Alchemist Session

CHAP 02 사이킷런으로 시작하는 머신러닝 – 실습 코드

CONTENTS

사이킷런으로 수행하는 타이타닉 생존자 예측

1. 데이터 탐색과 전처리
2. 여러 ML 알고리즘으로 학습/예측/평가
3. 교차 검증
4. 최적 하이퍼 파라미터 찾기

01.

데이터 탐색과 전처리

데이터 칼럼 살펴보기

titanic_train.csv

- passengerid: 탑승자 데이터 일련번호
- survived: 생존 여부 / 0=사망, 1=생존
- pclass: 티켓의 선실 등급 / 1=일등석, 2=이등석, 3=삼등석
- sex: 탑승자 성별
- name: 탑승자 이름
- age: 탑승자 나이
- sibsp: 같이 탑승한 형제자매 또는 배우자 인원수
- parch: 같이 탑승한 부모님 또는 어린이 인원수
- ticket: 티켓 번호
- fare: 요금
- cabin: 선실 번호
- embarked: 중간 정착 항구

  / C=Cherbourg, Q=Queenstown, S=Southampton

데이터 로딩 및 전체 칼럼 보기

- **read_csv(), head**

```
# 필요 라이브러리 임포트
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# 파일을 DataFrame으로 로딩 (read_csv())
titanic_df = pd.read_csv('./titanic_train.csv')

# 앞의 3행을 살펴보기 (head)
titanic_df.head(3)
```

	PassengerId	Survived	Pclass	Name
0	1	0	3	Braund, Mr. Owen Harris
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...
2	3	1	3	Heikkinen, Miss. Laina

	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	female	38.0	1	0	PC 17599	71.2833	C85	C
2	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

데이터 칼럼 타입 살펴보기

- **info()**

```
# 데이터 칼럼 타입 살펴보기 (info)
print('\n ### 학습 데이터 정보 ### \n')
print(titanic_df.info())
```

전체 로우(행): 891개

전체 칼럼(열): 12개

판다스의 object는 string이라 봐도 무방

Age, Cabin, Embarked는 Null값 존재

→ 사이킷럼 ML 알고리즘은 Null값 허용 X

→ Null 처리 필요

```
 ### 학습 데이터 정보 ###

<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
None
```

Null 값 처리

- **fillna()**

```
# Null 값 처리 ( fillna(바꿀 값, inplace=원본DF에 반영 여부) )
titanic_df['Age'].fillna(titanic_df['Age'].mean(), inplace=True)
titanic_df['Cabin'].fillna('N', inplace=True)
titanic_df['Embarked'].fillna('N', inplace=True)

# 처리 후 Null 개수 확인 (isnull().sum().sum())
print('데이터 세트 Null 값 개수\n', titanic_df.isnull().sum())
print('\n전체 데이터 세트 Null 값 개수', titanic_df.isnull().sum().sum())
```

```
데이터 세트 Null 값 개수
 PassengerId    0
Survived        0
Pclass          0
Name            0
Sex             0
Age             0
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin           0
Embarked        0
dtype: int64

전체 데이터 세트 Null 값 개수 0
```

다른 문자열 피처 살펴보기

- **value_counts()**

```
# 다른 문자열 피처 살펴보기 (value_counts)
print(' Sex 값 분포 :\n', titanic_df['Sex'].value_counts())
print('\n Cabin 값 분포 :\n', titanic_df['Cabin'].value_counts())
print('\n Embarked 값 분포 :\n', titanic_df['Embarked'].value_counts())
```

```
 Sex 값 분포 :
 male      577
female     314
Name: Sex, dtype: int64

 Cabin 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

 Embarked 값 분포 :
 S    644
C    168
Q     77
N      2
Name: Embarked, dtype: int64
```

Cabin 속성값이 제대로 정리되지 않음

→ 선실 등급을 나타내는 첫 알파벳을 추출!

Cabin 칼럼 전처리

```
# Cabin 속성의 앞 문자로 대체
titanic_df['Cabin'] = titanic_df['Cabin'].str[:1] # 슬라이싱, 0~1전까지 = 0
print(titanic_df['Cabin'].head(3))
```

```
0    N
1    C
2    N
Name: Cabin, dtype: object
```

각 피처의 유형별 생존자 확인

- **성별에 따른 생존자 수**

```
# 성별에 따른 생존자 수
titanic_df.groupby(['Sex', 'Survived'])['Survived'].count()
```

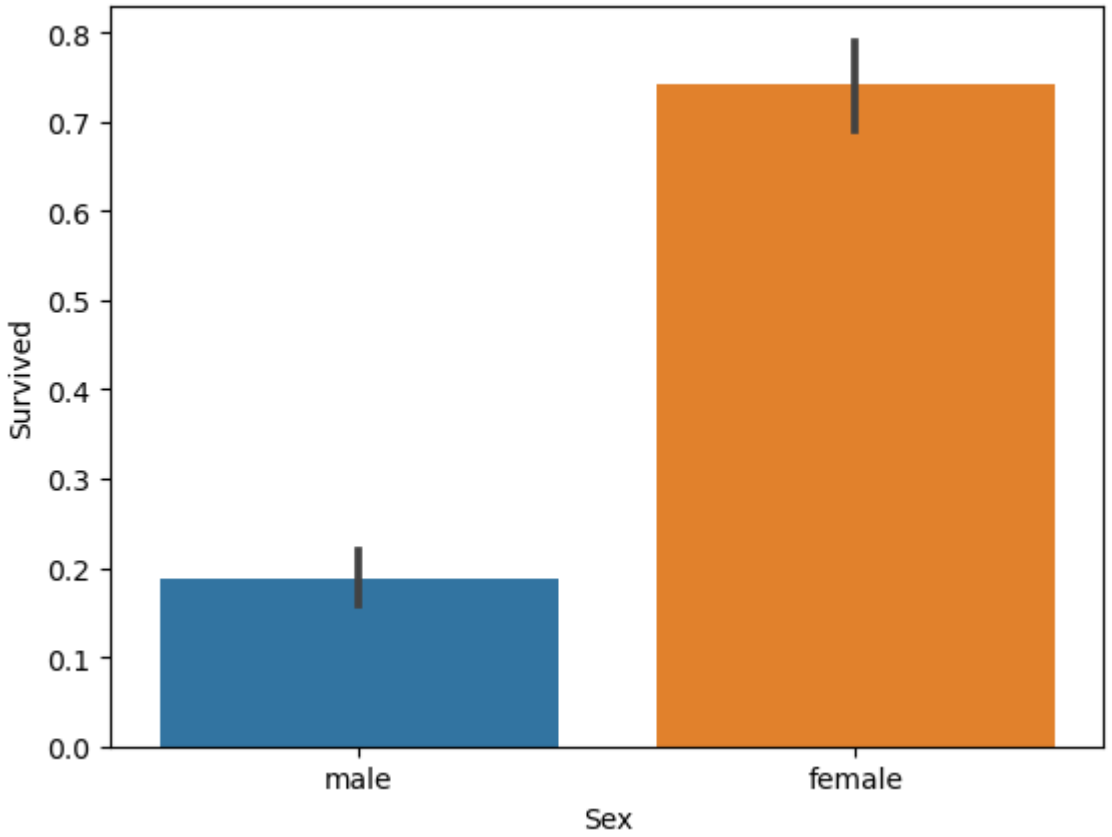
```
Sex     Survived
female  0            81
        1           233
male    0           468
        1           109
Name: Survived, dtype: int64
```

Sex와 Survived 두 열을 기준으로 데이터를 그룹화하고,

Survived 열에 있는 값의 수 카운트

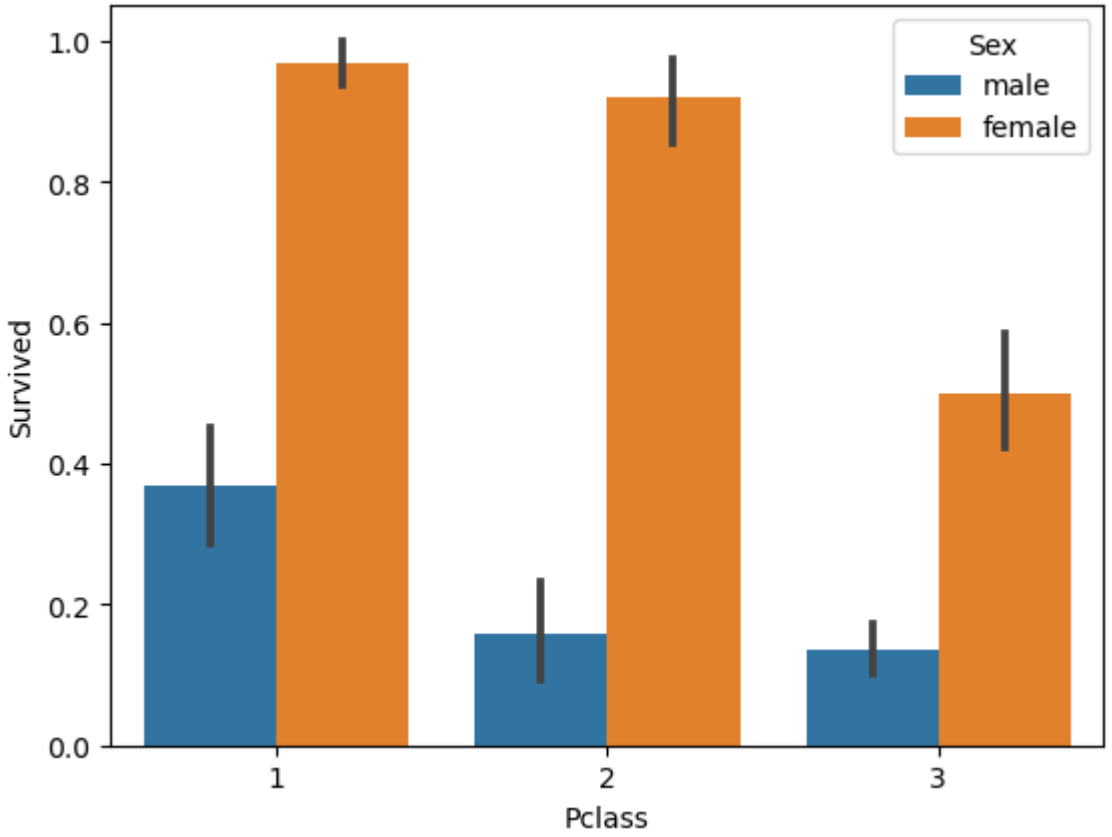
성별에 따른 생존 확률

```
# 시본 패키지를 이용한 시각화
sns.barplot(x='Sex', y='Survived', data=titanic_df) # x축, y축 칼럼 지정
```



부에 따른 생존 확률

```
# 부에 따른 생존 확률
sns.barplot(x='Pclass', y='Survived', hue='Sex', data=titanic_df)
```



hue:

- 그래프에서 추가적인 범주형 변수에 따라 데이터를 색상으로 구분
- 한 그래프에서 여러 범주를 시각적으로 비교할 수 있도록 돕는 역할

나이에 따른 생존 확률

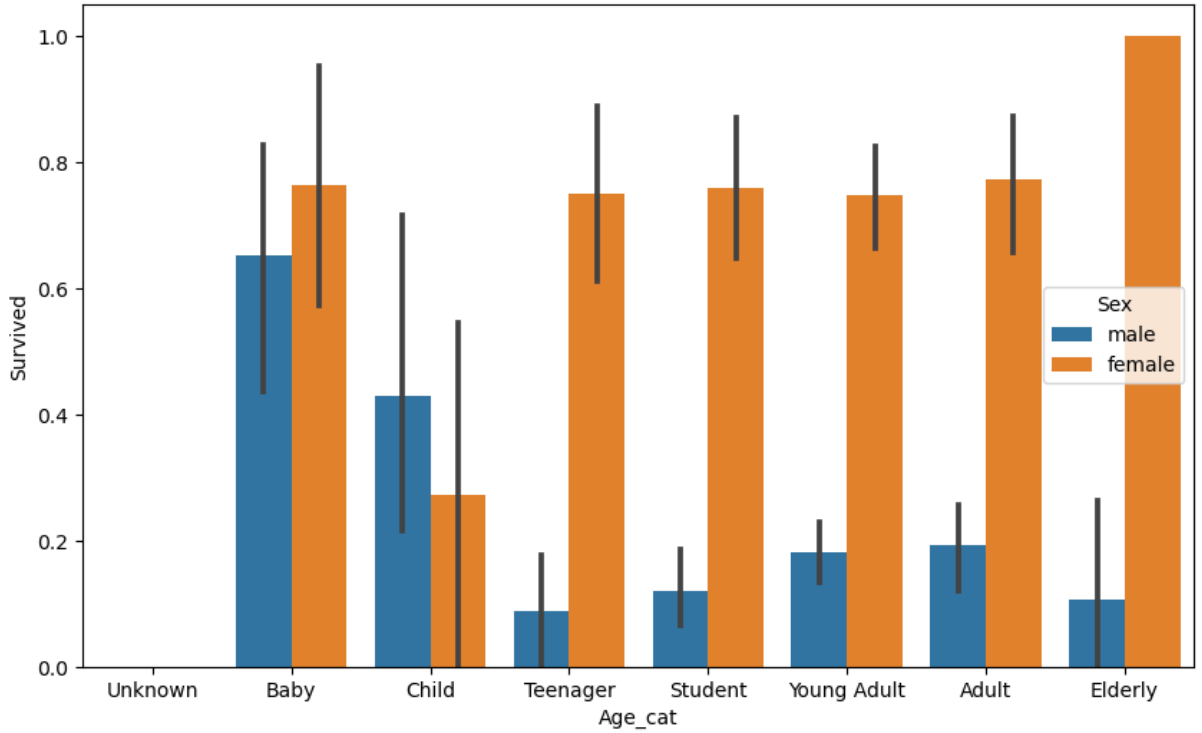
```
# 나이에 따른 생존 확률

# 입력 age에 따라 구분 값을 반환하는 함수 설정. DataFrame의 apply lambda 사용
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
# 막대그래프의 크기 figure를 더 크게 설정
plt.figure(figsize=(10, 6))

# X축의 값을 순차적으로 표시하기 위한 설정
group_names = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', 'Young Adult', 'Adult', 'Elderly']

# lamda 식에 위에서 생성한 get_category() 함수를 반환값으로 지정
# get_category(X)는 입력값으로 'Age'칼럼 값을 받아 해당하는 cat 반환
titanic_df['Age_cat'] = titanic_df['Age'].apply(lambda x: get_category(x))
sns.barplot(x='Age_cat', y='Survived', hue='Sex', data=titanic_df, order=group_names)
titanic_df.drop('Age_cat', axis=1, inplace=True) # 다시 Age_cat 삭제
```



문자열 피처 인코딩

- **LabelEncoder**

```
# 문자열 피처 인코딩
from sklearn.preprocessing import LabelEncoder

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

    return dataDF

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

Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	22.0	1	0	A/5 21171	7.2500	7	3
0	38.0	1	0	PC 17599	71.2833	2	0
0	26.0	0	0	STON/O2. 3101282	7.9250	7	3
0	35.0	1	0	113803	53.1000	2	3
1	35.0	0	0	373450	8.0500	7	3

전처리 함수 제작

```
# 전처리 함수 제작

# Null 처리 함수
def fillna(df):
    df['Age'].fillna(df['Age'].mean(), inplace=True)
    df['Cabin'].fillna('N', inplace=True)
    df['Embarked'].fillna('N', inplace=True)
    df['Fare'].fillna(0, inplace=True)
    return df

# 머신러닝 알고리즘에 불필요한 피처 제거
def drop_features(df):
    df.drop(['PassengerId', 'Name', 'Ticket'], axis=1, inplace=True)
    return df
```

```
# 레이블 인코딩 수행
def format_features(df):
    df['Cabin'] = df['Cabin'].str[:1]
    features = ['Cabin', 'Sex', 'Embarked']
    for feature in features:
        le = LabelEncoder()
        le = le.fit(df[feature])
        df[feature] = le.transform(df[feature])
    return df

# 앞에서 설정한 데이터 전처리 함수 호출
def transform_features(df):
    df = fillna(df)
    df = drop_features(df)
    df = format_features(df)
    return df
```

02.

여러 ML 알고리즘으로 학습/예측/평가

데이터 재로딩, 데이터 세트 분리

```
# 원본 데이터 재로딩, 피처/레이블 데이터 세트 분리
titanic_df = pd.read_csv('./titanic_train.csv')
y_titanic_df = titanic_df['Survived']
X_titanic_df = titanic_df.drop('Survived', axis=1)

# 전처리
X_titanic_df = transform_features(X_titanic_df)
```

```
# 학습/테스트 데이터 세트 분리
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)
```

학습/예측/평가

```python
# 여러 ML 알고리즘으로 학습/예측/평가
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 결정 트리, 랜덤 포레스트, 로지스틱 회귀를 위한 사이킷런 Classifier 클래스 생성
dt_clf = DecisionTreeClassifier(random_state=11)
rf_clf = RandomForestClassifier(random_state=11)
lr_clf = LogisticRegression(solver='liblinear')

# DecisionTreeClassifier 학습/예측/평가
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
print('DecisionTreeClassifier 정확도: {0:.4f}'.format(accuracy_score(y_test, dt_pred)))

# RandomForestClassifier 학습/예측/평가
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
print('RandomForestClassifier 정확도: {0:.4f}'.format(accuracy_score(y_test, rf_pred)))

# LogisticRegressionClassifier 학습/예측/평가
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
print('LogisticRegressionClassifier 정확도: {0:.4f}'.format(accuracy_score(y_test, lr_pred)))
```

```
DecisionTreeClassifier 정확도: 0.7877
RandomForestClassifier 정확도: 0.8547
LogisticRegressionClassifier 정확도: 0.8659
```

03.

교차 검증

KFold 교차 검증

```
# 교차 검증으로 결정 트리 모델 평가
# KFold 교차 검증
from sklearn.model_selection import KFold

def exec_kfold(clf, folds=5):
    # 폴드 세트가 5개인 KFold 객체를 생성, 폴드 수만큼 예측결과 저장을 위한 리스트 객체 생성
    kfold = KFold(n_splits=folds)
    scores = []

    # KFold 교차 검증 수행
    for iter_count, (train_index, test_index) in enumerate(kfold.split(X_titanic_df)):
        # X_titanic_df 데이터에서 교차 검증별로 학습과 검증 데이터를 가리키는 index 생성
        X_train, X_test = X_titanic_df.values[train_index], X_titanic_df.values[test_index]
        y_train, y_test = y_titanic_df.values[train_index], y_titanic_df.values[test_index]

        # Classifier 학습/예측/평가(정확도)
        clf.fit(X_train, y_train)
        predictions = clf.predict(X_test)
        accuracy = accuracy_score(y_test, predictions)
        scores.append(accuracy)
        print("교차 검증 {0} 정확도: {1:.4f}".format(iter_count, accuracy))

    # 5개의 fold에서의 평균 정확도 계산
    mean_score = np.mean(scores)
    print("평균 정확도: {0:.4f}".format(mean_score))
```

```
# exec_kfold 호출
exec_kfold(dt_clf, folds=5)
```

```
교차 검증 0 정확도: 0.7542
교차 검증 1 정확도: 0.7809
교차 검증 2 정확도: 0.7865
교차 검증 3 정확도: 0.7697
교차 검증 4 정확도: 0.8202
평균 정확도: 0.7823
```

cross_val_score()

```python
# cross_val_score() API로 교차 검증
from sklearn.model_selection import cross_val_score

scores = cross_val_score(dt_clf, X_titanic_df, y_titanic_df, cv=5)
for iter_count, accuracy in enumerate(scores):
    print("교차 검증 {0} 정확도: {1:.4f}".format(iter_count, accuracy))

print("평균 정확도: {0:.4f}".format(np.mean(scores)))
```

```
교차 검증 0 정확도: 0.7430
교차 검증 1 정확도: 0.7753
교차 검증 2 정확도: 0.7921
교차 검증 3 정확도: 0.7865
교차 검증 4 정확도: 0.8427
평균 정확도: 0.7879
```

KFold와 정확도 다른 이유?

StratifiedKFold 이용해 폴드 세트 분할하기 때문

→ 각 폴드에 있는 클래스의 비율이 원본 데이터의 클래스 비율과 동일하게 유지되도록 폴드 분할!!

클래스 불균형이 있는 데이터셋에서 유용!

04.

최적 하이퍼 파라미터 찾기

GridSearchCV

```
# GridSearchCV로 DecisionTreeClassifier 최적 하이퍼 파라미터 찾기
from sklearn.model_selection import GridSearchCV

parameters = {'max_depth': [2, 3, 5, 10], 'min_samples_split': [2, 3, 5], 'min_samples_leaf': [1, 5, 8]}

grid_dclf = GridSearchCV(dt_clf, param_grid=parameters, scoring='accuracy', cv=5)
grid_dclf.fit(X_train, y_train)

print('GridSearchCV 최적 하이퍼 파라미터:', grid_dclf.best_params_)
print('GridSearchCV 최고 정확도: {0:.4f}'.format(grid_dclf.best_score_))
best_dclf = grid_dclf.best_estimator_

# GridSearchCV의 최적 하이퍼 파라미터로 학습된 Estimator로 예측 및 평가 수행
dpredictions = best_dclf.predict(X_test)
accuracy = accuracy_score(y_test, dpredictions)
print('테스트 세트에서의 DecisionTreeClassifier 정확도: {0:.4f}'.format(accuracy))
```

```
GridSearchCV 최적 하이퍼 파라미터: {'max_depth': 3, 'min_samples_leaf': 5, 'min_samples_split': 2}
GridSearchCV 최고 정확도: 0.7992
테스트 세트에서의 DecisionTreeClassifier 정확도: 0.8715
```

정리

사이킷런으로 수행하는 타이타닉 생존자 예측

1. 데이터 탐색하며 전처리 결정 (중요~~)
2. 전처리 함수 제작
3. 피처 / 레이블 데이터 세트 분리
4. 학습 / 테스트 데이터 세트 분리
5. 여러 ML 알고리즘으로 학습/예측/평가
6. 교차 검증
7. 최적 하이퍼 파라미터 찾고, 그 파라미터로 학습된 Estimator로 예측/평가

수고하셨습니다