



The image shows a 'cupcake ORDER FORM' with a pink header. The form includes fields for 'NAME + T:', 'ORDER DATE:', 'ORDER NO.', 'PICK UP', and 'DELIVERY'. Below these is a table with 'FLAVOR' and 'QUANTITY' columns. At the bottom, there are checkboxes for 'CASH' and 'OTHER' payment methods, and a field for 'ORDER TOTAL'. To the right of the form are three cupcakes with different frosting colors (yellow, pink, and white) and scattered heart-shaped confetti.

gar
un

cupcake ORDER FORM

NAME +
T:

ORDER
DATE:

ORDER
NO.

PICK
UP

DEL

FLAVOR	QUANTITY

ORDER TOTAL

CASH ☐

OTHER ☐

Bakery Distribution System Design

OMIS 6350 FA20

Spreadsheet/Worksheet

In the project workbook, the group has created 16 worksheets. First, a **“Welcome”** sheet is built to guide the users through the instructions to schedule the daily delivery routes that help improve delivery efficiency and maintain customer satisfaction. Next, the group created the **“Customer Info”** sheet, the **“Product”** sheet, and the **“Order”** sheet to display the information of customers (Customer ID, Name, Address, Telephone Number, X and Y coordinates, Frequency of Weekly Shipments, Corresponding TimeTable, Beginning Balance, and Current Balance), products (Product ID, Name, Unit Cost, Unit Price, Average Production per Week), and the order demand of each customer (Customer Name, Product Name, Quantity Ordered, Expected Time), providing access to all the data required to generate the daily routes. With the necessary information extracted from the former three sheets, the **“Model”** sheet was then created to calculate the expected time, decide which orders to pick, and set up the delivery plan using the Nearest Neighbor Heuristic algorithm. Besides that, the group created the **“Historical Order”** sheet, the **“Simulation Order”** sheet, the **“Replication”** sheet, and the **“Simulation Input”** sheet to test the reliability of the decision support system. By adjusting the simulation inputs, the simulation orders are generated randomly based on the historical orders of last year, each with the delivery time approximation. Calculate the on-time delivery ratio, the satisfactory delivery probability of every round of replication is known. The **“Historical Order”** sheet also helps determine the weekly schedule. Extracting the order information of the past seven days from the **“Historical Order”** sheet to the **“Weekly Order”** sheet and running the model, the group created the **“Weekly Report”** sheet to display the weekly delivery schedule, and other statistics that reflect customer satisfaction degrees. Then, in the **“Summary”** sheet, the **“Histogram 1”** sheet, the **“Histogram 2”** sheet, and the **“Histogram 3”** sheet the group concluded the simulation results statistics, and presented the on-time delivery probability distribution, the estimated delivery time distribution, and the delivery cost distribution. Finally, the **“Historical Report”** sheet is created to display the weekly analysis of last year and the customer demand analysis.

Range Name

AnchorX	The X Coordinate of Customer Address	“Model”	N21
AnchorY	The Y Coordinate of Customer Address	“Model”	O21
DCost	Delivery Cost	“Model”	O33
Delivery_Cost	Delivery Cost in simulation	“Replication”	F2
DeliveryPath	Delivery Path Anchor	“Model”	N23
EndDate	End Date of Weekly Schedule	“Weekly Schedule”	E1
Max_Expected_Time	The Upper Limit of the Expected Time	“Simulation Input”	B5
Max_Orders_Num	The Upper Limit of the Order Number	“Simulation Input”	B3

Max_Product_Number	The Upper Limit of the Product Number	"Simulation Input"	B6
Min_Number_of_Orders	The Lower Limit of the Order Number	"Simulation Input"	B4
NoDupRandN	The Generated Random Number without Duplication Anchor	"Simulation Input"	J1
Number_of_On_Time_Delivery	On-time Delivery Number Anchor	"Replication"	C2
Number_of_Total_Delivery	Total Delivery Number	"Replication"	B2
NumofDelivery	Delivery Number	"Model"	O24
NumofOnTime	On-time Delivery Number	"Model"	O27
Probability_of_On_Time_Delivery	The On-time Delivery probability Anchor	"Replication"	D2
RandN	Random Number Anchor	"Simulation Input"	I1
Random_Order_Percentage	Random Order Percentage	"Simulation Input"	B7
Round	Round Times Anchor	"Replication"	A2
Oil_Price	Oil Price per Miles	"Simulation Input"	B8
StartDate	End Date of Weekly Schedule	"Weekly Schedule"	C1
Total_Delivery_Time	Total Delivery Time	"Model"	O36
Total_Distance	Total Delivery Distance	"Model"	O30
TotalDTime	Total Delivery Time in simulation	"Replication"	E2

User Interface

When the workbook is open, the sub procedure **workbook_open** runs automatically, hiding all the other sheets and leaving the "**Welcome**" sheet visible only. On the "**Welcome**" sheet, there are five buttons, three of which at the top leading to a user form respectively. The group has altogether created eight user forms.

To manage the customer and order information, click the button **Record Management**, and user form "**frmStart**" will show up, which also contains five buttons. Follow the button caption, you can create a new customer, change and delete the existing customer profile, and enter the order information to update the order information. To create a customer profile, Customer ID, Name, Telephone, Address (X,Y coordinates), Detailed Address, and Beginning Balance are required to enter manually. To update a customer profile, users should select the customer ID in the combo box list, input the updated value and click submit. If the user wants to delete certain customers' profile, select customer ID or select customer name and click submit, and the process is completed. After the customer information database is all set, the user can click the **Create an Order** button, select the customer name and product name in the combo box, and input the expected time and order quantity to create an order. Many error checkings are involved in this process, and the

userforms the group designed made clear explanations about what to input and what not to, to facilitate the later analysis and calculation and simplify the database management.

To run the simulation, click the button **Simulation**. You can view/change the simulation inputs to run the simulation in different ways and times. If one day, the bakery decides to change its maximum order numbers, change the random order percentage if a change of order proportion is observed since the adoption of the decision system, or they decide to change other default settings, the users can always change the inputs values listed on the ***“Simulation Input”*** sheet to satisfy their needs and test the stability of the system by clicking the **view/change inputs** button. If the user doesn’t want to make a change and simply wants to run the simulation a certain number of times based on the current settings, he can click the **Run the Simulation** button, input how many times he wants to run the simulation, and click submit and the process will run automatically and guide you to the ***“Summary”*** sheets, where display the simulation results.

To view all the important dataset that the users might have interest in, click **the Worksheet Navigation** button, and you can direct yourself to view all the data in the ***“Customer Info”*** sheet, the ***“Product”*** sheet, the ***“Order”*** sheet, and the ***“Historical Order”*** sheet.

The last two buttons on the welcome sheet will guide you to the ***“Weekly Schedule”*** sheet and the ***“historical analysis”*** sheet that the group has created.

Code

The module ***NNH*** is created to generate the delivery path based on the Nearest Neighborhood Heuristics Algorithm. It contains 14 sub procedures.

First, the sub procedure ***ProcessOrder*** is built to initialize the ***NNH_model*** procedure, which calculates the daily order number and further activates the other 4 sub procedures.

Sub procedure ***ResetData*** clears all the previous data, while ***GetCustomerName*** extracts Customer Name information from the ***“Order”*** Sheet to the ***“Model”*** sheet so that the previous sub procedure ***NNH_model*** could calculate the order number, facilitating the following procedure ***ExpectedTime***, ***GetLocation*** and ***GetLoadandPrice*** to extract the expected time, customer location, the load size, and the cost of delivery given the Customer Name. The former three sub procedures can also assist the error-checking process, observing whether the information exists in the database and whether it has been successfully generated. After the cost of delivery is calculated, the sub procedure ***UpdateBalance*** will subtract the price of the new order from the current balance of customers who made an order and update the current balance of each customer.

Then, sub procedure ***NNH*** is created to find the delivery path, inside which contains two sub procedures (***GetDistance***, ***TotalTime***) and two loops to reach the goal. Sub procedure ***LoadTime*** is designed to estimate the unload time based on the load size and the given function. Assuming that unloading time will follow a normal distribution pattern with a relatively small variance, the group uses the Rnd and the given standard deviation value to generate its estimated value. Similarly, sub procedure ***TravelTime*** is created to estimate the travel time based on the “distance”. Distance value can be accessed from the Sub procedure ***GetDistance***, which calculates the Euclidean distances between each customer and the “changing anchor”. According to the Nearest Neighborhood Heuristics algorithm, the process of choosing the nearest neighbor should be repeated whenever a movement is made. Therefore, every time when the delivery man reaches a delivery point, a new anchor should be set to determine the updated nearest neighbor, until the total delivery time

exceeds the expected time. The process of updating anchor is done by the function **UpdateAnchor**. Sub procedure **TotalTime** gives the user the total delivery time by adding up values of the former two procedures.

Finally, after the delivery path is settled, the group uses the **Output** Sub procedure to count how many deliveries are done on time, and calculate the total delivery time.

The group made an assumption that 80% of the bakery orders come from organizations such as schools and small businesses, which stay unchanged, while 20% of the orders come from individual customers, which is highly random. Therefore, the Module **Simulation_Data_Generation** is built to generate the simulation order, consisting 80% of the historical orders and 20% randomly generated orders.

In this module, **Simulation_Data** is the main sub procedure, which activates the other four sub procedures (**Clear_Data**, **Count_Customer_and_Product**, **RandomOrder**, **HistoricalRandOrder**), determines the proportion and amount of the historical data and random data, and generates random integer from the minimum of order numbers the group sets to the upper limit of order numbers.

First, sub procedure **Clear_Data** clears the range of the previous simulation. **Count_Customer_and_Product** then counts the customer number in the “**Customer**” sheet and the product number in the “**Product**” sheet, which are later used to set up the range where the simulation process extracts information from.

Next, Sub procedure **RandomOrder**, consisting of three sub procedures (**GenerateCustomerName**, **GenerateExpectedTime**, **GenerateOrderName**), is created to realize the generation of the random orders. In order to eliminate the possibility of generating the same order, the group defined the **GenerateNoDupRand** function. In the function, a list of random numbers are first generated and then ranked internally based on values, after which duplicates are removed. Using this function, the sub procedure **GenerateCustomerName** randomly picks from the “**Customer Info**” sheet unduplicated data of an amount settled at the beginning. Similarly, sub procedure **GenerateOrderName** was created, selecting contents from the “**Product**” sheet, except that this time, the function is skipped in that we assume there will be the same products being ordered every day. Another assumption the group made is that the delivery man works at a maximum of 8 hours every day and the expected time of customers is at least 3 hours. Therefore, in the sub procedure **GenerateExpectedTime**, values are randomly generated from 3 to 8 only.

Sub procedure **HistoricalRandOrder** also uses the **GenerateNoDupRand** function to select the past order information from the “**Order**” sheet.

In order to run the simulation, the module **Simulation** is created, containing four sub procedures. Sub procedure **Run_Simulation** captures the value users input (no larger than 200), and passes the value into **Replication** to run the simulation for that number of times. **CollectStats** collects the simulation result, including probability of delivery, total delivery time, and delivery cost, and summarizes it in the “**Summary**” sheets. Similarly, **UpdateHistograms** presents the simulation results distribution in the “**Histogram 1**” sheet, the “**Histogram 2**” sheet, and the “**Histogram 3**” sheet.

The module **Weekly_Report** is created to generate the weekly report. The main sub procedure **WeeklyReport** first clears the content, and then call the sub procedure **GetWeeklyData** to extract the order information of the past seven days from the “**Historical**

Order" sheet to the "**Weekly Order**" sheet, run the NNH model by calling the **NNH_Model** sub procedure, and then generate the last week report by calling the **GenerateReport** sub procedure. The last sub procedure transfers the delivery path, the delivery time, the delivery cost, on-time delivery number, and total delivery number from the "**Model**" sheet to the "**Weekly Report**" sheet

The last module **Historical_Analysis** records the process of building the pivot table to help analyze the customer demand and last year's weekly revenue, profit and cost of the bakery shop.

Difficulties/ Challenges

1. Identifying the problems: In this assignment, we need to build a decision support system for a bakery, which requires us to have a basic understanding of small business delivery systems. We need to make assumptions that can help us simplify the modelling process but are also defensible in the real-business world. For example, we assume that all products are preloaded onto the truck so we can avoid the iterated loading time calculation process.
2. Using some unlearned functions: During the process of designing and coding the application, some requirements might need some additional functions that we have not learned in the class. For example, when retrieving the data from the information sheet, we need the **Find** function to locate the relevant data of a particular customer or product. Also, when randomly generating the simulation order, we need to get a way to generate non-duplicate random numbers. Even though learning new things during the project is interesting, these difficulties cost us more time and effort to reach a reasonable solution.
3. Debugging the project: Debugging becomes a big problem due to the complete online cooperation. An update in one's part of the project might cause the bug in the other's part. For example, when editing the worksheet, a single column change might lead to a bug in the simulation order generation or model processing. From this, we learned the importance of the version control when cooperating in coding, especially online.
4. Input validation: Validating input perfectly for userforms can be challenging and time-consuming. Programmers need to consider every input possibility in order to prevent users from ruining the whole program. During this process, we found that On Error Statement is practical to handle all of the unexpected errors. Besides, asking more people to sufficiently test the input validation is essential for adjusting the program to make it impeccable.