1. (30pt) Propose the contents of the life pattern report. Examining the examples of the report as shown at the end of the document, try to design the format and contents of the report for a specific user for a month to improve his/her well-being.

1. Data Loading

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(rc={'figure.figsize':(11.7,8.27)})
%matplotlib inline
```

In [2]:
```
### userinformation
```

In [3]:
```
userinfo = pd.read_csv('user_information.csv')
userinfo_505 = userinfo.loc[userinfo.user_id == 505,:]
userinfo_505
```

Out[3]:

	user_id	birth year	age	sex	etc	depression_score	depression_class
20	505	1946	76	F	Not applicable	0.5	Moderate

In [4]:
```
### annotation file
```

In [5]:
```
data_505 = pd.read_csv('505.csv')
```

In [6]:
```
data_505['start'] = pd.to_datetime(data_505['start'])
data_505['end'] = pd.to_datetime(data_505['end'])
```

In [7]:
```
data_505.dtypes
```

Out[7]:
```
start        datetime64[ns]
end          datetime64[ns]
labelName            object
dtype: object
```

In [26]:
```
data_505.head()
```

Out[26]:

	start	end	labelName
4795	2021-03-11 14:30:00	2021-03-11 15:00:00	Go to bathroom
4796	2021-03-11 15:00:00	2021-03-11 15:30:00	Cook
4797	2021-03-11 15:30:00	2021-03-11 16:00:00	Go to bathroom
4798	2021-03-11 16:00:00	2021-03-11 16:30:00	Cook
4799	2021-03-11 16:00:00	2021-03-11 16:30:00	Cook

In [9]:
```
data_505['labelName'].value_counts()
```

Out[9]:
```
Go to bathroom    1516
Cook              1243
Wash dishes        643
Eat                532
Sleep              374
Take shower        216
Wake up            197
Take medicine       61
Go walk             17
Watch TV             1
Name: labelName, dtype: int64
```

In [10]:
```
### device uplink
```

In [11]:
```
device = pd.read_csv('device_uplink.csv')
device = device.drop(['Unnamed: 0'], axis=1)
device = device.dropna()
device = device.loc[device['client_time']>device['client_time'].min()]
device = device.loc[device['step']>device['step'].min()]
device = device.loc[device['owner_id']>0]
index = device[device['tag_id']==0].index
device.drop(index, inplace=True)
index = device[device['tag_id']>17].index
device.drop(index, inplace = True)
```

In [12]:
```
device_505 = device.loc[device.owner_id == 505,:].reset_index()
device_505
```

Out[12]:

	index	uplink_id	owner_id	client_time	tag_id	step	battery_low	is_charge	tag_battery_low
0	16439	543864	505	2020-08-07 18:06:39	2.0	10	9	0	0
1	16441	543882	505	2020-08-07 18:08:51	2.0	57	7	0	0
2	16443	543890	505	2020-08-07 18:09:42	2.0	72	5	0	0
3	16558	545631	505	2020-08-07 20:22:21	9.0	97	87	0	0
4	16925	551726	505	2020-08-08 10:38:31	10.0	62	57	0	0
...	...	...	...	...	...	...	...	...	...
10077	324798	3771783	505	2021-03-11 15:55:13	10.0	6954	36	0	0
10078	324799	3771784	505	2021-03-11 15:55:22	10.0	6967	35	0	0
10079	324801	3771787	505	2021-03-11 15:55:32	10.0	6969	35	0	0
10080	324811	3771847	505	2021-03-11 15:58:53	9.0	7081	35	0	0
10081	324822	3771903	505	2021-03-11 16:05:40	9.0	7177	35	0	0

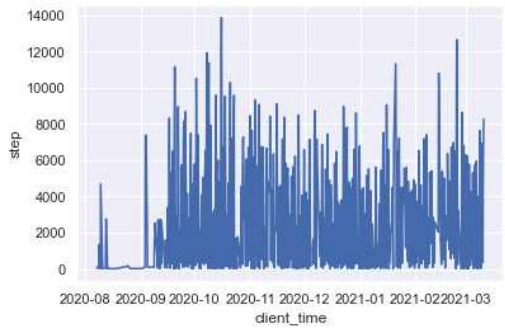
10082 rows × 9 columns

2.Visualize

2-1. Analysis of step pattern

In [16]:
```
sns.lineplot(
    data=device_year,
    x="client_time", y="step"
)
```

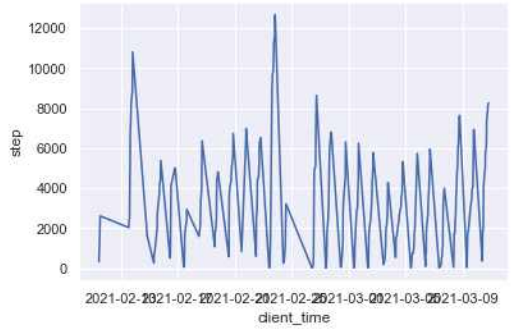
Out[16]:
```
<AxesSubplot:xlabel='client_time', ylabel='step'>
```



This is the result of analyzing the step pattern for 1 year. It can be seen that data is concentrated under 10000 steps.

```
In [19]: sns.lineplot(
             data=device_month,
             x="client_time", y="step"
         )
```

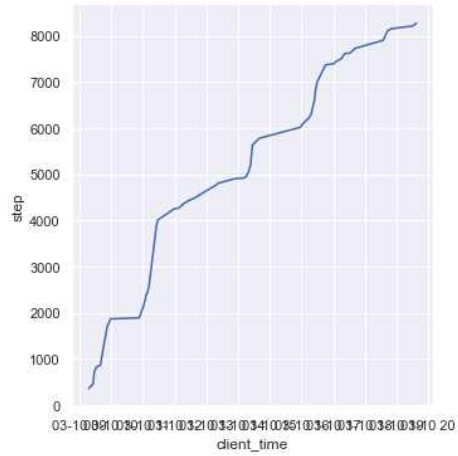
```
Out[19]: <AxesSubplot:xlabel='client_time', ylabel='step'>
```



This is the result of analyzing the step pattern for 1 month. It can be seen that the deviation of the number of steps in a day is large.

```
In [25]: sns.relplot(
             data=device_day, kind="line",
             x="client_time", y="step"
         )
```

```
Out[25]: <seaborn.axisgrid.FacetGrid at 0x7f7ff7d47490>
```



This is the result of analyzing the step pattern for 1 day. It can be seen that the step increase rate in the daytime is greater than the step increase rate in the evening.

If the number of steps today is above the average, a message of compliment is delivered.

In [23]:
```
device_day_avg = device[(device['client_time']>'2021-03-10')&(device['client_time']<'2021-03-11
if (device_day['step'].mean() >= device_day_avg) :
    print("You walked more than other users!")
else :
    print("You walked less than other users")
```

```
You walked less than other users
```

2-2 Daily noticification of chores to do today

In [24]:
```
# 12: sink, 16: vacuum cleaner, 17: washer
import numpy as np
checklist = np.full(3, False, dtype=bool)

for i in range(len(device_day)):
    if(np.any(device_day.tag_id == 12.0)):
        checklist[0] = True;
    elif(np.any(device_day.tag_id == 16.0)):
        checklist[1] = True;
    elif(np.any(device_day.tag_id == 17.0)):
        checklist[2] = True;
```

In [25]:
```
print("[A list of chores to do today]")

print("washing dishes : ")
if(checklist[0] == False):
    print("- You did it!")
else : print("- You need to do it.")

print("vacuum cleaner : ")
if(checklist[1] == False):
    print("- You did it!")
else : print("- You need to do it.")

print("washing : ")
if(checklist[2] == False):
    print("- You did it!")
else : print("- You need to do it.")
```

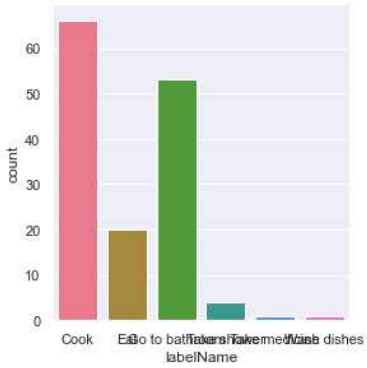
```
[A list of chores to do today]
washing dishes :
- You need to do it.
vacuum cleaner :
- You did it!
washing :
- You did it!
```

It helps you manage the housework that you have to do for a day easily through a checklist. In the case of this user, it can be seen that other household chores have been completed, while washing dishes have not done yet.

2-3 Montly review of your lifestyle

In [45]:
```
ts = pd.Timestamp
data_sample = pd.DataFrame(index=range(0,len(data_505)), columns = {'hour', 'day', 'month', 'ye
for i in range(len(data_sample)):
    data_sample['hour'][i] = ts(data_505['start'][i]).hour
    data_sample['day'][i] = ts(data_505['start'][i]).day
    data_sample['month'][i] = ts(data_505['start'][i]).month
    data_sample['year'][i] = ts(data_505['start'][i]).year
    data_sample['labelName'][i] = data_505['labelName'][i]
```

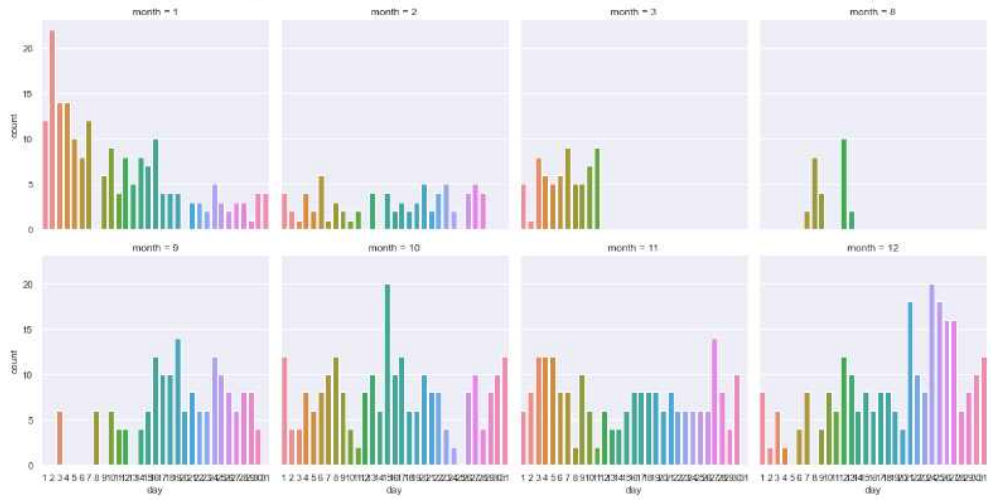
In [46]:
```
# What kind of activities did you do the most during the month?
sns.catplot(x="labelName",
                data=data_sample.loc[data_sample.month==3,:], kind="count",
                height=4, palette="husl");
```



It provides information on what activities you have done a lot in a month. In the case of this user, it was found that he cooked the most.

2-4. Anlysis of your cooking pattern

In [47]:
```
#How many times did you cook last month?
sns.catplot(x="day", col="month", col_wrap=4,
                data=data_sample.loc[data_sample.labelName=='Cook',:], kind="count",
                height=4);
```



It provides detailed information on the user's cooking pattern. By synthesizing previous past data, you can see how much you cooked at a glance. In the case of

this user, he started cooking a lot again in September.

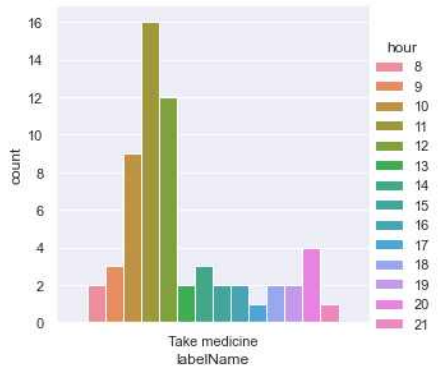
If you have cooked more than last month, provide a message of praise, or give a message of encouragement.

```
In [29]: #Compare the number of times you cooked this month
         #and the number of times you cooked last month

         cook_thismonth = len(data_sample.loc[(data_sample.labelName=='Cook') & (data_sample.year==2021)
         cook_lastmonth = len(data_sample.loc[(data_sample.labelName=='Cook') & (data_sample.year==2021)

         if(cook_thismonth > cook_lastmonth) :
             print("You cooked more than last month")
         else :
             print("You cooked less than last month")
```

```
You cooked less than last month
```

2-5. Analysis of Time to take medicine

```
In [50]: # Time to take medicine (1 year)
         sns.catplot(x="labelName", hue="hour",
                     data=data_sample.loc[data_sample.labelName=='Take medicine',:], kind="count",
                     height=4);
```



Summarizing the data for a year, it shows what time the user takes the medicine. In the case of this user, it can be seen that he often takes medicine from 10 to 12 o'clock.

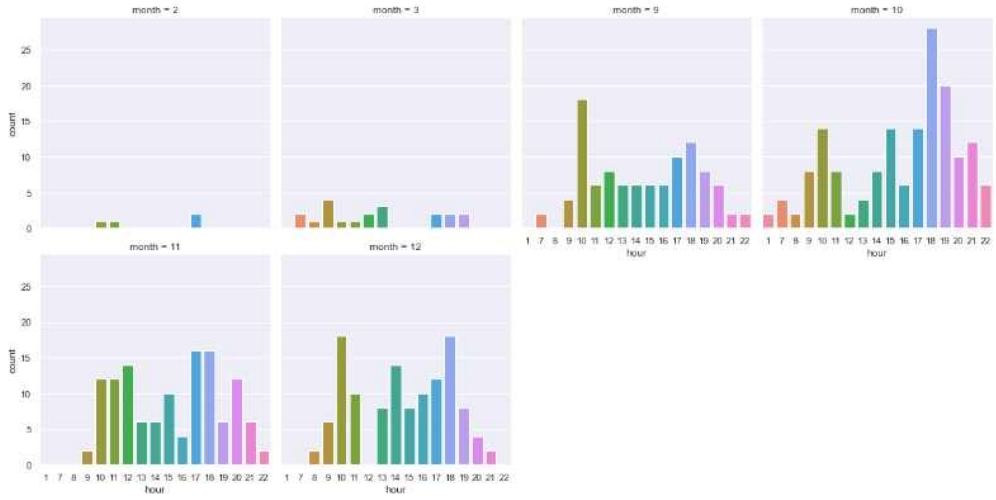
In the case of a user suffering from a disease, an alarm is provided after checking whether the user is taking medicine. (Daily noticification of taking medicine)

In [30]:
```
# 6:Medicine
# If the monthly data shows that the user took the medicine,
# we suppose the user takes the medicine.
medicine_checklist = np.full(1, False, dtype=bool)
if(any(device_month.tag_id == 6)):
    if(any(device_day.tag_id == 6)):
        medicine_checklist[0] = True;

print("[Taking Medicine]")
if(medicine_checklist[0] == False):
    print("- You need to take medicine")
else :
    print("- You already took it!")
```

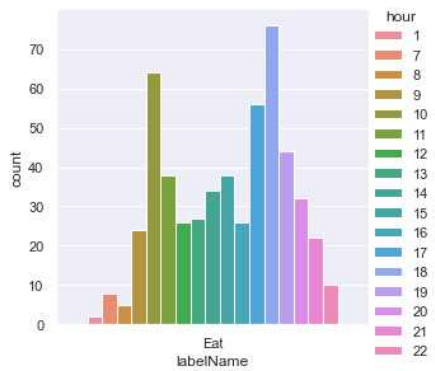
```
[Taking Medicine]
- You already took it!
```

2-6. Analysis of Time to Eat food

In [31]:
```
#Past data of the time when you eat food most often
sns.catplot(x="hour", col="month", col_wrap=4,
                data=data_sample.loc[data_sample.labelName=='Eat',:], kind="count",
                height=4);
```



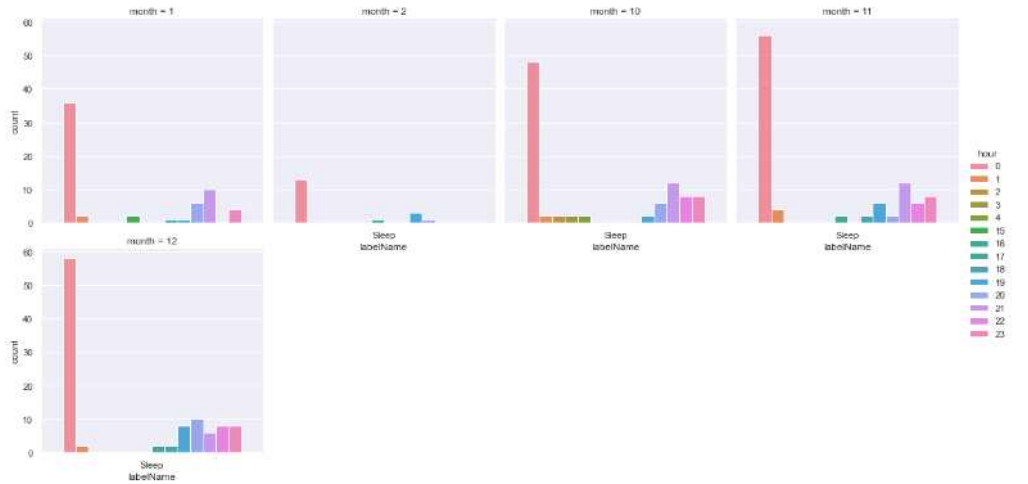
In [32]:
```
#You can check the time when you eat food most often and when you don't.
sns.catplot(x="labelName", hue="hour",
                data=data_sample.loc[data_sample.labelName=='Eat',:], kind="count",
                height=4);
```



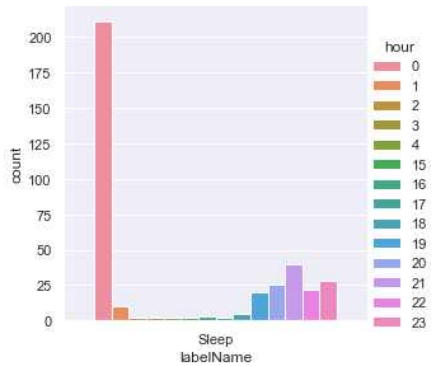
This shows what time the user eats. In the case of this user, it can be seen that they consume the most food at 10 o'clock and 18 o'clock.

2-7. Analysis of sleep pattern

```
In [33]: sns.catplot(x="labelName", col="month", col_wrap=4, hue="hour",
                     data=data_sample.loc[data_sample.labelName=='Sleep',:], kind="count",
                     height=4);
```




```
In [34]: #total
sns.catplot(x="labelName", hue="hour",
                     data=data_sample.loc[data_sample.labelName=='Sleep',:], kind="count",
                     height=4);
```



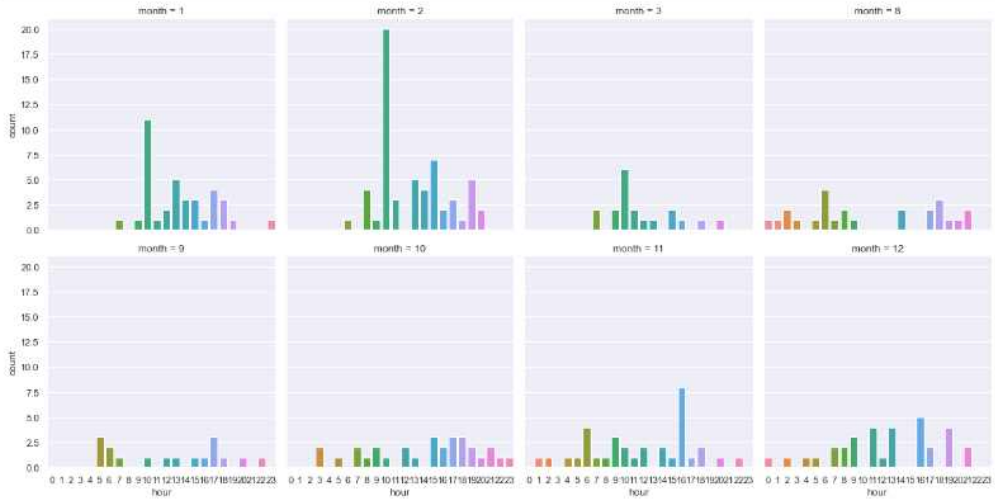
It shows what time the user sleeps. In the case of this user, it can be seen that he usually sleeps at 12 o'clock.

An alarm for a sleep time is provided using the mode of the user's sleep time.

```
In [38]: sleep_mode = data_sample.loc[data_sample.labelName=='Sleep',:]['hour'].mode()[0]
         sleep_today = data_sample.loc[data_sample.labelName=='Sleep',:].iloc[-1,:]['hour']

         if(20 < sleep_today < sleep_mode+24):
             print("You slept early")
         elif(sleep_mode < sleep_today < 6):
             print("You slept late")

         You slept early
```

2-8. Analysis of bathroom pattern

```
In [293]: sns.catplot(x="hour", col="month", col_wrap=4,
                      data=data_sample.loc[data_sample.labelName=='Go to Bathroom',:], kind="count",
                      height=4);
```



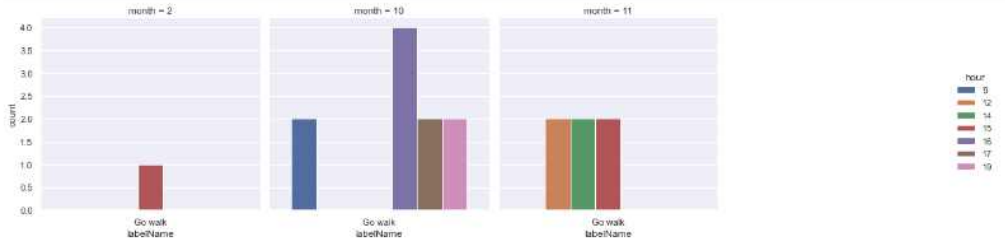
It shows what time the user usually goes to the bathroom a lot.

After calculating the average number of toilet visits by the user, an alarm message is provided if the number of toilet visits today is less than the average.

```
In [39]: bathroom_lastmonth = len(data_sample.loc[(data_sample.labelName=='Go to Bathroom') & (data_samp
         bathroom_today = len(data_sample.loc[(data_sample.labelName=='Go to Bathroom') & (data_sample.y

         if(bathroom_today > bathroom_lastmonth) :
             print("You're going to the bathroom well!")
         else :
             print("You need to go to bathroom more often.")

         You need to go to bathroom more often.
```

2-9. Analysis of Exercise pattern

```
In [81]: sns.catplot(x="labelName", col="month", col_wrap=4, hue="hour",
                     data=data_sample.loc[(data_sample.labelName=='Go walk'),:], kind="count",
                     height=4);
```



It shows the time zone and frequency of exercise at a glance. In the case of this user, he did not go a walk regularly.

We can provide advisory messages of exercise based on whether the user has worked out or not.

```
In [83]: workout_lastmonth = len(data_sample.loc[(data_sample.labelName=='Go walk') & (data_sample.year=
         workout_today = len(data_sample.loc[(data_sample.labelName=='Go walk') & (data_sample.year==202

         if(workout_today > workout_lastmonth) :
             print("You did a good job!")
         else :
             print("You need to go out and exercise more!")
```

```
You need to go out and exercise!
```

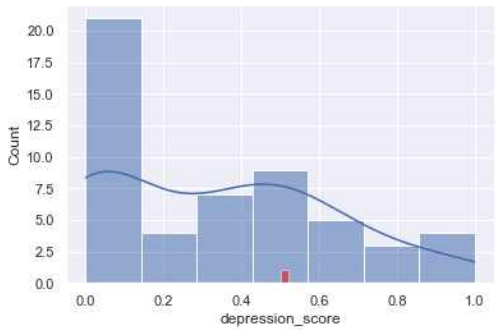
2-10. Analysis of Mental health

```
In [84]: userinfo_505
```

Out[84]:

	user_id	birth year	age	sex	etc	depression_score	depression_class
20	505	1946	76	F	Not applicable	0.5	Moderate

```
In [85]: sns.histplot(x="depression_score",
                      data=userinfo.loc[:,['depression_score']], kde=True)
         # The user's depression level is red line
         plt.hist(userinfo_505.iloc[0].depression_score, bins=50, alpha=0.5, color='red')
         plt.show()
```



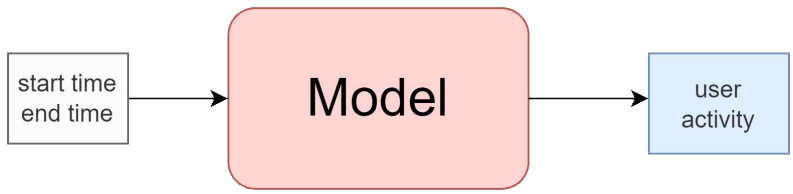
On a graph showing the depression score of all users, the depression score of the current user is shown. It can be seen that the depression score of many users is 0(Normal).

If the depression score is higher or lower than the overall user average, a message is provided accordingly.

```
In [87]: # 사용자 전체 평균
depression_avg = userinfo.loc[:,['depression_score']].mean()[0]
if( userinfo_505.iloc[0].depression_score < depression_avg):
    print("Don't be depressed. You're doing great job!")
else:
    print("You have a healthier mind than other users.")

You have a healthier mind than other users.
```

2. (30pt) Decide the appropriate data mining models for filling out the information at the report. Depending on the life patterns that you want to provide, several different models should be chosen and optimized with different hyperparameters.

We designed a model to be used predict users's life patter to write a report, using the user's activity name as input of the model and the user's start and end times as target.



And we're going to make a predictive model using artificial intelligence, and we're going to make a model using three techniques. Linear Regression, Neural Net, and RandomForests.

Before implementing the model, time was extracted from data in the start and end columns using to_datetime funcion and datetime.time.

```
#데이터 불러오기
user1= pd.read_csv ('505b.csv')
user1 ['start'] = pd.to_datetime (user1 ['start'])
user1 ['end'] = pd.to_datetime (user1 ['end'])
import  datetime as  d
for  g in  range (len (user1 )):
    time = d.datetime.time (user1 ['start'].iloc [g ])
    user1 ['start'].iloc [g ] = time
    time = d.datetime.time (user1 ['end'].iloc [g ])
```

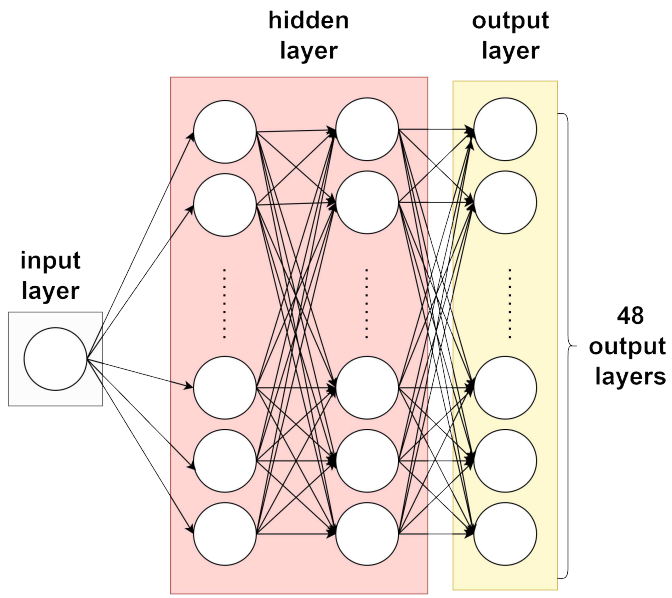
```
    user1 ['end'].iloc [g ] = time
```

First, I use Linear regression to make data mining model. To implement this model, we used the linear regression library provided by sklearn. Below is the code that I implemented the model.

```
from  sklearn.model_selection import  train_test_split
from  sklearn.linear_model import  LinearRegression
lr = LinearRegression ()
train_target = user1 [['start','end']].astype ('category')
train_input = user1 ['labelName'].astype ('category')
train_input = pd.get_dummies (train_input )
train_target = pd.get_dummies (train_target )
train_scaled ,val_scaled ,train_target ,val_target = train_test_split
(train_input , train_target ,test_size=0.2 , random_state=42 )
lr.fit (train_scaled ,train_target )
pred=lr.predict (train_scaled )
```

Label name was given as input and start time and end time was given as output. And so that LinearRegression function can not calculate datetime.time for, we changed the time of datetime to category form. The label name was also changed to category. The get_dummies function was then used to change category type to binary value which LinearRegression function can calculate.

Second, in Neural Net, we try to implement the model in the form of output that the time divided into 48 classes when we put the activity in the artificial neural network as input.



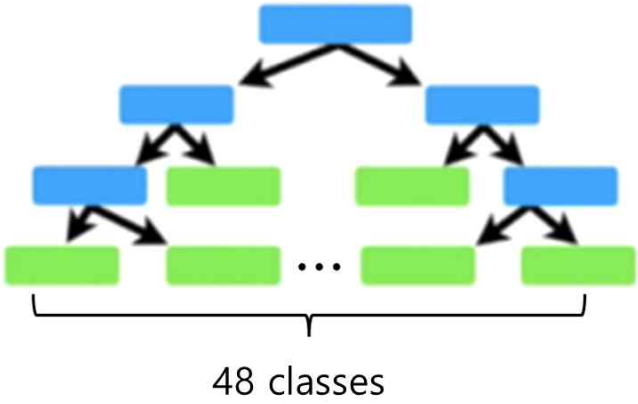
To implement this model, I use the MLPClassifier function which is provided by sklearn.neural_network. Below is the code that how I implemented the model.

```
#Neural Net
#데이터 불러오기
import  matplotlib.pyplot as  plt
#라이브러리 임포트
from  tensorflow import  keras
#테스트세트와 검증세트 나누기
from  sklearn.model_selection import  train_test_split
from  sklearn.neural_network import  MLPClassifier
train_target = user1 [['start','end']].astype ('category')
train_input = user1 ['labelName'].astype ('category')
train_input = pd.get_dummies (train_input )
train_target = pd.get_dummies (train_target )
train_scaled ,val_scaled ,train_target ,val_target = train_test_split
(train_input , train_target ,test_size=0.2 , random_state=42 )
#신경망 만들기
print (train_scaled.shape )
model = MLPClassifier
(hidden_layer_sizes=(3),activation='logistic',solver='lbfgs',random_state
=1 )
model.fit (train_scaled ,train_target)
```

Same as LinearRegression, label name was given as input and start time and end time was given as target. And so that MLPClassifier function can not calculate datetime type, we changed train_target's datetime.time form to category form and make it dummy variable by get_dummies function.

When designing the MLPCassifier model, I use logistice function as activation function.

Last, Random Forest has high accuracy, and trains are simple and fast. In addition, it is possible to deal with thousands of input variables without erasing variables, and because it produces good generalization performance through randomization, we try to select a random forest and make a model. We will design a model to classify into 48 time classes.



To implement this model, I use the RandomForestClassifier function which is provided by sklearn.ensemble. Below is the code that how I implemented the model.

```
#Random Forest
import  numpy as  np
import  pandas as  pd
from  sklearn.model_selection import  train_test_split
train_target = user1 [['start','end']].astype ('category')
train_input = user1 ['labelName'].astype ('category')
train_input = pd.get_dummies (train_input )
train_target = pd.get_dummies (train_target )
train_scaled ,val_scaled ,train_target ,val_target = train_test_split
(train_input , train_target ,test_size=0.2 , random_state=42 )
#랜덤포레스트사용하기
from  sklearn.model_selection import  cross_validate
from  sklearn.ensemble import  RandomForestClassifier
rf = RandomForestClassifier (n_jobs = -1 , random_state=42)#모든코어사용
#검증세트점수와 훈련세트점수반환
scores = cross_validate (rf ,train_scaled ,train_target
,return_train_score=True ,n_jobs=-1 ) #교차검증사용
print (np.mean (scores ['train_score']),np.mean (scores ['test_score']))
#랜덤포레스트모델 훈련
rf.fit (train_scaled ,train_target)
```

Same as LinearRegression and neural net, label name was given as input and start time and end time was given as target. We changed train_target's datetime.time form to category form and change it dummy variable by using get_dummies function, because RandomForestClassifier function can not calculate datetime type. And split the validation set and the training set at a ratio of 2 to 8.

And we use cross validation function to create a better working model for new data and to prevent overfitting.

3. Evaluate the models with the validation data and demonstrate the usefulness of the data mining process to complete the service of summarizing the life patterns.

As mentioned in the proposal, we split the user's log information data into 60% of training data and 40% of validation data. We decided to use 3 methods: linear regression, neural net, and random forest. After constructing the models with the training data, we evaluated each model with the validation data.

```
train_scaled ,val_scaled ,train_target ,val_target = train_test_split (train_input , train_target ,test_size=0.2 , random_state=0)
lr.fit (train_scaled ,train_target )
pred=lr.predict (train_scaled )
pred_val = lr.predict(val_scaled)

#검증
print(lr.score(val_scaled,val_target))
print(pred_val[0])
```

```
0.09196725854235895
```

The validation process of linear regression model

```
model = MLPClassifier (hidden_layer_sizes=(3),activation='logistic',solver='lbfgs',random_state=1 ,max_iter= 10000)
model.fit (train_scaled ,train_target)
#검증
pred_val = model.predict(val_scaled)
from sklearn.metrics import accuracy_score,classification_report

print(accuracy_score(val_target,pred_val))
```

```
0.48333333333333334
```

The validation process of neural net model

```
rf.fit (train_scaled ,train_target)

#검증
pred_v = rf.predict(val_scaled)
print(accuracy_score(val_target,pred_v))
print(classification_report(val_target,pred_v))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_sele
  UserWarning,
0.49166666666666664
                precision    recall  f1-score   support

          Cook       0.69      0.39      0.50       260
           Eat       0.00      0.00      0.00        98
Go to bathroom       0.40      0.87      0.55       298
       Go walk       0.00      0.00      0.00         1
         Sleep       0.99      1.00      0.99        81
 Take medicine       0.00      0.00      0.00        13
   Take shower       0.00      0.00      0.00        47
       Wake up       0.37      0.50      0.43        46
   Wash dishes       0.58      0.06      0.11       116

      accuracy                           0.49       960
     macro avg       0.34      0.31      0.29       960
  weighted avg       0.48      0.49      0.42       960
```

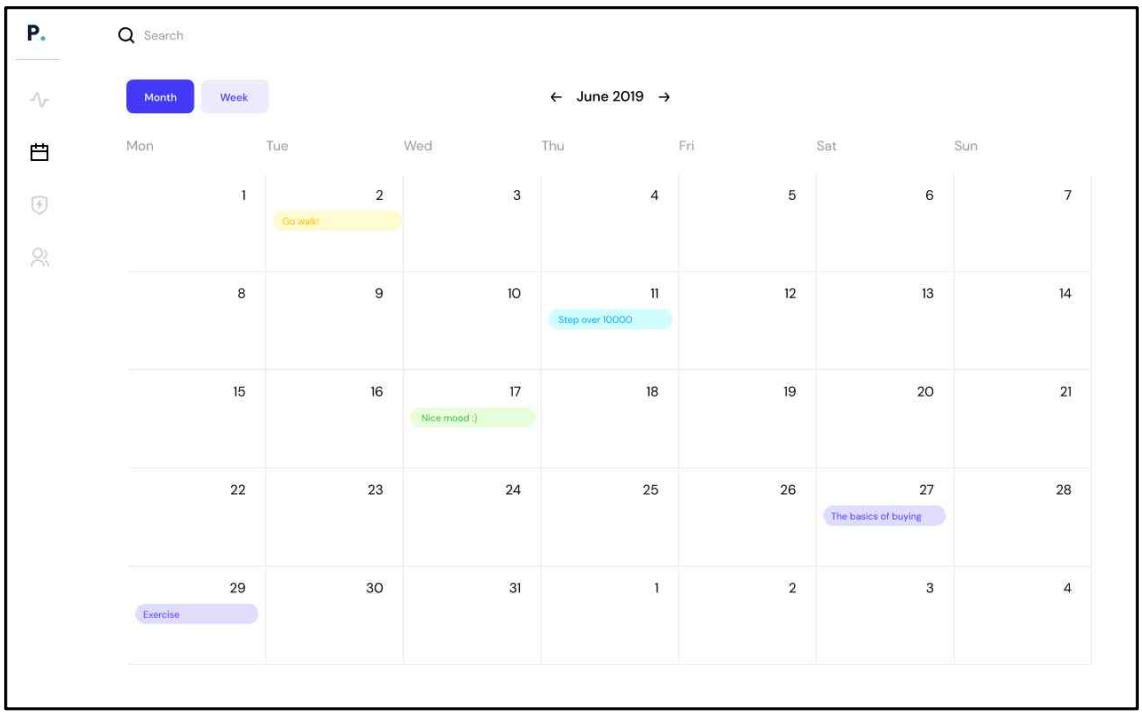
The validation process of random forest model

The actual value and the value predicted by the model were compared through the following code. We first tried to measure with f1-score, but f1-score was used for only binary classification, so we measured accuracy. First, in the case of linear regression, it was difficult to obtain an accurate classification value because both input and output were changed into dummy variables. The accuracy rate of the linear regression was 9%. Next, the accuracy rate of the neural net was 48.3%. Last, the accuracy rate of the random forest was 49.2%. We chose the random forest model for our final model because the accuracy was slightly higher and the execution time was a lot faster than the neural net. Although we have used various methods to boost the accuracy, there was a limit to getting a higher accuracy than this since the absolute number of data was small and tagging information was concentrated on specific tags like cook and toilet.

Using the constructed model, it is possible to predict the user's behavior by using the current time as input. By using this result, we can provide various functions for the users' convenience.

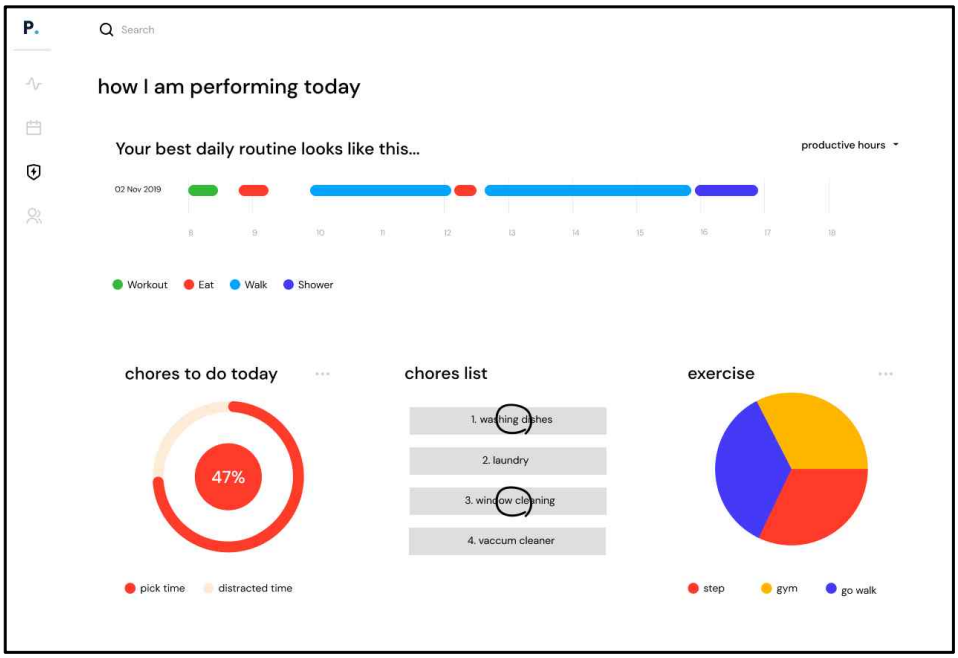
regular life management.
When a tag such as sleep and tag is generated by the detector, if the class predicted by the model is different from the generated tag, the program can determine that the user is not having a regular life.
Health care
For those who regularly take medicine, the model can alarm users when the input is the current time and the output is ‘medicine’.
To-do checklist
When a tag such as ‘bath’ or ‘medicine’ is not generated even though it is predicted, the model can leave a message for it.

Data mining is important not only for models but also for understanding and processing the data. The pictures below are examples of the combination of our model and the data pre-processing process.



This is an example of analyzing a user's daily remarkable activities and recording them in a calendar. This allows the user to obtain a record of what he did on which day.

It is possible to analyze the user's tag information and present it as a report to the user. The current productivity can be checked, and whether the user has lived a regular life also can be checked.



These figures are tables showing the details of previous examples. It also includes to-do checklists. The user can identify the daily routine of the day and determine whether the day's work is finished or not through the chart above. The user can check at a glance what time he/she is most productive. Analyzing the data and making a model can provide a lot of analysis information to the user, which is expected to help the user's health management and pattern identification.