Importing Data into Python from Common Binary Data File Formats

Xavier Morera
BIG DATA INC.

@xmorera www.xaviermorera.com

SAS

Statistical Analysis System

Quantitative business professionals

Data set contains data values

- Table of observations (rows)
- Variables (columns)

Reading SAS files with

- SAS7BDAT
- Pandas

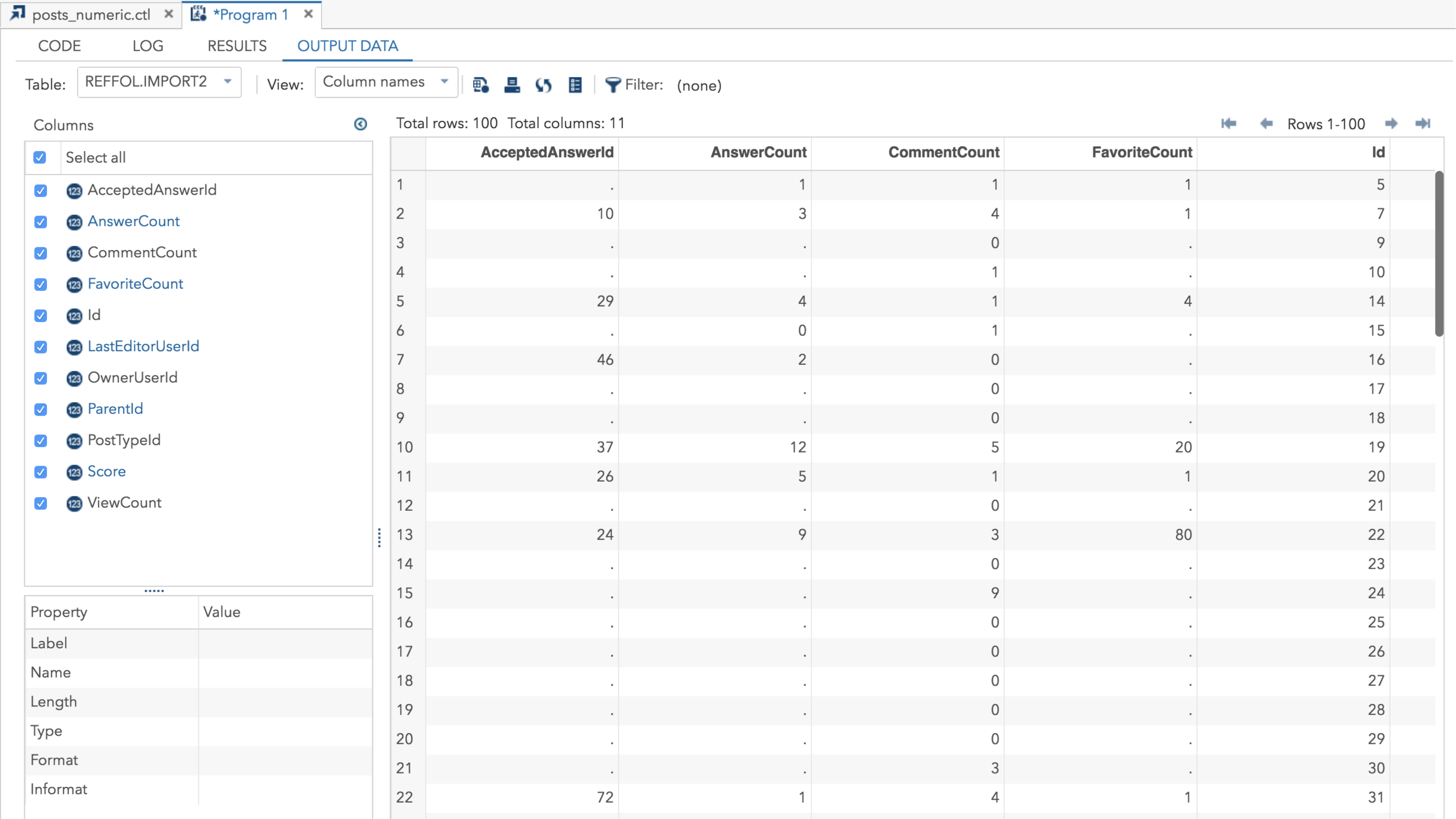


Table: REFFOL.IMPORT2 View: Column names Filter: (none)

Columns

☑ Select all

☑ AcceptedAnswerId
☑ AnswerCount
☑ CommentCount
☑ FavoriteCount
☑ Id
☑ LastEditorUserId
☑ OwnerUserId
☑ ParentId
☑ PostTypeId
☑ Score
☑ ViewCount

Property	Value
Label	
Name	
Length	
Type	
Format	
Informat	

Total rows: 100 Total columns: 11

Rows 1-100

	AcceptedAnswerId	AnswerCount	CommentCount	FavoriteCount	Id
1	.	1	1	1	5
2	10	3	4	1	7
3	.	.	0	.	9
4	.	.	1	.	10
5	29	4	1	4	14
6	.	0	1	.	15
7	46	2	0	.	16
8	.	.	0	.	17
9	.	.	0	.	18
10	37	12	5	20	19
11	26	5	1	1	20
12	.	.	0	.	21
13	24	9	3	80	22
14	.	.	0	.	23
15	.	.	9	.	24
16	.	.	0	.	25
17	.	.	0	.	26
18	.	.	0	.	27
19	.	.	0	.	28
20	.	.	0	.	29
21	.	.	3	.	30
22	72	1	4	1	31

```
from sas7bdat import SAS7BDAT
with SAS7BDAT('posts-100.sas7bdat') as sas_file:
    users_sas_df = sas_file.to_data_frame()
sas_file
dir(sas_file)
sas_file.column_names
sas_file.header
type(users_sas_df)
```

Reading SAS Files with SAS7BDAT

Use the SAS7BDAT package

Inspect the functionality, and use it

Pandas DataFrame

Pandas	SAS
DataFrame	SAS data set
slice	sub-set
row	observation
column	variable
axis 0	observation
axis 1	column

```
import pandas as pd
posts_sas = pd.read_sas('posts-100.sas7bdat')
type(posts_sas)
posts_sas.head()
posts_sas.columns
posts_sas_reader = pd.read_sas('posts-100.sas7bdat', chunksize=10)
posts_sas_reader.read()
```

Reading SAS Files with Pandas

Read SAS file using read_sas

Get a DataFrame and business as usual

For large files, use chunksize, which returns a SAS7BDATReader

- Use read

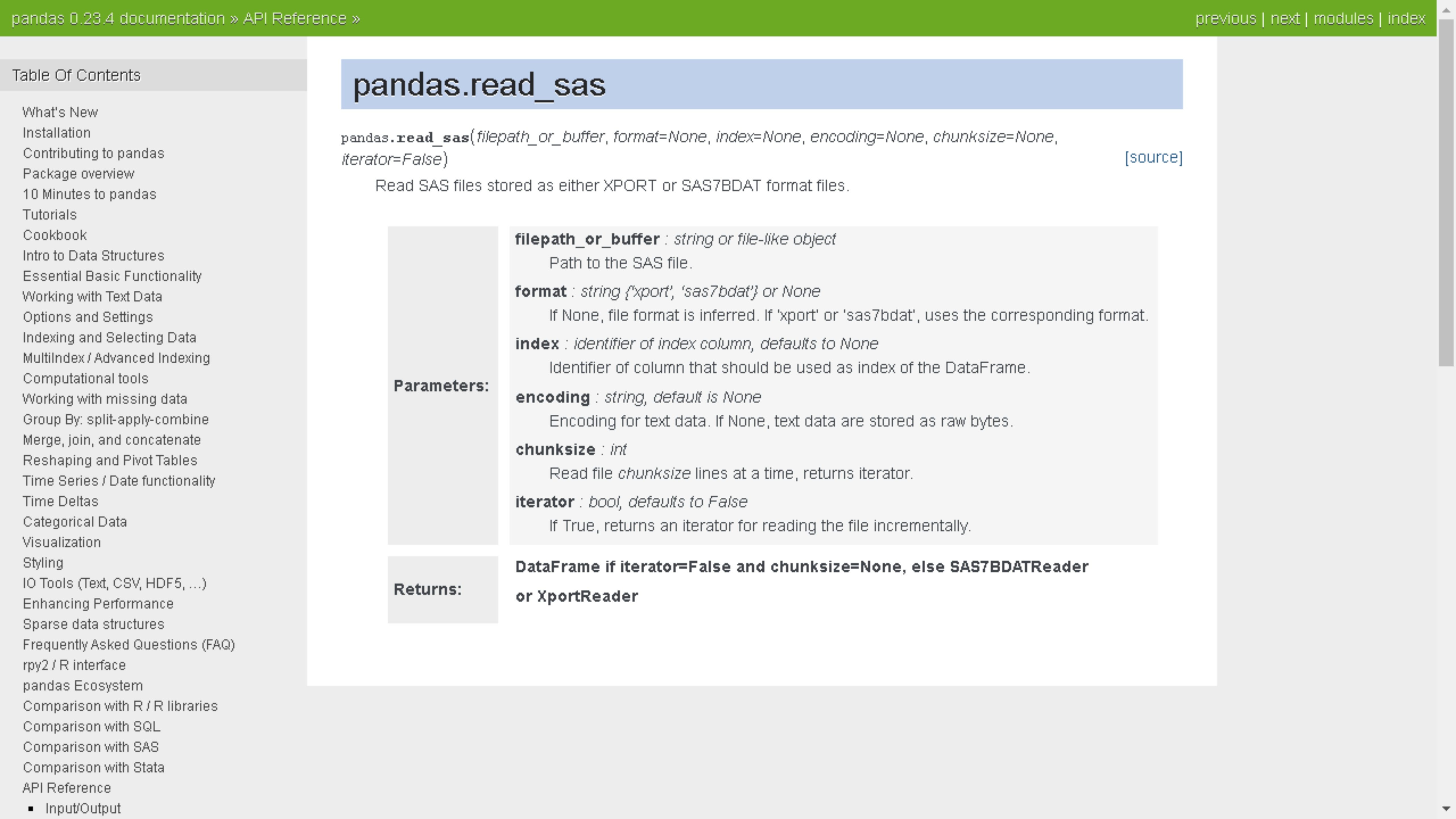


Table Of Contents

pandas.read_sas

`pandas.read_sas`(*filepath_or_buffer*, *format=None*, *index=None*, *encoding=None*, *chunksize=None*, *iterator=False*) [source]

Read SAS files stored as either XPORT or SAS7BDAT format files.

Parameters:	**filepath_or_buffer** : *string or file-like object* Path to the SAS file. **format** : *string {'xport', 'sas7bdat'} or None* If None, file format is inferred. If 'xport' or 'sas7bdat', uses the corresponding format. **index** : *identifier of index column, defaults to None* Identifier of column that should be used as index of the DataFrame. **encoding** : *string, default is None* Encoding for text data. If None, text data are stored as raw bytes. **chunksize** : *int* Read file *chunksize* lines at a time, returns iterator. **iterator** : *bool, defaults to False* If True, returns an iterator for reading the file incrementally.
Returns:	**DataFrame if iterator=False and chunksize=None, else SAS7BDATReader or XportReader**

Stata

Statistical software package

- Economics, sociology, political science…

STATistics + DatA

Reading Stata files with Pandas

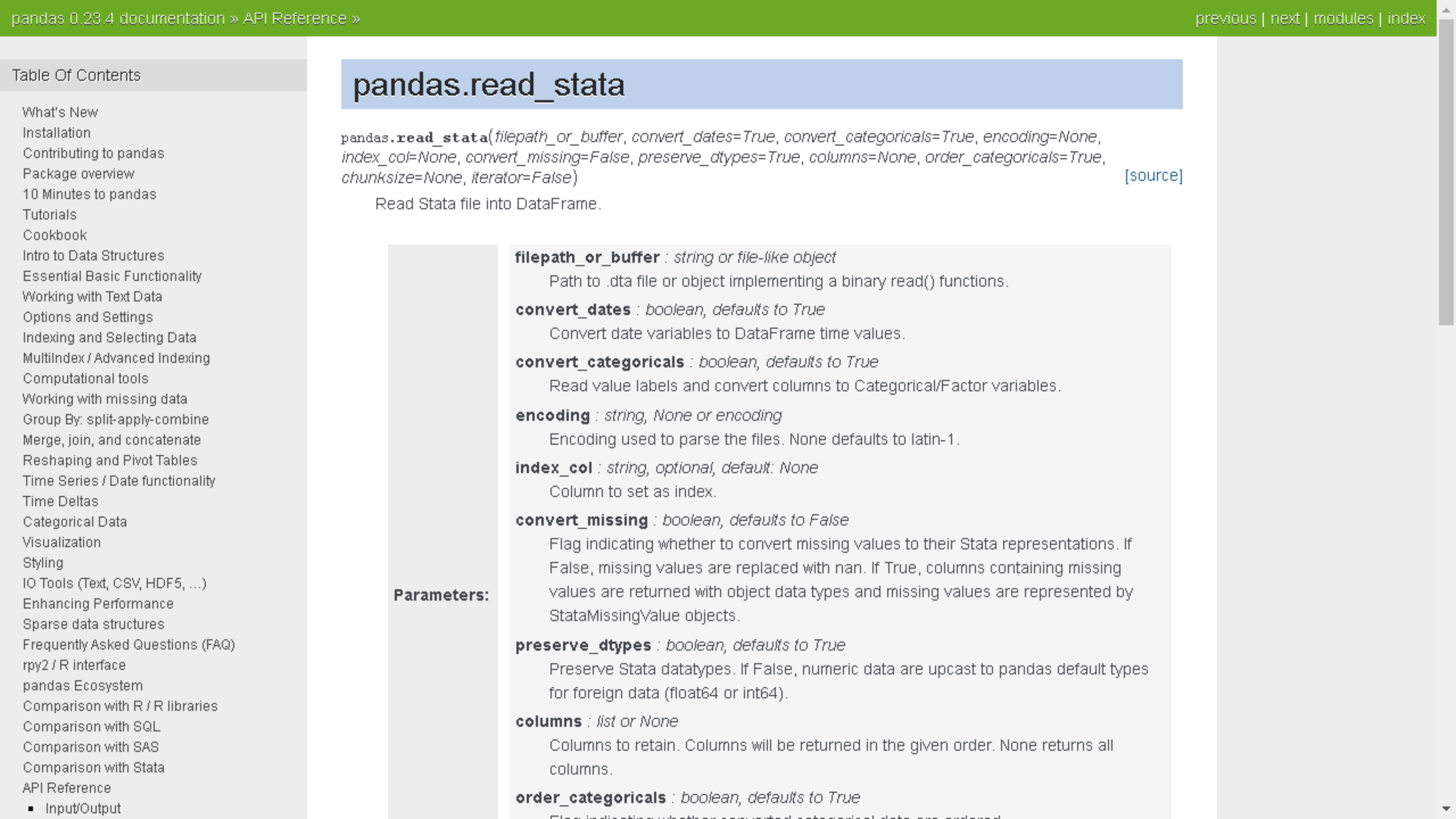
```
import pandas as pd
posts_stata = pd.read_stata('posts-100.dta')
type(posts_stata)
dir(posts_stata)
posts_stata.columns
posts_stata.head()
```

Reading Stata Files with Pandas

Use read_stata

Load into a DataFrame

Several parameters available to work with your imported data



pandas.read_stata

`pandas.read_stata`(*filepath_or_buffer*, *convert_dates=True*, *convert_categoricals=True*, *encoding=None*, *index_col=None*, *convert_missing=False*, *preserve_dtypes=True*, *columns=None*, *order_categoricals=True*, *chunksize=None*, *iterator=False*) [source]

Read Stata file into DataFrame.

Parameters:

filepath_or_buffer : *string or file-like object*

Path to .dta file or object implementing a binary read() functions.

convert_dates : *boolean, defaults to True*

Convert date variables to DataFrame time values.

convert_categoricals : *boolean, defaults to True*

Read value labels and convert columns to Categorical/Factor variables.

encoding : *string, None or encoding*

Encoding used to parse the files. None defaults to latin-1.

index_col : *string, optional, default: None*

Column to set as index.

convert_missing : *boolean, defaults to False*

Flag indicating whether to convert missing values to their Stata representations. If False, missing values are replaced with nan. If True, columns containing missing values are returned with object data types and missing values are represented by StataMissingValue objects.

preserve_dtypes : *boolean, defaults to True*

Preserve Stata datatypes. If False, numeric data are upcast to pandas default types for foreign data (float64 or int64).

columns : *list or None*

Columns to retain. Columns will be returned in the given order. None returns all columns.

order_categoricals : *boolean, defaults to True*

Flag indicating whether converted categorical data are ordered.

HDF5

Hierarchical Data Format version 5

Store large quantities of numerical data

Reading HDF5 files

- h5py
- Pandas

Requires PyTables

```
import h5py
file = h5py.File("posts-100.h5",'r')
dataset = file['posts']
for x in dataset['table']:
    print(x)
```

Reading HDF5 Files with h5py

Use h5py module

Import into a File object, get a DataSet

Start working with your data

```
import pandas as pd

posts_hdf = pd.read_hdf('posts-100.h5', 'posts')

posts_hdf.columns

posts_hdf.keys()

pd.read_hdf('posts-100.h5', 'posts', start=2, stop=5,
columns=['CreationDate','Title','Tags']).head()

pd.read_hdf('posts-100.h5', 'posts', columns=['Score', 'Tags'], where='Score>10 or Tags =
"<machine-learning>"').head()
```

Reading HDF5 Files with Pandas

Use read_hdf

- Available: mode, where, start, stop, columns, where, chunksize…

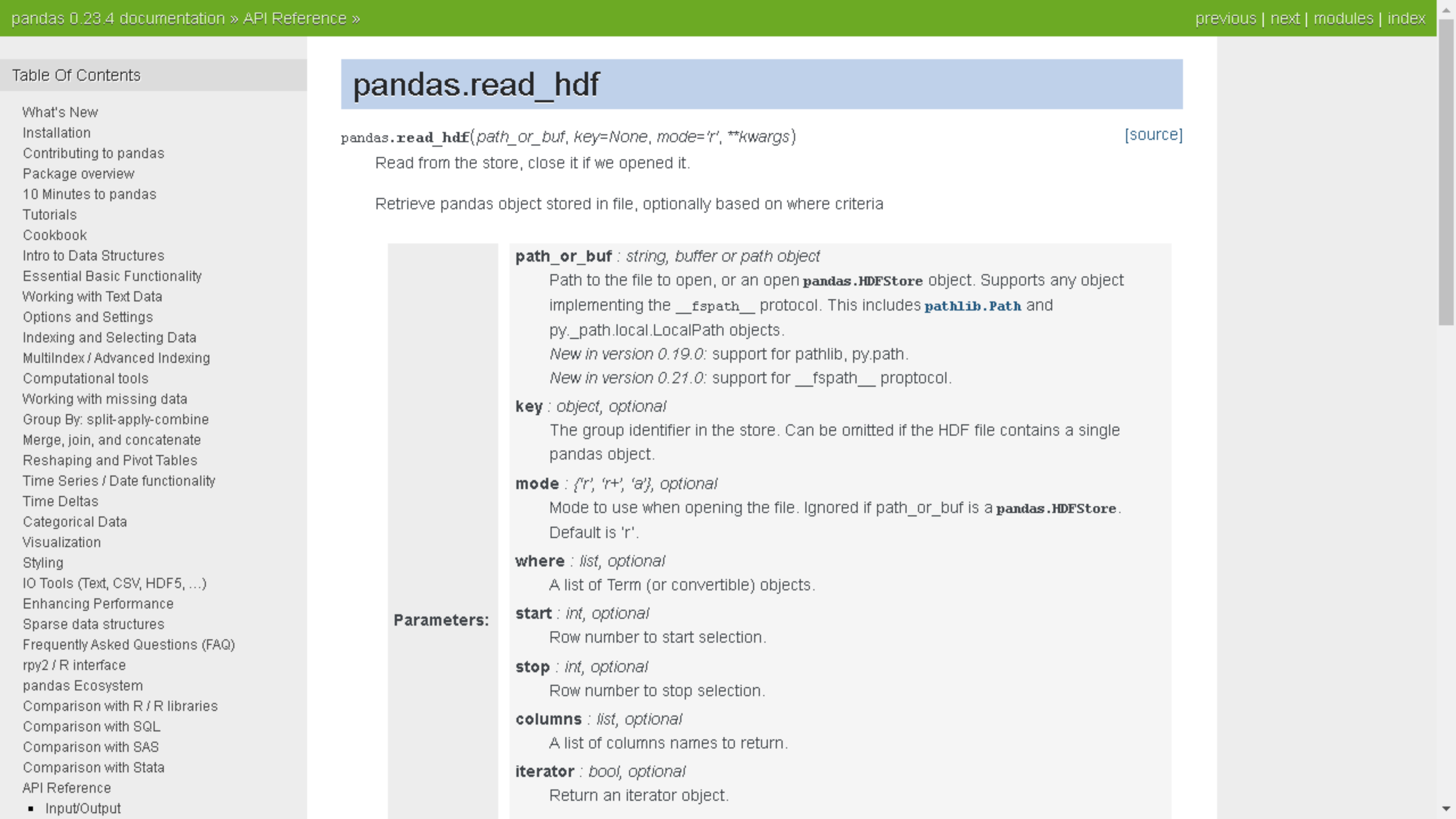


Table Of Contents

pandas.read_hdf

`pandas.read_hdf`(*path_or_buf*, *key=None*, *mode='r'*, ***kwargs*) [source]

Read from the store, close it if we opened it.

Retrieve pandas object stored in file, optionally based on where criteria

Parameters:

path_or_buf : *string, buffer or path object*

Path to the file to open, or an open `pandas.HDFStore` object. Supports any object implementing the `__fspath__` protocol. This includes `pathlib.Path` and py._path.local.LocalPath objects.

New in version 0.19.0: support for pathlib, py.path.

New in version 0.21.0: support for __fspath__ proptocol.

key : *object, optional*

The group identifier in the store. Can be omitted if the HDF file contains a single pandas object.

mode : *{'r', 'r+', 'a'}, optional*

Mode to use when opening the file. Ignored if path_or_buf is a `pandas.HDFStore`. Default is 'r'.

where : *list, optional*

A list of Term (or convertible) objects.

start : *int, optional*

Row number to start selection.

stop : *int, optional*

Row number to stop selection.

columns : *list, optional*

A list of columns names to return.

iterator : *bool, optional*

Return an iterator object.

MATLAB

MAtrix LABoratory

Intended primarily for numerical computing

Industry standard, proprietary

Read using SciPy

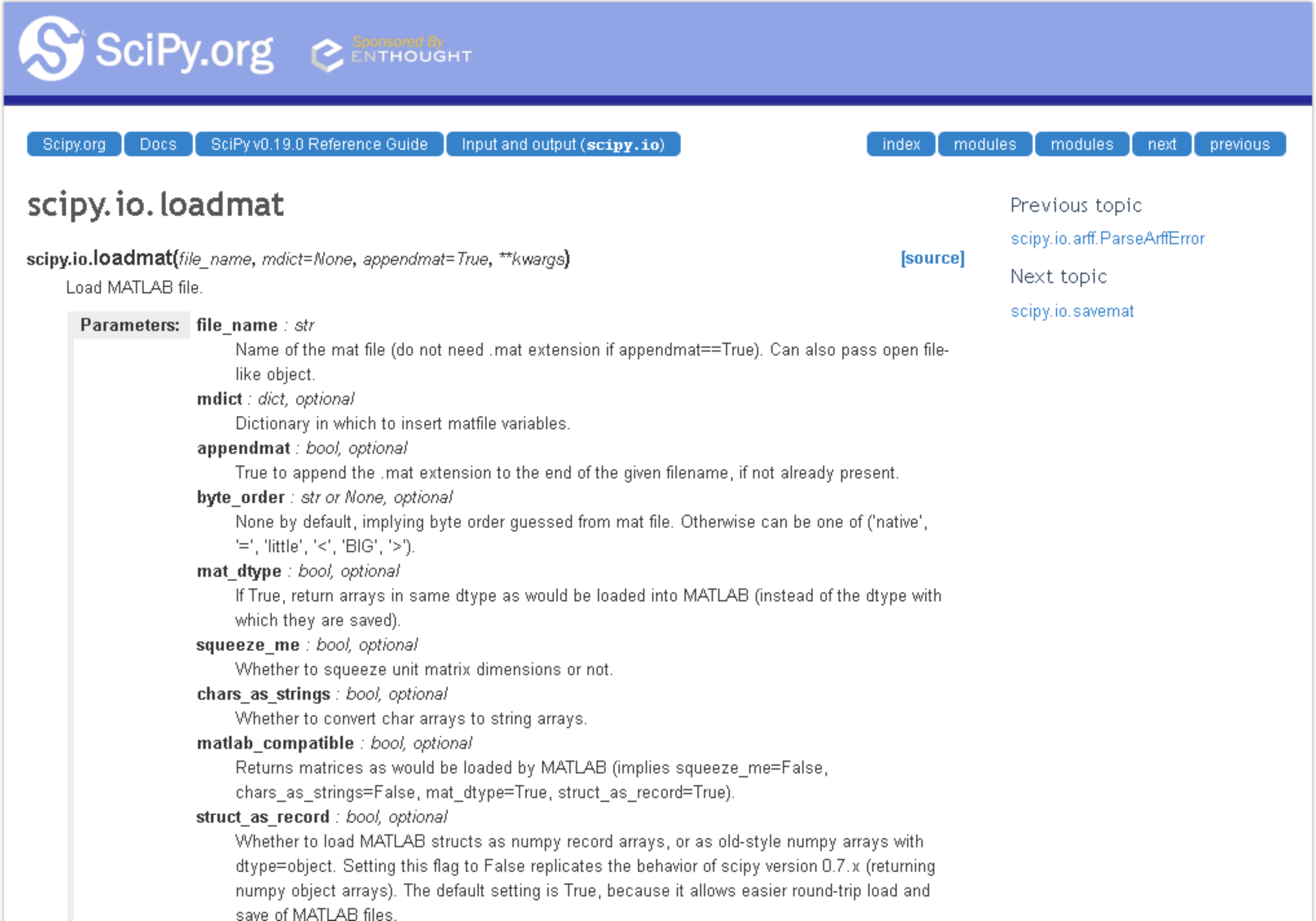
```
import scipy.io

posts_mat = scipy.io.loadmat('posts-100.mat')

type(posts_mat)

posts_mat.keys()

posts_mat['posts']
```

Reading Matlab Files

Import scipy.io

- Use loadmat
- **Get a dictionary**

Review the keys, and start working with your data



scipy.io.loadmat

scipy.io.loadmat(*file_name*, *mdict=None*, *appendmat=True*, ***kwargs*) [source]

Load MATLAB file.

Parameters:

file_name : *str*

Name of the mat file (do not need .mat extension if appendmat==True). Can also pass open file-like object.

mdict : *dict, optional*

Dictionary in which to insert matfile variables.

appendmat : *bool, optional*

True to append the .mat extension to the end of the given filename, if not already present.

byte_order : *str or None, optional*

None by default, implying byte order guessed from mat file. Otherwise can be one of ('native', '=', 'little', '<', 'BIG', '>').

mat_dtype : *bool, optional*

If True, return arrays in same dtype as would be loaded into MATLAB (instead of the dtype with which they are saved).

squeeze_me : *bool, optional*

Whether to squeeze unit matrix dimensions or not.

chars_as_strings : *bool, optional*

Whether to convert char arrays to string arrays.

matlab_compatible : *bool, optional*

Returns matrices as would be loaded by MATLAB (implies squeeze_me=False, chars_as_strings=False, mat_dtype=True, struct_as_record=True).

struct_as_record : *bool, optional*

Whether to load MATLAB structs as numpy record arrays, or as old-style numpy arrays with dtype=object. Setting this flag to False replicates the behavior of scipy version 0.7.x (returning numpy object arrays). The default setting is True, because it allows easier round-trip load and save of MATLAB files.

Previous topic

scipy.io.arff.ParseArffError

Next topic

scipy.io.savemat

Pickle

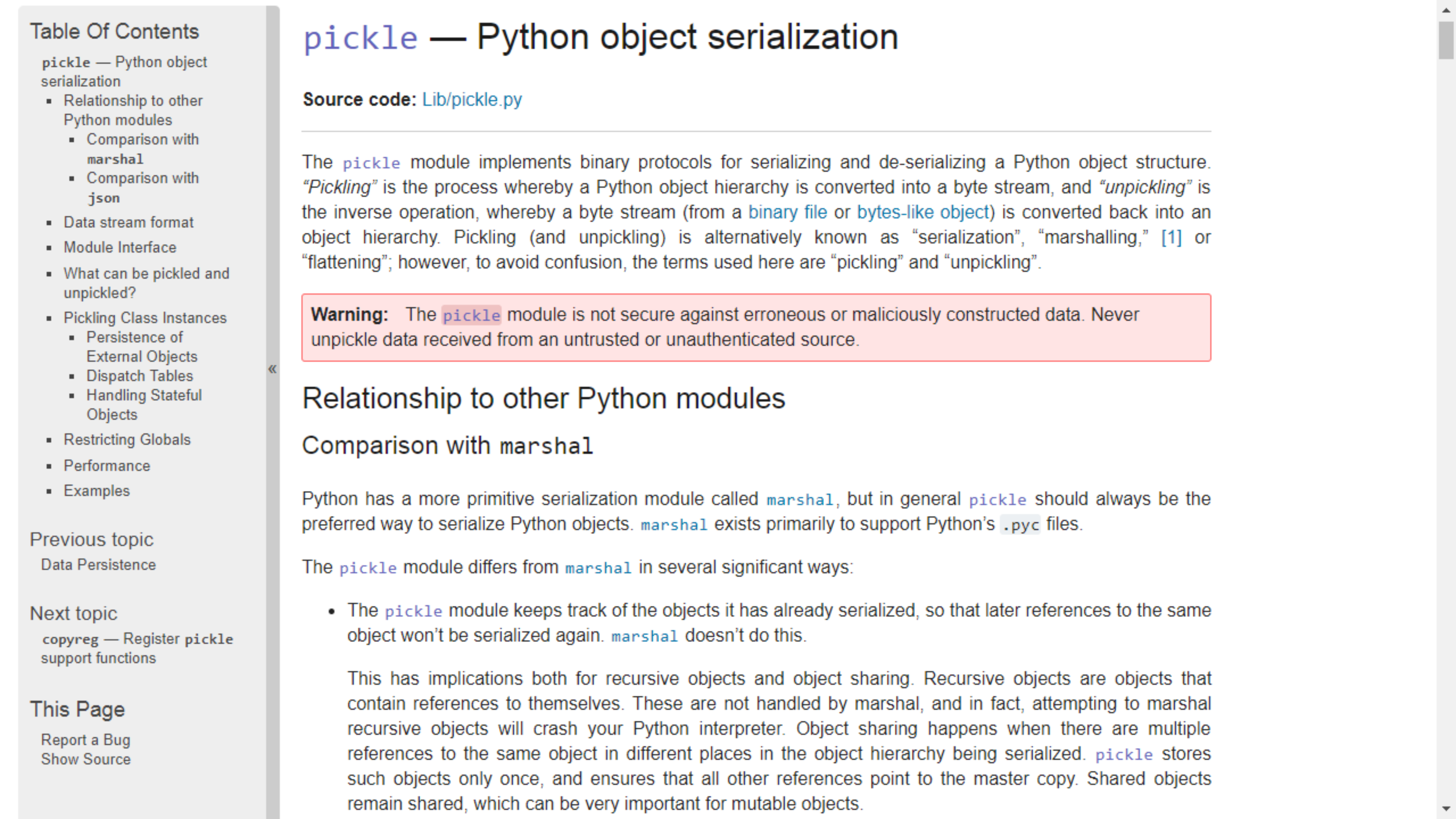
Serializing and deserializing objects

Convenient, binary format

Python specific

Read using

- Pickle module
- Pandas



Previous topic

Data Persistence

Next topic

copyreg — Register pickle support functions

This Page

Report a Bug
Show Source

pickle — Python object serialization

Source code: Lib/pickle.py

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Warning: The pickle module is not secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

Relationship to other Python modules

Comparison with marshal

Python has a more primitive serialization module called marshal, but in general pickle should always be the preferred way to serialize Python objects. marshal exists primarily to support Python's .pyc files.

The pickle module differs from marshal in several significant ways:

- The pickle module keeps track of the objects it has already serialized, so that later references to the same object won't be serialized again. marshal doesn't do this.

  This has implications both for recursive objects and object sharing. Recursive objects are objects that contain references to themselves. These are not handled by marshal, and in fact, attempting to marshal recursive objects will crash your Python interpreter. Object sharing happens when there are multiple references to the same object in different places in the object hierarchy being serialized. pickle stores such objects only once, and ensures that all other references point to the master copy. Shared objects remain shared, which can be very important for mutable objects.

```
import pickle

with open('posts-100.pkl.gz', 'rb') as pickle_file:

    posts_pickle = pickle.load(pickle_file)

type(posts_pickle)

posts_pickle.columns

posts_pickle.head()
```

Reading Pickle Files Using the pickle Module

Use the pickle module

- Open the file, using rb.
- We get our DataFrame, supports compression

Work with the objects you serialized

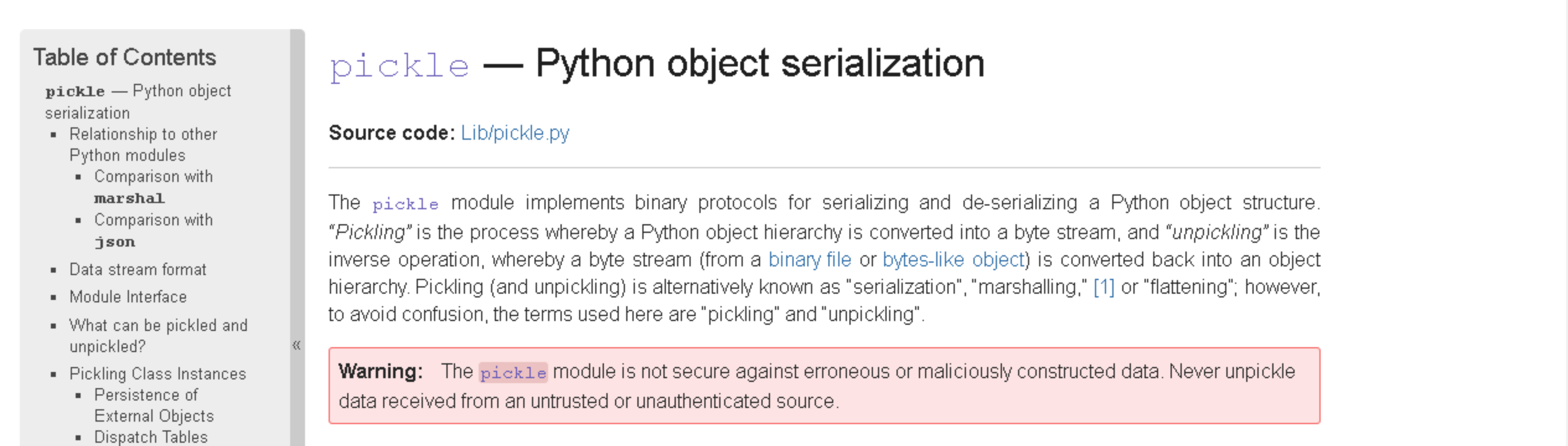
```
import pandas as pd

posts_pickle = pd.read_pickle('posts-100.pkl')

type(posts_pickle)

posts_pickle.columns

posts_pickle.head()
```

Reading Pickle Files Using Pandas

Load using read_pickle

- Get your DataFrame

Work with your data

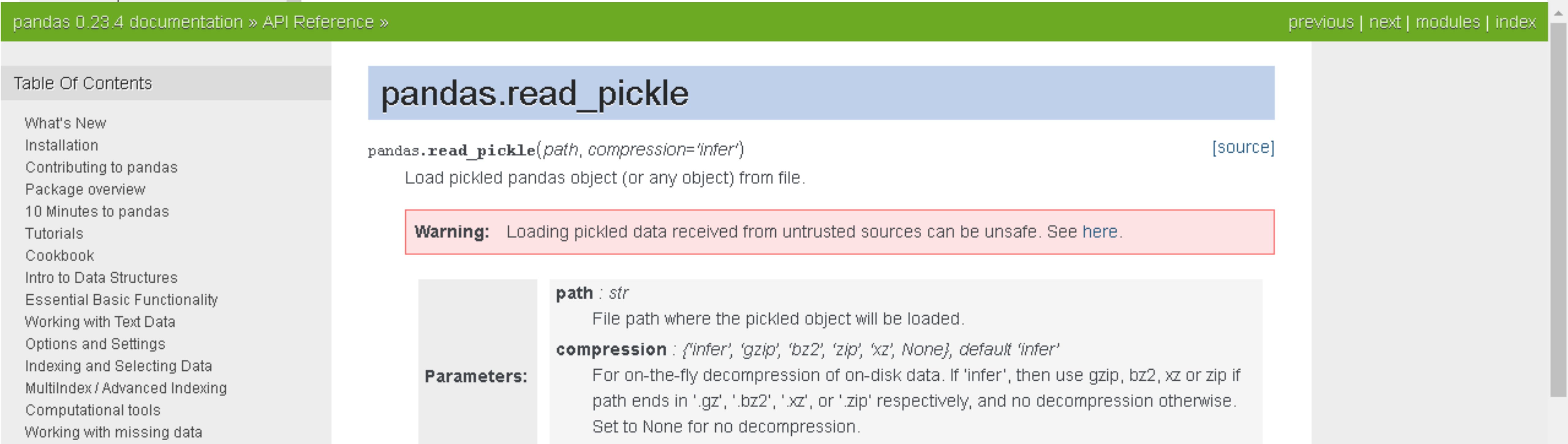


`pickle` — Python object serialization

Source code: Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Warning: The `pickle` module is not secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.



pandas 0.23.4 documentation » API Reference »

previous | next | modules | index

pandas.read_pickle

`pandas.read_pickle`(*path*, *compression='infer'*) [source]

Load pickled pandas object (or any object) from file.

Warning: Loading pickled data received from untrusted sources can be unsafe. See here.

Parameters:	**path** : *str* File path where the pickled object will be loaded. **compression** : *{'infer', 'gzip', 'bz2', 'zip', 'xz', None}, default 'infer'* For on-the-fly decompression of on-disk data. If 'infer', then use gzip, bz2, xz or zip if path ends in '.gz', '.bz2', '.xz', or '.zip' respectively, and no decompression otherwise. Set to None for no decompression.