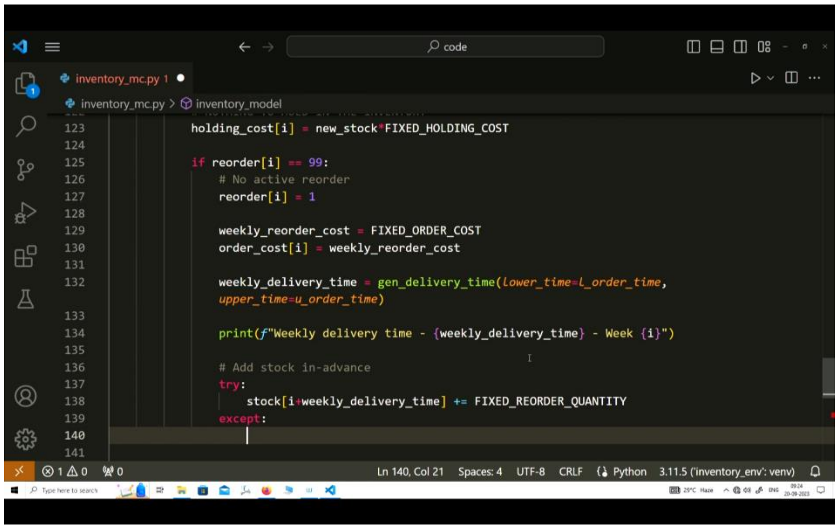
Advanced Business Decision Support Systems
Professor Deepu Philip
Department of Industrial Engineering and Management Engineering
Indian Institute of Technology, Kanpur
Professor Amandeep Singh
Imagineering Laboratory
Dr. Prabal Pratap Singh
Indian Institute of Technology, Kanpur

Lecture 37
DSS For Monte Carlo Inventory Simulation (Part 2 of 2)

Hello everyone, I welcome you all to the lecture series on Advanced Business Decision Support Systems. I am Prabal Pratap Singh from IIT Kanpur and we are discussing How to Create Various Decision Support Systems. So, today we will be continuing with our creating the code for this Inventory Simulation and let us jump into our Visual Studio Code.



```python
            holding_cost[i] = new_stock*FIXED_HOLDING_COST

            if reorder[i] == 99:
                # No active reorder
                reorder[i] = 1

                weekly_reorder_cost = FIXED_ORDER_COST
                order_cost[i] = weekly_reorder_cost

                weekly_delivery_time = gen_delivery_time(lower_time=l_order_time,
                upper_time=u_order_time)

                print(f"Weekly delivery time - {weekly_delivery_time} - Week {i}")

                # Add stock in-advance
                try:
                    stock[i+weekly_delivery_time] += FIXED_REORDER_QUANTITY
                except:
                    
```

inventory_mc.py 1
inventory_mc.py > inventory_model
Ln 140, Col 21   Spaces: 4   UTF-8   CRLF   Python   3.11.5 ('inventory_env': venv)

So, till now, we have covered How to Create Our Fixed-Order Quantity Inventory Simulation. So, we were iterating for all the weeks one-by-one by using this 'for loop' and we have created our conditional statement for checking when the demand is less than stock, then what should happen. So, we were actually calculating the new stock for that week, then we were updating that stock for the next weeks, and we also checked whether after fulfilling the demand of the consumer the new stock is less than equal to fixed reorder point or not, and whether there is any active reorder happening in the system or not.

So, for that situation also, we developed our model. Now, in the last lecture, we left while developing the condition where the demand is greater than stock. So, since the demand is greater than stock, therefore there should be an unmet demand and the new stock will be 0 in the inventory. Now, if the new stock is 0, then the logical thinking says that we should reorder, but we must check whether there is an active order or not.

Since, our assumption is if there is any active order, then we will not create a reorder point. So, let us continue and start checking for the active reorder. So, to do that, we will start creating a new conditional statement using the 'if else' statement. So, if reorder on the ith week = 99 because 99 means that there is no reorder and we have not updated it with 0 or 1 anything. So, we can write here that no active reorder.

Now, if there is no active reorder, then we should update this ith week reorder with 1, because now we are going to make an order. So, reorder at ith week should be 1. After that, we should add in our books that there should be a weekly order cost. So, this is, weekly reorder cost =, this is the fixed quantity. So, we can store this in Fixed-Order Cost and we can update the order cost for the ith week as weekly reorder cost. Now, if we are giving an order, then we should get a delivery time for that.

So, we should generate the next delivery time using our function created earlier which was gen delivery time and it should need a lower time which should be the 'l' order time, the upper time it should be 'u' order time, but whatever this going to return to our execution, we must capture this in some variable. So, let us say, this should store weekly delivery time and we can write a print statement saying that this should be a string. So, weekly delivery time is this variable weekly delivery time, and we know that this is the week 'i'.

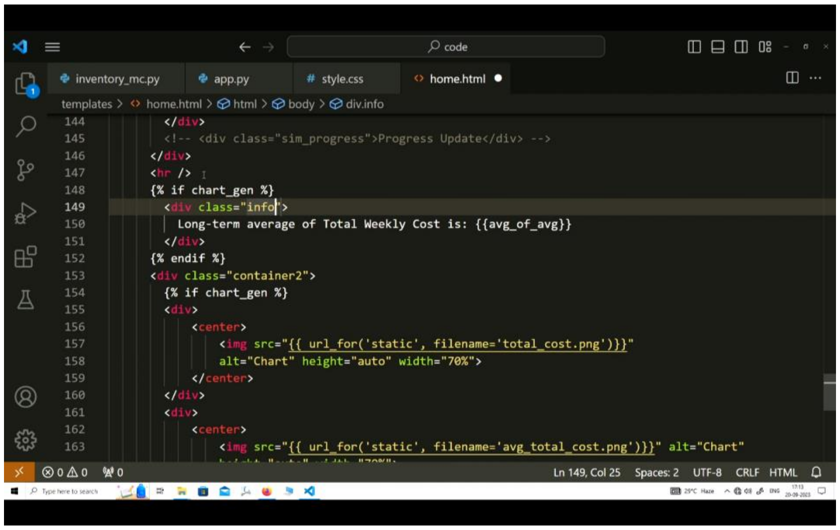
Now, if we are ordering, then we know that this will add to the inventory in the next few weeks whenever the delivery time is over. So, let us add this new quantity in advance. So, add stock in advance. So, like previous times we will check whether this will happen or not using the try and except block. So, stock i + weekly delivery time = fixed reorder quantity.

So, this remains the same for each iteration and for the complete simulation model except we can write a print statement that prints out of index adding stock and it has a demand greater than stock. So, these kinds of print statements will tell you where the error is coming from.

Now, whenever we order, we used to block further reorders. So, we need to block further reorder. For that, we need a new looping statement with a different variable which should loop from i + 1 to i + weekly delivery time + 1 because the last number is exclusive for the range function. So, again we try reorder at jth week should be 0 which actually means that during the actual iteration of this jth week, if the stock again dips below the reorder quantity, then even though the stock is below the reorder quantity, we will not make a new order. So, that is what this 0 indicates here except index error. Again, here also we should mention that this is for the index error, otherwise all the exceptions will get captured by this except block. and we can again print that index up to bound stock reorder demand greater than stock.

Now, this 'if' statement of creating a new order is complete. So, what if this is not equal to 99, that is there is an active order. So, if there is an active order, then what should happen? So, we can write that in our else block and we must provide a comment here that there is that except 99. So, we are not reordering here. So, the weekly reorder cost is 0 for this week and the order cost for this ith week is the weekly reorder cost.

So, this 'for loop' is complete for us. So, we can see that this is the indentation level we started with. So, to come out of this 'for loop', we should add more code at this indentation level. So, we can write basic print statements to check what is happening after each iteration. So, we can write that the final system stock is the stock variable which we were using earlier.


code
inventory_mc.py
app.py
style.css
home.html
templates > home.html > html > body > div.info

```html
    </div>
    <!-- <div class="sim_progress">Progress Update</div> -->
  </div>
  <hr />
  {% if chart_gen %}
    <div class="info">
      Long-term average of Total Weekly Cost is: {{avg_of_avg}}
    </div>
  {% endif %}
  <div class="container2">
    {% if chart_gen %}
    <div>
        <center>
            <img src="{{ url_for('static', filename='total_cost.png')}}"
            alt="Chart" height="auto" width="70%">
        </center>
    </div>
    <div>
        <center>
            <img src="{{ url_for('static', filename='avg_total_cost.png')}}" alt="Chart"
```

Ln 149, Col 25 Spaces: 2 UTF-8 CRLF HTML

The next variable of importance is the system reorder which can be printed using the reorder array. The next is the order cost for each week. So, again let us write order cost and if we are writing order cost, let us write all the cost. So, we can write the holding cost and there was a stop out cost. And, the final one was the total cost.

Also, our function can return the important variables using a dictionary which we can create here. Like, the demand key will store the demand array, the stock detail will hold the stock array, reorder point will hold the reorder, the order cost will hold the order cost, holding cost will be stored in the holding cost key and stock out cost was in the same variable name. Finally, the total cost is also important.
Now, you can see that this whole simulation model is complete and we can try running this. So, now since our simulation is complete, we can try running it. And, to run our simulation, we need to first open our terminal, and then activate our environment.

So, our environment is stored in this inventory env which is actually the name of the environment. So, let us write.inventory_env scripts to activate this environment. Now, it is activated and we have already installed NumPy before creating the simulation model. So, this simulation model should run. So, let us start running our model and this should be done using this command python inventory mc.py.

So, for this simulation, let us check what is the output of the model. So, the first thing we did was to create a demand. So, this is the randomly created demand array and in the first week it is saying that the requirement is only one unit, in the second week it is two and so on. Then, it also shows that in week one, it had created an order, and the delivery time for that order is two weeks. Let us come below and check what the stock level is. So, the system stock array should hold the fixed quantity as six in the first place. So, it is correct.

After that, our first week requirement was only one unit. So, that is why, in the first week the stock becomes 5, 6 - 1 is 5. And, since this is not a reorder point. So, we will move forward in the second week. The demand is two, so from the stock of 5 from the first week we will reduce 2 more units which becomes 3 and these 3 units is the reorder point level. So, that is why, the system is printing that weekly delivery time for week one is two weeks.

So, it is creating a new reorder in the first week and the delivery time is two. So, that is why, if you see the system reorder array. In the first week, there was no reorder, in the second week there is a reorder. So, the model makes it as one from 99 and since the delivery time is for two weeks. So, for the next two weeks, the 99 becomes 0.

And, the order cost is also gets updated for the first week which is thirty units, the holding cost is calculated using the number of items in the stock multiplied with the two units of this. And, these stock out costs, in this run of the simulation the stock out cost remains zero which means that there was no stock out for the whole simulation, but since this is a random simulation. So, as many times we repeat the simulation this variable keeps on changing and it will total cost.

After all the calculations of the different costs, the total cost array is like this. So, now let us create our Decision Support System based on this model. So, the beauty of creating a simulation model inside a function is that now we can call this inventory model function from a different module. So, let us create a different module and we can call it app.py. As we did in our previous exercises, now we are going to create a Flask Application.

So, before creating the application, we first need to install a flask in this environment. So, let us write python m pip install flask and let it install it first. So, since the installation is successful, let us now start creating our application. So, the first thing we need to do is import flasks. So, we can write from flask import the flask function and also the render template function and we can also import our inventory model simulation using inventory mc.

So, the name of this file inventory mc gets written here and from that import inventory model. So, our two imports are ready and now we can start creating our home function for the main home page and let us just return DSS for Inventory Simulation for the time being and we need to first initialize the app also. So, app = flask name and we can route this app for the home page using the app.route. There should be a doc string for this function as well. So, we should call this app using the main function with the argument as debug equals true.

Now, this much code can initialize this app and run this app. So, let us try running this python app.py. So, you can see that this basic thing is over. So, now let us try to create our template section as well. So, to do that, we need to create new folder templates and we also need styles for this.

So, let us create the static page inside this initialize a new file named style.css. And, inside the template, we need to use a home.html page. So, before doing anything in our app, we first need to create this home page. And, since this is a simulation model, we need the user to modify the simulation parameters as well while running each simulation.

So, we need to create a form on the html, and then we need to capture the inputs created by the user and transfer those inputs to the Flask Application to create the simulation model and run it based on the user input. So, let us start creating the html forms. So, to do that, we first need to generate our html page and we can write here DSS for Inventory Simulation.

Now, the first thing is, we need to create a nav bar and it should contain a link which should go to the home page and this should be the text it shows. The next thing is an about page to tell the users what this application is all about and you can simply write this is an about page. Next thing you can do is, create a name for this in the nav bar only. So, to do that, let us create a new div and write Monte Carlo Simulation for Inventory Management and let us also create a class for this with the name title. Now, the nav bar is complete. So, we can start creating our different containers.

So, this should be the container one, and inside this we should create a new container for the user form and let us start creating the form itself. So, we do not require an action here, but we definitely require a method which is the post method and we can start creating our different field sets. So, the legend here should be Configure Inventory Model and the field set should have a class for the field it is and the legend should be demanded. So, in this section the user will provide the demand.

So, the label for this is a lower demand. So, it is 'l' demand the lower limit of demand. So, this is a number. So, an ID should be 'l' demand, name should be 'l' demand, the minimum value should be 1. So, here, you can keep on customizing the application. We are making it so that the minimum value which a user can put inside this field set is only 1 and the value by default should also be 1. Just close this, so that we can have more space. So, this thing is complete, let us write another field detail using a different field set, class equals to field. Now, we should update this value as 4. So, the default value for the upper limit of demand is 4.

Let us try to link this HTML file to our application. We can write a link and your href should be using the ginger syntax. We can write a URL for static because this is the name of the folder which we provided and this is the file name which is the style.css. And, we can also provide that we are entering this template, render template home.html. So, the style thing is undefined what it was and we need to put this in quotes because this is the actual file name, similarly like the folder name.

So, let us rerun our application. So, now you can see here that this is a simple nav bar which we created until now. This is the title of the model which we are creating for the DSS, then this is the user form which we are creating. Now, let me complete this form, and then we will again restart so that we should not keep on using this time because we have already discussed how to create forms and other things in our previous course. So, let me just complete this form and we will switch back again.

Now, we have filled our template file and also the style file. So, we can look here that this was the container one which we created earlier. Similarly, this was the lower limit and upper limit of demand, then this is the order delivery time field set and we are again capturing the lower and upper limit for the same thing, then the inventory cost which the user should also fill the fixed inventory cost which we are making the default as 30, after that the holding cost which should be 2 per unit item.

Similarly, the stock out cost is 20 here, and then the simulation parameters like the number of weeks which is 20 by default and the number of simulations which is 7 by default here. Also, the initial stock provided by the user is also a capability of the simulation model that the user can change this.

Similarly, you can also provide a random seed and by default it is 99. Now, the replenishment details which is nothing but the reorder point. So, the reorder point should be 3 which means it should be less than equal to 3 and the quantity for each reorder is 6 which is the fixed quantity. So, the form is complete and similarly we have updated the style of the body, the header one the nav bar for all the links and links, whether it is visited or not and what happens when it is hovered over. Similarly, we have also created a style for images, because whatever the simulation run says, it should be printed on a different image each time.

So, we will create this image today. After that, these are the different classes that we created for the title. We are making these styles for container 1, container 2, and the user form. These are different classes which we still need to create. So, let us see how our app looks now. So, we can read on this. So, the style and the complete form will look similar to this if you follow whatever happens. This is the basic HTML thing which we already covered.

So, this is the title and this is the home link. So, whenever you click it you come to the same page. This is the award section which we have not yet created in the application. So, it is saying that this is not found. So, let us now complete our application because our application is not doing anything other than just ending this template. So, we can create a different page name as about and we can write this quickly as this is the about page and we can return about page.

So, now you can see that the about page is active. Now, let us get into the details of how to simulate this model by using our Inventory Simulation Model. So, now since our form is ready and it is providing the details of the user input using this post method which we created here, this post method. So, you can use a new function. That is the new capability of this Flask Application by using the request function. In this request function, we can use the user form data provided by the user. So, we must first check whether this request variable is available to this application or not. To do that, we can check whether the request.method = post.

So, this section of the code will only run when the form created by us will deliver the data using the post method. If it is returning the data with the get method, then this section will not run. So, what if it is returning the section with the get method, then we can write it in else and we can say that just return this template. So, what it means is that the function is getting simpler. If the return method is get, then we must only return this home.html, otherwise we can print here that we are getting the data is posted.

So, now let us start creating our application for the decision that when the user is submitting the form, then what should happen. So, the first thing is we should create a chart. So, let us create a new variable and make it as one, after that we should store this result of the form in a new variable and the variable name should be form_res and we can store the request.form value.

Now, if the value of form.res.gets because this is a dictionary, we must check whether the user is providing a seed or not. If it is providing a seed, then we should do different things otherwise things will happen differently. So, if the default value of the seed was 99. So, if this value = this, then what we should do is, we can say that nothing is required because we are not making the seed as a particular number. Otherwise, we need to generate a random seed using the NumPy module and the value of the seed comes from the form.

So, we again write form.res.get seed, but one thing to note here is, that whatever the value the user provides using form the Flask Application will get it as a string. So, to convert the string into an integer, we know that we can use the int function available with python. So, everywhere we are using this value, we should make it as an integer. So, this should be form_res. Now, seed is the first thing which we need to capture. So, it is over.

Now, let us create a new blank list name as a sim result list. Since, we are running a different number of simulations each time, we need a blank list to capture all the results of all these simulations and now for each simulation we need to iterate some different commands in every time. So, to do that, we again use our looping statements like the 'for loop'. So, for each sim in the range, which range, the data provided by the user.

So, int form_res.get nsim. So, nsim is the name given by the label for this number of simulations. So, this was nsim. So, this is the name which we are using. So, we can move ahead and write that the simulation result for this particular simulation should be stored in the sim result and use how to run the simulation using our inventory model which we imported here. So, we can write an inventory_model and this requires some arguments which we can provide using the form data.

So, we can write n_weeks equals to convert whatever we are getting from the form using int and write form_res.get 'n' week and lower_demand equals int. So, let us copy this and change it here as 'l' demand. This should be upper demand. So, the next thing which we need to capture is the order time. So, this was stored in 'l' order time and the form reports this using 'l' od time, l_t time, 'u' order time equal to int form.res.get. This get function is applicable in the dictionary. So, our form_res is our dictionary. That is why we keep on using this.get method. This is u_ot time. It should be no double equal to because we are not doing conditional checks here. We are just providing the argument values.
Next, we need to capture the fixed ordering cost which is stored in order cost. Next thing is fixed holding cost. Next, the stop out cost is captured by the stop out cost. Fixed reorder quantity, this gets captured by roq in the

form. Then, the last fixed reorder point, this gets captured into roq reorder point. And finally, the initial stock, this is available in 'i' stock.

So, if you remember the inventory model simulation which we created here, these are all the arguments, and here we are making it as by default. So, these are the default values, but the user can also change this. So, here the user is changing based on the form. So, this is the form and the user is filling the different details here and we are capturing in this application from where it gets transferred to this model. So, this is the whole process flow.

Now, the same results hold all the written statements provided by this inventory model which was this. So, it will hold this dictionary in it. and we can work on with this dictionary to perform different operations. So, we can also add one new argument to the dictionary provided by this simulation model as nsim. So, what this will do is, it will capture this key nsim and it will store the number of simulations that the current simulation for which this result was generated.

So, we can write int nsim + 1. This is int sim. Now, this sim goes from 0 to the number provided by the user, but the actual counting does not start from 0. So, that is why, we are adding 1, so that it should be useful to the user who is reading the output. Next few things which we need to append, this is to update our sim result list which we created here. So, this was initially blank, but now with each iteration, we can update this list by appending the result of this simulation in a dictionary.

So, we can write sim_number and we can store this in our result. So, we can store this key using this statement sim_result. So, now what we can do is, we can write a new return statement for this form method when the method becomes post. So, for this we can write, this should be out of this 'for loop', because here we are doing this for each simulation iteration. So, the return statement should be for this function and since we are creating two conditioners. So, for this function when this condition is true, that is for this if, we should write a different return statement here.

So, we can say render_template which template home.html and provide these different variables to this template. So, inside the result list, we can write sim result and we can provide a total cost variable and it should be fetched from the same result.get total_cost. So, now we have created a form, we have used the input from the form. But, before using anything inside this application, we must also tell the route function that we are capturing this post or get method. So, to do that, there is a function named as method equals to, and we should provide a list of methods that we are working with. So, let us write a post comma get. This should be methods and now since we are transferring these variables we can try capturing it in the template as well.

So, let us create a new container 2, this is the class. Inside this, we can write the total_cost. So, we run this now. You can see that the default values are refilled here. So, just putting the get method works like this, but if I put this using submit, then you can see that this output is generated. So, what is this? This is the total cost which we just created here. And, from where it is coming? It is coming from this application using this return template home.html and here we are storing this total cost for the last simulation run using this sim_result dictionary. and this was this if block where the request for the method was posted only. So, whenever we are trying to submit the form this block will run otherwise this basic block will run we will only render the template.

Now, in the last time, we tried this submit button, but if we just go to this home page again. So, this is the get method and this returns the basic rendering of the html page. So, all the output vanished. Now, let us try to do a few more things to see how our complete simulation is running. So, to do that, we can create an image of all the data points using a different library available in python.

So, we have not discussed this library, but it is a very useful library and it is an advanced library for visualization. So, you can first install this by using python icon m pip matplotlib. We will also need another library named pandas, this is usually required for tabular data.

Now, if you remember I have created this variable chart_gen. So, this is the generation of charts. So, to generate a chart, we first need to create a chart. So, to do that, let us create a new function here with the name def system_stat and this should capture the complete list of the simulation result. And, we can provide a doc string saying that generates system statistics and here using matplotlib we first need to create a figure and axis. So, let us say, this is ax1 and we can use it. We first need to import this.

So, all import statements should go here. So, we can write import matplotlib.pyplot. So, we are using only this pyplot function and using an alias named as plt. So, this was the actual library which we installed. This is the function in it and this is the alias. Now, using this alias we can create subplots, plt.subplot without any arguments and now let us start working with our generated data.

So, let us write a variable name each sim average total cost. So, make it an empty list, so that we can keep on updating this list for each iteration inside the simulation result list. So, for sim in sim_ref result list. we can write the total cost and we can get this cost using sim.get. So, it is stored in the sim result key. And, inside the sim result key, there is another dictionary which we can use using the get function and move on to our total cost key.

Now, each sim average total cost should be updated using the append method and we should write np.min. So this, we discussed in our earlier lectures that this will create an average from the NumPy array.

Which NumPy array? Which we are using here is the total cost. So, use this total cost array. So, every time a simulation gets run completely. For each iteration of the simulation, we will have a total cost and for that total cost array we will try to find the average of that array and store it in this empty list. So, this list gets filled for the number of values that are there in the simulation for which we are running. So, now we have the values we need to fill this in our plot. So, to do that, we will use this axis and there is a function named as plot that we can use and now we need to fill the x and y axis of this plot.

So, to do that, we can write range length of total cost. So, this total cost variable has some number of field locations. So, based on the length of that field location, we are generating this series of sequence data for the x axis variable. So, let us say, if there are 20 weeks, for the 20 weeks we will generate the value from 0 to 19.

So, this way our x axis gets filled. Now, the y axis is nothing but the total cost. So, this was an array and we can provide a label here also for this particular simulation and we can write the sim using an f string sim sim.get sim_number. Now, our 'for loop' is over. So, there is one more thing which we can do is we can get the average of averages. That is for all the simulation runs, we can find a particular average. So, to do that, we have the total cost for each simulation run and we have stored the average of each simulation run in this variable.

So, we can write the mean for each sim average total cost. So, this way we will get the average of average, this is the long-term average. Now, we can also print this average in our terminal using and we need to fill the title for this chart which is the weekly total cost. and this was our first axis. So, for the same chart, we need to also provide the x label and we also need to provide the y label.

So, we can also provide the xtick for this and every chart should have a minimum limit for its axis. So, we can use this as x limit and we can write x min equals to 0, y min equals 0. Now, one chart is complete and we can save this figure. So, this was figure one, we can save this fig one.save fig static. Inside this static folder, we can write total cost.png and these are some of the arguments that the saved image will have.

Similarly, we can create another figure and this will report the weekly average total cost. So, again we will write plt.subplots. What it needs to plot is, each sim average total cost and the label should be ppg dc.

Again, we will need to provide the title for this image. Let us just copy this. And, again we need to save this figure using the fig two object and we can write here average total cost. And, this function should return this fig one, fig two, and average of average.

So, now what we can do is, we can use this function inside our main home function for this post method and we can run the newly created function for all the simulations by using system wonderful stat and this is sim result list and since our the function is returning a few variables, so we need to capture them as well by creating new variables of all sim figure comma average tc fig, average of average. And now, we can provide these arguments to our template, so we can write.

So, this is a Binary variable which we created to check whether we need to create a chart or not and we also need to provide average of average. So, now let us move back again to our html and now we need to create a few more things to generate our chart. So, in this container 2, we do not need a simple total cost error, we need to print our images. To do that, we first need to check whether we are transferring our images to this html page or not using this chart gen variable.

So, to do that, we can write an 'if' statement inside our template. So, we can write if chart_gen. So, if chart_gen is 1, then this statement becomes true. We can close this 'if' statement using and if, and we can center these images. Now, we can write image src (the source of these images) the url_for st tic inside the static page, the file name is total_cost.png. We can provide some other attributes of this image 's' chart height 's' auto and width 's' 70 percent.

So, we can close this image, the next image is in a different div. So, we should also create a div here, so that these two different images remain in different blocks. So, what we will change here is, only the name of the second image width for average total cost. So, since we are using a NumPy module here also and we are generating a range.

So, we should update this with a range function and we must add this NumPy call in this application as well because this NumPy call will hold for this module only here in this module we are calling this NumPy function, but this call is applicable for this module only, otherwise you will get an error.

So, till now we have created our complete model and we have also tried to run the model and now we are trying to show the images generated by the model on the Decision Support Systems on the web browser.

So, to do that, we have created our two div elements inside this container class and we are running this. We will get these images only when this 'if' statement will be true, that is if our chart gen variable is 1, then only these two images will get printed. So, we were transferring this chart gen variable from here. So, whenever the method of request is posted, then chart gen becomes 1 and here in our return statement we are transferring this chart gen variable to the template. So, using Jinja templating index, we are fetching the same variable here in the template home.html, and then trying to find whether this condition is true or not.

Now, let us try to run this and see whether we are getting to different images or not. So, we have already activated our environment. So, let us write python app.py. So, this is the screen where we can see our generated Decision Support System and now if we just click on home, then this will create a get request, but we need a post request. So, we need to transfer the complete data of this user using the submit button.

So, after clicking the submit button, what will happen is let me show you both windows. So, what will happen is, that this is our development environment where this development server is currently running. So, whatever computations happen after submitting this data, all the computations we can see here in this terminal window. So, let us press the submit button.

So, now you can see that the function this app.py module called the Inventory Simulation Model and executed all the commands present in that module, and after that we can cross verify all the generated arrays like order cost, holding cost, stock out cost, total cost and after that it also created the charts using this matplotlib.

So, now you can see the complete DSS and here are the two charts which we generated. So, in the first chart we can see that the title is the weekly total cost and for each week number. So, here we are saying that the number of weeks is 20. So, it is generated from 0 to 19, and the total cost varies from 0 to 35. So, in a particular week, the maximum cost went somewhere around or greater than 35 in this simulation, and this simulation ran for 7 times.

So, that is why, for each simulation, we can get a particular average of the total cost. Now, since there are 7 different simulations, we will get 7 different averages. So, we can see here that there are 7 different values from 0 to 6. So, for 7 weeks we have their own corresponding average total cost.

Now, what is Monte Carlo Simulation? It will help you to decide the long-term behavior of a particular system. So, to get the long-term behavior, we need to run this simulation many times and try to find the average of average which we have already calculated in our model and we can see that we were storing that average of average in this argument. So, we can also transfer this average to our template and try to see whether we can print this value as well in this template or not.

So, to do that, we can print this between the two containers. So, let us write if chart gen. So, we are again testing whether we are generating some charts or not only, then this will run in this container and we can create a new div element here to print our average of averages and we can write long-term average of total weekly cost is and here we will use the double curly braces to get the same variable average of average.

And, we can create this. We have already created this class in our style sheet and now we are printing it using it here which was info. So, this is all we need to print our average of averages. So, let us run the same thing again, but first let us clear everything. So, just using this home button, we have cleared everything.

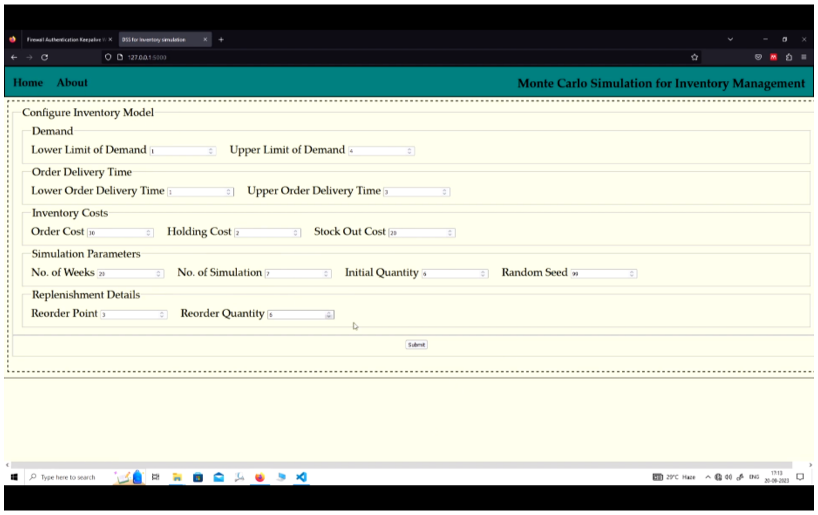
Now, if we submit this time, now you can see that we are getting this long-term average of total weekly cost is 10.64 and some long decimal places and you can also see that these graphs are changing dynamically. So, why this happening is, because every time the simulation is running for a different random sheet, but if we and our default random sheet is 99 which we are checking that if this value is 99.

So, we would not restrict to a particular random sheet, but we can restrict to a random sheet if we want to and whenever we restrict this to a particular number, then these graphs will stop changing. So, once you create your own DSS, you can try and check whether this is happening with your system or not, but here we can check that only for 7 simulations, this is the graph and the weekly cost is 10.64.

If we increase this to 15, let us say, what will happen. So, for all the same parameters just changing this to 15, it is showing that the weekly cost is 14.81, and now why it is showing 7 here because we have already told our form template that the default value will always remain 7. So, whenever we submit it, the actual value gets transferred to cross check whether it ran for 15 or not. We can check this graph and we can see here that the week number changed from 0 to 14 which is actually 15. So, that is how we are changing the simulation and now you can see that the total weekly cost is changing.

Now, let us run this same simulation for 100 times. Now, you can see that variance has increased and weekly cost came down to 13.84. If you run the same simulation to 1000 times, let us see how much this will change the long-term average of the total weekly cost. So, this will take some time because behind the scenes this is running here.

So, let us come again. So, now it is not reloading so it ran and you can see that for 1000 times the total weekly cost is 13.39. So, for 100 times, it was 13.84 now you can see that as you keep on increasing the number of simulations this deviation from the actual total weekly cost is going to diminish. So, that is how, now you can confidently say that the long-term average of these parameters like the lower limit of demand is 1, upper limit of demand 4. So, for all these parameters the total average weekly cost is something around 13.



Home About Monte Carlo Simulation for Inventory Management

Configure Inventory Model

Demand

Lower Limit of Demand 1 Upper Limit of Demand 4

Order Delivery Time

Lower Order Delivery Time 1 Upper Order Delivery Time 3

Inventory Costs

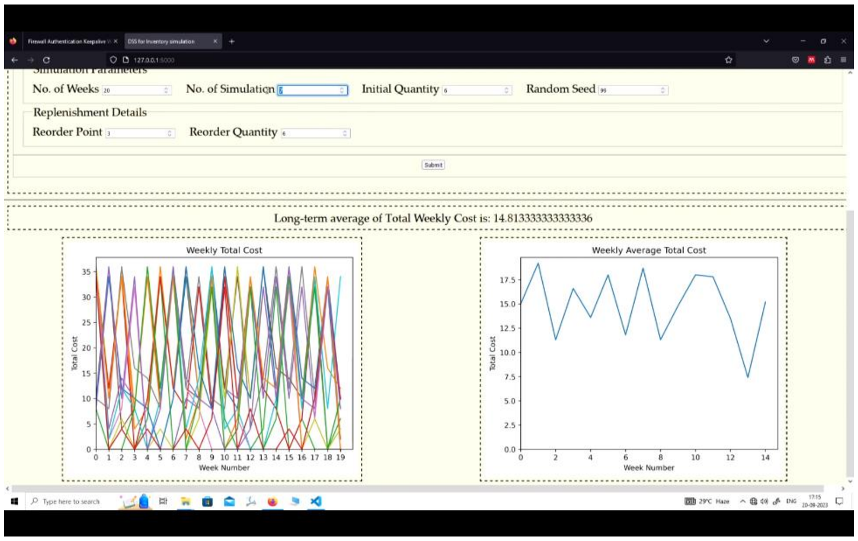
Order Cost 30 Holding Cost 2 Stock Out Cost 20

Simulation Parameters

No. of Weeks 20 No. of Simulation 7 Initial Quantity 6 Random Seed 99

Replenishment Details

Reorder Point 3 Reorder Quantity 6



No. of Weeks 20 No. of Simulation 5 Initial Quantity 6 Random Seed 99

Replenishment Details

Reorder Point 3 Reorder Quantity 6

Long-term average of Total Weekly Cost is: 14.813333333333336

-Some images from the app

So, that is how you can tell your organization that this is how much the inventory will cost you over the long run, if you keep these parameters constant. But, if your organization says that no, my parameters have changed for the next 6 months. So, then you can change these parameters here and your Decision Support System will tell you what are the different long-term averages that they will have to pay for their Inventory Management.

So, this is all the complete end-to-end creation of a Decision Support System from Basic Inventory Simulation Model to Using Flask Application, and then creating templates and styles, and then how to get insights from the simulation. So, thank you and we will meet in the next lecture.