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**

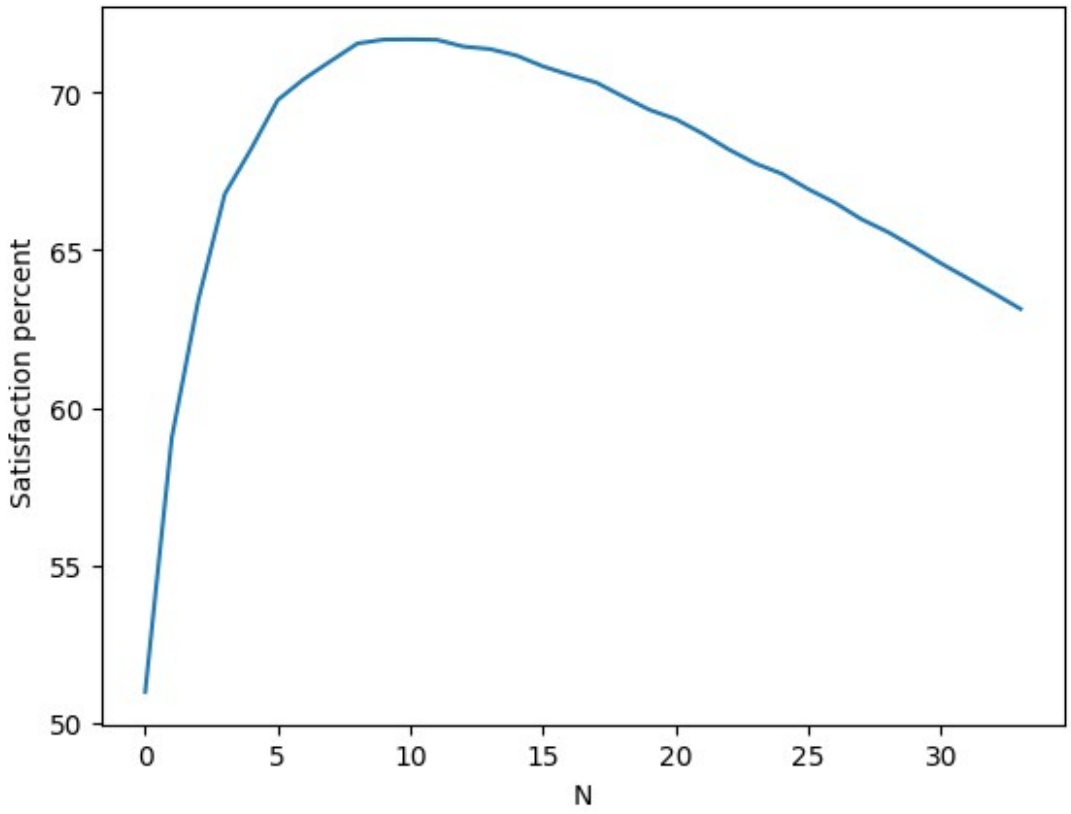
```
Total_breads = 365 #The total number of breads we buy
Tests = 10000 #The number of times we test the approach
satisfaction_probability = [0.3, 0.5, 0.8, 0.2, 0.4] #probability of
buying a good bread(for binary satisfaction)

def Buy_bread_general(bakery_index): #for general satisfaction(0, 1)
  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

def Buy_bread(bakery_index): #for binary satisfaction
  X = random()
  return 1 if X <= satisfaction_probability[bakery_index] else 0
```

First approach for choosing the bakery

```
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

results_list = list()

for N in range(1, 35):
  results = [First_approach(N) for _ in range(Tests)]
  Average_percent_of_satisfaction = sum(results) / len(results) /
Total_breads * 100
  results_list.append(Average_percent_of_satisfaction)

print(max(results_list), results_list.index(max(results_list)))
plt.xlabel("N")
plt.ylabel("Satisfaction percent ")
plt.plot(results_list)
plt.show()
  #print(Average_percent_of_satisfaction, "%")
```

```
71.67476712328768 10
```



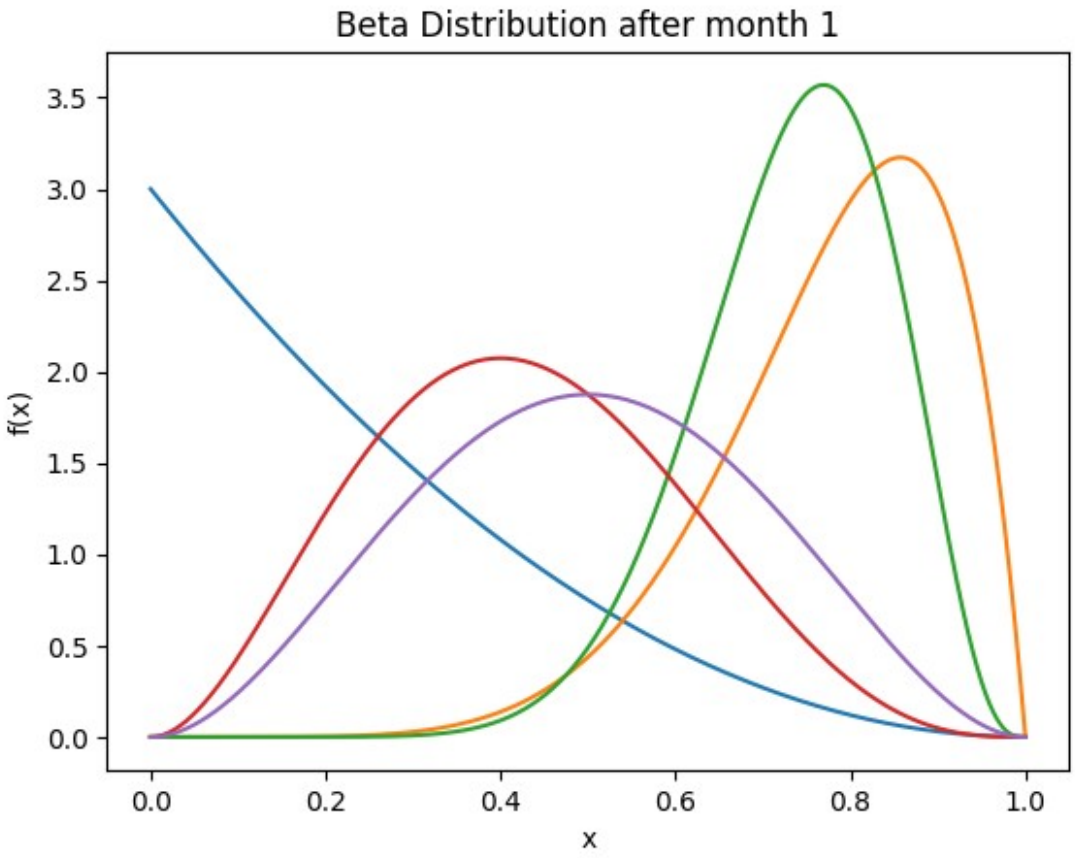
Second approach for choosing the bakery

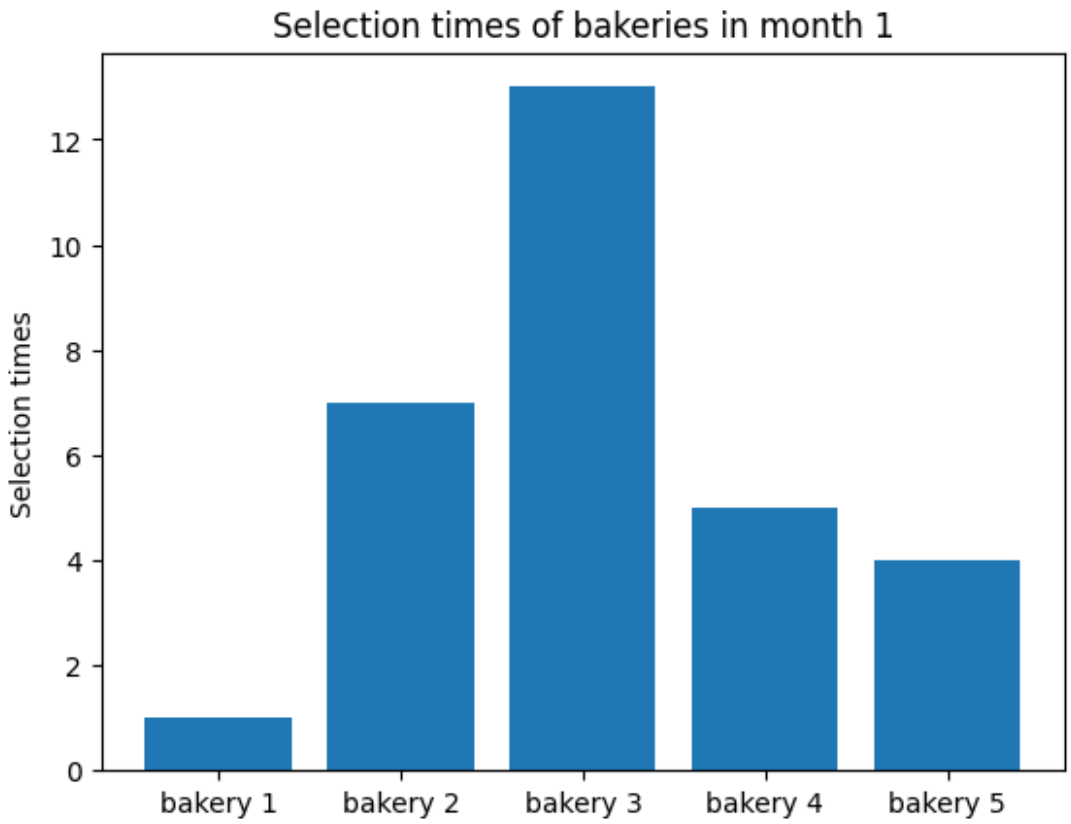
Binary satisfaction

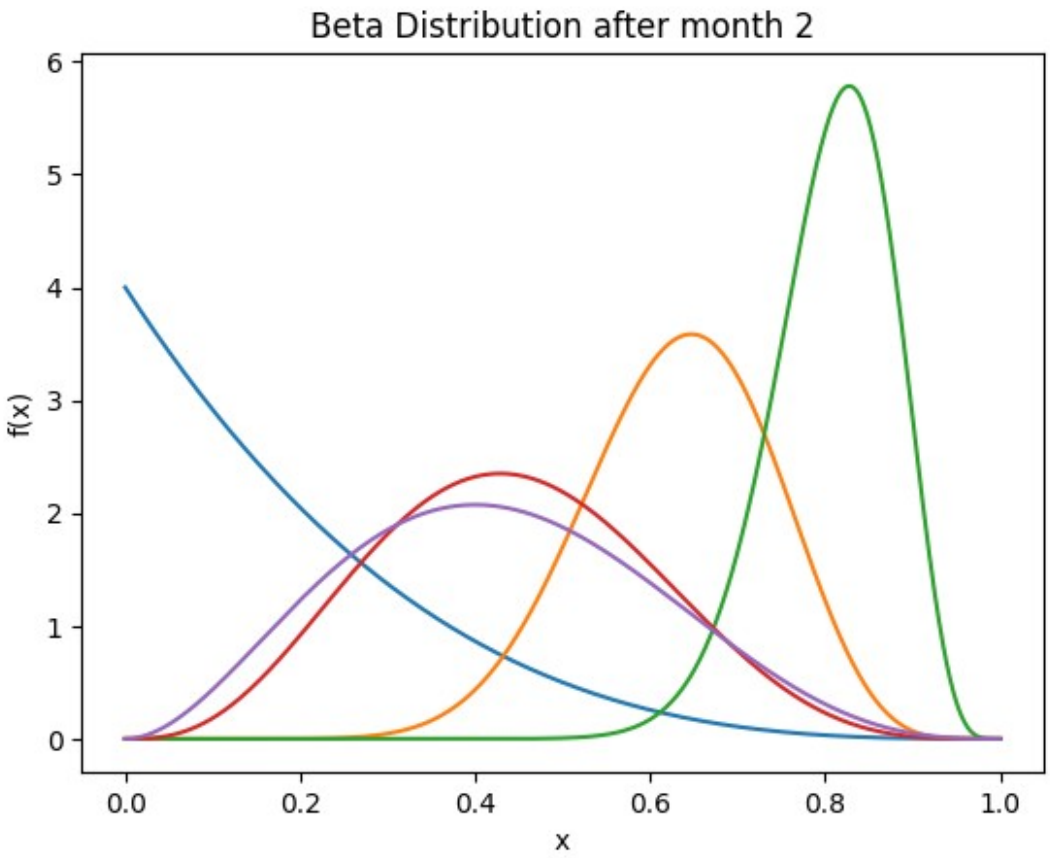
```
first_time = True #This is true just once for visualizing data
def Second_approach():
  global first_time
  success = [0] * 5 #An array to count good breads for each bakery
  failure = [0] * 5 #An array to count bad breads for each bakery

  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)],
```

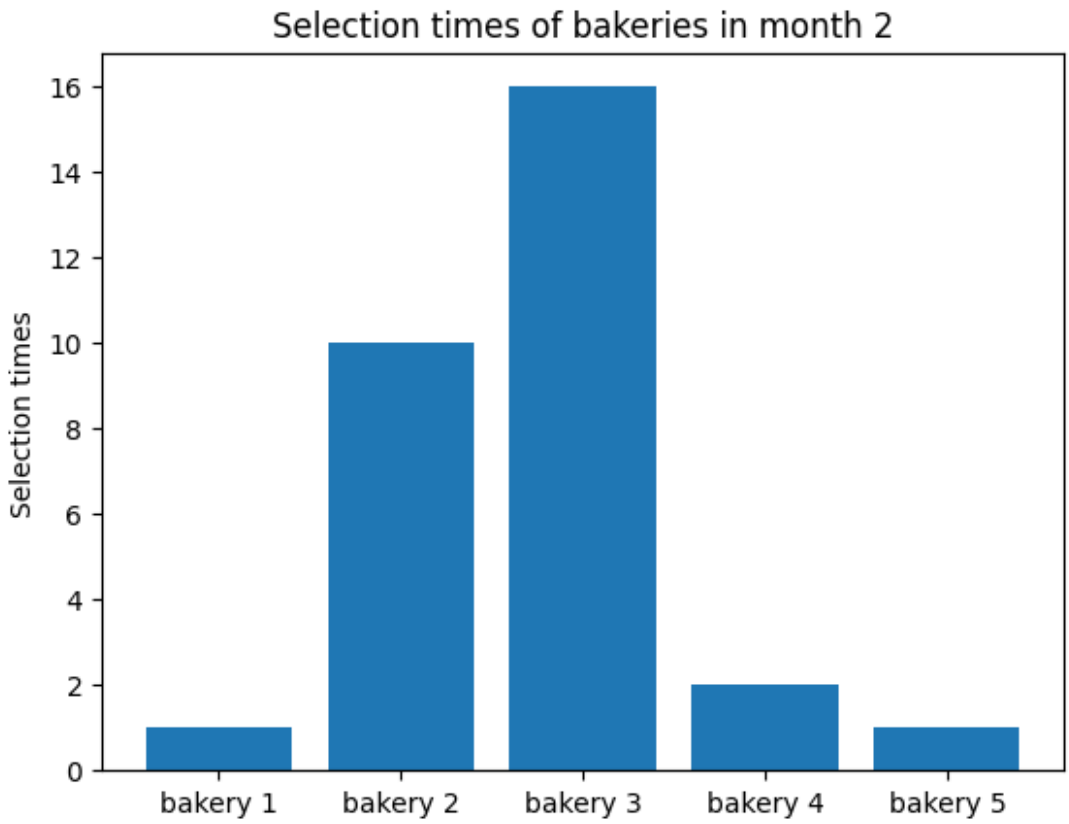
```
Selection_times)

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

  first_time = False
  return sum(success)

first_time = True
results = [Second_approach() for _ in range(Tests)]
Average_percent_of_satisfaction = sum(results) / len(results) /
Total_breads * 100
print(Average_percent_of_satisfaction, "%")
```

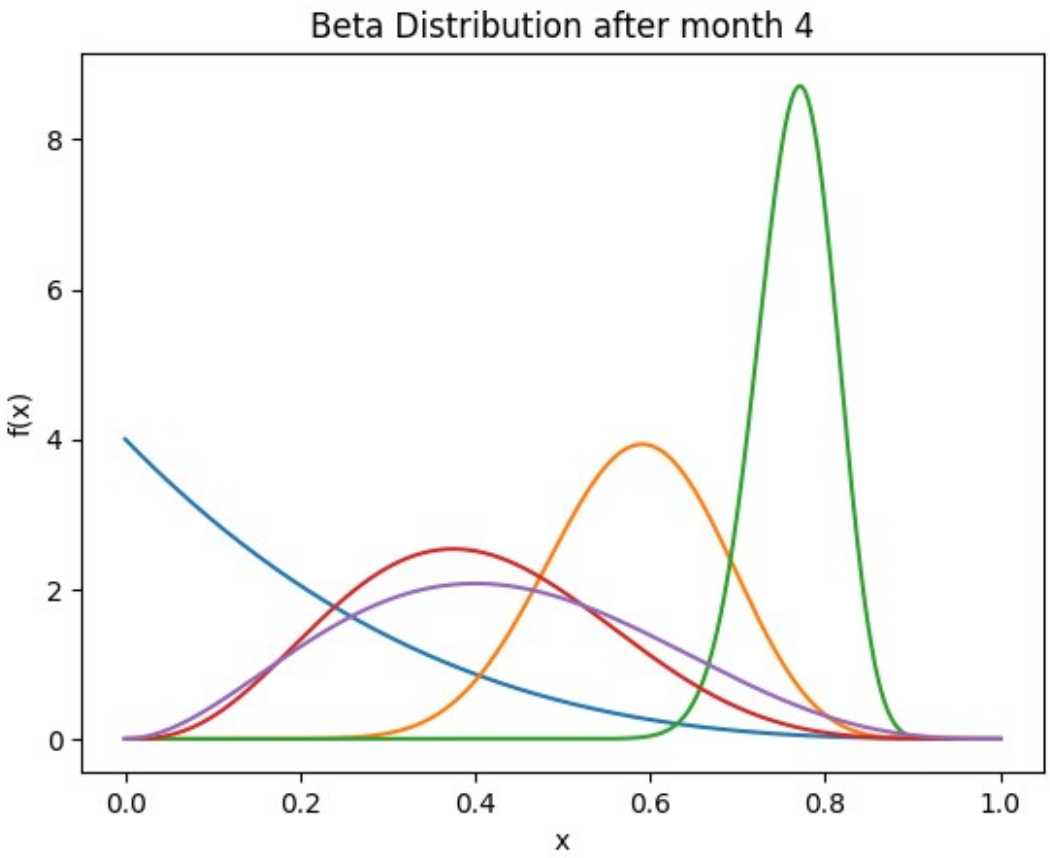
```
75.9748493150685 %
```

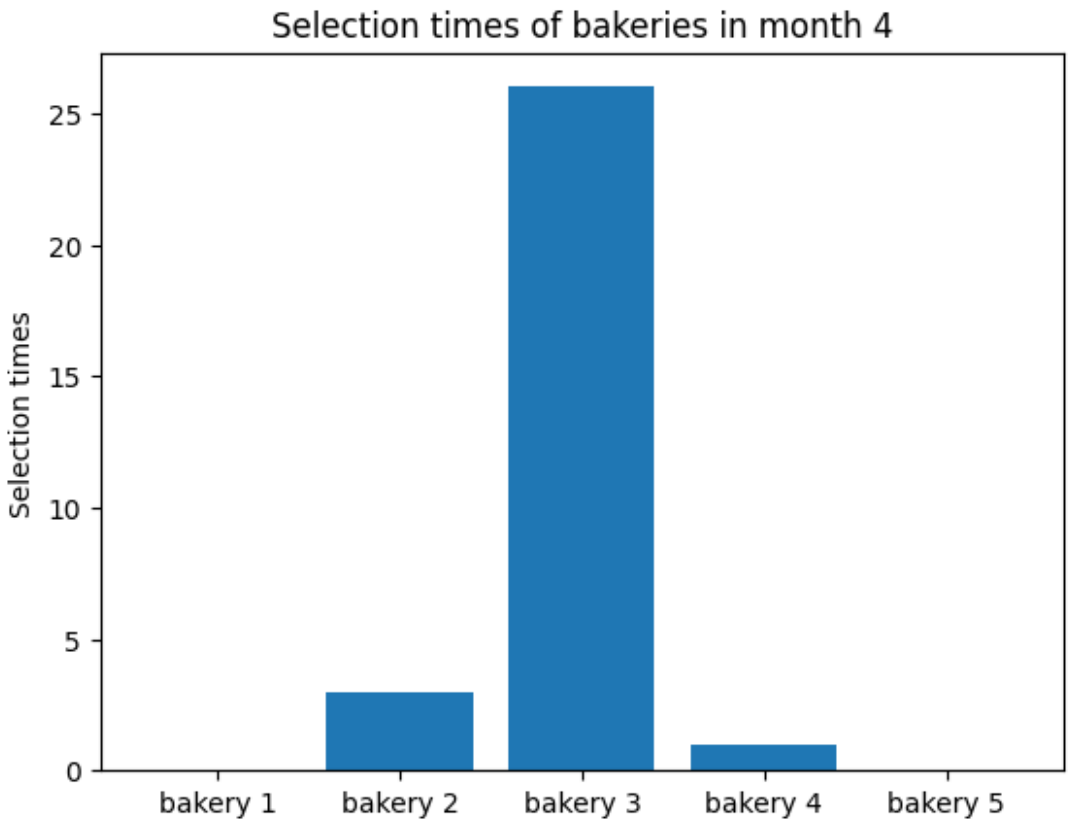


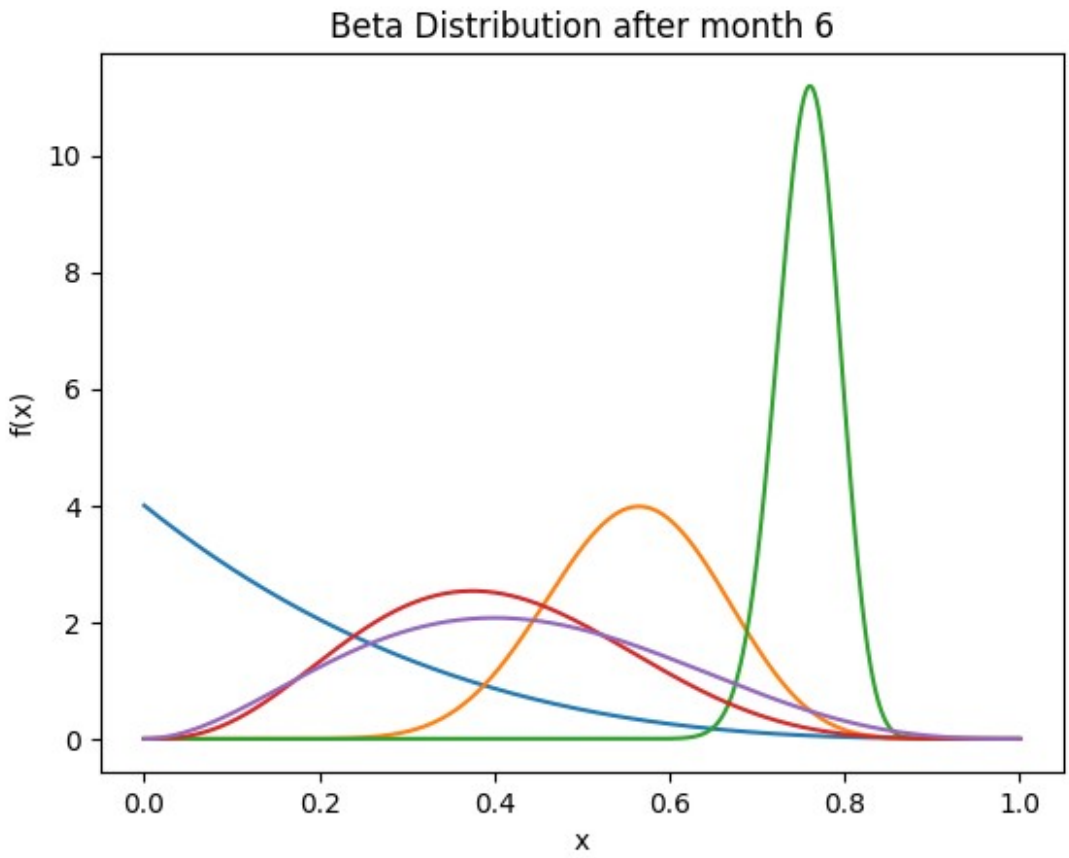

Selection times of bakeries in month 1
Selection times
0
2
4
6
8
10
12
bakery 1
bakery 2
bakery 3
bakery 4
bakery 5

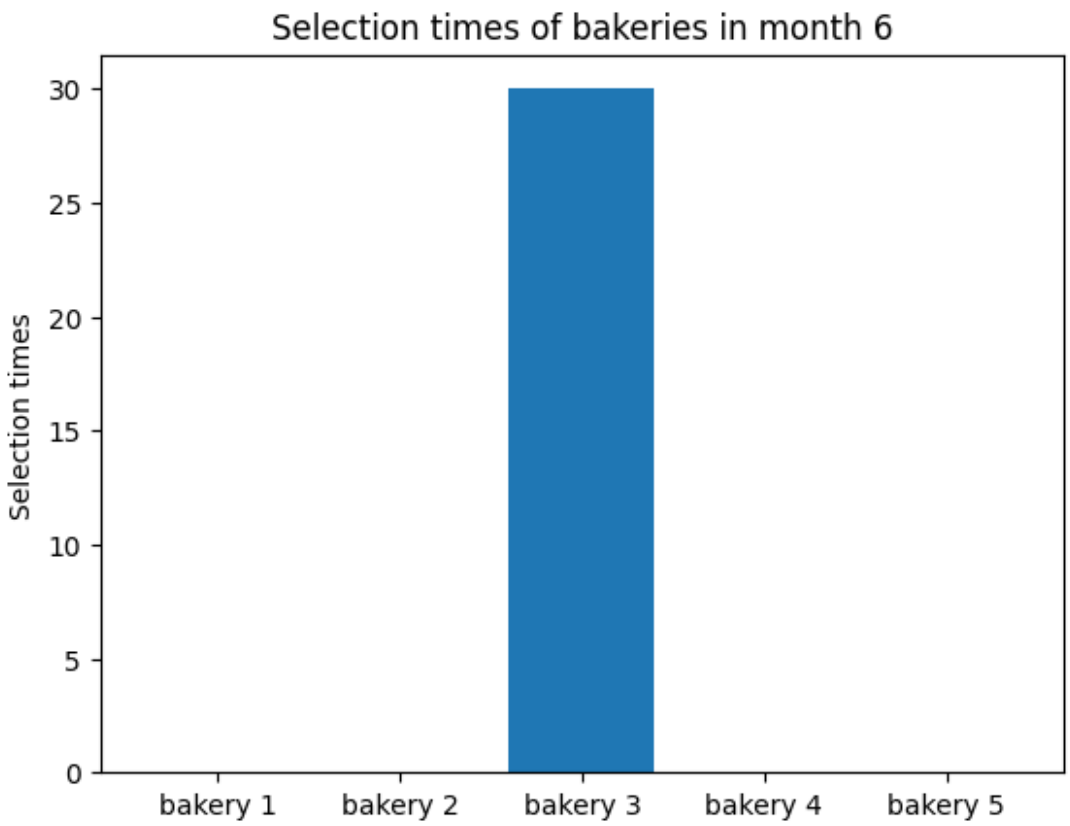

Beta Distribution after month 2
f(x)
x
0
1
2
3
4
5
6
0.0
0.2
0.4
0.6
0.8
1.0

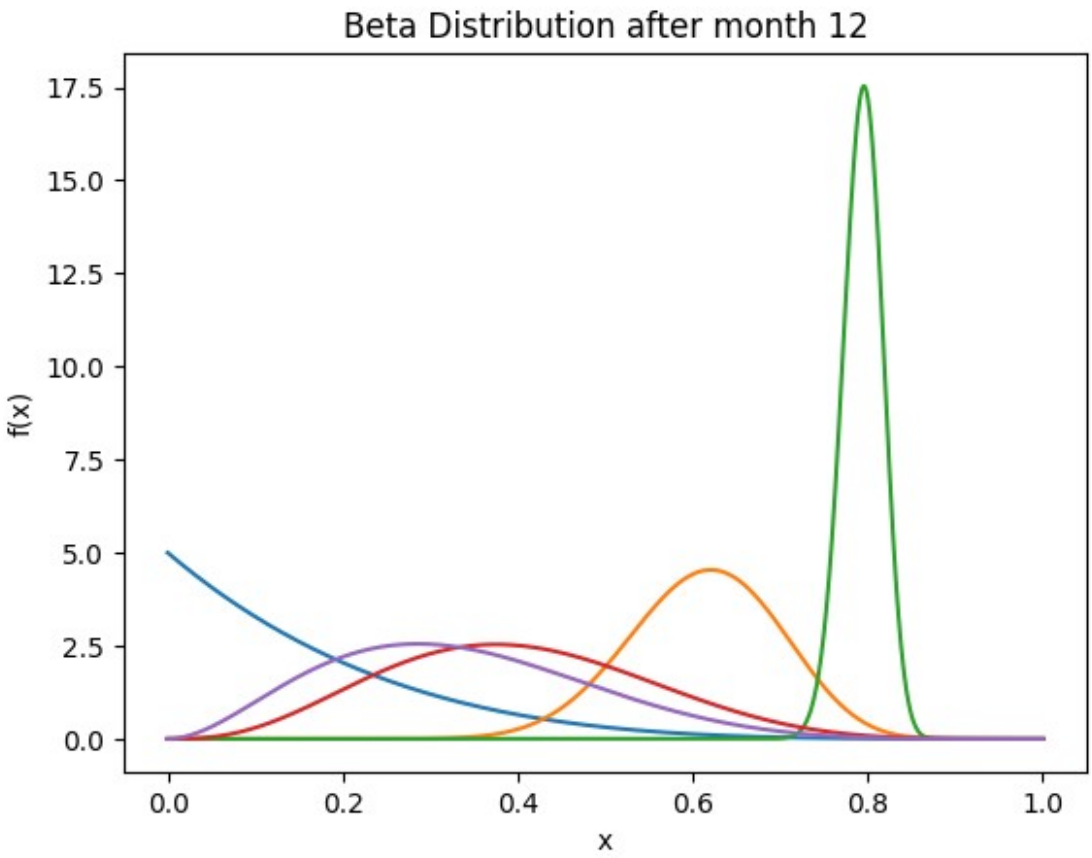

Selection times of bakeries in month 2
Selection times
0
2
4
6
8
10
12
14
16
bakery 1
bakery 2
bakery 3
bakery 4
bakery 5

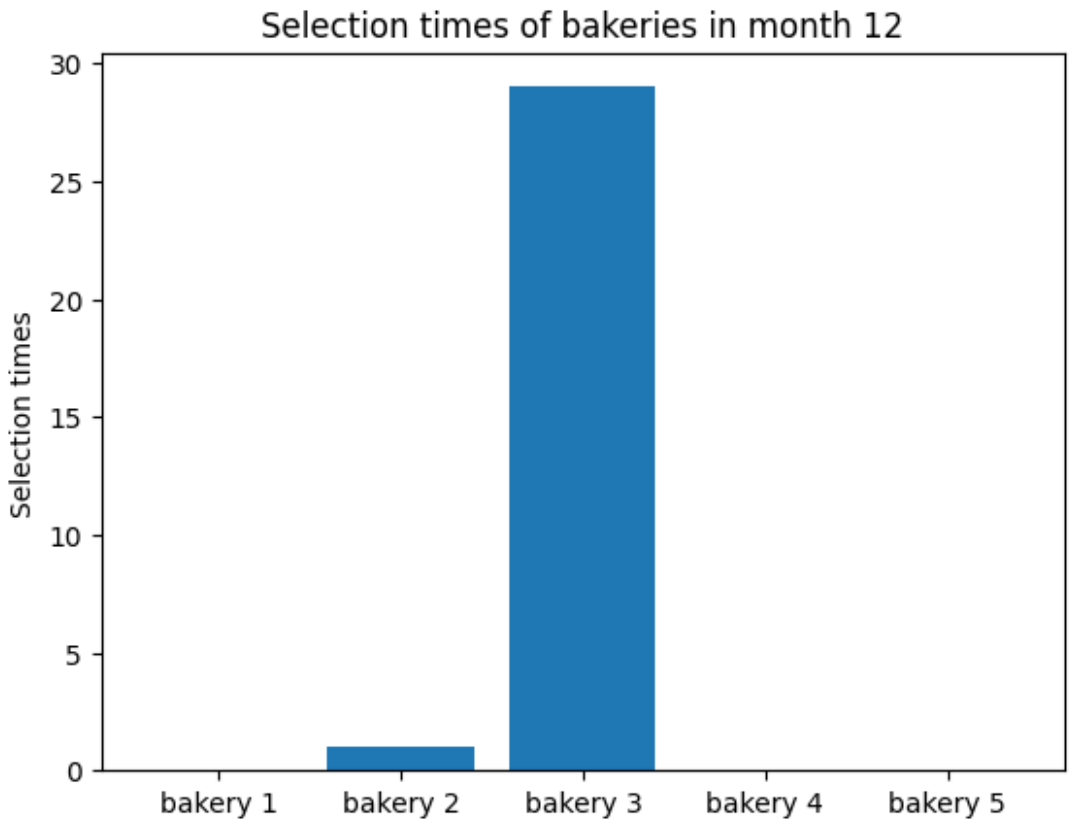

Beta Distribution after month 4
f(x)
x
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0


Selection times of bakeries in month 4
Selection times
0
5
10
15
20
25
bakery 1
bakery 2
bakery 3
bakery 4
bakery 5


Beta Distribution after month 6
f(x)
x
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0


Selection times of bakeries in month 6
Selection times
0
5
10
15
20
25
30
bakery 1
bakery 2
bakery 3
bakery 4
bakery 5


Beta Distribution after month 12
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
f(x)
0.0
0.2
0.4
0.6
0.8
1.0
x

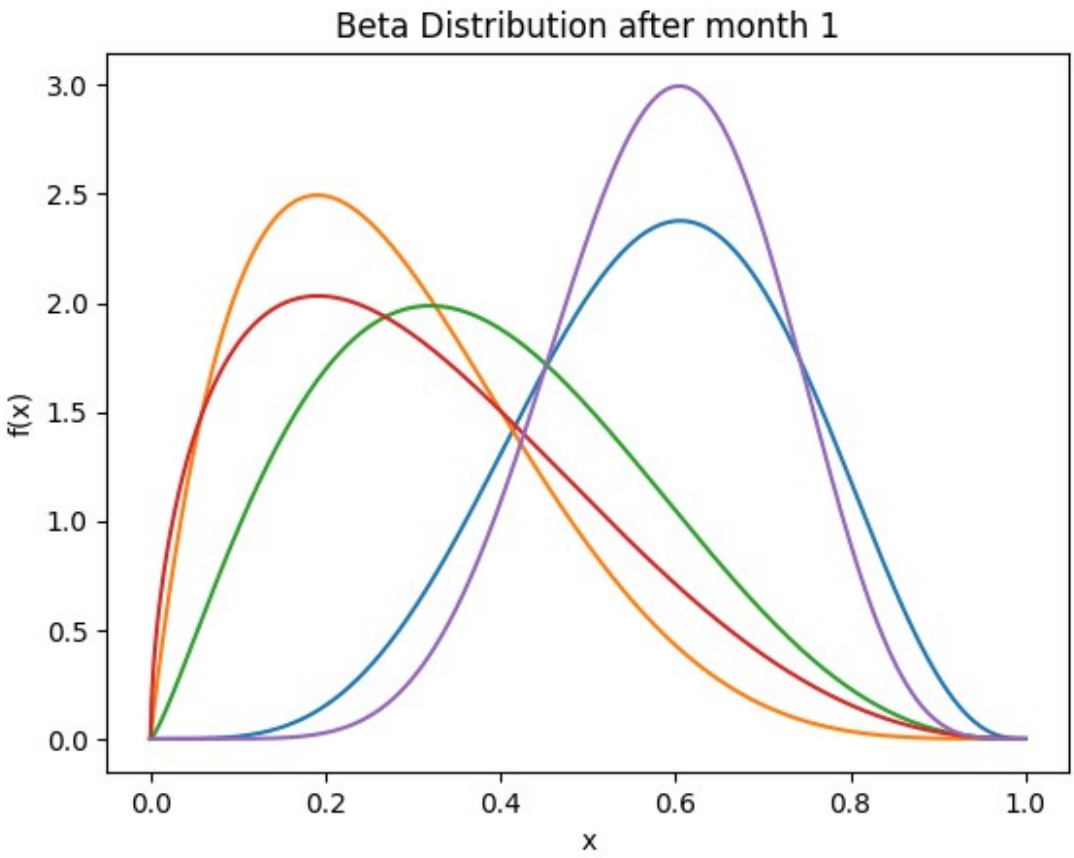


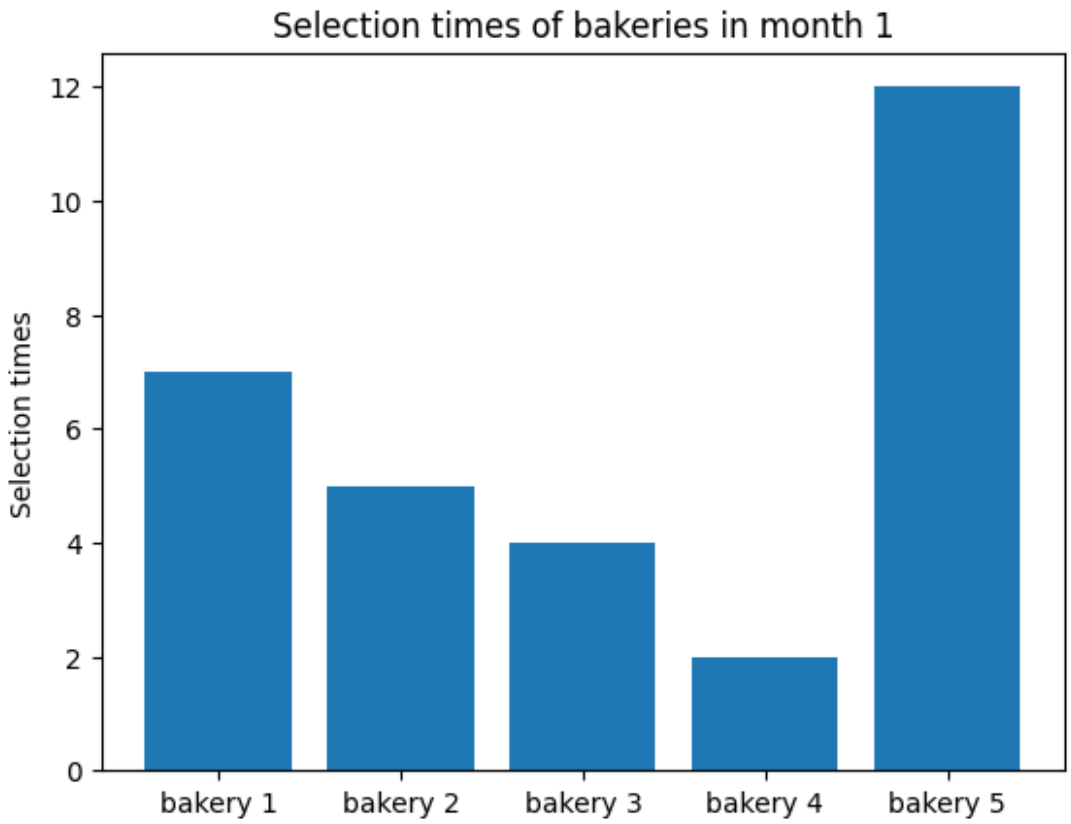
General satisfaction

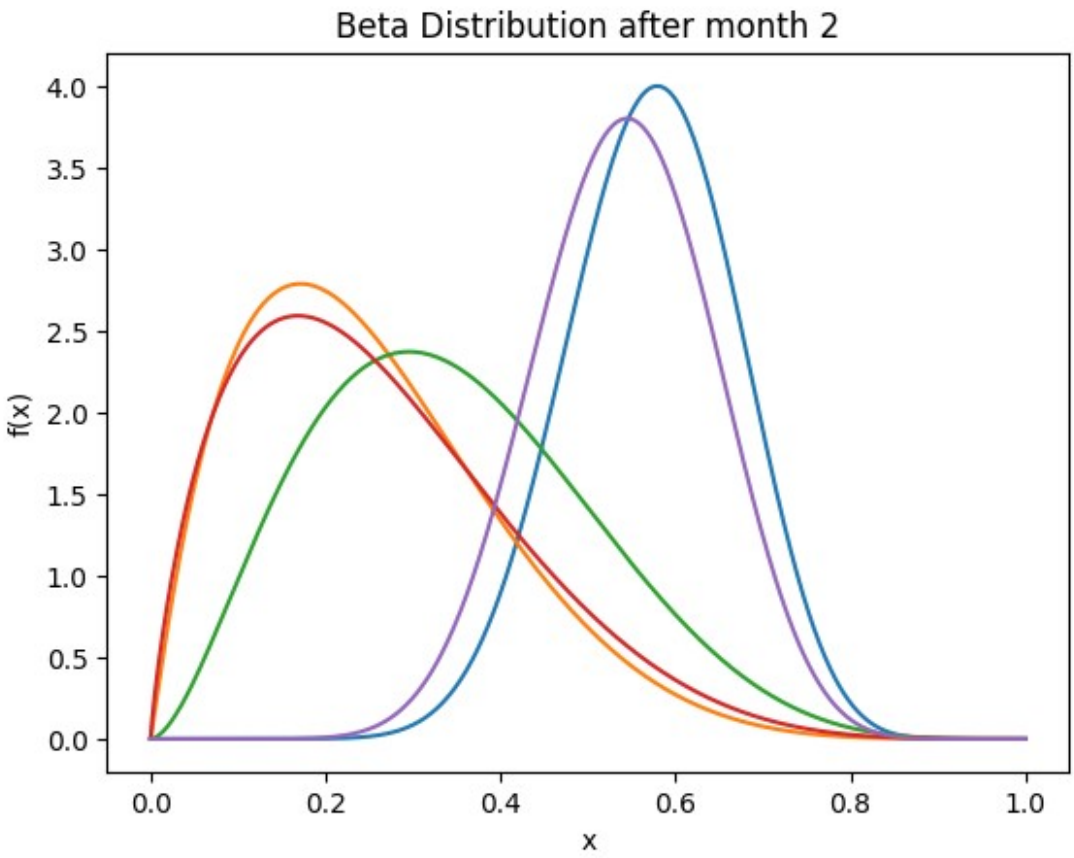
```
first_time = True #This is true just once for visualizing data
def Second_approach():
  global first_time
  success = [0] * 5 #An array to count good breads for each bakery
  failure = [0] * 5 #An array to count bad breads for each bakery

  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_general(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) #""
```

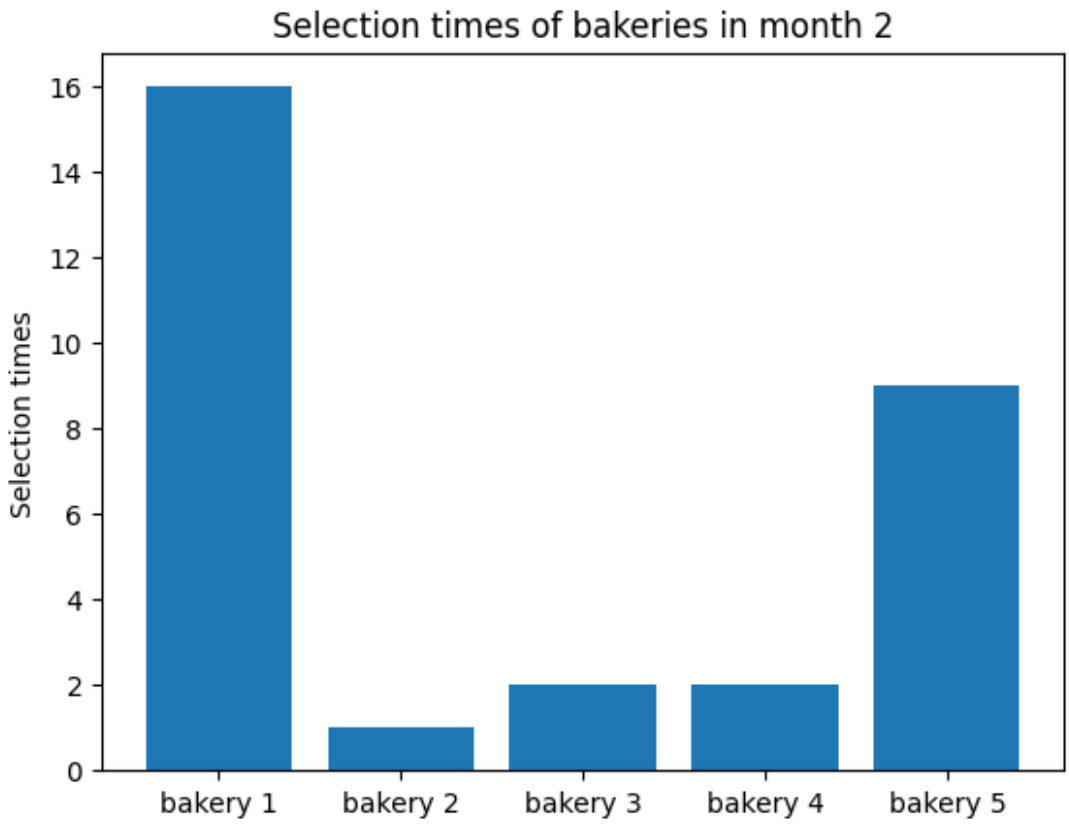
```
    if not first_time: #We just want to visualize at the first test,
Others are for making the conclusion more precise
      continue


    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)

first_time = True
results = [Second_approach() for _ in range(Tests)]
Average_percent_of_satisfaction = sum(results) / len(results) /
Total_breads * 100
print(Average_percent_of_satisfaction, "%")
```

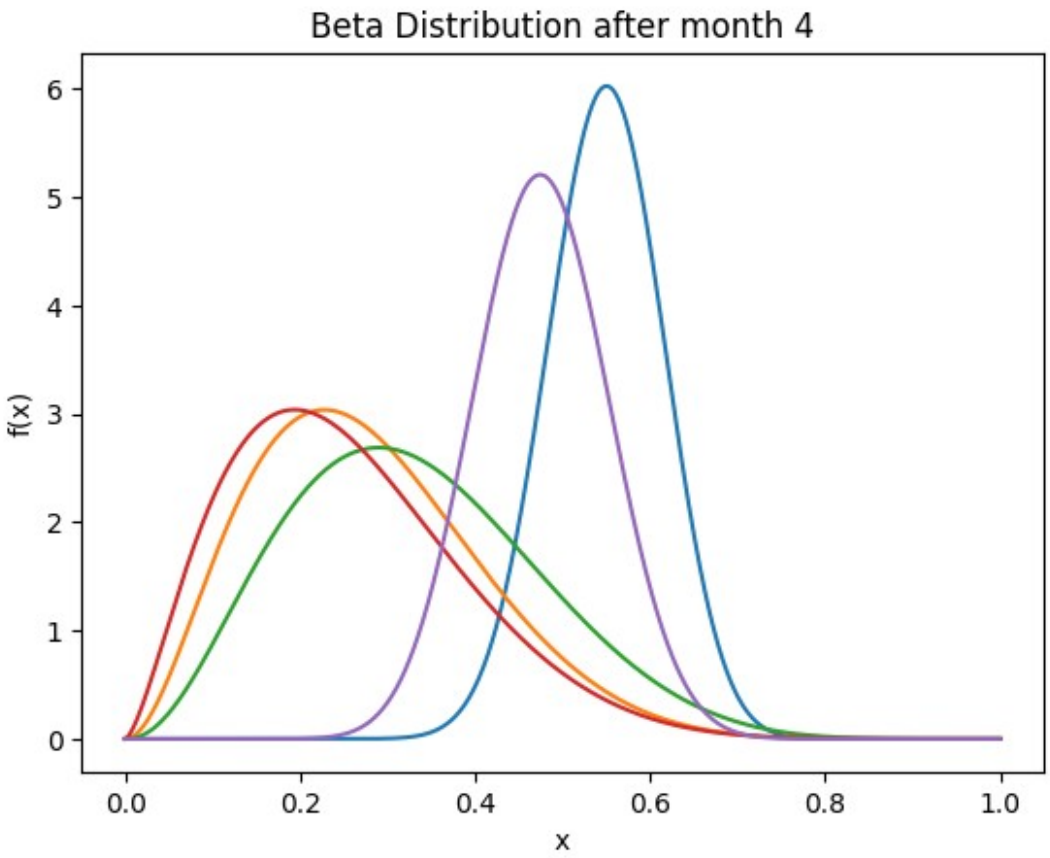
```
49.97832593832741 %
```

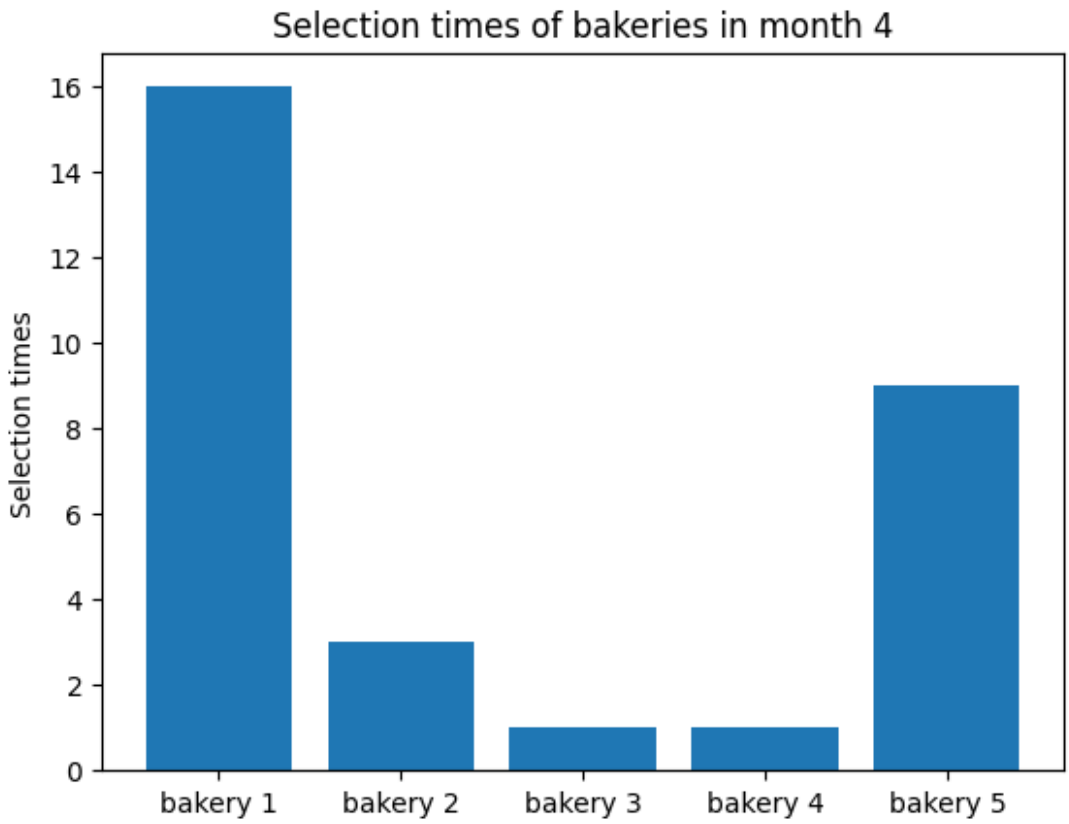

Beta Distribution after month 1
f(x)
x
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.2
0.4
0.6
0.8
1.0

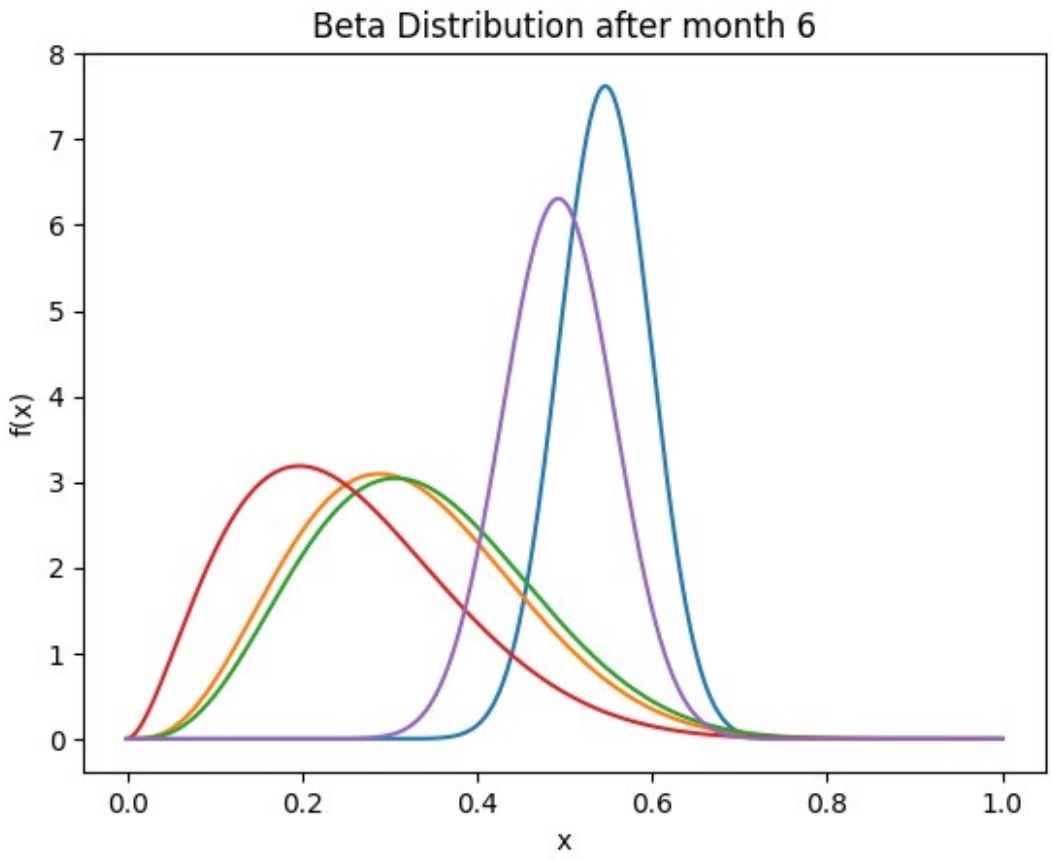

Selection times of bakeries in month 1
Selection times
0
2
4
6
8
10
12
bakery 1
bakery 2
bakery 3
bakery 4
bakery 5

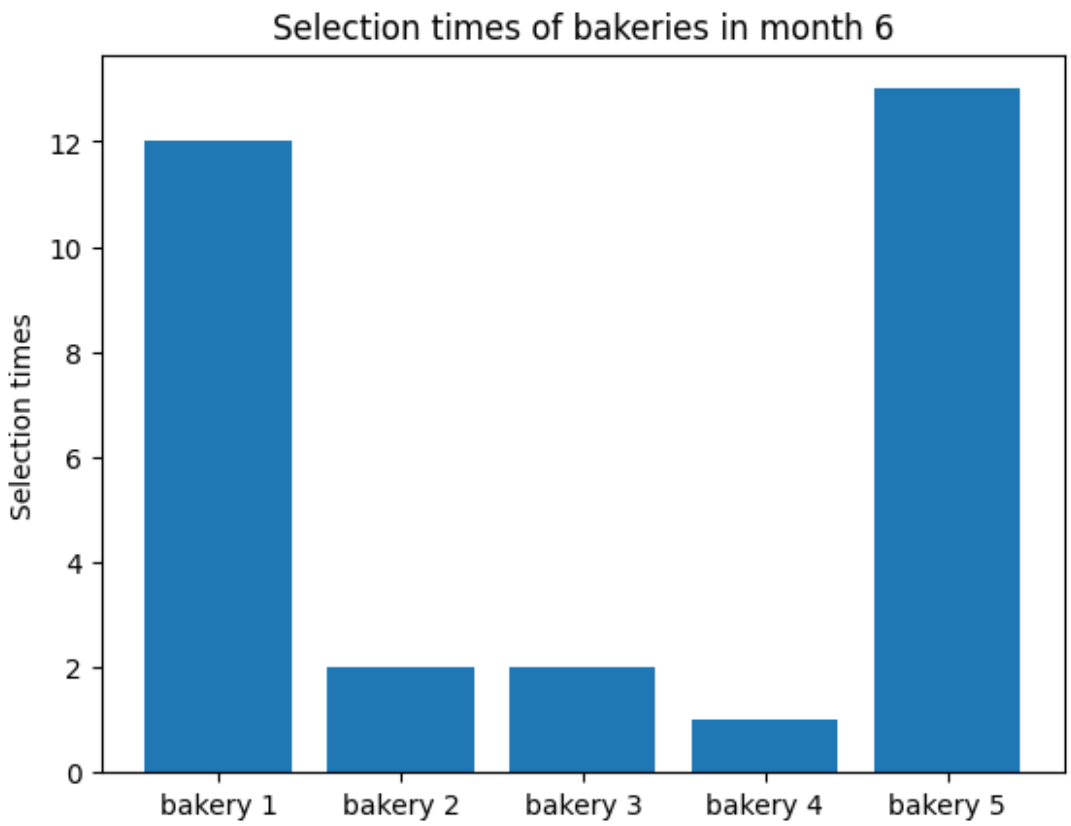

Beta Distribution after month 2
f(x)
x
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
0.2
0.4
0.6
0.8
1.0

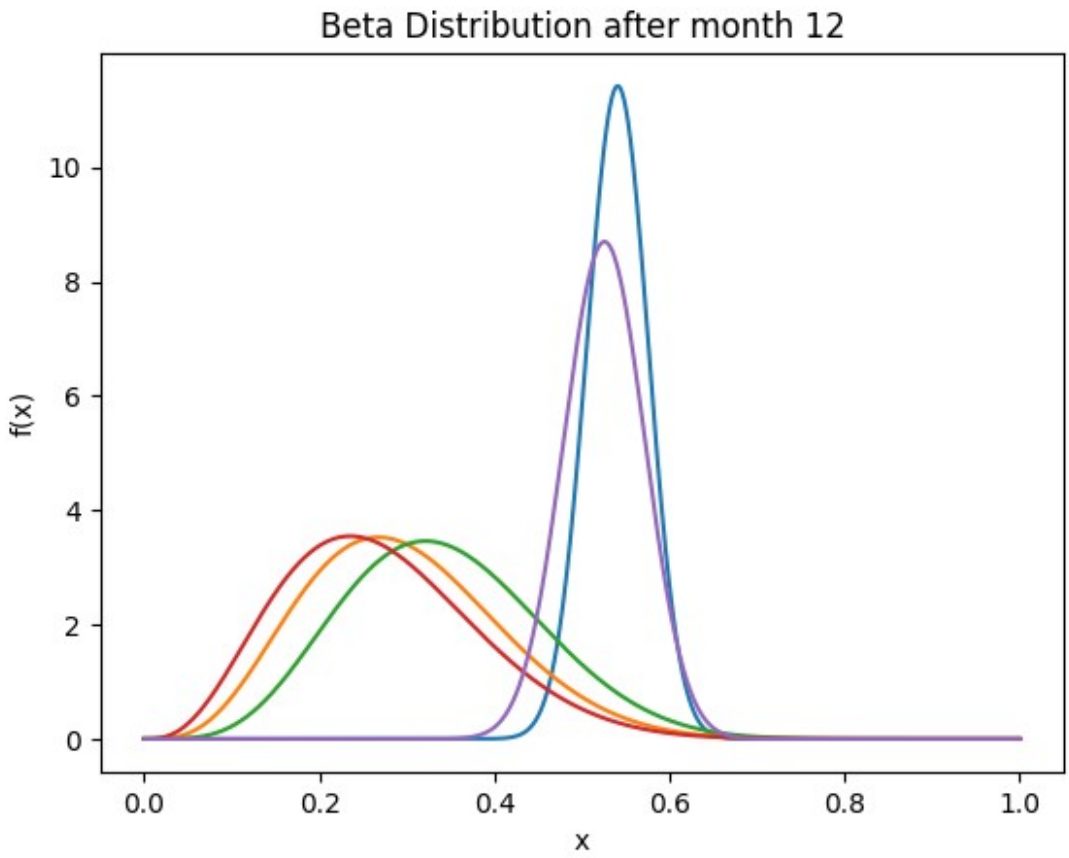

Selection times of bakeries in month 2
Selection times
0
2
4
6
8
10
12
14
16
bakery 1
bakery 2
bakery 3
bakery 4
bakery 5

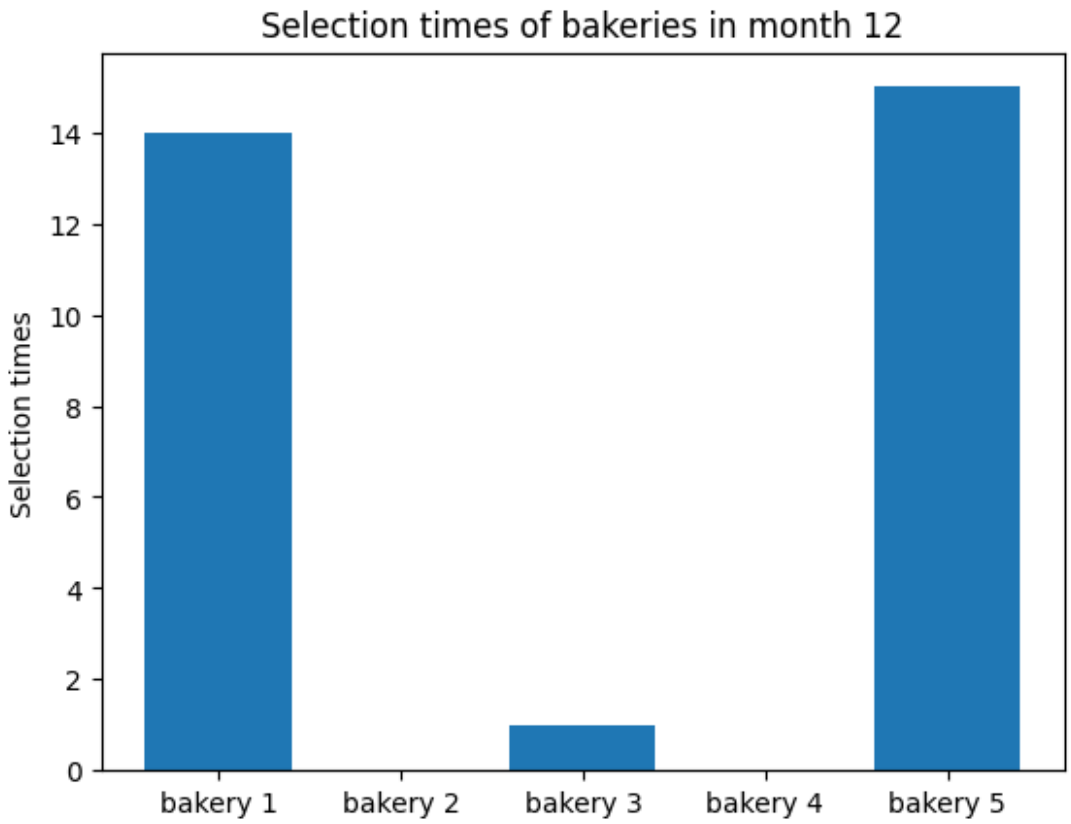

Beta Distribution after month 4
f(x)
x
0
1
2
3
4
5
6
0.0
0.2
0.4
0.6
0.8
1.0


Selection times of bakeries in month 4
Selection times
0
2
4
6
8
10
12
14
16
bakery 1
bakery 2
bakery 3
bakery 4
bakery 5


Beta Distribution after month 6
f(x)
x
0
1
2
3
4
5
6
7
8
0.0
0.2
0.4
0.6
0.8
1.0


Selection times of bakeries in month 6
Selection times
0
2
4
6
8
10
12
bakery 1
bakery 2
bakery 3
bakery 4
bakery 5

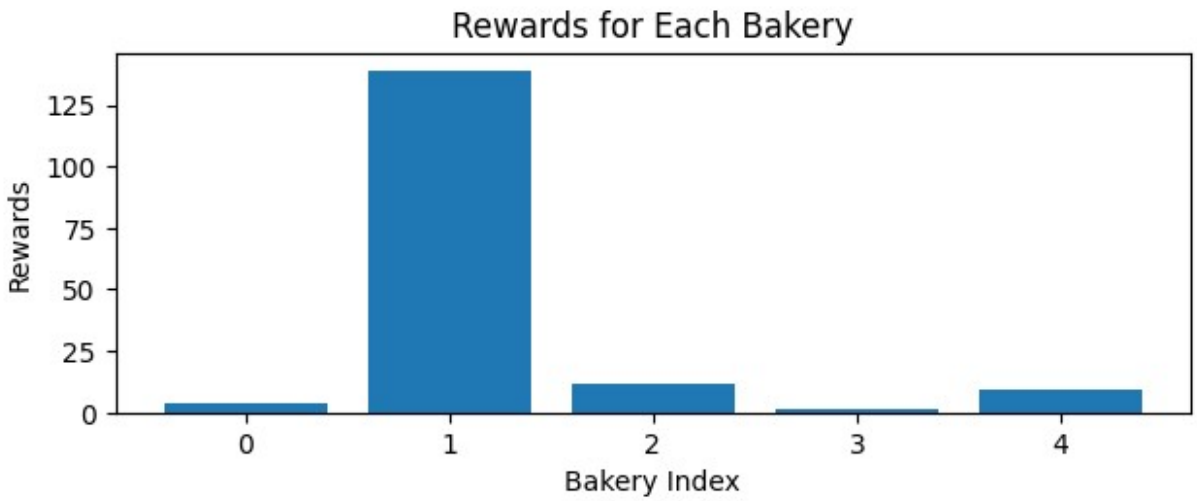

Beta Distribution after month 12
f(x)
x
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0



Third approach for choosing the bakery

```
import random
import matplotlib.pyplot as plt

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
```

```
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


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: 0.45%
```