Using Markov Chains and Random Walk to Model the Stock Market

1. Introduction

This study was conducted in hopes of determining whether or not the price of a stock can be accurately predicted using differing stochastic process methods. The first and most simple being a random walk process and then moving on to more complicated methods such as Markov Chains. The stock market is one of the most complex financial entities and even just having a singular percentage point above the rest of the field when it comes to predicting and choosing stocks to invest in can be extremely lucrative. It is well known that there are a number of factors that go into determining a company's stock price at any given time. These include the financial standing of the company, financial statements tell investors a great deal about how a company conducts business and also provide a number of ratios that investors use when deciding what to invest their money into. Events going on within the world, country, sector the company resides in, or just the company itself can also affect its stock price by providing information on future events as well as by affecting the sentiment of investors and persuading or dissuading them from acting. The main goal of this study is to hopefully simplify the prediction methods for both markets as a whole and individual stocks through the identification of certain trends.

1.1 Efficient Market Hypothesis

The efficient market hypothesis is a financial theory that claims stock (and other asset prices) immediately adjust to reflect all information in the market and thus it is impossible for investors to act on such information. First introduced in 1900 by Louis Bachelier and was popularized in the 1970s by Eugene Farm, the theory generalizes that even professional investors cannot accurately predict stocks that will outperform the market and thus the market is inherently completely random. More or less the theory claims that stocks follow a random walk process

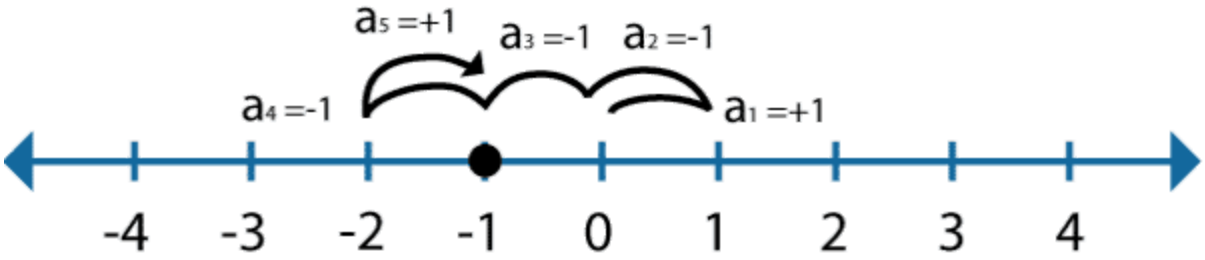
with slight drift. Burton Malkiel of Princeton University and the author of *A Random Walk Down Wall Street: The Time-Tested Strategy for Successful Investing*, claims that "a blindfolded chimpanzee throwing darts at the Wall Street Journal could select a portfolio that would do as well as the experts."

1.2 Random Walk

Random walk is a process consisting of successive steps independent of any previous steps where a value can either move up or down one unit of its current value. The probability of moving up is given by p and the probability of moving down is given by 1-p, where p is a value between 0 and 1. If $p = 0.5$ then the process is expected to stay at its original state whereas if $p < 0.5$ the process will drift to $-\infty$ and if $p > 0.5$ it will drift towards ∞. In Malkiel's book he argues that investors are much better off simply buying a broad index fund and allowing it to grow over time rather than wasting time attempting to use specific analysis to predict prices. The assumption that the stock market is unpredictable and efficient leads to the idea that random walk theory is a viable option in stock trading where investors instead focus on long term decisions.

Obviously there are many arguments against the use of such a simple process to model such a complicated institution that is the stock market. Some of these include the argument that historical data can provide information on trends, the fact that markets are not self-contained and are affected by many outside sources, primarily governments and central banks which control interest rates and taxes, and unethical trading in the form of insider trading. Also because information is not distributed evenly some investors are thought to have an edge over others with insufficient information. All of these things cause discrepancies in the market and thus cause it to be inefficient, or at least less efficient, putting into question the random walk theory.

However despite these arguments against random walk theory there is proof of it working to some degree. While Malkiel's previous metaphor to dart throwing monkeys was seemingly



hyperbole, the Wall Street Journal decided to make what seemed like such an outlandish idea a reality in 1988. They had four staffers throw a dart to choose a stock and another four 'experts' choose a stock on their own. They then compared the changes after six months compared against one another as well as against the Dow Jones Industrial Average. After 142 of these contests the experts won 87 times or 61.27% of the time, while this may seem significant the experts only beat the DJIA 76 times or 53.52% which does not seem very significant with only 142 competitions.

2. Using Python to Run a Geometric Random Walk Simulation

Geometric random walk occurs when instead of moving up or down a predetermined unit the change is based on a multiplicative constant and the current value in the process. The multiplicative constant causes compounding.

```
import random
# set counter for first while loop that will iterate the main process 1000
times
count1 = 0
# prompt user to input a starting value
start = float(input('Input a starting stock price '))
```

```
# create an empty list for that each price will be added to after each
iteration
total = []
# start the first while loop
while count1 < 1000:
# set the price equal to the originally input start price each iteration
of the loop
    price = start
# set counter for second while loop that will iterate the random walk
process 100 times
    count2 = 0
# start the second while loop
    while count2 < 100:
# randomly select a number between 0 and 1
        p = random.random()
# if the value of p is greater than 0.5 multiply the price after the
previous iteration by 1.011 (1.1% increase)
        if p > 0.5:
            price = price*1.01
# if the value of p is less than 0.5 multiply the price after the previous
iteration by 0.99 (1% decrease)
        else:
            price = price*0.99
# increase the count of the second while loop by 1 and restart
        count2 = count2 + 1
# increase the count of the first while loop by 1 and restart
    count1 = count1 + 1
```

```
# append the ending price after 100 iterations of the second loop to the
total list
    total.append(price)
# print the average ending price of each iteration of the first loop
print(sum(total)/len(total))
# create a definition for variance that can be run
def variance(data):
    n = len(data)
    mean = sum(data)/n
    deviations = ((x-mean) ** 2 for x in data)
    # sum up squared deviations and divide by n-1 to calculate the sample
variance
    variance = sum(deviations)/(n-1)
    return variance
# print the sample standard deviation
print(variance(total)**(1/2))
```

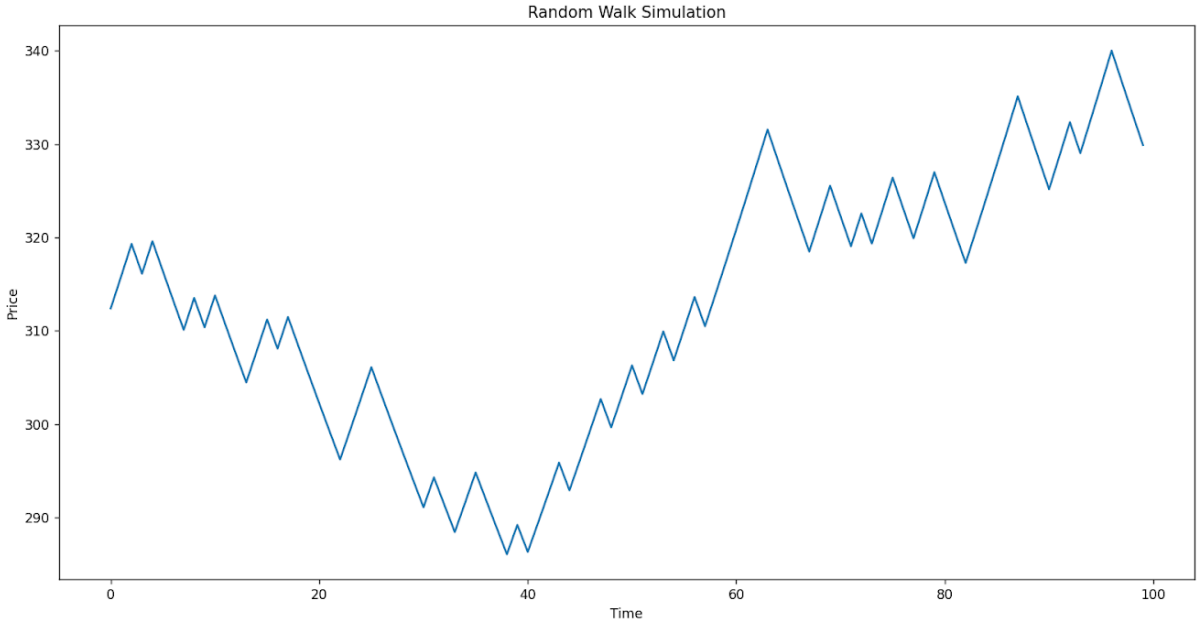
Above is code used to produce a random walk process 1000 times then determine the average final price and standard deviation after 100 days worth of iterations

The code takes into account the slight drift stocks have that tends to favor increasing in value over time by increasing by 1.1% when $p > 0.5$ but decreasing by 1% when $p < 0.5$. If the simulation is only run one time it can produce graphical output that attempts to mimic the trends in stock movement.

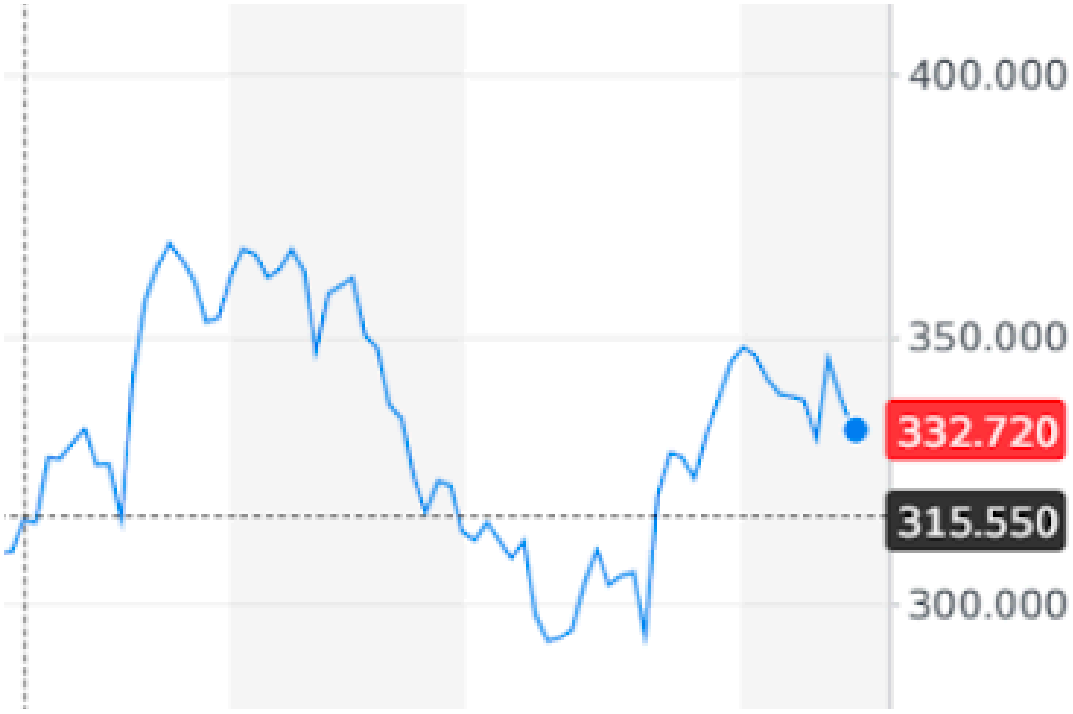
```
import random
import matplotlib.pyplot as plt
price = float(input('Input a starting stock price '))
```

```
# create an empty list for that each price will be added to after each
iteration
prices = []
# create an empty list for the value of each iteration that will be added
each time
x = []
count = 0
while count < 100:
    p = random.random()
    if p > 0.5:
        price = price*1.011
    else:
        price = price*0.99
# append each price to the prices list after each iteration
    prices.append(price)
# append the integer value that the count is at to the x list that will be
used as the x axis for the graph
    x.append(count)
    count = count + 1
# plot the x list on the x axis and prices on the y axis
plt.plot(x, prices)
# name the graph Random Walk Simulation with x label time and y label
price
plt.xlabel('Time')
plt.ylabel('Price')
plt.title('Random Walk Simulation')
plt.show()
```

Adjusted code to run one iteration and print a graph of what happens within the 100 steps



In this simulation the stock started at $315.55 in attempts to mimic Netflix's stock.



The simulation does not necessarily follow the same path as the actual Netflix stock over a 100 day period but it is pretty close and both the simulation and real stock finish with a closing price of around $330. Obviously this is just one simulation and is not indicative of every iteration, which could have drastically different outcomes. But it does help to show how through random walk a very realistic outcome can be produced. When the loop is run 1000 times the average ending value is $332.53 or an increase of 5.38% with a standard deviation of $36.38.

It is reasonable to apply a t-test to this outcome because;

a. The population standard deviation is unknown
b. The sample is large, with 1000 samples the central limit theorem will apply and the price outcomes will be normally distributed
c. Each outcome is randomly generated and thus each outcome is independent

$$\overline{x} \pm t_{\alpha/2} s/\sqrt{n}$$

Creating a 99% confidence interval would result in

$$332.53 \pm 2.58 * (36.38/\sqrt{1000}) = (329.56, 335.5)$$

Obviously the random walk process was designed to generate a sample that would result in an increase in price, however it is still abundantly clear that this is in fact the case. The model itself does a relatively good job at modeling the real stock market as investors can expect about a 10% return in a year (252 trading days) and an increase of about 5% in a 100 day period is only slightly outpacing that projection.

3. Markov Chains

Markov chains are processes where the probability of the next event occurring is dependent only on the current state of the process. Similar to random walks they are memoryless

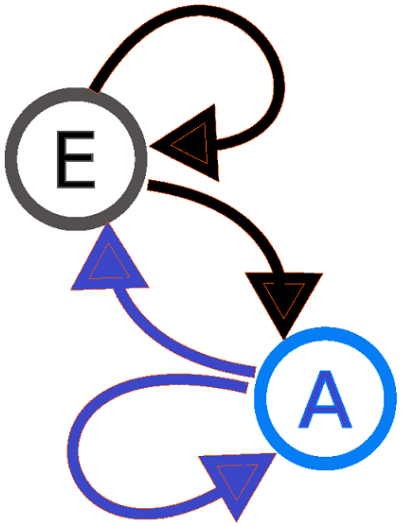
and are unrelated to previous events. Due to this memoryless property markov chains can be modeled with the equation:

$$P(X_{n+1} = j | X_0 = x_0, X_1 = x_1, \ldots, X_{n-1} = x_{n-1}, X_n = i) = P(X_{n+1} = j | X_n = 1)$$

Where $X_0, \ldots, X_n$, i, j are all contained in the state space of the markov chain and $n \geq 0$

Markov chains are most often modeled through the use of a transition matrix that provides the probability of moving from one state, i, to another state, j on the next step.

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,j} & \cdots & P_{1,\alpha} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,j} & \cdots & P_{2,\alpha} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{i,1} & P_{i,2} & \cdots & P_{i,j} & \cdots & P_{i,\alpha} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{\alpha,1} & P_{\alpha,2} & \cdots & P_{\alpha,j} & \cdots & P_{\alpha,\alpha} \end{bmatrix}.$$



Using a markov chain to predict the future outcome of a stock is something that is much more rooted in actual data that the market provides by being able to detect trends as opposed to simply just randomizing what will happen next. Many experts claim trends can be predicted based on past historical data and what has happened recently to a stock. While a markov chain does not necessarily take into account the long run trend of stocks it can be useful to predict what will happen next based on its current state. This obviously implies that only past information and outside factors have no effect on how well a stock will perform, which is most likely untrue, but the question still arises, how well can such a model mimic and predict the market.

4. NYSE Historical Data

In order to create both steady states, the long term probability of being in a given state, and the transition probabilities data from the New York Stock Exchange was gathered and adjusted in order to determine these values. First monthly data dating from January 1, 2000 to April 1, 2023 was gathered, 280 months worth of data. The hope of this was to detect more long term trends in data rather than simply week to week or even day to day changes which can be extremely volatile. The percentage change from month to month was focused on and sorted into one of five groups; bull, a change of 5% or more from open to close over the month, slight bull, a change of 0.5% to 5%, no change, a change of -0.5% to 0.5%, slight bear, a change of -5% to -0.5%, and bear, a change of -5% or greater.

	Count	P
Bull (P_0)	31	0.110714
Slight Bull (P_1)	118	0.421429
No Change (P_2)	26	0.092857
Slight Bear (P_3)	77	0.275
Bear (P_4)	28	0.10

Next each of the states was isolated on its own and the probability of moving from a given starting state to another state was calculated in order to create a transition matrix. Which can be represented by the following:

	0	1	2	3	4
0	0.16129	0.419355	0.096774	0.322581	0
1	0.017094	0.52991 5	0.076923	0.290598	0.08547
2	0.115385	0.461538	0.115385	0.192308	0.115385
3	0.155844	0.311688	0.142857	0.25974	0.12987
4	0.321429	0.25	0	0.25	0.17857 1

Finally the average change given each group was calculated so that that change could be applied to the price at any given time based on its state. Those averages are the following:

- Bull, 7.605%
- Slight Bull, 2.402%
- No Change, 0.069%
- Slight Bear, -2.427%
- Bear, -8.327%

5. Using Python to Run a Markov Chain Simulation

Similar to the Python code used before this code runs a process that decides what the next state will be; however instead of using arbitrary numbers, 50/50 chance of increasing or decreasing and 1.1% or 1% increase or decrease depending on the outcome, it uses the data gathered to simulate what will happen next. The code also conditions on the current value in the chain to decide its next move.

```
import random
import matplotlib.pyplot as plt
startprice = float(input('Input a starting stock price '))
# create a blank list of prices that will be appended after each iteration
finalprices = []
count2 = 0
while count2 < 1000:
    start = random.random()
    if start < 0.110714:
        state = 0
    elif start < 0.532143:
        state = 1
```

```
    elif start < 0.625:
        state = 2
    elif start < .9:
        state = 3
    else:
        state = 4
    count = 1
    price = startprice
    while count < 281:
        p = random.random()
# condition each outcome based on which state the loop is currently in and
the transition probabilities of each state and multiply current price by
the average change for that state then set loop to new state
        if (p < 0.16129 and state == 0) or (p < 0.017094 and state == 1)
or (p < 0.115385 and state == 2) or (p < 0.155844 and state == 3) or (p <
0.321429 and state == 4):
            price = price*(1.07605)
            state = 0
        elif (p < 0.580645 and state == 0) or (p < 0.547009 and state ==
1) or (p < 0.576923 and state == 2) or (p < 0.467532 and state == 3) or (p
< 0.521429 and state == 4):
            price = price*(1.02401)
            state = 1
        elif (p < 0.677419 and state == 0) or (p < 0.623932 and state ==
1) or (p < 0.692308 and state == 2) or (p < 0.610389 and state == 3):
            state = 2
```

```
        elif (p < 1 and state == 0) or (p < 0.91453 and state == 1) or (p
< 0.884616 and state == 2) or (p < 0.870129 and state == 3) or (p <
0.771429 and state == 4):
            price = price*(0.97573)
            state = 3
        else:
            price = price*(0.91673)
            state = 4
        count = count + 1
    finalprices.append(price)
    count2 = count2 + 1
def variance(data):
    n = len(data)
    mean = sum(data)/n
    deviations = ((x-mean) ** 2 for x in data)
    variance = sum(deviations)/(n-1)
    return variance
print(sum(finalprices)/len(finalprices))
print(variance(finalprices)**(1/2))
```

When run 1000 times the code provides a mean final price of $231.45 and a standard deviation of $186.44, based on a starting price of 100. Similar to the random walk a t-test can be applied to the markov chain simulation to create a confidence interval. In this case a 99% confidence interval would result in:

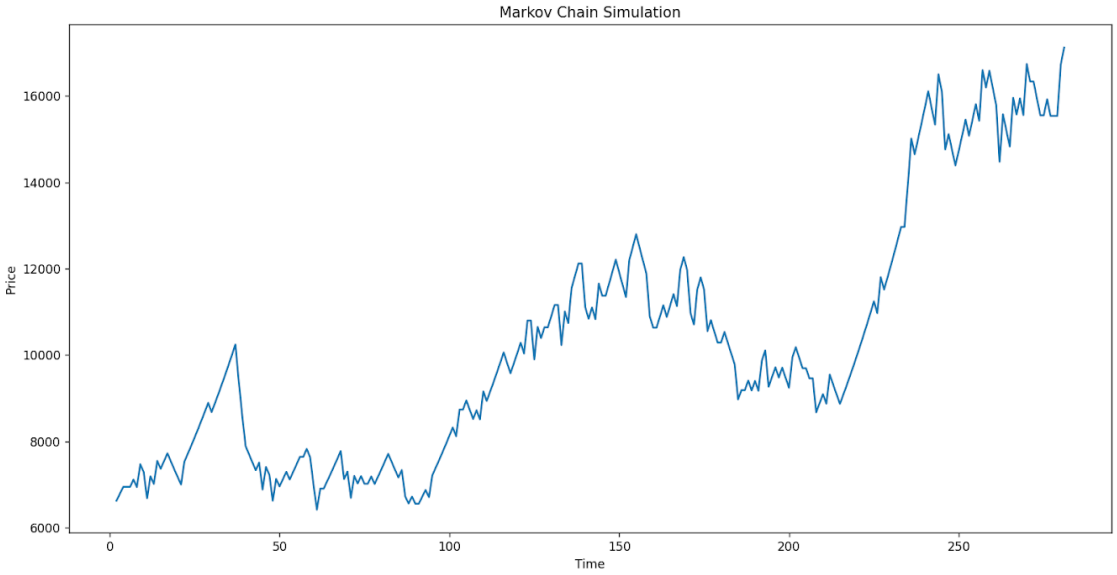
$$231.45 \pm 2.58 * (186.44/\sqrt{1000}) = (216.23, 246.67)$$

Since 2000 the NYSE has risen from $6574 to $15500.90 or a 135.79% increase which is very similar to the 131.45% increase shown in the simulation.

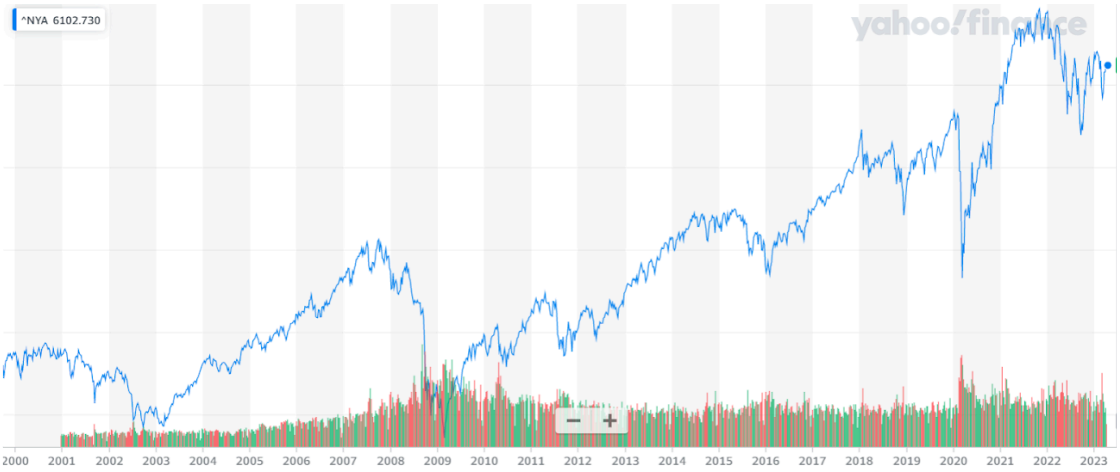
Just as the random walk process could provide information on a single loop's change in price throughout, so can the markov chain simulation with a few quick adjustments.

```
import random
import matplotlib.pyplot as plt
count = 1
state = 0
price = float(input('Input a starting stock price '))
monthly_prices = []
x = []
start = 0
if start < 0.110714:
    state = 0
elif start < 0.532143:
    state = 1
elif start < 0.625:
    state = 2
elif start < .9:
    state = 3
else:
    state = 4
while count < 281:
    p = random.random()
    if (p < 0.16129 and state == 0) or (p < 0.017094 and state == 1) or (p
< 0.115385 and state == 2) or (p < 0.155844 and state == 3) or (p <
0.321429 and state == 4):
```

```
        price = price*1.07605
        state = 0
    elif (p < 0.580645 and state == 0) or (p < 0.547009 and state == 1) or
(p < 0.576923 and state == 2) or (p < 0.467532 and state == 3) or (p <
0.521429 and state == 4):
        price = price*1.02401
        state = 1
    elif (p < 0.677419 and state == 0) or (p < 0.623932 and state == 1) or
(p < 0.692308 and state == 2) or (p < 0.610389 and state == 3):
        state = 2
    elif (p < 1 and state == 0) or (p < 0.91453 and state == 1) or (p <
0.884616 and state == 2) or (p < 0.870129 and state == 3) or (p < 0.771429
and state == 4):
        price = price*0.97573
        state = 3
    else:
        price = price*0.91673
        state = 4
    count = count + 1
    monthly_prices.append(price)
    x.append(count)
plt.plot(x, monthly_prices,)
plt.xlabel('Time')
plt.ylabel('Price')
plt.title('Markov Chain Simulation')
plt.show()
print(price)
```



A single simulation compared to the NYSE since 2000



Obviously this simulation works when attempting to describe the NYSE over the last 23 years as that was the information used to develop the values used in the code. The real question is how well is the simulation able to mimic other stocks (such as single companies as opposed to the entire stock exchange) as well as for different time frames (such as a year as opposed to 23 years).

5.1 Adjusting for Time

After using the NYSE values and simply just shortening the time period that the simulation runs over, including time spans of five, two, and one years, it is very difficult to come to conclusion that the simulation can be scaled down and applied to different periods very accurately, even when using the same stock exchange that resulted in the original values. At times the value was overestimated and at others it was underestimated. One thing that could have been adjusted would have possibly been taking smaller incremental values as opposed to months which tend to have larger jumps in the value. If values were taken every week it may have led to more precise data since there would be more data points or the data could be condensed to less years and thus be less variable. The goal of collecting the data over 23 years was in attempts to get an overall picture of the market however this may have been a flaw that resulted in averaged data not necessarily reflecting the market well at specific points because of how drastically the market can change in such a large time span.

5.2 Adjusting for Other Stocks

When adjusting to other stocks both indexes or ETFs such as the S&P 500 or the Russell 2000 were used as well as individual stocks

Indexes/ETFs - Indexes and ETFs once again were not very accurately measured, at times simulating changes up to 50% off of what the actual percentage change was, this was extremely surprising as the point of many of these funds is to mimic the market and create an affordable, low risk stock that the general public can easily invest in. The funds follow a similar pattern to the market as a whole but despite this were not estimated very well using the data collected from the NYSE.

Individual Stocks - Individual stocks are obviously much more volatile than either funds or the stock exchange as a whole. In attempts to counteract this issue and make the the variability in the

real market reflected in the simulation the differences from start to final price of the stock was amplified using a constant multiplier of beta.

In finance beta is the measure of a stock's volatility in comparison to the market. The higher a beta the more volatile a stock is. The beta of the market as a whole is 1, meaning for every increase in percentage point of the market a stock increases by beta percentage points. This process was run with many stocks including Nike and Boeing, all of which resulted in underestimates. This may be due to the fact that many stocks simply do not last 23 years or more without either going bankrupt, returning private, or significantly beating the market. Due to this the use of such a long time period may make it difficult to draw any actual conclusions.

6. Conclusions

There is a lack of evidence to support the idea that using historical monthly data from the NYSE since 2000 can accurately predict stock market behaviors of both other stocks and of time periods not of such a long length. If this was to be done again, using smaller increments over a shorter, more stable period of time would likely lead to more concrete results that may provide information on exactly how well a stock will perform in comparison to the rest of the market. It would also be smart to sample data from differing markets and create models that could be used to predict specific industries that may differ in how they act as opposed to the stock market as a whole which often acts very differently than individual stocks due to its collective nature. Overall using a markov chain based on historical data has proven no more reasonable than simply a random walk process to estimate the future of individual stocks using simple randomness.