데이터 입출력

CSV

- 대부분의 정형 데이터가 저장되고 공유되는 표준 형식
- 쉼표로 구분된 값(Comma-Separated Values) 형식의 텍스트 파일
- **주의사항**
  - 파일 경로, 인코딩 문제, 구분자 불일치로 인해 에러가 발생 가능
  - 대용량 파일의 경우 `chunksize` 인자를 사용하여 단계적으로 처리하는 것이 좋음
- **코드 예시**

```
import pandas as pd
import io

csv_data = "col1,col2,col3\n1,a,True\n2,b,False\n3,c,True"
df_csv = pd.read_csv(io.StringIO(csv_data))
print(df_csv)
```

- **결과 해석**

```
   col1 col2   col3
0     1    a   True
1     2    b  False
2     3    c   True
```

- **주요 하이퍼파라미터 (인자)**
  - `filepath_or_buffer`: 파일 경로, URL 또는 파일과 유사한 객체.
  - `sep`: 필드 구분자. 기본값은 `,`.
  - `header`: 헤더로 사용할 행의 번호. 기본값은 `0`. 헤더가 없으면 `None`.
  - `encoding`: 파일 인코딩 형식. (e.g., `'utf-8'`, `'cp949'`)
  - `usecols`: 읽어올 열의 이름이나 인덱스 리스트.
  - `nrows`: 읽어올 파일의 행 수.

Excel

- Microsoft Excel 파일(`.xls`, `.xlsx`)을 읽어 DataFrame으로 변환하여 사용
- 비즈니스 환경의 보고서나 데이터 분석 시 사용
- **주의사항**:
  - `read_excel` 사용을 위해 `openpyxl` 라이브러리가 설치 필요
  - 최신 `.xlsx` 파일을 다루기 위해 `openpyxl` 사용을 권장
- **코드 예시**

```
```

```
import pandas as pd
import io

# 예제 DataFrame 생성
source_df = pd.DataFrame({
  "제품명": ["노트북", "마우스", "키보드"],
  "가격": [1200000, 50000, 75000],
  "재고": [15, 120, 80]
})

# 가상의 엑셀 파일을 메모리(BytesIO 버퍼)에 저장
# 실제 파일로 저장 시: source_df.to_excel("my_data.xlsx", index=False,
sheet_name="재고 현황")
buffer = io.BytesIO()
source_df.to_excel(buffer, index=False, sheet_name="재고 현황")
buffer.seek(0) # 스트림 포인터를 처음으로 이동

# pd.read_excel()로 데이터 읽기
# 실제 파일에서 읽을 시: pd.read_excel("my_data.xlsx", sheet_name="재고 현황")
df_from_excel = pd.read_excel(buffer, sheet_name="재고 현황")

print(df_from_excel)
```

- **주요 하이퍼파라미터 (인자)**
  - `io`: 파일 경로, URL 또는 Excel 파일 객체.
  - `sheet_name`: 읽어올 시트의 이름(문자열)이나 번호(0부터 시작). 기본값은 `0` (첫 번째 시트).
  - `header`: 헤더로 사용할 행의 번호. 기본값은 `0`.
  - `usecols`: 읽어올 열의 범위 (e.g., `'A:C'`) 또는 리스트.
  - `engine`: 사용할 파서 엔진. `.xlsx`는 `'openpyxl'`, 구형 `.xls`는 `'xlrd'`가 사용됩니다.

TXT

- 로그 파일이나 정형화되지 않은 텍스트 데이터를 읽을 때 사용
- 일반 텍스트 파일(로그, 비정형 텍스트 등)을 읽음
- Python 내장 `open()` 함수를 사용하는 것이 일반적
- **주의사항**
  - `encoding`을 명확히 지정해야 한글 깨짐을 방지
  - 대용량 파일은 한 번에 `read()`로 읽기보다 한 줄씩 순회하며 읽는 것이 메모리 효율적
- **코드 예시**

```
# 파일을 한 줄씩 순회하며 읽기 (메모리 효율적)
lines = []
with open('file.txt', 'r', encoding='utf-8') as f:
  for line in f:
      lines.append(line.strip()) # 줄바꿈 문자 제거
print(lines)
```

- **주요 인자 (`open` 함수)**

    - `file`: 파일 경로.
    - `mode`: 파일 열기 모드. `'r'`(읽기), `'w'`(쓰기), `'a'`(추가).
    - `encoding`: 파일 인코딩.

JSON

- 웹 API 응답 데이터나, 계층 구조를 가진 설정 파일을 다룰 때 사용
- **주의사항**
    - JSON 데이터의 구조에 따라 `orient` 인자를 설정 필요
    - `lines=True`는 각 줄이 유효한 JSON 객체일 때 사용
- **pandas 코드 예시**

```
import pandas as pd
import io

# 레코드 리스트 형태의 JSON
json_data = '[{"col1": 1, "col2": "a"}, {"col1": 2, "col2": "b"}]'
df_json = pd.read_json(io.StringIO(json_data), orient='records')
print(df_json)
```

- **주요 하이퍼파라미터 (인자)**
    - `path_or_buf`: 파일 경로 또는 JSON 문자열.
    - `orient`: JSON 문자열의 형식. (`'records'`, `'columns'`, `'index'`, `'split'`, `'values'`) 데이터 구조에 맞게 지정해야 합니다.
    - `lines`: 파일의 각 줄을 JSON 객체로 읽을지 여부. 기본값은 `False`.
    - `encoding`: 파일 인코딩.
- **json 코드 예시**

```
# json 모듈로 읽어들이기 (딕셔너리 형태)
import json

with open('test.json', 'r') as f:
    json_data = json.load(f)
print(json.dumps(json_data))
# 들여쓰기 옵션
print(json.dumps(json_data, indent="\t"))
```

XML

- 복잡한 계층 구조와 메타데이터를 가진 레거시 시스템의 데이터를 다룰 때 사용

방법 1: `pandas.read_xml` 사용

- **용도**: 테이블 형태의 간단한 XML을 DataFrame으로 빠르게 변환할 때 사용
- **주의사항**
    - `lxml` 라이브러리 설치가 필요 (`pip install lxml`)

    - `xpath` 인자를 사용하여 DataFrame으로 변환할 노드를 정확히 지정 필요
- **코드 예시**

```
import pandas as pd
import io

xml_data = """
<data>
    <person>
        <name>David</name>
        <age>25</age>
    </person>
    <person>
        <name>Jane</name>
        <age>30</age>
    </person>
</data>
"""
df_xml = pd.read_xml(io.StringIO(xml_data), xpath='.//person')
print(df_xml)
```

- **주요 하이퍼파라미터 (인자)**
    - `path_or_buf`: 파일 경로 또는 XML 문자열.
    - `xpath`: 파싱할 자식 노드를 선택하기 위한 XPath. 기본값은 `./*`.
    - `parser`: 사용할 파서. 기본값은 `'lxml'`.

방법 2: `xml.etree.ElementTree` 사용

- **용도**
    - Python 표준 라이브러리만으로 XML을 파싱해야 할 때 사용
    - `read_xml`로 처리하기 어려운 복잡한 구조의 XML에서 데이터를 섬세하게 추출해야 할 때 사용
- **주의사항**
    - XML 구조를 순회(iterate)하면서 필요한 데이터를 직접 추출하고,
      리스트나 딕셔너리 형태로 가공한 후, DataFrame으로 변환해야 함
- **코드 예시**

```
import xml.etree.ElementTree as ET
import pandas as pd
import io

xml_data = """
<data>
    <person>
        <name>David</name>
        <age>25</age>
    </person>
    <person>
        <name>Jane</name>
        <age>30</age>
```

```
    </person>
</data>
"""

root = ET.fromstring(xml_data) # 문자열로부터 파싱. 파일은
ET.parse('file.xml')

rows = []
# 모든 'person' 태그를 순회
for person in root.findall('person'):
    name = person.find('name').text
    age = person.find('age').text
    rows.append({'name': name, 'age': int(age)})

df_et = pd.DataFrame(rows)
print(df_et)
```

to_csv

- DataFrame을 CSV(쉼표로 구분된 값) 파일로 저장
- **주의사항**
  - 기본적으로 행 인덱스가 파일에 포함됨
    - 인덱스를 제외하고 싶다면 `index=False` 옵션 사용
  - 한글 데이터가 포함된 경우 `encoding`을 `'utf-8-sig'`로 지정해야 Excel에서 파일을 열 때 글자가 깨지지 않음
- **코드 예시**

```
import pandas as pd

df = pd.DataFrame({'Name': ['Alice', '이순신'], 'Age': [25, 30]})

# 파일로 저장 (인덱스 제외, utf-8-sig 인코딩)
# df.to_csv('output.csv', index=False, encoding='utf-8-sig')

# 문자열로 결과 확인
csv_output = df.to_csv(index=False)
print(csv_output)
```

- **주요 하이퍼파라미터 (인자)**
  - `path_or_buf`: 저장할 파일 경로 또는 버퍼. 지정하지 않으면 문자열로 결과를 반환합니다.
  - `sep`: 사용할 필드 구분자. 기본값은 `,`.
  - `na_rep`: 결측치(NaN)를 나타낼 문자열. 기본값은 빈 문자열.
  - `columns`: 파일에 쓸 열을 선택하는 리스트.
  - `header`: 열 이름을 쓸지 여부. 기본값은 `True`.
  - `index`: 행 인덱스를 쓸지 여부. 기본값은 `True`이며, 보통 `False`로 설정하여 불필요한 열 생성을 방지합니다.
  - `encoding`: 파일 인코딩 형식. 한글 데이터를 저장할 경우 `'utf-8-sig'`를 권장합니다.
  - `mode`: 파일 쓰기 모드. 기본값은 `'w'`(덮어쓰기), `'a'`는 이어쓰기입니다.

to_excel

- DataFrame을 Excel 파일(`.xlsx`)로 저장
- **주의사항**
  - `openpyxl` 라이브러리 설치 필요
  - `index=False` 옵션을 사용하여 불필요한 인덱스 저장을 방지
  - 여러 DataFrame을 한 파일의 다른 시트에 저장 가능
- **코드 예시**

```
import pandas as pd

df1 = pd.DataFrame({'Data1': [1, 2, 3]})
df2 = pd.DataFrame({'Data2': [4, 5, 6]})

# 단일 시트에 저장
# df1.to_excel("output_single.xlsx", index=False, sheet_name="MyData")

# 여러 시트에 저장
with pd.ExcelWriter("output_multiple.xlsx") as writer:
  df1.to_excel(writer, index=False, sheet_name="Sheet_A")
  df2.to_excel(writer, index=False, sheet_name="Sheet_B")

print("Excel 파일 저장이 완료되었습니다.")
```

- **주요 하이퍼파라미터 (인자)**
  - `excel_writer`: 저장할 파일 경로 또는 `ExcelWriter` 객체.
  - `sheet_name`: 시트 이름. 기본값은 `'Sheet1'`.
  - `na_rep`: 결측치(NaN)를 나타낼 문자열.
  - `columns`: 파일에 쓸 열을 선택하는 리스트.
  - `header`: 열 이름을 쓸지 여부.
  - `index`: 행 인덱스를 쓸지 여부. 기본값은 `True`.
  - `engine`: 사용할 엔진. (e.g., `'openpyxl'`)
  - `startrow`, `startcol`: 데이터 쓰기를 시작할 좌상단 셀의 위치 (0-based).