

3. Pandas 기초

3.1 Pandas 데이터 구조

- Pandas에서 제공하는 데이터 구조는 Series와 DataFrame이 있음

3.1.1 Series

- 각 값에 라벨(인덱스)이 있고 하나의 데이터 형식으로 이루어진 1차원 배열
- 라벨로 데이터에 접근할 수 있음

3.1.2 DataFrame

- 행과 열에 라벨이 있는 2차원 배열 → Excel
- 각 열은 서로 다른 데이터 타입을 지닐 수 있음

Series		DataFrame			
라벨	값	라벨			
사과	400		서울대	고려대	연세대
4	달러	김철수	불합격	합격	[불합격, 대기1]
4	원	김철수	(합격, 등록)	불합격	합격
배열	[1,2,3]	김영희	합격	합격	불합격
(1,2)	튜플				

3.2 DataFrame 기본

Pandas의 DataFrame을 사용하여 분석하기 위한 가장 첫 단계는 데이터를 호출하고 데이터의 내용과 요약, 통계정보를 확인하는 것입니다. 데이터나 분석 도구에 따라 컬럼명이나 컬럼의 타입을 변경해야 할 때도 있습니다.

3.2.1 Pandas 사용 준비

```
In [ ]: %pip install pandas
import pandas as pd
```

- as 문법을 통해 pandas를 별칭 pd로 사용

3.2.2 DataFrame 선언하기

```
In [ ]: import pandas as pd
```

```
import numpy as np
dataset = np.array([[ 'kor', 80], ['math',70]])
df = pd.DataFrame(dataset, columns=['subject', 'score'])
df
```

```
Out[ ]:
  subject  score
0      kor     80
1      math     70
```

📌 DataFrame 생성의 다양한 방법

Case1. columns 인자를 사용

```
In [ ]: df1 = pd.DataFrame([[ 'kor', 80], ['math',70]], columns=['subject', 'score'])
df1
```

```
Out[ ]:
  subject  score
0      kor     80
1      math     70
```

Case2. dictionary를 사용

- dictionary의 value는 배열 형태여야 함

```
In [ ]: df2 = pd.DataFrame({'subject':['kor', 'math'], 'score':[80, 70]})
df2
```

```
Out[ ]:
  subject  score
0      kor     80
1      math     70
```

3.2.3 DataFrame 출력하기

- `sklearn` 패키지의 기본 제공 데이터 세트인 iris 데이터를 불러오고 출력
- `head()`, `tail()` 함수를 사용해 데이터를 부분적으로 확인
- 함수의 인자로 아무 숫자도 지정하지 않으면 컬럼명을 포함하여 6줄을 출력

📌 sklearn 패키지

scikit-learn은 Python에서 데이터 과학과 머신러닝을 위한 강력한 라이브러리입니다. 데이터 전처리, 모델 학습, 평가, 그리고 다양한 머신러닝 알고리즘을 제공하여 데이터 분석과 모델링 작업을 쉽게 수행할 수 있도록 도와줍니다.

```
In [ ]: from sklearn.datasets import load_iris
iris = load_iris()
iris = pd.DataFrame(iris.data, columns=iris.feature_names)
iris
```

Out []:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In []: `# 데이터프레임의 상위 행만 확인`
`iris.head()`

Out []:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In []: `# 데이터프레임의 하위 행만 확인`
`iris.tail()`

Out []:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

3.2.4 DataFrame 요약 및 통계 확인하기

- 데이터의 내용을 확인했다면 그 이후에 데이터프레임의 요약, 통계정보를 확인함
- `info()` 함수를 통해 총 데이터의 건수, 각 컬럼의 데이터 형식 그리고 결측치의 건수를 확인할 수 있음
- 수치형 컬럼이 존재하는 경우 `describe()` 함수를 사용하여 수치형 컬럼들의 `n-percentile` 분포도, 평균값, 최대값, 최소값을 확인할 수 있음
- 이러한 과정은 EDA에 속함

```
In [ ]: iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
#   Column                Non-Null Count  Dtype
---  -
0   sepal length (cm)      150 non-null   float64
1   sepal width (cm)       150 non-null   float64
2   petal length (cm)      150 non-null   float64
3   petal width (cm)       150 non-null   float64
dtypes: float64(4)
memory usage: 4.8 KB
```

info 함수 결과해석


- iris 데이터는 150개의 행과 4개의 컬럼을 가지고 있음
- 각 컬럼의 이름은 'sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'
- Non-Null Count가 모두 non-null이므로 결측값을 가지고 있지 않음
- 모든 데이터의 타입은 float64로 실수형 데이터

```
In [ ]: iris.describe()
```

```
Out[ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

describe 함수 결과해석

- describe를 확인하는 경우 주로 mean, min, max를 확인
- iris의 sepal length (cm) 컬럼의 범위는 4.3~7.9
- petal width (cm) 컬럼의 범위는 0.1~2.5  크기가 매우 작은 것이 존재함

더 알아보기

- 🤔 : 상대적으로 값이 작은 컬럼이 모델에 미치는 영향이 어떻게 되나요?
- 😊 : 데이터 값의 크기가 상대적으로 작은 경우 실제로 큰 관계가 있더라도 다른 컬럼들에 비해 분석 모델에 영향을 덜 미칠 수 있습니다. 따라서 모델링 전 변수가 관계로만 영향을 미칠 수 있도록 전처리를 수행해야 합니다.

3.2.5 DataFrame 인덱스 핸들링

- 인덱스의 추가/변경/삭제

```
In [ ]: # 원본 데이터프레임 확인
df
```

```
Out[ ]:   subject  score
0      kor     80
1     math     70
```

```
In [ ]: # index를 list로 변경하여 확인
list(df.index)
```

```
Out[ ]: [0, 1]
```

```
In [ ]: # index 수정
df.index = ['A', 'B']

# index를 list로 변경하여 확인
list(df.index)
```

```
Out[ ]: ['A', 'B']
```

```
In [ ]: # 인덱스가 변경된 데이터프레임 확인
df
```

```
Out[ ]:   subject  score
A      kor     80
B     math     70
```

📌 set_index

DataFrame 내의 열을 인덱스로 변경

```
DataFrame.set_index(keys, drop=True, append=False, inplace=True)
```

- keys: 인덱스로 사용하고자 하는 컬럼의 이름을 문자형으로 입력
- drop: 인덱스로 세팅한 컬럼을 DataFrame 내에서 삭제할지를 결정
- append: 기존에 존재하던 인덱스를 삭제할지, 컬럼으로 추가할지를 결정
- inplace: 원본 객체를 변경할지를 결정

```
In [ ]: # class 열을 인덱스로 지정 후 삭제
# 기존의 인덱스는 삭제됨
# 원본 df를 영구히 변경
```

```
df.set_index('subject', drop=True, append=False, inplace=True)
df
```

```
Out[ ]:      score
subject
kor      80
math     70
```

reset_index

인덱스를 0부터 시작하는 정수로 재설정

```
DataFrame.reset_index(drop=False, inplace=False)
```

- drop: 기존 인덱스를 DataFrame 내에서 삭제할지, 컬럼으로 추가할지를 결정
- inplace: 원본 객체를 변경할지를 결정

```
In [ ]: df.reset_index(drop=False, inplace=True)
df
```

```
Out[ ]:   subject  score
0      kor      80
1     math      70
```

3.2.6 DataFrame의 열(column) 핸들링

```
In [ ]: # 데이터프레임의 열 확인
iris.columns
```

```
Out[ ]: Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
              'petal width (cm)'],
              dtype='object')
```

```
In [ ]: # 데이터프레임의 열 이름 변경
iris.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
iris
```

```
Out[ ]:
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows x 4 columns

3.2.8 DataFrame 컬럼의 데이터 타입 확인 및 변경

- dtypes를 통해 각 컬럼의 속성을 확인할 수 있음
- astype()을 통해 데이터 타입을 변경할 수 있음

📌 데이터프레임의 컬럼

- int: 정수형
- float: 실수형, 소수점을 가짐
- bool: True or False
- datetime: 날짜와 시간 표현
- object: 문자열&복합형

```
In [ ]: # 각 컬럼의 데이터 타입 확인
iris.dtypes
```

```
Out[ ]: sepal_length    float64
sepal_width          float64
petal_length         float64
petal_width          float64
dtype: object
```

```
In [ ]: # astype을 이용한 데이터 타입 변경
iris['sepal_length'] = iris['sepal_length'].astype('int')
iris.dtypes
```

```
Out[ ]: sepal_length    int64
sepal_width          float64
petal_length         float64
petal_width          float64
dtype: object
```

3.3 row/ column 핸들링

데이터프레임은 행과 열이 각각의 label을 가지고 있어 레이블의 이름이나 그 위치를 통해 행 또는 열을 추가하고 삭제할 수 있습니다.

3.3.1 row/ column의 선택 조회

1. row 선택 조회

- 슬라이싱을 통한 행 선택

📌 슬라이싱

시작과 끝을 지정해 잘라내는 방법, 데이터프레임이 아닌 list 등에서도 사용 가능

```
DataFrame[n:m]
```

- n번째 행부터 m-1번째 행까지 데이터프레임을 자름

```
In [ ]: from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()
iris = pd.DataFrame(iris.data, columns=iris.feature_names)

# 1번째 행부터 4-1번째 행의 데이터 확인
iris[1:4]
```

```
Out [ ]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
1              4.9             3.0             1.4             0.2
2              4.7             3.2             1.3             0.2
3              4.6             3.1             1.5             0.2
```

- 데이터프레임은 0번 인덱스부터 수행함

2. column 선택 조회

- 컬럼명 또는 컬럼명의 리스트를 이용함

🔥 key point

- 컬럼명 → Series
- 컬럼명 리스트 → DataFrame

```
In [ ]: # 컬럼명으로 데이터 확인
iris['petal length (cm)'].head()
```

```
Out [ ]: 0    1.4
1    1.4
2    1.3
3    1.5
4    1.4
Name: petal length (cm), dtype: float64
```

```
In [ ]: # 컬럼명의 리스트로 데이터 확인
iris[['petal length (cm)', 'petal width (cm)']].head()
```



```
Out [ ]:
```

	petal length (cm)	petal width (cm)
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2

3 row와 column 선택 조회

- `iloc` 또는 `loc`을 사용해 행과 열 모두 지정할 수 있음

`iloc`

정수로 특정 행과 열을 선택하고 싶은 경우 사용

```
DataFrame.iloc[row, column]
```

- 인수를 하나만 설정할 경우 행을 지정
- 인수를 두 개 설정할 경우 순서대로 행과 열 지정
- 인수는 정수, 리스트, 슬라이싱을 사용할 수 있음

```
In [ ]:
```

```
# 슬라이싱, 열은 지정하지 않았기에 모든 열을 선택
# 1번째 행부터 4-1번째 행의 데이터 확인
iris.iloc[1:4]
```

```
Out [ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2

```
In [ ]:
```

```
# 행은 리스트로 특정 인덱스들을 지정
# 열은 슬라이싱으로 특정 열 범위 지정
iris.iloc[[1,3,5],2:4]
```

```
Out [ ]:
```

	petal length (cm)	petal width (cm)
1	1.4	0.2
3	1.5	0.2
5	1.7	0.4

`loc`

레이블(인덱스 이름 혹은 컬럼명)로 행과 열을 선택하고 싶은 경우 사용

```
DataFrame.loc[row, column]
```

- 인수를 하나만 설정할 경우 행을 지정
- 인수를 두 개 설정할 경우 순서대로 행과 열 지정
- 인수는 정수, 리스트, 슬라이싱을 사용할 수 있음
- `loc`의 슬라이싱은 일반적인 파이썬의 슬라이싱과 다름 (진짜 시작부터 끝)

```
In [ ]: # loc에 대한 이해를 위해 인덱스를 변경
iris_index = list(iris.index)
iris_index[0] = '파이썬'
iris.index = iris_index
iris.head()
```

```
Out [ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
파이썬	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
In [ ]: # '파이썬' 라벨의 행부터 3번째 행까지 데이터 확인
iris.loc['파이썬':3]
```

```
Out [ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
파이썬	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2

```
In [ ]: # 컬럼명도 슬라이스로 지정할 수 있음
iris.loc[['파이썬', 3, 5], 'sepal width (cm)': 'petal width (cm)']
```

```
Out [ ]:
```

	sepal width (cm)	petal length (cm)	petal width (cm)
파이썬	3.5	1.4	0.2
3	3.1	1.5	0.2
5	3.9	1.7	0.4

4. 선택한 값 변경하기

- 특정한 위치를 지정했다면 다른 값을 대입해 값을 변경할 수 있음

```
In [ ]: # 값을 변경할 데이터 확인
print(iris.loc['파이썬', 'sepal length (cm)'])

5.1
```

```
In [ ]: # 값 변경
iris.loc['파이썬', 'sepal length (cm)'] = 100
iris.head()
```

```
Out [ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
파이썬	100.0	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

3.3.2 row/ column Concat

concat은 데이터프레임간의 단순한 연결을 의미합니다. concat을 통해 두 개 이상의 데이터프레임을 열 혹은 행 방향으로 붙일 수 있으며, 이를 통해 열 혹은 행을 추가할 수 있습니다.

concat

```
pandas.concat(objs, axis=0, ignore_index=False)
```

- objs: concat을 실행할 객체의 리스트
- axis: 0이면 행 추가, 1이면 열 추가
- ignore_index: True이면 기존 index를 무시하고 0부터 시작하는 정수로 재설정

```
In [ ]: score = pd.DataFrame({'국어': [80, 90, 70], '수학': [90, 70, 60]}, index=['홍길동', '김철수', '이영희'])
```

```
Out [ ]:
```

	국어	수학
홍길동	80	90
김철수	90	70
이영희	70	60

```
In [ ]: new_student = pd.DataFrame({'국어': [100], '수학': [100]}, index=['엄친아'])
```

```
Out [ ]:
```

	국어	수학
엄친아	100	100

```
In [ ]: # 행 추가
score = pd.concat([score, new_student], axis=0)
```

```
Out [ ]:
```

	국어	수학
홍길동	80	90
김철수	90	70
이영희	70	60
엄친아	100	100

```
In [ ]: new_subject = pd.DataFrame({'과학': [100, 90, 80, 100]}, index=['홍길동', '김철수', '이영희', '엄친아'])
```

Out []:

과학	
홍길동	100
김철수	90
이영희	80
엄친아	100

```
In [ ]: # 열 추가
score = pd.concat([score, new_subject], axis=1)
score
```

Out []:

	국어	수학	과학
홍길동	80	90	100
김철수	90	70	90
이영희	70	60	80
엄친아	100	100	100

3.3.3 row/column 삭제

- drop() 함수를 사용하여 행과 열을 삭제할 수 있음
- 행과 열 둘다 지정하여 삭제하는 것은 불가능함

drop

```
DataFrame.drop(index=None, columns=None, inplace=False)
```

- index: 삭제할 행의 인덱스 혹은 인덱스의 리스트를 지정
- columns: 삭제할 컬럼의 이름 혹은 컬럼명의 리스트를 지정
- inplace: True이면 작업 수행과 동시에 원본에 반영

```
In [ ]: # index를 기준으로 행 삭제
score = score.drop('엄친아', inplace=False)
score
```

Out []:

	국어	수학	과학
홍길동	80	90	100
김철수	90	70	90
이영희	70	60	80

```
In [ ]: # column명을 기준으로 열 삭제
score = score.drop(columns=['과학'], inplace=False)
score
```

```
Out [ ]:
```

	국어	수학
홍길동	80	90
김철수	90	70
이영희	70	60

3.4 조건에 맞는 데이터 탐색 및 수정

3.4.1 조건 탐색

📌 데이터프레임 조건 탐색

DataFrame[조건식]

1. 조건이 하나인 경우

```
In [ ]: score[score['국어'] > 80]
```

```
Out [ ]:
```

	국어	수학
김철수	90	70

2. 조건이 두 개인 경우

```
In [ ]: # 둘 다 만족해야 함 (and)
score[(score['국어'] > 70) & (score['수학'] > 80)]
```

```
Out [ ]:
```

	국어	수학
홍길동	80	90

```
In [ ]: # 둘 중 하나만 만족해도 됨 (or)
score[(score['국어'] > 85) | (score['수학'] > 85)]
```

```
Out [ ]:
```

	국어	수학
홍길동	80	90
김철수	90	70

3.4.2 결측값 탐색 및 수정

1. 결측값 탐색

- 결측값 탐색 함수: `isna()`, `isnull()`
- 결측값이 아닌 값 탐색: `notna()`, `notnull()`
- 결측값 탐색의 결과는 원본 데이터프레임에 `True`, `False`로 나타남

```
In [ ]: # 결측값 탐색을 위한 결측치 추가
new_student = pd.DataFrame({'수학': [100]}, index=['엄친아'])
score = pd.concat([score, new_student], axis=0)
score
```

```
Out[ ]:
```

	국어	수학
홍길동	80.0	90
김철수	90.0	70
이영희	70.0	60
엄친아	NaN	100

```
In [ ]: # 결측치 탐색
score.isnull()
```

```
Out[ ]:
```

	국어	수학
홍길동	False	False
김철수	False	False
이영희	False	False
엄친아	True	False

```
In [ ]: # 결측이 아닌 값 탐색
score.notna()
```

```
Out[ ]:
```

	국어	수학
홍길동	True	True
김철수	True	True
이영희	True	True
엄친아	False	True

2. 결측치 합계

- True는 1 False는 0으로 집계
- sum()은 컬럼별, sum(axis=1)은 행별 집계 수행

```
In [ ]: # 결측치 탐색 후 결측치 개수 확인 (열별)
score.isnull().sum()
```

```
Out[ ]:
```

국어	1
수학	0

dtype: int64

```
In [ ]: # 결측치 탐색 후 결측치 개수 확인 (행별)
score.isnull().sum(axis=1)
```

```
Out[ ]:
```

홍길동	0
김철수	0
이영희	0
엄친아	1

dtype: int64

3. 결측값 제거

- dropna()는 결측값이 존재하는 행 또는 열을 삭제

dropna

```
DataFrame.dropna(axis=0, how='any', thresh=None, inplace=False)
```

- axis: 0이면 결측값이 포함된 행삭제, 1이면 결측값이 포함된 열 삭제
- how: 'any'면 결측값이 존재한 모든 행/열 삭제, 'all'이면 모든 값이 결측값일때 삭제
- thresh: 정수 값을 지정하면 결측값이 아닌 값이 그보다 많을 때 행 또는 열을 유지

```
In [ ]: # 결측치 제거  
# inplace=True로 설정하면 원본 데이터프레임이 변경됨  
score.dropna()
```

```
Out [ ]:
```

	국어	수학
홍길동	80.0	90
김철수	90.0	70
이영희	70.0	60

4. 결측값 대체

- fillna() 함수를 활용하여 다양한 방법으로 결측값을 대체할 수 있음

fillna

```
DataFrame.fillna(value=None, method=None, axis=None, inplace=False)
```

- value: 결측치를 대체할 값 직접 입력
- method: 'pad', 'fill'은 이전값 대체, 'backfill', 'bfill'은 다음 값으로 대체
- axis: 0이면 행 방향 대체, 1이면 열 방향 대체

```
In [ ]: # 0으로 대체  
score.fillna(0)
```

```
Out [ ]:
```

	국어	수학
홍길동	80.0	90
김철수	90.0	70
이영희	70.0	60
엄친아	0.0	100

```
In [ ]: # 평균값으로 대체  
score.fillna(score.mean())
```

```
Out [ ]:
```

	국어	수학
홍길동	80.0	90
김철수	90.0	70
이영희	70.0	60
엄친아	80.0	100

```
In [ ]: # 이전값 대체
score.fillna(method='ffill')
```

```
Out [ ]:
```

	국어	수학
홍길동	80.0	90
김철수	90.0	70
이영희	70.0	60
엄친아	70.0	100

```
In [ ]: # 다음값 대체
score.fillna(method='bfill')
```

```
Out [ ]:
```

	국어	수학
홍길동	80.0	90
김철수	90.0	70
이영희	70.0	60
엄친아	NaN	100

더 알아보기

- 🤔 : 결측값이 존재하면 지우는 방법과 대체하는 방법 중 무엇을 사용해야 하나요?
- 😊 : 데이터 셋이 클 경우 결측값이 약 5%미만이면 지워도 무방하고 그렇지 않다면 적절한 값으로 대체하는 것이 분석 모델의 설명력이 높아집니다.

5. 중복행 삭제

- drop_duplicates() 함수를 사용해 중복을 삭제할 수 있음

drop_duplicates

```
DataFrame.drop_duplicates()
```

- inplace가 없으므로 재할당 필요

```
In [ ]: # 중복행 생성 <=> 수학 점수가 중복됨
score.loc['이영희', '수학'] = 70
score
```

```
Out [ ]:
```

	국어	수학
홍길동	80.0	90
김철수	90.0	70
이영희	70.0	70
엄친아	NaN	100

```
In [ ]: # 컬럼을 지정하지 않은 경우
score.drop_duplicates()
```



```
Out [ ]:
```

	국어	수학
홍길동	80.0	90
김철수	90.0	70
이영희	70.0	70
엄친아	NaN	100

```
In [ ]: # 컬럼을 지정
score['수학'].drop_duplicates()
```

```
Out [ ]: 홍길동      90
김철수      70
엄친아     100
Name: 수학, dtype: int64
```

3.5 데이터 정렬

3.5.1 인덱스 기준 정렬

📌 sort_index

```
DataFrame.sort_index(ascending=True, inplace=False)
```

- ascending: True-오름차순, False-내림차순

```
In [ ]: import pandas as pd
from sklearn.datasets import load_iris
# 데이터 불러오기
iris = load_iris()
# 데이터프레임으로 변환
iris = pd.DataFrame(iris.data, columns=iris.feature_names)
iris.head()
```

```
Out [ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
In [ ]: # 인덱스 기준 내림차순 정렬
iris.sort_index(ascending=False, inplace=True)
iris.head()
```

```
Out [ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
149	5.9	3.0	5.1	1.8
148	6.2	3.4	5.4	2.3
147	6.5	3.0	5.2	2.0
146	6.3	2.5	5.0	1.9
145	6.7	3.0	5.2	2.3

3.5.2 컬럼 기준 정렬

sort_values

```
DataFrame.sort_values(by, ascending=True, inplace=False)
```

- by: 정렬 기준으로 사용할 컬럼 혹은 컬럼 리스트
 - 리스트일 경우 리스트의 순서대로 우선순위 부여
- ascending: True-오름차순, False-내림차순

```
In [ ]: # sepal length (cm) 컬럼 기준 내림차순 정렬
iris.sort_values(by='sepal length (cm)', ascending=False, inplace=True)
iris.head()
```

```
Out [ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
131	7.9	3.8	6.4	2.0
135	7.7	3.0	6.1	2.3
117	7.7	3.8	6.7	2.2
118	7.7	2.6	6.9	2.3
122	7.7	2.8	6.7	2.0

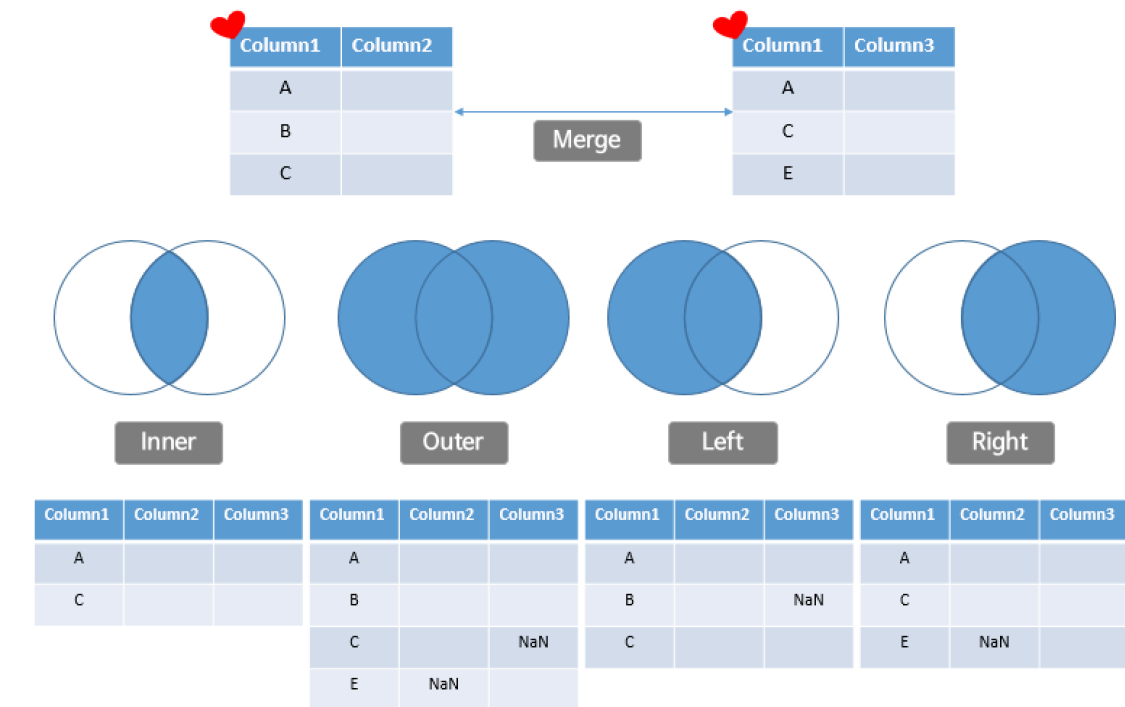
```
In [ ]: # 컬럼명 리스트로 정렬
# 각각 다른 정렬 방식을 적용할 수 있음
iris.sort_values(by=['sepal length (cm)', 'sepal width (cm)'], ascending=False)
iris.head()
```

```
Out [ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
131	7.9	3.8	6.4	2.0
118	7.7	2.6	6.9	2.3
122	7.7	2.8	6.7	2.0
135	7.7	3.0	6.1	2.3
117	7.7	3.8	6.7	2.2

3.6 데이터 병합

앞서 수행한 concat(연결)과 다른 방식인 merge(병합) 방식은 두 데이터프레임이 동시에 가지고 있는 특정한 컬럼의 값인 key를 기준으로 병합을 수행한다.



📌 merge

```
pandas.merge(left, right, how='inner', on=None)
```

- left: 왼쪽에 해당하는 데이터프레임
- right: 오른쪽에 해당하는 데이터프레임
- how: 병합 방식
- on: key 컬럼 집합

🔥 Key Point

- 병합은 이미지로 생각하여 수행하면 이해하기 편함
- 병합시 key 컬럼은 존재하나 동일한 값이 없는 경우 NaN을 반환함
- 병합후 결측치에 대한 처리 필요

```
In [ ]: # 병합 예시를 위한 데이터프레임 생성
product = pd.DataFrame({'product': ['a', 'b', 'c', 'd', 'e'], 'price': [100, 150, 200, 250, 300]})
```

```
Out [ ]:
   product  price
0        a    100
1        b    150
2        c    200
3        d    250
4        e    300
```

```
In [ ]: # 병합 예시를 위한 데이터프레임 생성
sale = pd.DataFrame({'product': ['a', 'b', 'c'], 'sale': [10, 20, 30]})
```

Out[]:

	product	sale
0	a	10
1	b	20
2	c	30

```
In [ ]: # inner join 수행
inner = pd.merge(product, sale, on='product', how='inner')
inner
```

Out[]:

	product	price	sale
0	a	100	10
1	b	150	20
2	c	200	30

```
In [ ]: # outer join 수행
outer = pd.merge(product, sale, on='product', how='outer')
outer
```

Out[]:

	product	price	sale
0	a	100	10.0
1	b	150	20.0
2	c	200	30.0
3	d	250	NaN
4	e	300	NaN

```
In [ ]: # left join 수행
left = pd.merge(product, sale, on='product', how='left')
left
```

Out[]:

	product	price	sale
0	a	100	10.0
1	b	150	20.0
2	c	200	30.0
3	d	250	NaN
4	e	300	NaN

```
In [ ]: # right join 수행
right = pd.merge(product, sale, on='product', how='right')
right
```

Out[]:

	product	price	sale
0	a	100	10
1	b	150	20
2	c	200	30

📌 결측치 처리 복습

```
In [ ]: # outer 데이터프레임에 대한 결측치 확인
        outer.isna().sum()
```

```
Out[ ]: product      0
        price       0
        sale        2
        dtype: int64
```

- sale 컬럼에 2개의 결측치 있음을 확인

```
In [ ]: # 결측치 대체
        outer.fillna(0)
```

```
Out[ ]:   product  price  sale
0         a    100  10.0
1         b    150  20.0
2         c    200  30.0
3         d    250   0.0
4         e    300   0.0
```

- 세일을 하지 않는 품목으로 할인율 0 적용

3.7 데이터 요약

데이터를 요약하여 원하는 정보를 얻는 것은 데이터가 지닌 값의 특징을 알게 해줍니다. 이와 비슷한 것을 3.2에서 수행해보았는데, 이번 장에서는 세부적으로 데이터를 요약하여 집계하는 방법을 알아보겠습니다.

3.7.1 그룹화 집계

👁 Definition

그룹화는 하나 이상의 데이터를 조건에 따라 여러개 그룹으로 묶는 것

- groupby() 함수를 사용함

📌 groupby

DataFrame.groupby(by=None, axis=0, sort=False, dropna=True).FUN()

- by: 그룹을 결정하는 데 사용
- axis: 행(0)과 열(1) 지정
- sort: 집계된 내용을 정렬할지 여부
- dropna: True이면 결측값이 행/열과 함께 삭제, False이면 결측값도 그룹의 키로 처리
- FUN: 집계함수

🔥 집계함수

- count(): 값의 개수
- sum(): 값들의 합
- std(): 표준편차
- var(): 분산
- min(): 최솟값
- max(): 최대값
- mean(): 평균값
- median(): 중앙값

```
In [ ]: import pandas as pd
        from sklearn.datasets import load_iris
        IRIS = load_iris()
        iris = pd.DataFrame(IRIS.data, columns=IRIS.feature_names)
        iris.head()
```

```
Out[ ]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0          5.1          3.5          1.4          0.2
1          4.9          3.0          1.4          0.2
2          4.7          3.2          1.3          0.2
3          4.6          3.1          1.5          0.2
4          5.0          3.6          1.4          0.2
```

```
In [ ]: # class 열 추가
        # map을 사용해 target 값{0,1,2}을 꽃 이름으로 변환
        iris['class'] = IRIS.target
        iris['class'] = iris['class'].map({0:'setosa', 1:'versicolor', 2:'virginica'})
        iris.head()
```

```
Out[ ]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  class
0          5.1          3.5          1.4          0.2  setosa
1          4.9          3.0          1.4          0.2  setosa
2          4.7          3.2          1.3          0.2  setosa
3          4.6          3.1          1.5          0.2  setosa
4          5.0          3.6          1.4          0.2  setosa
```

```
In [ ]: # 집계함수를 이용한 그룹화
        iris.groupby(by='class').mean()
```

```
Out[ ]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
class
setosa          5.006          3.428          1.462          0.246
versicolor      5.936          2.770          4.260          1.326
virginica       6.588          2.974          5.552          2.026
```

3.7.2 도수분포표

👁️ Definition

자료를 몇개의 구간으로 나누고, 나누어진 각 구간에 속한 자료의 개수를 정리한 표

- 데이터의 개수가 어떻게 분포되어 있는지 확인
- 이 표를 시각화 한 것을 히스토그램이라고 함
- `value_counts()` 함수를 사용

```
In [ ]: # 도수분포표
pd.Series(iris['class']).value_counts()
```

```
Out[ ]: setosa      50
versicolor  50
virginica    50
Name: class, dtype: int64
```

🖋️ 더 알아보기

- 😬: 자료를 몇개의 구간으로 나눈다고 했는데 이걸 개수만 세는 걸까요?
- 😊: 수치형 데이터에 대해서 자료를 구간화 할 수 있습니다!

```
In [ ]: # 데이터를 3등분 하고, 각 구간에 레이블 부여
three_cut = pd.qcut(iris['petal width (cm)'], 3, labels=['low', 'medium', 'high'])
three_cut
```

```
Out[ ]: 0      low
1      low
2      low
3      low
4      low
...
145    high
146    high
147    high
148    high
149    high
Name: petal width (cm), Length: 150, dtype: category
Categories (3, object): ['low' < 'medium' < 'high']
```

```
In [ ]: # 도수분포표를 만들 새로운 열 추가
iris['petal width level'] = three_cut
iris.head()
```

```
Out[ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class	petal width level
0	5.1	3.5	1.4	0.2	setosa	low
1	4.9	3.0	1.4	0.2	setosa	low
2	4.7	3.2	1.3	0.2	setosa	low
3	4.6	3.1	1.5	0.2	setosa	low
4	5.0	3.6	1.4	0.2	setosa	low

```
In [ ]: iris['petal width level'].value_counts()
```

```
Out[ ]: medium    52
        low       50
        high      48
        Name: petal width level, dtype: int64
```

✎ 더 알아보기

- 🤔 : 하나의 기준이 아닌 여러개의 기준으로 분포표를 생성할 수 있나요?
- 😊 : `crosstab()` 함수를 통해 교차표를 그릴 수 있습니다!

```
In [ ]: pd.crosstab(iris['petal width level'], iris['class'])
```

```
Out[ ]:
```

	class	setosa	versicolor	virginica
petal width level				
low		50	0	0
medium		0	48	4
high		0	2	46

3.8 데이터프레임에 함수 적용하기

Apply와 Map 함수를 사용하여 데이터의 행 또는 열에 자유롭게 함수를 적용하고 값을 변경할 수 있습니다.

- apply/ map 둘 다 동일한 기능 수행
- 데이터프레임에 행 또는 열 방향으로 지정한 함수 실행

🔥 꿀팁

- lambda는 apply를 시작할 때 "lambda 뒤에 오는 변수를 가지고 수행할거야!"라는 뜻
- 이 변수는 하나의 컬럼일 수도 있고 여러 컬럼일 수도 있으며 다양한 처리를 수행할 수 있음
- 대상이 여러 컬럼일 경우 axis=1를 꼭 명시해야함

```
In [ ]: # 하나의 컬럼에 대한 수행
iris['double sepal'] = iris['sepal width (cm)'].apply(lambda x: x*2)
iris.head()
```

```
Out[ ]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class	petal width level	double sepal
0	5.1	3.5	1.4	0.2	setosa	low	7.0
1	4.9	3.0	1.4	0.2	setosa	low	6.0
2	4.7	3.2	1.3	0.2	setosa	low	6.4
3	4.6	3.1	1.5	0.2	setosa	low	6.2
4	5.0	3.6	1.4	0.2	setosa	low	7.2

```
In [ ]: # 여러 컬럼에 대한 수행
iris['sepal petal sum'] = iris.apply(lambda x: x['sepal length (cm)'] + x['petal length (cm)'], axis=1)
iris.head()
```


Out []:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class	petal width level	double sepal	sepal petal sum
0	5.1	3.5	1.4	0.2	setosa	low	7.0	6.5
1	4.9	3.0	1.4	0.2	setosa	low	6.0	6.3
2	4.7	3.2	1.3	0.2	setosa	low	6.4	6.0
3	4.6	3.1	1.5	0.2	setosa	low	6.2	6.1
4	5.0	3.6	1.4	0.2	setosa	low	7.2	6.4

3.9 문자열 데이터 변환하기

데이터프레임의 문자열 데이터를 열 방향으로 한 번에 변환하여 정제할 수 있습니다. 이때 Series의 `str`이라는 속성을 사용합니다.

🔥 Key point

- `str`은 Series에 있는 속성, DataFrame에는 없음
- `DataFrame['컬럼명']`은 Series
- 즉, 특정 컬럼에 대해 수행할 수 있음

3.9.1 인덱싱

- 슬라이스를 사용한 인덱싱 수행

In []:

```
import pandas as pd

# 데이터프레임 생성
landmark = pd.DataFrame({'name': ['광화문', '호미곶', '첨성대'], 'location': ['서울 종로구 사직로 161', '경북 포항시 남구 호미곶면 대보리 150', '경북 경주시 인왕동 839-1']})
```

Out []:

	name	location
0	광화문	서울 종로구 사직로 161
1	호미곶	경북 포항시 남구 호미곶면 대보리 150
2	첨성대	경북 경주시 인왕동 839-1


In []:

```
# location 문자열의 3번째부터 6번째 문자까지 추출
landmark['location'].str[3:6]
```

Out []:

```
0    종로구
1    포항시
2    경주시
Name: location, dtype: object
```

3.9.2 분할

- 특정한 값을 기준으로 문자열을 나눌 수 있음
- `split()` 함수 사용  결과는 리스트
- `expand` 조건을 통해 분할된 문자열을 별도의 문자열로 변환할 수 있음

```
In [ ]: # 공백을 기준으로 문자열을 나누어 리스트로 반환
string_list = landmark['location'].str.split(' ')
string_list
```

```
Out[ ]: 0          [서울, 종로구, 사직로, 161]
1      [경북, 포항시, 남구, 호미곶면, 대보리, 150]
2          [경북, 경주시, 인왕동, 839-1]
Name: location, dtype: object
```

```
In [ ]: # 각 리스트의 1번째 문자열 반환
string_list.str[0]
```

```
Out[ ]: 0      서울
1      경북
2      경북
Name: location, dtype: object
```

```
In [ ]: # location 컬럼을 공백을 기준으로 나누어 첫번째 문자열을 새로운 컬럼으로 추가
landmark['location_1'] = landmark['location'].str.split(' ').str[0]
landmark
```

```
Out[ ]:   name          location location_1
0  광화문      서울 종로구 사직로 161      서울
1  호미곶  경북 포항시 남구 호미곶면 대보리 150      경북
2  첨성대      경북 경주시 인왕동 839-1      경북
```

```
In [ ]: # expand=True로 설정하여 리스트를 컬럼으로 확장
landmark['location'].str.split(' ', expand=True)
```

```
Out[ ]:   0      1      2      3      4      5
0  서울  종로구  사직로      161  None  None
1  경북  포항시  남구  호미곶면  대보리    150
2  경북  경주시  인왕동   839-1  None  None
```

3.9.3 탐색

시작 글자, 끝나는 글자, 포함하는 글자를 기준으로 데이터프레임을 탐색할 수 있습니다. 탐색의 결과는 True/ False로 반환됩니다.

```
In [ ]: # 서울로 시작하는 행 탐색
landmark['location'].str.startswith('서울')
```

```
Out[ ]: 0      True
1      False
2      False
Name: location, dtype: bool
```

```
In [ ]: # 1로 끝나는 행 탐색
landmark['location'].str.endswith('1')
```

```
Out[ ]: 0      True
        1      False
        2      True
        Name: location, dtype: bool
```

```
In [ ]: # 경북이 포함된 행 탐색
        landmark['location'].str.contains('경북')
```

```
Out[ ]: 0      False
        1      True
        2      True
        Name: location, dtype: bool
```

3.9.4 치환

특정한 데이터를 다른 데이터로 치환합니다.

```
In [ ]: # 경북을 경상북도로 변경
        landmark['location'].str.replace('경북', '경상북도')
```

```
Out[ ]: 0      서울 종로구 사직로 161
        1      경상북도 포항시 남구 호미곶면 대보리 150
        2      경상북도 경주시 인왕동 839-1
        Name: location, dtype: object
```

3.10 날짜 데이터 핸들링

날짜와 시간을 다루기 위해 datetime 모듈의 datetime 패키지를 이용합니다.

3.10.1 현재 날짜 사용하기

```
In [ ]: from datetime import datetime
        datetime.today()
```

```
Out[ ]: datetime.datetime(2024, 6, 20, 16, 25, 42, 104759)
```

```
In [ ]: # 현재 시간에서 년, 월, 일 추출
        [datetime.today().year, datetime.today().month, datetime.today().day]
```

```
Out[ ]: [2024, 6, 20]
```

3.10.2 날짜 형식의 변환

1. 문자 형식의 데이터프레임을 날짜로 변환

```
In [ ]: # 데이터프레임 생성
        df = pd.DataFrame({'Datetime':['20230101', '20230102', '20230103', '20230104']})
        df
```

Out[]: **Datetime**

0 20230101

1 20230102

2 20230103

3 20230104

```
In [ ]: # 데이터프레임 컬럼 타입 확인  
df.dtypes
```

Out[]: Datetime object
dtype: object

```
In [ ]: # 문자열을 날짜로 변환  
df['Datetime'] = pd.to_datetime(df['Datetime'], format='%Y-%m-%d')  
df
```

Out[]: **Datetime**

0 2023-01-01

1 2023-01-02

2 2023-01-03

3 2023-01-04

```
In [ ]: # 데이터프레임 컬럼 타입 확인  
df.dtypes
```

Out[]: Datetime datetime64[ns]
dtype: object

2. 문자를 날짜로 변환

```
In [ ]: datetime.strptime('20230101', '%Y%m%d')
```

Out[]: datetime.datetime(2023, 1, 1, 0, 0)

3. 날짜를 문자로 변환

```
In [ ]: time = datetime.today()  
time.strftime('%Y-%m-%d %H:%M:%S')
```

Out[]: '2024-06-20 16:25:42'

🔥 날짜 포맷

구 분	내 용
%d	0을 채운 10진수 표기로 날짜를 표시
%m	0을 채운 10진수 표기로 월을 표시
%y	0을 채운 10진수 표기로 2자리 연도를 표시
%Y	0을 채운 10진수 표기로 4자리 연도를 표시
%H	0을 채운 10진수 표기로 시간(24시간 표기)을 표시
%I	0을 채운 10진수 표기로 시간(12시간 표기)을 표시
%M	0을 채운 10진수 표기로 분을 표시
%S	0을 채운 10진수 표기로 초를 표시
%f	0을 채운 10진수 표기로 마이크로 초(6자리)를 표시
%A	locale 요일을 표시
%a	locale 요일(단축 표기)을 표시
%B	locale 월을 표시
%b	locale 월(단축 표기)을 표시
%j	0을 채운 10진수 표기로 연 중 몇 번째 일인지 표시
%U	0을 채운 10진수 표기로 연 중 몇 번째 주인지 표시(일요일 시작 기준)

3.10.3 날짜 데이터의 연산

- datetime 모듈의 timedelta 패키지를 사용하여 시간 연산이 가능함

timedelta

```
datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0,
minutes=0, hours=0, weeks=0)
```

```
In [ ]: from datetime import timedelta
# 오늘로부터 1주일 1일 후 날짜 연산
time = datetime.today()
time + timedelta(days=1, weeks=1)
```

```
Out[ ]: datetime.datetime(2024, 6, 28, 16, 25, 42, 211577)
```