# EML4930/EML6934: Lecture 07

Statistical distributions and functions with scipy.stats

Charles Jekel

October 12, 2017

## Supplementary textbook for statistics in Python

Freely available.

Think Stats by Allen B. Downey. Exploratory Data Analysis in Python.

`http://greenteapress.com/wp/think-stats-2e/`

## Statistics with scipy.stats

Today we're going to discuss many features of scipy.stats

The full documentation is available:
`https://docs.scipy.org/doc/scipy/reference/stats.html`

Part of this lecture will cover the tutorial here: `https://docs.scipy.org/doc/scipy/reference/tutorial/stats.html`

## What will we cover today

- Random variable distributions
- Continuous and discrete distributions
- PDF, CDF, SF, and other associated functions of distributions
- Maximum likelihood estimation to fit distributions to data
- Hypothesis test: are two samples from the same distribution?
- Hypothesis test: are these samples from this distribution?

## Reminder of viewing documentation

In Python the documentation is included with the library.

```
from scipy import stats
# documentation of the namespace one screen at a time
stats?
# print the entire documentation
print(stats.__doc__)
# Don't forget the help function
help(stats)
# for a particular function
stats.norm?
# or
print(stats.norm.__doc__)
# __doc__ is a built in attribute created from the
# comments following your function definition!
# view all of the contents in scipy.stats
dir(stats)
```

## scipy.stats at a glance

- 95 continuous distributions
- 13 discrete distributions
- 9 multivariate distributions
- A number of other statistical functions and methods

## Common methods for continuous random variable distributions

| Method | Description |
|---|---|
| rvs | Random Variates |
| pdf | Probability Density Function |
| cdf | Cumulative Distribution Function |
| sf | Survival Function (1-CDF) |
| ppf | Percent Point Function (Inverse of CDF) |
| isf | Inverse Survival Function (Inverse of SF) |
| stats | Return mean, variance, (Fishers) skew, or (Fishers) kurtosis |
| moment | non-central moments of the distribution |

## scipy.stats namespace

```python
from scipy import stats

# the namespace follows:
# scipy.stats.distribution.method()

# demonstrating how to access methods of the
# Maxwell continuous distribution
stats.maxwell.rvs()
stats.maxwell.pdf(1)
stats.maxwell.cdf(1)
stats.maxwell.sf(1)
stats.maxwell.ppf(0.5)
```

## It’s safe to import the distributions themselves

If you’re only going to use a few functions and distributions, it may make sense to import them directly.

```
from scipy.stats import norm
from scipy.stats import binom
from scipy.stats import poisson

# now we can call the distributions directly
norm.pdf(0.9)

binom.cdf(8,n=100,p=0.1)

poisson.rvs(1)
```

## Basic methods take vectorized input

Basic methods such as pdf, and cdf are vectorized with np.vectorize.

pdf(0.5) will calculate the pdf at 0.5

cdf(0.5) will calculate the cdf at 0.5

However often we want to calculate the pdf and cdf at more than one value. For these cases we pass a numpy array of values into the functions.

pdf(x) will return an array of the same shape x of the pdf value of each item in x

## Let’s take a look at the normal distribution

```
scipy.stats.norm
```

scipy.stats._continuous_distns.norm_gen object A normal continuous random variable.

The location (loc) keyword specifies the mean. The scale (scale) keyword specifies the standard deviation.

```
import matplotlib.pyplot as plt
x = np.linspace(-3,3,100)
# the default normal distribution has a mean of 0
# and standard deviation of 1
y = norm.pdf(x)
plt.figure()
plt.plot(x,y)
plt.show()
```

## We can specify the mean and standard deviation

```
mu = 2.0
std = 2.0
x = np.linspace(-4.0,8.0,100)
y = norm.pdf(x, mu, std)
z = norm.cdf(x, mu, std)
w = norm.sf(x, mu, std)

# plot the pdf, cdf, and sf
plt.figure()
plt.plot(x,y)
plt.plot(x,z)
plt.plot(x,w)
plt.show()
```

## Digression: Meaning of PDF, CDF, and SF

These are for continuous random variable $x$.

PDF - probability density function

PDF tells you the probability of a particular random variate occurring.
$\mathrm{P}(x = 0.5) = \mathrm{PDF}(0.5)$

CDF - cumulative distribution function

CDF tells you the probability of a particular random variate being less than or equal to a value.

$\mathrm{P}(-\infty < t \leq m) = \mathrm{CDF}(m) = \int_{-\infty}^{m} PDF(t)dt$

$\mathrm{P}(-\infty < x \leq 0.5) = \mathrm{CDF}(0.5)$

SF - survival function

SF tells you the probability of a particular random variate being greater than a value.

$\mathrm{P}(x > 0.5) = \mathrm{SF}(0.5) = 1 - \mathrm{CDF}(0.5)$

## Back to the normal distribution

It's tedious to always specify the $mu, \sigma$ for each function. So what we do is create a *Frozen* instance of the object. You won't be able to change the $mu, \sigma$ of the *Frozen* instance.

```
mu1 = 2.0; std1 = 2.0
mu2 = 3.0; std2 = 1.0

# create a frozen normal distribution object
# from mu1, std1
my_norm1 = norm(mu1,std1)

# create a second frozen normal distribution object
my_norm2 = norm(mu2,std2)
```

## Working with a Frozen instance

```
# you'll see this will have all of the functions of norm
print(dir(my_norm1))

# let's plot the pdf of my_norm1 and my_norm2
x = np.linspace(-4.0,8.0,100)
plt.figure()
plt.plot(x, my_norm1.pdf(x))
plt.plot(x, my_norm2.pdf(x))
plt.show()
```

## Working with a Frozen instance: generating random variables

We can use the *rvs* method to generate samples of random variables. The random generation relies on numpy's pseudorandom number generator. This means we can seed the random number generator to ensure the result is reproducible.

```
# seed the random number generator
np.random.seed(1012301)

# generate 100 samples from my_norm1
rv1 = my_norm1.rvs(100)
# generate 10000 samples from my_norm2
rv2 = my_norm2.rvs(10000)
```

## Compare the histogram of the generated samples to the pdf

```
# plot the histogram of the random samples and compare
# to the pdf values

plt.figure()
plt.hist(rv1, normed=True)
plt.hist(rv2, bins=30, normed=True)
plt.plot(x, my_norm1.pdf(x))
plt.plot(x, my_norm2.pdf(x))
plt.show()
```

## Compare cumulative distribution of samples to the cdf

```
# plot the cumulative histogram of the samples
# and compare with the true cdf

plt.figure()
plt.hist(rv1, cumulative=True, normed=True)
plt.hist(rv2, bins=30, cumulative=True, normed=True)
plt.plot(x, my_norm1.cdf(x))
plt.plot(x, my_norm2.cdf(x))
plt.show()
```

## What does the following code do?

Does this code generate 10 samples from the normal distribution?

```
norm.rvs(10)
```

... **Be careful!**

## Be explicit by providing specifying the keywords

Use the *size* keyword to specify the number of samples to generate from a distribution.

The following code will produce the same random samples

```
np.random.seed(82)
aa = norm.rvs(10,3,1000)

np.random.seed(82)
bb = norm.rvs(loc=10, scale=3, size=1000)
# is aa exactly equal to bb?
print((aa == bb).all())

# if you specify the keywords, you can change the order
np.random.seed(82)
cc = norm.rvs(size=1000, loc=10, scale=3)
# is cc exactly equal to bb?
print((cc == bb).all())
```

## Continuous distributions in general

Use *loc* and *scale* to shift and scale distributions. However some distributions require additional shape parameters. To view the number of additional shape parameters use

```
distrubituion_name.numargs
```

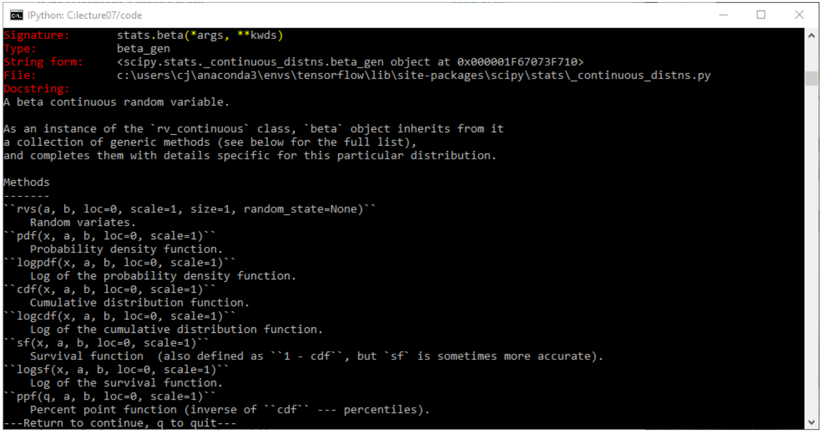
Some examples:

```
in : print(stats.norm.numargs)
out: 0

in : print(stats.gamma.numargs)
out: 1

in : print(stats.beta.numargs)
out: 2
```

**Note**: there is not a shape keyword.

## How do you figure out how to specify the shape parameters?

```
stats.beta?
```



```
Signature:      stats.beta(*args, **kwds)
Type:           beta_gen
String form:    <scipy.stats._continuous_distns.beta_gen object at 0x000001F67073F710>
File:           c:\users\cj\anaconda3\envs\tensorflow\lib\site-packages\scipy\stats\_continuous_distns.py
Docstring:
A beta continuous random variable.

As an instance of the `rv_continuous` class, `beta` object inherits from it
a collection of generic methods (see below for the full list),
and completes them with details specific for this particular distribution.

Methods
-------
``rvs(a, b, loc=0, scale=1, size=1, random_state=None)``
    Random variates.
``pdf(x, a, b, loc=0, scale=1)``
    Probability density function.
``logpdf(x, a, b, loc=0, scale=1)``
    Log of the probability density function.
``cdf(x, a, b, loc=0, scale=1)``
    Cumulative distribution function.
``logcdf(x, a, b, loc=0, scale=1)``
    Log of the cumulative distribution function.
``sf(x, a, b, loc=0, scale=1)``
    Survival function  (also defined as ``1 - cdf``, but `sf` is sometimes more accurate).
``logsf(x, a, b, loc=0, scale=1)``
    Log of the survival function.
``ppf(q, a, b, loc=0, scale=1)``
    Percent point function (inverse of ``cdf`` --- percentiles).
---Return to continue, q to quit---
```

The shape parameters for the beta random variate distribution are *a* and *b*. In this case the order of the shape parameters matters!

## Methods for fitting distributions

You most often use *fit* to perform a maximum likelihood estimation of a distribution's parameters. Maximum likelihood estimations (MLE) can be very ill-posed, so it's recommended to provide *fit* with a reasonable starting point.

| **Method** | **Description** |
|---|---|
| fit | maximum likelihood estimation of parameters |
| fit_loc_scale | estimation of loc and scale provided shape parameters |
| nnlf | negative log likelihood function |
| expect | Calculate the expectation of a function against the pdf or pmf |

**Note**: these methods aren't available for every distribution

## From the tutorial: Performance issues and cautionary remarks

- the performance of each distribution is different
- some methods within distributions are explicitly calculated (by evaluating an analytic expression)
- others are calculated with more costly methods if there is no analytically form (ie using generic algorithms)
- MLE isn't a good choice for some distributions

## You can build your own distributions using Subclassing

Making a continuous distribution is simple and many values will be computed automatically.

```
from scipy import stats
class deterministic_gen(stats.rv_continuous):
    def _cdf(self, x):
        return np.where(x < 0, 0., 1.)
    def _stats(self):
        return 0., 0., 0., 0.
```

## Whats different with discrete distributions?

- Discrete distributions don't have a pdf, however they do have a pmf (probability mass function)
- No estimation methods such as *fit*
- *scale* is not valid, but *loc* still is
- cdf is a step function, thus inverse cdf (ppf) works a little bit differently
$$\mathrm{ppf}(q) = \min(x : \mathrm{cdf}(x) \geq q, x\mathrm{integer}) \tag{1}$$

## Enter the binomial distribution

```python
from scipy.stats import binom
# binom requires two arguments, n and p
n = 100   # number of trials
p = 1./6. # probability that you win

my_binom = binom(n,p)

n_wins = np.arange(0,40,1)
plt.figure()
plt.plot(n_wins,my_binom.pmf(n_wins), 'o')
plt.show()

# what is the proability that you'll win at least 16 times?
print(my_binom.sf(15))

# what is the probability that you won't win more than 25 times?
print(my_binom.cdf(24))
```

## Example: fitting a distribution 1 of 3

```
# let's generate random samples from a normal distribution
# and fit a beta distribution to the samples

np.random.seed(467)
rv = norm.rvs(loc=4.0,scale=1.0,size=10000)

# estimate the mean and variance
est_loc = np.mean(rv)
est_scale = np.std(rv)

# perform the MLE to determine the beta parameters
beta_param = stats.beta.fit(rv, loc=est_loc, scale=est_scale)

# create the beta model
beta_fit = stats.beta(*beta_param)
# equiv to stats.beta(beta_param[0], beta_param[1], ...
```

## Example: fitting a distribution 2 of 3

```
# let's compare the pdf from the fitted beta distribution
# to the histogram of the samples

x = np.linspace(1.0, 7.0, 100)

plt.figure()
plt.hist(rv, bins=30, normed=True)
plt.plot(x, beta_fit.pdf(x))
plt.show()

# and the cdf to the cumulative histogram
plt.figure()
plt.hist(rv, bins=30, cumulative=True, normed=True)
plt.plot(x, beta_fit.cdf(x))
plt.show()
```

## Example: fitting a distribution 3 of 3

```
# we can use the Kolmogorov-Smirnov test to perform a
# hypothesis test to see if our samples do come from beta

# Hypothesis: the rv does indeed come from beta distribution
# for 99% confidence reject if pvalue is less than 0.01

test_res = stats.kstest(rv, 'beta', beta_param)
print('KS-statistic D = %6.3f pvalue = %6.4f' % test_res )

# is this better than a ufitted normal estimate?
test_norm = stats.kstest(rv, 'norm', (np.mean(rv), np.std(rv)))
print('KS-statistic D = %6.3f pvalue = %6.4f' % test_norm )
```

**Note**: Dont' use pvalues to deice if anything is quantitative better than anything else! The pvalue is just an accept or reject of the hypothesis.

## Kolmogorov-Smirnov test for two sample

This is a hypothesis test for to determine whether two sets of samples originate from the same distribution.

Hypotheses: The samples come from the same distribution. Reject if he hypothesis when the pvalue is small.

```
np.random.seed(1212)
rvs1 = stats.norm.rvs(loc=5, scale=10, size=500)
rvs2 = stats.norm.rvs(loc=5, scale=10, size=100)
rvs3 = stats.norm.rvs(loc=4, scale=14, size=200)

# test wheter 1 and 2 come from the same distribution
# for 95% confidece we reject if the pvalue is less than 0.05

test_12 = stats.ks_2samp(rvs1, rvs2)
test_13 = stats.ks_2samp(rvs1, rvs3)
test_23 = stats.ks_2samp(rvs2, rvs3)
```

## So what did we cover today

- Introduction to statistics with scipy.stats
- How the object oriented frozen distributions work
- Random variable distributions
- Continuous and discrete distributions
- PDF, CDF, SF, and other associated functions of distributions
- Maximum likelihood estimation to fit distributions to data
- Hypothesis test: are two samples from the same distribution?
- Hypothesis test: are these samples from this distribution?

## So whats still out there?

- There is a lot more that we can't cover in one day...
- We can make an entire class on statistics in Python
- We didn't cover multivariate distributions...
- There are numerous other functions in scipy.stats
- There are plotting tools in scipy.stats (probability plots, box plots, etc.)