기계학습 팀프로젝트 1

19011494 조국희

01. EMA 데이터를 이용한 우울증 예측

- **프로젝트 목적**

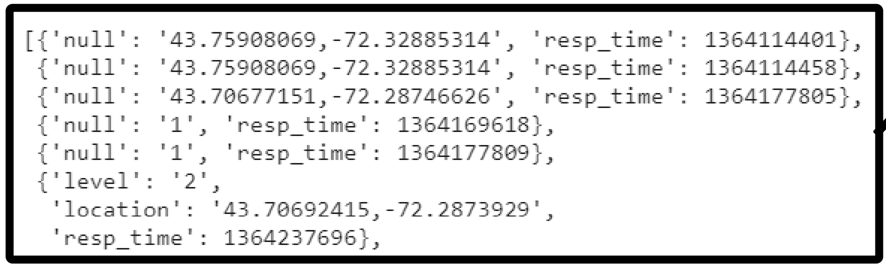
  실제 RAW 데이터를 직접 가공(Data Preprocessing)하고, 가공된 데이터에서 유의미한 피처를 추출하는 과정(Feature Extraction)을 배울 수 있다.

- **데이터셋**
  - ✓ 다트머스대학교 학생들을 대상으로 일정기간 동안 수집된 데이터
  - ✓ 학생들이 응답한 다양한 생태순간평가(EMA: Ecological Momentary Assessment) 결과로 우울증 여부를 예측
  - ✓ User의 EMA 응답 결과(Index)는 json파일 형태로 되어 있음

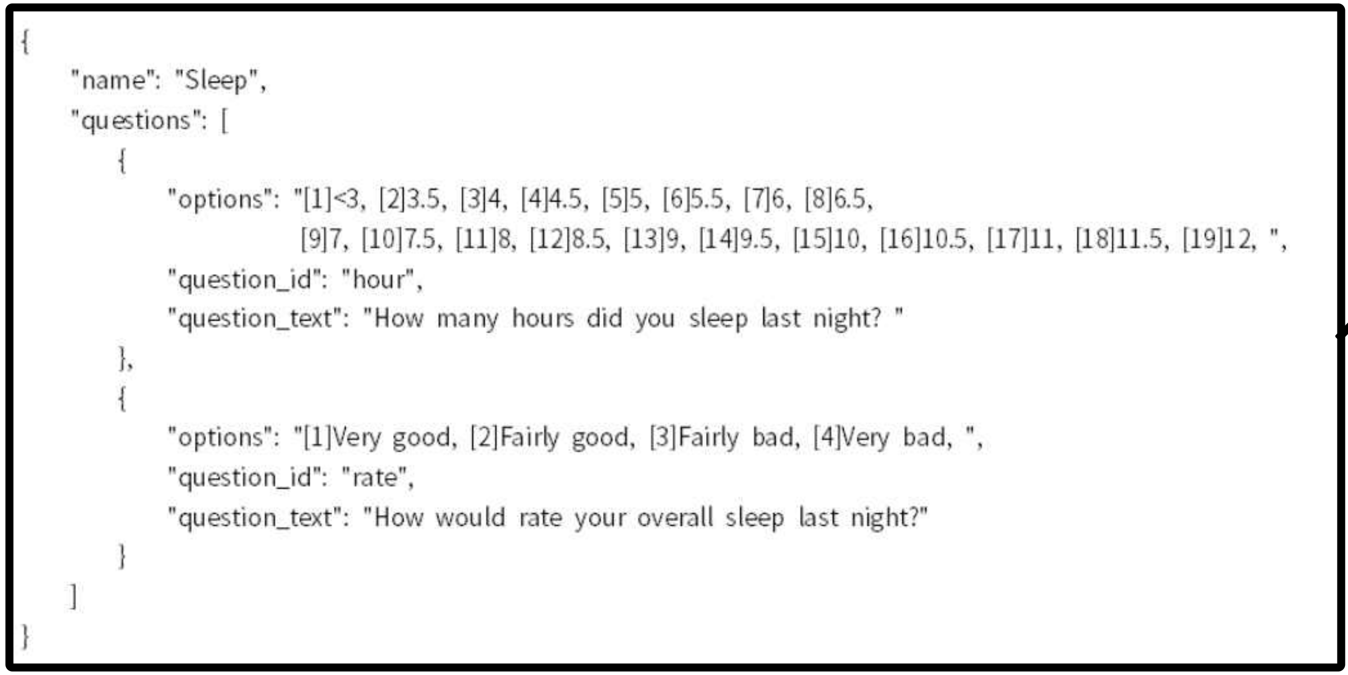
01. EMA 데이터를 이용한 우울증 예측

▪ EMA 데이터 파싱 - Sleep

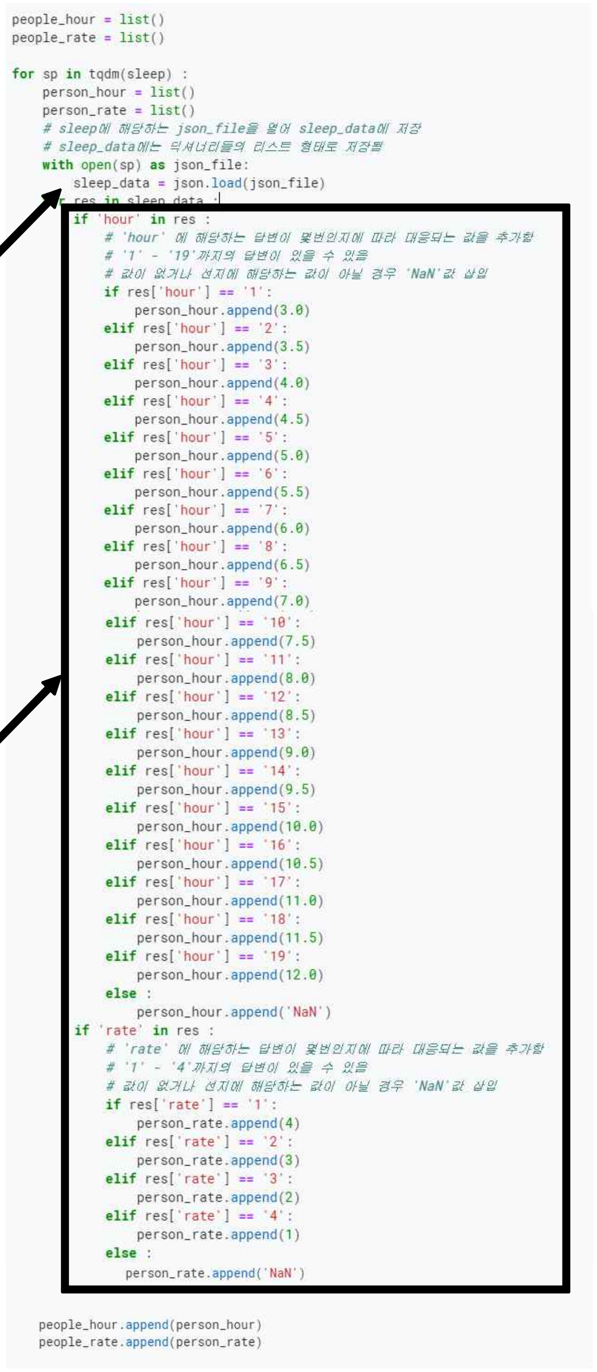
Sleep data



```
[{'null': '43.75908069,-72.32885314', 'resp_time': 1364114401},
 {'null': '43.75908069,-72.32885314', 'resp_time': 1364114458},
 {'null': '43.70677151,-72.28746626', 'resp_time': 1364177805},
 {'null': '1', 'resp_time': 1364169618},
 {'null': '1', 'resp_time': 1364177809},
 {'level': '2',
  'location': '43.70692415,-72.2873929',
  'resp_time': 1364237696},
```



```
{
    "name": "Sleep",
    "questions": [
        {
            "options": "[1]<3, [2]3.5, [3]4, [4]4.5, [5]5, [6]5.5, [7]6, [8]6.5,
                        [9]7, [10]7.5, [11]8, [12]8.5, [13]9, [14]9.5, [15]10, [16]10.5, [17]11, [18]11.5, [19]12, ",
            "question_id": "hour",
            "question_text": "How many hours did you sleep last night? "
        },
        {
            "options": "[1]Very good, [2]Fairly good, [3]Fairly bad, [4]Very bad, ",
            "question_id": "rate",
            "question_text": "How would rate your overall sleep last night?"
        }
    ]
}
```

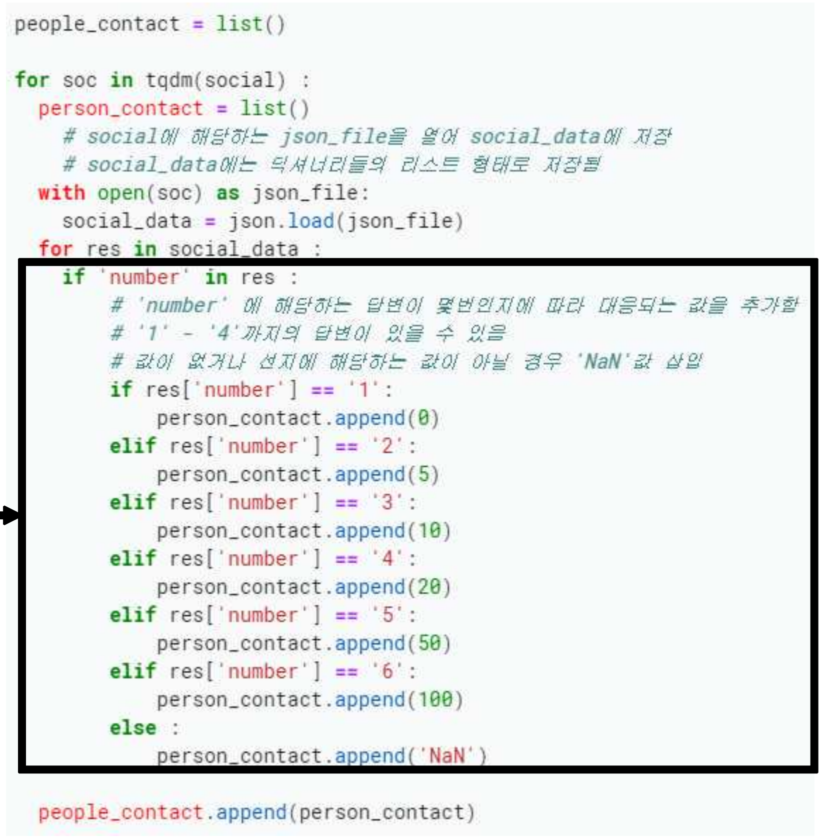


```python
people_hour = list()
people_rate = list()

for sp in tqdm(sleep) :
    person_hour = list()
    person_rate = list()
    # sleep에 해당하는 json_file을 열어 sleep_data에 저장
    # sleep_data에는 딕셔너리들의 리스트 형태로 저장됨
    with open(sp) as json_file:
        sleep_data = json.load(json_file)
    r res in sleep_data :
            if 'hour' in res :
                # 'hour' 에 해당하는 답변이 몇번인지에 따라 대응되는 값을 추가함
                # '1' - '19'까지의 답변이 있을 수 있음
                # 값이 없거나 선지에 해당하는 값이 아닐 경우 'NaN'값 삽입
                if res['hour'] == '1':
                    person_hour.append(3.0)
                elif res['hour'] == '2':
                    person_hour.append(3.5)
                elif res['hour'] == '3':
                    person_hour.append(4.0)
                elif res['hour'] == '4':
                    person_hour.append(4.5)
                elif res['hour'] == '5':
                    person_hour.append(5.0)
                elif res['hour'] == '6':
                    person_hour.append(5.5)
                elif res['hour'] == '7':
                    person_hour.append(6.0)
                elif res['hour'] == '8':
                    person_hour.append(6.5)
                elif res['hour'] == '9':
                    person_hour.append(7.0)
                elif res['hour'] == '10':
                    person_hour.append(7.5)
                elif res['hour'] == '11':
                    person_hour.append(8.0)
                elif res['hour'] == '12':
                    person_hour.append(8.5)
                elif res['hour'] == '13':
                    person_hour.append(9.0)
                elif res['hour'] == '14':
                    person_hour.append(9.5)
                elif res['hour'] == '15':
                    person_hour.append(10.0)
                elif res['hour'] == '16':
                    person_hour.append(10.5)
                elif res['hour'] == '17':
                    person_hour.append(11.0)
                elif res['hour'] == '18':
                    person_hour.append(11.5)
                elif res['hour'] == '19':
                    person_hour.append(12.0)
                else :
                    person_hour.append('NaN')
            if 'rate' in res :
                # 'rate' 에 해당하는 답변이 몇번인지에 따라 대응되는 값을 추가함
                # '1' - '4'까지의 답변이 있을 수 있음
                # 값이 없거나 선지에 해당하는 값이 아닐 경우 'NaN'값 삽입
                if res['rate'] == '1':
                    person_rate.append(4)
                elif res['rate'] == '2':
                    person_rate.append(3)
                elif res['rate'] == '3':
                    person_rate.append(2)
                elif res['rate'] == '4':
                    person_rate.append(1)
                else :
                   person_rate.append('NaN')

    people_hour.append(person_hour)
    people_rate.append(person_rate)
```

01. EMA 데이터를 이용한 우울증 예측

▪ EMA 데이터 파싱 - Social

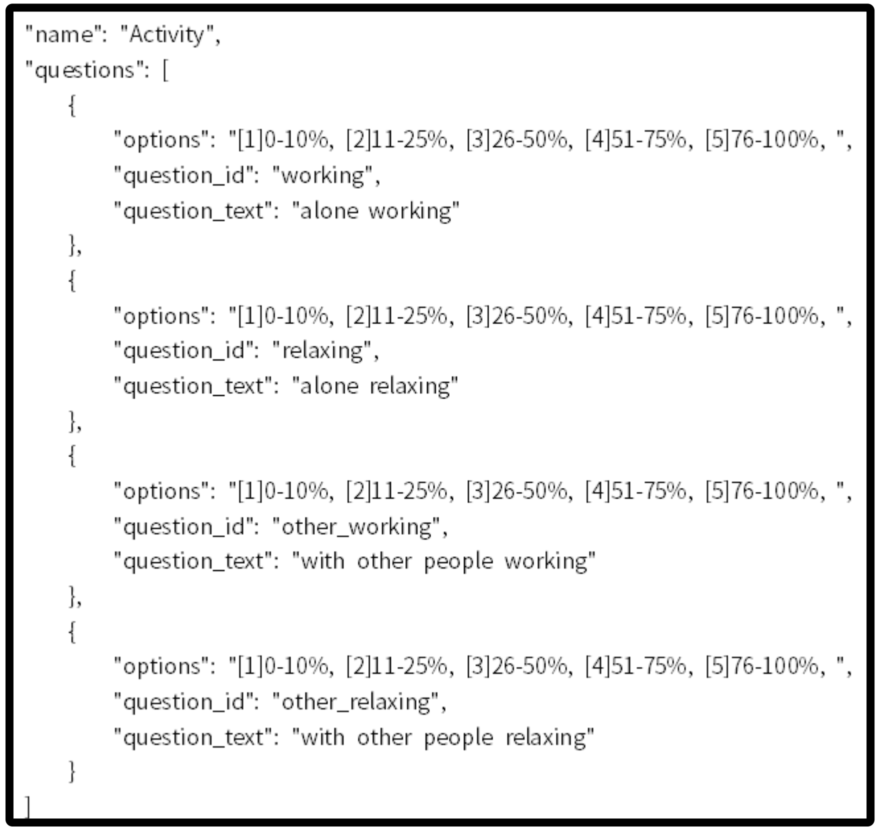
```
"name": "Social",
"questions": [
    {
        "options": "[1]0-4 persons, [2]5-9 persons, [3]10-19 persons,
                    [4]20-49 persons, [5]50-99 persons, [6]over 100 persons, ",
        "question_id": "number",
        "question_text": "How many people did you have contact with yesterday,
                         including anyone you said hello to, chatted, talked or discussed matters with,
                         whether you did it face-to-face, by telephone, by mail or on the internet,
                         and whether you personally knew the person or not? Please select one of the
                         following categories that best matches your estimate."
    }
]
```



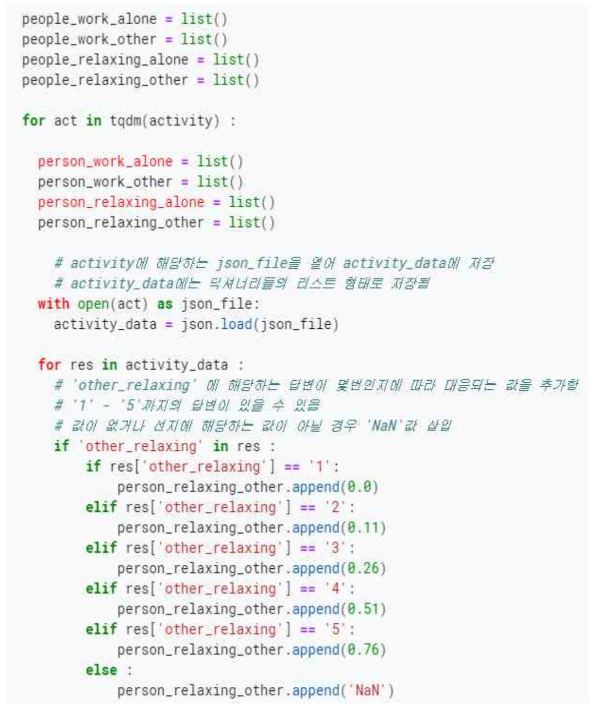
```python
people_contact = list()

for soc in tqdm(social) :
  person_contact = list()
    # social에 해당하는 json_file을 열어 social_data에 저장
    # social_data에는 딕셔너리들의 리스트 형태로 저장됨
  with open(soc) as json_file:
    social_data = json.load(json_file)
  for res in social_data :
    if 'number' in res :
        # 'number' 에 해당하는 답변이 몇번인지에 따라 대응되는 값을 추가함
        # '1' - '4'까지의 답변이 있을 수 있음
        # 값이 없거나 선지에 해당하는 값이 아닐 경우 'NaN'값 삽입
        if res['number'] == '1':
            person_contact.append(0)
        elif res['number'] == '2':
            person_contact.append(5)
        elif res['number'] == '3':
            person_contact.append(10)
        elif res['number'] == '4':
            person_contact.append(20)
        elif res['number'] == '5':
            person_contact.append(50)
        elif res['number'] == '6':
            person_contact.append(100)
        else :
            person_contact.append('NaN')

  people_contact.append(person_contact)
```

01. EMA 데이터를 이용한 우울증 예측

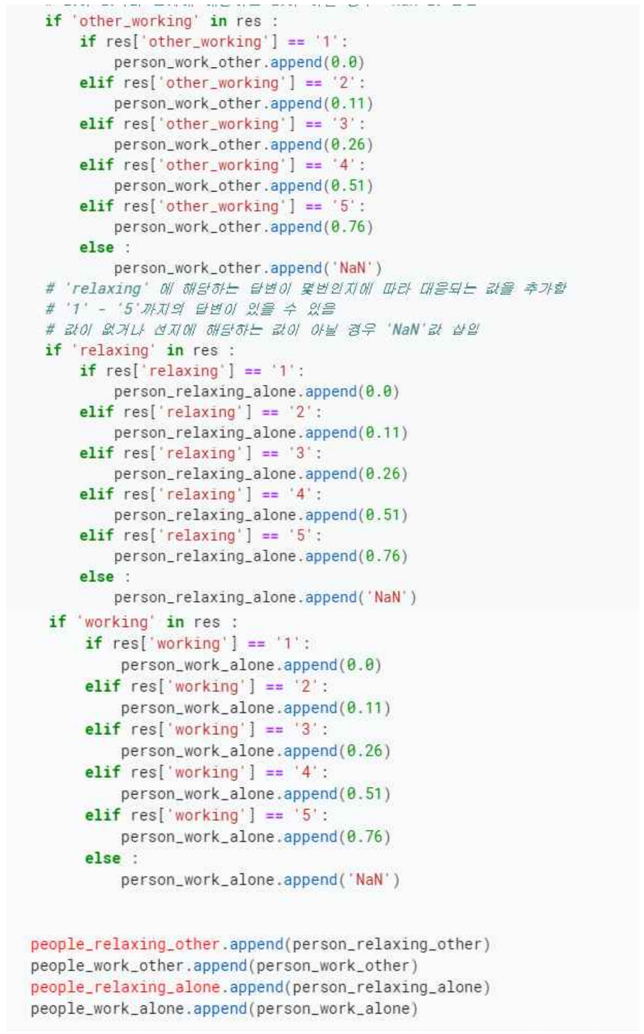
▪ EMA 데이터 파싱 - Activity



```
"name": "Activity",
"questions": [
    {
        "options": "[1]0-10%, [2]11-25%, [3]26-50%, [4]51-75%, [5]76-100%, ",
        "question_id": "working",
        "question_text": "alone working"
    },
    {
        "options": "[1]0-10%, [2]11-25%, [3]26-50%, [4]51-75%, [5]76-100%, ",
        "question_id": "relaxing",
        "question_text": "alone relaxing"
    },
    {
        "options": "[1]0-10%, [2]11-25%, [3]26-50%, [4]51-75%, [5]76-100%, ",
        "question_id": "other_working",
        "question_text": "with other people working"
    },
    {
        "options": "[1]0-10%, [2]11-25%, [3]26-50%, [4]51-75%, [5]76-100%, ",
        "question_id": "other_relaxing",
        "question_text": "with other people relaxing"
    }
]
```

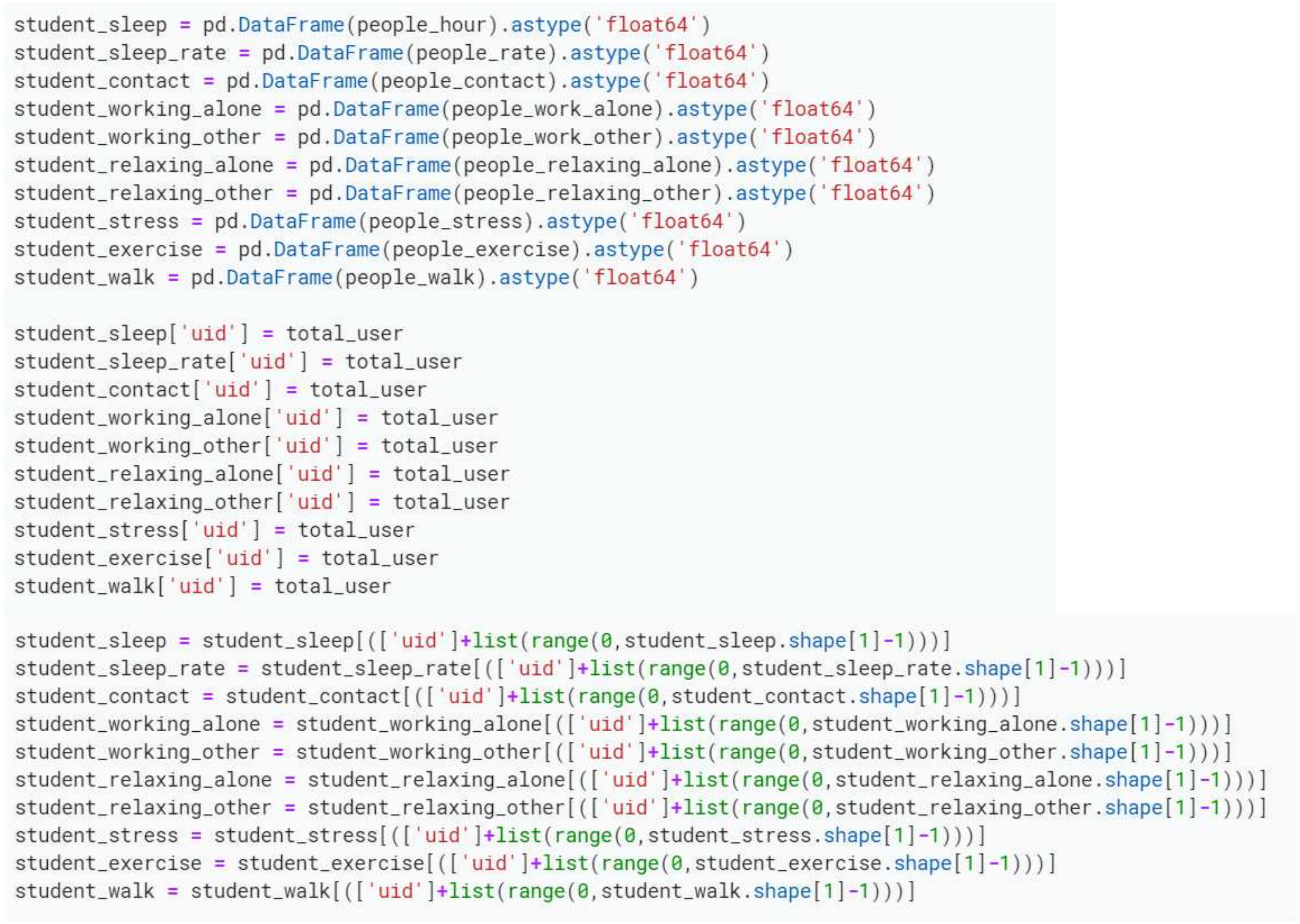


```
people_work_alone = list()
people_work_other = list()
people_relaxing_alone = list()
people_relaxing_other = list()

for act in tqdm(activity) :

  person_work_alone = list()
  person_work_other = list()
  person_relaxing_alone = list()
  person_relaxing_other = list()

    # activity에 해당하는 json_file을 열어 activity_data에 저장
    # activity_data에는 딕셔너리들의 리스트 형태로 저장됨
  with open(act) as json_file:
    activity_data = json.load(json_file)

  for res in activity_data :
    # 'other_relaxing' 에 해당하는 답변이 몇번인지에 따라 대응되는 값을 추가함
    # '1' - '5'까지의 답변이 있을 수 있음
    # 값이 없거나 선지에 해당하는 값이 아닐 경우 'NaN'값 삽입
    if 'other_relaxing' in res :
        if res['other_relaxing'] == '1':
            person_relaxing_other.append(0.0)
        elif res['other_relaxing'] == '2':
            person_relaxing_other.append(0.11)
        elif res['other_relaxing'] == '3':
            person_relaxing_other.append(0.26)
        elif res['other_relaxing'] == '4':
            person_relaxing_other.append(0.51)
        elif res['other_relaxing'] == '5':
            person_relaxing_other.append(0.76)
        else :
            person_relaxing_other.append('NaN')
```



```
    if 'other_working' in res :
        if res['other_working'] == '1':
            person_work_other.append(0.0)
        elif res['other_working'] == '2':
            person_work_other.append(0.11)
        elif res['other_working'] == '3':
            person_work_other.append(0.26)
        elif res['other_working'] == '4':
            person_work_other.append(0.51)
        elif res['other_working'] == '5':
            person_work_other.append(0.76)
        else :
            person_work_other.append('NaN')
    # 'relaxing' 에 해당하는 답변이 몇번인지에 따라 대응되는 값을 추가함
    # '1' - '5'까지의 답변이 있을 수 있음
    # 값이 없거나 선지에 해당하는 값이 아닐 경우 'NaN'값 삽입
    if 'relaxing' in res :
        if res['relaxing'] == '1':
            person_relaxing_alone.append(0.0)
        elif res['relaxing'] == '2':
            person_relaxing_alone.append(0.11)
        elif res['relaxing'] == '3':
            person_relaxing_alone.append(0.26)
        elif res['relaxing'] == '4':
            person_relaxing_alone.append(0.51)
        elif res['relaxing'] == '5':
            person_relaxing_alone.append(0.76)
        else :
            person_relaxing_alone.append('NaN')
    if 'working' in res :
        if res['working'] == '1':
            person_work_alone.append(0.0)
        elif res['working'] == '2':
            person_work_alone.append(0.11)
        elif res['working'] == '3':
            person_work_alone.append(0.26)
        elif res['working'] == '4':
            person_work_alone.append(0.51)
        elif res['working'] == '5':
            person_work_alone.append(0.76)
        else :
            person_work_alone.append('NaN')


  people_relaxing_other.append(person_relaxing_other)
  people_work_other.append(person_work_other)
  people_relaxing_alone.append(person_relaxing_alone)
  people_work_alone.append(person_work_alone)
```

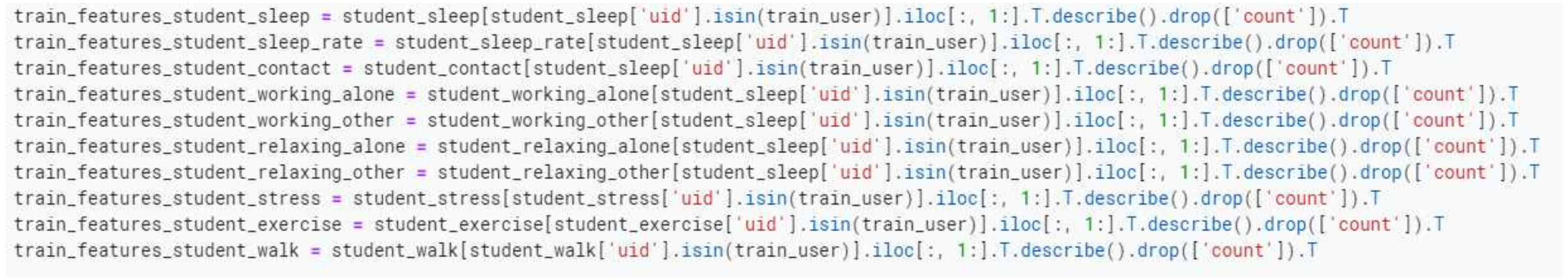
앞에서 얻은 데이터를 데이터 프레임 형식으로 만들고, 제일 앞 열에 유저 아이디를 추가



```
student_sleep = pd.DataFrame(people_hour).astype('float64')
student_sleep_rate = pd.DataFrame(people_rate).astype('float64')
student_contact = pd.DataFrame(people_contact).astype('float64')
student_working_alone = pd.DataFrame(people_work_alone).astype('float64')
student_working_other = pd.DataFrame(people_work_other).astype('float64')
student_relaxing_alone = pd.DataFrame(people_relaxing_alone).astype('float64')
student_relaxing_other = pd.DataFrame(people_relaxing_other).astype('float64')
student_stress = pd.DataFrame(people_stress).astype('float64')
student_exercise = pd.DataFrame(people_exercise).astype('float64')
student_walk = pd.DataFrame(people_walk).astype('float64')

student_sleep['uid'] = total_user
student_sleep_rate['uid'] = total_user
student_contact['uid'] = total_user
student_working_alone['uid'] = total_user
student_working_other['uid'] = total_user
student_relaxing_alone['uid'] = total_user
student_relaxing_other['uid'] = total_user
student_stress['uid'] = total_user
student_exercise['uid'] = total_user
student_walk['uid'] = total_user

student_sleep = student_sleep[(['uid']+list(range(0,student_sleep.shape[1]-1)))]
student_sleep_rate = student_sleep_rate[(['uid']+list(range(0,student_sleep_rate.shape[1]-1)))]
student_contact = student_contact[(['uid']+list(range(0,student_contact.shape[1]-1)))]
student_working_alone = student_working_alone[(['uid']+list(range(0,student_working_alone.shape[1]-1)))]
student_working_other = student_working_other[(['uid']+list(range(0,student_working_other.shape[1]-1)))]
student_relaxing_alone = student_relaxing_alone[(['uid']+list(range(0,student_relaxing_alone.shape[1]-1)))]
student_relaxing_other = student_relaxing_other[(['uid']+list(range(0,student_relaxing_other.shape[1]-1)))]
student_stress = student_stress[(['uid']+list(range(0,student_stress.shape[1]-1)))]
student_exercise = student_exercise[(['uid']+list(range(0,student_exercise.shape[1]-1)))]
student_walk = student_walk[(['uid']+list(range(0,student_walk.shape[1]-1)))]
```

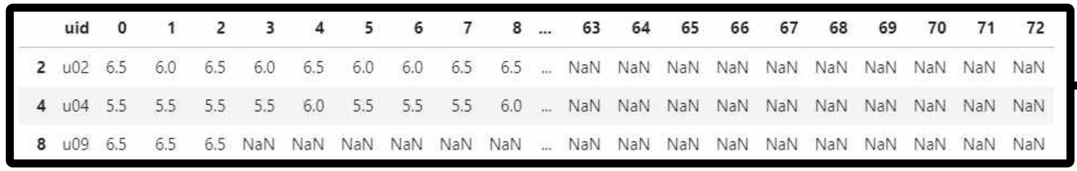
01. EMA 데이터를 이용한 우울증 예측

▪ Feature extract



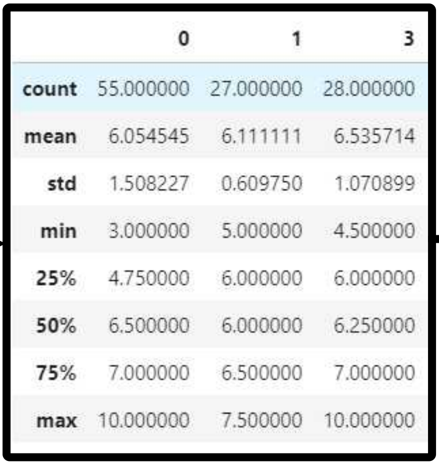
```
train_features_student_sleep = student_sleep[student_sleep['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
train_features_student_sleep_rate = student_sleep_rate[student_sleep['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
train_features_student_contact = student_contact[student_sleep['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
train_features_student_working_alone = student_working_alone[student_sleep['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
train_features_student_working_other = student_working_other[student_sleep['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
train_features_student_relaxing_alone = student_relaxing_alone[student_sleep['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
train_features_student_relaxing_other = student_relaxing_other[student_sleep['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
train_features_student_stress = student_stress[student_stress['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
train_features_student_exercise = student_exercise[student_exercise['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
train_features_student_walk = student_walk[student_walk['uid'].isin(train_user)].iloc[:, 1:].T.describe().drop(['count']).T
```

- ✓ train/test 데이터에 대한 통계적 feature 추출
- ✓ A[A['uid'].isin(train_user)].iloc[:, 1:] : train_user가 있는 데이터의 uid만 제외한 Dataframe을 추출한 것
- ✓ .T.describe() : 위의 Dataframe의 행과 열을 바꾸어(각 유저의 통계를 구하기 위함) 통계적 feature를 추출
- ✓ .drop(['count']).T : count column은 제거한후 행과 열을 바꿈(각 유저를 하나의 행으로 볼 수 있도록 만들기 위함)



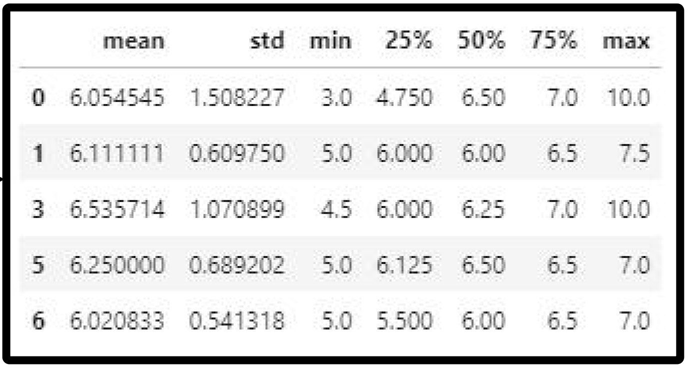
	uid	0	1	2	3	4	5	6	7	8	...	63	64	65	66	67	68	69	70	71	72
2	u02	6.5	6.0	6.5	6.0	6.5	6.0	6.0	6.5	6.5	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	u04	5.5	5.5	5.5	5.5	6.0	5.5	5.5	5.5	6.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	u09	6.5	6.5	6.5	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

student_sleep(student_sleep['uid'].isin(train_user)])



	0	1	3
count	55.000000	27.000000	28.000000
mean	6.054545	6.111111	6.535714
std	1.508227	0.609750	1.070899
min	3.000000	5.000000	4.500000
25%	4.750000	6.000000	6.000000
50%	6.500000	6.000000	6.250000
75%	7.000000	6.500000	7.000000
max	10.000000	7.500000	10.000000

.T.describe()



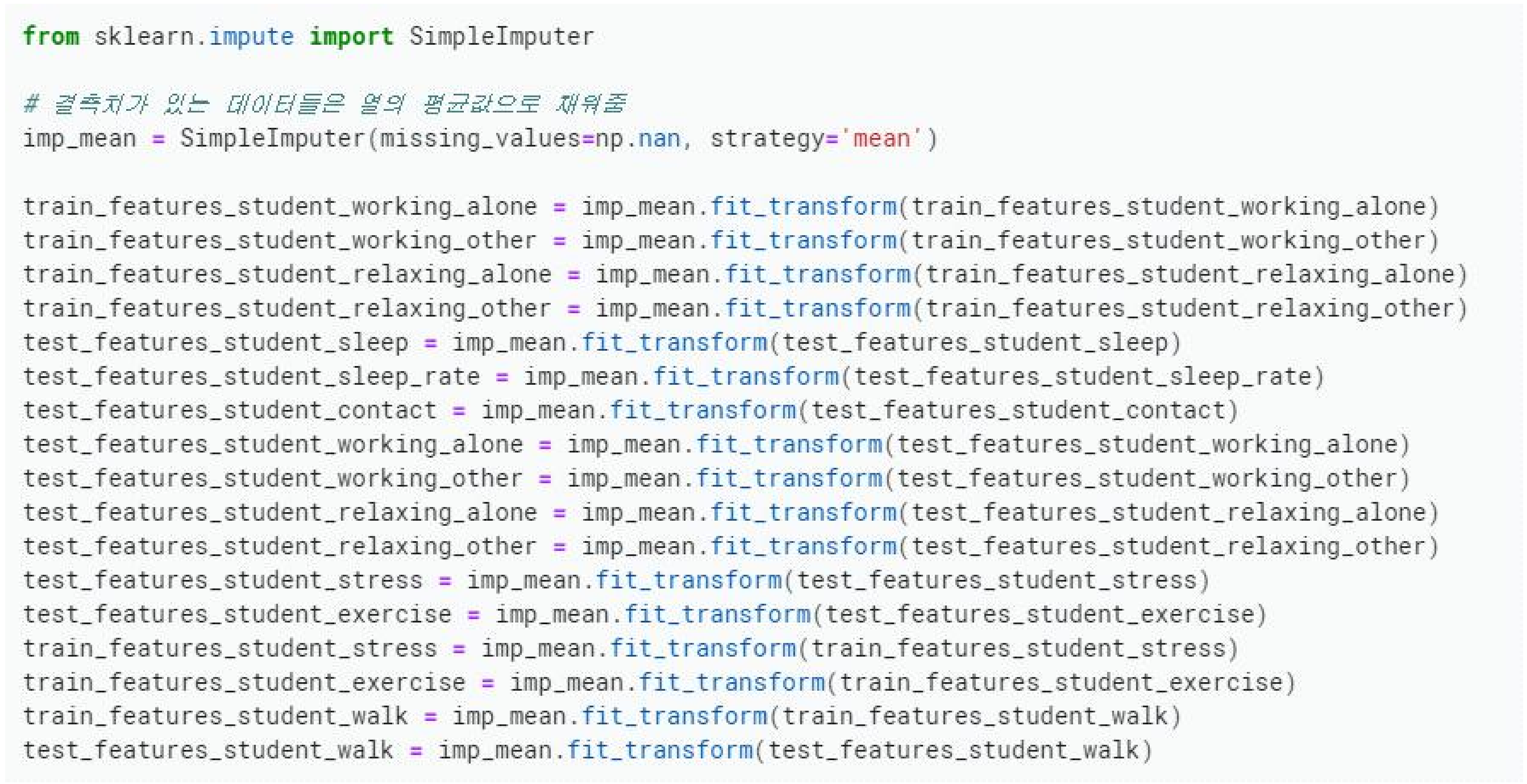
	mean	std	min	25%	50%	75%	max
0	6.054545	1.508227	3.0	4.750	6.50	7.0	10.0
1	6.111111	0.609750	5.0	6.000	6.00	6.5	7.5
3	6.535714	1.070899	4.5	6.000	6.25	7.0	10.0
5	6.250000	0.689202	5.0	6.125	6.50	6.5	7.0
6	6.020833	0.541318	5.0	5.500	6.00	6.5	7.0

.drop(['count']).T

01. EMA 데이터를 이용한 우울증 예측

▪ 결측치 처리

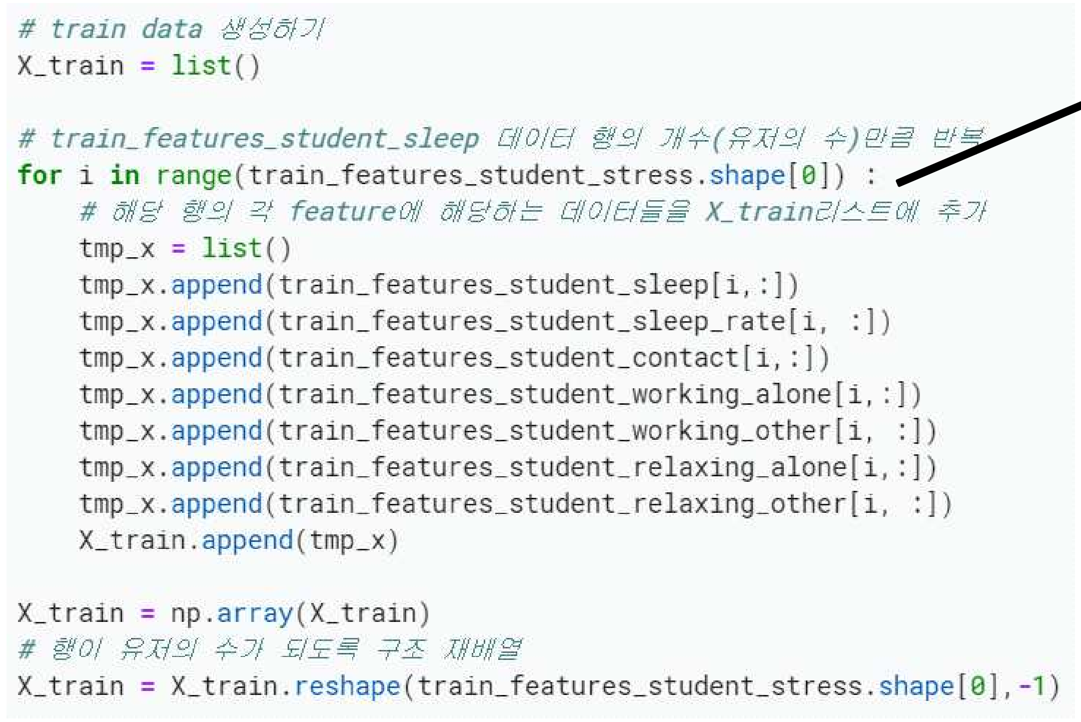
sklearn의 SimpleImputer 함수를 이용해 결측치를 열의 평균값으로 채워줌



```python
from sklearn.impute import SimpleImputer

# 결측치가 있는 데이터들은 열의 평균값으로 채워줌
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')

train_features_student_working_alone = imp_mean.fit_transform(train_features_student_working_alone)
train_features_student_working_other = imp_mean.fit_transform(train_features_student_working_other)
train_features_student_relaxing_alone = imp_mean.fit_transform(train_features_student_relaxing_alone)
train_features_student_relaxing_other = imp_mean.fit_transform(train_features_student_relaxing_other)
test_features_student_sleep = imp_mean.fit_transform(test_features_student_sleep)
test_features_student_sleep_rate = imp_mean.fit_transform(test_features_student_sleep_rate)
test_features_student_contact = imp_mean.fit_transform(test_features_student_contact)
test_features_student_working_alone = imp_mean.fit_transform(test_features_student_working_alone)
test_features_student_working_other = imp_mean.fit_transform(test_features_student_working_other)
test_features_student_relaxing_alone = imp_mean.fit_transform(test_features_student_relaxing_alone)
test_features_student_relaxing_other = imp_mean.fit_transform(test_features_student_relaxing_other)
test_features_student_stress = imp_mean.fit_transform(test_features_student_stress)
test_features_student_exercise = imp_mean.fit_transform(test_features_student_exercise)
train_features_student_stress = imp_mean.fit_transform(train_features_student_stress)
train_features_student_exercise = imp_mean.fit_transform(train_features_student_exercise)
train_features_student_walk = imp_mean.fit_transform(train_features_student_walk)
test_features_student_walk = imp_mean.fit_transform(test_features_student_walk)
```

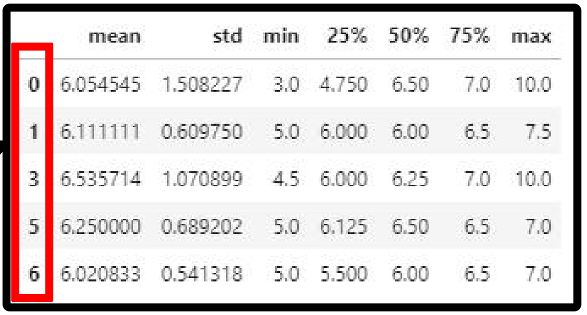
01. EMA 데이터를 이용한 우울증 예측

▪ 전처리한 데이터를 바탕으로 우울증 예측



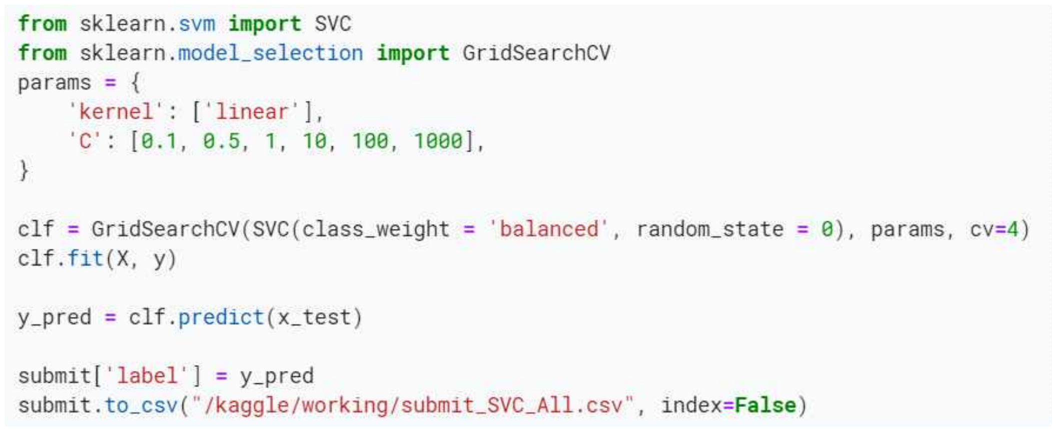
```python
# train data 생성하기
X_train = list()

# train_features_student_sleep 데이터 행의 개수(유저의 수)만큼 반복
for i in range(train_features_student_stress.shape[0]) :
    # 해당 행의 각 feature에 해당하는 데이터들을 X_train리스트에 추가
    tmp_x = list()
    tmp_x.append(train_features_student_sleep[i,:])
    tmp_x.append(train_features_student_sleep_rate[i, :])
    tmp_x.append(train_features_student_contact[i,:])
    tmp_x.append(train_features_student_working_alone[i,:])
    tmp_x.append(train_features_student_working_other[i, :])
    tmp_x.append(train_features_student_relaxing_alone[i,:])
    tmp_x.append(train_features_student_relaxing_other[i, :])
    X_train.append(tmp_x)

X_train = np.array(X_train)
# 행이 유저의 수가 되도록 구조 재배열
X_train = X_train.reshape(train_features_student_stress.shape[0],-1)
```

train, test 데이터 가공



	mean	std	min	25%	50%	75%	max
0	6.054545	1.508227	3.0	4.750	6.50	7.0	10.0
1	6.111111	0.609750	5.0	6.000	6.00	6.5	7.5
3	6.535714	1.070899	4.5	6.000	6.25	7.0	10.0
5	6.250000	0.689202	5.0	6.125	6.50	6.5	7.0
6	6.020833	0.541318	5.0	5.500	6.00	6.5	7.0

유저



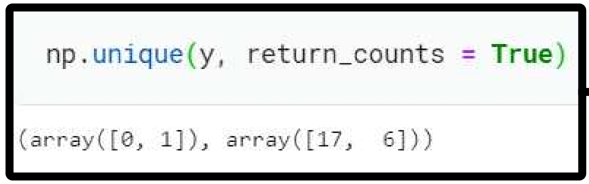
```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
params = {
    'kernel': ['linear'],
    'C': [0.1, 0.5, 1, 10, 100, 1000],
}

clf = GridSearchCV(SVC(class_weight = 'balanced', random_state = 0), params, cv=4)
clf.fit(X, y)

y_pred = clf.predict(x_test)

submit['label'] = y_pred
submit.to_csv("/kaggle/working/submit_SVC_All.csv", index=False)
```

GridSearchCV를 통해 최적의 파라미터를 찾아 모델학습 후 예측

01. EMA 데이터를 이용한 우울증 예측

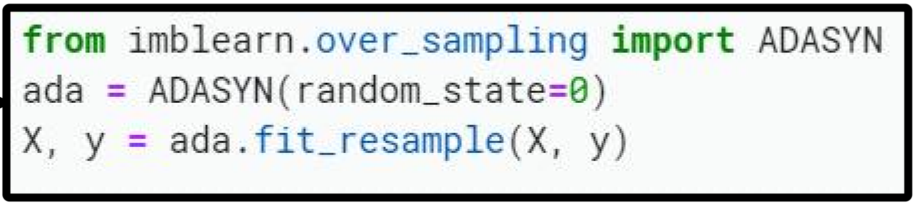
▪ 성능을 높이기 위한 시도

✓ 데이터 불균형을 해결하기 위해 oversampling



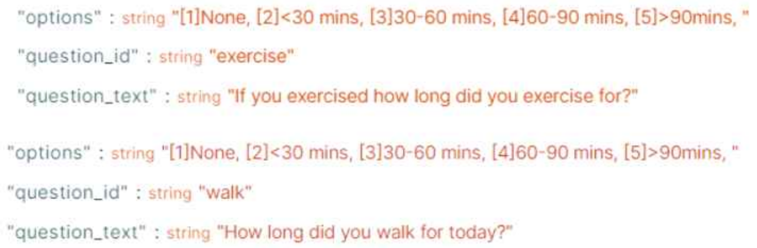
```
np.unique(y, return_counts = True)
```

```
(array([0, 1]), array([17,  6]))
```



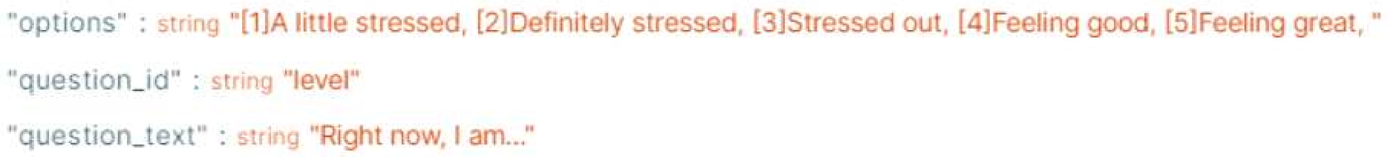
```
from imblearn.over_sampling import ADASYN
ada = ADASYN(random_state=0)
X, y = ada.fit_resample(X, y)
```

✓ 데이터셋에 있는 stress, exercise data 추가



```
"options" : string "[1]None, [2]<30 mins, [3]30-60 mins, [4]60-90 mins, [5]>90mins, "
"question_id" : string "exercise"
"question_text" : string "If you exercised how long did you exercise for?"
"options" : string "[1]None, [2]<30 mins, [3]30-60 mins, [4]60-90 mins, [5]>90mins, "
"question_id" : string "walk"
"question_text" : string "How long did you walk for today?"
```

exercise



```
"options" : string "[1]A little stressed, [2]Definitely stressed, [3]Stressed out, [4]Feeling good, [5]Feeling great, "
"question_id" : string "level"
"question_text" : string "Right now, I am..."
```

stress

01. EMA 데이터를 이용한 우울증 예측

- **학습 성능**

베이스라인	성능
Baseline1(Social, Acticity)	0.52173
Baseline2(Sleep)	0.65217
Baseline3(Social+ Activity+Sleep)	0.82608

모델	성능
SVM + Activity	0.73913
SVM + Social	0.56521
SVM + Sleep	0.65217
SVM + Social+Sleep+Activity	0.82608
SVM + Social+Sleep+Activity+ADASYN	**0.86956**
SVM+Social+Sleep+Activity+Stress+Excercise	0.56521
SVM+Social+Sleep+Activity+Stress+Excercise + ADASYN	0.56521