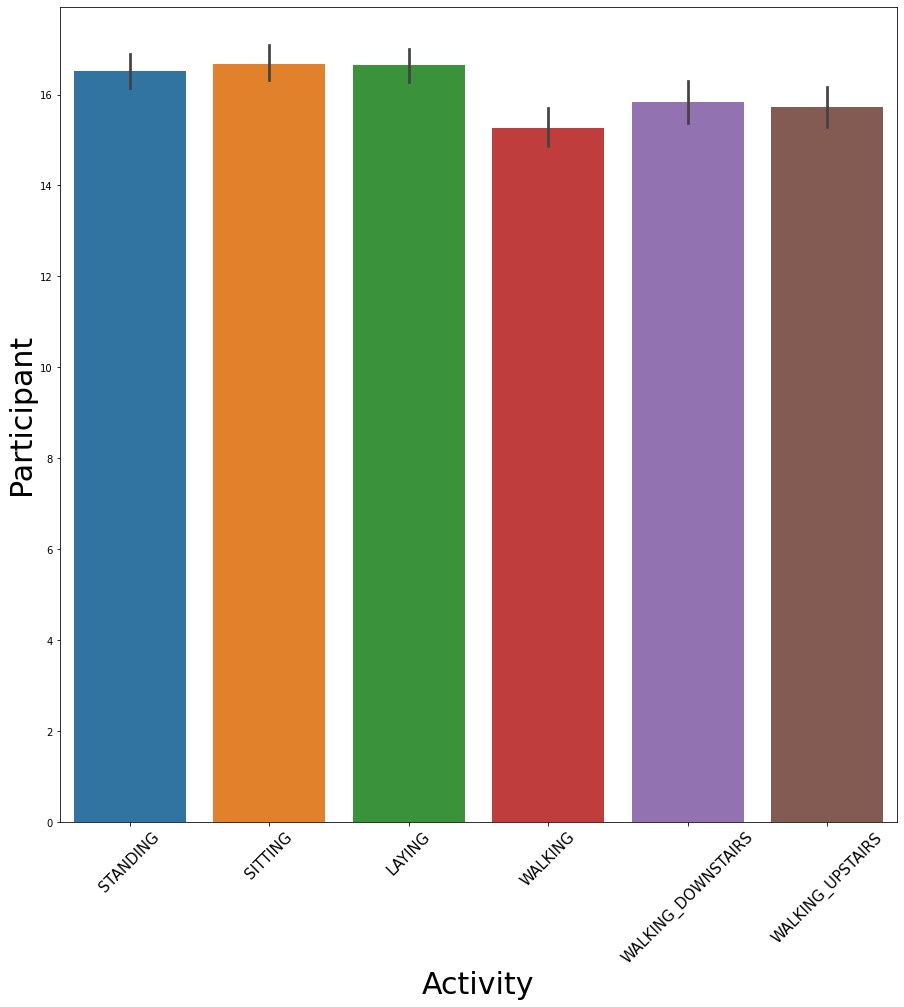
fig, ax = plt.subplots(figsize=fig\_dims)

sns.barplot(x = "Activity", y = "subject/Participant", data = data, ax = ax)

plt.ylabel("Participant", fontsize = 30)

plt.xlabel("Activity", fontsize = 30)

plt.xticks(rotation = 45, fontsize = 15)



fig\_dims = (15, 15)

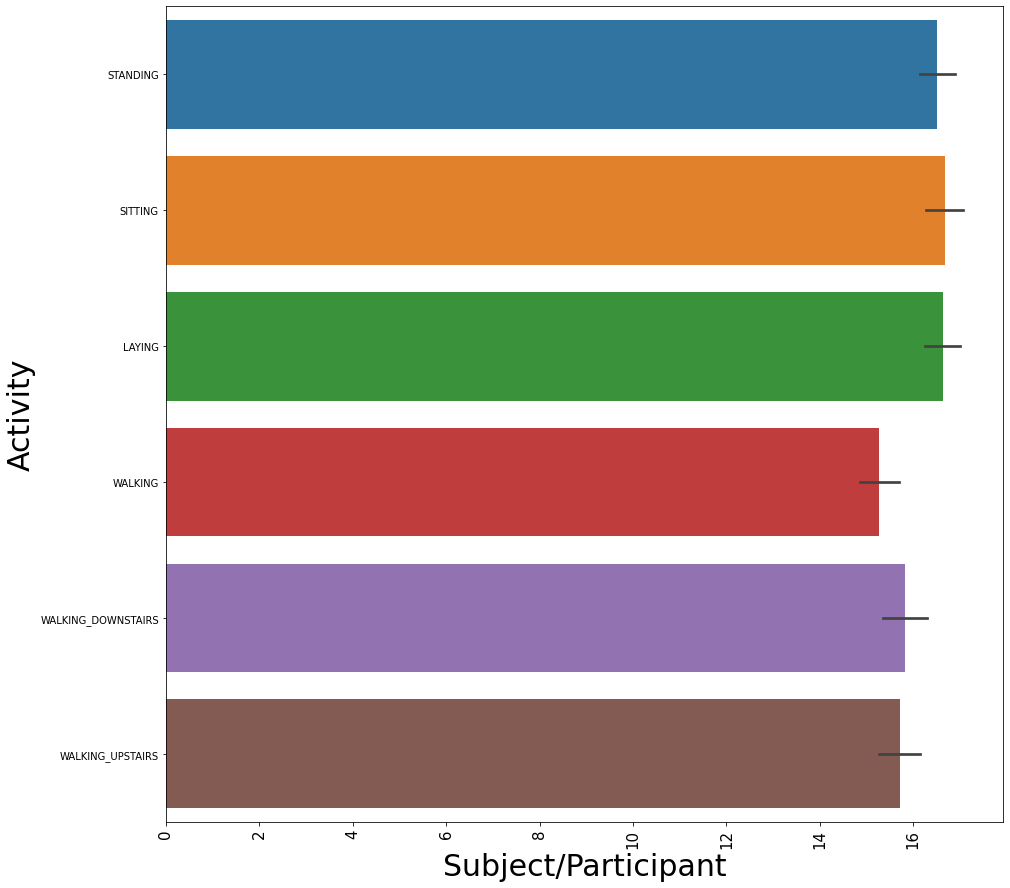
fig, ax = plt.subplots(figsize=fig\_dims)

sns.barplot(x = "subject/Participant", y = "Activity", data = data, ax = ax)

plt.ylabel("Activity", fontsize = 30)

plt.xlabel("Subject/Participant", fontsize = 30)

plt.xticks(rotation = 90, fontsize = 15)

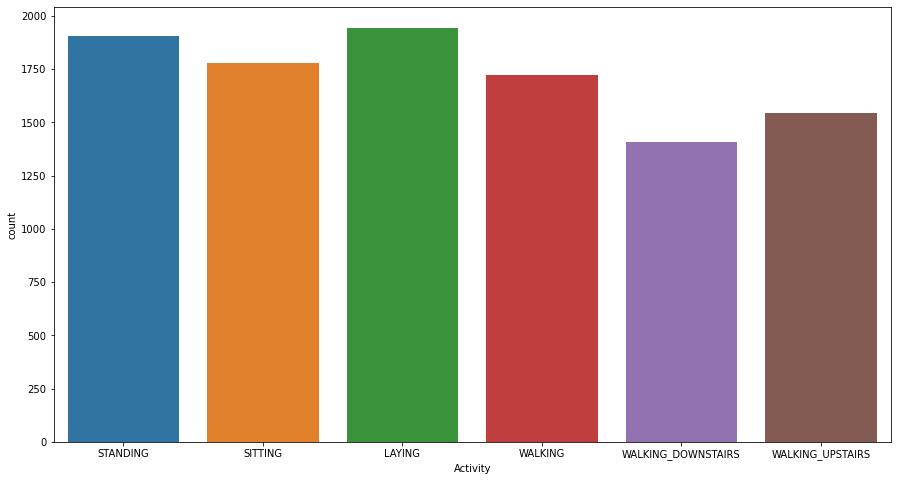


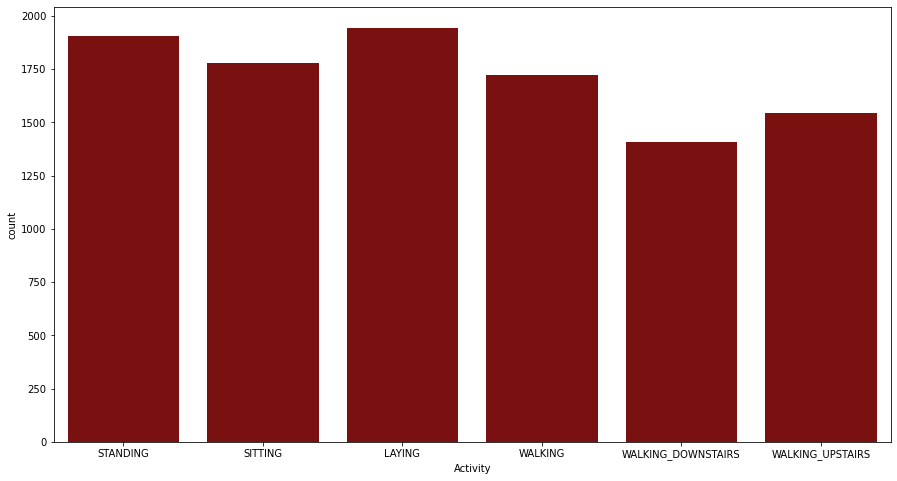
# Try exploring the Data and see what insights can be drawn from the datset

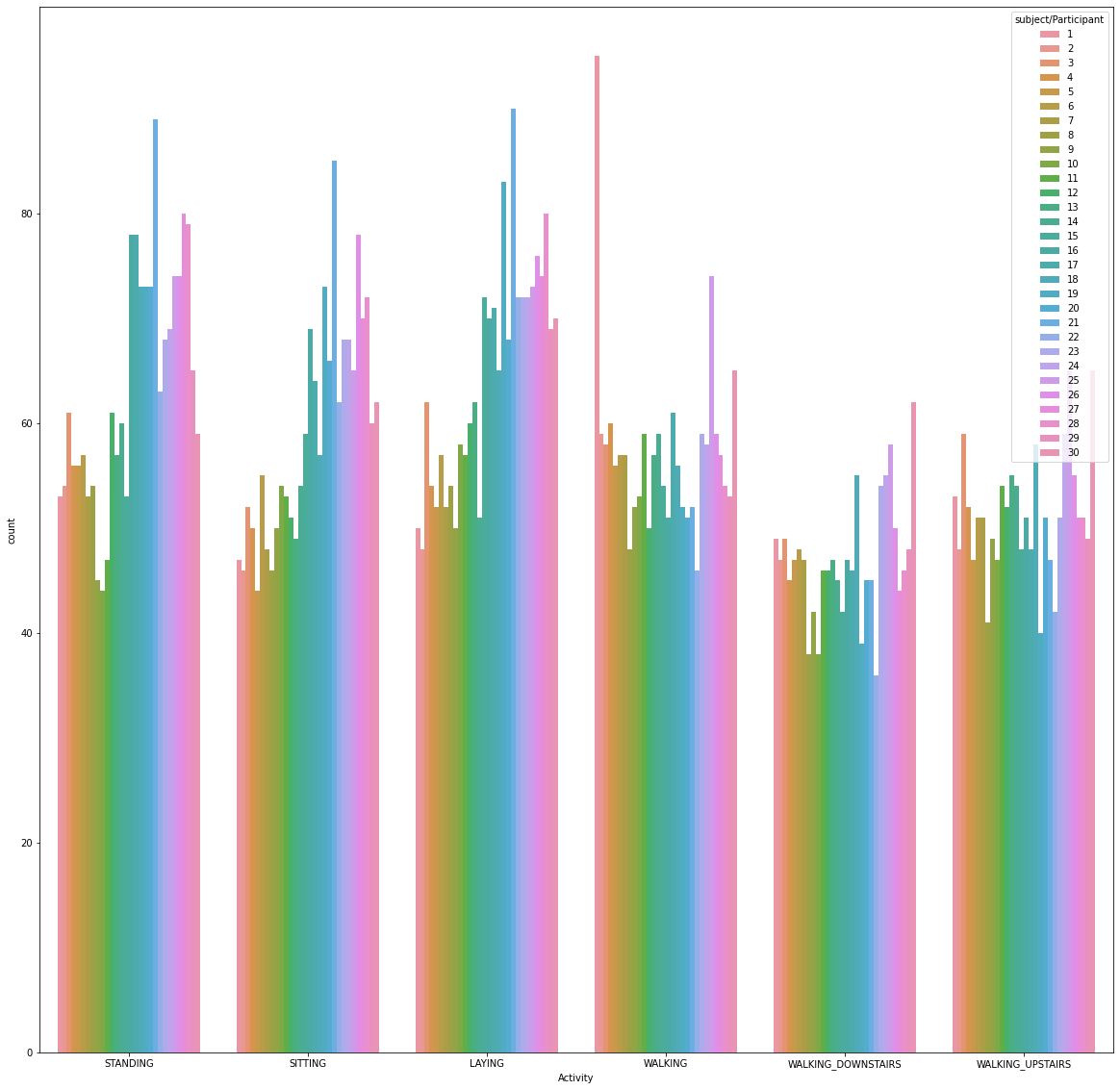
We have performed exploratory data analysis using

* histogram
* count plots
* kdeplot
* distplot
* box plots
* violin plots

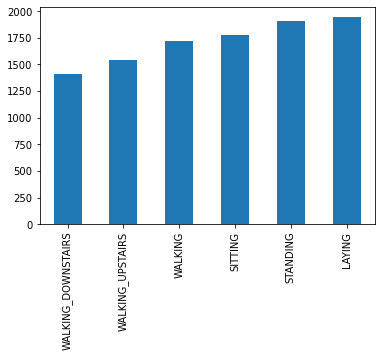
# COUNT PLOTS



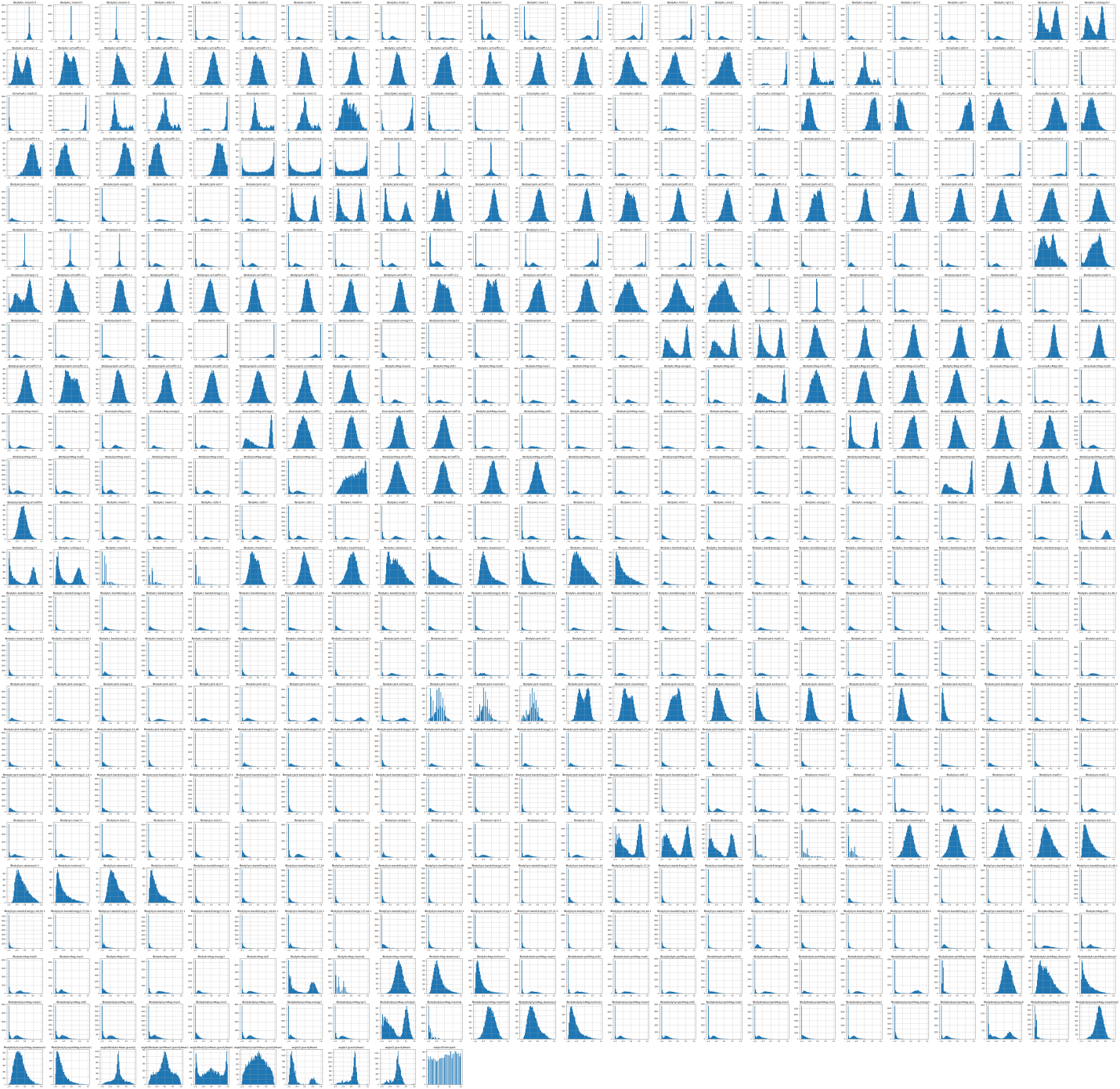




# BAR PLOTS



# HISTOGRAM



# Data Preprocessing and Cleaning

1. Do the appropiate pre-processing steps,

1. Identify NULL or Missing Values based on column. Apply appropiate feature Engineering Techniques for them

Using

.info()

.shape

.dtypes

.columns

.columns.values

list(set(data.dtypes.tolist()))

for column in data:

print(data[column].unique())

for column in data:

print(f'{column}:{data[column].unique()}')

df\_num = data.select\_dtypes(include = ['float64', 'int64'])

df\_num.head()

.isnull()

.isnull().sum()

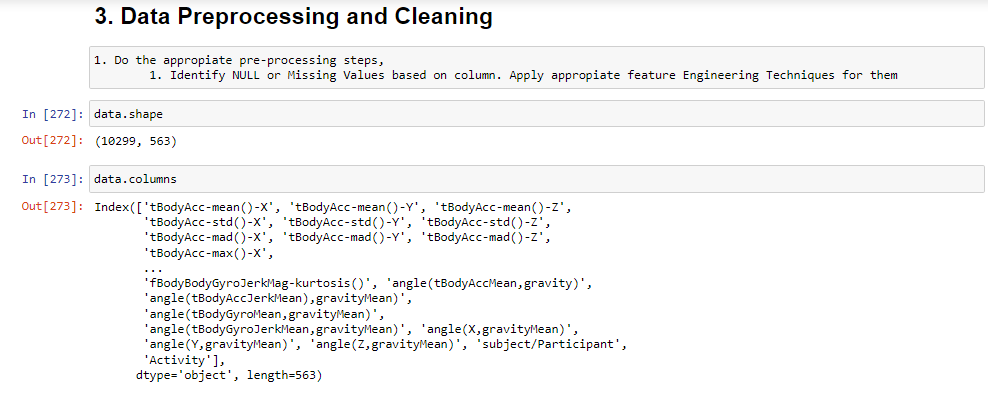
.info()

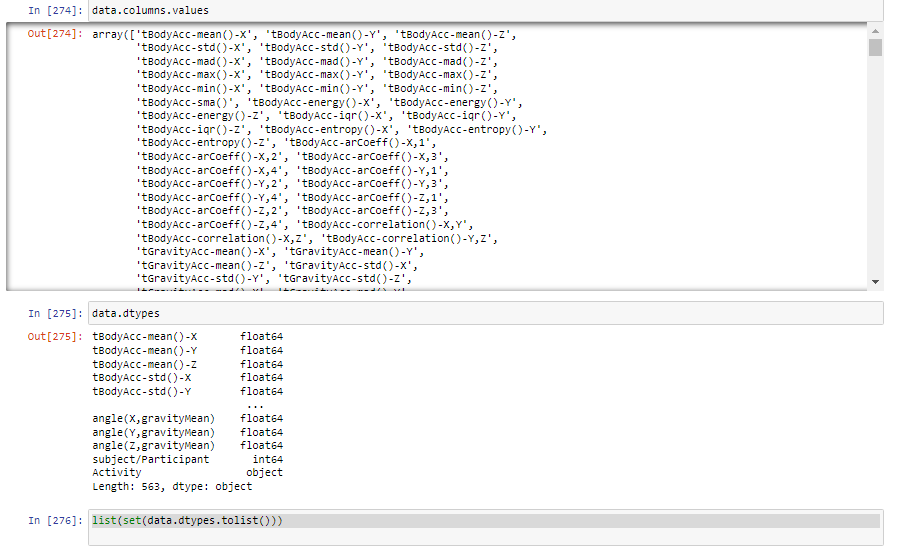
sns.heatmap(data.isnull(), yticklabels = False, cmap = "viridis")

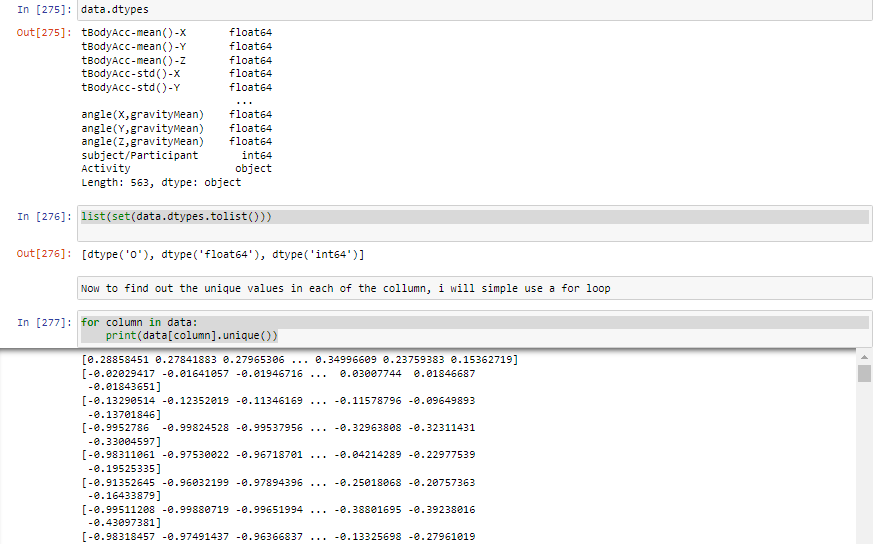
msno.heatmap(data)

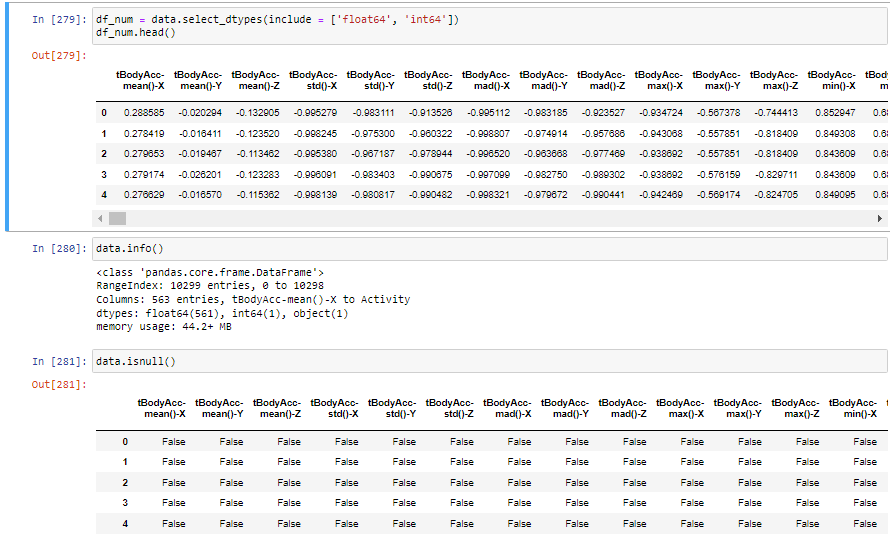
msno.dendrogram(data)

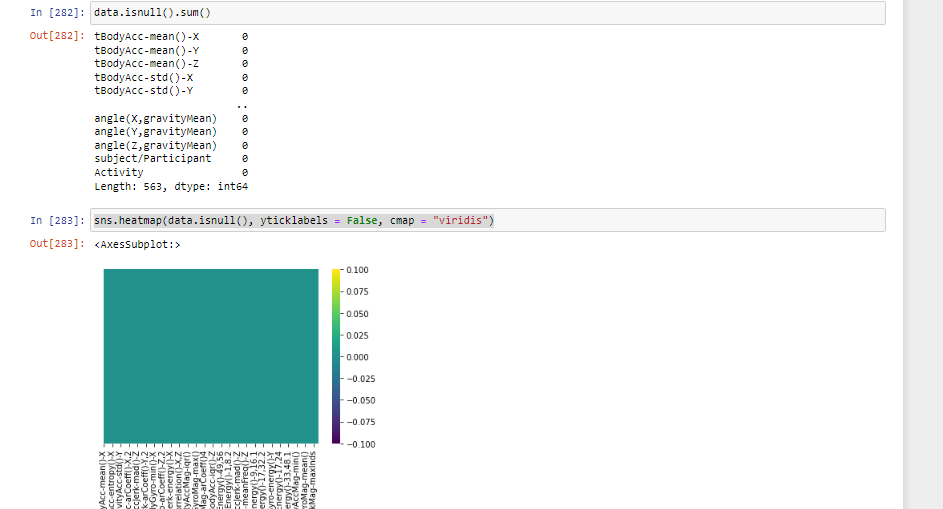
We have performed data preprocessing and cleaning











## 2. Use Min Max Normalization for Feature Transformation

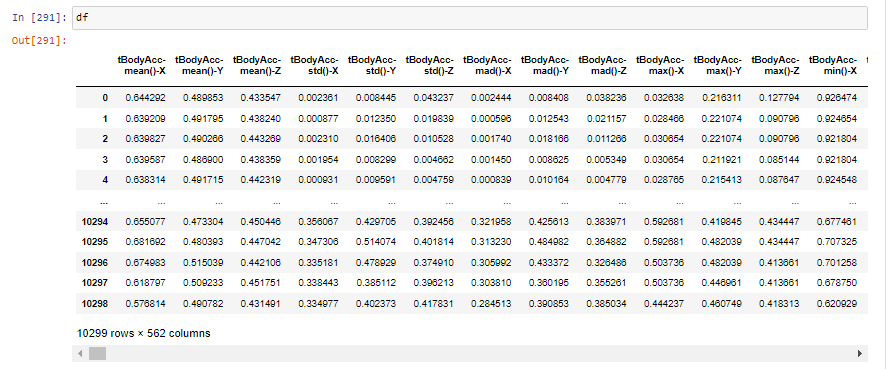
## Python Code

MinMax Scaler transforms are the values between 0 and 1

min\_max\_scaler = preprocessing.MinMaxScaler()

x\_scaled = min\_max\_scaler.fit\_transform(x)

df = pd.DataFrame(x\_scaled)



# 3. Do coorelational analysis on the dataset. Provide Visualization for the Same

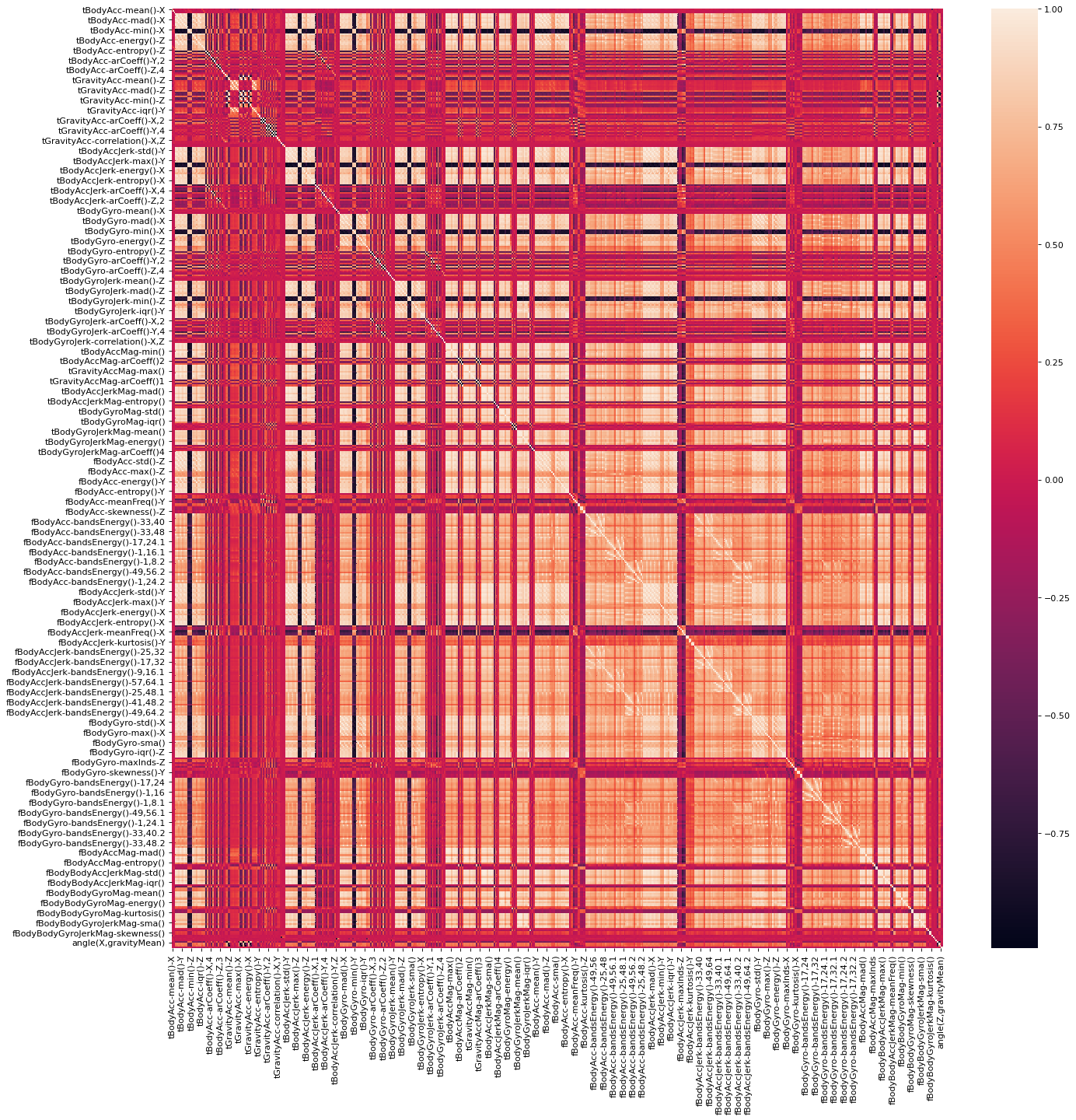
To do correlational analysis I will write

data.corr()

plt.figure(figsize=(20,20), dpi= 80)

sns.heatmap(data.corr())





# Data Preparation

# Do the final feature selection and extract them into column X and Class Label into collumn Y

For feature selection I will use Extra Tree Classifier

from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()

x = df

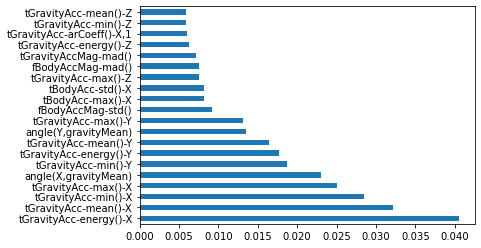
y = data['Activity']

model.fit(x, y)

print(model.feature\_importances\_)

features\_importances = pd.Series(model.feature\_importances\_, index = x.columns)

features\_importances.nlargest(20).plot(kind = 'barh')



# 2. Split the Datasets into Training and Test Sets

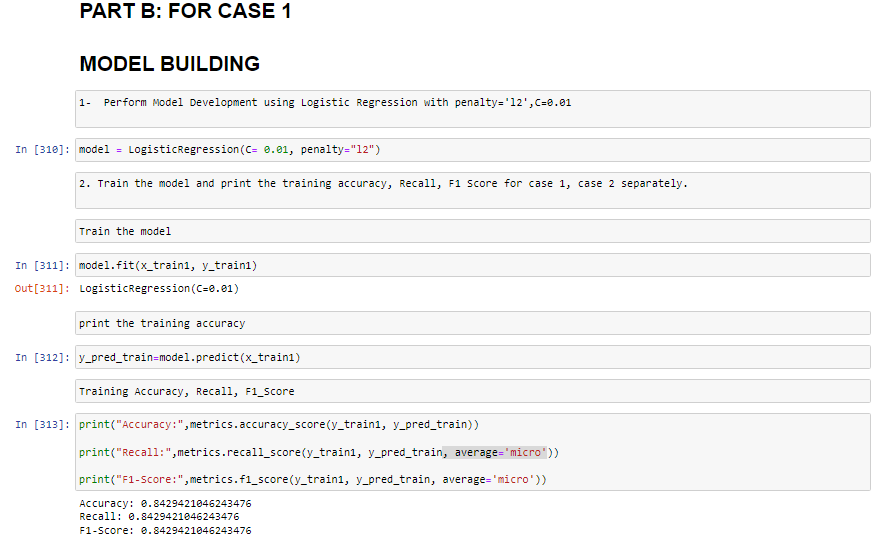
# CASE 1

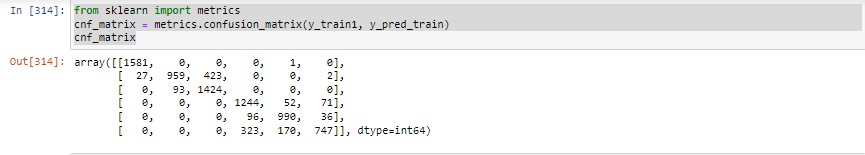
x\_train1, x\_test1, y\_train1, y\_test1 = train\_test\_split(x, y, test\_size = 0.20, random\_state = 1)

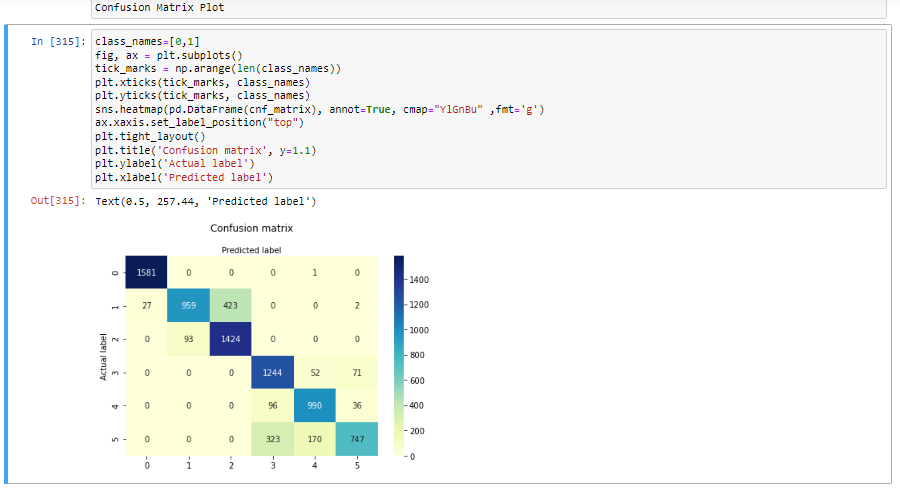
# CASE 2

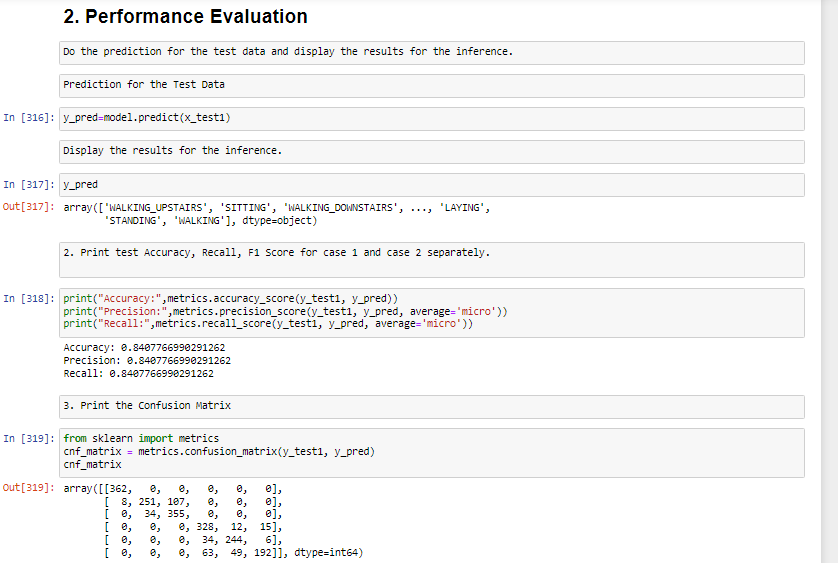
x\_train2, x\_test2, y\_train2, y\_test2 = train\_test\_split(x, y, test\_size = 0.90, random\_state = 1)

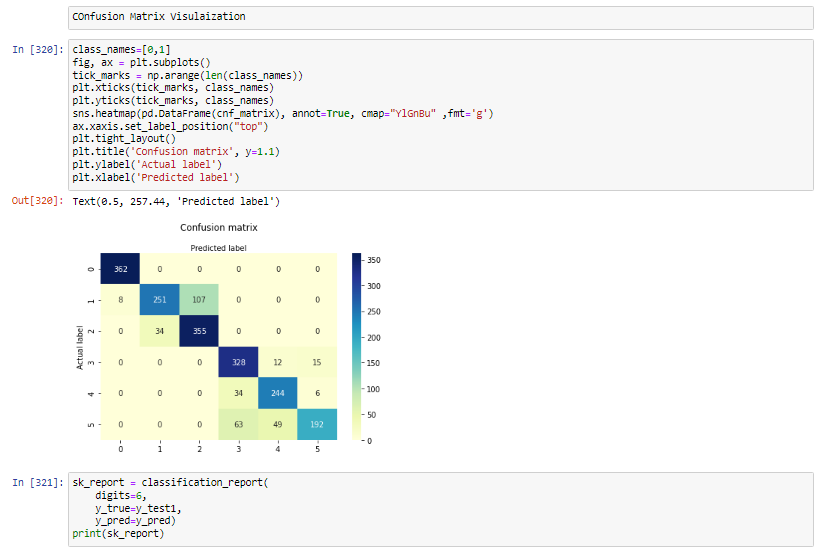
# PART B MODEL BUILDING

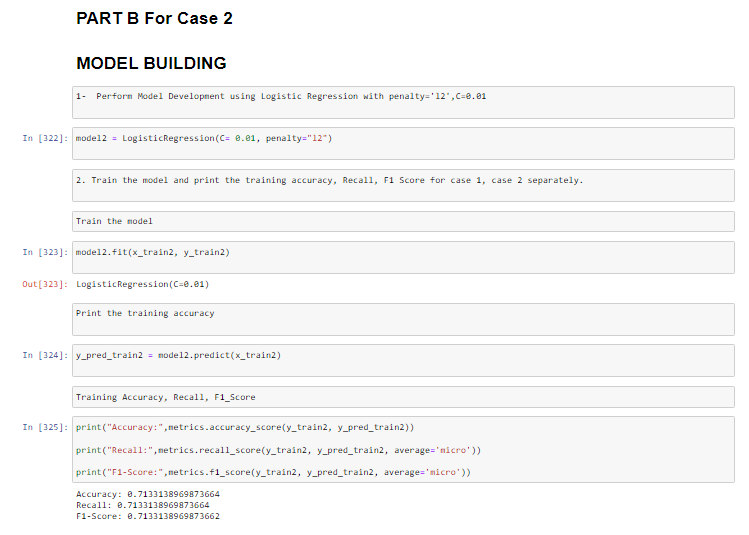


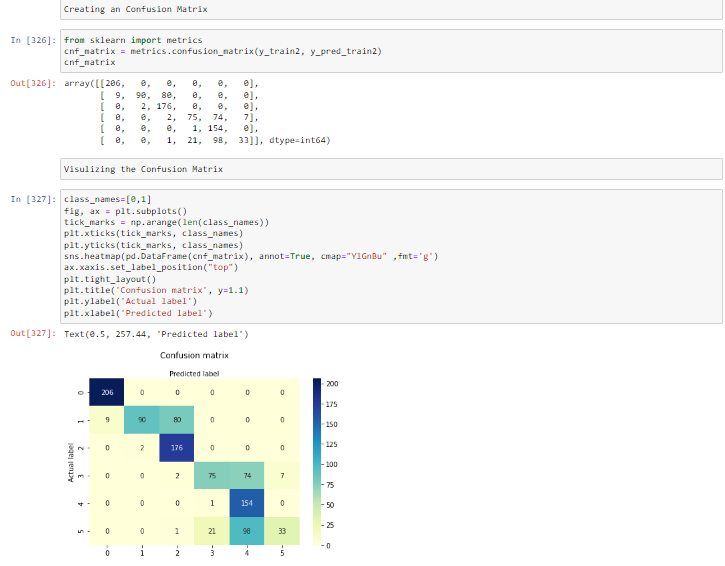


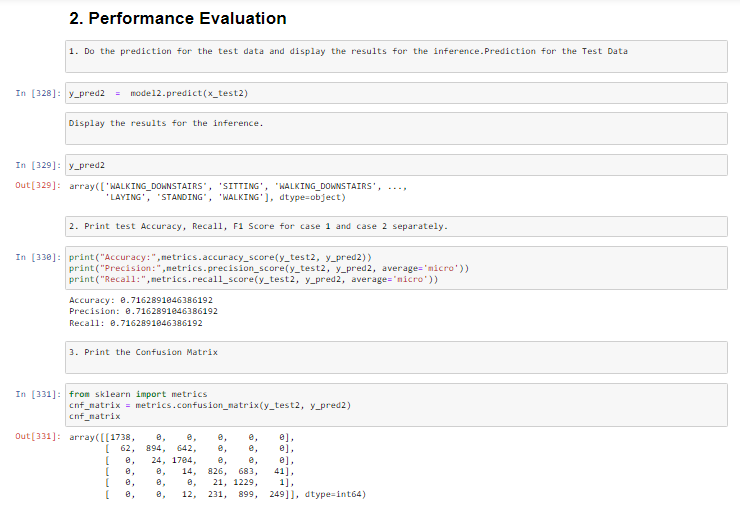


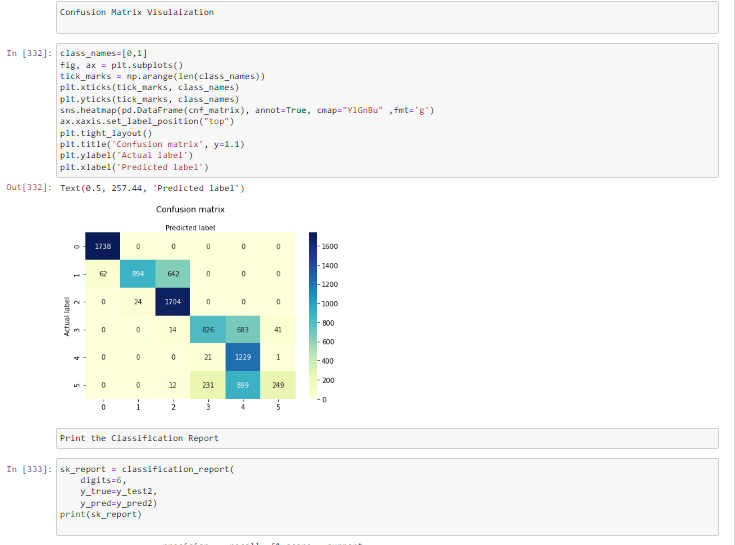












In the Case 1:

* 80% of the data was considered as the training data
* 20% of the data was consdiered as the testing data

So, as we have more data to train our model so, our model will perform, because there is a large dataset on which model trains and it learns

The Trainig Accuracy: 84.2942% while the testing acuracy obtained was Accuracy: 84.077%. So, it can be seen that the training accuracy is a bit more than the testing accuracy. Why the trainig accuracy is a bit more than the testing accuracy because we are testing our model on the same data on which we have trained it, so the training accuracy was a bit more testing accuracy, While the testing data is the hidden data our model isnt trained on this data

In the Case 2:

* 10% of the data was considered as the training data
* 90% of the data was consdiered as the testing data

So, only 10% of the data is being used to train the model in case 2 while 90% of the data is being used to test the model. So in the case 2 we have very less data to train our model, while there is a large amount of data on which we can test our model. The training accuracy in this case is Accuracy: 71.331% while the testing accuracy is 71. 628%. We can see that in case 2 our model training and testing accuracy is very less as compared to the case 1, the reason is that we have very less data to train our model.