Python
CH10: Pandas I

전종준 교수

서울시립대학교 통계학과/통계데이터사이언스대학원

Contents

- Pandas 의 소개

- DataFrame 다루기

- 결측자료의 처리

Pandas 의 소개

강의 목표

- Pandas의 핵심 개념과 자료구조 이해
- Series와 DataFrame을 직접 생성하고 조작해보기
- 다양한 파일 포맷(CSV, Excel, JSON) 입출력 실습

Pandas란 무엇인가?

- 표 형식 데이터를 다루기 위한 파이썬 라이브러리
- 구조화된 데이터를 빠르고 직관적으로 조작 가능
- 데이터 분석, 머신러닝, 통계 처리의 기반 도구
- NumPy 기반으로 설계되어 배열과도 호환됨

Pandas의 핵심 자료구조

- **Series**: 1차원 데이터 구조, 레이블된 배열
- **DataFrame**: 2차원 테이블 구조, 여러 Series의 집합
- 인덱스(Index)를 통한 강력한 데이터 정렬/선택 기능 제공
- 데이터 전처리, 통계, 시각화 등 다양한 작업의 중심

Series 생성 예시

```
import pandas as pd
s = pd.Series([1000, 2000, 3000], index=["1월", "2월", "3월"])
print(s)
```

- 리스트와 인덱스를 인자로 받아 생성
- 인덱스를 통해 값에 쉽게 접근 가능

Series의 주요 기능

- 인덱싱: 레이블 또는 정수 위치를 사용해 값 선택 가능
- 슬라이싱: 연속된 값 추출 (예: `s["1월":"2월"]`)
- 연산: 브로드캐스팅 및 벡터화된 계산 지원
- 결측치 처리: `isnull()`, `fillna()` 등 사용 가능
- 통계 함수: `sum()`, `mean()`, `max()`, `min()` 등

Series의 주요 기능

```
s = pd.Series([1000, 2000, 3000], index=["1월", "2월", "3월"])
# 인덱싱
print(s["2월"])     # 2000
# 슬라이싱
print(s["1월":"2월"])  # "1월"과 "2월" 포함
# 연산
print(s * 1.1)
# 통계 함수
print(s.mean())
print(s.sum())
```

- 인덱스를 사용한 접근과 슬라이싱이 가능함
- 연산 시 각 요소에 자동으로 적용됨 (브로드캐스팅)
- `mean()`, `sum()`, `min()`, `max()` 등 통계 함수 지원

DataFrame 생성 예시

```
data = {
    "이름": ["영희", "철수", "민수"],
    "국어": [90, 85, 82],
    "수학": [88, 92, 79]
}
df = pd.DataFrame(data)
print(df)
```

- 딕셔너리를 사용해 열 단위로 생성
- 컬럼 이름과 인덱스를 지정할 수 있음

DataFrame의 주요 기능

```
# 열 선택
print(df["국어"])
# 행 선택 (loc, iloc)
print(df.loc[0])
print(df.iloc[1])
# 열 추가
df["총점"] = df["국어"] + df["수학"]
# 조건 필터링
print(df[df["국어"] >= 85])
# 통계 함수
print(df.mean(numeric_only=True))
```

- 열/행 선택: 열 이름, loc/iloc을 활용한 접근
- 열 추가 및 수식 연산: 새로운 열 계산 가능
- 조건 필터링: 특정 조건 만족하는 행만 추출
- 통계 요약: 평균, 합계 등 기본 분석 가능

NumPy와 Pandas 비교

특징	NumPy	Pandas
구조	다차원 배열	표 형식 (행+열)
인덱싱	정수 인덱스	라벨 인덱스 가능
결측치 처리	불편	용이 (NaN 지원)
사용 목적	수치 계산 중심	데이터 분석 중심

NumPy 배열을 DataFrame으로 변환

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
df = pd.DataFrame(arr, columns=["A", "B", "C"])
print(df)
```

데이터 입출력 개요

- Pandas는 다양한 외부 데이터 파일을 쉽게 읽고 쓸 수 있음
- 주요 지원 포맷: CSV, Excel, JSON, SQL, HTML 등
- 읽기 함수: `read_csv()`, `read_excel()`, `read_json()`
- 저장 함수: `to_csv()`, `to_excel()`, `to_json()`

CSV 파일 불러오기와 저장

```
df = pd.read_csv("students.csv")
print(df.head())

df.to_csv("students_saved.csv", index=False)
```

- head(): 상위 5개 행 미리보기
- index=False: 인덱스 열 저장 생략

Excel과 JSON 파일 읽기

```
df_excel = pd.read_excel("data.xlsx", sheet_name="Sheet1")
df_json = pd.read_json("data.json")
```

- Excel 파일은 sheet 이름을 명시할 수 있음
- JSON 파일은 딕셔너리 형식의 데이터를 자동 파싱

DataFrame 다루기

DataFrame 기본 조작: 개요

- 행과 열 선택: loc, iloc
- 구조 탐색 및 요약 통계: head, info, describe, shape
- 열 추가/삭제 및 정렬

열 선택

```
print(df["국어"])
print(df[["국어", "수학"]])
```

- 단일 열: 문자열로 지정, DataFrame에서 열을 먼저 선택하면 Series가 반환됨
- 여러 열: 리스트로 묶어서 지정, 이 경우 DataFrame이 반환됨

행 선택: loc와 iloc

```
print(df.loc[[True, False, False]])    # 첫 번째 행
print(df.loc[0])    # 첫 번째 행 (라벨 기준)
print(df.iloc[1])   # 두 번째 행 (정수 위치 기준)
```

- bool list: True 또는 False 값을 가지는 리스트를 이용해서 필터링
- loc: 인덱스 라벨 (index key) 기준으로 행을 선택
  - DataFrame 생성 시 index를 지정하지 않으면 0부터 시작하는 정수형 인덱스가 자동으로 부여됨
  - 이 경우 `loc[0]`과 `iloc[0]`은 동일한 첫 번째 행을 반환함
  - 인덱스를 수동으로 문자 등으로 지정하면 `loc[0]`은 오류 발생 가능
- iloc: 정수 인덱스 (순서)기준 선택함. index key가 할당된 상태에서도 순서로 선택가능

개별 원소에 접근

```
data = {
    "이름": ["영희", "철수", "민수", "지수"],
    "국어": [90, 85, 78, 92],
    "수학": [88, 91, 84, 95]
}
df = pd.DataFrame(data, index=["a", "b", "c", "d"])
print(df)

# 개별 원소 접근
print(df.loc["b", "수학"])   # 라벨 기준 접근
print(df.iloc[1, 2])         # 정수 위치 기준 접근
```

- `df.loc[행라벨, 열라벨]`: 인덱스와 컬럼명을 이용한 접근
- `df.iloc[행번호, 열번호]`: 위치 기반 접근

열을 먼저 선택한 후 원소 접근

```
print(df["수학"][0])      # 방법 1: 열 선택 후 행 번호
print(df["수학"].loc[0])  # 방법 2: Series에서 라벨 접근
print(df["수학"].iloc[1]) # 방법 3: Series에서 위치 접근
```

- DataFrame에서 열을 먼저 선택하면 Series가 반환됨
- 이후 Series에서 라벨 또는 위치를 사용해 원소 접근 가능
- `df["열이름"][행번호]`는 직관적이지만 경고가 발생할 수 있음

조건 필터링

```
print(df[df["국어"] >= 85])
```

- 조건식을 통해 특정 행만 선택 가능
- Boolean indexing 기반: True값에 대응되는 원소만 선택됨
- 국어가 [90, 95, 78, 92] 이므로 df[”국어”]>=85 는 [True, True, Fasle, True] 를 반환
- df[[True, True, Fasle, True]] 는 행을 선택하는 것으로 True 에 대응되는 행만 추출됨

read_csv()에서 인덱스 지정하기

```
# 예시 CSV: students.csv
# 이름,국어,수학
# 영희,90,88
# 철수,85,91
# 민수,78,84

# 기본: 자동 정수 인덱스
df1 = pd.read_csv("students.csv")

# '이름' 열을 인덱스로 지정
df2 = pd.read_csv("students.csv", index_col="이름")
```

- 기본적으로는 0부터 시작하는 정수 인덱스가 자동 지정됨
- `index_col` 매개변수로 특정 열을 인덱스로 지정할 수 있음
- 문자열 인덱스를 지정하면 `loc["영희"]`처럼 이름으로 접근 가능

head(), tail(), shape

```
print(df.head())      # 상위 5개 행
print(df.tail(2))     # 하위 2개 행
print(df.shape)       # (행, 열)
```

- 데이터의 크기와 일부 미리보기 가능

info(), describe()

```
print(df.info())        # 데이터 타입, 결측치 등
print(df.describe())    # 수치형 요약 통계
```

- 데이터 구조와 통계적 개요 파악

열 추가

```
df["총점"] = df["국어"] + df["수학"]
print(df)
```

- 연산을 통해 새로운 열 생성 가능

열 삭제

```
df = df.drop("총점", axis=1)
```

- `drop()` 함수로 열 또는 행 삭제 가능
- `axis=1`은 열, `axis=0`은 행

sort_values()로 정렬하기

```
df = pd.DataFrame({
    "이름": ["영희", "철수", "민수"],
    "국어": [90, 85, 78],
    "수학": [88, 91, 84]
})

# 국어 점수를 기준으로 오름차순 정렬
df_sorted = df.sort_values("국어")

# 수학 점수를 기준으로 내림차순 정렬
df_sorted_desc = df.sort_values("수학", ascending=False)
```

- `sort_values(by=...)`: 특정 열을 기준으로 행을 정렬
- `ascending=False`: 내림차순 정렬
- 원본은 변경되지 않으며, 결과는 새로운 DataFrame으로 반환됨

여러 기준으로 정렬하기 (double sort)

```
df = pd.DataFrame({
    "이름": ["영희", "철수", "민수", "지수"],
    "국어": [90, 85, 85, 92],
    "수학": [88, 91, 84, 95]
})

# 국어 기준 오름차순, 국어가 같을 경우 수학 기준 내림차순
sorted_df = df.sort_values(by=["국어", "수학"], ascending=[True, False])
print(sorted_df)
```

- by 매개변수에 리스트 형태로 여러 열 지정 가능
- ascending도 각 열에 대해 리스트로 지정
- 첫 번째 기준이 동일할 경우 두 번째 기준으로 정렬

데이터 정렬: sort_index()

```
print(df.sort_index())
```

- 인덱스를 기준으로 정렬
- 문자열/숫자 모두 적용 가능

실습 과제

- 열 선택: 국어, 수학 열만 선택해 출력해보세요
- 조건 필터링: 수학 점수가 90점 이상인 학생 찾기
- 열 추가: 평균 점수 열을 추가해보세요
- 데이터 정렬: 국어 점수 기준으로 내림차순 정렬

결측자료의 처리

결측치와 데이터 정제 (I): 개요

- 결측값(NaN)을 탐지하고 처리하는 방법 학습
- 데이터 타입 변환 및 정제 방법 실습

read_csv()에서 결측치 처리

```
import pandas as pd
# 예시 CSV 내용:
# 이름,국어,수학
# 영희,90,
# 철수,,88
# 민수,85,84

# 빈 셀은 자동으로 NaN 처리됨
df = pd.read_csv("students.csv")

# 특정 문자열을 NaN으로 처리 (예: '-')
df2 = pd.read_csv("students.csv", na_values=["-"])
```

- 기본적으로 빈 셀은 자동으로 NaN으로 인식됨
- `na_values` 옵션을 사용해 NaN으로 처리할 값을 지정할 수 있음
- 이후 `isnull()`, `fillna()` 등으로 결측치 처리 가능

결측치 확인: isnull(), notnull()

```
import pandas as pd
import numpy as np

df = pd.DataFrame({
    "이름": ["영희", "철수", "민수"],
    "수학": [90, np.nan, 85],
    "영어": [np.nan, 88, 92]
})

print(df.isnull())
print(df.notnull())
```

- `np.nan` 은 numpy 에서 정의한 특수한 값
  - `np.nan`은 숫자가 없음을 나타내는 특수한 부동소수점 값
  - `np.nan == np.nan`은 False이므로 비교 대신 `np.isnan()` 사용
  - 연산에 포함되면 결과가 NaN이 되는 특성 있음
- `isnull()`: NaN 여부를 True/False로 반환
- `notnull()`: NaN이 아닌 값에 True 반환

결측치 제거: dropna()

```
print(df.dropna())              # NaN이 포함된 행 제거
print(df.dropna(axis=1))        # NaN이 포함된 열 제거
```

- 기본값: 행 기준 제거 (axis=0)
- 열을 기준으로 제거하고 싶다면 axis=1 지정

결측치 대체: fillna()

```
print(df.fillna(0))           # NaN을 0으로 대체
print(df.fillna(method="ffill"))  # 이전 값으로 채우기
```

- 수치 대체 또는 보간 기법으로 결측값 채움
- `method="ffill"`: forward fill, 이전 값 사용

열의 평균으로 결측치 대체

```
# 각 열의 평균값으로 NaN 대체
df_filled = df.fillna(df.mean(numeric_only=True))
print(df_filled)
```

- `df.mean()`을 이용해 각 열의 평균값 계산
- `fillna()`에 전달하면 각 열별로 NaN을 평균값으로 대체함
- 수치형 열에만 적용되도록 `numeric_only=True` 설정 가능

데이터 타입 확인 및 변환

```
print(df.dtypes)

df["수학"] = df["수학"].astype("float")
```

- `dtypes` 속성으로 각 열의 타입 확인
- `astype()`으로 수동 타입 변환 가능

실습 과제

- NaN이 있는 데이터를 만들어서 isnull()로 확인하기
- dropna(), fillna()를 각각 적용해 보기
- astype()을 이용해 열 타입 변경