모델 성능 검증을 위한, 검증데이터 분리

모델의 성능을 검증하는 것은 중요합니다
단순히 학습 - 테스트 데이터 분리해서 한번의 테스트를 진행하기보다는, 다수의 데이터셋으로 구분하여 평균정확도를 검증합니다.
다음의 방법을 통해 데이터셋을 확보하여 진행합니다

- KFold 를 통한 데이터 분리
- GridSearchCV를 통한 하이퍼파라미터튜닝 및 검증

KFOLD 통한 데이터 분리

train_test_split이 아닌, KFold를 통해 데이터셋을 구분합니다.
검증을 위해 k개의 샘플 데이터셋을 구성하므로 (n_split을 통해 셋팅) GridSearchCV처럼 검증효과를 볼 수 있습니다

KFold

풀이 1 . np로 풀이

In [12]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

# 전체 데이터 확보
iris = load_iris()
data = iris.data
label_ = iris.target

# 검증 데이터셋 객체 셋팅
kfold = KFold(n_splits=5)
cv_accuracy = []


# 모델생성
clf = DecisionTreeClassifier(random_state=1)

n_iter = 0
for train_idx, test_idx in kfold.split(data):
    X_train, X_test = data[train_idx], data[test_idx]
    y_train, y_test = label_[train_idx], label_[test_idx]

    # 학습 및 예측
    clf.fit(X_train,y_train)
    pred = clf.predict(X_test)
    n_iter += 1

    # 평가
    acc = np.round(accuracy_score(y_test,pred),4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]

    print(f'n_iter : {n_iter}번째 \n 교차검증 정확도 : {acc}, 학습 데이터 크기 : {train_size}, 테스트 데이터 크기 :
    print("************************************************")
    print()
    cv_accuracy.append(acc)

# 평균검증정확도 계산
print(f'{n_iter} fold 검증 평균정확도 : {np.mean(cv_accuracy):.4f}')
```

```
n_iter : 1번째
 교차검증 정확도 : 1.0, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 2번째
 교차검증 정확도 : 1.0, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 3번째
 교차검증 정확도 : 0.9, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 4번째
 교차검증 정확도 : 0.9333, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 5번째
 교차검증 정확도 : 0.7333, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

5 fold 검증 평균정확도 : 0.9133
```

풀이2. df로 풀이

In [27]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

# 전체 데이터 확보
iris = load_iris()
df =pd.DataFrame(data = iris.data, columns= iris.feature_names)
df['label'] = iris.target

# 검증 데이터셋 객체 셋팅
kfold = KFold(n_splits=5)
cv_accuracy = []


# 모델생성
clf = DecisionTreeClassifier(random_state=1)

n_iter = 0
for train_idx, test_idx in kfold.split(df):
    X_train, X_test = df.loc[train_idx, df.columns.difference(['label'])], df.loc[test_idx, df.columns.differe
    y_train, y_test = df.loc[train_idx, 'label'], df.loc[test_idx, 'label']

    # 학습 및 예측
    clf.fit(X_train,y_train)
    pred = clf.predict(X_test)
    n_iter += 1

    # 평가
    acc = np.round(accuracy_score(y_test,pred),4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]

    print(f'n_iter : {n_iter}번째 \n 교차검증 정확도 : {acc}, 학습 데이터 크기 : {train_size}, 테스트 데이터 크기 :
    print("************************************************")
    print()
    cv_accuracy.append(acc)

# 평균검증정확도 계산
print(f'{n_iter} fold 검증 평균정확도 : {np.mean(cv_accuracy):.4f}')
```

```
n_iter : 1번째
 교차검증 정확도 : 1.0, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 2번째
 교차검증 정확도 : 1.0, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 3번째
 교차검증 정확도 : 0.8333, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 4번째
 교차검증 정확도 : 0.9333, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 5번째
 교차검증 정확도 : 0.8, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

5 fold 검증 평균정확도 : 0.9133
```

StratifiedKFold

종속변수 (타겟값)이 다중분류인 경우는 stratifiedkfold를 적용하여 샘플링을 하면 보다 정확한 결과를 얻을 수 있음

In [30]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score

# 전체 데이터 확보
iris = load_iris()
df =pd.DataFrame(data = iris.data, columns= iris.feature_names)
df['label'] = iris.target

# 검증 데이터셋 객체 셋팅
skfold = StratifiedKFold(n_splits=5)
cv_accuracy = []


# 모델생성
clf = DecisionTreeClassifier(random_state=1)

n_iter = 0
for train_idx, test_idx in skfold.split(df, df['label']):
    X_train, X_test = df.loc[train_idx, df.columns.difference(['label'])], df.loc[test_idx, df.columns.differe
    y_train, y_test = df.loc[train_idx, 'label'], df.loc[test_idx, 'label']

    # 학습 및 예측
    clf.fit(X_train,y_train)
    pred = clf.predict(X_test)
    n_iter += 1

    # 평가
    acc = np.round(accuracy_score(y_test,pred),4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]

    print(f'n_iter : {n_iter}번째 \n 교차검증 정확도 : {acc}, 학습 데이터 크기 : {train_size}, 테스트 데이터 크기 :
    print("************************************************")
    print()
    cv_accuracy.append(acc)

# 평균검증정확도 계산
print(f'{n_iter} fold 검증 평균정확도 : {np.mean(cv_accuracy):.4f}')
```

```
n_iter : 1번째
 교차검증 정확도 : 0.9667, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 2번째
 교차검증 정확도 : 0.9667, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 3번째
 교차검증 정확도 : 0.9, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 4번째
 교차검증 정확도 : 1.0, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

n_iter : 5번째
 교차검증 정확도 : 1.0, 학습 데이터 크기 : 120, 테스트 데이터 크기 : 30
************************************************

5 fold 검증 평균정확도 : 0.9667
```

GridSearchCV

train_test_split 된 데이터에 대하여, 교차검증과 최적의 하이퍼파라미터* 튜닝을 동시에 하기위해, GridSearchCV를 활용합니다.

- 하이퍼파라미터 : 모델에 적용/변경하는 셋팅 값으로, 데이터 형태에 따라 모델의 성능을 높이기 위해서 최적의 값을 찾아줄 필요가 있습니다.

In [38]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# 전체 데이터 확보
iris = load_iris()
df =pd.DataFrame(data = iris.data, columns= iris.feature_names)
df['label'] = iris.target

X_train, X_test, y_train, y_test = train_test_split(df[df.columns.difference(['label'])], df['label'], test_s

# 검증할 하이퍼파라미터 그리드 생성
grid_parameters = {'max_depth':[1,2,3],
'min_samples_split':[2,3]
}

clf = DecisionTreeClassifier()
grid_clf = GridSearchCV(clf, param_grid= grid_parameters, cv=3, refit = True)
#refit=True 가 default로, 가장 성능 높은 값으로 선택

grid_clf.fit(X_train, y_train) # 학습

score_df = pd.DataFrame(grid_clf.cv_results_)
score_df[['params','mean_test_score', 'rank_test_score',
       'split0_test_score', 'split1_test_score', 'split2_test_score'
       ]]
```

Out[38]:

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'max_depth': 1, 'min_samples_split': 2}	0.675000	5	0.675	0.675	0.675
1	{'max_depth': 1, 'min_samples_split': 3}	0.675000	5	0.675	0.675	0.675
2	{'max_depth': 2, 'min_samples_split': 2}	0.916667	3	0.900	0.900	0.950
3	{'max_depth': 2, 'min_samples_split': 3}	0.916667	3	0.900	0.900	0.950
4	{'max_depth': 3, 'min_samples_split': 2}	0.950000	1	1.000	0.900	0.950
5	{'max_depth': 3, 'min_samples_split': 3}	0.933333	2	0.950	0.900	0.950

In [43]:

```
# 최적 파라미터 적용
estimator = grid_clf.best_estimator_
print(f'GridSearchCV 최고 정확도 : {grid_clf.best_score_:.4f}')
# 테스트세트 예측 및 검증
pred = estimator.predict(X_test)
print(f'테스트세트 정확도 : {accuracy_score(y_test,pred)}')
```

```
GridSearchCV 최고 정확도 : 0.9500
테스트세트 정확도 : 1.0
```