Academic year 2020/2021

Data Intelligence Application Project

Pricing and Matching

Offline and Online learning

Barbieri Fabio
Castellazzi Luca
Mussi Giulia

Professor: *Gatti Nicola*
Tutor: *Bernasconi De Luca Martino*

September 20, 2021

Contents

1 Introduction

The aim of this work is to study an online pricing and matching problem, step by step, starting from its offline version, and gradually reaching a more general formulation - and solution - of it. To give a more concrete idea of the study, we thought of a practical application, which we used as an intuitive reference to model all the variables and parameters involved.

The setting is the following: a shop has a number of promo codes to incentivise the customers, that buy a first item, to buy a second different item. These customers can belong to four different classes and there are three promo codes that can provide different discounts. In our example, we thought of an outdoor sports shop, selling mountaineering boots as first item, and possibly a pair of trekking sticks as second item. The four client types we thought about are:

[CL0] Junior Professionals (less than 30 y.o.)
[CL1] Junior Amateurs
[CL2] Senior Professionals
[CL3] Senior Amateurs.

The customer visiting the shop, may or may not buy the boots, at a given price, according to her/his conversion rate. In case of purchase, she/he can decide to buy the sticks too, which are discounted or not, and the discount is applied *a priori* by the business unit (meaning that the promo is assigned independently of the choice of buying the first item). The study is performed during an entire year, and each client entering the shop provides data and information to update the knowledge of the bandit machine.

Before diving into the solution of the problem, we introduce two different settings, thought as two different choices of daily promo allocations, to test our solution in two possible contexts. The first setting, referred to as *setting0*, sees `PROMO_PROB = [0.4, 0.2, 0.22, 0.18]`, so we have 20% of the customers getting the discount identified by P1, 22% taking P2, P3 going to 18% of them, and the remaining 40% receives no promo (P0). On the other hand, *setting1* is described by `PROMO_PROB = [0.2, 0.3, 0.1, 0.4]`. The mentioned P1, P2 and P3 are the percentage discounts, and are respectively equal to 15%, 25% and 40%.

2 Offline problem

The offline problem can be seen as a Linear Programming problem, where an objective function and some suitable constraints need to be provided. The goal is to find the *best* couple of prices and the *best* matching between customers and promos, where *best* is intended as the solution that maximises the daily reward. In this setting we assume as known parameters the number of customers per class arriving each day, the conversion rates $cr(p, cl)$ associated with all the possible prices and classes, and the reward for each customer. N is the total number of customers visiting the shop per day, which must equal the amount of promo codes (both effective and fictitious).

2.1 Mathematical formulation

Objective function

$$\max \sum_{i=1}^{N} \sum_{j=1}^{N} cr_1(p_1, cl_i) \, \{ \, m_1 \; + \; x_{ij} \; cr_2(p_2 \, (1 - promo_j), cl_i) \; m_2(promo_j) \, \} \tag{1}$$

where $m_1 = p_1 - c_1$ is the margin from the sale of item 1, defined as price p_1 deduced by its cost c_1, and similarly $m_2(promo_j) = p_2 \, (1 - promo_j) - c_2$ is the margin obtained by the second item, sold with the promo specified by $promo_j$. Finally, $x_{ij} \in \{0, 1\}$ is a binary variable representing our unknown, so the matching of client i with promo j.

Constraints

$$\begin{aligned} \sum_{i=1}^{N} x_{ij} \leq 1 \quad & \forall j = 1, 2, \ldots, N \\ \sum_{j=1}^{N} x_{ij} \leq 1 \quad & \forall i = 1, 2, \ldots, N \end{aligned} \tag{2}$$

meaning that we want to assign every customer to one and only one promo, and viceversa.

This optimization problem has to be solved for every candidate couple of prices (p_1, p_2), and the best solution is given by the interdependent combo of prices (p_1, p_2) and matching $\{x_{ij}(p_1, p_2)\}_{ij}$ maximising the objective function (daily reward).

3 Variables introduction

Among all the quantities introduced in section 2, we identified some non deterministic ones which we modelled with random variables, in the online setting.

- **Number of customers per class per day**: for each of the four classes we chose a suitable truncated Gaussian distribution by specifying their means, standard deviations and the boundaries of the distribution. See Appendix A for further details.

- **Daily rewards**: to simplify its description, we split it into smaller bricks easily represented by Bernoulli random variables. The daily reward is nothing but the sum of the rewards generated by every customer in that day, which in turns are made up of the reward from the first and, possibly, the second item. The choice of a Bernoulli distribution is due to the nature of the problem: 0 means no purchase, while 1 means that the item has been purchased. As regards the first item, we identified four different Bernoulli distributions, whose means correspond to the conversion rates of each class. On the other hand, in the case of the second item the conversion rates change depending on the promo, therefore we needed to introduce *number of promos* $\times$ *number of customer classes* distributions, one for each combination of promo and customer class.
  Finally, the reward of a customer for a fixed price is

  $$r_1(c) * (\, m_1 + r_2(c, p) * m_2(p) \,) \tag{3}$$

  where $r_i \in \{0, 1\}$ is the reward of item i, for $i = 1, 2$; m_i is the margin obtained from the sale of item i, for $i = 1, 2$; $c \in \{0, 1, 2, 3\}$ stands for the customer class and $p \in \{0, 1, 2, 3\}$ stands for the promo assigned to the client. In the end, the daily reward is the sum of all the customer rewards collected during an entire day.

4 Online pricing problem

The main scope of this section is to describe the implementation of the solution of the pricing problem over the first product that the shop is selling.

4.1 Step 3

We started from the most basic scenario of an online pricing problem: almost all the parameters, such as the assignment of promos, the price and the conversion rates of the second item, and the number of customers that visit the shop each day are fixed, therefore known to the Learner. Moreover, the following assumptions have been made: the prices are the same for all the classes of customers and the conversion rates do not change. The goal is to learn the optimal price of the first item w.r.t. a range of alternatives given by the shop.

To reach the aforementioned goal we used two different approaches: Thompson Sampling and UCB (Upper Confidence Bound). Indeed, the problem can be seen as a MAB (Multi-Armed Bandit) problem where the different alternatives of price for the first item are the candidates (arms).

Concerning the number of clients per class that visit the shop each day, we considered a typical scenario where most of the customers are amateurs (20 customers for CL0, 40 for CL1, 10 for CL2, 30 for CL3). The prices for the first item range between 150€ and 250€, while the fixed price for the second item is 29.99€. The number of promos of a given type reserved for a given class, instead, varies according to the choice of the setting, as mentioned in the Introduction.

4.1.1 Environment

In order to simulate the behaviour of a customer, we designed a dedicated class, named `Environment_3` (which is common for the two approaches we used). With its `round` method, an Environment object extracts for each customer, the reward associated to the conversion rate over a price of the first item, a specific promo and a reward associated to the conversion rate over the discounted price of the second item, w.r.t. the previously selected promo.

The Environment knows the two sets of conversion rates (more details about their modelling in Appendix B), and uses them in the reward generation. The rewards are extracted from a Bernoulli distribution, as previously stated in the Variables Introduction.

4.1.2 Algorithm

The core idea is the following:

1. Build a set of customers that will visit the shop, w.r.t. the fixed number of customers per class;
2. Create an Environment object and a Learner object;
3. Simulate a day in the shop where the customers visit it;
4. Each day, for each customer, the price for the first item that is considered the best (called `pulled_arm`) is selected by the Learner with the `pull_arm` method and subsequently the Environment, with its `round` method, produces the rewards and a specific promo accordingly;
5. After having observed the rewards, the Learner updates its parameters with the `update` method and the shop's profit - relative to the current best price for the first item - is computed with the formula (3).

The difference between the two approaches we used is the way in which a Learner selects the price that appears to be the best in a specific situation and the way in which it updates its parameters.

Thompson Sampling approach
Here the Learner implements the Thompson Sampling Algorithm in order to learn the optimal price for the first item. The class we build for this Learner is called `TS_Learner_3`.

At each iteration, the best price for the first item is chosen considering all the profits associated with each possible price and then selecting the one that maximizes them. A single profit is given by the following formula:

$$profit = cr_1 * (m_1 + cr_2 * matching\_prob * m_2) \tag{4}$$

where cr_1 is an estimation of the conversion rate over the price of the first item obtained sampling a Beta distribution that is kept updated by the Learner, after being initialized as a Beta(1, 1), so a Uniform; cr_2 is the known conversion rate relative to the price of the second item; m_i is the margin relative to a given price for the i^{th} item, and $matching\_prob$ represents the probability that a given class receives a given promo (it is derived from the fixed matching between promos and customers' classes).

Once the best arm is selected, having observed the reward given by the Environment, the Learner updates the Beta distribution associated with *cr1*. In particular, the first parameter of the Beta is updated in case of $reward = 1$ while the second parameter is updated in case of $reward = 0$.

Upper Confidence Bound (UCB1) approach
In this second case, the Learner implements the UCB1 Algorithm in order to learn the optimal price for the first item. The class we designed for this Learner is called `UCB_Learner_3`.

This approach differs from the previous one because the choice of the best price for the first item is done by selecting the one that presents the highest upper confidence bound. In addition, at the beginning, all prices are selected once, ensuring the exploration of all the alternatives, at least one time.
The upper confidence bounds are computed, for each possible price for the first item, adding

a confidence term to the empirical mean of the profits related to the current price. In our scenario, the confidence term is

$$\sqrt{\frac{2\ \log(t)}{\max\{1,\ n\_pulled\_arm\}}} * profit \tag{5}$$

where t is the counter of customers that visited the shop, *n_pulled_arm* is the counter of how many times the selected price has been chosen so far and $profit$ is given by (3), the same formula already shown. Note that the usual confidence term of UCB1 is multiplied by the *profit*: this choice has been made in order to ensure that the empirical mean and confidence term have the same scale so to allow consistent exploration of arms. This is possible thanks to the fact that the reward lies in a compact set (0, *max_profit*).

At the end, the update of the parameters is done first increasing the counter of customers that visited the shop, then computing the *profit* and using it to extend the empirical mean of the *pulled_arm* and finally computing the new confidence term for all the arms.

4.1.3 Results

The following plots show our solution. In particular, we did one test per setting, and both tests share the same number of experiments (50) and the same number of alternatives for the price of the first item (65). Moreover, all Confidence Intervals have a confidence level of $\alpha = 0.05$.

The plots show the differences that we expected: UCB explores more and performs slightly worse than TS, that achieves better results, in terms of regret w.r.t. the Clairvoyant Algorithm (black dashed line in the plots). Those differences can be seen in the plots of expected reward and cumulative expected regret.

Setting 0

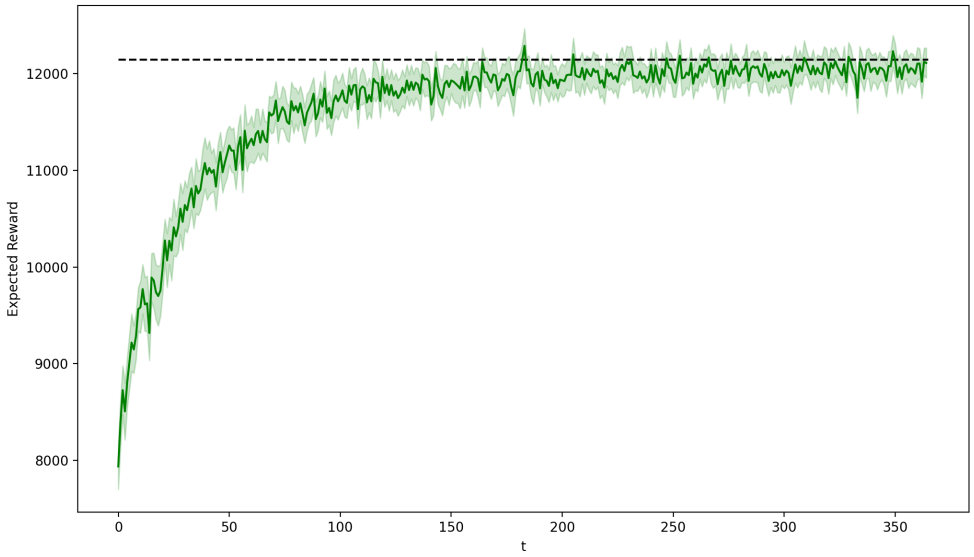


Figure 1: TS, Expected Reward, setting 0

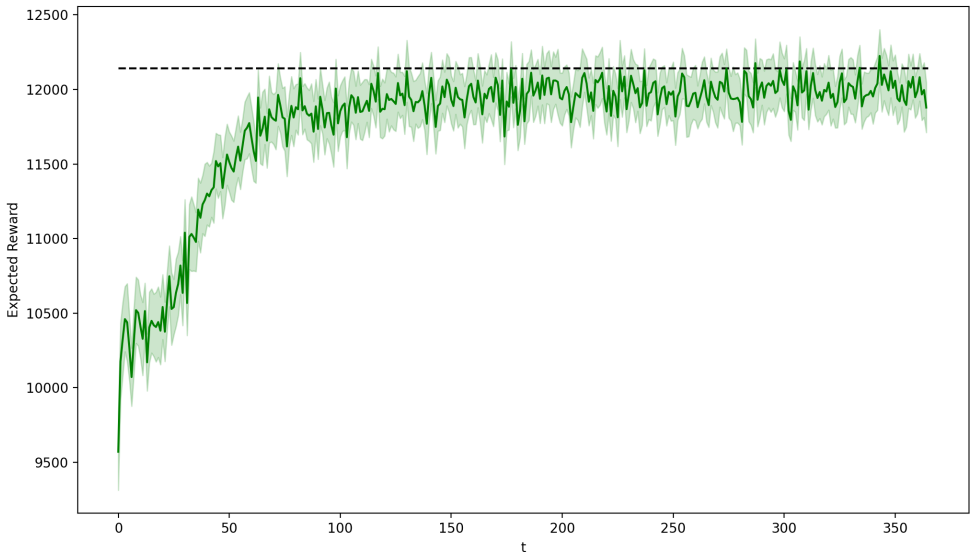


Figure 2: UCB, Expected Reward, setting 0

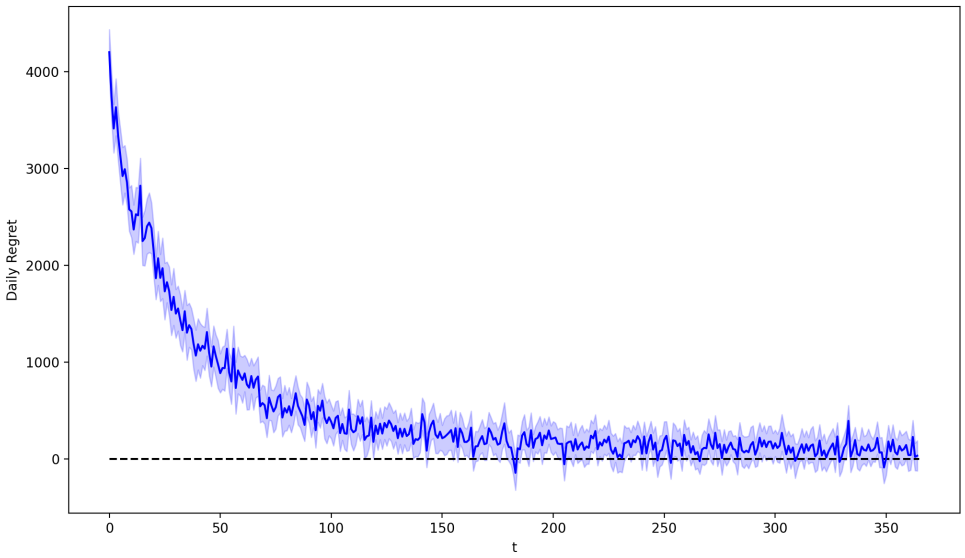


Figure 3: TS, Daily Regret, setting 0

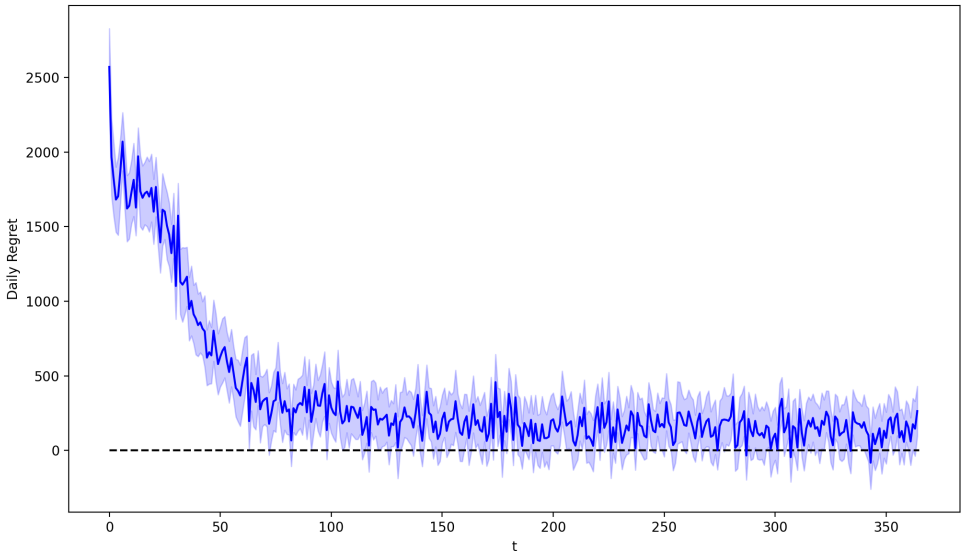


Figure 4: UCB, Daily Regret, setting 0

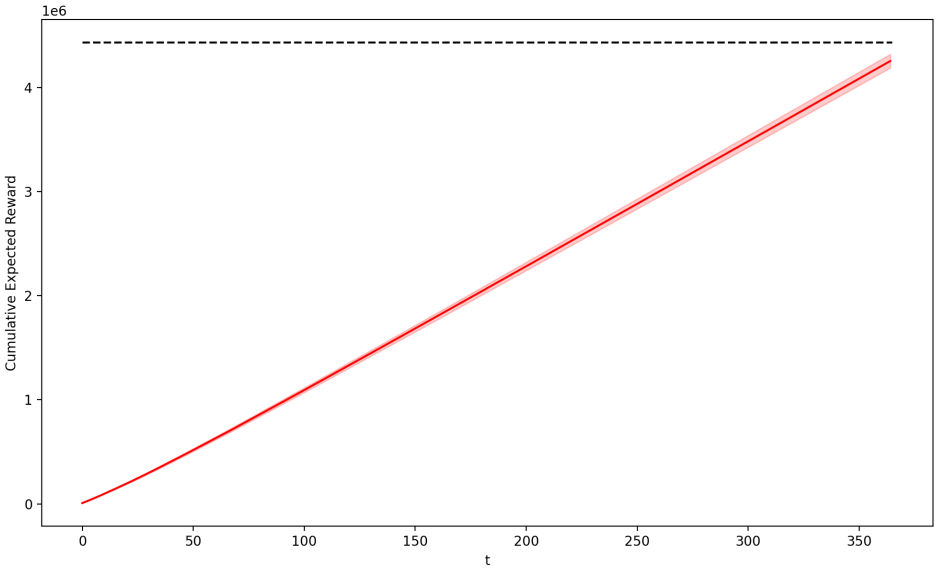


Figure 5: TS, Cumulative Expected Reward, setting 0

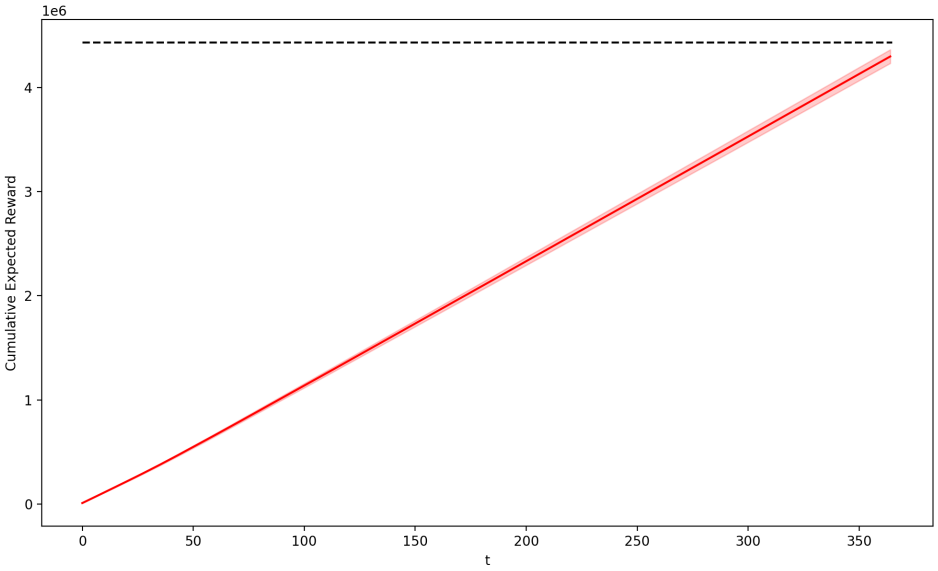


Figure 6: UCB, Cumulative Expected Reward, setting 0

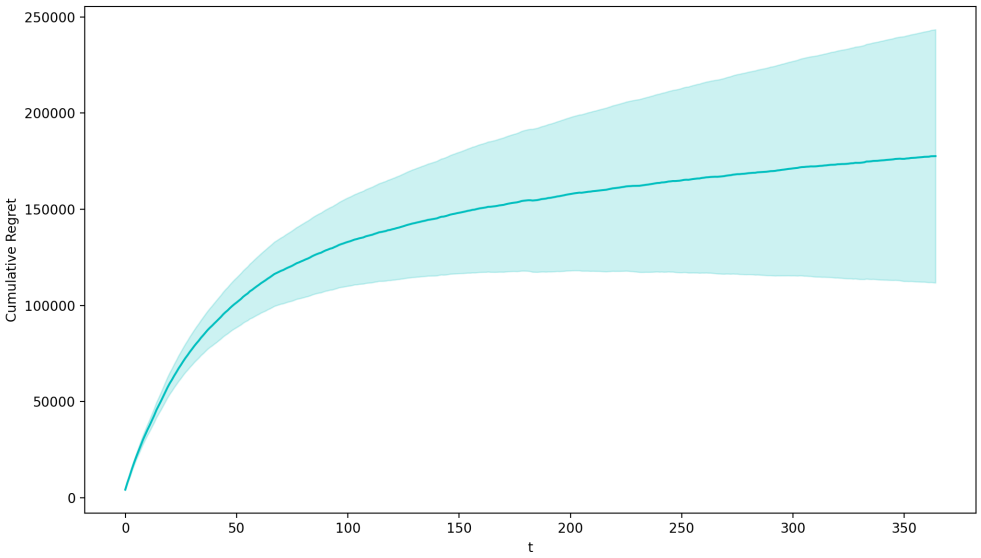


Figure 7: TS, Cumulative Expected Regret, setting 0

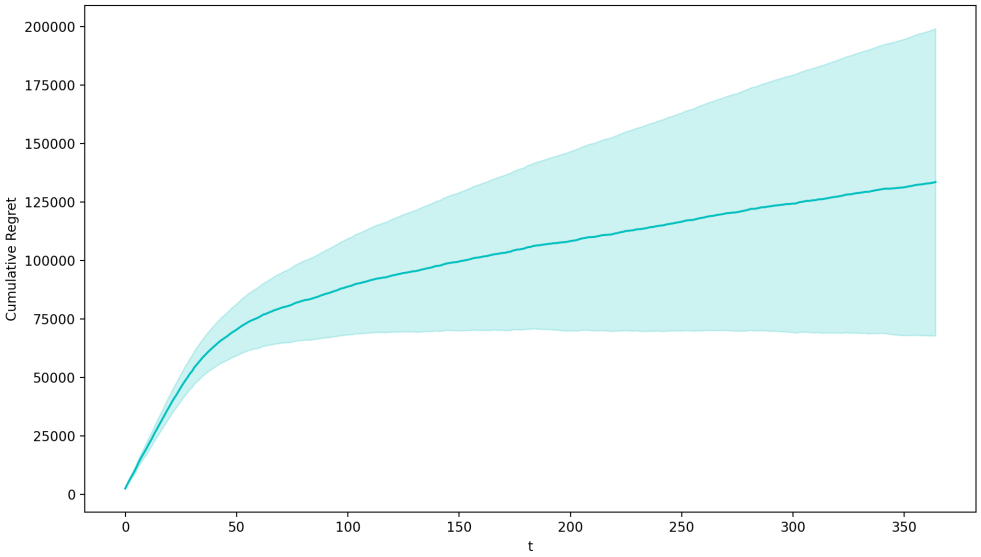


Figure 8: UCB, Cumulative Expected Regret, setting 0

Setting 1

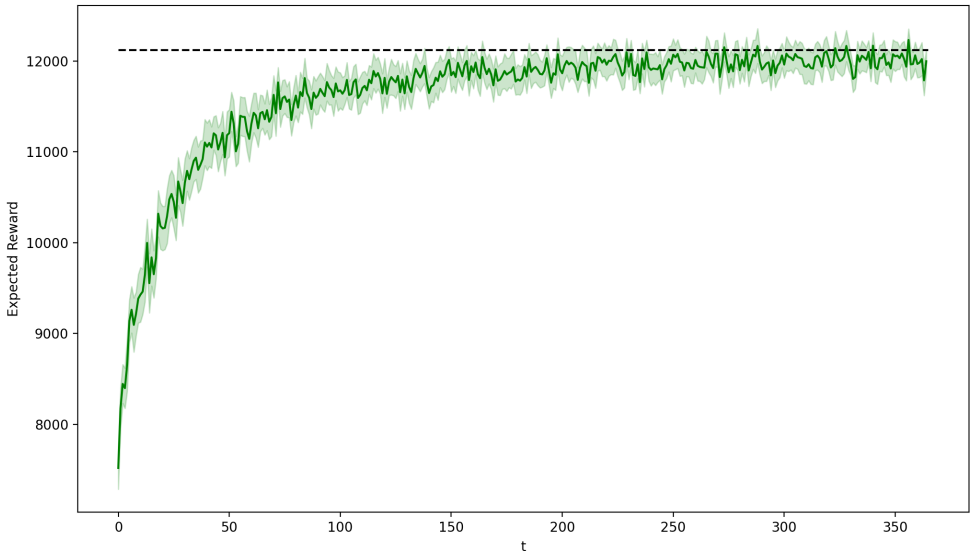


Figure 9: TS, Expected Reward, setting 1

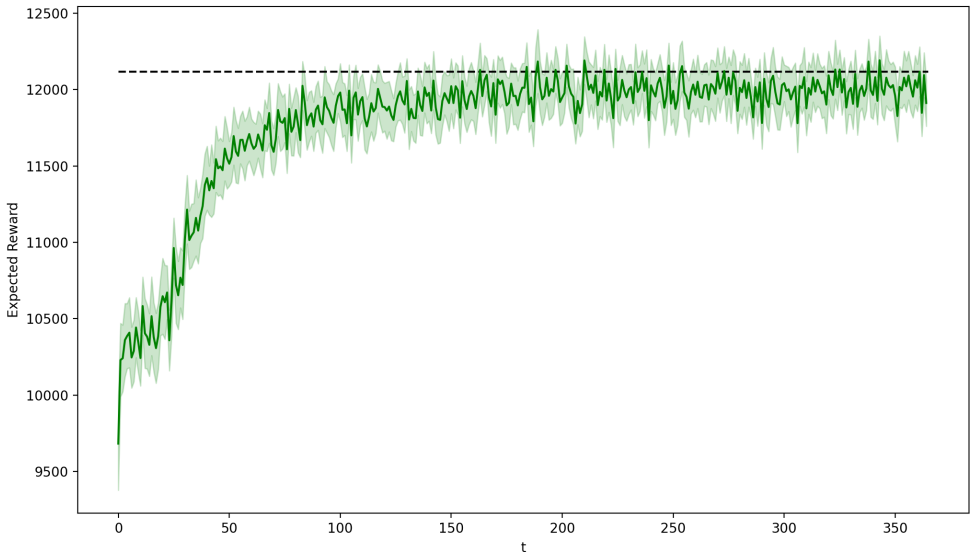


Figure 10: UCB, Expected Reward, setting 1

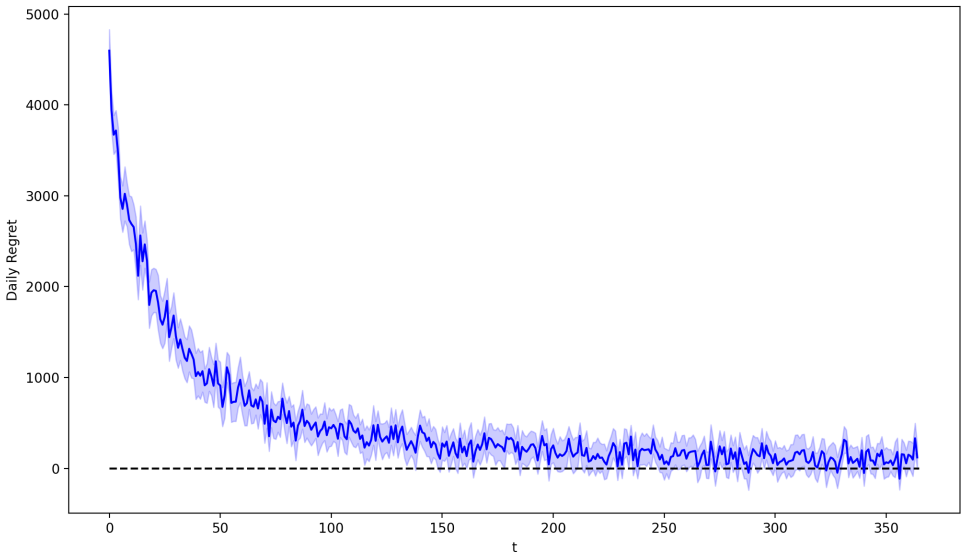


Figure 11: TS, Daily Regret, setting 1

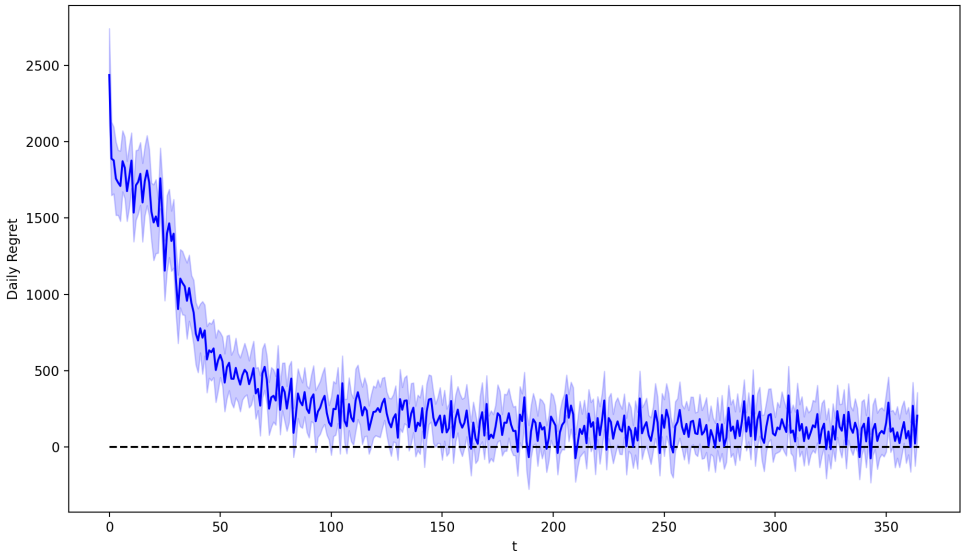


Figure 12: UCB, Daily Regret, setting 1

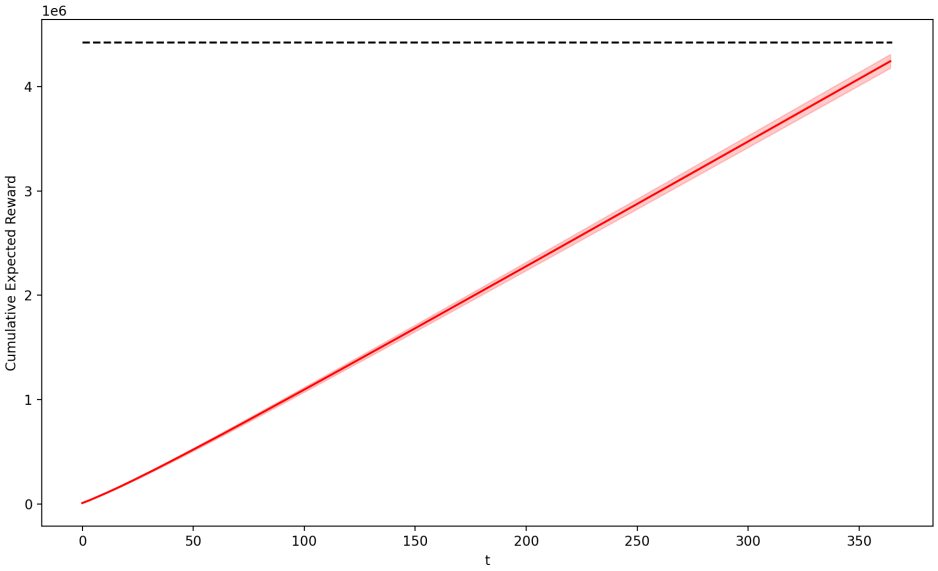


Figure 13: TS, Cumulative Expected Reward, setting 1

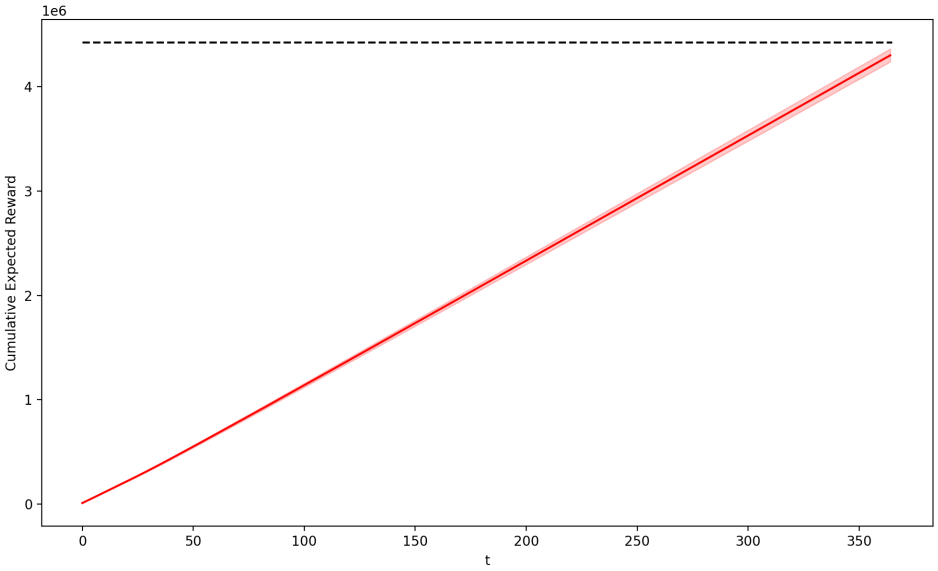


Figure 14: UCB, Cumulative Expected Reward, setting 1

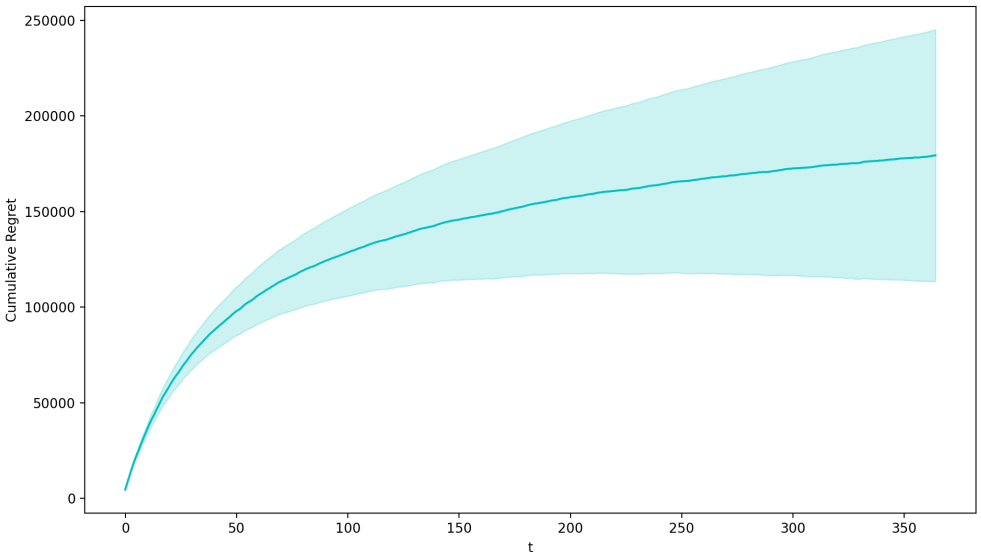


Figure 15: TS, Cumulative Expected Regret, setting 1

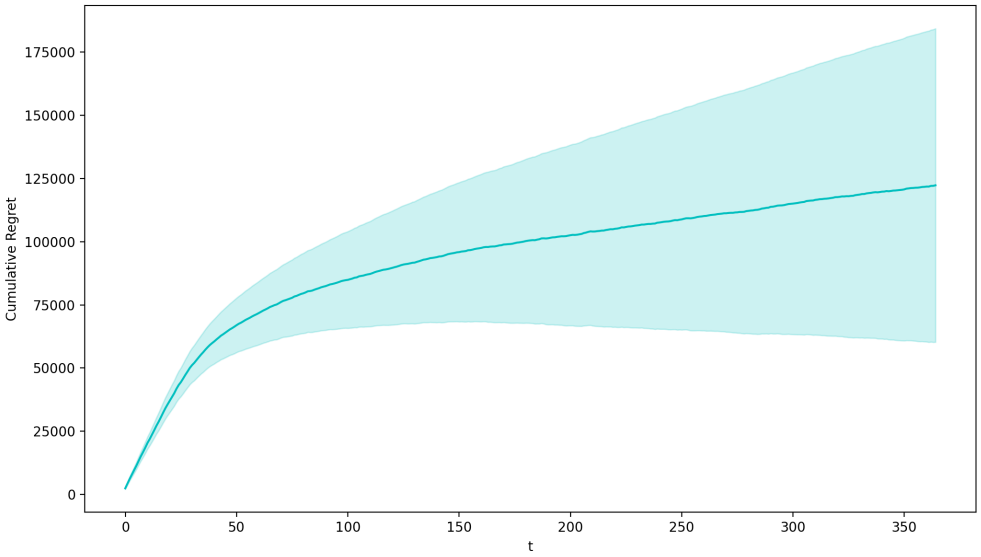


Figure 16: UCB, Cumulative Expected Regret, setting 1

4.2 Step 4

In this step we took into account also the fact that we didn't have any information on the conversion rates of both the first and the second product, as well as the daily number of customers that visit the shop.

4.2.1 Environment

The way in which the Environment provides the reward associated to each new customer visiting the shop remained unchanged.

The main point in this step, meaning the fact that we have no information on the number of arriving customers, has been addressed by designing truncated Normal distribution for each one of the classes in which they are divided (see Appendix A). Each distribution has its own mean, standard deviation and clipping values which determine its domain boundaries. The arrivals are simulated by extracting the number of customers per class from these functions, creating an array containing them and shuffling it. Then, during the experiment, each element of the array is fed to the learning algorithm, one at each time step.

4.2.2 Algorithm

Also in this step both UCB and Thompson Sampling based algorithms have been implemented to solve the problem. Both Learners have in common a snippet of code that is utilised in order to make them able to update their beliefs on the distribution of the customer classes at the end of each day. This snippet of code is implemented in the `compute_posterior` function, that is part of the `Learner_4` class. The way in which the two algorithms update their knowledge about the problem is different.

Thompson Sampling approach
The Thompson Sampling Learner keeps its information updated in the classical way, using the Bernoulli rewards provided by the Environment to adjust the Beta distributions associated to the different arms. It is worth mentioning that the Beta related to the conversion rate over the second item is updated only if the reward for the sale of the first item is 1.

What has been adapted in the implementation of this algorithm is the way in which the arm to be pulled is chosen. For each arm an expected profit is calculated. This expectation is due to the fact that we use the updated Beta distributions to estimate the conversion rates of both the first and second product for a specific class at a specific price. Another source of uncertainty for which we need to work in expectation, is the fact that this profit is calculated on the total number of customers that the Learner is expecting to see. The computation of the expected profit of each arm is implemented in the `profit` function that is part of the `TS_Learner_4` class.

Upper Confidence Bound (UCB1) approach
The UCB1 algorithm, instead, is still working as in Step 3 because we are still solving the pricing problem just in the first price. As a matter of fact, the empirical means associated to each arm contain the information about the purchasing of the second product, being computed on the total profit generated by a customer.

4.2.3 Results

The following plots show our solution. In particular, we did one test per setting, and both tests share the same number of experiments (50) and the same number of alternatives for the price of the first item (65). Moreover, all Confidence Intervals have a confidence level of $\alpha = 0.05$.

In analogy with the previous step, the plots show the differences that we expected: UCB explores more and performs slightly worse than TS, that achieves better results, if we look in the long run. On the other hand, the curves of the expected rewards behave more nervously, and remains a bit under the optimum. As a consequence, the regret stabilises at higher values, and the cumulative one is a bit steeper. A possible explanation of these phenomena is that there are additional variables to learn, the number of daily customers and the conversion rate of the second item.

Setting 0

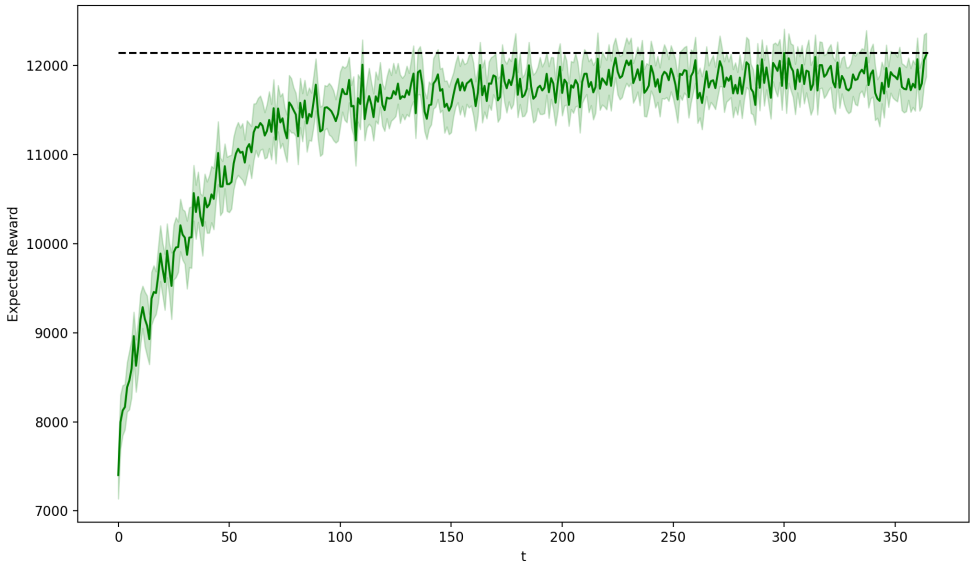


Figure 17: TS, Expected Reward, setting 0

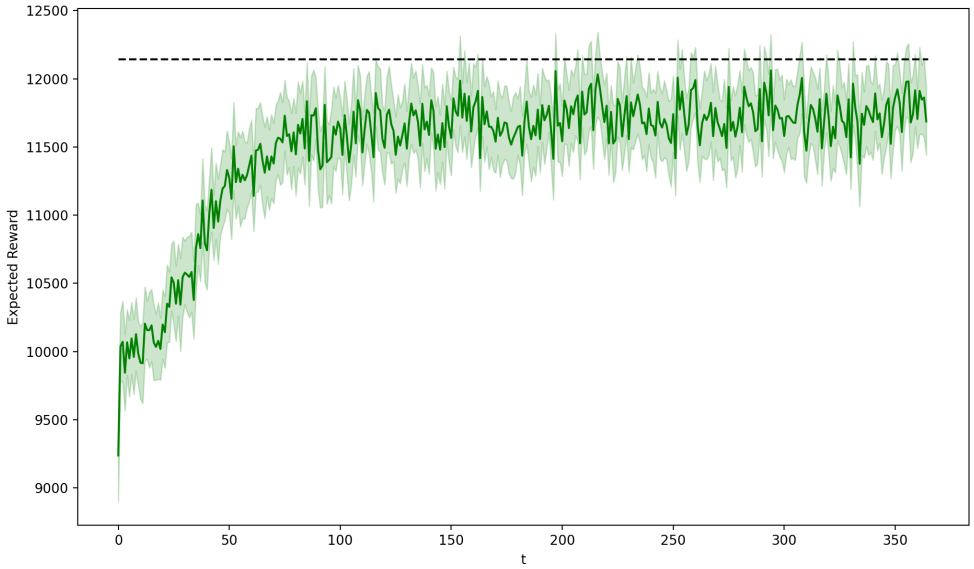


Figure 18: UCB, Expected Reward, setting 0

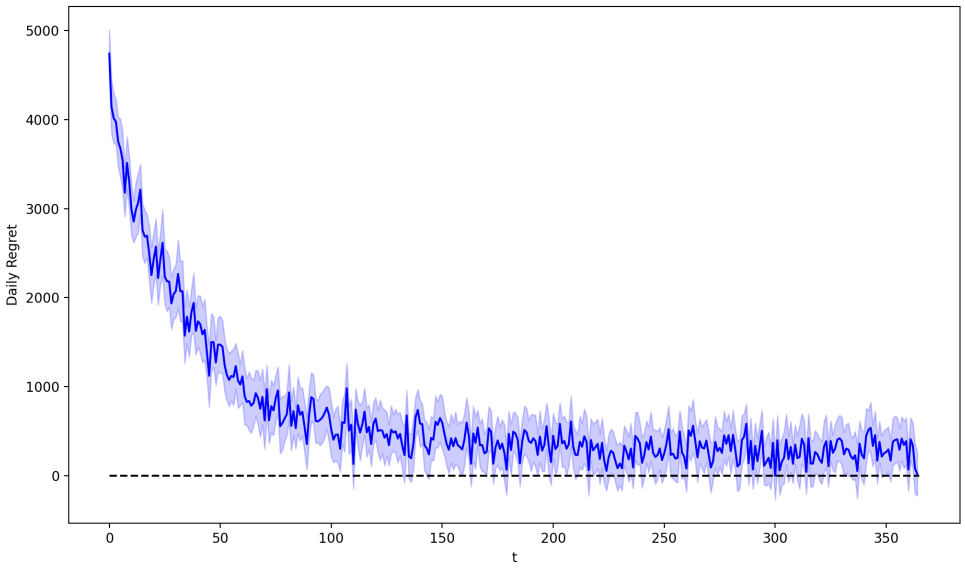


Figure 19: TS, Daily Regret, setting 0

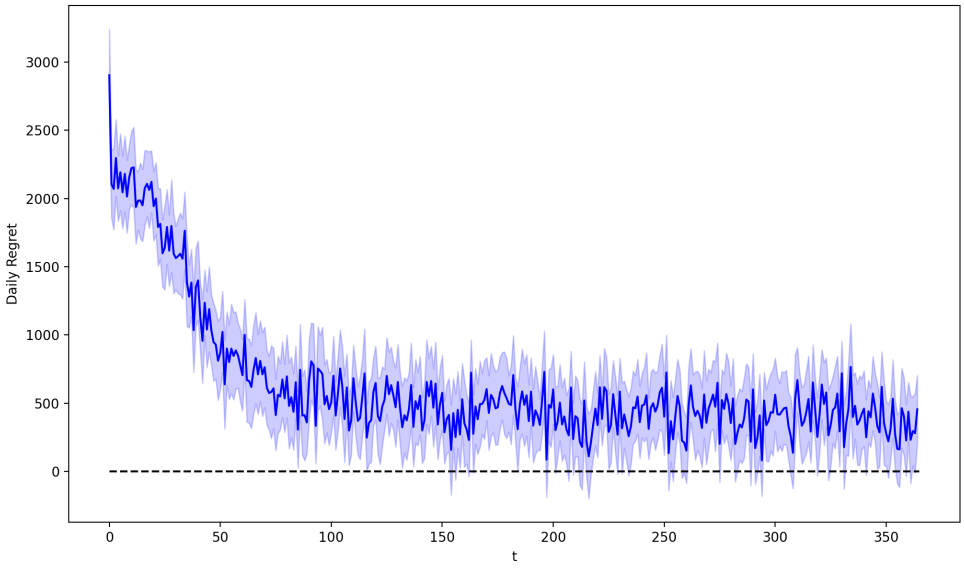


Figure 20: UCB, Daily Regret, setting 0

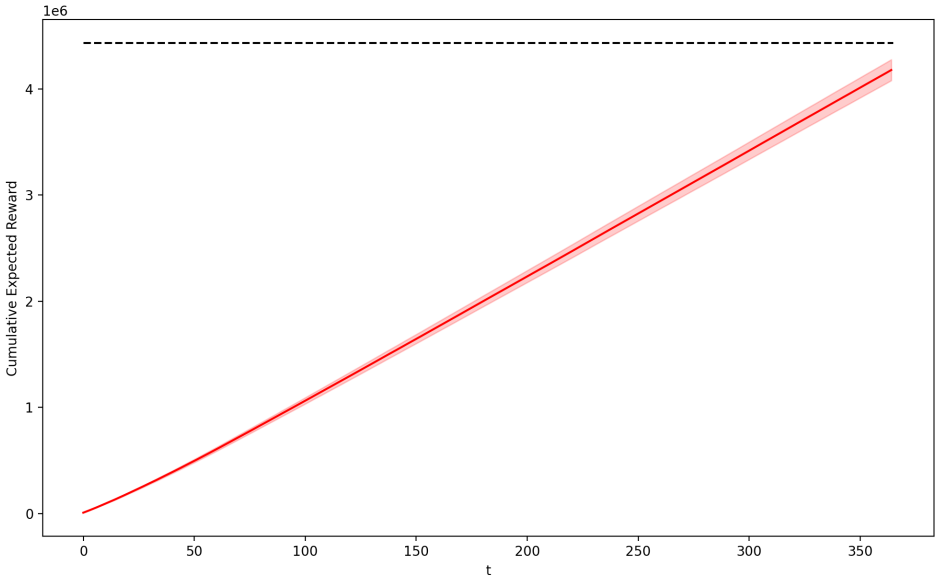


Figure 21: TS, Cumulative Expected Reward, setting 0

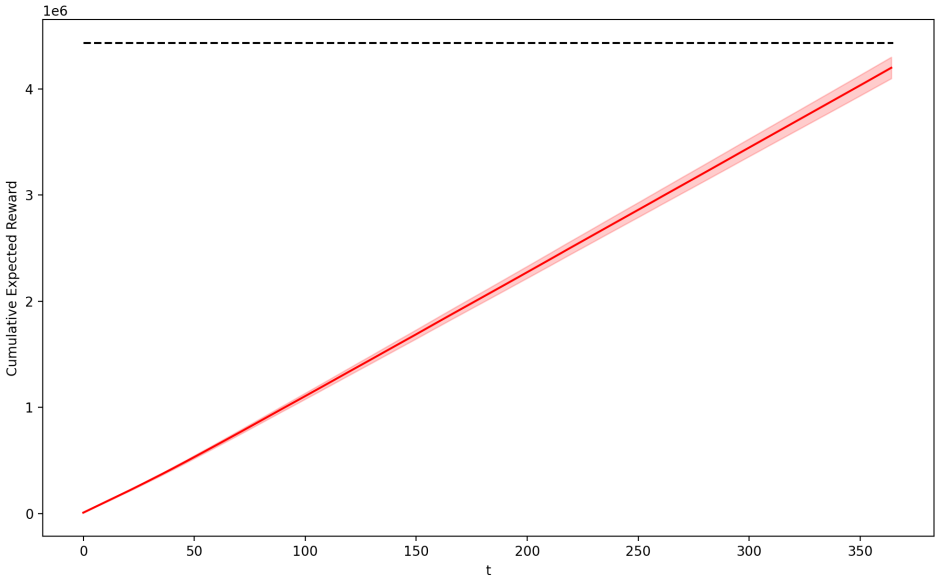


Figure 22: UCB, Cumulative Expected Reward, setting 0

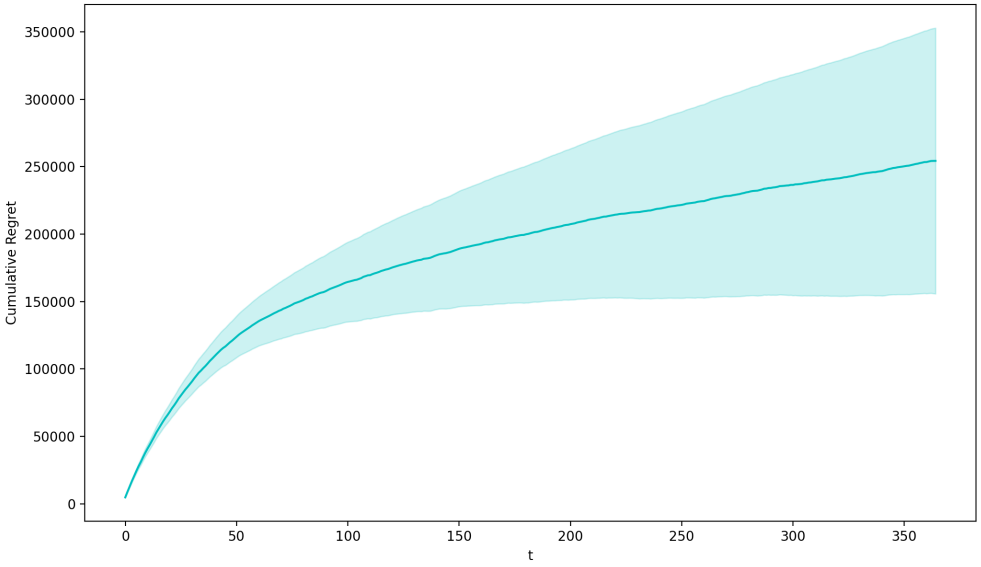


Figure 23: TS, Cumulative Regret, setting 0

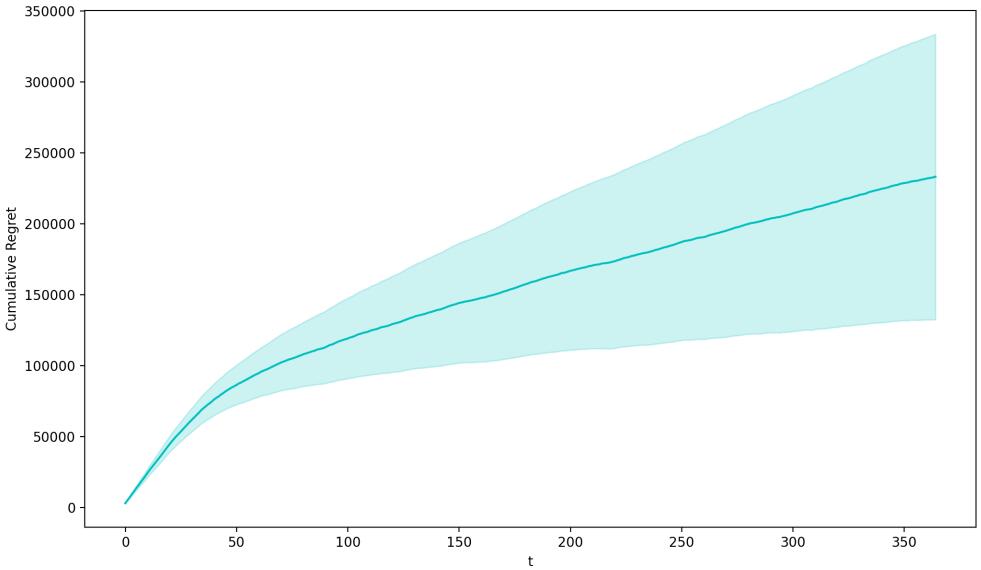


Figure 24: UCB, Cumulative Regret, setting 0

Setting 1

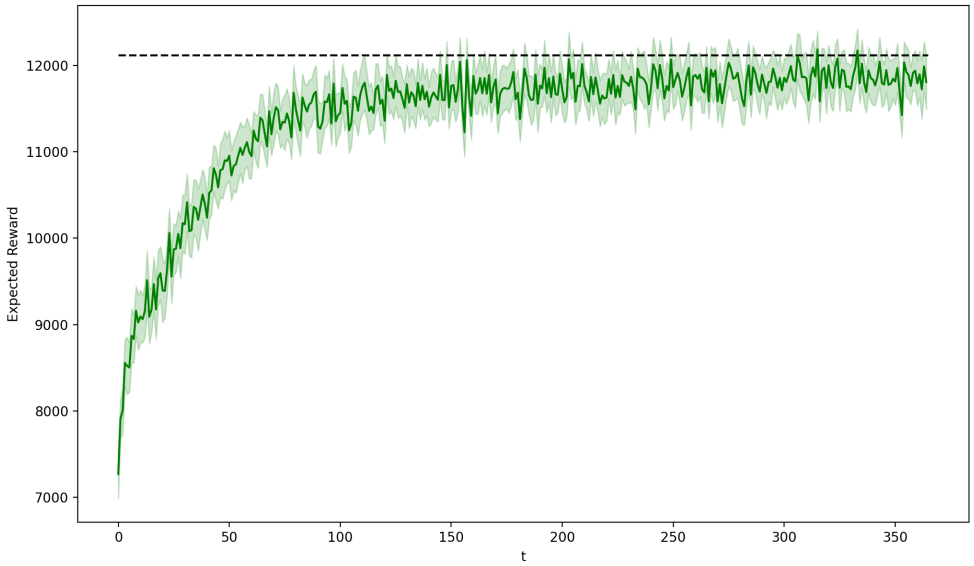


Figure 25: TS, Expected Reward, setting 1

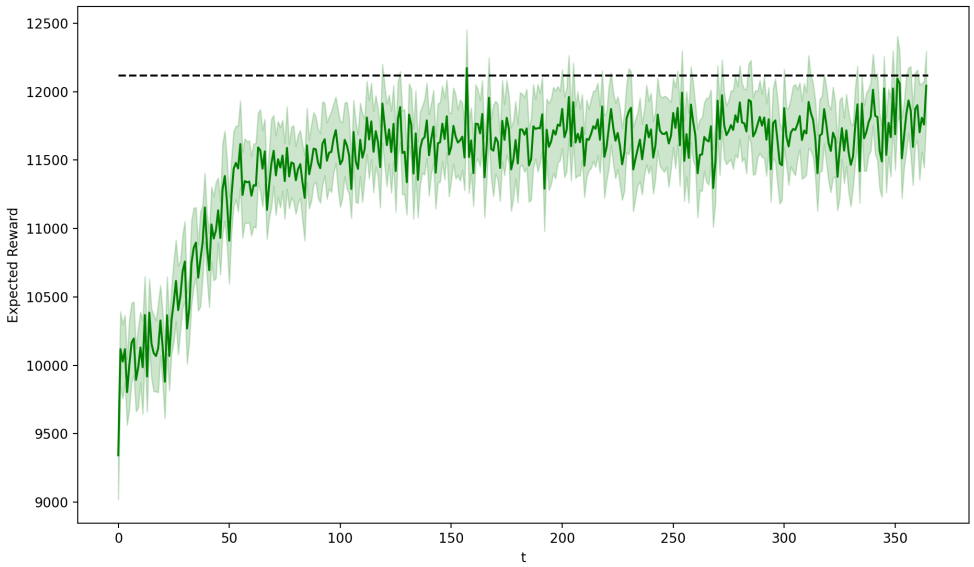


Figure 26: UCB, Expected Reward, setting 1

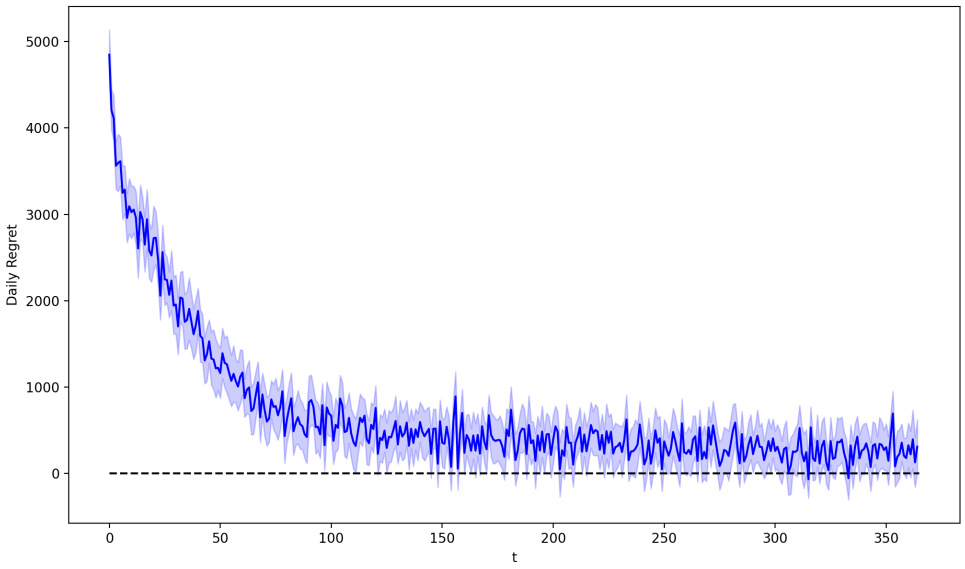


Figure 27: TS, Daily Regret, setting 1

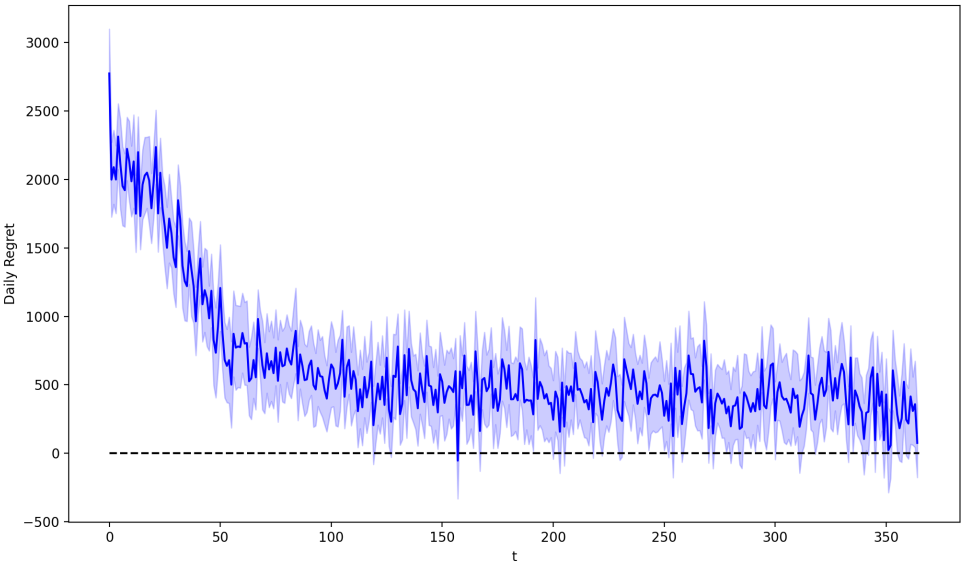


Figure 28: UCB, Daily Regret, setting 1

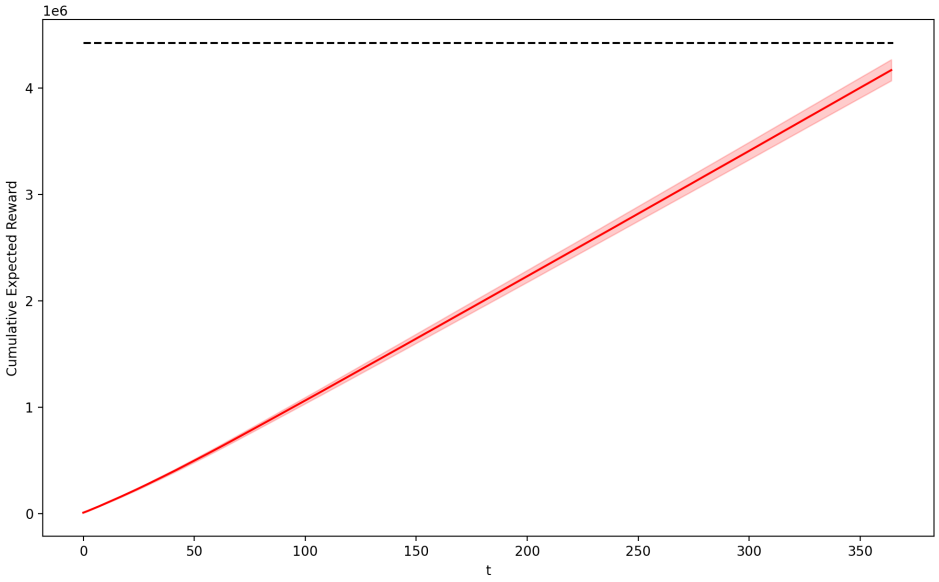


Figure 29: TS, Cumulative Expected Reward, setting 1

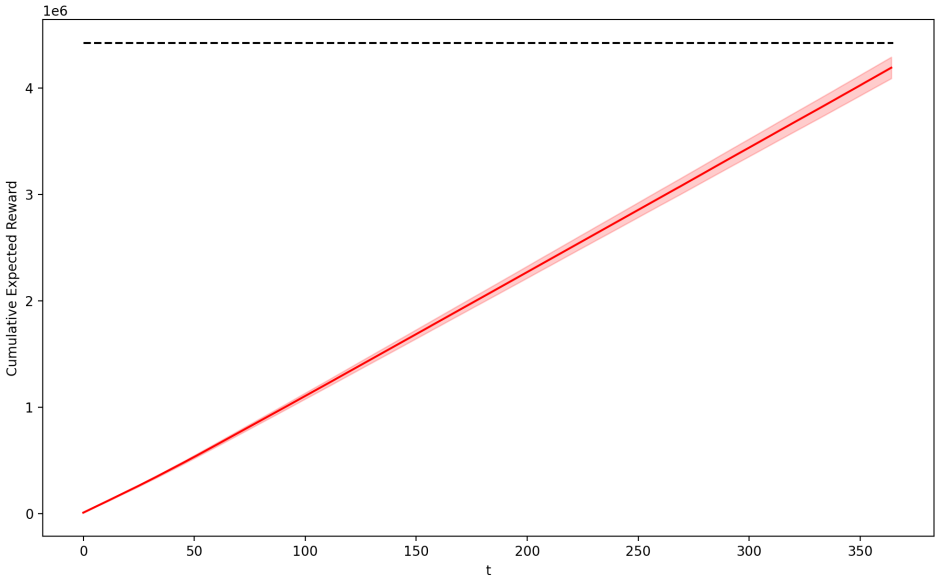


Figure 30: UCB, Cumulative Expected Reward, setting 1

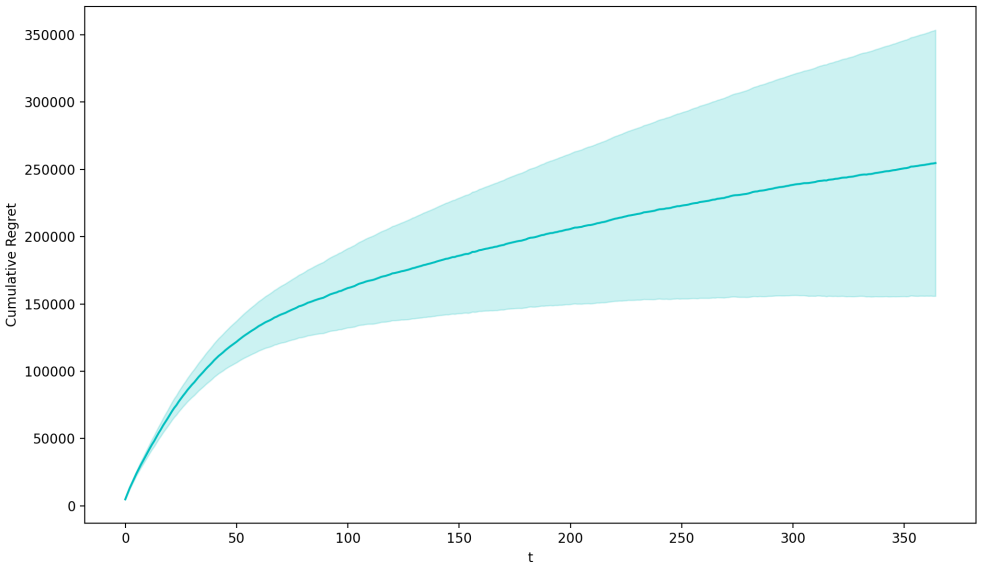


Figure 31: TS, Cumulative Regret, setting 1

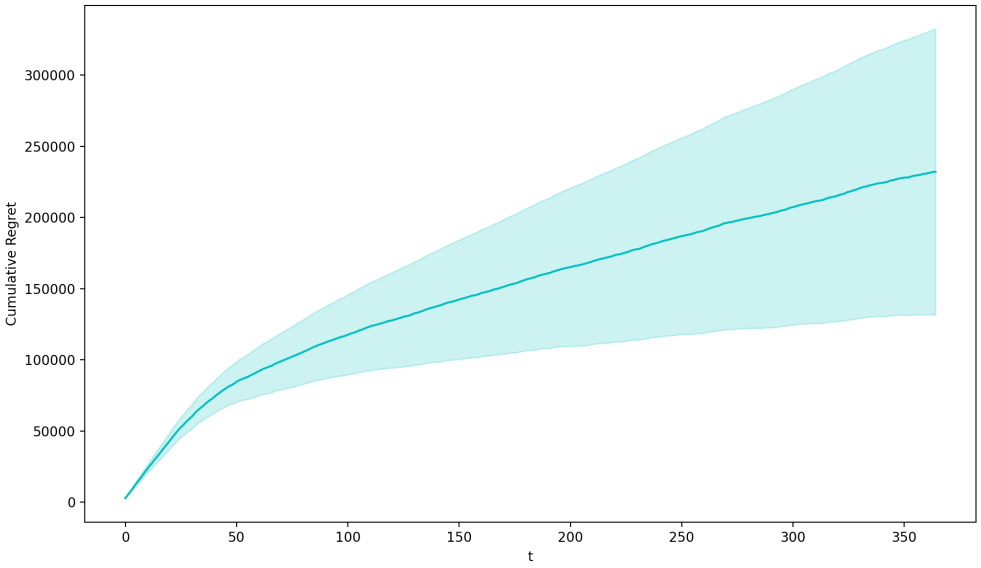


Figure 32: UCB, Cumulative Regret, setting 1

5 Online matching problem

5.1 Step 5

In this section the goal is complementary to the previous one, namely we have to find the optimal matching (promo assignment) in order to maximise the daily profit, having both prices fixed but with the associated conversion rates unknown, as well as all the other parameters.

5.1.1 Environment

The first things provided by the Environment are the variables representing the unknown quantities for the Learner. The two conversion rates cr_1 and cr_2 are scalar, in this case, since the prices are fixed. Then, in analogy with the second part of Step 4, when the number of customers per class is not deterministic, we needed to introduce the parameters required to build the four truncated Gaussian distributions: the means, standard deviations and the two domain boundaries vectors (Appendix A). Following the same reasoning, the `customer` function is introduced. Finally, at the heart of the Environment class there is the `round` function, which has the role of generating the binary rewards associated with each customer entering the shop.

As usual, to generate the reward linked with the selling of the second item, we need to know the promo proposed to the client. The promo is randomly selected among the four at hand, but it is worth mentioning that, in this step, the choice is based on the probability obtained from the progressive solution of the assignment problem.

5.1.2 Algorithm

From a practical point of view, to find the optimal matching (and so the probabilities to assign a certain promo code to a certain customer class) we applied the Hungarian Algorithm. In particular, we exploited the `linear_sum_assignment` function imported from the `optimize` package of the `scipy` library.

For what concerns the input matrix required by the algorithm, we built it in a way that every row corresponds to a customer and every column to a promo. The number of customers per class is obtained from the Learner (we refer to it as *expected customers*), which starting from an initial non-informative *a priori* assumption, updates its knowledge thanks to the observed entrances at the end of each day. Consequently, the number of promos (which sums up to the number of *expected customers*) is computed to follow the proportions initially set by the business unit of the shop, so we start by calculating the number of effective promos, approximated to the unit, and then, the remaining promo codes are set to *no promo*. The values contained inside every matrix element are the customer expected rewards, as previously said, and they are computed according to the usual formula (3). They are *expected* since the quantities involved in its computation are not all known, so we need to exploit the corresponding values learnt along the way. In particular, the conversion rates are modelled by the well-known Beta-Bernoulli model; on the other hand, the margins are known.

Since the underlying idea of the Hungarian Algorithm is to find the matching minimising the costs, we had the care of converting the input matrix into a cost matrix, by subtracting to each element the maximum element of the matrix, and then reverting the sign.
The outputs of this algorithm are the `matching_mask` and the `matching_value`. The former is an $N \times N$ matrix of 0-1 values where 1 indicates that the i^{th} client is associated

with the j^{th} promo, and it is used both in the Learner, for the update of the Betas, and in the Environment. In fact, to extract the promo to be associated with the customer who has just entered the shop, there are needed the number of *expected customers* and the following conditional probability values $p_{ij} = \mathbb{P}(\text{choose promo } j \,|\, \text{customer of class } i)$, which are stored in the `matching_prob` matrix, computed from the `matching_mask`. The `matching_value`, is nothing more than the input matrix multiplied by the `matching_mask` (so only the chosen matches are preserved), and the sum of all this values is the optimal expected daily reward (see figure 41). We recall that this algorithm is run once per day, at its beginning, and the outputs are valid for the entire day.

5.1.3 Results

A quick note, all the plots are accompanied by Confidence Intervals computed thanks to the experiment repetitions, and by the optimal level (dashed line in the reward plots) that can be reached in a utopian way with the Clairvoyant Algorithm. In this case we used 200 experiments, and the confidence level is $\alpha = 0.05$.

The following plots may appear not to be satisfying like the previous ones, but we are not trying to learn the optimal price here - it is fixed - but the optimal matching. Thus, the room for improvement is limited, compared with the previous steps for we can only maneuver the promo assignments. To really appreciate the job done here, one can look at the expected matching values, which show quite clean and not noisy lines, and that after an initial settling period, progressively grow (and learn) as expected.

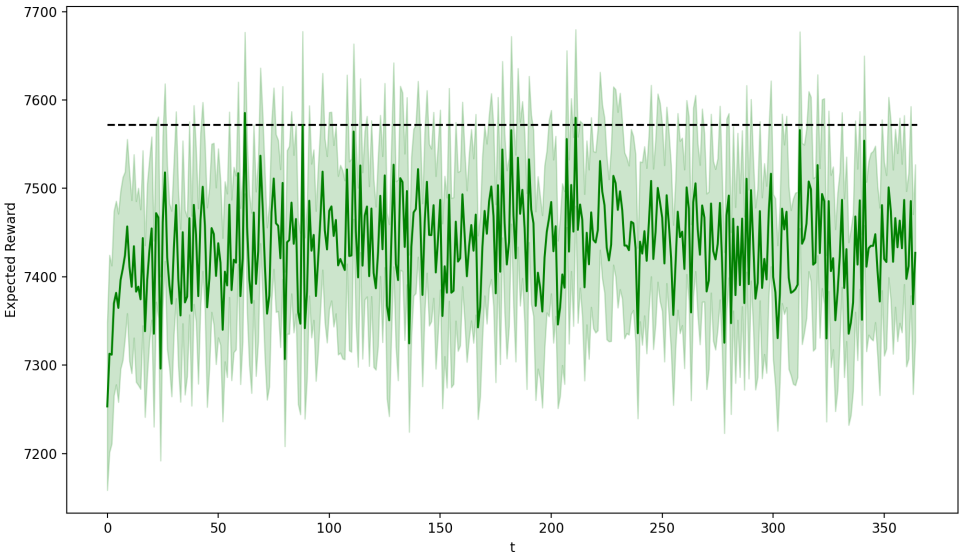


Figure 33: Expected Reward, setting 0, Online Matching problem

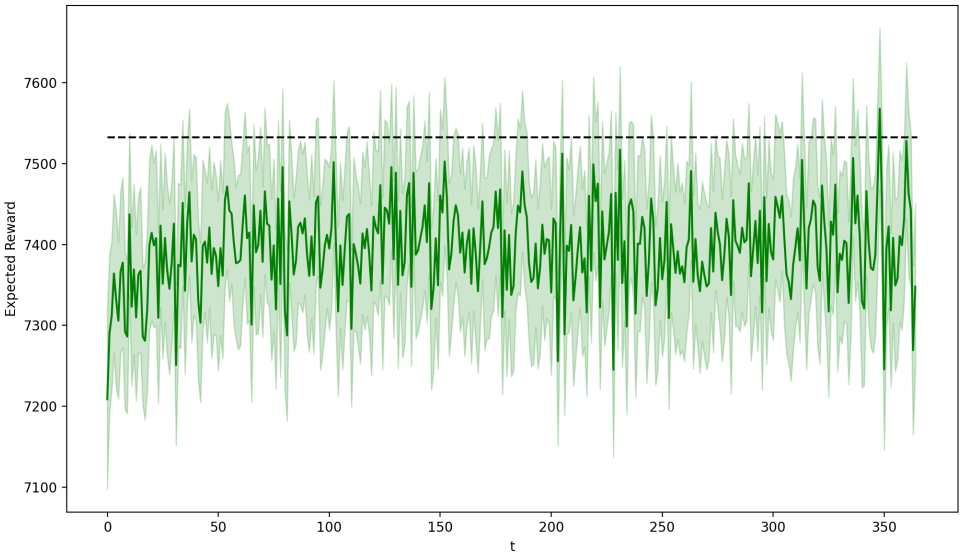


Figure 34: Expected Reward, setting 1, Online Matching problem

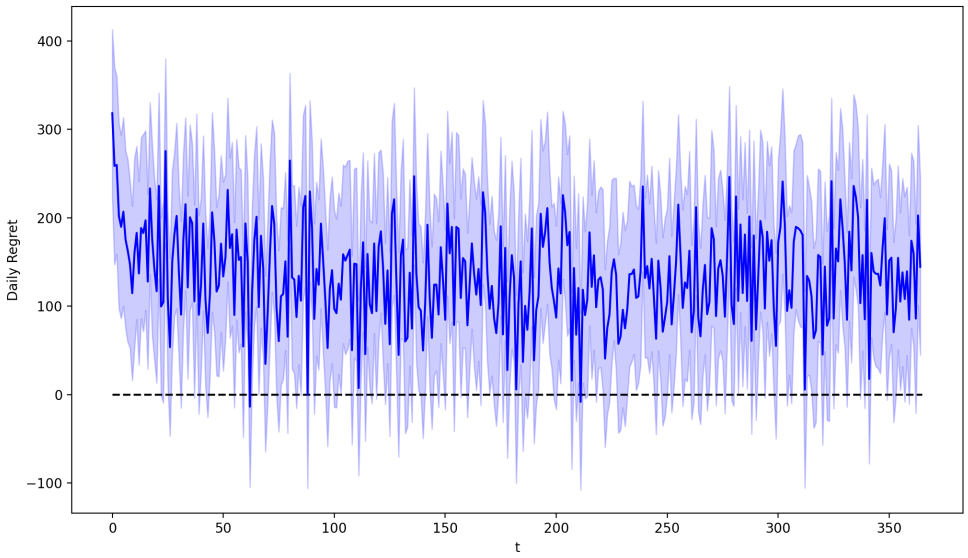


Figure 35: Daily Regret, setting 0, Online Matching problem

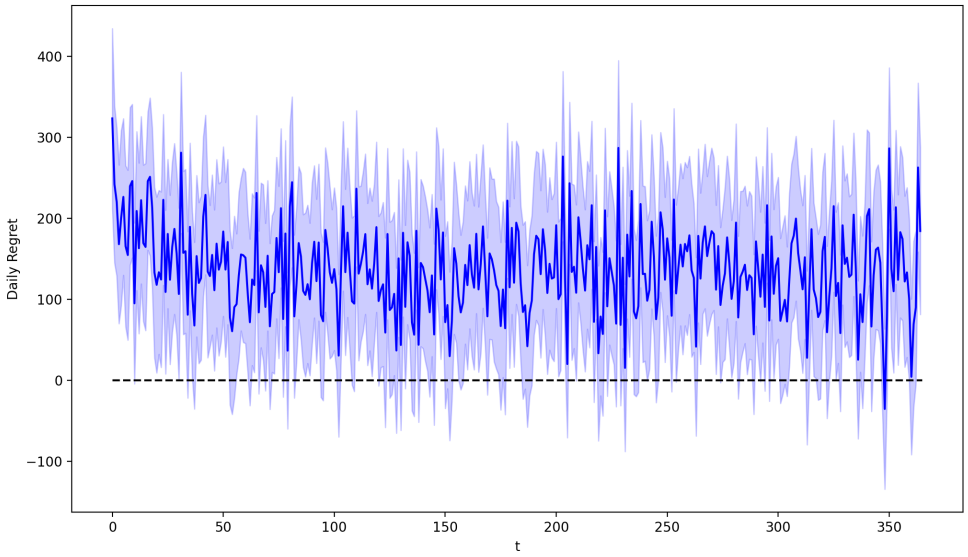


Figure 36: Daily Regret, setting 1, Online Matching problem

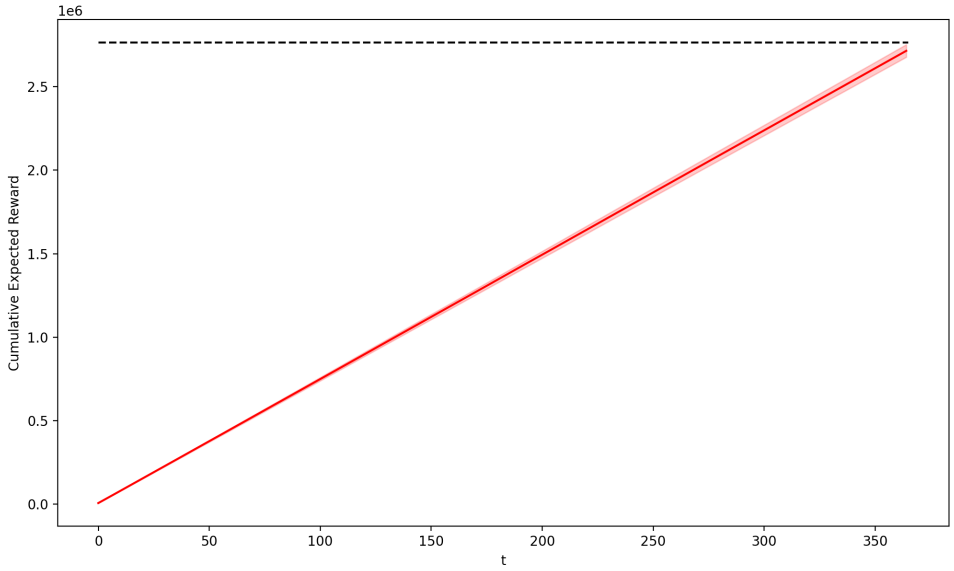


Figure 37: Cumulative Expected Reward, setting 0, Online Matching problem

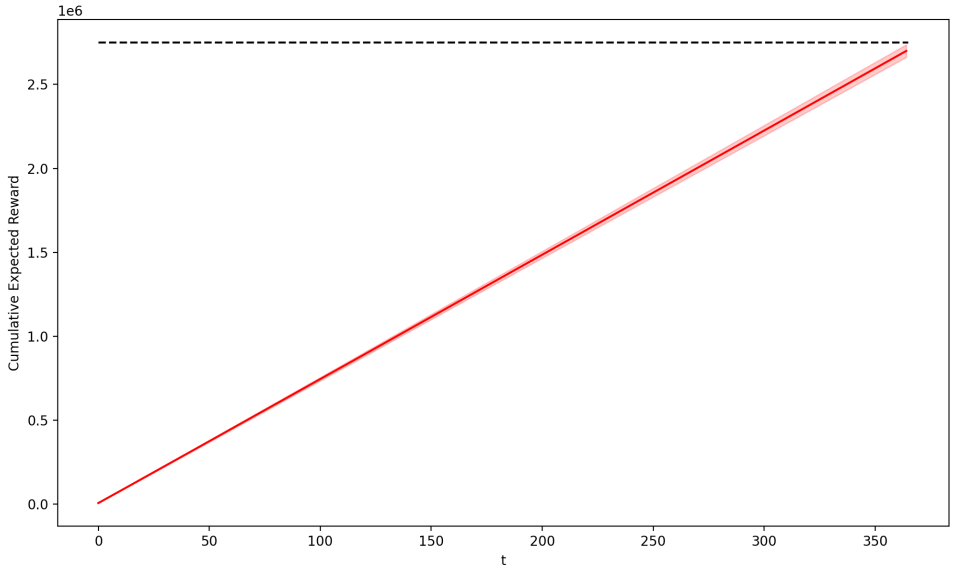


Figure 38: Cumulative Expected Reward, setting 1, Online Matching problem

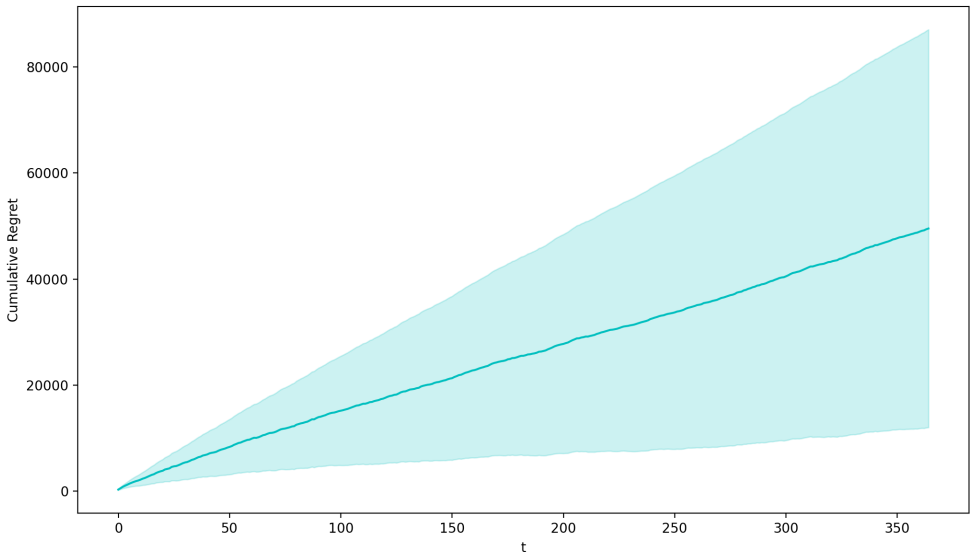


Figure 39: Cumulative Expected Regret, setting 0, Online Matching problem

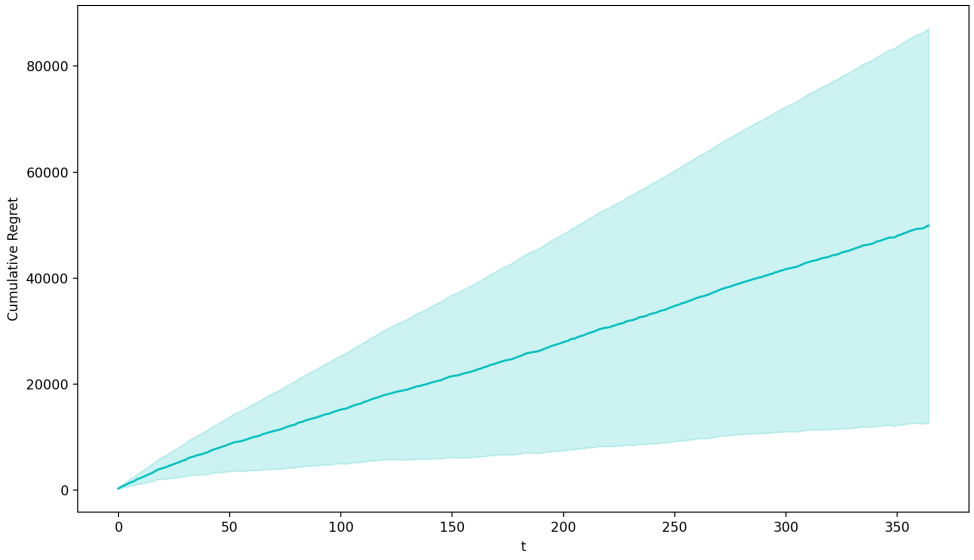


Figure 40: Cumulative Expected Regret, setting 1, Online Matching problem

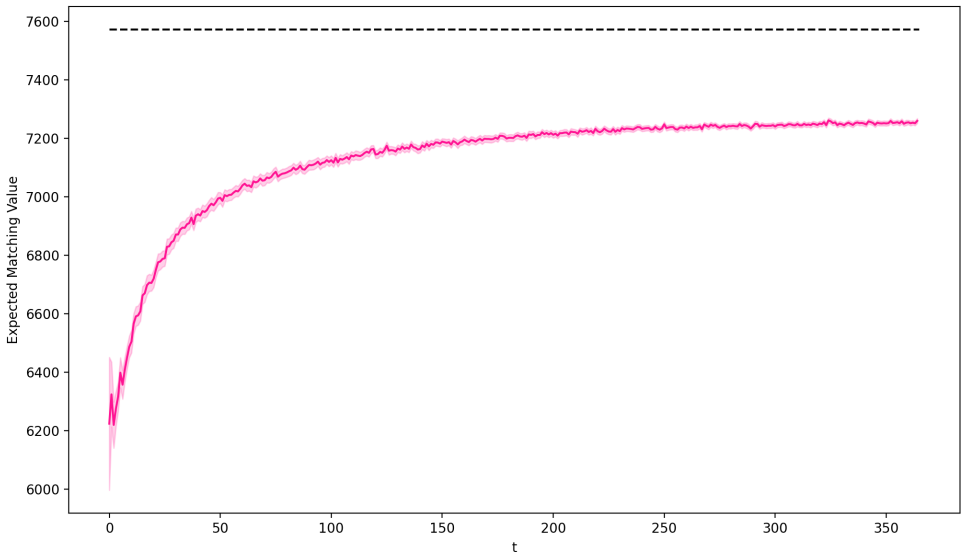


Figure 41: Expected Matching Value, setting 0, Online Matching problem

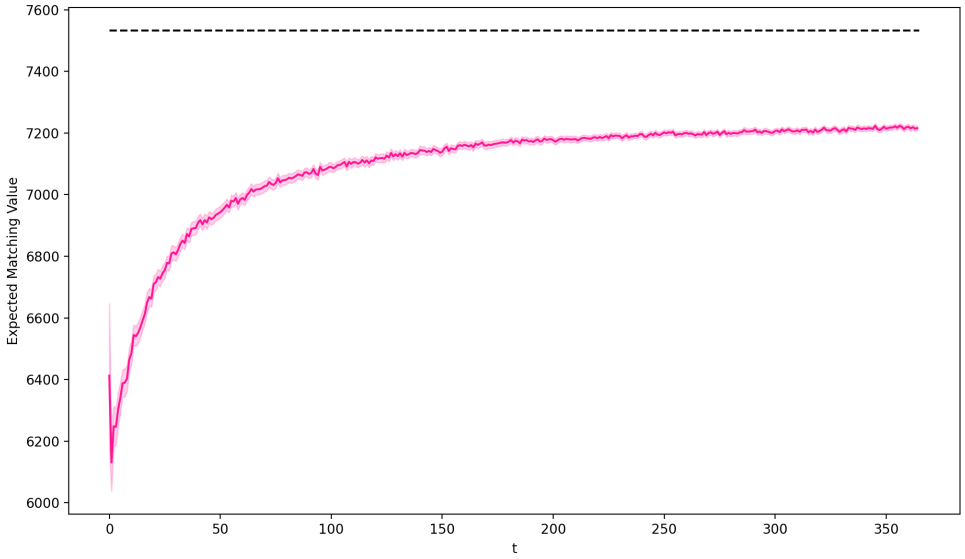


Figure 42: Expected Matching Value, setting 1, Online Matching problem

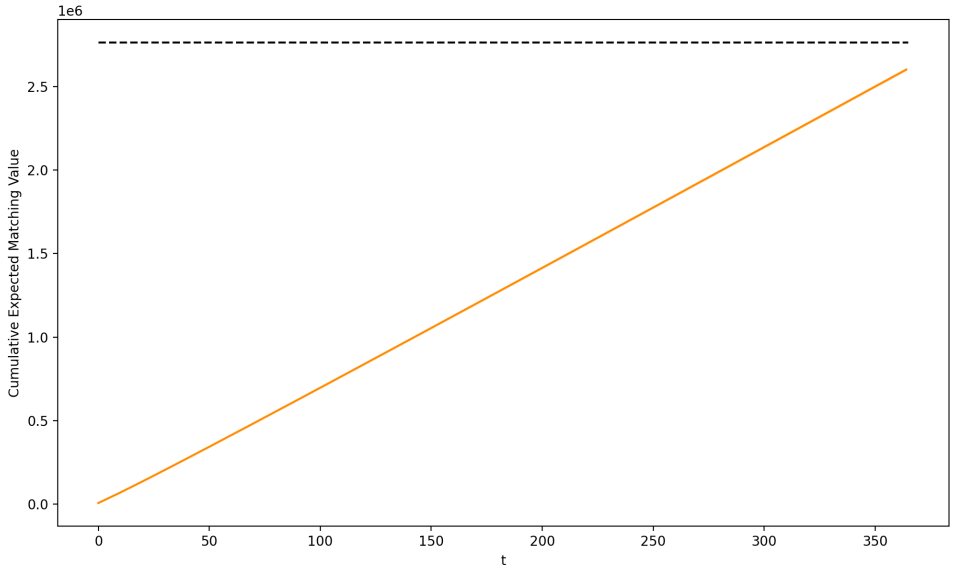


Figure 43: Cumulative Expected Matching Value, setting 0, Online Matching problem

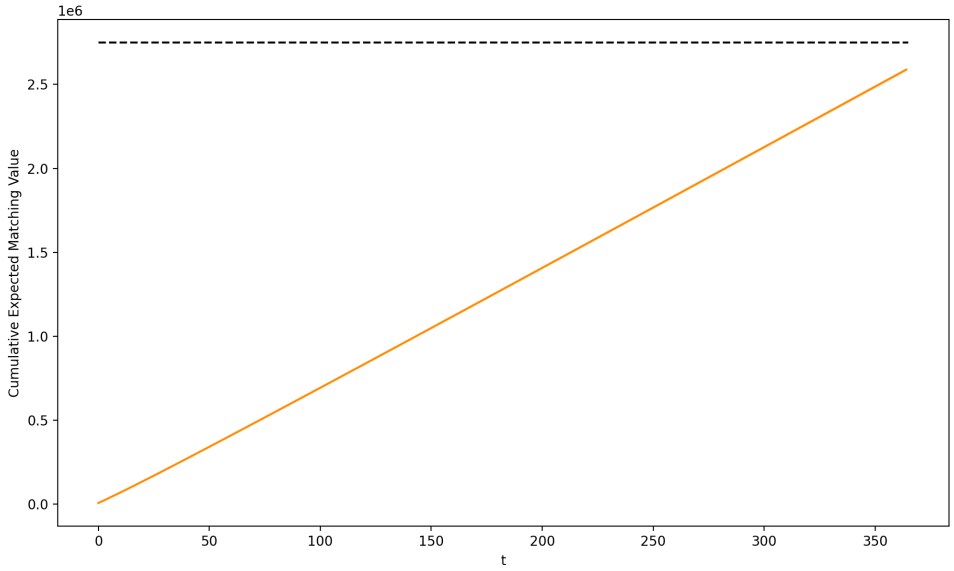


Figure 44: Cumulative Expected Matching Value, setting 1, Online Matching problem

6 Full online problem

In this section the implementation of the last steps of our analysis will be discussed, whose aim is to address the full problem, meaning the pricing problem over the two products combined with the search of the optimal assignment of promo codes to the customers visiting the shop.
This has been done in stationary conditions, as well as in non-stationary contexts. In particular, the proposed solution comprehends three different approaches: a stationary Thompson Sampling, a Sliding-Window Thompson Sampling and a Change-Detection UCB. All of them share the same unknown parameters, which need to be learnt:

- The number of customers, per class, that visit the shop each day;
- The conversion rates over the prices of both items;
- The assignment of promos to the customers.

6.1 Step 6

We started from the stationary case of the problem, where the conversion rates do not change during the year.
In our scenario, we kept the prices of the first item between 150€ and 250€ , while we opted for the interval between 25€ and 35€ for the prices - to be discounted - of the second item.

6.1.1 Environment

The Environment class we propose is almost the same w.r.t. the one we used in Step 5: it is able to generate the set of customers that visit the store each day and, most importantly, it generates the rewards and the promo. The difference is that, each time, it uses the best price for the first item and the best price for the second item in order to generate rewards and promo.

6.1.2 Algorithm

The core idea of the algorithm is very similar to the one already shown in Step 3:

1. Create an Environment object and a Learner object;
2. Each day, simulate the arrival of customers with the Environment;
3. For each customer that visits the shop, the Learner detects the best assignment of promos to the customers and the couple of prices realizing it;
4. At the end of the day, after observing the rewards generated by the Environment, the Learner updates its parameters.

The Learner (class `Learner_6`) uses a Thompson Sampling approach very similar to the one used in Step 3 and Step 4, but here the choice of the best prices is based on the profitability of the assignment of promos to the customers. In particular, the `pull_arm` method of the Learner computes the assignment of promos to the customers (called *matching* in the following) for each couple ($price_1$, $price_2$) using the Hungarian Algorithm and then selects the one that ensures the highest profit and also the prices that allow to derive it. The

current best *matching* is useful to compute the `matching_prob` matrix, fundamental to extract the promo in the `round` method of the Environment.

Finally, in the update phase, the Learner updates its estimation over the number of customers that will enter the shop, and then it updates also the Beta distributions, that are useful to simulate the unknown conversion rates.

6.1.3 Results

The proposed solution has been obtained with 15 experiments, 9 alternative prices for the first item and 5 different prices for the second one. The choice of using much fewer arms than the previous steps is driven by computational reasons, and it reflects in the plots of daily regret and reward. Indeed, the learning is very fast and the curve reaches almost immediately the optimal performance.

Given the small amount of experiments and the large set of variables to learn, the plots are still very nervous. However, the cumulative regret tends to flatten out, after a short increasing phase.

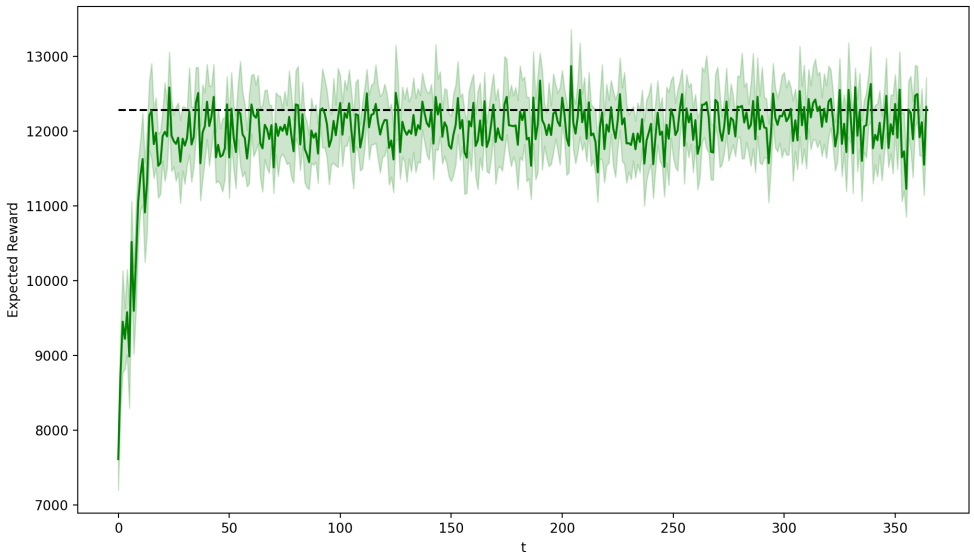


Figure 45: Expected Reward, setting 0

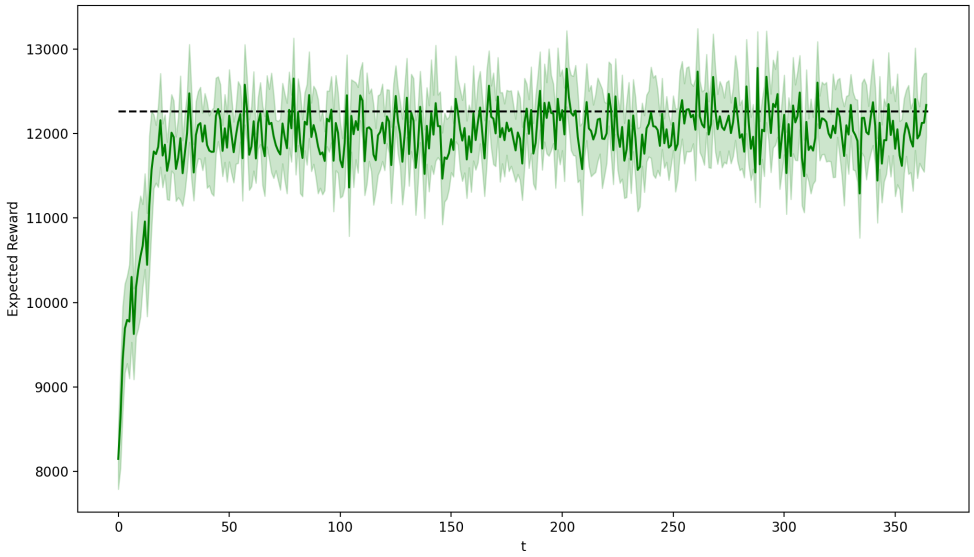


Figure 46: Expected Reward, setting 1

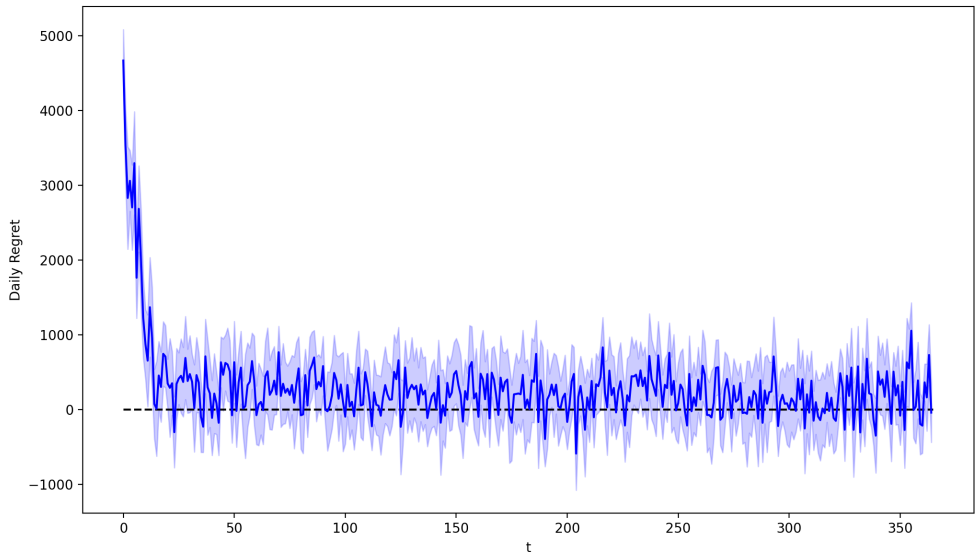


Figure 47: Daily Regret, setting 0

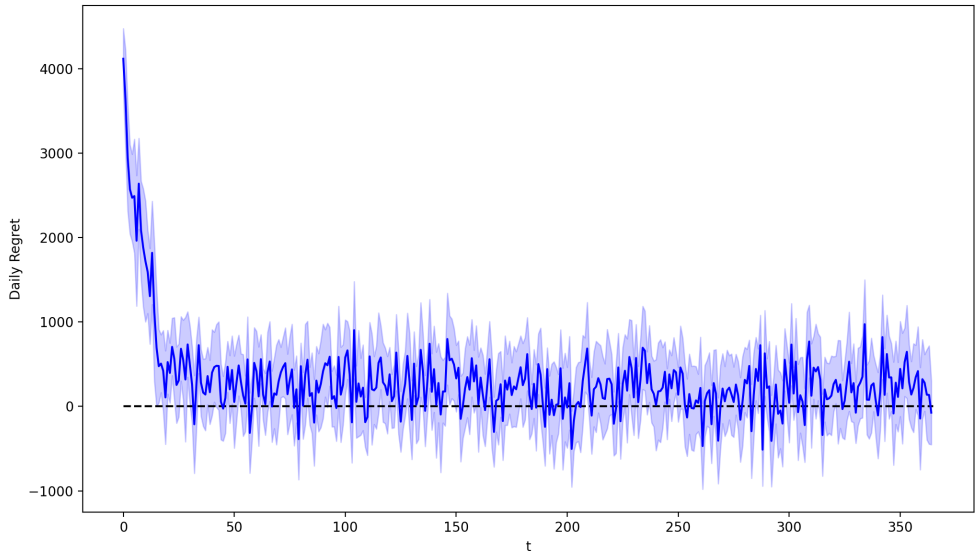


Figure 48: Daily Regret, setting 1

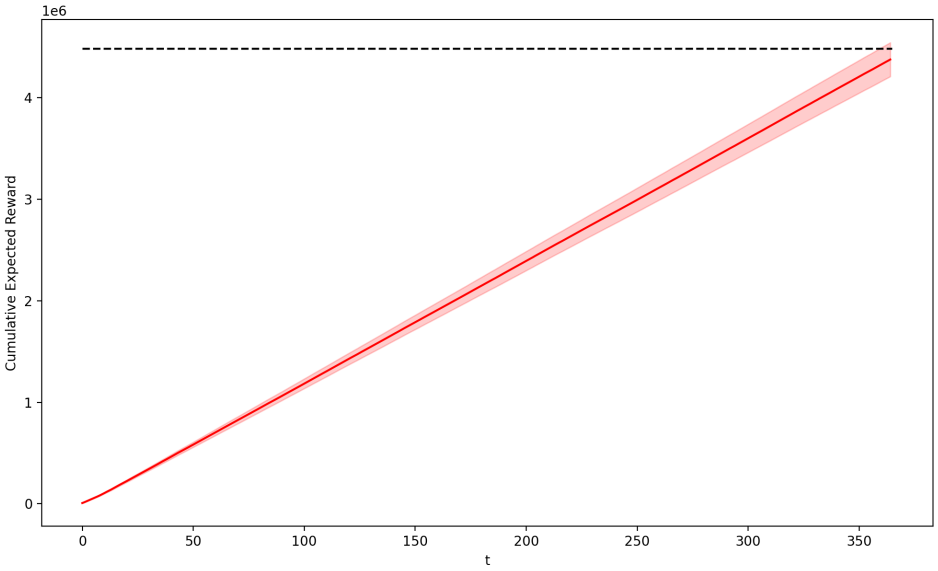


Figure 49: Cumulative Expected Reward, setting 0

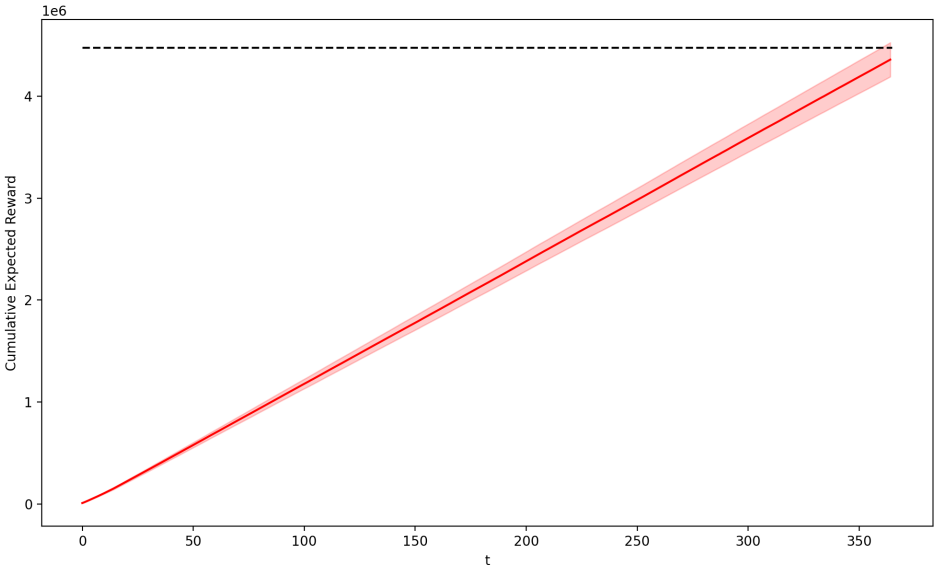


Figure 50: Cumulative Expected Reward, setting 1

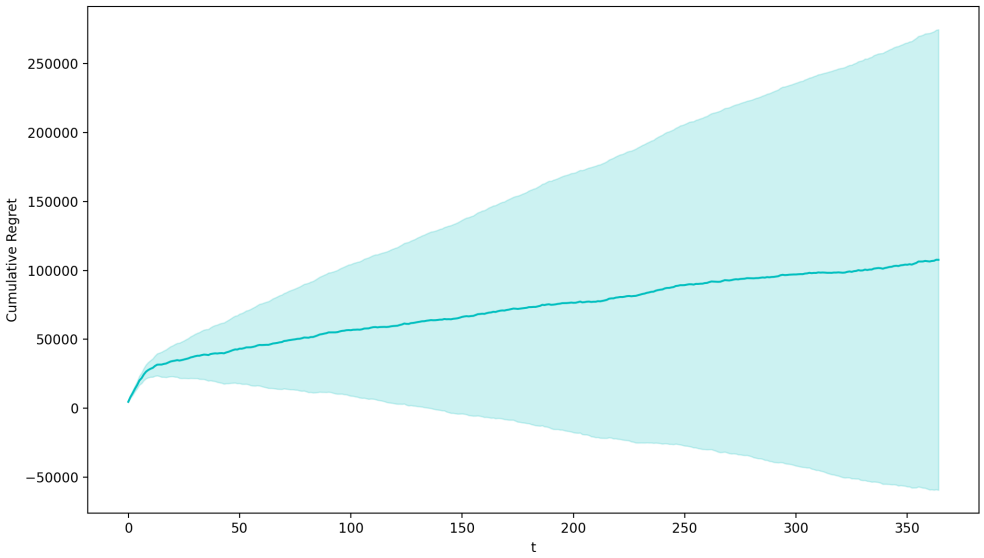


Figure 51: Cumulative Expected Regret, setting 0

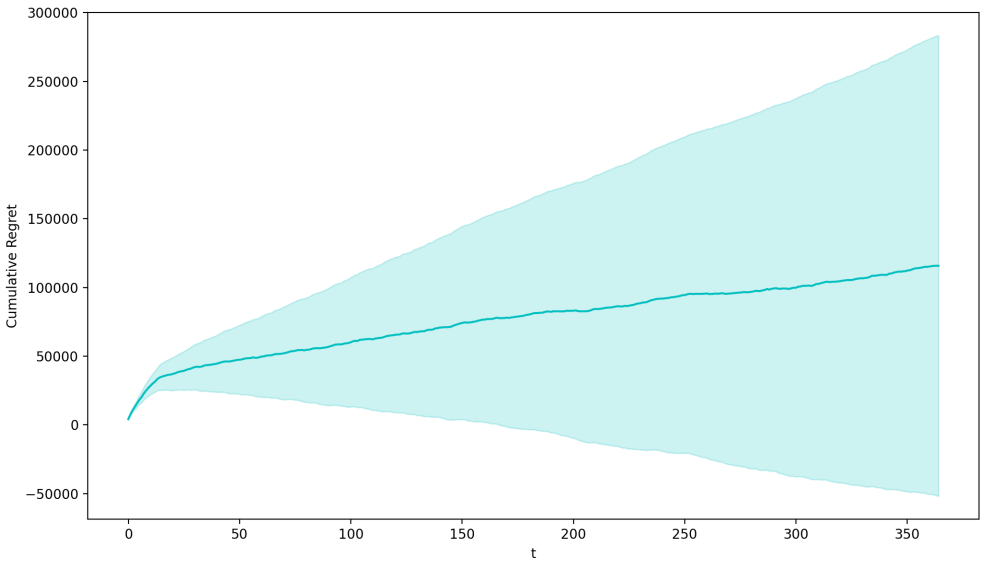


Figure 52: Cumulative Expected Regret, setting 1

6.2 Step 7

As in Step 6, the aim of the Step 7 is to solve the combined problem of pricing and matching. This time, however, what changes is the fact that the conversion rates associated to the customer classes are no more stationary. We thought the interest in our products to be subject to the phenomenon of seasonality. Due to this assumption we designed the conversion rate functions to model different behaviours of the customers in different time intervals, during the year. More details about this can be found in the Appendix B.

6.2.1 Environment

The seasonality phenomenon implied some changes in the Environment class, with respect to the Step 6. As a matter of fact, the number of *phases*, which is four since they correspond to the seasons, and their duration, that is approximately *365 / 4* days, are included in this class.

When playing a round, that as usual is the visit of a new customer, the Environment provides Bernoulli rewards. These Bernoulli rewards must be extracted, this time, from a distribution that has its mean in the value of the conversion rate associated to the class to which the customer belongs, referred to the right season, that is the one in which the current round is played.

6.2.2 Algorithm

To face the new problem affected by seasonality we implemented a Thompson Sampling algorithm with the Sliding-Window approach. The implementation of the sliding-window is the main characteristic that differentiate this step from the previous one. As a matter of fact, the choice of the arm to be pulled during the time steps of the experiment, as well as the computation of the optimal matching, have been left practically unchanged.

As it has been done in the previous steps, we use Beta distributions to give an estimate of the customers' behaviours. The difference here consists in the amount of rewards considered to update the data about the Betas, being limited. The hyper-parameter that sets how many samples to retrieve from the rewards records is the `window_size`. It has been defined in the `config_7.py` file, and has been set, in accordance to well known best practices, to $\sqrt{T}$, where T is the experiments time horizon.

6.2.3 Results

The proposed solution has been obtained with 15 experiments, 9 alternative prices for the first item and 5 different prices for the second one.

As can be seen from the expected daily reward plots, the seasonality phenomenon is accurately detected, after an initial settling phase. This reflects in the daily regret plots in which we can appreciate an overall constant behaviour with almost perceptible spikes in correspondence of the change of season. In accordance with the previous observations, the cumulative regret curves do not tend to flatten out.

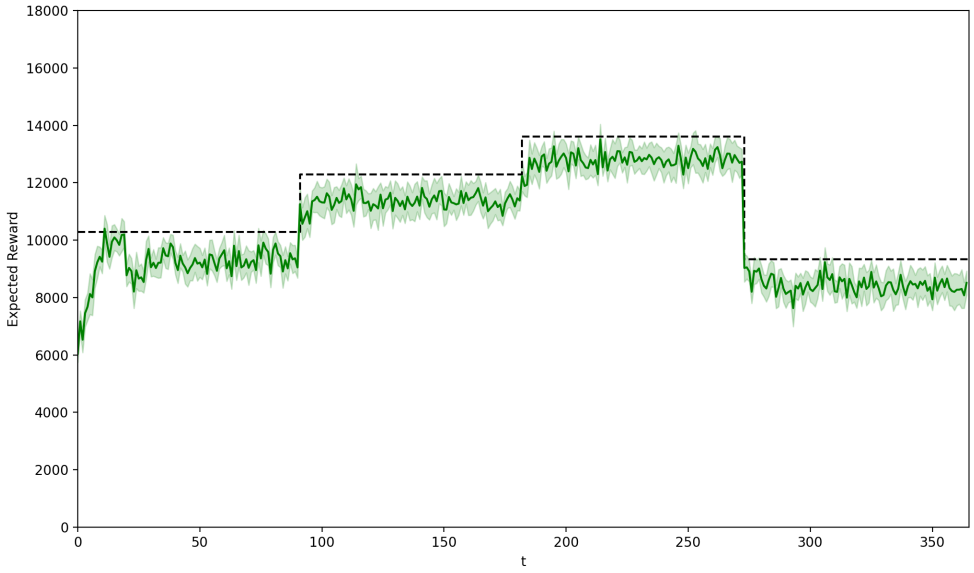


Figure 53: Expected Reward, setting 0

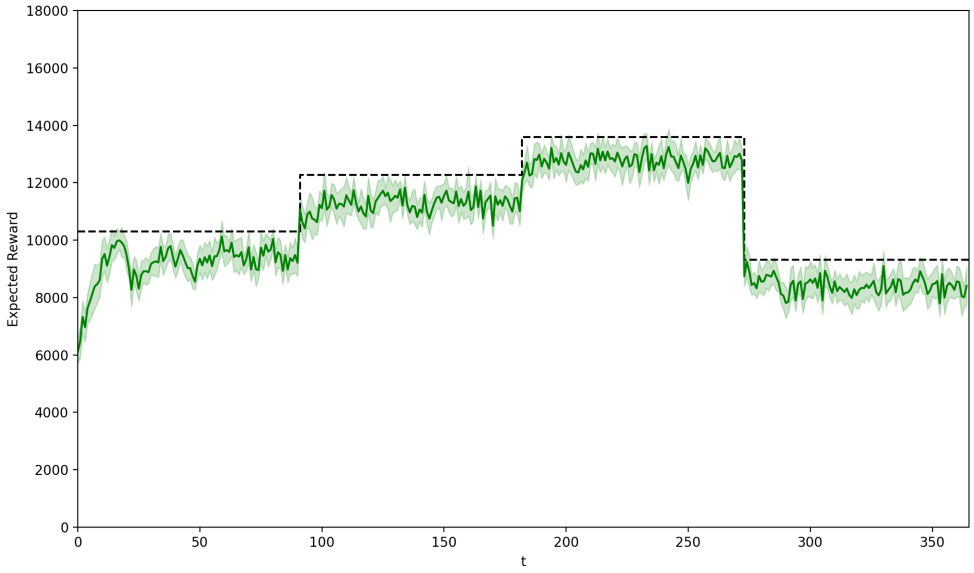


Figure 54: Expected Reward, setting 1

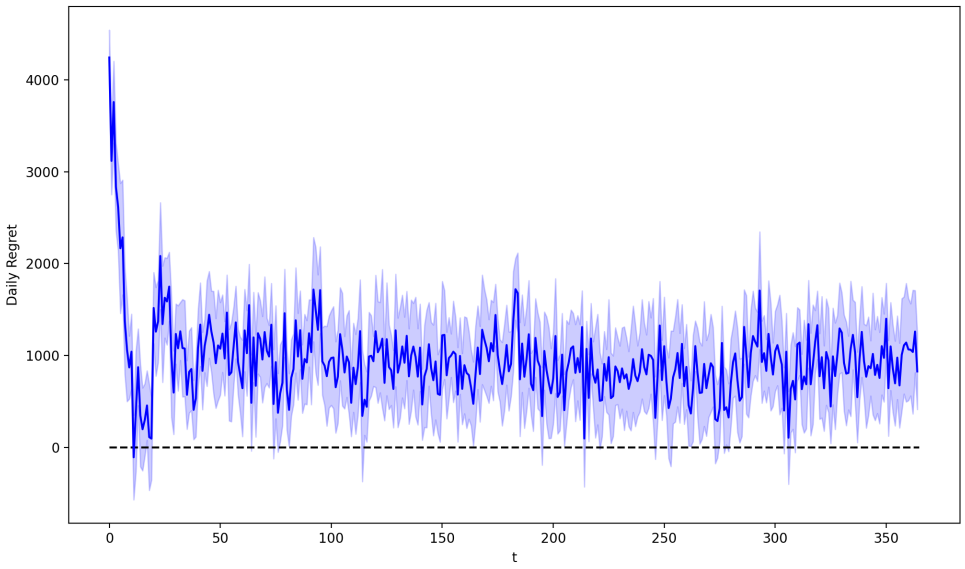


Figure 55: Daily Regret, setting 0

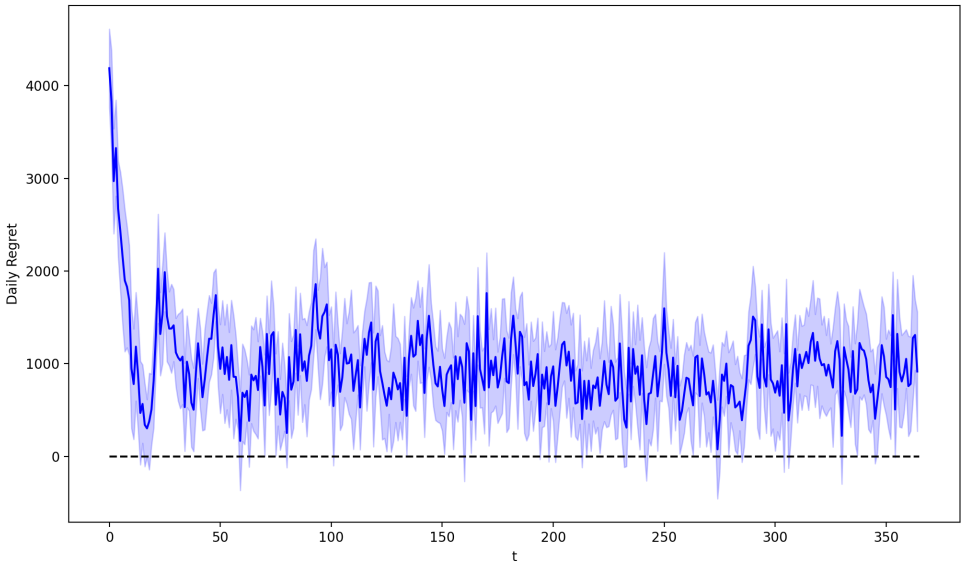


Figure 56: Daily Regret, setting 1

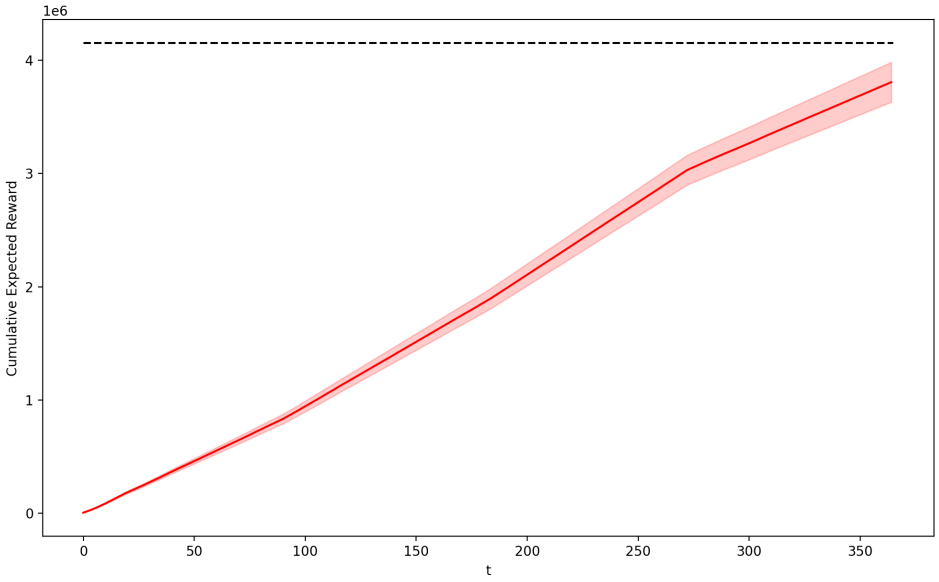


Figure 57: Cumulative Expected Reward, setting 0

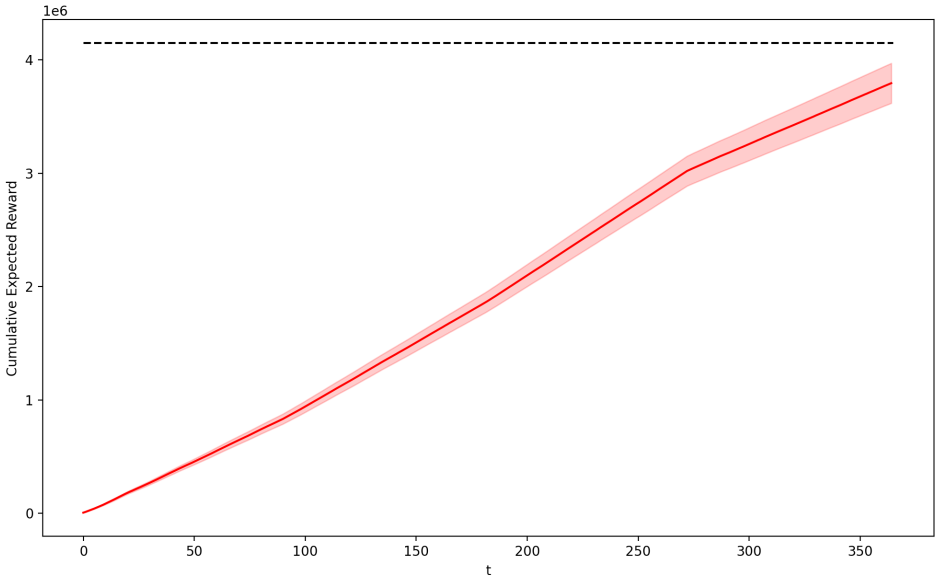


Figure 58: Cumulative Expected Reward, setting 1

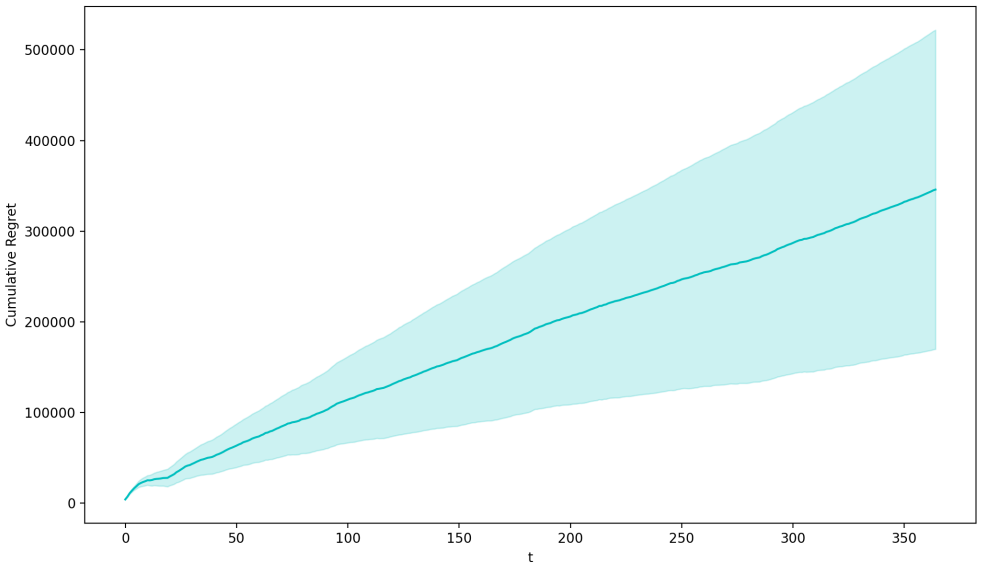


Figure 59: Cumulative Expected Regret, setting 0

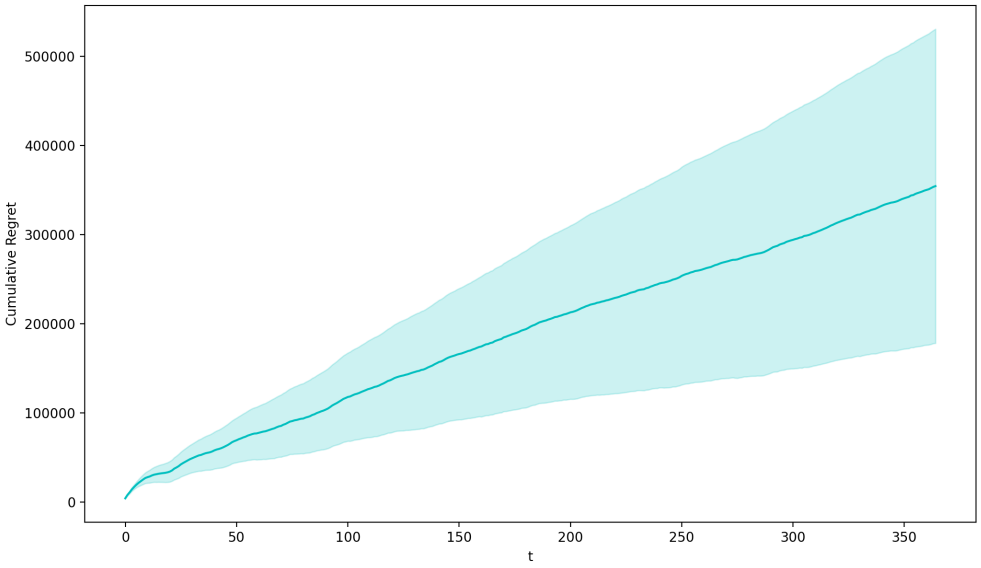


Figure 60: Cumulative Expected Regret, setting 1

6.3 Step 8

6.3.1 Environment

In this last Step, the Environment is non-stationary, but differently from above another learning approach is used: the change-detection approach. For what concerns the Environment, the exact same structure of the previous Step is preserved.

6.3.2 Algorithm

The algorithm on which the solution is based is the Cumulative Sum UCB Matching. The main idea is the following:

1. For every experiment, create an Environment object and a Learner object;
2. For every day, build a set of customers that will visit the shop;
3. For each customer entering the shop, the combined pricing and matching problem is solved by the Learner, which outputs both the *pulled_arm* and the *matching_prob* $[\mathbf{P}]_{ij} = \mathbb{P}(\text{choose promo } j \,|\, \text{customer of class } i)$;
4. Then, the normalised[1] profit associated with the current customer is computed, and this enters the `update` function of the Learner as one of the arguments.
5. At the end of the day, we update the knowledge about the number of customers according to the Bayesian Normal posterior update rule.

At this point we would like to go into the details of the CUSUM UCB Algorithm, adapted to our case. First, we describe the new classes introduced to implement it. There are `CUSUM_UCB_Matching` objects, and there exist one of them for every couple of ($price_1$, $price_2$) in the Learner. They contain:

- a 16-dimensional vector, `change_detection`, which is crucial for the detection of a change in the behaviour of the customers, so in their conversion rates, during the year. It is derived from the 1D rewriting of a 4×4 matrix, with rows corresponding to classes and columns to promos - which we recall is specific for the current couple of prices. Each element of the vector is a `CUSUM` object, which contains the logic behind the change detection, and is made up of the following parameters:
  - M, an integer, representing the number of observations required to compute the reference point;
  - $s^+ = (sample - reference) - \varepsilon$ and $s^- = -(sample - reference) - \varepsilon$ measuring the "corrected"gap between the *sample value* and the *reference*;
  - ε a small positive number;
  - the cumulative gaps g^+ and g^-, defined as the maximum between zero and the cumulative sum of the corresponding s^+ or s^-.
  - h is the threshold for the detection.
- a 16-dimensional vector of `empirical_means`, one per `CUSUM` object;

[1] The CUSUM UCB Matching Algorithm is implemented assuming rewards in the range $[0, 1]$, and to be compliant with that, we divide our values by the maximum reachable reward, equal to $\max\{m_1 + m_2\}$. The reason is mostly practical, since in the assumed scenario all the values of the parameters characterising the algorithm (M, ε, h, and α) have already been tuned.

- the corresponding `confidences` vector;
- a list of `valid_rewards_per_cell` - where a cell refers to a `CUSUM` object - containing the collection of rewards starting from the first time instant after the latest detection;
- a list of `detections` per `CUSUM` object;
- the parameter α, which is the small probability of constructing a random matching, instead of returning the matching obtained by maximising the upper confidence bound.

Let us now unravel what is behind the `pull_arm` function:

> for every couple of $(price_1, price_2)$ we create the best matching by means of the `pull_cells` function of the `CUSUM_UCB_Matching` object associated with this couple of prices. This is achieved with the `linear_sum_assignment` function (with the maximising option set to *True*), providing as input matrix, with probability equal to $1 - \alpha$, a matrix filled with upper confidences according to the UCB approach, and with probability α a randomly generated matrix, with positive values in its cells.

> the winning couple, so the one with the maximum matching value, is the one pulled for the current customer, and its respective matching is the one that will be used to assign the promo.

Knowing the pulled prices and the promo probabilities, the update can begin. The Learner class calls the `update` function of the `CUSUM_UCB_Matching` object associated with the `pulled_arm`, which does the following actions:

>> increment the counter `t`, number of times the arm has been pulled, since the last detection;

>> check whether the new normalised profit causes a change detection, by relying on the `update` function of `CUSUM` object - namely, if at least one of the g^+ or g^- quantities is grater than the threshold;

>> if there is a detection, this is appended to the specific list, `valid_rewards_per_cell` is reinitialised with just this last reward, and then the counter, g^+ and g^- and the reference M of the `CUSUM` object are reset to 0;

>> otherwise, the reward is added to the list of valid rewards, the empirical mean of the `pulled_arm` is consequently updated and the same happens for the confidences, in this case, of all the arms.

6.3.3 Results

The proposed solution has been obtained with 15 experiments, 9 alternative prices for the first item and 5 different prices for the second one.

As it clearly appears, the detection is successful in two out of three cases, when the change in the optimal value is sufficiently evident. However, we notice that, except in the third phase, the gap between the achieved reward and the optimal one, reduces. Consequently, we can observe in the daily regret plot, that zero is progressively approached, except for the third season. Overall, due to the aforementioned bad performance, the cumulative reward remains quite far from the optimal value.

Finally, by looking at the cumulative regret curve, that in this case is piecewise linear, we notice that after a relevant increase in slope in the third phase, the last one flattens out considerably, since, as we mentioned above, the last change is punctually detected.

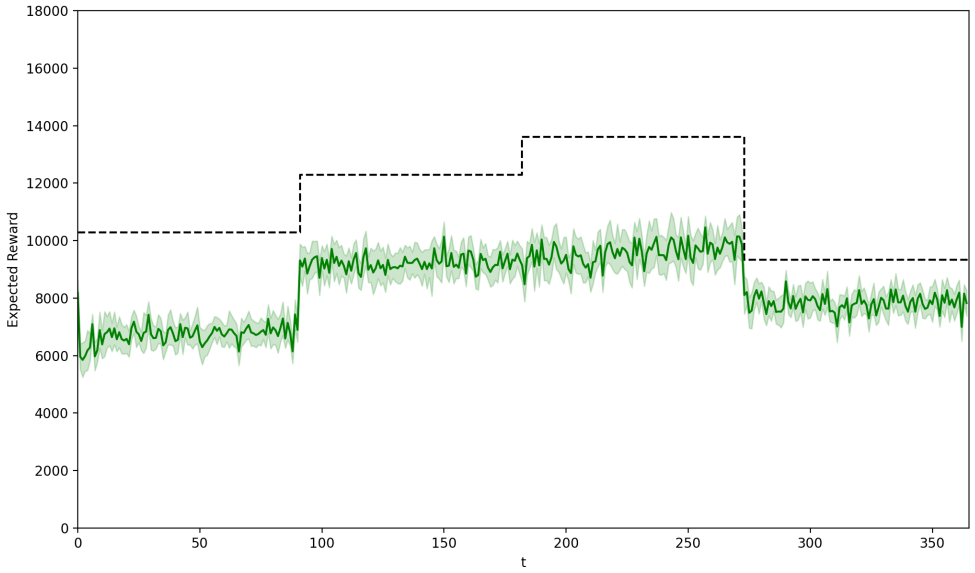


Figure 61: Expected Reward, setting 0

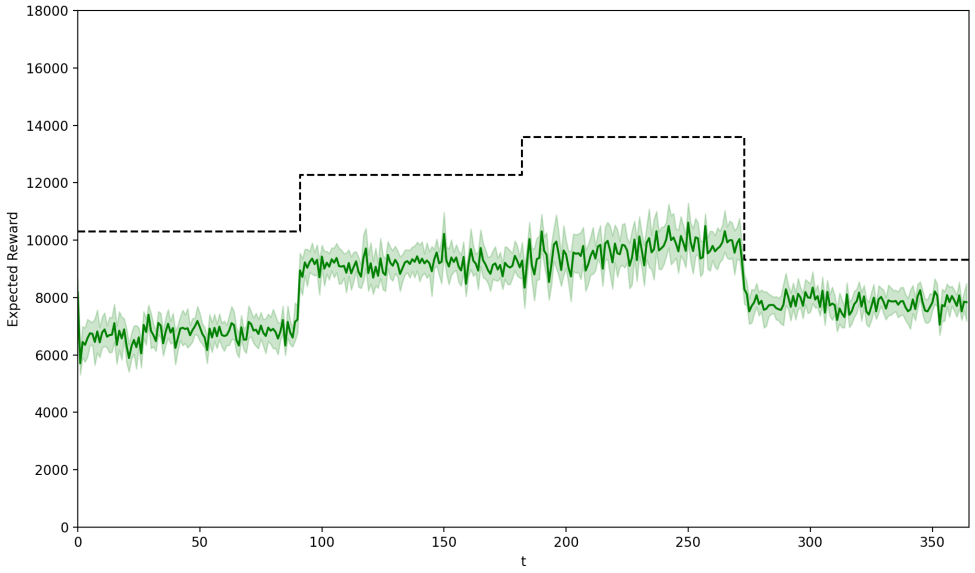


Figure 62: Expected Reward, setting 1

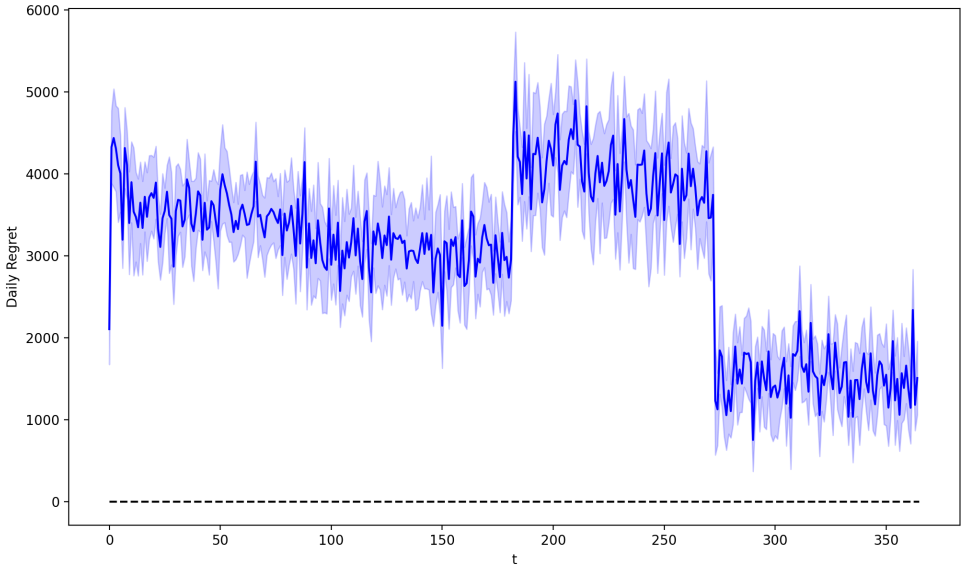


Figure 63: Daily Regret, setting 0

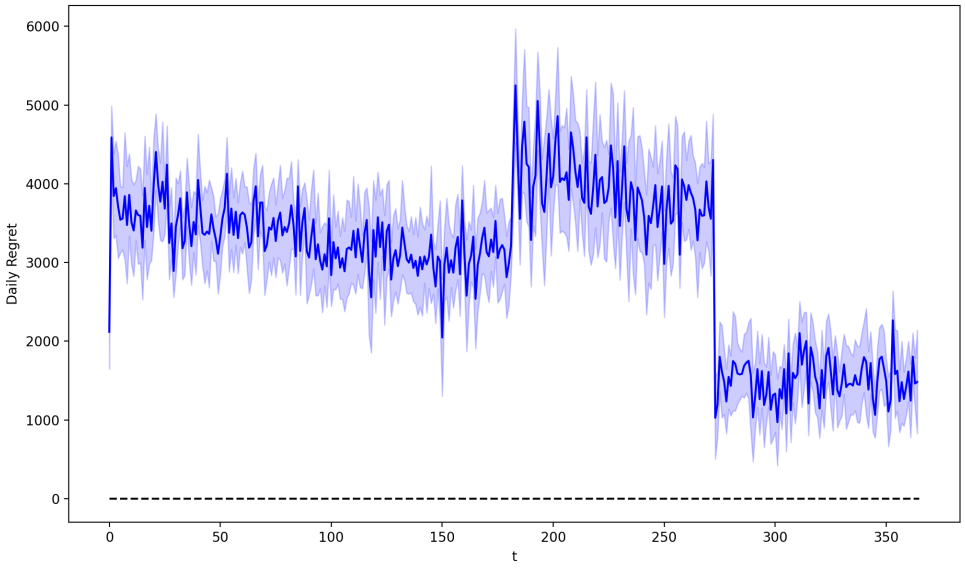


Figure 64: Daily Regret, setting 1

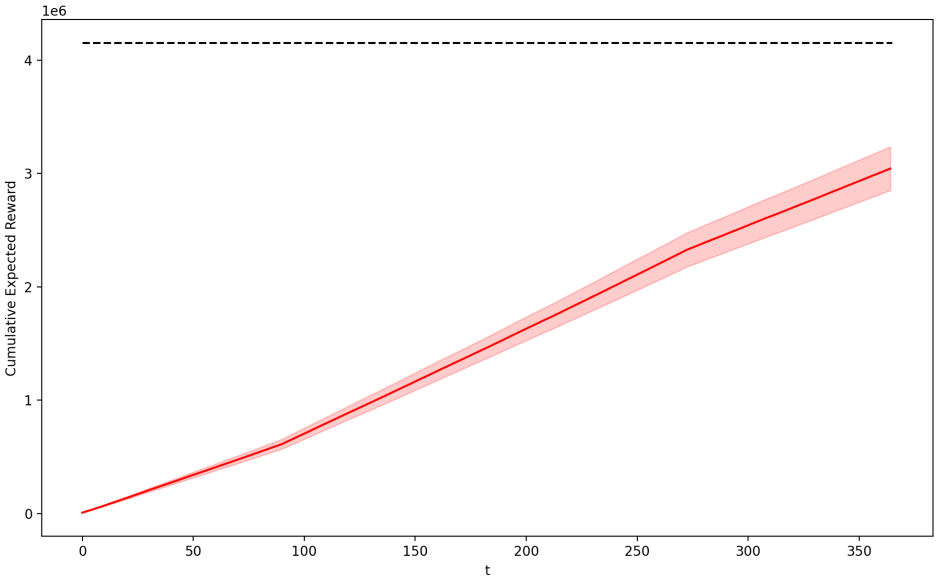


Figure 65: Cumulative Expected Reward, setting 0

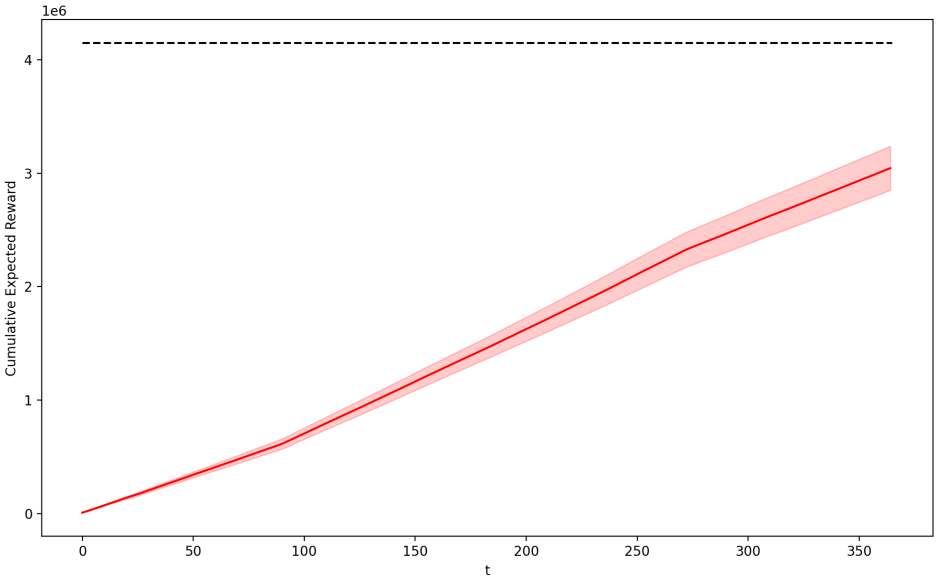


Figure 66: Cumulative Expected Reward, setting 1

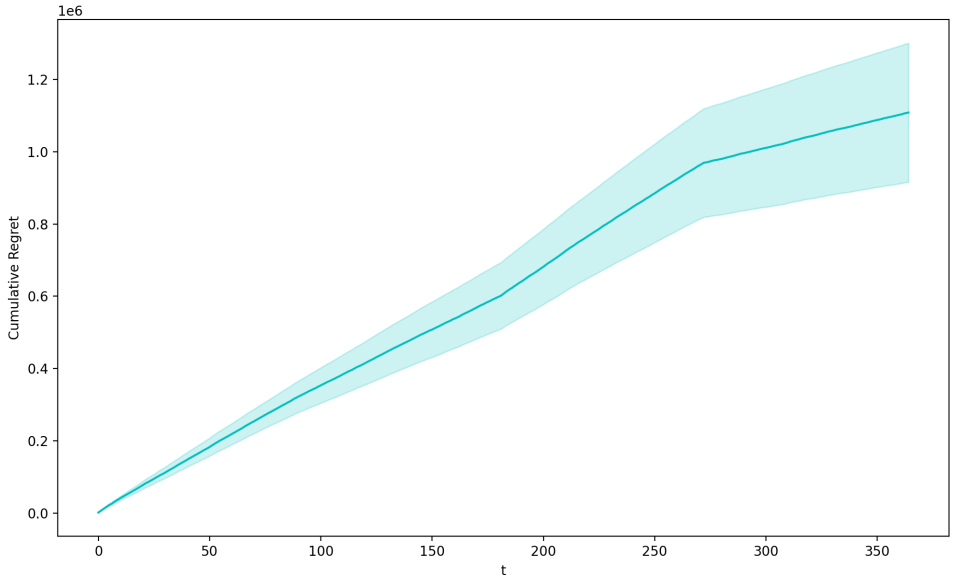


Figure 67: Cumulative Expected Regret, setting 0

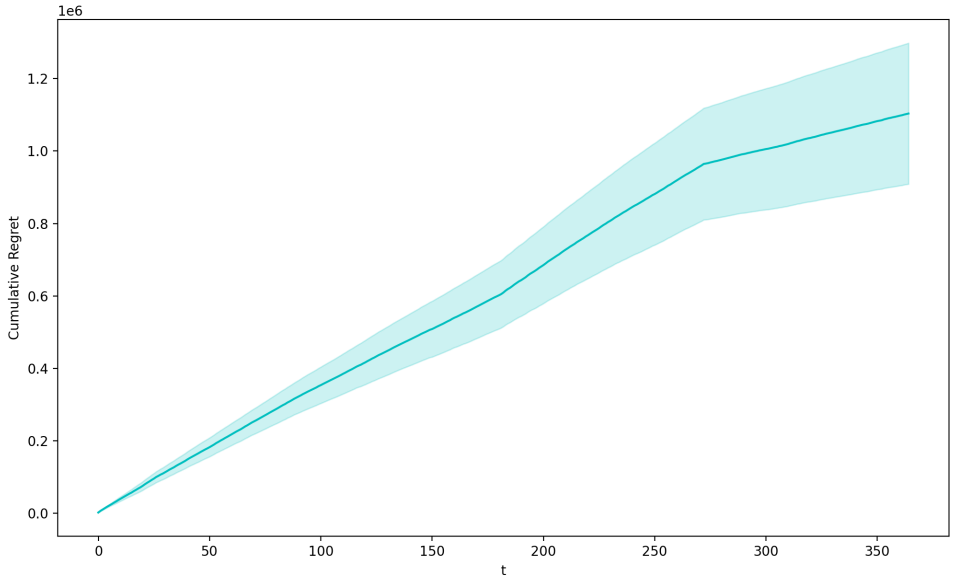


Figure 68: Cumulative Expected Regret, setting 1

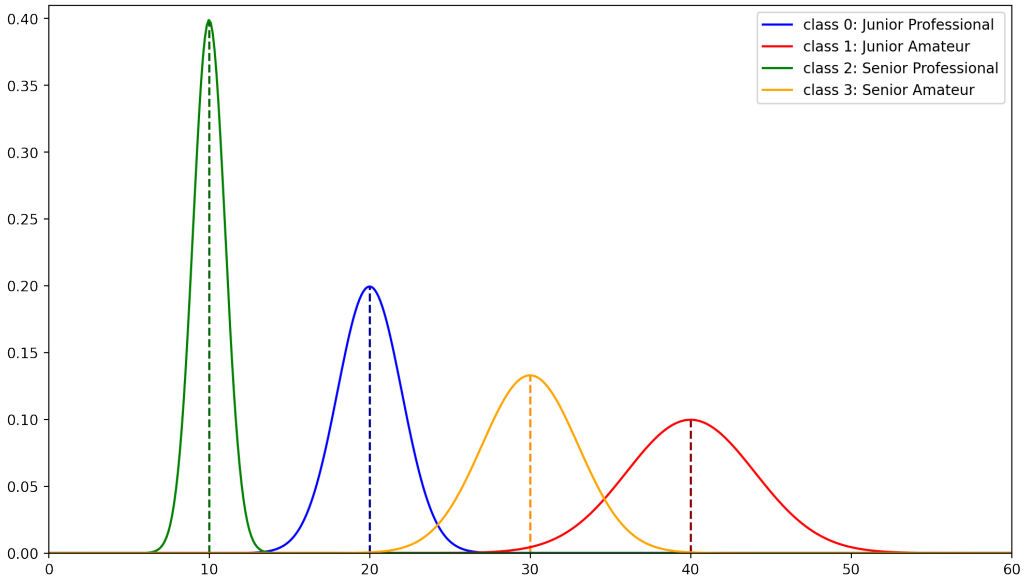
Appendix

A Customers

We report here the choices made to build the four truncated Gaussian distributions used to model the daily number of customers for each class.

	mean	stdev	lower bound	upper bound
Class 0 (JP)	20	2	0	50
Class 1 (JA)	40	4	0	100
Class 2 (SP)	10	1	0	25
Class 3 (SA)	30	3	0	75

Parameters of the truncated Gaussian distributions



Truncated Gaussian distribution for daily number of customers

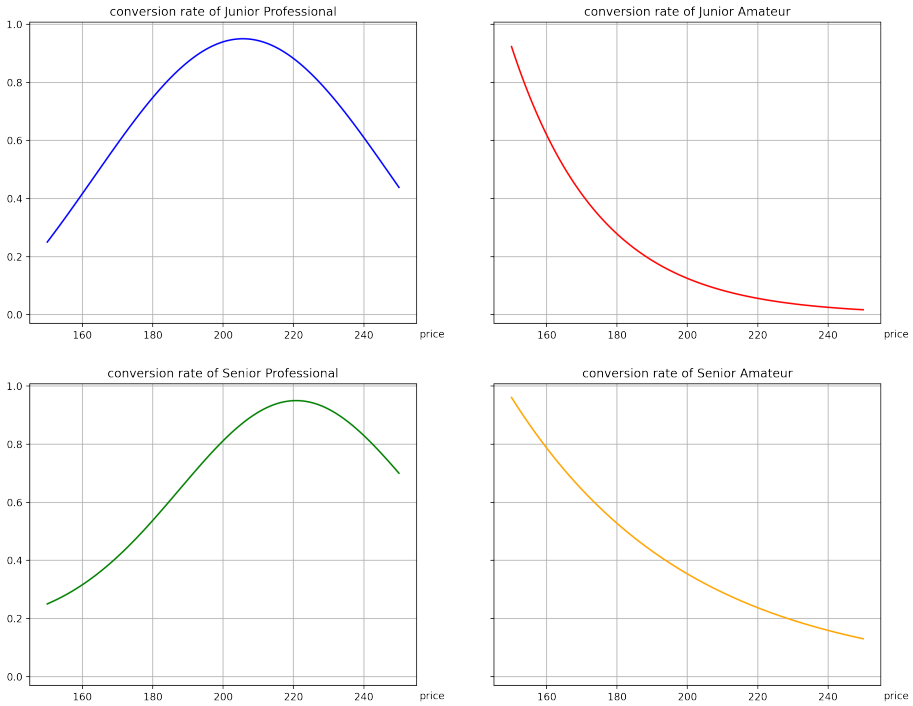
B Conversion rates

In this section we show the mathematical functions we used to represent the conversion rates of every customer class. Those functions take as input a price and return the corresponding conversion rate.

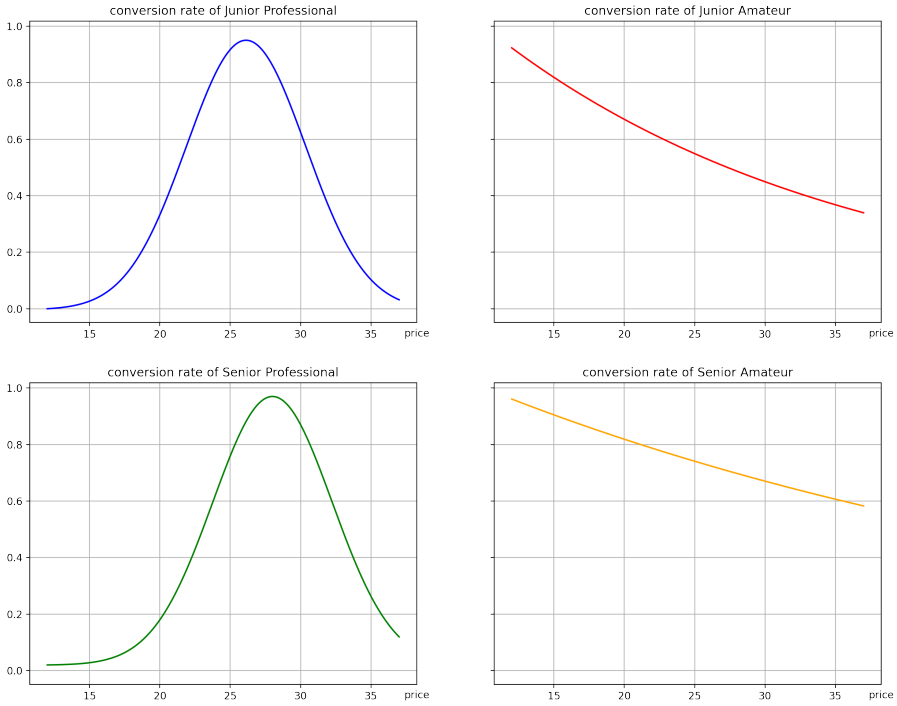
B.1 Stationary case

Here we explain the idea behind the assumptions made to build the conversion rates for the four classes. For their purposes, professionals use products of a high level, whose costs stay around a certain - high - price. The amateurs, instead, behave in an opposite way, meaning that they buy with a higher probability when prices are convenient.

The reasoning above holds in general for both items, but to be more precise, about the second item (the trekking sticks) we went for a smaller variance for the professionals, since they may prefer other pieces of equipment rather than the sticks, maybe more technical. Conversely, mainly for safety reasons, amateurs might feel safer with a pair of sticks, so their average purchasing probability is higher.



Conversion rate functions for the four customer classes for the first item

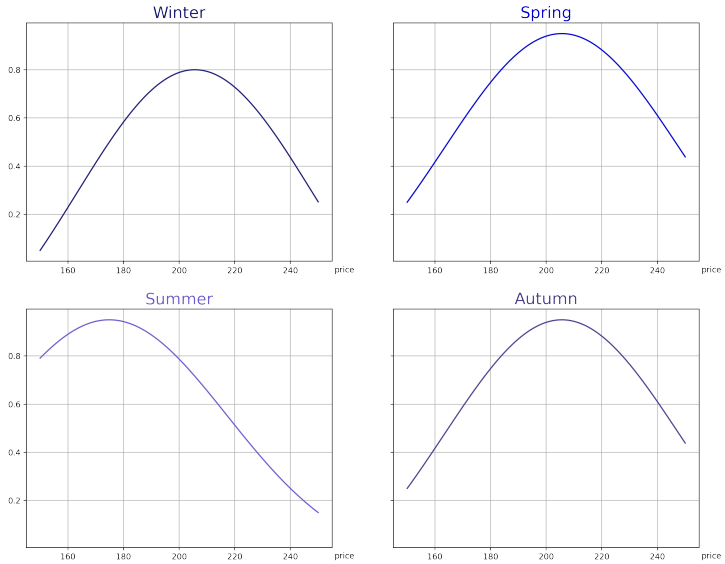


Conversion rate functions for the four customer classes for the second item

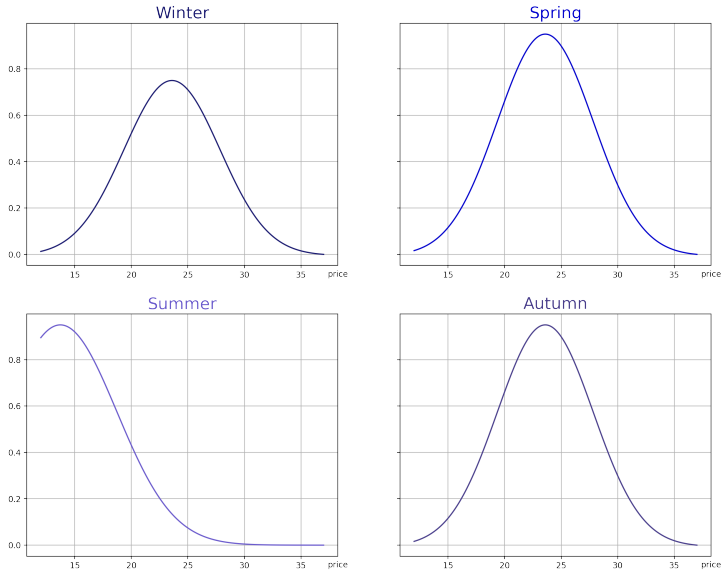
B.2 Non-stationary case

Here we explain the reasons behind the modelling choices made for the non-stationary case. We decided to adapt the previously built conversion rates of the two items according to some reasonable ideas.

As regard the first class of customers, the Junior Professionals, they could tend to buy shoes in Spring just before the beginning of the warm trekking season, or right at the end of it, in Autumn. Then, during the Winter period the buying trend follows the Spring/Autumn one but on a lower level. Conversely, in the Summer they just buy in case of extreme necessity or of appealing discounts.

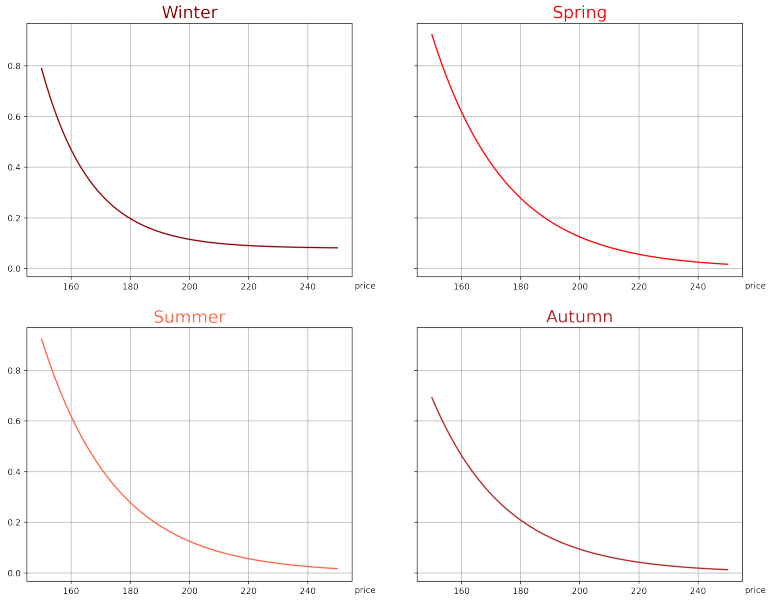


Conversion rate functions for the Junior Professionals (customer class 0) for the first item

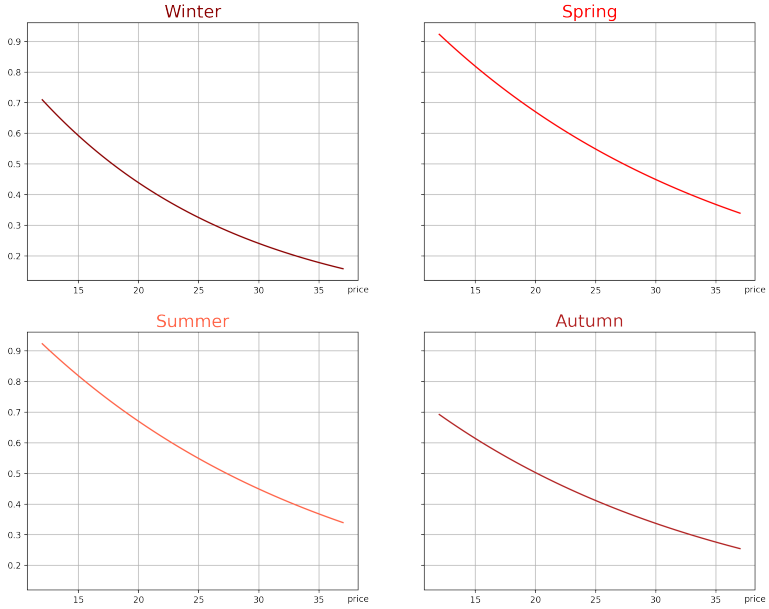


Conversion rate functions for the Junior Professionals (customer class 0) for the second item

For the second class, the Junior Amateurs, they could tend to buy shoes in Spring and Summer, before and during the the warm trekking season, since usually this period is the preferred for trips in the mountains. Then, during the Autumn, since the climate can still be mild and the landscape colours very appealing, they may be interested in the purchase. Instead, in the Winter the interest is likely to be very low, and only risen by relevant discounts.

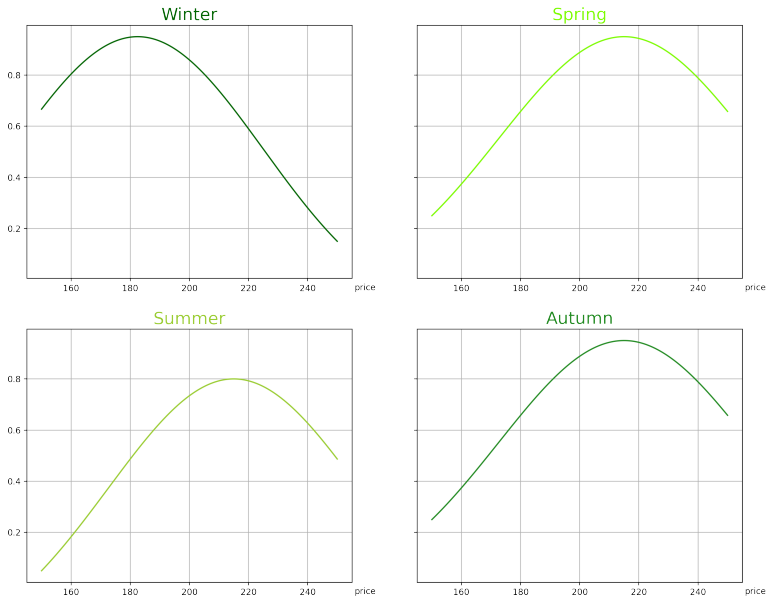


Conversion rate functions for the Junior Amateurs (customer class 1) for the first item

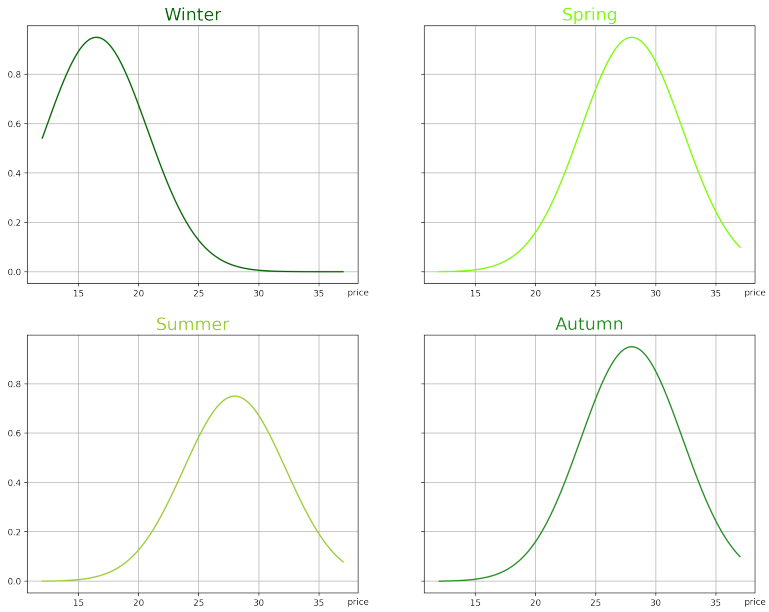


Conversion rate functions for the Junior Amateurs (customer class 1) for the second item

In the case of Senior Professionals Spring and Autumn are identified as top periods, based on the idea that they buy those items before the beginning of the season or at its end (if they need to be substituted due to intensive usage). Differently, during the high season they do not generally have the necessity to buy them, so the purchase happens only in some special occasions. Finally, during the Winter, due to unfavourable weather conditions that cause a reduced usage of the equipment, they may buy only in presence of extreme discounts.

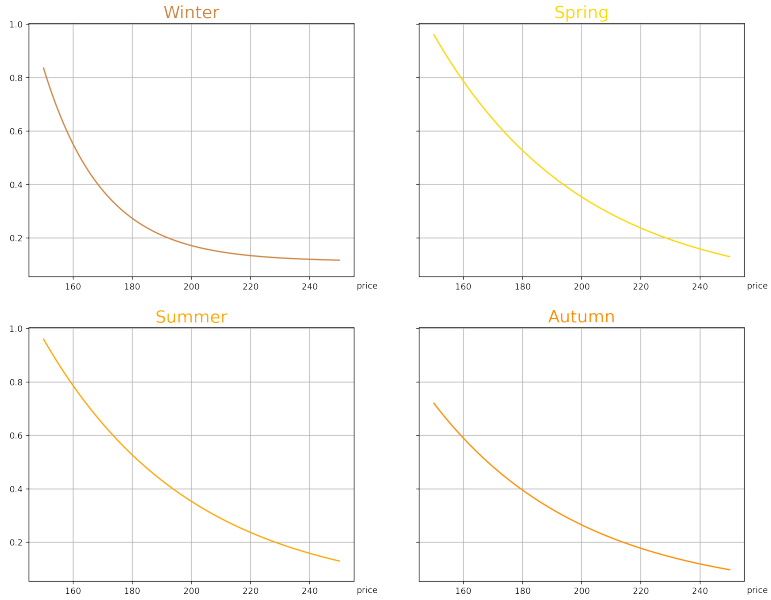


Conversion rate functions for the Senior Professionals (customer class 2) for the first item

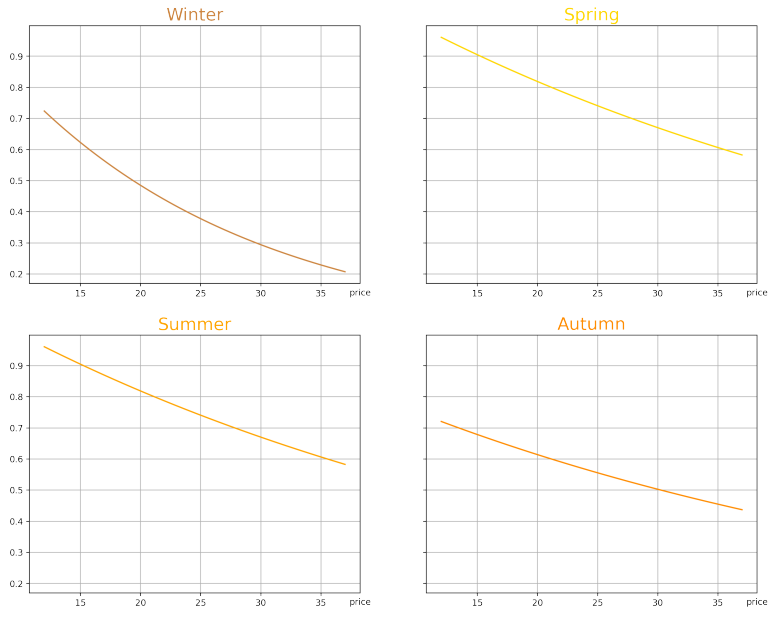


Conversion rate functions for the Senior Professionals (customer class 2) for the second item

Finally, the behaviour of the Senior Amateurs is very similar to the Junior Amateurs' one, with the differences that they may have a higher disposable income, and they might prefer to invest in a product of higher quality for safety reasons.



Conversion rate functions for the Senior Amateurs (customer class 3) for the first item



Conversion rate functions for the Senior Amateurs (customer class 3) for the second item

The generating code for this functions can be found in the configuration file (`config_i.py`) contained in each step folder.