SK쉴더스 생성형AI 활용 사이버보안 전문인력 양성과정 26기AI

*과목명: 생성형AI 활용을 위한 머신러닝 딥러닝

기초-1

모범 답안

```
# 필요한 라이브러리 임포트
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# 데이터 로드
df = pd.read_csv("diabetes.csv")

# 1. 결측치 처리 (값이 0인 경우를 결측치로 간주)
columns_to_check = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI']
for col in columns_to_check:
    df[col].replace(0, np.nan, inplace=True)
    df[col].fillna(df[col].mean(), inplace=True)

# 2. 이상치 처리 (상위 1% 값을 이상치로 간주)
for col in ['SkinThickness', 'Insulin']:
    upper_limit = df[col].quantile(0.99)
    df[col] = df[col].apply(lambda x: np.nan if x > upper_limit else x)
    df[col].fillna(df[col].mean(), inplace=True)

# 3. 데이터 정규화
scaler = MinMaxScaler()
df[df.columns] = scaler.fit_transform(df[df.columns])

# 4. 탐색적 데이터 분석
# 각 열의 결측치 개수 출력
missing_values = df.isnull().sum()
print("결측치 개수:\n", missing_values)

# Outcome 에 따른 Glucose 의 평균 값 계산
glucose_mean_by_outcome = df.groupby('Outcome')['Glucose'].mean()
print("\nOutcome 에 따른 Glucose 평균:\n", glucose_mean_by_outcome)

# 결과 확인
print("\n처리된 데이터셋 첫 5개 행:\n", df.head())
```

SK쉴더스 생성형AI 활용 사이버보안 전문인력 양성과정 26기AI

기본-1

모범 답안

```python
# 1. 필요한 라이브러리 불러오기
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# 2. 데이터 읽어오기
data = pd.read_csv('train.csv')

# 3. 데이터 전처리
# 결측값 확인 및 처리
missing_cols = data.isnull().sum()
high_missing_cols = missing_cols[missing_cols > 0.3 * len(data)].index
data.drop(high_missing_cols, axis=1, inplace=True)

# LotFrontage 결측값 평균 대체
data['LotFrontage'] = data['LotFrontage'].fillna(data['LotFrontage'].mean())

# 범주형 데이터 인코딩
data = pd.get_dummies(data, drop_first=True)

# 불필요한 열 제거
data.drop(['Id'], axis=1, inplace=True)

# 4. 데이터 분리
X = data.drop('SalePrice', axis=1)  # 특징 데이터
y = data['SalePrice']               # 타겟 데이터

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

# 5. 모델 학습
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)
```

```python
# 6. 모델 평가
y_pred = model.predict(X_test)

# 성능 평가 지표 계산
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# 결과 출력
print(f"평균 절대 오차 (MAE): {mae:.2f}")
print(f"평균 제곱 오차 (MSE): {mse:.2f}")
print(f"제곱근 평균 제곱 오차 (RMSE): {rmse:.2f}")
print(f"R² 점수: {r2:.2f}")
```

```
PS D:\ai_proj> & C:/Users/jin/AppData/Local/Programs/Python/Python312/python.exe d:/ai_proj/pbl_2.py
평균 절대 오차 (MAE): 26386.46
평균 제곱 오차 (MSE): 1750367239.41
제곱근 평균 제곱 오차 (RMSE): 41837.39
R² 점수: 0.77
```

모델이 평균적으로 약 26,386달러(MAE)의 오차를 내고 있으며, 큰 오차를 포함한 평균적인 오차는 약 41,837달러(RMSE)입니다.
모델은 주택 가격 변동의 약 77%를 설명(R²)할 수 있으며, 나머지 23%는 설명하지 못한 부분입니다.
모델의 성능은 비교적 양호하지만, MAE와 RMSE가 비교적 크다는 점에서 더 개선이 가능할 여지가 있습니다.

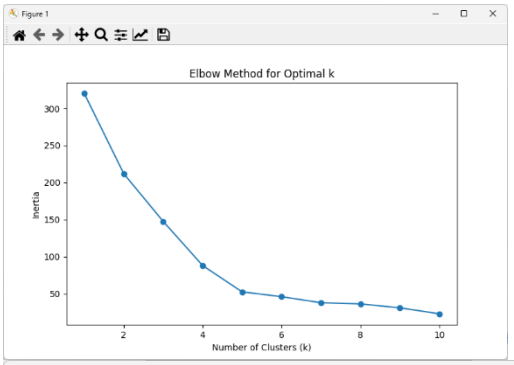
기본-2

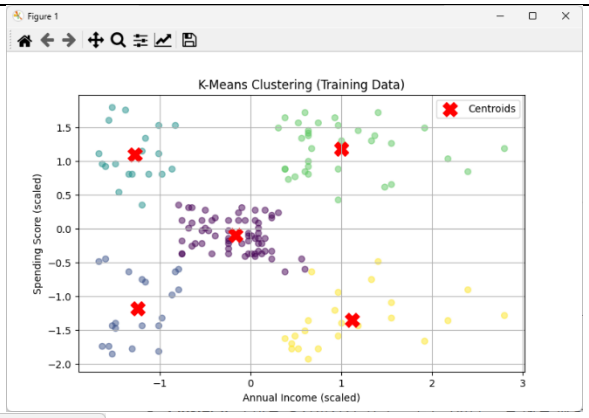
모범 답안

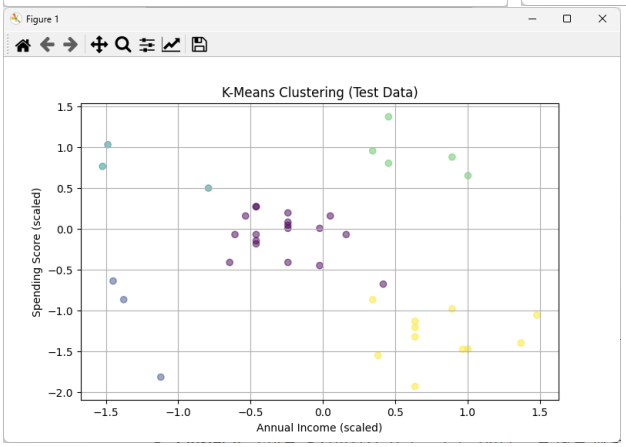
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.model_selection import train_test_split


# 1. 데이터 불러오기
# Mall_Customers.csv 파일에서 데이터를 불러옵니다.
data = pd.read_csv("Mall_Customers.csv")
print(data.head())

# 필요한 열 선택
# 고객의 연소득(Annual Income)과 소비 점수(Spending Score)만 선택하여
분석합니다.
X = data[['Annual Income (k$)', 'Spending Score (1-100)']]

# 데이터 분리 (80% 학습, 20% 테스트)
# 데이터셋을 학습용 80%, 테스트용 20%로 나눕니다.
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

# 데이터 표준화
# K-Means 알고리즘의 성능을 높이기 위해 데이터를 표준화합니다.
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # 학습 데이터 표준화
X_test_scaled = scaler.transform(X_test)       # 테스트 데이터 표준화


# 2. 최적의 클러스터 수 찾기 (엘보우 방법)
# 클러스터 수(k)를 1부터 10까지 변경하며 각 클러스터의 관성값(inertia)을
계산합니다.
inertia = []
k_range = range(1, 11)

for k in k_range:
```

```
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)  # 학습 데이터로 모델 학습
    inertia.append(kmeans.inertia_)  # 각 k 에 대한 관성값 저장

# 엘보우 그래프
# 관성값을 시각화하여 최적의 k 값을 결정합니다.
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.xlabel('Number of Clusters (k)')  # 클러스터 수 (k)
plt.ylabel('Inertia')                 # 관성값 (Inertia)
plt.title('Elbow Method for Optimal k')  # 제목
plt.show()

# 최적의 클러스터 수 선택
# 엘보우 그래프에서 최적의 클러스터 수를 5로 선택합니다.
optimal_k = 5
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train_scaled)  # 학습 데이터로 최적의 모델 학습


# 3. 모델 테스트 및 평가
# 테스트 데이터 클러스터 할당
# 학습된 K-Means 모델을 사용해 테스트 데이터에 클러스터를 할당합니다.
test_clusters = kmeans.predict(X_test_scaled)

# Silhouette Score 계산
# 테스트 데이터에 대해 Silhouette Score 를 계산하여 클러스터링 성능을 평가합니다.
silhouette_avg = silhouette_score(X_test_scaled, test_clusters)
print(f"Silhouette Score on Test Data: {silhouette_avg:.2f}")  # Silhouette
점수 출력


# 4. 결과 시각화
# 학습 데이터 시각화
# 학습 데이터를 2차원 공간에 클러스터별로 시각화하고 중심점(Centroid)을
표시합니다.
plt.figure(figsize=(8, 5))
plt.scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c=kmeans.labels_,
cmap='viridis', alpha=0.5)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='red', marker='X', label='Centroids')  # 클러스터 중심점
표시
plt.title('K-Means Clustering (Training Data)')  # 제목
plt.xlabel('Annual Income (scaled)')  # X 축: 연소득 (표준화)
plt.ylabel('Spending Score (scaled)')  # Y 축: 소비 점수 (표준화)
```

```
plt.legend()  # 범례
plt.grid(True)  # 그리드 추가
plt.show()

# 테스트 데이터 시각화
# 테스트 데이터를 2차원 공간에 클러스터별로 시각화합니다.
plt.figure(figsize=(8, 5))
plt.scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], c=test_clusters,
cmap='viridis', alpha=0.5)
plt.title('K-Means Clustering (Test Data)')  # 제목
plt.xlabel('Annual Income (scaled)')  # X 축: 연소득 (표준화)
plt.ylabel('Spending Score (scaled)')  # Y 축: 소비 점수 (표준화)
plt.grid(True)  # 그리드 추가
plt.show()


# 클러스터 결과 출력
# 테스트 데이터의 각 고객에 대해 클러스터 할당 결과를 확인합니다.
print("Test Data Cluster Assignments:")
print(pd.DataFrame({'Annual Income': X_test['Annual Income (k$)'].values,
                    'Spending Score': X_test['Spending Score (1-
100)'].values,
                    'Cluster': test_clusters}))
```







결과 요약

Cluster 0: 중간 소득과 중간 소비 성향의 고객.

Cluster 1: 소득과 소비가 모두 낮은 고객.

Cluster 2: 소득은 낮지만 소비 성향이 높은 고객.

Cluster 3: 소득과 소비가 모두 높은 프리미엄 고객.

Cluster 4: 소득은 높지만 소비 성향이 낮은 고객.

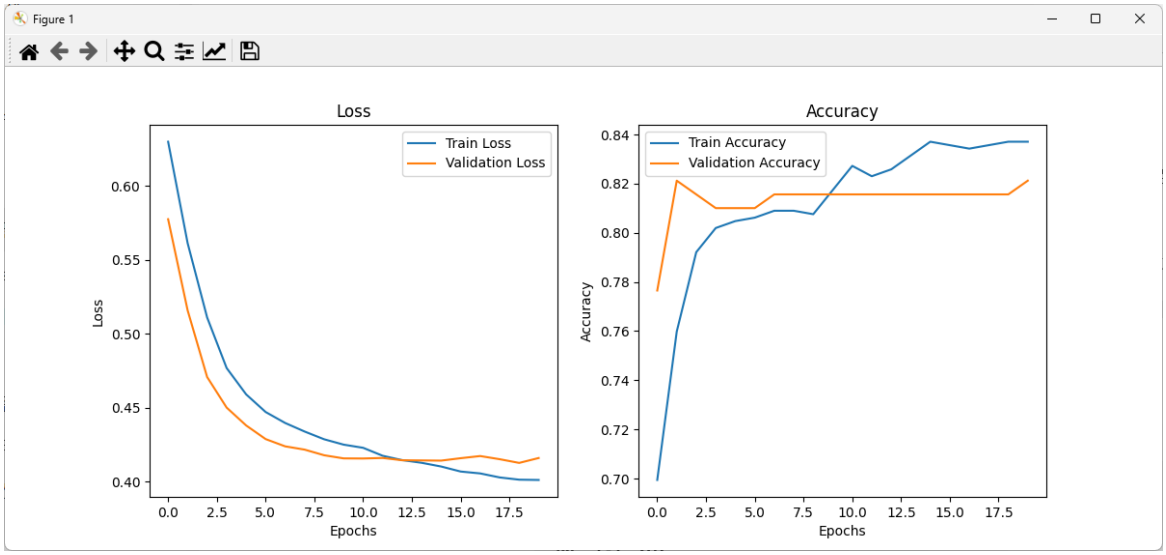
응용-1

모범 답안

```
# 필요한 라이브러리 임포트
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# 1. 데이터 로드 및 탐색
data = pd.read_csv('titanic.csv')

# 데이터 구조 확인
print("데이터 헤드:")
print(data.head())
print("\n데이터 정보:")
print(data.info())

# 2. 데이터 전처리
# 필요한 특성 선택
selected_features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
target = 'Survived'

# 결측값 처리
# Age는 평균값으로 채움
if data['Age'].isnull().sum() > 0:
    data['Age'].fillna(data['Age'].mean(), inplace=True)
# Fare는 중앙값으로 채움 (결측값이 있을 경우)
if data['Fare'].isnull().sum() > 0:
    data['Fare'].fillna(data['Fare'].median(), inplace=True)

# 범주형 데이터 처리 (Sex 컬럼 One-Hot Encoding)
if 'Sex' in data.columns:
    data = pd.get_dummies(data, columns=['Sex'], drop_first=True)
    # 각 고유값을 기준으로 새로운 컬럼을 만듭니다. 예: Sex_male, Sex_female
    # drop_first=True로 첫 번째 고유값을 삭제하여 다중공선성 문제를 방지합니다.
```

```
# 선택한 특성 + One-Hot Encoding 으로 생성된 Sex_male 컬럼 포함
if 'Sex_male' in data.columns:
    selected_features += ['Sex_male']
else:
    raise KeyError("The column 'Sex_male' was not created properly during
One-Hot Encoding.")

# 특성과 타겟 분리
X = data[selected_features]
y = data[target]

# 데이터 정규화
scaler = StandardScaler()
X = scaler.fit_transform(X)

# 3. 데이터 분할
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

# 4. 딥러닝 모델 구축
model = Sequential([
    Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')  # 이진 분류이므로 sigmoid 사용
])

# 모델 컴파일
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# 모델 학습
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
epochs=20, batch_size=32)

# 5. 결과 시각화
# 학습 손실(loss) 및 정확도(accuracy) 시각화
plt.figure(figsize=(12, 5))

# Loss 그래프
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.legend()

# Accuracy 그래프
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()

# 모델 평가
loss, accuracy = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
```



응용-2

모범 답안

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# 1. 데이터 로드
# CSV 파일로부터 로그 데이터를 로드하여 DataFrame 으로 저장합니다.
data = pd.read_csv("./web_server_logs_2.csv")

# 2. 속성 생성
# 요청 시간대(hour)를 추출하여 새로운 열로 추가합니다.
data['hour'] = pd.to_datetime(data['timestamp']).dt.hour

# HTTP 상태 코드가 400 이상인 경우 에러로 간주하고 플래그를 설정합니다.
data['is_error'] = (data['status_code'] >= 400).astype(int)

# 악성 요청의 레이블(label)을 생성합니다.
# 에러 플래그(`is_error`)를 기반으로 레이블을 설정합니다 (1: 악성, 0: 정상).
data['label'] = data['is_error']

# 3. 학습 데이터 준비
# 학습에 사용할 주요 속성(features)을 선택합니다.
# - hour: 요청이 이루어진 시간
# - size: 응답 크기
# - is_error: 에러 여부 플래그
features = data[['hour', 'size', 'is_error']]

# 라벨(label)은 악성 여부를 나타내는 `label` 열입니다.
labels = data['label']

# 4. 데이터 분할
# 데이터를 학습 데이터와 테스트 데이터로 분리합니다 (70% 학습, 30% 테스트).
X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.3, random_state=42)

# 5. 모델 학습
# Logistic Regression 모델을 학습 데이터에 대해 학습시킵니다.
```

```
model = LogisticRegression()
model.fit(X_train, y_train)

# 6. 모델 평가
# 테스트 데이터를 사용하여 모델의 성능을 평가합니다.
y_pred = model.predict(X_test)

# 정확도(accuracy)를 계산하고 출력합니다.
accuracy = accuracy_score(y_test, y_pred)
print(f"모델 정확도: {accuracy:.2f}")

# 상세한 분류 성능 평가 지표(precision, recall, f1-score 등)를 출력합니다.
print(classification_report(y_test, y_pred))
```

응용-3

모범 답안

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import class_weight
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from sklearn.metrics import accuracy_score, f1_score, classification_report,
confusion_matrix

# 1. 데이터 로드
data = pd.read_csv('customer_data_balanced.csv')  # 고객 데이터를 로드

# 2. 데이터 섞기
data = data.sample(frac=1, random_state=42).reset_index(drop=True)  # 데이터를
무작위로 섞음

# 3. 범주형 데이터 변환 (One-hot Encoding)
data = pd.get_dummies(data, columns=['ContractType'], drop_first=True)  # 계약
유형을 One-hot 인코딩으로 변환

# 4. 주요 특징과 레이블 분리
X = data.drop('IsChurn', axis=1)  # 특징 데이터 (독립 변수)
y = data['IsChurn'].values  # 레이블 데이터 (종속 변수, numpy array로 변환)

# 5. 데이터 정규화
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # 데이터 스케일 조정 (평균 0, 분산 1로 정규화)

# 6. 데이터셋 분리
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
# 학습 데이터(80%)와 테스트 데이터(20%)로 분리

# 7. 클래스 가중치 설정
class_weights = {0: 1.0, 1: 2.0}  # 유지 고객(0)과 이탈 고객(1)에 대해 가중치 설정
```

```
# 8. 모델 설계: 딥러닝 모델 생성
model = Sequential([
    Input(shape=(X_train.shape[1],)),  # 입력 레이어, 특징 개수와 동일한 입력 크기
    Dense(128, activation='relu'),  # 첫 번째 Dense 레이어 (128개 노드, ReLU 활성화
함수)
    Dropout(0.3),  # 과적합 방지를 위한 Dropout
    Dense(64, activation='relu'),  # 두 번째 Dense 레이어 (64개 노드, ReLU 활성화
함수)
    Dropout(0.3),  # 과적합 방지를 위한 Dropout
    Dense(32, activation='relu'),  # 세 번째 Dense 레이어 (32개 노드, ReLU 활성화
함수)
    Dropout(0.2),  # 과적합 방지를 위한 Dropout
    Dense(1, activation='sigmoid')  # 출력 레이어 (이진 분류를 위한 Sigmoid 활성화
함수)
])

# 9. 모델 컴파일
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
# 옵티마이저: Adam, 손실 함수: Binary Crossentropy, 평가지표: Accuracy

# 10. 모델 학습
history = model.fit(
    X_train, y_train,
    epochs=30,  # 학습 횟수(Epoch)
    batch_size=32,  # 배치 크기
    validation_split=0.2,  # 학습 데이터의 20%를 검증 데이터로 사용
    class_weight=class_weights  # 클래스 가중치 적용
)

# 11. 테스트 데이터 평가
y_pred = (model.predict(X_test) > 0.5).astype(int)  # 예측 결과를 0 또는 1로 변환
accuracy = accuracy_score(y_test, y_pred)  # 정확도 계산
f1 = f1_score(y_test, y_pred)  # F1-Score 계산

print("Accuracy:", accuracy)
print("F1-Score:", f1)

# 12. 성능 분석 보고서 출력
print("\nClassification Report:")
print(classification_report(y_test, y_pred))  # 정밀도, 재현율, F1-Score 출력

# 13. 혼동 행렬 출력
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
```

```
print(conf_matrix)  # 혼동 행렬 출력
```

중요 내용 설명

1. 클래스 가중치 설정

class_weights = {0: 1.0, 1: 1.5}

유지 고객(0)보다 이탈 고객(1)에 더 높은 가중치를 부여.

이탈 고객 데이터가 적기 때문에, 모델이 이탈 고객을 더 중요하게 학습하도록 유도.

2. 모델 설계

Dense(128, activation='relu'),

Dropout(0.3)

Dense Layer: 뉴런 개수를 점진적으로 줄여가며 특징을 학습.

Dropout: 과적합 방지를 위해 학습 중 일부 뉴런을 비활성화.

3. 출력 레이어

Dense(1, activation='sigmoid')

이진 분류 문제이므로 Sigmoid 활성화 함수를 사용해 출력값을 0과 1 사이로 제한.

4. 손실 함수

loss='binary_crossentropy'

이진 분류 문제에 적합한 손실 함수로, 모델이 예측한 확률과 실제 라벨 간의 차이를 계산.

5. 성능 평가

y_pred = (model.predict(X_test) > 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred)

f1 = f1_score(y_test, y_pred)

예측값이 0.5보다 크면 1(이탈), 작으면 0(유지)로 변환.

정확도(Accuracy)와 F1-Score 로 모델 성능 평가.

6. 혼동 행렬

conf_matrix = confusion_matrix(y_test, y_pred)

혼동 행렬을 통해 각 클래스(유지, 이탈)의 True Positive, False Positive 등을 분석.