EBook Gratuito

APPRENDIMENTO pandas

Free unaffiliated eBook created from
Stack Overflow contributors.

#pandas

Sommario

Di

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.

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

Capitolo 1: Iniziare con i panda

Osservazioni

Pandas è un pacchetto Python che fornisce strutture di dati veloci, flessibili ed espressive progettate per rendere il lavoro con dati "relazionali" o "etichettati" sia facile che intuitivo. Mira ad essere il blocco fondamentale di alto livello per fare analisi pratiche dei dati reali in Python.

La documentazione ufficiale di Pandas può essere trovata qui .

Versioni

Pandas

Versione	Data di rilascio
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

Versione	Data di rilascio
0.12.0	2013/07/23

Examples

Installazione o configurazione

Istruzioni dettagliate su come installare o installare i panda possono essere trovate qui nella documentazione ufficiale .

Installazione di panda con Anaconda

Installare panda e il resto dello stack NumPy e SciPy può essere un po 'difficile per gli utenti inesperti.

Il modo più semplice per installare non solo i panda, ma Python e i pacchetti più popolari che compongono lo stack SciPy (IPython, NumPy, Matplotlib, ...) è con Anaconda , una piattaforma multipiattaforma (Linux, Mac OS X, Windows) Distribuzione Python per analisi dei dati e calcolo scientifico.

Dopo aver eseguito un semplice programma di installazione, l'utente avrà accesso ai panda e al resto dello stack SciPy senza bisogno di installare altro e senza dover attendere la compilazione di alcun software.

Le istruzioni di installazione per Anaconda possono essere trovate qui .

Un elenco completo dei pacchetti disponibili come parte della distribuzione di Anaconda può essere trovato qui .

Un ulteriore vantaggio dell'installazione con Anaconda è che non è necessario disporre dei diritti di amministratore per installarlo, verrà installato nella home directory dell'utente e ciò rende inoltre banale l'eliminazione di Anaconda in un secondo momento (basta eliminare tale cartella).

Installazione dei panda con Miniconda

La sezione precedente delineava come ottenere i panda installati come parte della distribuzione di Anaconda. Tuttavia, questo approccio implica l'installazione di oltre un centinaio di pacchetti e comporta il download del programma di installazione di poche centinaia di megabyte.

Se vuoi avere più controllo su quali pacchetti, o avere una larghezza di banda internet limitata, installare dei panda con Miniconda potrebbe essere una soluzione migliore.

Conda è il gestore di pacchetti su cui è costruita la distribuzione di Anaconda. È un gestore di pacchetti che è indipendente dalla piattaforma e dalla lingua (può giocare un ruolo simile a una combinazione pip e virtualenv).

Miniconda consente di creare un'installazione Python autonoma e minimale, quindi utilizzare il comando Conda per installare pacchetti aggiuntivi.

Innanzitutto avrai bisogno di Conda per essere installato e il download e l'esecuzione di Miniconda lo faranno per te. L'installer può essere trovato qui .

Il passo successivo è quello di creare un nuovo ambiente conda (questi sono analoghi a un virtualenv ma consentono anche di specificare con precisione quale versione Python installare anche). Esegui i seguenti comandi da una finestra di terminale:

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

Questo creerà un ambiente minimale con solo Python installato in esso. Per metterti dentro questo ambiente, corri:

```
source activate name_of_my_env
```

Su Windows il comando è:

```
activate name_of_my_env
```

Il passaggio finale richiesto è installare i panda. Questo può essere fatto con il seguente comando:

```
conda install pandas
```

Per installare una versione di panda specifica:

```
conda install pandas=0.13.1
```

Per installare altri pacchetti, ad esempio IPython:

```
conda install ipython
```

Per installare la distribuzione completa di Anaconda:

```
conda install anaconda
```

Se hai bisogno di pacchetti disponibili per pip ma non di conda, installa semplicemente pip e usa pip per installare questi pacchetti:

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

Di solito, si installano i panda con uno dei gestori di pacchetti.

esempio pip:

```
pip install pandas
```

Ciò richiederà probabilmente l'installazione di un numero di dipendenze, tra cui NumPy, richiederà

un compilatore per compilare i bit di codice richiesti e può richiedere alcuni minuti per essere completato.

Installa via anaconda

Per prima cosa scarica anaconda dal sito Continuum. Tramite il programma di installazione grafico (Windows / OSX) o eseguendo uno script di shell (OSX / Linux). Questo include i panda!

Se non vuoi che i 150 pacchetti siano comodamente raggruppati in anaconda, puoi installare miniconda . Tramite il programma di installazione grafico (Windows) o lo script di shell (OSX / Linux).

Installa i panda su miniconda usando:

```
conda install pandas
```

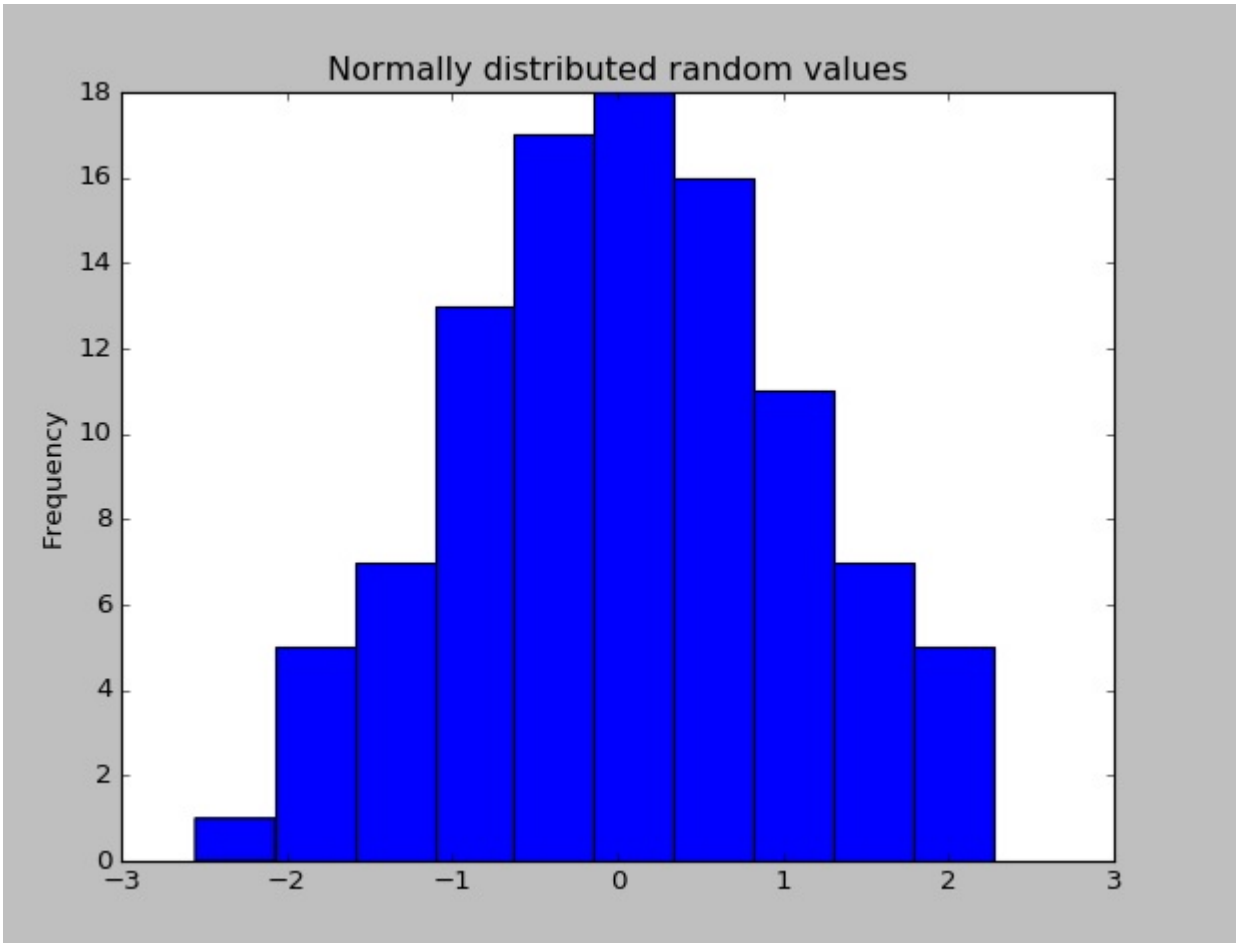
Per aggiornare i panda all'ultima versione in anaconda o miniconda usare:

```
conda update pandas
```

Ciao mondo

Una volta installato Pandas, è possibile verificare se funziona correttamente creando un set di dati con valori distribuiti casualmente e tracciando il proprio istogramma.

```
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()
```



Controlla alcune delle statistiche dei dati (media, deviazione standard, ecc.)

```
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
```

Statistiche descrittive

Le statistiche descrittive (media, deviazione standard, numero di osservazioni, minimo, massimo e quartili) di colonne numeriche possono essere calcolate utilizzando il metodo `.describe()` , che restituisce un dataframe panda di statistiche descrittive.

```
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
```

Si noti che poiché `C` non è una colonna numerica, è esclusa dall'output.

```
In [4]: df['C'].describe()
Out[4]:
count     10
unique     3
freq       5
Name: C, dtype: object
```

In questo caso il metodo riepiloga i dati categoriali per numero di osservazioni, numero di elementi unici, modalità e frequenza della modalità.

Leggi Iniziare con i panda online: https://riptutorial.com/it/pandas/topic/796/iniziare-con-i-panda

Capitolo 2: Aggiunta a DataFrame

Examples

Aggiunta di una nuova riga 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: []
```

Aggiunta di una riga per un singolo valore di colonna:

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

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

Aggiunta di una riga, dato un elenco di valori:

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

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

Aggiunta di una riga a un dizionario:

```
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
```

Il primo input in .loc [] è l'indice. Se si utilizza un indice esistente, si sovrascriveranno i valori in tale riga:

```
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
```

Aggiungi un DataFrame ad un altro DataFrame

Supponiamo di avere i seguenti due DataFram:

```
In [7]: df1
Out[7]:
    A   B
0  a1  b1
1  a2  b2

In [8]: df2
Out[8]:
    B   C
0  b1  c1
```

Non è necessario che i due DataFrames abbiano lo stesso set di colonne. Il metodo append non modifica nessuno dei DataFrames originali. Invece, restituisce un nuovo DataFrame aggiungendo i due originali. Aggiungere un DataFrame a un altro è abbastanza semplice:

```
In [9]: df1.append(df2)
Out[9]:
     A   B    C
0   a1  b1  NaN
1   a2  b2  NaN
0  NaN  b1   c1
```

Come puoi vedere, è possibile avere indici duplicati (0 in questo esempio). Per evitare questo problema, puoi chiedere a Panda di reindicizzare il nuovo DataFrame per te:

```
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
```

Leggi Aggiunta a DataFrame online: https://riptutorial.com/it/pandas/topic/6456/aggiunta-a-

dataframe

Capitolo 3: Analisi: riunire tutto e prendere decisioni

Examples

Analisi quintile: con dati casuali

L'analisi a quintile è una struttura comune per valutare l'efficacia dei fattori di sicurezza.

Qual è un fattore

Un fattore è un metodo per classificare / classificare insiemi di titoli. Per un particolare momento nel tempo e per un particolare insieme di titoli, un fattore può essere rappresentato come una serie di panda in cui l'indice è una matrice degli identificatori di sicurezza ei valori sono i punteggi o i gradi.

Se prendiamo i punteggi dei fattori nel tempo, possiamo, in ogni momento, dividere il set di titoli in 5 bucket uguali, o quintili, in base all'ordine dei punteggi dei fattori. Non c'è nulla di particolarmente sacro nel numero 5. Avremmo potuto usare 3 o 10. Ma usiamo 5 spesso. Infine, monitoriamo le prestazioni di ciascuno dei cinque bucket per determinare se esiste una differenza significativa nei rendimenti. Tendiamo a concentrarci più intensamente sulla differenza nei rendimenti del bucket con il rango più alto rispetto a quello del rango più basso.

Iniziamo impostando alcuni parametri e generando dati casuali.

Per facilitare la sperimentazione con la meccanica, forniamo un codice semplice per creare dati casuali per darci un'idea di come funzioni.

Include dati casuali

- **Resi** : genera rendimenti casuali per il numero specificato di titoli e periodi.
- **Segnali** : generano segnali casuali per il numero specificato di titoli e periodi e con il livello prescritto di correlazione con i **rendimenti** . Affinché un fattore sia utile, deve esserci qualche informazione o correlazione tra i punteggi / ranghi e i successivi rendimenti. Se non ci fosse una correlazione, lo vedremmo. Sarebbe un buon esercizio per il lettore, duplicare questa analisi con dati casuali generati con `0` correlazione.

Inizializzazione

```
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
```

Ora generiamo un indice delle serie temporali e un indice che rappresenta gli ID di sicurezza. Quindi usali per creare dataframes per ritorni e segnali

```
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]` per `25` per ridimensionare a qualcosa che assomiglia a rendimenti azionari. Aggiungo anche `1e-7` per dare un modesto ritorno medio positivo.

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

`pd.qcut` - Crea `pd.qcut`

Usiamo `pd.qcut` per dividere i miei segnali in bucket quintili per ogni periodo.

```
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)
```

Usa questi tagli come indice dei nostri rendimenti

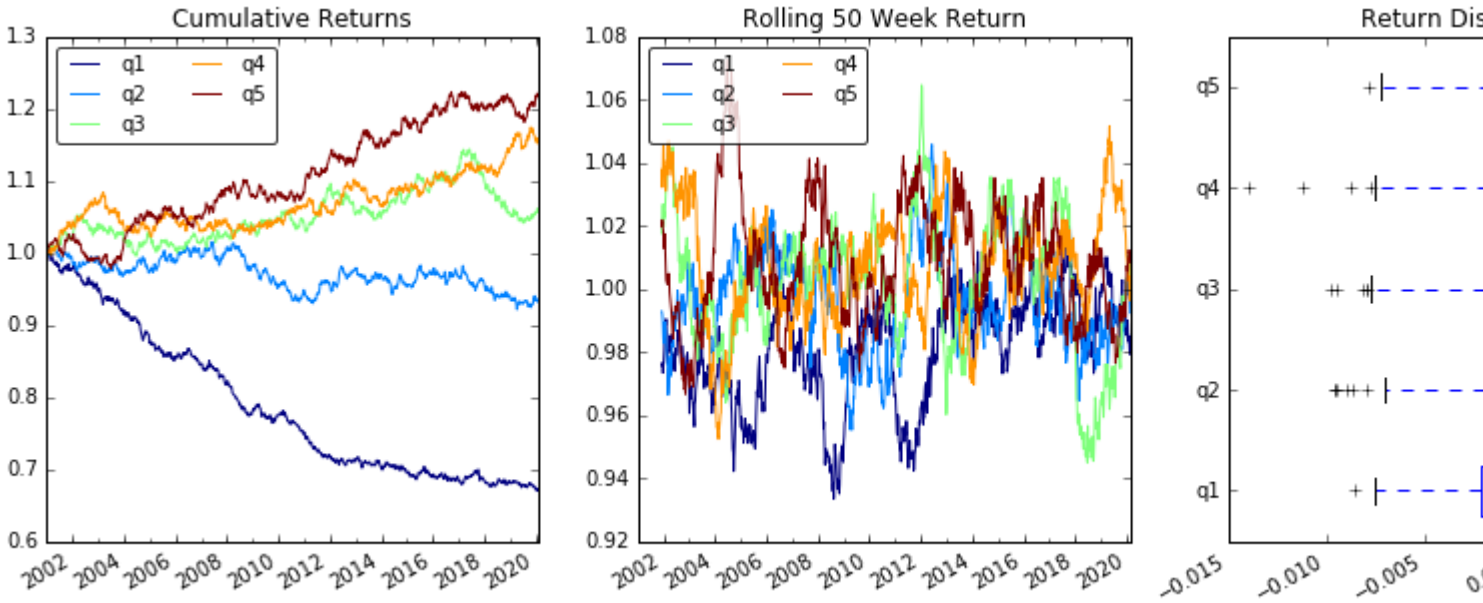
```
returns_cut = security_returns.stack().rename('returns') \
    .to_frame().set_index(cut, append=True) \
    .swaplevel(2, 1).sort_index().squeeze() \
    .groupby(level=[0, 1]).mean().unstack()
```

Analisi

Riporta il grafico

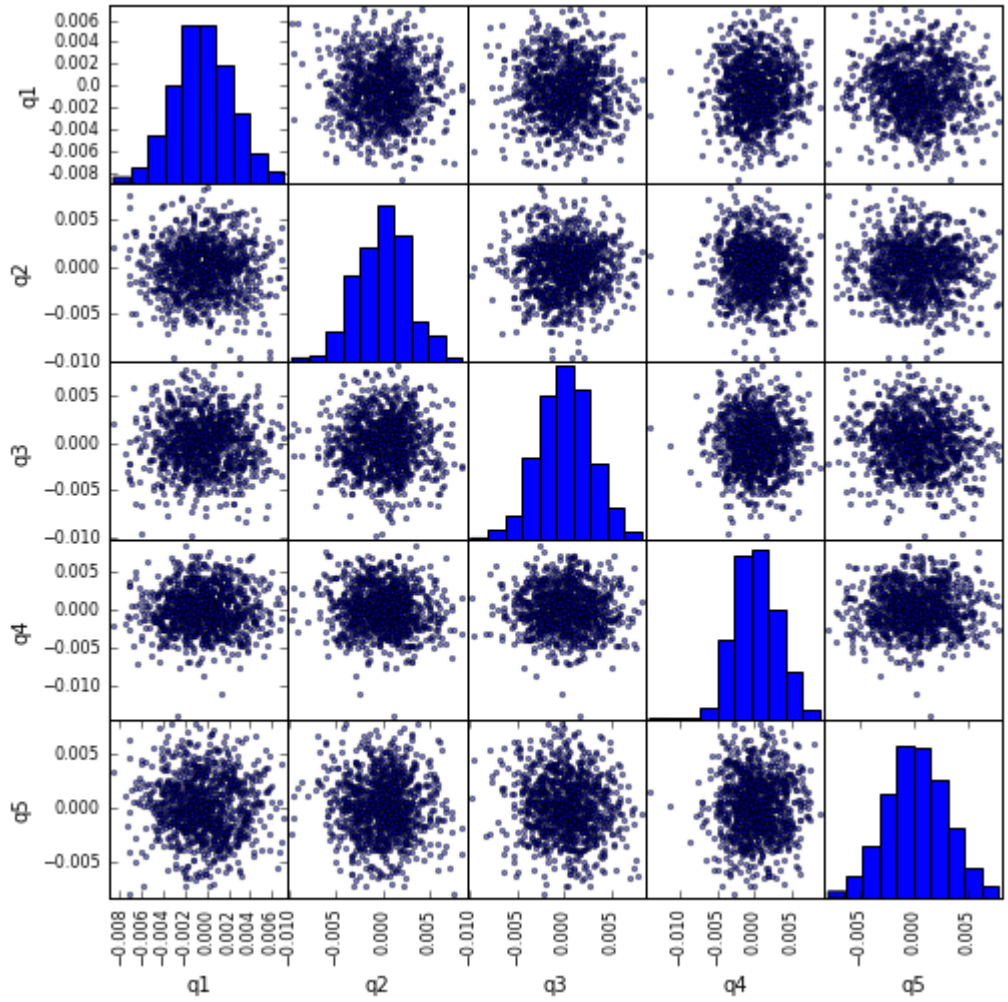
```
import matplotlib.pyplot as plt
```

```
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(vert=False, ax=ax3, title="Return Distribution")

fig.autofmt_xdate()

plt.show()
```



Visualizza la correlazione `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()
```



Calcola e visualizza Maximum 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.argmin()
    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)
```

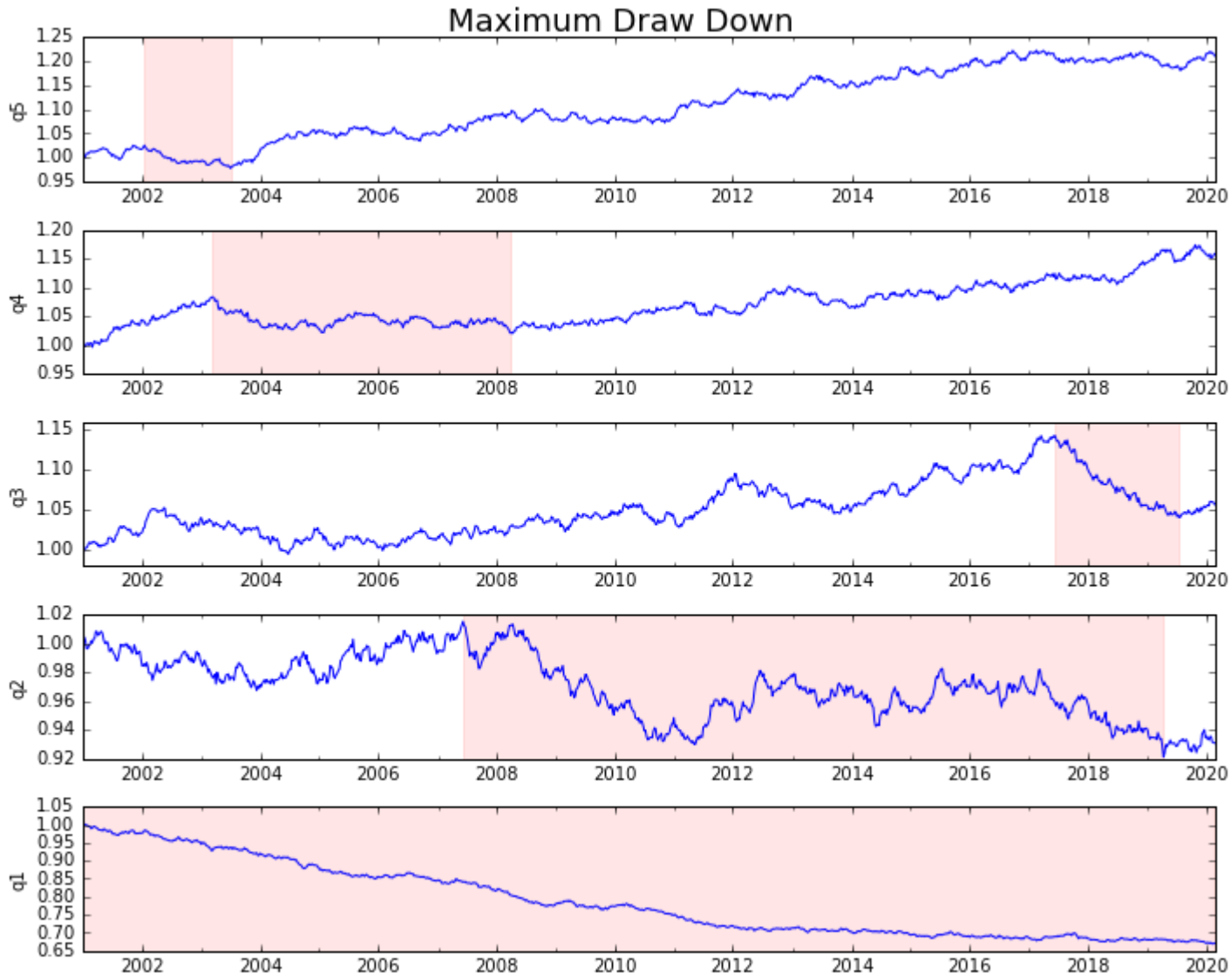
Cosa sembra questo

```
max_dd_df(returns_cut)
```

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

Tracciamolo

```
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)
```



Calcola statistiche

Esistono molte statistiche potenziali che possiamo includere. Qui di seguito sono solo alcuni, ma dimostra come semplicemente possiamo incorporare nuove statistiche nel nostro sommario.

```
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')
```

Finiremo per utilizzare solo la funzione `describe` mentre riunisce tutti gli altri.

```
describe(returns_cut)
```

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

Questo non è inteso per essere completo. È pensato per riunire molte delle caratteristiche dei panda e dimostrare come puoi usarlo per rispondere a domande importanti per te. Questo è un sottoinsieme dei tipi di metriche che uso per valutare l'efficacia dei fattori quantitativi.

Leggi Analisi: riunire tutto e prendere decisioni online:
https://riptutorial.com/it/pandas/topic/5238/analisi--riunire-tutto-e-prendere-decisioni

Capitolo 4: Calendari delle festività

Examples

Crea un calendario personalizzato

Ecco come creare un calendario personalizzato. L'esempio dato è un calendario francese - quindi fornisce molti esempi.

```
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 personalizzato

Ecco come utilizzare il calendario personalizzato.

Ricevi le vacanze tra due date

```
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)
```

Contare il numero di giorni lavorativi tra due date

A volte è utile ottenere il numero di giorni lavorativi per mese qualunque sia l'anno nel futuro o nel passato. Ecco come farlo con un calendario personalizzato.

```
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
```

Leggi Calendari delle festività online: https://riptutorial.com/it/pandas/topic/7976/calendari-delle-festivita

Capitolo 5: Creazione di DataFrames

introduzione

DataFrame è una struttura dati fornita dalla libreria pandas, oltre a *Series* & *Panel* . È una struttura bidimensionale e può essere paragonata a una tabella di righe e colonne.

Ogni riga può essere identificata da un indice intero (0..N) o un'etichetta impostata in modo esplicito durante la creazione di un oggetto DataFrame. Ogni colonna può essere di tipo distinto ed è identificata da un'etichetta.

Questo argomento copre vari modi per costruire / creare un oggetto DataFrame. Ex. dagli array di Numpy, dalla lista delle tuple, dal dizionario.

Examples

Creare un campione DataFrame

```
import pandas as pd
```

Crea un DataFrame da un dizionario, contenente due colonne: `numbers` e `colors` . Ogni chiave rappresenta un nome di colonna e il valore è una serie di dati, il contenuto della colonna:

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

Mostra contenuti di dataframe:

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

I panda ordinano le colonne in ordine alfabetico, in quanto non vengono ordinate le `dict` . Per specificare l'ordine, utilizzare il parametro `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 campione DataFrame usando Numpy

Crea un `DataFrame` di numeri casuali:

```
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
```

Crea un `DataFrame` con numeri interi:

```
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
```

Crea un `DataFrame` e includi nans (`NaT, NaN, 'nan', None`) su colonne e righe:

```
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
```

Crea un campione DataFrame da più raccolte utilizzando Dizionario

```
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
```

Crea un DataFrame da un elenco di tuple

È possibile creare un DataFrame da un elenco di tuple semplici e persino scegliere gli elementi specifici delle tuple che si desidera utilizzare. Qui creeremo un DataFrame utilizzando tutti i dati di ogni tupla tranne l'ultimo 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
```

Crea un DataFrame da un dizionario di liste

Crea un DataFrame da più elenchi passando un dict i cui valori sono elencati. Le chiavi del dizionario sono usate come etichette di colonne. Le liste possono anche essere ndarrays. Gli elenchi / ndarray devono essere tutti della stessa lunghezza.

```
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
```

Se gli array non sono della stessa lunghezza viene generato un errore

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

Utilizzando 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
```

Vedi ulteriori dettagli su: http://pandas.pydata.org/pandas-docs/stable/dsintro.html#from-dict-of-ndarrays-lists

Creare un campione DataFrame 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` per il parametro `freq` in `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 endfrequency
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
```

Creare un campione DataFrame con MultiIndex

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

Utilizzando `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'])
```

Utilizzando `from_product` :

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

Quindi, usa questo 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
```

Salva e carica un DataFrame nel 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)
```

Crea un DataFrame da un elenco di dizionari

Un DataFrame può essere creato da un elenco di dizionari. Le chiavi sono usate come nomi di colonne.

```
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
```

I valori mancanti sono riempiti 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
```

Leggi Creazione di DataFrames online: https://riptutorial.com/it/pandas/topic/1595/creazione-di-dataframes

Capitolo 6: Dati categoriali

introduzione

Le categorie sono un tipo di dati panda, che corrispondono a variabili categoriali nella statistica: una variabile, che può assumere solo un numero limitato e solitamente fisso di valori possibili (categorie; livelli in R). Esempi sono genere, classe sociale, gruppi sanguigni, affiliazioni paese, tempo di osservazione o valutazioni tramite scale Likert. Fonte: documenti di Pandas

Examples

Creazione di oggetti

```
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
```

Creazione di set di dati casuali di grandi dimensioni

```
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)
```

Leggi Dati categoriali online: https://riptutorial.com/it/pandas/topic/3887/dati-categoriali

Capitolo 7: Dati duplicati

Examples

Seleziona duplicato

Se è necessario impostare il valore `0` sulla colonna `B` , dove nella colonna `A` sono presenti dati duplicati, innanzitutto creare la maschera per `Series.duplicated` e quindi utilizzare `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
```

Se è necessario utilizzare la maschera invertita `~` :

```
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
```

Eliminato duplicato

Usa `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
```

Quando non vuoi ottenere una copia di un frame di dati, ma per modificare quello esistente:

```
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
```

Conta e ottieni elementi unici

Numero di elementi unici in 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
```

Ottieni elementi unici in 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
```

Numero di elementi unici in ciascun gruppo:

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

Ottieni elementi unici in ogni gruppo:

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

Ottieni valori unici da una colonna.

```
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
```

Per ottenere valori univoci nella colonna A e 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])
```

Per ottenere i valori univoci nella colonna A come elenco (notare che `unique()` può essere utilizzato in due modi leggermente diversi)

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

Ecco un esempio più complesso. Supponiamo di voler trovare i valori univoci dalla colonna "B" dove "A" è uguale a 1.

Per prima cosa, introduciamo un duplicato in modo da poter vedere come funziona. Sostituiamo il 6 nella riga '4', la colonna '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
```

Ora seleziona i dati:

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

Questo può essere suddiviso pensando innanzitutto al DataFrame interno:

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

Questo trova i valori nella colonna A che sono uguali a 1 e applica True o False ad essi. Possiamo quindi usarlo per selezionare i valori dalla colonna "B" di DataFrame (la selezione DataFrame esterna)

Per confronto, ecco la lista se non usiamo univoco. Recupera ogni valore nella colonna 'B' dove la colonna 'A' è 1

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

Leggi Dati duplicati online: https://riptutorial.com/it/pandas/topic/2082/dati-duplicati

Capitolo 8: Dati mancanti

Osservazioni

Dovremmo includere il `ffill` e il `bfill` non documentati?

Examples

Riempiendo i valori mancanti

```
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
```

Riempi i valori mancanti con un singolo valore:

```
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
```

Questo restituisce un nuovo DataFrame. Se si desidera modificare il DataFrame originale, utilizzare il parametro `inplace` (`df.fillna(0, inplace=True)`) o assegnarlo nuovamente a DataFrame originale (`df = df.fillna(0)`).

Riempi i valori mancanti con quelli precedenti:

```
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
```

Riempi con i prossimi:

```
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
```

Riempi usando un altro 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
```

Eliminazione di valori mancanti

Quando si crea un oggetto DataFrame `None` (il valore mancante di python) viene convertito in `NaN` (valore mancante di 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
```

Trascina le righe se almeno una colonna ha un valore mancante

```
In [12]: df.dropna()
Out[12]:
     0    1    2     3
2  7.0  8.0  9.0  10.0
```

Questo restituisce un nuovo DataFrame. Se si desidera modificare il DataFrame originale, utilizzare il parametro `inplace` (`df.dropna(inplace=True)`) o assegnarlo nuovamente a DataFrame

originale (`df = df.dropna()`).

Trascina le righe se mancano tutti i valori in quella riga

```
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
```

Rilascia *colonne* che non hanno almeno 3 valori non mancanti

```
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
```

interpolazione

```
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
```

Verifica dei valori mancanti

Per verificare se un valore è NaN, è possibile utilizzare le funzioni `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
```

Nota che `np.nan == np.nan` restituisce False, quindi dovresti evitare il confronto con np.nan:

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

Entrambe le funzioni sono anche definite come metodi su Serie e DataFrames.

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

Test su 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
```

Leggi Dati mancanti online: https://riptutorial.com/it/pandas/topic/1896/dati-mancanti

Capitolo 9: Gotchas di panda

Osservazioni

Gotcha in generale è un costrutto che è ben documentato, ma non intuitivo. I trucchi producono un output che normalmente non è previsto a causa del suo carattere controintuitivo.

Il pacchetto Pandas ha diversi trucchi, che possono confondere qualcuno, che non ne è a conoscenza, e alcuni di essi sono presentati in questa pagina di documentazione.

Examples

Rilevamento di valori mancanti con np.nan

Se vuoi rilevare le missioni con

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

otterrai il seguente risultato:

```
col
0    False
1    False
```

Questo perché confrontando il valore mancante con qualcosa si ottiene un Falso, invece di questo dovresti usare

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

che risulta in:

```
col
0    False
1    True
```

Integer e NA

I panda non supportano la mancanza negli attributi di tipo intero. Ad esempio, se hai delle missioni nella colonna dei voti:

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

In questo caso devi solo usare float invece di numeri interi o impostare il tipo di oggetto.

Allineamento automatico dei dati (comportamento basato sull'indice)

Se vuoi aggiungere una serie di valori [1,2] alla colonna di dataframe df, otterrai NaN:

```
import pandas as pd

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

   col
3    NaN
4    NaN
```

perché l'impostazione di una nuova colonna allinea automaticamente i dati con l'indexe, ei tuoi valori 1 e 2 otterrebbero gli indici 0 e 1, e non 3 e 4 come nel tuo data frame:

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

   col
1      2.0
2      NaN
```

Se vuoi ignorare l'indice, devi impostare i valori. Alla fine:

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

   col
3    1
4    2
```

Leggi Gotchas di panda online: https://riptutorial.com/it/pandas/topic/6425/gotchas-di-panda

Capitolo 10: Grafici e visualizzazioni

Examples

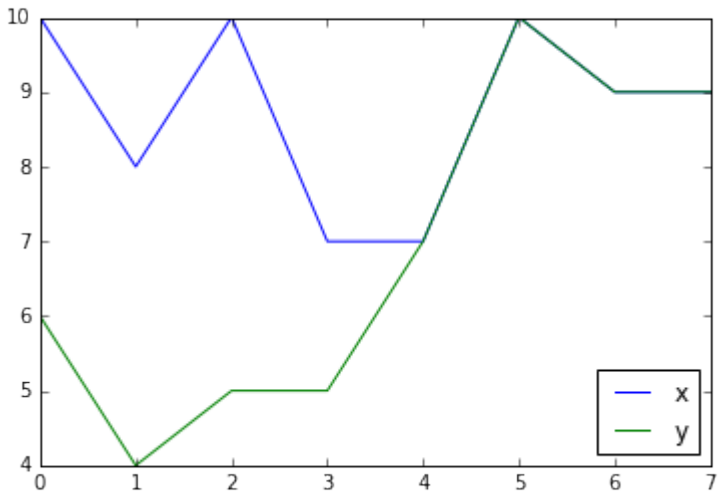
Grafici dati di base

Gli usi di Pandas forniscono diversi modi per creare grafici dei dati all'interno del frame di dati. Utilizza Matplotlib per questo scopo.

I grafici di base hanno i loro wrapper per gli oggetti DataFrame e Series:

Line Plot

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



È possibile chiamare lo stesso metodo per un oggetto Serie per tracciare un sottoinsieme del Data Frame:

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

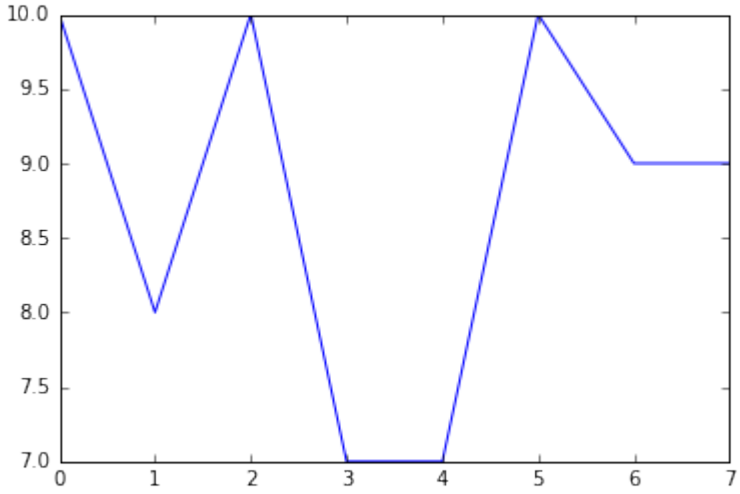
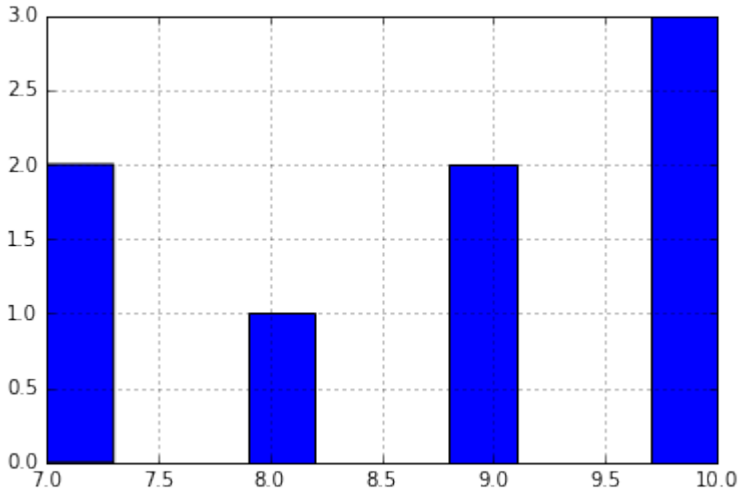


Grafico a barre

Se vuoi esplorare la distribuzione dei tuoi dati, puoi usare il metodo `hist()` .

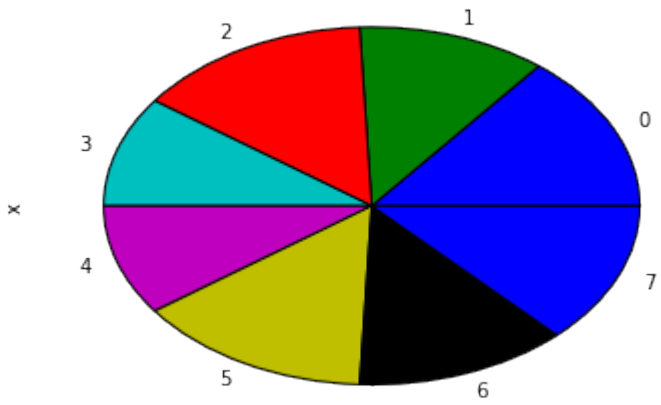
```
df['x'].hist()
```



Metodo generale per tracciare la trama ()

Tutti i possibili grafici sono disponibili attraverso il metodo di stampa. Il tipo di grafico è selezionato dall'argomento **tipo** .

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



Nota In molti ambienti, il grafico a torta uscirà da un ovale. Per renderlo un cerchio, usa il seguente:

```
from matplotlib import pyplot

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

Disegnare la trama

`plot()` può accettare argomenti che vengono passati a matplotlib per modellare la trama in modi diversi.

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

Tracciare su un asse matplotlib esistente

Di default, `plot()` crea una nuova figura ogni volta che viene chiamata. È possibile tracciare su un asse esistente passando il parametro `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
```

Leggi Grafici e visualizzazioni online: https://riptutorial.com/it/pandas/topic/3839/grafici-e-visualizzazioni

Capitolo 11: Indicizzazione booleana dei dataframes

introduzione

Accedere alle righe in un dataframe utilizzando gli oggetti `.ix` `.loc` `.ix` , `.loc` , `.iloc` e in che modo si differenzia dall'uso di una maschera booleana.

Examples

Accesso a un DataFrame con un indice booleano

Questo sarà il nostro esempio di dati:

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

Accedere con `.loc`

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

Accedere con `.iloc`

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

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

> È importante notare che le versioni precedenti dei panda non distinguevano tra input booleano e intero, quindi `.iloc[True]` restituirebbe lo stesso di `.iloc[1]`

Accesso con `.ix`

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

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

Come puoi vedere, `.ix` ha due comportamenti. Questa è una pessima pratica nel codice e quindi dovrebbe essere evitata. Si prega di utilizzare `.iloc` o `.loc` per essere più espliciti.

Applicazione di una maschera booleana ad un dataframe

Questo sarà il nostro esempio di dati:

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

Usando l' `__getitem__` magico `__getitem__` o `[]` . Dandogli una lista di True e False della stessa lunghezza del dataframe ti darà:

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

Mascheramento dei dati in base al valore della colonna

Questo sarà il nostro esempio di dati:

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

Accedendo a una singola colonna da un frame di dati, possiamo usare un semplice confronto `==` per confrontare ogni elemento della colonna con la variabile data, producendo un `pd.Series` di True e False

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

Questo `pd.Series` è un'estensione di un `np.array` che è un'estensione di un `list` semplice, quindi possiamo `__getitem__` o `[]` come nell'esempio precedente.

```
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
```

Mascherare i dati in base al valore dell'indice

Questo sarà il nostro esempio di dati:

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

Possiamo creare una maschera basata sui valori dell'indice, proprio come su un valore di colonna.

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

Ma farlo è *quasi* lo stesso di

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

La differenza importante è che, quando `.loc` incontra solo una riga nell'indice che corrisponde, restituirà un `pd.Series` , se incontra più righe corrispondenti, restituirà un `pd.DataFrame` . Questo rende questo metodo piuttosto instabile.

Questo comportamento può essere controllato dando a `.loc` un elenco di una singola voce. Questo lo costringerà a restituire un frame di dati.

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

Leggi Indicizzazione booleana dei dataframes online:
https://riptutorial.com/it/pandas/topic/9589/indicizzazione-booleana-dei-dataframes

Capitolo 12: Indicizzazione e selezione dei dati

Examples

Seleziona colonna per etichetta

```
# 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
```

Ulteriori dettagli su: http://pandas.pydata.org/pandas-docs/version/0.18.0/indexing.html#selection-by-label

Seleziona per posizione

Il `iloc` (abbreviazione di *posizione intera*) consente di selezionare le righe di un dataframe in base al loro indice di posizione. In questo modo è possibile suddividere i dataframes proprio come si fa con l'affettamento delle liste di 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 posizione della riga può essere combinata con la posizione della colonna

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

Vedi anche: Selezione per posizione

Affettare con etichette

Quando si usano le etichette, sia l'inizio che l'arresto sono inclusi nei risultati.

```
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
```

Righe da `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
```

Nota come `loc` distingue da `iloc` perché `iloc` esclude l'indice finale

```
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
```

Colonne da `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
```

Posizione mista e selezione basata sull'etichetta

dataframe:

```
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
```

Seleziona righe per posizione e colonne per etichetta:

```
df.ix[1:3, 'C':'E']
Out[19]:
     C   D   E
R1   5  58   0
R2  36  51  27
```

Se l'indice è intero, `.ix` utilizzerà le etichette anziché le posizioni:

```
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: []
```

Indicizzazione booleana

Si possono selezionare righe e colonne di un dataframe usando array booleani.

```
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
```

Altro nella documentazione di panda .

Filtraggio delle colonne (selezionando "interessante", non necessario, utilizzando RegEx, ecc.)

genera campione 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
```

mostra colonne contenenti la lettera '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
```

Mostra colonne utilizzando il 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
```

mostra tutte le colonne tranne quelle che iniziano con `a` (in altre parole rimuovi / rimuovi tutte le colonne che soddisfano la RegEx data)

```
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
```

Filtrare / selezionare le righe usando il metodo `.query ()`

```
import pandas as pd
```

generare DF casuale

```
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
```

selezionare le righe in cui i valori nella colonna `A > 2` e i valori nella colonna `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
```

usando il metodo `.query()` con le variabili per il filtraggio

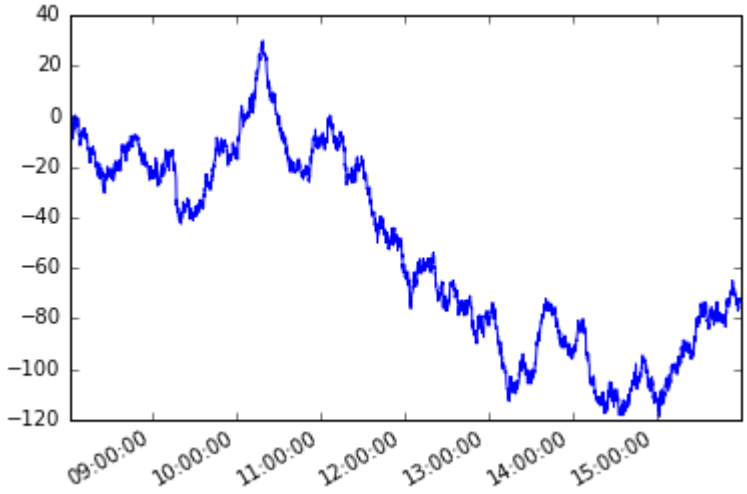
```
In [23]: B_filter = [1,7]

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
```

Affettatura dipendente dal percorso

Potrebbe essere necessario attraversare gli elementi di una serie o le righe di un dataframe in modo che l'elemento successivo o la riga successiva dipendano dall'elemento o dalla riga precedentemente selezionati. Questo è chiamato path dependency.

Si consideri il seguente serie temporale `s` con frequenza irregolare.

```
#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();
```



Assumiamo una condizione dipendente dal percorso. A partire dal primo membro della serie, voglio prendere ogni elemento successivo in modo tale che la differenza assoluta tra quell'elemento e l'elemento corrente sia maggiore o uguale a `x` .

Risolveremo questo problema usando i generatori Python.

Funzione del generatore

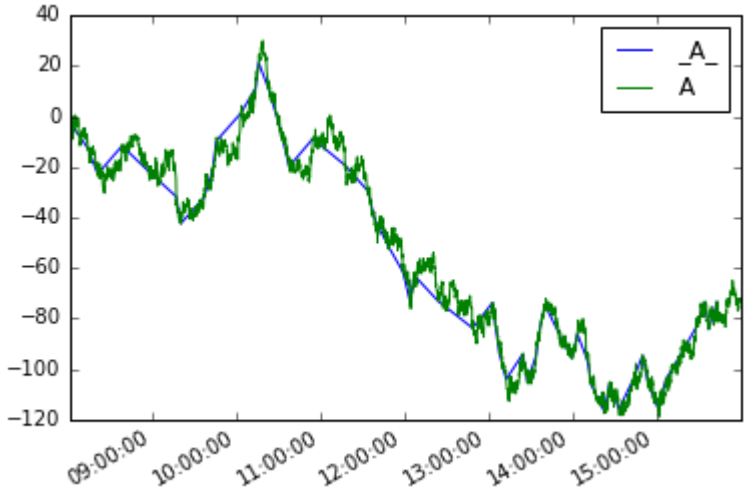
```
def mover(s, move_size=10):
    """Given a reference, find next value with
    an absolute difference >= move_size"""
    ref = None
    for i, v in s.iteritems():
        if ref is None or (abs(ref - v) >= move_size):
            yield i, v
            ref = v
```

Quindi possiamo definire una nuova serie si `moves` modo

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

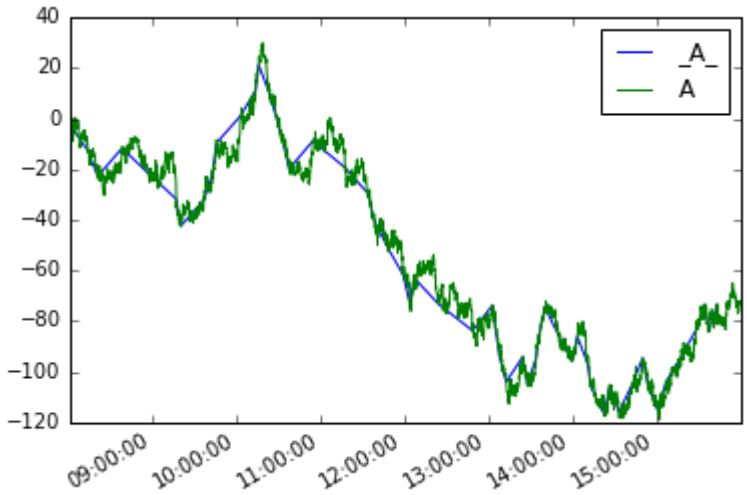
Tracciandoli entrambi

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



L'analogo per i dataframes sarebbe:

```
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)
```



Ottieni la prima / ultima n file di un dataframe

Per visualizzare il primo o l'ultimo record di un dataframe, puoi utilizzare i metodi `head` e `tail`

Per restituire le prime n righe utilizzare `DataFrame.head([n])`

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

Per restituire le ultime n righe utilizzare `DataFrame.tail([n])`

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

Senza l'argomento n, queste funzioni restituiscono 5 righe.

Si noti che la notazione di sezione per la `head` / `tail` sarebbe:

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

Seleziona righe distinte su dataframe

Permettere

```
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
```

Per ottenere i valori distinti in `col_1` puoi utilizzare `Series.unique()`

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

Ma Series.unique () funziona solo per una singola colonna.

Per simulare la *selezione unica di col_1, col_2* di SQL puoi usare `DataFrame.drop_duplicates()` :

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

Questo ti porterà tutte le righe univoche nel dataframe. Quindi se

```
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
```

Per specificare le colonne da considerare quando si selezionano i record univoci, passarli come argomenti

```
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
```

Fonte: come "selezionare distinti" tra più colonne di frame di dati nei panda? .

Filtra le righe con dati mancanti (NaN, None, NaT)

Se hai un dataframe con dati mancanti (`NaN` , `pd.NaT` , `None`) puoi filtrare le righe incomplete

```
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 tutte le righe contenenti almeno un campo con dati mancanti

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

Per eliminare semplicemente le righe che mancano i dati nelle colonne specificate utilizzare il `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
```

Utilizzare l'opzione `inplace = True` per la sostituzione sul posto con il frame filtrato.

Leggi Indicizzazione e selezione dei dati online:
https://riptutorial.com/it/pandas/topic/1751/indicizzazione-e-selezione-dei-dati

Capitolo 13: IO per Google BigQuery

Examples

Lettura dei dati da BigQuery con le credenziali dell'account utente

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

Per eseguire una query in BigQuery devi avere il tuo progetto BigQuery. Possiamo richiedere alcuni dati di esempio pubblici:

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

Questo verrà stampato:

```
Your browser has been opened to visit:

    https://accounts.google.com/o/oauth2/v2/auth...[looong 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
```

Se stai operando da un computer locale, verrà visualizzato il browser. Dopo aver concesso i privilegi, i panda continueranno con l'output:

```
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.
```

Risultato:

```
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
```

```
4               .BLP  2664340           1659
```

Come effetto collaterale, i panda creeranno il file json `bigquery_credentials.dat` che ti permetterà di eseguire ulteriori query senza bisogno di concedere più privilegi:

```
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
```

Lettura dei dati da BigQuery con le credenziali dell'account di servizio

Se hai creato un account di servizio e disponi di un file json per la chiave privata, puoi utilizzare questo file per autenticarti con i panda

```
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
```

Leggi IO per Google BigQuery online: https://riptutorial.com/it/pandas/topic/5610/io-per-google-bigquery

Capitolo 14: JSON

Examples

Leggi JSON

può passare una stringa di JSON, o un percorso di file in un file con JSON valido

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

In alternativa per risparmiare memoria:

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

Dataframe in JSON annidato come nei file flare.js utilizzati in 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")
```

Leggi JSON dal file

Contenuto di file.json (un oggetto JSON per riga):

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

Come leggere direttamente da un file locale:

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

Leggi JSON online: https://riptutorial.com/it/pandas/topic/4752/json

Capitolo 15: Lavorare con Time Series

Examples

Creazione di serie temporali

Ecco come creare una semplice serie storica.

```
import pandas as pd
import numpy as np

# The number of sample to generate
nb_sample = 100

# Seeding to obtain a reproductible 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
```

Indicizzazione parziale delle stringhe

Un modo molto pratico per impostare sottoinsiemi di serie temporali consiste nell'utilizzare l' **indicizzazione parziale delle stringhe** . Permette di selezionare un intervallo di date con una sintassi chiara.

Ottenere dati

Stiamo utilizzando il set di dati nell'esempio Creazione di serie temporali

Visualizzazione di testa e coda per vedere i confini

```
se.head(2).append(se.tail(2))

# 2016-09-24    44
# 2016-09-25    47
# 2016-12-31    85
# 2017-01-01    48
```

subsetting

Ora possiamo sottogruppi per anno, mese, giorno in modo molto intuitivo.

Per anno

```
se['2017']

# 2017-01-01    48
```

Per mese

```
se['2017-01']

# 2017-01-01    48
```

Di giorno

```
se['2017-01-01']

# 48
```

Con un intervallo di anno, mese, giorno in base alle proprie esigenze.

```
se['2016-12-31':'2017-01-01']

# 2016-12-31    85
# 2017-01-01    48
```

panda fornisce anche una funzione `truncate` dedicata per questo utilizzo attraverso i parametri `after` e `before` - ma penso che sia meno chiaro.

```
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
```

Leggi Lavorare con Time Series online: https://riptutorial.com/it/pandas/topic/7029/lavorare-con-time-series

Capitolo 16: Leggi MySQL su DataFrame

Examples

Utilizzando sqlalchemy e 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)
```

Per leggere mysql in dataframe, in caso di grandi quantità di dati

Per recuperare dati di grandi dimensioni possiamo usare i generatori nei panda e caricare i dati in blocchi.

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


# sqlalchemy engine
engine = create_engine(URL(
    drivername="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
```

Leggi Leggi MySQL su DataFrame online: https://riptutorial.com/it/pandas/topic/8809/leggi-mysql-su-dataframe

Capitolo 17: Leggi SQL Server su Dataframe

Examples

Utilizzando pyodbc

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

Specificare i parametri

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

Crea la connessione

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

Interrogazione su dataframe panda

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

Utilizzando pyodbc con loop di connessione

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

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 = pdsql.read_sql(query, con,
                                                           params=params)
                else: df = pdsql.read_sql(query, con)
                cur.close()
```

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

Leggi Leggi SQL Server su Dataframe online: https://riptutorial.com/it/pandas/topic/2176/leggi-sql-server-su-dataframe

Capitolo 18: Lettura dei file in DataFrame panda

Examples

Leggi la tabella in DataFrame

File di tabella con intestazione, piè di pagina, nomi di riga e colonna indice:

file: 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
```

codice:

```
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)
```

produzione:

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

File tabella senza nomi di riga o indice:

file: table.txt

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

codice:

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

produzione:

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

Tutte le opzioni sono disponibili nella documentazione di panda qui

Leggi il file CSV

Dati con intestazione, separati da punti e virgola anziché virgole

file: table.csv

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

codice:

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

uscita :

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

Tabella senza nomi di riga o indice e virgole come separatori

file: table.csv

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

codice:

```
import pandas as pd
```

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

produzione:

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

ulteriori chiarimenti possono essere trovati nella pagina di documentazione `read_csv`

Raccogli i dati del foglio di calcolo google in dataframe panda

A volte abbiamo bisogno di raccogliere dati da fogli di lavoro di google. Possiamo utilizzare le librerie **gspread** e **oauth2client** per raccogliere dati da fogli di lavoro di google. Ecco un esempio per raccogliere i dati:

Codice:

```
from __future__ import print_function
import gspread
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 = gspread.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())
```

Leggi Lettura dei file in DataFrame panda online:
https://riptutorial.com/it/pandas/topic/1988/lettura-dei-file-in-dataframe-panda

Capitolo 19: Manipolazione delle stringhe

Examples

Espressioni regolari

```
# 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')
```

Per informazioni su come abbinare le stringhe usando regex, vedi Guida introduttiva alle espressioni regolari .

Affettare le corde

Le stringhe in una serie possono essere tagliate usando il metodo `.str.slice()` o, più convenientemente, usando parentesi (`.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
```

Ottieni il primo carattere di ogni stringa:

```
In [3]: ser.str[0]
Out[3]:
0    L
1    d
2    c
dtype: object
```

Ottieni i primi tre caratteri di ciascuna stringa:

```
In [4]: ser.str[:3]
Out[4]:
0    Lor
1    dol
2    con
dtype: object
```

Ottieni l'ultimo carattere di ogni stringa:

```
In [5]: ser.str[-1]
```

```
Out[5]:
0    m
1    t
2    t
dtype: object
```

Ottieni gli ultimi tre caratteri di ciascuna stringa:

```
In [6]: ser.str[-3:]
Out[6]:
0    sum
1    met
2    lit
dtype: object
```

Ottieni tutti gli altri personaggi dei primi 10 personaggi:

```
In [7]: ser.str[:10:2]
Out[7]:
0    Lrmis
1    dlrst
2    cnett
dtype: object
```

I panda si comportano in modo simile a Python quando gestiscono fette e indici. Ad esempio, se un indice non rientra nell'intervallo, Python genera un errore:

```
In [8]:'Lorem ipsum'[12]
# IndexError: string index out of range
```

Tuttavia, se una porzione non rientra nell'intervallo, viene restituita una stringa vuota:

```
In [9]: 'Lorem ipsum'[12:15]
Out[9]: ''
```

Panda restituisce NaN quando un indice non è compreso nell'intervallo:

```
In [10]: ser.str[12]
Out[10]:
0    NaN
1      e
2      a
dtype: object
```

E restituisce una stringa vuota se una sezione non è compresa nell'intervallo:

```
In [11]: ser.str[12:15]
Out[11]:
0
1     et
2    adi
dtype: object
```

Verifica del contenuto di una stringa

`str.contains()` metodo `str.contains()` può essere usato per verificare se un pattern si verifica in ogni stringa di una serie. `str.startswith()` e `str.endswith()` possono anche essere usati come versioni più specializzate.

```
In [1]: animals = pd.Series(['cat', 'dog', 'bear', 'cow', 'bird', 'owl', 'rabbit', 'snake'])
```

Controlla se le stringhe contengono la lettera '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
```

Questo può essere usato come indice booleano per restituire solo gli animali contenenti la lettera 'a':

```
In [3]: animals[animals.str.contains('a')]
Out[3]:
0       cat
2      bear
6    rabbit
7     snake
dtype: object
```

`str.startswith` metodi `str.startswith` e `str.endswith` funzionano in modo simile, ma accettano anche tuple come input.

```
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
```

Capitalizzazione di stringhe

```
In [1]: ser = pd.Series(['lORem ipSuM', 'Dolor sit amet', 'Consectetur Adipiscing Elit'])
```

Converti tutto in maiuscolo:

```
In [2]: ser.str.upper()
Out[2]:
0                    LOREM IPSUM
1                 DOLOR SIT AMET
2    CONSECTETUR ADIPISCING ELIT
dtype: object
```

Tutto in minuscolo:

```
In [3]: ser.str.lower()
Out[3]:
0                    lorem ipsum
1                 dolor sit amet
2    consectetur adipiscing elit
dtype: object
```

Inserisci in maiuscolo il primo carattere e in minuscolo il rimanente:

```
In [4]: ser.str.capitalize()
Out[4]:
0                    Lorem ipsum
1                 Dolor sit amet
2    Consectetur adipiscing elit
dtype: object
```

Converti ogni stringa in un titlecase (capitalizza il primo carattere di ogni parola in ogni stringa, in minuscolo il rimanente):

```
In [5]: ser.str.title()
Out[5]:
0                    Lorem Ipsum
1                 Dolor Sit Amet
2    Consectetur Adipiscing Elit
dtype: object
```

Casi di scambio (conversione da lettere minuscole a maiuscole e viceversa):

```
In [6]: ser.str.swapcase()
Out[6]:
0                    LorEM IPsUm
1                 dOLOR SIT AMET
2    cONSECTETUR aDIPISCING eLIT
dtype: object
```

Oltre a questi metodi che modificano le lettere maiuscole, è possibile utilizzare diversi metodi per verificare la capitalizzazione delle stringhe.

```
In [7]: ser = pd.Series(['LOREM IPSUM', 'dolor sit amet', 'Consectetur Adipiscing Elit'])
```

Controlla se è tutto in minuscolo:

```
In [8]: ser.str.islower()
Out[8]:
```

```
0    False
1     True
2    False
dtype: bool
```

È tutto maiuscolo:

```
In [9]: ser.str.isupper()
Out[9]:
0     True
1    False
2    False
dtype: bool
```

È una stringa in titlecas:

```
In [10]: ser.str.istitle()
Out[10]:
0    False
1    False
2     True
dtype: bool
```

Leggi Manipolazione delle stringhe online:
https://riptutorial.com/it/pandas/topic/2372/manipolazione-delle-stringhe

Capitolo 20: Meta: linee guida per la documentazione

Osservazioni

Questo meta post è simile alla versione python http://stackoverflow.com/documentation/python/394/meta-documentation-guidelines#t=20160724005840635952 1 .

Si prega di fare suggerimenti di modifica e commentare quelli (al posto di commenti appropriati), in modo che possiamo incarnare / iterare su questi suggerimenti :)

Examples

Visualizzazione di frammenti di codice e output

Due opzioni popolari sono:

notazione ipython:

```
In [11]: df = pd.DataFrame([[1, 2], [3, 4]])

In [12]: df
Out[12]:
   0  1
0  1  2
1  3  4
```

In alternativa (questo è molto popolare nella documentazione di Python) e in modo più conciso:

```
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
```

Generalmente, questo è meglio per gli esempi più piccoli.

Nota: la distinzione tra stampa e output. ipython lo rende chiaro (le stampe si verificano prima che l'output venga restituito):

```
In [21]: [print(col) for col in df]
0
1
Out[21]: [None, None]
```

stile

Utilizza la libreria pandas come `pd` , questo può essere assunto (l'importazione non deve essere in ogni esempio)

```
import pandas as pd
```

PEP8!

- 4 rientranze spaziali
- kwargs non dovrebbe usare spazi `f(a=1)`
- Limite di 80 caratteri (l'intera linea che si adatta allo snippet di codice reso dovrebbe essere fortemente preferita)

Supporto per la versione di Pandas

La maggior parte degli esempi funzionerà su più versioni, se si utilizza una funzione "nuova" da menzionare quando è stata introdotta.

Esempio: `sort_values` .

stampare dichiarazioni

La maggior parte delle volte la stampa dovrebbe essere evitata in quanto può essere una distrazione (Out dovrebbe essere preferito).
Questo è:

```
a
# Out: 1
```

è sempre meglio di

```
print(a)
# prints: 1
```

Preferisco supportare python 2 e 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)
```

Leggi Meta: linee guida per la documentazione online:
https://riptutorial.com/it/pandas/topic/3253/meta--linee-guida-per-la-documentazione

Capitolo 21: MultiIndex

Examples

Seleziona da MultiIndex per livello

Dato il seguente 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]:
                          C
A         B
 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
```

Ottieni i valori di `A` , per nome:

```
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 per numero di livello:

```
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')
```

E per un intervallo specifico:

```
In [16]: df.loc[(df.index.get_level_values('A') > 0.5) & (df.index.get_level_values('A') <
2.1)]
Out[16]:
                          C
A        B
0.902764 -0.259656 -1.864541
1.696989 -1.221131 -2.975839
```

L'intervallo può includere anche più colonne:

```
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
```

Per estrarre un valore specifico puoi usare xs (sezione trasversale):

```
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
```

Iterare su DataFrame con MultiIndex

Dato il seguente 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
```

È possibile eseguire l'iterazione da qualsiasi livello di MultiIndex. Ad esempio, `level=0` (puoi anche selezionare il livello per nome es. `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
```

Puoi anche selezionare i livelli per nome es. Livello = '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
```

Impostazione e ordinamento di un MultiIndex

Questo esempio mostra come usare i dati della colonna per impostare un `MultiIndex` in 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
```

Puoi ordinare l'indice subito dopo averlo impostato:

```
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
```

Avere un indice ordinato, si tradurrà in ricerche leggermente più efficienti al primo livello:

```
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
```

Dopo aver impostato l'indice, è possibile eseguire ricerche per record o gruppi di record specifici:

```
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
```

Come modificare le colonne MultiIndex in colonne standard

Dato un DataFrame con colonne 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
```

Se si desidera modificare le colonne in colonne standard (non MultiIndex), basta rinominare le colonne.

```
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
```

Come cambiare le colonne standard in MultiIndex

Inizia con un DataFrame standard

```
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
```

Ora per passare a MultiIndex, creare un oggetto `MultiIndex` e assegnarlo 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
```

Colonne MultiIndex

MultiIndex può anche essere utilizzato per creare DataFrames con colonne multilivello. Basta usare la parola chiave `columns` nel 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
```

Visualizzazione di tutti gli elementi nell'indice

Per visualizzare tutti gli elementi nell'indice, modificare le opzioni di stampa che "sparsifica" la visualizzazione 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
```

Leggi MultiIndex online: https://riptutorial.com/it/pandas/topic/3840/multiindex

Capitolo 22: Ottenere informazioni su DataFrames

Examples

Ottieni informazioni su DataFrame e utilizzo della memoria

Per ottenere informazioni di base su un DataFrame inclusi i nomi delle colonne e i tipi di dati:

```
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
```

Per ottenere l'utilizzo della memoria di 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
```

Elenca i nomi delle colonne DataFrame

```
df = pd.DataFrame({'a': [1, 2, 3], 'b': [4, 5, 6], 'c': [7, 8, 9]})
```

Per elencare i nomi delle colonne in un DataFrame:

```
>>> list(df)
['a', 'b', 'c']
```

Questo metodo di comprensione degli elenchi è particolarmente utile quando si utilizza il debugger:

```
>>> [c for c in df]
['a', 'b', 'c']
```

Questa è la lunga strada:

```
sampledf.columns.tolist()
```

Puoi anche stamparli come un indice invece di un elenco (questo non sarà molto visibile per i dataframes con molte colonne):

```
df.columns
```

Le varie statistiche di riepilogo di Dataframe.

```
import pandas as pd
df = pd.DataFrame(np.random.randn(5, 5), columns=list('ABCDE'))
```

Per generare varie statistiche riassuntive. Per valori numerici il numero di valori non NA / null (`count`), la media (`mean`), la deviazione standard `std` e valori noti come riepilogo a cinque numeri :

- `min` : minimo (osservazione minima)
- `25%` : quartile inferiore o primo quartile (Q1)
- `50%` : mediana (valore medio, Q2)
- `75%` : quartile superiore o terzo quartile (Q3)
- `max` : massimo (massima osservazione)

```
>>> 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
```

Leggi Ottenere informazioni su DataFrames online:
https://riptutorial.com/it/pandas/topic/6697/ottenere-informazioni-su-dataframes

Capitolo 23: Pandas Datareader

Osservazioni

Il daseadeader di Pandas è un pacchetto secondario che consente di creare un dataframe da varie origini dati Internet, attualmente incluso:

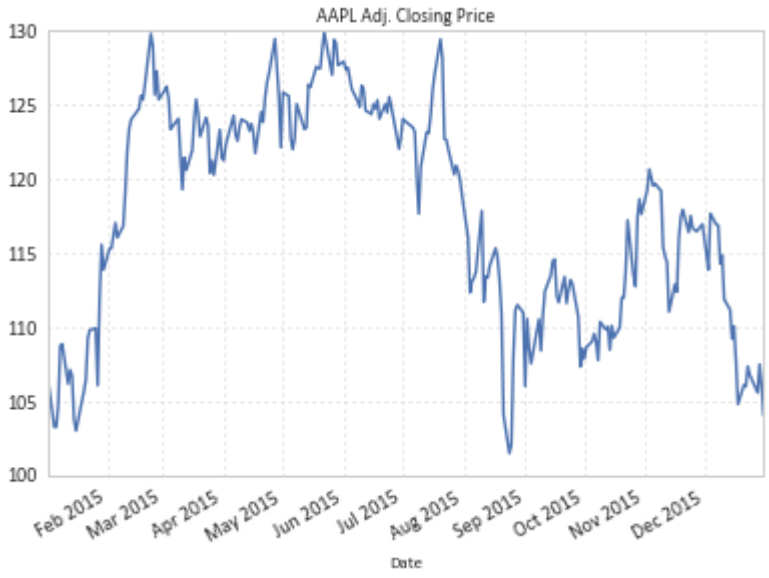
- Yahoo! Finanza
- Google Finanza
- St.Louis FED (FRED)
- La biblioteca di dati di Kenneth French
- Banca Mondiale
- statistiche di Google

Per ulteriori informazioni, vedere qui .

Examples

Esempio di base di 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')
```

Lettura di dati finanziari (per più ticker) nel pannello pandas - demo

```
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` - è un pannello panda, con il quale possiamo fare cose divertenti:

vediamo cosa abbiamo nel nostro pannello

```
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')
```

selezionando e affettando i dati

```
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
```

Leggi Pandas Datareader online: https://riptutorial.com/it/pandas/topic/1912/pandas-datareader

Capitolo 24: pd.DataFrame.apply

Examples

pandas.DataFrame.apply Uso di base

Il metodo pandas.DataFrame.apply () viene utilizzato per applicare una data funzione a un intero `DataFrame` --- ad esempio, calcolando la radice quadrata di ogni voce di un dato `DataFrame` o sommando tutte le righe di un `DataFrame` per restituire una `Series` .

Il seguente è un esempio di base di utilizzo di questa funzione:

```
# 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
```

Leggi pd.DataFrame.apply online: https://riptutorial.com/it/pandas/topic/7024/pd-dataframe-apply

Capitolo 25: Raggruppamento di dati

Examples

Raggruppamento di base

Raggruppa per una colonna

Utilizzando il seguente 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
```

Raggruppa per colonna A e ottieni il valore medio di altre colonne:

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

Raggruppa per colonne multiple

```
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
```

Nota come dopo aver raggruppato ogni riga nel DataFrame risultante viene indicizzato da una tupla o MultiIndex (in questo caso una coppia di elementi dalle colonne A e B).

Per applicare contemporaneamente più metodi di aggregazione, ad esempio per contare il numero di elementi in ciascun gruppo e calcolare la media, utilizzare la funzione `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
```

Raggruppamento di numeri

Per il seguente 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
```

Gruppo `Age` in tre categorie (o bin). I bidoni possono essere dati come

- un intero `n` indica il numero di bin: in questo caso i dati del dataframe sono divisi in `n` intervalli di uguale dimensione
- una sequenza di numeri interi che denotano il punto finale degli intervalli aperti a sinistra in cui i dati sono divisi in: `bins=[19, 40, 65, np.inf]` per esempio `bins=[19, 40, 65, np.inf]` crea tre gruppi di età `(19, 40]` , `(40, 65]` e `(65, np.inf]` .

Pandas assegna automaticamente le versioni stringa degli intervalli come etichetta. È anche possibile definire le proprie etichette definendo un parametro `labels` come un elenco di stringhe.

```
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]]
```

Usalo in `groupby` per ottenere il numero medio di 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
```

Croce tabulare gruppi di età e sesso:

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

Selezione della colonna di un gruppo

Quando fai un groupby puoi selezionare una singola colonna o un elenco di colonne:

```
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
```

Puoi anche utilizzare `agg` per specificare colonne e aggregazione da eseguire:

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

Aggregazione per dimensione in base al conteggio

La differenza tra `size` e `count` è:

`size` conta i valori `NaN` , il `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
```

Gruppi aggreganti

```
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
```

Per più colonne:

```
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
```

Esporta gruppi in diversi file

È possibile eseguire l'iterazione sull'oggetto restituito da `groupby()` . L'iteratore contiene tuple `(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', factor'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))
```

utilizzando la trasformazione per ottenere statistiche a livello di gruppo preservando il dataframe originale

esempio:

```
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
```

Voglio ottenere il conteggio delle osservazioni non mancanti di B per ciascuna combinazione di `group1` e `group2` . `groupby.transform` è una funzione molto potente che fa esattamente questo.

```
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
```

Leggi Raggruppamento di dati online: https://riptutorial.com/it/pandas/topic/1822/raggruppamento-di-dati

Capitolo 26: Raggruppamento di dati di serie temporali

Examples

Genera serie temporali di numeri casuali e poi giù campione

```
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
```

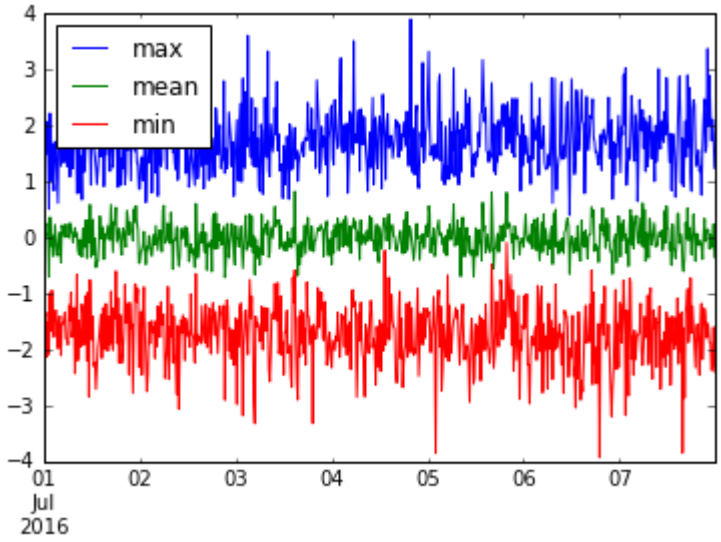
Prendiamo questi 7 giorni di dati al minuto e scendiamo il campione ogni 15 minuti. Tutti i codici di frequenza possono essere trovati qui .

```
# 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 qualsiasi altra cosa che possiamo fare a un oggetto `groupby` , alla documentazione .

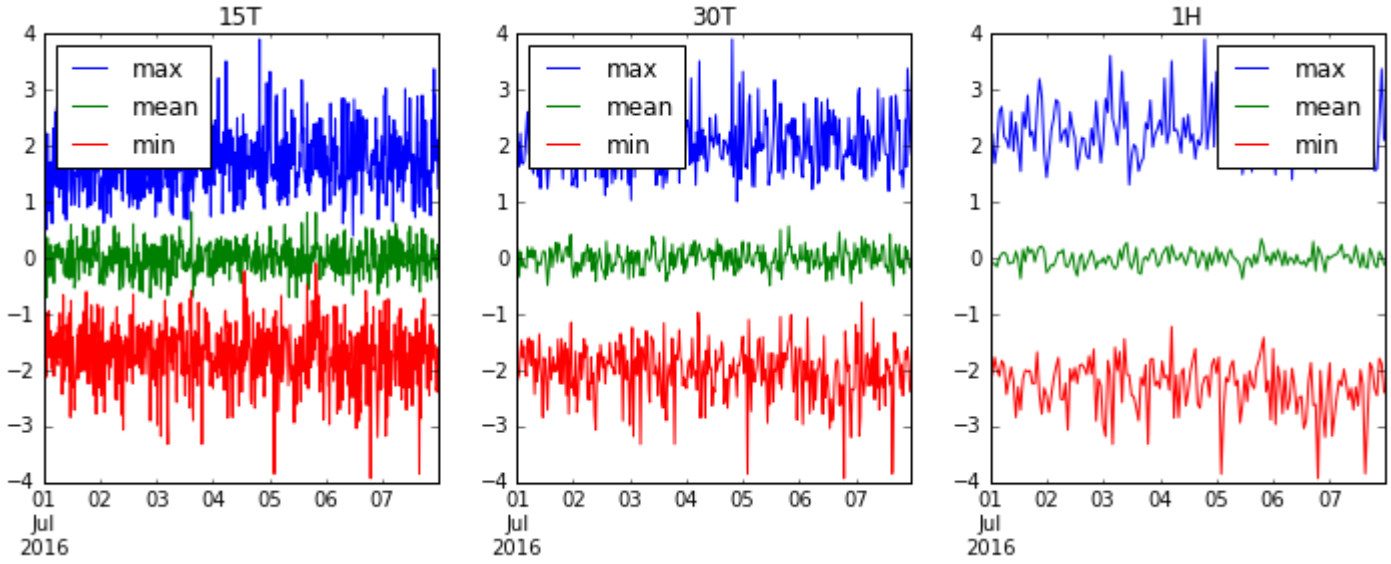
Possiamo anche aggregare diverse cose utili. Tracciamo il `min` , la `mean` e il `max` di questi dati `resample('15M')` .

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



Ricapitoliamo su `'15T'` (15 minuti), `'30T'` (mezz'ora) e `'1H'` (1 ora) e vediamo come i nostri dati diventano più fluidi.

```
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)
```



Leggi Raggruppamento di dati di serie temporali online:
https://riptutorial.com/it/pandas/topic/4747/raggruppamento-di-dati-di-serie-temporali

Capitolo 27: Rendere i panda divertenti con i tipi di dati nativi di Python

Examples

Spostamento dei dati da panda nelle strutture native di Python e Numpy Data

```
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
```

Ottenere una lista python da una serie:

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

DataFrames non ha un metodo `tolist()` . Provando, si ottiene un errore 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.__getattribute__(self, name)
   2745
   2746     def __setattr__(self, name, value):

AttributeError: 'DataFrame' object has no attribute 'tolist'
```

Ottenere una matrice numpy da una serie:

```
In [5]: df['B'].values
Out[5]: array([ 1.,  2.,  3.])
```

Puoi anche ottenere un array di colonne come array numpy individuali da un intero dataframe:

```
In [6]: df.values
Out[6]:
array([[1, 1.0, 'a', True],
       [2, 2.0, 'b', False],
```

```
       [3, 3.0, 'c', True]], dtype=object)
```

Ottenere un dizionario da una serie (usa l'indice come chiave):

```
In [7]: df['C'].to_dict()
Out[7]: {0: 'a', 1: 'b', 2: 'c'}
```

Puoi anche recuperare l'intero DataFrame come dizionario:

```
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}}
```

Il metodo `to_dict` ha alcuni parametri diversi per regolare la formattazione dei dizionari. Per ottenere un elenco di dict per ogni riga:

```
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}]
```

Vedere la documentazione per l'elenco completo delle opzioni disponibili per creare dizionari.

Leggi Rendere i panda divertenti con i tipi di dati nativi di Python online: https://riptutorial.com/it/pandas/topic/8008/rendere-i-panda-divertenti-con-i-tipi-di-dati-nativi-di-python

Capitolo 28: ricampionamento

Examples

Downsampling e upsampling

```
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
```

Leggi ricampionamento online: https://riptutorial.com/it/pandas/topic/2164/ricampionamento

Capitolo 29: Rimodellamento e rotazione

Examples

Semplice rotazione

Prima prova usa il `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
```

Se è necessario reimpostare l'indice, rimuovere i nomi delle colonne e riempire i valori 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
```

Facendo perno con l'aggregazione

```
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
```

Se usi `pivot` , ottieni l'errore:

```
print (df.pivot(index='Position', columns='City', values='Age'))
```

ValueError: l'indice contiene voci duplicate, non può essere rimodellato

Utilizza `pivot_table` con funzione di aggregazione:

```
#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
```

Un altro agg funzioni:

```
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
```

Se necessario aggregare per colonne con valori `string` :

```
print (df.pivot_table(index='Position', columns='City', values='Name'))
```

DataError: nessun tipo numerico da aggregare

Puoi usare queste funzioni aggragating:

```
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           -
```

Le informazioni riguardanti il *sesso non* sono ancora state usate. Potrebbe essere cambiato da una delle colonne o potrebbe essere aggiunto come un altro livello:

```
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
```

È possibile specificare più colonne in qualsiasi indice, colonna e valore degli attributi.

```
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
```

Applicare diverse funzioni di aggregazione

Puoi facilmente applicare più funzioni durante un singolo pivot:

```
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 volte, potresti voler applicare funzioni specifiche a colonne specifiche:

```
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
```

Si può passare un elenco di funzioni da applicare anche alle singole colonne:

```
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
```

Impilabile e disimpilabile

```
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
               B    0.400157
       two     A    0.978738
               B    2.240893
foo    one     A    1.867558
               B   -0.977278
       two     A    0.950088
               B   -0.151357
qux    one     A   -0.103219
               B    0.410599
       two     A    0.144044
               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` (nuovo in `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
```

panda muggito 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
```

Tabulazione incrociata

```
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'])

Hearth Disease  N  Y
Sex
F               2  3
M               3  2
```

Usando la notazione a punti:

```
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
```

Ottenere la trasposizione di 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
```

Ottenere margini o cumulativi:

```
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
```

Ottenere trasposizione cumulativa:

```
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
```

Ottenere percentuali:

```
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
```

Ottenendo cumulativo e moltiplicando per 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
```

Rimozione di una colonna da DF (solo andata):

```
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
```

I panda si sciolgono per andare da larghi a lunghi

```
>>> 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
```

Dividi (risagoma) le stringhe CSV in colonne in più righe, con un elemento per riga

```
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)
```

Produzione:

```
        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
```

Leggi Rimodellamento e rotazione online:
https://riptutorial.com/it/pandas/topic/1463/rimodellamento-e-rotazione

Capitolo 30: Salva pandas dataframe in un file csv

Parametri

Parametro	Descrizione
path_or_buf	stringa o file handle, default None Percorso file o oggetto, se viene fornito Nessuno, il risultato viene restituito come stringa.
settembre	carattere, default ',' Delimitatore di campo per il file di output.
na_rep	string, default '' Rappresentazione di dati mancanti
float_format	string, default Nessuno Stringa di formato per numeri in virgola mobile
colonne	sequenza, Colonne facoltative da scrivere
intestazione	booleano o elenco di stringhe, i nomi di colonna True Write out predefiniti. Se viene fornito un elenco di stringhe, si presume che siano alias per i nomi delle colonne
indice	booleano, nomi di riga True Write di default (indice)
index_label	stringa o sequenza o False, impostazione predefinita Nessuna etichetta di colonna per le colonne indice, se necessario. Se viene dato None e l'intestazione e l'indice sono True, vengono utilizzati i nomi dell'indice. Una sequenza dovrebbe essere data se il DataFrame utilizza MultiIndex. Se False non stampa i campi per i nomi di indice. Usa index_label = False per facilitare l'importazione in R
nanRep	Nessuno deprecato, utilizzare na_rep
modalità	str Modalità di scrittura Python, default 'w'
codifica	string, facoltativo Una stringa che rappresenta la codifica da utilizzare nel file di output, imposta come default 'ascii' su Python 2 e 'utf-8' su Python 3.
compressione	string, facoltativo una stringa che rappresenta la compressione da utilizzare nel file di output, i valori consentiti sono 'gzip', 'bz2', 'xz', usati solo quando il primo argomento è un nome file
line_terminator	string, default 'n' Il carattere di nuova riga o sequenza di caratteri da utilizzare nel file di output
citando	costante facoltativa dal modulo csv predefinito a csv.QUOTE_MINIMAL

Parametro	Descrizione
quotechar	stringa (lunghezza 1), carattere "" "predefinito usato per quotare i campi
doublequote	booleano, quotazione True Control predefinita di quotechar all'interno di un campo
EscapeChar	stringa (lunghezza 1), default Nessun carattere usato per uscire sep e quotechar quando appropriato
chunksize	int o Nessuna riga da scrivere alla volta
tupleize_cols	boolean, default False scrive colonne multi_index come un elenco di tuple (se True) o nuovo (formato espanso) se False)
formato data	string, default Nessuno Stringa di formato per oggetti datetime
decimale	string, default '.' Carattere riconosciuto come separatore decimale. Ad esempio, usa "," per i dati europei

Examples

Crea un DataFrame casuale e scrivi in .csv

Crea un semplice DataFrame.

```
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
```

Adesso scrivi in un file CSV:

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

Contenuto di 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
```

Si noti che specifichiamo `index=False` modo che gli indici generati automaticamente (riga #s 0,1,2,3,4) non siano inclusi nel file CSV. Includilo se hai bisogno della colonna indice, in questo modo:

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

Contenuto di 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
```

Si noti inoltre che è possibile rimuovere l'intestazione se non è necessaria con `header=False` . Questo è l'output più semplice:

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

Contenuto di 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
```

Il delimitatore può essere impostato da `sep=` argomento, sebbene il separatore standard per i file csv sia `','` .

```
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
```

Salva Pandas DataFrame da elenco a dicts a csv senza indice e con codifica dei dati

```
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')
```

Leggi Salva pandas dataframe in un file csv online:
https://riptutorial.com/it/pandas/topic/1558/salva-pandas-dataframe-in-un-file-csv

Capitolo 31: Semplice manipolazione di DataFrames

Examples

Elimina una colonna in un DataFrame

Ci sono un paio di modi per eliminare una colonna in 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) Utilizzando `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 la `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) Utilizzo di `drop` con i numeri delle colonne

Per utilizzare i numeri interi delle colonne anziché i nomi (ricordare gli indici delle colonne iniziano da zero):

```
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
```

Rinominare una colonna

```
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
```

Per rinominare una o più colonne, passa i vecchi nomi e i nuovi nomi come dizionario:

```
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 funzione:

```
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
```

Puoi anche impostare `df.columns` come elenco dei nuovi nomi:

```
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
```

Maggiori dettagli possono essere trovati qui .

Aggiungere una nuova colonna

```
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
```

Assegna direttamente

```
df['C'] = [7, 8, 9]

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

Aggiungi una colonna costante

```
df['C'] = 1

print(df)

# Output:
#    A  B  C
# 0  1  4  1
# 1  2  5  1
# 2  3  6  1
```

Colonna come espressione in altre colonne

```
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
```

Le operazioni sono calcolate in base al componente, quindi se avessimo colonne come elenchi

```
a = [1, 2, 3]
b = [4, 5, 6]
```

la colonna nell'ultima espressione si otterrebbe come

```
c = [x**y for (x,y) in zip(a,b)]

print(c)
# Output:
# [1, 32, 729]
```

Crealo al volo

```
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
```

aggiungi più colonne

```
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
```

aggiungi più colonne al volo

```
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
```

Individua e sostituisci i dati in una colonna

```
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
```

Per codificare il maschio a 0 e la femmina 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
```

Aggiunta di una nuova riga a DataFrame

Dato 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
```

Produzione:

```
  C1 C2 C3
0  1  2  3
1  a  b  c
```

Aggiungiamo una nuova riga, `[10,11,12]` :

```
df = pd.DataFrame(np.array([[10,11,12]]), \
        columns=["C1", "C2", "C3"]).append(df, ignore_index=True)
print df
```

Produzione:

```
   C1  C2  C3
```

```
0  10  11  12
1   1   2   3
2   a   b   c
```

Elimina / elimina righe da DataFrame

generiamo prima un DataFrame:

```
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
```

rilasciare righe con indici: `0` e `4` usando il metodo `drop([...], inplace=True)` :

```
df.drop([0,4], inplace=True)

print(df)
# Output
#    a  b
# 1  2  3
# 2  4  5
# 3  6  7
```

rilasciare righe con indici: `0` e `4` usando il metodo `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
```

utilizzando il metodo di selezione negativo:

```
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
# 3  6  7
```

Riordina colonne

```
# 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]
```

Leggi Semplice manipolazione di DataFrames online:
https://riptutorial.com/it/pandas/topic/6694/semplice-manipolazione-di-dataframes

Capitolo 32: Serie

Examples

Esempi di creazione di serie semplici

Una serie è una struttura dati a una dimensione. È un po 'come un array sovralimentato o un dizionario.

```
import pandas as pd

s = pd.Series([10, 20, 30])

>>> s
0    10
1    20
2    30
dtype: int64
```

Ogni valore in una serie ha un indice. Per impostazione predefinita, gli indici sono numeri interi, che vanno da 0 alla lunghezza della serie meno 1. Nell'esempio sopra puoi vedere gli indici stampati a sinistra dei valori.

Puoi specificare i tuoi indici:

```
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
```

Serie con datetime

```
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]
```

Alcuni suggerimenti rapidi su Series in Pandas

Supponiamo di avere la seguente 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
```

Le seguenti sono alcune semplici cose che tornano a portata di mano quando si lavora con la serie:

Per ottenere la lunghezza di s:

```
>>> len(s)
8
```

Per accedere a un elemento in s:

```
>>> s[4]
8
```

Per accedere a un elemento in s usando l'indice:

```
>>> s.loc[2]
6
```

Per accedere a una sottoserie all'interno di:

```
>>> s[1:3]
1    4
2    6
dtype: int64
```

Per ottenere una sottoserie di s con valori superiori a 5:

```
>>> s[s > 5]
2    6
4    8
5    7
dtype: int64
```

Per ottenere il minimo, il massimo, la media e la deviazione standard:

```
>>> s.min()
1
>>> s.max()
8
>>> s.mean()
4.75
>>> s.std()
2.2519832529192065
```

Per convertire il tipo di serie in float:

```
>>> 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
```

Per ottenere i valori in s come un array numpy:

```
>>> s.values
array([1, 4, 6, 3, 8, 7, 4, 5])
```

Per fare una copia di s:

```
>>> d = s.copy()
>>> d
0    1
1    4
2    6
3    3
4    8
5    7
6    4
7    5
dtype: int64
```

Applicazione di una funzione a una serie

Pandas fornisce un modo efficace per applicare una funzione a ogni elemento di una serie e ottenere una nuova serie. Supponiamo di avere la seguente 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
```

e una funzione quadrata:

```
>>> def square(x):
...     return x*x
```

Possiamo semplicemente applicare un quadrato a ogni elemento di s e ottenere una nuova serie:

```
>>> t = s.apply(square)
>>> t
0     9
1    49
2    25
3    64
4    81
5     1
6     0
7    16
dtype: int64
```

In alcuni casi è più facile usare un'espressione 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
```

oppure possiamo usare qualsiasi funzione incorporata:

```
>>> q = pd.Series(['Bob', 'Jack', 'Rose'])
>>> q.apply(str.lower)
```

```
0     bob
1    jack
2    rose
dtype: object
```

Se tutti gli elementi della serie sono stringhe, esiste un metodo più semplice per applicare i metodi stringa:

```
>>> q.str.lower()
0     bob
1    jack
2    rose
dtype: object
>>> q.str.len()
0    3
1    4
2    4
```

Leggi Serie online: https://riptutorial.com/it/pandas/topic/1898/serie

Capitolo 33: Sezioni trasversali di diversi assi con MultiIndex

Examples

Selezione delle sezioni trasversali usando .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` accetta un `level` (o il nome di detto livello o un intero), e un `axis` : 0 per le righe, 1 per le colonne.

`.xs` è disponibile sia per `pandas.Series` che per `pandas.DataFrame` .

Selezione su righe:

```
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
```

Selezione su colonne:

```
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` funziona solo per la selezione, l'assegnazione NON è possibile (ottenere, non impostare): ¨

```
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
```

Utilizzando .loc e affettatrici

A differenza del metodo `.xs` , questo consente di assegnare valori. L'indicizzazione mediante slicer è disponibile dalla versione `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
```

Selezione su righe :

```
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
```

Selezione su colonne:

```
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
```

Selezione su entrambi gli assi :

```
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
```

L'assegnazione funziona (a differenza di `.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
```

Leggi Sezioni trasversali di diversi assi con MultiIndex online:
https://riptutorial.com/it/pandas/topic/8099/sezioni-trasversali-di-diversi-assi-con-multiindex

Capitolo 34: Spostamento e ritardo dei dati

Examples

Spostando o ritardando i valori in un dataframe

```
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
```

Il primo argomento di `.shift()` è `periods` , il numero di spazi per spostare i dati. Se non specificato, il valore predefinito è `1` .

Leggi Spostamento e ritardo dei dati online:
https://riptutorial.com/it/pandas/topic/7554/spostamento-e-ritardo-dei-dati

Capitolo 35: Strumenti computazionali

Examples

Trova la correlazione tra le colonne

Supponiamo di avere un DataFrame di valori numerici, ad esempio:

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

Poi

```
>>> 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
```

troverà la correlazione di Pearson tra le colonne. Nota come la diagonale è 1, poiché ogni colonna è (ovviamente) pienamente correlata con se stessa.

`pd.DataFrame.correlation` accetta un parametro di `method` opzionale, specificando quale algoritmo utilizzare. L'impostazione predefinita è `pearson` . Per usare la correlazione di Spearman, ad esempio, usa

```
>>> 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
```

Leggi Strumenti computazionali online: https://riptutorial.com/it/pandas/topic/5620/strumenti-computazionali

Capitolo 36: Strumenti IO di Pandas (lettura e salvataggio di set di dati)

Osservazioni

La documentazione ufficiale di panda include una pagina su IO Tools con un elenco di funzioni rilevanti per leggere e scrivere su file, oltre ad alcuni esempi e parametri comuni.

Examples

Lettura file CSV in DataFrame

Esempio per leggere il file `data_file.csv` come:

File:

```
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
```

Codice:

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

Produzione:

```
   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
```

Alcuni argomenti utili:

- **`sep`** Il delimitatore di campo predefinito è una virgola `,` . Usa questa opzione se hai bisogno di un delimitatore diverso, ad esempio `pd.read_csv('data_file.csv', sep=';')`
- **`index_col`** Con `index_col = n` (`n` un intero) dici ai panda di utilizzare la colonna `n` per

indicizzare il DataFrame. Nell'esempio sopra:

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

Produzione:

```
          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`** Per impostazione predefinita, le righe vuote vengono saltate. Usa `skip_blank_lines=False` per includere righe vuote (saranno riempite con valori `NaN`)

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

Produzione:

```
         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`** Utilizzare questa opzione per analizzare i dati della data.

File:

```
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
```

Codice per analizzare le colonne `0` e `1` come date:

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

Produzione:

```
  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
```

Per impostazione predefinita, il formato della data viene dedotto. Se si desidera specificare

un formato di data che è possibile utilizzare per esempio

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

Produzione:

```
  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
```

Ulteriori informazioni sui parametri della funzione sono disponibili nella documentazione ufficiale .

Salvataggio di base in un file 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')
```

Date di analisi durante la lettura da CSV

È possibile specificare una colonna che contenga date, in modo che i panda le analizzino automaticamente durante la lettura dal CSV

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

Foglio di calcolo a dettare di DataFrames

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

Leggi un foglio specifico

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

Ci sono molte opzioni di analisi per `read_excel` (simile alle opzioni in `read_csv` .

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

Testare 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
```

Comprensione delle liste

Tutti i file sono in `files` cartelle. Prima di creare la lista dei DataFrames e poi `concat` loro:

```
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
```

Leggi in blocchi

```
import pandas as pd

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

Salva nel file CSV

Salva con parametri predefiniti:

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

Scrivi colonne specifiche:

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

Il delimitatore Difault è "," - per cambiarlo:

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

Scrivi senza l'intestazione:

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

Scrivi con un dato header:

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

Per usare una codifica specifica (es. 'Utf-8') usa l'argomento di codifica:

df.to_csv (file_name, encoding = 'utf-8')

Parsing date columns with read_csv

Le date hanno sempre un formato diverso, possono essere analizzate utilizzando una specifica funzione parse_dates.

Questo *input.csv* :

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

Può essere analizzato in questo modo:

```
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)
```

L' argomento *parse_dates* è la colonna da analizzare
date_parser è la funzione parser

Leggi e unisci più file CSV (con la stessa struttura) in un unico 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())
```

Se si desidera unire i file CSV orizzontalmente (aggiungendo colonne), utilizzare `axis=1` quando si chiama la funzione `pd.concat()` :

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

Leggendo il file cvs in un frame di dati panda quando non c'è una riga di intestazione

Se il file non contiene una riga di intestazione,

File:

```
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
```

puoi utilizzare i `names` parole chiave per fornire i nomi delle colonne:

```
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
```

Utilizzando HDFStore

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

genera DF di esempio con vari tipi di dtype

```
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  EEEEEEEEEE
```

fare un DF più grande (10 * 100.000 = 1.000.000 di righe)

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

creare (o aprire un file HDFStore esistente)

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

salva il nostro frame di dati nel file h5 (HDFStore), indicizzando le colonne [int32, int64, string]:

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

mostra dettagli 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}
```

mostra colonne indicizzate

```
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}
```

chiudere (flush su disco) il nostro file di archivio

```
store.close()
```

Leggi il log di accesso di Nginx (più quotechar)

Per più quotechar utilizzare regex al posto di sep:

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

Leggi Strumenti IO di Pandas (lettura e salvataggio di set di dati) online: https://riptutorial.com/it/pandas/topic/2896/strumenti-io-di-pandas--lettura-e-salvataggio-di-set-di-dati-

Capitolo 37: Tipi di dati

Osservazioni

i dtypes non sono nativi dei panda. Sono il risultato di un accoppiamento architettonico dei panda vicino a Numpy.

il dtype di una colonna non deve in alcun modo correlare al tipo python dell'oggetto contenuto nella colonna.

Qui abbiamo un `pd.Series` con float. Il dtype sarà `float` .

Quindi usiamo `astype` per "lanciarlo" su oggetto.

```
pd.Series([1.,2.,3.,4.,5.]).astype(object)
0    1
1    2
2    3
3    4
4    5
dtype: object
```

Il dtype è ora oggetto, ma gli oggetti nell'elenco sono ancora mobili. Logico se sai che in python tutto è un oggetto e può essere upcasted per oggetto.

```
type(pd.Series([1.,2.,3.,4.,5.]).astype(object)[0])
float
```

Qui proviamo a "lanciare" i float alle stringhe.

```
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
```

Il dtype è ora oggetto, ma il tipo di voci nell'elenco sono stringhe. Questo perché `numpy` non ha a che fare con le stringhe e quindi agisce come se fossero solo oggetti e di nessun interesse.

```
type(pd.Series([1.,2.,3.,4.,5.]).astype(str)[0])
str
```

Non fidatevi dei dtypes, sono un artefatto di un difetto architettonico nei panda. Specificate come necessario, ma non fare affidamento su quale dtype è impostato su una colonna.

Examples

Verifica dei tipi di colonne

I tipi di colonne possono essere controllati da `.dtypes` atrribute di 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
```

Per una singola serie, puoi usare `.dtype` attributo `.dtype` .

```
In [4]: df['A'].dtype
Out[4]: dtype('int64')
```

Cambiare i dtypes

`astype()` metodo `astype()` cambia il dtype di una serie e restituisce una nuova 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
```

Cambia il tipo di colonna A in float e il tipo di colonna B in intero:

```
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()` metodo `astype()` è per la conversione del tipo specifico (cioè è possibile specificare `.astype(float64')` , `.astype(float32)` o `.astype(float16)`). Per la conversione generale, è possibile utilizzare `pd.to_numeric` , `pd.to_datetime` e `pd.to_timedelta` .

Cambiando il tipo in numerico

`pd.to_numeric` cambia i valori in un tipo numerico.

```
In [6]: pd.to_numeric(df['E'])
Out[6]:
0    1
1    2
2    3
Name: E, dtype: int64
```

Per impostazione predefinita, `pd.to_numeric` genera un errore se un input non può essere convertito in un numero. È possibile modificare tale comportamento utilizzando il parametro `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
```

Se necessario, controlla tutte le righe con input non può essere convertito in numerico usa `boolean indexing` 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
```

Modifica del tipo in data / ora

```
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]
```

Si noti che il 2.1.2011 viene convertito al 1 ° febbraio 2011. Se si desidera invece il 2 gennaio 2011, è necessario utilizzare il parametro `dayfirst` .

```
In [13]: pd.to_datetime('2.1.2011', dayfirst=True)
Out[13]: Timestamp('2011-01-02 00:00:00')
```

Cambiando il tipo in timedelta

```
In [14]: pd.to_timedelta(df['D'])
Out[14]:
0   1 days
1   2 days
2   3 days
Name: D, dtype: timedelta64[ns]
```

Selezione delle colonne in base al dtype

`select_dtypes` metodo `select_dtypes` può essere utilizzato per selezionare le colonne in base al 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 i parametri di `include` ed `exclude` puoi specificare quali tipi desideri:

```
# 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
```

Riassumendo i dtypes

`get_dtype_counts` metodo `get_dtype_counts` può essere utilizzato per vedere una ripartizione dei dtypes.

```
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
```

Leggi Tipi di dati online: https://riptutorial.com/it/pandas/topic/2959/tipi-di-dati

Capitolo 38: Trattare con variabili categoriali

Examples

Codifica unica 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
```

Leggi Trattare con variabili categoriali online: https://riptutorial.com/it/pandas/topic/5999/trattare-con-variabili-categoriali

Capitolo 39: Unisci, unisciti e concatena

Sintassi

- Dataframe. **unire** (a *destra, come = 'inner', on = None, left_on = None, right_on = None, left_index = False, right_index = False, sort = False, suffixes = ('_ x', '_y'), copy = True, indicatore = Falso*)
- Unisci oggetti DataFrame eseguendo un'operazione di join in stile database per colonne o indici.
- Se si uniscono colonne su colonne, gli indici DataFrame verranno ignorati. Altrimenti, se si uniscono indici su indici o indici su una colonna o colonne, l'indice verrà passato.

Parametri

parametri	Spiegazione
destra	dataframe
Come	{'left', 'right', 'outer', 'inner'}, default 'inner'
lasciato acceso	etichetta o lista, o array-like. Nomi dei campi per unirsi in DataFrame a sinistra. Può essere un vettore o un elenco di vettori della lunghezza di DataFrame per utilizzare un vettore particolare come chiave di join anziché colonne
right_on	etichetta o lista, o array-like. Nomi dei campi per unirmi a destra in DataFrame o in un vettore / elenco di vettori per left_on docs
left_index	booleano, predefinito False. Utilizzare l'indice da sinistra DataFrame come chiave di join. Se è un MultiIndex, il numero di chiavi nell'altro DataFrame (l'indice o un numero di colonne) deve corrispondere al numero di livelli
right_index	booleano, predefinito False. Utilizzare l'indice dal DataFrame corretto come chiave di join. Gli stessi avvertimenti di left_index
ordinare	booleano, predefinito Fals. Ordinare le chiavi di join lessicograficamente nel risultato DataFrame
suffissi	Sequenza di 2 lunghezze (tupla, lista, ...). Suffisso da applicare ai nomi di colonne sovrapposte rispettivamente sul lato sinistro e destro
copia	booleano, predefinito True. Se False, non copiare i dati inutilmente
indicatore	booleano o stringa, predefinito False. Se True, aggiunge una colonna per emettere DataFrame chiamato "_merge" con informazioni sull'origine di ogni riga. Se stringa, la colonna con le informazioni sull'origine di ogni riga verrà

parametri	Spiegazione
	aggiunta all'output DataFrame e la colonna verrà denominata con il valore di stringa. La colonna Informazioni è di tipo Categoriale e assume un valore di "left_only" per le osservazioni la cui chiave di unione viene visualizzata solo in "Data" di sinistra, "right_only" per le osservazioni la cui chiave di unione viene visualizzata solo in "Data" DataFrame e "entrambi" se la la chiave di fusione dell'osservazione si trova in entrambi.

Examples

fondersi

Ad esempio, vengono date due tabelle,

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
```

L'obiettivo è ottenere la nuova tabella T3:

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

Che è quello di creare colonne `s1` , `s2` e `s3` , ciascuna corrispondente a una riga (il numero di righe per `id` è sempre fisso e uguale a 3)

Applicando `join` (che accetta un argomento opzionale che può essere una colonna o più nomi di colonne, che specifica che il DataFrame passato deve essere allineato su quella colonna in DataFrame). Quindi la soluzione può essere come mostrato di seguito:

df = df1.merge (df2.groupby ('id') ['signal']. 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
```

Se li 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
```

Unione di due 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
```

Join interno:

Utilizza l'intersezione di chiavi da due 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
```

In alternativa, specificare l'intersezione di chiavi da due 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
```

Join esterno:

Utilizza l'unione delle chiavi da due 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
```

Unire a sinistra:

Utilizza solo le chiavi da DataFrame sinistro.

```
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
```

Giusto Iscriviti

Utilizza solo le chiavi dal giusto DataFrame.

```
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
```

Unione / concatenazione / unione di più frame di dati (orizzontalmente e verticalmente)

genera frame di dati di esempio:

```
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
```

unire / unire / concatenare i frame di dati [df1, df2, df3] verticalmente - aggiungere righe

```
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
```

unione / unione / concatenazione di frame di dati orizzontalmente (allineamento per indice):

```
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
```

Unisci, Unisci e Concat

L'unione dei nomi delle chiavi è la stessa

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

L'unione dei nomi delle chiavi è diversa

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

Diversi tipi di unione

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

Unione su più chiavi

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

Trattamento di colonne sovrapposte

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

Utilizzo dell'indice di riga invece delle chiavi di unione

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

Evitare l'uso della sintassi `.join` in quanto fornisce un'eccezione per le colonne sovrapposte

Fusione sull'indice dataframe sinistro e colonna destra dataframe

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

Concetti i dati

Incollato verticalmente

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

Incollato orizzontalmente

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

Qual è la differenza tra join e merging

Considera i dati a `left` e a `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

Pensa di `join` come vuoi combinare ai dataframes in base ai loro rispettivi indici. Se ci sono colonne sovrapposte, `join` ti consente di aggiungere un suffisso al nome della colonna sovrapposta dal dataframe sinistro. I nostri due dataframe hanno un nome di colonna sovrapposto `A`

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

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

Si noti che l'indice è conservato e abbiamo 4 colonne. 2 colonne da `left` e 2 da `right` .

Se gli indici non si allineano

```
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
```

Ho usato un'unione esterna per illustrare meglio il punto. Se gli indici non si allineano, il risultato sarà l'unione degli indici.

Possiamo dire a `join` di utilizzare una colonna specifica nel dataframe sinistro da utilizzare come chiave di join, ma continuerà a utilizzare l'indice dalla destra.

```
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

Pensa a `merge` come allineato sulle colonne. Per impostazione predefinita, l' `merge` cerca le colonne sovrapposte su cui fondersi. `merge` consente un migliore controllo sulle chiavi di unione consentendo all'utente di specificare un sottoinsieme delle colonne sovrapposte da utilizzare con il parametro `on` o consentire separatamente la specifica di quali colonne a sinistra e di quali colonne sulla destra devono essere unite.

`merge` restituirà un dataframe combinato in cui l'indice verrà distrutto.

Questo semplice esempio trova la colonna sovrapposta come `'A'` e combina in base ad essa.

```
left.merge(right)

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

Nota che l'indice è `[0, 1]` e non più `['X', 'Y']`

È possibile specificare esplicitamente che si sta unendo l'indice con il parametro `left_index` o `right_index`

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

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

E questo sembra esattamente come l'esempio di `join` sopra.

Leggi Unisci, unisciti e concatena online: https://riptutorial.com/it/pandas/topic/1966/unisci--unisciti-e-concatena

Capitolo 40: Utilizzando .ix, .iloc, .loc, .at e .iat per accedere a un DataFrame

Examples

Utilizzando .iloc

.iloc usa interi per leggere e scrivere dati su un DataFrame.

Per prima cosa, creiamo un DataFrame:

```
df = pd.DataFrame({'one': [1, 2, 3, 4, 5],
                   'two': [6, 7, 8, 9, 10],
                }, index=['a', 'b', 'c', 'd', 'e'])
```

Questo DataFrame si presenta come:

```
   one  two
a    1    6
b    2    7
c    3    8
d    4    9
e    5   10
```

Ora possiamo usare .iloc per leggere e scrivere valori. Leggiamo la prima riga, prima colonna:

```
print df.iloc[0, 0]
```

Questo verrà stampato:

```
1
```

Possiamo anche impostare valori. Consente di impostare la seconda colonna, la seconda riga su qualcosa di nuovo:

```
df.iloc[1, 1] = '21'
```

E poi dai un'occhiata per vedere cosa è successo:

```
print df

   one two
a    1   6
b    2  21
c    3   8
d    4   9
e    5  10
```

Utilizzando .loc

.loc utilizza le **etichette** per leggere e scrivere dati.

Impostiamo un DataFrame:

```
df = pd.DataFrame({'one': [1, 2, 3, 4, 5],
                   'two': [6, 7, 8, 9, 10],
                  }, index=['a', 'b', 'c', 'd', 'e'])
```

Quindi possiamo stampare DataFrame per dare un'occhiata alla forma:

```
print df
```

Questo uscirà

```
   one  two
a    1    6
b    2    7
c    3    8
d    4    9
e    5   10
```

Utilizziamo le **etichette** di colonne e righe per accedere ai dati con .loc. Imposta la riga 'c', la colonna 'due' sul valore 33:

```
df.loc['c', 'two'] = 33
```

Ecco come appare DataFrame:

```
   one  two
a    1    6
b    2    7
c    3   33
d    4    9
e    5   10
```

Da notare che l'uso di `df['two'].loc['c'] = 33` potrebbe non riportare un avviso, e potrebbe anche funzionare, tuttavia, usando `df.loc['c', 'two']` è garantito che funzioni correttamente mentre il primo non lo è.

Possiamo leggere fette di dati, per esempio

```
print df.loc['a':'c']
```

stamperà le righe da a a c. Questo è inclusivo.

```
   one  two
a    1    6
b    2    7
```

```
c    3    8
```

E infine, possiamo fare entrambi insieme:

```
print df.loc['b':'d', 'two']
```

Emetterà le righe da b a c della colonna 'due'. Si noti che l'etichetta della colonna non viene stampata.

```
b    7
c    8
d    9
```

Se .loc viene fornito con un argomento intero che non è un'etichetta, viene ripristinato l'indicizzazione dei numeri interi degli assi (il comportamento di .iloc). Ciò rende possibile l'indicizzazione di etichette e interi misti:

```
df.loc['b', 1]
```

restituirà il valore nella seconda colonna (indice che inizia da 0) nella riga 'b':

```
7
```

Leggi Utilizzando .ix, .iloc, .loc, .at e .iat per accedere a un DataFrame online: https://riptutorial.com/it/pandas/topic/7074/utilizzando--ix---iloc---loc---at-e--iat-per-accedere-a-un-dataframe

Capitolo 41: Valori Mappa

Osservazioni

va detto che se il valore della chiave non esiste allora questo solleverà `KeyError` , in quelle situazioni è forse meglio usare l' `merge` o `get` che consente di specificare un valore predefinito se la chiave non esiste

Examples

Mappa dal Dizionario

A partire da un datafame `df` :

```
  U   L
111  en
112  en
112  es
113  es
113  ja
113  zh
114  es
```

Immagina di voler aggiungere una nuova colonna chiamata `S` prendendo i valori dal seguente dizionario:

```
d = {112: 'en', 113: 'es', 114: 'es', 111: 'en'}
```

Puoi utilizzare la `map` per eseguire una ricerca sui tasti restituendo i valori corrispondenti come una nuova colonna:

```
df['S'] = df['U'].map(d)
```

che restituisce:

```
  U   L   S
111  en  en
112  en  en
112  es  en
113  es  es
113  ja  es
113  zh  es
114  es  es
```

Leggi Valori Mappa online: https://riptutorial.com/it/pandas/topic/3928/valori-mappa

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con i panda	Alexander, Andy Hayden, ayhan, Bryce Frank, Community, hashcode55, Nikita Pestrov, user2314737
2	Aggiunta a DataFrame	shahins
3	Analisi: riunire tutto e prendere decisioni	piRSquared
4	Calendari delle festività	Romain
5	Creazione di DataFrames	Ahamed Mustafa M, Alexander, ayhan, Ayush Kumar Singh, bernie, Gal Dreiman, Geeklhem, Gorkem Ozkaya, jasimpson, jezrael, JJD, Julien Marrec, MaxU, Merlin, pylang, Romain, SerialDev, user2314737, vaerek, ysearka
6	Dati categoriali	jezrael, Julien Marrec
7	Dati duplicati	ayhan, Ayush Kumar Singh, bee-sting, jezrael
8	Dati mancanti	Andy Hayden, ayhan, EdChum, jezrael, Zdenek
9	Gotchas di panda	vlad.rad
10	Grafici e visualizzazioni	Ami Tavory, Nikita Pestrov, Scimonster
11	Indicizzazione booleana dei dataframes	firelynx
12	Indicizzazione e selezione dei dati	amin, Andy Hayden, ayhan, double0darbo, jasimpson, jezrael, Joseph Dasenbrock, MaxU, Merlin, piRSquared, SerialDev, user2314737
13	IO per Google BigQuery	ayhan, tworec
14	JSON	PinoSan, SerialDev, user2314737
15	Lavorare con Time Series	Romain

16	Leggi MySQL su DataFrame	andyabel, rrawat
17	Leggi SQL Server su Dataframe	bernie, SerialDev
18	Lettura dei file in DataFrame panda	Arthur Camara, bee-sting, Corey Petty, Sirajus Salayhin
19	Manipolazione delle stringhe	ayhan, mnoronha, SerialDev
20	Meta: linee guida per la documentazione	Andy Hayden, ayhan, Stephen Leppik
21	MultiIndex	Andy Hayden, benten, danielhadar, danio, Pedro M Duarte
22	Ottenere informazioni su DataFrames	Alexander, ayhan, Ayush Kumar Singh, bernie, Romain, ysearka
23	Pandas Datareader	Alexander, MaxU
24	pd.DataFrame.apply	ptsw, Romain
25	Raggruppamento di dati	Andy Hayden, ayhan, danio, Geeklhem, jezrael, ℕ𝕠𝕠B̲I̲E̲, QM.py, Romain, user2314737
26	Raggruppamento di dati di serie temporali	ayhan, piRSquared
27	Rendere i panda divertenti con i tipi di dati nativi di Python	DataSwede
28	ricampionamento	jezrael
29	Rimodellamento e rotazione	Albert Camps, ayhan, bernie, DataSwede, jezrael, MaxU, Merlin
30	Salva pandas dataframe in un file csv	amin, bernie, eraoul, Gal Dreiman, maxliving, Musafir Safwan, Nikita Pestrov, Olel Daniel, Stephan
31	Semplice manipolazione di DataFrames	Alexander, ayhan, Ayush Kumar Singh, Gal Dreiman, Geeklhem, MaxU, paulo.filip3, R.M., SerialDev, user2314737, ysearka

32	Serie	Alexander, daphshez, EdChum, jezrael, shahins
33	Sezioni trasversali di diversi assi con MultiIndex	Julien Marrec
34	Spostamento e ritardo dei dati	ASGM
35	Strumenti computazionali	Ami Tavory
36	Strumenti IO di Pandas (lettura e salvataggio di set di dati)	amin, Andy Hayden, bernie, Fabich, Gal Dreiman, jezrael, João Almeida, Julien Spronck, MaxU, Nikita Pestrov, SerialDev, user2314737
37	Tipi di dati	Andy Hayden, ayhan, firelynx, jezrael
38	Trattare con variabili categoriali	Gorkem Ozkaya
39	Unisci, unisciti e concatena	ayhan, Josh Garlitos, MaThMaX, MaxU, piRSquared, SerialDev, varunsinghal
40	Utilizzando .ix, .iloc, .loc, .at e .iat per accedere a un DataFrame	bee-sting, DataSwede, farleytpm
41	Valori Mappa	EdChum, Fabio Lamanna