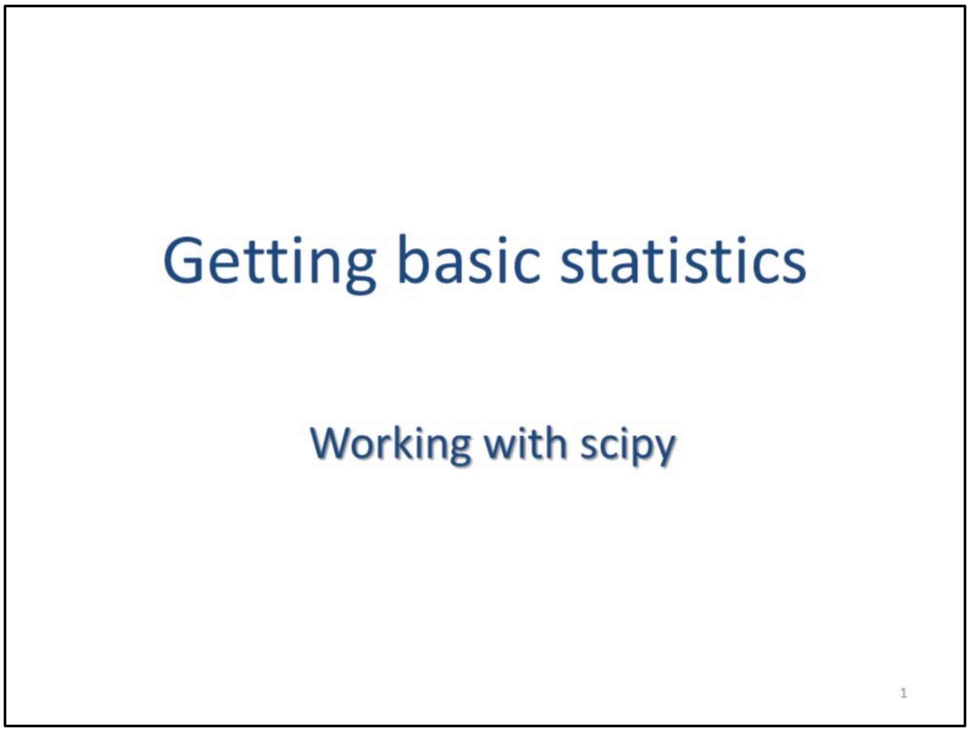
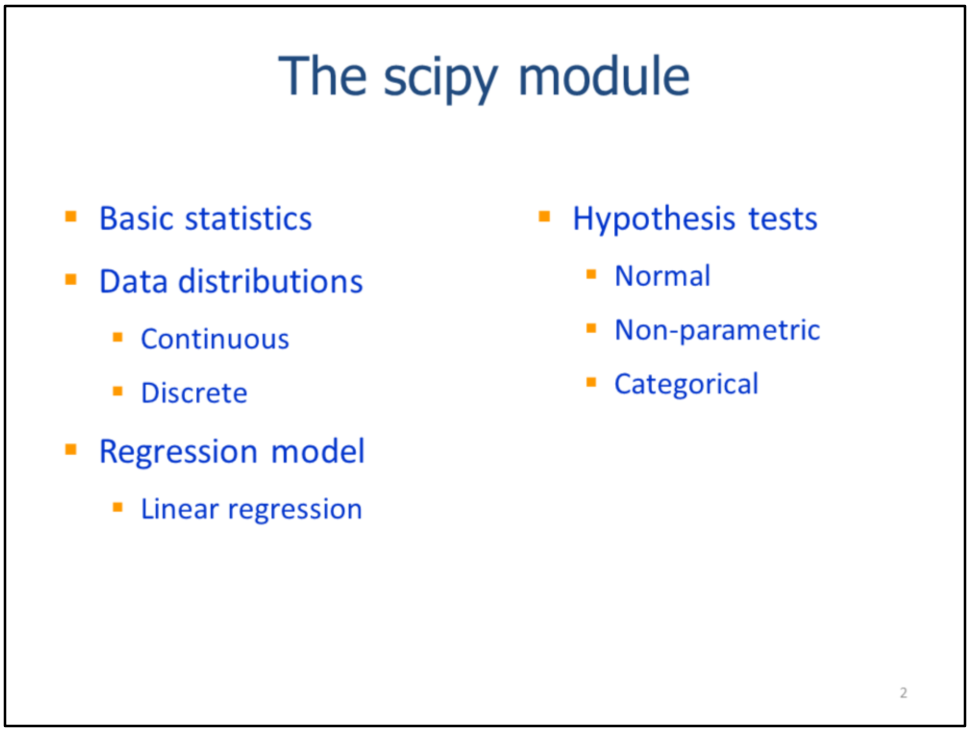


Getting basic statistics

Working with scipy

1

This video will present scipy and discuss how to obtain descriptive statistics for a dataset using scipy.

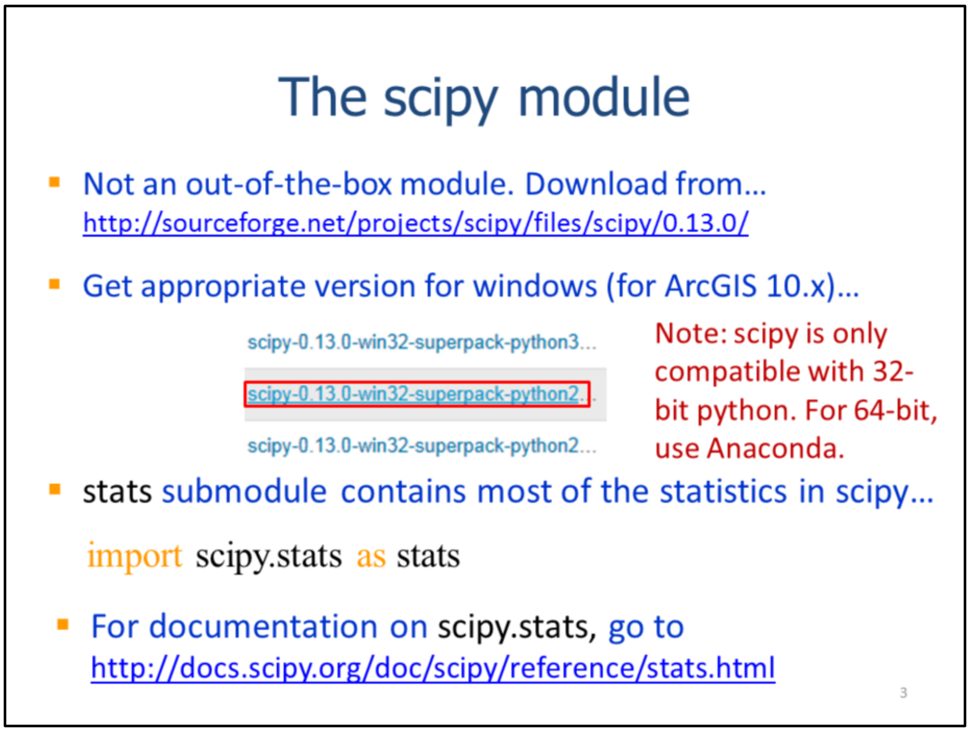


The scipy module

- Basic statistics
- Data distributions
  - Continuous
  - Discrete
- Regression model
  - Linear regression
- Hypothesis tests
  - Normal
  - Non-parametric
  - Categorical

2

The scipy module contains many statistical tools – in the next few videos, we'll look at a number of scipy tools that calculate basic statistics, create random distributions, perform linear regression, and conduct hypothesis tests. This week's lectures will focus on running the statistical tools in scipy but will only provide general information about when to use the tools. Statistical tools can give misleading results if used incorrectly – so make sure to get the proper statistical background or consult a statistical advisor when deciding on how to conduct a statistical analysis.


The scipy module

- Not an out-of-the-box module. Download from…
  http://sourceforge.net/projects/scipy/files/scipy/0.13.0/
- Get appropriate version for windows (for ArcGIS 10.x)…

  scipy-0.13.0-win32-superpack-python3…
  scipy-0.13.0-win32-superpack-python2…
  scipy-0.13.0-win32-superpack-python2…

  Note: scipy is only compatible with 32-bit python. For 64-bit, use Anaconda.
- stats submodule contains most of the statistics in scipy…

```
import scipy.stats as stats
```

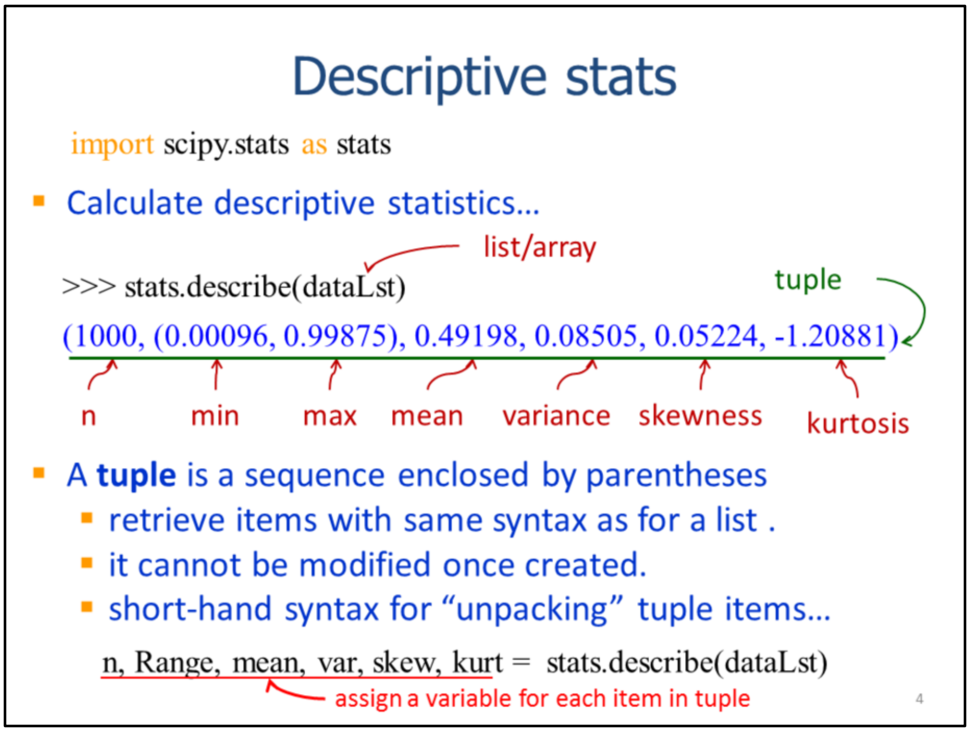
- For documentation on scipy.stats, go to
  http://docs.scipy.org/doc/scipy/reference/stats.html

3

The scipy module is not an out-of-the-box module and needs to be downloaded in order to be installed. The scipy installer can be downloaded for 32-bit Python. Note that an installer does not currently exist for 64-bit Python with Windows. Scipy can be run through Anaconda which is a scientific Python package that can run on either 32- or 64-bit windows.

The lecture videos will focus on scipy's stats module which contains many basic statistical tools.

Note that scipy has far more capabilities than we'll be able to discuss in this course. See the link provided here for documentation for scipy and the stats module.



Descriptive stats

```
import scipy.stats as stats
```

- Calculate descriptive statistics…

```
>>> stats.describe(dataLst)
(1000, (0.00096, 0.99875), 0.49198, 0.08505, 0.05224, -1.20881)
```

list/array
tuple
n
min
max
mean
variance
skewness
kurtosis

- A **tuple** is a sequence enclosed by parentheses
  - retrieve items with same syntax as for a list .
  - it cannot be modified once created.
  - short-hand syntax for “unpacking” tuple items…

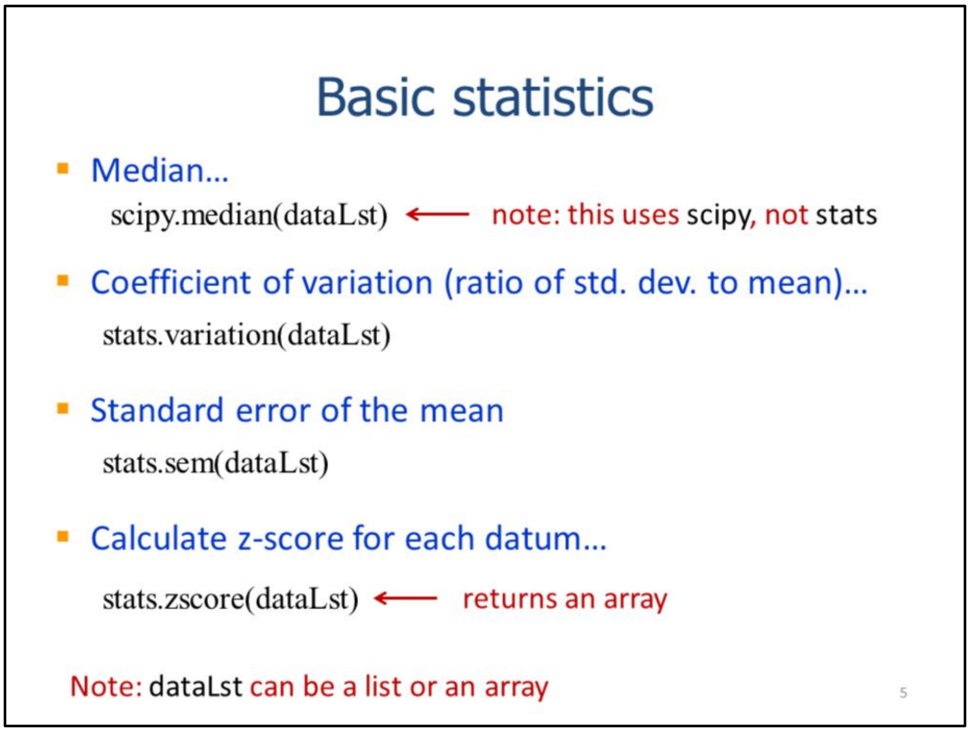
```
n, Range, mean, var, skew, kurt = stats.describe(dataLst)
```

The stats sub-module’s **describe** function returns basic descriptive statistics for a dataset.

Note that the statistics are returned as a **tuple**. Note that **n** is the number of data points in the dataset.

A tuple is a read-only data collection that cannot be modified. Values are retrieved from a tuple in the same way that they’re retrieved from a list.

The values from the **describe** function can be “unpacked” by specifying variables for each item in the tuple.



Basic statistics

- Median…

  scipy.median(dataLst) ⟵ note: this uses scipy, not stats

- Coefficient of variation (ratio of std. dev. to mean)…

  stats.variation(dataLst)

- Standard error of the mean

  stats.sem(dataLst)

- Calculate z-score for each datum…

  stats.zscore(dataLst) ⟵ returns an array

Note: dataLst can be a list or an array

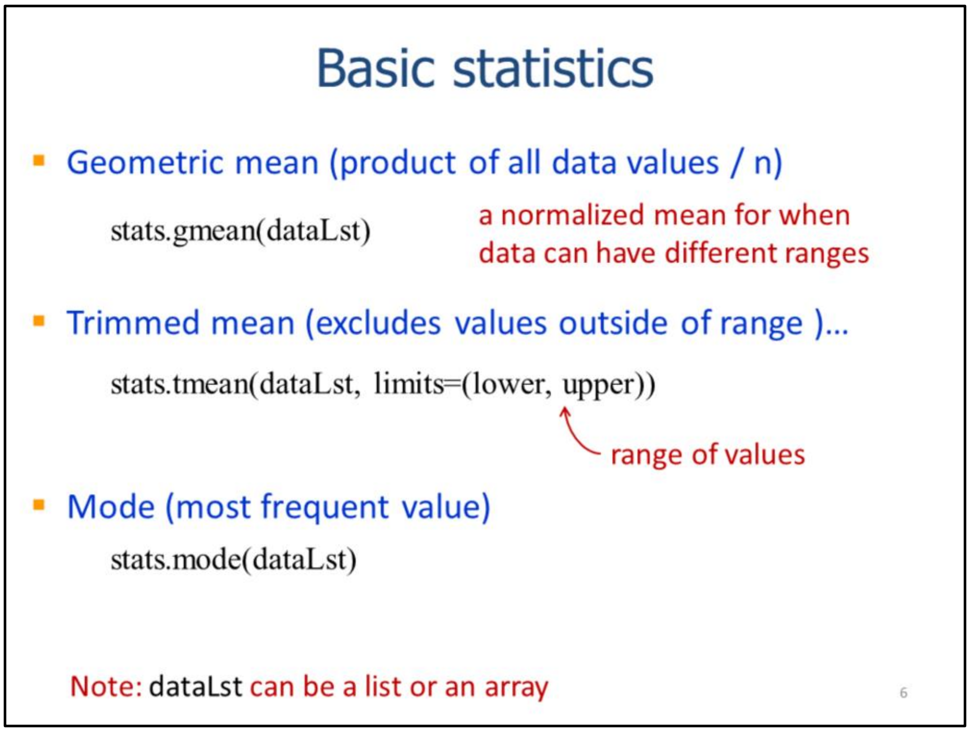
5

The median for a dataset can be calculated with scipy’s **median** function.

The stats submodule’s **variation** function calculates the coefficient of variation which is the ratio of the standard deviation to the mean. This coefficient can be used to compare standard deviations for datasets that different ranges of values.

The standard error of the mean can be calculated using stats’ **sem** tool. The standard error is often used to create error bars in a plot.

Stats’ zscore function calculates the z-score value for each datum in the dataset. Zscore assumes a normally distributed dataset and indicates how many standard deviations away from the mean the data point is located.



Basic statistics

- Geometric mean (product of all data values / n)

```
stats.gmean(dataLst)
```

a normalized mean for when data can have different ranges

- Trimmed mean (excludes values outside of range)...

```
stats.tmean(dataLst, limits=(lower, upper))
```

range of values

- Mode (most frequent value)

```
stats.mode(dataLst)
```

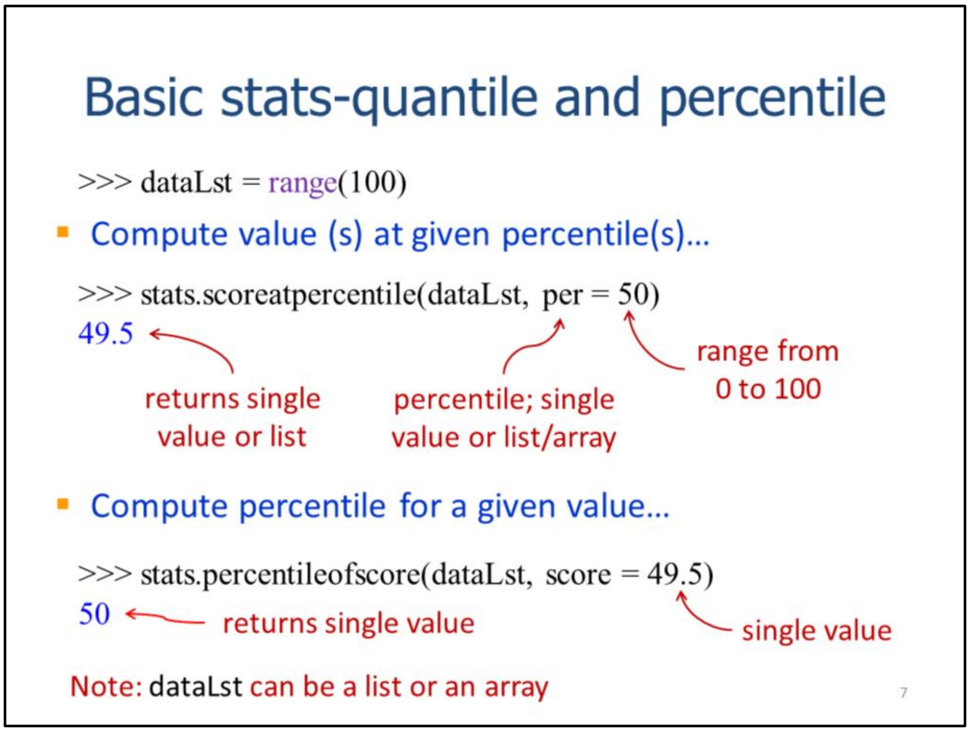
Note: dataLst can be a list or an array

6

The **gmean** tool calculates the geometric mean which can be used when comparing the means of multiple datasets that have different ranges of values.

The **tmean** tool calculates the trimmed mean which excludes values outside the specified range. The trimmed mean can be used to exclude outliers from the mean.

The mode is the most frequently occurring value in the dataset.


Basic stats-quantile and percentile

```
>>> dataLst = range(100)
```

- Compute value (s) at given percentile(s)…

```
>>> stats.scoreatpercentile(dataLst, per = 50)
49.5
```

returns single value or list
percentile; single value or list/array
range from 0 to 100

- Compute percentile for a given value…

```
>>> stats.percentileofscore(dataLst, score = 49.5)
50
```

returns single value
single value

Note: dataLst can be a list or an array

7

The **scoreatpercentile** will return the value in the dataLst that corresponds to the percentile value specified. Note that the value returned here is not in the original dataset - a value will be interpolated if the percentile falls between two data points. The value at the 50th percentile is the median. The values at the 25th and 75th percentile are used to create the lower and upper sides of a boxplot.

The **percentileofscore** gives the percentile for the specified value - this value does not need to be in the dataset.