

파이썬 라이브러리를 활용한 데이터 분석

6장 데이터 로딩과 저장, 파일형식

2020.06.26 1h

텍스트 파일에서 데이터를 읽고 쓰는 법

- 메소드 `read_csv()`, `read_table()`

Table 6-1. Parsing functions in pandas

Function	Description
<code>read_csv</code>	Load delimited data from a file, URL, or file-like object; use comma as default delimiter
<code>read_table</code>	Load delimited data from a file, URL, or file-like object; use tab (<code>'\t'</code>) as default delimiter
<code>read_fwf</code>	Read data in fixed-width column format (i.e., no delimiters)
<code>read_clipboard</code>	Version of <code>read_table</code> that reads data from the clipboard; useful for converting tables from web pages
<code>read_excel</code>	Read tabular data from an Excel XLS or XLSX file
<code>read_hdf</code>	Read HDF5 files written by pandas
<code>read_html</code>	Read all tables found in the given HTML document
<code>read_json</code>	Read data from a JSON (JavaScript Object Notation) string representation
<code>read_msgpack</code>	Read pandas data encoded using the MessagePack binary format
<code>read_pickle</code>	Read an arbitrary object stored in Python pickle format

옵션

- 색인
 - 하나 이상의 컬럼을 색인으로 지정
- 자료형 추론과 데이터 변환
 - `read_csv()` 등은 자료형을 따로 지정하지 않으므로 자료형 추론을 실행
- 날짜 분석
- 반복
- 정제되지 않은 데이터 처리
 - 주식 건너뛰기, 천 단위 자릿수

메소드 `pd.read_csv()` 개요

- `pandas.read_csv(filepath_or_buffer: Union[str, pathlib.Path, IO[~AnyStr]], sep=',', delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None, decimal: str = '.', lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, dialect=None, error_bad_lines=True, warn_bad_lines=True, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None)`

메소드 `pd.read_csv()`

```
In [15]: !type examples#ex1.csv
```

```
a,b,c,d,message
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

```
In [16]: df = pd.read_csv('examples/ex1.csv')
df
```

```
Out[16]:
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [18]: pd.read_table('examples/ex1.csv', sep=',')
```

```
Out[18]:
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [20]: df.index
```

```
Out[20]: RangeIndex(start=0, stop=3, step=1)
```

```
In [21]: df.columns
```

```
Out[21]: Index(['a', 'b', 'c', 'd', 'message'], dtype='object')
```

칼럼 명 해제 및 지정

- 옵션: **header=None**
- 옵션: **names=[...]**
- 특정 칼럼을 인덱스로 활용
 - `index_col = 'message'`

In [8]: `!type examples/ex2.csv`

```
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

In [9]: `pd.read_csv('examples/ex2.csv', header=None)`

Out[9]:

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [10]: `pd.read_csv('examples/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])`

Out[10]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [11]: `names = ['a', 'b', 'c', 'd', 'message']`
`pd.read_csv('examples/ex2.csv', names=names, index_col='message')`

Out[11]:

	a	b	c	d
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

계층적 색인 지정

• 칼럼 번호나 이름의 리스트 사용

```
In [22]: !type examples/csv_mindex.csv
```

```
key1, key2, value1, value2
one, a, 1, 2
one, b, 3, 4
one, c, 5, 6
one, d, 7, 8
two, a, 9, 10
two, b, 11, 12
two, c, 13, 14
two, d, 15, 16
```

```
In [24]: parsed = pd.read_csv('examples/csv_mindex.csv',
                                index_col=['key1', 'key2'])
parsed
```

Out [24]:

		value1	value2
one	key1	key2	
	a	1	2
	b	3	4
	c	5	6
two	d	7	8
	a	9	10
	b	11	12
	c	13	14
	d	15	16

```
In [25]: parsed1 = pd.read_csv('examples/csv_mindex.csv',
                                index_col=[0, 1])
parsed1
```

Out [25]:

		value1	value2
one	key1	key2	
	a	1	2
	b	3	4
	c	5	6
two	d	7	8
	a	9	10
	b	11	12
	c	13	14
	d	15	16

데이터를 공백이나 패턴으로 구분

- 옵션 `sep=`

- 'Ws'

- 정규 표현식

- 여러 개의 공백 문자

- 데이터 추론

- 첫 로우가 데이터가 하나 적은

- 칼럼 명으로 추론

```
In [26]: !type examples\ex3.txt
```

```

          A          B          C
aaa -0.264438 -1.026059 -0.619500
bbb  0.927272  0.302904 -0.032399
ccc -0.264273 -0.386314 -0.217601
ddd -0.871858 -0.348382  1.100491
```

```
In [27]: list(open('examples/ex3.txt'))
```

```
Out[27]: ['          A          B          C\n',
'aaa -0.264438 -1.026059 -0.619500\n',
'bbb  0.927272  0.302904 -0.032399\n',
'ccc -0.264273 -0.386314 -0.217601\n',
'ddd -0.871858 -0.348382  1.100491\n']
```

```
In [28]: result = pd.read_table('examples/ex3.txt', sep='Ws+')
result
```

```
Out[28]:
```

	A	B	C
aaa	-0.264438	-1.026059	-0.619500
bbb	0.927272	0.302904	-0.032399
ccc	-0.264273	-0.386314	-0.217601
ddd	-0.871858	-0.348382	1.100491

옵션 skiprows=

• 행 건너뛰기

In [29]: `!type examples/ex4.csv`

```
# hey!
a,b,c,d,message
# just wanted to make things more difficult for you
# who reads CSV files with computers, anyway?
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

In [30]: `pd.read_csv('examples/ex4.csv')`

Out[30]:

					# hey!
	a	b	c	d	message
	# just wanted to make things more difficult for you				NaN
	# who reads CSV files with computers				NaN
	1	2	3	4	hello
	5	6	7	8	world
	9	10	11	12	foo

In [31]: `pd.read_csv('examples/ex4.csv', skiprows=[0, 2, 3])`

Out[31]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

누락 값 처리

• NA, NULL, 또는 빈 부분 사용

```
In [32]: !type examples#ex5.csv
```

```
something,a,b,c,d,message
one,1,2,3,4,NA
two,5,6,,8,world
three,9,10,11,12,foo
```

```
In [33]: result = pd.read_csv('examples/ex5.csv')
result
```

Out[33]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

```
In [34]: pd.isnull(result)
```

Out[34]:

	something	a	b	c	d	message
0	False	False	False	False	False	True
1	False	False	False	True	False	False
2	False	False	False	False	False	False

옵션 `na_values=`

- 직접 특정한 값을 누락 값으로 지정
 - 리스트나 사전 형식
 - 칼럼마다 다른 NA 값을 사전 형식으로 지정

In [38]: result

Out[38]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

In [39]: result = pd.read_csv('examples/ex5.csv', na_values=['NULL', 4])
result

Out[39]:

	something	a	b	c	d	message
0	one	1	2	3.0	NaN	NaN
1	two	5	6	NaN	8.0	world
2	three	9	10	11.0	12.0	foo

In [40]: sentinels = {'message': ['foo', 'NA'], 'something': ['two']}
pd.read_csv('examples/ex5.csv', na_values=sentinels)

Out[40]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	NaN	5	6	NaN	8	world
2	three	9	10	11.0	12	NaN

조금씩 읽어 오기

- 처음 일부 읽기
 - 옵션 `nrows=n`
- 여러 조각으로 나누어 읽기
 - 옵션 `chunksize=n`
 - key 열에서 값의 횟수 세기
 - `chunker`를 `for in`에 사용

```
In [43]: pd.read_csv('examples/ex6.csv', nrows=5)
```

```
Out[43]:
```

	one	two	three	four	key
0	0.467976	-0.038649	-0.295344	-1.824726	L
1	-0.358893	1.404453	0.704965	-0.200638	B
2	-0.501840	0.659254	-0.421691	-0.057688	G
3	0.204886	1.074134	1.388361	-0.982404	R
4	0.354628	-0.133116	0.283763	-0.837063	Q

```
In [44]: chunker = pd.read_csv('examples/ex6.csv', chunksize=1000)
chunker
```

```
Out[44]: <pandas.io.parsers.TextFileReader at 0x1b1dfcba148>
```

```
In [49]: chunker = pd.read_csv('examples/ex6.csv', chunksize=1000)

tot = pd.Series([], dtype='float64')
for piece in chunker:
    tot = tot.add(piece['key'].value_counts(), fill_value=0)

tot = tot.sort_values(ascending=False)
```

```
In [50]: tot[:10]
```

```
Out[50]: E    368.0
X    364.0
L    346.0
O    343.0
Q    340.0
M    338.0
J    337.0
F    335.0
K    334.0
H    330.0
dtype: float64
```

더하는 두 시리즈의 하나가 NA이면 0으로 처리