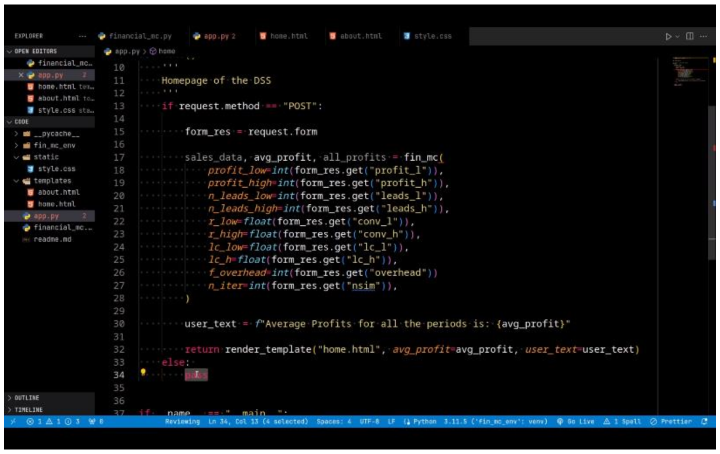
Advanced Business Decision Support Systems

Professor Deepu Philip
Department of Industrial Engineering and Management Engineering
Indian Institute of Technology, Kanpur
Professor Amandeep Singh
Imagineering Laboratory
Dr. Prabal Pratap Singh
Indian Institute of Technology, Kanpur

Lecture 39

Financial DSS using Monte Carlo Simulation (Part 2 0f 2)

Hello everyone, I welcome you all to the lecture series on Advanced Business Stitching and Support Systems. I am Prabal Pratap Singh from IIT, Kanpur and we are discussing How to Create a Basic Financial Simulation Model Using Monte Carlo Simulation for a creation of the Web-Based Decision Support System. In the last lecture, we have Created our Decision Model in Our Python File and now we are going to Develop the Flask Application for Generating our Web-Based Decision Support System. To do that, we will now create a new file with the name app.py. So, here is our file where we will create our complete Flask Application. To create a Flask Application, we first need to install a flask library.



```
    '''
    Homepage of the DSS
    '''
    if request.method == "POST":

        form_res = request.form

        sales_data, avg_profit, all_profits = fin_mc(
            profit_low=int(form_res.get("profit_l")),
            profit_high=int(form_res.get("profit_h")),
            n_leads_low=int(form_res.get("leads_l")),
            n_leads_high=int(form_res.get("leads_h")),
            r_low=float(form_res.get("conv_l")),
            r_high=float(form_res.get("conv_h")),
            lc_low=float(form_res.get("lc_l")),
            lc_h=float(form_res.get("lc_h")),
            f_overhead=int(form_res.get("overhead"))
            n_iter=int(form_res.get("nsim")),
        )

        user_text = f"Average Profits for all the periods is: {avg_profit}"

        return render_template("home.html", avg_profit=avg_profit, user_text=user_text)
    else:
        pass


if __name__ == "__main__":
```

```
Reviewing  Ln 34, Col 13 (4 selected)  Spaces: 4  UTF-8  LF  Python  3.11.5 ('fin_mc_env': venv)  Go Live  1 Spell  Prettier
```

To do that, we will write the command python m pip install flask. So, flask is now available in this environment. So, we can write flasks, import the Flask Application, the render template method and the request. Now, we can start writing to initialize our app.

To do that, we will create a new variable that will contain our complete app and we can use the Flask Application. Flask double_name variable to initialize the application. Lastly, to run this app, we can write ‘if’ name = to_name. If somebody is executing this file, then what to do? Run the application app,.run into debug mode. So, let us now create the home page of our Decision Support System by creating a new function named as home and create a doc string for this function and page of the DSS.

And, to create the route for this function, we can use the decorator app.route and let us write the flash as the route function and we can complete the function by writing the basic return statement and we can write here the home page of DSS. Let us try to run this much code, so that we can see whether our basic template of this Flask Application is working fine or not.

We can write python app.py and we can see here that at this local host, we can be the string which we are returning using our home page function. So, let us move back again to our VS code and start developing the next functionality. So, we have already developed our module, the decision model in this file, which is again in the root directory of this application.

So, we can import this file also. Now, we can write here from financial mc import fin mc function. And, we can use this fin mc function inside our home page, so fin_mc. Now, if you recall what this function will return, it gets called through a different module. So, the return includes these three things in a tuple.

And, we have already learned the tuple unpacking. So, let us just write these three things, sales data, average profit and all profits. So, sales_data comma average profit comma all profit. Now, this statement will first call this fin mc function.

And, since we are not providing any arguments here, it will run with the base arguments, the default arguments, which the decision model contains here like these. And then, once the successful execution gets over, it will return three things and these three things will be captured into these three new variables in this function.

Let us now create our template as well. So, to do that, we can just write make directory template, make directory static to store our static files. Then, we can create a new file with the command touch in Linux and we can write templates home.html.

Another file that we can create is template about.html. Similarly, we can also create our stylesheet. So, we can write touch static slash style.CSS. Now, we can verify whether these things are created here or not. So, these are available here and stylesheet is also available.

So, now let us start editing our home page template. So, to do that, we can run this shortcut command in our vs code, and we can write DSS. So, now to use our template, we need to use this function from the flask, that is render_template, what this needs is a template name, which we have already created that is home.html. And, we can put various variables into our template, and use a ginger templating engine to access all those things.

So, let us just provide average profit, a variable which we are trying to transfer into our template. we can write here average_profit. So, now if we can run this application again, we can see whether these things are utilizing the newer template or not.

Now, you can see that this is a financial DSS, which was written by us in our home.html template. So, we can move from here, we can stop this. Now, let me create a form in the home.html template and I will join back again with the complete form template inside our home.html, so that the user can provide every detail for each simulation run. So, let me just complete all the steps, and then we will understand how this form gets utilized. So, now our template is complete.

So, today we are using a bootstrap CSS framework to style our template, and, first of all, we are creating a navbar in our template. Once the navbar was complete, we created different containers. So, in the first container, we are creating a form and this form will capture all the simulation input parameters like the low profit or high profit. And, we are also validating all these things. Since all of these details are already discussed in our previous course, we are skipping here to not waste our time by creating these templates.

So, you can see that all of the template is complete and we are also using these static. Our static query generated images, which we will generate after creating the complete DSS system. Also, we have created a similar about page by using the same navbar and we have also created our stylesheet by just creating various details for the body of the HTML page.

Now, we can again come back to our Flask Application, and we start fetching the details from the HTML page, and then doing the various computations. So, now to utilize our data inputs received from the form using the home.html template, we can start tweaking our Flask Application by using the methods argument.

Method is equal to get and post. So, what we are trying to say here is that we are capturing both get and post requests. Now, when the user is giving the get request, then this Flask Application will run differently. Otherwise, if we are receiving the post request, which is useful to us, then we will run some other operations.

So, to do that, we need to write a conditional statement using the if else conditions. So, let us write the first if else, which says that if request.method is post. So, if the user has sent a post, if the user has used the post method, that is, it has submitted some information through the HTML form, then this application should run the thing. Otherwise, else run all these things. So, let us just write pass for the timing and we can use this return statement for our post method.

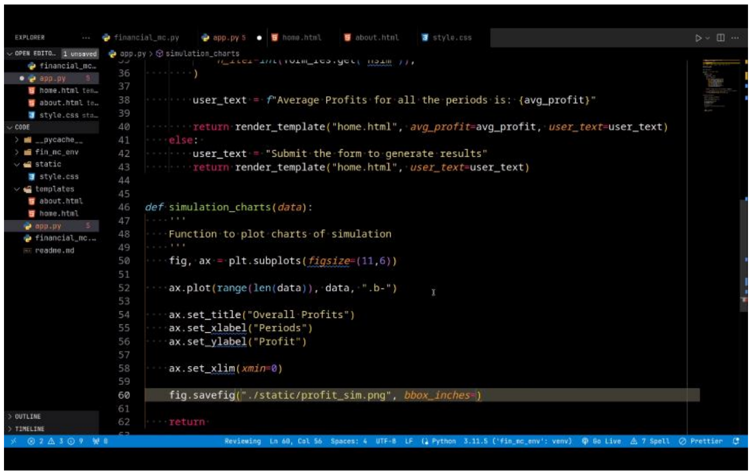
We must follow our indentation correctly, otherwise the application will not run. So, the first thing is that since the user has submitted the form, we can use the result from the form in a new variable form_res. And, we can do request.form because this form variable contains all the data. Now, we can start using the arguments in our financial model and we can write here profit_low, form_res.get. Because it is a dictionary, we are using the get method and we can write here profit_else. So, now from where we are using this profit_else, so we can go to home.HTML and we can see that in our form, the first thing which we are capturing is this profit low and we are using the ID profit l name as profit_l.

So, this is the thing which we are capturing here using our Flask Application. Now, if you remember that this will be a string, to convert this string into an integer, we can use the integer function int and now we will get an integer value into this argument for this fin mc function. Similarly, we can write other things like profit_high and this should be int form_res.that profit_high. Similarly, we can write for the lead, n_leads_low = int (form_res.get (“leads_l”)). And, this should be the right string because it is a same thing. Similarly, we can write n leads high int, it should be an integer, so let us put int form_res.get leads_h.

The next thing is r low. Since this is a fraction, we need to use float to use the decimal value form_res.get and for the conversion, we are using form_l. Similarly, we can write conversion high as r high equals float form_res.get form_h. Another thing is the lead cost low, which can be used as form_res.get, lead cost low, lead cost high is float, form_res.get lc_h.

And, the second last thing which we need to fetch from the form is the fixed overhead cost, so fixed overhead equals to int form_res.get overhead. The last thing is the number of iterations that want to run, int form_res.get nsim. So, now our call to the main decision model is complete and we are also using the data given by the user using the HTML form.

Now, the next thing which we can do is to create a new user string, the use of this is that we can show in our DSS, in Web-Based DSS, that what is the average output, so we can write here f string, which can say average profits for all the periods is average_profit. Now, we can send this user string to our template using the argument user text equals user_text.



```python
        )

        user_text = f"Average Profits for all the periods is: {avg_profit}"

        return render_template("home.html", avg_profit=avg_profit, user_text=user_text)
    else:
        user_text = "Submit the form to generate results"
        return render_template("home.html", user_text=user_text)


def simulation_charts(data):
    '''
    Function to plot charts of simulation
    '''
    fig, ax = plt.subplots(figsize=(11,6))

    ax.plot(range(len(data)), data, ".b-")

    ax.set_title("Overall Profits")
    ax.set_xlabel("Periods")
    ax.set_ylabel("Profit")

    ax.set_xlim(xmin=0)

    fig.savefig("./static/profit_sim.png", bbox_inches=)

    return
```

Now, if the user is given a get request, that is, it is just reloading our DSS, then what should we do? So, To mention all of these operations, we need to populate our else clause, so we can write here, we can also provide a user_text, when we are getting a get request and you can say that submit the form to get or to generate result. So, this could be a predictive text that the user will be when he or she is just reloading our page. And during a get request, we should only return the template that is the home.html and the user text. Now, we are in a position to run our model again, so let us run it again, python app.py and it shows some text errors, here we missed a comma.

So, now you can see that this is how our home.html template looks, so you can see here that in the left-hand side, we are getting all the simulation input properties like profit low which is by default 47 but we can change it and similarly profit high, leads low, leads high, conversion rate low, infraction, conversion rate high also infraction, similarly lead cost and the overhead and the number of simulations that one requires to run.

Now, in the right-hand side, we can see the result, that is, since this is just a get request like if I am just putting my cursor on the home link and pressing it, then this is a get request because I am not submitting anything, so the post method will not work. So, in our get method, we have mentioned that this string should print, submit the form to generate the result and in the third section, we are providing a chart, so we will see how this will work.

So, let us just write from here. So, when we submit, we can see that this value keeps on changing, because every time the simulation is running for different random numbers. Now, if we change this value to, let us say 100, these things will have different results. Now, let us move on to the last part of this development which is to create charts. So, to do that, we need to install a new library that is matplotlib.

So, python m pip installs matplotlib. So, our library is installed and we need to first write some more import statements. So, we can write here import matplotlib.pyplot as plt and we can write import matplotlib as mpl. So, this is another relias which we are using. And, let us also mention that we are trying to use the following font.

So, the weight of the font should be bold. And, the size should be sixteen. To update the thing to our matplotlib, we need to write mpl.rc, so we are updating the configuration for matplotlib for the font section only. And, we are using our doublet command. And, we can also mention another style change for the style of the matplotlib.

And, we can write plt.style.use.ggplot. So, now we can start creating our simulating chart. To do that, we need to create a new function. We can develop this chart in this new function as well, but to keep all the code into different modules, we should create a new function. So, we can write as simulation_charts and let us say it gets a data argument and let us populate the doc string quickly.

So, this function to plot charts of simulation. And, let us write a blank return statement here. And, we can initialize our figures. plt.subplots figsize is, let us say, 11 comma 6. Let us plot the data. The x axis should be the range of length of data. That means the count of the values that are available in the data argument.

So, what would be data? We are trying to create a chart for all the overall profits for each simulation. So, we are capturing this detail in this all profits variable. So, once we are sending this list, this is a list, we can clarify it from here that this all profits list is created using our list comprehension.

This list will get transferred to this function as a data variable and we are trying to first count the number of profits stored in this data variable, and then we are creating a range of values from 0 to the last element to create our x axis.

The next thing is to create our y axis which is simply data because we are trying to plot the profits on the chart and we can use the color and line style by mentioning.as the marker, b as the color and - as the line type. So, this will generate a blue line with.markers at each point of the data at the y axis with a continuous line.

Now, we can initialize our title for the chart and we can write here ax.set title overall profit. We can write the label for ax axis as well by using ax.set xlabel. We can write here period. So, every x value is a particular period, let us say week, month or whatever period you are trying to calculate or run the simulation, ylabel is profits.

Now, let us limit our x axis to 0. So, to do that, we can write ax.set.xlimit. Where minimum of x should be 0. So, we do not have any negative values for x axis but we have negative values for y, so we are not initializing y limit as 0 here because our profits can be negative as per our decision model. Now, we can save this figure in our static directory.static. We can write the name of the figure as profit_sim.png. And, we can perform some other keyword arguments as well like bbox inches to create a tight image. And, also mention the dpi for the image as 300. And, this function should return this figure object.

Now, our simulation chart function is complete. So, we can call this function in our main homepage by following this function here. We can write simulation charts and what should this get as an argument is the data of all profits. So, this simulation chart function will get the list of all profits and it will generate a figure and it will return that figure too.

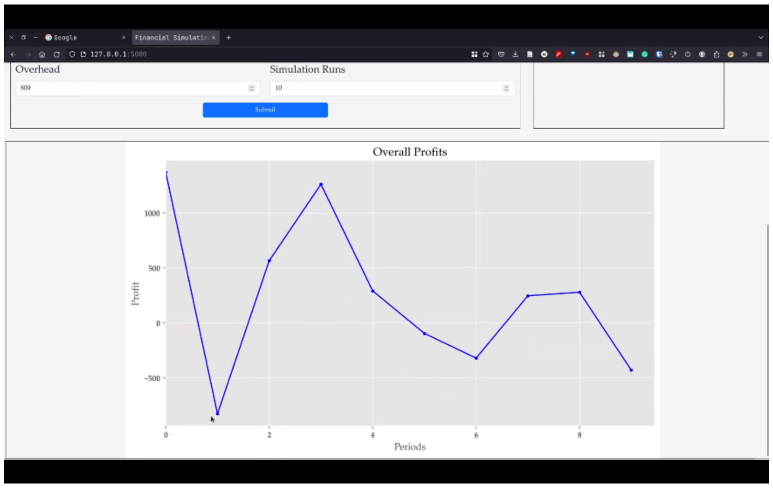
So, we can store this figure in a new variable named sim fig. So, what this chart function will do is, it will first generate a figure, then it saves the figure in this static folder, alongside this style.css. And, here in our home.html in the last section, we are trying to generate a centered image using the same url at the static with the file name profit_sim.png.

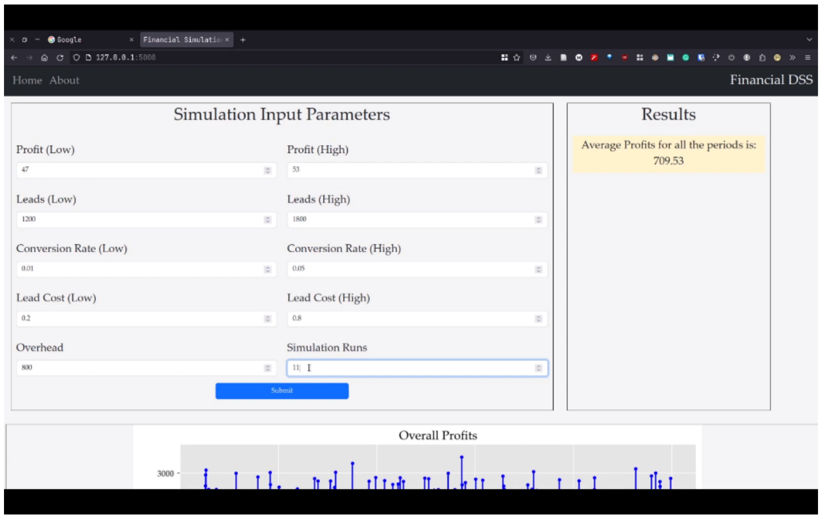
So, every time the user runs this model, it will create the average profit for all the simulation runs, and then a figure will also get generated, so that the user can see the variations for all the simulation runs. So, let us now again run this application again. So, let us reload it and we can see that these are all the default values, so let us just try to submit them.

And, you will see that a nice chart is also getting generated now and it is showing that this simulation gets run for these 10 simulation runs and for each simulation run these.markers are showing the actual overall profit value that gets evaluated.

So, sometimes it also gets negative below zero line. Now, let us try to increase the simulation runs and let us see what happens to the chart and the value. So, initially it is 234, so let us write 25 simulations. Now, it has executed for 25 runs and you can see here the axis, the period also becomes 25, and you can see that there is a lot of variation. So, if you are a manager and you are trying to see what will happen to my profits in the long run, so you can keep on increasing or decreasing these simulation runs and you can see that these results will keep on varying, but to get the overall idea of what will be the actual likelihood of these overall profits, so you can try to run this simulation for a longer period of time, so let us say 1000 and check. If you see that the average profit is still 641, now if you run for 1100 it is 719, so it is still changing a bit.

So, let us try to run this model 5000 simulation runs and it is again 709, so by this way you can see whether your simulation gets stable or not after a large amount of simulation runs. So, let us also run this for 10000 runs. So, you can see here that it is 700 times, so something around 700 to 720 or something you can get the overall average profit but you have to see losses also that is the negative value of profit as well.


Overhead	Simulation Runs
800	10



Simulation Input Parameters

Profit (Low)	Profit (High)
47	53
Leads (Low)	Leads (High)
1200	1800
Conversion Rate (Low)	Conversion Rate (High)
0.01	0.05
Lead Cost (Low)	Lead Cost (High)
0.2	0.8
Overhead	Simulation Runs
800	11

Results

Average Profits for all the periods is: 709.53

So, this is the complete model and this model is also available at GitHub. This is the complete repository for this model and you can download this repository and try to run the same command and I have also generated this requirement file, so that you can create the same python environment and run this simulation model easily. So, thank you everyone and we will meet in the next lecture.