Simulation and Monte Carlo methods for Finance

MARCO LUPIA*

* *lupia.1694700@studenti.uniroma1.it*

A Monte Carlo simulation is a method that allows for the generation of future potential outcomes of a given event. In this paper I analyze Monte Carlo simulations and implement a program to forecast stock prices for a given asset.

1. INTRODUCTION

Risk analysis is part of every decision we make. We are constantly faced with uncertainty, ambiguity, and variability. And even though we have unprecedented access to information, we can't accurately predict the future. Monte Carlo simulation (also known as the Monte Carlo Method) lets you see all the possible outcomes of your decisions and assess the impact of risk, allowing for better decision making under uncertainty.

Monte Carlo simulation is a computerized mathematical technique that allows people to account for risk in quantitative analysis and decision making.

It furnishes the decision-maker with a range of possible outcomes and the probabilities they will occur for any choice of action.

The technique was first used by scientists working on the atom bomb; it was named for Monte Carlo, the Monaco resort town renowned for its casinos. Since its introduction in World War II, Monte Carlo simulation has been used to model a variety of physical and conceptual systems.

2. MONTE CARLO METHODS AND SIMULATIONS

Definition Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle.

The input model is simulated hundreds or thousands of times (or sometimes hundreds of thousands of times), where each end simulation is equally likely. The result is a probability distribution of possible outcomes. This could be one, or many different distributions including the normal distribution, chi-squared distribution, uniform distribution, or one of dozens more different probability distributions.

The Monte Carlo method tells you: All of the possible events that could or will happen, The probability of each possible outcome.

Working principles Monte-Carlo methods generally follow the following steps:

1. Determine the statistical properties of possible inputs
2. Generate many sets of possible inputs which follows the above properties
3. Perform a deterministic calculation with these sets
4. Analyze statistically the results

The error on the results typically decreases as $\frac{1}{\sqrt{N}}$

Monte Carlo simulation performs risk analysis by building models of possible results by substituting a range of values—a probability distribution—for any factor that has inherent uncertainty. It then calculates results over and over, each time using a different set of random values from the probability functions. Depending upon the number of uncertainties and the ranges specified for them, a Monte Carlo simulation could involve thousands or tens of thousands of recalculations before it is complete. Monte Carlo simulation produces distributions of possible outcome values.

By using probability distributions, variables can have different probabilities of different outcomes occurring. Probability distributions are a much more realistic way of describing uncertainty in variables of a risk analysis.

In a Monte Carlo simulation, a random value is selected for each of the tasks, based on the range of estimates. The model is calculated based on this random value. The result of the model is recorded, and the process is repeated. A typical Monte Carlo simulation calculates the model hundreds or thousands of times, each time using different randomly-selected values. When the simulation is complete, we have a large number of results from the model, each based on random input values. These results are used to describe the likelihood, or probability, of reaching various results in the model.

Applications Monte Carlo simulations are used to model the probability of different outcomes in a process that cannot easily be predicted due to the intervention of random variables. It is a technique used to understand the impact of risk and uncertainty in prediction and forecasting models.

A Monte Carlo simulation is an attempt to predict the future many times over. At the end of the simulation, thousands or millions of "random trials" produce a distribution of outcomes that can be analyzed.

Monte Carlo methods are used in corporate finance and mathematical finance to value and analyze instruments, portfolios and investments by simulating the various sources of uncertainty affecting their value, and then determining the distribution of their value over the range of resultant outcomes. This is usually done by help of stochastic asset models.

When faced with significant uncertainty in the process of making a forecast or estimation, rather than just replacing the uncertain variable with a single average number, the Monte Carlo Simulation might prove to be a better solution. Since business and finance are plagued by random variables, Monte Carlo simulations have a vast array of potential applications in these fields. They are used to estimate the probability of cost overruns in large projects and the likelihood that an asset price will move in a certain way.

Advantages Monte Carlo simulation provides a number of advantages over deterministic, or “single-point estimate” analysis:

- Probabilistic Results. Results show not only what could happen, but how likely each outcome is.
- Graphical Results. Because of the data a Monte Carlo simulation generates, it’s easy to create graphs of different outcomes and their chances of occurrence. This is important for communicating findings to other stakeholders.
- Sensitivity Analysis. With just a few cases, deterministic analysis makes it difficult to see which variables impact the outcome the most. In Monte Carlo simulation, it’s easy to see which inputs had the biggest effect on bottom-line results.
- Scenario Analysis: In deterministic models, it’s very difficult to model different combinations of values for different inputs to see the effects of truly different scenarios. Using Monte Carlo

simulation, analysts can see exactly which inputs had which values together when certain outcomes occurred. This is invaluable for pursuing further analysis.

- Correlation of Inputs. In Monte Carlo simulation, it's possible to model interdependent relationships between input variables. It's important for accuracy to represent how, in reality, when some factors goes up, others go up or down accordingly.

Disadvantages

- Time consuming as there is a need to generate large number of sampling to get the desired output.
- The results of this method are only the approximation of true values, not the exact.

A. Asset Price Modeling

One way to employ a Monte Carlo simulation is to model possible movements of asset prices. There are two components to an asset's price movements: *drift*, which is a constant directional movement, and a *random input*, which represents market volatility. By analyzing historical price data, it is possible to determine the drift, standard deviation, variance, and average price movement for a security. These are the building blocks of a Monte Carlo simulation.

To project one possible price trajectory, use the historical price data of the asset to generate a series of periodic daily returns using the natural logarithm:

$$\text{Periodic Daily Return} = \ln\left(\frac{\text{Day's Price}}{\text{Previous Day's Price}}\right)$$

Next use the `mean()`, `var()`, and `std()` functions on the entire resulting series to obtain the average daily return, standard deviation, and variance inputs, respectively.

The drift is equal to:

$$\text{Drift} = \text{Average Daily Return} - \frac{\text{Variance}}{2}$$

where:

- Average Daily Return=Produced from `mean()` function from periodic daily returns series
- Variance= Produced from `var()` function from periodic daily returns series

Next obtain a random input:

$$\text{Random Value} = \sigma \times \text{ np.random.normal())}$$

where:

- σ=Standard deviation
- `np.random.normal()` calculates the normal function

The equation for the following day's price is:

$$\text{Next Day's Price} = \text{Today's Price} \times e^{(\text{ Drift} + \text{Random Value})}$$

This calculation is repeated the desired number of times (each repetition represents one day) to obtain a simulation of future price movement.

By generating an arbitrary number of simulations, you can assess the probability that a security's price will follow given trajectory.

The frequencies of different outcomes generated by this simulation will form a normal distribution, that is, a bell curve. The most likely return is at the middle of the curve, meaning there is an equal chance that the actual return will be higher or lower than that value. There is no guarantee that the most expected outcome will occur, or that actual movements will not exceed the wildest projections.

B. Geometric Brawnian Motion

Geometric Brownian motion (GBM), which is a Markov process, meaning that the stock price follows a random walk and is consistent with the weak form of the efficient market hypothesis (EMH): past price information is already incorporated, and the next price movement is "conditionally independent" of past price movements.

Random walk theory proclaims that stocks take a random and unpredictable path that makes all methods of predicting stock prices futile in the long run.

The formula for GBM is

$$\frac{\Delta S}{S} = \mu \Delta t + \sigma \epsilon \sqrt{\Delta t} \tag{1}$$

where

- S=The stock price
- δS=The change in stock price
- μ=The expected return
- σ=The standard deviation of returns
- ϵ=The random variable
- δt=The elapsed time period

Rearranging the formula to solve just for the change in stock price, it's possible to see that the change in stock price is the stock price "S" multiplied by the two terms found inside the parenthesis below:

$$\Delta S = S \cdot (\mu \Delta t + \sigma \epsilon \sqrt{\Delta t}) \tag{2}$$

The first term is a "drift" and the second term is a "shock." For each time period, the model assumes the price will "drift" up by the expected return. But the drift will be shocked (added or subtracted) by a random shock. The random shock will be the standard deviation "s" multiplied by a random number "e." This is simply a way of scaling the standard deviation.

That is the essence of GBM, as illustrated in Figure 1. The stock price follows a series of steps, where each step is a drift plus or minus a random shock (itself a function of the stock's standard deviation).

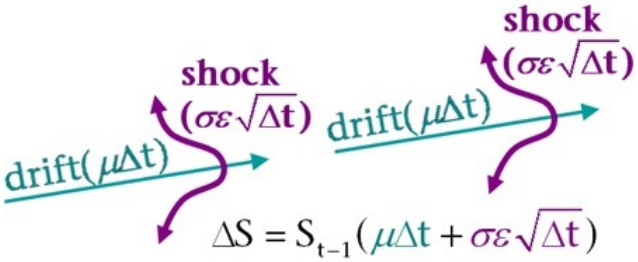

Fig. 1. Brownian motion

C. Value at Risk

Value at risk (VaR) is a statistic that measures and quantifies the level of financial risk within a firm, portfolio or position over a specific time frame.

This metric is most commonly used by investment and commercial banks to determine the extent and occurrence ratio of potential losses in their institutional portfolios.

Investment banks commonly apply VaR modeling to firm-wide risk due to the potential for independent trading desks to unintentionally expose the firm to highly correlated assets.

VaR modeling determines the potential for loss in the entity being assessed and the probability of occurrence for the defined loss. One measures VaR by assessing the amount of potential loss, the probability of occurrence for the amount of loss, and the timeframe.

Using a firm-wide VaR assessment allows for the determination of the cumulative risks from aggregated positions held by different trading desks and departments within the institution. Using the data provided by VaR modeling, financial institutions can determine whether they have sufficient capital reserves in place to cover losses or whether higher-than-acceptable risks require them to reduce concentrated holdings.

Mathematical definition The VaR of X at the confidence level $\alpha \in (0,1)$ is the smallest number y such that the probability that $Y := -X$ does not exceed y is at least $1-\alpha$.

$$\mathrm{VaR}_\alpha(X) = -\inf\{x \in \mathbb{R} : F_X(x) > \alpha\} = F_Y^{-1}(1-\alpha)$$

Risk managers typically assume that some fraction of the bad events will have undefined losses, either because markets are closed or illiquid, or because the entity bearing the loss breaks apart or loses the ability to compute accounts.

3. IMPLEMENTATION

A Monte Carlo simulation is a method that allows for the generation of future potential outcomes of a given event.

In this implementation I'll model the **price pattern of a given stock or portfolio of assets a predefined amount of days into the future**.

Monte Carlo simulations aren't just used to generate future stock prices, they are often used to estimate risk. For example, **Value at Risk** is often calculated using a Monte Carlo method, where I would be using theoretical future data rather than historical data. I included a method in the class I will build below to estimate Value at Risk over a given confidence interval with our generated stock prices.

The key takeaway from Monte Carlo simulations is the fact that there is some sort of random variable involved. The stock market is a perfect application of a model that uses a type of Monte Carlo simulation due to the level of statistical noise within the markets. I am trying to model the probability of different outcomes, simple as that.

In the script I wrote, the intention was to design a class that could use different Monte Carlo methods, and different graphical outputs of the simulation with descriptive statistics as well. Note I can use a single ticker or portfolio of assets in the simulation. I also have multiple models I can use.

- Model 1 – Simulation using *daily volatility*:

  Volatility is a statistical measure of the dispersion of returns for a given security or market index. In most cases, the higher the volatility, the riskier the security.

  The random "shock" element here will be determined by the historical volatility of a stock over a given timeframe.

- Model 2 – Simulation using *Brownian Motion*:

  Here, I are assuming the stocks will "drift" a certain amount over time. I can calculate this with the average daily return in our series as well as the variance of the returns.

Armed with a model specification, I then proceed to run random trials.

To start, I need to import the various dependencies that will allow for this script to run as well as creating the class and initial variables I will need. In this case, the variables are the start and end date for the time series data for the asset or portfolio of assets. Besides the classical NumPy and Pandas, we will need "norm" from SciPy and some specific Matplotlib features.

```
import pandas_datareader.data as web
import pandas as pd
import datetime
import numpy as np
import math
from matplotlib import style
import matplotlib.pyplot as plt
from scipy.stats import norm

class monte_carlo:
    def __init__(self, start, end):
        self.start = start
        self.end = end
```

Next, I need to obtain our data. I can do this using pandas' remote data access feature which uses *Yahoo Finance* APIs to download the values for the requested time frame.

I will make two methods within our class, one for a single asset, and one for a portfolio of assets.

```
    def get_asset(self, symbol):
        #Dates
        start = self.start
        end = self.end

        prices = web.DataReader(symbol, 'yahoo', start, end)['Close']
        returns = prices.pct_change()

        self.returns = returns
        self.prices = prices

    def get_portfolio(self, symbols, weights):
        start = self.start
        end = self.end

        #Get Price Data
        df = web.DataReader(symbols, 'yahoo',start, end)['Close']
        #Percent Change
        returns = df.pct_change()
        returns += 1

        #Define dollar amount in each asset
```

```
        port_val = returns * weights
        port_val['Portfolio Value'] = port_val.sum(axis=1)

        #Portfolio Dollar Values
        prices = port_val['Portfolio Value']

        #Portfolio Returns
        returns = port_val['Portfolio Value'].pct_change()
        returns = returns.replace([np.inf, -np.inf], np.nan)

        self.returns = returns
        self.prices = prices
```

The datas are taken from Yahoo Finance, the company that I'll use for this analysis is Apple (AAPL). The time frame under consideration reflects the past 6 months. Figure 2 shows that Apple's price has been gradually growing during the time frame.

Fig. 2. Apple stock prices

Figure 3 shows the plot of the log returns of AAPL. They are normally distributed and have a stable mean.

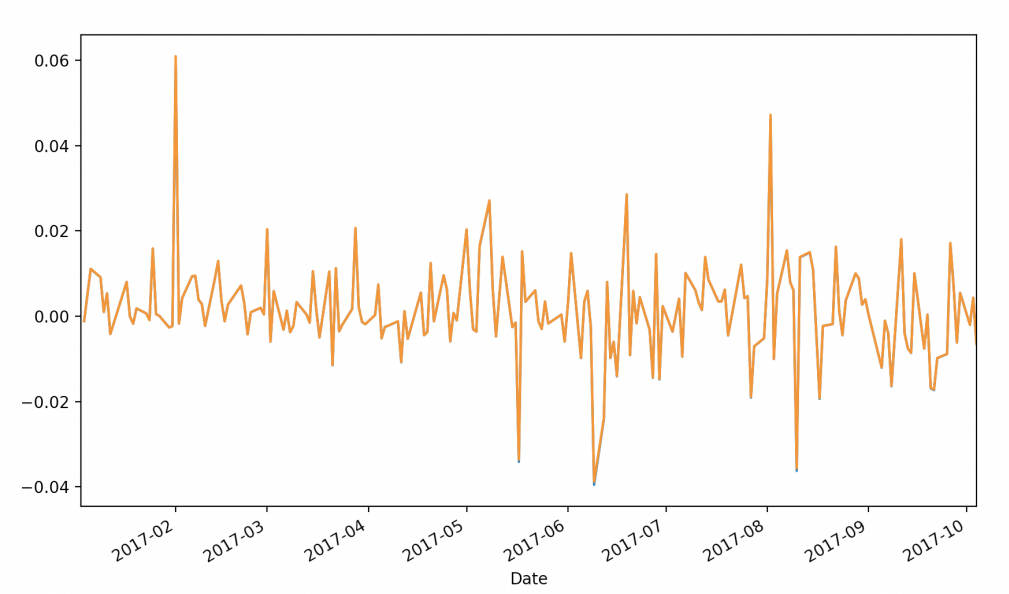

Fig. 3. Apple stock prices logarithmic scale

Next, I can create two types of simulations here. The first one is a basic simulation that only uses the asset's daily volatility.

```
    def monte_carlo_sim(self, num_simulations, predicted_days):
        returns = self.returns
        prices = self.prices

        last_price = prices[-1]

        simulation_df = pd.DataFrame()

        #Create Each Simulation as a Column in df
        for x in range(num_simulations):
            count = 0
            daily_vol = returns.std()

            price_series = []

            #Append Start Value
            price = last_price * (1 + np.random.normal(0, daily_vol))
            price_series.append(price)

            #Series for Preditcted Days
            for i in range(predicted_days):
                if count == 251:
                    break
                price = price_series[count] * (1 + np.random.normal(0, daily_vol))
                price_series.append(price)
                count += 1

```

```
77              simulation_df[x] = price_series
78              self.simulation_df = simulation_df
79              self.predicted_days = predicted_days
```

The second simulation will be using the concept of Geometric Brownian Motion.

I compute the drift component, the best approximation of future rates of return of the stock. The formula to use here will be `avg_daily_ret`, which equals the average log return, minus half its variance.

Next, I create a variable, called `daily_vol`, and I assign to it the standard deviation of returns. The Brownian motion comprises the sum of the drift and a variance adjusted by "E" to the power of "R", so we will use this block in the second part of the expression. $r = avg\_daily\_ret + (daily\_vol) * e^r$

Until now, we obtained the drift and standard deviation values we will need for the calculation of daily returns.

The second component of the Brownian motion is a random variable, z, a number corresponding to the distance between the mean and the events, expressed as the number of standard deviations.

SciPy's norm dot PPF allows us to obtain this result.

All I have to do is create a price list. Each price must equal the product of the price observed the previous day and the simulated daily return. Therefore, once we obtain the price in day T, we can estimate the expected stock price we will have in day T plus 1.

In the parentheses on line 112, we will have the value of the drift and the product of the standard deviation and the random component, created with the help of the "norm" module. Its percentage value was generated with NumPy's "rand" function, using "time intervals" and "iterations" specifying the dimensions of the array filled with values from 0 to 1.

To make credible predictions about the future, the first stock price in our list must be the last one in our data set. It is the current market price.

```
81      def brownian_motion(self, num_simulations, predicted_days):
82          returns = self.returns
83          prices = self.prices
84
85          last_price = prices[-1]
86
87          #Note we are assuming drift here
88          simulation_df = pd.DataFrame()
89
90          #Create Each Simulation as a Column in df
91          for x in range(num_simulations):
92
93              #Inputs
94              count = 0
95              avg_daily_ret = returns.mean()
96              variance = returns.var()
97
98              daily_vol = returns.std()
99              daily_drift = avg_daily_ret - (variance/2)
100             drift = daily_drift - 0.5 * daily_vol ** 2
101
102             #Append Start Value
103             prices = []
104
```

```
            shock = drift + daily_vol * np.random.normal()
            last_price * math.exp(shock)
            prices.append(last_price)

            for i in range(predicted_days):
                if count == 251:
                    break
                shock = drift + daily_vol * np.random.normal()
                price = prices[count] * math.exp(shock)
                prices.append(price)


                count += 1
            simulation_df[x] = prices
            self.simulation_df = simulation_df
            self.predicted_days = predicted_days
```

The simulation produced a distribution of hypothetical future outcomes. Next, I want to visualize the simulations so I can create a line graph and histogram to visualize the data with Matplotlib.

```
    def line_graph(self):
        prices = self.prices
        predicted_days = self.predicted_days
        simulation_df = self.simulation_df

        last_price = prices[-1]
        fig = plt.figure()
        style.use('bmh')

        title = "Monte Carlo Simulation: " + str(predicted_days) + " Days"
        plt.plot(simulation_df)
        fig.suptitle(title,fontsize=18, fontweight='bold')
        plt.xlabel('Day')
        plt.ylabel('Price ($USD)')
        plt.grid(True,color='grey')
        plt.axhline(y=last_price, color='r', linestyle='-')
        plt.show()
```

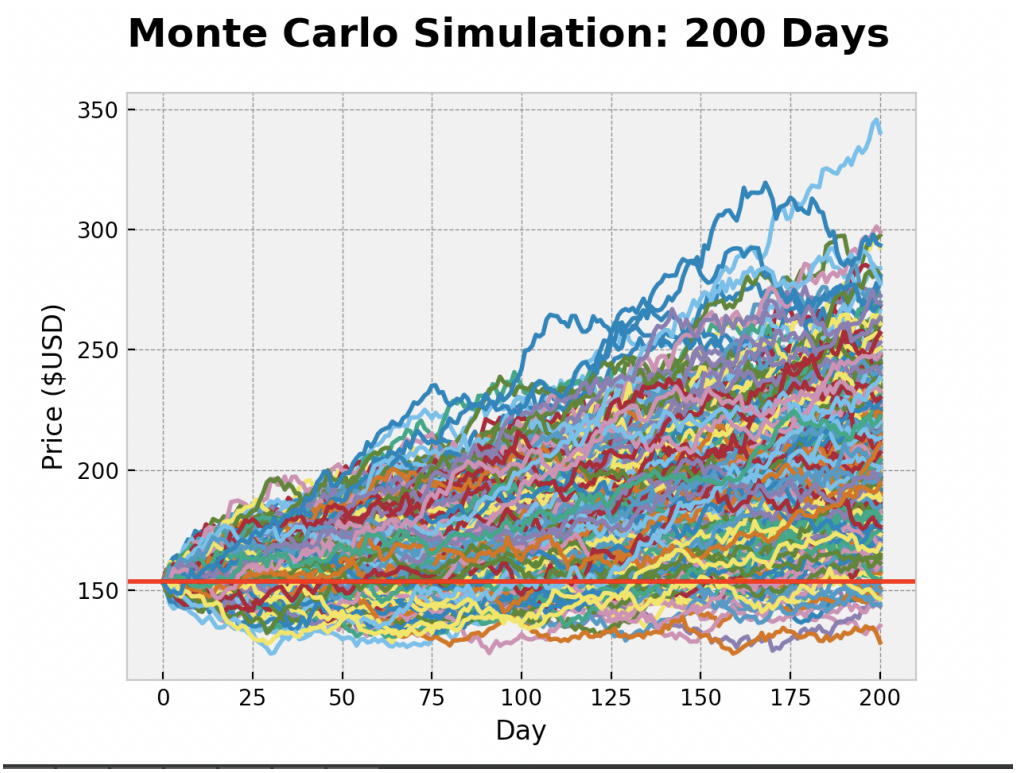

Fig. 4. Linechart

```
141     def histogram(self):
142         simulation_df = self.simulation_df
143 
144         ser = simulation_df.iloc[-1, :]
145         x = ser
146         mu = ser.mean()
147         sigma = ser.std()
148 
149         num_bins = 20
150         # the histogram of the data
151         n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='blue',
             ↪  alpha=0.5, edgecolor='white', linewidth=1.2)
152 
153         # add a 'best fit' line
154         y = norm.pdf(bins, mu, sigma)
155         plt.plot(bins, y, 'r--')
156         plt.xlabel('Price')
157         plt.ylabel('Probability')
158         plt.title(r'Histogram of Speculated Stock Prices', fontsize=18,
             ↪  fontweight='bold')
159 
160         # Tweak spacing to prevent clipping of ylabel
161         plt.subplots_adjust(left=0.15)
162         plt.show()
```

The GBM model assumes normality; price returns are normally distributed with expected return

(mean) "m" and standard deviation "s."

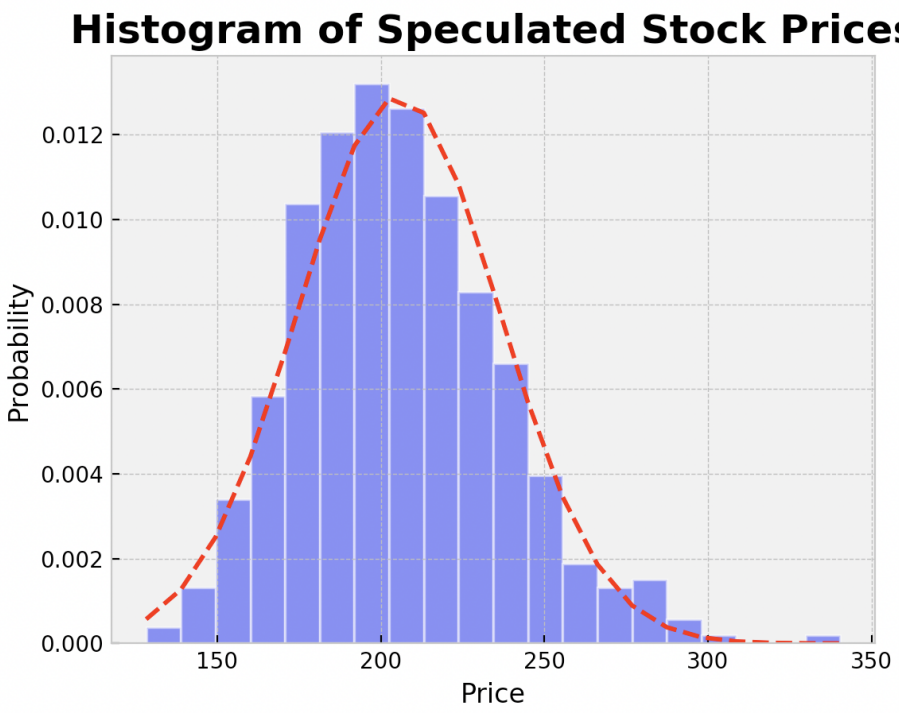

Fig. 5. Histogram

I can also calculate our theoretical *Value at Risk* with the given portfolio and simulation outputs. The Value at Risk is simply the difference between the current price and specified price at a given confidence interval.

```
164     def VaR(self):
165         simulation_df = self.simulation_df
166         prices = self.prices
167 
168         last_price = prices[-1]
169 
170         price_array = simulation_df.iloc[-1, :]
171         price_array = sorted(price_array, key=int)
172         var =  np.percentile(price_array, 1)
173 
174         val_at_risk = last_price - var
175         print ("Value at Risk: ", val_at_risk)
176 
177         #Histogram
178         fit = norm.pdf(price_array, np.mean(price_array), np.std(price_array))
179         plt.plot(price_array,fit,'-o')
180         plt.hist(price_array,normed=True, edgecolor='white', linewidth=1.2)
181         plt.xlabel('Price')
182         plt.ylabel('Probability')
183         plt.title(r'Histogram of Speculated Stock Prices', fontsize=18,
          ↪  fontweight='bold')
```

```
        plt.axvline(x=var, color='r', linestyle='--', label='Price at Confidence
         ↪  Interval: ' + str(round(var, 2)))
        plt.axvline(x=last_price, color='k', linestyle='--', label = 'Current Stock
         ↪  Price: ' + str(round(last_price, 2)))
        plt.legend(loc="upper right")
        plt.show()
```

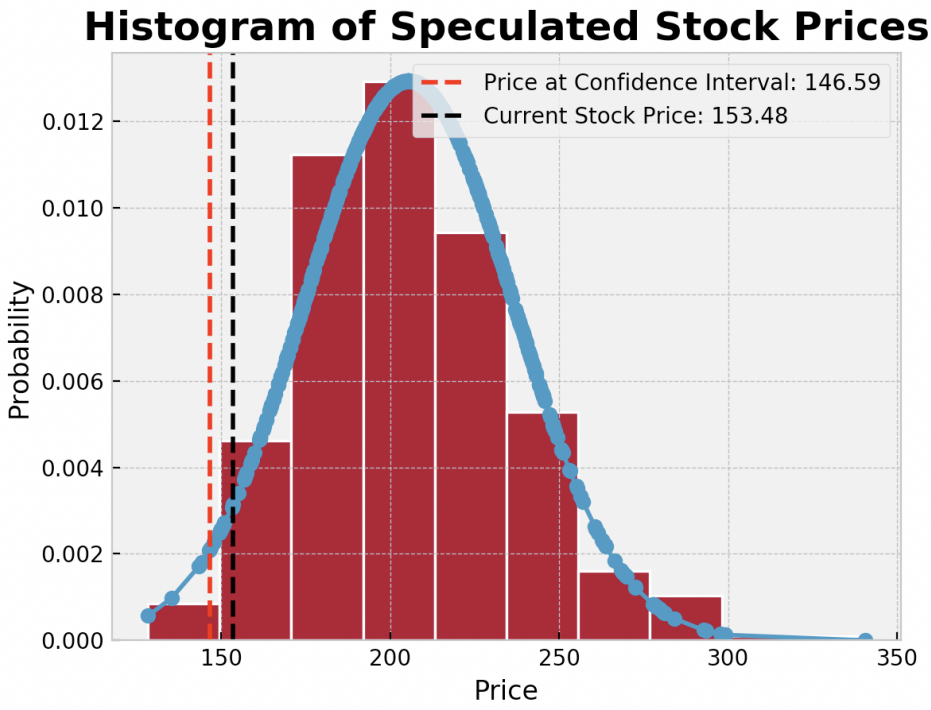

Fig. 6. Value at Risk

Next, I want to analyze the outputted simulations.

```
    def key_stats(self):
        simulation_df = self.simulation_df

        print ('#------------------Simulation Stats------------------#')
        count = 1
        for column in simulation_df:
            print ("Simulation", count, "Mean Price: ",
             ↪  simulation_df[column].mean())
            print( "Simulation", count, "Median Price: ",
             ↪  simulation_df[column].median())
            count += 1

        print ('\n')

        print ('#---------------------Last Price Stats--------------------#')
        print ("Mean Price: ", np.mean(simulation_df.iloc[-1,:]))
```

```
203         print ("Maximum Price: ",np.max(simulation_df.iloc[-1,:]))
204         print ("Minimum Price: ", np.min(simulation_df.iloc[-1,:]))
205         print ("Standard Deviation: ",np.std(simulation_df.iloc[-1,:]))
206         print ('\n')
207
208         print ('#----------------------Descriptive Stats-------------------#')
209         price_array = simulation_df.iloc[-1, :]
210         print (price_array.describe())
211
212         print ('\n')
213
214         print ('#--------------Annual Expected Returns for Trials-----------#')
215         count = 1
216         future_returns = simulation_df.pct_change()
217         for column in future_returns:
218             print ("Simulation", count, "Annual Expected Return",
                ↪  "{0:.2f}%".format((future_returns[column].mean() * 252) * 100))
219             print ("Simulation", count, "Total Return",
                ↪  "{0:.2f}%".format((future_returns[column].iloc[1] /
                ↪  future_returns[column].iloc[-1] - 1) * 100))
220             count += 1
221
222         print ('\n')
223
224         #Create Column For Average Daily Price Across All Trials
225         simulation_df['Average'] = simulation_df.mean(axis=1)
226         ser = simulation_df['Average']
227
228         print
            ↪  ('#----------------------Percentiles-------------------------------#')
229         percentile_list = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70,
            ↪  75, 80, 85, 90, 95]
230         for per in percentile_list:
231             print( "{}th Percentile: ".format(per), np.percentile(price_array,
                ↪  per))
232
233         print ('\n')
234
235         print ('#-----------------Calculate
            ↪  Probabilities-------------------------#')
236         print ( "Probability price is between 30 and 40: ",  "{0:.2f}%".format(
237             (float(len( price_array[(price_array > 30) & (price_array< 40)] )) /
                ↪  float(len(price_array)) * 100)
238             ))
239         print ("Probability price is > 45: ",
            ↪  "{0:.2f}%".format((float(len(price_array[price_array > 45])) /
            ↪  float(len(price_array)))* 100))
```

Finally, I run the script with 500 simulations and 200 predicted days.

```
242 if __name__== "__main__":
243
244     start = datetime.datetime(2017, 1, 3)
245     end = datetime.datetime(2017, 10, 4)
246
247     simulation = monte_carlo(start, end)
248     simulation.get_asset('AAPL') #Apple, single asset
249
250     #symbols = ['AAPL', 'KO', 'HD', 'PM'] #multiple assets
251     #weights = [1000,1000,2000,3000]
252     #simulation.get_portfolio(symbols, weights)
253
254     simulation.brownian_motion(num_simulations=500, predicted_days=200)
255     simulation.line_graph()
256     simulation.histogram()
257     simulation.VaR()
258     simulation.key_stats()
```

Below are the statistics outputted for a simulation consisting of 5 trials with Apple Stock

```
#------------------Simulation Stats------------------#
Simulation 1 Mean Price:  179.04436498007504
Simulation 1 Median Price:  177.6325383370973
Simulation 2 Mean Price:  167.35907643895987
Simulation 2 Median Price:  167.31656366950844
Simulation 3 Mean Price:  169.61529813934007
Simulation 3 Median Price:  168.3357936261102
Simulation 4 Mean Price:  165.1215561959754
Simulation 4 Median Price:  165.69688032235064
Simulation 5 Mean Price:  203.2993677588893
Simulation 5 Median Price:  202.26104405249976


#----------------------Last Price Stats--------------------#
Mean Price:  203.32930656373074
Maximum Price:  272.17643190698124
Minimum Price:  155.3227447840619
Standard Deviation:  38.37743160363899


#----------------------Descriptive Stats-------------------#
count      5.000000
mean     203.329307
std       42.907273
min      155.322745
25%      186.220832
50%      198.307672
75%      204.618852
max      272.176432
Name: 200, dtype: float64
```

```
#--------------Annual Expected Returns for Trials-----------#
Simulation 1 Annual Expected Return 37.47%
Simulation 1 Total Return -36.35%
Simulation 2 Annual Expected Return 25.81%
Simulation 2 Total Return 42.91%
Simulation 3 Annual Expected Return 33.67%
Simulation 3 Total Return -203.51%
Simulation 4 Annual Expected Return 2.95%
Simulation 4 Total Return -137.78%
Simulation 5 Annual Expected Return 73.73%
Simulation 5 Total Return -112.50%


#----------------------Percentiles---------------------------------#
5th Percentile:  161.50236221237864
15th Percentile:  173.86159706901205
25th Percentile:  186.22083192564548
35th Percentile:  191.05556813162104
45th Percentile:  195.8903043375966
50th Percentile:  198.30767244058438
55th Percentile:  199.56990830474368
60th Percentile:  200.83214416890291
65th Percentile:  202.0943800330622
70th Percentile:  203.35661589722145
75th Percentile:  204.61885176138074
80th Percentile:  218.13036779050083
85th Percentile:  231.64188381962094
90th Percentile:  245.15339984874103
95th Percentile:  258.6649158778611


#-----------------Calculate Probabilities-------------------------#
Probability price is between 30 and 40:  0.00%
Probability price is > 45:  100.00%
```