Monte Carlo Simulation

To evaluate a decision with a Monte Carlo simulation, an analyst

- Identifies parameters that are not known with a high degree of certainty and
- Treats these parameters as **random**, or uncertain, **variables**.

The values for the random variables are randomly generated from the specified probability distributions. The simulation model uses the randomly generated values of the random variables and the relationships between parameters and decisions to compute the corresponding values of an output.

Monte Carlo Simulation: An experiment that produces a distribution of output values that correspond to the randomly generated values of the uncertain input variables.

An analyst is often able to make decision recommendations for the controllable inputs that address not only the average output but also the variability of the output.

Sanotronics Risk Analysys (Monte Carlo Simulation)

A decision maker should be interested not only in

- The average, or expected, outcome,

but also in

- Information regarding the range of possible outcomes.

Objective:

To quantifying the likelihood and magnitude of an undesirable outcome.

The Problem

Sanotronics LLC is a start-up company that manufactures medical devices for use in hospital clinics. Their new device features an innovative design and has the potential to capture a substantial share of the market.

The Goal: Sanotronics would like an analysis of the first-year profit potential for the device.

Sanotronics has identified the key parameters in determining first-year profit:

1. Selling price per unit (p),
2. First-year administrative and advertising costs (c_a),
3. Direct labor cost per unit (c_i),
4. Parts cost per unit (c_p), and
5. First-year demand (d).

Sanotronics estimates with a high level of **certainty** that:

- The device's selling price will be $p = \$249$ per and
- The first-year administrative and advertising costs will total $c_a = \$1,000,000$

Sanotronics is **not certain** about the values for the:

- Cost of direct labor (c_i) - best estimate \$45 per unit
- Cost of parts (c_p) - best estimate \$90 per unit
- The first-year demand (d). - best estimate 15,000 units for the first-year demand

Profit Formula

$$\text{Profit} = (p - c_i - c_p) \times d - c_a$$

Base-Case Scenario

In [3]:

```
# mathematicians'
p = 249.0
ca = 1000000.0
ci = 45
cp = 90
d = 15000
profit = (p-ci-cp)*d-ca
print "Profit $", profit
```

```
Profit $ 710000.0
```

In [4]:

```
# computer programmers'
sellingPricePerUnit = 249.0
administrativeAdvCost = 1000000.0
directLaborCostPerUnit = 45
partsCostPerUnit = 90
demand = 15000
profit = (sellingPricePerUnit-directLaborCostPerUnit-partsCostPerUnit)*demand-administr
ativeAdvCost
print "Profit $", profit
```

```
Profit $ 710000.0
```

Sanotronics is aware that the values of:

- Direct labor cost per unit (c_i),
- Parts cost per unit (c_p), and
- First-year demand (d).

are uncertain.

A **what-if analysis** involves considering alternative values for the random variables c_i, c_p, and d

What If

- Cost of direct labor (c_i) ranges between \$43 and \$47 per unit,
- Cost of parts (c_p) ranges between \$80 and \$100 per unit, and
- The first-year demand (d) ranges between 0 to 30,000 units?

Worst-Case Scenario

In [5]:

```
p = 249.0 # fixed
ca = 1000000.0 # fixed
# worst-case values
ci = 47
cp = 100
d = 0
worstCaseScenarioProfit = (p-ci-cp)*d-ca
print "Worst Case Scenario Profit $", worstCaseScenarioProfit
```

```
Worst Case Scenario Profit $ -1000000.0
```

Best-Case Scenario

write the code to compute the profit in the best-case scenario

In [6]:

```
# best-case values
ci = 43
cp = 80
d = 30000
bestCaseScenarioProfit = (p-ci-cp)*d-ca
print "Best Case Scenario Profit $", bestCaseScenarioProfit
```

```
Best Case Scenario Profit $ 2780000.0
```

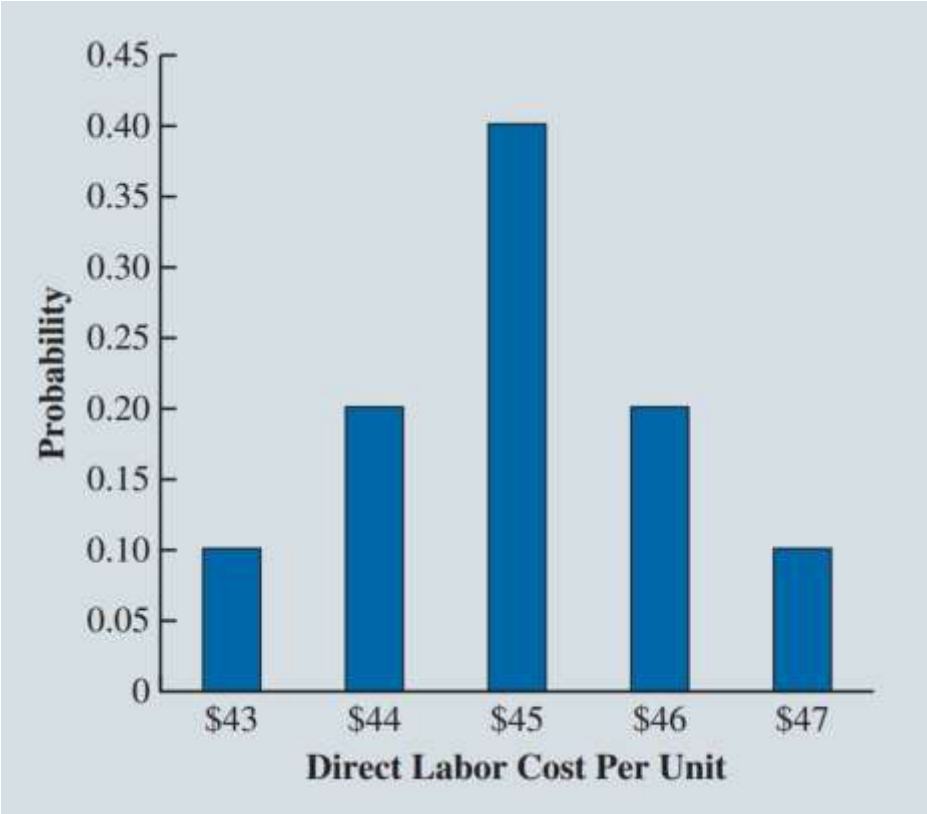
Use of Probability Distributions to Represent Random Variables

- Monte Carlo simulation randomly generates values for the random variables so that the values used reflect what we might observe in practice
- A probability distribution describes the possible values of a random variable and the relative likelihood of the random variable taking on these values.

Random Number Intervals for Generating Value of Direct Labor Cost per Unit

Sanotronics believes that:

- Direct labor cost will range from $43 to \$47 per unit and is described by the **discrete probability distribution shown in the figure.**



We see that there is:

- 0.1 probability that the direct labor cost will be $43 per unit,
- 0.2 probability that the direct labor cost will be $44 per unit,
- and so on.

The highest probability, 0.4, is associated with a direct labor cost of $45 per unit. Because we have assumed that the direct labor cost per unit is best described by a discrete probability distribution, the direct labor cost per unit can take on only the values of \$43, $44, \$45, $46, or \$47.

Table 11.1 illustrates the process of partitioning the interval from 0 to 1 into subintervals so that the probability of generating a random number in a subinterval is equal to the probability of the corresponding direct labor cost.

The interval of random numbers from 0 up to but not including 0.1, [0, 0.1), is associated with a direct labor cost of $43; the interval of random numbers from 0.1 up to but not including 0.3, [0.1, 0.3), is associated with a direct labor cost of \$44, and so on.

In [7]:

```
import random
def directLaborCost(rnd):
    if rnd < 0.1:
        return 43.0
    elif rnd <0.3:
        return 44.0
    elif rnd < 0.7:
        return 45.0
    elif rnd < 0.9:
        return 46.0
    else:
        return 47.0
```

Generating Values for Parts Cost per Unit

Sanotronics is confident that the parts cost will be between $80 and \$100 per unit but is unsure as to whether any particular values between $80 and 100 are more likely than others. Therefore, Sanotronics

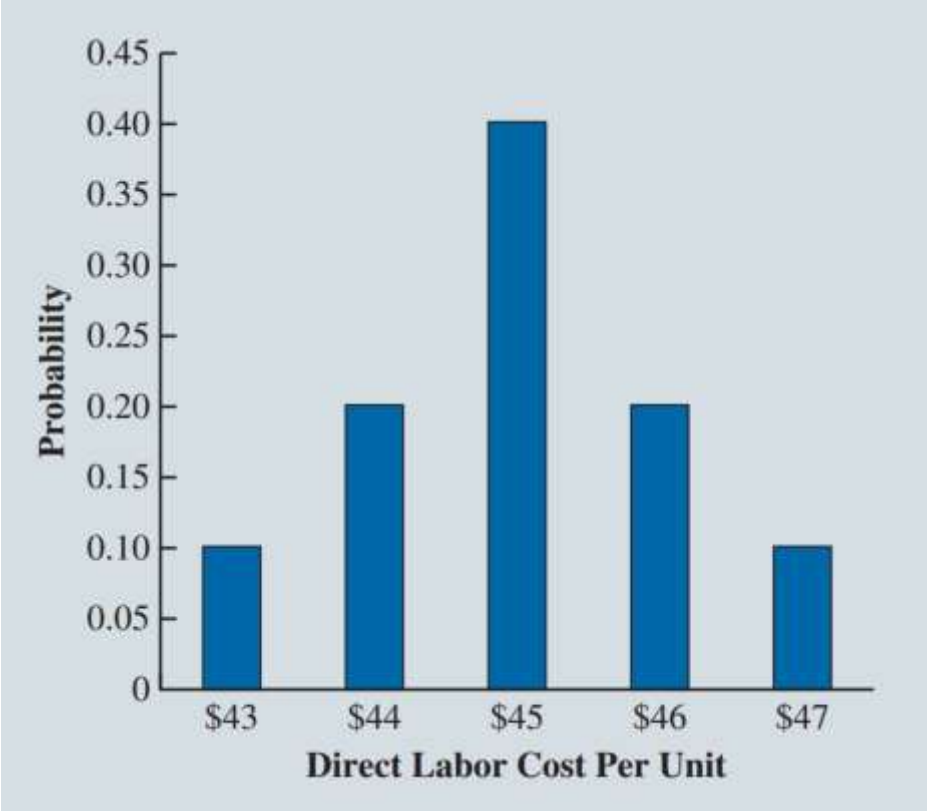
decides to describe the uncertainty in parts cost with a uniform probability distribution, as shown in the figure.

Costs per unit between $80 and \$100 are equally likely. A uniform probability distribution is an example of a continuous probability distribution, which means that the parts cost can take on any value between $80 and \$100.

Value of uniform random variable = lower bound (upper bound 2 3lower bound) RANDOM()

For example, suppose that a random number of 0.4576 is generated by the RAND function. As illustrated by Figure 11.5, the value for the parts cost would be:

Parts cost $= 180 + 20 \times 0.4576 = 80 + 9.15 = 89.15$ per unit.



Suppose that a random number of 0.5842 is generated on the next trial. The value for the parts cost would be:

Parts cost $= 180 + 20 \times 0.5842 = 80 + 11.68 = 91.68$ per unit.

In [8]:

```
import random
def partsCost(rnd):
    return 80 + 20 * rnd
```

Generating Values for Demand

Sanotronics believes that first-year demand is described by the normal probability distribution shown in the figure below. The mean (μ) of first-year demand is 15,000 units. The standard deviation (σ) of 4,500 units describes the variability in the first-year demand.

The normal probability distribution is a continuous probability distribution in which any value is possible, but values extremely larger or smaller than the mean are increasingly unlikely.

To generate a value for a random variable characterized by a normal distribution with a specified mean and standard deviation, the following formula is used:

Value of normal random variable = f(RANDOM(), mean, standard deviation)

We use the Percent Point Function (PPF) for the function f, which is the inverse of the Cumulative Distribution Function (CDF) to generate the demand value bound by $\mu = 15,000$ and $\sigma = 4,500$

Suppose that the random number of 0.6026 is produced by the *RANDOM()* function; applying the above equation then results in

In [9]:

```
import scipy.stats as sct
print 'Demand = {:,}'.format(int(round(sct.norm.ppf(0.6026, loc=15000, scale=4500)))),
'units'
```

```
Demand = 16,170 units
```

Now suppose that the random number produced by the *RANDNDOM()* function is 0.3551. Applying the same equation then results in

write the code

In [10]:

```
import scipy.stats as sct
print 'Demand = {:,} units'.format(int(round(sct.norm.ppf(0.3551, loc=15000, scale=4500
))))
```

```
Demand = 13,328 units
```

define a function to calculate the Demand

In [11]:

```
import scipy.stats as sct
def demandFunction(rnd):
    val = int(round(sct.norm.ppf(rnd, loc=15000, scale=4500)))
    if (val < 0):
        return 0 # to avoid -ve demand.
    if (val > 30000):
        return 30000 # cap the value to 30000
    return val
```

define the function to calculate the profit using the three functions defined above

In [12]:

```
def profit(dlc, pc, d): # dlc=direct labor cost, pc=parts cost, d = demand
    sppu = 249.0 #selling price per unit
    aac = 1000000.0 # admin and advertising cost
    return ((sppu - dlc - pc)*d)-aac
```

Executing Simulation Trials

Each trial in the simulation involves randomly generating values for the random variables (direct labor cost, parts cost, and first-year demand) and computing profit.

Let us make 10,000 simulation trials

In [13]:

```
sim = []
for i in range(1,10000): # for each trial
    rnd = random.random() # get a random number
    dlc = directLaborCost(rnd) # compute the direct labor cost
    pc = partsCost(rnd) # compute the parts cost
    d = demandFunction(rnd) # compute the demand
    p = profit(dlc, pc, d) # compute the profit for the CURRENT trial
    sim.append([i, dlc, pc, d, p]) # save the data in a list
```

Inspect a single sample in the simulation

In [14]:

```
sample = sim[3]
print "Sample # ", sample[0]
print "Labor Cost $", "{:.2f}".format(sample[1])
print "Parts Cost $", "{:.2f}".format(sample[2])
print "Demand ", sample[3]
print "Profit $", "{:.2f}".format(sample [4])
```

```
Sample #  4
Labor Cost $ 46.00
Parts Cost $ 97.12
Demand  19785
Profit $ 1094756.56
```

write the code to inspect sample #281

In [15]:

```
sample = sim[281]
print "Sample # ", sample[0]
print "Labor Cost $", "{:.2f}".format(sample[1])
print "Parts Cost $", "{:.2f}".format(sample[2])
print "Demand ", sample[3]
print "Profit $", "{:.2f}".format(sample [4])
```

```
Sample #  282
Labor Cost $ 45.00
Parts Cost $ 89.07
Demand  14472
Profit $ 663326.33
```

Measuring and Analyzing Simulation Output

For the collection of simulation trials, it is helpful to compute descriptive statistics such as sample count, minimum sample value, maximum sample value, sample mean, sample standard deviation, sample proportion, and sample standard error of the proportion.

Let us look at the data first

Let us use **pandas** library to get our trials list organized

In [16]:

```
import numpy as np
from pandas import DataFrame as df
import pandas as pd

# pandas number formatting configuration
pd.options.display.float_format = '{:,.2f}'.format

dd = np.array(sim)
# convert our list into a dataframe
data = df(data = {'sn': dd[:,0],
                  'Labor Cost': dd[:,1],
                  'Parts Cost': dd[:,2],
                  'Demand': dd[:,3],
                  'Profit': dd[:,4]
                 })
```

In [17]:

```
# configure our dataframe to format the values accordingly
data['Profit'] = data['Profit'].map('${:.2f}'.format)
data['Parts Cost'] = data['Parts Cost'].map('${:.2f}'.format)
data['Labor Cost'] = data['Labor Cost'].map('${:.2f}'.format)
data['Demand'] = data['Demand'].map('{:.0f}'.format)
data['sn'] = data['sn'].map('{:.0f}'.format)
print data.head()
```

```
  Demand Labor Cost Parts Cost       Profit sn
0  24203     $47.00     $99.59  $1478593.75  1
1  15974     $45.00     $91.71   $793653.83  2
2  11664     $44.00     $84.59   $404515.75  3
3  19785     $46.00     $97.12  $1094756.56  4
4  11170     $44.00     $83.95   $352155.06  5
```

Lets compute the compute descriptive statistics

In [18]:

```
# recreate the dataframe
# WITHOUT formatting
data = df(data = {
                  'Labor Cost': dd[:,1],
                  'Parts Cost': dd[:,2],
                  'Demand': dd[:,3],
                  'Profit': dd[:,4]
                 })
data.describe()
```

Out[18]:

	Demand	Labor Cost	Parts Cost	Profit
count	9,999.00	9,999.00	9,999.00	9,999.00
mean	15,005.48	45.00	90.01	680,330.15
std	4,498.68	1.10	5.78	414,241.85
min	0.00	43.00	80.00	-1,000,000.00
25%	11,987.50	44.00	85.03	438,114.57
50%	14,960.00	45.00	89.93	706,497.46
75%	18,073.50	46.00	95.05	950,958.65
max	30,000.00	47.00	100.00	2,060,236.11

Compute the Standard Error of the Mean

The standard error of the sample mean is an estimate of how far the sample mean is likely to be from the population mean

In [19]:

```
data.sem()
```

Out[19]:

```
Demand          44.99
Labor Cost       0.01
Parts Cost       0.06
Profit       4,142.63
dtype: float64
```

Confidence Interval

The 95% confidence interval relates only to the confidence we have in the estimation of the mean profit, **not** the likelihood of an individual profit observation.

95% Confidence Interval on the Mean Profit Range

In [20]:

```
CI_lower = data.describe()['Profit']['mean'] - sct.norm.ppf(q = 0.975) * data.sem()['Profit']
CI_upper = data.describe()['Profit']['mean'] + sct.norm.ppf(q = 0.975) * data.sem()['Profit']
print 'lower: ${:,.2f}'.format(CI_lower), ' to upper: ${:,.2f}'.format(CI_upper)
```

```
lower: $672,210.75  to upper: $688,449.54
```

In [21]:

```
neg = data[data['Profit'] < 0]
neg.describe()
```

Out[21]:

	Demand	Labor Cost	Parts Cost	Profit
count	596.00	596.00	596.00	596.00
mean	6,041.55	43.00	80.60	-242,976.16
std	1,784.87	0.00	0.35	222,347.56
min	0.00	43.00	80.00	-1,000,000.00
25%	5,169.50	43.00	80.29	-350,138.28
50%	6,576.50	43.00	80.61	-175,387.68
75%	7,377.50	43.00	80.90	-77,094.81
max	8,009.00	43.00	81.20	-499.02

Compute The Standard Error (negative profit proportion)

$\sqrt{\bar{p}(1-\bar{p})/n}$, where $\bar{p}$ is the sample proportion of observations satisfying a criterion (profit less than $0 in this case) and n is the sample size.

In [22]:

```
import math
negative_profit_count = neg.describe()['Profit']['count']
print 'Number of Negative Profit Samples: {:.0f}'.format(negative_profit_count)
all_sample_count = data.describe()['Profit']['count']
print 'Total number of Samples: {:.0f}'.format(all_sample_count)
negative_profit_probability = negative_profit_count/all_sample_count
print 'P(Profit<0)=', negative_profit_probability

standard_error_of_proportion = math.sqrt(negative_profit_probability * (1-negative_profit_probability)/all_sample_count)
print 'Standard error of the proportion:', standard_error_of_proportion
```

```
Number of Negative Profit Samples: 596
Total number of Samples: 9999
P(Profit<0)= 0.059605960596059604
Standard error of the proportion: 0.00236767176625
```

95% Confidence Interval on the Probability of a Negative Profit Range

In [23]:

```
CI_neg_profit_lower = negative_profit_probability - sct.norm.ppf(q = 0.975) * standard_
error_of_proportion
CI_neg_profit_upper = negative_profit_probability + sct.norm.ppf(q = 0.975) * standard_
error_of_proportion
print 'Confidence Interval on P(Profit<0) -- lower: {:,.2f}'.format(CI_neg_profit_lower
), 'upper: {:,.2f}'.format(CI_neg_profit_upper)
```

```
Confidence Interval on P(Profit<0) -- lower: 0.05 upper: 0.06
```

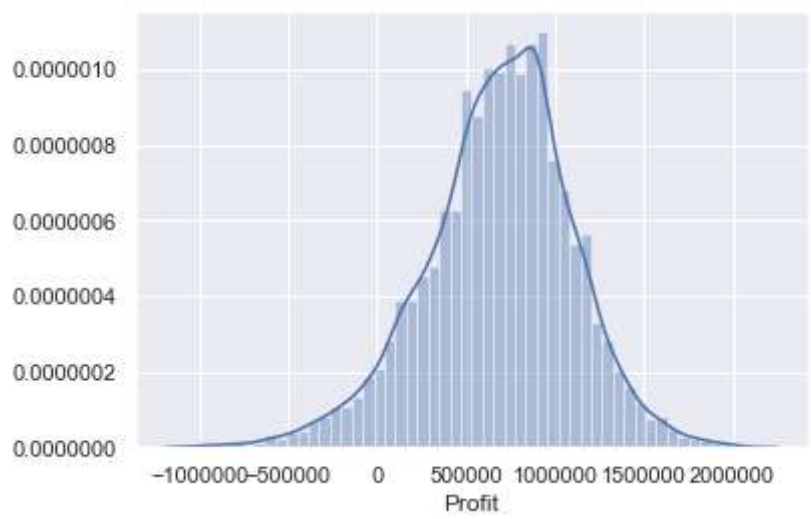
Visualise the Distribution of Profit

In [68]:

```
import seaborn as sns
sns.set(color_codes=True)
sns.distplot(data['Profit'])
```

Out[68]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xdbc3da0>
```



In [ ]: