

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

## 데이터셋: 타이타닉 데이터셋 (출처: Kaggle.com)

```
In [2]: train = pd.read_csv('https://bit.ly/fc-ml-titanic')  
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

- PassengerId: 승객 아이디
- Survived: 생존 여부, 1: 생존, 0: 사망
- Pclass: 등급
- Name: 성함
- Sex: 성별
- Age: 나이
- SibSp: 형제, 자매, 배우자 수
- Parch: 부모, 자식 수
- Ticket: 티켓번호
- Fare: 요즘

- Cabin: 좌석번호
- Embarked: 탑승 항구

## 전처리: train / validation 세트 나누기

1. 먼저, feature 와 label을 정의합니다.
2. feature / label을 정의했으면, 적절한 비율로 train / validation set을 나눕니다.

```
In [3]: feature = [  
    'Pclass', 'Sex', 'Age', 'Fare'  
]
```

```
In [4]: label = [  
    'Survived'  
]
```

```
In [5]: train[feature].head()
```

```
Out[5]:   Pclass      Sex   Age     Fare  
          0         male  22.0  7.2500  
          1      female  38.0 71.2833  
          2      female  26.0  7.9250  
          3      female  35.0 53.1000  
          4         male  35.0  8.0500
```

```
In [6]: train[label].head()
```

Out[6]: **Survived**

	Survived
0	0
1	1
2	1
3	1
4	0

In [7]: `from sklearn.model_selection import train_test_split`

- **test\_size**: validation set에 할당할 비율 (20% -> 0.2)
- **shuffle**: 셔플 옵션 (기본 True)
- **random\_state**: 랜덤 시드값

return받는 데이터의 순서가 중요합니다.

In [8]: `x_train, x_valid, y_train, y_valid = train_test_split(train[feature], train[label], test_size=0.2, shuffle=True, random_state=30)`

In [9]: `x_train.shape, y_train.shape`

Out[9]: `((712, 4), (712, 1))`

In [10]: `x_valid.shape, y_valid.shape`

Out[10]: `((179, 4), (179, 1))`

## 전처리: 결측치

In [11]: `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          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
```

결측치를 확인하는 방법은 pandas의 **isnull()**

그리고 합계를 구하는 sum()을 통해 한 눈에 확인할 수 있습니다.

```
In [12]: train.isnull().sum()
```

```
Out[12]: PassengerId      0
Survived        0
Pclass          0
Name            0
Sex             0
Age            177
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          687
Embarked       2
dtype: int64
```

개별 column의 결측치에 대하여 확인하는 방법은 다음과 같습니다.

```
In [13]: train['Age'].isnull().sum()
```

```
Out[13]: np.int64(177)
```

[Impute](#) 도큐먼트

## 1. 수치형 (Numerical Column) 데이터에 대한 결측치 처리

```
In [14]: train['Age'].fillna(0).describe()
```

```
Out[14]: count    891.000000
          mean     23.799293
          std      17.596074
          min      0.000000
          25%     6.000000
          50%    24.000000
          75%    35.000000
          max     80.000000
          Name: Age, dtype: float64
```

```
In [15]: train['Age'].fillna(train['Age'].mean()).describe()
```

```
Out[15]: count    891.000000
          mean     29.699118
          std      13.002015
          min      0.420000
          25%    22.000000
          50%    29.699118
          75%    35.000000
          max     80.000000
          Name: Age, dtype: float64
```

imputer: 2개 이상의 column을 한 번에 처리할 때

```
In [16]: from sklearn.impute import SimpleImputer
```

결측치를 어떤 값으로 채울지는 strategy에 따라 달라집니다.

1. mean : 평균
2. median : 중앙값
3. most\_frequent : 최빈값
4. constant : 고정값(fill\_value="해당 값"을 같이 넣어줘야함)

```
In [17]: imputer = SimpleImputer(strategy='mean')
```

fit() 을 통해 결측치에 대한 학습을 진행합니다.

```
In [18]: imputer.fit(train[['Age', 'Pclass']])
```

```
Out[18]: ▾ SimpleImputer
```

```
SimpleImputer()
```

transform() 은 실제 결측치에 대한 처리를 해주는 함수입니다.

```
In [19]: result = imputer.transform(train[['Age', 'Pclass']])
```

```
In [20]: result
```

```
Out[20]: array([[22.        ,  3.        ],
   [38.        ,  1.        ],
   [26.        ,  3.        ],
   ...,
   [29.69911765,  3.        ],
   [26.        ,  1.        ],
   [32.        ,  3.        ]])
```

```
In [21]: train[['Age', 'Pclass']] = result
```

```
In [22]: train[['Age', 'Pclass']].isnull().sum()
```

```
Out[22]: Age      0  
Pclass     0  
dtype: int64
```

```
In [23]: train[['Age', 'Pclass']].describe()
```

```
Out[23]:          Age      Pclass  
count    891.000000  891.000000  
mean     29.699118   2.308642  
std      13.002015   0.836071  
min      0.420000   1.000000  
25%     22.000000   2.000000  
50%     29.699118   3.000000  
75%     35.000000   3.000000  
max     80.000000   3.000000
```

fit\_transform()은 fit()과 transform()을 한 번에 해주는 함수입니다.

```
In [24]: train = pd.read_csv('https://bit.ly/fc-ml-titanic')
```

```
In [25]: train[['Age', 'Pclass']].isnull().sum()
```

```
Out[25]: Age      177  
Pclass     0  
dtype: int64
```

```
In [26]: imputer = SimpleImputer(strategy='median')
```

```
In [27]: result = imputer.fit_transform(train[['Age', 'Pclass']])
```

```
In [28]: train[['Age', 'Pclass']] = result
```

```
In [29]: train[['Age', 'Pclass']].isnull().sum()
```

```
Out[29]: Age      0  
Pclass    0  
dtype: int64
```

```
In [30]: train[['Age', 'Pclass']].describe()
```

```
Out[30]:          Age      Pclass  
count  891.000000  891.000000  
mean   29.361582  2.308642  
std    13.019697  0.836071  
min    0.420000  1.000000  
25%   22.000000  2.000000  
50%   28.000000  3.000000  
75%   35.000000  3.000000  
max   80.000000  3.000000
```

## 2. (Categorical Column) 데이터에 대한 결측치 처리

```
In [31]: train = pd.read_csv('https://bit.ly/fc-ml-titanic')  
train_copy = train.copy()
```

1개의 column을 처리하는 경우

```
In [32]: train['Embarked'].fillna('S')
```

```
Out[32]: 0      S
         1      C
         2      S
         3      S
         4      S
        ..
        886     S
        887     S
        888     S
        889     C
        890     Q
Name: Embarked, Length: 891, dtype: object
```

Imputer를 사용하는 경우 : 2개 이상의 column을 처리할 때

```
In [33]: imputer = SimpleImputer(strategy='most_frequent')
```

```
In [34]: result = imputer.fit_transform(train[['Embarked', 'Cabin']])
```

```
In [35]: train[['Embarked', 'Cabin']] = result
```

```
In [36]: train[['Embarked', 'Cabin']].isnull().sum()
```

```
Out[36]: Embarked    0
          Cabin       0
          dtype: int64
```

## Label Encoding : 문자(categorical)를 수치(numerical)로 변환

학습을 위해서 모든 문자로된 데이터는 수치로 변환하여야 합니다.

```
In [37]: 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          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        891 non-null    object  
 11  Embarked     891 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
In [38]: def convert(data):
    if data == 'male':
        return 1
    elif data == 'female':
        return 0
```

```
In [39]: train['Sex'].value_counts()
```

```
Out[39]: Sex
male    577
female   314
Name: count, dtype: int64
```

```
In [40]: train['Sex'].apply(convert)
```

```
Out[40]: 0      1
         1      0
         2      0
         3      0
         4      1
         ..
        886     1
        887     0
        888     0
        889     1
        890     1
Name: Sex, Length: 891, dtype: int64
```

```
In [41]: from sklearn.preprocessing import LabelEncoder
```

```
In [42]: le = LabelEncoder()
```

```
In [43]: train['Sex_num'] = le.fit_transform(train['Sex'])
```

```
In [44]: train['Sex_num'].value_counts()
```

```
Out[44]: Sex_num
         1    577
         0    314
Name: count, dtype: int64
```

```
In [45]: le.classes_
```

```
Out[45]: array(['female', 'male'], dtype=object)
```

```
In [46]: le.inverse_transform([0, 1, 1, 0, 0, 1, 1])
```

```
Out[46]: array(['female', 'male', 'male', 'female', 'female', 'male', 'male'],
              dtype=object)
```

Nan 값이 포함되어 있다면, `LabelEncoder` 가 Nan 값도 하나의 Category로 인식하여 라벨링을 진행합니다.

```
In [47]: # Embarked 컬럼의 Nan 값 개수 확인
print(train_copy['Embarked'].isna().sum())
```

```
# df.unique()를 사용하여 총 카테고리가 몇 개인지 확인 가능
print(train_copy['Embarked'].unique())
print(len(train_copy['Embarked'].unique()))
```

```
2
['S' 'C' 'Q' nan]
4
```

```
In [48]: # 1. 결측치 처리를 하지 않고 라벨링을 했을 경우
# 라벨링한 결과를 le_Embarked 컬럼을 새로 만들어서 확인해봅니다.
train_copy["le_Embarked"] = le.fit_transform(train_copy['Embarked'])
# 카테고리가 0, 1, 2, 3으로 총 4개인 것을 확인할 수 있습니다.
print(train_copy["le_Embarked"].unique())
# Embarked 컬럼은 NaN 값이 그대로 있고, 라벨링한 결과가 3인 모습
train_copy[train_copy["Embarked"].isna()]
```

```
[2 0 1 3]
```

```
Out[48]:   PassengerId  Survived  Pclass          Name     Sex   Age  SibSp  Parch  Ticket  Fare Cabin Embarked  le_Embarked
           61         62       1      1  Icard, Miss. Amelie  female  38.0      0      0  113572  80.0    B28      NaN            3
          829         830      1      1  Stone, Mrs. George
                                         Nelson (Martha Evelyn)  female  62.0      0      0  113572  80.0    B28      NaN            3
```

```
In [49]: train_copy[train_copy["le_Embarked"] == 3]
```

```
Out[49]:   PassengerId  Survived  Pclass          Name     Sex   Age  SibSp  Parch  Ticket  Fare Cabin Embarked  le_Embarked
           61         62       1      1  Icard, Miss. Amelie  female  38.0      0      0  113572  80.0    B28      NaN            3
          829         830      1      1  Stone, Mrs. George
                                         Nelson (Martha Evelyn)  female  62.0      0      0  113572  80.0    B28      NaN            3
```

```
In [50]: # 2. 결측치 처리를 하고 나서 라벨링을 했을 경우
train['Embarked'] = train['Embarked'].fillna('S')
train["le_Embarked"] = le.fit_transform(train['Embarked'])
# 카테고리가 0, 1, 2로 총 3개인 것을 확인할 수 있습니다.
train["le_Embarked"].unique()
```

```
Out[50]: array([2, 0, 1])
```

## 원 핫 인코딩 (One Hot Encoding)

```
In [51]: train = pd.read_csv('https://bit.ly/fc-ml-titanic')
```

```
In [52]: train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Embarked 를 살펴봅시다

```
In [53]: train['Embarked'].value_counts()
```

```
Out[53]: Embarked
S    644
C    168
Q     77
Name: count, dtype: int64
```

```
In [54]: train['Embarked'] = train['Embarked'].fillna('S')
```

```
In [55]: train['Embarked'].value_counts()
```

```
Out[55]: Embarked
S    646
C    168
Q     77
Name: count, dtype: int64
```

```
In [56]: train['Embarked_num'] = LabelEncoder().fit_transform(train['Embarked'])
```

```
In [57]: train['Embarked_num'].value_counts()
```

```
Out[57]: Embarked_num
2    646
0    168
1     77
Name: count, dtype: int64
```

Embarked는 탑승 항구의 이니셜을 나타냈습니다.

그런데, 우리는 이전에 배운 내용처럼 `LabelEncoder`를 통해서 수치형으로 변환해주었습니다.

하지만, 이대로 데이터를 기계학습을 시키면, 기계는 데이터 안에서 관계를 학습합니다.

즉, 'S' = 2, 'Q' = 1이라고 되어 있는데, `Q + Q = S` 가 된다라고 학습해버린다는 것이죠.

그렇기 때문에, 독립적인 데이터는 별도의 column으로 분리하고, 각각의 컬럼에 해당 값에만 `True` 나머지는 `False`를 갖습니다. 우리는 이것을 원 핫 인코딩 한다라고 합니다.

```
In [58]: train['Embarked'][:6]
```

```
Out[58]: 0    S
1    C
2    S
3    S
4    S
5    Q
Name: Embarked, dtype: object
```

```
In [59]: train['Embarked_num'][:6]
```

```
Out[59]: 0    2  
1    0  
2    2  
3    2  
4    2  
5    1  
Name: Embarked_num, dtype: int64
```

```
In [60]: pd.get_dummies(train['Embarked_num'][:6]).astype(int)
```

```
Out[60]:   0  1  2  
0  0  0  1  
1  1  0  0  
2  0  0  1  
3  0  0  1  
4  0  0  1  
5  0  1  0
```

```
In [61]: one_hot = pd.get_dummies(train['Embarked_num'][:6])
```

```
In [62]: one_hot.columns = ['C', 'Q', 'S']
```

```
In [63]: one_hot.astype(int)
```

```
Out[63]:   C  Q  S
          0  0  1
          1  1  0
          2  0  1
          3  0  1
          4  0  1
          5  0  0
```

위와 같이 column 을 분리시켜 카테고리형 -> 수치형으로 변환하면서 생기는 수치형 값의 관계를 끊어주어서 독립적인 형태로 바꾸어 줍니다.

원핫인코딩은 카테고리 (계절, 항구, 성별, 종류, 한식/일식/중식...)의 특성을 가지는 column에 대해서 적용 합니다.

## 전처리: Normalize (정규화)

column 간에 다른 min, max 값을 가지는 경우, 정규화를 통해 최소치/ 최대값의 척도를 맞추어 주는 것

- 네이버 영화평점 (0점 ~ 10점): [2, 4, 6, 8, 10]
- 넷플릭스 영화평점 (0점 ~ 5점): [1, 2, 3, 4, 5]

```
In [64]: movie = {'naver': [2, 4, 6, 8, 10],
              'netflix': [1, 2, 3, 4, 5]
             }
```

```
In [65]: movie = pd.DataFrame(data=movie)
movie
```

```
Out[65]:    naver  netflix
```

	naver	netflix
0	2	1
1	4	2
2	6	3
3	8	4
4	10	5

```
In [66]: from sklearn.preprocessing import MinMaxScaler
```

```
In [67]: min_max_scaler = MinMaxScaler()
```

```
In [68]: min_max_movie = min_max_scaler.fit_transform(movie)
```

```
In [69]: pd.DataFrame(min_max_movie, columns=['naver', 'netflix'])
```

```
Out[69]:    naver  netflix
```

	naver	netflix
0	0.00	0.00
1	0.25	0.25
2	0.50	0.50
3	0.75	0.75
4	1.00	1.00

## 표준화 (Standard Scaling)

평균이 0과 표준편차가 1이 되도록 변환

```
In [70]: from sklearn.preprocessing import StandardScaler
```

```
standard_scaler = StandardScaler()
```

샘플데이터를 생성합니다.

```
In [71]: x = np.arange(10)  
# outlier 추가  
x[9] = 1000
```

```
In [72]: x.mean(), x.std()
```

```
Out[72]: (np.float64(103.6), np.float64(298.8100399919654))
```

```
In [73]: scaled = standard_scaler.fit_transform(x.reshape(-1, 1))
```

```
In [74]: x.mean(), x.std()
```

```
Out[74]: (np.float64(103.6), np.float64(298.8100399919654))
```

```
In [75]: scaled.mean(), scaled.std()
```

```
Out[75]: (np.float64(4.4408920985006264e-17), np.float64(1.0))
```

```
In [76]: round(scaled.mean(), 2), scaled.std()
```

```
Out[76]: (np.float64(0.0), np.float64(1.0))
```

```
In [ ]:
```

```
In [ ]:
```