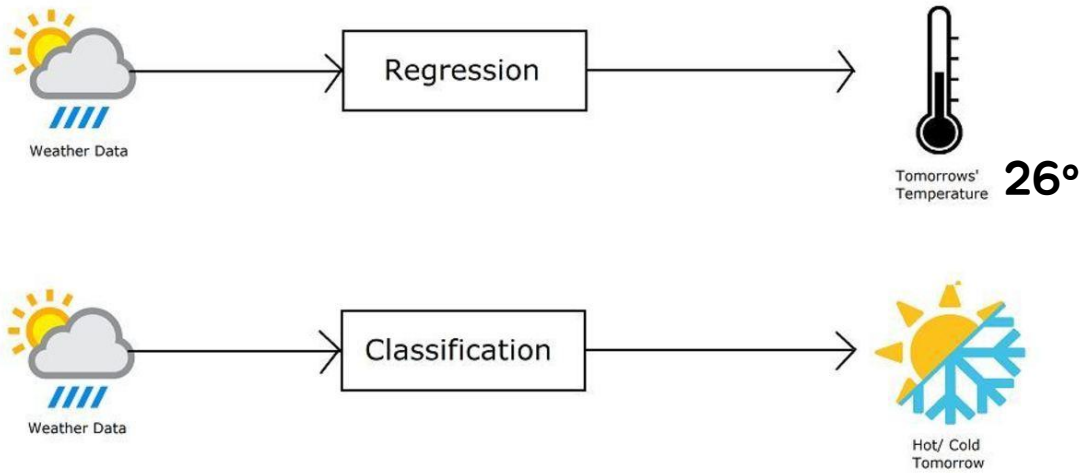
14. 데이터 기반 분류 모델

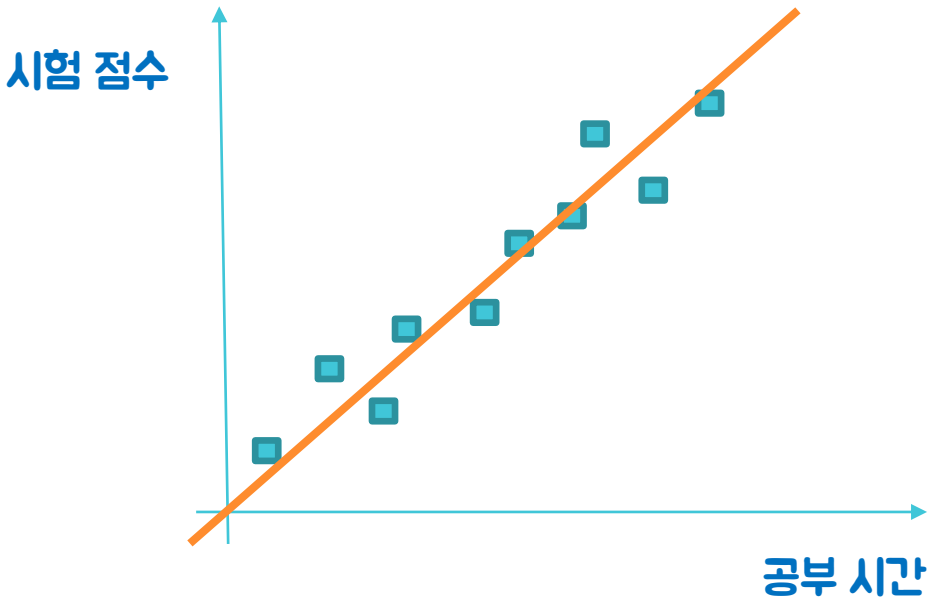
Logistic Regression (로지스틱 회귀) 모델

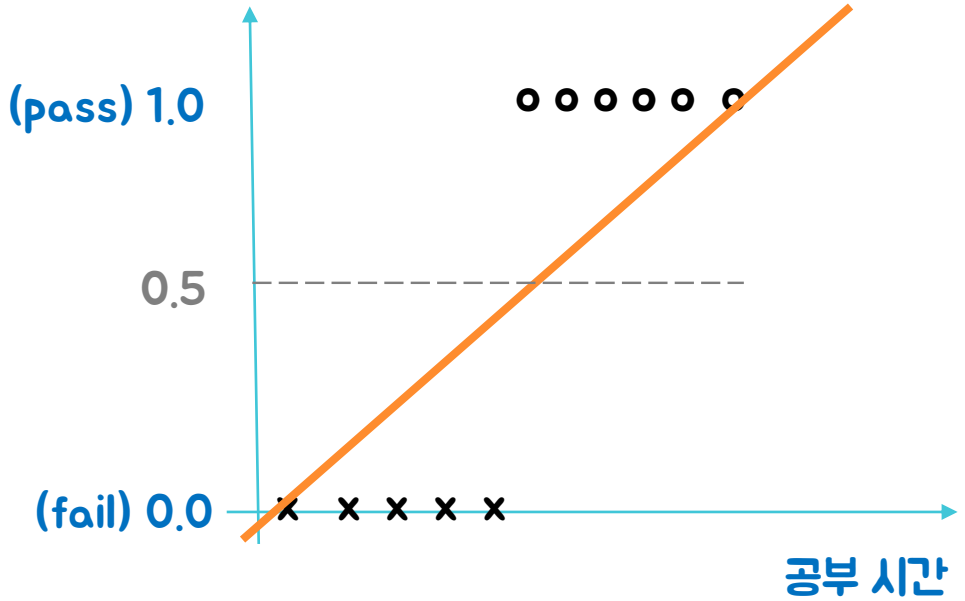
- 입력에 대한 이진 분류 (binary classification) 학습 모델
- 독립 변수의 선형 결합으로 종속 변수를 설명한다는 관점에서는 선형 회귀 모델과 유사
- 종속 변수가 0과 1 사이의 확률로 예측된 후 두개의 범주 중 어디서 속하는지 분류하는 모델
  - 예시: 수신한 이메일이 스팸인지 아닌지
    경기에서 승리할 수 있을지 없을지
    오늘 비가 오는데 더울지 추울지



예시: 공부시간 기반 시험 통과 여부

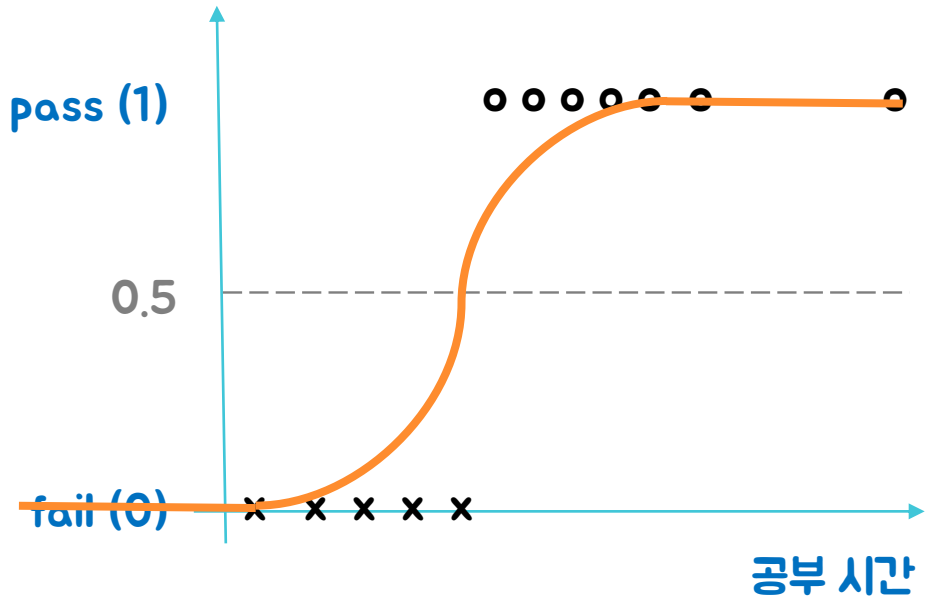
- 연속형 수치 결과 예측 : 90점, 85점 (선형 회귀 분석이 적합함)
- 이분 범주형 결과 예측 : 합격, 불합격 (선형 회귀 분석을 적용하면?)





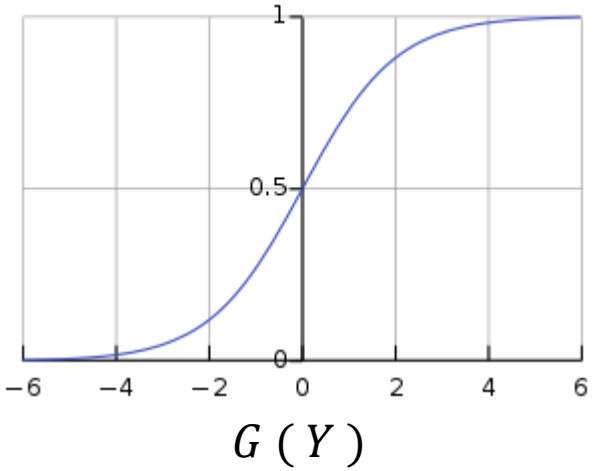
예시: 공부시간 기반 시험 통과 여부

- 이분 범주형 모델은 결과가 0~1 로 한정 지을 필요가 있음



$$Y = aX + b$$

$$H(X) = G(Y) = \frac{1}{1 + e^{-Y}}$$



예시: 공부시간 기반 시험 통과 여부

- 사이킷런 라이브러리 **LinearRegression** 로 학습을 시킨다면?

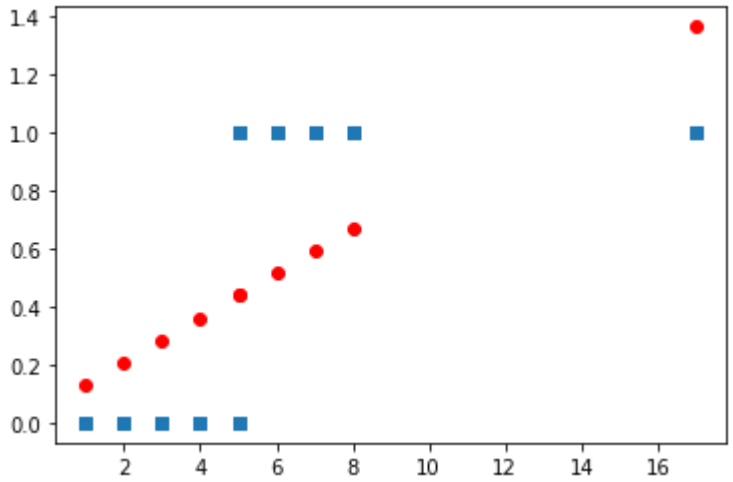
```
from sklearn.linear_model import LinearRegression

study_hour = [ [2], [5], [1], [7], [3], [4], [17], [6], [8], [5] ]

class_pass = [ 0, 1, 0, 1, 0, 0, 1, 1, 1, 0 ]

lr = LinearRegression()

train_X = study_hour
train_Y = class_pass

lr.fit(train_X, train_Y)
```

예시: 공부시간 기반 시험 통과 여부

- 학습된 모델이 데이터를 얼마나 잘 설명할 수 있는지 score( ) 함수로 확인

```
lr.score(train_X, train_Y)
pred_Y = lr.predict(train_X)
plt.scatter(train_X, train_Y, marker='s')
plt.scatter(train_X, pred_Y, color='red')
plt.show()
```

```
0.43171806167400884
```



- 학습된 모델로 새로운 데이터 입력에 대해 예측 값을 확인

```
test_X = [ [4.5], [5.5] ]
pred_Y = lr.predict( test_X )
print( pred_Y )
```

```
[0.39977974 0.47687225]
```

예시: 공부시간 기반 시험 통과 여부

- 사이킷런 라이브러리에서 **LogisticRegression** (로지스틱 회귀) 로 학습

```
from sklearn.linear_model import LogisticRegression

study_hour = [ [2], [5], [1], [7], [3], [4], [17], [6], [8], [5] ]

class_pass = [ 0, 1, 0, 1, 0, 0, 1, 1, 1, 0 ]

lr = LogisticRegression()

train_X = study_hour
train_Y = class_pass

lr.fit(train_X, train_Y)
```

예시: 공부시간 기반 시험 통과 여부

- 학습된 모델이 데이터를 얼마나 잘 설명할 수 있는지 score( ) 함수로 확인

```
lr.score(train_X, train_Y)
pred_Y = lr.predict(train_X)
print( list( pred_Y ) )
print(  train_Y  )
```

```
0.9
[0, 0, 0, 1, 0, 0, 1, 1, 1, 0]
[0, 1, 0, 1, 0, 0, 1, 1, 1, 0]
```

- 학습된 모델로 새로운 데이터 입력에 대해 예측 값을 확인

```
test_X = [ [4.5], [5.5] ]
pred_Y = lr.predict( test_X )
print( pred_Y )
```

```
[0 1]
```

생존 가능 여부 분류 예측 : 타이타닉 데이터셋

- 타이타닉호 사건에서 생존 여부를 예측할 수 있을까?
- 타이타닉 데이터셋 전처리 후 데이터 탐색을 통해 학습에 사용할 변수 선정
- 데이터셋을 학습용과 평가용으로 나눠 학습용 데이터로 LogisticRegression 모델 학습
- 학습 된 모델을 평가용 데이터로 생존 여부 예측 성능을 평가

생존 가능 여부 분류 예측 : 타이타닉 데이터셋

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
print(titanic)
```

```
     survived  pclass     sex   age  sibsp  parch     fare embarked   class      who  adult_male deck  embark_town alive  alone
0           0       3    male  22.0      1      0   7.2500        S   Third      man        True  NaN  Southampton    no  False
1           1       1  female  38.0      1      0  71.2833        C   First    woman       False    C    Cherbourg   yes  False
2           1       3  female  26.0      0      0   7.9250        S   Third    woman       False  NaN  Southampton   yes   True
3           1       1  female  35.0      1      0  53.1000        S   First    woman       False    C  Southampton   yes  False
4           0       3    male  35.0      0      0   8.0500        S   Third      man        True  NaN  Southampton    no   True
..        ...     ...     ...   ...    ...    ...      ...      ...     ...      ...         ...  ...          ...   ...    ...
886         0       2    male  27.0      0      0  13.0000        S  Second      man        True  NaN  Southampton    no   True
887         1       1  female  19.0      0      0  30.0000        S   First    woman       False    B  Southampton   yes   True
888         0       3  female   NaN      1      2  23.4500        S   Third    woman       False  NaN  Southampton    no  False
889         1       1    male  26.0      0      0  30.0000        C   First      man        True    C    Cherbourg   yes   True
890         0       3    male  32.0      0      0   7.7500        Q   Third      man        True  NaN   Queenstown    no   True

[891 rows x 15 columns]
```

타이타닉 데이터셋 설명

컬럼 명	의미	데이터 값
survived	생존여부	0 (사망) / 1 (생존)
pclass	객실 등급 (숫자)	1 / 2 / 3
sex	성별	male/female
age	나이	0 ~ 80
sibsp	형제자매 / 배우자 인원수	0 ~ 8
parch:	부모 / 자식 인원수	0 ~ 6
fare:	요금	0 ~ 512.3292
embarked	탑승 항구	S (Southampton) C (Cherbourg) Q (Queenstown)
class	좌석등급 (영문)	First, Second, Third
who	성별	man / woman
deck	객실 고유 번호 가장 앞자리 알파벳	A,B,C,D,E,F,G
embark_town	탑승 항구 (영문)	Southampton / Cherbourg / Queenstown
alive	생존여부 (영문)	no (사망) / yes (생존)
alone	혼자인지 여부	True (가족 X) / False (가족 O)

데이터 전처리

```
dataset = titanic.copy()
dataset.drop( ['class', 'who', 'deck', 'embark_town', 'alive' ], axis=1, inplace=True )
print( dataset.count() )
print( dataset.info() )
```

- 필요한 컬럼만 남김
- 데이터 개수 확인
- 데이터 타입 확인

```
survived       891
pclass         891
sex            891
age            714
sibsp          891
parch          891
fare           891
embarked       889
adult_male     891
alone          891
dtype: int64
```

```
#   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   adult_male   891 non-null    bool
 9   alone        891 non-null    bool
```

데이터 변환

- 선형 상관계수를 계산하거나,

  LogisticRegression 학습을 하기 위해

  선택된 데이터를 수치형 데이터로 변환

```
dataset['sex'] = dataset['sex'].apply(encoding_sex)
dataset['embarked'] = dataset['embarked'].apply(encoding_embarked)
dataset['alone'] = dataset['alone'].apply(encoding_alone)
dataset['adult_male'] = dataset['adult_male'].apply(encoding_adult_male)
```

```
def encoding_sex(x):
  if x == 'male':
    return 0
  else:
    return 1

def encoding_embarked(x):
  if x == 'S':
    return 0
  elif x == 'C':
    return 1
  else:
    return 2

def encoding_adult_male(x):
  if x == True:
    return 1
  else:
    return 0

def encoding_alone(x):
  if x == True:
    return 1
  else:
    return 0
```

데이터셋 분리 (학습용 & 평가용)

- 데이터 약 20% 를 평가용으로 분리
- 두 데이터셋의 평균 값을 확인

```
trainset = dataset.iloc[ 150 :  ]
testset = dataset.iloc[ : 150 ]
print( trainset.mean() )
print( testset.mean() )
```

```
survived        0.426984
pclass          2.207937
sex             0.376190
age            30.219654
sibsp           0.500000
parch           0.419048
fare           36.265979
embarked        0.341270
adult_male      0.571429
alone           0.574603
```

```
survived        0.353333
pclass          2.406667
sex             0.366667
age            28.053780
sibsp           0.633333
parch           0.413333
fare           28.794249
embarked        0.400000
adult_male      0.586667
alone           0.53333
```

결측 데이터 제거

- 두 데이터셋의 결측치 여부를 확인 후 결측 데이터는 제거

```
#print( trainset.isna().sum() )
#print( testset.isna().sum() )

trainset.dropna(inplace=True)
testset.dropna(inplace=True)
print( trainset.count() )
print( testset.count() )
```

```
survived      549
pclass        549
sex           549
age           549
sibsp         549
parch         549
fare          549
embarked      549
adult_male    549
alone         549
```

```
survived      127
pclass        127
sex           127
age           127
sibsp         127
parch         127
fare          127
embarked      127
adult_male    127
alone         127
```

데이터 탐색 (상관계수)

- 선형 상관계수 값 확인
  - 생존 여부와 선형 관계성이 높은 수치들을 확인
  - 선형 관계성이 낮아 보이는 항목에 대해서는 시각화를 통해 (비선형성) 관련성 여부를 확인

```
res = trainset.corr()
print( res['survived'] )
```

```
survived      1.000000
pclass       -0.401491
sex           0.509082
age          -0.082350
sibsp        -0.015502
parch         0.086772
fare          0.297462
embarked      0.103253
adult_male   -0.530561
alone        -0.217397
```

데이터 탐색 (시각화)

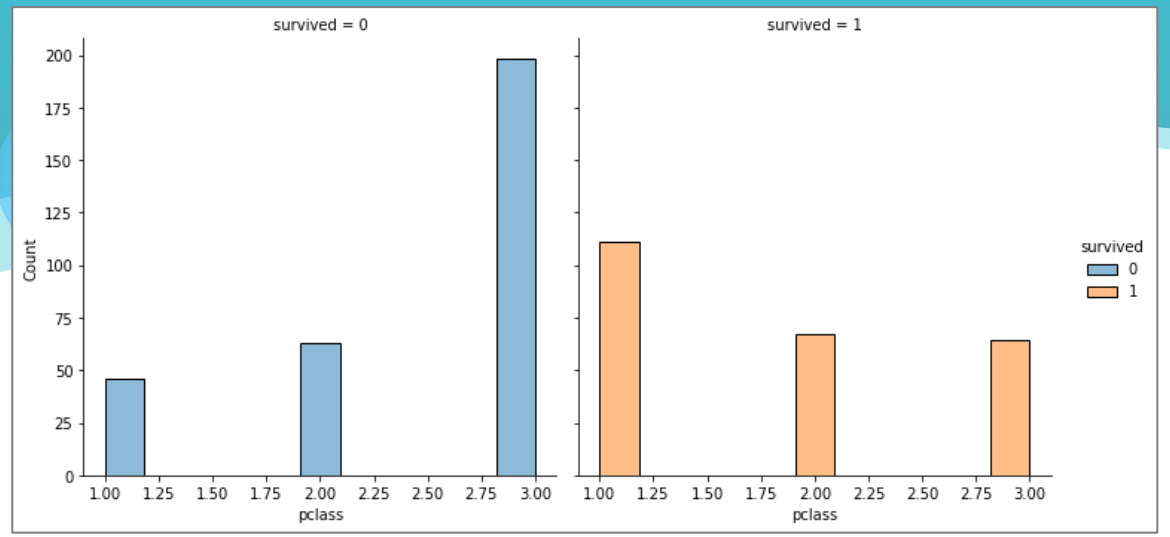
- 목표 변수 (종속 변수) 가 연속형 수치 데이터라면 산점도 그래프가 유용하지만,
  범주형이라면 히스토그램 그래프이 더 효과적일 수 있음

```
for c in trainset.columns:
  if c != 'survived':
    sns.histplot( x=c, hue='survived', data=trainset)
    plt.show()
```

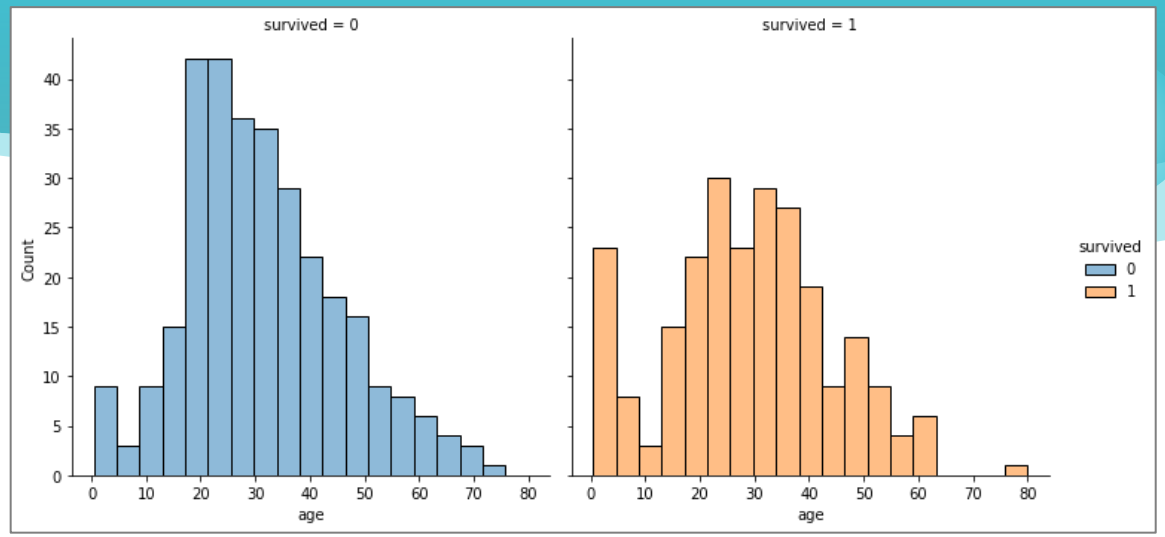
histplot( ) 은 겹쳐지는 부분이 가독성이 낮음

displot( ) 은 범주별로 그래프를 생성

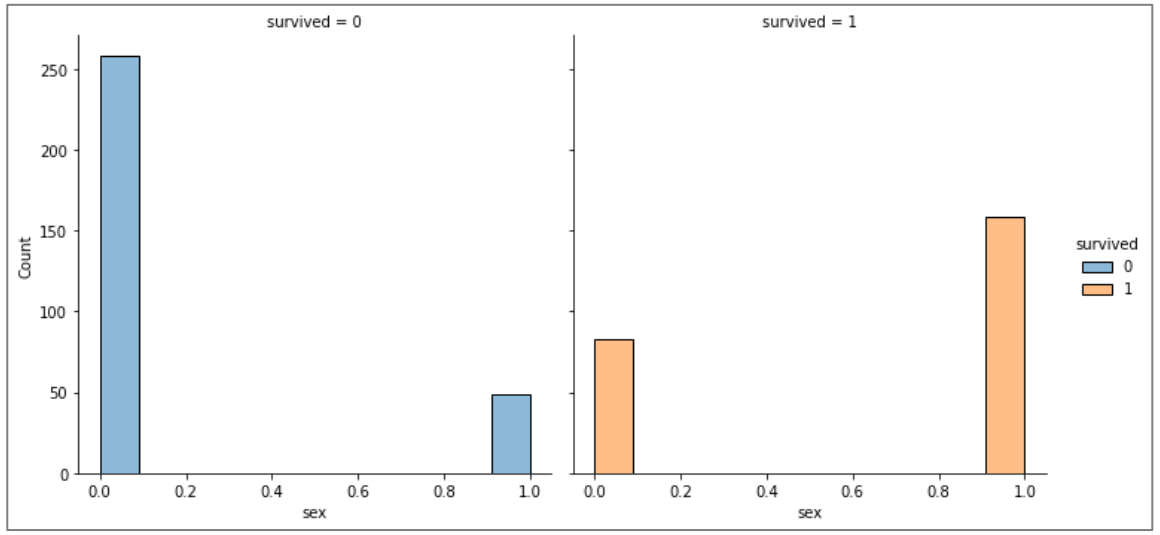
```
for c in trainset.columns:
  if c != 'survived':
    sns.displot( x=c, col='survived', hue='survived', data=trainset)
    plt.show()
```



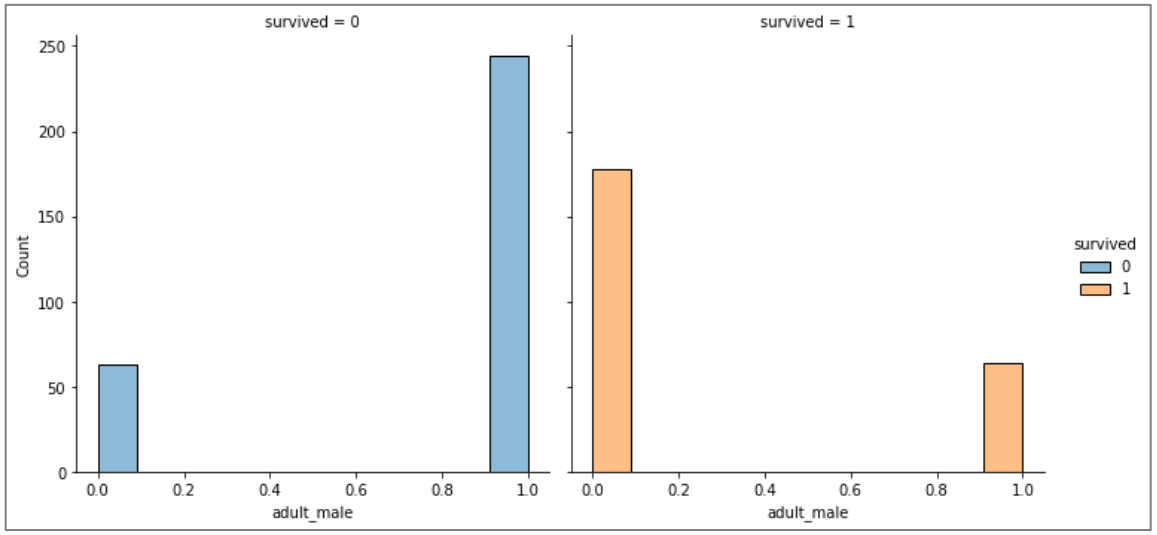
pclass



age



sex



adult_male

로지스틱 회귀 모델 학습

- 독립변수와 종속변수를 분리하여 LogisticRegression 모델 학습

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
train_X = trainset.drop(['survived'], axis=1)
train_Y = trainset['survived']
test_X = testset.drop(['survived'], axis=1)
test_Y = testset['survived']
lr.fit( train_X, train_Y )
```

ConvergenceWarning 발생!

아직 모델이 충분히 학습이 되지 못함

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1): STOP: TOTAL NO. of ITERATIONS
REACHED LIMIT.
```

로지스틱 회귀 모델 학습

- LogisticRegression 모델 학습이 최적으로 수렴될 수 있게 max_iter 값을 높임

```
# 모델 학습 최대 횟수를 기본값 100 에서 300 으로 변경
lr = LogisticRegression(max_iter=300)
lr.fit(train_X, train_Y)
# 학습 된 모델로 학습용 & 평가용 데이터에 대해 score 확인
lr.score(train_X, train_Y)
lr.score(test_X, test_Y)
```

```
LogisticRegression(max_iter=300)
0.8105646630236795
0.8031496062992126
```

학습된 모델 계수 (기울기 확인)

- 학습된 모델 계수 (기울기) 값을 확인
  - .coef_ 를 출력해보면 2차원 배열임

```
print( lr.coef_ )
coeff = pd.Series(data=lr.coef_[0], index=train_X.columns)
print( coeff.sort_values(ascending=False) )
```

```
[[-1.15388331  0.41996327 -0.03024565 -0.62060609 -0.43242452
   0.0036257 0.00703928 -2.21347505 -0.77219266]]
```

```
sex            0.419963
embarked       0.007039
fare           0.003626
age           -0.030246
parch         -0.432425
sibsp         -0.620606
alone         -0.772193
pclass        -1.153883
adult_male    -2.213475
```

- 선형 회귀 모델처럼 계수를 해석하기 위해서는 추가 작업이 요구됨. 본 강의에서는 참고만.

독립 변수 선택에 따른 모델 성능

- 모델 학습 시 사용할 독립 변수를 선정
  - 데이터 탐색, 시각화 과정에서 얻은 통찰력을 토대로 선정

```
columns = ['pclass','sex', 'age', 'sibsp', 'parch', 'fare', 'adult_male']
train_X = trainset[columns]
train_Y = trainset['survived']
test_X = testset[columns]
test_Y = testset['survived']
```

```
lr.fit( train_X, train_Y )
print( lr.score(train_X, train_Y) )
print( lr.score(test_X, test_Y) )
```

```
0.8214936247723132
0.8267716535433071
```

결측 데이터 활용에 따른 모델 성능

- 학습용 데이터셋에서 age 결측 된 행을 모두 제거하는 대신 age 값을 평균 값으로 대체
  - 평가용 데이터는 동일하게 사용
  - 학습 데이터가 증가되는 효과 등으로 모델의 일반화 성능이 향상될 수 있음

```
trainset = dataset.iloc[ 150 :  ].copy()
trainset.fillna( trainset['age'].mean(), inplace=True )
columns = ['pclass','sex', 'age', 'sibsp', 'parch', 'fare', 'adult_male']
train_X = trainset[columns]
train_Y = trainset['survived']
# testset 은 결측 데이터가
# 제거된 버전을 동일하게 사용
```

```
lr.fit( train_X, train_Y )
print( lr.score(train_X, train_Y) )
print( lr.score(test_X, test_Y) )
```

```
0.8
0.8346456692913385
```

맺음말

- 이진 분류를 학습할 수 있는 로지스틱 회귀 모델을 배움
- 데이터 탐색을 통해 이진 분류에 관련성이 높을 입력 데이터를 선정
- 결측 데이터를 적절히 활용한다면 모델의 성능 (일반화) 향상을 기대할 수 있음
- 본 과목에서는 간단한 이진 분류 (로지스틱 회귀) 모델만 소개하였으며 다중 분류 모델 및 고급 학습 모델 기법이 궁금한 학생은 "기계학습 (Machine Learning)" 과목 수강을 추천!