1.머신러닝 모델 하이퍼파라미터 튜닝

(1) 회귀모델튜닝

◯ 선형회귀 하이퍼파라미터

In [4]:
```
import pandas as pd
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# 데이터 로드 및 전처리
df = pd.read_csv('datasets/Clean_Dataset.csv')
df = df.drop(['flight', 'departure_time', 'stops', 'arrival_time'], axis=1) # 학
df = pd.get_dummies(df, columns=['airline', 'source_city', 'destination_city', '
X = df.drop('price', axis=1)  # 독립 변수
y = df['price']  # 종속 변수

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Ridge Regression (L2 정규화)
ridge = Ridge(alpha=1.0)  # L2 정규화 강도
ridge.fit(X_train, y_train)
ridge_preds = ridge.predict(X_test)
ridge_mse = mean_squared_error(y_test, ridge_preds)

# Lasso Regression (L1 정규화)
lasso = Lasso(alpha=0.1)  # L1 정규화 강도
lasso.fit(X_train, y_train)
lasso_preds = lasso.predict(X_test)
lasso_mse = mean_squared_error(y_test, lasso_preds)

print("Ridge Regression MSE:", ridge_mse)
print("Lasso Regression MSE:", lasso_mse)
```

```
Ridge Regression MSE: 50508878.30765215
Lasso Regression MSE: 50508855.78114255
```

◯ 랜덤 포레스트 하이퍼파라미터

In [6]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# 데이터 로드 및 전처리
df = pd.read_csv('datasets/Clean_Dataset.csv')
df = df.drop(['flight', 'departure_time', 'stops', 'arrival_time'], axis=1) # 학
df = pd.get_dummies(df, columns=['airline', 'source_city', 'destination_city', '
X = df.drop('price', axis=1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# 랜덤 포레스트 모델
```

```
rf = RandomForestRegressor(
    n_estimators=100,      # 트리 개수 (100개의 트리)
    max_depth=10,          # 각 트리의 최대 깊이
    min_samples_split=5,   # 노드를 분할하기 위한 최소 샘플 수
    min_samples_leaf=2,    # 리프 노드에 있어야 하는 최소 샘플 수
    random_state=42        # 결과 재현성을 위한 설정
)
rf.fit(X_train, y_train)  # 모델 학습
rf_preds = rf.predict(X_test)  # 테스트 데이터 예측
rf_mse = mean_squared_error(y_test, rf_preds)  # MSE 계산

print("Random Forest MSE:", rf_mse)
```

```
Random Forest MSE: 20024154.727510292
```

○ 그래디언 부스트 하이퍼파라미터

In [8]:
```
from sklearn.ensemble import GradientBoostingRegressor

# 데이터 로드 및 전처리
df = pd.read_csv('datasets/Clean_Dataset.csv')
df = df.drop(['flight', 'departure_time', 'stops', 'arrival_time'], axis=1) # 학
df = pd.get_dummies(df, columns=['airline', 'source_city', 'destination_city', '
X = df.drop('price', axis=1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# 그래디언트 부스트 모델
gb = GradientBoostingRegressor(
    learning_rate=0.1,     # 학습률
    n_estimators=100,      # 트리 개수
    max_depth=5,           # 각 트리의 최대 깊이
    random_state=42        # 결과 재현성을 위한 설정
)
gb.fit(X_train, y_train)  # 모델 학습
gb_preds = gb.predict(X_test)  # 테스트 데이터 예측
gb_mse = mean_squared_error(y_test, gb_preds)  # MSE 계산

print("Gradient Boosting MSE:", gb_mse)
```

```
Gradient Boosting MSE: 20996731.384828538
```

(2) 분류모델튜닝

○ 의사결정나무모델 하이퍼파라미터

In [11]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# 데이터 로드 및 전처리
df = pd.read_csv('datasets/heart.csv')
X = df.drop('output', axis=1)  # 독립 변수
y = df['output']               # 종속 변수 (심장병 여부: 1=있음, 0=없음)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

```
# 의사결정 나무 모델
dt = DecisionTreeClassifier(
    max_depth=5,          # 트리의 최대 깊이
    min_samples_split=10, # 노드를 분할하기 위한 최소 샘플 수
    min_samples_leaf=5,   # 리프 노드에 있어야 하는 최소 샘플 수
    random_state=42       # 결과 재현성을 위한 설정
)
dt.fit(X_train, y_train)  # 모델 학습

# 모델 평가
dt_preds = dt.predict(X_test)  # 테스트 데이터 예측
accuracy = accuracy_score(y_test, dt_preds)  # 정확도 계산
print("Decision Tree Classifier Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, dt_preds))
```

```
Decision Tree Classifier Accuracy: 0.8524590163934426

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.97      0.86        29
           1       0.96      0.75      0.84        32

    accuracy                           0.85        61
   macro avg       0.87      0.86      0.85        61
weighted avg       0.87      0.85      0.85        61
```

◯ 로지스틱 회귀 모델 하이퍼파라미터

In [25]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 데이터 로드 및 전처리
df = pd.read_csv('datasets/heart.csv')
X = df.drop('output', axis=1)
y = df['output']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# 로지스틱 회귀 모델
log_reg = LogisticRegression(
    C=1.0,           # 정규화 강도
    penalty='l2',    # L2 정규화
    solver='liblinear'  # 소규모 데이터셋에 적합한 솔버
)
log_reg.fit(X_train, y_train)  # 모델 학습

# 모델 평가
log_preds = log_reg.predict(X_test)  # 테스트 데이터 예측
accuracy = accuracy_score(y_test, log_preds)  # 정확도 계산
print("Logistic Regression Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, log_preds))
```

```
Logistic Regression Accuracy: 0.8688524590163934

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.86      0.86        29
           1       0.88      0.88      0.88        32

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

○ 랜덤 포레스트 하이퍼파라미터

In [17]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# 데이터 로드 및 전처리
df = pd.read_csv('datasets/heart.csv')
X = df.drop('output', axis=1)
y = df['output']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# 랜덤 포레스트 분류 모델
rf = RandomForestClassifier(
    n_estimators=100,      # 트리 개수
    max_depth=10,          # 트리의 최대 깊이
    min_samples_split=5,   # 노드를 분할하기 위한 최소 샘플 수
    min_samples_leaf=3,    # 리프 노드에 있어야 하는 최소 샘플 수
    random_state=42        # 결과 재현성을 위한 설정
)
rf.fit(X_train, y_train)  # 모델 학습

# 모델 평가
rf_preds = rf.predict(X_test)  # 테스트 데이터 예측
accuracy = accuracy_score(y_test, rf_preds)  # 정확도 계산
print("Random Forest Classifier Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, rf_preds))
```

```
Random Forest Classifier Accuracy: 0.8360655737704918

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        29
           1       0.84      0.84      0.84        32

    accuracy                           0.84        61
   macro avg       0.84      0.84      0.84        61
weighted avg       0.84      0.84      0.84        61
```

○ 그래디언트 부스트 하이퍼파라미터

In [20]:
```python
from sklearn.ensemble import GradientBoostingClassifier

# 데이터 로드 및 전처리
```

```
df = pd.read_csv('datasets/heart.csv')
X = df.drop('output', axis=1)
y = df['output']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# 그래디언트 부스트 분류 모델
gb = GradientBoostingClassifier(
    learning_rate=0.1,     # 학습률
    n_estimators=100,      # 부스팅 반복 횟수 (트리 개수)
    max_depth=5,           # 각 트리의 최대 깊이
    min_samples_split=10,  # 노드를 분할하기 위한 최소 샘플 수
    min_samples_leaf=5,    # 리프 노드에 있어야 하는 최소 샘플 수
    random_state=42        # 결과 재현성을 위한 설정
)
gb.fit(X_train, y_train)  # 모델 학습

# 모델 평가
gb_preds = gb.predict(X_test)  # 테스트 데이터 예측
accuracy = accuracy_score(y_test, gb_preds)  # 정확도 계산
print("Gradient Boosting Classifier Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, rf_preds))
```

```
Gradient Boosting Classifier Accuracy: 0.819672131147541

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        29
           1       0.84      0.84      0.84        32

    accuracy                           0.84        61
   macro avg       0.84      0.84      0.84        61
weighted avg       0.84      0.84      0.84        61
```

○ SVM 하이퍼파라미터

In [23]:

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# 데이터 로드 및 전처리
df = pd.read_csv('datasets/heart.csv')  # heart.csv 파일 로드
X = df.drop('output', axis=1)  # 독립 변수 (특징 데이터)
y = df['output']               # 종속 변수 (1: 심장병 있음, 0: 없음)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# SVM 분류 모델 설정
svm = SVC(
    kernel='rbf',    # RBF 커널 (비선형 데이터 처리)
    C=1.0,           # 정규화 강도
    gamma='scale'    # 감마 설정 (특성 스케일에 따라 자동 설정)
)
svm.fit(X_train, y_train)  # 모델 학습

# 모델 평가
svm_preds = svm.predict(X_test)  # 테스트 데이터 예측
accuracy = accuracy_score(y_test, svm_preds)  # 정확도 계산
```

```
print("SVM Classifier Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, svm_preds))
```

```
SVM Classifier Accuracy: 0.7049180327868853

Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.52      0.62        29
           1       0.67      0.88      0.76        32

    accuracy                           0.70        61
   macro avg       0.73      0.70      0.69        61
weighted avg       0.73      0.70      0.69        61
```