Measures of Central Tendency

By Evgenia "Jenny" Nitishinskaya, Maxwell Margenot, and Delaney Granizo-Mackenzie.

Part of the Quantopian Lecture Series:

- www.quantopian.com/lectures (https://www.quantopian.com/lectures)
- github.com/quantopian/research_public (https://github.com/quantopian/research_public)

In this notebook we will discuss ways to summarize a set of data using a single number. The goal is to capture information about the distribution of data.

Arithmetic mean

The arithmetic mean is used very frequently to summarize numerical data, and is usually the one assumed to be meant by the word "average." It is defined as the sum of the observations divided by the number of observations:

$$\mu = \frac{\sum_{i=1}^N X_i}{N}$$

where $X_1, X_2, \ldots, X_N$ are our observations.

In [1]:
```
# Two useful statistical libraries
import scipy.stats as stats
import numpy as np

# We'll use these two data sets as examples
x1 = [1, 2, 2, 3, 4, 5, 5, 7]
x2 = x1 + [100]

print 'Mean of x1:', sum(x1), '/', len(x1), '=', np.mean(x1)
print 'Mean of x2:', sum(x2), '/', len(x2), '=', np.mean(x2)
```

```
Mean of x1: 29 / 8 = 3.625
Mean of x2: 129 / 9 = 14.3333333333
```

We can also define a *weighted* arithmetic mean, which is useful for explicitly specifying the number of times each observation should be counted. For instance, in computing the average value of a portfolio, it is more convenient to say that 70% of your stocks are of type X rather than making a list of every share you hold.

The weighted arithmetic mean is defined as

$$\sum_{i=1}^n w_i X_i$$

where $\sum_{i=1}^n w_i = 1$. In the usual arithmetic mean, we have $w_i = 1/n$ for all i.

Median

The median of a set of data is the number which appears in the middle of the list when it is sorted in increasing or decreasing order. When we have an odd number n of data points, this is simply the value in position $(n+1)/2$. When we have an even number of data points, the list splits in half and there is no item in the middle; so we define the median as the average of the values in positions $n/2$ and $(n+2)/2$.

The median is less affected by extreme values in the data than the arithmetic mean. It tells us the value that splits the data set in half, but not how much smaller or larger the other values are.

In [2]:
```
print 'Median of x1:', np.median(x1)
print 'Median of x2:', np.median(x2)
```

```
Median of x1: 3.5
Median of x2: 4.0
```

Mode

The mode is the most frequently occuring value in a data set. It can be applied to non-numerical data, unlike the mean and the median. One situation in which it is useful is for data whose possible values are independent. For example, in the outcomes of a weighted die, coming up 6 often does not mean it is likely to come up 5; so knowing that the data set has a mode of 6 is more useful than knowing it has a mean of 4.5.

In [3]:
```
# Scipy has a built-in mode function, but it will return exactly one value
# even if two values occur the same number of times, or if no value appears more than once
print 'One mode of x1:', stats.mode(x1)[0][0]

# So we will write our own
def mode(l):
    # Count the number of times each element appears in the list
    counts = {}
    for e in l:
        if e in counts:
            counts[e] += 1
        else:
            counts[e] = 1

    # Return the elements that appear the most times
    maxcount = 0
    modes = {}
    for (key, value) in counts.iteritems():
        if value > maxcount:
            maxcount = value
            modes = {key}
        elif value == maxcount:
            modes.add(key)

    if maxcount > 1 or len(l) == 1:
        return list(modes)
    return 'No mode'

print 'All of the modes of x1:', mode(x1)
```

```
One mode of x1: 2
All of the modes of x1: [2, 5]
```

For data that can take on many different values, such as returns data, there may not be any values that appear more than once. In this case we can bin values, like we do when constructing a histogram, and then find the mode of the data set where each value is replaced with the name of its bin. That is, we find which bin elements fall into most often.

In [4]:
```
# Get return data for an asset and compute the mode of the data set
start = '2014-01-01'
end = '2015-01-01'
pricing = get_pricing('SPY', fields='price', start_date=start, end_date=end)
returns = pricing.pct_change()[1:]
print 'Mode of returns:', mode(returns)

# Since all of the returns are distinct, we use a frequency distribution to get an alternative mode.
# np.histogram returns the frequency distribution over the bins as well as the endpoints of the bins
hist, bins = np.histogram(returns, 20) # Break data up into 20 bins
maxfreq = max(hist)
# Find all of the bins that are hit with frequency maxfreq, then print the intervals corresponding to them
print 'Mode of bins:', [(bins[i], bins[i+1]) for i, j in enumerate(hist) if j == maxfreq]
```

```
Mode of returns: No mode
Mode of bins: [(-0.0012151920594887407, 0.0011015927020092131)]
```

Geometric mean

While the arithmetic mean averages using addition, the geometric mean uses multiplication:

$$G = \sqrt[n]{X_1 X_1 \ldots X_n}$$

for observations $X_i \geq 0$. We can also rewrite it as an arithmetic mean using logarithms:

$$\ln G = \frac{\sum_{i=1}^n \ln X_i}{n}$$

The geometric mean is always less than or equal to the arithmetic mean (when working with nonnegative observations), with equality only when all of the observations are the same.

In [5]:
```
# Use scipy's gmean function to compute the geometric mean
print 'Geometric mean of x1:', stats.gmean(x1)
print 'Geometric mean of x2:', stats.gmean(x2)
```

```
Geometric mean of x1: 3.09410402498
Geometric mean of x2: 4.55253458762
```

What if we want to compute the geometric mean when we have negative observations? This problem is easy to solve in the case of asset returns, where our values are always at least -1. We can add 1 to a return R_t to get $1 + R_t$, which is the ratio of the price of the asset for two consecutive periods (as opposed to the percent change between the prices, R_t). This quantity will always be nonnegative. So we can compute the geometric mean return,

$$R_G = \sqrt[T]{(1 + R_1)\dots(1 + R_T)} - 1$$

In [6]:
```
# Add 1 to every value in the returns array and then compute R_G
ratios = returns + np.ones(len(returns))
R_G = stats.gmean(ratios) - 1
print 'Geometric mean of returns:', R_G
```

```
Geometric mean of returns: 0.000463575391327
```

The geometric mean is defined so that if the rate of return over the whole time period were constant and equal to R_G, the final price of the security would be the same as in the case of returns $R_1, \dots, R_T$.

In [7]:
```
T = len(returns)
init_price = pricing[0]
final_price = pricing[T]
print 'Initial price:', init_price
print 'Final price:', final_price
print 'Final price as computed with R_G:', init_price*(1 + R_G)**T
```

```
Initial price: 182.95
Final price: 205.52
Final price as computed with R_G: 205.52
```

Harmonic mean

The harmonic mean is less commonly used than the other types of means. It is defined as

$$H = \frac{n}{\sum_{i=1}^n \frac{1}{X_i}}$$

As with the geometric mean, we can rewrite the harmonic mean to look like an arithmetic mean. The reciprocal of the harmonic mean is the arithmetic mean of the reciprocals of the observations:

$$\frac{1}{H} = \frac{\sum_{i=1}^n \frac{1}{X_i}}{n}$$

The harmonic mean for nonnegative numbers X_i is always at most the geometric mean (which is at most the arithmetic mean), and they are equal only when all of the observations are equal.

In [8]:
```
print 'Harmonic mean of x1:', stats.hmean(x1)
print 'Harmonic mean of x2:', stats.hmean(x2)
```

```
Harmonic mean of x1: 2.55902513328
Harmonic mean of x2: 2.86972365624
```

The harmonic mean can be used when the data can be naturally phrased in terms of ratios. For instance, in the dollar-cost averaging strategy, a fixed amount is spent on shares of a stock at regular intervals. The higher the price of the stock, then, the fewer shares an investor following this strategy buys. The average (arithmetic mean) amount they pay for the stock is the harmonic mean of the prices.

Point Estimates Can Be Deceiving

Means by nature hide a lot of information, as they collapse entire distributions into one number. As a result often 'point estimates' or metrics that use one number, can disguise large programs in your data. You should be careful to ensure that you are not losing key information by summarizing your data, and you should rarely, if ever, use a mean without also referring to a measure of spread.

Underlying Distribution Can be Wrong

Even when you are using the right metrics for mean and spread, they can make no sense if your underlying distribution is not what you think it is. For instance, using standard deviation to measure frequency of an event will usually assume normality. Try not to assume distributions unless you have to, in which case you should rigourously check that the data do fit the distribution you are assuming.

change and may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.