Programming in Python

notes on statistics

Topics covered

1. scipy vs numpy
2. Statistical distributions
3. Random number generation
4. Statistical parameters
5. Plots

Numpy vs Scipy

- numpy has several methods and functions for statistics
- scipy is a library created for scientific computing and has much more
- https://scipy.org/about.html

What is the difference between NumPy and SciPy?

- From: https://www.scipy.org/scipylib/faq.html
  - *In an ideal world, NumPy would contain nothing but the array data type and the most basic operations: indexing, sorting, reshaping, basic elementwise functions, et cetera. All numerical code would reside in SciPy. However, one of NumPy's important goals is compatibility, so NumPy tries to retain all features supported by either of its predecessors. Thus NumPy contains some linear algebra functions, even though these more properly belong in SciPy. In any case, SciPy contains more fully-featured versions of the linear algebra modules, as well as many other numerical algorithms. If you are doing scientific computing with python, you should probably install both NumPy and SciPy. Most new features belong in SciPy rather than NumPy.*
- A (not so) side note, in scipy:
  - *the time-critical loops are usually implemented in C or Fortran. Much of SciPy is a thin layer of code on top of the scientific routines that are freely available at http://www.netlib.org/ Netlib is a huge repository of incredibly valuable and robust scientific algorithms written in C and Fortran.*

numpy statistics

Order statistics

amin(a[, axis, out, keepdims, initial])	Return the minimum of an array or minimum along an axis.
amax(a[, axis, out, keepdims, initial])	Return the maximum of an array or maximum along an axis.
nanmin(a[, axis, out, keepdims])	Return minimum of an array or minimum along an axis, ignoring any NaNs.
nanmax(a[, axis, out, keepdims])	Return the maximum of an array or maximum along an axis, ignoring any NaNs.
ptp(a[, axis, out, keepdims])	Range of values (maximum - minimum) along an axis.
percentile(a, q[, axis, out, ...])	Compute the q-th percentile of the data along the specified axis.
nanpercentile(a, q[, axis, out, ...])	Compute the qth percentile of the data along the specified axis, while ignoring nan values.
quantile(a, q[, axis, out, overwrite_input, ...])	Compute the q-th quantile of the data along the specified axis.
nanquantile(a, q[, axis, out, ...])	Compute the qth quantile of the data along the specified axis, while ignoring nan values.

Averages and variances

median(a[, axis, out, overwrite_input, keepdims])	Compute the median along the specified axis.
average(a[, axis, weights, returned])	Compute the weighted average along the specified axis.
mean(a[, axis, dtype, out, keepdims])	Compute the arithmetic mean along the specified axis.
std(a[, axis, dtype, out, ddof, keepdims])	Compute the standard deviation along the specified axis.
var(a[, axis, dtype, out, ddof, keepdims])	Compute the variance along the specified axis.
nanmedian(a[, axis, out, overwrite_input, ...])	Compute the median along the specified axis, while ignoring NaNs.
nanmean(a[, axis, dtype, out, keepdims])	Compute the arithmetic mean along the specified axis, ignoring NaNs.
nanstd(a[, axis, dtype, out, ddof, keepdims])	Compute the standard deviation along the specified axis, while ignoring NaNs.
nanvar(a[, axis, dtype, out, ddof, keepdims])	Compute the variance along the specified axis, while ignoring NaNs.

Correlating

corrcoef(x[, y, rowvar, bias, ddof])	Return Pearson product-moment correlation coefficients.
correlate(a, v[, mode])	Cross-correlation of two 1-dimensional sequences.
cov(m[, y, rowvar, bias, ddof, fweights, ...])	Estimate a covariance matrix, given data and weights.

Histograms

histogram(a[, bins, range, normed, weights, ...])	Compute the histogram of a set of data.
histogram2d(x, y[, bins, range, normed, ...])	Compute the bi-dimensional histogram of two data samples.
histogramdd(sample[, bins, range, normed, ...])	Compute the multidimensional histogram of some data.
bincount(x[, weights, minlength])	Count number of occurrences of each value in array of non-negative ints.

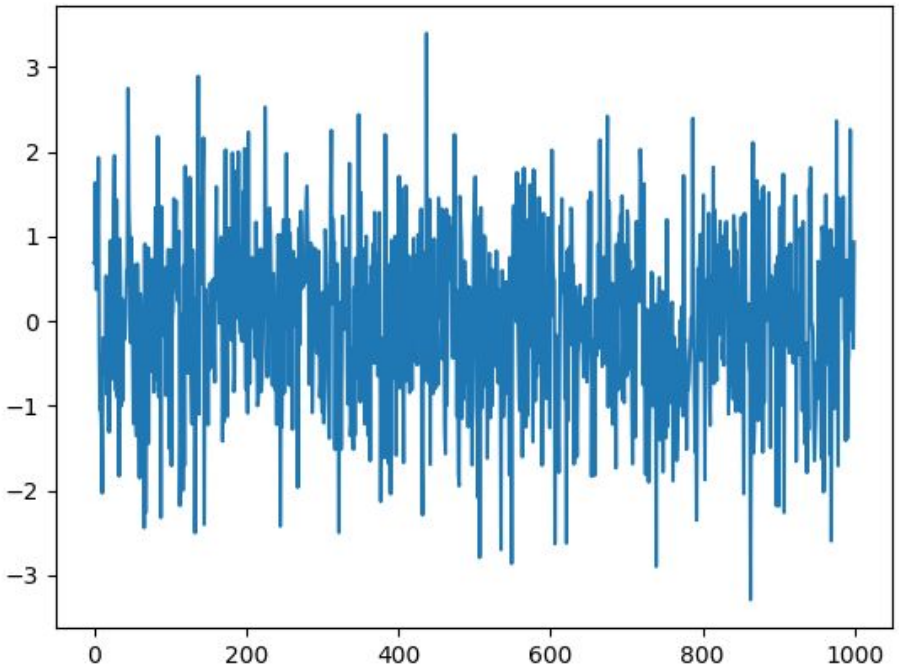
- https://docs.scipy.org/doc/numpy/reference/routines.statistics.html

scipy statistics

- https://docs.scipy.org/doc/scipy-1.2.0/reference/stats.html
- Over 80 continuous random variables (RVs) and 10 discrete random variables are implemented
- All of the statistics functions are located in the sub-package scipy.stats
- First of all, all distributions are accompanied with help functions. To obtain just some basic information we print the relevant docstring: print(stats.norm.__doc__)

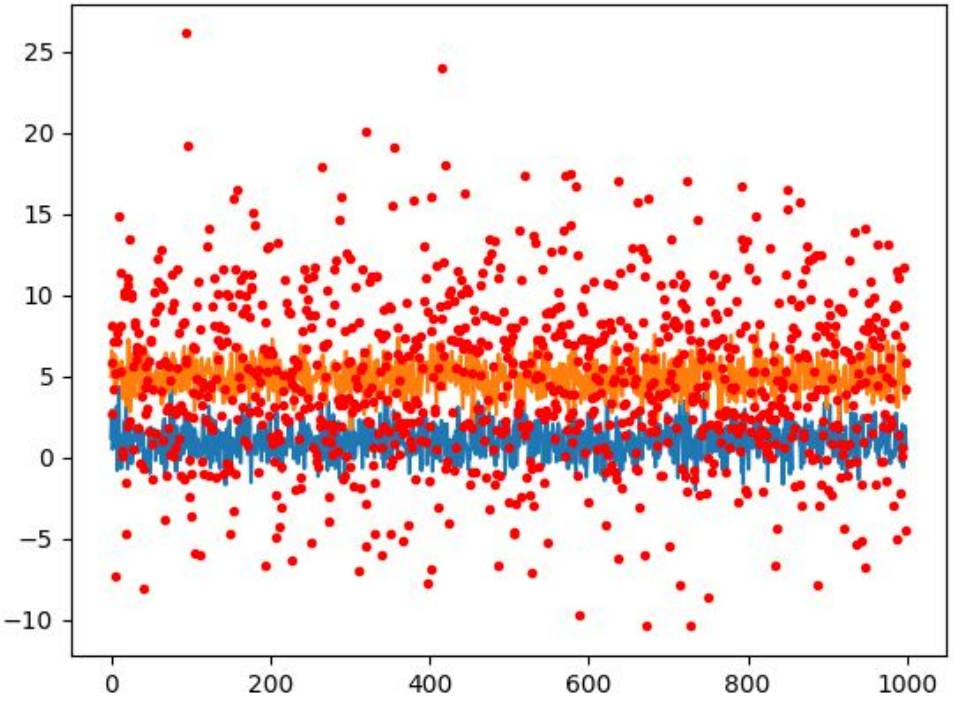
First example

```
import scipy.stats as stat
import numpy as np
import matplotlib.pyplot as plt
y = stat.norm.rvs(size = 1000)
plt.plot(y)
plt.show()
```



Location and scale

```
import scipy.stats as stat
import numpy as np
import matplotlib.pyplot as plt
y = stat.norm.rvs(1, 1, size = 1000)
plt.plot(y)
y = stat.norm.rvs(5, 1, size = 1000)
plt.plot(y)
y = stat.norm.rvs(5, 5, size = 1000)
plt.plot(y, 'r.')
plt.show()
```



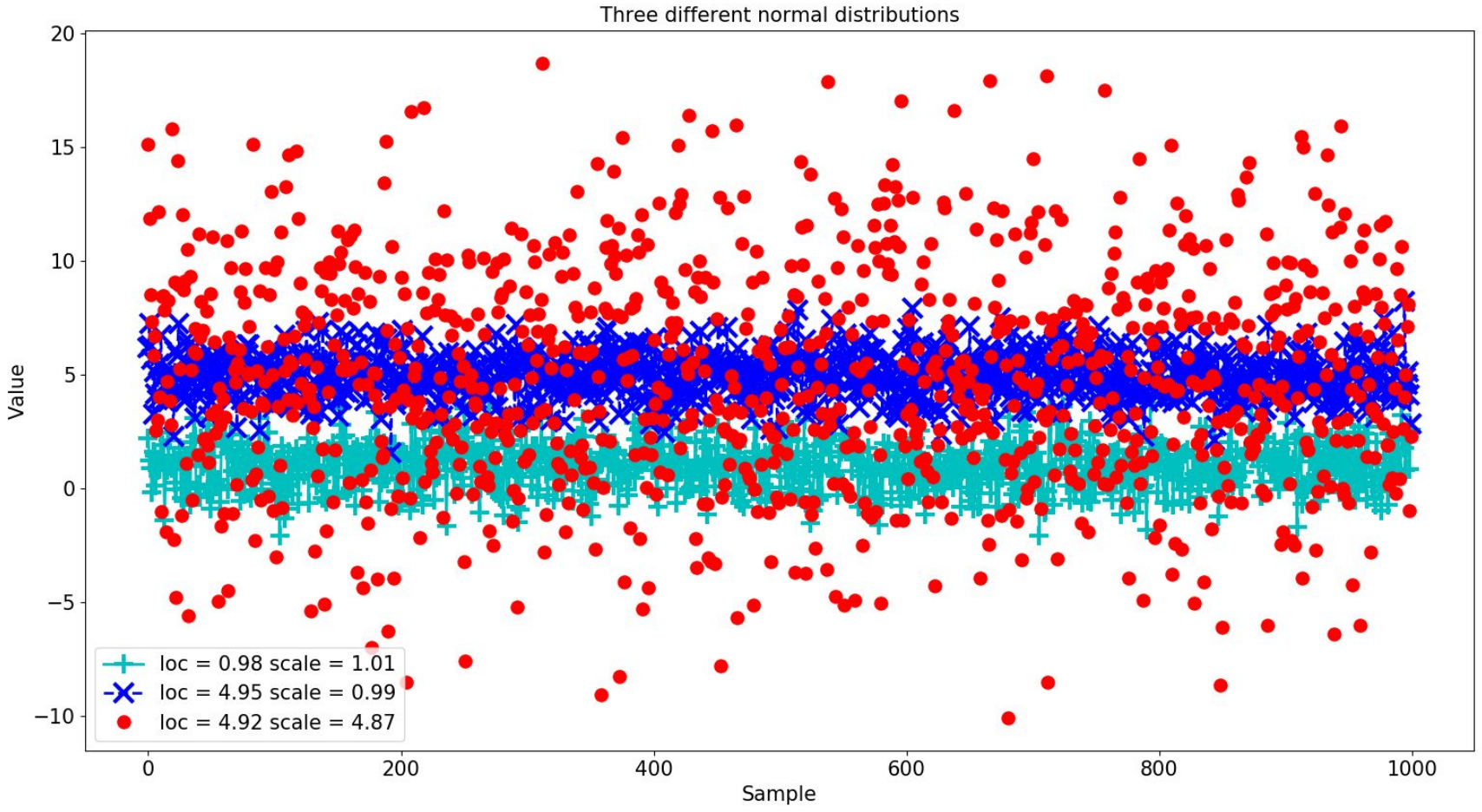
Descriptive statistics

```
import scipy.stats as stat
import numpy as np
import matplotlib.pyplot as plt
y = stat.norm.rvs(1, 1, size = 1000)
print(stat.describe(y))
y = stat.norm.rvs(5, 1, size = 1000)
print(stat.describe(y))
y = stat.norm.rvs(5, 5, size = 1000)
print(stat.describe(y))
```

Compute several descriptive statistics of the passed array

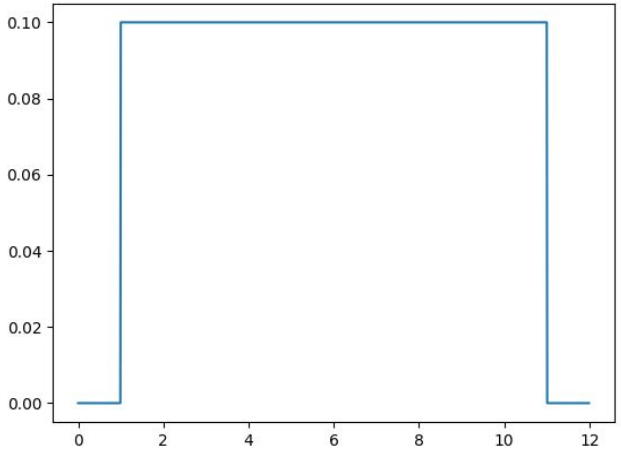
```
>>> DescribeResult(nobs=1000,
minmax=(-2.05892087454428, 4.6876407515170895),
mean=1.0089018367372018,
variance=0.8905933473932526,
skewness=-0.10647245290115326,
kurtosis=0.1705432170870762)
DescribeResult(nobs=1000,
minmax=(1.2211249141934535, 8.520534977954336),
mean=4.975644011985953,
variance=0.9596468187757006,
skewness=-0.05758158737624451,
kurtosis=0.09465047726889475)
DescribeResult(nobs=1000,
minmax=(-11.864506863135453,
21.757590172525674), mean=4.761939651932576,
variance=26.110097089984563,
skewness=0.1138538458229636,
kurtosis=-0.03558238566952188)
```

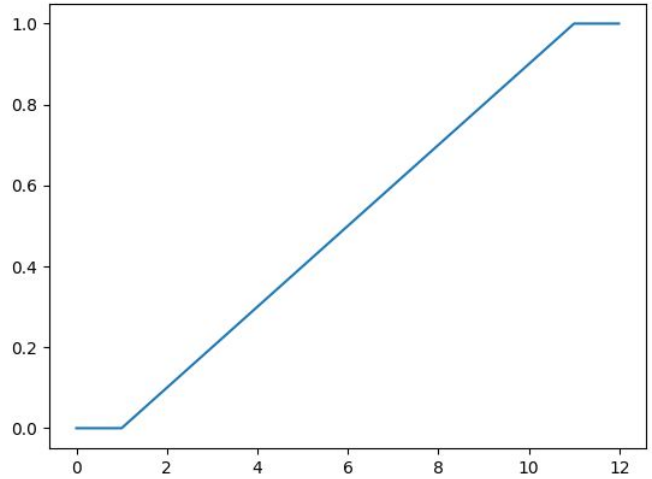
Now you do this



PDF and CDF

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 12, 0.01)
y1 = stat.uniform.pdf(x, 1, 10)
y2 = stat.uniform.cdf(x, 1, 10)
plt.figure()
plt.plot(x, y1)
plt.figure()
plt.plot(x, y2)
plt.show()
```





Now you do this