statistics_tools

January 25, 2022

0.1 Statistics Tools

NOTE: In this notebook I use the`stats` submodule of `scipy` for all statistics functions, including generation of random numbers. There are other modules with some overlapping functionality, e.g., the regular python `random` module, and the `np.random` module, but I do not use them here. The `stats` submodule includes tools for a large number of distributions, it includes a large and growing set of statistical functions, and there is a unified class structure. In addition, potential namespace issues are minimized. See https://docs.scipy.org/doc/scipy/reference/stats.html.

Marty Ligare, August 2020

Modified by Tom Solomon, February 2021

Modified by Ned Ladd, January 2022 to remove np.random references in favor of stats.randint.rvs

[1]:
```
import numpy as np
from scipy import stats

import matplotlib as mpl
import matplotlib.pyplot as plt
```

[2]:
```
# Following is an Ipython magic command that puts figures in notebook.
%matplotlib notebook

# M.L. modifications of matplotlib defaults
# Changes can also be put in matplotlibrc file,
# or effected using mpl.rcParams[]
mpl.style.use('classic')
plt.rc('figure', figsize = (6, 4.5)) # Reduces overall size of figures
plt.rc('axes', labelsize=16, titlesize=14)
plt.rc('figure', autolayout = True) # Adjusts supblot parameters for new size
```

0.1.1 Generating random integers

If you want to generate just one random integer, here is a way to do it. You have to specify a "low" and "high" for the range; i.e., you want a random integer between "low" and "high". Note that you have to add one to the "high" limit.

This example pulls a random integer between 3 and 7. Execute the cell a few times to get a feel for it.

[3]:
```
low = 3
high = 7
stats.randint.rvs(low,high+1)
```

[3]:
```
5
```

It is more likely that you will want to get a full array of integers in one fell swoop. Here is how to do that.

[4]:
```
# Generate n integers between low and high:
low = -3
high = 6
n = 100

# or equivalently:
# loq, high, n = (-3, 6, 100)

stats.randint.rvs(low, high+1, size=n)
```

[4]:
```
array([-1, -3,  5,  1, -2,  6, -1,  3, -3,  4,  3, -3,  0,  4,  2, -3,  4,
        3,  1,  3, -3,  4,  2, -1,  1,  2, -1,  2, -3,  5,  0,  0,  6,  6,
        0,  5,  5,  1,  2, -3,  0,  6,  5, -2, -2,  3,  5,  3,  5,  1,  5,
       -1,  1,  6, -1, -3,  0,  0,  3,  2, -1,  5,  5,  6,  6, -1,  2,  0,
        4,  3,  0,  5,  6,  4,  4,  2, -2, -2,  2, -3,  3,  2,  5,  4,  2,
        2,  1,  6,  5, -3,  5,  0,  3,  4,  4, -2,  6,  4,  4,  5])
```

0.1.2 Sampling random numbers from a uniform p.d.f.

[5]:
```
# Sample n random numbers in interval [0.0,1.0]:
n = 10
stats.uniform.rvs(size=n)        # "uniform" in the command indicates all␣
↪values are equally likely
```

[5]:
```
array([0.10742844, 0.94249297, 0.63852152, 0.22815016, 0.10158707,
       0.51507948, 0.01284555, 0.93608384, 0.50174739, 0.79077662])
```

[6]:
```
# Or, if you need a uniform distribution within a different interval
# you can specify the lower bound of the interval, and the range
#
low = 2
rng = 7
stats.uniform.rvs(low,rng,size=10)    # this command will produce 10 random␣
↪values in the interval [2.0,9.0]
```

[6]:
```
array([5.55931838, 4.94966096, 4.80710816, 4.20737306, 2.2157093 ,
       2.34498209, 4.43134214, 8.22540896, 3.3606835 , 7.55013214])
```

0.1.3 Sampling random numbers from a normal distribution

Sample n random numbers from the normal distribution with mean μ, standard deviation σ, and pdf of Eq.(2.4) of Hughes & Hase:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{\sigma^2}\right] \tag{1}$$

[7]:
```
# Sampling from normal distribution
n = 10
mean = 10.
sigma = 2.
stats.norm.rvs(mean, sigma, size=n)   # "norm" in the command indicates a␣
 ↪"normal" or "Gaussian" distribution
```

[7]:
```
array([10.35925742, 10.37432354,  8.62835812, 10.8504949 , 12.00586471,
        8.20935899, 12.69328577, 11.79954121, 15.40961675,  6.62114139])
```

0.1.4 Graph the Probability Distribution Function (PDF) of a Normal Distribution

[8]:
```
plt.figure()
x = np.linspace(mean-3.*sigma, mean+3.*sigma,200)  # make an array of 200␣
 ↪evenly spaced values, starting with
                                                   # mean-3*sigma = 4 and going␣
 ↪to mean+3*sigma = 10
y = stats.norm.pdf(x, mean, sigma)                 # determine the value of the␣
 ↪pdf at each of the points in 'x'
plt.title("pdf of normal distribution")
plt.xlabel("$x$")
plt.ylabel("$p(x)$")
plt.grid()
plt.plot(x, y);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

0.1.5 Graph of the Cumulative Distribution Function (CDF) of normal distribution

[9]:
```
plt.figure()
x = np.linspace(mean-3.*sigma, mean+3.*sigma, 200)   # don't really need to do␣
 ↪this again....
y = stats.norm.cdf(x, mean, sigma)
plt.title("cdf of normal distribution")
plt.xlabel("$x$")
plt.ylabel("$\int_{-\infty}^x p(x^\prime)\, dx^\prime$")
```

```
plt.grid()
plt.plot(x, y);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

0.2 Sampling random numbers from a Poisson distribution

Sample n random numbers from the Poisson distribution with average count $\overline{N}$, and probability distibution given by Eq.(3.1) of Hughes & Hase:

$$p(N;\overline{N}) = \frac{\exp\left(-\overline{N}\right)\overline{N}^{N}}{N!} \tag{2}$$

The standard deviation of the Poisson distribution is given by

$$\sigma = \sqrt{\overline{N}}.$$

[10]:
```
# Sampling from a Poisson distribution
n = 100
mean = 2
stats.poisson.rvs(mean, size=n)
```

[10]:
```
array([4, 2, 3, 3, 1, 2, 1, 3, 1, 0, 2, 2, 3, 2, 1, 1, 2, 2, 1, 1, 4, 1,
       1, 1, 5, 3, 7, 2, 3, 2, 2, 1, 0, 2, 6, 2, 0, 2, 2, 3, 6, 3, 0, 3,
       3, 2, 3, 1, 0, 0, 1, 1, 2, 1, 1, 2, 2, 5, 0, 1, 3, 3, 2, 3, 3, 3,
       1, 2, 2, 4, 4, 3, 1, 2, 2, 2, 2, 4, 2, 2, 2, 1, 2, 3, 2, 3, 2, 2,
       0, 3, 0, 1, 2, 0, 3, 3, 1, 3, 6, 3])
```

[11]:
```
np.mean(_)    # The underscore "_" is similar to Mathematicas "%"
              # It refers to the output of the previously executed cell
```

[11]:
```
2.15
```

[12]:
```
np.std(__)    # Notice the double underscore "__"
```

[12]:
```
1.3955285736952863
```

[13]:
```
np.sqrt(mean)
```

[13]:
```
1.4142135623730951
```

0.2.1 Graph of the Probability Mass Function (PMF; it's like a PDF) of Poisson distribution

[14]:
```
plt.figure()
x = np.linspace(0, 12, 13)
y = stats.poisson.pmf(x, mean)
plt.title("pmf of Poisson distribution")
plt.xlabel("$n$")
plt.ylabel("$p(n)$")
plt.grid()
plt.axhline(0)
plt.scatter(x, y);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

0.2.2 Graph of the CDF of Poisson distribution

[15]:
```
plt.figure()
x = np.linspace(0, 12, 13)
y = stats.poisson.cdf(x, mean)
plt.title("cdf of Poisson distribution")
plt.xlabel("$x$")
plt.ylabel("$C_{DF}$")
plt.xlim(-1, 13)
plt.grid()
plt.axhline(0)
plt.scatter(x, y);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

0.3 Sampling random numbers from a binomial distribution

Consider n trials, with probability of success p in each trial. The array below is the number successes in each of *size* trials.

[16]:
```
# Sampling from a binomial distribution
n = 2
p = 0.4
stats.binom.rvs(n, p, size=100)
```

[16]:
```
array([0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 2, 1, 1, 0, 1, 1, 1, 0,
       2, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 2, 2, 0, 0, 1, 0, 1, 2, 2, 2, 1, 0, 0, 1,
       0, 1, 1, 1, 0, 2, 2, 1, 1, 2, 2, 0, 1, 1, 0, 0, 1, 1, 2, 2, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0])
```

[17]:
```
np.mean(_)
```

[17]:
```
0.74
```

The probablity of getting x successes is given by the probability mass function (pmf). This is analogous to the continous pdf (and it's called the PDF in Mathematica). As an example, the probability of 2 successes in 3 trials with a probability of success in each trial of 0.4 is 29%:

[18]:
```
n, s, p = (3, 2, 0.4)
stats.binom.pmf(s, n, p)
```

[18]:
```
0.288
```

Graph of pmf (~pdf) of binomial distribution

[19]:
```
plt.figure()
n = 5
x = np.linspace(0, n, n+1)
y = stats.binom.pmf(x, n, p)

plt.title("pmf ($\sim$pdf) of binom. dist.; $n=5$, $p = 0.4$")
plt.xlabel("$n$")
plt.ylabel("probability of $n$ successes")
plt.grid()
plt.axhline(0)
plt.scatter(x, y);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Graph of cdf of binomial distribution

[20]:
```
plt.figure()
n = 5
x = np.linspace(0, n, n+1)
y = stats.binom.cdf(x, n, p)
plt.title("cdf of binomial dist.; $n=5$, $p = 0.4$")
plt.xlabel("$n$")
plt.ylabel("cdf")
plt.grid()
```

```
plt.axhline(0)
plt.scatter(x, y);
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

0.4 Histograms

Generate some random data from a normal distribution.

[21]:
```
n = 100
mean = 10.
sigma = 2.
data = stats.norm.rvs(mean, sigma, size=n)
```

Select number of bins between low and high values. NOTE: `plt.hist` plots the histogram, and ouputs the binned data

[22]:
```
plt.figure()
nbins = 6
low = mean - 3*sigma
high = mean + 3*sigma
plt.xlabel("value")
plt.ylabel("occurences")
plt.title("Histogram; equal sized bins")
out = plt.hist(data, nbins, [low,high])
out[0],out[1]    # occurrences and bin boundaries
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[22]:
```
(array([ 3., 16., 28., 38., 12.,  3.]),
 array([ 4.,  6.,  8., 10., 12., 14., 16.]))
```

OR ... Select specific bin boundaries Again, `plt.hist` outputs the binned data and plots the histogram.

[23]:
```
plt.figure()
bins = [4, 7, 8, 10, 13, 16]
plt.xlabel("value")
plt.ylabel("occurences")
plt.title("Histogram; specified (nonuniform) bins")
out = plt.hist(data, bins)
```

```
out[0],out[1]    # occurrences and bin boundaries
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[23]:
```
(array([10.,  9., 28., 43., 10.]), array([ 4,  7,  8, 10, 13, 16]))
```

Version Information `version_information` is from J.R. Johansson (jrjohansson at gmail.com); see Introduction to scientific computing with Python for more information and instructions for package installation.

`version_information` is installed on the linux network at Bucknell

[24]:
```
%load_ext version_information
```

[25]:
```
version_information numpy, scipy, matplotlib
```

[25]:

Software	Version
Python	3.7.8 64bit [GCC 7.5.0]
IPython	7.17.0
OS	Linux 3.10.0 1160.36.2.el7.x86_64 x86_64 with centos 7.9.2009 Core
numpy	1.19.1
scipy	1.5.0
matplotlib	3.3.0
Tue Jan 25 10:37:44 2022 EST	

[ ]:
```

```