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_exc
pandas.ExcelWriter
pandas.read_json
pandas.json_normalize
pandas.DataFrame.to_json
pandas.io.json.build_table_schema
pandas.read_html
pandas.DataFrame.to_html
pandas.io.formats.style.Styler.to_htn
pandas.read_xml
pandas.DataFrame.to_xml
pandas.DataFrame.to_latex
pandas.io.formats.style.Styler.to_late
pandas.read_hdf

pandas.read_csv

pandas.read_csv(*filepath_or_buffer, sep=NoDefault.no_default, delimiter=None, header='infer', names=NoDefault.no_default, index_col=None, usecols=None, squeeze=None, prefix=NoDefault.no_default, 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=None, 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='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None, error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None, storage_options=None*) [source]

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

Also supports optionally iterating or breaking of the file into chunks.

Additional help can be found in the online docs for IO Tools.

Parameters: **filepath_or_buffer : *str, path object or file-like object***

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, gs, and file. For file URLs, a host is expected. A local file could be: file://localhost/path/to/table.csv.

If you want to pass in a path object, pandas accepts any `os.PathLike`.

By file-like object, we refer to objects with a `read()` method, such as a file handle (e.g. via builtin `open` function) or `StringIO`.

sep : *str, default ','*

Delimiter to use. If sep is None, the C engine cannot automatically detect the separator, but the Python parsing engine can, meaning the latter will be used and automatically detect the separator by Python's builtin sniffer tool, `csv.Sniffer`. In addition, separators longer than 1 character and different from `'\s+'` will be interpreted as regular expressions and will also force the use of the Python parsing engine. Note that regex delimiters are prone to ignoring quoted data. Regex example: `'\r\t'`.

delimiter : *str, default None*

Alias for sep.

header : *int, list of int, None, default 'infer'*

Row number(s) to use as the column names, and the start of the data. Default behavior is to infer the column names: if no names are passed the behavior is identical to `header=0` and column names are inferred from the first line of the file, if column names are passed explicitly then the behavior is identical to `header=None`. Explicitly pass `header=0` to be able to replace existing names. The header can be a list of integers that specify row locations for a multi-index on the columns e.g. [0,1,3]. Intervening rows that are not specified will be skipped (e.g. 2 in this example is skipped). Note that this parameter ignores commented lines and empty lines if `skip_blank_lines=True`, so `header=0` denotes the first line of data rather than the first line of the file.

names : *array-like, optional*

List of column names to use. If the file contains a header row, then you should explicitly pass `header=0` to override the column names. Duplicates in this list are not allowed.

index_col : *int, str, sequence of int / str, or False, optional, default None*

Column(s) to use as the row labels of the `DataFrame`, either given as string name or column index. If a sequence of int / str is given, a MultiIndex is used.

Note: `index_col=False` can be used to force pandas to *not* use the first column as the index, e.g. when you have a malformed file with delimiters at the end of each line.

usecols : *list-like or callable, optional*

Return a subset of the columns. If list-like, all elements must either be positional (i.e. integer indices into the document columns) or strings that correspond to column names provided either by the user in *names* or inferred from the document header row(s). If `names` are given, the document header row(s) are not taken into account. For example, a valid list-like *usecols* parameter would be `[0, 1, 2]` or `['foo', 'bar', 'baz']`. Element order is ignored, so `usecols=[0, 1]` is the same as `[1, 0]`. To instantiate a DataFrame from `data` with element order preserved use `pd.read_csv(data, usecols=['foo', 'bar'])[['foo', 'bar']]` for columns in `['foo', 'bar']` order or `pd.read_csv(data, usecols=['foo', 'bar'])[['bar', 'foo']]` for `['bar', 'foo']` order.

If callable, the callable function will be evaluated against the column names, returning names where the callable function evaluates to True. An example of a valid callable argument would be `lambda x: x.upper() in ['AAA', 'BBB', 'DDD']`. Using this parameter results in much faster parsing time and lower memory usage.

squeeze : *bool, default False*

If the parsed data only contains one column then return a Series.

> ***Deprecated since version 1.4.0:*** Append `.squeeze("columns")` to the call to `read_csv` to squeeze the data.

prefix : *str, optional*

Prefix to add to column numbers when no header, e.g. 'X' for X0, X1, ...

> ***Deprecated since version 1.4.0:*** Use a list comprehension on the DataFrame's columns after calling `read_csv`.

mangle_dupe_cols : *bool, default True*

Duplicate columns will be specified as 'X', 'X.1', ...'X.N', rather than 'X'...'X'. Passing in False will cause data to be overwritten if there are duplicate names in the columns.

dtype : *Type name or dict of column -> type, optional*

Data type for data or columns. E.g. {'a': np.float64, 'b': np.int32, 'c': 'Int64'} Use *str* or *object* together with suitable *na_values* settings to preserve and not interpret dtype. If converters are specified, they will be applied INSTEAD of dtype conversion.

engine : *{'c', 'python', 'pyarrow'}, optional*

Parser engine to use. The C and pyarrow engines are faster, while the python engine is currently more feature-complete. Multithreading is currently only supported by the pyarrow engine.

> ***New in version 1.4.0:*** The "pyarrow" engine was added as an *experimental* engine, and some features are unsupported, or may not work correctly, with this engine.

converters : *dict, optional*

Dict of functions for converting values in certain columns. Keys can either be integers or column labels.

true_values : *list, optional*

Values to consider as True.

false_values : *list, optional*

Values to consider as False.

skipinitialspace : *bool, default False*

Skip spaces after delimiter.

skiprows : *list-like, int or callable, optional*

Line numbers to skip (0-indexed) or number of lines to skip (int) at the start of

Whether or not to include the default NaN values when parsing the data.
Depending on whether *na_values* is passed in, the behavior is as follows:

- If *keep_default_na* is True, and *na_values* are specified, *na_values* is appended to the default NaN values used for parsing.
- If *keep_default_na* is True, and *na_values* are not specified, only the default NaN values are used for parsing.
- If *keep_default_na* is False, and *na_values* are specified, only the NaN values specified *na_values* are used for parsing.
- If *keep_default_na* is False, and *na_values* are not specified, no strings will be parsed as NaN.

Note that if *na_filter* is passed in as False, the *keep_default_na* and *na_values* parameters will be ignored.

na_filter : *bool, default True*

Detect missing value markers (empty strings and the value of na_values). In data without any NAs, passing na_filter=False can improve the performance of reading a large file.

verbose : *bool, default False*

Indicate number of NA values placed in non-numeric columns.

skip_blank_lines : *bool, default True*

If True, skip over blank lines rather than interpreting as NaN values.

parse_dates : *bool or list of int or names or list of lists or dict, default False*

The behavior is as follows:

- boolean. If True -> try parsing the index.
- list of int or names. e.g. If [1, 2, 3] -> try parsing columns 1, 2, 3 each as a separate date column.

If True, use a cache of unique, converted dates to apply the datetime conversion. May produce significant speed-up when parsing duplicate date strings, especially ones with timezone offsets.

> ***New in version 0.25.0.***

iterator : *bool, default False*

Return TextFileReader object for iteration or getting chunks with `get_chunk()`.

> ***Changed in version 1.2:*** `TextFileReader` is a context manager.

Returns: **DataFrame or TextParser**

A comma-separated values (csv) file is returned as two-dimensional data structure with labeled axes.

See also

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

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

`read_fwf`
: Read a table of fixed-width formatted lines into DataFrame.

Examples

```
>>> pd.read_csv('data.csv')
```

Previous
pandas.read_table

Next
pandas.DataFrame.to_csv