[MAD][DATA1018][FINAL-PROJECT]

Physical Activity Monitoring Assistant

Ivan Cernicharo Ortiz
IronHack Data Analytics Bootcamp

1.Description and Objectives

1. Description and Objectives

Physical Activity Monitoring Dataset - UCI

A. Physical Activities (18 different types)
B. Sensors:
    a. Hand
    b. Chest
    c. Ankle
C. Data (54 Features):
    a. Time
    b. Activity ID
    c. Heart Rate
    d. Temperature
    e. Acceleration
    f. Magnetic Field
    g. Angular acceleration
    h. Gyroscopes

Main objective:

Being able to predict which activity is the user doing in each moment to help him on an exercise routine

2 . Data Acquisition

2. Data Acquisition

First of all is to download the dataset from the URL I have provided below and decompress the data.

What we find is that we have 2 folders of data (Protocol and Optional) and 4 documents describing the dataset, the activities and some factors to take in count.

Protocol and Optional files together make 1.7 GB of data. Since I do not have a workstation and I have limited RAM (8GB + 2GB of SWAP partition) I have to be careful when loading data. Avoid duplicities and just have the needed data in my work environment each moment.

My workflow open these files in a loop and concatenate them into a two final dataframes, final_data and final_data_optional

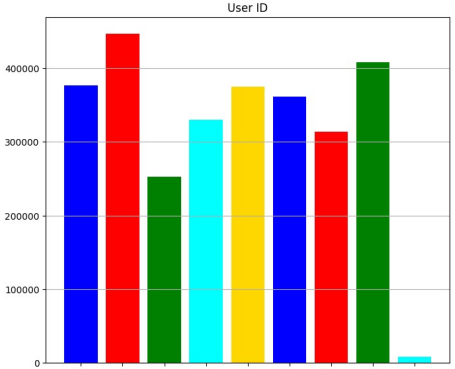
3. Data Exploration

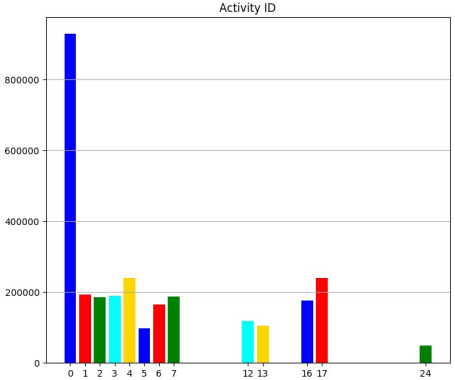
3. Data Exploration

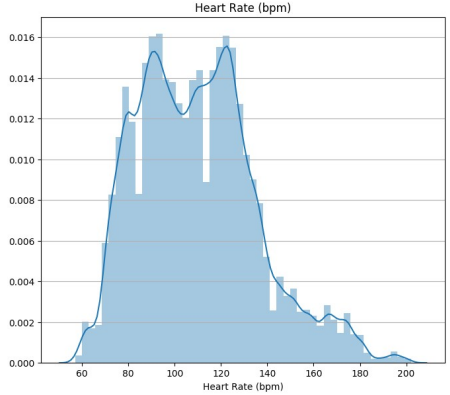
One of the best factors of this dataset is that all our Features are numerical, and only 2 of them are categorical. These makes reference to the **user ID** (This one has been defined by myself for analysis purposes) and to the **activity ID**.

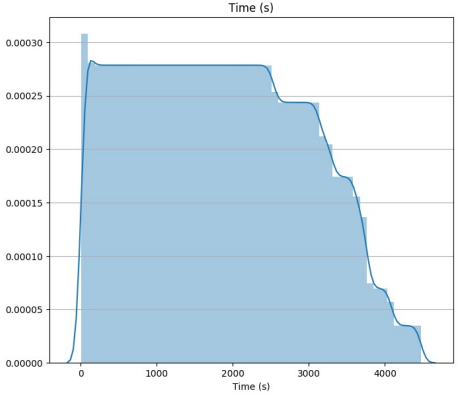
For the numerical Features, we have 12 of them that are invalid to this study, does not make any sense. These features are the one for the orientation and in my workflow are named as **Hand O1**, **Hand O2**, **Hand O3**, **Hand O4**, **Chest O1**, **Chest O2**, **Chest O3**, **Chest O4**, **Ankle O1**, **Ankle O2**, **Ankle O3**, and **Ankle O4**. This step reduces from 55 to 43 Features.

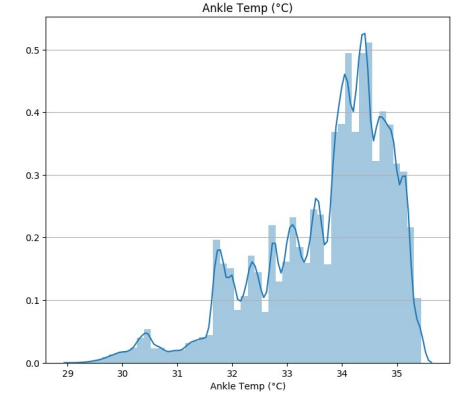
3. Data Exploration

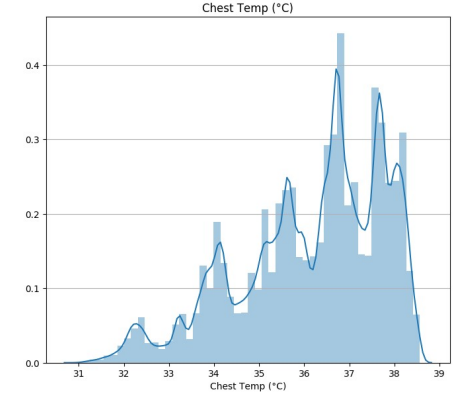


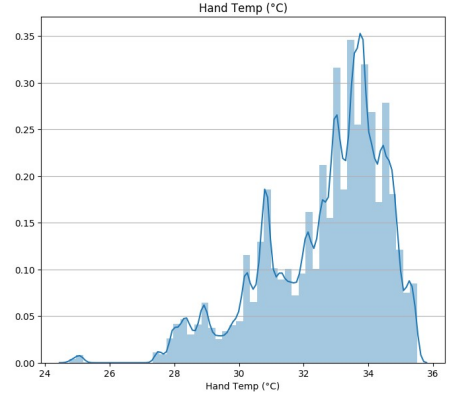






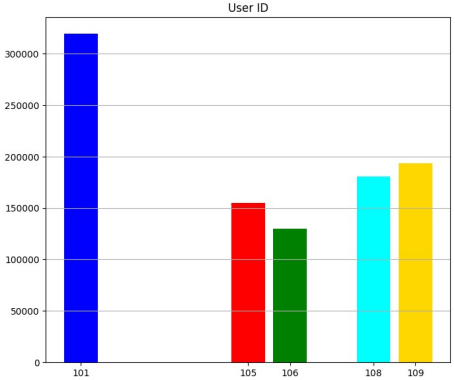


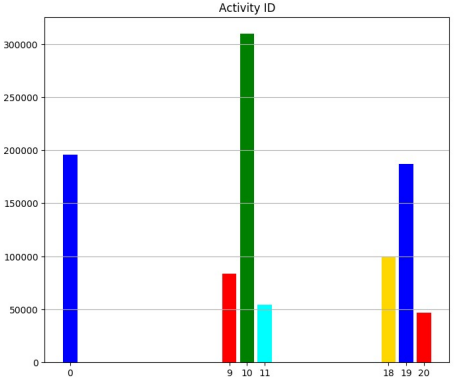


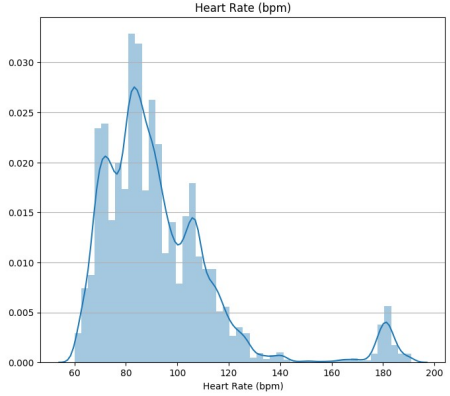


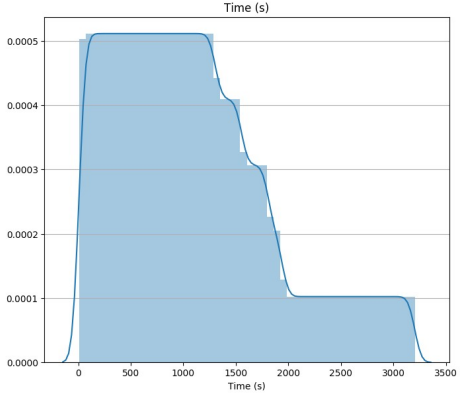
Protocol DataSet

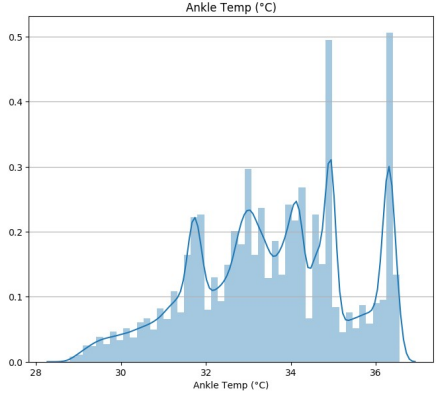
3. Data Exploration

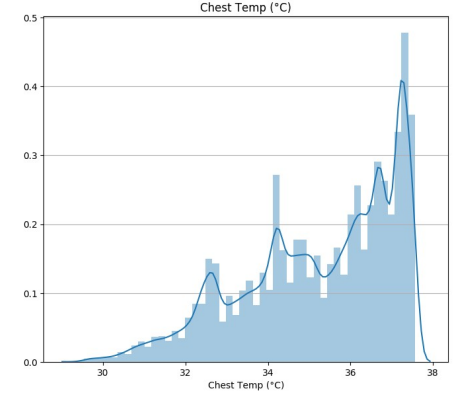


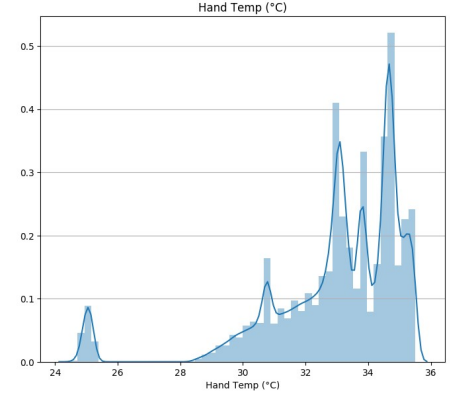












Optional DataSet

3. Data Exploration

As we can observe, **Activity ID** == 0 has the major number of registers. We can see the difference of activities in each dataframe, for Optional dataframe there are less activities and registers. Also, we can observe the Heart Ratio has very low number of registers compared to the rest of the Features.

4. Data Cleaning and Wrangling

4. Data Cleaning and Wrangling

First step is to interpolate the Heart Rate Feature to fill the 90% of missing data for this Feature (These are taken with a lower frequency so that is the why are less values)

Next step is to deal with missing values. We are going to check how many values are missing and how they affect to the data frames if we consider that one register with one missing value, taking in count that we have 43 Features, is consider as null.

> Protocol DF >> N NaN rows for Heart Rate: 46 | Total rows: 2872533 | % of NaN rows: 0.0016
> Protocol DF >> NaN rows: 27665 | Original rows: 2872533 | % of NaN rows: 0.96
>
> Optional DF >> N NaN rows for Heart Rate: 54 | Total rows: 977972 | % of NaN rows: 0.0055
> Optional DF >> NaN rows: 3493 | Original rows: 977972 | % of NaN rows: 0.36

My choice is to delete all these NaN values. Of course they could be filled with other options. **I am open to any suggestion for this step**.

4. Data Cleaning and Wrangling

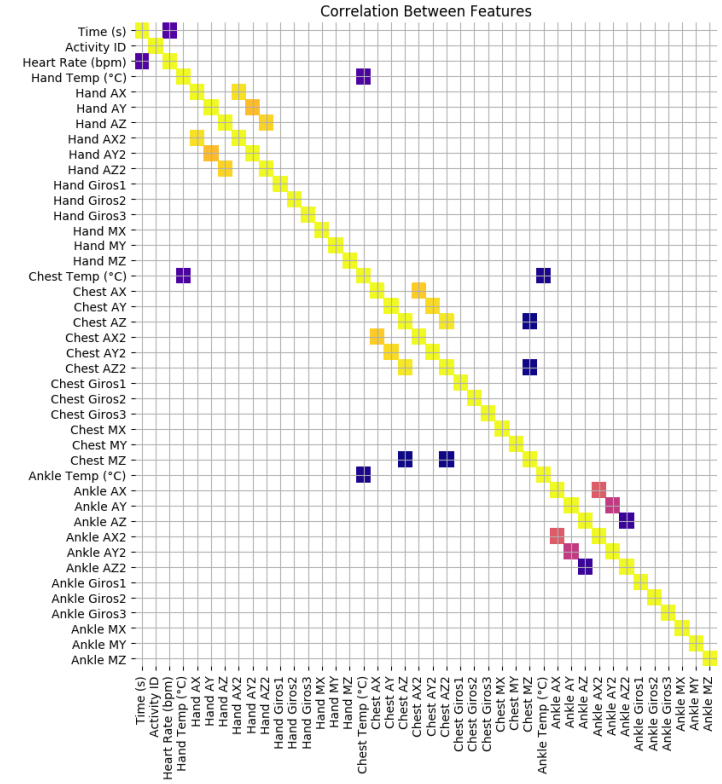
Once I have managed to drop all NaN values I have to drop even more. Why? Because the next note of the authors of the original Study:

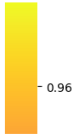
Data labeled with activityID=0 should be discarded in any kind of analysis. This data mainly covers transient activities between performing different activities, e.g. going from one location to the next activity's location, or waiting for the preparation of some equipment. Also, different parts of one subject's recording (in the case when the data collection was aborted for some reason) was put together during these transient activities (noticeable by some “jumping” in the HR-data).

Protocol DF rows for activity 0: 923391 > Total rows of the DF: 2844822 > 32.46 % of bad Data
Optional DF rows for activity 0: 194846 > Total rows of the DF: 974425   > 20.0 % of bad Data

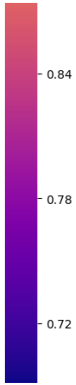
Also, I have made dropped the **Users ID** because is not longer needed in the study.

4. Data Cleaning and Wrangling

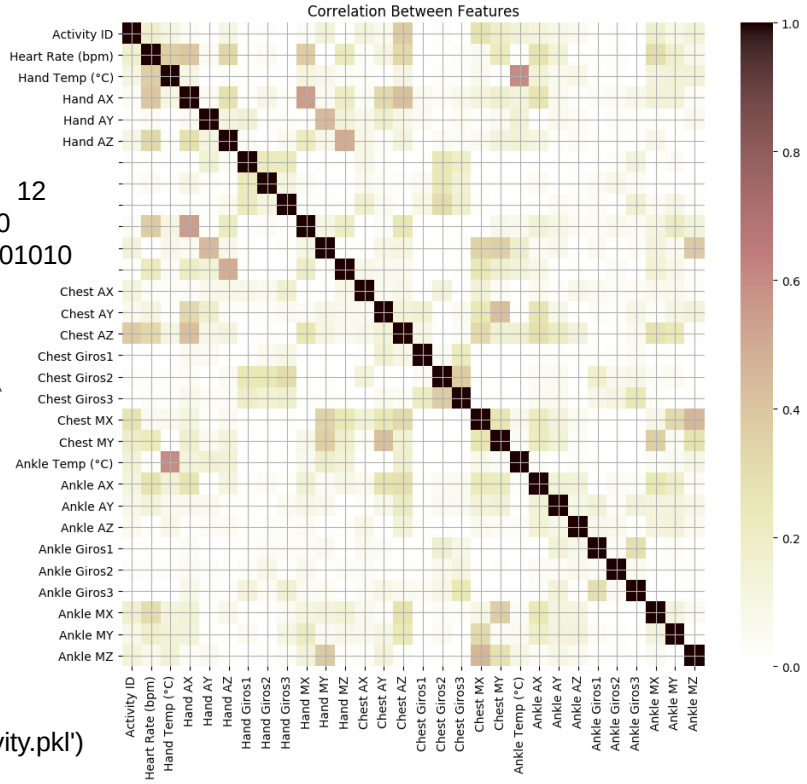




Number of Features dropped: 12
Final Features of columns: 30
Final number of registers: 2701010



data.to_pickle('physical_activity.pkl')



5. Model Test and Validation

5. Model Test

My first thought was to use unsupervised learning, **KMEANS**, to clusterize data. To do that, I am going to create a new Feature with only one purpouse, check if the clusterization has any meaning.

So first step, creating a JSON file (dictionary) with all the activities and activities ID to relate them to any kind of labelization. I have proposed the variable Intensity3 from 1-3 Scale and Intensity5 from 1-5 Scale. The porpouse of these variables are just to check how good have been the clusterization and to simplify the 18 different types of Activities.

ID	Name	intesity3	intensity5
1	lying	1	1
2	sitting	1	1
3	standing	1	2
4	walking	2	2
5	running	3	5
6	cycling	3	4
7	Nordic walking	3	3
9	watching TV	1	1
10	computer work	1	2
11	car driving	2	2
12	ascending stairs	2	4
13	descending stairs	2	3
16	vacuum cleaning	2	3
17	ironing	1	2
18	folding laundry	1	2
19	house cleaning	2	3
20	playing soccer	3	4
24	rope jumping	3	5

5. Model Test

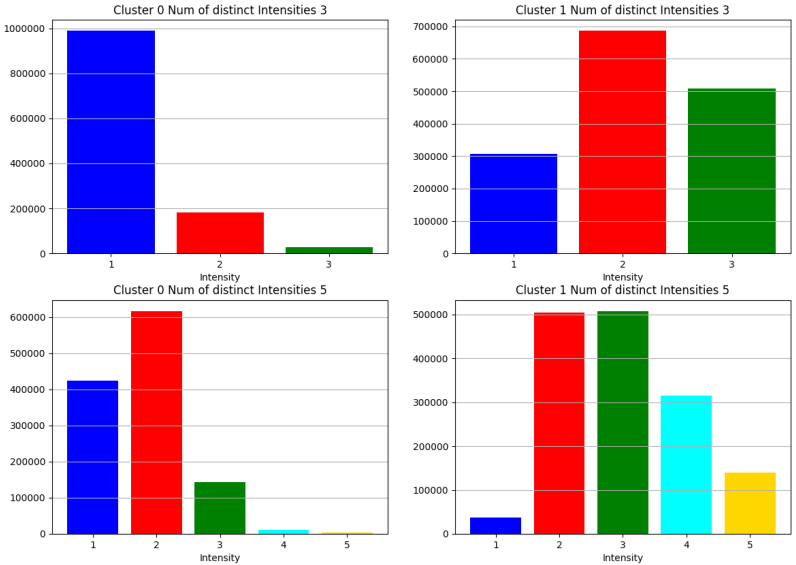
Once this has been done, I proceed to calculate the Features **Intensity3** and **Intensity5** with the values of the dictionary

```
if sum([type(item)==str for item in act_lab.keys()])!=0:
    df['Intensity3'] = list(map(lambda x: act_lab[str(x)]['intensity3'],df['Activity ID']))
    df['Intensity5'] = list(map(lambda x: act_lab[str(x)]['intensity5'],df['Activity ID']))
else:
    df['Intensity3'] = list(map(lambda x: act_lab[x]['intensity3'],df['Activity ID']))
    df['Intensity5'] = list(map(lambda x: act_lab[x]['intensity5'],df['Activity ID']))
```

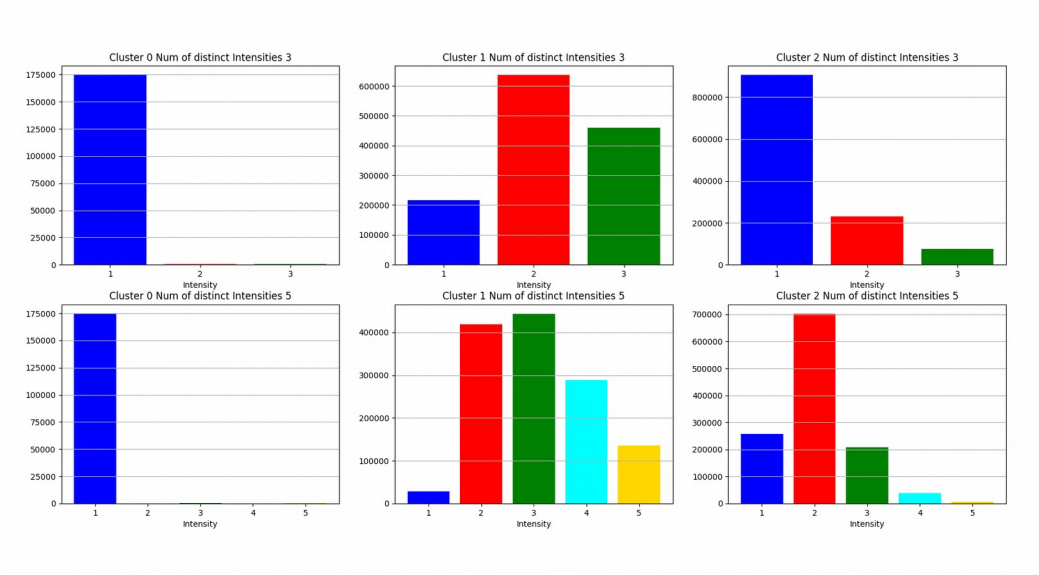
I drop the **Activity ID** Feature. Then, I use the Standard Scaler from sklearn module to scale all data not related with categorical Features (**Intensity3** and **Intensity5**).

5. Model Test - KMEANS

2 Clusters

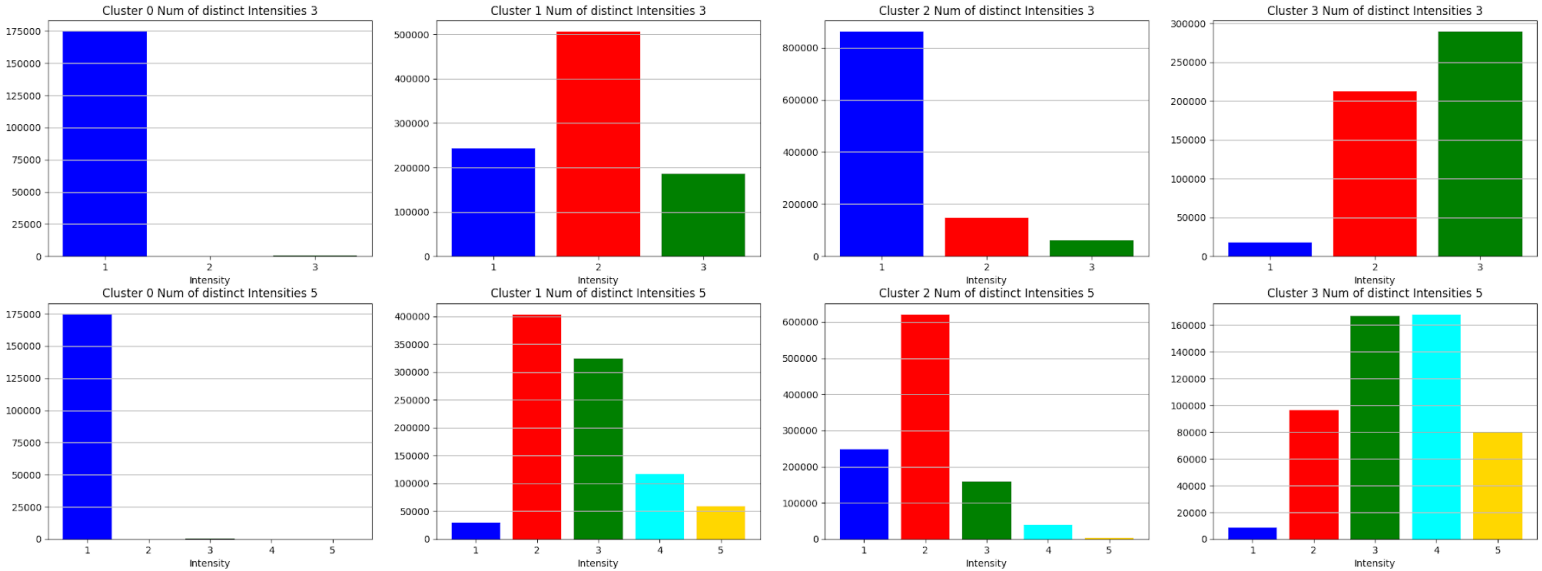


3 Clusters



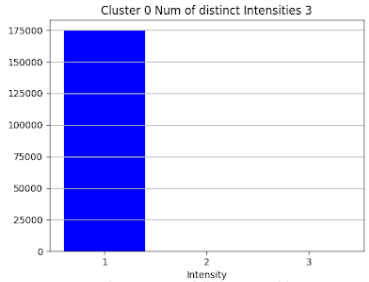
5. Model Test - KMEANS

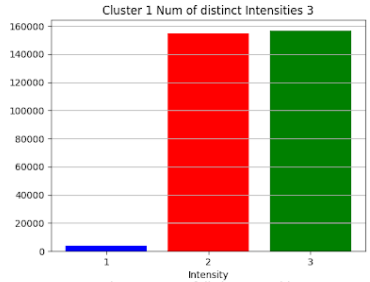
4 Clusters

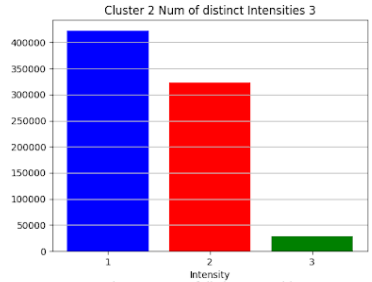


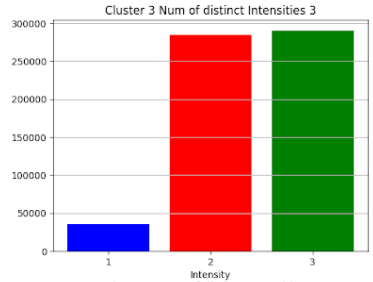
5. Model Test - KMEANS

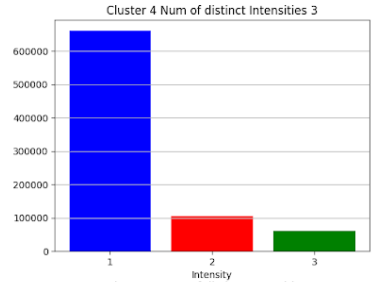
5 Clusters

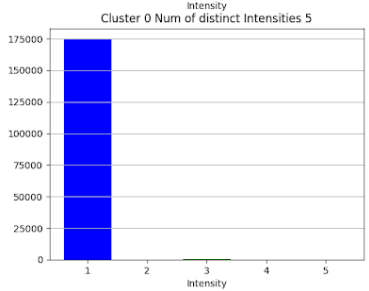


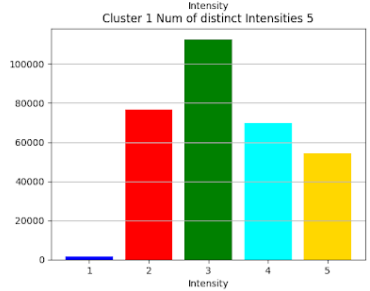


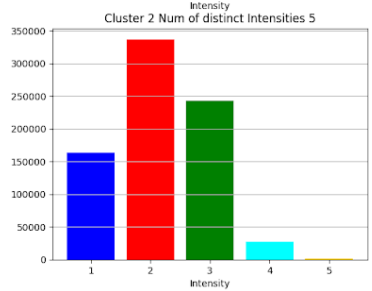


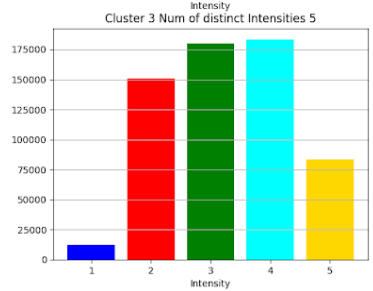


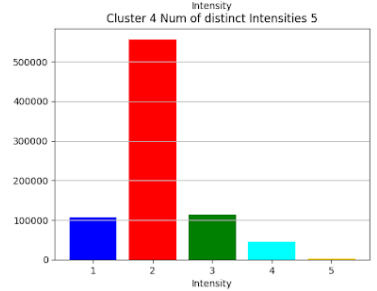










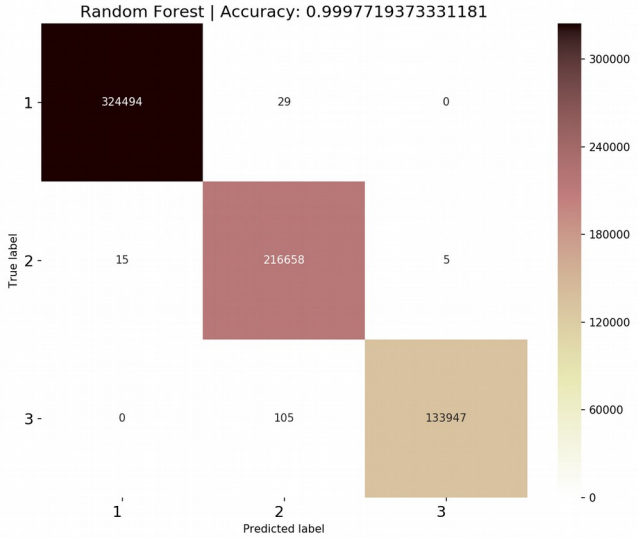


KMEANS Results: **Not** satisfactory!

5. Model Test - Random Forest

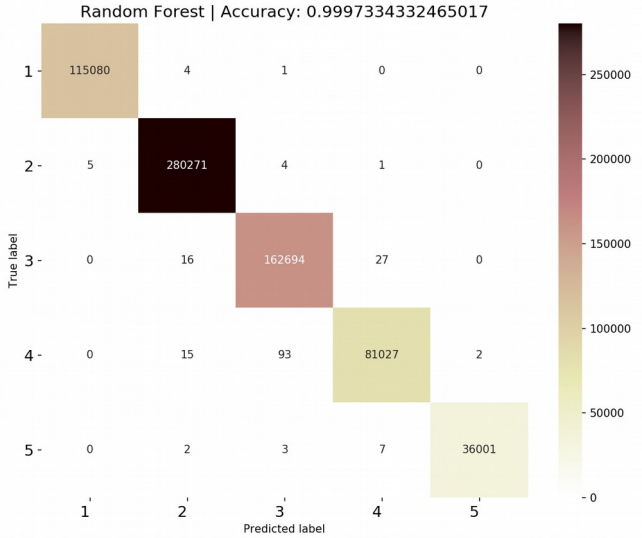
First I am going to check If random forest could do the job and compare the results when it has 3 categories for the target, **Intensity3**, tp the case when it has 5 categories, **Intensity5** and print the confusion matrix with its Accuracies.

3 Categories: Intensity3



NICE!

5 Categories: Intensity5



5. Model Test - Random Forest - Grid Search and Cross Validation

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'max_features': [3, 10], 'n_estimators': [10, 20], 'criterion':
['entropy']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)

GridSearchCV took 2873.80 seconds for 4 candidate parameter settings.
```

Model with rank: 1
Mean validation score: 0.783 (std: 0.042)
Parameters: {'criterion': 'entropy', 'max_features': 3, 'n_estimators': 20}

Model with rank: 2
Mean validation score: 0.775 (std: 0.041)
Parameters: {'criterion': 'entropy', 'max_features': 3, 'n_estimators': 10}

Model with rank: 3
Mean validation score: 0.760 (std: 0.060)
Parameters: {'criterion': 'entropy', 'max_features': 10, 'n_estimators': 10}

CPU times: user 5min 51s, sys: 2.78 s, total: 5min 54s
Wall time: 47min 53s

6. Conclusions

6. Conclusions

Well, seems that the best model for this dataset is the Random Forest with:

Model with rank: 1
Mean validation score: 0.783 (std: 0.042)
Parameters: {'criterion': 'entropy', 'max_features': 3, 'n_estimators': 20}

This could be better optimized with time and better equipment so finally we can use this model to predict which activity is going on and guide our users in a training course to improve his physical skills.