quoting [optional constant from csv module] Defaults to csv.QUOTE_MINIMAL. If you have set a *float_format* then floats are converted to strings and thus csv.QUOTE_NONNUMERIC will treat them as non-numeric.

quotechar [str, default '"'] String of length 1. Character used to quote fields.

line_terminator [str, optional] The newline character or character sequence to use in the output file. Defaults to *os.linesep*, which depends on the OS in which this method is called ('n' for linux, 'rn' for Windows, i.e.).

Changed in version 0.24.0.

chunksize [int or None] Rows to write at a time.

date_format [str, default None] Format string for datetime objects.

doublequote [bool, default True] Control quoting of *quotechar* inside a field.

escapechar [str, default None] String of length 1. Character used to escape *sep* and *quotechar* when appropriate.

decimal [str, default '.'] Character recognized as decimal separator. E.g. use ',' for European data.

Returns

None or str If path_or_buf is None, returns the resulting csv format as a string. Otherwise returns None.

See also:

read_csv Load a CSV file into a DataFrame.

to_excel Write DataFrame to an Excel file.

Examples

```
>>> df = pd.DataFrame({'name': ['Raphael', 'Donatello'],
...                    'mask': ['red', 'purple'],
...                    'weapon': ['sai', 'bo staff']})
>>> df.to_csv(index=False)
'name,mask,weapon\nRaphael,red,sai\nDonatello,purple,bo staff\n'
```

Create 'out.zip' containing 'out.csv'

```
>>> compression_opts = dict(method='zip',
...                         archive_name='out.csv')
>>> df.to_csv('out.zip', index=False,
...           compression=compression_opts)
```

pandas.DataFrame.to_dict

DataFrame.**to_dict**(*self*, *orient='dict'*, *into=<class 'dict'>*)

Convert the DataFrame to a dictionary.

The type of the key-value pairs can be customized with the parameters (see below).

Parameters

orient [str {'dict', 'list', 'series', 'split', 'records', 'index'}] Determines the type of the values of the dictionary.

- 'dict' (default) : dict like {column -> {index -> value}}
- 'list' : dict like {column -> [values]}
- 'series' : dict like {column -> Series(values)}
- 'split' : dict like {'index' -> [index], 'columns' -> [columns], 'data' -> [values]}
- 'records' : list like [{column -> value}, ... , {column -> value}]
- 'index' : dict like {index -> {column -> value}}

Abbreviations are allowed. *s* indicates *series* and *sp* indicates *split*.

into [class, default dict] The collections.abc.Mapping subclass used for all Mappings in the return value. Can be the actual class or an empty instance of the mapping type you want. If you want a collections.defaultdict, you must pass it initialized.

New in version 0.21.0.

Returns

dict, list or collections.abc.Mapping Return a collections.abc.Mapping object representing the DataFrame. The resulting transformation depends on the *orient* parameter.

See also:

DataFrame.from_dict Create a DataFrame from a dictionary.

DataFrame.to_json Convert a DataFrame to JSON format.

Examples

```
>>> df = pd.DataFrame({'col1': [1, 2],
...                    'col2': [0.5, 0.75]},
...                   index=['row1', 'row2'])
>>> df
      col1  col2
row1     1  0.50
row2     2  0.75
>>> df.to_dict()
{'col1': {'row1': 1, 'row2': 2}, 'col2': {'row1': 0.5, 'row2': 0.75}}
```

You can specify the return orientation.

```
>>> df.to_dict('series')
{'col1': row1    1
         row2    2
Name: col1, dtype: int64,
'col2': row1    0.50
        row2    0.75
Name: col2, dtype: float64}
```

```
>>> df.to_dict('split')
{'index': ['row1', 'row2'], 'columns': ['col1', 'col2'],
 'data': [[1, 0.5], [2, 0.75]]}
```

```
>>> df.to_dict('records')
[{'col1': 1, 'col2': 0.5}, {'col1': 2, 'col2': 0.75}]
```

```
>>> df.to_dict('index')
{'row1': {'col1': 1, 'col2': 0.5}, 'row2': {'col1': 2, 'col2': 0.75}}
```

You can also specify the mapping type.

```
>>> from collections import OrderedDict, defaultdict
>>> df.to_dict(into=OrderedDict)
OrderedDict([('col1', OrderedDict([('row1', 1), ('row2', 2)])),
             ('col2', OrderedDict([('row1', 0.5), ('row2', 0.75)]))])
```

If you want a *defaultdict*, you need to initialize it:

```
>>> dd = defaultdict(list)
>>> df.to_dict('records', into=dd)
[defaultdict(<class 'list'>, {'col1': 1, 'col2': 0.5}),
 defaultdict(<class 'list'>, {'col1': 2, 'col2': 0.75})]
```

pandas.DataFrame.to_excel

DataFrame.**to_excel**(*self*, *excel_writer*, *sheet_name='Sheet1'*, *na_rep=''*, *float_format=None*, *columns=None*, *header=True*, *index=True*, *index_label=None*, *startrow=0*, *startcol=0*, *engine=None*, *merge_cells=True*, *encoding=None*, *inf_rep='inf'*, *verbose=True*, *freeze_panes=None*) → None

Write object to an Excel sheet.

To write a single object to an Excel .xlsx file it is only necessary to specify a target file name. To write to multiple sheets it is necessary to create an *ExcelWriter* object with a target file name, and specify a sheet in the file to write to.

Multiple sheets may be written to by specifying unique *sheet_name*. With all data written to the file it is necessary to save the changes. Note that creating an *ExcelWriter* object with a file name that already exists will result in the contents of the existing file being erased.

Parameters

excel_writer [str or ExcelWriter object] File path or existing ExcelWriter.

sheet_name [str, default 'Sheet1'] Name of sheet which will contain DataFrame.

na_rep [str, default ''] Missing data representation.

float_format [str, optional] Format string for floating point numbers. For example `float_format="%.2f"` will format 0.1234 to 0.12.

columns [sequence or list of str, optional] Columns to write.

header [bool or list of str, default True] Write out the column names. If a list of string is given it is assumed to be aliases for the column names.

index [bool, default True] Write row names (index).

index_label [str or sequence, optional] Column label for index column(s) if desired. If not specified, and *header* and *index* are True, then the index names are used. A sequence should be given if the DataFrame uses MultiIndex.

startrow [int, default 0] Upper left cell row to dump data frame.

startcol [int, default 0] Upper left cell column to dump data frame.

engine [str, optional] Write engine to use, 'openpyxl' or 'xlsxwriter'. You can also set this via the options `io.excel.xlsx.writer`, `io.excel.xls.writer`, and `io.excel.xlsm.writer`.

merge_cells [bool, default True] Write MultiIndex and Hierarchical Rows as merged cells.

encoding [str, optional] Encoding of the resulting excel file. Only necessary for xlwt, other writers support unicode natively.

inf_rep [str, default 'inf'] Representation for infinity (there is no native representation for infinity in Excel).

verbose [bool, default True] Display more information in the error logs.

freeze_panes [tuple of int (length 2), optional] Specifies the one-based bottommost row and rightmost column that is to be frozen.

See also:

to_csv Write DataFrame to a comma-separated values (csv) file.

ExcelWriter Class for writing DataFrame objects into excel sheets.

read_excel Read an Excel file into a pandas DataFrame.

read_csv Read a comma-separated values (csv) file into DataFrame.

Notes

For compatibility with *to_csv()*, to_excel serializes lists and dicts to strings before writing.

Once a workbook has been saved it is not possible write further data without rewriting the whole workbook.

Examples

Create, write to and save a workbook:

```
>>> df1 = pd.DataFrame([['a', 'b'], ['c', 'd']],
...                    index=['row 1', 'row 2'],
...                    columns=['col 1', 'col 2'])
>>> df1.to_excel("output.xlsx")
```

To specify the sheet name:

```
>>> df1.to_excel("output.xlsx",
...              sheet_name='Sheet_name_1')
```

If you wish to write to more than one sheet in the workbook, it is necessary to specify an ExcelWriter object:

```
>>> df2 = df1.copy()
>>> with pd.ExcelWriter('output.xlsx') as writer:
...     df1.to_excel(writer, sheet_name='Sheet_name_1')
...     df2.to_excel(writer, sheet_name='Sheet_name_2')
```

ExcelWriter can also be used to append to an existing Excel file:

```
>>> with pd.ExcelWriter('output.xlsx',
...                     mode='a') as writer:
...     df.to_excel(writer, sheet_name='Sheet_name_3')
```

To set the library that is used to write the Excel file, you can pass the *engine* keyword (the default engine is automatically chosen depending on the file extension):

```
>>> df1.to_excel('output1.xlsx', engine='xlsxwriter')
```

pandas.DataFrame.to_feather

DataFrame.**to_feather**(*self*, *path*) → None

Write out the binary feather-format for DataFrames.

Parameters

path [str] String file path.

pandas.DataFrame.to_gbq

DataFrame.**to_gbq**(*self*, *destination_table*, *project_id=None*, *chunksize=None*, *reauth=False*, *if_exists='fail'*, *auth_local_webserver=False*, *table_schema=None*, *location=None*, *progress_bar=True*, *credentials=None*) → None

Write a DataFrame to a Google BigQuery table.

This function requires the pandas-gbq package.

See the How to authenticate with Google BigQuery guide for authentication instructions.

Parameters

destination_table [str] Name of table to be written, in the form `dataset.tablename`.

project_id [str, optional] Google BigQuery Account project ID. Optional when available from the environment.

chunksize [int, optional] Number of rows to be inserted in each chunk from the dataframe. Set to `None` to load the whole dataframe at once.

reauth [bool, default False] Force Google BigQuery to re-authenticate the user. This is useful if multiple accounts are used.

if_exists [str, default 'fail'] Behavior when the destination table exists. Value can be one of:

`'fail'` If table exists raise pandas_gbq.gbq.TableCreationError.

`'replace'` If table exists, drop it, recreate it, and insert data.

`'append'` If table exists, insert data. Create if does not exist.

auth_local_webserver [bool, default False] Use the local webserver flow instead of the console flow when getting user credentials.

New in version 0.2.0 of pandas-gbq.

table_schema [list of dicts, optional] List of BigQuery table fields to which according DataFrame columns conform to, e.g. `[{'name': 'col1', 'type': 'STRING'},...]`. If schema is not provided, it will be generated according to dtypes of DataFrame columns. See BigQuery API documentation on available names of a field.

New in version 0.3.1 of pandas-gbq.

location [str, optional] Location where the load job should run. See the BigQuery locations documentation for a list of available locations. The location must match that of the target dataset.

New in version 0.5.0 of pandas-gbq.

progress_bar [bool, default True] Use the library *tqdm* to show the progress bar for the upload, chunk by chunk.

New in version 0.5.0 of pandas-gbq.

credentials [google.auth.credentials.Credentials, optional] Credentials for accessing Google APIs. Use this parameter to override default credentials, such as to use Compute Engine `google.auth.compute_engine.Credentials` or Service Account `google.oauth2.service_account.Credentials` directly.

New in version 0.8.0 of pandas-gbq.

New in version 0.24.0.

See also:

`pandas_gbq.to_gbq` This function in the pandas-gbq library.

`read_gbq` Read a DataFrame from Google BigQuery.

pandas.DataFrame.to_hdf

DataFrame.**to_hdf**(*self*, *path_or_buf*, *key: str*, *mode: str = 'a'*, *complevel: Union[int, NoneType] = None*, *complib: Union[str, NoneType] = None*, *append: bool = False*, *format: Union[str, NoneType] = None*, *index: bool = True*, *min_itemsize: Union[int, Dict[str, int], NoneType] = None*, *nan_rep=None*, *dropna: Union[bool, NoneType] = None*, *data_columns: Union[List[str], NoneType] = None*, *errors: str = 'strict'*, *encoding: str = 'UTF-8'*) → None

Write the contained data to an HDF5 file using HDFStore.

Hierarchical Data Format (HDF) is self-describing, allowing an application to interpret the structure and contents of a file with no outside information. One HDF file can hold a mix of related objects which can be accessed as a group or as individual objects.

In order to add another DataFrame or Series to an existing HDF file please use append mode and a different a key.

For more information see the *user guide*.

Parameters

path_or_buf [str or pandas.HDFStore] File path or HDFStore object.

key [str] Identifier for the group in the store.

mode [{'a', 'w', 'r+'}, default 'a'] Mode to open file:

- 'w': write, a new file is created (an existing file with the same name would be deleted).
- 'a': append, an existing file is opened for reading and writing, and if the file does not exist it is created.
- 'r+': similar to 'a', but the file must already exist.

complevel [{0-9}, optional] Specifies a compression level for data. A value of 0 disables compression.

complib [{'zlib', 'lzo', 'bzip2', 'blosc'}, default 'zlib'] Specifies the compression library to be used. As of v0.20.2 these additional compressors for Blosc are supported (default if no compressor specified: 'blosc:blosclz'): {'blosc:blosclz', 'blosc:lz4', 'blosc:lz4hc', 'blosc:snappy', 'blosc:zlib', 'blosc:zstd'}. Specifying a compression library which is not available issues a ValueError.

append [bool, default False] For Table formats, append the input data to the existing.

format [{'fixed', 'table', None}, default 'fixed'] Possible values:

- 'fixed': Fixed format. Fast writing/reading. Not-appendable, nor searchable.
- 'table': Table format. Write as a PyTables Table structure which may perform worse but allow more flexible operations like searching / selecting subsets of the data.
- If None, pd.get_option('io.hdf.default_format') is checked, followed by fallback to "fixed"

errors [str, default 'strict'] Specifies how encoding and decoding errors are to be handled. See the errors argument for `open()` for a full list of options.

encoding [str, default "UTF-8"]

min_itemsize [dict or int, optional] Map column names to minimum string sizes for columns.

nan_rep [Any, optional] How to represent null values as str. Not allowed with append=True.

data_columns [list of columns or True, optional] List of columns to create as indexed data columns for on-disk queries, or True to use all columns. By default only the axes of the object are indexed. See *Query via data columns*. Applicable only to format='table'.

See also:

DataFrame.read_hdf Read from HDF file.

DataFrame.to_parquet Write a DataFrame to the binary parquet format.

DataFrame.to_sql Write to a sql table.

DataFrame.to_feather Write out feather-format for DataFrames.

DataFrame.to_csv Write out to a csv file.

Examples

```
>>> df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]},
...                   index=['a', 'b', 'c'])
>>> df.to_hdf('data.h5', key='df', mode='w')
```

We can add another object to the same file:

```
>>> s = pd.Series([1, 2, 3, 4])
>>> s.to_hdf('data.h5', key='s')
```

Reading from HDF file:

```
>>> pd.read_hdf('data.h5', 'df')
A  B
a  1  4
b  2  5
c  3  6
>>> pd.read_hdf('data.h5', 's')
0    1
1    2
2    3
3    4
dtype: int64
```

Deleting file with data:

```
>>> import os
>>> os.remove('data.h5')
```

pandas.DataFrame.to_html

DataFrame.**to_html**(*self*, *buf=None*, *columns=None*, *col_space=None*, *header=True*, *index=True*, *na_rep='NaN'*, *formatters=None*, *float_format=None*, *sparsify=None*, *index_names=True*, *justify=None*, *max_rows=None*, *max_cols=None*, *show_dimensions=False*, *decimal='.'*, *bold_rows=True*, *classes=None*, *escape=True*, *notebook=False*, *border=None*, *table_id=None*, *render_links=False*, *encoding=None*)

Render a DataFrame as an HTML table.

Parameters

buf [str, Path or StringIO-like, optional, default None] Buffer to write to. If None, the output is returned as a string.

columns [sequence, optional, default None] The subset of columns to write. Writes all columns by default.

col_space [str or int, optional] The minimum width of each column in CSS length units. An int is assumed to be px units.

New in version 0.25.0: Ability to use str.

header [bool, optional] Whether to print column labels, default True.

index [bool, optional, default True] Whether to print index (row) labels.

na_rep [str, optional, default 'NaN'] String representation of NAN to use.

formatters [list, tuple or dict of one-param. functions, optional] Formatter functions to apply to columns' elements by position or name. The result of each function must be a unicode string. List/tuple must be of length equal to the number of columns.

float_format [one-parameter function, optional, default None] Formatter function to apply to columns' elements if they are floats. The result of this function must be a unicode string.

sparsify [bool, optional, default True] Set to False for a DataFrame with a hierarchical index to print every multiindex key at each row.

index_names [bool, optional, default True] Prints the names of the indexes.

justify [str, default None] How to justify the column labels. If None uses the option from the print configuration (controlled by set_option), 'right' out of the box. Valid values are

- left
- right
- center
- justify
- justify-all
- start
- end
- inherit
- match-parent
- initial

- unset.

max_rows [int, optional] Maximum number of rows to display in the console.

min_rows [int, optional] The number of rows to display in the console in a truncated repr (when number of rows is above *max_rows*).

max_cols [int, optional] Maximum number of columns to display in the console.

show_dimensions [bool, default False] Display DataFrame dimensions (number of rows by number of columns).

decimal [str, default '.'] Character recognized as decimal separator, e.g. ',' in Europe.

bold_rows [bool, default True] Make the row labels bold in the output.

classes [str or list or tuple, default None] CSS class(es) to apply to the resulting html table.

escape [bool, default True] Convert the characters <, >, and & to HTML-safe sequences.

notebook [{True, False}, default False] Whether the generated HTML is for IPython Notebook.

border [int] A `border=border` attribute is included in the opening *<table>* tag. Default `pd.options.display.html.border`.

encoding [str, default "utf-8"] Set character encoding.

New in version 1.0.

table_id [str, optional] A css id is included in the opening *<table>* tag if specified.

New in version 0.23.0.

render_links [bool, default False] Convert URLs to HTML links.

New in version 0.24.0.

Returns

str or None If buf is None, returns the result as a string. Otherwise returns None.

See also:

to_string Convert DataFrame to a string.

pandas.DataFrame.to_json

DataFrame.**to_json**(*self*, *path_or_buf: Union[str, pathlib.Path, IO[~ AnyStr], NoneType] = None*, *orient: Union[str, NoneType] = None*, *date_format: Union[str, NoneType] = None*, *double_precision: int = 10*, *force_ascii: bool = True*, *date_unit: str = 'ms'*, *default_handler: Union[Callable[[Any], Union[str, int, float, bool, List, Dict]], NoneType] = None*, *lines: bool = False*, *compression: Union[str, NoneType] = 'infer'*, *index: bool = True*, *indent: Union[int, NoneType] = None*) → Union[str, NoneType]

Convert the object to a JSON string.

Note NaN's and None will be converted to null and datetime objects will be converted to UNIX timestamps.

Parameters

path_or_buf [str or file handle, optional] File path or object. If not specified, the result is returned as a string.

orient [str] Indication of expected JSON string format.

- Series:
  - default is 'index'
  - allowed values are: {'split','records','index','table'}.
- DataFrame:
  - default is 'columns'
  - allowed values are: {'split', 'records', 'index', 'columns', 'values', 'table'}.
- The format of the JSON string:
  - 'split' : dict like {'index' -> [index], 'columns' -> [columns], 'data' -> [values]}
  - 'records' : list like [{column -> value}, ... , {column -> value}]
  - 'index' : dict like {index -> {column -> value}}
  - 'columns' : dict like {column -> {index -> value}}
  - 'values' : just the values array
  - 'table' : dict like {'schema': {schema}, 'data': {data}}

    Describing the data, where data component is like `orient='records'`.

Changed in version 0.20.0.

date_format [{None, 'epoch', 'iso'}] Type of date conversion. 'epoch' = epoch milliseconds, 'iso' = ISO8601. The default depends on the *orient*. For `orient='table'`, the default is 'iso'. For all other orients, the default is 'epoch'.

double_precision [int, default 10] The number of decimal places to use when encoding floating point values.

force_ascii [bool, default True] Force encoded string to be ASCII.

date_unit [str, default 'ms' (milliseconds)] The time unit to encode to, governs timestamp and ISO8601 precision. One of 's', 'ms', 'us', 'ns' for second, millisecond, microsecond, and nanosecond respectively.

default_handler [callable, default None] Handler to call if object cannot otherwise be converted to a suitable format for JSON. Should receive a single argument which is the object to convert and return a serialisable object.

lines [bool, default False] If 'orient' is 'records' write out line delimited json format. Will throw ValueError if incorrect 'orient' since others are not list like.

compression [{'infer', 'gzip', 'bz2', 'zip', 'xz', None}] A string representing the compression to use in the output file, only used when the first argument is a filename. By default, the compression is inferred from the filename.

New in version 0.21.0.

Changed in version 0.24.0: 'infer' option added and set to default

index [bool, default True] Whether to include the index values in the JSON string. Not including the index (`index=False`) is only supported when orient is 'split' or 'table'.

New in version 0.23.0.

indent [int, optional] Length of whitespace used to indent each record.

New in version 1.0.0.

Returns

None or str If path_or_buf is None, returns the resulting json format as a string. Otherwise returns None.

See also:

read_json

Notes

The behavior of `indent=0` varies from the stdlib, which does not indent the output but does insert newlines. Currently, `indent=0` and the default `indent=None` are equivalent in pandas, though this may change in a future release.

Examples

```
>>> df = pd.DataFrame([['a', 'b'], ['c', 'd']],
...                   index=['row 1', 'row 2'],
...                   columns=['col 1', 'col 2'])
>>> df.to_json(orient='split')
'{"columns":["col 1","col 2"],
  "index":["row 1","row 2"],
  "data":[["a","b"],["c","d"]]}'
```

Encoding/decoding a Dataframe using `'records'` formatted JSON. Note that index labels are not preserved with this encoding.

```
>>> df.to_json(orient='records')
'[{"col 1":"a","col 2":"b"},{"col 1":"c","col 2":"d"}]'
```

Encoding/decoding a Dataframe using `'index'` formatted JSON:

```
>>> df.to_json(orient='index')
'{"row 1":{"col 1":"a","col 2":"b"},"row 2":{"col 1":"c","col 2":"d"}}'
```

Encoding/decoding a Dataframe using `'columns'` formatted JSON:

```
>>> df.to_json(orient='columns')
'{"col 1":{"row 1":"a","row 2":"c"},"col 2":{"row 1":"b","row 2":"d"}}'
```

Encoding/decoding a Dataframe using `'values'` formatted JSON:

```
>>> df.to_json(orient='values')
'[["a","b"],["c","d"]]'
```

Encoding with Table Schema

```
>>> df.to_json(orient='table')
'{"schema": {"fields": [{"name": "index", "type": "string"},
                        {"name": "col 1", "type": "string"},
                        {"name": "col 2", "type": "string"}],
              "primaryKey": "index",
              "pandas_version": "0.20.0"},
  "data": [{"index": "row 1", "col 1": "a", "col 2": "b"},
           {"index": "row 2", "col 1": "c", "col 2": "d"}]}'
```

pandas.DataFrame.to_latex

DataFrame.**to_latex**(*self*, *buf=None*, *columns=None*, *col_space=None*, *header=True*, *index=True*, *na_rep='NaN'*, *formatters=None*, *float_format=None*, *sparsify=None*, *index_names=True*, *bold_rows=False*, *column_format=None*, *longtable=None*, *escape=None*, *encoding=None*, *decimal='.'*, *multicolumn=None*, *multicolumn_format=None*, *multirow=None*, *caption=None*, *label=None*)

Render object to a LaTeX tabular, longtable, or nested table/tabular.

Requires `\usepackage{booktabs}`. The output can be copy/pasted into a main LaTeX document or read from an external file with `\input{table.tex}`.

Changed in version 0.20.2: Added to Series.

Changed in version 1.0.0: Added caption and label arguments.

Parameters

buf [str, Path or StringIO-like, optional, default None] Buffer to write to. If None, the output is returned as a string.

columns [list of label, optional] The subset of columns to write. Writes all columns by default.

col_space [int, optional] The minimum width of each column.

header [bool or list of str, default True] Write out the column names. If a list of strings is given, it is assumed to be aliases for the column names.

index [bool, default True] Write row names (index).

na_rep [str, default 'NaN'] Missing data representation.

formatters [list of functions or dict of {str: function}, optional] Formatter functions to apply to columns' elements by position or name. The result of each function must be a unicode string. List must be of length equal to the number of columns.

float_format [one-parameter function or str, optional, default None] Formatter for floating point numbers. For example `float_format="%.2f"` and `float_format="{:0.2f}".format` will both result in 0.1234 being formatted as 0.12.

sparsify [bool, optional] Set to False for a DataFrame with a hierarchical index to print every multiindex key at each row. By default, the value will be read from the config module.

index_names [bool, default True] Prints the names of the indexes.

bold_rows [bool, default False] Make the row labels bold in the output.

column_format [str, optional] The columns format as specified in LaTeX table format e.g. 'rcl' for 3 columns. By default, 'l' will be used for all columns except columns of numbers, which default to 'r'.

longtable [bool, optional] By default, the value will be read from the pandas config module. Use a longtable environment instead of tabular. Requires adding a usepackage{longtable} to your LaTeX preamble.

escape [bool, optional] By default, the value will be read from the pandas config module. When set to False prevents from escaping latex special characters in column names.

encoding [str, optional] A string representing the encoding to use in the output file, defaults to 'utf-8'.

decimal [str, default '.'] Character recognized as decimal separator, e.g. ',' in Europe.

multicolumn [bool, default True] Use multicolumn to enhance MultiIndex columns. The default will be read from the config module.

multicolumn_format [str, default 'l'] The alignment for multicolumns, similar to *column_format* The default will be read from the config module.

multirow [bool, default False] Use multirow to enhance MultiIndex rows. Requires adding a usepackage{multirow} to your LaTeX preamble. Will print centered labels (instead of top-aligned) across the contained rows, separating groups via clines. The default will be read from the pandas config module.

caption [str, optional] The LaTeX caption to be placed inside `\caption{}` in the output.

New in version 1.0.0.

label [str, optional] The LaTeX label to be placed inside `\label{}` in the output. This is used with `\ref{}` in the main `.tex` file.

New in version 1.0.0.

Returns

str or None If buf is None, returns the result as a string. Otherwise returns None.

See also:

DataFrame.to_string Render a DataFrame to a console-friendly tabular output.

DataFrame.to_html Render a DataFrame as an HTML table.

Examples

```
>>> df = pd.DataFrame({'name': ['Raphael', 'Donatello'],
...                    'mask': ['red', 'purple'],
...                    'weapon': ['sai', 'bo staff']})
>>> print(df.to_latex(index=False))
\begin{tabular}{lll}
 \toprule
       name &    mask &    weapon \\
 \midrule
    Raphael &     red &       sai \\
  Donatello &  purple &  bo staff \\
```

(continues on next page)

(continued from previous page)

```
\bottomrule
\end{tabular}
```

pandas.DataFrame.to_markdown

DataFrame.**to_markdown**(*self*, *buf: Union[IO[str], NoneType] = None*, *mode: Union[str, NoneType] = None*, ***kwargs*) → Union[str, NoneType]

Print DataFrame in Markdown-friendly format.

New in version 1.0.0.

Parameters

buf [writable buffer, defaults to sys.stdout] Where to send the output. By default, the output is printed to sys.stdout. Pass a writable buffer if you need to further process the output.

mode [str, optional] Mode in which file is opened.

****kwargs** These parameters will be passed to *tabulate*.

Returns

str DataFrame in Markdown-friendly format.

Examples

```
>>> df = pd.DataFrame(
...     data={"animal_1": ["elk", "pig"], "animal_2": ["dog", "quetzal"]}
... )
>>> print(df.to_markdown())
|    | animal_1   | animal_2   |
|---:|:-----------|:-----------|
|  0 | elk        | dog        |
|  1 | pig        | quetzal    |
```

pandas.DataFrame.to_numpy

DataFrame.**to_numpy**(*self*, *dtype=None*, *copy=False*) → numpy.ndarray

Convert the DataFrame to a NumPy array.

New in version 0.24.0.

By default, the dtype of the returned array will be the common NumPy dtype of all types in the DataFrame. For example, if the dtypes are `float16` and `float32`, the results dtype will be `float32`. This may require copying data and coercing values, which may be expensive.

Parameters

dtype [str or numpy.dtype, optional] The dtype to pass to `numpy.asarray()`.

copy [bool, default False] Whether to ensure that the returned value is a not a view on another array. Note that `copy=False` does not *ensure* that `to_numpy()` is no-copy. Rather, `copy=True` ensure that a copy is made, even if not strictly necessary.

Returns

numpy.ndarray

See also:

Series.to_numpy Similar method for Series.

Examples

```
>>> pd.DataFrame({"A": [1, 2], "B": [3, 4]}).to_numpy()
array([[1, 3],
       [2, 4]])
```

With heterogeneous data, the lowest common type will have to be used.

```
>>> df = pd.DataFrame({"A": [1, 2], "B": [3.0, 4.5]})
>>> df.to_numpy()
array([[1. , 3. ],
       [2. , 4.5]])
```

For a mix of numeric and non-numeric types, the output array will have object dtype.

```
>>> df['C'] = pd.date_range('2000', periods=2)
>>> df.to_numpy()
array([[1, 3.0, Timestamp('2000-01-01 00:00:00')],
       [2, 4.5, Timestamp('2000-01-02 00:00:00')]], dtype=object)
```

pandas.DataFrame.to_parquet

DataFrame.**to_parquet**(*self*, *path*, *engine='auto'*, *compression='snappy'*, *index=None*, *partition_cols=None*, ***kwargs*) → None

Write a DataFrame to the binary parquet format.

New in version 0.21.0.

This function writes the dataframe as a parquet file. You can choose different parquet backends, and have the option of compression. See *the user guide* for more details.

Parameters

path [str] File path or Root Directory path. Will be used as Root Directory path while writing a partitioned dataset.

Changed in version 1.0.0.

Previously this was "fname"

engine [{'auto', 'pyarrow', 'fastparquet'}, default 'auto'] Parquet library to use. If 'auto', then the option `io.parquet.engine` is used. The default `io.parquet.engine` behavior is to try 'pyarrow', falling back to 'fastparquet' if 'pyarrow' is unavailable.

compression [{'snappy', 'gzip', 'brotli', None}, default 'snappy'] Name of the compression to use. Use `None` for no compression.

index [bool, default None] If `True`, include the dataframe's index(es) in the file output. If `False`, they will not be written to the file. If `None`, similar to `True` the

dataframe's index(es) will be saved. However, instead of being saved as values, the RangeIndex will be stored as a range in the metadata so it doesn't require much space and is faster. Other indexes will be included as columns in the file output.

New in version 0.24.0.

partition_cols [list, optional, default None] Column names by which to partition the dataset. Columns are partitioned in the order they are given.

New in version 0.24.0.

****kwargs** Additional arguments passed to the parquet library. See *pandas io* for more details.

See also:

read_parquet Read a parquet file.

DataFrame.to_csv Write a csv file.

DataFrame.to_sql Write to a sql table.

DataFrame.to_hdf Write to hdf.

Notes

This function requires either the fastparquet or pyarrow library.

Examples

```
>>> df = pd.DataFrame(data={'col1': [1, 2], 'col2': [3, 4]})
>>> df.to_parquet('df.parquet.gzip',
...               compression='gzip')
>>> pd.read_parquet('df.parquet.gzip')
   col1  col2
0     1     3
1     2     4
```

pandas.DataFrame.to_period

DataFrame.**to_period**(*self*, *freq=None*, *axis=0*, *copy=True*) → 'DataFrame'

Convert DataFrame from DatetimeIndex to PeriodIndex.

Convert DataFrame from DatetimeIndex to PeriodIndex with desired frequency (inferred from index if not passed).

Parameters

freq [str, default] Frequency of the PeriodIndex.

axis [{0 or 'index', 1 or 'columns'}, default 0] The axis to convert (the index by default).

copy [bool, default True] If False then underlying input data is not copied.

Returns

TimeSeries with PeriodIndex

pandas.DataFrame.to_pickle

DataFrame.**to_pickle**(*self*, *path*, *compression: Union[str, NoneType] = 'infer'*, *protocol: int = 4*) → None

Pickle (serialize) object to file.

Parameters

path [str] File path where the pickled object will be stored.

compression [{'infer', 'gzip', 'bz2', 'zip', 'xz', None}, default 'infer'] A string representing the compression to use in the output file. By default, infers from the file extension in specified path.

protocol [int] Int which indicates which protocol should be used by the pickler, default HIGHEST_PROTOCOL (see [1] paragraph 12.1.2). The possible values are 0, 1, 2, 3, 4. A negative value for the protocol parameter is equivalent to setting its value to HIGHEST_PROTOCOL.

New in version 0.21.0..

See also:

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

DataFrame.to_hdf Write DataFrame to an HDF5 file.

DataFrame.to_sql Write DataFrame to a SQL database.

DataFrame.to_parquet Write a DataFrame to the binary parquet format.

Examples

```
>>> original_df = pd.DataFrame({"foo": range(5), "bar": range(5, 10)})
>>> original_df
   foo  bar
0    0    5
1    1    6
2    2    7
3    3    8
4    4    9
>>> original_df.to_pickle("./dummy.pkl")
```

```
>>> unpickled_df = pd.read_pickle("./dummy.pkl")
>>> unpickled_df
   foo  bar
0    0    5
1    1    6
2    2    7
3    3    8
4    4    9
```

```
>>> import os
>>> os.remove("./dummy.pkl")
```

pandas.DataFrame.to_records

DataFrame.**to_records**(*self*, *index=True*, *column_dtypes=None*, *index_dtypes=None*) → numpy.recarray

Convert DataFrame to a NumPy record array.

Index will be included as the first field of the record array if requested.

Parameters

index [bool, default True] Include index in resulting record array, stored in 'index' field or using the index label, if set.

column_dtypes [str, type, dict, default None] New in version 0.24.0.

If a string or type, the data type to store all columns. If a dictionary, a mapping of column names and indices (zero-indexed) to specific data types.

index_dtypes [str, type, dict, default None] New in version 0.24.0.

If a string or type, the data type to store all index levels. If a dictionary, a mapping of index level names and indices (zero-indexed) to specific data types.

This mapping is applied only if *index=True*.

Returns

numpy.recarray NumPy ndarray with the DataFrame labels as fields and each row of the DataFrame as entries.

See also:

DataFrame.from_records Convert structured or record ndarray to DataFrame.

numpy.recarray An ndarray that allows field access using attributes, analogous to typed columns in a spreadsheet.

Examples

```
>>> df = pd.DataFrame({'A': [1, 2], 'B': [0.5, 0.75]},
...                   index=['a', 'b'])
>>> df
   A     B
a  1  0.50
b  2  0.75
>>> df.to_records()
rec.array([('a', 1, 0.5 ), ('b', 2, 0.75)],
          dtype=[('index', 'O'), ('A', '<i8'), ('B', '<f8')])
```

If the DataFrame index has no label then the recarray field name is set to 'index'. If the index has a label then this is used as the field name:

```
>>> df.index = df.index.rename("I")
>>> df.to_records()
rec.array([('a', 1, 0.5 ), ('b', 2, 0.75)],
          dtype=[('I', 'O'), ('A', '<i8'), ('B', '<f8')])
```

The index can be excluded from the record array:

```
>>> df.to_records(index=False)
rec.array([(1, 0.5 ), (2, 0.75)],
          dtype=[('A', '<i8'), ('B', '<f8')])
```

Data types can be specified for the columns:

```
>>> df.to_records(column_dtypes={"A": "int32"})
rec.array([('a', 1, 0.5 ), ('b', 2, 0.75)],
          dtype=[('I', 'O'), ('A', '<i4'), ('B', '<f8')])
```

As well as for the index:

```
>>> df.to_records(index_dtypes="<S2")
rec.array([(b'a', 1, 0.5 ), (b'b', 2, 0.75)],
          dtype=[('I', 'S2'), ('A', '<i8'), ('B', '<f8')])
```

```
>>> index_dtypes = f"<S{df.index.str.len().max()}"
>>> df.to_records(index_dtypes=index_dtypes)
rec.array([(b'a', 1, 0.5 ), (b'b', 2, 0.75)],
          dtype=[('I', 'S1'), ('A', '<i8'), ('B', '<f8')])
```

pandas.DataFrame.to_sql

DataFrame.**to_sql**(*self*, *name: str*, *con*, *schema=None*, *if_exists: str = 'fail'*, *index: bool = True*, *index_label=None*, *chunksize=None*, *dtype=None*, *method=None*) → None

Write records stored in a DataFrame to a SQL database.

Databases supported by SQLAlchemy [1] are supported. Tables can be newly created, appended to, or overwritten.

Parameters

name [str] Name of SQL table.

con [sqlalchemy.engine.Engine or sqlite3.Connection] Using SQLAlchemy makes it possible to use any DB supported by that library. Legacy support is provided for sqlite3.Connection objects. The user is responsible for engine disposal and connection closure for the SQLAlchemy connectable See here

schema [str, optional] Specify the schema (if database flavor supports this). If None, use default schema.

if_exists [{'fail', 'replace', 'append'}, default 'fail'] How to behave if the table already exists.

- fail: Raise a ValueError.
- replace: Drop the table before inserting new values.
- append: Insert new values to the existing table.

index [bool, default True] Write DataFrame index as a column. Uses *index_label* as the column name in the table.

index_label [str or sequence, default None] Column label for index column(s). If None is given (default) and *index* is True, then the index names are used. A sequence should be given if the DataFrame uses MultiIndex.

chunksize [int, optional] Specify the number of rows in each batch to be written at a time. By default, all rows will be written at once.

dtype [dict or scalar, optional] Specifying the datatype for columns. If a dictionary is used, the keys should be the column names and the values should be the SQLAlchemy types or strings for the sqlite3 legacy mode. If a scalar is provided, it will be applied to all columns.

method [{None, 'multi', callable}, optional] Controls the SQL insertion clause used:

- None : Uses standard SQL `INSERT` clause (one per row).
- 'multi': Pass multiple values in a single `INSERT` clause.
- callable with signature `(pd_table, conn, keys, data_iter)`.

Details and a sample callable implementation can be found in the section *insert method*.

New in version 0.24.0.

Raises

ValueError When the table already exists and *if_exists* is 'fail' (the default).

See also:

read_sql Read a DataFrame from a table.

Notes

Timezone aware datetime columns will be written as `Timestamp with timezone` type with SQLAlchemy if supported by the database. Otherwise, the datetimes will be stored as timezone unaware timestamps local to the original timezone.

New in version 0.24.0.

References

[1], [2]

Examples

Create an in-memory SQLite database.

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://', echo=False)
```

Create a table from scratch with 3 rows.

```
>>> df = pd.DataFrame({'name' : ['User 1', 'User 2', 'User 3']})
>>> df
     name
0  User 1
1  User 2
2  User 3
```

```
>>> df.to_sql('users', con=engine)
>>> engine.execute("SELECT * FROM users").fetchall()
[(0, 'User 1'), (1, 'User 2'), (2, 'User 3')]
```

```
>>> df1 = pd.DataFrame({'name' : ['User 4', 'User 5']})
>>> df1.to_sql('users', con=engine, if_exists='append')
>>> engine.execute("SELECT * FROM users").fetchall()
[(0, 'User 1'), (1, 'User 2'), (2, 'User 3'),
 (0, 'User 4'), (1, 'User 5')]
```

Overwrite the table with just `df1`.

```
>>> df1.to_sql('users', con=engine, if_exists='replace',
...            index_label='id')
>>> engine.execute("SELECT * FROM users").fetchall()
[(0, 'User 4'), (1, 'User 5')]
```

Specify the dtype (especially useful for integers with missing values). Notice that while pandas is forced to store the data as floating point, the database supports nullable integers. When fetching the data with Python, we get back integer scalars.

```
>>> df = pd.DataFrame({"A": [1, None, 2]})
>>> df
     A
0  1.0
1  NaN
2  2.0
```

```
>>> from sqlalchemy.types import Integer
>>> df.to_sql('integers', con=engine, index=False,
...           dtype={"A": Integer()})
```

```
>>> engine.execute("SELECT * FROM integers").fetchall()
[(1,), (None,), (2,)]
```

pandas.DataFrame.to_stata

DataFrame.**to_stata**(*self*, *path*, *convert_dates=None*, *write_index=True*, *byteorder=None*, *time_stamp=None*, *data_label=None*, *variable_labels=None*, *version=114*, *convert_strl=None*)

Export DataFrame object to Stata dta format.

Writes the DataFrame to a Stata dataset file. "dta" files contain a Stata dataset.

Parameters

path [str, buffer or path object] String, path object (pathlib.Path or py._path.local.LocalPath) or object implementing a binary write() function. If using a buffer then the buffer will not be automatically closed after the file data has been written.

Changed in version 1.0.0.

Previously this was "fname"

convert_dates [dict] Dictionary mapping columns containing datetime types to stata internal format to use when writing the dates. Options are 'tc', 'td', 'tm', 'tw', 'th',

'tq', 'ty'. Column can be either an integer or a name. Datetime columns that do not have a conversion type specified will be converted to 'tc'. Raises NotImplementedError if a datetime column has timezone information.

write_index [bool] Write the index to Stata dataset.

byteorder [str] Can be ">", "<", "little", or "big". default is *sys.byteorder*.

time_stamp [datetime] A datetime to use as file creation date. Default is the current time.

data_label [str, optional] A label for the data set. Must be 80 characters or smaller.

variable_labels [dict] Dictionary containing columns as keys and variable labels as values. Each label must be 80 characters or smaller.

version [{114, 117, 118, 119, None}, default 114] Version to use in the output dta file. Set to None to let pandas decide between 118 or 119 formats depending on the number of columns in the frame. Version 114 can be read by Stata 10 and later. Version 117 can be read by Stata 13 or later. Version 118 is supported in Stata 14 and later. Version 119 is supported in Stata 15 and later. Version 114 limits string variables to 244 characters or fewer while versions 117 and later allow strings with lengths up to 2,000,000 characters. Versions 118 and 119 support Unicode characters, and version 119 supports more than 32,767 variables.

New in version 0.23.0.

Changed in version 1.0.0: Added support for formats 118 and 119.

convert_strl [list, optional] List of column names to convert to string columns to Stata StrL format. Only available if version is 117. Storing strings in the StrL format can produce smaller dta files if strings have more than 8 characters and values are repeated.

New in version 0.23.0.

Raises

NotImplementedError

- If datetimes contain timezone information
- Column dtype is not representable in Stata

ValueError

- Columns listed in convert_dates are neither datetime64[ns] or datetime.datetime
- Column listed in convert_dates is not in DataFrame
- Categorical label contains more than 32,000 characters

See also:

read_stata Import Stata data files.

io.stata.StataWriter Low-level writer for Stata data files.

io.stata.StataWriter117 Low-level writer for version 117 files.

Examples

```
>>> df = pd.DataFrame({'animal': ['falcon', 'parrot', 'falcon',
...                               'parrot'],
...                    'speed': [350, 18, 361, 15]})
>>> df.to_stata('animals.dta')
```

pandas.DataFrame.to_string

DataFrame.**to_string**(*self*, *buf: Union[str, pathlib.Path, IO[str], NoneType] = None*, *columns: Union[Sequence[str], NoneType] = None*, *col_space: Union[int, NoneType] = None*, *header: Union[bool, Sequence[str]] = True*, *index: bool = True*, *na_rep: str = 'NaN'*, *formatters: Union[List[Callable], Tuple[Callable, ...], Mapping[Union[str, int], Callable], NoneType] = None*, *float_format: Union[str, Callable, ForwardRef('EngFormatter'), NoneType] = None*, *sparsify: Union[bool, NoneType] = None*, *index_names: bool = True*, *justify: Union[str, NoneType] = None*, *max_rows: Union[int, NoneType] = None*, *min_rows: Union[int, NoneType] = None*, *max_cols: Union[int, NoneType] = None*, *show_dimensions: bool = False*, *decimal: str = '.'*, *line_width: Union[int, NoneType] = None*, *max_colwidth: Union[int, NoneType] = None*, *encoding: Union[str, NoneType] = None*) → Union[str, NoneType]

Render a DataFrame to a console-friendly tabular output.

Parameters

buf [str, Path or StringIO-like, optional, default None] Buffer to write to. If None, the output is returned as a string.

columns [sequence, optional, default None] The subset of columns to write. Writes all columns by default.

col_space [int, optional] The minimum width of each column.

header [bool or sequence, optional] Write out the column names. If a list of strings is given, it is assumed to be aliases for the column names.

index [bool, optional, default True] Whether to print index (row) labels.

na_rep [str, optional, default 'NaN'] String representation of NAN to use.

formatters [list, tuple or dict of one-param. functions, optional] Formatter functions to apply to columns' elements by position or name. The result of each function must be a unicode string. List/tuple must be of length equal to the number of columns.

float_format [one-parameter function, optional, default None] Formatter function to apply to columns' elements if they are floats. The result of this function must be a unicode string.

sparsify [bool, optional, default True] Set to False for a DataFrame with a hierarchical index to print every multiindex key at each row.

index_names [bool, optional, default True] Prints the names of the indexes.

justify [str, default None] How to justify the column labels. If None uses the option from the print configuration (controlled by set_option), 'right' out of the box. Valid values are

- left

- right
- center
- justify
- justify-all
- start
- end
- inherit
- match-parent
- initial
- unset.

max_rows [int, optional] Maximum number of rows to display in the console.

min_rows [int, optional] The number of rows to display in the console in a truncated repr (when number of rows is above *max_rows*).

max_cols [int, optional] Maximum number of columns to display in the console.

show_dimensions [bool, default False] Display DataFrame dimensions (number of rows by number of columns).

decimal [str, default '.'] Character recognized as decimal separator, e.g. ',' in Europe.

line_width [int, optional] Width to wrap a line in characters.

max_colwidth [int, optional] Max width to truncate each column in characters. By default, no limit.

New in version 1.0.0.

encoding [str, default "utf-8"] Set character encoding.

New in version 1.0.

Returns

str or None If buf is None, returns the result as a string. Otherwise returns None.

See also:

to_html Convert DataFrame to HTML.

Examples

```
>>> d = {'col1': [1, 2, 3], 'col2': [4, 5, 6]}
>>> df = pd.DataFrame(d)
>>> print(df.to_string())
   col1  col2
0     1     4
1     2     5
2     3     6
```