Data Science

Python Data Processing Basics

홍익대학교 소프트웨어융합학과

노기섭 교수

(kafa46@hongik.ac.kr)

Lecture Goals

- Python으로 데이터 불러오기·가공·요약 능력 습득
- NumPy로 배열 연산, 브로드캐스팅, 선형대수 처리
- Pandas로 데이터프레임 생성, 인덱싱, 결측치 처리, 그룹 집계
- 예제를 통해 단계별 실습 능력 강화
- Toy Project

Python 데이터 처리 개요

데이터 처리와 Python

Python은 데이터과학에서 사실상 표준 도구

- 배우기 쉬운 문법, 풍부한 라이브러리, 방대한 커뮤니티
- 데이터 수집에서 분석 시각화 모델링 배포까지 전 과정 지원

Numpy, Pandas 비교

NumPy

- 고성능 배열 자료구조와 벡터화 연산을 제공하는 수치 계산 라이브러리

Pandas

- 표 형태 데이터 처리를 위한 고수준 API를 제공하는 분석 라이브러리

NumPy 고성능 수치 연산 배열 라이브러리	VS	**Pandas** 데이터프레임 기반 데이터 조작 라이브러리
• 다차원 균일 배열(ndarray) 자료구조 • 빠른 벡터화 연산으로 성능 최적화 • 선형대수 함수 및 수치 계산 지원		• 표 형태의 데이터(DataFrame) 구조 • 레이블 기반 강력한 인덱싱 기능 • 결측치 처리 및 그룹화/집계 기능

Numpy Basics

Numpy Overview

NumPy 소개

- 고성능 다차원 배열(ndarray) 제공
- 벡터화 연산으로 순수 Python 루프 대비 큰 속도 향상
- 선형대수, 푸리에 변환, 난수 생성 등 수치 계산 도구 제공

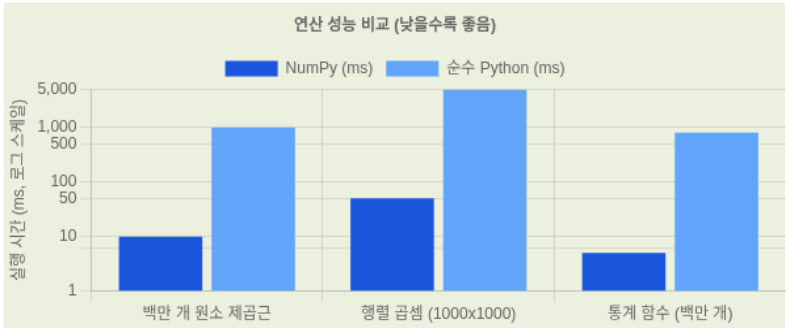
NumPy 활용 분야

- 이미지·신호 처리 행렬 연산
- 대규모 로그·측정 데이터 통계 처리
- 머신러닝 전처리 및 피처 엔지니어링

왜 NumPy를 학습해야 하는가?

- 대부분의 데이터 과학·머신러닝 라이브러리의 기반
- 브로드캐스팅으로 코드 간결화·성능 향상

- 이미지 처리를 위한 행렬 연산이 **벡터화**되어 빠른 처리 가능
- 필터 적용, 이미지 회전, 크기 변경 등 복잡한 연산을 **간 결한 코드**로 구현
- 컴퓨터 비전, 자율주행, 의료영상 등에 응용되는 핵심 기 술



Numpy 설치

```
# 설치
$ pip install numpy
```

기초 사용법

```
import numpy as np  # 수치 계산을 위한 핵심 라이브러리 로드

a = np.array([1, 2, 3])   # 1차원 배열 생성
b = np.zeros((2, 3))      # 2x3 크기의 0으로 채워진 배열
c = np.arange(0, 10, 2)   # 0부터 10 미만까지 2 간격으로 생성
d = np.linspace(0, 1, 5)  # 0부터 1까지 5개 균등 분할 값
```

실행 결과 설명

- shape는 행과 열의 개수를 의미
- Indexing과 Slicing으로 원하는 위치의 값을 선택
- mean과 std로 요약 통계 확인

Numpy 초급 예제

시나리오

- 스마트 홈 센서에서 하루 동안 수집된 기온 데이터에서 이상치를 제거하고 평균을 구하라

numpy.mean(a, axis=None, dtype=None, keepdims=False)
- 입력: 배열 a, 집계 축 axis 선택 가능
- 출력: 지정 축의 평균 스칼라 또는 배열
- 기능: 원소들의 산술평균 계산
- 브로드캐스팅: clean * 9/5 + 32는 스칼라 연산을 모든 원소에 적용하는 벡터화 연산

```
import numpy as np

# 원시 온도 데이터(℃) 로드
temps_c = np.array([14.2, 15.8, 99.0, 16.0, 17.1, -50.0, 18.3, 19.0])

# 물리적으로 불가능한 온도 범위를 마스킹하여 제거
mask = (temps_c >= -20) & (temps_c <= 50)  # 정상 범위 조건
clean = temps_c[mask]                      # 조건을 만족하는 값만 추출

# 섭씨 평균과 화씨 평균 계산
mean_c = clean.mean()
fahrenheit = clean * 9/5 + 32   # ℃ → ℉ 변환
mean_f = fahrenheit.mean()

print(mean_c, mean_f)  # 결과 출력 16.12 60.99
```

Numpy 중급 예제

시나리오

- 웹 서비스 로그 데이터에서 사용자 세션 별 클릭 수, 구매 수, 체류 시간을 열로 정리한다.
- 각 열을 평균 0으로 중심화(mean-centering)한다.

중심화(mean-centering)란 무엇인가?

- 각 특성(열)에서 해당 열의 평균을 빼서 평균이 0이 되도록 변환하는 과정이다
- 수식: 변환 전 행렬이 $X \in \mathbb{R}^{n \times p}$, 열 평균 $\mu \in \mathbb{R}^{p}$일 때,
- 중심화 결과는 $X_c = X - 1 \cdot \mu^T$ (여기서 1은 길이 n의 1벡터)
  - 1벡터: $1 \in \mathbb{R}^{n \times 1}$, 따라서 $1 \cdot \mu^T \in \mathbb{R}^{n \times p}$
  - 각 행마다 동일하게 평균 벡터 μ가 복제된 행렬
- 목적: 스케일 차이가 큰 열들 간에 평균 위치를 동일한 기준(0)으로 맞춰 비교·학습 안정성 향상
- 활용: 회귀, 주성분분석(PCA), 표준화(Z-score) 전 단계에서 매우 빈번하게 사용

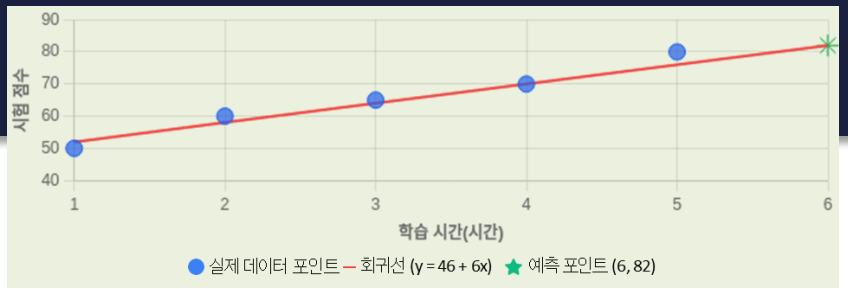
Numpy 중급 예제 (실습 코드)

```python
import numpy as np

# 열이 서로 다른 규모를 가질 때 공정 비교를 위해 평균 중심화 적용
data = np.array([[120, 12, 35],
                 [ 80,  9, 40],
                 [200, 20, 30]])

col_means = data.mean(axis=0)   # 열 평균 계산
centered = data - col_means     # 각 값에서 해당 열 평균을 빼서 중심화

print("열 평균:", col_means)
print("중심화 결과:\n", centered)
```

실행 결과

```
열 평균: [130. 13.2 40. ]
중심화 결과:
 [[-10. -1.2 -5. ]
  [-50. -4.2 0. ]
  [ 70. 6.8 -10. ]]
```

Numpy 고급 예제



시나리오

- 어떤 학급에서 학습 시간(X)과 시험 점수(y)를 기록했다.
- 선형회귀로 학습시간 6시간일 때 점수를 예측하라

메서드 사용법: 다음 슬라이드 참조

```python
import numpy as np

# 학습 시간과 점수의 관계를 최소제곱으로 적합
X = np.array([1, 2, 3, 4, 5])        # 독립변수
y = np.array([50, 60, 65, 70, 80])   # 종속변수

A = np.vstack([np.ones_like(X), X]).T   # 설계행렬 [1, x]
coeffs, residuals, rank, s = np.linalg.lstsq(A, y, rcond=None)  # 최소제곱 해
b0, b1 = coeffs  # 절편과 기울기

y_pred_6 = b0 + b1 * 6  # x=6에서의 예측값
print("b0:", b0, "b1:", b1, "pred@6:", y_pred_6)
```

실행 결과

```
b0: 46.0 b1: 6.0 pred@6: 82.0
```

회귀식은 y = 46 + 6x로 근사

학습시간 6시간일 때 예측 점수는 82로 계산

메서드 사용법

numpy.vstack(tup)

- 입력: 같은 열 길이를 가진 1D/2D 배열들의 시퀀스
- 출력: 배열들을 위로 쌓아 행 방향으로 결합한 2D 배열
- 기능: 설계행렬 구성 시 상수항(1)과 독립변수 x를 한 배열로 결합할 때 사용

numpy.linalg.lstsq(a, b, rcond=None)

- $\min_x \|ax - b\|_2$의 해를 구함
- 공식 문서: https://numpy.org/doc/2.2/reference/generated/numpy.linalg.lstsq.html
- 입력: 계수행렬 a, 타깃벡터 b, 자르기 파라미터 rcond
- 출력: (x, residuals, rank, s) 튜플 반환
  - x: 최소제곱 해(회귀계수)
  - residuals: 잔차 제곱합
  - rank: 행렬의 랭크
  - s: 특이값 배열

[참고] 행렬의 Rank

[참고] 행렬의 랭크 (Rank)

- 변수의 독립적 개수 → 어떤 데이터를 나타내는 데 필요한 최소한의 차원 수
- 행렬의 선형 독립한 행(row) 또는 열(column)의 최대 개수를 의미한다.
- 즉, 행렬을 행 공간(Row space)이나 열 공간(Column space)으로 봤을 때, 그 공간을 생성하는 기저 벡터의 개수
- Data Science 또는 Machine Learning에서 데이터 행렬의 랭크는 피처(feature)들이 서로 얼마나 독립적인지를 나타냄
- 보통 PCA 같은 차원 축소 기법은 사실상 랭크를 고려해 데이터의 본질적 차원을 찾는 과정

Pandas Basics

Pandas 소개

- **Pandas 소개**
  - 표 형태 데이터의 로딩·정제·변환·요약·집계를 위한 고수준 API 제공
  - CSV, Excel, SQL, Parquet 등 다양한 포맷 입출력 지원
- **언제 Pandas를 사용하는가**
  - CSV·Excel·데이터베이스에서 읽은 표 형태 데이터를 정리·가공할 때 사용
  - 결측치 처리, 형변환, 날짜 처리 등 전처리 루틴을 빠르게 구성할 때 사용
  - 그룹 요약, 피벗, 멀티인덱스 등 보고서용 요약표가 필요할 때 사용
  - 시계열 리샘플링, 롤링 통계 등 시간 데이터 분석이 필요할 때 사용
- 장점
  - 일관된 고수준 API 제공으로 빠른 개발 가능
  - 풍부한 입출력 지원과 시각화 연계 용이
  - 멀티인덱스·피벗 등 강력한 요약 기능 제공
- 단점
  - 메모리 기반 실행 특성으로 초대용량 데이터 처리 한계 존재
  - 연산 체이닝이 길어지면 가독성 저하 가능

다른 도구와의 차이점

Excel과의 차이점

- Excel은 GUI 중심, Pandas는 코드 중심
- Excel은 클릭·드래그로 빠른 처리, Pandas는 재현 가능한 스크립트로 자동화·버전 관리 용이
- Excel은 수동 작업 오류 위험, Pandas는 테스트·리뷰로 품질 관리 유리

	Excel	Pandas
인터페이스 방식	**GUI 기반** 클릭과 드래그로 직관적 조작	**코드 기반** Python 스크립트로 명령 실행
재현성 (Reproducibility)	수동 작업 과정 기록 어려움 단계별 문서화 필요	**코드로 모든 분석 과정 기록** 동일한 결과 재현 용이
자동화 & 배치 처리	VBA 매크로로 제한적 자동화 대규모 반복 작업에 비효율적	**Python 스크립트로 완전 자동화** 대량 파일/데이터 일괄 처리 가능
버전 관리	파일 자체 비교 어려움 여러 버전 관리 복잡	**Git 등 버전 관리 도구와 연동** 코드 변경사항 추적 및 협업 용이

다른 도구와의 차이점

R과의 차이점

- R은 통계·시각화 문법이 풍부, Pandas는 Python 생태계와의 연계가 강점
- R은 파이프 문법으로 선언적 데이터 처리, Pandas는 메서드 체이닝으로 패턴 구현
- Pandas는 머신러닝·배포 측면에서 Python 생태계(scikit-learn, FastAPI 등)와 연동이 쉬움

R	Python + Pandas
통계 및 시각화 특화	범용성과 확장성
통계 분석 중심 언어로 설계	범용 프로그래밍 언어 기반
ggplot2로 선언적 그래프 생성 통계 모델링 함수 풍부	matplotlib, seaborn 시각화 다양 일반 개발 지식 활용 가능

Pandas와 R의 설계 철학

	Pandas	R
출발점	2008년, Wes McKinney가 금융 데이터 분석용으로 개발	1990년대 초반, Ihaka & Gentleman이 통계 계산용으로 개발
철학적 지향	Pythonic, 범용 데이터 처리 · 분석 라이브러리	데이터 분석 전용 언어 (Domain-Specific)
핵심 개념	DataFrame 기반, 메서드 체이닝 중심	벡터화, 데이터 중심 함수형 설계
강점	- Python 생태계와 호환성 (NumPy, scikit-learn, TensorFlow 등) - 실무 · 엔지니어링 · 배포까지 확장 쉬움	- 풍부한 통계/시각화 패키지 내장 - 선언적 데이터 처리 (dplyr, ggplot2)
사용 맥락	데이터 전처리 → 머신러닝 → API 배포까지 이어지는 실무 파이프라인	통계 모델링, 학술 연구, 데이터 탐색적 분석에 최적
철학적 요약	"범용 프로그래밍 언어 속 데이터프레임 도구"	"통계와 분석을 위한 연구자 중심 언어"

대학·교육 현장에서의 경향

Python 표준화

- 전 세계적으로 데이터사이언스·머신러닝 교육의 기본 언어가 Python으로 거의 굳어짐
- Coursera, edX, Kaggle, Google Colab, fast.ai 등 대부분의 플랫폼이 Pandas 중심

R의 위치

- R은 여전히 통계학과, 생물정보학, 사회과학 쪽에서는 강력하게 사용
- 범용적인 데이터사이언스 커리큘럼(특히 컴퓨터공학/소프트웨어융합/AI 관련 학과)에서는
  - Python + Pandas가 사실상 표준

산업 연계성

- 기업에서 데이터 처리 → 머신러닝/딥러닝 → 배포까지 워크플로우는 대부분 Python 기반
- Pandas가 NumPy, scikit-learn, PyTorch, TensorFlow와 이어지므로 실무 연결성이 강력

언제 pandas를 써야 할까?

표 형태 데이터 처리

CSV, Excel, SQL 등에서 가져온
구조화된 데이터를 효율적으로
정리하고 가공할 때

```
df = pd.read_csv('data.csv')
```

전처리 루틴 구성

결측치 처리, 형변환, 날짜 데이터 처리 등
일관된 데이터 정제 과정을 빠르게 구축할 때

```
df.fillna(0).astype('int')
```

보고서용 요약표 생성

그룹 요약, 피벗 테이블, 멀티인덱스 등
복잡한 집계와 분석 보고서를 만들 때

```
df.groupby('category').agg({'sales': 'sum'})
```

시계열 데이터 분석

날짜/시간 데이터의 리샘플링, 롤링 통계,
시간 기반 집계, 계절성 분석 등이 필요할 때

```
df.resample('M').mean()
```

```
# 설치
$ pip install pandas
```

주피터 노트북 활용하고자 하는 경우

```
# 설치
$ pip install jupyter
```

```
# 실행
$ jupyter notebook
```

Pandas 기본 문법

```python
import pandas as pd  # 표 형태 데이터 처리를 위한 라이브러리 로드

# 간단한 데이터프레임 생성
df = pd.DataFrame({
    "Name": ["Alice", "Bob"],
    "Age": [25, 30]
})

print(df.head())     # 상위 5행 미리보기
print(df.describe()) # 수치형 열의 요약 통계
```

실행 결과

- **head**는 데이터 스냅샷을 보여주며 구조 파악에 유용
- **describe**는 평균·표준편차·사분위수 등 기본 통계를 한 번에 확인

Series vs. DataFrame

구분	Series	DataFrame
차원	1차원 자료구조	2차원 자료구조
구성 단위	값(value) + 인덱스(index)	여러 Series(열)의 집합
인덱스 구조	단일 인덱스	행 인덱스 + 열 이름
형태	벡터(Vector)	표(Table)
유사 구조	NumPy 1D 배열	엑셀 시트, SQL 테이블
주요 활용	단일 변수 저장, 시계열 데이터 처리	다변량 데이터 분석, 전처리, 집계
확장성	단순, 개별 연산 중심	복잡한 데이터 조작 및 머신러닝 파이프라인에 적합
공식 문서	https://pandas.pydata.org/docs/reference/api/pandas.Series.html	https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

Series 예시

- **정의**: 1차원 배열 형태의 자료구조로, **값(value)** 과 **인덱스(index)** 쌍으로 구성
- **특징**:
  - Numpy 배열 기반으로 만들어져 빠른 연산 가능
  - 인덱스를 통해 위치기반 뿐만 아니라 라벨 기반 접근 가능
  - 벡터 연산이 가능해 수학/통계 처리에 유용

```python
import pandas as pd

s = pd.Series(
    [10, 20, 30], index=['a', 'b', 'c'])
print(s)
```

실행 결과

```
a    10
b    20
c    30
dtype: int64
```

DataFrame 예시

정의

- 2차원 테이블 형태의 자료구조
- 열(column)은 Series의 모음으로 구성
- 행(row)과 열(column) 모두에 인덱스를 가짐
- 엑셀 시트나 SQL 테이블과 유사한 구조

주요 특징

- 다양한 데이터 소스와 연동 가능 (CSV, Excel, SQL, JSON 등)
- 행(row) 단위, 열(column) 단위 접근이 모두 가능
- 데이터 정제, 집계, 병합, 피벗 등 복잡한 데이터 처리 지원
- NumPy 기반으로 빠른 연산 수행
- 머신러닝 라이브러리(scikit-learn 등)와 쉽게 연동 가능

```python
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Score': [85, 90, 95]
}
df = pd.DataFrame(data)
print(df)
```

실행 결과

```
      Name  Age  Score
0    Alice   25     85
1      Bob   30     90
2  Charlie   35     95
```

Pandas로 엑셀 파일 불러오기

```
import pandas as pd
# 1) 엑셀 파일 불러오기
df = pd.read_excel("02_sample_data.xlsx")

# 2) 데이터 확인
print(df.head())          # 처음 5행 출력
print(df.info())          # 데이터 요약 정보
print(df.describe())      # 숫자형 데이터 통계 요약

# 3) 특정 컬럼 선택
print(df["Name"])
print(df[["Name", "Salary"]])

# 4) 조건 필터링 (나이가 28 이상인 사람)
print(df[df["Age"] >= 28])

# 5) 그룹별 평균 (부서별 평균 연봉)
print(df.groupby("Department")["Salary"].mean())
```

Pandas 중급 예제 (1/2)

시나리오

- 카페 매출 데이터를 요일과 카테고리별로 분석하여 평균 가격과 수량을 요약하고, 요일-카테고리 피벗을 작성

피벗(Pivot)이란?

- 데이터를 재구조화(restructuring)하거나 회전시켜서 새로운 관점으로 보는 것
- 엑셀(Excel)에서의 Pivot
  - Table행/열 기준을 바꿔 데이터를 요약·집계하는 기능
  - 예: 판매 데이터를 "제품별 합계" → "지역별 합계"로 바꿔보기
- 데이터 분석에서의 Pivot
  - 테이블(데이터프레임)을 특정 열을 기준으로 행·열·값 구조로 재배치하는 과정
  - 본질적으로는 "데이터를 넓은(wide) 형태 ↔ 긴(long) 형태로 변환"하는 과정의 일부
- 일반적 의미
  - "축을 중심으로 회전하다"라는 뜻에서 파생
  - 데이터에서는 관점(축)을 바꿔서 새롭게 표현한다는 개념으로 사용

Pandas 중급 예제 (2/2)

```python
import pandas as pd

# 예시 매출 데이터
sales = pd.DataFrame({
    "Day":      ["Mon","Mon","Tue"],
    "Item":     ["Coffee","Cookie","Tea"],
    "Category": ["Drink","Dessert","Drink"],
    "Price":    [3000,2000,2500],
    "Qty":      [10,15,8]
})

# 그룹화로 카테고리별 평균 가격과 총수량 계산
group = sales.groupby("Category").agg(
    avg_price=("Price","mean"),
    total_qty=("Qty","sum")
)

# 피벗 테이블로 요일×카테고리 교차 합계 구성
pivot = sales.pivot_table(
    values="Qty", index="Day", columns="Category",
    aggfunc="sum", fill_value=0
)

print(group)
print(pivot)
```

- DataFrame.groupby(keys): 지정 키로 그룹을 만들고 각 그룹에 연산 적용
- .agg(new_col=("원본열","집계함수")): 여러 열에 대해 사용자 정의 집계 지정
- DataFrame.pivot_table(values, index, columns, aggfunc, fill_value)
  - values: 집계 대상 열 이름
  - index: 행 인덱스로 사용할 열
  - columns: 열 머리글로 사용할 열
  - aggfunc: 집계 함수(mean, sum, count 등)
  - fill_value: 결측 대체값

실행 결과

```
          avg_price  total_qty
Category
Dessert        2000         15
Drink          2750         18

Category  Dessert  Drink
Day
Mon            15     10
Tue             0      8
```

Pandas 고급 예제 (1/2)

시나리오

- 하루 단위 웹사이트 방문자 수 데이터가 있다. 주간 합계와 3일 이동평균을 계산하라

메서드

- DataFrame.rolling(window)은 이동 창 객체를 반환, .mean() 등 집계 적용
- DataFrame.resample(rule)은 주기 변경 집계 수행, rule="W"는 주 단위 의미

Pandas 고급 예제 (2/2)

```python
import pandas as pd

# 날짜 인덱스를 가진 시계열 데이터 구성
dates  = pd.date_range("2025-03-01", periods=5, freq="D")
visits = [120, 135, 128, 150, 160]
ts = pd.DataFrame({"date": dates, "visits": visits}).set_index("date")

# 3일 이동평균과 주간 합계 계산
# 최근 3일 평균
rolling = ts.rolling(window=3).mean()
# 주간 ("W") 단위 합계
weekly  = ts.resample("W").sum()

print(rolling)
print(weekly)
```

- rolling은 창 크기만큼의 평균을 구해 단기 추세 파악에 유용
- resample은 주기 변경 집계를 수행하여 기간 합계나 평균을 구할 때 사용

실행 결과

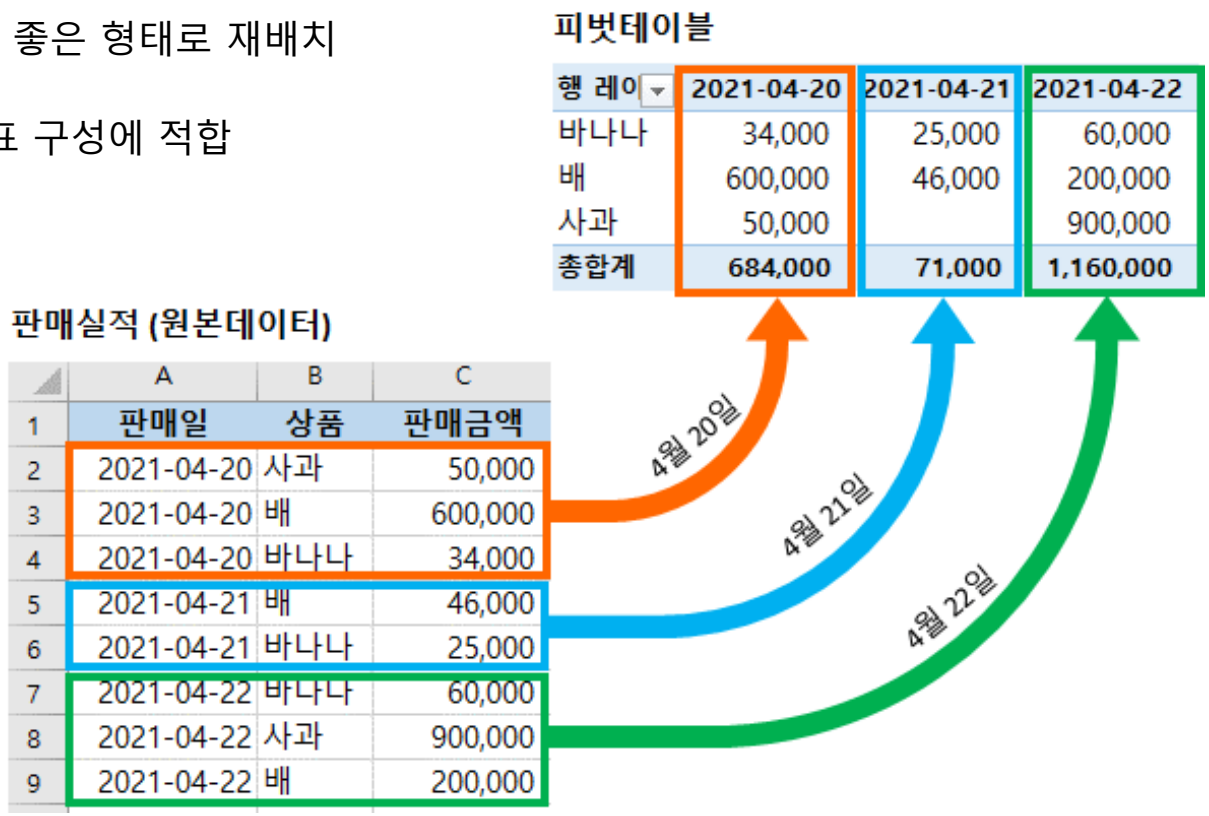
```
            visits
date
2025-03-01     NaN
2025-03-02     NaN
2025-03-03  127.67
2025-03-04  137.67
2025-03-05  146.00

            visits
date
2025-03-02     255
2025-03-09     438
```

헷갈리는 Pivot, 좀 더 알아보기

Pivot Table?

피벗 테이블(Pivot)?

- 행과 열 두 축으로 데이터를 교차 요약하는 2차원 표
- groupby 결과를 사람이 읽기 좋은 형태로 재배치
- 다차원 교차 분석과 보고서 표 구성에 적합



Pandas pivot, pivot_table 이해하기

- **기본 개념: pivot 및 pivot_table은 모두 데이터프레임을 재구조화 하는 데 사용**
- **공통 인자:**
  - index: 행 인덱스로 사용할 열, 해당 열의 유니크(unique)한 값들이 새로운 테이블의 인덱스로 사용
  - columns: 열 인덱스로 사용할 열, 해당 열의 유니크한 값(조합)의 개수만큼 새로운 열이 만들어짐
  - values: 셀에 채워질 값이 담긴 열, index와 values에서 만들어진 셀에 대응하는 값으로 채워짐 (값이 없다면 NaN으로 채워짐)
- **pivot**
  - 단순 재구조화 기능만 제공, (index, columns) 조합이 중복되면 ValueError 발생
- **pivot_table**
  - 재구조화에 더해 집계 기능 포함
  - aggfunc로 mean, sum, count 등 지정 가능
  - fill_value로 결측치 채우기 가능
  - margins=True를 통해 행/열 총합(마진) 포함 가능

pivot, pivot_table 사용법

```
# pivot 사용법
DataFrame.pivot(
Index=None,   # index: 행 인덱스로 사용할 열
Columns=None, # columns: 새 열 인덱스로 사용할 열
Values=None   # values: 데이터 값으로 채울 열
)

# pivot_table 사용법
DataFrame.pivot_table(
    index=None,     # 행 인덱스로 사용할 열
    columns=None,   # 열 인덱스로 사용할 열
    values=None,    # 값으로 채울 열
    aggfunc='mean', # 집계 함수 (기본값은 평균)
    fill_value=None # NaN 대신 채울 값
)
```

pivot 간단 예제

```python
import pandas as pd

# 샘플 데이터
data = {
    'Name': ['Alice', 'Alice', 'Bob', 'Bob'],
    'Subject': ['Math', 'English', 'Math', 'English'],
    'Score': [85, 90, 95, 80]
}

df = pd.DataFrame(data)
print("원본 데이터:")
print(df)

# pivot 사용
pivot_df = df.pivot(index='Name', columns='Subject', values='Score')

print("\n피벗 결과:")
print(pivot_df)
```

실행 결과

```
원본 데이터:
    Name  Subject  Score
0  Alice     Math     85
1  Alice  English     90
2    Bob     Math     95
3    Bob  English     80

피벗 결과:
Subject  English  Math
Name
Alice         90    85
Bob           80    95
```

pivot_table간단 예제

```
import pandas as pd
toy = pd.DataFrame({
    "Sex": ["female","female","male","male"],
    "Pclass": [1,3,1,3],
    "Survived": [1,0,1,0]
})
pv = pd.pivot_table(
    toy,
    values="Survived",
    index="Sex",
    columns="Pclass",
    aggfunc="mean",
    fill_value=0
)
print(pv)
```

실행 결과

```
Pclass     1    3
Sex
female   1.0  0.0
male     1.0  0.0
```

- 각 셀은 특정 성별×좌석등급의 평균 생존률 의미
- 시각화와 함께 사용하면 패턴을 빠르게 파악 가능

Pivot 에러를 처리하는 방법 (1/3)

```python
import pandas as pd

data = {
'Name': ['Alice', 'Alice', 'Bob'],
'Subject': ['Math', 'Math', 'English'],
'Score': [85, 90, 80]
}

df = pd.DataFrame(data)
print("원본 데이터:")
print(df)

# pivot 시도
pivot_df = df.pivot(
        index='Name’,
        columns='Subject’,
        values='Score’
)
print(pivot_df)
```

실행 결과

```
원본 데이터:
    Name Subject  Score
0  Alice    Math     85
1  Alice    Math     90
2    Bob English     80

Traceback (most recent call last):
  ...
ValueError: Index contains duplicate entries,
cannot reshape
```

```
index='Name', columns='Subject', alues='Score'
로 지정했을 때:

  Alice + Math → 85

  Alice + Math → 90

  Bob + English → 80

여기서 (Name='Alice', Subject='Math') 조합이
두 번 등장하여 에러가 발생함
```

Pivot 에러를 처리하는 방법 (2/3)

에러 대처 방법

- `pivot_table`의 집계 함수(aggregation function, agg_func) 활용

중복된 (index, column) 조합 확인

- 에러 상황: (Name="Alice", Subject="Math") 조합이 85와 90 두 번 존재
- aggfunc 적용

  aggfunc='mean' → 평균값 ((85+90)/2 = 87.5), aggfunc='sum' → 합계 (85+90 = 175)

  aggfunc='max' → 최댓값 (90), aggfunc='min' → 최솟값 (85)

  aggfunc='count' → 개수 (2),

결과 테이블에 단일 값으로 반영

따라서 pivot에서 발생하는 “중복 문제”를 해결할 수 있음

Pivot 에러를 처리하는 방법 (3/3)

```python
import pandas as pd

data = {
'Name': ['Alice', 'Alice', 'Bob'],
'Subject': ['Math', 'Math', 'English'],
'Score': [85, 90, 80]
}

df = pd.DataFrame(data)

# aggfunc 적용
mean_pivot = df.pivot_table(index='Name', columns='Subject', values='Score', aggfunc='mean')
sum_pivot  = df.pivot_table(index='Name', columns='Subject', values='Score', aggfunc='sum')
count_pivot = df.pivot_table(index='Name', columns='Subject', values='Score', aggfunc='count')

print("평균:\n", mean_pivot)
print("\n합계:\n", sum_pivot)
print("\n개수:\n", count_pivot)
```

실행 결과

```
평균:
Subject  English   Math
Name
Alice        NaN   87.5
Bob         80.0    NaN

합계:
Subject  English  Math
Name
Alice        NaN   175
Bob         80.0   NaN

개수:
Subject  English  Math
Name
Alice        0.0     2
Bob          1.0     0
```

Toy Project

Toy Project Description

■ **목표: Titanic 데이터셋으로 전처리, 그룹 요약, 피벗 분석, 시각화를 수행한다**

■ **데이터: Kaggle Titanic Dataset 다운로드 링크**

- https://www.kaggle.com/c/titanic/data

■ **핵심 개념**

- GroupBy: 특정 컬럼으로 데이터를 묶고 요약 통계 계산
- Pivot Table: 행과 열 두 축을 동시에 사용해 교차 분석 수행
  - 예: 행=Sex, 열=Pclass, 값=Survived 평균

Toy Project Requirements

과제 요구사항

- 결측치 처리 전략 수립 및 적용
- GroupBy와 Pivot Table을 이용한 데이터 요약
- 최소 2개의 시각화 산출
- 데이터에서 인사이트 3가지 이상 도출

제출물

- 정리된 노트북 또는 .py 스크립트
- 결과 표·그래프 이미지 2개 이상
- 인사이트 요약 5줄 이내

평가 기준

- **전처리의 타당성**
- **GroupBy와 Pivot 분석의 정확성**
- **시각화의 적정성 및 해석 명확성**
- **인사이트 도출의 설득력**

연관 메서드 소개

결측값을 상수 또는 함수(function)으로 대체

- DataFrame에서 적용: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html
- Series에서 적용: https://pandas.pydata.org/docs/reference/api/pandas.Series.fillna.html

Series.mode() / Series.median()

- 최빈값·중앙값 계산에 사용
- https://pandas.pydata.org/docs/reference/api/pandas.Series.mode.html

key로 묶은 그룹의 평균 계산

- DataFrame.groupby(key)["col"].mean()
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.mean.html

DataFrame.pivot_table(values, index, columns, aggfunc)

- 교차 축으로 집계하여 2차원 요약표 생성
- https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html

솔루션 코드

```python
import pandas as pd
import matplotlib.pyplot as plt

# 1) 데이터 로드
df = pd.read_csv("train.csv")  # Kaggle에서 내려받은 train.csv 사용

# 2) 빠른 점검
print("shape:", df.shape)              # 데이터 크기 확인
print("null counts:\n", df.isnull().sum())  # 결측치 개수 확인

# 3) 전처리
df["Age"] = df["Age"].fillna(df["Age"].median())     # Age는 중앙값으로 대치
df["Embarked"] = df["Embarked"].fillna(df["Embarked"].mode()[0])  # Embarked는 최빈값으로 대치

# 4) GroupBy 예시: 성별별 평균 생존률
survival_by_sex = df.groupby("Sex")["Survived"].mean()
print("\nGroupBy - Survival rate by Sex:\n", survival_by_sex)

# 5) Pivot Table 예시: 성별×좌석등급 평균 생존률
pivot_survival = pd.pivot_table(df, values="Survived", index="Sex", columns="Pclass", aggfunc="mean")
print("\nPivot Table - Survival rate by Sex and Pclass:\n", pivot_survival)

# 6) 시각화 (영문 라벨 권장)
plt.figure()
df["Age"].plot(kind="hist", bins=20, title="Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.tight_layout()
plt.savefig("./imgs/chap_02/age_distribution.png", dpi=200)

plt.figure()
survival_by_sex.plot(kind="bar", title="Survival Rate by Sex")
plt.ylabel("Survival Rate")
plt.tight_layout()
plt.savefig("./imgs/chap_02/survival_by_sex.png", dpi=200)
```

실행 결과

다음 슬라이드 참조

결과 해석

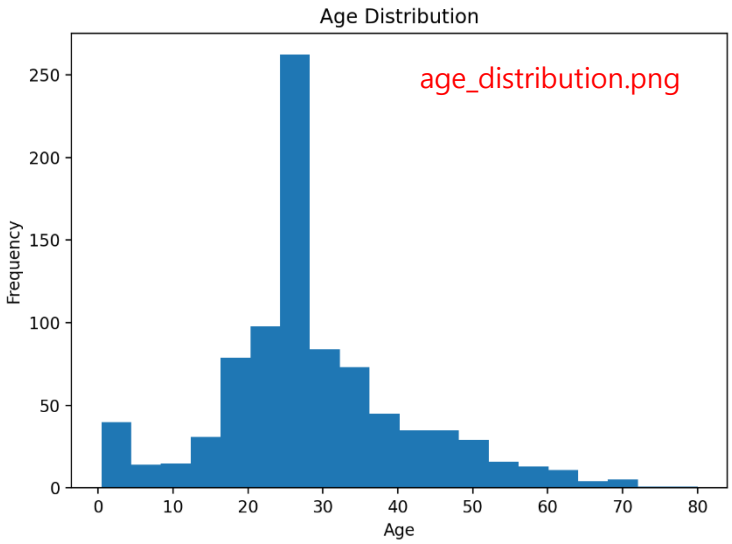
실행 결과

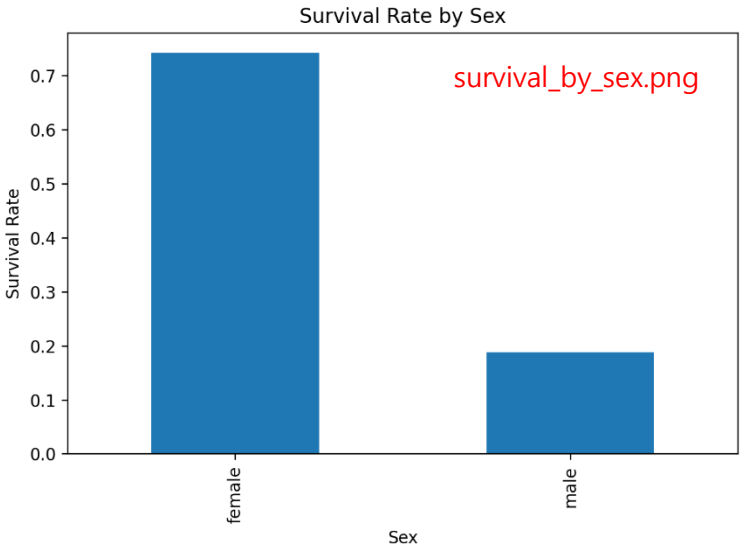
```
shape: (891, 12)
null counts:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64

GroupBy - Survival rate by Sex:
 Sex
female    0.742038
male      0.188908
Name: Survived, dtype: float64

Pivot Table - Survival rate by Sex and Pclass:
 Pclass         1         2         3
Sex
female  0.968085  0.921053  0.500000
male    0.368852  0.157407  0.135447
```

결과 해석

- 여성 승객의 평균 생존률이 남성보다 훨씬 높음
- 좌석 등급이 높을수록 평균 생존률이 증가하는 경향 관찰

➔ 승선에 따라 운임 차이가 존재하며 사회경제적 요인과의 연관성 있음을 시사





수고하셨습니다 ..^^..