
IE4214 – Revenue Management and Pricing Analytics

Group 4 – Option 1: Choice Prediction and Assortment Optimization

2022/2023, Semester 2


| Group Members | Matriculation Number |
|---|---|
| Bauer Charles William | A0268013L |
| Cai Xuanxuan | A0220596A |
| Chen Yingling | A0221681L |
| Liu Wei | A0223137N |

# Table of Contents

# 1. Data pre-processing

In this section, we will explain the data pre-processing steps that we have taken, to transform the data according to the requirements of the models which we will be using.

## 1.1. Original given dataset

- *"assortment.txt"*
- *"assortment_test.txt"*
- *"probability.txt"*

## 1.2. Assortment with probability

Combined *"assortment.txt"* with *"probability.txt"* into a new csv file named as *"clean_train.csv"* (see Fig.1).

| | Assortment | Probability |
|---|---|---|
| **0** | [0.0, 2.0] | [0.6171762553809882, 0.382823744619012] |
| **1** | [0.0, 22.0, 27.0] | [0.49010203287014026, 0.3856208197783525, 0.12... |
| **2** | [0.0, 7.0, 17.0, 22.0] | [0.17383055343275639, 0.44812960860117884, 0.2... |
| **3** | [0.0, 17.0, 7.0, 21.0, 14.0] | [0.1338762504510885, 0.1933610050447225, 0.365... |
| **4** | [0.0, 7.0, 2.0, 25.0, 30.0, 13.0] | [0.09520137647807961, 0.4258881190119289, 0.02... |
| **...** | ... | ... |
| **2495** | [0.0, 13.0] | [0.37037705537265336, 0.6296229446273469] |
| **2496** | [0.0, 23.0, 17.0] | [0.25241095695520893, 0.10976818151988744, 0.6... |
| **2497** | [0.0, 11.0, 10.0, 29.0] | [0.13702096921422796, 0.17458857209162085, 0.6... |
| **2498** | [0.0, 16.0, 25.0, 13.0, 1.0] | [0.167781680054737, 0.29015312192773385, 0.148... |
| **2499** | [0.0, 22.0, 10.0, 15.0, 4.0, 17.0] | [0.0784077752482451, 0.03763805678406826, 0.51... |

*Figure 1. "clean_train.csv" file.*

## 1.3. Feature Table

Based on the information given in the question, we incorporate the 30 products with their corresponding features and transform them into a list in the order of "Cores", "Frequency", "TDP" and "Price". For the outside option, features would have a value of 0 as $e^0$ is 1 based on the Multinomial Logit Model (MNL) (see Fig.2).

| | Cores | Frequency | TDP | Price |
|---|---|---|---|---|
| **0** | 0 | 0.0 | 0 | 0 |
| **1** | 4 | 3.2 | 95 | 3000 |
| **2** | 4 | 3.2 | 95 | 2700 |
| **3** | 4 | 3.2 | 95 | 2400 |
| **4** | 4 | 3.2 | 95 | 2100 |
| **5** | 4 | 3.2 | 95 | 1800 |
| **6** | 8 | 2.9 | 60 | 3000 |

```
[[0.0, 0.0, 0.0, 0.0],
 [4.0, 3.2, 95.0, 3000.0],
 [4.0, 3.2, 95.0, 2700.0],
 [4.0, 3.2, 95.0, 2400.0],
 [4.0, 3.2, 95.0, 2100.0],
 [4.0, 3.2, 95.0, 1800.0],
 [8.0, 2.9, 60.0, 3000.0],
```

*Figure 2. An illustration of the outside option and the first 5 products with their respective features.*

# 2. Part (a). Prediction of Choice Probability

## 2.1. Biogeme

The first method being used is Biogeme which is an open-source Python package which mainly used to work on discrete choice model [1]. In this section 2.1, please refer to *"Part (a)-Biogeme_MAE_MSE_RMSE.ipynb"* and *"biogeme_nice.py"*.

Biogeme only allows input for y as discrete, however the "Probability" file contains continuous values which represent the choice probability. If we simply change the choice probability to the discrete choice of product, there will be loss of information as probability of 0.51 or 0.99 both leads to the product being chosen. Hence, we decided to expand the dataset such that the loss of information is minimized.

Taking the assortment [0,2] as example, it has a choice probability of [0.6171762553809882, 0.382823744619012]. 100 sets of [0,2] assortments will be created while 62 sets have chosen 0 and 38 sets chosen 2 (see Fig.3). Therefore, a new set of training data, named as *"av_df100.csv"* has 250000 datasets which is expanded from the original 2500 set of datapoints. Biogeme also requires an availability column for each choice denoting whether the choice is present in the assortment. We took the information from *"assortment.txt"* to create these columns.

```
In [17]: av_df = pd.read_csv("av_df100.csv")
         av_df[:100]
```

Out[17]:

| | Choice | Av0 | Av1 | Av2 | Av3 | Av4 | Av5 | Av6 | Av7 | Av8 | ... | Av21 | Av22 | Av23 | Av24 | Av25 | Av26 | Av27 | Av28 | Av29 | Av30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 95 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 96 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 97 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 98 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 99 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

100 rows × 32 columns

*Figure 3. New train data used for Biogeme training.*

With this new set of training data, it was split into a train set and a test set which are 80% and 20% respectively. Based on the training sets and the feature table being produced in Section 1.3, Biogeme is used to predict two sets of Beta values which are the set without intercepts and the set with intercepts (see Fig.4).

```
# Without intercept
b_dict = biogeme_nice.calc_beta(train, x)
# With intercept
b_dict_intercept= biogeme_nice.calc_beta(train, x, intercept = True)
b_dict_intercept, b_dict
```

```
({'B1': 0.3689119733638879,
  'B2': 0.45561663518583523,
  'B3': -0.024650420427150353,
  'B4': -0.0005100529345544204,
  'a1': -0.14011197853611174,
  'a10': 0.04777133113413791,
  'a11': 0.18115942026783077,
  'a12': 0.13486113295127966,
  'a13': 0.07216628575817102,
  'a14': 0.017863101277800438,
  'a15': 0.019326247123335846,
  'a16': 0.7639650808762702,
  'a17': 0.7843176986659827,
  'a18': 0.8850969581702269,
  'a19': 1.0218025972283502,
  'a2': -0.22568849847533812,
  'a20': 1.1740748701985007,
  'a21': -0.8350159040673462,
  'a22': -0.7615151736234451,
  'a23': -0.6233351341860597,
  'a24': -0.5106991939607336,
  'a25': -0.42544384352226666,
  'a26': -0.0030480554449849964,
  'a27': -0.04020905424073837,
  'a28': -0.17222839587701394,
  'a29': 0.019037269301225443,
  'a3': -0.1512995315668845,
  'a30': 0.08645806858696514,
  'a4': -0.18978745200115965,
  'a5': -0.13354441279867066,
  'a6': -0.2227404669202979,
  'a7': -0.2294352986794968,
  'a8': -0.2316693883660647,
  'a9': -0.15697555320614898},
 {'B1': 0.32128960041067456,
  'B2': 0.47677715513439084,
  'B3': -0.023165748712894552,
  'B4': -0.0004657626706053981})
```

*Figure 4. Beta values from Biogeme.*

The training set's Beta values are then used to calculate the choice probability of the test set based on their features. These predicted choice probabilities are then used to compare with

the true values in the test set to compute mean-absolute error (MAE) and root-mean-squared error (RMSE) for comparison with other models.

## 2.2. Loglikelihood

The second method uses NumPy and SciPy.Optimize to define and optimize the loglikelihood function as covered in the lecture. In this section 2.2, please refer to *"Part (a) & Part (b) Loglikelihood.ipynb"* and *"loglikelihood.py"*.

First, the loglikelihood function is written as a function of the training data and beta. Then, the minimize function finds the beta values that result in the lowest loglikelihood given the training data. The training data contains choice probability instead of discrete choice. Under this method, the information from *"availibility.txt"* and *"probability.txt"* are combined into a two-dimensional y array. For each row, the y contains 31 values with the decimal choice probabilities and zeros for choices not in that row's assortment (see Fig.5). The same x feature array is used.

```
[[0.61717626 0.         0.38282374 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.        ]
 [0.49010203 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.38562082 0.
  0.         0.         0.         0.12427715 0.         0.
  0.        ]]
```

*Figure 5: The first two rows of the y array.*

"*loglikehood.py*" uses multiple helper functions that were used to check that the code met its intended functionality. softmax2d() finds the predicted choice probability from the utility. predict() calculates the utility from beta, x, and the availability information stored in y. ll() calculates the loglikelihood, which is the true y times the log of the predicted probability summed over the available choices for each row. The calc_beta() function calls each of these functions and SciPy to return the beta that minimizes the negative loglikelihood.

Once the model's beta is found, the same prediction functions can be used as in Section 2.1. The same train-test split is used to compare error metrics.

## 2.3. Gradient Boosting

This section describes the implementation of the Gradient Boosting method in conjunction with a Multioutput Regressor, Softmax function, and normalization. For an in-depth understanding, please refer to the accompanying notebook *"Part (a)-Gradient Boosting.ipynb"*.

Due to the varying dimensions of the original dataset containing assortments and their respective probabilities, the data must be pre-processed to maintain consistency. To achieve this, the table is expanded such that each data point consists of six products and six associated probabilities. Additional products are assigned a value of negative infinity, and the corresponding choice probabilities are set to zero. A visualization of the resulting data structure can be found in Fig.6. This pre-processing step ensures that the Gradient Boosting model can be effectively trained and applied to the dataset.

| | Assortment | Probability |
|---|---|---|
| **0** | [0, 2, -inf, -inf, -inf, -inf] | [0.6171762553809882, 0.382823744619012, 0.0, 0... |
| **1** | [0, 22, 27, -inf, -inf, -inf] | [0.49010203287014026, 0.3856208197783525, 0.12... |
| **2** | [0, 7, 17, 22, -inf, -inf] | [0.17383055343275639, 0.44812960860117884, 0.2... |
| **3** | [0, 17, 7, 21, 14, -inf] | [0.1338762504510885, 0.1933610050447225, 0.365... |
| **4** | [0, 7, 2, 25, 30, 13] | [0.09520137647807961, 0.4258881190119289, 0.02... |
| **...** | ... | ... |
| **2495** | [0, 13, -inf, -inf, -inf, -inf] | [0.37037705537265336, 0.6296229446273469, 0.0,... |
| **2496** | [0, 23, 17, -inf, -inf, -inf] | [0.25241095695520893, 0.10976818151988744, 0.6... |
| **2497** | [0, 11, 10, 29, -inf, -inf] | [0.13702096921422796, 0.17458857209162085, 0.6... |
| **2498** | [0, 16, 25, 13, 1, -inf] | [0.167781680054737, 0.29015312192773385, 0.148... |
| **2499** | [0, 22, 10, 15, 4, 17] | [0.0784077752482451, 0.03763805678406826, 0.51... |

2500 rows × 2 columns

*Figure 6. Appended data with consistent dimensions.*

Building on the pre-processed table, a new 2D matrix, *x_fea_mat*, is created for the assortments, where in each product is replaced with its corresponding features. The '-inf' products are assigned four features, each with a value of -1.0e+09. This particular value is chosen because, according to the MNL equation, the $e^{-1.0e+09}$ approximates to 0.

In addition to *x_fea_mat*, another 2D matrix *y_mat*, is constructed to represent the probabilities of the assortments. A visualization of both matrices can be found in Fig.7.

```
x_fea_mat[0],y_mat[0]
```

```
(array([ 0.0e+00,  0.0e+00,  0.0e+00,  0.0e+00,  4.0e+00,  3.2e+00,
         9.5e+01,  2.7e+03, -1.0e+09, -1.0e+09, -1.0e+09, -1.0e+09,
        -1.0e+09, -1.0e+09, -1.0e+09, -1.0e+09, -1.0e+09, -1.0e+09,
        -1.0e+09, -1.0e+09, -1.0e+09, -1.0e+09, -1.0e+09, -1.0e+09]),
 array([0.61717626, 0.38282374, 0.        , 0.        , 0.        ,
        0.        ]))
```

*Figure 7. An illustration of the x_fea_mat & y_mat for assortment [0,2].*

Utilizing the *x_fea_mat* and *y_mat* matrices, a Gradient Boosting model is implemented and wrapped with a Multioutput Regressor. This approach generates six predicted choice probabilities for each assortment. However, these probabilities do not necessarily sum up to 1. To address this, the Softmax function is applied, converting the outputs into probability distributions that lie between 0 and 1 and sum up to 1.

A subsequent issue arises when '-inf' products receive non-zero choice probabilities, which is unrealistic. To resolve this, a loop function is introduced to set the choice probabilities to 0 when the corresponding four features have a value of -1.0e+09. This modification creates another challenge, as the probabilities no longer sum up to 1 for each set of choice probabilities. The final step, therefore, is to normalize the predicted choice probabilities to ensure they sum up to 1.

To evaluate the model's performance, the dataset is divided into a train set (80%) and a test set (20%). The MAE and RMSE are computed using these sets, providing insight into the model's accuracy and overall effectiveness.

## 2.4. Error Metrics Results

The performance of the three models is assessed by comparing their MAE and RMSE. The dataset is split into 80% for training and 20% for testing purposes. The results of the evaluation are displayed in Table 1 below:

| Model | Biogeme (without intercept) | Biogeme (with intercept) | Loglikelihood | Gradient Boosting |
|---|---|---|---|---|
| **MAE** | 0.073158 | 0.034049 | 0.07364777318 | 0.07726193544834971 |
| **RMSE** | 0.102957 | 0.047718 | 0.11285702670592076 | 0.11290395172268695 |

*Table 1. MAE and RMSE for Biogeme, Loglikelihood and Gradient Boosting.*

Considering the performance metrics presented in the table above, the Biogeme model with intercept demonstrates the best performance among the three models. Consequently, this

model is selected for predicting the choice probabilities of products in each assortment found in the *"assortment_test.txt"* file. The predicted choice probabilities will be written to a new file called "*Group04.txt"*.

# 3. Part (b). Optimal Price

In this section, we will utilize the Beta values obtained from the entire dataset, rather than the values derived from the 80% training set used in the previous section. This approach is taken to enhance the accuracy of the findings related to optimal pricing. The Biogeme Beta values without intercept terms are employed, as they align with the equation of the MNL model below, which exclusively considers the Beta values for the features. In this project context, the $c_i = 0$ as there are no marginal costs.

$$maximize\ R(p) = \sum_{i=1}^{n} (p_i - c_i) \frac{\exp\ (a_i - b_i p_i)}{1 + \sum_{j=1}^{n} \exp\ (a_j - b_i p_j)}$$

$$a_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3}$$

$$b_i = -\beta_4$$

## 3.1. Minimization of Objective Value

In Section 3.1, the Beta values can be found in the following notebooks: "*Part (b)-biogeme_get_beta (Assortments with 1 product).ipynb*", "*Part (b)-biogeme_get_beta (All assortment).ipynb*", and "*Part (a) & Part (b) Loglikelihood*". The results of the optimal price computations can be referred to in the notebook "*Part (b)-Optimal price based on minimization of objective value.ipynb*".

The Biogeme model is not only applied to all assortments but also specifically to assortments with a single product, excluding the outside option. The 1-product assortment Biogeme model yields satisfactory MAE and RMSE results and is therefore employed in this section as well. Additionally, the Beta values obtained from the log-likelihood function are considered to explore different optimal prices based on various models. These Beta values pertain to Cores, Frequency, TDP, and Price (see Table 2).

| Model | Biogeme (All Assortment) | Biogeme (1 Product Assortment) | Loglikelihood |
|---|---|---|---|
| Beta for Cores | 0.3226564842308442 | 0.19776311682165013 | 0.3263116854084238 |
| Beta for Frequency | 0.4786270678113503 | 0.5507275166611583 | 0.49357456645769804 |
| Beta for TDP | -0.023199633313091498 | 0.015516593042817809 | -0.02355774017834474 |
| Beta for Price | -0.0004719818858727145 | 0.00047533994340908655 | -0.00048044594543504777 |

*Table 2. Beta Values of the 3 different models*

Given that product 'a' is the one being offered, the Cores, Frequency, and TDP values remain constant, with Price being the only variable. Consequently, a revenue function has been defined, which is based on the equation presented in Fig.8.

```
x_a = [4, 3.2, 95] # from the problem description
a = np.dot(beta[:3], x_a) #beta is the Beta we found for features
b = beta[3]

# It is (a+b*p) as the b is negative value in our case
def R(p):
    return p*np.exp(a+b*p)/(np.exp(a+b*p)+1)
```

*Figure 8. Revenue function code.*

Utilizing the defined revenue function, the changes in revenue with respect to price have been plotted for all three sets of Beta values. An illustration of the graph based on the Beta values obtained from Biogeme (All Assortment) version which is provided below (see Fig.9).

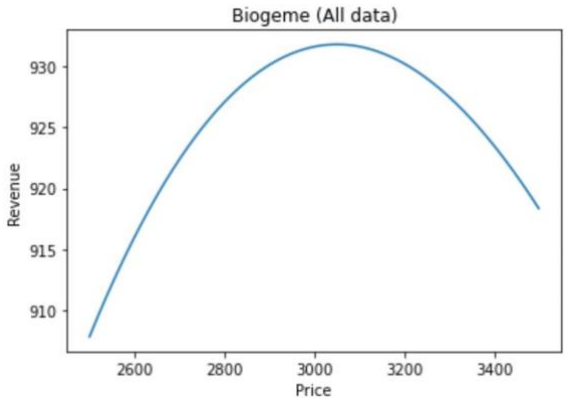


*Figure 9. An illustration of the Revenue V.S. Price graph based on the Beta value of all dataset.*

Lastly, an optimization algorithm is conducted by using the 'fmin' function to find the optimal price that corresponds to the optimal revenue (see Fig.10).

```
# Finding of the optimal Price
from scipy.optimize import fmin
p_star = fmin(lambda x: -R(x), 0)[0]
R_star = R(p_star)
f"Maximum expected revenue of %f is acheived with price %f" % (R_star, p_star)
```

*Figure 10. Opimization algorithum used for finding of optimal price.*

With this algorithm, the 3 optimal price being found is shown below (see Table 3).

| Biogeme (All Assortment) | Biogeme (1 Product Assortment) | Loglikelihood |
|---|---|---|
| 3050.477750 | 3358.240438 | 3014.937375 |

*Table 3. Optimal Price for the 3 models based on Minimization Objective Algorithun.*

### 3.2. Final Selection of Optimal Price

In conclusion, we have validated the optimal prices by inputting the four different price points obtained under various Beta value scenarios. As a result, three optimal prices have been chosen for this section, as shown in the Table 4 below. These selected prices align with the output results from Section 3.1.

| Biogeme (All Assortment) | Biogeme (1 Product Assortment) | Loglikelihood |
|---|---|---|
| 3050.477750 | 3358.240438 | 3014.937375 |

*Table 4. Final optimal Price choosen for this section.*

## 4. Part (c). Revenue-optimized Assortment

### 4.1. Methodology

In this section, we present an assortment optimization problem with a limited capacity of three products in a store. Let the prices for the stock-keeping units (SKUs) a, b, c, d, e, and f be set at 2700, 2400, 2700, 2100, 2400, and 2400, respectively. The objective is to determine the optimal assortment of at most three SKUs that maximizes revenue.

The capacity-constrained assortment optimization problem can be addressed by identifying the maximum possible profit generated by the combination of SKUs which give rise to the optimized assortment.

The profit maximization problem can be written as:

$$Z^* = \max_{S \subseteq N} \frac{\sum_{i \in S} w_i v_i}{1 + \sum_{i \in S} v_i} \quad s.t. |S| \leq C$$

Where $v_i = \exp(\mu_i)$ and $w_i = p_i - c_i$.

From section 2, we obtain two sets of *beta* values by using *Biogeme* (All Assortment) and *Logliklihood* method:

$$\beta_{Biogeme} = [0.3226564842308442, 0.4786270678113503,$$
$$-0.023199633313091498, -0.0004719818858727145 ]$$

$$\beta_{Logliklihood} = [0.3263116854084238, 0.49357456645769804,$$

$$-0.02355774017834474, -0.0004804459454350477 7]$$

The $v_i$ of product $i$ can be obtained by multiplying their feature values (see section 1.3) with one of the betas above. Since the marginal costs for the products are set to zero due to the high production volume, $w_i$ is equivalent to the price of product $i$ .

In order to identify the optimized assortment that yields the highest profit, we suggest two approaches. The first approach is the brute force method, applicable when dealing with a small number of products. This method involves evaluating all possible combinations of SKUs within the capacity constraint and selecting the one with the maximum profit.

The second approach is the graphical method, which is more appropriate for scenarios with a larger number of products. This method involves identify the variable $\lambda$ and function $(w_i - \lambda)v_i$ of the optimization problem for each product, plotting the function against the variable and identify the optimized assortment under the capacity constraints.

Both methods serve as practical tools for assortment optimization, with the choice of method depending on the scale and complexity of the problem.

## 4.2. Brute Force Method

After obtaining the $v_i \; and \; p_i$ for each SKU by using $\beta_{Logliklihood}$, we will then employ the function $\frac{\sum_{i \in S} w_i v_i}{1+\sum_{i \in S} v_i}$ to calculate the revenue generated by all possible combinations among the available SKUs. By evaluating this function for each potential assortment, we can determine the revenue associated with each combination and identify the optimal assortment that maximizes the revenue while adhering to the capacity constraints (see Fig.11 and Fig.12).

The python program:

```
def z(S): #function to calculate the revenue
    return np.dot(w[S],v[S])/(np.sum(v[S])+1)
```

```
# try all combinations
from itertools import combinations

z_list = [] #list of revenue
S_list = [] #list of possible assortmnet
Szall_list = []
for r in range(1, 4):
    for S in combinations(range(0,6), r):
        S = list(S)
        S_chr = [chr(ord('a')+n) for n in S]
        S_list.append(S_chr)
        z_list.append(z(S))
        Szall_list.append([S_chr, z(S)])
Szall_list
```

*Figure 11: Brutforce Method to Calculate Revenue of Each Assortment Combination*

Program output:

```
Out[19]: [[['a'], 925.7651125633203],
          [['b'], 1953.27700293793],
          [['c'], 1684.8691321461254],
          [['d'], 1213.6174044276013],
          [['e'], 1389.2963792868654],
          [['f'], 300.96094903960125],
          [['a', 'b'], 2019.3800444536955],
          [['a', 'c'], 1851.3539257328202],
          [['a', 'd'], 1481.890920473609],
          [['a', 'e'], 1625.4207108255962],
          [['a', 'f'], 1052.7054191287039],
          [['b', 'c'], 2129.5199263474638],
          [['b', 'd'], 1983.0754347048933],
          [['b', 'e'], 2044.2884664130927],
          [['b', 'f'], 1964.8892563512395],
          [['c', 'd'], 1825.9458245931205],
          [['c', 'e'], 1928.5291229431248],
          [['c', 'f'], 1721.4480618878436],
          [['d', 'e'], 1649.2169809805066],
          [['d', 'f'], 1281.3188401264908],
          [['e', 'f'], 1446.8488516003692],
          [['a', 'b', 'c'], 2168.9251208999563],
          [['a', 'b', 'd'], 2034.5771890700403],
          [['a', 'b', 'e'], 2091.3578414598664],
          [['a', 'b', 'f'], 2028.4189410400372],
          [['a', 'c', 'd'], 1926.1643676514084],
          [['a', 'c', 'e'], 2016.8806211057836],
          [['a', 'c', 'f'], 1875.0131706420664],
          [['a', 'd', 'e'], 1777.7539615265998],
          [['a', 'd', 'f'], 1525.273884082792],
          [['a', 'e', 'f'], 1661.956471394051],
          [['b', 'c', 'd'], 2124.7090458160887],
          [['b', 'c', 'e'], 2173.7457431666776],
          [['b', 'c', 'f'], 2134.924570740644],
          [['b', 'd', 'e'], 2053.686821856209],
          [['b', 'd', 'f'], 1991.7578805329388],
          [['b', 'e', 'f'], 2051.6903431743312],
          [['c', 'd', 'e'], 1971.9775700562534],
          [['c', 'd', 'f'], 1845.6730184813157],
          [['c', 'e', 'f'], 1944.7101187167336],
          [['d', 'e', 'f'], 1676.9102014675113]]
```

*Figure 12: Out Put for Brute Force Method with Beta_Logliklihood.*

The top 3 best assortment can be observed by sorting the output (see Fig.13 and Fig.14):

```
Sz_df = pd.DataFrame(Szall_list, columns=["Assortment", "Predicted Revenue"])
Sz_df.sort_values("Predicted Revenue", ascending=False, inplace=True)
Sz_df[:3] # top three predicted assortments for this beta
```

*Figure 13: Sorting the Output*

| | Assortment | Predicted Revenue |
|---|---|---|
| **32** | [b, c, e] | 2173.745743 |
| **21** | [a, b, c] | 2168.925121 |
| **33** | [b, c, f] | 2134.924571 |

*Figure 14: Output with beta_Logliklihood.*

This method is repeated by using $\beta_{Biogeme}$ to obtain a different set of output (see Fig.15):

| | Assortment | Predicted Revenue |
|---|---|---|
| **32** | [b, c, e] | 2167.666540 |
| **21** | [a, b, c] | 2162.713397 |
| **33** | [b, c, f] | 2127.951242 |

*Figure 15: Output with beta_Biogeme.*

The two outputs, generated using different β values, result in distinct predicted revenues. However, both methods yield the same optimal assortment.

## 4.3. Graphical Method

Using the $\beta_{Logliklihood}$ values, we calculate the $v_i$ for each product. By setting the $\lambda$ range from 0 to 3000, we can obtain the function value $(w_i - \lambda)v_i$ by substituting the $\lambda$. A graph can be plotted for $(w_i - \lambda)v_i$ against $\lambda$, as demonstrated below (see Fig.16):

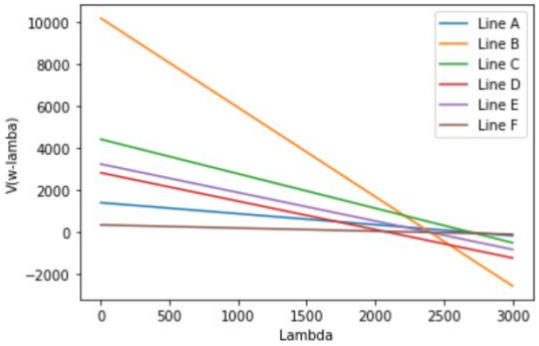


*Figure 16: Graphical Method.*

There are 12 intersections points on the graph. The order of product for each intersection point is determined, this represents the optimal assortment at different intersection points. The function $\frac{\sum_{i\in S} w_i v_i}{1+\sum_{i\in S} v_i}$ is used to calculate the revenue for different assortment at various $\lambda$ value. Based on the revenue calculations, the top three assortments are identified as the most optimal choices that maximize revenue while adhering to the capacity constraints.

The top 3 assortment are as such (see Fig.17):

| | assortment | Z |
|---|---|---|
| **2** | [b, c, e] | 2173.75 |
| **0** | [a, b, c] | 2168.93 |
| **1** | [a, c, f] | 1875.01 |

*Figure 17: Output of Graphical Method wth beta_Logliklihood.*

A different set of assortment is calculated by using $\beta_{biogeme}$, the result is as such (see Fig.18):

| | assortment | Z |
|---|---|---|
| **2** | [b, c, e] | 2167.666540 |
| **7** | [c, a, b] | 2162.713397 |
| **1** | [c, a, f] | 1868.303170 |

*Figure 18: Output of Graphical Method wth beta_Biogeme.*

## 4.4. Final Selection of Revenue-maximizing Assortment

After applying both the brute force method and the graphical method, we have obtained four sets of results. These results can be presented in the Table 5 below:

| Method | Assortment | Revenue |
|---|---|---|
| Brute Force $\beta_{Biogeme}$ | [b,c,e] | 2167.67 |
| | [a,b,c] | 2162.71 |
| | [b,c,f] | 2127.95 |
| Brute Force $\beta_{Logliklihood}$ | [b,c,e] | 2173.75 |
| | [a,b,c] | 2168.93 |
| | [b,c,f] | 2134.92 |
| Graphic $\beta_{Biogeme}$ | [b,c,e] | 2167.67 |
| | [a,b,c] | 2162.71 |
| | [a,c,f] | 1868.30 |
| Graphic $\beta_{Logliklihood}$ | [b,c,e] | 2173.75 |
| | [a,b,c] | 2168.93 |
| | [a,c,f] | 1875.01 |

*Table 5: Assortment and Revenue.*

By comparing the total revenue of different assortments under different β values, we have selected the top 3 best-performing assortments. They are **[b,c,e]**, **[a,b,c]**, and **[b,c,f].** These optimal assortments maximize revenue while adhering to the capacity constraints, making them ideal choices in the given scenario. The relevant program can be found in '*Part(c).ipynb'*.

# Reference

[1] *About, Biogeme 3.11*. Available at: https://biogeme.epfl.ch/ (Accessed: April 15, 2023).