Machine Learning with Python

Life is too short, You need Python

실습 내용

- 머신러닝 모델링을 할 때 자주 사용되는 전처리 방법을 리뷰합니다.
- 익숙하지 않은 방법은 반복 실습을 통해 익숙해져야 합니다.
- 다룰 내용
  - 라이브러리 불러오기
  - 데이터 불러오기
  - 불필요한 변수 제거
  - NaN 조치
  - 가변수화

1.라이브러리, 데이터 불러오기

- 우선 사용할 라이브러리와 분석 대상 데이터를 불러옵니다.

1.1. 라이브러리 불러오기

- 사용할 라이브러리를 불러옵니다.

In [1]:
```
# 라이브러리 불러오기
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

1.2. 데이터 읽어오기

- 분석 대상 데이터를 읽어옵니다.

In [2]:
```
# 데이터 읽어오기
path = "https://raw.githubusercontent.com/jangrae/csv/master/titanic.csv"
titanic = pd.read_csv(path)
```

In [3]:
```
# 상위 데이터 확인
titanic.head()
```

Out[3]:

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

2.불필요한 변수 제거

- Cabin은 77.1%가 NaN이기에 채울 방법이 마땅치 않으니 제거합니다.
- PassengerId, Name, Ticket은 Unique 한 값이므로 제거합니다.
- axis=0는 행, axis=1은 열을 의미함을 기억하세요.

In [4]:
```
# 여러 열 동시 제거
drop_cols = ['Cabin', 'PassengerId', 'Name', 'Ticket']
titanic.drop(drop_cols, axis=1, inplace=True)
```

In [5]:
```
# 확인
titanic.head()
```

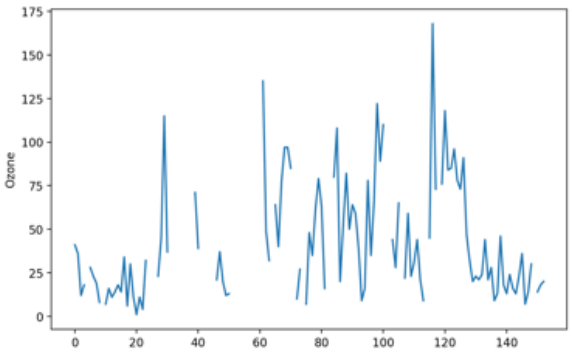
Out[5]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

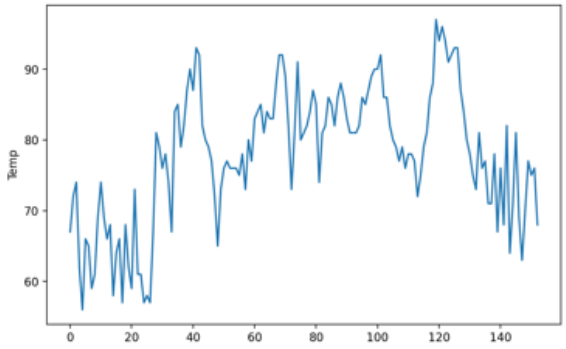
In [6]:
```
# 이후 반복 실습을 위해 원본 보관
titanic_bk = titanic.copy()
```

3.NaN 조치

- NaN 값이 포함되어 있으면 정확한 분석와 예측을 할 수 없으니 이에 대한 처리가 필요합니다.



결측치가 있는 경우



결측치가 없는 경우

3.1. NaN 확인

- NaN 값이 있는지 우선 확인합니다.

In [7]:
```
# 변수들의 NaN 포함 상태 확인
titanic.isna().sum()
```

Out[7]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

3.2. NaN 삭제

- NaN 값이 포함된 행이나 열이 중요하지 않다면 해당 행이나 열을 제거합니다.
- NaN 값이 너무 많이 포함된 경우, 적절히 채울 수 없다면 해당 행과 열을 제거합니다.

3.2.1. 행 제거

- NaN 값이 포함된 행이 그리 많지 않다면 해당 행을 제거합니다.
- 모든 행을 제거하거나 일부 행을 제거할 수 있습니다.

1) 모든 행 제거

	A	B	C	D
0	94500	92100	92200	92300
1	96500	93200	NaN	94300
2	93400	NaN	93400	92100
3	94200	92100	NaN	92400
4	94500	92500	94300	92600
...	...	...	...	...

In [8]:
```
# 처리전 확인
titanic.isna().sum()
```

Out[8]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

In [9]:
```
# NaN이 포함된 모든 행(axis=0) 제거
titanic.dropna(axis=0, inplace=True)

# 확인
titanic.isna().sum()
```

Out[9]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

In [10]:
```
# 데이터 크기 확인
titanic.shape
```

Out[10]:
```
(712, 8)
```

In [11]:
```
# 이후 실습을 위해 원복
titanic = titanic_bk.copy()
```

2) 일부 행 제거

	A	B	C	D
0	94500	92100	92200	92300
1	96500	93200	NaN	94300
2	93400	NaN	93400	92100
3	94200	92100	NaN	92400
4	94500	92500	94300	92600
...	...	...	...	...

In [12]:
```
# 처리전 확인
titanic.isna().sum()
```

Out[12]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

In [13]:
```
# Age 변수에 NaN이 포함된 행 제거
titanic.dropna(subset=['Age'], axis=0, inplace=True)

# 확인
titanic.isna().sum()
```

Out[13]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    2
dtype: int64
```

In [14]:
```
# 이후 실습을 위해 원복
titanic = titanic_bk.copy()
```

3.2.2. 변수 제거

- NaN 값이 포함된 변수가 그리 중요하지 않거나, NaN 값이 너무 많다면 해당 변수를 제거합니다.

	A	B	C	D
0	94500	92100	92200	92300
1	96500	93200	NaN	94300
2	93400	NaN	93400	92100
3	94200	92100	NaN	92400
4	94500	92500	94300	92600
...	...	...	...	...

In [15]:
```
# 처리전 확인
titanic.isna().sum()
```

Out[15]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

In [16]:
```
# NaN 열이 포함된 모든 변수(axis=1) 제거
titanic.dropna(axis=1, inplace=True)

# 확인
titanic.isna().sum()
```

Out[16]:
```
Survived    0
Pclass      0
Sex         0
SibSp       0
Parch       0
Fare        0
dtype: int64
```

In [31]:
```
# 이후 실습을 위해 원복
titanic = titanic_bk.copy()
```

3.3. NaN 채우기

- NaN 값이 포함된 행이나 열을 제거할 수 없다면 특정 값으로 채웁니다.

3.3.1. 특정 값으로 채우기

- 임의의 값을 지정해 NaN 값을 채웁니다.
- 평균값이나 최빈값으로 채우는 경우가 많습니다.

In [32]:
```
# 처리전 확인
titanic.isna().sum()
```

Out[32]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

1) 평균값으로 채우기

In [34]:
```
# Pclass Age 평균 구하기 # type이 맞지 않아 fillna로 채울 수 없음
titanic.groupby(by='Pclass', as_index=False)['Age'].mean()
```

Out[34]:

	Pclass	Age
0	1	38.233441
1	2	29.877630
2	3	25.140620

In [35]:
```
# Pclass Age 평균 구하기
titanic.groupby(by='Pclass', as_index=False)['Age'].transform('mean')
```

Out[35]:
```
0      25.140620
1      38.233441
2      25.140620
3      38.233441
4      25.140620
         ...
886    29.877630
887    38.233441
888    25.140620
889    38.233441
890    25.140620
Name: Age, Length: 891, dtype: float64
```

In [36]:
```
# Age 평균 구하기
#mean_age = titanic['Age'].mean()
mean_age = titanic.groupby(by='Pclass', as_index=False)['Age'].transform('mean')

# NaN을 평균값으로 채우기
titanic['Age'].fillna(mean_age, inplace=True)

# 확인
titanic.isna().sum()
```

Out[36]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    2
dtype: int64
```

2) 최빈값으로 채우기

In [37]:
```
# Embarked 변수 값 확인
titanic['Embarked'].value_counts(dropna=True) # 결측치는 빼고
```

Out[37]:
```
Embarked
S    644
C    168
Q     77
Name: count, dtype: int64
```

In [39]:
```
# Embarked 변수 값 확인
titanic['Embarked'].value_counts(dropna=False) # 결측치 포함
```

Out[39]:
```
Embarked
S    646
C    168
Q     77
Name: count, dtype: int64
```

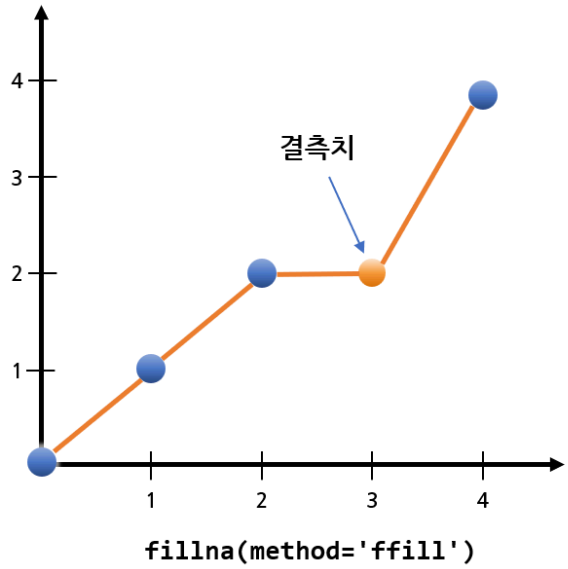
In [43]:
```
#최빈값 확인
mode_titamic = titanic['Embarked'].mode()[0]
```

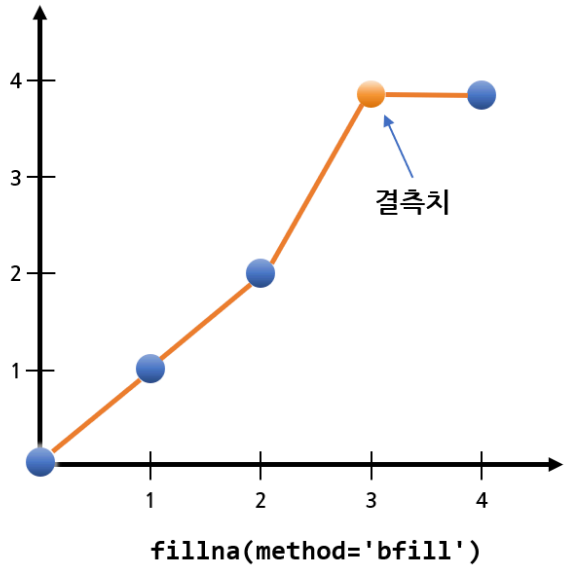
In [44]:
```
# NaN 값을 가장 빈도가 높은 값으로 채우기
titanic['Embarked'].fillna(mode_titamic, inplace=True)

# 확인
titanic.isna().sum()
```

Out[44]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

3.3.2. 앞/뒤 값으로 채우기

- 시계열 데이터인 경우 많이 사용하는 방법입니다.
- method='ffill': 바로 앞의 값으로 채우기
- method='bfill': 바로 뒤의 값으로 채우기





In [45]:
```
# 데이터 불러오기
path = 'https://raw.githubusercontent.com/jangrae/csv/master/airquality.csv'
air = pd.read_csv(path)

# 확인
air.head(10)
```

Out[45]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	NaN	NaN	14.3	56	5	5
5	28.0	NaN	14.9	66	5	6
6	23.0	299.0	8.6	65	5	7
7	19.0	99.0	13.8	59	5	8
8	8.0	19.0	20.1	61	5	9
9	NaN	194.0	8.6	69	5	10

In [46]:
```
# 이후 반복 실습을 위해 원본 보관
air_bk = air.copy()
```

In [47]:
```
# 처리전 확인
air.isna().sum()
```

Out[47]:
```
Ozone      37
Solar.R     7
Wind        0
Temp        0
Month       0
Day         0
dtype: int64
```

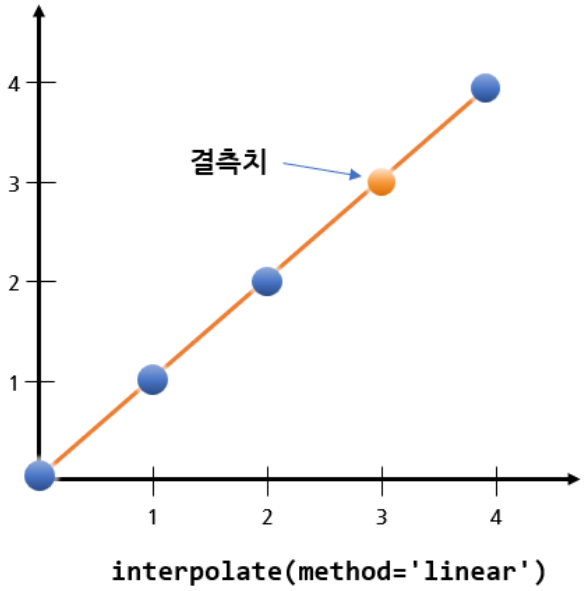
In [48]:
```
# Ozone 변수 NaN 값을 바로 앞의 값으로 채우기
air['Ozone'].fillna(method='ffill', inplace=True)

# Solar.R 변수 NaN 값을 바로 뒤의 값으로 채우기
air['Solar.R'].fillna(method='bfill', inplace=True)

# 확인
air.isna().sum()
```

Out[48]:
```
Ozone      0
Solar.R    0
Wind       0
Temp       0
Month      0
Day        0
dtype: int64
```

In [49]:
```
# 이후 실습을 위해 원복
air = air_bk.copy()
```

3.3.3. 선형 보간법으로 채우기

- interpolate 메서드에 method='linear' 옵션을 지정해 선형 보간법으로 채웁니다.



In [50]:
```
# 처리전 확인
air.isna().sum()
```

Out[50]:
```
Ozone     37
Solar.R    7
Wind       0
Temp       0
Month      0
Day        0
dtype: int64
```

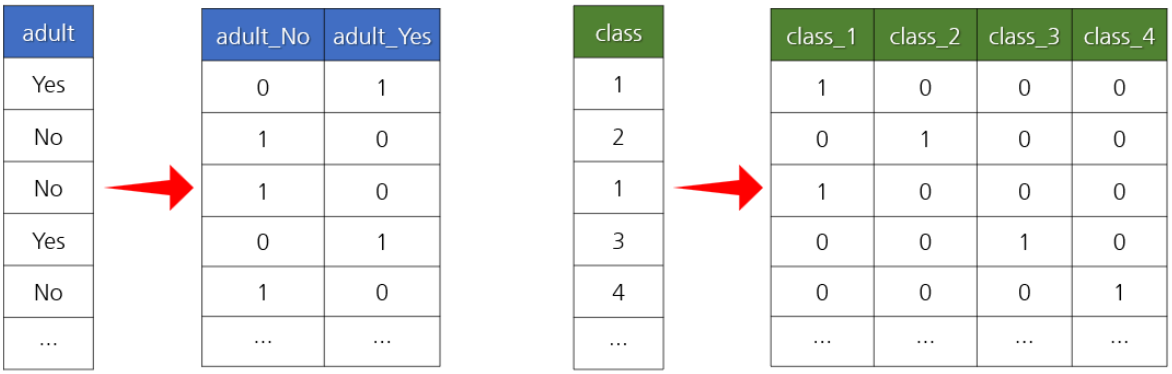
In [51]:
```
# 선형 보간법으로 채우기
air['Ozone'].interpolate(method='linear', inplace=True)
air['Solar.R'].interpolate(method='linear', inplace=True)
```

```
# 확인
air.isna().sum()
```

Out[51]:
```
Ozone      0
Solar.R    0
Wind       0
Temp       0
Month      0
Day        0
dtype: int64
```

4.가변수화

- 범주형 값을 갖는 변수에 대한 One-Hot Encoding을 진행합니다.



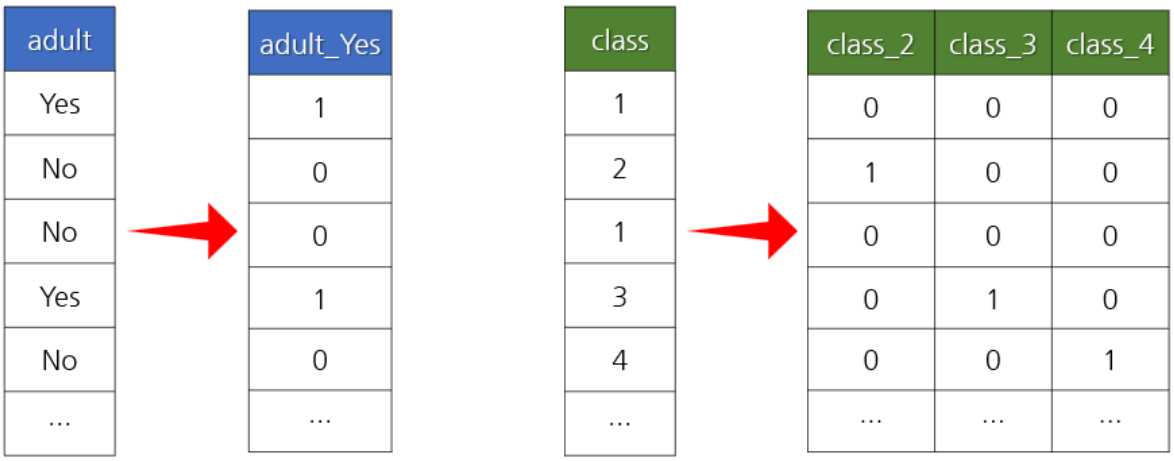
adult
Yes
No
No
Yes
No
...

adult_No	adult_Yes
0	1
1	0
1	0
0	1
1	0
...	...

class
1
2
1
3
4
...

class_1	class_2	class_3	class_4
1	0	0	0
0	1	0	0
1	0	0	0
0	0	1	0
0	0	0	1
...	...	...	...

- 다중공선성 문제를 없애기 위해 drop_first=True 옵션을 지정합니다.



adult
Yes
No
No
Yes
No
...

adult_Yes
1
0
0
1
0
...

class
1
2
1
3
4
...

class_2	class_3	class_4
0	0	0
1	0	0
0	0	0
0	1	0
0	0	1
...	...	...

다중공선성(多重共線性)문제(Multicollinearity)

통계학의 회귀분석에서 독립변수들 간에 강한 상관관계가 나타나는 문제이다. 독립변수들 간에 정확한 선형관계가 존재하는 **완전공선성**의 경우와 독립변수들 간에 높은 선형관계가 존재하는 **다중공선성**으로 구분하기도 한다. 이는 회귀분석의 전제 가정을 위배하는 것이므로 적절한 회귀분석을 위해 해결해야 하는 문제가 된다. (위키백과)

In [52]:
```
# 처리전 확인
titanic.head()
```

Out[52]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

In [53]:
```
# 가변수 대상 변수 식별
dumm_cols = ['Pclass', 'Sex', 'Embarked']

# 가변수화
titanic = pd.get_dummies(titanic, columns=dumm_cols, drop_first=True, dtype=int)

# 확인
titanic.head()
```

Out[53]:

	Survived	Age	SibSp	Parch	Fare	Pclass_2	Pclass_3	Sex_male	Embarked_Q	Embarked_S
0	0	22.0	1	0	7.2500	0	1	1	0	1
1	1	38.0	1	0	71.2833	0	0	0	0	0
2	1	26.0	0	0	7.9250	0	1	0	0	1
3	1	35.0	1	0	53.1000	0	0	0	0	1
4	0	35.0	0	0	8.0500	0	1	1	0	1

In [ ]:
```

```