numpy.histogram2d

numpy.histogram2d(*x*, *y*, *bins=10*, *range=None*, *normed=False*, *weights=None*) [source]

Compute the bi-dimensional histogram of two data samples.

Parameters:

x : *array_like, shape (N,)*
An array containing the x coordinates of the points to be histogrammed.

y : *array_like, shape (N,)*
An array containing the y coordinates of the points to be histogrammed.

bins : *int or array_like or [int, int] or [array, array], optional*
The bin specification:

- If int, the number of bins for the two dimensions (nx=ny=bins).
- If array_like, the bin edges for the two dimensions (x_edges=y_edges=bins).
- If [int, int], the number of bins in each dimension (nx, ny = bins).
- If [array, array], the bin edges in each dimension (x_edges, y_edges = bins).
- A combination [int, array] or [array, int], where int is the number of bins and array is the bin edges.

range : *array_like, shape(2,2), optional*
The leftmost and rightmost edges of the bins along each dimension (if not specified explicitly in the *bins* parameters): `[[xmin, xmax], [ymin, ymax]]`. All values outside of this range will be considered outliers and not tallied in the histogram.

normed : *bool, optional*
If False, returns the number of samples in each bin. If True, returns the bin density `bin_count / sample_count / bin_area`.

weights : *array_like, shape(N,), optional*
An array of values `w_i` weighing each sample `(x_i, y_i)`. Weights are normalized to 1 if *normed* is True. If *normed* is False, the values of the returned histogram are equal to the sum of the weights belonging to the samples falling into each bin.

Returns:

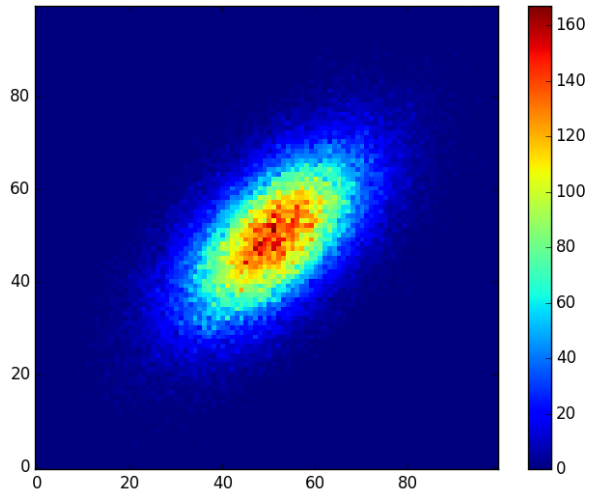
H : *ndarray, shape(nx, ny)*
The bi-dimensional histogram of samples *x* and *y*. Values in *x* are histogrammed along the first dimension and values in *y* are histogrammed along the second dimension.

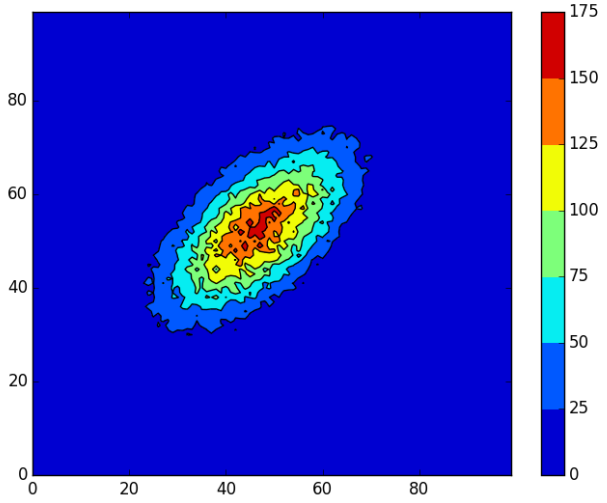
xedges : *ndarray, shape(nx,)*
The bin edges along the first dimension.

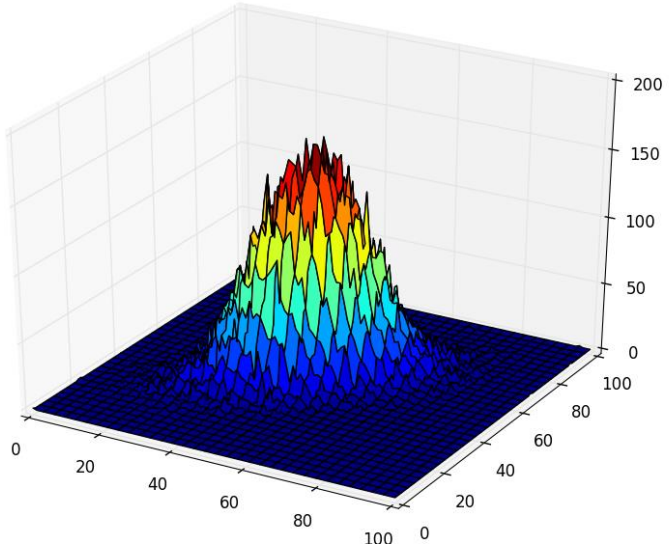
yedges : *ndarray, shape(ny,)*
The bin edges along the second dimension.

Zadanie

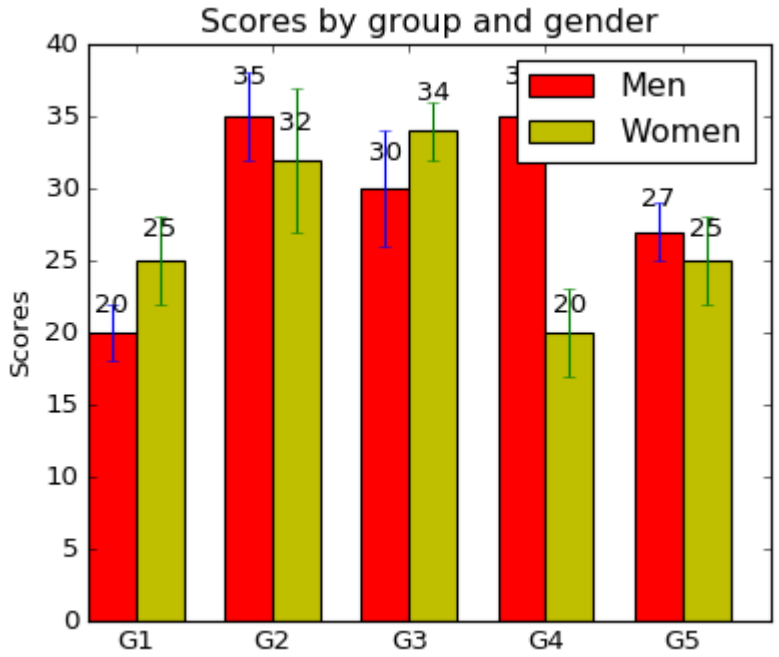
- Ściągnij pliki zawierający parametry X i Y reprezentujące jakieś pomiary z rzeczywistego świata. Stwórz histogram dwuwymiarowy i wyświetl go jako obraz, jako wykres konturowy i jako wykres trójwymiarowy. Wykorzystaj funkcje histogram2D żeby stworzyć histogram dwuwymiarowy.







plt.bar()



matplotlib.pyplot.bar(*left*, *height*, *width=0.8*, *bottom=None*, *hold=None*, *data=None*, ***kwargs*)

Make a bar plot.

Make a bar plot with rectangles bounded by:

`left`, `left + width`, `bottom`, `bottom + height`

(left, right, bottom and top edges)

Parameters:

left : sequence of scalars

the x coordinates of the left sides of the bars

height : sequence of scalars

the heights of the bars

width : scalar or array-like, optional

the width(s) of the bars default: 0.8

bottom : scalar or array-like, optional

the y coordinate(s) of the bars default: None

color : scalar or array-like, optional

the colors of the bar faces

edgecolor : scalar or array-like, optional

the colors of the bar edges

linewidth : scalar or array-like, optional

width of bar edge(s). If None, use default linewidth; If 0, don't draw edges. default: None

tick_label : string or array-like, optional

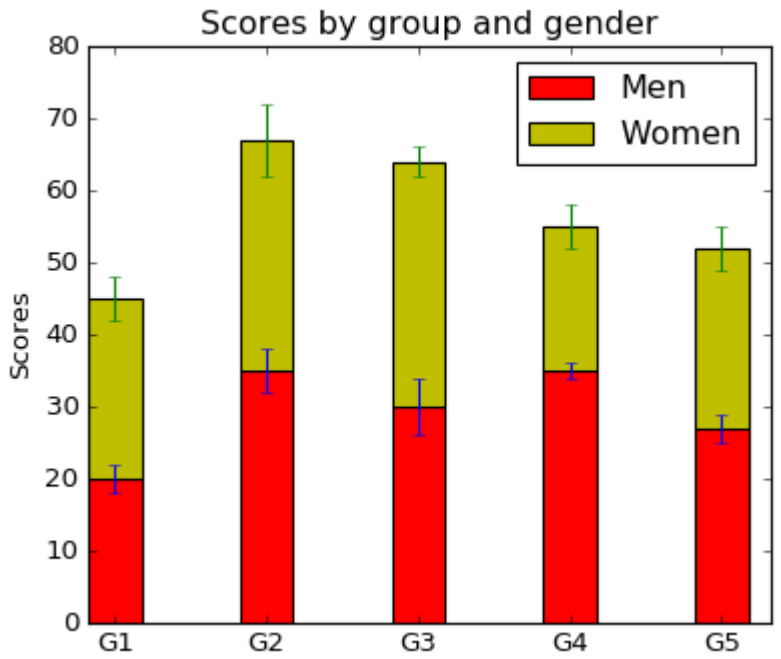
the tick labels of the bars default: None

xerr : scalar or array-like, optional

if not None, will be used to generate errorbar(s) on the bar chart default: None

yerr : scalar or array-like, optional

plt.bar()



matplotlib.pyplot.bar(*left*, *height*, *width=0.8*, *bottom=None*, *hold=None*, *data=None*, ***kwargs*)

Make a bar plot.

Make a bar plot with rectangles bounded by:

`left`, `left + width`, `bottom`, `bottom + height`

(left, right, bottom and top edges)

Parameters:

left : sequence of scalars

the x coordinates of the left sides of the bars

height : sequence of scalars

the heights of the bars

width : scalar or array-like, optional

the width(s) of the bars default: 0.8

bottom : scalar or array-like, optional

the y coordinate(s) of the bars default: None

color : scalar or array-like, optional

the colors of the bar faces

edgecolor : scalar or array-like, optional

the colors of the bar edges

linewidth : scalar or array-like, optional

width of bar edge(s). If None, use default linewidth; If 0, don't draw edges. default: None

tick_label : string or array-like, optional

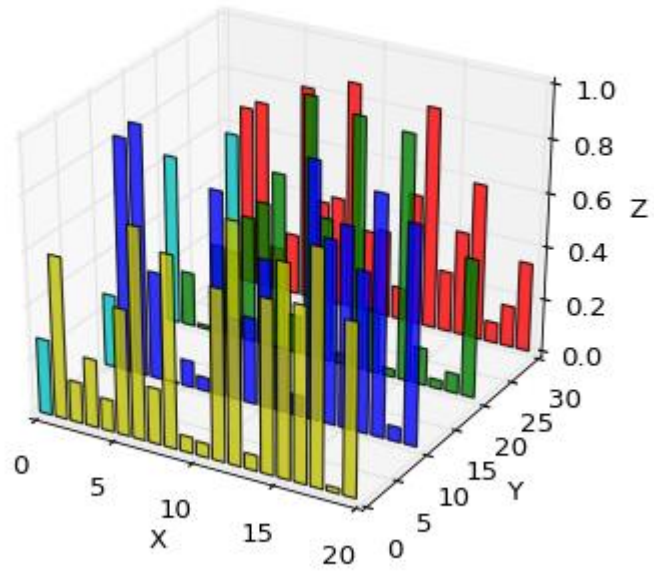
the tick labels of the bars default: None

xerr : scalar or array-like, optional

if not None, will be used to generate errorbar(s) on the bar chart default: None

yerr : scalar or array-like, optional

Axes3D.bar()



`bar(left, height, zs=0, zdir='z', *args, **kwargs)`

Add 2D bar(s).

Argument	Description
left	The x coordinates of the left sides of the bars.
height	The height of the bars.
zs	Z coordinate of bars, if one value is specified they will all be placed at the same z.
zdir	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.

Keyword arguments are passed onto `bar()`.

Returns a `Patch3DCollection`

Axes3D.bar()

- Axes3D(plt.gcf()).bar(x, y, z)

`bar(left, height, zs=0, zdir='z', *args, **kwargs)`

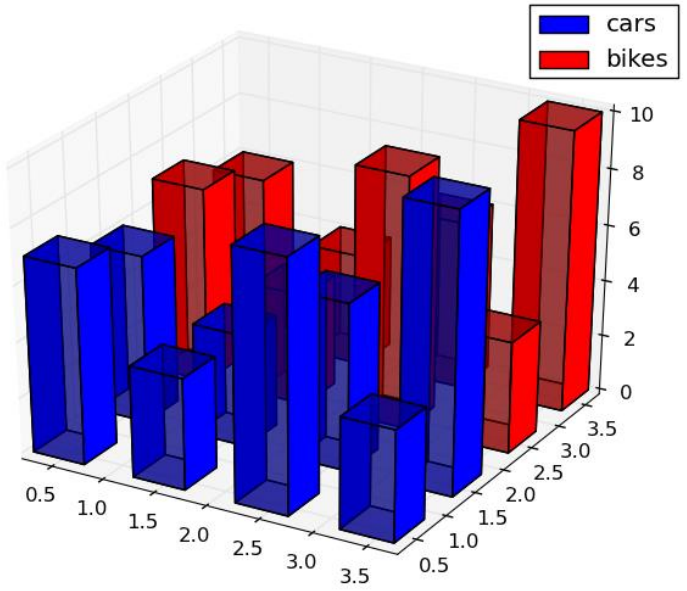
Add 2D bar(s).

Argument	Description
left	The x coordinates of the left sides of the bars.
height	The height of the bars.
zs	Z coordinate of bars, if one value is specified they will all be placed at the same z.
zdir	Which direction to use as z ('x', 'y' or 'z') when plotting a 2D set.

Keyword arguments are passed onto `bar()`.

Returns a `Patch3DCollection`

Axes3D.bar3d()



`bar3d(x, y, z, dx, dy, dz, color='b', zsort='average', *args, **kwargs)`

Generate a 3D bar, or multiple bars.

When generating multiple bars, x, y, z have to be arrays. dx, dy, dz can be arrays or scalars.

color can be:

- A single color value, to color all bars the same color.
- An array of colors of length N bars, to color each bar independently.
- An array of colors of length 6, to color the faces of the bars similarly.
- An array of colors of length 6 * N bars, to color each face independently.

When coloring the faces of the boxes specifically, this is the order of the coloring:

1. -Z (bottom of box)
2. +Z (top of box)
3. -Y
4. +Y
5. -X
6. +X

Keyword arguments are passed onto `Poly3DCollection()`

Axes3D.bar3d()

- Axes3D(plt.gcf()).bar3d(x, y, z, dx, dy, dz)
- Wszystkie tablice muszą być jednowymiarowe!

`bar3d(x, y, z, dx, dy, dz, color='b', zsort='average', *args, **kwargs)`

Generate a 3D bar, or multiple bars.

When generating multiple bars, x, y, z have to be arrays. dx, dy, dz can be arrays or scalars.

color can be:

- A single color value, to color all bars the same color.
- An array of colors of length N bars, to color each bar independently.
- An array of colors of length 6, to color the faces of the bars similarly.
- An array of colors of length 6 * N bars, to color each face independently.

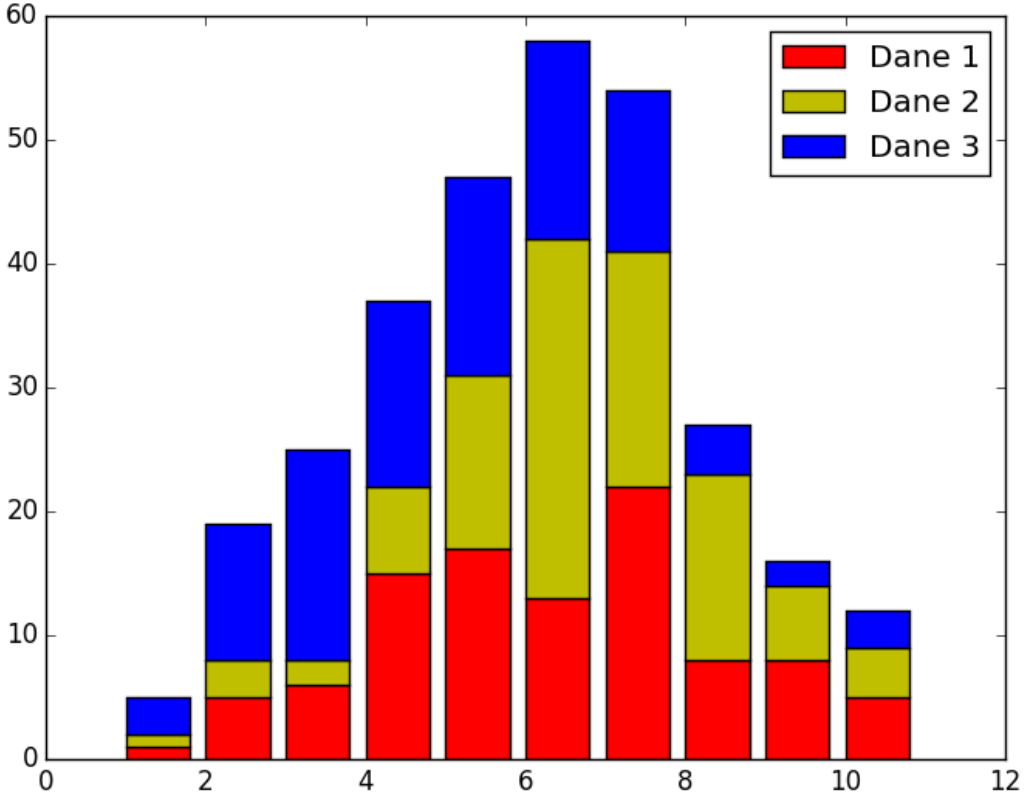
When coloring the faces of the boxes specifically, this is the order of the coloring:

1. -Z (bottom of box)
2. +Z (top of box)
3. -Y
4. +Y
5. -X
6. +X

Keyword arguments are passed onto `Poly3DCollection()`

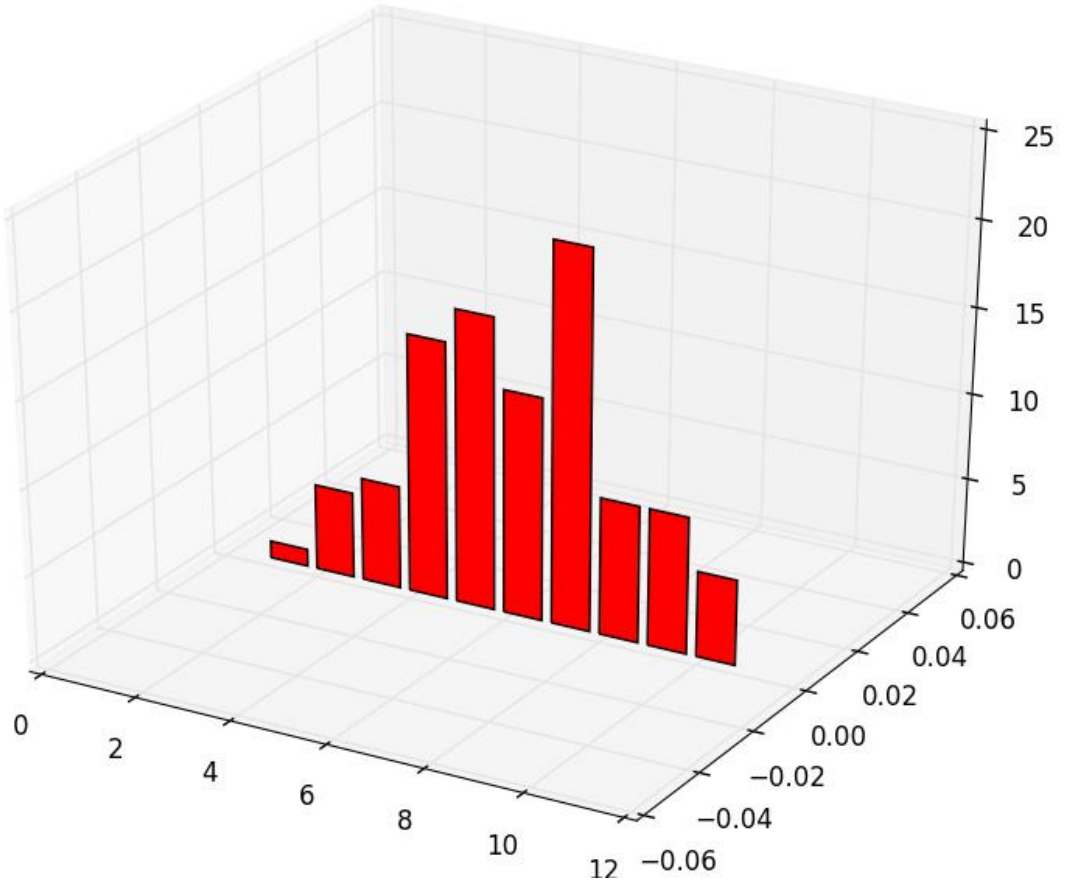
Bar plots

- Ściągnij plik zawierający dane trzech rozkładów empirycznych (każdy rozkład jest w osobnym wierszu). Stwórz wykres kolumnowy tych danych tak jak na obrazie. Wykorzystaj funkcje **plt.bar()** i parametr **bottom**.



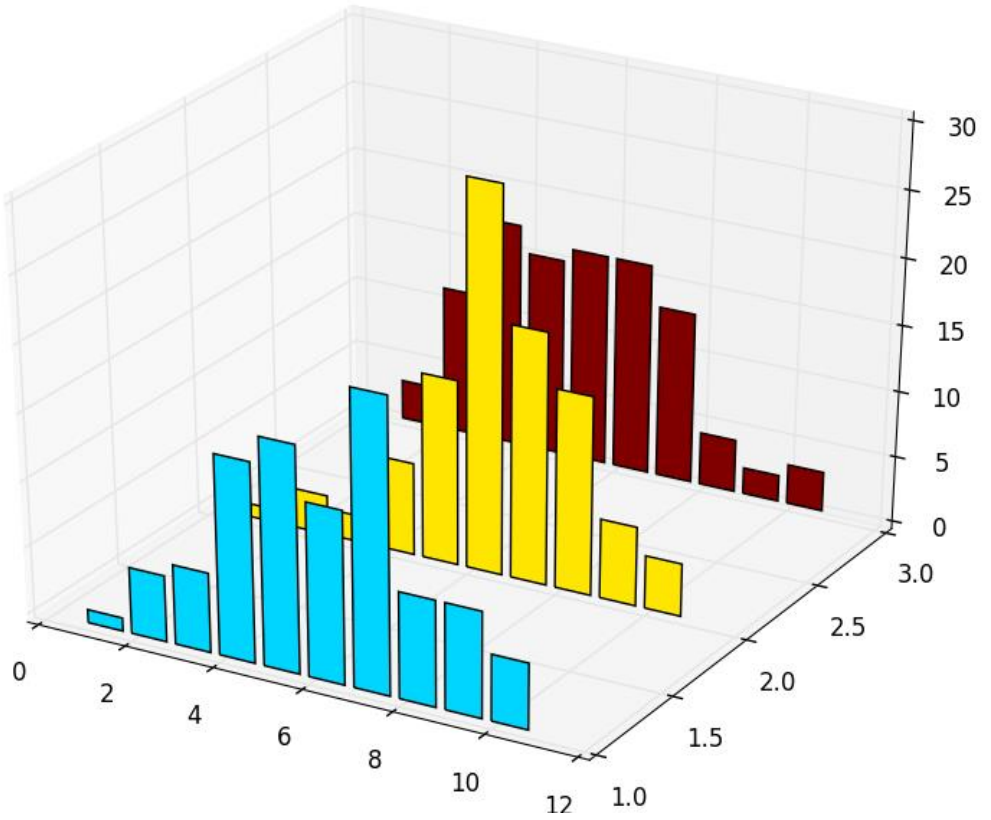
Bar plots 3D

- Stwórz wykres kolumnowy trójwymiarowy poprzednich danych tak jak na obrazie (tylko jeden rozkład ma być zwizualizowany). Wykorzystaj funkcje **Axes3D.bar(x, y)** i parametr **zdir**.



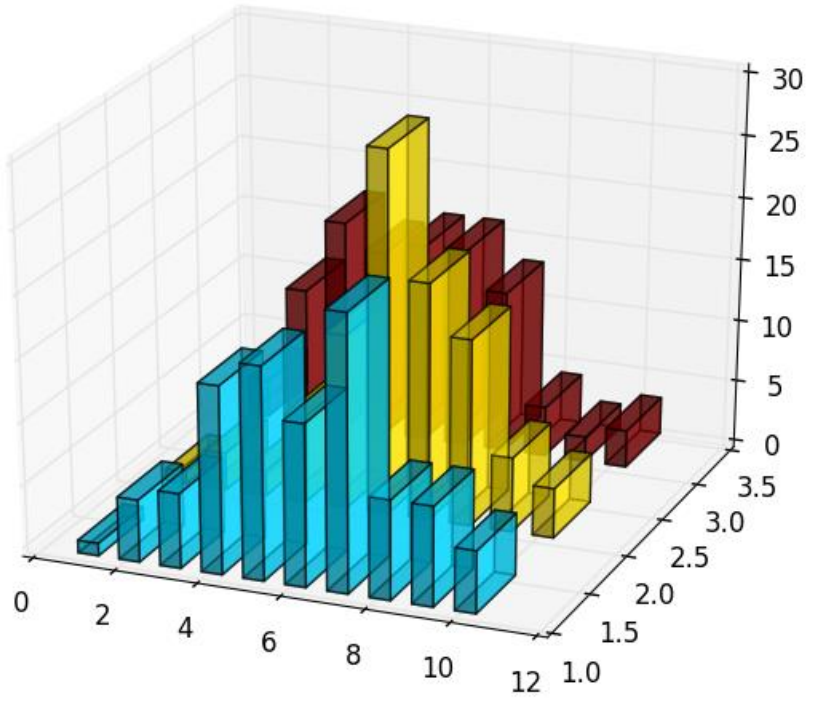
Bar plots 3D

- Stwórz wykres kolumnowy trójwymiarowy poprzednich danych tak jak na obrazie. Wykorzystaj funkcje **Axes3D.bar(x, y, z)** i funkcje **flatten()**. Zastosuj mapę kolorów **jet** względem osi y.



Bar plots 3D

- Stwórz wykres kolumnowy trójwymiarowy poprzednich danych tak jak na obrazie. Wykorzystaj funkcje **Axes3D.bar3d(), flatten()** i parametr **alpha**. Zastosuj mapę kolorów **jet** względem osi y.



numpy.mean

numpy.**mean**(*a*, *axis=None*, *dtype=None*, *out=None*, *keepdims=False*) [source]

Compute the arithmetic mean along the specified axis.

Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. float64 intermediate and return values are used for integer inputs.

Examples

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a)
2.5
>>> np.mean(a, axis=0)
array([ 2.,  3.])
>>> np.mean(a, axis=1)
array([ 1.5,  3.5])
```

numpy.std (odchylenie standardowe)

numpy.std(*a, axis=None, dtype=None, out=None, ddof=0, keepdims=False*) [source]

Compute the standard deviation along the specified axis.

Returns the standard deviation, a measure of the spread of a distribution, of the array elements. The standard deviation is computed for the flattened array by default, otherwise over the specified axis.

Examples

```
>>>
>>> a = np.array([[1, 2], [3, 4]])
>>> np.std(a)
1.1180339887498949
>>> np.std(a, axis=0)
array([ 1.,  1.])
>>> np.std(a, axis=1)
array([ 0.5,  0.5])
```

numpy.cov

numpy.cov(*m*, *y=None*, *rowvar=1*, *bias=0*, *ddof=None*, *fweights=None*, *aweights=None*) [source]

Estimate a covariance matrix, given data and weights.

Covariance indicates the level to which two variables vary together. If we examine N-dimensional samples, $X = [x_1, x_2, ...x_N]^T$, then the covariance matrix element C_{ij} is the covariance of x_i and x_j. The element C_{ii} is the variance of x_i.

```
>>> x
array([[0, 1, 2],
       [2, 1, 0]])

>>> np.cov(x)
array([[ 1., -1.],
       [-1.,  1.]])
```

numpy.any

numpy.any(*a*, *axis=None*, *out=None*, *keepdims=False*) [source]

Test whether any array element along a given axis evaluates to True.

Returns single boolean unless *axis* is not `None`

Parameters: **a** : *array_like*
Input array or object that can be converted to an array.

axis : *None or int or tuple of ints, optional*
Axis or axes along which a logical OR reduction is performed. The default (*axis* = *None*) is to perform a logical OR over all the dimensions of the input array. *axis* may be negative, in which case it counts from the last to the first axis.
New in version 1.7.0.
If this is a tuple of ints, a reduction is performed on multiple axes, instead of a single axis or all the axes as before.

Examples

```
>>> np.any([[True, False], [True, True]])
True
```

```
>>> np.any([[True, False], [False, False]], axis=0)
array([ True, False], dtype=bool)
```

numpy.argmax

numpy.argmax(*a, axis=None, out=None***)¶** [source]

Returns the indices of the maximum values along an axis.

Examples

```
>>> a = np.arange(6).reshape(2,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.argmax(a)
5
>>> np.argmax(a, axis=0)
array([1, 1, 1])
>>> np.argmax(a, axis=1)
array([2, 2])
```

numpy.gradient metoda różnic skończonych

numpy.**gradient**(*f*, **varargs*, ***kwargs*) [source]

Return the gradient of an N-dimensional array.

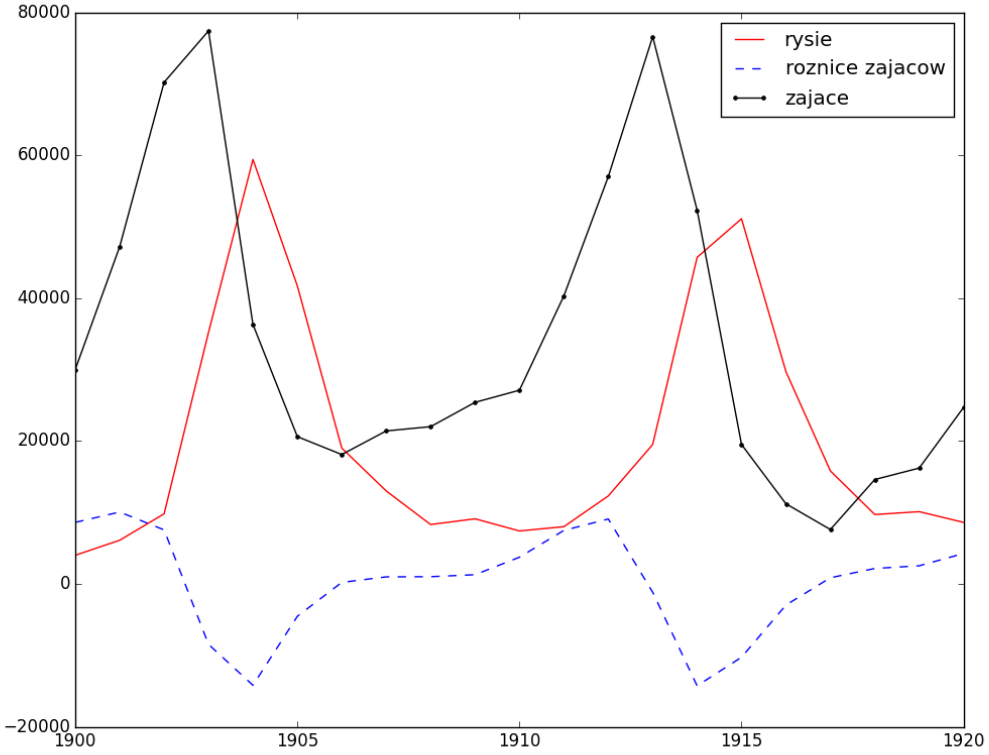
The gradient is computed using second order accurate central differences in the interior and either first differences or second order accurate one-sides (forward or backwards) differences at the boundaries. The returned gradient hence has the same shape as the input array.

Examples

```
>>> x = np.array([1, 2, 4, 7, 11, 16], dtype=np.float)
>>> np.gradient(x)
array([ 1. ,  1.5,  2.5,  3.5,  4.5,  5. ])
```

Zadania

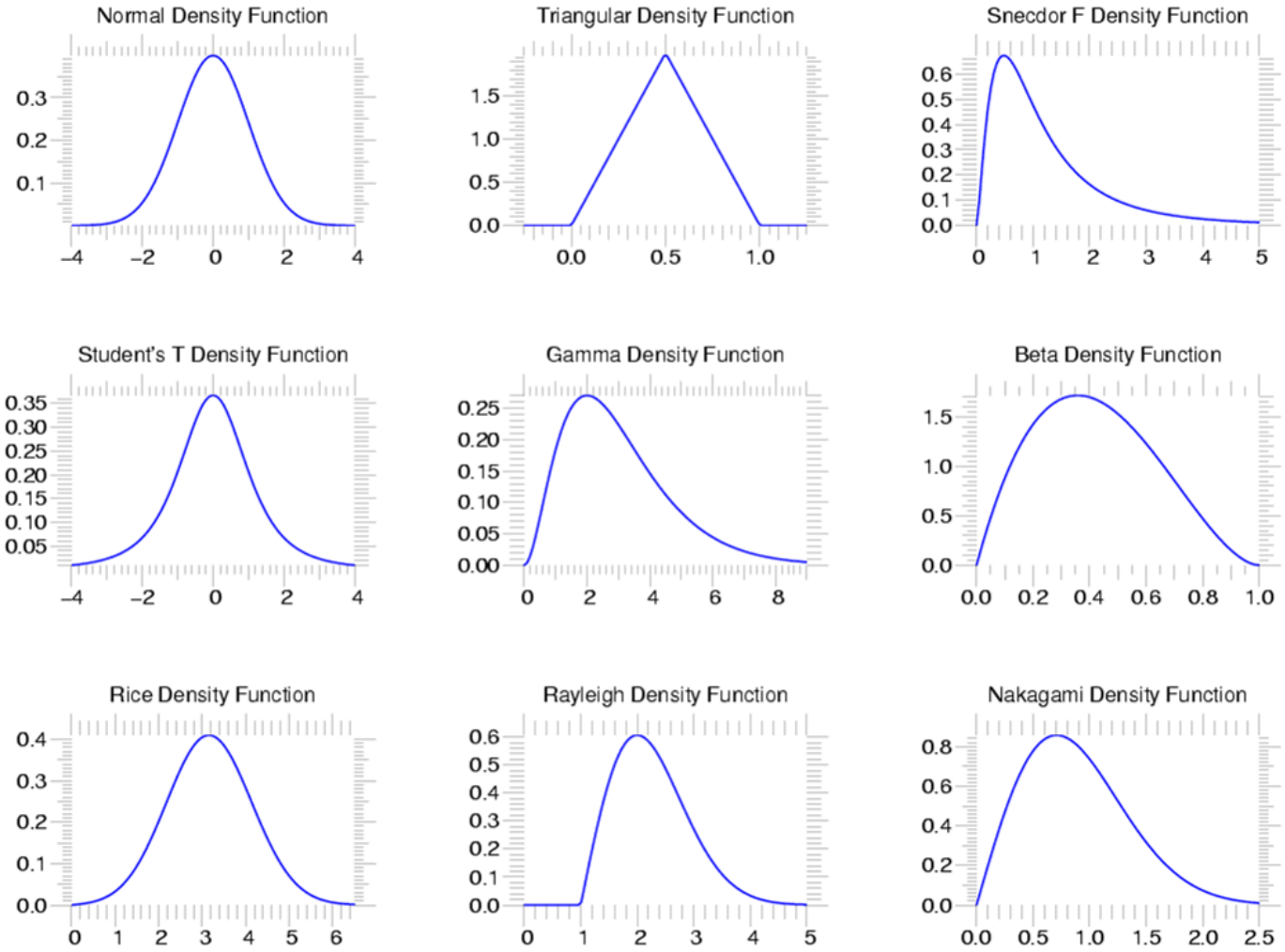
- Ściągnij plik populacje.txt który podaje liczby zajęcy, lisów i marchewek w jakimś przedziale czasowym. Oblicz średnią i odchylenie standardowe populacji; podaj rok w którym każdy gatunek miał największą populacje; dla każdego roku jaki gatunek miał największą populacje; w jakich latach jaka populacja była powyżej 50,000 istot; stwórz wykresy *różnic* w populacji zajęcy, liczby zajęcy i liczby lisów – oblicz kowariancje pomiędzy różnicami populacji zajęcy i populacji lisów.



Statystyka: scipy.stats

- Ponad 80 ciągłych rozkładów stochastycznych

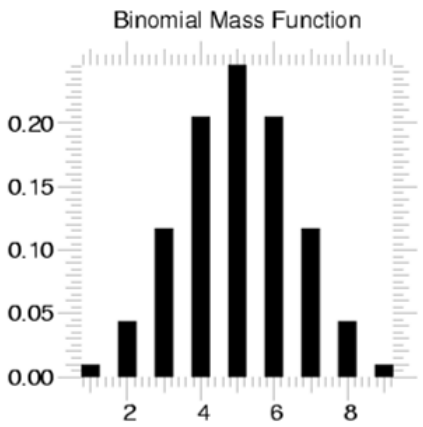
```
pdf
cdf
Rvs
ppf
fit
var
Mean
std
```

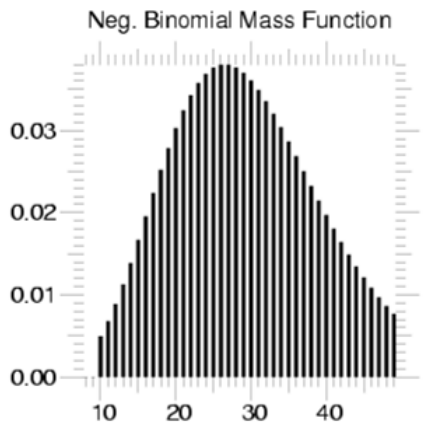


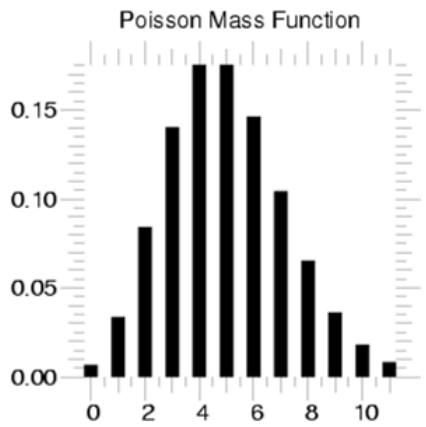
Statystyka: scipy.stats

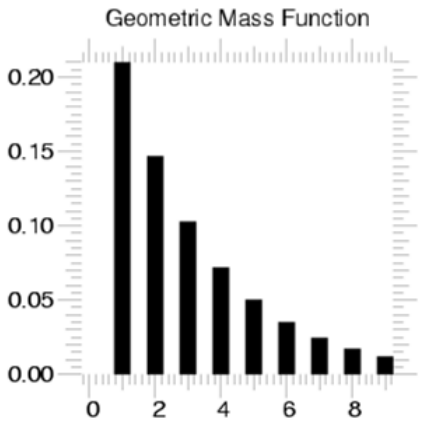
- 10 dyskretnych rozkładów stochastycznych

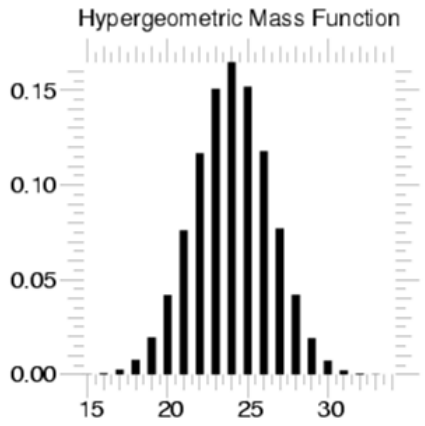
```
pdf
cdf
Rvs
ppf
fit
var
Mean
std
```

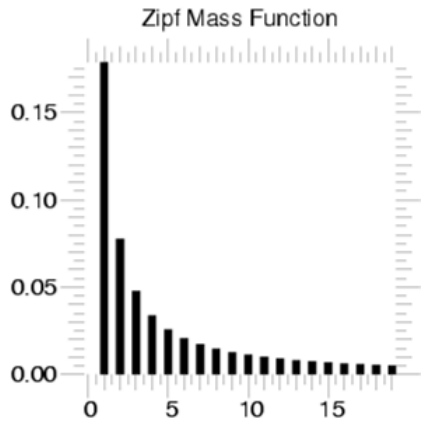








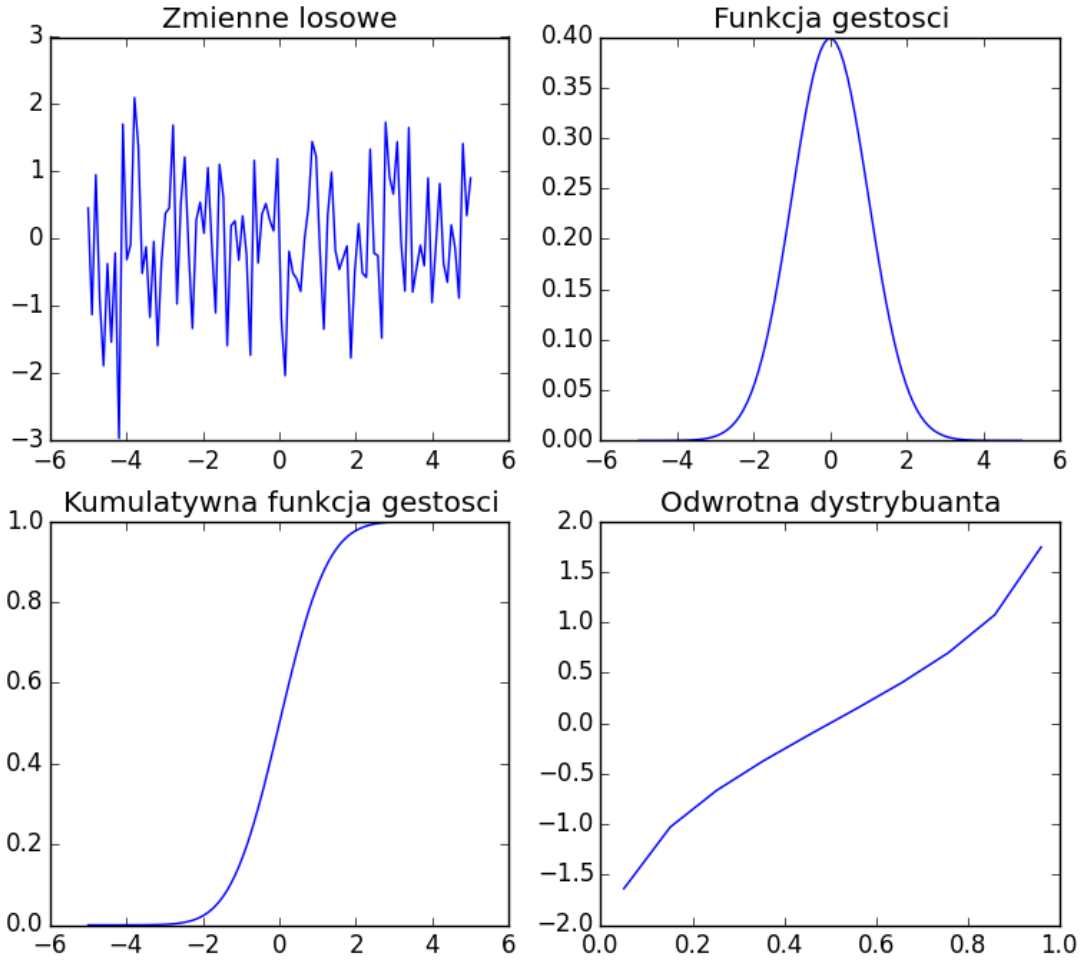




Obiekty stats

```
In [1]: from scipy.stats import norm
In [11]: x = np. linspace(-5, 5, 100)
In [12]: pdf = norm.pdf(x)
In [13]: cdf = norm.cdf(x)
In [14]: ppf = norm.ppf(x)
```

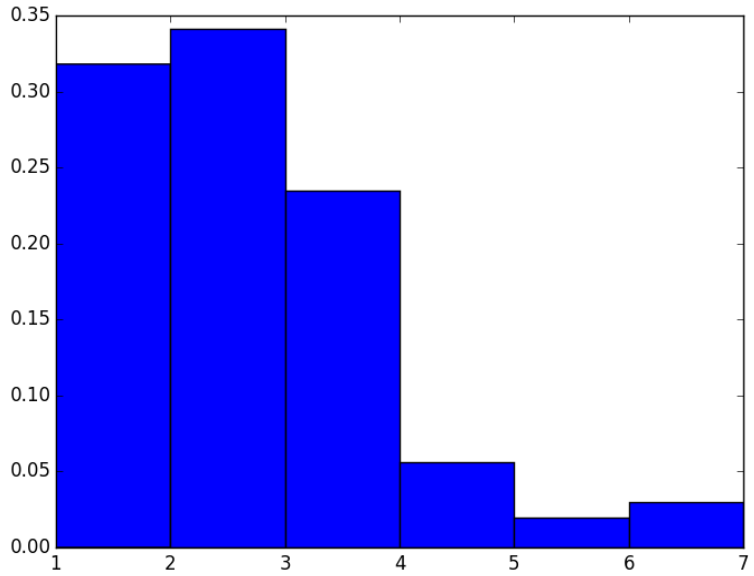
```
In [10]: probka = norm.rvs(size = 100)
In [54]: mu, sigma = norm.fit(probka)
In [58]: print mu, sigma
0.0539679827053 1.03236227751
```



Używanie obiektów stats

```
from scipy.stats import rv_discrete

xk = [1,2,3,4,5,6]
pk = [0.3,0.35,0.25,0.05,0.025,0.025]
new = rv_discrete(values=(xk,pk))
samples = new.rvs(size=1000)
```



scipy.stats.gaussian_kde¶

class **scipy.stats.gaussian_kde(***dataset, bw_method=None***)** [source]

Representation of a kernel-density estimate using Gaussian kernels.

Kernel density estimation is a way to estimate the probability density function (PDF) of a random variable in a non-parametric way. gaussian_kde works for both uni-variate and multi-variate data. It includes automatic bandwidth determination. The estimation works best for a unimodal distribution; bimodal or multi-modal distributions tend to be oversmoothed.

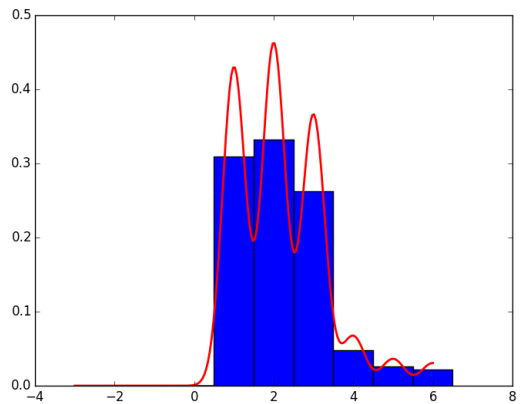
Parameters:

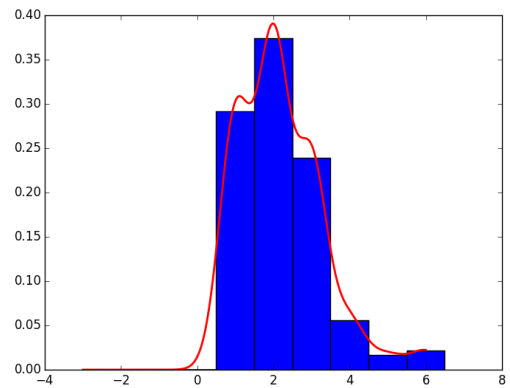
dataset : *array_like*

Datapoints to estimate from. In case of univariate data this is a 1-D array, otherwise a 2-D array with shape (# of dims, # of data).

bw_method : *str, scalar or callable, optional*

The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If a scalar, this will be used directly as *kde.factor*. If a callable, it should take a gaussian_kde instance as only parameter and return a scalar. If None (default), 'scott' is used. See Notes for more details.





Testowanie hipotez statystycznych

scipy.stats.ttest_ind

scipy.stats.ttest_ind(*a*, *b*, *axis=0*, *equal_var=True*)¶ [source]

Calculates the T-test for the means of TWO INDEPENDENT samples of scores.

This is a two-sided test for the null hypothesis that 2 independent samples have identical average (expected) values. This test assumes that the populations have identical variances.

Parameters:

a, b : *array_like*
The arrays must have the same shape, except in the dimension corresponding to *axis* (the first, by default).

axis : *int, optional*
Axis can equal None (ravel array first), or an integer (the axis over which to operate on a and b).

equal_var : *bool, optional*
If True (default), perform a standard independent 2 sample test that assumes equal population variances [R315]. If False, perform Welch's t-test, which does not assume equal population variance [R316].
New in version 0.11.0.

Returns:

t : *float or array*
The calculated t-statistic.

prob : *float or array*
The two-tailed p-value.

numpy.polyfit

Dopasywowanie wielomianu metoda najmnieszych kwadratow

numpy.**polyfit**(*x*, *y*, *deg*, *rcond=None*, *full=False*, *w=None*, *cov=False*) [source]

Least squares polynomial fit.

Fit a polynomial `p(x) = p[0] * x**deg + ... + p[deg]` of degree *deg* to points *(x, y)*. Returns a vector of coefficients *p* that minimises the squared error.

Examples

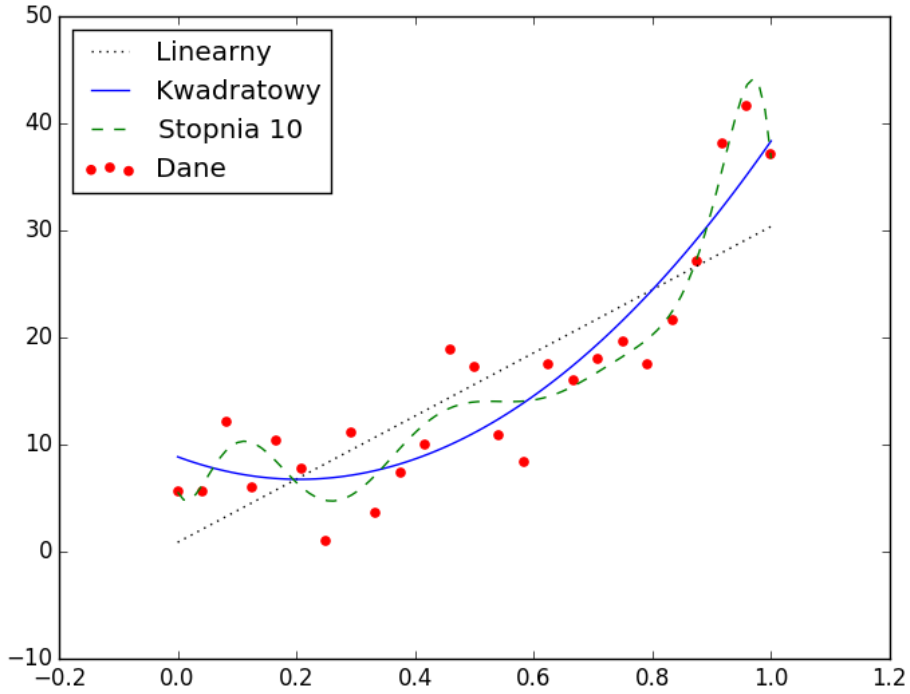
```
>>> x = np.array([0.0, 1.0, 2.0, 3.0,  4.0,  5.0])
>>> y = np.array([0.0, 0.8, 0.9, 0.1, -0.8, -1.0])
>>> z = np.polyfit(x, y, 3)
>>> z
array([ 0.08703704, -0.81349206,  1.69312169, -0.03968254])
```

It is convenient to use poly1d objects for dealing with polynomials:

```
>>> p = np.poly1d(z)
>>> p(0.5)
0.6143849206349179
>>> p(3.5)
-0.34732142857143039
>>> p(10)
22.579365079365115
```

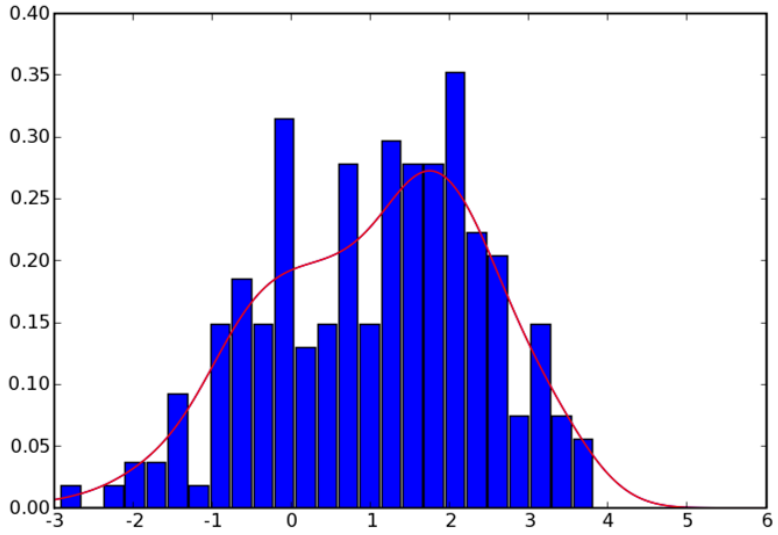
Zadanie

- Dopasuj wielomian do danych x i y minimalizując sumę kwadratów błędów: linearny, kwadratowy, 10 stopnia. Wyświetl punkty na wykresie i narysuj wielomiany aproksymujące.



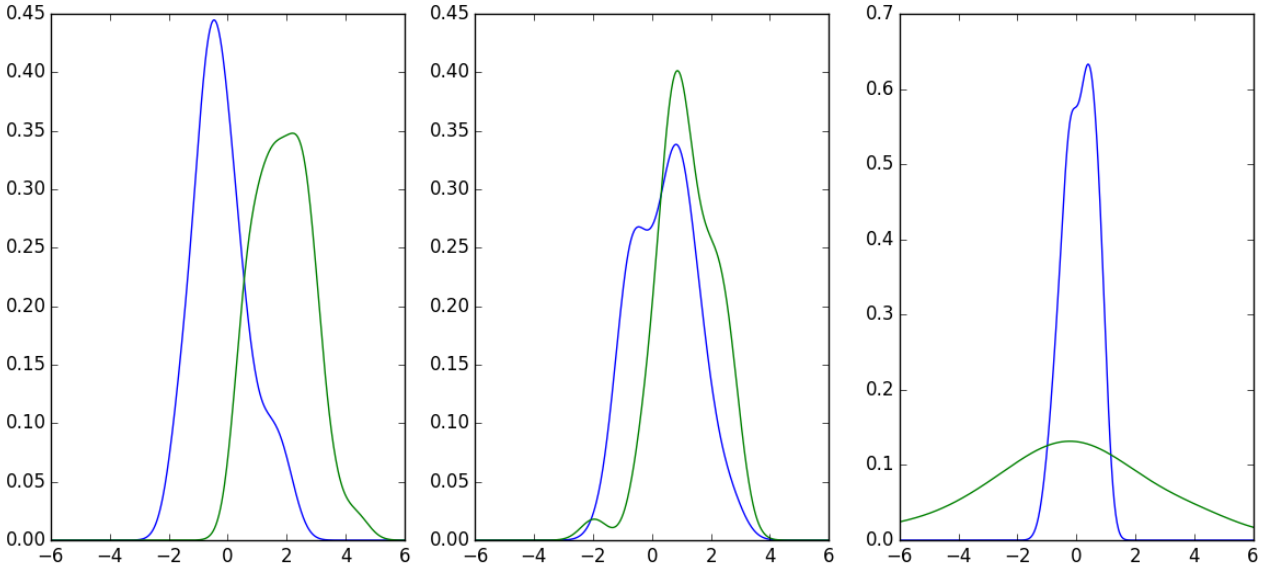
Zadanie

- Stwórz rozkład danych który się składa z dwóch rozkładów normalnych (mu, sigma = (0.0, 1.0) i (2.0, 0.8)). Stwórz jądrowy estymator gęstości (kernel density estimation) tego rozkładu. Narysuj wykres estymatora jak i histogram danych.



Zadanie

- Dla różnych par rozkładów empirycznych (dwóch niezależnych zmiennych) stwórz wykresy jądrowych estymatorów gęstości i sprawdź czy średnie się znacząco różnią od siebie. Zbadaj dla każdego rozkładu hipotezę zerową: że średnia jest równa 0 (stats.ttest_1samp).



numpy.linalg.eig

Wartosci i wektory wlasne

numpy.linalg.eig(*a*) [source]

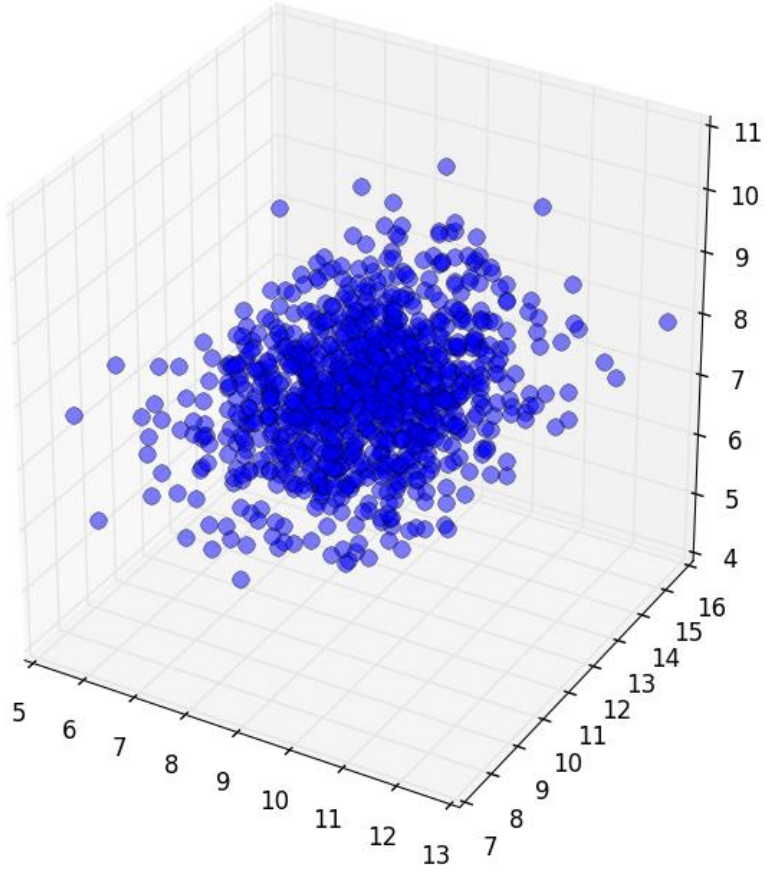
Compute the eigenvalues and right eigenvectors of a square array.

```
>>> w, v = LA.eig(np.diag((1, 2, 3)))
>>> w; v
array([ 1.,  2.,  3.])
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

>>>

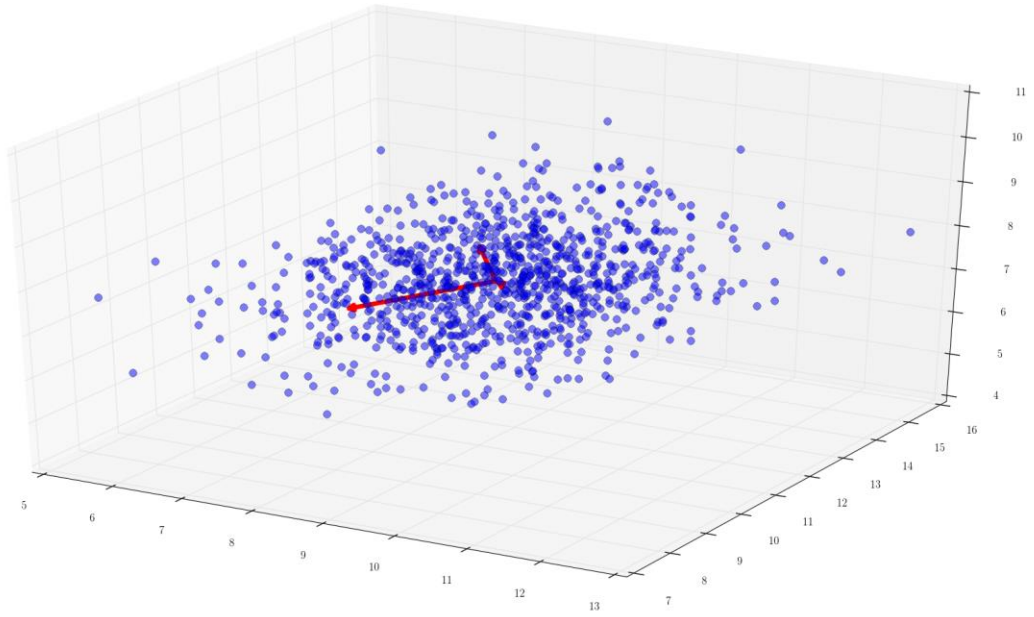
Zadanie

- Zwizualizuj pomiary płatek i działek kwiatu *ludwiga octovalvis* na wykresie trójwymiarowym. Wypisz średnią jak i macierz kowariancji – co można o zależnościach tych trzech parametrów (długość płatek, długość działek, grubość działek) powiedzieć?



Zadanie

- Oblicz wektory i wartości własne macierzy kowariancji. Wyświetl wektory na poprzednim wykresie (kod). Żeby zredukować wymiar danych z trzech do dwóch wymiarów (minimalizując utratę informacji) jaką podprzestrzeń można zaproponować?



Zadanie

- Zrzuć dane z obecnej przestrzeni trójwymiarowej na nową przestrzeń dwuwymiarową, która jest wyznaczona przez dwóch największych wektorów własnych. Zwizualizuj dane jako wykres dwuwymiarowy. Porównaj wyniki z metodą PCA z modułu matplotlib.mlab. Jakie różnice da się zauważyć?