**Pandas**

**Pandas** is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

**Pandas** consists of the following elements:

* A set of labeled array data structures, the primary of which are Series and DataFrame.
* Index objects enabling both simple axis indexing and multi-level / hierarchical axis indexing.
* An integrated group by engine for aggregating and transforming data sets.
* Date range generation (date\_range) and custom date offsets enabling the implementation of customized frequencies.
* Input/Output tools: loading tabular data from flat files (CSV, delimited, Excel 2003), and saving and loading pandas objects from the fast and efficient PyTables/HDF5 format.
* Memory-efficient “sparse” versions of the standard data structures for storing data that is mostly missing or mostly constant (some fixed value).
* Moving window statistics (rolling mean, rolling standard deviation, etc.).

## Data Structures[¶](https://pandas.pydata.org/pandas-docs/version/0.23.0/overview.html#data-structures)

| **Dimensions** | **Name** | **Description** |
| --- | --- | --- |
| 1 | Series | 1D labeled homogeneously-typed array |
| 2 | DataFrame | General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column |

## Missing Data

pandas primarily uses the value np.nan to represent missing data. It is by default not included in computations. See the [Missing Data section](https://pandas.pydata.org/pandas-docs/version/0.23.0/missing_data.html#missing-data).

Reindexing allows you to change/add/delete the index on a specified axis.

Grouping

By “group by” we are referring to a process involving one or more of the following steps:

* **Splitting** the data into groups based on some criteria
* **Applying** a function to each group independently
* **Combining** the results into a data structure

# Indexing and Selecting Data

Link: https://pandas.pydata.org/pandas-docs/version/0.23.0/indexing.html

The axis labeling information in pandas objects serves many purposes:

* Identifies data (i.e. provides metadata) using known indicators, important for analysis, visualization, and interactive console display.
* Enables automatic and explicit data alignment.
* Allows intuitive getting and setting of subsets of the data set.

## Different Choices for Indexing

Object selection has had a number of user-requested additions in order to support more explicit location based indexing. Pandas now supports three types of multi-axis indexing.

* .loc is primarily label based, but may also be used with a boolean array. .loc will raise KeyError when the items are not found. Allowed inputs are:
  + A single label, e.g. 5 or 'a' (Note that 5 is interpreted as a label of the index. This use is **not** an integer position along the index.).
  + A list or array of labels ['a', 'b', 'c'].
  + A slice object with labels 'a':'f' (Note that contrary to usual python slices, **both** the start and the stop are included, when present in the index! See [Slicing with labels](https://pandas.pydata.org/pandas-docs/version/0.23.0/indexing.html#indexing-slicing-with-labels).).
  + A boolean array
  + A callable function with one argument (the calling Series, DataFrame or Panel) and that returns valid output for indexing (one of the above).

.iloc is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array. .iloc will raise IndexError if a requested indexer is out-of-bounds, except slice indexers which allow out-of-bounds indexing. (this conforms with Python/NumPy slice semantics). Allowed inputs are:

* An integer e.g. 5.
* A list or array of integers [4, 3, 0].
* A slice object with ints 1:7.
* A boolean array.
* A callable function with one argument (the calling Series, DataFrame or Panel) and that returns valid output for indexing (one of the above).

.loc, .iloc, and also [] indexing can accept a callable as indexer. Getting values from an object with multi-axes selection uses the following notation (using .loc as an example, but the following applies to .iloc as well). Any of the axes accessors may be the null slice :. Axes left out of the specification are assumed to be :, e.g. p.loc['a'] is equivalent to p.loc['a', :, :].

| **Object Type** | **Indexers** |
| --- | --- |
| Series | s.loc[indexer] |
| DataFrame | df.loc[row\_indexer,column\_indexer] |
| Panel | p.loc[item\_indexer,major\_indexer,minor\_indexer] |

| **Object Type** | **Selection** | **Return Value Type** |
| --- | --- | --- |
| Series | series[label] | scalar value |
| DataFrame | frame[colname] | Series corresponding to colname |
| Panel | panel[itemname] | DataFrame corresponding to the itemname |

## IX Indexer is Deprecated[¶](https://pandas.pydata.org/pandas-docs/version/0.23.0/indexing.html#ix-indexer-is-deprecated)

.ix offers a lot of magic on the inference of what the user wants to do. To wit, .ix can decide to index positionallyOR via labels depending on the data type of the index. This has caused quite a bit of user confusion over the years.

The recommended methods of indexing are:

* .loc if you want to label index.
* .iloc if you want to positionally index.

# IO Tools (Text, CSV, HDF5, …)

The pandas I/O API is a set of top level reader functions accessed like [**pandas.read\_csv()**](https://pandas.pydata.org/pandas-docs/version/0.23.0/generated/pandas.read_csv.html#pandas.read_csv) that generally return a pandas object. The corresponding writer functions are object methods that are accessed like [**DataFrame.to\_csv()**](https://pandas.pydata.org/pandas-docs/version/0.23.0/generated/pandas.DataFrame.to_csv.html#pandas.DataFrame.to_csv). Below is a table containing available readers and writers.

| **Format Type** | **Data Description** | **Reader** | **Writer** |
| --- | --- | --- | --- |
| Text | [CSV](https://en.wikipedia.org/wiki/Comma-separated_values) | [read\_csv](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-read-csv-table) | [to\_csv](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-store-in-csv) |
| Text | [JSON](http://www.json.org/) | [read\_json](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-json-reader) | [to\_json](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-json-writer) |
| Text | [HTML](https://en.wikipedia.org/wiki/HTML) | [read\_html](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-read-html) | [to\_html](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-html) |
| Text | Local clipboard | [read\_clipboard](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-clipboard) | [to\_clipboard](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-clipboard) |
| Binary | [MS Excel](https://en.wikipedia.org/wiki/Microsoft_Excel) | [read\_excel](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-excel-reader) | [to\_excel](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-excel-writer) |
| Binary | [HDF5 Format](https://support.hdfgroup.org/HDF5/whatishdf5.html) | [read\_hdf](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-hdf5) | [to\_hdf](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-hdf5) |
| Binary | [Feather Format](https://github.com/wesm/feather) | [read\_feather](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-feather) | [to\_feather](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-feather) |
| Binary | [Parquet Format](https://parquet.apache.org/) | [read\_parquet](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-parquet) | [to\_parquet](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-parquet) |
| Binary | [Msgpack](http://msgpack.org/index.html) | [read\_msgpack](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-msgpack) | [to\_msgpack](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-msgpack) |
| Binary | [Stata](https://en.wikipedia.org/wiki/Stata) | [read\_stata](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-stata-reader) | [to\_stata](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-stata-writer) |
| Binary | [SAS](https://en.wikipedia.org/wiki/SAS_(software)) | [read\_sas](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-sas-reader) |  |
| Binary | [Python Pickle Format](https://docs.python.org/3/library/pickle.html) | [read\_pickle](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-pickle) | [to\_pickle](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-pickle) |
| SQL | [SQL](https://en.wikipedia.org/wiki/SQL) | [read\_sql](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-sql) | [to\_sql](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-sql) |
| SQL | [Google Big Query](https://en.wikipedia.org/wiki/BigQuery) | [read\_gbq](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-bigquery) | [to\_gbq](https://pandas.pydata.org/pandas-docs/version/0.23.0/io.html#io-bigquery) |

# Intro to Data Structures

## Series

**Series** is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the **index**. The basic method to create a Series is to call:

**>>>** s = pd.Series(data, index=index)

Here, data can be many different things:

* a Python dict
* an ndarray
* a scalar value (like 5)

The passed **index** is a list of axis labels

DataFrame

**DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:

* Dict of 1D ndarrays, lists, dicts, or Series
* 2-D numpy.ndarray
* Structured or record ndarray
* A Series
* Another DataFrame

Along with the data, you can optionally pass **index** (row labels) and **columns** (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

Panel

Panel is a somewhat less-used, but still important container for 3-dimensional data. The term panel data is derived from econometrics and is partially responsible for the name pandas: pan(el)-da(ta)-s. The names for the 3 axes are intended to give some semantic meaning to describing operations involving panel data and, in particular, econometric analysis of panel data. However, for the strict purposes of slicing and dicing a collection of DataFrame objects, you may find the axis names slightly arbitrary:

* **items**: axis 0, each item corresponds to a DataFrame contained inside
* **major\_axis**: axis 1, it is the **index** (rows) of each of the DataFrames
* **minor\_axis**: axis 2, it is the **columns** of each of the DataFrames

### From 3D ndarray with optional axis labels[¶](https://pandas.pydata.org/pandas-docs/stable/dsintro.html#from-3d-ndarray-with-optional-axis-labels)

**In [121]:** wp = pd.Panel(np.random.randn(2, 5, 4), items=['Item1', 'Item2'],

**.....:**  major\_axis=pd.date\_range('1/1/2000', periods=5),

**.....:**  minor\_axis=['A', 'B', 'C', 'D'])

**In [122]:** wp

**Out[122]:**

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 5 (major\_axis) x 4 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Minor\_axis axis: A to D

### From dict of DataFrame objects

**In [123]:** data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),

**.....:**  'Item2' : pd.DataFrame(np.random.randn(4, 2))}

**.....:**

**In [124]:** pd.Panel(data)

**Out[124]:**

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 4 (major\_axis) x 3 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 0 to 3

Minor\_axis axis: 0 to 2

**Numpy**

NumPy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called *axes*.

For example, the coordinates of a point in 3D space [1, 2, 1] has one axis. That axis has 3 elements in it, so we say it has a length of 3. In the example pictured below, the array has 2 axes. The first axis has a length of 2, the second axis has a length of 3.

[[ 1., 0., 0.],

[ 0., 1., 2.]]

NumPy’s array class is called ndarray. It is also known by the alias array. Note that numpy.array is not the same as the Standard Python Library class array.array, which only handles one-dimensional arrays and offers less functionality. The more important attributes of an ndarray object are:

**ndarray.ndim**

the number of axes (dimensions) of the array.

**ndarray.shape**

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with *n*rows and *m* columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

**ndarray.size**

the total number of elements of the array. This is equal to the product of the elements of shape.

**ndarray.dtype**

an object describing the type of the elements in the array. One can create or specify dtype’s using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

**ndarray.itemsize**

the size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize.

**ndarray.data**

the buffer containing the actual elements of the array. Normally, we won’t need to use this attribute because we will access the elements in an array using indexing facilities.

**numpy.arange([***start***, ]***stop***, [***step***, ]***dtype=None***)**

Return evenly spaced values within a given interval.

Values are generated within the half-open interval [start, stop) (in other words, the interval including start but excluding stop). For integer arguments the function is equivalent to the Python built-in range function, but returns an ndarray rather than a list.

**numpy.array(***object***,***dtype=None***,***copy=True***,***order='K'***,***subok=False***,***ndmin=0***)**

Create an array.

**numpy.zeros(**shape**,**dtype=float**,**order='C'**)**

Return a new array of given shape and type, filled with zeros.

**numpy.linspace(**start**,**stop**,**num=50**,**endpoint=True**,**retstep=False**,**dtype=None**)**

Return evenly spaced numbers over a specified interval.

Returns num evenly spaced samples, calculated over the interval [start, stop].

The endpoint of the interval can optionally be excluded.

**numpy.random.rand(***d0***,***d1***,***...***,***dn***)**

Random values in a given shape.

Create an array of the given shape and populate it with random samples from a uniform distribution over [0, 1).

**numpy.random.randn(***d0***,***d1***,***...***,***dn***)**

Return a sample (or samples) from the “standard normal” distribution.

If positive, int\_like or int-convertible arguments are provided, [**randn**](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randn.html#numpy.random.randn) generates an array of shape (d0, d1, ..., dn), filled with random floats sampled from a univariate “normal” (Gaussian) distribution of mean 0 and variance 1 (if any of the are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use [**numpy.random.standard\_normal**](https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.standard_normal.html#numpy.random.standard_normal) instead.

|  |  |
| --- | --- |
| **Parameters:** | **d0, d1, …, dn : *int, optional***  The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned. |
| **Returns:** | **Z : *ndarray or float***  A (d0, d1, ..., dn)-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied. |

**>>>** np.random.randn()

2.1923875335537315 #random

Two-by-four array of samples from N(3, 6.25):

**>>>** 2.5 \* np.random.randn(2, 4) + 3

array([[-4.49401501, 4.00950034, -1.81814867, 7.29718677], #random

[ 0.39924804, 4.68456316, 4.99394529, 4.84057254]]) #random

**numpy.fromfunction(**function**,**shape**,**\*\*kwargs**)**

Construct an array by executing a function over each coordinate.

The resulting array therefore has a value fn(x, y, z) at coordinate (x, y, z).

|  |  |
| --- | --- |
| **Parameters:** | **function : *callable***  The function is called with N parameters, where N is the rank of **shape**. Each parameter represents the coordinates of the array varying along a specific axis. For example, if **shape** were (2, 2), then the parameters would be array([[0, 0], [1, 1]]) and array([[0, 1], [0, 1]])  **shape : *(N,) tuple of ints***  Shape of the output array, which also determines the shape of the coordinate arrays passed to *function*.  **dtype : *data-type, optional***  Data-type of the coordinate arrays passed to *function*. By default, [**dtype**](https://docs.scipy.org/doc/numpy/reference/generated/numpy.dtype.html#numpy.dtype) is float. |
| **Returns:** | **fromfunction : *any***  The result of the call to *function* is passed back directly. Therefore the shape of [**fromfunction**](https://docs.scipy.org/doc/numpy/reference/generated/numpy.fromfunction.html#numpy.fromfunction) is completely determined by *function*. If *function* returns a scalar value, the shape of [**fromfunction**](https://docs.scipy.org/doc/numpy/reference/generated/numpy.fromfunction.html#numpy.fromfunction)would match the **shape** parameter. |

**>>>** np.fromfunction(**lambda** i, j: i == j, (3, 3), dtype=int)

array([[ True, False, False],

[False, True, False],

[False, False, True]])

**numpy.fromfile(***file***,***dtype=float***,***count=-1***,***sep=''***)**

Construct an array from data in a text or binary file.

A highly efficient way of reading binary data with a known data-type, as well as parsing simply formatted text files. Data written using the *tofile* method can be read using this function.