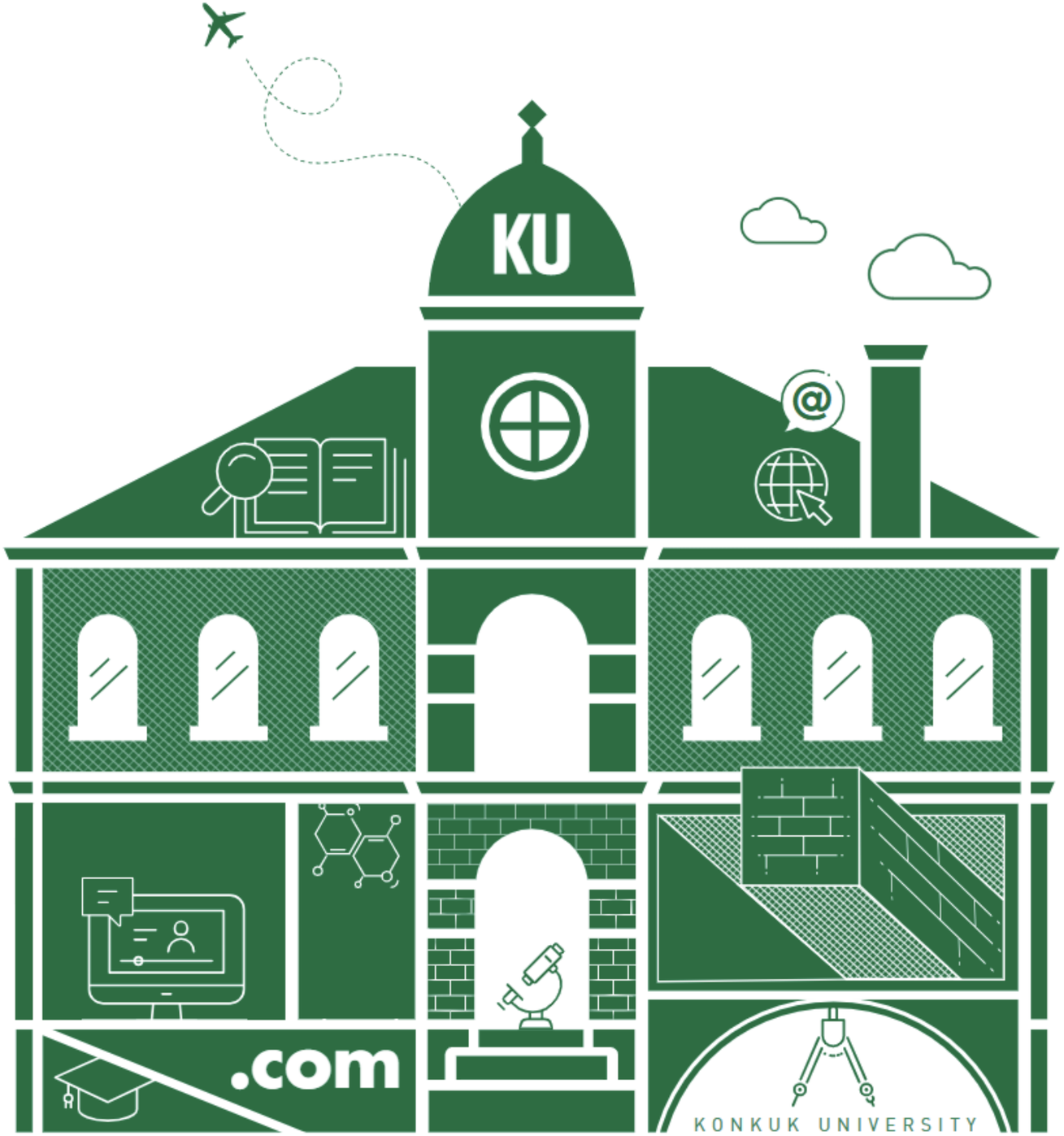


KUGGLE 12기 1차 프로젝트

김세영
정민경
천시원
황천조

K
U

CONTENTS

KONKUK

데이터 소개

1. 데이터 소개

#	Column	Non-Null Count	Dtype
0	id	140700 non-null	int64
1	Name	140700 non-null	object
2	Gender	140700 non-null	object
3	Age	140700 non-null	int64
4	City	140700 non-null	object
5	Working Professional or Student	140700 non-null	object
6	Profession	104070 non-null	object
7	Academic Pressure	27897 non-null	float64
8	Work Pressure	112782 non-null	float64
9	CGPA	27898 non-null	float64
10	Study Satisfaction	27897 non-null	float64
11	Job Satisfaction	112790 non-null	float64
12	Sleep Duration	140700 non-null	object
13	Dietary Habits	140696 non-null	object
14	Degree	140698 non-null	object
15	Have you ever had suicidal thoughts ?	140700 non-null	object
16	Work/Study Hours	140700 non-null	int64
17	Financial Stress	140696 non-null	float64
18	Family History of Mental Illness	140700 non-null	object
19	Depression	140700 non-null	int64

1. 데이터 구성
   - id,
   - Gender(성별),
   - Age(나이),
   - City(도시),
   - Working Professional or Student(직업상태),
   - Profession(직업),
   - Academic Pressure(학업 스트레스),
   - Work Pressure(업무 스트레스),
   - CGPA(학점),
   - Study Satisfaction(학업 만족도),
   - Job Satisfaction(직업 만족도),
   - Sleep Duration(수면시간),
   - Dietary Habits(식습관),
   - Degree(학위),
   - Have you ever had suicidal thoughts?(자살 생각 경험),
   - Work/Study Hours(일/공부 시간),
   - Financial Stress(경제적 스트레스),
   - Family History of Mental Illness(가족력),
   - Depression(우울증 여부) 등

2. 데이터 규모
   - 약 27,901명(학생+직장인) 데이터, 16개 컬럼
   - 직장인 데이터는 약 112,799명, 학생 데이터는 약 27,784명 등으로 분리되어 있음

EDA & 데이터 전처리

2. EDA & 데이터 전처리

```
[ ]  1 # 중복된 행 확인
     2 print('중복된 데이터 개수 : ', Health_df_train.duplicated().sum())
```

```
중복된 데이터 개수 :  0
```

```
[ ]  1 # 분석간 필요없는 열을 제거 (id, name : 개인 정보이므로 분석에 필요없다고 판단)
     2 Health_df_train_1 = Health_df_train.drop(columns=['Name'])
     3 Health_df_test_1 = Health_df_test.drop(columns=['Name'])
```

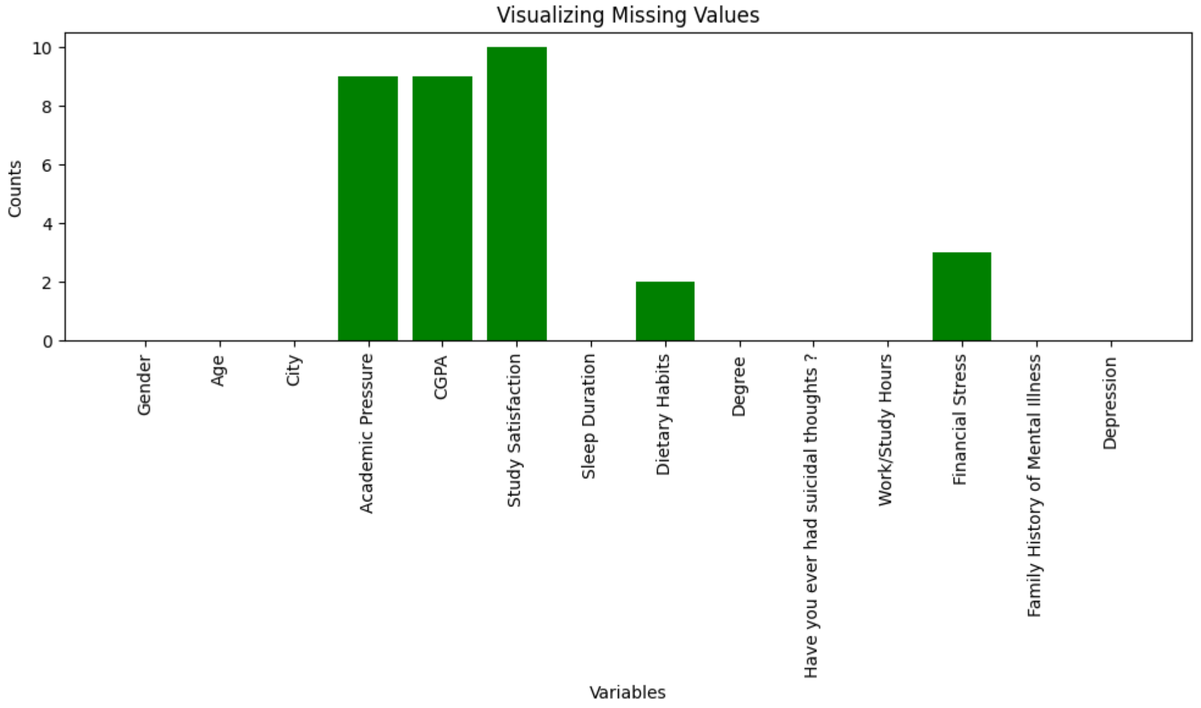
```
1 # 학생 전용, 직장인 전용 칼럼이 따로 있기 때문에 나누어 전처리를 진행
2 print(Health_df_train_1["Working Professional or Student"].value_counts())
3 print(Health_df_test_1['Working Professional or Student'].value_counts())
4
5 # 학생과 직장인으로 그룹을 나누기
6 student_train = Health_df_train_1[Health_df_train_1['Working Professional or Student']=='Student'].copy()
7 worker_train = Health_df_train_1[Health_df_train_1['Working Professional or Student']=='Working Professional'].copy()
8 student_test = Health_df_test_1[Health_df_test_1['Working Professional or Student']=='Student'].copy()
9 worker_test = Health_df_test_1[Health_df_test_1['Working Professional or Student']=='Working Professional'].copy()
```

```
Working Professional or Student
Working Professional    112799
Student                  27901
Name: count, dtype: int64
Working Professional or Student
Working Professional    75028
Student                 18772
Name: count, dtype: int64
```

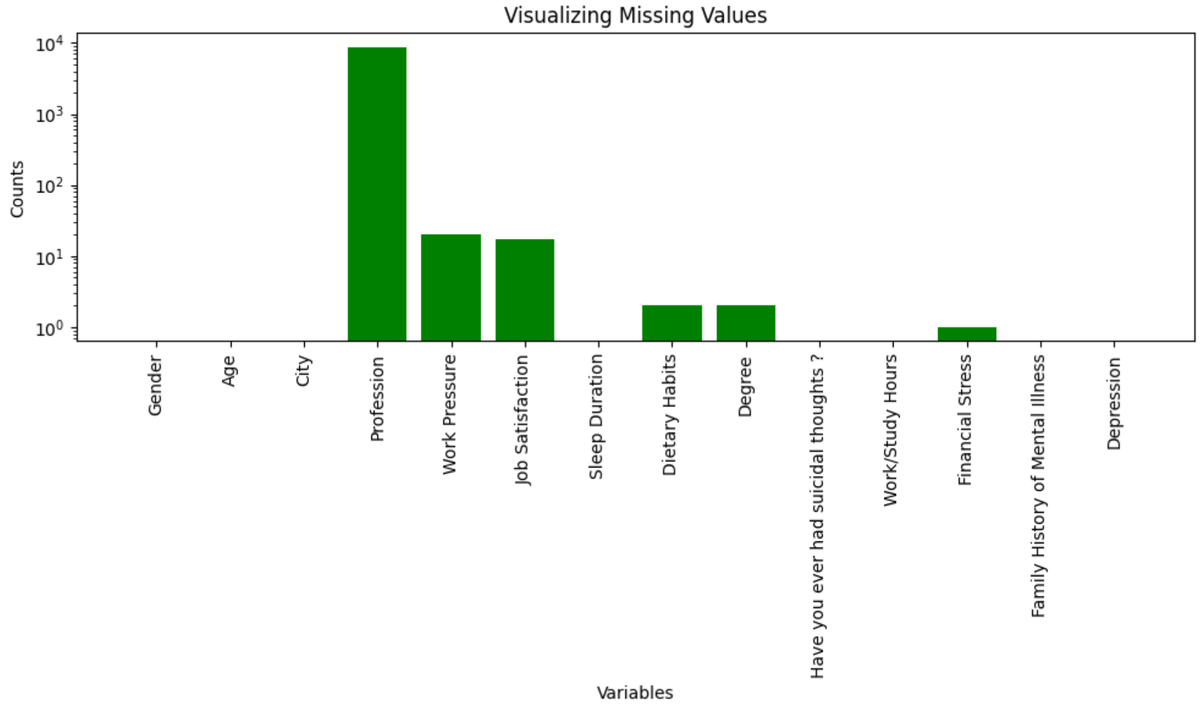
학생 그룹과 직장인 그룹을 나누어서 진행하는 것이 더 적합하다고 생각하여 그룹을 나누고 데이터 전처리를 진행

```
[ ]
1 # 컬럼 삭제
2 student_train.drop(columns = ['Profession', 'Work Pressure', 'Job Satisfaction'], inplace = True)
3 student_test.drop(columns = ['Profession', 'Work Pressure', 'Job Satisfaction'], inplace = True)
4 worker_train.drop(columns = ['Academic Pressure', 'CGPA', 'Study Satisfaction'], inplace = True)
5 worker_test.drop(columns = ['Academic Pressure', 'CGPA', 'Study Satisfaction'], inplace = True)
```

학생 그룹에서 직업, 업무 압박, 직업 만족도는 대부분 결측값 -> 삭제
직장인 그룹에서 학업 압박, 성적, 공부 만족도는 대부분 결측값 -> 삭제



전처리 전 학생 그룹 데이터 결측값



전처리 전 직장인 그룹 데이터 결측값

```
# 35세 이상 데이터는 모두 삭제
# test 데이터는 삭제하지 않고 최빈값으로 대체
student_train = student_train[student_train['Age']<=34]
student_test.loc[student_test['Age'] > 34, 'Age'] = student_test['Age'].median()
```

```
# 연속형 변수인 CGPA는 결측치 개수가 적기도 하니까 평균값으로 결측 값을 대체하자.
student_train['CGPA'].fillna(student_train['CGPA'].mean(), inplace=True)
student_test['CGPA'].fillna(student_test['CGPA'].mean(), inplace=True)

print('----------------전처리 결과 확인----------------')
student_train['CGPA'].value_counts()
```

```
# 결측치 처리 : 자료형이 float이긴 하지만 사실상 범주형 자료형이기 때문에 최빈값으로 대체
student_train['Academic Pressure'] = student_train['Academic Pressure'].fillna(student_train['Academic Pressure'].mode()[0])
student_test['Academic Pressure'] = student_test['Academic Pressure'].fillna(student_test['Academic Pressure'].mode()[0])
```

```
# 결측치 처리
student_train['Study Satisfaction'] = student_train['Study Satisfaction'].fillna(student_train['Study Satisfaction'].mode()[0])
student_test['Study Satisfaction'] = student_test['Study Satisfaction'].fillna(student_test['Study Satisfaction'].mode()[0])
```

```python
# 결측값의 개수가 많아서 삭제하긴 부담스러워서 Unknown으로 대체
missing = worker_train['Profession'].isna().sum()
print('결측값 비율 : ', (missing/len(worker_train)) * 100)
```

Profession Group	
Education	38586
Business	18704
Service	17779
Professional	10932
Medical	10113
Unknown	8763
IT	7884
Other	38

```python
# 결측치 처리 : 자료형이 float이긴 하지만 사실상 범주형 자료형이기 때문에 최빈값으로 대체
worker_train['Work Pressure'] = worker_train['Work Pressure'].fillna(worker_train['Work Pressure'].mode()[0])
worker_test['Work Pressure'] = worker_test['Work Pressure'].fillna(worker_test['Work Pressure'].mode()[0])
```

```python
# 결측치 처리
worker_train['Job Satisfaction'] = worker_train['Job Satisfaction'].fillna(worker_train['Job Satisfaction'].mode()[0])
worker_test['Job Satisfaction'] = worker_test['Job Satisfaction'].fillna(worker_test['Job Satisfaction'].mode()[0])
```

```python
pre_student_train = student_train.shape[0]
pre_worker_train = worker_train.shape[0]

print('전처리 과정에서 삭제된 학생 훈련 데이터 비율 : {0:.3f}%'.format((raw_student_train - pre_student_train)/raw_student_train*100))
print('전처리 과정에서 삭제된 직장인 훈련 데이터 비율 : {0:.3f}%'.format((raw_worker_train - pre_worker_train)/raw_worker_train*100))
```

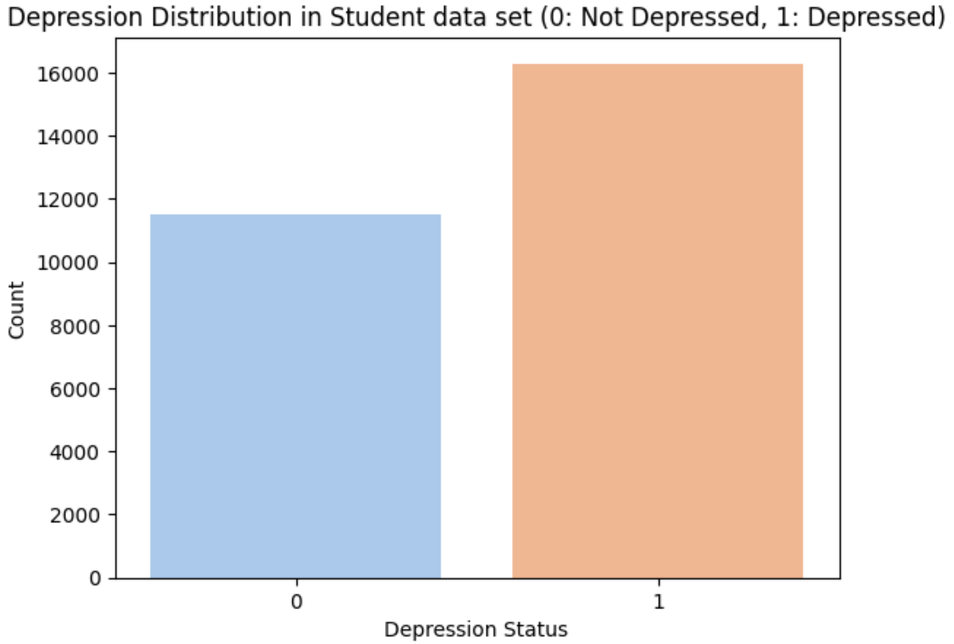
```
전처리 과정에서 삭제된 학생 훈련 데이터 비율 : 0.419%
전처리 과정에서 삭제된 직장인 훈련 데이터 비율 : 0.173%
```

시각화

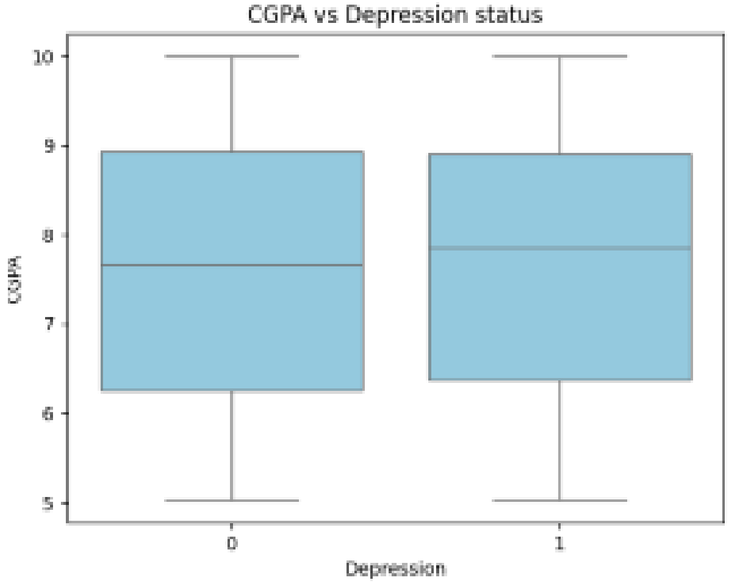
Student group

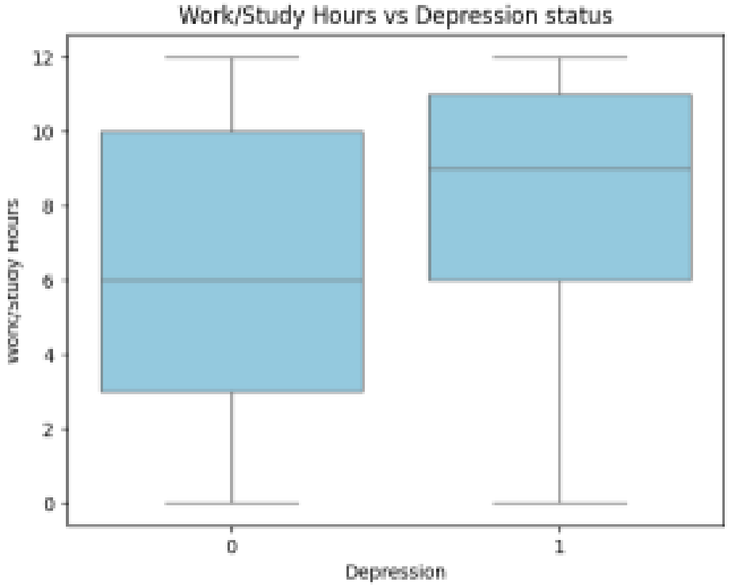
```
##STUDENT train data set##

sns.countplot(x='Depression', data=student_train, palette='pastel')
plt.title('Depression Distribution in Student data set (0: Not Depressed, 1: Depressed)')
plt.xlabel('Depression Status')
plt.ylabel('Count')
plt.show()
```

```
<ipython-input-137-8cafbbdc5984>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the

  sns.countplot(x='Depression', data=student_train, palette='pastel')
```



- 타깃변수(즉 Depression) 분포(우울증 vs 비우울증) .



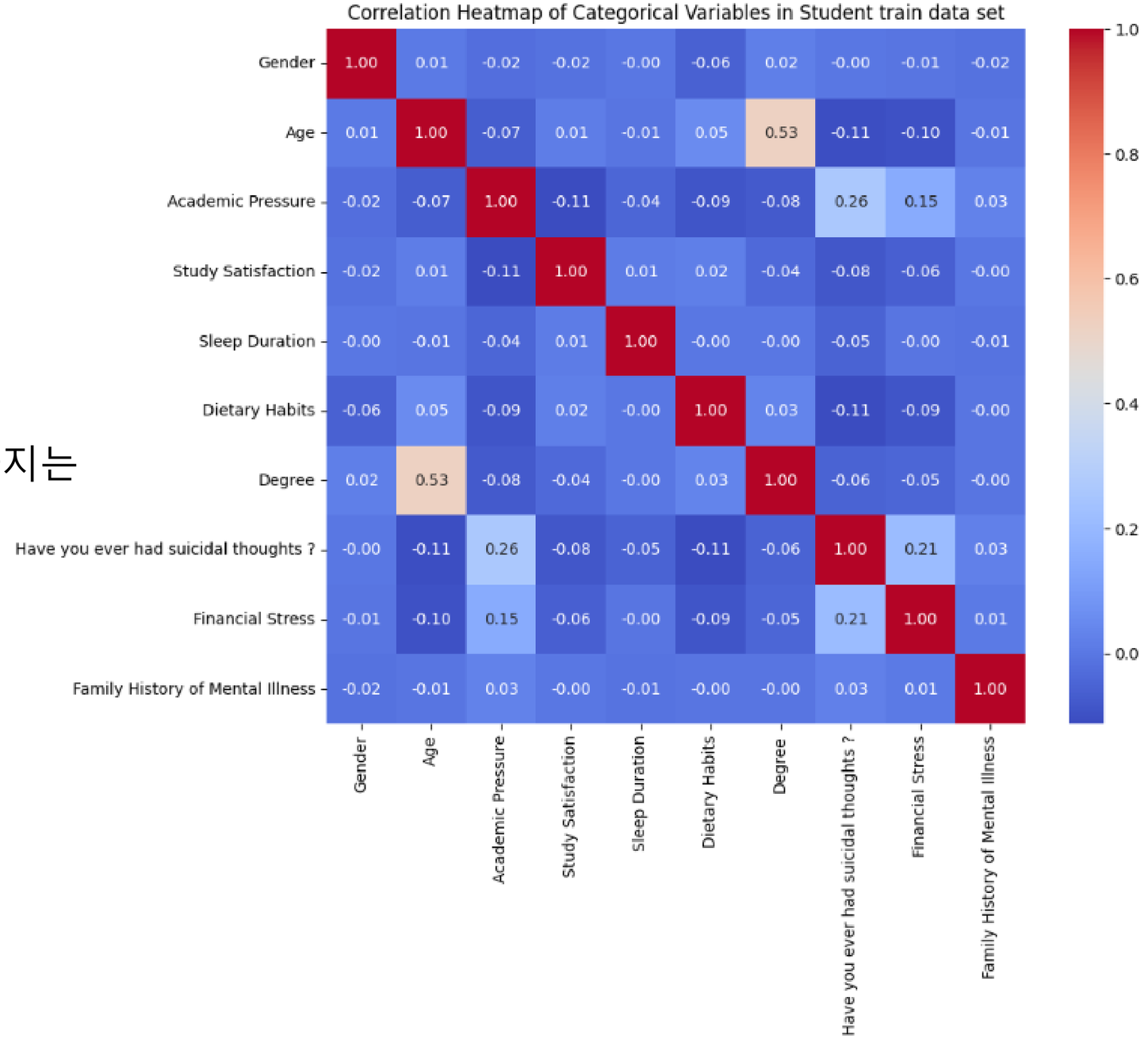


우울증 여부에 따른 수치형 변수의 분포

Student group

범주형 변수 간의 상관관계 히트맵

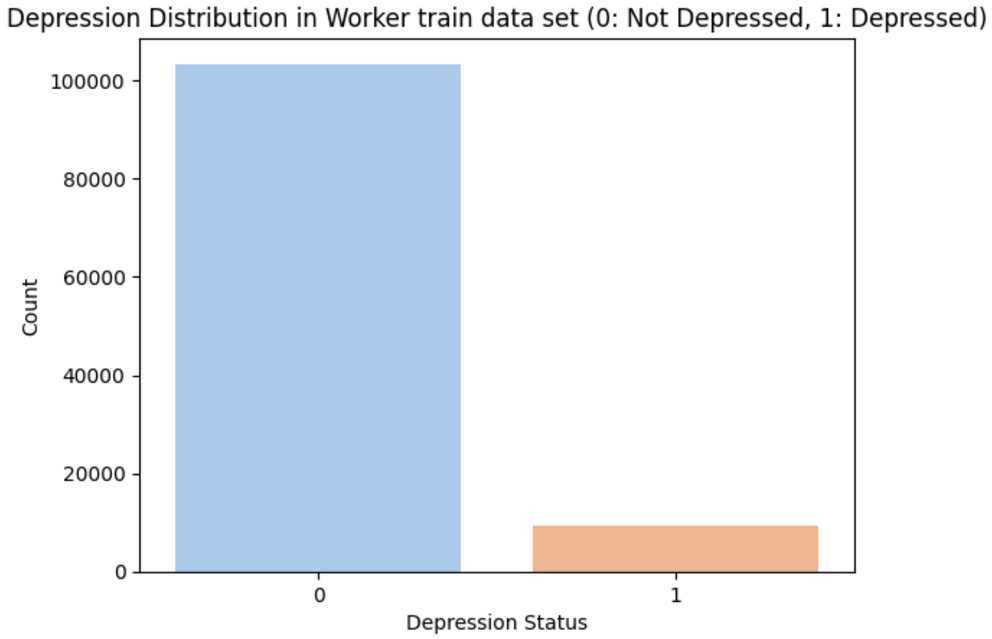
- 변수들 간의 상관관계가 높은 경우 다중공선성 문제가 발생할 수 있음.
- 그러나 눈에 띄게 높은 상관관계를 가지는 변수들은 없는 듯함



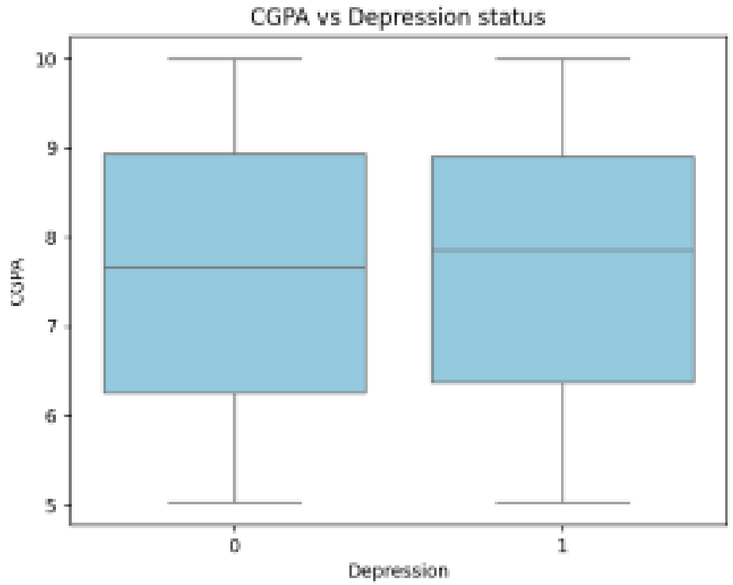
Worker group

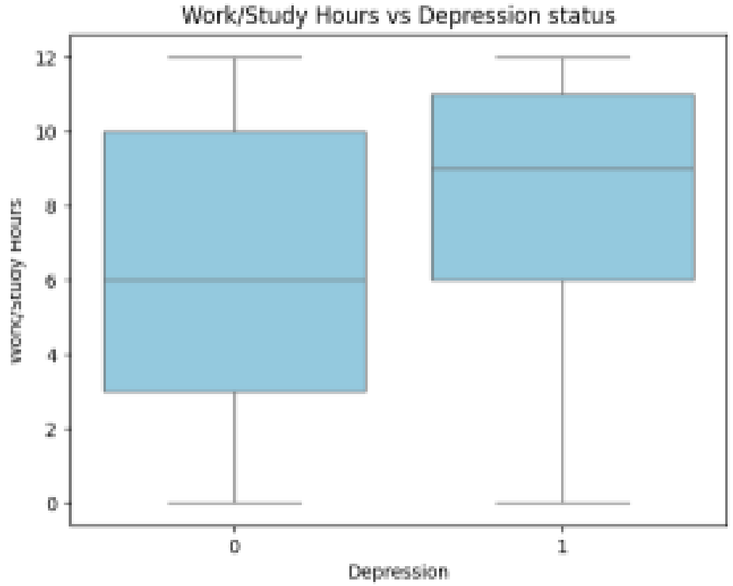
```
##WORKER train data set##

sns.countplot(x='Depression', data=worker_train, palette='pastel')
plt.title('Depression Distribution in Worker train data set (0: Not Depressed, 1: Depressed)')
plt.xlabel('Depression Status')
plt.ylabel('Count')
plt.show()

#worker train에서 클래스 불균형이 너무 심해보이는 문제 발생.
```

```
<ipython-input-138-aee378712717>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `:

  sns.countplot(x='Depression', data=worker_train, palette='pastel')
```



- 타깃변수(즉 Depression) 분포(우울증 vs 비우울증) .
직장인 traim 데이터에서 클래스 불균형이 너무 심해보이는 문제 발생



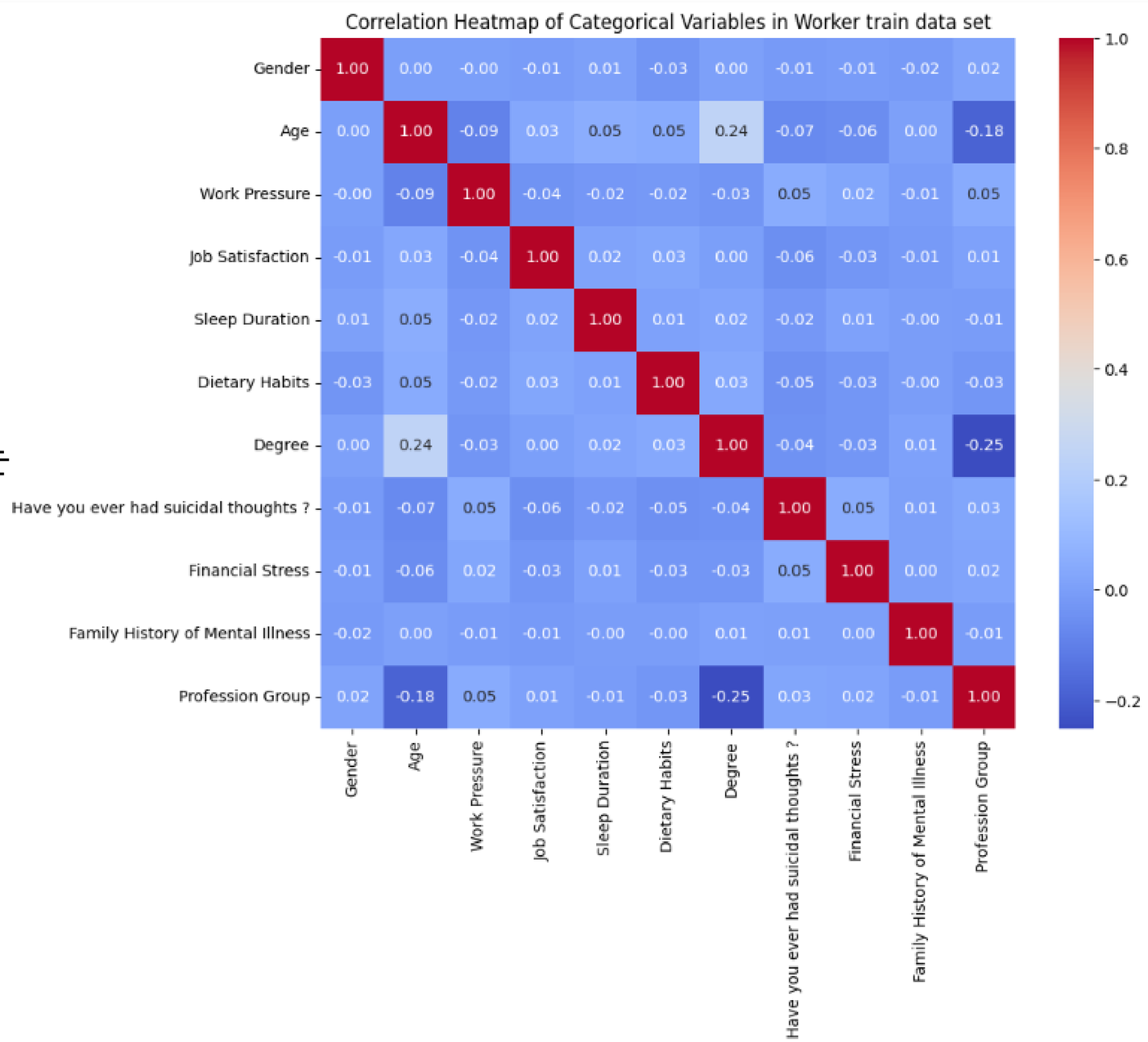


우울증 여부에 따른 수치형 변수의 분포

Worker group

범주형 변수 간의 상관관계 히트맵

- 변수들 간의 상관관계가 높은 경우 다중공선성 문제가 발생할 수 있음.
- 그러나 눈에 띄게 높은 상관관계를 가지는 변수들은 없는 듯함



K4

XGBoost 모델링 및 하이퍼 파라미터

4. 모델링

적절한 모델 탐색

1.LightGBM - 특징 : 대용량 데이터에서 빠르고 효율적으로 동작하는 트리 기반 부스팅 알고리즘, leaf-wise 트리 성장, GOSS, EFB 등으로 높은 예측 성능과 학습 속도를 제공. 범주형 변수와 결측치를 자동 처리하지만, 작은 데이터에서는 과적합에 주의

2. 랜덤 포레스트 (Random Forest) - 특징 : 트리 기반 모델로, 변수 중요도를 feature importances 속성으로 쉽게 확인 가능 - 장점 : 비선형 데이터도 잘 처리하고, 이상치에 강한 특징이 있음. 의학 데이터에 자주 사용됨

3. XGBoost - 특징 : 부스팅 알고리즘을 사용하여 변수 중요도를 시각화하고 해석하기 유리 - 장점 : 변수 상호작용이 있는 복잡한 데이터에도 강함.

4. 하이퍼 파라미터 튜닝

셋중 가장 정확도가 높았던 XGBoost 채택

```python
# 학습
xg_model1 = XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=5, random_state=42, eval_metric='logloss')
xg_model1.fit(X_worker_train, y_worker_train)

xg_model2 = XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=5, random_state=42, eval_metric='logloss')
xg_model2.fit(X_student_train, y_student_train)

# 성능 평가
y_worker_pred = xg_model1.predict(X_worker_val)
y_student_pred = xg_model2.predict(X_student_val)

worker_acc = accuracy_score(y_worker_val, y_worker_pred)
student_acc = accuracy_score(y_student_val, y_student_pred)

y_worker_proba = xg_model1.predict_proba(X_worker_val)[:, 1]
worker_roc_auc = roc_auc_score(y_worker_val, y_worker_proba)
y_student_proba = xg_model2.predict_proba(X_student_val)[:, 1]
student_roc_auc = roc_auc_score(y_student_val, y_student_proba)
```

```
Worker Classification Report :
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     20678
           1       0.80      0.66      0.73      1843

    accuracy                           0.96     22521
   macro avg       0.89      0.82      0.85     22521
weighted avg       0.96      0.96      0.96     22521

Student Classification Report :
              precision    recall  f1-score   support

           0       0.83      0.79      0.81      2300
           1       0.86      0.88      0.87      3257

    accuracy                           0.84      5557
   macro avg       0.84      0.84      0.84      5557
weighted avg       0.84      0.84      0.84      5557

Worker Accuracy :  0.9589716264819502

Student Accuracy :  0.8448803311139104

Worker ROC AUC Score :  0.9658386057207596

Student ROC AUC Score :  0.9158405307631724
```

정확도 : 0.93484

4. 모델링

정확도 향상을 위해 베이지안 최적화 이용

```python
# 베이지안 최적화를 위한 목표 함수 정의
import optuna
def objective_worker(trial):
  param = {
      'n_estimators' : trial.suggest_int('n_estimators',50,300),
      'max_depth' : trial.suggest_int('max_depth',3,10),
      'learning_rate' : trial.suggest_float('learning_rate',0.01,0.3),
      'subsample': trial.suggest_float('subsample', 0.5, 1.0),
      'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1.0),
      'gamma': trial.suggest_float('gamma', 0, 5),
      'reg_alpha': trial.suggest_float('reg_alpha', 0.0, 5.0),
      'reg_lambda': trial.suggest_float('reg_lambda', 0.0, 5.0),
      'use_label_encoder': False,
      'eval_metric': 'logloss',
      'random_state': 42
  }
  model = XGBClassifier(**param)

  model.fit(X_worker_train, y_worker_train)
  y_pred_proba = model.predict_proba(X_worker_val)[:,1]

  score = roc_auc_score(y_worker_val, y_pred_proba)
  return score
```

```python
# 베이지안 최적화로 구한 파라미터를 이용해 다시 학습
# auc가 불균형 데이터에 좋다고 해서 사용.
worker_best_params = study_worker.best_params
worker_best_params['use_label_encoder'] = False
worker_best_params['eval_metric'] = 'auc'
worker_best_params['random_state'] = 42

xg_model_worker = XGBClassifier(**worker_best_params)
xg_model_worker.fit(Worker_X_train_encoded, Worker_y_train)

student_best_params = study_worker.best_params
student_best_params['use_label_encoder'] = False
student_best_params['eval_metric'] = 'auc'
student_best_params['random_state'] = 42

xg_model_student = XGBClassifier(**student_best_params)
xg_model_student.fit(Student_X_train_encoded, Student_y_train)
```

K5

성능평가

5. 성능 평가

```
# test 데이터로 예측
y_pred_worker = xg_model_worker.predict(worker_y_test_encoded)
y_pred_student = xg_model_student.predict(student_y_test_encoded)
```

```
worker_result = pd.DataFrame({
    'id' : worker_test['id'],
    'Depression' : y_pred_worker
}, index = worker_test.index)

student_result = pd.DataFrame({
    'id' : student_test['id'],
    'Depression' : y_pred_student
}, index = student_test.index)

combined_result = pd.concat([worker_result, student_result])
combined_result = combined_result.sort_index()

combined_result.to_csv('final_pred_xg.csv',index=False)
```

xgboost 모델링 결과

 # Private Score : 0.93787

 # Public Score : 0.94056

지금까지 발표를 들어주셔서

감사합니다