



EBook Gratis

# APRENDIZAJE pandas

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

#pandas

# Tabla de contenido

Acerca de.....	1
<b>Capítulo 1: Empezando con los pandas.....</b>	<b>2</b>
Observaciones.....	2
Versiones.....	2
Examples.....	3
Instalación o configuración.....	3
Instalar via anaconda.....	5
Hola Mundo.....	5
Estadísticas descriptivas.....	6
<b>Capítulo 2: Agrupar datos de series de tiempo.....</b>	<b>8</b>
Examples.....	8
Generar series de tiempo de números aleatorios y luego abajo muestra.....	8
<b>Capítulo 3: Análisis: Reunirlo todo y tomar decisiones.....</b>	<b>10</b>
Examples.....	10
Análisis quintil: con datos aleatorios.....	10
Que es un factor.....	10
<b>Inicialización.....</b>	<b>10</b>
<b>pd.qcut - Crea cubos quintiles.....</b>	<b>11</b>
<b>Análisis.....</b>	<b>11</b>
Devoluciones de parcela.....	11
Visualizar la correlación del scatter_matrix con scatter_matrix.....	12
Calcula y visualiza Máximo Draw Down.....	13
Calcular estadísticas.....	15
<b>Capítulo 4: Anexando a DataFrame.....</b>	<b>17</b>
Examples.....	17
Anexando una nueva fila a DataFrame.....	17
Añadir un DataFrame a otro DataFrame.....	18
<b>Capítulo 5: Calendarios de vacaciones.....</b>	<b>20</b>
Examples.....	20
Crear un calendario personalizado.....	20

Usa un calendario personalizado.....	20
<b>Consigue las vacaciones entre dos fechas.....</b>	<b>20</b>
<b>Cuenta el número de días laborables entre dos fechas.....</b>	<b>21</b>
<b>Capítulo 6: Creando marcos de datos.....</b>	<b>22</b>
Introducción.....	22
Examples.....	22
Crear un DataFrame de muestra.....	22
Crea un DataFrame de muestra usando Numpy.....	23
Cree un DataFrame de muestra a partir de múltiples colecciones usando el Diccionario.....	24
Crear un DataFrame a partir de una lista de tuplas.....	24
Crear un DataFrame de un diccionario de listas.....	25
Crear un DataFrame de muestra con datetime.....	25
Crear un DataFrame de muestra con MultiIndex.....	27
Guardar y cargar un DataFrame en formato pickle (.plk).....	28
Crear un DataFrame a partir de una lista de diccionarios.....	28
<b>Capítulo 7: Datos categóricos.....</b>	<b>29</b>
Introducción.....	29
Examples.....	29
Creación de objetos.....	29
Creando grandes conjuntos de datos al azar.....	29
<b>Capítulo 8: Datos de agrupación.....</b>	<b>31</b>
Examples.....	31
Agrupacion basica.....	31
Agrupar por una columna.....	31
Agrupar por columnas múltiples.....	31
Números de agrupación.....	32
Columna de selección de un grupo.....	33
Agregando por tamaño versus por cuenta.....	34
Agregando grupos.....	34
Exportar grupos en diferentes archivos.....	35
usar la transformación para obtener estadísticas a nivel de grupo mientras se preserva el .....	35
<b>Capítulo 9: Datos duplicados.....</b>	<b>37</b>

Examples.....	37
Seleccione duplicado.....	37
Drop duplicado.....	37
Contando y consiguiendo elementos únicos.....	38
Obtener valores únicos de una columna.....	39
<b>Capítulo 10: Datos perdidos.....</b>	<b>41</b>
Observaciones.....	41
Examples.....	41
Relleno de valores perdidos.....	41
Rellene los valores faltantes con un solo valor:.....	41
Rellene los valores faltantes con los anteriores:.....	41
Rellena con los siguientes:.....	41
Rellene utilizando otro DataFrame:.....	42
Bajando valores perdidos.....	42
Eliminar filas si al menos una columna tiene un valor perdido.....	42
Eliminar filas si faltan todos los valores de esa fila.....	43
Eliminar columnas que no tengan al menos 3 valores no perdidos.....	43
Interpolación.....	43
Comprobación de valores perdidos.....	43
<b>Capítulo 11: Desplazamiento y desplazamiento de datos.....</b>	<b>45</b>
Examples.....	45
Desplazar o retrasar valores en un marco de datos.....	45
<b>Capítulo 12: Fusionar, unir y concatenar.....</b>	<b>46</b>
Sintaxis.....	46
Parámetros.....	46
Examples.....	47
Unir.....	47
Fusionando dos DataFrames.....	48
<b>Unir internamente:.....</b>	<b>48</b>
<b>Unión externa:.....</b>	<b>49</b>
<b>Unirse a la izquierda:.....</b>	<b>49</b>

<b>Unirse a la derecha</b>	<b>49</b>
Fusionar / concatenar / unir múltiples marcos de datos (horizontal y verticalmente)	50
Fusionar, Unir y Concat	51
¿Cuál es la diferencia entre unirse y fusionarse?	51
<b>Capítulo 13: Gotchas de pandas</b>	<b>54</b>
Observaciones	54
Examples	54
Detectando valores perdidos con np.nan	54
Integer y NA	54
Alineación automática de datos (comportamiento indexado)	55
<b>Capítulo 14: Gráficos y visualizaciones</b>	<b>56</b>
Examples	56
Gráficos de datos básicos	56
Estilo de la trama	58
Parcela en un eje de matplotlib existente	58
<b>Capítulo 15: Guardar pandas dataframe en un archivo csv</b>	<b>59</b>
Parámetros	59
Examples	60
Crear un marco de datos aleatorio y escribir en .csv	60
Guarde Pandas DataFrame de la lista a los dictados a CSV sin índice y con codificación de	62
<b>Capítulo 16: Herramientas computacionales</b>	<b>63</b>
Examples	63
Encuentra la correlación entre columnas	63
<b>Capítulo 17: Herramientas de Pandas IO (leer y guardar conjuntos de datos)</b>	<b>64</b>
Observaciones	64
Examples	64
Leyendo el archivo csv en DataFrame	64
Expediente:	64
Código:	64
Salida:	64
Algunos argumentos útiles:	64

Guardado básico en un archivo csv .....	66
Fechas de análisis al leer de CSV .....	66
Hoja de cálculo para dictado de DataFrames .....	66
Lee una hoja específica .....	66
Prueba de read_csv .....	66
Lista de comprensión .....	67
Leer en trozos .....	68
Guardar en archivo CSV .....	68
Análisis de columnas de fecha con read_csv .....	69
Lea y combine varios archivos CSV (con la misma estructura) en un DF .....	69
Leyendo el archivo cvs en un marco de datos pandas cuando no hay una fila de encabezado .....	69
Usando HDFStore .....	70
<b>Generar muestra DF con diversos tipos .....</b>	<b>70</b>
<b>hacer un DF más grande (10 * 100.000 = 1.000.000 filas) .....</b>	<b>71</b>
<b>crear (o abrir un archivo HDFStore existente) .....</b>	<b>71</b>
<b>guarde nuestro marco de datos en el archivo h5 (HDFStore), indexando [int32, int64, string] .....</b>	<b>71</b>
<b>Mostrar detalles de HDFStore .....</b>	<b>71</b>
<b>mostrar columnas indexadas .....</b>	<b>71</b>
<b>cerrar (vaciar al disco) nuestro archivo de tienda .....</b>	<b>72</b>
Lea el registro de acceso de Nginx (varias cotillas) .....	72
<b>Capítulo 18: Indexación booleana de marcos de datos .....</b>	<b>73</b>
Introducción .....	73
Examples .....	73
Accediendo a un DataFrame con un índice booleano .....	73
Aplicar una máscara booleana a un marco de datos .....	74
Datos de enmascaramiento basados en el valor de la columna .....	74
Datos de enmascaramiento basados en el valor del índice .....	75
<b>Capítulo 19: Indexación y selección de datos .....</b>	<b>76</b>
Examples .....	76
Seleccionar columna por etiqueta .....	76
Seleccionar por posición .....	76
Rebanar con etiquetas .....	77

Posición mixta y selección basada en etiqueta .....	78
Indexación booleana .....	79
Filtrado de columnas (selección de "interesante", eliminación innecesaria, uso de RegEx, e.....	80
<b>generar muestra DF .....</b>	<b>80</b>
<b>mostrar columnas que contengan la letra 'a' .....</b>	<b>80</b>
<b>muestre las columnas usando el filtro RegEx (b c d) - b o c o d : .....</b>	<b>80</b>
<b>mostrar todas las columnas excepto los que empiezan por a (en otras palabras remove / deja ..</b>	<b>81</b>
Filtrar / seleccionar filas usando el método `.query ()` .....	81
generar DF aleatorio .....	81
seleccione las filas donde los valores en la columna A > 2 y los valores en la columna B < .....	81
utilizando el método .query() con variables para filtrar .....	82
Rebanado Dependiente del Camino .....	82
Obtener las primeras / últimas n filas de un marco de datos .....	84
Seleccionar filas distintas en el marco de datos .....	85
Filtrar las filas con datos faltantes (NaN, Ninguno, NaT) .....	86
<b>Capítulo 20: IO para Google BigQuery .....</b>	<b>88</b>
Examples .....	88
Lectura de datos de BigQuery con credenciales de cuenta de usuario .....	88
Lectura de datos de BigQuery con credenciales de cuenta de servicio .....	89
<b>Capítulo 21: JSON .....</b>	<b>90</b>
Examples .....	90
Leer json .....	90
<b>puede pasar la cadena del json o una ruta de archivo a un archivo con json válido .....</b>	<b>90</b>
Marco de datos en JSON anidado como en los archivos flare.js utilizados en D3.js .....	90
Lee JSON del archivo .....	91
<b>Capítulo 22: Leer MySQL a DataFrame .....</b>	<b>92</b>
Examples .....	92
Usando sqlalchemy y PyMySQL .....	92
Para leer mysql a dataframe, en caso de gran cantidad de datos .....	92
<b>Capítulo 23: Leer SQL Server a Dataframe .....</b>	<b>93</b>
Examples .....	93

Utilizando pyodbc.....	93
Usando pyodbc con bucle de conexión.....	93
<b>Capítulo 24: Leyendo archivos en pandas DataFrame.....</b>	<b>95</b>
Examples.....	95
Leer la tabla en DataFrame.....	95
Archivo de tabla con encabezado, pie de página, nombres de fila y columna de índice:.....	95
Archivo de tabla sin nombres de fila o índice:.....	95
Leer archivo CSV.....	96
Datos con encabezado, separados por punto y coma en lugar de comas.....	96
Tabla sin nombres de filas o índice y comas como separadores.....	96
Recopila datos de la hoja de cálculo de Google en el marco de datos de pandas.....	97
<b>Capítulo 25: Making Pandas Play Nice con tipos de datos nativos de Python.....</b>	<b>98</b>
Examples.....	98
Mover datos de pandas a estructuras nativas Python y Numpy.....	98
<b>Capítulo 26: Manipulación de cuerdas.....</b>	<b>100</b>
Examples.....	100
Expresiones regulares.....	100
Rebanar cuerdas.....	100
Comprobando el contenido de una cadena.....	102
Capitalización de cuerdas.....	102
<b>Capítulo 27: Manipulación sencilla de DataFrames.....</b>	<b>105</b>
Examples.....	105
Eliminar una columna en un DataFrame.....	105
Renombrar una columna.....	106
Añadiendo una nueva columna.....	107
Asignar directamente.....	107
Añadir una columna constante.....	107
Columna como expresión en otras columnas.....	107
Crealo sobre la marcha.....	108
agregar columnas múltiples.....	108
añadir múltiples columnas sobre la marcha.....	108
Localice y reemplace los datos en una columna.....	109



Añadiendo una nueva fila a DataFrame.....	109
Eliminar / eliminar filas de DataFrame.....	110
Reordenar columnas.....	111
<b>Capítulo 28: Meta: Pautas de documentación.....</b>	<b>112</b>
Observaciones.....	112
Examples.....	112
Mostrando fragmentos de código y salida.....	112
estilo.....	113
Compatibilidad con la versión pandas.....	113
imprimir declaraciones.....	113
Prefiero el apoyo de python 2 y 3:.....	113
<b>Capítulo 29: Multiindex.....</b>	<b>114</b>
Examples.....	114
Seleccione de MultiIndex por Nivel.....	114
Iterar sobre DataFrame con MultiIndex.....	115
Configuración y clasificación de un MultiIndex.....	116
Cómo cambiar columnas MultiIndex a columnas estándar.....	118
Cómo cambiar columnas estándar a MultiIndex.....	118
Columnas multiindex.....	119
Visualización de todos los elementos en el índice.....	119
<b>Capítulo 30: Obteniendo información sobre DataFrames.....</b>	<b>120</b>
Examples.....	120
Obtener información de DataFrame y el uso de la memoria.....	120
Lista de nombres de columna de DataFrame.....	120
Las diversas estadísticas de resumen de Dataframe.....	121
<b>Capítulo 31: Pandas Datareader.....</b>	<b>122</b>
Observaciones.....	122
Examples.....	122
Ejemplo básico de Datareader (Yahoo Finance).....	122
Lectura de datos financieros (para múltiples tickers) en el panel de pandas - demostración.....	123
<b>Capítulo 32: pd.DataFrame.apply.....</b>	<b>125</b>
Examples.....	125

pandas.DataFrame.apply Uso Básico.....	125
<b>Capítulo 33: Remodelación y pivotamiento.....</b>	<b>127</b>
Examples.....	127
Simple pivotante.....	127
Pivotando con la agregación.....	128
Apilamiento y desapilamiento.....	131
Tabulación cruzada.....	132
Las pandas se derriten para ir de lo ancho a lo largo.....	134
Dividir (remodelar) cadenas CSV en columnas en varias filas, con un elemento por fila.....	135
<b>Capítulo 34: Remuestreo.....</b>	<b>137</b>
Examples.....	137
Downsampling y upmpling.....	137
<b>Capítulo 35: Secciones transversales de diferentes ejes con MultiIndex.....</b>	<b>139</b>
Examples.....	139
Selección de secciones utilizando .xs.....	139
Usando .loc y slicers.....	140
<b>Capítulo 36: Serie.....</b>	<b>142</b>
Examples.....	142
Ejemplos de creación de series simples.....	142
Series con fecha y hora.....	142
Algunos consejos rápidos sobre Series in Pandas.....	143
Aplicando una función a una serie.....	145
<b>Capítulo 37: Tipos de datos.....</b>	<b>147</b>
Observaciones.....	147
Examples.....	148
Comprobando los tipos de columnas.....	148
Cambiando dtypes.....	148
Cambiando el tipo a numérico.....	149
Cambiando el tipo a datetime.....	150
Cambiando el tipo a timedelta.....	150
Seleccionando columnas basadas en dtype.....	150
Resumiendo dtypes.....	151

<b>Capítulo 38: Trabajando con series de tiempo</b>	<b>152</b>
Examples	152
Creación de series de tiempo	152
Indización parcial de cuerdas	152
<b>Obteniendo datos</b>	<b>152</b>
<b>Subconjunto</b>	<b>152</b>
<b>Capítulo 39: Tratar variables categóricas</b>	<b>154</b>
Examples	154
Codificación instantánea con <code>`get_dummies ()`</code>	154
<b>Capítulo 40: Uso de <code>.ix</code>, <code>.iloc</code>, <code>.loc</code>, <code>.at</code> y <code>.iat</code> para acceder a un DataFrame</b>	<b>155</b>
Examples	155
Utilizando <code>.iloc</code>	155
Utilizando <code>.loc</code>	156
<b>Capítulo 41: Valores del mapa</b>	<b>158</b>
Observaciones	158
Examples	158
Mapa del Diccionario	158
<b>Creditos</b>	<b>159</b>

---

## Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [pandas](#)

It is an unofficial and free pandas ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official pandas.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

---

# Capítulo 1: Empezando con los pandas

## Observaciones

Pandas es un paquete de Python que proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para hacer que el trabajo con datos "relacionales" o "etiquetados" sea fácil e intuitivo. Pretende ser el elemento fundamental de alto nivel para realizar análisis de datos prácticos y del mundo real en Python.

La documentación oficial de Pandas [se puede encontrar aquí](#) .

## Versiones

Pandas

Versión	Fecha de lanzamiento
0.19.1	2016-11-03
0.19.0	2016-10-02
0.18.1	2016-05-03
0.18.0	2016-03-13
0.17.1	2015-11-21
0.17.0	2015-10-09
0.16.2	2015-06-12
0.16.1	2015-05-11
0.16.0	2015-03-22
0.15.2	2014-12-12
0.15.1	2014-11-09
0.15.0	2014-10-18
0.14.1	2014-07-11
0.14.0	2014-05-31
0.13.1	2014-02-03
0.13.0	2014-01-03

Versión	Fecha de lanzamiento
0.12.0	2013-07-23

## Examples

### Instalación o configuración

Las instrucciones detalladas para configurar o instalar pandas se pueden encontrar [aquí en la documentación oficial](#).

#### Instalando pandas con anaconda

Instalar pandas y el resto de la pila [NumPy](#) y [SciPy](#) puede ser un poco difícil para los usuarios inexpertos.

La forma más sencilla de instalar no solo pandas, sino Python y los paquetes más populares que forman la pila SciPy (IPython, NumPy, Matplotlib, ...) es con [Anaconda](#), una multiplataforma (Linux, Mac OS X, Windows) Distribución en Python para análisis de datos y computación científica.

Después de ejecutar un instalador simple, el usuario tendrá acceso a los pandas y al resto de la pila SciPy sin necesidad de instalar nada más, y sin tener que esperar a que se compile ningún software.

Las instrucciones de instalación de Anaconda [se pueden encontrar aquí](#).

Una lista completa de los paquetes disponibles como parte de la distribución de Anaconda [se puede encontrar aquí](#).

Una ventaja adicional de la instalación con Anaconda es que no requiere derechos de administrador para instalarlo, se instalará en el directorio de inicio del usuario, y esto también hace que sea trivial eliminar Anaconda en una fecha posterior (solo elimine esa carpeta).

#### Instalando pandas con miniconda

La sección anterior describía cómo instalar pandas como parte de la distribución de Anaconda. Sin embargo, este enfoque significa que instalará más de cien paquetes e implica descargar el instalador, que tiene un tamaño de unos pocos cientos de megabytes.

Si desea tener más control sobre qué paquetes, o tiene un ancho de banda de Internet limitado, entonces instalar pandas con [Miniconda](#) puede ser una mejor solución.

[Conda](#) es el gestor de paquetes sobre el que se basa la distribución de Anaconda. Es un gestor de paquetes que es multiplataforma y es independiente del lenguaje (puede desempeñar un papel similar al de una combinación pip y virtualenv).

[Miniconda](#) le permite crear una instalación de Python mínima e independiente, y luego usar el comando [Conda](#) para instalar paquetes adicionales.

Primero, necesitará que se instale Conda, la descarga y la ejecución de Miniconda lo harán por usted. El instalador [se puede encontrar aquí](#) .

El siguiente paso es crear un nuevo entorno conda (estos son análogos a un virtualenv pero también le permiten especificar con precisión qué versión de Python se instalará también). Ejecuta los siguientes comandos desde una ventana de terminal:

```
conda create -n name_of_my_env python
```

Esto creará un entorno mínimo con solo Python instalado en él. Para ponerte dentro de este entorno corre:

```
source activate name_of_my_env
```

En Windows el comando es:

```
activate name_of_my_env
```

El paso final requerido es instalar pandas. Esto se puede hacer con el siguiente comando:

```
conda install pandas
```

Para instalar una versión específica de pandas:

```
conda install pandas=0.13.1
```

Para instalar otros paquetes, IPython por ejemplo:

```
conda install ipython
```

Para instalar la distribución completa de Anaconda:

```
conda install anaconda
```

Si necesita paquetes disponibles para pip pero no conda, simplemente instale pip y use pip para instalar estos paquetes:

```
conda install pip  
pip install django
```

Por lo general, instalaría pandas con uno de los administradores de paquetes.

### **ejemplo de pip**

```
pip install pandas
```

Esto probablemente requerirá la instalación de una serie de dependencias, incluyendo NumPy,

requerirá un compilador para compilar los bits de código requeridos, y puede tardar unos minutos en completarse.

## Instalar via anaconda

Primera [descarga de anaconda](#) desde el sitio de Continuum. Ya sea a través del instalador gráfico (Windows / OSX) o ejecutando un script de shell (OSX / Linux). Esto incluye pandas!

---

Si no desea que los 150 paquetes [estén](#) convenientemente agrupados en anaconda, puede instalar [miniconda](#) . Ya sea a través del instalador gráfico (Windows) o shell script (OSX / Linux).

Instala pandas en miniconda usando:

```
conda install pandas
```

---

Para actualizar pandas a la última versión en anaconda o miniconda use:

```
conda update pandas
```

## Hola Mundo

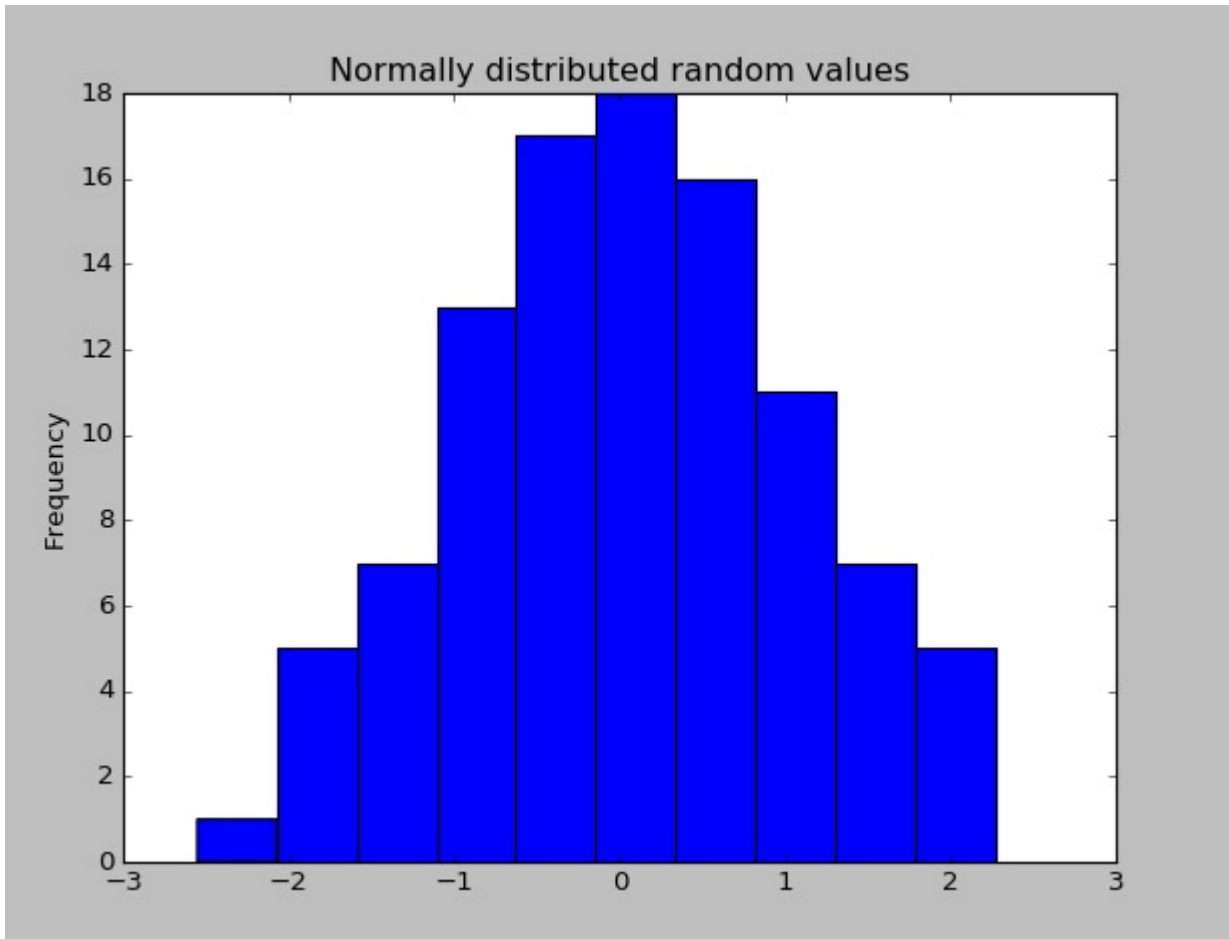
Una vez que se haya instalado Pandas, puede verificar si está funcionando correctamente creando un conjunto de datos de valores distribuidos aleatoriamente y trazando su histograma.

```
import pandas as pd # This is always assumed but is included here as an introduction.
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)

values = np.random.randn(100) # array of normally distributed random numbers
s = pd.Series(values) # generate a pandas series
s.plot(kind='hist', title='Normally distributed random values') # hist computes distribution
plt.show()
```





Compruebe algunas de las estadísticas de los datos (media, desviación estándar, etc.)

```
s.describe()
# Output: count      100.000000
# mean          0.059808
# std           1.012960
# min          -2.552990
# 25%          -0.643857
# 50%           0.094096
# 75%           0.737077
# max           2.269755
# dtype: float64
```

## Estadísticas descriptivas

Las estadísticas descriptivas (media, desviación estándar, número de observaciones, mínimo, máximo y cuartiles) de las columnas numéricas se pueden calcular utilizando el método `.describe()` , que devuelve un marco de datos de pandas de estadísticas descriptivas.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 1, 4, 3, 5, 2, 3, 4, 1],
                           'B': [12, 14, 11, 16, 18, 18, 22, 13, 21, 17],
                           'C': ['a', 'a', 'b', 'a', 'b', 'c', 'b', 'a', 'b', 'a']})

In [2]: df
Out[2]:
   A  B  C
0  1 12  a
```

```
1  2  14  a
2  1  11  b
3  4  16  a
4  3  18  b
5  5  18  c
6  2  22  b
7  3  13  a
8  4  21  b
9  1  17  a
```

```
In [3]: df.describe()
```

```
Out[3]:
```

	A	B
count	10.000000	10.000000
mean	2.600000	16.200000
std	1.429841	3.705851
min	1.000000	11.000000
25%	1.250000	13.250000
50%	2.500000	16.500000
75%	3.750000	18.000000
max	5.000000	22.000000

Tenga en cuenta que dado que `c` no es una columna numérica, se excluye de la salida.

```
In [4]: df['C'].describe()
```

```
Out[4]:
```

```
count      10
unique       3
freq        5
Name: C, dtype: object
```

En este caso, el método resume los datos categóricos por número de observaciones, número de elementos únicos, modo y frecuencia del modo.

Lea Empezando con los pandas en línea: <https://riptutorial.com/es/pandas/topic/796/empezando-con-los-pandas>

# Capítulo 2: Agrupar datos de series de tiempo

## Examples

### Generar series de tiempo de números aleatorios y luego abajo muestra

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# I want 7 days of 24 hours with 60 minutes each
periods = 7 * 24 * 60
tidx = pd.date_range('2016-07-01', periods=periods, freq='T')
#           ^           ^
#           |           |
#           Start Date   Frequency Code for Minute
# This should get me 7 Days worth of minutes in a datetimeindex

# Generate random data with numpy. We'll seed the random
# number generator so that others can see the same results.
# Otherwise, you don't have to seed it.
np.random.seed([3,1415])

# This will pick a number of normally distributed random numbers
# where the number is specified by periods
data = np.random.randn(periods)

ts = pd.Series(data=data, index=tidx, name='HelloTimeSeries')

ts.describe()

count      10080.000000
mean        -0.008853
std         0.995411
min         -3.936794
25%         -0.683442
50%          0.002640
75%          0.654986
max          3.906053
Name: HelloTimeSeries, dtype: float64
```

Tomemos estos 7 días de datos por minuto y tomamos muestras cada 15 minutos. Todos los códigos de frecuencia se pueden encontrar [aquí](#) .

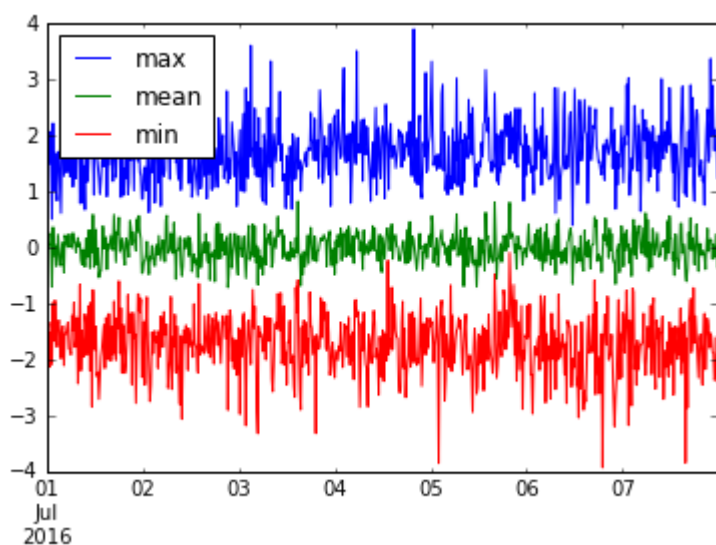
```
# resample says to group by every 15 minutes. But now we need
# to specify what to do within those 15 minute chunks.

# We could take the last value.
ts.resample('15T').last()
```

O cualquier otra cosa que podamos hacer a un `groupby` objeto, [documentación](#) .

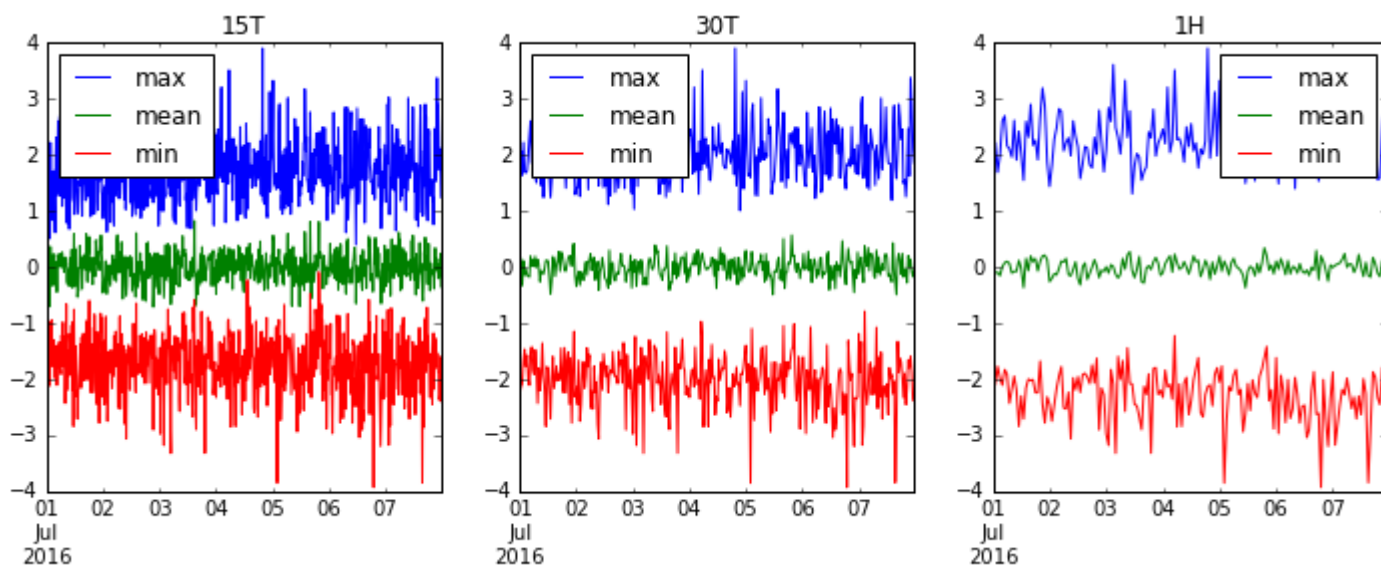
Incluso podemos agregar varias cosas útiles. Vamos a trazar el `min`, el `mean` y el `max` de estos datos de `resample('15M')`.

```
ts.resample('15T').agg(['min', 'mean', 'max']).plot()
```



Volvamos a muestrear sobre `'15T'` (15 minutos), `'30T'` (media hora) y `'1H'` (1 hora) para ver cómo nuestros datos se vuelven más suaves.

```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))
for i, freq in enumerate(['15T', '30T', '1H']):
    ts.resample(freq).agg(['max', 'mean', 'min']).plot(ax=axes[i], title=freq)
```



Lea Agrupar datos de series de tiempo en línea:

<https://riptutorial.com/es/pandas/topic/4747/agrupar-datos-de-series-de-tiempo>

---

# Capítulo 3: Análisis: Reunirlo todo y tomar decisiones.

## Examples

### Análisis quintil: con datos aleatorios

El análisis quintil es un marco común para evaluar la eficacia de los factores de seguridad.

#### Que es un factor

Un factor es un método para calificar / clasificar conjuntos de valores. Para un punto particular en el tiempo y para un conjunto particular de valores, un factor puede representarse como una serie de pandas donde el índice es una matriz de los identificadores de seguridad y los valores son las puntuaciones o rangos.

Si tomamos las puntuaciones de los factores a lo largo del tiempo, podemos, en cada momento, dividir el conjunto de valores en 5 grupos o quintiles iguales, según el orden de las puntuaciones de los factores. No hay nada particularmente sagrado en el número 5. Podríamos haber usado 3 o 10. Pero usamos 5 a menudo. Finalmente, hacemos un seguimiento del rendimiento de cada uno de los cinco grupos para determinar si hay una diferencia significativa en las devoluciones. Tendemos a enfocarnos más intensamente en la diferencia en los rendimientos del grupo con el rango más alto en relación con el rango más bajo.

---

Comencemos estableciendo algunos parámetros y generando datos aleatorios.

Para facilitar la experimentación con los mecanismos, proporcionamos un código simple para crear datos aleatorios que nos dan una idea de cómo funciona esto.

#### Incluye datos aleatorios

- **Devoluciones** : generar devoluciones aleatorias para un número específico de valores y periodos.
- **Señales** : genere señales aleatorias para un número específico de valores y períodos y con el nivel prescrito de correlación con las **devoluciones** . Para que un factor sea útil, debe haber alguna información o correlación entre las puntuaciones / rangos y los rendimientos posteriores. Si no hubiera correlación, lo veríamos. Ese sería un buen ejercicio para el lector, duplique este análisis con datos aleatorios generados con 0 correlaciones.

---

## Inicialización

```
import pandas as pd
import numpy as np
```

```

num_securities = 1000
num_periods = 1000
period_frequency = 'W'
start_date = '2000-12-31'

np.random.seed([3,1415])

means = [0, 0]
covariance = [[ 1., 5e-3],
               [5e-3, 1.]]

# generates to sets of data m[0] and m[1] with ~0.005 correlation
m = np.random.multivariate_normal(means, covariance,
                                   (num_periods, num_securities)).T

```

Ahora generemos un índice de series de tiempo y un índice que represente los identificadores de seguridad. Luego úselos para crear marcos de datos para devoluciones y señales

```

ids = pd.Index(['s{:05d}'.format(s) for s in range(num_securities)], 'ID')
tidx = pd.date_range(start=start_date, periods=num_periods, freq=period_frequency)

```

Divido `m[0]` por 25 para reducir a algo que se parece a los rendimientos de las acciones. También agrego `1e-7` para dar un rendimiento promedio positivo modesto.

```

security_returns = pd.DataFrame(m[0] / 25 + 1e-7, tidx, ids)
security_signals = pd.DataFrame(m[1], tidx, ids)

```

## pd.qcut - Crea cubos quintiles

`pd.qcut` para dividir mis señales en `pd.qcut` de quintiles para cada período.

```

def qcut(s, q=5):
    labels = ['q{}'.format(i) for i in range(1, 6)]
    return pd.qcut(s, q, labels=labels)

cut = security_signals.stack().groupby(level=0).apply(qcut)

```

Utilice estos recortes como un índice en nuestras devoluciones

```

returns_cut = security_returns.stack().rename('returns') \
    .to_frame().set_index(cut, append=True) \
    .swaplevel(2, 1).sort_index().squeeze() \
    .groupby(level=[0, 1]).mean().unstack()

```

## Análisis

### Devoluciones de parcela

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(15, 5))
ax1 = plt.subplot2grid((1,3), (0,0))
ax2 = plt.subplot2grid((1,3), (0,1))
ax3 = plt.subplot2grid((1,3), (0,2))

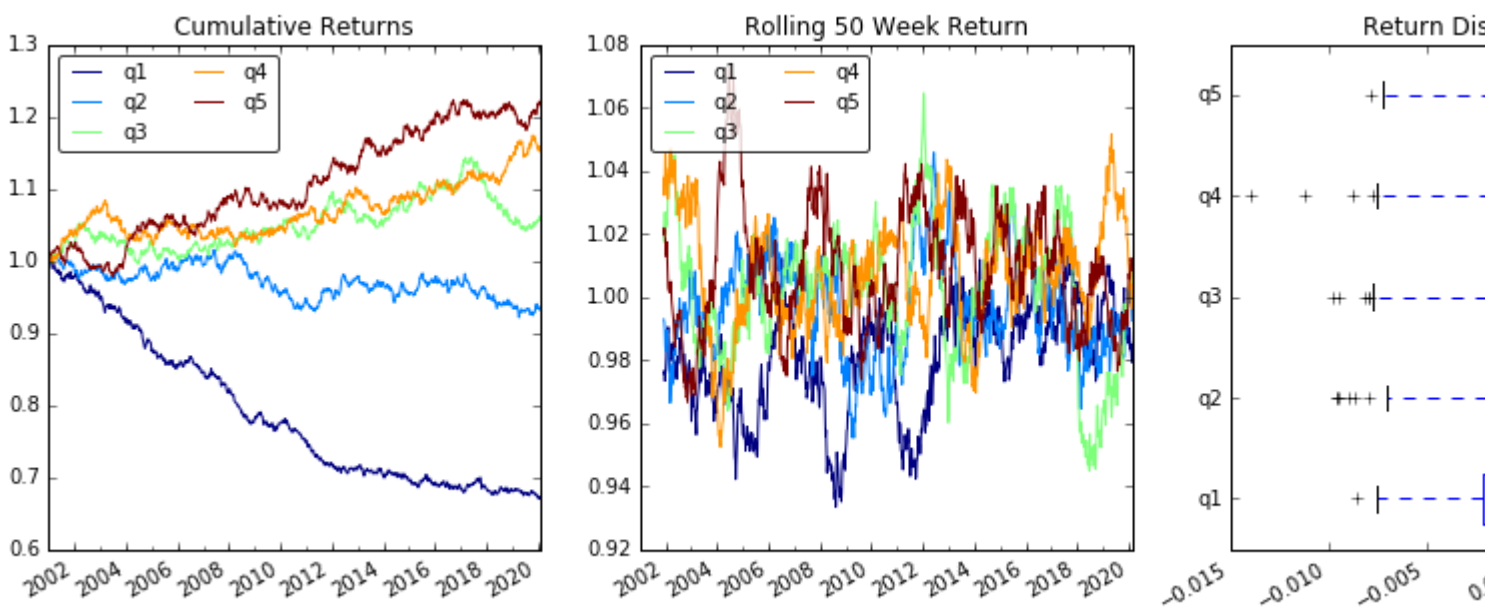
# Cumulative Returns
returns_cut.add(1).cumprod() \
    .plot(colormap='jet', ax=ax1, title="Cumulative Returns")
leg1 = ax1.legend(loc='upper left', ncol=2, prop={'size': 10}, fancybox=True)
leg1.get_frame().set_alpha(.8)

# Rolling 50 Week Return
returns_cut.add(1).rolling(50).apply(lambda x: x.prod()) \
    .plot(colormap='jet', ax=ax2, title="Rolling 50 Week Return")
leg2 = ax2.legend(loc='upper left', ncol=2, prop={'size': 10}, fancybox=True)
leg2.get_frame().set_alpha(.8)

# Return Distribution
returns_cut.plot.box(verte=False, ax=ax3, title="Return Distribution")

fig.autofmt_xdate()

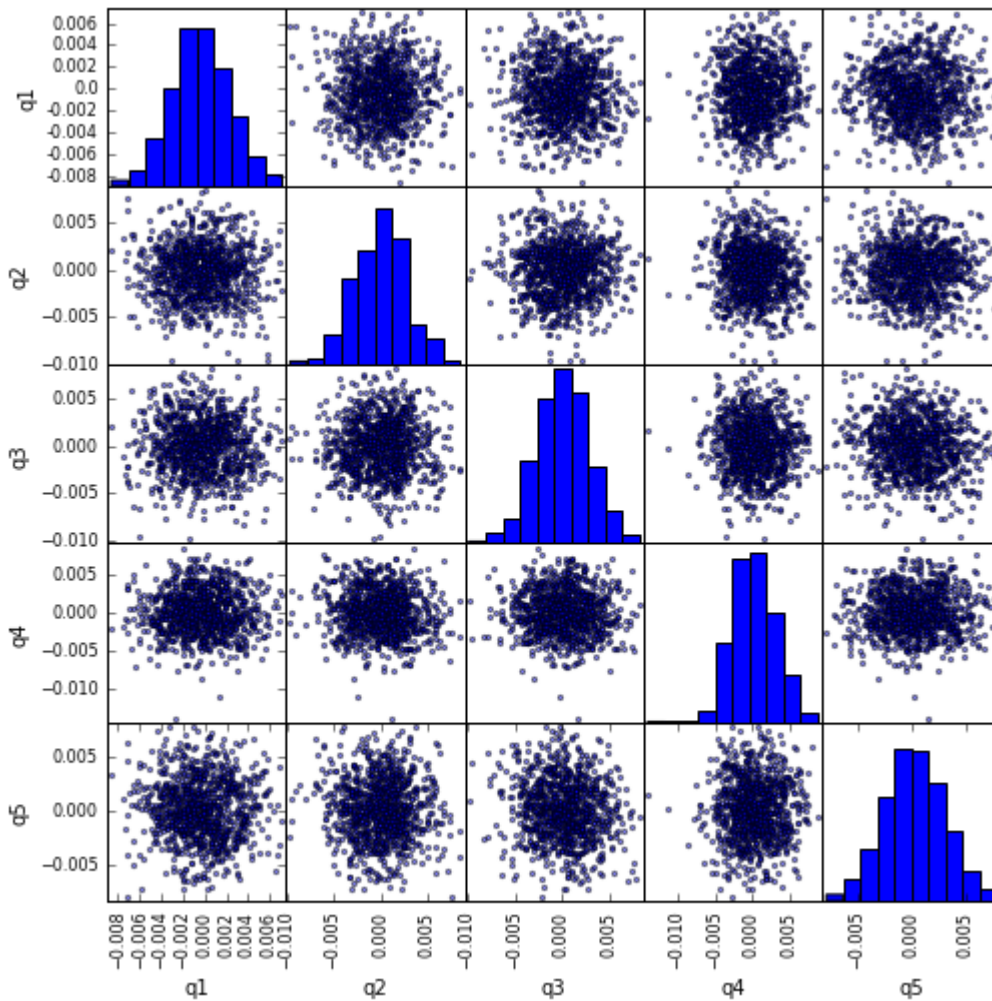
plt.show()
```



## Visualizar la correlación del `scatter_matrix` con `scatter_matrix`

```
from pandas.tools.plotting import scatter_matrix

scatter_matrix(returns_cut, alpha=0.5, figsize=(8, 8), diagonal='hist')
plt.show()
```



## Calcula y visualiza Máximo Draw Down

```
def max_dd(returns):
    """returns is a series"""
    r = returns.add(1).cumprod()
    dd = r.div(r.cummax()).sub(1)
    mdd = dd.min()
    end = dd.argmax()
    start = r.loc[:end].argmax()
    return mdd, start, end

def max_dd_df(returns):
    """returns is a dataframe"""
    series = lambda x: pd.Series(x, ['Draw Down', 'Start', 'End'])
    return returns.apply(max_dd).apply(series)
```

A qué se parece esto

```
max_dd_df(returns_cut)
```



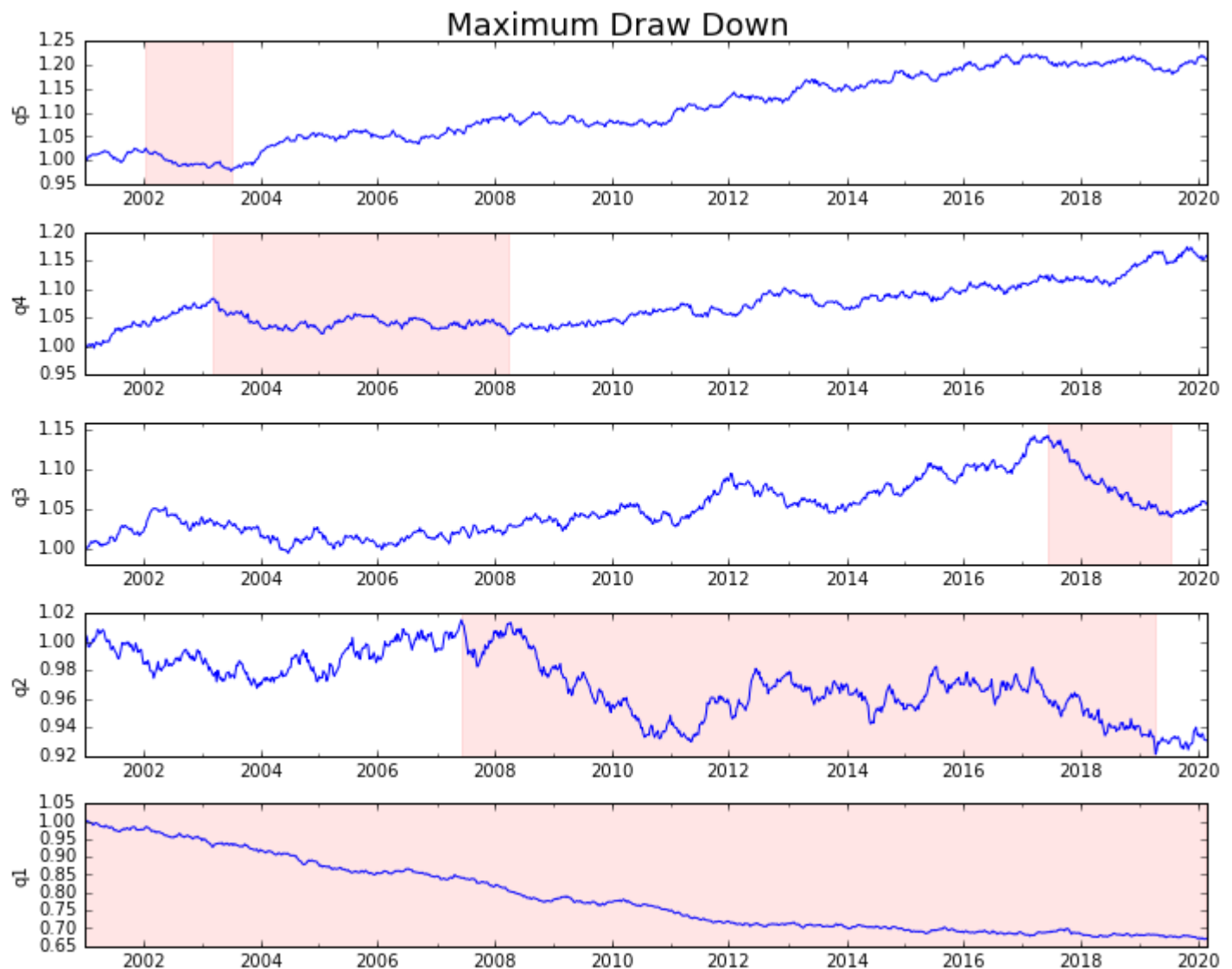
	<b>Draw Down</b>	<b>Start</b>	<b>End</b>
<b>q1</b>	-0.333527	2001-01-07	2020-02-16
<b>q2</b>	-0.092659	2007-06-10	2019-04-14
<b>q3</b>	-0.089682	2017-06-11	2019-07-21
<b>q4</b>	-0.058225	2003-03-16	2008-03-30
<b>q5</b>	-0.046822	2002-01-20	2003-07-06

Vamos a trazarlo

```
draw_downs = max_dd_df(returns_cut)

fig, axes = plt.subplots(5, 1, figsize=(10, 8))
for i, ax in enumerate(axes[:-1]):
    returns_cut.iloc[:, i].add(1).cumprod().plot(ax=ax)
    sd, ed = draw_downs[['Start', 'End']].iloc[i]
    ax.axvspan(sd, ed, alpha=0.1, color='r')
    ax.set_ylabel(returns_cut.columns[i])

fig.suptitle('Maximum Draw Down', fontsize=18)
fig.tight_layout()
plt.subplots_adjust(top=.95)
```



## Calcular estadísticas

Hay muchas estadísticas potenciales que podemos incluir. A continuación se muestran solo algunas, pero demuestre cómo podemos incorporar nuevas estadísticas en nuestro resumen.

```
def frequency_of_time_series(df):
    start, end = df.index.min(), df.index.max()
    delta = end - start
    return round((len(df) - 1.) * 365.25 / delta.days, 2)

def annualized_return(df):
    freq = frequency_of_time_series(df)
    return df.add(1).prod() ** (1 / freq) - 1

def annualized_volatility(df):
    freq = frequency_of_time_series(df)
    return df.std().mul(freq ** .5)

def sharpe_ratio(df):
    return annualized_return(df) / annualized_volatility(df)

def describe(df):
```

```

r = annualized_return(df).rename('Return')
v = annualized_volatility(df).rename('Volatility')
s = sharpe_ratio(df).rename('Sharpe')
skew = df.skew().rename('Skew')
kurt = df.kurt().rename('Kurtosis')
desc = df.describe().T

return pd.concat([r, v, s, skew, kurt, desc], axis=1).T.drop('count')

```

Terminaremos usando solo la función de `describe`, ya que une a todos los demás.

```
describe(returns_cut)
```

	q1	q2	q3	q4	q5
<b>Return</b>	-0.007609	-0.001375	0.001067	0.002821	0.003687
<b>Volatility</b>	0.019584	0.020445	0.020629	0.021185	0.020172
<b>Sharpe</b>	-0.388525	-0.067278	0.051709	0.133176	0.182792
<b>Skew</b>	0.040430	-0.085828	-0.078071	-0.067522	0.005652
<b>Kurtosis</b>	-0.174206	0.203038	0.026385	0.370249	-0.160678
<b>mean</b>	-0.000395	-0.000068	0.000060	0.000151	0.000196
<b>std</b>	0.002711	0.002830	0.002856	0.002933	0.002792
<b>min</b>	-0.008608	-0.009614	-0.009845	-0.014037	-0.007913
<b>25%</b>	-0.002196	-0.002018	-0.001956	-0.001833	-0.001694
<b>50%</b>	-0.000434	0.000065	0.000210	0.000029	0.000146
<b>75%</b>	0.001444	0.001768	0.001989	0.002107	0.002081
<b>max</b>	0.007070	0.008432	0.008100	0.008687	0.007791

Esto no pretende ser exhaustivo. Está pensado para reunir muchas de las características de los pandas y demostrar cómo se puede usar para responder preguntas importantes para usted. Este es un subconjunto de los tipos de métricas que utilizo para evaluar la eficacia de los factores cuantitativos.

Lea **Análisis: Reunirlo todo y tomar decisiones**. en línea:

<https://riptutorial.com/es/pandas/topic/5238/analisis--reunirlo-todo-y-tomar-decisiones->

# Capítulo 4: Anexando a DataFrame

## Examples

### Anexando una nueva fila a DataFrame

```
In [1]: import pandas as pd

In [2]: df = pd.DataFrame(columns = ['A', 'B', 'C'])

In [3]: df
Out[3]:
Empty DataFrame
Columns: [A, B, C]
Index: []
```

### Anexando una fila por un solo valor de columna:

```
In [4]: df.loc[0, 'A'] = 1

In [5]: df
Out[5]:
   A    B    C
0  1  NaN  NaN
```

### Anexando una fila, dada la lista de valores:

```
In [6]: df.loc[1] = [2, 3, 4]

In [7]: df
Out[7]:
   A    B    C
0  1  NaN  NaN
1  2    3    4
```

### Anexando una fila dado un diccionario:

```
In [8]: df.loc[2] = {'A': 3, 'C': 9, 'B': 9}

In [9]: df
Out[9]:
   A    B    C
0  1  NaN  NaN
1  2    3    4
2  3    9    9
```

La primera entrada en `.loc []` es el índice. Si usa un índice existente, sobrescribirá los valores en esa fila:

```
In [17]: df.loc[1] = [5, 6, 7]
```

```
In [18]: df
Out[18]:
```

	A	B	C
0	1	NaN	NaN
1	5	6	7
2	3	9	9

```
In [19]: df.loc[0, 'B'] = 8
```

```
In [20]: df
Out[20]:
```

	A	B	C
0	1	8	NaN
1	5	6	7
2	3	9	9

## Añadir un DataFrame a otro DataFrame

Supongamos que tenemos los siguientes dos DataFrames:

```
In [7]: df1
Out[7]:
```

	A	B
0	a1	b1
1	a2	b2

```
In [8]: df2
Out[8]:
```

	B	C
0	b1	c1

No se requiere que los dos DataFrames tengan el mismo conjunto de columnas. El método de adición no cambia ninguno de los DataFrames originales. En su lugar, devuelve un nuevo DataFrame agregando los dos originales. Anexar un DataFrame a otro es bastante simple:

```
In [9]: df1.append(df2)
Out[9]:
```

	A	B	C
0	a1	b1	NaN
1	a2	b2	NaN
0	NaN	b1	c1

Como puede ver, es posible tener índices duplicados (0 en este ejemplo). Para evitar este problema, puede pedir a Pandas que vuelva a indexar el nuevo DataFrame:

```
In [10]: df1.append(df2, ignore_index = True)
Out[10]:
```

	A	B	C
0	a1	b1	NaN
1	a2	b2	NaN
2	NaN	b1	c1

Lea Anexando a DataFrame en línea: <https://riptutorial.com/es/pandas/topic/6456/anexando-a->

dataframe

---

# Capítulo 5: Calendarios de vacaciones

## Examples

### Crear un calendario personalizado

Aquí es cómo crear un calendario personalizado. El ejemplo dado es un calendario francés, por lo que proporciona muchos ejemplos.

```
from pandas.tseries.holiday import AbstractHolidayCalendar, Holiday, EasterMonday, Easter
from pandas.tseries.offsets import Day, CustomBusinessDay

class FrBusinessCalendar(AbstractHolidayCalendar):
    """ Custom Holiday calendar for France based on
        https://en.wikipedia.org/wiki/Public_holidays_in_France
        - 1 January: New Year's Day
        - Moveable: Easter Monday (Monday after Easter Sunday)
        - 1 May: Labour Day
        - 8 May: Victory in Europe Day
        - Moveable Ascension Day (Thursday, 39 days after Easter Sunday)
        - 14 July: Bastille Day
        - 15 August: Assumption of Mary to Heaven
        - 1 November: All Saints' Day
        - 11 November: Armistice Day
        - 25 December: Christmas Day
    """
    rules = [
        Holiday('New Years Day', month=1, day=1),
        EasterMonday,
        Holiday('Labour Day', month=5, day=1),
        Holiday('Victory in Europe Day', month=5, day=8),
        Holiday('Ascension Day', month=1, day=1, offset=[Easter(), Day(39)]),
        Holiday('Bastille Day', month=7, day=14),
        Holiday('Assumption of Mary to Heaven', month=8, day=15),
        Holiday('All Saints Day', month=11, day=1),
        Holiday('Armistice Day', month=11, day=11),
        Holiday('Christmas Day', month=12, day=25)
    ]
```

### Usa un calendario personalizado

Aquí es cómo utilizar el calendario personalizado.

---

## Consigue las vacaciones entre dos fechas.

```
import pandas as pd
from datetime import date

# Creating some boundaries
year = 2016
start = date(year, 1, 1)
```

```

end = start + pd.offsets.MonthEnd(12)

# Creating a custom calendar
cal = FrBusinessCalendar()
# Getting the holidays (off-days) between two dates
cal.holidays(start=start, end=end)

# DatetimeIndex(['2016-01-01', '2016-03-28', '2016-05-01', '2016-05-05',
#                '2016-05-08', '2016-07-14', '2016-08-15', '2016-11-01',
#                '2016-11-11', '2016-12-25'],
#                dtype='datetime64[ns]', freq=None)

```

## Cuenta el número de días laborables entre dos fechas.

A veces es útil obtener el número de días laborables por mes, sea cual sea el año en el futuro o en el pasado. Aquí es cómo hacer eso con un calendario personalizado.

```

from pandas.tseries.offsets import CDay

# Creating a series of dates between the boundaries
# by using the custom calendar
se = pd.bdate_range(start=start,
                    end=end,
                    freq=CDay(calendar=cal)).to_series()
# Counting the number of working days by month
se.groupby(se.dt.month).count().head()

# 1    20
# 2    21
# 3    22
# 4    21
# 5    21

```

Lea Calendarios de vacaciones en línea: <https://riptutorial.com/es/pandas/topic/7976/calendarios-de-vacaciones>



# Capítulo 6: Creando marcos de datos

## Introducción

**DataFrame** es una estructura de datos proporcionada por la biblioteca pandas, además de *Series* & *Panel*. Es una estructura bidimensional y puede compararse con una tabla de filas y columnas.

Cada fila puede identificarse mediante un índice entero (0..N) o una etiqueta establecida explícitamente al crear un objeto DataFrame. Cada columna puede ser de un tipo distinto y se identifica mediante una etiqueta.

Este tema cubre varias formas de construir / crear un objeto DataFrame. Ex. de matrices numpy, de la lista de tuplas, del diccionario.

## Examples

### Crear un DataFrame de muestra

```
import pandas as pd
```

Cree un marco de datos a partir de un diccionario que contenga dos columnas: `numbers` y `colors`. Cada clave representa un nombre de columna y el valor es una serie de datos, el contenido de la columna:

```
df = pd.DataFrame({'numbers': [1, 2, 3], 'colors': ['red', 'white', 'blue']})
```

Mostrar los contenidos del marco de datos:

```
print(df)
# Output:
#   colors  numbers
# 0    red         1
# 1  white         2
# 2   blue         3
```

Pandas ordena las columnas alfabéticamente como `dict` no están ordenadas. Para especificar el orden, use el parámetro de `columns`.

```
df = pd.DataFrame({'numbers': [1, 2, 3], 'colors': ['red', 'white', 'blue']},
                  columns=['numbers', 'colors'])

print(df)
# Output:
#   numbers colors
# 0         1    red
# 1         2  white
# 2         3   blue
```

## Crea un DataFrame de muestra usando Numpy

Crear un DataFrame de números aleatorios:

```
import numpy as np
import pandas as pd

# Set the seed for a reproducible sample
np.random.seed(0)

df = pd.DataFrame(np.random.randn(5, 3), columns=list('ABC'))

print(df)
# Output:
#           A           B           C
# 0  1.764052  0.400157  0.978738
# 1  2.240893  1.867558 -0.977278
# 2  0.950088 -0.151357 -0.103219
# 3  0.410599  0.144044  1.454274
# 4  0.761038  0.121675  0.443863
```

Crear un DataFrame con enteros:

```
df = pd.DataFrame(np.arange(15).reshape(5,3), columns=list('ABC'))

print(df)
# Output:
#      A  B  C
# 0   0  1  2
# 1   3  4  5
# 2   6  7  8
# 3   9 10 11
# 4  12 13 14
```

Cree un DataFrame e incluya nans ( NaT, NaN, 'nan', None ) en columnas y filas:

```
df = pd.DataFrame(np.arange(48).reshape(8,6), columns=list('ABCDEF'))

print(df)
# Output:
#      A  B  C  D  E  F
# 0   0  1  2  3  4  5
# 1   6  7  8  9 10 11
# 2  12 13 14 15 16 17
# 3  18 19 20 21 22 23
# 4  24 25 26 27 28 29
# 5  30 31 32 33 34 35
# 6  36 37 38 39 40 41
# 7  42 43 44 45 46 47

df.ix[:,2,0] = np.nan # in column 0, set elements with indices 0,2,4, ... to NaN
df.ix[:,4,1] = pd.NaT # in column 1, set elements with indices 0,4, ... to np.NaT
df.ix[3,2] = 'nan'    # in column 2, set elements with index from 0 to 3 to 'nan'
df.ix[:,5] = None     # in column 5, set all elements to None
df.ix[5,:] = None     # in row 5, set all elements to None
df.ix[7,:] = np.nan   # in row 7, set all elements to NaN
```

```
print(df)
# Output:
#      A      B      C      D      E      F
# 0 NaN   NaT   nan    3     4   None
# 1  6     7   nan    9    10   None
# 2 NaN   13   nan   15    16   None
# 3 18    19   nan   21    22   None
# 4 NaN   NaT   26   27    28   None
# 5 NaN   None  None  NaN   NaN   None
# 6 NaN   37   38   39    40   None
# 7 NaN   NaN   NaN  NaN   NaN   NaN
```

## Cree un DataFrame de muestra a partir de múltiples colecciones usando el Diccionario

```
import pandas as pd
import numpy as np

np.random.seed(123)
x = np.random.standard_normal(4)
y = range(4)
df = pd.DataFrame({'X':x, 'Y':y})
>>> df
```

	X	Y
0	-1.085631	0
1	0.997345	1
2	0.282978	2
3	-1.506295	3

## Crear un DataFrame a partir de una lista de tuplas

Puede crear un DataFrame a partir de una lista de tuplas simples, e incluso puede elegir los elementos específicos de las tuplas que desea usar. Aquí crearemos un DataFrame utilizando todos los datos de cada tupla, excepto el último elemento.

```
import pandas as pd

data = [
    ('p1', 't1', 1, 2),
    ('p1', 't2', 3, 4),
    ('p2', 't1', 5, 6),
    ('p2', 't2', 7, 8),
    ('p2', 't3', 2, 8)
]

df = pd.DataFrame(data)

print(df)
```

	0	1	2	3
# 0	p1	t1	1	2
# 1	p1	t2	3	4
# 2	p2	t1	5	6
# 3	p2	t2	7	8
# 4	p2	t3	2	8

## Crear un DataFrame de un diccionario de listas

Cree un DataFrame a partir de varias listas pasando un dict cuyos valores se enumeran. Las claves del diccionario se utilizan como etiquetas de columna. Las listas también pueden ser ndarrays. Las listas / ndarrays deben tener la misma longitud.

```
import pandas as pd

# Create DF from dict of lists/ndarrays
df = pd.DataFrame({'A' : [1, 2, 3, 4],
                  'B' : [4, 3, 2, 1]})

df
# Output:
#      A  B
#  0  1  4
#  1  2  3
#  2  3  2
#  3  4  1
```

Si las matrices no tienen la misma longitud, se genera un error.

```
df = pd.DataFrame({'A' : [1, 2, 3, 4], 'B' : [5, 5, 5]}) # a ValueError is raised
```

## Usando ndarrays

```
import pandas as pd
import numpy as np

np.random.seed(123)
x = np.random.standard_normal(4)
y = range(4)
df = pd.DataFrame({'X':x, 'Y':y})

df
# Output:
#      X  Y
#  0 -1.085631  0
#  1  0.997345  1
#  2  0.282978  2
#  3 -1.506295  3
```

Ver detalles adicionales en: <http://pandas.pydata.org/pandas-docs/stable/dsintro.html#from-dict-of-ndarrays-lists>

## Crear un DataFrame de muestra con datetime

```
import pandas as pd
import numpy as np

np.random.seed(0)
# create an array of 5 dates starting at '2015-02-24', one per minute
rng = pd.date_range('2015-02-24', periods=5, freq='T')
df = pd.DataFrame({'Date': rng, 'Val': np.random.randn(len(rng)) })

print (df)
# Output:
```

```
#
#      Date      Val
# 0 2015-02-24 00:00:00 1.764052
# 1 2015-02-24 00:01:00 0.400157
# 2 2015-02-24 00:02:00 0.978738
# 3 2015-02-24 00:03:00 2.240893
# 4 2015-02-24 00:04:00 1.867558

# create an array of 5 dates starting at '2015-02-24', one per day
rng = pd.date_range('2015-02-24', periods=5, freq='D')
df = pd.DataFrame({'Date': rng, 'Val' : np.random.randn(len(rng))})

print (df)
# Output:
#      Date      Val
# 0 2015-02-24 -0.977278
# 1 2015-02-25  0.950088
# 2 2015-02-26 -0.151357
# 3 2015-02-27 -0.103219
# 4 2015-02-28  0.410599

# create an array of 5 dates starting at '2015-02-24', one every 3 years
rng = pd.date_range('2015-02-24', periods=5, freq='3A')
df = pd.DataFrame({'Date': rng, 'Val' : np.random.randn(len(rng))})

print (df)
# Output:
#      Date      Val
# 0 2015-12-31  0.144044
# 1 2018-12-31  1.454274
# 2 2021-12-31  0.761038
# 3 2024-12-31  0.121675
# 4 2027-12-31  0.443863
```

## DataFrame con `DatetimeIndex` :

```
import pandas as pd
import numpy as np

np.random.seed(0)
rng = pd.date_range('2015-02-24', periods=5, freq='T')
df = pd.DataFrame({'Val' : np.random.randn(len(rng)) }, index=rng)

print (df)
# Output:
#
#      Val
# 2015-02-24 00:00:00 1.764052
# 2015-02-24 00:01:00 0.400157
# 2015-02-24 00:02:00 0.978738
# 2015-02-24 00:03:00 2.240893
# 2015-02-24 00:04:00 1.867558
```

`Offset-aliases` para el parámetro `freq` en `date_range` :

Alias	Description
B	business day frequency
C	custom business day frequency (experimental)
D	calendar day frequency
W	weekly frequency
M	month end frequency

BM	business month end frequency
CBM	custom business month end frequency
MS	month start frequency
BMS	business month start frequency
CBMS	custom business month start frequency
Q	quarter end frequency
BQ	business quarter end frequency
QS	quarter start frequency
BQS	business quarter start frequency
A	year end frequency
BA	business year end frequency
AS	year start frequency
BAS	business year start frequency
BH	business hour frequency
H	hourly frequency
T, min	minutely frequency
S	secondly frequency
L, ms	milliseconds
U, us	microseconds
N	nanoseconds

## Crear un DataFrame de muestra con MultiIndex

```
import pandas as pd
import numpy as np
```

### Utilizando `from_tuples` :

```
np.random.seed(0)
tuples = list(zip(*[['bar', 'bar', 'baz', 'baz',
                    'foo', 'foo', 'qux', 'qux'],
                   ['one', 'two', 'one', 'two',
                    'one', 'two', 'one', 'two']]))

idx = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])
```

### Utilizando `from_product` :

```
idx = pd.MultiIndex.from_product(['bar', 'baz', 'foo', 'qux'], ['one', 'two'])
```

Luego, use este MultiIndex:

```
df = pd.DataFrame(np.random.randn(8, 2), index=idx, columns=['A', 'B'])
print (df)
```

		A	B
first	second		
bar	one	1.764052	0.400157
	two	0.978738	2.240893
baz	one	1.867558	-0.977278
	two	0.950088	-0.151357
foo	one	-0.103219	0.410599
	two	0.144044	1.454274
qux	one	0.761038	0.121675
	two	0.443863	0.333674

## Guardar y cargar un DataFrame en formato pickle (.plk)

```
import pandas as pd

# Save dataframe to pickled pandas object
df.to_pickle(file_name) # where to save it usually as a .plk

# Load dataframe from pickled pandas object
df= pd.read_pickle(file_name)
```

## Crear un DataFrame a partir de una lista de diccionarios

Un DataFrame se puede crear a partir de una lista de diccionarios. Las claves se utilizan como nombres de columna.

```
import pandas as pd
L = [{'Name': 'John', 'Last Name': 'Smith'},
      {'Name': 'Mary', 'Last Name': 'Wood'}]
pd.DataFrame(L)
# Output: Last Name  Name
# 0      Smith  John
# 1      Wood   Mary
```

Los valores faltantes se llenan con NaN s

```
L = [{'Name': 'John', 'Last Name': 'Smith', 'Age': 37},
      {'Name': 'Mary', 'Last Name': 'Wood'}]
pd.DataFrame(L)
# Output:      Age Last Name  Name
#         0    37      Smith  John
#         1   NaN       Wood  Mary
```

Lea Creando marcos de datos en línea: <https://riptutorial.com/es/pandas/topic/1595/creando-marcos-de-datos>

# Capítulo 7: Datos categóricos

## Introducción

**Las categorías** son un tipo de datos pandas, que corresponden a variables categóricas en estadísticas: una variable, que puede tomar solo un número limitado, y generalmente fijo, de valores posibles (categorías; niveles en R). Algunos ejemplos son género, clase social, tipos de sangre, afiliaciones de países, tiempo de observación o calificaciones a través de escalas de Likert. Fuente: [Pandas Docs](#)

## Examples

### Creación de objetos

```
In [188]: s = pd.Series(["a","b","c","a","c"], dtype="category")
```

```
In [189]: s
```

```
Out[189]:
```

```
0    a
1    b
2    c
3    a
4    c
```

```
dtype: category
```

```
Categories (3, object): [a, b, c]
```

```
In [190]: df = pd.DataFrame({"A":["a","b","c","a","c"]})
```

```
In [191]: df["B"] = df["A"].astype('category')
```

```
In [192]: df["C"] = pd.Categorical(df["A"])
```

```
In [193]: df
```

```
Out[193]:
```

```
   A  B  C
0  a  a  a
1  b  b  b
2  c  c  c
3  a  a  a
4  c  c  c
```

```
In [194]: df.dtypes
```

```
Out[194]:
```

```
A      object
B    category
C    category
dtype: object
```

### Creando grandes conjuntos de datos al azar

```
In [1]: import pandas as pd
import numpy as np
```



```
In [2]: df = pd.DataFrame(np.random.choice(['foo', 'bar', 'baz'], size=(100000, 3)))  
      df = df.apply(lambda col: col.astype('category'))
```

```
In [3]: df.head()
```

```
Out[3]:
```

	0	1	2
0	bar	foo	baz
1	baz	bar	baz
2	foo	foo	bar
3	bar	baz	baz
4	foo	bar	baz

```
In [4]: df.dtypes
```

```
Out[4]:
```

```
0    category  
1    category  
2    category  
dtype: object
```

```
In [5]: df.shape
```

```
Out[5]: (100000, 3)
```

Lea Datos categóricos en línea: <https://riptutorial.com/es/pandas/topic/3887/datos-categoricos>

# Capítulo 8: Datos de agrupación

## Examples

### Agrupacion basica

### Agrupar por una columna

Usando el siguiente DataFrame

```
df = pd.DataFrame({'A': ['a', 'b', 'c', 'a', 'b', 'b'],
                  'B': [2, 8, 1, 4, 3, 8],
                  'C': [102, 98, 107, 104, 115, 87]})
```

```
df
# Output:
#   A  B   C
# 0  a  2 102
# 1  b  8  98
# 2  c  1 107
# 3  a  4 104
# 4  b  3 115
# 5  b  8  87
```

Agrupe por columna A y obtenga el valor medio de otras columnas:

```
df.groupby('A').mean()
# Output:
#           B      C
# A
# a  3.000000  103
# b  6.333333  100
# c  1.000000  107
```

### Agrupar por columnas múltiples

```
df.groupby(['A', 'B']).mean()
# Output:
#           C
# A B
# a 2  102.0
#   4  104.0
# b 3  115.0
#   8   92.5
# c 1  107.0
```

Observe cómo después de agrupar cada fila en el DataFrame resultante se indexa por una tupla o [MultiIndex](#) (en este caso, un par de elementos de las columnas A y B).

Para aplicar varios métodos de agregación a la vez, por ejemplo, para contar el número de elementos en cada grupo y calcular su media, use la función `agg` :

```
df.groupby(['A','B']).agg(['count', 'mean'])
# Output:
#           C
#    count  mean
# A B
# a 2      1 102.0
#    4      1 104.0
# b 3      1 115.0
#    8      2  92.5
# c 1      1 107.0
```

## Números de agrupación

Para el siguiente DataFrame:

```
import numpy as np
import pandas as pd
np.random.seed(0)
df = pd.DataFrame({'Age': np.random.randint(20, 70, 100),
                  'Sex': np.random.choice(['Male', 'Female'], 100),
                  'number_of_foo': np.random.randint(1, 20, 100)})

df.head()
# Output:
#    Age    Sex  number_of_foo
# 0   64  Female              14
# 1   67  Female              14
# 2   20  Female              12
# 3   23   Male              17
# 4   23  Female              15
```

Grupo de `Age` en tres categorías (o contenedores). Los contenedores se pueden dar como

- un entero `n` indica el número de contenedores; en este caso, los datos del marco de datos se dividen en `n` intervalos de igual tamaño
- una secuencia de enteros que denota el punto final de los intervalos abiertos a la izquierda en los que los datos se dividen en: por ejemplo, `bins=[19, 40, 65, np.inf]` crea tres grupos de edad `(19, 40]`, `(40, 65]`, y `(65, np.inf]`.

Pandas asigna automáticamente las versiones de cadena de los intervalos como etiqueta.

También es posible definir etiquetas propias definiendo un parámetro de `labels` como una lista de cadenas.

```
pd.cut(df['Age'], bins=4)
# this creates four age groups: (19.951, 32.25] < (32.25, 44.5] < (44.5, 56.75] < (56.75, 69]
Name: Age, dtype: category
Categories (4, object): [(19.951, 32.25] < (32.25, 44.5] < (44.5, 56.75] < (56.75, 69]]

pd.cut(df['Age'], bins=[19, 40, 65, np.inf])
# this creates three age groups: (19, 40], (40, 65] and (65, infinity)
Name: Age, dtype: category
Categories (3, object): [(19, 40] < (40, 65] < (65, inf]]
```

Úsalo en `groupby` para obtener el número medio de `foo`:

```
age_groups = pd.cut(df['Age'], bins=[19, 40, 65, np.inf])
df.groupby(age_groups)['number_of_foo'].mean()
# Output:
# Age
# (19, 40]      9.880000
# (40, 65]      9.452381
# (65, inf]     9.250000
# Name: number_of_foo, dtype: float64
```

Tablas cruzadas por grupos de edad y género:

```
pd.crosstab(age_groups, df['Sex'])
# Output:
# Sex      Female  Male
# Age
# (19, 40]      22    28
# (40, 65]      18    24
# (65, inf]       3     5
```

## Columna de selección de un grupo.

Cuando haces un grupo, puedes seleccionar una sola columna o una lista de columnas:

```
In [11]: df = pd.DataFrame([[1, 1, 2], [1, 2, 3], [2, 3, 4]], columns=["A", "B", "C"])

In [12]: df
Out[12]:
   A  B  C
0  1  1  2
1  1  2  3
2  2  3  4

In [13]: g = df.groupby("A")

In [14]: g["B"].mean()          # just column B
Out[14]:
A
1    1.5
2    3.0
Name: B, dtype: float64

In [15]: g[["B", "C"]].mean()   # columns B and C
Out[15]:
   B    C
A
1  1.5  2.5
2  3.0  4.0
```

También puede usar `agg` para especificar columnas y agregación para realizar:

```
In [16]: g.agg({'B': 'mean', 'C': 'count'})
Out[16]:
   C    B
A
1  2  1.5
2  1  3.0
```

## Agregando por tamaño versus por cuenta

La diferencia entre `size` y `count` es:

`size` cuenta los valores de `NaN`, el `count` no.

```
df = pd.DataFrame(
    {"Name": ["Alice", "Bob", "Mallory", "Mallory", "Bob", "Mallory"],
     "City": ["Seattle", "Seattle", "Portland", "Seattle", "Seattle", "Portland"],
     "Val": [4, 3, 3, np.nan, np.nan, 4]})

df
# Output:
#      City      Name  Val
# 0  Seattle    Alice  4.0
# 1  Seattle     Bob   3.0
# 2  Portland  Mallory  3.0
# 3  Seattle  Mallory  NaN
# 4  Seattle     Bob   NaN
# 5  Portland  Mallory  4.0

df.groupby(["Name", "City"])["Val"].size().reset_index(name='Size')
# Output:
#      Name      City  Size
# 0   Alice  Seattle     1
# 1    Bob   Seattle     2
# 2  Mallory  Portland     2
# 3  Mallory  Seattle     1

df.groupby(["Name", "City"])["Val"].count().reset_index(name='Count')
# Output:
#      Name      City  Count
# 0   Alice  Seattle     1
# 1    Bob   Seattle     1
# 2  Mallory  Portland     2
# 3  Mallory  Seattle     0
```

## Agregando grupos

```
In [1]: import numpy as np
In [2]: import pandas as pd

In [3]: df = pd.DataFrame({'A': list('XYZXYZXYZX'), 'B': [1, 2, 1, 3, 1, 2, 3, 3, 1, 2],
                          'C': [12, 14, 11, 12, 13, 14, 16, 12, 10, 19]})

In [4]: df.groupby('A')['B'].agg({'mean': np.mean, 'standard deviation': np.std})
Out[4]:
      standard deviation      mean
A
X           0.957427  2.250000
Y           1.000000  2.000000
Z           0.577350  1.333333
```

Para múltiples columnas:

```
In [5]: df.groupby('A').agg({'B': [np.mean, np.std], 'C': [np.sum, 'count']})
```

```
Out[5]:
```

	C		B	
	sum	count	mean	std
A				
X	59	4	2.250000	0.957427
Y	39	3	2.000000	1.000000
Z	35	3	1.333333	0.577350

## Exportar grupos en diferentes archivos.

Puede iterar en el objeto devuelto por `groupby()`. El iterador contiene tuplas `(Category, DataFrame)`.

```
# Same example data as in the previous example.
import numpy as np
import pandas as pd
np.random.seed(0)
df = pd.DataFrame({'Age': np.random.randint(20, 70, 100),
                   'Sex': np.random.choice(['Male', 'Female'], 100),
                   'number_of_foo': np.random.randint(1, 20, 100)})

# Export to Male.csv and Female.csv files.
for sex, data in df.groupby('Sex'):
    data.to_csv("{}{}.csv".format(sex))
```

## usar la transformación para obtener estadísticas a nivel de grupo mientras se preserva el marco de datos original

ejemplo:

```
df = pd.DataFrame({'group1' : ['A', 'A', 'A', 'A',
                              'B', 'B', 'B', 'B'],
                  'group2' : ['C', 'C', 'C', 'D',
                              'E', 'E', 'F', 'F'],
                  'B'       : ['one', np.NaN, np.NaN, np.NaN,
                              np.NaN, 'two', np.NaN, np.NaN],
                  'C'       : [np.NaN, 1, np.NaN, np.NaN,
                              np.NaN, np.NaN, np.NaN, 4]})
```

```
df
Out[34]:
```

	B	C	group1	group2
0	one	NaN	A	C
1	NaN	1.0	A	C
2	NaN	NaN	A	C
3	NaN	NaN	A	D
4	NaN	NaN	B	E
5	two	NaN	B	E
6	NaN	NaN	B	F
7	NaN	4.0	B	F

Quiero obtener el recuento de las observaciones no faltantes de B para cada combinación de `group1` y `group2`. `groupby.transform` es una función muy poderosa que hace exactamente eso.

```
df['count_B']=df.groupby(['group1','group2']).B.transform('count')
```

df

Out[36]:

	B	C	group1	group2	count_B
0	one	NaN	A	C	1
1	NaN	1.0	A	C	1
2	NaN	NaN	A	C	1
3	NaN	NaN	A	D	0
4	NaN	NaN	B	E	1
5	two	NaN	B	E	1
6	NaN	NaN	B	F	0
7	NaN	4.0	B	F	0

Lea Datos de agrupación en línea: <https://riptutorial.com/es/pandas/topic/1822/datos-de-agrupacion>

# Capítulo 9: Datos duplicados

## Examples

### Seleccione duplicado

Si es necesario, establezca el valor 0 en la columna B , donde en la columna A los datos duplicados primero crean una máscara mediante `Series.duplicated` y luego usan `DataFrame.ix` o `Series.mask` :

```
In [224]: df = pd.DataFrame({'A':[1,2,3,3,2],
...:                        'B':[1,7,3,0,8]})

In [225]: mask = df.A.duplicated(keep=False)

In [226]: mask
Out[226]:
0    False
1     True
2     True
3     True
4     True
Name: A, dtype: bool

In [227]: df.ix[mask, 'B'] = 0

In [228]: df['C'] = df.A.mask(mask, 0)

In [229]: df
Out[229]:
   A  B  C
0  1  1  1
1  2  0  0
2  3  0  0
3  3  0  0
4  2  0  0
```

Si necesita una máscara invertida use ~ :

```
In [230]: df['C'] = df.A.mask(~mask, 0)

In [231]: df
Out[231]:
   A  B  C
0  1  1  0
1  2  0  2
2  3  0  3
3  3  0  3
4  2  0  2
```

### Drop duplicado

Utilice `drop_duplicates` :



```

In [216]: df = pd.DataFrame({'A':[1,2,3,3,2],
...:                        'B':[1,7,3,0,8]})

In [217]: df
Out[217]:
   A  B
0  1  1
1  2  7
2  3  3
3  3  0
4  2  8

# keep only the last value
In [218]: df.drop_duplicates(subset=['A'], keep='last')
Out[218]:
   A  B
0  1  1
3  3  0
4  2  8

# keep only the first value, default value
In [219]: df.drop_duplicates(subset=['A'], keep='first')
Out[219]:
   A  B
0  1  1
1  2  7
2  3  3

# drop all duplicated values
In [220]: df.drop_duplicates(subset=['A'], keep=False)
Out[220]:
   A  B
0  1  1

```

Cuando no desea obtener una copia de un marco de datos, pero para modificar el existente:

```

In [221]: df = pd.DataFrame({'A':[1,2,3,3,2],
...:                        'B':[1,7,3,0,8]})

In [222]: df.drop_duplicates(subset=['A'], inplace=True)

In [223]: df
Out[223]:
   A  B
0  1  1
1  2  7
2  3  3

```

## Contando y consiguiendo elementos únicos.

Número de elementos únicos en una serie:

```

In [1]: id_numbers = pd.Series([111, 112, 112, 114, 115, 118, 114, 118, 112])
In [2]: id_numbers.nunique()
Out[2]: 5

```

Consigue elementos únicos en una serie:

```
In [3]: id_numbers.unique()
Out[3]: array([111, 112, 114, 115, 118], dtype=int64)

In [4]: df = pd.DataFrame({'Group': list('ABAABABAAB'),
                           'ID': [1, 1, 2, 3, 3, 2, 1, 2, 1, 3]})

In [5]: df
Out[5]:
   Group  ID
0      A    1
1      B    1
2      A    2
3      A    3
4      B    3
5      A    2
6      B    1
7      A    2
8      A    1
9      B    3
```

Número de elementos únicos en cada grupo:

```
In [6]: df.groupby('Group')['ID'].nunique()
Out[6]:
Group
A      3
B      2
Name: ID, dtype: int64
```

Consiga de elementos únicos en cada grupo:

```
In [7]: df.groupby('Group')['ID'].unique()
Out[7]:
Group
A      [1, 2, 3]
B      [1, 3]
Name: ID, dtype: object
```

## Obtener valores únicos de una columna.

```
In [15]: df = pd.DataFrame({"A": [1,1,2,3,1,1], "B": [5,4,3,4,6,7]})

In [21]: df
Out[21]:
   A  B
0  1  5
1  1  4
2  2  3
3  3  4
4  1  6
5  1  7
```

Para obtener valores únicos en las columnas A y B.

```
In [22]: df["A"].unique()
```

```
Out[22]: array([1, 2, 3])

In [23]: df["B"].unique()
Out[23]: array([5, 4, 3, 6, 7])
```

Para obtener los valores únicos en la columna A como una lista (tenga en cuenta que `unique()` se puede utilizar de dos maneras ligeramente diferentes)

```
In [24]: pd.unique(df['A']).tolist()
Out[24]: [1, 2, 3]
```

Aquí hay un ejemplo más complejo. Digamos que queremos encontrar los valores únicos de la columna 'B' donde 'A' es igual a 1.

Primero, introduzcamos un duplicado para que puedas ver cómo funciona. Vamos a reemplazar el 6 en la fila '4', columna 'B' con un 4:

```
In [24]: df.loc['4', 'B'] = 4
Out[24]:
```

	A	B
0	1	5
1	1	4
2	2	3
3	3	4
4	1	4
5	1	7

Ahora seleccione los datos:

```
In [25]: pd.unique(df[df['A'] == 1]['B']).tolist()
Out[25]: [5, 4, 7]
```

Esto se puede descomponer pensando primero en el DataFrame interno:

```
df['A'] == 1
```

Esto encuentra valores en la columna A que son iguales a 1 y les aplica Verdadero o Falso. Luego podemos usar esto para seleccionar valores de la columna 'B' del DataFrame (la selección externa del DataFrame)

A modo de comparación, aquí está la lista si no utilizamos único. Recupera cada valor en la columna 'B' donde la columna 'A' es 1

```
In [26]: df[df['A'] == 1]['B'].tolist()
Out[26]: [5, 4, 4, 7]
```

Lea Datos duplicados en línea: <https://riptutorial.com/es/pandas/topic/2082/datos-duplicados>

# Capítulo 10: Datos perdidos

## Observaciones

¿Debemos incluir el `ffill` y el `bfill` no documentados?

## Examples

### Relleno de valores perdidos

```
In [11]: df = pd.DataFrame([[1, 2, None, 3], [4, None, 5, 6],  
                             [7, 8, 9, 10], [None, None, None, None]])
```

```
Out[11]:  
   0    1    2    3  
0  1.0  2.0  NaN  3.0  
1  4.0  NaN  5.0  6.0  
2  7.0  8.0  9.0 10.0  
3  NaN  NaN  NaN  NaN
```

### Rellene los valores faltantes con un solo valor:

```
In [12]: df.fillna(0)  
Out[12]:  
   0    1    2    3  
0  1.0  2.0  0.0  3.0  
1  4.0  0.0  5.0  6.0  
2  7.0  8.0  9.0 10.0  
3  0.0  0.0  0.0  0.0
```

Esto devuelve un nuevo DataFrame. Si desea cambiar el DataFrame original, use el parámetro `inplace` (`df.fillna(0, inplace=True)`) o vuelva a asignarlo al DataFrame original (`df = df.fillna(0)`).

### Rellene los valores faltantes con los anteriores:

```
In [13]: df.fillna(method='pad') # this is equivalent to both method='ffill' and .ffill()  
Out[13]:  
   0    1    2    3  
0  1.0  2.0  NaN  3.0  
1  4.0  2.0  5.0  6.0  
2  7.0  8.0  9.0 10.0  
3  7.0  8.0  9.0 10.0
```

### Rellena con los siguientes:

```
In [14]: df.fillna(method='bfill') # this is equivalent to .bfill()
Out[14]:
```

	0	1	2	3
0	1.0	2.0	5.0	3.0
1	4.0	8.0	5.0	6.0
2	7.0	8.0	9.0	10.0
3	NaN	NaN	NaN	NaN

## Rellene utilizando otro DataFrame:

```
In [15]: df2 = pd.DataFrame(np.arange(100, 116).reshape(4, 4))
          df2
Out[15]:
```

	0	1	2	3
0	100	101	102	103
1	104	105	106	107
2	108	109	110	111
3	112	113	114	115

```
In [16]: df.fillna(df2) # takes the corresponding cells in df2 to fill df
Out[16]:
```

	0	1	2	3
0	1.0	2.0	102.0	3.0
1	4.0	105.0	5.0	6.0
2	7.0	8.0	9.0	10.0
3	112.0	113.0	114.0	115.0

## Bajando valores perdidos

Cuando se crea un DataFrame, `None` (el valor faltante de python) se convierte a `NaN` (valor faltante de los pandas):

```
In [11]: df = pd.DataFrame([[1, 2, None, 3], [4, None, 5, 6],
                             [7, 8, 9, 10], [None, None, None, None]])
Out[11]:
```

	0	1	2	3
0	1.0	2.0	NaN	3.0
1	4.0	NaN	5.0	6.0
2	7.0	8.0	9.0	10.0
3	NaN	NaN	NaN	NaN

## Eliminar filas si al menos una columna tiene un valor perdido

```
In [12]: df.dropna()
Out[12]:
```

	0	1	2	3
2	7.0	8.0	9.0	10.0

Esto devuelve un nuevo DataFrame. Si desea cambiar el DataFrame original, use el parámetro `inplace` ( `df.dropna(inplace=True)` ) o vuelva a asignarlo al DataFrame original ( `df = df.dropna()` ).

## Eliminar filas si faltan todos los valores de esa fila

```
In [13]: df.dropna(how='all')
Out[13]:
```

	0	1	2	3
0	1.0	2.0	NaN	3.0
1	4.0	NaN	5.0	6.0
2	7.0	8.0	9.0	10.0

## Eliminar *columnas* que no tengan al menos 3 valores no perdidos

```
In [14]: df.dropna(axis=1, thresh=3)
Out[14]:
```

	0	3
0	1.0	3.0
1	4.0	6.0
2	7.0	10.0
3	NaN	NaN

## Interpolación

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'A': [1, 2, np.nan, 3, np.nan],
                   'B': [1.2, 7, 3, 0, 8]})

df['C'] = df.A.interpolate()
df['D'] = df.A.interpolate(method='spline', order=1)

print (df)
```

	A	B	C	D
0	1.0	1.2	1.0	1.000000
1	2.0	7.0	2.0	2.000000
2	NaN	3.0	2.5	2.428571
3	3.0	0.0	3.0	3.000000
4	NaN	8.0	3.0	3.714286

## Comprobación de valores perdidos

Para verificar si un valor es NaN, se pueden usar las funciones `isnull()` o `notnull()` .

```
In [1]: import numpy as np
In [2]: import pandas as pd
In [3]: ser = pd.Series([1, 2, np.nan, 4])
In [4]: pd.isnull(ser)
Out[4]:
```

0	False
1	False
2	True
3	False

```
dtype: bool
```

Tenga en cuenta que `np.nan == np.nan` devuelve `False`, por lo que debe evitar la comparación con `np.nan`:

```
In [5]: ser == np.nan
Out[5]:
0    False
1    False
2    False
3    False
dtype: bool
```

Ambas funciones también se definen como métodos en Series y DataFrames.

```
In [6]: ser.isnull()
Out[6]:
0    False
1    False
2     True
3    False
dtype: bool
```

Pruebas en DataFrames:

```
In [7]: df = pd.DataFrame({'A': [1, np.nan, 3], 'B': [np.nan, 5, 6]})
In [8]: print(df)
Out[8]:
   A    B
0  1.0 NaN
1  NaN  5.0
2  3.0  6.0

In [9]: df.isnull() # If the value is NaN, returns True.
Out[9]:
   A    B
0  False True
1  True  False
2  False False

In [10]: df.notnull() # Opposite of .isnull(). If the value is not NaN, returns True.
Out[10]:
   A    B
0  True False
1  False True
2  True  True
```

Lea Datos perdidos en línea: <https://riptutorial.com/es/pandas/topic/1896/datos-perdidos>

# Capítulo 11: Desplazamiento y desplazamiento de datos

## Examples

### Desplazar o retrasar valores en un marco de datos

```
import pandas as pd

df = pd.DataFrame({'eggs': [1,2,4,8,], 'chickens': [0,1,2,4,]})

df

#   chickens  eggs
# 0         0     1
# 1         1     2
# 2         2     4
# 3         4     8

df.shift()

#   chickens  eggs
# 0       NaN  NaN
# 1       0.0  1.0
# 2       1.0  2.0
# 3       2.0  4.0

df.shift(-2)

#   chickens  eggs
# 0       2.0  4.0
# 1       4.0  8.0
# 2       NaN  NaN
# 3       NaN  NaN

df['eggs'].shift(1) - df['chickens']

# 0     NaN
# 1     0.0
# 2     0.0
# 3     0.0
```

El primer argumento para `.shift()` es `periods`, el número de espacios para mover los datos. Si no se especifica, el valor predeterminado es `1`.

Lea Desplazamiento y desplazamiento de datos en línea:

<https://riptutorial.com/es/pandas/topic/7554/desplazamiento-y-desplazamiento-de-datos>



# Capítulo 12: Fusionar, unir y concatenar

## Sintaxis

- Marco de datos. **fusionar** (a la *derecha*, *cómo* = 'interno', *en* = Ninguna, *left\_on* = Ninguna, *right\_on* = Ninguna, *left\_index* = False, *right\_index* = False, *sort* = False, *sufijos* = ('\_x', '\_y'), *copy* = True, *indicador* = falso )
- Combine los objetos DataFrame realizando una operación de combinación de estilo de base de datos por columnas o índices.
- Si se unen columnas en columnas, los índices de DataFrame se ignorarán. De lo contrario, si se unen índices en índices o índices en una columna o columnas, se pasará el índice.

## Parámetros

Parámetros	Explicación
Correcto	Marco de datos
cómo	{'izquierda', 'derecha', 'exterior', 'interno'}, por defecto 'interno'
dejado en	etiqueta o lista, o como una matriz. Los nombres de los campos para unirse en el marco de datos izquierdo. Puede ser un vector o una lista de vectores de la longitud del DataFrame para usar un vector particular como la clave de unión en lugar de columnas
tocar el asunto exacto	etiqueta o lista, o como una matriz. Nombres de campo para unir en el marco de datos correcto o vector / lista de vectores por documentos left_on
left_index	booleano, por defecto Falso. Utilice el índice del DataFrame izquierdo como la (s) clave (s) de unión. Si es un MultiIndex, el número de claves en el otro DataFrame (ya sea el índice o un número de columnas) debe coincidir con el número de niveles
right_index	booleano, por defecto Falso. Utilice el índice del DataFrame correcto como la clave de unión. Las mismas advertencias que left_index
ordenar	booleano, por defecto Fals. Ordenar las claves de unión lexicográficamente en el resultado DataFrame
sufijos	Secuencia de 2 longitudes (tupla, lista, ...). Sufijo para aplicar a los nombres de columnas superpuestas en el lado izquierdo y derecho, respectivamente
dupdo	booleano, por defecto verdadero. Si es falso, no copie datos

Parámetros	Explicación
	innecesariamente
indicador	booleano o cadena, predeterminado Falso. Si es verdadero, agrega una columna para generar el marco de datos llamado "_merge" con información sobre el origen de cada fila. Si es una cadena, la columna con información sobre el origen de cada fila se agregará a DataFrame de salida, y la columna se llamará valor de cadena. La columna de información es de tipo categórico y toma un valor de "left_only" para las observaciones cuya clave de combinación solo aparece en el marco de datos 'izquierdo', 'right_only' para las observaciones cuya clave de combinación solo aparece en el correcto 'DataFrame' y 'ambos' si La clave de combinación de la observación se encuentra en ambos.

## Examples

### Unir

Por ejemplo, se dan dos tablas,

#### T1

id	x	y
8	42	1.9
9	30	1.9

#### T2

id	signal
8	55
8	56
8	59
9	57
9	58
9	60

El objetivo es conseguir la nueva tabla T3:

id	x	y	s1	s2	s3
8	42	1.9	55	56	58
9	30	1.9	57	58	60

Lo que consiste en crear las columnas `s1` , `s2` y `s3` , cada una correspondiente a una fila (el número de filas por `id` siempre es fijo e igual a 3)

Aplicando `join` (que toma un opcional en el argumento que puede ser una columna o varios nombres de columna, que especifica que el DataFrame pasado se alineará en esa columna en el DataFrame). Así que la solución puede ser como se muestra a continuación:

```
df = df1.merge(df2.groupby('id')['señal'].apply(lambda x: x.reset_index(drop=True)).unstack().reset_index())
```

```
df
Out[63]:
```

	id	x	y	0	1	2
0	8	42	1.9	55	56	59
1	9	30	1.9	57	58	60

Si los separo:

```
df2t = df2.groupby('id')['signal'].apply(lambda x:
x.reset_index(drop=True)).unstack().reset_index()
```

```
df2t
Out[59]:
```

	id	0	1	2
0	8	55	56	59
1	9	57	58	60

```
df = df1.merge(df2t)
```

```
df
Out[61]:
```

	id	x	y	0	1	2
0	8	42	1.9	55	56	59
1	9	30	1.9	57	58	60

## Fusionando dos DataFrames

```
In [1]: df1 = pd.DataFrame({'x': [1, 2, 3], 'y': ['a', 'b', 'c']})
```

```
In [2]: df2 = pd.DataFrame({'y': ['b', 'c', 'd'], 'z': [4, 5, 6]})
```

```
In [3]: df1
```

```
Out[3]:
```

	x	y
0	1	a
1	2	b
2	3	c

```
In [4]: df2
```

```
Out[4]:
```

	y	z
0	b	4
1	c	5
2	d	6

## Unir internamente:

Utiliza la intersección de claves de dos DataFrames.

```
In [5]: df1.merge(df2) # by default, it does an inner join on the common column(s)
```

```
Out[5]:
   x  y  z
0  2  b  4
1  3  c  5
```

Alternativamente, especifique la intersección de claves de dos Dataframes.

```
In [5]: merged_inner = pd.merge(left=df1, right=df2, left_on='y', right_on='y')
Out[5]:
   x  y  z
0  2  b  4
1  3  c  5
```

---

## Unión externa:

Utiliza la unión de las claves de dos DataFrames.

```
In [6]: df1.merge(df2, how='outer')
Out[6]:
   x  y  z
0  1.0 a NaN
1  2.0 b 4.0
2  3.0 c 5.0
3  NaN d 6.0
```

---

## Unirse a la izquierda:

Utiliza solo claves de DataFrame izquierdo.

```
In [7]: df1.merge(df2, how='left')
Out[7]:
   x  y  z
0  1  a NaN
1  2  b 4.0
2  3  c 5.0
```

---

## Unirse a la derecha

Utiliza solo claves del DataFrame correcto.

```
In [8]: df1.merge(df2, how='right')
Out[8]:
   x  y  z
0  2.0 b  4
1  3.0 c  5
2  NaN d  6
```

## Fusionar / concatenar / unir múltiples marcos de datos (horizontal y verticalmente)

generar marcos de datos de muestra:

```
In [57]: df3 = pd.DataFrame({'col1':[211,212,213], 'col2': [221,222,223]})

In [58]: df1 = pd.DataFrame({'col1':[11,12,13], 'col2': [21,22,23]})

In [59]: df2 = pd.DataFrame({'col1':[111,112,113], 'col2': [121,122,123]})

In [60]: df3 = pd.DataFrame({'col1':[211,212,213], 'col2': [221,222,223]})

In [61]: df1
Out[61]:
   col1  col2
0     11    21
1     12    22
2     13    23

In [62]: df2
Out[62]:
   col1  col2
0    111   121
1    112   122
2    113   123

In [63]: df3
Out[63]:
   col1  col2
0    211   221
1    212   222
2    213   223
```

fusionar / unir / concatenar marcos de datos [df1, df2, df3] verticalmente - agregar filas

```
In [64]: pd.concat([df1,df2,df3], ignore_index=True)
Out[64]:
   col1  col2
0     11    21
1     12    22
2     13    23
3    111   121
4    112   122
5    113   123
6    211   221
7    212   222
8    213   223
```

fusionar / unir / concatenar marcos de datos horizontalmente (alineación por índice):

```
In [65]: pd.concat([df1,df2,df3], axis=1)
Out[65]:
   col1  col2  col1  col2  col1  col2
0     11    21    111   121    211   221
1     12    22    112   122    212   222
2     13    23    113   123    213   223
```

## Fusionar, Unir y Concat

### Fusionar nombres de claves son iguales

```
pd.merge(df1, df2, on='key')
```

### Fusionar nombres de claves son diferentes

```
pd.merge(df1, df2, left_on='l_key', right_on='r_key')
```

### Diferentes tipos de unión.

```
pd.merge(df1, df2, on='key', how='left')
```

### Fusionando en múltiples claves

```
pd.merge(df1, df2, on=['key1', 'key2'])
```

### Tratamiento de columnas superpuestas

```
pd.merge(df1, df2, on='key', suffixes=('_left', '_right'))
```

### Usando el índice de filas en lugar de combinar claves

```
pd.merge(df1, df2, right_index=True, left_index=True)
```

Evite el uso de la sintaxis `.join` ya que da una excepción para las columnas superpuestas

### Fusión en el índice de marco de datos izquierdo y la columna de marco de datos derecha

```
pd.merge(df1, df2, right_index=True, left_on='l_key')
```

### Marcos de datos concat

#### Pegado verticalmente

```
pd.concat([df1, df2, df3], axis=0)
```

#### Pegado horizontalmente

```
pd.concat([df1, df2, df3], axis=1)
```

## ¿Cuál es la diferencia entre unirse y fusionarse?

Considere los marcos de datos de `left` y `right`

```
left = pd.DataFrame(['a', 1], ['b', 2], list('XY'), list('AB'))
left
```

	A	B
X	a	1
Y	b	2

```
right = pd.DataFrame(['a', 3], ['b', 4], list('XY'), list('AC'))
right
```

	A	C
X	a	3
Y	b	4

### join

Piense en `join` como si quisiera combinarlos a los marcos de datos en función de sus índices respectivos. Si hay columnas superpuestas, `join` querrá que agregue un sufijo al nombre de la columna superpuesta del marco de datos de la izquierda. Nuestros dos marcos de datos tienen un nombre de columna superpuesto `A`

```
left.join(right, lsuffix='_')
```

	A_	B	A	C
X	a	1	a	3
Y	b	2	b	4

Note que el índice se conserva y tenemos 4 columnas. 2 columnas de `left` y 2 de `right`.

Si los índices no se alineaban

```
left.join(right.reset_index(), lsuffix='_', how='outer')
```

	A_	B	index	A	C
0	NaN	NaN	X	a	3.0
1	NaN	NaN	Y	b	4.0
X	a	1.0	NaN	NaN	NaN
Y	b	2.0	NaN	NaN	NaN

Utilicé una combinación externa para ilustrar mejor el punto. Si los índices no se alinean, el resultado será la unión de los índices.

Podemos decirle a `join` que use una columna específica en el marco de datos de la izquierda para usarla como clave de join, pero seguirá usando el índice de la derecha.

```
left.reset_index().join(right, on='index', lsuffix='_')
```

	index	A_	B	A	C
0	X	a	1	a	3
1	Y	b	2	b	4

### merge

Piense en la `merge` como alineación en columnas. Por defecto, la `merge` buscará columnas

superpuestas en las que se fusionará. `merge` proporciona un mejor control sobre las claves de combinación al permitir al usuario especificar un subconjunto de las columnas superpuestas para usar con el parámetro `on`, o permitir por separado la especificación de qué columnas de la izquierda y qué columnas de la derecha se fusionan.

`merge` devolverá un marco de datos combinado en el que se destruirá el índice.

Este sencillo ejemplo encuentra que la columna superpuesta es 'A' y se combina en función de ella.

```
left.merge(right)
```

	A	B	C
0	a	1	3
1	b	2	4

Tenga en cuenta que el índice es `[0, 1]` y ya no es `['X', 'Y']`

Puede especificar explícitamente que está fusionando en el índice con la `left_index` o `right_index` parametro

```
left.merge(right, left_index=True, right_index=True, suffixes=['_', ''])
```

	A_	B	A	C
X	a	1	a	3
Y	b	2	b	4

Y esto se ve exactamente como el ejemplo de `join` anterior.

Lea **Fusionar, unir y concatenar en línea**: <https://riptutorial.com/es/pandas/topic/1966/fusionar--unir-y-concatenar>



---

# Capítulo 13: Gotchas de pandas

## Observaciones

Gotcha en general es una construcción que está documentada, pero no es intuitiva. Los gotchas producen una salida que normalmente no se espera debido a su carácter contraintuitivo.

El paquete de Pandas tiene varios errores, que pueden confundir a alguien que no los conoce, y algunos de ellos se presentan en esta página de documentación.

## Examples

### Detectando valores perdidos con np.nan

Si quieres detectar faltas con

```
df=pd.DataFrame({'col':[1,np.nan]})
df==np.nan
```

Obtendrás el siguiente resultado:

```
col
0   False
1   False
```

Esto se debe a que comparar el valor faltante con cualquier cosa da como resultado un Falso; en lugar de esto, debe usar

```
df=pd.DataFrame({'col':[1,np.nan]})
df.isnull()
```

lo que resulta en:

```
col
0   False
1    True
```

### Integer y NA

Las pandas no admiten la falta de atributos de tipo entero. Por ejemplo, si tiene faltas en la columna de calificación:

```
df= pd.read_csv("data.csv", dtype={'grade': int})
error: Integer column has NA values
```

En este caso, solo debes usar float en lugar de enteros o establecer el tipo de objeto.

## Alineación automática de datos (comportamiento indexado)

Si desea agregar una serie de valores [1,2] a la columna de dataframe df, obtendrá NaNs:

```
import pandas as pd

series=pd.Series([1,2])
df=pd.DataFrame(index=[3,4])
df['col']=series
df
```

	col
3	NaN
4	NaN

porque la configuración de una nueva columna alinea automáticamente los datos por el índice, y sus valores 1 y 2 obtendrían los índices 0 y 1, y no 3 y 4 como en su marco de datos:

```
df=pd.DataFrame(index=[1,2])
df['col']=series
df
```

	col
1	2.0
2	NaN

Si desea ignorar el índice, debe configurar los valores al final:

```
df['col']=series.values
```

	col
3	1
4	2

Lea Gotchas de pandas en línea: <https://riptutorial.com/es/pandas/topic/6425/gotchas-de-pandas>

# Capítulo 14: Gráficos y visualizaciones

## Examples

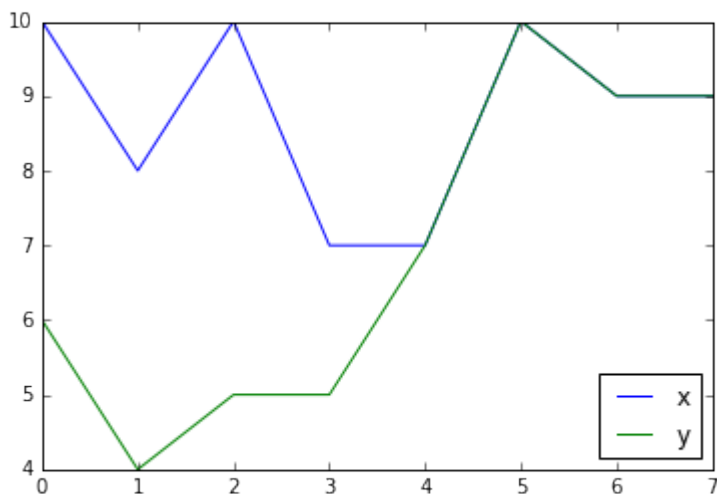
### Gráficos de datos básicos

Los usos de Pandas proporcionan múltiples formas de hacer gráficos de los datos dentro del marco de datos. Utiliza [matplotlib](#) para ese propósito.

Los gráficos básicos tienen sus envoltorios para los objetos DataFrame y Series:

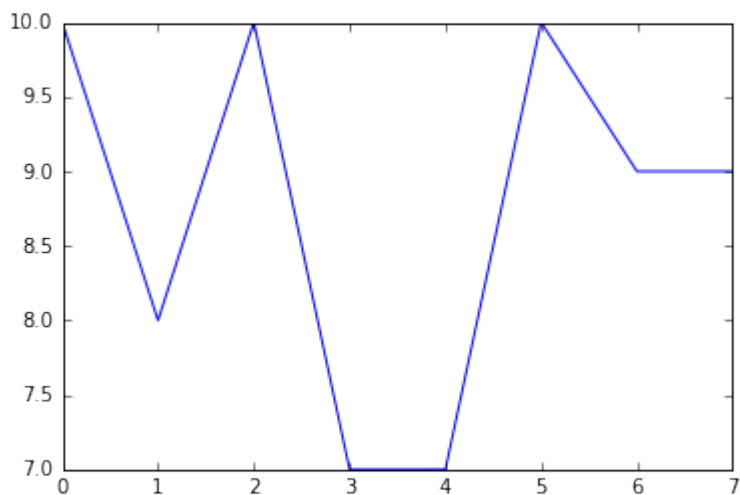
#### Línea Plot

```
df = pd.DataFrame({'x': [10, 8, 10, 7, 7, 10, 9, 9],  
                  'y': [6, 4, 5, 5, 7, 10, 9, 9]})  
df.plot()
```



Puede llamar al mismo método para un objeto Serie para trazar un subconjunto del Marco de datos:

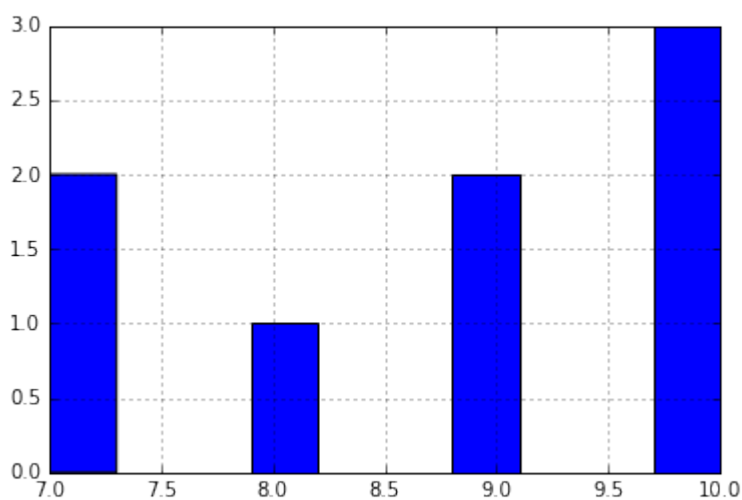
```
df['x'].plot()
```



## Gráfico de barras

Si desea explorar la distribución de sus datos, puede utilizar el método `hist()` .

```
df['x'].hist()
```



## Método general para trazar `parcels()`

Todos los gráficos posibles están disponibles a través del método de trazado. El tipo de gráfico es seleccionado por el argumento **kind** .

```
df['x'].plot(kind='pie')
```



**Nota** En muchos entornos, el gráfico circular saldrá un óvalo. Para hacer un círculo, usa lo siguiente:

```
from matplotlib import pyplot

pyplot.axis('equal')
df['x'].plot(kind='pie')
```

## Estilo de la trama

`plot()` puede tomar argumentos que se pasan a matplotlib para diseñar la trama de diferentes maneras.

```
df.plot(style='o') # plot as dots, not lines
df.plot(style='g--') # plot as green dashed line
df.plot(style='o', markeredgcolor='white') # plot as dots with white edge
```

## Parcela en un eje de matplotlib existente

Por defecto, `plot()` crea una nueva figura cada vez que se llama. Es posible trazar en un eje existente pasando el parámetro `ax`.

```
plt.figure() # create a new figure
ax = plt.subplot(121) # create the left-side subplot
df1.plot(ax=ax) # plot df1 on that subplot
ax = plt.subplot(122) # create the right-side subplot
df2.plot(ax=ax) # and plot df2 there
plt.show() # show the plot
```

Lea Gráficos y visualizaciones en línea: <https://riptutorial.com/es/pandas/topic/3839/graficos-y-visualizaciones>

# Capítulo 15: Guardar pandas dataframe en un archivo csv

## Parámetros

Parámetro	Descripción
path_or_buf	cadena o identificador de archivo, por defecto Ninguno Ruta de archivo de archivo u objeto, si se proporciona Ninguno, el resultado se devuelve como una cadena.
sep	carácter, predeterminado ',' delimitador de campo para el archivo de salida.
na_rep	cadena, por defecto " Representación de datos faltantes
float_format	cadena, por defecto Ninguno Cadena de formato para números de punto flotante
columnas	Secuencias, columnas opcionales para escribir.
encabezamiento	booleano o lista de cadenas, nombres de columna de escritura verdadera predeterminados. Si se proporciona una lista de cadenas, se asume que son alias para los nombres de columna
índice	booleano, nombres de fila de escritura verdadera predeterminados (índice)
index_label	cadena o secuencia, o Falso, predeterminado Ninguno Etiqueta de columna para columnas de índice, si lo desea. Si se da Ninguno, y el encabezado y el índice son Verdaderos, entonces se usan los nombres del índice. Se debe dar una secuencia si el DataFrame usa MultiIndex. Si es falso, no imprima campos para los nombres de índice. Use index_label = False para facilitar la importación en R
nanRep	Ninguno en desuso, use na_rep
modo	modo de escritura str Python, por defecto 'w'
codificación	cadena, opcional Una cadena que representa la codificación a usar en el archivo de salida, por defecto es 'ascii' en Python 2 y 'utf-8' en Python 3.
compresión	cadena, opcional una cadena que representa la compresión a usar en el archivo de salida, los valores permitidos son 'gzip', 'bz2', 'xz', solo se usan cuando el primer argumento es un nombre de archivo

Parámetro	Descripción
line_terminator	cadena, por defecto 'n' El carácter de nueva línea o secuencia de caracteres para usar en el archivo de salida
citando	constante opcional del módulo csv por defecto a csv.QUOTE_MINIMAL
cotizar	cadena (longitud 1), por defecto el carácter " 'usado para citar campos
doble cita	booleano, cotización True Control por defecto de quotechar dentro de un campo
escapechar	cadena (longitud 1), por defecto Ninguno carácter utilizado para escapar de sep y quotechar cuando sea apropiado
tamaño de porción	Int o None filas para escribir a la vez
tupleize_cols	booleano, predeterminado False escribe columnas multi_index como una lista de tuplas (si es verdadero) o nuevo (formato expandido) si es falso)
formato de fecha	cadena, por defecto Ninguno Cadena de formato para objetos de fecha y hora
decimal	cadena, por defecto '.' Carácter reconocido como separador decimal. Por ejemplo, use ',' para datos europeos

## Examples

### Crear un marco de datos aleatorio y escribir en .csv

Crear un DataFrame simple.

```
import numpy as np
import pandas as pd

# Set the seed so that the numbers can be reproduced.
np.random.seed(0)

df = pd.DataFrame(np.random.randn(5, 3), columns=list('ABC'))

# Another way to set column names is
"columns=['column_1_name','column_2_name','column_3_name']"

df
```

	A	B	C
0	1.764052	0.400157	0.978738
1	2.240893	1.867558	-0.977278
2	0.950088	-0.151357	-0.103219
3	0.410599	0.144044	1.454274
4	0.761038	0.121675	0.443863

Ahora, escribe en un archivo CSV:

```
df.to_csv('example.csv', index=False)
```

Contenido de example.csv:

```
A,B,C
1.76405234597,0.400157208367,0.978737984106
2.2408931992,1.86755799015,-0.977277879876
0.950088417526,-0.151357208298,-0.103218851794
0.410598501938,0.144043571161,1.45427350696
0.761037725147,0.121675016493,0.443863232745
```

Tenga en cuenta que especificamos `index=False` para que los índices generados automáticamente (fila #s 0,1,2,3,4) no se incluyan en el archivo CSV. Inclúyelo si necesita la columna de índice, así:

```
df.to_csv('example.csv', index=True) # Or just leave off the index param; default is True
```

Contenido de example.csv:

```
,A,B,C
0,1.76405234597,0.400157208367,0.978737984106
1,2.2408931992,1.86755799015,-0.977277879876
2,0.950088417526,-0.151357208298,-0.103218851794
3,0.410598501938,0.144043571161,1.45427350696
4,0.761037725147,0.121675016493,0.443863232745
```

También tenga en cuenta que puede eliminar el encabezado si no es necesario con `header=False` . Esta es la salida más simple:

```
df.to_csv('example.csv', index=False, header=False)
```

Contenido de example.csv:

```
1.76405234597,0.400157208367,0.978737984106
2.2408931992,1.86755799015,-0.977277879876
0.950088417526,-0.151357208298,-0.103218851794
0.410598501938,0.144043571161,1.45427350696
0.761037725147,0.121675016493,0.443863232745
```

El delimitador se puede establecer por `sep=` argumento, aunque el separador estándar para archivos csv es `,` .

```
df.to_csv('example.csv', index=False, header=False, sep='\t')
```

```
1.76405234597    0.400157208367    0.978737984106
2.2408931992    1.86755799015    -0.977277879876
0.950088417526   -0.151357208298   -0.103218851794
0.410598501938   0.144043571161    1.45427350696
0.761037725147   0.121675016493    0.443863232745
```



## Guarde Pandas DataFrame de la lista a los dictados a CSV sin índice y con codificación de datos

```
import pandas as pd
data = [
    {'name': 'Daniel', 'country': 'Uganda'},
    {'name': 'Yao', 'country': 'China'},
    {'name': 'James', 'country': 'Colombia'},
]
df = pd.DataFrame(data)
filename = 'people.csv'
df.to_csv(filename, index=False, encoding='utf-8')
```

Lea Guardar pandas dataframe en un archivo csv en línea:

<https://riptutorial.com/es/pandas/topic/1558/guardar-pandas-dataframe-en-un-archivo-csv>

# Capítulo 16: Herramientas computacionales

## Examples

### Encuentra la correlación entre columnas

Supongamos que tiene un DataFrame de valores numéricos, por ejemplo:

```
df = pd.DataFrame(np.random.randn(1000, 3), columns=['a', 'b', 'c'])
```

Entonces

```
>>> df.corr()
      a      b      c
a  1.000000  0.018602  0.038098
b  0.018602  1.000000 -0.014245
c  0.038098 -0.014245  1.000000
```

Encontrará la [correlación de Pearson](#) entre las columnas. Observe cómo la diagonal es 1, ya que cada columna está (obviamente) completamente correlacionada consigo misma.

`pd.DataFrame.correlation` toma un parámetro de `method` opcional, especificando qué algoritmo usar. El valor predeterminado es `pearson`. Para usar la correlación de Spearman, por ejemplo, use

```
>>> df.corr(method='spearman')
      a      b      c
a  1.000000  0.007744  0.037209
b  0.007744  1.000000 -0.011823
c  0.037209 -0.011823  1.000000
```

Lea [Herramientas computacionales en línea](#):

<https://riptutorial.com/es/pandas/topic/5620/herramientas-computacionales>

# Capítulo 17: Herramientas de Pandas IO (leer y guardar conjuntos de datos)

## Observaciones

La documentación oficial de pandas incluye una página en [IO Tools](#) con una lista de funciones relevantes para leer y escribir en archivos, así como algunos ejemplos y parámetros comunes.

## Examples

### Leyendo el archivo csv en DataFrame

Ejemplo para leer el archivo `data_file.csv` como:

### Expediente:

```
index,header1,header2,header3
1,str_data,12,1.4
3,str_data,22,42.33
4,str_data,2,3.44
2,str_data,43,43.34

7, str_data, 25, 23.32
```

### Código:

```
pd.read_csv('data_file.csv')
```

### Salida:

	index	header1	header2	header3
0	1	str_data	12	1.40
1	3	str_data	22	42.33
2	4	str_data	2	3.44
3	2	str_data	43	43.34
4	7	str_data	25	23.32

### Algunos argumentos útiles:

- **sep** El delimitador de campo predeterminado es una coma `,`. Use esta opción si necesita un delimitador diferente, por ejemplo `pd.read_csv('data_file.csv', sep=';')`
- **index\_col** Con `index_col = n` (`n` un entero) le dice a pandas que use la columna `n` para

indexar el DataFrame. En el ejemplo anterior:

```
pd.read_csv('data_file.csv', index_col=0)
```

Salida:

	header1	header2	header3
index			
1	str_data	12	1.40
3	str_data	22	42.33
4	str_data	2	3.44
2	str_data	43	43.34
7	str_data	25	23.32

- **skip\_blank\_lines** Por defecto, las líneas en blanco se omiten. Use `skip_blank_lines=False` para incluir líneas en blanco (se llenarán con valores de `NaN` )

```
pd.read_csv('data_file.csv', index_col=0, skip_blank_lines=False)
```

Salida:

	header1	header2	header3
index			
1	str_data	12	1.40
3	str_data	22	42.33
4	str_data	2	3.44
2	str_data	43	43.34
NaN	NaN	NaN	NaN
7	str_data	25	23.32

- **parse\_dates** Use esta opción para analizar datos de fecha.

Expediente:

```
date_begin;date_end;header3;header4;header5
1/1/2017;1/10/2017;str_data;1001;123,45
2/1/2017;2/10/2017;str_data;1001;67,89
3/1/2017;3/10/2017;str_data;1001;0
```

Código para analizar las columnas 0 y 1 como fechas:

```
pd.read_csv('f.csv', sep=';', parse_dates=[0,1])
```

Salida:

	date_begin	date_end	header3	header4	header5
0	2017-01-01	2017-01-10	str_data	1001	123,45
1	2017-02-01	2017-02-10	str_data	1001	67,89
2	2017-03-01	2017-03-10	str_data	1001	0

Por defecto, el formato de fecha es inferido. Si desea especificar un formato de fecha,

puede utilizar, por ejemplo,

```
dateparse = lambda x: pd.datetime.strptime(x, '%d/%m/%Y')
pd.read_csv('f.csv', sep=';', parse_dates=[0,1], date_parser=dateparse)
```

Salida:

	date_begin	date_end	header3	header4	header5
0	2017-01-01	2017-10-01	str_data	1001	123,45
1	2017-01-02	2017-10-02	str_data	1001	67,89
2	2017-01-03	2017-10-03	str_data	1001	0

Puede encontrar más información sobre los parámetros de la función en la [documentación oficial](#) .

## Guardado básico en un archivo csv

```
raw_data = {'first_name': ['John', 'Jane', 'Jim'],
            'last_name': ['Doe', 'Smith', 'Jones'],
            'department': ['Accounting', 'Sales', 'Engineering'],}
df = pd.DataFrame(raw_data, columns=raw_data.keys())
df.to_csv('data_file.csv')
```

## Fechas de análisis al leer de CSV

Puede especificar una columna que contenga fechas para que los pandas las analicen automáticamente al leer desde el csv

```
pandas.read_csv('data_file.csv', parse_dates=['date_column'])
```

## Hoja de cálculo para dictado de DataFrames

```
with pd.ExcelFile('path_to_file.xls') as xl:
    d = {sheet_name: xl.parse(sheet_name) for sheet_name in xl.sheet_names}
```

## Lee una hoja específica

```
pd.read_excel('path_to_file.xls', sheetname='Sheet1')
```

*Hay muchas opciones de análisis para [read\\_excel](#) (similares a las opciones en `read_csv` .*

```
pd.read_excel('path_to_file.xls',
              sheetname='Sheet1', header=[0, 1, 2],
              skiprows=3, index_col=0) # etc.
```

## Prueba de read\_csv

```
import pandas as pd
import io
```

```
temp=u"""index; header1; header2; header3
1; str_data; 12; 1.4
3; str_data; 22; 42.33
4; str_data; 2; 3.44
2; str_data; 43; 43.34
7; str_data; 25; 23.32"""
#after testing replace io.StringIO(temp) to filename
df = pd.read_csv(io.StringIO(temp),
                  sep = ';',
                  index_col = 0,
                  skip_blank_lines = True)

print (df)
```

	header1	header2	header3
index			
1	str_data	12	1.40
3	str_data	22	42.33
4	str_data	2	3.44
2	str_data	43	43.34
7	str_data	25	23.32

## Lista de comprensión

Todos los archivos están en `files` carpeta. Primero crea una lista de DataFrames y luego `concat` :

```
import pandas as pd
import glob

#a.csv
#a,b
#1,2
#5,8

#b.csv
#a,b
#9,6
#6,4

#c.csv
#a,b
#4,3
#7,0

files = glob.glob('files/*.csv')
dfs = [pd.read_csv(fp) for fp in files]
```

```
#duplicated index inherited from each Dataframe
df = pd.concat(dfs)
print (df)
```

	a	b
0	1	2
1	5	8
0	9	6
1	6	4
0	4	3
1	7	0

```
#'reseting' index
df = pd.concat(dfs, ignore_index=True)
```

```

print (df)
   a  b
0  1  2
1  5  8
2  9  6
3  6  4
4  4  3
5  7  0
#concat by columns
df1 = pd.concat(dfs, axis=1)
print (df1)
   a  b  a  b  a  b
0  1  2  9  6  4  3
1  5  8  6  4  7  0
#reset column names
df1 = pd.concat(dfs, axis=1, ignore_index=True)
print (df1)
   0  1  2  3  4  5
0  1  2  9  6  4  3
1  5  8  6  4  7  0

```

## Leer en trozos

```

import pandas as pd

chunksize = [n]
for chunk in pd.read_csv(filename, chunksize=chunksize):
    process(chunk)
    delete(chunk)

```

## Guardar en archivo CSV

### Guardar con los parámetros por defecto:

```
df.to_csv(file_name)
```

### Escribir columnas específicas:

```
df.to_csv(file_name, columns =['col'])
```

### El delimitador de falla es ',' - para cambiarlo:

```
df.to_csv(file_name, sep="|")
```

### Escribir sin el encabezado:

```
df.to_csv(file_name, header=False)
```

### Escribir con un encabezado dado:

```
df.to_csv(file_name, header = ['A','B','C',...])
```

**Para usar una codificación específica (por ejemplo, 'utf-8') use el argumento de codificación:**

`df.to_csv (nombre_archivo, codificación = 'utf-8')`

## Análisis de columnas de fecha con `read_csv`

Las fechas siempre tienen un formato diferente, se pueden analizar utilizando una función específica `parse_dates`.

Esta *entrada.csv*:

```
2016 06 10 20:30:00    foo
2016 07 11 19:45:30    bar
2013 10 12  4:30:00    foo
```

Se puede analizar de esta manera:

```
mydateparser = lambda x: pd.datetime.strptime(x, "%Y %m %d %H:%M:%S")
df = pd.read_csv("file.csv", sep='\t', names=['date_column', 'other_column'],
parse_dates=['date_column'], date_parser=mydateparser)
```

El argumento `parse_dates` es la columna a analizar  
`date_parser` es la función del analizador

## Lea y combine varios archivos CSV (con la misma estructura) en un DF

```
import os
import glob
import pandas as pd

def get_merged_csv(flist, **kwargs):
    return pd.concat([pd.read_csv(f, **kwargs) for f in flist], ignore_index=True)

path = 'C:/Users/csvfiles'
fmask = os.path.join(path, '*mask*.csv')

df = get_merged_csv(glob.glob(fmask), index_col=None, usecols=['col1', 'col3'])

print(df.head())
```

Si desea combinar archivos CSV horizontalmente (agregando columnas), use `axis=1` cuando llame a la función `pd.concat()`:

```
def merged_csv_horizontally(flist, **kwargs):
    return pd.concat([pd.read_csv(f, **kwargs) for f in flist], axis=1)
```

## Leyendo el archivo cvs en un marco de datos pandas cuando no hay una fila



## de encabezado

Si el archivo no contiene una fila de encabezado,

Expediente:

```
1;str_data;12;1.4
3;str_data;22;42.33
4;str_data;2;3.44
2;str_data;43;43.34

7; str_data; 25; 23.32
```

puede utilizar los `names` palabras clave para proporcionar nombres de columna:

```
df = pandas.read_csv('data_file.csv', sep=';', index_col=0,
                     skip_blank_lines=True, names=['a', 'b', 'c'])
```

```
df
Out:
      a  b  c
1  str_data  12  1.40
3  str_data  22  42.33
4  str_data  2  3.44
2  str_data  43  43.34
7  str_data  25  23.32
```

## Usando HDFStore

```
import string
import numpy as np
import pandas as pd
```

# Generar muestra DF con diversos tipos.

```
df = pd.DataFrame({
    'int32': np.random.randint(0, 10**6, 10),
    'int64': np.random.randint(10**7, 10**9, 10).astype(np.int64)*10,
    'float': np.random.rand(10),
    'string': np.random.choice([c*10 for c in string.ascii_uppercase], 10),
})
```

```
In [71]: df
```

```
Out[71]:
```

	float	int32	int64	string
0	0.649978	848354	5269162190	DDDDDDDDDD
1	0.346963	490266	6897476700	OOOOOOOOOO
2	0.035069	756373	6711566750	ZZZZZZZZZZ
3	0.066692	957474	9085243570	FFFFFFFFFF
4	0.679182	665894	3750794810	MMMMMMMMMM
5	0.861914	630527	6567684430	TTTTTTTTTT
6	0.697691	825704	8005182860	FFFFFFFFFF
7	0.474501	942131	4099797720	QQQQQQQQQQ

```
8 0.645817 951055 8065980030 VVVVVVVVVV
9 0.083500 349709 7417288920 EEEEEEEEE
```

## hacer un DF más grande (10 \* 100.000 = 1.000.000 filas)

```
df = pd.concat([df] * 10**5, ignore_index=True)
```

## crear (o abrir un archivo HDFStore existente)

```
store = pd.HDFStore('d:/temp/example.h5')
```

## guarde nuestro marco de datos en el archivo h5 (HDFStore), indexando [int32, int64, string] columnas:

```
store.append('store_key', df, data_columns=['int32','int64','string'])
```

## Mostrar detalles de HDFStore

```
In [78]: store.get_storer('store_key').table
Out[78]:
/store_key/table (Table(10,)) ''
description := {
  "index": Int64Col(shape=(), dflt=0, pos=0),
  "values_block_0": Float64Col(shape=(1,), dflt=0.0, pos=1),
  "int32": Int32Col(shape=(), dflt=0, pos=2),
  "int64": Int64Col(shape=(), dflt=0, pos=3),
  "string": StringCol(itemsize=10, shape=(), dflt=b'', pos=4)}
byteorder := 'little'
chunkshape := (1724,)
autoindex := True
colindexes := {
  "index": Index(6, medium, shuffle, zlib(1)).is_csi=False,
  "int32": Index(6, medium, shuffle, zlib(1)).is_csi=False,
  "string": Index(6, medium, shuffle, zlib(1)).is_csi=False,
  "int64": Index(6, medium, shuffle, zlib(1)).is_csi=False}
```

## mostrar columnas indexadas

```
In [80]: store.get_storer('store_key').table.colindexes
Out[80]:
{
  "int32": Index(6, medium, shuffle, zlib(1)).is_csi=False,
  "index": Index(6, medium, shuffle, zlib(1)).is_csi=False,
  "string": Index(6, medium, shuffle, zlib(1)).is_csi=False,
  "int64": Index(6, medium, shuffle, zlib(1)).is_csi=False}
```

---

## cerrar (vaciar al disco) nuestro archivo de tienda

```
store.close()
```

### Lea el registro de acceso de Nginx (varias cotillas)

Para varias cotillas use expresiones regulares en lugar de sep:

```
df = pd.read_csv(log_file,
                 sep=r'\s(?:[^\s]*"[^"]*"|' + "[^\s]*$)(?![^\s]*\\)',",
                 engine='python',
                 usecols=[0, 3, 4, 5, 6, 7, 8],
                 names=['ip', 'time', 'request', 'status', 'size', 'referer', 'user_agent'],
                 na_values='-',
                 header=None
                 )
```

Lea Herramientas de Pandas IO (leer y guardar conjuntos de datos) en línea:

<https://riptutorial.com/es/pandas/topic/2896/herramientas-de-pandas-io-leer-y-guardar-conjuntos-de-datos->

# Capítulo 18: Indexación booleana de marcos de datos

## Introducción

Acceso a las filas en un marco de datos utilizando los objetos del indexador `.ix`, `.loc`, `.iloc` y cómo se diferencia de usar una máscara booleana.

## Examples

### Accediendo a un DataFrame con un índice booleano

Este será nuestro marco de datos de ejemplo:

```
df = pd.DataFrame({"color": ['red', 'blue', 'red', 'blue']},
                  index=[True, False, True, False])

   color
True  red
False blue
True  red
False blue
```

#### Accediendo con `.loc`

```
df.loc[True]
   color
True  red
True  red
```

#### Accediendo con `.iloc`

```
df.iloc[True]
>> TypeError

df.iloc[1]
   color  blue
dtype: object
```

Es importante tener en cuenta que las versiones anteriores de los pandas no distinguían entre la entrada booleana y la de enteros, por lo que `.iloc[True]` devolvería lo mismo que `.iloc[1]`

#### Accediendo con `.ix`

```
df.ix[True]
   color
True  red
```

```
True    red

df.ix[1]
color    blue
dtype: object
```

Como puedes ver, `.ix` tiene dos comportamientos. Esta es una muy mala práctica en el código y, por lo tanto, debe evitarse. Por favor use `.iloc` o `.loc` para ser más explícito.

## Aplicar una máscara booleana a un marco de datos

Este será nuestro marco de datos de ejemplo:

```
   color    name  size
0   red     rose   big
1  blue  violet   big
2   red   tulip  small
3  blue harebell  small
```

Usando el `__getitem__` mágico `__getitem__` o `[]`. Dándole una lista de Verdadero y Falso de la misma longitud que el marco de datos le dará:

```
df[[True, False, True, False]]
   color    name  size
0   red     rose   big
2   red   tulip  small
```

## Datos de enmascaramiento basados en el valor de la columna

Este será nuestro marco de datos de ejemplo:

```
   color    name  size
0   red     rose   big
1  blue  violet  small
2   red   tulip  small
3  blue harebell  small
```

Accediendo a una sola columna desde un marco de datos, podemos usar una comparación simple `==` para comparar cada elemento de la columna con la variable dada, produciendo un `pd.Series` de Verdadero y Falso

```
df['size'] == 'small'
0    False
1     True
2     True
3     True
Name: size, dtype: bool
```

Esta `pd.Series` es una extensión de un `np.array` que es una extensión de una `list` simple. Por lo tanto, podemos entregar esto al `__getitem__` o `[]` accessor como en el ejemplo anterior.

```
size_small_mask = df['size'] == 'small'
df[size_small_mask]
   color    name  size
1  blue  violet  small
2   red   tulip  small
3  blue harebell  small
```

## Datos de enmascaramiento basados en el valor del índice

Este será nuestro marco de datos de ejemplo:

```
   color  size
name
rose    red   big
violet  blue  small
tulip    red  small
harebell blue  small
```

Podemos crear una máscara basada en los valores del índice, al igual que en un valor de columna.

```
rose_mask = df.index == 'rose'
df[rose_mask]
   color size
name
rose   red  big
```

Pero hacer esto es *casi* lo mismo que

```
df.loc['rose']
color    red
size     big
Name: rose, dtype: object
```

La diferencia importante es que cuando `.loc` solo encuentra una fila en el índice que coincide, devolverá un `pd.Series`, si encuentra más filas que coinciden, devolverá un `pd.DataFrame`. Esto hace que este método sea bastante inestable.

Este comportamiento puede controlarse dando a `.loc` una lista de una sola entrada. Esto lo obligará a devolver un marco de datos.

```
df.loc[['rose']]
   color  size
name
rose    red  big
```

Lea [Indexación booleana de marcos de datos en línea](https://riptutorial.com/es/pandas/topic/9589/indexacion-booleana-de-marcos-de-datos):

<https://riptutorial.com/es/pandas/topic/9589/indexacion-booleana-de-marcos-de-datos>

# Capítulo 19: Indexación y selección de datos.

## Examples

### Seleccionar columna por etiqueta

```
# Create a sample DF
df = pd.DataFrame(np.random.randn(5, 3), columns=list('ABC'))

# Show DF
df

```

	A	B	C
0	-0.467542	0.469146	-0.861848
1	-0.823205	-0.167087	-0.759942
2	-1.508202	1.361894	-0.166701
3	0.394143	-0.287349	-0.978102
4	-0.160431	1.054736	-0.785250

```

# Select column using a single label, 'A'
df['A']

```

0	-0.467542
1	-0.823205
2	-1.508202
3	0.394143
4	-0.160431

```

# Select multiple columns using an array of labels, ['A', 'C']
df[['A', 'C']]

```

	A	C
0	-0.467542	-0.861848
1	-0.823205	-0.759942
2	-1.508202	-0.166701
3	0.394143	-0.978102
4	-0.160431	-0.785250

Detalles adicionales en: <http://pandas.pydata.org/pandas-docs/version/0.18.0/indexing.html#selection-by-label>

### Seleccionar por posición

El `iloc` (abreviatura de *ubicación de enteros*) permite seleccionar las filas de un marco de datos según su índice de posición. De esta manera, se pueden dividir los marcos de datos como se hace con la división de la lista de Python.

```
df = pd.DataFrame([[11, 22], [33, 44], [55, 66]], index=list("abc"))

df

```

```
# Out:
#      0    1
# a   11   22
# b   33   44
# c   55   66
```

```
df.iloc[0] # the 0th index (row)
# Out:
# 0    11
# 1    22
# Name: a, dtype: int64

df.iloc[1] # the 1st index (row)
# Out:
# 0    33
# 1    44
# Name: b, dtype: int64

df.iloc[:2] # the first 2 rows
#      0    1
# a   11   22
# b   33   44

df[::-1] # reverse order of rows
#      0    1
# c   55   66
# b   33   44
# a   11   22
```

La ubicación de la fila se puede combinar con la ubicación de la columna

```
df.iloc[:, 1] # the 1st column
# Out[15]:
# a    22
# b    44
# c    66
# Name: 1, dtype: int64
```

Ver también: [Selección por Posición](#).

## Rebanar con etiquetas

Cuando se usan etiquetas, tanto el inicio como la parada se incluyen en los resultados.

```
import pandas as pd
import numpy as np
np.random.seed(5)
df = pd.DataFrame(np.random.randint(100, size=(5, 5)), columns = list("ABCDE"),
                  index = ["R" + str(i) for i in range(5)])

# Out:
#      A    B    C    D    E
# R0  99   78   61   16   73
# R1   8   62   27   30   80
# R2   7   76   15   53   80
# R3  27   44   77   75   65
# R4  47   30   84   86   18
```

Filas R0 a R2 :

```
df.loc['R0':'R2']
# Out:
```



```
#      A    B    C    D    E
# R0   9   41   62    1   82
# R1  16   78    5   58    0
# R2  80    4   36   51   27
```

Observe cómo `loc` diferencia de `iloc` porque `iloc` excluye el índice final

```
df.loc['R0':'R2'] # rows labelled R0, R1, R2
# Out:
#      A    B    C    D    E
# R0   9   41   62    1   82
# R1  16   78    5   58    0
# R2  80    4   36   51   27

# df.iloc[0:2] # rows indexed by 0, 1
#      A    B    C    D    E
# R0  99   78   61   16   73
# R1   8   62   27   30   80
```

Columnas `C` a `E`:

```
df.loc[:, 'C':'E']
# Out:
#      C    D    E
# R0  62    1   82
# R1   5   58    0
# R2  36   51   27
# R3  68   38   83
# R4   7   30   62
```

## Posición mixta y selección basada en etiqueta

Marco de datos:

```
import pandas as pd
import numpy as np
np.random.seed(5)
df = pd.DataFrame(np.random.randint(100, size=(5, 5)), columns = list("ABCDE"),
                  index = ["R" + str(i) for i in range(5)])

df
Out[12]:
      A    B    C    D    E
R0  99   78   61   16   73
R1   8   62   27   30   80
R2   7   76   15   53   80
R3  27   44   77   75   65
R4  47   30   84   86   18
```

Seleccione filas por posición y columnas por etiqueta:

```
df.ix[1:3, 'C':'E']
Out[19]:
```

	C	D	E
R1	5	58	0
R2	36	51	27

Si el índice es entero, `.ix` utilizará etiquetas en lugar de posiciones:

```
df.index = np.arange(5, 10)

df
Out[22]:
   A  B  C  D  E
5  9 41 62  1 82
6 16 78  5 58  0
7 80  4 36 51 27
8 31  2 68 38 83
9 19 18  7 30 62

#same call returns an empty DataFrame because now the index is integer
df.ix[1:3, 'C':'E']
Out[24]:
Empty DataFrame
Columns: [C, D, E]
Index: []
```

## Indexación booleana

Uno puede seleccionar filas y columnas de un marco de datos utilizando matrices booleanas.

```
import pandas as pd
import numpy as np
np.random.seed(5)
df = pd.DataFrame(np.random.randint(100, size=(5, 5)), columns = list("ABCDE"),
                  index = ["R" + str(i) for i in range(5)])

print (df)
#      A  B  C  D  E
# R0  99  78  61  16  73
# R1   8  62  27  30  80
# R2   7  76  15  53  80
# R3  27  44  77  75  65
# R4  47  30  84  86  18
```

```
mask = df['A'] > 10
print (mask)
# R0      True
# R1     False
# R2     False
# R3      True
# R4      True
# Name: A, dtype: bool

print (df[mask])
#      A  B  C  D  E
# R0  99  78  61  16  73
# R3  27  44  77  75  65
# R4  47  30  84  86  18

print (df.ix[mask, 'C'])
```

```
# R0      61
# R3      77
# R4      84
# Name: C, dtype: int32

print(df.ix[mask, ['C', 'D']])
#          C    D
# R0     61   16
# R3     77   75
# R4     84   86
```

Más en la [documentación de los pandas](#) .

**Filtrado de columnas (selección de "interesante", eliminación innecesaria, uso de RegEx, etc.)**

## generar muestra DF

```
In [39]: df = pd.DataFrame(np.random.randint(0, 10, size=(5, 6)),
columns=['a10', 'a20', 'a25', 'b', 'c', 'd'])
```

```
In [40]: df
```

```
Out[40]:
```

	a10	a20	a25	b	c	d
0	2	3	7	5	4	7
1	3	1	5	7	2	6
2	7	4	9	0	8	7
3	5	8	8	9	6	8
4	8	1	0	4	4	9

## mostrar columnas que contengan la letra 'a'

```
In [41]: df.filter(like='a')
```

```
Out[41]:
```

	a10	a20	a25
0	2	3	7
1	3	1	5
2	7	4	9
3	5	8	8
4	8	1	0

## muestre las columnas usando el filtro RegEx

(b|c|d) = b **O** c **O** d :

```
In [42]: df.filter(regex='(b|c|d)')
```

```
Out[42]:
```

	b	c	d
0	5	4	7

```
1  7  2  6
2  0  8  7
3  9  6  8
4  4  4  9
```

**mostrar todas las columnas excepto los que empiezan por `a` (en otras palabras remove / dejar todas las columnas satisfacer RegEx dado)**

```
In [43]: df.ix[:, ~df.columns.str.contains('^a')]
Out[43]:
```

	b	c	d
0	5	4	7
1	7	2	6
2	0	8	7
3	9	6	8
4	4	4	9

**Filtrar / seleccionar filas usando el método `.query()`**

```
import pandas as pd
```

**generar DF aleatorio**

```
df = pd.DataFrame(np.random.randint(0,10,size=(10, 3)), columns=list('ABC'))

In [16]: print(df)
```

	A	B	C
0	4	1	4
1	0	2	0
2	7	8	8
3	2	1	9
4	7	3	8
5	4	0	7
6	1	5	5
7	6	7	8
8	6	7	3
9	6	4	5

**seleccione las filas donde los valores en la columna `A > 2` y los valores en la columna `B < 5`**

```
In [18]: df.query('A > 2 and B < 5')
Out[18]:
```

	A	B	C
0	4	1	4
4	7	3	8
5	4	0	7
9	6	4	5

## utilizando el método `.query()` con variables para filtrar

```
In [23]: B_filter = [1,7]

In [24]: df.query('B == @B_filter')
Out[24]:
   A  B  C
0  4  1  4
3  2  1  9
7  6  7  8
8  6  7  3

In [25]: df.query('@B_filter in B')
Out[25]:
   A  B  C
0  4  1  4
```

## Rebanado Dependiente del Camino

Puede ser necesario atravesar los elementos de una serie o las filas de un marco de datos de manera que el siguiente elemento o la siguiente fila dependa del elemento o fila previamente seleccionado. Esto se llama dependencia de ruta.

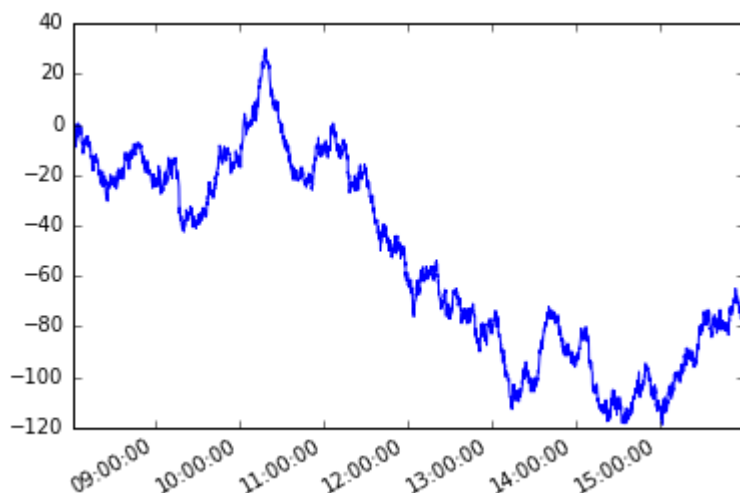
Considere las siguientes series de tiempo `s` con frecuencia irregular.

```
#starting python community conventions
import numpy as np
import pandas as pd

# n is number of observations
n = 5000

day = pd.to_datetime(['2013-02-06'])
# irregular seconds spanning 28800 seconds (8 hours)
seconds = np.random.rand(n) * 28800 * pd.Timedelta(1, 's')
# start at 8 am
start = pd.offsets.Hour(8)
# irregular timeseries
tidx = day + start + seconds
tidx = tidx.sort_values()

s = pd.Series(np.random.randn(n), tidx, name='A').cumsum()
s.plot();
```



Asumamos una condición dependiente del camino. Comenzando con el primer miembro de la serie, quiero tomar cada elemento subsiguiente de manera que la diferencia absoluta entre ese elemento y el elemento actual sea mayor o igual que  $x$ .

Vamos a resolver este problema utilizando generadores de pitón.

### Función generadora

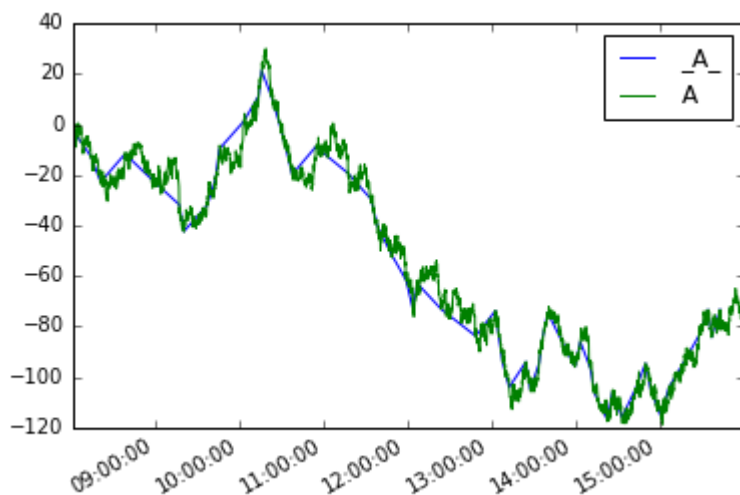
```
def mover(s, move_size=10):
    """Given a reference, find next value with
    an absolute difference >= move_size"""
    ref = None
    for i, v in s.iteritems():
        if ref is None or (abs(ref - v) >= move_size):
            yield i, v
            ref = v
```

Entonces podemos definir una nueva serie de `moves` como tal.

```
moves = pd.Series({i:v for i, v in mover(s, move_size=10)},
                  name='_{}_{}'.format(s.name))
```

### Trazando ambos

```
moves.plot(legend=True)
s.plot(legend=True)
```

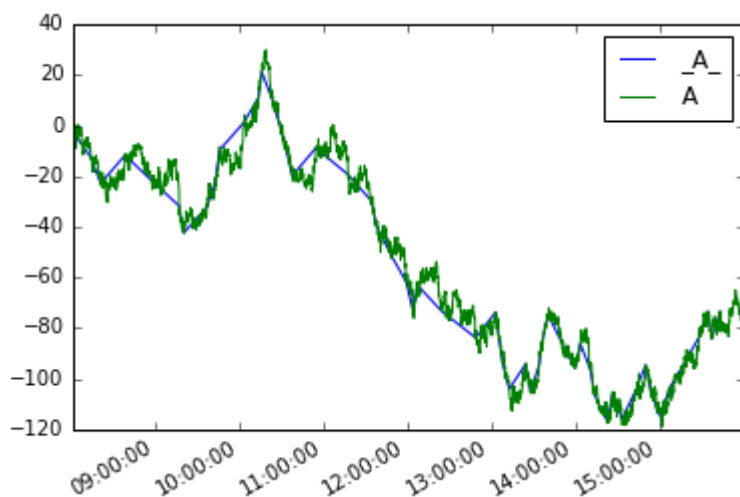


El análogo para los marcos de datos sería:

```
def mover_df(df, col, move_size=2):
    ref = None
    for i, row in df.iterrows():
        if ref is None or (abs(ref - row.loc[col]) >= move_size):
            yield row
            ref = row.loc[col]

df = s.to_frame()
moves_df = pd.concat(mover_df(df, 'A', 10), axis=1).T

moves_df.A.plot(label='_A_', legend=True)
df.A.plot(legend=True)
```



## Obtener las primeras / últimas n filas de un marco de datos

Para ver los primeros o últimos registros de un marco de datos, puede usar los métodos `head` y `tail`

Para devolver las primeras n filas, use `DataFrame.head([n])`

```
df.head(n)
```

Para devolver las últimas n filas, use `DataFrame.tail([n])`

```
df.tail(n)
```

Sin el argumento n, estas funciones devuelven 5 filas.

Tenga en cuenta que la notación de corte para `head / tail` sería:

```
df[:10] # same as df.head(10)
df[-10:] # same as df.tail(10)
```

## Seleccionar filas distintas en el marco de datos

Dejar

```
df = pd.DataFrame({'col_1': ['A', 'B', 'A', 'B', 'C'], 'col_2': [3, 4, 3, 5, 6]})
df
# Output:
#   col_1  col_2
# 0     A      3
# 1     B      4
# 2     A      3
# 3     B      5
# 4     C      6
```

Para obtener los valores distintos en `col_1` puede usar `Series.unique()`

```
df['col_1'].unique()
# Output:
# array(['A', 'B', 'C'], dtype=object)
```

Pero `Series.unique()` solo funciona para una sola columna.

Para simular el *col\_1* único seleccionado, *col\_2* de SQL, puede usar `DataFrame.drop_duplicates()` :

```
df.drop_duplicates()
#   col_1  col_2
# 0     A      3
# 1     B      4
# 3     B      5
# 4     C      6
```

Esto te dará todas las filas únicas en el marco de datos. Así que si

```
df = pd.DataFrame({'col_1': ['A', 'B', 'A', 'B', 'C'], 'col_2': [3, 4, 3, 5, 6],
                    'col_3': [0, 0.1, 0.2, 0.3, 0.4]})
df
# Output:
#   col_1  col_2  col_3
# 0     A      3     0.0
# 1     B      4     0.1
# 2     A      3     0.2
```



```
# 3      B      5      0.3
# 4      C      6      0.4

df.drop_duplicates()
#   col_1  col_2  col_3
# 0     A      3      0.0
# 1     B      4      0.1
# 2     A      3      0.2
# 3     B      5      0.3
# 4     C      6      0.4
```

Para especificar las columnas a considerar al seleccionar registros únicos, páselos como argumentos

```
df = pd.DataFrame({'col_1': ['A', 'B', 'A', 'B', 'C'], 'col_2': [3, 4, 3, 5, 6],
                  'col_3': [0, 0.1, 0.2, 0.3, 0.4]})
df.drop_duplicates(['col_1', 'col_2'])
# Output:
#   col_1  col_2  col_3
# 0     A      3      0.0
# 1     B      4      0.1
# 3     B      5      0.3
# 4     C      6      0.4

# skip last column
# df.drop_duplicates(['col_1', 'col_2'])[['col_1', 'col_2']]
#   col_1  col_2
# 0     A      3
# 1     B      4
# 3     B      5
# 4     C      6
```

Fuente: [¿Cómo "seleccionar distintas" en varias columnas de marcos de datos en pandas?](#) .

## Filtrar las filas con datos faltantes (NaN, Ninguno, NaT)

Si tiene un marco de datos con datos faltantes ( `NaN` , `pd.NaT` , `None` ) puede filtrar filas incompletas

```
df = pd.DataFrame([[0, 1, 2, 3],
                  [None, 5, None, pd.NaT],
                  [8, None, 10, None],
                  [11, 12, 13, pd.NaT]], columns=list('ABCD'))

df
# Output:
#   A  B  C  D
# 0  0  1  2  3
# 1 NaN  5 NaN NaT
# 2  8 NaN 10 None
# 3 11 12 13 NaT
```

`DataFrame.dropna` elimina todas las filas que contienen al menos un campo con datos faltantes

```
df.dropna()
# Output:
#   A  B  C  D
# 0  0  1  2  3
```

Para soltar las filas que faltan datos en las columnas especificadas, use el `subset`

```
df.dropna(subset=['C'])  
# Output:  
#      A    B    C    D  
# 0     0    1    2     3  
# 2     8 NaN  10  None  
# 3    11  12  13   NaT
```

Use la opción `inplace = True` para el reemplazo en el lugar con el marco filtrado.

Lea [Indexación y selección de datos](https://riptutorial.com/es/pandas/topic/1751/indexacion-y-seleccion-de-datos-). en línea:

<https://riptutorial.com/es/pandas/topic/1751/indexacion-y-seleccion-de-datos->

# Capítulo 20: IO para Google BigQuery

## Examples

### Lectura de datos de BigQuery con credenciales de cuenta de usuario

```
In [1]: import pandas as pd
```

Para ejecutar una consulta en BigQuery necesita tener su propio proyecto de BigQuery. Podemos solicitar algunos datos de muestra públicos:

```
In [2]: data = pd.read_gbq('''SELECT title, id, num_characters
...:                        FROM [publicdata:samples.wikipedia]
...:                        LIMIT 5''',
...:                        project_id='<your-project-id>')
```

Esto imprimirá:

Your browser has been opened to visit:

[https://accounts.google.com/o/oauth2/v2/auth...\[loooong url cutted\]](https://accounts.google.com/o/oauth2/v2/auth...[loooong url cutted])

If your browser is on a different machine then exit and re-run this application with the command-line parameter

```
--noauth_local_webserver
```

Si está operando desde una máquina local, entonces aparecerá el navegador. Después de otorgar privilegios, los pandas continuarán con la salida:

```
Authentication successful.
Requesting query... ok.
Query running...
Query done.
Processed: 13.8 Gb

Retrieving results...
Got 5 rows.

Total time taken 1.5 s.
Finished at 2016-08-23 11:26:03.
```

Resultado:

```
In [3]: data
Out[3]:
```

	title	id	num_characters
0	Fusidic acid	935328	1112
1	Clark Air Base	426241	8257
2	Watergate scandal	52382	25790
3	2005	35984	75813

Como efecto secundario, los pandas crearán el archivo json `bigquery_credentials.dat` que le permitirá ejecutar más consultas sin necesidad de otorgar privilegios:

```
In [9]: pd.read_gbq('SELECT count(1) cnt FROM [publicdata:samples.wikipedia]'
              , project_id='<your-project-id>')
Requesting query... ok.
[rest of output cutted]

Out[9]:
      cnt
0  313797035
```

## Lectura de datos de BigQuery con credenciales de cuenta de servicio

Si ha creado una [cuenta de servicio](#) y tiene un archivo json de clave privada para ella, puede usar este archivo para autenticarse con pandas

```
In [5]: pd.read_gbq(''SELECT corpus, sum(word_count) words
                  FROM [bigquery-public-data:samples.shakespeare]
                  GROUP BY corpus
                  ORDER BY words desc
                  LIMIT 5''
              , project_id='<your-project-id>'
              , private_key='<private key json contents or file path>')
Requesting query... ok.
[rest of output cutted]

Out[5]:
   corpus  words
0   hamlet  32446
1 kingrichardiii  31868
2   coriolanus  29535
3   cymbeline  29231
4  2kinghenryiv  28241
```

Lea IO para Google BigQuery en línea: <https://riptutorial.com/es/pandas/topic/5610/io-para-google-bigquery>

---

# Capítulo 21: JSON

## Examples

### Leer json

---

**puede pasar la cadena del json o una ruta de archivo a un archivo con json válido**

```
In [99]: pd.read_json('["A": 1, "B": 2], {"A": 3, "B": 4}']  
Out[99]:  
   A  B  
0  1  2  
1  3  4
```

Como alternativa para conservar la memoria:

```
with open('test.json') as f:  
    data = pd.DataFrame(json.loads(line) for line in f)
```

## Marco de datos en JSON anidado como en los archivos flare.js utilizados en D3.js

```
def to_flare_json(df, filename):  
    """Convert dataframe into nested JSON as in flare files used for D3.js"""  
    flare = dict()  
    d = {"name": "flare", "children": []}  
  
    for index, row in df.iterrows():  
        parent = row[0]  
        child = row[1]  
        child_size = row[2]  
  
        # Make a list of keys  
        key_list = []  
        for item in d['children']:  
            key_list.append(item['name'])  
  
        #if 'parent' is NOT a key in flare.JSON, append it  
        if not parent in key_list:  
            d['children'].append({"name": parent, "children": [{"value": child_size, "name":  
child}]}))  
        # if parent IS a key in flare.json, add a new child to it  
        else:  
            d['children'][key_list.index(parent)]['children'].append({"value": child_size,  
"name": child})  
        flare = d  
    # export the final result to a json file  
    with open(filename + '.json', 'w') as outfile:
```

```
json.dump(flare, outfile, indent=4)
return ("Done")
```

## Lee JSON del archivo

Contenido de file.json (un objeto JSON por línea):

```
{"A": 1, "B": 2}
{"A": 3, "B": 4}
```

Cómo leer directamente desde un archivo local:

```
pd.read_json('file.json', lines=True)
# Output:
#    A  B
# 0  1  2
# 1  3  4
```

Lea JSON en línea: <https://riptutorial.com/es/pandas/topic/4752/json>

---

# Capítulo 22: Leer MySQL a DataFrame

## Examples

### Usando sqlalchemy y PyMySQL

```
from sqlalchemy import create_engine

cnx = create_engine('mysql+pymysql://username:password@server:3306/database').connect()
sql = 'select * from mytable'
df = pd.read_sql(sql, cnx)
```

### Para leer mysql a dataframe, en caso de gran cantidad de datos

Para obtener grandes cantidades de datos, podemos usar generadores en pandas y cargar datos en trozos.

```
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.engine.url import URL

# sqlalchemy engine
engine = create_engine(URL(
    drivename="mysql"
    username="user",
    password="password"
    host="host"
    database="database"
))

conn = engine.connect()

generator_df = pd.read_sql(sql=query, # mysql query
                           con=conn,
                           chunksize=chunksize) # size you want to fetch each time

for dataframe in generator_df:
    for row in dataframe:
        pass # whatever you want to do
```

Lea Leer MySQL a DataFrame en línea: <https://riptutorial.com/es/pandas/topic/8809/leer-mysql-a-dataframe>

# Capítulo 23: Leer SQL Server a Dataframe

## Examples

### Utilizando pyodbc

```
import pandas.io.sql
import pyodbc
import pandas as pd
```

### Especificar los parametros

```
# Parameters
server = 'server_name'
db = 'database_name'
UID = 'user_id'
```

### Crear la conexión

```
# Create the connection
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=' + server + ';DATABASE=' + db + '; UID = '
+ UID + '; PWD = ' + UID + 'Trusted_Connection=yes')
```

### Consulta en marco de datos pandas

```
# Query into dataframe
df= pandas.io.sql.read_sql('sql_query_string', conn)
```

### Usando pyodbc con bucle de conexión

```
import os, time
import pyodbc
import pandas.io.sql as pdsq

def todf(dsn='yourdsn', uid=None, pwd=None, query=None, params=None):
    ''' if `query` is not an actual query but rather a path to a text file
        containing a query, read it in instead '''
    if query.endswith('.sql') and os.path.exists(query):
        with open(query, 'r') as fin:
            query = fin.read()

    connstr = "DSN={};UID={};PWD={}".format(dsn, uid, pwd)
    connected = False
    while not connected:
        try:
            with pyodbc.connect(connstr, autocommit=True) as con:
                cur = con.cursor()
                if params is not None: df = pdsq.read_sql(query, con,
                                                            params=params)
                else: df = pdsq.read_sql(query, con)
                cur.close()
```



```
        break
    except pyodbc.OperationalError:
        time.sleep(60) # one minute could be changed
    return df
```

Lea Leer SQL Server a Dataframe en línea: <https://riptutorial.com/es/pandas/topic/2176/leer-sql-server-a-dataframe>

# Capítulo 24: Leyendo archivos en pandas DataFrame

## Examples

### Leer la tabla en DataFrame

### Archivo de tabla con encabezado, pie de página, nombres de fila y columna de índice:

archivo: table.txt

```
This is a header that discusses the table file
to show space in a generic table file

index  name      occupation
1      Alice    Salesman
2      Bob      Engineer
3      Charlie  Janitor

This is a footer because your boss does not understand data files
```

código:

```
import pandas as pd
# index_col=0 tells pandas that column 0 is the index and not data
pd.read_table('table.txt', delim_whitespace=True, skiprows=3, skipfooter=2, index_col=0)
```

salida:

```
      name occupation
index
1      Alice    Salesman
2       Bob    Engineer
3    Charlie    Janitor
```

### Archivo de tabla sin nombres de fila o índice:

archivo: table.txt

```
Alice    Salesman
Bob       Engineer
Charlie   Janitor
```

código:

```
import pandas as pd
pd.read_table('table.txt', delim_whitespace=True, names=['name', 'occupation'])
```

**salida:**

	name	occupation
0	Alice	Salesman
1	Bob	Engineer
2	Charlie	Janitor

Todas las opciones se pueden encontrar en la documentación de los pandas [aquí](#).

## Leer archivo CSV

## Datos con encabezado, separados por punto y coma en lugar de comas.

**archivo: table.csv**

```
index;name;occupation
1;Alice;Saleswoman
2;Bob;Engineer
3;Charlie;Janitor
```

**código:**

```
import pandas as pd
pd.read_csv('table.csv', sep=';', index_col=0)
```

**salida :**

	name	occupation
index		
1	Alice	Salesman
2	Bob	Engineer
3	Charlie	Janitor

## Tabla sin nombres de filas o índice y comas como separadores

**archivo: table.csv**

```
Alice,Saleswoman
Bob,Engineer
Charlie,Janitor
```

**código:**

```
import pandas as pd
pd.read_csv('table.csv', names=['name','occupation'])
```

### salida:

```
   name occupation
0  Alice  Salesman
1   Bob   Engineer
2 Charlie   Janitor
```

Puede encontrar más información en la página de documentación de [read\\_csv](#).

## Recopila datos de la hoja de cálculo de Google en el marco de datos de pandas

A veces necesitamos recopilar datos de las hojas de cálculo de google. Podemos usar las bibliotecas **gsread** y **oauth2client** para recopilar datos de las hojas de cálculo de Google. Aquí hay un ejemplo para recopilar datos:

### Código:

```
from __future__ import print_function
import gsread
from oauth2client.client import SignedJwtAssertionCredentials
import pandas as pd
import json

scope = ['https://spreadsheets.google.com/feeds']

credentials = ServiceAccountCredentials.from_json_keyfile_name('your-authorization-file.json',
scope)

gc = gsread.authorize(credentials)

work_sheet = gc.open_by_key("spreadsheet-key-here")
sheet = work_sheet.sheet1
data = pd.DataFrame(sheet.get_all_records())

print(data.head())
```

Lea [Leyendo archivos en pandas DataFrame en línea](#):

<https://riptutorial.com/es/pandas/topic/1988/leyendo-archivos-en-pandas-dataframe>

# Capítulo 25: Making Pandas Play Nice con tipos de datos nativos de Python

## Examples

### Mover datos de pandas a estructuras nativas Python y Numpy

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0], 'C': ['a', 'b', 'c'],  
                          'D': [True, False, True]})
```

```
In [2]: df
```

```
Out[2]:
```

	A	B	C	D
0	1	1.0	a	True
1	2	2.0	b	False
2	3	3.0	c	True

Obtención de una lista de python de una serie:

```
In [3]: df['A'].tolist()
```

```
Out[3]: [1, 2, 3]
```

Los DataFrames no tienen un método `tolist()` . Intentarlo da como resultado un `AttributeError`:

```
In [4]: df.tolist()
```

```
-----  
AttributeError                                Traceback (most recent call last)  
<ipython-input-4-fc6763af1ff7> in <module>()  
----> 1 df.tolist()  
  
//anaconda/lib/python2.7/site-packages/pandas/core/generic.pyc in __getattr__(self, name)  
    2742         if name in self._info_axis:  
    2743             return self[name]  
-> 2744         return object.__getattr__(self, name)  
    2745  
    2746     def __setattr__(self, name, value):  
  
AttributeError: 'DataFrame' object has no attribute 'tolist'
```

Obtener una matriz numpy de una serie:

```
In [5]: df['B'].values
```

```
Out[5]: array([ 1.,  2.,  3.])
```

También puede obtener una matriz de las columnas como matrices de números individuales de un marco de datos completo:

```
In [6]: df.values
```

```
Out[6]:
```

```
array([[1, 1.0, 'a', True],
       [2, 2.0, 'b', False],
       [3, 3.0, 'c', True]], dtype=object)
```

Obtención de un diccionario de una serie (utiliza el índice como claves):

```
In [7]: df['C'].to_dict()
Out[7]: {0: 'a', 1: 'b', 2: 'c'}
```

También puede recuperar todo el DataFrame como un diccionario:

```
In [8]: df.to_dict()
Out[8]:
{'A': {0: 1, 1: 2, 2: 3},
 'B': {0: 1.0, 1: 2.0, 2: 3.0},
 'C': {0: 'a', 1: 'b', 2: 'c'},
 'D': {0: True, 1: False, 2: True}}
```

El método `to_dict` tiene algunos parámetros diferentes para ajustar cómo se formatean los diccionarios. Para obtener una lista de dictados para cada fila:

```
In [9]: df.to_dict('records')
Out[9]:
[{'A': 1, 'B': 1.0, 'C': 'a', 'D': True},
 {'A': 2, 'B': 2.0, 'C': 'b', 'D': False},
 {'A': 3, 'B': 3.0, 'C': 'c', 'D': True}]
```

Consulte [la documentación](#) para ver la lista completa de opciones disponibles para crear diccionarios.

Lea [Making Pandas Play Nice con tipos de datos nativos de Python en línea](#):

<https://riptutorial.com/es/pandas/topic/8008/making-pandas-play-nice-con-tipos-de-datos-nativos-de-python>

# Capítulo 26: Manipulación de cuerdas

## Examples

### Expresiones regulares

```
# Extract strings with a specific regex
df= df['col_name'].str.extract(r'[Aa-Zz]')

# Replace strings within a regex
df['col_name'].str.replace('Replace this', 'With this')
```

Para obtener información sobre cómo hacer coincidir las cadenas [con expresiones regulares](#) , consulte [Introducción a las expresiones regulares](#) .

### Rebanar cuerdas

Las cadenas en una serie pueden cortarse utilizando el método `.str.slice()` , o más convenientemente, utilizando corchetes ( `.str[]` ).

```
In [1]: ser = pd.Series(['Lorem ipsum', 'dolor sit amet', 'consectetur adipiscing elit'])
In [2]: ser
Out[2]:
0      Lorem ipsum
1      dolor sit amet
2  consectetur adipiscing elit
dtype: object
```

Consigue el primer carácter de cada cadena:

```
In [3]: ser.str[0]
Out[3]:
0      L
1      d
2      c
dtype: object
```

Consigue los tres primeros caracteres de cada cadena:

```
In [4]: ser.str[:3]
Out[4]:
0      Lor
1      dol
2      con
dtype: object
```

Consigue el último carácter de cada cadena:

```
In [5]: ser.str[-1]
```

```
Out[5]:
0      m
1      t
2      t
dtype: object
```

Consigue los últimos tres caracteres de cada cadena:

```
In [6]: ser.str[-3:]
Out[6]:
0      sum
1      met
2      lit
dtype: object
```

Consigue los otros caracteres de los primeros 10 caracteres:

```
In [7]: ser.str[:10:2]
Out[7]:
0      Lrmis
1      dlrst
2      cnett
dtype: object
```

Las pandas se comportan de manera similar a Python cuando manejan rebanadas e índices. Por ejemplo, si un índice está fuera del rango, Python genera un error:

```
In [8]: 'Lorem ipsum'[12]
# IndexError: string index out of range
```

Sin embargo, si una porción está fuera del rango, se devuelve una cadena vacía:

```
In [9]: 'Lorem ipsum'[12:15]
Out[9]: ''
```

Pandas devuelve NaN cuando un índice está fuera de rango:

```
In [10]: ser.str[12]
Out[10]:
0      NaN
1         e
2         a
dtype: object
```

Y devuelve una cadena vacía si una porción está fuera de rango:

```
In [11]: ser.str[12:15]
Out[11]:
0
1      et
2      adi
dtype: object
```



## Comprobando el contenido de una cadena

`str.contains()` método `str.contains()` se puede usar para verificar si se produce un patrón en cada cadena de una serie. `str.startswith()` y `str.endswith()` también se pueden usar como versiones más especializadas.

```
In [1]: animals = pd.Series(['cat', 'dog', 'bear', 'cow', 'bird', 'owl', 'rabbit', 'snake'])
```

Compruebe si las cadenas contienen la letra 'a':

```
In [2]: animals.str.contains('a')
Out[2]:
0      True
1     False
2      True
3     False
4     False
5     False
6      True
7      True
8      True
dtype: bool
```

Esto se puede usar como un índice booleano para devolver solo los animales que contienen la letra 'a':

```
In [3]: animals[animals.str.contains('a')]
Out[3]:
0      cat
2     bear
6   rabbit
7     snake
dtype: object
```

`str.startswith` métodos `str.startswith` y `str.endswith` funcionan de manera similar, pero también aceptan tuplas como entradas.

```
In [4]: animals[animals.str.startswith(('b', 'c'))]
# Returns animals starting with 'b' or 'c'
Out[4]:
0      cat
2     bear
3      cow
4     bird
dtype: object
```

## Capitalización de cuerdas

```
In [1]: ser = pd.Series(['lORem iPsuM', 'Dolor sit amet', 'Consectetur Adipiscing Elit'])
```

Convertir todo a mayúsculas:

```
In [2]: ser.str.upper()
Out[2]:
0          LOREM IPSUM
1      DOLOR SIT AMET
2  CONSECTETUR ADIPISCING ELIT
dtype: object
```

Todo en minúsculas:

```
In [3]: ser.str.lower()
Out[3]:
0      lorem ipsum
1      dolor sit amet
2  consectetur adipiscing elit
dtype: object
```

Capitaliza el primer carácter y minúscula el resto:

```
In [4]: ser.str.capitalize()
Out[4]:
0      Lorem ipsum
1      Dolor sit amet
2  Consectetur adipiscing elit
dtype: object
```

Convierta cada cadena en un título (mayúscula el primer carácter de cada palabra en cada cadena, minúsculas en el resto):

```
In [5]: ser.str.title()
Out[5]:
0      Lorem Ipsum
1      Dolor Sit Amet
2  Consectetur Adipiscing Elit
dtype: object
```

Intercambiar casos (convertir minúsculas a mayúsculas y viceversa):

```
In [6]: ser.str.swapcase()
Out[6]:
0      LorEM IPsUm
1      dOLOR SIT AMET
2  cONSECTETUR aDIPISCING eLIT
dtype: object
```

Aparte de estos métodos que cambian la capitalización, se pueden utilizar varios métodos para verificar la capitalización de las cadenas.

```
In [7]: ser = pd.Series(['LOREM IPSUM', 'dolor sit amet', 'Consectetur Adipiscing Elit'])
```

Compruebe si está todo en minúsculas:

```
In [8]: ser.str.islower()
Out[8]:
```

```
0    False
1     True
2    False
dtype: bool
```

¿Es todo en mayúsculas?

```
In [9]: ser.str.isupper()
Out[9]:
0     True
1    False
2    False
dtype: bool
```

Es una cadena titlecased:

```
In [10]: ser.str.istitle()
Out[10]:
0    False
1    False
2     True
dtype: bool
```

Lea Manipulación de cuerdas en línea: <https://riptutorial.com/es/pandas/topic/2372/manipulacion-de-cuerdas>

# Capítulo 27: Manipulación sencilla de DataFrames.

## Examples

### Eliminar una columna en un DataFrame

Hay un par de formas de eliminar una columna en un DataFrame.

```
import numpy as np
import pandas as pd

np.random.seed(0)

pd.DataFrame(np.random.randn(5, 6), columns=list('ABCDEF'))

print(df)
# Output:
#           A           B           C           D           E           F
# 0 -0.895467  0.386902 -0.510805 -1.180632 -0.028182  0.428332
# 1  0.066517  0.302472 -0.634322 -0.362741 -0.672460 -0.359553
# 2 -0.813146 -1.726283  0.177426 -0.401781 -1.630198  0.462782
# 3 -0.907298  0.051945  0.729091  0.128983  1.139401 -1.234826
# 4  0.402342 -0.684810 -0.870797 -0.578850 -0.311553  0.056165
```

#### 1) usando `del`

```
del df['C']

print(df)
# Output:
#           A           B           D           E           F
# 0 -0.895467  0.386902 -1.180632 -0.028182  0.428332
# 1  0.066517  0.302472 -0.362741 -0.672460 -0.359553
# 2 -0.813146 -1.726283 -0.401781 -1.630198  0.462782
# 3 -0.907298  0.051945  0.128983  1.139401 -1.234826
# 4  0.402342 -0.684810 -0.578850 -0.311553  0.056165
```

#### 2) Usando `drop`

```
df.drop(['B', 'E'], axis='columns', inplace=True)
# or df = df.drop(['B', 'E'], axis=1) without the option inplace=True

print(df)
# Output:
#           A           D           F
# 0 -0.895467 -1.180632  0.428332
# 1  0.066517 -0.362741 -0.359553
# 2 -0.813146 -0.401781  0.462782
# 3 -0.907298  0.128983 -1.234826
# 4  0.402342 -0.578850  0.056165
```

### 3) Usando `drop` con números de columna

Para usar números enteros de columna en lugar de nombres (recuerde que los índices de columna comienzan en cero):

```
df.drop(df.columns[[0, 2]], axis='columns')

print(df)
# Output:
#          D
# 0 -1.180632
# 1 -0.362741
# 2 -0.401781
# 3  0.128983
# 4 -0.578850
```

### Renombrar una columna

```
df = pd.DataFrame({'old_name_1': [1, 2, 3], 'old_name_2': [5, 6, 7]})

print(df)
# Output:
#   old_name_1  old_name_2
# 0          1           5
# 1          2           6
# 2          3           7
```

Para cambiar el nombre de una o más columnas, pase los nombres antiguos y los nuevos nombres como un diccionario:

```
df.rename(columns={'old_name_1': 'new_name_1', 'old_name_2': 'new_name_2'}, inplace=True)
print(df)
# Output:
#   new_name_1  new_name_2
# 0          1           5
# 1          2           6
# 2          3           7
```

O una función:

```
df.rename(columns=lambda x: x.replace('old_', '_new'), inplace=True)
print(df)
# Output:
#   new_name_1  new_name_2
# 0          1           5
# 1          2           6
# 2          3           7
```

También puede establecer `df.columns` como la lista de los nuevos nombres:

```
df.columns = ['new_name_1', 'new_name_2']
print(df)
# Output:
#   new_name_1  new_name_2
```

```
# 0      1      5
# 1      2      6
# 2      3      7
```

Más detalles [se pueden encontrar aquí](#) .

## Añadiendo una nueva columna

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

print(df)
# Output:
#      A  B
# 0    1  4
# 1    2  5
# 2    3  6
```

## Asignar directamente

```
df['C'] = [7, 8, 9]

print(df)
# Output:
#      A  B  C
# 0    1  4  7
# 1    2  5  8
# 2    3  6  9
```

## Añadir una columna constante

```
df['C'] = 1

print(df)

# Output:
#      A  B  C
# 0    1  4  1
# 1    2  5  1
# 2    3  6  1
```

## Columna como expresión en otras columnas.

```
df['C'] = df['A'] + df['B']

# print(df)
# Output:
#      A  B  C
# 0    1  4  5
# 1    2  5  7
# 2    3  6  9

df['C'] = df['A']**df['B']
```

```
print(df)
# Output:
#      A  B      C
# 0  1  4      1
# 1  2  5     32
# 2  3  6    729
```

Las operaciones se calculan por componentes, por lo que si tuviéramos columnas como listas

```
a = [1, 2, 3]
b = [4, 5, 6]
```

La columna en la última expresión se obtendría como

```
c = [x**y for (x,y) in zip(a,b)]

print(c)
# Output:
# [1, 32, 729]
```

## Crealo sobre la marcha

```
df_means = df.assign(D=[10, 20, 30]).mean()

print(df_means)
# Output:
# A      2.0
# B      5.0
# C      7.0
# D     20.0 # adds a new column D before taking the mean
# dtype: float64
```

## agregar columnas múltiples

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df[['A2', 'B2']] = np.square(df)

print(df)
# Output:
#      A  B  A2  B2
# 0  1  4   1  16
# 1  2  5   4  25
# 2  3  6   9  36
```

## añadir múltiples columnas sobre la marcha

```
new_df = df.assign(A3=df.A*df.A2, B3=5*df.B)

print(new_df)
# Output:
```

```
#      A  B  A2  B2  A3  B3
# 0    1  4   1  16   1  20
# 1    2  5   4  25   8  25
# 2    3  6   9  36  27  30
```

## Localice y reemplace los datos en una columna

```
import pandas as pd

df = pd.DataFrame({'gender': ["male", "female", "female"],
                      'id': [1, 2, 3] })

>>> df
  gender  id
0    male   1
1  female   2
2  female   3
```

Para codificar el macho a 0 y la hembra a 1:

```
df.loc[df["gender"] == "male", "gender"] = 0
df.loc[df["gender"] == "female", "gender"] = 1

>>> df
  gender  id
0      0   1
1      1   2
2      1   3
```

## Añadiendo una nueva fila a DataFrame

Dado un DataFrame:

```
s1 = pd.Series([1,2,3])
s2 = pd.Series(['a','b','c'])

df = pd.DataFrame([list(s1), list(s2)], columns = ["C1", "C2", "C3"])
print df
```

Salida:

```
   C1  C2  C3
0    1   2   3
1    a   b   c
```

Permite agregar una nueva fila, [10,11,12] :

```
df = pd.DataFrame(np.array([[10,11,12]]), \
                  columns=["C1", "C2", "C3"]).append(df, ignore_index=True)
print df
```

Salida:



	C1	C2	C3
0	10	11	12
1	1	2	3
2	a	b	c

## Eliminar / eliminar filas de DataFrame

vamos a generar un DataFrame primero:

```
df = pd.DataFrame(np.arange(10).reshape(5,2), columns=list('ab'))

print(df)
# Output:
#    a  b
# 0  0  1
# 1  2  3
# 2  4  5
# 3  6  7
# 4  8  9
```

suelte filas con índices: 0 y 4 usando el método `drop(..., inplace=True)` :

```
df.drop([0,4], inplace=True)

print(df)
# Output
#    a  b
# 1  2  3
# 2  4  5
# 3  6  7
```

suelte filas con índices: 0 y 4 usando el método `df = drop(...)` :

```
df = pd.DataFrame(np.arange(10).reshape(5,2), columns=list('ab'))

df = df.drop([0,4])

print(df)
# Output:
#    a  b
# 1  2  3
# 2  4  5
# 3  6  7
```

utilizando el método de selección negativa:

```
df = pd.DataFrame(np.arange(10).reshape(5,2), columns=list('ab'))

df = df[~df.index.isin([0,4])]

print(df)
# Output:
#    a  b
# 1  2  3
# 2  4  5
```

## Reordenar columnas

```
# get a list of columns
cols = list(df)

# move the column to head of list using index, pop and insert
cols.insert(0, cols.pop(cols.index('listing')))

# use ix to reorder
df2 = df.ix[:, cols]
```

Lea Manipulación sencilla de DataFrames. en línea:

<https://riptutorial.com/es/pandas/topic/6694/manipulacion-sencilla-de-dataframes->

# Capítulo 28: Meta: Pautas de documentación.

## Observaciones

Esta meta publicación es similar a la versión de python

<http://stackoverflow.com/documentation/python/394/meta-documentation-guidelines#t=201607240058406359521> .

Por favor, haga sugerencias de edición, y comente sobre ellas (en lugar de los comentarios apropiados), para que podamos desarrollarlas / iterarlas sobre estas sugerencias :)

## Examples

### Mostrando fragmentos de código y salida

Dos opciones populares son usar:

notación ipython:

```
In [11]: df = pd.DataFrame([[1, 2], [3, 4]])

In [12]: df
Out[12]:
   0  1
0  1  2
1  3  4
```

Alternativamente (esto es popular en la documentación de python) y más concisamente:

```
df.columns # Out: RangeIndex(start=0, stop=2, step=1)

df[0]
# Out:
# 0    1
# 1    3
# Name: 0, dtype: int64

for col in df:
    print(col)
# prints:
# 0
# 1
```

*En general, esto es mejor para ejemplos más pequeños.*

Nota: La distinción entre salida e impresión. ipython lo aclara (las impresiones se producen antes de que se devuelva la salida):

```
In [21]: [print(col) for col in df]
```

```
0
1
Out[21]: [None, None]
```

## estilo

Utilice la biblioteca de pandas como `pd`, esto puede ser asumido (la importación no necesita estar en todos los ejemplos)

```
import pandas as pd
```

## PEP8!

- Sangría de 4 espacios
- los kwargs no deben usar espacios `f(a=1)`
- Límite de 80 caracteres (la línea completa ajustada en el fragmento de código renderizado debe ser altamente preferida)

## Compatibilidad con la versión pandas

La mayoría de los ejemplos funcionarán en varias versiones, si está utilizando una característica "nueva", debe mencionar cuándo se introdujo.

Ejemplo: `sort_values`.

## imprimir declaraciones

La mayoría de las veces se debe evitar la impresión, ya que puede ser una distracción (se debe preferir la salida).

Es decir:

```
a
# Out: 1
```

siempre es mejor que

```
print(a)
# prints: 1
```

## Prefiero el apoyo de python 2 y 3:

```
print(x)      # yes! (works same in python 2 and 3)
print x       # no! (python 2 only)
print(x, y)   # no! (works differently in python 2 and 3)
```

Lea Meta: Pautas de documentación. en línea: <https://riptutorial.com/es/pandas/topic/3253/meta--pautas-de-documentacion->

# Capítulo 29: Multiindex

## Examples

### Seleccione de MultiIndex por Nivel

Dado el siguiente DataFrame:

```
In [11]: df = pd.DataFrame(np.random.randn(6, 3), columns=['A', 'B', 'C'])

In [12]: df.set_index(['A', 'B'], inplace=True)

In [13]: df
Out[13]:
```

A	B	C
0.902764	-0.259656	-1.864541
-0.695893	0.308893	0.125199
1.696989	-1.221131	-2.975839
-1.132069	-1.086189	-1.945467
2.294835	-1.765507	1.567853
-1.788299	2.579029	0.792919

Obtenga los valores de A , por nombre:

```
In [14]: df.index.get_level_values('A')
Out[14]:
Float64Index([0.902764041011, -0.69589264969, 1.69698924476, -1.13206872067,
              2.29483481146, -1.788298829],
              dtype='float64', name='A')
```

O por número de nivel:

```
In [15]: df.index.get_level_values(level=0)
Out[15]:
Float64Index([0.902764041011, -0.69589264969, 1.69698924476, -1.13206872067,
              2.29483481146, -1.788298829],
              dtype='float64', name='A')
```

Y para un rango específico:

```
In [16]: df.loc[(df.index.get_level_values('A') > 0.5) & (df.index.get_level_values('A') <
2.1)]
Out[16]:
```

A	B	C
0.902764	-0.259656	-1.864541
1.696989	-1.221131	-2.975839

El rango también puede incluir múltiples columnas:

```
In [17]: df.loc[(df.index.get_level_values('A') > 0.5) & (df.index.get_level_values('B') < 0)]
Out[17]:
```

			C
A	B		
0.902764	-0.259656	-1.864541	
1.696989	-1.221131	-2.975839	
2.294835	-1.765507	1.567853	

Para extraer un valor específico puede usar `xs` (sección transversal):

```
In [18]: df.xs(key=0.9027639999999999)
Out[18]:
```

		C
B		
-0.259656	-1.864541	

```
In [19]: df.xs(key=0.9027639999999999, drop_level=False)
Out[19]:
```

			C
A	B		
0.902764	-0.259656	-1.864541	

## Iterar sobre DataFrame con MultiIndex

Dado el siguiente DataFrame:

```
In [11]: df = pd.DataFrame({'a':[1,1,1,2,2,3], 'b':[4,4,5,5,6,7], 'c':[10,11,12,13,14,15]})
In [12]: df.set_index(['a','b'], inplace=True)
In [13]: df
Out[13]:
```

		c
a	b	
1	4	10
	4	11
	5	12
2	5	13
	6	14
3	7	15

Puedes iterar por cualquier nivel del MultiIndex. Por ejemplo, `level=0` (también puede seleccionar el nivel por nombre, por ejemplo, `level='a'`):

```
In[21]: for idx, data in df.groupby(level=0):
        print('---')
        print(data)
---
      c
a b
1 4 10
  4 11
  5 12
---
      c
a b
```

```

2 5 13
   6 14
---
      c
a b
3 7 15

```

También puede seleccionar los niveles por nombre, por ejemplo `level = 'b'`:

```

In[22]: for idx, data in df.groupby(level='b'):
        print('---')
        print(data)

---
      c
a b
1 4 10
   4 11
---
      c
a b
1 5 12
2 5 13
---
      c
a b
2 6 14
---
      c
a b
3 7 15

```

## Configuración y clasificación de un MultiIndex

Este ejemplo muestra cómo usar los datos de columna para establecer un `MultiIndex` en un `pandas.DataFrame`.

```

In [1]: df = pd.DataFrame([['one', 'A', 100], ['two', 'A', 101], ['three', 'A', 102],
...:                      ['one', 'B', 103], ['two', 'B', 104], ['three', 'B', 105]],
...:                      columns=['c1', 'c2', 'c3'])

```

```

In [2]: df
Out[2]:
   c1 c2  c3
0  one  A 100
1  two  A 101
2 three  A 102
3  one  B 103
4  two  B 104
5 three  B 105

```

```

In [3]: df.set_index(['c1', 'c2'])
Out[3]:
      c3
c1    c2

```

```

one    A    100
two    A    101
three  A    102
one    B    103
two    B    104
three  B    105

```

Puede ordenar el índice justo después de establecerlo:

```

In [4]: df.set_index(['c1', 'c2']).sort_index()
Out[4]:
           c3
c1  c2
one  A    100
     B    103
three A    102
     B    105
two  A    101
     B    104

```

Tener un índice ordenado, dará como resultado búsquedas un poco más eficientes en el primer nivel:

```

In [5]: df_01 = df.set_index(['c1', 'c2'])

In [6]: %timeit df_01.loc['one']
1000 loops, best of 3: 607 µs per loop

In [7]: df_02 = df.set_index(['c1', 'c2']).sort_index()

In [8]: %timeit df_02.loc['one']
1000 loops, best of 3: 413 µs per loop

```

Una vez que se ha establecido el índice, puede realizar búsquedas para registros específicos o grupos de registros:

```

In [9]: df_indexed = df.set_index(['c1', 'c2']).sort_index()

In [10]: df_indexed.loc['one']
Out[10]:
           c3
c2
A    100
B    103

In [11]: df_indexed.loc['one', 'A']
Out[11]:
c3    100
Name: (one, A), dtype: int64

In [12]: df_indexed.xs((slice(None), 'A'))
Out[12]:
           c3
c1

```



```
one    100
three  102
two    101
```

## Cómo cambiar columnas MultiIndex a columnas estándar

Dado un DataFrame con columnas MultiIndex

```
# build an example DataFrame
midx = pd.MultiIndex(levels=[['zero', 'one'], ['x', 'y']], labels=[[1,1,0],[1,0,1]])
df = pd.DataFrame(np.random.randn(2,3), columns=midx)
```

In [2]: df

Out[2]:

```
      one          zero
      y          x          y
0  0.785806 -0.679039  0.513451
1 -0.337862 -0.350690 -1.423253
```

Si desea cambiar las columnas a columnas estándar (no MultiIndex), simplemente cambie el nombre de las columnas.

```
df.columns = ['A', 'B', 'C']
```

In [3]: df

Out[3]:

```
      A          B          C
0  0.785806 -0.679039  0.513451
1 -0.337862 -0.350690 -1.423253
```

## Cómo cambiar columnas estándar a MultiIndex

Comience con un DataFrame estándar

```
df = pd.DataFrame(np.random.randn(2,3), columns=['a', 'b', 'c'])
```

In [91]: df

Out[91]:

```
      a          b          c
0 -0.911752 -1.405419 -0.978419
1  0.603888 -1.187064 -0.035883
```

Ahora para cambiar a MultiIndex, cree un objeto MultiIndex y df.columns a df.columns .

```
midx = pd.MultiIndex(levels=[['zero', 'one'], ['x', 'y']], labels=[[1,1,0],[1,0,1]])
df.columns = midx
```

In [94]: df

Out[94]:

```
      one          zero
      y          x          y
0 -0.911752 -1.405419 -0.978419
1  0.603888 -1.187064 -0.035883
```

## Columnas multiindex

MultiIndex también se puede utilizar para crear DataFrames con columnas multinivel. Simplemente use la palabra clave de las `columns` en el comando DataFrame.

```
midx = pd.MultiIndex(levels=[['zero', 'one'], ['x', 'y']], labels=[[1,1,0,],[1,0,1,]])
df = pd.DataFrame(np.random.randn(6,4), columns=midx)
```

```
In [86]: df
```

```
Out[86]:
```

	one		zero	
	y	x	y	
0	0.625695	2.149377	0.006123	
1	-1.392909	0.849853	0.005477	

## Visualización de todos los elementos en el índice.

Para ver todos los elementos en el índice, cambie las opciones de impresión que "clasifican" la visualización del MultiIndex.

```
pd.set_option('display.multi_sparse', False)
df.groupby(['A', 'B']).mean()
```

```
# Output:
```

```
#      C
# A B
# a 1  107
# a 2  102
# a 3  115
# b 5   92
# b 8   98
# c 2   87
# c 4  104
# c 9  123
```

Lea Multiindex en línea: <https://riptutorial.com/es/pandas/topic/3840/multiindex>

# Capítulo 30: Obteniendo información sobre DataFrames

## Examples

### Obtener información de DataFrame y el uso de la memoria

Para obtener información básica sobre un DataFrame, incluidos los nombres de las columnas y los tipos de datos:

```
import pandas as pd

df = pd.DataFrame({'integers': [1, 2, 3],
                  'floats': [1.5, 2.5, 3],
                  'text': ['a', 'b', 'c'],
                  'ints with None': [1, None, 3]})

df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3 entries, 0 to 2
Data columns (total 4 columns):
floats                3 non-null float64
integers              3 non-null int64
ints with None        2 non-null float64
text                  3 non-null object
dtypes: float64(2), int64(1), object(1)
memory usage: 120.0+ bytes
```

Para obtener el uso de memoria del DataFrame:

```
>>> df.info(memory_usage='deep')
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3 entries, 0 to 2
Data columns (total 4 columns):
floats                3 non-null float64
integers              3 non-null int64
ints with None        2 non-null float64
text                  3 non-null object
dtypes: float64(2), int64(1), object(1)
memory usage: 234.0 bytes
```

### Lista de nombres de columna de DataFrame

```
df = pd.DataFrame({'a': [1, 2, 3], 'b': [4, 5, 6], 'c': [7, 8, 9]})
```

Para listar los nombres de columna en un DataFrame:

```
>>> list(df)
['a', 'b', 'c']
```

Este método de comprensión de lista es especialmente útil cuando se utiliza el depurador:

```
>>> [c for c in df]
['a', 'b', 'c']
```

Este es el camino largo:

```
sampledf.columns.tolist()
```

También puede imprimirlos como un índice en lugar de una lista (aunque esto no será muy visible para los marcos de datos con muchas columnas):

```
df.columns
```

## Las diversas estadísticas de resumen de Dataframe.

```
import pandas as pd
df = pd.DataFrame(np.random.randn(5, 5), columns=list('ABCDE'))
```

Para generar varias estadísticas de resumen. Para los valores numéricos el número de no-NA / valores nulos ( `count` ), la media ( `mean` ), la desviación estándar `std` y los valores conocido como el **resumen de cinco números** :

- `min` : mínimo (observación más pequeña)
- `25%` : cuartil inferior o primer cuartil (Q1)
- `50%` : mediana (valor medio, Q2)
- `75%` : cuartil superior o tercer cuartil (Q3)
- `max` : maximo (mayor observación)

```
>>> df.describe()
      A         B         C         D         E
count  5.000000  5.000000  5.000000  5.000000  5.000000
mean   -0.456917 -0.278666  0.334173  0.863089  0.211153
std     0.925617  1.091155  1.024567  1.238668  1.495219
min    -1.494346 -2.031457 -0.336471 -0.821447 -2.106488
25%    -1.143098 -0.407362 -0.246228 -0.087088 -0.082451
50%    -0.536503 -0.163950 -0.004099  1.509749  0.313918
75%     0.092630  0.381407  0.120137  1.822794  1.060268
max     0.796729  0.828034  2.137527  1.891436  1.870520
```

Lea Obteniendo información sobre DataFrames en línea:

<https://riptutorial.com/es/pandas/topic/6697/obteniendo-informacion-sobre-dataframes>

# Capítulo 31: Pandas Datareader

## Observaciones

Pandas datareader es un subpaquete que permite crear un marco de datos a partir de varias fuentes de datos de Internet, que actualmente incluyen:

- Yahoo! Financiar
- Google Finance
- St.Louis FED (FRED)
- Biblioteca de datos de Kenneth French
- Banco Mundial
- Google analítico

Para más información, [ver aquí](#).

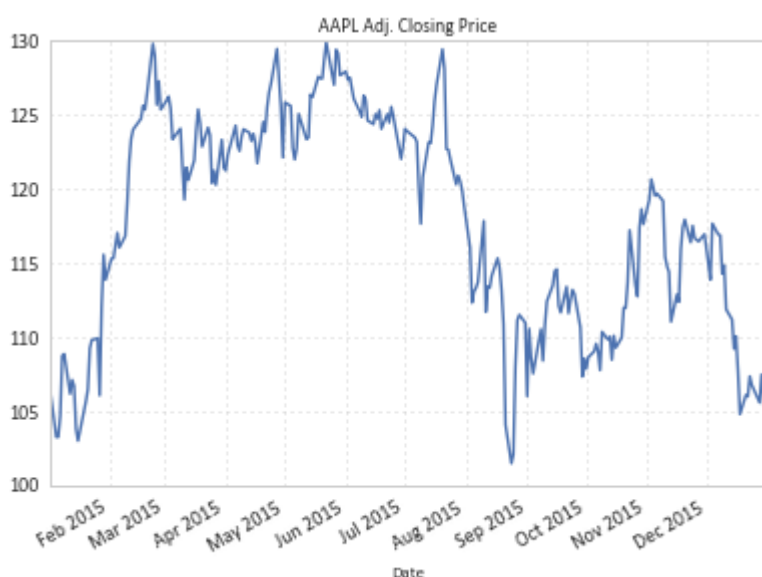
## Examples

### Ejemplo básico de Datareader (Yahoo Finance)

```
from pandas_datareader import data

# Only get the adjusted close.
aapl = data.DataReader("AAPL",
                       start='2015-1-1',
                       end='2015-12-31',
                       data_source='yahoo')['Adj Close']

>>> aapl.plot(title='AAPL Adj. Closing Price')
```



```
# Convert the adjusted closing prices to cumulative returns.
returns = aapl.pct_change()
```

```
>>> ((1 + returns).cumprod() - 1).plot(title='AAPL Cumulative Returns')
```



## Lectura de datos financieros (para múltiples tickers) en el panel de pandas - demostración

```
from datetime import datetime
import pandas_datareader.data as wb

stocklist = ['AAPL', 'GOOG', 'FB', 'AMZN', 'COP']

start = datetime(2016, 6, 8)
end = datetime(2016, 6, 11)

p = wb.DataReader(stocklist, 'yahoo', start, end)
```

p - es un panel de pandas, con el que podemos hacer cosas divertidas:

A ver que tenemos en nuestro panel.

```
In [388]: p.axes
Out[388]:
[Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'], dtype='object'),
 DatetimeIndex(['2016-06-08', '2016-06-09', '2016-06-10'], dtype='datetime64[ns]',
 name='Date', freq='D'),
 Index(['AAPL', 'AMZN', 'COP', 'FB', 'GOOG'], dtype='object')]

In [389]: p.keys()
Out[389]: Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'], dtype='object')
```

## selección y corte de datos

```
In [390]: p['Adj Close']
Out[390]:
```

	AAPL	AMZN	COP	FB	GOOG
Date					
2016-06-08	98.940002	726.640015	47.490002	118.389999	728.280029
2016-06-09	99.650002	727.650024	46.570000	118.559998	728.580017

```
2016-06-10  98.830002  717.909973  44.509998  116.620003  719.409973
```

```
In [391]: p['Volume']
```

```
Out[391]:
```

	AAPL	AMZN	COP	FB	GOOG
Date					
2016-06-08	20812700.0	2200100.0	9596700.0	14368700.0	1582100.0
2016-06-09	26419600.0	2163100.0	5389300.0	13823400.0	985900.0
2016-06-10	31462100.0	3409500.0	8941200.0	18412700.0	1206000.0

```
In [394]: p[:, :, 'AAPL']
```

```
Out[394]:
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2016-06-08	99.019997	99.559998	98.680000	98.940002	20812700.0	98.940002
2016-06-09	98.500000	99.989998	98.459999	99.650002	26419600.0	99.650002
2016-06-10	98.529999	99.349998	98.480003	98.830002	31462100.0	98.830002

```
In [395]: p[:, '2016-06-10']
```

```
Out[395]:
```

	Open	High	Low	Close	Volume	Adj Close
AAPL	98.529999	99.349998	98.480003	98.830002	31462100.0	98.830002
AMZN	722.349976	724.979980	714.210022	717.909973	3409500.0	717.909973
COP	45.900002	46.119999	44.259998	44.509998	8941200.0	44.509998
FB	117.540001	118.110001	116.260002	116.620003	18412700.0	116.620003
GOOG	719.469971	725.890015	716.429993	719.409973	1206000.0	719.409973

Lea Pandas Datareader en línea: <https://riptutorial.com/es/pandas/topic/1912/pandas-datareader>

# Capítulo 32: pd.DataFrame.apply

## Examples

### pandas.DataFrame.apply Uso Básico

El método `pandas.DataFrame.apply()` se usa para aplicar una función dada a un `DataFrame` completo, por ejemplo, calculando la raíz cuadrada de cada entrada de un `DataFrame` dado o sumando cada fila de un `DataFrame` para devolver una `Series`.

El siguiente es un ejemplo básico del uso de esta función:

```
# create a random DataFrame with 7 rows and 2 columns
df = pd.DataFrame(np.random.randint(0,100,size = (7,2)),
                  columns = ['fst','snd'])

>>> df
   fst  snd
0   40   94
1   58   93
2   95   95
3   88   40
4   25   27
5   62   64
6   18   92

# apply the square root function to each column:
# (this returns a DataFrame where each entry is the sqrt of the entry in df;
# setting axis=0 or axis=1 doesn't make a difference)
>>> df.apply(np.sqrt)
   fst      snd
0  6.324555  9.695360
1  7.615773  9.643651
2  9.746794  9.746794
3  9.380832  6.324555
4  5.000000  5.196152
5  7.874008  8.000000
6  4.242641  9.591663

# sum across the row (axis parameter now makes a difference):
>>> df.apply(np.sum, axis=1)
0    134
1    151
2    190
3    128
4     52
5    126
6    110
dtype: int64

>>> df.apply(np.sum)
fst    386
snd    505
dtype: int64
```



Lea `pd.DataFrame.apply` en línea: <https://riptutorial.com/es/pandas/topic/7024/pd-dataframe-apply>

# Capítulo 33: Remodelación y pivotamiento

## Examples

### Simple pivotante

Primero intente usar `pivot` :

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'Name': ['Mary', 'Josh', 'Jon', 'Lucy', 'Jane', 'Sue'],
                  'Age': [34, 37, 29, 40, 29, 31],
                  'City': ['Boston', 'New York', 'Chicago', 'Los Angeles', 'Chicago',
                           'Boston'],
                  'Position': ['Manager', 'Programmer', 'Manager', 'Manager', 'Programmer',
                              'Programmer']},
                  columns=['Name', 'Position', 'City', 'Age'])

print (df)
```

	Name	Position	City	Age
0	Mary	Manager	Boston	34
1	Josh	Programmer	New York	37
2	Jon	Manager	Chicago	29
3	Lucy	Manager	Los Angeles	40
4	Jane	Programmer	Chicago	29
5	Sue	Programmer	Boston	31

```
print (df.pivot(index='Position', columns='City', values='Age'))
```

City	Boston	Chicago	Los Angeles	New York
Position				
Manager	34.0	29.0	40.0	NaN
Programmer	31.0	29.0	NaN	37.0

Si necesita restablecer el índice, elimine los nombres de las columnas y complete los valores de NaN:

```
#pivoting by numbers - column Age
print (df.pivot(index='Position', columns='City', values='Age')
        .reset_index()
        .rename_axis(None, axis=1)
        .fillna(0))
```

	Position	Boston	Chicago	Los Angeles	New York
0	Manager	34.0	29.0	40.0	0.0
1	Programmer	31.0	29.0	0.0	37.0

```
#pivoting by strings - column Name
print (df.pivot(index='Position', columns='City', values='Name'))
```

City	Boston	Chicago	Los Angeles	New York
Position				
Manager	Mary	Jon	Lucy	None
Programmer	Sue	Jane	None	Josh

## Pivotando con la agregación.

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'Name':['Mary', 'Jon','Lucy', 'Jane', 'Sue', 'Mary', 'Lucy'],
                  'Age':[35, 37, 40, 29, 31, 26, 28],
                  'City':['Boston', 'Chicago', 'Los Angeles', 'Chicago', 'Boston', 'Boston',
                           'Chicago'],
                  'Position':['Manager','Manager','Manager','Programmer',
                              'Programmer','Manager','Manager'],
                  'Sex':['Female','Male','Female','Female', 'Female','Female','Female']},
                  columns=['Name','Position','City','Age','Sex'])

print (df)
```

	Name	Position	City	Age	Sex
0	Mary	Manager	Boston	35	Female
1	Jon	Manager	Chicago	37	Male
2	Lucy	Manager	Los Angeles	40	Female
3	Jane	Programmer	Chicago	29	Female
4	Sue	Programmer	Boston	31	Female
5	Mary	Manager	Boston	26	Female
6	Lucy	Manager	Chicago	28	Female

Si usa `pivot` , obtenga error:

```
print (df.pivot(index='Position', columns='City', values='Age'))
```

**ValueError: el índice contiene entradas duplicadas, no se puede reformar**

Utilice `pivot_table` con función de agregación:

```
#default aggfunc is np.mean
print (df.pivot_table(index='Position', columns='City', values='Age'))
```

City	Boston	Chicago	Los Angeles
Position			
Manager	30.5	32.5	40.0
Programmer	31.0	29.0	NaN

```
print (df.pivot_table(index='Position', columns='City', values='Age', aggfunc=np.mean))
```

City	Boston	Chicago	Los Angeles
Position			
Manager	30.5	32.5	40.0
Programmer	31.0	29.0	NaN

Otras funciones agg:

```
print (df.pivot_table(index='Position', columns='City', values='Age', aggfunc=sum))
```

City	Boston	Chicago	Los Angeles
Position			
Manager	61.0	65.0	40.0
Programmer	31.0	29.0	NaN

```
#lost data !!!
print (df.pivot_table(index='Position', columns='City', values='Age', aggfunc='first'))
```

City	Boston	Chicago	Los Angeles
------	--------	---------	-------------

Position				
Manager	35.0	37.0	40.0	
Programmer	31.0	29.0	NaN	

Si necesita agregar por columnas con valores de `string`:

```
print (df.pivot_table(index='Position', columns='City', values='Name'))
```

**DataError: No hay tipos numéricos para agregar**

Puede utilizar estas funciones de `agagating`:

```
print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc='first'))
City          Boston Chicago Los Angeles
Position
Manager      Mary      Jon      Lucy
Programmer    Sue      Jane      None

print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc='last'))
City          Boston Chicago Los Angeles
Position
Manager      Mary      Lucy      Lucy
Programmer    Sue      Jane      None

print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc='sum'))
City          Boston  Chicago Los Angeles
Position
Manager      MaryMary  JonLucy      Lucy
Programmer      Sue      Jane      None

print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc=', '.join))
City          Boston    Chicago Los Angeles
Position
Manager      Mary, Mary  Jon, Lucy      Lucy
Programmer      Sue      Jane      None

print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc=', '.join,
fill_value='-')
      .reset_index()
      .rename_axis(None, axis=1))
      Position    Boston    Chicago Los Angeles
0   Manager  Mary, Mary  Jon, Lucy      Lucy
1  Programmer      Sue      Jane      -
```

La información sobre el sexo aún no ha sido utilizada. Podría ser cambiado por una de las columnas, o podría agregarse como otro nivel:

```
print (df.pivot_table(index='Position', columns=['City','Sex'], values='Age',
aggfunc='first'))
```

City	Boston	Chicago	Los Angeles	
Sex	Female	Female	Male	Female
Position				
Manager	35.0	28.0	37.0	40.0
Programmer	31.0	29.0	NaN	NaN

Se pueden especificar varias columnas en cualquiera de los atributos, columnas y valores.

```
print (df.pivot_table(index=['Position','Sex'], columns='City', values='Age',
aggfunc='first'))
```

City		Boston	Chicago	Los Angeles
Position	Sex			
Manager	Female	35.0	28.0	40.0
	Male	NaN	37.0	NaN
Programmer	Female	31.0	29.0	NaN

## Aplicando varias funciones de agregación.

Puede aplicar fácilmente múltiples funciones durante un solo pivote:

```
In [23]: import numpy as np
```

```
In [24]: df.pivot_table(index='Position', values='Age', aggfunc=[np.mean, np.std])
Out[24]:
```

	mean	std
Position		
Manager	34.333333	5.507571
Programmer	32.333333	4.163332

A veces, es posible que desee aplicar funciones específicas a columnas específicas:

```
In [35]: df['Random'] = np.random.random(6)
```

```
In [36]: df
```

```
Out[36]:
```

	Name	Position	City	Age	Random
0	Mary	Manager	Boston	34	0.678577
1	Josh	Programmer	New York	37	0.973168
2	Jon	Manager	Chicago	29	0.146668
3	Lucy	Manager	Los Angeles	40	0.150120
4	Jane	Programmer	Chicago	29	0.112769
5	Sue	Programmer	Boston	31	0.185198

For example, find the mean age, and standard deviation of random by Position:

```
In [37]: df.pivot_table(index='Position', aggfunc={'Age': np.mean, 'Random': np.std})
```

```
Out[37]:
```

	Age	Random
Position		
Manager	34.333333	0.306106
Programmer	32.333333	0.477219

Uno puede pasar una lista de funciones para aplicar a las columnas individuales también:

```
In [38]: df.pivot_table(index='Position', aggfunc={'Age': np.mean, 'Random': [np.mean,
np.std]}))
```

```
Out[38]:
```

	Age	Random	
	mean	mean	std
Position			
Manager	34.333333	0.325122	0.306106
Programmer	32.333333	0.423712	0.477219

## Apilamiento y desapilamiento.

```
import pandas as pd
import numpy as np

np.random.seed(0)
tuples = list(zip(*[['bar', 'bar', 'foo', 'foo', 'qux', 'qux'],
                    ['one', 'two', 'one', 'two', 'one', 'two']]))

idx = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])
df = pd.DataFrame(np.random.randn(6, 2), index=idx, columns=['A', 'B'])
print (df)
```

		A	B
first	second		
bar	one	1.764052	0.400157
	two	0.978738	2.240893
foo	one	1.867558	-0.977278
	two	0.950088	-0.151357
qux	one	-0.103219	0.410599
	two	0.144044	1.454274

```
print (df.stack())
first second
bar      one      A      1.764052
          two      A      0.978738
          two      B      2.240893
foo      one      A      1.867558
          two      A      0.950088
          two      B     -0.151357
qux      one      A     -0.103219
          two      A      0.144044
          two      B      1.454274

dtype: float64

#reset index, rename column name
print (df.stack().reset_index(name='val2').rename(columns={'level_2': 'val1'}))
```

	first	second	val1	val2
0	bar	one	A	1.764052
1	bar	one	B	0.400157
2	bar	two	A	0.978738
3	bar	two	B	2.240893
4	foo	one	A	1.867558
5	foo	one	B	-0.977278
6	foo	two	A	0.950088
7	foo	two	B	-0.151357
8	qux	one	A	-0.103219
9	qux	one	B	0.410599
10	qux	two	A	0.144044
11	qux	two	B	1.454274

```
print (df.unstack())
```

		A		B	
	second	one	two	one	two
first					
bar		1.764052	0.978738	0.400157	2.240893

```
foo      1.867558  0.950088 -0.977278 -0.151357
qux     -0.103219  0.144044  0.410599  1.454274
```

`rename_axis` (nuevo en pandas 0.18.0):

```
#reset index, remove columns names
df1 = df.unstack().reset_index().rename_axis((None, None), axis=1)
#reset MultiIndex in columns with list comprehension
df1.columns = ['_'.join(col).strip('_') for col in df1.columns]
print (df1)
   first  A_one  A_two  B_one  B_two
0   bar  1.764052  0.978738  0.400157  2.240893
1   foo  1.867558  0.950088 -0.977278 -0.151357
2   qux -0.103219  0.144044  0.410599  1.454274
```

## los pandas braman 0.18.0

```
#reset index
df1 = df.unstack().reset_index()
#remove columns names
df1.columns.names = (None, None)
#reset MultiIndex in columns with list comprehension
df1.columns = ['_'.join(col).strip('_') for col in df1.columns]
print (df1)
   first  A_one  A_two  B_one  B_two
0   bar  1.764052  0.978738  0.400157  2.240893
1   foo  1.867558  0.950088 -0.977278 -0.151357
2   qux -0.103219  0.144044  0.410599  1.454274
```

## Tabulación cruzada

```
import pandas as pd
df = pd.DataFrame({'Sex': ['M', 'M', 'F', 'M', 'F', 'F', 'M', 'M', 'F', 'F'],
                   'Age': [20, 19, 17, 35, 22, 22, 12, 15, 17, 22],
                   'Heart Disease': ['Y', 'N', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y']})
```

df

```
   Age  Heart Disease Sex
0    20           Y    M
1    19           N    M
2    17           Y    F
3    35           N    M
4    22           N    F
5    22           Y    F
6    12           N    M
7    15           Y    M
8    17           N    F
9    22           Y    F
```

```
pd.crosstab(df['Sex'], df['Heart Disease'])
```

```
Heart Disease  N  Y
Sex
F              2  3
M              3  2
```

Usando la notación de puntos:

```
pd.crosstab(df.Sex, df.Age)
```

Age	12	15	17	19	20	22	35
Sex							
F	0	0	2	0	0	3	0
M	1	1	0	1	1	0	1

Conseguir la transposición de DF:

```
pd.crosstab(df.Sex, df.Age).T
```

Sex	F	M
Age		
12	0	1
15	0	1
17	2	0
19	0	1
20	0	1
22	3	0
35	0	1

Obtención de márgenes o acumulativos:

```
pd.crosstab(df['Sex'], df['Heart Disease'], margins=True)
```

Heart Disease	N	Y	All
Sex			
F	2	3	5
M	3	2	5
All	5	5	10

Consiguiendo transposición de acumulativa:

```
pd.crosstab(df['Sex'], df['Age'], margins=True).T
```

Sex	F	M	All
Age			
12	0	1	1
15	0	1	1
17	2	0	2
19	0	1	1
20	0	1	1
22	3	0	3
35	0	1	1
All	5	5	10

Obtención de porcentajes:

```
pd.crosstab(df["Sex"],df['Heart Disease']).apply(lambda r: r/len(df), axis=1)
```

Heart Disease	N	Y
Sex		



F	0.2	0.3
M	0.3	0.2

Obtención acumulativa y multiplicación por 100:

```
df2 = pd.crosstab(df["Age"],df['Sex'], margins=True ).apply(lambda r: r/len(df)*100, axis=1)
```

```
df2
```

Sex	F	M	All
Age			
12	0.0	10.0	10.0
15	0.0	10.0	10.0
17	20.0	0.0	20.0
19	0.0	10.0	10.0
20	0.0	10.0	10.0
22	30.0	0.0	30.0
35	0.0	10.0	10.0
All	50.0	50.0	100.0

Eliminando una columna del DF (una forma):

```
df2[["F","M"]]
```

Sex	F	M
Age		
12	0.0	10.0
15	0.0	10.0
17	20.0	0.0
19	0.0	10.0
20	0.0	10.0
22	30.0	0.0
35	0.0	10.0
All	50.0	50.0

Las pandas se derriten para ir de lo ancho a lo largo.

```
>>> df
   ID  Year  Jan_salary  Feb_salary  Mar_salary
0   1  2016         4500         4200         4700
1   2  2016         3800         3600         4400
2   3  2016         5500         5200         5300

>>> melted_df = pd.melt(df,id_vars=['ID','Year'],
                        value_vars=['Jan_salary','Feb_salary','Mar_salary'],
                        var_name='month',value_name='salary')

>>> melted_df
   ID  Year  month  salary
0   1  2016  Jan_salary    4500
1   2  2016  Jan_salary    3800
2   3  2016  Jan_salary    5500
3   1  2016  Feb_salary    4200
4   2  2016  Feb_salary    3600
5   3  2016  Feb_salary    5200
6   1  2016  Mar_salary    4700
7   2  2016  Mar_salary    4400
```

```

8    3    2016    Mar_salary    5300

>>> melted_['month'] = melted_['month'].str.replace('_salary','')

>>> import calendar
>>> def mapper(month_abbr):
...     # from http://stackoverflow.com/a/3418092/42346
...     d = {v: str(k).zfill(2) for k,v in enumerate(calendar.month_abbr)}
...     return d[month_abbr]

>>> melted_df['month'] = melted_df['month'].apply(mapper)
>>> melted_df
   ID  Year month  salary
0    1  2016    01    4500
1    2  2016    01    3800
2    3  2016    01    5500
3    1  2016    02    4200
4    2  2016    02    3600
5    3  2016    02    5200
6    1  2016    03    4700
7    2  2016    03    4400
8    3  2016    03    5300

```

## Dividir (remodelar) cadenas CSV en columnas en varias filas, con un elemento por fila

```

import pandas as pd

df = pd.DataFrame([{'var1': 'a,b,c', 'var2': 1, 'var3': 'XX'},
                   {'var1': 'd,e,f,x,y', 'var2': 2, 'var3': 'ZZ'}])

print(df)

reshaped = \
(df.set_index(df.columns.drop('var1',1).tolist())
 .var1.str.split(',', expand=True)
 .stack()
 .reset_index()
 .rename(columns={0:'var1'})
 .loc[:, df.columns]
)

print(reshaped)

```

Salida:

```

      var1  var2 var3
0      a,b,c     1  XX
1  d,e,f,x,y     2  ZZ

      var1  var2 var3
0        a     1  XX
1        b     1  XX
2        c     1  XX
3        d     2  ZZ
4        e     2  ZZ
5        f     2  ZZ

```

6	x	2	ZZ
7	y	2	ZZ

Lea Remodelación y pivotamiento en línea:

<https://riptutorial.com/es/pandas/topic/1463/remodelacion-y-pivotamiento>

# Capítulo 34: Remuestreo

## Examples

### Downsampling y upmpling

```
import pandas as pd
import numpy as np

np.random.seed(0)
rng = pd.date_range('2015-02-24', periods=10, freq='T')
df = pd.DataFrame({'Val' : np.random.randn(len(rng))}, index=rng)
print (df)
```

```
                Val
2015-02-24 00:00:00  1.764052
2015-02-24 00:01:00  0.400157
2015-02-24 00:02:00  0.978738
2015-02-24 00:03:00  2.240893
2015-02-24 00:04:00  1.867558
2015-02-24 00:05:00 -0.977278
2015-02-24 00:06:00  0.950088
2015-02-24 00:07:00 -0.151357
2015-02-24 00:08:00 -0.103219
2015-02-24 00:09:00  0.410599
```

```
#downsampling with aggregating sum
print (df.resample('5Min').sum())
```

```
                Val
2015-02-24 00:00:00  7.251399
2015-02-24 00:05:00  0.128833
```

```
#5Min is same as 5T
```

```
print (df.resample('5T').sum())
```

```
                Val
2015-02-24 00:00:00  7.251399
2015-02-24 00:05:00  0.128833
```

```
#upsampling and fill NaN values method forward filling
```

```
print (df.resample('30S').ffill())
```

```
                Val
2015-02-24 00:00:00  1.764052
2015-02-24 00:00:30  1.764052
2015-02-24 00:01:00  0.400157
2015-02-24 00:01:30  0.400157
2015-02-24 00:02:00  0.978738
2015-02-24 00:02:30  0.978738
2015-02-24 00:03:00  2.240893
2015-02-24 00:03:30  2.240893
2015-02-24 00:04:00  1.867558
2015-02-24 00:04:30  1.867558
2015-02-24 00:05:00 -0.977278
2015-02-24 00:05:30 -0.977278
2015-02-24 00:06:00  0.950088
2015-02-24 00:06:30  0.950088
2015-02-24 00:07:00 -0.151357
2015-02-24 00:07:30 -0.151357
```

```
2015-02-24 00:08:00 -0.103219
2015-02-24 00:08:30 -0.103219
2015-02-24 00:09:00 0.410599
```

Lea Remuestreo en línea: <https://riptutorial.com/es/pandas/topic/2164/remuestreo>

# Capítulo 35: Secciones transversales de diferentes ejes con MultiIndex.

## Examples

### Selección de secciones utilizando .xs.

```
In [1]:
import pandas as pd
import numpy as np
arrays = [['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],
          ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two']]
idx_row = pd.MultiIndex.from_arrays(arrays, names=['Row_First', 'Row_Second'])
idx_col = pd.MultiIndex.from_product(['A', 'B'], ['i', 'ii'])
names=['Col_First', 'Col_Second'])
df = pd.DataFrame(np.random.randn(8,4), index=idx_row, columns=idx_col)
```

```
Out[1]:
```

Col_First		A		B	
Col_Second		i	ii	i	ii
Row_First	Row_Second				
bar	one	-0.452982	-1.872641	0.248450	-0.319433
	two	-0.460388	-0.136089	-0.408048	0.998774
baz	one	0.358206	-0.319344	-2.052081	-0.424957
	two	-0.823811	-0.302336	1.158968	0.272881
foo	one	-0.098048	-0.799666	0.969043	-0.595635
	two	-0.358485	0.412011	-0.667167	1.010457
qux	one	1.176911	1.578676	0.350719	0.093351
	two	0.241956	1.082138	-0.516898	-0.196605

`.xs` acepta un `level` (ya sea el nombre de dicho nivel o un entero) y un `axis` : 0 para las filas, 1 para las columnas.

`.xs` está disponible para `pandas.Series` y `pandas.DataFrame`.

### Selección en filas:

```
In [2]: df.xs('two', level='Row_Second', axis=0)
Out[2]:
```

Col_First		A		B	
Col_Second		i	ii	i	ii
Row_First					
bar		-0.460388	-0.136089	-0.408048	0.998774
baz		-0.823811	-0.302336	1.158968	0.272881
foo		-0.358485	0.412011	-0.667167	1.010457
qux		0.241956	1.082138	-0.516898	-0.196605

### Selección en columnas:

```
In [3]: df.xs('ii', level=1, axis=1)
Out[3]:
```

Col_First		A	B
Row_First	Row_Second		
bar	one	-1.872641	-0.319433
	two	-0.136089	0.998774
baz	one	-0.319344	-0.424957
	two	-0.302336	0.272881
foo	one	-0.799666	-0.595635
	two	0.412011	1.010457
qux	one	1.578676	0.093351
	two	1.082138	-0.196605

**.xs solo funciona para la selección, la asignación NO es posible (obtener, no configurar): "**

```
In [4]: df.xs('ii', level='Col_Second', axis=1) = 0
File "<ipython-input-10-92e0785187ba>", line 1
      df.xs('ii', level='Col_Second', axis=1) = 0
                                             ^
SyntaxError: can't assign to function call
```

## Usando .loc y slicers

A diferencia del método `.xs`, esto le permite asignar valores. La indexación utilizando máquinas de cortar está disponible desde la versión 0.14.0 .

```
In [1]:
import pandas as pd
import numpy as np
arrays = [['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],
          ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two']]
idx_row = pd.MultiIndex.from_arrays(arrays, names=['Row_First', 'Row_Second'])
idx_col = pd.MultiIndex.from_product(['A', 'B'], ['i', 'ii'])
names=['Col_First', 'Col_Second'])
df = pd.DataFrame(np.random.randn(8,4), index=idx_row, columns=idx_col)
```

```
Out[1]:
Col_First      A      B
Col_Second     i      ii     i      ii
Row_First Row_Second
bar      one    -0.452982 -1.872641  0.248450 -0.319433
        two    -0.460388 -0.136089 -0.408048  0.998774
baz      one     0.358206 -0.319344 -2.052081 -0.424957
        two    -0.823811 -0.302336  1.158968  0.272881
foo      one    -0.098048 -0.799666  0.969043 -0.595635
        two    -0.358485  0.412011 -0.667167  1.010457
qux      one     1.176911  1.578676  0.350719  0.093351
        two     0.241956  1.082138 -0.516898 -0.196605
```

## Selección en filas :

```
In [2]: df.loc[(slice(None), 'two'), :]
Out[2]:
Col_First      A      B
Col_Second     i      ii     i      ii
Row_First Row_Second
bar      two    -0.460388 -0.136089 -0.408048  0.998774
baz      two    -0.823811 -0.302336  1.158968  0.272881
```

foo	two	-0.358485	0.412011	-0.667167	1.010457
qux	two	0.241956	1.082138	-0.516898	-0.196605

## Selección en columnas:

```
In [3]: df.loc[:, (slice(None), 'ii')]
Out[3]:
```

Col_First		A	B
Col_Second		ii	ii
Row_First	Row_Second		
bar	one	-1.872641	-0.319433
	two	-0.136089	0.998774
baz	one	-0.319344	-0.424957
	two	-0.302336	0.272881
foo	one	-0.799666	-0.595635
	two	0.412011	1.010457
qux	one	1.578676	0.093351
	two	1.082138	-0.196605

## Selección en ambos ejes ::

```
In [4]: df.loc[(slice(None), 'two'), (slice(None), 'ii')]
Out[4]:
```

Col_First		A	B
Col_Second		ii	ii
Row_First	Row_Second		
bar	two	-0.136089	0.998774
baz	two	-0.302336	0.272881
foo	two	0.412011	1.010457
qux	two	1.082138	-0.196605

## Trabajos de asignación (a diferencia de .xs):

```
In [5]: df.loc[(slice(None), 'two'), (slice(None), 'ii')]=0
df
Out[5]:
```

Col_First		A		B	
Col_Second		i	ii	i	ii
Row_First	Row_Second				
bar	one	-0.452982	-1.872641	0.248450	-0.319433
	two	-0.460388	0.000000	-0.408048	0.000000
baz	one	0.358206	-0.319344	-2.052081	-0.424957
	two	-0.823811	0.000000	1.158968	0.000000
foo	one	-0.098048	-0.799666	0.969043	-0.595635
	two	-0.358485	0.000000	-0.667167	0.000000
qux	one	1.176911	1.578676	0.350719	0.093351
	two	0.241956	0.000000	-0.516898	0.000000

Lea Secciones transversales de diferentes ejes con MultiIndex. en línea:

<https://riptutorial.com/es/pandas/topic/8099/secciones-transversales-de-diferentes-ejes-con-multiindex->



# Capítulo 36: Serie

## Examples

### Ejemplos de creación de series simples

Una serie es una estructura de datos de una dimensión. Es un poco como una matriz sobrealimentada, o un diccionario.

```
import pandas as pd

s = pd.Series([10, 20, 30])

>>> s
0    10
1    20
2    30
dtype: int64
```

Cada valor en una serie tiene un índice. De forma predeterminada, los índices son enteros, que van desde 0 hasta la longitud de la serie menos 1. En el ejemplo anterior, puede ver los índices impresos a la izquierda de los valores.

Puedes especificar tus propios índices:

```
s2 = pd.Series([1.5, 2.5, 3.5], index=['a', 'b', 'c'], name='my_series')

>>> s2
a    1.5
b    2.5
c    3.5
Name: my_series, dtype: float64

s3 = pd.Series(['a', 'b', 'c'], index=list('ABC'))

>>> s3
A    a
B    b
C    c
dtype: object
```

### Series con fecha y hora

```
import pandas as pd
import numpy as np

np.random.seed(0)
rng = pd.date_range('2015-02-24', periods=5, freq='T')
s = pd.Series(np.random.randn(len(rng)), index=rng)
print (s)

2015-02-24 00:00:00    1.764052
```

```

2015-02-24 00:01:00    0.400157
2015-02-24 00:02:00    0.978738
2015-02-24 00:03:00    2.240893
2015-02-24 00:04:00    1.867558
Freq: T, dtype: float64

rng = pd.date_range('2015-02-24', periods=5, freq='T')
s1 = pd.Series(rng)
print (s1)

0    2015-02-24 00:00:00
1    2015-02-24 00:01:00
2    2015-02-24 00:02:00
3    2015-02-24 00:03:00
4    2015-02-24 00:04:00
dtype: datetime64[ns]

```

## Algunos consejos rápidos sobre Series in Pandas

Supongamos que tenemos la siguiente serie:

```

>>> import pandas as pd
>>> s = pd.Series([1, 4, 6, 3, 8, 7, 4, 5])
>>> s
0    1
1    4
2    6
3    3
4    8
5    7
6    4
7    5
dtype: int64

```

Las siguientes son algunas cosas simples que resultan útiles cuando se trabaja con Series:

Para obtener la longitud de s:

```

>>> len(s)
8

```

Para acceder a un elemento en s:

```

>>> s[4]
8

```

Para acceder a un elemento en s usando el índice:

```

>>> s.loc[2]
6

```

Para acceder a una subserie dentro de s:

```
>>> s[1:3]
1    4
2    6
dtype: int64
```

Para obtener una sub-serie de s con valores mayores a 5:

```
>>> s[s > 5]
2    6
4    8
5    7
dtype: int64
```

Para obtener la desviación mínima, máxima, media y estándar:

```
>>> s.min()
1
>>> s.max()
8
>>> s.mean()
4.75
>>> s.std()
2.2519832529192065
```

Para convertir el tipo Serie a flotar:

```
>>> s.astype(float)
0    1.0
1    4.0
2    6.0
3    3.0
4    8.0
5    7.0
6    4.0
7    5.0
dtype: float64
```

Para obtener los valores en s como una matriz numpy:

```
>>> s.values
array([1, 4, 6, 3, 8, 7, 4, 5])
```

Para hacer una copia de s:

```
>>> d = s.copy()
>>> d
0    1
1    4
2    6
3    3
4    8
5    7
6    4
7    5
dtype: int64
```

## Aplicando una función a una serie

Pandas proporciona una forma efectiva de aplicar una función a cada elemento de una serie y obtener una nueva serie. Supongamos que tenemos la siguiente serie:

```
>>> import pandas as pd
>>> s = pd.Series([3, 7, 5, 8, 9, 1, 0, 4])
>>> s
0    3
1    7
2    5
3    8
4    9
5    1
6    0
7    4
dtype: int64
```

y una función cuadrada:

```
>>> def square(x):
...     return x*x
```

Simplemente podemos aplicar el cuadrado a cada elemento de `s` y obtener una nueva serie:

```
>>> t = s.apply(square)
>>> t
0     9
1    49
2    25
3    64
4    81
5     1
6     0
7    16
dtype: int64
```

En algunos casos es más fácil usar una expresión lambda:

```
>>> s.apply(lambda x: x ** 2)
0     9
1    49
2    25
3    64
4    81
5     1
6     0
7    16
dtype: int64
```

o podemos usar cualquier función incorporada:

```
>>> q = pd.Series(['Bob', 'Jack', 'Rose'])
>>> q.apply(str.lower)
```

```
0    bob
1    jack
2    rose
dtype: object
```

Si todos los elementos de la Serie son cadenas, hay una forma más fácil de aplicar métodos de cadena:

```
>>> q.str.lower()
0    bob
1    jack
2    rose
dtype: object
>>> q.str.len()
0     3
1     4
2     4
```

Lea Serie en línea: <https://riptutorial.com/es/pandas/topic/1898/serie>

# Capítulo 37: Tipos de datos

## Observaciones

Los tipos no son nativos de los pandas. Son el resultado de pandas cerca del acoplamiento arquitectónico para adormecer.

el dtype de una columna no tiene que correlacionarse de ninguna manera con el tipo python del objeto contenido en la columna.

Aquí tenemos un `pd.Series` con flotadores. El dtype será `float`.

Luego usamos `astype` para "astype" a objeto.

```
pd.Series([1.,2.,3.,4.,5.]).astype(object)
0      1
1      2
2      3
3      4
4      5
dtype: object
```

El dtype ahora es objeto, pero los objetos en la lista todavía son flotantes. Lógico si sabes que en Python, todo es un objeto, y se puede actualizar al objeto.

```
type(pd.Series([1.,2.,3.,4.,5.]).astype(object)[0])
float
```

Aquí intentamos "echar" las carrozas a las cuerdas.

```
pd.Series([1.,2.,3.,4.,5.]).astype(str)
0      1.0
1      2.0
2      3.0
3      4.0
4      5.0
dtype: object
```

El dtype ahora es objeto, pero el tipo de las entradas en la lista es cadena. Esto se debe a que `numpy` no se ocupa de las cadenas y, por lo tanto, actúa como si solo fueran objetos y no preocupa.

```
type(pd.Series([1.,2.,3.,4.,5.]).astype(str)[0])
str
```

No confíe en los tipos, son un artefacto de un defecto arquitectónico en los pandas. Especifíquelos como debe, pero no confíe en el tipo de dtype establecido en una columna.

# Examples

## Comprobando los tipos de columnas

Los tipos de columnas se pueden verificar mediante `.dtypes` attribute of DataFrames.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0], 'C': [True, False, True]})

In [2]: df
Out[2]:
   A    B    C
0  1  1.0  True
1  2  2.0 False
2  3  3.0  True

In [3]: df.dtypes
Out[3]:
A      int64
B    float64
C         bool
dtype: object
```

Para una sola serie, puede usar el atributo `.dtype`.

```
In [4]: df['A'].dtype
Out[4]: dtype('int64')
```

## Cambiando dtypes

`astype()` método `astype()` cambia el tipo de letra de una serie y devuelve una nueva serie.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0],
                          'C': ['1.1.2010', '2.1.2011', '3.1.2011'],
                          'D': ['1 days', '2 days', '3 days'],
                          'E': ['1', '2', '3']})

In [2]: df
Out[2]:
   A    B    C    D  E
0  1  1.0  1.1.2010  1 days  1
1  2  2.0  2.1.2011  2 days  2
2  3  3.0  3.1.2011  3 days  3

In [3]: df.dtypes
Out[3]:
A      int64
B    float64
C    object
D    object
E    object
dtype: object
```

Cambie el tipo de columna A a flotante y el tipo de columna B a entero:

```
In [4]: df['A'].astype('float')
```

```
Out[4]:
0    1.0
1    2.0
2    3.0
Name: A, dtype: float64

In [5]: df['B'].astype('int')
Out[5]:
0    1
1    2
2    3
Name: B, dtype: int32
```

`astype()` **método** `astype()` es para conversión de tipo específico (es decir, puede especificar `.astype(float64)` , `.astype(float32)` o `.astype(float16)` ). Para la conversión general, puede usar `pd.to_numeric` , `pd.to_datetime` y `pd.to_timedelta` .

## Cambiando el tipo a numérico

`pd.to_numeric` cambia los valores a un tipo numérico.

```
In [6]: pd.to_numeric(df['E'])
Out[6]:
0    1
1    2
2    3
Name: E, dtype: int64
```

De forma predeterminada, `pd.to_numeric` genera un error si una entrada no se puede convertir en un número. Puedes cambiar ese comportamiento usando el parámetro de `errors` .

```
# Ignore the error, return the original input if it cannot be converted
In [7]: pd.to_numeric(pd.Series(['1', '2', 'a']), errors='ignore')
Out[7]:
0    1
1    2
2    a
dtype: object

# Return NaN when the input cannot be converted to a number
In [8]: pd.to_numeric(pd.Series(['1', '2', 'a']), errors='coerce')
Out[8]:
0    1.0
1    2.0
2    NaN
dtype: float64
```

Si es necesario, compruebe que todas las filas con entrada no se pueden convertir a [boolean indexing](#) uso numérico con `isnull` :

```
In [9]: df = pd.DataFrame({'A': [1, 'x', 'z'],
                           'B': [1.0, 2.0, 3.0],
                           'C': [True, False, True]})
```



```
In [10]: pd.to_numeric(df.A, errors='coerce').isnull()
Out[10]:
0    False
1     True
2     True
Name: A, dtype: bool

In [11]: df[pd.to_numeric(df.A, errors='coerce').isnull()]
Out[11]:
   A    B    C
1  x  2.0 False
2  z  3.0  True
```

## Cambiando el tipo a datetime

```
In [12]: pd.to_datetime(df['C'])
Out[12]:
0    2010-01-01
1    2011-02-01
2    2011-03-01
Name: C, dtype: datetime64[ns]
```

Tenga en cuenta que 2.1.2011 se convierte al 1 de febrero de 2011. Si desea el 2 de enero de 2011, debe utilizar el parámetro `dayfirst`.

```
In [13]: pd.to_datetime('2.1.2011', dayfirst=True)
Out[13]: Timestamp('2011-01-02 00:00:00')
```

## Cambiando el tipo a timedelta

```
In [14]: pd.to_timedelta(df['D'])
Out[14]:
0    1 days
1    2 days
2    3 days
Name: D, dtype: timedelta64[ns]
```

## Seleccionando columnas basadas en dtype

El método `select_dtypes` se puede utilizar para seleccionar columnas basadas en `dtype`.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0], 'C': ['a', 'b', 'c'],
                          'D': [True, False, True]})

In [2]: df
Out[2]:
   A    B  C    D
0  1  1.0  a  True
1  2  2.0  b False
2  3  3.0  c  True
```

Con los parámetros de `include` y `exclude`, puede especificar qué tipos desea:

```
# Select numbers
In [3]: df.select_dtypes(include=['number']) # You need to use a list
Out[3]:
   A    B
0  1  1.0
1  2  2.0
2  3  3.0

# Select numbers and booleans
In [4]: df.select_dtypes(include=['number', 'bool'])
Out[4]:
   A    B    D
0  1  1.0  True
1  2  2.0 False
2  3  3.0  True

# Select numbers and booleans but exclude int64
In [5]: df.select_dtypes(include=['number', 'bool'], exclude=['int64'])
Out[5]:
   B    D
0  1.0  True
1  2.0 False
2  3.0  True
```

## Resumiendo dtypes

`get_dtype_counts` método `get_dtype_counts` se puede usar para ver un desglose de los tipos.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0], 'C': ['a', 'b', 'c'],
                          'D': [True, False, True]})

In [2]: df.get_dtype_counts()
Out[2]:
bool      1
float64    1
int64      1
object     1
dtype: int64
```

Lea Tipos de datos en línea: <https://riptutorial.com/es/pandas/topic/2959/tipos-de-datos>

---

# Capítulo 38: Trabajando con series de tiempo

## Examples

### Creación de series de tiempo

Aquí es cómo crear una serie de tiempo simple.

```
import pandas as pd
import numpy as np

# The number of sample to generate
nb_sample = 100

# Seeding to obtain a reproducible dataset
np.random.seed(0)

se = pd.Series(np.random.randint(0, 100, nb_sample),
               index = pd.date_range(start = pd.to_datetime('2016-09-24'),
                                     periods = nb_sample, freq='D'))

se.head(2)

# 2016-09-24    44
# 2016-09-25    47

se.tail(2)

# 2016-12-31    85
# 2017-01-01    48
```

### Indización parcial de cuerdas

Una forma muy útil de subconjuntar series temporales es usar **la indexación parcial de cadenas**. Permite seleccionar el rango de fechas con una sintaxis clara.

---

## Obteniendo datos

Estamos utilizando el conjunto de datos en el ejemplo de [Creación de series temporales](#).

Viendo cabeza y cola para ver los límites.

```
se.head(2).append(se.tail(2))

# 2016-09-24    44
# 2016-09-25    47
# 2016-12-31    85
# 2017-01-01    48
```

# Subconjunto

Ahora podemos subdividir por año, mes, día de manera muy intuitiva.

Por año

```
se['2017']  
  
# 2017-01-01    48
```

Por mes

```
se['2017-01']  
  
# 2017-01-01    48
```

Por día

```
se['2017-01-01']  
  
# 48
```

Con un rango de año, mes, día de acuerdo a sus necesidades.

```
se['2016-12-31':'2017-01-01']  
  
# 2016-12-31    85  
# 2017-01-01    48
```

pandas también proporciona una función `truncate` dedicada para este uso a través de los parámetros de `before` y `after` , pero creo que es menos claro.

```
se.truncate(before='2017')  
  
# 2017-01-01    48  
  
se.truncate(before='2016-12-30', after='2016-12-31')  
  
# 2016-12-30    13  
# 2016-12-31    85
```

Lea Trabajando con series de tiempo en línea:

<https://riptutorial.com/es/pandas/topic/7029/trabajando-con-series-de-tiempo>

# Capítulo 39: Tratar variables categóricas

## Examples

### Codificación instantánea con `get\_dummies`()

```
>>> df = pd.DataFrame({'Name':['John Smith', 'Mary Brown'],  
                       'Gender':['M', 'F'], 'Smoker':['Y', 'N']})  
>>> print(df)
```

	Gender	Name	Smoker
0	M	John Smith	Y
1	F	Mary Brown	N

```
>>> df_with_dummies = pd.get_dummies(df, columns=['Gender', 'Smoker'])  
>>> print(df_with_dummies)
```

	Name	Gender_F	Gender_M	Smoker_N	Smoker_Y
0	John Smith	0.0	1.0	0.0	1.0
1	Mary Brown	1.0	0.0	1.0	0.0

Lea Tratar variables categóricas en línea: <https://riptutorial.com/es/pandas/topic/5999/tratar-variables-categoricas>

# Capítulo 40: Uso de .ix, .iloc, .loc, .at y .iat para acceder a un DataFrame

## Examples

### Utilizando .iloc

.iloc usa números enteros para leer y escribir datos en un DataFrame.

Primero, vamos a crear un DataFrame:

```
df = pd.DataFrame({'one': [1, 2, 3, 4, 5],
                   'two': [6, 7, 8, 9, 10],
                   }, index=['a', 'b', 'c', 'd', 'e'])
```

Este DataFrame se ve como:

	one	two
a	1	6
b	2	7
c	3	8
d	4	9
e	5	10

Ahora podemos usar .iloc para leer y escribir valores. Vamos a leer la primera fila, primera columna:

```
print df.iloc[0, 0]
```

Esto imprimirá:

```
1
```

También podemos establecer valores. Permite establecer la segunda columna, segunda fila a algo nuevo:

```
df.iloc[1, 1] = '21'
```

Y luego echar un vistazo para ver qué pasó:

```
print df
```

	one	two
a	1	6
b	2	21
c	3	8
d	4	9

```
e    5    10
```

## Utilizando .loc

.loc usa **etiquetas** para leer y escribir datos.

Vamos a configurar un DataFrame:

```
df = pd.DataFrame({'one': [1, 2, 3, 4, 5],  
                  'two': [6, 7, 8, 9, 10],  
                  }, index=['a', 'b', 'c', 'd', 'e'])
```

Luego podemos imprimir el marco de datos para ver la forma:

```
print df
```

Esto dará salida

	one	two
a	1	6
b	2	7
c	3	8
d	4	9
e	5	10

Usamos las **etiquetas de** columna y fila para acceder a los datos con .loc. Fijemos la fila 'c', columna 'dos' al valor 33:

```
df.loc['c', 'two'] = 33
```

Así es como se ve ahora el DataFrame:

	one	two
a	1	6
b	2	7
c	3	33
d	4	9
e	5	10

Es de destacar que el uso de `df['two'].loc['c'] = 33` puede no informar una advertencia, e incluso puede funcionar, sin embargo, usar `df.loc['c', 'two']` está garantizado para funcionar correctamente , mientras que el primero no lo es.

Podemos leer porciones de datos, por ejemplo

```
print df.loc['a':'c']
```

imprimirá las filas a a c. Esto es inclusivo.

	one	two
a	1	6
b	2	7
c	3	8

Y finalmente, podemos hacer las dos cosas juntos:

```
print df.loc['b':'d', 'two']
```

Saldrá las filas b a c de la columna 'dos'. Observe que la etiqueta de la columna no está impresa.

b	7
c	8
d	9

Si `.loc` se suministra con un argumento entero que no es una etiqueta, vuelve a la indexación entera de los ejes (el comportamiento de `.iloc`). Esto hace posible la indexación de etiquetas y enteros mixtos:

```
df.loc['b', 1]
```

devolverá el valor en la segunda columna (el índice comienza en 0) en la fila 'b':

```
7
```

Lea [Uso de .ix, .iloc, .loc, .at y .iat para acceder a un DataFrame en línea](https://riptutorial.com/es/pandas/topic/7074/uso-de--ix---iloc---loc---at-y--iat-para-acceder-a-un-dataframe):

<https://riptutorial.com/es/pandas/topic/7074/uso-de--ix---iloc---loc---at-y--iat-para-acceder-a-un-dataframe>



# Capítulo 41: Valores del mapa

## Observaciones

se debe mencionar que si el valor de la clave no existe, esto generará `KeyError`, en esas situaciones tal vez sea mejor usar la `merge` o la `get` que le permite especificar un valor predeterminado si la clave no existe

## Examples

### Mapa del Diccionario

A partir de un `df` marco de datos:

```
U    L
111  en
112  en
112  es
113  es
113  ja
113  zh
114  es
```

Imagina que quieres agregar una nueva columna llamada `s` tomando valores del siguiente diccionario:

```
d = {112: 'en', 113: 'es', 114: 'es', 111: 'en'}
```

Puede usar el `map` para realizar una búsqueda en las claves que devuelven los valores correspondientes como una nueva columna:

```
df['S'] = df['U'].map(d)
```

que devuelve:

```
U    L    S
111  en  en
112  en  en
112  es  en
113  es  es
113  ja  es
113  zh  es
114  es  es
```

Lea Valores del mapa en línea: <https://riptutorial.com/es/pandas/topic/3928/valores-del-mapa>

# Creditos

S. No	Capítulos	Contributors
1	Empezando con los pandas	<a href="#">Alexander</a> , <a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">Bryce Frank</a> , <a href="#">Community</a> , <a href="#">hashcode55</a> , <a href="#">Nikita Pestrov</a> , <a href="#">user2314737</a>
2	Agrupar datos de series de tiempo	<a href="#">ayhan</a> , <a href="#">piRSquared</a>
3	Análisis: Reunirlo todo y tomar decisiones.	<a href="#">piRSquared</a>
4	Anexando a DataFrame	<a href="#">shahins</a>
5	Calendarios de vacaciones	<a href="#">Romain</a>
6	Creando marcos de datos	<a href="#">Ahamed Mustafa M</a> , <a href="#">Alexander</a> , <a href="#">ayhan</a> , <a href="#">Ayush Kumar Singh</a> , <a href="#">bernie</a> , <a href="#">Gal Dreiman</a> , <a href="#">Geeklhern</a> , <a href="#">Gorkem Ozkaya</a> , <a href="#">jasimpson</a> , <a href="#">jezrael</a> , <a href="#">JJD</a> , <a href="#">Julien Marrec</a> , <a href="#">MaxU</a> , <a href="#">Merlin</a> , <a href="#">pylang</a> , <a href="#">Romain</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a> , <a href="#">vaerek</a> , <a href="#">ysearka</a>
7	Datos categóricos	<a href="#">jezrael</a> , <a href="#">Julien Marrec</a>
8	Datos de agrupación	<a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">danio</a> , <a href="#">Geeklhern</a> , <a href="#">jezrael</a> , <a href="#">NooBx</a> , <a href="#">QM.py</a> , <a href="#">Romain</a> , <a href="#">user2314737</a>
9	Datos duplicados	<a href="#">ayhan</a> , <a href="#">Ayush Kumar Singh</a> , <a href="#">bee-sting</a> , <a href="#">jezrael</a>
10	Datos perdidos	<a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">EdChum</a> , <a href="#">jezrael</a> , <a href="#">Zdenek</a>
11	Desplazamiento y desplazamiento de datos	<a href="#">ASGM</a>
12	Fusionar, unir y concatenar	<a href="#">ayhan</a> , <a href="#">Josh Garlitos</a> , <a href="#">MaThMaX</a> , <a href="#">MaxU</a> , <a href="#">piRSquared</a> , <a href="#">SerialDev</a> , <a href="#">varunsinghal</a>
13	Gotchas de pandas	<a href="#">vlad.rad</a>
14	Gráficos y visualizaciones	<a href="#">Ami Tavory</a> , <a href="#">Nikita Pestrov</a> , <a href="#">Scimonster</a>
15	Guardar pandas	<a href="#">amin</a> , <a href="#">bernie</a> , <a href="#">eraoul</a> , <a href="#">Gal Dreiman</a> , <a href="#">maxliving</a> , <a href="#">Musafir Safwan</a> ,

	dataframe en un archivo csv	<a href="#">Nikita Pestrov</a> , <a href="#">Olel Daniel</a> , <a href="#">Stephan</a>
16	Herramientas computacionales	<a href="#">Ami Tavory</a>
17	Herramientas de Pandas IO (leer y guardar conjuntos de datos)	<a href="#">amin</a> , <a href="#">Andy Hayden</a> , <a href="#">bernie</a> , <a href="#">Fabich</a> , <a href="#">Gal Dreiman</a> , <a href="#">jezrael</a> , <a href="#">João Almeida</a> , <a href="#">Julien Spronck</a> , <a href="#">MaxU</a> , <a href="#">Nikita Pestrov</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a>
18	Indexación booleana de marcos de datos	<a href="#">firelynx</a>
19	Indexación y selección de datos.	<a href="#">amin</a> , <a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">double0darbo</a> , <a href="#">jasimpson</a> , <a href="#">jezrael</a> , <a href="#">Joseph Dasenbrock</a> , <a href="#">MaxU</a> , <a href="#">Merlin</a> , <a href="#">piRSquared</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a>
20	IO para Google BigQuery	<a href="#">ayhan</a> , <a href="#">tworec</a>
21	JSON	<a href="#">PinoSan</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a>
22	Leer MySQL a DataFrame	<a href="#">andyabel</a> , <a href="#">rrawat</a>
23	Leer SQL Server a Dataframe	<a href="#">bernie</a> , <a href="#">SerialDev</a>
24	Leyendo archivos en pandas DataFrame	<a href="#">Arthur Camara</a> , <a href="#">bee-sting</a> , <a href="#">Corey Petty</a> , <a href="#">Sirajus Salayhin</a>
25	Making Pandas Play Nice con tipos de datos nativos de Python	<a href="#">DataSwede</a>
26	Manipulación de cuerdas	<a href="#">ayhan</a> , <a href="#">mnoronha</a> , <a href="#">SerialDev</a>
27	Manipulación sencilla de DataFrames.	<a href="#">Alexander</a> , <a href="#">ayhan</a> , <a href="#">Ayush Kumar Singh</a> , <a href="#">Gal Dreiman</a> , <a href="#">Geeklhern</a> , <a href="#">MaxU</a> , <a href="#">paulo.filip3</a> , <a href="#">R.M.</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a> , <a href="#">ysearka</a>
28	Meta: Pautas de documentación.	<a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">Stephen Leppik</a>
29	Multiindex	<a href="#">Andy Hayden</a> , <a href="#">benten</a> , <a href="#">danielhadar</a> , <a href="#">danio</a> , <a href="#">Pedro M Duarte</a>

30	Obteniendo información sobre DataFrames	<a href="#">Alexander</a> , <a href="#">ayhan</a> , <a href="#">Ayush Kumar Singh</a> , <a href="#">bernie</a> , <a href="#">Romain</a> , <a href="#">ysearka</a>
31	Pandas Datareader	<a href="#">Alexander</a> , <a href="#">MaxU</a>
32	pd.DataFrame.apply	<a href="#">ptsw</a> , <a href="#">Romain</a>
33	Remodelación y pivotamiento	<a href="#">Albert Camps</a> , <a href="#">ayhan</a> , <a href="#">bernie</a> , <a href="#">DataSwede</a> , <a href="#">jezrael</a> , <a href="#">MaxU</a> , <a href="#">Merlin</a>
34	Remuestreo	<a href="#">jezrael</a>
35	Secciones transversales de diferentes ejes con MultiIndex.	<a href="#">Julien Marrec</a>
36	Serie	<a href="#">Alexander</a> , <a href="#">daphshez</a> , <a href="#">EdChum</a> , <a href="#">jezrael</a> , <a href="#">shahins</a>
37	Tipos de datos	<a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">firelynx</a> , <a href="#">jezrael</a>
38	Trabajando con series de tiempo	<a href="#">Romain</a>
39	Tratar variables categóricas	<a href="#">Gorkem Ozkaya</a>
40	Uso de .ix, .iloc, .loc, .at y .iat para acceder a un DataFrame	<a href="#">bee-sting</a> , <a href="#">DataSwede</a> , <a href="#">farleytpm</a>
41	Valores del mapa	<a href="#">EdChum</a> , <a href="#">Fabio Lamanna</a>