# 제2유형_연습하기_iris 종 분류

## ✅ 데이터 분석 순서



1. 라이브러리 및 데이터 확인

2. 데이터 탐색(EDA)

3. 데이터 전처리 및 분리

4. 모델링 및 성능평가

5. 예측값 제출

## ✅ 1. 라이브러리 및 데이터 확인

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
###############  실기환경 복사 영역  ###############
import pandas as pd
import numpy as np
# 실기 시험 데이터셋으로 셋팅하기 (수정금지)
from sklearn.datasets import load_iris
# Iris 데이터셋을 로드
iris = load_iris()
x = pd.DataFrame(iris.data, columns=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'])
y = iris.target   # 'setosa'=0, 'versicolor'=1, 'virginica'=2
y = np.where(y>0, 1, 0) # setosa 종은 0, 나머지 종은 1로 변경

# 실기 시험 데이터셋으로 셋팅하기 (수정금지)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
                                                    stratify=y,
                                                    random_state=2023)

x_test = pd.DataFrame(x_test)
x_train = pd.DataFrame(x_train)
y_train = pd.DataFrame(y_train)
y_train.columns = ['species']

# 결측치 삽입
x_test['sepal_length'].iloc[0] = None
x_train['sepal_length'].iloc[0] = None
# 이상치 삽입
x_train['sepal_width'].iloc[0] = 150
###############  실기환경 복사 영역  ###############


### 참고사항 ###
# y_test 는 실기 문제상에 주어지지 않음

# ★Tip : X를 대문자로 쓰지말고 소문자 x로 쓰세요. 시험에서 실수하기 쉽습니다.(문제풀기 전에 소문자로 변경!)
# (참고 : 보통 X는 2차원 배열(행렬)이기 때문에 대문자로 쓰고, y는 1차원 배열(벡터)이기 때문에 소문자로 씀)

# (~23년 10월말) 실기시험 데이터 형식 (실제 시험장에서는 다를 수 있으니 반드시 체크)
# X_test = pd.read_csv("data/X_test.csv")
# X_train = pd.read_csv("data/X_train.csv")
# y_train = pd.read_csv("data/y_train.csv")

# ★(23년 10월말~) 기준으로 체험환경에서 제공되는 데이터셋이 조금 변경되었습니다.
# train = pd.read_csv("data/customer_train.csv")
# test = pd.read_csv("data/customer_test.csv")
# x_train과 y_train, x_test를 별도로 할당해주셔야 합니다.
```

붓꽃의 종(Species)을 분류해보자

- 데이터의 결측치, 이상치에 대해 처리하고



- 분류모델을 사용하여 정확도, F1 score, AUC 값을 산출하시오.

- 제출은 result 변수에 담아 양식에 맞게 제출하시오



In [3]:
```
# 데이터 설명
```

```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

## ✓ 2. 데이터 탐색(EDA)

In [4]:
```
# 데이터의 행/열 확인

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
```

```
(120, 4)
(30, 4)
(120, 1)
```

In [5]:
```
# 초기 데이터 확인

print(x_train.head(3))
print(x_test.head(3))
print(y_train.head(3))
```

```
    sepal_length  sepal_width  petal_length  petal_width
2            NaN        150.0           1.3          0.2
49           5.0          3.3           1.4          0.2
66           5.6          3.0           4.5          1.5
     sepal_length  sepal_width  petal_length  petal_width
93            NaN          2.3           3.3          1.0
69            5.6          2.5           3.9          1.1
137           6.4          3.1           5.5          1.8
   species
0        0
1        0
2        1
```

In [6]:
```
# 변수명과 데이터 타입이 매칭이 되는지, 결측치가 있는지 확인해보세요

print(x_train.info())
print(x_test.info())
print(y_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 120 entries, 2 to 44
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  119 non-null    float64
 1   sepal_width   120 non-null    float64
 2   petal_length  120 non-null    float64
 3   petal_width   120 non-null    float64
dtypes: float64(4)
memory usage: 4.7 KB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30 entries, 93 to 55
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  29 non-null     float64
 1   sepal_width   30 non-null     float64
 2   petal_length  30 non-null     float64
 3   petal_width   30 non-null     float64
dtypes: float64(4)
memory usage: 1.2 KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   species  120 non-null    int32
dtypes: int32(1)
memory usage: 612.0 bytes
None
```

In [7]:
```
# x_train 과 x_test 데이터의 기초통계량을 잘 비교해보세요.

print(x_train.describe())  # x_train.describe().T 둘중에 편한거 사용하세요
print(x_test.describe())
print(y_train.describe())
```

```
       sepal_length  sepal_width  petal_length  petal_width
count    119.000000     120.0000    120.000000   120.000000
mean       5.920168       4.2950      3.816667     1.226667
std        0.841667      13.4191      1.798848     0.780512
min        4.300000       2.2000      1.100000     0.100000
25%        5.150000       2.8000      1.575000     0.300000
50%        6.000000       3.0000      4.400000     1.350000
75%        6.500000       3.4000      5.225000     1.800000
max        7.900000     150.0000      6.900000     2.500000
       sepal_length  sepal_width  petal_length  petal_width
count     29.000000    30.000000     30.000000     30.00000
mean       5.596552     3.000000      3.523333      1.09000
std        0.709367     0.522593      1.631518      0.68549
min        4.600000     2.000000      1.000000      0.10000
25%        5.000000     2.625000      1.600000      0.35000
50%        5.500000     3.000000      4.050000      1.15000
75%        5.900000     3.300000      4.925000      1.57500
max        7.600000     4.200000      6.600000      2.30000
         species
count  120.000000
mean     0.666667
std      0.473381
min      0.000000
25%      0.000000
50%      1.000000
75%      1.000000
max      1.000000
```

In [8]:
```
# y데이터도 구체적으로 살펴보세요.
print(y_train.head())
```

```
   species
0        0
1        0
2        1
3        1
4        1
```

In [9]:
```
# y데이터도 구체적으로 살펴보세요.
print(y_train.value_counts())
```

```
species
1          80
0          40
dtype: int64
```

## ✅ 3. 데이터 전처리 및 분리

### 1) 결측치, 2) 이상치, 3) 변수 처리하기

In [10]:
```
# 결측치 확인
print(x_train.isnull().sum())
print(x_test.isnull().sum())
print(y_train.isnull().sum())
```

```
sepal_length    1
sepal_width     0
petal_length    0
petal_width     0
dtype: int64
sepal_length    1
sepal_width     0
petal_length    0
petal_width     0
dtype: int64
species    0
dtype: int64
```

In [11]:
```
# 결측치 제거
# df = df.dropna()
# print(df)

# 참고사항
# print(df.dropna().shape)  # 행 기준으로 삭제

# ★주의사항
# x_train의 행을 제거해야 하는 경우, 그에 해당하는 y_train 행도 제거해야 합니다.
# 해결방법 : train = pd.concat([x_train, y_train], axis=1)
# 위와 같이 데이터를 결합한 후에 행을 제거하고 다시 데이터 분리를 수행하면 됩니다.
# (만약 원데이터가 x_train/y_train이 결합된 형태로 주어진다면 전처리를 모두 수행한 후에 분리하셔도 됩니다)
```

In [12]:
```
# 결측치 대체(평균값, 중앙값, 최빈값)
# ** 주의사항 : train 데이터의 중앙값/평균값/최빈값 등으로 test 데이터의 결측치도 변경해줘야 함 **

# 연속형 변수 : 중앙값, 평균값
#  - df['변수명'].median()
#  - df['변수명'].mean()
# 범주형 변수 : 최빈값

# df['변수명'] = df['변수명'].fillna(대체할 값)
```

In [13]:
```
# 결측치 대체(중앙값)
# ** 주의사항 : train 데이터의 중앙값으로 test 데이터도 변경해줘야 함 **

median = x_train['sepal_length'].median()
x_train['sepal_length'] = x_train['sepal_length'].fillna(median)
x_test['sepal_length'] = x_test['sepal_length'].fillna(median)
```

In [14]:
```
# 이상치 확인
cond1 = (x_train['sepal_width']>=10)
print(len(x_train[cond1]))
```

```
1
```

In [15]:
```
# 이상치 대체
# (참고) df['변수명'] = np.where( df['변수명'] >= 5, 대체할 값, df['변수명'] )

# 예를 들어 'sepal_width' 값이 10이 넘으면 이상치라고 가정해본다면
# 이상치를 제외한 Max 값을 구해서 대체해보자
cond1 = (x_train['sepal_width'] <= 10)
max_sw = x_train[cond1]['sepal_width'].max()
print(max_sw)

x_train['sepal_width'] = np.where( x_train['sepal_width'] >= 10, max_sw, x_train['sepal_width'] )
print(x_train.describe())
```

```
4.4
       sepal_length  sepal_width  petal_length  petal_width
count    120.000000   120.000000    120.000000   120.000000
mean       5.920833     3.081667      3.816667     1.226667
std        0.838155     0.429966      1.798848     0.780512
min        4.300000     2.200000      1.100000     0.100000
25%        5.175000     2.800000      1.575000     0.300000
50%        6.000000     3.000000      4.400000     1.350000
75%        6.500000     3.400000      5.225000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```



In [16]:
```python
# 변수처리

# 불필요한 변수 제거
# df = df.drop(columns = ['변수1','변수2'])
# df = df.drop(['변수1','변수2'], axis=1)

# 필요시 변수 추가(파생변수 생성)
# df['파생변수명'] = df['A'] * df['B'] (파생변수 생성 수식)

# 원핫인코딩(가변수 처리)
# x_train = pd.get_dummies(x_train)
# x_test = pd.get_dummies(x_test)
# print(x_train.info())
# print(x_test.info())
```

## 데이터 분리

In [17]:
```python
# 데이터를 훈련 세트와 검증용 세트로 분할 (80% 훈련, 20% 검증용)
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train['species'], test_size=0.2, random_state=2023

print(x_train.shape)
print(x_val.shape)
print(y_train.shape)
print(y_val.shape)
```

```
(96, 4)
(24, 4)
(96,)
(24,)
```



## ✓ 4. 모델링 및 성능평가

In [18]:
```python
# 랜덤포레스트 모델 사용 (참고 : 회귀모델은 RandomForestRegressor)
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(x_train, y_train)
```

Out[18]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [19]:
```python
# 모델을 사용하여 테스트 데이터 예측
y_pred = model.predict(x_val)
```

In [20]:
```python
# 모델 성능 평가 (accuracy, f1 score, AUC 등)
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, recall_score, precision_score
acc = accuracy_score(y_val, y_pred)  # (실제값, 예측값)
f1 = f1_score(y_val, y_pred)         # (실제값, 예측값)
# 다중분류일 경우 f1 = f1_score(y_val, y_pred, average = 'macro')
auc = roc_auc_score(y_val, y_pred)   # (실제값, 예측값)
```

In [21]:
```python
# 정확도(Accuracy)
print(acc)
```

```
1.0
```

In [22]:
```python
# F1 Score
print(f1)
```

```
1.0
```

In [23]:
```python
# AUC
print(auc)
```

```
1.0
```



In [24]:
```python
# 참고사항
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_val, y_pred) # (실제값, 예측값)
print(cm)

# #####  예측
# #####   0   1
```

```
# 실제 0 TN  FP
# 실제 1 FN  TP
```

```
[[11  0]
 [ 0 13]]
```

## ✓ 5. 예측값 제출



(주의) x_test 를 모델에 넣어 나온 예측값을 제출해야함

In [25]:
```
# (실기시험 안내사항)
# 아래 코드 예측변수와 수험번호를 개인별로 변경하여 활용
# pd.DataFrame({ 'result': y_result }).to_csv('수험번호.csv', index=False)

# 모델을 사용하여 테스트 데이터 예측

# 1. 특정 클래스로 분류할 경우 (predict)
y_result = model.predict(x_test)
print(y_result[:5])

# 2. 특정 클래스로 분류될 확률을 구할 경우 (predict_proba)
y_result_prob = model.predict_proba(x_test)
print(y_result_prob[:5])

# 이해해보기
result_prob = pd.DataFrame({
    'result': y_result,
    'prob_0': y_result_prob[:,0]
})
# setosa 일 확률 : y_result_prob[:,0]
# 그 외 종일 확률 : y_result_prob[:,1]

print(result_prob[:5])
```

```
[1 1 1 0 1]
[[0.   1.  ]
 [0.   1.  ]
 [0.   1.  ]
 [1.   0.  ]
 [0.02 0.98]]
   result  prob_0
0       1    0.00
1       1    0.00
2       1    0.00
3       0    1.00
4       1    0.02
```



In [26]:
```
# ★tip : 데이터를 저장한다음 불러와서 제대로 제출했는지 확인해보자
# pd.DataFrame({'result': y_result}).to_csv('수험번호.csv', index=False)
# df2 = pd.read_csv("수험번호.csv")
# print(df2.head())
```