타이타닉 생존자 예측 대회

학습 내용

- 1-1 데이터 불러오기
- 1-2 데이터 탐색하기
- 1-3 결측치 처리
- 1-4 모델 선택 및 평가

데이터

Data Fields

구분	설명	값
Survival	생존 여부	Survival. 0 = No, 1 = Yes
Pclass	티켓의 클래스	Ticket class. 1 = 1st, 2 = 2nd, 3 = 3rd
Sex	성별(Sex)	남(male)/여(female)
Age	나이(Age in years.)	
SibSp	함께 탑승한 형제와 배우자의 수 /siblings, spouses aboard the Titanic.	
Parch	함께 탑승한 부모, 아이의 수	# of parents / children aboard the Titanic.
Ticket	티켓 번호(Ticket number)	(ex) CA 31352, A/5. 2151
Fare	탑승료(Passenger fare)	
Cabin	객실 번호(Cabin number)	
Embarked	탑승 항구(Port of Embarkation)	C = Cherbourg, Q = Queenstown, S = Southampton

- siblings : 형제, 자매, 형제, 의붓 형제
- spouses : 남편, 아내 (정부와 약혼자는 무시)
- Parch : Parent(mother, father), child(daughter, son, stepdaughter, stepson)

In [1]:

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

In [2]:

```
train = pd.read_csv("data/titanic/train.csv")
test = pd.read_csv("data/titanic/test.csv")
sub = pd.read_csv("data/titanic/gender_submission.csv")
```

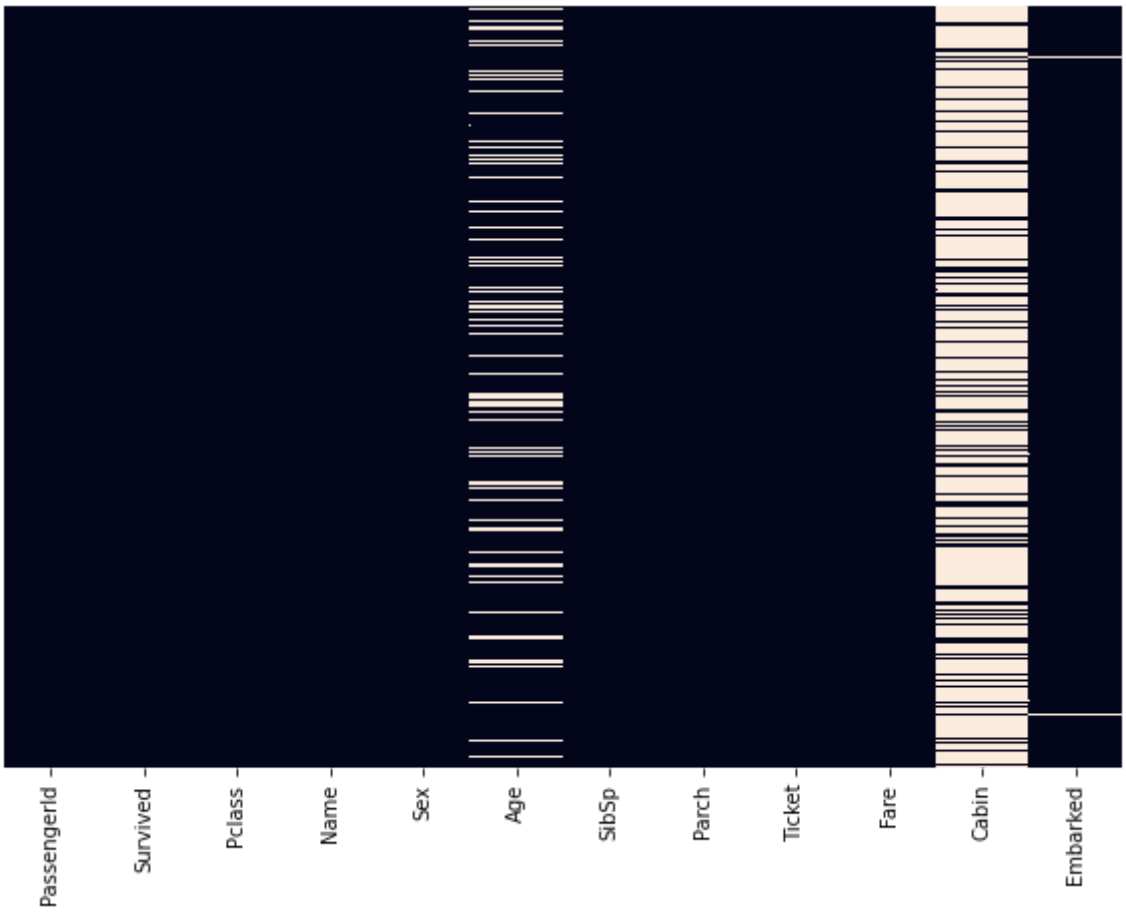
데이터 EDA

In [3]:

```
plt.figure(figsize=(10,7))
sns.heatmap(train.isnull(), yticklabels=False, cbar=False)  # cbar : colorbar를 그리지 않음.
```

Out[3]:

<matplotlib.axes._subplots.AxesSubplot at 0x2aa76d7b0d0>

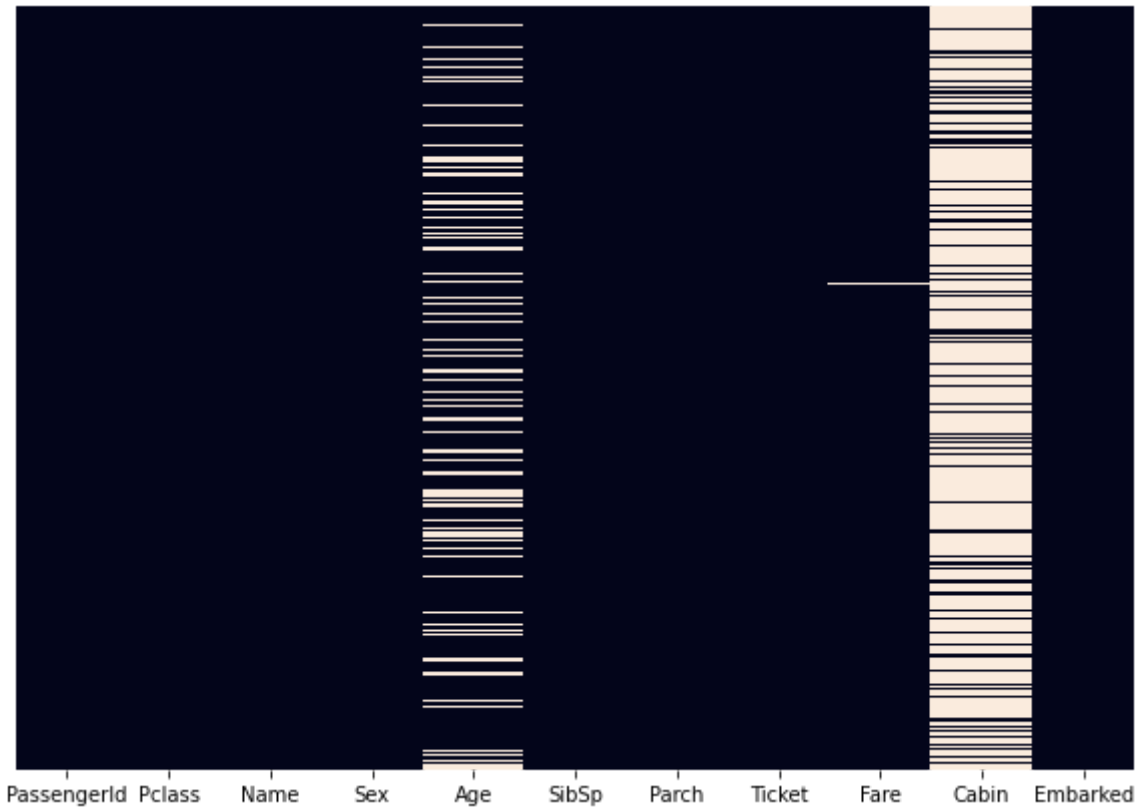


In [4]:

```
plt.figure(figsize=(10,7))
sns.heatmap(test.isnull(), yticklabels=False, cbar=False)  # cbar : colorbar를 그리지 않음.
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2aa71f2c700>
```



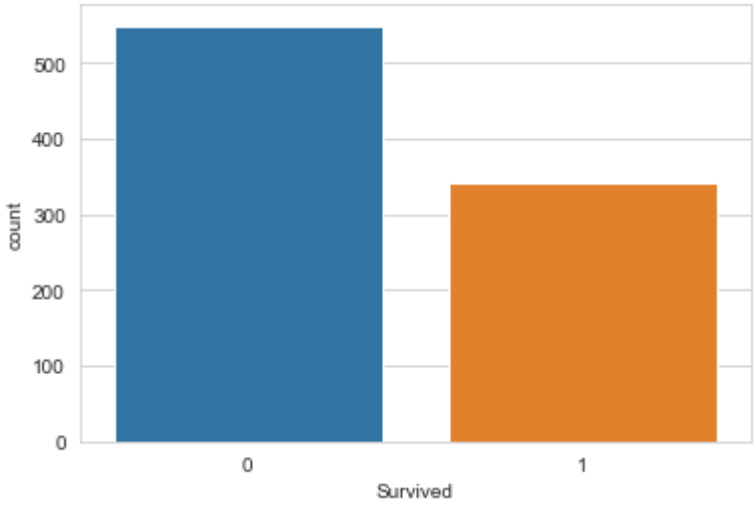
생존자와 사망자의 비율은 어떻게 될까?

In [5]:

```
sns.set_style('whitegrid')
sns.countplot(x='Survived', data=train)
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2aa77523f40>
```



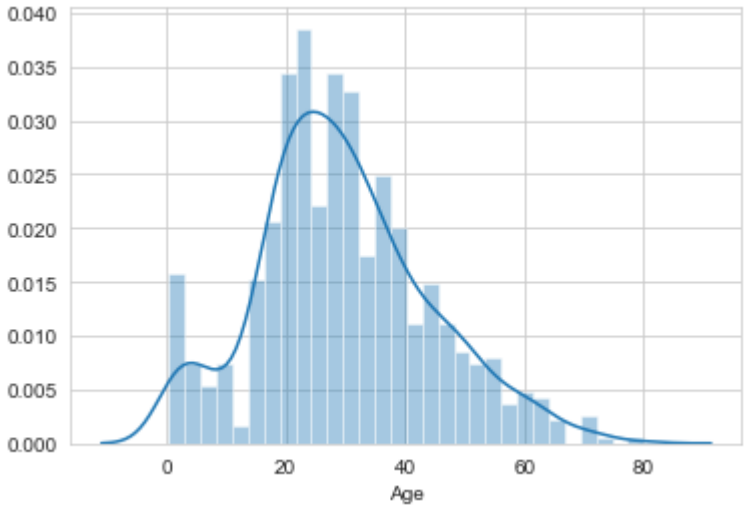
나이에 대해 살펴보기

In [6]:

```
sns.distplot(train['Age'].dropna(), bins=30)
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2aa775929d0>
```



- 20~50대에 많이 분포, 어린아이들도 있음.

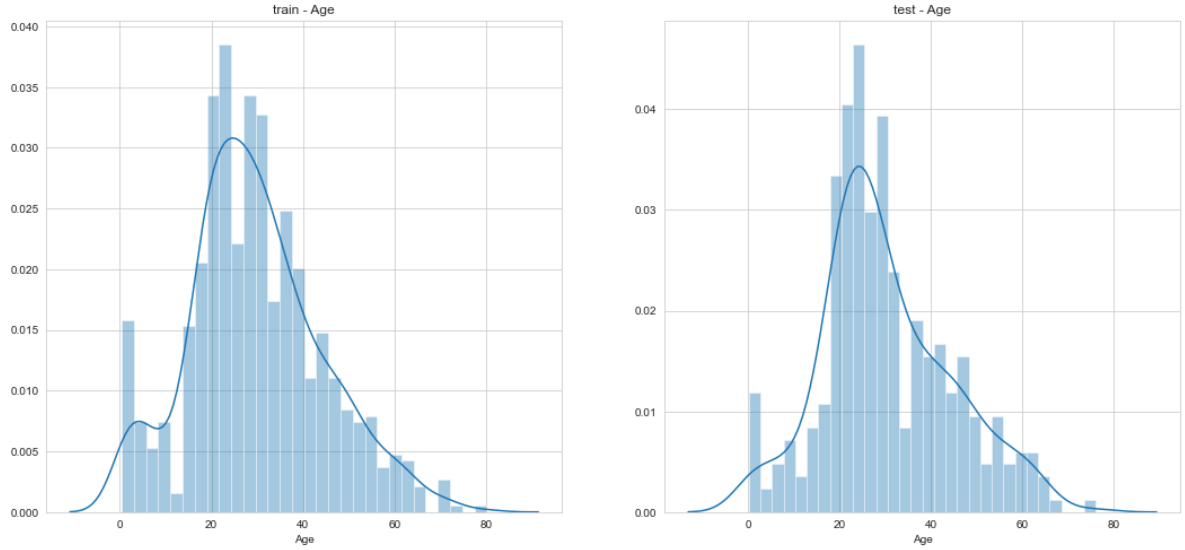
데이터(train, test)를 비교해 보기

In [7]:

```
f,ax=plt.subplots(1,2,figsize=(18,8))

# 첫번째 그래프
sns.distplot(train['Age'].dropna(), bins=30,ax=ax[0])
ax[0].set_title('train - Age')

# 두번째 그래프
sns.distplot(test['Age'].dropna(), bins=30,ax=ax[1])
ax[1].set_title('test - Age')
plt.show()
```



데이터 결측치 확인 후, 이를 처리해보자

In [9]:

```
print( train.info() )
print()
print( test.info() )
```

```
<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

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
None
```

Age에 대한 처리

In [10]:

```
train['Age'] = train['Age'].fillna(train['Age'].mean())
test['Age'] = test['Age'].fillna(test['Age'].mean())
```

In [11]:

```
print(train.isnull().sum())
print(test.isnull().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

승선항(Embarked) 처리

In [12]:

```
train['Embarked'].value_counts()
```

Out[12]:

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

- S = Southampton 의 승선한 사람들이 많다.

In [13]:

```
train['Embarked'] = train['Embarked'].fillna('S')
```

Test 데이터의 Fare(탑승료) 처리

In [14]:

```
test['Fare'] = test['Fare'].fillna(test['Fare'].mean())
```

In [15]:

```
print(train.isnull().sum())
print(test.isnull().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         0
dtype: int64
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

In [16]:

```
train.info()
```

```
<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          891 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     891 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [17]:

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          418 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         418 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

데이터를 나눠서 평가를 해 보자.

In [26]:

```
sel = ['PassengerId', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare']

all_X = train[sel]
all_y = train['Survived']

last_X_test = test[sel]
```

In [27]:

```
from sklearn.model_selection import train_test_split
```

In [28]:

```
X_train, X_test, y_train, y_test = train_test_split(all_X,
                                                    all_y,
                                                    stratify=all_y,
                                                    test_size=0.3,
                                                    random_state=77 )
```

모델 만들고 평가

In [34]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [35]:

```
model = DecisionTreeClassifier().fit(X_train, y_train)
acc_tr = model.score(X_train, y_train)
acc_test = model.score(X_test, y_test)
acc_tr, acc_test
```

Out[35]:

```
(1.0, 0.6194029850746269)
```

실습

- 의사결정트리의 하이퍼 파리미터를 활용하여 좋은 모델을 선택하고 이를 적용하여 제출해 보자.

In [38]:

```
depth_param = range(1,10)

for i in depth_param:
    model = DecisionTreeClassifier(max_depth=i).fit(X_train, y_train)
    acc_tr = model.score(X_train, y_train)
    acc_test = model.score(X_test, y_test)
    print("max_depth : {} , 정확도 : {} {}".format(i, acc_tr, acc_test) )
```

```
max_depth : 1 , 정확도 : 0.6211878009630819 0.5970149253731343
max_depth : 2 , 정확도 : 0.6918138041733547 0.6604477611940298
max_depth : 3 , 정확도 : 0.7255216693418941 0.664179104477612
max_depth : 4 , 정확도 : 0.7512038523274478 0.6977611940298507
max_depth : 5 , 정확도 : 0.7704654895666132 0.6940298507462687
max_depth : 6 , 정확도 : 0.8009630818619583 0.6716417910447762
max_depth : 7 , 정확도 : 0.8298555377207063 0.6791044776119403
max_depth : 8 , 정확도 : 0.8667736757624398 0.6604477611940298
max_depth : 9 , 정확도 : 0.9036918138041734 0.6305970149253731
```

In [39]:

```
model = DecisionTreeClassifier(max_depth=4).fit(X_train, y_train)
model.fit(all_X, all_y)
pred = model.predict(last_X_test)
sub['Survived'] = pred
sub.to_csv("tree_second_sub.csv", index=False)  # 0.65879
```

In [46]:

```
import os
files = os.listdir()
print("파일 유무 확인 : ", "tree_second_sub.csv" in files)
```

```
파일 유무 확인 :  True
```

실습

- LogisticRegression, LinearSVC, Knn 모델을 만들고, 가장 성능이 좋은 모델로 제출해 보자

In [47]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
```

In [48]:

```
model = LogisticRegression().fit(X_train, y_train)
acc_tr = model.score(X_train, y_train)
acc_test = model.score(X_test, y_test)
acc_tr, acc_test
```

```
C:₩Users₩WJ₩anaconda3₩lib₩site-packages₩sklearn₩linear_model₩_logistic.py:762: Conve
rgenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-lear
n.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (h
ttps://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[48]:

```
(0.6918138041733547, 0.7164179104477612)
```

In [49]:

```
model = LinearSVC().fit(X_train, y_train)
acc_tr = model.score(X_train, y_train)
acc_test = model.score(X_test, y_test)
acc_tr, acc_test
```

```
C:₩Users₩WJ₩anaconda3₩lib₩site-packages₩sklearn₩svm₩_base.py:976: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[49]:

```
(0.39646869983948635, 0.3843283582089552)
```

In [50]:

```
model = KNeighborsClassifier().fit(X_train, y_train)
acc_tr = model.score(X_train, y_train)
acc_test = model.score(X_test, y_test)
acc_tr, acc_test
```

Out[50]:

```
(0.7367576243980738, 0.6268656716417911)
```

In [51]:

```
model = LogisticRegression()
model.fit(all_X, all_y)
pred = model.predict(last_X_test)
sub['Survived'] = pred
sub.to_csv("third_lgreg_sub.csv", index=False)
```

```
C:₩Users₩WJ₩anaconda3₩lib₩site-packages₩sklearn₩linear_model₩_logistic.py:762: Conve
rgenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-lear
n.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (h
ttps://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

In [52]:

```
files = os.listdir()
print("파일 유무 확인 : ", "third_lgreg_sub.csv" in files)  # 0.66746
```

```
파일 유무 확인 :  True
```