In [ ]:

```
#imports and setup
import requests
import pandas as pd
import numpy as np
import json
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from datetime import datetime, timedelta
token = 'FF0D4AB80BDB63716462F02BB9291897'
from functions import *
from process_tests import *
```

I've decided to make the functions in a seperate Python file for readability.

Lets start with getting the unique IDs for all participants...

In [ ]:

```
listOfParticipantIds(token)
```

Out[ ]:

```
['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11']
```

Here we can see that this function gives us back a list of all of the participant IDs in our record. Lets now try to get a general overview of our data by getting the participant ID and type of experiment for each recorded test. The data is by default given in this way, with each test being a seperate entry, and its worth noting that the same participant can have multiple entries for the same test.

In [ ]:

```
raw = allTestsOverview(token)
# print(raw) #uncomment this to look at the json output
print("Here are all entries in current dataset:")
for r in raw:
    print("Participant ID: "+str(r.get('participant_id'))+", Test Type: "+r.get('test_type').upper
```

```
Here are all entries in current dataset:
Participant ID: 1, Test Type: DEM
Participant ID: 2, Test Type: DEM
Participant ID: 2, Test Type: RAN
Participant ID: 2, Test Type: RAN
Participant ID: 2, Test Type: GAI
Participant ID: 3, Test Type: DEM
Participant ID: 4, Test Type: DEM
Participant ID: 5, Test Type: DEM
Participant ID: 5, Test Type: RAN
Participant ID: 5, Test Type: RAN
Participant ID: 5, Test Type: GAI
Participant ID: 5, Test Type: TAP
Participant ID: 5, Test Type: PEG
Participant ID: 5, Test Type: PEG
Participant ID: 5, Test Type: TIM
Participant ID: 6, Test Type: DEM
Participant ID: 7, Test Type: DEM
Participant ID: 8, Test Type: DEM
Participant ID: 9, Test Type: DEM
Participant ID: 9, Test Type: RAN
Participant ID: 10, Test Type: DEM
Participant ID: 10, Test Type: RAN
Participant ID: 11, Test Type: DEM
```

We see here that we have categorized the type of test that each entry represents. These three letter shorthands will be standard throughout the experiment.

Lets now start to look at the data. We will start with participant 5 as they have the most entered tests.

In [ ]:

```
data = allDataForParticipant(5, token)
print(str(data)[0:3000])
print()
print("Number of Characters in Output: "+str(len(str(data))))
```

```
[{'participant_id': '5', 'redcap_repeat_instrument': '', 'redcap_repeat_instance': '', 'dem_firstn
ame': 'Sam', 'dem_lastname': 'Test', 'dem_zerodate': '', 'dem_code': 'U-N6HNJJUYURXJZ6TD4FTT', 'de
m_joindate': '2023-01-13 17:05:18', 'dem_pushids': '["e7uRyDJqTkMuqAob2nZUpO:APA91bFMrg2lvVUScPuWr
LxQeF-lm26tyG_9MduL15_i0OnxcazaZP6koyLlV4gRToI_aHnBoHF68_tsahfeL4h3gTx9dSFxQHiP8ZpFgfTiREsCYnRRXyb
hIOmcgmT-4fjL_SA3D2ie"]', 'demographic_complete': '0', 'ran_uuid': '', 'ran_startdate': '', 'ran_e
nddate': '', 'ran_scheduledate': '', 'ran_status': '', 'ran_supplementaldata': '', 'ran_serialized
result': '', 'ran_flexion': '', 'ran_extension': '', 'ran_devicemotion': '', 'range_of_motion_comp
lete': '', 'gai_uuid': '', 'gai_startdate': '', 'gai_enddate': '', 'gai_scheduledate': '', 'gai_st
atus': '', 'gai_supplementaldata': '', 'gai_serializedresult': '', 'gai_outacc': '', 'gai_outdevic
e': '', 'gai_returnacc': '', 'gai_returndevice': '', 'gai_restacc': '', 'gai_restdevice': '', 'gai
t_walking_complete': '', 'tap_uuid': '', 'tap_startdate': '', 'tap_enddate': '', 'tap_scheduledat
e': '', 'tap_status': '', 'tap_supplementaldata': '', 'tap_serializedresult': '', 'tap_leftjson':
'', 'tap_leftaccelerometer': '', 'tap_rightjson': '', 'tap_rightaccelerometer': '', 'tapping_compl
ete': '', 'peg_uuid': '', 'peg_startdate': '', 'peg_enddate': '', 'peg_scheduledate': '', 'peg_sta
tus': '', 'peg_supplementaldata': '', 'peg_serializedresult': '', 'peg_dom_place': '', 'peg_dom_re
move': '', 'peg_nondom_place': '', 'peg_nondom_remove': '', 'peg_test_complete': '', 'tim_uuid':
'', 'tim_startdate': '', 'tim_enddate': '', 'tim_scheduledate': '', 'tim_status': '', 'tim_supplem
entaldata': '', 'tim_serializedresult': '', 'tim_trial1': '', 'tim_turnaround': '', 'tim_trial2':
'', 'timed_walk_complete': '', 'tes948_uuid': '', 'tes948_startdate': '', 'tes948_enddate': '', 't
es948_scheduledate': '', 'tes948_status': '', 'tes948_supplementaldata': '', 'tes948_serializedres
ult': '', 'tes948_json': '', 'test_taks_complete': ''}, {'participant_id': '5', 'redcap_repeat_ins
trument': 'range_of_motion', 'redcap_repeat_instance': 1, 'dem_firstname': '', 'dem_lastname': '',
'dem_zerodate': '', 'dem_code': '', 'dem_joindate': '', 'dem_pushids': '', 'demographic_complete':
'', 'ran_uuid': '76C7FDD9-A151-4017-835A-AB4BB1502F03', 'ran_startdate': '2023-01-13 17:09:59', 'r
an_enddate': '2023-01-13 17:10:42', 'ran_scheduledate': '', 'ran_status': '1', 'ran_supplementalda
ta': '{\n  "devicemanufacturer" : "Apple",\n  "deviceplatform" : "iOS",\n  "syncdate" : "2023-01-1
3 17:10:42",\n  "configversion" : "1",\n  "deviceuuid" : "AA59EA11-CF26-450E-80E8-853A62FC9221",\n
"deviceversion" : "16.0.2",\n  "appversion" : "2.20.0",\n  "devicemodel" : "iPhone",\n  "percentCo
mplete" : "100"\n}', 'ran_serializedresult': 'result.zip', 'ran_flexion': '', 'ran_extension': '',
'ran_devicemotion': '', 'range_of_motion_complete': '0', 'gai_uuid': '', 'gai_startdate': '', 'gai
_enddate': '', 'gai_scheduledate': '', 'gai_status': '',

Number of Characters in Output: 65199
```

Ok, that looks like its a looot of stuff. Lets focus in on the demographic information for now

In [ ]:

```
data = oneTypeOfTestForParticipant(5, 'dem', token)
# print(data) # uncomment to see uncleaned data with extra unused variables.
data = cleanTest(data) # by default this returns the first test in the inputted data.
print(data)
```

```
{'dem_firstname': 'Sam', 'dem_lastname': 'Test', 'dem_zerodate': '', 'dem_code': 'U-N6HNJJUYURXJZ6
TD4FTT', 'dem_joindate': '2023-01-13 17:05:18', 'dem_pushids': '["e7uRyDJqTkMuqAob2nZUpO:APA91bFMr
g2lvVUScPuWrLxQeF-lm26tyG_9MduL15_i0OnxcazaZP6koyLlV4gRToI_aHnBoHF68_tsahfeL4h3gTx9dSFxQHiP8ZpFgfT
iREsCYnRRXybhIOmcgmT-4fjL_SA3D2ie"]', 'demographic_complete': '0', 'participant_id': '5'}
```

Beautiful, this is starting to look somewhat nicer. Lets try to see how this works on the easiest to examine test, the peg test. For this participant we also know that two of this type of test was recorded, so lets look at the second test.

In [ ]:

```
data = cleanTest(oneTypeOfTestForParticipant(5,'peg',token)[1])
print(data)
```

```
{'peg_uuid': 'E59C9115-457F-41E8-B3A2-34AB5F1A86A4', 'peg_startdate': '2023-01-29 19:57:29', 'peg_
enddate': '2023-01-29 19:59:21', 'peg_scheduledate': '', 'peg_status': '1', 'peg_supplementaldat
a': '{\n  "percentComplete" : "100",\n  "syncdate" : "2023-01-29 19:59:21",\n  "devicemanufacture
r" : "Apple",\n  "appversion" : "2.20.0",\n  "devicemodel" : "iPhone",\n  "configversion" : "4",\n
"deviceuuid" : "AA59EA11-CF26-450E-80E8-853A62FC9221",\n  "deviceversion" : "16.2",\n  "deviceplat
form" : "iOS"\n}', 'peg_serializedresult': 'result.zip', 'peg_dom_place': '{"rotated":"false","tot
alTime":"37.11786541599997","movingDirection":"Left","samples":[{"distance":"242.2830200195312
5","time":"1.2701712917187251"},{"distance":"258.8034973144531","time":"7.308940125047229"},{"dist
ance":"282.5774230957031","time":"2.9633507916587405"},{"distance":"245.19444274902344","time":"6.
510091916657984"},{"distance":"244.26036071777344","time":"2.8087082499987446"},{"distance":"239.2
8343200683594","time":"2.4663711666944437"},{"distance":"277.4166564941406","time":"4.083882166654
803"},{"distance":"284.6515502929687 5","time":"7.516064999974333"},{"distance":"283.3947143554687
5","time":"2.1979512500111014"}],"threshold":"0.2","totalSuccesses":"9","totalFailures":"9","total
Distance":"2357.8650970458984","dominantHandTested":"true","numberOfPegs":"9"}', 'peg_dom_remove':
'{"threshold":"0.2","rotated":"false","totalFailures":"6","totalTime":"23.374694166000012","totalS
uccesses":"9","totalDistance":"2758.103271484375","movingDirection":"Right","dominantHandTeste
d":"true","numberOfPegs":"9","samples":[{"distance":"289.8866271972656","time":"4.05745370831573
4"},{"distance":"357.08306884765625","time":"2.9833075416390784"},{"distance":"282.710327148437
5","time":"1.6166018333169632"},{"distance":"303.57916259765625","time":"1.6248125416459516"},{"di
stance":"325.55078125","time":"6.991270125028677"},{"distance":"306.2516174316406","time":"1.48331
83749578893"},{"distance":"299.4362487792969","time":"1.5502688332926482"},{"distance":"297.297943
1152344","time":"1.500050500035286"},{"distance":"296.3074951171875","time":"1.558081249997485
4"}]}', 'peg_nondom_place': '{"totalTime":"19.29132149999998","totalDistance":"2374.44969177246
1","totalFailures":"1","samples":[{"distance":"251.24652099609375","time":"3.304625541670248"},{"d
istance":"251.4341430664062 5","time":"2.1164832083159126"},{"distance":"263.5380554199219","tim
e":"2.0833316250354983"},{"distance":"266.47747802734375","time":"2.00920929166022 7"},{"distanc
e":"268.567626953125","time":"1.7485805416363291"},{"distance":"270.9816589355469","time":"2.27498
82916687056"},{"distance":"268.247314453125","time":"1.74143845832441"},{"distance":"288.946563720
7031","time":"2.0253207499627024"},{"distance":"245.0103302001953","time":"1.981629041663836
7"}],"dominantHandTested":"false","movingDirection":"Right","threshold":"0.2","numberOfPeg
s":"9","totalSuccesses":"9","rotated":"false"}', 'peg_nondom_remove': '{"threshold":"0.2","rotate
d":"false","totalFailures":"1","totalTime":"14.759709790999977","totalSuccesses":"9","totalDistanc
e":"2794.0029296875","movingDirection":"Left","dominantHandTested":"false","numberOfPegs":"9","sam
ples":[{"distance":"324.8277282714844","time":"2.4279085416928865"},{"distance":"291.3610229492187
5","time":"1.2987260000081733"},{"distance":"312.922119140625","time":"1.2512457916745916"},{"dist
ance":"324.5309143066406","time":"1.5660187083412893"},{"distance":"302.01531982421875","time":"1.
300726666697301"},{"distance":"286.5397033691406","time":"1.299611166701652"},{"distance":"297.311
7980957031","time":"1.2830854166531935"},{"distance":"314.1058654785156","time":"1.392887875030282
9"},{"distance":"340.3884582519531","time":"2.9330882916692644"}]}', 'peg_test_complete': '0', 'pa
rticipant_id': '5'}
```

2/10/2023

I've been working with a general pattern on the process for how we go about using these tests. Functions will be used in the following manner: get->clean->process with the process part including the downloading and placement of zipped data files. The following cell does all of this for participant 5's Gait Walking test

In [ ]:

```
d = process_gai(cleanTest(oneTypeOfTestForParticipant(5,'gai',token)), token)
print(d.keys())
print("# Obs. = "+str(len(pd.json_normalize(d['gai_outacc']))))
df_outdevice = dictListToDF(d['gai_outdevice'])
df_outacc = dictListToDF(d['gai_outacc'])
dictListToDF(d['gai_outdevice']).head()
```

```
dict_keys(['gai_uuid', 'gai_startdate', 'gai_enddate', 'gai_scheduledate', 'gai_status', 'gai_supp
lementaldata', 'gai_serializedresult', 'gai_outacc', 'gai_outdevice', 'gai_returnacc', 'gai_return
device', 'gai_restacc', 'gai_restdevice', 'gait_walking_complete', 'participant_id'])
# Obs. = 2825
```

Out[ ]:

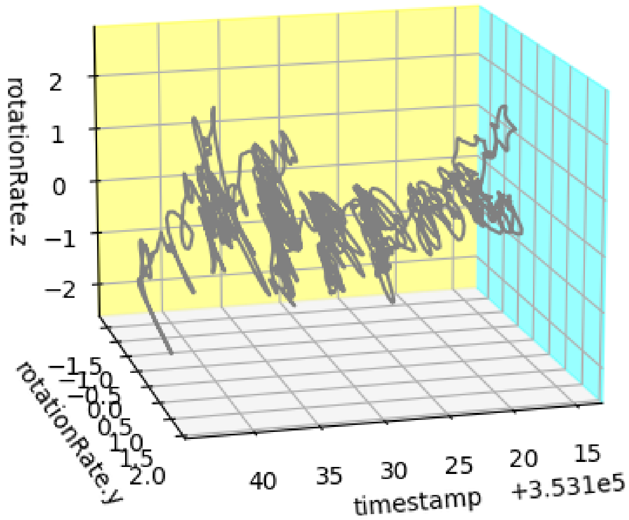
| | timestamp | attitude.y | attitude.w | attitude.z | attitude.x | rotationRate.x | rotationRate.y | rotationRate.z | userAc |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 353114.990463 | -0.088940 | 0.960291 | 1.387779e-17 | 0.264445 | -0.049830 | -0.027565 | 0.126862 | |
| **1** | 353115.000471 | -0.089232 | 0.960348 | 5.710771e-04 | 0.264137 | -0.053728 | -0.023853 | 0.132588 | |
| **2** | 353115.010479 | -0.089525 | 0.960414 | 1.133002e-03 | 0.263797 | -0.063731 | -0.029999 | 0.122687 | |
| **3** | 353115.020487 | -0.089858 | 0.960479 | 1.604844e-03 | 0.263446 | -0.063301 | -0.046252 | 0.107445 | |
| **4** | 353115.030495 | -0.090239 | 0.960534 | 2.007688e-03 | 0.263110 | -0.052569 | -0.059569 | 0.102781 | |

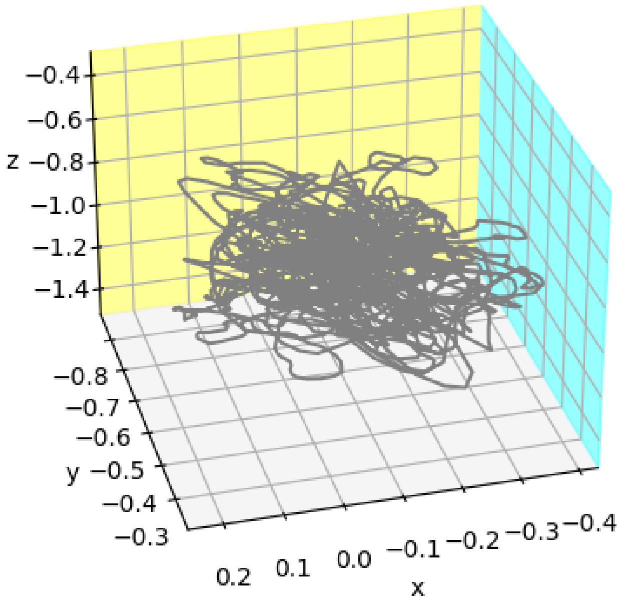
For the walking test I've begun to start looking at various features in the timeseries data through 3D graphs. I've also made a basic wrapper to do so...

In [ ]:

```
ThreeDPlot(df_outdevice, 'timestamp', 'rotationRate.y', 'rotationRate.z')
ThreeDPlot(df_outacc, 'x', 'y', 'z',  view_angle_x=30)
```

```
c:\Users\benst\OneDrive\Desktop\CodingProjects\DegenCervicalMyleopathyLab\functions.py:124: UserWa
rning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  ax.scatter3D(data[x], data[y], data[z], c=c, cmap='Greens')
```



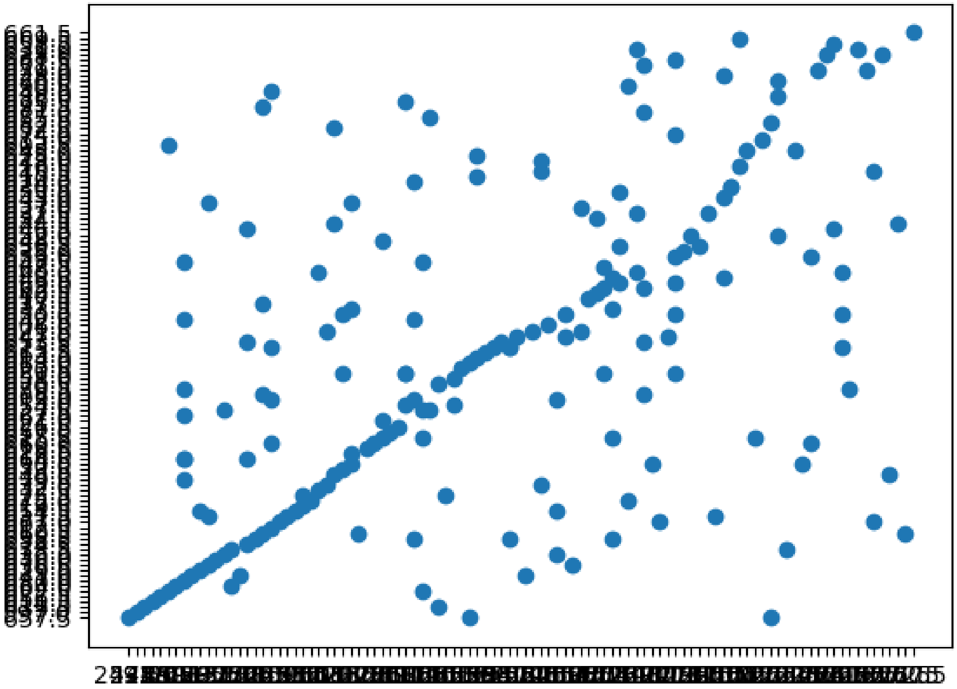


Here is some exploration for the tapping test... as you can see there seems to be some strange stuff happening here although I'd like to do the test myself to see if this kind of strange data is reproducible.

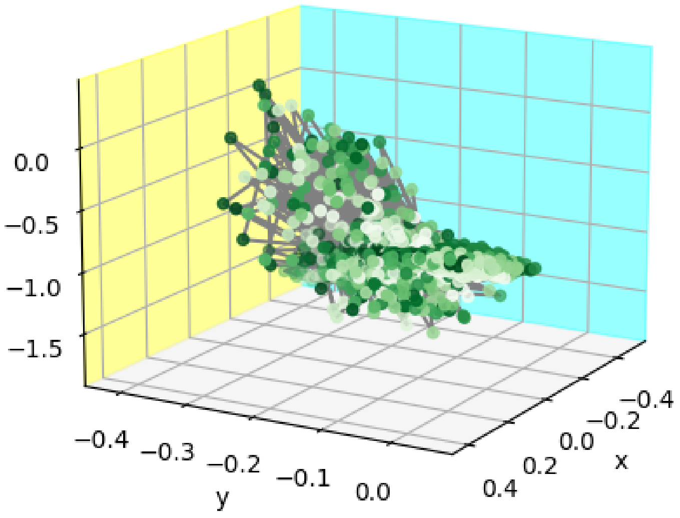
In [ ]:

```
d = process_tap(cleanTest(oneTypeOfTestForParticipant(5,'tap',token)),token)
print(d.keys())
df_tap_leftjson_samples = dictListToDF(d['tap_leftjson.samples'])
df_tap_leftaccelerometer = dictListToDF(d['tap_leftaccelerometer'])
print(df_tap_leftaccelerometer.columns)
plt.scatter(df_tap_leftjson_samples['locationX'],df_tap_leftjson_samples['locationY'])
plt.show()
ThreeDPlot(df_tap_leftaccelerometer, 'x', 'y', 'z', c='timestamp', view_angle_y=30)
```

```
dict_keys(['tap_uuid', 'tap_startdate', 'tap_enddate', 'tap_scheduledate', 'tap_status', 'tap_supp
lementaldata', 'tap_serializedresult', 'tap_leftaccelerometer', 'tap_rightaccelerometer', 'tapping
_complete', 'participant_id', 'tap_leftjson.samples', 'tap_leftjson.stepViewSize', 'tap_leftjson.b
uttonRect1', 'tap_leftjson.buttonRect2', 'tap_rightjson.samples', 'tap_rightjson.stepViewSize', 't
ap_rightjson.buttonRect1', 'tap_rightjson.buttonRect2'])
Index(['y', 'timestamp', 'z', 'x'], dtype='object')
```



```
0       353756.127247
1       353756.137255
2       353756.147263
3       353756.157271
4       353756.167280
             ...
1998    353776.123521
1999    353776.133529
2000    353776.143537
2001    353776.153546
2002    353776.163554
Name: timestamp, Length: 2003, dtype: float64
```



The peg test is much simpler in terms of the data. No pulling of zipped files.

```
In [ ]: d=cleanTest(oneTypeOfTestForParticipant(5,'peg',token))
In [ ]: print(d.keys())
        d['peg_nondom_place']
```

```
dict_keys(['peg_uuid', 'peg_startdate', 'peg_enddate', 'peg_scheduledate', 'peg_status', 'peg_supp
lementaldata', 'peg_serializedresult', 'peg_dom_place', 'peg_dom_remove', 'peg_nondom_place', 'peg
_nondom_remove', 'peg_test_complete', 'participant_id'])
```

```
Out[ ]: '{"dominantHandTested":"false","movingDirection":"Right","totalDistance":"2323.0948944091797","sam
ples":[{"time":"3.7110564166796394","distance":"241.86631774902344"},{"time":"2.27512862498406
3","distance":"251.11203002929688"},{"time":"1.824287874973379","distance":"241.75323486328125"},
{"time":"2.4766482916893438","distance":"264.37060546875"},{"time":"2.482281458331272","distanc
e":"269.09210205078125"},{"time":"5.399795499979518","distance":"214.89895629882812"},{"time":"2.3
99718041648157","distance":"278.7796936035156"},{"time":"2.9429723333450966","distance":"283.79064
94140625"},{"time":"2.6818896666518413","distance":"277.4313049316406"}],"rotated":"false","number
OfPegs":"9","totalSuccesses":"9","threshold":"0.2","totalTime":"26.205035124999995","totalFailure
s":"3"}'
```

... Although its not as simple as the timed walk a.k.a. my favorite test. Here it looks like there are just three variables, although currently this data shows that the test may be incomplete so would need to redo to be certain.

```
In [ ]: d=cleanTest(oneTypeOfTestForParticipant(5,'tim',token))
        d.keys()
```

```
Out[ ]: dict_keys(['tim_uuid', 'tim_startdate', 'tim_enddate', 'tim_scheduledate', 'tim_status', 'tim_supp
lementaldata', 'tim_serializedresult', 'tim_trial1', 'tim_turnaround', 'tim_trial2', 'timed_walk_c
omplete', 'participant_id'])
```

```
In [ ]: d
```

```
Out[ ]: {'tim_uuid': 'A3B1C02A-446A-47C4-9F5B-D35D195BDBE0',
 'tim_startdate': '2023-01-15 08:34:10',
 'tim_enddate': '2023-01-15 08:34:56',
 'tim_scheduledate': '',
 'tim_status': '1',
 'tim_supplementaldata': '{\n  "deviceuuid" : "AA59EA11-CF26-450E-80E8-853A62FC9221",\n  "devicemo
del" : "iPhone",\n  "percentComplete" : "100",\n  "deviceversion" : "16.0.2",\n  "syncdate" : "202
3-01-15 08:34:56",\n  "configversion" : "4",\n  "deviceplatform" : "iOS",\n  "appversion" : "2.20.
0",\n  "devicemanufacturer" : "Apple"\n}',
 'tim_serializedresult': 'result.zip',
 'tim_trial1': '14.202383166',
 'tim_turnaround': '3.046138666',
 'tim_trial2': '8.246413415999999',
 'timed_walk_complete': '0',
 'participant_id': '5'}
```