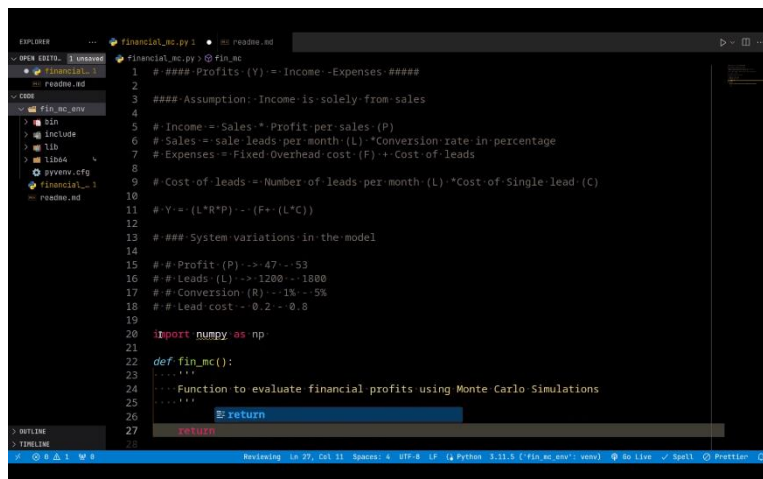


Advanced Business Decision Support Systems
Professor Deepu Philip
Department of Industrial Engineering and Management Engineering
Indian Institute of Technology, Kanpur
Professor Amandeep Singh
Imaginer Engineering Laboratory
Dr. Prabal Pratap Singh
Indian Institute of Technology, Kanpur
Lecture
Financial DSS using Monte Carlo Simulation (Part 1 of 2)

Hello everyone, I welcome you all to the lecture series on Advanced Business Theory and Support Systems. I am Prabal Pratap Singh from IIT Kanpur and we are discussing How to Develop the Latest Theme Support System.

In the previous week, we have seen how to create a complete Inventory Simulation Model Using Monte Carlo Simulation with the help of python to create a complete Web-Based Decision Support System. But, even the simplest of the inventory model can get trickier to develop and understand in python. So, in this week, we are trying to Develop a very Basic Financial Simulation Model which was already discussed by Professor Deepu in the week 2 of this lecture series. So, we are trying to Develop this Financial Simulation Model Using The Same Technologies and let us start Creating This System From Scratch.



```
1  @ ### Profits (Y) = Income - Expenses #####
2
3  ### Assumption: Income is solely from sales
4
5  # Income = Sales * Profit per sales (P)
6  # Sales = sale leads per month (L) * Conversion rate in percentage
7  # Expenses = Fixed Overhead cost (F) + Cost of leads
8
9  # Cost of leads = Number of leads per month (L) * Cost of Single lead (C)
10
11 # Y = (L * P) - (F + (L * C))
12
13 # ### System variations in the model
14
15 # # Profit (P) -> 47 - 53
16 # # Leads (L) -> 1200 - 1800
17 # # Conversion (R) -> 1% - 5%
18 # # Lead cost - 0.2 - 0.8
19
20 import numpy as np
21
22 def fin_mc():
23     """
24     Function to evaluate financial profits using Monte Carlo Simulations
25     """
26     return
27
28 return
```

I am currently in a blank workspace and today I am using a different operating system named Linux. So, we will be coding today into the environment of Linux. So, let us start our Visual Studio code. We can start this repository by creating a basic file named as financial_mc.py.

Since we are creating this complete simulation model, we should also create a readme file, so that other users can understand what this whole repository contains. Now, let us start building our financial models. So, to quickly recall what this financial model is, we were trying to calculate the profits for a particular month every

time we simulated this model. Let us write the complete mathematical model in comments in this file. So, we can write profits recall 'u' by the variable 'Y' as the difference between income - expense rate. So, this is the main thing.

Now, we need to first see what was the assumption in this model. So, the assumption was income is solely from A. Now, what income consists of? So, income included sales time profit per year. It was denoted by capital P. Sales are evaluated by using sales leads per month into conversion rate in percentage. Expenses include fixed overhead cost, it was denoted by capital F and cost of leads.

After that, cost of leads can be further calculated by using number of leads per month, it was denoted by L. Times cost of single leads which can be denoted by capital G. Now, the final calculation for this complete model was profits that were denoted by $Y = L * R * P - S + L$ to P. Now, this was the complete model.

But, this model can include various kinds of random variation. To make it more to the real world example, we have discussed that the system variation in this model included model, we assume that let us say, profit per year denoted by P can range from 47 to 53 using a uniform distribution.

The next thing that can vary leads denoted by L and it can vary from let us say, 1200 to 1800 in a period. The other thing would be conversion which can vary from 1 percent to 5 percent and the last thing which can vary were lead costs. It can vary from 0.2 to 0.8.

So, currently we are assuming that all these variations follow random uniform distribution and these things can vary using other distributions also. But for the sake of simplicity, we are trying to develop this model by using just uniform probability distribution. Further once the complete simulation model gets developed we can tweak the model to include any kind of random variation. This was the model.

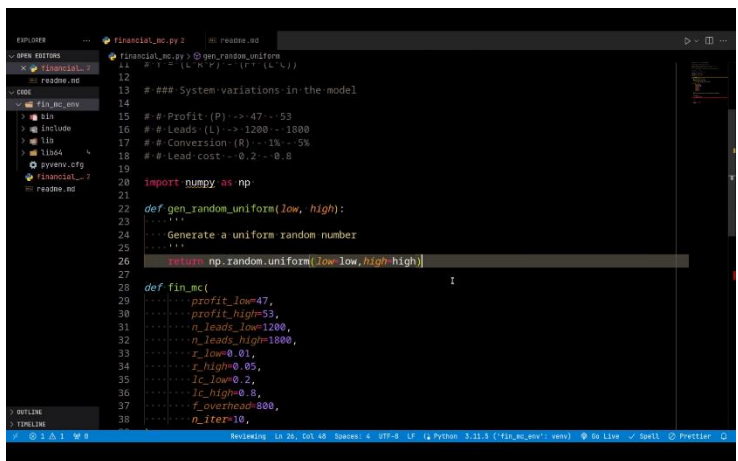
Now, we will start creating our system but before that we need to first create our environment. We can write python -venv. Let us say, this is the name of the python environment which we are creating today. So, now you can see that the environment is created here.

Now, in Linux, you can activate an environment by using source mc env bin activate. So, this dot (.) operator says that in the current directory find this folder name, inside that find the bin folder name and inside that there is an activate by run that. So, after running that we can see here that our environment variable python environment has been activated. Now, we can first install our NumPy because that is required for creating our simulation model and to utilize our probability distribution that is available in this NumPy library.

So, we can write python m pip install NumPy. So, now NumPy is available in our environment, so import NumPy as np. Now, before developing our model, we can decide whether we need to just create everything in this module or we can create this simulation model as a single module, and then create a block application in a different module, similar to what we have done in the earlier lecture. So, to do that, we need to create everything that we need to develop this financial model in a particular function.

So, we can write def, let us say, the name of the function is financial mc. Let us provide the dotting here which

can pay function to evaluate financial profit using Monte Carlo Simulation. Let us just create a blank return statement.



```
11 # T = (L * R * P) - (C + (L * LC))
12
13 ### System variations in the model
14
15 # # Profit (P) -> 47 - 53
16 # # Leads (L) -> 1200 - 1800
17 # # Conversion (R) - 1% - 5%
18 # # Lead cost - 0.2 - 0.8
19
20 import numpy as np
21
22 def gen_random_uniform(low, high):
23     """
24     Generate a uniform random number
25     """
26     return np.random.uniform(low=low, high=high)
27
28 def fin_mc(
29     profit_low=47,
30     profit_high=53,
31     n_leads_low=1200,
32     n_leads_high=1800,
33     r_low=0.01,
34     r_high=0.05,
35     lc_low=0.2,
36     lc_high=0.8,
37     f_overhead=800,
38     n_iter=10,
```

Now, to make this module executable, we should write name equals main and this module should execute this function. Now, the basic structure is complete, so we can start creating our model.

So, let us create a new variable sales_data as a blank list, so that everything that we calculate we will store in this list. Now, let us decide what are the different kinds of arguments that this function will need to create this financial model.

Since we know that the variation occurs in these variables only like profit, lead, conversion, lead cost, so let us create an argument with the name profit_low and let us say, the value is 47 by default, otherwise if the user wants to change it, then it will get overwritten. The next thing is profit_high which is 53 as a default in our case.

The next thing is the lead, so number of leads. The low value of lead 1200. Similarly, the number of leads high is 1800. Conversion rate low is 0.01 because it is actually percentage so we are creating it into fraction. And, conversion rate high is 0.05. The next thing is lead cost so let us write this as LC low, so that our argument name does not get bigger and we can initialize it to 0.2. And, the value LC high, LC is the lead cost, and the high value is 0.8.

The fixed overhead cost which will not change once the user initializes it and we are making it a default of 800. The last thing we need as an argument is the number of iterations or the number of simulations we need to run to find the average output of our profits. So, you can use an argument named as n_iter. And, let us say that this value is 10. By default we will run 10 simulations each time.

So, now we can put what happens in each simulation run. We need to perform some kind of operation every time a simulation gets executed, so we need to create a loop. To create a loop, we have already discussed that we can use a 'while loop' or 'for loop'. Here, we are using 'for loop' so we can write for 'i' in the range of 'n' iter. So, this 'for loop' will run 10 times for this by default.

Now, we can initialize our profits because profit here is something that can vary, so we need to create a random number. So, either we can generate a random number here or we can create a new function that can generate a random number each time. Since we are using a single probability distribution, that is uniform.

So, we can create a common function that can take a low and high value and return a generated value from the uniform probability distribution. So, we can write here to initialize the function and let us name the function as `gen_random_uniform`. And, this function will take two arguments `uni`, `low` and `high`.

Let us create a doc for this function which should inform the user that this function will generate a uniform random number. So, by creating this function, we do not need to write the same command every time to generate a new random number. So, we can write here `return np.random.uniform` and it should get two values `low` on the low argument provided in this function and `high`.

And, just look at the higher rate. Now, this function will return a single value by picking a value from the random uniform distribution. Now, we need to generate our property. So, let us say, in the first iteration we need to generate one value of profit.

So, to do that, we need to call our newly created function named as `gen random uniform` and let us provide the low value for this as `profit low` and high value as `profit high`. Similarly, we need to also generate the lead and we can write the same thing. Now, we need to change this low value here to `end lead flow` because that is what we are getting from the user from the arguments of this function and the high value is `'n'_lead_high`.

The next thing is conversion. So, we can again write $\text{gen_random_uniform low} = R_{\text{flow}}$ and $\text{high} = R_{\text{source}}$. The last thing we need to generate is the lead cost. So, we can write lead cost equal to $\text{gen_random_uniform low}$ equals lc_{low} and high equals lc_{high} .

Now, we have generated all the values that get randomly picked from a random uniform distribution. Now, to evaluate the overall profit for this period for the single simulation run, we can create a new variable named as overall profit and we can write the same formula which we discussed above which is $L * R * P - S + L * P$.

So, we can write here: $\text{lead} * \text{conversion} * \text{profit} - f_{\text{overhead}} + \text{leads} * \text{lead cost}$. Now, we have the overall profit. So, we can update our entities that we have evaluated in this iteration of the simulation into this sales data list.

So, we can write `sales_data.append`. The append function of a list will create a new item every time this loop will execute. So, we can create a dictionary inside the list at every location and we can write 'p' sim for capturing the value of the particular iteration in which we are saving this data. So, we can write $\text{sim} = i + 1$ because in python `i` will start from 0.

So, at the first iteration, this same value will hold 1, that means the data contained at the first location contains the iteration of the first simulation run. So, the next thing which we need to store is the profit which is available in the variable name profit. Similar, for other variables as well, like lead, conversion, lead_cost, overall profit.

Now, at every iteration, we can print a particular statement, so that once we are debugging, we can use these print statements. So, we can write here using an 'f' string that the overall profit for the period $i + 1$ is the overall profit. Now, whatever we need to calculate at every simulation run is completed by now. Now, we can also collect all the profits that gets calculated at every simulation run. So, we know now that this sales data list contains the detail of all the costs at every simulation run.

So, we can just output and fetch this overall profit value from the sales data dictionary, from the sales data list that contains the number of dictionaries. So, to do that, we can use our list comprehension which we learnt. So, we can write `all_profit` and to create a list comprehension, we first need to use open and close square brackets and start writing our for do.

So, we can write for sale in sale data. And now, what we need to do every time this loop runs is, we need to use the dictionary sale and the key overall profit. So, now what happened is, this all profits variable will contain a list of all the profits that gets evaluated using this formula for every simulation run.

So, this will be useful to analyze the profits and see how they get varied in multiple simulation runs. We can also use a print statement here and we can write `print of _profit`. Now, let us say, by default we are running this simulation for 10 times. So, after simulating, we have 10 overall profits. So, to find the average of all these profits, we can write `average_profit` as the `np.mean all_profit`.

Now, this statement will give you the average of all the profits but since the python uses a large position, we can also use a round function that will round up our value to, let us say, two decimal places. So, here I am

using multiple operations in a particular state in a single statement. So, the average profit will be the average of all the profits to two decimal places.

Now, our twist function is completed, so we can write the return here and we can return, let us say, the complete sales data which will be useful for creating a Web-Based Decision Support System, the average profit that we calculated and let us say, all profits also. So, until now we have created our complete decision model.

So, let us try to run this model and see what happens for the default value, that is properties, low properties 47, high properties 50k and all that. So, to run this, we can write python financial_mc.py. So, here we can see that the overall profit for period one is this much per second, all these things are varying.

Now, our first module for the complete DSS is completed. So, we can move on to the next part of the module which is the creation of the Web-Based Decision Support System. So, we will continue this development in the next lecture. Thank you.