The bakery problem

(Probability course project)

Authors:
Ali Tavakoli & Maryam Hosseinali

The Bakery Problem

At first we will state the problem in detail.

- ➔ We just moved to a new neighborhood.
- ➔ This neighborhood has five bakeries.
- ➔Each of them gives us good bread witha certain probability.
- ➔There is no preference in the type of bread of the bakeries.
- ➔ We want to buy bread many times and don't know which one is more likely to give us good bread.
- ➔ The goal is to rank bakeries based on good experience so that we have the highest probability of having a good experience every day.

For that at first we reduce the problem: "How to find the probability of a bakery giving good bread by its previous breads ?". Simply put, we want to find the probability of probabilities. And we use **Beta** distribution to model this problem.

Why Beta distribution?

The beta distribution has several features that make it well-suited for modeling uncertainty in reward probabilities (a situation similar to our bakery problem):

Support for a bounded range: The beta distribution is defined on the interval [0, 1], which makes it a natural choice for modeling probabilities and in the bakery problem the range of the distribution needs to match the range of the probabilities. The beta distribution's range aligns well with this requirement.

Conjugacy to the binomial distribution: If the prior distribution is a beta distribution and the observed rewards follow a binomial distribution, the resulting posterior distribution will also be a beta distribution. This property allows for efficient and tractable updates of the belief distribution as new rewards are observed.

Flexibility in shape: The beta distribution allows for a wide range of shapes, from uniform distributions to highly skewed distributions. By adjusting the parameters of the distribution, the designer can capture different levels of uncertainty and skewness in the bakery rankings.

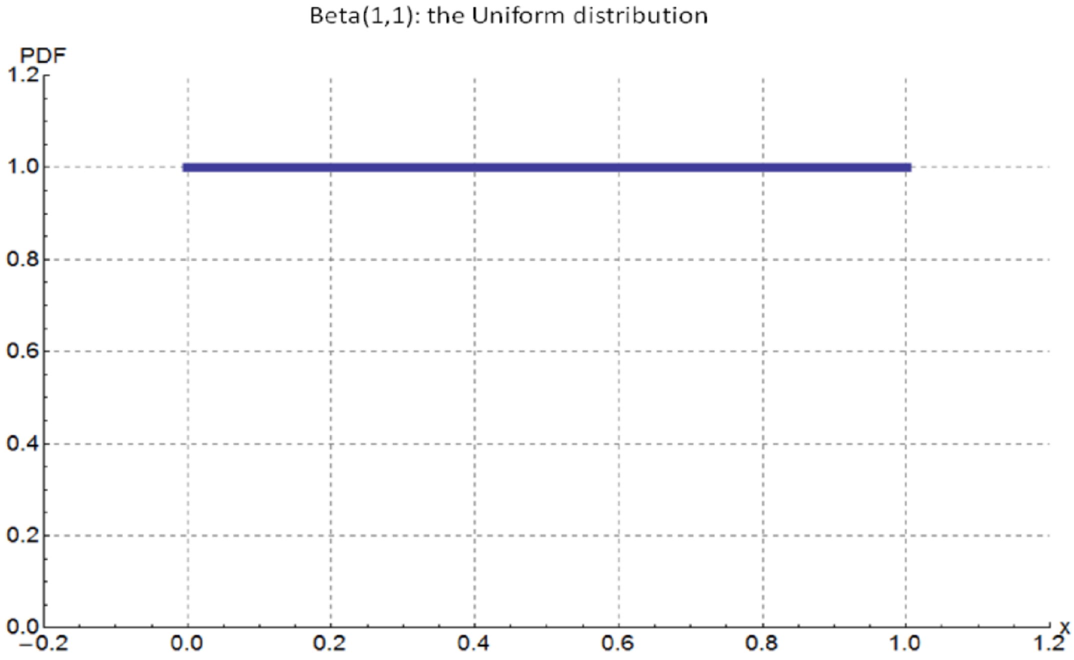
Parameter interpretation: The parameters of the beta distribution have a straightforward interpretation. The shape parameters, $N_i(1)+1$ and $N_i(0)+1$, represent the number of observed successes and failures, respectively, which aligns with the number of good and bad experiences for each bakery. By incorporating the observed experiences into the distribution's parameters, the designer can update their beliefs as more data becomes available.

Setting the parameters:

Why p_i~ Beta(N_i(1)+1,N_i(0)+1)?

The Beta distribution takes two parameters, 'α' (alpha) and 'β' (beta). In our problem these parameters can be respectively interpreted as the count of successes and failures in the purchases.

Initially the probability of a successful purchase from any of the bakeries is not known , so we start by setting both 'α' and 'β' to one, which produces a flat line Uniform distribution:



This initial guess is the probability of the specific event occurring before we have collected any evidence and represented by the Beta distribution *Beta(1,1)*.

If the experience of buying from a bakery is successful , the reward will be 1 and ‘α’, the count of the number of successes, will increase by 1. The count of the number of failures, ‘β’, will not increase. If instead no reward was obtained, then ‘α’ will stay the same and ‘β’ will increment by
1.

As more data is collected the Beta distribution moves from being a flat line to become an increasingly accurate model of the probability of the mean reward.

By setting the parameters as $N_i(1)+1$ and $N_i(0)+1$, we ensure that the beta distribution captures the prior beliefs while remaining flexible to update with new experiences.

Therefore, the beta distribution in our problem can be represented as ***Beta(*N_i*(1)+1,*N_i*(0)+1)***.

Choose the bakery for the next day:

➔ The goal is to select the bakery that is most likely to provide good bread for the next day.

➔To solve this problem, we use a powerful approach called Thompson sampling which is a statistical algorithm that balances exploration and exploitation to make smart decisions.

➔The algorithm maintains a probability distribution (beta distribution) for each bakery, representing the uncertainty about the true success probability (probability of good bread) of that bakery.

➔At the start, the algorithm assumes a uniform prior distribution for each bakery(*Beta(1,1)*), which means that all success probabilities are equally likely.

➔As the algorithm collects data through observations of good or bad bread, it updates the probability distributions for each bakery based on the observed data.

➔To choose the bakery for the next day, the algorithm samples from the posterior distributions of each bakery and selects the bakery with the highest sampled value.

➔This sampling process considers the uncertainty in the success probabilities and allows exploration of different options.

➔Over time, as more data is collected, the algorithm converges towards the true success probabilities of each bakery. This enables it to make best decisions on which bakery to choose.

➔Thompson sampling strikes a balance between exploring new options and exploiting the bakeries with higher estimated quality. It allows us to gradually learn and adapt our choices based on the observed outcomes.Therefore, it is an optimal algorithm that we simulate as the second and main approach in the following.

Approaches to select the bakery

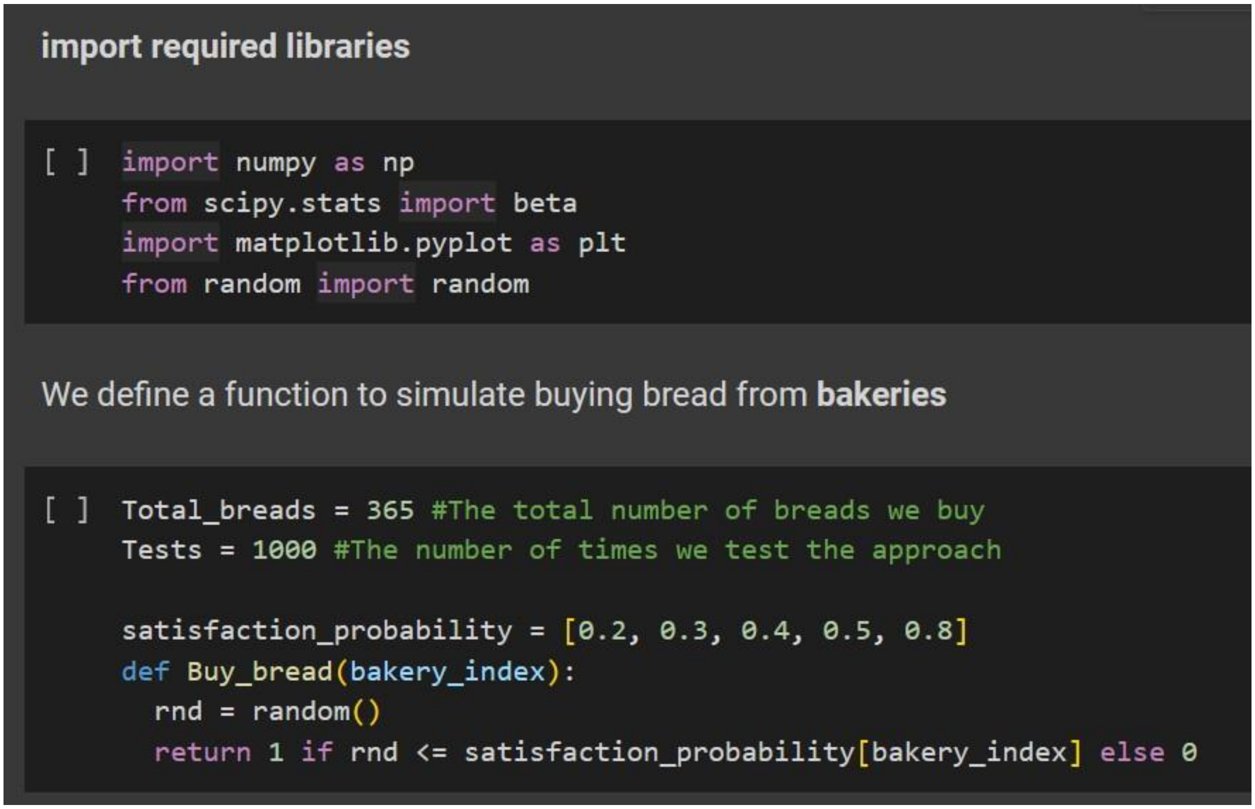
We try three different approaches to choose the bakery and then we compare the results.

1) **N-Test Selection Strategy:** first, buy **N** breads from each bakery and find which of them gives us more good bread and then choose that from now to end as the best bakery. We can test for several different **N** and find the best result.
2) **Beta Distribution-Based Selection Strategy:** for each selection, generate a value according to the Beta distribution of each bakery (with the parameters defined before) and then choose the bakery with larger value.
3) **Greedy Exploration Strategy:** this approach is an exploration-exploitation strategy for solving the bakery

problem. It uses a greedy approach with a certain exploration rate to balance between choosing the bakery with the highest rewards and exploring other options(in order not to get stuck in the local optimum). The chosen bakery indices and rewards are recorded over a given number of days. The average percent of satisfaction is calculated based on the rewards obtained. The approach aims to maximize the overall satisfaction by adaptively selecting the best bakery choices.

Description of our simulation

We used *python* to simulate our approaches to choose the bakery.



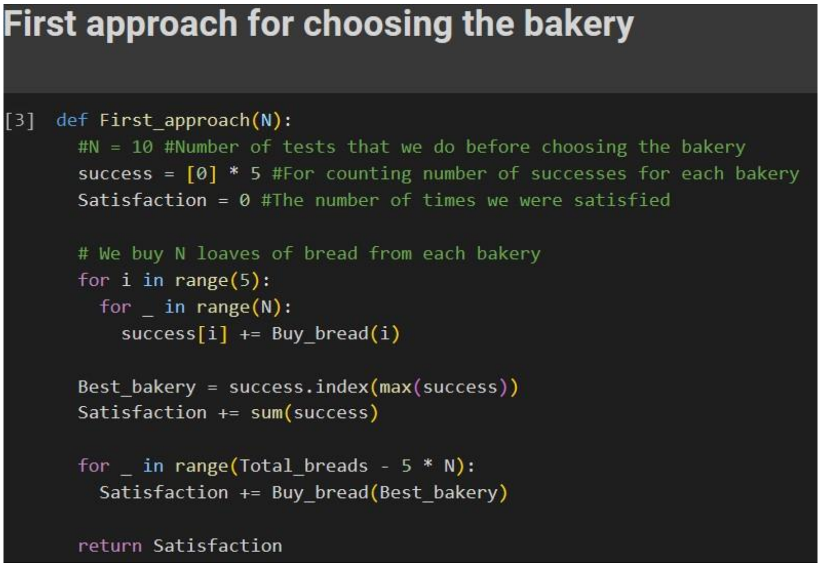
import required libraries

```
[ ]  import numpy as np
     from scipy.stats import beta
     import matplotlib.pyplot as plt
     from random import random
```

We define a function to simulate buying bread from **bakeries**

```
[ ]  Total_breads = 365 #The total number of breads we buy
     Tests = 1000 #The number of times we test the approach

     satisfaction_probability = [0.2, 0.3, 0.4, 0.5, 0.8]
     def Buy_bread(bakery_index):
       rnd = random()
       return 1 if rnd <= satisfaction_probability[bakery_index] else 0
```

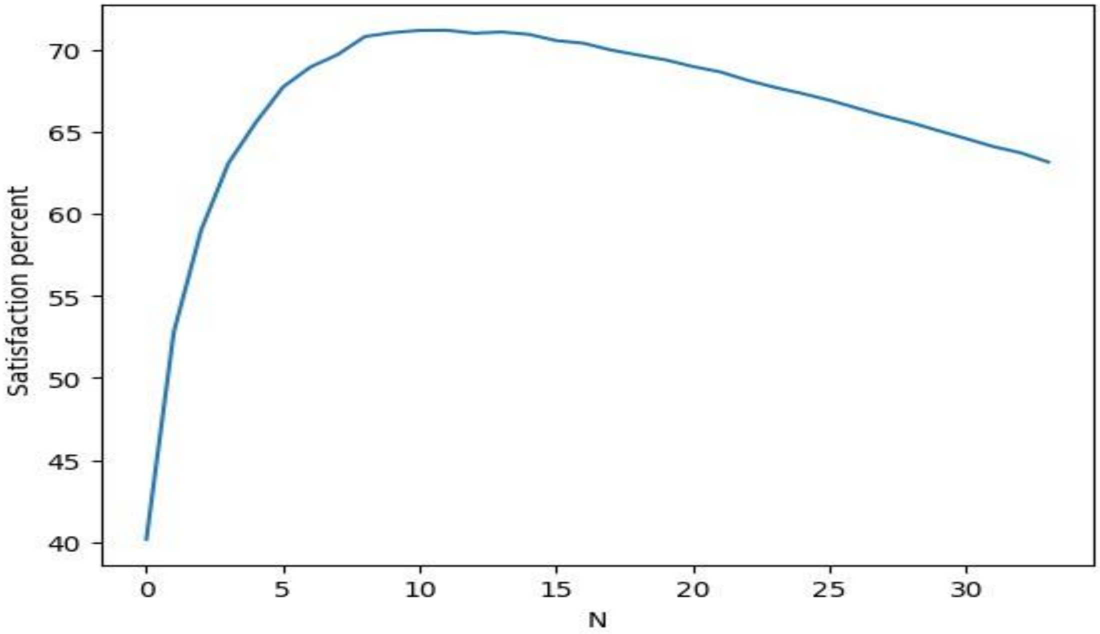
First approach



First approach for choosing the bakery

```
[3] def First_approach(N):
      #N = 10 #Number of tests that we do before choosing the bakery
      success = [0] * 5 #For counting number of successes for each bakery
      Satisfaction = 0 #The number of times we were satisfied

      # We buy N loaves of bread from each bakery
      for i in range(5):
        for _ in range(N):
          success[i] += Buy_bread(i)

      Best_bakery = success.index(max(success))
      Satisfaction += sum(success)

      for _ in range(Total_breads - 5 * N):
        Satisfaction += Buy_bread(Best_bakery)

      return Satisfaction
```

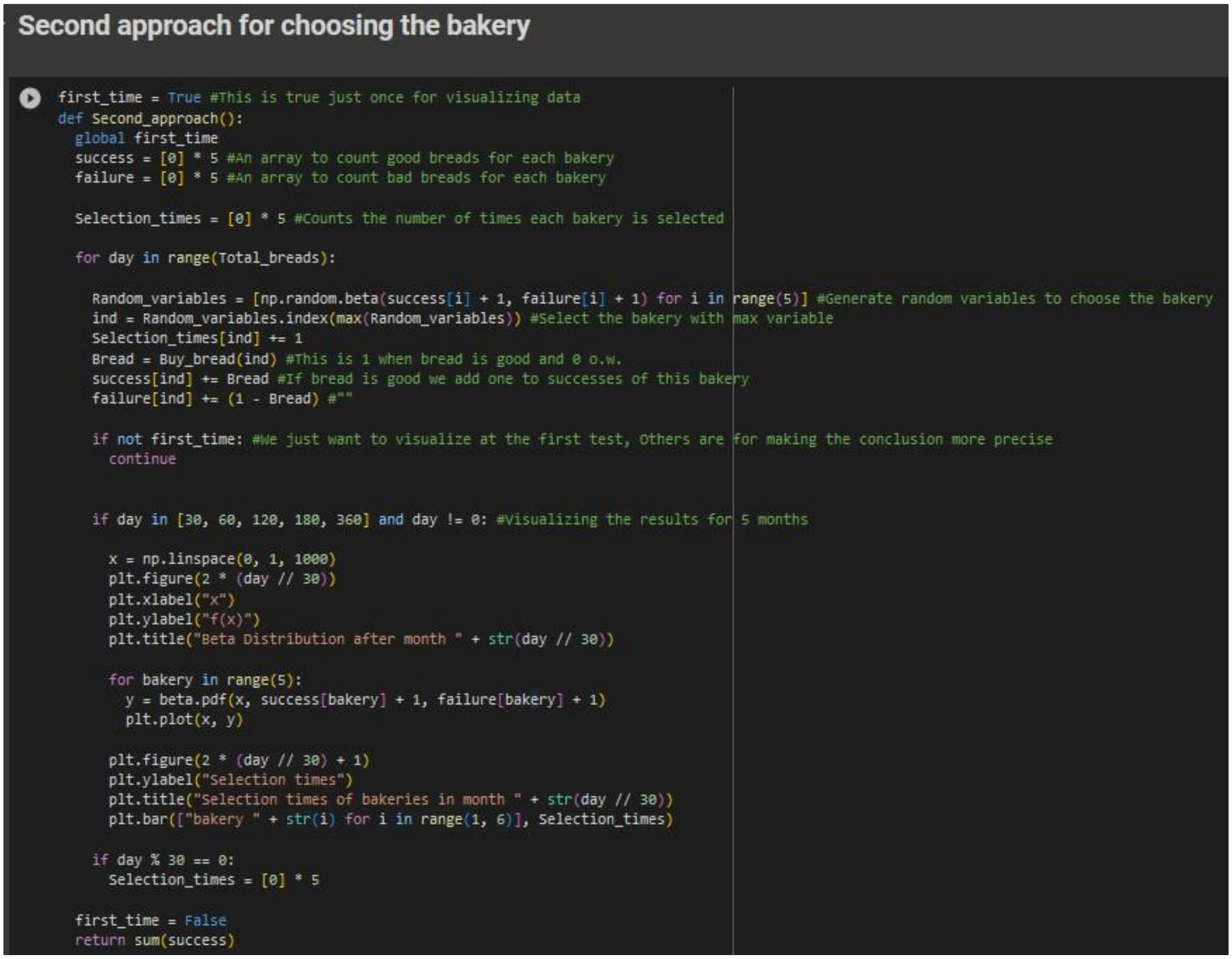
We tested this approach for **N** from 1 to 25 , 10000 times for 1 year and got these results:



This means the best results happen when we set **N** to 12 or a number about that and with **N = 12** we get an average **71.3%**

good bread then we have to buy 12 breads from each bakery at first and then choose the best one for the rest of the year.

Second approach (The main)



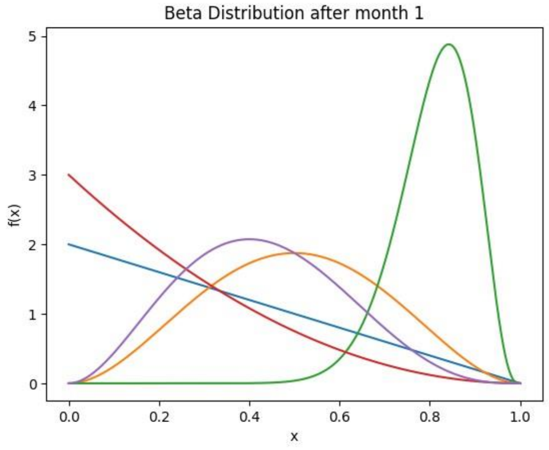
Second approach for choosing the bakery

```python
first_time = True #This is true just once for visualizing data
def Second_approach():
  global first_time
  success = [0] * 5 #An array to count good breads for each bakery
  failure = [0] * 5 #An array to count bad breads for each bakery

  Selection_times = [0] * 5 #Counts the number of times each bakery is selected

  for day in range(Total_breads):

    Random_variables = [np.random.beta(success[i] + 1, failure[i] + 1) for i in range(5)] #Generate random variables to choose the bakery
    ind = Random_variables.index(max(Random_variables)) #Select the bakery with max variable
    Selection_times[ind] += 1
    Bread = Buy_bread(ind) #This is 1 when bread is good and 0 o.w.
    success[ind] += Bread #If bread is good we add one to successes of this bakery
    failure[ind] += (1 - Bread) #""

    if not first_time: #We just want to visualize at the first test, Others are for making the conclusion more precise
      continue


    if day in [30, 60, 120, 180, 360] and day != 0: #Visualizing the results for 5 months

      x = np.linspace(0, 1, 1000)
      plt.figure(2 * (day // 30))
      plt.xlabel("x")
      plt.ylabel("f(x)")
      plt.title("Beta Distribution after month " + str(day // 30))

      for bakery in range(5):
        y = beta.pdf(x, success[bakery] + 1, failure[bakery] + 1)
        plt.plot(x, y)

      plt.figure(2 * (day // 30) + 1)
      plt.ylabel("Selection times")
      plt.title("Selection times of bakeries in month " + str(day // 30))
      plt.bar(["bakery " + str(i) for i in range(1, 6)], Selection_times)

    if day % 30 == 0:
      Selection_times = [0] * 5

  first_time = False
  return sum(success)
```

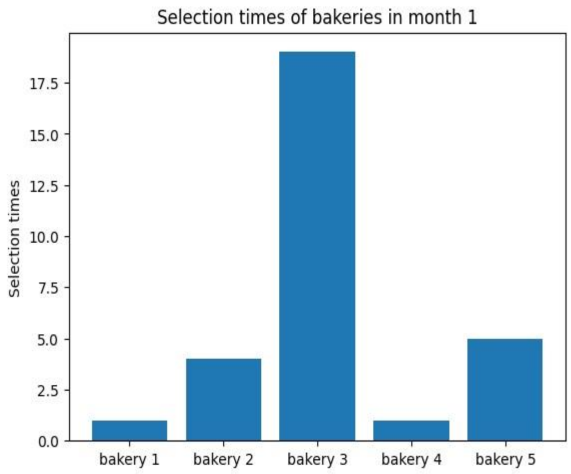
We tested this 10000 times for 1 year and got an average **76.1%** good bread by using the **Beta** distribution of each bakery.

The distribution and the method of selection changes during the year as follows:

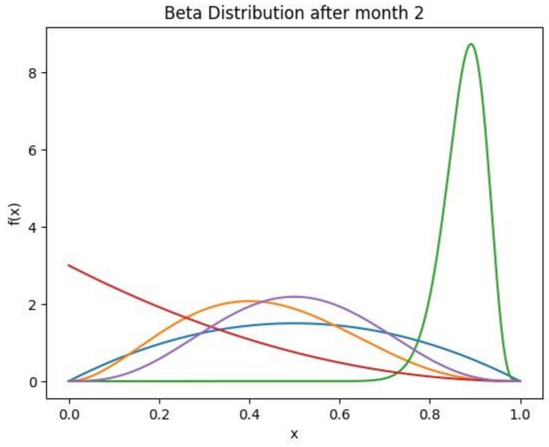
Colors in the graph: (Bakery 1 - Bekery 2 - Bekery 3 - Bekery 4 - Bekery 5)



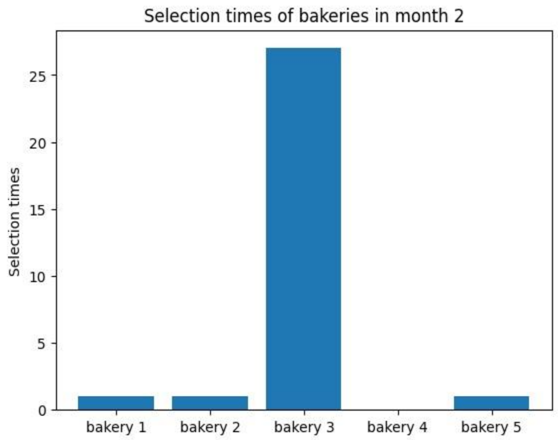
Beta Distribution after month 1



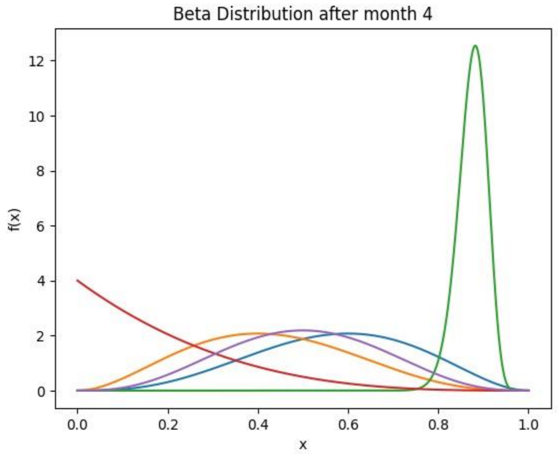
Selection times of bakeries in month 1



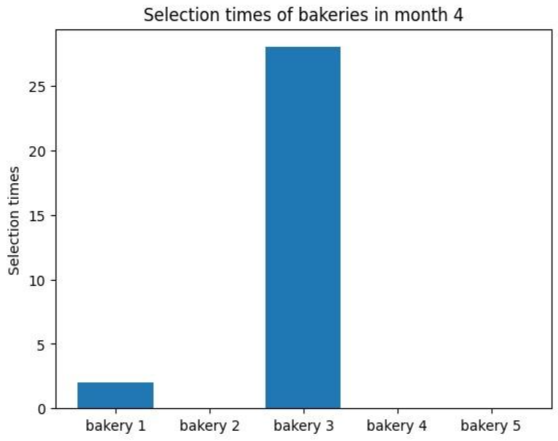
Beta Distribution after month 2



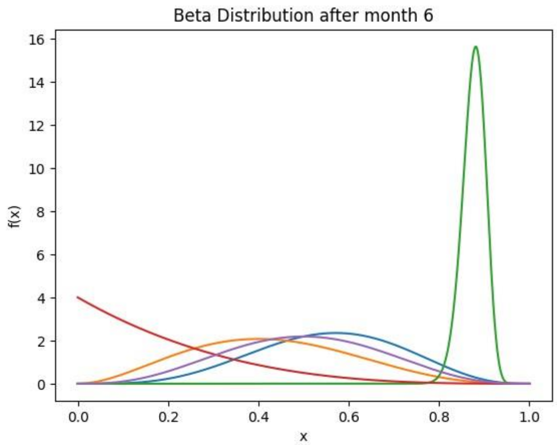
Selection times of bakeries in month 2

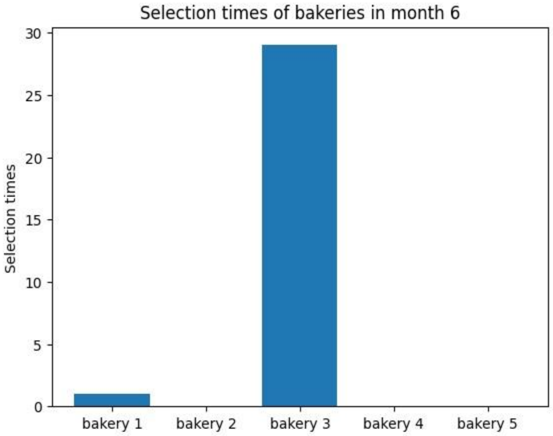


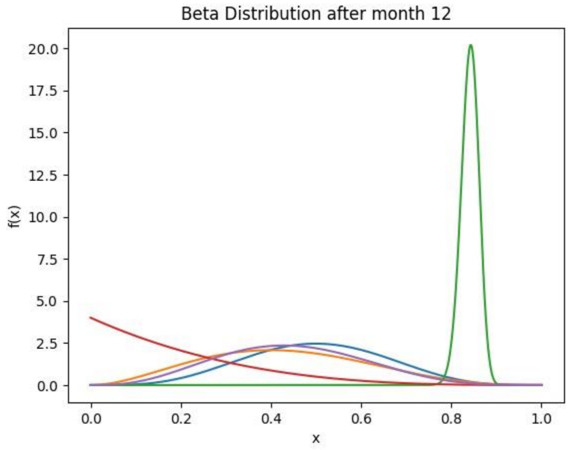
Beta Distribution after month 4

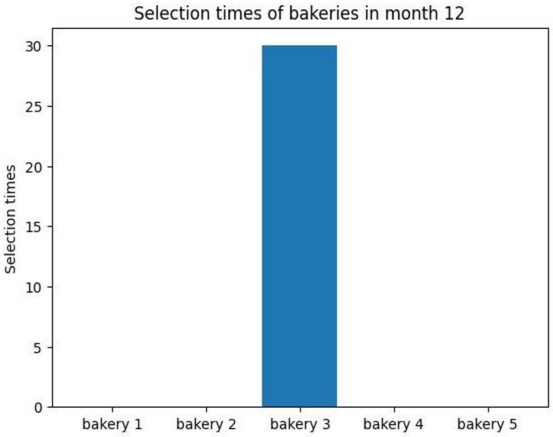


Selection times of bakeries in month 4





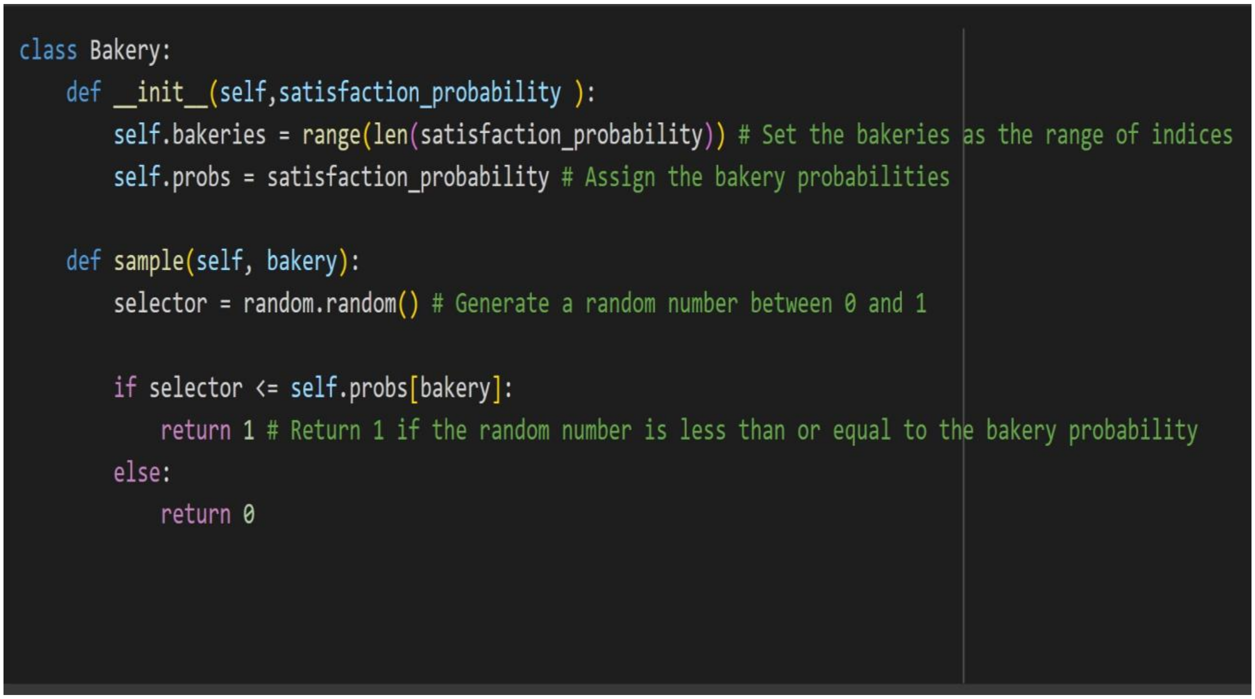




In the first month there are many days that we bought bread from different bakeries, but as we approach the end of the year the algorithm finds that the third bakery is the best one and buys more bread from it and in the last month as we see all of the breads are bought from the third bakery.

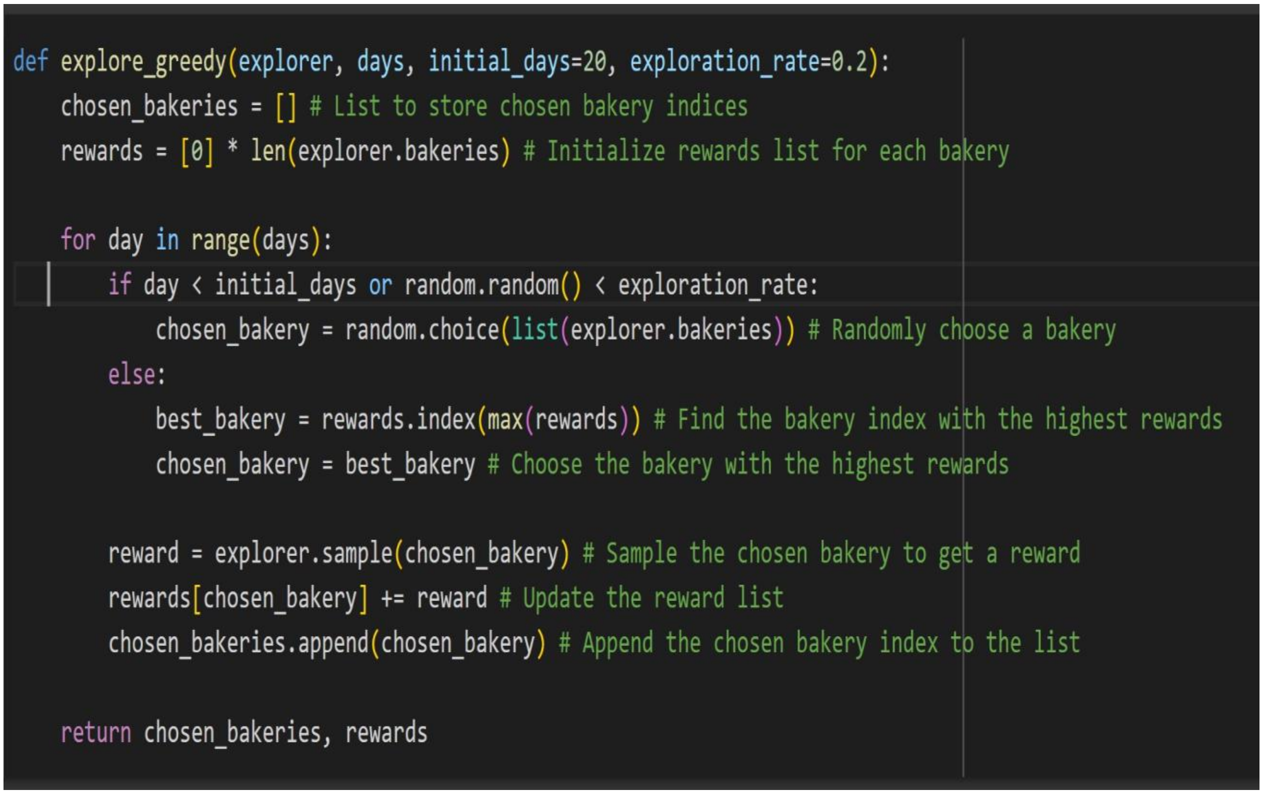
Third approach

Bakery class:



```
class Bakery:
    def __init__(self,satisfaction_probability ):
        self.bakeries = range(len(satisfaction_probability)) # Set the bakeries as the range of indices
        self.probs = satisfaction_probability # Assign the bakery probabilities

    def sample(self, bakery):
        selector = random.random() # Generate a random number between 0 and 1

        if selector <= self.probs[bakery]:
            return 1 # Return 1 if the random number is less than or equal to the bakery probability
        else:
            return 0
```

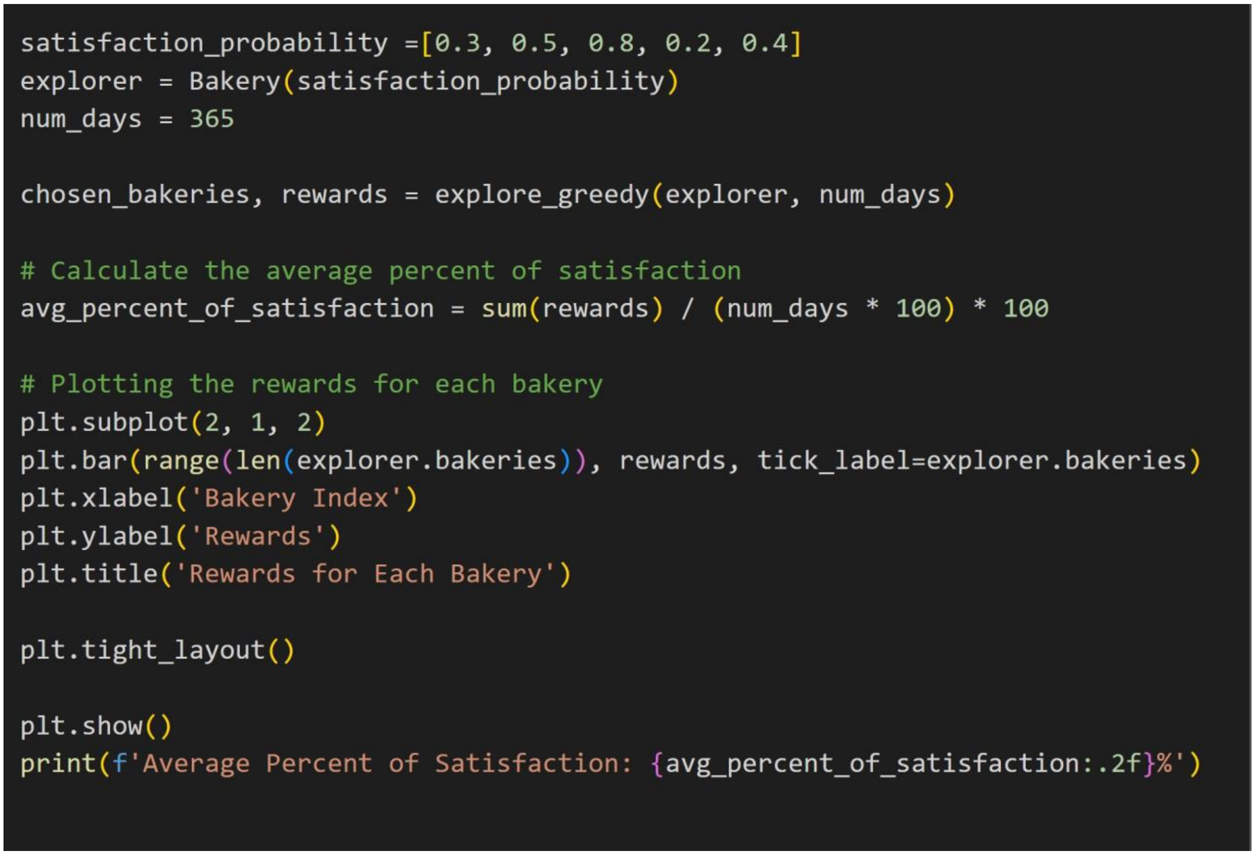
This class represents the bakery problem. explore_greedy Function:



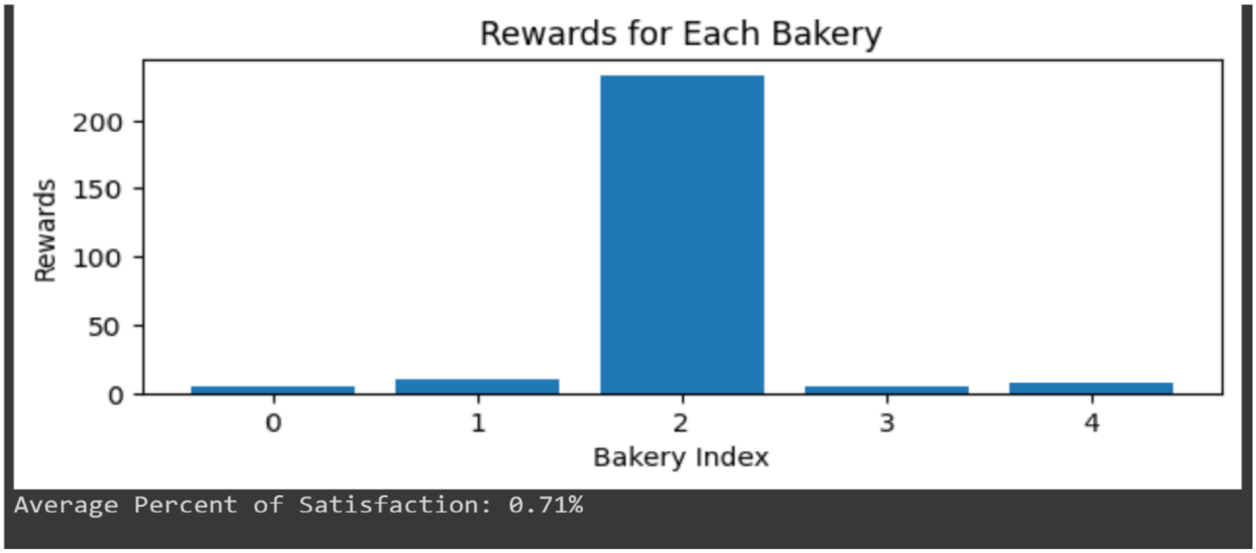
```
def explore_greedy(explorer, days, initial_days=20, exploration_rate=0.2):
    chosen_bakeries = [] # List to store chosen bakery indices
    rewards = [0] * len(explorer.bakeries) # Initialize rewards list for each bakery

    for day in range(days):
        if day < initial_days or random.random() < exploration_rate:
            chosen_bakery = random.choice(list(explorer.bakeries)) # Randomly choose a bakery
        else:
            best_bakery = rewards.index(max(rewards)) # Find the bakery index with the highest rewards
            chosen_bakery = best_bakery # Choose the bakery with the highest rewards

        reward = explorer.sample(chosen_bakery) # Sample the chosen bakery to get a reward
        rewards[chosen_bakery] += reward # Update the reward list
        chosen_bakeries.append(chosen_bakery) # Append the chosen bakery index to the list

    return chosen_bakeries, rewards
```

This function implements the explore_greedy approach for choosing bakeries.

Output and Results:



```
satisfaction_probability =[0.3, 0.5, 0.8, 0.2, 0.4]
explorer = Bakery(satisfaction_probability)
num_days = 365

chosen_bakeries, rewards = explore_greedy(explorer, num_days)

# Calculate the average percent of satisfaction
avg_percent_of_satisfaction = sum(rewards) / (num_days * 100) * 100

# Plotting the rewards for each bakery
plt.subplot(2, 1, 2)
plt.bar(range(len(explorer.bakeries)), rewards, tick_label=explorer.bakeries)
plt.xlabel('Bakery Index')
plt.ylabel('Rewards')
plt.title('Rewards for Each Bakery')

plt.tight_layout()

plt.show()
print(f'Average Percent of Satisfaction: {avg_percent_of_satisfaction:.2f}%')
```



```
Average Percent of Satisfaction: 0.71%
```

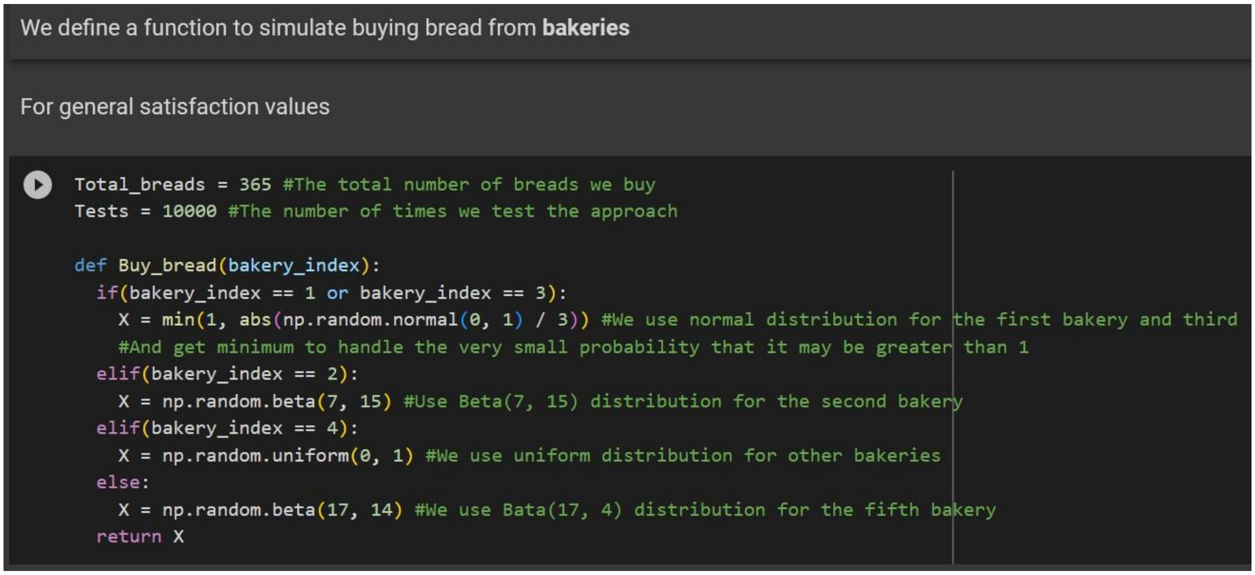
In 365 days the explore_greedy approach successfully selects bakeries based on their rewards. we got an average **71%** good bread by using this approach.

We found that the best way is the **Second Approach (Beta**

Distribution-Based Selection Strategy)with a 76.1% success rate.

We can **generalize this approach to satisfaction in the interval [0, 1]** and simulate this as follows:

(We will consider the satisfaction rates different distributions because this is not mentioned in the project description)
We just need to change the function of determining the level of satisfaction. (Normal, Beta and Uniform distribution are used)



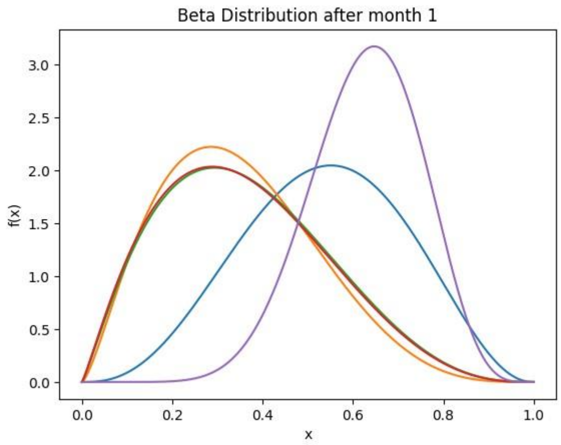
We define a function to simulate buying bread from **bakeries**

For general satisfaction values

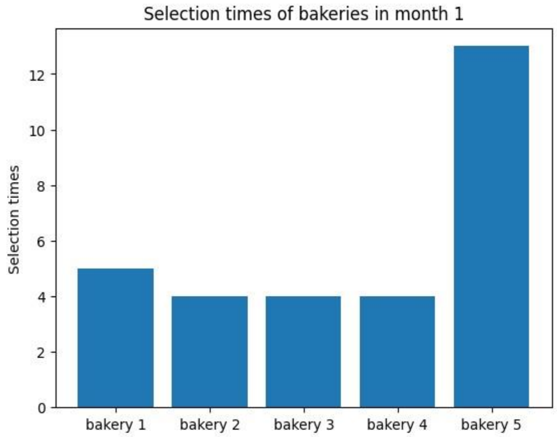
```
Total_breads = 365 #The total number of breads we buy
Tests = 10000 #The number of times we test the approach

def Buy_bread(bakery_index):
  if(bakery_index == 1 or bakery_index == 3):
    X = min(1, abs(np.random.normal(0, 1) / 3)) #We use normal distribution for the first bakery and third
    #And get minimum to handle the very small probability that it may be greater than 1
  elif(bakery_index == 2):
    X = np.random.beta(7, 15) #Use Beta(7, 15) distribution for the second bakery
  elif(bakery_index == 4):
    X = np.random.uniform(0, 1) #We use uniform distribution for other bakeries
  else:
    X = np.random.beta(17, 14) #We use Bata(17, 4) distribution for the fifth bakery
  return X
```

The results are as follows:

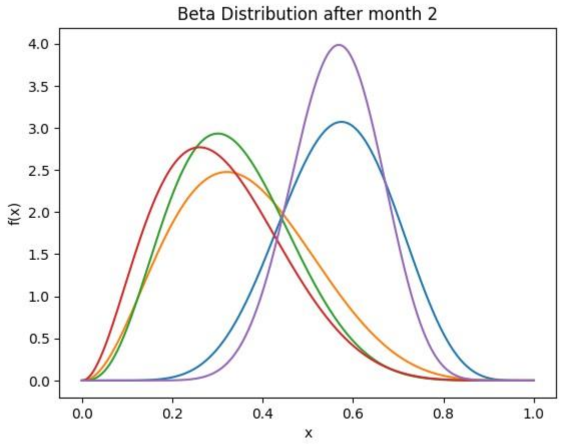
(Colors in the graph: Bakery 1 - Bekery 2 - Bekery 3 - Bekery 4 - Bekery 5)



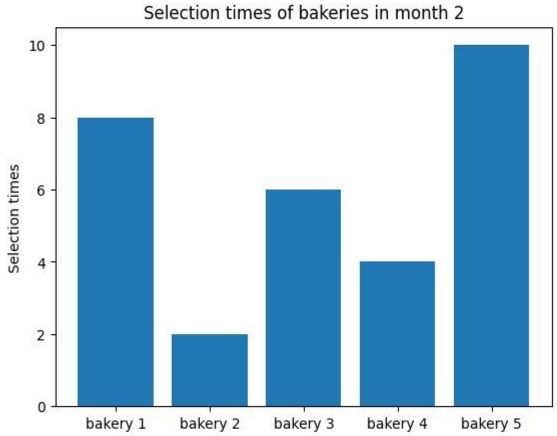
Beta Distribution after month 1



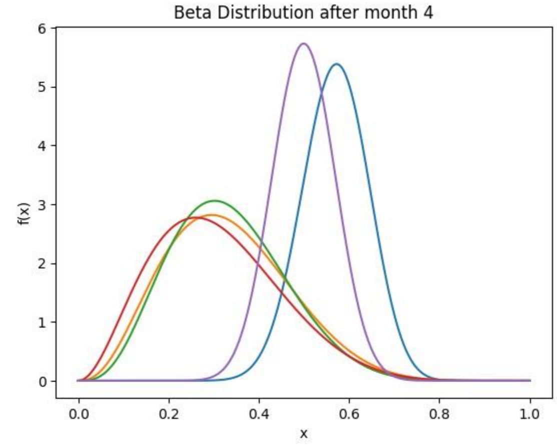
Selection times of bakeries in month 1



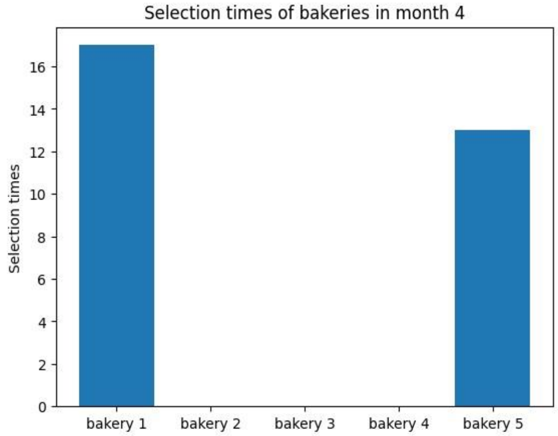
Beta Distribution after month 2



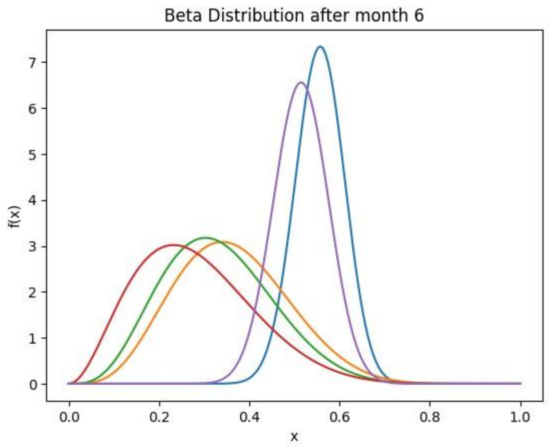
Selection times of bakeries in month 2

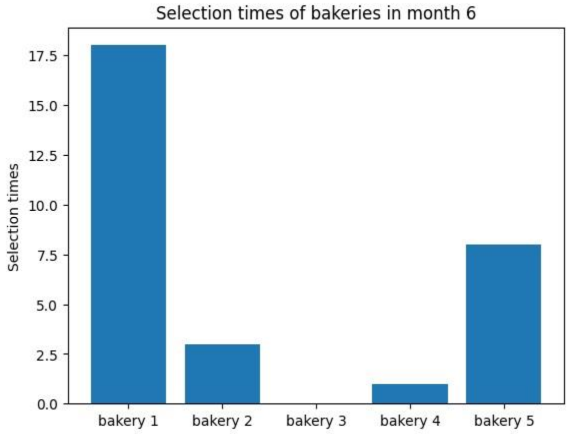


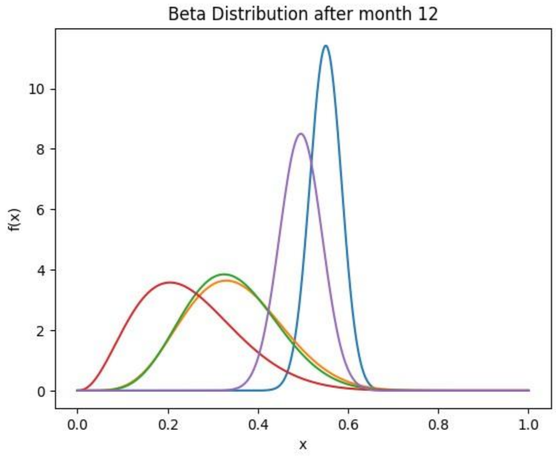
Beta Distribution after month 4

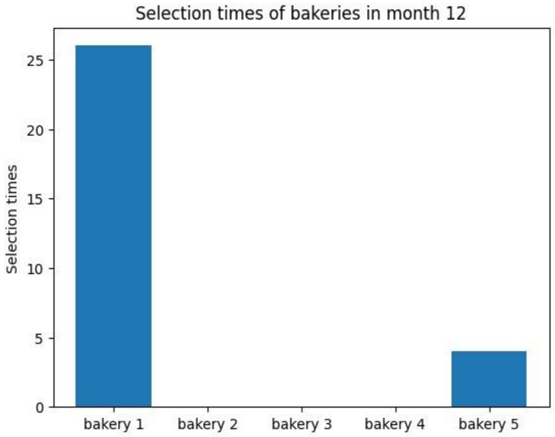


Selection times of bakeries in month 4









In the first months there are many days that we bought bread from different bakeries, but as we approach the end of the year the algorithm finds better bakeries and buys more bread from them and in the last month as we see all of the breads are bought from the first and fifth bakery which are better.

References

1. https://youtu.be/nkyDGGQ5h60
2. https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf
3. https://youtu.be/aVCImOiJklM
4. https://youtu.be/ZA4JkHKZM50
5. https://youtu.be/juF3r12nM5A
6. https://youtu.be/1k8lF3BriXM
7. https://towardsdatascience.com/thompson-sampling-fc28817eacb8
8. https://medium.com/analytics-vidhya/thompsonsampling-for-multi-armed-bandit-problem-68e4d367a21e
9. https://towardsdatascience.com/solving-multi-armed-bandit-problems-53c73940244a