In [32]:
```
#머신러닝 1 EDA 진행하고 차원 축소 필요한지 판다.
# cormap 보고 상관성이 많으면 pca 진행
import pandas as pd
import matplotlib.pyplot as plt

df=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/28_problem1.csv")
df.isna().sum()
```

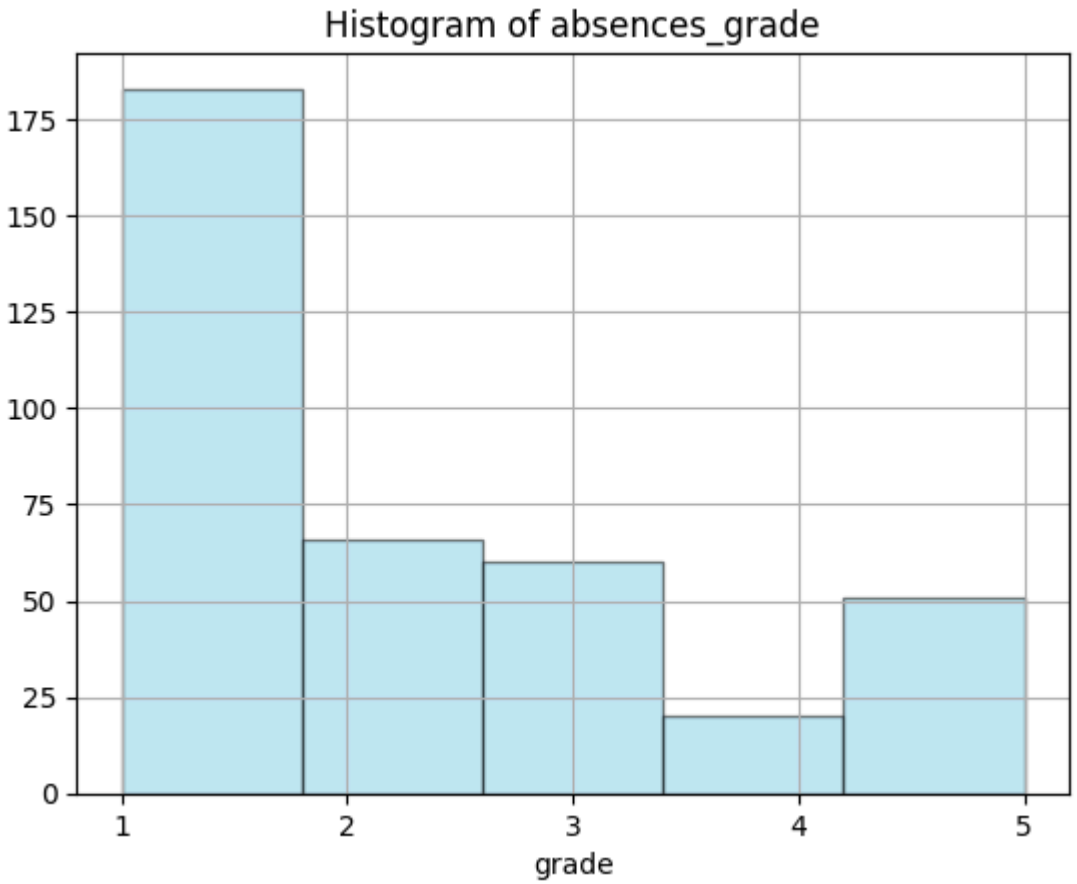
Out[32]:
```
school            0
sex               0
age               0
address           0
famsize           0
Pstatus           0
Medu              0
Fedu              0
reason            0
guardian          0
traveltime        0
studytime         0
failures          0
paid              0
activities        0
famrel            0
freetime          0
absences_grade    15
dtype: int64
```

In [33]:
```
df['absences_grade']
```

Out[33]:
```
0      3.0
1      2.0
2      4.0
3      1.0
4      2.0
      ...
390    5.0
391    2.0
392    2.0
393    1.0
394    2.0
Name: absences_grade, Length: 395, dtype: float64
```

In [34]:
```
plt.figure()
plt.hist(df['absences_grade'],bins=5,color='skyblue',edgecolor='black',alpha=0.5)
```

```
plt.title('Histogram of absences_grade')
plt.xlabel('grade')
plt.xticks(range(1,6))
plt.grid(True)
plt.show()
```



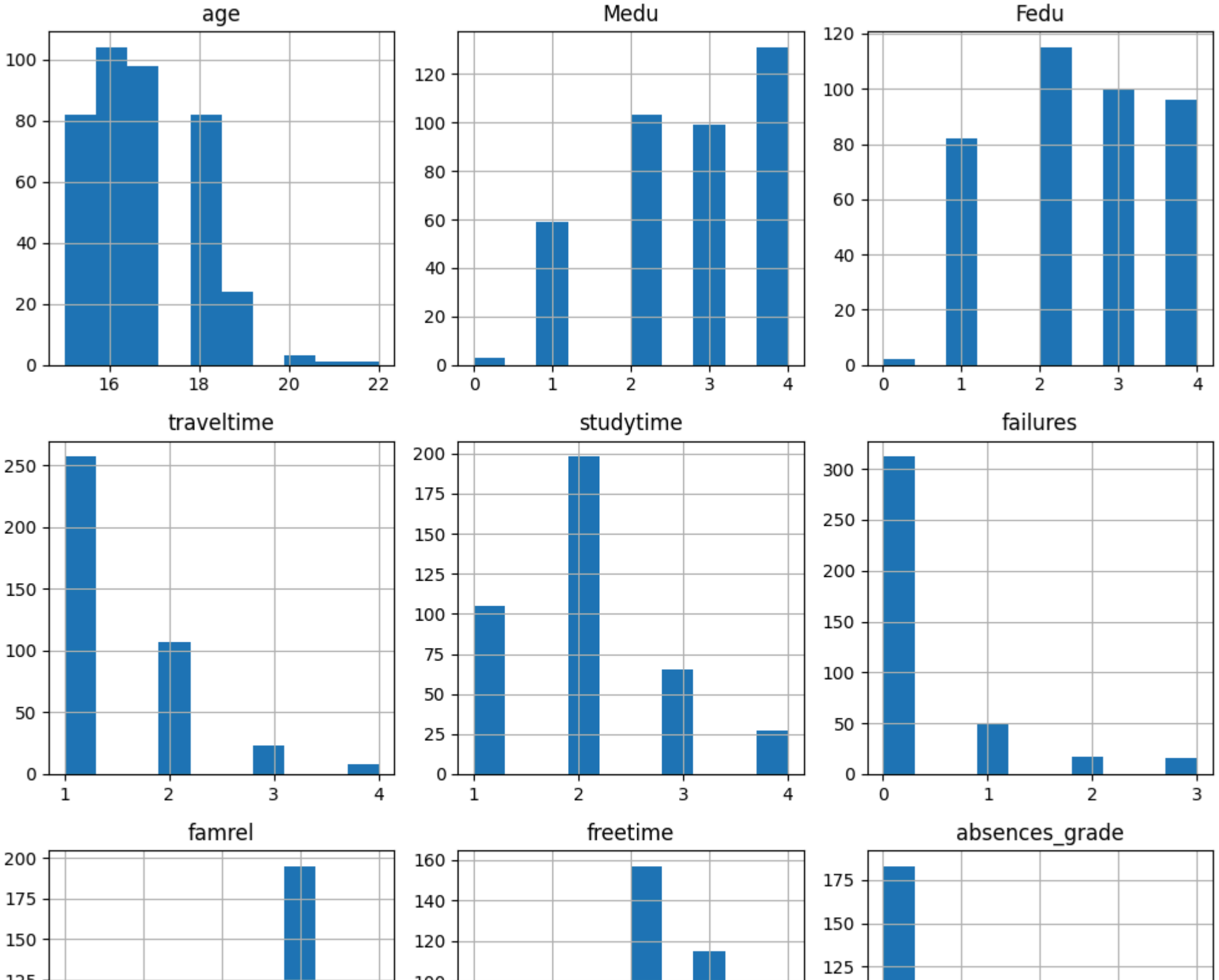
In [35]:
```
df.describe()
```

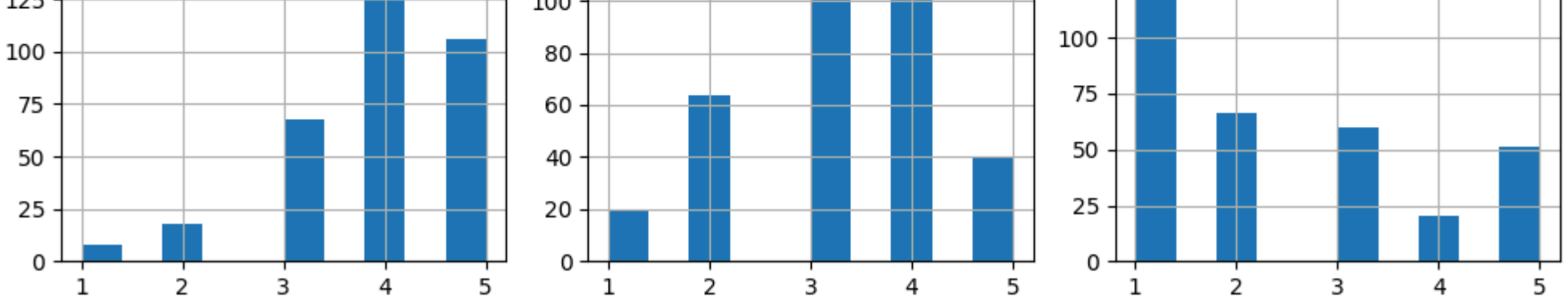
Out[35]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	absences_grade
count	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	395.000000	380.000000
mean	16.696203	2.749367	2.521519	1.448101	2.035443	0.334177	3.944304	3.235443	2.184211
std	1.276043	1.094735	1.088201	0.697505	0.839240	0.743651	0.896659	0.998862	1.424536
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000
25%	16.000000	2.000000	2.000000	1.000000	1.000000	0.000000	4.000000	3.000000	1.000000
50%	17.000000	3.000000	2.000000	1.000000	2.000000	0.000000	4.000000	3.000000	2.000000
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000	4.000000	3.000000
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000	5.000000	5.000000

In [36]:

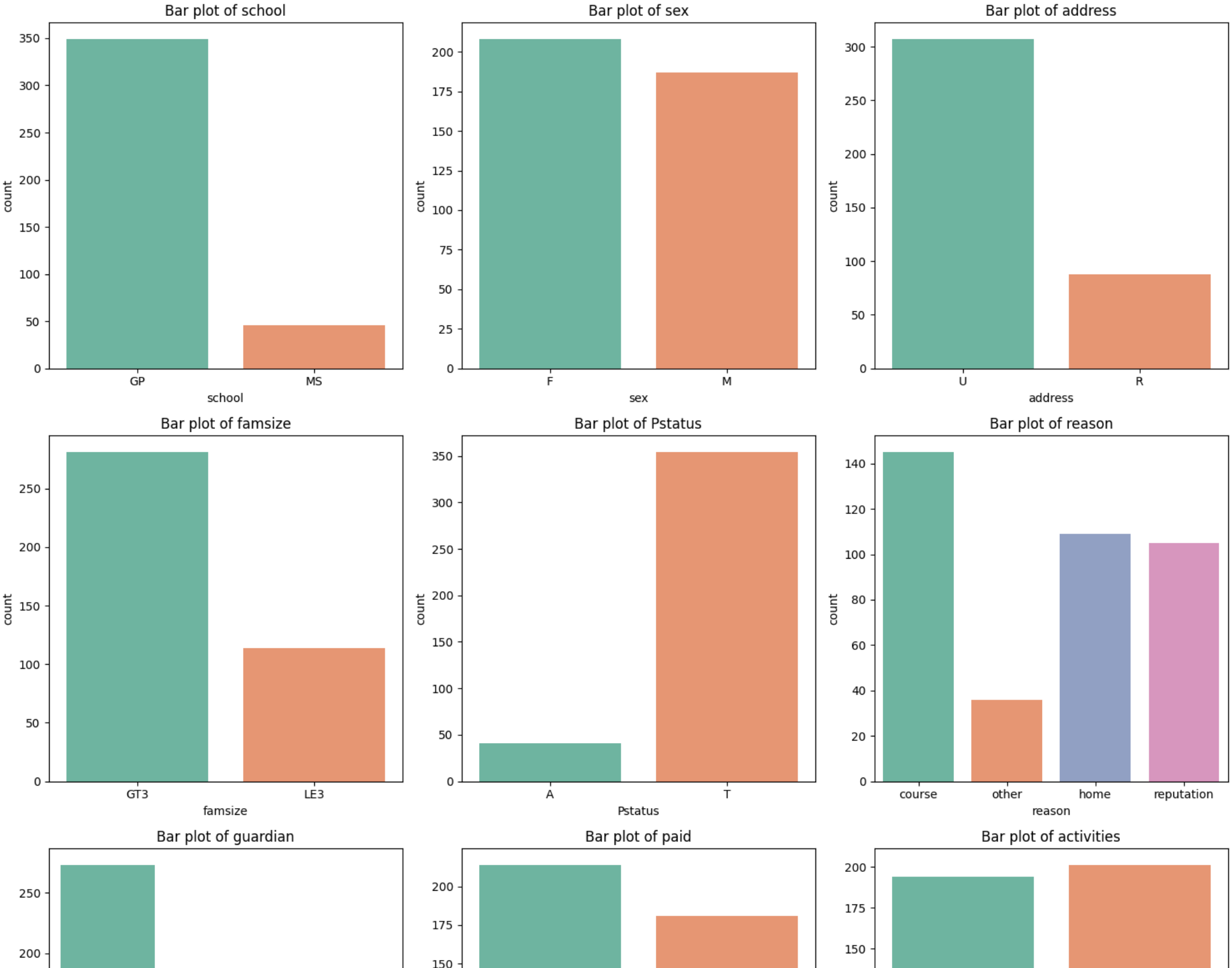
```
df.hist(figsize=(10,10))
plt.tight_layout()
plt.show()
```

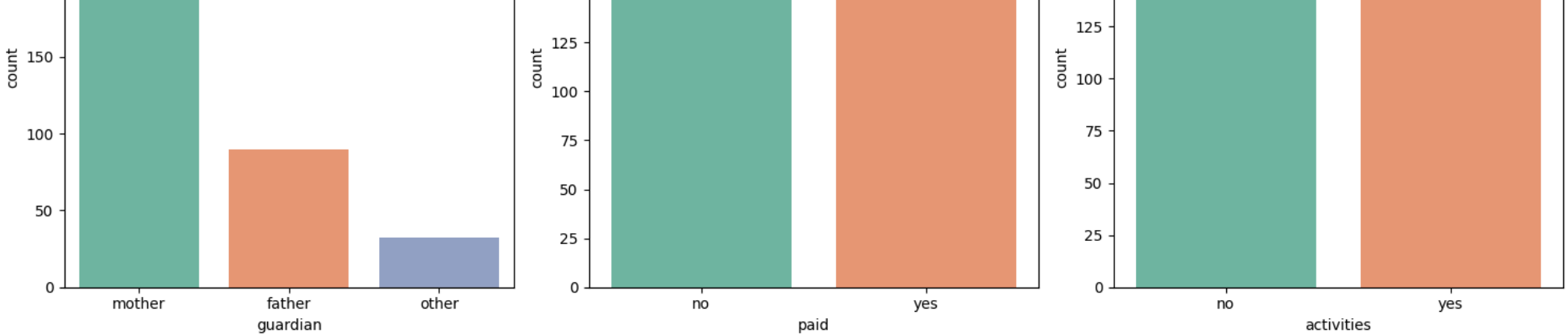

age
Medu
Fedu
traveltime
studytime
failures
famrel
freetime
absences_grade



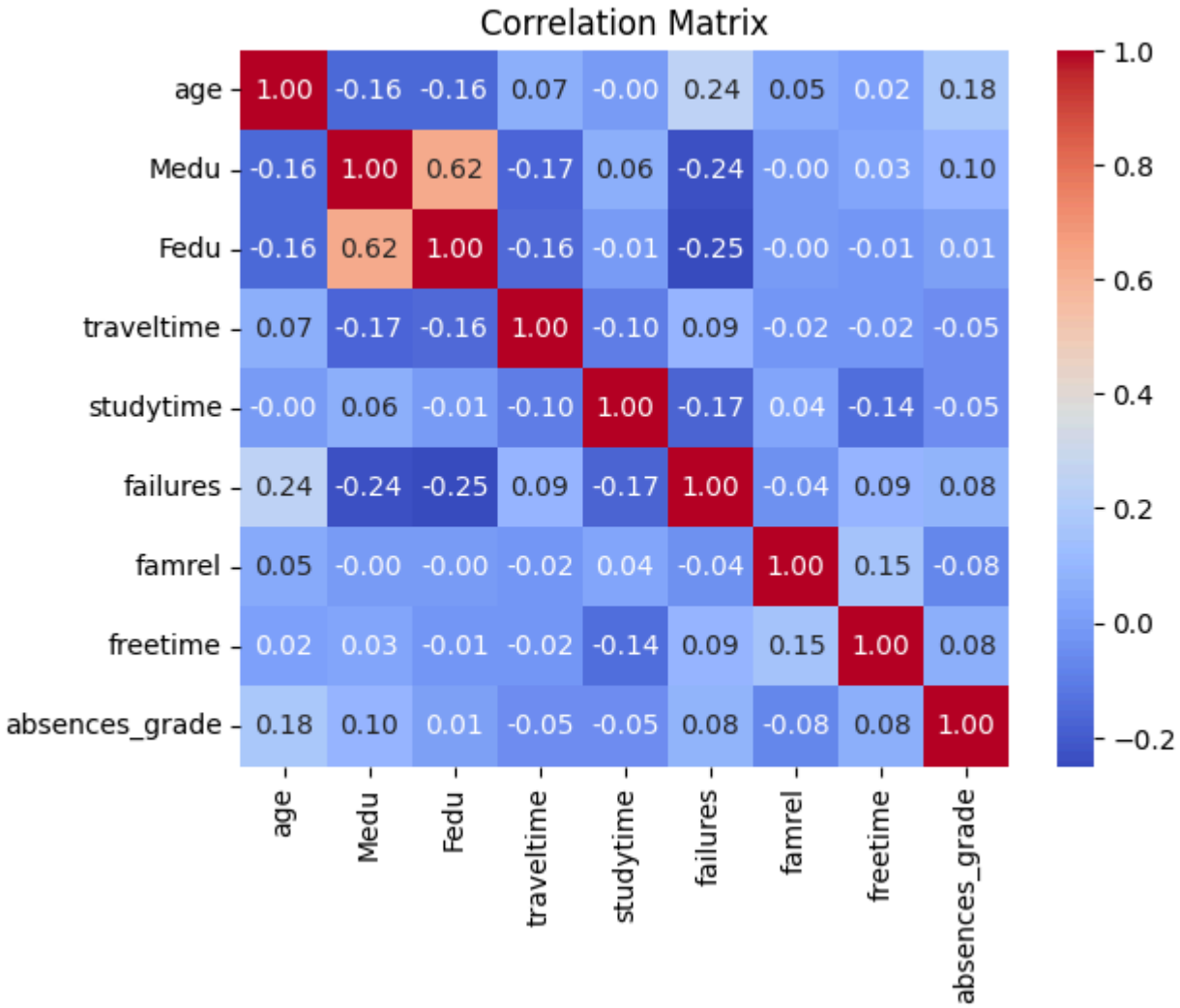
In [37]:

```
import seaborn as sns
categoriical_features= df.select_dtypes(include=['object']).columns
num_col=3
num_rows=(len(categoriical_features)-1)//num_col+1
plt.figure(figsize=(15,5*num_rows))
for i, col in enumerate(categoriical_features,start=1):
    plt.subplot(num_rows,num_col,i)
    sns.countplot(data=df,x=col,hue=col, palette='Set2')
    plt.title(f'Bar plot of {col}')
plt.tight_layout()
plt.show()
```

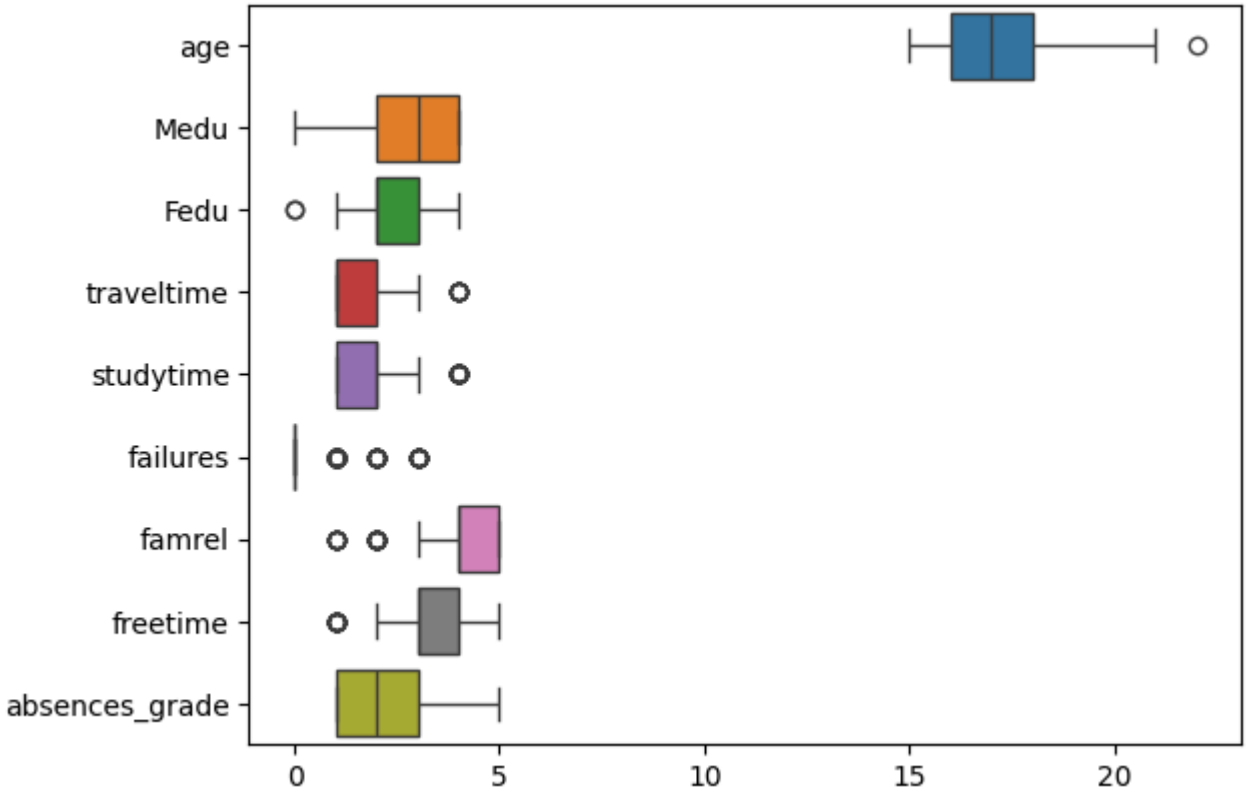

Bar plot of school
count
school
GP
MS
Bar plot of sex
count
sex
F
M
Bar plot of address
count
address
U
R
Bar plot of famsize
count
famsize
GT3
LE3
Bar plot of Pstatus
count
Pstatus
A
T
Bar plot of reason
count
reason
course
other
home
reputation
Bar plot of guardian
Bar plot of paid
Bar plot of activities



In [38]:
```
plt.figure()
numeric_features= df.select_dtypes(exclude=['object']).columns
sns.heatmap(df[numeric_features].corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



In [39]:
```
plt.figure()
sns.boxplot(data=df,orient='h')
plt.show()
```



In [40]:
```
#머신러닝 2 데이터 품질 개선할 수 있는 방법을 제안하고 데이터 세트를 재생성하시오
# 결측치를  채우기
df['absences_grade']= df['absences_grade'].fillna(df['absences_grade'].mean())
# 범주형 인코딩
df_encode= pd.get_dummies(df, columns=categoriical_features, drop_first=True)

df_encode=df_encode.replace(True, 1)
df_encode=df_encode.replace(False, 0)
```

In [41]:
```
# 2에서 제시한 방법이 데이터가 과적합이 된다는 가정하에 과적합을 해결할수 있는 2가지 방안
# 규제 추가
# 장점 복잡도 감소, 파라미터가 지나치게 크게 하는걸 방지
# 단점 규제 강도를 잘못 설정하면 성능 저하
# 교차 검증
# 장점 모델의 일반화 성능을 평가하는데 도움
# 단점 계산 비용 증가

# 2 데이터 기준으로 rf nn lgbm 등 모델링 후 성능 비교
#f1 스코어 기준이니깐 absences_grade를 범주형으로 변환
df_encode['absences_grade']= df_encode['absences_grade'].round().astype(int)
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from lightgbm import LGBMClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings('ignore')
X=df_encode.drop('absences_grade',axis=1)
y=df_encode['absences_grade']
X_train, X_test, y_train, y_test= train_test_split(X,y,test_size=0.2, random_state=42)

rf=RandomForestClassifier(random_state=42)
rf.fit(X_train,y_train)
y_pred_rf= rf.predict(X_test)
f1_rf= f1_score(y_test,y_pred_rf, average='weighted')
print("Random Forest F1 Score:", f1_rf)
print(classification_report(y_test,y_pred_rf))

nn= MLPClassifier(random_state=42, max_iter=500)
nn.fit(X_train,y_train)
y_pred_nn= nn.predict(X_test)
f1_nn= f1_score(y_test,y_pred_nn, average='weighted')
print("Neural Network F1 Score:", f1_nn)
print(classification_report(y_test,y_pred_nn))

lgbm= LGBMClassifier(random_state=42,verbose=-1)
lgbm.fit(X_train,y_train)
y_pred_lgbm= lgbm.predict(X_test)
f1_lgbm= f1_score(y_test,y_pred_lgbm, average='weighted')
print("LightGBM F1 Score:", f1_lgbm)
print(classification_report(y_test,y_pred_lgbm))
```

```
Random Forest F1 Score: 0.3504864774264744
              precision    recall  f1-score   support

           1       0.48      0.82      0.60        38
           2       0.27      0.25      0.26        12
           3       0.33      0.06      0.11        16
           4       0.00      0.00      0.00         5
           5       0.00      0.00      0.00         8

    accuracy                           0.44        79
   macro avg       0.22      0.23      0.19        79
weighted avg       0.34      0.44      0.35        79

Neural Network F1 Score: 0.4181560820801328
              precision    recall  f1-score   support

           1       0.57      0.61      0.59        38
           2       0.25      0.42      0.31        12
           3       0.36      0.25      0.30        16
           4       0.00      0.00      0.00         5
           5       0.29      0.25      0.27         8

    accuracy                           0.43        79
   macro avg       0.29      0.30      0.29        79
weighted avg       0.42      0.43      0.42        79

LightGBM F1 Score: 0.3297663369145126
              precision    recall  f1-score   support

           1       0.47      0.58      0.52        38
           2       0.19      0.25      0.21        12
           3       0.25      0.12      0.17        16
           4       0.00      0.00      0.00         5
           5       0.17      0.12      0.14         8

    accuracy                           0.35        79
   macro avg       0.21      0.22      0.21        79
weighted avg       0.32      0.35      0.33        79
```

In [51]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
from torch.utils.data import TensorDataset, DataLoader

# ======================
# 1️⃣ 데이터 준비
# ======================
iris = load_iris()
X = iris.data
y = iris.target

# train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 표준화
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# numpy → torch tensor
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)

# DataLoader 구성
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# ======================
# 2️⃣ 모델 정의
# ======================
class IrisNet(nn.Module):
    def __init__(self, input_dim, hidden1=64, hidden2=32, output_dim=3, dropout_p=0.5):
        super(IrisNet, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden1),
            nn.ReLU(),
            nn.Dropout(dropout_p),
            nn.Linear(hidden1, hidden2),
            nn.ReLU(),
            nn.Dropout(dropout_p),
            nn.Linear(hidden2, output_dim),
```

```
            nn.Softmax(dim=1)
        )

    def forward(self, x):
        return self.model(x)

model = IrisNet(input_dim=X_train.shape[1])
print(model)

# ======================
# 3️⃣ 손실함수 & 옵티마이저
# ======================
criterion = nn.CrossEntropyLoss()   # categorical_crossentropy와 동일
optimizer = optim.Adam(model.parameters(), lr=0.001)

# ======================
# 4️⃣ 학습 루프
# ======================
num_epochs = 100
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0

    for batch_X, batch_y in train_loader:
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    # 검증 정확도 계산
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for X_val, y_val in test_loader:
            preds = model(X_val)
            predicted = torch.argmax(preds, dim=1)
            total += y_val.size(0)
            correct += (predicted == y_val).sum().item()
    acc = correct / total

    if (epoch + 1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}] | Loss: {running_loss/len(train_loader):.4f} | Val Acc: {acc:.4f}")

# ======================
```

```python
# 5️⃣ 최종 평가
# ======================
model.eval()
with torch.no_grad():
    outputs = model(X_test)
    predicted = torch.argmax(outputs, dim=1)
    accuracy = (predicted == y_test).sum().item() / len(y_test)

print(f"\n✅ Test Accuracy: {accuracy:.4f}")
```

```
IrisNet(
  (model): Sequential(
    (0): Linear(in_features=4, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=32, out_features=3, bias=True)
    (7): Softmax(dim=1)
  )
)
Epoch [10/100] | Loss: 0.8958 | Val Acc: 0.9000
Epoch [20/100] | Loss: 0.7516 | Val Acc: 0.9000
Epoch [30/100] | Loss: 0.7204 | Val Acc: 0.9333
Epoch [40/100] | Loss: 0.6977 | Val Acc: 0.9667
Epoch [50/100] | Loss: 0.6784 | Val Acc: 1.0000
Epoch [60/100] | Loss: 0.6462 | Val Acc: 1.0000
Epoch [70/100] | Loss: 0.6309 | Val Acc: 1.0000
Epoch [80/100] | Loss: 0.6261 | Val Acc: 1.0000
Epoch [90/100] | Loss: 0.6133 | Val Acc: 1.0000
Epoch [100/100] | Loss: 0.5942 | Val Acc: 1.0000

✅ Test Accuracy: 1.0000
```

In [56]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from torch.utils.data import TensorDataset, DataLoader
import copy
import numpy as np

# ====================================
# 1️⃣ 데이터 준비
```

```python
# ==================================
iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)

train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# ==================================
# 2️⃣ 모델 정의
# ==================================
class RegularizedNet(nn.Module):
    def __init__(self, input_dim, hidden1=64, hidden2=32, output_dim=3):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, hidden1)
        self.fc2 = nn.Linear(hidden1, hidden2)
        self.fc3 = nn.Linear(hidden2, output_dim)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.softmax(self.fc3(x))
        return x

model = RegularizedNet(input_dim=X_train.shape[1])
print(model)

# ==================================
# 3️⃣ 손실함수, 옵티마이저, layer별 정규화 계수
```

```python
# ====================================
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Layer별 λ 설정
l2_fc1 = 0.01   # Ridge
l1_fc2 = 0.01   # Lasso
l1_fc3 = 0.01   # ElasticNet L1
l2_fc3 = 0.01   # ElasticNet L2

# ====================================
# 4️⃣ EarlyStopping 클래스 정의
# ====================================
class EarlyStopping:
    def __init__(self, patience=10, verbose=True):
        self.patience = patience
        self.counter = 0
        self.best_loss = np.inf
        self.best_state = None
        self.verbose = verbose

    def step(self, val_loss, model):
        if val_loss < self.best_loss - 1e-6:  # 손실이 줄었을 때
            self.best_loss = val_loss
            self.best_state = copy.deepcopy(model.state_dict())
            self.counter = 0
        else:
            self.counter += 1
            if self.verbose:
                print(f"  🔶 EarlyStopping counter: {self.counter}/{self.patience}")
        return self.counter >= self.patience

    def restore_best_weights(self, model):
        if self.best_state is not None:
            model.load_state_dict(self.best_state)

early_stopping = EarlyStopping(patience=10)

# ====================================
# 5️⃣ 학습 루프
# ====================================
num_epochs = 100
for epoch in range(num_epochs):
    model.train()
    train_loss = 0.0
```

```
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        outputs = model(X_batch)
        base_loss = criterion(outputs, y_batch)

        # 🔷 Layer별 정규화항 추가
        reg_loss = (
            l2_fc1 * torch.sum(model.fc1.weight ** 2) +
            l1_fc2 * torch.sum(torch.abs(model.fc2.weight)) +
            l1_fc3 * torch.sum(torch.abs(model.fc3.weight)) +
            l2_fc3 * torch.sum(model.fc3.weight ** 2)
        )

        loss = base_loss + reg_loss
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

    # 🔷 검증 손실 및 정확도 계산
    model.eval()
    val_loss = 0.0
    correct, total = 0, 0
    with torch.no_grad():
        for X_val, y_val in test_loader:
            preds = model(X_val)
            base_val_loss = criterion(preds, y_val)
            reg_val_loss = (
                l2_fc1 * torch.sum(model.fc1.weight ** 2) +
                l1_fc2 * torch.sum(torch.abs(model.fc2.weight)) +
                l1_fc3 * torch.sum(torch.abs(model.fc3.weight)) +
                l2_fc3 * torch.sum(model.fc3.weight ** 2)
            )
            val_loss += (base_val_loss + reg_val_loss).item()
            predicted = torch.argmax(preds, dim=1)
            total += y_val.size(0)
            correct += (predicted == y_val).sum().item()
    val_acc = correct / total
    if (epoch+1)%10==0:
        print(f"Epoch [{epoch+1}/{num_epochs}] | Train Loss: {train_loss/len(train_loader):.4f} | Val Loss: {val_loss/len(test_

    # 🔷 Early Stopping 체크
    if early_stopping.step(val_loss, model):
        print(f"🔲 Early stopping triggered at epoch {epoch+1}")
        break

# 🔷 최적 가중치 복원
```

```python
early_stopping.restore_best_weights(model)

# ==================================
# 6️⃣ 최종 평가
# ==================================
model.eval()
with torch.no_grad():
    outputs = model(X_test)
    predicted = torch.argmax(outputs, dim=1)
    accuracy = (predicted == y_test).sum().item() / len(y_test)

print(f"\n✅ Test Accuracy (Best Weights): {accuracy:.4f}")
```

```
RegularizedNet(
  (fc1): Linear(in_features=4, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=3, bias=True)
  (relu): ReLU()
  (softmax): Softmax(dim=1)
)
Epoch [10/100] | Train Loss: 1.4805 | Val Loss: 1.4399 | Val Acc: 0.6333
Epoch [20/100] | Train Loss: 1.1756 | Val Loss: 1.1703 | Val Acc: 0.8333
Epoch [30/100] | Train Loss: 1.0444 | Val Loss: 1.0339 | Val Acc: 0.7000
Epoch [40/100] | Train Loss: 0.9841 | Val Loss: 0.9602 | Val Acc: 0.7000
Epoch [50/100] | Train Loss: 0.9688 | Val Loss: 0.9378 | Val Acc: 0.7000
Epoch [60/100] | Train Loss: 0.9442 | Val Loss: 0.9229 | Val Acc: 0.7000
Epoch [70/100] | Train Loss: 0.9378 | Val Loss: 0.9104 | Val Acc: 0.7667
Epoch [80/100] | Train Loss: 0.9183 | Val Loss: 0.8975 | Val Acc: 0.8333
Epoch [90/100] | Train Loss: 0.9007 | Val Loss: 0.8836 | Val Acc: 0.8667
Epoch [100/100] | Train Loss: 0.8820 | Val Loss: 0.8662 | Val Acc: 0.9333

✅ Test Accuracy (Best Weights): 0.9333
```

In [ ]:

```python
class FixedTorchNNClassifier:
    """이미 학습된 PyTorch 모델을 sklearn VotingClassifier에서 사용할 수 있게 감싼 클래스"""
    def __init__(self, trained_model):
        self.model = trained_model  # 이미 학습된 torch.nn.Module

    def fit(self, X, y=None):
        # 아무것도 하지 않음 (VotingClassifier 내부 fit 호출 시 무시)
        return self

    def predict(self, X):
        self.model.eval()
        X = torch.FloatTensor(X)
        with torch.no_grad():
            out = self.model(X)
```

```
        preds = torch.argmax(out, dim=1).numpy()
        return preds

    def predict_proba(self, X):
        self.model.eval()
        X = torch.FloatTensor(X)
        with torch.no_grad():
            out = self.model(X)
        return out.numpy()

import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.base import BaseEstimator, ClassifierMixin
import numpy as np

class TorchNNClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, input_dim, hidden1=64, hidden2=32, output_dim=3, lr=0.001, epochs=50, batch_size=16):
        self.input_dim = input_dim
        self.hidden1 = hidden1
        self.hidden2 = hidden2
        self.output_dim = output_dim
        self.lr = lr
        self.epochs = epochs
        self.batch_size = batch_size

        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden1),
            nn.ReLU(),
            nn.Linear(hidden1, hidden2),
            nn.ReLU(),
            nn.Linear(hidden2, output_dim),
            nn.Softmax(dim=1)
        )

    def fit(self, X, y):
        X = torch.FloatTensor(X)
        y = torch.LongTensor(y)
        dataset = torch.utils.data.TensorDataset(X, y)
        loader = torch.utils.data.DataLoader(dataset, batch_size=self.batch_size, shuffle=True)

        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(self.model.parameters(), lr=self.lr)

        for epoch in range(self.epochs):
            self.model.train()
```

```
            for xb, yb in loader:
                optimizer.zero_grad()
                out = self.model(xb)
                loss = criterion(out, yb)
                loss.backward()
                optimizer.step()
        return self

    def predict(self, X):
        self.model.eval()
        X = torch.FloatTensor(X)
        with torch.no_grad():
            out = self.model(X)
        preds = torch.argmax(out, dim=1).numpy()
        return preds

    def predict_proba(self, X):
        self.model.eval()
        X = torch.FloatTensor(X)
        with torch.no_grad():
            out = self.model(X)
        return out.numpy()

from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from lightgbm import LGBMClassifier

# 1️⃣ PyTorch 모델 학습 (이전 코드 그대로)
nn_model = RegularizedNet(input_dim=X_train.shape[1])
# 학습 코드 수행 …
# (train loop로 이미 학습 완료했다고 가정)

# 2️⃣ 고정 래퍼로 감싸기
nn_fixed = FixedTorchNNClassifier(trained_model=nn_model)

# 3️⃣ 다른 모델들과 앙상블
rf = RandomForestClassifier(random_state=42)
lgbm = LGBMClassifier(random_state=42)

hard_voting = VotingClassifier(
    estimators=[('rf', rf), ('nn', nn_fixed), ('lgbm', lgbm)],
    voting='hard'
)

# 4️⃣ 이제 fit() 호출해도 nn_fixed는 재학습하지 않음
hard_voting.fit(X_train, y_train)
y_pred = hard_voting.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print(f"Hard Voting Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

In [42]:

```
#2-2 하드 보팅 소프트 보팅 장단점 설명하고 3가지 모델에 적용
# 하드 보팅 : 다수결 투표 방식으로 각 모델의 예측 결과 중 가장 많이 나온 클래스를 최종 예측으로 선택
# 장점 : 구현이 간단하고 직관적, 개별 모델의 성능이 비슷할 때 효과적, 이상치 에 덜민감.
# 단점 : 개별 모델의 확신도를 반영하지 않음, 일부 모델이 다른 모델보다 훨씬 뛰어날 때 최적이 아닐 수 있음, 모델을 다 동급
# 소프트 보팅 : 각 모델의 예측 확률을 평균내어 가장 높은 확률을 가진 클래스를 최종 예측으로 선택
# 장점 : 개별 모델의 확신도를 반영, 개별 모델의 성능 차이가 클 때 더 나은 성능을 보일 수 있음, 더많은 정보가 있기 때문
# 단점 : 구현이 다소 복잡, 확률을 제공하지 않는 모델에는 적용 불가

from sklearn.ensemble import VotingClassifier

rf=RandomForestClassifier(random_state=42)
nn= MLPClassifier(random_state=42, max_iter=500)
lgbm= LGBMClassifier(random_state=42,verbose=-1)

hard_voting= VotingClassifier(estimators=[('rf',rf),('nn',nn),('lgbm',lgbm)], voting='hard')
hard_voting.fit(X_train,y_train)
y_pred_hard= hard_voting.predict(X_test)
f1_hard= f1_score(y_test,y_pred_hard, average='weighted')
print("Hard Voting F1 Score:", f1_hard)
print(classification_report(y_test,y_pred_hard))

soft_voting= VotingClassifier(estimators=[('rf',rf),('nn',nn),('lgbm',lgbm)], voting='soft')
soft_voting.fit(X_train,y_train)
y_pred_soft= soft_voting.predict(X_test)
f1_soft= f1_score(y_test,y_pred_soft, average='weighted')
print("Soft Voting F1 Score:", f1_soft)
print(classification_report(y_test,y_pred_soft))
```

```
Hard Voting F1 Score: 0.37480086749563457
              precision    recall  f1-score   support

           1       0.49      0.76      0.60        38
           2       0.31      0.33      0.32        12
           3       0.40      0.12      0.19        16
           4       0.00      0.00      0.00         5
           5       0.00      0.00      0.00         8

    accuracy                           0.44        79
   macro avg       0.24      0.24      0.22        79
weighted avg       0.36      0.44      0.37        79

Soft Voting F1 Score: 0.3917970490342532
              precision    recall  f1-score   support

           1       0.50      0.71      0.59        38
           2       0.28      0.42      0.33        12
           3       0.40      0.12      0.19        16
           4       0.00      0.00      0.00         5
           5       0.50      0.12      0.20         8

    accuracy                           0.44        79
   macro avg       0.34      0.28      0.26        79
weighted avg       0.41      0.44      0.39        79
```

In [57]:

```python
#통계5 두회사의 데이터를 분석하시오
#5-1 Kaplan Meier 방법을 사용하여 생존 분석을 수행하고 회사 부품별 25,35,45 생존확율을 구하세요
#5-2 두 회사간 생존시간 차이를 log-rank 검정을 통해 분석하고 결과를 해석하세요

df=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/28_problem4.csv")
print(df)
from lifelines import KaplanMeierFitter
kmf= KaplanMeierFitter()

company_A= df[df['company']=='A']
kmf.fit(durations=company_A['time(month)'], event_observed=company_A['survival']=='Y')
survival_prob_A_25=kmf.survival_function_.iloc[25].values[0]
survival_prob_A_35=kmf.survival_function_.iloc[35].values[0]
survival_prob_A_45=kmf.survival_function_.iloc[45].values[0]
print(f"Company A Survival Probabilities: 25 months: {survival_prob_A_25:.4f}, 35 months: {survival_prob_A_35:.4f}, 45 months:


company_A= df[df['company']=='B']
kmf.fit(durations=company_A['time(month)'], event_observed=company_A['survival']=='Y')
```

```
survival_prob_A_25=kmf.survival_function_.iloc[25].values[0]
survival_prob_A_35=kmf.survival_function_.iloc[35].values[0]
survival_prob_A_45=kmf.survival_function_.iloc[45].values[0]
print(f"Company B Survival Probabilities: 25 months: {survival_prob_A_25:.4f}, 35 months: {survival_prob_A_35:.4f}, 45 months:
```

```
      time(month) survival company
0              39        Y       B
1              29        Y       B
2              15        N       B
3              43        N       A
4               8        Y       A
...           ...      ...     ...
1995           34        N       B
1996           36        Y       B
1997            6        N       B
1998           39        Y       A
1999           42        N       B

[2000 rows x 3 columns]
Company A Survival Probabilities: 25 months: 0.7140, 35 months: 0.5155, 45 months: 0.2315
Company B Survival Probabilities: 25 months: 0.7164, 35 months: 0.5190, 45 months: 0.2677
```

In [44]:

```
from lifelines.statistics import logrank_test
company_A= df[df['company']=='A']["time(month)"]
company_B= df[df['company']=='B']["time(month)"]
event_observed_A= df[df['company']=='A']['survival'].apply(lambda x: 1 if x=='Y' else 0)
event_observed_B= df[df['company']=='B']['survival'].apply(lambda x: 1 if x=='Y' else 0)
resuts= logrank_test(company_A,company_B,event_observed_A, event_observed_B)
print(resuts.summary)
# print(resuts.summary)
# 귀무가설 두회사간의 생존시간에는 유의미한 차이가 없다.
# 대립가설 두회사간 생존시간에는 유의미한 차이가 있다.
```

```
   test_statistic        p  -log2(p)
0        0.604542  0.43685   1.19479
```

In [ ]:

```
#C# 5-3 Cox 비례위험모형을 사용하여 회사 부품별 생존시간에 영향을 미치는 요인을 분석하고 결과를 해석하세요
from lifelines import CoxPHFitter
df['event']= df['survival'].apply(lambda x: 1 if x=='Y' else 0)
df_encoded= pd.get_dummies(df, columns=['company'], drop_first=True)
cph= CoxPHFitter()
cph.fit(df_encoded[['company_B','time(month)','event']], duration_col='time(month)', event_col='event')
cph.print_summary()
# “company_B에 속한 사람은 기준 그룹에 비해 사건 발생 위험이 약 5% 낮지만,
# 이 차이는 통계적으로 유의하지 않으며(p=0.46),
# 95% 신뢰구간(0.84–1.08)이 1을 포함하므로
# ‘company_B가 생존시간(또는 위험률)에 유의한 영향을 미친다고 보기 어렵다.’”
```

```
# 또한 모델의 concordance가 0.50 수준으로
# 예측력이 거의 무작위 수준임을 보여줍니다.
```

model	lifelines.CoxPHFitter
duration col	'time(month)'
event col	'event'
baseline estimation	breslow
number of observations	2000
number of events observed	982
partial log-likelihood	-6495.66
time fit was run	2025-10-07 15:07:44 UTC

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	cmp to	z	p	-log2(p)
company_B	-0.05	0.95	0.06	-0.17	0.08	0.84	1.08	0.00	-0.75	0.46	1.13

Concordance	0.50
Partial AIC	12993.32
log-likelihood ratio test	0.56 on 1 df
-log2(p) of ll-ratio test	1.13

열 이름	의미	해석 포인트
coef	회귀계수 (log hazard ratio)	+이면 위험 증가, –이면 위험 감소
exp(coef)	Hazard Ratio (위험비, HR)	해석에 가장 많이 쓰이는 값
se(coef)	표준오차	계수 추정의 불확실성
z	z-통계량 = coef / se(coef)	통계적 유의성 판단
p	p-value	0.05 미만이면 유의한 변수
lower 0.95, upper 0.95	95% 신뢰구간 (HR 기준)	HR이 1을 포함하지 않으면 유의함

변수	coef	exp(coef)	해석
age	+0.03	**1.03**	나이가 1세 많을수록 사망 위험이 **3% 증가**
sex (남성=1)	-0.70	**0.50**	남성의 사망 위험이 여성의 **50% 수준** (위험 감소)
bp_high	+0.50	**1.65**	고혈압이면 위험이 **1.65배 높음**

exp(coef) = 1 → 영향 없음

exp(coef) > 1 → 위험 증가 (사건 빨리 발생)

exp(coef) < 1 → 위험 감소 (생존기간 길어짐)

In [47]:
```
#통계 2 시식여부가 구매의사에 영향을 주는지 분석하고 결과를 해석하세요
#귀무가설 시식여부가 구매의사에 영향을 미치지 않는다
#대립가설 시식여부가 구매의사에 영향을 미친다

df=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/28_problem5.csv")
import pandas as pd
import scipy.stats as stats
contingency_table= pd.crosstab(df['시식전'], df['시식후'])
chi2, p, dof, expected= stats.chi2_contingency(contingency_table)
print(f"Chi-square Statistic: {chi2}, p-value: {p}")
```

```
Chi-square Statistic: 0.11136462407648856, p-value: 0.7385958474438541
```

In [48]:
```
#통계3 2개의 고등학교 두학교 시험범수 분포 차이가 있는지 검정하시오

df=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/28_problem6.csv")
from scipy import stats
school1= df[df['School']=='A']['Score']
school2= df[df['School']=='B']['Score']
t_stat, p_value= stats.ttest_ind(school1, school2)
print(f"T-statistic: {t_stat}, p-value: {p_value}")
#귀무가설 두학교 시험범수 분포 차이가 없다
#대립가설 두학교 시험범수 분포 차이가 있다
```

```
T-statistic: 0.9792447503166659, p-value: 0.32865402565919544
```

In [49]:
```
#통계4 몸무게를 제어한다고 생각하고 나이와 콜레스테롤 상관 계수 및 유의 확율으 ㄹ구하시오
import pandas as pd
import scipy.stats as stats
import numpy as np
import statsmodels.api as sm
```

```python
df=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/28_problem7.csv")
# 나이와 콜레스테롤의 상관계수
corr_age_chol, p_value_age_chol= stats.pearsonr(df['age'], df['Cholesterol'])
print(f"Correlation between Age and Cholesterol: {corr_age_chol}, p-value: {p_value_age_chol}")
# 상관 없다.
# 나이와 몸무게를 독립변수, 콜레스테롤을 종속변수로 회귀모형 적합
X = df[['age', 'weight']]
X = sm.add_constant(X)  # 상수항 추가
y = df['Cholesterol']

model = sm.OLS(y, X).fit()
print(model.summary())
```

```
Correlation between Age and Cholesterol: 0.003696499024083223, p-value: 0.9070599415159853
                            OLS Regression Results
==============================================================================
Dep. Variable:            Cholesterol   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                 -0.002
Method:                 Least Squares   F-statistic:                    0.01703
Date:                Tue, 07 Oct 2025   Prob (F-statistic):              0.983
Time:                        22:15:15   Log-Likelihood:                -5457.3
No. Observations:                1000   AIC:                         1.092e+04
Df Residuals:                     997   BIC:                         1.094e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        201.2584      8.130     24.756      0.000     185.305      217.212
age            0.0124      0.103      0.121      0.904      -0.190        0.214
weight        -0.0109      0.076     -0.143      0.886      -0.161        0.139
==============================================================================
Omnibus:                      451.801   Durbin-Watson:                   2.103
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               53.370
Skew:                           0.006   Prob(JB):                     2.58e-12
Kurtosis:                       1.868   Cond. No.                         443.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [ ]: