



[베이직 반] 판다스 (2)

데이터 전처리(판다스) (2)

0) 학습 목표

1) 조건 필터링 나도 해볼래!

Titanic 데이터 실습

정리

2) 데이터 정렬 및 정제는 어떻게 하는걸까?

Titanic 데이터 실습

3) 결측치 처리 꼭 해야 하니?

Titanic 데이터 실습

4) 중복 제거 반드시 해야지

Titanic 데이터 실습



실습 데이터

타이타닉 데이터 :

https://drive.google.com/file/d/11AAborf8AHbFdmy6QtVUXQZvTH_FY8wE/view?usp=sharing

중복 타이타닉 데이터 : https://drive.google.com/file/d/1AkfGLBDj0jOc_E-qFgBtfSxc1lq-ooH/view?usp=sharing

데이터 전처리(판다스) (2)

0) 학습 목표

Titanic 승객 데이터([train.csv](#))를 활용하여 Pandas의 핵심 데이터 다루기 기법 4가지 학습

- 조건 필터링: 원하는 조건에 맞는 행만 선택하기 (불린 인덱싱)
- 데이터 정렬 및 정제: 값을 기준으로 데이터프레임 정렬하고, 분석에 불필요한 데이터 제거하기
- 결측치 처리: 비어있는 데이터(NaN)의 확인, 제거 및 채우기
- 중복 제거: 중복된 데이터 확인 및 삭제로 데이터 깔끔하게 만들기

1) 조건 필터링 나도 해볼래!

- 데이터 분석에서 **필터링**은 말 그대로 원하는 조건에 맞는 데이터만 **걸러내는** 작업
- How? Pandas에서는 이러한 필터링을 **불린 인덱싱(Boolean Indexing)** 기법 → 불린 인덱싱이란 **조건식의 결과(True/False)를 이용하여 True인 행만** 선택하는 방법
- `df[조건]` or `df.loc[조건]` → 조건은 **불린 시리즈**
- 나이와 도시 정보가 있는 데이터프레임에서 나이가 30 이상인 행만 필터링

```
import pandas as pd

data = {'이름': ['철수', '영희', '민수', '수지', '호영'],
        '나이': [25, 32, 40, 28, 18],
        '도시': ['서울', '대전', '인천', '대구', '부산']}
df = pd.DataFrame(data)
print(df)
```

1. 조건식 : 나이 조건(`df['나이'] >= 30`)으로 불린 Series 생성
2. 대괄호 `[]` 로 인덱싱

```
# 나이가 30 이상인 행만 선택
df_filtered = df[df['나이'] >= 30]
print(df_filtered)
```

- 조건식을 여러 개 결합 가능
- 여러 조건을 결합할 때는 **AND (&), OR (|), NOT (~)** 연산자를 사용하고, **각 조건을 괄호로 묶어 표현**
- 도시가 '서울'이고 나이가 30 이상인 경우
- 조건식 : `(df['도시']=='서울') & (df['나이'] >= 30)` 인 불린 Series 생성
- 대괄호 `[]` 로 인덱싱

Titanic 데이터 실습

```
# Titanic 데이터 로드
titanic_df = pd.read_csv('train.csv')
titanic_df.head(5)
```

- 예제 1: 나이(Age)가 70세를 넘는 승객 필터링 – Titanic 승객 중 고령자만 뽑아봅니다.

▼ 정답

```
# Age가 70 초과인 승객만 추출
elderly_df = titanic_df[titanic_df['Age'] > 70]
print(elderly_df[['Name', 'Age', 'Survived']])
print("고령 승객 수:", len(elderly_df))
```

- 예제 2: 1등석이면서 여성인 승객 필터링

▼ 정답

```
# 1등실(Pclass=1) AND 여성(Sex='female') 승객만 추출
firstclass_female_df = titanic_df[(titanic_df['Pclass'] == 1) & (titanic_df['Sex'] == 'female')]
print(firstclass_female_df[['Name', 'Pclass', 'Sex']].head(3))
print("1등실 여성 승객 수:", len(firstclass_female_df))
```

- 예제 3: 나이가 15세 미만 또는 80세 이상인 승객 필터링

▼ 정답

```
young_or_old_df = titanic_df[(titanic_df['Age'] < 15) | (titanic_df['Age'] >= 80)]
print(young_or_old_df[['Name', 'Age']].head(5))
print("15세 미만 또는 80세 이상 승객 수:", len(young_or_old_df))
```

1. 퀴즈 1: 생존자(`Survived = 1`) 중 나이가 50세 이상인 승객만 추출하려면 어떻게 해야 할까요?
2. 퀴즈 2: 2등실(`Pclass = 2`) 또는 3등실(`Pclass = 3`)에 탑승한 남성 승객만 필터링하는 코드를 작성해보세요.

▼ 정답

```
# 퀴즈 1 정답: 생존자 & 50세 이상
survived_over50 = titanic_df[(titanic_df['Survived'] == 1) & (titanic_df['Age'] >= 50)]
print(survived_over50[['Name', 'Age', 'Survived']].head())
print("조건에 맞는 승객 수:", len(survived_over50))
```

```
# 퀴즈 2 정답: 2등실 또는 3등실 & 남성
lowclass_male = titanic_df[((titanic_df['Pclass'] == 2) | (titanic_df['Pclass'] == 3)) & (titanic_df['Sex'] == 'male')]

print(lowclass_male[['Pclass','Sex']].head())
print("조건에 맞는 승객 수:", len(lowclass_male))
```

정리

- **불린 인덱싱(Boolean Indexing):** `df[조건식]` 형태로 사용하며, 조건식을 만족(True)하는 행만 선택합니다. 조건식은 `==, !=, >, <, >=, <=` 연산자를 사용합니다.
- **다중 조건 결합:** 여러 조건을 결합할 때는 **AND**는 `&`, **OR**는 `|`, **NOT**는 `~`를 사용합니다. 각 조건을 반드시 소괄호 `()`로 묶어야 하며, 파이썬의 `and/or` 키워드는 Pandas에서는 사용할 수 없으니 주의하세요.
- **참/거짓 배열 활용:** `df['Age'] > 30` 과 같은 표현은 각 행에 대해 조건을 평가한 **불린 배열(Series)**을 돌려주며, 이를 다시 `df[...]`에 사용함으로써 조건을 만족하는 행을 얻습니다.
- **주의:** 조건 결합 시 괄호를 빼먹거나 `and/or`를 쓰면 오류가 발생합니다. 또한 `df['컬럼'] == 값` 과 같이 이중 등호(`==`)를 사용해야 하며, `=`는 대입으로 쓰이므로 혼동하지 않도록 합니다.

2) 데이터 정렬 및 정제는 어떻게 하는걸까?

데이터 정렬

- 데이터를 어떤 기준에 따라 순서대로 나열하는 것
- 데이터프레임도 특정 열의 값을 기준으로 오름차순 또는 내림차순으로 배치
- `df.sort_values(by="열", ascending=True or False)`

데이터 정제

- 데이터를 깨끗하게 다듬는 작업 → 부정확하거나 불완전한 데이터를 탐지하여 수정 또는 삭제
- 원본 데이터에는 분석에 불필요한 컬럼이 있거나 잘못된 값, 이상치가 포함
- 열 삭제 - `df.drop(columns='열')`
- 행 삭제 - `df.drop(index=인덱스번호)`
- 열 이름 변경 - `df.rename(columns={'현재 열': '바꿀 열'})`

간단한 예제

```
data2 = {'이름': ['철수', '영희', '민수', '수지'],
        '점수': [85, 92, 78, 92]}
df2 = pd.DataFrame(data2)
df2
```

- **정렬 예제:** 예를 들어 점수(점수 열)를 내림차순(큰 값 → 작은 값)으로 정렬하려면 `ascending=False` 옵션:

```
# '점수' 열을 기준으로 내림차순 정렬
sorted_df = df2.sort_values(by='점수', ascending=False)
print(sorted_df)
```

- 만약 **동점자 정렬 기준**을 추가하고 싶다면 `by` 에 목록을 넣어 다중 기준 정렬도 가능
- 예를 들어, `df2.sort_values(by=['점수', '이름'], ascending=[False, True])` → 점수 내림차순 우선 정렬하되 점수가 같으면 이름 오름차순(가나다순)으로 정렬

• 정제 예제

- **불필요한 열 제거** → '메모'라는 쓸모없는 열을 `df.drop()` 메서드를 사용하여 해당 열을 삭제

```
df2['메모'] = ['임시', '필요없음', '임시', '필요없음'] # 예시로 메모 열 추가
print(df2)
```

```
df2_cleaned = df2.drop(columns='메모')
print(df2_cleaned)
```

- **불필요한 행 삭제** → 2번째 행 삭제

```
df2_cleaned = df2.drop(index=1)
print(df2_cleaned)
```

- **열 이름 변경** → "점수" 열을 "수학 점수"로 변경

```
df2_cleaned = df2.rename(columns={'점수': '수학 점수'})
print(df2_cleaned)
```

Titanic 데이터 실습

- 예제 1: 티켓 요금(Fare) 순 정렬(내림차순)

- ▼ 정답

```
# 티켓 요금(Fare) 내림차순으로 데이터 정렬 (가장 비싼 요금 순)
fare_sorted_df = titanic_df.sort_values(by='Fare', ascending=False)
print(fare_sorted_df[['Name', 'Fare']].head(5))
```

- 예제 2: 나이와 이름으로 다중 정렬 – 나이(Age) 오름차순, 이름(Name) 오름차순

- ▼ 정답

```
# 나이 오름차순, 이름 오름차순으로 정렬
age_name_sorted_df = titanic_df.sort_values(by=['Age', 'Name'], ascending=[True, True])
print(age_name_sorted_df[['Name', 'Age']].head(5))
print(age_name_sorted_df[['Name', 'Age']].tail(5))
```

- 예제 3: 불필요한 열 제거

- Titanic 데이터에는 실제 분석에 당장 쓰이지 않거나 결측치가 너무 많은 열 존재
- **Cabin(객실번호)** 열은 값이 대부분 비어 있고, **Ticket(티켓 번호)**이나 **PassengerId** 열 제거

- ▼ 정답

```
# 분석에 불필요한 열 제거: 'Ticket', 'Cabin', 'PassengerId' 등을 제거
titanic_df_cleaned = titanic_df.drop(columns=['PassengerId', 'Ticket', 'Cabin'])
print("제거 후 컬럼 목록:", titanic_df_cleaned.columns)
```

퀴즈: Titanic 데이터 정렬/정제 관련 문제를 풀어봅시다.

1. 퀴즈 1: 가장 **나이가 많은 승객 5명**의 기록을 보고 싶습니다. Titanic 데이터프레임 `titanic_df` 를 나이 순으로 내림차순 정렬하여 상위 5명을 출력하는 코드를 작성하세요.
2. 퀴즈 2: 분석에 필요 없는 '**Name**' 열을 **DataFrame**에서 **제거(drop)**하는 코드를 작성해보세요.
3. 퀴즈 3: 기존 "Age", "Fare" 컬럼 명을 "나이", "요금" 으로 변경하세요.

- ▼ 정답

```
# 퀴즈 1 정답: 나이 내림차순 정렬 후 상위 5명 출력
oldest5 = titanic_df.sort_values(by='Age', ascending=False).head(5)
print(oldest5[['Name', 'Age']])


# 퀴즈 2 정답: 'Name' 열 제거
titanic_df_no_name = titanic_df.drop('Name', axis=1)
print("Name 열 제거 후 컬럼:", titanic_df_no_name.columns)

# 퀴즈 3 정답: 컬럼 명 변경
titanic_df_change_columns = titanic_df_no_name.rename(columns={"Age": "나이", "Fare": "요금"})
print("열 이름 변경 후:", titanic_df_change_columns)
```

3) 결측치 처리 꼭 해야 하니?

- 결측치(Missing Value)란 말 그대로 **비어 있는 값** → **NaN** (Not a Number) 또는 **None** 으로 표시
- 결측치는 데이터 분석과 모델링에 문제를 일으킬 수 있기 때문에, 이를 **어떻게 처리할지 결정할지?**

결측치를 다루는 일반적인 방법은 두 가지:

1. **결측치가 있는 행/열 삭제** – 결측치가 극히 일부거나 해당 행/열이 크게 중요하지 않다면 아예 제거
2. **결측치를 특정 값으로 대체 (대치, imputation)** – 결측치가 많은 경우 삭제하면 데이터 손실 
 - a. 수치형 변수 : 평균값이나 최빈값 등 **합리적인 값으로 채워 넣기**
 - b. 범주형 변수 : “Unknown” 이라는 **합리적인 값으로 채워 넣기**

어떤 방법을 택할지는 상황에 따라 다릅니다.

- 결측이 **적은 비율**이면 삭제가 간편하고, **많은 비율**이면 대체를 고려
- 결측치가 **어떻게 발생했는지** (무작위인지 어떤 패턴인지) 도 중요한 판단 기준 → 현재는 모름

간단한 예제

```
# 결측치가 포함된 예제 Series
s = pd.Series([10, None, 30, None, 50])
```

```
print("원본 Series:")
print(s)
```

위 Series `s`에는 인덱스 1과 3에 `NaN`이 있습니다. 이 결측치를 처리하는 방법을 연습해보겠습니다.

• 결측치 확인

- `isnull()` (또는 `isna()`) 메서드는 각 값이 결측인지 True/False로 확인
- `sum()` 을 하면 True를 1로 간주하여 결측치 개수 count 가능

```
print("각 원소의 결측 여부:")
print(s.isnull())
print("결측치 개수:", s.isnull().sum())
```

• 결측치 제거

- `dropna()` 메서드는 NaN이 있는 **행(혹은 값)들을 제거**
- Series의 경우 NaN 값을 제거하고 남은 값만 반환
- DataFrame의 경우에는 기본적으로 결측치가 있는 **행을 날려버리며**, `axis=1` 옵션을 주면 열을 날릴 수도 있습니다.

```
s_dropped = s.dropna()
print("결측치 제거 후 Series:")
print(s_dropped)
```

• 결측치 대체

- `fillna()` 메서드는 결측값을 주어진 값으로 채움
- 예를 들어 NaN을 모두 0으로 채우면:

```
s_filled_zero = s.fillna(0)
print("NaN을 0으로 채운 Series:")
print(s_filled_zero)
```

보통 0보다는 **평균값, 중앙값** 또는 **이웃한 값** 등 데이터 특성에 맞는 값으로 채우는 경우가 대부분

예를 들어 이 Series의 평균은 30이므로 NaN을 30으로 채울 수도 있고, 중앙 값으로 채울 수도 있음.


```
print("평균값으로 채우기:", s.fillna(s.mean()))
print("중앙값으로 채우기:", s.fillna(s.median()))
```

Titanic 데이터 실습

- 결측치 확인

- Titanic 데이터 모든 열의 결측치 확인

- ▼ 정답

```
# 각 열별 결측치 개수 확인
print(titanic_df.isnull().sum())
```

- 결측치가 있는 행 조회: Embarked(승선항구) 정보가 없는 승객이 누군지 확인

- ▼ 정답

```
print(titanic_df[titanic_df['Embarked'].isnull()][['Name','Pclass','Embarked']])
```

- 결측치 처리 전략 수립: 각 열의 결측치에 대해 처리 방법을 결정해야 합니다:

- Embarked (2건 결측): 어떻게 처리할래?

- ▼ 아이디어

전체 891명 중 2명만 없으므로 해당 행을 삭제하거나, 혹은 가장 흔한 승선항(C가 Cherbourg, Q=Queenstown, S=Southampton 중 S가 가장 많음)으로 대체할 수 있습니다. 수가 적으니 삭제해도 무방하지만, 예시로 최빈값으로 채워보겠습니다.

- Age (177건 결측): 어떻게 처리할래?

- ▼ 아이디어

- 20%가 비어 있으므로 그냥 삭제하면 데이터 손실이 큼니다. 따라서 평균 또는 중앙값으로 채우기를 고려합니다. Age는 한쪽으로 치우친 분포가 있을 수 있어 중앙값(Median)으로 채우는 것이 흔한 전략입니다.

- Cabin (687건 결측): 어떻게 처리할래?

- ▼ 아이디어

- 77%가 비어 있으므로 유효한 정보가 거의 없습니다. 대부분 분석에서 Cabin은 그냥 열 자체를 삭제(drop)하거나, 'Unknown' 등으로 채우기도 하지만 의미가 크지 않습니다. 이번에는 Cabin 열을 삭제하는 방향으로 정제하겠습니다.

- 결측치 채우기 (Embarked)

- Embarked의 최빈값을 찾아 채워보자. → 값 분포 확인 후 → 최빈 값으로 대체 → 대체 후 결측 확인

- ▼ 정답

```
# Embarked 값 분포 확인 -> 최빈 값 확인
print(titanic_df['Embarked'].value_counts())
```

```
# Embarked 결측치를 최빈값 'S'로 채우기
titanic_df['Embarked'] = titanic_df['Embarked'].fillna('S')
print(titanic_df['Embarked'].isnull().sum())
```

- 결측치 채우기 (Age)

- Age 결측 177건은 **중앙값**으로 채워보자 → 먼저 중앙값 구하고 → 해당 값으로 채우기 → 대체 후 결측 확인

- ▼ 정답

```
median_age = titanic_df['Age'].median()
print("나이 중앙값:", median_age)
titanic_df['Age'] = titanic_df['Age'].fillna(median_age)
print("Age 결측치 개수 (채운 후):", titanic_df['Age'].isnull().sum())
```

- 결측치 제거 (Cabin): Cabin 열 삭제

- ▼ 정답

```
titanic_df = titanic_df.drop(columns='Cabin')
titanic_df.columns # Cabin 열 삭제 확인
```

- 결측치 처리 결과 확인

```
print(titanic_df.isnull().sum())
```

퀴즈: 결측치 처리 개념을 확인해봅시다.

1. 퀴즈: Titanic 데이터에서 각 열의 결측치 개수를 한 번에 파악하려고 합니다. 어떤 코드를 사용하면 **열별 결측치 개수**를 알 수 있을까요?
2. 퀴즈: DataFrame `df` 에서 결측치가 하나라도 있는 **모든 행을 삭제**하려면 어떤 메서드를 이용해야 할까요?

▼ 정답

```
# 퀴즈 1 정답: 열별 결측치 개수 확인
print(df.isnull().sum())
```

```
# 퀴즈 2 정답: 결측치 포함 행 삭제
df_dropped = df.dropna() # 어떤 열이든 NaN이 있는 행은 제거
```

4) 중복 제거 반드시 해야지

- 중복 데이터란 동일한 값의 행이 데이터셋에 반복되어 나타나는 경우
- 중복이 생기는 이유는 여러 가지인데
 - 데이터를 여러 소스로부터 합치는 과정에서 **같은 사람이 두 번 기록**되었을 때,
 - 센서 오작동 등으로 **동일한 로그가 반복** 저장되었을 때,
 - 크롤링이나 입력 실수로 **똑같은 행이 중복 입력**되었을 때 등이 있습니다.

특정 케이스가 과대대표되어 잘못된 결과를 초래 → 어떤 고객이 이중으로 기록되어 구매 횟수가 2배로 집계된다면 실제보다 부풀려진 통계 발생

간단한 예제

```
df3 = pd.DataFrame({'제품': ['A', 'B', 'B', 'C'],
                    '가격': [1000, 2000, 2000, 3000]})
print(df3)
```

	제품	가격
0	A	1000
1	B	2000
2	B	2000
3	C	3000

- 중복 여부 확인

- `df.duplicated()` 각 행이 이전에 이미 나타난 행과 중복인지 True/False로 표시하는 불린 Series를 반환
- 기본적으로 첫 등장 행은 False, 그 다음부터의 중복행은 True로 표시
- `df.duplicated().sum()` 을 사용하면 전체 중복 행 개수 파악

```
print(df3.duplicated())
print(df3.duplicated().sum())
```

• 중복 행 제거

- `df.drop_duplicates()` 중복된 행을 제거하고 유일(unique)한 행만 남긴 DataFrame을 반환
- 기본 설정에서는 중복이 발생하면 가장 처음 나타난 행을 남기고 그 이후 중복된 행들을 제거 (`keep='first'`).

```
df3_unique = df3.drop_duplicates()
print(df3_unique)
```

- 참고: `drop_duplicates` 에는 `keep='last'` 옵션을 주면 반대로 마지막 값을 남기고 앞의 중복들을 제거 하며, `keep=False` 로 하면 중복인 행을 모두 제거해버립니다.
- `subset` 매개변수로 특정 열들만 고려하여 중복 판정할 수도 있습니다 (예: `subset=['제품']` 이면 제품 명이 같으면 중복으로 보고 제거).

Titanic 데이터 실습

실제 Titanic 데이터의 경우 동일한 승객이 두 번 기록되는 일은 없습니다.

그래서 튜터가 임의로 만든 Titanic 데이터를 써봅시다.

- 중복 여부 확인 후 중복 제거 → 제거 전후 행 개수 비교

▼ 정답

```
print(titanic_df.duplicated().sum())
titanic_unique_df = titanic_df.drop_duplicates()
print("중복 제거 전 행 개수:", len(titanic_df))
print("중복 제거 후 행 개수:", len(titanic_unique_df))
```

- 퀴즈: DataFrame `df` 에서 중복된 행을 모두 제거하려고 합니다. 어떤 메서드를 사용하면 될까요? 또한, 첫 번째 등장만 남기고 나머지 중복을 지우는 것과, 아예 중복된 모든 행을 제거하는 것의 차이를 설명해보세요.

▼ 정답

정답: `df.drop_duplicates()` 메서드를 사용합니다. 이 메서드는 기본 설정(`keep='first'`)에서 중복되는 행 중 첫 번째만 남기고 나머지를 제거합니다. 만약 중복된 행을 모두 제거하고 하나도 남기지 않으려면 `df.drop_duplicates(keep=False)` 라고 설정해야 합니다.



미니 과제

1. H&M
 - a. 데이터
 - b. 실습 파일 :
 - c. 정답 :
2. 서울시 부동산 데이터
 - a. 데이터 :
 - b. 실습 파일 :
 - c. 정답 :



자료

1. 자료 PDF :
2. 이전 전처리 라이브 세션 자료 :

