

# 02\_머신러닝 시작

<학습내용>

0. 머신러닝 시험 운용

1. 전처리

2. 학습

3. 평가

4. 분류, 회귀, 군집 알고리즘

# 학습흐름 ; 학습->예측->평가 => 반복

- 사이킷런 소개와 특징

- > 사이킷런(scikit-learn)은 파이썬에서 머신러닝 학습을 위한 라이브러리이다.
- > <https://scikit-learn.org>

```
import sklearn  
print(sklearn.__version__)
```

scikit-learn  
머신러닝 필수 라이브러리

**Out :** 1.3.0

## • 머신러닝 따라하기

### > 붓꽃 품종 분류하기

- 꽃잎의 길이와 너비, 꽃받침의 길이와 너비 4개의 피쳐를 통해 붓꽃 품종을 분류한다.
- 종속변수는 아래의 3가지 품종의 종류이다.

**iris setosa**



petal    sepal

**iris versicolor**



petal    sepal

**iris virginica**



petal    sepal

## • 머신러닝 따라하기

- 붓꽃 품종 분류하기

### > 라이브러리 호출

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

의사결정나무 - 분류알고리즘  
학습데이터와 테스트데이터 나누는 모듈

`datasets`은 사이킷런의 데이터 세트를 생성하는 모듈이다.

`tree`는 트리 기반 ML 알고리즘을 구현하는 모듈이다.

`model_selection`은 학습 데이터와 검증 데이터, 예측 데이터로 데이터를 분리하는 다양한 모듈이다.

`train_test_split` 함수는 데이터를 학습, 테스트 데이터로 분리하는 함수이다.

## • 머신러닝 따라하기

- 붓꽃 품종 분류하기

### > 데이터 살펴보기

```
import pandas as pd
iris = load_iris()
iris_data = iris.data
iris_label = iris.target
print('iris target값:', iris_label)
print('iris target명:', iris.target_names)
iris_df = pd.DataFrame(data=iris_data, columns = iris.feature_names)
iris_df['label'] = iris.target
iris_df.head(3)
```

- 머신러닝 따라하기

- ## ■ 붓꽃 품종 분류하기

## > 데이터 살펴보기

```
Out: iris target값: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2]
iris target명: ['setosa' 'versicolor' 'virginica']
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

## • 머신러닝 따라하기

- 붓꽃 품종 분류하기

> 학습, 테스트 데이터 나누기

데이터 전처리

```
X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_label,  
                                                    test_size = 0.2, random_state = 11)  
  
print(X_train[0])
```

**Out :** [5.1 3.5 1.4 0.2]

```
train_test_split(  
    *arrays,  
    test_size=None,  
    train_size=None,  
    random_state=None,  
    shuffle=True,  
    stratify=None,
```

수량 제한 없음  
보통 train과 8:2, 7:3으로 나눔

## • 머신러닝 따라하기

- 붓꽃 품종 분류하기

> 학습하기      # 학습흐름 ; 학습->예측->평가 => 반복 / 성능향상을 위한 작업3

```
# 학습 / 예측
```

```
dt_clf = DecisionTreeClassifier(random_state = 11)
```

```
dt_clf.fit(X_train, y_train)
```

```
pred = dt_clf.predict(X_test)      예측
```

```
# 평가
```

```
from sklearn.metrics import accuracy_score
```

```
acc = accuracy_score(y_test, pred)      정확도 측정
```

```
print('예측 정확도: {0:.4f}'.format(acc))
```

**Out :** 예측 정확도: 0.9333

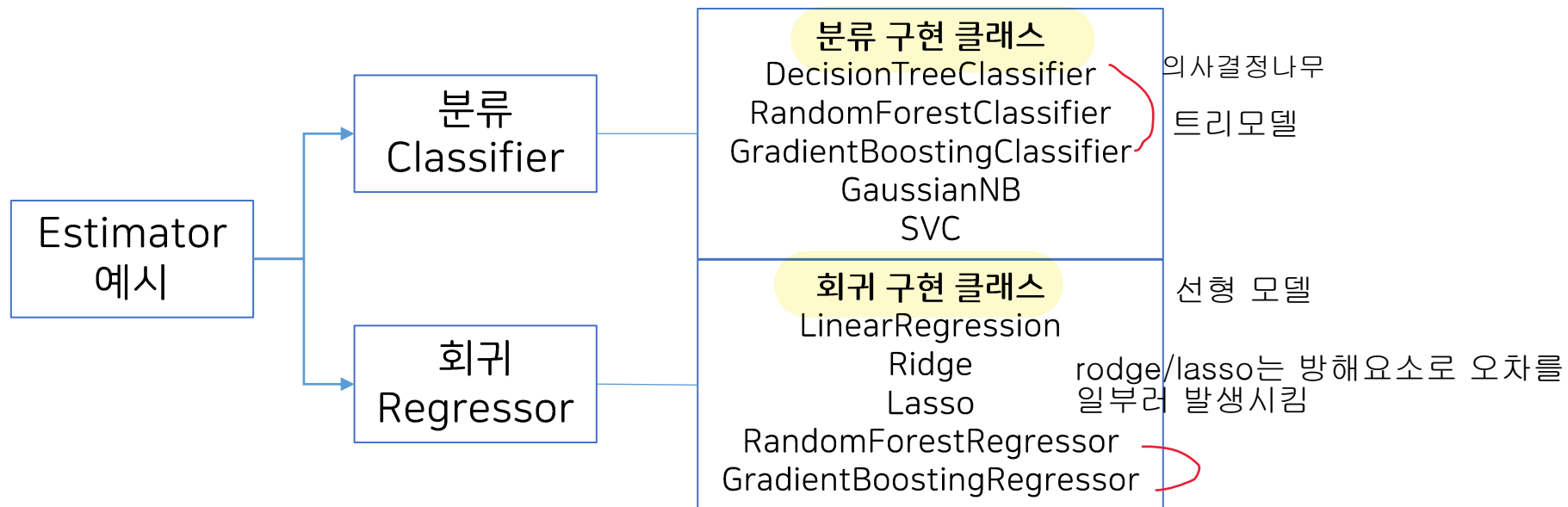


## • 사이킷런의 기반 프레임워크 익히기

09

### > Estimator 이해

- 사이킷런은 일관성과 개발 편의성을 제공하기 위해 통일된 알고리즘의 클래스를 제공함.
- 이를 Estimator라고 한다.
- fit()과 predict() 또는 fit()과 transform()을 내장하고 있다.
- 이러한 Estimator는 다양한 모듈에 들어가 있고, 통일된 사용법을 제공한다.



• 사이킷런의 기반 프레임워크 익히기

010



사이킷런의 주요 모듈

분류	모듈명	설명	
예제 데이터	datasets	예제로 제공하는 데이터 세트	<code>from sklearn.datasets import load_iris/ OneHotEncoder /t StandardScaler/ MinMaxScaler /</code>
피처 처리 변수 처리 (target도 포함)	preprocessing	데이터 전처리에 필요한 기능 제공	<code>from sklearn.preprocessing import LabelEncoder</code>
	feature_selection	피처를 우선순위대로 선택션 작업을 수행하는 다양한 기능 제공	
	feature_extraction	텍스트 데이터나 이미지 데이터의 벡터화 기능 제공	
피처 처리 & 차원 축소	decomposition	차원 축소와 관련한 알고리즘	
데이터 분리, 검증 & 파라미터 튜닝	model_selection	학습에 도움이 되는 다양한 기능 제공	<code>from sklearn.model_selection import train_test_split StratifiedKFold/ cross_val_score/ GridSearchCV</code>
평가	metrics	학습된 모델에 대한 성능 측정 방법 제공	<code>from sklearn.metrics import accuracy_score</code>
	ensemble	앙상블 알고리즘	<code>from sklearn.ensemble import RandomForestClassifier</code>
	linear_model	선형 회귀 알고리즘	
ML 알고리즘	naive_bayes	나이브 베이즈 알고리즘	
	neighbors	최근접 이웃 알고리즘	
	svm	서포트 벡터 머신 알고리즘	
	tree	의사 결정 트리 알고리즘	<code>from sklearn.tree import DecisionTreeClassifier</code>
	cluster	비지도 학습 알고리즘 군집 관련 알고리즘	
유틸리티	pipeline	전처리, 학습, 예측 등 함께 묶어서 실행할 수 있는 유틸리티 제공	

## • 사이킷런의 기반 프레임워크 익히기

011

> load\_datasets

```
from sklearn.datasets import load_iris
```

```
iris_data = load_iris()
```

```
print(type(iris_data))
```

train\_test\_split으로 나눔

**Out :** <class 'sklearn.utils.Bunch'>

```
keys = iris_data.keys()
```

```
print(keys)
```

독립변수

**Out :** dict\_keys(['data', 'target', 'frame', 'target\_names', 'DESCR', 'feature\_names', 'filename', 'data\_module'])

독립변수의 이름

- 사이킷런의 기반 프레임워크 익히기

012

> train\_test\_split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data.target,
                                                    test_size = 0.2, random_state = 11)

print('학습 데이터의 수:', len(X_train))
print('테스트 데이터의 수:', len(X_test))  --> .shape 로 확인가능
```

**Out :** 학습 데이터의 수: 120  
테스트 데이터의 수: 30

## • 교차검증

데이터가 너무 적을때  
결과를 일반화 검증할 때 사용

013

### > 교차검증이란?

- 보통의 데이터는 학습, 테스트 데이터로 모델을 검증한다.
- 하지만 고정된 테스트 데이터로만 검증을 반복하면 테스트 데이터에만 최적화되어 실제 데이터에서 성능이 떨어지게 된다.
- 이를 해결하기 위해 학습 데이터에서 검증 데이터를 분리한 뒤, 검증 데이터를 통해 검증한다.

### > 장점

- 모든 데이터셋을 훈련, 평가에 활용할 수 있다.
- 정확도를 향상시킬 수 있다.
- 데이터 편종을 막을 수 있다.

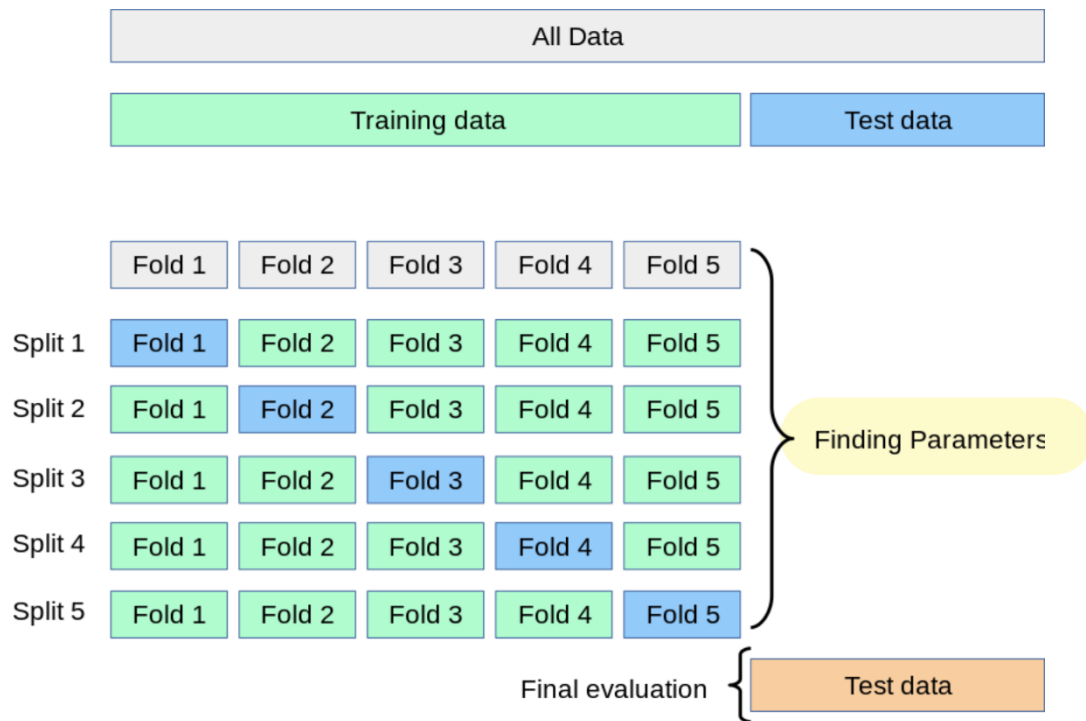
### > 단점

- 학습 횟수가 증가하기때문에 오랜 시간이 걸린다.
- 같은 데이터를 여러 번 학습하기에 과적합이 걸릴 수 있다.

## • 교차검증

### > K겹 교차 검증

- 보편적으로 많이 사용되는 교차 검증 기법
- K번 만큼 데이터를 나누어 돌아가며 학습, 검증 데이터로 사용
- 데이터가 독립적이고 동일한 분포를 가진 경우에 사용



train\_test\_split으로 나눔

## • 교차검증

### > K겹 교차 검증 절차

- 전체 데이터셋을 학습, 테스트 데이터로 나눈다.
- 학습 데이터를 K개의 폴드로 나눈다.
- 첫 번째 폴드를 검증 데이터로 사용하고 나머지 데이터는 학습 데이터로 사용한다.
- 모델을 학습한 뒤, 검증 데이터로 평가한다.
- 차례대로 다음 폴드를 사용하여 반복한다.
- 총 K개의 성능 결과가 나오며, 이 K개의 평균을 해당 학습 모델의 성능이라고 한다.

- 교차검증

- > 교차검증 해보기

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

iris = load_iris()
features = iris.data
label = iris.target
dt_clf = DecisionTreeClassifier(random_state = 156)
```



- 교차검증

- > 교차검증 해보기

```
kfold = KFold(n_splits = 5)
cv_accuracy = []
n_iter = 0

for train_index, test_index in kfold.split(features):
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]

    dt_clf.fit(X_train, y_train)
    pred = dt_clf.predict(X_test)
    n_iter += 1
```

## • 교차검증

### > 교차검증 해보기

```
# 이전 코드에 이어서 작성
accuracy = np.round(accuracy_score(y_test, pred), 4)
train_size = X_train.shape[0]
test_size = X_test.shape[0]
print('#',n_iter)
print('교차 검증 정확도 :',accuracy)
print('학습데이터 크기 :',train_size)
print('검증데이터 크기 :',test_size)
print('검증 인덱스 :',test_index)
cv_accuracy.append(accuracy)

print('평균 검증 정확도 :', np.mean(cv_accuracy))
```

## • 교차검증

### > 교차검증 해보기

**Out :**

```
# 1
교차 검증 정확도 : 1.0
학습데이터 크기 : 120
검증데이터 크기 : 30
검증 인덱스 : [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29]

# 2
교차 검증 정확도 : 0.9667
학습데이터 크기 : 120
검증데이터 크기 : 30
검증 인덱스 : [30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
 54 55 56 57 58 59]

# 3
교차 검증 정확도 : 0.8667
학습데이터 크기 : 120
검증데이터 크기 : 30
검증 인덱스 : [60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
 84 85 86 87 88 89]
```

평균 검증 정확도 : 0.9

## • 교차검증

### > 교차검증 해보기

```
import pandas as pd

iris = load_iris()
iris_df = pd.DataFrame(data = iris.data, columns = iris.feature_names)
iris_df['label'] = iris.target
iris_df.head(3)
```

Out :

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

- 교차검증

- > 교차검증 해보기

```
kfold = KFold(n_splits = 3)
n_iter = 0
for train_index, test_index in kfold.split(iris_df):
    n_iter += 1
    label_train = iris_df['label'].iloc[train_index]
    label_test = iris_df['label'].iloc[test_index]
    print(f'#{n_iter} 교차검증')
    print('학습레이블 분포 :\n', label_train.value_counts())
    print('검증레이블 분포 :\n', label_test.value_counts())
```

## • 교차검증

### > 교차검증 해보기

Out :

#1 교차검증

학습레이블 분포 :

1 50

2 50

Name: label, dtype: int64

검증레이블 분포 :

0 50

Name: label, dtype: int64

#2 교차검증

학습레이블 분포 :

0 50

2 50

Name: label, dtype: int64

검증레이블 분포 :

1 50

Name: label, dtype: int64

#3 교차검증

학습레이블 분포 :

0 50

1 50

Name: label, dtype: int64

검증레이블 분포 :

2 50

Name: label, dtype: int64

- 교차검증

- > Stratified K-Fold

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits = 3)
n_iter = 0

for train_index, test_index in skf.split(iris_df, iris_df['label']):
    n_iter += 1
    label_train = iris_df['label'].iloc[train_index]
    label_test = iris_df['label'].iloc[test_index]
    print(f'#{n_iter} 교차검증')
    print('학습레이블 분포 :\n', label_train.value_counts())
    print('검증레이블 분포 :\n', label_test.value_counts())
```

- 교차검증

- > Stratified K-Fold

Out :

```
#1 교차검증
학습레이블 분포 :
 2    34
0    33
1    33
Name: label, dtype: int64
검증레이블 분포 :
 0    17
1    17
2    16
Name: label, dtype: int64
#2 교차검증
학습레이블 분포 :
 1    34
0    33
2    33
Name: label, dtype: int64
검증레이블 분포 :
 0    17
2    17
1    16
Name: label, dtype: int64
```

```
#3 교차검증
학습레이블 분포 :
 0    34
1    33
2    33
Name: label, dtype: int64
검증레이블 분포 :
 1    17
2    17
0    16
Name: label, dtype: int64
```



- 교차검증

- > Stratified K-Fold

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

iris = load_iris()
features = iris.data
label = iris.target
dt_clf = DecisionTreeClassifier(random_state = 156)
```

- 교차검증

- > Stratified K-Fold

```
skf = StratifiedKFold(n_splits = 5)
cv_accuracy = []
n_iter = 0

for train_index, test_index in skf.split(features, label):
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]

    dt_clf.fit(X_train, y_train)
    pred = dt_clf.predict(X_test)
    n_iter += 1
```

## • 교차검증

### > Stratified K-Fold

Type text here

```
# 이전 코드에 이어서 작성
accuracy = np.round(accuracy_score(y_test, pred), 4)
train_size = X_train.shape[0]
test_size = X_test.shape[0]
print('#',n_iter)
print('교차 검증 정확도 :',accuracy)
print('학습데이터 크기 :',train_size)
print('검증데이터 크기 :',test_size)
print('검증 인덱스 :',test_index)
cv_accuracy.append(accuracy)

print('평균 검증 정확도 :', np.mean(cv_accuracy))
```

## • 교차검증

### > Stratified K-Fold

**Out :**

```
# 1
교차 검증 정확도 : 0.9667
학습데이터 크기 : 120
검증데이터 크기 : 30
검증 인덱스 : [ 0  1  2  3  4  5  6  7  8  9 50 51 52 53 54 55 56 57
58 59 100 101 102 103 104 105 106 107 108 109]

# 2
교차 검증 정확도 : 0.9667
학습데이터 크기 : 120
검증데이터 크기 : 30
검증 인덱스 : [ 10 11 12 13 14 15 16 17 18 19 60 61 62 63 64 65 66 67
68 69 110 111 112 113 114 115 116 117 118 119]

# 3
교차 검증 정확도 : 0.9
학습데이터 크기 : 120
검증데이터 크기 : 30
검증 인덱스 : [ 20 21 22 23 24 25 26 27 28 29 70 71 72 73 74 75 76 77
78 79 120 121 122 123 124 125 126 127 128 129]
```

평균 검증 정확도 : 0.9600200000000001

## • 교차검증

- 교차 검증을 보다 간편하게 처리하기

> `cross_val_score()`



기본으로 stratified K-fold를 따른다.

```
from sklearn.model_selection import cross_val_score

iris_data = load_iris()
dt_clf = DecisionTreeClassifier(random_state= 156)
data = iris_data.data
label = iris_data.target
scores = cross_val_score(dt_clf, data, label, scoring = 'accuracy', cv = 5)
print('교차 검증별 정확도: ', np.round(scores, 4))
print('평균 검증 정확도: ', np.round(np.mean(scores), 4))
```

**Out :** 교차 검증별 정확도: [0.9667 0.9667 0.9 0.9667 1. ]

평균 검증 정확도: 0.96

## • 교차검증

- 교차 검증을 보다 간편하게 처리하기

> `cross_validate()`

```
cross_validate(dt_clf, data, label, scoring = ['accuracy', 'roc_auc_ovo'], cv = 5)
```

```
Out: {'fit_time': array([0.00099707, 0.00099063, 0.00099993, 0.00099993, 0.00099993]),
      'score_time': array([0.00401044, 0.00200033, 0.00200081, 0.00200033, 0.00200057]),
      'test_accuracy': array([0.96666667, 0.96666667, 0.96666667, 0.96666667, 0.96666667]),
      'test_roc_auc_ovo': array([0.975, 0.975, 0.925, 0.975, 1.0])}
```

> `cross_val_predict()` 예측결과 출력

```
cross_val_predict(dt_clf, data, label, cv = 5)
```

```
Out: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2,
            2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2,
            2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

- 하이퍼 파라미터 :모델의 중요한 매개변수들

## > GridSearchCV

```
from sklearn.model_selection import GridSearchCV

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
                                                    test_size=0.2, random_state=121)

dtree = DecisionTreeClassifier()
parameters = {'max_depth':[1,2,3], 'min_samples_split':[2,3]}
grid_dtree = GridSearchCV(dtree, param_grid=parameters, cv=3, refit=True)
grid_dtree.fit(X_train, y_train)
```

- 하이퍼 파라미터

032

- > GridSearchCV

```
import pandas as pd
```

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

Out :

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'max_depth': 1, 'min_samples_split': 2}	0.700000	5	0.700	0.7	0.70
1	{'max_depth': 1, 'min_samples_split': 3}	0.700000	5	0.700	0.7	0.70
2	{'max_depth': 2, 'min_samples_split': 2}	0.958333	3	0.925	1.0	0.95
3	{'max_depth': 2, 'min_samples_split': 3}	0.958333	3	0.925	1.0	0.95
4	{'max_depth': 3, 'min_samples_split': 2}	0.975000	1	0.975	1.0	0.95
5	{'max_depth': 3, 'min_samples_split': 3}	0.975000	1	0.975	1.0	0.95



- 하이퍼 파라미터

033

> GridSearchCV

```
print('GridSearchCV 최적 파라미터:', grid_dtree.best_params_)  
print(f'GridSearchCV 최고 정확도: {grid_dtree.best_score_:.4f}')
```

**Out :** GridSearchCV 최적 파라미터: {'max\_depth': 3,  
                                  'min\_samples\_split': 2}  
GridSearchCV 최고 정확도: 0.9750

```
estimator = grid_dtree.best_estimator_  
pred = estimator.predict(X_test)  
print(f'테스트 데이터 세트 정확도:{accuracy_score(y_test,pred):.4f}')
```

**Out :** 테스트 데이터 세트 정확도: 0.9667

## • 결정 트리 실습

### > GridSearchCV

```
from sklearn.model_selection import GridSearchCV
params = {'max_depth' : [ 6, 8 ,10, 12, 16 ,20, 24]}
grid_cv = GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, verbose=1 )
grid_cv.fit(X_train , y_train)
print(f'GridSearchCV 최고 정확도 수치:{grid_cv.best_score_:4f}')
print('GridSearchCV 최적 하이퍼 파라미터:', grid_cv.best_params_)
```

**Out :** Fitting 5 folds for each of 7 candidates, totalling 35 fits  
GridSearchCV 최고 정확도 수치:0.9500  
GridSearchCV 최적 하이퍼 파라미터: {'max\_depth': 6}

- 결정 트리 실습

035

> GridSearchCV

st

```
cv_results_df = pd.DataFrame(grid_cv.cv_results_)
```

```
cv_results_df[['param_max_depth', 'mean_test_score']]
```

Out :

	param_max_depth	mean_test_score
0	6	0.95
1	8	0.95
2	10	0.95
3	12	0.95
4	16	0.95
5	20	0.95
6	24	0.95

## • 결정 트리 실습

### > 최적의 하이퍼 파라미터 튜닝

```
max_depths = [ 6, 8 ,10, 12, 16 ,20, 24]
for depth in max_depths:
    dt_clf = DecisionTreeClassifier(max_depth=depth, min_samples_split=16,
                                    random_state=156)

    dt_clf.fit(X_train , y_train)
    pred = dt_clf.predict(X_test)
    accuracy = accuracy_score(y_test , pred)
    print(f'max_depth = {depth} 정확도: {accuracy:.4f}')
```

**Out :**

```
max_depth = 6 정확도: 0.9667
max_depth = 8 정확도: 0.9667
max_depth = 10 정확도: 0.9667
max_depth = 12 정확도: 0.9667
max_depth = 16 정확도: 0.9667
max_depth = 20 정확도: 0.9667
max_depth = 24 정확도: 0.9667
```

## • 결정 트리 실습

### > 최적의 하이퍼 파라미터 튜닝

```
params = {'max_depth' : [ 8 , 12, 16 ,20],  
          'min_samples_split' : [16, 24],}
```

```
grid_cv = GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, verbose=1 )  
grid_cv.fit(X_train , y_train)  
print(f'GridSearchCV 최고 정확도 수치: {grid_cv.best_score_: .4f}')  
print('GridSearchCV 최적 하이퍼 파라미터:', grid_cv.best_params_)
```

**Out :** Fitting 5 folds for each of 8 candidates, totalling 40 fits  
GridSearchCV 최고 정확도 수치: 0.9667  
GridSearchCV 최적 하이퍼 파라미터: {'max\_depth': 8, 'min\_samples\_split': 16}

- 결정 트리 실습

> 최적의 하이퍼 파라미터 튜닝

```
best_df_clf = grid_cv.best_estimator_  
pred1 = best_df_clf.predict(X_test)  
accuracy = accuracy_score(y_test , pred1)  
print(f'결정 트리 예측 정확도:{accuracy:.4f}')
```

**Out :** 결정 트리 예측 정확도:0.9667

## • 데이터 전처리

### > 레이블 인코딩

```
from sklearn.preprocessing import LabelEncoder

items = ['TV', '냉장고', '전자레인지', '컴퓨터', '선풍기', '선풍기', '믹서', '냉장고']

encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
print('인코딩 변환값 :', labels)
```

**Out :** 인코딩 변환값 : [0 1 4 5 3 3 2 1]

- 데이터 전처리

- > 레이블 인코딩

```
print('인코딩 클래스 :', encoder.classes_)
```

**Out :** 인코딩 클래스 : ['TV' '냉장고' '믹서' '선풍기' '전자레인지' '컴퓨터']

```
print('디코딩 :', encoder.inverse_transform([4, 5, 2, 1]))
```

**Out :** 디코딩 : ['전자레인지' '컴퓨터' '믹서' '냉장고']



## • 데이터 전처리

### > 원-핫 인코딩

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

items = ['TV', '냉장고', '전자레인지', '컴퓨터', '선풍기', '선풍기', '믹서', '냉장고']

items = np.array(items).reshape(-1, 1)  # 2차원에서 사용됨

oh_encoder = OneHotEncoder()
oh_encoder.fit(items)
oh_labels = oh_encoder.transform(items).fit_transform()
```

oh\_encoder.categories\_

## • 데이터 전처리

### > 원-핫 인코딩

str

```
print('원-핫 인코딩 데이터')  
print(oh_labels.toarray())  
print('원-핫 인코딩 데이터 크기')  
print(oh_labels.shape)
```

Out :

```
원-핫 인코딩 데이터  
[[1.  0.  0.  0.  0.  0.]  
 [0.  1.  0.  0.  0.  0.]  
 [0.  0.  0.  0.  1.  0.]  
 [0.  0.  0.  0.  0.  1.]  
 [0.  0.  0.  1.  0.  0.]  
 [0.  0.  0.  1.  0.  0.]  
 [0.  0.  1.  0.  0.  0.]  
 [0.  1.  0.  0.  0.  0.]]  
원-핫 인코딩 데이터 크기  
(8, 6)
```


## • 데이터 전처리

### > 원-핫 인코딩

```
import pandas as pd
```

```
df = pd.DataFrame({'item':['TV', '냉장고', '전자레인지', '컴퓨터',  
                           '선풍기', '선풍기', '믹서', '냉장고']})
```

2차원



```
pd.get_dummies(df)
```

원 핫 인코딩 / 자동으로 해줌

Out :

	item_TV	item_냉장고	item_믹서	item_선풍기	item_전자레인지	item_컴퓨터
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	0	0	1	0
3	0	0	0	0	0	1
4	0	0	0	1	0	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0
7	0	1	0	0	0	0

## • 데이터 전처리

### > 피쳐 스케일링과 정규화

- 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업

### > 표준화

- 평균이 0, 분산이 1인 가우시안 정규 분포를 가진 값으로 변환

0에 가까운 수로 변환 0에 가까운 값으로 변경

$$x_{i\_new} = \frac{x_i - \text{mean}(x)}{\text{stdev}(x)}$$

### > 정규화

- 값을 범위를 모두 0 ~ 1의 값으로 변환

min max scaling

$$x_{i\_new} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

## • 데이터 전처리

### > StandardScaler

- 표준화

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

iris = load_iris()
iris_data = iris.data
iris_df = pd.DataFrame(data = iris_data, columns = iris.feature_names)

print('feature 들의 평균 값 :', iris_df.mean())
print('feature 들의 분산 값 :', iris_df.var())
```

## • 데이터 전처리

### > StandardScaler

- 표준화

**Out :**

```
feature 들의 평균 값 :
  sepal length (cm)    5.843333
  sepal width (cm)     3.057333
  petal length (cm)    3.758000
  petal width (cm)     1.199333
dtype: float64
feature 들의 분산 값 :
  sepal length (cm)    0.685694
  sepal width (cm)     0.189979
  petal length (cm)    3.116278
  petal width (cm)     0.581006
dtype: float64
```

## • 데이터 전처리

### > StandardScaler

- 표준화

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

iris_df_scaled = pd.DataFrame(data = iris_scaled, columns = iris.feature_names)

print('feature 들의 평균 값 :\\n', iris_df_scaled.mean())
print('feature 들의 분산 값 :\\n', iris_df_scaled.var())
```

## • 데이터 전처리

### > StandardScaler

- 표준화

**Out :**

```
feature 들의 평균 값 :
sepal length (cm)    -1.690315e-15
sepal width (cm)     -1.842970e-15
petal length (cm)    -1.698641e-15
petal width (cm)     -1.409243e-15
dtype: float64
feature 들의 분산 값 :
sepal length (cm)     1.006711
sepal width (cm)      1.006711
petal length (cm)     1.006711
petal width (cm)      1.006711
dtype: float64
```



## • 데이터 전처리

### > MinMaxScaler

- 정규화

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

표준화와 모두 같고 MinMaxScaler() 여기만 다름  
\*\* 무조건 0-1 사이 값임

```
scaler.fit(iris_df)
```

```
iris_scaled = scaler.transform(iris_df)
```

```
iris_df_scaled = pd.DataFrame(data = iris_scaled, columns = iris.feature_names)
```

```
print('feature 들의 최소 값 :\\n', iris_df_scaled.min())
```

```
print('feature 들의 최대 값 :\\n', iris_df_scaled.max())
```

## • 데이터 전처리

### > StandardScaler

- 표준화

**Out :**

```
feature 들의 최소 값 :
  sepal length (cm)    0.0
  sepal width (cm)     0.0
  petal length (cm)    0.0
  petal width (cm)     0.0
dtype: float64
feature 들의 최대 값 :
  sepal length (cm)    1.0
  sepal width (cm)     1.0
  petal length (cm)    1.0
  petal width (cm)     1.0
dtype: float64
```

## • 타이타닉 생존자 예측

051

### > 라이브러리 호출 및 데이터 불러오기

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

titanic_df = sns.load_dataset('titanic')
titanic_df.head(3)
```

Out :

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

## • 타이타닉 생존자 예측

052

### > 라이브러리 호출 및 데이터 불러오기

```
titanic_df.info()
```

**Out :**

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 12 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   PassengerId  891 non-null    int64  
1   Survived     891 non-null    int64  
2   Pclass       891 non-null    int64  
3   Name         891 non-null    object  
4   Sex          891 non-null    object  
5   Age          714 non-null    float64  
6   SibSp        891 non-null    int64  
7   Parch        891 non-null    int64  
8   Ticket       891 non-null    object  
9   Fare         891 non-null    float64  
10  Cabin        204 non-null    object  
11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)  
memory usage: 83.7+ KB
```

- 타이타닉 생존자 예측

- > 누락값 처리

```
titanic_df['Age'].fillna(titanic_df['Age'].mean(), inplace=True)
titanic_df['Cabin'].fillna('N', inplace=True)
titanic_df['Embarked'].fillna('N', inplace=True)
print('데이터 세트 Null 값 개수 :', titanic_df.isnull().sum().sum())
```

**Out :** 데이터 세트 Null 값 개수 : 0

## • 타이타닉 생존자 예측

### > 피쳐 살펴보기

```
print('성별 값 분포 :\n', titanic_df['Sex'].value_counts())  
print('선실 값 분포 :\n', titanic_df['Cabin'].value_counts())  
print(' 값 분포 :\n', titanic_df['Embarked'].value_counts())
```

**Out :**

```
성별 값 분포 :  
  male      577  
  female    314  
Name: Sex, dtype: int64  
선실 값 분포 :  
  N      687  
  C23 C25 C27      4  
  G6      4  
  B96 B98      4  
  C22 C26      3  
  ...  
  E34      1  
  C7      1  
  C54      1  
  E36      1  
  C148     1  
Name: Cabin, Length: 148, dtype: int64  
 값 분포 :  
  S      644  
  C      168  
  Q       77  
  N        2  
Name: Embarked, dtype: int64
```

## • 타이타닉 생존자 예측

055

### > 피쳐 살펴보기

```
titanic_df['Cabin'] = titanic_df['Cabin'].str[:1]  
print(titanic_df['Cabin'].head(3))
```

**Out :**

0	N
1	C
2	N

Name: Cabin, dtype: object

```
titanic_df.groupby(['Sex', 'Survived'])['Survived'].count()
```

**Out :**

	Sex	Survived
female	0	81
	1	233
male	0	468
	1	109

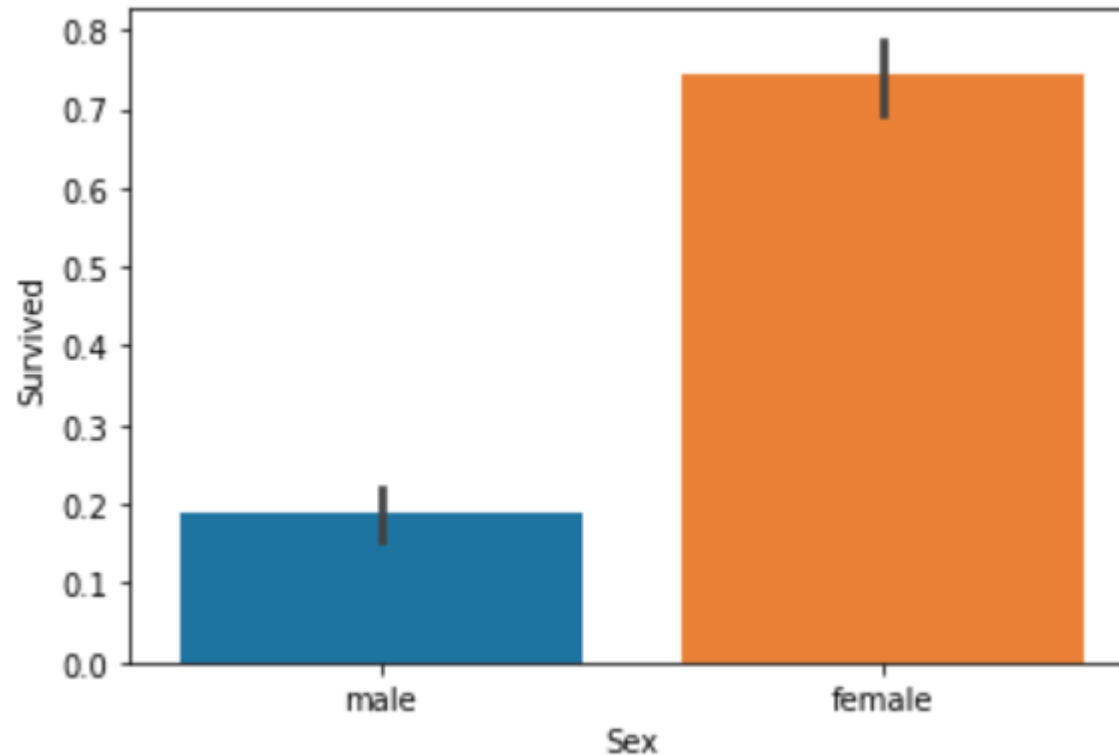
Name: Survived, dtype: int64

- 타이타닉 생존자 예측

> 피쳐 살펴보기

```
sns.barplot(x = 'Sex', y = 'Survived', data = titanic_df)
```

Out :



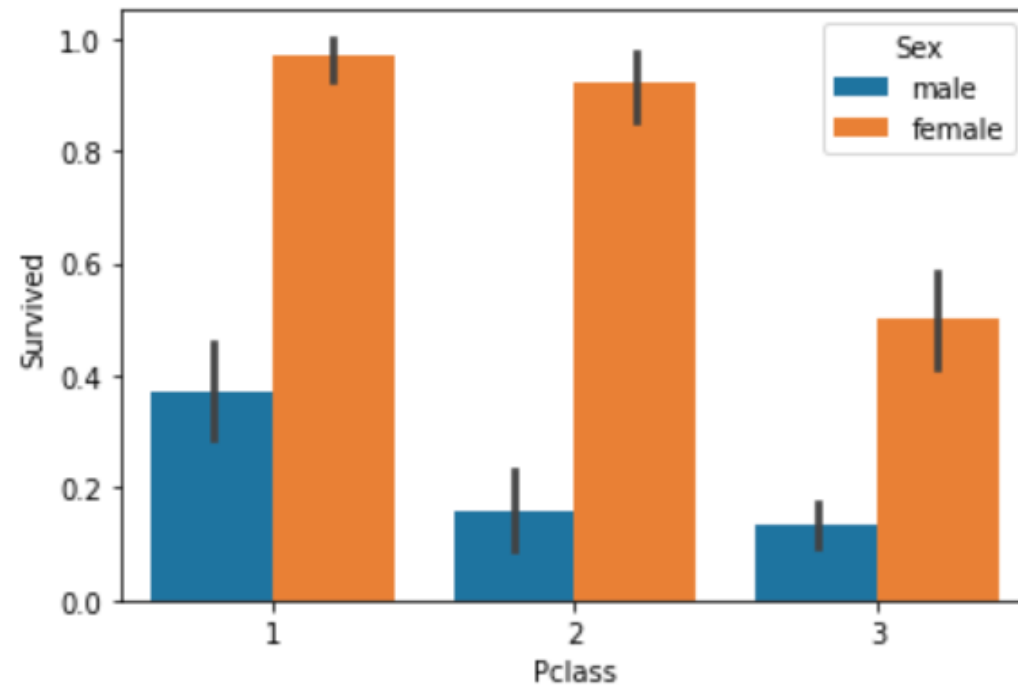


- 타이타닉 생존자 예측

> 피쳐 살펴보기

```
sns.barplot(x = 'Pclass', y = 'Survived', hue = 'Sex', data = titanic_df)
```

Out :



- 타이타닉 생존자 예측

- > 피쳐 살펴보기

```
def get_category(age):  
    cat = "  
    if age <= -1: cat = 'Unknown'  
    elif age <= 5: cat = 'Baby'  
    elif age <= 12: cat = 'Child'  
    elif age <= 18: cat = 'Teenager'  
    elif age <= 25: cat = 'Student'  
    elif age <= 35: cat = 'Young Adult'  
    elif age <= 60: cat = 'Adult'  
    else : cat = 'Elderly'  
    return cat
```

## • 타이타닉 생존자 예측

### > 피쳐 살펴보기

```
plt.figure(figsize=(10, 6))

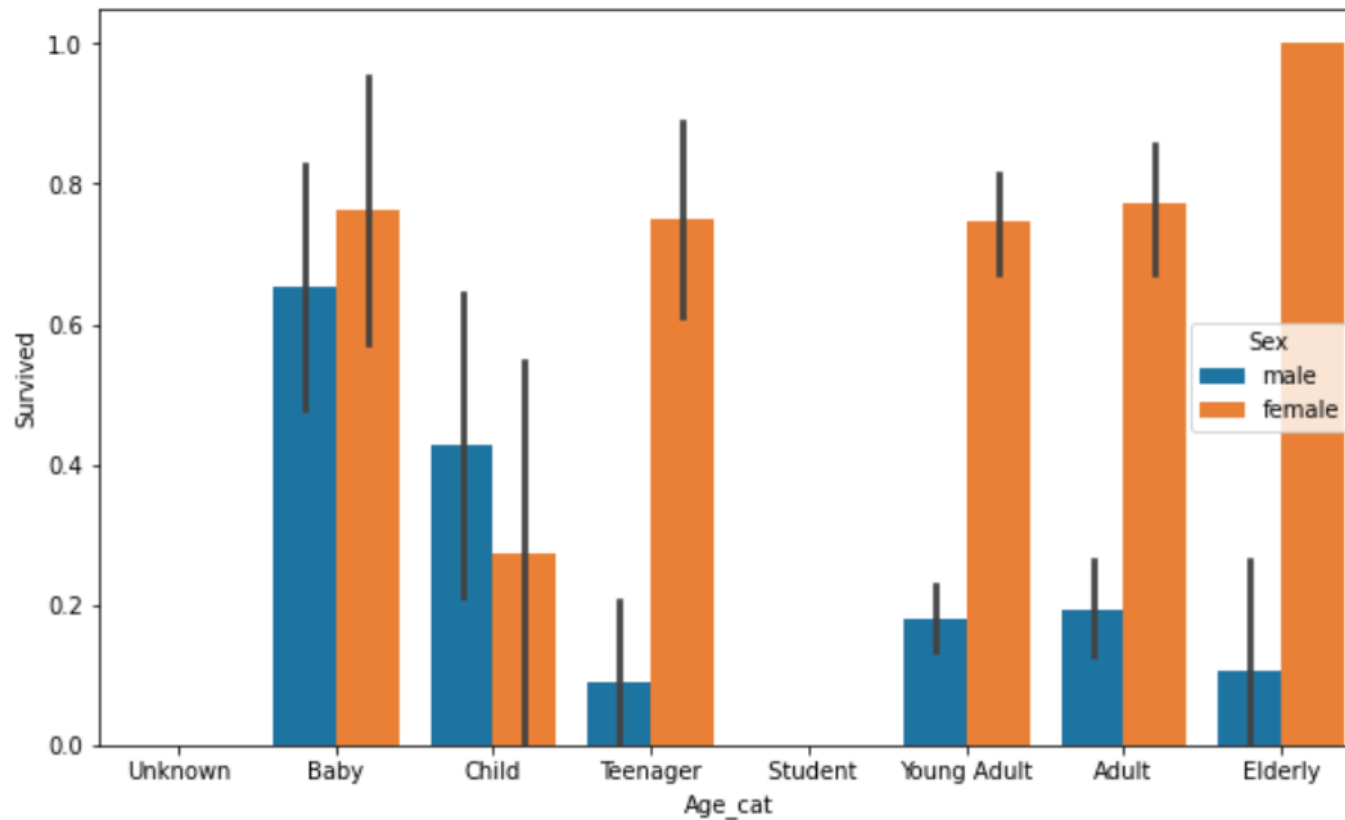
group_names = ['Unknown', 'Baby', 'Child', 'Teenager',
               'Student', 'Young Adult', 'Adult', 'Elderly']

titanic_df['Age_cat'] = titanic_df['Age'].apply(lambda x : 2 if x < 16 else 1)
sns.barplot(x = 'Age_cat', y = 'Survived', hue = 'Sex',
            data = titanic_df, order = group_names)
```

- 타이타닉 생존자 예측

> 피쳐 살펴보기

Out :



- 타이타닉 생존자 예측

- > 데이터 전처리

```
from sklearn.preprocessing import LabelEncoder

def encode_features(dataDF):
    features = ['Cabin', 'Sex', 'Embarked']
    for feature in features:
        le = LabelEncoder()
        dataDF[feature] = le.fit_transform(dataDF[feature])
    return dataDF

titanic_df = encode_features(titanic_df)
titanic_df.head()
```

• 타이타닉 생존자 예측

> 데이터 전처리

Out :

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_cat
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	7	3	4
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	2	0	0
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	7	3	6
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	2	3	6
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	7	3	6

- 타이타닉 생존자 예측

> 불필요한 컬럼 제거 & 독립변수/종속변수 나누기

```
def drop_features(df):  
    df.drop(['PassengerId', 'Name', 'Ticket'], axis = 1, inplace=True)  
    return df  
  
y_titanic_df = titanic_df['Survived']  
X_titanic_df = titanic_df.drop('Survived', axis = 1)  
  
X_titanic_df = drop_features(X_titanic_df)  
X_titanic_df.head()
```

- 타이타닉 생존자 예측

> 학습, 테스트 데이터 분배

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_titanic_df, y_titanic_df,
                                                    test_size = 0.2, random_state=11)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

**Out :** (712, 9) (179, 9) (712,) (179,)



- 타이타닉 생존자 예측

- > 다양한 머신러닝 알고리즘 임포트

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

dt_clf = DecisionTreeClassifier(random_state = 11)
rf_clf = RandomForestClassifier(random_state = 11)
lr_clf = LogisticRegression(solver='liblinear')
```

- 타이타닉 생존자 예측

- > 학습/예측/평가

```
# DecisionTreeClassifier 학습/예측/평가
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
acc = accuracy_score(y_test, dt_pred)
print(f'DecisionTree 정확도 : {acc:.4f}' )
```

```
# RandomForestClassifier 학습/예측/평가
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
acc = accuracy_score(y_test, rf_pred)
print(f'RandomForest 정확도 : {acc:.4f}' )
```

- 타이타닉 생존자 예측

- > 학습/예측/평가

```
# LogisticRegression 학습/예측/평가
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
acc = accuracy_score(y_test, lr_pred)
print(f'Logistic 정확도 : {acc:.4f}' )
```

**Out :** DesitionTree 정확도 : 0.8045  
RandomForest 정확도 : 0.8659  
Logistic 정확도 : 0.8659

- 타이타닉 생존자 예측

> 각 알고리즘에 cross\_val\_score() 검증 실시

```
from sklearn.model_selection import cross_val_score

def exec_cvs(clf, folds = 5):
    scores = cross_val_score(clf, X_titanic_df, y_titanic_df, cv = folds)
    for iter_count, accuracy in enumerate(scores):
        print(f"교차 검증 {iter_count} 정확도 : {accuracy:.4f}")
    print(f'평균 정확도 : {np.mean(scores):.4f}')
```

## • 타이타닉 생존자 예측

069

> 각 알고리즘에 cross\_val\_score() 검증 실시

```
df_clf = DecisionTreeClassifier(random_state = 11)
exec_kfold(dt_clf)
rf_clf = RandomForestClassifier(random_state = 11)
exec_kfold(rf_clf)
lr_clf = LogisticRegression(solver='liblinear')
exec_kfold(lr_clf)
```

**Out :**

교차 검증 0 정확도 : 0.7318	교차 검증 0 정확도 : 0.7821	교차 검증 0 정확도 : 0.7877
교차 검증 1 정확도 : 0.7697	교차 검증 1 정확도 : 0.8146	교차 검증 1 정확도 : 0.7921
교차 검증 2 정확도 : 0.7921	교차 검증 2 정확도 : 0.8371	교차 검증 2 정확도 : 0.7697
교차 검증 3 정확도 : 0.7697	교차 검증 3 정확도 : 0.7640	교차 검증 3 정확도 : 0.7528
교차 검증 4 정확도 : 0.8146	교차 검증 4 정확도 : 0.8708	교차 검증 4 정확도 : 0.8427
평균 정확도 : 0.7756	평균 정확도 : 0.8137	평균 정확도 : 0.7890

- 타이타닉 생존자 예측

> 각 알고리즘에 KFold 검증 실시

```
from sklearn.model_selection import KFold
# 교차검증 함수 생성
def exec_kfold(clf, X_train, y_train, folds = 5):
    kfold = KFold(n_splits = 5)
    scores = []

    for iter_count, (train_index, test_index) in enumerate(kfold.split(X_train)):
        X_train_f, X_val_f = X_train.iloc[train_index], X_train.iloc[test_index]
        y_train_f, y_val_f = y_train.iloc[train_index], y_train.iloc[test_index]
        clf.fit(X_train_f, y_train_f)
```

- 타이타닉 생존자 예측

- > 각 알고리즘에 KFold 검증 실시

```
# 이전 코드에 이어서 작성
predictions = clf.predict(X_val_f)
accuracy = accuracy_score(y_val_f, predictions)
scores.append(accuracy)
print(f"교차 검증 {iter_count} 정확도 : {accuracy:.4f}")

mean_score = np.mean(scores)
print(f'평균 정확도 : {mean_score:.4f}')
return clf
```

## • 타이타닉 생존자 예측

### > 각 알고리즘에 KFold 검증 실시

```
df_clf = DecisionTreeClassifier(random_state = 11)
model = exec_kfold(dt_clf, X_train, y_train)
pred = model.predict(X_test)
```

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

**Out :** 교차 검증 0 정확도 : 0.7203  
교차 검증 1 정확도 : 0.7273  
교차 검증 2 정확도 : 0.7465  
교차 검증 3 정확도 : 0.8028  
교차 검증 4 정확도 : 0.7746  
평균 정확도 : 0.7543  
예측 정확도: 0.7709



## • 타이타닉 생존자 예측

> 각 알고리즘에 KFold 검증 실시

```
rf_clf = RandomForestClassifier(random_state = 11)
model = exec_kfold(rf_clf, X_train, y_train)
pred = model.predict(X_test)

acc = accuracy_score(y_test, pred)
print('예측 정확도: {0:.4f}'.format(acc))
```

**Out :**

교차 검증 0 정확도 :	0.7552
교차 검증 1 정확도 :	0.7902
교차 검증 2 정확도 :	0.7958
교차 검증 3 정확도 :	0.8380
교차 검증 4 정확도 :	0.7676
평균 정확도 :	0.7894
예측 정확도:	0.8659

## • 타이타닉 생존자 예측

> 각 알고리즘에 KFold 검증 실시

```
lr_clf = LogisticRegression(solver='liblinear')  
model = exec_kfold(lr_clf, X_train, y_train)  
pred = model.predict(X_test)
```

```
acc = accuracy_score(y_test, pred)  
print('예측 정확도: {0:.4f}'.format(acc))
```

**Out :**

```
교차 검증 0 정확도 : 0.7762  
교차 검증 1 정확도 : 0.7902  
교차 검증 2 정확도 : 0.7746  
교차 검증 3 정확도 : 0.8099  
교차 검증 4 정확도 : 0.7676  
평균 정확도 : 0.7837  
예측 정확도: 0.8603
```