

Introduzione a Pandas

Contents

- 5.1. Series
- 5.2. DataFrame
- 5.3. Lettura di dati da file
- 5.4. Gestione dei dati mancanti
- 5.5. Rinominare le colonne
- 5.6. Estrarre i dati dal DataFrame
- 5.7. Selezione casuale di un sottoinsieme di righe
- 5.8. Selezione di colonne
- 5.9. Creare nuove colonne
- 5.10. Formato long e wide
- 5.11. Watermark



La libreria **Pandas** è delegata alla gestione e lettura dei dati provenienti da sorgenti eterogenee, tra cui fogli Excel, file CSV, o anche JSON e database di tipo SQL.

Pandas dipende da due strutture dati principali:

- la **Series** che viene utilizzata per rappresentare righe o colonne di un DataFrame (molto simile ad un NumPy array),
- il DataFrame, una sorta di tabella, strutturata su colonne dove i dati di ciascuna unità di osservazione sono distribuiti per righe.

Lo scopo di questo capitolo è iniziare a prendere confidenza con i DataFrame ed imparare a manipolare i dati al loro interno. Per un approfondimento, consiglio il capitolo 10 di [Python for Data Analysis, 3E](#).

Iniziamo a caricare le librerie necessarie.

```
import pandas as pd
```

[Skip to main content](#)

```
import statistics as st
```

5.1. Series

In Pandas, una `Series` è un array unidimensionale composto da una sequenza di valori omogenei, simile ad un `ndarray`, accompagnato da un array di etichette chiamato “index”. A differenza degli indici degli array Numpy, che sono sempre interi e partono da zero, gli oggetti `Series` supportano etichette personalizzate che possono essere, ad esempio, delle stringhe. Inoltre, gli oggetti `Series` possono contenere dati mancanti che vengono ignorati da molte delle operazioni della classe.

Il modo più semplice di creare un oggetto `Series` è di convertire una lista. Per esempio:

```
grades = pd.Series([27, 30, 24, 18, 22, 20, 29])
```

È possibile ottenere la rappresentazione dell’`array` dell’oggetto e dell’indice dell’oggetto `Series` tramite i suoi attributi `array` e `index`, rispettivamente.

```
grades.array
```

```
<PandasArray>  
[27, 30, 24, 18, 22, 20, 29]  
Length: 7, dtype: int64
```

```
grades.index
```

```
RangeIndex(start=0, stop=7, step=1)
```

Oppure, possiamo semplicemente stampare i contenuti dell’oggetto `Series` direttamente:

```
print(grades)
```

```
0    27  
1    30  
2    24  
3    18  
4    22
```

[Skip to main content](#)

```
6      29
dtype: int64
```

Per accedere agli elementi di un oggetto `Series` si usano le parentesi quadre contenenti un indice:

```
grades[0]
```

```
27
```

```
grades[0:3]
```

```
0      27
1      30
2      24
dtype: int64
```

È possibile filtrare gli elementi di un oggetto `Series` con un array booleano:

```
grades > 24
```

```
0      True
1      True
2     False
3     False
4     False
5     False
6      True
dtype: bool
```

```
grades[grades > 24]
```

```
0      27
1      30
6      29
dtype: int64
```

È possibile manipolare gli elementi di un oggetto `Series` con le normali operazioni aritmetiche:

[Skip to main content](#)

```
grades / 10
```

```
0    2.7
1    3.0
2    2.4
3    1.8
4    2.2
5    2.0
6    2.9
dtype: float64
```

```
np.sqrt(grades)
```

```
0    5.196152
1    5.477226
2    4.898979
3    4.242641
4    4.690416
5    4.472136
6    5.385165
dtype: float64
```

Gli oggetti `Series` hanno diversi metodi per svolgere varie operazioni, per esempio per ricavare alcune statistiche descrittive:

```
[grades.count(), grades.mean(), grades.min(), grades.max(), grades.std(), gr
```

```
[7, 24.285714285714285, 18, 30, 4.572172558506722, 170]
```

Molto utile è il metodo `.describe()`:

```
grades.describe()
```

```
count      7.000000
mean      24.285714
std        4.572173
min       18.000000
25%       21.000000
50%       24.000000
75%       28.000000
max       30.000000
dtype: float64
```

[Skip to main content](#)

5.2. DataFrame

Un `pandas.DataFrame` è composto da righe e colonne. Ogni colonna di un dataframe è un oggetto `pandas.Series`: quindi, un dataframe è una collezione di serie. A differenza di un array NumPy, un dataframe può combinare più tipi di dati, come numeri e testo, ma i dati in ogni colonna sono dello stesso tipo.

Esistono molti modi per costruire un DataFrame. Il più semplice è quello di utilizzare un dizionario che include una o più liste o array Numpy di uguale lunghezza. Per esempio:

```
data = {
    "name": [
        "Maria",
        "Anna",
        "Francesco",
        "Cristina",
        "Gianni",
        "Gabriella",
        "Stefano",
    ],
    "sex": ["f", "f", "m", "f", "m", "f", "m"],
    "group": ["a", "b", "a", "b", "b", "c", "a"],
    "x": [1, 2, 3, 4, 5, 6, 7],
    "y": [8, 9, 10, 11, 12, 13, 14],
    "z": [15, 16, 17, 18, 19, 20, 21],
}
frame = pd.DataFrame(data)
frame
```

	name	sex	group	x	y	z
0	Maria	f	a	1	8	15
1	Anna	f	b	2	9	16
2	Francesco	m	a	3	10	17
3	Cristina	f	b	4	11	18
4	Gianni	m	b	5	12	19
5	Gabriella	f	c	6	13	20
6	Stefano	m	a	7	14	21

Oppure possiamo procedere nel modo seguente:

```
lst1 = [1, 2, 3, 4, 5, 6, 7]
lst2 = [8, 9, 10, 11, 12, 13, 14]
lst3 = [15, 16, 17, 18, 19, 20, 21]
```

[Skip to main content](#)

```

lst5 = ["f", "f", "m", "f", "m", "f", "m"]
lst6 = ["Maria", "Anna", "Francesco", "Cristina", "Gianni", "Gabriella", "Stefano"]

df = pd.DataFrame()

df["x"] = lst1
df["y"] = lst2
df["z"] = lst3
df["group"] = lst4
df["sex"] = lst5
df["name"] = lst6

df

```

	x	y	z	group	sex	name
0	1	8	14.4	a	f	Maria
1	2	9	15.1	b	f	Anna
2	3	10	16.7	a	m	Francesco
3	4	11	17.3	b	f	Cristina
4	5	12	18.9	b	m	Gianni
5	6	13	19.3	c	f	Gabriella
6	7	14	20.2	a	m	Stefano

Molto spesso un DataFrame viene creato dal caricamento di dati da file.

5.3. Lettura di dati da file

Di solito la quantità di dati da analizzare è tale che non è pensabile di poterli immettere manualmente in una o più liste. Normalmente i dati sono memorizzati su un file ed è necessario importarli. La lettura (importazione) dei file è il primo fondamentale passo nel processo più generale di analisi dei dati.

In un primo esempio, importiamo i dati da un repository remoto.

```

url = "https://raw.githubusercontent.com/pandas-dev/pandas/master/doc/data/titanic.csv"
titanic = pd.read_csv(url, index_col='Name')

```

È possibile usare il metodo `.head()` per visualizzare le prime cinque righe.

[Skip to main content](#)

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket
Name								
Braund, Mr. Owen Harris	1	0	3	male	22.0	1	0	A/5 211
Cumings, Mrs. John Bradley (Florence Briggs Thayer)	2	1	1	female	38.0	1	0	PC 175
Heikkinen, Miss. Laina	3	1	3	female	26.0	0	0	STON/C 31012
Futrelle, Mrs. Jacques Heath (Lily May Peel)	4	1	1	female	35.0	1	0	1138
Allen, Mr. William Henry	5	0	3	male	35.0	0	0	3734

Le statistiche descrittive per ciascuna colonna si ottengono con il metodo `describe`.

```
titanic.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parc
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.38159
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.80605
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000

In questo modo possiamo ottenere informazioni sui nomi dei passeggeri, la sopravvivenza

[Skip to main content](#)

media è di 29,7 anni, il prezzo massimo del biglietto è di 512 USD, il 38% dei passeggeri è sopravvissuto, ecc.

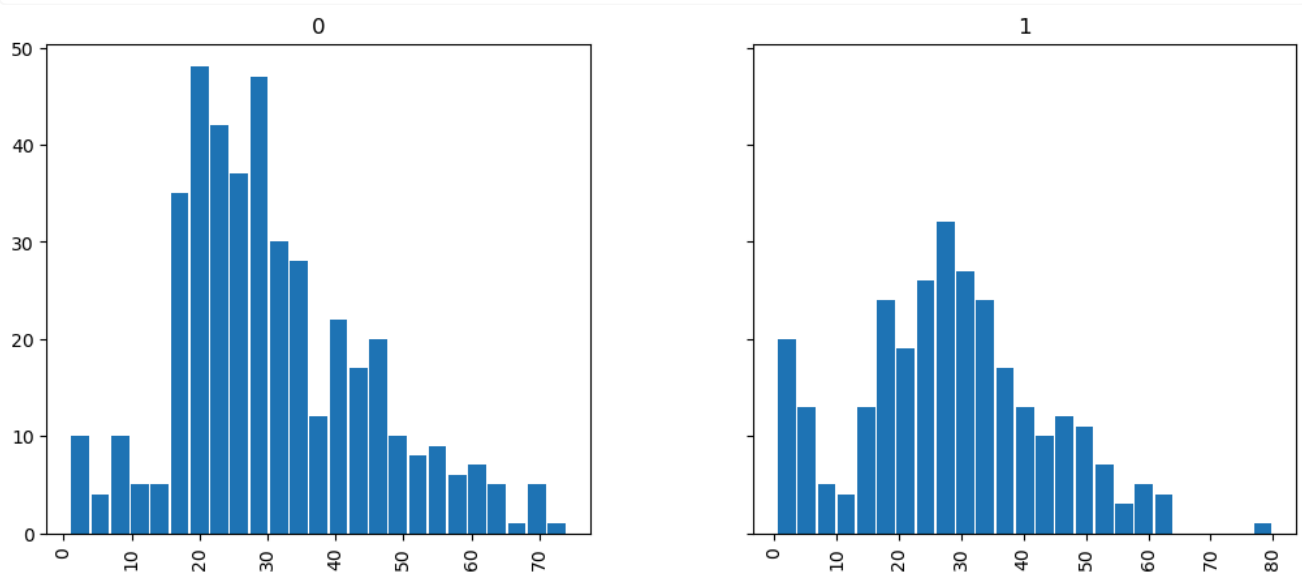
Supponiamo di essere interessati alla probabilità di sopravvivenza per diverse fasce d'età. Con due righe di codice, possiamo trovare l'età media di coloro che sono sopravvissuti o non sono sopravvissuti e generare gli istogrammi corrispondenti della distribuzione dell'età:

```
print(titanic.groupby("Survived")["Age"].mean())
```

```
Survived
0      30.626179
1      28.343690
Name: Age, dtype: float64
```

```
titanic.hist(
    column="Age",
    by="Survived",
    bins=25,
    figsize=(12, 5),
    layout=(1, 2),
    zorder=2,
    sharey=True,
    rwidth=0.9,
)
```

```
array([<Axes: title={'center': '0'}>, <Axes: title={'center': '1'}>],
      dtype=object)
```



È chiaro che i dataframes di Pandas ci permettono di condurre analisi avanzate con pochi comandi, ma acquisire familiarità con la sintassi corretta richiede un po' di tempo.

[Skip to main content](#)

Per fare un secondo esempio, importo i dati dal file `penguins.csv` situato nella directory “data” del mio computer. I dati relativi ai pinguini di Palmer sono resi disponibili da [Kristen Gorman](#) e dalla [Palmer station, Antarctica LTER](#). La seguente cella legge il contenuto del file `penguins.csv` e lo inserisce nell’oggetto `df` utilizzando la funzione `read_csv()` di Pandas.

```
df = pd.read_csv("data/penguins.csv")
```

Per il DataFrame `df` il significato delle colonne è il seguente:

- `species`: a factor denoting penguin type (Adélie, Chinstrap and Gentoo)
- `island`: a factor denoting island in Palmer Archipelago, Antarctica (Biscoe, Dream or Torgersen)
- `bill_length_mm`: a number denoting bill length (millimeters)
- `bill_depth_mm`: a number denoting bill depth (millimeters)
- `flipper_length_mm`: an integer denoting flipper length (millimeters)
- `body_mass_g`: an integer denoting body mass (grams)
- `sex`: a factor denoting sexuality (female, male)
- `year`: the year of the study

Usiamo il metodo `.head()` per visualizzare le prime cinque righe.

```
df.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
3	Adelie	Torgersen	NaN	NaN	NaN	
4	Adelie	Torgersen	36.7	19.3	193.0	

A volte potrebbero esserci dati estranei alla fine del file, quindi è importante anche controllare le ultime righe:

```
df.tail()
```

[Skip to main content](#)

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
339	Chinstrap	Dream	55.8	19.8	207.0	
340	Chinstrap	Dream	43.5	18.1	202.0	
341	Chinstrap	Dream	49.6	18.2	193.0	
342	Chinstrap	Dream	50.8	19.0	210.0	
343	Chinstrap	Dream	50.2	18.7	198.0	

Le istruzioni seguenti ritornano le prime e le ultime tre righe del DataFrame.

```
display(df.head(3))
display(df.tail(3))
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
341	Chinstrap	Dream	49.6	18.2	193.0	
342	Chinstrap	Dream	50.8	19.0	210.0	
343	Chinstrap	Dream	50.2	18.7	198.0	

Tip

Un breve tutorial in formato video è disponibile tramite il seguente [collegamento](#), il quale illustra come effettuare la lettura dei dati da un file esterno in Visual Studio Code.

L'attributo `.dtypes` restituisce il tipo dei dati:

```
df.dtypes
```

```
species          object
island           object
bill_length_mm  float64
bill_depth_mm   float64
flipper_length_mm  float64
body            float64
```

[Skip to main content](#)

```
flipper_length_mm    float64
body_mass_g          float64
sex                  object
year                 int64
dtype: object
```

Gli attributi più comunemente usati sono elencati di seguito:

Attributo Ritorna

<code>dtypes</code>	Il tipo di dati in ogni colonna
<code>shape</code>	Una tupla con le dimensioni del DataFrame object (numero di righe, numero di colonne)
<code>index</code>	L'oggetto <code>Index</code> lungo le righe del DataFrame
<code>columns</code>	Il nome delle colonne
<code>values</code>	I dati contenuti nel DataFrame
<code>empty</code>	Check if the DataFrame object is empty

Per esempio, l'istruzione della cella seguente restituisce l'elenco con i nomi delle colonne del DataFrame `df`:

```
df.columns
```

```
Index(['species', 'island', 'bill_length_mm', 'bill_depth_mm',
       'flipper_length_mm', 'body_mass_g', 'sex', 'year'],
      dtype='object')
```

L'attributo `.shape` ritorna il numero di righe e di colonne del DataFrame. Nel caso presente, ci sono 344 righe e 8 colonne.

```
df.shape
```

```
(344, 8)
```

Come abbiamo già visto in precedenza, un sommario dei dati si ottiene con il metodo

`.describe()`:

[Skip to main content](#)

```
df.describe()
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	
count	342.000000	342.000000	342.000000	342.000000	344.0
mean	43.921930	17.151170	200.915205	4201.754386	2008.0
std	5.459584	1.974793	14.061714	801.954536	0.8
min	32.100000	13.100000	172.000000	2700.000000	2007.0
25%	39.225000	15.600000	190.000000	3550.000000	2007.0
50%	44.450000	17.300000	197.000000	4050.000000	2008.0
75%	48.500000	18.700000	213.000000	4750.000000	2009.0
max	59.600000	21.500000	231.000000	6300.000000	2009.0

Una descrizione del DataFrame si ottiene con il metodo `.info()`.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   species                344 non-null   object
1   island                 344 non-null   object
2   bill_length_mm         342 non-null   float64
3   bill_depth_mm          342 non-null   float64
4   flipper_length_mm      342 non-null   float64
5   body_mass_g            342 non-null   float64
6   sex                    333 non-null   object
7   year                   344 non-null   int64
dtypes: float64(4), int64(1), object(3)
memory usage: 21.6+ KB
```

⚠ Warning

Si noti che, alle volte, abbiamo utilizzato la sintassi `df.word` e talvolta la sintassi `df.word()`. Tecnicamente, la classe Pandas Dataframe ha sia attributi che metodi. Gli attributi sono `.word`, mentre i metodi sono `.word()` o `.word(arg1, arg2, ecc.)`. Per sapere se qualcosa è un metodo o un attributo è necessario leggere la documentazione.

[Skip to main content](#)

Abbiamo visto in precedenza come possiamo leggere i dati in un dataframe utilizzando la funzione `read_csv()`. Pandas comprende anche molti altri formati, ad esempio utilizzando le funzioni `read_excel()`, `read_hdf()`, `read_json()`, ecc. (e i corrispondenti metodi per scrivere su file: `to_csv()`, `to_excel()`, `to_hdf()`, `to_json()`, ecc.).

5.4. Gestione dei dati mancanti

Nell'output di `.info()` troviamo la colonna “Non-Null Count”, ovvero il numero di dati non mancanti per ciascuna colonna del DataFrame. Da questo si nota che le colonne del DataFrame `df` contengono alcuni dati mancanti. La gestione dei dati mancanti è un argomento complesso. Per ora ci limitiamo ad escludere tutte le righe che, in qualche colonna, contengono dei dati mancanti.

Ottengo il numero di dati per ciascuna colonna del DataFrame:

```
df.isnull().sum()
```

```
species          0
island           0
bill_length_mm   2
bill_depth_mm    2
flipper_length_mm 2
body_mass_g      2
sex              11
year             0
dtype: int64
```

Rimuovo i dati mancanti con il metodo `.dropna()`. L'argomento `inplace=True` specifica il DataFrame viene trasformato in maniera permanente.

```
df.dropna(inplace=True)
```

Verifico che i dati mancanti siano stati rimossi.

```
df.shape
```

```
(333, 8)
```

[Skip to main content](#)

5.5. Rinominare le colonne

È possibile rinominare tutte le colonne passando al metodo `.rename()` un dizionario che specifica quali colonne devono essere mappate a cosa. Nella cella seguente faccio prima una copia del DataFrame con il metodo `copy()` e poi rinomino `sex` che diventa `gender` e `year` che diventa `year_of_the_study`:

```
df1 = df.copy()

# rename(columns={"OLD_NAME": "NEW_NAME"})
df1.rename(columns={"sex": "gender", "year": "year_of_the_study"}, inplace=True)
df1.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
4	Adelie	Torgersen	36.7	19.3	193.0	
5	Adelie	Torgersen	39.3	20.6	190.0	

⚠ Warning

Si noti che in Python valgono le seguenti regole.

- Il nome di una variabile deve iniziare con una lettera o con il trattino basso (*underscore*) `_`.
- Il nome di una variabile non può iniziare con un numero.
- Un nome di variabile può contenere solo caratteri alfanumerici e il trattino basso (A-z, 0-9 e `_`).
- I nomi delle variabili fanno distinzione tra maiuscole e minuscole (`age`, `Age` e `AGE` sono tre variabili diverse).

Gli spazi non sono consentiti nel nome delle variabili: come separatore usate il trattino basso.

[Skip to main content](#)

5.6. Estrarre i dati dal DataFrame

Una parte cruciale del lavoro con i DataFrame è l'estrazione di sottoinsiemi di dati: vogliamo trovare le righe che soddisfano un determinato insieme di criteri, vogliamo isolare le colonne/righe di interesse, ecc. Per rispondere alle domande di interesse dell'analisi dei dati, molto spesso è necessario selezionare un sottoinsieme del DataFrame.

5.6.1. Colonne

È possibile estrarre una colonna da un DataFrame usando una notazione simile a quella che si usa per il dizionario (`DataFrame['word']`) o utilizzando la notazione

`DataFrame.word`. Per esempio:

```
df["bill_length_mm"]
```

```
0      39.1
1      39.5
2      40.3
4      36.7
5      39.3
...
339    55.8
340    43.5
341    49.6
342    50.8
343    50.2
Name: bill_length_mm, Length: 333, dtype: float64
```

```
df.bill_length_mm
```

```
0      39.1
1      39.5
2      40.3
4      36.7
5      39.3
...
339    55.8
340    43.5
341    49.6
342    50.8
343    50.2
Name: bill_length_mm, Length: 333, dtype: float64
```

[Skip to main content](#)

ottenere le prime 3 righe del DataFrame `df` nel modo seguente:

```
df[0:3]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	

Si noti che in Python una sequenza è determinata dal valore iniziale e quello finale *ma si interrompe ad $n-1$* . Pertanto, per selezionare una singola riga (per esempio, la prima) dobbiamo procedere nel modo seguente:

```
df[0:1]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	

5.6.3. Indicizzazione, selezione e filtraggio

Poiché l'oggetto DataFrame è bidimensionale, è possibile selezionare un sottoinsieme di righe e colonne utilizzando le etichette degli assi (`loc`) o gli indici delle righe (`iloc`).

Per esempio, usando l'attributo `iloc` posso selezionare la prima riga del DataFrame:

```
df.iloc[0]
```

```
species          Adelie
island           Torgersen
bill_length_mm    39.1
bill_depth_mm     18.7
flipper_length_mm 181.0
body_mass_g       3750.0
sex              male
year             2007
Name: 0, dtype: object
```

[Skip to main content](#)

```
df.iloc[0:3]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	

L'attributo `loc` consente di selezionare simultaneamente righe e colonne per "nome". Il "nome" delle righe è l'indice di riga. Per esempio, visualizzo il quinto valore della colonna

`body_mass_g`:

```
df.loc[4, "body_mass_g"]
```

3450.0

oppure, il quinto valore delle colonne `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`:

```
df.loc[4, ["bill_length_mm", "bill_depth_mm", "flipper_length_mm"]]
```

```
bill_length_mm      36.7
bill_depth_mm       19.3
flipper_length_mm   193.0
Name: 4, dtype: object
```

Visualizzo ora le prime tre righe sulle tre colonne precedenti. Si noti l'uso di `:` per definire un intervallo di valori sull'indice di riga.

```
df.loc[0:2, ["bill_length_mm", "bill_depth_mm", "flipper_length_mm"]]
```

	bill_length_mm	bill_depth_mm	flipper_length_mm
0	39.1	18.7	181.0
1	39.5	17.4	186.0
2	40.3	18.0	195.0

[Skip to main content](#)

Una piccola variante della sintassi precedente si rivela molto utile. Qui, il segno di due punti (`:`) significa “tutte le righe”:

```
keep_cols = ["bill_length_mm", "bill_depth_mm", "flipper_length_mm"]
print(df.loc[:, keep_cols])
```

	bill_length_mm	bill_depth_mm	flipper_length_mm
0	39.1	18.7	181.0
1	39.5	17.4	186.0
2	40.3	18.0	195.0
4	36.7	19.3	193.0
5	39.3	20.6	190.0
...
339	55.8	19.8	207.0
340	43.5	18.1	202.0
341	49.6	18.2	193.0
342	50.8	19.0	210.0
343	50.2	18.7	198.0

[333 rows x 3 columns]

5.6.4. Filtrare righe in maniera condizionale

In precedenza abbiamo utilizzato la selezione delle righe in un DataFrame in base alla loro posizione. Tuttavia, è più comune selezionare le righe del DataFrame utilizzando una condizione logica, cioè tramite l'indicizzazione booleana.

Iniziamo con un esempio relativo ad una condizione specificata sui valori di una sola colonna. Quando applichiamo un operatore logico come `>`, `<`, `==`, `!=` ai valori di una colonna del DataFrame, il risultato è una sequenza di valori booleani (`True`, `False`), uno per ogni riga nel DataFrame, i quali indicano se, per quella riga, la condizione è vera o falsa. Ad esempio:

```
df["island"] == "Torgersen"
```

0	True
1	True
2	True
4	True
5	True
...	...
339	False
340	False
...	...

[Skip to main content](#)

```
343      False
      Name: island, Length: 333, dtype: bool
```

Utilizzando i valori booleani che sono stati ottenuti in questo modo è possibile filtrare le righe del DataFrame, ovvero, ottenere un nuovo DataFrame nel quale la condizione logica specificata è vera su tutte le righe. Per esempio, nella cella seguente selezioniamo solo le osservazioni relative all'isola Torgersen, ovvero tutte le righe del DataFrame nelle quali la colonna `island` assume il valore `Torgersen`.

```
only_torgersen = df[df["island"] == "Torgersen"]
only_torgersen.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
4	Adelie	Torgersen	36.7	19.3	193.0	
5	Adelie	Torgersen	39.3	20.6	190.0	

In maniera equivalente, possiamo scrivere:

```
only_torgersen = df.loc[df["island"] == "Torgersen"]
only_torgersen.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
4	Adelie	Torgersen	36.7	19.3	193.0	
5	Adelie	Torgersen	39.3	20.6	190.0	

È possibile combinare più condizioni logiche usando gli operatori `&` (e), `|` (oppure). Si presti attenzione all'uso delle parentesi.

```
df.loc[(df["island"] == "Torgersen") & (df["sex"] == "female")].head()
```

[Skip to main content](#)

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
4	Adelie	Torgersen	36.7	19.3	193.0	
6	Adelie	Torgersen	38.9	17.8	181.0	
12	Adelie	Torgersen	41.1	17.6	182.0	

5.6.5. Metodo `.query`

È anche possibile filtrare le righe del DataFrame usando il metodo `query()`. Ci sono diversi modi per generare sottoinsiemi con Pandas. I metodi `loc` e `iloc` consentono di recuperare sottoinsiemi in base alle etichette di riga e colonna o all'indice intero delle righe e delle colonne. E Pandas ha una notazione a parentesi quadre che consente di utilizzare condizioni logiche per recuperare righe di dati specifiche. Ma la sintassi di questi metodi non è la più trasparente. Inoltre, tali metodi sono difficili da usare insieme ad altri metodi di manipolazione dei dati in modo organico.

Il metodo `.query` di Pandas cerca di risolvere questi problemi. Il metodo `.query` consente di “interrogare” un DataFrame e recuperare sottoinsiemi basati su condizioni logiche. La sintassi è un po' più snella rispetto alla notazione a parentesi quadre di Pandas. Inoltre, il metodo `.query` può essere utilizzato con altri metodi di Pandas in modo snello e semplice, rendendo la manipolazione dei dati maggiormente fluida e diretta.

La sintassi è la seguente:

```
your_data_frame.query(expression, inplace = False)
```

L'espressione utilizzata nella query è una sorta di espressione logica che descrive quali righe restituire in output. Se l'espressione è vera per una particolare riga, la riga verrà inclusa nell'output. Se l'espressione è falsa per una particolare riga, quella riga verrà esclusa dall'output.

Il parametro `inplace` consente di specificare se si desidera modificare direttamente il DataFrame con cui si sta lavorando.

Per esempio:

[Skip to main content](#)

```
eval_string = "island == 'Torgersen' & sex == 'female' & year != 2009"  
df.query(eval_string)[["bill_depth_mm", "flipper_length_mm"]]
```

	bill_depth_mm	flipper_length_mm
1	17.4	186.0
2	18.0	195.0
4	19.3	193.0
6	17.8	181.0
12	17.6	182.0
15	17.8	185.0
16	19.0	195.0
18	18.4	184.0
68	16.6	190.0
70	19.0	190.0
72	17.2	196.0
74	17.5	190.0
76	16.8	191.0
78	16.1	187.0
80	17.2	189.0
82	18.8	187.0

Un altro esempio usa la keyword `in` per selezionare solo le righe relative alle due isole specificate.

```
eval_string = "island in ['Torgersen', 'Dream']"  
df.query(eval_string)[["bill_depth_mm", "flipper_length_mm"]]
```

[Skip to main content](#)

	bill_depth_mm	flipper_length_mm
0	18.7	181.0
1	17.4	186.0
2	18.0	195.0
4	19.3	193.0
5	20.6	190.0
...
339	19.8	207.0
340	18.1	202.0
341	18.2	193.0
342	19.0	210.0
343	18.7	198.0

170 rows × 2 columns

Il metodo `query()` può anche essere utilizzato per selezionare le righe di un DataFrame in base alle relazioni tra le colonne. Ad esempio,

```
df.query("bill_length_mm > 3*bill_depth_mm")[["bill_depth_mm", "flipper_length_mm"]]
```

	bill_depth_mm	flipper_length_mm
152	13.2	211.0
153	16.3	230.0
154	14.1	210.0
155	15.2	218.0
156	14.5	215.0
...
272	14.3	215.0
273	15.7	222.0
274	14.8	212.0
275	16.1	213.0
...

[Skip to main content](#)

106 rows × 2 columns

È anche possibile fare riferimento a variabili non contenute nel DataFrame usando il carattere `@`.

```
outside_var = 21
df.query("bill_depth_mm > @outside_var")[["bill_depth_mm", "flipper_length_mm"]]
```

	bill_depth_mm	flipper_length_mm
13	21.2	191.0
14	21.1	198.0
19	21.5	194.0
35	21.1	196.0
49	21.2	191.0
61	21.1	195.0

5.7. Selezione casuale di un sottoinsieme di righe

Il metodo `sample()` viene usato per ottenere un sottoinsieme casuale di righe del DataFrame. L'argomento `replace=False` indica l'estrazione senza rimessa (default); se specifichiamo `replace=True` otteniamo un'estrazione con rimessa. L'argomento `n` specifica il numero di righe che vogliamo ottenere. Ad esempio

```
df_sample = df.sample(4)
df_sample
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
142	Adelie	Dream	32.1	15.5	188.0	
259	Gentoo	Biscoe	53.4	15.8	219.0	
20	Adelie	Biscoe	37.8	18.3	174.0	
329	Chinstrap	Dream	50.7	19.7	203.0	

[Skip to main content](#)

df_sample

	bill_length_mm	bill_depth_mm
31	37.2	18.1
306	40.9	16.6
302	50.5	18.4
140	40.2	17.1

5.8. Selezione di colonne

Il metodo `drop()` prende in input una lista con i nomi di colonne che vogliamo escludere dal DataFrame e può essere usato per creare un nuovo DataFrame o per sovrascrivere quello di partenza. È possibile usare le espressioni regolari (*regex*) per semplificare la ricerca dei nomi delle colonne.

Tip

In *regex* il simbolo `$` significa “la stringa finisce con”; il simbolo `^` significa “la stringa inizia con”. L'espressione `regex` può contenere (senza spazi) il simbolo `|` che significa “oppure”.

Nel codice della cella seguente, alla funzione `.columns.str.contains()` viene passata l'espressione regolare `mm$|year` che significa: tutte le stringhe (in questo caso, nomi di colonne) che finiscono con `mm` oppure la stringa (nome di colonna) `year`.

```
mask = df.columns.str.contains("mm$|year", regex=True)
columns_to_drop = df.columns[mask]
columns_to_drop
```

```
Index(['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'year'], dtype=object)
```

```
df_new = df.drop(columns=columns_to_drop)
df_new.head()
```

[Skip to main content](#)

	species	island	body_mass_g	sex
0	Adelie	Torgersen	3750.0	male
1	Adelie	Torgersen	3800.0	female
2	Adelie	Torgersen	3250.0	female
4	Adelie	Torgersen	3450.0	female
5	Adelie	Torgersen	3650.0	male

In un altro esempio, creiamo l'elenco delle colonne che iniziano con la lettera “b”, insieme a `year` e `sex`.

```
mask = df.columns.str.contains("^b|year|sex", regex=True)
columns_to_drop = df.columns[mask]
columns_to_drop
```

```
Index(['bill_length_mm', 'bill_depth_mm', 'body_mass_g', 'sex', 'year'], dtype=
```

Oppure l'elenco delle colonne che contengono il patten “length”.

```
mask = df.columns.str.contains("length")
columns_to_drop = df.columns[mask]
columns_to_drop
```

```
Index(['bill_length_mm', 'flipper_length_mm'], dtype='object')
```

5.9. Creare nuove colonne

Per ciascuna riga, calcoliamo

- `bill_length_mm - bill_depth_mm`
- `bill_length_mm / (body_mass_g / 1000)`

Per ottenere questo risultato possiamo usare una *lambda function*.

```
df = df.assign(
    bill_difference=lambda x: x.bill_length_mm - x.bill_depth_mm,
    bill_ratio=lambda x: x.bill_length_mm / (x.body_mass_g / 1000),
```

[Skip to main content](#)

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

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
4	Adelie	Torgersen	36.7	19.3	193.0	
5	Adelie	Torgersen	39.3	20.6	190.0	

In maniera più semplice possiamo procedere nel modo seguente:

```
df["bill_ratio2"] = df["bill_length_mm"] / (df["body_mass_g"] / 1000)
df.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
4	Adelie	Torgersen	36.7	19.3	193.0	
5	Adelie	Torgersen	39.3	20.6	190.0	

Un'utile funzionalità è quella che consente di aggiungere una colonna ad un DataFrame (o di modificare una colonna già esistente) sulla base di una condizione True/False. Questo risultato può essere raggiunto usando `np.where()`, con la seguente sintassi:

```
np.where(condition, value if condition is true, value if condition is false)
```

Supponiamo di avere un DataFrame `df` con due colonne, `A` e `B`, e vogliamo creare una nuova colonna `C` che contenga il valore di `A` quando questo è maggiore di 0, e il valore di `B` altrimenti. Possiamo utilizzare la funzione `where()` per ottenere ciò come segue:

```
# Creiamo un DataFrame di esempio
df = pd.DataFrame({'A': [-1, 2, 3, -4], 'B': [5, 6, 0, 8]})

# Creiamo una nuova colonna 'C' usando la funzione where()
df['C'] = df['A'].where(df['A'] > 0, df['B'])
```

[Skip to main content](#)

```
print(df)
```

```
   A  B  C
0 -1  5  5
1  2  6  2
2  3  0  3
3 -4  8  8
```

5.10. Formato long e wide

Nella data analysis, i termini “formato long” e “formato wide” sono usati per descrivere la struttura di un set di dati. Il formato wide (in inglese “wide format”) rappresenta una struttura di dati in cui ogni riga rappresenta una singola osservazione e ogni variabile è rappresentata da più colonne. Un esempio è il seguente, nel quale per ciascun partecipante, identificato da

`Name` e `ID` abbiamo i punteggi di un ipotetico test per 6 anni consecutivi.

```
scores = {
    "Name": ["Maria", "Carlo", "Giovanna", "Irene"],
    "ID": [1, 2, 3, 4],
    "2017": [85, 87, 89, 91],
    "2018": [96, 98, 100, 102],
    "2019": [100, 102, 106, 106],
    "2020": [89, 95, 98, 100],
    "2021": [94, 96, 98, 100],
    "2022": [100, 104, 104, 107],
}

wide_data = pd.DataFrame(scores)
wide_data
```

	Name	ID	2017	2018	2019	2020	2021	2022
0	Maria	1	85	96	100	89	94	100
1	Carlo	2	87	98	102	95	96	104
2	Giovanna	3	89	100	106	98	98	104
3	Irene	4	91	102	106	100	100	107

Il formato long (in inglese “long format”) rappresenta una struttura di dati in cui ogni riga rappresenta una singola osservazione e ogni colonna rappresenta una singola variabile. Questo formato è quello che viene richiesto per molte analisi statistiche. In Pandas è possibile usare la funzione `melt` per trasformare i dati dal formato wide al formato long. Un

[Skip to main content](#)

partecipante, ma i dati del test, che prima erano distribuiti su sei colonne, ora sono presenti in una singola colonna. Al DataFrame, inoltre, è stata aggiunta una colonna che riporta l'anno.

```
long_data = wide_data.melt(  
    id_vars=["Name", "ID"], var_name="Year", value_name="Score"  
)  
long_data
```

[Skip to main content](#)

	Name	ID	Year	Score
0	Maria	1	2017	85
1	Carlo	2	2017	87
2	Giovanna	3	2017	89
3	Irene	4	2017	91
4	Maria	1	2018	96
5	Carlo	2	2018	98
6	Giovanna	3	2018	100
7	Irene	4	2018	102
8	Maria	1	2019	100
9	Carlo	2	2019	102
10	Giovanna	3	2019	106
11	Irene	4	2019	106
12	Maria	1	2020	89
13	Carlo	2	2020	95
14	Giovanna	3	2020	98
15	Irene	4	2020	100
16	Maria	1	2021	94
17	Carlo	2	2021	96
18	Giovanna	3	2021	98
19	Irene	4	2021	100
20	Maria	1	2022	100
21	Carlo	2	2022	104
22	Giovanna	3	2022	104
23	Irene	4	2022	107

Per migliorare la leggibilità dei dati, è possibile riordinare le righe del set di dati utilizzando la funzione `sort_values`. In questo modo, le informazioni saranno presentate in un ordine specifico, che può rendere più facile la lettura dei dati.

```
long_data.sort_values(by=["ID", "Year"])
```

[Skip to main content](#)

	Name	ID	Year	Score
0	Maria	1	2017	85
4	Maria	1	2018	96
8	Maria	1	2019	100
12	Maria	1	2020	89
16	Maria	1	2021	94
20	Maria	1	2022	100
1	Carlo	2	2017	87
5	Carlo	2	2018	98
9	Carlo	2	2019	102
13	Carlo	2	2020	95
17	Carlo	2	2021	96
21	Carlo	2	2022	104
2	Giovanna	3	2017	89
6	Giovanna	3	2018	100
10	Giovanna	3	2019	106
14	Giovanna	3	2020	98
18	Giovanna	3	2021	98
22	Giovanna	3	2022	104
3	Irene	4	2017	91
7	Irene	4	2018	102
11	Irene	4	2019	106
15	Irene	4	2020	100
19	Irene	4	2021	100
23	Irene	4	2022	107

5.11. Watermark

```
%load_ext watermark
%watermark -n -u -v -iv -w
```

[Skip to main content](#)

Last updated: Sat Jun 17 2023

Python implementation: CPython

Python version : 3.11.3

IPython version : 8.12.0

pandas: 1.5.3

numpy : 1.24.3

Watermark: 2.3.1