Python 데이터 분석 라이브러리 활용

목차

파일 다루기

파일 읽기

- 파이썬에서는 텍스트 파일을 다루기 위해 open( ) 함수를 사용한다.

```
f = open("파일명", "파일 열기 모드")
f.close()
```

종류	설명
r	읽기 모드: 파일을 읽기만 할 때 사용
w	쓰기 모드: 파일에 내용을 쓸 때 사용
a	추가 모드: 파일의 마지막에 새로운 내용을 추가할 때 사용

```
1 f = open("dream.txt", "r")
2 contents = f.read()
3 print(contents)
4 f.close()
```

파일 다루기

■ 파일 읽기 : with문과 함께 사용하기

- 하나의 파이썬 프로그램이 하나의 파일을 쓰고 있을 때 사용을 완료하면 반드시 해당 파일을 종료해야 한다.
- with문과 함께 open( ) 함수를 사용할 수 있다. with문은 들여쓰기를 사용해 들여쓰기가 있는 코드에서는 open( ) 함수가 유지되고, 들여쓰기가 종료되면 open( ) 함수도 끝나는 방식이다.

```
1 with open("dream.txt","r") as my_file:
2     contents = my_file.read()
3     print(type(contents), contents)
```

파일 다루기

■ 파일 읽기 : 한 줄씩 읽어 리스트형으로 반환하기

- 파일 전체의 텍스트를 문자열로 반환하는 read( ) 함수 대신, readlines( ) 함수를 사용하여 한 줄씩 내용을 읽어 와 문자열 형태로 저장할 수 있다.

```
1 with open("dream.txt","r") as my_file:
2     content_list = my_file.readlines()          # 파일 전체를 리스트로 반환
3     print(type(content_list))                   # 자료형 확인
4     print(content_list)                         # 리스트값 출력
```

파일 다루기

■ 파일 읽기 : 실행할 때마다 한 줄씩 읽어 오기

- readline( ) 함수는 실행할 때마다 차례대로 한 줄 씩 읽어오는 함수이다.

```
1 with open("dream.txt", "r") as my_file:
2     i = 0
3     while 1:
4         line = my_file.readline()
5         if not line:
6             break
7         print(str(i)+" === "+ line.replace("\n",""))     # 한 줄씩 값 출력
8         i = i + 1
```

파일 다루기

■ 파일 읽기 : 파일 안 글자의 통계 정보 출력하기

- 때로는 파일 안 텍스트의 통계 정보를 읽어 와야 할 때가 있다. 이를 위해 split( ) 함수와 len( ) 함수를 함께 사용하는 것이다.

```
1 with open("dream.txt", "r") as my_file:
2     contents = my_file.read()
3     word_list = contents.split(" ")          # 빈칸 기준으로 단어를 분리 리스트
4     line_list = contents.split("\n")         # 한 줄씩 분리하여 리스트
5
6 print("총 글자의 수:", len(contents))
7 print("총 단어의 수:", len(word_list))
8 print("총 줄의 수:", len(line_list))
```

파일 다루기

파일 쓰기

- 텍스트 파일을 저장하기 위해서는 텍스트 파일을 저장할 때 사용하는 표준을 지정해야 하는데, 이것을 인코딩(encoding)이라고 한다.

```
1 f = open("count_log.txt", 'w', encoding = "utf8")
2 for i in range(1,11):
3     data = "%d번째 줄이다.\n"% i
4     f.write(data)
5 f.close()
```

파일 다루기

■ 파일 쓰기 : 파일 열기 모드 a로 새로운 글 추가하기

- 상황에 따라 파일을 계속 추가해야 하는 작업이 있을 수도 있으므로, 모드a 를 사용하면 기존 파일에 내용을 추가할 수 있다

```
1 with open("count_log.txt", 'a', encoding = "utf8") as f:
2     for i in range(1, 11):
3         data = "%d번째 줄이다.\n"% i
4         f.write(data)
```

CSV 파일 로드, 저장

- pandas library의 read_csv() 함수를 사용해서 외부 text 파일, csv 파일을 불러와서 DataFrame으로 저장
  - csv 파일은 구분자(separator, delimiter) - ',' (comma)

```
import pandas as pd
csv_test = pd.read_csv('C:/Users/Administrator/Documents/Python/test_csv_file.csv')

# DataFrame.shape 을 사용해서 행(row)과 열(column)의 개수 확인
csv_test.shape # number of rows, columns
# 데이터 확인
print(csv_test )
```

  - 구분자가 콤마(,)가 아닌 다른 기호, 수직 막대기 '|' 인 경우의 text 파일 로드
  - 구분자(separator, delimiter)에 sep='|' 를 추가

```
text_test = pd.read_csv('C:/Users/Administrator/Documents/Python/test_text_file.txt', sep='|')
print(text_test)
```

CSV 파일 로드, 저장

- pandas library의 read_csv()
  - 파일 불러올 때 index 지정해주기 : index_col
  - 첫번째 열인 'ID'라는 이름의 변수를 Index 로 지정해주고 싶으면 index_col=0 (위치)이나 index_col='ID' 처럼 직접 변수 이름을 지정

```
import pandas as pd
csv_test = pd.read_csv('C:/Users/Administrator/Documents/Python/test_text_file.txt', sep='|', index_col=0)
print(csv_test )

csv_test2 =pd.read_csv('C:/Users/Administrator/Documents/Python/test_text_file.txt', sep='|', index_col='ID')
print(csv_test2 )
```

  - 변수 이름(column name, header) 이 없는 파일 로드할 때 이름 부여 : names=['X1', 'X2', ...]
  - header=None 은 칼럼 이름이 없다의 의미
  - 1번째 행이 칼럼 이름이라면 header=0 으로 지정

```
text_test = pd.read_csv('C:/Users/Administrator/Documents/Python/text_without_column_name.txt', sep='|',
names=['ID', 'A', 'B', 'C', 'D'], header=None, index_col='ID')
print(text_test)
```

CSV 파일 로드, 저장

- pandas library의 read_csv()
  - 유니코드 디코드 에러, UnicodeDecodeError: 'utf-8' codec can't decode byte

```
import pandas as pd
# Windows에서 많이 사용하는 'CP949'로 아래처럼 encoding을 설정
f = pd.read_csv('directory/file', sep='|'', encoding='CP949')
print(f )
# encoding='latin' ('ISO-8859-1' 의 alias) 로 설정
csv_test2 =pd.read_csv('directory/file', sep='|'', encoding='latin')
print(csv_test2 )
```

  - 특정 줄은 제외하고 파일 내용 로드 : skiprows = [x, x]

```
text_test = pd.read_csv("C:/Users/admin/Documents/data/test_csv_file.csv",  skiprows = [1, 2] )
print(text_test)
```

CSV 파일 로드, 저장

- pandas library의 read_csv()
  - n 개의 행 데이터만 로드 : nrows = n

```
import pandas as pd
csv_test = pd.read_csv("C:/Users/admin/Documents/data/test_csv_file.csv", nrows = 3 )
print(csv_test )
```

  - 사용자 정의 결측값 기호를 pandas가 결측값으로 인식할 수 있도록 설정하는 옵션 na_values

```
text_test = pd.read_csv('C:/Users/Administrator/Documents/Python/test_text_file.txt',
                        na_values = ['?', '??', 'N/A', 'NA', 'nan', 'NaN', '-nan', '-NaN', 'null')
print(text_test)
```

CSV 파일 로드, 저장

- pandas library의 read_csv()
  - pandas는 데이터셋을 읽어들일 때 첫번째 행의 데이터를 기준으로 각 칼럼별 데이터 유형을 추정해서 자동으로 설정
  - 데이터 유형 설정 (Setting the data type per each column) dtpye 옵션으로 사전형(dictionary)으로 각 칼럼(key) 별 데이터 유형(value)를 짝을 지어서 명시적으로 설정

```
import pandas as pd
csv_test = pd.read_csv('C:/Users/Administrator/Documents/Python/test_text_file.txt',
                       dtype = {"ID": int,  "LAST_NAME": str,  "AGE": float} )
print(csv_test )
```

  - 날짜/시간 형태(date/time format)의 데이터의 경우 infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates 등의 시계열 데이터 형태에 특화된 옵션 제공
  - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

JSON 데이터 읽고 쓰기

- JSON (JavaScript Object Notation)
  - XML, YAML 과 함께 효율적으로 데이터를 저장하고 교환(exchange data)하는데 사용하는 텍스트 데이터 포맷
  - 사람이 읽고 쓰기에 쉬우며,
  - 기계가 파싱하고 생성하기에도 용이
  - JavaScript의 프로그래밍 언어의 부분에 기반하고 있으며,
  - C-family 프로그램밍 언어 (C, C++, C#, Java, JavaScript, Perl, Python 등)의 규약을 따르고 있어서 C-family 프로그래밍 언어 간 데이터를 교환하는데 적합

JSON 데이터 읽고 쓰기

- JSON 구조
  - 이름/값 쌍의 집합 (A collection of name/value pairs): object, record, struct, dictionary, hash table, keyed list, associative array
  - 정렬된 값의 리스트 (An ordered list of values): array, vector, list, sequence

```
{
    "1.FirstName": "Gildong",
    "2.LastName": "Hong",
    "3.Age": 20,
    "4.University": "Yonsei University",
    "5.Courses": [
        {
            "Classes": [
                "Probability",
                "Generalized Linear Model",
                "Categorical Data Analysis"
            ],
            "Major": "Statistics"
        },
        {
            "Classes": [
                "Data Structure",
                "Programming",
                "Algorithms"
            ],
            "Minor": "ComputerScience"
        }
    ]
}
```

JSON 데이터 읽고 쓰기

- Python 객체를 JSON 데이터로 쓰기, 직렬화, 인코딩: json.dumps()
- JSON 포맷 데이터를 Python 객체로 읽기, 역직렬화, 디코딩: json.loads()
  - 파이썬의 내장 json 모듈이 필요

from Python ⟶	to JSON
dict	object
list, tuple	array
str	string
int, long, float	number
True	true
False	false
None	null

from JSON ⟶	to Python
object	dict
array	list
string	str
number(int)	int
number(real)	float
true	True
false	False
null	None

JSON 데이터 읽고 쓰기

- Python 객체를 JSON 데이터로 쓰기, 직렬화, 인코딩: json.dumps()
  - with open(): json.dump() 를 사용해서 JSON 포맷 데이터를 디스크에 쓰기

```
student_data = {
    "1.FirstName": "Gildong",
    "2.LastName": "Hong",
    "3.Age": 20,
    "4.University": "Yonsei University",
    "5.Courses": [
        {
            "Major": "Statistics",
            "Classes": ["Probability",
                        "Generalized Linear Model",
                        "Categorical Data Analysis"]
        },
        {
            "Minor": "ComputerScience",
            "Classes": ["Data Structure",
                        "Programming",
                        "Algorithms"]
        }
    ]
}
```

```
import json

with open("student_file.json", "w") as json_file:

    json.dump(student_data, json_file)
```

JSON 데이터 읽고 쓰기

➢ Python 객체를 JSON 데이터로 쓰기, 직렬화, 인코딩: json.dumps()

- json.dumps()를 사용해서 JSON 포맷 데이터를 메모리에 만들기

```
import json

st_json = json.dumps(student_data)
print(st_json)
```

- indent = int' 로 들여쓰기(indentation) 옵션 : json.dumps()로 파이썬 객체를 직렬화해서 JSON으로 쓸 때 사람이 보기에 좀더 쉽도록 설정

```
import json

st_json2 = json.dumps(student_data, indent=4)
print(st_json2)
```

- 'sort_keys=True' : 키(keys)를 기준으로 정렬해서 직렬화

```
import json

st_json3 = json.dumps(student_data, indent=4, sort_keys=True)
print(st_json3)
```

JSON 데이터 읽고 쓰기

➢ JSON 포맷 데이터를 Python 객체로 읽기, 역직렬화, 디코딩: json.loads()

- 디스크에 있는 JSON 포맷 데이터를 json.load()를 사용하여 Python 객체로 읽어오기

```
import json

with open("student_file.json", "r") as st_json:
    st_python = json.load(st_json)

print(st_python)
```

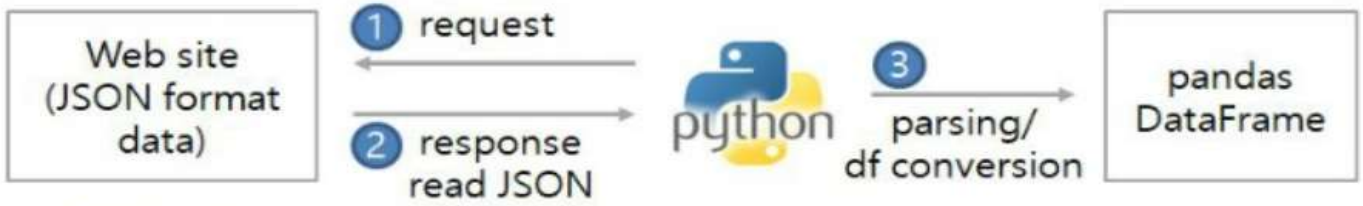
- 메모리에 있는 JSON 포맷 데이터를 json.loads()로 Python 객체로 읽기

```
import json

st_python2 = json.loads(st_json3)
print(st_python2)
```

JSON 데이터 읽고 쓰기

➢ 웹으로 부터 JSON 포맷 데이터 읽어와서 pandas DataFrame으로 만들기




```
import json
import urllib
import pandas as pd
import sys

1 if sys.version_info[0] == 3:
      from urllib.request import urlopen # for Python 3.x
  else:
      from urllib import urlopen # for Python 2.x

  with urlopen("http://api.abc/def.json") as url:
      json_file = url.read()

2 py_json = json.loads(json_file.decode('utf-8'))

3 py_json_df = pd.DataFrame(py_json["key"][0]["key2"],
                            columns = ['col1', 'col2', 'col3', 'col4'])
```

JSON 데이터 읽고 쓰기

➢ 웹으로부터 JSON 포맷 데이터 읽어와서 pandas DataFrame으로 만들기

- "Awesome JSON Datasets" (https://awesomerank.github.io/lists/jdorfman/awesome-json-datasets.html)
- 'Novel Prize' JSON 포맷 데이터(http://api.nobelprize.org/v1/prize.json)를 읽어와서 DataFrame으로 생성
- urllib 모듈의 rulopen 함수를 사용
- JSON 데이터가 있는 URL로 요청(request)을 보내서 URL을 열고 JSON 데이터를 읽어와서, python의 json.loads() 를 사용하여 novel_prize_json 이라 이름의 Python 객체로 생성

```
import json
import urllib
import pandas as pd
import sys

if sys.version_info[0] == 3:
    from urllib.request import urlopen # for Python 3.x
else:
    from urllib import urlopen          # for Python 2.x

with urlopen("http://api.nobelprize.org/v1/prize.json") as url:
    novel_prize_json_file = url.read()
# 읽어온 JSON 포맷 데이터를 Python의 json.loads() 메소드를 이용해서 decoding
novel_prize_json = json.loads(novel_prize_json_file.decode('utf-8'))
```

JSON 데이터 읽고 쓰기

➢ 웹으로부터 JSON 포맷 데이터 읽어와서 pandas DataFrame으로 만들기

```
#keys() 메소드로 키를 확인
novel_prize_json.keys()
novel_prize_json['prizes'][0].keys()
novel_prize_json['prizes'][0]

print(json.dumps(novel_prize_json['prizes'][0], indent=4))

#키(keys)를 기준으로 정렬
print(json.dumps(novel_prize_json['prizes'][0], indent=4, sort_keys=True))
```

- JSON 포맷의 데이터 중의 일부분을 indexing하여 pandas DataFrame으로 생성

```
#['prizes'][0] 은 물리학(physics) 노벨상이며, ['prizes'][0]['laureates'] 로 물리학 노벨상 수상자 정보만 선별해서 DataFrame으로 생성
novel_prize_physics = pd.DataFrame(novel_prize_json['prizes'][0]["laureates"])
print(novel_prize_physics)
#DataFrame을 만들 때 칼럼 순서를 columns 로 지정
novel_prize_physics = pd.DataFrame(novel_prize_json['prizes'][0]["laureates"],
                                   columns = ['id', 'firstname', 'surname', 'share', 'motivation'])
print(novel_prize_physics)
```

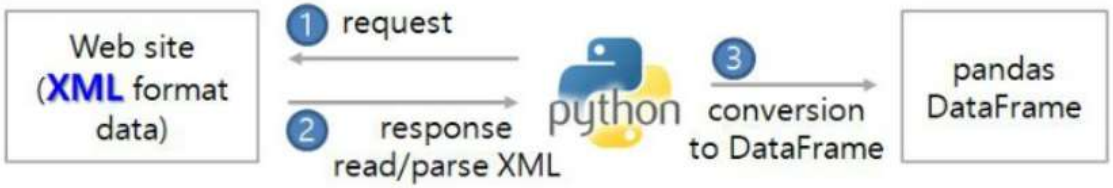
XML 데이터 읽고 쓰기

- XML (Extensible Markup Language)
  - 인간과 기계가 모두 읽을 수 있는 형태로 문서를 인코딩하는 규칙의 집합을 정의하는 마크업 언어(Markup Language)
  - 다양한 인간 언어들을 유니코드를 통해 강력하게 지원하는 텍스트 데이터 포맷
  - XML의 설계 목적 : 단순성, 범용성, 인터넷에서의 활용성
  - XML의 설계가 문서에 중점을 두고는 있지만, XML은 임의의 데이터 구조를 띠는 웹 서비스와 같은 용도의 재표현을 위한 용도로 광범위하게 사용되고 있음

```
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
<CD>
<TITLE>Unchain my heart</TITLE>
<ARTIST>Joe Cocker</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>EMI</COMPANY>
<PRICE>8.20</PRICE>
<YEAR>1987</YEAR>
</CD>
</CATALOG>
```

XML 데이터 읽고 쓰기

➢ 웹에서 XML 데이터를 읽어와서 pandas DataFrame으로 생성




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

1
```
import sys
if sys.version_info[0] == 3:
    from urllib.request import urlopen # for Python 3.x
else:
    from urllib import urlopen # for Python 2.x
url = "https://www.w3schools.com/xml/cd_catalog.xml"
response = urlopen(url).read()
xtree = ET.fromstring(response)
```

2
```
rows = []
for node in xtree:
    n_title = node.find("TITLE").text
    n_artist = node.find("ARTIST").text
    rows.append({"title": n_title,
                 "artist": n_artist})
```

3
```
catalog_cd_df = pd.DataFrame(rows, columns = ["title", "artist"])
```

XML 데이터 읽고 쓰기

- 웹에서 XML 데이터를 읽어와서 pandas DataFrame으로생성
  - xml.etree.ElementTree 모듈 - XML 파싱하는데 필요

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

if sys.version_info[0] == 3:
    from urllib.request import urlopen
else:
    from urllib import urlopen

url = "https://www.w3schools.com/xml/cd_catalog.xml"
response = urlopen(url).read()
xtree = ET.fromstring(response)
print(xtree)
```

XML 데이터 읽고 쓰기

- 웹에서 XML 데이터를 읽어와서 pandas DataFrame으로 생성
  - for loop을 돌면서 나무의 노드들(nodes of tree)에서 필요한 정보를 찾아 파싱(find and parse XML data)하여 텍스트 데이터(text)로 변환하여 사전형(Dictionary)의 키, 값의 쌍으로 추가(append)

```
rows = []

# iterate through each node of the tree
for node in xtree:
    n_title = node.find("TITLE").text
    n_artist = node.find("ARTIST").text
    n_country = node.find("COUNTRY").text
    n_company = node.find("COMPANY").text
    n_price = node.find("PRICE").text
    n_year = node.find("YEAR").text

    rows.append({"title": n_title,
                 "artist": n_artist,
                 "country": n_country,
                 "company": n_company,
                 "price": n_price,
                 "year": n_year})
print(rows)
```

XML 데이터 읽고 쓰기

➢ 웹에서 XML 데이터를 읽어와서 pandas DataFrame으로 생성

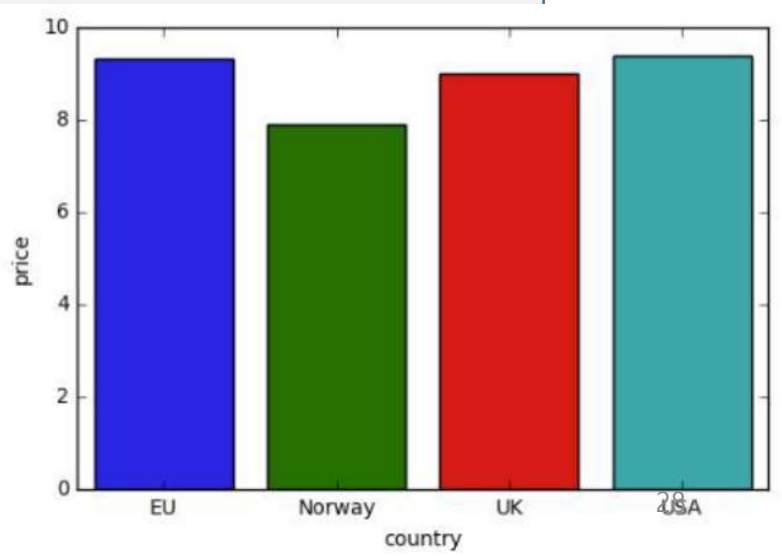
```
#XML text data를 dict로  저장된 list를 pandas DataFrame으로 변환
columns = ["title", "artist", "country", "company", "price", "year"]
catalog_cd_df = pd.DataFrame(rows, columns = columns)
catalog_cd_df.head(10)

#df.dtypes 로 각 칼럼의 데이터 형태를 확인 - 문자열 객체(string object)
print(catalog_cd_df.dtypes)

# astype()을 이용하여 칼럼 중에서 price는 float64, year는 int32로 변환
mport numpy as np
catalog_cd_df = catalog_cd_df.astype({'price': np.float,    'year': int})
print(catalog_cd_df.dtypes)

country_mean = catalog_cd_df.groupby('country').price.mean()
country_mean

country_mean_df = pd.DataFrame(country_mean).reset_index()
import seaborn as sns
sns.barplot(x='country', y='price', data=country_mean_df)
plt.show()
```



excel 데이터 읽고 쓰기

- Excel 데이터를 읽어와서 pandas DataFrame으로 생성
  - pandas의 read_excel() 함수
  - sheet_name = 데이터를 읽어올 쉬트이름
  - header = 컬럼이름 행번호 , names = 컬럼이름을 직접 입력
  - index_col=index로 사용할 열번호 또는 이름
  - dtype = 데이터 형 변환
  - thousands = ',' #천 단위 구분 기호
  - number of rows)=읽어올 행의 개수

```
import numpy as np
import pandas as pd
import os

base_dir = 'D:/admin/Documents'
excel_file = 'sales_per_region.xlsx'
excel_dir = os.path.join(base_dir, excel_file)

df_from_excel = pd.read_excel(excel_dir,  sheet_name = 'Sheet1', header = 2,
                              #names = ['region', 'sales_representative', 'sales_amount'],
                              dtype = {'region': str, 'sales_representative': np.int64,  'sales_amount': float}, # dictionary type
                              index_col = 'id',  na_values = 'NaN',     thousands = ',',
                              nrows = 10, comment = '#')
```

excel 데이터 읽고 쓰기

- DataFrame 을 Excel로 내보내기
  - 데이터프레임 값을 CSV 파일로 출력하고 싶으면 to_csv 메서드를 사용
  - 리눅스나 맥에서는 cat 셸 명령으로 파일의 내용을 확인할 수 있다.
  - 파일로 출력할 때도 sep 인수로 구분자를 바꿀 수 있다.기

```
import pandas as pd

%%writefile sample3.txt
c1        c2        c3        c4
0.179181 -1.538472  1.347553  0.43381
1.024209  0.087307 -1.281997  0.49265
0.417899 -2.002308  0.255245 -1.10515
df =pd.read_table('sample3.txt', sep='₩s+')
df
df.to_csv('sample6.csv')
!cat sample6.csv  # 윈도우에서는 !type sample6.csv 함수를 사용
df.to_csv('sample7.txt', sep='|')
```

excel 데이터 읽고 쓰기

- DataFrame 을 Excel로 내보내기
  - na_rep 인수로 NaN 표시값을 바꿀 수도 있다.
  - index, header 인수를 지정하여 인덱스 및 헤더 출력 여부를 지정하는 것도 가능하다.

```
import pandas as pd

%%writefile sample1.csv
c1,         c2,         c3,         c4
0.179181, -1.538472,   1.347553,   0.43381
1.024209,   , -1.281997,   0.49265
0.417899, -2.002308,   0.255245, -1.10515
df = pd.read_csv('sample1.csv', na_values=['누락'])
df
df.to_csv('sample3.csv', na_rep='누락')
df.to_csv('sample4.csv', index=False, header=False)
```

excel 데이터 읽고 쓰기

- 인터넷 상의 CSV 파일 로드
  - read_csv 명령 사용시 URL을 지정하면 해당 파일을 다운로드하여 읽어 들인다.

```
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/datascienceschool/docker_rpython/master/data/titanic.csv")
pd.set_option("display.max_rows", 20)  # 앞뒤로 모두 20행만 보여준다.
print( df )
```

DB로부터 데이터 읽기

SQLite DB의 특징

- 내장형 데이터베이스
- 외부 접근 불가
- RDB (관계지향 데이터베이스)
- 일반 DB와 동일한 쿼리문 사용
- 다운로드 사이트 : https://www.sqlite.org/index.html

DB로부터 데이터 읽기

SQLite 모듈 함수

```
import sqlite3

print(sqlite3.sqlite_version_info) # (3, 28, 0)
```

모듈 함수	설명
sqlite3.connect(database[timeout, isolation_level, detect_types, factory])	SQLite3 DB 연결
sqlite3.complete_statement(sql)	SQL 문장에 대해서 True를 반환
sqlite3.register_adapter(type, callable)	사용자 정의 파이썬 자료형을 SQLite3에서 사용하도록 등록
sqlite3.register_converter(typename, callable)	SQLite3에 저장된 자료를 사용자 정의 자료형으로 변환하는 함수를 등록

DB로부터 데이터 읽기

Connection 클래스

메서드	설명
Connection.cursor()	Cursor 객체 생성
Connection.commit()	현재 트랜잭션의 변경내역을 DB에 반영(commit)함
Connection.rollback()	가장 최근의 commit() 이후 지금까지 작업한 내용에 대해서 되돌림
Connection.close()	DB 연결을 종료
Connection.isolation_level()	트랜잭션의 격리 수준(isolation level)을 확인/설정

```
import sqlite3
con = sqlite3.connect("test.db")
#메모리를 이용한 Connection 객체 생성
con = sqlite3.connect(":memory:")
```

DB로부터 데이터 읽기

Cursor 클래스

- Cursor.execute () - SQL문을 인자로 전달하여 SQL문을 수행

메서드	설명
Cursor.execute(sql[, parameters])	Cursor 객체 생성
Cursor.create_aggregate(name, num_params, aggregate_class)	현재 트랜잭션의 변경내역을 DB에 반영(commit)함
Cursor.create_collation(name, callable)	가장 최근의 commit() 이후 지금까지 작업한 내용에 대해서 되돌림
Cursor.iterdump()	연결된 DB의 내용을 SQL 질의 형태로 출력
Cursor.executemany(sql, seq_of_parameters)	동일한 SQL 문장을 파라미터만 변경하며 수행
Cursor.executescript(sql_script)	세미콜론으로 구분된 연속된 SQL 문장을 수행
Cursor.fetchone()	조회된 결과(Record Set)로부터 데이터 1개를 반환
Cursor.fetchmany([size=cursor.arraysize])	조회된 결과로부터 입력받은 size 만큼의 데이터를 리스트 형태로 반환
Cursor.fetchall()	조회된 결과 모두를 리스트 형태로 반환

DB로부터 데이터 읽기

- **SQLite 사용**

```
import sqlite3

try :
    # db 연동 객체 : db 생성 + 연동
    conn = sqlite3.connect("chap08_Database/data/sqlite.db")
    # sql문 실행
    cursor = conn.cursor()
    # table 생성
    sql = """create table if not exists test_tab(
    name text(10),
    phone text(15),
    addr text(50)
    )
    """
    cursor.execute(sql)  # 실제 table 생성
    # 레코드 추가
    cursor.execute("insert into test_tab values('홍길동', '010-111-1111', '서울시')")
    cursor.execute("insert into test_tab values('이순신', '010-222-2222', '해남시')")
    cursor.execute("insert into test_tab values('유관순', '010-333-3333', '충남시')")
```

DB로부터 데이터 읽기

- **SQLite 사용**

```
# 레코드 조회
      cursor.execute("select * from test_tab")
      rows = cursor.fetchall()
      # fetchall() : 객체 안에 있는 모든 값을 가져온다
      for row in rows :
            #print(row) # tuple
            print(row[0], row[1], row[2])
except :
      # db 연동 error 처리
      print('db 연동 error')
      conn.rollback() # db 반영 취소
finally :
      # db 연동 객체 close - 나중에 만들어진 객체를 먼저 닫는게 일반적
      cursor.close()
      conn.close()
```

DB로부터 데이터 읽기

SQLite 사용

- 인자를 전달하여 SQL문을 수행

```
name = "Someone"
phoneNumber = '010-5678-1234'
cur.execute("INSERT INTO PhoneBook VALUES(?, ?);", (name, phoneNumber))
```

- 사전을 이용한 인자 전달

```
cur.execute("INSERT INTO PhoneBook VALUES(:inputName, :inputNum);",
{"inputNum":phoneNumber, "inputName":name})
```

- 동일한 문장을 매개변수만 바꾸면 연속적으로 수행하는 경우

```
datalist = (('Tom', '010-5432-5432'), ('DSP', '010-1234-1234'))
cur.executemany("INSERT INTO PhoneBook VALUES(?, ?);", datalist)
```

DB로부터 데이터 읽기

SQLite 사용

- fetch를 이용한 레코드 조회

```
cur.execute("SELECT * FROM PhoneBook; ")
for row in cur:
    print(row)
```

- fetchone, fetchmany를 이용한 레코드 조회

```
cur.execute("SELECT * FROM PhoneBook; ")
cur.fetchone()
cur.fetchall()
```

DB로부터 데이터 읽기

SQLite 사용

- 트랜잭션 처리

```
import sqlite3
con = sqlite3.connect(“commit.db")
cur = con.cursor()
cur.execute("CREATE TABLE PhoneBook(Name text, PhoneNum text);")
cur.execute("INSERT INTO PhoneBook VALUES('Someone', '010-1234-5678');")
con.commit()
cur.execute("SELECT * FROM PhoneBook; ")
print(cur.fetchall())
```

DB로부터 데이터 읽기

SQLite 사용

- 사용자 임의로 정렬 방식을 변경하는 경우

```
def  OrderFunc(str1, str2) : # 대소문자 구별 없이 정렬하는 함수
    s1 = str1.upper()
    s2 = str2.upper()
    return (s1 > s2) - (s1 < s2)

con.create_collation('myordering', OrderFunc) #SQL 구문에서 호출할 이름과 함수를 등록
cur.execute("SELECT Name FROM  PhoneBook ORDER BY Name COLLATE myordering")
[r[0] for r in cur]
```

DB로부터 데이터 읽기

내장/집계 함수

함수	설명
abs(x)	절대값을 반환
length(x)	문자열의 길이를 반환
lower(x)	소문자로 변환해서 반환
upper(x)	대문자로 변환해서 반환
min(...)	최소값을 반환
max(...)	최대값을 반환
random(*)	임의의 정수를 반환
count(x)	NULL이 아닌 튜플의 개수를 반환
count(*)	튜플의 개수를 반환
sum(x)	합을 반환

DB로부터 데이터 읽기

- **SQLite 자료형과 그에 대응되는 파이썬 자료형**

SQLite 자료형	Python 자료형
NULL	None
INTEGER	int
REAL	float
TEXT	str, float
BLOB	buffer

```
cur.execute("CREATE TABLE tbl_1 (Name TEXT, Age INTEGER, Money REAL); ")
cur.execute("CREATE TABLE tbl_2 (Name str, AGE int, Money float);")
```

DB로부터 데이터 읽기

- **데이터베이스 덤프 만들기**

```
import sqlite3
con = sqlite3.connect(":memory:")
cur = con.cursor()

cur.execute("CREATE TABLE PhoneBook (Name TEXT, PhoneNum text); ")
cur.execute("INSERT INTO  PhoneBook  VALUES ('Derick', '010-1234-5678);")
list = (('Tom', '010-5432-5432', ('DSP', '010-1234-1234'))
cur.executemany("INSERT INTO PhoneBook VALUES(?, ?);", list)

for row in con.iterdump():
    print(row)
```

DB로부터 데이터 읽기

- cx_Oracle 모듈 설치

```
python -m pip install cx_Oracle --upgrade (기본 python 다운 받은 환경)
python -m pip install cx_Oracle --upgrade --user (아나콘다 환경에서 다운로드 할때.)
pip install cx_Oracle
conda install -c https://conda.anaconda.org/anaconda cx_oracle
```

DB로부터 데이터 읽기

- python oracle 연동

```
from cx_Oracle
#한글 지원 방법
import os
os.putenv('NLS_LANG', '.UTF8')

#연결에 필요한 기본 정보(유저, 비밀번호, 데이터베이스 서버 주소)
connection = cx_Oracle.connect('scott', 'oracle', 'localhost/orcl')
cursor = connection.cursor()

cursor.execute("""
    select ename
    from emp
    where deptno = 10""",
    texting="테스트"
)
for name in cursor:
    print("테스트 이름 리스트 : ", name)
```

DB로부터 데이터 읽기

- mariaDB 연동 – pymysql 모듈

```
import pymysql

print(pymysql.version_info) # (1, 3, 12, 'final', 0)

config = {
    'host' : '127.0.0.1',
    'user' : 'root',
    'password' : '1234',
    'database' : 'work',
    'port' : 3306,
    'charset':'utf8',
    'use_unicode' : True}

try :
    # db 환경변수 -> db 연동 객체
    conn = pymysql.connect(**config)
        # **config : config에 들어있는 7개의 환경변수를 이용해서 DB를 연동한다는 의미
    # sql 실행 객체
    cursor = conn.cursor()
    print("db 연동 성공!")

    sql = "show tables"
    cursor.execute(sql)
    tables = cursor.fetchall()

    if tables :
        print('table 있음')
    else :
        print('table 없음')

except Exception as e :
    print('db 연동 error :', e)
finally :
    cursor.close()
    conn.close()
```

DB로부터 데이터 읽기

- mariaDB 연동 – pymysql 모듈

```
import pymysql

print(pymysql.version_info) # (1, 3, 12, 'final', 0)

config = {
    'host' : '127.0.0.1',
    'user' : 'root',
    'password' : '1234',
    'database' : 'work',
    'port' : 3306,
    'charset':'utf8',
    'use_unicode' : True}

try :
    # db 환경변수 -> db 연동 객체
    conn = pymysql.connect(**config)
        # **config : config에 들어있는 7개의 환경변수를 이용해서 DB를 연동한다는 의미
    # sql 실행 객체
    cursor = conn.cursor()
    print("db 연동 성공!")

    sql = "show tables"
    cursor.execute(sql)
    tables = cursor.fetchall()

    if tables :
        print('table 있음')
    else :
        print('table 없음')

except Exception as e :
    print('db 연동 error :', e)
finally :
    cursor.close()
    conn.close()
```