Satis.AI Coding exercise

SOLUTION OVERVIEW

Data processing

Restaurant
First of all, we process the restaurant's data, to allocate which spaces we have for cooking, assembling and packing the orders, and the time spent in every step as well as the available stock the restaurant has.

Orders
We store the orders in a data frame, sorted by date and time of arrival, so that we can process them in order.

Note: it is assumed the orders should be processed by time not by orderID (simulating a live real case scenario)

Order processing

We define an order is **pending** to be processed if it arrives at a time in which the kitchen is full and we can not process it automatically. For these cases we wait until the kitchen is free to process at least a burger, and when at least one slot is free in the kitchen then we analyze all the orders that are pending (those that arrived while the kitchen was busy) and we decide which orders (from all the pending ones) we want to accept and reject

We process the orders sorted by date time.

1. When we receive and order first we check the slot availability in the kitchen.
    - 1.1. If we have available places now, and we do not have any previous orders pending, we automatically process the order.
    - 1.2. If we have available places now, but there are pending orders to be processed, we analyze from all the orders in the queue which ones we prefer to take, in accordance to our optimization function. Once we decide what orders we want to accept and which ones we want to reject the queue is reset and the process starts again.
    - 1.3. If we cannot process the order automatically (because the kitchen is busy), we insert it in the pending list (queue), for further processing.

To process the order:

1. First we check if we have enough stock to prepare all the burgers in the order.
    - 1.1. If we have enough stock, then we check the cooking, assembly and package availability in the restaurant. Here we check if the order can be processed in less than 20 minutes.
        - 1.1.1. If we can, we update the cooking, assembly and package availability, and **ACCEPT** the order.
        - 1.1.2. If we cannot, we **REJECT** the order, and do not update any status of the cooking, assembly or packaging.
    - 1.2. If we have not enough stock in the restaurant to process it, we **REJECT** the order.

Optimization function:
1. In the scenario where we have orders queued and at least one of our slots in the kitchen get released we need to decide what orders we want to accept and which ones we want to reject (as aforementioned). In order to do this we create a list with all the posible combinations (keeping the arrival time) of the pending orders. Example:

For the case in which we have O4, O5, O6 as pending orders.
We analyze all the possible combination: O4, O5, O6, O4O5, O4O6, O5O6, O4O5O6.

2. For every order combination we:
    2.1. check if we have inventory for it
    2.2. if we have inventory we analyze if we can serve all in less than 20 minutes
    2.3. If both inventory and time past the checks we calculate a score function that determines how much we would like to take that order, the objective is to take the order combination that maximize the score.
        2.3.1. The score grows as the number of burgers grows (as the more burgers we sell the more revenue we obtain)
        2.3.2. The score is penalized depending on the products we need to (the more products in the order we have not much in the inventory the more we penalize and viceversa, in order to prevent running out of any ingredient because that would mean rejecting future orders that contain it).

The formal definition of the score function is:

$$nHamburgers - \frac{((1-\frac{nTomato}{Max}\dot{H}T + (1-\frac{nLetuce}{Max}\dot{H}L + (1-\frac{nVeg}{Max}\dot{H}V + (1-\frac{nPattie}{Max}\dot{H}P + (1-\frac{nBacon}{Max}\dot{H}B)}{nHamburgers}$$

Where:

$$Max = Max(nTomato, nLetuce, nVeg, nPattie, nBacon)$$

and where:

nHamburgers: number of burgers of the order
nTomato: tomato in the stock
nLetuce: letuce in the stock
…
HT: number of burgers with tomato in the order
NT: number of burgers with lettuce in the order
…

Example:

BURGERS: 90
LETTUCE: 150
TOMATO: 100
VEGGIE: 50
BACON: 36

We take the max of the product = 150
We calculate the penalization of each product depending on the amount of the product he have in stock

For Burgers: 1- 90/150 = 0,4
For lettuce: 1- 150/150= 0
For Tomato: 1- 100/150 = 0,33
For Veggie: 1- 50/150 = 0,66
For Bacon: 1- 36/150 = 0,76

We can see the bacon is the most penalize product because we do not have a lot, and the lettuce is not penalized at all, because it is the product with more stock.

En each order, depending on the current stock, we penalize each product with this values, and sum the total penalization of the order

3. After obtaining the final penalization in all the possible combinations of the pending orders, we process the combination with higher number of burgers - penalization (our score function). And update the cooking, assembly and packaging status in accordance to that.
    3.1. The orders that are not in the combination are **REJECTED.**
    3.2. The orders in the combination are **ACCEPTED**.

SOLUTION OUTPUT

When we process the input provided in the task we can see this is the output printed:

```
R1,O1,ACCEPTED 5
R1,O2,ACCEPTED 7
R1,O6,REJECTED
R1,O3,ACCEPTED 11
R1,O7,ACCEPTED 14
R1,O4,ACCEPTED 18
R1,O8,REJECTED
R1,O9,REJECTED
R1,O5,ACCEPTED 18
R1,O10,ACCEPTED 17
R1,O11,ACCEPTED 14
R1,O12,ACCEPTED 13
R1,TOTAL,26
R1,Inventory,83,172,166,83,90
```

In it we can see how the 3rd order in time (O6) is rejected while we could have taken it when it arrived, this happens because orders 3 to 7 (in time) arrive while orders 1 and 2 (in time) are being cooked, when the kitchen is released then we analyze from orders 3 to 7 which ones we prefer to take, and orders 4,5,6 are the ones which maximize our score function, consequently we accept those and reject the others, the same process happens with the rest of the orders.

We can see in this example how the optimization function helps us when taking the decision of taking the orders as without this function we would normally take orders simply as they arrive meaning we could not be taking the best (most profitable) solution when looking at future orders

ASSUMPTIONS

We can wait to accept-reject the order within the 20 mins we have (this has been done to simplify the problem, in a real case scenario we would set a maximum time of a few minutes)

We always process orders in time (FIFO)

We want to process orders by time not by order ID

If the kitchen is free and we can process orders we do not wait for more time, because we understand we want to maximize the usage of the kitchen

IMPROVEMENTS

We can change the FIFO processing so we do not necessarily process orders in the same order they arrive, giving us more flexibility.

Modify the optimization function based in certain criteria:

1. Increase the constant K (that multiply the penalization, by default k=1) as the time increases or as we run out of inventory
2. Use a different function at some point in accordance to any criteria

Try other optimization functions and compare with the best combination: once we know all the orders for a single day we can find the best combination (in termes of revenue) and compare with the solutions found by several different optimization functions to determine with of them performs better. In order to do that we would to run the process through a sample of daily orders for different restaurants and compare the distributions of revenue obtained in all of them. We could also do it differentiating by day of the week, country etc