

# Distributions, Sampling, Central Limit Theorem, and Standard Error

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

---

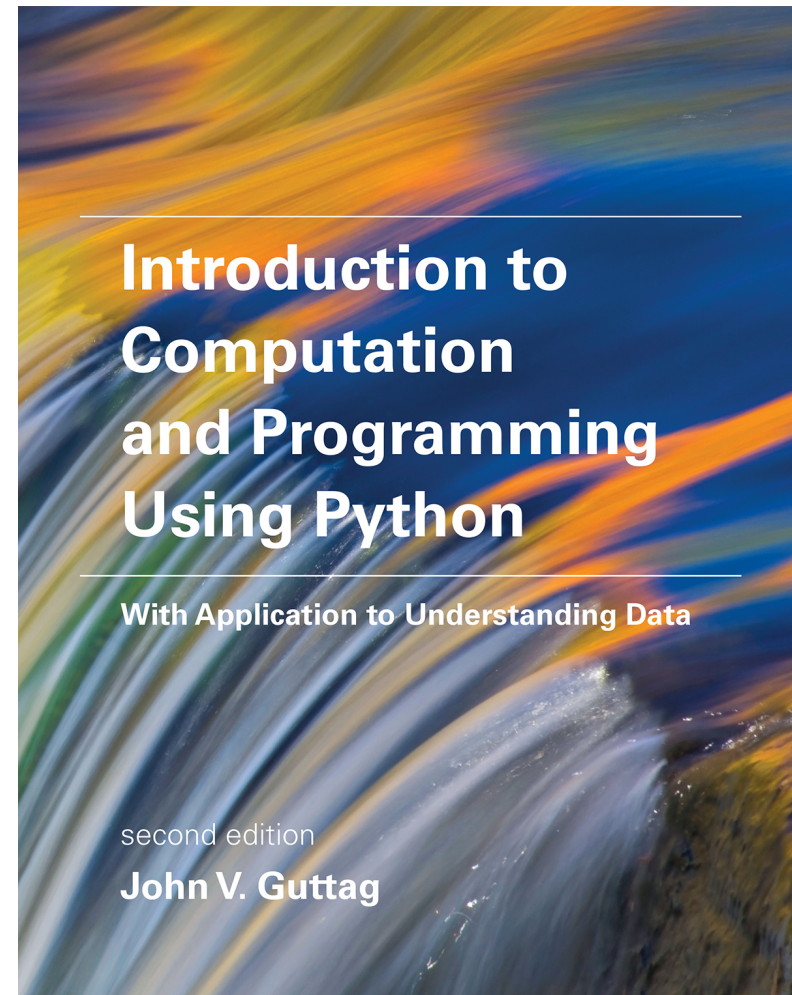
Eric Grimson

MIT Department Of Electrical Engineering and  
Computer Science

# Assigned Reading

---

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



# Recall Inferential Statistics

---



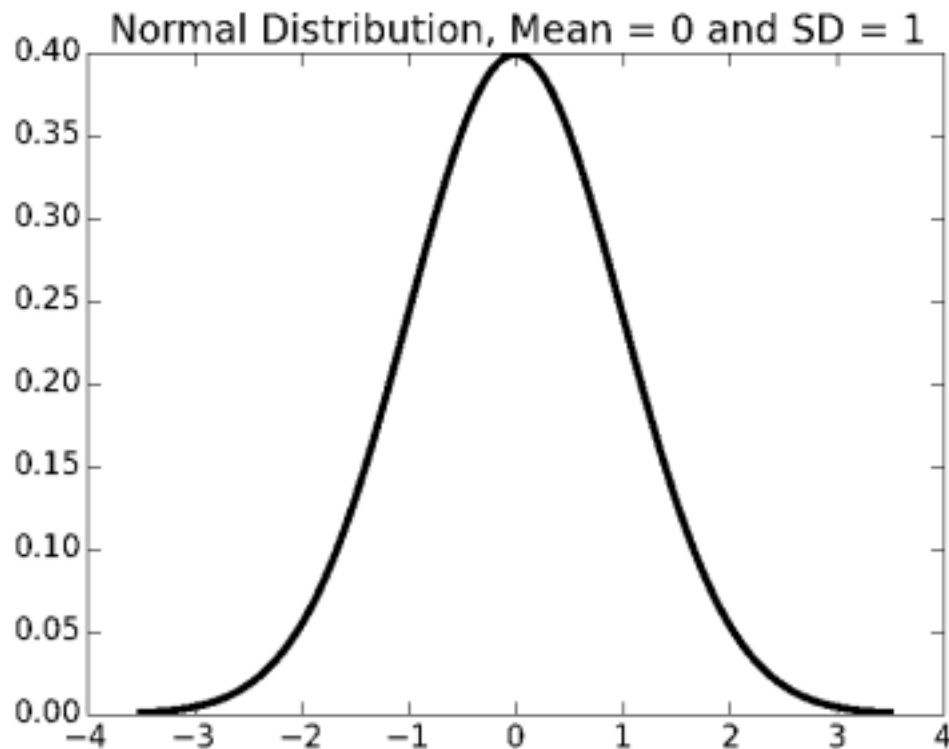
- Make an inference about some property of a population by examining one or more random samples drawn from that population
  - infer property of population based on statistics of sub-population
  - avoid cost of having to look at entire population, if very large
  - handle cases where not feasible to examine whole population
- With Monte Carlo simulation, generate lots of random samples and use them to make inferences and to compute confidence intervals for the inferences using empirical rule
  - allows us to estimate likelihood of inference
  - useful when variation in value due to noise or random effects

# Recall Assumptions for Empirical Rule

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

Note: **not** that estimate of mean is zero, but mean of errors of estimate is zero

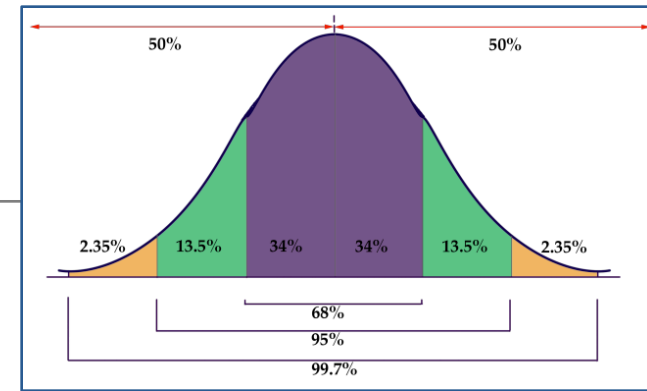
What does this mean? We'll get to that shortly



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 ( $\mu$ ) and standard deviation ( $\sigma$ ) of estimates, 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:

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

Poll

# Generating Normally Distributed Data

---

What is a Gaussian (or normal) distribution?

First, what does it look like?

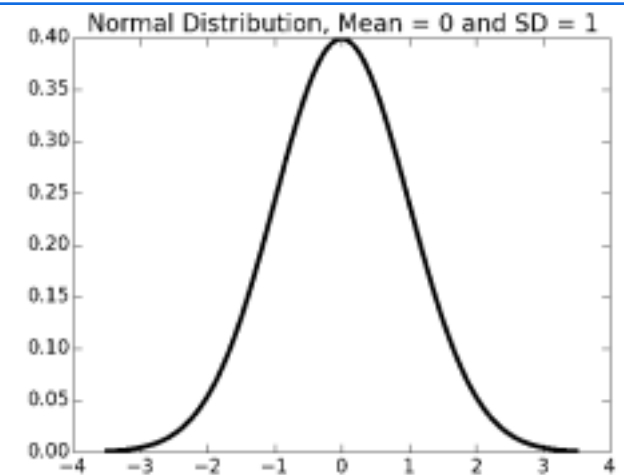
```
dist, numSamples = [], 1000000

for i in range(numSamples):
    dist.append(random.gauss(0, 1))

pylab.hist(dist, bins = 100)
pylab.xlabel('Value')
pylab.ylabel('Frequency')
```

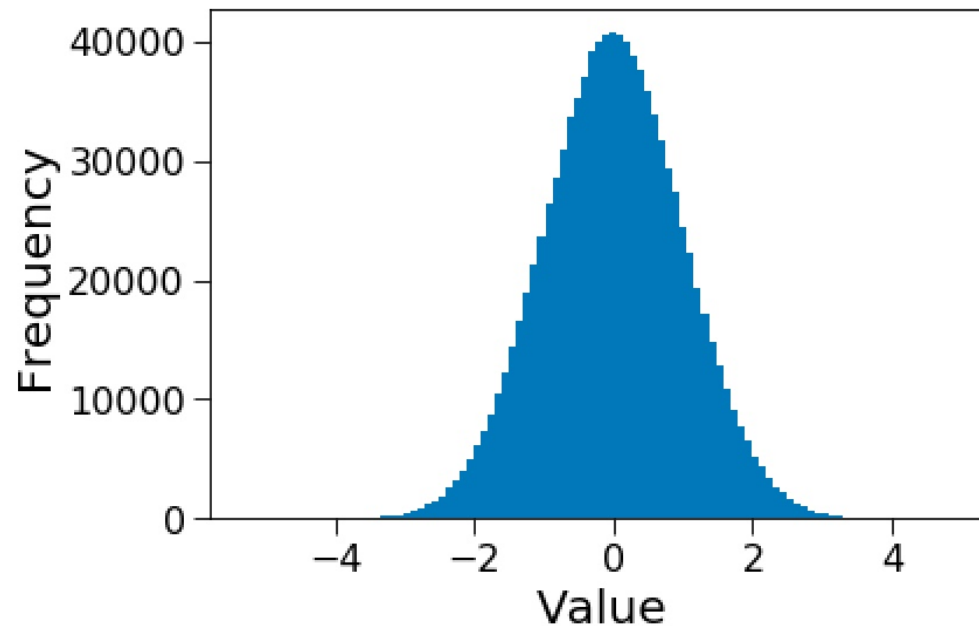
Sampling from `random.gauss` will return a value along x axis, based on relative probability graph below

Taking 1M samples gives an approximation to Gaussian distribution



# Output

---



- This is a discrete approximation of a Gaussian distribution – histogram of 1M samples from a Gaussian function with standard deviation of 1, split into 100 buckets along x axis (each bucket roughly 0.1 units wide)
- Ideally, for given resolution of values along x axis, this describes probability of seeing that value (e.g., divide each histogram value by 1M)
- But want to define probability, independent of resolution of values

# Defining Distributions

---

- Probability distribution captures notion of relative frequency with which a random variable takes on certain values
  - Discrete random variables drawn from finite set of values
  - Continuous random variables drawn from real numbers between two numbers (i.e., infinite set of values)
- For discrete variable, simply list the probability of each value; all probabilities must add up to 1
- Case for continuous variable trickier, can't enumerate probability for each of an infinite set of values
  - Could select some resolution for values on x axis, and use histogram to find discrete probabilities, but this limits accuracy of estimate of variable

# PDFs

---

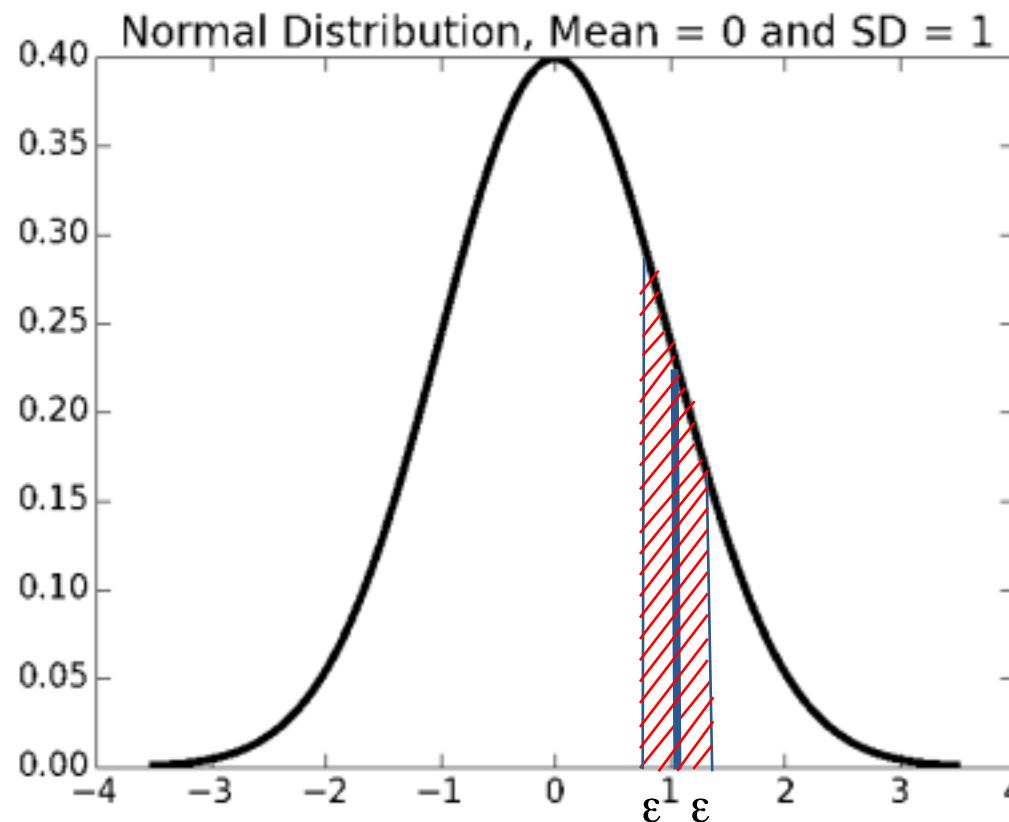


- Distributions defined by *probability density functions* (PDFs) – aka *probability distribution functions*
- PDF at a point describes relative likelihood of that sample; more typically used to describe probability that a random variable's value lies between two points
- Defines a curve where the values on the x-axis lie between minimum and maximum value of the variable
  - Area under curve between two points is probability of variable's value falling between those two points
- For small range, PDF can be thought of as defining probability at a point



# PDF's define probabilities

- Area under curve over small span of  $x$  defines probability of value lying in that range



Limit as  $\epsilon$  tends to zero defines probability

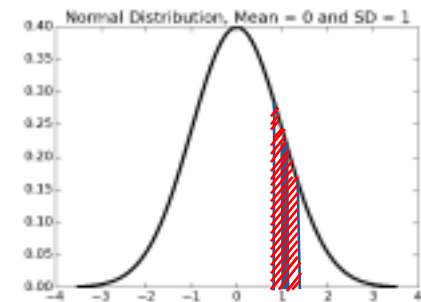
# PDF's More Formally

- Let  $f(x)$  denote the probability density function
  - E.g. for a normal distribution

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

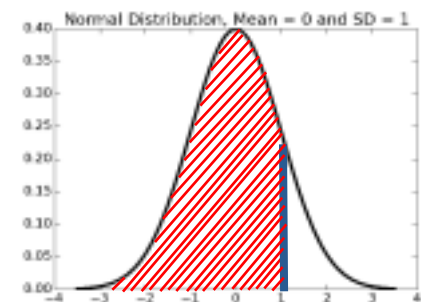
- Probability of value lying in range near  $x$ :

$$P(x) = \int_{x-\varepsilon}^{x+\varepsilon} f(t)dt$$



- Cumulative distribution function (probability that random variable is less than or equal to  $x$ ):

$$F(x) = \int_{-\infty}^x f(t)dt$$



- Note, integral over full range of values is 1, but value of PDF at a point can be greater than 1 (which is why we want integral over a range, or area under the curve)

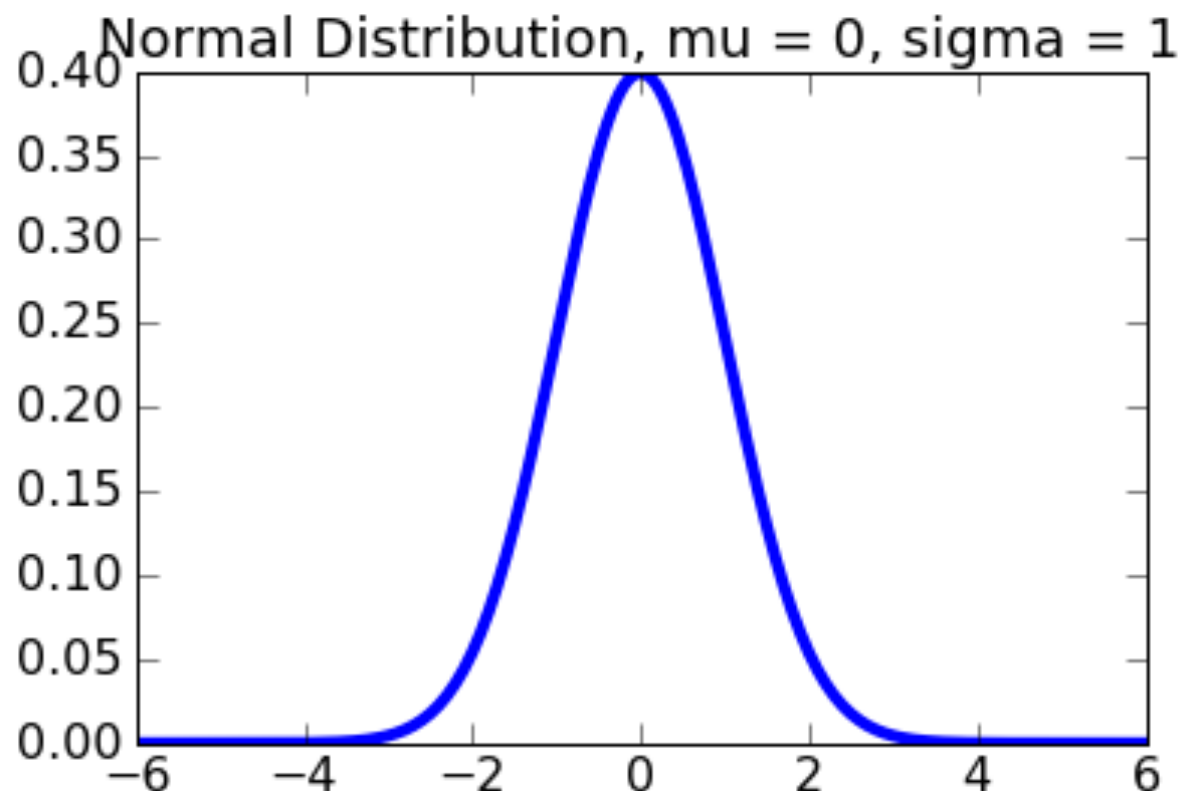
# PDF for Normal Distribution

```
def gaussian(x, mu, sigma):  
    factor1 = (1.0/(sigma*((2*pylab.pi)**0.5)))  
    factor2 = pylab.e**(-(((x-mu)**2)/(2*sigma**2)))  
    return factor1*factor2
```

```
xVals, yVals = [], []  
mu, sigma = 0, 1  
x = -6  
step = 0.05  
while x <= 6:  
    xVals.append(x)  
    yVals.append(gaussian(x, mu, sigma))  
    x += step  
pylab.plot(xVals, yVals)  
pylab.title('Normal Distribution, mu = ' + str(mu) \  
            + ', sigma = ' + str(sigma))
```

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

# Output



Are values on y-axis probabilities?

They are densities; i.e., derivative of cumulative distribution function

Hence we use integration to interpret a PDF

This gives us a sense of the mathematical definition of a distribution

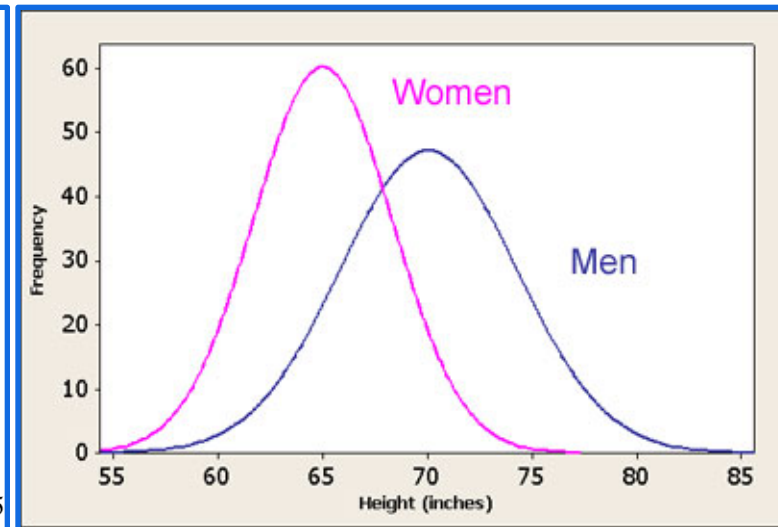
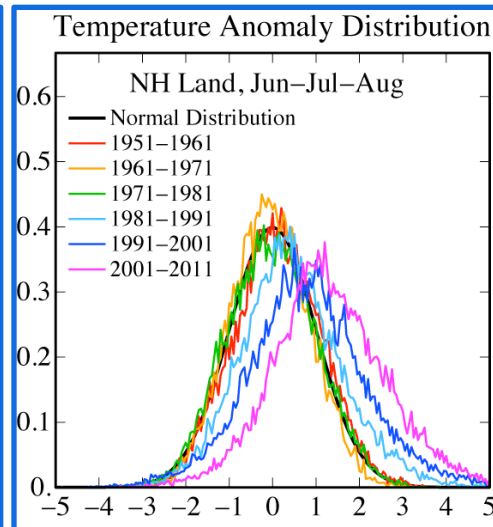
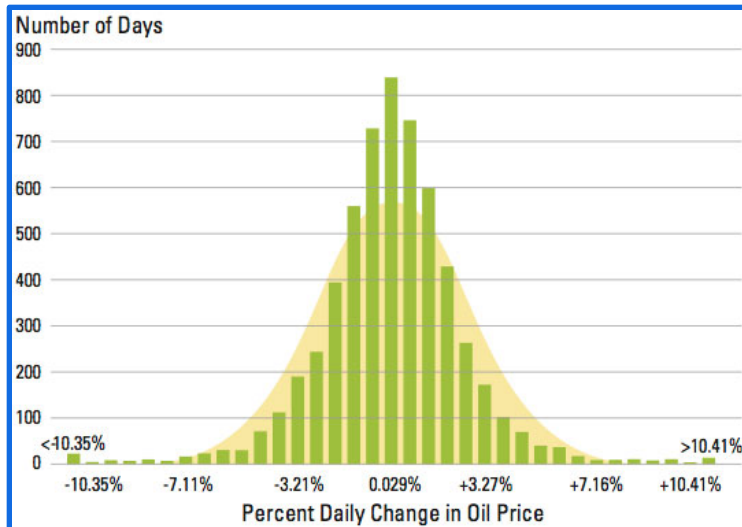
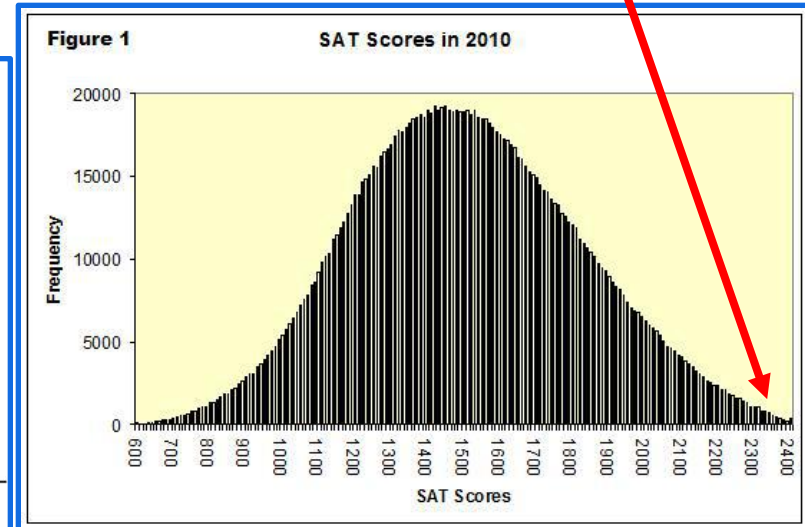
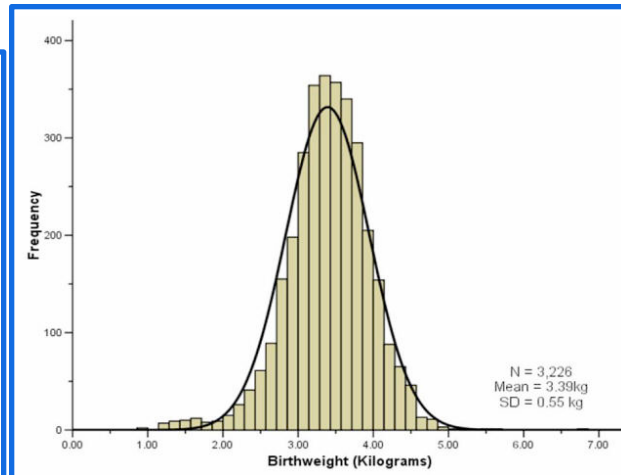
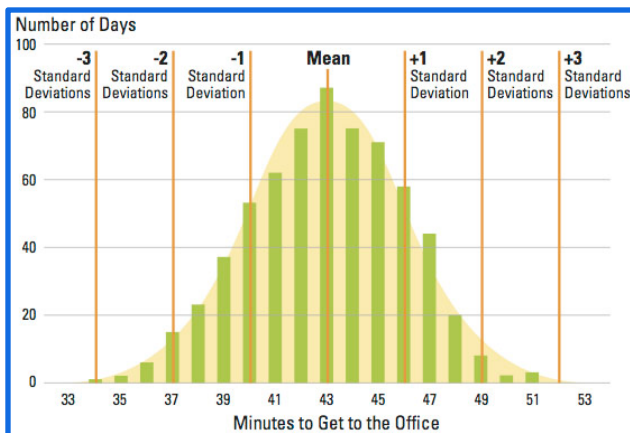
Our goal is to use Empirical Rule, but it only applies if we have a normal distribution of errors in our estimate of the mean

**Are distributions of errors in an estimate typically normal?**

# Everybody Likes Normal Distributions

- They occur a lot!

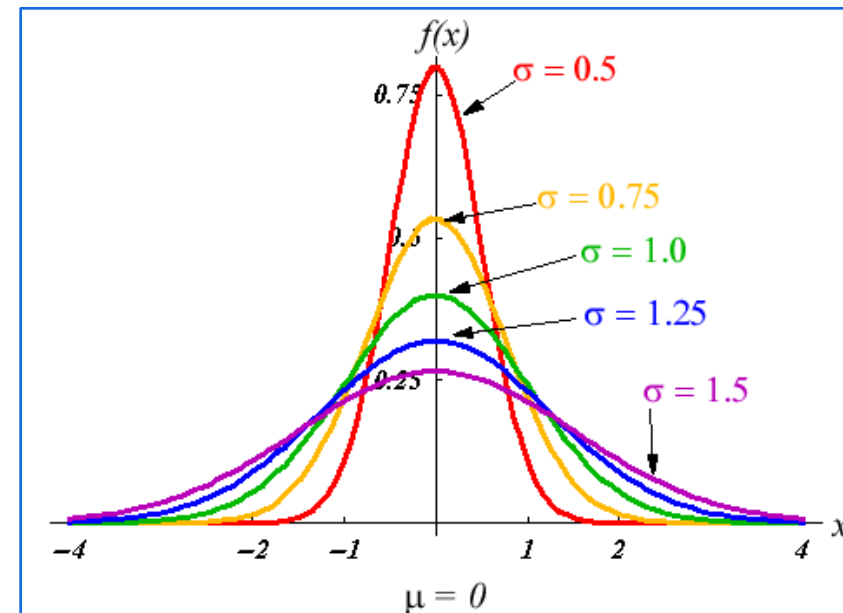
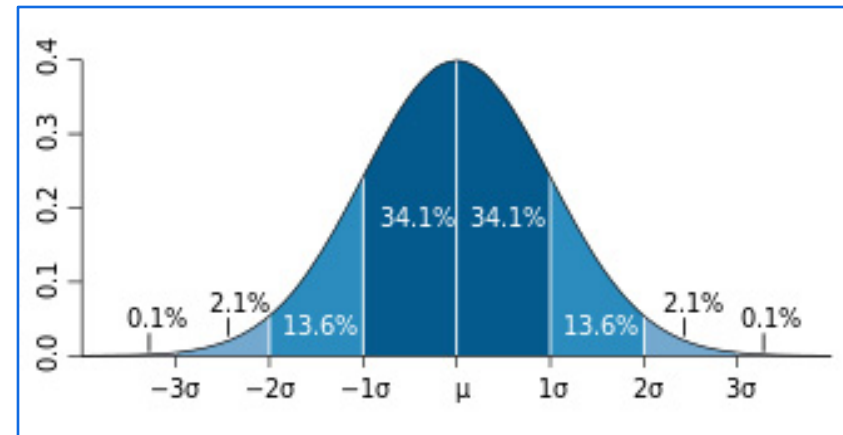
MIT students





# Everybody Likes Normal Distributions

- They occur a lot!
- Nice mathematical properties
  - Symmetric around mean
  - Mean is also mode and median
  - Area under curve is 1
  - Its density is infinitely differentiable
  - It is unimodal – its first derivative is positive to the left of the mean, negative to the right of the mean and zero only at the mean



# But There Are Other Distributions!

---

- Uniform distributions
- Binomial distributions
- Exponential distributions
- Other, more esoteric, distributions
  - Log-normal
  - Pareto
  - Bernoulli
  - Poisson
  - many others

# Uniform Distributions

---

- All intervals of the same length have the same probability
- Probability that a value falls between  $x$  and  $y$  (where total range of possible values is  $a$  to  $b$ ) is:

$$P(x, y) = \begin{cases} \frac{y - x}{b - a} & \text{if } x \geq a \text{ and } y \leq b \\ 0 & \text{otherwise} \end{cases}$$

- `random.uniform(min, max)` will draw an element within range with uniform probability

- Discrete version

$$P(x) = \begin{cases} \frac{1}{|S|} & \text{if } x \in S \\ 0. & \text{otherwise} \end{cases}$$

- `random.choice(S)` will select an element from set with uniform probability

# Uniform Distributions: Examples

- Coin flipping
- Dice rolling
- Roulette
- Waiting times, e.g., arrival of bus
- Spacing of territorial animals
  - Social distancing?



# Binomial Distributions

---

- What is the probability that a test succeeds exactly  $k$  times out of  $n$  independent trials (e.g., flip a coin  $n$  times, probability of exactly  $k$  heads)?
- If  $p$  is probability of success on one trial, then desired probability is:

$$P(k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

where

$$\binom{n}{k} = \frac{n!}{k! (n - k)!} \quad \text{aka "n choose k"}$$

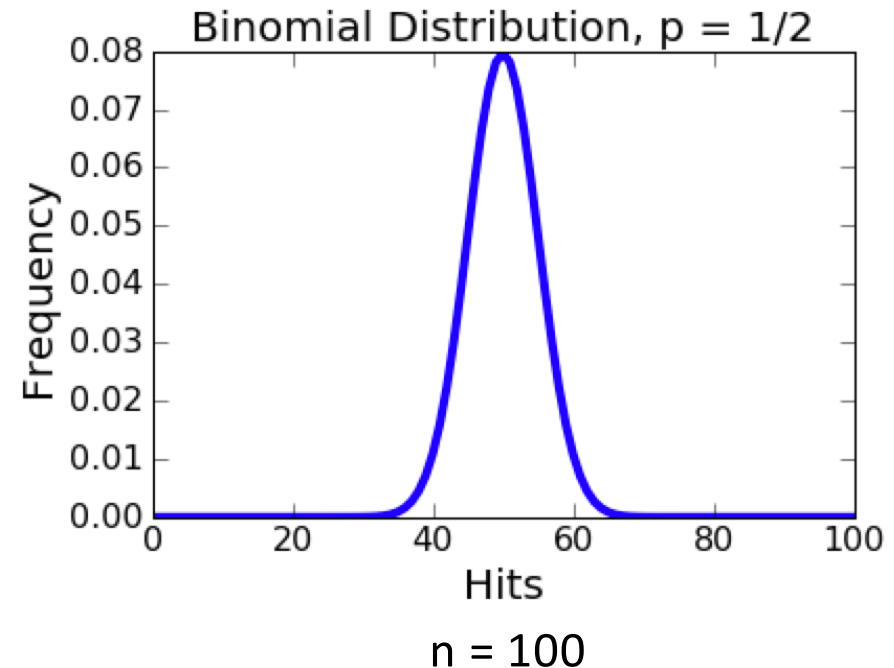
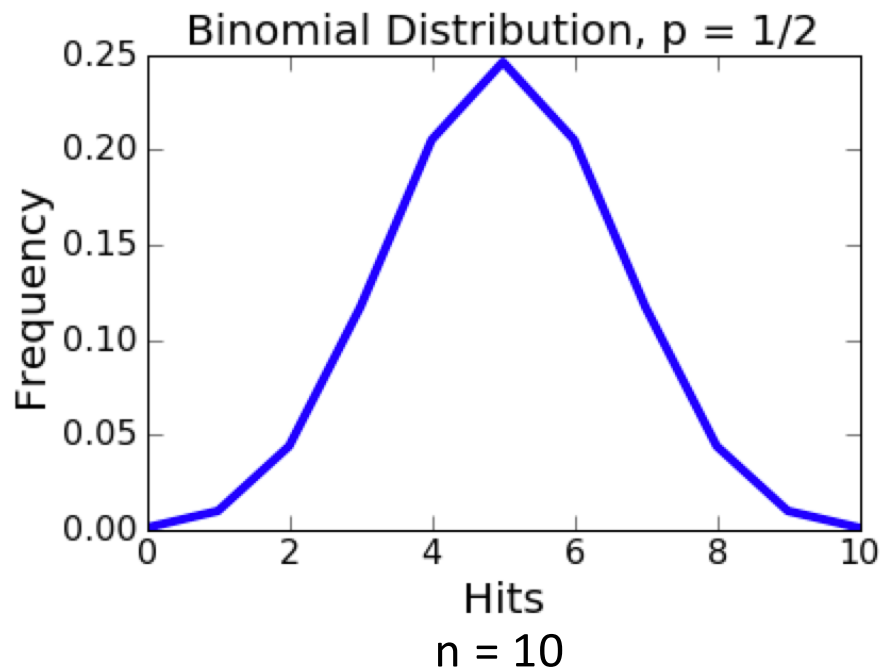
- Multinomial distribution generalizes to case of more than two, but a discrete, number of outcomes on each trial



# Binomial Distributions

---

- Mean:  $np$
- Variance:  $np(1 - p)$
- If  $n$  is large enough, then binomial distribution is approximated by a normal distribution, with mean  $np$  and variance  $np(1 - p)$



# Binomial Distributions: Examples

- Any problem where have two outcomes on each trial, and want to know probability of exactly  $k$  trials succeeding out of  $n$ 
  - If probability of a false positive on test of equipment is known, how many independent tests are needed to ensure equipment is reliable with given probability
  - Probability of exactly  $k$  “heads” in  $n$  flips
  - Suppose individuals with a specific gene have a known probability of eventually contracting a certain disease. Run a lifetime study on a set of individuals, and determine probability of number of individuals who will contract the disease

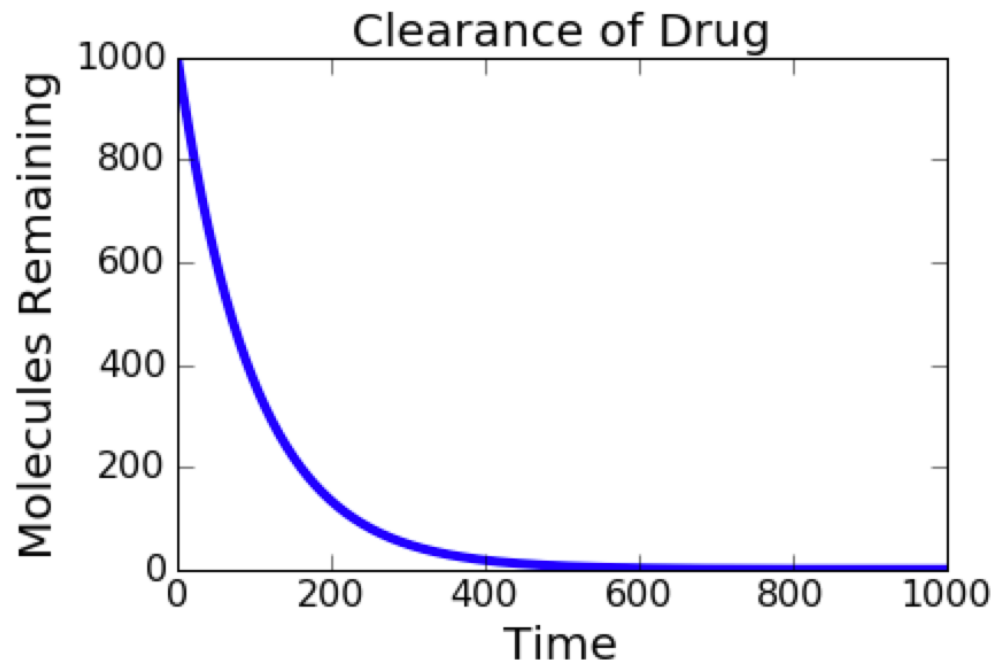


# Exponential Distributions

---

- Suppose  $p$  is probability of an event occurring (e.g., a molecule of a drug being cleared from the body)
- Probability event has not occurred after  $t$  time steps (e.g., molecule still in body):

$$(1 - p)^t$$



# Exponential Distributions

---

- Probability density function

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0. & \text{if } x < 0 \end{cases}$$

- Mean:  $\frac{1}{\lambda}$

- Variance:  $\frac{1}{\lambda^2}$

- Cumulative distribution function

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

- Can generate exponential distributions using `random.expovariate(lambd)`, where `lambd` is 1 divided by mean of distribution
- Discrete version called the geometric distribution

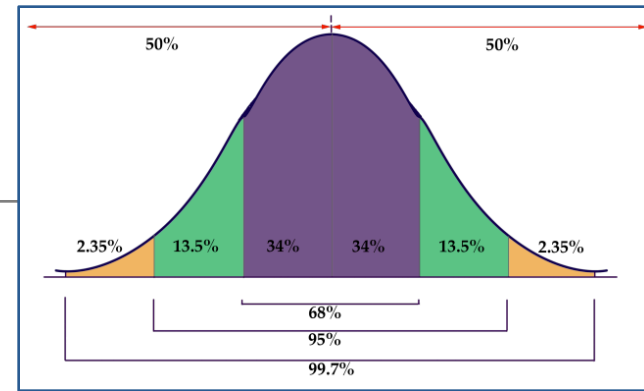
# Exponential Distributions: Examples

- Modeling inter-arrival times (time between events), e.g.:
  - cars entering a highway,
  - or requests for a Web page,
  - or job requests on a server
- Time for a radioactive particle to decay (clicks on a Geiger counter)
- Time until default on payment to debt holders
- Service time of agents in a system (how long a bank teller takes to serve a customer; or waiting times for an airline call center)





# Back to our Empirical Rule



## ■ If we assume that

- Mean estimation error is zero
- Distribution of the errors in the estimates is normal (Gaussian)

## ■ Then by computing mean ( $\mu$ ) and standard deviation ( $\sigma$ ) of estimates, 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:

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

Now know what a normal distribution looks like, and appears that they occur frequently

# 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).
- Can we check that Empirical Rule actually works?

# A Digression

---

- **SciPy** library contains many useful mathematical functions used by scientists and engineers
- `scipy.integrate.quad` has up to four arguments
  - a function or method to be integrated,
  - a number representing the lower limit of the integration,
  - a number representing the upper limit of the integration, and
  - an optional tuple supplying values for all arguments, except the first, of the function to be integrated
- `scipy.integrate.quad` returns a tuple
  - Approximation to result
  - Estimate of absolute error

# Checking the Empirical Rule

---

```
import scipy.integrate

def checkEmpirical(numTrials):
    for t in range(numTrials):
        mu = random.randint(-100, 100)
        sigma = random.randint(1, 100)
        print('For mu =', mu, 'and sigma =', sigma)
        for numStd in (1, 1.96, 3):
            area = scipy.integrate.quad(gaussian,
                                        mu-numStd*sigma,
                                        mu+numStd*sigma,
                                        (mu, sigma))[0]
            print(' Fraction within', numStd,
                  'std =', round(area, 4))
```

```
checkEmpirical(5)
```

# Checking Empirical Rule

---

For  $\mu = 58$  and  $\sigma = 54$

Fraction within 1 std = 0.6827

Fraction within 1.96 std = 0.95

Fraction within 3 std = 0.9973

For  $\mu = -69$  and  $\sigma = 36$

Fraction within 1 std = 0.6827

Fraction within 1.96 std = 0.95

Fraction within 3 std = 0.9973

For  $\mu = -22$  and  $\sigma = 18$

Fraction within 1 std = 0.6827

Fraction within 1.96 std = 0.95

Fraction within 3 std = 0.9973

For  $\mu = -18$  and  $\sigma = 53$

Fraction within 1 std = 0.6827

Fraction within 1.96 std = 0.95

Fraction within 3 std = 0.9973

For  $\mu = 48$  and  $\sigma = 82$

Fraction within 1 std = 0.6827

Fraction within 1.96 std = 0.95

Fraction within 3 std = 0.9973

# Checking Empirical Rule

---

- Empirical rule really applies to normal distributions
- But if binomial distribution approaches normal, is empirical rule a decent approximation if errors follow a binomial distribution?

For  $n = 50$

Fraction within 1.0 std = 0.7974

Fraction within 1.96 std = 0.9672

Fraction within 3.0 std = 0.9991

For  $n = 200$

Fraction within 1.0 std = 0.7112

Fraction within 1.96 std = 0.96

Fraction within 3.0 std = 0.9977

For  $n = 1000$

Fraction within 1.0 std = 0.7033

Fraction within 1.96 std = 0.9537

Fraction within 3.0 std = 0.9974

Normal  
distribution:

- 68%
- 95%
- 99.7%

# Where are we?

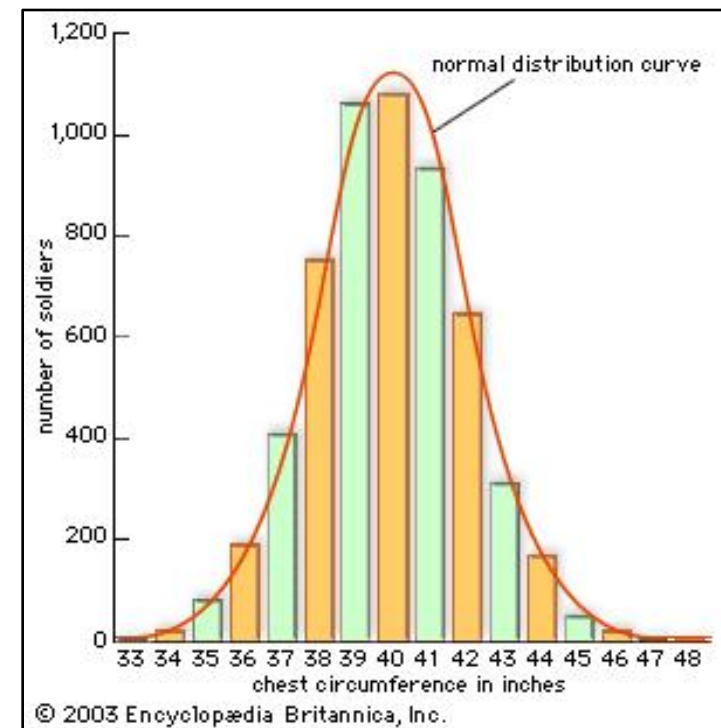
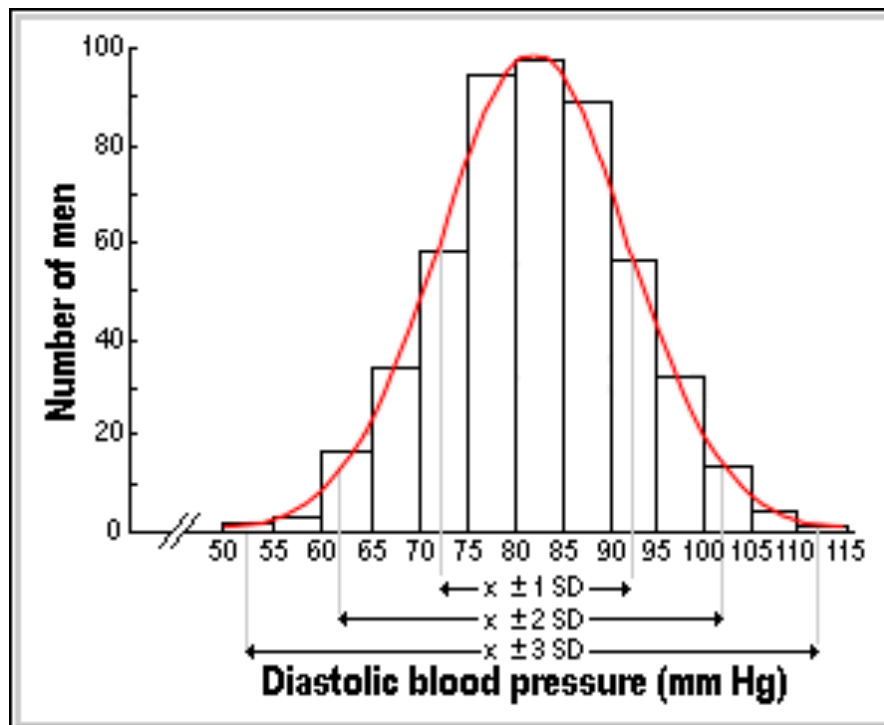
---



- 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).
- Can we check that Empirical Rule actually works?
  - “Yes” for normal distribution of errors
  - Roughly “yes” for binomial distribution of errors
- How do we know distribution is normal?
  - Could measure it empirically and see how well a Gaussian fits to it? – Expensive!
  - Might know from first principles that distribution is normal

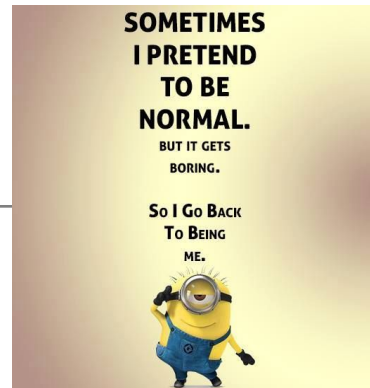
# Good News: Lots of Normal Distributions

- Occur a lot in real world!





# But: Not All Distributions Are Normal



- Empirical rule works for normal distributions
- While many distributions in real life are normal, should not just assume everything is
- For example, 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 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
  - This suggests that we can use the empirical rule even if the distribution of events is not normal, provided the distribution of errors over many trials is normal
- We'll come back to the CLT and what it implies shortly
- But first, let's think about what to do if we can't run multiple simulation trials



# Recall Inferential Statistics



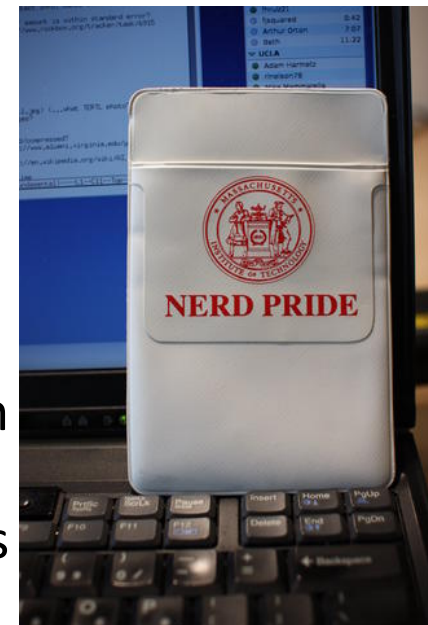
- Make an inference about some property of a population by examining one or more random samples drawn from that population
  - infer property of population based on statistics of sub-population
  - avoid cost of having to look at entire population, if very large
  - handle cases where not feasible to sample whole population
- With Monte Carlo simulation, generate lots of random samples and use them to make inferences and to compute confidence intervals for the inferences using empirical rule
  - allows us to estimate likelihood of inference
  - useful when variation in value due to noise or random effects
- 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% disapprove. The poll surveyed 1,972 registered voters and has a margin of error of plus or minus 2 percentage points."* – August 2017
  - need different approach when can't simulate randomness

# Probability Sampling



"The Tortoise And The Hare" is actually a fable about small sample sizes.

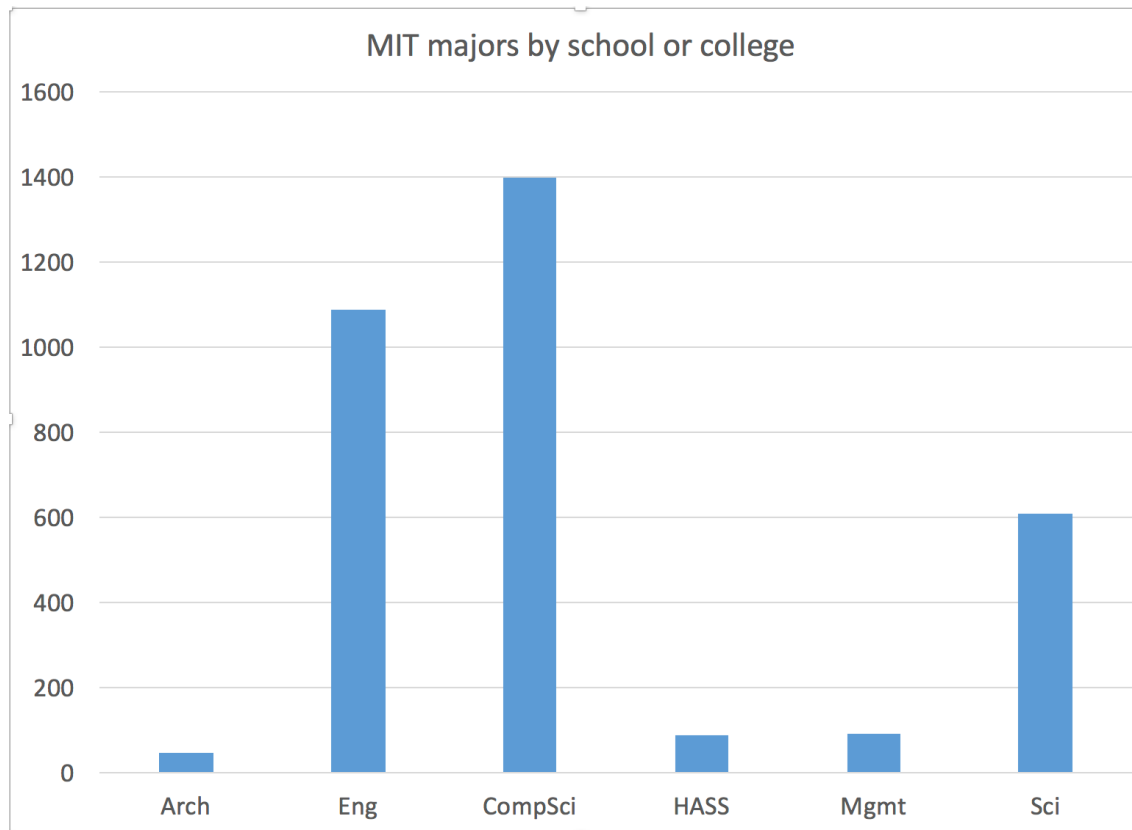
- Alternative to simulation is to directly sample from population
- Assume each member of population has nonzero probability of being included in sample
  - Select subpopulation by sampling at random
- **Simple random sampling**: each member of whole population has equal chance of selection
- Not always appropriate
  - Popular myth: all MIT undergrads are CS majors
  - Take a simple random sample of 100 students
    - What might you conclude about MIT student majors from such a sample?
    - What might you conclude about interests of MIT students based on such a sample?



# Probability Sampling



"The Tortoise And The Hare" is actually a fable about small sample sizes.



In random sample of 100, expect:

- 1 Architecture & Planning
- 33 Engineering
- 42 Computer Science
- 3 HASS
- 3 Management
- 18 Science

In simple sampling over 10,000 trials, saw that on average

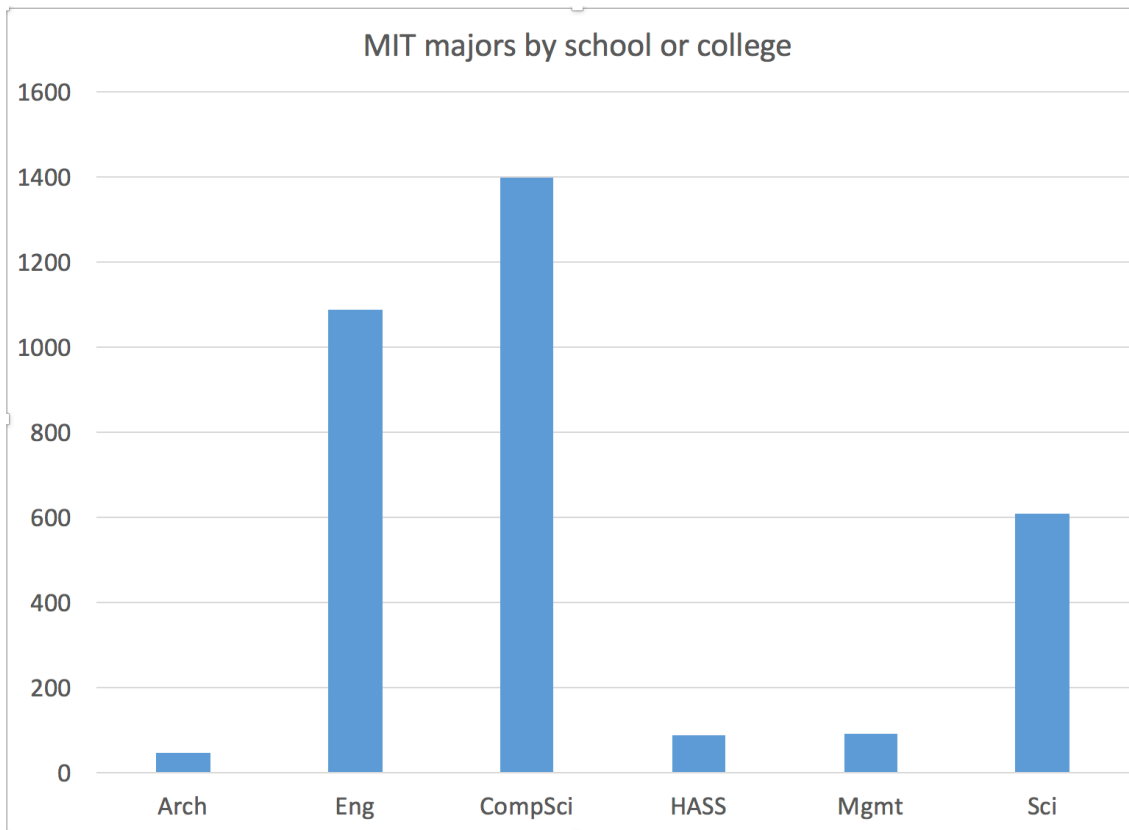
But in simple sampling over 10,000 trials

- 26% of trials have no SAP, 7% no SHASS, 6% no SLOAN
- **0.14% with only SOE and SOS and CompSci**
- **so 2 of 1000 times, would conclude only Engineering & Science & Computation, but might conclude no SAP a quarter of time**

# Probability Sampling



"The Tortoise And The Hare" is actually a fable about small sample sizes.



In random sample of 100, expect:

- 1 Architecture & Planning
- 33 Engineering
- 42 Computer Science
- 3 HASS
- 3 Management
- 18 Science

In simple sampling over 10,000 trials, saw that on average

Suppose I sample 1% (or 33) of the UG population over 10,000 trials

- 64% of trials have no SAP, 42% no SHASS, 40% no SLOAN
- **10.3% with only SOE and SOS and Computation**
- **so 1 of 10 times, would conclude only Engineering & Science & Computation; might conclude no SAP 2/3rds of time**



# Stratified Sampling



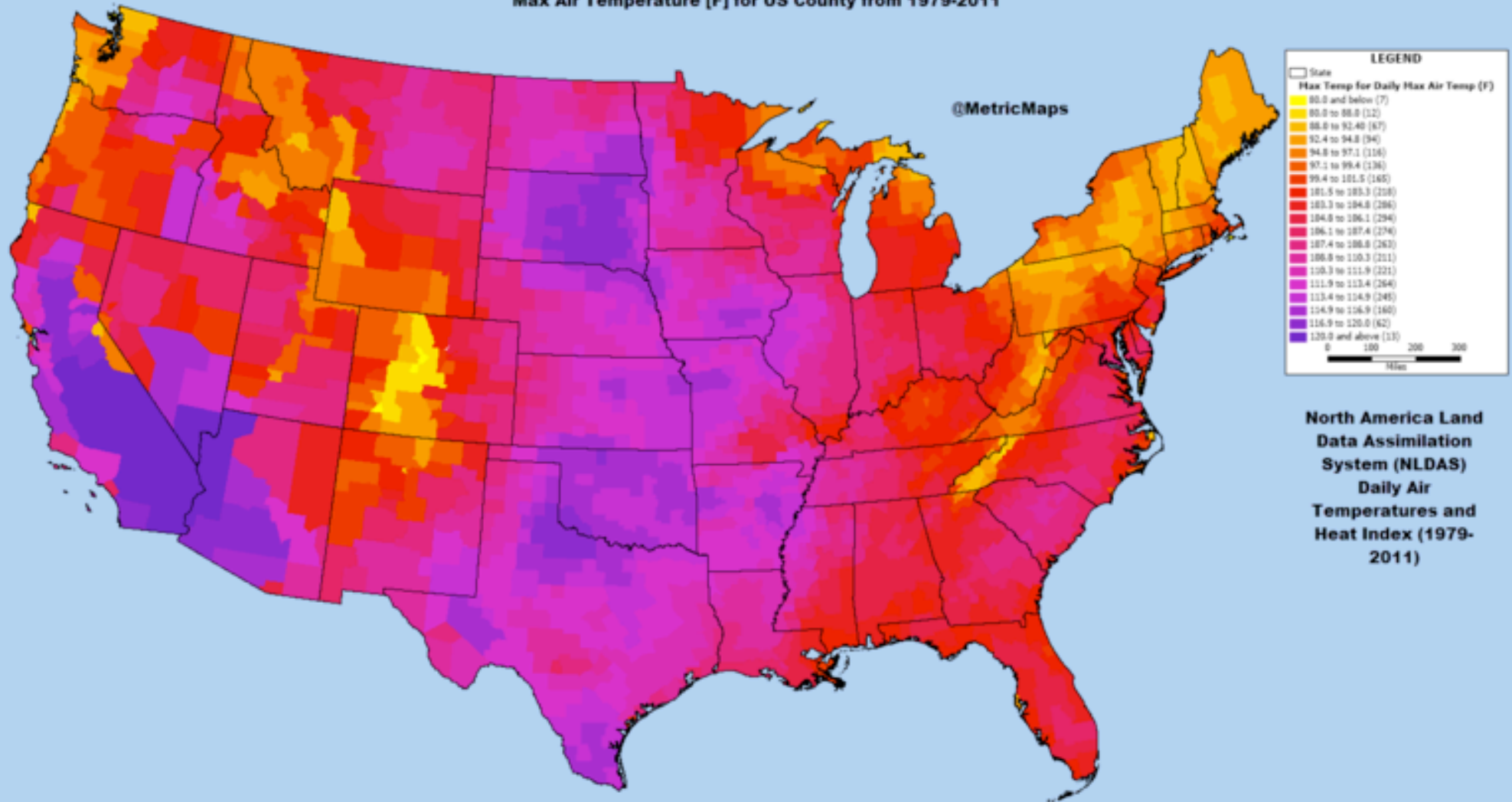
- Stratified sampling
  - Partition population into subgroups
  - Take simple random sample from each subgroup (size of sample reflects subgroup's relative size)
- Useful when there are small subgroups that should be represented (e.g., political polls)
- Useful when subgroups should be represented proportional to their share of population
  - E.g. if want to get opinions from MIT UG population, for sample of size 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 in entire population
- Requires care to do properly
  - How many samples to draw from each subgroup?
  - What are appropriate subgroups?
  - Why do we think subgroups' variability is smaller?
- **We'll stick to simple random samples**

Poll

Note: to sample 1% of eligible voters in the US, you would need ~2.35M opinions!  
This size is why good pollsters use stratified sampling

# Sampling to Estimate Mean Temperature

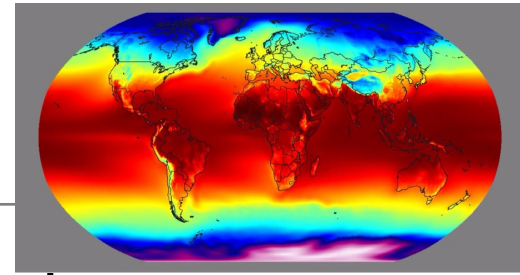
Max Air Temperature [F] for US County from 1979-2011





# Data

---

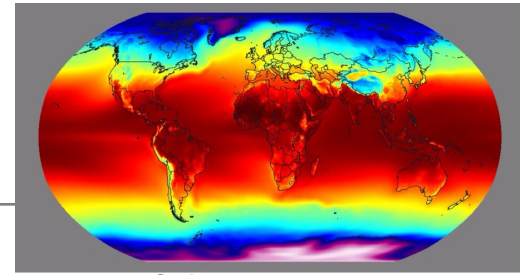


- From U.S. National Centers for Environmental Information (NCEI)
- Daily high and low temperatures 1961-2015
  - 21 different US cities
    - ALBUQUERQUE, BALTIMORE, BOSTON, CHARLOTTE, CHICAGO, DALLAS, DETROIT, LAS VEGAS, LOS ANGELES, MIAMI, NEW ORLEANS, NEW YORK, PHILADELPHIA, PHOENIX, PORTLAND, SAN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



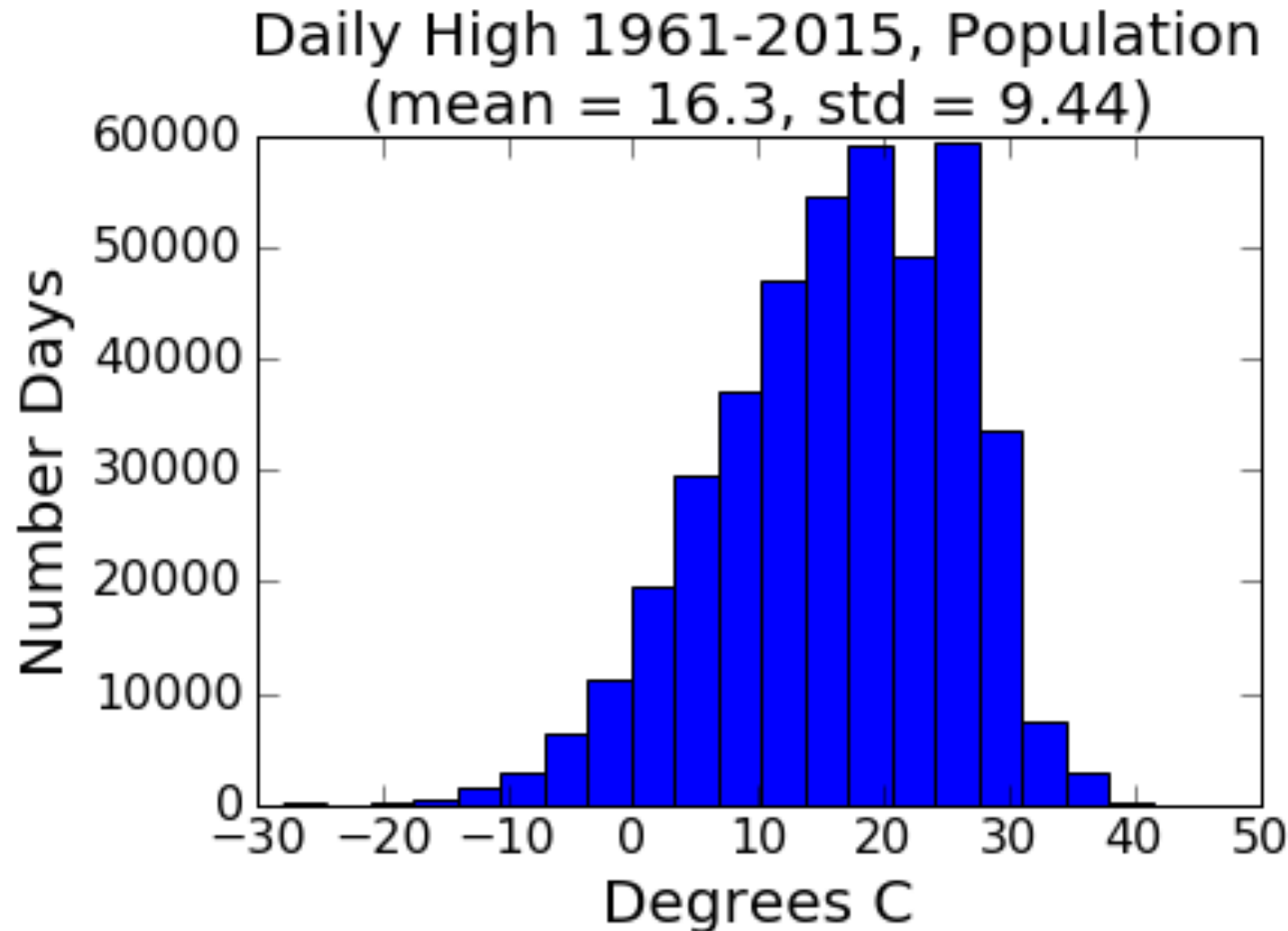
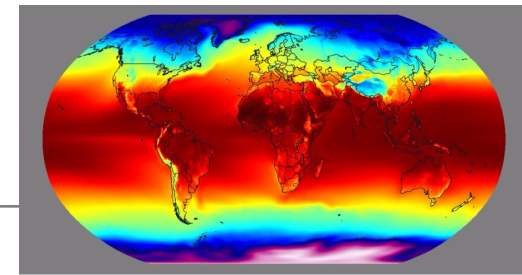
# New in Code

---



- Code `getHighs` extracts high temperatures from file, code `getMeansAndSDs` computes mean and standard deviation for entire population and sample from population
- `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
- Going to plot using histogram – count number of times a particular value occurs

# Histogram of Entire Population



Or in °F:  
61.3  $\pm$  17.0

Why is  
 $\sigma$  so large?

Is this really a  
normal  
distribution?

Would like to infer average temperature with some level of confidence. Empirical rule would help, but if this distribution is not normal, then errors about mean not normal?

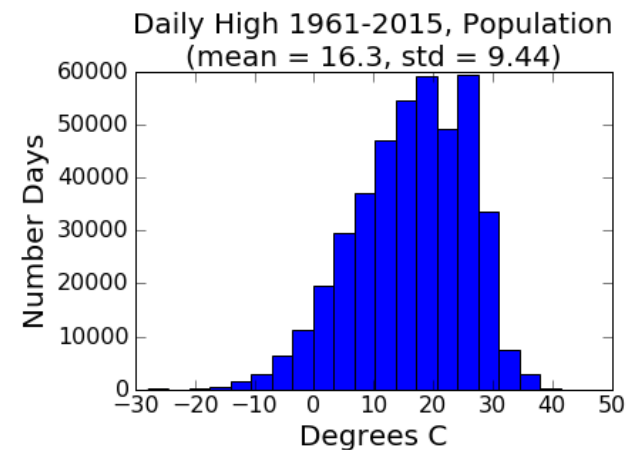
# Some Observations



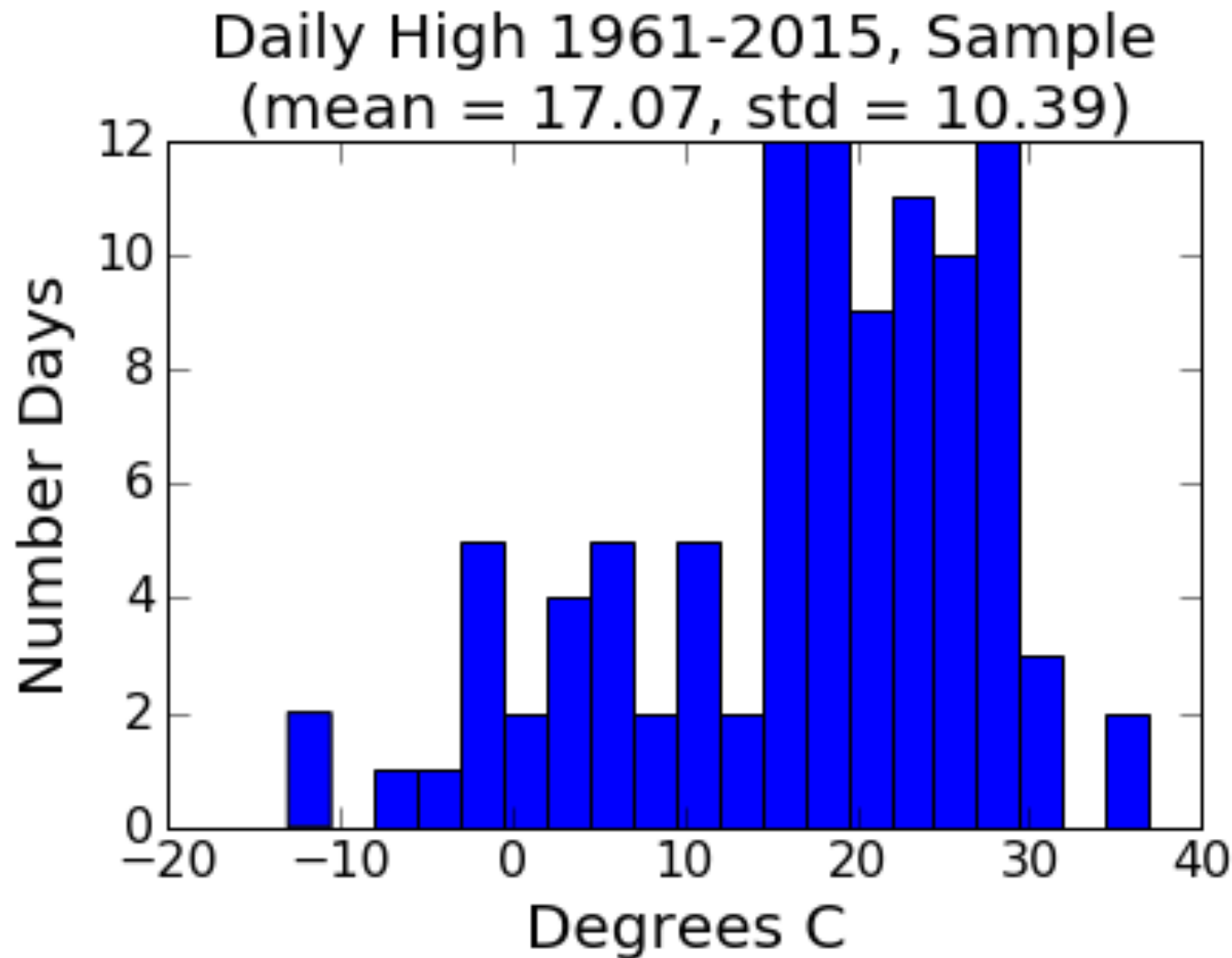
- Why large standard deviation:

Poll

- 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 use 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 take a different subsample of 100 temperatures, compute their mean, and do this 1000 times. What do those means look like?



# 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

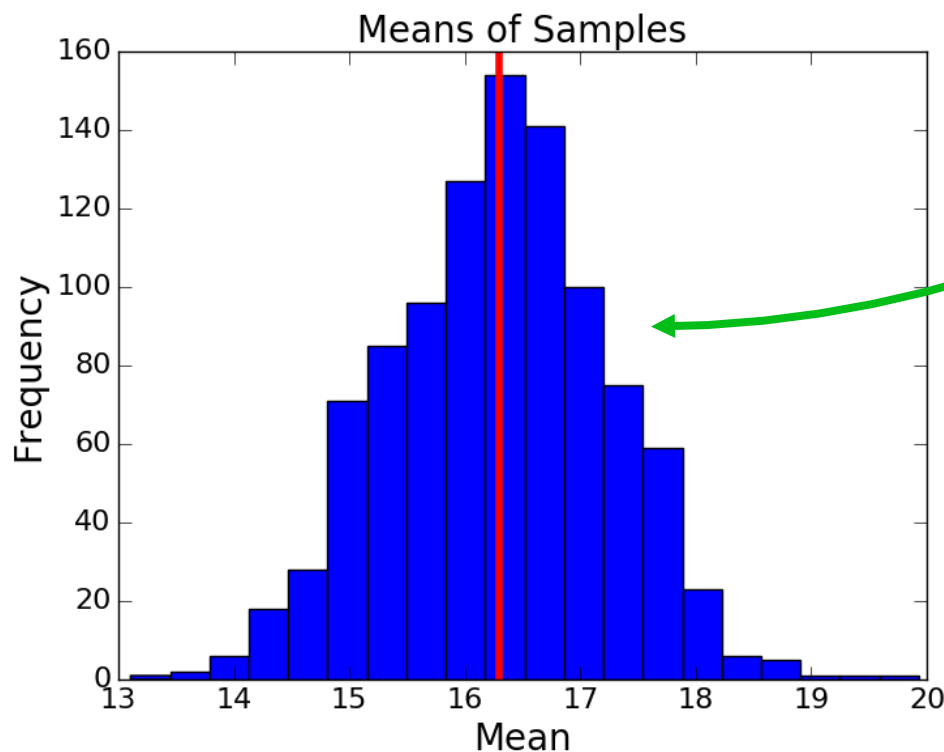
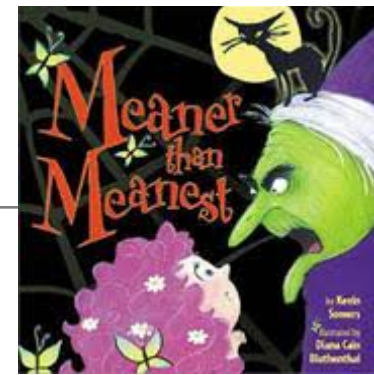
---

- 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**





# Mean of 1000 Means

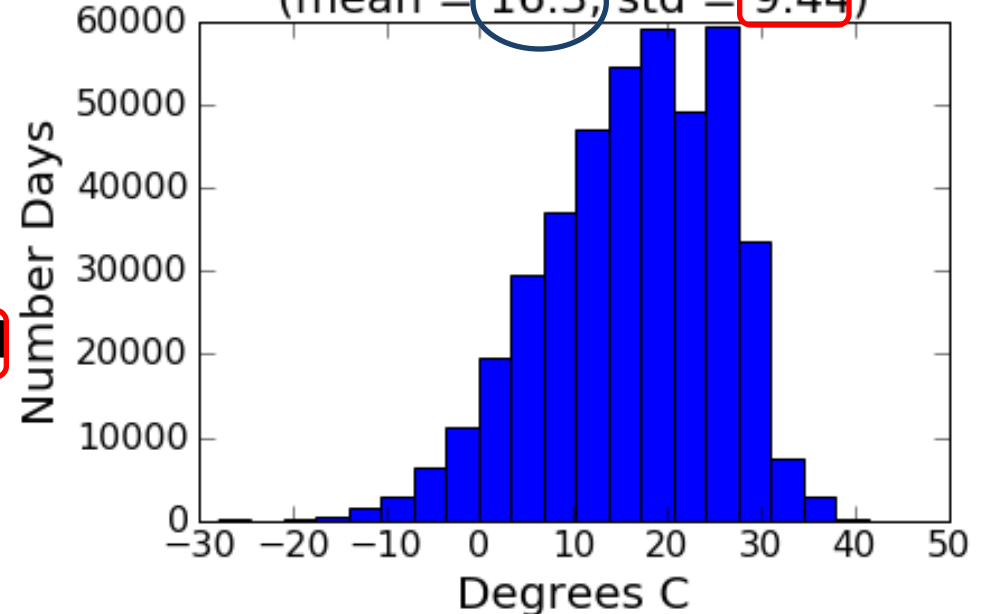


Looks pretty normal!

In comparison

Mean high temperature over entire population

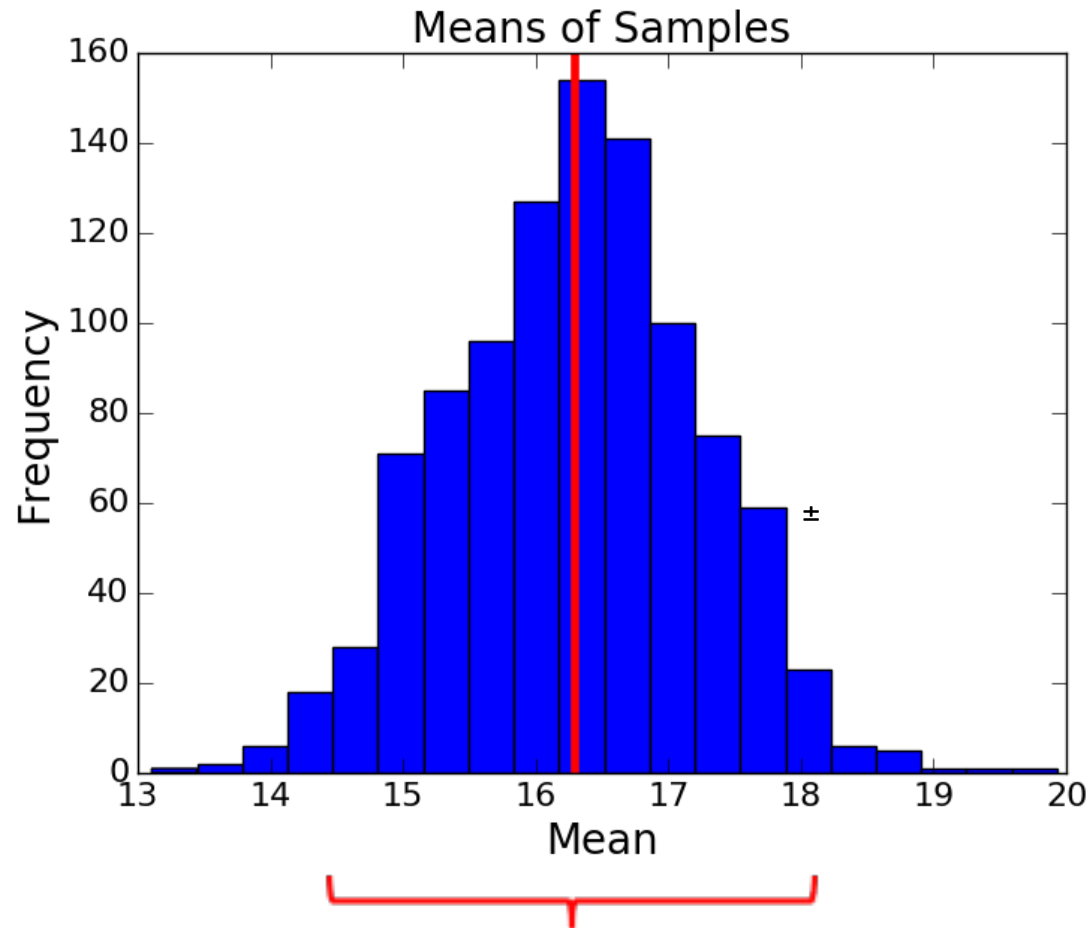
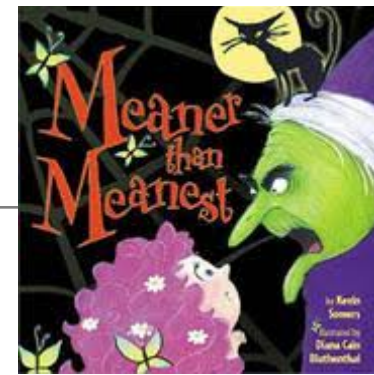
Daily High 1961-2015, Population  
(mean = 16.3, std = 9.44)



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

Mean of mean high temperatures,  
100 samples, 1000 trials

# Mean of 1000 Means



Mean of sample Means = 16.3

Standard deviation of sample means = 0.94

What's the 95% confidence interval?

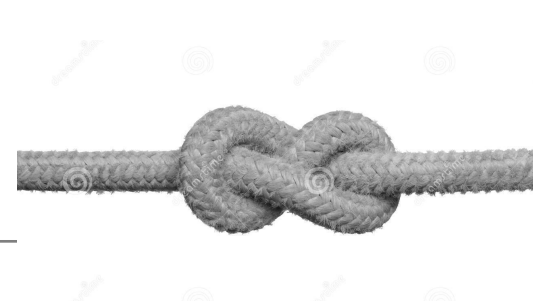
$16.28 \pm 1.96 \times 0.94$

Or 14.5 - 18.1

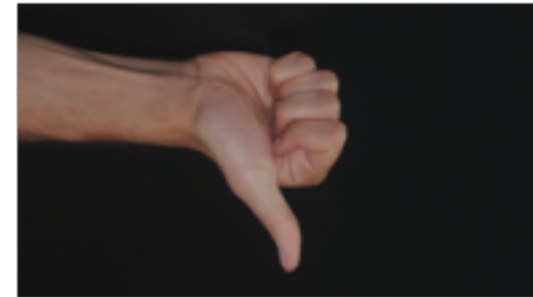
Includes population mean, but still pretty wide interval

Suppose we want a tighter bound?

# Getting a Tighter Bound

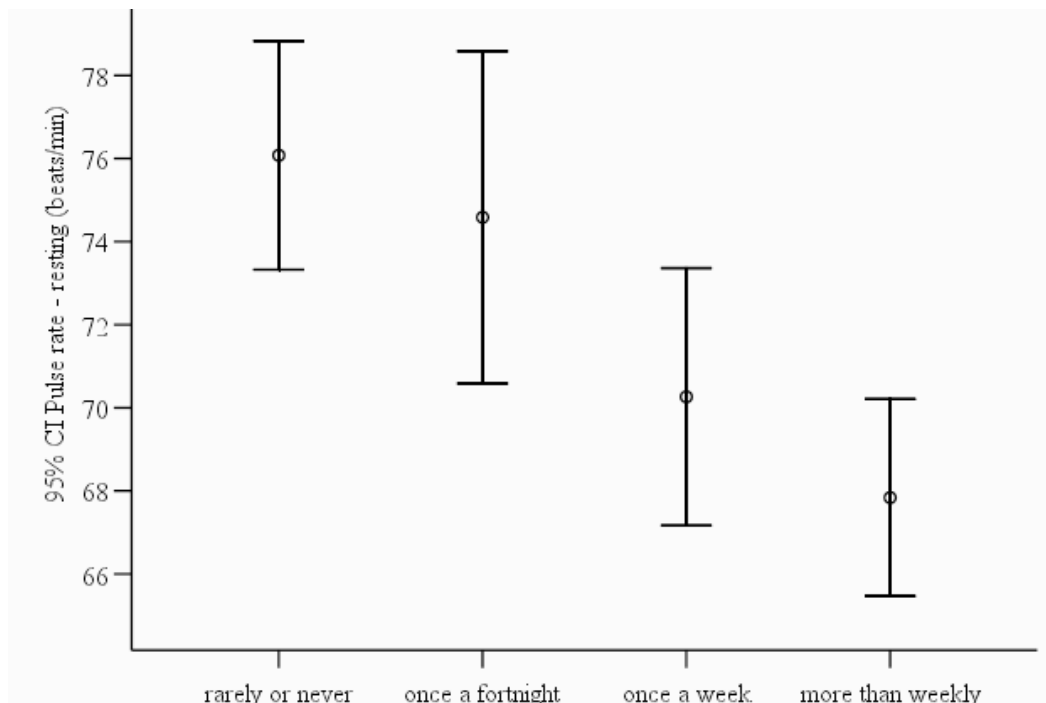


- Will drawing more samples help?
  - Let's try increasing from 1000 to 2000 trials
  - Standard deviation of estimated mean temperature goes from **0.943** to **0.946**
- How about larger samples?
  - Let's try increasing sample size from 100 to 200, but sticking with 1000 trials
  - Standard deviation of estimated mean temperature goes from **0.943** to **0.662**
- This suggests that if we want reasonable confidence in our result, we need to think about how big a sample size to use

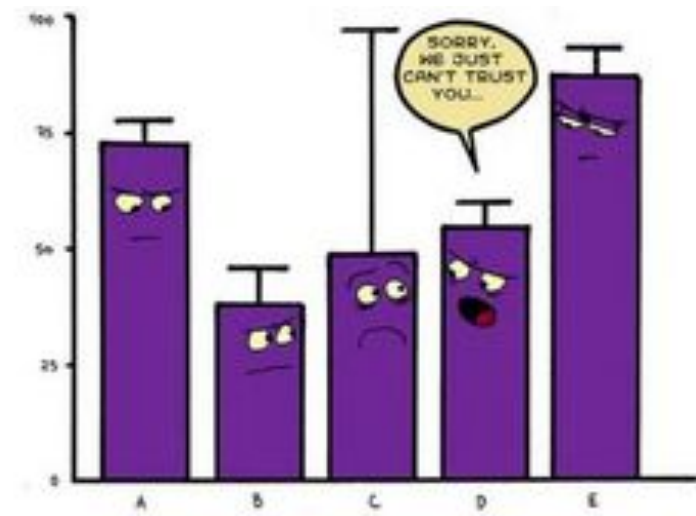


# Error Bars, a Digression

- Graphical representation of the variability of data
- Way to visualize uncertainty
  - Plot mean value and size of variance



[https://upload.wikimedia.org/wikipedia/commons/1/1d/Pulse\\_Rate\\_Error\\_Bar\\_By\\_Exercise\\_Level.png](https://upload.wikimedia.org/wikipedia/commons/1/1d/Pulse_Rate_Error_Bar_By_Exercise_Level.png)



When confidence intervals don't overlap, we can conclude that means are **statistically significantly different** at some level of confidence (e.g., 95%).

Overlapping confidence intervals does not imply lack of significance.

# Let's Look at Error Bars for Temperatures

```
def showErrorBars(population, sizes, numTrials):
```

```
    xVals = []
```

```
    sizeMeans, sizeSDs = [], []
```

```
    for sampleSize in sizes:
        xVals.append(sampleSize)
        trialMeans = []
```

Record mean and std of trials for  
a size;  
Repeat for different sample sizes

For given  
sample size,  
run set of trials  
and compute  
mean of each  
trial

```
        for t in range(numTrials):
            sample = random.sample(population, sampleSize)
            popMean, sampleMean, popSD, sampleSD = \
                getMeansAndSDs(population, sample)
            trialMeans.append(sampleMean)
```

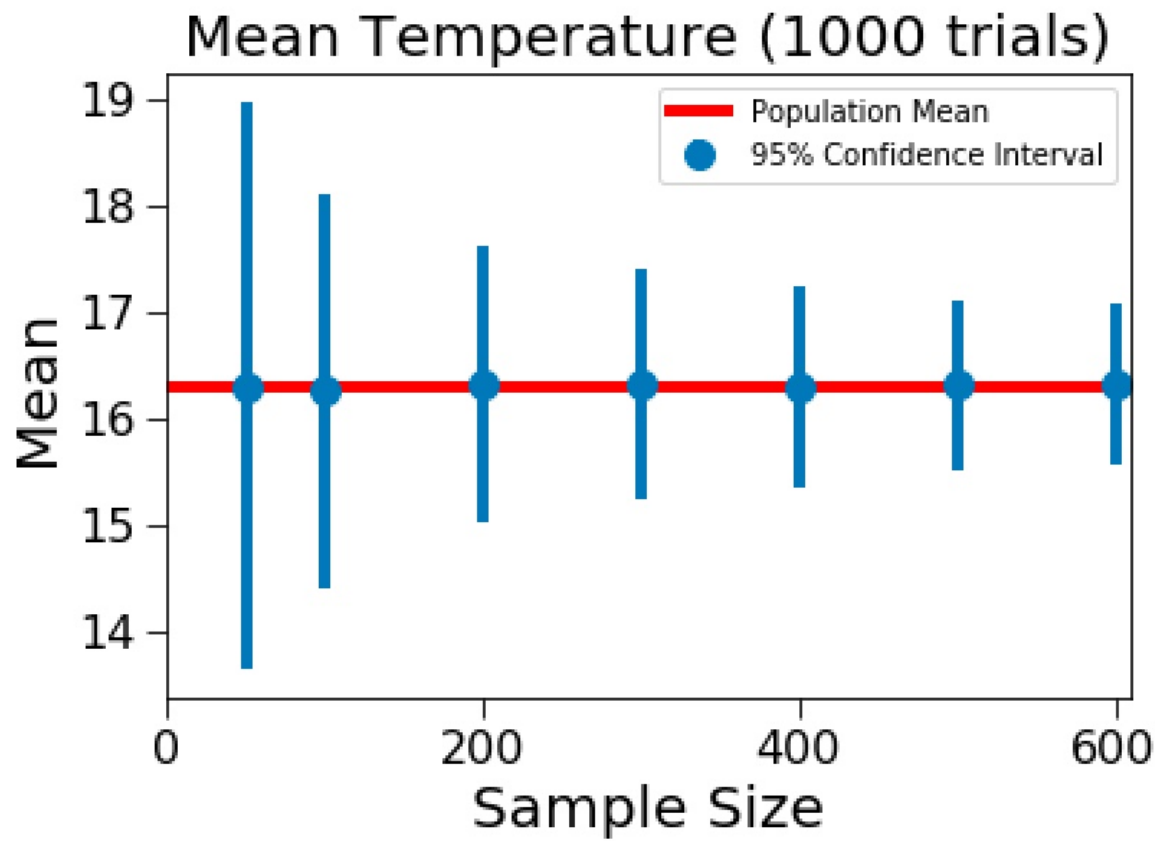
```
        sizeMeans.append(sum(trialMeans)/len(trialMeans))
        sizeSDs.append(numpy.std(trialMeans))
```

```
    pylab.errorbar(xVals, sizeMeans,
                    yerr = 1.96*pylab.array(sizeSDs),
                    fmt = 'o',
                    label = '95% Confidence Interval')
```

Empirical rule:  
Using  $1.96 \times \text{STD}$  accounts for  
95% of normal distribution

# Sample Size and Standard Deviation

```
population = getHighs()  
showErrorBars(population,  
               (50, 100, 200, 300, 400, 500, 600), 1000)
```



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

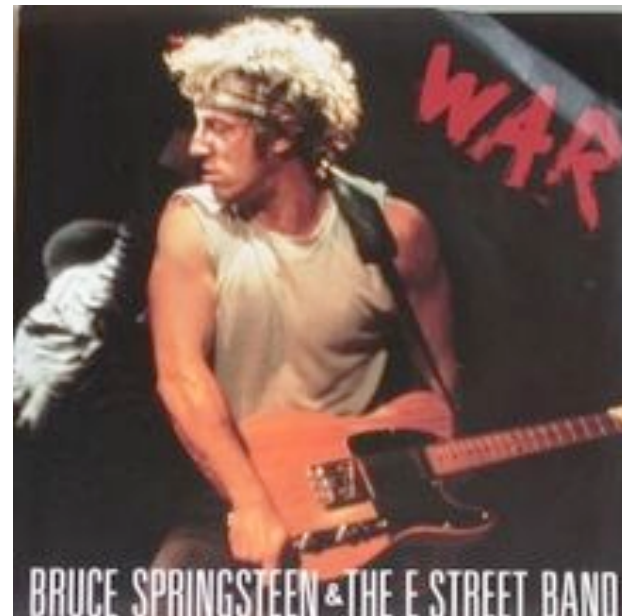
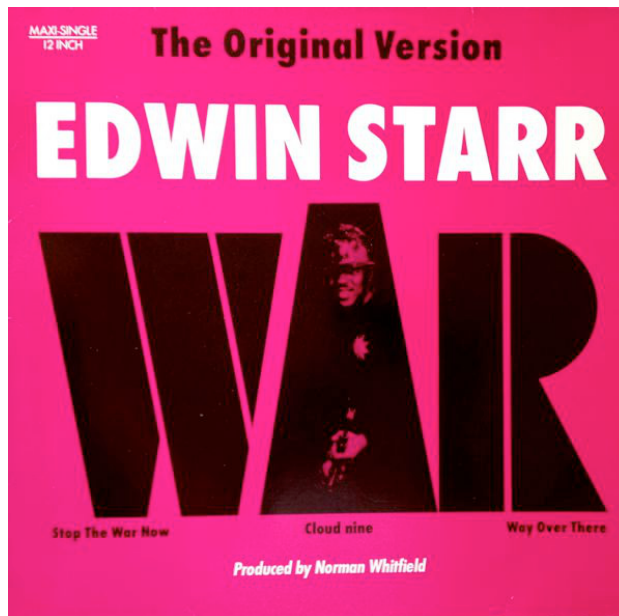
Note how size of error bars stabilizes after about 500 samples (or even 200 samples)

Does this help decide sample size?

# Larger Samples Seem to Be Better



- Going from a sample size of 50 to 600 reduced the standard deviation from  $1.36C$  to  $0.39C$
- But we are now looking at  $600 * 1000 = 600,000$  examples
- Sampling – what is it good for? (what has it bought us?)
  - **“Absolutely Nothing!”**
  - Entire population contained  $\sim 422,000$  samples





# What Can We Conclude from 1 Sample?

---

- Mean of means suggests can draw inference with some level of confidence about population from set of samples
- But would like to avoid taking lots of samples
- What if we just took **one** sample? What can we conclude?
- More than you might think, thanks to the **Central Limit Theorem!**



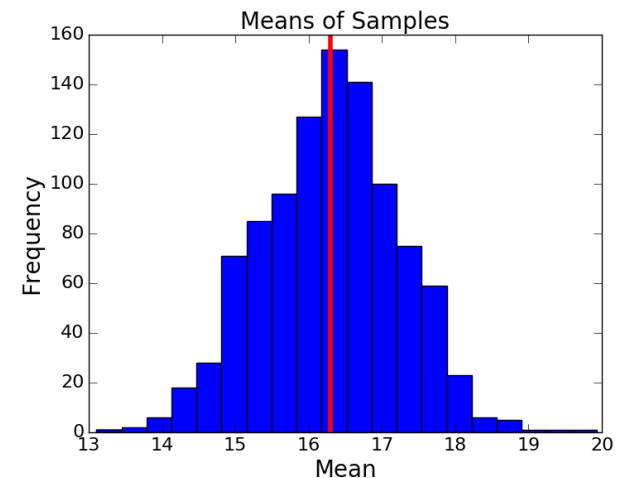


# The Central Limit Theorem (CLT)

- Given a sufficiently large sample:
  - 1) The means of the samples in a set of independent 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.

If sample size is large enough, variance of sample means is small; hence mean of one sample will be close to mean of means, and thus to population mean

Recall temperature example



Mean of means: 16.294  
Population mean: 16.299

Variance of sample means: 0.889  
Population variance: 89.068  
Sample size: 100

# So let's check this out

---

- Central Limit Theorem appears to work for mean high temperature data
- What about other cases where we know the distribution of events is not normal?
  - Rolling a set of dice



- Betting on roulette



# Checking CLT for a Continuous Die



Roll different groups of dice, and measure mean value across group

```
def plotMeans(numDice, numRolls, numBins, legend, color, style):  
    means = []
```

Run  
multiple  
trials

```
    for i in range(numRolls//numDice):  
        vals = 0  
        for j in range(numDice):  
            vals += 5*random.random() + 1  
        means.append(vals/float(numDice))
```

Using continuous values between 1  
and 6

Compute average over set of dice

```
    pylab.hist(means, numBins, color = color, label = legend,  
               weights = [1/len(means)]*len(means),  
               hatch = style)
```

Plot histogram  
of means

```
    return getMeanAndStd(means)
```

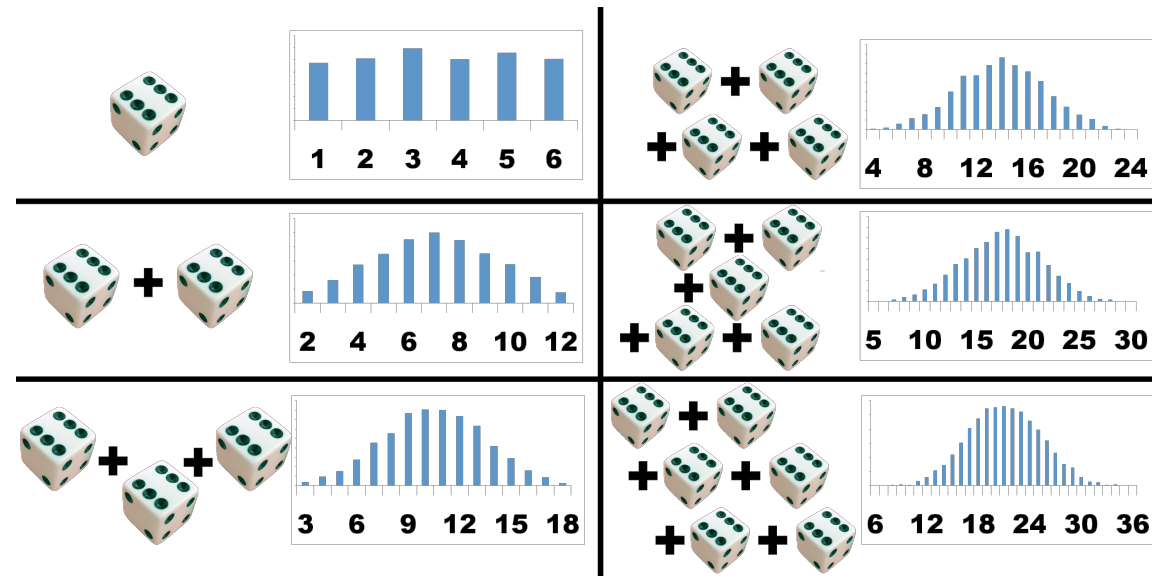
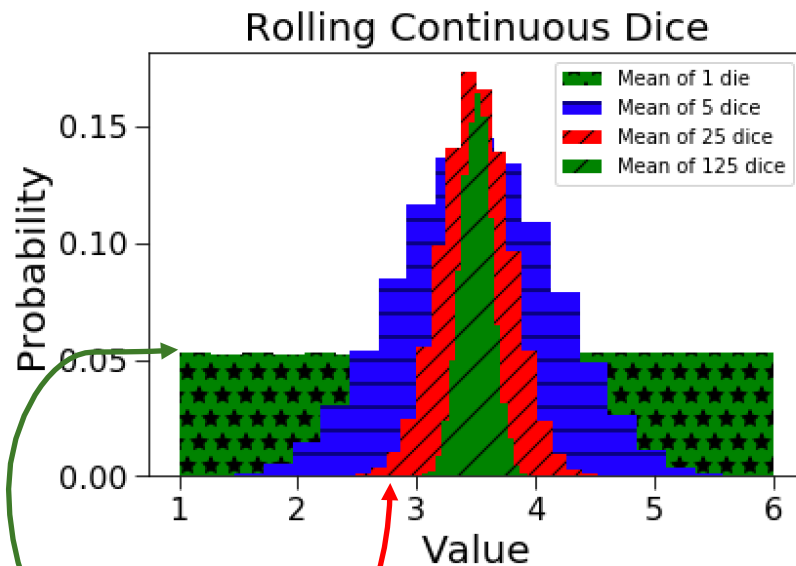
```
mean, std = plotMeans(1, 1000000, 19, '1 die', 'b', '*')  
print('Mean of rolling 1 die =', str(mean) + ', ', 'Std =', std)  
mean, std = plotMeans(50, 1000000, 19, 'Mean of 50 dice', 'r', '//')  
print('Mean of rolling 50 dice =', str(mean) + ', ', 'Std =', std)  
pylab.title('Rolling Continuous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



Single event

Set of events

Distribution of mean values of 1 die -- uniform

Distribution of mean values of 25 dice -- normal

# Try It for Roulette

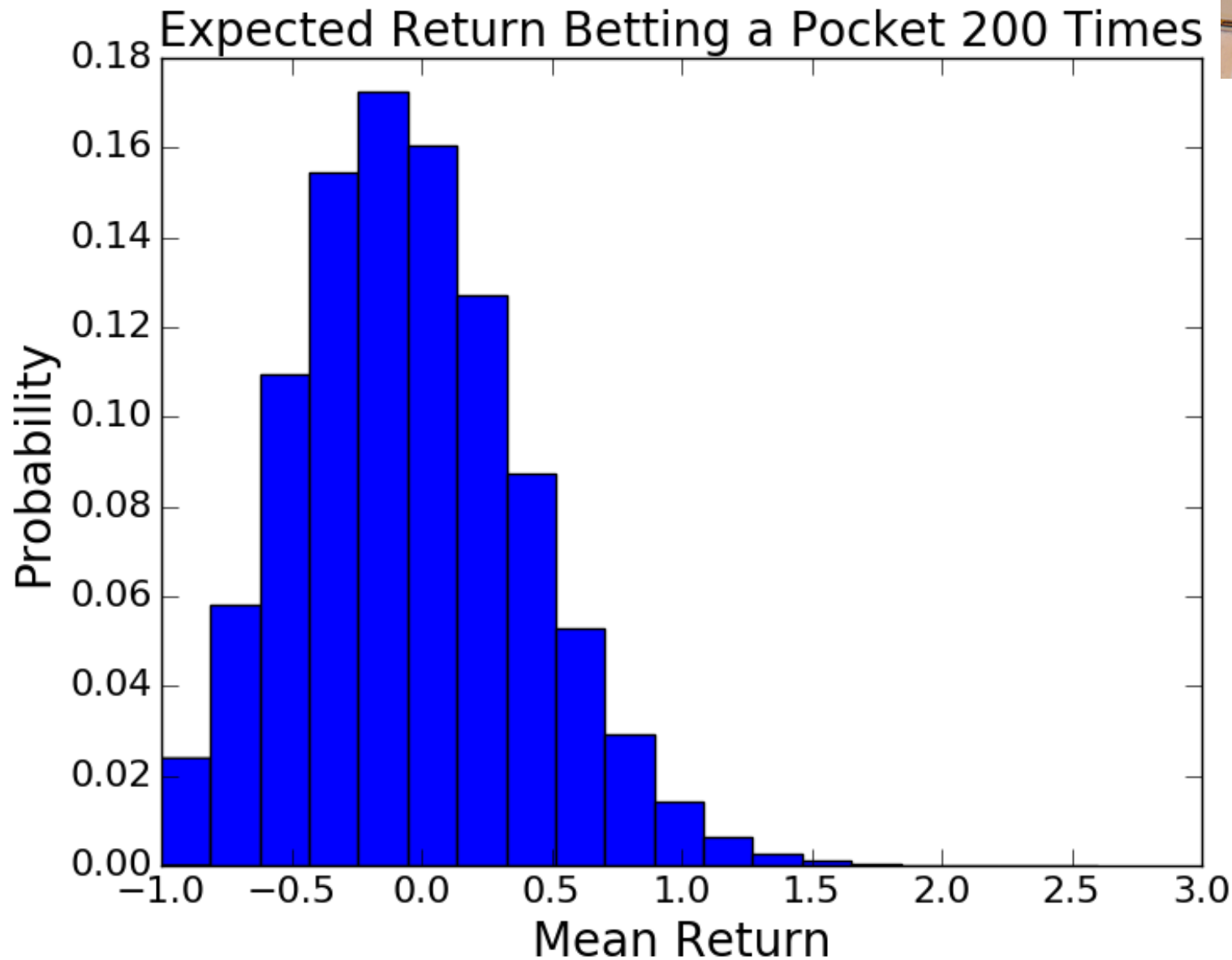


```
numTrials = 1000000
numSpins = 200
game = FairRoulette()
```

```
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')
```

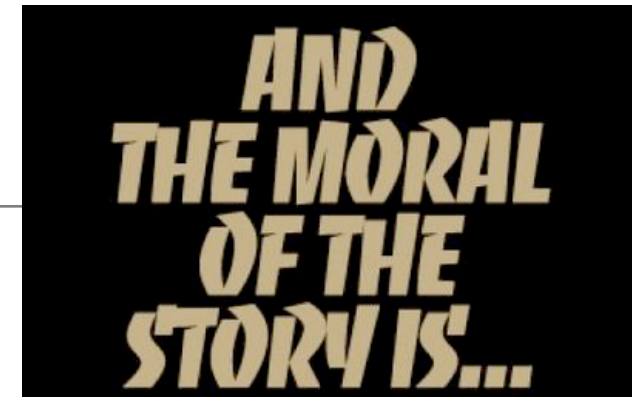
# Betting a Pocket in Fair Roulette



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

---



**It's Time For A Break**



# Where are we?

---



- Goal: given a population, want to infer properties of mean of population
  - Take a single sample (of large enough size) of population
  - Can measure mean of sample, but is it close to mean of population, and with what confidence?
  - Central Limit Theorem says mean of set of samples is close to population mean, and single sample mean is close to mean of set of samples if large enough sample size
  - Empirical rule says if error estimates are normally distributed, then standard deviation lets us set confidence level
    - e.g., values in  $[\mu - 1.96 * \sigma, \mu + 1.96 * \sigma]$  account for 95% of values
  - Central Limit Theorem implies empirical rule can work even if population not normally distributed, because errors are normally distributed. Just need to estimate standard deviation of population

# 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.
  - Mean of sufficiently large sample close to sample mean
  - But don't know variance of sample means because don't know variance of population. So can't set confidence level
- Time to use the 3<sup>rd</sup> feature
  - Compute *standard error of the mean* (SEM or SE)

# Standard Error of the Mean

---

- Standard error of a statistic (e.g., a mean) based on a sample is the deviation of the statistic for the whole population divided by the root of the sample size

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

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

- This should measure standard deviation in estimating the mean of sub-population
- 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')
```

Compute SEM for given sample size

Estimate SD for 50 trials of give sample size

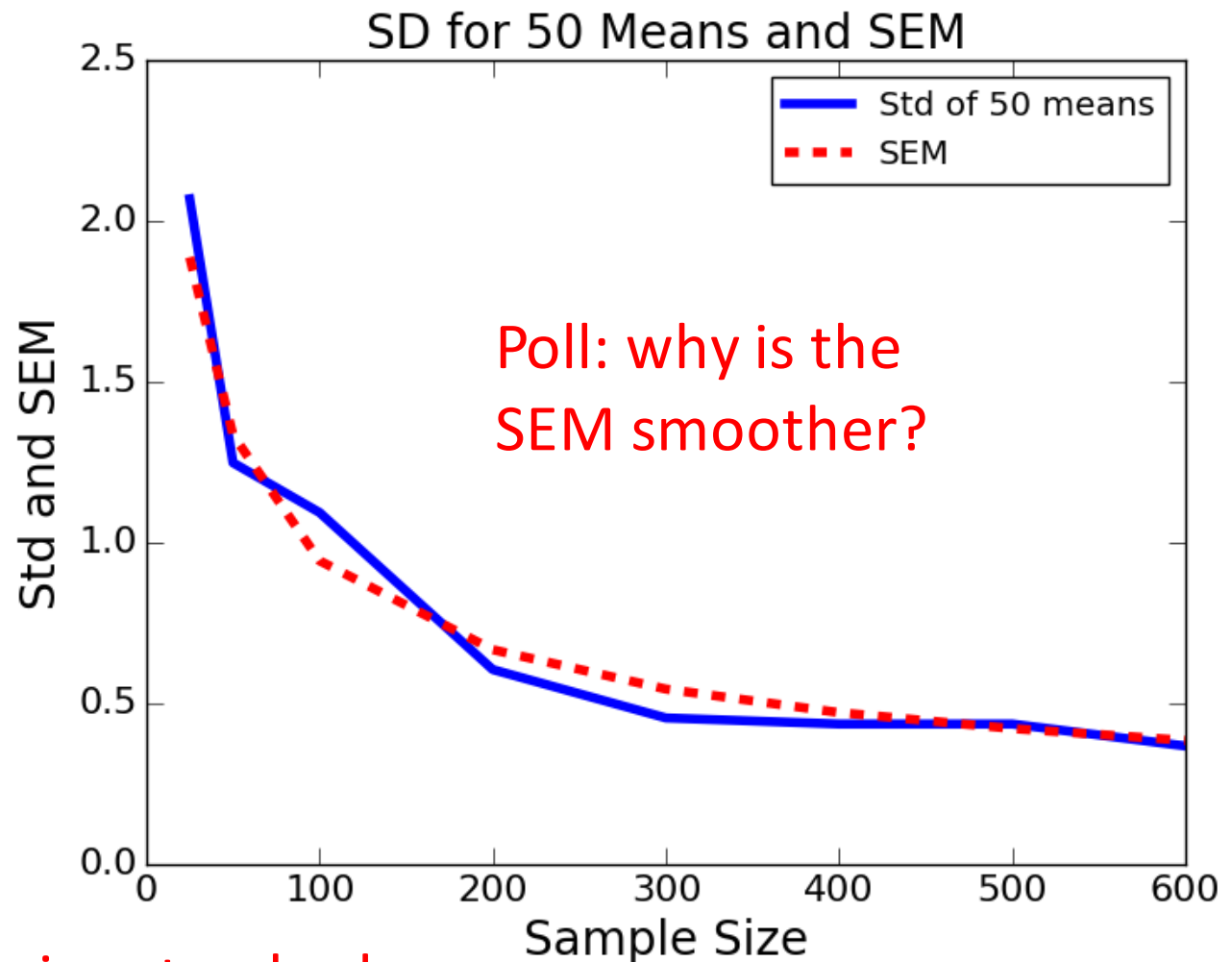
# Standard Error of the Mean

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

But, we don't know standard deviation of population

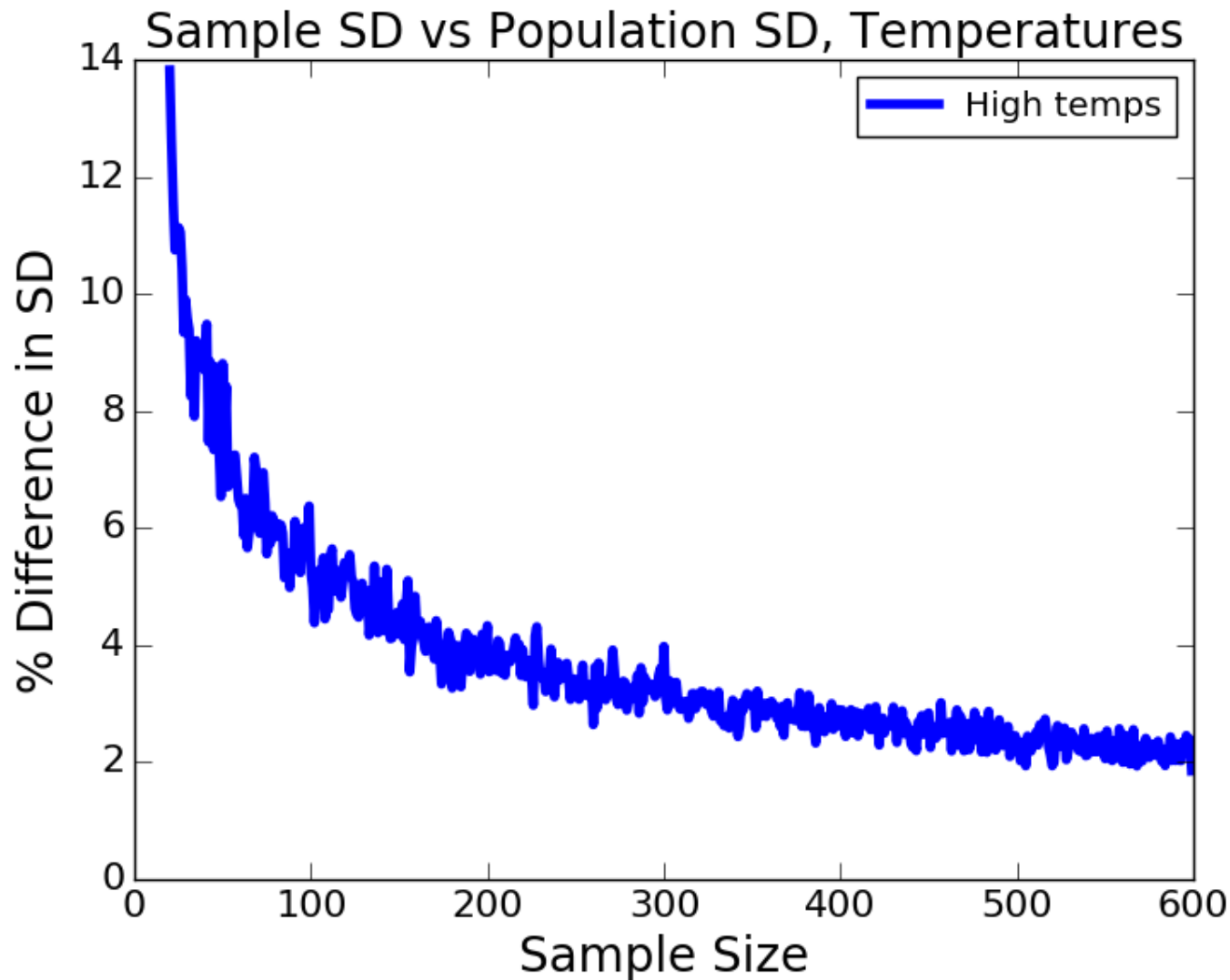
How might we approximate it?

What about just using standard deviation of sample?



# Sample SD vs. Population SD

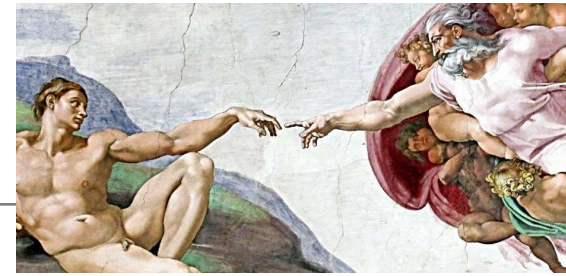
For different sample sizes, compare sample SD to population SD  
Find mean % difference over 100 trials for each sample size



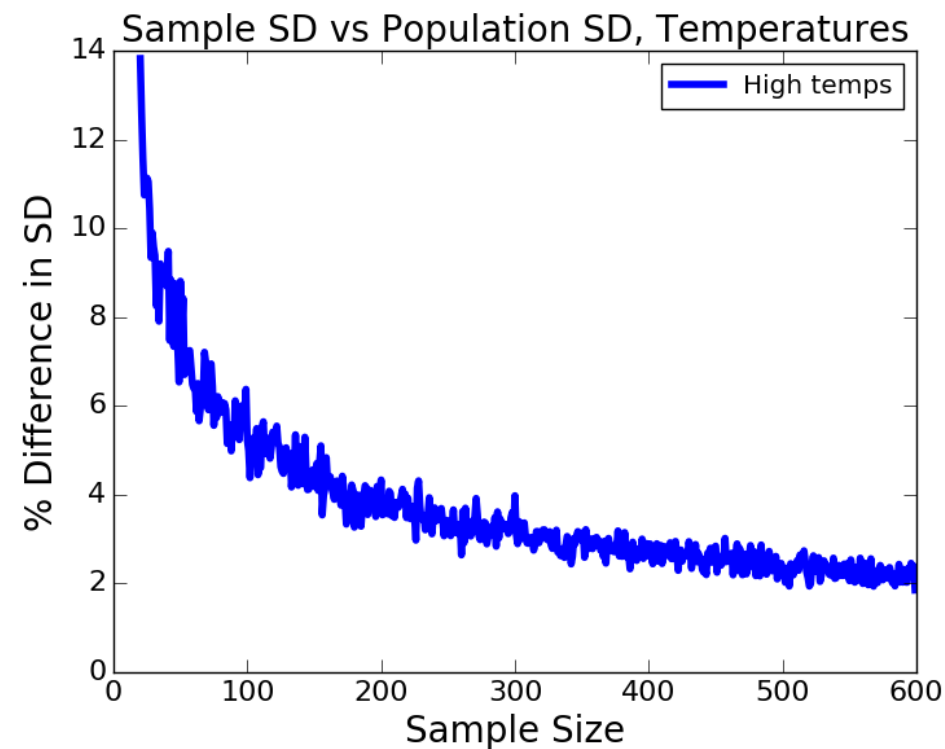
Looks like with large enough sample size, we might be able to just use the sample SD to compute SEM



# 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?
  - Does this depend on distribution of population?
  - Does this depend on size of population?

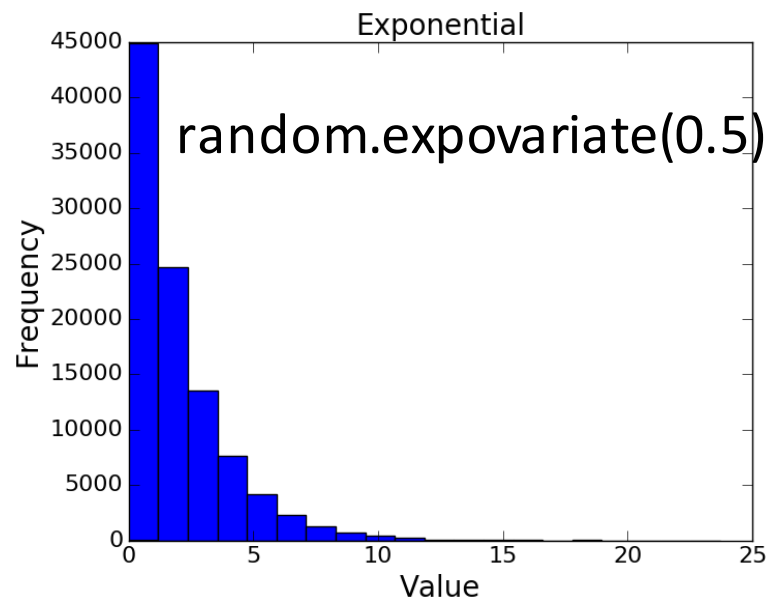
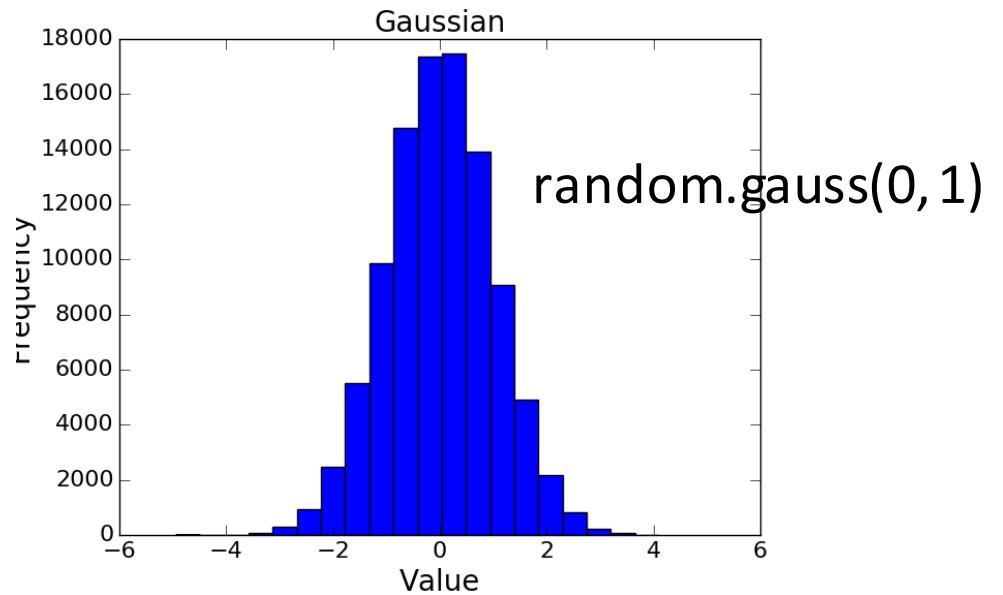
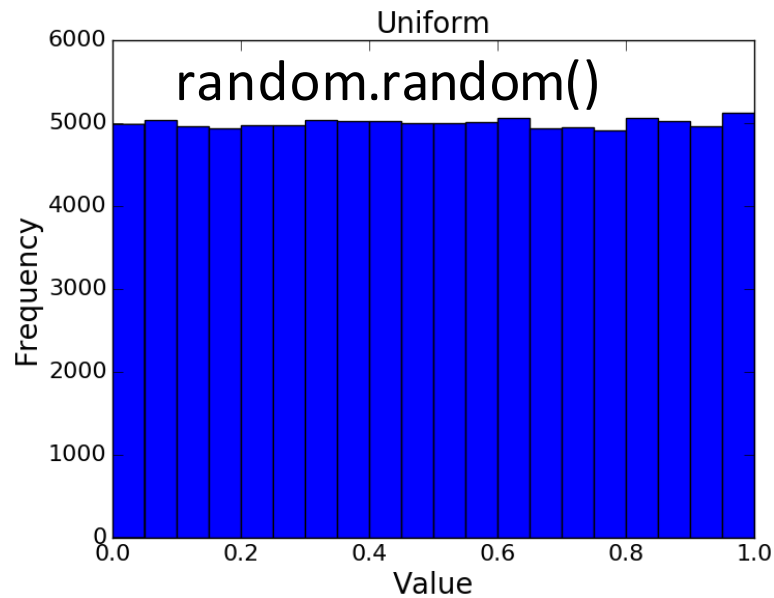


# Looking at Distributions

---

```
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')
```

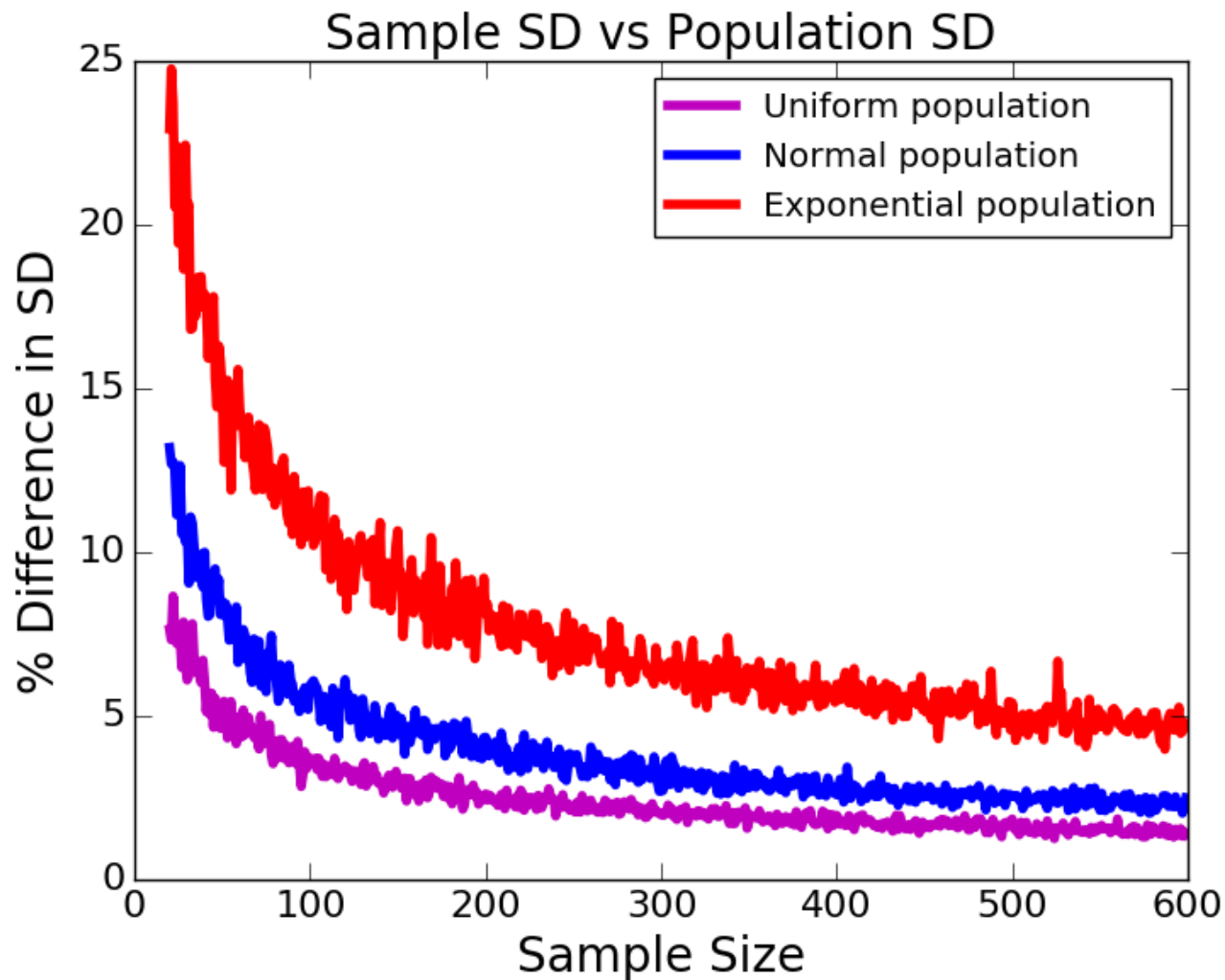
# Three Different Distributions



# Does Distribution Matter?

For different sample sizes, compare sample SD to population SD

Find mean % difference over 100 trials for each sample size



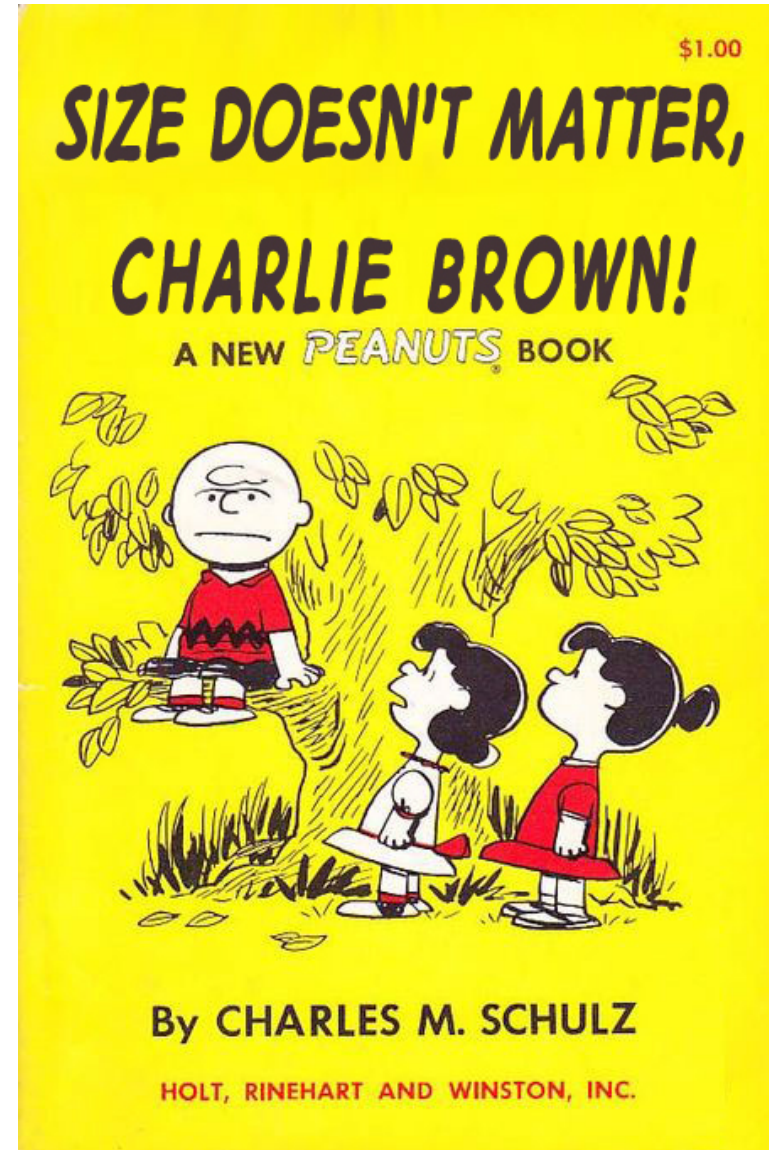
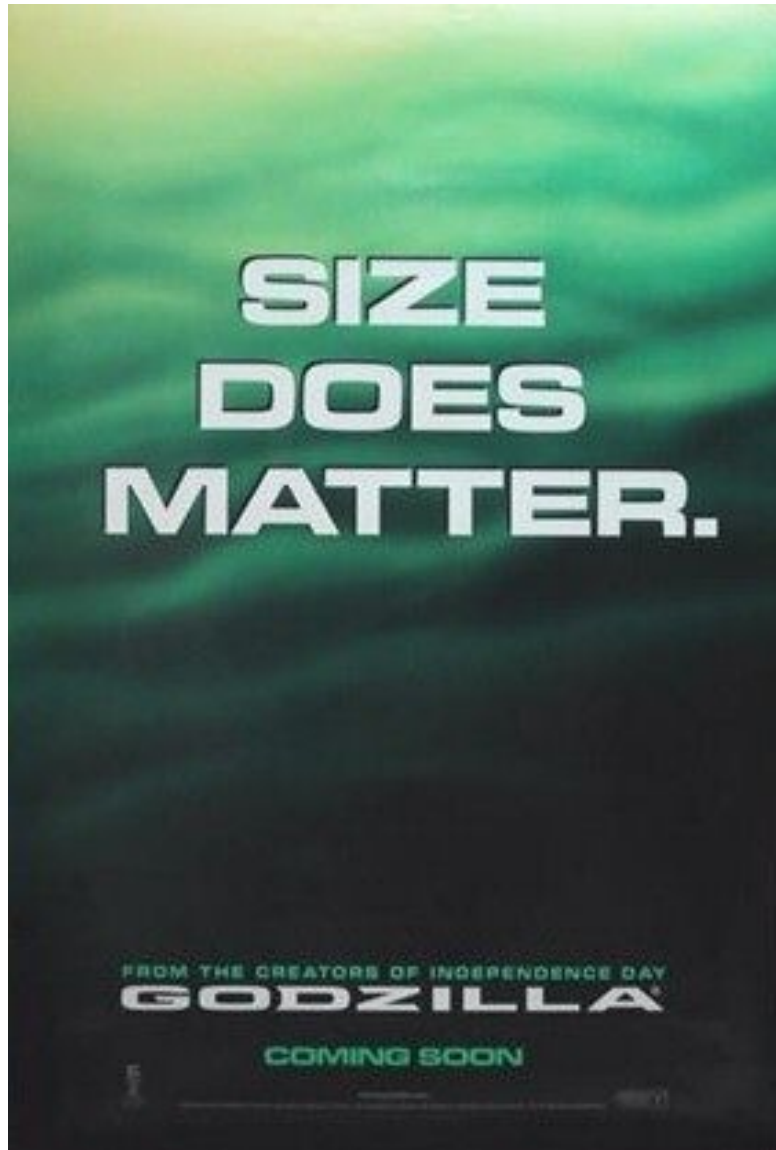
All show same convergence, but not equally good at using sample SD for population SD

**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**

# Does Size Matter?

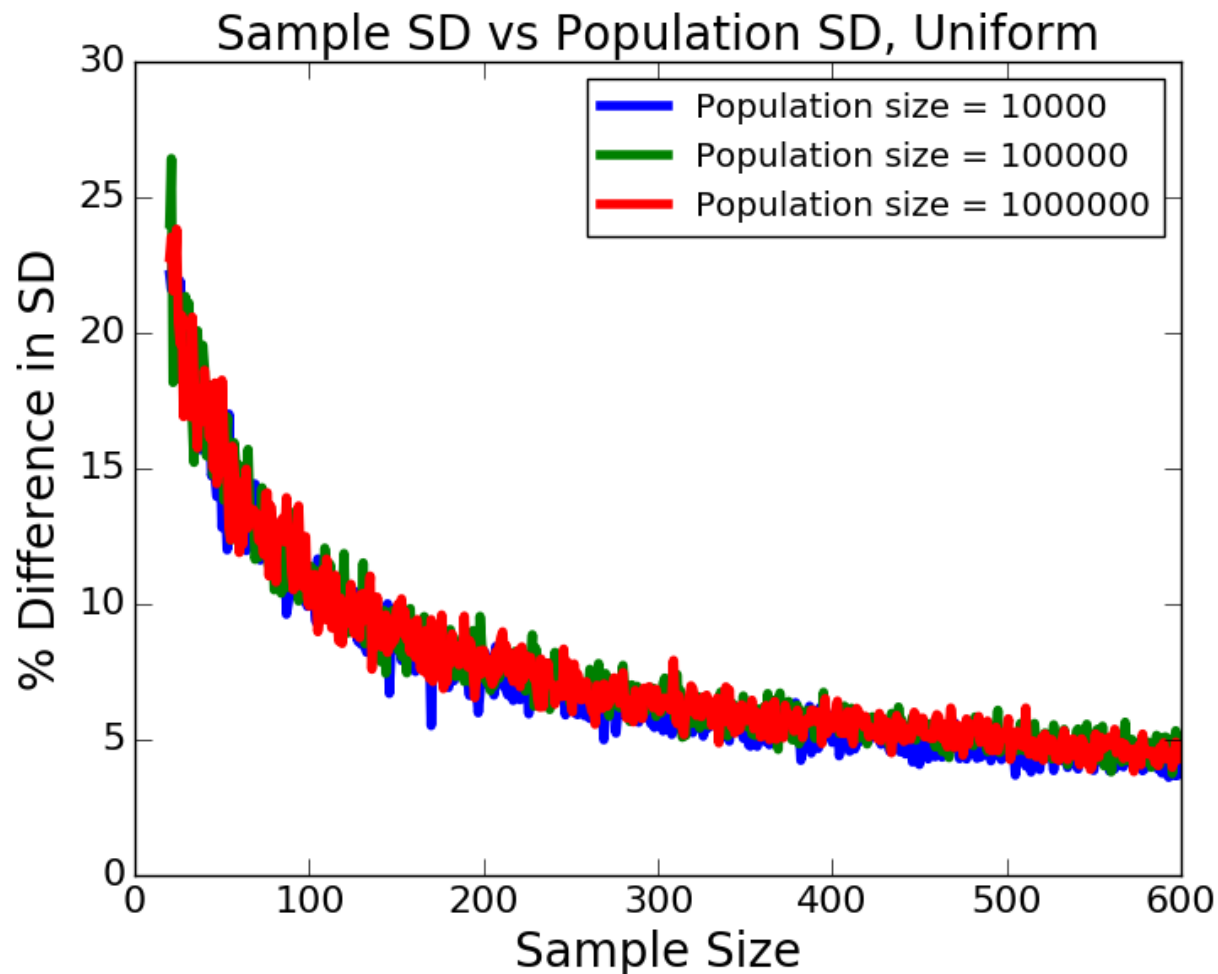
---



# Does Population Size Matter?

For different sample sizes from exponential distribution, compare sample SD to population SD

Find mean % difference over 100 trials for each sample size



# To Estimate Mean from a Single Sample

---

- 1) Choose sample size (based on estimate of skew in population or by some other means)
  - 2) Chose a simple random sample of that size from the population
  - 3) Compute the mean and standard deviation of that sample
  - 4) Use the standard deviation of that sample as estimate of the standard deviation of the population
  - 5) Compute the SE and use Empirical rule to generate confidence intervals around the sample mean for whole population
- 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 Few Lectures

---

- Using Monte Carlo simulation to build models
  - The world is mostly stochastic
    - But 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**

# “Live” Demo of Central Limit Theorem

---

- Galton Board
  - Also known as a quincunx or a bean machine
  - Demonstration that a binomial distribution approximates a normal one

