0.1 Statistics Functions

`mean`

`mean` computes the average of an array. An optional second argument provides the axis to use (default is to use entire array). `mean` can be used either as a function or as a method on an array.

```
>>> x = arange(10.0)
>>> x.mean()
4.5
>>> mean(x)
4.5
>>> x= reshape(arange(20.0),(4,5))
>>> mean(x,0)
231
array([ 7.5, 8.5, 9.5, 10.5, 11.5])
>>> x.mean(1)
array([ 2., 7., 12., 17.])
```

`median`

`median` computed the median value in an array. An optional second argument provides the axis to use (default is to use entire array).

```
>>> x= randn(4,5)
>>> x
array([[0.74448693,
0.63673031,
0.40608815,
0.40529852, 0.93803737],
[ 0.77746525, 0.33487689, 0.78147524, 0.5050722
, 0.58048329],
[0.51451403,
0.79600763,
0.92590814, 0.53996231,
0.24834136],
[0.83610656,
0.29678017, 0.66112691,
0.10792584, 1.23180865]])
>>> median(x)
0.45558017286810903
>>> median(x, 0)
array([0.62950048,
0.16997507,
0.18769355, 0.19857318,
0.59318936])
```

Note that when an array or axis dimension contains an even number of elements (n), median returns the average of the 2 inner elements.

`std`

`std` computes the standard deviation of an array. An optional second argument provides the axis to use (default is to use entire array). std can be used either as a function or as a method on an array.

`var`

`var` computes the variance of an array. An optional second argument provides the axis to

`corrcoef`

`corrcoef(x)` computes the correlation between the rows of a 2-dimensional array x . `corrcoef(x, y)` computes the correlation between two 1- dimensional vectors. An optional keyword argument rowvar can be used to compute the correlation between the columns of the input – this is corrcoef(x, rowvar=False) and `corrcoef(x.T)` are identical

```
>>> x= randn(3,4)
>>> corrcoef(x)
array([[ 1. , 0.36780596, 0.08159501],
[ 0.36780596, 1. , 0.66841624],
[ 0.08159501, 0.66841624, 1. ]])
>>> corrcoef(x[0],x[1])
array([[ 1. , 0.36780596],
[ 0.36780596, 1. ]])
>>> corrcoef(x, rowvar=False)
array([[ 1. , 0.98221501,
0.19209871,
0.81622298],
[0.98221501,
1. , 0.37294497, 0.91018215],
[0.19209871,
0.37294497, 1. , 0.72377239],
[0.81622298,
0.91018215, 0.72377239, 1. ]])
>>> corrcoef(x.T)
array([[ 1. , 0.98221501,
0.19209871,
0.81622298],
[0.98221501,
1. , 0.37294497, 0.91018215],
[0.19209871,
0.37294497, 1. , 0.72377239],
[0.81622298,
0.91018215, 0.72377239, 1. ]])
```

`cov`

`cov(x)` computes the covariance of an array x . `cov(x,y)` computes the covariance between two 1-dimensional vectors. An optional keyword argument rowvar can be used to compute the covariance between the columns of the input – this is `cov(x, rowvar=False)` and `cov(x.T)` are identical.

`histogram`

`histogram` can be used to compute the histogram (empirical frequency, using k bins) of a set of data. An optional second argument provides the number of bins. If omitted, `k =10` bins are used. histogram returns two outputs, the first with a k-element vector containing the number of observations in each bin, and the second with the k + 1 endpoints of the k bins.

```
>>> x = randn(1000)
>>> count, binends = histogram(x)
>>> count
array([ 7, 27, 68, 158, 237, 218, 163, 79, 36, 7])
>>> binends
array([3.06828057,
2.46725067,
1.86622077,
1.26519086,
0.66416096,
0.06313105,
0.53789885, 1.13892875, 1.73995866, 2.34098856,
2.94201846])
>>> count, binends = histogram(x, 25)
```

`histogram2d`

`histogram2d(x,y)` computes a 2-dimensional histogram for 1-dimensional vectors. An optional keyword argument bins provides the number of bins to use. bins can contain either a single scalar integer or a 2-element list or array containing the number of bins to use in each dimension.

0.2 More Statistical Functions

`Full Specification`

For the sake of brevity and space, the functions names only are presented below. It may be necessary to use their full specifications, depending on how you have specified the relevant `import` functions. Examples of full specifications are:

```
np.random.randint()
np.stats.linregress()
```

`mode`

mode computes the mode of an array. An optional second argument provides the axis to use (default is to use entire array). Returns two outputs: the first contains the values of the mode, the second contains the number of occurrences.

```
>>> x=randint(1,11,1000)
>>> stats.mode(x)
(array([ 4.]), array([ 112.]))
```

`moment`

moment computed the rth central moment for an array. An optional second argument provides the axis to use (default is to use entire array).

```
>>> x = randn(1000)
>>> moment = stats.moment
>>> moment(x,2) moment(
x,1)**2
0.94668836546169166
>>> var(x)
0.94668836546169166
>>> x = randn(1000,2)
>>> moment(x,2,0) # axis 0
array([ 0.97029259, 1.03384203])
```

`skew`

skew computes the skewness of an array. An optional second argument provides the axis to use (default is to use entire array).

```
>>> x = randn(1000)
>>> skew = stats.skew
>>> skew(x)
0.027187705042705772
>>> x = randn(1000,2)
>>> skew(x,0)
array([ 0.05790773, 0.00482564])
```

`kurtosis`

kurtosis computes the excess kurtosis (actual kurtosis minus 3) of an array. An optional second argument provides the axis to use (default is to use entire array). Setting the keyword argument fisher=False will compute the actual kurtosis.

```
>>> x = randn(1000)
>>> kurtosis = stats.kurtosis
>>> kurtosis(x)
0.2112381820194531
>>> kurtosis(x, fisher=False)
2.788761817980547
>>> kurtosis(x, fisher=False) kurtosis(
x) # Must be 3
3.0
>>> x = randn(1000,2)
>>> kurtosis(x,0)
array([0.13813704,
0.08395426])
```

`pearsonr`

pearsonr computes the Pearson correlation between two 1-dimensional vectors. It also returns the 2- tailed p-value for the null hypothesis that the correlation is 0.

```
>>> x = randn(10)
>>> y = x + randn(10)
>>> pearsonr = stats.pearsonr
>>> corr, pval = pearsonr(x, y)
>>> corr
0.40806165708698366
>>> pval
0.24174029858660467
```

`spearmanr`

spearmanr computes the Spearmancorrelation (rank correlation). It can be used with a single 2-dimensional array input, or 2 1-dimensional arrays. Takes an optional keyword argument axis indicating whether to treat columns (0) or rows (1) as variables. If the input array has more than 2 variables, returns the correlation matrix. If the input array as 2 variables, returns only the correlation between the variables.

```
>>> x = randn(10,3)
>>> spearmanr = stats.spearmanr
>>> rho, pval = spearmanr(x)
>>> rho
array([[ 1. , 0.02087009,
0.05867387],
[0.02087009,
1. , 0.21258926],
[0.05867387,
0.21258926, 1. ]])
>>> pval
array([[ 0. , 0.83671325, 0.56200781],
[ 0.83671325, 0. , 0.03371181],
[ 0.56200781, 0.03371181, 0. ]])
>>> rho, pval = spearmanr(x[:,1],x[:,2])
>>> corr
0.020870087008700869
>>> pval
0.83671325461864643
```

`linregress`

linregress estimates a linear regression between 2 1-dimensional arrays. It takes two inputs, the independent variables (regressors) and the dependent variable (regressand). Models always include a constant.

```
>>> x = randn(10)
>>> y = x + randn(10)
>>> linregress = stats.linregress
>>> slope, intercept, rvalue, pvalue, stderr = linregress(x,y)
>>> slope
1.6976690163576993
>>> rsquare = rvalue**2
>>> rsquare
0.59144988449163494
>>> x.shape = 10,1
>>> y.shape = 10,1
>>> z = hstack((x,y))
>>> linregress(z) # Alternative form, [x y]
(1.6976690163576993,
0.79983724584931648,
0.76905779008578734,
0.0093169560056056751,
0.4988520051409559)
```

0.3 Select Statistical Tests

`normaltest`

`normaltest` tests for normality in an array of data. An optional second argument provides the axis to use (default is to use entire array). Returns the test statistic and the p-value of the test. This test is a small sample modified version of the Jarque-Bera test statistic.

`kstest`

`kstest` implements the Kolmogorov-Smirnov test. Requires two inputs, the data to use in the test and the distribution, which can be a string or a frozen random variable object. If the distribution is provided as a string, then any required shape parameters are passed in the third argument using a tuple containing these parameters, in order.

```
>>> x = randn(100)
>>> kstest = stats.kstest
>>> stat, pval = kstest(x, 'norm')
>>> stat
0.11526423481470172
>>> pval
0.12963296757465059
>>> ncdf = stats.norm().cdf # No () on cdf to get the function
>>> kstest(x, ncdf)
(0.11526423481470172, 0.12963296757465059)
>>> x = gamma.rvs(2, size = 100)
>>> kstest(x, 'gamma', (2,)) # (2,) contains the shape parameter
(0.079237623453142447, 0.54096739528138205)
>>> gcdf = gamma(2).cdf
>>> kstest(x, gcdf)
(0.079237623453142447, 0.54096739528138205)
```

0.3.1 `ks_2samp`

`ks_2samp` implements a 2-sample version of the Kolmogorov-Smirnov test. It is called `ks_2samp(x,y)` where both inputs are 1-dimensonal arrays, and returns the test statistic and p-value for the null that the distribution of x is the same as that of y .

0.3.2 `shapiro`

`shapiro` implements the Shapiro-Wilk test for normality on a 1-dimensional array of data. It returns the test statistic and p-value for the null of normality.

1 Statsmodels

`Statsmodels` is a Python module that allows users to explore data, estimate statistical models, and perform statistical tests. An extensive list of descriptive statistics, statistical tests, plotting functions, and result statistics are available for different types of data and each estimator. Researchers across fields may find that statsmodels fully meets their needs for statistical computing and data analysis in Python.

Features include:

- Linear regression models
- Generalized linear models
- Discrete choice models
- Robust linear models
- Many models and functions for time series analysis
- Nonparametric estimators
- A collection of datasets for examples
- A wide range of statistical tests
- Input-output tools for producing tables in a number of formats (Text, LaTex, HTML) and for reading Stata files into NumPy and Pandas.
- Plotting functions
- Extensive unit tests to ensure correctness of results
- Many more models and extensions in development