Corso Python per docenti

Lezione A07 – Pandas

Alberto Montresor

Università di Trento

2021/02/10

Acknowledgments: Stefano Teso, Pandas Documentation

http://disi.unitn.it/~teso/courses/sciprog/python_pandas.html

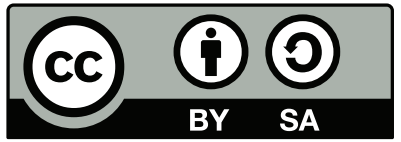


Table of contents

Cos'è Pandas?

Pandas

Una libreria gratuita per caricare, manipolare e visualizzare dati sequenziali e in forma di tabella, come serie temporali o fogli csv.

Caratteristiche

- Caricare e salvare in formati tabulari "standard":
  - CSV (Comma-separated Values)
  - TSV (Tab-separated Values)
  - File Excel
  - Formati Database, etc.
- Indicizzazione e aggregazione flessibile di serie e tabelle
- Operazioni numeriche e statistiche efficienti
- Visualizzazione facile e immediata

Alcuni link

Website Pandas

`http://pandas.pydata.org/`

Documentazione officiale

`http://pandas.pydata.org/pandas-docs/stable/dsintro.html`

Codice sorgente

`https://github.com/pandas-dev/pandas/`

Una dimostrazione rapida – Iris Dataset

```
SepalLength,SepalWidth,PetalLength,PetalWidth,Name
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
...
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
...
5.7,2.8,4.1,1.3,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
...
```

https://drive.google.com/open?id=0B0wILN942aEVYTVBekRHLTNON3c
https://en.wikipedia.org/wiki/Iris_flower_data_set

Una dimostrazione rapida – Iris Dataset

Per comprendere il dataset, vorremmo visualizzare la relazione fra le quattro proprietà per le sole righe Iris Virginica.

- Carica il set di dati analizzando tutte le righe nel file
- Conserva solo le righe di Iris virginica
- Calcola statistiche sui valori delle righe, assicurandoti di convertire da stringa a float
- Realizziamo un plot "rapido" utilizzando una `matplotlib`.

Una dimostrazione rapida – Iris Dataset

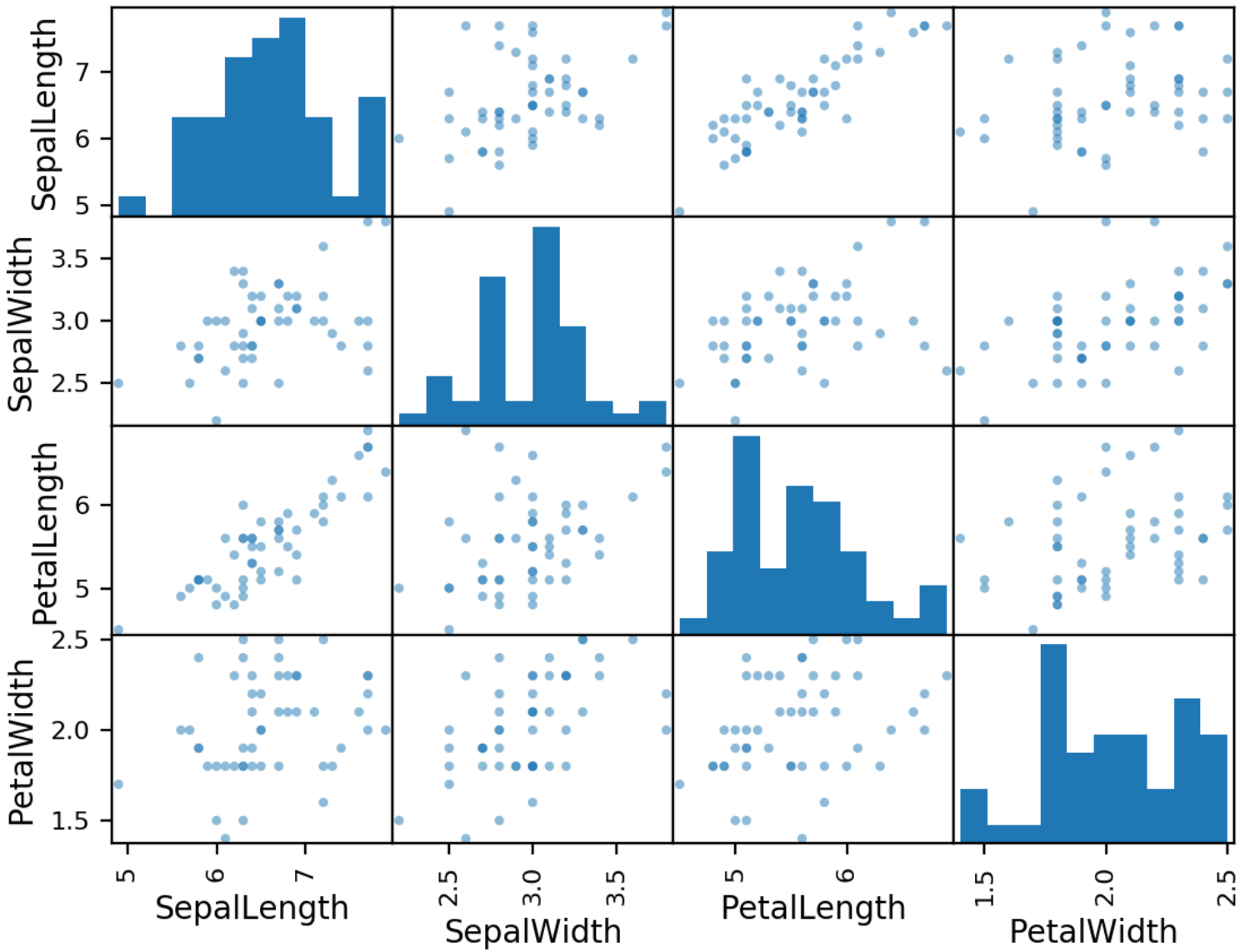
```
import pandas as pd
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt

dataframe = pd.read_csv("iris.csv")

scatter_matrix(dataframe[dataframe.Name == "Iris-virginica"])

plt.show()
```

Una dimostrazione rapida – Iris Dataset



Introduction to Pandas

Pandas fornisce due datatype principali:

- Le `Series` rappresentano dati 1D, come sequenze temporali, calendari, l'output di una funzione ad una variabile, etc.
- I `DataFrame` rappresentano dati 2D, come file CSV, una tabella di Database, una matrice, etc.

Ogni colonna di un `DataFrame` è una `Series`.

- Per questo motivo analizzeremo prima le `Series`.
- Molto di quello che diremo per le `Series` si applica anche ai `DataFrames`.

Table of contents

Pandas: `Series`

Series

Una `Series` è un array monodimensionale con un asse etichettato, che può contenere oggetti arbitrari.

L'asse è chiamato index, e può essere utilizzato per accedere agli elementi. è molto flessibile, e non necessariamente numerico.

Funziona in parte come una `list` e in parte come un `dict`.

Creare una `Series` (1)

È possibile specificare semplicemente la serie di dati, associando un indice numerico implicito.

```
import pandas as pd
s = pd.Series(["a", "b", "c"])
print(s)
```

```
0    a
1    b
2    c
dtype: object
```

Creare una `Series` (2)

È possibile specificare sia la serie di dati che l'indice esplicito separatamente:

```
import pandas as pd
s = pd.Series(["a", "b", "c"], index=[2, 5, 8])
print(s)
```

```
2    a
5    b
8    c
dtype: object
```

Creare una `Series` (3)

È possibile specificare sia la serie che l'indice, tramite un singolo dizionario:

```
import pandas as pd
s = pd.Series({"a": "A", "b": "B", "c": "C"})
print(s)
```

```
a    A
b    B
c    C
dtype: object
```

Creare una `Series` (4)

Se viene dato un singolo valore scalare (ad es. intero), il costruttore della serie lo replicherà per tutti gli indici (che devono essere specificati)

```
import pandas as pd
s = pd.Series(3, index=range(5))
print(s)
```

```
0    3
1    3
2    3
3    3
4    3
dtype: int64
```

Accedere a una `Series`

Creiamo una serie che rappresenta le ore lavorative nell'ultima settimana. Possiamo accedervi attraverso la posizione (come una lista) o l'indice (come un dizionario).

```
import pandas as pd
days = ["mon", "tue", "wed", "thu", "fri"]
workhours = [6, 2, 8, 5, 9]
work = pd.Series(workhours, index=days)
print(work["mon"])
work["tue"]=3
print(seelp[1])
```

```
6
3
```

Accedere a una Series

- Se un'etichetta non esiste, viene restituito un errore
- Usando il metodo `get()`, nel caso di etichetta mancante viene restituito `None` oppure un valore di default specificato

```
import pandas as pd
days =       ["mon", "tue", "wed", "thu", "fri"]
workhours = [6,      2,     8,     5,     9]
work = pd.Series(workhours, index=days)

print(work.get('sat'))
print(work.get('sat',0))
print(work["sat"])
```

```
None
0
KeyError: 'sat'
```

Iterating over a `Series`

- Potete usare `for index, value in <series>.items()` per iterare su una serie

```
import pandas as pd
days =       ["mon", "tue", "wed", "thu", "fri"]
workhours = [6,      2,     8,     5,     9]
work = pd.Series(workhours, index=days)
for index, value in work.items():
  print(index, value)
```

```
mon 6
tue 2
wed 8
thu 5
fri 9
```

Slicing su Series

Possiamo anche utilizzare l'operatore slice, come faremmo con una lista. Notate che sia i dati che l'indice vengono estratti correttamente. funziona anche con le etichette.

```
print(work[-3:])
print(work["tue":"thu"])
```

```
wed    8
thu    5
fri    9
dtype: int64
tue    2
wed    8
thu    5
dtype: int64
```

Head e tail

I primi e ultimi `n` elementi possono essere estratti utilizzando `head()` e `tail()`, rispettivamente.

```
print(work.head(2))
print(work.tail(3))
```

```
mon    6
tue    2
dtype: int64

wed    8
thu    5
fri    9
dtype: int64
```

List of indexes

Potete anche esplicitare una lista di posizioni. Le tuple non funzionano, perché sono interpretate come indici potenziali.

```
print(work[[0, 1, 2]])
print(work[["mon", "wed", "fri"]])
```

```
mon    6
tue    2
wed    8
dtype: int64

mon    6
wed    8
fri    9
dtype: int64
```

Operator broadcasting

La classe `Series` fa il broadcast delle operazioni aritmetiche per uno scalere a tutti i suoi elementi.

```
print(work)
```

```
mon    6
tue    2
wed    8
thu    5
fri    9
dtype: int64
```

```
print(work+1)
```

```
mon     7
tue     3
wed     9
thu     6
fri    10
dtype: int64
```

```
print(work*2)
```

```
mon    12
tue     4
wed    16
thu    10
fri    18
dtype: int64
```

Nota

Il concetto di operator broadcasting è preso dalla libreria `numpy`, una delle librerie principali per scrivere codice numerico in Python.

Può essere visto come una versione generalizzata del prodotto scalare dell'algebra lineare.

Le regole che governano il broadcasting possono essere molto complesse. per il momento, parleremo solo di operazioni per costanti.

Masking e filtering

- Oltre agli operatori numerici, possiamo applicare anche condizioni booleane. Il risultato è detto mask.
- Le mask possono essere utilizzate per filtrare filter gli elementi di una `Series` in accordo con una certa condizione.

```
print(work)
```

```
mon    6
tue    2
wed    8
thu    5
fri    9
dtype: int64
```

```
print(work>=6)
```

```
mon     True
tue    False
wed     True
thu    False
fri     True
dtype: bool
```

```
print(work[work>=6])
```

```
mon    6
wed    8
fri    9
dtype: int64
```

Assegnamento automatico di etichette

Le operazioni fra serie multiple vengono allineate per etichetta, il che significa che gli elementi con la stessa etichetta vengono associati per effettuare le operazioni.

```
print(work)
```

```
mon    6
tue    2
wed    8
thu    5
fri    9
dtype: int64
```

```
print(fun)
```

```
tue    1
wed    1
thu    3
fri    5
sat    8
sun    8
dtype: int64
```

```
print(work+fun)
```

```
fri    14.0
mon     NaN
sat     NaN
sun     NaN
thu     8.0
tue     3.0
wed     9.0
dtype: float64
```

Not-a-Number (`NaN`)

L'indice della `Series` risultante è l'unione degli indici degli operandi. Ciò che accade dipende dalla presenza o meno di una determinata etichetta in entrambe le serie di input:

- Per le etichette comuni (nel nostro caso `"tue"`, `"wed"`, `"thu"`), la `Series` di output contiene la somma degli elementi allineati.
- Per le etichette che compaiono solo in uno degli operandi (`"mon"` e `"fri"`), il risultato è un `NaN`, cioè not-a-number

`NaN` è solo una costante simbolica che specifica che l'oggetto è un numero entità con un valore non valido o indefinito.

Gestire i valori mancanti

Ci sono strategie differenti per gestire NaNs, e nessuna è la migliore: dovete scegliere quella più indicata per il problema che volete risolvere.

```
t = work+fun
print(t)
```

```
fri    14.0
mon     NaN
sat     NaN
sun     NaN
thu     8.0
tue     3.0
wed     9.0
dtype: float64
```

```
nt = t.dropna()
print(nt)
```

```
fri    14.0
thu     8.0
tue     3.0
wed     9.0
```

```
zt = t.fillna(0.0)
print(zt)
```

```
fri    14.0
mon     0.0
sat     0.0
sun     0.0
thu     8.0
tue     3.0
wed     9.0
```

Assegnamenti automatici delle etichette

Attraverso il metodo `add()`, è possibile assegnare un valore di riempimento per le entry mancanti, al fine di ottenere una somma realistica.

```
print(work)
```

```
mon    6
tue    2
wed    8
thu    5
fri    9
dtype: int64
```

```
print(fun)
```

```
tue    1
wed    1
thu    3
fri    5
sat    8
sun    8
dtype: int64
```

```
print(work.add(fun,
  fill_value=0))
```

```
fri     9.0
mon     6.0
thu    10.0
tue     4.0
wed    16.0
dtype: float64
```

Calcolo statistiche

```
print(s)
```

```
mon    6
tue    2
wed    8
thu    5
fri    9
dtype: int64
```

```
print(s.sum())
print(s.prod())
print(s.max())
print(s.argmax())
print(s.mean())
print(s.var())
print(s.std())
print(s.median())
```

```
30
4320
9
fri
6.0
7.5
2.7386127875258306
6.0
```

```
print(s.cumsum())
```

```
mon     6
tue     8
wed    16
thu    21
fri    30
dtype: int64
```

Calcolo statistiche

```
print(s)
```

```
mon    6
tue    2
wed    8
thu    5
fri    9
dtype: int64
```

```
print(s.quantile(0.5))
```

```
6.0
```

```
print(s.quantile(
  [0.25, 0.5, 0.75]
))
```

```
0.25    5.0
0.50    6.0
0.75    8.0
dtype: float64
```

```
print(s[s >=
  s.quantile(0.5)
])
```

```
mon    6
wed    8
fri    9
dtype: int64
```

Calcolo statistiche

```
print(s)
```

```
mon    6
tue    2
wed    8
thu    5
fri    9
dtype: int64
```

```
# Pearson corr.
print(s.corr(s))                1.0

# Spearman corr.
print(s.corr(s,                 1.0
  method="spearman"
)

# Autocorrelation
# with time lag
print(s.autocorr(lag=0))        1.0
print(s.autocorr(lag=1))        -0.548128127763
print(s.autocorr(lag=2))        0.995870594886
```

Series: Conclusioni

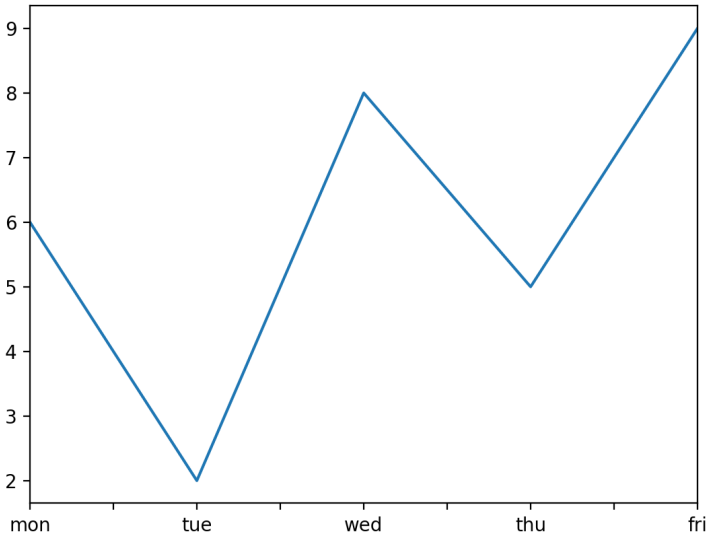
Un modo veloce per ottenere statistiche utili è quello di utilizzare `s.describe()`. In ogni caso, la lista dei metodi statistici associati alle `Series` è molto più grande.

```
print(s.describe())
```

```
count    5.000000
mean     6.000000
std      2.738613
min      2.000000
25%      5.000000
50%      6.000000
75%      8.000000
max      9.000000
dtype: float64
```

Series: Plotting

```
import pandas as pd
import matplotlib.pyplot as plt
workhours = [6, 2, 8, 5, 9]
days = ["mon", "tue", "wed", "thu", "fri"]
s = pd.Series(workhours, index=days)
s.plot()
plt.show()
```




```
import pandas as pd
import matplotlib.pyplot as plt
workhours = [6, 2, 8, 5, 9]
days = ["mon", "tue", "wed", "thu", "fri"]
s = pd.Series(workhours, index=days)
s.plot(kind="bar")
plt.show()
```

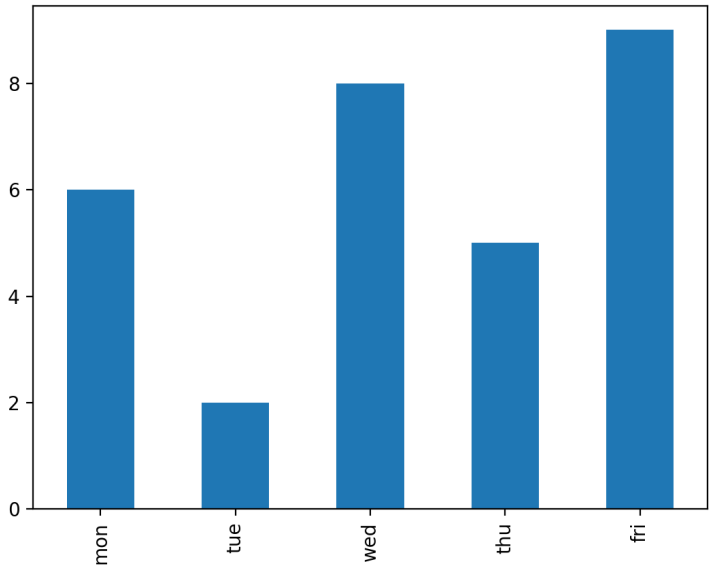


Table of contents

DataFrame

I `DataFrame` di Pandas sono l'analogo 2D delle `Series`: sono essenzialmente una tabella di oggetti eterogenei.

Un `DataFrame` è dato da tre attributi principali

- l'index, che contiene le etichette delle righe
- le column, che contiene le etichette delle colonne
- la shape, che descrive le dimensioni delle tabella

Quando estraete una colonna da un `DataFrame`, ottenete una `Series` e potete utilizzare i meccanismi visti nelle sezioni precedenti.

Inoltre, gran parte delle operazioni che possono essere eseguite su una `Series`, possono essere eseguite su un intero `DataFrame`.

Creare un DataFrame da un dizionario di Series

```
import pandas as pd
d = { "name": pd.Series(["bobby", "ronald", "ronald", "ronald"]),
  "surname": pd.Series(["fisher", "fisher", "reagan", "mcdonald"]) }
df = pd.DataFrame(d)
print(df)
print(df.columns)
print(df.index)
print(df.shape)
```

```
     name   surname
0   bobby    fisher
1  ronald    fisher
2  ronald    reagan
3  ronald  mcdonald
Index(['name', 'surname'], dtype='object')
RangeIndex(start=0, stop=4, step=1)
(4, 2)
```

- Le chiavi del dizionario diventano i nomi delle colonne dei dataframe
- L'indice delle varie Series diventa l'indice del dataframe.

Creare un DataFrame da un dizionario di Series

```
import pandas as pd
d = { "x": pd.Series([0, 0], index=["a", "b"]),
      "y": pd.Series([0, 0], index=["b", "c"])}
df = pd.DataFrame(d)
print(df)
print(df.columns)
print(df.index)
print(df.shape)
```

```
     x    y
a  0.0  NaN
b  0.0  0.0
c  NaN  0.0
Index(['x', 'y'], dtype='object')
Index(['a', 'b', 'c'], dtype='object')
(3,2)
```

- Se gli indice delle serie di input non corrispondono, le regole di allineamento vengono applicate e i valori mancanti diventano `NaN`.

Creare un `DataFrame` da un dizionario di `list`

```
import pandas as pd
d = { "column1": [1.0, 2.0, 6.0, -1.0],
      "column2": [0.0, 1.0, -2.0, 4.0] }
df = pd.DataFrame(d)
print(df)
print(df.columns)
print(df.index)
print(df.shape)
```

- I nomi delle colonne sono presi dalle chiavi
- L'indice è settato a quello standard (numerico)

```
   column1  column2
0      1.0      0.0
1      2.0      1.0
2      6.0     -2.0
3     -1.0      4.0
Index(['column1', 'column2'], dtype='object')
RangeIndex(start=0, stop=4, step=1)
(4,2)
```

Creare un `DataFrame` da un dizionario di `list`, con indice

```
import pandas as pd
d = { "column1": [1., 2., 6., -1.],
      "column2": [0., 1., -2., 4.] }
df = pd.DataFrame(d, index=["a", "b", "c", "d"])
print(df)
print(df.columns)
print(df.index)
print(df.shape)
```

- È possibile specificare un indice custom

```
   column1  column2
a      1.0      0.0
b      2.0      1.0
c      6.0     -2.0
d     -1.0      4.0
Index(['column1', 'column2'], dtype='object')
RangeIndex(start=0, stop=4, step=1)
(4,2)
```

Creare un `DataFrame` da una lita di dizionari

```
import pandas as pd
d = [ {"a": 1, "b": 2},
      {"a": 2, "c": 3},
]
df = pd.DataFrame(d)
print(df)
print(df.columns)
print(df.index)
print(df.shape)
```

```
   a    b    c
0  1  2.0  NaN
1  2  NaN  3.0
Index(['a', 'b', 'c'], dtype='object')
RangeIndex(start=0, stop=2, step=1)
(2,3)
```

- I nomi delle colonne sono prese dagli indicizi dei dizionari
- L'indice è quello di default
- Poiché non tutte le chiavi appaiono in tutti i dizionari, i valori mancanti diventano `NaN`.

Caricare un file CSV (Excel, etc.)

```
import pandas as pd
df = pd.read_csv("iris.csv")
print(df.columns)
print(df.index)
print(df.shape)
```

```
Index(['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', 'Name'],
  dtype='object')
RangeIndex(start=0, stop=150, step=1)
(150, 5)
     SepalLength  SepalWidth  PetalLength  PetalWidth            Name
0            5.1         3.5          1.4         0.2     Iris-setosa
1            4.9         3.0          1.4         0.2     Iris-setosa
2            4.7         3.2          1.3         0.2     Iris-setosa
3            4.6         3.1          1.5         0.2     Iris-setosa
4            5.0         3.6          1.4         0.2     Iris-setosa
5            5.4         3.9          1.7         0.4     Iris-setosa
...
```

help(pd.read_csv)

```
Help on function read_csv in module pandas.io.parsers:

read_csv(filepath_or_buffer, sep=',', delimiter=None, header='infer',
names=None, index_col=None, usecols=None, squeeze=False, prefix=None,
mangle_dupe_cols=True, dtype=None, engine=None, converters=None,
true_values=None, false_values=None, skipinitialspace=False,
skiprows=None, nrows=None, na_values=None, keep_default_na=True,
na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False,
infer_datetime_format=False, keep_date_col=False, date_parser=None,
dayfirst=False, iterator=False, chunksize=None, compression='infer',
thousands=None, decimal=b'.', lineterminator=None, quotechar='"',
quoting=0, escapechar=None, comment=None, encoding=None, dialect=None,
tupleize_cols=False, error_bad_lines=True, warn_bad_lines=True,
skipfooter=0, skip_footer=0, doublequote=True, delim_whitespace=False,
as_recarray=False, compact_ints=False, use_unsigned=False,
low_memory=True, buffer_lines=None, memory_map=False, float_precision=None)

Read CSV (comma-separated) file into DataFrame

Also supports optionally iterating or breaking of the file into chunks.
```

Caricare un file CSV separato da tab, malformato

```
https://drive.google.com/open?id=0B0wILN942aEVeWdScy1wQnA3LTA
```

Descrive un mapping fra ID di protein prese da UniProt (sequenze) a occorrenze nella Protein Data Bank (i.e. structure). Il file TSV è come questo:

```
# 2014/07/08 - 14:59
SP_PRIMARY  PDB
A0A011      3vk5;3vka;3vkb;3vkc;3vkd
A0A2Y1      2jrd
A0A585      4mnq
A0A5B4      2ij0
A0A5B9      2bnq;2eyr;2eys;2eyt;3kxf;3o6f;3o8x;3o9w;3qux;3t0e;3
A0A5E3      1hq4;1ob1
A0AEF5      4iu2;4iu3
A0AEF6      4iu2;4iu3
```

Caricare un file CSV separato da tab, malformato

Possiamo usare l'argomento `sep` (separatore) di `read_csv()` per separare attraverso TAB.

```
df = pd.read_csv("uniprot_pdb.tsv", sep="\t")
print(df.shape)
print(df.columns)
print(df.head(3))
```

```
(33637, 1)
Index(['# 2014/07/08 - 14:59'], dtype='object')
                 # 2014/07/08 - 14:59
SP_PRIMARY                        PDB
A0A011      3vk5;3vka;3vkb;3vkc;3vkd
A0A2Y1                         2jrd
```

Problema: la prima riga contiene solo una colonna, quindi Pandas pensa che ci sia solo una colonna.

Caricare un file CSV separato da tab, malformato

L'argomento `skiprows` viene utilizzato per saltare la prima riga, che è un commento

```
df = pd.read_csv("uniprot_pdb.tsv", sep="\t", skiprows=1)
print(df.shape)
print(df.columns)
print(df.head(3))
```

```
(33636, 2)
Index(['SP_PRIMARY', 'PDB'], dtype='object')
  SP_PRIMARY                                PDB
0     A0A011  3vk5;3vka;3vkb;3vkc;3vkd
1     A0A2Y1                               2jrd
2     A0A585                               4mnq
```

Caricare un file CSV separato da tab, malformato

L'argomento comment viene utilizzato per saltare tutte le righe che iniziano con #

```
df = pd.read_csv("uniprot_pdb.tsv", sep="\t", comment="#")
print(df.shape)
print(df.columns)
print(df.head(3))
```

```
(33636, 2)
Index(['SP_PRIMARY', 'PDB'], dtype='object')
  SP_PRIMARY                                PDB
0     A0A011  3vk5;3vka;3vkb;3vkc;3vkd
1     A0A2Y1                               2jrd
2     A0A585                               4mnq
```

Estrarre righe e colonne

Operazione	**Sintassi**	**Risultato**
Selezionare colonne	`df[col]`	`Series`
Selezionare colonne multiple	`df[[col1,col2]]`	`DataFrame`
Selezionare righe per indice	`df.loc[label]`	`Series`
Selezionare righe per posizione	`df.iloc[loc]`	`Series`
Fare slicing sulle righe	`df[5:10]`	`DataFrame`
Selezionare righe tramite vettore booleano	`df[bool_vec]`	`DataFrame`

Estrarre un sottoinsieme del Dataset Iris

Per semplicità, in quanto segue utilizzeremo un esempio random preso dal Dataset Iris, calcolato come segue:

```
import numpy as np
import pandas as pd
np.random.seed(0)
df = pd.read_csv("iris.csv")
small = df.iloc[np.random.permutation(df.shape[0])].head()
print(small.shape)
print(small)
```

Breve spiegazione: utilizziamo `numpy.random.permutation()` per generare una permutazione random degli indici da `0` a `df.shape[0]-1`, ovvero il numero di righe del Dataset Iris; quindi utilizziamo questa permutazione come indice di riga per permutare tutte le righe in `df`; infine, prendiamo le prime 5 righe della versione permutata di `df` utilizzando `head()`.

Estrarre un sottoinsieme del Dataset Iris

```
(5, 5)
     SepalLength  SepalWidth  PetalLength  PetalWidth           Na
114          5.8         2.8          5.1         2.4  Iris-virgin
62           6.0         2.2          4.0         1.0  Iris-versico
33           5.5         4.2          1.4         0.2     Iris-set
107          7.3         2.9          6.3         1.8  Iris-virgin
7            5.0         3.4          1.5         0.2     Iris-set
```

Estrarre una colonna

È possibile accedere ad una colonna di `small` con la notazione `[]`. Il risultato è una `Series`.

Se il nome della colonna è compatibile con le convenzioni di Python per i nomi delle variabili, è possibile usarlo come se fosse un attributo del dataframe.

```
print(small["Name"])        OR        print(small.Name)
```

```
114     Iris-virginica
62     Iris-versicolor
33         Iris-setosa
107     Iris-virginica
7          Iris-setosa
Name: Name, dtype: object
```

Estrarre colonne multiple

È possibile estrarre colonne multiple, specificando una lista di nomi di colonna. Il risultato è un `DataFrame`

```
print(small[["SepalLength", "PetalLength"]])
```

```
     SepalLength  PetalLength
114          5.8          5.1
62           6.0          4.0
33           5.5          1.4
107          7.3          6.3
7            5.0          1.5
```

Estrarre una riga

Per estrarre una riga, è possibile utilizzare gli attributi `loc` e `iloc` specificsando un'etichetta o una posizione, rispettivamente.
Il risultato è una `Series`

```
print(small.loc[114])        OR        print(small.iloc[0])
```

```
SepalLength                  5.8
SepalWidth                   2.8
PetalLength                  5.1
PetalWidth                   2.4
Name              Iris-virginica
Name: 114, dtype: object
```

Estrarre righe multiple

Per estrarre righe multiple, è possibile usare `loc` e `iloc` specificando una lista di etichette o posizioni.
Il risultato è un `Dataframe`

```
print(small.loc[[114,62,33]]) OR print(small.iloc[[0,1,2]])
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
114	5.8	2.8	5.1	2.4	Iris-virginica
62	6.0	2.2	4.0	1.0	Iris-versicolor
33	5.5	4.2	1.4	0.2	Iris-setosa

Broadcasting

Il broadcasting viene applicato automaticamente a tutte le righe, oppure a tutte le colonne.

```
print(small["SepalLength"] + small["SepalWidth"])
print(small+small)
```

```
114     8.6
62      8.2
33      9.7
107    10.2
7       8.4
dtype: float64
```

```
     SepalLength  SepalWidth  PetalLength  PetalWidth
114         11.6         5.6         10.2         4.8   Iris-virginicaIris-
62          12.0         4.4          8.0         2.0 Iris-versicolorIris-v
33          11.0         8.4          2.8         0.4         Iris-setosaIr
107         14.6         5.8         12.6         3.6   Iris-virginicaIris-
7           10.0         6.8          3.0         0.4         Iris-setosaIr
```

Statistics

Le statistiche possono essere applicati ad una riga, ad un colonna, o all'intera tabella.

```
print(small.loc[114][:-1].mean()) # Exclude last column, Name
print(small.PetalLength.mean())
print(small.mean())
```

```
4.025

3.66

SepalLength    5.92
SepalWidth     3.10
PetalLength    3.66
PetalWidth     1.12
dtype: float64
```

All together!

```
print(small[["Name", "PetalLength", "SepalLength"]][
  small.PetalLength > small.PetalLength.mean()])
```

```
                Name  PetalLength  SepalLength
114   Iris-virginica          5.1          5.8
62   Iris-versicolor          4.0          6.0
107   Iris-virginica          6.3          7.3
```

Merge

L'unione di dataframe differenti viene eseguita tramite la funzione `merge()`.

Fare merging significa che, date due tabelle con un nome di colonna in comune, vengono associate le righe con lo stesso valore in quella colonna e quindi una nuova tabella viene creata concatendando una riga in comune.

```
sequences = pd.DataFrame({
    "id":  ["Q99697", "O18400", "P78337", "Q9W5Z2"],
    "seq": ["METNCR", "MDRSSA", "MDAFKG", "MTSMKD"],
})

names = pd.DataFrame({
    "id":   ["Q99697", "O18400", "P78337", "P59583"],
    "name": ["PITX2_HUMAN", "PITX_DROME",
             "PITX1_HUMAN", "WRK32_ARATH"],
```

Merge - Inner

```
print(sequences)
print(names)
print(pd.merge(sequences, names, on="id", how="inner"))
```

```
       id     seq
0  Q99697  METNCR
1  O18400  MDRSSA
2  P78337  MDAFKG
3  Q9W5Z2  MTSMKD
```

```
       id          name
0  Q99697  PITX2_HUMAN
1  O18400   PITX_DROME
2  P78337  PITX1_HUMAN
3  P59583  WRK32_ARATH
```

```
       id     seq          name
0  Q99697  METNCR  PITX2_HUMAN
1  O18400  MDRSSA   PITX_DROME
2  P78337  MDAFKG  PITX1_HUMAN
```

Id senza match vengono eliminati

Merge - Left

```
print(sequences)
print(names)
print(pd.merge(sequences, names, on="id", how="left"))
```

```
       id     seq
0  Q99697  METNCR
1  O18400  MDRSSA
2  P78337  MDAFKG
3  Q9W5Z2  MTSMKD
```

```
       id          name
0  Q99697  PITX2_HUMAN
1  O18400   PITX_DROME
2  P78337  PITX1_HUMAN
3  P59583  WRK32_ARATH
```

```
       id     seq          name
0  Q99697  METNCR  PITX2_HUMAN
1  O18400  MDRSSA   PITX_DROME
2  P78337  MDAFKG  PITX1_HUMAN
3  Q9W5Z2  MTSMKD           NaN
```

Gli id vengono presi dalla tabella di sinistra

Merge - Right

```
print(sequences)
print(names)
print(pd.merge(sequences, names, on="id", how="right"))
```

```
       id     seq
0  Q99697  METNCR
1  O18400  MDRSSA
2  P78337  MDAFKG
3  Q9W5Z2  MTSMKD
```

```
       id          name
0  Q99697  PITX2_HUMAN
1  O18400   PITX_DROME
2  P78337  PITX1_HUMAN
3  P59583  WRK32_ARATH
```

```
       id     seq          name
0  Q99697  METNCR  PITX2_HUMAN
1  O18400  MDRSSA   PITX_DROME
2  P78337  MDAFKG  PITX1_HUMAN
3  P59583     NaN  WRK32_ARATH
```

Gli id vengono presi dalla tabella di destra

Merge - Outer

```
print(sequences)
print(names)
print(pd.merge(sequences, names, on="id", how="outer"))
```

```
       id     seq
0  Q99697  METNCR
1  O18400  MDRSSA
2  P78337  MDAFKG
3  Q9W5Z2  MTSMKD
```

```
       id          name
0  Q99697  PITX2_HUMAN
1  O18400   PITX_DROME
2  P78337  PITX1_HUMAN
3  P59583  WRK32_ARATH
```

```
       id     seq          name
0  Q99697  METNCR  PITX2_HUMAN
1  O18400  MDRSSA   PITX_DROME
2  P78337  MDAFKG  PITX1_HUMAN
3  Q9W5Z2  MTSMKD           NaN
4  P59583     NaN  WRK32_ARATH
```

Si utilizzano tutti gli ID

Grouping Tables

Il metodo `groupby` è essenziale per fare operazioni su gruppi di righe.

Dato il dataset Iris, vogliamo calcolare la media delle quattro colonne numeriche per ognuna delle tre differenti specie di Iris.

Il risultato dovrebbe essere un dataframe con 3 righe (specie) per quattro colonne (petal/sepal length/width).

Raggruppare tabelle

```
import pandas as pd
iris = pd.read_csv("iris.csv")
print(iris.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
SepalLength    150 non-null float64
SepalWidth     150 non-null float64
PetalLength    150 non-null float64
PetalWidth     150 non-null float64
Name           150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

Raggruppare tabelle

Iterando sulla variabile `grouped` si ottengono tuple (value-of-Name, DataFrame):

- Il 1° item è il valore del nome della colonna condiviso dal gruppo
- Il 2° item è il `DataFrame` che include solo le righe di quel gruppo

```
grouped = iris.groupby(iris.Name)
for group in grouped:
    print(group[0], group[1].shape)
```

```
Iris-setosa (50, 5)
Iris-versicolor (50, 5)
Iris-virginica (50, 5)
```

Raggruppare tabelle

È possibile eseguire alcune trasformazioni (e.g. `mean()`) ai gruppi individuali in modo automatico, utilizzando il metodo `aggregate()` sulla variabile raggruppata. Il risultato di `aggregate()` è un dataframe.

```
iris_mean_by_name = grouped.aggregate(pd.DataFrame.mean)
print(iris_mean_by_name)
```

Name	SepalLength	SepalWidth	PetalLength	PetalWidth
Iris-setosa	5.006	3.418	1.464	0.244
Iris-versicolor	5.936	2.770	4.260	1.326
Iris-virginica	6.588	2.974	5.552	2.026