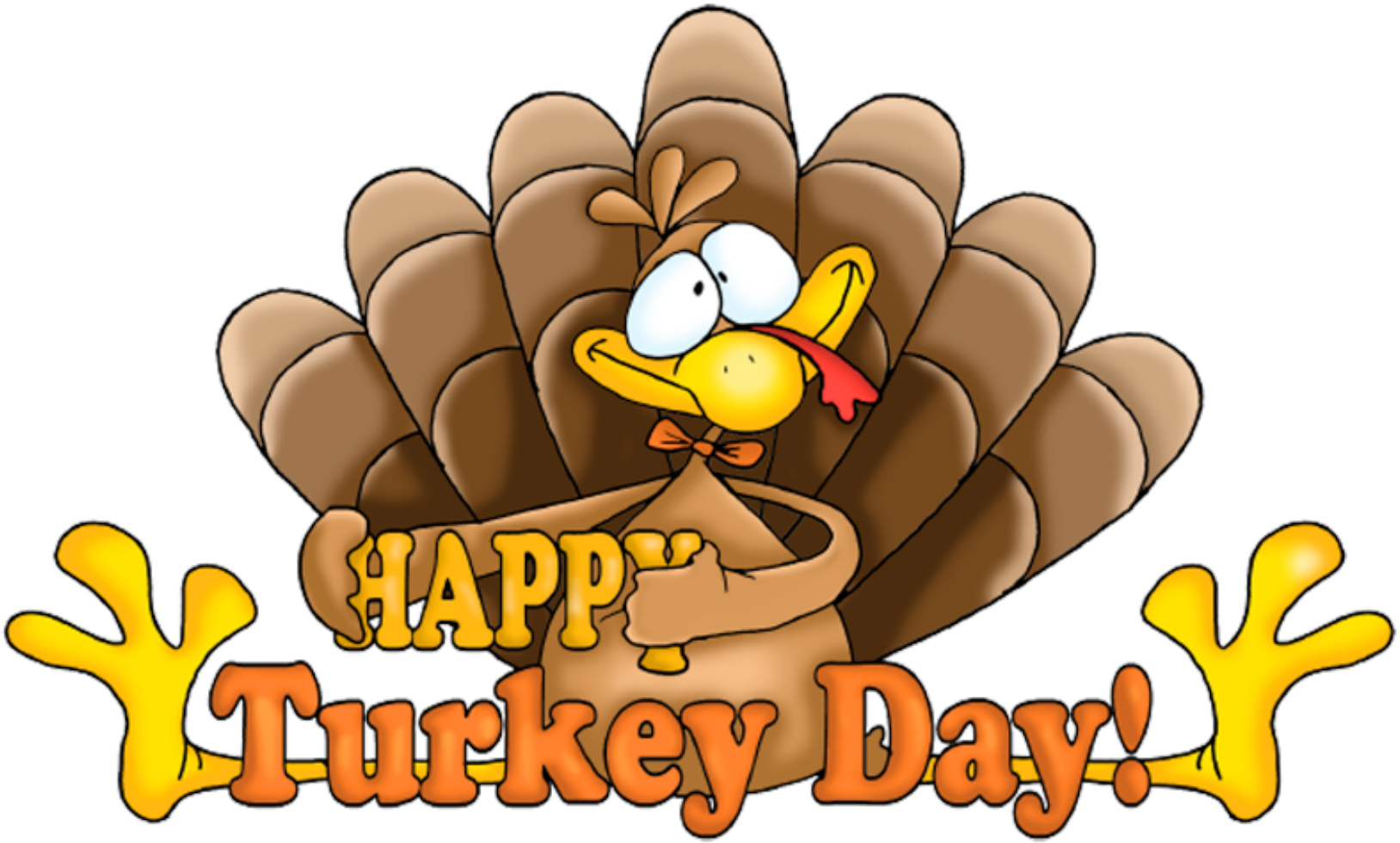
Sampling, Central Limit Theorem, and Standard Error

(download slides and .py files from Stellar to follow along)

Eric Grimson

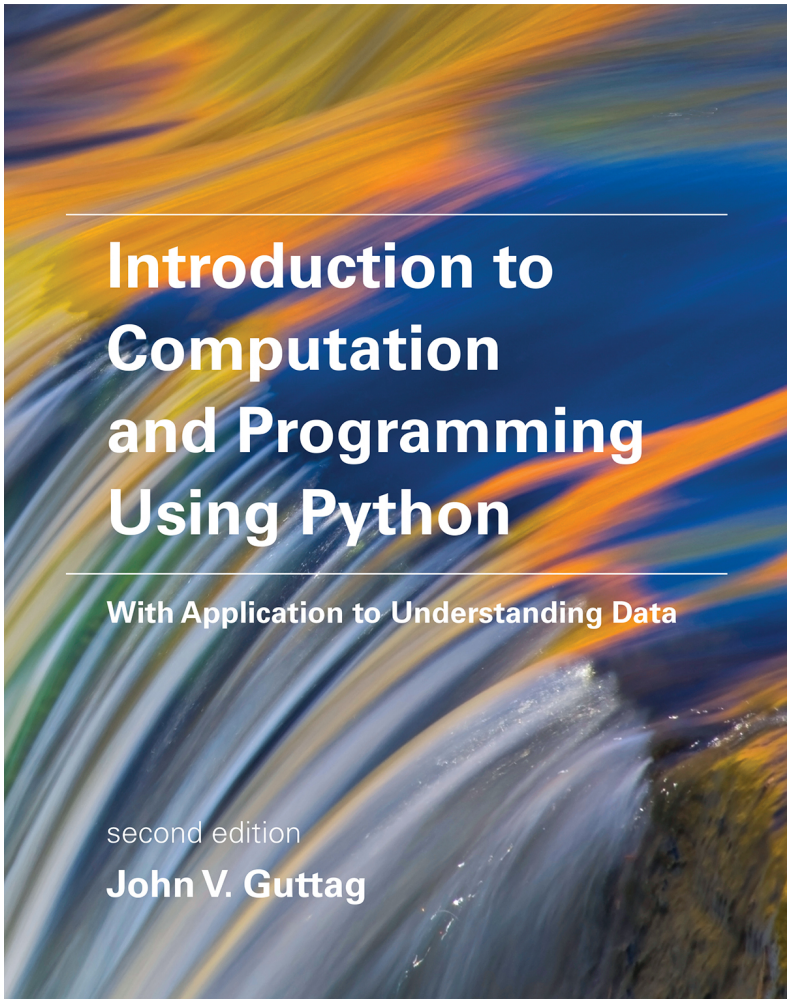
MIT Department Of Electrical Engineering and Computer Science

Remember: No Lecture on Wednesday



Assigned Reading

- Today:
  - Chapter 17
- Next lecture:
  - Chapter 18

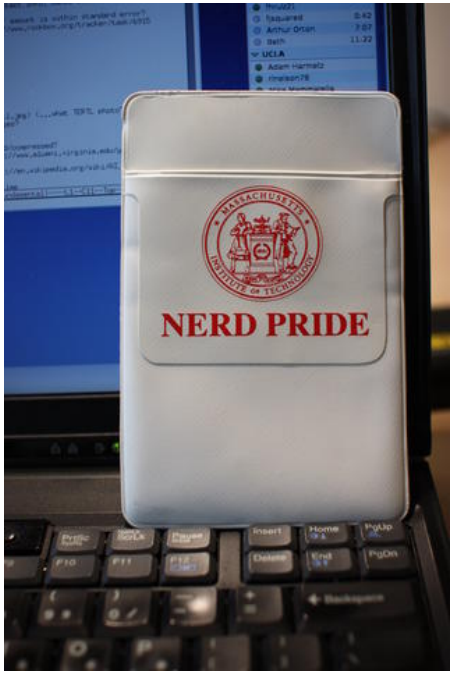


Recall Inferential Statistics

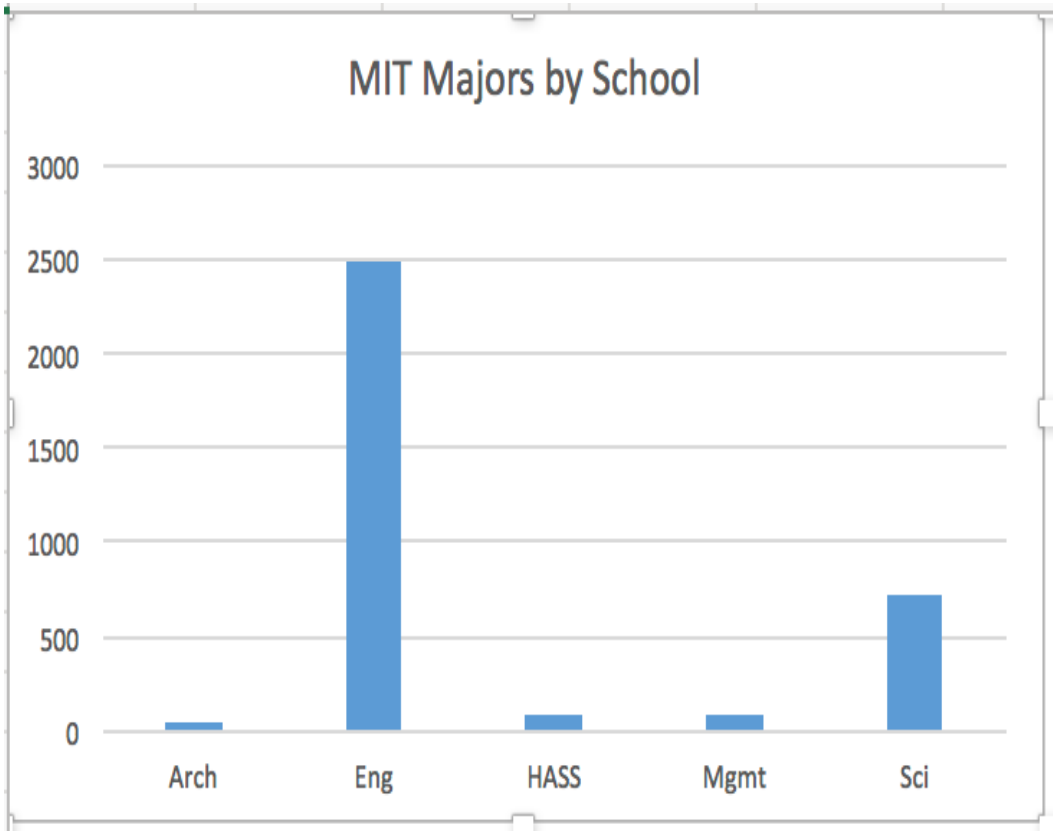
- Inferential statistics: making inferences about a population by examining one or more random samples drawn from that population
  - infer something about whole population based on statistics of sub-population
  - avoid cost of having to look at whole population (may not even be feasible?)
- With Monte Carlo simulation, we can generate lots of random samples and use them draw inferences and to compute confidence intervals about the inferences
  - allows us to estimate likelihood of inference
- But suppose we can't create samples by simulation?
  - "In a new poll, 42 % of voters approve of the job Trump is doing as president, while 53 percent disapprove. The poll surveyed 1,972 registered voters and has a margin of error of plus or minus 2 percentage points." – August 2017

Probability Sampling

- Alternative to simulation is to directly sample from population
- Assume each member of population has nonzero probability of being included in sample
  - Select subpopulation by sampling at random
- Simple random sampling: each member has an equal chance of being chosen
- Not always appropriate
  - Popular myth: all MIT undergraduates major in engineering or science
  - Consider a random sample of 100 students
    - What might you conclude about the myth from such a sample?



Probability Sampling



In random sample of 100 undergraduates, we expect:

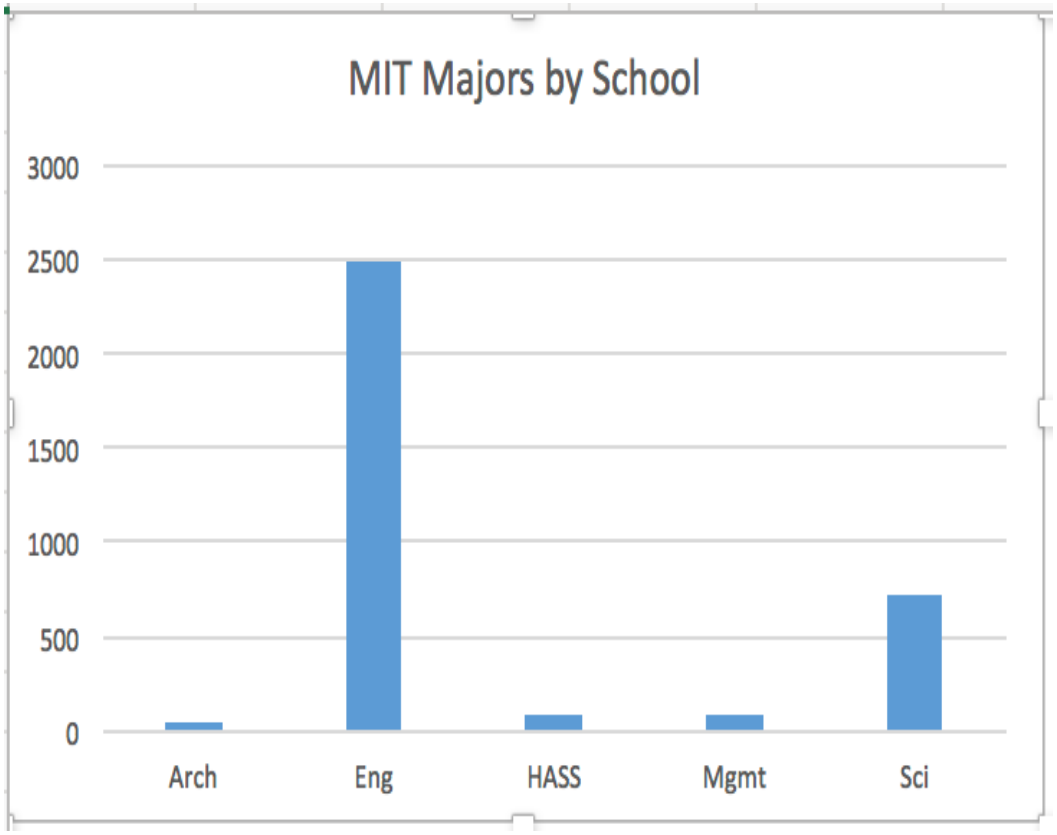
- 1 Architecture & Planning
- 72 Engineering
- 3 HASS
- 3 Management
- 21 Science

In simulated sampling over 10,000 trials, saw that distribution on average

But in simulated sampling over 10,000 trials

- 25% of trials have no SAP, 7% no SHASS, 8% no SLOAN
- **0.2% with only SOE and SOS**
- **so 2 of 1000 times, would conclude only Engineering & Science, but might conclude no SAP a quarter of the time**

Probability Sampling



In random sample of 100 undergraduates, we expect:

- 1 Architecture & Planning
- 72 Engineering
- 3 HASS
- 3 Management
- 21 Science

In simulated sampling over 10,000 trials, saw that distribution on average

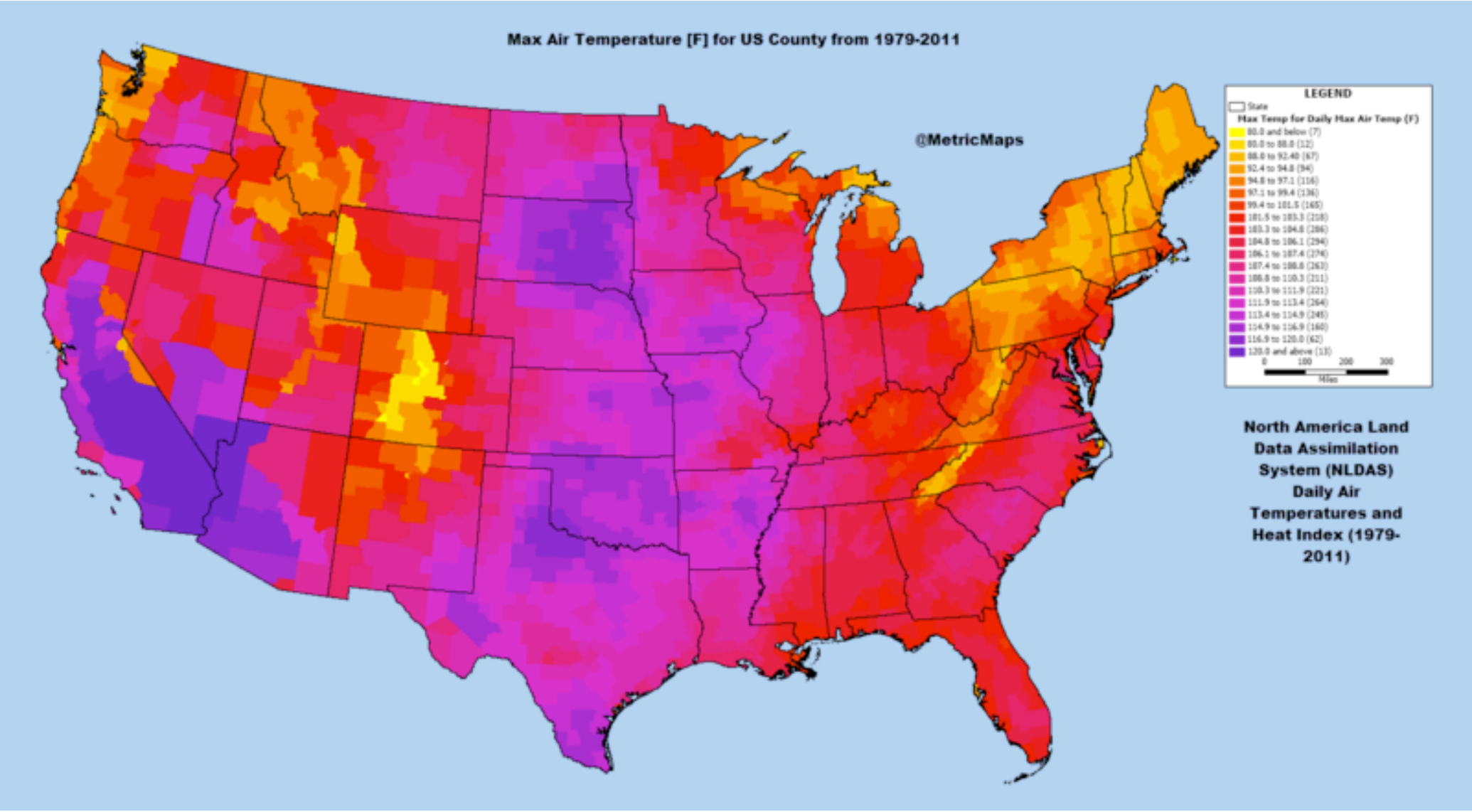
Suppose I had decided to only sample 1% (or 34) of the UG population over 10,000 trials

- 63% of trials have no SAP, 41% no SHASS, 41% no SLOAN
- **9.7% with only SOE and SOS**

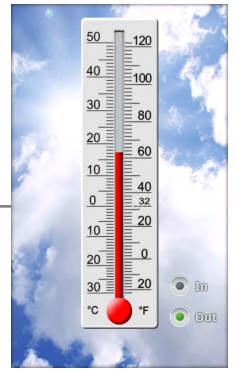
Stratified Sampling

- Stratified sampling
  - Partition population into subgroups
  - Take a simple random sample from each subgroup (size of sample depends on relative size of subgroup)
- Useful when there are small subgroups that should be represented (think about political polls)
- Useful when it is important that subgroups be represented proportionally to their size in the population
  - E.g. if want to get opinions from MIT UG population, for sample of 100 randomly pick 1 A&P student, 3 HASS students, 3 Sloan students, etc.
- Can be used to reduce the needed size of sample
  - Variability within subgroups often less than variability of entire population
- Requires care to do properly
  - How many samples to draw from each subgroup?
- **We'll stick to simple random samples**

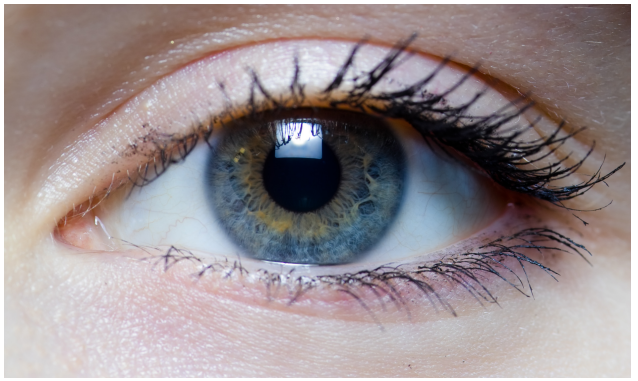
Using Sampling to Estimate Temperatures



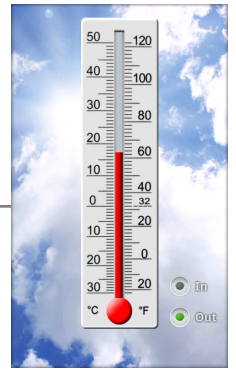
Data



- From U.S. National Centers for Environmental Information (NCEI)
- Daily high and low temperatures 1961-2015
  - 21 different US cities
    - ALBUQUERQUE, BALTIMORE, BOSTON, CHARLOTTE, CHICAGO, DALLAS, DETROIT, LAS VEGAS, LOS ANGELES, MIAMI, NEW ORLEANS, NEW YORK, PHILADELPHIA, PHOENIX, PORTLAND, SAN DIEGO, SAN FRANCISCO, SAN JUAN, SEATTLE, ST LOUIS, TAMPA
    - 421,848 data points (examples)
- Let's use some code to look at the data

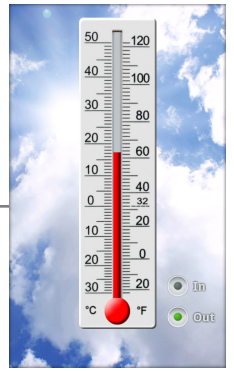


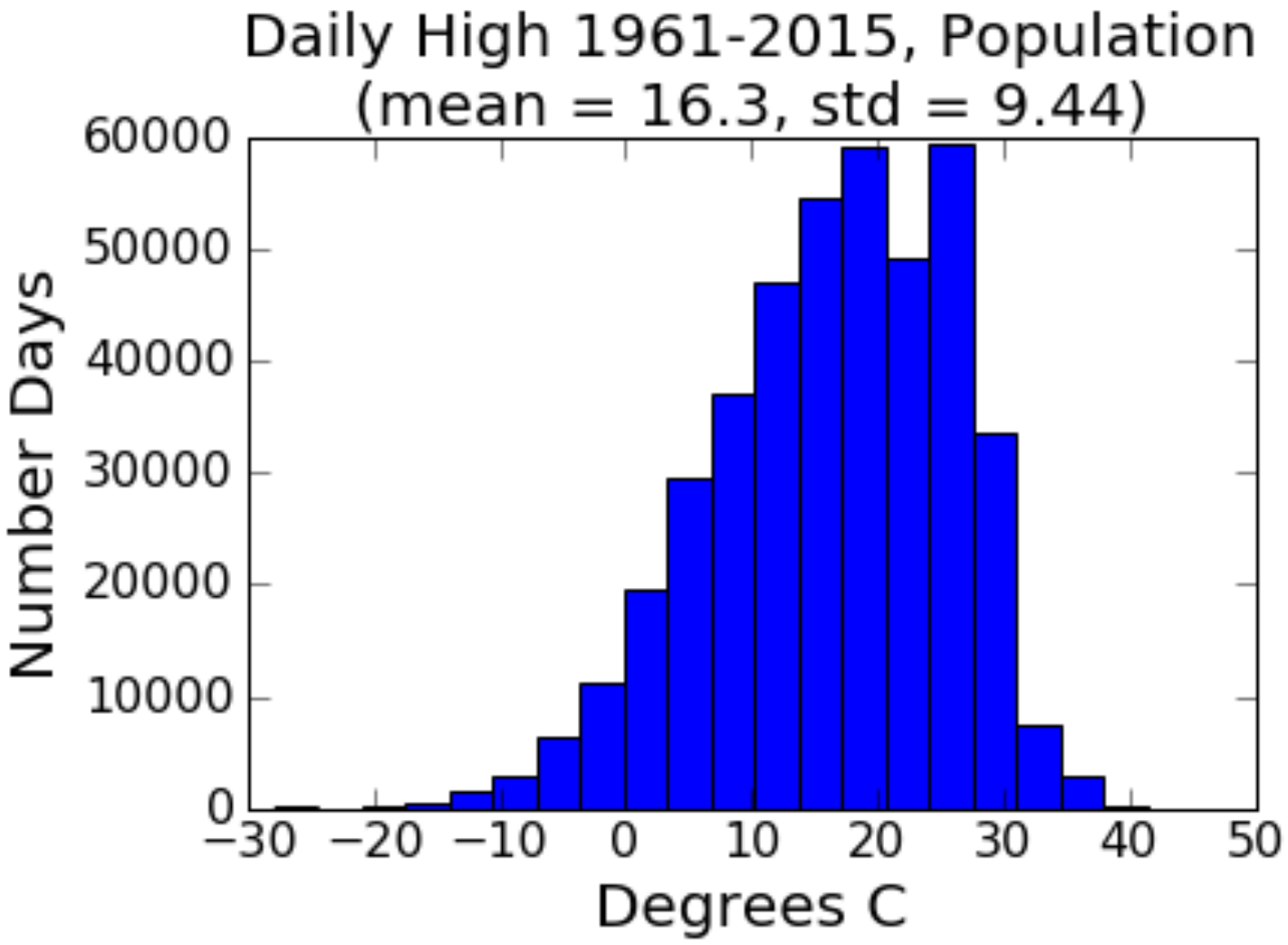
New in Code



- Code `getHighs` extracts high temperatures from file, code `getMeansAndSDs` computes mean and standard deviation for entire population and sample
- `numpy.std` is function in the `numpy` module that returns the standard deviation
- `random.sample(population, sampleSize)` returns a list containing `sampleSize` randomly chosen distinct elements of `population`
  - Sampling without replacement

Histogram of Entire Population





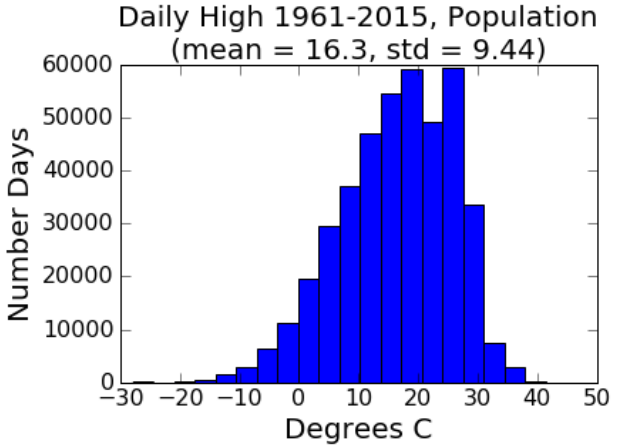
Or in °F:
61.3 +/-17.0

Why is
σ so large?

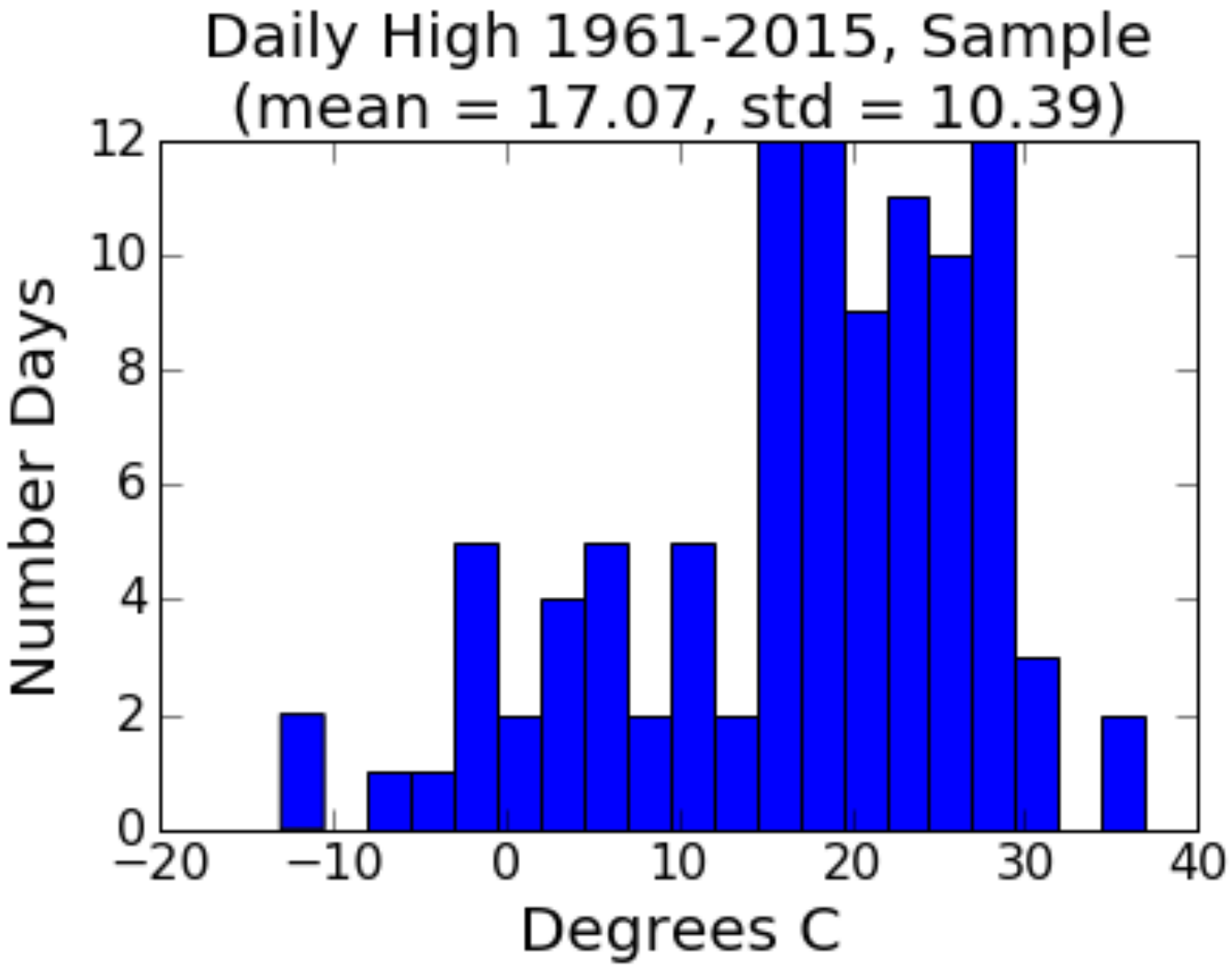
Is this really a normal distribution?

Some Observations

- Why large standard deviation:
  - Including many locations together
    - Weather in Phoenix different from Boston
  - Including weather from entire year
    - In Boston, weather in January not like weather in July
- Could look at data by month or location or year. But for now, let's just focus on mean temperature.
- Can we get a good approximation without looking at all the data?
- And this doesn't really look like a normal distribution. Can we infer something with confidence if this is not a normal distribution?
  - Remember that we used empirical rule to estimate confidence, and that assumes a normal distribution of errors



Histogram of Random Sample of Size 100



Looks even less like a normal distribution

Means and Standard Deviations

- Population mean = 16.3
- Sample mean = 17.1
- Standard deviation of population = 9.44
- Standard deviation of sample = 10.4
- A happy accident, or something we should expect?
- Let's try it 1000 times and plot the results

New in Code

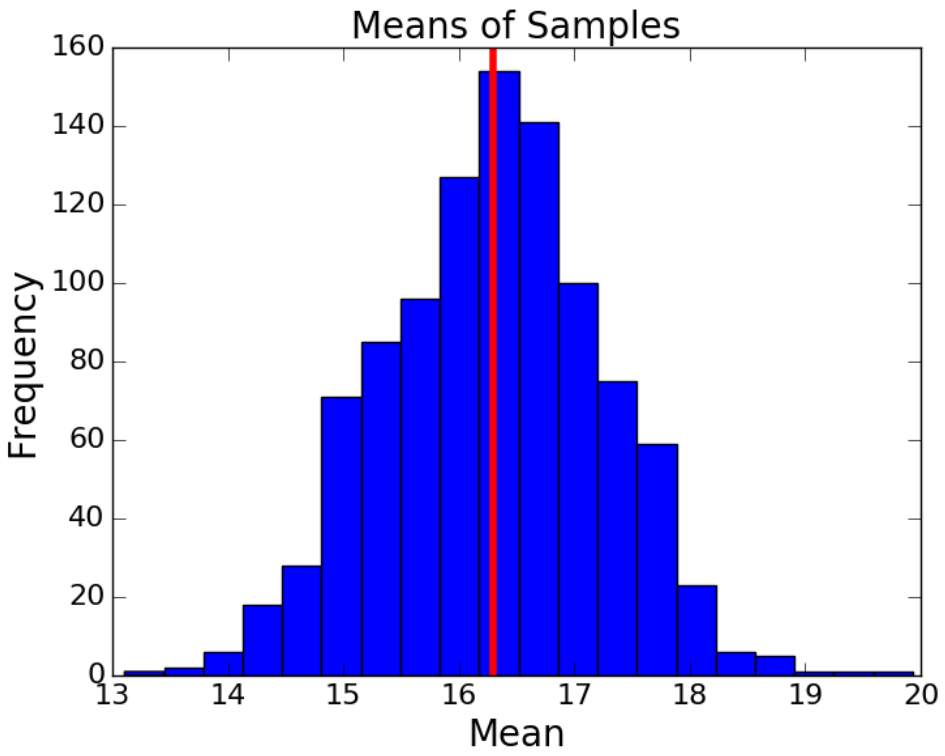
- Code in handout extracts sample of size `sampleSize`, computes mean, then repeats this for `numSamples` trials and computes the mean of those means

- `pylab.axvline(x = popMean, color = 'r')` draws a red vertical line at popMean on the x-axis
- There's also a `pylab.axhline` function

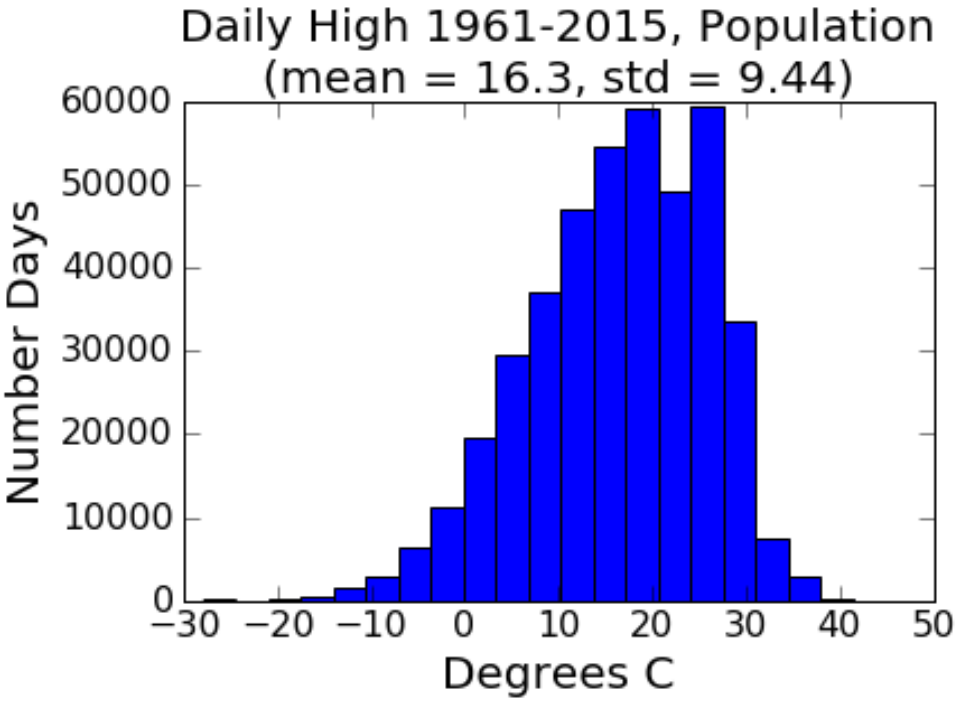
Try It 1000 Times

- Draw a sample of 100 measurements from entire set of measurements
- Compute mean of that sample
- Repeat this sampling process 1000 times (each trial with a different sample of 100 data points)
  - Record the mean of each sample trial
- Compute the **mean of the means**

Try It 1000 Times



±



In comparison

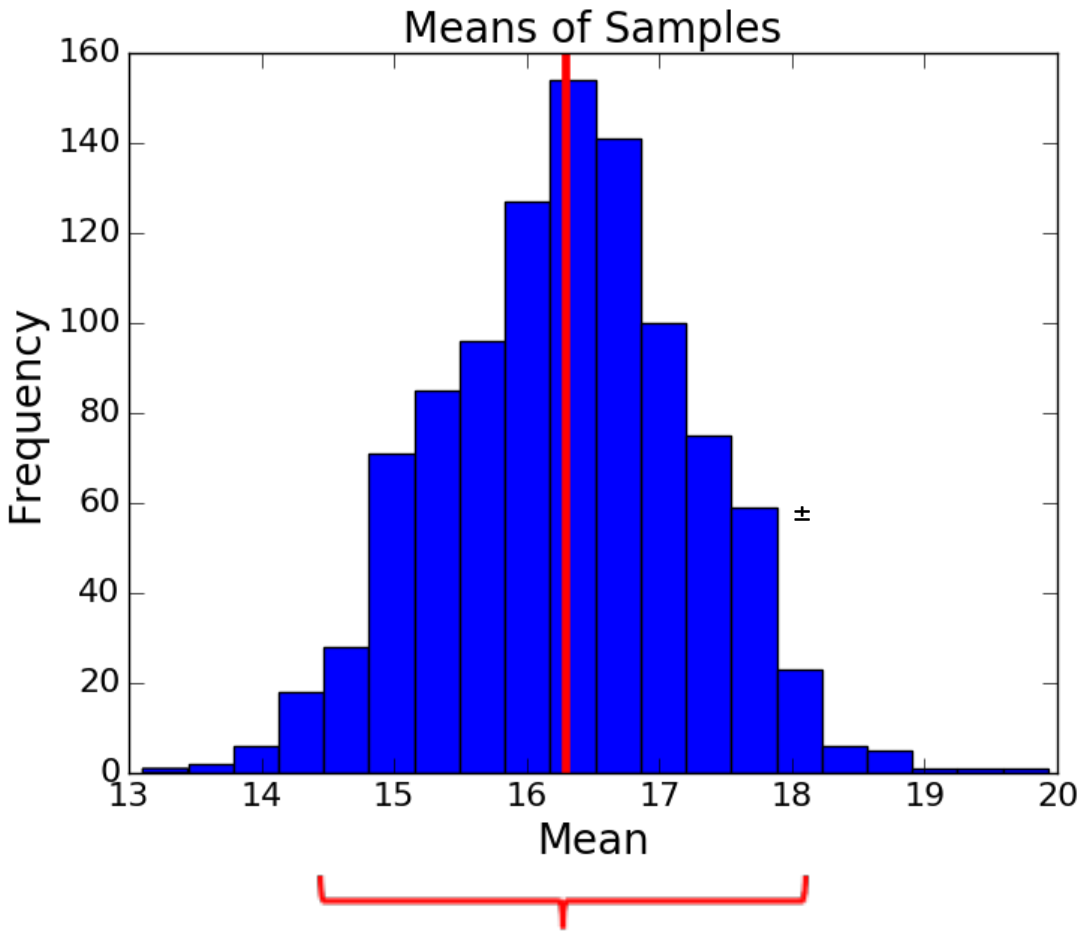
Mean high temperature over all samples

Mean of sample Means = 16.3

Standard deviation of sample means = 0.94

Mean of mean high temperatures, 100 samples, 1000 trials

Try It 1000 Times



Mean of sample Means = 16.3
Standard deviation of sample means = 0.94

What's the 95% confidence interval?

16.28 +/- 1.96*0.94
Or 14.5 - 18.1

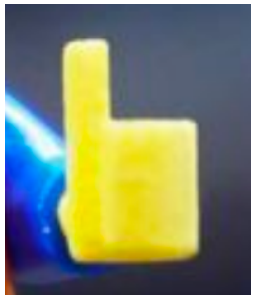
Includes population mean, but pretty wide interval

Suppose we want a tighter bound?

Getting a Tighter Bound

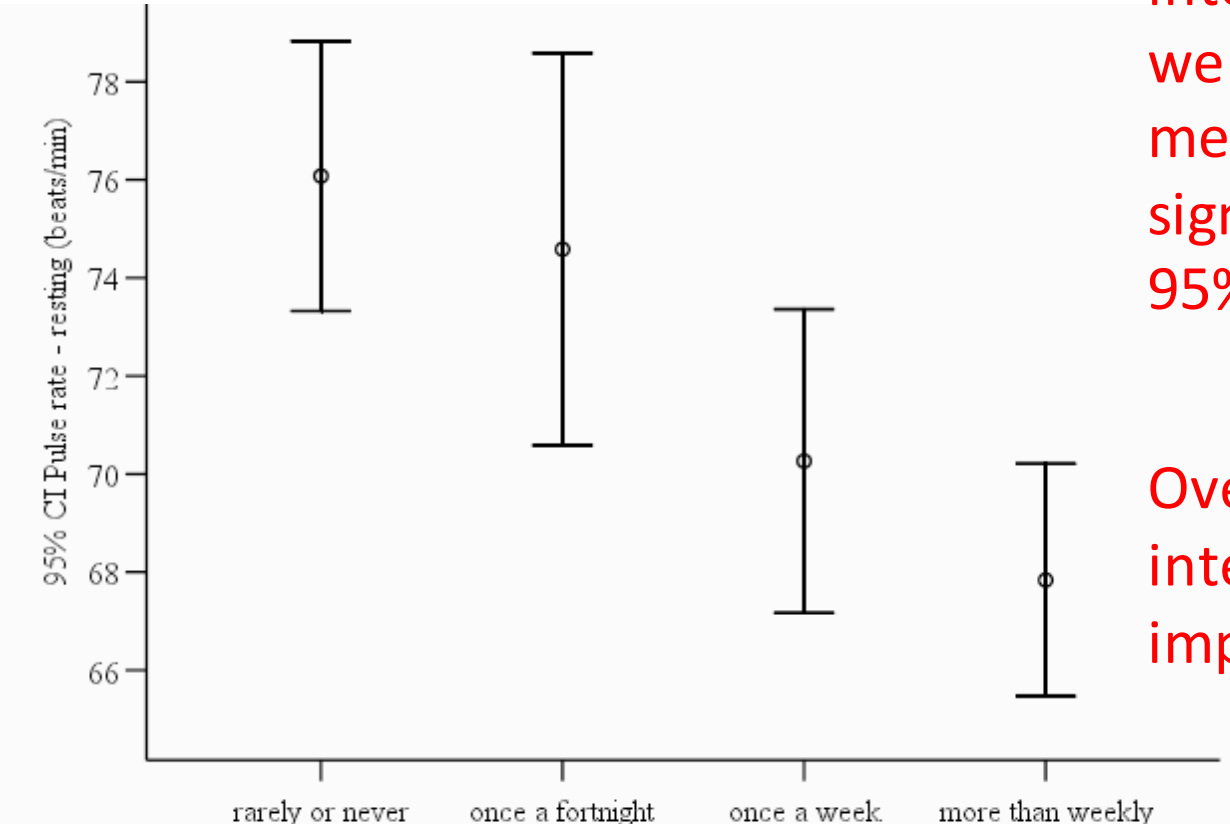
- Will drawing more samples help?
  - Let's try increasing from 1000 to 2000 trials
  - Standard deviation of estimated mean temperature goes from 0.943 to 0.946

- How about larger samples?
  - Let's try increasing sample size from 100 to 200, but sticking with 1000 trials
  - Standard deviation of estimated mean temperature goes from 0.943 to 0.662



Error Bars, a Digression

- Graphical representation of the variability of data
- Way to visualize uncertainty



When confidence intervals don't overlap, we can conclude that means are statistically significantly different at 95% level.

Overlapping confidence intervals does not imply lack of significance.

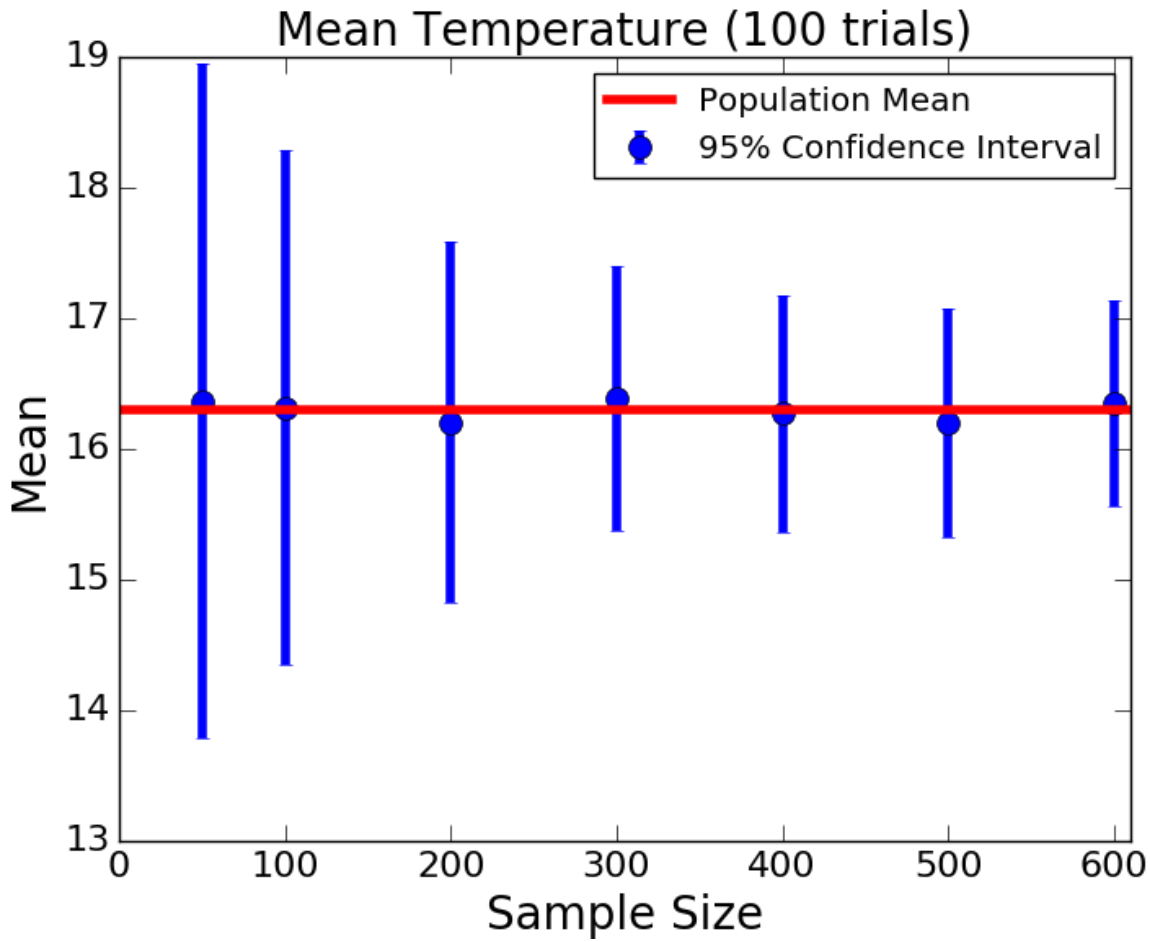
https://upload.wikimedia.org/wikipedia/commons/1/1d/Pulse_Rate_Error_Bar_By_Exercise_Level.png

Let's Look at Error Bars for Temperatures

```
def showErrorBars(population, sizes, numTrials):
    xVals = []
    sizeMeans, sizeSDs = [], []
    for sampleSize in sizes:
        xVals.append(sampleSize)
        trialMeans = []
        for t in range(numTrials):
            sample = random.sample(population, sampleSize)
            popMean, sampleMean, popSD, sampleSD =\
               getMeansAndSDs(population, sample)
            trialMeans.append(sampleMean)
        sizeMeans.append(sum(trialMeans)/len(trialMeans))
        sizeSDs.append(numpy.std(trialMeans))
    pylab.errorbar(xVals, sizeMeans,
                   yerr = 1.96*pylab.array(sizeSDs),
                   fmt = 'o',
                   label = '95% Confidence Interval')
```

Sample Size and Standard Deviation

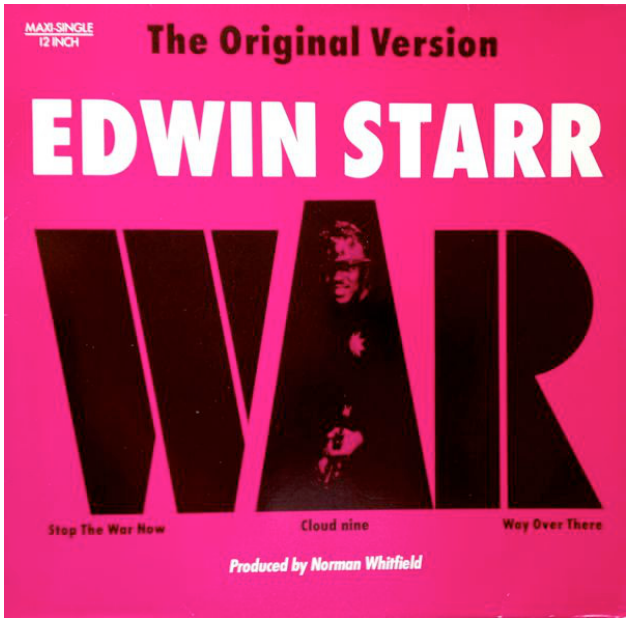
```
population = getHighs()
showErrorBars(population,
              (50, 100, 200, 300, 400, 500, 600), 100)
```



Note how sample means are close to actual mean of entire population

Larger Samples Seem to Be Better

- Going from a sample size of 50 to 600 reduced the confidence interval from about 1.2C to about 0.34C.
- But we are now looking at 600*100 = 60,000 examples
  - What has sampling bought us?
  - "Absolutely Nothing!"
    - Entire population contained ~422,000 samples

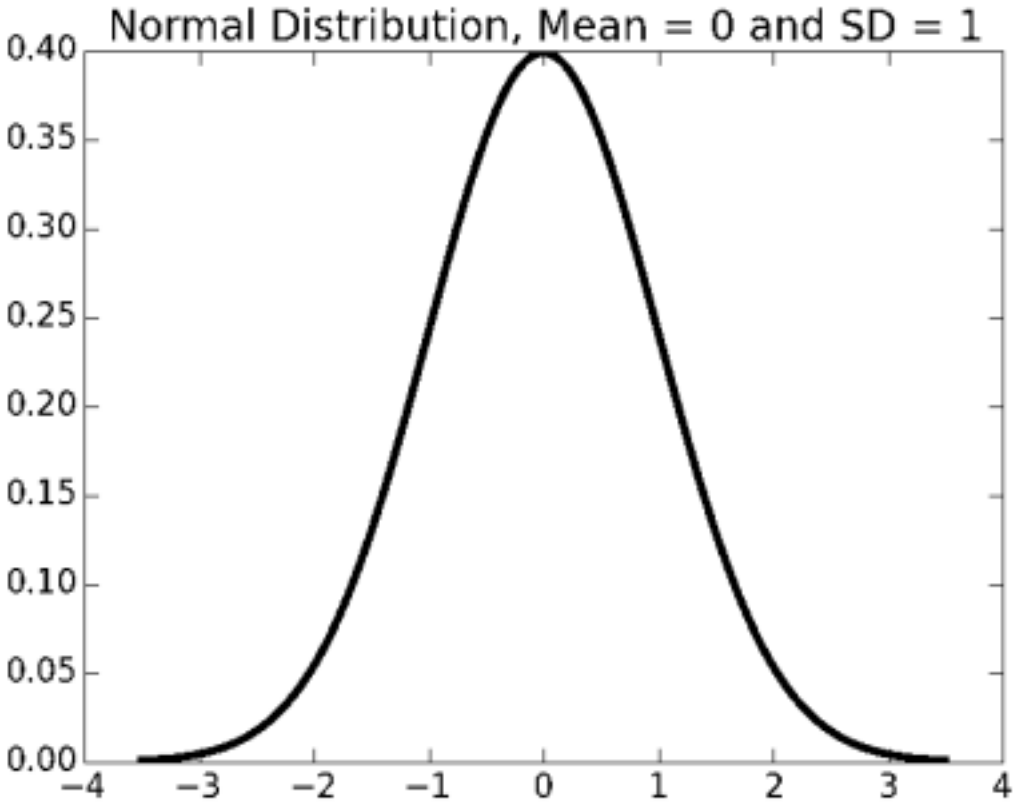


What Can We Conclude from 1 Sample?

- More than you might think
- Thanks to the **Central Limit Theorem**!

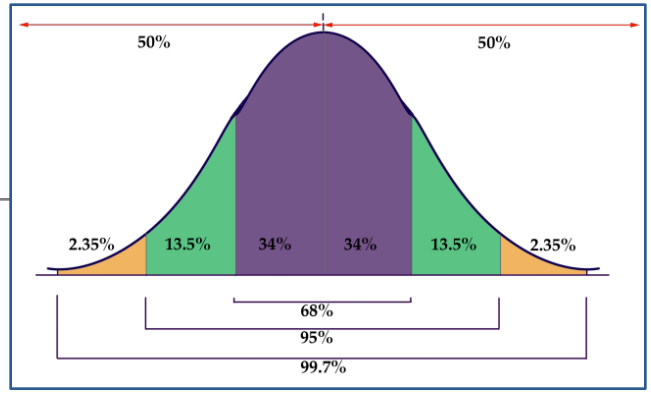
Recall Assumptions for Empirical Rule

- The mean estimation error is zero
- The distribution of the errors in the estimates is normal (Gaussian) – also called a bell curve



Carl Friedrich Gauss: 1777-1855

Empirical Rule (recap)



- If we assume that
  - Mean estimation error is zero
  - Distribution of the errors in the estimates is normal (Gaussian)
- Then by computing mean (μ) and standard deviation (σ), can set confidence intervals:
  - ~68% of data within one standard deviation of mean
  - ~95% of data within 1.96 standard deviations of mean
  - ~99.7% of data within 3 standard deviations of mean
- Common to use 95% confidence interval – state that value is in range:

$$\mu \pm 1.96\sigma \text{ with 95\% confidence}$$

What does this mean?

- What does it mean to say error estimates are characterized by zero mean and normal (or Gaussian) distribution?

- Let's recall how we defined distributions, and then recap a normal or Gaussian distribution

PDF's (recapping)

- When there are only a finite number of possible outcomes, can define probabilities by computing fraction of outcomes with a specific value
- Can't do this when there are an infinite number of outcomes, e.g., when a variable can take on a continuous range of values, rather than a finite range of values
- Thus we need a new way to define probabilities
- Probability distributions defined by *probability density functions* (PDFs)

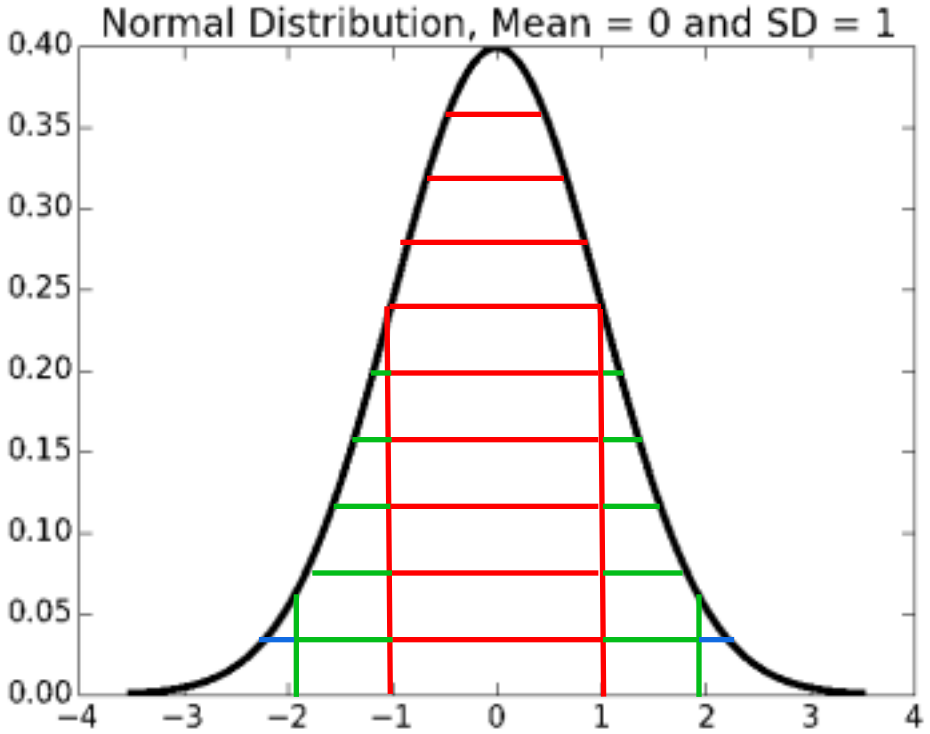
PDF's (recapping)

- PDF at a point describes relative likelihood of that sample point; more typically used to describe probability that a random variable's value lies between two points
- Defines a curve where the values on the x-axis lie between minimum and maximum value of the variable
  - Area under entire curve integrates to 1
  - Area under curve between two points is probability of sample value falling within that range
- For very small range, PDF can be thought of as defining probability at a point

Normal Distributions

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$



~68% of data within one standard deviation of mean

~95% of data within 1.96 standard deviations of mean

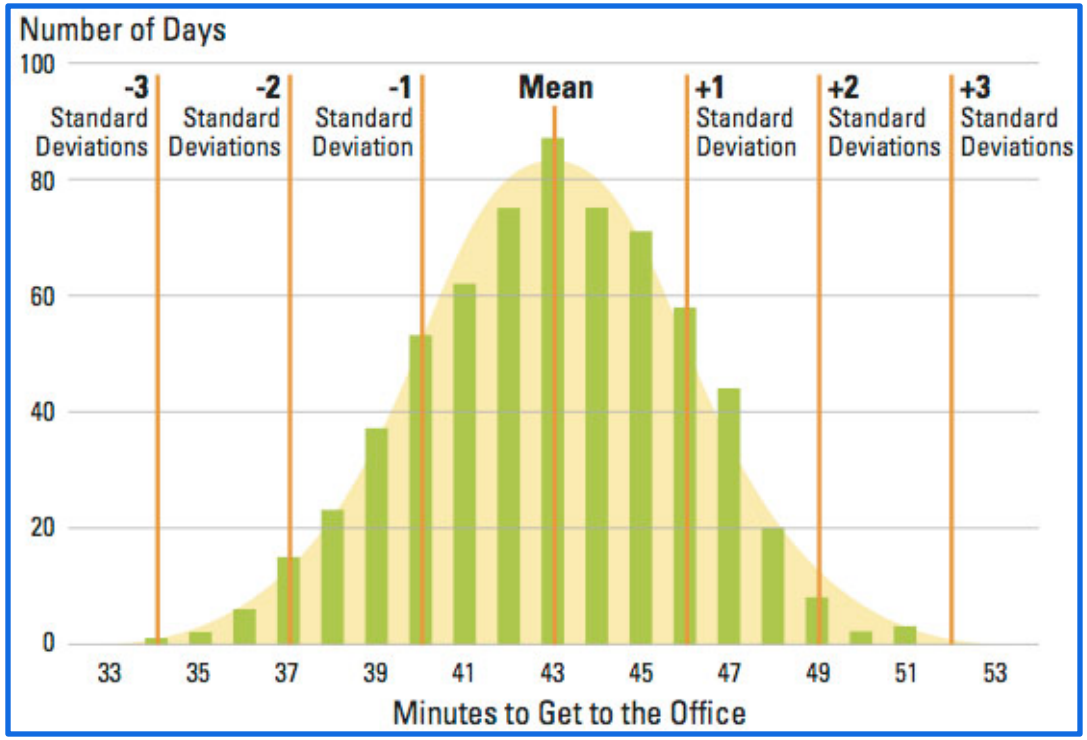
~99.7% of data within 3 standard deviations of mean

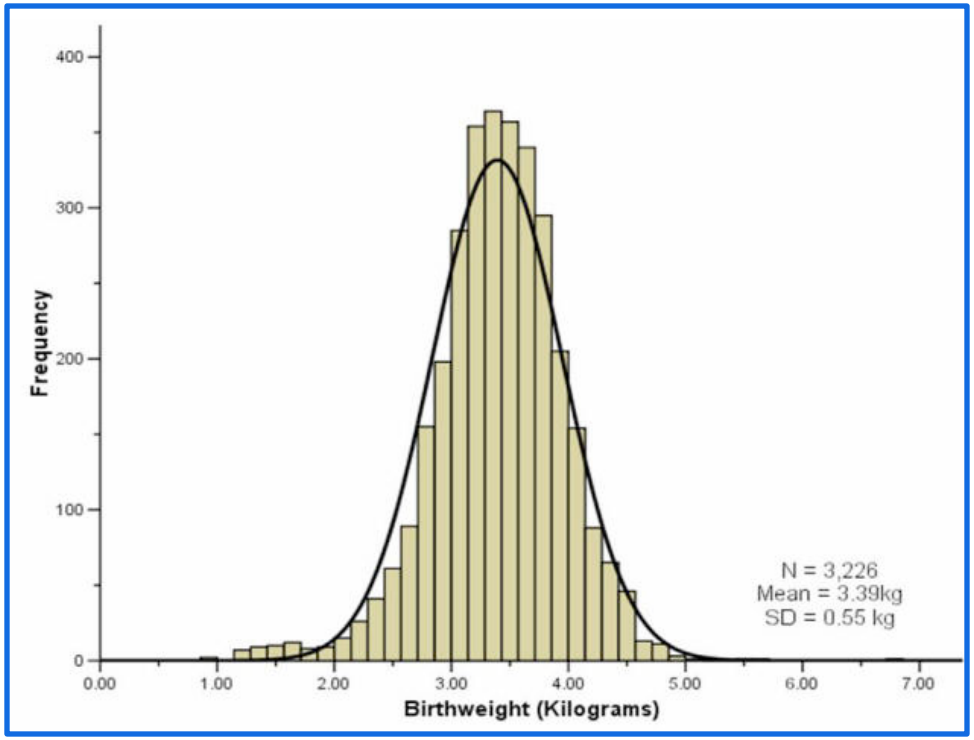
Where are we?

- So if we have a set of samples of a parameter where:
  - The distribution of errors from the estimate has zero mean and is normally (or Gaussian) distributed,
- Then:
  - The Empirical Rule applies and we can state a range of values within which we are confident the actual value lies, with a 95% certainty (or some other certainty).
- How do we know distribution is normal?
  - Could measure it empirically and see how well a Gaussian fits to it? – Expensive!
  - May know from first principles that distribution is normal

Good News: Lots of Normal Distributions

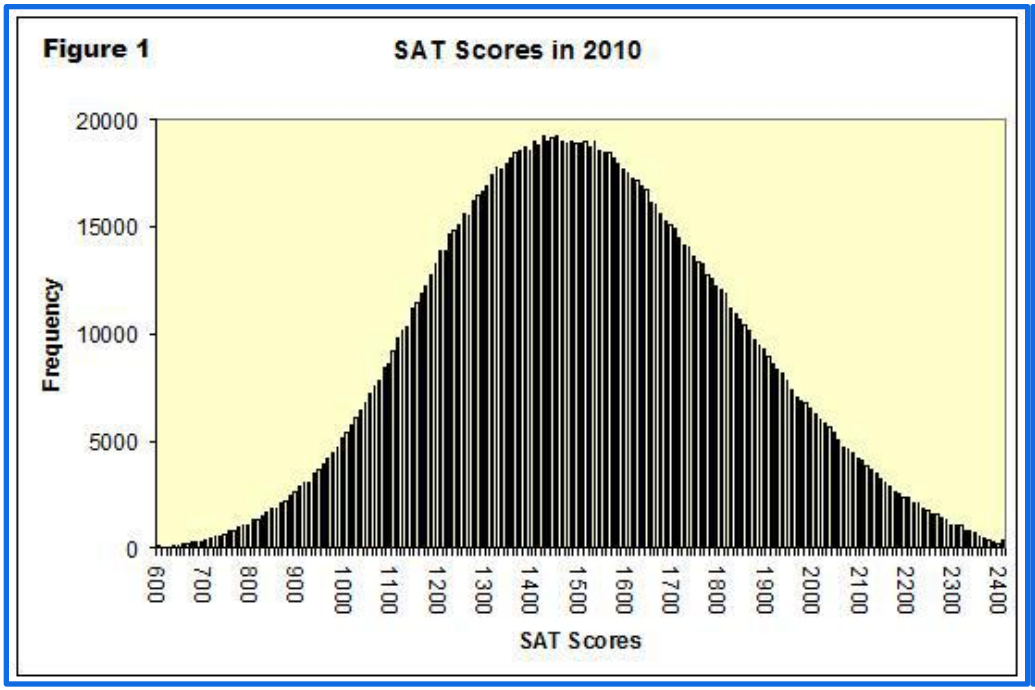
- Occur a lot in real world!
- Nice mathematical properties

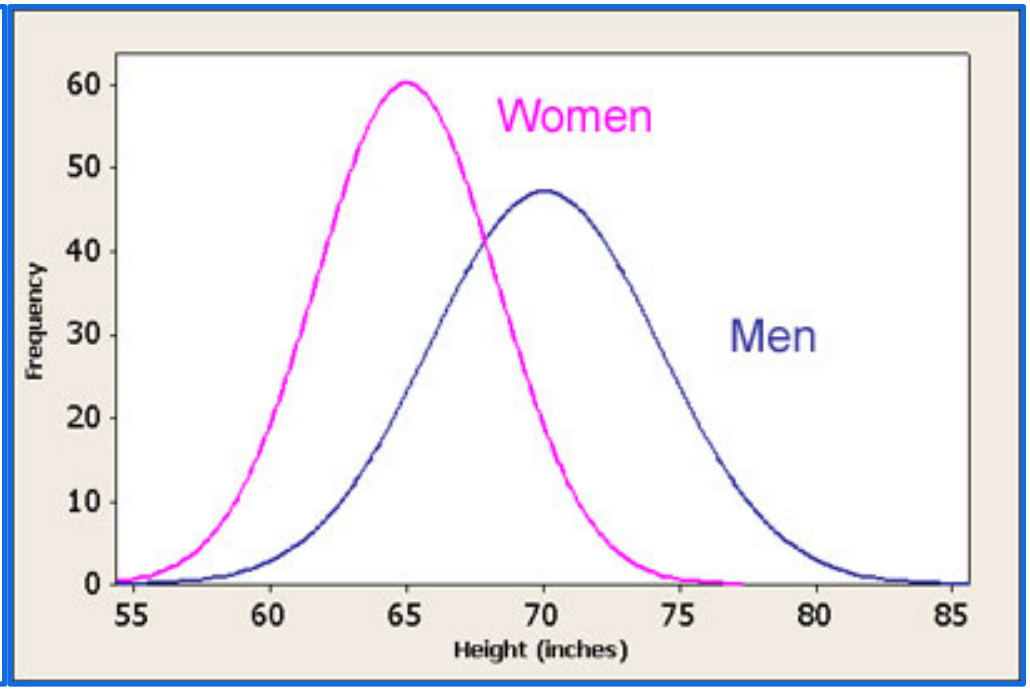




Good News: Lots of Normal Distributions

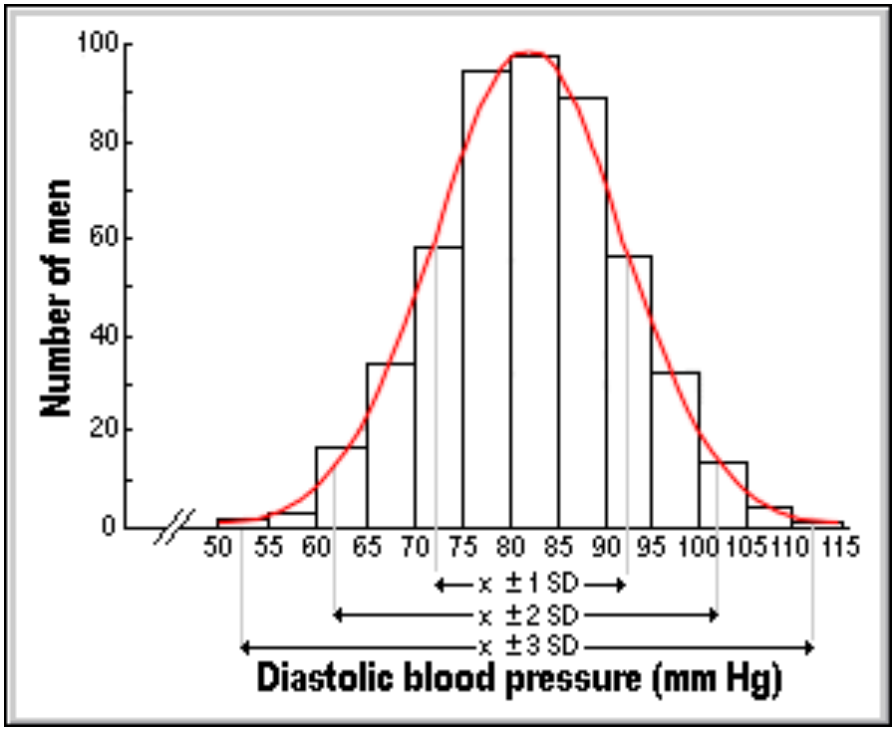
- Occur a lot in real world!
- Nice mathematical properties

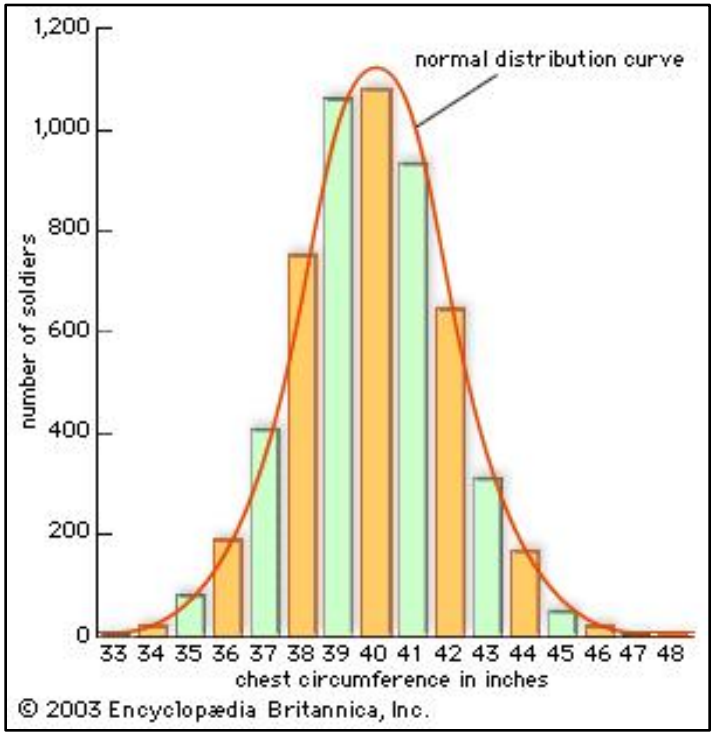




Good News: Lots of Normal Distributions

- Occur a lot in real world!
- Nice mathematical properties





But:
Not All Distributions Are Normal

- Empirical rule works for normal distributions
- But are the outcomes of spins of a roulette wheel normally distributed?
- No, they are uniformly distributed
  - Each outcome is equally probable
- So, why did the empirical rule work when we analyzed betting on roulette?

Why Did the Empirical Rule Work?

- Because we are reasoning not about a single spin, but about the **mean** of a set of spins
- And the Central Limit Theorem applies to that mean of the set

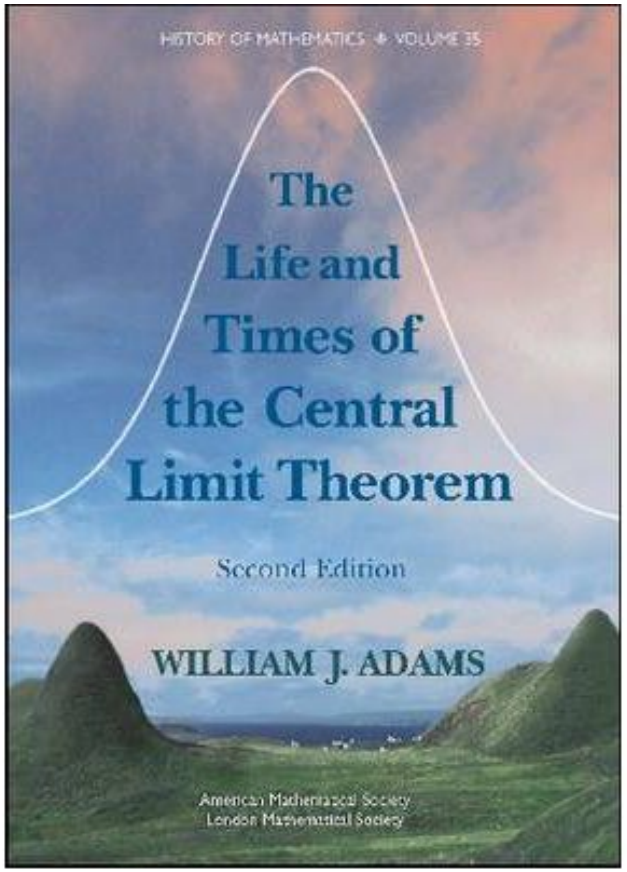
The Central Limit Theorem (CLT)

- Given a sufficiently large sample:

  1) The means of the samples in a set of samples (the sample means) will be approximately normally distributed,

  2) This normal distribution will have a mean close to the mean of the population, and

  3) The variance of the sample means will be close to the variance of the population divided by the sample size.



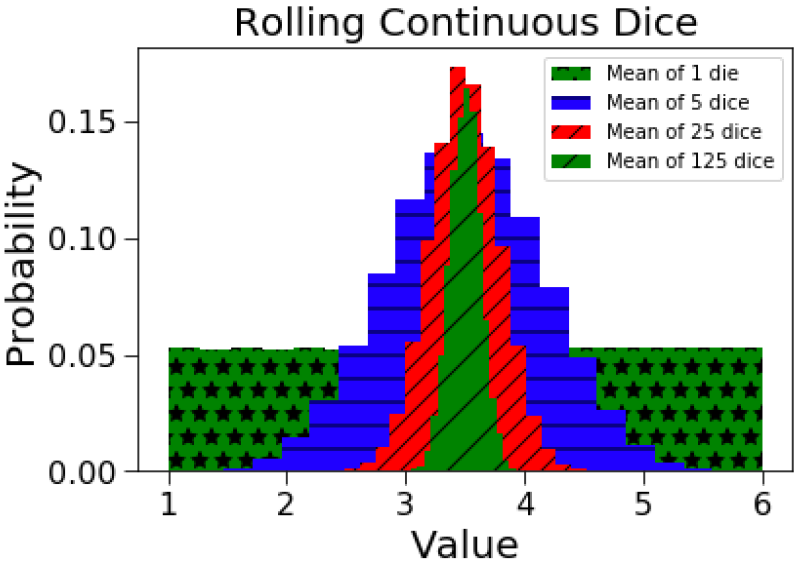
Checking CLT for a Continuous Die

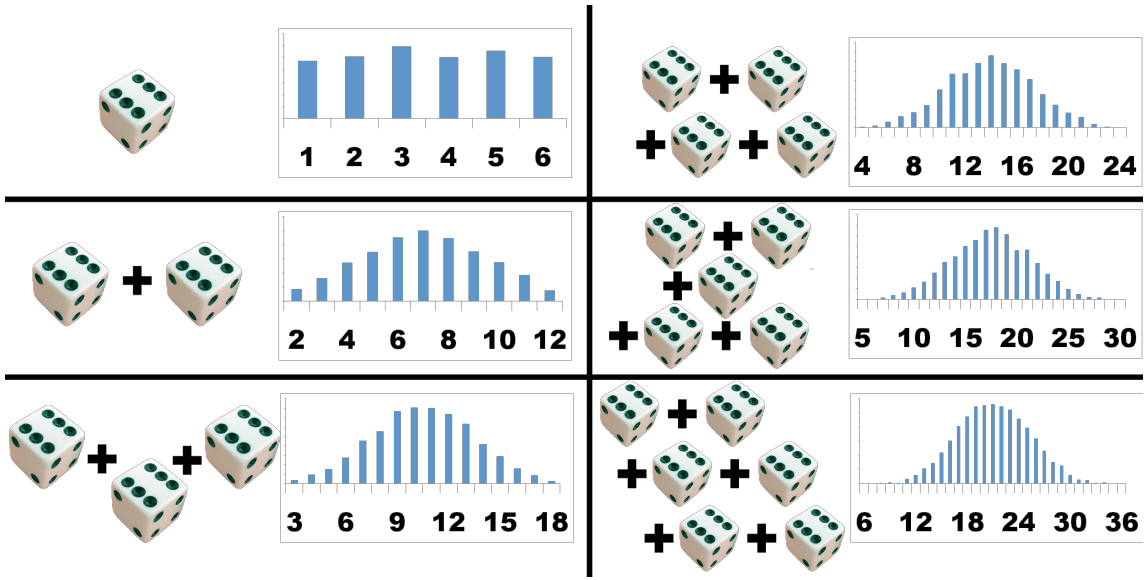
```
def plotMeans(numDice, numRolls, numBins, legend, color, style):
    means = []
    for i in range(numRolls//numDice):
        vals = 0
        for j in range(numDice):
            vals += 5*random.random() + 1
        means.append(vals/float(numDice))
    pylab.hist(means, numBins, color = color, label = legend,
               weights = [1/len(means)]*len(means),
               hatch = style)
    return getMeanAndStd(means)

mean, std = plotMeans(1, 1000000, 19, '1 die', 'b', '*')
print('Mean of rolling 1 die =', str(mean) + ',', 'Std =', std)
mean, std = plotMeans(50, 1000000, 19, 'Mean of 50 dice', 'r', '//')
print('Mean of rolling 50 dice =', str(mean) + ',', 'Std =', std)
pylab.title('Rolling Continuous Dice')
pylab.xlabel('Value')
pylab.ylabel('Probability')
pylab.legend()
```

Output

Mean of rolling 1 die = 3.49759575528, Std = 1.4439045633
Mean of rolling 50 dice = 3.49985051798, Std = 0.204887274645





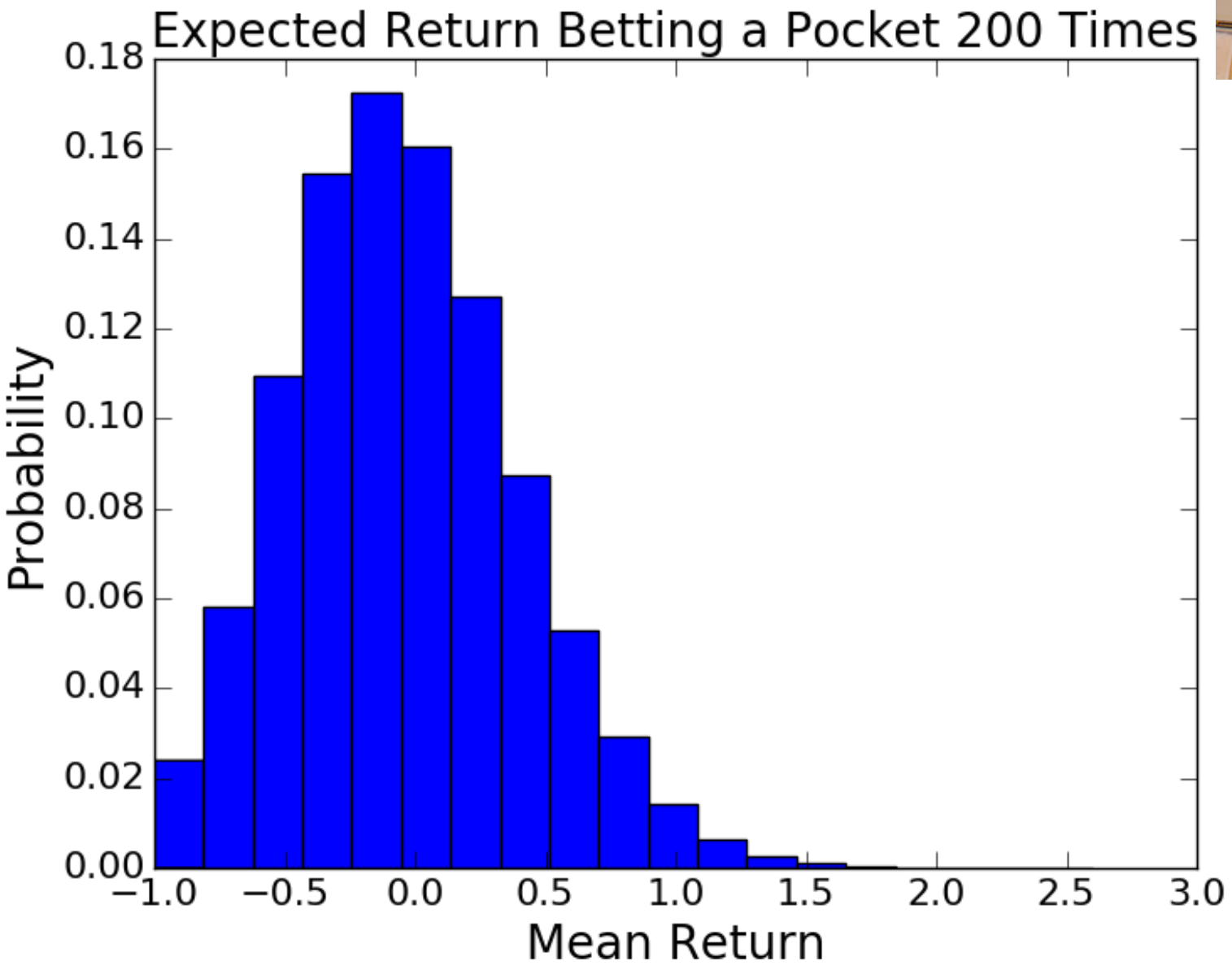
Try It for Roulette

```
numTrials = 1000000
numSpins = 200
game = FairRoulette()

means = []
for i in range(numTrials):
    means.append(findPocketReturn(game, 1, numSpins,
                                  False)[0])

pylab.hist(means, bins = 19,
           weights = [1/len(means)]*len(means))
pylab.xlabel('Mean Return')
pylab.ylabel('Probability')
pylab.title('Expected Return Betting a Pocket 200 Times')
```

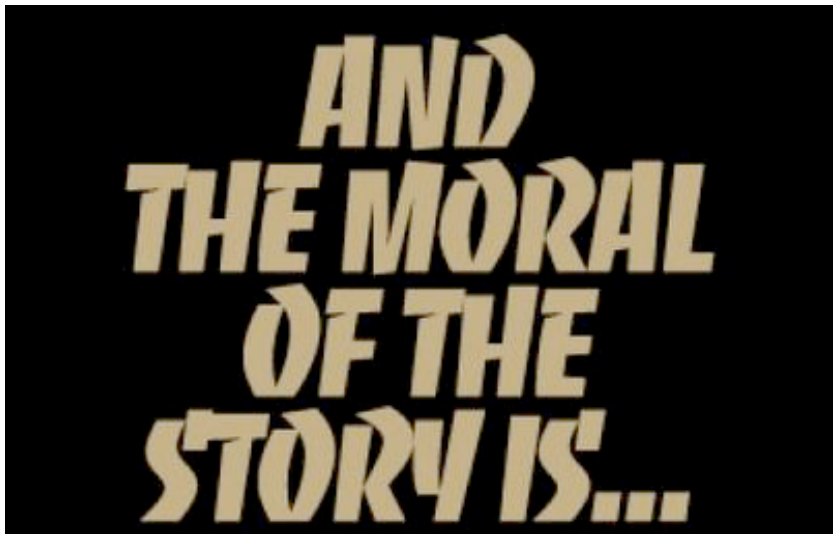
Betting a Pocket in Fair Roulette



Sure looks like a normal distribution

Moral

- It doesn't matter what the shape of the distribution of values happens to be, we can use the Central Limit Theorem to estimate the mean of a population using sufficiently large samples
- The Central Limit Theorem also allows us to use the empirical rule when computing confidence intervals associated with an estimated mean



5 Minute Break

Using the Central Limit Theorem

- Given a sufficiently large sample:
  - 1) The means of the samples in a set of samples (the sample means) will be approximately normally distributed,
  - 2) This normal distribution will have a mean close to the mean the population, and
  - 3) The variance of the sample means will be close to the variance of the population divided by the sample size.

- Time to use the 3rd feature
- Compute *standard error of the mean* (SEM or SE)

Standard Error of the Mean

$$SE = \frac{\sigma}{\sqrt{n}}$$

```
def sem(popSD, sampleSize):
    return popSD/sampleSize**0.5
```

- Does it work?

Testing the SEM

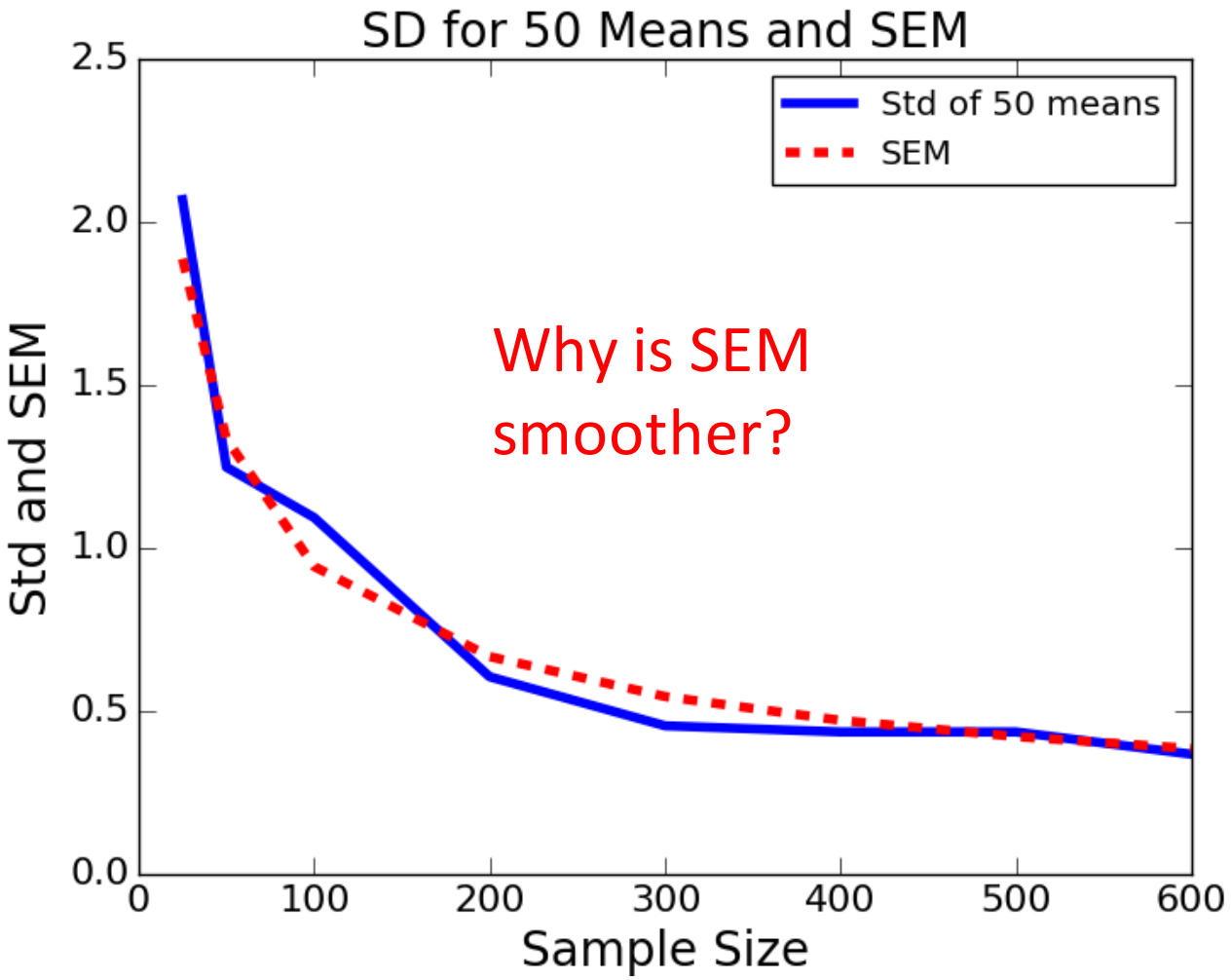
```
sampleSizes = (25, 50, 100, 200, 300, 400, 500, 600)
numTrials = 50
population = getHighs()
popSD = numpy.std(population)
sems = []
sampleSDs = []
for size in sampleSizes:
    sems.append(sem(popSD, size))
    means = []
    for t in range(numTrials):
        sample = random.sample(population, size)
        means.append(sum(sample)/len(sample))
    sampleSDs.append(numpy.std(means))
pylab.plot(sampleSizes, sampleSDs,
           label = 'Std of ' + str(numTrials) + ' means')
pylab.plot(sampleSizes, sems, 'r--', label = 'SEM')
```

Standard Error of the Mean

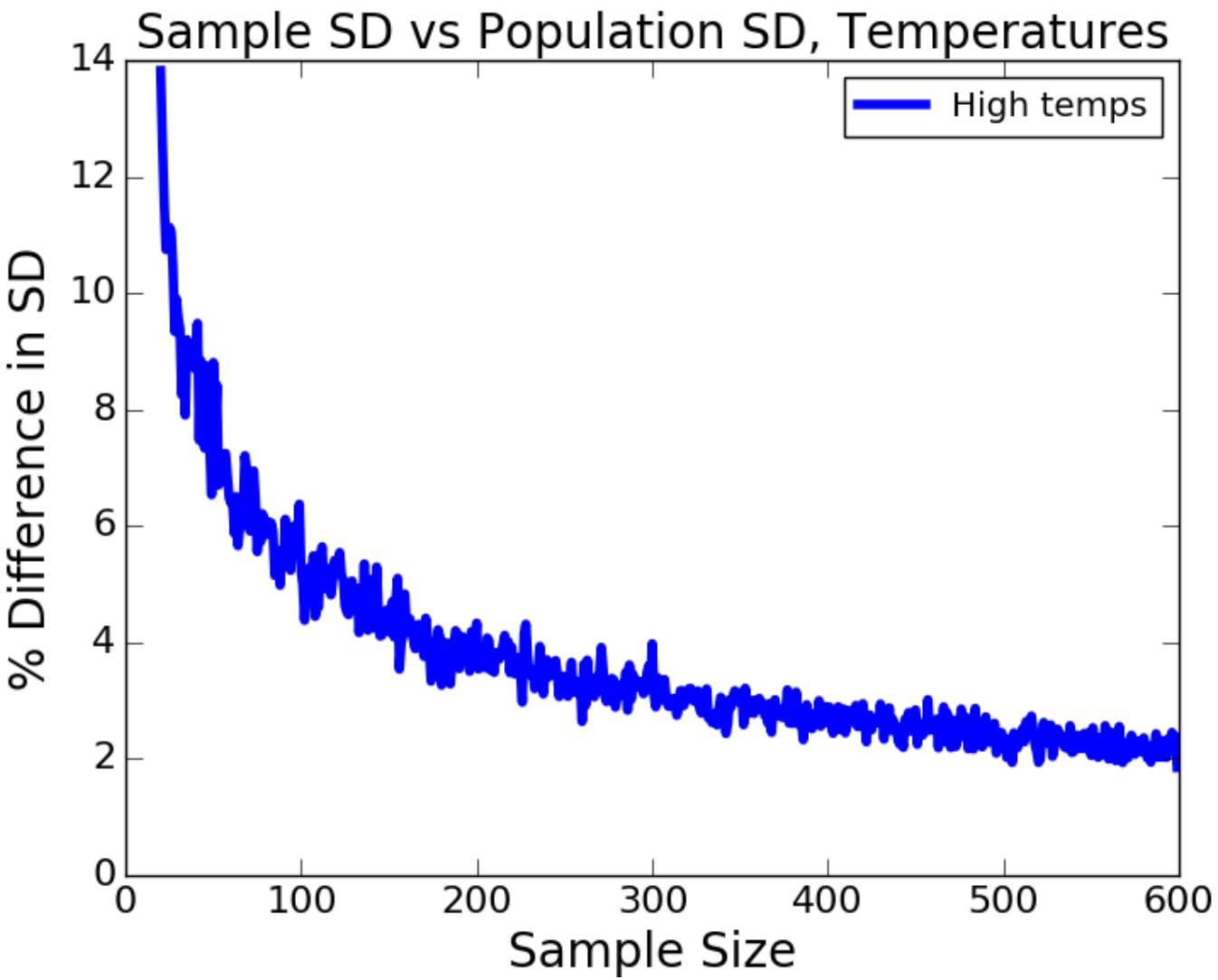
$$SE = \frac{\sigma}{\sqrt{n}}$$

But, we don't know standard deviation of population

How might we approximate it?

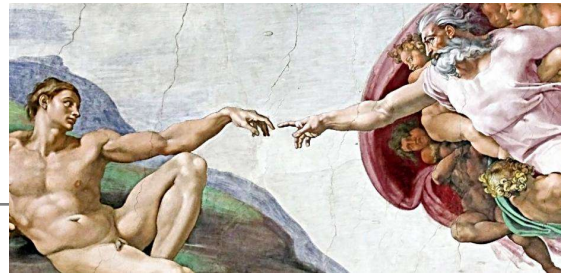


Sample SD vs. Population SD

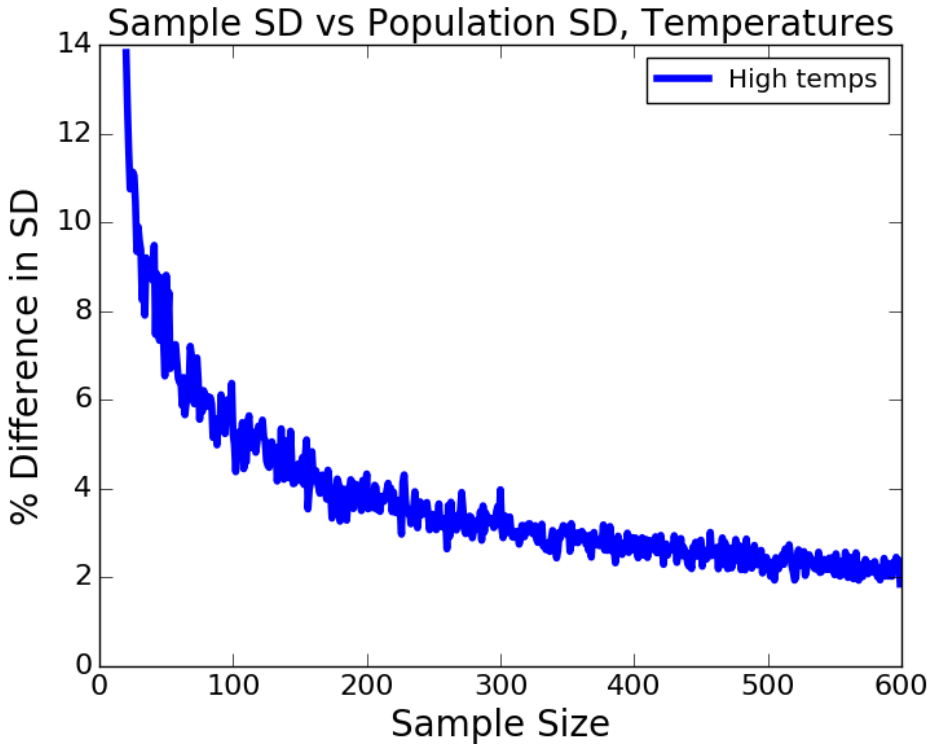


Looks like with large enough sample size, we might be able to just use the sample SD to compute SEM

The Point



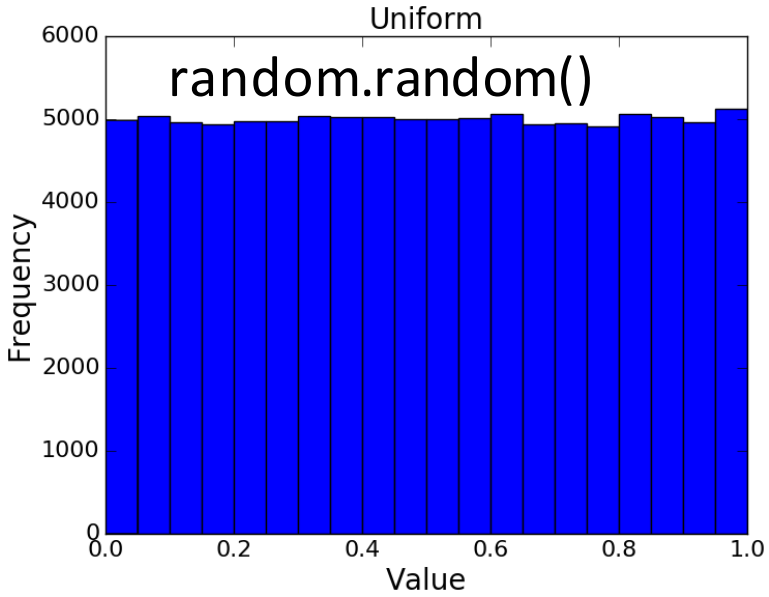
- Once sample reaches a reasonable size, sample standard deviation is a pretty good approximation to population standard deviation
- True only for this example?
  - Distribution of population?
  - Size of population?

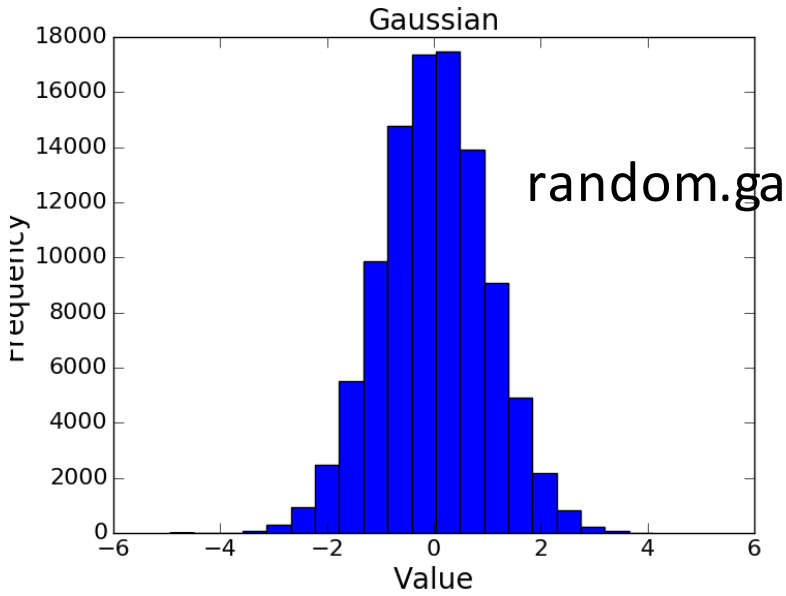


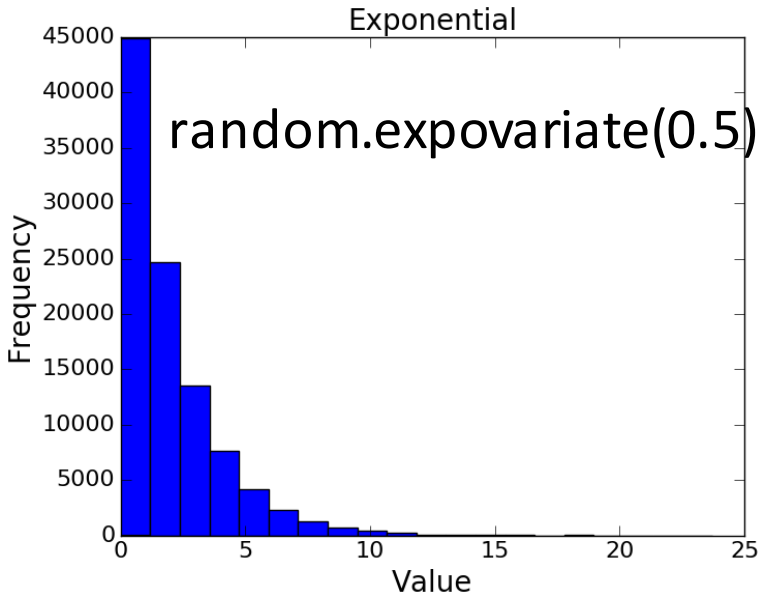
Looking at Distributions

```
def plotDistributions():
    uniform, normal, exp = [], [], []
    for i in range(100000):
        uniform.append(random.random())
        normal.append(random.gauss(0, 1))
        exp.append(random.expovariate(0.5))
    makeHist(uniform,'Uniform','Value','Frequency')
    pylab.figure()
    makeHist(normal,'Gaussian','Value','Frequency')
    pylab.figure()
    makeHist(exp,'Exponential','Value','Frequency')
```

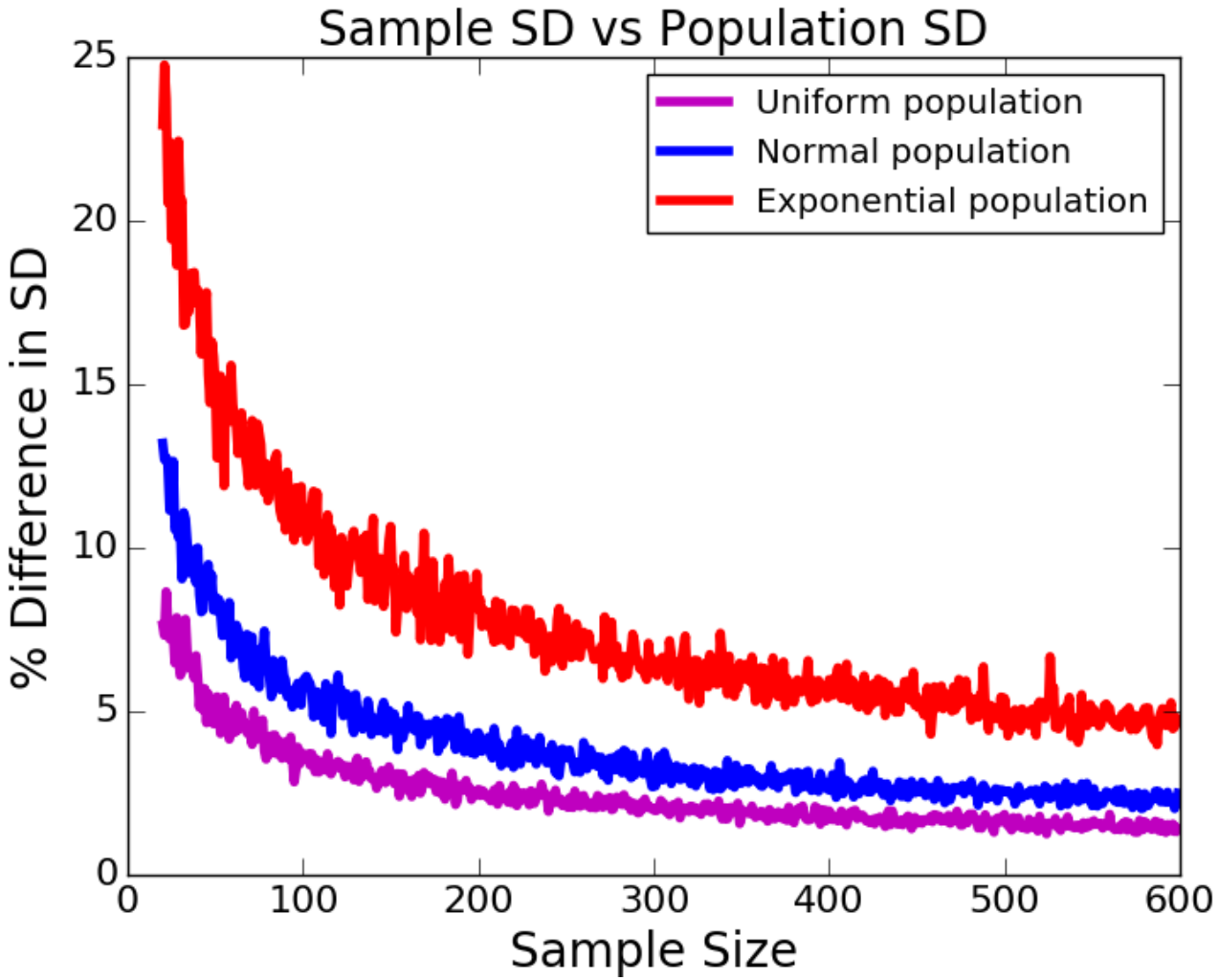
Three Different Distributions







Does Distribution Matter?



All show same convergence, but not equally good at using sample SD for population SD

Skew, a measure of the asymmetry of a probability distribution, matters

The more skewed a distribution, the more samples you need for SD to become similar

A Quick Aside on Skew

- Intuitively, just think of this as measuring the asymmetry of a probability distribution function
- More formally, typical measure is Pearson's moment coefficient of skewness:

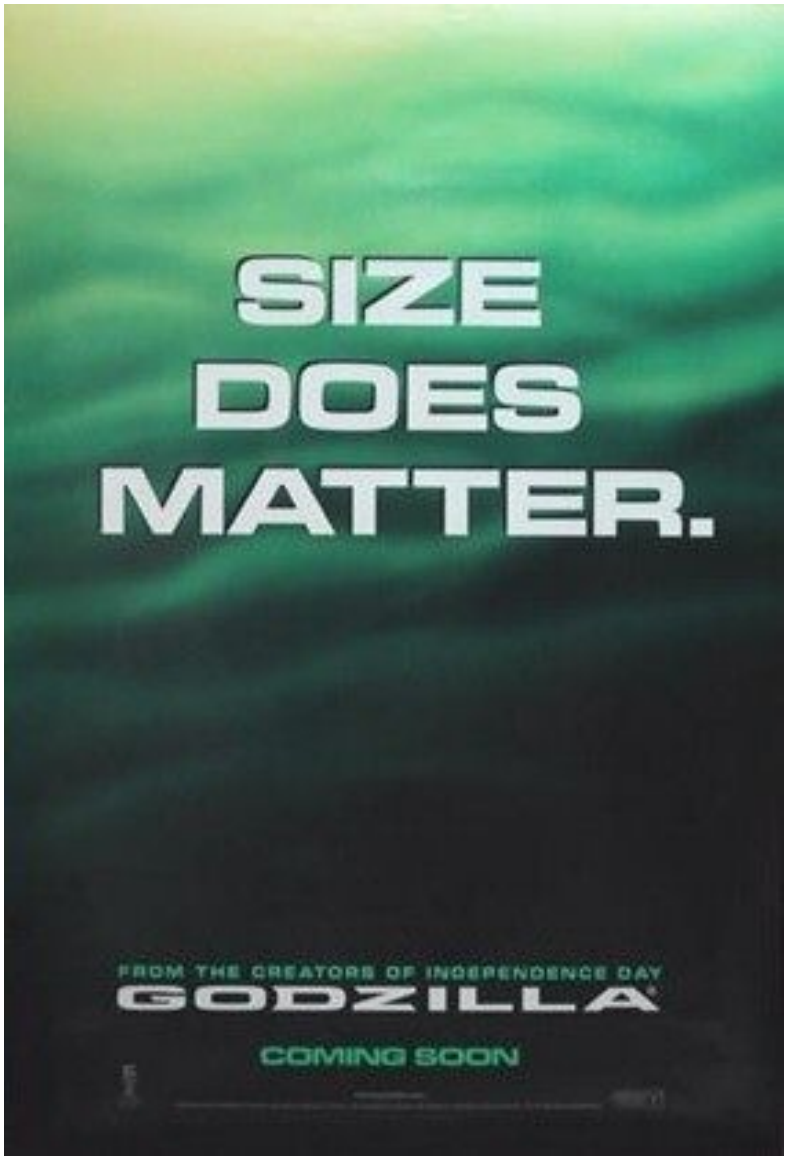
$$E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right]$$

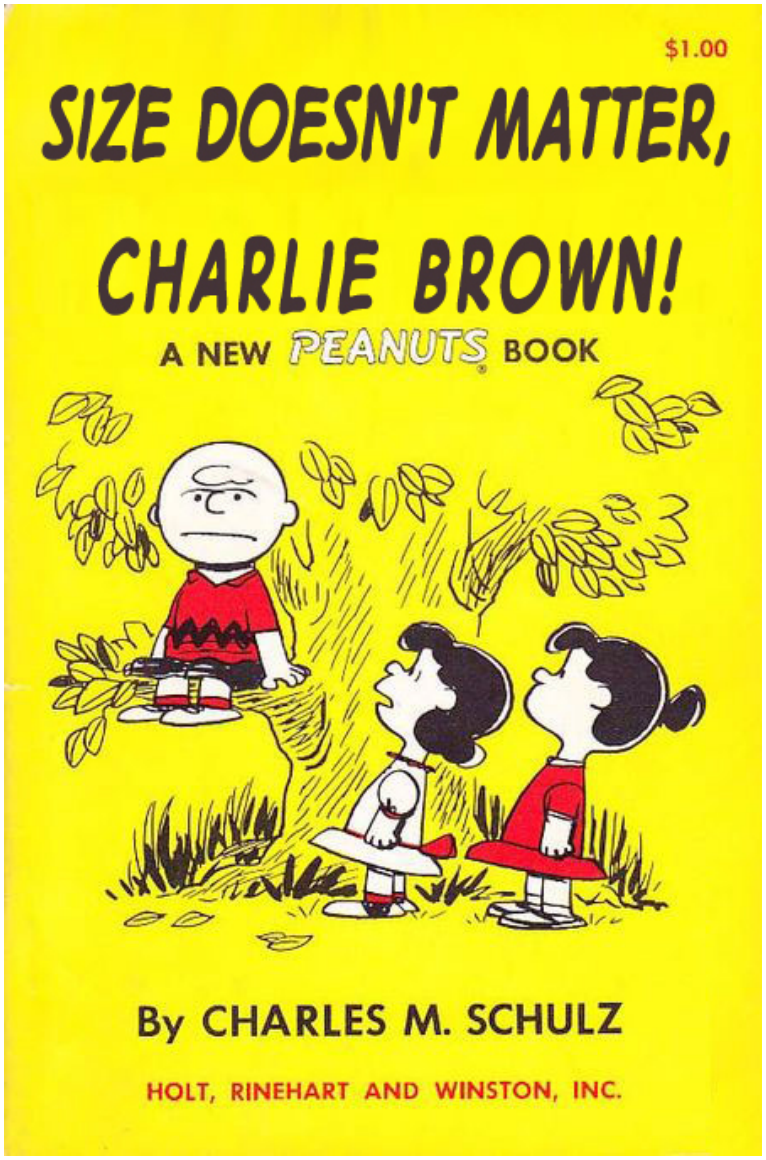
where X is a random variable; μ is the mean value of the variable; σ is the standard deviation of the variable; and $E[.]$ is the expected value of a random variable expression or the mean of the expression:

$$E[X] = \sum_{x \in X} x\, p(x)$$

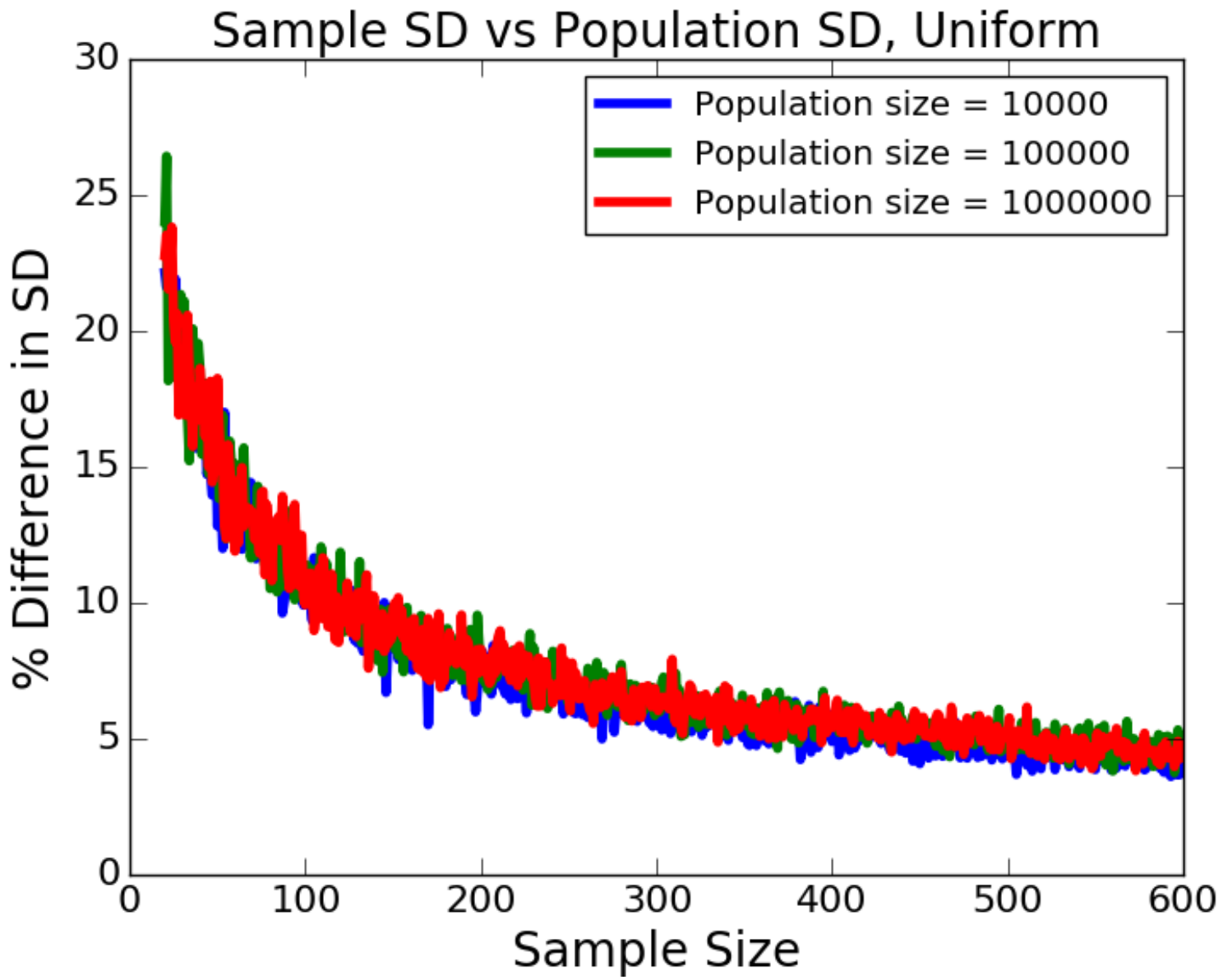
p(x) is the probability of x occurring

Does Size Matter?





Does Population Size Matter?



To Estimate Mean from a Single Sample

- 1) Choose sample size based on estimate of skew in population
- 2) Chose a random sample from the population
- 3) Compute the mean and standard deviation of that sample
- 4) Use the standard deviation of that sample to estimate the SE
- 5) Use the estimated SE to generate confidence intervals around the sample mean

Works great when we choose independent random samples.

Not always so easy to do, as political polls demonstrate.

Are 200 Samples Enough?

```
random.seed(0)
temps = getHighs()
sampleSize = 200
numTrials = 10000
popMean = sum(temps)/len(temps)
numBad = 0
for t in range(numTrials):
    sample = random.sample(temps, sampleSize)
    sampleMean = sum(sample)/sampleSize
    SEM = numpy.std(sample)/sampleSize**0.5
    if abs(popMean - sampleMean) > 1.96*SEM:
        numBad += 1
print('Fraction outside 95% confidence interval =',
      numBad/numTrials)
```

Fraction outside 95% confidence interval = 0.0511

Recapping Last Three Lectures

- Using Monte Carlo simulation to build models
  - The world is mostly stochastic
    - Useful tool even when randomness not present
  - Estimating reliability of simulation results
    - Don't confuse statistical assertions with factual assertions
- Understanding populations
  - Cannot examine all members
  - Rely on sampling
  - Estimating validity of conclusions based on samples
  - Central Limit Theorem lets us use a single sample, and still assert inferences with specific confidence levels
- Some math, but goal was to use computation to help develop intuition
- **Next unit: building models of data**