pandas.read_xml

pandas.read_xml(*path_or_buffer*, *xpath='./*'*, *namespaces=None*, *elems_only=False*, *attrs_only=False*, *names=None*, *dtype=None*, *converters=None*, *parse_dates=None*, *encoding='utf-8'*, *parser='lxml'*, *stylesheet=None*, *compression='infer'*, *storage_options=None*) [source]

Read XML document into a `DataFrame` object.

> **New in version 1.3.0.**

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

String, path object (implementing `os.PathLike[str]`), or file-like object implementing a `read()` function. The string can be any valid XML string or a path. The string can further be a URL. Valid URL schemes include http, ftp, s3, and file.

xpath : *str, optional, default './*'*

The XPath to parse required set of nodes for migration to DataFrame. XPath should return a collection of elements and not a single element. Note: The `etree` parser supports limited XPath expressions. For more complex XPath, use `lxml` which requires installation.

namespaces : *dict, optional*

The namespaces defined in XML document as dicts with key being namespace prefix and value the URI. There is no need to include all namespaces in XML, only the ones used in `xpath` expression. Note: if XML document uses default namespace denoted as *xmlns='<URI>'* without a prefix, you must assign any temporary namespace prefix such as 'doc' to the URI in order to parse underlying nodes and/or attributes. For example,

```
namespaces = {"doc": "https://example.com"}
```

elems_only : *bool, optional, default False*

Parse only the child elements at the specified `xpath`. By default, all child elements and non-empty text nodes are returned.

attrs_only : *bool, optional, default False*

Parse only the attributes at the specified `xpath`. By default, all attributes are returned.

names : *list-like, optional*

Column names for DataFrame of parsed XML data. Use this parameter to rename original element names and distinguish same named elements.

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.

> **New in version 1.5.0.**

converters : *dict, optional*

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

> **New in version 1.5.0.**

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

Identifiers to parse index or columns to datetime. 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.
- list of lists. 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’

> ***New in version 1.5.0.***

encoding : *str, optional, default ‘utf-8’*

Encoding of XML document.

parser : *{‘lxml’,’etree’}, default ‘lxml’*

Parser module to use for retrieval of data. Only ‘lxml’ and ‘etree’ are supported. With ‘lxml’ more complex XPath searches and ability to use XSLT stylesheet are supported.

stylesheet : *str, path object or file-like object*

A URL, file-like object, or a raw string containing an XSLT script. This stylesheet should flatten complex, deeply nested XML documents for easier parsing. To use this feature you must have `lxml` module installed and specify ‘lxml’ as `parser`. The `xpath` must reference nodes of transformed XML document generated after XSLT transformation and not the original XML document. Only XSLT 1.0 scripts and not later versions is currently supported.

compression : *str or dict, default ‘infer’*

For on-the-fly decompression of on-disk data. If ‘infer’ and ‘path_or_buffer’ is path-like, then detect compression from the following extensions: ‘.gz’, ‘.bz2’, ‘.zip’, ‘.xz’, or ‘.zst’ (otherwise no compression). If using ‘zip’, 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'`} and other key-value pairs are forwarded to `zipfile.ZipFile`, `gzip.GzipFile`, `bz2.BZ2File`, or `zstandard.ZstdDecompressor`, 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}`.

> ***Changed in version 1.4.0:*** Zstandard support.

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.

Returns: **df**

A DataFrame.

See also

`read_json`
: Convert a JSON string to pandas object.

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

Notes

This method is best designed to import shallow XML documents in following format which is the ideal fit for the two-dimensions of a `DataFrame` (row by column).

```
<root>
    <row>
      <column1>data</column1>
      <column2>data</column2>
      <column3>data</column3>
      ...
   </row>
   <row>
      ...
   </row>
   ...
</root>
```

As a file format, XML documents can be designed any way including layout of elements and attributes as long as it conforms to W3C specifications. Therefore, this method is a convenience handler for a specific flatter design and not all possible XML structures.

However, for more complex XML documents, `stylesheet` allows you to temporarily redesign original document with XSLT (a special purpose language) for a flatter version for migration to a DataFrame.

This function will *always* return a single **`DataFrame`** or raise exceptions due to issues with XML document, `xpath`, or other parameters.

Examples

```
>>> xml = '''<?xml version='1.0' encoding='utf-8'?>
... <data xmlns="http://example.com">
...  <row>
...    <shape>square</shape>
...    <degrees>360</degrees>
...    <sides>4.0</sides>
...  </row>
...  <row>
...    <shape>circle</shape>
...    <degrees>360</degrees>
...    <sides/>
...  </row>
...  <row>
...    <shape>triangle</shape>
...    <degrees>180</degrees>
...    <sides>3.0</sides>
...  </row>
... </data>'''
```

```
>>> df = pd.read_xml(xml)
>>> df
      shape  degrees  sides
0    square      360    4.0
1    circle      360    NaN
2  triangle      180    3.0
```

```
>>> xml = '''<?xml version='1.0' encoding='utf-8'?>
... <data>
...   <row shape="square" degrees="360" sides="4.0"/>
...   <row shape="circle" degrees="360"/>
...   <row shape="triangle" degrees="180" sides="3.0"/>
... </data>'''
```

```
>>> df = pd.read_xml(xml, xpath=".//row")
>>> df
      shape  degrees  sides
0    square      360    4.0
1    circle      360    NaN
2  triangle      180    3.0
```

```
>>> xml = '''<?xml version='1.0' encoding='utf-8'?>
... <doc:data xmlns:doc="https://example.com">
...   <doc:row>
...     <doc:shape>square</doc:shape>
...     <doc:degrees>360</doc:degrees>
...     <doc:sides>4.0</doc:sides>
...   </doc:row>
...   <doc:row>
...     <doc:shape>circle</doc:shape>
...     <doc:degrees>360</doc:degrees>
...     <doc:sides/>
...   </doc:row>
...   <doc:row>
...     <doc:shape>triangle</doc:shape>
...     <doc:degrees>180</doc:degrees>
...     <doc:sides>3.0</doc:sides>
...   </doc:row>
... </doc:data>'''
```

```
>>> df = pd.read_xml(xml,
...                  xpath="//doc:row",
...                  namespaces={"doc": "https://example.com"})
>>> df
      shape  degrees  sides
0    square      360    4.0
1    circle      360    NaN
2  triangle      180    3.0
```

Previous
pandas.io.formats.style.Styler.to_html

Next
pandas.DataFrame.to_xml

Created using Sphinx 4.4.0.