Data Mining: Individual Project I Report

Q1. Take samples to explore what types of users' lifelogging tags have. What type of data should be for each attribute? Give some sampling examples to explore the given data.

Data scientists use sampling methods to save resources and explore the data on a smaller scale. There are several ways to sample the data, and I will introduce two methods. The first method is simple random sampling, meaning that the computer creates a subset of data entirely randomly from the original data.

```
import pandas as pd

sensor = pd.read_csv('device_uplink.csv')
sensor['client_time'] = pd.to_datetime(sensor.client_time, format='%Y-%m-%d %H:%M:%S')
sample1 = sensor.sample(frac=.1)
```

It is the code for simple random sampling. I converted the data type of 'client_time' from object to datetime64 for further use. Sample 1 has a size of 10% compared to the raw data.

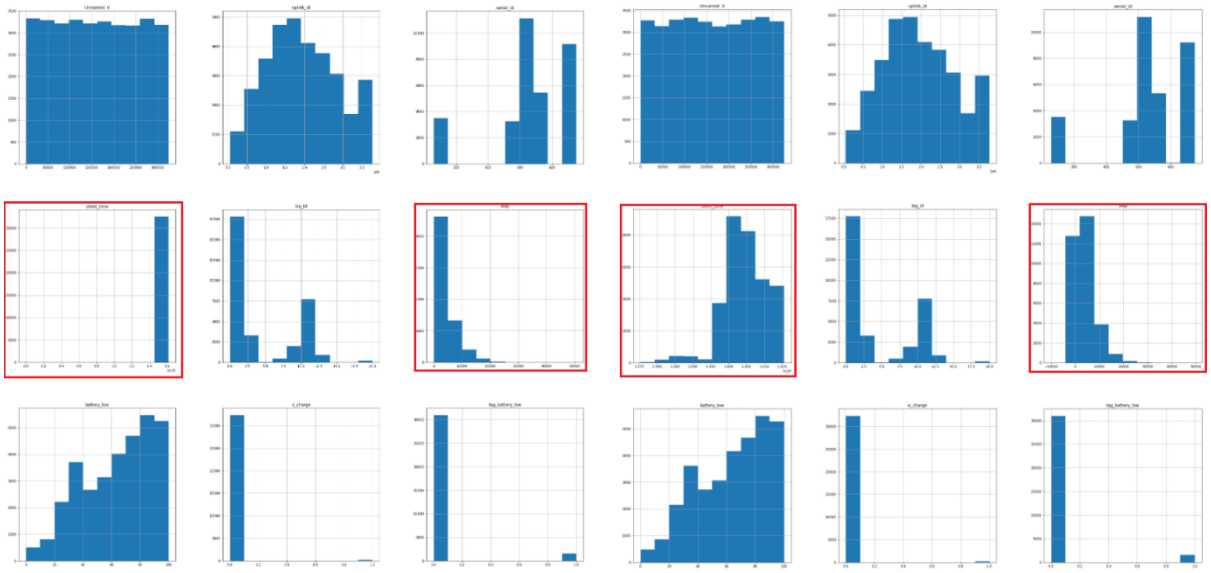
The second method is to add weights when sampling. Adding a large amount of weight to a specific area of the raw data results in more samples from that area.

```
sensor['client_time']= pd.to_numeric(sensor.client_time)
sample2 = sensor.sample(frac=.1, weights='client_time')
```

I converted the data type of 'client_time' from datatime64 to int64 to use 'client_time' as a weight factor. (The more recent it is, the more likely it is to be included in the sample.)

```
sample1.hist(figsize=(30,30))
sample2.hist(figsize=(30,30))
```

Code A



Sample1 histogram **Sample2 histogram**

I created the histogram to approximate the distribution of variables with code A. As the weight changed, the distribution of the histogram of the right red box(‘step’) also changed. Attempts of multiple sampling methods can be the first step in understanding the dataset. For example, it is possible to try to infer the correlation between ‘step’ and ‘client_time’ due to the change in the histogram.

We can also determine the type of variables through samples and domain knowledge. The greatest difference between a numerical variable and a categorical variable is whether the variable has a quantitative characteristic. In other words, if the number itself in the variable contains information without natural language explanation, that variable is the numerical variable. ‘uplink_id’, ‘owner_id’, and ‘tag_id’ are the categorical variables because they are numbers that people have put together for classification. ‘is_charge’ and ‘tag_battery_low’ are binary variables, therefore they are also categorical variables. The number itself has no meaning for these variables. Conversely, ‘client_time’, ‘step’, ‘battery_low’ are numerical variables since they have quantitative meanings in their numbers. Numerical variables and categorical variables can be transformed into each other through methods such as dummy variables and aggregation.

Q2. There are some cases that are incomplete or uncertain since it is raw data recorded in daily life. In this case, data values may not be provided. Then, how do you solve these missing values? Explain how to handle this problem with justification.

```
sensor.isna().sum()
```

```
Unnamed: 0          0
uplink_id           0
owner_id            0
client_time         0
tag_id           1206
step                0
battery_low         0
is_charge           0
tag_battery_low     0
dtype: int64
```

To find the missing values from the raw data, I counted the number of null values from 'device_uplink.csv' with the following code. The code says there is no missing value except 'tag_id'. This dataset is collected to track user information and provide a suitable life for users. For that purpose, the 'tag_id', which informs where the user is, is important information. Therefore, if 'tag_id' has a null value, I decided to delete those data because it might confuse the observation since it has no tag value..

```
sensor.count()
```

```
Unnamed: 0          324823
uplink_id           324823
owner_id            324823
client_time         324823
tag_id              323617
step                324823
battery_low         324823
is_charge           324823
tag_battery_low     324823
dtype: int64
```

```
sensor = sensor.dropna(axis=0)
sensor.count()
```

```
Unnamed: 0          323617
uplink_id           323617
owner_id            323617
client_time         323617
tag_id              323617
step                323617
battery_low         323617
is_charge           323617
tag_battery_low     323617
dtype: int64
```

With this code, it is observable that the row with the missing value has been successfully deleted.

Q3. There are some cases that are out of normal range due to misleading tagging sensors and errors while recording users' lifelogging. Could you find which records have to do with these cases? If it exists, explain how to handle these records with justification.

To find an out-of-range value due to an error, the domain knowledge for a normal range is required. Since the error values are not in the normal range, they can be determined by

finding out the maximum and minimum values.

```
sensor.describe(datetime_is_numeric=True)
```

	Unnamed: 0	uplink_id	owner_id	client_time	tag_id	step	battery_low	is_charge	tag_battery_low
count	323617.000000	3.236170e+05	323617.000000	323617	323617.000000	323617.000000	323617.000000	323617.000000	323617.000000
mean	162436.454037	1.938412e+06	528.838507	2020-11-03 11:26:44.112843520	3.747087	4015.824852	63.480593	0.006072	0.048783
min	0.000000	5.078200e+04	-1.000000	1970-01-01 00:00:00	0.000000	-9999.000000	0.000000	0.000000	0.000000
25%	81239.000000	1.252895e+06	501.000000	2020-09-27 16:40:09	0.000000	925.000000	43.000000	0.000000	0.000000
50%	162443.000000	1.860795e+06	530.000000	2020-11-02 15:50:10	0.000000	2755.000000	68.000000	0.000000	0.000000
75%	243623.000000	2.593409e+06	635.000000	2020-12-22 18:40:07	10.000000	5719.000000	84.000000	0.000000	0.000000
max	324822.000000	3.771903e+06	676.000000	2021-03-11 16:05:40	20.000000	65535.000000	100.000000	1.000000	1.000000
std	93761.913801	9.105514e+05	121.895119	NaN	4.743068	4209.676178	24.552630	0.077686	0.215414

By ‘sensor.describe()’, I could get minimum and maximum values for each column. I also added ‘datetime_is_numeric = True’ in the parameter to use ‘client_time’ for describe() function. I found anomalies from the minimum value of ‘owner_id’, ‘client_time’, and ‘step’. Judging from the fact that data collection began in 2019, ‘client_time’ in 1970 is an error value. ‘step’ cannot be a negative value, so the -9999 step is an error value. -1, the minimum value for ‘owner_id’, might be an out-of-range value since -1 is used as the initial value for an integer and there is no -1 in the ‘id’ column of ‘user_information.csv’. Mismeasured values cannot be used to analyze data, and there is a possibility that the other attributes of that row may have been mismeasured, so I decided to delete them.

```
sensor = sensor.drop(sensor[sensor['owner_id'] == -1].index)
sensor = sensor.drop(sensor[sensor['step'] == -9999].index)
sensor = sensor.drop(sensor[sensor['client_time'].dt.year == 1970].index)
sensor.describe(datetime_is_numeric=True)
```

	Unnamed: 0	uplink_id	owner_id	client_time	tag_id	step	battery_low	is_charge	tag_battery_low
count	323554.000000	3.235540e+05	323554.000000	323554	323554.000000	323554.000000	323554.000000	323554.000000	323554.000000
mean	162454.425098	1.938401e+06	528.854324	2020-11-05 18:29:42.092899584	3.743962	4016.201354	63.480016	0.006073	0.048783
min	55.000000	5.078200e+04	230.000000	2019-12-02 10:58:10	0.000000	5.000000	0.000000	0.000000	0.000000
25%	81266.250000	1.252892e+06	501.000000	2020-09-27 16:52:30.750000128	0.000000	925.000000	43.000000	0.000000	0.000000
50%	162462.500000	1.860831e+06	530.000000	2020-11-02 16:00:13	0.000000	2755.000000	68.000000	0.000000	0.000000
75%	243630.750000	2.593221e+06	635.000000	2020-12-22 18:42:48.500000	10.000000	5719.000000	84.000000	0.000000	0.000000
max	324822.000000	3.771903e+06	676.000000	2021-03-11 16:05:40	20.000000	65535.000000	100.000000	1.000000	1.000000
std	93749.589082	9.105519e+05	121.858253	NaN	4.738237	4208.936682	24.552347	0.077694	0.215415

By this code, we can observe the weird minimum values are deleted and in-range minimum values are there.

I am worried about the maximum value of ‘step’, because 65535 is the maximum value for 2-byte memory. However, there is a possibility that the person may have just walked a lot to exceed the maximum value of the record, so I decided to keep the data for now.

```
user = pd.read_csv('user_information.csv')

for i in user['id']:
    sensor = sensor.drop(sensor.loc[sensor['owner_id']==i].index)

sensor.count()
```

```
Unnamed: 0          35905
uplink_id           35905
owner_id            35905
client_time         35905
tag_id              35905
step                35905
battery_low         35905
is_charge           35905
tag_battery_low     35905
dtype: int64
```

Next, I used the code above to see if there is data for people who do not exist in 'user_information.csv'. As shown in the picture, there were 35905 unrecorded user data. Using data from people who do not exist in the company's database reduces the reliability of the collected data. In addition, it is difficult to analyze the correlation between a user's individual characteristics and the user's tagging information because the company does not know the user's information. Therefore, I decided to delete the data for these unclear users as well.

```
sensor.to_csv('temp.csv')
temp = pd.read_csv('temp.csv')
temp.groupby('owner_id').count()
```

owner_id	Unnamed: 0	Unnamed: 0.1	...
478	321	321	
502	1167	1167	
506	7796	7796	
514	4530	4530	
524	9559	9559	
543	717	717	
545	13	13	
551	1554	1554	
645	238	238	
669	5292	5292	
670	4713	4713	
676	5	5	

Code 1

```
unknown_user = [478,502,506,514,524,543,545,551,645,669,670,676]
for i in unknown_user:
    sensor = sensor.drop(sensor.loc[sensor['owner_id']==i].index)
sensor.describe()
```

	Unnamed: 0	uplink_id	owner_id	tag_id	step
count	287649.000000	2.876490e+05	287649.000000	287649.000000	287649.000000
mean	168028.857500	1.991220e+06	524.889466	3.790776	4174.117678
std	94949.352288	9.294058e+05	126.335185	4.756945	4354.625585
min	55.000000	5.078200e+04	230.000000	0.000000	5.000000
25%	86361.000000	1.291852e+06	499.000000	0.000000	951.000000
50%	171205.000000	1.932387e+06	536.000000	0.000000	2844.000000
75%	251496.000000	2.691237e+06	635.000000	10.000000	6001.000000
max	324822.000000	3.771903e+06	674.000000	20.000000	65535.000000

Code 2

I used code 1 to find the ID for unknown users present in 'upink_device.csv' and

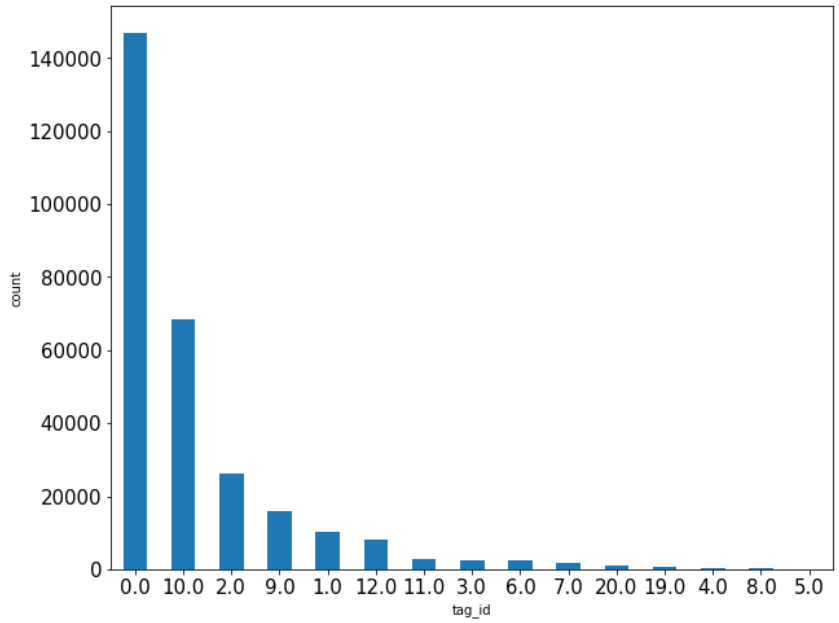
successfully deleted their data with code 2.

Q4. Tagging information is the most basic attribute that can express a user's daily life. However, some data may be overlapped with others or be unnecessary (out of range) to predict user activities. How are you going to select the necessary variable over the data?

What does the company ultimately want to do with this analysis? The company will want to use this data analysis result to apply it to its IoT devices. To do so, the most important variable for the company would be 'tag_id'.

```
tag_category = pd.value_counts(sensor['tag_id'].values)
tag_category.plot.bar(xlabel='tag_id',ylabel='count',rot=0, figsize=(10,8), fontsize=15)  # figsize, fontsize to adjust chart size
```

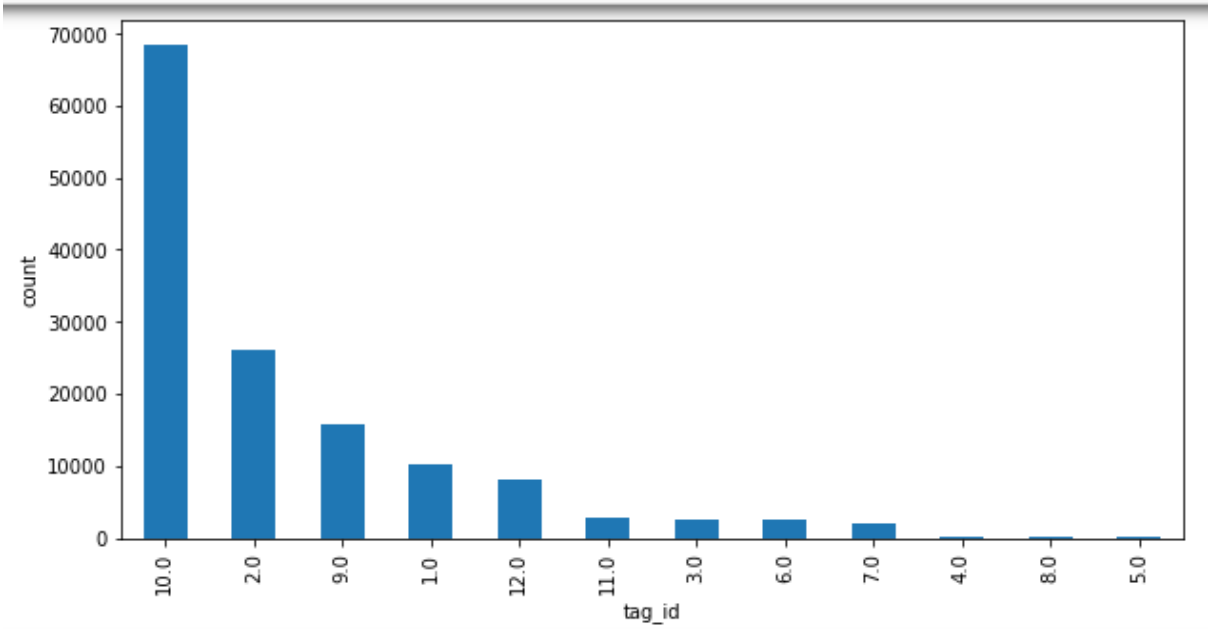
```
<AxesSubplot:xlabel='tag_id', ylabel='count'>
```



This bar chart represents the distribution value of 'tag_id' category. Tag 0 means that there is no tag information for the data. Although tag 0 has the largest portion of the information, it goes against the company's purpose to analyze the user behaviors through the tags. Since the amount of information on tag 0 is large and the proportion of other information is relatively small, it is difficult to analyze the users' behavior pattern, so it seems appropriate to delete data with tag 0. For the same reason, I deleted the tag 19 and tag 20.

```
sensor = sensor.drop(sensor[sensor['tag_id'] == 0].index)

sensor = sensor.drop(sensor[sensor['tag_id'] == 19].index)
sensor = sensor.drop(sensor[sensor['tag_id'] == 20].index)
tag_category = pd.value_counts(sensor['tag_id'].values)
tag_category.plot.bar(xlabel='tag_id',ylabel='count',figsize=(10,5))
```



We can see the unnecessary tags have been properly deleted. Number of the variables of 'tag_id' decreased, and therefore the number of column will be decreased since it will be converted to dummy variables.

Q5. Using the Inputs from multiple tags as-is can lead to the input dimension becoming too large. This can cause the models to become too complex and may hinder in extracting the information you want. How can we reduce the dimension of the input? What benefits are there in reducing the input dimension?

If the dataset has too many dimensions, many problems may occur. For example, there can be an increase in resources required for calculation and a decrease in data accuracy due to interference with unnecessary variables.

	Unnamed: 0	uplink_id	owner_id	client_time	tag_id	step	battery_low	is_charge	tag_battery_low
count	140655.000000	1.406550e+05	140655.000000	140655	140655.000000	140655.000000	140655.000000	140655.000000	140655.000000
mean	169461.723501	2.014562e+06	525.965668	2020-11-11 13:58:22.258817792	7.752394	4131.678014	63.593672	0.002275	0.100430
min	55.000000	5.078200e+04	230.000000	2019-12-02 10:58:10	1.000000	7.000000	0.000000	0.000000	0.000000
25%	89015.500000	1.312860e+06	499.000000	2020-09-30 16:06:48	2.000000	1052.000000	44.000000	0.000000	0.000000
50%	169394.000000	1.919977e+06	536.000000	2020-11-06 05:15:00	10.000000	2894.000000	68.000000	0.000000	0.000000
75%	253787.000000	2.719805e+06	644.000000	2020-12-31 19:04:29.500000	10.000000	5922.000000	84.000000	0.000000	0.000000
max	324822.000000	3.771903e+06	674.000000	2021-03-11 16:05:40	20.000000	65535.000000	100.000000	1.000000	1.000000
std	95221.235011	9.369441e+05	129.086448	NaN	3.945223	4119.669455	24.704737	0.047644	0.300574

As you can see from this table, we have 9 columns. The first column labeled with 'Unnamed:0' is the index column so we can delete it. To do further analysis, I made a correlation matrix.

```
sensor['client_time'] = pd.to_numeric(sensor.client_time)
sensor.corr()
```

	uplink_id	owner_id	client_time	tag_id	step	battery_low	is_charge	tag_battery_low
uplink_id	1.000000	0.117412	0.970883	0.057058	0.019353	0.123343	-0.011202	0.085663
owner_id	0.117412	1.000000	0.200293	0.047492	-0.051734	0.092102	-0.004606	-0.068054
client_time	0.970883	0.200293	1.000000	0.069874	-0.012671	0.126678	-0.008749	0.062848
tag_id	0.057058	0.047492	0.069874	1.000000	-0.019852	0.014968	-0.011301	-0.096769
step	0.019353	-0.051734	-0.012671	-0.019852	1.000000	-0.096345	0.000652	-0.025188
battery_low	0.123343	0.092102	0.126678	0.014968	-0.096345	1.000000	-0.003606	-0.034402
is_charge	-0.011202	-0.004606	-0.008749	-0.011301	0.000652	-0.003606	1.000000	0.006882
tag_battery_low	0.085663	-0.068054	0.062848	-0.096769	-0.025188	-0.034402	0.006882	1.000000

With this correlation matrix, we can see 'client_time' and 'uplink_id' have a very strong relationship with each other. Since time has much more information than the uplink id, I decided to delete 'uplink_id'.

'client_time' can be reduced by using the 'set_index' function, but I didn't convert datetime to the index for easier data visualization. I also thought to convert 'tag_id' and 'owner_id' to dummy variables and reduce the columns using pivot tables. But 'tag_id' itself cannot be reduced since tagging information would be the most important information to build an IoT home automation system. The ability to control the state of the house according to the tag information is essential in the IoT environment. And each 'owner_id' has individual characteristics like medicine history, medical records, and other information. So, I decided to keep them.

```
sensor.describe(datetime_is_numeric=True)
```

	owner_id	client_time	tag_id	step	battery_low	is_charge	tag_battery_low
count	138928.000000	138928	138928.000000	138928.000000	138928.000000	138928.000000	138928.000000
mean	526.046139	2020-11-11 15:13:48.529396736	7.605220	4128.771054	63.588866	0.002239	0.101160
min	230.000000	2019-12-02 10:58:10	1.000000	7.000000	0.000000	0.000000	0.000000
25%	499.000000	2020-09-30 16:15:05.500000	2.000000	1053.000000	44.000000	0.000000	0.000000
50%	536.000000	2020-11-05 22:02:26.500000	10.000000	2894.000000	68.000000	0.000000	0.000000
75%	644.000000	2020-12-31 18:16:19.500000	10.000000	5919.000000	84.000000	0.000000	0.000000
max	674.000000	2021-03-11 16:05:40	12.000000	65535.000000	100.000000	1.000000	1.000000
std	129.053386	NaN	3.740474	4113.212989	24.708683	0.047261	0.301542

Final attributes of dataset

We can make ‘tag_id’ and ‘owner_id’ to dummy variables before making models.

Q6. Visualize the records corresponding to users’ daily life from the preprocessed data. How could tagging information correspond to the expected daily routine of the user?

```
sensor['hour'] = sensor['client_time'].dt.hour
df_1 = sensor.query("tag_id == 1")
tag1_sum = df_1.groupby(df_1['hour'])['tag_id'].count()

df_2 = sensor.query("tag_id == 2")
tag2_sum = df_2.groupby(df_2['hour'])['tag_id'].count()

df_3 = sensor.query("tag_id == 3")
tag3_sum = df_3.groupby(df_3['hour'])['tag_id'].count()

df_4 = sensor.query("tag_id == 4")
tag4_sum = df_4.groupby(df_4['hour'])['tag_id'].count()

df_5 = sensor.query("tag_id == 5")
tag5_sum = df_5.groupby(df_5['hour'])['tag_id'].count()

df_6 = sensor.query("tag_id == 6")
tag6_sum = df_6.groupby(df_6['hour'])['tag_id'].count()

df_7 = sensor.query("tag_id == 7")
tag7_sum = df_7.groupby(df_7['hour'])['tag_id'].count()

df_8 = sensor.query("tag_id == 8")
tag8_sum = df_8.groupby(df_8['hour'])['tag_id'].count()

df_9 = sensor.query("tag_id == 9")
tag9_sum = df_9.groupby(df_9['hour'])['tag_id'].count()

df_10 = sensor.query("tag_id == 10")
tag10_sum = df_10.groupby(df_10['hour'])['tag_id'].count()

df_11 = sensor.query("tag_id == 11")
tag11_sum = df_11.groupby(df_11['hour'])['tag_id'].count()

df_12 = sensor.query("tag_id == 12")
tag12_sum = df_12.groupby(df_12['hour'])['tag_id'].count()
```

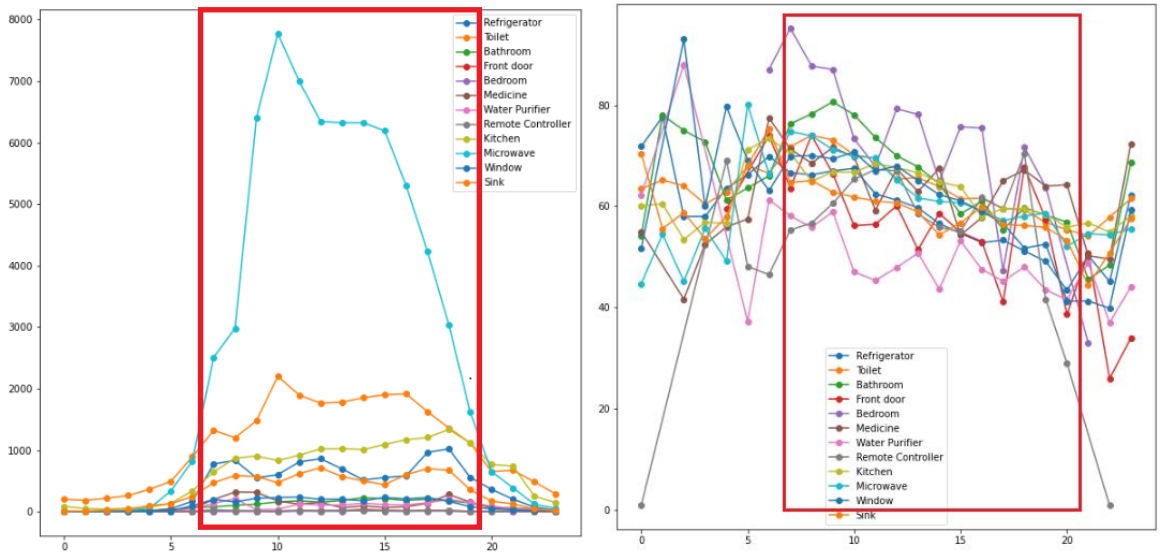
```
fig= plt.figure(figsize=(10,10))
fig.set_facecolor('white')
ax = fig.add_subplot()

ax.plot(tag1_sum,marker='o',label='Refrigerator')
ax.plot(tag2_sum,marker='o',label='Toilet')
ax.plot(tag3_sum,marker='o',label='Bathroom')
ax.plot(tag4_sum,marker='o',label='Front door')
ax.plot(tag5_sum,marker='o',label='Bedroom')
ax.plot(tag6_sum,marker='o',label='Medicine')
ax.plot(tag7_sum,marker='o',label='Water Purifier')
ax.plot(tag8_sum,marker='o',label='Remote Controller')
ax.plot(tag9_sum,marker='o',label='Kitchen')
ax.plot(tag10_sum,marker='o',label='Microwave')
ax.plot(tag11_sum,marker='o',label='Window')
ax.plot(tag12_sum,marker='o',label='Sink')


ax.legend()
```

The following code was used to understand the behavioral patterns of the aggregated data. I add the hour column to categorize data by hours. But I keep the ‘client_time’ just in case. By the code on the left, the number of times each tag was used was counted and organized. And through the code on the right, each tag was labeled and graphed with 'hour' on

the x-axis and 'count' on the y-axis. I also used the same method to create another graph where the y-axis is the average of the battery level.

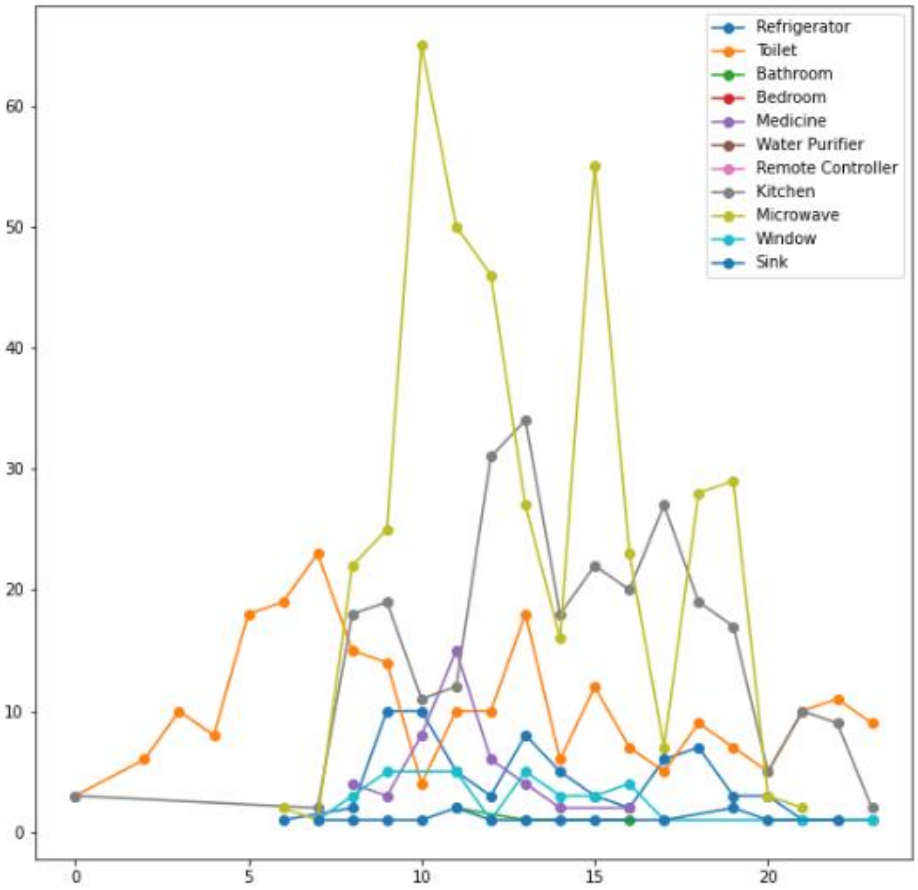


Y=count of 'tag_id' Y=mean of 'battry_low'

In most situations, the number of times the tag is used increases between 6 and 7 in the morning and starts to decrease around 8 in the evening. The device's battery also starts to drain at 7 am and recharges after 8 pm. From this result, it can be confirmed that most people start their activities at 7 am and finish their activities at 8 pm. In addition, the most tagged device is a microwave, and it can be hypothesized that elderly people frequently consume food using a microwave oven. It can be understood that the protruding part of the graph of the microwave oven and refrigerator means the participants' general mealtime.

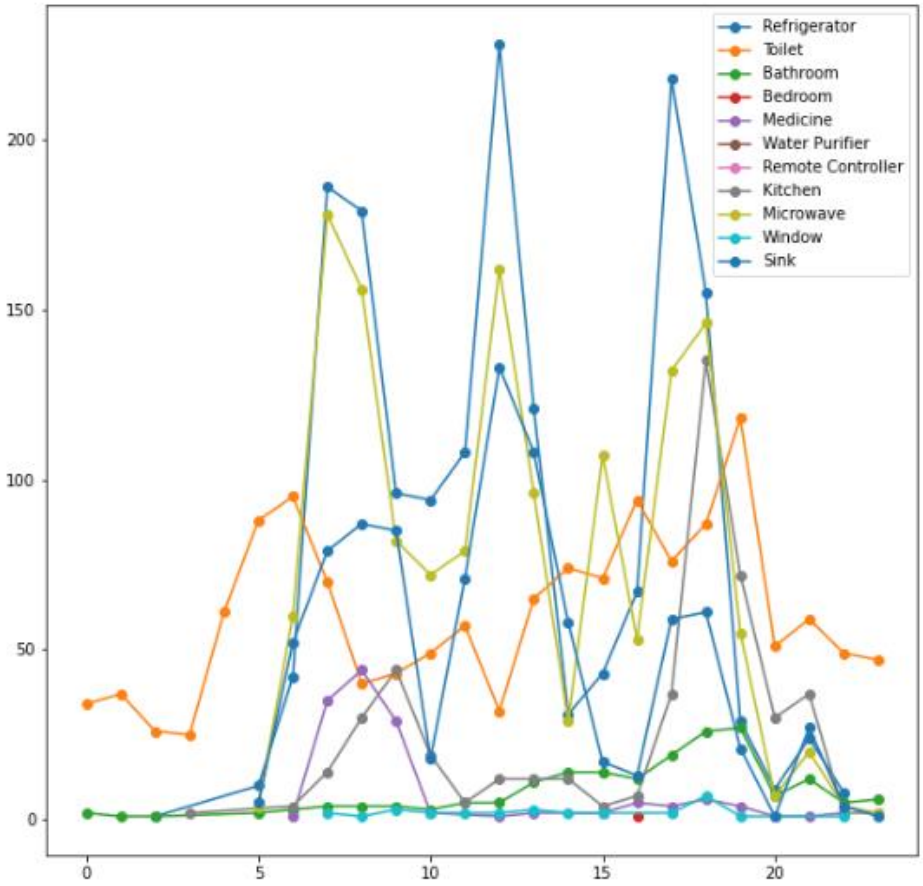
Q7. Visualize lifelogging patterns for each user from the preprocessed data. Do you agree that it is possible to make a new service based on various patterns you discover over data? If possible do you need any additional information to develop your idea?

IoT services are a growing business that is being used in many places recently. The biggest advantage of this IoT service is that it provides a service optimized for users by analyzing individual patterns. I will explain how it can be applied to individuals with two examples. I re-used the method I used in Q6 because I thought that the frequency of tags over time was the most representative of the user's pattern.



519

This is the individual data for participant 519. Looking at this data, the most visible data is the peaks of the microwave oven. This participant is more likely to eat at 10 a.m., 3 p.m., and 8 p.m. According to 'user_information.csv', participant 519 is taking medication. And the graph shows that the participant often takes the medicine at 11 o'clock. Using this information, it becomes possible to send an alarm to the user's device in time for the user's meal or medicine. In addition, when looking at this user's toilet usage record, the company can infer that the user cannot sleep deeply and recommend appropriate countermeasures to the user through this information.



635

The second one is the individual data for participant 635. The number of tags on this user's refrigerator, microwave, sink, and kitchen indicates that she is a person who likes to cook. Then, the company will be able to provide support such as an automatic ventilating system according to the time the user frequently cooks. Additionally, this user has dramatically reduced activity between 10 pm and 5 am. During this time, the company can turn off the lights in the whole house and close the gas valve for the safety.

In addition to this, if the change in the number of steps and tagging information does not change for a long time, the company will be able to find out the user's problem more quickly. For easier and more support, the company need more accurate user information. The current 'tag_id' and 'step' are limited in knowing exactly what the user is doing. So, in addition to the current system, I think it would be good to get the user's GPS information. Currently, most smartphones collect GPS information, so it is easy to collect information, and this method has the potential to support users not only indoors but also outdoors.