재직자교육 Python

Table of contents

원활한 환경을 위해 먼저 아래 있는 코드를 입력하고 실행시키고 런타임 > 런타임 다시 시작 을 클릭한다.

```
!sudo apt-get install -y fonts-nanum
!sudo fc-cache -fv
!rm ~/.cache/matplotlib -rf
```

1 파이썬

파이썬이란?

파이썬(Python)은 고수준의 프로그래밍 언어로, 다양한 용도로 사용될 수 있는 범용 프로그래밍 언어입니다.

파이썬 개발환경은 다양하다. 마치 인터넷을 쓸 때 Edge, firefox, safari, chrome 과 같은 브라우저를 쓰는 것처럼 원하는 환경에서 작업하면 된다.

(a) IDLE

(b) PyCharm

(c) Jupyter Notebook

(d) vscode

(e) anaconda

Figure 1: 유명한 개발 환경들

여러가지 개발환경이 있지만 우리는 쥬피터 노트북을 이용하는 방식인 Google 에서 제공하는 colab 을 이용할 것입니다. colab 에 대한 소개는 여기에서 볼 수 있습니다.

1.1 연산자

R에서 배운 연산자와 비교해 보자.

산술 연산자

연산자	파이썬	R	설명
더하기	+	+	두 숫자를 더합니다.
빼기	-	-	두 숫자 중 앞의 숫자에서 뒤의 숫자를 뺍니다.
곱하기	*	*	두 숫자를 곱합니다.
나누기	/	/	두 숫자를 나눕니다.
나머지	%	%%	앞의 숫자를 뒤의 숫자로 나눈 후 나머지를 반환합니다.
몫	//	%/%	두 숫자를 나눈 몫을 반환합니다.
거듭제곱	**	^	첫 번째 숫자를 두 번째 숫자만큼 거듭제곱합니다.

비교 연산자

숫자들의 크기를 비교하는 표현식을 실행하면 참(True)과 거짓(False)이 생성된다. 여기서 유의할 점은 True 와 False 는 참과 거짓을 나타내는 미리 지정된 예약어로서 다른 용도로 사용할 수 없다.

연산자	파이썬	R	설명
크다	>	>	왼쪽 값이 오른쪽 값보다 큰지 비교합니다.
작다	<	<	왼쪽 값이 오른쪽 값보다 작은지 비교합니다.
같음	==	==	두 값이 같은지 비교합니다.
다름	!=	!=	두 값이 다른지 비교합니다.
크거나 같음	>=	>=	왼쪽 값이 오른쪽 값보다 크거나 같은지 비교합니다.
작거나 같음	<=	<=	왼쪽 값이 오른쪽 값보다 작거나 같은지 비교합니다.

논리 연산자

- 논리연산자는 `and`, `or`, `not` 으로 구성되어 있으며 논리식을 표현한다. 논리연산자도 예약어로서 다른 용도로 사용할 수 없다. 예를 들어 논리 연산자는 뒤에 배울 변수의 이름으로 사용할 수 없다.

연산자	파이썬	R	설명
그리고	and, &	&	두 조건이 모두 참인지 확인
또는	or, ‖	‖	두 조건 중 하나라도 참인지 확인
부정	not, !	!	조건이 거짓인지 확인

1.2 라이브러리 (Library)

정의

라이브러리는 특정 기능을 수행하기 위해 미리 작성된 코드의 집합이다.

사용방법

- 파이썬에서는 `import` 를 사용하여 라이브러리를 불러온다.
- R에서는 `library()` 함수를 사용하여 패키지를 불러온다.

2 데이터프레임

컴퓨터를 이용해서 자료를 처리하고 분석할 때 가장 많이 사용하는 형식은 MS EXCEL 의 스프레드 시트와 유사한 `dataframe` 형식이다.

`dataframe` 형식은 아래와 같은 특징을 가지고 있다.

DataFrame

- 행(row, 로우)은 전체 자료에서 하나의 구성 요소(entry, recored, unit)에 대한 데이터들로 이루어 진다.
- 열(column, 컬럼)로 전체 자료에서 하나의 특성을 나타내는 데이터로 이루어 진다. 열은 같은 형식의 자료들로서 구성된다.
- 각 행과 열은 이를 지칭하는 **인덱스(index)** 을 가지고 있으며 인덱스는 숫자 또는 문자로 정의된다.
  - 열을 나타내는 문자로 된 인덱스는 보통 **열이름(column name)** 이라고 부른다.

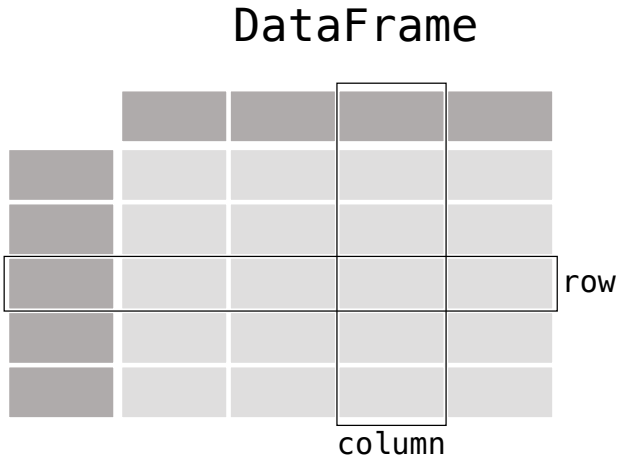


Figure 2: dataframe

데이터프레임은 `pandas` 라이브러리를 통해서 사용할 수 있다. 데이터프레임을 사용하는 경우 초기에 다음과 같이 `pandas` 라이브러리를 불러 주어야 한다.

```
import pandas as pd
```

Series

`Series`은 직접 만들어서 작업을 할 수도 있고 또는 데이터프레임에서 단일 열을 슬라이싱하면 그 자체가 `Series`이다. 열을 선택하려면 대괄호 `[]` 사이에 해당 열을 지정한다.

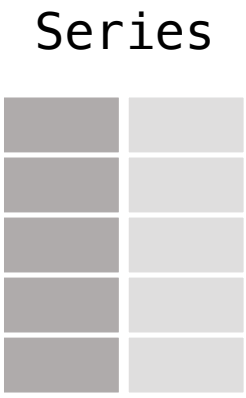


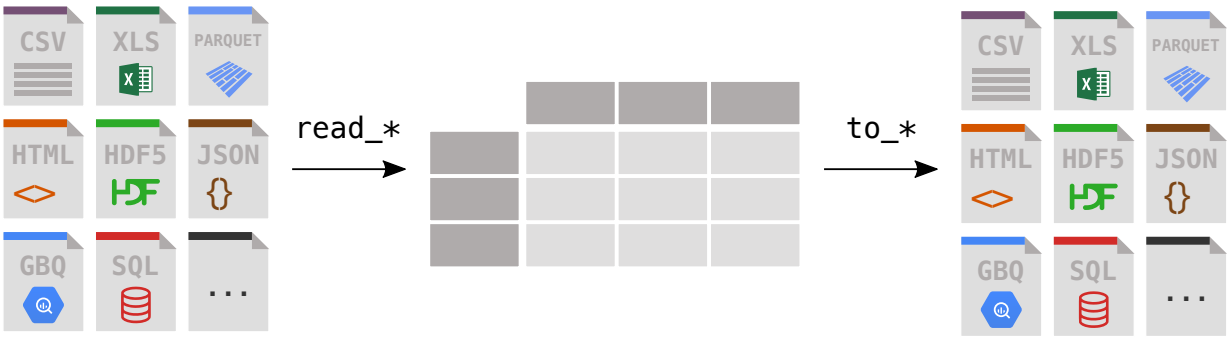
Figure 3: series

참고로 데이터프레임에서 열 또는 행의 일부를 선택하여 추출하는 작업을 슬라이싱(*slicing*) 이라고 한다.

2.1 데이터 가져오기

데이터를 가져와 보자

밑에 그림처럼 파이썬은 아주 다양한 형식의 파일을 읽어올 수 있고 또한 내보낼 수도 있다.



pandas로 데이터 읽어오기

원하는 파일을 불러오기 위해서는 해당 파일의 경로를 입력해야 한다.

```
pd.read_csv("data/readfile.csv")
```

데이터 내보내기

내보내는 것도 마찬가지 이다.

```
pd.to_csv("data/outfile.csv")
```

2.2 간단한 예시를 들어보자.

예시를 들기 위해 `seaborn` 라이브러리에서 `titanic` 데이터셋을 가져오려 한다.

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

밑에 code cell은 교육을 위해 자료를 축약하기 위해 쓰이므로 표현식의 이해는 뒤에서 따로 설명하겠습니다.

```
titanic = titanic.drop(columns=['adult_male','embark_town','class','alive','alone'])
```

Titanic

타이타닉 데이터셋은 유명한 타이타닉 호의 침몰 사건을 바탕으로 한 데이터셋으로, 생존자와 사망자를 분석하기 위해 만들어졌다. 이 데이터는 다양한 변수들을 포함하고 있어, 생존에 영향을 미치는 요인들을 분석하는 데 사용된다.

- survived: 생존 여부 (0 = 사망, 1 = 생존)
- pclass: 티켓 클래스 (1 = 1등석, 2 = 2등석, 3 = 3등석)
- sex: 성별 (male = 남성, female = 여성)
- age: 나이
- sibsp: 함께 탑승한 형제자매 또는 배우자의 수
- parch: 함께 탑승한 부모 또는 자녀의 수
- fare: 운임 요금
- embarked: 탑승한 항구 (C = 셰르부르, Q = 퀸즈타운, S = 사우샘프턴)

- who: 성별과 성인/아동 구분 (man, woman, child)
- deck: 탑승한 갑판 (A, B, C, D, E, F, G, U)

데이터를 가져온 후에는 항상 데이터가 잘 불러왔는 지 확인해야 한다.

head(),tail()

데이터프레임의 처음 8개의 행을 보고 싶으면

```
titanic.head(8)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	deck
0	0	3	male	22.0	1	0	7.2500	S	man	NaN
1	1	1	female	38.0	1	0	71.2833	C	woman	C
2	1	3	female	26.0	0	0	7.9250	S	woman	NaN
3	1	1	female	35.0	1	0	53.1000	S	woman	C
4	0	3	male	35.0	0	0	8.0500	S	man	NaN
5	0	3	male	NaN	0	0	8.4583	Q	man	NaN
6	0	1	male	54.0	0	0	51.8625	S	man	E
7	0	3	male	2.0	3	1	21.0750	S	child	NaN

또한, 마지막 n개의 행을 보고 싶으면 `titanic.tail(n)`을 요청하면 된다.

속성

각 열 데이터의 형식을 pandas가 어떻게 해석했는지 확인하려면 pandas의 `dtypes`메서드를 통해 입력하면 된다.

```
titanic.dtypes
```

```
survived       int64
pclass         int64
sex           object
age          float64
sibsp          int64
parch          int64
fare         float64
embarked      object
who           object
deck        category
dtype: object
```

`titanic`데이터프레임의 데이터 유형은 다음과 같다.

- 정수 `int64`
- 부동 소숫점 `float64`
- 문자열 `object`
- 범주형 `category`

요약

데이터프레임의 기술적 요약을 확인해 보고 싶다면 `info()`를 입력하면 된다.

`info()`의 출력값 중에 각 열의 dtypes도 출력하고 또한 각 행마다 결측값 어느정도 있는지도 확인할 수 있다.

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   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   who       891 non-null    object
 9   deck      203 non-null    category
dtypes: category(1), float64(2), int64(4), object(3)
memory usage: 64.0+ KB
```

이렇게 불러온 데이터가 어떠한 형태를 갖고 있는 지 간단하게 확인하는 방법을 확인했다.

2.3 슬라이싱

데이터프레임을 다루기 위해 원하는 조건의 행 또는 열을 선택해서 추출하는 방법을 다룬다.

- 행 선택
- 열 선택
- 행과 열 둘다 선택

열 선택

데이터프레임에서 다음과 같이 괄호 [] 안에 열이름 또는 열이름으로 구성된 리스트를 넣어서 일부의 열을 선택할 수 있다. 이 경우 열 이름은 문자열이어야 한다.

타이타닉이라는 데이터프레임에서 나이만 뽑아서 확인해보고 싶다면

```
titanic["age"]
```

```
0      22.0
1      38.0
2      26.0
3      35.0
```

Figure 4: 열 선택

```
4      35.0
       ...
886    27.0
887    19.0
888     NaN
889    26.0
890    32.0
Name: age, Length: 891, dtype: float64
```

`titanic`의 데이터프레임에서 `age`에 해당하는 행만 추출하는 것을 의미한다.

또한 `age`는 하나의 행만 선택했기에 `Series`가 된다는 것을 알수 있다.

```
type(titanic["age"])
```

```
pandas.core.series.Series
```

또한, 선택한 행의 데이터 갯수가 몇개인지 확인해보고 싶을 때 `len()`함수를 사용하면 된다.

```
len(titanic["age"])
```

```
891
```

나이와 성별에 관심이 있다면 2개의 행을 선택해야한다. 대괄호 `[]` 안에 리스트를 입력하면 된다.

```
titanic[["age", "sex"]]
```

	age	sex
0	22.0	male
1	38.0	female
2	26.0	female
3	35.0	female
4	35.0	male
...	...	...
886	27.0	male
887	19.0	female

	age	sex
888	NaN	female
889	26.0	male
890	32.0	male

하나의 열만 슬라이싱하는 경우는 데이터프레임 이름 뒤에 마침표를 붙이고 열이름을 붙이면 된다.

```
titanic.sex
```

```
0        male
1      female
2      female
3      female
4        male
       ...
886      male
887    female
888    female
889      male
890      male
Name: sex, Length: 891, dtype: object
```

행 선택

내가 원하는 행만, 즉 자료의 구성 단위의 일부만을 선택해 보자. 데이터프레임 뒤 괄호 [] 안에 비교연산자를 이용한 조건 표현식을 넣어주면 조건이 만족하는 행만 추출된다.

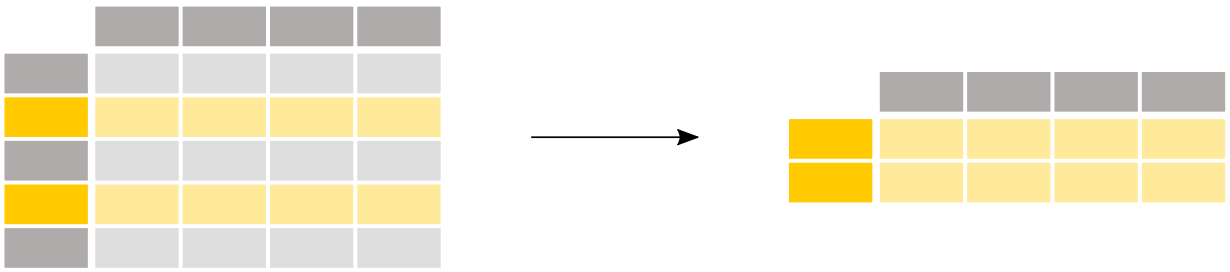


Figure 5: 행 선택

예를 들어 나이가 35세 이상(여기선 초과)인 승객만 본다고 한다면,

```
titanic[titanic["age"] > 35]
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	deck
1	1	1	female	38.0	1	0	71.2833	C	woman	C
6	0	1	male	54.0	0	0	51.8625	S	man	E
11	1	1	female	58.0	0	0	26.5500	S	woman	C
13	0	3	male	39.0	1	5	31.2750	S	man	NaN
15	1	2	female	55.0	0	0	16.0000	S	woman	NaN

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	deck
...	...	...	...	...	...	...	...	...	...	...
865	1	2	female	42.0	0	0	13.0000	S	woman	NaN
871	1	1	female	47.0	1	1	52.5542	S	woman	D
873	0	3	male	47.0	0	0	9.0000	S	man	NaN
879	1	1	female	56.0	0	1	83.1583	C	woman	C
885	0	3	female	39.0	0	5	29.1250	Q	woman	NaN

위에서 대괄호[] 안에 사용한 조건식은 `titanic["age"] > 35` 로서 `titanic["age"]` 의 자료를 각각 35와 과 비교하여 35보다 크면 `True`, 그렇지 않으면 `False` 로 결과를 생성하고 `True` 인 행만 슬라이싱 해준다.

```
titanic["age"] > 35
```

```
0      False
1       True
2      False
3      False
4      False
       ...
886    False
887    False
888    False
889    False
890    False
Name: age, Length: 891, dtype: bool
```

자료에서 선택해야할 경우의 수가 많은 경우 다음과 같은 `.isin()` 메소드를 사용할 수 있다. `.isin` 메서드의 괄호 안에 선택할 문자 또는 숫자를 리스트 형태로 넣어준다.

예를 들어 티켓 클래스가 2, 3등급인 승객들만 슬라이싱 해보자.

```
titanic[titanic["pclass"].isin([2, 3])]
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	deck
0	0	3	male	22.0	1	0	7.2500	S	man	NaN
2	1	3	female	26.0	0	0	7.9250	S	woman	NaN
4	0	3	male	35.0	0	0	8.0500	S	man	NaN
5	0	3	male	NaN	0	0	8.4583	Q	man	NaN
7	0	3	male	2.0	3	1	21.0750	S	child	NaN
...	...	...	...	...	...	...	...	...	...	...
884	0	3	male	25.0	0	0	7.0500	S	man	NaN
885	0	3	female	39.0	0	5	29.1250	Q	woman	NaN
886	0	2	male	27.0	0	0	13.0000	S	man	NaN
888	0	3	female	NaN	1	2	23.4500	S	woman	NaN
890	0	3	male	32.0	0	0	7.7500	Q	man	NaN

정말 2, 3등급인 승객들만 슬라이싱 되었는 지 보려면 밑에 code cell을 입력해 확인해보면 된다. 설명은 뒤에서 다시 언급합니다.

```
titanic[titanic["pclass"].isin([2, 3])]["pclass"].value_counts()
```

```
pclass
3    491
2    184
Name: count, dtype: int64
```

`.isin()`을 쓰지 않고 슬라이싱 하는 방법을 알아보면

```
titanic[(titanic["pclass"] == 2) | (titanic["pclass"] == 3)]
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	deck
0	0	3	male	22.0	1	0	7.2500	S	man	NaN
2	1	3	female	26.0	0	0	7.9250	S	woman	NaN
4	0	3	male	35.0	0	0	8.0500	S	man	NaN
5	0	3	male	NaN	0	0	8.4583	Q	man	NaN
7	0	3	male	2.0	3	1	21.0750	S	child	NaN
...	...	...	...	...	...	...	...	...	...	...
884	0	3	male	25.0	0	0	7.0500	S	man	NaN
885	0	3	female	39.0	0	5	29.1250	Q	woman	NaN
886	0	2	male	27.0	0	0	13.0000	S	man	NaN
888	0	3	female	NaN	1	2	23.4500	S	woman	NaN
890	0	3	male	32.0	0	0	7.7500	Q	man	NaN

데이터프레임에서 행을 조건에 따라서 슬라이싱 할 때 다음과 같은 규칙을 따른다.

두 개 이상의 비교 표현식을 연결할 경우 사용하는 연산자 `|` 와 `&` 는 비트단위연산자(bit-wise operator) 라고 부른다. 데이터프레임에 조건식을 사용하여 슬라이싱 할 경우 논리연산자 `or` 와 `and` 는 사용할 수 없다.

행과 열선택

이제 특정한 열과 행을 동시에 슬라이싱 해보자.

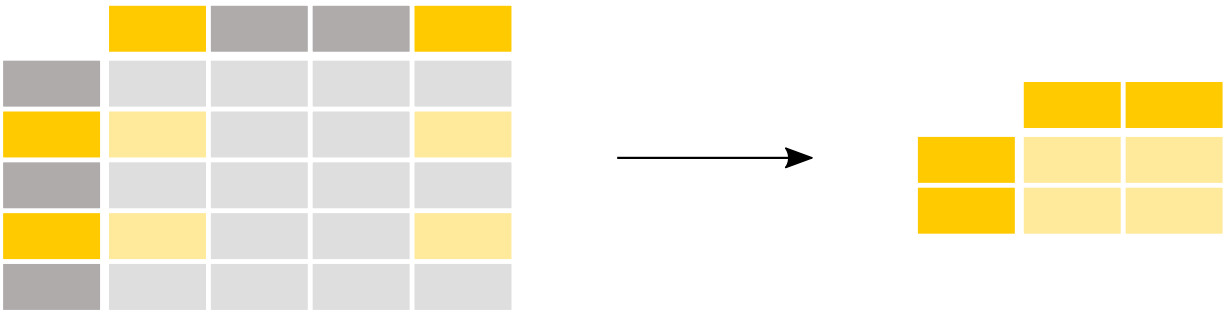


Figure 6: 행과 열 선택

***loc*[]**

만약, 생존한 승객중 성별을 확인하는 슬라이싱은 다음과 같은 표현식를 사용한다.

괄호에서 첫 부분은 **행에 대한 조건을 쓰고** 컴마 **,** 로 분리한 후 **열이름**를 써준다.

R을 같이 사용하다보면 많이 실수하는 경우가 행과 열을 동시에 슬라이싱할 경우 데이터프레임 이름 뒤에 `.loc`를 붙여주지 않아 슬라이싱이 안된다고 하는 경우가 있다. 만약 붙여주지 않으면 오류가 발생한다.

```
titanic.loc[titanic.survived==1, "sex"]
```

```
1      female
2      female
3      female
8      female
9      female
        ...
875    female
879    female
880    female
887    female
889      male
Name: sex, Length: 342, dtype: object
```

iloc[]

데이터프레임의 행과 열에 대한 인덱스를 숫자로 사용하여 슬라이싱하려면 데이터프레임 이름 뒤에 `iloc` 을 붙여주고 괄호 `[]` 안에 행과 열 순서의 위치를 써주면 된다. 이 경우 **파이썬은 모든 위치는 0으로 시작하는데 유의하자**.

하나의 위치가 아닌 여러개의 연속적인 위치는 `x:y` 형태이 범위 형식을 사용하여 지칭한다. 모든 열과 행을 지칭하는 경우 공란으로 넣거나 `:`을 사용한다.

```
titanic.iloc[0,0]
```

```
np.int64(0)
```

```
titanic.iloc[4,3]
```

```
np.float64(35.0)
```

```
titanic.iloc[3,1:3]
```

```
pclass         1
sex       female
Name: 3, dtype: object
```

```
titanic.iloc[:,2]
```

```
0        male
1      female
2      female
3      female
4        male
        ...
886      male
887    female
888    female
889      male
890      male
Name: sex, Length: 891, dtype: object
```

```
titanic.iloc[9:15, 2:5]
```

	sex	age	sibsp
9	female	14.0	1
10	female	4.0	1
11	female	58.0	0
12	male	20.0	0
13	male	39.0	1
14	female	14.0	0

정리를 해보자면

데이터프레임의 인덱스는 행과 열을 지칭할 때 사용한다. 데이터프레임을 슬라이싱할 때 인덱스는 다음과 같은 두 종류의 형식을 사용할 수 있다.

- 라벨(label, 이름)에 기반한 인덱스
  - 데이터프레임의 열이름(column name)으로 지칭하거나 또는 행을 조건식을 이용한 `True`, `False` 로 선택할 때 사용한다.
  - 행과 열을 모두 슬라이싱 하는 경우 `.loc[]`을 사용한다.
- 위치(integer position)에 기반한 인덱스
  - 행과 열 순서의 위치를 나타내는 정수를 사용하여 슬라이싱한다.
  - 모든 위치는 0으로 시작한다.
  - `.iloc[]` 을 사용한다.

2.4 데이터 핸들링

데이터프레임을 내가 원하는 방식으로 핸들링 하는 방법을 보여보자.

정렬

데이터프레임을 볼 때 예를 들어, 운임 요금(fare)을 기준으로 정렬된 데이터셋을 보고 싶다면

```
titanic.sort_values(by='fare', ascending=False)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	deck
258	1	1	female	35.0	0	0	512.3292	C	woman	NaN
737	1	1	male	35.0	0	0	512.3292	C	man	B
679	1	1	male	36.0	0	1	512.3292	C	man	B
88	1	1	female	23.0	3	2	263.0000	S	woman	C
27	0	1	male	19.0	3	2	263.0000	S	man	C
...	...	...	...	...	...	...	...	...	...	...
633	0	1	male	NaN	0	0	0.0000	S	man	NaN
413	0	2	male	NaN	0	0	0.0000	S	man	NaN
822	0	1	male	38.0	0	0	0.0000	S	man	NaN
732	0	2	male	NaN	0	0	0.0000	S	man	NaN
674	0	2	male	NaN	0	0	0.0000	S	man	NaN

내림차순은 `ascending=False`로 해주면 되고

오름차순은 `ascending=TRUE`로 해주면 된다(기본값이기에 넣지 않아도 된다.)

정렬 할때 기준은 꼭 1개일 필요가 없다. 이렇게 정렬 기준을 2개로 지정해도 가능하다.

```
titanic.sort_values(by=['pclass', 'fare'])
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	deck
263	0	1	male	40.0	0	0	0.00	S	man	B
633	0	1	male	NaN	0	0	0.00	S	man	NaN
806	0	1	male	39.0	0	0	0.00	S	man	A
815	0	1	male	NaN	0	0	0.00	S	man	B
822	0	1	male	38.0	0	0	0.00	S	man	NaN
...	...	...	...	...	...	...	...	...	...	...
201	0	3	male	NaN	8	2	69.55	S	man	NaN
324	0	3	male	NaN	8	2	69.55	S	man	NaN
792	0	3	female	NaN	8	2	69.55	S	woman	NaN
846	0	3	male	NaN	8	2	69.55	S	man	NaN
863	0	3	female	NaN	8	2	69.55	S	woman	NaN

문자열 비교 시 대문자와 소문자는 ASCII 값에 따라 정렬된다. 대문자의 ASCII 값이 소문자보다 작기 때문에, 대문자는 소문자보다 먼저 나온다. 예를 들어, 'A'는 'a'보다 앞에 오고, 'B'는 'b'보다 앞에 온다.

만약 데이터프레임에 대문자와 소문자가 혼합된 문자열이 포함되어 있거나 숫자와 문자가 섞여 있다면, 기본적인 정렬 규칙에 따라 정렬된다:

- 대문자와 소문자 혼합: 대문자가 소문자보다 먼저 온다.
- 숫자와 문자 혼합: 숫자가 문자보다 먼저 온다.

열이름 변경

`rename()`을 사용하여 열 이름을 변경해봅니다. 예를 들어, `fare`와 `pclass`를 각각 요금과 등급으로 변경합니다.

```
titanic.rename(columns={'fare': '요금', 'pclass': '등급'})
```

	survived	등급	sex	age	sibsp	parch	요금	embarked	who	deck
0	0	3	male	22.0	1	0	7.2500	S	man	NaN
1	1	1	female	38.0	1	0	71.2833	C	woman	C
2	1	3	female	26.0	0	0	7.9250	S	woman	NaN
3	1	1	female	35.0	1	0	53.1000	S	woman	C
4	0	3	male	35.0	0	0	8.0500	S	man	NaN
...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	man	NaN
887	1	1	female	19.0	0	0	30.0000	S	woman	B
888	0	3	female	NaN	1	2	23.4500	S	woman	NaN
889	1	1	male	26.0	0	0	30.0000	C	man	C
890	0	3	male	32.0	0	0	7.7500	Q	man	NaN

요약

데이터 프레임의 해당 열이 수치형(숫자로 이루어진)이라면 `describe()`메서드는 해당 열의 기본적인 수치적 요약본을 보여준다.

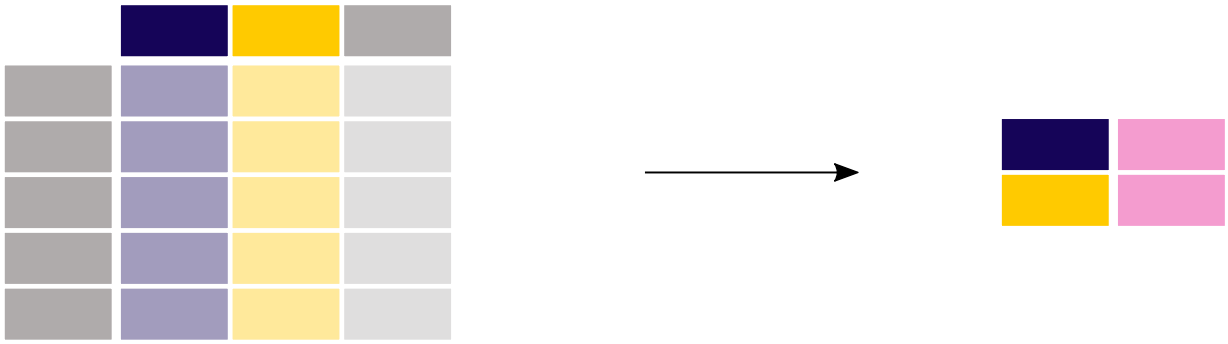


Figure 7: summary

```
titanic[['age','fare']].describe()
```

	age	fare
count	714.000000	891.000000
mean	29.699118	32.204208
std	14.526497	49.693429
min	0.420000	0.000000
25%	20.125000	7.910400
50%	28.000000	14.454200
75%	38.000000	31.000000
max	80.000000	512.329200

데이터를 확인할 때 중위수, 평균, 최대값 등 일부만 확인하고 싶다면 `median()`,`mean()`,`max()`,`count()` 등의 메서드를 이용하면 된다.

그룹화

그룹화(Grouping)는 데이터를 특정 기준에 따라 묶어서 집계(aggregation)하거나, 요약 통계를 계산하는 데 사용되는 기법이다.

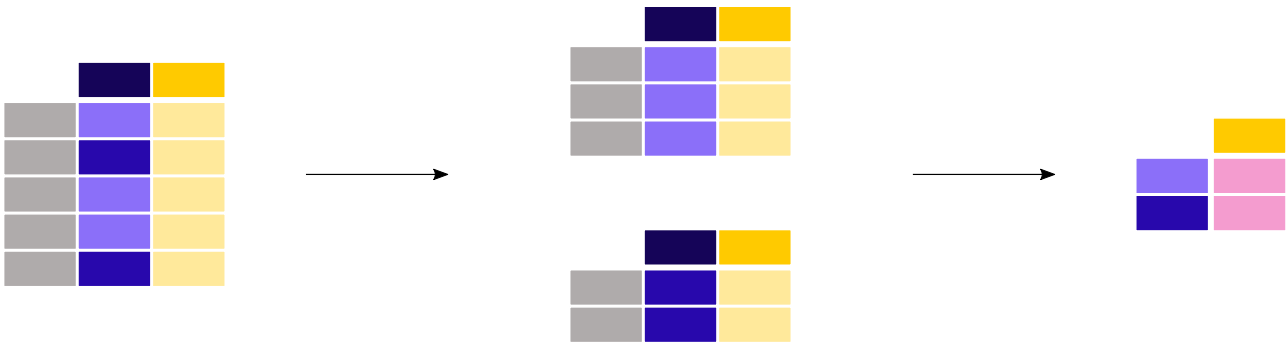


Figure 8: 그룹별 요약

예시를 들어보자면, 승객을 남여로 그룹을 나누고 그 그룹의 나이의 평균이 어떻게 되는 지 보고 싶다면

```
titanic[["sex", "age"]].groupby("sex").mean()
```

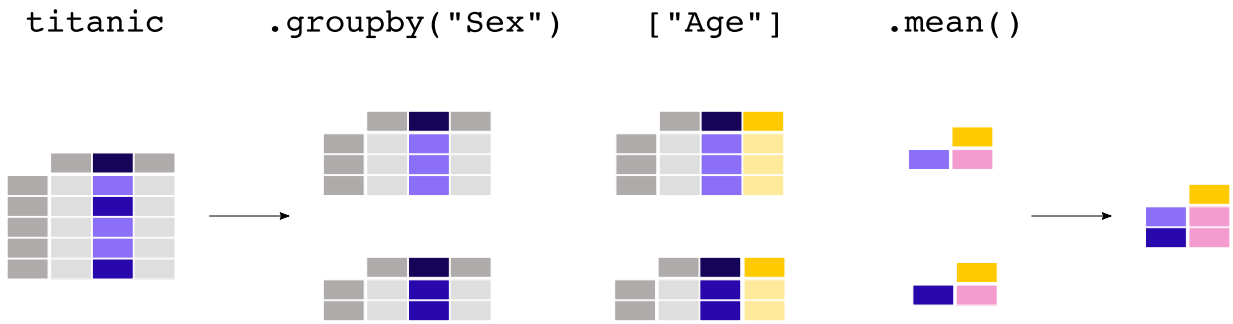
	age
sex	
female	27.915709
male	30.726645

이렇게 데이터프레임은 `groupby()` 메소드를 이용하여 여러 개의 그룹으로 나누어 요약할 수 있다.

```
titanic.groupby("sex")["age"].mean()
```

```
sex
female    27.915709
male      30.726645
Name: age, dtype: float64
```

위 표현식의 메서드가 이뤄지는 과정을 보면



또다른 경우로, 각 성별과 객실 등급 조합에 따른 평균 항공권 운임 가격은 얼마일지 궁금하다면

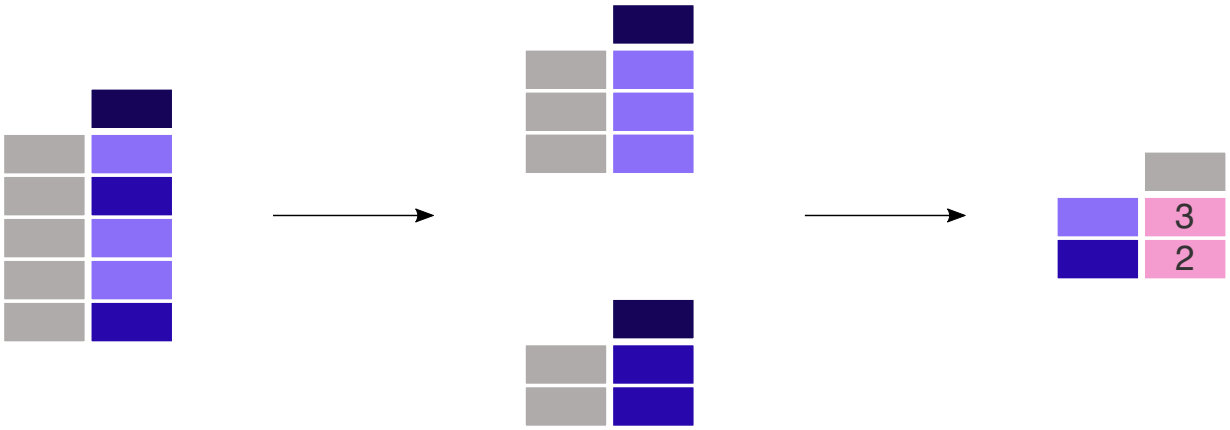
```
titanic.groupby(["sex", "pclass"])["fare"].mean()
```

```
sex     pclass
female  1         106.125798
        2          21.970121
        3          16.118810
male    1          67.226127
        2          19.741782
        3          12.661633
Name: fare, dtype: float64
```

카테고리별 레코드 수 계산

그룹화를 이해하고 나면 다음과 같은 경우도 생각해 볼 수 있다.

각 객실 등급별 승객 수는 얼마일까?




```
titanic["pclass"].value_counts()
```

```
pclass
3    491
1    216
2    184
Name: count, dtype: int64
```

위 표현식 처럼 `value_counts()`메서드를 이용해도 되고 밑에 표현식 처럼 `groupby()`를 이용하는 것도 가능하다.

```
titanic.groupby("pclass")["pclass"].count()
titanic.groupby("who")["who"].count()
```

```
who
child     83
man      537
woman    271
Name: who, dtype: int64
```

흔히 2차원 분할표라고 부르는 2개의 행(변수)를 통해 레코드 수 계산방법을 알아보자.

`crosstab()` 을 사용해서 쉽게 구할 수 있다.

```
pd.crosstab(titanic['who'],titanic['pclass'])
```

pclass who	1	2	3
child	6	19	58
man	119	99	319
woman	91	66	114

2.5 데이터프레임 추가

데이터 프레임을 지금까지는 원하는 행과 열을 슬라이스하거나 원하는 그룹의 빈도수를 살펴보는 것을 확인했다면 이번에는 데이터프레임에 행을 추가해보는 과정을 알아보려 한다.

다음과 데이터프레임이 있다고 하자.

```
# 첫 번째 데이터프레임: 학생 성적 데이터
df_grades = pd.DataFrame({
    'Student_ID': [1, 2, 3, 4],
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Math': [85, 92, 78, 88],
    'English': [90, 85, 95, 80]
})
df_grades
```

	Student_ID	Name	Math	English
0	1	Alice	85	90
1	2	Bob	92	85
2	3	Charlie	78	95
3	4	David	88	80

assign()

학생들의 성적(수학,영어) 총합을 계산하여 데이터프레임에 추가하려고 한다.

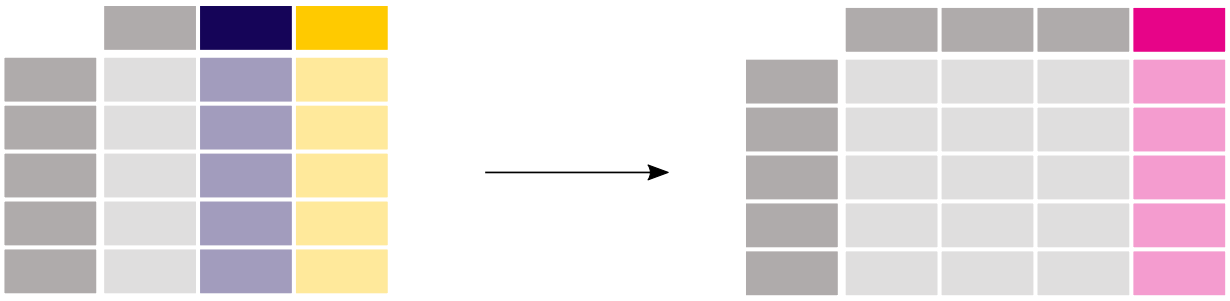


Figure 9: assign

```
df_grades.assign(Total=df_grades['Math'] + df_grades['English'])
```

	Student_ID	Name	Math	English	Total
0	1	Alice	85	90	175
1	2	Bob	92	85	177
2	3	Charlie	78	95	173
3	4	David	88	80	168

concat

기존에 학생들 성적 말고 추가로 새로운 학생들의 성적을 얻었다고 하면 첫번째 데이터프레임에 새로운 데이터프레임 추가하려고 한다.

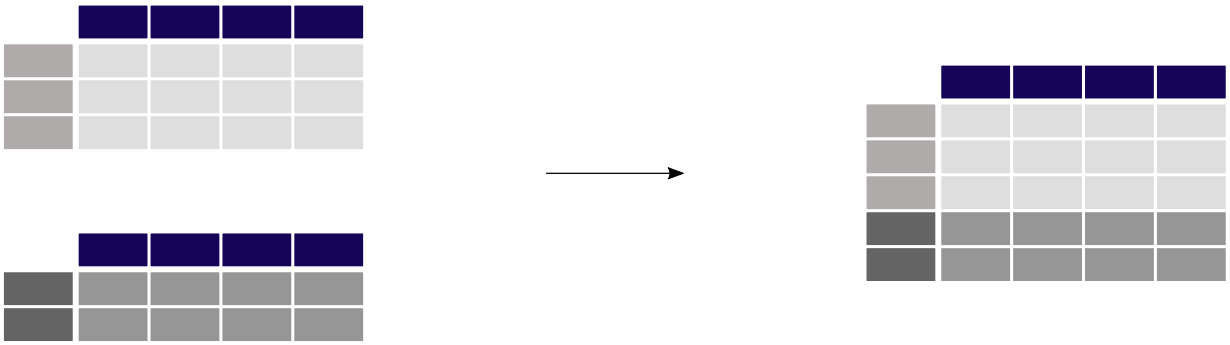


Figure 10: concat_row

새로운 학생들의 성적 데이터프레임이다.

```
# 새로운 학생 성적 데이터
df_grades_2 = pd.DataFrame({
    'Student_ID': [5, 6, 7, 8],
    'Name': ['Eva', 'Frank', 'Grace', 'Hannah'],
    'Math': [91, 87, 84, 79],
    'English': [89, 93, 88, 85]
})
```

`concat` 메서드를 사용하여 `df_grades`과 `df_grades_2`를 행 단위로 이어붙인다. `ignore_index=True`는 새로운 데이터프레임의 인덱스를 재설정합니다.

```
pd.concat([df_grades, df_grades_2], ignore_index=True)
```

	Student_ID	Name	Math	English
0	1	Alice	85	90
1	2	Bob	92	85
2	3	Charlie	78	95
3	4	David	88	80
4	5	Eva	91	89
5	6	Frank	87	93
6	7	Grace	84	88

	Student_ID	Name	Math	English
7	8	Hannah	79	85

`ignore_index`가 `False`이면 어떻게 되는 지 확인해보자.

```
pd.concat([df_grades, df_grades_2], ignore_index=False)
```

	Student_ID	Name	Math	English
0	1	Alice	85	90
1	2	Bob	92	85
2	3	Charlie	78	95
3	4	David	88	80
0	5	Eva	91	89
1	6	Frank	87	93
2	7	Grace	84	88
3	8	Hannah	79	85

2.6 데이터프레임 결합

merge()

데이터를 이용하여 분석을 하는 경우 사용할 자료가 두 개 이상이 경우는 매우 흔한 일이다. **실제로 데이터 분석에서 하나의 자료만 가지고 수행하는 일은 매우 드물다.** 이 절에서는 여러 개의 자료를 사용하는 경우 두 개의 자료를 서로 결합하여 새로운 자료를 만드는 것을 실습한다. 살펴 볼 내용은 다음과 같다.

- 데이터의 결합
- 식별자의 불일치

`df_grades`와 새로운 데이터 `df_clubs`를 사용해 보고자 한다.

```
# 두 번째 데이터프레임: 학생 동아리 활동 데이터
df_clubs = pd.DataFrame({
    'Student_ID': [1, 2, 4, 5],
    'Name': ['Alice', 'Bob', 'David', 'Eva'],
    'Club': ['Science', 'Math', 'Drama', 'Art']
})
df_clubs
```

	Student_ID	Name	Club
0	1	Alice	Science
1	2	Bob	Math
2	4	David	Drama
3	5	Eva	Art

- `merge()` 함수의 첫 번째(**왼쪽**)와 두 번째 인자(**오른쪽**)에는 결합할 데이터프레임의 이름을 넣어준다.

- 선택문 on= 에 두 데이터프레임에 **결합의 기준이 되는 열이름**을 문자열로 지정해준다. 결합의 기준으로 사용되는 열이름은 두 개의 데이터프레임에 모두 존재해야 한다.
- 자료의 결합에 사용되는 공통으로 포함된 열의 내용을 **식별자(key, identifier,..)** 라고 부른다. 이 예제에서 식별자는 사람의 이름이다.

다음 코드의 결과를 먼저 보자.

```
pd.merge(df_grades, df_clubs, on="Name", how='left')
```

	Student_ID_x	Name	Math	English	Student_ID_y	Club
0	1	Alice	85	90	1.0	Science
1	2	Bob	92	85	2.0	Math
2	3	Charlie	78	95	NaN	NaN
3	4	David	88	80	4.0	Drama

결과를 보면 df_grades에서 이름은 있지만, df_clubs에서 df_grades안에 들어 있지 않은 이름의 데이터는 사라진 것을 확인할 수 있다.

Figure 11: merge_left

이러한 식별자가 같지 않는 경우를 **식별자의 불일치**라고 한다.

일반적으로 데이터를 다루다보면 **식별자의 불일치**인 경우가 거의 대부분이기에 우리는 이 부분을 잘 알아두어야 할 필요가 있다.

밑에 표현식처럼 merge()를 쓰는 방법을 나타낸다.

```
pd.merge(left_df, right_df, on="name", how="inner")
```

- how='left' : 식별자는 왼쪽 데이터프레임에만 있는 것으로 선택
- how='right' : 식별자는 오른쪽 데이터프레임에만 있는 것으로 선택
- how='inner' : 식별자는 두 데이터프레임에 공통인 것으로 선택
- how='outer' : 식별자는 두 데이터프레임에 나타난 모든 것으로 선택

식별자를 선택하는 선택표현식 how= 을 지정하지 않으면 자동으로 how='inner' 이 지정된다.

2.7 날짜 계산

데이터프레임을 분석하다보면 날짜 데이터를 다루는 경우가 있다.

날짜는 yyyy-mm-dd, yyyy/mm/dd 또는 ddmmyy 등 여러가지 format으로 존재 하는데 날짜데이터를 파이썬이 날짜라고 인식하게 해주는 과정이 필요하다.

to_datetime

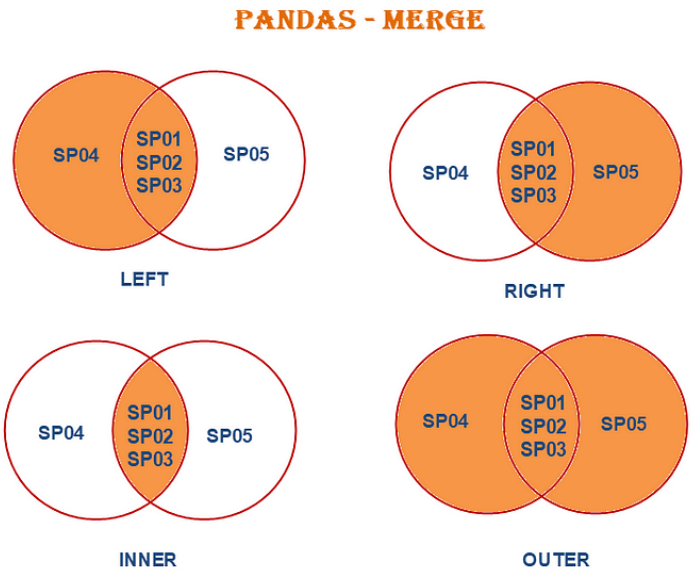


Figure 12: merge

Pandas에서 DateTime은 단일 시점을 나타내는 데이터 유형이다. 특히 주가, 날씨 기록, 경제 지표 등과 같은 시계열 데이터를 처리할 때 유용하다.

간단하게 예제를 살펴보자.

```
time = pd.DataFrame({
    'employee_id': [1, 2, 3],
    'start_date': ['2020-01-15', '2019-06-01', '2018-03-22'],
    'end_date': ['2021-07-10', '2021-06-30', '2020-08-15']
})
time
```

	employee_id	start_date	end_date
0	1	2020-01-15	2021-07-10
1	2	2019-06-01	2021-06-30
2	3	2018-03-22	2020-08-15

`start_date`와 `end_data`의 type이 `object` 즉 문자형이란 걸 알 수 있다.

```
time.dtypes
```

```
employee_id     int64
start_date     object
end_date       object
dtype: object
```

이 문자형 데이터를 날짜 형식으로 변환해주는 과정이 필요하다.

변환하고 나서 날짜 데이터의 형식을 살펴보면 datetime64라는 새로운 형식이 나온 것을 볼 수 있다. 파이썬이 시점을 잘 이해했다는 것을 말한다.

```
time['start_date'] = pd.to_datetime(time['start_date'])
time['end_date'] = pd.to_datetime(time['end_date'])
time.dtypes
```

```
employee_id              int64
start_date     datetime64[ns]
end_date       datetime64[ns]
dtype: object
```

그렇다면, **시작시점과 끝나는 시점을 알고 있으니 걸린기간(일)이 얼마인지 계산하고 싶다고 해보자.**

start_date와 end_data가 날짜라는 것을 파이썬에게 학습했으니 두 날짜(시점)사이의 기간(일)을 계산해서 days_worked 변수에 입력하자.

```
time['days_worked'] = (time['end_date'] - time['start_date']).dt.days
time
```

	employee_id	start_date	end_date	days_worked
0	1	2020-01-15	2021-07-10	542
1	2	2019-06-01	2021-06-30	760
2	3	2018-03-22	2020-08-15	877

.dt.days을 써서 days_worked의 데이터 형식이 int64로 바꾼 것을 알수 있다.

```
time['days_worked'].dtypes
```

```
dtype('int64')
```

그렇다면 **계산한 기간을 개월로 계산할 수 있을까?**

아쉽게도 그러한 형태의 메서드는 pandas에 존재하지 않기 때문에 필요하면 직접 계산식을 만들어야 한다.

한달을 평균적으로 365/12 = 30.42일로 가정하고 나눠주면 쉽게 구할 수 있다.

```
time['months_worked'] = time['days_worked']/30.42
time
```

	employee_id	start_date	end_date	days_worked	months_worked
0	1	2020-01-15	2021-07-10	542	17.817226
1	2	2019-06-01	2021-06-30	760	24.983563
2	3	2018-03-22	2020-08-15	877	28.829717

마지막으로 개월수에 소숫점이 있기에 반올림해서 계산해주는 방법이 있다.

```
time['months_worked'] = (time['days_worked']/30.42).round()
time
```

	employee_id	start_date	end_date	days_worked	months_worked
0	1	2020-01-15	2021-07-10	542	18.0
1	2	2019-06-01	2021-06-30	760	25.0
2	3	2018-03-22	2020-08-15	877	29.0

year, month가 분리되어 있을 경우

R에서는 년,월이 분리되어 있을 때 합치는 함수가 있지만 파이썬에서는 간편하게 쓸 수 있는 대표적인 함수가 없다.

그래서 직접 만들어보는 과정을 실습해 보려 한다.

```
time1 = pd.DataFrame({
    '년': [2020, 2021, 2022],
    '월': [10, 12, 3]
})
time1.dtypes
```

```
년   int64
월   int64
dtype: object
```

년,월 이 정수의 형식을 갖고 있으므로 그대로 합치면 오류가 발생되게 된다.

그렇기 때문에,

1. 년,월을 문자형태로 변환해 주고
2. 년-월로 합쳐준 다음에
3. 날짜 데이터로 변환해주는

과정이 필요하다.

밑에 표현식을 보면서 이해해 보자.

```
time1['년월'] = time1['년'].astype(str) + '-' + time1['월'].astype(str)
time1['년월'] = pd.to_datetime(time1['년월'])
time1
```

	년	월	년월
0	2020	10	2020-10-01
1	2021	12	2021-12-01
2	2022	3	2022-03-01

3 산재보험패널 데이터 실습

이제 실제 데이터를 파이썬을 통한 실습을 해본다.

Google colab에 파일을 업로드해야하는 과정을 다음과 같은 표현식이 필요하다.

산재보험패널조사는 산재근로자의 의료적 치료 종결(요양종결) 이후의 상황을 종합적으로 진단하고, 산재보험서비스를 평가할 수 있는 객관적인 자료로 제 2차 코호트 조사가 2017년도 요양종결 산재근로자를 대상으로 2018년도에서 2022년도까지 이루어졌다.

```
from google.colab import files
uploaded = files.upload()

import pandas as pd
df = pd.read_csv('pswci2_05_long.csv', encoding = 'cp949')
```

3.1 전처리

데이터 전처리(data preprocessing)는 분석에 앞서 데이터를 정제하고 변환하는 과정을 말한다.

어떤 목적인지에 따라 전처리를 하는 과정은 다르다.

1~5차 조사년도 중 1,2차 조사년도에서 재취업자의 고용유지기간 및 고용유지여부 파악

```
import pandas as pd
import numpy as np
```

실제 Raw 데이터의 경우는 데이터를 읽고 구성을 보는 것이 어렵다.

사실상 `head()`메서드를 이용해서 데이터가 잘 읽혔는 지 확인만 하고 필요한 변수(열)만 CODEBOOK 엑셀파일에서 확인하고 직접 슬라이싱 하는 걸 추천한다.

```
df.head(3)
```

	pid	p	wave	nonresponse	workperiod14	accident	injurytype	injurypart	con16	acc1	...	I00900
0	1	2	5	2.0	7	1	1	9	6	2014	...	NaN
1	1	2	4	1.0	7	1	1	9	6	2014	...	NaN
2	1	2	3	6.0	7	1	1	9	6	2014	...	NaN

1-1. 재취업자를 조사 1차년도(2018)를 기준으로 취업이 되었는 지 확인

조사 1차년도에 기록된 재취업자 중에 재취업을 더 먼 과거에서부터 2018년도 안에 했다면

1. `wave = 1` 1차년도만 확인해봐야 하고
2. `E2002001,E2002002` 가 NaN이 아니여야 한다.

보다 정확히는 E2002002가 입사한 날짜의 월 데이터인데 응답자가 정확히 몇월에 취업을 했는 지 모르기에 공란 또는 다른 옵션의 답변을 했을 경우가 있다.

그럼, 우리는 그러한 답변을 내놓은 응답자의 응답은 다루기 어렵기에 무시하고 슬라이싱 한다.

```
# wave가 1이고 E2002001가 NaN이 아닌 행 선택
condition_wave_1_not_nan = (df['wave'] == 1) & (df['E2002002'].notna())
selected_wave_1_not_nan = df[condition_wave_1_not_nan]
```

1-2. 1번에서 만족하는 응답자 pid(personal id)만 선택

```
# 위 조건에 해당하는 pid 값들 추출
selected_pids = selected_wave_1_not_nan['pid']
```

1-3. 2번에서 만족한 pid의 1차년도,2차년도 (wave=1,2) 만 슬라이싱

1차년도에 재취업을 한 응답자중에 고용유지를 계속 이어가고 있었다면

2차년도에 조사했을 때 퇴사를 한 기록이 없어야 한다.

즉, wave가 1,2 그리고 해당 pid만 추출하면 된다.

```
# 조건을 만족하는 pid에서 wave가 1과 2인 행 선택
condition_wave_1_and_2 = df['pid'].isin(selected_pids) & df['wave'].isin([1, 2])
result = df[condition_wave_1_and_2]
```

1-4. 고용유지를 한 응답자와 그렇지 않은 응답자를 0과 1로 구분

F001002, F001003은 퇴사한 시기의 년,월이다.

슬라이싱한 데이터프레임으로 F001003가 빈칸(NaN)이면 고용을 유지 한걸로 인지해 1, 그렇지 않으면 0으로 해서 고용유지 변수를 만든다.

```
# 더미변수 만들기 유지되면 1 아니면 0
result.loc[:,'고용유지'] = np.where(result['F001003'].isna(), 1, 0)
```

아직 슬라이싱을 완성 하진 않았지만, 고용유지변수를 잘 만들었단 걸 확인할 수 있다.

```
result[['pid','고용유지']]
```

	pid	고용유지
53	11	1
54	11	0
73	15	1
74	15	0
78	16	1
...	...	...
16414	3283	0
16438	3288	1
16439	3288	0
16443	3289	0
16444	3289	0

2-1. 고용유지기간(개월) 생성

조사 1차년도에 재취업을 한 응답자들만 슬라이싱한 result 데이터프레임을 갖고 그 재취업응답자들의 고용유지기간을 계산해 보려고 한다.

우선, 2차년도에 퇴사한 날짜 즉, wave가 2이면서 F001002, F001003가 빈칸(NaN) 이면 날짜계산을 할 수가 없다.

그렇기에, 조사 2차년도에 F001002, F001003가 빈칸(NaN) 이면 2019년 11월로 대체해서 입력해준다.(패널조사 기간 11월 중)

```
#NaN 부분 2019년 11월로 대체
result.loc[result['F001003'].isna(), ['F001002','F001003']] = [2019, 11]
```

2-2. wave가 1과 2를 분류하기

```
result[['pid','wave','E2002001','E2002002','F001002','F001003','고용유지']]
```

	pid	wave	E2002001	E2002002	F001002	F001003	고용유지
53	11	2	NaN	NaN	2019.0	11.0	1
54	11	1	2018.0	1.0	2013.0	11.0	0
73	15	2	NaN	NaN	2019.0	11.0	1
74	15	1	2018.0	8.0	2017.0	10.0	0
78	16	2	NaN	NaN	2019.0	11.0	1
...	...	...	...	...	...	...	...
16414	3283	1	2017.0	12.0	2017.0	1.0	0
16438	3288	2	NaN	NaN	2019.0	11.0	1
16439	3288	1	2017.0	12.0	2016.0	12.0	0
16443	3289	2	2019.0	3.0	2018.0	11.0	0
16444	3289	1	2018.0	3.0	2016.0	11.0	0

위 데이터프레임에서 다뤄야 할 재취업날짜는 wave가 1에 있고 퇴사날짜는 wave가 2에 있기에 따로 나눠고 다시 합쳐주는 과정이 필요하다.

```
#wave가 1인 df1, wave가 2인 df2
df1 = result[result.wave == 1]
df2 = result[result.wave == 2]
```

2-3. 년월 날짜로 변환하기

df1 과 df2를 합치기전에 년과 월을 먼저 합치자. 물론 합치는 것에 먼저는 없다.

직전에 다뤘던 년월 합치는 방식을 배웠으니 적용해보자.

1. 부동소숫점을 문자로 변환하기

E2002001,E2002002 변수를 보면, 바로astype(str) 변환을 해주어야 하지만 여기서는 날짜로 변환할 때 to_datetime이 인식을 제대로 하지 못하는 경우가 발생해 먼저 astype(int)으로 정수로 변경해줘서 소숫점을 제거하고 그 후에 astype(str) 변환을 해주면 된다.

```
df1.E2002001 = df1['E2002001'].astype(int).astype(str)
df1.E2002002 = df1['E2002002'].astype(int).astype(str)
```

2. 년-월 로 합치기

E2002001,E2002002를 합쳐서 재취업_입사일에 입력한다.

```
df1['재취업_입사일'] = df1['E2002001'] + '-' + df1['E2002002']
```

3. 날짜 형식으로 변환하기

```
df1['재취업_입사일'] = pd.to_datetime(df1['재취업_입사일'])
```

4. df2도 마찬가지로 진행

```
df2['F001002'] = df2['F001002'].astype(int).astype(str)
df2['F001003'] = df2['F001003'].astype(int).astype(str)
df2['재취업_퇴사일'] = df2['F001002'] + '-' + df2['F001003']
df2['재취업_퇴사일'] = pd.to_datetime(df2['재취업_퇴사일'])
```

2-4. df merge하기

데이터프레임 결합 부분에 있던 내용이다.

df2의 일부분을 df1에 merge해서 고용유지,재취업_퇴사일 변수를 입력한다.

식별자(Key)는 pid 이다.

```
df3 = pd.merge(df1,df2[['pid','고용유지','재취업_퇴사일']], on='pid', how='outer')
df3[['고용유지_y','재취업_입사일','재취업_퇴사일']]
```

	고용유지_y	재취업_입사일	재취업_퇴사일
0	1	2018-01-01	2019-11-01
1	1	2018-08-01	2019-11-01
2	1	2017-06-01	2019-11-01
3	1	2018-06-01	2019-11-01
4	1	2017-06-01	2019-11-01
…	…	…	…
973	1	2017-11-01	2019-11-01
974	1	2018-03-01	2019-11-01
975	0	2017-12-01	2019-06-01
976	1	2017-12-01	2019-11-01
977	0	2018-03-01	2018-11-01

2-5. '재취업_입사일','재취업_퇴사일'을 통한 근속기간(개월) 생성

```
df3['근속기간(일)'] = (df3['재취업_퇴사일'] - df3['재취업_입사일']).dt.days
df3['근속기간(개월)'] = (df3['근속기간(일)']/30.42).round()
df3[['pid','재취업_입사일','재취업_퇴사일','근속기간(일)','근속기간(개월)']]
```

	pid	재취업_입사일	재취업_퇴사일	근속기간(일)	근속기간(개월)
0	11	2018-01-01	2019-11-01	669	22.0
1	15	2018-08-01	2019-11-01	457	15.0
2	16	2017-06-01	2019-11-01	883	29.0
3	98	2018-06-01	2019-11-01	518	17.0
4	104	2017-06-01	2019-11-01	883	29.0
...	...	...	...	...	...
973	3279	2017-11-01	2019-11-01	730	24.0
974	3281	2018-03-01	2019-11-01	610	20.0
975	3283	2017-12-01	2019-06-01	547	18.0
976	3288	2017-12-01	2019-11-01	700	23.0
977	3289	2018-03-01	2018-11-01	245	8.0

재취업자의 고용유지여부와 고용기간을 계산해보는 과정을 돌려봤다.

마지막전처리(선택)

마지막으로 많은 설명변수중에 NaN이 존재하는 변수(열)을 제거해주는 과정을 돌려보자.

```
# 각 열에 NaN 값이 없는지 여부를 확인
non_nan_columns = df3.columns[df3.isna().sum() == 0]

df4 = df3[non_nan_columns]
df4.isnull().values.any() # NaN 값이 있으면 True, 없으면 False
```

```
np.False_
```

이렇게 목적을 둔 전처리를 진행하면 이제 우리는 데이터분석이라 불리는 과정으로 넘어갈 수 있다.

그러나 남은 시간에 통계적지식을 배우고 분석을 하는 이른바 모델링을 하는 것은 불가능하기에 데이터를 보다 구체적으로 탐색하는 과정을 가져보자.

4 시각화

앞에서 데이터프레임의 기초를 배우고 재취업자의 고용유지 여부 및 기간을 전처리하는 것을 다뤄보는 시간을 가졌다.

하지만, 데이터프레임안에 어떠한 변수들이 있는 지 확인하고 비교해 보는 시간을 가져보지 않았기에 다뤄보자.

그 전에, 코랩에서는 기본적으로 한국어를 지원하지 않기 때문에 한국어 폰트를 설치를 해줘야 한다. 이를 설치하기 위해 아래의 코드를 실행시킨다.

```
!sudo apt-get install -y fonts-nanum
!sudo fc-cache -fv
!rm ~/.cache/matplotlib -rf
```

위 코드는 코랩에 나눔폰트를 설치해주는 코드이다. 이를 실행하고, 상단에 런타임 > 런타임 다시 시작 을 클릭한다.

그 후, 파이썬에서 나눔폰트를 사용한다는 의미로 `plt.rcParams`를 이용한다.

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

plt.rcParams['font.family'] = 'NanumBarunGothic'
```

4.1 Seaborn

Seaborn은 파이썬에서 통계적 데이터 시각화를 위한 라이브러리다. matplotlib을 기반으로 하여 더 간편하고 미려한 그래프를 그릴 수 있게 해준다.

```
import seaborn as sns
pd.set_option('mode.chained_assignment', None)
sns.set_theme(style = 'whitegrid')
```

새로운 마음으로 전처리한 df4를 다시 df로 할당하고 시작해 본다.

또한 시각화에 쓰일 변수들만 선택해 슬라이싱 한다.

```
df = df4
cols = {'col' : ['pid', '고용유지_y', '재취업_입사일', '재취업_퇴사일', '근속기간(개월)',
                 'E2001002', 'E2007003', 'E2008001', 'E2019001', 'gender',
                 'age4', 'accident', 'injurypart', 'con16', 'area6'],
        'name' : ['응답자ID', '고용유지_여부', '재취업_입사일', '재취업_퇴사일', '근속기간(개월)',
                  '산업분류', '직업분류', '종사상지위', '월평균임금', '성별',
                  '연령대', '사고여부', '상해부위', '요양기간', '권역']}
df = df[cols['col']]
df.columns = cols['name']
df
```

	응답자ID	고용유지_여부	재취업_입사일	재취업_퇴사일	근속기간(개월)	산업분류	직업분류	종사상지위	월평
0	11	1	2018-01-01	2019-11-01	22.0	15.0	3.0	2.0	150
1	15	1	2018-08-01	2019-11-01	15.0	3.0	2.0	2.0	60
2	16	1	2017-06-01	2019-11-01	29.0	19.0	2.0	1.0	61

	응답자ID	고용유지_여부	재취업_입사일	재취업_퇴사일	근속기간(개월)	산업분류	직업분류	종사상지위	월평
3	98	1	2018-06-01	2019-11-01	17.0	6.0	8.0	3.0	250
4	104	1	2017-06-01	2019-11-01	29.0	3.0	9.0	1.0	430
...	...	...	...	...	...	...	...	...	...
973	3279	1	2017-11-01	2019-11-01	24.0	3.0	9.0	1.0	170
974	3281	1	2018-03-01	2019-11-01	20.0	15.0	9.0	2.0	27.
975	3283	0	2017-12-01	2019-06-01	18.0	17.0	4.0	3.0	170
976	3288	1	2017-12-01	2019-11-01	23.0	9.0	9.0	1.0	150
977	3289	0	2018-03-01	2018-11-01	8.0	15.0	9.0	2.0	27.

또한 원활하게 시각화를 다루기 위해 일부 변수들의 형식을 변경한다.

```
df['산업분류'] = df['산업분류'].astype(int)
df['직업분류'] = df['직업분류'].astype(int)
df['종사상지위'] = df['종사상지위'].astype(int)
df['연령대'] = df['연령대'].astype(int)
df['권역'] = df['권역'].astype(int)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 978 entries, 0 to 977
Data columns (total 15 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   응답자ID     978 non-null    int64
 1   고용유지_여부  978 non-null    int64
 2   재취업_입사일  978 non-null    datetime64[ns]
 3   재취업_퇴사일  978 non-null    datetime64[ns]
 4   근속기간(개월)  978 non-null    float64
 5   산업분류    978 non-null    int64
 6   직업분류    978 non-null    int64
 7   종사상지위   978 non-null    int64
 8   월평균임금   978 non-null    float64
 9   성별     978 non-null    int64
 10  연령대    978 non-null    int64
 11  사고여부    978 non-null    int64
 12  상해부위    978 non-null    int64
 13  요양기간    978 non-null    int64
 14  권역     978 non-null    int64
dtypes: datetime64[ns](2), float64(2), int64(11)
memory usage: 114.7 KB
```

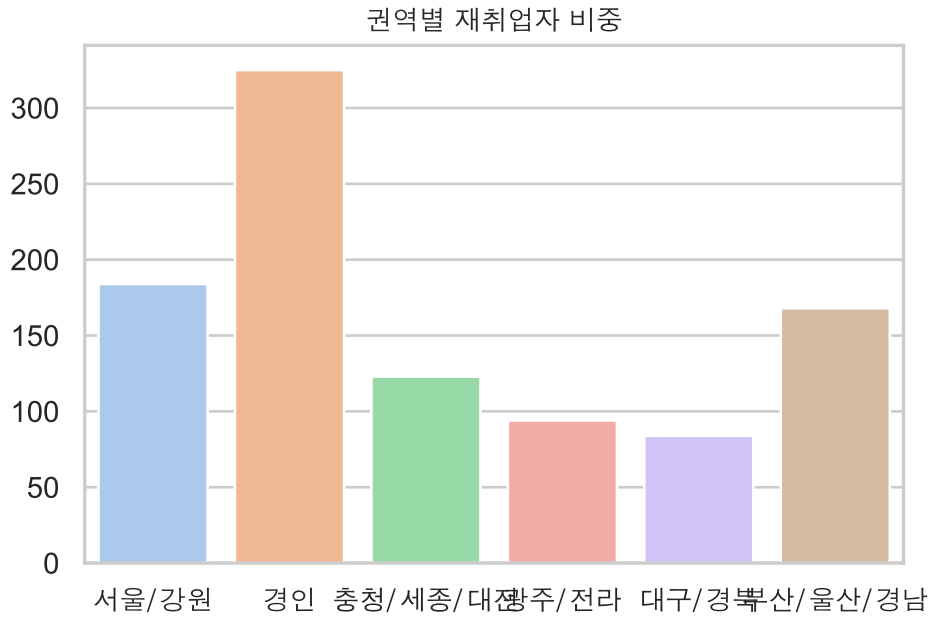
4.2 Countplot

가장 간단한 plot부터 다뤄보려고 한다.

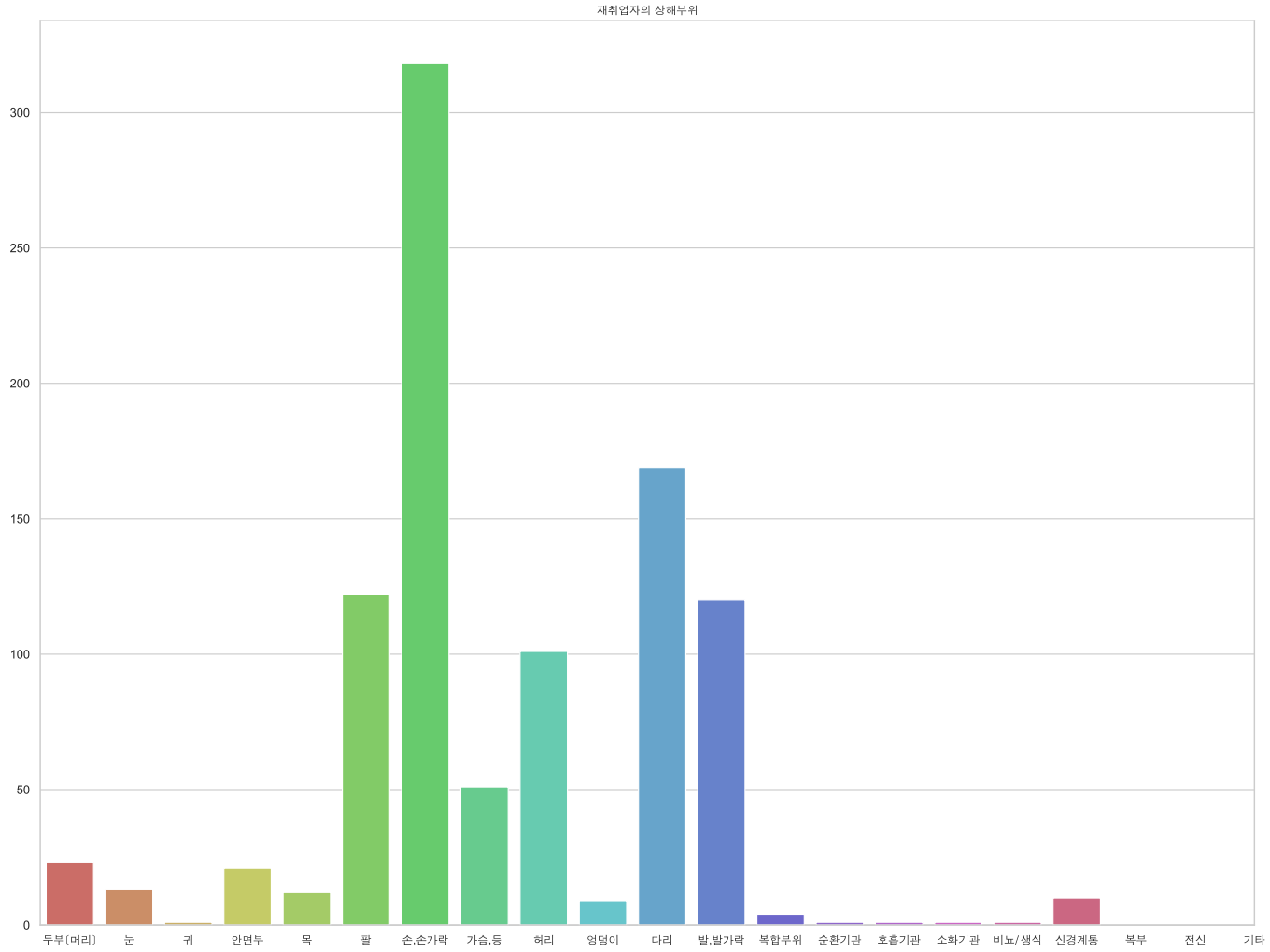
한 변수를 지정해서 그 변수의 범주별 빈도수를 확인해 보는 plot이다.

- `x` : x축에 사용할 변수 이름을 지정 **[필수]**
- `data` : 그래프를 그릴 데이터프레임을 지정 **[필수]**
- `hue` : 막대의 색상을 그룹화할 변수 이름을 지정
- `order` : x축에 사용할 범주의 순서를 지정
- `orient` : 막대의 방향을 지정 ('v' 또는 'h', 기본값: 'v')
- `palette` : 색상 팔레트를 지정('pastel', 'dark', 'colorblind', 'Blues', 'BuGn', 'cubehelix', 'husl', 'hls', 등등)

```
ax = sns.countplot(x='권역', data = df, palette = 'pastel')
ax.set_title('권역별 재취업자 비중') # 그래프 제목 추가
ax.set_xticks(range(0,6,1),
              labels = ['서울/강원', '경인', '충청/세종/대전', '광주/전라',
                        '대구/경북', '부산/울산/경남']) # X축 눈금 변경
plt.show()
```



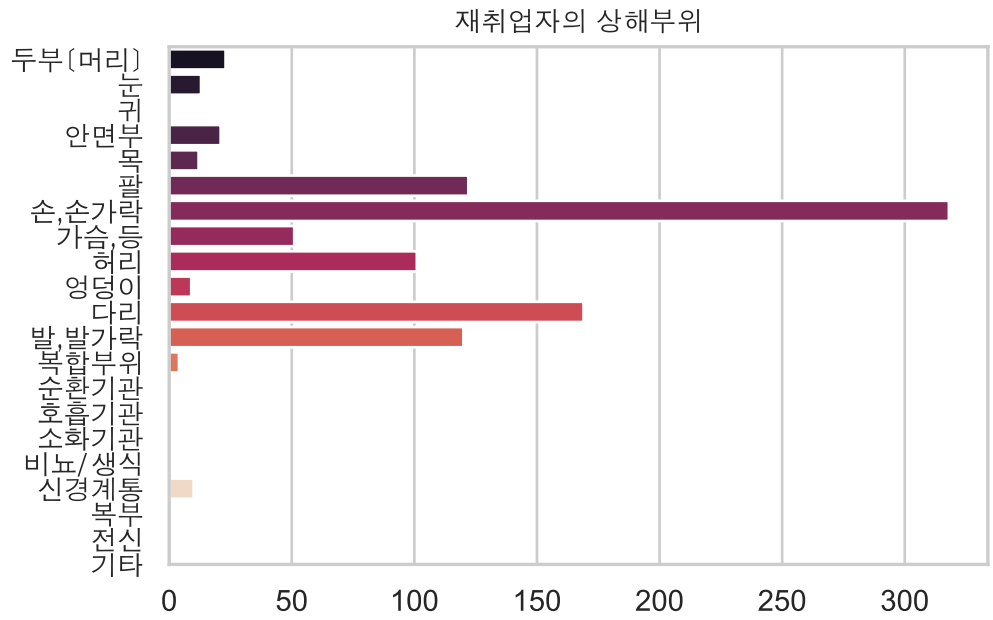

```
fig = plt.figure(figsize=(20, 15))
ax = sns.countplot(x='상해부위', data = df, palette='hls')
ax.set_title('재취업자의 상해부위',fontsize=20) # 그래프 제목 추가
ax.set_xticks(range(0, 21, 1),
              labels = ['두부[머리]', '눈', '귀', '안면부', '목', '팔', '손,손가락','가슴,등',
                        '허리','엉덩이','다리','발,발가락','복합부위','순환기관','호흡기관',
                        '소화기관','비뇨/생식','신경계통','복부','전신','기타']) # X축 눈금 변경


plt.show()
```



그래프의 위치를 수직에서 수평으로 바꾸는 방법도 있다.

`orient`와 `x`에 포함되는 부분들을 `y`로 변환해주면 된다.

```
ax = sns.countplot(y='상해부위', data = df, orient='h', palette='rocket')
ax.set_title('재취업자의 상해부위') # 그래프 제목 추가
ax.set_yticks(range(0, 21, 1),
              labels = ['두부[머리]', '눈', '귀', '안면부', '목', '팔', '손,손가락','가슴,등',
                        '허리','엉덩이','다리','발,발가락','복합부위','순환기관','호흡기관',
                        '소화기관','비뇨/생식','신경계통','복부','전신','기타']) # X축 눈금 변경

plt.show()
```

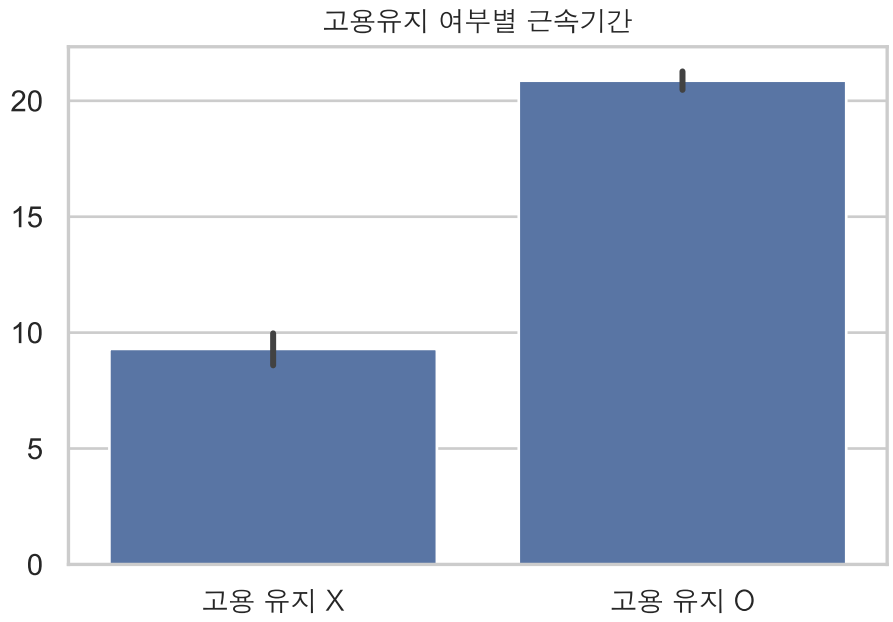


4.3 Barplot

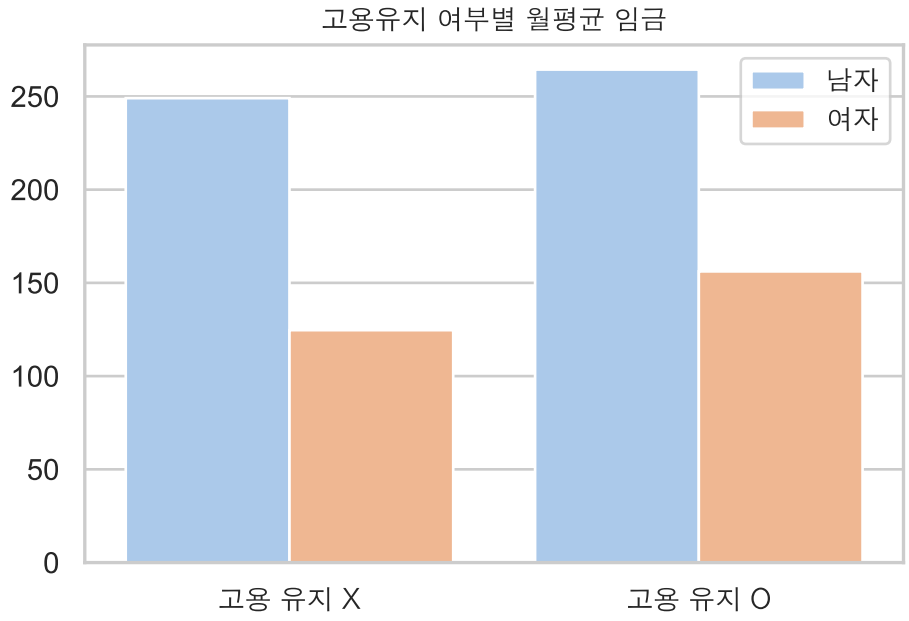
countplot과 비슷하지만 x 범주별로 y의 평균을 확인해보기에 선택해야 하는 변수가 하나더 필요하다.

- `x` : x축에 사용할 변수 이름을 지정 **[필수]**
- `y` : y축에 사용할 변수 이름을 지정 **[필수]**
- `data` : 그래프를 그릴 데이터프레임을 지정 **[필수]**
- `hue` : 막대의 색상을 그룹화할 변수 이름을 지정
- `order` : x축에 사용할 범주의 순서를 지정
- `orient` : 막대의 방향을 지정 ('v' 또는 'h', 기본값: 'v')
- `palette` : 색상 팔레트를 지정

```
ax = sns.barplot(x = '고용유지_여부', y = '근속기간(개월)', data = df)
ax.set_title('고용유지 여부별 근속기간') # 그래프 제목 추가
ax.set_xticks([0, 1], labels = ['고용 유지 X', '고용 유지 O']) # X축 눈금 변경


plt.show()
```

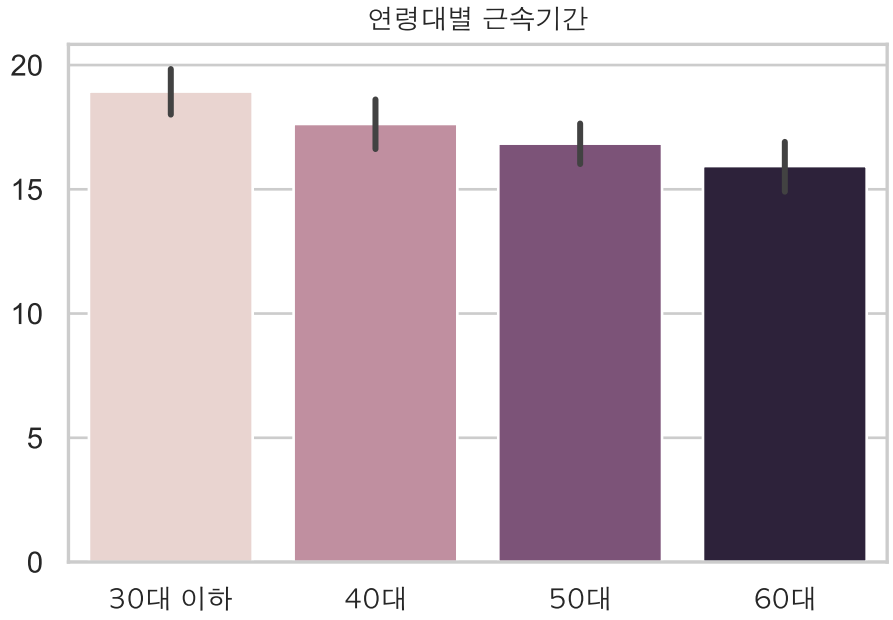



```
ax = sns.barplot(x = '고용유지_여부', y = '월평균임금', data = df,
                 hue = '성별', palette = 'pastel', ci=None) # palette 변경

ax.set_title('고용유지 여부별 월평균 임금') # 그래프 제목 추가
ax.set_xticks([0, 1], labels = ['고용 유지 X', '고용 유지 O']) # X축 눈금 변경

# 범례 변경
ax.legend(labels= ['남자', '여자'])

plt.show()
```




```
ax = sns.barplot(x = '연령대', y = '근속기간(개월)', data = df, hue = '연령대', legend = False)
ax.set_title('연령대별 근속기간') # 그래프 제목 추가
ax.set_xticks([0, 1, 2, 3], labels = ['30대 이하', '40대', '50대', '60대']) # X축 눈금 변경
```

```
plt.show()
```



밑에 2개는 코드만 제공합니다.

```
ax = sns.barplot(x = '사고여부', y = '월평균임금', data = df,
                 hue = '권역',
                 palette = 'pastel') # palette 변경

ax.set_title('사고 여부별 월평균 임금') # 그래프 제목 추가
ax.set_xticks([0, 1, 2], labels = ['업무상 사고', '업무상 질병', '출퇴근 재해']) # X축 눈금 변경

# 범례 변경
handles, _ = ax.get_legend_handles_labels()
ax.legend(handles, ['서울/강원', '경인', '충청/세종/대전', '광주/전라', '대구/경북', '부산/울산/경남'])
sns.move_legend(ax, 'lower left') # 범례 위치 변경

plt.show()
```

```
ax = sns.barplot(x = '권역', y = '월평균임금', data = df,
                 hue = '사고여부',
                 palette = 'pastel') # palette 변경

ax.set_title('사고 여부별 월평균 임금') # 그래프 제목 추가
ax.set_xticks(range(0,6,1),
              labels = ['서울/강원', '경인', '충청/세종/대전', '광주/전라', '대구/경북', '부산/울산/경남'])

# 범례 변경
handles, _ = ax.get_legend_handles_labels()
ax.legend(handles, ['업무상 사고', '업무상 질병', '출퇴근 재해'])
sns.move_legend(ax, 'lower right') # 범례 위치 변경
```

```
plt.show()
```

4.4 Scatterplot

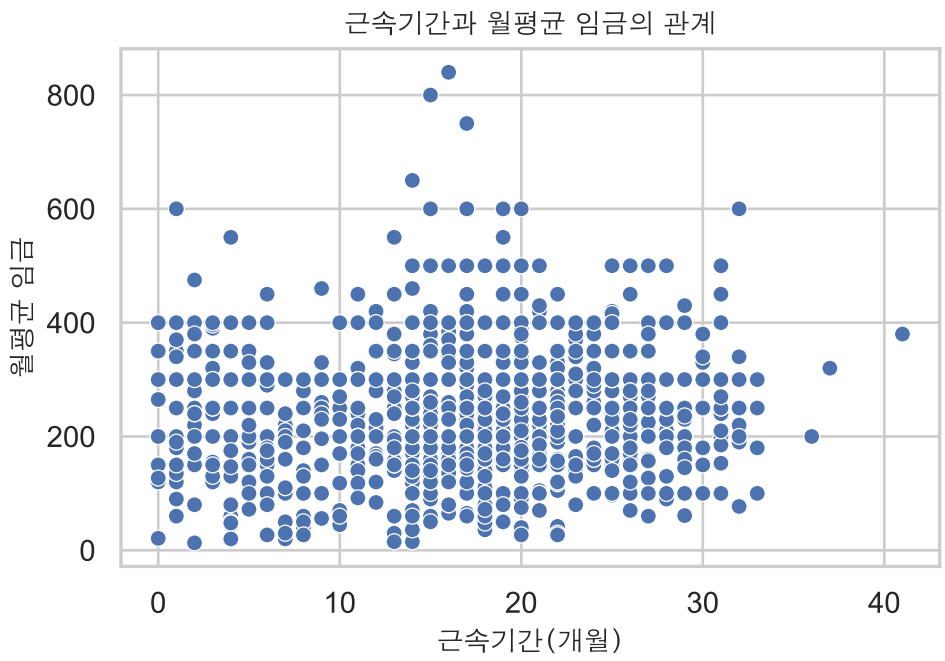
Scatterplot(산점도)은 두 변수 간의 관계를 시각적으로 표현하는 그래프이다.

각 데이터 포인트는 두 변수의 값을 나타내며, x축과 y축에 해당 변수의 값을 사용하여 점을 찍는다.

산점도는 변수 간의 상관 관계를 파악하고, 데이터의 분포나 패턴, 이상치 등을 식별하는 데 유용하다.

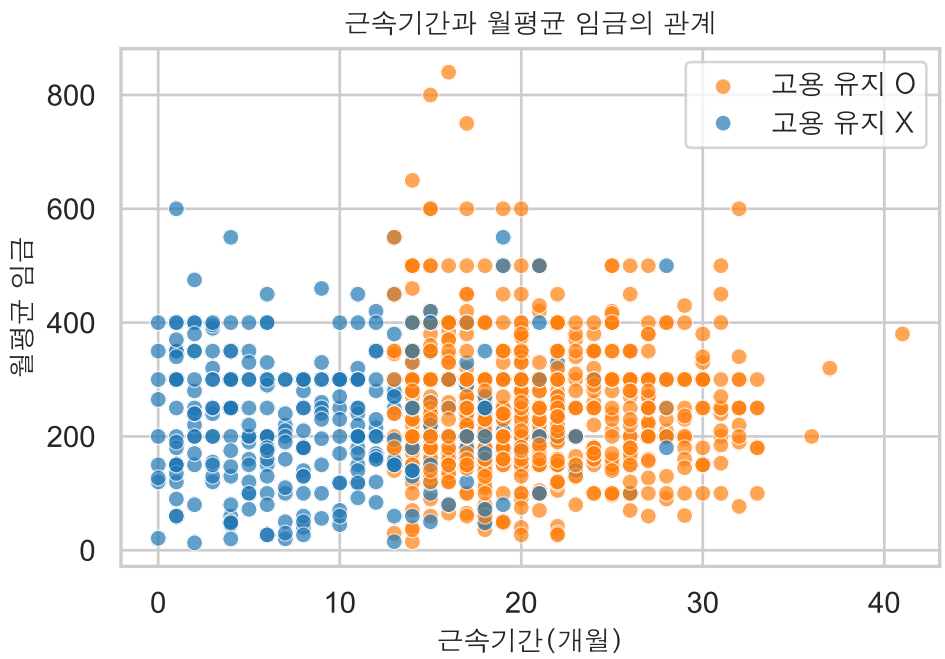
- `x` : x축에 사용할 변수 이름을 지정 **[필수]**
- `y` : y축에 사용할 변수 이름을 지정 **[필수]**
- `data` : 그래프를 그릴 데이터프레임을 지정 **[필수]**
- `style` : 점의 색상을 그룹화할 변수 이름을 지정
- `order` : 점의 스타일을 그룹화할 변수 이름을 지정
- `size` : 점의 크기를 그룹화할 변수 이름을 지정
- `alpha` : 점의 투명도를 지정
- `palette` : 색상 팔레트를 지정

```
ax = sns.scatterplot(x = '근속기간(개월)', y = '월평균임금', data = df)

ax.set_title('근속기간과 월평균 임금의 관계') # 그래프 제목 추가
plt.show()
```




```
ax = sns.scatterplot(x = '근속기간(개월)', y = '월평균임금', data = df,
                     hue = '고용유지_여부',  # 점의 색상을 그룹화할 변수 이름 지정
                     palette = 'tab10',      # 색상 팔레트 지정
                     alpha = 0.7             # 점의 투명도 지정
                     )

ax.set_title('근속기간과 월평균 임금의 관계') # 그래프 제목 추가
```

```
# 범례 변경
ax.legend(labels=['고용 유지 O', '고용 유지 X'])
plt.show()
```



4.5 Histogram

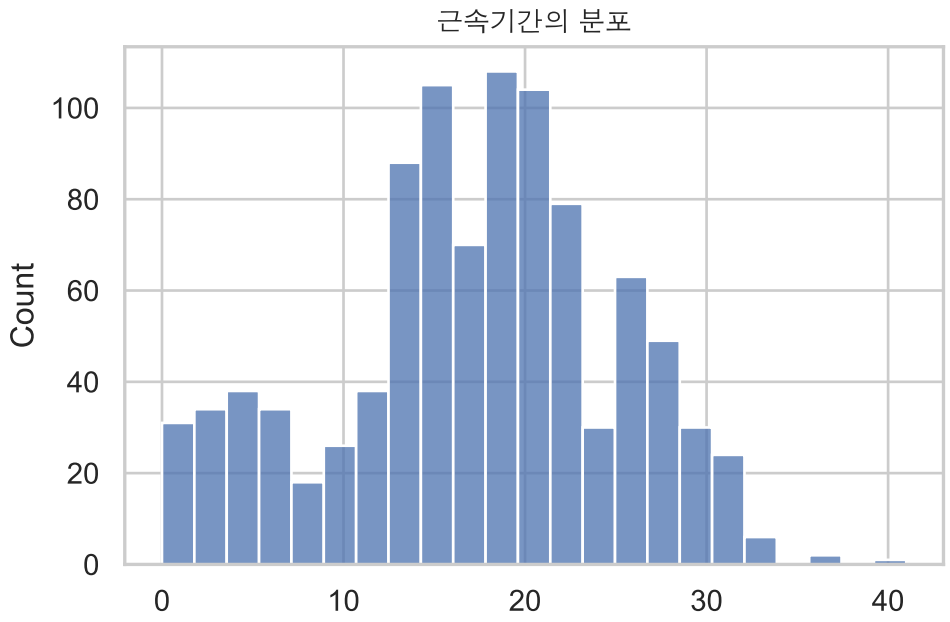
히스토그램(Histogram)은 데이터의 분포를 시각적으로 표현하는 그래프이다.

연속형 데이터를 일정한 구간(bin)으로 나누고, 각 구간에 속하는 데이터 포인트의 빈도를 막대로 나타낸다.

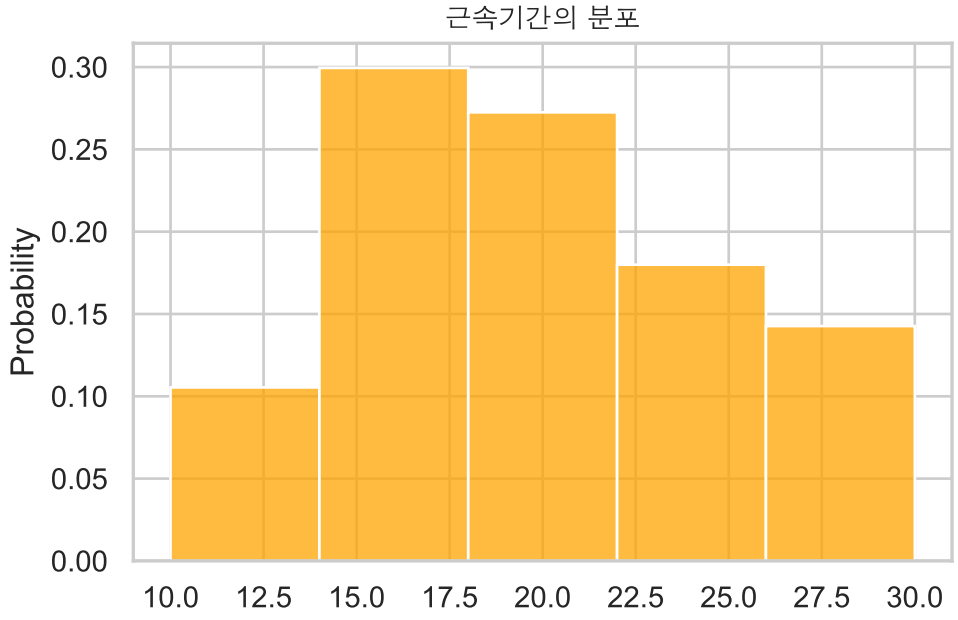
히스토그램은 데이터의 분포, 중앙값, 분산, 그리고 이상치를 파악하는 데 유용하다.

- `x` : x축에 변수 이름을 지정 **[필수]**
- `y` : y축에 변수 이름을 지정
- `data` : 그래프를 그릴 데이터프레임을 지정 **[필수]**
- `hue` : 히스토그램을 색상으로 그룹화할 변수 이름을 지정
- `bins` : 히스토그램의 빈(bin)의 개수를 지정
- `binwidth` : 각 빈의 너비를 지정
- `binrange` : 히스토그램의 x축 범위를 지정 (min, max)
- `stat` : 히스토그램의 y축에 표시할 통계량을 지정합니다. ('count', 'frequency', 'density', 'probability', 기본값 : 'count')
- `multiple` : 히스토그램의 스타일을 지정 ('layer', 'dodge', 'stack', 'fill')
- `element` : 히스토그램의 요소를 지정 ('bars', 'step', 'poly')
- `color` : 히스토그램의 색상을 지정
- `palette` : hue로 구분된 그룹에 대해 색상 팔레트를 지정

```
ax = sns.histplot(x = '근속기간(개월)', data = df)
ax.set_title('근속기간의 분포') # 그래프 제목 추가
plt.show()
```

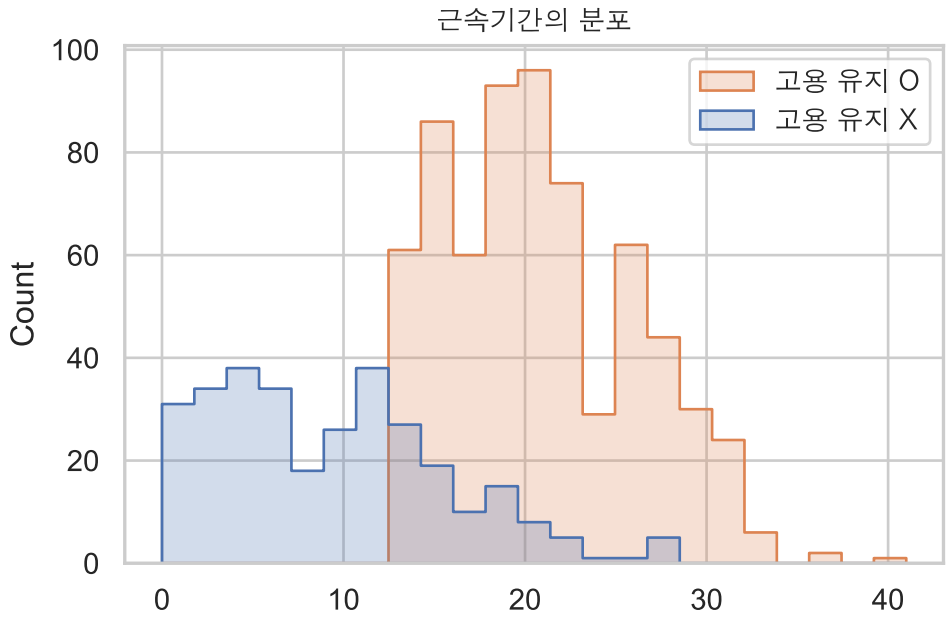



```
ax = sns.histplot(x = '근속기간(개월)', data = df,
                  bins = 5,                # 빈 개수 지정
                  binrange = (10, 30),     # x축 범위 지정
                  stat = 'probability',    # y축 통계량 지정
                  color = 'orange'         # 색상 지정
                  )

ax.set_title('근속기간의 분포') # 그래프 제목 추가

plt.show()
```

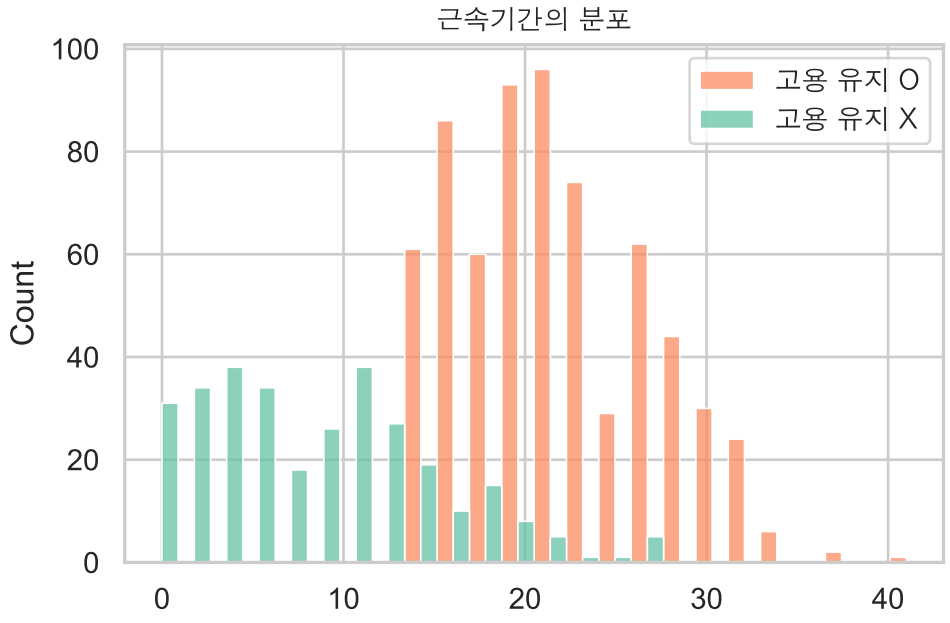



```
ax = sns.histplot(x = '근속기간(개월)', data = df,
                  hue = '고용유지_여부',     # 그룹화할 변수 이름 지정
                  element = 'step'          # 히스토그램 요소 지정
                  )
```

```
ax.set_title('근속기간의 분포') # 그래프 제목 추가

# 범례 변경
ax.legend(labels=['고용 유지 O', '고용 유지 X'])

plt.show()
```




```
ax = sns.histplot(x = '근속기간(개월)', data = df,
                  hue = '고용유지_여부',  # 그룹화할 변수 이름 지정
                  multiple = 'dodge',     # 스타일 지정
                  palette = 'Set2'        # 색상 팔레트 지정
                  )

ax.set_title('근속기간의 분포') # 그래프 제목 추가

# 범례 변경
ax.legend(labels=['고용 유지 O', '고용 유지 X'])

plt.show()
```


근속기간의 분포
Count
100
80
60
40
20
0
0
10
20
30
40
고용 유지 O
고용 유지 X