Python Programming: Coffee Bar Simulation

November 14th 2022

Paper writer: RAVEY Bruce

Code writers: RAVEY Bruce, Daniel Lopez

Time of the bitbucket timestamp:

We furnish here the timestamp of the code. We did some changes in part 2, when we did the part 3. Indeed, we noticed that writing budget, foods, drinks and tips in the "__init__" function of our class leads to have the list of drinks, foods, budget or the tip shared by all the instances. So, we had to change and write them down in the "self.variable = variable". We advise you to look at the last commit in order to do not see error that were corrected after the simulation.

Link to the repository: https://bitbucket.org/barucee/ravey_lopez_python_eee/src/master/

Timestamp part 1 & 2: 6b0a09b - https://bitbucket.org/barucee/ravey_lopez_python_eee/commits/6b0a09ba9f0dba3910f4de5270993bc60661b873

Timestamp part 3: 7010726 - https://bitbucket.org/barucee/ravey_lopez_python_eee/commits/7010726d4ef512417b96763985d67c864359039a

Timestamp part 4 + Final commit: e21ad73 - https://bitbucket.org/barucee/ravey_lopez_python_eee/commits/e21ad73c4bc42e61f3f1287d298670a146dde4e8

Table of contents:

Description of the project:

This paper presents the methodology and the results of the project given as part of the course of Python Programming. I will also present the problems that we encountered along my explanations

In this project we try to replicate the data set of different customers of a Coffee bar day-to-day over 5 years. The bar receives for some time stamp one customer and he buys a drink and sometimes a food.

In the first part, we will explore the data set given by giving some information about the data set and doing some charts. Furthermore, we will calculate the probabilities for each product at each time stamp in order to do the simulation. Secondly, I will present you the classes and the functions of the customers we have created in order to be able to simulate the coffee bar selling. Thirdly, I will present you the class of the coffee bar which allow us to do the simulation. Fourthly, I will present you some refinements of the part 2 and 3 in order to do the additional question and some additional graphs. Finally, I will present some additional graphs.

I – Exploratory Analysis:

The first step has been to read the data set thanks to an abstract path [1] and do some data visualization on the data set given.

First, we noticed that the column "TIME" was an object. We needed to transform it as "DateTime" type in order to better handle it, so we changed it [2]. Then we looked at if there was missing values on this dataset. We noticed that there was no missing value for all the columns except for the column food, which means that a customer does not always buy food.

After that, we try to answer the question 1. In order to have a list of the unique values of the columns drink and food we use the function "unique" [3]. Then, we print the products inside a sentence [4]. About, counting the number of unique customers we use the function "nunique" [5], we also print it in a sentence [4].

For the question 2, we have to plot the number of foods/drinks for each year over five years. We use the group by and count function [6]. Then, to plot it we use the library "matplotlib" with the bar function [7]. Because we could not see the difference of products sold, we changed the beginning of the y-axis [8]. We also display the count above the bar plot [9]. To finish, we export the graphs in our result directory [10].

[Figure 1 & 2 Here]

The question 3 has been little bit trickier because we first tried to get the probabilities in one line of code. However, it has been very difficult to find a function which does that. So, we finally did in more lines thanks to the function group by and lambda function [11]. Furthermore, it did not give us the "rest" of probabilities when the customer does not buy, the "random.choice()" function in part 3 was not functioning well. So, we had to fill in the missing value by a string with one space, in order to have a column [12]. Moreover, we did not manage to find a loop which was able to extract these probabilities from this 2 stages index (Time as first index, and then type of products as second index). So, we found a function to display the second index as column: "unstack()" [13]. Finally, to avoid to do a loop on two different data frames, we thought that our code will be quicker if we merge the two probabilities data frame. So, we use the function "concat()" to merge these two data frames [14]. To finish this question, we save the data set of probability created in order to use it in our simulation python file [15].

II – Creation of the customer classes:

First, we created the different classes of our customers [1][16]**. We first created classes with only ID and budget.** Then, we created the subclasses of recurrent customer, we let the budget empty, and then create the two subclasses of recurrent customers: regular recurrent and hipster. We just changed there the value of the budget. About the one-time consumer we created a subclass "Only Once" with the same budget because it is shared by the "two subclasses". After that, we created only the TripAdvisor with the tip, as done for the Erasmus student in the course [1]. To simulate the tip, we use the "random.uniform()" function and to simulate only 2 decimals we use the "round()" function [17].

The budget was inside the "__init__" function without any list, but then we read again the instruction and it was said that the customer was able to "say" each drink and food he bought and furthermore the amount he paid. So, we decided to create an empty list of food, drinks and budget, in order to keep a track of what they did. It has been tricky to do that, because we first thought that it would take an empty list as empty and not take it as a real initialisation. **However, we let it inside the "__init__" function, the biggest error we did. Indeed, during the simulation, food, drinks and budget were shared by all the customers and so could not simulate all of it because we were out of budget very quickly. It took us 3 hours to understand that it was our class functions which were not written well [18]. Indeed, we had to move out the budget, drinks and foods from our "__init__" function and put it only below in the: "self.variable 1 = variable1".**

We then had to create the methods of the customers. We first create a method which allow us to tell if the customer is in budget or not. Not very useful for the one-time customer, but in case we still use it. We use the max function in order to get the maximum of the prices [19]. Then we return a True or False value. **First, we would like to change the statement of the object directly with the "__bool__" [20]. But it did not function, so I asked a question on Stack Overflow[21]** and they advised us, to return the true or false value with a return statement. They also advise us to use the "super()" function in our initialisation, so we did. We compare the maximum price of food and drinks with the last item of our budget for the returning customer, the last one which has been appended. **We also thought about doing the calculus of maximum price only one time and keep it in order to save time of computation but it would lead to a problem for a question of the part 4 when we change the prices at the middle of our simulation**.

We also created a method which give us the purchase history of each customer, with some refinement for each subclass. For the classic Only Once customer, we print inside a sentence, ther ID, and the first element of their budget, drink, food, and the last element of their budget in order to get the difference between the difference between their two budgets, so we have the amount they spend. We add an "if" statement in order to avoid to have an empty sentence when the customer does not buy any food. For the TripAdvisor, we had the amount of tip they paid. Finally, for the returning customer we do a loop over the list of drink in order to print all of what they bought.

After that, we created the buying method**. First we did in the simulation part (part 3), nevertheless after reading the instruction it was said that it is the customer "who is able to buy", not an "append()" of the choices inside the different lists from part 3.** So, we use the "random.choices()" function [22] in order to have the item from a list depending on the probability. The result was given inside a list, so I had to look at on internet how to remove the result inside a list [23]. We first used the "random.choices()" function with pandas data frame, but we finally did with dictionary **in order to speed up the run of the code**, it will be explained in the next part. Then, we append the different results in the history lists (foods, drink, time, budget). For, the food and drinks it is just appending the result. For the time, we use the two variables: date and time which comes from the loop of the simulation (part 3). We use the "dt.datetime.combine()" function [24] in order to combine the date and the time of the loop and "append()" it inside the time history. We also had to do a treatment of the time because it comes from a dictionary, so we used the "dt.datetime.strptime()" function in order to precise that it is a time type. About, the budget we use the last budget in our list, so the budget he had before coming to the coffee bar and we deduce to it the price thanks to a dictionary which contains each price. For the TripAdvisor customer we use the same function but we also deduce to the last budget the tip given.

[Figure 3 & 4 Here]

III – Simulation class and functions:

We have to create a simulation class in order to avoid to write each time the same code for each additional question of the question 4. So, we created a class simulation with attributes which will be used to simulate our coffee bar. We try to implement the maximum of value to our attributes in order to do not have to write them when we launch the simulation. If we change something in part 4, we simply write over the initial value in order to change the initial value. **We changed that when we began the part 4.**

About the methods, we had to create a function which give us ID. To do that, we first create a function which give us a random ID [26]. We use the "random.randint()", to give an integer between 0 and 9. The all under a loop of 8 in order to have 8 integer. We then stick it to the CID integer.

After that, we create a function which allow us to be sure that this ID has not been already given [27]. So, we create an empty set, **not a list because it is faster to do a loop in a set** [28], in order to keep a list of all the ID already given. Then, when we create a "while loop" in order to do generate a random ID while it is inside the set.

We had to create the functions which allow us to say if the customer which comes to restaurant are from a type or another. **Probably, the more interesting part because we had to play with a lot of if statement, but also the more exhausting because we had to think about all the possibilities.** We have thought this part in **backward induction**. I mean we create first the functions of random choice between the subclasses and then create the function between Only Once customer and recurrent.

First, we create the function which tells us randomly if it is a "Classic Only Once" or a "TripAdvisor" one. To do that, we compare a random probability between 0 and 1 with the probability given by the instruction [29]. Depending on the output of the preceding if statement, we generate a "Classic Only Once" or a "TripAdvisor" by giving it a random ID which has not already been attributed thanks to the function previously presented. We do a while loop in order to do again the generation of the customer if he is out of budget. It is not very useful because the "one time" customer will always be in budget, but we did it in case we had to change the budget of these customers.

Secondly, we had to create a thousand of returning customers. We first tried to choose randomly an "once customer" which becomes a returning customer, but we would have to change their budget and the class returning customer would not be used totally. So, we have thought about creating before the simulation a "pool" of returning customer. Then, a "Recurrent customer" will be picked randomly depending on the probability given. Like the last part, we do an if statement with the probability given by the exam [29], and we create a "Regular Recurrent" or a "Hipster". For the hipster, we **append his budget here in order to do the question 5 of the part 4.**

Finally, we create a function which generate randomly all kind of customers. We do, as done in the other part, a random choice between a random variable and the probability given in the exam between a "recurrent" or a "only once" customer [29]. We also check that the "returning customer pool" is not empty [30], otherwise we generate only one time customer. Furthermore, we have to delete from our pool of customer the customer who are out of budget, in order to avoid to run long time the "random.choice()" function in the while loop when the list is close to be full of "out-of-budget customers". To do it, we first append the "out-of-budget customer" inside a new list in order to still be able to display his purchase history. **Then, we do a loop over the length of the "pool of returning customer" and we delete the customer. We had a problem on this loop, because the loop was keep going after the deletion and the list was reduced so an error occurred because the last element of the loop was out of range [31]. To avoid this problem, we stop the loop after the deletion, it makes also the code speeder.**

Our code was long at the first runs, it took us 7-8 minutes on my computer to run the code. It was not practical to work and do some small change. So, we looked at on internet how to speed our code. On internet they said that the loop was speeder **if we used dictionary [32]**, so it is what we tried to

do. Simply transform the data frame as dictionary did not allow us to launch the simulation. Indeed, the "random.choices()" function [22] from our buying function (part 2) need a list of probability. **The dictionary given was of the form: {…, Coffee: {0: 0.xxx, 1: 0.yyy ….} ….}, so, we had to change the form of our dictionary in order to have something which gives us a list of probability for each timeslot**. To do that, we use a loop on the length of the dictionary, and we "append()" to a list the different probability**. Finally, we put these list in a new dictionary, one for the drinks and one for the food.**

Now, I will present you the hearth of our simulation, the function "fill_in_Dictionary()". We first fill in the "pool of returning customers" thanks to the function "create_pool_of_returning_ customers()" and extract the probabilities into a dictionary thanks to the function "extract_ probabilities_as_dictionary()". Then, we do two loops: **one while loops over the date** [33][34], between the two dates implemented as attribute of the simulation class and **a"classic" loop over the different timeslot of a day**. In these loops, we generate randomly a customer thanks to the function "generate_customer()". After that, we make him buy the items and "append()" the different results in the output dictionary which will become after a data frame. At the end we append the "recurrent customers" of the pool who did not finish their budget to the output pool of returning customers.

To finish, we create methods functions which allow us to transform the output dictionary as a data frame [35]. First, the "purchase_history_of_one_returning_customer()" function which create the same kind of data frame as our input data frame, I mean time, ID, food and drinks columns. Furthermore, in this function we save the data frame in a csv file [15]. Secondly, the "data_frame_with_extra_column()" function which allow us to have the revenue for each time slot in addition to the classic columns.

In the Simulation file we proceed to the simulation of the class simulation. We first create an object simulation and input the name of the file where we want the data frame output is saved. We then use the 2 methods created before: "fill_in_Dictionary()" and "transform_in_Data_Frame()" to save the data frame output. Furthermore, we will plot the revenue for each day. So, we use the method "data_frame_with_extra_column()" to get the revenue column. Finally, we plot with a line graph the revenue [7] and the number of "returning customers" still in budget.

[Figure 5 & 6 Here]

IV – Additional questions of the part 4:

For the first question, we simply simulate the data frame as explained in the part 3. Then, we use a method which has not been already presented: "purchase_history_of_one_returning_customer()" method. It allows us to randomly choose a returning customer in our returning output list and print his purchasing history thanks to the method "purchase_history_of_one_returning _customer()" from the class customer.

[Figure 7]

For the second question, we first count the number of returning customers. To do it, we create a data frame with only duplicated ID in order to have only returning customers thanks to the function "duplicated" [36]. Then, we count the number of returning with the function "nunique()" [5] and we print it in a sentence [4]. To display, if there is time where they show up more and get the probability of having a type of customer, we group by hour/timeslot and count the number of recurrent [6], and we also group by and calculate probabilities of returning and once customers to come [11]. After that, we plot them using the library "matplotlib" with the bar and line functions [7]. We also display the count above the bar plot [9]. To finish these plots, we save them [10].

[Figure 8 & 9 Here]

The returning customers come more less during lunch, so they are present when food is less sold. Their purchasing history is so affected by a lower probability to have food than the "only once

customer". We do a Chi-square test to check the correlation between the type of customer and the item they buy [37]**. We accept the null hypothesis, so the choices are uncorrelated**. If, we look at directly the probability we can see that it is not correlated. Indeed, for argument's sake**, the probability that a "one time" customer buys a coffee is 0.13 and a "returning" buy it with a probability of 0.32. We can also quote the sandwich which has a probability of 0.45 and 0.25.**

For the third question, **we simply write over the number of returning customers** while we create the simulation object.

We supposed that the revenue would increase. Indeed, there is less returning customer which will come, so more once customer will come**. So, more TripAdvisor will come, and more tips will be given**, let's check if it is true. At first sight, it is not a big change but we can see a small difference.

[Figure 10 & 11 Here]

For the fourth question, we change the price of the items at the middle of the simulation. To do it, we have implemented in the "fill_in_Dictionary()" function of the simulation class an if statement after the while loop of the date. This if statement says that if the date is equal to the date which is implemented during the creation of the object, the price change depending on the value implemented. **We filled in the attribute as none in case there is no "TimeChangeOfPrice", in order to have an "optional value"** [41]**.** To do that, we use the "update()" function of the object dictionary [38].

[Figure 12 & 13 Here]

For the fifth question, we do as the third question, we simply write over the "__init__" function of the class by giving another budget during the creation of the object. Then, we simulate as usual with the "fill_in_Dictionary()" and "transform_in_Data_Frame()" methods.

[Figure 14 & 15 Here]

For the sixth question, we noticed that the pool of returning customer were not empty, so the hipster was not using all of their budget. So, the coffee bar asks a question to Ravey & Lopez Economics consulting group what could they do to make them use all of their budget. Our first thought was that they are recurrent customer so they have a habit to come to this coffee bar. Furthermore, hipster like using new technology. So, we advise them to do a partnership with UberEat in order to deliver foods and drinks to hipster and make an UberEat advertise inside the coffee bar. We assume that at the same time that a classic customer comes a hipster can order from ubereat, so we can now have duplicate of timeslot. To do it, we use an if statement with the probability that an UberEat customer comes [29] and if there is at least one hipster in the "returning customer pool" with a budget enough high [39][40]. Then, as done in the simulation (part 3] we pick a hipster customer randomly [22]. We check that it is a hipster [40] and that he is in budget thanks to a while loop. We also delete it from the list if he is out of budget. Finally, we make it buy thanks to the "buy()" method of the customer classes, and append the results in the dictionary output. As expected, the number of returning customers decrease. As done for the change of price "Q4", we implement an "optional value" for the Uber Eat variables [41].

[Figure 16 & 17 Here]

V – Additional graphs:

We also have plotted additional graph; we plotted the number of items per week day and month. Furthermore, we plotted the total number that each item has been sold [7] [8] [9].

[Figure 18, 19, 20, 21, 22 & 23]

Moreover, we plotted the probabilities for each kind of item per timeslot [7].

[Figure 24 & 25]

Annex:

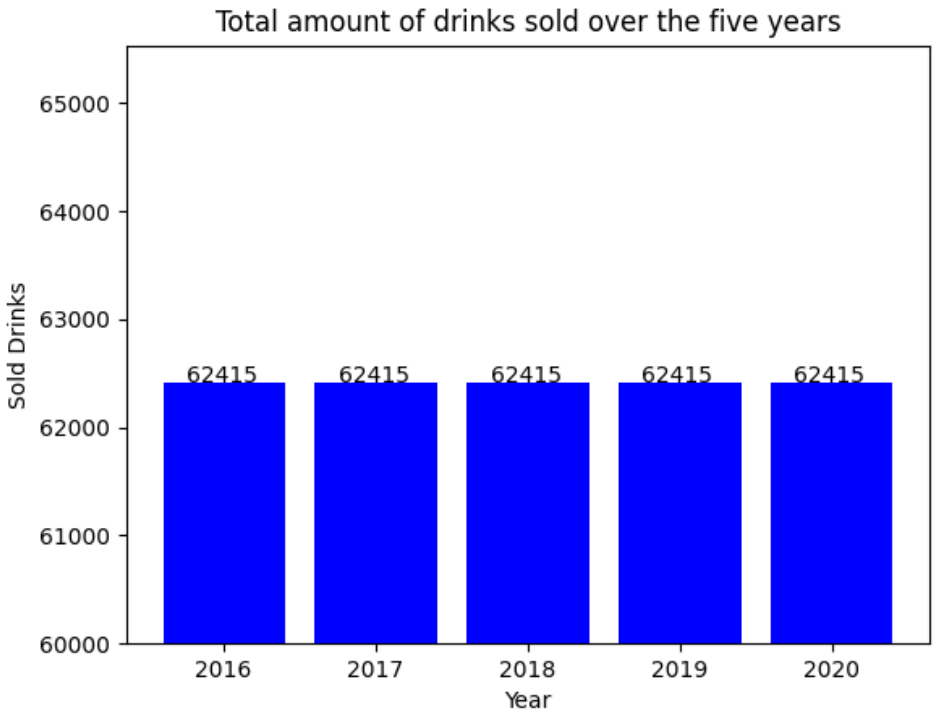


Figure 1 above - Number of drinks sold per year:

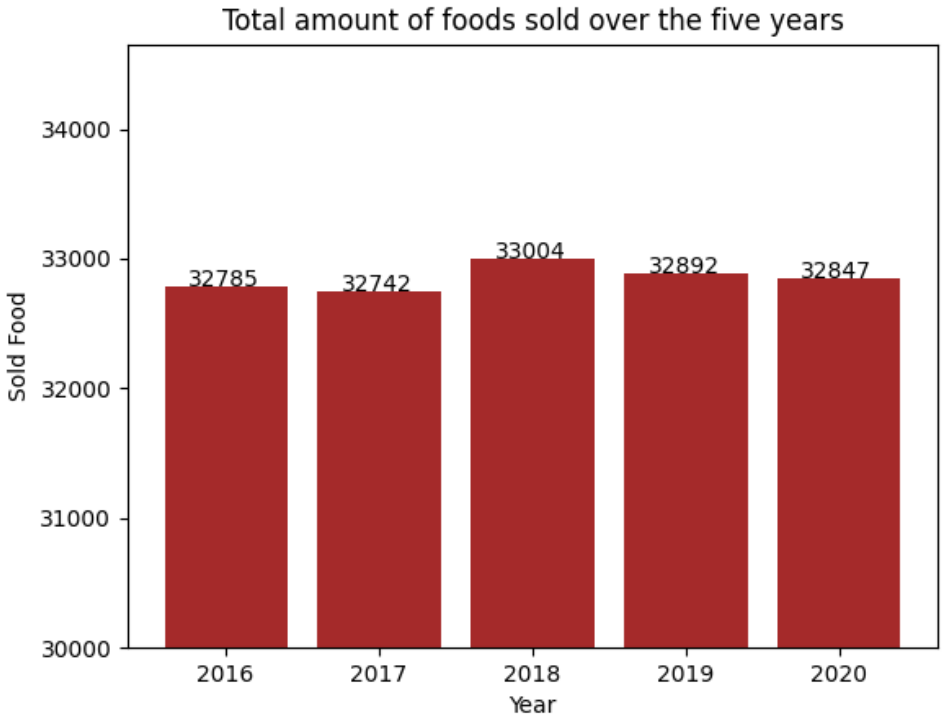


Figure 2 above – Number of foods sold per year:

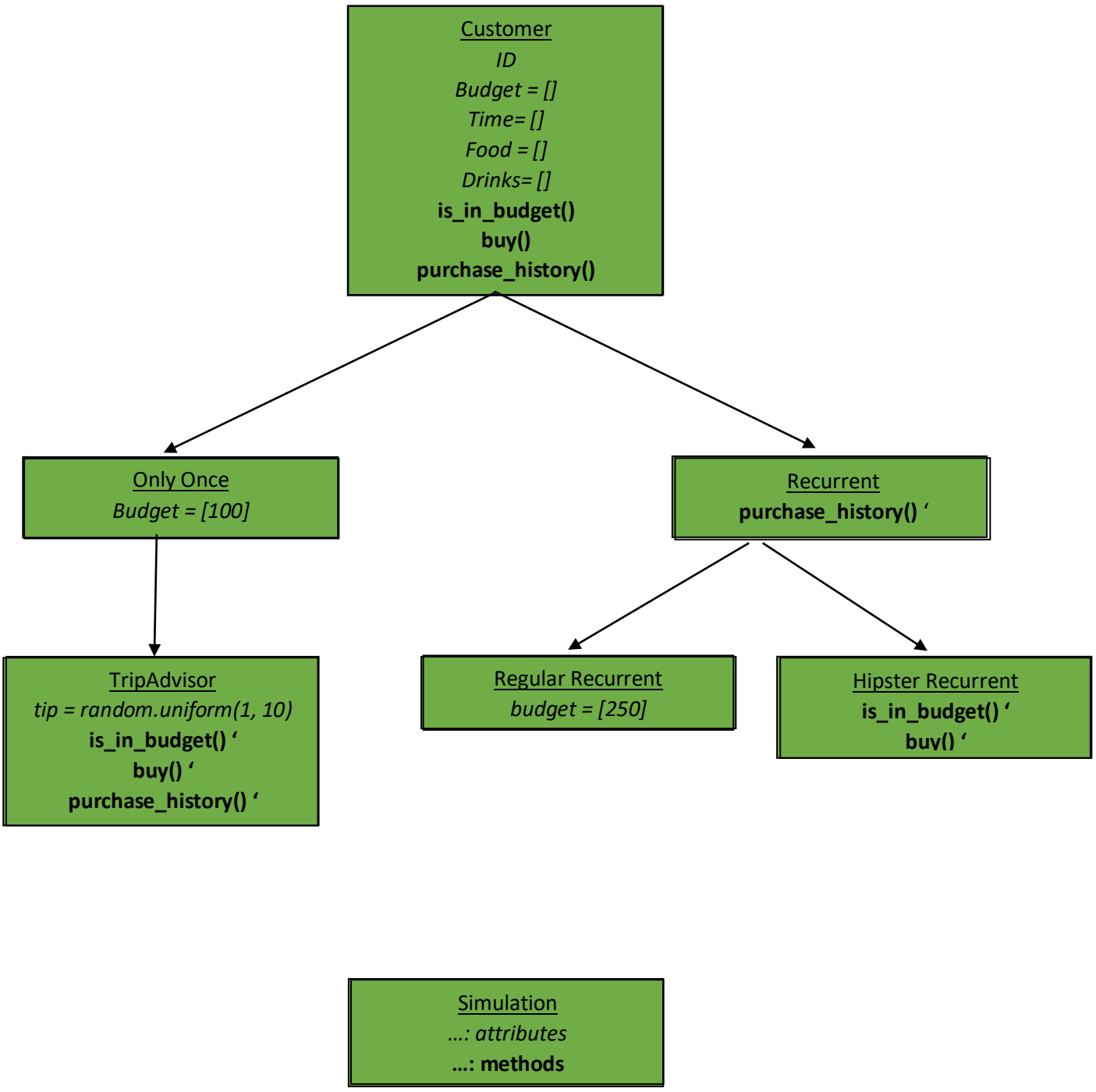
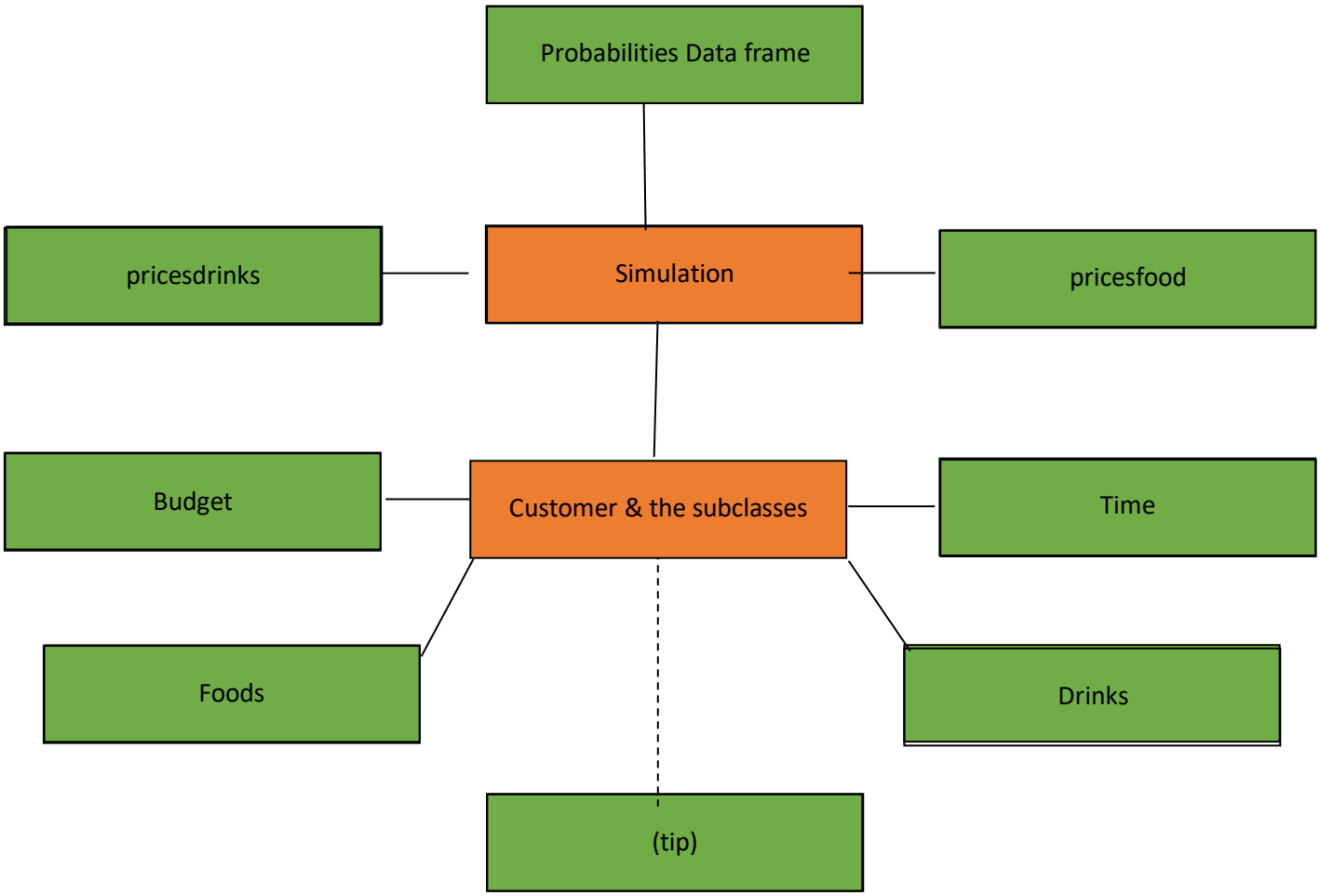


Figure 3 above – Class Hierarchy



Classes
Attributes needed

Figure 4 above – Object Hierarchy

Simulation Part 3

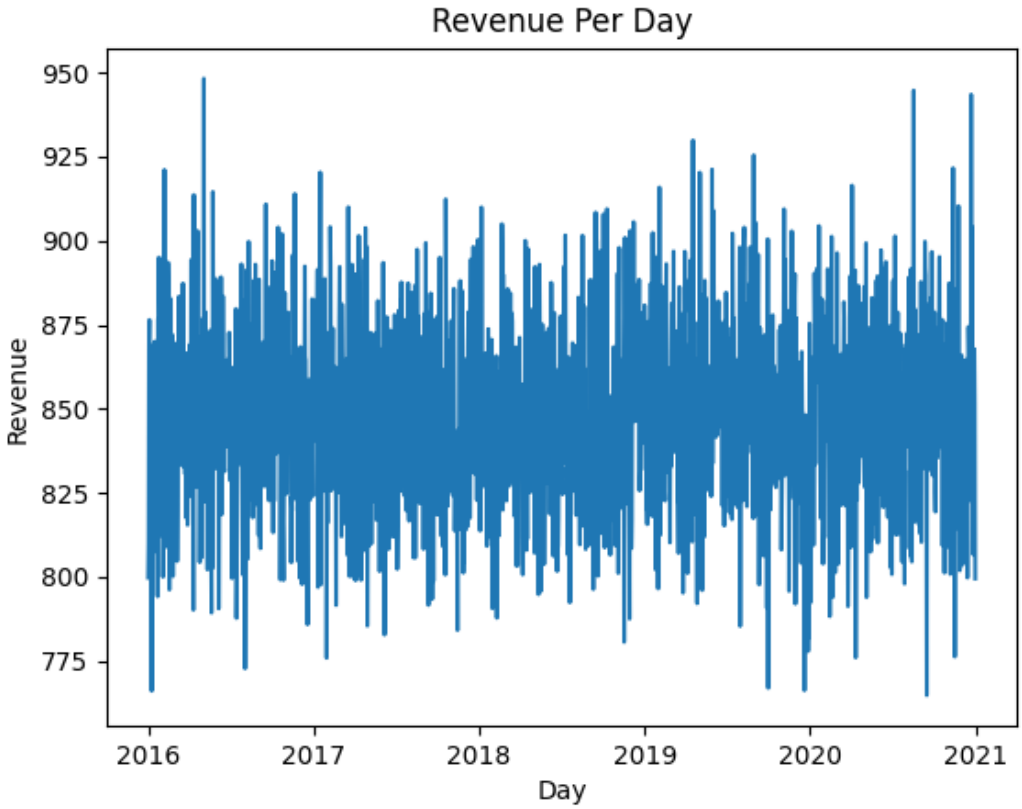


Figure 5 above – Revenue simulation part 3

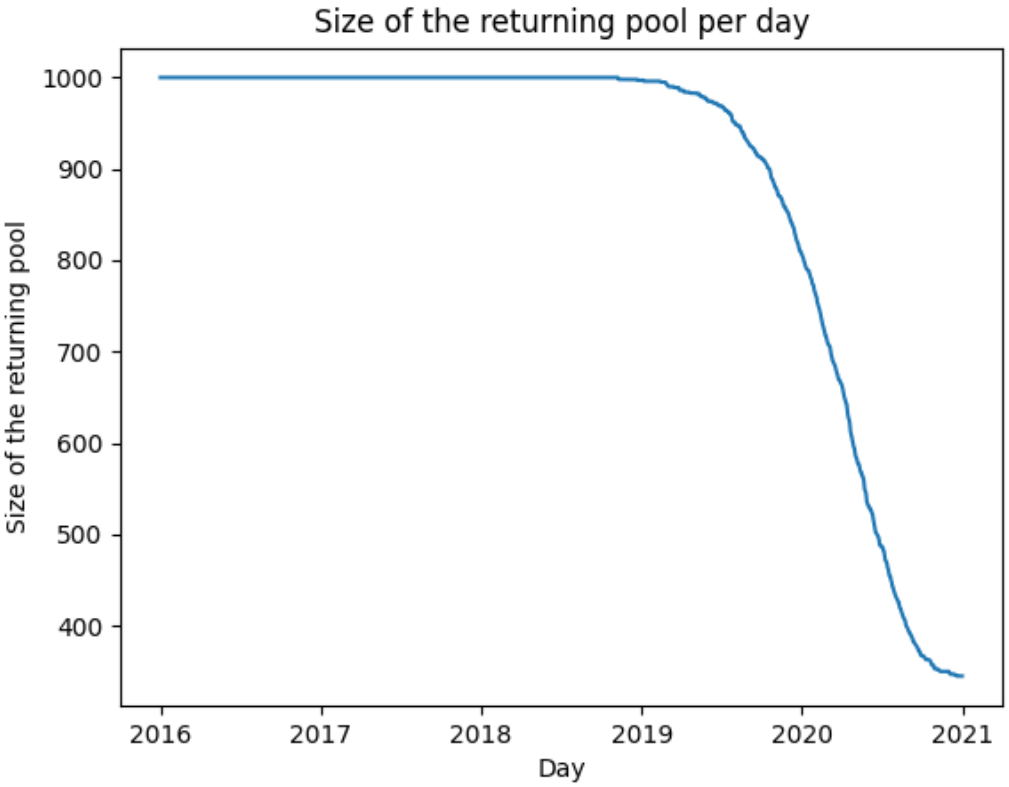
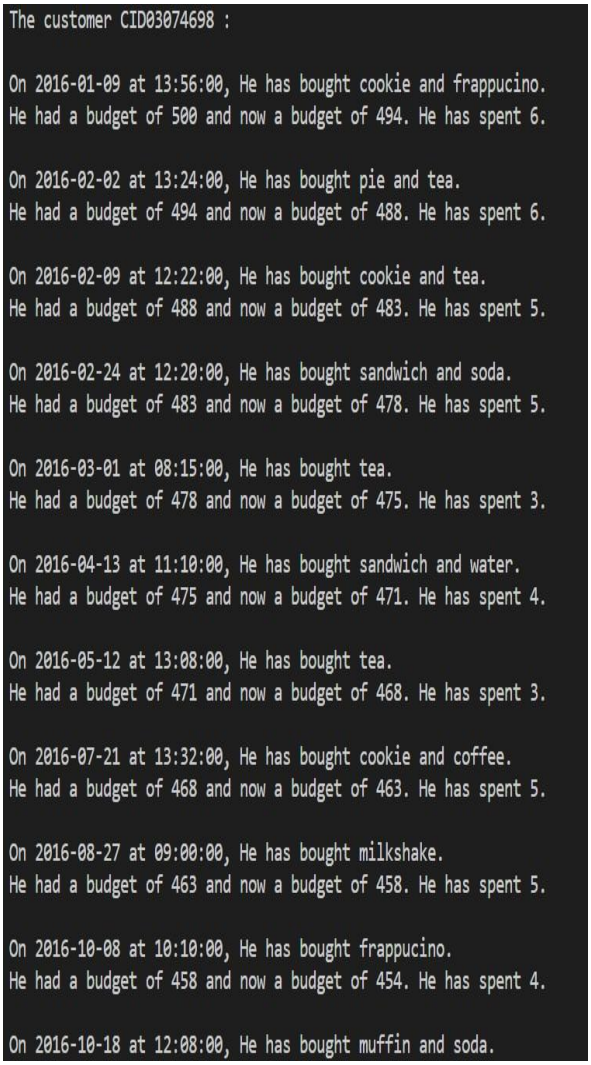


Figure 6 above – Number of returning customers still in budget

Simulation Part 4 – Q1 – Buying History



```
The customer CID03074698 :

On 2016-01-09 at 13:56:00, He has bought cookie and frappucino.
He had a budget of 500 and now a budget of 494. He has spent 6.

On 2016-02-02 at 13:24:00, He has bought pie and tea.
He had a budget of 494 and now a budget of 488. He has spent 6.

On 2016-02-09 at 12:22:00, He has bought cookie and tea.
He had a budget of 488 and now a budget of 483. He has spent 5.

On 2016-02-24 at 12:20:00, He has bought sandwich and soda.
He had a budget of 483 and now a budget of 478. He has spent 5.

On 2016-03-01 at 08:15:00, He has bought tea.
He had a budget of 478 and now a budget of 475. He has spent 3.

On 2016-04-13 at 11:10:00, He has bought sandwich and water.
He had a budget of 475 and now a budget of 471. He has spent 4.

On 2016-05-12 at 13:08:00, He has bought tea.
He had a budget of 471 and now a budget of 468. He has spent 3.

On 2016-07-21 at 13:32:00, He has bought cookie and coffee.
He had a budget of 468 and now a budget of 463. He has spent 5.

On 2016-08-27 at 09:00:00, He has bought milkshake.
He had a budget of 463 and now a budget of 458. He has spent 5.

On 2016-10-08 at 10:10:00, He has bought frappucino.
He had a budget of 458 and now a budget of 454. He has spent 4.

On 2016-10-18 at 12:08:00, He has bought muffin and soda.
```

```
The customer CID67862056 :

On 2016-02-23 at 16:16:00, He has bought milkshake.
He had a budget of 500 and now a budget of 495. He has spent 5.

On 2016-03-29 at 11:40:00, He has bought cookie and coffee.
He had a budget of 495 and now a budget of 490. He has spent 5.

On 2016-04-08 at 15:48:00, He has bought milkshake.
He had a budget of 490 and now a budget of 485. He has spent 5.

On 2016-05-26 at 12:42:00, He has bought muffin and soda.
He had a budget of 485 and now a budget of 479. He has spent 6.

On 2016-06-07 at 11:12:00, He has bought cookie and tea.
He had a budget of 479 and now a budget of 474. He has spent 5.

On 2016-07-03 at 15:52:00, He has bought coffee.
He had a budget of 474 and now a budget of 471. He has spent 3.

On 2016-08-08 at 17:36:00, He has bought soda.
He had a budget of 471 and now a budget of 468. He has spent 3.

On 2016-08-16 at 13:52:00, He has bought soda.
He had a budget of 468 and now a budget of 465. He has spent 3.

On 2016-09-29 at 17:36:00, He has bought pie and frappucino.
He had a budget of 465 and now a budget of 458. He has spent 7.

On 2016-10-17 at 10:35:00, He has bought coffee.
He had a budget of 458 and now a budget of 455. He has spent 3.

On 2016-11-05 at 12:36:00, He has bought sandwich and tea.
He had a budget of 455 and now a budget of 450. He has spent 5.

On 2016-12-11 at 17:12:00, He has bought milkshake.
```

…… ……

Figure 7 Above – Some Purchasing history

Simulation Part 4 – Q2 – Returning customer in the dataset provided

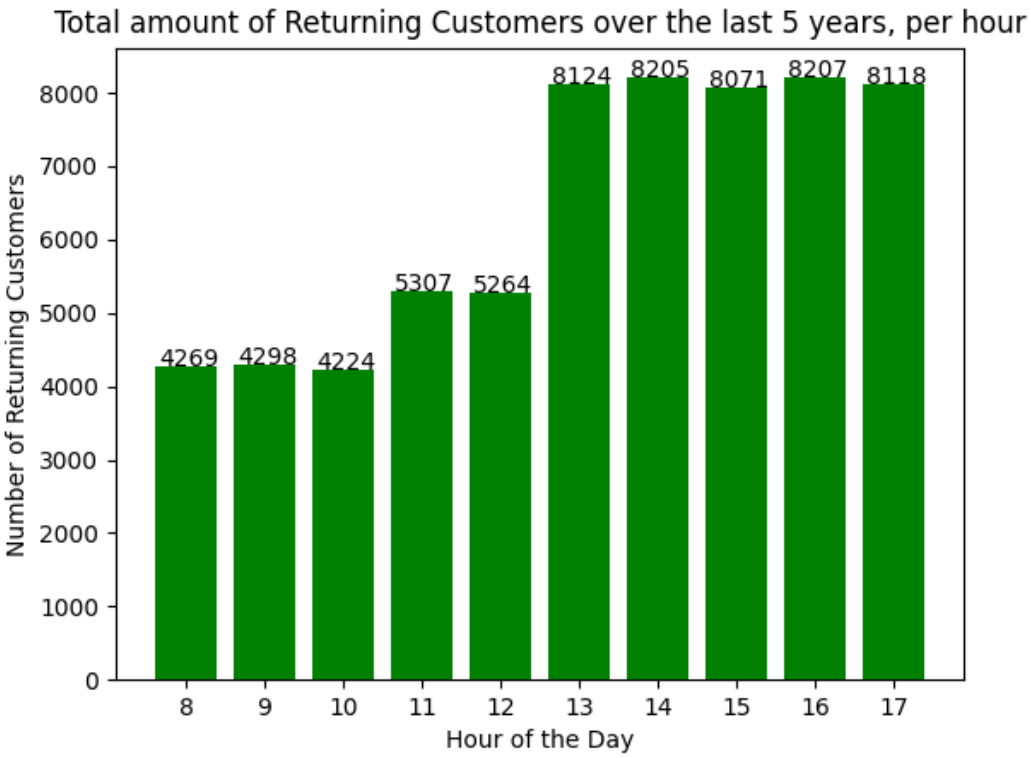


Figure 8 Above – Number of Returning Customer per hour in total

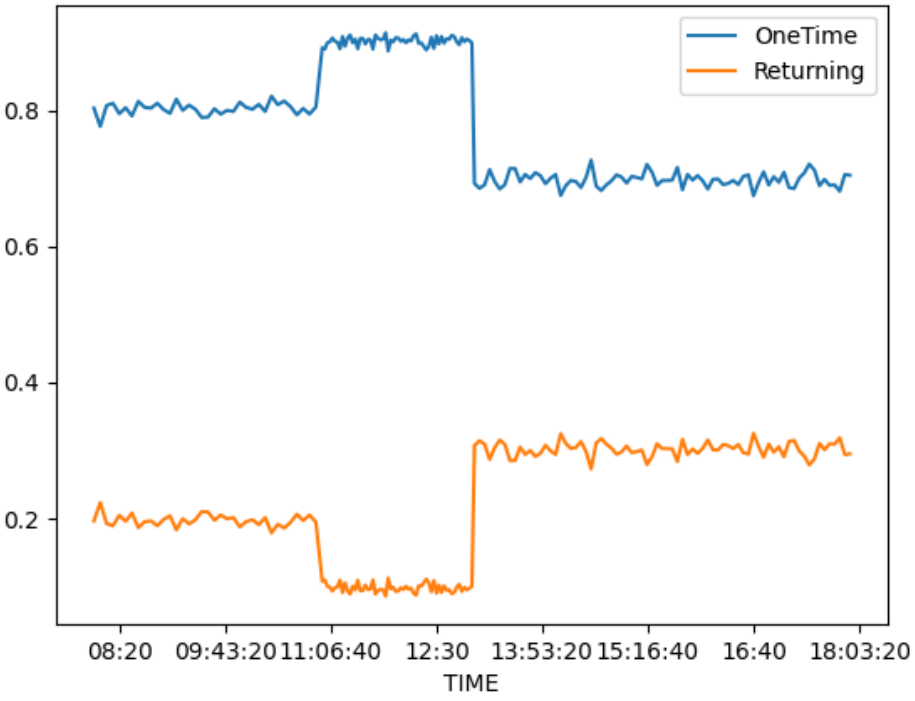


Figure 9 Above – Probability of returning and one time per hour

Simulation Part 4 – Q3 – 50 returning customers

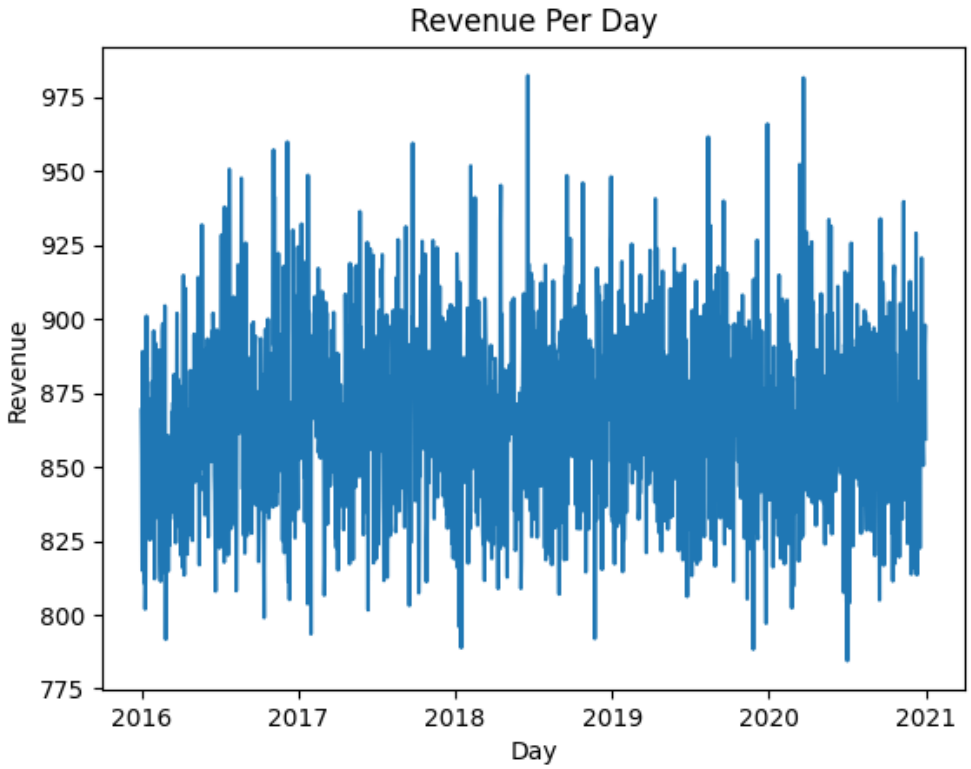
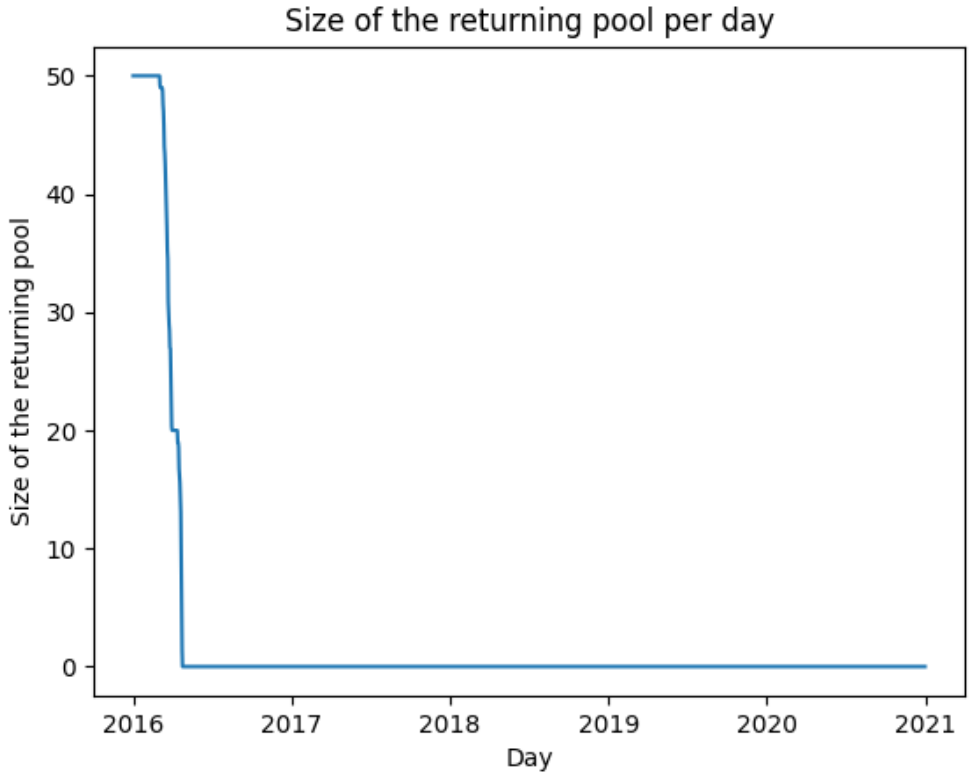


Figure 10 above – Revenue simulation part 4-Q 3



F

Figure 11 above – Number of returning customers still in budget – Part 4 Q3

Simulation Part 4 – Q4 – Price change of 20 %

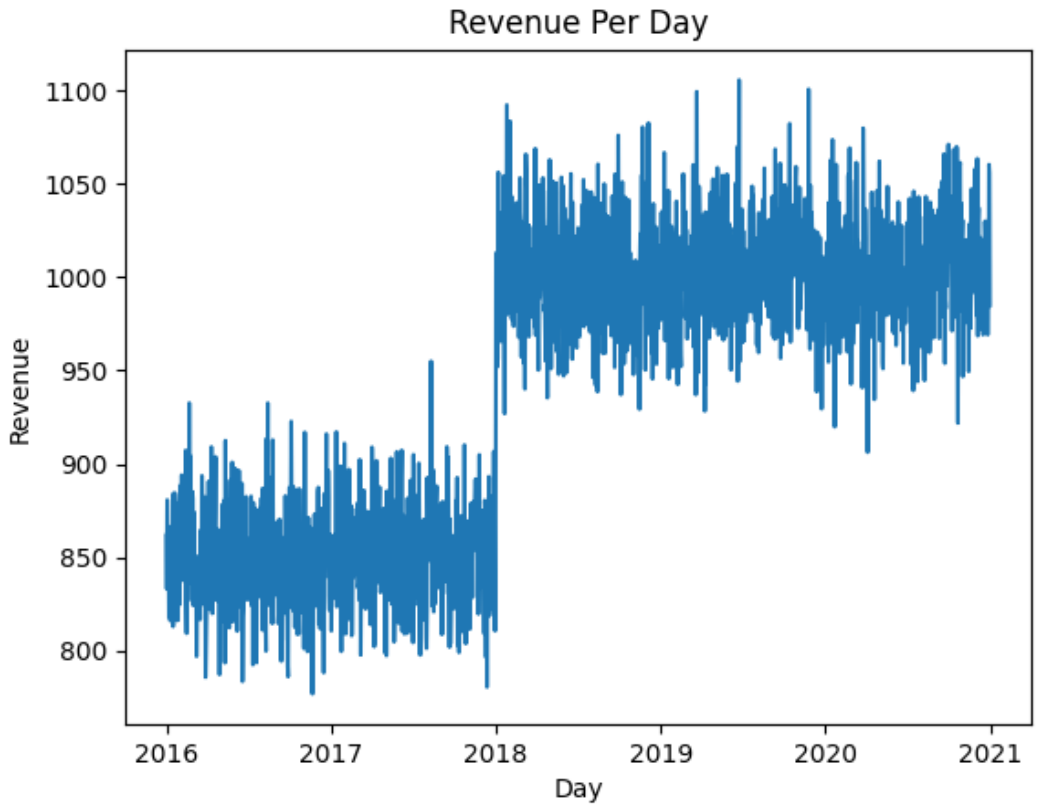


Figure 12 above – Revenue simulation part 4-Q 4

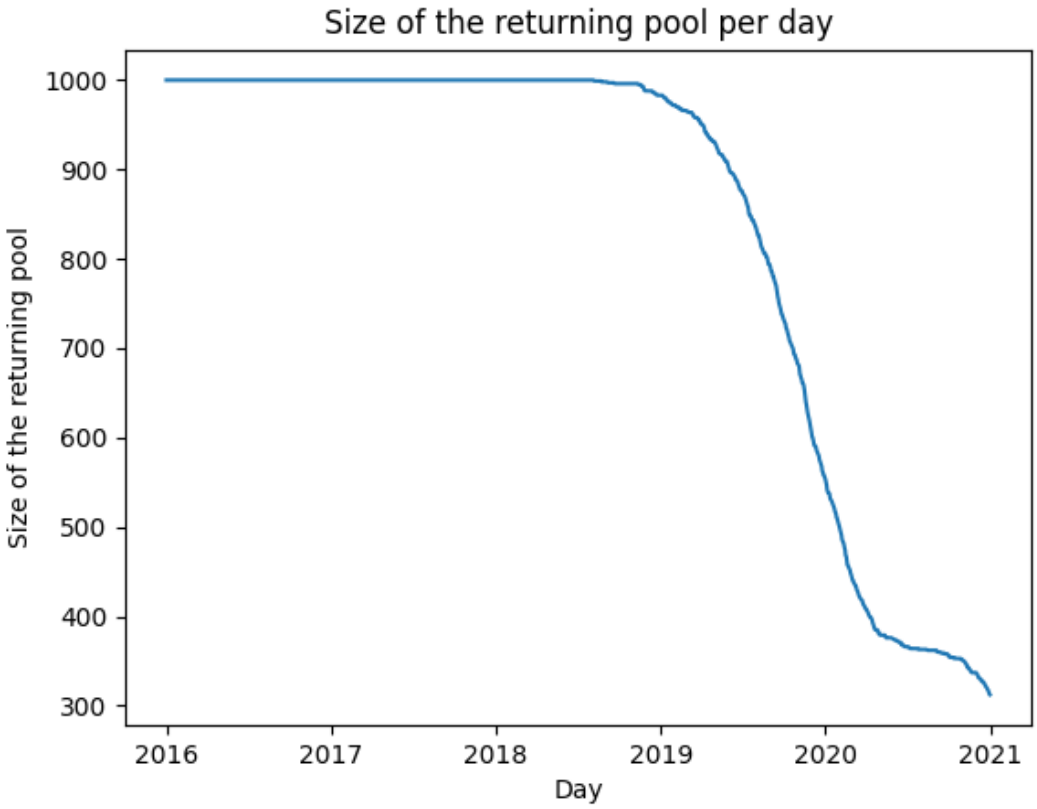


Figure 13 above – Number of returning customers still in budget – Part 4 Q4

Simulation Part 4 – Q5 – Budget of hipster down to 40

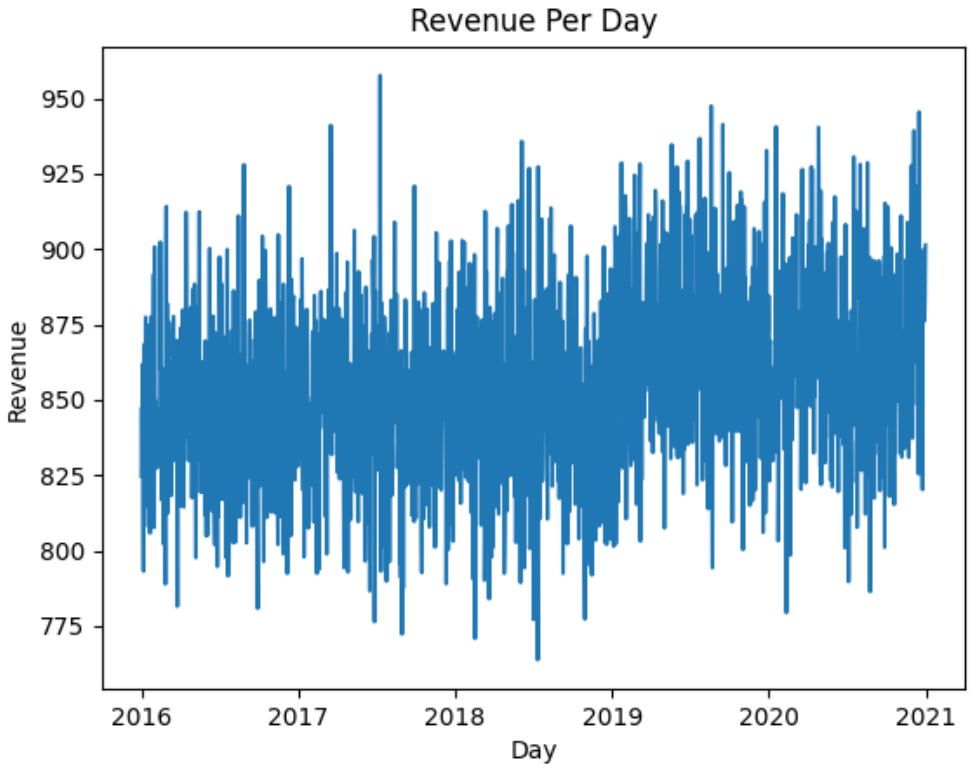


Figure 14 above – Revenue simulation part 4-Q 5

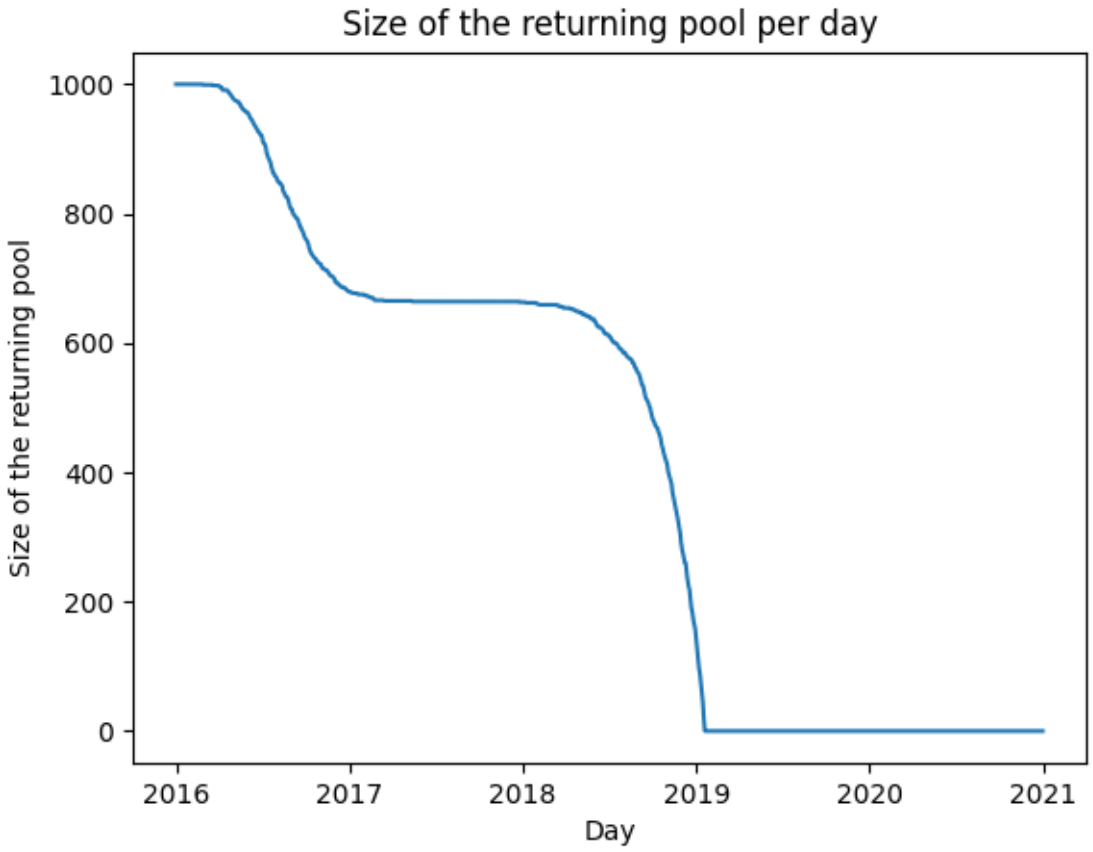


Figure 15 above – Number of returning customers still in budget – Part 4 Q5

Simulation Part 4 – Q6 – Implementation of Uber eat

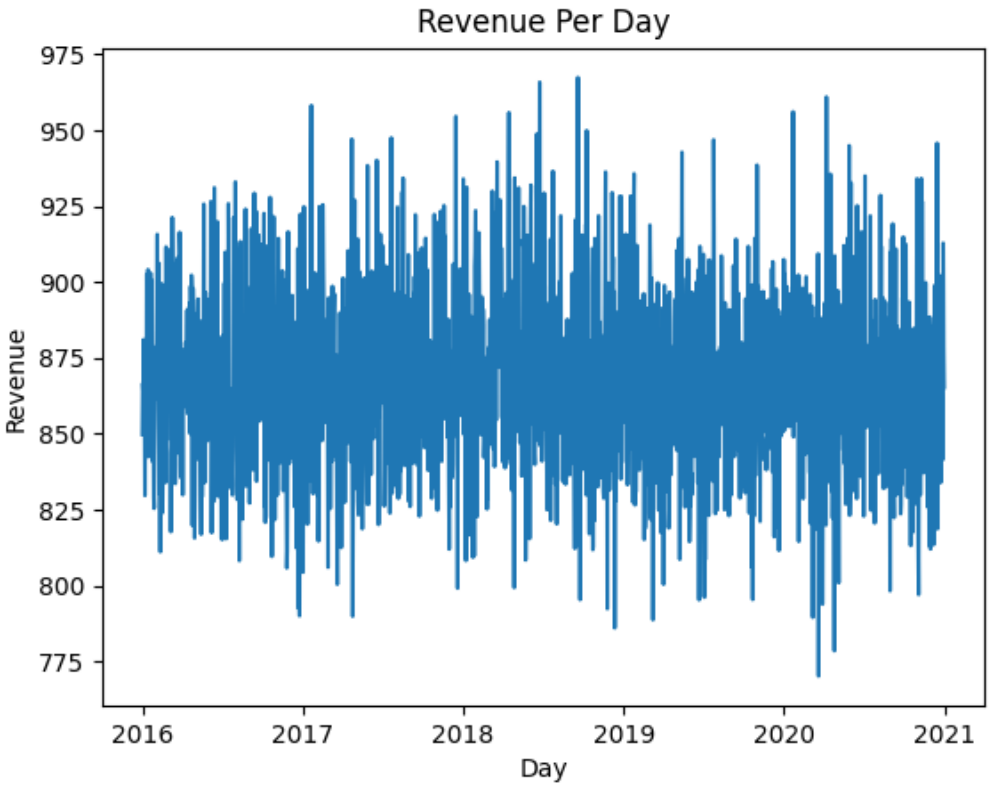


Figure 16 above – Revenue simulation part 4 - Q6

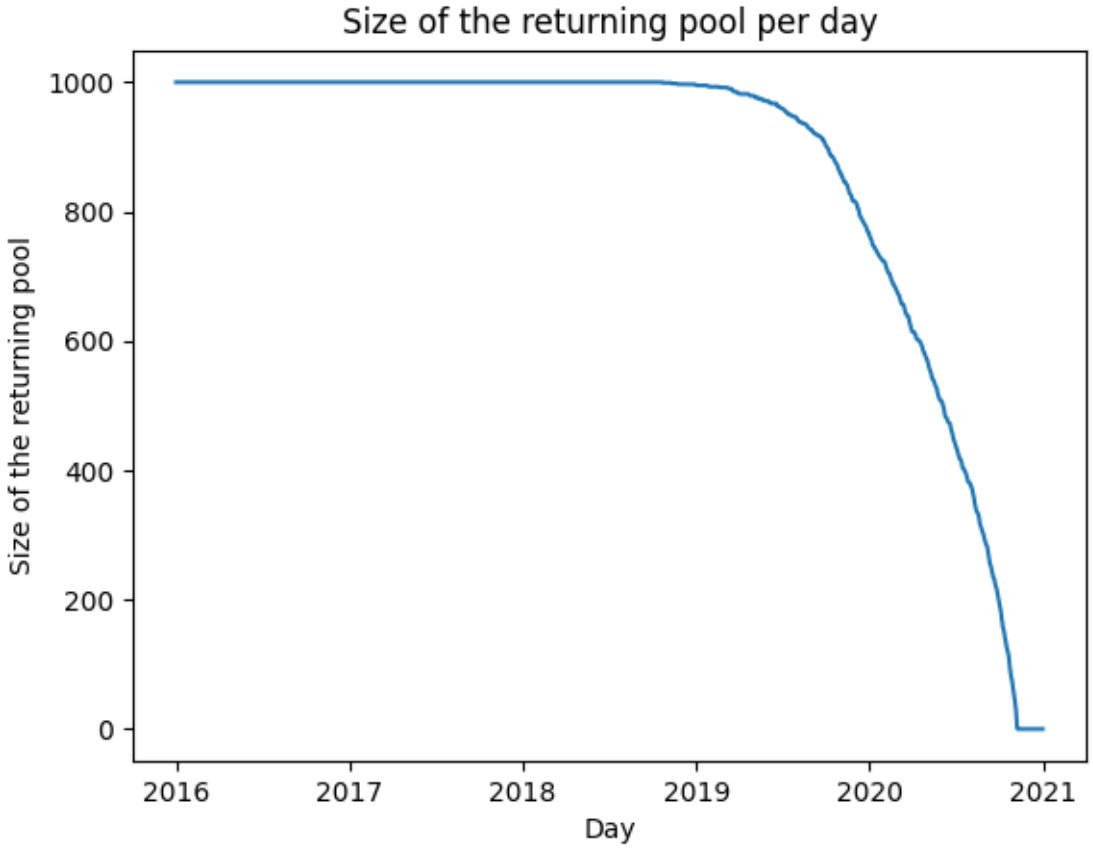


Figure 17 above – Number of returning customers still in budget – Part 4 Q6

Additional Graphs

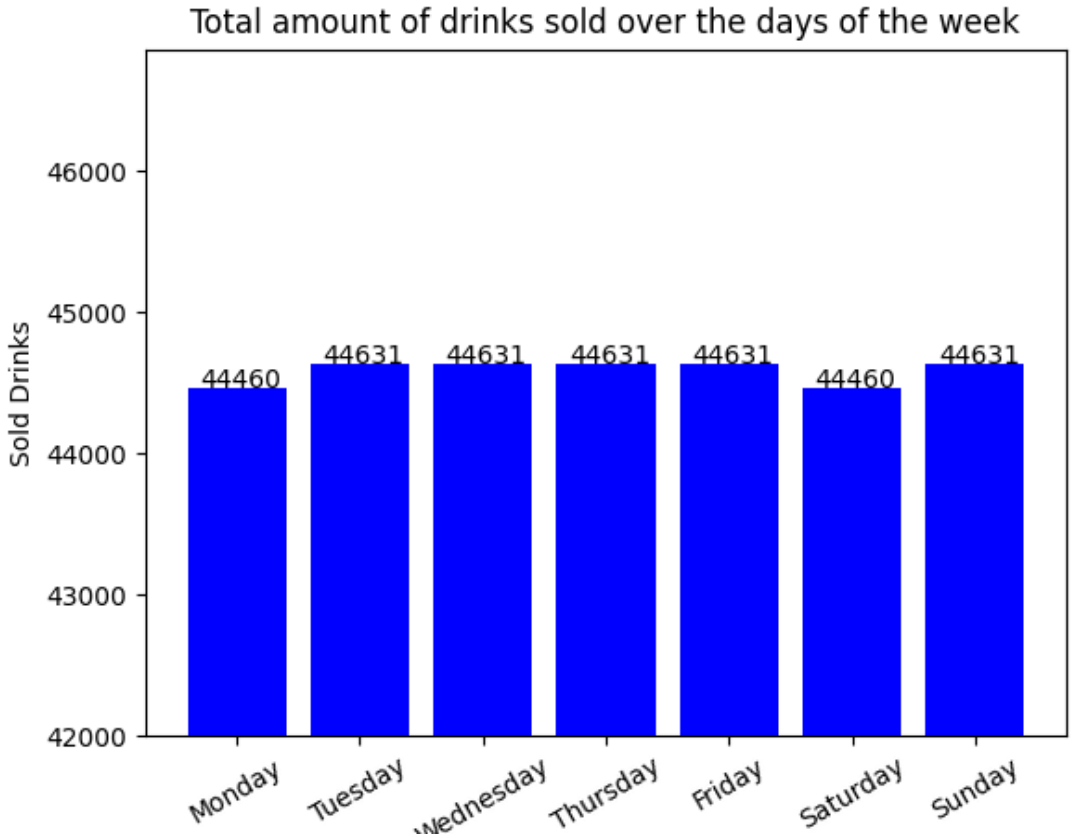


Figure 18 above – Number of drinks sold over the days of the week

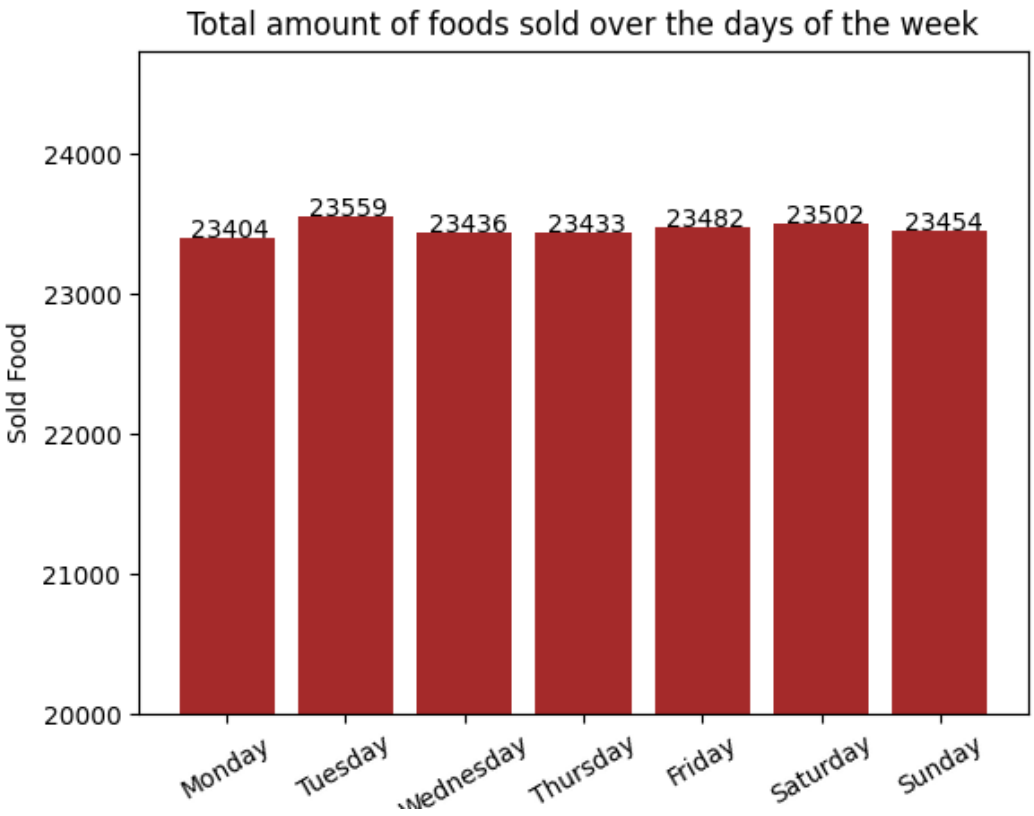


Figure 19 above – Nuimber of food sold over the days of the week

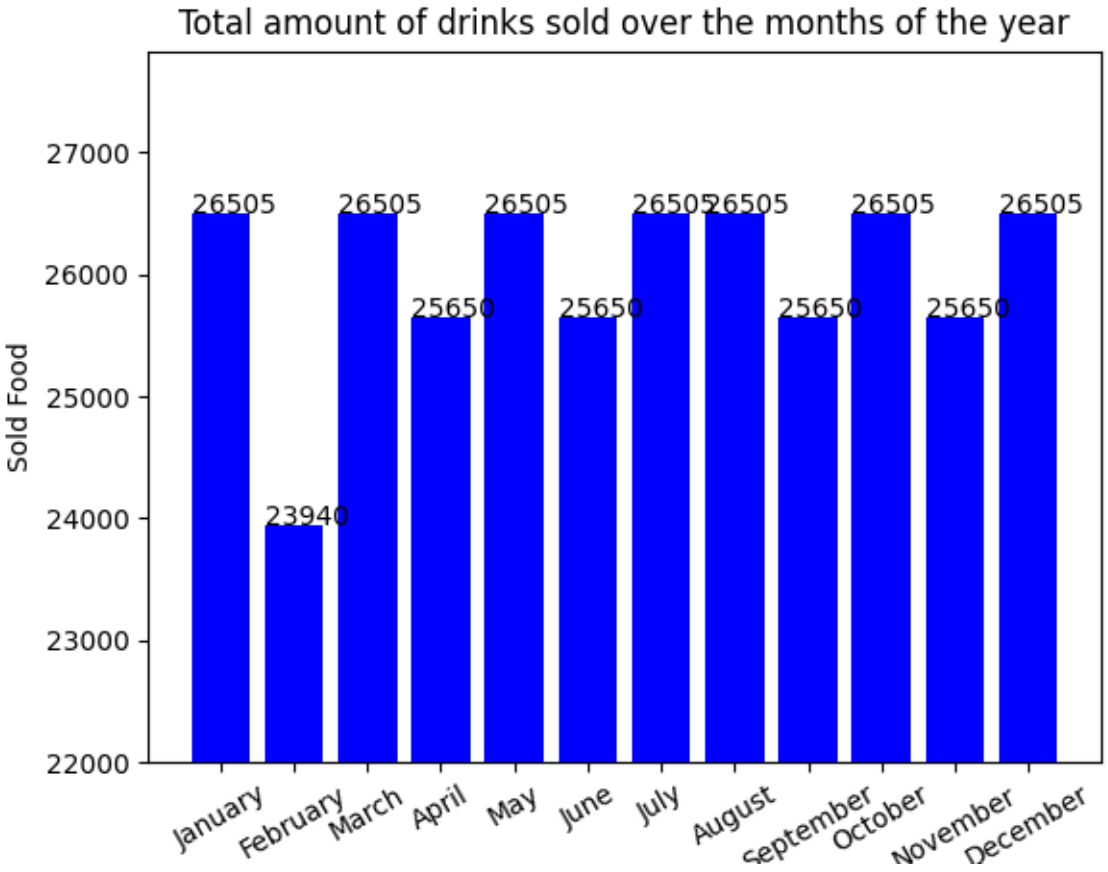


Figure 20 above – Number of drinks sold over the months of the year

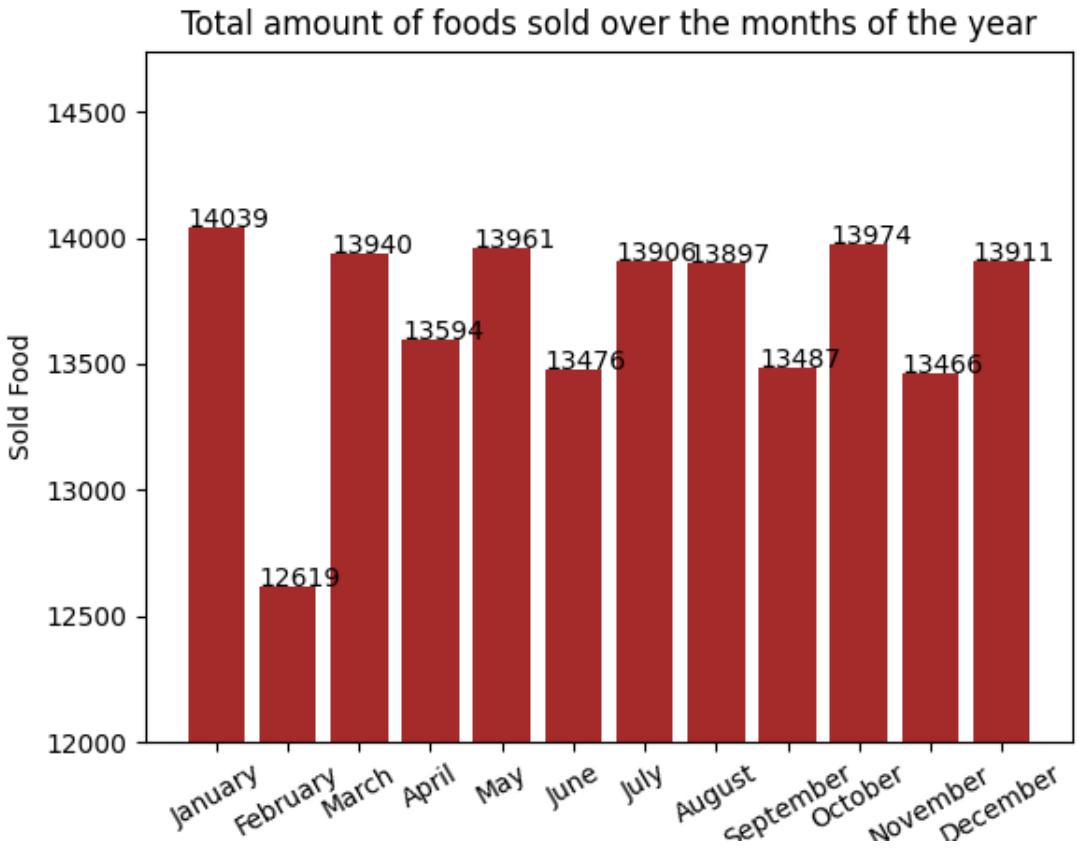


Figure 21 above – Number of food sold over the months of the year

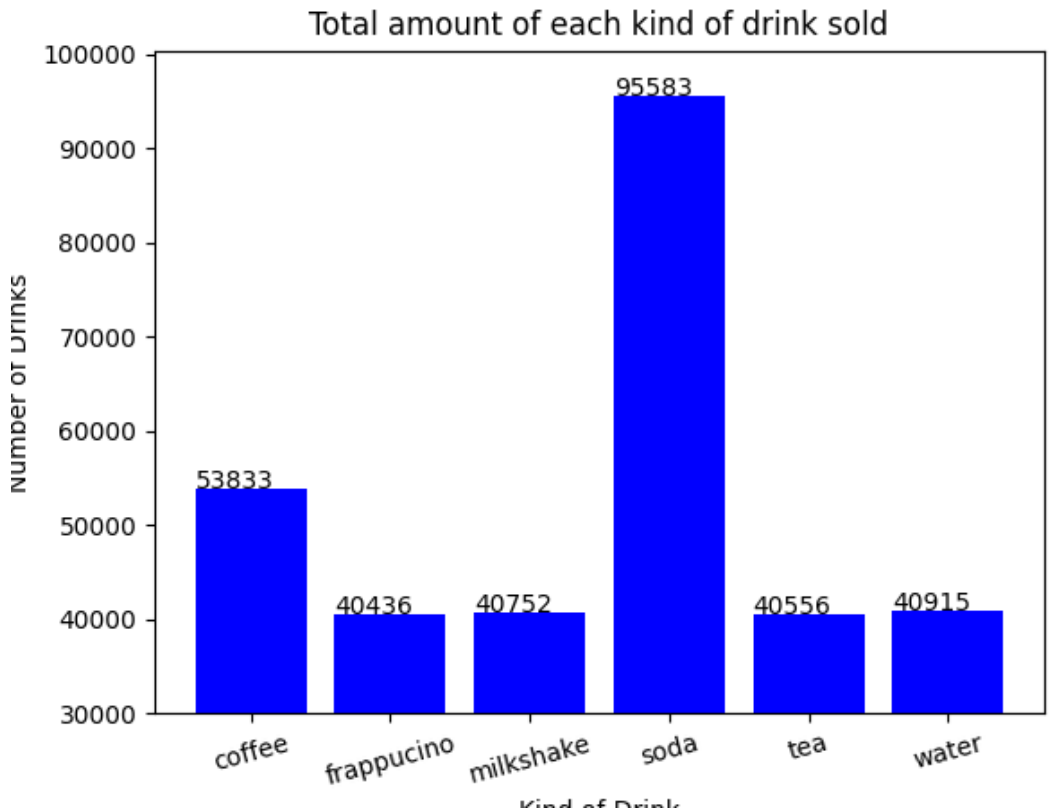


Figure 22 above – Number of items sold for each drink item

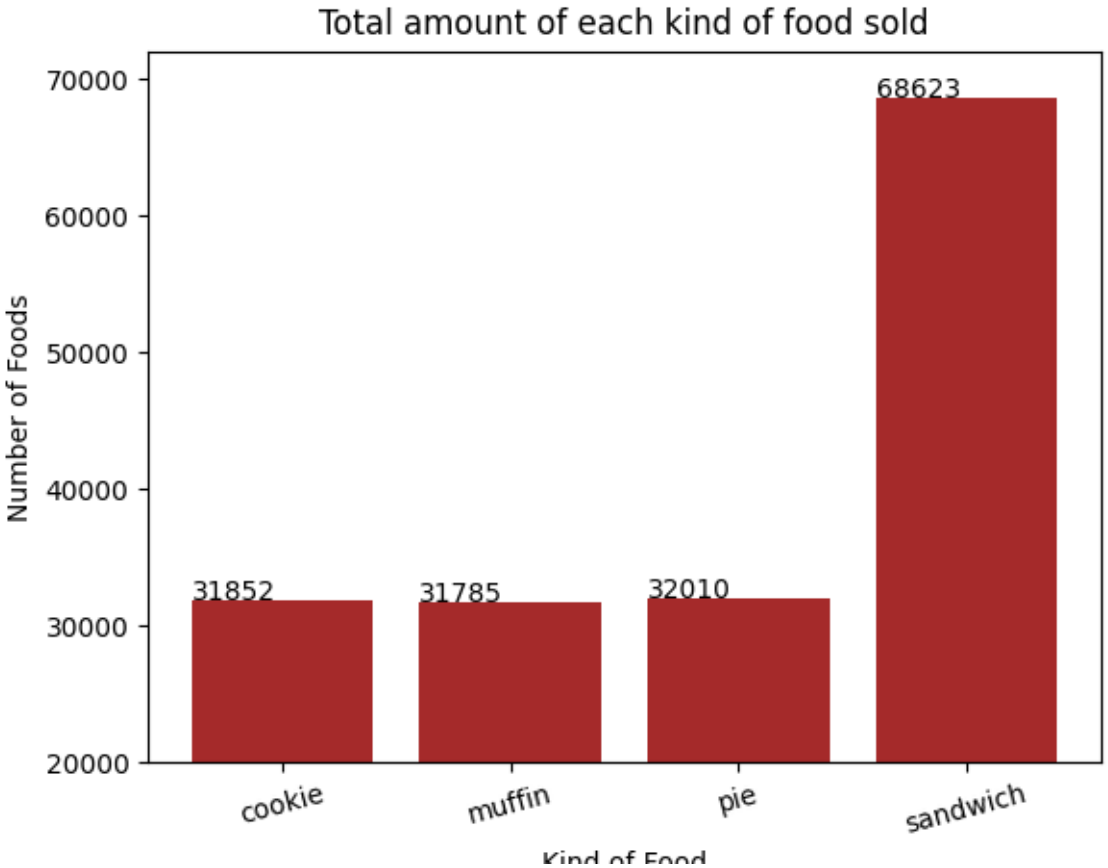


Figure 23 above – Number of items sold for each food item

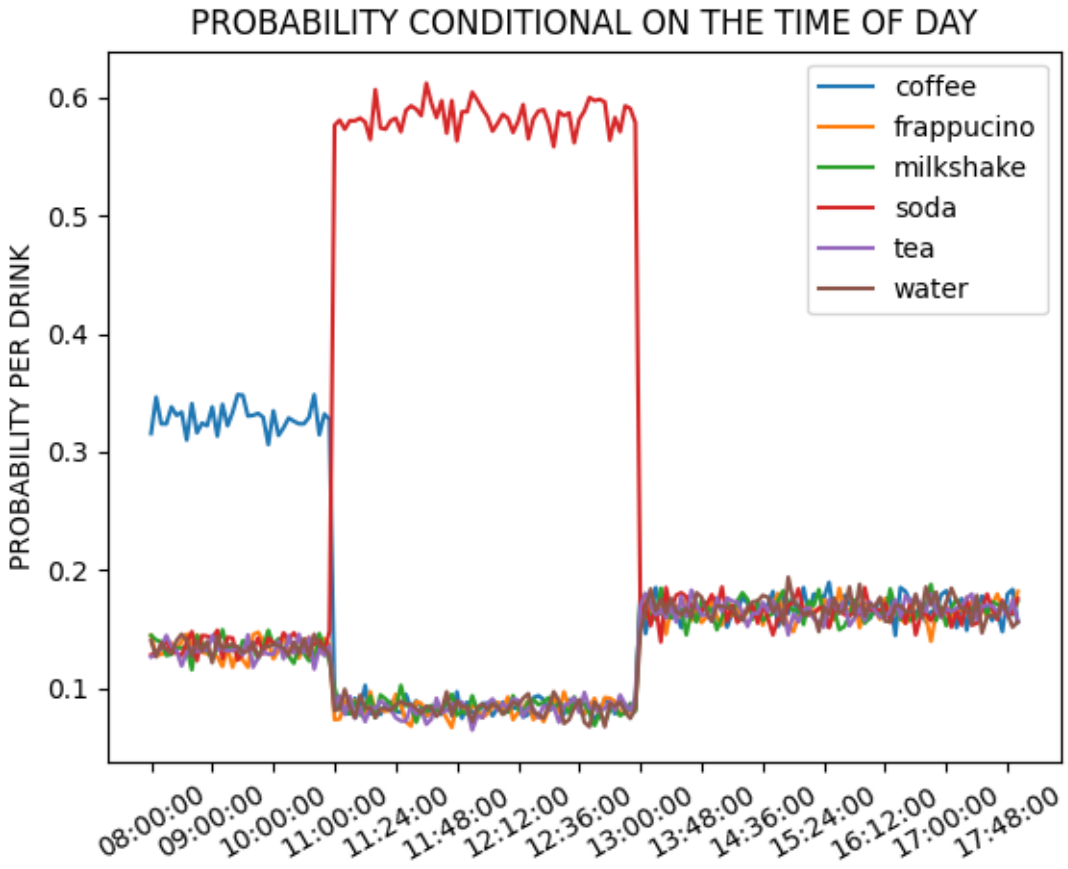


Figure 24 above – Probability of each drink item per timeslot

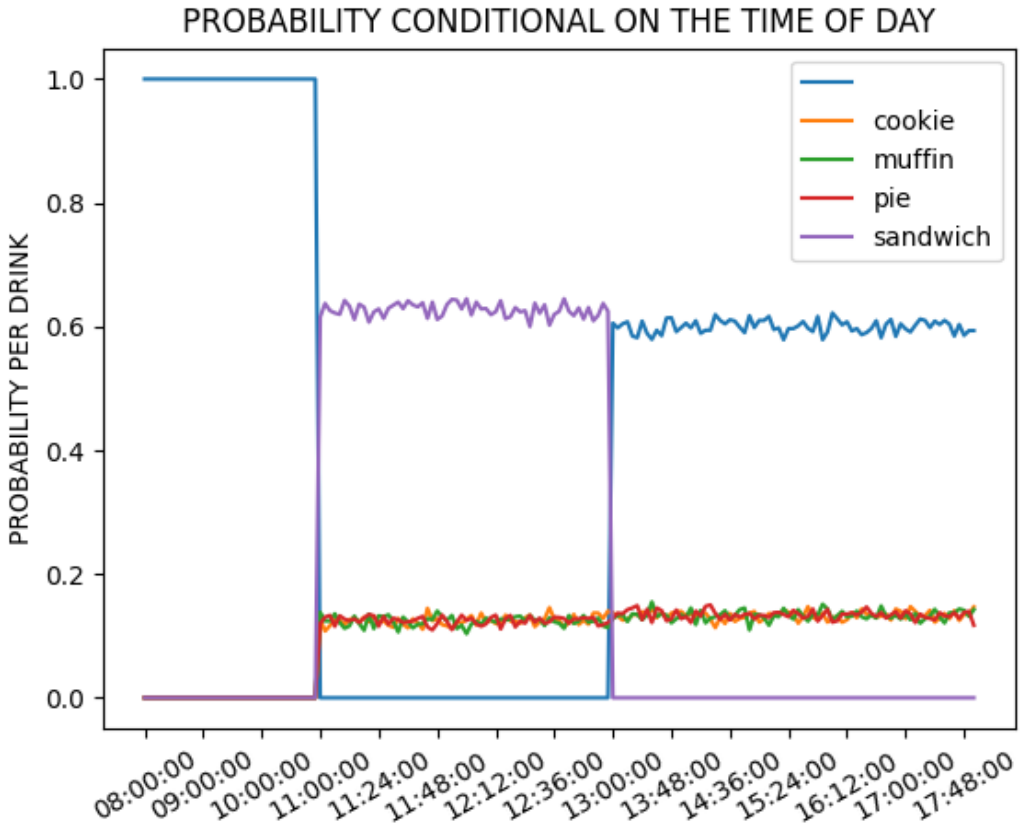


Figure 25 above – Probability of each food item per timeslot

Bibliography:

[1] Course of Dr. De Geyter in Python
[2] https://stackoverflow.com/questions/26763344/convert-pandas-column-to-datetime
[3] https://stackoverflow.com/questions/27241253/print-the-unique-values-in-every-column-in-a-pandas-dataframe
[4] https://stackoverflow.com/questions/39782689/print-a-list-of-items-separated-by-commas-but-with-the-word-and-before-the-las
[5] https://stackoverflow.com/questions/45759966/counting-unique-values-in-a-column-in-pandas-dataframe-like-in-qlik
[6] https://stackoverflow.com/questions/11391969/how-to-group-pandas-dataframe-entries-by-date-in-a-non-unique-column
[7] https://matplotlib.org/cheatsheets/_images/cheatsheets-1.png
[8] https://stackoverflow.com/questions/22642511/change-y-range-to-start-from-0-with-matplotlib
[9] https://stackoverflow.com/questions/53066633/how-to-show-values-on-top-of-bar-plot
[10] https://stackoverflow.com/questions/11373610/save-matplotlib-file-to-a-directory
[11] https://stackoverflow.com/questions/33468976/pandas-conditional-probability-of-a-given-specific-b
[12] https://www.statology.org/pandas-replace-nan-with-string/
[13] https://stackoverflow.com/questions/24640399/how-to-unstack-or-pivot-in-pandas
[14] https://stackoverflow.com/questions/12850345/how-do-i-combine-two-dataframes
[15] https://datatofish.com/export-dataframe-to-csv/
[16] https://docs.python.org/3/tutorial/classes.html
[17] https://pynative.com/python-get-random-float-numbers/
[18] https://stackoverflow.com/questions/2757116/list-in-a-python-class-shares-the-same-object-over-2-different-instances
[19] https://stackoverflow.com/questions/42044090/return-the-maximum-value-from-a-dictionary
[20] https://www.pythontutorial.net/python-oop/python-__bool__/
[21] https://stackoverflow.com/questions/74090268/def-bool-oriented-object-programming-python#comment130813792_74090268
[22] https://www.geeksforgeeks.org/choose-elements-from-list-with-different-probability-in-python/amp/?fbclid=IwAR1XMOOrg-T5uax7L99L4lQYlGFkSC9uw_VAWCKqQD3x0H1QDNH7RujVDgk
[23] https://stackoverflow.com/questions/53857813/use-of-index-0-in-random-choices
[24] https://stackoverflow.com/questions/9578906/easiest-way-to-combine-date-and-time-strings-to-single-datetime-object-using-pyt
[25] https://stackoverflow.com/questions/19480028/attributeerror-datetime-module-has-no-attribute-strptime
[26] https://stackoverflow.com/questions/3996904/generate-random-integers-between-0-and-9*
[27] https://stackoverflow.com/questions/55096390/generate-a-id-check-if-it-exists-if-yes-loop-until-new-id-is-generated
[28] https://stackoverflow.com/questions/7571635/fastest-way-to-check-if-a-value-exists-in-a-list
[29] https://stackoverflow.com/questions/74178813/probability-in-if-statement*
[30] https://flexiple.com/python/check-if-list-is-empty-python/
[31] https://stackoverflow.com/questions/60420722/remove-an-object-from-a-list-of-objects-by-attribute
[32] https://towardsdatascience.com/heres-the-most-efficient-way-to-iterate-through-your-pandas-dataframe-4dad88ac92ee
[33] https://stackoverflow.com/questions/1060279/iterating-through-a-range-of-dates-in-python
[34] https://www.adamsmith.haus/python/answers/how-to-iterate-through-a-range-of-dates-in-python
[35]https://www.google.com/search?q=transform+a+dictionnary+to+a+dataframe&rlz=1C1VDKB_frFR945FR945&oq=transform+a+dictionnary+to+a+dataframe&aqs=chrome..69i57j0i22i30l9.7007j0j4&sourceid=chrome&ie=UTF-8

[36] https://www.statology.org/pandas-find-duplicates/
[37] https://thinkingneuron.com/how-to-measure-the-correlation-between-two-categorical-variables-in-python/
[38] https://bobbyhadz.com/blog/python-multiply-dictionary-values
[39] https://stackoverflow.com/questions/9371114/check-if-list-of-objects-contain-an-object-with-a-certain-attribute-value
[40] https://pynative.com/python-isinstance-explained-with-examples/#:~:text=Using%20isinstance()%20function%2C%20we,parent%20class%20of%20an%20object.&text=For%20example%2C%20isinstance(x%2C,instance%20of%20a%20class%20int%20
[41] https://careerkarma.com/blog/python-optional-arguments/#:~:text=A%20Python%20optional%20argument%20is%20an%20argument%20with%20a%20default,if%20one%20is%20not%20specified.