

Lecture: Monte Carlo Simulation, an Extended Example

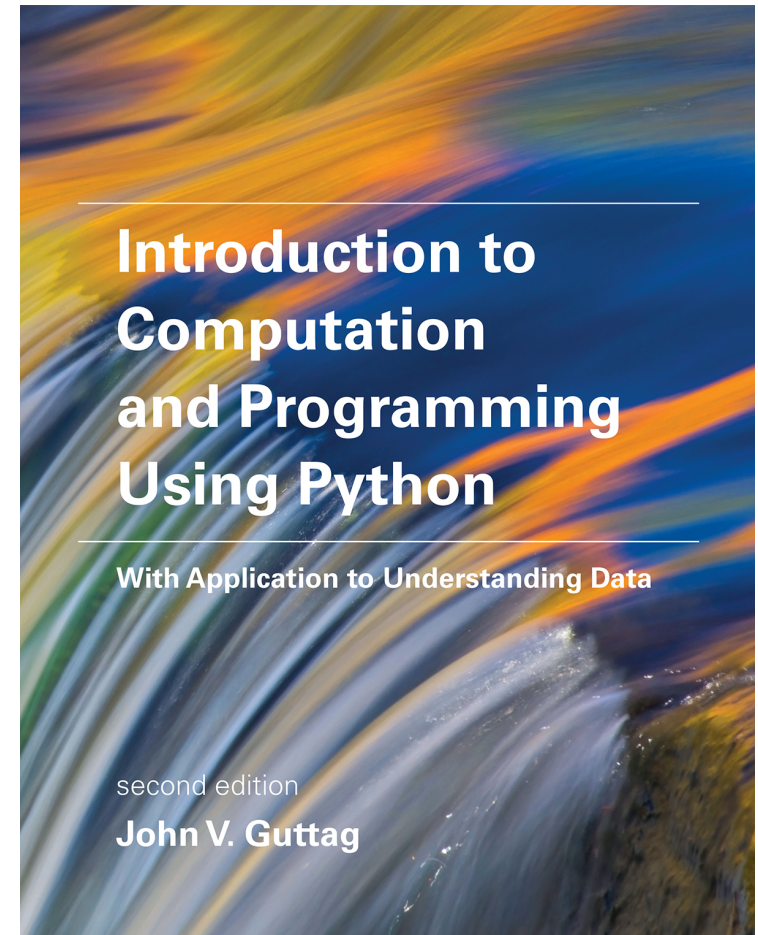
(download slides and .py files from Stellar to follow along)

Eric Grimson

MIT Department of Electrical Engineering and
Computer Science

Relevant Reading

- Today
 - None
- Next lecture
 - Sections 15.3, 15.4
 - Chapter 17

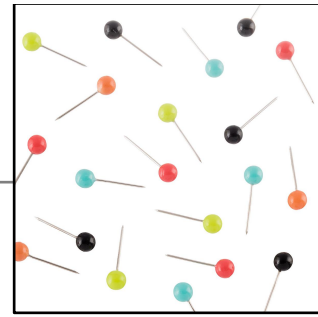


Where are we?

- Have been discussing methods to sample statistics from uncertain or hard to measure data sets
 - Random walks
 - Monte Carlo simulations
 - Buffon-Laplace needle dropping
- Provide a quick recap of these methods
- Then, going to look at an extended example of using Monte Carlo to explore a domain



A Useful Technique (recap)



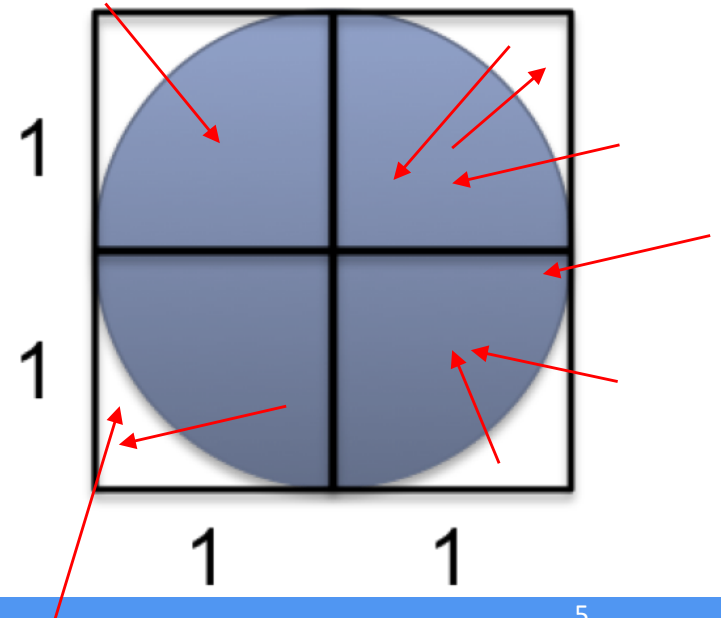
- To estimate the area of some region, R
 - Pick an enclosing region, E, such that the area of E is easy to calculate and R lies completely within E
 - Pick a set of random points that lie within E (drop pins)
 - Let F be the fraction of the points that fall within R
 - Multiply the area of E by F
- Way to estimate integrals

$$\int_0^{\pi} \sin(x) dx$$

Simulating Buffon-Laplace Method (recap)

```
def throwNeedles(numNeedles):  
    inCircle = 0  
    for needle in range(1, numNeedles + 1, 1):  
        x = random.random()  
        y = random.random()  
        if (x*x + y*y)**0.5 <= 1.0:  
            inCircle += 1  
        if needle%10000000 == 0:  
            print('Dropped another 10 million needles')  
    return 4*(inCircle/numNeedles)
```

Since area of circle of unit radius is just π , this give us a way to estimate that constant



Simulating Buffon-Laplace Method (recap)

```
import numpy as np

def getEst(numNeedles, numTrials, printLevel = 0):
    estimates = []
    for t in range(numTrials):
        piGuess = throwNeedles(numNeedles)
        estimates.append(piGuess)
        if printLevel > 1:
            print('Finished trial', t, 'Est. =', piGuess)
    sDev = np.std(estimates)
    curEst = sum(estimates)/len(estimates)
    if printLevel > 0:
        print("{:13s} {:13s} {}".format(str(round(curEst, 10)),
                                          str(round(sDev, 10)), numNeedles))
    return (curEst, sDev)
```

Simulating Buffon-Laplace Method (recap)

```
def estPi(precision, numTrials, printLevel = 0):
    numNeedles = 100
    sDev = precision
    if printLevel > 0:
        print("{:13s} {:13s} {}".format('Estimate', 'Std', 'Needles'))
    while sDev >= precision/1.96:
        if printLevel > 1:
            print('Trying', numNeedles, 'needles')
            curEst, sDev = getEst(numNeedles, numTrials,
                                  printLevel)
            numNeedles *= 2
    return curEst
```

π 3.152

2.96 - 3.34

est = 3.152

std = 0.9516

Est+1.96*std = 3.3385

3.3385- π = 0.197

Output

Estimate	Std	Needles
3.152	0.1417603612	100
3.1415	0.1154458748	200
3.131	0.0745922248	400
3.138875	0.0574073983	800
3.1390625	0.0468631235	1600
3.1445625	0.0323536971	3200
3.14053125	0.0203694155	6400
3.1415	0.015885362	12800
3.14271875	0.0110747967	25600
3.142375	0.007102177	51200
3.1408544922	0.0055614574	102400
3.1409003906	0.0037853099	204800
3.1417026367	0.0023156432	409600

Is accuracy of estimates monotonically improving?
What is monotonically improving?

Being Right is Not Good Enough

- Not sufficient to produce a good answer
- Need to have reason to believe that it is close to right
- In this case, small standard deviation implies that we are close to the true value of π

Right?

Is it Correct to State

- 95% of the time we run this simulation, we will estimate that the value of π is between 3.137163976028 and 3.1462412973719998 ?
- With a probability of 0.95 the actual value of π is between 3.137163976028 and 3.1462412973719998 ?
- Both are factually correct
- But only one of these statements can be inferred from our simulation
- *statistically valid* \neq *true*

Introduce a Bug



Is it Correct to State

- 95% of the time we run this simulation, we will estimate that the value of π is between 3.137163976028 and 3.1462412973719998 ?
- With a probability of 0.95 the actual value of π is between 3.137163976028 and 3.1462412973719998 ?
- Both are factually correct
- But only one of these statements can be inferred from our simulation
- *statistically valid* \neq *true*



Simulating the Stock Market

- You won the lottery
- Want to invest in the stock market
 - Index fund or stock picking?
 - Investigate using a Monte Carlo simulation
- Get a sense of how people actually build simulations
 - Build it a little bit at a time
 - Test each piece as we build
 - Start with simplifying assumptions
 - Examine results
 - Refine as needed



Underlying Hypothesis

- Simulation is an experimental model of something
- We will start with efficient market hypothesis
 - 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 that is unknown in the present and thus appears as a random event in the future. This random information will be the cause of future stock 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 try and build a random walk of stock market, and see how it does
- Start with classes Stock and Market

What is most important question to consider?

How to Model Price Changes



- Already decided should be random, but still lots of questions
 - What is the most appropriate way to choose a 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?

How to Model Price Changes



- Already decided should be random, but still lots of questions
 - What is the most appropriate way to choose a 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 \approx variance (of a probability distribution)

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 change in stock price

Price can't go negative; once 0 always 0

Change is zero mean; uniform over range

In this model, price equally like to rise or fall

Getting 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 over time, as a percentage

Stock Market

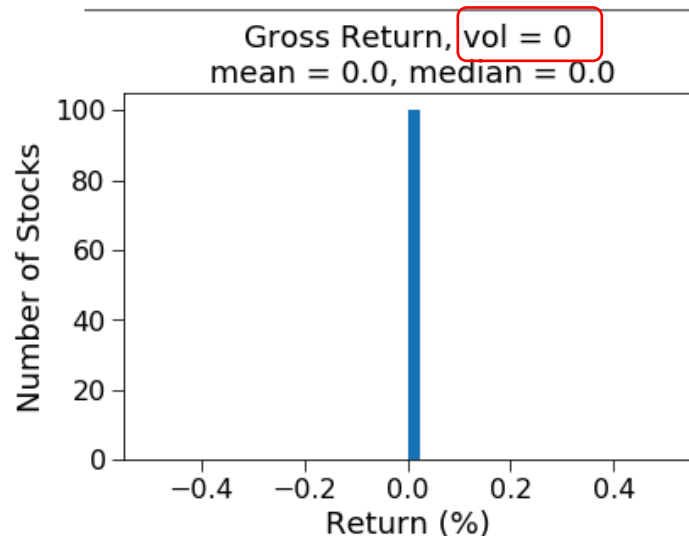
```
class Market(object):  
    def __init__(self):  
        self._stks = []  
    def addStk(self, stk):  
        self._stks.append(stk)  
    def getStks(self):  
        return self._stks.copy()
```

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')
```

Smoke Test

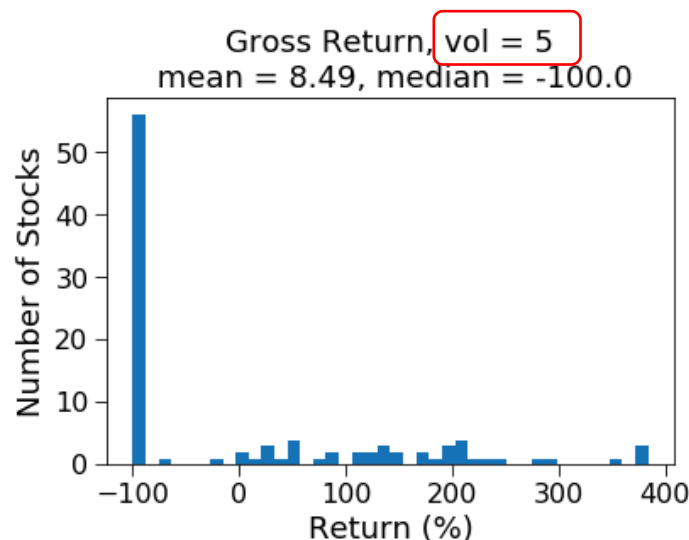


Looks fine

*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!*



Modeling Price Changes

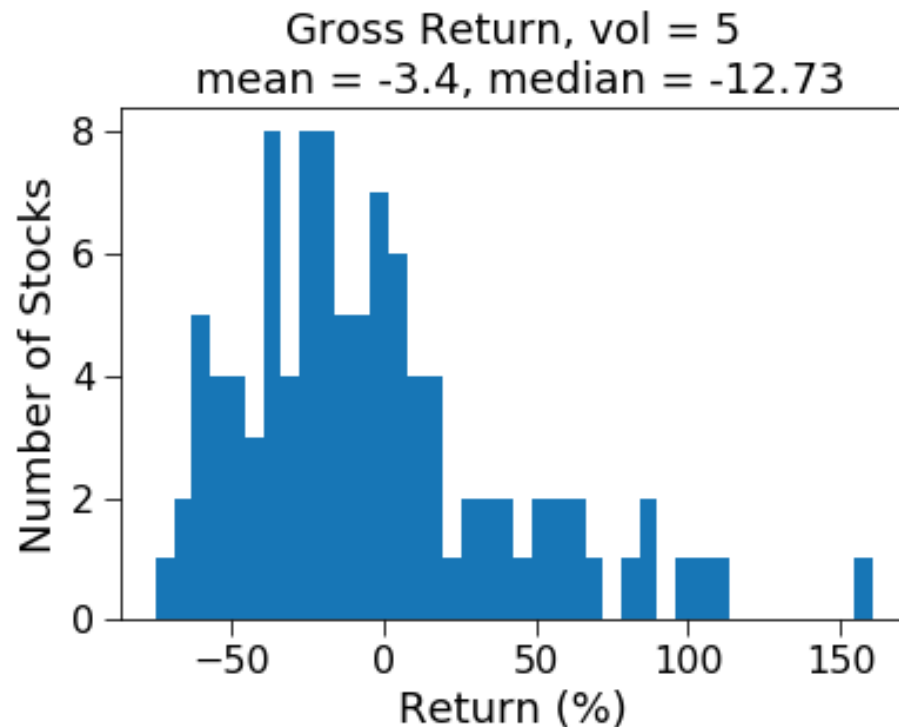
- Should magnitude of change be independent of stock price?
- Change volatility to be percentage change
- Go to file `simStocks1.py`

```
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

Multiply percentage change by price and add as actual change

Result of Change



Try a few times with different seeds

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

This doesn't match observations

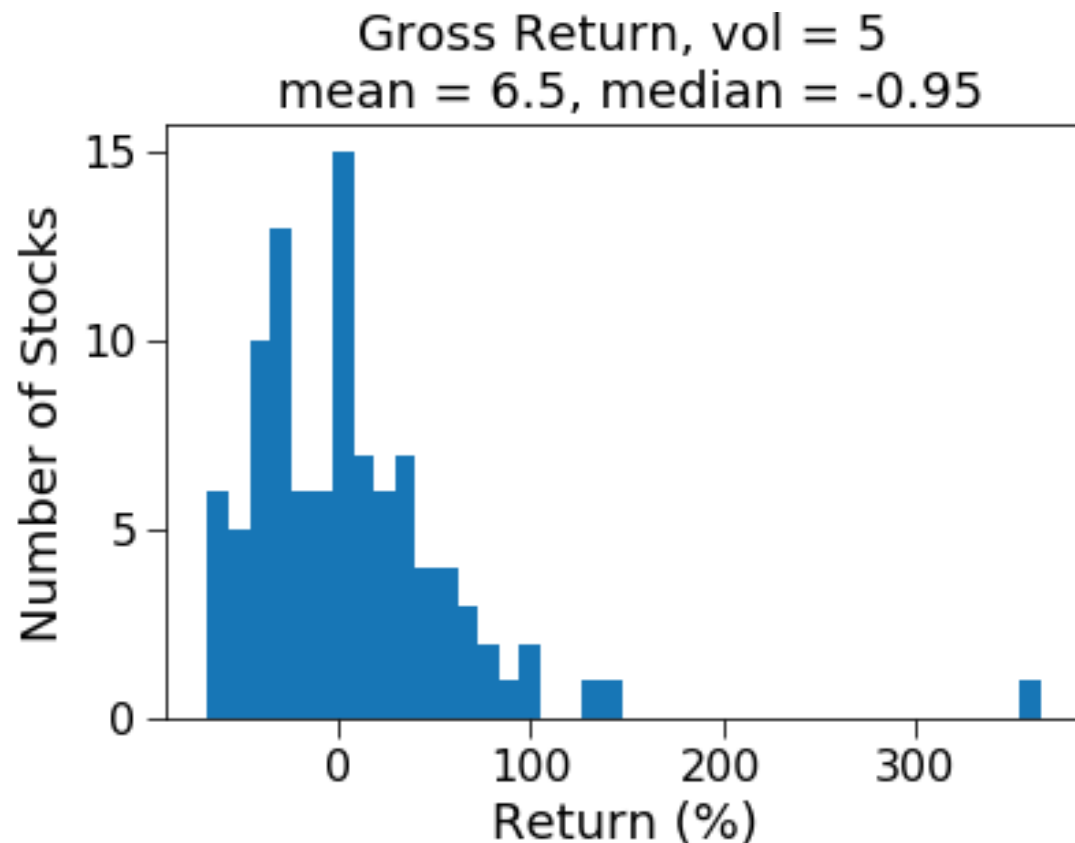
What Does Actual Market Do?

- Goes up about 6%/year (inflation adjusted)
- 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 general upward bias

Result of Change



Mean seems plausible

What about median?

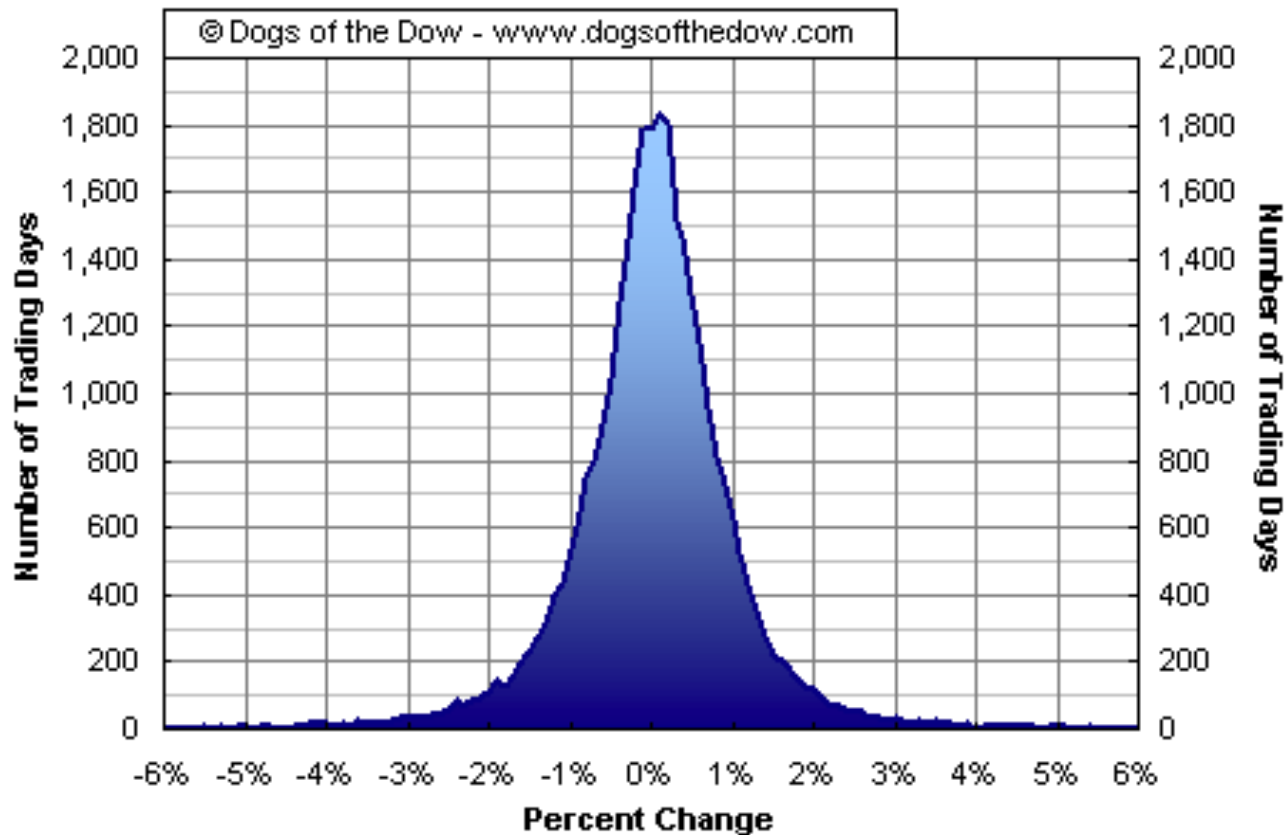
Too small

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

What about tails?

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

Are Price Changes Uniformly Distributed?



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)
```

Assume inputs are percentages;
convert to decimal fraction

Sample from Gaussian; scaled by
price since percent change

Add change to previous price

simStocks3.py

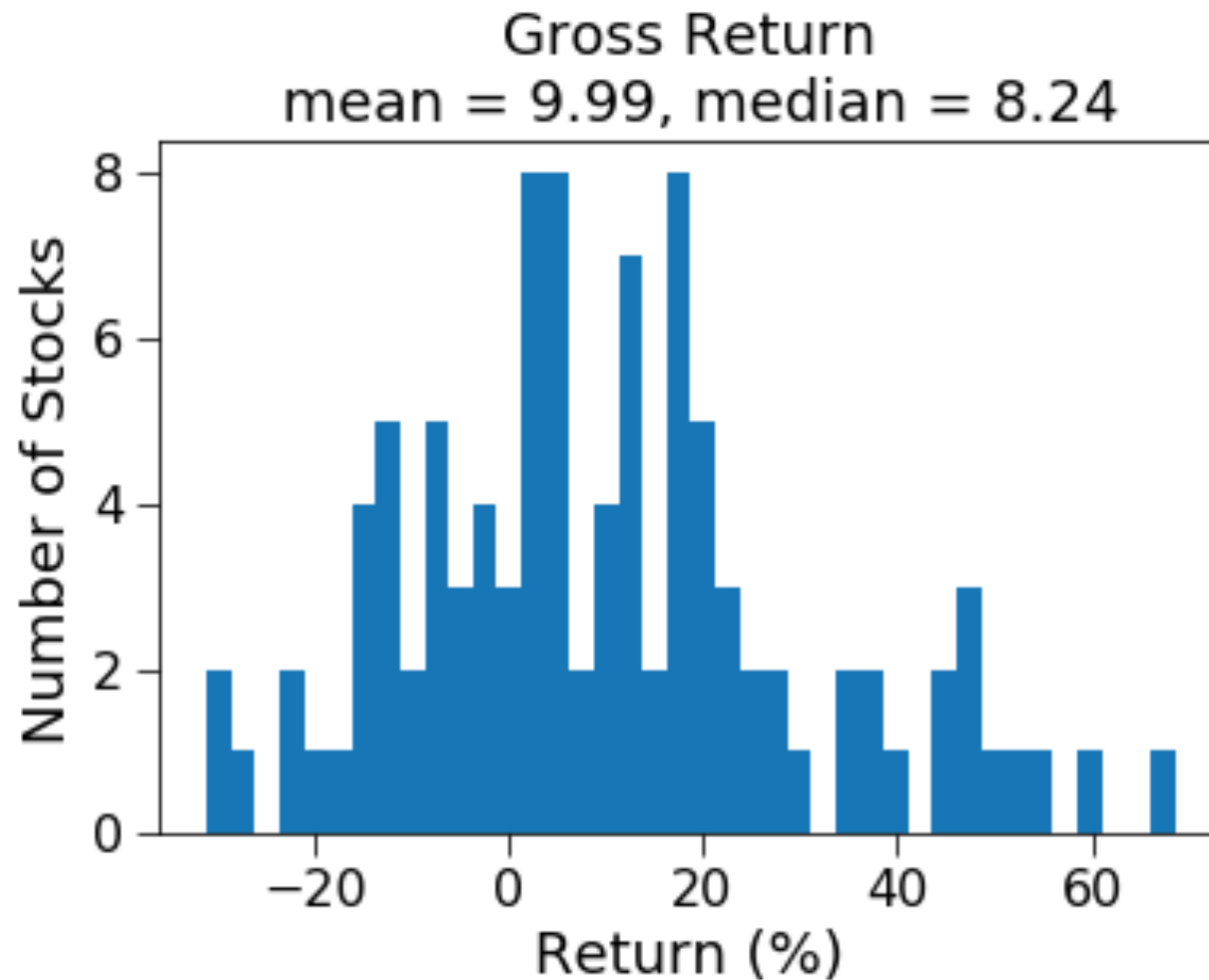
And Restructured Testing Code

```
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
```

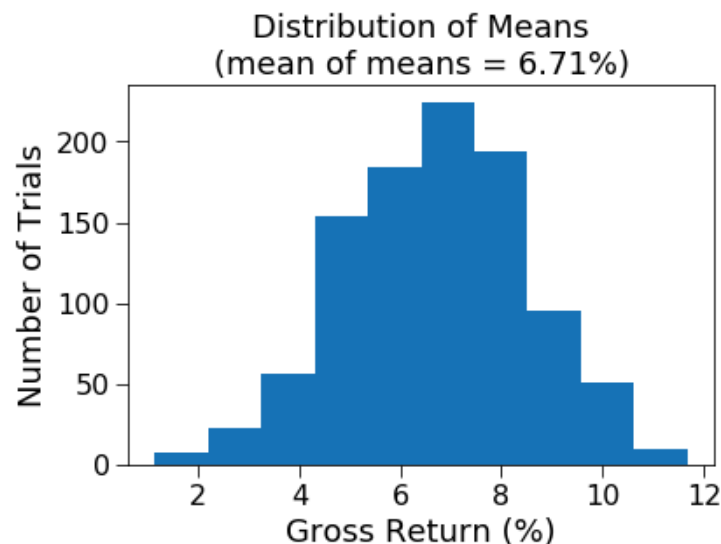
Actual values come
from historical data
(previous slide)

Volatility(0.026, 1.07)

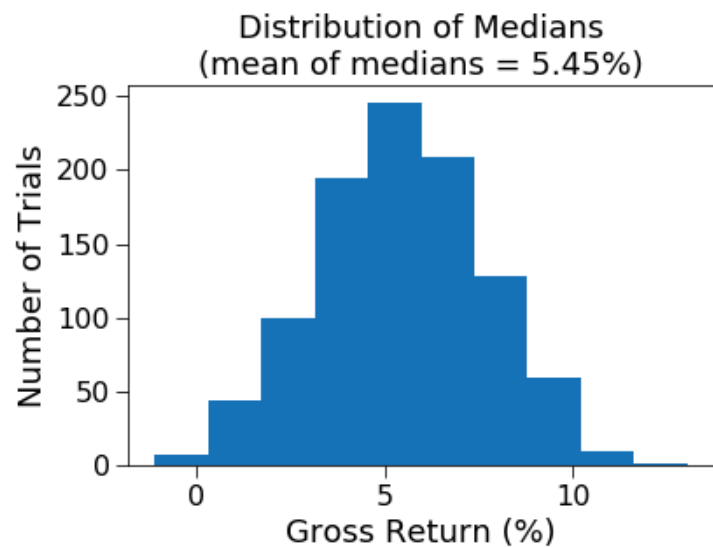


*But this is only
one Trial*

Try 1000 Trials



Averaged over all stocks for each trial, means are 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

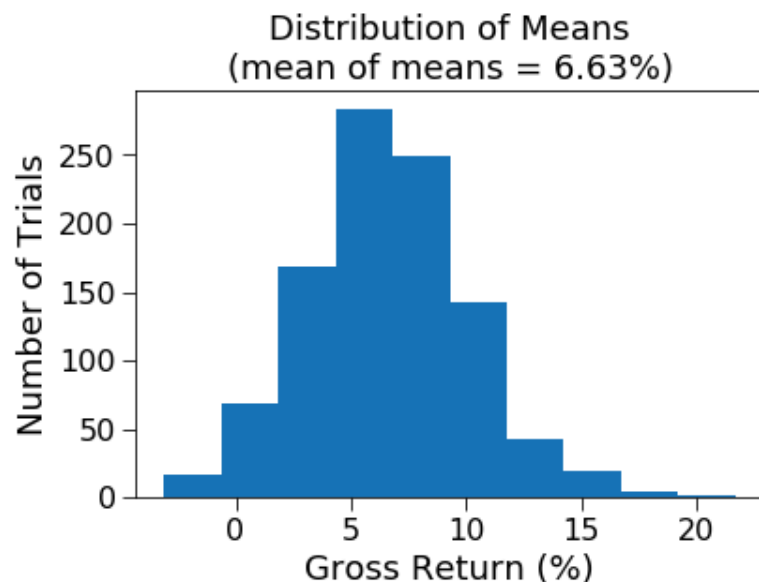
- Buying whole market (or index fund), gives return of ~6.71%
- Buying one stock at random, likely to under-perform buying an index fund
- But perhaps this is a function of all stocks in our simulated market having identical properties
 - Do we expect Amazon and GE to behave similarly?

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
```

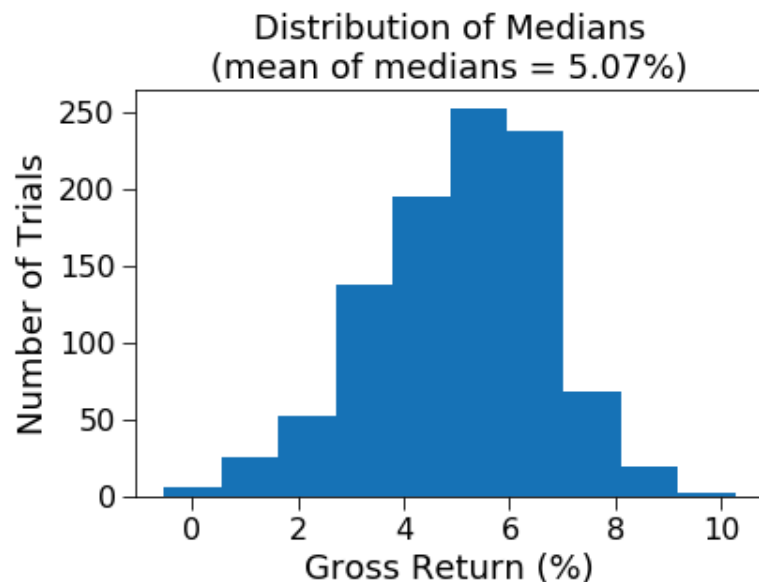
simStocks4

Try It Out



Variance is small

Most years are average
years



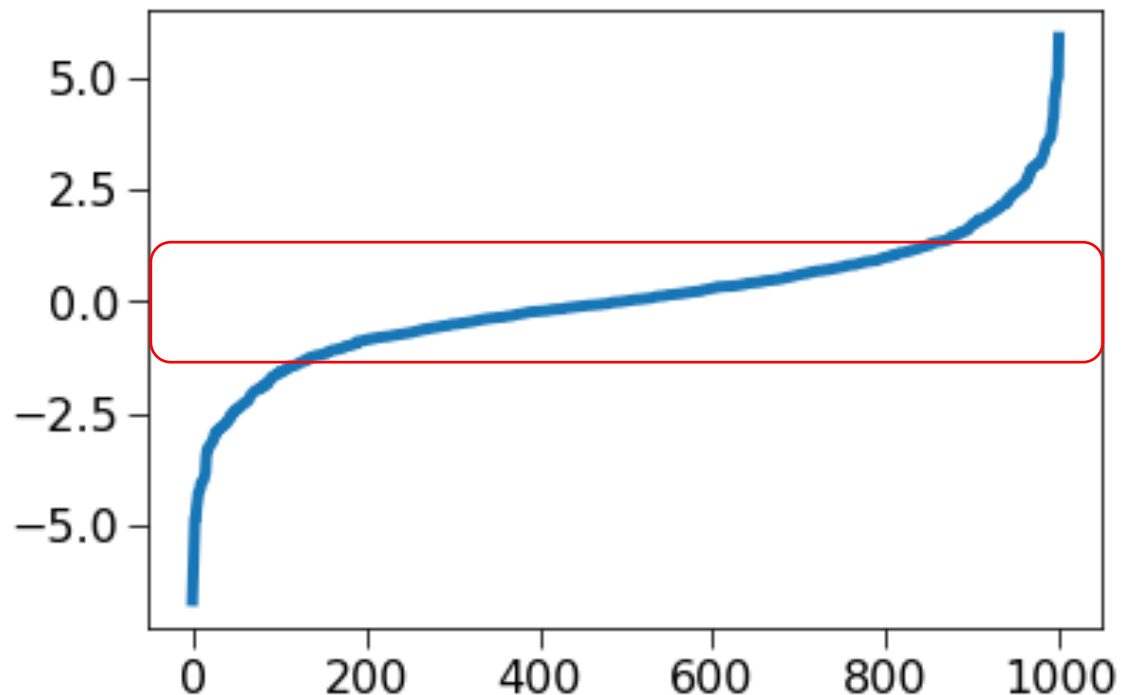
And almost ALL years,
good years!

What's Going On

- 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 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))
```



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)
```

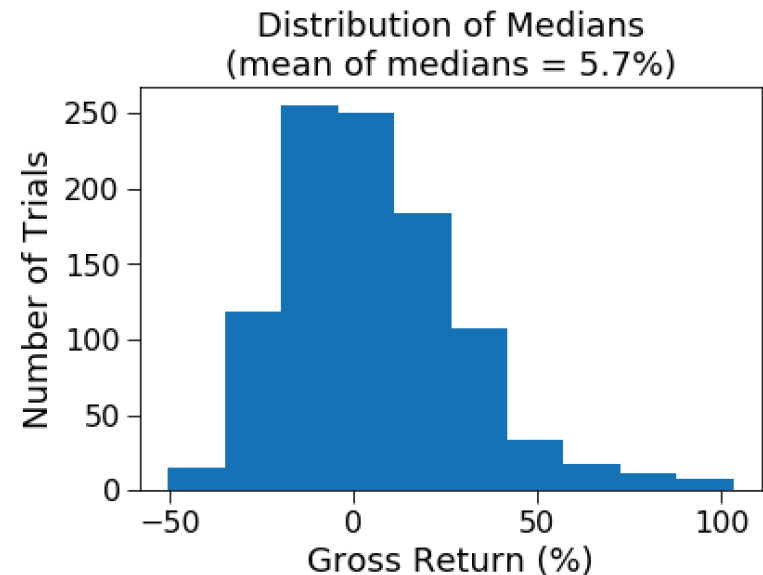
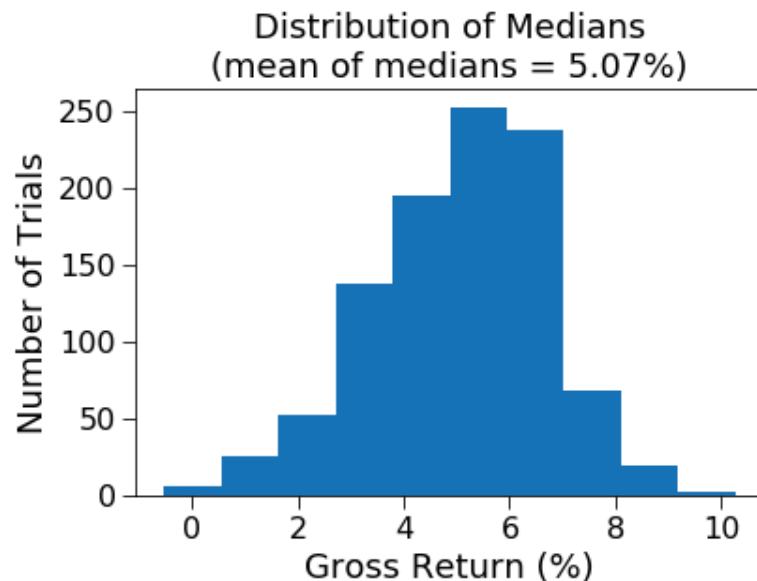
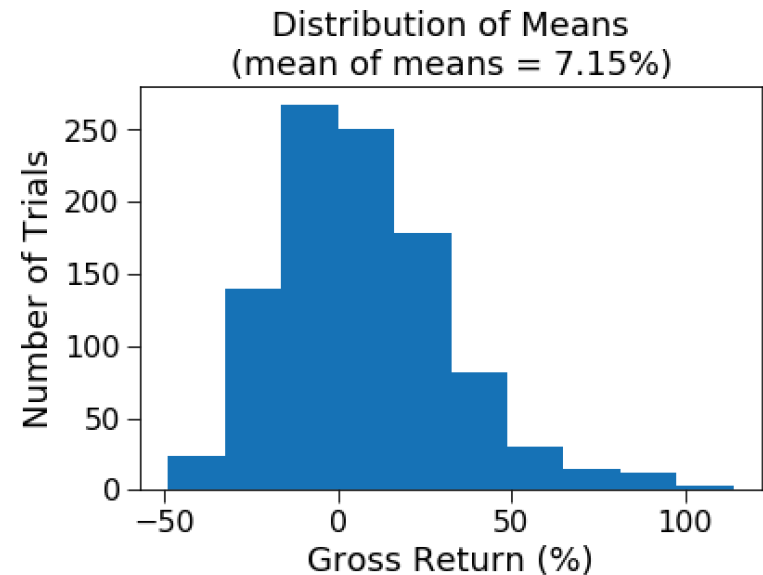
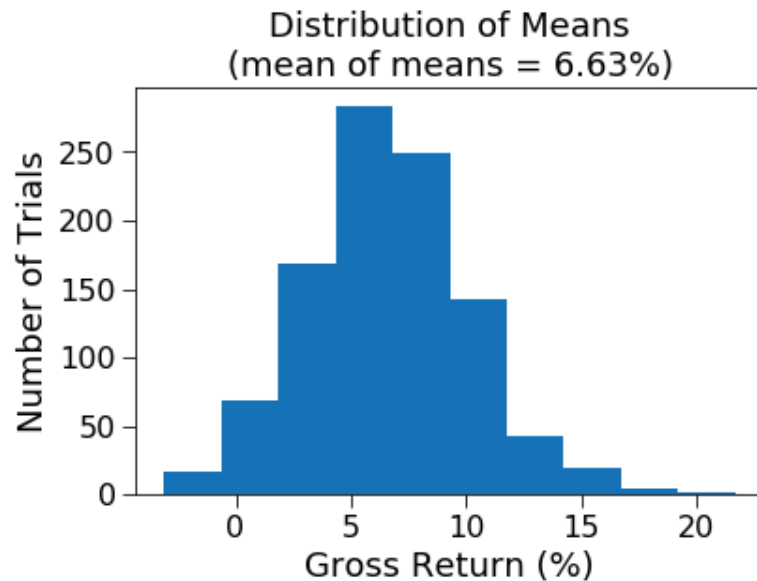
simStocks5.py

Change Testing Code

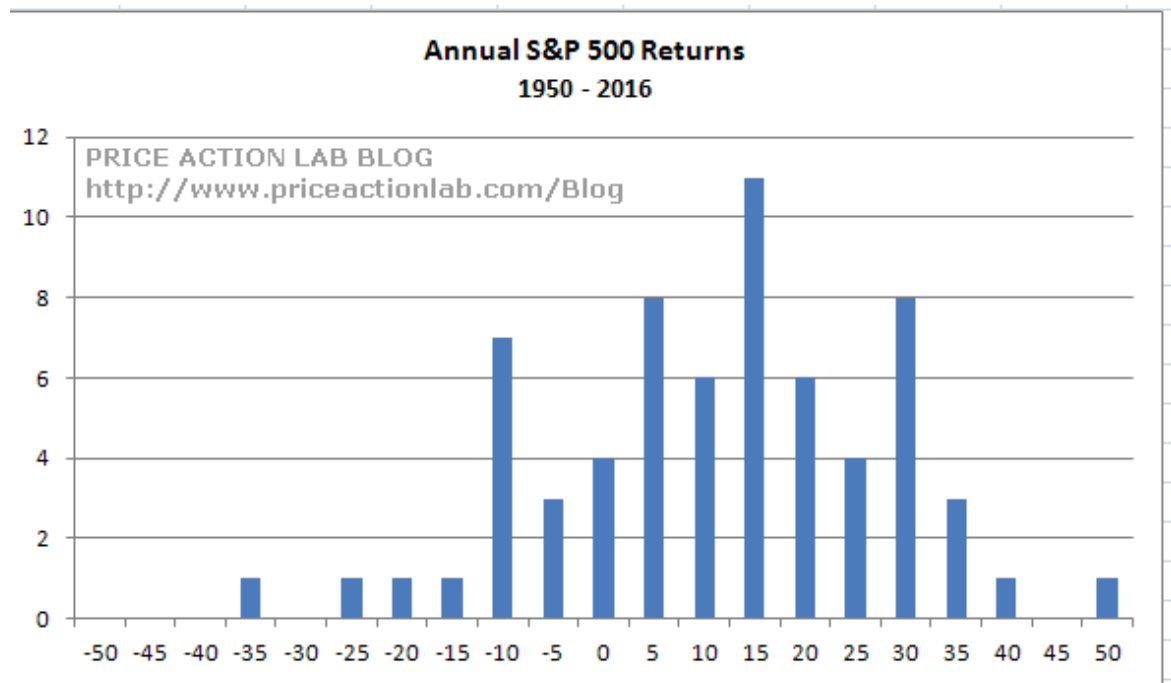
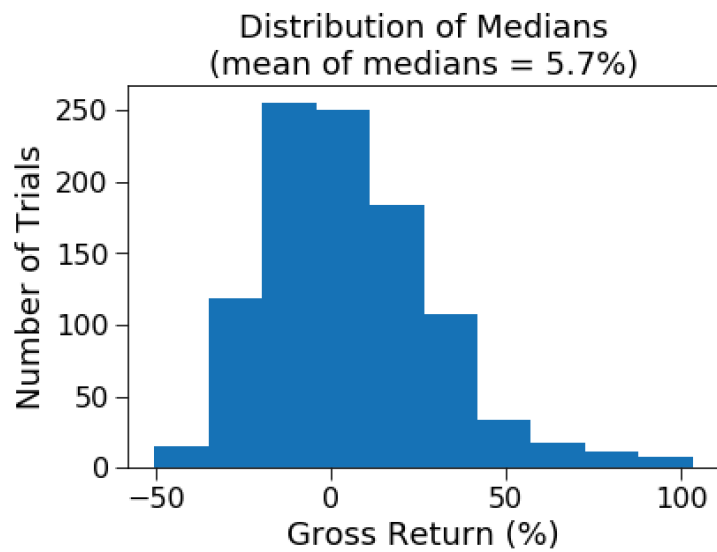
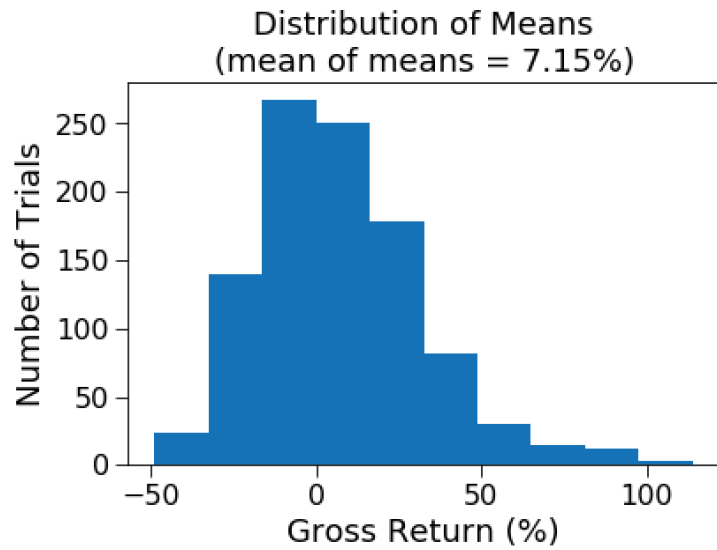
```
def getAnnRet(stk, dailyBiases):  
    for d in range(TRADINGDAYS):  
        stk.makeMove(dailyBiases[d])  
    hist = stk.getHistory()  
    return 100*((hist[-1] - hist[0])/hist[0])
```

```
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)
```

Without and With Daily Bias ($\lambda = 1$)



1000 Trials, $\lambda = 1$



Plausibly like actual data

Once again, median
smaller than mean

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

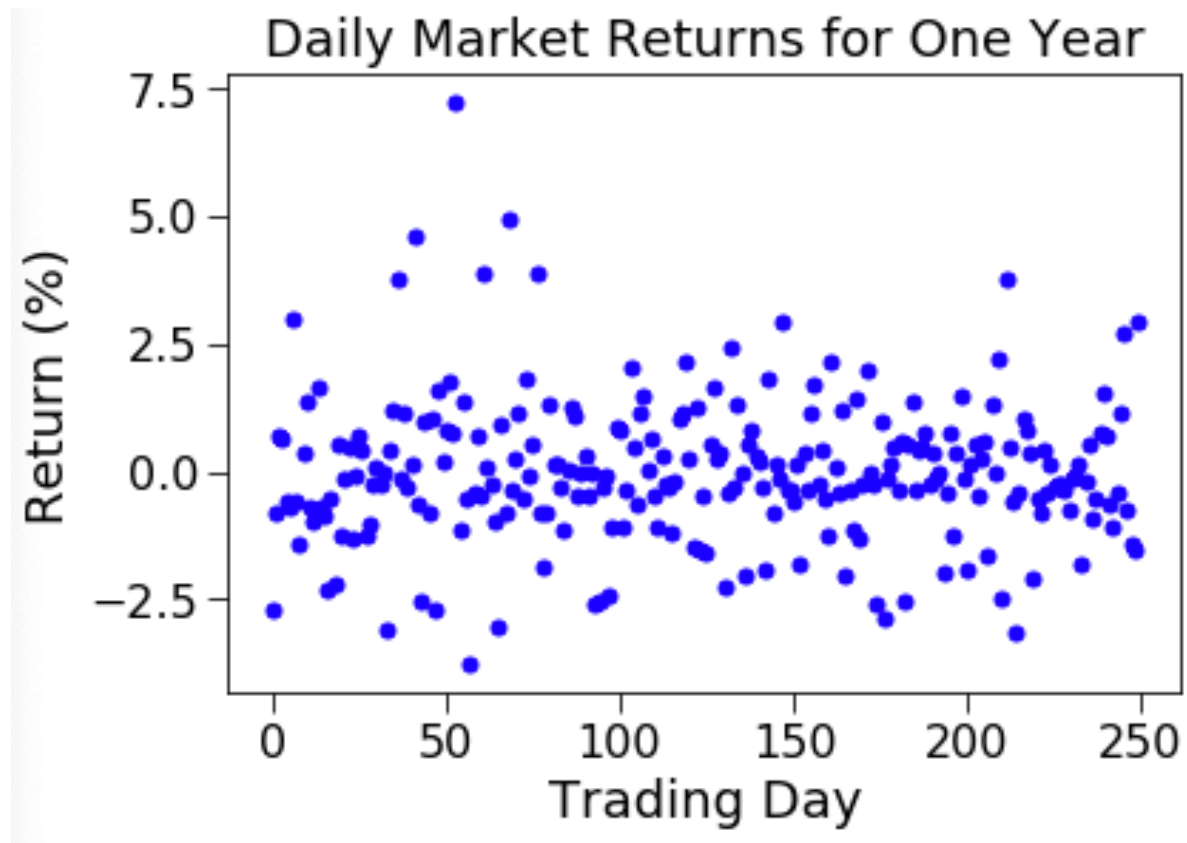
days starts as an array of 0's, one for
each day; on each iteration,
percentage change of each stock is
added per day;

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

An example of a "list
comprehension"

simStocks6.py

One Randomly Chosen Year

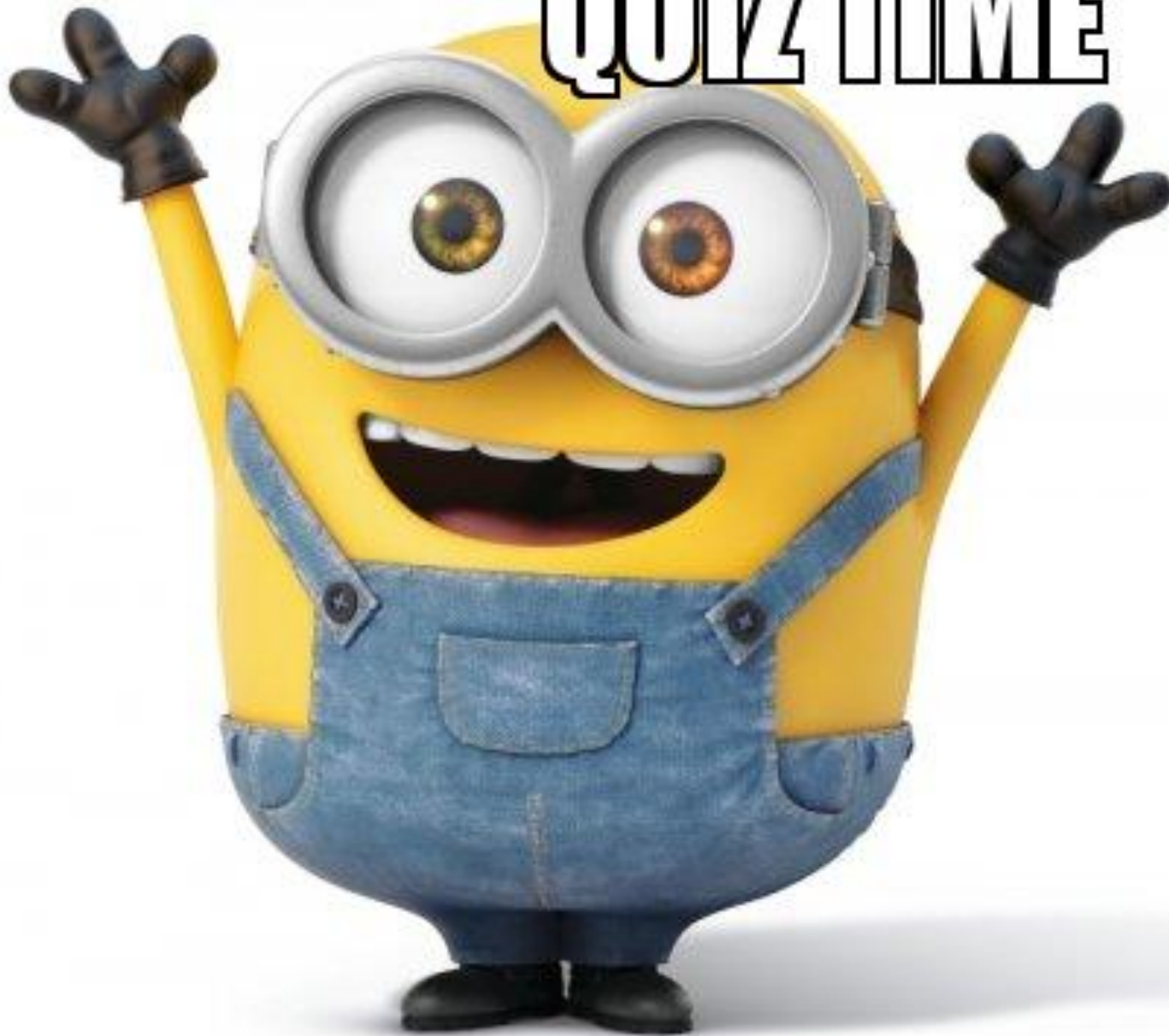


Timing the market is risky

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

QUIZ TIME



makeameme.org