# pandas.DataFrame

## *class* pandas.DataFrame(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=None*)

Two-dimensional, size-mutable, potentially heterogeneous tabular data. [source]

Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

**Parameters:**

**data : *ndarray (structured or homogeneous), Iterable, dict, or DataFrame***

Dict can contain Series, arrays, constants, dataclass or list-like objects. If data is a dict, column order follows insertion-order. If a dict contains Series which have an index defined, it is aligned by its index. This alignment also occurs if data is a Series or a DataFrame itself. Alignment is done on Series/DataFrame inputs.
If data is a list of dicts, column order follows insertion-order.

**index : *Index or array-like***

Index to use for resulting frame. Will default to RangeIndex if no indexing information part of input data and no index provided.

**columns : *Index or array-like***

Column labels to use for resulting frame when data does not have them, defaulting to RangeIndex(0, 1, 2, …, n). If data contains column labels, will perform column selection instead.

**dtype : *dtype, default None***

Data type to force. Only a single dtype is allowed. If None, infer.

**copy : *bool or None, default None***

Copy data from inputs. For dict data, the default of None behaves like `copy=True`. For DataFrame or 2d ndarray input, the default of None behaves like `copy=False`. If data is a dict containing one or more Series (possibly of different dtypes), `copy=False` will ensure that these inputs are not copied.

> *Changed in version 1.3.0.*

**See also**

`DataFrame.from_records`
Constructor from tuples, also record arrays.

`DataFrame.from_dict`
From dicts of Series, arrays, or dicts.

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

`read_table`
Read general delimited file into DataFrame.

`read_clipboard`
Read text from clipboard into DataFrame.

Read text from clipboard into DataFrame.

### Notes

Please reference the User Guide for more information.

### Examples

Constructing DataFrame from a dictionary.

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

Notice that the inferred dtype is int64.

```
>>> df.dtypes
col1    int64
col2    int64
dtype: object
```

To enforce a single dtype:

```
>>> df = pd.DataFrame(data=d, dtype=np.int8)
>>> df.dtypes
col1    int8
col2    int8
dtype: object
```

Constructing DataFrame from a dictionary including Series:

```
>>> d = {'col1': [0, 1, 2, 3], 'col2': pd.Series([2, 3], index=[2, 3])}
>>> pd.DataFrame(data=d, index=[0, 1, 2, 3])
   col1  col2
0     0   NaN
1     1   NaN
2     2   2.0
3     3   3.0
```

Constructing DataFrame from numpy ndarray:

```
>>> df2 = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
...                    columns=['a', 'b', 'c'])
>>> df2
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

Constructing DataFrame from a numpy ndarray that has labeled columns:

```
>>> data = np.array([(1, 2, 3), (4, 5, 6), (7, 8, 9)],
...                 dtype=[("a", "i4"), ("b", "i4"), ("c", "i4")])
>>> df3 = pd.DataFrame(data, columns=['c', 'a'])
...
>>> df3
   c  a
0  3  1
1  6  4
2  9  7
```

Constructing DataFrame from dataclass:

```
>>> from dataclasses import make_dataclass
>>> Point = make_dataclass("Point", [("x", int), ("y", int)])
>>> pd.DataFrame([Point(0, 0), Point(0, 3), Point(2, 3)])
   x  y
0  0  0
1  0  3
2  2  3
```

Constructing DataFrame from Series/DataFrame:

```
>>> ser = pd.Series([1, 2, 3], index=["a", "b", "c"])
>>> df = pd.DataFrame(data=ser, index=["a", "c"])
>>> df
   0
a  1
c  3
```

```
>>> df1 = pd.DataFrame([1, 2, 3], index=["a", "b", "c"], columns=["x"])
>>> df2 = pd.DataFrame(data=df1, index=["a", "c"])
>>> df2
   x
a  1
c  3
```

**Attributes**

| | |
|---|---|
| **T** | The transpose of the DataFrame. |
| **at** | Access a single value for a row/column label pair. |
| **attrs** | Dictionary of global attributes of this dataset. |
| **axes** | Return a list representing the axes of the DataFrame. |
| **columns** | The column labels of the DataFrame. |
| **dtypes** | Return the dtypes in the DataFrame. |
| **empty** | Indicator whether Series/DataFrame is empty. |
| **flags** | Get the properties associated with this pandas object. |
| **iat** | Access a single value for a row/column pair by integer position. |
| **iloc** | Purely integer-location based indexing for selection by position. |
| **index** | The index (row labels) of the DataFrame. |
| **loc** | Access a group of rows and columns by label(s) or a boolean array. |
| **ndim** | Return an int representing the number of axes / array dimensions. |
| **shape** | Return a tuple representing the dimensionality of the DataFrame. |
| **size** | Return an int representing the number of elements in this object. |

| | |
|---|---|
| `style` | Returns a Styler object. |
| `values` | Return a Numpy representation of the DataFrame. |

**Methods**

| | |
|---|---|
| `abs` () | Return a Series/DataFrame with absolute numeric value of each element. |
| `add` (other[, axis, level, fill_value]) | Get Addition of dataframe and other, element-wise (binary operator *add*). |
| `add_prefix` (prefix[, axis]) | Prefix labels with string *prefix*. |
| `add_suffix` (suffix[, axis]) | Suffix labels with string *suffix*. |
| `agg` ([func, axis]) | Aggregate using one or more operations over the specified axis. |
| `aggregate` ([func, axis]) | Aggregate using one or more operations over the specified axis. |
| `align` (other[, join, axis, level, copy, ...]) | Align two objects on their axes with the specified join method. |
| `all` ([axis, bool_only, skipna]) | Return whether all elements are True, potentially over an axis. |
| `any` (*[, axis, bool_only, skipna]) | Return whether any element is True, potentially over an axis. |
| `apply` (func[, axis, raw, result_type, args, ...]) | Apply a function along an axis of the DataFrame. |
| `applymap` (func[, na_action]) | (DEPRECATED) Apply a function to a Dataframe elementwise. |
| `asfreq` (freq[, method, how, normalize, ...]) | Convert time series to specified frequency. |
| `asof` (where[, subset]) | Return the last row(s) without any NaNs before *where*. |
| `assign` (**kwargs) | Assign new columns to a DataFrame. |
| `astype` (dtype[, copy, errors]) | Cast a pandas object to a specified dtype `dtype` . |
| `at_time` (time[, asof, axis]) | Select values at particular time of day (e.g., 9:30AM). |
| `backfill` (*[, axis, inplace, limit, downcast]) | (DEPRECATED) Fill NA/NaN values by using the next valid observation to fill the gap. |
| `between_time` (start_time, end_time[, ...]) | Select values between particular times of the day (e.g., 9:00-9:30 AM). |
| `bfill` (*[, axis, inplace, limit, downcast]) | Fill NA/NaN values by using the next valid observation to fill the gap. |
| `bool` () | (DEPRECATED) Return the bool of a single element Series or DataFrame. |
| `boxplot` ([column, by, ax, fontsize, rot, ...]) | Make a box plot from DataFrame columns. |
| `clip` ([lower, upper, axis, inplace]) | Trim values at input threshold(s). |
| `combine` (other, func[, fill_value, overwrite]) | Perform column-wise combine with another DataFrame. |

| | |
|---|---|
| `combine_first` (other) | Update null elements with value in the same location in *other*. |
| `compare` (other[, align_axis, keep_shape, ...]) | Compare to another DataFrame and show the differences. |
| `convert_dtypes` ([infer_objects, ...]) | Convert columns to the best possible dtypes using dtypes supporting `pd.NA`. |
| `copy` ([deep]) | Make a copy of this object's indices and data. |
| `corr` ([method, min_periods, numeric_only]) | Compute pairwise correlation of columns, excluding NA/null values. |
| `corrwith` (other[, axis, drop, method, ...]) | Compute pairwise correlation. |
| `count` ([axis, numeric_only]) | Count non-NA cells for each column or row. |
| `cov` ([min_periods, ddof, numeric_only]) | Compute pairwise covariance of columns, excluding NA/null values. |
| `cummax` ([axis, skipna]) | Return cumulative maximum over a DataFrame or Series axis. |
| `cummin` ([axis, skipna]) | Return cumulative minimum over a DataFrame or Series axis. |
| `cumprod` ([axis, skipna]) | Return cumulative product over a DataFrame or Series axis. |
| `cumsum` ([axis, skipna]) | Return cumulative sum over a DataFrame or Series axis. |
| `describe` ([percentiles, include, exclude]) | Generate descriptive statistics. |
| `diff` ([periods, axis]) | First discrete difference of element. |
| `div` (other[, axis, level, fill_value]) | Get Floating division of dataframe and other, element-wise (binary operator *truediv*). |
| `divide` (other[, axis, level, fill_value]) | Get Floating division of dataframe and other, element-wise (binary operator *truediv*). |
| `dot` (other) | Compute the matrix multiplication between the DataFrame and other. |
| `drop` ([labels, axis, index, columns, level, ...]) | Drop specified labels from rows or columns. |
| `drop_duplicates` ([subset, keep, inplace, ...]) | Return DataFrame with duplicate rows removed. |
| `droplevel` (level[, axis]) | Return Series/DataFrame with requested index / column level(s) removed. |
| `dropna` (*[, axis, how, thresh, subset, ...]) | Remove missing values. |
| `duplicated` ([subset, keep]) | Return boolean Series denoting duplicate rows. |
| `eq` (other[, axis, level]) | Get Equal to of dataframe and other, element-wise (binary operator *eq*). |
| `equals` (other) | Test whether two objects contain the same elements. |
| `eval` (expr, *[, inplace]) | Evaluate a string describing operations on DataFrame columns. |
| `ewm` ([com, span, halflife, alpha, ...]) | Provide exponentially weighted (EW) calculations. |

| | |
|---|---|
| `expanding` ([min_periods, axis, method]) | Provide expanding window calculations. |
| `explode` (column[, ignore_index]) | Transform each element of a list-like to a row, replicating index values. |
| `ffill` (*[, axis, inplace, limit, downcast]) | Fill NA/NaN values by propagating the last valid observation to next valid. |
| `fillna` ([value, method, axis, inplace, ...]) | Fill NA/NaN values using the specified method. |
| `filter` ([items, like, regex, axis]) | Subset the dataframe rows or columns according to the specified index labels. |
| `first` (offset) | Select initial periods of time series data based on a date offset. |
| `first_valid_index` () | Return index for first non-NA value or None, if no non-NA value is found. |
| `floordiv` (other[, axis, level, fill_value]) | Get Integer division of dataframe and other, element-wise (binary operator *floordiv*). |
| `from_dict` (data[, orient, dtype, columns]) | Construct DataFrame from dict of array-like or dicts. |
| `from_records` (data[, index, exclude, ...]) | Convert structured or record ndarray to DataFrame. |
| `ge` (other[, axis, level]) | Get Greater than or equal to of dataframe and other, element-wise (binary operator *ge*). |
| `get` (key[, default]) | Get item from object for given key (ex: DataFrame column). |
| `groupby` ([by, axis, level, as_index, sort, ...]) | Group DataFrame using a mapper or by a Series of columns. |
| `gt` (other[, axis, level]) | Get Greater than of dataframe and other, element-wise (binary operator *gt*). |
| `head` ([n]) | Return the first *n* rows. |
| `hist` ([column, by, grid, xlabelsize, xrot, ...]) | Make a histogram of the DataFrame's columns. |
| `idxmax` ([axis, skipna, numeric_only]) | Return index of first occurrence of maximum over requested axis. |
| `idxmin` ([axis, skipna, numeric_only]) | Return index of first occurrence of minimum over requested axis. |
| `infer_objects` ([copy]) | Attempt to infer better dtypes for object columns. |
| `info` ([verbose, buf, max_cols, memory_usage, ...]) | Print a concise summary of a DataFrame. |
| `insert` (loc, column, value[, allow_duplicates]) | Insert column into DataFrame at specified location. |
| `interpolate` ([method, axis, limit, inplace, ...]) | Fill NaN values using an interpolation method. |
| `isetitem` (loc, value) | Set the given value in the column with position *loc*. |
| `isin` (values) | Whether each element in the DataFrame is contained in values. |
| `isna` () | Detect missing values. |

| | |
|---|---|
| `isnull` () | DataFrame.isnull is an alias for DataFrame.isna. |
| `items` () | Iterate over (column name, Series) pairs. |
| `iterrows` () | Iterate over DataFrame rows as (index, Series) pairs. |
| `itertuples` ([index, name]) | Iterate over DataFrame rows as namedtuples. |
| `join` (other[, on, how, lsuffix, rsuffix, ...]) | Join columns of another DataFrame. |
| `keys` () | Get the 'info axis' (see Indexing for more). |
| `kurt` ([axis, skipna, numeric_only]) | Return unbiased kurtosis over requested axis. |
| `kurtosis` ([axis, skipna, numeric_only]) | Return unbiased kurtosis over requested axis. |
| `last` (offset) | Select final periods of time series data based on a date offset. |
| `last_valid_index` () | Return index for last non-NA value or None, if no non-NA value is found. |
| `le` (other[, axis, level]) | Get Less than or equal to of dataframe and other, element-wise (binary operator *le*). |
| `lt` (other[, axis, level]) | Get Less than of dataframe and other, element-wise (binary operator *lt*). |
| `map` (func[, na_action]) | Apply a function to a Dataframe elementwise. |
| `mask` (cond[, other, inplace, axis, level]) | Replace values where the condition is True. |
| `max` ([axis, skipna, numeric_only]) | Return the maximum of the values over the requested axis. |
| `mean` ([axis, skipna, numeric_only]) | Return the mean of the values over the requested axis. |
| `median` ([axis, skipna, numeric_only]) | Return the median of the values over the requested axis. |
| `melt` ([id_vars, value_vars, var_name, ...]) | Unpivot a DataFrame from wide to long format, optionally leaving identifiers set. |
| `memory_usage` ([index, deep]) | Return the memory usage of each column in bytes. |
| `merge` (right[, how, on, left_on, right_on, ...]) | Merge DataFrame or named Series objects with a database-style join. |
| `min` ([axis, skipna, numeric_only]) | Return the minimum of the values over the requested axis. |
| `mod` (other[, axis, level, fill_value]) | Get Modulo of dataframe and other, element-wise (binary operator *mod*). |
| `mode` ([axis, numeric_only, dropna]) | Get the mode(s) of each element along the selected axis. |
| `mul` (other[, axis, level, fill_value]) | Get Multiplication of dataframe and other, element-wise (binary operator *mul*). |
| `multiply` (other[, axis, level, fill_value]) | Get Multiplication of dataframe and other, element-wise (binary operator *mul*). |

| | |
|---|---|
| `ne` (other[, axis, level]) | Get Not equal to of dataframe and other, element-wise (binary operator *ne*). |
| `nlargest` (n, columns[, keep]) | Return the first *n* rows ordered by *columns* in descending order. |
| `notna` () | Detect existing (non-missing) values. |
| `notnull` () | DataFrame.notnull is an alias for DataFrame.notna. |
| `nsmallest` (n, columns[, keep]) | Return the first *n* rows ordered by *columns* in ascending order. |
| `nunique` ([axis, dropna]) | Count number of distinct elements in specified axis. |
| `pad` (*[, axis, inplace, limit, downcast]) | (DEPRECATED) Fill NA/NaN values by propagating the last valid observation to next valid. |
| `pct_change` ([periods, fill_method, limit, freq]) | Fractional change between the current and a prior element. |
| `pipe` (func, *args, **kwargs) | Apply chainable functions that expect Series or DataFrames. |
| `pivot` (*, columns[, index, values]) | Return reshaped DataFrame organized by given index / column values. |
| `pivot_table` ([values, index, columns, ...]) | Create a spreadsheet-style pivot table as a DataFrame. |
| `plot` | alias of `PlotAccessor` |
| `pop` (item) | Return item and drop from frame. |
| `pow` (other[, axis, level, fill_value]) | Get Exponential power of dataframe and other, element-wise (binary operator *pow*). |
| `prod` ([axis, skipna, numeric_only, min_count]) | Return the product of the values over the requested axis. |
| `product` ([axis, skipna, numeric_only, min_count]) | Return the product of the values over the requested axis. |
| `quantile` ([q, axis, numeric_only, ...]) | Return values at the given quantile over requested axis. |
| `query` (expr, *[, inplace]) | Query the columns of a DataFrame with a boolean expression. |
| `radd` (other[, axis, level, fill_value]) | Get Addition of dataframe and other, element-wise (binary operator *radd*). |
| `rank` ([axis, method, numeric_only, ...]) | Compute numerical data ranks (1 through n) along axis. |
| `rdiv` (other[, axis, level, fill_value]) | Get Floating division of dataframe and other, element-wise (binary operator *rtruediv*). |
| `reindex` ([labels, index, columns, axis, ...]) | Conform DataFrame to new index with optional filling logic. |
| `reindex_like` (other[, method, copy, limit, ...]) | Return an object with matching indices as other object. |
| `rename` ([mapper, index, columns, axis, copy, ...]) | Rename columns or index labels. |
| `rename_axis` ([mapper, index, columns, axis, ...]) | Set the name of the axis for the index or columns. |

| | |
|---|---|
| `reorder_levels` (order[, axis]) | Rearrange index levels using input order. |
| `replace` ([to_replace, value, inplace, limit, ...]) | Replace values given in *to_replace* with *value*. |
| `resample` (rule[, axis, closed, label, ...]) | Resample time-series data. |
| `reset_index` ([level, drop, inplace, ...]) | Reset the index, or a level of it. |
| `rfloordiv` (other[, axis, level, fill_value]) | Get Integer division of dataframe and other, element-wise (binary operator *rfloordiv*). |
| `rmod` (other[, axis, level, fill_value]) | Get Modulo of dataframe and other, element-wise (binary operator *rmod*). |
| `rmul` (other[, axis, level, fill_value]) | Get Multiplication of dataframe and other, element-wise (binary operator *rmul*). |
| `rolling` (window[, min_periods, center, ...]) | Provide rolling window calculations. |
| `round` ([decimals]) | Round a DataFrame to a variable number of decimal places. |
| `rpow` (other[, axis, level, fill_value]) | Get Exponential power of dataframe and other, element-wise (binary operator *rpow*). |
| `rsub` (other[, axis, level, fill_value]) | Get Subtraction of dataframe and other, element-wise (binary operator *rsub*). |
| `rtruediv` (other[, axis, level, fill_value]) | Get Floating division of dataframe and other, element-wise (binary operator *rtruediv*). |
| `sample` ([n, frac, replace, weights, ...]) | Return a random sample of items from an axis of object. |
| `select_dtypes` ([include, exclude]) | Return a subset of the DataFrame's columns based on the column dtypes. |
| `sem` ([axis, skipna, ddof, numeric_only]) | Return unbiased standard error of the mean over requested axis. |
| `set_axis` (labels, *[, axis, copy]) | Assign desired index to given axis. |
| `set_flags` (*[, copy, allows_duplicate_labels]) | Return a new object with updated flags. |
| `set_index` (keys, *[, drop, append, inplace, ...]) | Set the DataFrame index using existing columns. |
| `shift` ([periods, freq, axis, fill_value, suffix]) | Shift index by desired number of periods with an optional time *freq*. |
| `skew` ([axis, skipna, numeric_only]) | Return unbiased skew over requested axis. |
| `sort_index` (*[, axis, level, ascending, ...]) | Sort object by labels (along an axis). |
| `sort_values` (by, *[, axis, ascending, ...]) | Sort by the values along either axis. |
| `sparse` | alias of `SparseFrameAccessor` |
| `squeeze` ([axis]) | Squeeze 1 dimensional axis objects into scalars. |
| `stack` ([level, dropna, sort, future_stack]) | Stack the prescribed level(s) from columns to index. |

| | |
|---|---|
| `std` ([axis, skipna, ddof, numeric_only]) | Return sample standard deviation over requested axis. |
| `sub` (other[, axis, level, fill_value]) | Get Subtraction of dataframe and other, element-wise (binary operator *sub*). |
| `subtract` (other[, axis, level, fill_value]) | Get Subtraction of dataframe and other, element-wise (binary operator *sub*). |
| `sum` ([axis, skipna, numeric_only, min_count]) | Return the sum of the values over the requested axis. |
| `swapaxes` (axis1, axis2[, copy]) | (DEPRECATED) Interchange axes and swap values axes appropriately. |
| `swaplevel` ([i, j, axis]) | Swap levels i and j in a `MultiIndex` . |
| `tail` ([n]) | Return the last *n* rows. |
| `take` (indices[, axis]) | Return the elements in the given *positional* indices along an axis. |
| `to_clipboard` ([excel, sep]) | Copy object to the system clipboard. |
| `to_csv` ([path_or_buf, sep, na_rep, ...]) | Write object to a comma-separated values (csv) file. |
| `to_dict` ([orient, into, index]) | Convert the DataFrame to a dictionary. |
| `to_excel` (excel_writer[, sheet_name, na_rep, ...]) | Write object to an Excel sheet. |
| `to_feather` (path, **kwargs) | Write a DataFrame to the binary Feather format. |
| `to_gbq` (destination_table[, project_id, ...]) | Write a DataFrame to a Google BigQuery table. |
| `to_hdf` (path_or_buf, key[, mode, complevel, ...]) | Write the contained data to an HDF5 file using HDFStore. |
| `to_html` ([buf, columns, col_space, header, ...]) | Render a DataFrame as an HTML table. |
| `to_json` ([path_or_buf, orient, date_format, ...]) | Convert the object to a JSON string. |
| `to_latex` ([buf, columns, header, index, ...]) | Render object to a LaTeX tabular, longtable, or nested table. |
| `to_markdown` ([buf, mode, index, storage_options]) | Print DataFrame in Markdown-friendly format. |
| `to_numpy` ([dtype, copy, na_value]) | Convert the DataFrame to a NumPy array. |
| `to_orc` ([path, engine, index, engine_kwargs]) | Write a DataFrame to the ORC format. |
| `to_parquet` ([path, engine, compression, ...]) | Write a DataFrame to the binary parquet format. |
| `to_period` ([freq, axis, copy]) | Convert DataFrame from DatetimeIndex to PeriodIndex. |
| `to_pickle` (path[, compression, protocol, ...]) | Pickle (serialize) object to file. |
| `to_records` ([index, column_dtypes, index_dtypes]) | Convert DataFrame to a NumPy record array. |
| `to_sql` (name, con, *[, schema, if_exists, ...]) | Write records stored in a DataFrame to a SQL database. |
| `to_stata` (path, *[, convert_dates, ...]) | Export DataFrame object to Stata dta format. |

| | |
|---|---|
| `to_string` ([buf, columns, col_space, header, ...]) | Render a DataFrame to a console-friendly tabular output. |
| `to_timestamp` ([freq, how, axis, copy]) | Cast to DatetimeIndex of timestamps, at *beginning* of period. |
| `to_xarray` () | Return an xarray object from the pandas object. |
| `to_xml` ([path_or_buffer, index, root_name, ...]) | Render a DataFrame to an XML document. |
| `transform` (func[, axis]) | Call `func` on self producing a DataFrame with the same axis shape as self. |
| `transpose` (*args[, copy]) | Transpose index and columns. |
| `truediv` (other[, axis, level, fill_value]) | Get Floating division of dataframe and other, element-wise (binary operator *truediv*). |
| `truncate` ([before, after, axis, copy]) | Truncate a Series or DataFrame before and after some index value. |
| `tz_convert` (tz[, axis, level, copy]) | Convert tz-aware axis to target time zone. |
| `tz_localize` (tz[, axis, level, copy, ...]) | Localize tz-naive index of a Series or DataFrame to target time zone. |
| `unstack` ([level, fill_value, sort]) | Pivot a level of the (necessarily hierarchical) index labels. |
| `update` (other[, join, overwrite, ...]) | Modify in place using non-NA values from another DataFrame. |
| `value_counts` ([subset, normalize, sort, ...]) | Return a Series containing the frequency of each distinct row in the Dataframe. |
| `var` ([axis, skipna, ddof, numeric_only]) | Return unbiased variance over requested axis. |
| `where` (cond[, other, inplace, axis, level]) | Replace values where the condition is False. |
| `xs` (key[, axis, level, drop_level]) | Return cross-section from the Series/DataFrame. |

Previous
DataFrame

Next
pandas.DataFrame.T

# 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, 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=_NoDefault.no_default, keep_date_col=False, date_parser=_NoDefault.no_default, date_format=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, on_bad_lines='error', delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None, storage_options=None, dtype_backend=_NoDefault.no_default*)** [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 ‘,’***

Character or regex pattern to treat as the delimiter. If `sep=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 from only the first valid row of the file 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, optional***

Alias for `sep`.

**header : *int, Sequence of int, ‘infer’ or None, default ‘infer’***

Row number(s) containing column labels and marking the start of the data (zero-indexed). 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 to `names` 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 MultiIndex on the columns e.g. `[0, 1, 3]`. Intervening rows that are not specified will be skipped (e.g.

locations for a `MultiIndex` 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 : *Sequence of Hashable, optional***

Sequence of column labels to apply. 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 : *Hashable, Sequence of Hashable or False, optional***

Column(s) to use as row label(s), denoted either by column labels or column indices. If a sequence of labels or indices is given, `MultiIndex` will be formed for the row labels.

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 of Hashable or Callable, optional***

Subset of columns to select, denoted either by column labels or column indices. 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.

**dtype : *dtype or dict of {Hashable : dtype}, optional***

Data type(s) to apply to either the whole dataset or individual 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.

> *New in version 1.5.0:* Support for `defaultdict` was added. Specify a `defaultdict` as input where the default determines the `dtype` of the columns which are not explicitly listed.

**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 of {Hashable : Callable}, optional***

Functions for converting values in specified columns. Keys can either be column labels or column indices.

**true_values : *list, optional***

Values to consider as `True` in addition to case-insensitive variants of 'True'.

**false_values : *list, optional***

Values to consider as `False` in addition to case-insensitive variants of 'False'.

**skipinitialspace : *bool, default False***

Skip spaces after delimiter.

**skiprows : *int, list of int or Callable, optional***

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

If callable, the callable function will be evaluated against the row indices, returning `True` if the row should be skipped and `False` otherwise. An example of a valid callable argument would be `lambda x: x in [0, 2]` .

**skipfooter : *int, default 0***

Number of lines at bottom of file to skip (Unsupported with `engine='c'` ).

**nrows : *int, optional***

Number of rows of file to read. Useful for reading pieces of large files.

**na_values : *Hashable, Iterable of Hashable or dict of {Hashable : Iterable}, optional***

Additional strings to recognize as `NA` / `NaN` . If `dict` passed, specific per-column `NA` values. By default the following values are interpreted as `NaN` : " ", "#N/A", "#N/A N/A", "#NA", "-1.#IND", "-1.#QNAN", "-NaN", "-nan", "1.#IND", "1.#QNAN", "<NA>", "N/A", "NA", "NULL", "NaN", "None", "n/a", "nan", "null ".

**keep_default_na : *bool, default True***

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 `NA` values, 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, list of Hashable, list of lists or dict of {Hashable : list}, default False***

The behavior is as follows:

- `bool` . 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.
- `list` of `list` . e.g. If `[[1, 3]]` -> combine columns 1 and 3 and parse as a single date column.
- `dict` , e.g. `{'foo' : [1, 3]}` -> parse columns 1, 3 as date and call result 'foo'

If a column or index cannot be represented as an array of `datetime` , say because of an unparsable value or a mixture of timezones, the column or index will be returned unaltered as an `object` data type. For non-standard `datetime` parsing, use `to_datetime()` after `read_csv()` .

Note: A fast path exists for iso8601-formatted dates.

Note: A fast-path exists for iso8601-formatted dates.

**infer_datetime_format : *bool, default False***

If `True` and `parse_dates` is enabled, pandas will attempt to infer the format of the `datetime` strings in the columns, and if it can be inferred, switch to a faster method of parsing them. In some cases this can increase the parsing speed by 5-10x.

> *Deprecated since version 2.0.0:* A strict version of this argument is now the default, passing it has no effect.

**keep_date_col : *bool, default False***

If `True` and `parse_dates` specifies combining multiple columns then keep the original columns.

**date_parser : *Callable, optional***

Function to use for converting a sequence of string columns to an array of `datetime` instances. The default uses `dateutil.parser.parser` to do the conversion. pandas will try to call `date_parser` in three different ways, advancing to the next if an exception occurs: 1) Pass one or more arrays (as defined by `parse_dates`) as arguments; 2) concatenate (row-wise) the string values from the columns defined by `parse_dates` into a single array and pass that; and 3) call `date_parser` once for each row using one or more strings (corresponding to the columns defined by `parse_dates`) as arguments.

> *Deprecated since version 2.0.0:* Use `date_format` instead, or read in as `object` and then apply `to_datetime()` as-needed.

**date_format : *str or dict of column -> format, optional***

Format to use for parsing dates when used in conjunction with `parse_dates`. For anything more complex, please read in as `object` and then apply `to_datetime()` as-needed.

> *New in version 2.0.0.*

**dayfirst : *bool, default False***

DD/MM format dates, international and European format.

**cache_dates : *bool, default True***

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.

**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.

**chunksize : *int, optional***

Number of lines to read from the file per chunk. Passing a value will cause the function to return a `TextFileReader` object for iteration. See the IO Tools docs for more information on `iterator` and `chunksize`.

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

**compression : *str or dict, default 'infer'***

For on-the-fly decompression of on-disk data. If 'infer' and 'filepath_or_buffer' is path-like, then detect compression from the following extensions: '.gz', '.bz2', '.zip', '.xz', '.zst', '.tar', '.tar.gz', '.tar.xz' or '.tar.bz2' (otherwise no compression). If using 'zip' or 'tar', the ZIP file must contain only one data file to be read in. Set to `None` for no decompression. Can also be a dict with key `'method'` set to one of {`'zip'`, `'gzip'`, `'bz2'`, `'zstd'`, `'xz'`, `'tar'`} and other key-value pairs are forwarded to `zipfile.ZipFile`, `gzip.GzipFile`, `bz2.BZ2File`, `zstandard.ZstdDecompressor`, `lzma.LZMAFile` or `tarfile.TarFile`, respectively. As an example, the following could be passed for Zstandard decompression using a custom compression dictionary: `compression={'method': 'zstd', 'dict_data': my_compression_dict}`.

> *New in version 1.5.0:* Added support for *.tar* files.

> *Changed in version 1.4.0:* Zstandard support.

**thousands : *str (length 1), optional***

Character acting as the thousands separator in numerical values.

**decimal : *str (length 1), default '.'***

Character to recognize as decimal point (e.g., use ',' for European data).

**lineterminator : *str (length 1), optional***

Character used to denote a line break. Only valid with C parser.

**quotechar : *str (length 1), optional***

Character used to denote the start and end of a quoted item. Quoted items can include the `delimiter` and it will be ignored.

**quoting : *{0 or csv.QUOTE_MINIMAL, 1 or csv.QUOTE_ALL, 2 or csv.QUOTE_NONNUMERIC, 3 or csv.QUOTE_NONE}, default csv.QUOTE_MINIMAL***

Control field quoting behavior per `csv.QUOTE_*` constants. Default is `csv.QUOTE_MINIMAL` (i.e., 0) which implies that only fields containing special characters are quoted (e.g., characters defined in `quotechar`, `delimiter`, or `lineterminator`.

**doublequote : *bool, default True***

When `quotechar` is specified and `quoting` is not `QUOTE_NONE`, indicate whether or not to interpret two consecutive `quotechar` elements INSIDE a field as a single `quotechar` element.

**escapechar : *str (length 1), optional***

Character used to escape other characters.

**comment : *str (length 1), optional***

Character indicating that the remainder of line should not be parsed. If found at the beginning of a line, the line will be ignored altogether. This parameter must be a single character. Like empty lines (as long as `skip_blank_lines=True`), fully commented lines are ignored by the parameter `header` but not by `skiprows`. For example, if `comment='#'`, parsing `#empty\na,b,c\n1,2,3` with `header=0` will result in `'a,b,c'` being treated as the header.

**encoding : *str, optional, default 'utf-8'***

Encoding to use for UTF when reading/writing (ex. `'utf-8'`). List of Python standard encodings.

> *Changed in version 1.2:* When `encoding` is `None`, `errors='replace'` is passed to `open()`. Otherwise, `errors='strict'` is passed to `open()`. This behavior was previously only the case for `engine='python'`.

> *Changed in version 1.3.0:* `encoding_errors` is a new argument. `encoding` has no longer an influence on how encoding errors are handled.

**encoding_errors : *str, optional, default 'strict'***

How encoding errors are treated. List of possible values .

> *New in version 1.3.0.*

**dialect : *str or csv.Dialect, optional***

If provided, this parameter will override values (default or not) for the following parameters: `delimiter`, `doublequote`, `escapechar`, `skipinitialspace`, `quotechar`, and `quoting`. If it is necessary to override values, a `ParserWarning` will be issued. See `csv.Dialect` documentation for more details.

**on_bad_lines : *{'error', 'warn', 'skip'} or Callable, default 'error'***

Specifies what to do upon encountering a bad line (a line with too many fields). Allowed values are :

- `'error'`, raise an Exception when a bad line is encountered.
- `'warn'`, raise a warning when a bad line is encountered and skip that line.
- `'skip'`, skip bad lines without raising or warning when they are encountered.

> *New in version 1.3.0.*

> *New in version 1.4.0:*
>
> - Callable, function with signature `(bad_line: list[str]) -> list[str] | None` that will process a single bad line. `bad_line` is a list of strings split by the `sep`. If the function returns `None`, the bad line will be ignored. If the function returns a new `list` of strings with more elements than expected, a `ParserWarning` will be emitted while dropping extra elements. Only supported when `engine='python'`

**delim_whitespace : *bool, default False***

Specifies whether or not whitespace (e.g. `' '` or `'\t'`) will be used as the `sep` delimiter. Equivalent to setting `sep='\s+'`. If this option is set to `True`, nothing should be passed in for the `delimiter` parameter.

**low_memory : *bool, default True***

Internally process the file in chunks, resulting in lower memory use while parsing, but possibly mixed type inference. To ensure no mixed types either set `False`, or specify the type with the `dtype` parameter. Note that the entire file is read into a single `DataFrame` regardless, use the `chunksize` or `iterator` parameter to return the data in chunks. (Only valid with C parser).

**memory_map : *bool, default False***

If a filepath is provided for `filepath_or_buffer`, map the file object directly onto memory and access the data directly from there. Using this option can improve performance because there is no longer any I/O overhead.

**float_precision : *{'high', 'legacy', 'round_trip'}, optional***

Specifies which converter the C engine should use for floating-point values. The options are `None` or `'high'` for the ordinary converter, `'legacy'` for the original lower precision pandas converter, and `'round_trip'` for the round-trip converter.

> *Changed in version 1.2.*

**storage_options : *dict, optional***

Extra options that make sense for a particular storage connection, e.g. host, port, username, password, etc. For HTTP(S) URLs the key-value pairs are forwarded to `urllib.request.Request` as header options. For other URLs (e.g. starting with "s3://", and "gcs://") the key-value pairs are forwarded to `fsspec.open`. Please see `fsspec` and `urllib` for more details, and for more examples on storage options refer here.

> *New in version 1.2.*

**dtype_backend : *{'numpy_nullable', 'pyarrow'}, default 'numpy_nullable'***

Back-end data type applied to the resultant `DataFrame` (still experimental). Behaviour is as follows:

- `"numpy_nullable"` : returns nullable-dtype-backed `DataFrame` (default).
- `"pyarrow"` : returns pyarrow-backed nullable `ArrowDtype` DataFrame.

> *New in version 2.0.*

**Returns:**

**DataFrame or TextFileReader**

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_table`
Read general delimited 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