

June 7, 2023

## Contents

<b>1</b>	<b>Setup</b>	<b>1</b>
<b>2</b>	<b>Combinatoria</b>	<b>1</b>
<b>3</b>	<b>Generazione di numeri casuali con numpy</b>	<b>2</b>
<b>4</b>	<b>Variabili casuali in scipy</b>	<b>2</b>
4.1	Funzioni e parametri principali . . . . .	2
4.2	Uso interattivo rapido . . . . .	4
4.3	Freezing di una distribuzione . . . . .	4
4.4	Generazione di numeri casuali con <code>numpy</code> e <code>scipy</code> . . . . .	4

## 1 Setup

Importiamo le librerie qui usate

```
>>> import math
>>> import itertools
>>> import scipy
>>> import numpy as np

>>> from scipy import stats
```

## 2 Combinatoria

**Fattoriale**

```
>>> math.factorial(4)
24
```

**Coefficiente binomiale**

```
>>> n = 4
>>> k = 2
>>> scipy.special.comb(n, k)
6.0
```

### Permutazioni di un vettore

```
>>> data = [1,2,3]
>>> list(itertools.permutations(data))
[(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]
```

### Combinazioni di elementi di un vettore

```
>>> list(itertools.combinations(data, 2))
[(1, 2), (1, 3), (2, 3)]
```

### Prodotto cartesiano

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> list(itertools.product(a,b)) # eventualmente posto in np.array
[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]
```

## 3 Generazione di numeri casuali con numpy

Il modulo `numpy.random` fornisce supporto base per la generazione di numeri casuali.

Si crea un generatore di numeri casuali impostando il seme e poi si generano utilizzando metodi per averne dalle distribuzioni

```
>>> rng = np.random.default_rng(seed = 6315744)
>>> data = rng.standard_normal((2, 3))
>>> data
array([[ 0.02015758,  2.22358201,  0.94127564],
       [-0.75983371, -2.37769907,  0.25145986]])
```

Questo generatore è pertanto separato da altro codice che potrebbe generare la generazione di numeri casuali. I metodi di interesse sono in tabella 1; si nota che comunque le distribuzioni statistiche possibili non sono molte. In questo viene aiuto `scipy.stats` come si vedrà.

## 4 Variabili casuali in scipy

### 4.1 Funzioni e parametri principali

In `scipy.stats` le variabili casuali di maggiore interesse sono riportate in tabella 2, i relativi metodi sono riportati in tabella 3.

I parametri di maggior interesse sono `loc` (locazione) e `scale` (dispersione); alcune quantitative anche un parametro `shape`. Nel caso della normale `loc` è la media ed è impostata di default a 0, `scale` la deviazione standard ed è impostata a 1.

Metodo	Descrizione
<code>permutation</code>	Return a random permutation of a sequence, or return a permuted range
<code>shuffle</code>	Randomly permute a sequence in place
<code>uniform</code>	Draw samples from a uniform distribution
<code>integers</code>	Draw random integers from a given low-to-high range
<code>binomial</code>	Draw samples a binomial distribution
<code>standard_normal</code>	Estrazioni da una normale 0, 1
<code>normal</code>	Draw samples from a normal (Gaussian) distribution
<code>beta</code>	Draw samples from a beta distribution
<code>chisquare</code>	Draw samples from a chi-square distribution
<code>gamma</code>	Draw samples from a gamma distribution
<code>uniform</code>	Draw samples from a uniform [0, 1) distribution

Table 1: Metodi per la generazione di numeri casuali in `numpy`

Famiglia	Funzione	Famiglia	Funzione
Bernoulli	<code>bernoulli</code>	Uniforme cont.	<code>uniform</code>
Binomiale	<code>binom</code>	Esponenziale	<code>expon</code>
Geometrica	<code>geom</code>	Normale	<code>norm</code>
Binomiale neg.	<code>nbinom</code>	Gamma	<code>gamma</code>
Ipergeometrica	<code>hypergeom</code>	Chi-quadrato	??
Poisson	<code>poisson</code>	Beta	<code>beta</code>
Uniforme disc.	<code>randint</code>	T di Student	<code>t</code>
		F	??
		Logistica	??
		Lognormale	??
		Weibull	??
		Pareto	??

Table 2: Funzioni `scipy.stats.*` per variabili casuali in Python

Metodo	Descrizione
<code>rvs</code>	generazione di numeri casuali; equivalente di <code>r*</code> di R
<code>pdf, pmf</code>	probability density e mass function; equivalente di <code>d*</code> di R
<code>cdf</code>	cumulative distribution function; equivalente di <code>p*</code> di R
<code>ppf</code>	percent point function (inversa di <code>cdf</code> ); equivalente di <code>q*</code> di R
<code>sf</code>	Survival Function (1-CDF)
<code>isf</code>	Inverse Survival Function (Inverse of SF)
<code>stats</code>	media, varianza, (Fisher's) skew, or (Fisher's) kurtosis
<code>moment</code>	non-central moments of the distribution

Table 3: Metodi variabili casuali `scipy`

## 4.2 Uso interattivo rapido

Un esempio di utilizzo interattivo rapido con normale a seguire. Per facilitare, le funzioni supportano il broadcasting

```
>>> # estrazioni di casuali normali standardizzate (come rnorm(5))
>>> stats.norm.rvs(size = 5)
array([ 0.21085942, -1.20663803, -1.57588166, -0.73419262,  0.53998574])

>>> # densità: come dnorm(0.5)
>>> stats.norm.pdf(0.5)
0.3520653267642995

>>> # cdf: come pnorm(1.96)
>>> stats.norm.cdf(1.96)
0.9750021048517795

>>> # ppf: come qnorm(0.5)
>>> stats.norm.ppf(0.5)
0.0
>>> # esempio con broadcasting
>>> stats.norm.pdf([0.025, 0.5, 0.975])
array([0.39881763, 0.35206533, 0.24801872])
```

## 4.3 Freezing di una distribuzione

Spesso capita di dover lavorare più volte con distribuzioni dai parametri differenti. Onde evitare di dover reinserire i parametri in ogni chiamata possiamo effettuare il freezing di una distribuzione

```
>>> n01 = stats.norm(loc = 0, scale = 1) # mu=0, sd=1 sono i valori di default
>>> n01.rvs(size = 5)
array([-1.289172, -0.69206753, -0.63938368,  0.53365036,  0.28064051])

>>> n25 = stats.norm(loc = 2, scale = 5) # mu=2, sd=5
>>> n25.rvs(size = 5)
array([ 3.48294826,  1.81936114, -1.23729315,  1.10612277,  5.64090112])
```

## 4.4 Generazione di numeri casuali con numpy e scipy

Per fare le cose bene creiamo il generatore di numeri casuali in numpy e lo passiamo nella creazione di un oggetto variabile normale

```
>>> rng = np.random.default_rng(302132)
>>> n01 = stats.norm()
>>> n01.rvs(size = 5, random_state=rng)
array([-0.14191574,  0.42922204, -0.33735776,  0.20712603, -0.9709426 ])
>>> n01.rvs(size = 5, random_state=rng)
```

```

array([ 0.64705971, -1.05564341,  0.94984921, -0.30577826, -0.61788831])

>>> # sequenza di sopra riprodotta con un nuovo generatore
>>> # avente lo stesso seme (usando la stessa variabile, le cui caratteristiche
>>> # non ci interessa cambiare)
>>> rng2 = np.random.default_rng(302132)
>>> n01.rvs(size = 5, random_state=rng2)
array([-0.14191574,  0.42922204, -0.33735776,  0.20712603, -0.9709426 ])
>>> n01.rvs(size = 5, random_state=rng2)
array([ 0.64705971, -1.05564341,  0.94984921, -0.30577826, -0.61788831])

```