AI활용프로그래밍

Week 12. 필터링·정렬

조건 검색 · sort_values · query + With AI EDA

학습 목표

- Pandas DataFrame에서 조건 필터링(boolean indexing)을 수행할 수 있다.
- 정렬(sort_values, sort_index)과 상위/하위 추출(nlargest 등)을 사용할 수 있다.
- LLM을 활용해 EDA(탐색적 분석) 질문을 만들고 코드를 검증한다.

오늘의 구성

Boolean indexing 기본
복합 조건(&, |)과 문자열 조건
정렬(sort_values/sort_index)과 Top-N
결측치/경고(SetWithCopy 등) 맛보기
With AI 실습 + 연습문제 + 미니 퀴즈

워밍업: 원하는 행만 골라내기

```
import pandas as pd
df = pd.DataFrame({'name':['a','b','c'],
'score':[70,85,90]})
# TODO: score가 80 이상인 행만 선택
```

핵심 포인트

- 데이터 분석의 대부분은 '조건으로 고르기'에서 시작한다.
- 필터 → 정렬 → 요약의 흐름을 만들자.
- AI에게: '원하는 조건'을 자연어로 설명하고 코드로 변환하게 해보자.

Step 1. Boolean indexing 기본 (df[조건])

해야 할 것

- 조건을 먼저 변수로 만든다
- 조건 결과(True/False)를 출력해 확인
- df[cond]로 필터

체크

b, c 행만 남고(score가 80 이상), cond는 True/False로 출력된다.

실행 코드

```
import pandas as pd

df = pd.DataFrame({'name':['a','b','c'],
'score':[70,85,90]})

cond = df['score'] >= 80
print(cond)

high = df[cond]
print(high)
```

Step 2. 복합 조건 (&, |) + 괄호

해야 할 것

- and/or 대신 &/| 사용(Pandas)
- 각 조건은 괄호로 묶는다
- NaN이 있으면 결과가 달라질 수 있음

체크

score가 80 이상 90 미만인 b 행만 출력된다.

실행 코드

```
import pandas as pd

df = pd.DataFrame({'name':['a','b','c'],
'score':[70,85,90]})

cond = (df['score'] >= 80) & (df['score'] < 90)
print(df[cond])
```

Step 3. 자주 쓰는 조건: isin / between / str.contains

해야 할 것

- isin: 특정 값 집합
- between: 범위
- contains: 문자열 포함(na=False로 안전)

체크

각 print마다 필터된 DataFrame이 나온다(contains는 결측이 있어도 에러가 나지 않는다).

실행 코드

```
import pandas as pd
df = pd.DataFrame({'region':['A','B','C',
None], 'score':[70,85,90,88],
'name':['kim','lee','park','choi']})
print(df[df.region.isin(['A','B'])])
print(df[df.score.between(80, 90)])
print(df[df.name.str.contains('k', na=False)])
```

Step 4. 정렬 sort_values (값 기준)

해야 할 것

- ascending=False면 내림차순
- 정렬 후 head()로 Top-N 보기
- 원본 변경(inplace)은 신중

체크

score가 큰 순서로 정렬되고, head(2)는 상위 2행만 보여준다.

실행 코드

```
import pandas as pd

df = pd.DataFrame({'name':['a','b','c'],
'score':[70,85,90]})

print(df.sort_values('score',
ascending=False))
print(df.sort_values('score',
ascending=False).head(2))
```

Step 5. Top-N: nlargest / nsmallest

해야 할 것

- 전체 정렬 없이 상/하위 뽑기
- 데이터가 큰 경우 효율적
- 동점(ties) 처리 확인

체크

nlargest는 상위 점수 2개, nsmallest는 하위 점수 2개가 출력된다.

실행 코드

```
import pandas as pd

df = pd.DataFrame({'name':['a','b','c','d'],
'score':[70,85,90,85]})

print(df.nlargest(2, 'score'))
print(df.nsmallest(2, 'score'))
```

Step 6. 필터 + 정렬 파이프라인(한 줄)

해야 할 것

- df[조건] 뒤에 .sort_values()
- 디버깅이 어려우면 중간 변수를 사용
- shape/head로 중간 확인

체크

score>=80만 남긴 뒤 내림차순 정렬되고, 상위 3개가 출력된다.

실행 코드

```
import pandas as pd
df =
pd.DataFrame({'name':['a','b','c','d'],
'score':[70,85,90,88]})
result = (df[df['score'] >=
80].sort_values('score',
ascending=False))
print(result.head(3))
```

Step 7. 결측치(NaN) 기본

해야 할 것

- isna()로 결측 확인
- dropna()는 "의미" 확인 후 사용
- fillna()로 대체값 넣기

체크

isna는 True/False를, dropna는 결측 행 제거 결과를 보여준다.

실행 코드

```
import pandas as pd

df = pd.DataFrame({'score':[70, None, 90]})

print(df['score'].isna())
print(df.dropna())
print(df.fillna(0))
```

Step 8. loc / iloc로 행·열 선택

해야 할 것

- loc: 라벨 기반(조건 + 컬럼 선택에 자주 사용)
- iloc: 위치 기반(0번째 행/1번째 열 등)
- 필터 후 필요한 컬럼만 선택

체크

loc 결과는 name/score 두 컬럼만, iloc는 첫 행의 score(70)가 출력된다.

실행 코드

```
import pandas as pd

df = pd.DataFrame({'name':['a','b','c'],
'score':[70,85,90]})

print(df.loc[df['score']>=80,
['name','score']])
print(df.iloc[0, 1])
```

Step 9. query()로 조건을 문자열로

해야 할 것

- 조건이 길면 가독성 개선
- 컬럼명에 공백/특수문자 있으면 불편
- 팀에서는 "읽기 쉬운 방식" 선택

체크

80 이상 90 미만인 b 행만 출력된다.

실행 코드

```
import pandas as pd

df = pd.DataFrame({'name':['a','b','c'],
'score':[70,85,90]})

print(df.query('score >= 80 and score < 90'))
```

문자열 필터링: str 메서드

```
df[df['name'].str.startswith('k')]
df[df['name'].str.contains('im',
na=False)]
```

핵심 포인트

na=False로 결측치 안전 처리
전처리(소문자 통일) 후 필터하면 정확도↑
AI에게: 정규표현식(regex) 사용 여부도 물어보기

Step 10. SettingWithCopyWarning 피하기

해야 할 것

- 필터 결과는 view/복사본이 모호할 수 있음
- .copy()로 명확히 복사
- 수정은 loc로 하는 습관

체크

경고 없이 level 컬럼이 추가되고 b,c 행에 high가 들어간다.

실행 코드

```
import pandas as pd

df = pd.DataFrame({'name':['a','b','c'],
'score':[70,85,90]})

sub = df.loc[df['score'] >= 80].copy()
sub['level'] = 'high'
print(sub)
```

성능/가독성 팁

```
# 조건은 먼저 작게 만들어 확인
# 필요한 컬럼만 선택해서 처리
# 정렬은 최소 횟수로
```

핵심 포인트

정답보다 '재현 가능하고 읽기 쉬운 분석'이 중요
중간 결과를 head()/shape로 확인하는 습관
AI에게: "내 코드가 더 간결해질 수 있는지" 리뷰 요청 가능

미니 예제: 랭킹 만들기(스켈레톤)

```
# 요구: 조건(예: region=='A')으로 필터 후
# score 기준 내림차순 정렬, Top 10 출력
# TODO: filter → sort_values → head
```

핵심 포인트

필터링과 정렬은 항상 '세트'로 등장
출력 전에 df.shape로 행 개수 확인
With AI: '분석 결과 요약 문장'을 자동 생성하게 해보자

With AI: EDA 프롬프트 템플릿

역할: 너는 데이터 분석가다.
데이터: df.head(), df.info(), df.describe() 결과 제공
목표: (예: 상위 고객 찾기, 성적 상위 10명 뽑기)
요청: (1) 필요한 필터 조건 제안 (2) Pandas 코드 (3) 결과 해석
추가: 데이터 품질 체크(결측/이상치)도 함께 해줘.

* EDA: Exploratory Data Analysis (탐색적 데이터 분석)

실습 1: 조건 필터 + Top-N

문제: 데이터에서 조건을 2개 이상 사용해 필터링한 뒤 Top 5를 출력하라.
예: (score>=80) AND (region in ['A','B'])
제출: 코드 + 결과 해석 2문장

실습 2: 디버깅 — 괄호/연산자 실수

```
# 잘못된 예(에러 또는 이상한 결과)
df[df['score'] >= 80 and df['score'] < 90]
```

핵심 포인트

Pandas에서는 and/or 대신 &/| 사용
올바른 예: df[(cond1) & (cond2)]
AI에게: 왜 and가 안 되는지(Series의 True/False) 설명 요청

실습 3: 리팩터링 — 필터 함수 만들기

```
def filter_top(df, cond, sort_col, n=5):
    filtered = df[cond]  # 조건으로 필터링
    sorted_df = filtered.sort_values(by=sort_col,
ascending=False)  # 정렬
    return sorted_df.head(n)  # 상위 n개 반환
```

핵심 포인트

분석 코드를 함수로 만들면 재사용성이 좋아진다.
AI에게: 함수 인터페이스(입력/출력)를 먼저 설계하게 하자.
검증: 다른 조건/컬럼에도 동작하는지 테스트

연습문제(필터링·정렬)

1) score가 평균 이상인 행만 선택하라.
2) 특정 카테고리만 선택(isin)하고, score로 내림차순 정렬하라.
3) 문자열 컬럼에서 특정 키워드가 포함된 행을 찾고(contains) 개수를 출력하라.
4) Top 3와 Bottom 3를 각각 구하라(nlargest/nsmallest).

미니 퀴즈(5문항)

문항
1) Pandas에서 복합 조건 결합에 쓰는 연산자는?
A. and/or B. &/| C. +/*
2) df.sort_values('col', ascending=False)는?
A. 오름차순 B. 내림차순 C. 랜덤
3) 상위 5개를 뽑는 대표 함수는?
A. head(5) B. tail(5) C. sum(5)
4) df['name'].str.contains('a')의 용도는?
A. 숫자 합 B. 문자열 포함 필터 C. 정렬
5) SettingWithCopyWarning의 의미는?
A. 메모리 부족 B. 뷰/복사본 수정 모호 C. 파일 없음