데이터 결합: concat, merge

예제 데이터 생성

```
import pandas as pd

df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                    'B': ['B0', 'B1', 'B2']},
                   index=[0, 1, 2])

df2 = pd.DataFrame({'A': ['A3', 'A4', 'A5'],
                    'B': ['B3', 'B4', 'B5']},
                   index=[0, 1, 2])

df3 = pd.DataFrame({'C': ['C0', 'C1', 'C2'],
                    'D': ['D0', 'D1', 'D2']},
                   index=[0, 1, 2])

df4 = pd.DataFrame({'C': ['C3', 'C4', 'C5'],
                    'D': ['D3', 'D4', 'D5']},
                   index=[3, 4, 5])

# merge를 위한 예제 데이터
left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key': ['K0', 'K1', 'K4', 'K5'],
                      'C': ['C0', 'C1', 'C4', 'C5'],
                      'D': ['D0', 'D1', 'D4', 'D5']})
```

`pd.concat()`(Concatenation)

- 여러 데이터프레임을 축(axis)을 따라 단순히 이어 붙이는 작업
- 기본적으로 행(row) 방향(axis=0)으로 데이터를 쌓으며, SQL의 `UNION ALL`과 유사한 개념
    - 열(column) 방향(axis=1)으로 붙이는 것도 가능

적용 가능한 상황

- 동일한 열 구조를 가진 여러 개의 데이터프레임을 하나로 합칠 때. (e.g., 여러 기간에 걸쳐 수집된 로그 데이터, 여러 지역의 판매 데이터)
- 구조가 다른 데이터프레임들을 단순히 옆으로(열 방향으로) 나란히 붙이고 싶을 때.

주의 사항

- 기본적으로 `axis=0` (행 방향)으로 동작
- `ignore_index=True` 옵션을 사용하지 않으면 기존 인덱스가 그대로 유지되어 중복 인덱스가 발생 가능
- 열 이름이 다른 데이터프레임을 합칠 경우, 해당 열이 없는 데이터프레임에는 `NaN`이 채워짐

코드 예제

```
# 행 방향으로 결합 (기본값)
result_row = pd.concat([df1, df2])
print("--- Row-wise concat ---")
print(result_row)
'''
--- Row-wise concat ---
    A   B
0  A0  B0
1  A1  B1
2  A2  B2
0  A3  B3
1  A4  B4
2  A5  B5
'''

# 인덱스 재설정
result_row_ignore = pd.concat([df1, df2], ignore_index=True)
print("\n--- Row-wise concat with ignore_index=True ---")
print(result_row_ignore)
'''
--- Row-wise concat with ignore_index=True ---
    A   B
0  A0  B0
1  A1  B1
2  A2  B2
3  A3  B3
4  A4  B4
5  A5  B5
'''

# 열 방향으로 결합
result_col = pd.concat([df1, df3], axis=1)
print("\n--- Column-wise concat (axis=1) ---")
print(result_col)
'''
--- Column-wise concat (axis=1) ---
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
2  A2  B2  C2  D2
'''

# 열 방향으로 결합 (서로 다른 인덱스)
result_col = pd.concat([df1, df4], axis=1)
print("\n--- Column-wise concat (axis=1) ---")
print(result_col)
'''
--- Column-wise concat (axis=1) ---
     A    B    C    D
0   A0   B0  NaN  NaN
```

```
1   A1   B1  NaN  NaN
2   A2   B2  NaN  NaN
3  NaN  NaN   C3   D3
4  NaN  NaN   C4   D4
5  NaN  NaN   C5   D5
'''

# 열 방향으로 결합 (서로 다른 인덱스, 인덱스 재설정)
result_col = pd.concat([df1, df4], axis=1, ignore_index=True)
print("\n--- Column-wise concat (axis=1) ---")
print(result_col)
'''
--- Column-wise concat (axis=1) ---
     0    1    2    3
0   A0   B0  NaN  NaN
1   A1   B1  NaN  NaN
2   A2   B2  NaN  NaN
3  NaN  NaN   C3   D3
4  NaN  NaN   C4   D4
5  NaN  NaN   C5   D5
'''
```

- **결과 해석**
  - `pd.concat([df1, df2])`는 `df1` 아래에 `df2`를 그대로 붙여 6행 2열의 데이터프레임을 만듭니다. 인덱스는 `0, 1, 2, 0, 1, 2`로 유지됩니다.
  - `ignore_index=True`를 사용하면 기존 인덱스를 무시하고 `0, 1, 2, 3, 4, 5`로 새로운 인덱스를 생성합니다.
  - `axis=1`을 사용하면 `df1` 옆에 `df3`를 붙여 3행 4열의 데이터프레임을 만듭니다. 두 데이터프레임의 인덱스가 같으므로 정상적으로 옆으로 결합됩니다.
  - `df4`를 붙일 경우, 인덱스가 서로 다르므로 없는 위치는 `NaN`이 생성됩니다.
  - 앞선 경우에 `ignore_index=True`를 사용할 경우, 열 이름인 `A, B, C, D`를 무시하여 `0, 1, 2, 3`으로 설정 후 합칩니다. 즉, 데이터 형태는 다름이 없습니다.
    - 행 이름 무시하고 붙이고 싶다면, `.reset_index(drop=True)`를 우선 적용한 후 `concat()`을 진행하면 됩니다.

- **주요 하이퍼파라미터 (인자)**
  - `objs`: 결합할 데이터프레임 객체들의 리스트 또는 튜플. `[df1, df2, ...]`
  - `axis`: 결합할 축 방향. `0` (기본값)은 행 방향(위아래), `1`은 열 방향(좌우).
  - `join`: 다른 축의 인덱스를 처리하는 방법. `'outer'` (기본값)는 모든 인덱스를 포함(합집합), `'inner'`는 공통된 인덱스만 포함(교집합).
  - `ignore_index`: `True`로 설정하면 기존 인덱스를 무시하고 0부터 시작하는 새로운 인덱스를 생성. 기본값은 `False`.

`pd.merge()` (Merging/Joining)

- 하나 이상의 공통된 열(key)을 기준으로 데이터프레임을 결합
- SQL의 `JOIN` 문과 매우 유사하며, `inner`, `left`, `right`, `outer` 등 다양한 조인 방식을 지원
- 관계형 데이터베이스의 테이블들을 조인하는 것과 같은 원리

적용 가능한 상황

- 서로 다른 정보를 담고 있지만, '고객 ID', '상품 코드' 등 공통된 식별자(key)를 가진 데이터프레임들을 결합할 때. (e.g., 고객 정보 테이블과 구매 내역 테이블을 '고객 ID'로 결합)
- 두 데이터 간의 관계(일대일, 일대다, 다대다)를 기반으로 정보를 풍부하게 만들고 싶을 때.

주의사항

- `on` 인자를 명시하지 않으면, 두 데이터프레임에 공통으로 존재하는 모든 열을 조인 키로 사용
- 조인 키의 이름이 두 데이터프레임에서 다를 경우 `left_on`과 `right_on`을 사용

코드 예제

```
# Inner Join (기본값): 양쪽에 모두 존재하는 키만 결합
inner_join = pd.merge(left, right, on='key')
print("--- Inner Join ---")
print(inner_join) # K0, K1만 포함
'''
--- Inner Join ---
  key   A   B   C   D
0  K0  A0  B0  C0  D0
1  K1  A1  B1  C1  D1
'''

# Left Outer Join: 왼쪽(left) 데이터프레임의 모든 키를 유지
left_join = pd.merge(left, right, on='key', how='left')
print("\n--- Left Join ---")
print(left_join) # K0, K1, K2, K3 모두 포함. K2, K3의 C, D열은 NaN
'''
--- Left Join ---
  key   A   B    C    D
0  K0  A0  B0   C0   D0
1  K1  A1  B1   C1   D1
2  K2  A2  B2  NaN  NaN
3  K3  A3  B3  NaN  NaN
'''

# Outer Join: 양쪽의 모든 키를 유지
outer_join = pd.merge(left, right, on='key', how='outer')
print("\n--- Outer Join ---")
print(outer_join) # K0, K1, K2, K3, K4, K5 모두 포함
'''
--- Outer Join ---
  key    A    B    C    D
0  K0   A0   B0   C0   D0
1  K1   A1   B1   C1   D1
2  K2   A2   B2  NaN  NaN
3  K3   A3   B3  NaN  NaN
4  K4  NaN  NaN   C4   D4
5  K5  NaN  NaN   C5   D5
'''
```

- **결과 해석**:
  - **Inner Join**: `left`와 `right`에 공통으로 존재하는 `key`인 'K0'와 'K1'에 대해서만 데이터가 결합됩니다.
  - **Left Join**: `left` 데이터프레임의 모든 `key`('K0'~'K3')가 유지됩니다. `right`에 해당 `key`가 없는 'K2', 'K3'의 경우, `right`에서 온 열(`C`, `D`)의 값은 `NaN`으로 채워집니다.
  - **Outer Join**: 두 데이터프레임에 존재하는 모든 `key`('K0'~'K5')가 결과에 포함됩니다. 짝이 없는 데이터는 `NaN`으로 채워집니다.

- **주요 하이퍼파라미터 (인자)**
  - `left`, `right`: 결합할 두 개의 데이터프레임.
  - `how`: 조인 방식을 지정. `'inner'` (기본값), `'left'`, `'right'`, `'outer'`.
  - `on`: 조인 키로 사용할 열 이름. 두 데이터프레임에 모두 존재해야 함.
  - `left_on`, `right_on`: 조인 키의 이름이 왼쪽과 오른쪽 데이터프레임에서 다를 때 각각 지정.
  - `suffixes`: 조인 키를 제외한 나머지 열 중 이름이 겹칠 경우, 각 열 이름에 붙일 접미사. 기본값은 `('_x', '_y')`.
  - `indicator`: `True`로 설정하면, `_merge`라는 열이 추가되어 각 행이 어떤 조인에 의해 생성되었는지(`'left_only'`, `'right_only'`, `'both'`)를 보여줌.

장단점 및 대안

함수	장점	단점	대안/팁
`pd.concat()`	사용법이 간단하고 직관적임. 여러 개의 데이터프레임을 한 번에 합칠 수 있음.	공통 키 기반의 복잡한 데이터 연결 로직에는 부적합.	`df.append()`는 두 개의 데이터프레임을 행 방향으로 합치는 더 간단한 방법이었으나, Pandas 1.4.0부터 공식적으로 deprecated 되었으므로 `concat` 사용이 권장됨.
`pd.merge()`	SQL 조인과 유사하여 관계형 데이터 처리에 매우 강력하고 유연함.	두 개의 데이터프레임만 결합 가능 (여러 개를 합치려면 연쇄적으로 사용해야 함). 대용량 데이터 조인 시 메모리 사용량과 속도에 주의 필요.	`df.join()` 메서드는 인덱스를 기준으로 조인할 때 더 편리한 인터페이스를 제공함. `merge`는 내부적으로 `join`을 활용하는 더 일반적인 함수임.

`merge` vs `join` 메서드

- `pd.merge()`는 함수이고, `df.join()`은 데이터프레임의 메서드입니다.
- `merge`는 기본적으로 **열(column)**을 기준으로 조인합니다.
- `join`은 기본적으로 **인덱스(index)**를 기준으로 조인합니다.
- `df1.join(df2)`는 `pd.merge(df1, df2, left_index=True, right_index=True, how='left')`와 유사하게 동작합니다. 따라서 인덱스 기반 조인이 필요할 때는 `join`이 더 간결한 코드를 제공합니다.

`pivot()` 함수 (참고)

- `pivot()` : 집계 후 데이터프레임 구조 변형해서 조회하는데 사용 (결합은 아님)
  - 순서 : 1→ groupby / 2→ pivot

```
# 1) 매장1의 일별 카테고리별 판매량을 집계
temp = pd.merge(sales1, products)
temp2 = temp.groupby(['Date', 'Category'], as_index = False)['Qty'].sum()
temp2

# 2) pivot (index = 행으로 삼을 값 / columns = 열로 삼을 값 / values = 나타낼 값)
temp3 = temp2.pivot(index = 'Category', columns = 'Date', values = 'Qty')
temp3
```