Importing Text File with Pandas

Xavier Morera

Helping developers create amazing applications

@xmorera / www.xaviermorera.com / www.bigdatainc.org

pandas

panel data

Pandas

Library for data manipulation and analysis

Used to analyze data sets

- That include observations
- Over multiple time periods
- For the same set of individuals or entities

Can also be used for more "generic" data sets

Pandas

Functionality for reading and writing in multiple formats

- CSV, Excel, SQL databases...

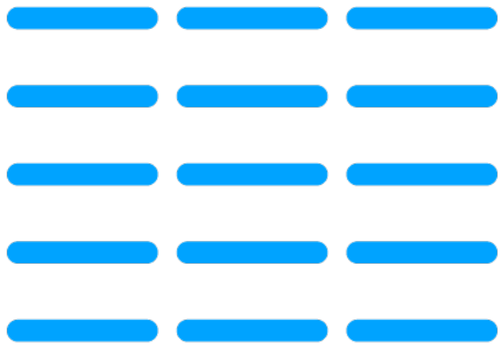
One of the most commonly used libraries for data manipulation in Python

- Built on top of Numpy

Dataframe is the first-class citizen

The Pandas Dataframe

Pandas Dataframe



Two-dimensional table-like data structure

Consists of rows and columns

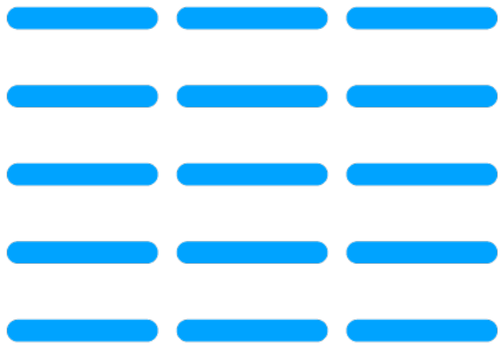
- Row represents observation or data point
- Column represents a variable or feature

Like a spreadsheet or SQL table

Each column can have a different data type

- String, integer, float...

Pandas Dataframe



Can be created from multiple data sources

Can handle a wide variety of data formats

Provide multiple built-in functions for data manipulation and data cleansing

Pandas Series

Dataframe is a collection of Pandas Series

- Series represents a column of data

One-dimensional labeled array

- Contains two arrays
- One for the data and another for the index

Index is a set of labels that identify each element in the data array

- Can be of any type and contain duplicates

Data array can hold any type of data

Pandas Dataframe

A file like this one can be loaded using pandas


```
5,"1","2014-05-13T23:58:30.457",9,448,"2014-05-14T00:36:31.077","How can I do simple
machine learning without hard-coding behavior?","<machine-learning>",1,1,1,"2014-05-
14T14:40:25.950"
7,"1","2014-05-14T00:11:06.457",4,388,"2014-05-16T13:45:00.237","What open-source books
(or other materials) provide a relatively thorough overview of data
science?","<education><open-source>",3,4,1,"2014-05-14T08:40:54.950"
9,"2","2014-05-14T00:36:31.077",5,"","2014-05-14T00:36:31.077","","","",0,"","
10,"2","2014-05-14T00:53:43.273",12,"","2014-05-14T00:53:43.273","","","",1,"","
14,"1","2014-05-14T01:25:59.677",21,1243,"2014-06-20T17:36:05.023","Is Data Science the
Same as Data Mining?","<data-mining><definitions>",4,1,4,""
15,"1","2014-05-14T01:41:23.110",2,543,"2014-05-14T01:41:23.110","What are the advantages
and disadvantages of SQL versus NoSQL in data science?","<databases>",0,1,"","2014-05-
14T07:41:49.437"
```

Pandas Dataframe

A file like this one can be loaded using pandas

```
"Id","PostTypeId","CreationDate","Score","ViewCount","LastActivityDate","Title","Tags","A
nswerCount","CommentCount","FavoriteCount","ClosedDate"
5,"1","2014-05-13T23:58:30.457",9,448,"2014-05-14T00:36:31.077","How can I do simple
machine learning without hard-coding behavior?","<machine-learning>",1,1,1,"2014-05-
14T14:40:25.950"
7,"1","2014-05-14T00:11:06.457",4,388,"2014-05-16T13:45:00.237","What open-source books
(or other materials) provide a relatively thorough overview of data
science?","<education><open-source>",3,4,1,"2014-05-14T08:40:54.950"
9,"2","2014-05-14T00:36:31.077",5,"","2014-05-14T00:36:31.077","","","",0,"",""
10,"2","2014-05-14T00:53:43.273",12,"","2014-05-14T00:53:43.273","","","",1,"",""
14,"1","2014-05-14T01:25:59.677",21,1243,"2014-06-20T17:36:05.023","Is Data Science the
Same as Data Mining?","<data-mining><definitions>",4,1,4,""
15,"1","2014-05-14T01:41:23.110",2,543,"2014-05-14T01:41:23.110","What are the advantages
and disadvantages of SQL versus NoSQL in data science?","<databases>",0,1,"","2014-05-
14T07:41:49.437"
```

Show Source

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 ','*

Show Source

pandas.read_table

```
pandas.read_table(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=False, 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 general delimited 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 '\t' (tab-stop)*

```
import pandas as pd
posts_csv = pd.read_csv('posts-100.csv')
```

Importing Text & CSV Files Using Pandas

After installing pandas, import it
Load CSV data using read_csv()
    Specify the filename and optionally parameters to indicate how to load the data
Data is imported into a dataframe
    Many methods to control how to manipulate the data

```
posts_csv = pd.read_csv('posts-100.csv')
posts_csv.head(2)
```

Importing with read_csv()

All rows loaded
Display the first n rows using head(n)
Useful for quickly inspecting contents of a data set without displaying all the data

```
posts_csv.values()
```

The ndarray in Your Pandas DataFrame

DataFrame depends and interoperates with NumPy

- Pandas uses internally a Numpy ndarray

values returns a NumPy representation of the dataframe

- With values, but axes labels are removed

```
posts = pd.read_csv('posts-100.csv', nrows=3)
posts = pd.read_csv('posts-100.csv', nrows=3, skiprows=3)
posts = pd.read_csv('posts-100.csv', nrows=3, skipfooter=1)
```

Read Pieces of Large Files

Specify how many rows to read with nrows

And where to start, with skiprows

Can also specify skipfooter to ignore lines at the end

```
posts = pd.read_csv('posts-100.csv',
                    skiprows=lambda x: x % 2 != 0)
```

Specify Rows Using a Function

More advanced method for specifying which rows to load
Referred as callable
- Use a named function or an anonymous function
- Evaluate against row indices and determine which rows to skip

```
posts = pd.read_csv('posts-100.csv', usecols=[0,6,7,8])
```

Loading (Certain) Columns

Specify which columns to load with usecols

- List or a function

Columns get a name

- Careful if your file does not have column header names in the first row

```
posts = pd.read_csv('posts-100.csv', header=None)

posts = pd.read_csv('posts-100.csv', header=None, prefix='Col')

hf = ['New_Id', 'New_PostTypeId', 'New_CreationDate]
posts = pd.read_csv('posts-100.csv', names=hf)
```

Column Headers Not Included in File

If file does not include header row, you need to indicate this to pandas

- One automatically assigned
- Possible to set a prefix

Or provide column names

```
posts_header = pd.read_csv('posts-100-header.csv’)

posts_header.columns
```

Headers Included in File

Some files include headers

- Use them, they are useful for referring to columns by name
- Retrieve using columns

Can infer header

```
pd.read_csv('posts-100-header.csv',
            usecols=[0, 1, 2, 7]).dtypes

pd.read_csv('posts-100-header.csv', usecols=[0, 1, 2, 7],
            dtype={'PostTypeId': float}).dtypes
```

Specify Column Types on Load

Types inferred on load

- Review using dtypes

You can manually set specific types using dtype

```
import re

posts = pd.read_csv('posts-100-header.csv',
            usecols=[0, 1, 2, 7],
            converters={'Tags': lambda x:
                        re.findall('<[A-Za-z0-9_-]*>',x)})
```

Apply Function to a Column

Use converters to apply functions on columns

Can use a named or anonymous function

For example, take a string and convert to a list

```
posts_date = pd.read_csv('posts-100-header.csv',
                         usecols=[0, 1, 2, 7],
                         parse_dates=['CreationDate'])
```

Loading Dates

Dates and times are important, yet they can be complex to deal with

- May be imported as strings

Use parse_dates to convert into a Timestamp

```
pd.read_csv('posts-100-header.csv',
            usecols=[0, 3, 4, 8, 9, 10], na_filter=False)

pd.read_csv('posts-100-header.csv',
            usecols=[0, 3, 4, 8, 9, 10], na_filter=True)
```

Missing Values

Data is not perfect, there may be missing values
Detect missing values with na_filter, which by default is set to True
    If True, missing values replaced with the default missing value representations
Use na_values for other markers and keep_default_na to include default values

```
pd.read_csv('posts-100.tsv', sep='\t').head()
```

```
pd.read_csv('posts-100.tsv', delimiter='\t').head()
```

Tabular Data

Other delimiters available

- Set \t for tab, with sep or delimiter

Use read_table for tab separated values

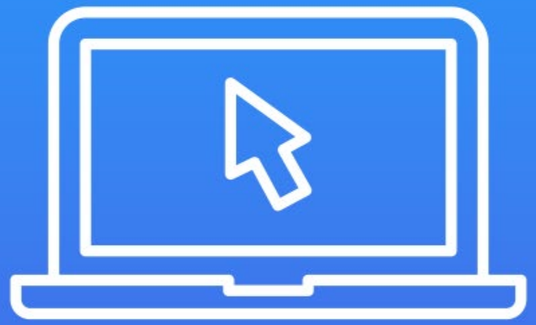
```
remote_file =
'https://raw.githubusercontent.com/xmorera/sample-
data/master/csv/posts-100.csv'

posts = pd.read_csv(remote_file)
```

Load from URL

Load data from a remote file using a URL
Valid URL schemes
    http, ftp, s3, and file

Importing Text Files Using Pandas with read_csv and read_table



Importing text files using Pandas with read_csv and read_table

Reading Other Kinds of Files with Pandas

pandas.read_json

pandas.read_json(*path_or_buf, *, orient=None, typ='frame', dtype=None, convert_axes=None, convert_dates=True, keep_default_dates=True, numpy=False, precise_float=False, date_unit=None, encoding=None, encoding_errors='strict', lines=False, chunksize=None, compression='infer', nrows=None, storage_options=None*) [source]

Convert a JSON string to pandas object.

Parameters: **path_or_buf** : *a valid JSON 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, and file. For file URLs, a host is expected. A local file could be: `file://localhost/path/to/table.json`.
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`.

orient : *str*

Indication of expected JSON string format. Compatible JSON strings can be produced by `to_json()` with a corresponding orient value. The set of possible orients is:

- `'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

The allowed and default values depend on the value of the *typ* parameter.

- when `typ == 'series'`,
  - allowed orients are `{'split','records','index'}`

Show Source

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,
iterparse=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,

Reading Other Kinds of Files with Pandas

JSON
and
XML

Excel

HTML tables

SQL databases

Pickle
Stata
SAS

Takeaway

Pandas is an open-source library

- Data manipulation and analysis

Read and write in multiple formats

- Tabular data (CSV, TSV)
- Excel, JSON, XML, SQL databases...

Takeaway

First-class citizen is the dataframe

- Internally uses Numpy ndarray

Two-dimensional table-like data structure

Consists of rows and columns

- Row represents observation or data point
- Column represents a variable or feature

Like a spreadsheet or SQL table

Takeaway

Use read_csv to load tabular data

- CSV files

Many parameters to control how files are loaded and displayed

- head, nrows, skiprows, skipfooter, usecols, names, header, columns...

Use read_table to load TSV files

Up Next:

Final Takeaway