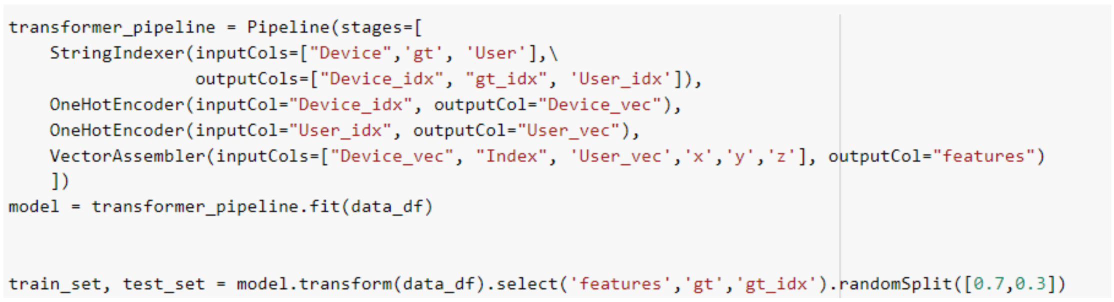
Project Part 2 Question 2

After the insights gained in the previous question, we decided to use a Random Forest model.

That's because we noticed the special behavior of x,y,z for every activity and predicted a module of a tree kind will be able to find the thresholds that pinpoint every activity.

First we used a pipeline as such:

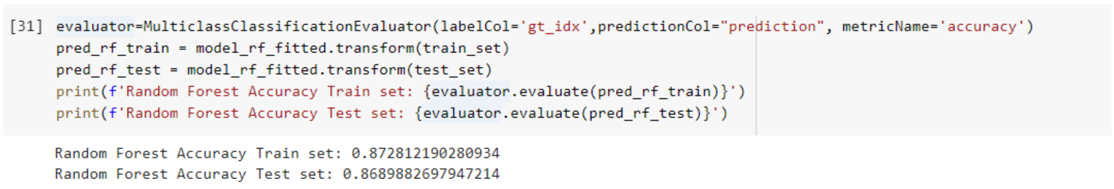


```
transformer_pipeline = Pipeline(stages=[
    StringIndexer(inputCols=["Device",'gt', 'User'],\
                  outputCols=["Device_idx", "gt_idx", 'User_idx']),
    OneHotEncoder(inputCol="Device_idx", outputCol="Device_vec"),
    OneHotEncoder(inputCol="User_idx", outputCol="User_vec"),
    VectorAssembler(inputCols=["Device_vec", "Index", 'User_vec','x','y','z'], outputCol="features")
    ])
model = transformer_pipeline.fit(data_df)


train_set, test_set = model.transform(data_df).select('features','gt','gt_idx').randomSplit([0.7,0.3])
```

We used the device, because it already contains the information of the column "Model" and we noticed that the pairing of Device, User and Index are unique. that means, that we can use them together and then index will function as a timestamp correctly.

We then transformed all strings into numbers, and used one hot encoding in order to ignore the order (if user a is 0 and user f is 5, then a is more similar to user 2 then f which is adding noise) and we decided to do the same for Device, but only out of convenience, since it has only two values in this static data set.

The Accuracy we got:



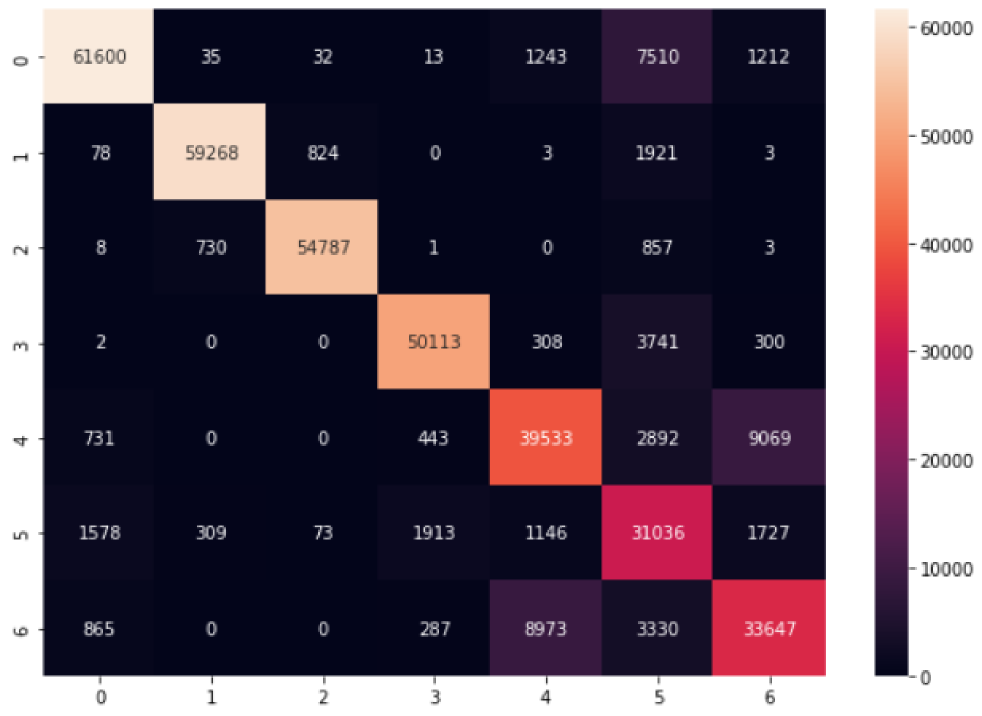
```
[31] evaluator=MulticlassClassificationEvaluator(labelCol='gt_idx',predictionCol="prediction", metricName='accuracy')
     pred_rf_train = model_rf_fitted.transform(train_set)
     pred_rf_test = model_rf_fitted.transform(test_set)
     print(f'Random Forest Accuracy Train set: {evaluator.evaluate(pred_rf_train)}')
     print(f'Random Forest Accuracy Test set: {evaluator.evaluate(pred_rf_test)}')

     Random Forest Accuracy Train set: 0.872812190280934
     Random Forest Accuracy Test set: 0.8689882697947214
```

The hyperparameters where tuned in regards to better the space complexity for question 3.

Visualization:

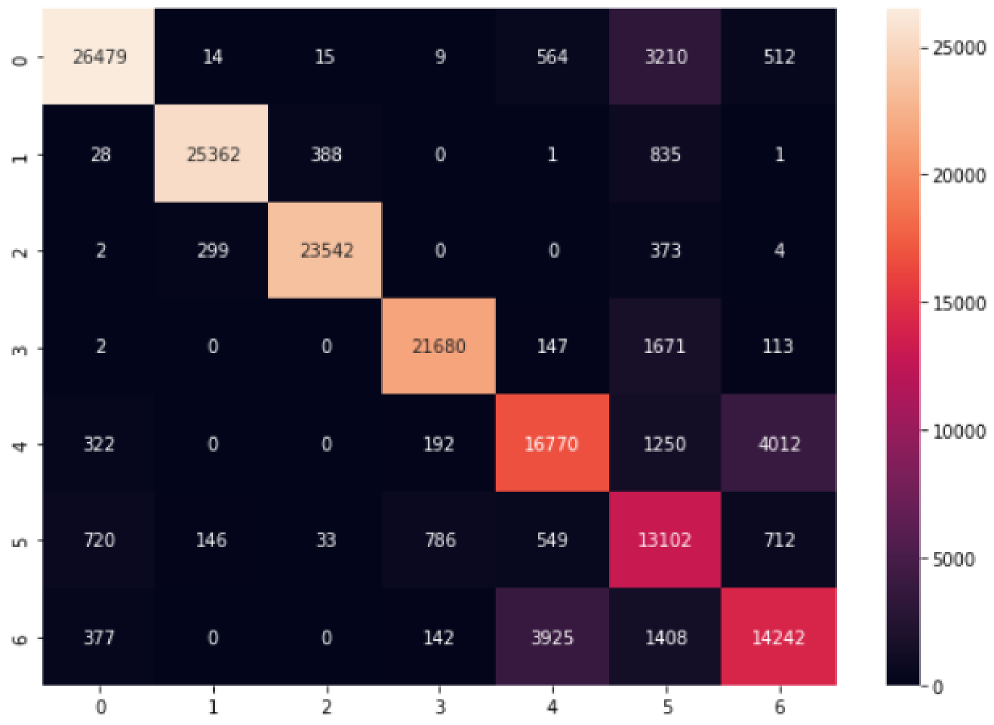
We created a confusion matrix for both train and test set.

Train:



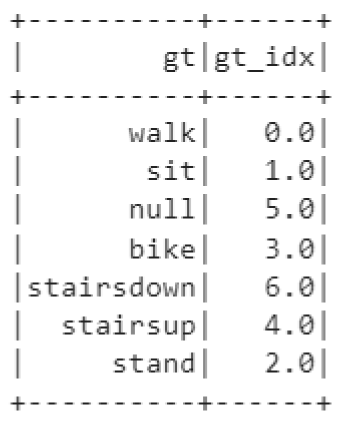
	0	1	2	3	4	5	6
0	61600	35	32	13	1243	7510	1212
1	78	59268	824	0	3	1921	3
2	8	730	54787	1	0	857	3
3	2	0	0	50113	308	3741	300
4	731	0	0	443	39533	2892	9069
5	1578	309	73	1913	1146	31036	1727
6	865	0	0	287	8973	3330	33647

Test:



	0	1	2	3	4	5	6
0	26479	14	15	9	564	3210	512
1	28	25362	388	0	1	835	1
2	2	299	23542	0	0	373	4
3	2	0	0	21680	147	1671	113
4	322	0	0	192	16770	1250	4012
5	720	146	33	786	549	13102	712
6	377	0	0	142	3925	1408	14242

The Legend is :



```
+----------+------+
|        gt|gt_idx|
+----------+------+
|      walk|   0.0|
|       sit|   1.0|
|      null|   5.0|
|      bike|   3.0|
|stairsdown|   6.0|
|  stairsup|   4.0|
|     stand|   2.0|
+----------+------+
```

As we can see, the model is pretty successful. most of the errors happened between label 5 (null) and other labels which can be explain by our explanation in question 1 that null is just switching between two activities - thus being very similar to one of them and labeled incorrectly.