AI기반 챗봇 및 OCR 개발 전문가 과정

교과목명 : 머신러닝

- 평가일 : 24. 8. 2
- 성명 : 최환욱
- 점수 :

Q1. load_breast_cancer 데이터셋을 불러와서 다음을 수행하세요.

- dt로 분류모델 생성 및 모델 정확도 평가(학습:검증 = 8:2)
- 하이퍼 파라미터는 분할 기준은 지니계수, 최대 깊이는 3으로 설정
- 결정트리를 시각화

```
from sklearn.datasets import load_breast_cancer
import pandas as pd

cancer = load_breast_cancer()
# print("cancer.keys():\n", cancer.keys())
load_breast_cancer().keys()
cancer_data=cancer.data
cancer_label=cancer.target
print(cancer.target_names)
cancer_df=pd.DataFrame(data=cancer_data,columns=cancer.feature_names)
cancer_df['label']=cancer.target
cancer_df.head()
```

```
['malignant' 'benign']
```

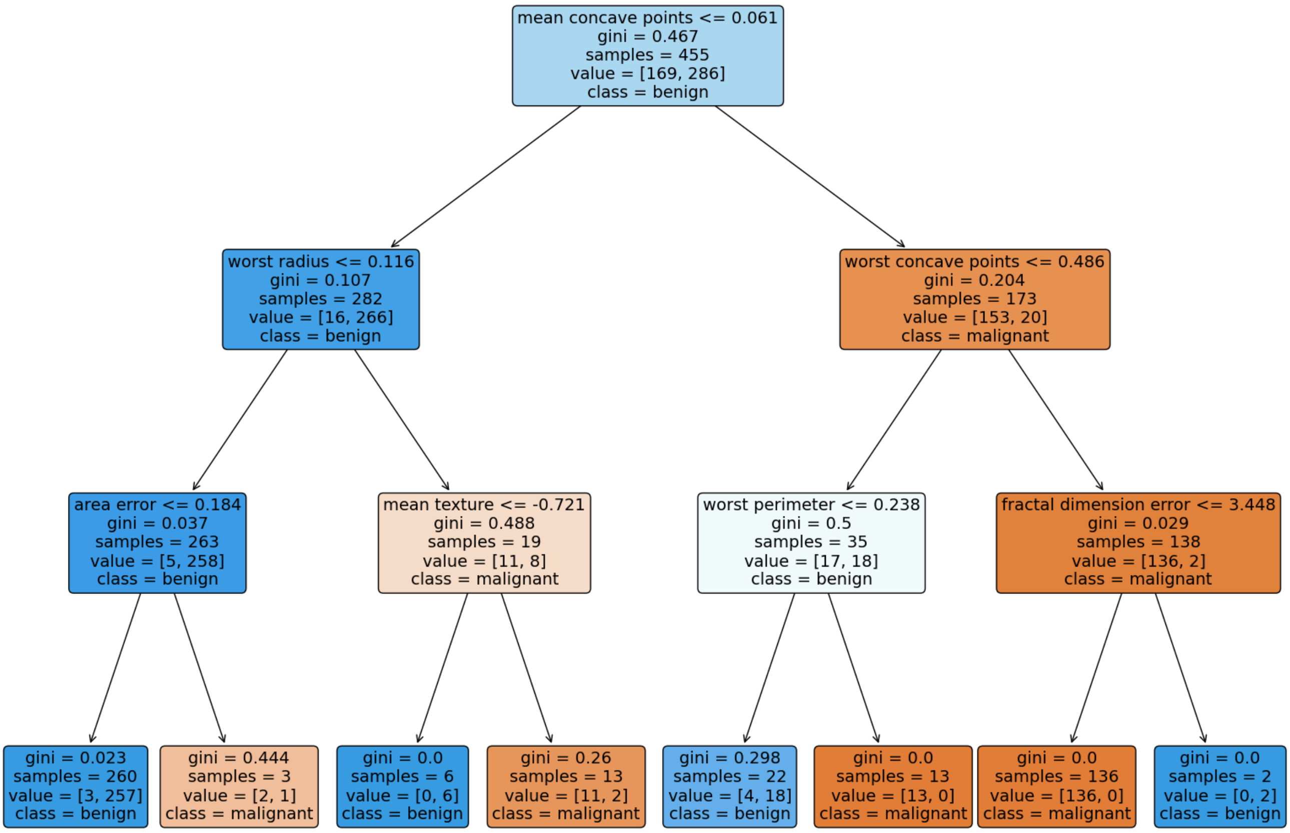
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20

5 rows × 31 columns

```
# 모델 학습 및 평가
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler

# 데이터 표준화
scaler = StandardScaler()
X_scaled = scaler.fit_transform(cancer_data)

# 학습용/테스트용 데이터 나누기
X_train,X_test,y_train,y_test=train_test_split(X_scaled,cancer_label,test_size=0.2,random_state=42)

## DTC 객체 생성
dt_clf=DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)

## 학습 수행
dt_clf.fit(X_train,y_train)

## 예측 수행: 학습이 완료된 DTC 객체에서 테스트 데이터 세트로 예측 수행
pred=dt_clf.predict(X_test)

from sklearn.metrics import accuracy_score
print('예측 정확도:{0:.4f}'.format(accuracy_score(y_test,pred)))
```

```
예측 정확도:0.9474
```

```
# 시각화
from sklearn import tree
import matplotlib.pyplot as plt
plt.figure(figsize=(20,15))

tree.plot_tree(dt_clf, filled=True, feature_names=cancer.feature_names, class_names=cancer.target_names, rounded=True, fontsize=14)
plt.show()

for i, node in enumerate(dt_clf.tree_.__getstate__()['nodes']):
    print(f"Node {i}: gini = {node['impurity']}, samples = {node['n_node_samples']}, value = {dt_clf.tree_.value[i]}")
```




```
Node 0: gini = 0.46693877551020413, samples = 455, value = [[169. 286.]]
Node 1: gini = 0.10703686937276802, samples = 282, value = [[ 16. 266.]]
Node 2: gini = 0.03729994650782864, samples = 263, value = [[  5. 258.]]
Node 3: gini = 0.022810650887573947, samples = 260, value = [[  3. 257.]]
Node 4: gini = 0.4444444444444444, samples = 3, value = [[2. 1.]]
Node 5: gini = 0.48753462603878117, samples = 19, value = [[11.  8.]]
Node 6: gini = 0.0, samples = 6, value = [[0. 6.]]
Node 7: gini = 0.2603550295857988, samples = 13, value = [[11.  2.]]
Node 8: gini = 0.2044839453372983, samples = 173, value = [[153.  20.]]
Node 9: gini = 0.4995918367346939, samples = 35, value = [[17. 18.]]
Node 10: gini = 0.2975206611570248, samples = 22, value = [[ 4. 18.]]
Node 11: gini = 0.0, samples = 13, value = [[13.  0.]]
Node 12: gini = 0.02856542743121193, samples = 138, value = [[136.   2.]]
Node 13: gini = 0.0, samples = 136, value = [[136.   0.]]
Node 14: gini = 0.0, samples = 2, value = [[0. 2.]]
```

코딩을 시작하거나 AI로 코드를 생성하세요.

Q2. 와인 데이터에 대해서 아래 사항을 고려하여 모델 생성 및 성능개선을 위한 하이퍼파라미터 튜닝을 수행한 후 테스트 데이터로 평가하세요.

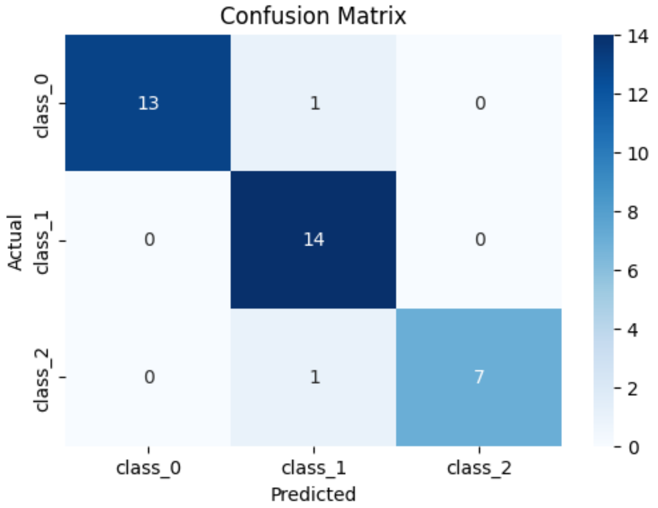
- dt를 알고리즘으로 적용
- cv = 5
- param_grid = {'max_depth': [3, 4, 5, 6], 'min_samples_split': [2, 3, 4]}

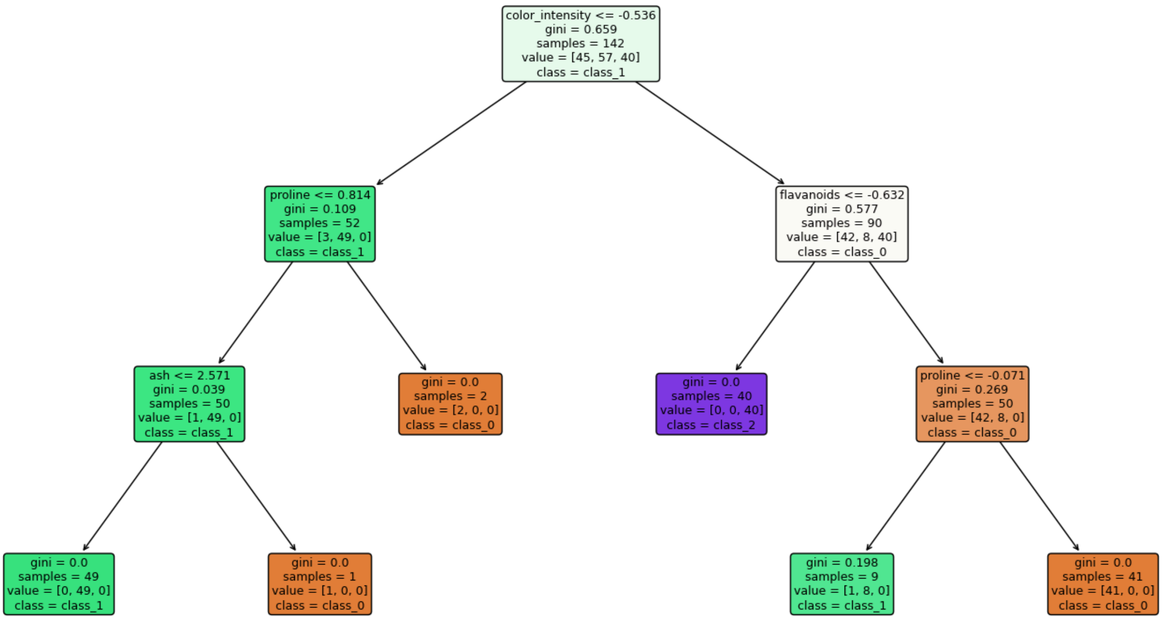
```
from sklearn import datasets
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# 와인 데이터 불러오기
wine = datasets.load_wine()
#print(wine.DESCR)
print(wine.feature_names)
print(wine.target_names)
X = wine.data
y = wine.target
```

```
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'co
['class_0' 'class_1' 'class_2']
```

```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# 데이터 표준화
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 학습용과 테스트용 데이터셋으로 나누기
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# 하이퍼파라미터 튜닝
param_grid = {
    "max_depth": [3, 4, 5, 6],
    "min_samples_split": [2, 3, 4]
}

# GridSearchCV를 사용하여 최적의 하이퍼파라미터 찾기
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),param_grid=param_grid,scoring='accuracy',cv=5, n_jobs=-1)

grid_search.fit(X_train, y_train)

# 최적의 하이퍼파라미터 출력
print(f"Best parameters: {grid_search.best_params_}")

# 최적의 하이퍼파라미터로 학습된 모델로 예측 수행
best_dt_clf = grid_search.best_estimator_
y_pred = best_dt_clf.predict(X_test)
y_pred_proba = best_dt_clf.predict_proba(X_test)[:, 1]

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, roc_curve, auc
# 성능 지표 출력
print('예측 정확도: {0:.4f}'.format(accuracy_score(y_test, y_pred)))
print('정밀도: {0:.4f}'.format(precision_score(y_test, y_pred, average='weighted')))
print('재현율: {0:.4f}'.format(recall_score(y_test, y_pred, average='weighted')))
print('F1 스코어: {0:.4f}'.format(f1_score(y_test, y_pred, average='weighted')))

# 혼동 행렬 출력
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=wine.target_names, yticklabels=wine.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# 트리 시각화
plt.figure(figsize=(18, 10))
tree.plot_tree(best_dt_clf, filled=True, feature_names=wine.feature_names, class_names=wine.target_names, rounded=True, fontsize=9)
plt.show()
```

```
Best parameters: {'max_depth': 3, 'min_samples_split': 2}
예측 정확도: 0.9444
정밀도: 0.9514
재현율: 0.9444
F1 스코어: 0.9449
```





Q3. 보스톤 주택가격 데이터셋에 대하여 규제 선형 모델인 릿지, 라쏘, 엘라스틱넷 모델로 교차검증을 수행하고 아래 각 모델의 알파값의 변화에 따른 회귀계수의 변화를 출력하세요. (단, 사용자 함수를 작성하여 수행)

- ridge_alphas = [0, 0.1, 1, 10, 100]
- lasso_alphas = [0.07,0.1,0.5,1,3]
- elastic_alphas = [0.07,0.1,0.5,1,3], L1:L2 = 0.7:0.3

```
import pandas as pd
from sklearn.datasets import fetch_openml
boston = fetch_openml(name="Boston", version=1, parser='auto')
boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)
boston_df['PRICE'] = boston.target
display(boston_df.head())
boston_df.info()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    category
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    category
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: category(2), float64(12)
memory usage: 49.0 KB
```

```
for col in boston_df.columns:
    if boston_df[col].dtype.name == 'category':
        # 카테고리형 데이터를 숫자로 변환 -> 범주형 데이터는 모델에 직접 사용할 수 없기 때문에 수치형으로 변환
        boston_df[col] = boston_df[col].cat.codes     ## 카테고리형 데이터를 각 카테고리에 할당된 숫자로 변환
        boston_df[col] = boston_df[col].astype(float)

X_data = boston_df.drop("PRICE", axis=1, inplace=False)
y_target = boston_df["PRICE"]
```

```
boston_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

```
# get_linear_reg_eval 사용자 함수
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.model_selection import cross_val_score
import numpy as np

def get_linear_reg_eval(model_name, params=None, X_data_n=None, y_target_n=None, verbose=True):
    coeff_df = pd.DataFrame()
    if verbose : print('######', model_name, '######')
    for param in params:
        if model_name =='Ridge': model = Ridge(alpha=param)
        elif model_name =='Lasso': model = Lasso(alpha=param)
        elif model_name =='ElasticNet': model = ElasticNet(alpha=param, l1_ratio=0.7)
        neg_mse_scores = cross_val_score(model, X_data_n, y_target_n, scoring="neg_mean_squared_error", cv=5)
        avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
        print(f'alpha {param}일 때 5 folds의 평균 RMSE : {avg_rmse}')
        model.fit(X_data_n, y_target_n)
        coeff=pd.Series(data=model.coef_, index=X_data_n.columns)
        colname='alpha:'+str(param)
        coeff_df[colname]=coeff
    return coeff_df
```

- ridge_alphas = [0, 0.1, 1, 10, 100]
- lasso_alphas = [0.07,0.1,0.5,1,3]
- elastic_alphas = [0.07,0.1,0.5,1,3], L1:L2 = 0.7:0.3

```
## 릿지 회귀
ridge_alphas=[0, 0.1, 1, 10, 100]
coeff_ridge_df=get_linear_reg_eval('Ridge', params=ridge_alphas, X_data_n=X_data, y_target_n=y_target)
coeff_ridge_df
```

```
###### Ridge ######
alpha 0일 때 5 folds의 평균 RMSE : 5.716928447470748
alpha 0.1일 때 5 folds의 평균 RMSE : 5.699712466274319
alpha 1일 때 5 folds의 평균 RMSE : 5.6406582449119185
alpha 10일 때 5 folds의 평균 RMSE : 5.5679786906719375
alpha 100일 때 5 folds의 평균 RMSE : 5.49923590644983
```

	alpha:0	alpha:0.1	alpha:1	alpha:10	alpha:100
CRIM	-0.065053	-0.064891	-0.063989	-0.062459	-0.058398
ZN	0.042019	0.042188	0.043162	0.045368	0.049593
INDUS	-0.054296	-0.057680	-0.076297	-0.103658	-0.119831
CHAS	3.083844	3.064099	2.926289	2.241946	0.743225
NOX	-15.309381	-14.393344	-9.352364	-2.065726	-0.224029
RM	4.113720	4.118217	4.133037	3.948086	2.506996
AGE	-0.004222	-0.005011	-0.009245	-0.013267	-0.000336
DIS	-1.502090	-1.488188	-1.411882	-1.303622	-1.215504
RAD	0.097228	0.096776	0.094484	0.094287	0.099261
TAX	0.001546	0.001373	0.000422	-0.000939	-0.001055
PTRATIO	-0.822800	-0.813403	-0.762389	-0.701501	-0.738285
B	0.008355	0.008411	0.008710	0.009040	0.008208
LSTAT	-0.515940	-0.517093	-0.524246	-0.550883	-0.658937

```
## 랏소 회귀
lasso_alphas=[0.07, 0.1, 0.5, 1,3]
coeff_lasso_df=get_linear_reg_eval('Lasso', params=lasso_alphas, X_data_n=X_data, y_target_n=y_target)
coeff_lasso_df
```

```
###### Lasso ######
alpha 0.07일 때 5 folds의 평균 RMSE : 5.63975318642497
alpha 0.1일 때 5 folds의 평균 RMSE : 5.639696014167153
alpha 0.5일 때 5 folds의 평균 RMSE : 5.721165611229234
alpha 1일 때 5 folds의 평균 RMSE : 5.910564972979698
alpha 3일 때 5 folds의 평균 RMSE : 6.225662896892307
```

	alpha:0.07	alpha:0.1	alpha:0.5	alpha:1	alpha:3
CRIM	-0.060899	-0.060272	-0.047400	-0.027592	-0.000000
ZN	0.044466	0.044379	0.043056	0.043497	0.036646
INDUS	-0.100872	-0.096489	-0.065961	-0.035079	-0.000000
CHAS	1.814288	1.342789	0.000000	0.000000	0.000000
NOX	-0.000000	-0.000000	-0.000000	-0.000000	0.000000
RM	4.041496	3.958515	2.765466	1.230766	0.000000
AGE	-0.014060	-0.012398	0.001813	0.018729	0.042256
DIS	-1.231606	-1.215476	-0.987331	-0.693531	-0.000000
RAD	0.073730	0.067353	0.000000	0.000000	0.000000
TAX	-0.001620	-0.001739	-0.002674	-0.003469	-0.005808
PTRATIO	-0.675171	-0.680718	-0.685039	-0.652850	-0.250913
B	0.009276	0.009267	0.008568	0.007424	0.006163
LSTAT	-0.550505	-0.558737	-0.647742	-0.754697	-0.810144

```
## 엘라스틱넷 회귀
elastic_alphas=[0.07, 0.1, 0.5, 1,3]
coeff_elastic_df=get_linear_reg_eval('ElasticNet', params=elastic_alphas, X_data_n=X_data, y_target_n=y_target)
coeff_elastic_df
```

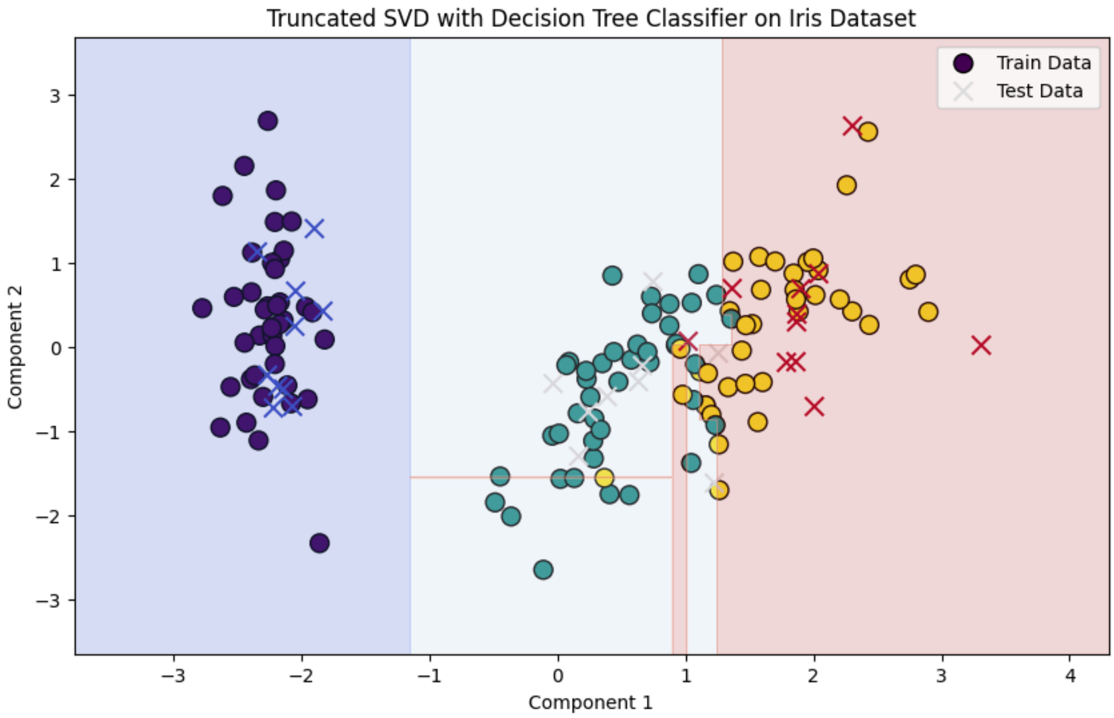
```
###### ElasticNet ######
alpha 0.07일 때 5 folds의 평균 RMSE : 5.5903593663528115
alpha 0.1일 때 5 folds의 평균 RMSE : 5.580020171066973
alpha 0.5일 때 5 folds의 평균 RMSE : 5.5950260435420205
alpha 1일 때 5 folds의 평균 RMSE : 5.773112322207657
alpha 3일 때 5 folds의 평균 RMSE : 6.137958613436409
```

	alpha:0.07	alpha:0.1	alpha:0.5	alpha:1	alpha:3
CRIM	-0.060944	-0.060245	-0.049926	-0.035752	-0.000000
ZN	0.045591	0.045875	0.046041	0.045563	0.036390
INDUS	-0.105402	-0.103734	-0.090124	-0.063498	-0.000000
CHAS	1.619638	1.247154	0.000000	0.000000	0.000000
NOX	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
RM	3.817480	3.653256	2.103637	1.070492	0.000000
AGE	-0.012495	-0.010434	0.006584	0.019592	0.042696
DIS	-1.244686	-1.231931	-1.028697	-0.771315	-0.030723
RAD	0.081576	0.077310	0.000000	0.000000	0.000000
TAX	-0.001503	-0.001560	-0.002155	-0.002844	-0.005168
PTRATIO	-0.686916	-0.694438	-0.719248	-0.674978	-0.397734
B	0.009127	0.009046	0.008065	0.007279	0.006442
LSTAT	-0.566374	-0.579565	-0.691720	-0.761681	-0.809659

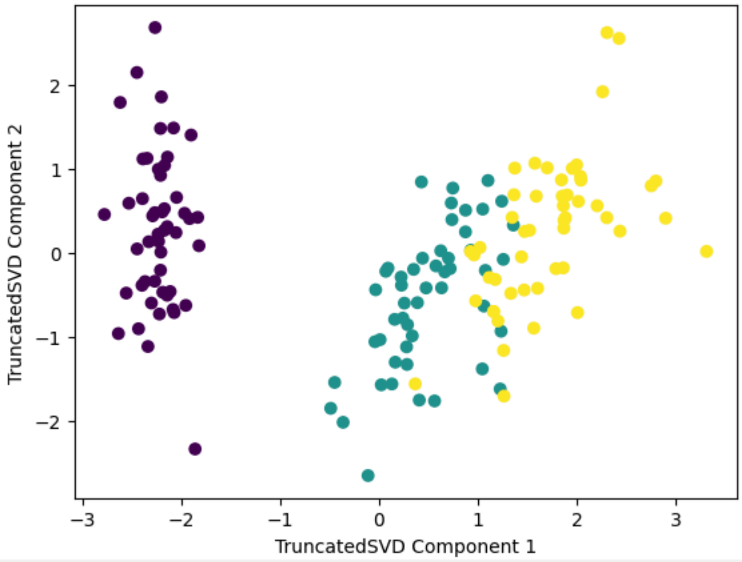
Q4. iris 데이터셋에 대하여 n_components=2를 적용하고 TruncatedSVD를 사용하여 추출된 2개의 component로 품종을 구분하는 것을 시각화 하세요.

```python
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

## 데이터 로드
iris = load_iris()
iris_ftrs = iris.data

X = iris.data
y = iris.target

# 데이터 표준화
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Truncated SVD로 차원 축소
svd = TruncatedSVD(n_components=2, random_state=42)
X_reduced = svd.fit_transform(X_scaled)

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.2, random_state=42)

# 결정 트리 분류기 학습
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# 예측 및 평가
from sklearn.metrics import accuracy_score, classification_report
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print('Classification Report:')
print(classification_rep)

# 시각화
plt.figure(figsize=(10, 6))

# Train 데이터 시각화
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis', edgecolor='k', s=100, label='Train Data')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='coolwarm', edgecolor='k', s=100, marker='x', label='Test Data')

# 경계 시각화
x_min, x_max = X_reduced[:, 0].min() - 1, X_reduced[:, 0].max() + 1
y_min, y_max = X_reduced[:, 1].min() - 1, X_reduced[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.2, cmap='coolwarm')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.title('Truncated SVD with Decision Tree Classifier on Iris Dataset')
plt.legend()
plt.show()
```

```
Accuracy: 0.9333333333333333
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.89      0.89      0.89         9
           2       0.91      0.91      0.91        11

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

<ipython-input-38-9184693fd42e>:43: UserWarning: You passed a edgecolor/edgecolors ('k') for an unfilled marker ('x').  Matplotlib is ignoring tl
  plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='coolwarm', edgecolor='k', s=100, marker='x', label='Test Data')
```




```
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler


iris = load_iris()
iris_ftrs = iris.data

# iris 데이터를 StandardScaler로 변환
scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_ftrs)

# 2개의 주요 component로 TruncatedSVD 변환
tsvd = TruncatedSVD(n_components=2)
tsvd.fit(iris_scaled)
iris_tsvd = tsvd.transform(iris_scaled)

# Scatter plot 2차원으로 TruncatedSVD 변환 된 데이터 표현. 품종은 색깔로 구분
plt.scatter(x=iris_tsvd[:,0], y= iris_tsvd[:,1], c= iris.target)
plt.xlabel('TruncatedSVD Component 1')
plt.ylabel('TruncatedSVD Component 2')
plt.show()
plt.close()
```



Q5. iris 데이터셋의 sepal length, sepal width, petal length, petal width 4개의 독립변수로 군집화를 수행 시 최적의 군집수를 산출하세요. 단, 군집개수별시뮬레이션을 시각화해서 최적의 군집수에 대한 이유도 설명

```
def visualize_silhouette(cluster_lists, X_features):
    from sklearn.datasets import make_blobs
    from sklearn.cluster import KMeans
    from sklearn.metrics import silhouette_samples, silhouette_score
    import matplotlib.pyplot as plt
    import matplotlib.cm as cm
    import math

    n_cols = len(cluster_lists)
    fig, axs = plt.subplots(figsize=(4*n_cols, 4), nrows=1, ncols=n_cols)
    for ind, n_cluster in enumerate(cluster_lists):
        clusterer = KMeans(n_clusters = n_cluster, n_init='auto', max_iter=500, random_state=0)
        cluster_labels = clusterer.fit_predict(X_features)
        sil_avg = silhouette_score(X_features, cluster_labels)
        sil_values = silhouette_samples(X_features, cluster_labels)
        y_lower = 10
        axs[ind].set_title('Number of Cluster : '+ str(n_cluster)+'₩n' ₩
                          'Silhouette Score :' + str(round(sil_avg,3)) )
        axs[ind].set_xlabel("The silhouette coefficient values")
        axs[ind].set_ylabel("Cluster label")
        axs[ind].set_xlim([-0.1, 1])
        axs[ind].set_ylim([0, len(X_features) + (n_cluster + 1) * 10])
        axs[ind].set_yticks([])  # Clear the yaxis labels / ticks
        axs[ind].set_xticks([0, 0.2, 0.4, 0.6, 0.8, 1])
        for i in range(n_cluster):
            ith_cluster_sil_values = sil_values[cluster_labels==i]
            ith_cluster_sil_values.sort()
            size_cluster_i = ith_cluster_sil_values.shape[0]
            y_upper = y_lower + size_cluster_i
            color = cm.nipy_spectral(float(i) / n_cluster)
            axs[ind].fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_sil_values, ₩
                                facecolor=color, edgecolor=color, alpha=0.7)
            axs[ind].text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
            y_lower = y_upper + 10
        axs[ind].axvline(x=sil_avg, color="red", linestyle="--")
```

코딩을 시작하거나 AI로 코드를 생성하세요.

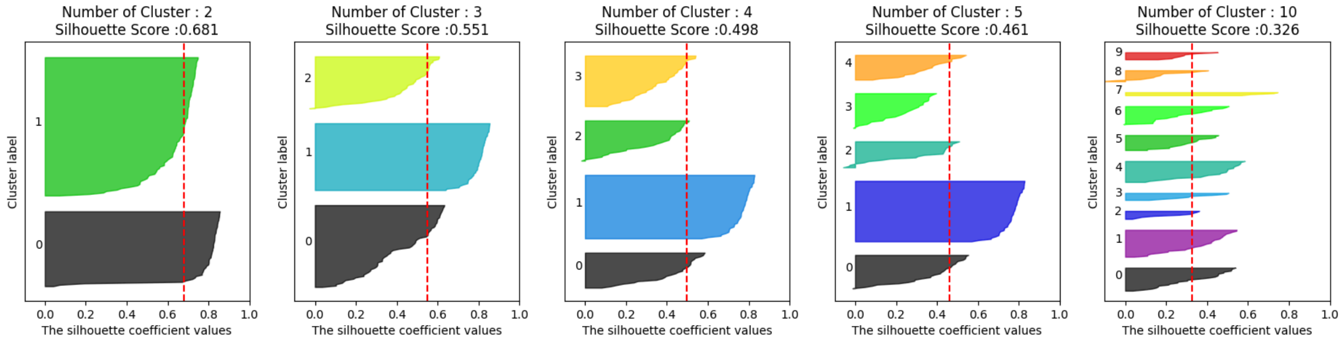
실루엣 다이어 그램

코딩을 시작하거나 AI로 코드를 생성하세요.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.datasets import load_iris

# 붓꽃 데이터 로드
iris = load_iris()
Feature_names = ['sepal_length','sepal_width','petal_length','petal_width']
df_iris = pd.DataFrame(data=iris.data, columns=Feature_names)
display(df_iris)
# 클러스터 개수를 2,3,4,5개일 때의 클러스터별 실루엣 계수 평균값을 시각화
## 4개의 군집일 때 가장 최적
visualize_silhouette([2,3,4,5,10], df_iris)
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

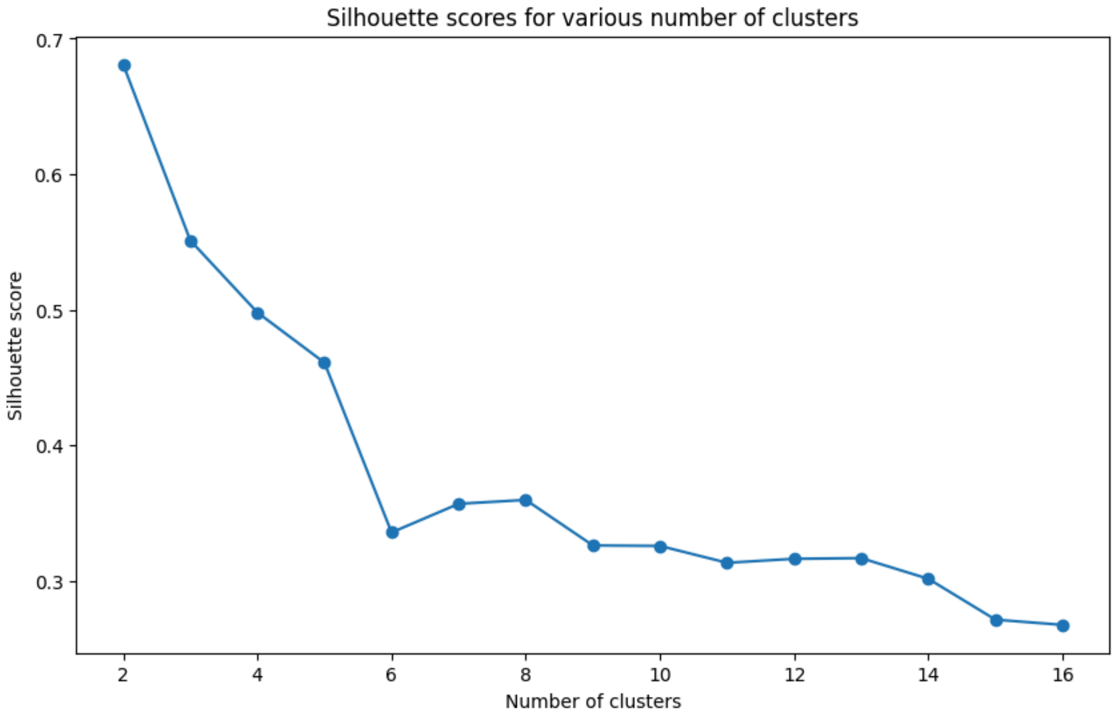


코딩을 시작하거나 AI로 코드를 생성하세요.

실루엣 스코어

코딩을 시작하거나 AI로 코드를 생성하세요.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.datasets import load_iris

# 붓꽃 데이터 로드
iris = load_iris()
iris_data = iris.data

# 군집 개수 후보 설정
range_n_clusters = list(range(2, 17))

silhouette_avg_scores = []

for n_clusters in range_n_clusters:
    # K-means 모델 초기화 및 학습
    kmeans = KMeans(n_clusters=n_clusters, n_init='auto',random_state=0)
    cluster_labels = kmeans.fit_predict(iris_data)
    # 실루엣 점수 계산
    silhouette_avg = silhouette_score(iris_data, cluster_labels)
    silhouette_avg_scores.append(silhouette_avg)
    # 각 샘플의 실루엣 계수 계산
    sample_silhouette_values = silhouette_samples(iris_data, cluster_labels)

# 실루엣 평균 점수 시각화
plt.figure(figsize=(10, 6))
plt.plot(range_n_clusters, silhouette_avg_scores, marker='o')
plt.title("Silhouette scores for various number of clusters")
plt.xlabel("Number of clusters")
plt.ylabel("Silhouette score")
plt.show()
```



코딩을 시작하거나 AI로 코드를 생성하세요.

실루엣 계수

코딩을 시작하거나 AI로 코드를 생성하세요.

```
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.preprocessing import scale
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

iris = load_iris()
Feature_names = ['sepal_length','sepal_width','petal_length','petal_width']
df_iris = pd.DataFrame(data=iris.data, columns=Feature_names)
kmeans=KMeans(n_clusters=3,n_init='auto', max_iter=300,random_state=0).fit(df_iris)
df_iris['cluster']=kmeans.labels_

score_samples=silhouette_samples(iris.data,df_iris['cluster'])
print('silhouette_samples() return 값의 shape : ', score_samples.shape)

df_iris['silhouette_coeff']=score_samples

average_score=silhouette_score(iris.data,df_iris['cluster'])
print('붓꽃 데이터 세트 Silhouette Analysis Score:{0:.4f}'.format(average_score))

df_iris.head(100)
# Calculate the average silhouette coefficient for each cluster using pivot table
pivot_table = df_iris.pivot_table(values='silhouette_coeff', index='cluster', aggfunc='mean')
print(pivot_table)
```

```
silhouette_samples() return 값의 shape :  (150,)
붓꽃 데이터 세트 Silhouette Analysis Score:0.5512
         silhouette_coeff
cluster
0                0.422323
1                0.797604
2                0.436842
```

코딩을 시작하거나 AI로 코드를 생성하세요.

최적의 군집수 평가: 3이 최적

- 최적의 군집수의 판단의 근거
  - 실루엣 스코어
  - 실루엣 계수
  - 실루엣 다이어그램의 형태 (너비/높이)

< Iris 데이터셋에서의 최적의 군집수 평가 근거 >

1. 실루엣 스코어: 이 값이 높을수록 군집화가 잘 됐다고 판단할 수 있지만 무조건 높다고 해서 군집화가 잘되었다고는 할 수 없다 iris 데이터셋의 경우 2~10까지 6에서 singular point 가 있지만 군집수 증가에 따라 score가 감소하는 형태
2. 실루엣 계수: 실루엣 계수는 -1에서 1 사이의 값, 1에 가까울수록 해당 데이터 포인트가 자신의 클러스터에 잘 속해 있음을 의미 iris 데이터셋의 경우 cluster 0/1에서 클러스터링 품질이 좋지 못함을 알수 있음
3. 실루엣 다이어그램의 형태: 일관된 너비와 높이가 좋은 클러스터링으로 볼 수 있지만, iris 데이터셋의 경우 , 군집수 4, 5, 특히 군집수 6이상에서는 특정 클러스터에서 peak가 보이는 문제가 있고, 클러스터 개수 2에서는 스코어 점수는 높지만, 하나의 클러스터가 다른 것보다 훨씬 많고 너비도 차이가 있음

==> 결론적으로는 군집수 4이상에서는 실루엣 스코어가 낮고, 클러스터의 너비 peak가 보이고, 2에서는 실루엣 스코어는 높지만, 실루엣 다이어그램의 형태에서 일관된 너비와 높이를 보이지 않아 군집수 3이 최적

iris 데이터셋의 경우 실루엣 계수로 부터 특정 클러스터 cluster 0/1에서 클러스터링 품질이 좋지 못함을 알 수 있음

코딩을 시작하거나 AI로 코드를 생성하세요.

실습과제1. 실습과제 코드를 작성하세요.

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
```

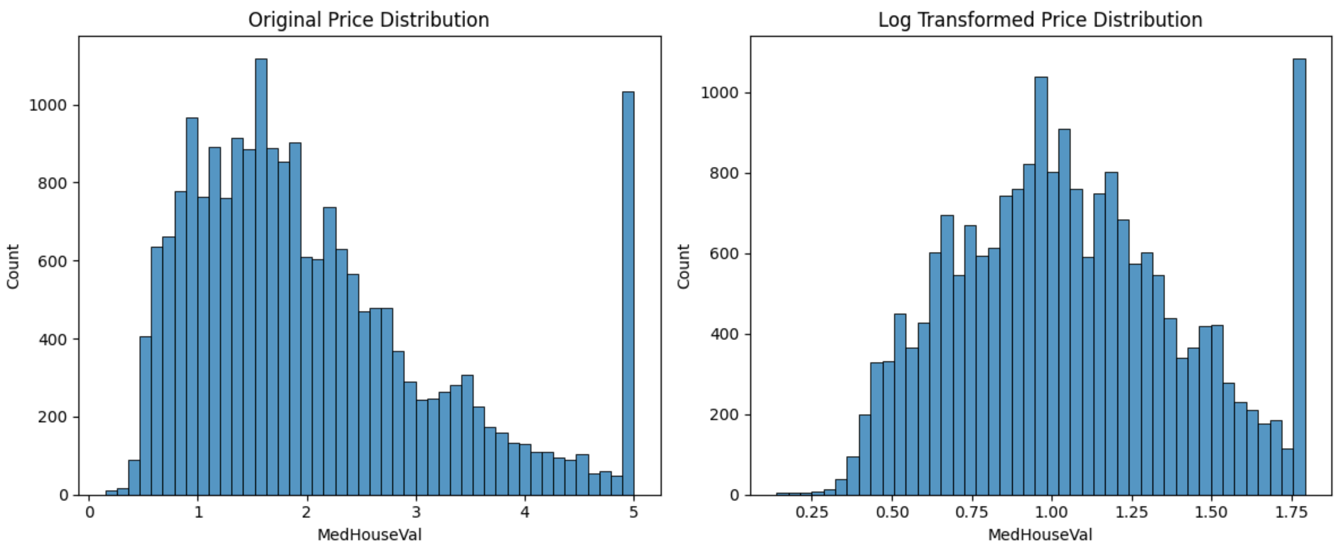
```
from sklearn.metrics import mean_squared_error, r2_score
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
```

코딩을 시작하거나 AI로 코드를 생성하세요.

```
# 1. 데이터 로드
from sklearn.datasets import fetch_california_housing
import pandas as pd
housing = fetch_california_housing()
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['MedHouseVal'] = housing.target
housing.feature_names
```

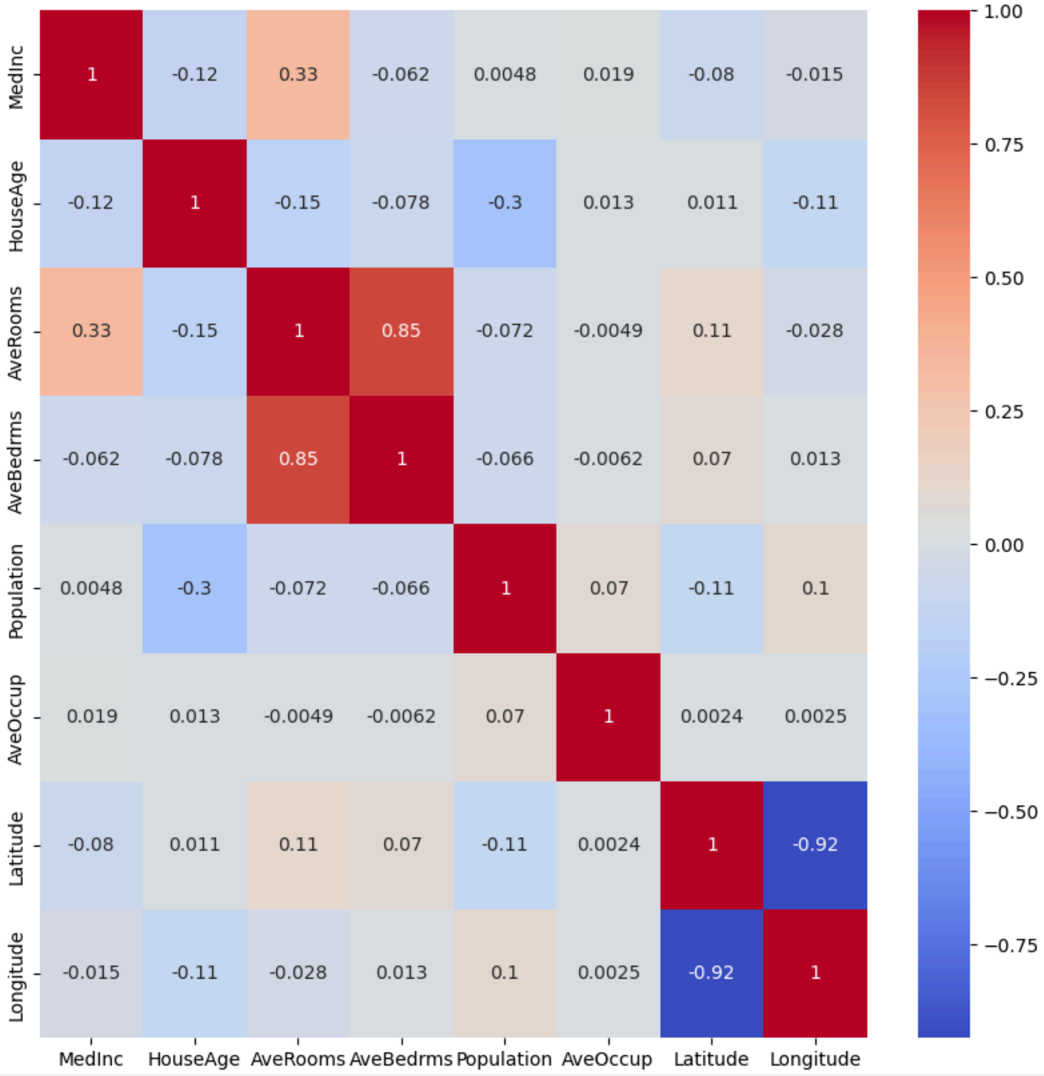
```
['MedInc',
 'HouseAge',
 'AveRooms',
 'AveBedrms',
 'Population',
 'AveOccup',
 'Latitude',
 'Longitude']
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

fig, axs = plt.subplots(figsize=(12, 5), ncols=2, nrows=1)

sns.histplot(df['MedHouseVal'], ax=axs[0])
axs[0].set_title('Original Price Distribution')

y_log = np.log1p(df['MedHouseVal'])
sns.histplot(y_log, ax=axs[1])
axs[1].set_title('Log Transformed Price Distribution')

plt.tight_layout()
plt.show()
```

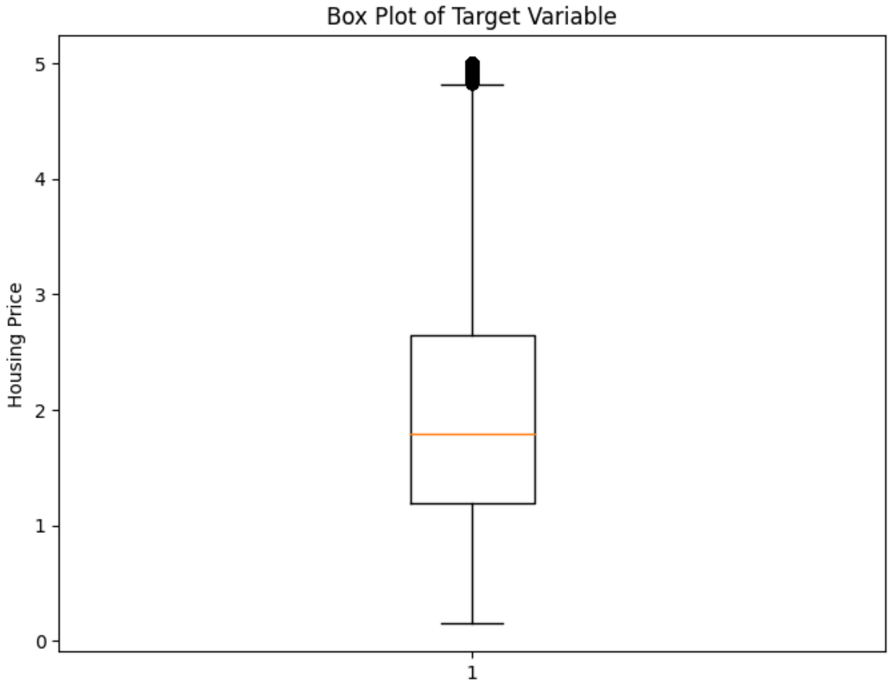


```
cdf = df[
    [
        "MedInc",
        "HouseAge",
        "AveRooms",
        "AveBedrms",
        "Population",
        "AveOccup",
        "Latitude",
        "Longitude",
    ]
]

corr = cdf.corr()

plt.figure(figsize=(10, 10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

<Axes: >




```
## 이상치 확인
housing=fetch_california_housing()
y=housing.target
# box plot
plt.figure(figsize=(8,6))
plt.boxplot(y)
plt.title("Box Plot of Target Variable")
plt.ylabel("Housing Price")
plt.show()
```




```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing

# Load the California housing dataset
housing = fetch_california_housing()
X = housing.data
y = housing.target

# Convert to a DataFrame for easier manipulation
df = pd.DataFrame(X, columns=housing.feature_names)
df['Target'] = y  # Add the target variable to the DataFrame

# Calculate IQR for the target variable
Q1 = df['Target'].quantile(0.25)
Q3 = df['Target'].quantile(0.75)
IQR = Q3 - Q1

# Define the bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify the outliers
outliers = df[(df['Target'] < lower_bound) | (df['Target'] > upper_bound)]

# Output the observations corresponding to outliers
print(len(outliers))
print(lower_bound)
print(upper_bound)
outliers[:5]
```

```
1071
-0.9808749999999995
4.824124999999999
```

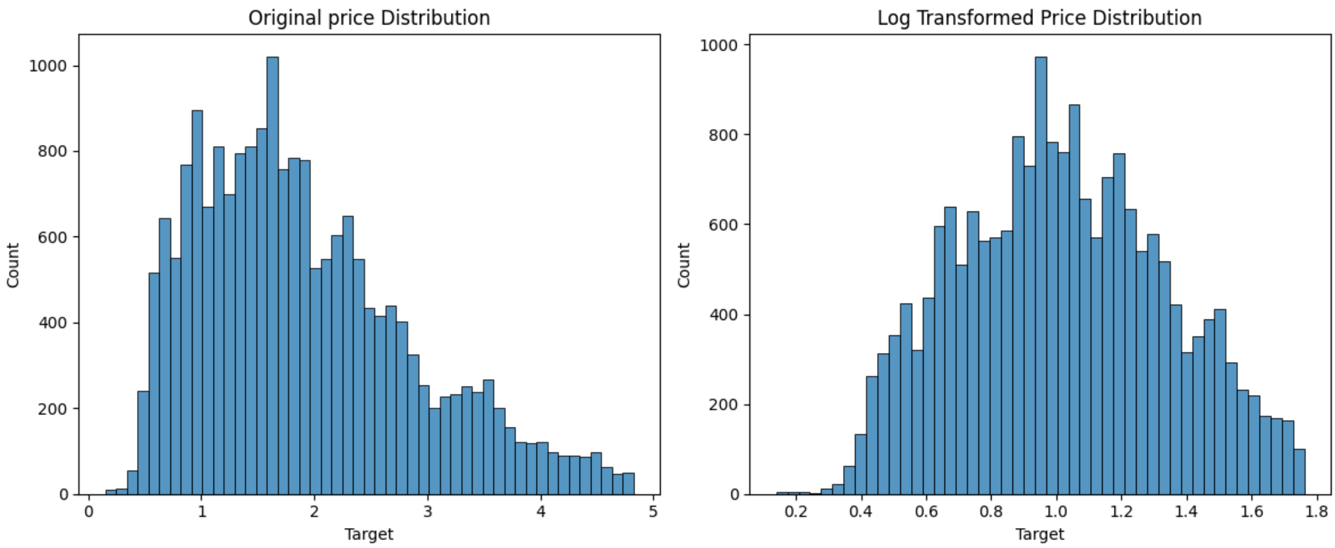
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Target
89	1.2434	52.0	2.929412	0.917647	396.0	4.658824	37.80	-122.27	5.00001
140	6.3624	30.0	5.615385	0.730769	126.0	2.423077	37.81	-122.18	4.83300
459	1.1696	52.0	2.436000	0.944000	1349.0	5.396000	37.87	-122.25	5.00001
489	3.0417	48.0	4.690632	1.126362	1656.0	3.607843	37.86	-122.25	4.89600
493	7.8521	52.0	7.794393	1.051402	517.0	2.415888	37.86	-122.24	5.00001

```
## Outlier 제거 후
## 20640-1071 = 19569

df_no_outliers=df[(df['Target']>=lower_bound) & (df['Target']<=upper_bound)]
df_no_outliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19569 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MedInc      19569 non-null  float64
 1   HouseAge    19569 non-null  float64
 2   AveRooms    19569 non-null  float64
 3   AveBedrms   19569 non-null  float64
 4   Population  19569 non-null  float64
 5   AveOccup    19569 non-null  float64
 6   Latitude    19569 non-null  float64
 7   Longitude   19569 non-null  float64
 8   Target      19569 non-null  float64
dtypes: float64(9)
memory usage: 1.5 MB
```

```
# histplot

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

fig, axs = plt.subplots(figsize=(12,5), ncols=2, nrows=1)

sns.histplot(df_no_outliers['Target'], ax=axs[0])
axs[0].set_title('Original price Distribution')

y_log = np.log1p(df_no_outliers['Target'])
sns.histplot(y_log, ax=axs[1])
axs[1].set_title('Log Transformed Price Distribution')
print(y_log.shape)
plt.tight_layout()
plt.show()
```

```
(19569,)
```



```python
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')

# 1. 데이터

# 파생 변수 생성
df_no_outliers['BedroomsPerRoom'] = df_no_outliers['AveBedrms'] / df_no_outliers['AveRooms']

X = df_no_outliers.drop(['Target'], axis=1)
y = df_no_outliers['Target']

# 데이터셋을 학습 세트와 테스트 세트로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 수치형 피처 목록
numerical_features = X.columns.tolist()

# 전처리기
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features)
    ])

# 모델과 하이퍼파라미터 그리드 딕셔너리
models = {
    "Linear Regression": {
        "model": LinearRegression(),
        "params": {}
    },
    "Ridge": {
        "model": Ridge(random_state=42),
        "params": {
            "classifier__alpha": [0.01, 0.1, 1, 10, 100]
        }
    },
    "Lasso": {
        "model": Lasso(random_state=42),
        "params": {
            "classifier__alpha": [0.01, 0.1, 1, 10, 100]
        }
    },
    "Elastic Net": {
        "model": ElasticNet(random_state=42),
        "params": {
            "classifier__alpha": [0.01, 0.1, 1, 10, 100],
            "classifier__l1_ratio": [0.1, 0.5, 0.9]
        }
    },
    "Decision Tree": {
        "model": DecisionTreeRegressor(random_state=42),
        "params": {
            "classifier__max_depth": [None, 10, 20, 30],
            "classifier__min_samples_split": [2, 10, 20],
            "classifier__min_samples_leaf": [1, 5, 10]
        }
    },
    "Random Forest": {
        "model": RandomForestRegressor(random_state=42),
        "params": {
            "classifier__n_estimators": [50, 100, 200],
            "classifier__max_depth": [None, 10, 20],
            "classifier__min_samples_split": [2, 10],
            "classifier__min_samples_leaf": [1, 5]
        }
    },
```

```
    "SVR": {
        "model": SVR(),
        "params": {
            "classifier__C": [0.1, 1, 10],
            "classifier__kernel": ['linear', 'rbf']
        }
    },
    "Gradient Boosting": {
        "model": GradientBoostingRegressor(random_state=42),
        "params": {
            "classifier__n_estimators": [50, 100, 200],
            "classifier__learning_rate": [0.01, 0.1, 0.5],
            "classifier__max_depth": [3, 5, 10]
        }
    },
    "AdaBoost": {
        "model": AdaBoostRegressor(random_state=42),
        "params": {
            "classifier__n_estimators": [50, 100, 200],
            "classifier__learning_rate": [0.01, 0.1, 0.5]
        }
    },
    "XGBoost": {
        "model": XGBRegressor(random_state=42),
        "params": {
            "classifier__n_estimators": [50, 100, 200],
            "classifier__learning_rate": [0.01, 0.1, 0.5],
            "classifier__max_depth": [3, 5, 10]
        }
    },
    "LightGBM": {
        "model": LGBMRegressor(random_state=42),
        "params": {
            "classifier__n_estimators": [50, 100, 200],
            "classifier__learning_rate": [0.01, 0.1, 0.5],
            "classifier__num_leaves": [31, 62, 124]
        }
    }
}

# 모델 학습 및 평가
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return mse, r2

best_estimators = {}
results = []

for model_name, model_info in models.items():
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', model_info["model"])
    ])

    grid_search = GridSearchCV(estimator=pipeline, param_grid=model_info["params"], cv=5, n_jobs=-1, scoring='neg_mean_squared_error'
    grid_search.fit(X_train, y_train)

    best_estimators[model_name] = grid_search.best_estimator_
    mse, r2 = evaluate_model(grid_search.best_estimator_, X_test, y_test)

    results.append({
        'Model': model_name,
        'Best Params': grid_search.best_params_,
        'MSE': mse,
        'R2': r2
    })

    print(f"모델: {model_name}")
    print(f"최적 하이퍼파라미터: {grid_search.best_params_}")
```