Sage Quick Reference: Basic Statistics

Thien-Y Le and Kelsey Kofmehl
Sage Version 4.8

Basic Functions

Mean: `mean([4, 6, 2.3])`
Median: `median([4, 6, 2.3])`
Mode: `mode([3, 3, 5, 8])`
Moving Average: `moving_average(v,n)`
v = list, n = number of values used in computing average
`moving_average([1, 2, 3, 10], 4)`
Standard Deviation: `std(v, bias = False)` v = list, bias = False by default (divide by len(v) -1) if True (divide by len(v))
`std([110], bias = True)`
Variance: `variance(v, bias = False)` v = list, bias = False by default (divide by len(v) -1) if True (divide by len(v))
`variance([1, 4, 5], bias = True)`

C Int Lists

List: `v = stats.IntList([1, 4, 5])`
Max: `v= list` `v.max()` or `v.max(index = True)`, index Boolean: default False(returns only int of largest value, if True (returns max and index of max)
`v=stats.IntList([1,5, 12]); v.max(index=True)`
Min: `v= list` `v.min()` or `v.min(index = True)`, index Boolean: default False(returns only int of minimum value, if True (returns min and index of min)
`v=stats.IntList([1,5, 12]); v.min(index=True)`
Plot: `stats.IntList([1,5, 12]).plot())`
Histogram Plot:
`stats.IntList([1,5,12]).plot_histogram()`
Product: (product of all the entries in list v)
`v = stats.IntList([1,5, 12]); v.prod()`
Sum: (sum of all the entries in list v)
`v = stats.IntList([1,5, 12]); v.sum()`
Time series: (changes entries to double, returns time series of self)
`v = stats.IntList([1,5, 12]); v.time_series()`

Time Series

Creating: `v = finance.TimeSeries([e, pi, 12])`
Absolute Value: `v.abs()`
Adding Entries: adds entries of self and input together
`v.add_entries(5, 6, 2)`
Adding Scalars: adds a scalar to every value
`v.add_scalar(8)`
Auto Correlation: return the n-th sample autocorrelation (default n=1)
`v.autocorrelation()`
Auto Covariance: returns the n-th value autocovariance (default n=0)
`v.autocovariance(1)`
Auto Regressive Fit: fit the time series to an autoregressive process of order m
`v.autoregressive_fit(100)`
Central Moment: returns n-th central moment of self
`v.central_moment(4)`
Clip Remove: removes all values that are ¡= min or ¿= max `v.clip_remove(4, 15)`
Correlation:
`v.correlation(finance.TimeSeries([1,2,3]))`
Covariance:
`v.covariance(finance.TimeSeries([1,2,3]))`
Difference: returns time series got by taking the difference of successive terms
`v.diffs()`
Exponent: applies exponent map to all terms `v.exp()`
Exponential Moving Average:
`v.exponential_moving_average(.5)`
Extend: adds elements to the right
`v.extend([2,8,pi])`
Histogram: `v.histogram(5)`
IFFT: `v.ifft()`
Plot: `v.plot()`
Candlestick Plot: `v.plot_candlestick(bins = 3)`
Histogram Plot:`v.plot_histogram(bins = 10)`
Power: Every element raised to the n-th power
`v.pow(2)`
Product:`v.prod()`
Randomize:
`finance.TimeSeries(5).randomize(normal, 5, 2)`
Scale:
returns self rescaled: `v.rescale(7)`
returns new time series: `v.scale(7)`
Reversed: `v.reversed()`
Standard Deviation: `v.standard_deviation()`
Simple Moving Average: `v.simple_moving_average(3)`
Vector: `v.vector()`

Using Scipy Stats

```
import numpy as np
from scipy import stats as sp
import warnings
warnings.simplefilter('ignore',
  DeprecationWarning)
```

Scipy offers at least 84 different continuous distributions and at least 12 different discrete distributions; I will outline a few common ones below.
We can list all methods and properties of the distribution with:
`dir(stats.type)` e.g. `dir(stats.norm)`
The main public methods are defined as:

- 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
- median: Median
- mean: Mean
- var: Variance
- std: Standard Deviation

For discrete distributions pdf is replaced the probability mass function pmf, and no estimation methods, such as fit, are available.
A complete list of distributions and methods can be found at: `http://docs.scipy.org/doc/scipy/reference/stats.html`

Continuous Distributions

(take the `loc` and `scale` as keyword parameters to adjust location and size of distribution e.g., for the standard normal distribution location is the mean and scale is the standard deviation)
Common types of continuous distributions:
Normal

```
  from scipy.stats import norm
  numargs = norm.numargs
  [  ] = [0.9,] * numargs
  rv = norm()
  R = norm.rvs(size = 10)
  R.std()
```

Cauchy

```
  from scipy.stats import cauchy
  numargs = cauchy.numargs
  [ ] = [0.9,] * numargs
  rv = cauchy()
  rv.cdf(.8)
```

Expontential

```
  from scipy.stats import expon
  numargs = expon.numargs
  [ ] = [0.9,] * numargs
  rv = expon()
  rv.std()
```

Discrete Distributions

(The location parameter, keyword `loc` can be used to shift the distribution) Common types of discrete distributions:

Bernoulli

```
  from scipy.stats import bernoulli
  [ pr ] = [.9,]
  rv = bernoulli(pr)
  bernoulli.var(pr)
```

Poisson

```
  from scipy.stats import poisson
  [ mu ] = [.9,]
  rv = poisson(mu)
  R = poisson.rvs(mu, size = 10)
```

Statistical Functions

Geometric mean: `stats.gmean(a, axis, dtype)`

a = array, axis = default 0, axis along which geometric mean is computed, dtype = type of returned array

```
stats.gmean([1,4, 6, 2, 9], axis=0, dtype=None)
```

Computed median: `stats.cmedian(a[, numbins])`

a = array, numbins = number of bins used to histogram the data

```
stats.cmedian([2, 3, 5, 6, 12, 345, 333], 2)
```

Trimmed mean: `stats.tmean(a, limits=None, inclusive=(True, True))`

Harmonic mean: `stats.hmean(a, axis=0, dtype=None)`

Skewed: `stats.skew(a, axis=0, bias=True)`

Signal to noise ratio:
`stats.signaltonoise(a, axis=0, ddof=0)`

Standard error of the mean:
`stats.sem(a, axis=0, ddof=1)`

Histogram: `stats.histogram2(a, bins)`

Relative Z-scores: `stats.zmap(scores, compare, axis=0, ddof=0`

Z-score of each value: `stats.zscore(a, axis=0,ddof=0)`

Regression line: `stats.linregress(x, y=None)`

```
x = np.random.random(20)
y = np.random.random(20)
slope, intercept, r_value, p_value,
  std_err = stats.linregress(x,y)
```

For a complete list see Statistical Functions: `http://docs.scipy.org/doc/scipy/reference/stats.html`

Plots

Probability plot: Calculate quantiles for a probability plot of sample data against a specified theoretical distribution.

```
stats.probplot(x, sparams=(), dist='norm',
  fit=True, plot=None)
```

x = array, sample response data, sparams = tuple, optional, dist = distribution function name (default = normal), fit = Boolean (default true) fit a least squares regression line to data, plot = If given, plots the quantiles and least squares fit. plot is an object with methods plot, title, xlabel, ylabel and text. By default, no plot is created. The figure is shown by probplot; plot.show()

```
stats.probplot([6, 23, 6, 23, 15, 6, 32, 1],
  sparams=(), dist"'norm", fit=True, plot=None)
```

Ppcc max (Returns the shape parameter that maximizes the probability plot correlation coefficient for the given data to a one-parameter family of distributions)

```
stats.ppcc_max(x,brack=(0.0,1.0),
  dist='tukeylambda'
```

Ppcc (Returns (shape, ppcc), and optionally plots shape vs. ppcc (probability plot correlation coefficient) as a function of shape parameter for a one-parameter family of distributions from shape value a to b.)

```
stats.ppcc_plot(x, a, b, dist='tukeylambda'
  plot=None, N=80
```

Using R from Sage

1) `%r` putting a "percent directive" in cell and just use normal R commands afterwards; commands entered in notebook will be sent directly to R program
   Caution: This method works on arbitrary Sage programs
2) `r.r_command_here` using R objective and dot-notation; outputs objects as an R Element in sage
   Type in "r." then tab key to get all r. functions
3) `r.eval('r_commmand_here')` inputting string containing R code and get outputs as a string
4) rpy2 is another interface that uses R in Sage
   For complete information: `http://rpy.sourceforge.net/rpy2/doc-2.3/html/index.html`

Installing packages

`r.install_packages(Hmisc)` installs 'Hmisc' package
`r.install_packages()` see all installed packages

Uploading data to use in R

1) Load .csv data normally into Sage
2) `r.eval("aaa <- read.csv('%s')" %(DATA+'File_Name_Here'))` loads data into R from Sage and assigns it as "aaa"
   `r("print(aaa)")` will print "aaa" data
   `r("print(aaa[1,])")` print first row of "aaa"
   `r("print(aaa[,1])")` print first column of "aaa"

Accessing R datasets

`r.library("R_dataset_name')`

For quickreference in normal R Program: `http://cran.r-project.org/doc/contrib/Short-refcard.pdf`

Creating Data in Sage With R Commands

***Note**: When using r.eval() functions, make sure x variable is also defined prior in r.eval()
Ex. `r.eval('x<-c(1,2,3)')` first
then `r.eval('mean(x)')` works
But y= `r.eval('c(3,2,1)')` first
then `r.eval('mean(y)')` won't work
because "y" is defined in Sage but not in R

`x=r.eval('c(1,7,30,4,5,40)')` inputting by hand
`y=r.eval('seq(from=0,to=12,by=2)')` y is even from 0 to 12
`z=r.eval('1:10')` z goes from 1 to 10 by 1

For more data types: `http://www.statmethods.net/input/datatypes.html`

Generating random numbers from distributions

`r.eval('runif(n, min=0, max=1)')` generates n random numbers from uniform distr. with upper and lower limits; finite
`r.rnorm(n, mean=0, sd=1)` from Gaussian dist.
`r.rgamma(n, shape, scale=1)` from gamma dist.
`r.rbinom(n,size,prob)` from binominal dist.
`r.rnbinom(n,size,prob)` from negative binominal dist.
`r.rexp(n, rate=1)` from exponential dist.
`r.rgeom(n, prob)` from geometric dist.
`r.rwilcox(nn, m, n)` from Wilcoxon statistics

Graphics in R

***Note**: For all plotting functions, need: `png()` at beginning and `.silent.me. <- dev.off()` at end for using `%r` mode in notebook; else need: `r.png()` before graphic function and variables to be plotted and `r.dev_off()` at end to print graphic.

General graphics

`r.plot(x,y,main="title",xlab="label",ylab="label"`
`type="p")` type: "p" points, "l" lines, "b" both
`r.hist(x)` histogram of x
`r.hist(x, breaks=n)` histogram n breaks for bin size
`r.hist(x, freq=F)` density scale histogram
`r.boxplot(x)` boxplot of x
`r.boxplot(x,y)` boxplot x,y
`r.qqnorm(x)` QQ plot of x to normal distribution
`r.qqline(x)` adds line to normal QQ plot
`r.qqline(x, y)` produces QQ plot of two datasets

For advanced graphics in R: `http://www.statmethods.net/advgraphs/index.html`

Basic R Statistical Functions

`r.mean(x)` arithmetic mean of x
`r.median(x, na.rm=F)` median of x, NA not removed
`%r; freq <- table(x);`
`as.numeric(names(freq)[which.max(freq)])`
returns mode of x
Warning: `r.mode(x)` will return the type or storage of an object, not the actual mode value itself
`r.quartile(x)` min and max of x
`r.range(x)` min and max of x
`r.var(x)` variance of x
`r.sd(x, na.rm=F)` std. deviation of x, NA not removed

Correlation in R

Regression

`r.plot(x,y)` scatterplot of x and y
`r.cor(x,y)` correlation of x and y
`r.eval('lm(y~x)')` regression fit, shows intercept and slope
`r.eval('abline(lm(y~x))')` draws the fit on scatterplot
`r.eval('summary(lm(y~x))')` gives residuals, coefficients, std. error, R-squared, F-stats, p-value

Goodness-of-fit

`lm1= r.eval('(lm(y~x)')` assigns regression to lm.1
`y_pred= r.eval('lm1$fitted.values)')` stored predicted y
`r.plot(y,y_pred)` plots actual y and predicted y values

Residuals

`r.plot(y_pred, lm1$residuals)` plotting residuals
`r.cor(ly_pred, lm1$residuals)` residuals correlation

Statistical Testing in R

Confidence Interval

`r.eval('t.test(x)$conf.int[1:2]')` shows two-sided 0.95 CI of x

One Sample t-test

`r.eval('t.test(x,alternative="two.sided",conf.=0.95)')`
0.95 t-test for x, alternative also: "less" or "greater"

Two Sample t-test

`r.eval('t.test(x,y,alternative="two.sided",conf.=0.95)')`
0.95 t-test for (x,y), alternative also: "less" or "greater"

ANOVA in R

`r.eval('summary(aov(y~x))')` summary of ANOVA
`r.eval('TukeyHSD(aov(y~x))')` Tukey's Test
x must be a factor; use `as.factor(x)` if needed

Many more statistical testings in R, including multiple regression, Power Analysis, and MANOVA can be found at: `http://www.statmethods.net/stats/index.html`.

Statistical testings can also be done in SciPy and can be found at: `http://docs.scipy.org/doc/scipy/reference/stats.html#plot-tests`.