pandas.read_html

pandas.**read_html**(*io*, *match='.+'*, *flavor=None*, *header=None*, *index_col=None*, *skiprows=None*, *attrs=None*, *parse_dates=False*, *tupleize_cols=False*, *thousands=', '*, *encoding=None*)

Read HTML tables into a `list` of `DataFrame` objects.

Parameters:

io *: str or file-like*

A URL, a file-like object, or a raw string containing HTML. Note that lxml only accepts the http, ftp and file url protocols. If you have a URL that starts with `'https'` you might try removing the `'s'`.

match *: str or compiled regular expression, optional*

The set of tables containing text matching this regex or string will be returned. Unless the HTML is extremely simple you will probably need to pass a non-empty string here. Defaults to '.+' (match any non-empty string). The default value will return all tables contained on a page. This value is converted to a regular expression so that there is consistent behavior between Beautiful Soup and lxml.

flavor *: str or None, container of strings*

The parsing engine to use. 'bs4' and 'html5lib' are synonymous with each other, they are both there for backwards compatibility. The default of `None` tries to use `lxml` to parse and if that fails it falls back on `bs4` + `html5lib`.

header *: int or list-like or None, optional*

The row (or list of rows for a **`MultiIndex`**) to use to make the columns headers.

index_col *: int or list-like or None, optional*

The column (or list of columns) to use to create the index.

skiprows *: int or list-like or slice or None, optional*

0-based. Number of rows to skip after parsing the column integer. If a sequence of integers or a slice is given, will skip the rows indexed by that sequence. Note that a single element sequence means 'skip the nth row' whereas an integer means 'skip n rows'.

attrs *: dict or None, optional*

This is a dictionary of attributes that you can pass to use to identify the table in the HTML. These are not checked for validity before being passed to lxml or Beautiful Soup. However, these attributes must be valid HTML table attributes to work correctly. For example,

```
attrs = {'id': 'table'}
```

is a valid attribute dictionary because the 'id' HTML tag attribute is a valid HTML attribute for *any* HTML tag as per this document.

```
attrs = {'asdf': 'table'}
```

is *not* a valid attribute dictionary because 'asdf' is not a valid HTML attribute even if it is a valid XML attribute. Valid HTML 4.01 table attributes can be found here. A working draft of the HTML 5 spec can be found here. It contains the latest information on table attributes for the modern web.

parse_dates *: bool, optional*

See **`read_csv()`** for more details.

tupleize_cols *: bool, optional*

If `False` try to parse multiple header rows into a **`MultiIndex`**, otherwise return raw tuples. Defaults to `False`.

thousands *: str, optional*

Separator to use to parse thousands. Defaults to `','`.

encoding *: str or None, optional*

The encoding used to decode the web page. Defaults to `None`.``None`` preserves the previous encoding behavior, which depends on the underlying parser library (e.g., the parser library will try to use the encoding provided by the document).

Returns: **dfs** *: list of DataFrames*

See also: **`pandas.read_csv`**

Notes

Before using this function you should read the *gotchas about the HTML parsing libraries*.

Expect to do some cleanup after you call this function. For example, you might need to manually assign column names if the column names are converted to NaN when you pass the *header=0* argument. We try to assume as little as possible about the structure of the table and push the idiosyncrasies of the HTML contained in the table to the user.

This function searches for `<table>` elements and only for `<tr>` and `<th>` rows and `<td>` elements within each `<tr>` or `<th>` element in the table. `<td>` stands for “table data”.

Similar to **`read_csv()`** the *header* argument is applied **after** *skiprows* is applied.

This function will *always* return a list of **`DataFrame`** *or* it will fail, e.g., it will *not* return an empty list.

Examples

See the *read_html documentation in the IO section of the docs* for some examples of reading in HTML tables.