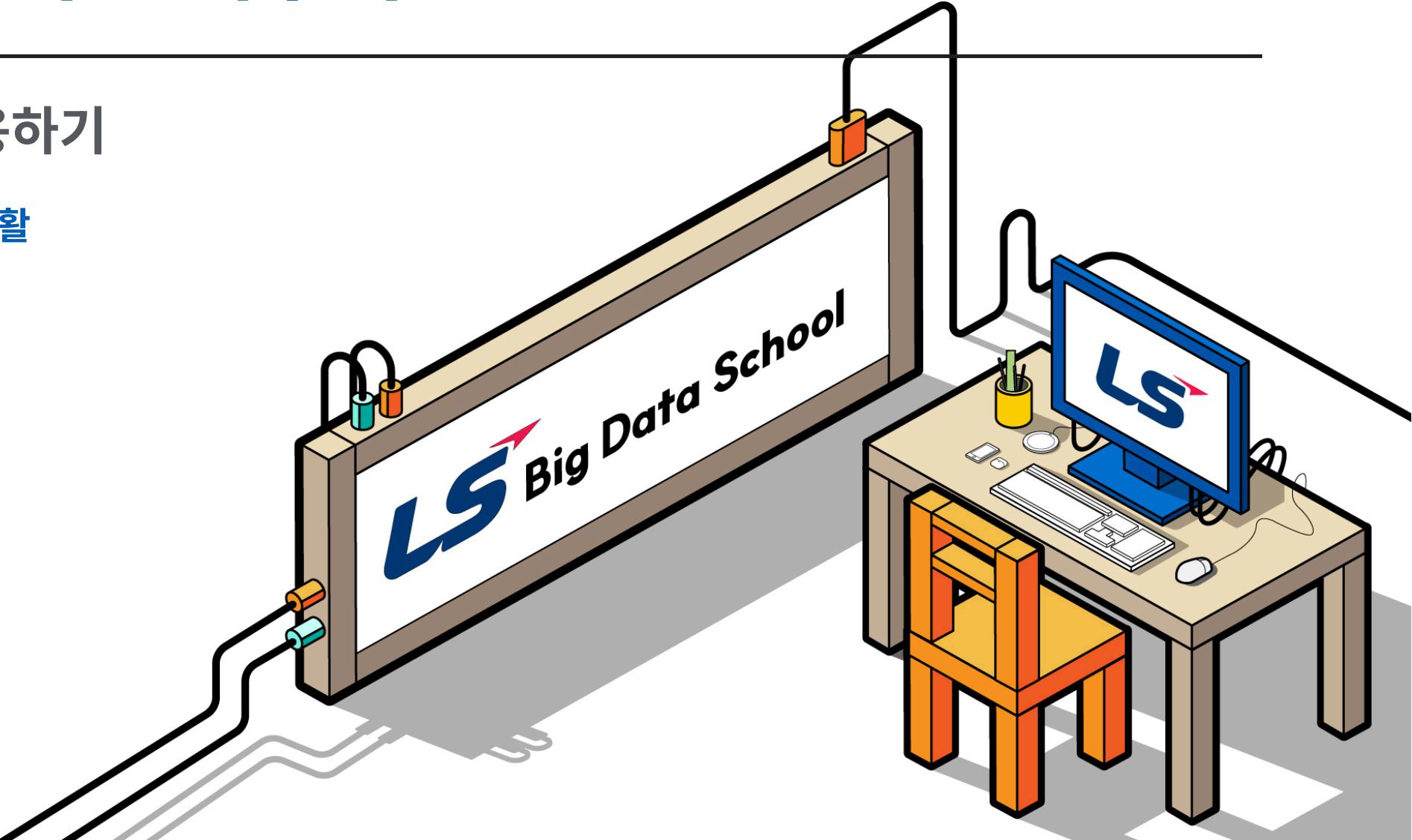


파이썬 기초 배우기

판다스 활용하기

슬기로운통계생활



코스 훑어보기.

파이썬의 판다스 라이브러리에 대해 학습합니다.



데이터 프레임(DataFrame)이란 무엇일까?

행렬의 경우 구성원들이 모두 같은 데이터 타입이어야만 합니다. 하지만 데이터 프레임의 경우는 각 열에 들어있는 데이터 타입이 달라도 됩니다.

```
import numpy as np

# 행렬을 생성할 때 모든 요소는 같은 데이터 타입이어야 합니다.
matrix = np.array([[ '1', '2', '3', '4', '5'], [6, 7, 8, 9, 10]])
print(matrix)
```

```
## [[ '1' '2' '3' '4' '5']
##   ['6' '7' '8' '9' '10']]
```

데이터 프레임(DataFrame) 예시



```
import pandas as pd

# 데이터 프레임 생성
df = pd.DataFrame({
    'col1': ['one', 'two', 'three', 'four',
    'col2': [6, 7, 8, 9, 10]
})
print(df)
```

```
##      col1  col2
## 0      one     6
## 1     two     7
## 2   three     8
## 3   four     9
## 4   five    10
```

```
print(df['col1'].dtype)
```

```
## object
```

```
print(df['col2'].dtype)
```

```
## int64
```

```
print(df.shape)
```

```
## (5, 2)
```



시리즈(Series)란 무엇일까?

데이터 프레임은 2차원 구조였습니다. 반면 시리즈는 1차원 구조를 갖습니다. 따라서 따라서 하나의 데이터 타입을 갖습니다. 데이터 프레임은 여러 개 시리즈의 집합이라고 볼 수 있습니다.

```
import pandas as pd  
data = [10, 20, 30]  
df_s = pd.Series(data, index=['one', 'two',  
print(df_s)
```

```
## one      10  
## two      20  
## three    30  
## Name: count, dtype: int64
```

```
print(df_s.dtype)
```

```
## int64
```

```
print(df_s.shape)
```

```
## (3, )
```



시리즈(Series)의 특징

시리즈는 `columns`을 통해 열 이름을 확인할 수 없습니다. 대신 `name`을 통해 시리즈 이름을 확인할 수 있습니다.

```
df_s.columns
```

```
## AttributeError: 'Series' object has no attribute 'columns'
```

```
df_s.name
```

```
## 'count'
```



빈 데이터 프레임 만들기

파이썬에서 빈 데이터 프레임을 만드는 방법은 pandas의 DataFrame 클래스를 사용하는 것입니다. 각 열의 이름과 데이터 타입을 지정해줄 수 있습니다. 다음의 코드를 보겠습니다.

```
my_df = pd.DataFrame({  
    '실수변수': pd.Series(dtype='float'),  
    '정수변수': pd.Series(dtype='int'),  
    '범주형변수': pd.Series(dtype='category'),  
    '논리변수': pd.Series(dtype='bool'),  
    '문자열변수': pd.Series(dtype='str'))}  
  
print(my_df.dtypes)
```

```
## 실수변수      float64  
## 정수변수      int64  
## 범주형변수    category  
## 논리변수      bool  
## 문자열변수    object  
## dtype: object
```

위의 데이터 타입에 대응하는 정보유형을 정리해보면 다음과 같습니다.

- float: 실수 정보
- int: 정수 정보
- category: 범주형 정보
- bool: 논리값 정보
- str: 문자열 정보



데이터 프레임 채우면서 만들기

데이터 프레임을 만들 때 각 열에 들어갈 리스트나 배열을 이름과 값을 연결하여 DataFrame 클래스에 차례대로 넣어줍니다.

```
# 데이터 프레임 생성  
my_df = pd.DataFrame({  
    'name': ['issac', 'bomi'],  
    'birthmonth': [5, 4]  
})  
print(my_df)
```

```
##      name birthmonth  
## 0    issac          5  
## 1    bomi          4
```

```
print(my_df.dtypes)  
  
## name          object  
## birthmonth    int64  
## dtype: object
```

```
print(my_df.shape)  
  
## (2, 2)
```

csv 파일로 읽어오기



중간고사 성적 데이터

주어진 링크를 사용하여 "examscore.csv" 파일을 다운로드하고 데이터를 불러옵니다. pandas 패키지의 `read_csv()` 함수를 사용하여 CSV 파일을 읽을 수 있습니다.

```
import pandas as pd  
url = "https://bit.ly/examscore-csv"  
mydata = pd.read_csv(url)  
print(mydata.head())
```

```
##      student_id gender  midterm  final  
## 0            1      F       38      46  
## 1            2      M       42      67  
## 2            3      F       53      56  
## 3            4      M       48      54  
## 4            5      M       46      39
```



데이터 프레임 인덱싱(indexing)

데이터 프레임 인덱싱(indexing)은 데이터 프레임 내의 특정 데이터를 효율적으로 필터링 및 조작할 수 있게 합니다. 데이터 프레임 인덱싱 방법에는 `[]` 연산자를 활용한 방법, `.iloc[]`, `.loc[]`를 활용한 방법 등이 있습니다.

인덱스(index)

열(Column)

행(Row)

	student_id	gender	midterm	final
0	1	F	38	46
1	2	M	42	67
2	3	F	53	56
3	4	M	48	54

→ `df.iloc[0]`

`df['student_id']`



[] 연산자를 이용한 필터링

[] 연산자를 사용하여 데이터 프레임의 특정 열에 접근할 수 있습니다.

```
print(mydata[ 'gender' ].head())
```

```
## 0      F
## 1      M
## 2      F
## 3      M
## 4      M
## Name: gender, dtype: object
```



[] 연산자를 이용한 필터링

한번에 여러 칼럼을 선택하고 싶은 경우, 리스트 형태로 넣어줍니다.

```
print(mydata[ [ 'gender' , 'midterm' ] ].head())
```

```
##   gender  midterm
## 0      F        38
## 1      M        42
## 2      F        53
## 3      M        48
## 4      M        46
```

[] 연산자를 이용한 필터링



특정 행을 선택하고 싶은 경우 조건식을 넣어주면 됩니다.

```
print(mydata[mydata['midterm'] <= 15].head())
```

```
##      student_id gender midterm final
## 19          20       M        9     33
## 21          22       M       15     12
```

데이터 프레임을 행렬 형태로 접근할 수도 있습니다. `iloc[]`를 사용하여 원하는 열이나 행에 접근할 수 있습니다.



iloc[]를 이용한 필터링

iloc(integer-based indexing)는 정수 기반 인덱싱으로 행과 열의 정수 위치를 활용하여 데이터를 필터링합니다.

첫 번째 열을 필터링합니다.

```
# 첫 번째 열에 접근  
print(mydata.iloc[:, 0].head(2))
```

```
## 0      1  
## 1      2  
## Name: student_id, dtype: int64
```

iloc[]를 이용한 필터링



단순 정수값을 활용하여 필터링할 경우 Series으로 변환됩니다.

```
print(mydata.iloc[:, 0].shape)
```

```
## (30, )
```

```
print(mydata.iloc[:, 0].info())
```

```
## <class 'pandas.core.series.Series'>
## RangeIndex: 30 entries, 0 to 29
## Series name: student_id
## Non-Null Count Dtype
## ----- -----
## 30 non-null    int64
## dtypes: int64(1)
## memory usage: 372.0 bytes
## None
```



iloc[]를 이용한 필터링

첫 번째, 두 번째 열을 필터링합니다. 리스트 형태로 필터링할 경우 DataFrame으로 변환됩니다.

```
# 첫 번째, 두 번째 열에 접근  
print(mydata.iloc[:, [0, 1]].head(2))
```

```
##   student_id gender  
## 0           1      F  
## 1           2      M
```

```
print(mydata.iloc[:, [0, 1]].shape)  
  
## (30, 2)
```

```
print(mydata.iloc[:, [0, 1]].info())
```

```
## <class 'pandas.core.frame.DataFrame'>  
## RangeIndex: 30 entries, 0 to 29  
## Data columns (total 2 columns):  
## #   Column   Non-Null Count  Dtype    
## ---  -----  --  --  --  
## 0   student_id  30 non-null    int64  
## 1   gender      30 non-null    object  
## dtypes: int64(1), object(1)  
## memory usage: 612.0+ bytes  
## None
```

squeeze() 활용하기



DataFrame을 Series로 변환하기 위해서 squeeze()를 활용합니다. 첫 번째 열을 필터링합니다. 리스트 형태로 데이터를 필터링했으므로, 하나의 열이어도 DataFrame으로 변환됩니다.

```
# 첫 번째 열에 접근  
print(mydata.iloc[:, [0]].head())
```

```
##      student_id  
## 0            1  
## 1            2  
## 2            3  
## 3            4  
## 4            5
```

```
print(mydata.iloc[:, [0]].info())
```

```
## <class 'pandas.core.frame.DataFrame'>  
## RangeIndex: 30 entries, 0 to 29  
## Data columns (total 1 columns):  
##   #   Column      Non-Null Count  Dtype    
##   ---  -----  -----    
##   0   student_id  30 non-null  int64  
## dtypes: int64(1)  
## memory usage: 372.0 bytes  
## None
```



squeeze() 활용하기

DataFrame을 Series로 변환합니다. Series와 DataFrame은 차원 구조가 다르므로, 데이터 전 처리 시 다르게 활용됩니다. 따라서 DataFrame과 Series를 정확히 구분하는 것이 중요합니다.

```
mydata.iloc[:, 0].squeeze().head()
```

```
## 0    1
## 1    2
## 2    3
## 3    4
## 4    5
## Name: student_id, dtype: int64
```

```
print(my_df.iloc[:, 0].squeeze().shape)
```

```
## (2, )
```

```
print(my_df.iloc[:, 0].squeeze().info())
```

```
## <class 'pandas.core.series.Series'>
## RangeIndex: 2 entries, 0 to 1
## Series name: name
## Non-Null Count Dtype
## ----- 
## 2 non-null      object
## dtypes: object(1)
## memory usage: 148.0+ bytes
## None
```



loc[]를 이용한 필터링

loc(label-based indexing)는 라벨 기반 인덱싱으로 행과 열의 라벨을 활용하여 데이터를 필터링 합니다. 다음과 같은 형식으로 조건을 만족하는 행들을 선택하는 코드를 생각해보겠습니다.

```
mydata[mydata[ 'midterm' ] <= 15].head()
```

```
##      student_id gender midterm final
## 19          20      M        9     33
## 21          22      M       15     12
```

위의 코드를 loc[] 함수를 사용하면 다음과 같이 나타낼 수 있습니다.

```
mydata.loc[mydata[ 'midterm' ] <= 15].head()
```

```
##      student_id gender midterm final
## 19          20      M        9     33
## 21          22      M       15     12
```



loc[]를 이용한 필터링

특정 열을 지정하지 않을 경우 전체 열이 선택됩니다.

```
mydata.loc[mydata['midterm'] <= 15, :].head()
```

```
##      student_id gender midterm final
## 19          20       M        9     33
## 21          22       M       15     12
```

조건을 만족하는 행과 열을 함께 필터링할 수 있습니다.

```
mydata.loc[mydata['midterm'] <= 15, ['student_id', 'final']].head()
```

```
##      student_id final
## 19          20     33
## 21          22     12
```



isin[] 활용하기

isin()은 특정 값이 데이터프레임 내에 존재하는지 확인하고, 조건을 만족하는 행을 필터링하는데 사용됩니다. loc[]과 함께 활용하면 특정 열의 여러 값을 한 번에 필터링할 수 있습니다.

```
# 특정 값이 포함된 행 필터링  
mydata[mydata['midterm'].isin([28, 38, 52])].head()
```

```
##      student_id gender midterm final  
## 0            1       F      38     46  
## 8            9       M      28     25  
## 9           10       M      38     59  
## 23          24       M      28     55  
## 27          28       F      52     66
```



isin[] 활용하기

특정 값이 포함된 행을 필터링하면서 원하는 열만 선택할 수도 있습니다. student_id와 final 컬럼만 출력하며, midterm 점수가 28, 38, 52인 경우만 선택됩니다.

```
mydata.loc[mydata['midterm'].isin([28, 38, 52]), ['student_id', 'final']].head()
```

```
##      student_id  final
## 0            1     46
## 8            9     25
## 9           10     59
## 23          24     55
## 27          28     66
```

isin[] 활용하기



isin() 앞에 ~(틸드)를 붙이면 특정 값을 제외할 수 있습니다. midterm 점수가 28, 38, 52인 행을 제외한 나머지 데이터를 선택합니다.

```
mydata.loc[~mydata['midterm'].isin([28, 38, 52])].head()
```

```
##      student_id gender  midterm  final
## 1            2       M       42     67
## 2            3       F       53     56
## 3            4       M       48     54
## 4            5       M       46     39
## 5            6       M       51     74
```



완전한 표본 체크하기

만약 데이터 프레임의 행렬이 다음과 같이 빈 칸이 삽입되어 있다고 생각해봅시다.

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

예제 데이터 프레임 생성

```
mydata = pd.DataFrame({  
    'student_id': [1, 2, 3, 4, 5],  
    'gender': ['F', 'M', 'F', 'M', 'M'],  
    'midterm': [38, 42, 53, 48, 46],  
    'final': [46, 67, 56, 54, 39]  
})
```

일부 데이터를 NA로 설정

```
mydata.iloc[0, 1] = np.nan  
mydata.iloc[4, 0] = np.nan
```

```
print(mydata)
```

```
##      student_id  gender  midterm  final  
## 0          1.0    NaN     38     46  
## 1          2.0     M     42     67  
## 2          3.0     F     53     56  
## 3          4.0     M     48     54  
## 4          NaN     M     46     39
```



완전한 표본 체크하기

`isna()` 함수로 빈칸이 들어있는 부분을 열 별로 체크할 수 있습니다.

```
print("gender 열의 빈칸 개수:", mydata['gender'].isna().sum())
```

```
## gender 열의 빈칸 개수: 1
```

```
print("student_id 열의 빈칸 개수:", mydata['student_id'].isna().sum())
```

```
## student_id 열의 빈칸 개수: 1
```

하지만, 빈 칸을 포함하지 않는 완전한 행을 가진 데이터를 얻으려면 `dropna()` 메서드를 사용합니다. `dropna()` 메서드는 모든 열이 꽉 채워져 있는 완전한 행들 만을 반환합니다.

NA가 제거된 꽉 찬 데이터 프레임 얻어내기



```
complete_row = mydata.dropna()  
print("완전한 행의 수:", len(complete_row))
```

```
## 완전한 행의 수: 3
```

```
print(complete_row)
```

```
##      student_id gender  midterm  final  
## 1        2.0      M       42     67  
## 2        3.0      F       53     56  
## 3        4.0      M       48     54
```



구성원소 추가/삭제/변경

변경 및 추가

[] 기호를 사용하여 새로운 열을 추가하거나 기존 열을 변경할 수 있습니다. 예를 들어, `mydata` 데이터 프레임에 `midterm` 열과 `final` 열의 값을 더하여 `total` 열을 추가하고, `total` 열의 앞부분 몇 개 행을 확인하려면 아래와 같이 코드를 작성합니다.

```
mydata['total'] = mydata['midterm'] + mydata['final']
print(mydata.iloc[0:3, [3, 4]])
```

```
##      final  total
## 0       46     84
## 1       67    109
## 2       56    109
```



변경 및 추가

`pd.concat()` 함수를 사용하여 새로운 열을 추가할 수도 있습니다. 예를 들어, `mydata` 데이터 프레임에 `total` 열의 값의 절반을 나타내는 `average` 열을 추가하고, 결과를 확인하려면 아래와 같이 코드를 작성합니다.

```
mydata = pd.concat([mydata, (mydata['total'] / 2).rename('average')], axis=1)
print(mydata.head())
```

```
##      student_id gender  midterm  final  total  average
## 0           1.0     NaN      38      46     84    42.0
## 1           2.0      M      42      67    109    54.5
## 2           3.0      F      53      56    109    54.5
## 3           4.0      M      48      54    102    51.0
## 4          NaN      M      46      39     85    42.5
```



변경 및 추가

마지막 칼럼 이름을 `rename()` 사용하여 다음과 같이 수정해줄 수 있습니다.

```
mydata.rename(columns={'average': 'my_average'}, inplace=True)  
print(mydata.head())
```

```
##      student_id gender  midterm  final  total  my_average  
## 0          1.0     NaN      38      46     84       42.0  
## 1          2.0      M      42      67    109       54.5  
## 2          3.0      F      53      56    109       54.5  
## 3          4.0      M      48      54    102       51.0  
## 4          NaN      M      46      39     85       42.5
```



del을 사용한 삭제

Python에서 데이터 프레임의 열을 삭제하는 방법으로 `del`을 사용할 수 있습니다. 예를 들어, `mydata` 데이터 프레임에서 `gender` 열을 삭제하려면 아래와 같이 코드를 작성합니다.

```
del mydata[ 'gender' ]  
print(mydata.head())
```

```
##      student_id  midterm  final  total  my_average  
## 0          1.0      38      46     84       42.0  
## 1          2.0      42      67    109       54.5  
## 2          3.0      53      56    109       54.5  
## 3          4.0      48      54    102       51.0  
## 4          NaN      46      39     85       42.5
```



pd.concat() 함수

pd.concat() 함수는 pandas에서 데이터 프레임이나 시리즈를 연결(concatenate)하는 데 사용됩니다. 이 함수는 여러 데이터 프레임이나 시리즈를 하나로 합치는 데 유용합니다. pd.concat() 함수는 여러 가지 옵션을 제공하여 다양한 방식으로 데이터를 합칠 수 있습니다.

주요 옵션

- `objs`: 연결할 데이터 프레임이나 시리즈의 리스트
- `axis`: 0 또는 'index'는 행 방향으로 연결, 1 또는 'columns'는 열 방향으로 연결 (기본값은 0)
- `join`: 'outer'(기본값) 또는 'inner'. 조인 방식 결정
- `ignore_index`: True로 설정하면 새로운 인덱스를 생성하여 기존 인덱스를 무시
- `keys`: 계층적 인덱스를 생성하는 데 사용되는 키 값들
- `sort`: True로 설정하면 조인 축을 따라 데이터를 정렬



예제 1: 두 개 데이터 프레임 연결하기

`pd.concat()` 함수는 기본적으로 두 데이터 프레임을 연결합니다. 기본 설정은 `axis=0`이므로 행 방향으로 연결됩니다. 결과적으로 `df1`의 행 아래에 `df2`의 행이 추가됩니다. 만약, 옆으로 합치고 싶은 경우에는 `axis` 옵션을 열 방향으로 바꿔주면 됩니다.

```
import pandas as pd

df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2'],
    'B': ['B0', 'B1', 'B2']
})

df2 = pd.DataFrame({
    'A': ['A3', 'A4', 'A5'],
    'B': ['B3', 'B4', 'B5']
})

result = pd.concat([df1, df2])
```

```
print(result)
```

```
##      A      B
## 0    A0    B0
## 1    A1    B1
## 2    A2    B2
## 0    A3    B3
## 1    A4    B4
## 2    A5    B5
```



예제 2: 열 방향으로 연결

다음은 두 데이터 프레임을 열 방향을 기준으로 합치는 코드입니다. `axis=1`의 옵션으로 인하여 df3가 행이 아닌 열 방향으로 합쳐진 것을 확인 할 수 있습니다.

```
df3 = pd.DataFrame({  
    'C': ['C0', 'C1', 'C2'],  
    'D': ['D0', 'D1', 'D2']  
})  
  
result = pd.concat([df1, df3], axis=1)  
print(result)
```

```
##      A    B    C    D  
## 0  A0  B0  C0  D0  
## 1  A1  B1  C1  D1  
## 2  A2  B2  C2  D2
```



예제 3: ignore_index 옵션 사용

기본 설정의 행방향으로 합쳤을 경우, 행 번호가 중복되어 출력되는 것을 볼 수 있습니다. 이런 행번호를 정리할 때 사용할 수 있는 옵션은 `ignore_index=True`입니다.

```
result = pd.concat([df1, df2])
print(result)
```

```
##      A    B
## 0   A0   B0
## 1   A1   B1
## 2   A2   B2
## 0   A3   B3
## 1   A4   B4
## 2   A5   B5
```

```
result = pd.concat([df1, df2], ignore_index=True)
print(result)
```

```
##      A    B
## 0   A0   B0
## 1   A1   B1
## 2   A2   B2
## 3   A3   B3
## 4   A4   B4
## 5   A5   B5
```



예제 4: join 옵션 사용

df4는 열 'A', 'B' 및 'C'를 가지고 있으며 각 열은 세 개의 문자열 값을 포함하는 데이터 프레임입니다.

- inner: 공통된 열만 합치기

```
df4 = pd.DataFrame({  
    'A': ['A2', 'A3', 'A4'],  
    'B': ['B2', 'B3', 'B4'],  
    'C': ['C2', 'C3', 'C4']  
})  
print(df1)
```

```
##      A      B  
## 0  A0  B0  
## 1  A1  B1  
## 2  A2  B2
```

```
print(df4)
```

```
##      A      B      C  
## 0  A2  B2  C2  
## 1  A3  B3  C3  
## 2  A4  B4  C4
```



예제 4: join 옵션 사용

`pd.concat()` 함수는 두 데이터 프레임을 내부 결합하여 연결합니다. `join='inner'` 옵션을 사용하여 두 데이터 프레임의 공통 열만 포함하는 결합을 수행합니다.

```
result = pd.concat([df1, df4], join='inner')
print(result)
```

```
##      A    B
## 0   A0   B0
## 1   A1   B1
## 2   A2   B2
## 0   A2   B2
## 1   A3   B3
## 2   A4   B4
```



예제 4: join 옵션 사용

- outer: 모든 열 합치기

join='outer'는 데이터 프레임을 결합할 때, 모든 열을 포함시키는 외부 결합(outer join)을 의미합니다. 이는 두 데이터 프레임의 모든 열을 포함하며, 하나의 데이터 프레임에만 존재하는 열은 NaN으로 채워집니다.

```
import pandas as pd

# 첫 번째 데이터 프레임 생성
df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2'],
    'B': ['B0', 'B1', 'B2']
})
```

```
# 네 번째 데이터 프레임 생성
df4 = pd.DataFrame({
    'A': ['A2', 'A3', 'A4'],
    'B': ['B2', 'B3', 'B4'],
    'C': ['C2', 'C3', 'C4']
})
```



예제 4: join 옵션 사용

결과 데이터 프레임에는 df1과 df4의 모든 열인 'A', 'B' 및 'C'가 포함되어 있습니다. join='outer' 옵션을 사용했기 때문에 열 'C'는 df1에서 존재하지 않기 때문에 NaN 값으로 채워졌습니다. 이는 외부 결합의 특성으로, 두 데이터 프레임의 모든 열을 포함하며 하나의 데이터 프레임에만 존재하는 열은 NaN으로 채웁니다.

```
# 두 데이터 프레임을 외부 결합하여 연결
result = pd.concat([df1, df4], join='outer')
print(result)
```

```
##      A      B      C
## 0    A0    B0    NaN
## 1    A1    B1    NaN
## 2    A2    B2    NaN
## 0    A2    B2    C2
## 1    A3    B3    C3
## 2    A4    B4    C4
```



예제 5: keys 옵션 사용

df1과 df2는 같은 칼럼 이름을 가지고 있는 데이터 프레임입니다. 이것들을 합칠 때 각 행이 어느 데이터 프레임에서 왔는지 기록하고 싶은 경우가 있겠죠?

- df1은 열 'A'와 'B'를 가지고 있으며, 각 열은 세 개의 문자열 값을 포함하는 데이터 프레임입니다.
- df2도 열 'A'와 'B'를 가지고 있으며, 각 열은 세 개의 문자열 값을 포함하는 데이터 프레임입니다.

```
print(df1)
```

```
##      A    B  
## 0   A0   B0  
## 1   A1   B1  
## 2   A2   B2
```

```
print(df2)
```

```
##      A    B  
## 0   A3   B3  
## 1   A4   B4  
## 2   A5   B5
```



예제 5: keys 옵션 사용

이런 경우 keys 옵션을 사용해서 기록해놓을 수 있습니다. keys 옵션을 사용하면, 연결된 데이터 프레임의 원본 출처를 식별하는 멀티인덱스가 생성됩니다. 여기서 'df1'과 'df2'가 각각 df1과 df2 데이터 프레임의 출처를 나타냅니다.

```
result = pd.concat([df1, df2], keys=['key1', 'key2'])  
print(result)
```

```
##          A    B  
## key1  0  A0  B0  
##      1  A1  B1  
##      2  A2  B2  
## key2  0  A3  B3  
##      1  A4  B4  
##      2  A5  B5
```



예제 5: keys 옵션 사용

다음은 key2에 해당하는 데이터 프레임의 2번째 3번째 행을 선택하는 코드입니다.

```
df1_rows = result.loc['key2'].iloc[1:3]  
print(df1_rows)
```

```
##      A    B  
## 1  A4  B4  
## 2  A5  B5
```

판다스 데이터 프레임에서 이용 가능한 메서드 정리



메서드	설명
head()	데이터 프레임의 처음 몇 개의 행을 반환
tail()	데이터 프레임의 마지막 몇 개의 행을 반환
describe()	데이터 프레임의 요약 통계를 반환
info()	데이터 프레임의 정보(컬럼, 타입 등)
sort_values()	특정 열을 기준으로 데이터 프레임 정렬
groupby()	특정 열을 기준으로 데이터 프레임 그룹화
mean()	데이터 프레임의 평균 계산
sum()	데이터 프레임의 합계 계산
merge()	두 데이터 프레임 병합
pivot_table()	피벗 테이블 생성



팔머 펭귄 데이터

팔머 펭귄 데이터는 남극의 팔머 제도에 서식하는 세 종의 펭귄(Adelie, Chinstrap, Gentoo)에 대한 다양한 생물학적 측정값을 포함하는 데이터셋입니다. 이 데이터는 데이터 과학 교육 및 실습에 자주 사용됩니다. 각 행은 하나의 펭귄을 나타내며, 다음과 같은 열로 구성되어 있습니다:

- species: 펭귄의 종(Adelie, Chinstrap, Gentoo)
- island: 펭귄이 서식하는 섬(Biscoe, Dream, Torgersen)
- bill_length_mm: 부리의 길이(밀리미터 단위)
- bill_depth_mm: 부리의 깊이(밀리미터 단위)
- flipper_length_mm: 날개의 길이(밀리미터 단위)
- body_mass_g: 몸무게(그램 단위)
- sex: 펭귄의 성별(Male, Female)
- year: 데이터가 수집된 연도

팔머 펭귄 데이터는 다양한 데이터 분석 및 시각화 기법을 실습하는데 유용합니다. 이를 통해 데이터의 분포를 살펴보고, 종별로 펭귄의 생물학적 특성을 비교하는 등의 작업을 수행할 수 있습니다.

```
df = pd.read_csv('./data/penguins.csv')
```



head()

head() 메서드는 데이터 프레임의 처음 몇 개의 행을 반환합니다. 기본적으로 5개의 행을 반환합니다.

```
import pandas as pd  
# 처음 5개의 행 반환  
print(df.head())
```

```
##   species     island bill_length_mm ... flipper_length_mm body_mass_g    sex  
## 0   Adelie  Torgersen        39.1 ...          181.0    3750.0  Male  
## 1   Adelie  Torgersen        39.5 ...          186.0    3800.0 Female  
## 2   Adelie  Torgersen        40.3 ...          195.0    3250.0 Female  
## 3   Adelie  Torgersen        NaN ...           NaN      NaN     NaN  
## 4   Adelie  Torgersen        36.7 ...          193.0    3450.0 Female  
##  
## [5 rows x 7 columns]
```



tail()

tail() 메서드는 head()와 대응하는 메서드로 데이터 프레임의 마지막 몇 개의 행을 반환합니다. 기본적으로 5개의 행을 반환합니다.

```
# 마지막 5개의 행 반환  
print(df.tail())
```

```
##      species  island bill_length_mm ... flipper_length_mm body_mass_g    sex  
## 339  Gentoo  Biscoe          NaN ...           NaN       NaN     NaN  
## 340  Gentoo  Biscoe        46.8 ...         215.0     4850.0 Female  
## 341  Gentoo  Biscoe        50.4 ...         222.0     5750.0  Male  
## 342  Gentoo  Biscoe        45.2 ...         212.0     5200.0 Female  
## 343  Gentoo  Biscoe        49.9 ...         213.0     5400.0  Male  
##  
## [5 rows x 7 columns]
```



describe()

describe() 메서드는 데이터 프레임의 요약 통계를 반환합니다. 수치형 데이터에 대해 count, mean, std, min, 25%, 50%, 75%, max 값을 제공합니다.

```
# 요약 통계 반환  
print(df.describe())
```

```
##          bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  
## count      342.000000      342.000000      342.000000      342.000000  
## mean       43.921930      17.151170     200.915205     4201.754386  
## std        5.459584      1.974793      14.061714      801.954536  
## min        32.100000      13.100000     172.000000     2700.000000  
## 25%        39.225000      15.600000     190.000000     3550.000000  
## 50%        44.450000      17.300000     197.000000     4050.000000  
## 75%        48.500000      18.700000     213.000000     4750.000000  
## max        59.600000      21.500000     231.000000     6300.000000
```



describe()

- count: 각 변수에 대한 관측치의 수입니다. 모든 변수에 대해 342개의 데이터가 있습니다.
- mean: 각 변수의 평균값을 나타냅니다.
- std: 각 변수의 표준편차를 나타냅니다. 이는 데이터가 평균값을 중심으로 얼마나 퍼져 있는지를 보여줍니다.
- min: 각 변수의 최소값입니다.
- 25%: 각 변수의 하위 25% 백분위수(Q1)입니다. 이는 데이터의 하위 25%가 이 값 이하임을 의미합니다.
- 50% (median): 각 변수의 중앙값입니다. 이는 데이터의 중간값을 나타내며, 데이터가 정렬되었을 때 중앙에 위치한 값입니다.
- 75%: 각 변수의 상위 25% 백분위수(Q3)입니다. 이는 데이터의 상위 25%가 이 값 이상임을 의미합니다.
- max: 각 변수의 최대값입니다.

해석



- 부리 길이(bill_length_mm): 평균 43.92 mm, 최소 32.1 mm, 최대 59.6 mm로 비교적 큰 변동성을 보입니다. 표준편차는 5.46 mm입니다.
- 부리 깊이(bill_depth_mm): 평균 17.15 mm, 최소 13.1 mm, 최대 21.5 mm로 변동이 적은 편입니다. 표준편차는 1.97 mm입니다.
- 날개 길이(flipper_length_mm): 평균 200.92 mm, 최소 172 mm, 최대 231 mm로 큰 변동성을 보입니다. 표준편차는 14.06 mm입니다.
- 몸무게(body_mass_g): 평균 4201.75 g, 최소 2700 g, 최대 6300 g로 매우 큰 변동성을 보입니다. 표준편차는 801.95 g입니다.

이 통계량들은 각 변수의 중심 경향 및 변동성을 이해하는 데 도움을 줍니다. 예를 들어, 부리 길이와 날개 길이는 펭귄 종 간의 차이를 나타낼 수 있는 중요한 생물학적 특성일 수 있습니다.



info()

info() 메서드는 데이터 프레임 정보(컬럼 이름, 데이터 타입, 누락된 값의 개수 등)를 반환합니다.

```
print(df.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 344 entries, 0 to 343
## Data columns (total 7 columns):
##   #   Column            Non-Null Count  Dtype  
##   --  --  
##   0   species          344 non-null    object 
##   1   island            344 non-null    object 
##   2   bill_length_mm    342 non-null    float64
##   3   bill_depth_mm     342 non-null    float64
##   4   flipper_length_mm 342 non-null    float64
##   5   body_mass_g       342 non-null    float64
##   6   sex               333 non-null    object 
## dtypes: float64(4), object(3)
## memory usage: 18.9+ KB
```

데이터 프레임 정보 해석



- 클래스 타입: pandas.core.frame.DataFrame 클래스의 객체로, 판다스의 데이터 프레임입니다.
- 행 인덱스: 344개의 행이 있으며, 인덱스는 0부터 343까지입니다.
- 열 정보: 총 7개의 열이 있습니다.
- species: 344개의 값이 있으며, 데이터 타입은 object (문자열)입니다.
- island: 344개의 값이 있으며, 데이터 타입은 object (문자열)입니다.
- bill_length_mm: 342개의 값이 있으며, 데이터 타입은 float64 (실수)입니다. 2개의 결측치 (NA)가 있습니다.
- bill_depth_mm: 342개의 값이 있으며, 데이터 타입은 float64 (실수)입니다. 2개의 결측치 (NA)가 있습니다.
- flipper_length_mm: 342개의 값이 있으며, 데이터 타입은 float64 (실수)입니다. 2개의 결측치(NA)가 있습니다.
- body_mass_g: 342개의 값이 있으며, 데이터 타입은 float64 (실수)입니다. 2개의 결측치(NA)가 있습니다.
- sex: 333개의 값이 있으며, 데이터 타입은 object (문자열)입니다. 11개의 결측치(NA)가 있습니다.



데이터 프레임 정보 해석

- 데이터 타입 요약:
 - float64 타입의 열이 4개입니다.
 - object 타입의 열이 3개입니다.
- 메모리 사용량: 데이터 프레임이 차지하는 메모리 용량은 약 18.9 KB입니다.
- 주요 특징
- 결측치: bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g 열에는 각각 2개의 결측치가 있습니다. sex 열에는 11개의 결측치가 있습니다.
- 데이터 타입: 숫자형 데이터(float64)는 4개의 열에 사용되었고, 문자열 데이터(object)는 3개의 열에 사용되었습니다.
- 메모리 사용량: 데이터 프레임은 344개의 행과 7개의 열로 구성되어 있으며, 메모리 사용량은 약 18.9 KB로 효율적입니다.



sort_values()

sort_values() 메서드는 특정 열을 기준으로 데이터 프레임을 정렬합니다. 기본값은 오름차순입니다.

```
# 'bill_length_mm' 열을 기준으로 데이터 프레임 정렬
sorted_df = df.sort_values(by='bill_length_mm', ascending=False)
print(sorted_df.head())
```

```
##          species   island bill_length_mm ... flipper_length_mm body_mass_g   sex
## 253      Gentoo  Biscoe        59.6   ...           230.0    6050.0 Male
## 169 Chinstrap   Dream        58.0   ...           181.0    3700.0 Female
## 321      Gentoo  Biscoe        55.9   ...           228.0    5600.0 Male
## 215 Chinstrap   Dream        55.8   ...           207.0    4000.0 Male
## 335      Gentoo  Biscoe        55.1   ...           230.0    5850.0 Male
##
## [5 rows x 7 columns]
```



sort_values()

두 개의 열, 즉 bill_length_mm와 bill_depth_mm를 기준으로 데이터프레임을 정렬하고 싶은 경우, by 옵션에 리스트 형식으로 입력해줍니다.

```
sorted_df = df.sort_values(  
    by=['bill_length_mm', 'bill_depth_mm'],  
    ascending=[False, True] # 'bill_length_mm'를 내림차순으로, 'bill_depth_mm'를 오름차순으로 정렬  
)  
print(sorted_df.head())
```

```
##          species  island  bill_length_mm  ...  flipper_length_mm  body_mass_g  sex  
## 253      Gentoo  Biscoe           59.6  ...            230.0     6050.0  Male  
## 169    Chinstrap  Dream           58.0  ...            181.0     3700.0 Female  
## 321      Gentoo  Biscoe           55.9  ...            228.0     5600.0  Male  
## 215    Chinstrap  Dream           55.8  ...            207.0     4000.0  Male  
## 335      Gentoo  Biscoe           55.1  ...            230.0     5850.0  Male  
##  
## [5 rows x 7 columns]
```



idxmax()와 idxmin()

idxmax()와 idxmin() 메서드는 각각 데이터프레임이나 시리즈에서 최대값과 최소값을 가지는 첫 번째 인덱스를 반환합니다. bill_length_mm 열에서 가장 큰 값을 가지는 행의 인덱스를 찾아 출력합니다. 또한, loc 메서드를 사용하여 해당 인덱스에 위치한 행의 데이터를 출력합니다.

```
# 'bill_length_mm' 열에서 최대값을 가지는 행의 인덱스  
max_idx = df['bill_length_mm'].idxmax()  
print(f"Maximum bill length index: {max_idx}")
```

```
## Maximum bill length index: 253
```

```
print(df.loc[max_idx])
```

```
## species           Gentoo  
## island            Biscoe  
## bill_length_mm    59.6  
## bill_depth_mm     17.0  
## flipper_length_mm 230.0  
## body_mass_g       6050.0  
## sex                Male  
## Name: 253, dtype: object
```



idxmax()와 idxmin()

`bill_length_mm` 열에서 가장 작은 값을 가지는 행의 인덱스를 찾아 출력합니다. 또한, `loc` 메서드를 사용하여 해당 인덱스에 위치한 행의 데이터를 출력합니다.

```
# 'bill_length_mm' 열에서 최소값을 가지는 행의 인덱스를 찾아 출력합니다.
min_idx = df['bill_length_mm'].idxmin()
print(f"Minimum bill length index: {min_idx}")

## Minimum bill length index: 142
```

```
print(df.loc[min_idx])
```

## species	Adelie
## island	Dream
## bill_length_mm	32.1
## bill_depth_mm	15.5
## flipper_length_mm	188.0
## body_mass_g	3050.0
## sex	Female
## Name:	142, dtype: object



sort_values() 와 idxmax() 의 결과 비교 예제

인덱스는 고유값(unique value)을 갖습니다. 예를 들어 국가 단위 데이터의 경우 대륙, 단순 설문 조사 데이터의 경우 사람이 인덱스 단위가 됩니다.

```
sorted_df = df.sort_values(by='bill_length_mm', ascending=False) # 'bill_length_mm' 열을 기준으로  
sorted_max_value_row = sorted_df.iloc[0]  
print(sorted_max_value_row)
```

```
## species           Gentoo  
## island            Biscoe  
## bill_length_mm    59.6  
## bill_depth_mm     17.0  
## flipper_length_mm 230.0  
## body_mass_g       6050.0  
## sex                Male  
## Name: 253, dtype: object
```



sort_values() 와 idxmax() 의 결과 비교 예제

```
# 'bill_length_mm' 열에서 최대값을 가지는 행의 인덱스를 찾기  
max_idx = df['bill_length_mm'].idxmax()  
max_value_row = df.loc[max_idx]  
print(max_value_row)
```

```
## species           Gentoo  
## island            Biscoe  
## bill_length_mm     59.6  
## bill_depth_mm      17.0  
## flipper_length_mm   230.0  
## body_mass_g        6050.0  
## sex                Male  
## Name: 253, dtype: object
```



sort_values() 와 idxmax() 의 결과 비교 예제

sort_values()로 정렬된 데이터프레임의 첫 번째 행과 idxmax()로 찾은 최대값의 행이 같은 것을 확인할 수 있습니다.

```
# 비교  
comparison = max_value_row.equals(sorted_max_value_row)  
print(f"\nDo the max value rows match? {comparison}")
```

```
##  
## Do the max value rows match? True
```



groupby()

groupby() 메서드는 특정 열을 기준으로 데이터 프레임을 그룹화합니다. 그룹화한 후에는 각 그룹별로 집계 연산을 수행할 수 있습니다.

```
# 'species' 열을 기준으로 그룹화하여 평균 계산  
grouped_df = df.groupby('species').mean(numeric_only = True)  
print(grouped_df)
```

```
##           bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  
## species  
## Adelie      38.791391      18.346358      189.953642  3700.662252  
## Chinstrap    48.833824      18.420588      195.823529  3733.088235  
## Gentoo      47.504878      14.982114      217.186992  5076.016260
```



mean()

mean() 메서드는 데이터 프레임의 각 열의 평균을 계산합니다. 각 열의 정보가 수치형인 경우에만 해당됩니다.

```
# 데이터 프레임의 각 열의 평균 계산  
print(df.mean(numeric_only = True))
```

```
## bill_length_mm      43.921930  
## bill_depth_mm       17.151170  
## flipper_length_mm   200.915205  
## body_mass_g        4201.754386  
## dtype: float64
```



sum()

sum() 메서드는 데이터 프레임의 각 열의 합계를 계산합니다.

```
# 데이터 프레임의 각 열의 합계 계산  
print(df.sum(numeric_only = True))
```

```
## bill_length_mm      15021.3  
## bill_depth_mm       5865.7  
## flipper_length_mm    68713.0  
## body_mass_g        1437000.0  
## dtype: float64
```



groupby() 활용 예제

각 섬(island)별로 flipper_length_mm의 합계를 구하고, 그 중 합계가 가장 큰 섬을 찾아보겠습니다. groupby()와 idxmax()를 활용해보겠습니다.

```
# island별로 그룹화하고 flipper_length_mm의 합계를 계산  
grouped_df = df.groupby('island')['flipper_length_mm'].sum()  
print("Grouped by island and summed flipper_length_mm:")
```

```
## Grouped by island and summed flipper_length_mm:
```

```
print(grouped_df)
```

```
## island  
## Biscoe      35021.0  
## Dream       23941.0  
## Torgersen   9751.0  
## Name: flipper_length_mm, dtype: float64
```



groupby() 활용 예제

```
# flipper_length_mm 합계가 가장 큰 island 찾기  
max_flipper_island = grouped_df.idxmax()  
print(f"\nIsland with maximum total flipper length: {max_flipper_island}")
```

```
##  
## Island with maximum total flipper length: Biscoe
```

```
print(grouped_df.loc[max_flipper_island])
```

```
## 35021.0
```



groupby() 활용 예제

groupby()와 sort_values()를 활용해보겠습니다.

groupby() 옵션 중 as_index = False 옵션을 활용하면 코드를 단순화할 수 있습니다. -
as_index=True : 그룹화된 열을 결과 데이터프레임의 인덱스로 설정

- as_index=False : 그룹화된 열을 결과 데이터프레임의 일반 열로 유지

```
# island별로 그룹화하고 flipper_length_mm의 합계를 계산한 데이터프레임 생성  
grouped_sum_df = df.groupby('island', as_index=False)[['flipper_length_mm']].sum()  
print(grouped_sum_df)
```

```
##          island  flipper_length_mm  
## 0        Biscoe      35021.0  
## 1        Dream      23941.0  
## 2    Torgersen      9751.0
```



groupby() 활용 예제

island 열이 인덱스가 아닌 하나의 열로 처리되는 것을 확인할 수 있습니다. `sort_values()`를 활용하여 데이터를 내림차순으로 정렬합니다.

```
# flipper_length_mm 합계를 기준으로 내림차순 정렬
sorted_grouped_sum_df = grouped_sum_df.sort_values(by='flipper_length_mm', ascending=False)
print("\nGrouped and sorted DataFrame by total flipper_length_mm:")
```

```
##
## Grouped and sorted DataFrame by total flipper_length_mm:
```

```
print(sorted_grouped_sum_df)
```

```
##      island  flipper_length_mm
## 0     Biscoe      35021.0
## 1     Dream       23941.0
## 2   Torgersen      9751.0
```

merge()



merge() 메서드는 두 데이터 프레임을 병합합니다. 기본적으로 공통된 열을 기준으로 병합합니다.

- `on='key'`: 병합할 때 기준이 되는 열을 지정합니다. 여기서는 key 열을 기준으로 병합합니다.
- `how='inner'`: 병합 방법을 지정합니다. inner는 내부 조인을 의미하며, 두 데이터 프레임 모두에 존재하는 키 값에 대해 병합합니다.

```
# 예제 데이터 프레임 생성
df1 = pd.DataFrame({'key': ['A', 'B', 'C'], 'value': [1, 2, 3]})
df2 = pd.DataFrame({'key': ['A', 'B', 'D'], 'value': [4, 5, 6]})

# 두 데이터 프레임 병합
merged_df = pd.merge(df1, df2, on='key', how='inner')
print(merged_df)
```

```
##   key  value_x  value_y
## 0   A        1        4
## 1   B        2        5
```



merge() 병합 방법(how) 옵션

how 옵션에는 여러 가지가 있으며, 각 옵션은 병합할 때 사용되는 방식에 따라 다릅니다.

- `inner` (기본값): 두 데이터 프레임 모두에 존재하는 키 값에 대해서만 병합합니다. 즉, 교집합을 구하는 방식입니다.
- `outer`: 두 데이터 프레임 중 하나에라도 존재하는 모든 키 값에 대해 병합합니다. 즉, 합집합을 구하는 방식입니다.
- `left`: 첫 번째 데이터 프레임(df1)의 모든 키 값을 기준으로 병합합니다. df1의 키 값이 유지되고, df2의 키 값은 df1과 일치하는 경우에만 포함됩니다.
- `right`: 두 번째 데이터 프레임(df2)의 모든 키 값을 기준으로 병합합니다. df2의 키 값이 유지되고, df1의 키 값은 df2와 일치하는 경우에만 포함됩니다.



merge() 병합 방법(how) 옵션 예제

예를 들어, `how='outer'`를 사용하면 다음과 같이 됩니다.

```
# 두 데이터 프레임을 outer join으로 병합
merged_df_outer = pd.merge(df1, df2, on='key', how='outer')
print(merged_df_outer)
```

```
##   key  value_x  value_y
## 0   A      1.0      4.0
## 1   B      2.0      5.0
## 2   C      3.0      NaN
## 3   D      NaN      6.0
```



데이터 재구조화

데이터 재구조화는 데이터 전처리 과정에서 매우 중요한 작업입니다. 데이터 재구조화를 적절히 수행할 경우 긴 코드를 효율적으로 줄일 수 있습니다. 데이터 재구조화에 활용되는 `melt()`, `pivot_table()`, `pivot()`에 대해 알아보겠습니다.

`melt()`

`melt()` 메서드는 데이터를 넓은 형식(wide form)에서 긴 형식(long form)으로 변환합니다.

주요 옵션

- `frame`: 재구조화할 데이터프레임
- `id_vars`: 변환되지 않고 그대로 유지될 칼럼 지정
- `value_vars`: 변환할 칼럼 지정(지정하지 않으면 `id_vars`를 제외한 모든 칼럼이 선택됨)
- `var_name`: `value_vars`의 칼럼명이 저장될 칼럼의 칼럼명 지정(디폴트 : `variable`)
- `value_name`: `value_vars`의 값이 저장될 칼럼명 지정(디폴트 : `value`)

melt()



실습을 위해 예제 데이터프레임을 만들어보겠습니다.

```
import pandas as pd

data = {
    'Date': ['2024-07-01', '2024-07-02', '2024-07-03', '2024-07-03'],
    'Temperature': [10, 20, 25, 20],
    'Humidity': [60, 65, 70, 21]
}

df = pd.DataFrame(data)
print(df)
```

```
##           Date Temperature Humidity
## 0  2024-07-01          10       60
## 1  2024-07-02          20       65
## 2  2024-07-03          25       70
## 3  2024-07-03          20       21
```



melt()

melt() 메서드를 사용하여 데이터프레임을 긴 형식(long form)으로 변환해보겠습니다.

```
df_melted = pd.melt(df,
                     id_vars=['Date'],
                     value_vars=['Temperature', 'Humidity'],
                     var_name='Variable',
                     value_name='Value')

print(df_melted.head(6))
```

```
##           Date   Variable  Value
## 0  2024-07-01 Temperature    10
## 1  2024-07-02 Temperature    20
## 2  2024-07-03 Temperature    25
## 3  2024-07-03 Temperature    20
## 4  2024-07-01     Humidity    60
## 5  2024-07-02     Humidity    65
```



pivot()

pivot() 메서드는 데이터를 긴 형식(long form)에서 넓은 형식(wide form)으로 변환합니다.

주요 옵션

- index: 새 데이터프레임에서 행 인덱스로 사용할 칼럼명 지정
- columns: 새 데이터프레임에서 열로 사용할 칼럼명 지정
- values: 새 데이터프레임에서 각 인덱스-열 조합에 대해 채워질 값으로 사용할 칼럼명 지정

melt() 메서드로 변환된 데이터를 pivot()을 활용하여 재변환해보겠습니다.

```
df_pivoted = df_melted.pivot(index='Date',  
                               columns='Variable',  
                               values='Value').reset_index()
```

```
## ValueError: Index contains duplicate entries, cannot reshape
```



pivot()

pivot() 메서드는 각 행을 고유하게 식별할 수 있는 인덱스가 필요합니다. index 옵션을 설정하지 않을 경우 오류가 발생할 수 있습니다. Date 칼럼의 경우 '2024-07-03'이 중복되므로, 인덱스의 역할을 할 수 없습니다. melt() 메서드를 적용하기 전에 각 행을 고유하게 식별할 수 있는 인덱스를 임의로 생성해줍니다.

```
df_melted2 = pd.melt(df.reset_index(),
                      id_vars=['index'],
                      value_vars=['Temperature', 'Humidity'],
                      var_name='Variable',
                      value_name='Value')

print(df_melted2.head(4))
```

```
##      index     Variable   Value
## 0          0 Temperature    10
## 1          1 Temperature    20
## 2          2 Temperature    25
## 3          3 Temperature    20
```



pivot()

index 칼럼을 기준으로 pivot() 메서드를 적용해보겠습니다.

```
df_pivoted = df_melted2.pivot(index = 'index',
                                columns='Variable',
                                values='Value')

print(df_pivoted)
```

```
## Variable  Humidity  Temperature
## index
## 0          60         10
## 1          65         20
## 2          70         25
## 3          21         20
```



pivot_table()

`pivot_table()` 메서드는 `pivot()`와 마찬가지로 데이터를 긴 형식(long form)에서 넓은 형식(wide form)으로 변환합니다.

주요 옵션

- `data`: 피벗할 데이터프레임
- `values`: 집계할 데이터 값의 칼럼명 지정
- `index`: 행 인덱스로 사용할 칼럼명 지정
- `columns`: 열로 사용할 칼럼명 지정
- `aggfunc`: 집계 함수(예: 'mean', 'sum', 'count' 등) 지정(디폴트 : 'mean')



pivot_table()

melt() 메서드로 변환된 데이터를 pivot_table()을 활용하여 재변환해보겠습니다. 이전에 df_melted 데이터는 pivot()을 활용할 경우 오류가 발생했습니다. pivot_table() 메서드의 경우 aggfunc 옵션을 활용하여 중복이 있는 날짜의 값을 평균으로 대치합니다.

```
df_pivot_table = df_melted.pivot_table(index='Date',
                                         columns='Variable',
                                         values='Value').reset_index()
print(df_pivot_table)
```

```
## Variable      Date  Humidity Temperature
## 0            2024-07-01     60.0       10.0
## 1            2024-07-02     65.0       20.0
## 2            2024-07-03     45.5       22.5
```



pivot_table()과 pivot()의 차이점

- 중복 데이터 처리
 - `pivot()` : 중복된 인덱스-열 조합이 있을 경우 오류 출력
 - `pivot_table` : 중복된 인덱스-열 조합이 있을 경우, 집계 함수(aggfunc)를 사용하여 데이터 집계
- 집계 함수
 - `pivot()` : 집계 기능 없음
 - `pivot_table` : 집계 함수를 통해 데이터 집계 가능



학생 성적 데이터

학생 성적 데이터셋은 학생들의 성적에 영향을 미치는 다양한 요인들을 포함하고 있습니다. 이 데이터셋은 교육 연구에서 학생들의 성적 향상에 기여하는 요인을 분석하는 데 사용됩니다. 각 행은 학생 한 명을 나타내며, 다음과 같은 칼럼으로 구성되어 있습니다:

1. school: 학생이 소속된 학교 ('GP' - Gabriel Pereira 또는 'MS' - Mousinho da Silveira)
2. sex: 학생의 성별 ('F' - 여성 또는 'M' - 남성)
3. paid: 추가 과외 수업 여부 ('yes', 'no')
4. famrel: 가족 관계의 질 (1 - 매우 나쁨, 2 - 나쁨, 3 - 보통, 4 - 좋음, 5 - 매우 좋음)
5. freetime: 자유 시간의 질 (1 - 매우 적음, 2 - 적음, 3 - 보통, 4 - 많음, 5 - 매우 많음)
6. goout: 친구와 외출 빈도 (1 - 매우 적음, 2 - 적음, 3 - 보통, 4 - 많음, 5 - 매우 많음)
7. Dalc: 주중 음주량 (1 - 매우 적음, 2 - 적음, 3 - 보통, 4 - 많음, 5 - 매우 많음)
8. Walc: 주말 음주량 (1 - 매우 적음, 2 - 적음, 3 - 보통, 4 - 많음, 5 - 매우 많음)
9. health: 현재 건강 상태 (1 - 매우 나쁨, 2 - 나쁨, 3 - 보통, 4 - 좋음, 5 - 매우 좋음)
10. absences: 결석 일수
11. grade: 최종 성적



학생 성적 데이터 불러오기

학생 성적 데이터를 불러오겠습니다.

```
df = pd.read_csv( './data/dat.csv' )
print(df.head())
```

	## school	## sex	## paid	## famrel	## freetime	## goout	## Dalc	## Walc	## health	## absences	## grade
## 0	GP	F	no	4	3	4.0	1	1	3	6	1
## 1	GP	F	no	5	3	3.0	1	1	3	4	1
## 2	GP	F	yes	4	3	2.0	2	3	3	10	4
## 3	GP	F	yes	3	2	2.0	1	1	5	2	9
## 4	GP	F	yes	4	3	2.0	1	2	5	4	4



학생 성적 데이터 불러오기

데이터를 처음 불러올 때는 데이터의 특성을 이해하기 위해 데이터 구조, 결측치, 이상치 존재 여부 등을 확인해야합니다. 이전 챕터에서 배운 `.info()`를 통해 확인해보겠습니다.

```
print(df.loc[:, ['school', 'sex', 'paid', 'goout']].info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 366 entries, 0 to 365
## Data columns (total 4 columns):
## #   Column  Non-Null Count  Dtype  
## ---  -----  --------------  ----- 
## 0   school   366 non-null   object 
## 1   sex       366 non-null   object 
## 2   paid      366 non-null   object 
## 3   goout     356 non-null   float64
## dtypes: float64(1), object(3)
## memory usage: 11.6+ KB
## None
```



rename()

rename() 메서드는 특정 칼럼의 이름을 변경합니다. 데이터셋을 불러왔을 때 칼럼명이 적절한 형태가 아닐 경우 칼럼명을 적절하게 변경해줘야 합니다. Dalc, Walc 칼럼의 경우 다른 칼럼과 달리 앞 글자가 대문자인 것을 확인할 수 있습니다. 칼럼명을 깔끔하게 정리하기 위해서 소문자로 변경해보겠습니다.

```
df = df.rename(columns = {'Dalc' : 'dalc', 'Walc' : 'walc'})  
print(df.columns)
```

```
## Index(['school', 'sex', 'paid', 'famrel', 'freetime', 'goout', 'dalc', 'walc',  
##         'health', 'absences', 'grade'],  
##        dtype='object')
```



astype()

astype()은 데이터프레임 또는 시리즈의 데이터 타입을 변경하는 데 사용됩니다. 넘파이에서는 numpy 배열의 데이터 타입을 변경하는 데 사용됩니다. 데이터셋을 불러왔을 때 각 칼럼별로 데이터 타입이 적절하게 설정되어 있지 않을 수도 있습니다. 이 경우 데이터 전처리 시작 전 astype()을 활용하여 데이터 타입을 적절하게 변경해줘야 합니다.

```
print(df.loc[:, ['famrel', 'dalc']].astype({'famrel' : 'object', 'dalc' : 'float64'}).info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 366 entries, 0 to 365
## Data columns (total 2 columns):
##   #   Column  Non-Null Count  Dtype  
##   --  --      --      --      --      
##   0   famrel  366 non-null    object 
##   1   dalc    366 non-null    float64
## dtypes: float64(1), object(1)
## memory usage: 5.8+ KB
## None
```



assign()

assign()은 새로운 칼럼을 생성(= 파생 변수를 생성)하거나 특정 칼럼의 값을 변경하는 데 사용됩니다. 예를 들어 famrel 칼럼의 특정 값을 기준으로 'Low', 'Medium', 'High'로 구분해볼까요?

먼저 다음과 같은 함수를 정의해보겠습니다.

```
def classify_famrel(famrel):
    if famrel <= 2:
        return 'Low'
    elif famrel <= 4:
        return 'Medium'
    else:
        return 'High'
```



assign()

apply()를 활용하여 정의한 함수를 활용하여 famrel_quality 칼럼을 새롭게 생성해보겠습니다.

```
df1 = df.copy()
df1 = df1.assign(famrel_quality=df1['famrel'].apply(classify_famrel))

print(df1[['famrel', 'famrel_quality']].head())
```

```
##      famrel famrel_quality
## 0        4       Medium
## 1        5        High
## 2        4       Medium
## 3        3       Medium
## 4        4       Medium
```



assign()

기존에 있던 famrel 칼럼의 값을 변경할 수도 있습니다.

```
df2 = df.copy()
df2 = df2.assign(famrel=df2['famrel'].apply(classify_famrel))

print(df2[['famrel']].head())
```

```
##      famrel
## 0    Medium
## 1     High
## 2    Medium
## 3    Medium
## 4    Medium
```



assign()

물론 assign()을 꼭 사용해야 하는 것은 아닙니다.

```
df3 = df.copy()
df3['famrel'] = df3['famrel'].apply(classify_famrel)

print(df3[['famrel']].head())
```

```
##      famrel
## 0    Medium
## 1     High
## 2    Medium
## 3    Medium
## 4    Medium
```



select_dtypes()

`select_dtypes()`은 데이터프레임에서 특정 데이터 타입을 가진 칼럼만 선택하는 데 사용되는 메서드입니다. 데이터프레임의 칼럼 중 원하는 데이터 타입의 칼럼만 선택할 수 있어, 데이터 전처리 시 매우 유용합니다. 데이터 타입을 선택하기 위한 옵션은 다음과 같습니다.

- `number`: 모든 수치형 데이터 타입을 포함합니다. (정수, 부동 소수점, 복소수)
- `float`: 부동 소수점 숫자 (float64, float32 등)
- `int`: 정수 (int64, int32 등)
- `complex`: 복소수 (complex64, complex128 등)
- `object`: 일반적인 객체 타입 (보통 문자열이 여기에 속함)
- `bool`: 불리언 값 (True, False)
- `category`: 범주형 데이터 타입
- `datetime`: 날짜 및 시간 (datetime64[ns] 등)



select_dtypes()

예를 들어 수치형 칼럼만 선택해볼까요?

```
print(df.select_dtypes('number').head(2))
```

```
##      famrel  freetime   goout  dalc  walc  health  absences  grade
## 0          4          3     4.0     1      1        3         6       1
## 1          5          3     3.0     1      1        3         4       1
```

범주형 칼럼만 선택해볼 수도 있습니다.

```
print(df.select_dtypes('object').head(2))
```

```
##    school sex paid
## 0      GP   F   no
## 1      GP   F   no
```



select_dtypes()

이전 예제와 같이 임의의 함수를 정의하고 적용해볼 수도 있습니다. 다음과 같은 함수를 정의했습니다.

```
def standardize(x):  
    return (x - np.nanmean(x)/np.std(x))
```

수치형 칼럼에 대해서 정의된 함수를 적용해보겠습니다.

```
df.select_dtypes('number').apply(standardize).head(2)
```

```
##      famrel  freetime     goout ...    health  absences     grade  
## 0 -0.41557 -0.242302  1.192457 ...  0.408977  5.310415 -0.639516  
## 1  0.58443 -0.242302  0.192457 ...  0.408977  3.310415 -0.639516  
##  
## [2 rows x 8 columns]
```



칼럼명 패턴을 활용하여 특정 칼럼 선택하기

데이터 탑에 따라 특정 칼럼을 선택하는 방법에 대해 알아봤습니다. 데이터 탑에 따라 칼럼을 선택할 수도 있지만, 칼럼명의 패턴에 따라 특정 칼럼을 선택할 수도 있습니다. 예시를 통해 알아보겠습니다.

```
print(df.columns)
```

```
## Index(['school', 'sex', 'paid', 'famrel', 'freetime', 'goout', 'dalc', 'walc',
##         'health', 'absences', 'grade'],
##        dtype='object')
```



'f'로 시작하는 칼럼 선택

`str.startswith()`를 이용해서 칼럼명이 'f'로 시작하는 경우 True, 아닐 경우 False로 지정합니다.

```
print(df.columns)
```

```
## Index(['school', 'sex', 'paid', 'famrel', 'freetime', 'goout', 'dalc', 'walc',
##         'health', 'absences', 'grade'],
##        dtype='object')
```

```
print(df.columns.str.startswith('f'))
```

```
## [False False False  True  True False False False False False]
```



'f'로 시작하는 칼럼 선택

loc[]를 활용하여, True인 칼럼을 필터링합니다.

```
print(df.loc[:, df.columns.str.startswith('f')].head())
```

```
##      famrel  freetime
## 0        4        3
## 1        5        3
## 2        4        3
## 3        3        2
## 4        4        3
```

'c'로 끝나는 칼럼 선택



`str.endswith()` 함수를 이용해서 칼럼명이 'c'로 끝나는 경우 True, 아닐 경우 False로 지정합니다.

```
print(df.columns)
```

```
## Index(['school', 'sex', 'paid', 'famrel', 'freetime', 'goout', 'dalc', 'walc',
##         'health', 'absences', 'grade'],
##        dtype='object')
```

```
print(df.columns.str.endswith('c'))
```

```
## [False False False False False False  True  True False False]
```



'c'로 끝나는 칼럼 선택

loc[]를 활용하여, True인 칼럼을 필터링합니다.

```
print(df.loc[:, df.columns.str.endswith('c')].head())
```

```
##      dalc  walc
## 0      1      1
## 1      1      1
## 2      2      3
## 3      1      1
## 4      1      2
```



'f'를 포함하는 칼럼 선택

`str.contains()`를 이용해서 칼럼명이 'f'를 포함하는 경우 True, 아닐 경우 False로 지정합니다.

```
print(df.columns)
```

```
## Index(['school', 'sex', 'paid', 'famrel', 'freetime', 'goout', 'dalc', 'walc',
##         'health', 'absences', 'grade'],
##        dtype='object')
```

```
print(df.columns.str.contains('f'))
```

```
## [False False False  True  True False False False False False]
```



'f'를 포함하는 칼럼 선택

loc[]를 활용하여, True인 칼럼을 필터링합니다.

```
print(df.loc[:, df.columns.str.contains('f')].head())
```

```
##      famrel  freetime
## 0        4        3
## 1        5        3
## 2        4        3
## 3        3        2
## 4        4        3
```



'f'를 포함하는 칼럼 선택

이전에 정의한 임의의 함수를 특정 'f'를 포함하는 칼럼에 적용해볼 수도 있습니다.

```
print(df.loc[:, df.columns.str.contains('f')].apply(standardize).head())
```

```
##      famrel   freetime
## 0 -0.41557 -0.242302
## 1  0.58443 -0.242302
## 2 -0.41557 -0.242302
## 3 -1.41557 -1.242302
## 4 -0.41557 -0.242302
```



판다스 데이터 프레임에서 저장하기

csv로 저장하기

CSV(Comma-Separated Values)는 가장 일반적으로 사용되는 파일 형식 중 하나입니다.

```
import pandas as pd

# 예제 데이터 생성
df = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Score': [90, 85, 88]
})
```

```
# CSV 파일로 저장 (인덱스 제외)
df.to_csv("data.csv", index=False)
```



csv로 저장하기

- `index=False`: 인덱스 저장 안 함 (기본값은 `True`)
- `sep=";"`: 구분자 변경 가능 (, 대신 ; 사용 가능)
- `encoding="utf-8"`: 한글 데이터 저장 시 사용

```
# CSV 파일로 저장 (인덱스 제외)
df.to_csv("data.csv", index=False, encoding="utf-8")
```

Excel로 저장하기



Excel 파일로 데이터를 저장할 수도 있습니다. Excel 파일로 저장하기 위해서는 openpyxl 라이브러리가 설치되어야 합니다.

```
# CSV 파일로 저장 (인덱스 제외)
df1.to_excel("data/ex_data.xlsx", sheet_name='Sheet_name_1')
```



Parquet 파일로 저장하기

Parquet은 대용량 데이터를 저장할 때 매우 효율적인 포맷으로, 압축 및 빠른 I/O를 제공합니다. parquet 파일로 저장하기 위해서는 pyarrow 라이브러리가 설치되어야 합니다.

```
# Parquet 파일로 저장  
df.to_parquet("data/data.parquet", engine="pyarrow")
```

```
# Parquet 파일 불러오기  
df = pd.read_parquet("data/data.parquet", engine="pyarrow")
```



Pickle 파일로 저장하기

Pickle은 CSV보다 빠르고, 데이터 타입 변형 없이 저장할 수 있습니다.

```
# Pickle 파일로 저장  
df.to_pickle("data/data.pkl")
```

```
# Pickle 파일 불러오기  
df_loaded = pd.read_pickle("data/data.pkl")
```



JSON 파일로 저장하기

JSON(JavaScript Object Notation)은 웹 API 및 데이터 교환에 많이 사용되는 포맷입니다. CSV 보다 가독성이 높고, 계층적 데이터 구조를 지원합니다.

JSON 저장 시 주요 옵션

- `orient="records"` → 각 행을 JSON 객체로 저장
- `indent=4` → JSON 데이터 가독성을 높이기 위한 들여쓰기

```
# JSON 파일로 저장  
df.to_json("data/data.json", orient="records", indent=4)
```

```
# JSON 파일 불러오기  
df_loaded = pd.read_json("data/data.json")
```

비교하기



Table: 파일 포맷 비교

포맷	저장속도	읽기속도	데이터.타입.유지	압축.가능	추천.사용.사례
CSV	보통	느림	✗	✗	범용 데이터 공유
JSON	느림	보통	✗	✗	웹 API 데이터 저장
Parquet	빠름	빠름	✓	✓	대용량 데이터 저장
Pickle	매우 빠름	매우 빠름	✓	✗	Python 전용 데이터 저장



연습 문제

학교 성적 데이터

```
df = pd.read_csv('./data/grade.csv')
print(df.head())
```

```
##      student_id      name gender midterm final assignment
## 0            1      Alice      F       85     88         95
## 1            2        Bob      M       78     74         82
## 2            3    Charlie      M       92     94         87
## 3            4     David      M       88     90         85
## 4            5       Eve      F       76     79         77
```

df 데이터 프레임의 정보를 출력하고, 각 열의 데이터 타입을 확인하세요.



```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 10 entries, 0 to 9
## Data columns (total 6 columns):
##   #   Column      Non-Null Count  Dtype  
##   --  --          -----          ----- 
##   0   student_id  10 non-null    int64  
##   1   name        10 non-null    object 
##   2   gender       10 non-null    object 
##   3   midterm     10 non-null    int64  
##   4   final       10 non-null    int64  
##   5   assignment   10 non-null    int64  
## dtypes: int64(4), object(2)
## memory usage: 612.0+ bytes
## None
```

midterm 점수가 85점 이상인 학생들의 데이터를 필터링하여 출력하세요.



```
##   student_id    name gender midterm final assignment
## 0          1    Alice      F      85     88        95
## 2          3  Charlie      M      92     94        87
## 3          4    David      M      88     90        85
## 5          6    Frank      M      95     97        98
## 6          7   Grace      F      89     91        84
## 7          8  Hannah      F      90     92        90
```

final 점수를 기준으로 데이터 프레임을 내림차순으로 정렬하고, 정렬된 데이터 프레임의 첫 5행을 출력하세요.



```
##   student_id     name gender midterm final assignment
## 5          6    Frank      M      95    97        98
## 2          3  Charlie      M      92    94        87
## 7          8  Hannah      F      90    92        90
## 6          7   Grace      F      89    91        84
## 3          4   David      M      88    90        85
```

gender 열을 기준으로 데이터 프레임을 그룹화하고, 각 그룹별 midterm과 final의 평균을 계산하여 출력하세요.



```
##           midterm   final
## gender
## F          85.00000  87.5
## M          85.66667  86.5
```

student_id 열을 문자열 타입으로 변환하고, 변환된 데이터 프레임의 정보를 출력하세요.



```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 10 entries, 0 to 9
## Data columns (total 6 columns):
##   #   Column      Non-Null Count  Dtype  
##   --  --          -----          ----- 
##   0   student_id  10 non-null    object 
##   1   name        10 non-null    object 
##   2   gender       10 non-null    object 
##   3   midterm     10 non-null    int64  
##   4   final       10 non-null    int64  
##   5   assignment   10 non-null    int64  
## dtypes: int64(3), object(3)
## memory usage: 612.0+ bytes
## None
```

assignment 점수의 최대값과 최소값을 가지는 행을 각각 출력하세요.



```
## Max value row:
```

```
## student_id      6
## name        Frank
## gender       M
## midterm     95
## final       97
## assignment   98
## Name: 5, dtype: object
```

```
##
```

```
## Min value row:
```

```
## student_id      5
## name        Eve
## gender       F
## midterm     76
## final       79
## assignment   77
## Name: 4, dtype: object
```

midterm, final, assignment 점수의 평균을 계산하여 average 열을 추가하고, 첫 5행을 출력하세요.



```
##   student_id      name gender midterm final assignment     average
## 0          1      Alice      F      85     88        95 89.3333333
## 1          2       Bob       M      78     74        82 78.0000000
## 2          3    Charlie      M      92     94        87 91.0000000
## 3          4     David       M      88     90        85 87.6666667
## 4          5      Eve       F      76     79        77 77.3333333
```

아래의 추가 데이터를 생성하고, 기존 데이터 프레임과 student_id를 기준으로 병합하여 출력하세요.



```
# 추가 데이터 생성  
additional_data = {  
    'student_id': ['1', '3', '5', '7', '9'],  
    'club': ['Art', 'Science', 'Math', 'Music', 'Drama']  
}  
df_additional = pd.DataFrame(additional_data)
```

```
##   student_id      name gender  midterm  final  assignment      average       club  
## 0          1     Alice      F        85      88          95  89.333333      Art  
## 1          2       Bob      M        78      74          82  78.000000      NaN  
## 2          3  Charlie      M        92      94          87  91.000000  Science  
## 3          4    David      M        88      90          85  87.666667      NaN  
## 4          5      Eve      F        76      79          77  77.333333    Math
```

gender를 인덱스로, student_id를 열로 사용하여 average 점수에 대한 피벗 테이블을 생성하고 출력하세요.



```
## student_id      1   10    2    3   ...      6    7      8      9  
## gender          ...  
## F             89.333333  NaN  NaN  NaN  ...  NaN  88.0  90.666667  NaN  
## M             NaN  86.0  78.0  91.0  ...  96.666667  NaN  NaN  78.666667  
##  
## [2 rows x 10 columns]
```

midterm, final, assignment의 평균을 구하고, average 열을 생성하시오. 또한, 성별, 성적 유형(assignment, average, final, midterm)별 평균 점수를 계산하시오.



```
df = pd.read_csv('./data/grade.csv')
```

```
##   gender  variable      score
## 0      F  assignment  86.500000
## 1      F      average  86.333333
## 2      F      final  87.500000
## 3      F     midterm  85.000000
## 4      M  assignment  86.833333
## 5      M      average  86.333333
## 6      M      final  86.500000
## 7      M     midterm  85.666667
```

midterm, final, assignment의 평균을 구하고, average 열을 생성하시오. 또한, 최대 평균 성적을 가진 학생의 이름과 평균 성적을 출력하시오.



```
## Student with highest average score: Frank
```

```
## name      Frank
## average   96.666667
## Name: 5, dtype: object
```



연습 문제 해답

```
df = pd.read_csv('./data/grade.csv')
```



df 데이터 프레임의 정보를 출력하고, 각 열의 데이터 타입을 확인하세요.

```
# 데이터 프레임 정보 출력  
print(df.info())
```

```
## <class 'pandas.core.frame.DataFrame'>  
## RangeIndex: 10 entries, 0 to 9  
## Data columns (total 6 columns):  
## #   Column      Non-Null Count  Dtype     
## ---  -----      -----          -----  
## 0   student_id  10 non-null    int64    
## 1   name        10 non-null    object    
## 2   gender       10 non-null    object    
## 3   midterm     10 non-null    int64    
## 4   final        10 non-null    int64    
## 5   assignment   10 non-null    int64    
## dtypes: int64(4), object(2)  
## memory usage: 612.0+ bytes  
## None
```

midterm 점수가 85점 이상인 학생들의 데이터를 필터링하여 출력하세요.



```
# midterm 점수가 85점 이상인 학생들의 데이터 필터링  
filtered_df = df.loc[df[ 'midterm' ] >= 85]  
print(filtered_df)
```

```
##      student_id     name gender midterm final assignment  
## 0           1    Alice      F      85     88        95  
## 2           3   Charlie      M      92     94        87  
## 3           4    David      M      88     90        85  
## 5           6    Frank      M      95     97        98  
## 6           7   Grace      F      89     91        84  
## 7           8  Hannah      F      90     92        90
```

final 점수를 기준으로 데이터 프레임을 내림차순으로 정렬하고, 정렬된 데이터 프레임의 첫 5행을 출력하세요.



```
# final 점수를 기준으로 데이터 프레임 내림차순 정렬  
sorted_df = df.sort_values(by='final', ascending=False)  
print(sorted_df.head())
```

```
##      student_id     name gender midterm  final assignment  
## 5          6    Frank      M      95     97        98  
## 2          3   Charlie      M      92     94        87  
## 7          8   Hannah      F      90     92        90  
## 6          7   Grace       F      89     91        84  
## 3          4   David       M      88     90        85
```

gender 열을 기준으로 데이터 프레임을 그룹화하고, 각 그룹별 midterm과 final의 평균을 계산하여 출력하세요.



```
# gender 열을 기준으로 그룹화하고, 각 그룹별 midterm과 final의 평균 계산  
grouped_df = df.groupby('gender')[['midterm', 'final']].mean()  
print(grouped_df)
```

```
##           midterm  final  
## gender  
## F          85.00000  87.5  
## M          85.66667  86.5
```

student_id 열을 문자열 타입으로 변환하고, 변환된 데이터 프레임의 정보를 출력하세요.



```
df['student_id'] = df['student_id'].astype('str')
print(df.info())
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 10 entries, 0 to 9
## Data columns (total 6 columns):
## #   #   Column      Non-Null Count  Dtype  
## ---  --  --          --          --    
## #   0   student_id  10 non-null    object 
## #   1   name        10 non-null    object 
## #   2   gender       10 non-null    object 
## #   3   midterm     10 non-null    int64  
## #   4   final        10 non-null    int64  
## #   5   assignment   10 non-null    int64  
## #   dtypes: int64(3), object(3)
## #   memory usage: 612.0+ bytes
## #   None
```

assignment 점수의 최대값과 최소값을 가지는 행을 각각 출력하세요.



```
# assignment 점수의 최대값과 최소값을 가지는 행 찾기  
max_idx = df['assignment'].idxmax()  
min_idx = df['assignment'].idxmin()  
  
max_value_row = df.loc[max_idx]  
min_value_row = df.loc[min_idx]
```

```
print(max_value_row)
```

```
## student_id      6  
## name          Frank  
## gender        M  
## midterm      95  
## final         97  
## assignment    98  
## Name: 5, dtype: object
```

```
print(min_value_row)
```

```
## student_id      5  
## name          Eve  
## gender        F  
## midterm      76  
## final         79  
## assignment    77  
## Name: 4, dtype: object
```

midterm, final, assignment 점수의 평균을 계산하여 average 열을 추가하고, 첫 5행을 출력하세요.



```
# midterm, final, assignment 점수의 평균을 계산하여 average 열 추가  
df['average'] = df[['midterm', 'final', 'assignment']].mean(axis=1)  
print(df.head())
```

	student_id	name	gender	midterm	final	assignment	average
## 0	1	Alice	F	85	88	95	89.333333
## 1	2	Bob	M	78	74	82	78.000000
## 2	3	Charlie	M	92	94	87	91.000000
## 3	4	David	M	88	90	85	87.666667
## 4	5	Eve	F	76	79	77	77.333333

아래의 추가 데이터를 생성하고, 기존 데이터 프레임과 student_id를 기준으로 병합하여 출력하세요.



```
# 추가 데이터 생성
additional_data = {
    'student_id': ['1', '3', '5', '7', '9'],
    'club': ['Art', 'Science', 'Math', 'Music', 'Drama']
}
df_additional = pd.DataFrame(additional_data)
```

```
# 기존 데이터 프레임과 병합
merged_df = pd.merge(df, df_additional, on='student_id', how='left')
print(merged_df.head(3))
```

	student_id	name	gender	midterm	final	assignment	average	club
## 0	1	Alice	F	85	88	95	89.333333	Art
## 1	2	Bob	M	78	74	82	78.000000	NaN
## 2	3	Charlie	M	92	94	87	91.000000	Science



gender를 인덱스로, student_id를 열로 사용하여 average 점수에 대한 피벗 테이블을 생성하고 출력하세요.

```
# gender를 인덱스로, student_id를 열로 사용하여 average 점수에 대한 피벗 테이블 생성
pivot_table = df.pivot_table(values='average', index='gender', columns='student_id')
print(pivot_table)
```

```
## student_id      1    10     2     3   ...      6     7     8     9
## gender
## F            89.333333  NaN   NaN   NaN   ...      NaN  88.0  90.666667  NaN
## M            NaN   86.0  78.0  91.0   ...  96.666667  NaN   NaN  78.666667
##
## [2 rows x 10 columns]
```

midterm, final, assignment의 평균을 구하고, average 열을 생성하시오. 또한, 성별, 성적 유형(assignment, average, final, midterm)별 평균 점수를 계산하시오.



```
df = pd.read_csv('./data/grade.csv')
```

```
# average 열 추가  
df['average'] = df[['midterm', 'final', 'as  
  
# melt 함수를 사용하여 데이터 프레임 변환  
melted_df = pd.melt(df, id_vars=['student_i  
  
# gender를 기준으로 그룹화하고 평균 점수 계산  
grouped_mean = melted_df.groupby(['gender',
```

```
print(grouped_mean)
```

```
##   gender      variable    score  
## 0     F   assignment  86.500000  
## 1     F      average  86.333333  
## 2     F       final  87.500000  
## 3     F   midterm  85.000000  
## 4     M   assignment  86.833333  
## 5     M      average  86.333333  
## 6     M       final  86.500000  
## 7     M   midterm  85.666667
```

midterm, final, assignment의 평균을 구하고, average 열을 생성하시오. 또한, 최대 평균 성적을 가진 학생의 이름과 평균 성적을 출력하시오.



```
# 학생별 평균 성적 계산하여 average 열에 저장  
df['average'] = df[['midterm', 'final', 'assignment']].mean(axis=1)  
  
# 최대 평균 성적을 가진 학생 찾기  
max_avg_student_idx = df['average'].idxmax()  
max_avg_student = df.loc[max_avg_student_idx, ['name', 'average']]  
print(f"Student with highest average score: {max_avg_student['name']}")
```

```
## Student with highest average score: Frank
```

```
print(max_avg_student)
```

```
## name      Frank  
## average   96.666667  
## Name: 5, dtype: object
```