Sourcedoc

API Documentation

April 18, 2013

Contents

1 Module Save_PDF_CSV

Created on 14 de Abr de 2013

Author: admin1

1.1 Variables

Name	Description
TITLE	**Value:** `<functools.partial object at 0x902811c>`
___package___	**Value:** `None`

1.2 Class PDF

```
object ─┐
        │
fpdf.fpdf.FPDF ─┐
                │
                Save_PDF_CSV.PDF
```

Responsavel por gerár o PDF com informação conultada pelo utilizador

1.2.1 Methods

footer(*self*)

Adiciona os números de página

Overrides: fpdf.fpdf.FPDF.footer

header(*self*)

Constroi o ficheiro pdf com informação gerada na pesquisa

Overrides: fpdf.fpdf.FPDF.header

setTitle(*self*, *title*)

Adiciona o titulo ao pdf

Inherited from fpdf.fpdf.FPDF

___init___(), accept_page_break(), add_font(), add_link(), add_page(), alias_nb_pages(), cell(), close(), code39(), error(), get_string_width(), get_x(), get_y(), image(), interleaved2of5(), line(), link(), ln(), multi_cell(), normalize_text(), open(), output(), page_no(), rect(), rotate(), set_author(), set_auto_page_break(), set_compression(), set_creator(), set_display_mode(), set_draw_color(), set_fill_color(), set_font(), set_font_size(), set_keywords(), set_left_margin(), set_line_width(), set_link(), set_margins(), set_right_margin(), set_subject(), set_text_color(), set_title(), set_top_margin(), set_x(), set_xy(), set_y(), text(), write()

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

1.2.2 Properties

Name	Description
Inherited from object	
___class___	

2 Module appSegInformatica

Created on 19 de Mar de 2013

Authors: António Baião N:5604, Carlos Palma N:5608

2.1 Functions

beta(*a*, *b*, *size*=None)

```
The Beta distribution over ``[0, 1]``.

The Beta distribution is a special case of the Dirichlet distribution,
and is related to the Gamma distribution.  It has the probability
distribution function

.. math:: f(x; a,b) = \frac{1}{B(\alpha, \beta)} x^{\alpha - 1}
                                                 (1 - x)^{\beta - 1},

where the normalisation, B, is the beta function,

.. math:: B(\alpha, \beta) = \int_0^1 t^{\alpha - 1}
                             (1 - t)^{\beta - 1} dt.

It is often seen in Bayesian inference and order statistics.

Parameters
----------
a : float
    Alpha, non-negative.
b : float
    Beta, non-negative.
size : tuple of ints, optional
    The number of samples to draw.  The ouput is packed according to
    the size given.

Returns
-------
out : ndarray
    Array of the given shape, containing values drawn from a
    Beta distribution.
```

binomial(*n*, *p*, *size*=None)

```
Draw samples from a binomial distribution.

Samples are drawn from a Binomial distribution with specified
parameters, n trials and p probability of success where
n an integer > 0 and p is in the interval [0,1]. (n may be
input as a float, but it is truncated to an integer in use)

Parameters
----------
n : float (but truncated to an integer)
        parameter, > 0.
p : float
        parameter, >= 0 and <=1.
size : {tuple, int}
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Returns
-------
samples : {ndarray, scalar}
          where the values are all integers in  [0, n].

See Also
--------
scipy.stats.distributions.binom : probability density function,
    distribution or cumulative density function, etc.

Notes
-----
The probability density for the Binomial distribution is

.. math:: P(N) = \binom{n}{N}p^N(1-p)^{n-N},

where :math:`n` is the number of trials, :math:`p` is the probability
of success, and :math:`N` is the number of successes.

When estimating the standard error of a proportion in a population by
using a random sample, the normal distribution works well unless the
product p*n <=5, where p = population proportion estimate, and n =
number of samples, in which case the binomial distribution is used
instead. For example, a sample of 15 people shows 4 who are left
handed, and 11 who are right handed. Then p = 4/15 = 27%. 0.27*15 = 4,
so the binomial distribution should be used in this case.
```



```
References
----------
.. [1] Dalgaard, Peter, "Introductory Statistics with R",
       Springer-Verlag, 2002.
```

chisquare(*df*, *size*=None)

```
Draw samples from a chi-square distribution.

When `df` independent random variables, each with standard normal
distributions (mean 0, variance 1), are squared and summed, the
resulting distribution is chi-square (see Notes).  This distribution
is often used in hypothesis testing.

Parameters
----------
df : int
     Number of degrees of freedom.
size : tuple of ints, int, optional
     Size of the returned array.  By default, a scalar is
     returned.

Returns
-------
output : ndarray
    Samples drawn from the distribution, packed in a `size`-shaped
    array.

Raises
------
ValueError
    When `df` <= 0 or when an inappropriate `size` (e.g. ``size=-1``)
    is given.

Notes
-----
The variable obtained by summing the squares of `df` independent,
standard normally distributed random variables:

.. math:: Q = \sum_{i=0}^{\mathtt{df}} X^2_i

is chi-square distributed, denoted

.. math:: Q \sim \chi^2_k.

The probability density function of the chi-squared distribution is

.. math:: p(x) = \frac{(1/2)^{k/2}}{\Gamma(k/2)}
                 x^{k/2 - 1} e^{-x/2},

where :math:`\Gamma` is the gamma function,

.. math:: \Gamma(x) = \int_0^{-\infty} t^{x - 1} e^{-t} dt.

References
```

exponential(*scale*=1.0, *size*=None)

```
Exponential distribution.

Its probability density function is

.. math:: f(x; \frac{1}{\beta}) = \frac{1}{\beta} \exp(-\frac{x}{\beta}),

for ``x > 0`` and 0 elsewhere. :math:`\beta` is the scale parameter,
which is the inverse of the rate parameter :math:`\lambda = 1/\beta`.
The rate parameter is an alternative, widely used parameterization
of the exponential distribution [3]_.

The exponential distribution is a continuous analogue of the
geometric distribution.  It describes many common situations, such as
the size of raindrops measured over many rainstorms [1]_, or the time
between page requests to Wikipedia [2]_.

Parameters
----------
scale : float
    The scale parameter, :math:`\beta = 1/\lambda`.
size : tuple of ints
    Number of samples to draw.  The output is shaped
    according to `size`.

References
----------
.. [1] Peyton Z. Peebles Jr., "Probability, Random Variables and
       Random Signal Principles", 4th ed, 2001, p. 57.
.. [2] "Poisson Process", Wikipedia,
       http://en.wikipedia.org/wiki/Poisson_process
.. [3] "Exponential Distribution, Wikipedia,
       http://en.wikipedia.org/wiki/Exponential_distribution
```

f(*dfnum*, *dfden*, *size*=None)

```
Draw samples from a F distribution.

Samples are drawn from an F distribution with specified parameters,
`dfnum` (degrees of freedom in numerator) and `dfden` (degrees of freedom
in denominator), where both parameters should be greater than zero.

The random variate of the F distribution (also known as the
Fisher distribution) is a continuous probability distribution
that arises in ANOVA tests, and is the ratio of two chi-square
variates.

Parameters
----------
dfnum : float
    Degrees of freedom in numerator. Should be greater than zero.
dfden : float
    Degrees of freedom in denominator. Should be greater than zero.
size : {tuple, int}, optional
    Output shape.  If the given shape is, e.g., ``(m, n, k)``,
    then ``m * n * k`` samples are drawn. By default only one sample
    is returned.

Returns
-------
samples : {ndarray, scalar}
    Samples from the Fisher distribution.

See Also
--------
scipy.stats.distributions.f : probability density function,
    distribution or cumulative density function, etc.

Notes
-----

The F statistic is used to compare in-group variances to between-group
variances. Calculating the distribution depends on the sampling, and
so it is a function of the respective degrees of freedom in the
problem.  The variable `dfnum` is the number of samples minus one, the
between-groups degrees of freedom, while `dfden` is the within-groups
degrees of freedom, the sum of the number of samples in each group
minus the number of groups.

References
----------
.. [1] Glantz, Stanton A. "Primer of Biostatistics.", McGraw-Hill,
       Fifth Edition, 2002.
.. [2] Wikipedia, "F-distribution",
```

gamma(*shape*, *scale*=1.0, *size*=None)

```
Draw samples from a Gamma distribution.

Samples are drawn from a Gamma distribution with specified parameters,
`shape` (sometimes designated "k") and `scale` (sometimes designated
"theta"), where both parameters are > 0.

Parameters
----------
shape : scalar > 0
    The shape of the gamma distribution.
scale : scalar > 0, optional
    The scale of the gamma distribution.  Default is equal to 1.
size : shape_tuple, optional
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Returns
-------
out : ndarray, float
    Returns one sample unless `size` parameter is specified.

See Also
--------
scipy.stats.distributions.gamma : probability density function,
    distribution or cumulative density function, etc.

Notes
-----
The probability density for the Gamma distribution is

.. math:: p(x) = x^{k-1}\frac{e^{-x/\theta}}{\theta^k\Gamma(k)},

where :math:`k` is the shape and :math:`\theta` the scale,
and :math:`\Gamma` is the Gamma function.

The Gamma distribution is often used to model the times to failure of
electronic components, and arises naturally in processes for which the
waiting times between Poisson distributed events are relevant.

References
----------
.. [1] Weisstein, Eric W. "Gamma Distribution." From MathWorld--A
       Wolfram Web Resource.
       http://mathworld.wolfram.com/GammaDistribution.html
.. [2] Wikipedia, "Gamma-distribution",
       http://en.wikipedia.org/wiki/Gamma-distribution

Examples
```

geometric(*p*, *size*=None)

```
Draw samples from the geometric distribution.

Bernoulli trials are experiments with one of two outcomes:
success or failure (an example of such an experiment is flipping
a coin).  The geometric distribution models the number of trials
that must be run in order to achieve success.  It is therefore
supported on the positive integers, ``k = 1, 2, ...``.

The probability mass function of the geometric distribution is

.. math:: f(k) = (1 - p)^{k - 1} p

where `p` is the probability of success of an individual trial.

Parameters
----------
p : float
    The probability of success of an individual trial.
size : tuple of ints
    Number of values to draw from the distribution.  The output
    is shaped according to `size`.

Returns
-------
out : ndarray
    Samples from the geometric distribution, shaped according to
    `size`.

Examples
--------
Draw ten thousand values from the geometric distribution,
with the probability of an individual success equal to 0.35:

>>> z = np.random.geometric(p=0.35, size=10000)

How many trials succeeded after a single run?

>>> (z == 1).sum() / 10000.
0.34889999999999999 #random
```

get_state()

```
Return a tuple representing the internal state of the generator.

For more details, see ‘set_state‘.

Returns
-------
out : tuple(str, ndarray of 624 uints, int, int, float)
    The returned tuple has the following items:

    1. the string ’MT19937’.
    2. a 1-D array of 624 unsigned integer keys.
    3. an integer ‘‘pos‘‘.
    4. an integer ‘‘has_gauss‘‘.
    5. a float ‘‘cached_gaussian‘‘.

See Also
--------
set_state

Notes
-----
‘set_state‘ and ‘get_state‘ are not needed to work with any of the
random distributions in NumPy. If the internal state is manually altered,
the user should know exactly what he/she is doing.
```

gumbel(*loc*=0.0, *scale*=1.0, *size*=None)

```
Gumbel distribution.

Draw samples from a Gumbel distribution with specified location and scale.
For more information on the Gumbel distribution, see Notes and References
below.

Parameters
----------
loc : float
    The location of the mode of the distribution.
scale : float
    The scale parameter of the distribution.
size : tuple of ints
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Returns
-------
out : ndarray
    The samples

See Also
--------
scipy.stats.gumbel_l
scipy.stats.gumbel_r
scipy.stats.genextreme
    probability density function, distribution, or cumulative density
    function, etc. for each of the above
weibull

Notes
-----
The Gumbel (or Smallest Extreme Value (SEV) or the Smallest Extreme Value
Type I) distribution is one of a class of Generalized Extreme Value (GEV)
distributions used in modeling extreme value problems.  The Gumbel is a
special case of the Extreme Value Type I distribution for maximums from
distributions with "exponential-like" tails.

The probability density for the Gumbel distribution is

.. math:: p(x) = \frac{e^{-(x - \mu)/ \beta}}{\beta} e^{ -e^{-(x - \mu)/
          \beta}},

where :math:`\mu` is the mode, a location parameter, and :math:`\beta` is
the scale parameter.

The Gumbel (named for German mathematician Emil Julius Gumbel) was used
very early in the hydrology literature, for modeling the occurrence of
```

hypergeometric(*ngood*, *nbad*, *nsample*, *size*=None)

```
Draw samples from a Hypergeometric distribution.

Samples are drawn from a Hypergeometric distribution with specified
parameters, ngood (ways to make a good selection), nbad (ways to make
a bad selection), and nsample = number of items sampled, which is less
than or equal to the sum ngood + nbad.

Parameters
----------
ngood : float (but truncated to an integer)
        parameter, > 0.
nbad  : float
        parameter, >= 0.
nsample  : float
           parameter, > 0 and <= ngood+nbad
size : {tuple, int}
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Returns
-------
samples : {ndarray, scalar}
          where the values are all integers in  [0, n].

See Also
--------
scipy.stats.distributions.hypergeom : probability density function,
    distribution or cumulative density function, etc.

Notes
-----
The probability density for the Hypergeometric distribution is

.. math:: P(x) = \frac{\binom{m}{n}\binom{N-m}{n-x}}{\binom{N}{n}},

where :math:`0 \le x \le m` and :math:`n+m-N \le x \le n`

for P(x) the probability of x successes, n = ngood, m = nbad, and
N = number of samples.

Consider an urn with black and white marbles in it, ngood of them
black and nbad are white. If you draw nsample balls without
replacement, then the Hypergeometric distribution describes the
distribution of black balls in the drawn sample.

Note that this distribution is very similar to the Binomial
distribution, except that in this case, samples are drawn without
replacement, whereas in the Binomial case samples are drawn with
```

laplace(*loc*=0.0, *scale*=1.0, *size*=None)

```
Draw samples from the Laplace or double exponential distribution with
specified location (or mean) and scale (decay).

The Laplace distribution is similar to the Gaussian/normal distribution,
but is sharper at the peak and has fatter tails. It represents the
difference between two independent, identically distributed exponential
random variables.

Parameters
----------
loc : float
    The position, :math:`\mu`, of the distribution peak.
scale : float
    :math:`\lambda`, the exponential decay.

Notes
-----
It has the probability density function

.. math:: f(x; \mu, \lambda) = \frac{1}{2\lambda}
                               \exp\left(-\frac{|x - \mu|}{\lambda}\right).

The first law of Laplace, from 1774, states that the frequency of an error
can be expressed as an exponential function of the absolute magnitude of
the error, which leads to the Laplace distribution. For many problems in
Economics and Health sciences, this distribution seems to model the data
better than the standard Gaussian distribution


References
----------
.. [1] Abramowitz, M. and Stegun, I. A. (Eds.). Handbook of Mathematical
       Functions with Formulas, Graphs, and Mathematical Tables, 9th
       printing.  New York: Dover, 1972.

.. [2] The Laplace distribution and generalizations
       By Samuel Kotz, Tomasz J. Kozubowski, Krzysztof Podgorski,
       Birkhauser, 2001.

.. [3] Weisstein, Eric W. "Laplace Distribution."
       From MathWorld--A Wolfram Web Resource.
       http://mathworld.wolfram.com/LaplaceDistribution.html

.. [4] Wikipedia, "Laplace distribution",
       http://en.wikipedia.org/wiki/Laplace_distribution

Examples
--------
```

logistic(*loc*=0.0, *scale*=1.0, *size*=None)

```
Draw samples from a Logistic distribution.

Samples are drawn from a Logistic distribution with specified
parameters, loc (location or mean, also median), and scale (>0).

Parameters
----------
loc : float

scale : float > 0.

size : {tuple, int}
    Output shape.  If the given shape is, e.g., ‘‘(m, n, k)‘‘, then
    ‘‘m * n * k‘‘ samples are drawn.

Returns
-------
samples : {ndarray, scalar}
          where the values are all integers in  [0, n].

See Also
--------
scipy.stats.distributions.logistic : probability density function,
    distribution or cumulative density function, etc.

Notes
-----
The probability density for the Logistic distribution is

.. math:: P(x) = P(x) = \frac{e^{-(x-\mu)/s}}{s(1+e^{-(x-\mu)/s})^2},

where :math:‘\mu‘ = location and :math:‘s‘ = scale.

The Logistic distribution is used in Extreme Value problems where it
can act as a mixture of Gumbel distributions, in Epidemiology, and by
the World Chess Federation (FIDE) where it is used in the Elo ranking
system, assuming the performance of each player is a logistically
distributed random variable.

References
----------
.. [1] Reiss, R.-D. and Thomas M. (2001), Statistical Analysis of Extreme
       Values, from Insurance, Finance, Hydrology and Other Fields,
       Birkhauser Verlag, Basel, pp 132-133.
.. [2] Weisstein, Eric W. "Logistic Distribution." From
       MathWorld--A Wolfram Web Resource.
       http://mathworld.wolfram.com/LogisticDistribution.html
.. [3] Wikipedia, "Logistic-distribution",
```

lognormal(*mean*=0.0, *sigma*=1.0, *size*=None)

```
Return samples drawn from a log-normal distribution.

Draw samples from a log-normal distribution with specified mean, standard
deviation, and shape. Note that the mean and standard deviation are not the
values for the distribution itself, but of the underlying normal
distribution it is derived from.


Parameters
----------
mean : float
    Mean value of the underlying normal distribution
sigma : float, >0.
    Standard deviation of the underlying normal distribution
size : tuple of ints
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

See Also
--------
scipy.stats.lognorm : probability density function, distribution,
    cumulative density function, etc.

Notes
-----
A variable `x` has a log-normal distribution if `log(x)` is normally
distributed.

The probability density function for the log-normal distribution is

.. math:: p(x) = \frac{1}{\sigma x \sqrt{2\pi}}
                 e^{(-\frac{(ln(x)-\mu)^2}{2\sigma^2})}

where :math:`\mu` is the mean and :math:`\sigma` is the standard deviation
of the normally distributed logarithm of the variable.

A log-normal distribution results if a random variable is the *product* of
a large number of independent, identically-distributed variables in the
same way that a normal distribution results if the variable is the *sum*
of a large number of independent, identically-distributed variables
(see the last example). It is one of the so-called "fat-tailed"
distributions.

The log-normal distribution is commonly used to model the lifespan of units
with fatigue-stress failure modes. Since this includes
most mechanical systems, the log-normal distribution has widespread
application.
```

logseries(*p*, *size*=None)

```
Draw samples from a Logarithmic Series distribution.

Samples are drawn from a Log Series distribution with specified
parameter, p (probability, 0 < p < 1).

Parameters
----------
loc : float

scale : float > 0.

size : {tuple, int}
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Returns
-------
samples : {ndarray, scalar}
          where the values are all integers in  [0, n].

See Also
--------
scipy.stats.distributions.logser : probability density function,
    distribution or cumulative density function, etc.

Notes
-----
The probability density for the Log Series distribution is

.. math:: P(k) = \frac{-p^k}{k \ln(1-p)},

where p = probability.

The Log Series distribution is frequently used to represent species
richness and occurrence, first proposed by Fisher, Corbet, and
Williams in 1943 [2].  It may also be used to model the numbers of
occupants seen in cars [3].

References
----------
.. [1] Buzas, Martin A.; Culver, Stephen J.,  Understanding regional
       species diversity through the log series distribution of
       occurrences: BIODIVERSITY RESEARCH Diversity & Distributions,
       Volume 5, Number 5, September 1999 , pp. 187-195(9).
.. [2] Fisher, R.A,, A.S. Corbet, and C.B. Williams. 1943. The
       relation between the number of species and the number of
       individuals in a random sample of an animal population.
       Journal of Animal Ecology, 12:42-58.
```

multinomial(*n*, *pvals*, *size*=None)

```
Draw samples from a multinomial distribution.

The multinomial distribution is a multivariate generalisation of the
binomial distribution.  Take an experiment with one of ``p``
possible outcomes.  An example of such an experiment is throwing a dice,
where the outcome can be 1 through 6.  Each sample drawn from the
distribution represents `n` such experiments.  Its values,
``X_i = [X_0, X_1, ..., X_p]``, represent the number of times the outcome
was ``i``.

Parameters
----------
n : int
    Number of experiments.
pvals : sequence of floats, length p
    Probabilities of each of the ``p`` different outcomes.  These
    should sum to 1 (however, the last element is always assumed to
    account for the remaining probability, as long as
    ``sum(pvals[:-1]) <= 1)``.
size : tuple of ints
    Given a `size` of ``(M, N, K)``, then ``M*N*K`` samples are drawn,
    and the output shape becomes ``(M, N, K, p)``, since each sample
    has shape ``(p,)``.

Examples
--------
Throw a dice 20 times:

>>> np.random.multinomial(20, [1/6.]*6, size=1)
array([[4, 1, 7, 5, 2, 1]])

It landed 4 times on 1, once on 2, etc.

Now, throw the dice 20 times, and 20 times again:

>>> np.random.multinomial(20, [1/6.]*6, size=2)
array([[3, 4, 3, 3, 4, 3],
       [2, 4, 3, 4, 0, 7]])

For the first run, we threw 3 times 1, 4 times 2, etc.  For the second,
we threw 2 times 1, 4 times 2, etc.

A loaded dice is more likely to land on number 6:

>>> np.random.multinomial(100, [1/7.]*5)
array([13, 16, 13, 16, 42])
```

multivariate_normal(*mean*, *cov*, *size*=...)

```
Draw random samples from a multivariate normal distribution.

The multivariate normal, multinormal or Gaussian distribution is a
generalization of the one-dimensional normal distribution to higher
dimensions.  Such a distribution is specified by its mean and
covariance matrix.  These parameters are analogous to the mean
(average or "center") and variance (standard deviation, or "width,"
squared) of the one-dimensional normal distribution.

Parameters
----------
mean : 1-D array_like, of length N
    Mean of the N-dimensional distribution.
cov : 2-D array_like, of shape (N, N)
    Covariance matrix of the distribution.  Must be symmetric and
    positive semi-definite for "physically meaningful" results.
size : tuple of ints, optional
    Given a shape of, for example, ``(m,n,k)``, ``m*n*k`` samples are
    generated, and packed in an `m`-by-`n`-by-`k` arrangement.  Because
    each sample is `N`-dimensional, the output shape is ``(m,n,k,N)``.
    If no shape is specified, a single (`N`-D) sample is returned.

Returns
-------
out : ndarray
    The drawn samples, of shape *size*, if that was provided.  If not,
    the shape is ``(N,)``.

    In other words, each entry ``out[i,j,...,:]`` is an N-dimensional
    value drawn from the distribution.

Notes
-----
The mean is a coordinate in N-dimensional space, which represents the
location where samples are most likely to be generated.  This is
analogous to the peak of the bell curve for the one-dimensional or
univariate normal distribution.

Covariance indicates the level to which two variables vary together.
From the multivariate normal distribution, we draw N-dimensional
samples, :math:`X = [x_1, x_2, ... x_N]`.  The covariance matrix
element :math:`C_{ij}` is the covariance of :math:`x_i` and :math:`x_j`.
The element :math:`C_{ii}` is the variance of :math:`x_i` (i.e. its
"spread").
```



```
Instead of specifying the full covariance matrix, popular
approximations include:
```

negative_binomial(*n*, *p*, *size*=None)

```
Draw samples from a negative_binomial distribution.

Samples are drawn from a negative_Binomial distribution with specified
parameters, `n` trials and `p` probability of success where `n` is an
integer > 0 and `p` is in the interval [0, 1].

Parameters
----------
n : int
    Parameter, > 0.
p : float
    Parameter, >= 0 and <=1.
size : int or tuple of ints
    Output shape. If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Returns
-------
samples : int or ndarray of ints
    Drawn samples.

Notes
-----
The probability density for the Negative Binomial distribution is

.. math:: P(N;n,p) = \binom{N+n-1}{n-1}p^{n}(1-p)^{N},

where :math:`n-1` is the number of successes, :math:`p` is the probability
of success, and :math:`N+n-1` is the number of trials.

The negative binomial distribution gives the probability of n-1 successes
and N failures in N+n-1 trials, and success on the (N+n)th trial.

If one throws a die repeatedly until the third time a "1" appears, then the
probability distribution of the number of non-"1"s that appear before the
third "1" is a negative binomial distribution.

References
----------
.. [1] Weisstein, Eric W. "Negative Binomial Distribution." From
       MathWorld--A Wolfram Web Resource.
       http://mathworld.wolfram.com/NegativeBinomialDistribution.html
.. [2] Wikipedia, "Negative binomial distribution",
       http://en.wikipedia.org/wiki/Negative_binomial_distribution

Examples
--------
Draw samples from the distribution:
```

noncentral_chisquare(*df*, *nonc*, *size*=None)

```
Draw samples from a noncentral chi-square distribution.

The noncentral :math:`\chi^2` distribution is a generalisation of
the :math:`\chi^2` distribution.

Parameters
----------
df : int
    Degrees of freedom, should be >= 1.
nonc : float
    Non-centrality, should be > 0.
size : int or tuple of ints
    Shape of the output.

Notes
-----
The probability density function for the noncentral Chi-square distribution
is

.. math:: P(x;df,nonc) = \sum^{\infty}_{i=0}
                       \frac{e^{-nonc/2}(nonc/2)^{i}}{i!}P_{Y_{df+2i}}(x),

where :math:`Y_{q}` is the Chi-square with q degrees of freedom.

In Delhi (2007), it is noted that the noncentral chi-square is useful in
bombing and coverage problems, the probability of killing the point target
given by the noncentral chi-squared distribution.

References
----------
.. [1] Delhi, M.S. Holla, "On a noncentral chi-square distribution in the
       analysis of weapon systems effectiveness", Metrika, Volume 15,
       Number 1 / December, 1970.
.. [2] Wikipedia, "Noncentral chi-square distribution"
       http://en.wikipedia.org/wiki/Noncentral_chi-square_distribution

Examples
--------
Draw values from the distribution and plot the histogram

>>> import matplotlib.pyplot as plt
>>> values = plt.hist(np.random.noncentral_chisquare(3, 20, 100000),
...                   bins=200, normed=True)
>>> plt.show()
```



```
Draw values from a noncentral chisquare with very small noncentrality,
and compare to a chisquare.
```

noncentral_f(*dfnum*, *dfden*, *nonc*, *size*=None)

```
Draw samples from the noncentral F distribution.

Samples are drawn from an F distribution with specified parameters,
`dfnum` (degrees of freedom in numerator) and `dfden` (degrees of
freedom in denominator), where both parameters > 1.
`nonc` is the non-centrality parameter.

Parameters
----------
dfnum : int
    Parameter, should be > 1.
dfden : int
    Parameter, should be > 1.
nonc : float
    Parameter, should be >= 0.
size : int or tuple of ints
    Output shape. If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Returns
-------
samples : scalar or ndarray
    Drawn samples.

Notes
-----
When calculating the power of an experiment (power = probability of
rejecting the null hypothesis when a specific alternative is true) the
non-central F statistic becomes important.  When the null hypothesis is
true, the F statistic follows a central F distribution. When the null
hypothesis is not true, then it follows a non-central F statistic.

References
----------
Weisstein, Eric W. "Noncentral F-Distribution." From MathWorld--A Wolfram
Web Resource.  http://mathworld.wolfram.com/NoncentralF-Distribution.html

Wikipedia, "Noncentral F distribution",
http://en.wikipedia.org/wiki/Noncentral_F-distribution

Examples
--------
In a study, testing for a specific alternative to the null hypothesis
requires use of the Noncentral F distribution. We need to calculate the
area in the tail of the distribution that exceeds the value of the F
distribution for the null hypothesis.  We'll plot the two probability
distributions for comparison.
```

normal(*loc*=0.0, *scale*=1.0, *size*=None)

```
Draw random samples from a normal (Gaussian) distribution.

The probability density function of the normal distribution, first
derived by De Moivre and 200 years later by both Gauss and Laplace
independently [2]_, is often called the bell curve because of
its characteristic shape (see the example below).

The normal distributions occurs often in nature.  For example, it
describes the commonly occurring distribution of samples influenced
by a large number of tiny, random disturbances, each with its own
unique distribution [2]_.

Parameters
----------
loc : float
    Mean ("centre") of the distribution.
scale : float
    Standard deviation (spread or "width") of the distribution.
size : tuple of ints
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

See Also
--------
scipy.stats.distributions.norm : probability density function,
    distribution or cumulative density function, etc.

Notes
-----
The probability density for the Gaussian distribution is

.. math:: p(x) = \frac{1}{\sqrt{ 2 \pi \sigma^2 }}
                 e^{ - \frac{ (x - \mu)^2 } {2 \sigma^2} },

where :math:`\mu` is the mean and :math:`\sigma` the standard deviation.
The square of the standard deviation, :math:`\sigma^2`, is called the
variance.

The function has its peak at the mean, and its "spread" increases with
the standard deviation (the function reaches 0.607 times its maximum at
:math:`x + \sigma` and :math:`x - \sigma` [2]_).  This implies that
`numpy.random.normal` is more likely to return samples lying close to the
mean, rather than those far away.
```



```
References
----------
.. [1] Wikipedia, "Normal distribution",
       http://en.wikipedia.org/wiki/Normal_distribution
```

pareto(*a*, *size*=None)

```
Draw samples from a Pareto II or Lomax distribution with specified shape.

The Lomax or Pareto II distribution is a shifted Pareto distribution. The
classical Pareto distribution can be obtained from the Lomax distribution
by adding the location parameter m, see below. The smallest value of the
Lomax distribution is zero while for the classical Pareto distribution it
is m, where the standard Pareto distribution has location m=1.
Lomax can also be considered as a simplified version of the Generalized
Pareto distribution (available in SciPy), with the scale set to one and
the location set to zero.

The Pareto distribution must be greater than zero, and is unbounded above.
It is also known as the "80-20 rule".  In this distribution, 80 percent of
the weights are in the lowest 20 percent of the range, while the other 20
percent fill the remaining 80 percent of the range.

Parameters
----------
shape : float, > 0.
    Shape of the distribution.
size : tuple of ints
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

See Also
--------
scipy.stats.distributions.lomax.pdf : probability density function,
    distribution or cumulative density function, etc.
scipy.stats.distributions.genpareto.pdf : probability density function,
    distribution or cumulative density function, etc.

Notes
-----
The probability density for the Pareto distribution is

.. math:: p(x) = \frac{am^a}{x^{a+1}}

where :math:`a` is the shape and :math:`m` the location

The Pareto distribution, named after the Italian economist Vilfredo Pareto,
is a power law probability distribution useful in many real world problems.
Outside the field of economics it is generally referred to as the Bradford
distribution. Pareto developed the distribution to describe the
distribution of wealth in an economy.  It has also found use in insurance,
web page access statistics, oil field sizes, and many other problems,
including the download frequency for projects in Sourceforge [1].  It is
one of the so-called "fat-tailed" distributions.
```

permutation(x)

```
Randomly permute a sequence, or return a permuted range.

If `x` is a multi-dimensional array, it is only shuffled along its
first index.

Parameters
----------
x : int or array_like
    If `x` is an integer, randomly permute ``np.arange(x)``.
    If `x` is an array, make a copy and shuffle the elements
    randomly.

Returns
-------
out : ndarray
    Permuted sequence or array range.

Examples
--------
>>> np.random.permutation(10)
array([1, 7, 4, 3, 0, 9, 2, 5, 8, 6])

>>> np.random.permutation([1, 4, 9, 12, 15])
array([15,  1,  9,  4, 12])

>>> arr = np.arange(9).reshape((3, 3))
>>> np.random.permutation(arr)
array([[6, 7, 8],
       [0, 1, 2],
       [3, 4, 5]])
```

poisson(*lam*=1.0, *size*=None)

```
Draw samples from a Poisson distribution.

The Poisson distribution is the limit of the Binomial
distribution for large N.

Parameters
----------
lam : float
    Expectation of interval, should be >= 0.
size : int or tuple of ints, optional
    Output shape. If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Notes
-----
The Poisson distribution

.. math:: f(k; \lambda)=\frac{\lambda^k e^{-\lambda}}{k!}

For events with an expected separation :math:`\lambda` the Poisson
distribution :math:`f(k; \lambda)` describes the probability of
:math:`k` events occurring within the observed interval :math:`\lambda`.

Because the output is limited to the range of the C long type, a
ValueError is raised when `lam` is within 10 sigma of the maximum
representable value.

References
----------
.. [1] Weisstein, Eric W. "Poisson Distribution." From MathWorld--A Wolfram
       Web Resource. http://mathworld.wolfram.com/PoissonDistribution.html
.. [2] Wikipedia, "Poisson distribution",
   http://en.wikipedia.org/wiki/Poisson_distribution

Examples
--------
Draw samples from the distribution:

>>> import numpy as np
>>> s = np.random.poisson(5, 10000)

Display histogram of the sample:

>>> import matplotlib.pyplot as plt
>>> count, bins, ignored = plt.hist(s, 14, normed=True)
>>> plt.show()
```

power(*a*, *size*=None)

```
Draws samples in [0, 1] from a power distribution with positive
exponent a - 1.

Also known as the power function distribution.

Parameters
----------
a : float
    parameter, > 0
size : tuple of ints
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
            ``m * n * k`` samples are drawn.

Returns
-------
samples : {ndarray, scalar}
    The returned samples lie in [0, 1].

Raises
------
ValueError
    If a<1.

Notes
-----
The probability density function is

.. math:: P(x; a) = ax^{a-1}, 0 \le x \le 1, a>0.

The power function distribution is just the inverse of the Pareto
distribution. It may also be seen as a special case of the Beta
distribution.

It is used, for example, in modeling the over-reporting of insurance
claims.

References
----------
.. [1] Christian Kleiber, Samuel Kotz, "Statistical size distributions
       in economics and actuarial sciences", Wiley, 2003.
.. [2] Heckert, N. A. and Filliben, James J. (2003). NIST Handbook 148:
       Dataplot Reference Manual, Volume 2: Let Subcommands and Library
       Functions", National Institute of Standards and Technology Handbook
       Series, June 2003.
       http://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/powpdf.pdf

Examples
--------
```

rand(*d0, d1, dn, ...*)

```
Random values in a given shape.

Create an array of the given shape and propagate it with
random samples from a uniform distribution
over ``[0, 1)``.

Parameters
----------
d0, d1, ..., dn : int
    Shape of the output.

Returns
-------
out : ndarray, shape ``(d0, d1, ..., dn)``
    Random values.

See Also
--------
random

Notes
-----
This is a convenience function. If you want an interface that
takes a shape-tuple as the first argument, refer to
`random`.

Examples
--------
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618],  #random
       [ 0.37601032,  0.25528411],  #random
       [ 0.49313049,  0.94909878]]) #random
```

randint(*low*, *high*=None, *size*=None)

```
Return random integers from `low` (inclusive) to `high` (exclusive).

Return random integers from the "discrete uniform" distribution in the
"half-open" interval [`low`, `high`). If `high` is None (the default),
then results are from [0, `low`).

Parameters
----------
low : int
    Lowest (signed) integer to be drawn from the distribution (unless
    ``high=None``, in which case this parameter is the *highest* such
    integer).
high : int, optional
    If provided, one above the largest (signed) integer to be drawn
    from the distribution (see above for behavior if ``high=None``).
size : int or tuple of ints, optional
    Output shape. Default is None, in which case a single int is
    returned.

Returns
-------
out : int or ndarray of ints
    `size`-shaped array of random integers from the appropriate
    distribution, or a single such random int if `size` not provided.

See Also
--------
random.random_integers : similar to `randint`, only for the closed
    interval [`low`, `high`], and 1 is the lowest value if `high` is
    omitted. In particular, this other one is the one to use to generate
    uniformly distributed discrete non-integers.

Examples
--------
>>> np.random.randint(2, size=10)
array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0])
>>> np.random.randint(1, size=10)
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

Generate a 2 x 4 array of ints between 0 and 4, inclusive:

>>> np.random.randint(5, size=(2, 4))
array([[4, 0, 2, 1],
       [3, 2, 2, 0]])
```

randn(*d1=. . . , dn=. . . , ...*)

```
Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided,
`randn` generates an array of shape ``(d1, ..., dn)``, filled
with random floats sampled from a univariate "normal" (Gaussian)
distribution of mean 0 and variance 1 (if any of the :math:`d_i` are
floats, they are first converted to integers by truncation). A single
float randomly sampled from the distribution is returned if no
argument is provided.

This is a convenience function.  If you want an interface that takes a
tuple as the first argument, use `numpy.random.standard_normal` instead.

Parameters
----------
d1, ..., dn : `n` ints, optional
    The dimensions of the returned array, should be all positive.

Returns
-------
Z : ndarray or float
    A ``(d1, ..., dn)``-shaped array of floating-point samples from
    the standard normal distribution, or a single such float if
    no parameters were supplied.

See Also
--------
random.standard_normal : Similar, but takes a tuple as its argument.

Notes
-----
For random samples from :math:`N(\mu, \sigma^2)`, use:

``sigma * np.random.randn(...) + mu``

Examples
--------
>>> np.random.randn()
2.1923875335537315 #random

Two-by-four array of samples from N(3, 6.25):

>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

random_integers(*low*, *high*=None, *size*=None)

```
Return random integers between `low` and `high`, inclusive.

Return random integers from the "discrete uniform" distribution in the
closed interval [`low`, `high`].  If `high` is None (the default),
then results are from [1, `low`].

Parameters
----------
low : int
    Lowest (signed) integer to be drawn from the distribution (unless
    ``high=None``, in which case this parameter is the *highest* such
    integer).
high : int, optional
    If provided, the largest (signed) integer to be drawn from the
    distribution (see above for behavior if ``high=None``).
size : int or tuple of ints, optional
    Output shape. Default is None, in which case a single int is returned.

Returns
-------
out : int or ndarray of ints
    `size`-shaped array of random integers from the appropriate
    distribution, or a single such random int if `size` not provided.

See Also
--------
random.randint : Similar to `random_integers`, only for the half-open
    interval [`low`, `high`), and 0 is the lowest value if `high` is
    omitted.

Notes
-----
To sample from N evenly spaced floating-point numbers between a and b,
use::

  a + (b - a) * (np.random.random_integers(N) - 1) / (N - 1.)

Examples
--------
>>> np.random.random_integers(5)
4
>>> type(np.random.random_integers(5))
<type 'int'>
>>> np.random.random_integers(5, size=(3.,2.))
array([[5, 4],
       [3, 3],
       [4, 5]])
```

random_sample(*size*=None)

```
Return random floats in the half-open interval [0.0, 1.0).

Results are from the "continuous uniform" distribution over the
stated interval.  To sample :math:‘Unif[a, b), b > a‘ multiply
the output of ‘random_sample‘ by ‘(b-a)‘ and add ‘a‘::

  (b - a) * random_sample() + a

Parameters
----------
size : int or tuple of ints, optional
    Defines the shape of the returned array of random floats. If None
    (the default), returns a single float.

Returns
-------
out : float or ndarray of floats
    Array of random floats of shape ‘size‘ (unless ‘‘size=None‘‘, in which
    case a single float is returned).

Examples
--------
>>> np.random.random_sample()
0.47108547995356098
>>> type(np.random.random_sample())
<type ’float’>
>>> np.random.random_sample((5,))
array([ 0.30220482,  0.86820401,  0.1654503 ,  0.11659149,  0.54323428])

Three-by-two array of random numbers from [-5, 0):

>>> 5 * np.random.random_sample((3, 2)) - 5
array([[-3.99149989, -0.52338984],
       [-2.99091858, -0.79479508],
       [-1.23204345, -1.75224494]])
```

rayleigh(*scale*=1.0, *size*=None)

```
Draw samples from a Rayleigh distribution.

The :math:`\chi` and Weibull distributions are generalizations of the
Rayleigh.

Parameters
----------
scale : scalar
    Scale, also equals the mode. Should be >= 0.
size : int or tuple of ints, optional
    Shape of the output. Default is None, in which case a single
    value is returned.

Notes
-----
The probability density function for the Rayleigh distribution is

.. math:: P(x;scale) = \frac{x}{scale^2}e^{\frac{-x^2}{2 \cdotp scale^2}}

The Rayleigh distribution arises if the wind speed and wind direction are
both gaussian variables, then the vector wind velocity forms a Rayleigh
distribution. The Rayleigh distribution is used to model the expected
output from wind turbines.

References
----------
..[1] Brighton Webs Ltd., Rayleigh Distribution,
      http://www.brighton-webs.co.uk/distributions/rayleigh.asp
..[2] Wikipedia, "Rayleigh distribution"
      http://en.wikipedia.org/wiki/Rayleigh_distribution

Examples
--------
Draw values from the distribution and plot the histogram

>>> values = hist(np.random.rayleigh(3, 100000), bins=200, normed=True)

Wave heights tend to follow a Rayleigh distribution. If the mean wave
height is 1 meter, what fraction of waves are likely to be larger than 3
meters?

>>> meanvalue = 1
>>> modevalue = np.sqrt(2 / np.pi) * meanvalue
>>> s = np.random.rayleigh(modevalue, 1000000)

The percentage of waves larger than 3 meters is:

>>> 100.*sum(s>3)/1000000.
```

seed(*seed*=None)

```
Seed the generator.

This method is called when ‘RandomState‘ is initialized. It can be
called again to re-seed the generator. For details, see ‘RandomState‘.

Parameters
----------
seed : int or array_like, optional
    Seed for ‘RandomState‘.

See Also
--------
RandomState
```

set_state(*state*)

```
Set the internal state of the generator from a tuple.

For use if one has reason to manually (re-)set the internal state of the
"Mersenne Twister"[1]_ pseudo-random number generating algorithm.

Parameters
----------
state : tuple(str, ndarray of 624 uints, int, int, float)
    The ‘state‘ tuple has the following items:

    1. the string ’MT19937’, specifying the Mersenne Twister algorithm.
    2. a 1-D array of 624 unsigned integers ‘‘keys‘‘.
    3. an integer ‘‘pos‘‘.
    4. an integer ‘‘has_gauss‘‘.
    5. a float ‘‘cached_gaussian‘‘.

Returns
-------
out : None
    Returns ’None’ on success.

See Also
--------
get_state

Notes
-----
‘set_state‘ and ‘get_state‘ are not needed to work with any of the
random distributions in NumPy. If the internal state is manually altered,
the user should know exactly what he/she is doing.

For backwards compatibility, the form (str, array of 624 uints, int) is
also accepted although it is missing some information about the cached
Gaussian value: ‘‘state = (’MT19937’, keys, pos)‘‘.

References
----------
.. [1] M. Matsumoto and T. Nishimura, "Mersenne Twister: A
   623-dimensionally equidistributed uniform pseudorandom number
   generator," *ACM Trans. on Modeling and Computer Simulation*,
   Vol. 8, No. 1, pp. 3-30, Jan. 1998.
```

shuffle(x)

```
Modify a sequence in-place by shuffling its contents.

Parameters
----------
x : array_like
    The array or list to be shuffled.

Returns
-------
None

Examples
--------
>>> arr = np.arange(10)
>>> np.random.shuffle(arr)
>>> arr
[1 7 5 2 9 4 3 6 0 8]

This function only shuffles the array along the first index of a
multi-dimensional array:

>>> arr = np.arange(9).reshape((3, 3))
>>> np.random.shuffle(arr)
>>> arr
array([[3, 4, 5],
       [6, 7, 8],
       [0, 1, 2]])
```

standard_cauchy(*size*=None)

```
Standard Cauchy distribution with mode = 0.

Also known as the Lorentz distribution.

Parameters
----------
size : int or tuple of ints
    Shape of the output.

Returns
-------
samples : ndarray or scalar
    The drawn samples.

Notes
-----
The probability density function for the full Cauchy distribution is

.. math:: P(x; x_0, \gamma) = \frac{1}{\pi \gamma \bigl[ 1+
          (\frac{x-x_0}{\gamma})^2 \bigr] }

and the Standard Cauchy distribution just sets :math:`x_0=0` and
:math:`\gamma=1`

The Cauchy distribution arises in the solution to the driven harmonic
oscillator problem, and also describes spectral line broadening. It
also describes the distribution of values at which a line tilted at
a random angle will cut the x axis.

When studying hypothesis tests that assume normality, seeing how the
tests perform on data from a Cauchy distribution is a good indicator of
their sensitivity to a heavy-tailed distribution, since the Cauchy looks
very much like a Gaussian distribution, but with heavier tails.

References
----------
..[1] NIST/SEMATECH e-Handbook of Statistical Methods, "Cauchy
      Distribution",
      http://www.itl.nist.gov/div898/handbook/eda/section3/eda3663.htm
..[2] Weisstein, Eric W. "Cauchy Distribution." From MathWorld--A
      Wolfram Web Resource.
      http://mathworld.wolfram.com/CauchyDistribution.html
..[3] Wikipedia, "Cauchy distribution"
      http://en.wikipedia.org/wiki/Cauchy_distribution

Examples
--------
Draw samples and plot the distribution:
```

standard_exponential(*size*=None)

```
Draw samples from the standard exponential distribution.

‘standard_exponential‘ is identical to the exponential distribution
with a scale parameter of 1.

Parameters
----------
size : int or tuple of ints
    Shape of the output.

Returns
-------
out : float or ndarray
    Drawn samples.

Examples
--------
Output a 3x8000 array:

>>> n = np.random.standard_exponential((3, 8000))
```

standard_gamma(*shape*, *size*=None)

```
Draw samples from a Standard Gamma distribution.

Samples are drawn from a Gamma distribution with specified parameters,
shape (sometimes designated "k") and scale=1.

Parameters
----------
shape : float
    Parameter, should be > 0.
size : int or tuple of ints
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Returns
-------
samples : ndarray or scalar
    The drawn samples.

See Also
--------
scipy.stats.distributions.gamma : probability density function,
    distribution or cumulative density function, etc.

Notes
-----
The probability density for the Gamma distribution is

.. math:: p(x) = x^{k-1}\frac{e^{-x/\theta}}{\theta^k\Gamma(k)},

where :math:`k` is the shape and :math:`\theta` the scale,
and :math:`\Gamma` is the Gamma function.

The Gamma distribution is often used to model the times to failure of
electronic components, and arises naturally in processes for which the
waiting times between Poisson distributed events are relevant.

References
----------
.. [1] Weisstein, Eric W. "Gamma Distribution." From MathWorld--A
       Wolfram Web Resource.
       http://mathworld.wolfram.com/GammaDistribution.html
.. [2] Wikipedia, "Gamma-distribution",
       http://en.wikipedia.org/wiki/Gamma-distribution
```



```
Examples
--------
Draw samples from the distribution:
```

standard_normal(*size*=None)

```
Returns samples from a Standard Normal distribution (mean=0, stdev=1).

Parameters
----------
size : int or tuple of ints, optional
    Output shape. Default is None, in which case a single value is
    returned.

Returns
-------
out : float or ndarray
    Drawn samples.

Examples
--------
>>> s = np.random.standard_normal(8000)
>>> s
array([ 0.6888893 ,  0.78096262, -0.89086505, ...,  0.49876311, #random
       -0.38672696, -0.4685006 ])                                #random
>>> s.shape
(8000,)
>>> s = np.random.standard_normal(size=(3, 4, 2))
>>> s.shape
(3, 4, 2)
```

standard_t(*df*, *size*=None)

```
Standard Student's t distribution with df degrees of freedom.

A special case of the hyperbolic distribution.
As `df` gets large, the result resembles that of the standard normal
distribution (`standard_normal`).

Parameters
----------
df : int
    Degrees of freedom, should be > 0.
size : int or tuple of ints, optional
    Output shape. Default is None, in which case a single value is
    returned.

Returns
-------
samples : ndarray or scalar
    Drawn samples.

Notes
-----
The probability density function for the t distribution is

.. math:: P(x, df) = \frac{\Gamma(\frac{df+1}{2})}{\sqrt{\pi df}
          \Gamma(\frac{df}{2})}\Bigl( 1+\frac{x^2}{df} \Bigr)^{-(df+1)/2}

The t test is based on an assumption that the data come from a Normal
distribution. The t test provides a way to test whether the sample mean
(that is the mean calculated from the data) is a good estimate of the true
mean.

The derivation of the t-distribution was forst published in 1908 by William
Gisset while working for the Guinness Brewery in Dublin. Due to proprietary
issues, he had to publish under a pseudonym, and so he used the name
Student.

References
----------
.. [1] Dalgaard, Peter, "Introductory Statistics With R",
       Springer, 2002.
.. [2] Wikipedia, "Student's t-distribution"
       http://en.wikipedia.org/wiki/Student's_t-distribution

Examples
--------
From Dalgaard page 83 [1]_, suppose the daily energy intake for 11
women in Kj is:
```

triangular(*left*, *mode*, *right*, *size*=None)

```
Draw samples from the triangular distribution.

The triangular distribution is a continuous probability distribution with
lower limit left, peak at mode, and upper limit right. Unlike the other
distributions, these parameters directly define the shape of the pdf.

Parameters
----------
left : scalar
    Lower limit.
mode : scalar
    The value where the peak of the distribution occurs.
    The value should fulfill the condition ``left <= mode <= right``.
right : scalar
    Upper limit, should be larger than `left`.
size : int or tuple of ints, optional
    Output shape. Default is None, in which case a single value is
    returned.

Returns
-------
samples : ndarray or scalar
    The returned samples all lie in the interval [left, right].

Notes
-----
The probability density function for the Triangular distribution is

.. math:: P(x;l, m, r) = \begin{cases}
          \frac{2(x-l)}{(r-l)(m-l)}& \text{for $l \leq x \leq m$},\\
          \frac{2(m-x)}{(r-l)(r-m)}& \text{for $m \leq x \leq r$},\\
          0& \text{otherwise}.
          \end{cases}

The triangular distribution is often used in ill-defined problems where the
underlying distribution is not known, but some knowledge of the limits and
mode exists. Often it is used in simulations.

References
----------
..[1] Wikipedia, "Triangular distribution"
      http://en.wikipedia.org/wiki/Triangular_distribution

Examples
--------
Draw values from the distribution and plot the histogram:

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

uniform(*low*=0.0, *high*=1.0, *size*=1)

```
Draw samples from a uniform distribution.

Samples are uniformly distributed over the half-open interval
``[low, high)`` (includes low, but excludes high).  In other words,
any value within the given interval is equally likely to be drawn
by `uniform`.

Parameters
----------
low : float, optional
    Lower boundary of the output interval.  All values generated will be
    greater than or equal to low.  The default value is 0.
high : float
    Upper boundary of the output interval.  All values generated will be
    less than high.  The default value is 1.0.
size : int or tuple of ints, optional
    Shape of output.  If the given size is, for example, (m,n,k),
    m*n*k samples are generated.  If no shape is specified, a single sample
    is returned.

Returns
-------
out : ndarray
    Drawn samples, with shape `size`.

See Also
--------
randint : Discrete uniform distribution, yielding integers.
random_integers : Discrete uniform distribution over the closed
                  interval ``[low, high]``.
random_sample : Floats uniformly distributed over ``[0, 1)``.
random : Alias for `random_sample`.
rand : Convenience function that accepts dimensions as input, e.g.,
       ``rand(2,2)`` would generate a 2-by-2 array of floats,
       uniformly distributed over ``[0, 1)``.

Notes
-----
The probability density function of the uniform distribution is

.. math:: p(x) = \frac{1}{b - a}

anywhere within the interval ``[a, b)``, and zero elsewhere.
```



```
Examples
--------
Draw samples from the distribution:
```

vonmises(*mu*, *kappa*, *size*=None)

```
Draw samples from a von Mises distribution.

Samples are drawn from a von Mises distribution with specified mode
(mu) and dispersion (kappa), on the interval [-pi, pi].

The von Mises distribution (also known as the circular normal
distribution) is a continuous probability distribution on the unit
circle.  It may be thought of as the circular analogue of the normal
distribution.

Parameters
----------
mu : float
    Mode ("center") of the distribution.
kappa : float
    Dispersion of the distribution, has to be >=0.
size : int or tuple of int
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

Returns
-------
samples : scalar or ndarray
    The returned samples, which are in the interval [-pi, pi].

See Also
--------
scipy.stats.distributions.vonmises : probability density function,
    distribution, or cumulative density function, etc.

Notes
-----
The probability density for the von Mises distribution is

.. math:: p(x) = \frac{e^{\kappa cos(x-\mu)}}{2\pi I_0(\kappa)},

where :math:`\mu` is the mode and :math:`\kappa` the dispersion,
and :math:`I_0(\kappa)` is the modified Bessel function of order 0.

The von Mises is named for Richard Edler von Mises, who was born in
Austria-Hungary, in what is now the Ukraine.  He fled to the United
States in 1939 and became a professor at Harvard.  He worked in
probability theory, aerodynamics, fluid mechanics, and philosophy of
science.
```



```
References
----------
Abramowitz, M. and Stegun, I. A. (ed.), *Handbook of Mathematical
```

wald(*mean*, *scale*, *size*=None)

```
Draw samples from a Wald, or Inverse Gaussian, distribution.

As the scale approaches infinity, the distribution becomes more like a
Gaussian.

Some references claim that the Wald is an Inverse Gaussian with mean=1, but
this is by no means universal.

The Inverse Gaussian distribution was first studied in relationship to
Brownian motion. In 1956 M.C.K. Tweedie used the name Inverse Gaussian
because there is an inverse relationship between the time to cover a unit
distance and distance covered in unit time.

Parameters
----------
mean : scalar
    Distribution mean, should be > 0.
scale : scalar
    Scale parameter, should be >= 0.
size : int or tuple of ints, optional
    Output shape. Default is None, in which case a single value is
    returned.

Returns
-------
samples : ndarray or scalar
    Drawn sample, all greater than zero.

Notes
-----
The probability density function for the Wald distribution is

.. math:: P(x;mean,scale) = \sqrt{\frac{scale}{2\pi x^3}}e^
                            \frac{-scale(x-mean)^2}{2\cdotp mean^2x}

As noted above the Inverse Gaussian distribution first arise from attempts
to model Brownian Motion. It is also a competitor to the Weibull for use in
reliability modeling and modeling stock returns and interest rate
processes.

References
----------
..[1] Brighton Webs Ltd., Wald Distribution,
      http://www.brighton-webs.co.uk/distributions/wald.asp
..[2] Chhikara, Raj S., and Folks, J. Leroy, "The Inverse Gaussian
      Distribution: Theory : Methodology, and Applications", CRC Press,
      1988.
..[3] Wikipedia, "Wald distribution"
```

weibull(*a*, *size*=None)

```
Weibull distribution.

Draw samples from a 1-parameter Weibull distribution with the given
shape parameter `a`.

.. math:: X = (-ln(U))^{1/a}

Here, U is drawn from the uniform distribution over (0,1].

The more common 2-parameter Weibull, including a scale parameter
:math:`\lambda` is just :math:`X = \lambda(-ln(U))^{1/a}`.

Parameters
----------
a : float
    Shape of the distribution.
size : tuple of ints
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.

See Also
--------
scipy.stats.distributions.weibull : probability density function,
    distribution or cumulative density function, etc.

gumbel, scipy.stats.distributions.genextreme

Notes
-----
The Weibull (or Type III asymptotic extreme value distribution for smallest
values, SEV Type III, or Rosin-Rammler distribution) is one of a class of
Generalized Extreme Value (GEV) distributions used in modeling extreme
value problems.  This class includes the Gumbel and Frechet distributions.

The probability density for the Weibull distribution is

.. math:: p(x) = \frac{a}
                 {\lambda}(\frac{x}{\lambda})^{a-1}e^{-(x/\lambda)^a},

where :math:`a` is the shape and :math:`\lambda` the scale.

The function has its peak (the mode) at
:math:`\lambda(\frac{a-1}{a})^{1/a}`.
```



```
When ``a = 1``, the Weibull distribution reduces to the exponential
distribution.

References
```

zipf(*a*, *size*=None)

```
Draw samples from a Zipf distribution.

Samples are drawn from a Zipf distribution with specified parameter
`a` > 1.

The Zipf distribution (also known as the zeta distribution) is a
continuous probability distribution that satisfies Zipf's law: the
frequency of an item is inversely proportional to its rank in a
frequency table.

Parameters
----------
a : float > 1
    Distribution parameter.
size : int or tuple of int, optional
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn; a single integer is equivalent in
    its result to providing a mono-tuple, i.e., a 1-D array of length
    *size* is returned.  The default is None, in which case a single
    scalar is returned.

Returns
-------
samples : scalar or ndarray
    The returned samples are greater than or equal to one.

See Also
--------
scipy.stats.distributions.zipf : probability density function,
    distribution, or cumulative density function, etc.

Notes
-----
The probability density for the Zipf distribution is

.. math:: p(x) = \frac{x^{-a}}{\zeta(a)},

where :math:`\zeta` is the Riemann Zeta function.

It is named for the American linguist George Kingsley Zipf, who noted
that the frequency of any word in a sample of a language is inversely
proportional to its rank in the frequency table.

References
----------
Zipf, G. K., *Selected Studies of the Principle of Relative Frequency
in Language*, Cambridge, MA: Harvard Univ. Press, 1932.
```

2.2 Variables

Name	Description
ALLOW_THREADS	**Value:** 1
BUFSIZE	**Value:** 8192
CLIP	**Value:** 0
ERR_CALL	**Value:** 3
ERR_DEFAULT	**Value:** 0
ERR_DEFAULT2	**Value:** 521
ERR_IGNORE	**Value:** 0
ERR_LOG	**Value:** 5
ERR_PRINT	**Value:** 4
ERR_RAISE	**Value:** 2
ERR_WARN	**Value:** 1
FLOATING_POINT_SUPPORT	**Value:** 1
FPE_DIVIDEBYZERO	**Value:** 1
FPE_INVALID	**Value:** 8
FPE_OVERFLOW	**Value:** 2
FPE_UNDERFLOW	**Value:** 4
False_	**Value:** False
Inf	**Value:** inf
Infinity	**Value:** inf
MAXDIMS	**Value:** 32
NAN	**Value:** nan
NINF	**Value:** -inf
NZERO	**Value:** -0.0
NaN	**Value:** nan
PINF	**Value:** inf
PZERO	**Value:** 0.0
RAISE	**Value:** 2
SHIFT_DIVIDEBYZERO	**Value:** 0
SHIFT_INVALID	**Value:** 9
SHIFT_OVERFLOW	**Value:** 3
SHIFT_UNDERFLOW	**Value:** 6
ScalarType	**Value:** (<type ’int’>, <type ’float’>, <type ’complex’>, <type ’l...
True_	**Value:** True
UFUNC_BUFSIZE_DEFAULT	**Value:** 8192
UFUNC_PYVALS_NAME	**Value:** ’UFUNC_PYVALS’
WRAP	**Value:** 1

continued on next page

Name	Description
__package__	**Value:** None
absolute	**Value:** <ufunc ’absolute’>
add	**Value:** <ufunc ’add’>
arccos	**Value:** <ufunc ’arccos’>
arccosh	**Value:** <ufunc ’arccosh’>
arcsin	**Value:** <ufunc ’arcsin’>
arcsinh	**Value:** <ufunc ’arcsinh’>
arctan	**Value:** <ufunc ’arctan’>
arctan2	**Value:** <ufunc ’arctan2’>
arctanh	**Value:** <ufunc ’arctanh’>
bitwise_and	**Value:** <ufunc ’bitwise_and’>
bitwise_not	**Value:** <ufunc ’invert’>
bitwise_or	**Value:** <ufunc ’bitwise_or’>
bitwise_xor	**Value:** <ufunc ’bitwise_xor’>
c_	**Value:** <numpy.lib.index_tricks.CClass object at 0x930204c>
cast	**Value:** {<type ’numpy.uint32’>: <function <lambda> at 0x9176a74>,...
ceil	**Value:** <ufunc ’ceil’>
conj	**Value:** <ufunc ’conjugate’>
conjugate	**Value:** <ufunc ’conjugate’>
copysign	**Value:** <ufunc ’copysign’>
cos	**Value:** <ufunc ’cos’>
cosh	**Value:** <ufunc ’cosh’>
deg2rad	**Value:** <ufunc ’deg2rad’>
degrees	**Value:** <ufunc ’degrees’>
divide	**Value:** <ufunc ’divide’>
e	**Value:** 2.71828182846
equal	**Value:** <ufunc ’equal’>
exp	**Value:** <ufunc ’exp’>
exp2	**Value:** <ufunc ’exp2’>
expm1	**Value:** <ufunc ’expm1’>
fabs	**Value:** <ufunc ’fabs’>
floor	**Value:** <ufunc ’floor’>
floor_divide	**Value:** <ufunc ’floor_divide’>
fmax	**Value:** <ufunc ’fmax’>
fmin	**Value:** <ufunc ’fmin’>
fmod	**Value:** <ufunc ’fmod’>
frexp	**Value:** <ufunc ’frexp’>
greater	**Value:** <ufunc ’greater’>
greater_equal	**Value:** <ufunc ’greater_equal’>
hypot	**Value:** <ufunc ’hypot’>

continued on next page

Name	Description
index_exp	**Value:** <numpy.lib.index_tricks.IndexExpression object at 0x930214c>
inf	**Value:** inf
infty	**Value:** inf
invert	**Value:** <ufunc ’invert’>
isfinite	**Value:** <ufunc ’isfinite’>
isinf	**Value:** <ufunc ’isinf’>
isnan	**Value:** <ufunc ’isnan’>
ldexp	**Value:** <ufunc ’ldexp’>
left_shift	**Value:** <ufunc ’left_shift’>
less	**Value:** <ufunc ’less’>
less_equal	**Value:** <ufunc ’less_equal’>
little_endian	**Value:** True
log	**Value:** <ufunc ’log’>
log10	**Value:** <ufunc ’log10’>
log1p	**Value:** <ufunc ’log1p’>
logaddexp	**Value:** <ufunc ’logaddexp’>
logaddexp2	**Value:** <ufunc ’logaddexp2’>
logical_and	**Value:** <ufunc ’logical_and’>
logical_not	**Value:** <ufunc ’logical_not’>
logical_or	**Value:** <ufunc ’logical_or’>
logical_xor	**Value:** <ufunc ’logical_xor’>
maximum	**Value:** <ufunc ’maximum’>
mgrid	**Value:** <numpy.lib.index_tricks.nd_grid object at 0x92fef8c>
minimum	**Value:** <ufunc ’minimum’>
mod	**Value:** <ufunc ’remainder’>
modf	**Value:** <ufunc ’modf’>
multiply	**Value:** <ufunc ’multiply’>
nan	**Value:** nan
nbytes	**Value:** {<type ’numpy.uint32’>: 4, <type ’numpy.bool_’>: 1, <type...
negative	**Value:** <ufunc ’negative’>
newaxis	**Value:** None
nextafter	**Value:** <ufunc ’nextafter’>
not_equal	**Value:** <ufunc ’not_equal’>
ogrid	**Value:** <numpy.lib.index_tricks.nd_grid object at 0x92fefac>
ones_like	**Value:** <ufunc ’ones_like’>
pi	**Value:** 3.14159265359
r_	**Value:** <numpy.lib.index_tricks.RClass object at 0x930202c>

continued on next page

Name	Description
rad2deg	**Value:** `<ufunc ’rad2deg’>`
radians	**Value:** `<ufunc ’radians’>`
reciprocal	**Value:** `<ufunc ’reciprocal’>`
remainder	**Value:** `<ufunc ’remainder’>`
right_shift	**Value:** `<ufunc ’right_shift’>`
rint	**Value:** `<ufunc ’rint’>`
s_	**Value:** `<numpy.lib.index_tricks.IndexExpression object at 0x93021ac>`
sctypeDict	**Value:** `{0: <type ’numpy.bool_’>, 1: <type ’numpy.int8’>, 2: <typ...`
sctypeNA	**Value:** `{’?’: ’Bool’, ’B’: ’UInt8’, ’Bool’: <type ’numpy.bool_’>,...`
sctypes	**Value:** `{’complex’: [<type ’numpy.complex64’>, <type ’numpy.compl...`
sign	**Value:** `<ufunc ’sign’>`
signbit	**Value:** `<ufunc ’signbit’>`
sin	**Value:** `<ufunc ’sin’>`
sinh	**Value:** `<ufunc ’sinh’>`
spacing	**Value:** `<ufunc ’spacing’>`
sqrt	**Value:** `<ufunc ’sqrt’>`
square	**Value:** `<ufunc ’square’>`
subtract	**Value:** `<ufunc ’subtract’>`
tan	**Value:** `<ufunc ’tan’>`
tanh	**Value:** `<ufunc ’tanh’>`
true_divide	**Value:** `<ufunc ’true_divide’>`
trunc	**Value:** `<ufunc ’trunc’>`
typeDict	**Value:** `{0: <type ’numpy.bool_’>, 1: <type ’numpy.int8’>, 2: <typ...`
typeNA	**Value:** `{’?’: ’Bool’, ’B’: ’UInt8’, ’Bool’: <type ’numpy.bool_’>,...`
typecodes	**Value:** `{’All’: ’?bhilqpBHILQPefdgFDGSUVOMm’, ’AllFloat’: ’efdgFD...`

2.3 Class bcolors

bcolors permite colorir o output gerado no terminal por forma a ser mais facil de interpertar

2.3.1 Methods

disable(*self*)

desativa qualquer cor que tiver ativa no terminal

2.3.2 Class Variables

Name	Description
ROXO	**Value:** `'\x1b[95m\x1b[1m\x1b[40m'`
AZUL	**Value:** `'\x1b[94m\x1b[1m\x1b[40m'`
VERDE	**Value:** `'\x1b[92m\x1b[1m\x1b[40m'`
AMARELO	**Value:** `'\x1b[93m\x1b[1m\x1b[40m'`
VERMELHO	**Value:** `'\x1b[91m\x1b[1m\x1b[40m'`
ENDC	**Value:** `'\x1b[0m'`

2.4 Class FicheiroLog

Classe de análise de ficheiro de logs

2.4.1 Methods

__init__(*self*, *caminhoFileLog*)

Inicializa a lista onde guardrá os objectos analizado e chama o metodo que fará a analise

Parametro: caminho para o ficheiro

analiseFicheiroLog(*self*, *caminho*, *caminhoFileLog*)

Método que faz a análise do Ficheiro de log Recebe como parametro o caminho do file GeoIP.dat

extraMenu(*self*)

Menu com opções extra, tais como imprimir a informação em PDF, CSV entre outros.

Este menu será apresentado no fim de cada acção

infoForGraph(*self*)

Constroi dicionario com informação para geração do gráfico estatistico

3 Module executarApp

Created on 11 de Abr de 2013

Author: admin1

3.1 Variables

Name	Description
__package__	**Value:** `None`

3.2 Class Dialgo

object
executarApp.Dialgo

Classe responsável por gerár o primeiro menu de dialogos que será a porta de interacção com o utilizador

3.2.1 Methods

__init__(*self*)

Construtor da classe que nao vai desenhpenhar nenhuma função

Overrides: object.__init__

dIncial(*self*)

Responsavel por mostrar a caixa de menus ao utilizador e devolver a resposta que o utilizador introduzio

validarDialgo(*self*)

Consoante a resposta do utilizador assim será a acção que será desencadeada.

O utilizador terá ao dispor 4 opções.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.2.2 Properties

Name	Description
Inherited from object	
__class__	

4 Module file

4.1 Variables

Name	Description
__package__	**Value:** `None`

5 Module nmap_portScanning

5.1 Variables

Name	Description
___package___	**Value:** `None`

5.2 Class PortScanning

object ─┐
nmap_portScanning.PortScanning

Realiza portscanning a todas as maquinas de uma rede.

5.2.1 Methods

___init___(*self*)

Pergunta ao utilizador qual o ip da rede, e a mascara

depois de os dados introduzidos será feito o scan

Overrides: object.___init___

makeScan(*self*)

Realiza o portScanning com base na informação introduzida pelo utilizador. Imprimindo depois o resultado na consola

extraMenu(*self*)

Menu com opções extra, tais como imprimir a informação em PDF, CSV entre outros.

Este menu será apresentado no fim de cada acção

analiseIP(*self*, *ip*)

Analisa se o ip é valido utilizando metodo construido em C

analiseMask(*self*, *mask*)

Analisa se a mascara é correcta

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

5.2.2 Properties

Name	Description
Inherited from object	
___class___	

6 Module nmap_scanningConections

Created on 15 de Abr de 2013

Author: xama

6.1 Variables

Name	Description
___package___	**Value:** `None`

6.2 Class ScanningConections

object ─┐
nmap_scanningConections.ScanningConections

Realiza o scan das conexões ativas de uma maquina.

6.2.1 Methods

___init___(*self*, **args*, ***kwargs*)

Pergunta ao utilizador qual o ip da maquina

depois de os dados introduzidos será feito o scan

Overrides: object.___init___

makeScan(*self*)

Realiza o scan com base na informação introduzida pelo utilizador. Imprimindo depois o resultado na consola

extraMenu(*self*)

Menu com opções extra, tais como imprimir a informação em PDF, CSV entre outros.

Este menu será apresentado no fim de cada acção

analiseIP(*self*, *ip*)

Analisa se o ip é valido utilizando metodo construido em C

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

6.2.2 Properties

Name	Description
Inherited from object	
__class__	

Index