제2유형_연습하기_타이타닉 생존자 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]:
```
###############  복사 영역  ###############
# 실기 시험 데이터셋으로 셋팅하기 (수정금지)

# Seaborn의 내장 타이타닉 데이터셋을 불러옵니다.
import seaborn as sns
df = sns.load_dataset('titanic')

x = df.drop('survived', axis=1)
y = df['survived']

# 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_test = pd.DataFrame(y_test) # 평가용

x_test.reset_index()
y_train.columns = ['target']
y_test.columns = ['target']
###############  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.
```

타이타닉 생존자 예측 문제

- 데이터의 결측치, 중복 변수값에 대해 처리하고

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

데이터 설명

- survival : 0 = No, 1 = Yes
- pclass : 객실 등급(1,2,3)

- sex : 성별
- age : 나이
- sibsp : 타이타닉호에 탑승한 형제/배우자의 수
- parch : 타이타닉호에 탑승한 부모/자녀의 수
- fare : 요금
- embarked : 탑승지 이름(C, Q, S) Cherbourg / Queenstown / Southampton
- (중복)class : 객실 등급(First, Second, Third)
- who : man, women, child
- adult_male : 성인남자인지 여부(True=성인남자, False 그외)
- deck : 선실번호 첫 알파벳(A,B,C,D,E,F,G)
- (중복) embark_town : 탑승지 이름(Cherbourg, Queenstown, Southampton)
- (중복) alive : 생존여부(no:사망, yes:생존)
- alone : 혼자 탑승했는지 여부(True=혼자, False=가족과 함께)

✓ 2. 데이터 탐색(EDA)

In [3]:
```
# 데이터의 행/열 확인

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

```
(712, 14)
(179, 14)
(712, 1)
```

In [4]:
```
# 초기 데이터 확인

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

```
     pclass     sex   age  sibsp  parch   fare embarked   class    who  \
3         1  female  35.0      1      0  53.10        S   First  woman
517       3    male   NaN      0      0  24.15        Q   Third    man
861       2    male  21.0      1      0  11.50        S  Second    man

     adult_male deck  embark_town alive  alone
3         False    C  Southampton   yes  False
517        True  NaN   Queenstown    no   True
861        True  NaN  Southampton    no  False
     pclass     sex   age  sibsp  parch   fare embarked   class    who  \
800       2    male  34.0      0      0   13.0        S  Second    man
341       1  female  24.0      3      2  263.0        S   First  woman
413       2    male   NaN      0      0    0.0        S  Second    man

     adult_male deck  embark_town alive  alone
800        True  NaN  Southampton    no   True
341       False    C  Southampton   yes  False
413        True  NaN  Southampton    no   True
     target
3         1
517       0
861       0
```

In [5]:
```
# 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: 712 entries, 3 to 608
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   pclass       712 non-null    int64
 1   sex          712 non-null    object
 2   age          579 non-null    float64
 3   sibsp        712 non-null    int64
 4   parch        712 non-null    int64
 5   fare         712 non-null    float64
 6   embarked     710 non-null    object
 7   class        712 non-null    category
 8   who          712 non-null    object
 9   adult_male   712 non-null    bool
 10  deck         164 non-null    category
 11  embark_town  710 non-null    object
 12  alive        712 non-null    object
 13  alone        712 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(3), object(5)
memory usage: 64.4+ KB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 179 entries, 800 to 410
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   pclass       179 non-null    int64
 1   sex          179 non-null    object
 2   age          135 non-null    float64
 3   sibsp        179 non-null    int64
 4   parch        179 non-null    int64
 5   fare         179 non-null    float64
 6   embarked     179 non-null    object
 7   class        179 non-null    category
 8   who          179 non-null    object
 9   adult_male   179 non-null    bool
 10  deck         39 non-null     category
 11  embark_town  179 non-null    object
 12  alive        179 non-null    object
 13  alone        179 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(3), object(5)
memory usage: 16.6+ KB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 712 entries, 3 to 608
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   target  712 non-null    int64
dtypes: int64(1)
memory usage: 11.1 KB
None
```

In [6]:
```
# 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())
```

```
           pclass         age       sibsp       parch        fare
count  712.000000  579.000000  712.000000  712.000000  712.000000
mean     2.307584   29.479568    0.518258    0.372191   31.741836
std      0.834926   14.355304    1.094522    0.792341   45.403910
min      1.000000    0.420000    0.000000    0.000000    0.000000
25%      2.000000   20.000000    0.000000    0.000000    7.895800
50%      3.000000   28.000000    0.000000    0.000000   14.454200
75%      3.000000   38.000000    1.000000    0.000000   31.275000
max      3.000000   74.000000    8.000000    6.000000  512.329200
           pclass         age       sibsp       parch        fare
count  179.000000  135.000000  179.000000  179.000000  179.000000
mean     2.312849   30.640741    0.541899    0.418994   34.043364
std      0.842950   15.258427    1.137797    0.859760   64.097184
min      1.000000    1.000000    0.000000    0.000000    0.000000
25%      2.000000   22.000000    0.000000    0.000000    7.925000
50%      3.000000   29.000000    0.000000    0.000000   14.500000
75%      3.000000   39.000000    1.000000    0.000000   30.285400
max      3.000000   80.000000    8.000000    5.000000  512.329200
           target
count  712.000000
mean     0.383427
std      0.486563
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      1.000000
```

In [7]:
```
# object, category 데이터도 추가 확인
```

```
print(x_train.describe(include='object'))
print(x_test.describe(include='object'))

print(x_train.describe(include='category'))
print(x_test.describe(include='category'))
```

```
         sex embarked  who  embark_town alive
count    712      710  712           710   712
unique     2        3    3             3     2
top     male        S  man  Southampton    no
freq     469      518  432           518   439
         sex embarked  who  embark_town alive
count    179      179  179           179   179
unique     2        3    3             3     2
top     male        S  man  Southampton    no
freq     108      126  105           126   110
        class deck
count     712  164
unique      3    7
top     Third    C
freq      391   47
        class deck
count     179   39
unique      3    7
top     Third    C
freq      100   12
```

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

```
     target
3         1
517       0
861       0
487       0
58        1
```

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

```
target
0         439
1         273
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())
```

```
pclass           0
sex              0
age            133
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           548
embark_town      2
alive            0
alone            0
dtype: int64
pclass           0
sex              0
age             44
sibsp            0
parch            0
fare             0
embarked         0
class            0
who              0
adult_male       0
deck           140
embark_town      0
alive            0
alone            0
dtype: int64
target    0
dtype: int64
```


In [11]:
```
# 결측치 제거
```

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
# x_train(712,14) : age(133), embarked(2), deck(548), embark_town(2)
# x_test(179,14) : age(44), deck(140)

# 변수 제거
# (중복) class
# (중복) embark_town
# (중복) alive
# (결측치 다수) deck
```

In [13]:
```
# 중복변수 제거
x_train = x_train.drop(['class', 'embark_town', 'alive', 'deck'], axis=1)
x_test = x_test.drop(['class', 'embark_town', 'alive', 'deck'], axis=1)
```

In [14]:
```
# 변수제거 확인
print(x_train.info())
print(x_test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 712 entries, 3 to 608
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   pclass      712 non-null    int64
 1   sex         712 non-null    object
 2   age         579 non-null    float64
 3   sibsp       712 non-null    int64
 4   parch       712 non-null    int64
 5   fare        712 non-null    float64
 6   embarked    710 non-null    object
 7   who         712 non-null    object
 8   adult_male  712 non-null    bool
 9   alone       712 non-null    bool
dtypes: bool(2), float64(2), int64(3), object(3)
memory usage: 51.5+ KB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 179 entries, 800 to 410
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   pclass      179 non-null    int64
 1   sex         179 non-null    object
 2   age         135 non-null    float64
 3   sibsp       179 non-null    int64
 4   parch       179 non-null    int64
 5   fare        179 non-null    float64
 6   embarked    179 non-null    object
 7   who         179 non-null    object
 8   adult_male  179 non-null    bool
 9   alone       179 non-null    bool
dtypes: bool(2), float64(2), int64(3), object(3)
memory usage: 12.9+ KB
None
```


In [15]:
```
# 결측치 대체
# x_train(712,14) : age(133), embarked(2)
# x_test(179,14) : age(44)

# age 변수
med_age = x_train['age'].median()
x_train['age'] = x_train['age'].fillna(med_age)
x_test['age'] = x_test['age'].fillna(med_age)  # train data의 중앙값으로

# embarked
mode_et = x_train['embarked'].mode()
x_train['embarked'] = x_train['embarked'].fillna(mode_et[0])  # 최빈값 [0] 주의
```


In [16]:
```
# 결측치 대체 여부 확인
print(x_train.isnull().sum())
print(x_test.isnull().sum())
```

```
pclass        0
sex           0
age           0
sibsp         0
parch         0
fare          0
embarked      0
who           0
adult_male    0
alone         0
dtype: int64
pclass        0
sex           0
age           0
sibsp         0
parch         0
fare          0
embarked      0
who           0
adult_male    0
alone         0
dtype: int64
```

In [17]:
```python
# 변수처리 (원핫인코딩)
x_train = pd.get_dummies(x_train)
x_test = pd.get_dummies(x_test)

print(x_train.info())
print(x_test.info())

# advanced 버전 사용
x_train_ad = x_train.copy()
x_test_ad = x_test.copy()
y_train_ad = y_train.copy()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 712 entries, 3 to 608
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   pclass      712 non-null    int64
 1   age         712 non-null    float64
 2   sibsp       712 non-null    int64
 3   parch       712 non-null    int64
 4   fare        712 non-null    float64
 5   adult_male  712 non-null    bool
 6   alone       712 non-null    bool
 7   sex_female  712 non-null    uint8
 8   sex_male    712 non-null    uint8
 9   embarked_C  712 non-null    uint8
 10  embarked_Q  712 non-null    uint8
 11  embarked_S  712 non-null    uint8
 12  who_child   712 non-null    uint8
 13  who_man     712 non-null    uint8
 14  who_woman   712 non-null    uint8
dtypes: bool(2), float64(2), int64(3), uint8(8)
memory usage: 40.3 KB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 179 entries, 800 to 410
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   pclass      179 non-null    int64
 1   age         179 non-null    float64
 2   sibsp       179 non-null    int64
 3   parch       179 non-null    int64
 4   fare        179 non-null    float64
 5   adult_male  179 non-null    bool
 6   alone       179 non-null    bool
 7   sex_female  179 non-null    uint8
 8   sex_male    179 non-null    uint8
 9   embarked_C  179 non-null    uint8
 10  embarked_Q  179 non-null    uint8
 11  embarked_S  179 non-null    uint8
 12  who_child   179 non-null    uint8
 13  who_man     179 non-null    uint8
 14  who_woman   179 non-null    uint8
dtypes: bool(2), float64(2), int64(3), uint8(8)
memory usage: 10.1 KB
None
```

In [18]:
```python
# (참고사항)원핫인코딩 후 변수의 수가 다른 경우
# => x_test의 변수의 수가 x_train 보다 많은 경우 (혹은 그 반대인 경우)

# 원핫인코딩 후 Feature 수가 다를 경우
# x_train = pd.get_dummies(x_train)
# x_test = pd.get_dummies(x_test)
# x_train.info()
```

```
# x_test.info()

# 해결방법(x_test의 변수가 수가 더 많은 경우의 코드)
# x_train = x_train.reindex(columns = x_test.columns, fill_value=0)
# x_train.info()
```

데이터 분리

In [19]:
```
# 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['target'],
                                                  test_size=0.2,
                                                  # stratify=y_train['target'], 옵션이 있을 때와 없을 때 차이점을 확
                                                  random_state=2023)

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

```
(569, 15)
(143, 15)
(569,)
(143,)
```

✅ 4. 모델링 및 성능평가

In [20]:
```
# 랜덤포레스트 모델 사용  (참고 : 회귀모델은 RandomForestRegressor)
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=2023)
model.fit(x_train, y_train)
```

Out[20]:
```
RandomForestClassifier(random_state=2023)
```

In [21]:
```
# 모델을 사용하여 테스트 데이터 예측
y_pred = model.predict(x_val)
```

In [22]:
```
# 모델 성능 평가 (정확도, F1 score, 민감도, 특이도 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)
auc = roc_auc_score(y_val, y_pred)     # (실제값, 예측값)
```

In [23]:
```
print(acc) # 정확도(Accuracy)
print(f1)  # f1 score
print(auc) # AUC
```

```
0.7902097902097902
0.7169811320754716
0.7751572327044025
```

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

# #####    예측
# #####   0   1
# 실제 0 TN  FP
# 실제 1 FN  TP
```

```
[[75 15]
 [15 38]]
```

실제 test셋으로 성능평가를 한다면?

In [25]:
```
# 모델을 사용하여 테스트 데이터 예측
y_pred_f = model.predict(x_test)

# 모델 성능 평가 (정확도, F1 score, AUC)
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
acc_f = accuracy_score(y_test, y_pred_f)    # (실제값, 예측값)
f1_f = f1_score(y_test, y_pred_f)           # (실제값, 예측값)
auc_f = roc_auc_score(y_test, y_pred_f)     # (실제값, 예측값)
```

In [26]:
```
print(acc_f) # 정확도(Accuracy)
print(f1_f)  # f1 score
print(auc_f) # AUC
```

```
0.7541899441340782
0.6716417910447761
0.7351778656126482
```

Advanced 버전

(주의) 전체 코드 실행시간이 1분으로 제한되어 있기 때문에, 가능하면 30초 미만으로 할 것!

In [27]:
```python
# GridSearch CV를 활용한 하이퍼파라미터 최적화
#  - GridSearch : 격자탐색
#  - CV = CrossValidation, 교차검증

# (주의) 별도로 train/val 분리가 필요하지 않음
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

rf_params = { 'n_estimators' : [30,70,100],
              'max_depth' : [6, 8, 10],
              'min_samples_leaf' : [1, 2, 3],
            }
# n_estimators : tree의 개수
# max_depth : tree의 최대 깊이
# min_samples_leaf : leaf node(더이상 분할되지 않는 마지막 노드)가 되기 위해 필요한 최소 샘플 수
# (이 값보다 작은 수의 샘플이 해당 노드에 있을 경우, 더 이상 분할하지 않음)

# RandomForestClassifier 객체 생성 후 GridSearchCV 수행
rf = RandomForestClassifier(random_state = 2023)
grid_rf = GridSearchCV(rf, param_grid = rf_params, cv = 10)
grid_rf.fit(x_train_ad, y_train_ad['target'])
# y값 입력 시 주의(1차원 형태로 들어가야함)

print('최적 하이퍼파라미터: ', grid_rf.best_params_)
print('Best 예측정확도: ', grid_rf.best_score_)
```

```
최적 하이퍼파라미터:  {'max_depth': 10, 'min_samples_leaf': 3, 'n_estimators': 70}
Best 예측정확도:  0.8427034428794992
```

In [28]:
```python
# (참고) help 함수를 통한 함수 사용법 확인

# from sklearn.model_selection import GridSearchCV
# help(GridSearchCV)

# from sklearn.ensemble import RandomForestClassifier
# help(RandomForestClassifier)
```


In [29]:
```python
# 위의 최적 하이퍼파라미터로 랜덤포레스트 모델을 생성
from sklearn.ensemble import RandomForestClassifier
model_h = RandomForestClassifier(n_estimators = 70,
                                 max_depth = 10,
                                 min_samples_leaf = 3,
                                 random_state = 2023)

model_h.fit(x_train_ad, y_train_ad['target'] )  # y 데이터 입력시 주의
```

Out[29]:
```
RandomForestClassifier(max_depth=10, min_samples_leaf=3, n_estimators=70,
                       random_state=2023)
```

In [30]:
```python
# 모델 성능 평가 (정확도, F1 score, AUC)
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

# test 데이터셋으로 성능평가
y_pred_h = model_h.predict(x_test)

acc_h = accuracy_score(y_test, y_pred_h) # (실제값, 예측값)
f1_h = f1_score(y_test, y_pred_h)        # (실제값, 예측값)
auc_h = roc_auc_score(y_test, y_pred_h)  # (실제값, 예측값)

print('HP 최적화 Acc:', acc_h )
print('HP 최적화 f1:', f1_h )
print('HP 최적화:AUC', auc_h )
# HP : Hyperparameter

print('기본모델 Acc:', acc_f)
print('기본모델 f1:', f1_f )
print('기본모델 AUC:', auc_f )
```



```
HP 최적화 Acc: 0.7988826815642458
HP 최적화 f1: 0.7272727272727272
HP 최적화:AUC 0.7796442687747035
기본모델 Acc: 0.7541899441340782
기본모델 f1: 0.6716417910447761
기본모델 AUC: 0.7351778656126482
```


5. 예측 및 제출

✔ 5. 예측값 제출

(주의) x_test 셋을 모델에 넣어 나온 예측값을 제출해야함

In [31]:
```
# (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
y_result = model.predict(x_test)

# 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]
})
# Class가 0일 확률 : y_result_prob[:,0]
# Class가 1일 확률 : y_result_prob[:,1]

print(result_prob[:5])
```

```
[0 1 0 0 0]
[[0.52116667 0.47883333]
 [0.13       0.87      ]
 [1.         0.        ]
 [0.9        0.1       ]
 [0.98       0.02      ]]
   result    prob_0
0       0  0.521167
1       1  0.130000
2       0  1.000000
3       0  0.900000
4       0  0.980000
```

In [32]:
```
# ★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())
```