Data Analysis

3차시
데이터 정제하기

20 . .

지은이: 이은재

Ⅰ. 데이터 무결성과 이상현상

- 데이터 무결성이란?

Data Integrity: 데이터가 정확하고, 일관되며, 신뢰할 수 있도록 유지하는 데에 필요한 기본 원칙

→ 무결하지 않은 데이터를 분석하는 것은 무의미하다!

주요 개념	특징
완전성	데이터가 빠진 부분 없이 충분히 존재하는가?
일관성	데이터 간 서로 모순이 없는가?
정확성	데이터가 실제 값을 제대로 반영하고 있는가?
신뢰성	데이터가 분석할 때마다 일관된 결과를 제공하는가?
원본성	데이터가 원본 또는 진정한 사본인가?

- 데이터 이상현상이란?

Data Anomaly: 정상적인 패턴에서 벗어나 데이터의 신뢰성과 분석 결과에 영향을 줄 수 있는 비정상적인 데이터 상태

- 데이터 이상현상의 유형

유형	예시	원인	해결방법
이상값	100세 미만 데이터에서 500세가 나옴	입력 오류, 센서 오류, 극단적인 데이터	IQR, 표준편차, 머신러닝 기반 탐지
결측치	나이 컬럼에 NULL 값이 많음	데이터 누락, 수집 과정 문제	삭제, 평균/중앙값 대체, 머신러닝 대체
다중값 속성	이름 컬럼에 두 개의 이름이 기록되어있음	데이터 모델링 오류, 수집 과정 문제	삭제, 데이터 분리
중복 데이터	동일한 사용자가 여러 번 저장됨	중복 저장, 시스템 오류	고유 식별자 활용, 유사 데이터 매칭
오류 데이터	온라인 주문의 출고일이 주문일보다 빠름	입력 오류, 단위 변환 문제	비즈니스 룰 적용, 데이터 정제
데이터 드리프트	고객 소비 패턴이 갑자기 변함	시장 변화, 데이터 패턴 변화	주기적 모니터링, 모델 재학습

Q. 해결할 수 없는 이상현상도 있을까?

Ⅱ. 데이터 정제

- 데이터 정제란?

Data Cleansing: 데이터 품질을 높이기 위하여 원시 데이터에서 오류, 결측치, 중복, 이상치 등을 찾아 수정하거나 제거해 불완전한 데이터를 정리하는 작업

- 데이터 이상현상 해결의 5대 원칙

1. 원본 데이터는 절대 변경하지 않는다

- 원본 데이터를 직접 수정하면 추후 오류를 추적하거나 복구하기 어려움
- 해결 방법:
    - 원본 데이터는 유지하고 정제된 데이터 사본을 생성
    - 원본과 정제된 데이터 간 버전 관리 (예: raw_data, cleaned_data 분리)

2. 이상현상을 수정하기 전에 반드시 원인을 분석한다

- 단순히 "이상값이니까 제거하자"가 아니라, 이 값이 왜 이상한지 이해해야 함
- 해결 방법:
    - 데이터 수집 과정 검토 (센서 오류? 입력 실수?)
    - 도메인 전문가 의견 반영
    - 시각화(박스플롯, 히스토그램 활용)로 이상값 확인

3. 이상현상 해결 방법은 재현 가능해야 한다

- "어떻게 데이터를 정제했는지"가 명확히 기록되어야 함
- 해결 방법:
    - 데이터 정제 코드와 로그 남기기 (예: Python Pandas 활용)
    - 문서화 (어떤 데이터를 제거했는지, 어떤 기준으로 대체했는지, 왜 그런 결정을 내렸는지)

4. 데이터 정제는 분석 목적에 맞게 수행한다

- 어떤 분석을 하느냐에 따라 이상값 처리가 달라질 수 있음
- 해결 방법:
    - 예측 모델링: 이상값을 제거하거나 변환 (예: log 변환)
    - 탐색적 분석(EDA): 이상값도 의미 있는 데이터일 수 있으므로 보존
    - 이상 탐지(Anomaly Detection): 이상값을 주요 분석 대상으로 삼음

5. 이상값을 제거하는 대신 대체하는 방법도 고려한다

- 무조건 결측치를 삭제하면 데이터 손실이 커질 수 있음

- 해결 방법:
    - 삭제 (이상값이 소수이고 분석에 큰 영향이 없을 때)
    - 대체 (평균, 중앙값, 머신러닝 예측값으로 채우기)
    - 도메인 지식 활용 (비즈니스 규칙에 맞게 보정)

Ⅲ. Pandas DataFrame에서 데이터 정제하기

1. 특정 열에 접근하기

특정 열에 접근하기

```
# 데이터프레임의 전체 열 이름을 출력합니다.
print(df.columns)

# 지정된 이름을 가진 열을 삭제합니다. (axis=1, 열 / axis=0, 행)
# inplace=True는 원본 데이터 프레임 직접 수정을 의미합니다.
df.drop(['열이름', '열이름2'], axis=1, inplace=True)

# NaN 값이 포함된 열을 모두 삭제합니다.
df.dropna(axis=1)
# 모든 값이 NaN인 열만 삭제합니다.
df.dropna(axis=1, how='all')
```

2. 특정 행에 접근하기

특정 행에 접근하기

```
# 특정 행(인덱스 0, 1)을 삭제합니다.
df.drop([0, 1])

# 인덱스 2번 이후의 데이터만 선택합니다.
df[2:]

# 인덱스 0~1번 데이터만 선택합니다.
df[0:2]

# '출판사'가 '한빛미디어'인 행을 선택합니다.
selected_rows = df['출판사'] =='한빛미디어'

# loc을 사용하여 조건을 만족하는 행을 선택합니다.
df.loc[selected_rows]

# '대출건수'가 1000건 이상인 행을 선택합니다.
ns_book2 = ns_book[ns_book['대출건수'] >1000]
```

3. 중복 행 찾기

중복 행 찾기

```
# 중복된 전체 행 개수를 출력합니다.
sum(df.duplicated())

# 특정 열 기준으로 중복된 행 개수를 출력합니다.
sum(df.duplicated(subset=['기준열']))

# 도서명, 저자, ISBN, 권을 기준으로 대출건수를 그룹화하여 합산합니다.
group_df = df.groupby(by=['도서명', '저자', 'ISBN', '권'], dropna=False)
loan_count = group_df.sum()
```

4. 정규표현식

정규표현식

```
# 특정 값 변환 (예: '2021' → '21')
ns_book4.replace({'발행년도': {'2021': '21'}})[100:102]

# 정규 표현식으로 연도 변환 (네 자리 연도에서 뒤 두 자리만 남김)
ns_book4.replace({'발행년도': {r'\d\d(\d\d)': r'\1'}}, regex=True)[100:102]

# 정규 표현식 패턴을 다르게 적용 (더 일반적인 형태)
ns_book4.replace({'발행년도': {r'\d{2}(\d{2})': r'\1'}}, regex=True)[100:102]

# '저자'와 '발행년도' 데이터 정리
ns_book4.replace({
    '저자': {r'(.*)\s\(지은이\)(.*)\s\(옮긴이\)': r'\1\2'},
    '발행년도': {r'\d{2}(\d{2})': r'\1'}
}, regex=True)[100:102]
```