M2 EEE Programming in Python

Exam Project - Coffeebar Simulation

Bitbucket Repository:
`https://bitbucket.org/Lupoll/coffeebar_exam2/src/master/`

Time stamp: 921c96b

Luca Poll

November 16, 2020

1 Objective

For the project we were given data for a coffee bar (in the following Cafe) that sells drinks and food to different type of customers (returning and one-time) at different times during the day. The customers could be identified through unique IDs, while every purchase contained one drink and optionally one food item. The individual purchases done by the customers had a time stamp and occurred during a period of five years at daily repeating time slots between 8:00 and 18:00 o'clock.

Given this data, we were asked to obtain the probabilities of a customer buying a certain drink item and food item for the respective time slots during a day. Additionally, we were given the information that the type of customer could further be divided into 'normal one-time', 'tripadvised', 'normal returning' and 'hipster', where the former two are one-timers and the latter two returning customers. The different types have different assumptions about the probability of being drawn in the "customer lottery"[1], as well as different initial budgets and marginal willingness to give tips. Combining the computed probabilities with the given information, we were asked to build a simulation that accounts for the choice probabilities and the customer lottery while keeping track of the remaining budget and the past purchases of the returning customers. Furthermore, we assumed that once the budget of a returning customer was not enough to buy the most expensive possible combination of food and drink item, he/she was to be excluded from the customer lottery.

2 Simulation

2.1 Analyzing the choice probabilities

In order to model the simulation, we first did some basic analysis of the given data. Some interesting insights included that coffee was mostly sold in the morning, while soda was the most common drink item during lunchtime (Figure 1a). Before 11am, the customers did not consume any food item while during lunchtime sandwiches were the most prominent choice and in the afternoon the majority again consumed no food item (Figure 1b). Overall, soda was the most sold drink[2] and sandwich the best selling food item[3]. Interestingly, all customers that consumed sandwiches during lunchtime were also buying a soda.[4]

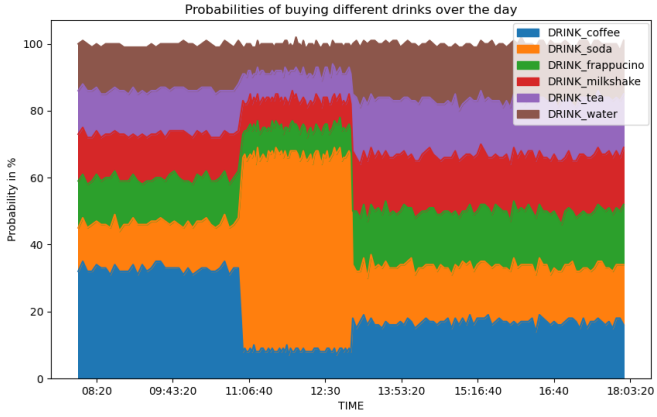
2.2 Methodology

Having investigated the given probabilities, the next step was to create from these choice probabilities combined with the assumptions about the customer lottery the structure for the simulation. In order to do so, the object hierarchy in Figure 2a had to be translated into the model as well as

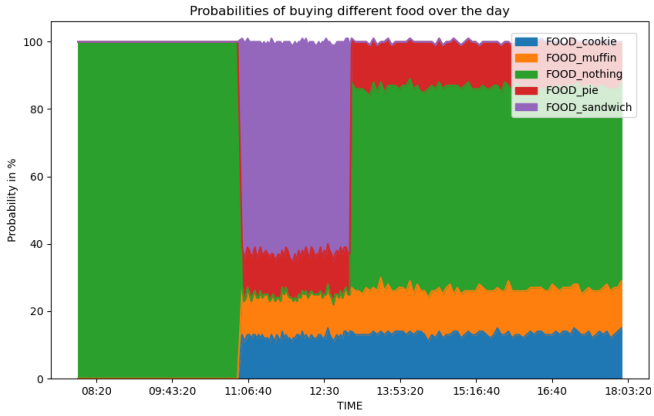
[1] "customer lottery" refers to the lottery of which type of customer will be chosen to do the purchase (with the respective structure and probabilities we were given in the exercise).
[2] Followed by coffee; water, milkshake, tea and frappuchino identical sale numbers.
[3] while pie, cookie and muffins were sold in the same quantities.
[4] Most of the graphs investigating the choice probabilities are excluded due to the limited space (five pages) in this report.

Figure 1: Item selling probabilities

(a) Drink items



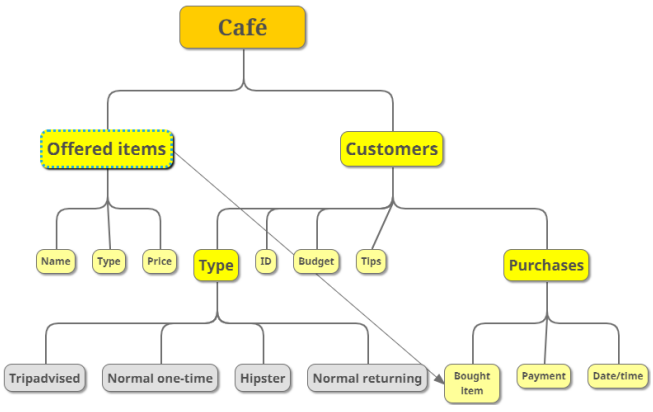
(b) Food items



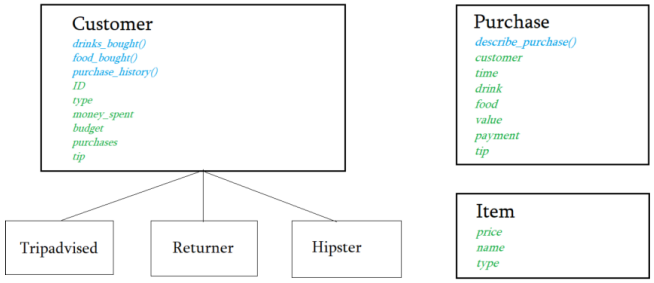
interactions between these instances had to be made possible. To account for that, declarative programming in the form of functional programming (FP) was combined with imperative, object oriented programming (OOP). The OOP allowed to define classes from which objects with respective methods and attributes (as shown in Figure 2b) could be created, while FP was applied to create an easily scalable environment for these objects to be combined, updated, included/excluded or interacted.

Figure 2: Hierarchies

(a) Object Hierarchy



(b) Class Hierarchy



2.3 Structure

The structure of the model is the following: The function *SimulateRange()* is called, which takes the choice probabilities per time slot, a list of returning customers (objects), a list with the defined item objects and optional start/end dates (default is five years) as arguments. Within this function, two other functions are called for every time slot: *ChooseCustomer()* and *MakePurchase()*.

ChooseCustomer() takes as argument the passed list of returning customer objects and firstly subsets this list to returning customer objects whose budget is larger than 8 EUR ("liquid customers").

If this list is not empty, and hence we still have liquid returning customers, a returning customer from this list will randomly be chosen. This returning customer will participate in a lottery against a 'normal one-time' and a 'tripadvised' customer with respective probabilities to be chosen of 20%, 72% and 8%. If the list is empty and we do not have liquid returning customers, we need to make an additional assumption so that the code is scalable and does not crash. Namely, the respective customer will in this case only be chosen among 'normal one-time' (90% prob.) and 'tripadvised' (10% prob.).

MakePurchase() on the other hand takes as inputs the time slot, the chosen respective customer object, the choice probabilities as well as the list of item objects. Taking into account these parameters, a purchase object is created. In the process of creating this purchase object, the relevant choice probabilities for the given time slot are selected in the first step. Then, the list of item objects that are sold at the Cafe are matched with their respective probabilities of being chosen and two lotteries (one for the drink item and one for the food item) are conducted in order to determine the drink and food item choice. Drawing from the item objects attributes 'price', the value of the purchase is determined. This value combined with the tip given by the customer (depending on the customer attribute 'tip') yields the payment of the purchase. Assigning these values among others as attributes to the created purchase object, allows us then in the next step of the *MakePurchase()* function to update the customer object attributes keeping track of the 'money spent', the remaining 'budget' and the 'purchase history'. This timely updating is essential since the chosen customer objects attribute 'budget' will be evaluated during the next iteration in the *SimulateRange()* function when determining the customer through *ChooseCustomer()*.

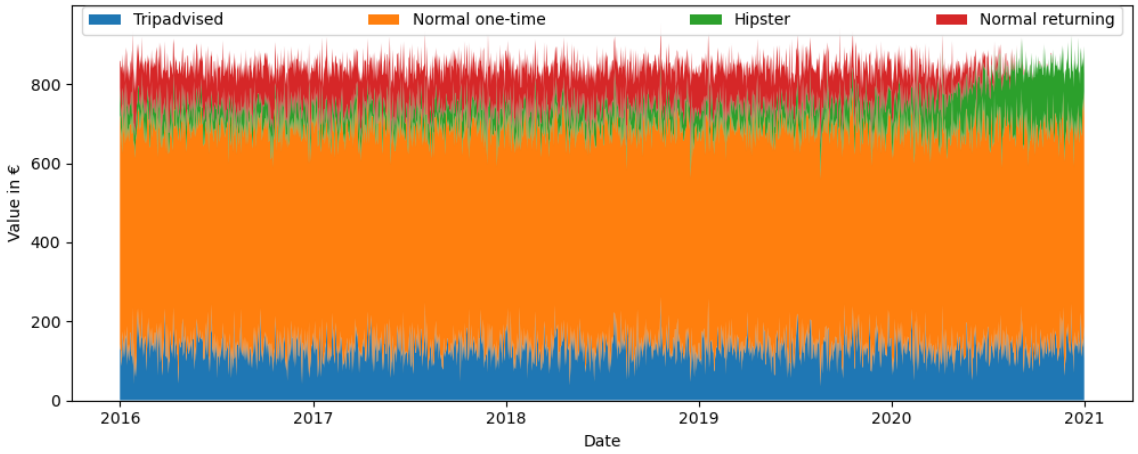
Since the iteration through all time slots for the specified time range in the *SimulateRange()* function takes some time, you will be asked when running the code if you want to run the simulation (takes about one hour with 8GB RAM) or if you want to load the simulation from a pickle file[5]. The result of the simulation, displayed as the aggregated expenditure ('payment') per day by type of customer, is shown if Figure 3. The graph clearly shows the differences in the probabilities for the different customer types being chosen as the respective customer. Also, we can see that towards the end of the simulation the 'normal returner' customers ran out of money and were replaced by the 'hipster' customers.

2.4 Variation in parameters

A big advantage of the chosen structure for the simulation is the easy scalability and the adjustment to a change in parameters like the number of returning customers and their attributes, the product range offered by the Cafe, the prices if the items, the customer lottery, etc.. In the following, some of these parameters will be adjusted and the necessary changes in the code as well as the effects on the outcome will be shown in order to demonstrate this scalability.

[5]A pickle file allows in contrast to a .csv file that the objects can be accessed as objects again when loaded instead of being converted to strings

Figure 3: Aggregated turnover per day by group



Change in number of returning customers to 50

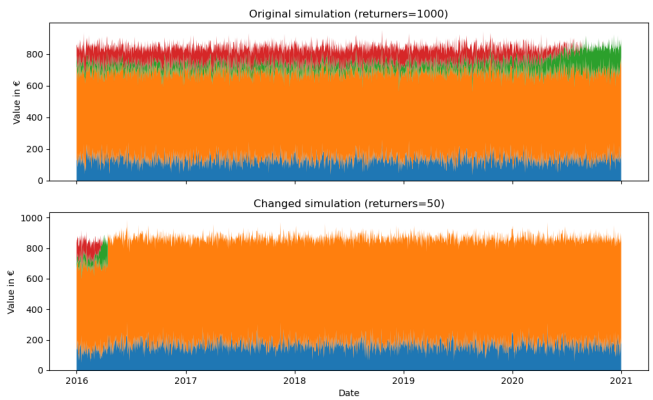
The adoption of this change is straight forward. The list of returning customer objects needs to be changed to not contain 667 'normal returning' and 333 'hipsters', but instead 34 'normal returning' and 16 'hipsters'. Since we previously made the additional assumption that if all returning customers are bankrupt, they will be replaced by 'normal one-timer' and 'tripadvised' customers. In Figure 4a we see that rather quickly all returners are out of money and the customers are solely one-timers.

Increase in prices in 2018

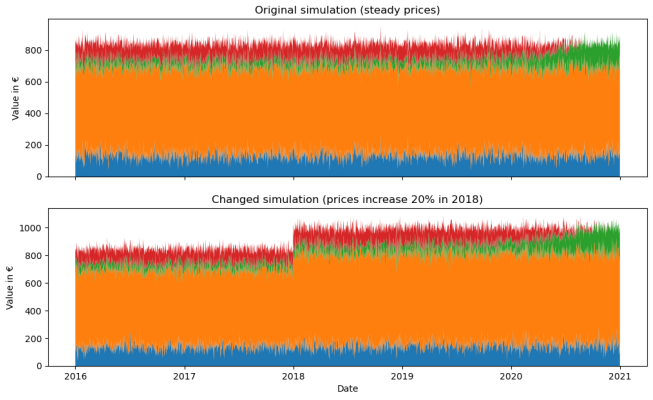
In order to account for the price changes, the *SimulateRange()* and *MakePurchase()* functions need to be adjusted. The date of the day that is modeled should be taken into account in the *MakePurchase()* function and determined whether this day is before 2018 or not. If yes, the purchase attribute 'value' is multiplied by 1.2 which reflects the prices increase of 20% for both types of items. In order to be able to define the additional input to the *MakePurchase()* function, it needs to be passed in the *SimulateRange()* function. In Figure 4b we can observe the erratic increase in daily turnover from 2018 onward.

Figure 4: Aggregated turnover per day by group variations (I)

(a) 50 returners



(b) Price change

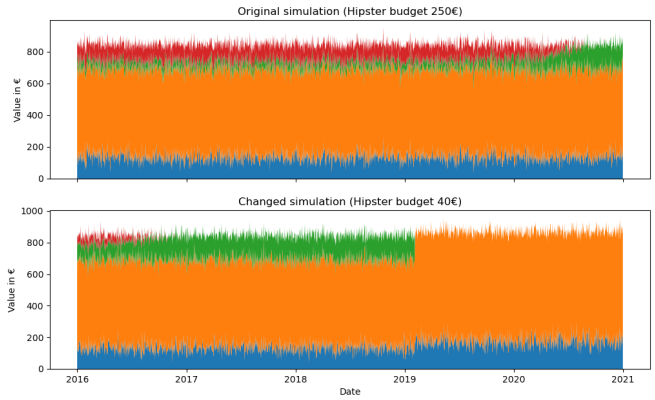


Change in budget of hipsters from 250 to 40

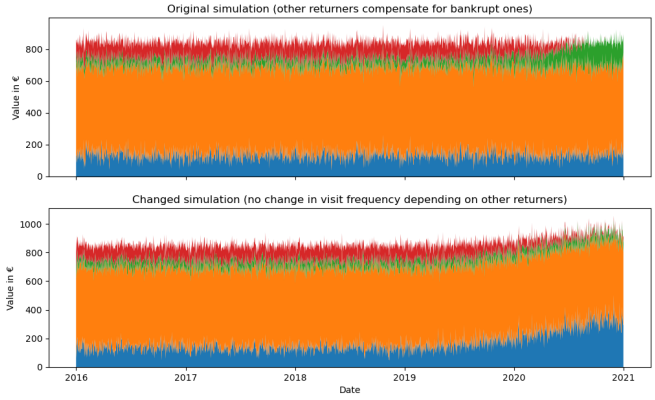
Also this change is straight forward, since we only need to loop through the list of returning customers before we run the simulation and update the respective customer object attribute 'budget' for 'hipsters' to 40 (instead of the default 250). From the graph in Figure 5a we see that the 'hipster' customers are bankrupt quite fast (note that colors of 'hipster' and 'normal returning' are exchanged in this graph) and are replaced by 'normal returner' customers which are also all bankrupt after a while.

Figure 5: Aggregated turnover per day by group variations (II)

(a) Hipster budget change



(b) Customer lottery modified



Change in assumptions about customer lottery

A further interesting topic to address is the customer lottery. In the setting where a returning customer object is randomly chosen from the list of liquid returning customers and later has a 20% probability of being chosen as the resulting customer, we implicitly make the assumption that once a returning customer is bankrupt, another liquid returning customer will visit the Cafe more often (the respective probability to be chosen among the returning customers increases when less returning customers are liquid). Since this assumption is rather unrealistic, we can modify the customer lottery the following: Initially, when all returning customers are liquid, the probability for an individual returning customer to be chosen as the resulting customer equals 20% divided by the number of returning customers (hence 0.02% if 1000 returners). This determined probability is solely dependent on the overall number of returning customers, not on the number of liquid customers. We now make the realistic additional assumption, that if a returning customer is bankrupt, he/she is replaced by a 'tripadvised' customer. This is due to the fact that the Cafe is likely to have good ratings since customers have spent their entire budget there, which will lure more 'tripadvised' customers into the Cafe. This change in customer type has furthermore a beneficial effect to the Cafe itself, as 'tripadvised' customers are the only customer type that give tips. From Figure 5b we can see two different things: Firstly, all customer types are represented throughout the whole simulation (since there is no compensating) and secondly, the daily aggregated turnover increases over time which is due to the tips given by the additional 'tripadvised' customers.

3 References

Since the coding of the fist Python project is challenging when coming from R, intense internet research was necessary in order to code the simulation. Most prominently, forums or individual blogs depending on the posed problem were consulted. Furthermore, for specific functions and examples, the documentations of the packages have proven helpful. The following urls are examples of such pages:

- https://stackoverflow.com/
- https://www.geeksforgeeks.org/
- https://www.digitalocean.com/community/tutorials/understanding-class-inheritance-in-python-3
- https://pandas.pydata.org/pandas-docs/stable/index.html
- https://matplotlib.org/
- https://python-graph-gallery.com/