|  |  |
| --- | --- |
| 교육 제목 | **함수 매핑 & 데이터 병합** |
| 교육 일시 | 21.10.05 |
| 교육 장소 | 오프라인 (영우글로벌러닝) |
| **교육 내용** | |
| 오전 | **▣ 함수 매핑**  -------------------------------------------------------------------------------------------  **∇ 시리즈를 변수로 함수 실행하기**  \* apply() method를 활용하여 시리즈의 각 원소에 동일한 함수 실행  \* 함수를 먼저 설정해줘야 함.  \* df['변수로 쓰일 컬럼명'].appply(함수명) ------------> 특정 열에 대해 함수 실행  \* df.appply(함수명, axis=0) ------------------> 각각의 모든 행 기준으로 함수 실행  \* df.appply(함수명, axis=1) ------------------> 각각의 모든 열 기준으로 함수 실행  \* df.apply(lambda x: x['행 이름'] + x['행 이름'], axis=0) ------> 행과 행끼리의 계산  \* df.apply(lambda x: x['열 이름'] + x['열 이름'], axis=1) ----> 열과 열 사이의 계산  \* df['변수로 쓰일 컬럼'].apply(lambda x: x+10)  \* df['변수로 쓰일 컬럼'].apply(lambda x: 함수명)  \* df['변수로 쓰일 컬럼'].apply(lambda x: x.max()) -> 선택 컬럼에서 최대값을 구함.  \* df\_map = df.applymap(함수명)  -------------------------------------------------------------------------------------------  **∇ pipe**  \* 계산 값들을 연결해서 가지고 옴  \* 다른 함수에서의 output을 input으로 받아옴  -------------------------------------------------------------------------------------------  **∇ 필터링**  \* 원하는 조건의 데이터만 추출: mask 또는 isin() 등 활용  \* mask 사용하기  **Ex)**  mask1 = (titanic.age <= 10) | (titanic.age >= 60)  df\_alive = titanic.loc[mask1, ['age', 'fare', 'sex', 'class', 'alive']]  df\_alive    \* mask 사용 안하기  **Ex)**  df\_female = titanic.loc[(titanic.age <10) & (titanic.sex == 'female'), :]  \* isin() 사용하기  **Ex)**  mask3 = titanic['sibsp'].isin([3, 4, 5])  df\_isin = titanic.loc[mask3, :]  ------------------------------------------------------------------------------------------- |
| 후 | **▣ 데이터 병합**  -------------------------------------------------------------------------------------------  **∇ 데이터 프레임과 데이터 프레임 합치기**  \* pd.concat([df1, df2]) ----------------------------> 단순하게 위 아래로 붙임  \* pd.concat([df1, df2], ignore\_index=True) ----------> 기존의 인덱스들을 무시하고 새로운 인덱스 부여  \* pd.concat([df1, df2], axis = 1) --------------------> 단순하게 옆으로 붙여줌.  \* pd.concat([df1, df2], axis = 1, join='inner') -------> join 키워드 -> join = outer: 모두 보여줌(디폴트 값), join = inner: 교집합만 보여줌  -------------------------------------------------------------------------------------------  **∇ 데이터 프레임과 시리즈 붙이기**  \* pd.concat([df1, sr1], axis = 1) ----------> 데이터 프레임에 시리즈가 옆으로 붙음  \* pd.concat([df1, sr1], axis = 0) -------> 데이터 프레임에 시리즈가 위아래로 붙음  \* pd.concat([df2, sr2], axis = 1) -----------> df2와 sr2 좌우 열방향으로 붙이기  (시리즈에 인덱스명이 존재 해서 열 명에 맞게 합쳐짐)  \* pd.concat([sr1, sr3], axis = 1) -----> sr1과 sr3을 옆으로 붙이기  (결과: 데이터 프레임)  \* pd.concat([sr1, sr3], axis = 0) ----------> sr1과 sr3 위아래 붙이기 (결과: 시리즈)  -------------------------------------------------------------------------------------------  **∇ 데이터 프레임 특정 컬럼을 기준으로 교집합 / 합집합으로 붙이기 (교집합만 붙이기)**  \* pd.merge(df1, df2) -------------------------------------> on = '컬럼명'이 없으면, 첫 번째 column을 기준으로 교집합만 붙임  \* pd.merge(df1, df2, how='outer', on='기준이 될 컬럼명') --------> 'id' column을 기준으로 교집합이든 합집합이든 싹다 붙여버림  \* pd.merge(df1, df2, how='left', left\_on = 'id', right\_on='id') --> df1 기준으로  교집합만 출력 (오른쪽에 없으면 NaN으로 출력)  \* pd.merge(df1, df2, how='right', left\_on = 'id', right\_on='id') -> 오른쪽 데이터 프레임(df2) 기준으로 교집합만 출력 (왼쪽에 없으면 NaN으로 출력)  \* df1.join(df2) --------------------------------------------> join() -> 인덱스 기준 교집합  -------------------------------------------------------------------------------------------  **∇ Group화 하기 -> groupby(), get\_group()**  \* df.groupby(['컬럼명']) -------------------------------> df.groupby(['기준이 되는 열 이름']), 열 기준으로 분할  \* grouped.get\_group('컬럼에 속한 특정 데이터')) -----------> '컬럼에 속한 특정 데이터' 들만 가지고 옴  \* grouped\_std = grouped.std() ---------------------------> 그룹의 표준편차를 구함. 이런식으로 그룹을 계산, 필터링 등을 할 때 쓰임.  ------------------------------------------------------------------------------------------- |