Search the docs ...

Input/output
pandas.read_pickle
pandas.DataFrame.to_pickle
pandas.read_table
pandas.read_csv
pandas.DataFrame.to_csv
pandas.read_fwf
pandas.read_clipboard
pandas.DataFrame.to_clipboard
pandas.read_excel
pandas.DataFrame.to_excel
pandas.ExcelFile.parse
pandas.io.formats.style.Styler.to_
pandas.ExcelWriter
pandas.ExcelWriter.book
pandas.ExcelWriter.cur_sheet
pandas.ExcelWriter.date_form
pandas.ExcelWriter.datetime_
pandas.ExcelWriter.engine
pandas.ExcelWriter.handles
pandas.ExcelWriter.if_sheet_e
pandas.ExcelWriter.path
pandas.ExcelWriter.sheets
pandas.ExcelWriter.supported
pandas.ExcelWriter.check_ext
pandas.ExcelWriter.close
pandas.ExcelWriter.save
pandas.ExcelWriter.write_cells
pandas.read_json
pandas.json_normalize
pandas.DataFrame.to_json
pandas.io.json.build_table_schem
pandas.read_html
pandas.DataFrame.to_html
pandas.io.formats.style.Styler.to_l
pandas.read_xml
pandas.DataFrame.to_xml
pandas.DataFrame.to_latex
pandas.io.formats.style.Styler.to_l
pandas.read_hdf
pandas.HDFStore.put
pandas.HDFStore.append
pandas.HDFStore.get
pandas.HDFStore.select

pandas.ExcelWriter

class **pandas.ExcelWriter**(*path*, *engine=None*, *date_format=None*, *datetime_format=None*, *mode='w'*, *storage_options=None*, *if_sheet_exists=None*, *engine_kwargs=None*, ***kwargs*) [source]

Class for writing DataFrame objects into excel sheets.

Default is to use : * xlwt for xls * xlsxwriter for xlsx if xlsxwriter is installed otherwise openpyxl * odf for ods. See DataFrame.to_excel for typical usage.

The writer should be used as a context manager. Otherwise, call *close()* to save and close any opened file handles.

Parameters: **path : *str or typing.BinaryIO***

Path to xls or xlsx or ods file.

engine : *str (optional)*

Engine to use for writing. If None, defaults to `io.excel.<extension>.writer`. NOTE: can only be passed as a keyword argument.

> ***Deprecated since version 1.2.0:*** As the xlwt package is no longer maintained, the `xlwt` engine will be removed in a future version of pandas.

date_format : *str, default None*

Format string for dates written into Excel files (e.g. ‘YYYY-MM-DD’).

datetime_format : *str, default None*

Format string for datetime objects written into Excel files. (e.g. ‘YYYY-MM-DD HH:MM:SS’).

mode : *{‘w’, ‘a’}, default ‘w’*

File mode to use (write or append). Append does not work with fsspec URLs.

storage_options : *dict, optional*

Extra options that make sense for a particular storage connection, e.g. host, port, username, password, etc., if using a URL that will be parsed by `fsspec`, e.g., starting “s3://”, “gcs://”.

> ***New in version 1.2.0.***

if_sheet_exists : *{‘error’, ‘new’, ‘replace’, ‘overlay’}, default ‘error’*

How to behave when trying to write to a sheet that already exists (append mode only).

- error: raise a ValueError.
- new: Create a new sheet, with a name determined by the engine.
- replace: Delete the contents of the sheet before writing to it.
- overlay: Write contents to the existing sheet without removing the old contents.

> ***New in version 1.3.0.***

> ***Changed in version 1.4.0:*** Added `overlay` option

engine_kwargs : *dict, optional*

Keyword arguments to be passed into the engine. These will be passed to the following functions of the respective engines:

- xlsxwriter: `xlsxwriter.Workbook(file, **engine_kwargs)`
- openpyxl (write mode): `openpyxl.Workbook(**engine_kwargs)`
- openpyxl (append mode): `openpyxl.load_workbook(file, **engine_kwargs)`
- odswriter: `odf.opendocument.OpenDocumentSpreadsheet(**engine_kwargs)`

> ***New in version 1.3.0.***

****kwargs : *dict, optional***

Keyword arguments to be passed into the engine.

> ***Deprecated since version 1.3.0:*** Use engine_kwargs instead.

Notes

For compatibility with CSV writers, ExcelWriter serializes lists and dicts to strings before writing.

Examples

Default usage:

```
>>> df = pd.DataFrame([["ABC", "XYZ"]], columns=["Foo", "Bar"])
>>> with pd.ExcelWriter("path_to_file.xlsx") as writer:
...     df.to_excel(writer)
```

To write to separate sheets in a single file:

```
>>> df1 = pd.DataFrame([["AAA", "BBB"]], columns=["Spam", "Egg"])
>>> df2 = pd.DataFrame([["ABC", "XYZ"]], columns=["Foo", "Bar"])
>>> with pd.ExcelWriter("path_to_file.xlsx") as writer:
...     df1.to_excel(writer, sheet_name="Sheet1")
...     df2.to_excel(writer, sheet_name="Sheet2")
```

You can set the date format or datetime format:

```
>>> from datetime import date, datetime
>>> df = pd.DataFrame(
...     [
...         [date(2014, 1, 31), date(1999, 9, 24)],
...         [datetime(1998, 5, 26, 23, 33, 4), datetime(2014, 2, 28, 13, 5, 13)],
...     ],
...     index=["Date", "Datetime"],
...     columns=["X", "Y"],
... )
>>> with pd.ExcelWriter(
...     "path_to_file.xlsx",
...     date_format="YYYY-MM-DD",
...     datetime_format="YYYY-MM-DD HH:MM:SS"
... ) as writer:
...     df.to_excel(writer)
```

You can also append to an existing Excel file:

```
>>> with pd.ExcelWriter("path_to_file.xlsx", mode="a", engine="openpyxl") as writer:
...     df.to_excel(writer, sheet_name="Sheet3")
```

Here, the *if_sheet_exists* parameter can be set to replace a sheet if it already exists:

```
>>> with ExcelWriter(
...     "path_to_file.xlsx",
...     mode="a",
...     engine="openpyxl",
...     if_sheet_exists="replace",
... ) as writer:
...     df.to_excel(writer, sheet_name="Sheet1")
```

You can also write multiple DataFrames to a single sheet. Note that the `if_sheet_exists` parameter needs to be set to `overlay`:

```
>>> with ExcelWriter("path_to_file.xlsx",
...     mode="a",
...     engine="openpyxl",
...     if_sheet_exists="overlay",
... ) as writer:
...     df1.to_excel(writer, sheet_name="Sheet1")
...     df2.to_excel(writer, sheet_name="Sheet1", startcol=3)
```

You can store Excel file in RAM:

```
>>> import io
>>> df = pd.DataFrame([["ABC", "XYZ"]], columns=["Foo", "Bar"])
>>> buffer = io.BytesIO()
>>> with pd.ExcelWriter(buffer) as writer:
...     df.to_excel(writer)
```

You can pack Excel file into zip archive:

```
>>> import zipfile
>>> df = pd.DataFrame([["ABC", "XYZ"]], columns=["Foo", "Bar"])
>>> with zipfile.ZipFile("path_to_file.zip", "w") as zf:
...     with zf.open("filename.xlsx", "w") as buffer:
...         with pd.ExcelWriter(buffer) as writer:
...             df.to_excel(writer)
```

You can specify additional arguments to the underlying engine:

```
>>> with pd.ExcelWriter(
...     "path_to_file.xlsx",
...     engine="xlsxwriter",
...     engine_kwargs={"options": {"nan_inf_to_errors": True}}
... ) as writer:
...     df.to_excel(writer)
```

In append mode, `engine_kwargs` are passed through to openpyxl's `load_workbook`:

```
>>> with pd.ExcelWriter(
...     "path_to_file.xlsx",
...     engine="openpyxl",
...     mode="a",
...     engine_kwargs={"keep_vba": True}
... ) as writer:
...     df.to_excel(writer, sheet_name="Sheet2")
```

Attributes

book	Book instance.
cur_sheet	(DEPRECATED) Current sheet for writing.
date_format	Format string for dates written into Excel files (e.g.
datetime_format	Format string for dates written into Excel files (e.g.
engine	Name of engine.
handles	(DEPRECATED) Handles to Excel sheets.
if_sheet_exists	How to behave when writing to a sheet that already exists in append mode.
path	(DEPRECATED) Path to Excel file.
sheets	Mapping of sheet names to sheet objects.
supported_extensions	Extensions that writer engine supports.

Methods

`check_extension`(ext)	checks that path's extension against the Writer's supported extensions.
`close`()	synonym for save, to make it more file-like
`save`()	(DEPRECATED) Save workbook to disk.
`write_cells`(cells[, sheet_name, startrow, ...])	(DEPRECATED) Write given formatted cells into Excel an excel sheet

Previous
pandas.io.formats.style.Styler.to_exc

Next
pandas.ExcelWriter.book

Created using Sphinx 4.3.2.