

# Lecture 5: Data Loading, Storage and File Formats

---

# 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 ( ' \t ' ) 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

Function	Description
<code>read_sas</code>	Read a SAS dataset stored in one of the SAS system's custom storage formats
<code>read_sql</code>	Read the results of a SQL query (using SQLAlchemy) as a pandas DataFrame
<code>read_stata</code>	Read a dataset from Stata file format
<code>read_feather</code>	Read the Feather binary file format

- *Indexing*
  - Can treat one or more columns as the returned DataFrame, and whether to get column names from the file, the user, or not at all.
- *Type inference and data conversion*
  - This includes the user-defined value conversions and custom list of missing value markers.
- *Datetime parsing*
  - Includes combining capability, including combining date and time information spread over multiple columns into a single column in the result.
- *Iterating*
  - Support for iterating over chunks of very large files.
- *Unclean data issues*
  - Skipping rows or a footer, comments, or other minor things like numeric data with thousands separated by commas.

```
In [8]: !cat examples/ex1.csv
a,b,c,d,message
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

```
In [9]: df = pd.read_csv('examples/ex1.csv')
```

```
In [10]: df
```

```
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 [12]: !cat examples/ex2.csv
```

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

```
In [11]: pd.read_table('examples/ex1.csv', sep=',')
```

```
Out[11]:
```

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

```
In [13]: pd.read_csv('examples/ex2.csv', header=None)
```

```
Out[13]:
```

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

```
In [14]: pd.read_csv('examples/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

```
Out[14]:
```

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

```
In [15]: names = ['a', 'b', 'c', 'd', 'message']
```

```
In [16]: pd.read_csv('examples/ex2.csv', names=names, index_col='message')
```

```
Out[16]:
```

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

```
In [17]: !cat 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 [18]: parsed = pd.read_csv('examples/csv_mindex.csv',  
.....:                        index_col=['key1', 'key2'])
```

```
In [19]: parsed
```

```
Out[19]:
```

		value1	value2
one	key1	key2	
	a		1
	b		3
	c		5
two	d		7
	a		9
	b		11
	c		13
	d		15

```
In [20]: list(open('examples/ex3.txt'))
```

```
Out[20]:
```

```
['          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 [21]: result = pd.read_table('examples/ex3.txt', sep='\s+')
```

```
In [22]: result
```

```
Out[22]:
```

	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 [23]: !cat 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 [24]: pd.read_csv('examples/ex4.csv', skiprows=[0, 2, 3])
```

```
Out[24]:
```

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

```
In [25]: !cat 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 [26]: result = pd.read_csv('examples/ex5.csv')
```

```
In [27]: result
```

```
Out[27]:
```

	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 [28]: pd.isnull(result)
```

```
Out[28]:
```

	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

```
In [29]: result = pd.read_csv('examples/ex5.csv', na_values=['NULL'])
```

```
In [30]: result
```

```
Out[30]:
```

	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 [31]: sentinels = {'message': ['foo', 'NA'], 'something': ['two']}
```

```
In [32]: pd.read_csv('examples/ex5.csv', na_values=sentinels)
```

```
Out[32]:
```

	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

# read\_csv/read\_table function arguments

Argument	Description
path	String indicating filesystem location, URL, or file-like object
sep or delimiter	Character sequence or regular expression to use to split fields in each row
header	Row number to use as column names; defaults to 0 (first row), but should be None if there is no header row
index_col	Column numbers or names to use as the row index in the result; can be a single name/number or a list of them for a hierarchical index
names	List of column names for result, combine with header=None



Argument	Description
<code>skiprows</code>	Number of rows at beginning of file to ignore or list of row numbers (starting from 0) to skip.
<code>na_values</code>	Sequence of values to replace with NA.
<code>comment</code>	Character(s) to split comments off the end of lines.
<code>parse_dates</code>	Attempt to parse data to <code>datetime</code> ; <code>False</code> by default. If <code>True</code> , will attempt to parse all columns. Otherwise can specify a list of column numbers or name to parse. If element of list is tuple or list, will combine multiple columns together and parse to date (e.g., if date/time split across two columns).
<code>keep_date_col</code>	If joining columns to parse date, keep the joined columns; <code>False</code> by default.
<code>converters</code>	Dict containing column number or name mapping to functions (e.g., <code>{ 'foo' : f }</code> would apply the function <code>f</code> to all values in the <code>'foo'</code> column).
<code>dayfirst</code>	When parsing potentially ambiguous dates, treat as international format (e.g., <code>7/6/2012</code> -> June 7, 2012); <code>False</code> by default.
<code>date_parser</code>	Function to use to parse dates.
<code>nrows</code>	Number of rows to read from beginning of file.
<code>iterator</code>	Return a <code>TextParser</code> object for reading file piecemeal.
<code>chunksize</code>	For iteration, size of file chunks.
<code>skip_footer</code>	Number of lines to ignore at end of file.
<code>verbose</code>	Print various parser output information, like the number of missing values placed in non-numeric columns.
<code>encoding</code>	Text encoding for Unicode (e.g., <code>'utf-8'</code> for UTF-8 encoded text).
<code>squeeze</code>	If the parsed data only contains one column, return a Series.
<code>thousands</code>	Separator for thousands (e.g., <code>' , '</code> or <code>' . '</code> ).

# Reading Text Files in Pieces

```
In [33]: pd.options.display.max_rows = 10
```

```
In [34]: result = pd.read_csv('examples/ex6.csv')
```

```
In [35]: result
```

```
Out[35]:
```

	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
...	...	...	...	...	..
9995	2.311896	-0.417070	-1.409599	-0.515821	L
9996	-0.479893	-0.650419	0.745152	-0.646038	E
9997	0.523331	0.787112	0.486066	1.093156	K
9998	-0.362559	0.598894	-1.843201	0.887292	G
9999	-0.096376	-1.012999	-0.657431	-0.573315	0

[10000 rows x 5 columns]

```
In [36]: pd.read_csv('examples/ex6.csv', nrows=5)
```

```
Out[36]:
```

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

```
In [38]: chunker
```

```
Out[38]: <pandas.io.parsers.TextFileReader at 0x7f6b1e2672e8>
```

```
chunker = pd.read_csv('examples/ex6.csv', chunksize=1000)

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

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

```
In [40]: tot[:10]
```

```
Out[40]:
```

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
```

# Writing Data to Text Format

```
In [41]: data = pd.read_csv('examples/ex5.csv')
```

```
In [42]: data
```

```
Out[42]:
```

	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 [43]: data.to_csv('examples/out.csv')
```

```
In [44]: !cat examples/out.csv
```

```
,something,a,b,c,d,message
```

```
0,one,1,2,3.0,4,
```

```
1,two,5,6,,8,world
```

```
2,three,9,10,11.0,12,foo
```

```
In [45]: import sys
```

```
In [46]: data.to_csv(sys.stdout, sep='|')
```

```
|something|a|b|c|d|message
```

```
0|one|1|2|3.0|4|
```

```
1|two|5|6||8|world
```

```
2|three|9|10|11.0|12|foo
```

```
In [47]: data.to_csv(sys.stdout, na_rep='NULL')
```

```
,something,a,b,c,d,message
```

```
0,one,1,2,3.0,4,NULL
```

```
1,two,5,6,NULL,8,world
```

```
2,three,9,10,11.0,12,foo
```

```
In [48]: data.to_csv(sys.stdout, index=False, header=False)
```

```
one,1,2,3.0,4,
```

```
two,5,6,,8,world
```

```
three,9,10,11.0,12,foo
```

```
In [49]: data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c'])
```

```
a,b,c
```

```
1,2,3.0
```

```
5,6,
```

```
9,10,11.0
```

```
In [50]: dates = pd.date_range('1/1/2000', periods=7)
```

```
In [51]: ts = pd.Series(np.arange(7), index=dates)
```

```
In [52]: ts.to_csv('examples/tseries.csv')
```

```
In [53]: !cat examples/tseries.csv
```

```
2000-01-01,0
```

```
2000-01-02,1
```

```
2000-01-03,2
```

```
2000-01-04,3
```

```
2000-01-05,4
```

```
2000-01-06,5
```

```
2000-01-07,6
```

# Working with Delimited Formats

```
In [54]: !cat examples/ex7.csv
```

```
"a","b","c"
```

```
"1","2","3"
```

```
"1","2","3"
```

```
import csv
```

```
f = open('examples/ex7.csv')
```

```
reader = csv.reader(f)
```

```
In [56]: for line in reader:
```

```
.....:     print(line)
```

```
['a', 'b', 'c']
```

```
['1', '2', '3']
```

```
['1', '2', '3']
```

```
In [57]: with open('examples/ex7.csv') as f:
```

```
.....:     lines = list(csv.reader(f))
```

```
In [58]: header, values = lines[0], lines[1:]
```

```
In [59]: data_dict = {h: v for h, v in zip(header, zip(*values))}
```

```
In [60]: data_dict
```

```
Out[60]: {'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}
```

```
class my_dialect(csv.Dialect):
```

```
    lineterminator = '\n'
```

```
    delimiter = ';' 
```

```
    quotechar = '"'
```

```
    quoting = csv.QUOTE_MINIMAL
```

```
reader = csv.reader(f, dialect=my_dialect)
```

```
reader = csv.reader(f, delimiter='|')
```



# CSV dialect options

Argument	Description
<code>delimiter</code>	One-character string to separate fields; defaults to <code>' , '</code> .
<code>lineterminator</code>	Line terminator for writing; defaults to <code>' \r\n '</code> . Reader ignores this and recognizes cross-platform line terminators.
<code>quotechar</code>	Quote character for fields with special characters (like a delimiter); default is <code>' \" '</code> .
<code>quoting</code>	Quoting convention. Options include <code>csv.QUOTE_ALL</code> (quote all fields), <code>csv.QUOTE_MINIMAL</code> (only fields with special characters like the delimiter), <code>csv.QUOTE_NONNUMERIC</code> , and <code>csv.QUOTE_NONE</code> (no quoting). See Python's documentation for full details. Defaults to <code>QUOTE_MINIMAL</code> .
<code>skipinitialspace</code>	Ignore whitespace after each delimiter; default is <code>False</code> .
<code>doublequote</code>	How to handle quoting character inside a field; if <code>True</code> , it is doubled (see online documentation for full detail and behavior).
<code>escapechar</code>	String to escape the delimiter if <code>quoting</code> is set to <code>csv.QUOTE_NONE</code> ; disabled by default.

# JSON Data

- JSON (short for JavaScript Object Notation) has become one of the standard formats for sending data by HTTP request between web browsers and other applications.
- ```
obj = """ {"name": "Wes", "places_lived":  
  ["United States", "Spain", "Germany"], "pet":  
  null, "siblings": [{"name": "Scott", "age": 30,  
  "pets": ["Zeus", "Zuko"]}, {"name": "Katie",  
  "age": 38, "pets": ["Sixes", "Stache",  
  "Cisco"]}]} """
```



- `import json`
- `result = json.loads(obj)`
- ```
{ 'name': 'Wes', 'places_lived': ['United States', 'Spain', 'Germany'], 'pet': None, 'siblings': [{ 'name': 'Scott', 'age': 30, 'pets': ['Zeus', 'Zuko'] }, { 'name': 'Katie', 'age': 38, 'pets': ['Sixes', 'Stache', 'Cisco'] } ] }
```

# Binary Data Formats

- One of the easiest ways to store data (also known as serialization) efficiently in binary format is using Python's built-in pickle serialization. pandas objects all have a `to_pickle` method that writes the data to disk in pickle format

```
In [87]: frame = pd.read_csv('examples/ex1.csv')
```

```
In [88]: frame
```

```
Out[88]:
```

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

```
In [90]: pd.read_pickle('examples/frame_pickle')
```

```
Out[90]:
```

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

```
In [89]: frame.to_pickle('examples/frame_pickle')
```

# Reading Microsoft Excel Files

- pandas also supports reading tabular data stored in Excel 2003 (and higher) files using either the ExcelFile class or pandas.read\_excel function. Internally these tools use the add-on packages xlrd and openpyxl to read XLS and XLSX files, respectively

```
In [104]: xlsx = pd.ExcelFile('examples/ex1.xlsx')
```

```
In [105]: pd.read_excel(xlsx, 'Sheet1')
```

```
Out[105]:
```

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

- To write pandas data to Excel format, you must first create an ExcelWriter, then write data to it using pandas objects' to\_excel method:

```
In [108]: writer = pd.ExcelWriter('examples/ex2.xlsx')
```

```
In [109]: frame.to_excel(writer, 'Sheet1')
```

```
In [110]: writer.save()
```

- You can also pass a file path to to\_excel and avoid the ExcelWriter:

```
In [111]: frame.to_excel('examples/ex2.xlsx')
```