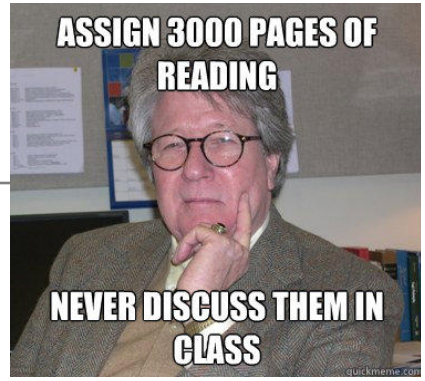
MONTE CARLO SIMULATION: AN EXTENDED EXAMPLE

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

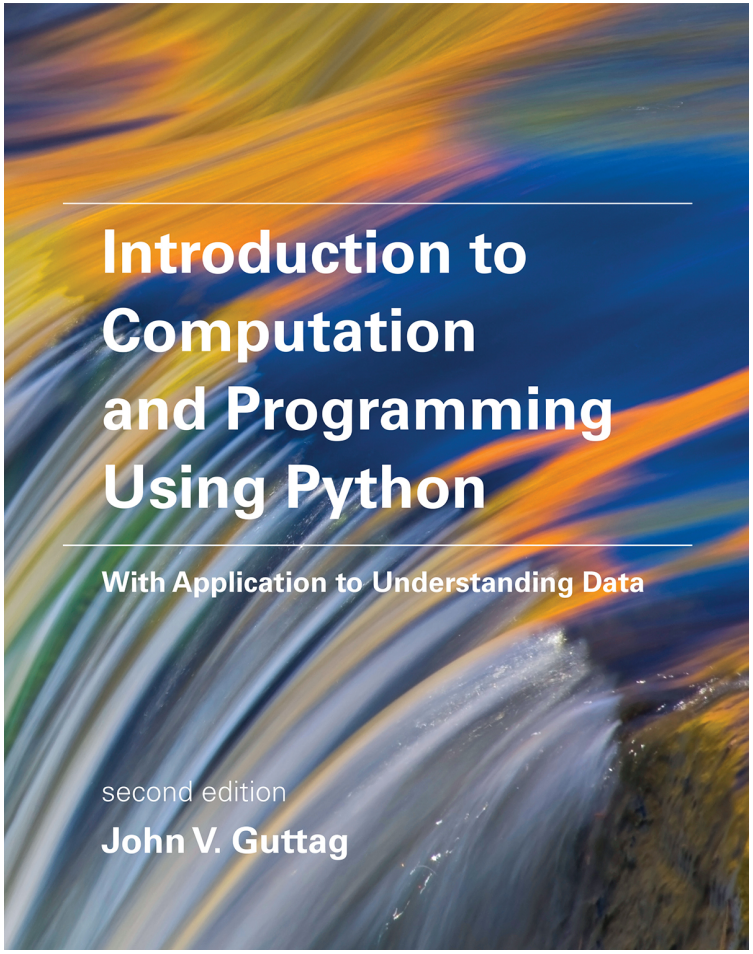
Eric Grimson

MIT Department of Electrical Engineering and Computer Science

Relevant Reading



- Today
  - Sections 15.3, 15.4
- Next lecture
  - Sections 15.3, 15.4
  - Chapter 17



Where are we?

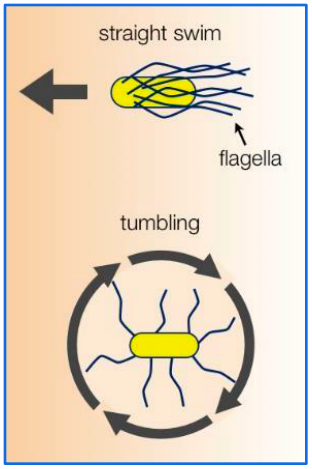
- Have been discussing methods to sample statistics from uncertain or hard to measure data sets
  - Random walks
  - Monte Carlo simulations
  - Random samples (e.g., Buffon-Laplace needle dropping)
- Will provide a quick review of these methods
- Then, will look at an extended example of using Monte Carlo and random walk to explore a problem domain

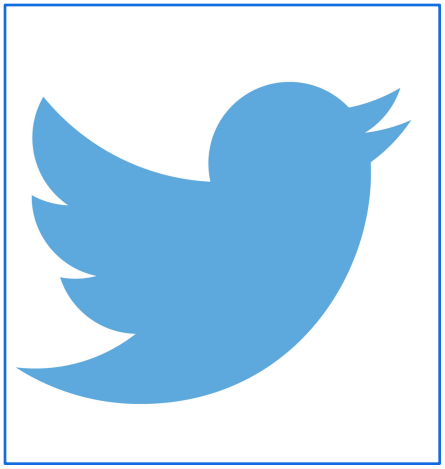
Reminder: Random Walks

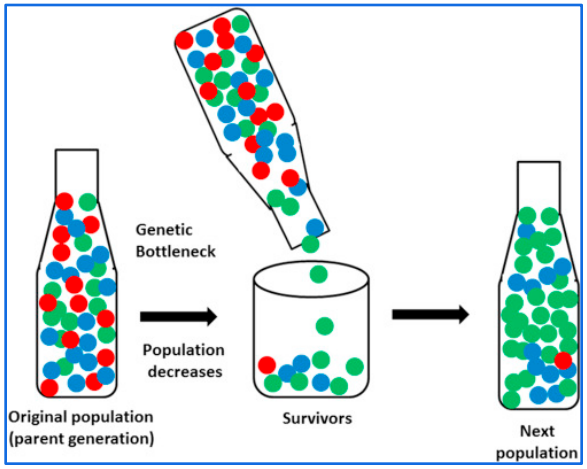
- Model systems where objects move at random according to some distribution of steps:

1. take **N steps at random based on some distribution** and ask how far away are you from the start (or where did you stop)?
2. with many **trials/repetitions**, what is the average distance from the start (or what characterizes the set of end points)?
3. **is there a relation between the number of steps and the average distance from the start?**

- Useful for modeling stock market price changes, Brownian motion of particles, cell movement, who to follow on Twitter, genetic drift of populations







Reminder: Monte Carlo Simulation

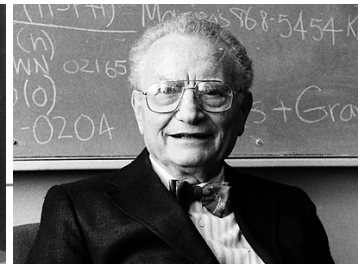
- A method of estimating the value of an unknown quantity using the principles of inferential statistics
- Inferential statistics
  - *Population*: a set of examples
  - *Sample*: a proper subset of a population
  - Key fact: a *random sample* tends to exhibit the same properties as the population from which it is drawn
  - Run multiple trials, with randomly selected samples, and estimate value of property of interest
- Empirical rule and confidence intervals:
  - If the mean estimation error across samples is zero, and
  - The errors in the estimate are normally distributed, then:
    - ~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
  - This lets us make statements about confidence in derived value


MONTY PYTHON'S
AND NOW FOR
SOMETHING
DIFFERENT

Simulating the Stock Market

- Congratulations, you won the lottery!
- Want to invest in the stock market
  - Index fund or stock picking?
    - Index fund – designed to track a large set of stocks, e.g., S&P 500
  - Investigate using a Monte Carlo simulation
- Get a sense of how people actually build simulations
  - Build it a little bit at a time
  - Start with simplifying assumptions
  - Test each piece as we build
  - Compare results to expectations
  - Refine as needed

Underlying Hypothesis

- Simulation is an experimental model of something: in this example we are going to model stock price changes
- We will start with **efficient market hypothesis** (Mandelbrot 1963, Samuelson 1965)
  - It is not possible to consistently outperform the market — appropriately adjusted for risk — by using any information that the market already knows, except through luck
- Information is defined as anything that may affect stock prices and is unknown in the present; thus it appears as a random event in the future. This random information will be the cause of future price changes.

Caveat: what is unknown to some, may not be unknown to others

Corollary

- Because future information cannot be predicted, people buy and sell stock at random
- Therefore appropriate to model stock prices as a random walk
- Let's build a random walk of stock market, and see how it does, using Monte Carlo simulation
- Start with creating classes for Stock and Market

What is most important question to consider?

How to Model Price Changes

- Already decided price changes should be random, but still lots of questions
  - What is the most appropriate way to choose a random change in price of a stock?
  - Should all stocks behave similarly?
- Recall that efficient market hypothesis talked about returns adjusted for "risk".
- What makes a stock "risky"?
  - Over-priced and therefore more likely to go down than up?
- Risk = volatility – stock price could move a large amount either up or down
  - Risk ≈ variance (of a probability distribution on change)

Stock (simStocks0.py)

```
class Stock(object):
    def __init__(self, ticker, volatility):
        self._volatility = volatility
        self._ticker = ticker
    def setPrice(self, price):
        self._price = price
        self._history = [price]
    def makeMove(self):
        if self._price <= 0:
            return 0
        opening = self._price
        baseMove = random.uniform(-self._volatility,
                                  self._volatility)
        self._price = max(0, self._price + baseMove)
        self._history.append(self._price)
        return 100*(self._price - opening)/opening
    def getHistory(self):
        return self._history
```

Measure of range of possible daily change in stock price

Method for simulating daily change in price

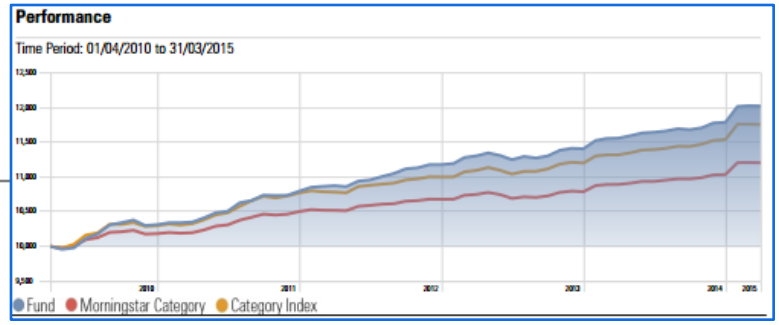
Price can't go negative; once 0, always 0

Change is zero mean; uniform over volatility

Keep track of price changes over year

In this model, price equally like to rise or fall

Annual Return for a Stock




```
def getAnnRet(stk):
    for d in range(TRADINGDAYS):
        stk.makeMove()
    hist = stk.getHistory()
    return 100*((hist[-1] - hist[0])/hist[0])
```

Report change in price over a year, as a percentage

TRADINGDAYS represents number of days market is open

Stock Market


```
class Market(object):
    def __init__(self):
        self._stks = []
    def addStk(self, stk):
        self._stks.append(stk)
    def getStks(self):
        return self._stks.copy()
```

Don't allow mutation of record; return a copy

Testing Market

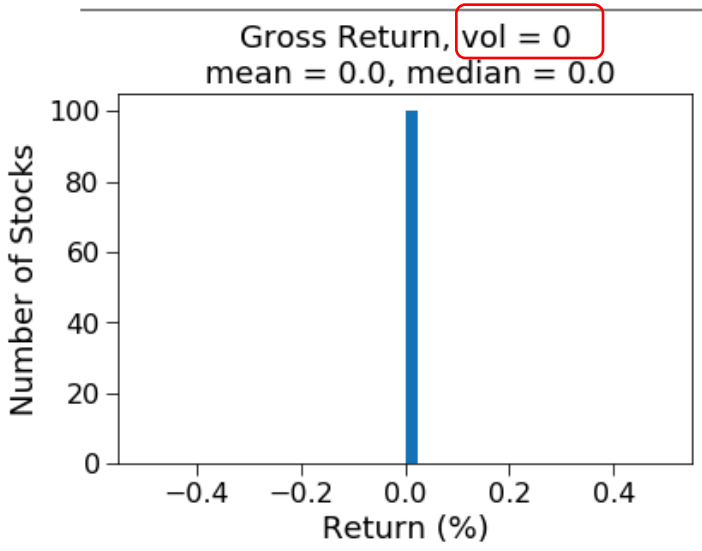
```
def testMkt(numStks, vol):
    mkt = Market()
    for i in range(numStks):
        stk = Stock(str(i), vol)
        stk.setPrice(25)
        mkt.addStk(stk)
    returns = []
    for stk in mkt.getStks():
        returns.append(getAnnRet(stk))
    mean = sum(returns)/len(returns)
    median = sorted(returns)[len(returns)//2]
    plt.hist(returns, bins = 40)
    plt.title('Gross Return, vol = ' + str(vol) +\
              '\n mean = ' + str(round(mean, 2)) +\
              ', median = ' + str(round(median, 2)))
    plt.xlabel('Return (%)')
    plt.ylabel('Number of Stocks')
```

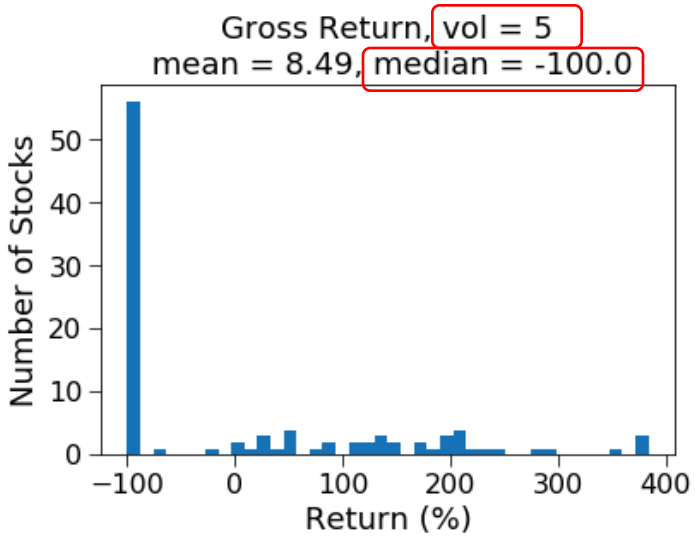
All stocks start out with same price and volatility

Get annual return (as %) for each stock

Compute market statistics over all stocks

Smoke Test





Looks fine

If volatility is uniform about zero, why isn't resulting distribution also uniform about zero?

Why do so many stocks end up with a value of 0?

Not realistic? The majority of stocks to not go to zero in year!

Something wrong with our model, not our code

Modeling Price Changes

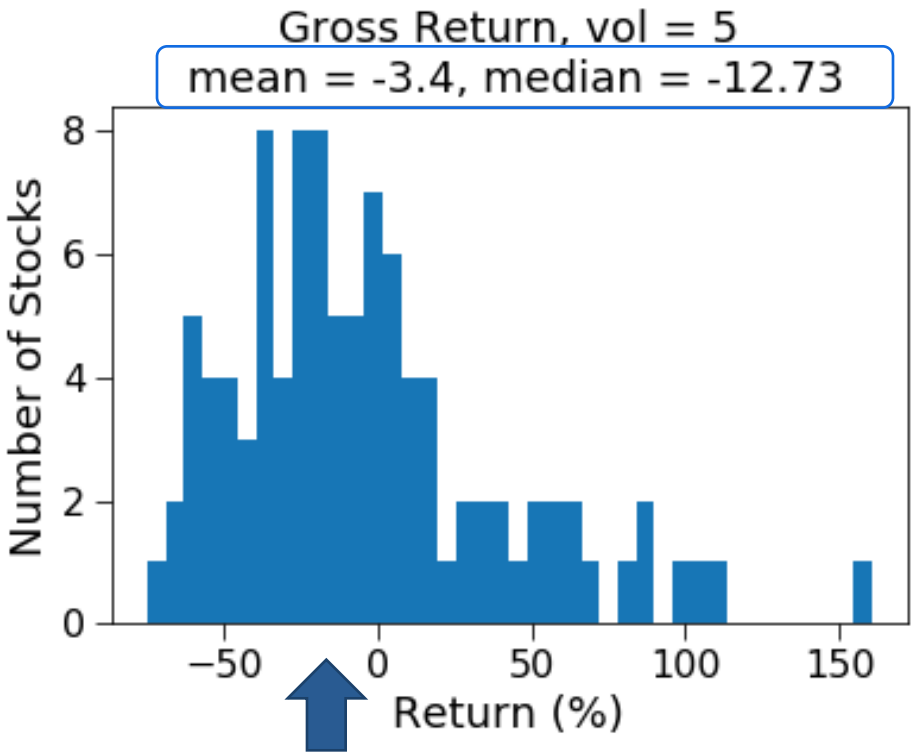
- Example of using simple simulation to start, then checking against expected behavior and adjusting simulation model
- Should magnitude of change be independent of price?
- Change volatility to be percentage change
- See file simStocks1.py

```
def makeMove(self):
    if self._price <= 0:
        return 0
    baseMove = random.uniform(-self._volatility,
                              self._volatility)/100
    self._price = max(0, self._price*(1 + baseMove))
```

baseMove is now a percentage; still uniform over a range

Multiply percentage change by price and add as actual change

Result of Change



Try a few times with different seeds

While actual values change, results are consistent

Seems to be a negative bias, why? P * .9 * 1.1 = .99P

This doesn't match observations

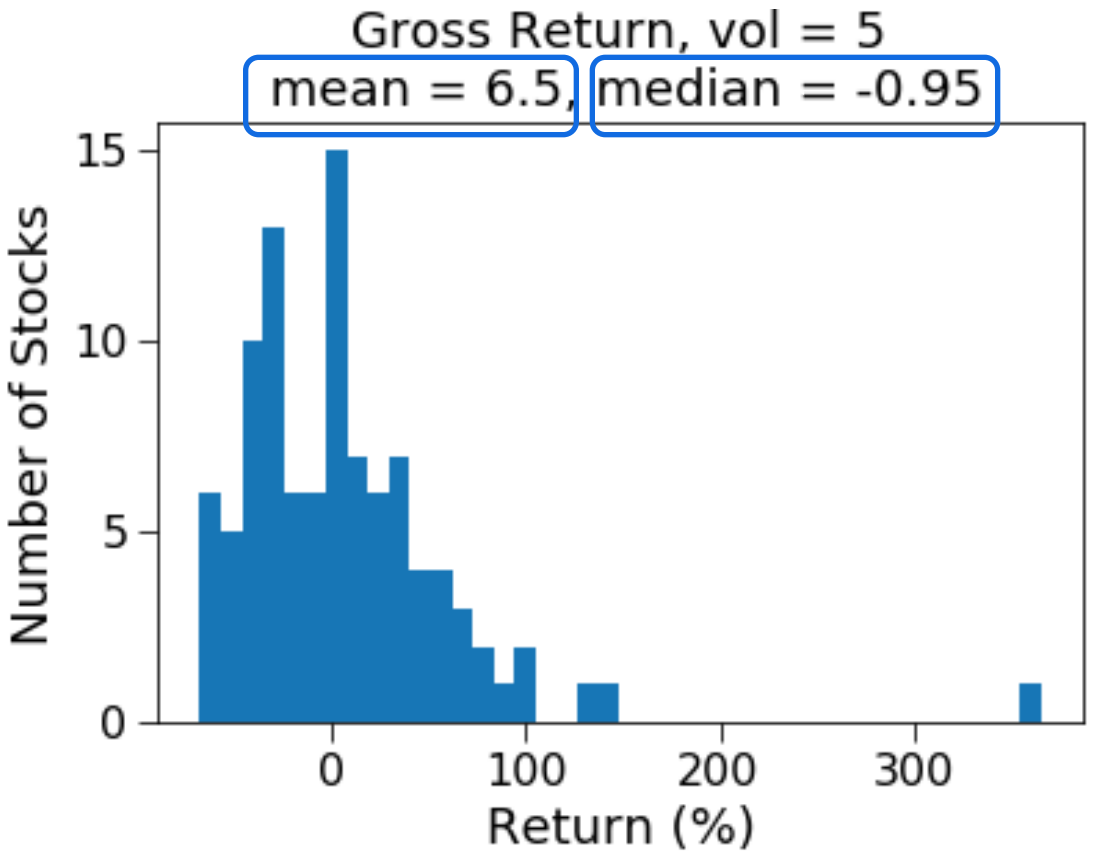
What Does Actual Market Do?

- Goes up about 6%/year (inflation adjusted)
  - Well, maybe not recently, but in general
- Let's try a **biased** random walk (simStocks2.py)

```
def makeMove(self):
    if self._price <= 0:
        return 0
    opening = self._price
    baseMove = 0.06/TRADINGDAYS +\
               random.uniform(-self._volatility,
                              self._volatility)/100
    self._price = max(0, self._price*(1 + baseMove))
    self._history.append(self._price)
    return 100*(self._price - opening)/opening
```

Base move now has uniform percentage volatility, but added to constant upward bias

Result of Change



Is assumption of uniform distribution of daily variation realistic?
What does the market do?

Mean seems plausible

What about median?

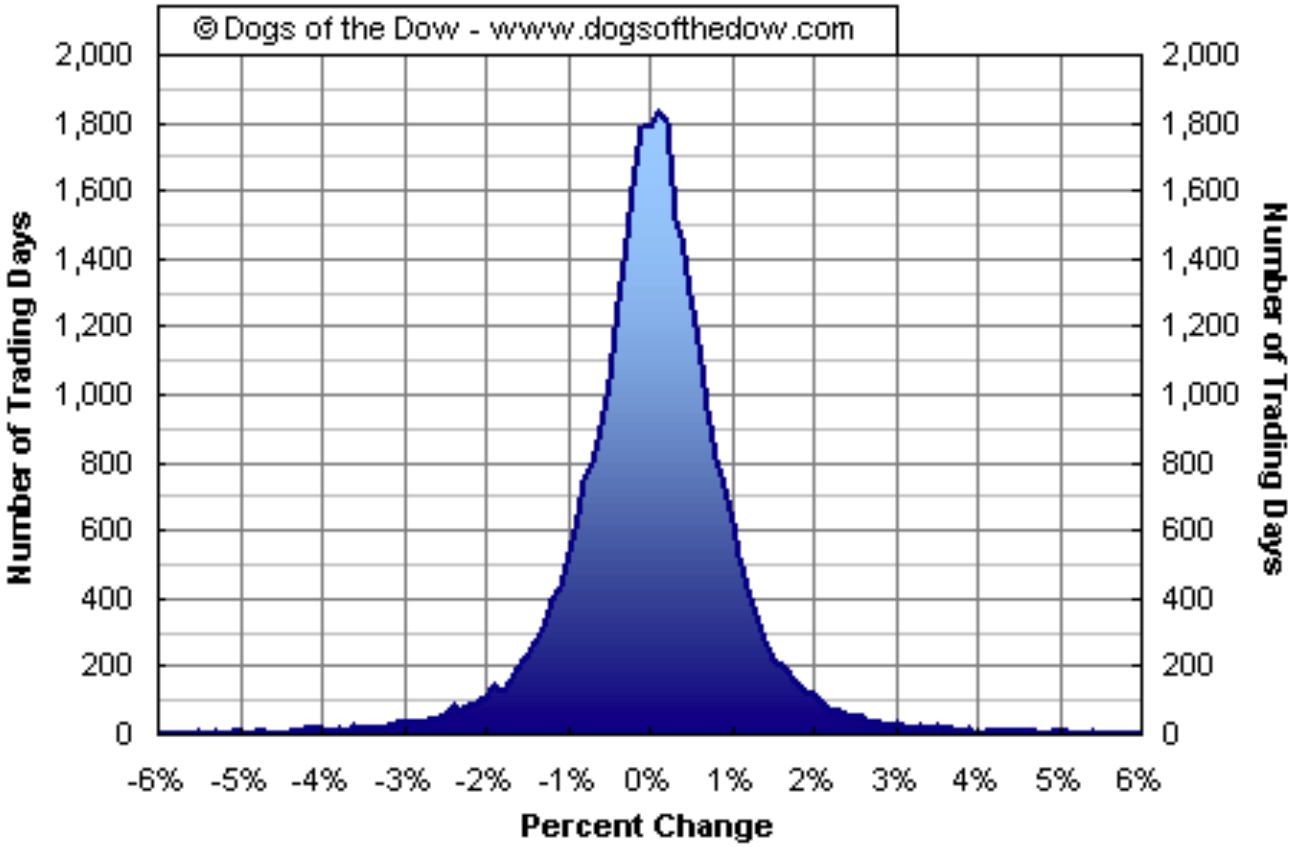
Too small

- Artifact of uniform distribution of daily changes
- Once near zero, hard to go up

What about tails?

Should be fatter; more cases of stocks that succeed or fail significantly

Are Price Changes Uniformly Distributed?



Daily change in stock market index as percentage

Accumulated across many years

This appears to be a normal or Gaussian distribution

Mean = 0.026%, std = 1.07%

A Better Model of Price Changes

```
class Stock(object):
    def __init__(self, ticker, bias, volatility):
        self._bias = bias/100
        self._vol = volatility/100
        self._ticker = ticker
    def makeMove(self):
        opening = self._price
        if self._price <= 0:
            return 0
        baseMove = self._price*(random.gauss(self._bias,
                                             self._vol))
        self._price = max(0, self._price + baseMove)
        self._history.append(self._price)
        return 100*((self._price - opening)/opening)
```

Now a stock has its own bias

Assume inputs are percentages; convert to decimal fraction

Sample from Gaussian; scaled by price since percent change

Add change to previous price

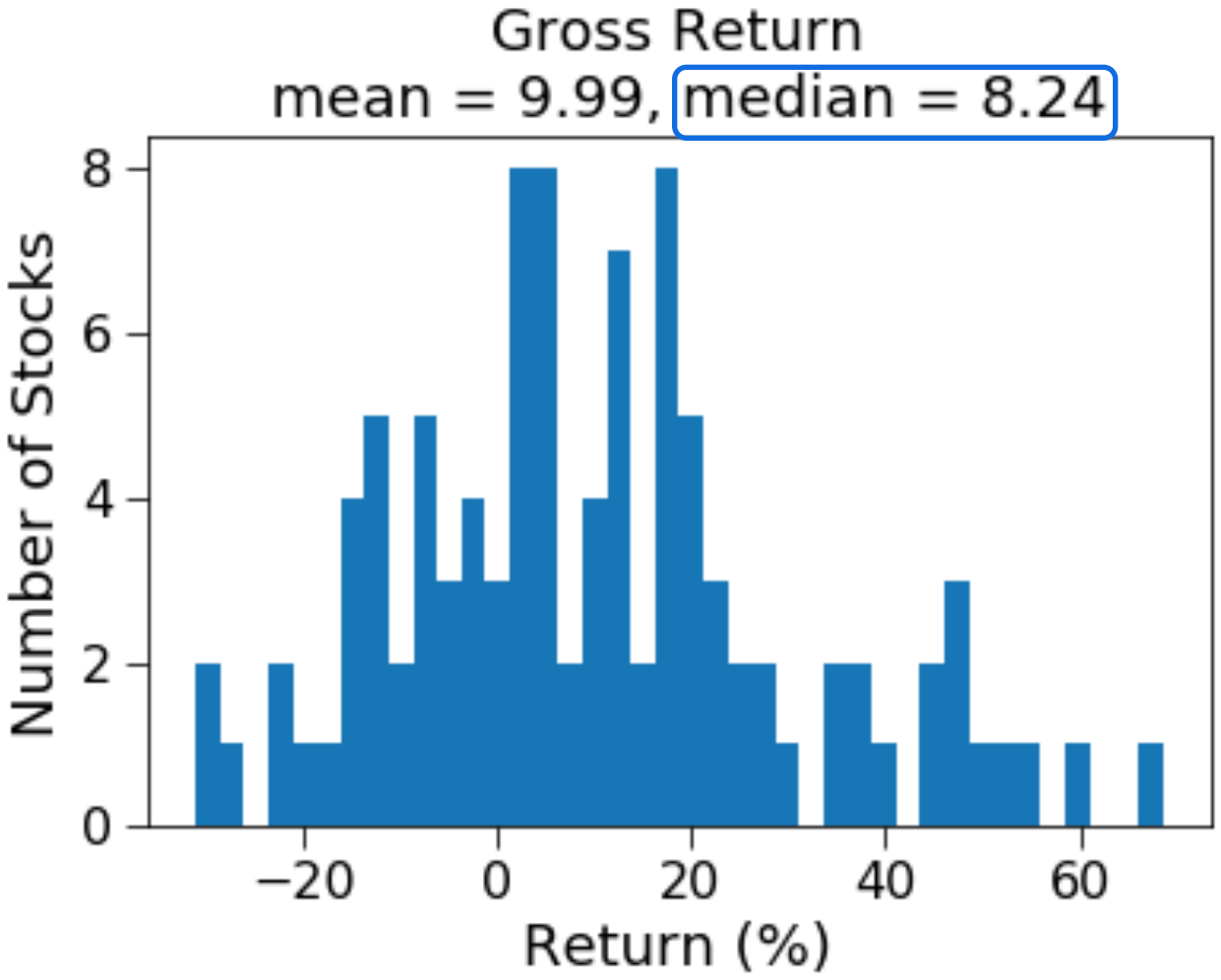
simStocks3.py

And Restructured Testing Code

```
def testMkt(mkt, toPlot = True):
    returns = []
    for stk in mkt.getStks():
        returns.append(getAnnRet(stk))
    mean = sum(returns)/len(returns)
    median = sorted(returns)[len(returns)//2]
    if toPlot:
        plt.hist(returns, bins = 40)
        plt.title('Gross Return' +\
                  '\n mean = ' + str(round(mean, 2)) +\
                  ', median = ' + str(round(median, 2)))
        plt.xlabel('Return (%)')
        plt.ylabel('Number of Stocks')
    return mean, median

def makeMkt(numStks):
    mkt = Market()
    for i in range(numStks):
        stk = Stock(str(i), 0.026, 1.07)
        stk.setPrice(25)
        mkt.addStk(stk)
    return mkt
```

Actual values come from historical data (previous slide)

Volatility(0.026, 1.07)



Note that median is much more realistic

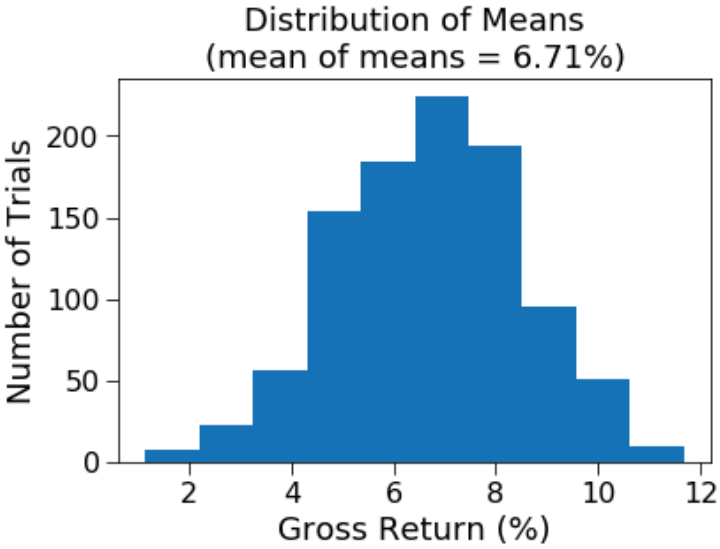
Note that tails are much fatter

But this is only one Trial

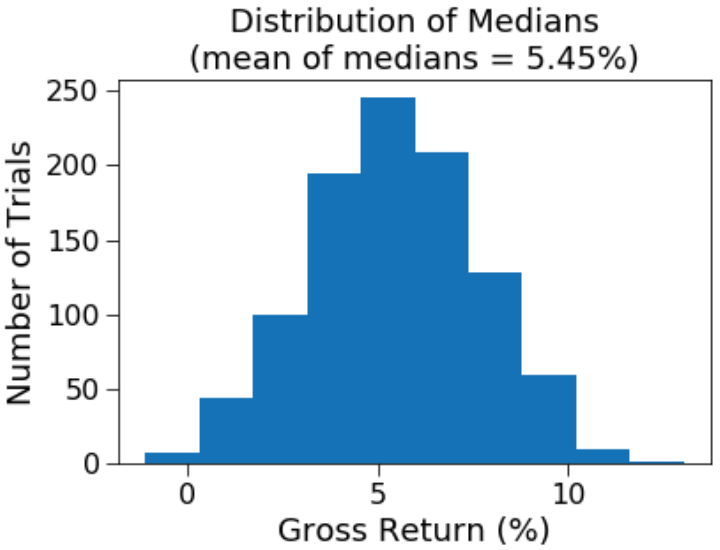
Simulate Many Different Versions of Year

- Each trial records mean and median change in price of a set of stocks over a year
- Because there is random variation, simulate this yearly change many times
  - Record mean and median change for each yearly trial
  - What are the distributions of mean price change and median price change across many trials?
    - Shape of the distributions
    - Mean over many trials of the mean annual price change
    - Mean over many trials of the median annual price change

Try 1000 Trials



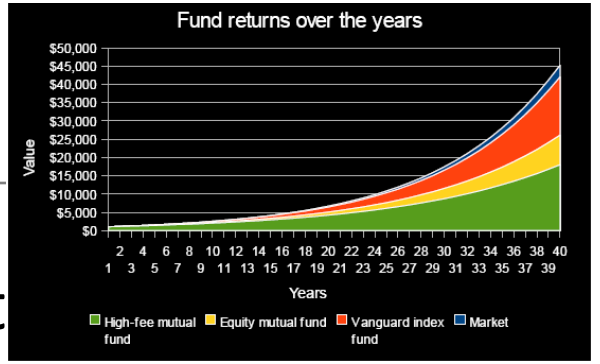
Distribution of mean annual return across trials is close to Gaussian (we'll see why next week)



Medians also roughly Gaussian, but **smaller** than mean

If we believe simulation, what should we conclude?

Implications of Simulation



- For this simulation, buying whole market (or index fund), gives return of ~6.71%
- Buying one stock at random is likely to under-perform buying an index fund (average median return is 5.45%)
  - Because mean of median is smaller than mean of means, typically more than half the individual stocks will not perform as well as index fund
  - So unless we are lucky (or skilled?) in picking stocks we are not likely to do as well as index fund
- But perhaps this is a function of all stocks in our simulated market having identical properties
  - Do we expect Amazon and GE to behave similarly?

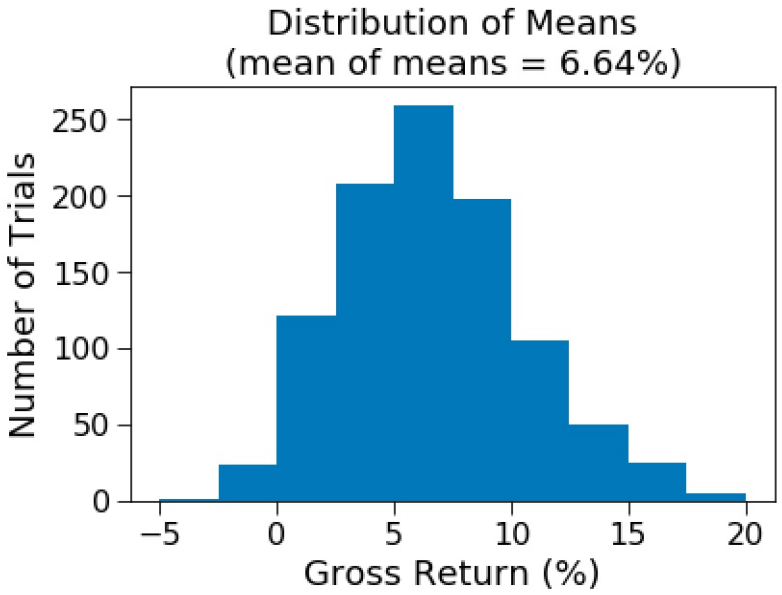
Stocks Have Different Risks (volatility)

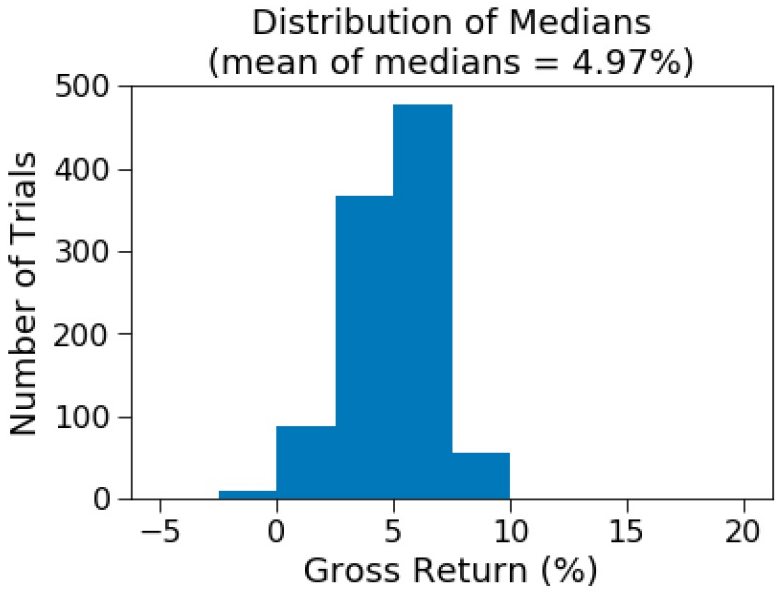
```
def makeMkt(numStks):
    mkt = Market()
    for i in range(numStks):
        vol = random.gauss(0, 2)
        stk = Stock(str(i), 0.026, vol)
        stk.setPrice(25)
        mkt.addStk(stk)
    return mkt
```

Let each stock have different volatility, selected randomly from normal distribution

simStocks4

Try It Out



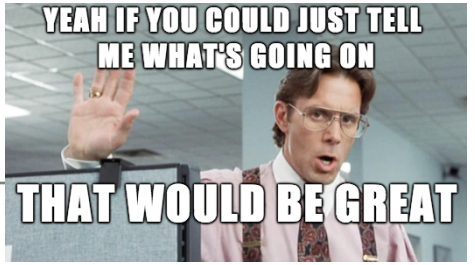


Variance is small

Most years are average years (remember there is an underlying bias of ~6%)

And almost ALL years are good years!

What's Going On



- Model assumes stock movements are independent of each other
- Not true in practice
  - Market has quiet days when market barely changes
  - Market has good days, when most stocks go up
  - Market has bad days, when most stocks go down
  - Let's add a daily bias term
- Distribution of daily bias **not** normal
  - A few really bad and a few really good days
  - Most days fair to middling

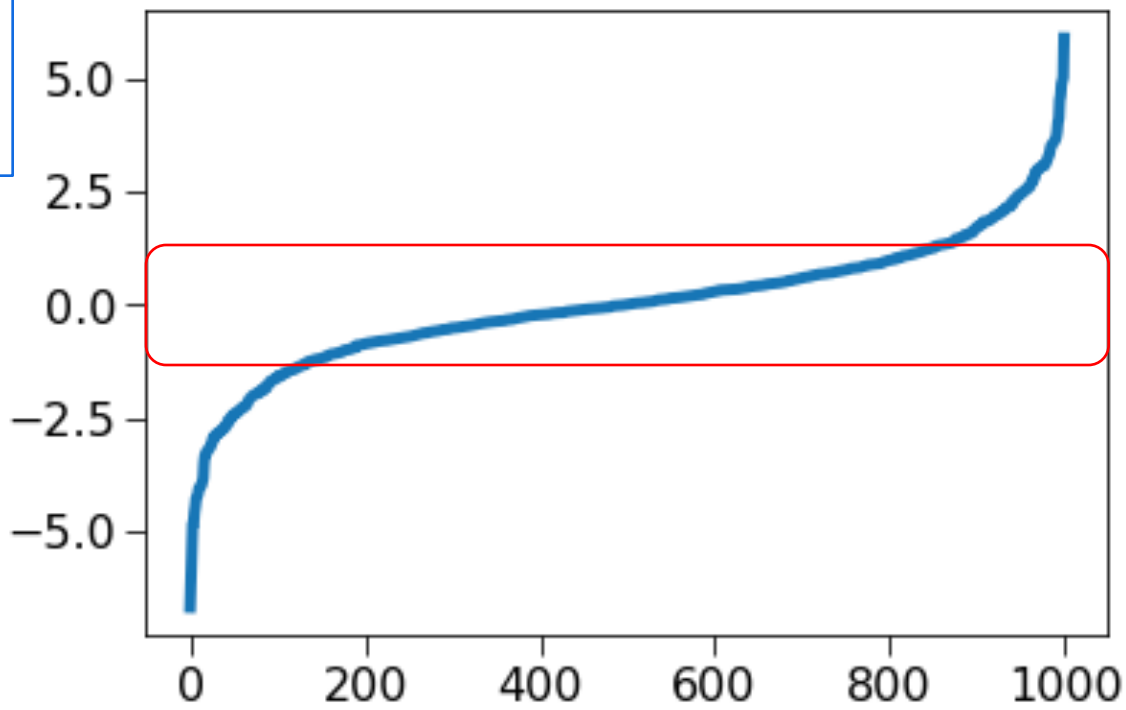
Use An Exponential Distribution

```
vals = []
for i in range(500):
    vals.append(random.expovariate(1))
    vals.append(-random.expovariate(1))
plt.figure()
plt.plot(sorted(vals))
```

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

Probability density function (more about that next week); lambda is a parameter to choose

Note that the vast majority of values are very close to 0; Only a small set of values are outliers; some with big values



Change makeMove

```
def makeMove(self, dayBias):
    opening = self._price
    if self._price <= 0:
        return 0
    baseMove = self._price*(random.gauss(self._bias +\
                                         dayBias,
                                         self._vol))
    self._price = max(0, self._price + baseMove)
    self._history.append(self._price)
    return 100*((self._price - opening)/opening)
```

In addition to a general bias in price change distributed across the year, add a second bias that varies by day and reflects overall market bias

simStocks5.py

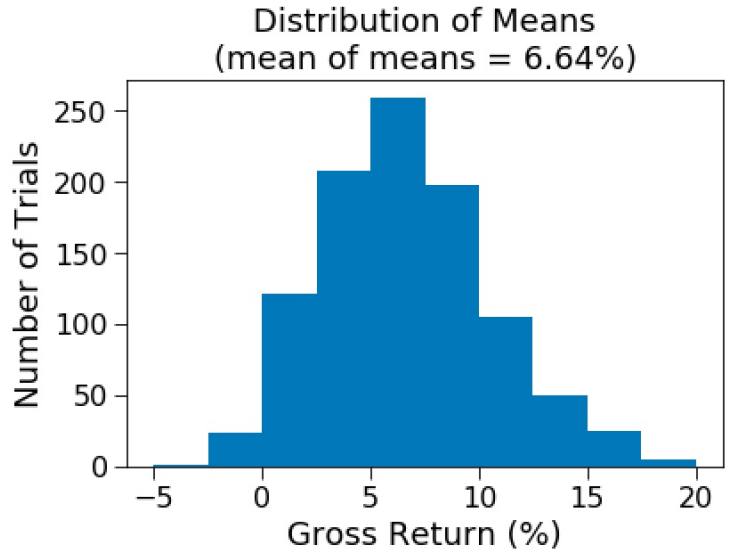
Change Testing Code

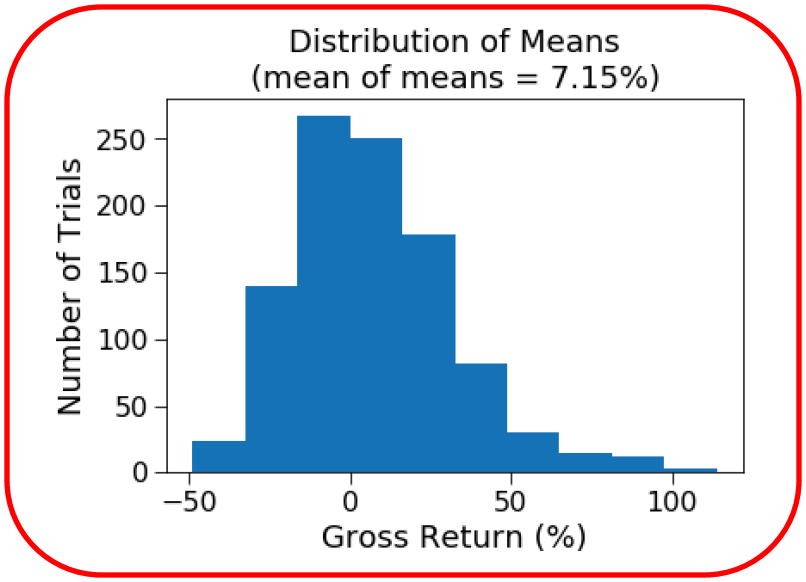
```
def getAnnRet(stk, dailyBiases):
    for d in range(TRADINGDAYS):
        stk.makeMove(dailyBiases[d])
    hist = stk.getHistory()
    return 100*((hist[-1] - hist[0])/hist[0])

def testMkt(mkt, toPlot = True):
    dayBiasLambda = 1
    dailyBiases = []
    for d in range(TRADINGDAYS):
        dayBias = random.expovariate(dayBiasLambda)/100
        if random.random() < 0.5:
            dayBias = -dayBias
        dailyBiases.append(dayBias)
    returns = []
    for stk in mkt.getStks():
        returns.append(getAnnRet(stk, dailyBiases))
```

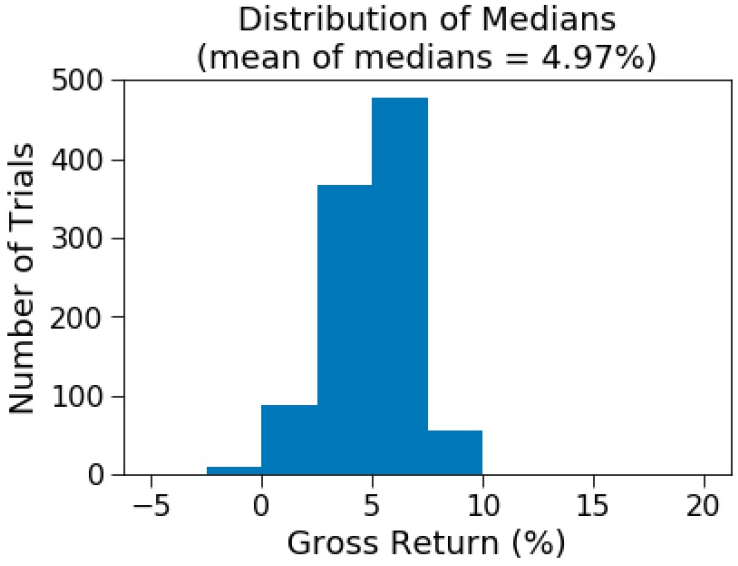
Use list of biases to add a daily variation

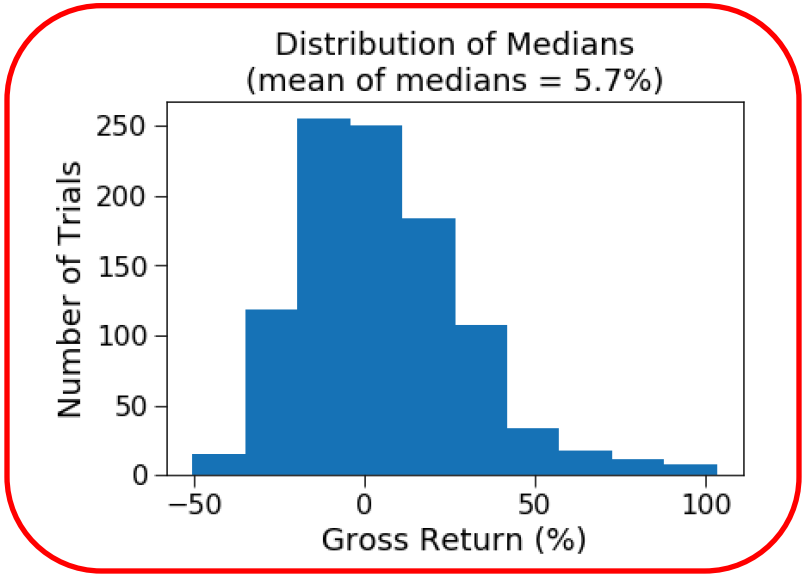
Collect a list of daily biases, sampled from an exponential distribution

Without and With Daily Bias (lambda = 1)



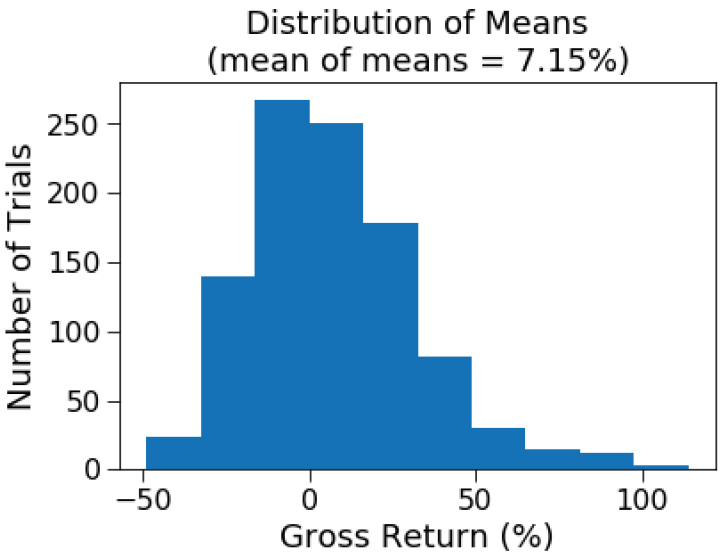


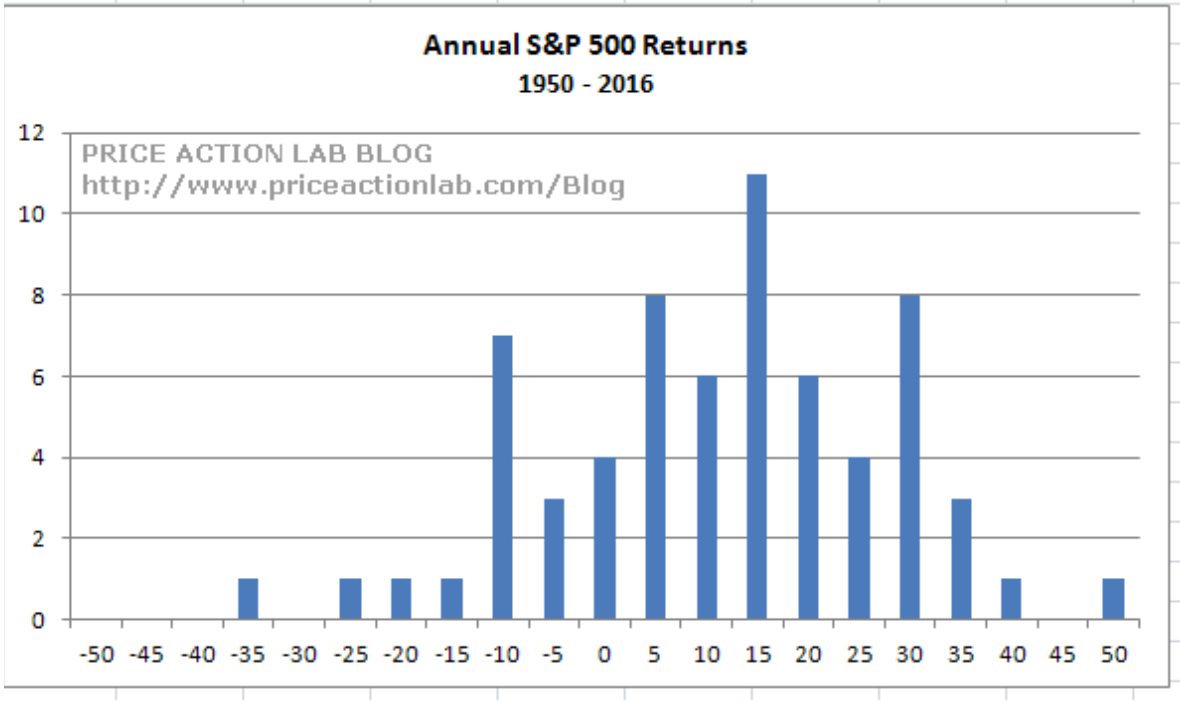


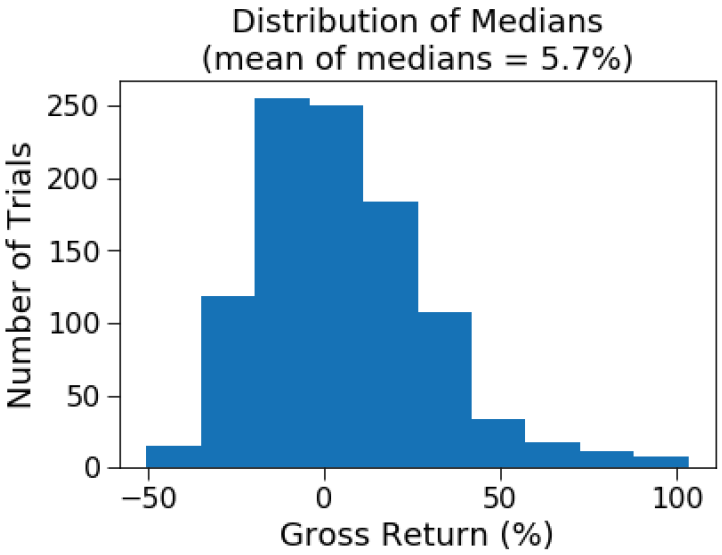


Note how the tails of the new distributions are much longer; and that there are many more years when average return is negative

1000 Trials, lambda = 1







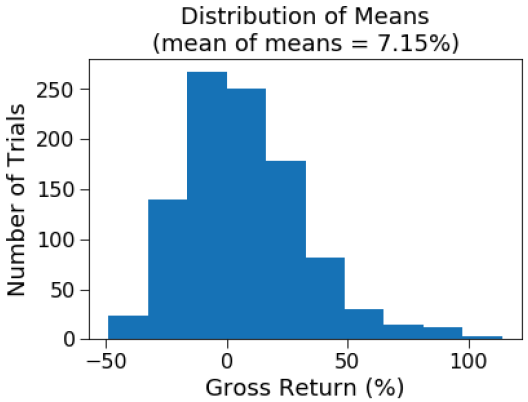
Actual data seems similar, including general shape of distribution, and instances of long tails

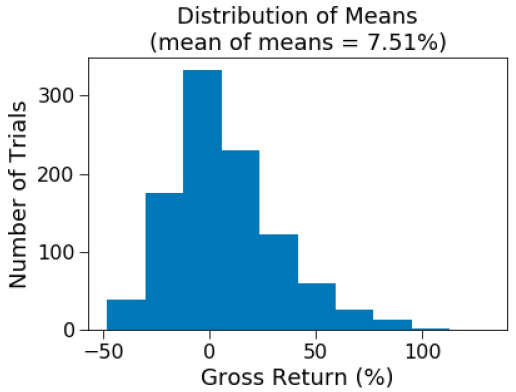
Once again, median smaller than mean

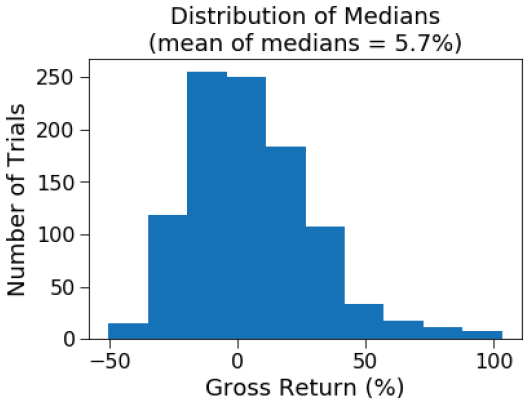
Are There Other Variations to Try?

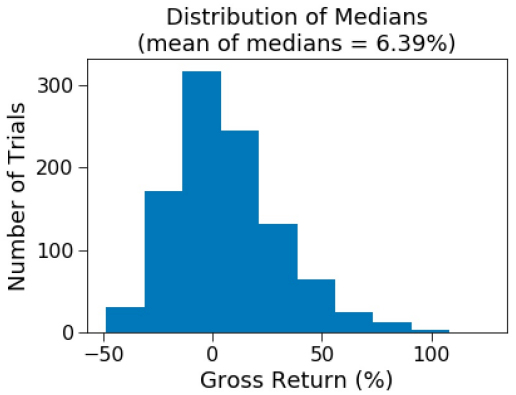
simStocks7.py

- We start all stocks at the same price
  - Easy to change code to set starting prices at random between 10 and 100









Fixed starting price

Variable starting price

Doesn't seem to be any significant change

If Stock Picking too Hard, What About Market Timing?

```
def getDailyReturn(mkt):
    days = np.array([0]*(TRADINGDAYS))
    for stk in mkt.getStks():
        prices = stk.getHistory()
        returns = [100*(prices[i]-prices[i-1])/prices[i-1]\
                   for i in range(1, len(prices))]
        days = days + np.array(returns)
    days = days/len(mkt.getStks())
    return days
```

returns is a list of percentage change by day for one stock

days starts as an array of 0's, one for each day

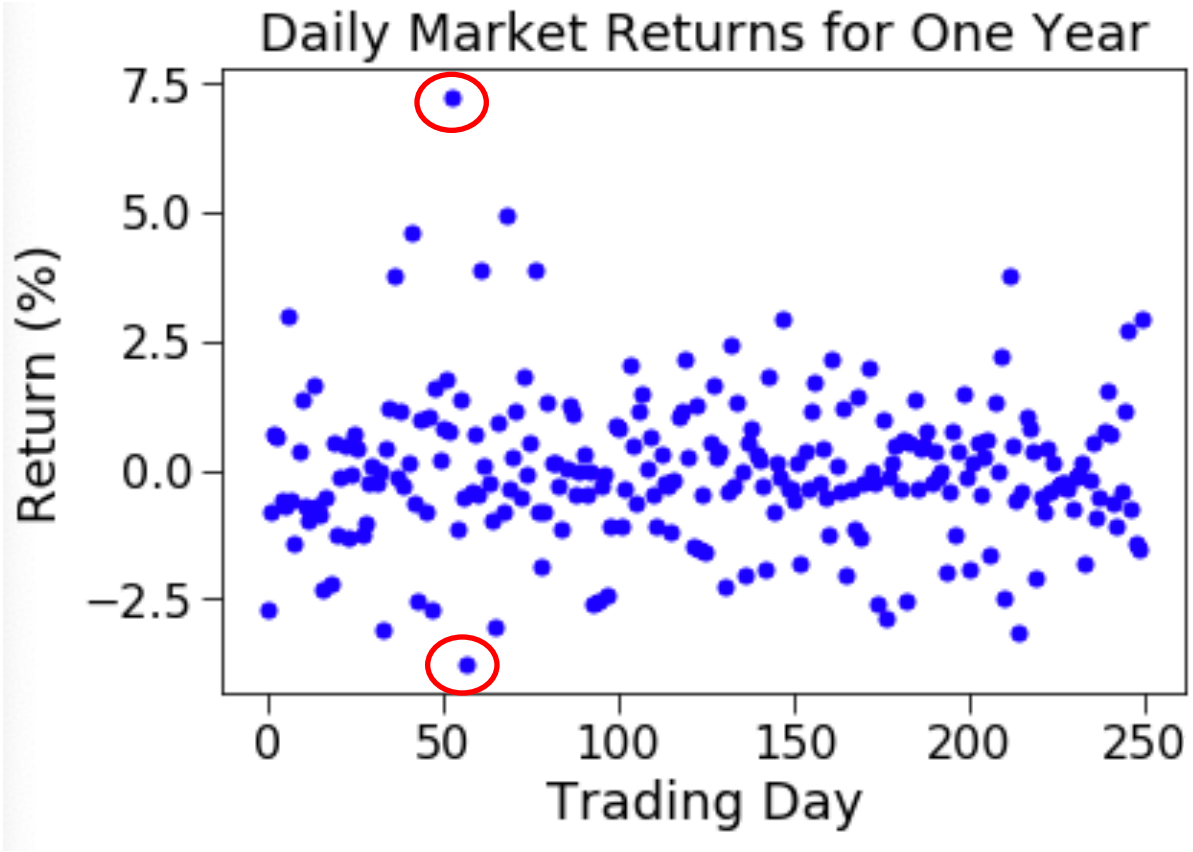
on each iteration, percent change of each stock is added for each day

finally get average change in price over all stocks for each day

An example of a "list comprehension"

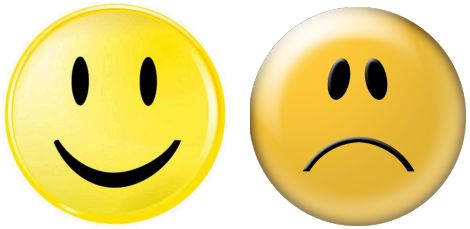
simStocks6.py

One Randomly Chosen Year



Sell Buy

Buy Sell

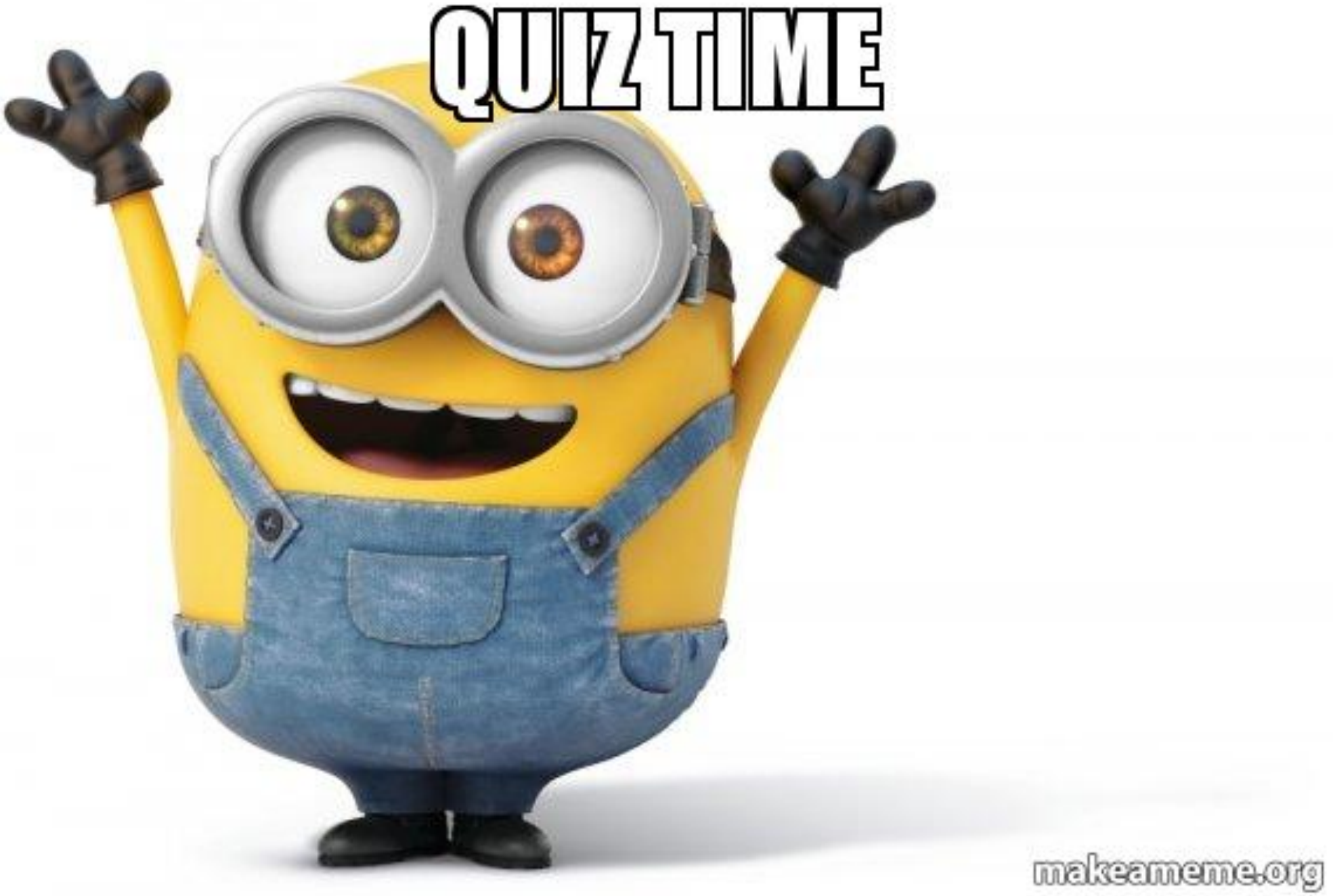


As expected: Most days quiet;
a few big swings, but…

… timing the market is risky

What to Take Away

- You know enough to start building interesting simulations
- Build them incrementally
  - Start with a question worth answering
  - Start with something simple
  - Build tools to inspect results (e.g., plots)
  - Think about how results compare to reality (smoke test)
  - Add complexity as needed to refine question
- Use Monte Carlo, random walk, inferential statistics to model systems and understand results


QUIZ TIME
makeameme.org