CS1090A 데이터 과학 입문
Pandas 기초 노트

Harvard University
(강의 내용 정리)

October 26, 2025

▣ **강의명:** CS109A: 데이터 과학 입문

▣ **주차:** Lecture 03

▣ **교수명:** Pavlos Protopapas, Kevin Rader, Chris Gumb

▣ **목적:** Lecture 03의 핵심 개념 학습

Contents

1 개요

□ 핵심 요약

이 문서는 Harvard CS1090A 데이터 과학 입문 강의의 Pandas 기초 부분을 다룹니다. Pandas는 파이썬에서 표 형태의 데이터를 다루는 데 필수적인 라이브러리입니다. 이 노트는 Pandas의 핵심 자료구조인 `Series`와 `DataFrame`의 개념, 생성 방법, 데이터 로딩, 검사, 정제, 기본적인 데이터 분석 방법을 설명합니다. 처음 배우는 사람도 쉽게 이해할 수 있도록 예시와 함께 단계별로 설명합니다. 이 노트만으로도 Pandas의 기본을 익히고 실습할 수 있도록 구성했습니다.

주요 학습 목표:

- Pandas의 기본 자료구조인 `Series`와 `DataFrame` 이해하기
- 다양한 방법으로 `Series`와 `DataFrame` 생성하기
- CSV 파일 등 외부 데이터를 Pandas `DataFrame`으로 불러오기
- 데이터를 검사하고 요약하는 기본 방법 익히기 (‘head‘, ‘info‘, ‘describe‘ 등)
- 데이터 정제 기법 배우기 (컬럼명 변경, 타입 변환, 결측치 및 중복값 처리 등)
- Boolean indexing, `loc`, `iloc`을 이용한 데이터 선택 및 필터링
- 데이터 정렬, 집계, 그룹화 기초 (‘sort_values‘, ‘groupby‘, ‘agg‘)
- 정제된 데이터를 파일로 저장하기 (‘to_csv‘)

참고: 이 노트는 실제 강의 내용과 코드를 바탕으로 재구성되었으며, 추가적인 설명과 예시가 포함되어 있습니다.

2 용어 정리

데이터 분석 여정에서 자주 만나게 될 Pandas 관련 용어들을 미리 알아두면 학습에 큰 도움이 됩니다. 아래 표는 이 노트에서 사용되는 주요 용어들을 정리한 것입니다.

용어	쉬운 설명	원어	비고
Pandas	파이썬으로 표 형태 데이터를 쉽게 다루게 해주는 도구 모음	Pandas	데이터 분석의 필수 라이브러리
Series	1차원 배열 형태의 데이터 (하나의 열)	Series	값과 인덱스로 구성됨
DataFrame	2차원 표 형태의 데이터 (여러 개의 열)	DataFrame	Series 여러 개가 모인 것
Index	데이터의 각 행(row)을 식별하는 이름표 또는 번호	Index	기본은 0부터 시작하는 숫자
dtype	데이터의 종류 (숫자, 문자열, 날짜 등)	Data Type	`int64`, `float64`, `object`, `bool`, `datetime64`, `category` 등
NaN	데이터가 없음을 나타내는 특별한 값	Not a Number	결측치(Missing Value)라고도 함
Boolean Indexing	참/거짓(True/False) 값으로 원하는 데이터만 골라내는 방법	Boolean Indexing	조건에 맞는 데이터를 필터링할 때 사용
loc	이름표(label) 기반으로 데이터를 선택하는 방법	loc (Label-based)	예: `df.loc[3]`, `df.loc['row_name']`
iloc	위치(position) 기반으로 데이터를 선택하는 방법	iloc (Integer position-based)	예: `df.iloc[0]`, `df.iloc[0:5]`
Method Chaining	여러 함수(메서드)를 점(.)으로 연결하여 순차적으로 실행하는 것	Method Chaining	코드를 간결하게 만듦. 예: `df.sort_values().head()`
GroupBy	특정 기준(컬럼 값)에 따라 데이터를 그룹으로 묶는 작업	GroupBy	그룹별 통계 계산 등에 사용
Aggregation	그룹으로 묶인 데이터에 대해 요약 통계(평균, 합계 등)를 계산하는 것	Aggregation	`agg()`, `mean()`, `sum()`, `count()` 등
Crosstab	두 변수(컬럼) 간의 빈도수를 표 형태로 교차 분석하는 것	Cross-tabulation	범주형 변수 간의 관계 파악에 유용
Explode	하나의 셀에 리스트 형태로 들어있는 값을 여러 행으로 펼치는 작업	Explode	콤마로 구분된 문자열 등을 분리할 때 사용

3 핵심 개념: Pandas 자료구조

Pandas는 파이썬에서 기본적으로 제공하지 않는, 데이터 분석에 특화된 두 가지 핵심 자료구조를 제공합니다: **Series**와 **DataFrame**.

3.1 Series: 1차원 데이터의 마법 지팡이

Series란?

`Series`는 마치 라벨(이름표)이 붙어 있는 1차원 배열과 같습니다. 엑셀 시트의 한 열(column)이나, 키(key)와 값(value)으로 이루어진 사전(dictionary)을 떠올리면 이해하기 쉽습니다. 각 데이터 값에는 고유한 이름표, 즉 **인덱스(index)**가 붙어 있어 데이터를 쉽게 찾고 다룰 수 있습니다.

왜 Series가 필요한가? 파이썬의 기본 리스트(list)도 1차원 데이터를 저장할 수 있지만, Series는 다음과 같은 장점을 가집니다.

- **명시적인 인덱스:** 단순한 숫자 위치뿐만 아니라 원하는 문자열 등으로 인덱스를 지정할 수 있습니다.
- **NumPy 기반 성능:** 내부적으로 고성능 NumPy 배열을 사용하므로 대량 데이터 처리 속도가 빠릅니다.
- **데이터 정렬 및 연산 용이성:** 인덱스를 기준으로 데이터를 정렬하거나, Series 간의 연산을 쉽게 수행할 수 있습니다.
- **다양한 데이터 타입 지원 및 결측치 처리:** 숫자, 문자열뿐 아니라 다양한 데이터 타입을 지원하며, 데이터가 없는 경우(결측치, NaN)를 효과적으로 다룰 수 있습니다.

3.1.1 Series 생성하기

가장 기본적인 방법은 파이썬 리스트를 사용하는 것입니다.

```python
import pandas as pd
import numpy as np

# 숫자리스트로  Series 생성
numbers = [1, 3, 5, np.nan, 6, 8]
s = pd.Series(numbers)
print(s)
```

Listing 1: 파이썬 리스트로 Series 생성

결과:

```
0    1.0
1    3.0
2    5.0
3    NaN
```

```
4    6.0
5    8.0
dtype: float64
```

왼쪽의 0, 1, 2...는 자동으로 생성된 기본 인덱스이고, 오른쪽은 리스트의 값입니다. np.nan은 데이터가 없음을 의미하는 결측치입니다. 데이터 타입(dtype)은 실수(float64)로 자동 지정되었습니다. (NaN 때문에)

인덱스를 직접 지정할 수도 있습니다.

```python
# 문자열리스트와사용자정의인덱스
data = ['a', 'b', 'c']
index = [5, 3, 1]
s_custom_index = pd.Series(data, index=index)
print(s_custom_index)

# 인덱스를문자열로지정
s_string_index = pd.Series([-3, -2, -1], index=['foo', 'bar', 'baz'])
print(s_string_index)
```

Listing 2: 인덱스를 지정하여 Series 생성

결과:

```
5    a
3    b
1    c
dtype: object

foo   -3
bar   -2
baz   -1
dtype: int64
```

인덱스가 숫자가 아닌 문자열이어도 되며, 순서대로가 아니어도 됩니다.

3.1.2 Series의 주요 속성

Series 객체는 데이터 자체 외에도 유용한 정보를 속성으로 가지고 있습니다.

```python
l = [-3, -2, -1]
series = pd.Series(l, name='ints!', dtype=str) # 이름과타입지정
series.index = ['foo', 'bar', 'foo'] # 인덱스변경중복 ( 가능!)

print(f"값 (Values): {series.values}")
print(f"인덱스 (Index): {series.index}")
print(f"이름 (Name): {series.name}")
print(f"데이터 타입 (dtype): {series.dtype}")
```

```
print(f"값의 타입: {type(series.values)}")
```

Listing 3: Series 속성 확인

결과:

```
값 (Values): ['-3' '-2' '-1']
인덱스 (Index): Index(['foo', 'bar', 'foo'], dtype='object')
이름 (Name): ints!
데이터 타입 (dtype): object
값의 타입: <class 'numpy.ndarray'>
```

- `values`: Series의 실제 데이터 값들을 NumPy 배열 형태로 반환합니다. (**주의:** NumPy 배열은 모든 원소가 동일한 데이터 타입이어야 합니다. 만약 다양한 타입의 데이터가 Series에 있다면, 가장 포괄적인 타입인 `object`로 변환됩니다. `object` 타입은 실제 값 대신 메모리 주소를 저장하므로 성능 저하의 원인이 될 수 있습니다.)
- `index`: Series의 인덱스 객체를 반환합니다. 인덱스는 단순한 숫자가 아니라, 데이터를 식별하는 라벨이며 중복될 수도 있습니다.
- `name`: Series의 이름을 문자열로 반환합니다. DataFrame의 컬럼명으로 사용됩니다.
- `dtype`: Series에 저장된 데이터의 타입을 알려줍니다.

3.1.3 Series 기본 메서드

Series에는 데이터를 탐색하고 요약하는 데 유용한 여러 메서드가 내장되어 있습니다.

```
data = [1, 3, 5, np.nan, 6, 8]
s = pd.Series(data)

print("--- 처음개 2 데이터 (head) ---")
print(s.head(2))

print("\n--- 마지막개 2 데이터 (tail) ---")
print(s.tail(2))

print("\n--- 기술통계요약   (describe) ---")
print(s.describe())
```

Listing 4: Series 기본 메서드 사용

결과:

```
--- 처음 2개 데이터 (head) ---
0    1.0
1    3.0
dtype: float64
```

```
--- 마지막 2개 데이터 (tail) ---
4    6.0
5    8.0
dtype: float64

--- 기술 통계 요약 (describe) ---
count    5.000000
mean     4.600000
std      2.701851
min      1.000000
25%      3.000000
50%      5.000000
75%      6.000000
max      8.000000
dtype: float64
```

- `head(n)`: Series의 처음 n개 데이터를 보여줍니다. (기본값 n=5)
- `tail(n)`: Series의 마지막 n개 데이터를 보여줍니다. (기본값 n=5)
- `describe()`: 수치형 데이터의 경우 개수(count), 평균(mean), 표준편차(std), 최솟값(min), 사분위수(25

3.2 DataFrame: 2차원 데이터의 강력한 테이블

DataFrame란?

`DataFrame`은 여러 개의 `Series`가 같은 인덱스를 공유하며 모여 있는 2차원 표 형태의 자료 구조입니다. 엑셀 스프레드시트나 데이터베이스 테이블과 매우 유사합니다. 각 열(column)은 하나의 `Series`에 해당하며, 고유한 컬럼 이름을 가집니다. 각 행(row)은 인덱스로 식별됩니다.

왜 DataFrame이 필요한가? `Series`가 1차원 데이터를 다루는 데 유용하다면, `DataFrame`은 다음과 같은 이유로 2차원 데이터를 다루는 데 필수적입니다.

- **표 형태 데이터 처리:** 엑셀, CSV 파일 등 일반적인 표 형태 데이터를 직관적으로 표현하고 조작할 수 있습니다.
- **다양한 데이터 타입:** 각 열(Series)마다 다른 데이터 타입(숫자, 문자, 날짜 등)을 가질 수 있습니다.
- **유연한 인덱싱 및 선택:** 행과 열의 이름 또는 위치를 이용해 원하는 데이터를 쉽게 선택하고 필터링할 수 있습니다.
- **강력한 데이터 정제 및 변환 기능:** 결측치 처리, 데이터 타입 변환, 데이터 합치기, 그룹화 등 다양한 데이터 처리 기능을 제공합니다.
- **다양한 입출력 지원:** CSV, Excel, 데이터베이스 등 다양한 형식의 데이터를 쉽게 읽고 쓸 수 있습니다.

3.2.1 DataFrame 생성하기

DataFrame을 만드는 방법은 여러 가지가 있습니다.

1. 딕셔너리의 리스트 사용 (행 단위 생성) 각 딕셔너리가 하나의 행(row)이 되고, 딕셔너리의 키가 열(column) 이름이 됩니다.

```
import pandas as pd

data = [{'fruit': 'apple', 'color': 'red'},
        {'fruit': 'grape', 'color': 'purple'}]

df_fruits = pd.DataFrame(data)
print(df_fruits)
```

Listing 5: 딕셔너리의 리스트로 DataFrame 생성

결과:

```
   fruit   color
0  apple     red
1  grape  purple
```

2. 리스트의 딕셔너리 사용 (열 단위 생성) 딕셔너리의 키가 열(column) 이름이 되고, 값으로

주어지는 리스트가 해당 열의 데이터가 됩니다.

```
import pandas as pd

data = {'fruit': ['apple', 'grape'],
        'color': ['red', 'purple']}

df_fruits_alt = pd.DataFrame(data)
print(df_fruits_alt)
```

Listing 6: 리스트의 딕셔너리로 DataFrame 생성

결과:

```
   fruit   color
0  apple     red
1  grape  purple
```

3. NumPy 배열 사용 2차원 NumPy 배열로부터 DataFrame을 생성할 수 있습니다. 이때 열 이름을 명시적으로 지정해주는 것이 좋습니다.

```
import pandas as pd
import numpy as np

# 차원2 NumPy 배열생성예  (: 100x2 크기, 다변량정규분포데이터  )
data_np = np.random.multivariate_normal(mean=[0,-1], cov
    =[[1,0.5],[0.5,1]], size=100)

# NumPy 배열과컬럼이름을사용하여    DataFrame 생성
df_np = pd.DataFrame(data=data_np, columns=["X", "Y"])
print(df_np.head()) # 처음개 5 행만출력
```

Listing 7: NumPy 배열로 DataFrame 생성

결과 (값은 실행 시마다 다름):

```
          X         Y
0 -1.558152 -1.781442
1 -0.114449 -2.243731
2 -1.675996 -2.528247
3  0.308387 -1.627142
4 -1.138339 -1.671097
```

3.2.2 DataFrame의 주요 속성

DataFrame도 Series와 유사하게 유용한 속성들을 제공합니다.

```
# 위의 df_np DataFrame 사용
print(f"형태 (Shape): {df_np.shape}")
```

```
print(f"컬럼 (Columns): {df_np.columns}")
print(f"인덱스 (Index): {df_np.index}")
print(f"값 (Values): \n{df_np.values[:2]}") # 값은 NumPy 배열, 처음행
    만 2 출력
```

Listing 8: DataFrame 속성 확인

결과 (값은 실행 시마다 다름):

```
형태 (Shape): (100, 2)
컬럼 (Columns): Index(['X', 'Y'], dtype='object')
인덱스 (Index): RangeIndex(start=0, stop=100, step=1)
값 (Values):
[[-1.55815228 -1.7814424 ]
 [-0.11444917 -2.24373138]]
```

- `shape`: DataFrame의 형태(행 개수, 열 개수)를 튜플로 반환합니다.
- `columns`: DataFrame의 열 이름(컬럼명)들을 Index 객체로 반환합니다.
- `index`: DataFrame의 행 이름(인덱스)들을 Index 객체로 반환합니다.
- `values`: DataFrame의 모든 데이터 값을 2차원 NumPy 배열 형태로 반환합니다.

3.2.3 DataFrame 기본 메서드

DataFrame의 데이터를 빠르게 파악하기 위한 기본 메서드들입니다.

```
# 위의 df_np DataFrame 사용
print("--- 처음개 3 데이터 (head) ---")
print(df_np.head(3))

print("\n--- 마지막개 2 데이터 (tail) ---")
print(df_np.tail(2))

print("\n--- 요약정보  (info) ---")
df_np.info()

print("\n--- 기술통계요약   (describe) ---")
print(df_np.describe())
```

Listing 9: DataFrame 기본 메서드 사용

결과 (값은 실행 시마다 다름):

```
--- 처음 3개 데이터 (head) ---
          X         Y
0 -1.558152 -1.781442
1 -0.114449 -2.243731
2 -1.675996 -2.528247
```

```
--- 마지막 2개 데이터 (tail) ---
           X         Y
98 -0.210365 -2.146889
99  1.032841  0.338218

--- 요약 정보 (info) ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X       100 non-null    float64
 1   Y       100 non-null    float64
dtypes: float64(2)
memory usage: 1.7 KB

--- 기술 통계 요약 (describe) ---
                X           Y
count  100.000000  100.000000
mean    -0.088686   -0.963459
std      0.981885    0.989715
min     -2.428570   -3.131102
25%     -0.781459   -1.639194
50%     -0.122675   -0.970172
75%      0.540450   -0.340263
max      2.308703    1.638069
```

- `head(n)`: DataFrame의 처음 n개 행을 보여줍니다. (기본값 n=5)
- `tail(n)`: DataFrame의 마지막 n개 행을 보여줍니다. (기본값 n=5)
- `info()`: DataFrame의 전반적인 정보를 요약하여 보여줍니다. 인덱스 타입, 컬럼 정보(이름, non-null 개수, 데이터 타입), 메모리 사용량 등을 확인할 수 있습니다. 데이터 타입을 확인하고 결측치 유무를 파악하는 데 매우 유용합니다.
- `describe()`: 수치형 컬럼들에 대한 기술 통계량을 계산하여 보여줍니다. 데이터의 분포를 빠르게 파악할 수 있습니다.

4 절차/방법: 데이터 다루기 실전

이제 실제 데이터를 Pandas DataFrame으로 불러와서 검사하고, 필요한 형태로 정제하고, 간단한 분석을 수행하는 과정을 단계별로 살펴보겠습니다. 예제 데이터로는 CS1090A 학생 설문조사 결과를 사용합니다.

4.1 1단계: 데이터 불러오기 및 초기 검사

가장 먼저 할 일은 데이터를 DataFrame으로 읽어 들이는 것입니다. CSV(Comma Separated Values) 파일은 가장 흔한 데이터 형식 중 하나이며, `pd.read_csv()` 함수를 사용합니다.

```
import pandas as pd

# CSV 파일을으로 DataFrame 불러오기
# 'data/cs1090a_survey_raw.csv' 파일이있다고가정
# 실제파일경로는환경에맞게수정해야합니다 .
try:
    df = pd.read_csv("data/cs1090a_survey_raw.csv")
    print("CSV 파일로딩성공 !")
except FileNotFoundError:
    print("오류: CSV 파일을찾을수없습니다 . 파일경로를확인하세요 .")
    # 예제진행을위해임시 DataFrame 생성
    data = {'Timestamp': ['9/9/2025 17:43:09'],
            'What\'s your Harvard affiliation?\n': ['Bachelor\'s (CS)'],
            'Have you used Jupyter Notebooks before?': ['Yes'],
            'How many years of Python programming experience do you have
               ?\n(Choose the range that best fits your experience.)': [
               'Less than 1 year'],
            'Rate your current Pandas skill level.': [2],
            'What is your primary OS?': ['MacOS'],
            'Do use normally code/browse in dark mode?': ['No'],
            'What languages do you speak? \n(Comma separated)\n\nExample
               : Hittite, Elvish, Cornish, Klingon': ['English'],
            'Which continents have you visited?': ['Africa, Asia, Europe
               , North America, South America'],
            'When were you born?': ['11/15/2004'],
            'What time do you usually wake up in the morning?': ['
               10:00:00 AM'],
            'What time do you usually go to bed?': ['12:00:00 AM'],
            'Favorite Season?': ['Spring'],
            'Where do you usually get your caffeine?': ['Tea'],
            'Which kind of pet do you prefer?': ['Pet rock'],
            'What\'s your favorite movie?': [None], # 예시로 None 사용
            'What movie genres do particularly enjoy?\n(select as many
               as you like)': ['Comedy'],
            'List up to 3 of your hobbies.\n(comma separated)\n\nExample
               : playing kazoo, bird watching, stamp collecting': ['
```

```
                tennis, movies, eating'],
30              'How was HW0?': [None]}
31      df = pd.DataFrame(data)
```

Listing 10: CSV 파일 읽어오기

데이터를 불러온 후에는 어떤 데이터가 들어 있는지 확인하는 것이 중요합니다. `head()`, `shape`, `columns`, `info()`, `describe()` 등을 사용합니다.

```
1  # 처음개 5 행확인
2  print("--- 데이터미리보기 (head) ---")
3  print(df.head())
4
5  # 데이터크기확인행 (, 열개수 )
6  print(f"\n--- 데이터크기 (shape) ---")
7  print(f"{df.shape[0]} 행, {df.shape[1]} 열")
8
9  # 컬럼명리스트확인
10 print("\n--- 컬럼명 (columns) ---")
11 print(list(df.columns))
12
13 # 데이터요약정보확인결측치 (, 데이터타입 )
14 print("\n--- 요약정보 (info) ---")
15 df.info()
16
17 # 수치형데이터기술통계확인
18 print("\n--- 기술통계 (describe) ---")
19 print(df.describe())
```

Listing 11: 데이터 초기 검사

초기 검사 결과 (예시 데이터 기준):

- `head()`: 데이터의 실제 값과 컬럼명을 눈으로 확인할 수 있습니다. 컬럼명이 너무 길거나 줄바꿈 문자가 포함되어 지저분해 보입니다. 일부 값은 비어있는 것 같습니다 (NaN).
- `shape`: 데이터의 전체 크기를 알려줍니다. 예제 데이터는 175행 19열입니다.
- `columns`: 모든 컬럼명을 리스트로 보여줍니다. 너무 길고 특수문자가 있어 다루기 불편합니다.
- `info()`: 각 컬럼의 non-null 값 개수와 데이터 타입을 보여줍니다. 대부분의 컬럼이 `object` 타입(주로 문자열)이며, 'Rate your current Pandas skill level.' 컬럼만 `int64`(정수) 타입입니다. Non-null 개수를 보면 여러 컬럼에 결측치가 있음을 알 수 있습니다 ('dob', 'wake_time', 'fav_movie', 'hobbies', 'hw0' 등).
- `describe()`: 현재는 'Rate your current Pandas skill level.' 컬럼에 대한 통계만 보여줍니다. 평균 2.5, 표준편차 1.1 정도임을 알 수 있습니다.

4.2 2단계: 데이터 정제

초기 검사 결과를 바탕으로 데이터를 분석하기 좋은 형태로 만드는 정제 작업을 수행합니다.

4.2.1 컬럼명 변경

길고 복잡한 컬럼명은 사용하기 불편하므로, 짧고 의미 있는 이름으로 변경합니다. 일반적으로 소문자와 언더스코어(_)를 사용하는 것이 좋습니다.

```
# 새컬럼명리스트정의
new_cols = [
    "timestamp", "program", "jupyter", "python_exp", "pandas_skill", "os
       ", "dark_mode",
    "languages", "continents", "dob", "wake_time", "sleep_time", "
       fav_season",
    "caffeine", "pet", "fav_movie", "fav_genres", "hobbies", "hw0"
]

# 의DataFrame columns 속성에새리스트할당
df.columns = new_cols

# 변경된컬럼명확인
print("--- 변경된컬럼명확인    (head) ---")
print(df.head())
```

Listing 12: 컬럼명 변경

이제 `df.program`이나 `df['pandas_skill']`처럼 훨씬 간결하게 컬럼에 접근할 수 있습니다.

4.2.2 불필요한 컬럼 제거

분석에 사용하지 않을 컬럼은 제거하여 DataFrame을 가볍게 만듭니다. 여기서는 'timestamp' 컬럼을 제거합니다. `drop()` 메서드를 사용하며, `axis=1`은 열을 기준으로 제거하라는 의미입니다. (`axis=0`은 행 기준)

```
# 'timestamp' 컬럼제거   (axis은=1 열을의미 )
# inplace=를True 사용하면원본이   DataFrame 바로수정됨
# 여기서는수정된결과를다시에      df 할당
df = df.drop("timestamp", axis=1)

# 제거확인   (shape 및 head)
print(f"제거 후데이터크기   : {df.shape}")
print(df.head(2))
```

Listing 13: 컬럼 제거

컬럼 개수가 19개에서 18개로 줄어든 것을 확인할 수 있습니다.

4.2.3 데이터 타입 변환

현재 대부분의 컬럼이 object 타입입니다. 분석 목적에 맞게 적절한 데이터 타입으로 변환해야 합니다.

Boolean 타입 변환: 'Yes'/'No' 형태의 데이터를 True/False로 변환합니다.

```
# 'dark_mode' 컬럼값이 'yes'이면 True, 아니면로 False 변환
# .str.lower()를 먼저적용하여대소문자구분없이처리
df['dark_mode'] = (df['dark_mode'].str.lower() == 'yes')

# 'jupyter' 컬럼도동일하게처리
df['jupyter'] = (df['jupyter'].str.lower() == 'yes')

# 타입변경확인
print("--- dark_mode dtype ---")
print(df['dark_mode'].dtype)
print("--- jupyter dtype ---")
print(df['jupyter'].dtype)
```

Listing 14: Boolean 타입 변환

이제 bool 타입이 되어 논리 연산에 사용하기 편리해졌습니다.

Category 타입 변환: 정해진 몇 가지 값 중 하나를 가지는 범주형 데이터는 category 타입으로 변환하는 것이 좋습니다. 메모리 효율성을 높이고, 해당 컬럼이 가질 수 있는 값을 제한할 수 있습니다.

```
# 'os' 컬럼을 category 타입으로변환
df['os'] = df['os'].astype('category')

# 타입및카테고리확인
print(df['os'].dtype)
print(df['os'].cat.categories)
```

Listing 15: Category 타입 변환

순서 있는 Category 타입 변환 (Ordinal): 'python_exp'처럼 순서가 있는 범주형 데이터는 순서를 명시하여 category 타입으로 변환합니다. 이렇게 하면 크기 비교나 정렬이 의미 있게 가능해집니다. CategoricalDtype을 사용합니다.

```
from pandas.api.types import CategoricalDtype

# Python 경험순서정의
experience_order = ['Less than 1 year', '1-2 years', '2-4 years', '4+
    years']

# 순서있는 CategoricalDtype 생성
experience_dtype = CategoricalDtype(categories=experience_order, ordered
    =True)

```

```
# 'python_exp' 컬럼을정의된타입으로변환
# 원본컬럼을유지하고새컬럼 'python_experience' 생성강의 ( 노트방식 )
# 실제로는 df['python_exp'] = df['python_exp'].astype(experience_dtype) 처럼덮어쓰는경우가많음
df['python_experience'] = df['python_exp'].astype(experience_dtype)

# 타입및순서확인
print(df['python_experience'].dtype)
print(df['python_experience'].head())
# 예: 경험이 '1-2 years보다' 많은사람필터링
print("\n--- 경험 '1-2 years' 초과 ---")
print(df[df['python_experience'] > '1-2 years']['python_experience'].head())
```

Listing 16: 순서 있는 Category 타입 변환

Datetime 타입 변환: 날짜/시간 관련 문자열은 `datetime` 타입으로 변환해야 날짜 계산 등을 수행할 수 있습니다. `pd.to_datetime()` 함수를 사용하며, `errors='coerce'` 옵션은 변환 불가능한 값을 NaT(Not a Time)로 처리합니다.

```
# 'dob' 컬럼을 datetime 타입으로변환
df['dob'] = pd.to_datetime(df['dob'], errors='coerce')

# 타입확인
print(df['dob'].dtype)
```

Listing 17: Datetime 타입 변환

시간 정보를 가진 'wake_time', 'sleep_time'도 유사하게 처리할 수 있으나, 시간만 다룰 경우 다른 방식이 더 적합할 수 있습니다. 이 노트에서는 일단 변환하지 않습니다.

4.2.4 텍스트 정규화

사용자가 직접 입력한 텍스트 데이터는 대소문자가 섞여 있거나 앞뒤 공백이 있을 수 있습니다. 분석의 일관성을 위해 모두 소문자로 변환하고 불필요한 공백을 제거하는 것이 좋습니다. `str.lower()`와 `str.strip()` 메서드를 사용합니다.

```
# object 타입주로( 문자열) 컬럼만선택
str_cols = df.select_dtypes(include='object').columns

# 각문자열컬럼에대해소문자변환적용
for c in str_cols:
    # 결측치가있을수있으므로 .str 접근자사용
    df[c] = df[c].str.lower()
    # 필요시앞뒤공백제거 : df[c] = df[c].str.strip()

# 변경확인예 (: program 컬럼)
print(df['program'].head())
```

Listing 18: 텍스트 소문자 변환

4.2.5 중복 데이터 확인 및 처리

완전히 동일한 행이 중복되어 있는지 확인합니다. `duplicated()`는 각 행이 중복인지 여부를 boolean Series로 반환하고, `sum()`을 사용하면 중복된 행의 개수를 알 수 있습니다.

```
duplicate_rows = df.duplicated().sum()
print(f"중복된 행의개수 : {duplicate_rows}")

# 만약중복행이있다면제거 :
# df = df.drop_duplicates()
```

Listing 19: 중복 행 확인

예제 데이터에는 중복 행이 없습니다.

4.2.6 결측치 확인 및 처리

데이터에 값이 없는 경우(NaN, NaT 등)를 결측치라고 합니다. `info()` 메서드로 컬럼별 non-null 개수를 확인하거나, `isna().sum()`으로 컬럼별 결측치 개수를 직접 확인할 수 있습니다.

```
print("--- 컬럼별결측치개수 ---")
print(df.isna().sum())
```

Listing 20: 컬럼별 결측치 개수 확인

결측치가 많은 컬럼('hw0', 'fav_movie', 'hobbies' 등)과 적은 컬럼이 있습니다.

결측치를 처리하는 방법은 여러 가지입니다.

- **제거:** `dropna()` 메서드를 사용하여 결측치가 포함된 행 또는 열을 제거합니다. 특정 컬럼에 결측치가 있는 행만 제거할 수도 있습니다(`subset` 인자 사용). 분석에 필수적인 정보가 없는 행을 제거할 때 사용합니다.
- **대치:** `fillna()` 메서드를 사용하여 결측치를 특정 값(예: 0, 평균값, 최빈값, 'Unknown' 등)으로 채웁니다. 데이터 손실을 최소화하고 싶을 때 사용합니다.

```
# 예시: 'hobbies' 컬럼에결측치가있는행을제거한결과확인원본 ( 변경 X)
df_dropped_hobbies = df.dropna(subset=['hobbies'])
print(f"원본 행개수 : {df.shape[0]}")
print(f"'hobbies' 결측치제거후행개수 : {df_dropped_hobbies.shape[0]}")

# 예시: 'languages' 컬럼에결측치가있는행을실제로제거원본 ( 변경 O)
print(f"\제거n 전 'languages' 결측치개
    수 : {df['languages'].isna().sum()}")
df.dropna(subset=['languages'], inplace=True) # inplace=는True 원본을직접
    수정
print(f"제거 후 'languages' 결측치개수 : {df['languages'].isna().sum()}")
```

```
print(f"최종 데이터크기 : {df.shape}")
```

Listing 21: 결측치 처리 예시 (제거)

주의사항

결측치 처리는 분석 결과에 큰 영향을 미칠 수 있으므로 신중하게 결정해야 합니다. 무조건 제거하거나 특정 값으로 채우기보다는, 데이터의 특성과 분석 목적을 고려하여 가장 적절한 방법을 선택해야 합니다. 경우에 따라서는 결측치 자체를 하나의 정보로 활용할 수도 있습니다.

4.3 3단계: 데이터 선택, 필터링, 정렬

정제된 데이터에서 원하는 부분만 선택하거나 특정 조건에 맞는 데이터를 필터링하고, 필요에 따라 정렬하는 방법을 알아봅니다.

4.3.1 Boolean Indexing (마스크 활용)

가장 강력하고 흔하게 사용되는 방법입니다. 특정 조건(예: 'pandas_skill' > 3)을 DataFrame이나 Series에 적용하면, 각 행(또는 원소)이 조건을 만족하는지 여부를 나타내는 True/False 값으로 이루어진 Series (이를 **마스크(mask)**라고 부름)가 반환됩니다. 이 마스크를 DataFrame의 인덱서([ ])에 넣어주면 조건이 True인 행들만 선택됩니다.

```
# 조건: Pandas 스킬레벨이보다 3 큰경우
mask = df['pandas_skill'] > 3

# 마스크를사용하여조건에맞는행들만선택
df_high_skill = df[mask]

# 결과확인 (pandas_skill 컬럼만출력 )
print(df_high_skill['pandas_skill'].head())

# 조건: dark 를mode 사용하고(True) 가OS 인MacOS 경우
# 여러조건은 & (AND), | (OR) 연산자로연결하며 , 각조건은괄호로묶어야함
mask_dark_mac = (df['dark_mode'] == True) & (df['os'] == 'macos')
df_dark_mac = df[mask_dark_mac]

# 결과확인 (os, dark_mode 컬럼및크기출력 )
print("\n--- Dark Mode 사용자 (MacOS) ---")
print(df_dark_mac[['os', 'dark_mode']].head())
print(f"해당 학생수 : {df_dark_mac.shape[0]}")
```

Listing 22: Boolean Indexing으로 필터링

4.3.2 loc 및 iloc 사용

loc과 iloc은 특정 행과 열을 선택하는 더 정교한 방법입니다.

- loc: **라벨(이름)** 기반 인덱싱. 행과 열의 이름(인덱스 라벨, 컬럼명)을 사용합니다. 슬라이싱 시 끝점을 포함합니다.
- iloc: **위치(정수)** 기반 인덱싱. 0부터 시작하는 행과 열의 순서(위치)를 사용합니다. 슬라이싱 시 끝점을 제외합니다 (파이썬 기본 슬라이싱과 동일).

```
# 예시 DataFrame 생성
data = {'A': [10, 20, 30, 40], 'B': [50, 60, 70, 80]}
index = ['r1', 'r2', 'r3', 'r4']
df_example = pd.DataFrame(data, index=index)
print("--- 예시 DataFrame ---")
print(df_example)
```

```
7
8  # loc 사용예시
9  print("\n--- loc 사용 ---")
10 # 행 'r2' 선택 (Series 반환)
11 print(f"행 'r2':\n{df_example.loc['r2']}")
12 # 행 'r1', 'r3'과' 열 'B' 선택 (DataFrame 반환)
13 print(f"\행n 'r1', 'r3', 열 'B':\n{df_example.loc[['r1', 'r3'], 'B']}")
14 # 행 'r2부터' 'r4까지', 열 'A' 선택 (Series 반환)
15 print(f"\행n 'r2':'r4', 열 'A':\n{df_example.loc['r2':'r4', 'A']}")
16
17 # iloc 사용예시
18 print("\n--- iloc 사용 ---")
19 # 첫번째행위치 ( 0) 선택 (Series 반환)
20 print(f"첫 번째행 (iloc[0]):\n{df_example.iloc[0]}")
21 # 첫번째 , 세번째행위치 ( 0, 2)과 두번째열위치 ( 1) 선택 (Series 반환)
22 print(f"\행n 0, 2, 열 1:\n{df_example.iloc[[0, 2], 1]}")
23 # 첫두행위치 ( 0, 1)과 모든열선택 (DataFrame 반환)
24 print(f"\처음n 두행 (iloc[0:2]):\n{df_example.iloc[0:2]}")
```

Listing 23: loc 및 iloc 사용 예시

주의사항

loc과 iloc의 차이를 명확히 이해하는 것이 중요합니다. 특히 정수 인덱스를 사용할 때 혼동하기 쉽습니다. df.loc[0]은 인덱스 라벨이 '0'인 행을 찾고, df.iloc[0]은 첫 번째 위치에 있는 행을 찾습니다. 데이터 정렬이나 필터링 후 인덱스가 순차적이지 않을 때 iloc이 유용할 수 있습니다.

4.3.3 데이터 정렬

특정 컬럼의 값을 기준으로 행을 정렬할 때는 sort_values() 메서드를 사용합니다.

```
1 # 'pandas_skill' 기준으로내림차순정렬 (ascending=False)
2 df_sorted_skill = df.sort_values(by='pandas_skill', ascending=False)
3 print(df_sorted_skill[['program', 'pandas_skill']].head())
4
5 # 여러컬럼기준으로정렬예 (: 'program' 오름차순, 'age' 내림차순)
6 # df_sorted_multi = df.sort_values(by=['program', 'age'], ascending=[
    True, False])
7 # print(df_sorted_multi[['program', 'age']].head())
```

Listing 24: 데이터 정렬

주의: 정렬 후에는 인덱스가 뒤섞이게 됩니다. 필요하다면 reset_index(drop=True)를 사용하여 인덱스를 0부터 다시 부여하는 것이 좋습니다.

4.4 4단계: 데이터 변환 및 집계

데이터를 분석하기 좋은 형태로 변환하거나 그룹별로 요약 통계를 계산합니다.

4.4.1 값 변환 (replace)

특정 값을 다른 값으로 바꿀 때 사용합니다. 정규 표현식(regex)을 사용하여 패턴 기반의 변환도 가능합니다.

```
# 'program' 컬럼에서 'certificate' 포함된문자열을하나로통일
df['program'] = df['program'].replace(r".*certificate.*",
                                      "graduate certificate [extension
                                         school]",
                                      regex=True)

# 변경확인 (value_counts)
print(df['program'].value_counts().head())
```

Listing 25: 값 변환 (replace)

4.4.2 문자열 분리 및 확장 (str.split, explode)

'languages', 'hobbies', 'fav_genres'처럼 콤마 등으로 구분된 여러 값이 하나의 문자열로 들어있는 경우, 각 값을 분리하여 분석하기 쉽게 만듭니다.

1. 문자열 분리 (str.split): 특정 구분자(예: ', ')를 기준으로 문자열을 나누어 리스트로 만듭니다.

2. 데이터 확장 (explode): 리스트가 들어있는 컬럼을 기준으로, 리스트의 각 원소가 별도의 행이 되도록 DataFrame을 확장합니다.

```
# 'languages' 컬럼을콤마와공백 (', ') 기준으로분리하여리스트생성
split_languages = df['languages'].str.split(', ')
print("--- 분리후 (Series of lists) ---")
print(split_languages.head())

# 'languages' 컬럼을기준으로 explode 수행원본 ( 변경 X)
df_exploded = df.explode('languages')
print("\n--- Explode 후 (languages 컬럼만확인 ) ---")
# 예: 첫번째학생 (index 0)이 여러행으로나타나는지확인
print(df_exploded[df_exploded.index == 0]['languages'])

# 전체언어목록확인중복 ( 제거및정규화포함 )
all_languages = df['languages'].str.split(', ').explode()
# 'mandarin' 관련표현통일예시 ()
all_languages = all_languages.replace(r'.*mandarin.*|madarin|(chinese$)'
    ,
                                      'chinese (mandarin)', regex=True).
                                         str.strip()
```

```
unique_languages = all_languages.unique()
print(f"\n--- 고유언어개수  : {len(unique_languages)} ---")
# print(sorted(list(unique_languages))) # 정렬하여출력생략  ()
```

Listing 26: 문자열 분리 및 확장

explode는 각 언어별 빈도수를 계산하거나, 특정 언어 사용자 그룹을 분석할 때 유용합니다.

4.4.3 함수 적용 (apply)

Series나 DataFrame의 각 원소 또는 행/열에 대해 사용자 정의 함수나 내장 함수를 적용할 때 사용합니다. 예를 들어, split된 리스트의 길이를 계산하여 각 학생이 구사하는 언어의 수를 계산할 수 있습니다.

```
# 'languages' 컬럼을분리한리스트의길이를계산하여  'num_languages' 컬럼생성
# 결측치(NaN)가 있는경우오류가발생할수있으므로  , fillna('') 등으로사전처리필요
# 여기서는 languages 결측치를이미제거했으므로바로적용
df['num_languages'] = df['languages'].str.split(', ').apply(len)

# 가장많은언어를구사하는경우확인
max_languages = df['num_languages'].max()
print(f"가장 많은언어구사수  : {max_languages}")
print(df.loc[df['num_languages'].idxmax(), ['languages', 'num_languages']]) # idxmax()는 최댓값의인덱스반환
```

Listing 27: apply를 이용한 계산

4.4.4 데이터 그룹화 및 집계 (groupby, agg)

특정 컬럼(들)의 값을 기준으로 데이터를 그룹화하고, 각 그룹에 대해 집계 함수(평균, 합계, 개수 등)를 적용하여 요약 통계를 계산합니다.

```
# 'program' 별평균  'pandas_skill' 계산
avg_skill_by_program = df.groupby('program')['pandas_skill'].mean()
print("--- 프로그램별평균  Pandas 스킬상위 ( 개5) ---")
print(avg_skill_by_program.sort_values(ascending=False).head())

# 여러집계함수를동시에적용예  (: 평균나이와학생수  )
# 'age' 컬럼이필요앞서  ( 로부터dob 계산했다고가정 )
if 'age' in df.columns:
    program_summary = df.groupby('program').agg(
        avg_age=('age', 'mean'),  # 'age' 컬럼에 mean 함수적용
        count=('program', 'size')  # 'program' 컬럼에 size 함수개수( 세기) 적용
    )
    # 학생수가명  2 이상인프로그램만필터링하고평균나이기준정렬
    program_summary_filtered = program_summary[program_summary['count'] >= 2].sort_values('avg_age')
```

```
    print("\n--- 주요프로그램별평균나이및학생수          ---")
    print(program_summary_filtered)
else:
    print("\n'age' 컬럼이없어프로그램별요약통계를계산할수없습니다          .")
```

Listing 28: groupby 및 agg 사용

4.4.5 교차 분석표 (crosstab)

두 범주형 변수 간의 관계를 파악하기 위해 각 조합에 해당하는 빈도수를 표 형태로 보여줍니다. 예를 들어, 운영체제(os)와 다크 모드(dark_mode) 선호도 간의 관계를 볼 수 있습니다.

```
# 'os'와 'dark_mode' 간의교차표생성
os_darkmode_crosstab = pd.crosstab(df['os'], df['dark_mode'])
print("--- 별OS Dark Mode 선호도교차표  ---")
print(os_darkmode_crosstab)

# 비율로보고싶다면   normalize 인자사용
# os_darkmode_crosstab_ratio = pd.crosstab(df['os'], df['dark_mode'],
    normalize='index')
# print("\n--- 별OS Dark Mode 선호도비율  ---")
# print(os_darkmode_crosstab_ratio)
```

Listing 29: crosstab 사용

4.5 5단계: 데이터 저장

정제 및 분석된 DataFrame을 나중에 다시 사용하기 위해 파일로 저장합니다. CSV 파일로 저장하는 것이 일반적이며, `to_csv()` 메서드를 사용합니다. `index=False` 옵션을 주면 DataFrame의 인덱스가 파일에 저장되지 않습니다.

```
# 정제된을 DataFrame 'survey_final.csv' 파일로저장인덱스 ( 제외)
try:
    df.to_csv('survey_final.csv', index=False)
    print("이DataFrame 'survey_final.csv' 파일로저장되었습니다 .")
except Exception as e:
    print(f"파일 저장중오류발생 : {e}")
```

Listing 30: DataFrame을 CSV 파일로 저장

5 실습 코드 종합 예제

다음은 위에서 설명한 주요 Pandas 작업을 연속적으로 보여주는 코드 예제입니다.

```
import pandas as pd
import numpy as np
from pandas.api.types import CategoricalDtype

# --- 1. 데이터로딩 ---
try:
    df = pd.read_csv("data/cs1090a_survey_raw.csv")
except FileNotFoundError:
    # 파일없을경우임시데이터사용위 ( 예제참고 )
    data = {'Timestamp': ['9/9/2025 17:43:09'], 'What\'s your Harvard
        affiliation?\n': ['Bachelor\'s (CS)'], 'Have you used Jupyter
        Notebooks before?': ['Yes'], 'How many years of Python
        programming experience do you have?\n(Choose the range that best
        fits your experience.)': ['Less than 1 year'], 'Rate your current
         Pandas skill level.': [2], 'What is your primary OS?': ['MacOS'
        ], 'Do use normally code/browse in dark mode?': ['No'], 'What
        languages do you speak? \n(Comma separated)\n\nExample: Hittite,
        Elvish, Cornish, Klingon': ['English'], 'Which continents have
        you visited?': ['Africa, Asia, Europe, North America, South
        America'], 'When were you born?': ['11/15/2004'], 'What time do
        you usually wake up in the morning?': ['10:00:00 AM'], 'What time
         do you usually go to bed?': ['12:00:00 AM'], 'Favorite Season?':
         ['Spring'], 'Where do you usually get your caffeine?': ['Tea'],
        'Which kind of pet do you prefer?': ['Pet rock'], 'What\'s your
        favorite movie?': [None], 'What movie genres do particularly
        enjoy?\n(select as many as you like)': ['Comedy'], 'List up to 3
        of your hobbies.\n(comma separated)\n\nExample: playing kazoo,
        bird watching, stamp collecting': ['tennis, movies, eating'], '
        How was HW0?': [None]}
    df = pd.DataFrame(data)

# --- 2. 데이터정제 ---
# 컬럼명변경
new_cols = ["timestamp", "program", "jupyter", "python_exp", "
    pandas_skill", "os", "dark_mode", "languages", "continents", "dob", "
    wake_time", "sleep_time", "fav_season", "caffeine", "pet", "fav_movie
    ", "fav_genres", "hobbies", "hw0"]
df.columns = new_cols

# 불필요한컬럼제거
df = df.drop("timestamp", axis=1)

# 데이터타입변환 (Boolean, Category, Datetime)
df['dark_mode'] = (df['dark_mode'].str.lower() == 'yes')
```

```
df['jupyter'] = (df['jupyter'].str.lower() == 'yes')
df['os'] = df['os'].astype('category')
experience_order = ['Less than 1 year', '1-2 years', '2-4 years', '4+
    years']
experience_dtype = CategoricalDtype(categories=experience_order, ordered
    =True)
df['python_experience'] = df['python_exp'].astype(experience_dtype)
df['dob'] = pd.to_datetime(df['dob'], errors='coerce')

# 텍스트정규화소문자 ( 변환)
str_cols = df.select_dtypes(include='object').columns
for c in str_cols:
    df[c] = df[c].str.lower().str.strip() # 공백제거추가

# 중복행제거있는 ( 경우)
df = df.drop_duplicates()

# 결측치처리예 (: 'languages가' 비어있는행제거 )
df.dropna(subset=['languages'], inplace=True)

# 파생변수생성예 (: 나이, 언어수 )
now = pd.Timestamp.now()
df['age'] = np.floor((now - df['dob']).dt.days / 365.25).astype('Int64')
df['num_languages'] = df['languages'].str.split(', ').apply(len)

# --- 3. 데이터분석예시 ---
# Pandas 스킬 4 이상인학생필터링
df_high_skill = df[df['pandas_skill'] >= 4]

# 프로그램별평균나이계산학생 ( 수명 2 이상)
program_summary = df.groupby('program').agg(avg_age=('age', 'mean'),
    count=('program', 'size'))
program_summary_filtered = program_summary[program_summary['count'] >=
    2].sort_values('avg_age')

# --- 4. 결과확인 ---
print("--- Pandas 스킬 4 이상학생일부 () ---")
print(df_high_skill[['program', 'pandas_skill']].head())

print("\n--- 주요프로그램별평균나이및학생수 ---")
print(program_summary_filtered)

# --- 5. 데이터저장 ---
# df.to_csv('survey_final.csv', index=False)
# print("\정제된n 데이터가 'survey_final.csv로' 저장되었습니다.")
```

Listing 31: Pandas 데이터 처리 파이프라인 예제

6 체크리스트

Pandas를 사용하여 데이터를 처리할 때 다음 사항들을 점검하면 실수를 줄이고 효율적인 분석을 수행하는 데 도움이 됩니다.

- ✓ **데이터 로딩:** 데이터가 올바르게 DataFrame으로 로드되었는가? (`head()`, `shape` 확인)
- ✓ **컬럼명 확인 및 변경:** 컬럼명이 너무 길거나 복잡하지 않은가? 의미 있고 사용하기 쉬운 이름으로 변경했는가? (소문자, 언더스코어 사용 권장)
- ✓ **불필요한 데이터 제거:** 분석에 사용하지 않을 행이나 열은 제거했는가? (`drop`)
- ✓ **데이터 타입 확인 및 변환:** 각 컬럼의 데이터 타입(`dtypes`, `info`)이 적절한가? 숫자여야 할 컬럼이 문자열(`object`)은 아닌가? 날짜, 범주형, boolean 등으로 변환이 필요한 컬럼은 없는가? (`astype`, `to_datetime`, `CategoricalDtype`)
- ✓ **텍스트 정규화:** 문자열 데이터에 대소문자나 앞뒤 공백 등 일관성 없는 부분이 있는가? (`str.lower()`, `str.strip()`, `replace`)
- ✓ **중복값 확인 및 처리:** 완전히 동일한 행이 중복되어 있는가? (`duplicated()`, `drop_duplicates`)
- ✓ **결측치 확인 및 처리:** 결측치(NaN)가 어디에 얼마나 있는지 파악했는가? (`isna().sum()`, `info`) 분석 목적에 맞게 결측치를 제거(`dropna`)하거나 적절한 값으로 대치(`fillna`)했는가?
- ✓ **인덱스 확인 및 재설정:** 데이터 필터링, 정렬 후 인덱스가 순차적이지 않게 되었는가? 필요하다면 인덱스를 재설정했는가? (`reset_index`)
- ✓ **데이터 선택/필터링 검증:** Boolean indexing, `loc`, `iloc` 등을 사용하여 원하는 데이터를 정확히 선택했는지 확인했는가?
- ✓ **집계 결과 검증:** `groupby`, `agg`, `crosstab` 등의 집계 결과가 예상과 일치하는가? 집계 함수를 올바르게 사용했는가?
- ✓ **결과 저장:** 최종적으로 정제되거나 분석된 데이터를 저장할 필요가 있는가? (`to_csv`)

7 FAQ (자주 묻는 질문)

Pandas를 처음 배울 때 흔히 궁금해하거나 혼동하는 부분들을 정리했습니다.

Q1: `df[column_name]` 과 `df.column_name` 중 어떤 것을 써야 하나요?

- `df['column_name']` (대괄호와 문자열 사용) 방식이 더 안전하고 권장됩니다. 컬럼 이름에 공백이나 특수문자가 있거나, DataFrame의 기존 메서드 이름(예: 'head', 'count')과 겹치는 경우에도 사용할 수 있습니다.
- `df.column_name` (점 표기법) 방식은 타이핑이 간편하지만, 위와 같은 제약 조건이 있습니다. 간단하고 이름 충돌 가능성이 없는 경우에만 사용하는 것이 좋습니다.

Q2: `loc`과 `iloc`은 언제 사용하나요? 왜 이렇게 복잡하게 나뉘어 있나요?

- 데이터를 선택하는 기준이 명확히 다릅니다. `loc`은 사람이 지정한 이름표(label)를 사용하고, `iloc`은 컴퓨터가 내부적으로 관리하는 순서(position, 0부터 시작하는 정수)를 사용합니다.
- 예를 들어, 학생 명단에서 '홍길동' 학생의 정보를 찾을 때는 이름표(`loc['홍길동']`)를 쓰는 것이 직관적입니다. 반면, 명단에서 그냥 '첫 번째 학생'의 정보를 볼 때는 위치(`iloc[0]`)를 쓰는 것이 편합니다.
- 데이터를 정렬하거나 필터링하면 원래의 순서와 이름표가 달라질 수 있기 때문에, 이 두 가지 방법을 명확히 구분하여 사용해야 원하는 데이터를 정확히 찾을 수 있습니다.

Q3: 데이터를 정렬하거나 필터링한 후에 왜 `reset_index()`를 자주 사용하나요?

- `sort_values()`나 boolean indexing으로 DataFrame을 조작하면, 기존의 인덱스 라벨은 그대로 유지된 채 행의 순서만 바뀝니다. 예를 들어, 원래 100번 인덱스였던 행이 정렬 후 첫 번째 행이 되어도 인덱스 라벨은 여전히 100입니다.
- 이렇게 되면 `iloc[0]` (첫 번째 위치의 행)과 `loc[0]` (인덱스 라벨이 0인 행)이 다른 행을 가리키게 되어 혼란을 야기할 수 있습니다.
- `reset_index(drop=True)`를 사용하면 현재 행 순서에 맞게 인덱스를 0부터 다시 깔끔하게 부여해주므로, 이후 작업에서 혼동을 줄일 수 있습니다. (`drop=False`로 하면 기존 인덱스가 새로운 컬럼으로 추가됩니다.)

Q4: 컬럼의 데이터 타입(`dtype`)이 왜 중요한가요? 그냥 `object`로 두면 안 되나요?

- **성능:** 숫자(`int`, `float`), boolean(`bool`) 등 명확한 타입은 메모리를 효율적으로 사용하고 계산 속도도 빠릅니다. 반면 `object` 타입은 다양한 종류의 데이터를 담을 수 있지만, 내부적으로는 메모리 주소를 저장하는 방식이라 처리 속도가 느리고 메모리도 많이 차지합니다.
- **기능:** 날짜/시간 타입(`datetime64`)으로 변환해야 날짜 관련 연산(예: 두 날짜 간의 차이 계산)이 가능합니다. 범주형 타입(`category`)은 메모리를 절약하고 특정 통계 분석에 유용합니다. 숫자 타입이어야 평균, 합계 등 수학적 연산이 가능합니다.
- **정확성:** '1', '2', '3'이 문자열(`object`)로 저장되어 있다면 숫자 크기 비교나 덧셈이 제대로 되지 않습니다. 반드시 적절한 숫자 타입으로 변환해야 합니다.

Q5: NumPy 배열과 Pandas Series/DataFrame은 무엇이 다른가요?

- **NumPy 배열 (`ndarray`):** 주로 숫자로 이루어진 다차원 배열을 효율적으로 다루기 위한 라이

브러리입니다. 모든 원소는 동일한 데이터 타입이어야 하며, 인덱스는 0부터 시작하는 정수 위치만 사용합니다. 수학/과학 계산에 강력합니다.

- **Pandas Series/DataFrame:** NumPy를 기반으로 만들어졌지만, 더 유연하고 다양한 기능을 제공합니다.
  - **명시적 인덱스:** 숫자가 아닌 라벨 기반 인덱스를 사용할 수 있습니다.
  - **다양한 데이터 타입:** DataFrame의 경우 열마다 다른 데이터 타입을 가질 수 있습니다.
  - **결측치 처리:** NaN 값을 내장하여 쉽게 처리할 수 있습니다.
  - **데이터 조작 기능:** 데이터 정렬, 필터링, 그룹화, 병합 등 데이터 분석에 특화된 고수준의 기능을 풍부하게 제공합니다.
  - **입출력 편의성:** CSV, Excel 등 다양한 파일 형식을 쉽게 다룰 수 있습니다.
- 요약하면, NumPy는 고성능 수치 계산의 기반이고, Pandas는 NumPy를 활용하여 표 형태의 데이터를 편리하게 분석할 수 있도록 확장한 도구입니다.

8 빠르게 훑어보기 (1페이지 요약)

Pandas 핵심 요약

Pandas? 파이썬에서 표(테이블) 데이터를 다루는 강력하고 편리한 라이브러리.

주요 자료구조

- **Series**: 1차원 배열 + 인덱스 (이름표). 엑셀의 한 열. `pd.Series(data, index=...)`
- **DataFrame**: 2차원 표. 여러 Series의 묶음. 엑셀 시트. `pd.DataFrame(data, columns=...)`

데이터 로딩 & 저장

- **읽기**: `pd.read_csv('파일경로')` (주로 사용), `pd.read_excel()`, ...
- **저장**: `df.to_csv('파일경로', index=False)`

기본 탐색

- `df.head(n)`: 처음 n개 행 보기
- `df.tail(n)`: 마지막 n개 행 보기
- `df.shape`: (행 개수, 열 개수) 확인
- `df.columns`: 열 이름 목록
- `df.index`: 행 이름(인덱스) 목록
- `df.info()`: 전체 요약 정보 (결측치, 타입 확인 필수!)
- `df.dtypes`: 열별 데이터 타입 확인
- `df.describe()`: 수치형 데이터 기술 통계 요약

데이터 정제

- **컬럼명 변경**: `df.columns = [...]`, `df.rename(columns=...)`
- **컬럼/행 제거**: `df.drop(['col1', 'row1'], axis=1/0)`
- **타입 변환**: `df['col'].astype('int')`, `pd.to_datetime(df['col'])`, ...
- **결측치 확인/처리**: `df.isna().sum()`, `df.dropna()`, `df.fillna(value)`
- **중복값 확인/처리**: `df.duplicated().sum()`, `df.drop_duplicates()`
- **텍스트 처리**: `df['col'].str.lower()`, `.str.strip()`, `.str.replace()`, `.str.split()`

데이터 선택 & 필터링

- **열 선택**: `df['col']`, `df[['col1', 'col2']]`
- **행 선택 (Boolean Indexing)**: `df[df['col'] > 10]`, `df[(cond1) & (cond2)]`
- **라벨 기반 선택 (loc)**: `df.loc['row_label', 'col_label']`, `df.loc['start':'end']` (끝 포함)
- **위치 기반 선택 (iloc)**: `df.iloc[0, 1]`, `df.iloc[0:5]` (끝 제외)

정렬 & 집계

- **정렬**: `df.sort_values(by='col', ascending=False)`
- **값 개수**: `df['col'].value_counts()`
- **그룹화**: `df.groupby('col')`
- **집계**: `grouped.agg('col1': 'mean', 'col2': 'count')`, `grouped.mean()`, `grouped.size()`

- **교차 분석**: `pd.crosstab(df['col1'], df['col2'])`

9 부록: 환경 설정 및 추가 정보

9.1 Python 환경 설정

Pandas를 사용하기 위해서는 Python과 Pandas 라이브러리가 설치되어 있어야 합니다. 강의에서는 가상 환경 사용을 권장하며, `uv` 또는 `conda`와 같은 도구를 사용할 수 있습니다.

uv 사용 (권장): 과제 파일과 함께 제공되는 `requirements.txt` 파일을 이용하여 필요한 라이브러리가 모두 포함된 가상 환경을 생성할 수 있습니다.

1. `uv` 설치 (pip install uv)
2. 프로젝트 폴더에서 `uv venv` 실행하여 가상 환경 생성 (.venv 폴더 생성됨)
3. `source .venv/bin/activate` (macOS/Linux) 또는 `.venv` (Windows) 실행하여 가상 환경 활성화
4. `uv pip install -r requirements.txt` 실행하여 라이브러리 설치

각 과제(프로젝트)마다 별도의 가상 환경을 만드는 것이 좋습니다.

Conda 사용: Conda 환경을 사용하고 있다면, `conda install pandas` 명령어로 Pandas를 설치하거나, `requirements.txt` 파일을 이용하여 환경을 구성할 수 있습니다.

직접 설치: 기존 환경에 직접 설치하려면 `pip install pandas` 또는 `uv pip install pandas` 명령어를 사용합니다. 과제 진행 중 필요한 라이브러리가 없다는 오류가 발생하면 해당 라이브러리를 추가로 설치해주어야 합니다.

9.2 Harvard OnDemand (클라우드 환경)

로컬 환경 설정에 어려움을 겪는 경우, Harvard OnDemand에서 제공하는 클라우드 기반 JupyterLab 환경을 사용할 수 있습니다.

- Canvas의 'Harvard OnDemand' 링크를 통해 접속합니다.
- 'Interactive Apps' 메뉴에서 'JupyterLab CS1090A'를 선택합니다.
- 사용 시간 등을 설정하고 'Launch' 버튼을 클릭합니다.
- 잠시 후 생성된 환경에 접속하여 Jupyter 노트북을 사용합니다.
- 과제 파일이나 강의 자료(zip 파일)를 업로드하여 작업할 수 있습니다.
- 사용 시간이 만료되어도 작업 내용은 유지되므로, 다시 접속하여 이어서 작업할 수 있습니다.

주의사항

Harvard OnDemand는 제한된 자원이므로, 로컬 환경 설정이 가능한 경우에는 로컬 환경 사용을 권장합니다. 꼭 필요한 경우의 안전망으로 활용하세요.

9.3 Jupyter Notebook 사용 팁

- **Tab 자동 완성:** 코드 입력 중 Tab 키를 누르면 사용 가능한 변수, 함수, 메서드 목록을 보여주거나 자동 완성해줍니다. (`df.` 입력 후 Tab)
- **Shift + Tab 도움말:** 함수 괄호 안에서 Shift + Tab 키를 누르면 해당 함수의 설명(docstring)을 보여줍니다. 인자나 사용법을 확인할 때 유용합니다.
- **셀 실행 순서 주의:** 노트북 셀은 원하는 순서대로 실행할 수 있지만, 이로 인해 변수 상태가 꼬여 예상치 못한 오류가 발생할 수 있습니다. 문제가 발생하면 커널을 재시작(Kernel > Restart)하고 처음부터 순서대로 실행해보는 것이 좋습니다.

9.4 추가 학습 자료

- **Pandas 공식 문서:** https://pandas.pydata.org/docs/ (가장 정확하고 방대한 정보 제공)
- **Pandas 10분 완성:** https://pandas.pydata.org/docs/user_guide/10min.html (빠른 시작 가이드)
- **강의 플랫폼(Ed) 자료:** Bedrock Data Science, Bedrock Codecasts & Tutorials 섹션의 추가 Pandas 자료 확인