Python\_for\_Data\_Analysis\_Wes\_McKinney\_First\_release\_c05

APTER 5

Getting Started with pandas

pandas will be the primary library of interest throughout much of the rest of the book. It contains high-level data structures and manipulation tools designed to make data analysis fast and easy in Python. pandas is built on top of NumPy and makes it easy to use in NumPy-centric applications.

pandas será la principal biblioteca de interés durante gran parte del resto del libro. Contiene estructuras de datos de alto nivel y herramientas de manipulación diseñadas para hacer que el análisis de datos sea rápido y sencillo en Python. pandas está construido sobre NumPy y facilita su uso en aplicaciones centradas en NumPy.

As a bit of background, I started building pandas in early 2008 during my tenure at AQR, a quantitative investment management firm. At the time, I had a distinct set of requirements that were not well-addressed by any single tool at my disposal:

Como contexto, comencé a construir pandas a principios de 2008 durante mi mandato en AQR, una empresa de gestión de inversiones cuantitativas. En ese momento, tenía un conjunto distinto de requisitos que no eran bien atendidos por ninguna herramienta a mi disposición:

• Data structures with labeled axes supporting automatic or explicit data alignment. This prevents common errors resulting from misaligned data and working with differently-indexed data coming from different sources.

Estructuras de datos con ejes etiquetados que admiten la alineación de datos automática o explícita. Esto evita errores comunes que resultan de datos desalineados y de trabajar con datos indexados de manera diferente provenientes de diferentes fuentes.

• Integrated time series functionality.

Funcionalidad integrada de series temporales.

• The same data structures handle both time series data and non-time series data.

Las mismas estructuras de datos manejan tanto datos de series temporales como datos que no son series temporales.

• Arithmetic operations and reductions (like summing across an axis) would pass on the metadata (axis labels).

Las operaciones aritméticas y las reducciones (como la suma a lo largo de un eje) transmitirían los metadatos (etiquetas de los ejes).

• Flexible handling of missing data.

Manejo flexible de datos faltantes.

• Merge and other relational operations found in popular database databases (SQL-based, for example).

Fusionar y otras operaciones relacionales que se encuentran en bases de datos de bases de datos populares (basadas en SQL, por ejemplo).

I wanted to be able to do all of these things in one place, preferably in a language wellsuited to general purpose software development. Python was a good candidate language for this, but at that time there was not an integrated set of data structures and tools providing this functionality.

Quería poder hacer todas estas cosas en un solo lugar, preferiblemente en un lenguaje adecuado para el desarrollo de software de propósito general. Python era un buen lenguaje candidato para esto, pero en ese momento no existía un conjunto integrado de estructuras de datos y herramientas que proporcionaran esta funcionalidad.

Over the last four years, pandas has matured into a quite large library capable of solving a much broader set of data handling problems than I ever anticipated, but it has expanded in its scope without compromising the simplicity and ease-of-use that I desired from the very beginning. I hope that after reading this book, you will find it to be just as much of an indispensable tool as I do.

En los últimos cuatro años, Pandas ha madurado hasta convertirse en una biblioteca bastante grande capaz de resolver un conjunto de problemas de manejo de datos mucho más amplio de lo que jamás anticipé, pero ha ampliado su alcance sin comprometer la simplicidad y facilidad de uso que deseaba. desde el principio. Espero que después de leer este libro, encuentres que es una herramienta tan indispensable como yo.

In [1]: from pandas import Series, DataFrame

In [2]: import pandas as pd

Thus, whenever you see pd. in code, it’s referring to pandas. Series and DataFrame are used so much that I find it easier to import them into the local namespace.

Por lo tanto, cada vez que veas pd. en código, se refiere a pandas. Series y DataFrame se usan tanto que me resulta más fácil importarlos al espacio de nombres local.

Introduction to pandas Data Structures

To get started with pandas, you will need to get comfortable with its two workhorse data structures: Series and DataFrame. While they are not a universal solution for every problem, they provide a solid, easy-to-use basis for most applications.

Para comenzar con pandas, deberá sentirse cómodo con sus dos estructuras de datos potentes: Series y DataFrame. Si bien no son una solución universal para todos los problemas, proporcionan una base sólida y fácil de usar para la mayoría de las aplicaciones.

**Series**

A Series is a one-dimensional array-like object containing an array of data (of any NumPy data type) and an associated array of data labels, called its index. The simplest Series is formed from only an array of data:

Una serie es un objeto unidimensional similar a una matriz que contiene una matriz de datos (de cualquier tipo de datos NumPy) y una matriz asociada de etiquetas de datos, llamada índice. La Serie más simple se forma a partir de sólo una serie de datos:

In [4]: obj = Series([4, 7, -5, 3])

In [5]: obj

Out[5]:

0

4

1

7

2 -5

3

3

The string representation of a Series displayed interactively shows the index on the left and the values on the right. Since we did not specify an index for the data, a default one consisting of the integers 0 through N - 1 (where N is the length of the data) is created. You can get the array representation and index object of the Series via its values and index attributes, respectively:

La representación de cadena de una Serie mostrada de forma interactiva muestra el índice a la izquierda y los valores a la derecha. Como no especificamos un índice para los datos, se crea uno predeterminado que consta de números enteros del 0 al N - 1 (donde N es la longitud de los datos). Puede obtener la representación de matriz y el objeto de índice de la Serie a través de sus valores y atributos de índice, respectivamente:

In [6]: obj.values

Out[6]: array([ 4,

7, -5, 3])

In [7]: obj.index

Out[7]: Int64Index([0, 1, 2, 3])

Often it will be desirable to create a Series with an index identifying each data point:

A menudo será deseable crear una Serie con un índice que identifique cada punto de datos:

In [8]: obj2 = Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])

In [9]: obj2

Out[9]:

d

4

b

7

a -5

c

3

Out[10]: Index([d, b, a, c], dtype=object)

Compared with a regular NumPy array, you can use values in the index when selecting single values or a set of values:

En comparación con una matriz NumPy normal, puede utilizar valores en el índice al seleccionar valores individuales o un conjunto de valores:

In [11]: obj2['a']

Out[11]: -5

In [12]: obj2['d'] = 6

In [13]: obj2[['c', 'a', 'd']]

Out[13]:

c

3

a -5

d

6

NumPy array operations, such as filtering with a boolean array, scalar multiplication, or applying math functions, will preserve the index-value link:

Las operaciones de matriz NumPy, como el filtrado con una matriz booleana, la multiplicación escalar o la aplicación de funciones matemáticas, preservarán el vínculo índice-valor:

In [14]: obj2

Out[14]:

d

6

b

7

a -5

c

3

In [15]: obj2[obj2 > 0]

Out[15]:

d

6

b

7

c

3

In [16]: obj2 \* 2

Out[16]:

d

12

b

14

a

-10

c

6

In [17]: np.exp(obj2)

Out[17]:

d

403.428793

b

1096.633158

a

0.006738

c

20.085537

Another way to think about a Series is as a fixed-length, ordered dict, as it is a mapping of index values to data values. It can be substituted into many functions that expect a dict:

Otra forma de pensar en una serie es como un dictado ordenado y de longitud fija, ya que es una asignación de valores de índice a valores de datos. Se puede sustituir en muchas funciones que esperan un dict:

In [18]: 'b' in obj2

Out[18]: True

In [19]: 'e' in obj2

Out[19]: False

Should you have data contained in a Python dict, you can create a Series from it by passing the dict:

Si tiene datos contenidos en un dictado de Python, puede crear una serie a partir de él pasando el dictado:

In [20]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

In [21]: obj3 = Series(sdata)

In [22]: obj3

Out[22]:

Ohio

35000

Oregon

16000

Utah

71000

5000

When only passing a dict, the index in the resulting Series will have the dict’s keys in sorted order.

Cuando solo se pasa un dict, el índice de la Serie resultante tendrá las claves del dict ordenadas.

In [23]: states = ['California', 'Ohio', 'Oregon', 'Texas']

In [24]: obj4 = Series(sdata, index=states)

In [25]: obj4

Out[25]:

California

NaN

Ohio

35000

Oregon

16000

Texas

71000

In this case, 3 values found in sdata were placed in the appropriate locations, but since no value for 'California' was found, it appears as NaN (not a number) which is considered in pandas to mark missing or NA values. I will use the terms “missing” or “NA” to refer to missing data. The isnull and notnull functions in pandas should be used to detect missing data:

En este caso, 3 valores encontrados en sdata se colocaron en las ubicaciones apropiadas, pero como no se encontró ningún valor para 'California', aparece como NaN (no un número), que se considera en pandas para marcar valores faltantes o NA. Usaré los términos “faltante” o “NA” para referirme a los datos faltantes. Las funciones isnull y notnull en pandas deben usarse para detectar datos faltantes:

In [26]: pd.isnull(obj4)

Out[26]:

California

True

Ohio

False

Oregon

False

Texas

False

In [27]: pd.notnull(obj4)

Out[27]:

California

False

Ohio

True

Oregon

True

Texas

True

Series also has these as instance methods:

La serie también tiene estos como métodos de instancia:

In [28]: obj4.isnull()

Out[28]:

California

True

Ohio

False

Oregon

False

Texas

False

I discuss working with missing data in more detail later in this chapter.

Más adelante en este capítulo analizo el trabajo con datos faltantes con más detalle.

A critical Series feature for many applications is that it automatically aligns differently-indexed data in arithmetic operations:

Una característica crítica de la Serie para muchas aplicaciones es que alinea automáticamente datos indexados de manera diferente en operaciones aritméticas:

In [29]: obj3

Out[29]:

Ohio

35000

Oregon

16000

Texas

71000

Utah

5000

In [30]: obj4

Out[30]:

California

NaN

Ohio

35000

Oregon

16000

Texas

71000

In [31]: obj3 + obj4

Out[31]:

California

NaN

Ohio

70000

Oregon

32000

114 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoTexas

Utah

142000

NaN

Data alignment features are addressed as a separate topic.

Las características de alineación de datos se tratan como un tema aparte.

Both the Series object itself and its index have a name attribute, which integrates with other key areas of pandas functionality:

Tanto el objeto Serie como su índice tienen un atributo de nombre, que se integra con otras áreas clave de la funcionalidad de pandas:

In [32]: obj4.name = 'population'

In [33]: obj4.index.name = 'state'

In [34]: obj4

Out[34]:

state

California

NaN

Ohio

35000

Oregon

16000

Texas

71000

Name: population

A Series’s index can be altered in place by assignment:

Un índice de serie se puede modificar mediante asignación:

In [35]: obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']

In [36]: obj

Out[36]:

Bob

4

Steve

7

Jeff

-5

Ryan

3

**DataFrame**

A DataFrame represents a tabular, spreadsheet-like data structure containing an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.). The DataFrame has both a row and column index; it can be thought of as a dict of Series (one for all sharing the same index). Compared with other such DataFrame-like structures you may have used before (like R’s data.frame), row-oriented and column-oriented operations in DataFrame are treated roughly symmetrically. Under the hood, the data is stored as one or more two-dimensional blocks rather than a list, dict, or some other collection of one-dimensional arrays. The exact details of DataFrame’s internals are far outside the scope of this book.

Un DataFrame representa una estructura de datos tabular similar a una hoja de cálculo que contiene una colección ordenada de columnas, cada una de las cuales puede tener un tipo de valor diferente (numérico, cadena, booleano, etc.). El DataFrame tiene un índice de fila y columna; Se puede considerar como un dictado de Serie (uno para todos los que comparten el mismo índice). En comparación con otras estructuras similares a DataFrame que haya usado antes (como el data.frame de R), las operaciones orientadas a filas y columnas en DataFrame se tratan de forma aproximadamente simétrica. En el fondo, los datos se almacenan como uno o más bloques bidimensionales en lugar de una lista, dictado o alguna otra colección de matrices unidimensionales. Los detalles exactos de los componentes internos de DataFrame están fuera del alcance de este libro.

While DataFrame stores the data internally in a two-dimensional format, you can easily represent much higher-dimensional data in a tabular format using hierarchical indexing, a subject of a later section and a key ingredient in many of the more advanced data-handling features in pandas.

Si bien DataFrame almacena los datos internamente en un formato bidimensional, puede representar fácilmente datos de dimensiones mucho más altas en un formato tabular utilizando indexación jerárquica, un tema de una sección posterior y un ingrediente clave en muchas de las funciones más avanzadas de manejo de datos. en pandas.

There are numerous ways to construct a DataFrame, though one of the most common is from a dict of equal-length lists or NumPy arrays

Existen numerosas formas de construir un DataFrame, aunque una de las más comunes es a partir de un diccionario de listas de igual longitud o matrices NumPy.

data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],

'year': [2000, 2001, 2002, 2001, 2002],

'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}

frame = DataFrame(data)

The resulting DataFrame will have its index assigned automatically as with Series, and the columns are placed in sorted order:

Al DataFrame resultante se le asignará su índice automáticamente como con la Serie, y las columnas se colocarán en orden:

In [38]: frame

Out[38]:

pop

state

0 1.5

Ohio

1 1.7

Ohio

2 3.6

Ohio

3 2.4 Nevada

4 2.9 Nevada

year

2000

2001

2002

2001

2002

If you specify a sequence of columns, the DataFrame’s columns will be exactly what you pass:

Si especifica una secuencia de columnas, las columnas del DataFrame serán exactamente lo que pase:

In [39]: DataFrame(data, columns=['year', 'state', 'pop'])

Out[39]:

year

state pop

0 2000

Ohio 1.5

1 2001

Ohio 1.7

2 2002

Ohio 3.6

3 2001 Nevada 2.4

4 2002 Nevada 2.9

As with Series, if you pass a column that isn’t contained in data, it will appear with NA values in the result:

Al igual que con Serie, si pasa una columna que no está contenida en datos, aparecerá con valores NA en el resultado:

In [40]: frame2 = DataFrame(data, columns=['year', 'state', 'pop', 'debt'],

....:

index=['one', 'two', 'three', 'four', 'five'])

In [41]: frame2

Out[41]:

year

state

one

2000

Ohio

two

2001

Ohio

three 2002

Ohio

four

2001 Nevada

five

2002 Nevada

pop

1.5

1.7

3.6

2.4

2.9

debt

NaN

NaN

NaN

NaN

NaN

In [42]: frame2.columns

Out[42]: Index([year, state, pop, debt], dtype=object)

A column in a DataFrame can be retrieved as a Series either by dict-like notation or by attribute:

Una columna en un DataFrame se puede recuperar como una Serie ya sea mediante notación tipo dict o por atributo:

In [43]: frame2['state']

Out[43]:

one

Ohio

In [44]: frame2.year

Out[44]:

one

2000

116 | Chapter 5: Getting Started with pandas

www.it-ebooks.infotwo

Ohio

three

Ohio

four

Nevada

five

Nevada

Name: state

two

2001

three

2002

four

2001

five

2002

Name: year

Note that the returned Series have the same index as the DataFrame, and their name attribute has been appropriately set.

Tenga en cuenta que la Serie devuelta tiene el mismo índice que el DataFrame y su atributo de nombre se ha configurado correctamente.

Rows can also be retrieved by position or name by a couple of methods, such as the ix indexing field (much more on this later):

Las filas también se pueden recuperar por posición o nombre mediante un par de métodos, como el campo de indexación ix (mucho más sobre esto más adelante):

In [45]: frame2.ix['three']

Out[45]:

year

2002

state

Ohio

pop

3.6

debt

NaN

Name: three

Columns can be modified by assignment. For example, the empty 'debt' column could be assigned a scalar value or an array of values:

Las columnas se pueden modificar mediante asignación. Por ejemplo, a la columna vacía 'deuda' se le podría asignar un valor escalar o una matriz de valores:

In [46]: frame2['debt'] = 16.5

In [47]: frame2

Out[47]:

year

state

one

2000

Ohio

two

2001

Ohio

three 2002

Ohio

four

2001 Nevada

five

2002 Nevada

pop

1.5

1.7

3.6

2.4

2.9

debt

16.5

16.5

16.5

16.5

16.5

In [48]: frame2['debt'] = np.arange(5.)

In [49]: frame2

Out[49]:

year

state

one

2000

Ohio

two

2001

Ohio

three 2002

Ohio

four

2001 Nevada

five

2002 Nevada

pop

1.5

1.7

3.6

2.4

2.9

debt

0

1

2

3

4

When assigning lists or arrays to a column, the value’s length must match the length of the DataFrame. If you assign a Series, it will be instead conformed exactly to the DataFrame’s index, inserting missing values in any holes:

Al asignar listas o matrices a una columna, la longitud del valor debe coincidir con la longitud del DataFrame. Si asigna una Serie, se ajustará exactamente al índice del DataFrame, insertando los valores faltantes en los agujeros:

In [50]: val = Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])

In [51]: frame2['debt'] = val

In [52]: frame2

Out[52]:

year

state

pop

debt

Introduction to pandas Data Structures | 117

www.it-ebooks.infoone

two

three

four

five

2000

2001

2002

2001

2002

Ohio

Ohio

Ohio

Nevada

Nevada

1.5

1.7

3.6

2.4

2.9

NaN

-1.2

NaN

-1.5

-1.7

Assigning a column that doesn’t exist will create a new column. The del keyword will

delete columns as with a dict:

Asignar una columna que no existe creará una nueva columna. La palabra clave del eliminará columnas como con un dict:

In [53]: frame2['eastern'] = frame2.state == 'Ohio'

In [54]: frame2

Out[54]:

year

state

one

2000

Ohio

two

2001

Ohio

three 2002

Ohio

four

2001 Nevada

five

2002 Nevada

pop

1.5

1.7

3.6

2.4

2.9

debt eastern

NaN

True

-1.2

True

NaN

True

-1.5

False

-1.7

False

In [55]: del frame2['eastern']

In [56]: frame2.columns

Out[56]: Index([year, state, pop, debt], dtype=object)

The column returned when indexing a DataFrame is a view on the underlying data, not a copy. Thus, any in-place modifications to the Series will be reflected in the DataFrame. The column can be explicitly copied using the Series’s copy method.

La columna devuelta al indexar un DataFrame es una vista de los datos subyacentes, no una copia. Por lo tanto, cualquier modificación implementada en la Serie se reflejará en el DataFrame. La columna se puede copiar explícitamente utilizando el método de copia de la Serie.

Another common form of data is a nested dict of dicts format:

Otra forma común de datos es un formato de dictado de dictados anidado:

In [57]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},

....:

'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}

If passed to DataFrame, it will interpret the outer dict keys as the columns and the inner keys as the row indices:

Si se pasa a DataFrame, interpretará las claves de dictado externas como las columnas y las claves internas como los índices de las filas:

In [58]: frame3 = DataFrame(pop)

In [59]: frame3

Out[59]:

Nevada Ohio

2000

NaN

1.5

2001

2.4

1.7

2002

2.9

3.6

Of course you can always transpose the result:

Por supuesto, siempre puedes transponer el resultado:

In [60]: frame3.T

Out[60]:

2000 2001

Nevada

NaN

2.4

Ohio

1.5

1.7

2002

2.9

3.6

118 | Chapter 5: Getting Started with pandas

[www.it-ebooks.info](http://www.it-ebooks.info)

The keys in the inner dicts are unioned and sorted to form the index in the result. This isn’t true if an explicit index is specified:

Las claves en los dictados internos se unen y ordenan para formar el índice en el resultado. Esto no es cierto si se especifica un índice explícito:

In [61]: DataFrame(pop, index=[2001, 2002, 2003])

Out[61]:

Nevada Ohio

2001

2.4

1.7

2002

2.9

3.6

2003

NaN

NaN

Dicts of Series are treated much in the same way:

Los dictados de series se tratan de la misma manera:

In [62]: pdata = {'Ohio': frame3['Ohio'][:-1],

....:

'Nevada': frame3['Nevada'][:2]}

In [63]: DataFrame(pdata)

Out[63]:

Nevada Ohio

2000

NaN

1.5

2001

2.4

1.7

For a complete list of things you can pass the DataFrame constructor, see Table 5-1.

Para obtener una lista completa de las cosas que puede pasar al constructor DataFrame, consulte la Tabla 5-1.

If a DataFrame’s index and columns have their name attributes set, these will also be displayed:

Si el índice y las columnas de un DataFrame tienen sus atributos de nombre establecidos, estos también se mostrarán:

In [64]: frame3.index.name = 'year'; frame3.columns.name = 'state'

In [65]: frame3

Out[65]:

state Nevada Ohio

year

2000

NaN

1.5

2001

2.4

1.7

2002

2.9

3.6

Like Series, the values attribute returns the data contained in the DataFrame as a 2D ndarray:

Al igual que Series, el atributo de valores devuelve los datos contenidos en el DataFrame como un ndarray 2D:

In [66]: frame3.values

Out[66]:

array([[ nan, 1.5],

[ 2.4, 1.7],

[ 2.9, 3.6]])

If the DataFrame’s columns are different dtypes, the dtype of the values array will be chosen to accomodate all of the columns:

Si las columnas del DataFrame son de diferentes tipos, se elegirá el tipo de la matriz de valores para acomodar todas las columnas:

In [67]: frame2.values

Out[67]:

array([[2000, Ohio, 1.5, nan],

[2001, Ohio, 1.7, -1.2],

[2002, Ohio, 3.6, nan],

[2001, Nevada, 2.4, -1.5],

[2002, Nevada, 2.9, -1.7]], dtype=object)

Introduction to pandas Data Structures | 119

[www.it-ebooks.info](http://www.it-ebooks.info)

Table 5-1. Possible data inputs to DataFrame constructor

Type Notes

2D ndarrayA matrix of data, passing optional row and column labels

dict of arrays, lists, or tuples Each sequence becomes a column in the DataFrame. All sequences must be the same length.

NumPy structured/record arrayTreated as the “dict of arrays” case

dict of SeriesEach value becomes a column. Indexes from each Series are unioned together to form the

result’s row index if no explicit index is passed.

dict of dictsEach inner dict becomes a column. Keys are unioned to form the row index as in the “dict of

Series” case.

list of dicts or SeriesEach item becomes a row in the DataFrame. Union of dict keys or Series indexes become the

DataFrame’s column labels

List of lists or tuplesTreated as the “2D ndarray” case

Another DataFrameThe DataFrame’s indexes are used unless different ones are passed

NumPy MaskedArrayLike the “2D ndarray” case except masked values become NA/missing in the DataFrame result

**Index Objects**

pandas’s Index objects are responsible for holding the axis labels and other metadata (like the axis name or names). Any array or other sequence of labels used when constructing a Series or DataFrame is internally converted to an Index:

Los objetos Index de pandas son responsables de contener las etiquetas de los ejes y otros metadatos (como el nombre o los nombres del eje). Cualquier matriz u otra secuencia de etiquetas utilizadas al construir una serie o un marco de datos se convierte internamente en un índice:

In [68]: obj = Series(range(3), index=['a', 'b', 'c'])

In [69]: index = obj.index

In [70]: index

Out[70]: Index([a, b, c], dtype=object)

In [71]: index[1:]

Out[71]: Index([b, c], dtype=object)

Index objects are immutable and thus can’t be modified by the user:

Los objetos de índice son inmutables y, por lo tanto, el usuario no puede modificarlos:

In [72]: index[1] = 'd'

---------------------------------------------------------------------------

Exception

Traceback (most recent call last)

<ipython-input-72-676fdeb26a68> in <module>()

----> 1 index[1] = 'd'

/Users/wesm/code/pandas/pandas/core/index.pyc in \_\_setitem\_\_(self, key, value)

302

def \_\_setitem\_\_(self, key, value):

303

"""Disable the setting of values."""

--> 304

raise Exception(str(self.\_\_class\_\_) + ' object is immutable')

305

306

def \_\_getitem\_\_(self, key):

Exception: <class 'pandas.core.index.Index'> object is immutable

120 | Chapter 5: Getting Started with pandas

[www.it-ebooks.info](http://www.it-ebooks.info)

Immutability is important so that Index objects can be safely shared among data structures:

La inmutabilidad es importante para que los objetos Index se puedan compartir de forma segura entre estructuras de datos:

In [73]: index = pd.Index(np.arange(3))

In [74]: obj2 = Series([1.5, -2.5, 0], index=index)

In [75]: obj2.index is index

Out[75]: True

Table 5-2 has a list of built-in Index classes in the library. With some development effort, Index can even be subclassed to implement specialized axis indexing functionality.

La Tabla 5-2 tiene una lista de clases de índice integradas en la biblioteca. Con algo de esfuerzo de desarrollo, Index puede incluso subclasificarse para implementar una funcionalidad de indexación de ejes especializada.

Many users will not need to know much about Index objects, but they’re nonetheless an important part of pandas’s data model.

Muchos usuarios no necesitarán saber mucho sobre los objetos Index, pero de todos modos son una parte importante del modelo de datos de Pandas.

Table 5-2. Main Index objects in pandas

Class Description

Index The most general Index object, representing axis labels in a NumPy array of Python objects.

Int64IndexSpecialized Index for integer values.

MultiIndex“Hierarchical” index object representing multiple levels of indexing on a single axis. Can be thought of

as similar to an array of tuples.

DatetimeIndexStores nanosecond timestamps (represented using NumPy’s datetime64 dtype).

PeriodIndexSpecialized Index for Period data (timespans).

In addition to being array-like, an Index also functions as a fixed-size set:

Además de ser similar a una matriz, un índice también funciona como un conjunto de tamaño fijo:

In [76]: frame3

Out[76]:

state Nevada Ohio

year

2000

NaN

1.5

2001

2.4

1.7

2002

2.9

3.6

In [77]: 'Ohio' in frame3.columns

Out[77]: True

In [78]: 2003 in frame3.index

Out[78]: False

Each Index has a number of methods and properties for set logic and answering other common questions about the data it contains. These are summarized in Table 5-3.

Cada índice tiene una serie de métodos y propiedades para establecer la lógica y responder otras preguntas comunes sobre los datos que contiene. Estos se resumen en la Tabla 5-3.

Introduction to pandas Data Structures | 121

[www.it-ebooks.info](http://www.it-ebooks.info)

Table 5-3. Index methods and properties

MethodDescription

appendConcatenate with additional Index objects, producing a new Index

diffCompute set difference as an Index

intersectionCompute set intersection

unionCompute set union

isinCompute boolean array indicating whether each value is contained in the passed collection

deleteCompute new Index with element at index i deleted

dropCompute new index by deleting passed values

insertCompute new Index by inserting element at index i

is\_monotonicReturns True if each element is greater than or equal to the previous element

is\_uniqueReturns True if the Index has no duplicate values

uniqueCompute the array of unique values in the Index

**Essential Functionality**

In this section, I’ll walk you through the fundamental mechanics of interacting with the data contained in a Series or DataFrame. Upcoming chapters will delve more deeply into data analysis and manipulation topics using pandas. This book is not intended to serve as exhaustive documentation for the pandas library; I instead focus on the most important features, leaving the less common (that is, more esoteric) things for you to explore on your own.

En esta sección, lo guiaré a través de la mecánica fundamental de interactuar con los datos contenidos en una Serie o DataFrame. Los próximos capítulos profundizarán más en temas de análisis y manipulación de datos utilizando pandas. Este libro no pretende servir como documentación exhaustiva para la biblioteca pandas; En cambio, me concentro en las características más importantes, dejando las cosas menos comunes (es decir, más esotéricas) para que las explores por tu cuenta.

**Reindexing**

A critical method on pandas objects is reindex, which means to create a new object with the data conformed to a new index. Consider a simple example from above:

Un método crítico en los objetos pandas es reindexar, lo que significa crear un nuevo objeto con los datos conformes a un nuevo índice. Considere un ejemplo simple de arriba:

In [79]: obj = Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])

In [80]: obj

Out[80]:

d

4.5

b

7.2

a -5.3

c

3.6

Calling reindex on this Series rearranges the data according to the new index, introducing missing values if any index values were not already present:

Llamar a reindex en esta serie reorganiza los datos de acuerdo con el nuevo índice, introduciendo valores faltantes si algún valor de índice aún no estaba presente:

In [81]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])

In [82]: obj2

Out[82]:

a -5.3

122 | Chapter 5: Getting Started with pandas

www.it-ebooks.infob

c

d

e

7.2

3.6

4.5

NaN

In [83]: obj.reindex(['a', 'b', 'c', 'd', 'e'], fill\_value=0)

Out[83]:

a -5.3

b

7.2

c

3.6

d

4.5

e

0.0

For ordered data like time series, it may be desirable to do some interpolation or filling of values when reindexing. The method option allows us to do this, using a method such as ffill which forward fills the values:

Para datos ordenados, como series temporales, puede ser conveniente realizar alguna interpolación o completar valores al volver a indexar. La opción método nos permite hacer esto, usando un método como relleno que rellena los valores hacia adelante:

In [84]: obj3 = Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])

In [85]: obj3.reindex(range(6), method='ffill')

Out[85]:

0

blue

1

blue

2

purple

3

purple

4

yellow

5

yellow

Table 5-4 lists available method options. At this time, interpolation more sophisticated than forward- and backfilling would need to be applied after the fact.

La Tabla 5-4 enumera las opciones de métodos disponibles. En este momento, sería necesario aplicar una interpolación más sofisticada que el forward-and-backfilling después del hecho.

Table 5-4. reindex method (interpolation) options

ArgumentDescription

ffill or padFill (or carry) values forward

bfill or backfillFill (or carry) values backward

With DataFrame, reindex can alter either the (row) index, columns, or both. When passed just a sequence, the rows are reindexed in the result:

Con DataFrame, la reindexación puede alterar el índice (fila), las columnas o ambos. Cuando se pasa solo una secuencia, las filas se reindexan en el resultado:

In [86]: frame = DataFrame(np.arange(9).reshape((3, 3)), index=['a', 'c', 'd'],

....:

columns=['Ohio', 'Texas', 'California'])

In [87]: frame

Out[87]:

Ohio Texas

a

0

1

c

3

4

d

6

7

California

2

5

8

In [88]: frame2 = frame.reindex(['a', 'b', 'c', 'd'])

In [89]: frame2

Out[89]:

Essential Functionality | 123

www.it-ebooks.infoa

b

c

d

Ohio

0

NaN

3

6

Texas

1

NaN

4

7

California

2

NaN

5

8

The columns can be reindexed using the columns keyword:

Las columnas se pueden reindexar usando la palabra clave columns:

In [90]: states = ['Texas', 'Utah', 'California']

In [91]: frame.reindex(columns=states)

Out[91]:

Texas Utah California

a

1 NaN

2

c

4 NaN

5

d

7 NaN

8

Both can be reindexed in one shot, though interpolation will only apply row-wise (axis 0):

Ambos se pueden reindexar de una sola vez, aunque la interpolación solo se aplicará en filas (eje 0):

In [92]: frame.reindex(index=['a', 'b', 'c', 'd'], method='ffill',

....:

columns=states)

Out[92]:

Texas Utah California

a

1 NaN

2

b

1 NaN

2

c

4 NaN

5

d

7 NaN

8

As you’ll see soon, reindexing can be done more succinctly by label-indexing with ix:

Como verá pronto, la reindexación se puede realizar de manera más sucinta indexando etiquetas con ix:

In [93]: frame.ix[['a', 'b', 'c', 'd'], states]

Out[93]:

Texas Utah California

a

1

NaN

2

b

NaN

NaN

NaN

c

4

NaN

5

d

7

NaN

8

Table 5-5. reindex function arguments

ArgumentDescription

indexNew sequence to use as index. Can be Index instance or any other sequence-like Python data structure. An

Index will be used exactly as is without any copying

methodInterpolation (fill) method, see Table 5-4 for options.

fill\_valueSubstitute value to use when introducing missing data by reindexing

limitWhen forward- or backfilling, maximum size gap to fill

levelMatch simple Index on level of MultiIndex, otherwise select subset of

copyDo not copy underlying data if new index is equivalent to old index. True by default (i.e. always copy data).

124 | Chapter 5: Getting Started with pandas

[www.it-ebooks.info](http://www.it-ebooks.info)

**Dropping entries from an axis**

Dropping one or more entries from an axis is easy if you have an index array or list without those entries. As that can require a bit of munging and set logic, the drop method will return a new object with the indicated value or values deleted from an axis:

Eliminar una o más entradas de un eje es fácil si tiene una matriz o lista de índice sin esas entradas. Como eso puede requerir un poco de lógica de configuración y manipulación, el método drop devolverá un nuevo objeto con el valor o valores indicados eliminados de un eje:

In [94]: obj = Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])

In [95]: new\_obj = obj.drop('c')

In [96]: new\_obj

Out[96]:

a

0

b

1

d

3

e

4

In [97]: obj.drop(['d', 'c'])

Out[97]:

a

0

b

1

e

4

With DataFrame, index values can be deleted from either axis:

Con DataFrame, los valores de índice se pueden eliminar de cualquiera de los ejes:

In [98]: data = DataFrame(np.arange(16).reshape((4, 4)),

....:

index=['Ohio', 'Colorado', 'Utah', 'New York'],

....:

columns=['one', 'two', 'three', 'four'])

In [99]: data.drop(['Colorado', 'Ohio'])

Out[99]:

one two three four

Utah

8

9

10

11

New York

12

13

14

15

In [100]: data.drop('two', axis=1)

Out[100]:

one three four

Ohio

0

2

3

Colorado

4

6

7

Utah

8

10

11

New York

12

14

15

In [101]: data.drop(['two', 'four'], axis=1)

Out[101]:

one three

Ohio

0

2

Colorado

4

6

Utah

8

10

New York

12

14

**Indexing, selection, and filtering**

Series indexing (obj[...]) works analogously to NumPy array indexing, except you can use the Series’s index values instead of only integers. Here are some examples this:

La indexación de series (obj[...]) funciona de manera análoga a la indexación de matrices NumPy, excepto que puede usar los valores de índice de series en lugar de solo números enteros. A continuación se muestran algunos ejemplos de esto:

In [102]: obj = Series(np.arange(4.), index=['a', 'b', 'c', 'd'])

In [103]: obj['b']

Out[103]: 1.0In [104]: obj[1]

Out[104]: 1.0

In [105]: obj[2:4]

Out[105]:In [106]: obj[['b', 'a', 'd']]

Out[106]:

Essential Functionality | 125

www.it-ebooks.infoc

d

2

3

b

a

d

In [107]: obj[[1, 3]]

Out[107]:

b

1

d

3

1

0

3

In [108]: obj[obj < 2]

Out[108]:

a

0

b

1

Slicing with labels behaves differently than normal Python slicing in that the endpoint is inclusive:

El corte con etiquetas se comporta de manera diferente al corte normal de Python en el sentido de que el punto final es inclusivo:

In [109]: obj['b':'c']

Out[109]:

b

1

c

2

Setting using these methods works just as you would expect:

La configuración utilizando estos métodos funciona tal como es de esperar:

In [110]: obj['b':'c'] = 5

In [111]: obj

Out[111]:

a

0

b

5

c

5

d

3

As you’ve seen above, indexing into a DataFrame is for retrieving one or more columns either with a single value or sequence:

In [112]: data = DataFrame(np.arange(16).reshape((4, 4)),

.....:

index=['Ohio', 'Colorado', 'Utah', 'New York'],

.....:

columns=['one', 'two', 'three', 'four'])

In [113]: data

Out[113]:

one two

Ohio

0

1

Colorado

4

5

Utah

8

9

New York

12

13

three

2

6

10

14

In [114]: data['two']

Out[114]:

Ohio

1

Colorado

5

Utah

9

New York

13

Name: two

four

3

7

11

15

In [115]: data[['three', 'one']]

Out[115]:

three one

Ohio

2

0

Colorado

6

4

Utah

10

8

New York

14

12

Indexing like this has a few special cases. First selecting rows by slicing or a boolean

array:

In [116]: data[:2]

Out[116]:

one two

three

four

In [117]: data[data['three'] > 5]

Out[117]:

one two three four

126 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoOhio

Colorado

0

4

1

5

2

6

3

7

Colorado

Utah

New York

4

8

12

5

9

13

6

10

14

7

11

15

This might seem inconsistent to some readers, but this syntax arose out of practicality

and nothing more. Another use case is in indexing with a boolean DataFrame, such as

one produced by a scalar comparison:

In [118]: data < 5

Out[118]:

one

two

Ohio

True

True

Colorado

True False

Utah

False False

New York False False

three

True

False

False

False

four

True

False

False

False

In [119]: data[data < 5] = 0

In [120]: data

Out[120]:

one two

Ohio

0

0

Colorado

0

5

Utah

8

9

New York

12

13

three

0

6

10

14

four

0

7

11

15

This is intended to make DataFrame syntactically more like an ndarray in this case.

For DataFrame label-indexing on the rows, I introduce the special indexing field ix. It

enables you to select a subset of the rows and columns from a DataFrame with NumPy-

like notation plus axis labels. As I mentioned earlier, this is also a less verbose way to

do reindexing:

In [121]: data.ix['Colorado', ['two', 'three']]

Out[121]:

two

5

three

6

Name: Colorado

In [122]: data.ix[['Colorado', 'Utah'], [3, 0, 1]]

Out[122]:

four one two

Colorado

7

0

5

Utah

11

8

9

In [123]: data.ix[2]

Out[123]:

one

8

two

9

three

10

four

11

Name: Utah

In [124]: data.ix[:'Utah', 'two']

Out[124]:

Ohio

0

Colorado

5

Utah

9

Name: two

In [125]: data.ix[data.three > 5, :3]

Out[125]:

Essential Functionality | 127

www.it-ebooks.infoone

0

8

12

Colorado

Utah

New York

two

5

9

13

three

6

10

14

So there are many ways to select and rearrange the data contained in a pandas object.

For DataFrame, there is a short summary of many of them in Table 5-6. You have a

number of additional options when working with hierarchical indexes as you’ll later

see.

When designing pandas, I felt that having to type frame[:, col] to select

a column was too verbose (and error-prone), since column selection is

one of the most common operations. Thus I made the design trade-off

to push all of the rich label-indexing into ix.

Table 5-6. Indexing options with DataFrame

TypeNotes

obj[val]Select single column or sequence of columns from the DataFrame. Special case con-

veniences: boolean array (filter rows), slice (slice rows), or boolean DataFrame (set

values based on some criterion).

obj.ix[val]Selects single row of subset of rows from the DataFrame.

obj.ix[:, val]Selects single column of subset of columns.

obj.ix[val1, val2]Select both rows and columns.

reindex methodConform one or more axes to new indexes.

xs methodSelect single row or column as a Series by label.

icol, irow methodsSelect single column or row, respectively, as a Series by integer location.

get\_value, set\_value methodsSelect single value by row and column label.

Arithmetic and data alignment

One of the most important pandas features is the behavior of arithmetic between ob-

jects with different indexes. When adding together objects, if any index pairs are not

the same, the respective index in the result will be the union of the index pairs. Let’s

look at a simple example:

In [126]: s1 = Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])

In [127]: s2 = Series([-2.1, 3.6, -1.5, 4, 3.1], index=['a', 'c', 'e', 'f', 'g'])

In [128]: s1

Out[128]:

a

7.3

c -2.5

d

3.4

In [129]: s2

Out[129]:

a

-2.1

c

3.6

e

-1.5

128 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoe

1.5

f

g

4.0

3.1

Adding these together yields:

In [130]: s1 + s2

Out[130]:

a

5.2

c

1.1

d

NaN

e

0.0

f

NaN

g

NaN

The internal data alignment introduces NA values in the indices that don’t overlap.

Missing values propagate in arithmetic computations.

In the case of DataFrame, alignment is performed on both the rows and the columns:

In [131]: df1 = DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),

.....:

index=['Ohio', 'Texas', 'Colorado'])

In [132]: df2 = DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),

.....:

index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [133]: df1

Out[133]:

b c

Ohio

0 1

Texas

3 4

Colorado 6 7

In [134]: df2

Out[134]:

b

d

Utah

0

1

Ohio

3

4

Texas

6

7

Oregon 9 10

d

2

5

8

e

2

5

8

11

Adding these together returns a DataFrame whose index and columns are the unions

of the ones in each DataFrame:

In [135]: df1 + df2

Out[135]:

b

c

d

e

Colorado NaN NaN NaN NaN

Ohio

3 NaN

6 NaN

Oregon

NaN NaN NaN NaN

Texas

9 NaN 12 NaN

Utah

NaN NaN NaN NaN

Arithmetic methods with fill values

In arithmetic operations between differently-indexed objects, you might want to fill

with a special value, like 0, when an axis label is found in one object but not the other:

In [136]: df1 = DataFrame(np.arange(12.).reshape((3, 4)), columns=list('abcd'))

In [137]: df2 = DataFrame(np.arange(20.).reshape((4, 5)), columns=list('abcde'))

In [138]: df1

Out[138]:

a b

c

d

In [139]: df2

Out[139]:

a b

c

d

e

Essential Functionality | 129

www.it-ebooks.info0 0

1 4

2 8

1

5

9

2

6

10

3

7

11

0

1

2

3

0 1

5 6

10 11

15 16

2

7

12

17

3

8

13

18

4

9

14

19

Adding these together results in NA values in the locations that don’t overlap:

In [140]: df1 + df2

Out[140]:

a

b

c

d

e

0 0

2

4

6 NaN

1 9 11 13 15 NaN

2 18 20 22 24 NaN

3 NaN NaN NaN NaN NaN

Using the add method on df1, I pass df2 and an argument to fill\_value:

In [141]: df1.add(df2, fill\_value=0)

Out[141]:

a

b

c

d

e

0 0

2

4

6

4

1 9 11 13 15

9

2 18 20 22 24 14

3 15 16 17 18 19

Relatedly, when reindexing a Series or DataFrame, you can also specify a different fill

value:

In [142]: df1.reindex(columns=df2.columns, fill\_value=0)

Out[142]:

a b

c

d e

0 0 1

2

3 0

1 4 5

6

7 0

2 8 9 10 11 0

Table 5-7. Flexible arithmetic methods

MethodDescription

addMethod for addition (+)

subMethod for subtraction (-)

divMethod for division (/)

mulMethod for multiplication (\*)

Operations between DataFrame and Series

As with NumPy arrays, arithmetic between DataFrame and Series is well-defined. First,

as a motivating example, consider the difference between a 2D array and one of its rows:

In [143]: arr = np.arange(12.).reshape((3, 4))

In [144]: arr

Out[144]:

array([[ 0.,

[ 4.,

1.,

5.,

2.,

6.,

3.],

7.],

130 | Chapter 5: Getting Started with pandas

www.it-ebooks.info[

8.,

9.,

10.,

11.]])

In [145]: arr[0]

Out[145]: array([ 0.,1., 2.,

In [146]: arr - arr[0]

Out[146]:

array([[ 0., 0., 0.,

[ 4., 4., 4.,

[ 8., 8., 8.,0.],

4.],

8.]])

3.])

This is referred to as broadcasting and is explained in more detail in Chapter 12. Op-

erations between a DataFrame and a Series are similar:

In [147]: frame = DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),

.....:

index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [148]: series = frame.ix[0]

In [149]: frame

Out[149]:

b

d

e

Utah

0

1

2

Ohio

3

4

5

Texas

6

7

8

Oregon 9 10 11

In [150]: series

Out[150]:

b

0

d

1

e

2

Name: Utah

By default, arithmetic between DataFrame and Series matches the index of the Series

on the DataFrame's columns, broadcasting down the rows:

In [151]: frame - series

Out[151]:

b d e

Utah

0 0 0

Ohio

3 3 3

Texas

6 6 6

Oregon 9 9 9

If an index value is not found in either the DataFrame’s columns or the Series’s index,

the objects will be reindexed to form the union:

In [152]: series2 = Series(range(3), index=['b', 'e', 'f'])

In [153]: frame + series2

Out[153]:

b

d

e

f

Utah

0 NaN

3 NaN

Ohio

3 NaN

6 NaN

Texas

6 NaN

9 NaN

Oregon 9 NaN 12 NaN

If you want to instead broadcast over the columns, matching on the rows, you have to

use one of the arithmetic methods. For example:

In [154]: series3 = frame['d']

In [155]: frame

In [156]: series3

Essential Functionality | 131

www.it-ebooks.infoOut[155]:

b

Utah

0

Ohio

3

Texas

6

Oregon 9

d

1

4

7

10

e

2

5

8

11

Out[156]:

Utah

1

Ohio

4

Texas

7

Oregon

10

Name: d

In [157]: frame.sub(series3, axis=0)

Out[157]:

b d e

Utah

-1 0 1

Ohio

-1 0 1

Texas -1 0 1

Oregon -1 0 1

The axis number that you pass is the axis to match on. In this case we mean to match

on the DataFrame’s row index and broadcast across.

Function application and mapping

NumPy ufuncs (element-wise array methods) work fine with pandas objects:

In [158]: frame = DataFrame(np.random.randn(4, 3), columns=list('bde'),

.....:

index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [159]: frame

Out[159]:

b

Utah

-0.204708

Ohio

-0.555730

Texas

0.092908

Oregon 1.246435

d

e

0.478943 -0.519439

1.965781 1.393406

0.281746 0.769023

1.007189 -1.296221

In [160]: np.abs(frame)

Out[160]:

b

d

Utah

0.204708 0.478943

Ohio

0.555730 1.965781

Texas

0.092908 0.281746

Oregon 1.246435 1.007189

e

0.519439

1.393406

0.769023

1.296221

Another frequent operation is applying a function on 1D arrays to each column or row.

DataFrame’s apply method does exactly this:

In [161]: f = lambda x: x.max() - x.min()

In [162]: frame.apply(f)

Out[162]:

b

1.802165

d

1.684034

e

2.689627

In [163]: frame.apply(f, axis=1)

Out[163]:

Utah

0.998382

Ohio

2.521511

Texas

0.676115

Oregon

2.542656

Many of the most common array statistics (like sum and mean) are DataFrame methods,

so using apply is not necessary.

The function passed to apply need not return a scalar value, it can also return a Series

with multiple values:

In [164]: def f(x):

.....:

return Series([x.min(), x.max()], index=['min', 'max'])

In [165]: frame.apply(f)

132 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoOut[165]:

b

min -0.555730

max 1.246435

d

e

0.281746 -1.296221

1.965781 1.393406

Element-wise Python functions can be used, too. Suppose you wanted to compute a

formatted string from each floating point value in frame. You can do this with applymap:

In [166]: format = lambda x: '%.2f' % x

In [167]: frame.applymap(format)

Out[167]:

b

d

e

Utah

-0.20 0.48 -0.52

Ohio

-0.56 1.97

1.39

Texas

0.09 0.28

0.77

Oregon

1.25 1.01 -1.30

The reason for the name applymap is that Series has a map method for applying an ele-

ment-wise function:

In [168]: frame['e'].map(format)

Out[168]:

Utah

-0.52

Ohio

1.39

Texas

0.77

Oregon

-1.30

Name: e

Sorting and ranking

Sorting a data set by some criterion is another important built-in operation. To sort

lexicographically by row or column index, use the sort\_index method, which returns

a new, sorted object:

In [169]: obj = Series(range(4), index=['d', 'a', 'b', 'c'])

In [170]: obj.sort\_index()

Out[170]:

a

1

b

2

c

3

d

0

With a DataFrame, you can sort by index on either axis:

In [171]: frame = DataFrame(np.arange(8).reshape((2, 4)), index=['three', 'one'],

.....:

columns=['d', 'a', 'b', 'c'])

In [172]: frame.sort\_index()

Out[172]:

d a b c

one

4 5 6 7

three 0 1 2 3

In [173]: frame.sort\_index(axis=1)

Out[173]:

a b c d

three 1 2 3 0

one

5 6 7 4

Essential Functionality | 133

www.it-ebooks.infoThe data is sorted in ascending order by default, but can be sorted in descending order,

too:

In [174]: frame.sort\_index(axis=1, ascending=False)

Out[174]:

d c b a

three 0 3 2 1

one

4 7 6 5

To sort a Series by its values, use its order method:

In [175]: obj = Series([4, 7, -3, 2])

In [176]: obj.order()

Out[176]:

2

-3

3

2

0

4

1

7

Any missing values are sorted to the end of the Series by default:

In [177]: obj = Series([4, np.nan, 7, np.nan, -3, 2])

In [178]: obj.order()

Out[178]:

4

-3

5

2

0

4

2

7

1

NaN

3

NaN

On DataFrame, you may want to sort by the values in one or more columns. To do so,

pass one or more column names to the by option:

In [179]: frame = DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})

In [180]: frame

Out[180]:

a b

0 0 4

1 1 7

2 0 -3

3 1 2

In [181]: frame.sort\_index(by='b')

Out[181]:

a b

2 0 -3

3 1 2

0 0 4

1 1 7

To sort by multiple columns, pass a list of names:

In [182]: frame.sort\_index(by=['a', 'b'])

Out[182]:

a b

2 0 -3

0 0 4

3 1 2

1 1 7

134 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoRanking is closely related to sorting, assigning ranks from one through the number of

valid data points in an array. It is similar to the indirect sort indices produced by

numpy.argsort, except that ties are broken according to a rule. The rank methods for

Series and DataFrame are the place to look; by default rank breaks ties by assigning

each group the mean rank:

In [183]: obj = Series([7, -5, 7, 4, 2, 0, 4])

In [184]: obj.rank()

Out[184]:

0

6.5

1

1.0

2

6.5

3

4.5

4

3.0

5

2.0

6

4.5

Ranks can also be assigned according to the order they’re observed in the data:

In [185]: obj.rank(method='first')

Out[185]:

0

6

1

1

2

7

3

4

4

3

5

2

6

5

Naturally, you can rank in descending order, too:

In [186]: obj.rank(ascending=False, method='max')

Out[186]:

0

2

1

7

2

2

3

4

4

5

5

6

6

4

See Table 5-8 for a list of tie-breaking methods available. DataFrame can compute ranks

over the rows or the columns:

In [187]: frame = DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],

.....:

'c': [-2, 5, 8, -2.5]})

In [188]: frame

Out[188]:

a

b

c

0 0 4.3 -2.0

1 1 7.0 5.0

2 0 -3.0 8.0

3 1 2.0 -2.5

In [189]: frame.rank(axis=1)

Out[189]:

a b c

0 2 3 1

1 1 3 2

2 2 1 3

3 2 3 1

Essential Functionality | 135

www.it-ebooks.infoTable 5-8. Tie-breaking methods with rank

MethodDescription

'average'Default: assign the average rank to each entry in the equal group.

'min'Use the minimum rank for the whole group.

'max'Use the maximum rank for the whole group.

'first'Assign ranks in the order the values appear in the data.

Axis indexes with duplicate values

Up until now all of the examples I’ve showed you have had unique axis labels (index

values). While many pandas functions (like reindex) require that the labels be unique,

it’s not mandatory. Let’s consider a small Series with duplicate indices:

In [190]: obj = Series(range(5), index=['a', 'a', 'b', 'b', 'c'])

In [191]: obj

Out[191]:

a

0

a

1

b

2

b

3

c

4

The index’s is\_unique property can tell you whether its values are unique or not:

In [192]: obj.index.is\_unique

Out[192]: False

Data selection is one of the main things that behaves differently with duplicates. In-

dexing a value with multiple entries returns a Series while single entries return a scalar

value:

In [193]: obj['a']

Out[193]:

a

0

a

1

In [194]: obj['c']

Out[194]: 4

The same logic extends to indexing rows in a DataFrame:

In [195]: df = DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])

In [196]: df

Out[196]:

0

1

2

a 0.274992 0.228913 1.352917

a 0.886429 -2.001637 -0.371843

b 1.669025 -0.438570 -0.539741

b 0.476985 3.248944 -1.021228

In [197]: df.ix['b']

Out[197]:

0

1

2

136 | Chapter 5: Getting Started with pandas

www.it-ebooks.infob 1.669025 -0.438570 -0.539741

b 0.476985 3.248944 -1.021228

Summarizing and Computing Descriptive Statistics

pandas objects are equipped with a set of common mathematical and statistical meth-

ods. Most of these fall into the category of reductions or summary statistics, methods

that extract a single value (like the sum or mean) from a Series or a Series of values from

the rows or columns of a DataFrame. Compared with the equivalent methods of vanilla

NumPy arrays, they are all built from the ground up to exclude missing data. Consider

a small DataFrame:

In [198]: df = DataFrame([[1.4, np.nan], [7.1, -4.5],

.....:

[np.nan, np.nan], [0.75, -1.3]],

.....:

index=['a', 'b', 'c', 'd'],

.....:

columns=['one', 'two'])

In [199]: df

Out[199]:

one two

a 1.40 NaN

b 7.10 -4.5

c NaN NaN

d 0.75 -1.3

Calling DataFrame’s sum method returns a Series containing column sums:

In [200]: df.sum()

Out[200]:

one

9.25

two -5.80

Passing axis=1 sums over the rows instead:

In [201]: df.sum(axis=1)

Out[201]:

a

1.40

b

2.60

c

NaN

d

-0.55

NA values are excluded unless the entire slice (row or column in this case) is NA. This

can be disabled using the skipna option:

In [202]: df.mean(axis=1, skipna=False)

Out[202]:

a

NaN

b

1.300

c

NaN

d -0.275

See Table 5-9 for a list of common options for each reduction method options.

Summarizing and Computing Descriptive Statistics | 137

www.it-ebooks.infoTable 5-9. Options for reduction methods

MethodDescription

axisAxis to reduce over. 0 for DataFrame’s rows and 1 for columns.

skipnaExclude missing values, True by default.

levelReduce grouped by level if the axis is hierarchically-indexed (MultiIndex).

Some methods, like idxmin and idxmax, return indirect statistics like the index value

where the minimum or maximum values are attained:

In [203]: df.idxmax()

Out[203]:

one

b

two

d

Other methods are accumulations:

In [204]: df.cumsum()

Out[204]:

one two

a 1.40 NaN

b 8.50 -4.5

c NaN NaN

d 9.25 -5.8

Another type of method is neither a reduction nor an accumulation. describe is one

such example, producing multiple summary statistics in one shot:

In [205]: df.describe()

Out[205]:

one

two

count 3.000000 2.000000

mean

3.083333 -2.900000

std

3.493685 2.262742

min

0.750000 -4.500000

25%

1.075000 -3.700000

50%

1.400000 -2.900000

75%

4.250000 -2.100000

max

7.100000 -1.300000

On non-numeric data, describe produces alternate summary statistics:

In [206]: obj = Series(['a', 'a', 'b', 'c'] \* 4)

In [207]: obj.describe()

Out[207]:

count

16

unique

3

top

a

freq

8

See Table 5-10 for a full list of summary statistics and related methods.

138 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoTable 5-10. Descriptive and summary statistics

MethodDescription

countNumber of non-NA values

describeCompute set of summary statistics for Series or each DataFrame column

min, maxCompute minimum and maximum values

argmin, argmaxCompute index locations (integers) at which minimum or maximum value obtained, respectively

idxmin, idxmaxCompute index values at which minimum or maximum value obtained, respectively

quantileCompute sample quantile ranging from 0 to 1

sumSum of values

meanMean of values

medianArithmetic median (50% quantile) of values

madMean absolute deviation from mean value

varSample variance of values

stdSample standard deviation of values

skewSample skewness (3rd moment) of values

kurtSample kurtosis (4th moment) of values

cumsumCumulative sum of values

cummin, cummaxCumulative minimum or maximum of values, respectively

cumprodCumulative product of values

diffCompute 1st arithmetic difference (useful for time series)

pct\_changeCompute percent changes

Correlation and Covariance

Some summary statistics, like correlation and covariance, are computed from pairs of

arguments. Let’s consider some DataFrames of stock prices and volumes obtained from

Yahoo! Finance:

import pandas.io.data as web

all\_data = {}

for ticker in ['AAPL', 'IBM', 'MSFT', 'GOOG']:

all\_data[ticker] = web.get\_data\_yahoo(ticker, '1/1/2000', '1/1/2010')

price = DataFrame({tic: data['Adj Close']

for tic, data in all\_data.iteritems()})

volume = DataFrame({tic: data['Volume']

for tic, data in all\_data.iteritems()})

I now compute percent changes of the prices:

In [209]: returns = price.pct\_change()

In [210]: returns.tail()

Summarizing and Computing Descriptive Statistics | 139

www.it-ebooks.infoOut[210]:

AAPL

GOOG

IBM

MSFT

Date

2009-12-24 0.034339 0.011117 0.004420 0.002747

2009-12-28 0.012294 0.007098 0.013282 0.005479

2009-12-29 -0.011861 -0.005571 -0.003474 0.006812

2009-12-30 0.012147 0.005376 0.005468 -0.013532

2009-12-31 -0.004300 -0.004416 -0.012609 -0.015432

The corr method of Series computes the correlation of the overlapping, non-NA,

aligned-by-index values in two Series. Relatedly, cov computes the covariance:

In [211]: returns.MSFT.corr(returns.IBM)

Out[211]: 0.49609291822168838

In [212]: returns.MSFT.cov(returns.IBM)

Out[212]: 0.00021600332437329015

DataFrame’s corr and cov methods, on the other hand, return a full correlation or

covariance matrix as a DataFrame, respectively:

In [213]: returns.corr()

Out[213]:

AAPL

GOOG

AAPL 1.000000 0.470660

GOOG 0.470660 1.000000

IBM 0.410648 0.390692

MSFT 0.424550 0.443334IBM

0.410648

0.390692

1.000000

0.496093MSFT

0.424550

0.443334

0.496093

1.000000

In [214]: returns.cov()

Out[214]:

AAPL

GOOG

AAPL 0.001028 0.000303

GOOG 0.000303 0.000580

IBM 0.000252 0.000142

MSFT 0.000309 0.000205IBM

0.000252

0.000142

0.000367

0.000216MSFT

0.000309

0.000205

0.000216

0.000516

Using DataFrame’s corrwith method, you can compute pairwise correlations between

a DataFrame’s columns or rows with another Series or DataFrame. Passing a Series

returns a Series with the correlation value computed for each column:

In [215]: returns.corrwith(returns.IBM)

Out[215]:

AAPL

0.410648

GOOG

0.390692

IBM

1.000000

MSFT

0.496093

Passing a DataFrame computes the correlations of matching column names. Here I

compute correlations of percent changes with volume:

In [216]: returns.corrwith(volume)

Out[216]:

AAPL

-0.057461

GOOG

0.062644

140 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoIBM

MSFT

-0.007900

-0.014175

Passing axis=1 does things row-wise instead. In all cases, the data points are aligned by

label before computing the correlation.

Unique Values, Value Counts, and Membership

Another class of related methods extracts information about the values contained in a

one-dimensional Series. To illustrate these, consider this example:

In [217]: obj = Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])

The first function is unique, which gives you an array of the unique values in a Series:

In [218]: uniques = obj.unique()

In [219]: uniques

Out[219]: array([c, a, d, b], dtype=object)

The unique values are not necessarily returned in sorted order, but could be sorted after

the fact if needed (uniques.sort()). Relatedly, value\_counts computes a Series con-

taining value frequencies:

In [220]: obj.value\_counts()

Out[220]:

c

3

a

3

b

2

d

1

The Series is sorted by value in descending order as a convenience. value\_counts is also

available as a top-level pandas method that can be used with any array or sequence:

In [221]: pd.value\_counts(obj.values, sort=False)

Out[221]:

a

3

b

2

c

3

d

1

Lastly, isin is responsible for vectorized set membership and can be very useful in

filtering a data set down to a subset of values in a Series or column in a DataFrame:

In [222]: mask = obj.isin(['b', 'c'])

In [223]: mask

Out[223]:

0

True

1

False

2

False

3

False

4

False

5

True

6

True

In [224]: obj[mask]

Out[224]:

0

c

5

b

6

b

7

c

8

c

Summarizing and Computing Descriptive Statistics | 141

www.it-ebooks.info7

8

True

True

See Table 5-11 for a reference on these methods.

Table 5-11. Unique, value counts, and binning methods

MethodDescription

isinCompute boolean array indicating whether each Series value is contained in the passed sequence of values.

uniqueCompute array of unique values in a Series, returned in the order observed.

value\_countsReturn a Series containing unique values as its index and frequencies as its values, ordered count in

descending order.

In some cases, you may want to compute a histogram on multiple related columns in

a DataFrame. Here’s an example:

In [225]: data = DataFrame({'Qu1': [1, 3, 4, 3, 4],

.....:

'Qu2': [2, 3, 1, 2, 3],

.....:

'Qu3': [1, 5, 2, 4, 4]})

In [226]: data

Out[226]:

Qu1 Qu2 Qu3

0

1

2

1

1

3

3

5

2

4

1

2

3

3

2

4

4

4

3

4

Passing pandas.value\_counts to this DataFrame’s apply function gives:

In [227]: result = data.apply(pd.value\_counts).fillna(0)

In [228]: result

Out[228]:

Qu1 Qu2 Qu3

1

1

1

1

2

0

2

1

3

2

2

0

4

2

0

2

5

0

0

1

Handling Missing Data

Missing data is common in most data analysis applications. One of the goals in de-

signing pandas was to make working with missing data as painless as possible. For

example, all of the descriptive statistics on pandas objects exclude missing data as

you’ve seen earlier in the chapter.

142 | Chapter 5: Getting Started with pandas

www.it-ebooks.infopandas uses the floating point value NaN (Not a Number) to represent missing data in

both floating as well as in non-floating point arrays. It is just used as a sentinel that can

be easily detected:

In [229]: string\_data = Series(['aardvark', 'artichoke', np.nan, 'avocado'])

In [230]: string\_data

Out[230]:

0

aardvark

1

artichoke

2

NaN

3

avocado

In [231]: string\_data.isnull()

Out[231]:

0

False

1

False

2

True

3

False

The built-in Python None value is also treated as NA in object arrays:

In [232]: string\_data[0] = None

In [233]: string\_data.isnull()

Out[233]:

0

True

1

False

2

True

3

False

I do not claim that pandas’s NA representation is optimal, but it is simple and reason-

ably consistent. It’s the best solution, with good all-around performance characteristics

and a simple API, that I could concoct in the absence of a true NA data type or bit

pattern in NumPy’s data types. Ongoing development work in NumPy may change this

in the future.

Table 5-12. NA handling methods

ArgumentDescription

dropnaFilter axis labels based on whether values for each label have missing data, with varying thresholds for how much

missing data to tolerate.

fillnaFill in missing data with some value or using an interpolation method such as 'ffill' or 'bfill'.

isnullReturn like-type object containing boolean values indicating which values are missing / NA.

notnullNegation of isnull.

Filtering Out Missing Data

You have a number of options for filtering out missing data. While doing it by hand is

always an option, dropna can be very helpful. On a Series, it returns the Series with only

the non-null data and index values:

In [234]: from numpy import nan as NA

In [235]: data = Series([1, NA, 3.5, NA, 7])

In [236]: data.dropna()

Out[236]:

Handling Missing Data | 143

www.it-ebooks.info0

2

4

1.0

3.5

7.0

Naturally, you could have computed this yourself by boolean indexing:

In [237]: data[data.notnull()]

Out[237]:

0

1.0

2

3.5

4

7.0

With DataFrame objects, these are a bit more complex. You may want to drop rows

or columns which are all NA or just those containing any NAs. dropna by default drops

any row containing a missing value:

In [238]: data = DataFrame([[1., 6.5, 3.], [1., NA, NA],

.....:

[NA, NA, NA], [NA, 6.5, 3.]])

In [239]: cleaned = data.dropna()

In [240]: data

Out[240]:

0

1

2

0 1 6.5

3

1 1 NaN NaN

2 NaN NaN NaN

3 NaN 6.5

3

In [241]: cleaned

Out[241]:

0

1 2

0 1 6.5 3

Passing how='all' will only drop rows that are all NA:

In [242]: data.dropna(how='all')

Out[242]:

0

1

2

0 1 6.5

3

1 1 NaN NaN

3 NaN 6.5

3

Dropping columns in the same way is only a matter of passing axis=1:

In [243]: data[4] = NA

In [244]: data

Out[244]:

0

1

2

4

0 1 6.5

3 NaN

1 1 NaN NaN NaN

2 NaN NaN NaN NaN

3 NaN 6.5

3 NaN

In [245]: data.dropna(axis=1, how='all')

Out[245]:

0

1

2

0 1 6.5

3

1 1 NaN NaN

2 NaN NaN NaN

3 NaN 6.5

3

A related way to filter out DataFrame rows tends to concern time series data. Suppose

you want to keep only rows containing a certain number of observations. You can

indicate this with the thresh argument:

In [246]: df = DataFrame(np.random.randn(7, 3))

In [247]: df.ix[:4, 1] = NA; df.ix[:2, 2] = NA

144 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoIn [248]: df

Out[248]:

0

1

2

0 -0.577087

NaN

NaN

1 0.523772

NaN

NaN

2 -0.713544

NaN

NaN

3 -1.860761

NaN 0.560145

4 -1.265934

NaN -1.063512

5 0.332883 -2.359419 -0.199543

6 -1.541996 -0.970736 -1.307030

In [249]: df.dropna(thresh=3)

Out[249]:

0

1

2

5 0.332883 -2.359419 -0.199543

6 -1.541996 -0.970736 -1.307030

Filling in Missing Data

Rather than filtering out missing data (and potentially discarding other data along with

it), you may want to fill in the “holes” in any number of ways. For most purposes, the

fillna method is the workhorse function to use. Calling fillna with a constant replaces

missing values with that value:

In [250]: df.fillna(0)

Out[250]:

0

1

2

0 -0.577087 0.000000 0.000000

1 0.523772 0.000000 0.000000

2 -0.713544 0.000000 0.000000

3 -1.860761 0.000000 0.560145

4 -1.265934 0.000000 -1.063512

5 0.332883 -2.359419 -0.199543

6 -1.541996 -0.970736 -1.307030

Calling fillna with a dict you can use a different fill value for each column:

In [251]: df.fillna({1: 0.5, 3: -1})

Out[251]:

0

1

2

0 -0.577087 0.500000

NaN

1 0.523772 0.500000

NaN

2 -0.713544 0.500000

NaN

3 -1.860761 0.500000 0.560145

4 -1.265934 0.500000 -1.063512

5 0.332883 -2.359419 -0.199543

6 -1.541996 -0.970736 -1.307030

fillna returns a new object, but you can modify the existing object in place:

# always returns a reference to the filled object

In [252]: \_ = df.fillna(0, inplace=True)

In [253]: df

Out[253]:

0

1

0 -0.577087 0.000000

1 0.523772 0.000000

2 -0.713544 0.000000

3 -1.860761 0.000000

2

0.000000

0.000000

0.000000

0.560145

Handling Missing Data | 145

www.it-ebooks.info4 -1.265934 0.000000 -1.063512

5 0.332883 -2.359419 -0.199543

6 -1.541996 -0.970736 -1.307030

The same interpolation methods available for reindexing can be used with fillna:

In [254]: df = DataFrame(np.random.randn(6, 3))

In [255]: df.ix[2:, 1] = NA; df.ix[4:, 2] = NA

In [256]: df

Out[256]:

0

1

2

0 0.286350 0.377984 -0.753887

1 0.331286 1.349742 0.069877

2 0.246674

NaN 1.004812

3 1.327195

NaN -1.549106

4 0.022185

NaN

NaN

5 0.862580

NaN

NaN

In [257]: df.fillna(method='ffill')

Out[257]:

0

1

2

0 0.286350 0.377984 -0.753887

1 0.331286 1.349742 0.069877

2 0.246674 1.349742 1.004812

3 1.327195 1.349742 -1.549106

4 0.022185 1.349742 -1.549106

5 0.862580 1.349742 -1.549106

In [258]: df.fillna(method='ffill', limit=2)

Out[258]:

0

1

2

0 0.286350 0.377984 -0.753887

1 0.331286 1.349742 0.069877

2 0.246674 1.349742 1.004812

3 1.327195 1.349742 -1.549106

4 0.022185

NaN -1.549106

5 0.862580

NaN -1.549106

With fillna you can do lots of other things with a little creativity. For example, you

might pass the mean or median value of a Series:

In [259]: data = Series([1., NA, 3.5, NA, 7])

In [260]: data.fillna(data.mean())

Out[260]:

0

1.000000

1

3.833333

2

3.500000

3

3.833333

4

7.000000

See Table 5-13 for a reference on fillna.

Table 5-13. fillna function arguments

ArgumentDescription

valueScalar value or dict-like object to use to fill missing values

methodInterpolation, by default 'ffill' if function called with no other arguments

axisAxis to fill on, default axis=0

inplaceModify the calling object without producing a copy

limitFor forward and backward filling, maximum number of consecutive periods to fill

146 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoHierarchical Indexing

Hierarchical indexing is an important feature of pandas enabling you to have multiple

(two or more) index levels on an axis. Somewhat abstractly, it provides a way for you

to work with higher dimensional data in a lower dimensional form. Let’s start with a

simple example; create a Series with a list of lists or arrays as the index:

In [261]: data = Series(np.random.randn(10),

.....:

index=[['a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'd', 'd'],

.....:

[1, 2, 3, 1, 2, 3, 1, 2, 2, 3]])

In [262]: data

Out[262]:

a 1

0.670216

2

0.852965

3

-0.955869

b 1

-0.023493

2

-2.304234

3

-0.652469

c 1

-1.218302

2

-1.332610

d 2

1.074623

3

0.723642

What you’re seeing is a prettified view of a Series with a MultiIndex as its index. The

“gaps” in the index display mean “use the label directly above”:

In [263]: data.index

Out[263]:

MultiIndex

[('a', 1) ('a', 2) ('a', 3) ('b', 1) ('b', 2) ('b', 3) ('c', 1)

('c', 2) ('d', 2) ('d', 3)]

With a hierarchically-indexed object, so-called partial indexing is possible, enabling

you to concisely select subsets of the data:

In [264]: data['b']

Out[264]:

1 -0.023493

2 -2.304234

3 -0.652469

In [265]: data['b':'c']

Out[265]:

b 1

-0.023493

2

-2.304234

3

-0.652469

c 1

-1.218302

2

-1.332610

In [266]: data.ix[['b', 'd']]

Out[266]:

b 1

-0.023493

2

-2.304234

3

-0.652469

d 2

1.074623

3

0.723642

Selection is even possible in some cases from an “inner” level:

In [267]: data[:, 2]

Out[267]:

a

0.852965

Hierarchical Indexing | 147

www.it-ebooks.infob

c

d

-2.304234

-1.332610

1.074623

Hierarchical indexing plays a critical role in reshaping data and group-based operations

like forming a pivot table. For example, this data could be rearranged into a DataFrame

using its unstack method:

In [268]: data.unstack()

Out[268]:

1

2

3

a 0.670216 0.852965 -0.955869

b -0.023493 -2.304234 -0.652469

c -1.218302 -1.332610

NaN

d

NaN 1.074623 0.723642

The inverse operation of unstack is stack:

In [269]: data.unstack().stack()

Out[269]:

a 1

0.670216

2

0.852965

3

-0.955869

b 1

-0.023493

2

-2.304234

3

-0.652469

c 1

-1.218302

2

-1.332610

d 2

1.074623

3

0.723642

stack and unstack will be explored in more detail in Chapter 7.

With a DataFrame, either axis can have a hierarchical index:

In [270]: frame = DataFrame(np.arange(12).reshape((4, 3)),

.....:

index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],

.....:

columns=[['Ohio', 'Ohio', 'Colorado'],

.....:

['Green', 'Red', 'Green']])

In [271]: frame

Out[271]:

Ohio

Green Red

a 1

0

1

2

3

4

b 1

6

7

2

9

10

Colorado

Green

2

5

8

11

The hierarchical levels can have names (as strings or any Python objects). If so, these

will show up in the console output (don’t confuse the index names with the axis labels!):

In [272]: frame.index.names = ['key1', 'key2']

In [273]: frame.columns.names = ['state', 'color']

In [274]: frame

148 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoOut[274]:

state

color

key1 key2

a

1

2

b

1

2

Ohio

GreenRedColorado

Green

0

3

6

91

4

7

102

5

8

11

With partial column indexing you can similarly select groups of columns:

In [275]: frame['Ohio']

Out[275]:

color

Green Red

key1 key2

a

1

0

1

2

3

4

b

1

6

7

2

9

10

A MultiIndex can be created by itself and then reused; the columns in the above Data-

Frame with level names could be created like this:

MultiIndex.from\_arrays([['Ohio', 'Ohio', 'Colorado'], ['Green', 'Red', 'Green']],

names=['state', 'color'])

Reordering and Sorting Levels

At times you will need to rearrange the order of the levels on an axis or sort the data

by the values in one specific level. The swaplevel takes two level numbers or names and

returns a new object with the levels interchanged (but the data is otherwise unaltered):

In [276]: frame.swaplevel('key1', 'key2')

Out[276]:

state

Ohio

Colorado

color

Green Red

Green

key2 key1

1

a

0

1

2

2

a

3

4

5

1

b

6

7

8

2

b

9

10

11

sortlevel, on the other hand, sorts the data (stably) using only the values in a single

level. When swapping levels, it’s not uncommon to also use sortlevel so that the result

is lexicographically sorted:

In [277]: frame.sortlevel(1)

Out[277]:

state

Ohio

Colorado

color

Green Red

Green

key1 key2

a

1

0

1

2

b

1

6

7

8

a

2

3

4

5

b

2

9

10

11

In [278]: frame.swaplevel(0, 1).sortlevel(0)

Out[278]:

state

Ohio

Colorado

color

Green Red

Green

key2 key1

1

a

0

1

2

b

6

7

8

2

a

3

4

5

b

9

10

11

Hierarchical Indexing | 149

www.it-ebooks.infoData selection performance is much better on hierarchically indexed

objects if the index is lexicographically sorted starting with the outer-

most level, that is, the result of calling sortlevel(0) or sort\_index().

Summary Statistics by Level

Many descriptive and summary statistics on DataFrame and Series have a level option

in which you can specify the level you want to sum by on a particular axis. Consider

the above DataFrame; we can sum by level on either the rows or columns like so:

In [279]: frame.sum(level='key2')

Out[279]:

state

Ohio

Colorado

color Green Red

Green

key2

1

6

8

10

2

12

14

16

In [280]: frame.sum(level='color', axis=1)

Out[280]:

color

Green Red

key1 key2

a

1

2

1

2

8

4

b

1

14

7

2

20

10

Under the hood, this utilizes pandas’s groupby machinery which will be discussed in

more detail later in the book.

Using a DataFrame’s Columns

It’s not unusual to want to use one or more columns from a DataFrame as the row

index; alternatively, you may wish to move the row index into the DataFrame’s col-

umns. Here’s an example DataFrame:

In [281]: frame = DataFrame({'a': range(7), 'b': range(7, 0, -1),

.....:

'c': ['one', 'one', 'one', 'two', 'two', 'two', 'two'],

.....:

'd': [0, 1, 2, 0, 1, 2, 3]})

In [282]: frame

Out[282]:

a b

c d

0 0 7 one 0

1 1 6 one 1

2 2 5 one 2

3 3 4 two 0

4 4 3 two 1

5 5 2 two 2

6 6 1 two 3

150 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoDataFrame’s set\_index function will create a new DataFrame using one or more of its

columns as the index:

In [283]: frame2 = frame.set\_index(['c', 'd'])

In [284]: frame2

Out[284]:

a b

c d

one 0 0 7

1 1 6

2 2 5

two 0 3 4

1 4 3

2 5 2

3 6 1

By default the columns are removed from the DataFrame, though you can leave them in:

In [285]: frame.set\_index(['c', 'd'], drop=False)

Out[285]:

a b

c d

c d

one 0 0 7 one 0

1 1 6 one 1

2 2 5 one 2

two 0 3 4 two 0

1 4 3 two 1

2 5 2 two 2

3 6 1 two 3

reset\_index, on the other hand, does the opposite of set\_index; the hierarchical index

levels are are moved into the columns:

In [286]: frame2.reset\_index()

Out[286]:

c d a b

0 one 0 0 7

1 one 1 1 6

2 one 2 2 5

3 two 0 3 4

4 two 1 4 3

5 two 2 5 2

6 two 3 6 1

Other pandas Topics

Here are some additional topics that may be of use to you in your data travels.

Integer Indexing

Working with pandas objects indexed by integers is something that often trips up new

users due to some differences with indexing semantics on built-in Python data

Other pandas Topics | 151

www.it-ebooks.infostructures like lists and tuples. For example, you would not expect the following code

to generate an error:

ser = Series(np.arange(3.))

ser[-1]

In this case, pandas could “fall back” on integer indexing, but there’s not a safe and

general way (that I know of) to do this without introducing subtle bugs. Here we have

an index containing 0, 1, 2, but inferring what the user wants (label-based indexing or

position-based) is difficult::

In [288]: ser

Out[288]:

0

0

1

1

2

2

On the other hand, with a non-integer index, there is no potential for ambiguity:

In [289]: ser2 = Series(np.arange(3.), index=['a', 'b', 'c'])

In [290]: ser2[-1]

Out[290]: 2.0

To keep things consistent, if you have an axis index containing indexers, data selection

with integers will always be label-oriented. This includes slicing with ix, too:

In [291]: ser.ix[:1]

Out[291]:

0

0

1

1

In cases where you need reliable position-based indexing regardless of the index type,

you can use the iget\_value method from Series and irow and icol methods from Da-

taFrame:

In [292]: ser3 = Series(range(3), index=[-5, 1, 3])

In [293]: ser3.iget\_value(2)

Out[293]: 2

In [294]: frame = DataFrame(np.arange(6).reshape(3, 2)), index=[2, 0, 1])

In [295]: frame.irow(0)

Out[295]:

0

0

1

1

Name: 2

Panel Data

While not a major topic of this book, pandas has a Panel data structure, which you can

think of as a three-dimensional analogue of DataFrame. Much of the development focus

of pandas has been in tabular data manipulations as these are easier to reason about,

152 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoand hierarchical indexing makes using truly N-dimensional arrays unnecessary in a lot

of cases.

To create a Panel, you can use a dict of DataFrame objects or a three-dimensional

ndarray:

import pandas.io.data as web

pdata = pd.Panel(dict((stk, web.get\_data\_yahoo(stk, '1/1/2009', '6/1/2012'))

for stk in ['AAPL', 'GOOG', 'MSFT', 'DELL']))

Each item (the analogue of columns in a DataFrame) in the Panel is a DataFrame:

In [297]: pdata

Out[297]:

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

Dimensions: 4 (items) x 861 (major) x 6 (minor)

Items: AAPL to MSFT

Major axis: 2009-01-02 00:00:00 to 2012-06-01 00:00:00

Minor axis: Open to Adj Close

In [298]: pdata = pdata.swapaxes('items', 'minor')

In [299]: pdata['Adj Close']

Out[299]:

<class 'pandas.core.frame.DataFrame'>

DatetimeIndex: 861 entries, 2009-01-02 00:00:00 to 2012-06-01 00:00:00

Data columns:

AAPL

861 non-null values

DELL

861 non-null values

GOOG

861 non-null values

MSFT

861 non-null values

dtypes: float64(4)

ix-based label indexing generalizes to three dimensions, so we can select all data at a

particular date or a range of dates like so:

In [300]: pdata.ix[:, '6/1/2012', :]

Out[300]:

Open

High

Low Close

AAPL 569.16 572.65 560.52 560.99

DELL

12.15

12.30

12.05 12.07

GOOG 571.79 572.65 568.35 570.98

MSFT

28.76

28.96

28.44 28.45

Volume

18606700

19396700

3057900

56634300

Adj Close

560.99

12.07

570.98

28.45

In [301]: pdata.ix['Adj Close', '5/22/2012':, :]

Out[301]:

AAPL

DELL

GOOG

MSFT

Date

2012-05-22 556.97 15.08 600.80 29.76

2012-05-23 570.56 12.49 609.46 29.11

2012-05-24 565.32 12.45 603.66 29.07

2012-05-25 562.29 12.46 591.53 29.06

2012-05-29 572.27 12.66 594.34 29.56

2012-05-30 579.17 12.56 588.23 29.34

Other pandas Topics | 153

www.it-ebooks.info2012-05-31

2012-06-01

577.73

560.99

12.33

12.07

580.86

570.98

29.19

28.45

An alternate way to represent panel data, especially for fitting statistical models, is in

“stacked” DataFrame form:

In [302]: stacked = pdata.ix[:, '5/30/2012':, :].to\_frame()

In [303]: stacked

Out[303]:

major

minor

2012-05-30 AAPL

DELL

GOOG

MSFT

2012-05-31 AAPL

DELL

GOOG

MSFT

2012-06-01 AAPL

DELL

GOOG

MSFT

OpenHighLowCloseVolumeAdj Close

569.20

12.59

588.16

29.35

580.74

12.53

588.72

29.30

569.16

12.15

571.79

28.76579.99

12.70

591.90

29.48

581.50

12.54

590.00

29.42

572.65

12.30

572.65

28.96566.56

12.46

583.53

29.12

571.46

12.33

579.00

28.94

560.52

12.05

568.35

28.44579.17

12.56

588.23

29.34

577.73

12.33

580.86

29.19

560.99

12.07

570.98

28.4518908200

19787800

1906700

41585500

17559800

19955500

2968300

39134000

18606700

19396700

3057900

56634300579.17

12.56

588.23

29.34

577.73

12.33

580.86

29.19

560.99

12.07

570.98

28.45

DataFrame has a related to\_panel method, the inverse of to\_frame:

In [304]: stacked.to\_panel()

Out[304]:

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

Dimensions: 6 (items) x 3 (major) x 4 (minor)

Items: Open to Adj Close

Major axis: 2012-05-30 00:00:00 to 2012-06-01 00:00:00

Minor axis: AAPL to MSFT

154 | Chapter 5: Getting Started with pandas

www.it-ebooks.infoCHAPTER 6

Data Loading, Storag