



# 데이터마이닝

이상민 교수님

## 선형회귀, 로지스틱회귀 분석

### 차례

#### 1. 일상생활 에서의 Odds 사례: 2028 미국 대선 2p

#### 2. 선형회귀분석, 로지스틱 회귀분석

##### - 선형회귀 2p

1. 데이터 선정 2p

2. 코드전문 3p

3. 코드 실행 결과 설명 8p

3.1 EDA & Feature Engineering & Data 전처리 9p

3.2 선형회귀 모델 구축 19p

3.3 선형회귀 모델 예측, 평가 20p

3.4 Shrinkage models (Lasso, Ridge, ElasticNet) 23p

##### - 로지스틱회귀 25p

1. 데이터 선정 25p

2. 코드전문 25p

3. 코드 실행 결과 설명 32p

3.1 EDA & Feature Engineering & Data 전처리 34p

3.2 로지스틱회귀 모델 구축 51p

3.3 로지스틱회귀 모델 예측, 평가 52p

3.4 Shrinkage models (Lasso, Ridge, ElasticNet) 53p

# 1. 일상생활 에서의 Odds 사례: 2028 미국 대선

US Presidential Election 2028 Winner

View all odds >

JD Vance	11/4	Gavin Newsom	8/1
Josh Shapiro	12/1	Michelle Obama	14/1

위 이미지는 메이저 베팅 사이트 <oddschecker>의 2028 미국 대선 배당 현황이다. 위 배당률을 통해서 각 후보가 선출될 가능성을 평가할 수 있다.

JD Vance의 배당률이 11/4 이므로, 그가 선거에서 선출될 Odds는 11:4 라는 의미이다. Odds는 떨어질 가능성 11와 선출될 가능성 4의 비율로 목록의 후보 중에서 가장 승리 확률이 높다고 볼 수 있다.

Michelle Obama가 Odds가 14:1로 가장 낮은 승률을 보이고 있다.

JD Vance의 승리에 베팅할 경우, 4달러를 베팅해서 맞추면 11달러의 수익을 얻을 수 있다.

가장 승리 확률이 낮은 Michelle Obama의 경우, 1달러를 베팅해 맞으면 14달러의 수익을 얻는다.

위의 사례와 같이 배당률은 예상하는 확률과 보상의 관계를 직관적으로 보여준다.

## 2. 선형회귀분석, 로지스틱 회귀분석

### <선형회귀>

#### 1. 데이터 선정

선정 데이터셋: Apartment for Rent Classified



Apartment for Rent Classified

Donated on 12/25/2019

This is a dataset of classified for apartments for rent in USA.

Dataset Characteristics

Multivariate

Subject Area

Business

Associated Tasks

Classification, Regression, Clustering

Feature Type

Categorical, Integer

# Instances

10000

# Features

21

<https://archive.ics.uci.edu/dataset/555/apartment+for+rent+classified>

## 2. 코드전문

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
apartments_df = pd.read_csv('apartments_for_rent_classified_100K.csv', sep=';', encoding='ISO-8859-1')
```

### 1. EDA, 전처리, Feature Engineering

```
apartments_df
```

```
apartments_df.info()
```

Null이 존재하는 피처들이 있음. 추가적인 feature를 만든 이후에 일괄적으로 제거

```
apartments_df["currency"].value_counts()
```

currency는 모두 일괄적으로 USD이므로 지워도 무방

```
apartments_df["id"].value_counts()
```

같은 아이디의 중복값이 존재

```
apartments_df[apartments_df["id"] == 5508802531]
# 중복된 아이디의 데이터 확인
```

```
apartments_df = apartments_df.drop_duplicates(subset="id", keep="first")
```

중복된 아이디의 데이터 처리

```
apartments_df
```

```
apartments_df['fee'].value_counts(dropna=False)
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=apartments_df, x='fee', y='price')
plt.title("Score Distribution by age")
plt.show()
```

fee는 유의미한 피처로 보이지 않음

```
apartments_df['category'].value_counts()
```

apartment가 대부분

```
apartments_df['currency'].value_counts()
```

USD로 통일하여 예측에 의미 없음

```
apartments_df = apartments_df.drop(["amenities", "category", "title", "body", "currency", "has_photo", "latitude", "longitude",
                                     "source", "address", "time", 'price_display', 'price_type', 'id', 'fee'], axis=1)
```

쓸모없는 object 피처는 제거

```
apartments_df['pets_allowed'].value_counts()
```

```
apartments_df['pets_allowed'] = np.where(apartments_df['pets_allowed'].isna(), 0, 1)
```

동물이 허가된 경우는 1로 하고 그렇지 않으면 0으로 맵핑

```
apartments_df
```

```
apartments_df['cityname'].value_counts()
```

도시별로 분명 집값이 다르기 때문에 이를 반영할 도시별 평균 집값과 그 분산을 새로운 feature로 추가

```
# 각 도시 이름의 개수 계산
city_counts = apartments_df['cityname'].value_counts()

# 10번 미만으로 등장한 도시는 other로 분류
apartments_df['cityname'] = apartments_df['cityname'].where(apartments_df['cityname'].map(city_counts) > 10, 'other')

apartments_df['cityname'].value_counts()
```

6281개의 데이터가 'other'로 분류되었음

```
# 각 도시 이름별로 price의 평균, 분산을 계산하여 새로운 피처에 추가
apartments_df['average_price_by_city'] = apartments_df.groupby('cityname')['price'].transform('mean')

apartments_df['var_price_by_city'] = apartments_df.groupby('cityname')['price'].transform('var')

apartments_df['state'].value_counts()
```

```
# 각 주의 개수 계산
state_counts = apartments_df['cityname'].value_counts()

# 10번 미만으로 등장한 주는 other로 분류
apartments_df['state'] = apartments_df['state'].where(apartments_df['state'].map(state_counts) > 10, 'other')
```

```
# 각 주별로 price의 평균과 분산을 계산하여 새로운 피처에 추가
apartments_df['average_price_by_state'] = apartments_df.groupby('state')['price'].transform('mean')

apartments_df['var_price_by_state'] = apartments_df.groupby('state')['price'].transform('var')

apartments_df = apartments_df.drop(["cityname"], axis=1)
```

```
# bathrooms와 bedrooms을 합쳐서 total_rooms 피처 생성
apartments_df['total_rooms'] = apartments_df['bathrooms'] + apartments_df['bedrooms']

# 방당 평균 면적을 나타내는 average_area_per_room 피처 생성
apartments_df['average_area_per_room'] = apartments_df['square_feet'] / apartments_df['total_rooms']

apartments_df['average_area_per_room'].describe()
```

```
plt.scatter(apartments_df['average_area_per_room'], apartments_df['price'], alpha=0.5)
plt.xlabel('Average Area Per Room')
plt.ylabel('Price')
plt.title('Relationship between Average Area Per Room and Price')
plt.show()
```

```
apartments_df['total_rooms'].describe()
```

```
plt.scatter(apartments_df['total_rooms'], apartments_df['price'], alpha=0.5)
plt.xlabel('total_rooms')
plt.ylabel('Price')
plt.title('Relationship between total_rooms and Price')
plt.show()
```

'average\_area\_per\_room' 과 'total\_rooms' 모두 price와의 산점도상으로는 그다지 유의미해보이지는 않음

```
apartments_df

print(apartments_df.isna().sum())

apartments_df = apartments_df.dropna()

print(apartments_df.isna().sum())
```

0값이 있어서 생기는 요소는 모두 데이터째로 제거

```
apartments_df['price'].describe()

apartments_df['price'].hist()
```

집값이 비싼 이상치가 많음

```
# 이상치 제거
lower_bound = apartments_df['price'].quantile(0.1)
upper_bound = apartments_df['price'].quantile(0.85)

apartments_df = apartments_df[(apartments_df['price'] >= lower_bound) & (apartments_df['price'] <= upper_bound)]

apartments_df['price'].describe()

apartments_df['price'].hist()
```

```
# 각 도시 이름의 개수 계산
city_counts = apartments_df['cityname'].value_counts()

# 10번 미만으로 등장한 도시는 other로 분류
apartments_df['cityname'] = apartments_df['cityname'].where(apartments_df['cityname'].map(city_counts) > 10, 'other')

apartments_df['cityname'].value_counts()
```

6281개의 데이터가 'other'로 분류되었음

```
# 각 도시 이름별로 price의 평균, 분산을 계산하여 새로운 피처에 추가
apartments_df['average_price_by_city'] = apartments_df.groupby('cityname')['price'].transform('mean')

apartments_df['var_price_by_city'] = apartments_df.groupby('cityname')['price'].transform('var')

apartments_df['state'].value_counts()
```

```
# 각 주의 개수 계산
state_counts = apartments_df['cityname'].value_counts()

# 10번 미만으로 등장한 주는 other로 분류
apartments_df['state'] = apartments_df['state'].where(apartments_df['state'].map(state_counts) > 10, 'other')
```

```
# 각 주별로 price의 평균과 분산을 계산하여 새로운 피처에 추가
apartments_df['average_price_by_state'] = apartments_df.groupby('state')['price'].transform('mean')

apartments_df['var_price_by_state'] = apartments_df.groupby('state')['price'].transform('var')

apartments_df = apartments_df.drop(["cityname"], axis=1)
```

```
# bathrooms와 bedrooms을 합쳐서 total_rooms 피처 생성
apartments_df['total_rooms'] = apartments_df['bathrooms'] + apartments_df['bedrooms']

# 방당 평균 면적을 나타내는 average_area_per_room 피처 생성
apartments_df['average_area_per_room'] = apartments_df['square_feet'] / apartments_df['total_rooms']

apartments_df['average_area_per_room'].describe()
```

```
plt.scatter(apartments_df['average_area_per_room'], apartments_df['price'], alpha=0.5)
plt.xlabel('Average Area Per Room')
plt.ylabel('Price')
plt.title('Relationship between Average Area Per Room and Price')
plt.show()
```

```
apartments_df['total_rooms'].describe()
```

```
plt.scatter(apartments_df['total_rooms'], apartments_df['price'], alpha=0.5)
plt.xlabel('total_rooms')
plt.ylabel('Price')
plt.title('Relationship between total_rooms and Price')
plt.show()
```

'average\_area\_per\_room' 과 'total\_rooms' 모두 price와의 산점도상으로는 그다지 유의미해보이지는 않음

```
apartments_df

print(apartments_df.isna().sum())

apartments_df = apartments_df.dropna()

print(apartments_df.isna().sum())
```

0값이 있어서 생기는 요소는 모두 데이터째로 제거

```
apartments_df['price'].describe()

apartments_df['price'].hist()
```

집값이 비싼 이상치가 많음

```
# 이상치 제거
lower_bound = apartments_df['price'].quantile(0.1)
upper_bound = apartments_df['price'].quantile(0.85)

apartments_df = apartments_df[(apartments_df['price'] >= lower_bound) & (apartments_df['price'] <= upper_bound)]

apartments_df['price'].describe()

apartments_df['price'].hist()
```

## Feature Engineering 요약

id - 제거  
category - 제거  
title - 제거  
body - 제거  
amenities - 제거  
bathrooms - 사용  
bedrooms - 사용 (방의 갯수를 더해 총 방의 갯수 사용)  
currency - 제거  
fee - 제거  
has\_photo - 제거  
pets\_allowed - 이진화하여 사용  
price\_display - 제거  
price\_type - 제거  
square\_feet - 사용 (방의 평균면적 추가)  
address - 제거  
cityname - 도시별 평균 집값과 분산 추가  
state - 주별 평균 집값과 분산 추가  
latitude - 제거  
longitude - 제거  
source - 제거  
time - 제거

## 선형회귀 모델 구축, 예측, 평가

```
import statsmodels.api as sm
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

X = apartments_df[['var_price_by_city', 'pets_allowed', 'total_rooms', 'square_feet', 'average_price_by_city', 'average_price_by_state']]
y = apartments_df['price']

# 데이터 준비 및 스케일링
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Statsmodels를 사용하여 p-value와 adj. R2, F-test 등의 정보 확인
X_train_const = sm.add_constant(X_train_scaled) # 절편 추가
model_sm = sm.OLS(y_train, X_train_const).fit()

# 모델 요약
print(model_sm.summary())

# 예측 수행 (log 스케일)
y_pred = model_sm.predict(sm.add_constant(X_test_scaled))

# 모델 평가
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print("Mean Squared Error (original scale):", mse)
print("Mean Absolute Error (original scale):", mae)

# y 예측값과 잔차간의 관계
plt.scatter(model_sm.fittedvalues, model_sm.resid)

import scipy.stats as stats
import pylab
# QQ plot 기반의 데이터 정상성 확인
stats.probplot(model_sm.resid, dist='norm', plot=pylab)

# VIF Score 계산
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X_scaled, i) for i in range(X_scaled.shape[1])]
print(vif_data)
```

## Lasso, Ridge, Elasticnet

```
from sklearn.linear_model import Ridge, ElasticNet
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import Lasso

# Lasso
lasso = Lasso(alpha=1, random_state=42, max_iter=100000)
lasso.fit(X_train_scaled, y_train)
y_pred = lasso.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Lasso")
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared ( $R^2$ ):", r2)
print("\n")

# Ridge
ridge = Ridge(alpha=1, random_state=42, max_iter=100000)
ridge.fit(X_train_scaled, y_train)
y_pred_ridge = ridge.predict(X_test_scaled)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print("Ridge")
print("Mean Squared Error (MSE):", mse_ridge)
print("Mean Absolute Error (MAE):", mae_ridge)
print("R-squared ( $R^2$ ):", r2_ridge)
print("\n")

# ElasticNet
elasticnet = ElasticNet(alpha=1, l1_ratio=0.5, random_state=42, max_iter=100000)
elasticnet.fit(X_train_scaled, y_train)
y_pred_elasticnet = elasticnet.predict(X_test_scaled)

mse_elasticnet = mean_squared_error(y_test, y_pred_elasticnet)
mae_elasticnet = mean_absolute_error(y_test, y_pred_elasticnet)
r2_elasticnet = r2_score(y_test, y_pred_elasticnet)
print("ElasticNet")
print("Mean Squared Error (MSE):", mse_elasticnet)
print("Mean Absolute Error (MAE):", mae_elasticnet)
print("R-squared ( $R^2$ ):", r2_elasticnet)
```

### 3. 코드 실행 결과 설명

#### - 데이터 구성 feature 개요

---

id - 아파트의 고유 식별자 (형식: int64)

category - 아파트 분류 카테고리 (형식: object)

title - 아파트 제목 텍스트 (형식: object)

body - 아파트 본문 텍스트 (형식: object)

amenities - 아파트 편의 시설 목록 (AC, 농구장, 케이블, 체육관, 인터넷, 수영장, 냉장고 등, 형식: object)

bathrooms - 욕실 개수 (형식: float64)

bedrooms - 침실 개수 (형식: float64)

currency - 가격 통화 (형식: object)

fee - 수수료 정보 (형식: object)

has\_photo - 아파트 사진 유무 (형식: object)

pets\_allowed - 허용되는 반려동물 (예: 개/고양이 등, 형식: object)

price - 아파트 임대 가격 (형식: float64)

price\_display - 사용자를 위한 가격 표시 형식 (형식: object)

price\_type - USD 기준 가격 여부 (형식: object)

square\_feet - 아파트 크기 (평방피트 단위, 형식: int64)

address - 아파트 주소 (형식: object)

cityname - 아파트 위치 도시명 (형식: object)

state - 아파트 위치 주명 (형식: object)

latitude - 아파트 위치 위도 (형식: float64)

longitude - 아파트 위치 경도 (형식: float64)

source - 분류 출처 (형식: object)

time - 분류 생성 시각 (형식: int64)

---

#### - 분석목표: 다른 Feature들을 이용해 price값을 예측



### 3.1 EDA & Feature Engineering & Data 전처리

```
[3]: apartments_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99492 entries, 0 to 99491
Data columns (total 22 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id                    99492 non-null  int64
1   category              99492 non-null  object
2   title                 99492 non-null  object
3   body                  99492 non-null  object
4   amenities             83448 non-null  object
5   bathrooms             99429 non-null  float64
6   bedrooms              99368 non-null  float64
7   currency              99492 non-null  object
8   fee                   99492 non-null  object
9   has_photo             99492 non-null  object
10  pets_allowed          39068 non-null  object
11  price                 99491 non-null  float64
12  price_display         99491 non-null  object
13  price_type            99492 non-null  object
14  square_feet           99492 non-null  int64
15  address               7943 non-null   object
16  cityname              99190 non-null  object
17  state                 99190 non-null  object
18  latitude              99467 non-null  float64
19  longitude             99467 non-null  float64
20  source                99492 non-null  object
21  time                  99492 non-null  int64
```

Null이 존재하는 피쳐들이 있음. 추가적인 feature를 만든 이후에 일괄적으로 제거

```
[6]: apartments_df["id"].value_counts()

[6]: id
5508802531    2
5197864265    2
5197858885    2
5197859052    2
5197859695    2
..
5508930359    1
5508930563    1
5508930632    1
5508934954    1
5121218844    1
Name: count, Length: 99408, dtype: int64
```

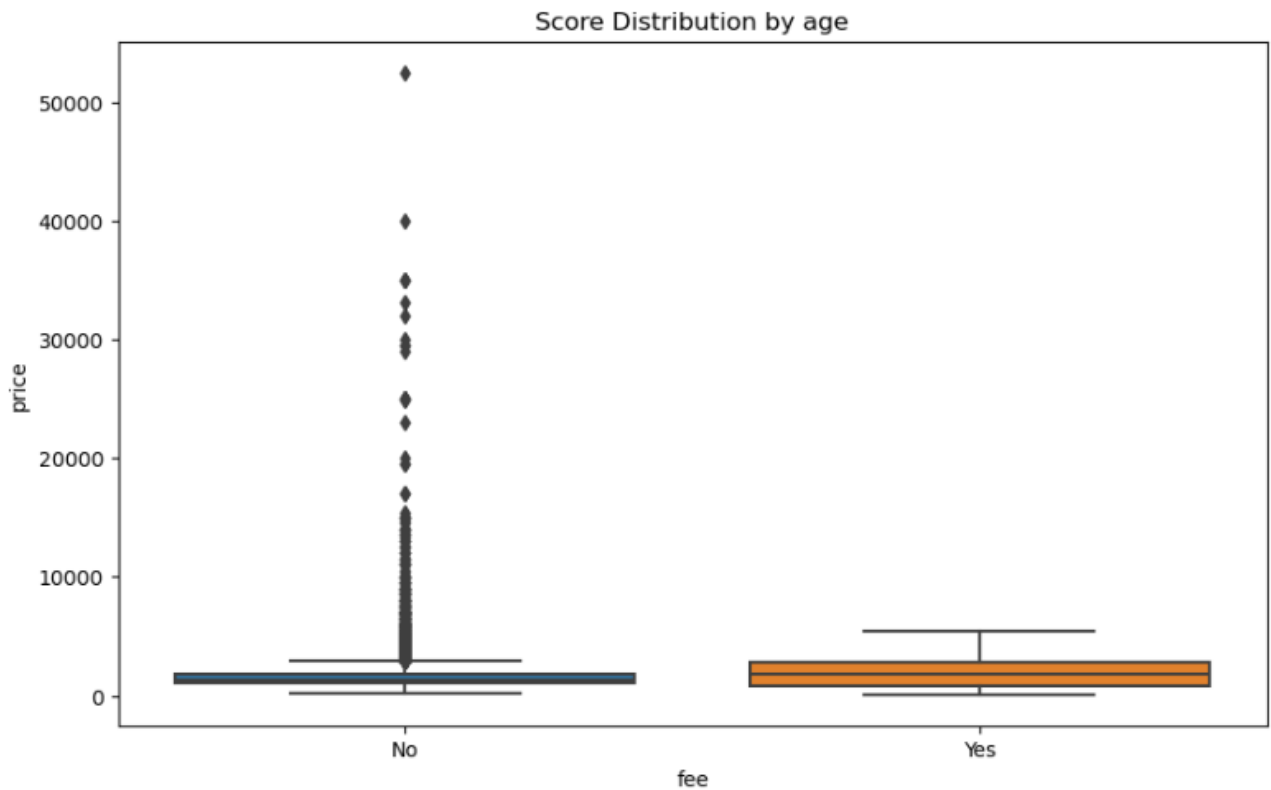
```
[8]: apartments_df = apartments_df.drop_duplicates(subset="id", keep="first")
```

id에 중복 값이 존재하여 중복되는 아이디는 제거

```
[10]: apartments_df['fee'].value_counts(dropna=False)
```

```
[10]: fee
No      99207
Yes       201
Name: count, dtype: int64
```

```
[57]: plt.figure(figsize=(10, 6))
sns.boxplot(data=apartments_df, x='fee', y='price')
plt.title("Score Distribution by age")
plt.show()
```



fee가 yes인 경우 자체가 그다지 많지 않아서 위 분포가 그다지 유의미하다고 볼 수는 없어 보임

```
: apartments_df = apartments_df.drop(["amenities", "category", "title", "body", "currency", "has_photo"], axis=1)
```

쓸모없는 object 피쳐는 제거

```
: apartments_df['category'].value_counts()
```

```
: category
housing/rent/apartment      99347
housing/rent/commercial/retail    42
housing/rent                7
housing/rent/home            4
housing/rent/short_term       4
housing/rent/condo            3
housing/rent/other            1
Name: count, dtype: int64
```

Category는 apartment에 쏠려 있어 나머지는 모델의 선형성에 오히려 악영향을 줄 수 있음

Title, body, amenities에는 사용불가능한 설명만 들어있음

```
[60]: apartments_df['currency'].value_counts()
```

```
[60]: currency
      USD    99408
      Name: count, dtype: int64
```

Currency는 모두 USD로 동일하여 예측에 의미 없음

이외, price\_display, price\_type, address, latitude, longitude, source, time 등의 feature는 제거

```
[19]: apartments_df['pets_allowed'].value_counts()
```

```
[19]: pets_allowed
      Cats,Dogs    37066
      Cats        1840
      Dogs         127
      Cats,Dogs,None    1
      Name: count, dtype: int64
```

```
[20]: apartments_df['pets_allowed'] = np.where(apartments_df['pets_allowed'].isna(), 0, 1)
```

동물이 허가된 경우는 1로 하고 그렇지 않으면 0으로 맵핑

동물이 허가된 경우 집값에 영향이 있을 수 있음. NaN값은 0으로 하고 이외에는 모두 1로 맵핑

```
[22]: apartments_df['cityname'].value_counts()
```

```
[22]: cityname
      Dallas    2856
      Denver    2750
      Los Angeles 2429
      Las Vegas  1846
      Arlington 1577
      ...
      Flomaton    1
      Murphysboro 1
      Highwood    1
      Del Mar     1
      Willow Grove 1
      Name: count, Length: 2979, dtype: int64
```

```
*[23]: # 각 도시 이름의 개수 계산
      city_counts = apartments_df['cityname'].value_counts()

      # 10번 미만으로 등장한 도시는 other로 분류
      apartments_df['cityname'] = apartments_df['cityname'].where(apartments_df['cityname'].map(city_counts) > 10, 'other')
```

```
[24]: apartments_df['cityname'].value_counts()
```

```
[24]: cityname
      other    6281
      Dallas    2856
      Denver    2750
      Los Angeles 2429
      Las Vegas  1846
      ...
      Sewickley    11
      Rural Hall   11
      Pittsfield   11
      Hopewell     11
      San Mateo    11
      Name: count, Length: 1004, dtype: int64
```

6281개의 데이터가 'other'로 분류되었음

```
[22]: apartments_df['cityname'].value_counts()

[22]: cityname
Dallas      2856
Denver      2750
Los Angeles 2429
Las Vegas   1846
Arlington   1577
...
Flomaton     1
Murphysboro  1
Highwood     1
Del Mar       1
Willow Grove 1
Name: count, Length: 2979, dtype: int64
```

•[23]: # 각 도시 이름의 개수 계산  
city\_counts = apartments\_df['cityname'].value\_counts()  
  
# 10번 미만으로 등장한 도시는 other로 분류  
apartments\_df['cityname'] = apartments\_df['cityname'].where(apartments\_df['cityname'].map(city\_counts) > 10, 'other')

```
[24]: apartments_df['cityname'].value_counts()

[24]: cityname
other      6281
Dallas     2856
Denver     2750
Los Angeles 2429
Las Vegas  1846
...
Sewickley  11
Rural Hall 11
Pittsfield 11
Hopewell   11
San Mateo  11
Name: count, Length: 1004, dtype: int64
```

6281개의 데이터가 'other'로 분류되었음

도시별로 집값이 다르다. 이를 one-hot encoding 방식으로 접근하여 feature수를 늘리기 보다는 그 도시의 평균 집값과 집값의 분산을 새로운 feature로 추가하는 것이 합리적이라 판단. 이때, 너무 데이터 수가 적은 도시는 평균을 신뢰하기 어렵기 때문에 10개의 데이터 이하의 도시는 other로 분류.

분류결과 6281개의 데이터가 'other'로 분류되었음

```
# 각 도시 이름별로 price의 평균, 분산을 계산하여 새로운 피처에 추가
apartments_df['average_price_by_city'] = apartments_df.groupby('cityname')['price'].transform('mean')

apartments_df['var_price_by_city'] = apartments_df.groupby('cityname')['price'].transform('var')
```

각 도시 이름별로 price의 평균과 분산을 계산하여 새로운 feature로 만든다.

```
[23]: # 각 도시 이름별로 price의 평균, 분산을 계산하여 새로운 피처에 추가
apartments_df['average_price_by_city'] = apartments_df.groupby('cityname')['price'].transform('mean')

apartments_df['var_price_by_city'] = apartments_df.groupby('cityname')['price'].transform('var')
```

```
[24]: apartments_df['state'].value_counts()
```

```
[24]: state
TX      11250
CA      10301
VA       8278
NC       6293
CO       6279
FL       5773
MD       5276
MA       5023
OH       4899
GA       4750
NJ       4444
NV       2813
WA       2595
AZ       2376
LA       1345
MO       1203
PA       1122
TN       1114
IL       1036
NE       1020
KY        995
OK        934
SC        908
KS        899
UT        809
ND        743
NH        735
MI        710
NY        659
AR        598
MN        581
CT        509
IN        509
WI        430
IA        372
AL        354
OR        277
VT        125
RI        119
MS        107
ID         96
DC         93
MT         87
SD         86
AK         58
ME         32
HI         31
NM         24
WY         16
WV         13
DE          7
Name: count, dtype: int64
```

```
[25]: # 각 주의 개수 계산
state_counts = apartments_df['cityname'].value_counts()

# 10번 미만으로 등장한 주는 other로 분류
apartments_df['state'] = apartments_df['state'].where(apartments_df['state'].map(state_counts) > 10, 'other')
```

```
•[26]: # 각 주별로 price의 평균과 분산을 계산하여 새로운 피처에 추가
apartments_df['average_price_by_state'] = apartments_df.groupby('state')['price'].transform('mean')

apartments_df['var_price_by_state'] = apartments_df.groupby('state')['price'].transform('var')
```

마찬가지로 주별로 집값의 분포가 다르다. 대상 주의 평균 집값과 집값의 분산을 새로운 feature로 추가하는 것이 합리적이라 판단. 이때, 너무 데이터 수가 적은 주는 평균을 신뢰하기 어렵기 때문에 10개의 데이터 이하의 주는 other로 분류.

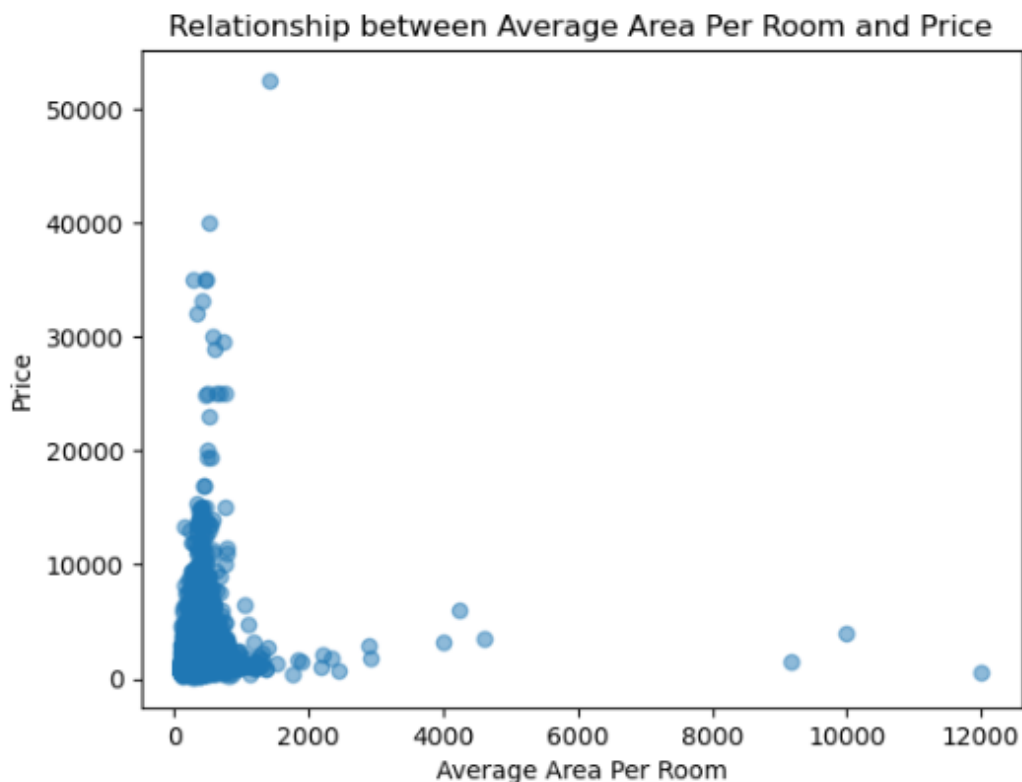
```
[29]: # bathrooms와 bedrooms를 합쳐서 total_rooms 피쳐 생성
apartments_df['total_rooms'] = apartments_df['bathrooms'] + apartments_df['bedrooms']

# 방당 평균 면적을 나타내는 average_area_per_room 피쳐 생성
apartments_df['average_area_per_room'] = apartments_df['square_feet'] / apartments_df['total_rooms']
```

```
[30]: apartments_df['average_area_per_room'].describe()
```

```
[30]: count    99222.000000
      mean     314.287742
      std     100.124271
      min      100.000000
      25%     262.500000
      50%     300.000000
      75%     361.500000
      max    12000.000000
      Name: average_area_per_room, dtype: float64
```

```
[31]: plt.scatter(apartments_df['average_area_per_room'], apartments_df['price'], alpha=0.5)
plt.xlabel('Average Area Per Room')
plt.ylabel('Price')
plt.title('Relationship between Average Area Per Room and Price')
plt.show()
```



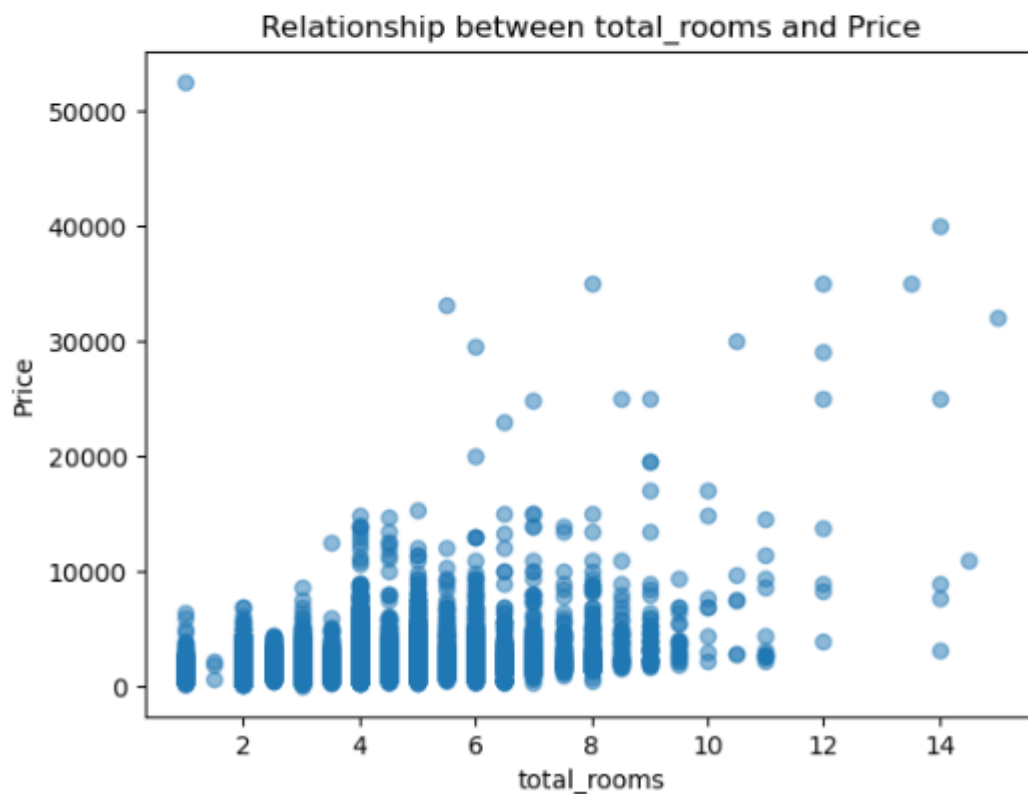
방이 클수록 집값의 예측에 긍정적인 영향을 줄 것으로 예상되어 집의 면적을 화장실 + 침실을 더한 총 방의 개수를 나누어 그 집의 방의 평균 면적을 새로운 feature로 추가.

그러나 산점도를 확인한 결과 선형적인 관계가 있다고 보기는 어려워보임

```
[32]: apartments_df['total_rooms'].describe()
```

```
[32]: count    99222.000000  
      mean      3.174215  
      std      1.190566  
      min      1.000000  
      25%      2.000000  
      50%      3.000000  
      75%      4.000000  
      max     15.000000  
      Name: total_rooms, dtype: float64
```

```
[33]: plt.scatter(apartments_df['total_rooms'], apartments_df['price'], alpha=0.5)  
      plt.xlabel('total_rooms')  
      plt.ylabel('Price')  
      plt.title('Relationship between total_rooms and Price')  
      plt.show()
```



방의 개수가 많으면 집값도 올라갈 것이라고 생각하였으나 산점도를 확인해보면 선형적인 경향성이 있다고 보기는 어려움. 방의 개수가 0인 것은 제외할 필요성이 있어 보임.

```
[33]: print(apartments_df.isna().sum())
```

```
bathrooms      63
bedrooms       124
pets_allowed     0
price            1
square_feet      0
state          302
average_price_by_city  0
var_price_by_city  0
average_price_by_state 302
var_price_by_state 302
total_rooms     186
average_area_per_room 186
dtype: int64
```

```
[34]: apartments_df = apartments_df.dropna()
```

```
[35]: print(apartments_df.isna().sum())
```

```
bathrooms      0
bedrooms       0
pets_allowed     0
price            0
square_feet      0
state           0
average_price_by_city  0
var_price_by_city  0
average_price_by_state 0
var_price_by_state 0
total_rooms     0
average_area_per_room 0
dtype: int64
```

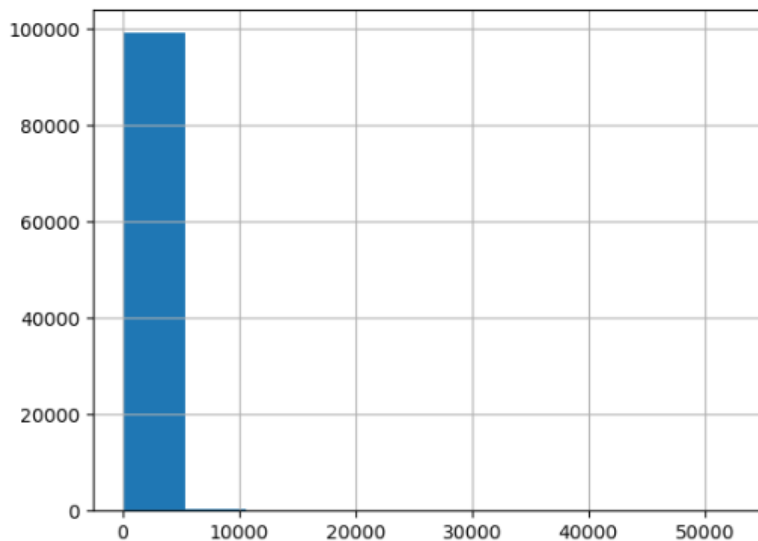
null값, 방이 0인 경우는 일괄적으로 제거

```
[37]: apartments_df['price'].describe()
```

```
[37]: count    98923.000000
      mean     1525.664213
      std      899.079201
      min       100.000000
      25%     1014.000000
      50%     1350.000000
      75%     1795.000000
      max     52500.000000
      Name: price, dtype: float64
```

```
[34]: apartments_df['price'].hist()
```

```
[34]: <Axes: >
```



예측대상인 price는 평균 1525달러의 변수로 이상치가 극심하여 선형예측에 부정적 영향을 줄 수 있음. 따라서 범위를 지정하여 최대한 price의 분포를 정규분포에 가깝도록 조정



```
[35]: # 이상치 제거
lower_bound = apartments_df['price'].quantile(0.1)
upper_bound = apartments_df['price'].quantile(0.85)

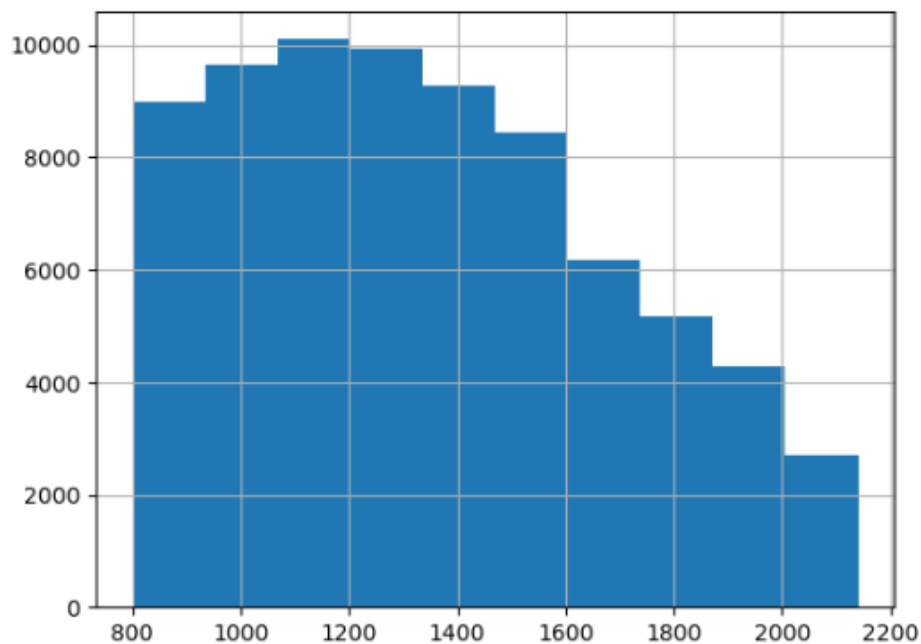
apartments_df = apartments_df[(apartments_df['price'] >= lower_bound) & (apartments_df['price'] <= upper_bound)]
```

```
[36]: apartments_df['price'].describe()
```

```
[36]: count    74660.000000
mean     1352.435012
std       343.323677
min        800.000000
25%      1069.000000
50%      1319.000000
75%      1600.000000
max      2140.000000
Name: price, dtype: float64
```

```
[37]: apartments_df['price'].hist()
```

```
[37]: <Axes: >
```



이상치 제거결과 분포가 상대적으로 정규분포와 유사한 모양이 되었음

- EDA, 피쳐엔지니어링 요약

Feature 1	Action 1	Feature 2	Action 2
id	제거	category	제거
title	제거	body	제거
amenities	제거	bathrooms	사용
bedrooms	사용 (방의 갯수를 더 해 총 방의 갯수 사 용)	currency	제거
fee	제거	has_photo	제거
pets_allowed	이진화하여 사용	price_display	제거
price_type	제거	square_feet	사용 (방의 평균 면적 추가)
address	제거	cityname	도시별 평균 집 값과 분산 추가
state	주별 평균 집값과 분 산 추가	latitude	제거
longitude	제거	source	제거
time	제거		

## 3.2 선형회귀 모델 구축

- sklearn.linear\_model의 LinearRegression은 모델의 통계적해석을 위한 지표를 제공하지 않는다.

따라서 Statsmodels 라이브러리를 사용

```
import statsmodels.api as sm
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

X = apartments_df[['var_price_by_city', 'pets_allowed', 'total_rooms', 'square_feet', 'average_price_by_city', 'average_price_by_state', 'var_price_by_state']]
y = apartments_df['price']

# 데이터 준비 및 스케일링
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Statsmodels를 사용하여 p-value와 adj. R², F-test 등의 정보 확인
X_train_const = sm.add_constant(X_train_scaled) # 절편 추가
model_sm = sm.OLS(y_train, X_train_const).fit()

# 모델 요약
print(model_sm.summary())
```

### OLS Regression Results

```
=====
Dep. Variable:          price    R-squared:                0.449
Model:                  OLS      Adj. R-squared:            0.449
Method:                 Least Squares    F-statistic:          6936.
Date:                   Fri, 15 Nov 2024    Prob (F-statistic):      0.00
Time:                   17:56:09    Log-Likelihood:         -4.1517e+05
No. Observations:       59643    AIC:                   8.304e+05
Df Residuals:           59635    BIC:                   8.304e+05
Df Model:                7
Covariance Type:        nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
x1	-71.1978	1.200	-59.346	0.000	-73.549	-68.846
x2	-3.3672	1.047	-3.216	0.001	-5.419	-1.315
x3	5.257e+12	4.06e+09	1294.808	0.000	5.25e+12	5.26e+12
x4	71.8509	3.929	18.286	0.000	64.149	79.552
x5	241.3922	1.206	200.241	0.000	239.029	243.755
const	-5.951e+15	4.6e+12	-1294.808	0.000	-5.96e+15	-5.94e+15
x6	-3.008e+11	2.32e+08	-1294.808	0.000	-3.01e+11	-3e+11
x7	-2.396e+12	1.85e+09	-1294.808	0.000	-2.4e+12	-2.39e+12
x8	-3.333e+12	2.57e+09	-1294.808	0.000	-3.34e+12	-3.33e+12
x9	32.4348	3.138	10.335	0.000	26.283	38.586

```
=====
Omnibus:                2235.510    Durbin-Watson:          2.003
Prob(Omnibus):           0.000    Jarque-Bera (JB):        3648.062
Skew:                    0.337    Prob(JB):                0.00
Kurtosis:                4.006    Cond. No.:               5.22e+18
=====
```

- 해석

Adj R-squared값이 0.449로, 모델이 예측하고자 하는 price변수의 분산을 44.9% 설명하고 있음을 의미한다.

Prob (f-statistic) 값이 0.00으로 매우 작아서, 전체 모델이 통계적으로 유의미함을 알 수 있다. 이는 모

든 feature의 coef가 0인 귀무가설을 기각한다는 의미이며 모델이 price에 유의미한 영향을 미친다고 볼 수 있다.

Coef는 해당 변수가 예측 대상인 price에 미치는 영향을 나타낸다. 계수가 양수이면 긍정적인 영향을, 음수이면 부정적인 영향을 준다. Coef의 크기가 큰 변수들을 해석하면 다음과 같다.

x3 (방의 갯수)가 1증가하면 price는 큰 폭 ( $5.257 \times 10^{12}$  USD)으로 증가한다.

x4 (집의 넓이)가 1증가하면 price는 71.85 USD만큼 증가한다.

x9 (도시별 평균가격)이 1증가하면 price는 241 USD만큼 증가한다.

그러나 주의 평균 집값이나 화장실, 침실의 갯수는 음의 상관관계를 가지고 있으며 이는 통상적인 상식과 배치된다. 이는 데이터에 다중공선성 문제가 존재하거나 데이터 자체의 설명력이 부족할 가능성이 있음을 의미한다.

pets\_allowed 변수를 제외한 모든 feature들의 p-values는 0.00으로 0.05보다 낮으므로 모든 변수들이 통계적으로 유의미하고 귀무가설을 기각한다. 이는 각 feature들이 price에 유의미한 영향을 미친다고 해석할 수 있다.

### 3.3 선형회귀 모델 예측, 평가

```
y_pred = model_sm.predict(sm.add_constant(X_test_scaled))
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
```

```
Mean Squared Error: 67287.30143358794
Mean Absolute Error: 199.75973952613631
```

MSE는 67287, MAE는 199 정도가 나왔다. MSE는 예측 값과 실제 값 간의 오차를 제공해서 평균을 낸 값이다. MSE는 오차의 제곱을 취하기 때문에 큰 오차에 민감하다.

MAE는 예측 값과 실제 값 간의 절대 오차의 평균을 나타내며, 예측값과 실제값사이에 평균적으로 약 199의 차이가 존재함을 의미한다.

p-value는 각 변수의 통계적 유의성을 보여주는 지표이다. x1(다니는 학교 이름), x7(학습 시간), x9(추가 사교육), x13(가족과의 관계)등의 변수는 p-value가 0.05 미만으로, pass\_fail과 유의미한 관계를 가지고 있을 가능성이 크다. 그러나 x4(어머니의 학력), x12(고등교육 희망여부) 등의 변수는 p-value가 높아서 통계적으로 유의미하지 않다고 볼 수 있다.

전체 모델의 p-value는  $7.179 \times 10^{-31}$ 로 매우 낮아서 전체 모델은 통계적으로 유의미하여, pass\_fail과 피쳐들 간에 의미 있는 관계가 있음을 보여준다.

```

difference = np.abs(y_test - y_pred)

# 차이가 큰 상위 5개의 인덱스와 차이 값 출력
top_5_diff = pd.DataFrame({'y_test': y_test, 'y_pred': y_pred, 'difference': difference})
top_5_diff = top_5_diff.nlargest(5, 'difference')

print(top_5_diff)

```

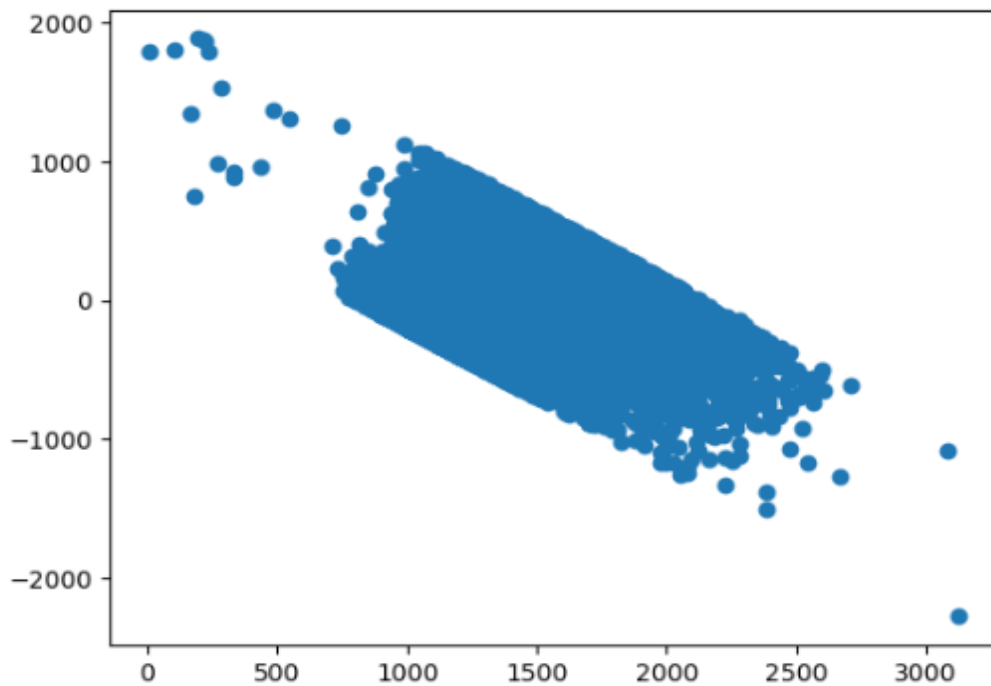
	y_test	y_pred	difference
29826	1475.0	7282.895508	5807.895508
5769	922.0	2368.444092	1446.444092
17381	1800.0	530.834717	1269.165283
90016	1400.0	2639.310547	1239.310547
93281	905.0	2084.887207	1179.887207

예측이 가장 크게 빗나간 상위 5개의 데이터를 살펴보면 위와 같이 큰 오차가 존재하여 전체적인 지표를 떨어뜨리고 있음을 알 수 있다.

## - Residual-fitted plot, QQ-plot, VIF

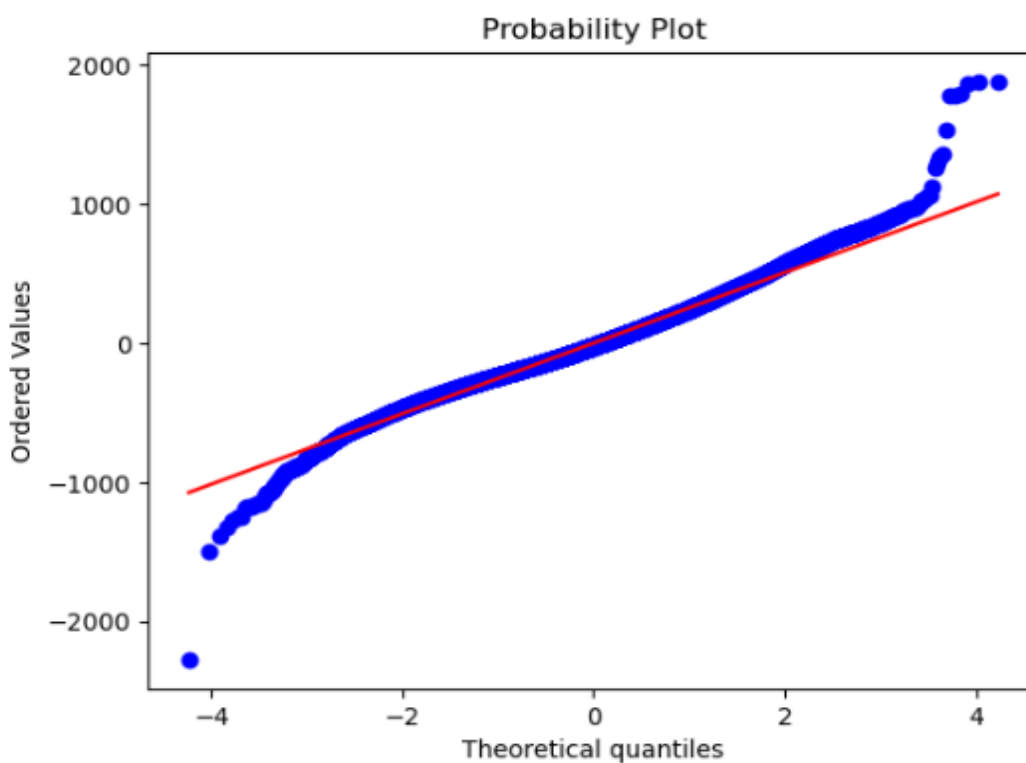
```
[49]: # y 예측값과 잔차간의 관계  
plt.scatter(model_sm.fittedvalues, model_sm.resid)
```

```
[49]: <matplotlib.collections.PathCollection at 0x22e455e0350>
```



```
[52]: import scipy.stats as stats  
import pylab  
# QQ plot 기반의 데이터 정상성 확인  
stats.probplot(model_sm.resid, dist='norm', plot=pylab)
```

```
[52]: ((array([-4.23121663, -4.02730977, -3.91621153, ..., 3.91621153,  
          4.02730977, 4.23121663]),  
      array([-2275.55712891, -1496.59277344, -1381.65625, ...,  
          1865.63134766, 1877.90673828, 1881.18066406])),  
      (253.72427690005335, -0.06354914180790813, 0.9941894911278619))
```



Residual-fitted plot은 종속변수와 독립변수 간의 선형성을 볼 수 있다.

잔차와 적합 값 사이에 랜덤 한 형태의 관계가 나와야 하지만 위 그래프에서는 마름모 모양의 기울어 진패턴이 존재한다. 이는 모델이 데이터의 비선형성을 잘 설명하지 못하고 있음을 의미한다.

QQ-plot은 대각선 위에 데이터 포인트가 놓여 있으면 모집단이 정규성을 따른다고 해석할 수 있다. 위의 그래프는 꼬리 쪽이 대각선에서 벗어나 있어 극단적인 값 들에서 차이가 난다고 볼 수 있다.

```
# VIF Score 계산
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X_scaled, i) for i in range(X_scaled.shape[1])]
print(vif_data)
```

	feature	VIF
0	var_price_by_city	1.275767
1	pets_allowed	1.004061
2	total_rooms	inf
3	square_feet	10.483510
4	average_price_by_city	1.331968
5	average_price_by_state	1.014178
6	var_price_by_state	NaN
7	bathrooms	inf
8	bedrooms	inf
9	average_area_per_room	5.498565

VIF는 다중공선성을 평가하는 지표로 5 이상이면 다중공선성 문제가 있다고 볼 수 있다. 현재 결과에서 total\_rooms, bathrooms, bedrooms 변수의 VIF 값이 무한대로 나타나고 있어, 심각한 다중공선성 문제가 있다. square\_feet와 average\_area\_per\_room 변수도 VIF 값이 상대적으로 높은 편이다.

### 3.4 Shrinkage models (Lasso, Ridge, ElasticNet)

```
from sklearn.linear_model import Ridge, ElasticNet
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import Lasso

# Lasso
lasso = Lasso(alpha=1, random_state=42, max_iter=100000)
lasso.fit(X_train_scaled, y_train)
y_pred = lasso.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Lasso")
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
print("\n")
```

```

# Ridge
ridge = Ridge(alpha=1, random_state=42, max_iter=100000)
ridge.fit(X_train_scaled, y_train)
y_pred_ridge = ridge.predict(X_test_scaled)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print("Ridge")
print("Mean Squared Error (MSE):", mse_ridge)
print("Mean Absolute Error (MAE):", mae_ridge)
print("R-squared (R²):", r2_ridge)
print("\n")

# ElasticNet
elasticnet = ElasticNet(alpha=1, l1_ratio=0.5, random_state=42, max_iter=100000)
elasticnet.fit(X_train_scaled, y_train)
y_pred_elasticnet = elasticnet.predict(X_test_scaled)

mse_elasticnet = mean_squared_error(y_test, y_pred_elasticnet)
mae_elasticnet = mean_absolute_error(y_test, y_pred_elasticnet)
r2_elasticnet = r2_score(y_test, y_pred_elasticnet)

print("ElasticNet")
print("Mean Squared Error (MSE):", mse_elasticnet)
print("Mean Absolute Error (MAE):", mae_elasticnet)
print("R-squared (R²):", r2_elasticnet)

```

Lasso  
Mean Squared Error (MSE): 66698.36244354265  
Mean Absolute Error (MAE): 200.08445929740853  
R-squared (R²): 0.4307383945954272

Ridge  
Mean Squared Error (MSE): 67237.56519237571  
Mean Absolute Error (MAE): 199.7554551113367  
R-squared (R²): 0.42613637122942505

ElasticNet  
Mean Squared Error (MSE): 74676.2126712906  
Mean Absolute Error (MAE): 222.79200757687528  
R-squared (R²): 0.36264850959758765

## - 결과해석

alpha 파라미터를 1로 하였다. 전체적으로 규제를 강하게 할수록 원본보다 성능이 감소하였다.

Lasso 모델은 기존의 모델과 성능차이가 크지 않았는데 불필요한 특성의 가중치를 0으로 하여 모델을 단순화시킨다는 특징을 고려하면 성능 감소가 크지 않다고 볼 수 있다.

Ridge는 기존 모델에 비해 설명력과 오차가 떨어졌다.


ElasticNet은 성능이 가장 떨어졌다. 규제가 과도하게 적용되어 지나치게 모델이 단순화되었을 것으로 보인다. 현재 구성한 모델의 과적합 위험성을 고려한다면 단순화된 Lasso 모델이 적합한 규제모델로 판단된다.



# <로지스틱회귀>

## 1. 데이터 선정

선정 데이터셋: Student Performance



### Student Performance

Donated on 11/26/2014

Predict student performance in secondary education (high school).

<b>Dataset Characteristics</b> Multivariate	<b>Subject Area</b> Social Science	<b>Associated Tasks</b> Classification, Regression
<b>Feature Type</b> Integer	<b># Instances</b> 649	<b># Features</b> 30

<https://archive.ics.uci.edu/dataset/320/student+performance>

[3]:	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	4	0	11	11
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	2	9	11	11
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	6	12	13	12
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	0	14	14	14
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	0	11	13	13
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
644	MS	F	19	R	GT3	T	2	3	services	other	...	5	4	2	1	2	5	4	10	11	10
645	MS	F	18	U	LE3	T	3	1	teacher	services	...	4	3	4	1	1	1	4	15	15	16
646	MS	F	18	U	GT3	T	1	1	other	other	...	1	1	1	1	1	5	6	11	12	9
647	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5	3	4	2	6	10	10	10
648	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1	3	4	5	4	10	11	11

649 rows × 33 columns

## 2. 코드전문

```
: import matplotlib.pyplot as plt
import seaborn as sns

: import pandas as pd
import numpy as np
student_df = pd.read_csv('student-por.csv', sep=';')

: student_df
```

### EDA, 전처리, Feature Engineering

```
student_df.info()

student_df.describe().T

student_df['pass_fail'] = student_df['G3'].apply(lambda x: 1 if x >= 12 else 0)
```

분류문제로 만들기 위해 G3 피처가 12점 이상인 경우를 합격으로 한다.

## 각 피처별 score 분포 확인

### 1. School

```
student_df['school'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='school', y='G3')
plt.title("Score Distribution by School")
plt.show()
```

GP 학교가 평균적으로 최종점수가 높다 (School feature는 예측에 유의미하다고 볼 수 있음)

### 2. Sex

```
student_df['sex'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='sex', y='G3')
plt.title("Score Distribution by Sex")
plt.show()
```

성별에 따라 점수차이가 있지만 결정적인 요소라고 판단하기에는 어려워보임

### 3. age

```
student_df['age'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='age', y='G3')
plt.title("Score Distribution by age")
plt.show()
```

19세 이상으로는 성적이 떨어지는 경향성이 있다

### 4. address

```
student_df['address'].value_counts() # U : 도시, R : 시/골
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='address', y='G3')
plt.title("Score Distribution by address")
plt.show()
```

도시 거주 학생이 시골거주 학생보다 성적이 약간 높은 경향성이 있지만 결정적인 변수는 아닌것으로 보임

### 5. famsize

```
student_df['famsize'].value_counts()
```

3보다 가족수가 많으면 GT3, 적으면 LE3

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='famsize', y='G3')
plt.title("Score Distribution by famsize")
plt.show()
```

가족의 숫자는 성적에 영향이 없으므로 제거하는게 좋아보인다.

### 6. Pstatus

```
student_df['Pstatus'].value_counts() # 부모님이랑 같이 살면 T 아니면 A
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='Pstatus', y='G3')
plt.title("Score Distribution by Pstatus")
plt.show()
```

부모님과 같이 사는지는 성적에 영향이 없다

## 7. Medu

```
student_df['Medu'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='Medu', y='G3')
plt.title("Score Distribution by Medu")
plt.show()
```

## 8. Fedu

```
student_df['Fedu'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='Fedu', y='G3')
plt.title("Score Distribution by Fedu")
plt.show()
```

부모의 교육수준이 올라갈 수록, 아이의 성적도 오르는 경향성이 있다.

## 9. Mjob

```
student_df['Mjob'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='Mjob', y='G3')
plt.title("Score Distribution by Mjob")
plt.show()
```

## 10. Fjob

```
student_df['Fjob'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='Fjob', y='G3')
plt.title("Score Distribution by Fjob")
plt.show()
```

부모님의 직장이 교사라면 성적이 높은 경향성이 있다. 따라서 부모중 교사의 존재 여부를 새로운 feature로 삼는게 좋아보임

## 11. reason

```
student_df['reason'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='reason', y='G3')
plt.title("Score Distribution by reason")
plt.show()
```

## 12. guardian

```
student_df['guardian'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='guardian', y='G3')
plt.title("Score Distribution by guardian")
plt.show()
```

보호자가 엄마, 아빠가 아닌 경우를 체크하는 feature를 생성하는 방향으로 접근하자

## 13. traveltime

```
student_df['traveltime'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='traveltime', y='G3')
plt.title("Score Distribution by traveltime")
plt.show()
```

통학시간이 길어지면 성적도 떨어지는 경향성이 있다

## 14. studytime

```
student_df['studytime'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='studytime', y='G3')
plt.title("Score Distribution by studytime")
plt.show()
```

공부시간과 성적은 어느정도 비례한다

## 15. failures

```
student_df['failures'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='failures', y='G3')
plt.title("Score Distribution by failures")
plt.show()
```

이전 수업에서 낙제한 경우가 있으면 성적이 크게 감소한다.

## 16. schoolsup

```
student_df['schoolsup'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='schoolsup', y='G3')
plt.title("Score Distribution by schoolsup")
plt.show()
```

## 17. famsup

```
student_df['famsup'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='famsup', y='G3')
plt.title("Score Distribution by famsup")
plt.show()
```

사교육은 성적에 그다지 영향을 주지 않는 것으로 보인다.

## 18. paid

```
student_df['paid'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='paid', y='G3')
plt.title("Score Distribution by paid")
plt.show()
```

## 19. activities

```
student_df['activities'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='activities', y='G3')
plt.title("Score Distribution by activities")
plt.show()
```

방과후 활동은 성적에 영향이 없음

## 20. nursery

```
student_df['nursery'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='nursery', y='G3')
plt.title("Score Distribution by nursery")
plt.show()
```

유치원에 다녔는지는 큰 영향이 없어보임

## 21. higher

```
student_df['higher'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='higher', y='G3')
plt.title("Score Distribution by higher")
plt.show()
```

더 높은 수준의 교육을 원하는 학생이 성적이 확실히 높다

## 22. internet

```
student_df['internet'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='internet', y='G3')
plt.title("Score Distribution by internet")
plt.show()
```

인터넷 여부는 성적에 큰 영향을 주지 않는다.

## 23. romantic

```
student_df['romantic'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='romantic', y='G3')
plt.title("Score Distribution by romantic")
plt.show()
```

전혀 관계없는 변수

## 24. famrel

```
student_df['famrel'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='famrel', y='G3')
plt.title("Score Distribution by famrel")
plt.show()
```

가족관계는 어느정도 보조변수로 사용할 수 있을 것으로 보인다

## 25. freetime

```
student_df['freetime'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='freetime', y='G3')
plt.title("Score Distribution by freetime")
plt.show()
```

차이가 있기는 한데 일관적이지 않고 영향이 적을 것으로 생각됨

## 26. goout

```
student_df['goout'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='goout', y='G3')
plt.title("Score Distribution by goout")
plt.show()
```

친구와 자주 노는지는 성적에 그다지 영향이 없어보인다

## 27. Dalc

```
student_df['Dalc'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='Dalc', y='G3')
plt.title("Score Distribution by Dalc")
plt.show()
```

음주 비율이 낮은 학생이 더 좋은 성적을 받는 경향이 있다.

## 28. Walc

```
student_df['Walc'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='Walc', y='G3')
plt.title("Score Distribution by Walc")
plt.show()
```

들다 쓸 필요는 없고 Dalc를 쓰는 편이 더 좋아보인다

## 30. absences

```
student_df['absences'].value_counts()
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='absences', y='G3')
plt.title("Score Distribution by absences")
plt.show()
```

결석횟수가 많을 수록 성적이 낮아지는 경향성이 있는 것 같다

## Correlation Matrix

```
student_df.corr(numeric_only=True)
```

```
plt.figure(figsize=(5, 5))
sns.heatmap(data=student_df.corr(numeric_only=True), annot=True, fmt='.3f', linewidths=0.5, cmap='Blues')
```

## Feature Engineering

```
student_df['age_19_or_above'] = student_df['age'].apply(lambda x: 1 if x >= 19 else 0)
student_df = student_df.drop(columns=['age'])

student_df = student_df.drop(columns=['address', 'famsize', 'activities', 'nursery', 'internet', \
                                     'romantic', 'freetime', 'goout', 'Walc', 'health', 'G1', 'G2', 'G3'])

student_df['parent_teacher'] = student_df[['Mjob', 'Fjob']].apply(lambda x: 1 if 'teacher' in x.values else 0, axis=1)
student_df = student_df.drop(columns=['Mjob', 'Fjob'])

student_df = student_df.drop(columns=['reason'])

student_df['guardian_other'] = student_df['guardian'].apply(lambda x: 1 if x == 'other' else 0)
student_df = student_df.drop(columns=['guardian'])

student_df['school'] = student_df['school'].apply(lambda x: 0 if x == 'GP' else 1)

student_df['sex'] = student_df['sex'].apply(lambda x: 0 if x == 'F' else 1)
student_df['higher'] = student_df['higher'].apply(lambda x: 0 if x == 'yes' else 1)

student_df['Pstatus'] = student_df['Pstatus'].apply(lambda x: 0 if x == 'A' else 1)
student_df['schoolsup'] = student_df['schoolsup'].apply(lambda x: 0 if x == 'yes' else 1)
student_df['famsup'] = student_df['famsup'].apply(lambda x: 0 if x == 'no' else 1)
student_df['paid'] = student_df['paid'].apply(lambda x: 0 if x == 'no' else 1)

student_df
```

## 학습-테스트데이터 분리, 데이터 스케일링

```
# 타겟변수 pass_fail 분리
X = student_df.drop(columns=['pass_fail'])
y = student_df['pass_fail']

# 스케일링
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

from sklearn.model_selection import train_test_split

# 7:1:2 비율로 학습, 검증, 테스트 데이터 분할
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=2/3, random_state=42)

print(X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test.shape, y_test.shape)
```

## 로지스틱회귀 모델 구축과 학습

sklearn.linear\_model의 LogisticRegression은 모델의 통계적해석을 위한 지표를 제공하지 않는다.

따라서 Statsmodels 라이브러리를 사용

```
import statsmodels.api as sm

X_train_with_const = sm.add_constant(X_train)
y_train = y_train

# 로지스틱 회귀 모델 구축, 학습
logit_model = sm.Logit(y_train, X_train_with_const)
logit_results = logit_model.fit()

from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score

# 예측 확률 계산 (0 ~ 1 사이의 값)
y_val_pred_prob = logit_results.predict(sm.add_constant(X_val))

# 임계값 0.5를 기준으로 클래스 예측 (0 또는 1)
y_val_pred = np.where(y_val_pred_prob >= 0.5, 1, 0)
```

## 모델의 통계 지표 확인

```
print(logit_results.summary())

con_mat = confusion_matrix(y_val, y_val_pred)
print("Confusion Matrix:\n", con_mat)

# 정밀도, 재현율, F1 점수 계산
precision = precision_score(y_val, y_val_pred)
recall = recall_score(y_val, y_val_pred)
f1 = f1_score(y_val, y_val_pred)

print("Precision:\n", precision)
print("Recall:\n", recall)
print("F1 Score:\n", f1)
```

## Lasso, Ridge, Elasticnet

```
from sklearn.linear_model import LogisticRegression

# Lasso
lasso_model = LogisticRegression(penalty='l1', solver='liblinear', max_iter=1000, random_state=42, C=0.05)
lasso_model.fit(X_train, y_train)
y_val_pred_prob_lasso = lasso_model.predict_proba(X_val)[: , 1]
y_val_pred_lasso = np.where(y_val_pred_prob_lasso >= 0.5, 1, 0)
precision_lasso = precision_score(y_val, y_val_pred_lasso)
recall_lasso = recall_score(y_val, y_val_pred_lasso)
f1_lasso = f1_score(y_val, y_val_pred_lasso)
confusion_lasso = confusion_matrix(y_val, y_val_pred_lasso)

print("Lasso Model:")
print("Precision:", precision_lasso)
print("Recall:", recall_lasso)
print("F1 Score:", f1_lasso)
print("Confusion Matrix:\n", confusion_lasso)
print()

# Ridge
ridge_model = LogisticRegression(penalty='l2', solver='liblinear', max_iter=1000, random_state=42, C=0.05)
ridge_model.fit(X_train, y_train)
y_val_pred_prob_ridge = ridge_model.predict_proba(X_val)[: , 1]
y_val_pred_ridge = np.where(y_val_pred_prob_ridge >= 0.5, 1, 0)
precision_ridge = precision_score(y_val, y_val_pred_ridge)
recall_ridge = recall_score(y_val, y_val_pred_ridge)
f1_ridge = f1_score(y_val, y_val_pred_ridge)
confusion_ridge = confusion_matrix(y_val, y_val_pred_ridge)

print("Ridge Model:")
print("Precision:", precision_ridge)
print("Recall:", recall_ridge)
print("F1 Score:", f1_ridge)
print("Confusion Matrix:\n", confusion_ridge)
print()

# ElasticNet
elasticnet_model = LogisticRegression(penalty='elasticnet', solver='saga', l1_ratio=0.5, max_iter=1000, random_state=42, C=0.05)
elasticnet_model.fit(X_train, y_train)
y_val_pred_prob_elasticnet = elasticnet_model.predict_proba(X_val)[: , 1]
y_val_pred_elasticnet = np.where(y_val_pred_prob_elasticnet >= 0.5, 1, 0)
precision_elasticnet = precision_score(y_val, y_val_pred_elasticnet)
recall_elasticnet = recall_score(y_val, y_val_pred_elasticnet)
f1_elasticnet = f1_score(y_val, y_val_pred_elasticnet)
confusion_elasticnet = confusion_matrix(y_val, y_val_pred_elasticnet)

print("ElasticNet:")
print("Precision:", precision_elasticnet)
print("Recall:", recall_elasticnet)
print("F1 Score:", f1_elasticnet)
print("Confusion Matrix:\n", confusion_elasticnet)
```

## 3. 코드 실행 결과 설명

### - 데이터 구성 feature 개요

---

school - 학교 ("GP" - Gabriel Pereira 또는 "MS" - Mousinho da Silveira, 형식: object)



sex - 성별 ("F" - 여자 또는 "M" - 남자, 형식: object)

age - 나이 (15세에서 22세까지, 형식: int64)

address - 집 주소 ("U" - 도시 또는 "R" - 시골, 형식: object)

famsize - 가족의 규모 ("LE3" - 3명 이하 또는 "GT3" - 3명 초과, 형식: object)

Pstatus - 부모와의 동거 여부 ("T" - 함께 생활 또는 "A" - 별거, 형식: object)

Medu - 어머니의 교육 수준 (0 - 없음, 1 - 초등 교육, 2 - 5~9학년, 3 - 중등 교육, 4 - 고등 교육, 형식: int64)

Fedu - 아버지의 교육 수준 (0 - 없음, 1 - 초등 교육, 2 - 5~9학년, 3 - 중등 교육, 4 - 고등 교육, 형식: int64)

Mjob - 어머니의 직업 ("teacher" - 교사, "health" - 건강 관련, civil "services" - 공무원(행정 또는 경찰 등), "at\_home" - 가정주부, "other" - 기타, 형식: object)

Fjob - 아버지의 직업 ("teacher" - 교사, "health" - 건강 관련, civil "services" - 공무원(행정 또는 경찰 등), "at\_home" - 가정주부, "other" - 기타, 형식: object)

reason - 학교를 선택한 이유 ("home" - 집에서 가까워서, "reputation" - 학교 평판, "course" - 과정 선호, "other" - 기타, 형식: object)

guardian - 보호자 ("mother" - 어머니, "father" - 아버지, "other" - 기타, 형식: object)

traveltime - 통학 시간 (1 - 15분 이하, 2 - 15분 ~ 30분, 3 - 30분 ~ 1시간, 4 - 1시간 이상, 형식: int64)

studytime - 한주당 공부 시간 (1 - 2시간이하, 2 - 2 ~ 5시간, 3 - 5 ~ 10시간, 4 - 10시간 이상, 형식: int64)

failures - 과거 낙제 횟수 (1, 2, 3 ~~~ 4 이상일 경우 4, 형식: int64)

schoolsup - 추가 교육 지원 (yes 또는 no, 형식: object)

famsup - 가족의 교육 지원 (yes 또는 no, 형식: object)

paid - 추가적으로 유료수업을 수강 (yes 또는 no, 형식: object)

activities - 방과후 활동 (yes 또는 no, 형식: object)

nursery - 유치원에 다닌 여부 (yes 또는 no, 형식: object)

higher - 고등 교육 희망 여부 (yes 또는 no, 형식: object)

internet - 집에서의 인터넷 사용가능 여부 (yes 또는 no, 형식: object)

romantic - 연애 경험 유무 (yes 또는 no, 형식: object)

famrel - 가족 관계가 화목한 정도 (1 - 매우 나쁨 ~ 5 - 매우 좋음, 형식: int64)

freetime - 방과후 여가 시간 (1 - 매우 적음 ~ 5 - 매우 많음, 형식: int64)

goout - 친구들과의 외출 빈도 (1 - 매우 적음 ~ 5 - 매우 많음, 형식: int64)

Dalc - 평일 음주 빈도(1 - 매우 적음 ~ 5 - 매우 많음, 형식: int64)

Walc - 주말 음주 빈도(1 - 매우 적음 ~ 5 - 매우 많음, 형식: int64)

health - 현재 건강 상태 (1 - 매우 나쁨 ~ 5 - 매우 좋음, 형식: int64)

absences - 학교 결석 횟수 (0 ~ 93, 형식: int64)

---

## - 분석목표: 최종점수(G3) 기반으로 합격(G3가 15점이상) 불합격을 분류

이때, G1, G2 시험성적을 사용하면 너무 예측이 간단하므로 G1, G2는 사용하지 않고 나머지 피쳐로만 예측을 수행

```
[6]: student_df['pass_fail'] = student_df['G3'].apply(lambda x: 1 if x >= 12 else 0)
```

분류문제로 만들기 위해 G3 피쳐가 12점 이상인 경우를 합격으로 한다.

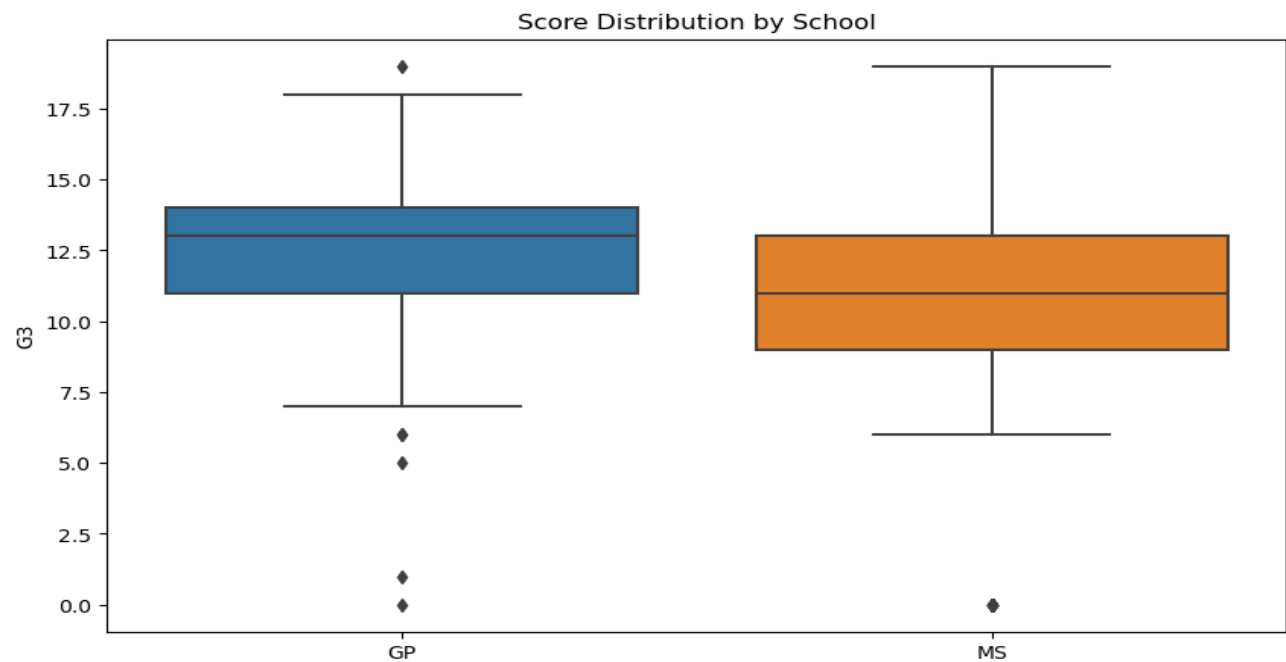
## 3.1 EDA & Feature Engineering & Data 전처리

### 1. School

```
student_df['school'].value_counts()
```

```
school
GP    423
MS    226
Name: count, dtype: int64
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='school', y='G3')
plt.title("Score Distribution by School")
plt.show()
```

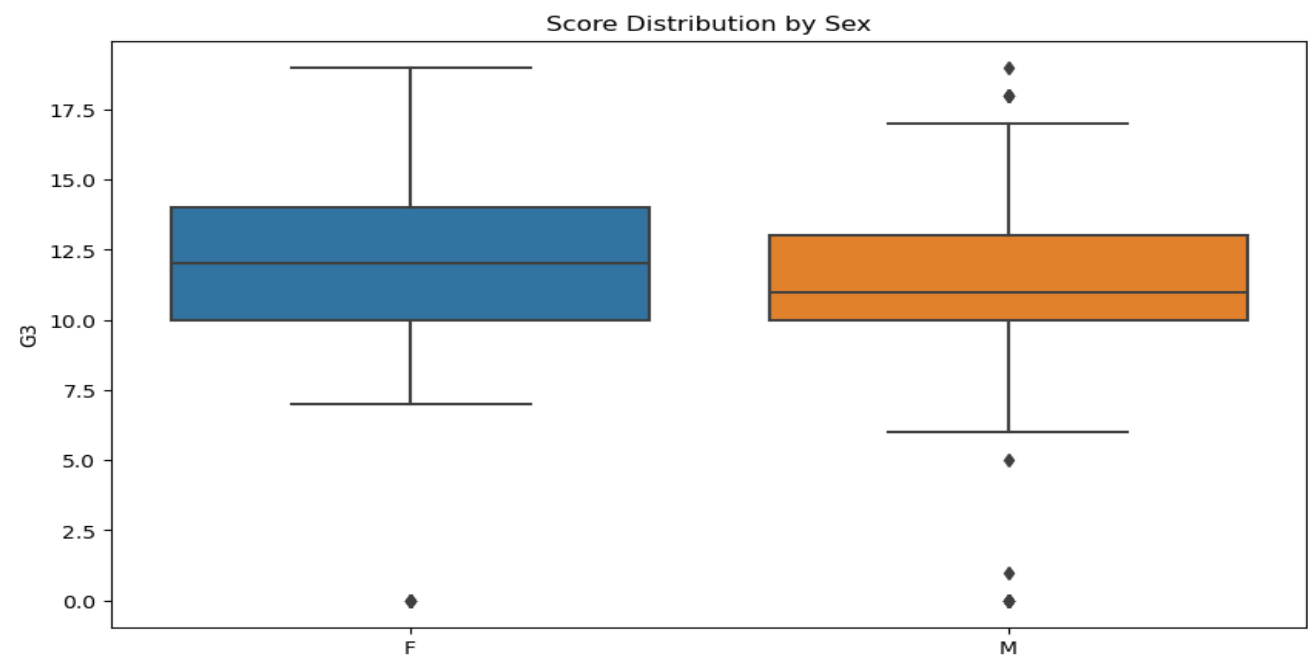


### 2. Sex

```
student_df['sex'].value_counts()
```

```
sex
F    383
M    266
Name: count, dtype: int64
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='sex', y='G3')
plt.title("Score Distribution by Sex")
plt.show()
```

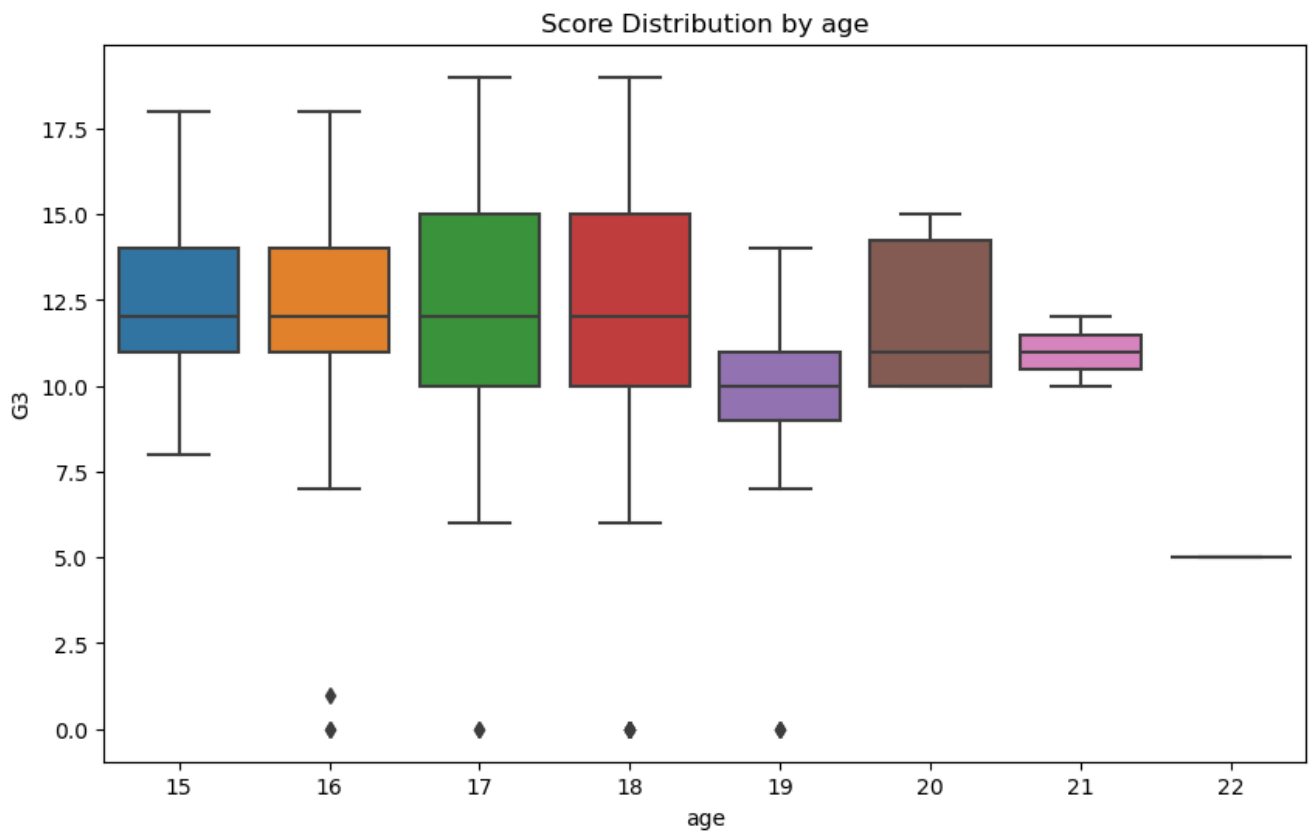


### 3. age

```
[13]: student_df['age'].value_counts()
```

```
[13]: age
17    179
16    177
18    140
15    112
19     32
20      6
21      2
22      1
Name: count, dtype: int64
```

```
[14]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='age', y='G3')
plt.title("Score Distribution by age")
plt.show()
```



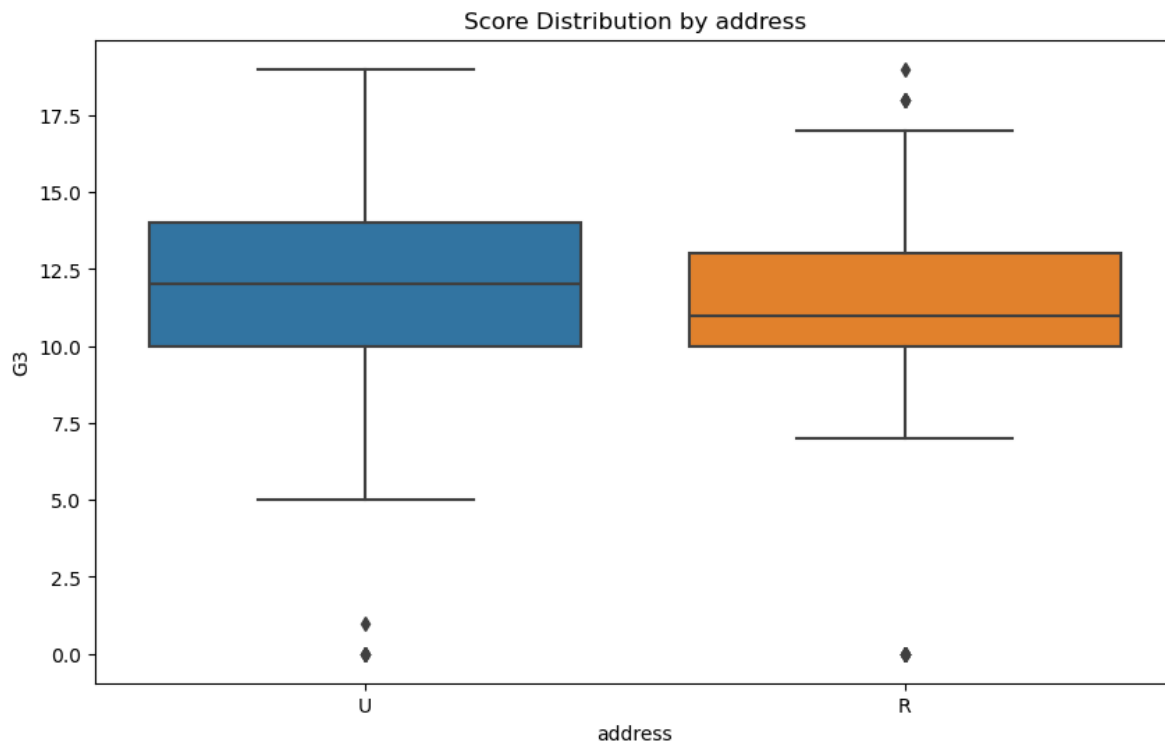
19세 이상으로는 성적이 떨어지는 경향성이 있다

### 4. address

```
[15]: student_df['address'].value_counts() # U : 도시, R : 시골
```

```
[15]: address
U     452
R     197
Name: count, dtype: int64
```

```
[16]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='address', y='G3')
plt.title("Score Distribution by address")
plt.show()
```



도시 거주 학생이 시골거주 학생보다 성적이 약간 높은 경향성이 있지만 결정적인 변수는 아닌것으로 보임

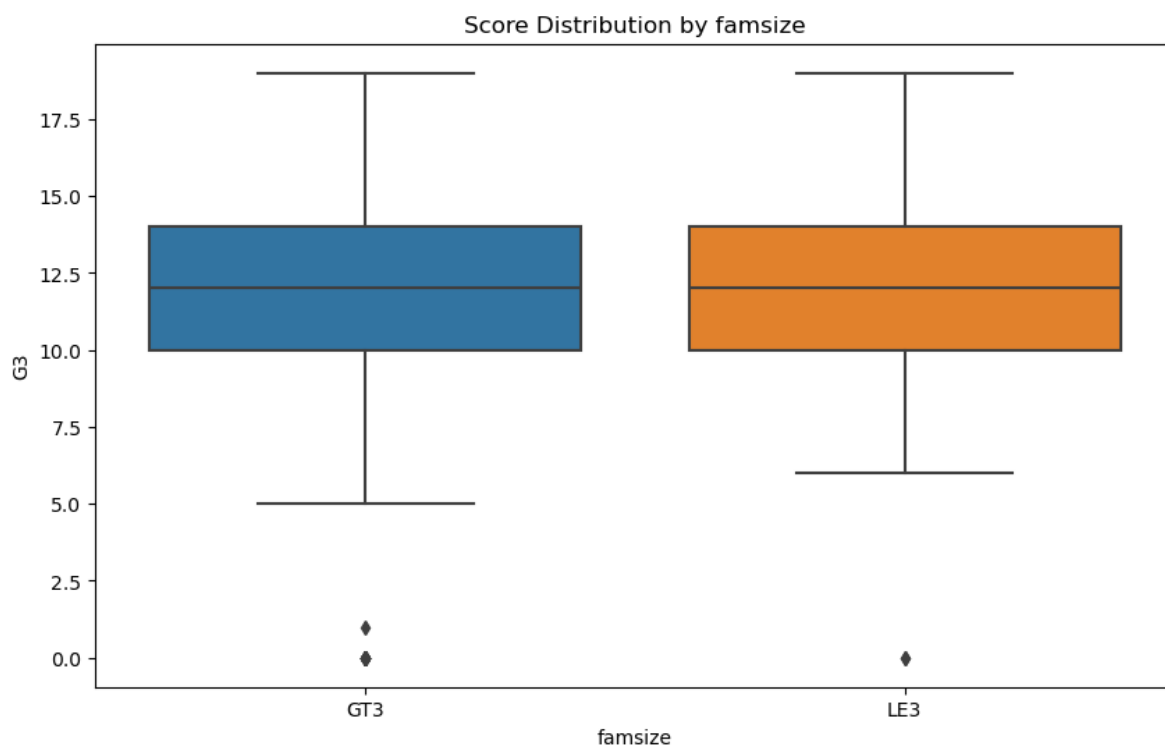
## 5. famsize

```
[17]: student_df['famsize'].value_counts()
```

```
[17]: famsize
GT3    457
LE3    192
Name: count, dtype: int64

3보다 가족수가 많으면 GT3, 적으면 LE3
```

```
[18]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='famsize', y='G3')
plt.title("Score Distribution by famsize")
plt.show()
```



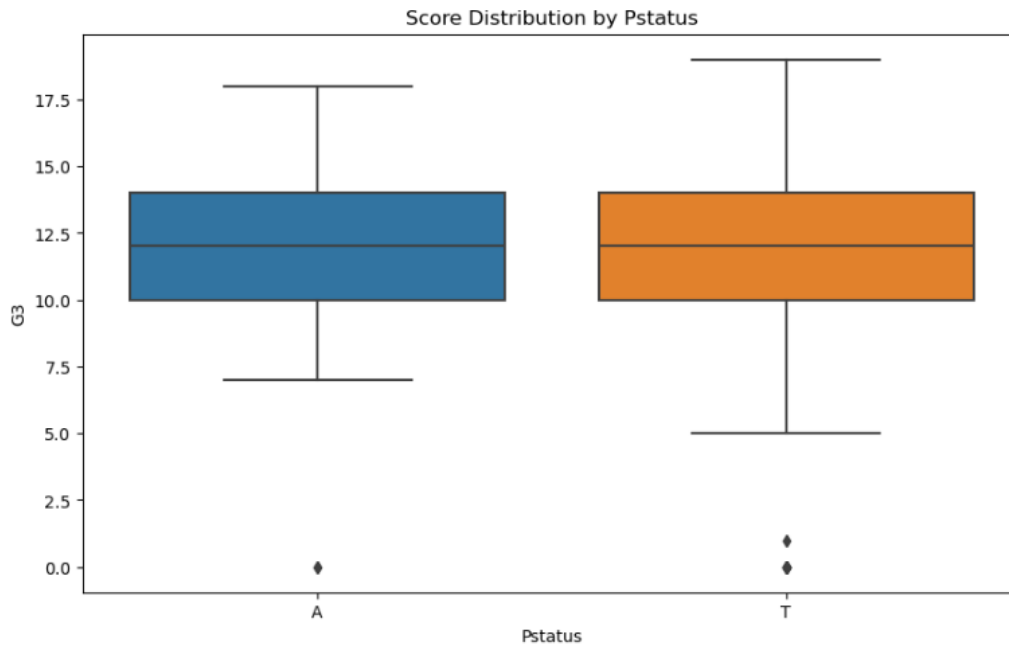
가족의 숫자는 성적에 영향이 없으므로 제거하는게 좋아보인다.

## 6. Pstatus

```
[19]: student_df['Pstatus'].value_counts() # 부모님이랑 같이 살면 T 아니면 A
```

```
[19]: Pstatus  
T    569  
A     80  
Name: count, dtype: int64
```

```
[20]: plt.figure(figsize=(10, 6))  
sns.boxplot(data=student_df, x='Pstatus', y='G3')  
plt.title("Score Distribution by Pstatus")  
plt.show()
```



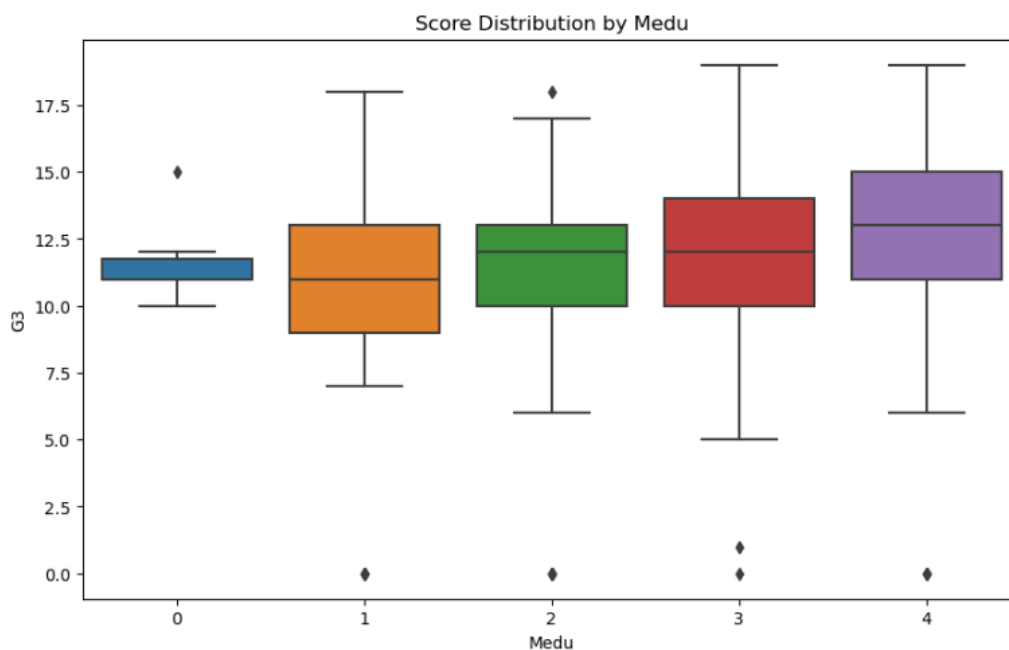
부모님과 같이 사는지는 성적에 영향이 없다

## 7. Medu

```
[21]: student_df['Medu'].value_counts()
```

```
[21]: Medu  
2    186  
4    175  
1    143  
3    139  
0      6  
Name: count, dtype: int64
```

```
[22]: plt.figure(figsize=(10, 6))  
sns.boxplot(data=student_df, x='Medu', y='G3')  
plt.title("Score Distribution by Medu")  
plt.show()
```

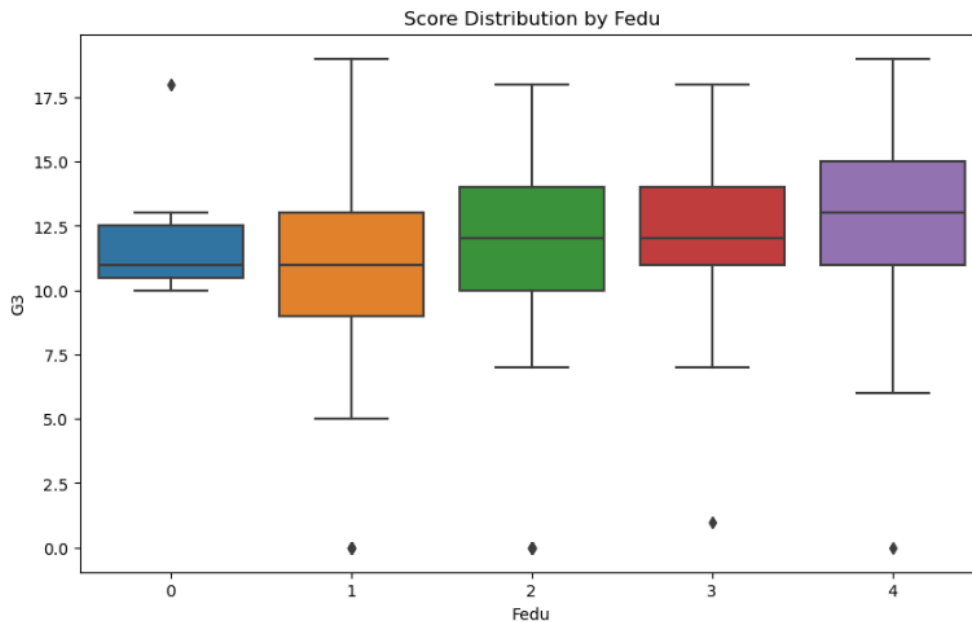


## 8. Fedu

```
[23]: student_df['Fedu'].value_counts()
```

```
[23]: Fedu
      2    209
      1    174
      3    131
      4    128
      0     7
      Name: count, dtype: int64
```

```
[24]: plt.figure(figsize=(10, 6))
      sns.boxplot(data=student_df, x='Fedu', y='G3')
      plt.title("Score Distribution by Fedu")
      plt.show()
```



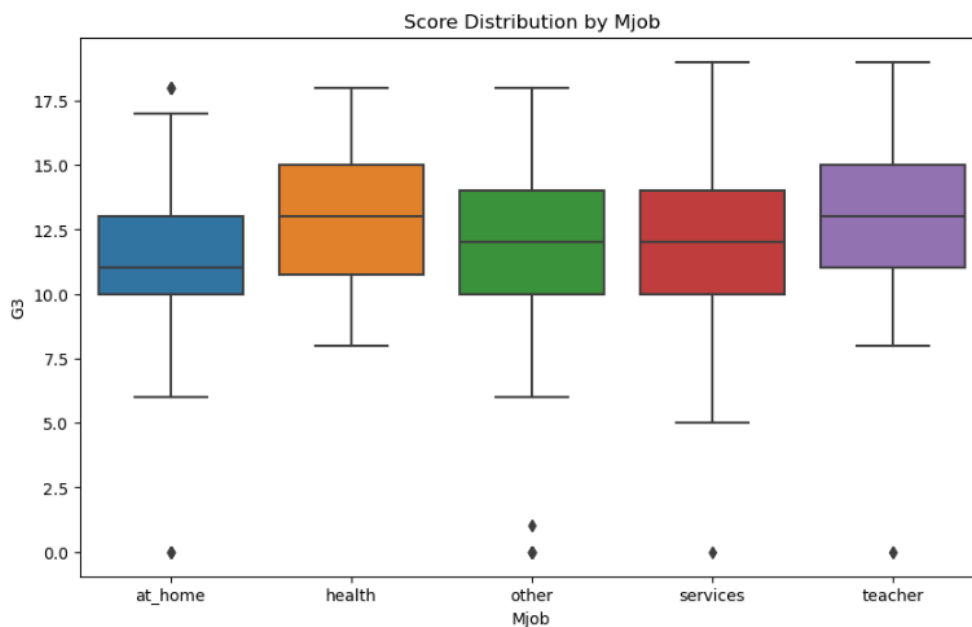
부모의 교육수준이 올라갈 수록, 아이의 성적도 오르는 경향성이 있다.

## 9. Mjob

```
[25]: student_df['Mjob'].value_counts()
```

```
[25]: Mjob
      other    258
      services 136
      at_home  135
      teacher   72
      health    48
      Name: count, dtype: int64
```

```
[26]: plt.figure(figsize=(10, 6))
      sns.boxplot(data=student_df, x='Mjob', y='G3')
      plt.title("Score Distribution by Mjob")
      plt.show()
```

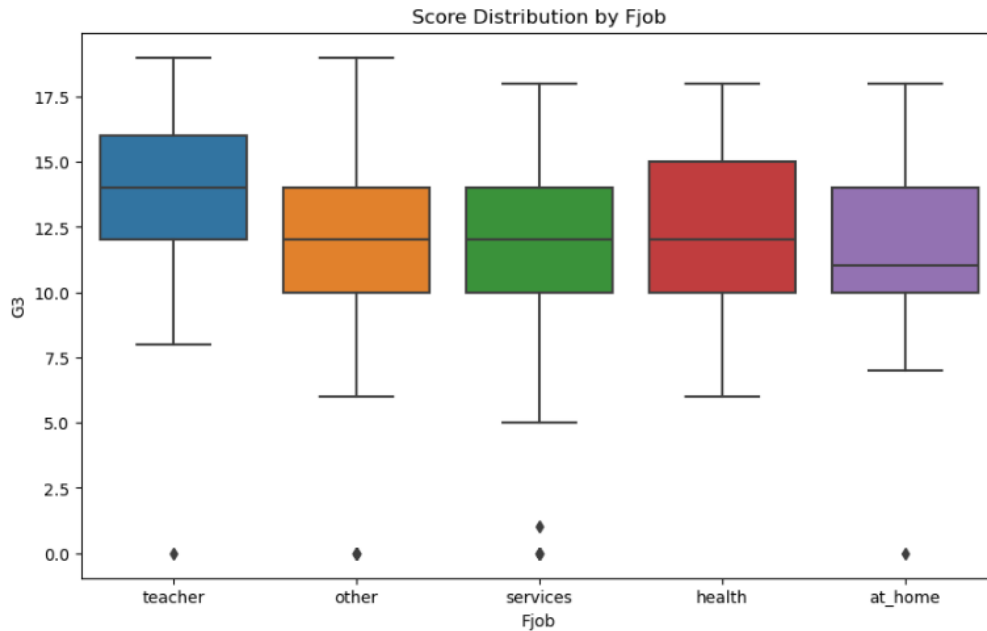


## 10. Fjob

```
[27]: student_df['Fjob'].value_counts()
```

```
[27]: Fjob
other      367
services   181
at_home    42
teacher     36
health      23
Name: count, dtype: int64
```

```
[28]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='Fjob', y='G3')
plt.title("Score Distribution by Fjob")
plt.show()
```



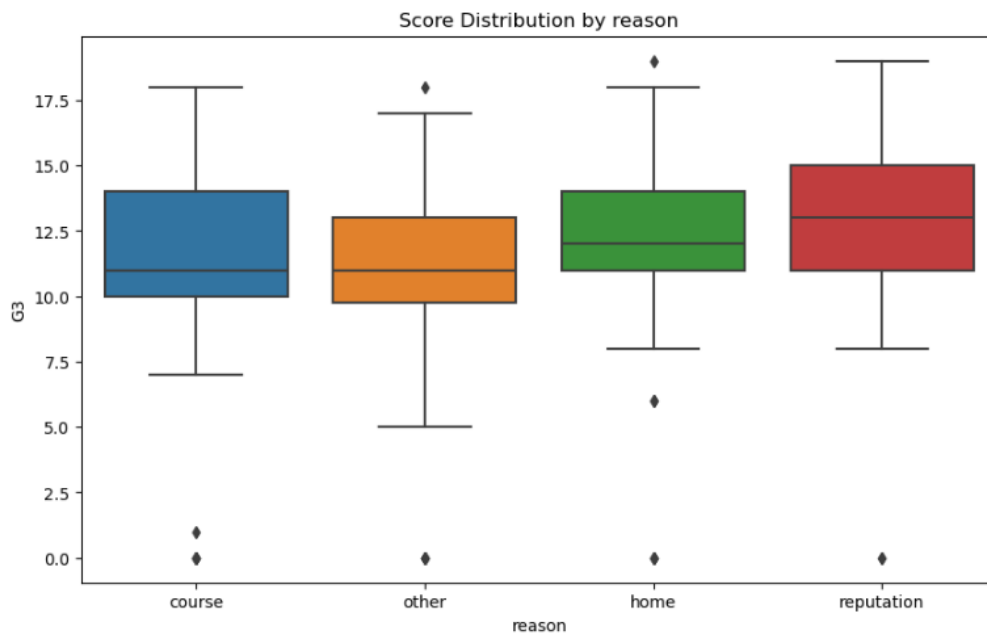
부모님의 직장이 교사라면 성적이 높은 경향성이 있다. 따라서 부모중 교사의 존재 여부를 새로운 feature로 삼는게 좋아보임

## 11. reason

```
[29]: student_df['reason'].value_counts()
```

```
[29]: reason
course      285
home        149
reputation  143
other        72
Name: count, dtype: int64
```

```
[30]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='reason', y='G3')
plt.title("Score Distribution by reason")
plt.show()
```

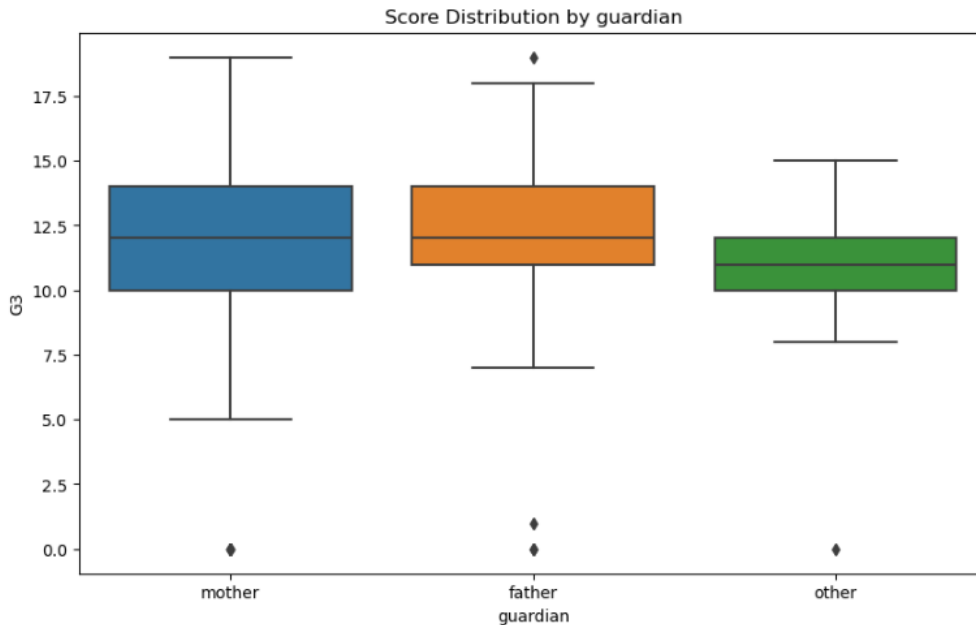


## 12. guardian ¶

```
[31]: student_df['guardian'].value_counts()
```

```
[31]: guardian
mother    455
father    153
other      41
Name: count, dtype: int64
```

```
[32]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='guardian', y='G3')
plt.title("Score Distribution by guardian")
plt.show()
```



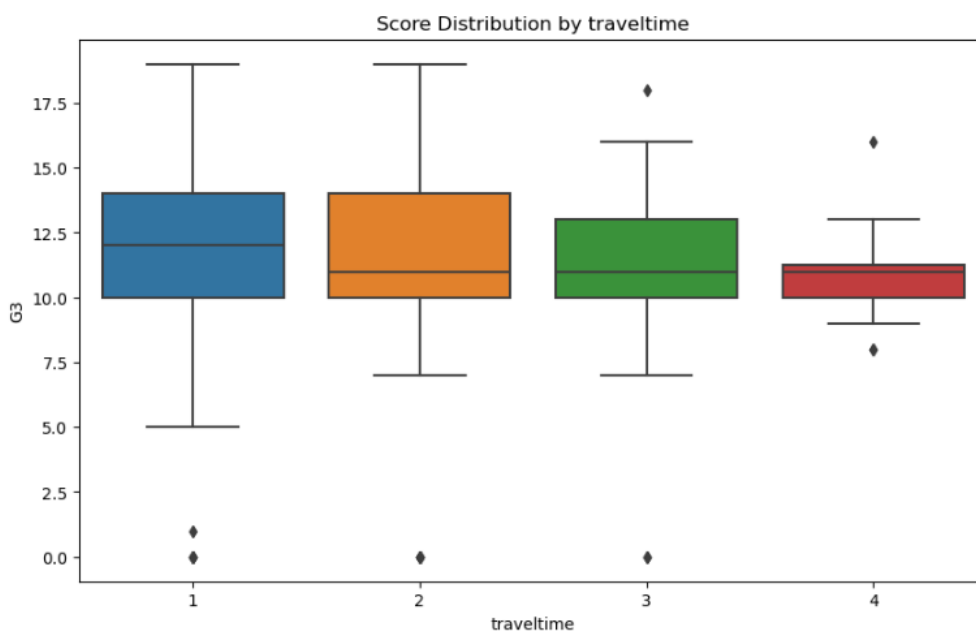
보호자가 엄마, 아빠가 아닌경우를 체크하는 feature를 생성하는 방향으로 접근하자

## 13. traveltime

```
[33]: student_df['traveltime'].value_counts()
```

```
[33]: traveltime
1    366
2    213
3     54
4     16
Name: count, dtype: int64
```

```
[34]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='traveltime', y='G3')
plt.title("Score Distribution by traveltime")
plt.show()
```



통학시간이 길어지면 성적도 떨어지는 경향성이 있다

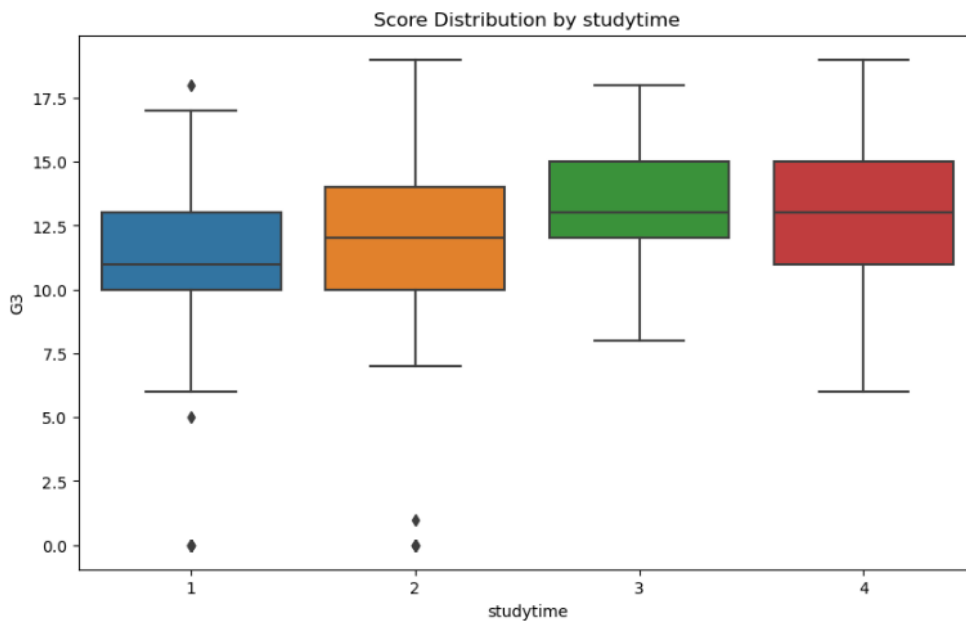


## 14. studytime

```
[35]: student_df['studytime'].value_counts()
```

```
[35]: studytime
2    305
1    212
3     97
4     35
Name: count, dtype: int64
```

```
[36]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='studytime', y='G3')
plt.title("Score Distribution by studytime")
plt.show()
```



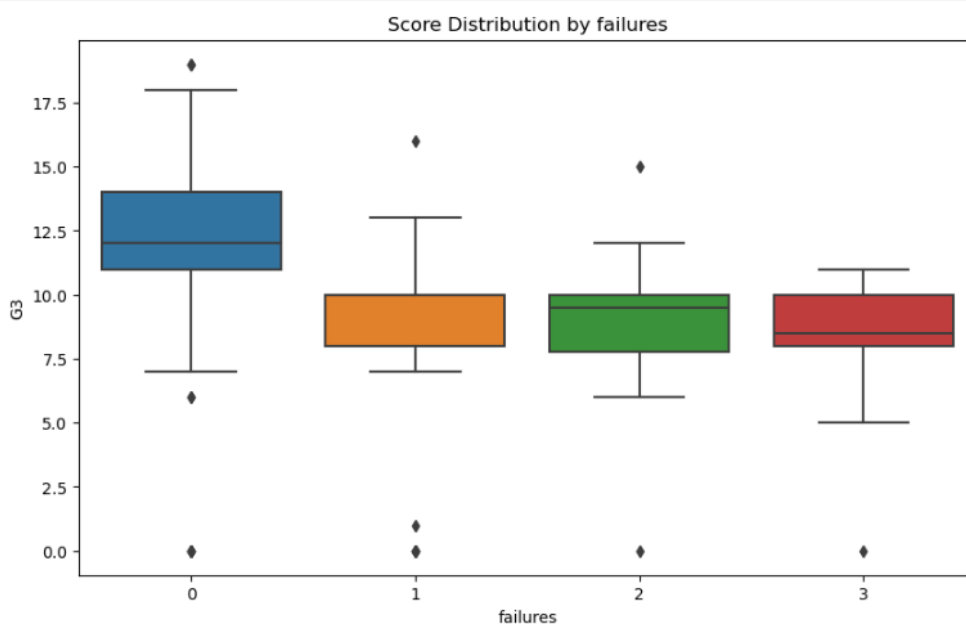
공부시간과 성적은 어느정도 비례한다

## 15. failures

```
[37]: student_df['failures'].value_counts()
```

```
[37]: failures
0    549
1     70
2     16
3     14
Name: count, dtype: int64
```

```
[38]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='failures', y='G3')
plt.title("Score Distribution by failures")
plt.show()
```



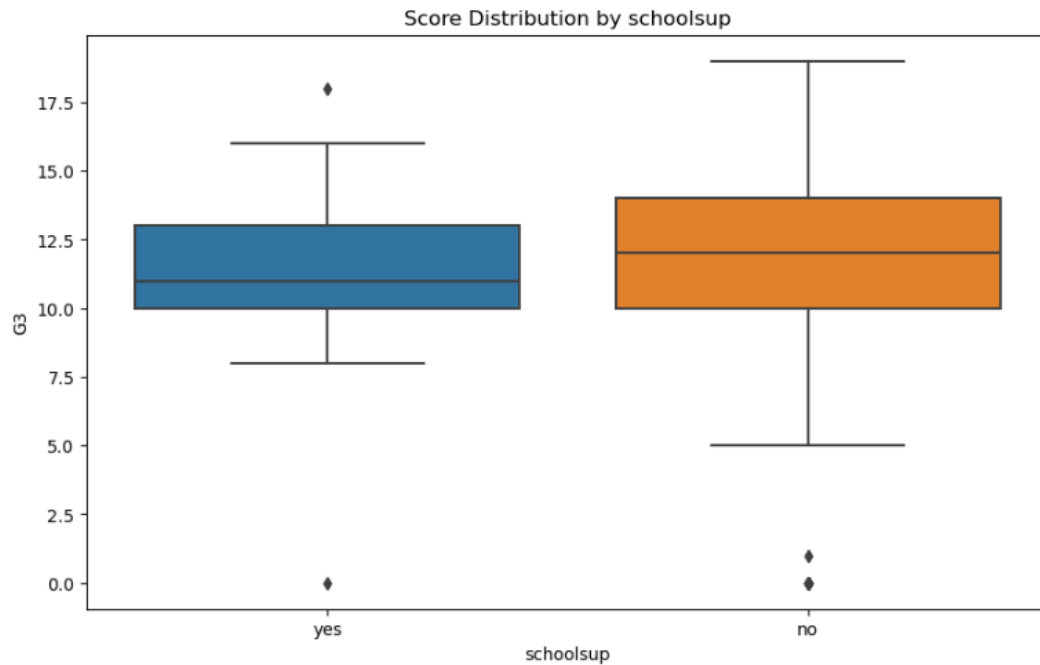
이전 수업에서 낙제한 경우가 있으면 성적이 크게 감소한다.

## 16. schoolsup

```
[39]: student_df['schoolsup'].value_counts()
```

```
[39]: schoolsup  
no    581  
yes    68  
Name: count, dtype: int64
```

```
[40]: plt.figure(figsize=(10, 6))  
sns.boxplot(data=student_df, x='schoolsup', y='G3')  
plt.title("Score Distribution by schoolsup")  
plt.show()
```

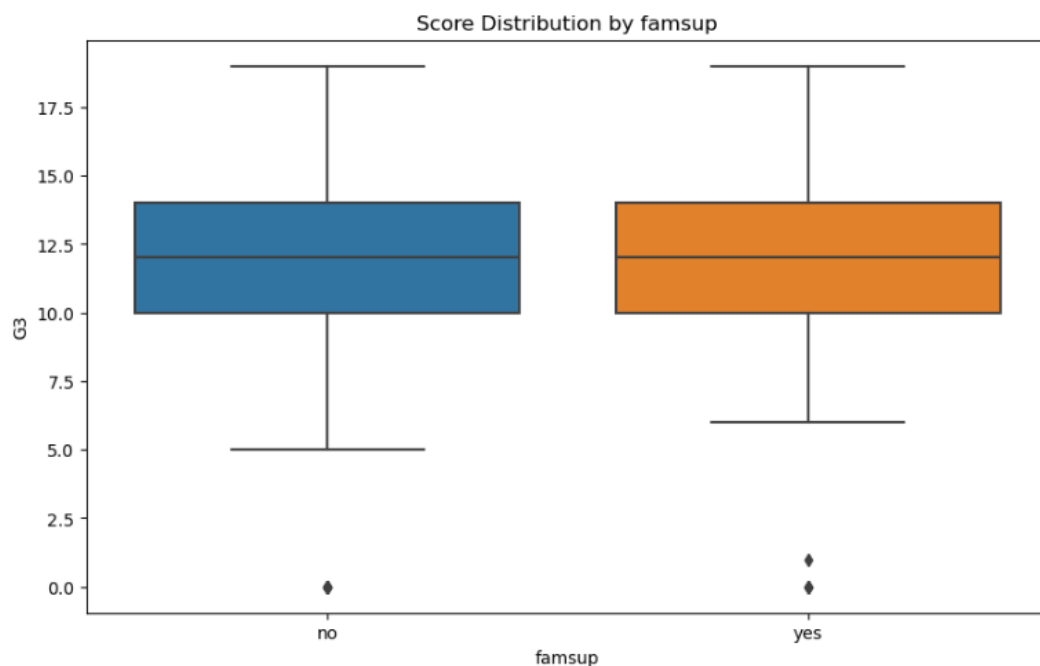


## 17. famsup

```
[41]: student_df['famsup'].value_counts()
```

```
[41]: famsup  
yes    398  
no    251  
Name: count, dtype: int64
```

```
[42]: plt.figure(figsize=(10, 6))  
sns.boxplot(data=student_df, x='famsup', y='G3')  
plt.title("Score Distribution by famsup")  
plt.show()
```



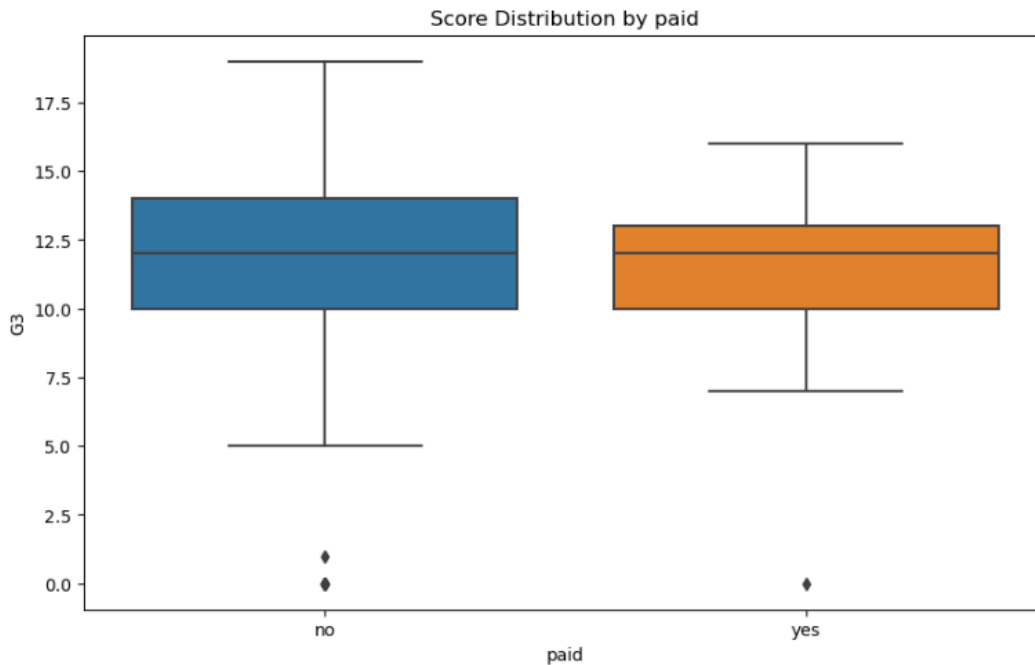
사교육은 성적에 그다지 영향을 주지 않는 것으로 보인다.

## 18. paid

```
[43]: student_df['paid'].value_counts()
```

```
[43]: paid
no      610
yes      39
Name: count, dtype: int64
```

```
[44]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='paid', y='G3')
plt.title("Score Distribution by paid")
plt.show()
```

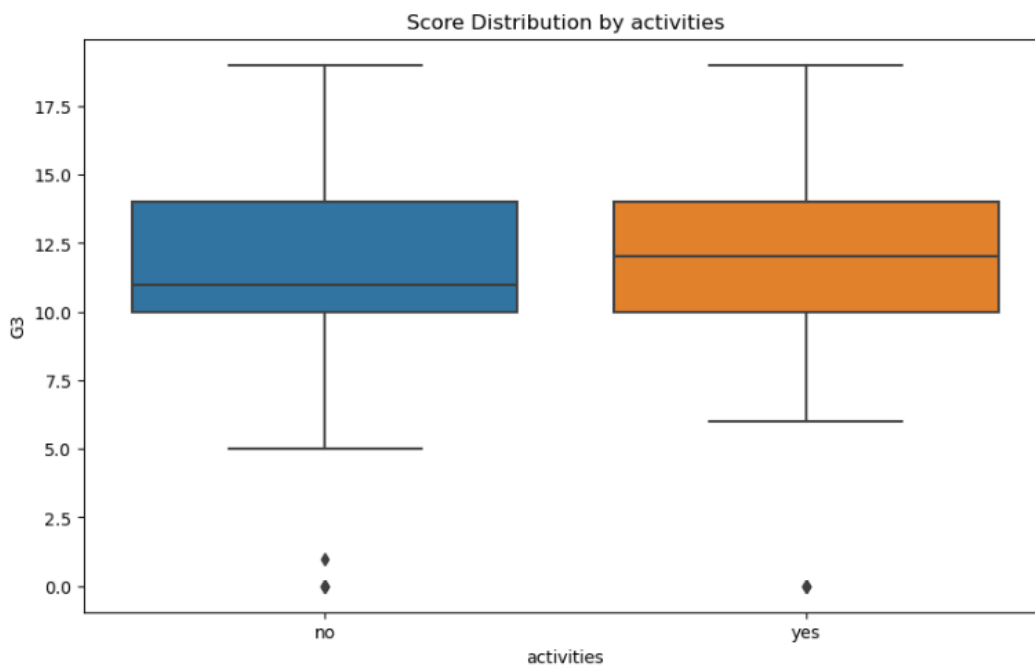


## 19. activities

```
[45]: student_df['activities'].value_counts()
```

```
[45]: activities
no      334
yes     315
Name: count, dtype: int64
```

```
[46]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='activities', y='G3')
plt.title("Score Distribution by activities")
plt.show()
```



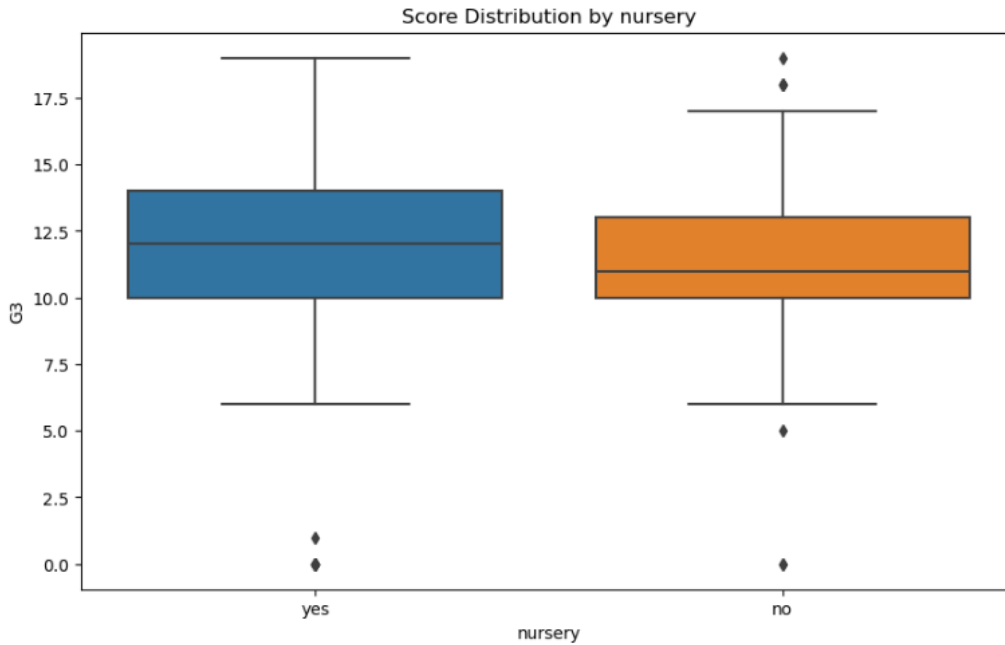
방과후 활동은 성적에 영향이 없음

## 20. nursery

```
[47]: student_df['nursery'].value_counts()

[47]: nursery
yes     521
no      128
Name: count, dtype: int64

[48]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='nursery', y='G3')
plt.title("Score Distribution by nursery")
plt.show()
```



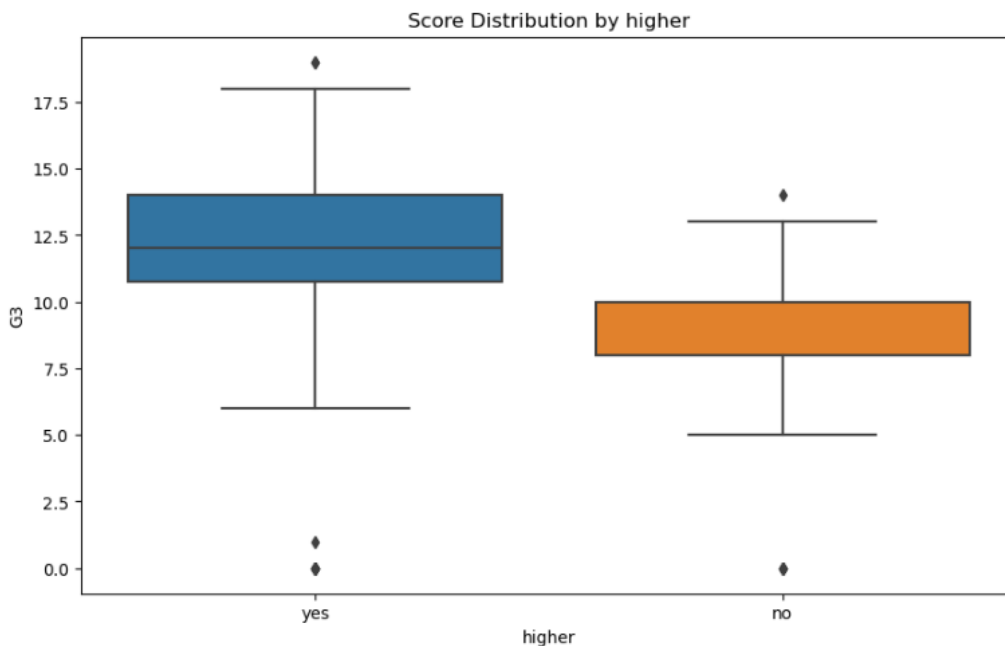
유치원에 다녔는지는 큰 영향이 없어보임

## 21. higher

```
[49]: student_df['higher'].value_counts()

[49]: higher
yes     580
no       69
Name: count, dtype: int64

[50]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='higher', y='G3')
plt.title("Score Distribution by higher")
plt.show()
```



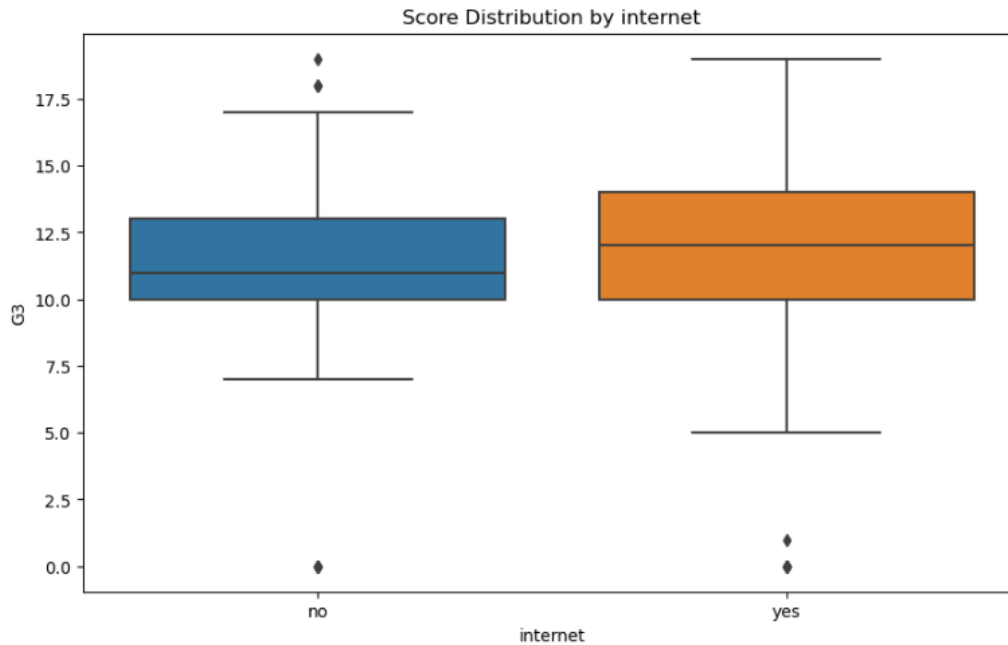
더 높은 수준의 교육을 원하는 학생이 성적이 확실히 높다

## 22. internet

```
[51]: student_df['internet'].value_counts()
```

```
[51]: internet
yes    498
no     151
Name: count, dtype: int64
```

```
[52]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='internet', y='G3')
plt.title("Score Distribution by internet")
plt.show()
```



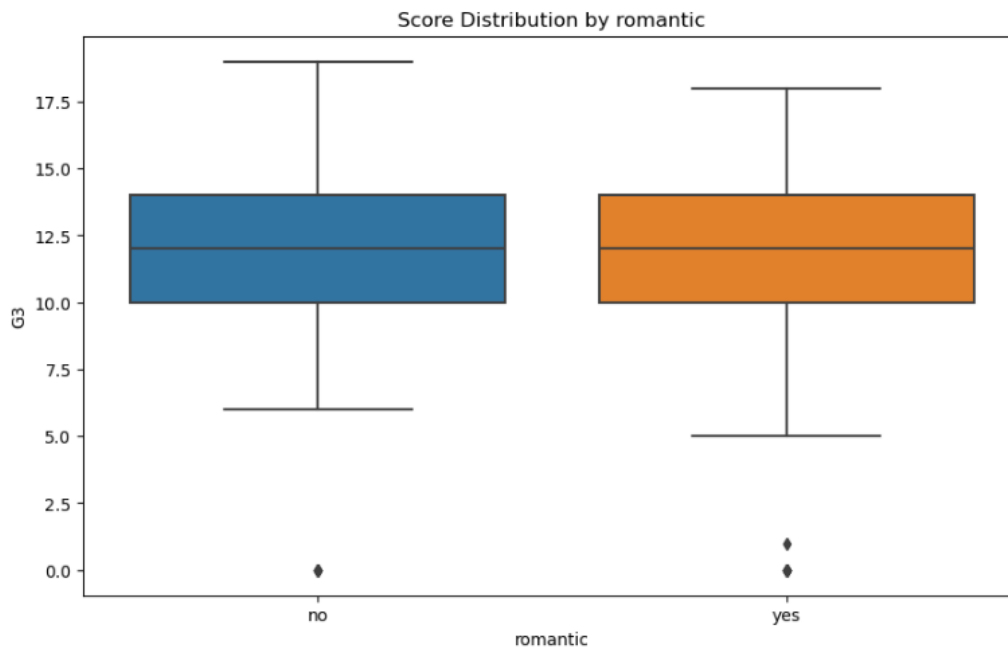
인터넷 여부는 성적에 큰 영향을 주지 않는다.

## 23. romantic

```
[53]: student_df['romantic'].value_counts()
```

```
[53]: romantic
no    410
yes   239
Name: count, dtype: int64
```

```
[54]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='romantic', y='G3')
plt.title("Score Distribution by romantic")
plt.show()
```



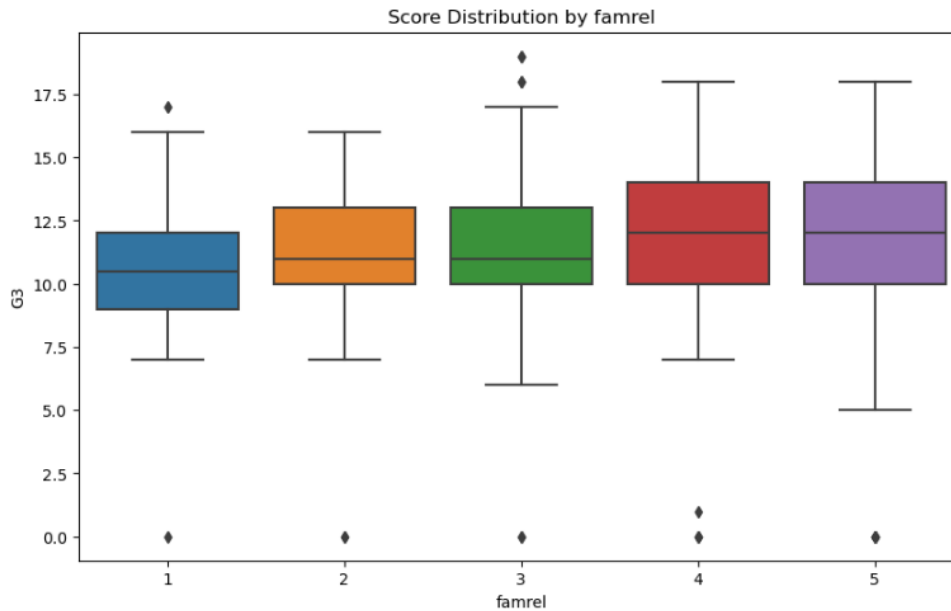
전혀 관계없는 변수

## 24. famrel

```
[55]: student_df['famrel'].value_counts()
```

```
[55]: famrel
4      317
5      180
3      101
2       29
1       22
Name: count, dtype: int64
```

```
[56]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='famrel', y='G3')
plt.title("Score Distribution by famrel")
plt.show()
```



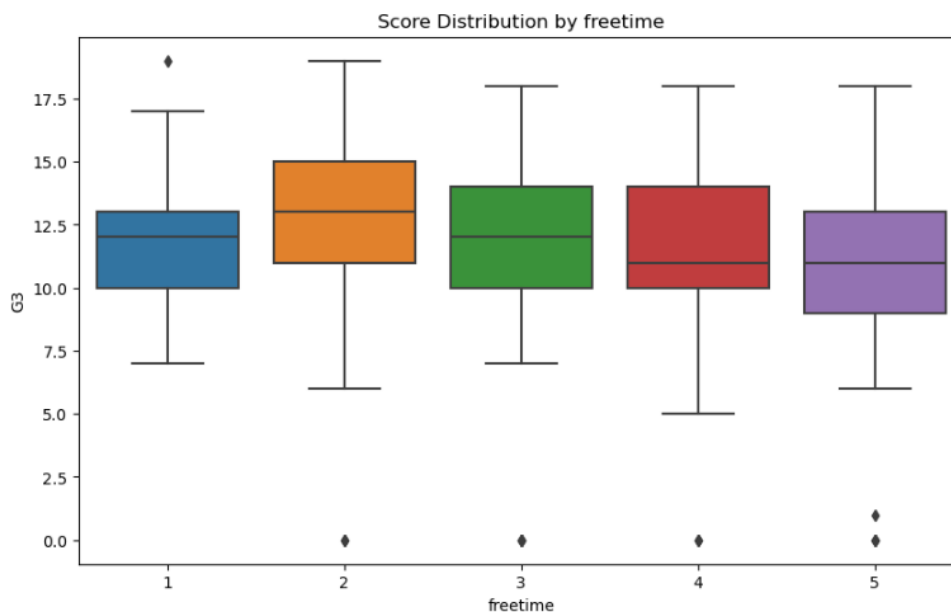
가족관계는 어느정도 보조변수로 사용할 수 있을 것으로 보인다

## 25. freetime

```
[57]: student_df['freetime'].value_counts()
```

```
[57]: freetime
3      251
4      178
2      107
5       68
1       45
Name: count, dtype: int64
```

```
[58]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='freetime', y='G3')
plt.title("Score Distribution by freetime")
plt.show()
```



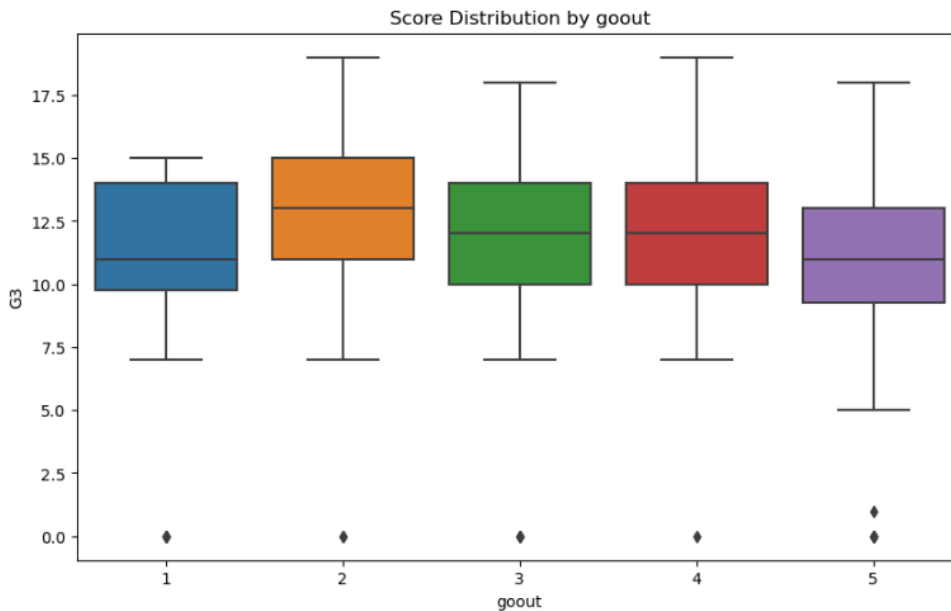
차이가 있기는 한데 일관적이지 않고 영향이 적을 것으로 생각됨

## 26. goout

```
[59]: student_df['goout'].value_counts()
```

```
[59]: goout
3    205
2    145
4    141
5    110
1     48
Name: count, dtype: int64
```

```
[60]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='goout', y='G3')
plt.title("Score Distribution by goout")
plt.show()
```



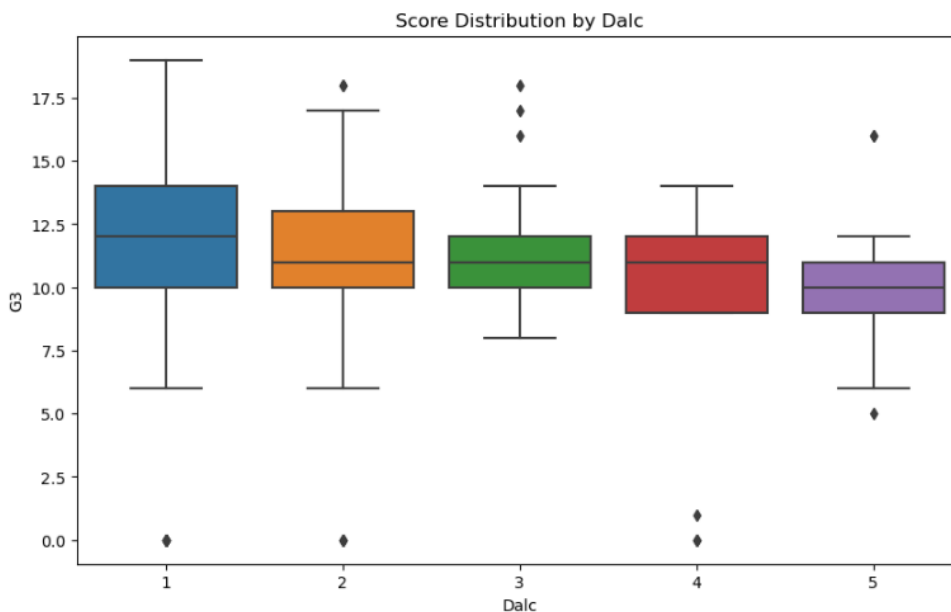
친구와 자주 노는지는 성적에 그다지 영향이 없어보인다

## 27. Dalc

```
[61]: student_df['Dalc'].value_counts()
```

```
[61]: Dalc
1    451
2    121
3     43
5     17
4     17
Name: count, dtype: int64
```

```
[62]: plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='Dalc', y='G3')
plt.title("Score Distribution by Dalc")
plt.show()
```



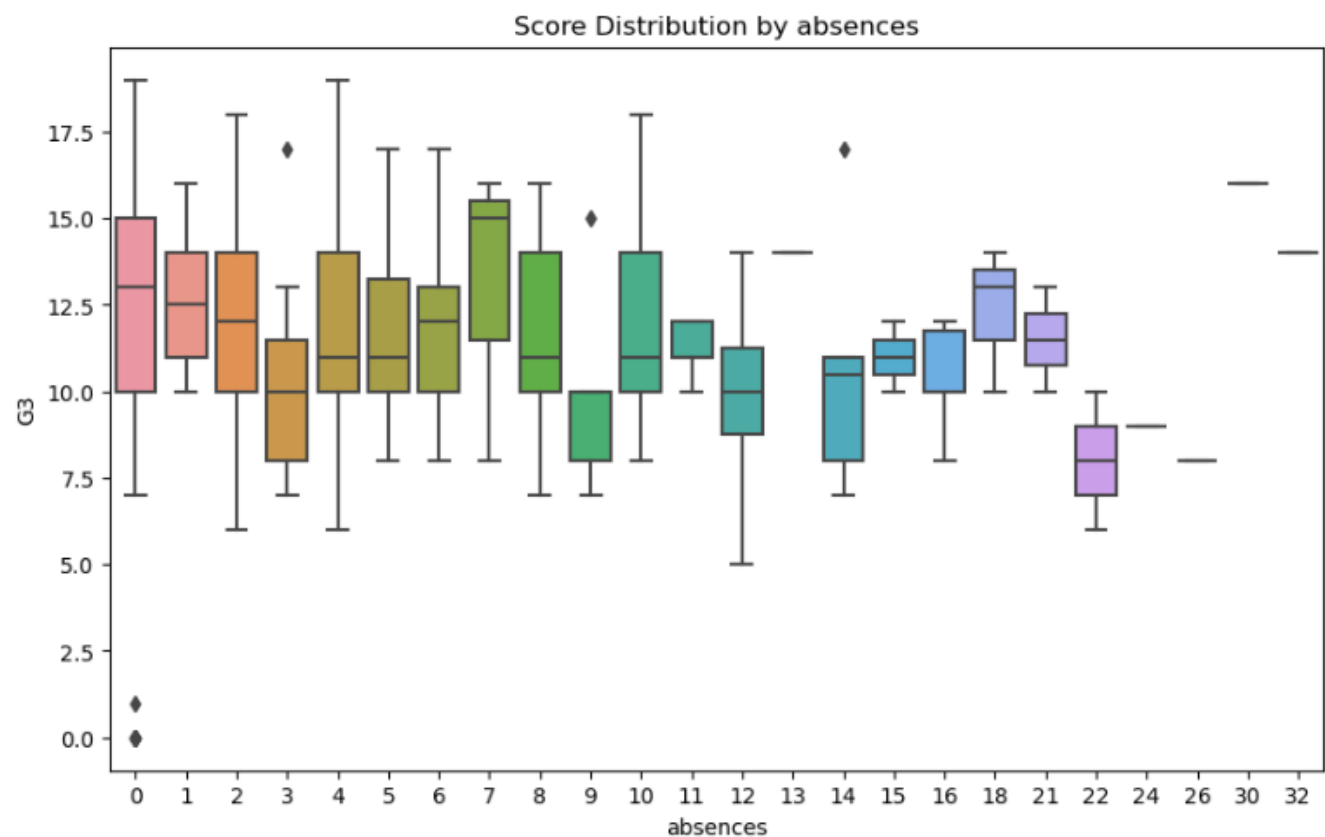
음주 비율이 낮은 학생이 더 좋은 성적을 받는 경향이 있다.

### 30. absences

```
student_df['absences'].value_counts()
```

```
absences
0      244
2      110
4       93
6       49
8       42
10      21
1       12
12      12
5       12
16      10
14       8
9        7
3        7
11       5
18       3
7        3
21       2
15       2
22       2
30       1
26       1
24       1
13       1
32       1
Name: count, dtype: int64
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=student_df, x='absences', y='G3')
plt.title("Score Distribution by absences")
plt.show()
```



결석횟수가 많을 수록 성적이 낮아지는 경향성이 있는 것 같다



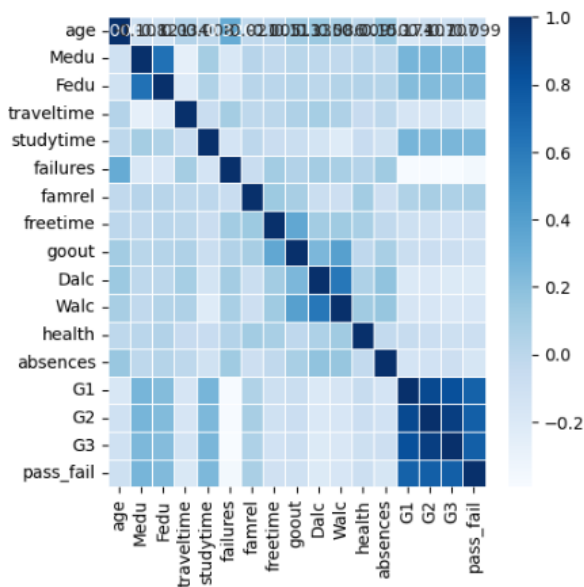
## - Correlation Matrix

```
[67]: student_df.corr(numeric_only=True)
```

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	health	absences	G1
age	1.000000	-0.107832	-0.121050	0.034490	-0.008415	0.319968	-0.020559	-0.004910	0.112805	0.134768	0.086357	-0.008750	0.149998	-0.174322
Medu	-0.107832	1.000000	0.647477	-0.265079	0.097006	-0.172210	0.024421	-0.019686	0.009536	-0.007018	-0.019766	0.004614	-0.008577	0.260472
Fedu	-0.121050	0.647477	1.000000	-0.208288	0.050400	-0.165915	0.020256	0.006841	0.027690	0.000061	0.038445	0.044910	0.029859	0.217501
traveltime	0.034490	-0.265079	-0.208288	1.000000	-0.063154	0.097730	-0.009521	0.000937	0.057454	0.092824	0.057007	-0.048261	-0.008149	-0.154120
studytime	-0.008415	0.097006	0.050400	-0.063154	1.000000	-0.147441	-0.004127	-0.068829	-0.075442	-0.137585	-0.214925	-0.056433	-0.118389	0.260875
failures	0.319968	-0.172210	-0.165915	0.097730	-0.147441	1.000000	-0.062645	0.108995	0.045078	0.105949	0.082266	0.035588	0.122779	-0.384210
famrel	-0.020559	0.024421	0.020256	-0.009521	-0.004127	-0.062645	1.000000	0.129216	0.089707	-0.075767	-0.093511	0.109559	-0.089534	0.048795
freetime	-0.004910	-0.019686	0.006841	0.000937	-0.068829	0.108995	0.129216	1.000000	0.346352	0.109904	0.120244	0.084526	-0.018716	-0.094497
goout	0.112805	0.009536	0.027690	0.057454	-0.075442	0.045078	0.089707	0.346352	1.000000	0.245126	0.388680	-0.015741	0.085374	-0.074053
Dalc	0.134768	-0.007018	0.000061	0.092824	-0.137585	0.105949	-0.075767	0.109904	0.245126	1.000000	0.616561	0.059067	0.172952	-0.195171
Walc	0.086357	-0.019766	0.038445	0.057007	-0.214925	0.082266	-0.093511	0.120244	0.388680	0.616561	1.000000	0.114988	0.156373	-0.155649
health	-0.008750	0.004614	0.044910	-0.048261	-0.056433	0.035588	0.109559	0.084526	-0.015741	0.059067	0.114988	1.000000	-0.030235	-0.051647
absences	0.149998	-0.008577	0.029859	-0.008149	-0.118389	0.122779	-0.089534	-0.018716	0.085374	0.172952	0.156373	-0.030235	1.000000	-0.147149
G1	-0.174322	0.260472	0.217501	-0.154120	0.260875	-0.384210	0.048795	-0.094497	-0.074053	-0.195171	-0.155649	-0.051647	-0.147149	1.000000
G2	-0.107119	0.264035	0.225139	-0.154489	0.240498	-0.385782	0.089588	-0.106678	-0.079469	-0.189480	-0.164852	-0.082179	-0.124745	0.864982
G3	-0.106505	0.240151	0.211800	-0.127173	0.249789	-0.393316	0.063361	-0.122705	-0.087641	-0.204719	-0.176619	-0.098851	-0.091379	0.826387
pass_fail	-0.098971	0.261387	0.231348	-0.185289	0.239055	-0.355557	0.074834	-0.113957	-0.106104	-0.206641	-0.169947	-0.084668	-0.160911	0.728893

```
[68]: plt.figure(figsize=(5, 5))
sns.heatmap(data=student_df.corr(numeric_only=True), annot=True, fmt='.3f', linewidths=0.5, cmap='Blues')
```

```
[68]: <Axes: >
```



당장 correlation matrix상에 G3와 관련이 크게 있어보이는 변수는 Medu, Fedu, Studytime정도 외에는 없다. Dalc와 Walc는 다중공선성 문제를 해결하기위해 하나만 쓰는게 좋아보인다.

자주 나가서 노는 학생은 음주비율도 높다.

부모님의 학위는 서로 어느정도 상관관계가 존재한다.

- EDA, 피쳐엔지니어링 요약

Feature 1	Action 1	Feature 2	Action 2
school	사용	sex	사용
age	새로운 피쳐 추가 (19세 이상 여부)	address	제거
famsize	제거	Pstatus	사용
Medu	사용	Fedu	사용
Mjob	사용	Fjob	변환 후 사용 (부모님 중 교사 여부)
reason	제거	guardian	새로운 피쳐 추가 (OTHER 여부)
traveltime	사용	studytime	사용
failures	사용	schoolsup	사용
famsup	사용	paid	사용
activities	제거	nursery	제거
higher	사용	internet	제거
romantic	제거	famrel	사용
freetime	제거	goout	제거
Dalc	사용	Walc	제거 (Dalc만 사용)
health	제거	absences	사용

### 3.2 로지스틱회귀 모델 구축

- sklearn.linear\_model의 LogisticRegression은 모델의 통계적해석을 위한 지표를 제공하지 않는다.

따라서 Statsmodels 라이브러리를 사용

```
[88]: import statsmodels.api as sm

X_train_with_const = sm.add_constant(X_train)
y_train = y_train

# 로지스틱 회귀 모델 구축, 학습
logit_model = sm.Logit(y_train, X_train_with_const)
logit_results = logit_model.fit()
```

```
Optimization terminated successfully.
    Current function value: 0.481219
    Iterations 7
```

```
[89]: from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score

# 예측 확률 계산 (0 ~ 1 사이의 값)
y_val_pred_prob = logit_results.predict(sm.add_constant(X_val))

# 임계값 0.5를 기준으로 클래스 예측 (0 또는 1)
y_val_pred = np.where(y_val_pred_prob >= 0.5, 1, 0)
```

모델의 통계 지표 확인

```
[90]: print(logit_results.summary())
```

```

                        Logit Regression Results
=====
Dep. Variable:          pass_fail    No. Observations:          454
Model:                  Logit        Df Residuals:              435
Method:                 MLE          Df Model:                  18
Date:                  Thu, 14 Nov 2024    Pseudo R-squ.:            0.3038
Time:                  23:53:19          Log-Likelihood:           -218.47
converged:              True            LL-Null:                  -313.82
Covariance Type:        nonrobust        LLR p-value:              7.179e-31
=====

```

	coef	std err	z	P> z	[0.025	0.975]
const	-0.0011	0.138	-0.008	0.994	-0.271	0.269
x1	-0.5268	0.131	-4.027	0.000	-0.783	-0.270
x2	-0.1911	0.131	-1.458	0.145	-0.448	0.066
x3	0.0280	0.121	0.231	0.818	-0.210	0.266
x4	0.1139	0.173	0.657	0.511	-0.226	0.454
x5	0.2857	0.162	1.768	0.077	-0.031	0.602
x6	-0.1447	0.120	-1.206	0.228	-0.380	0.090
x7	0.3456	0.132	2.620	0.009	0.087	0.604
x8	-1.1724	0.266	-4.414	0.000	-1.693	-0.652
x9	0.4764	0.118	4.024	0.000	0.244	0.708
x10	-0.0618	0.123	-0.503	0.615	-0.303	0.179
x11	-0.0256	0.114	-0.224	0.823	-0.250	0.199
x12	-0.5727	0.166	-3.453	0.001	-0.898	-0.248
x13	0.1480	0.121	1.227	0.220	-0.088	0.385
x14	-0.4101	0.136	-3.022	0.003	-0.676	-0.144
x15	-0.2909	0.125	-2.320	0.020	-0.537	-0.045
x16	-0.0605	0.189	-0.319	0.750	-0.432	0.311
x17	-0.0662	0.146	-0.455	0.649	-0.351	0.219
x18	0.2915	0.175	1.669	0.095	-0.051	0.634

## - 해석

Coef는 해당 변수가 예측 대상인 pass\_fail에 미치는 영향을 나타낸다. 계수가 양수이면 긍정적인 영향을, 음수이면 부정적인 영향을 준다. 예를 들어 x8 (failures)이 1 증가하면 pass\_fail이 1.1724 log-odds 만큼 감소한다. 위 결과에 따르면 x5 (아버지의 학력)과 x9 (추가 사교육)등은 pass\_fail 양의 상관관계가 있으며 x15 (결석 횟수)는 pass\_fail과 음의 상관관계가 있음을 확인가능하다.

p-value는 각 변수의 통계적 유의성을 보여주는 지표이다. x1(다니는 학교 이름), x7(학습 시간), x9(추가 사교육), x13(가족과의 관계)등의 변수는 p-value가 0.05 미만으로, pass\_fail과 유의미한 관계를 가지고 있을 가능성이 크다. 그러나 x4(어머니의 학력), x12(고등교육 희망여부) 등의 변수는 p-value가 높아서 통계적으로 유의미하지 않다고 볼 수 있다.

전체 모델의 p-value는 7.179e-31로 매우 낮아서 전체 모델은 통계적으로 유의미하여, pass\_fail과 피쳐들 간에 의미 있는 관계가 있음을 보여준다.

Pseudo R-squared 값은 0.3038로, 모델이 종속 변수의 변동성을 약 30.38% 설명하고 있다.

(로지스틱 회귀에서는 일반적인 R-squared값이 적게 나오는 경향성이 있음)

### 3.3 로지스틱회귀 모델 예측, 평가

```
[91]: con_mat = confusion_matrix(y_val, y_val_pred)
      print("Confusion Matrix:\n", con_mat)

      # 정밀도, 재현율, F1 점수 계산
      precision = precision_score(y_val, y_val_pred)
      recall = recall_score(y_val, y_val_pred)
      f1 = f1_score(y_val, y_val_pred)

      print("Precision:\n", precision)
      print("Recall:\n", recall)
      print("F1 Score:\n", f1)

      Confusion Matrix:
      [[20 14]
       [ 7 24]]
      Precision:
      0.631578947368421
      Recall:
      0.7741935483870968
      F1 Score:
      0.6956521739130435
```

- 혼동행렬 :

**True Negative (TN):** 20 — 실제로 불합격인데, 불합격이라고 예측한 수 **20명**

**False Positive (FP):** 14 — 실제로 불합격인데, 합격이라고 잘못 예측한 수 **14명**

**False Negative (FN):** 7 — 실제로 합격인데, 불합격이라고 잘못 예측한 수 **7명**

**True Positive (TP):** 24 — 실제로 합격인데, 합격이라고 예측한 수 **24명**

- **정밀도 (Precision):** 모델이 PASS로 예측했을 경우 실제로 PASS인 비율이 **63.16%**

- **재현율 (Recall):** 실제 PASS였을때, 모델이 PASS로 예측한 비율 **77.42%**

- **F1 Score:** precision과 recall의 조화평균으로 분류 모델의 대략적인 성능을 의미 **69.57%**

## 3.4 Shrinkage models

### Parameters:

**penalty** : {'l1', 'l2', 'elasticnet', None}, default='l2'

Specify the norm of the penalty:

- **None** : no penalty is added;
- **'l2'** : add a L2 penalty term and it is the default choice;
- **'l1'** : add a L1 penalty term;
- **'elasticnet'** : both L1 and L2 penalty terms are added.

solver	penalty
'lbfgs'	'l2', None
'liblinear'	'l1', 'l2'
'newton-cg'	'l2', None
'newton-cholesky'	'l2', None
'sag'	'l2', None
'saga'	'elasticnet', 'l1', 'l2', None

LogisticRegression의 penalty 항목을 사용하여 Lasso, Ridge, Elasticnet을 분류모델에서 구현할 수 있다.

```
from sklearn.linear_model import LogisticRegression

# Lasso
lasso_model = LogisticRegression(penalty='l1', solver='liblinear', max_iter=1000, random_state=42, C=0.05)
lasso_model.fit(X_train, y_train)
y_val_pred_prob_lasso = lasso_model.predict_proba(X_val)[: , 1]
y_val_pred_lasso = np.where(y_val_pred_prob_lasso >= 0.5, 1, 0)
precision_lasso = precision_score(y_val, y_val_pred_lasso)
recall_lasso = recall_score(y_val, y_val_pred_lasso)
f1_lasso = f1_score(y_val, y_val_pred_lasso)
confusion_lasso = confusion_matrix(y_val, y_val_pred_lasso)

print("Lasso Model:")
print("Precision:", precision_lasso)
print("Recall:", recall_lasso)
print("F1 Score:", f1_lasso)
print("Confusion Matrix:\n", confusion_lasso)
print()

# Ridge
ridge_model = LogisticRegression(penalty='l2', solver='liblinear', max_iter=1000, random_state=42, C=0.05)
ridge_model.fit(X_train, y_train)
y_val_pred_prob_ridge = ridge_model.predict_proba(X_val)[: , 1]
y_val_pred_ridge = np.where(y_val_pred_prob_ridge >= 0.5, 1, 0)
precision_ridge = precision_score(y_val, y_val_pred_ridge)
recall_ridge = recall_score(y_val, y_val_pred_ridge)
f1_ridge = f1_score(y_val, y_val_pred_ridge)
confusion_ridge = confusion_matrix(y_val, y_val_pred_ridge)

print("Ridge Model:")
print("Precision:", precision_ridge)
print("Recall:", recall_ridge)
print("F1 Score:", f1_ridge)
print("Confusion Matrix:\n", confusion_ridge)
print()

# ElasticNet
elasticnet_model = LogisticRegression(penalty='elasticnet', solver='saga', l1_ratio=0.5, max_iter=1000, random_state=42, C=0.05)
elasticnet_model.fit(X_train, y_train)
y_val_pred_prob_elasticnet = elasticnet_model.predict_proba(X_val)[: , 1]
y_val_pred_elasticnet = np.where(y_val_pred_prob_elasticnet >= 0.5, 1, 0)
precision_elasticnet = precision_score(y_val, y_val_pred_elasticnet)
recall_elasticnet = recall_score(y_val, y_val_pred_elasticnet)
f1_elasticnet = f1_score(y_val, y_val_pred_elasticnet)
confusion_elasticnet = confusion_matrix(y_val, y_val_pred_elasticnet)

print("ElasticNet:")
print("Precision:", precision_elasticnet)
print("Recall:", recall_elasticnet)
print("F1 Score:", f1_elasticnet)
print("Confusion Matrix:\n", confusion_elasticnet)
```

```
Lasso Model:
Precision: 0.5853658536585366
Recall: 0.7741935483870968
F1 Score: 0.6666666666666666
Confusion Matrix:
[[17 17]
 [ 7 24]]
```

```
Ridge Model:
Precision: 0.631578947368421
Recall: 0.7741935483870968
F1 Score: 0.6956521739130435
Confusion Matrix:
[[20 14]
 [ 7 24]]
```

```
ElasticNet:
Precision: 0.6097560975609756
Recall: 0.8064516129032258
F1 Score: 0.6944444444444445
Confusion Matrix:
[[18 16]
 [ 6 25]]
```

## - 결과해석

C 파라미터를 0.05로 하여 규제를 다소 강하게 적용했다.

Lasso 모델은 기존의 모델보다 불합격을 올바르게 예측하는 성능이 감소하였다.

Ridge는 기존 모델과 동일한 분류성능을 보였다.

ElasticNet은 합격을 예측하는 성능을 증가하였으나 불합격을 올바르게 예측하는 성능이 감소하였다.

결론적으로, 규제모델을 사용해도 모델 성능향상에 그다지 영향이 없었고, 오히려 열화했다.

이는 EDA 과정에서 엄선한 피쳐들이 pass\_fail 예측에 대부분 주요한 역할을 했거나, 데이터 셋에 비선형적인 관계가 있는 것으로 추정된다.