Choice Modeling, Assortment Optimization and Pricing

May 2, 2023

Final Project | ORIE 5132

Cornell Tech, SP 2023

Ting-Wei Lu (tl574), Sakshi Mittal (sa875), Renata Anastasia (ra568)

[ ]:
```
!gdown "1tNtCfUbO8BFGbB9kIOKymUfYIY_vjL7Q"
!tar xzvf project.tar
```

```
Downloading...
From: https://drive.google.com/uc?id=1tNtCfUbO8BFGbB9kIOKymUfYIY_vjL7Q
To: /content/project.tar
100% 1.40M/1.40M [00:00<00:00, 117MB/s]
Project.pdf
data.csv
data1.csv
data2.csv
data3.csv
data4.csv
```

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from scipy.optimize import minimize
import warnings
from tqdm import tqdm
warnings.filterwarnings('ignore')
pd.set_option('display.max_colwidth',1000)
```

[ ]:
```
data = pd.read_csv('data.csv')
columns = [col for col in data.columns if col.startswith('p')]
data.head()
```

[ ]:
```
   srch_id  prop_starrating  prop_review_score  prop_brand_bool  \
0        1                4                  3                1
1        1                3                  5                1
2        1                4                  4                1
3        1                4                  3                1
```

```
4        1                4                    4                  1

   prop_location_score  prop_accesibility_score  prop_log_historical_price  \
0                    2                        0                          5
1                    2                        0                          5
2                    3                        0                          5
3                    3                        0                          5
4                    2                        0                          5

   price_usd  promotion_flag  srch_booking_window  srch_adults_count  \
0        140               0                    0                  4
1        211               0                    0                  4
2        150               0                    0                  4
3        144               0                    0                  4
4        191               0                    0                  4

   srch_children_count  srch_room_count  srch_saturday_night_bool  \
0                    0                1                         1
1                    0                1                         1
2                    0                1                         1
3                    0                1                         1
4                    0                1                         1

   booking_bool
0             0
1             0
2             0
3             0
4             0
```

1 Problem 1: MNL Model

Question: Estimate the parameters $\beta_i, \forall i = 1, .., 8$ using MLE estimation. Comment on the coefficient of each of the features.

- v_j: preference weight of hotel j
- x_{ji}: feature i of hotel $j, \forall i = 1, .., 8$
- β_i is the sensitivity of customer to feature i
- Probability of customer choosing hotel j given a set hotels S $P(j|S) = \frac{v_j}{1+\sum_{p \in S} v_p}$

$$u(j) = \beta_0 + \sum_{i=1}^{8} \beta_i x_{ij} \tag{1}$$

$$v_j = e^{u(j)} \tag{2}$$

$$\mathbb{P}(j|S) = \frac{v_j}{1 + \sum_{i \in S} v_i} \tag{3}$$

$$L = \sum_{t=1}^{T} log\mathbb{P}(j_t|S_t) \tag{4}$$

$$= \sum_{t=1}^{T} u(j_t) - log(1 + \sum_{i \in S_t} e^{u(j_t)})) \tag{5}$$

$$= \sum_{t=1}^{T} (\beta_0 + \sum_{i=1}^{8} \beta_i x_{ij_t}) - log(1 + \sum_{l \in S_t} e^{\beta_0 + \sum_{i=1}^{8} \beta_i x_{li}})) \tag{6}$$

[ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(data[columns])
```

[ ]:
```
assortments = data.groupby('srch_id')[columns].apply(lambda g: g.values.
 ↪tolist()).tolist()
assortments = [scaler.transform(assortment) for assortment in assortments] #␣
 ↪Using the same scaler
assortments = [np.concatenate([np.ones((len(assortment), 1)), assortment],␣
 ↪axis=1) for assortment in assortments]
bookings = data.groupby('srch_id')['booking_bool'].apply(lambda g: g.values.
 ↪tolist()).tolist()
bookings = [np.array(booking) for booking in bookings]
```

[ ]:
```
def log_likelihood(coefs, assortment, booking):
    utility = (assortment * coefs).sum(axis=1)
    shift = np.max(utility)
    return (utility * booking).sum(axis=0) - shift - np.log(np.exp(-shift) + np.
 ↪exp(utility-shift).sum(axis=0))
def nll_loss(coefs, assortments, bookings):
    return -sum([log_likelihood(coefs, assortment, booking) for assortment,␣
 ↪booking in zip(assortments, bookings)])
res = minimize(nll_loss, np.zeros(9), args=(assortments, bookings),␣
 ↪method='Powell', options={'disp': True})
```

```
Optimization terminated successfully.
         Current function value: 20611.364103
         Iterations: 4
         Function evaluations: 364
```

[ ]:
```
p1_res = pd.DataFrame({'Features': ['intercept'] + columns,
                       'Coefs': res.x})
p1_res
```

[ ]:

	Features	Coefs
0	intercept	-1.746314
1	prop_starrating	0.412161
2	prop_review_score	0.105785
3	prop_brand_bool	0.100825
4	prop_location_score	0.020200
5	prop_accesibility_score	0.043412
6	prop_log_historical_price	-0.069802
7	price_usd	-1.331183
8	promotion_flag	0.159481

Based on the estimated β_i, the features that give the most positive impact on probability is star rating. Higher star rating will lead to higher probability. Accessibility and promotion also give positive impact although smaller.On the other hand, negative coefficient is found at price that suggest higher price will significantly decrease probability of booking. Furthermore, brand, location, review, and historical price give negative coefficient although small impact.

2 Problem 2: Assortment Optimization under MNL

Assume customers make choices according to the MNL model we estimated in Problem 1.

Given the set of hotels in data1.csv, suppose you want to show a subset of these hotels to the customers, what is the optimal subset of hotels to display? Give the expected revenue under this optimal assortment. Repeat the same question for data2.csv, data3.csv and data4.csv

[ ]:
```
data1 = pd.read_csv('data1.csv')
data1.head()
```

[ ]:

	prop_starrating	prop_review_score	prop_brand_bool	prop_location_score	\
0	3	4	1	0	
1	3	5	1	1	
2	3	5	1	1	
3	4	5	1	0	
4	3	5	1	0	

	prop_accesibility_score	prop_log_historical_price	price_usd	\
0	0	5	150	
1	0	5	140	
2	0	5	145	
3	0	5	125	
4	0	5	154	

	promotion_flag
0	0

```
1             0
2             0
3             0
4             0
```

```
[ ]: def revenue(features, param, real_prices):
         utility = (features * param).sum(axis=1)
         shift = utility.max()
         preference_weight = np.exp(utility - shift)
         prob_purchase = preference_weight / (np.exp(-shift) + preference_weight.
      ↪sum(axis=0))
         rev = (prob_purchase * real_prices).sum(axis=0)
         return rev
```

Theorem: suppose $p_1 \geq p_2 \geq ... \geq p_n$, the optimal assortment is of the form $1, 2, .., j$ for some $j = 1, 2, .., n$ which referred to "nested by price" assortment

```
[ ]: p2_res = {'Dataset': [], 'Assortment': [], 'Max Revenue': []}
     for x in range(1, 5):
         data_x = pd.read_csv(f'data{x}.csv')
         data_x = data_x.sort_values(by='price_usd', ascending=False)
         data_x_scaled= scaler.transform(data_x) # Using the same scaler
         data_x_scaled = np.concatenate([np.ones((len(data_x_scaled), 1)),␣
      ↪data_x_scaled], axis=1)
         best_assortment, max_revenue = None, -1
         for n in range(1, len(data_x)):
           param = p1_res['Coefs'].values
           prices = data_x['price_usd'].iloc[:n]
           rev = revenue(data_x_scaled[:n], param, prices)
           if rev > max_revenue:
               best_assortment, max_revenue = data_x.iloc[:n].index, rev
         p2_res['Dataset'].append(f'data{x}')
         p2_res['Assortment'].append(best_assortment.tolist())
         p2_res['Max Revenue'].append(max_revenue)
     p2_res['Assortment'] = [sorted(x) for x in p2_res['Assortment']]
     pd.DataFrame(p2_res)
```

```
[ ]:  Dataset                                                 Assortment  \
     0   data1  [0, 1, 2, 3, 4, 5, 6, 12, 15, 17, 18, 19, 20, 21, 22, 23, 24, 26]
     1   data2                              [0, 1, 6, 7, 8, 9, 10, 21, 23, 25]
     2   data3   [0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 13, 14, 15, 16, 18, 19, 23, 24]
     3   data4                          [3, 4, 6, 8, 10, 15, 18, 19, 20, 21, 26]

        Max Revenue
     0   107.345894
     1   131.321571
     2   121.065894
```

3   97.619354

3 Problem 3: Pricing under MNL

Assume customers make choices according to the MNL model we estimated in Problem 1. Consider the set of hotels in data1.csv. Suppose the company will display all these hotels to customers.

The company wants to change the price of each of these hotels (column price usd in the data). What are the optimal hotel prices that maximizes the expected revenue under MNL model given that we display all of them. Repeat the same question for data2.csv, data3.csv and data4.csv

[ ]:
```
p3_res = {'Dataset': [], 'Prices': []}
def revenue_loss(prices, data_x_):
      # Replacing prices value in dataset and add column "1" for intercept
      data_x_['price_usd'] = prices
      data_x_ = scaler.transform(data_x_.values)
      data_x_ = np.concatenate([np.ones((len(data_x_), 1)), data_x_], axis=1)
      param= p1_res['Coefs'].values
      rev = revenue(data_x_, param, prices)
      return -rev

for x in range(1, 5):
    data_x = pd.read_csv(f'data{x}.csv')
    res = minimize(revenue_loss, np.zeros(len(data_x)), args=(data_x),␣
 ↪method='Powell')
    p3_res['Dataset'].append(f'data{x}')
    p3_res['Prices'].append(res.x)
```

[ ]:
```
pd3_res= pd.DataFrame(p3_res)
pd3_res['Prices'] = pd3_res['Prices'].apply(lambda x: [int(round(price)) for␣
 ↪price in x]).apply(lambda x: sum(x) / len(x))
pd3_res
```

[ ]:

	Dataset	Prices
0	data1	314.0
1	data2	386.0
2	data3	313.0
3	data4	352.0

4 Problem 4: Mixture of MNL

Two types of customer: 1. Early customer: booking window more or equal to 7 days (probability θ_1) 2. Late customer: booking window less than 7 days (probability θ_2)

Estimate $\vartheta 1$ and $\vartheta 2$ by computing the size of customers of each type

[ ]:
```
theta_1= len(data[data['srch_booking_window'] >= 7]['srch_id'].unique()) /␣
↪len(data['srch_id'].unique())
theta_2= len(data[data['srch_booking_window'] < 7]['srch_id'].unique()) /␣
↪len(data['srch_id'].unique())
print(f"theta_1 =", theta_1)
print(f"theta_2 =", theta_2)
```

```
theta_1 = 0.5430931290399809
theta_2 = 0.45690687096001914
```

[ ]:
```
columns = [col for col in data.columns if col.startswith('p')]
```

Estimate an MNL model for each type and estimate the sensitivity parameters for each type of customers using MLE estimation

$$u(j_k) = \beta_{0_k} + \sum_{i=1}^{8} \beta_{i_k} x_{ij} \tag{7}$$

$$v_{j_k} = e^{u(j_k)} \tag{8}$$

$$\mathbb{P}(j|S) = \sum_{k=1}^{2} \theta_k \frac{v_{j_k}}{1 + \sum_{i \in S} v_{i_k}} \tag{9}$$

$$L = \sum_{t=1}^{T} log\mathbb{P}(j_t|S_t) \tag{10}$$

$$= \sum_{k=1}^{2} \theta_k \sum_{t=1}^{T_k} log\mathbb{P}(j_t|S_t) \tag{11}$$

$$= \sum_{k=1}^{2} \theta_k (\sum_{t=1}^{T_k} u(j_t) - log(1 + \sum_{i \in S_t} e^{u(j_t)}))) \tag{12}$$

$$= \sum_{k=1}^{2} \theta_k (\sum_{t=1}^{T_k} (\beta_{0_k} + \sum_{i=1}^{8} \beta_{i_k} x_{ij_t}) - log(1 + \sum_{l \in S_t} e^{\beta_{0_k} + \sum_{i=1}^{8} \beta_{i_k} x_{li}})) \tag{13}$$

[ ]:
```
columns = [col for col in data.columns if col.startswith('p')]
assortments_early = data[data['srch_booking_window'] >= 7].
↪groupby('srch_id')[columns].apply(lambda g: g.values.tolist()).tolist()
assortments_early = [np.array(assortment) for assortment in assortments_early]
assortments_early = [scaler.transform(assortment) for assortment in␣
↪assortments_early]
assortments_early = [np.concatenate([np.ones((len(assortment), 1)), assortment],␣
↪axis=1) for assortment in assortments_early]
bookings_early = data[data['srch_booking_window'] >= 7].
↪groupby('srch_id')['booking_bool'].apply(lambda g: g.values.tolist()).tolist()
bookings_early = [np.array(booking) for booking in bookings_early]
```

```
assortments_late = data[data['srch_booking_window'] < 7].
↪groupby('srch_id')[columns].apply(lambda g: g.values.tolist()).tolist()
assortments_late = [np.array(assortment) for assortment in assortments_late]
assortments_late = [scaler.transform(assortment) for assortment in␣
↪assortments_late]
assortments_late = [np.concatenate([np.ones((len(assortment), 1)), assortment],␣
↪axis=1) for assortment in assortments_late]
bookings_late = data[data['srch_booking_window'] < 7].
↪groupby('srch_id')['booking_bool'].apply(lambda g: g.values.tolist()).tolist()
bookings_late = [np.array(booking) for booking in bookings_late]

def log_likelihood(param, assortment, booking):
    utility = (assortment * param).sum(axis=1) #u_j= beta_i * x_ij
    shift = np.max(utility)
    exp = np.exp(utility - shift) #scaler to avoid overflow since we're going to␣
↪compute np.exp(utility)
    if 1 in booking:
        index = np.where(booking == 1)[0][0]
        prob = exp[index] / (np.exp(-shift) + np.sum(exp))
    else:
        prob = np.exp(-shift) / (np.exp(-shift) + np.sum(exp))
    return np.log(prob)

def sum_log(param, x, y):
    logL = 0
    for assortment, booking in zip(x, y):
        logL += log_likelihood(param, assortment, booking)
    return -logL

def sum_log_early_late(param1, param2, x1, x2, y1, y2, theta1, theta2):
  sum_logL= (theta1 * sum_log(param1, x1, y1)) + (theta2 * sum_log(param2,x2,y2))
  return sum_logL
```

[ ]:
```
param_early_initial = np.zeros(9)
param_late_initial = np.zeros(9)
initial_guess = np.concatenate([param_early_initial, param_late_initial])
result = minimize(lambda params: sum_log_early_late(params[:9], params[9:],␣
↪assortments_early, assortments_late, bookings_early, bookings_late, theta_1,␣
↪theta_2), x0=initial_guess, method='Powell')
```

[ ]:
```
beta_hat_early= result.x[:9]
beta_hat_late= result.x[9:]
columns = ['coef_' + col for col in data.columns if col.startswith('p')]
df_p4 = pd.DataFrame([['Early'] + beta_hat_early.tolist(), ['Late'] +␣
↪beta_hat_late.tolist()], columns=['Type', 'Intercept'] + columns)
df_p4
```

```
[ ]:    Type  Intercept  coef_prop_starrating  coef_prop_review_score  \
0  Early  -1.918206              0.383180                0.122693
1   Late  -1.538378              0.466183                0.090930

   coef_prop_brand_bool  coef_prop_location_score  \
0              0.091707                 -0.022468
1              0.111943                  0.084050

   coef_prop_accesibility_score  coef_prop_log_historical_price  \
0                      0.057837                       -0.096866
1                      0.025509                       -0.036682

   coef_price_usd  coef_promotion_flag
0       -1.073844             0.134395
1       -1.691385             0.193909
```

For early customer, the most influential features are star rating, promotions of the property and review score that have positive coefficient - suggests that higher score will lead to higher probability. Other than these, coefficient for other features such as brand and accessibility also have positive coefficient albeit smaller impact and this is probably due to imbalanced dataset. On the other hand, price significantly have negative effect where higher price resulted in decreasing the probablity of booking. Other coffiecients such as historical price and location score also have negative coefficient although smaller impact.

Meanwhile for late customer, the most influential features are star rating, promotion, and brand of the property. Higher value will increase the probability of last-minute booking. One possible explanation is last-minute customer might have limited time in booking the property hence they will look at star rating to see if the property is reliable. Furthermore, hotel brand is also important for last-minute customer where brand with high reputation is usually more trustable. Other coefficient like accessibility, review, and location also have positive coefficient but smaller impact. Coefficient for price is negative that suggest high price will decrease probability of last-minute booking. Historical price also has negative coefficient although smaller impact.

5 Problem 5: Early vs Late Reservations

Assume customers make choices according to the mixture of MNL model we estimated in Problem 4. Given the set of hotels in data1.csv, suppose you want to show a subset of these hotels to the customers that maximizes the revenue of the company. * Assume we don't know the type of an arriving customer. What is the optimal subset of hotels to display? Let's call it S. You need to solve an integer program here to compute S * Suppose we know that the arriving customer is of type 1. What is the optimal subset of hotels to display? Let's call it S1 * Suppose we know that the arriving customer is of type 2. What is the optimal subset of hotels to display? Let's call it S2

To find the optimal subset of hotels to display when we don't know the type of arriving customers, we'll have to use IP to solve the MMNL.

Objective function:

$$\text{Maximize} \sum_{k=1}^{2} \theta_k z_k \tag{14}$$

Constraint:

$$\begin{aligned}
z_k &\leq \sum_{i=1}^{n} x_{ik} v_{i_k} && \forall k && (15)\\
-My_i \leq\ & x_{ik} \leq My_i, && \forall i, \forall k && (16)\\
(p_i - z_k) - M(1-y_i) \leq\ & x_{ik} \leq (p_i - z_k) + M(1-y_i) && \forall i, \forall k && (17)\\
y_i &\in \{0,1\} && \forall i && (18)\\
& && && (19)
\end{aligned}$$

Data:

- θ_k: proportion of customer type- k
- p_i: price of hotel-i
- v_{i_k}: preference weight of hotel-i for customer type- k
- a_{mi_k}: value of feature-m for hotel-i for customer type-k

Decision variables:

- y_i: indicator if hotel-i is included in the assortment

Note:

- $u(i_k) = \beta_{0_k} + \sum_{m=1}^{8} \beta_{m_k} a_{mi_k}$
- $v_{i_k} = e^{u(i_k)}$

[ ]:
```
%pip install gurobipy
from gurobipy import *
import pandas as pd
import math
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting gurobipy
  Downloading gurobipy-10.0.1-cp310-cp310-manylinux2014_x86_64.whl (12.7 MB)
      12.7/12.7 MB
64.2 MB/s eta 0:00:00
Installing collected packages: gurobipy
Successfully installed gurobipy-10.0.1
```

[ ]:
```
def optimal_assortment(data, beta, theta, K_type):
  myModel= Model('Mixture MNL')

  n= len(data)
  K= K_type
  M= 100000000
```

```
u= np.zeros((n,K))
v= np.zeros((n,K))
data_norm= scaler.transform(data.values)

for k in range(K):
  for i in range(n):
    u[i,k]= beta[k][0] + np.sum(beta[k][1:] * data_norm[i])
    v[i,k]= np.exp(u[i,k])

#Variables
y = myModel.addVars(n, vtype=GRB.BINARY, name="y")
z= myModel.addVars(K, vtype=GRB.CONTINUOUS, name='z')
x= myModel.addVars(n,K, vtype=GRB.CONTINUOUS, name='x')

#Objective function
objExpr= LinExpr()
for k in range(K):
  objExpr += theta[k] * z[k]
myModel.setObjective(objExpr, GRB.MAXIMIZE)
myModel.update()

#Constraint 1
for k in range(K):
  rhs1= 0
  for i in range(n):
    rhs1 += x[i,k] * v[i,k]
  myModel.addConstr(lhs= z[k], sense= GRB.LESS_EQUAL, rhs=rhs1)
  myModel.update()

#Constraint 2
for k in range(K):
  for i in range(n):
    myModel.addConstr(lhs= -M* y[i], sense= GRB.LESS_EQUAL, rhs= x[i,k])
    myModel.addConstr(lhs= x[i,k], sense= GRB.LESS_EQUAL, rhs= M* y[i])
    myModel.update()

#Constraint 3
for k in range(K):
  for i in range(n):
    myModel.addConstr(lhs= (data.iloc[i]['price_usd'] - z[k]) - M*(1-y[i]),␣
↪sense= GRB.LESS_EQUAL, rhs= x[i,k])
    myModel.addConstr(lhs= x[i,k], sense= GRB.LESS_EQUAL, rhs= (data.
↪iloc[i]['price_usd'] - z[k]) + M*(1-y[i]))
    myModel.update()

myModel.optimize()
```

```
    assortment = []
    for i in range(n):
        if y[i].x > 0.5:
            assortment.append(i+1)
    assortment= np.array(assortment).tolist()

    return assortment, myModel.objVal
```

```
[ ]: beta= [beta_hat_early, beta_hat_late]
theta= [theta_1, theta_2]

p5_res = {'Dataset': [], 'Assortment with Unknown Type': [], 'Expected Revenue␣
↪with Unknown Type':[],
          'Assortment for Early Cust': [], 'Expected Revenue for Early Cust':[],
          'Assortment for Late Cust': [], 'Expected Revenue for Late Cust':[]}
for x in range(1, 5):
    data_x = pd.read_csv(f'data{x}.csv')
    p5_res['Dataset'].append(f'data{x}')

    #S
    opt_assortment, rev= optimal_assortment(data_x, beta, theta, 2)
    p5_res['Assortment with Unknown Type'].append(opt_assortment)
    p5_res['Expected Revenue with Unknown Type'].append(rev)

    #S1 and S2
    data_x = data_x.sort_values(by='price_usd', ascending=False)
    data_x_norm= scaler.transform(data_x.values)
    data_x_scaled = np.concatenate([np.ones((len(data_x_norm), 1)),␣
↪data_x_norm], axis=1)
    best_assortment_early, max_revenue_early = None, -1
    best_assortment_late, max_revenue_late = None, -1

    for n in range(1, len(data_x_scaled)):
        prices = data_x['price_usd'].iloc[:n]
        rev_early = revenue(data_x_scaled[:n], beta[0], prices)
        if rev_early > max_revenue_early:
            best_assortment_early, max_revenue_early = data_x.iloc[:n].index,␣
↪rev_early

        rev_late = revenue(data_x_scaled[:n], beta[1], prices)
        if rev_late > max_revenue_late:
            best_assortment_late, max_revenue_late = data_x.iloc[:n].index,␣
↪rev_late

    p5_res['Assortment for Early Cust'].append(best_assortment_early.tolist())
    p5_res['Assortment for Early Cust'] = [sorted(x) for x in p5_res['Assortment␣
↪for Early Cust']]
```

```
    p5_res['Expected Revenue for Early Cust'].append(max_revenue_early)

    p5_res['Assortment for Late Cust'].append(best_assortment_late.tolist())
    p5_res['Assortment for Late Cust'] = [sorted(x) for x in p5_res['Assortment␣
↪for Late Cust']]
    p5_res['Expected Revenue for Late Cust'].append(max_revenue_late)
```

```
Restricted license - for non-production use only - expires 2024-10-28
Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (linux64)

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 218 rows, 83 columns and 596 nonzeros
Model fingerprint: 0xbcf42463
Variable types: 56 continuous, 27 integer (27 binary)
Coefficient statistics:
  Matrix range     [8e-02, 1e+08]
  Objective range  [5e-01, 5e-01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+08, 1e+08]
Found heuristic solution: objective -0.0000000
Presolve removed 54 rows and 0 columns
Presolve time: 0.00s
Presolved: 164 rows, 83 columns, 488 nonzeros
Variable types: 56 continuous, 27 integer (27 binary)

Root relaxation: objective 3.158544e+02, 123 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  315.85440    0   27   -0.00000  315.85440      -      -    0s
H    0     0                      34.3060950  315.85440   821%     -    0s
H    0     0                      63.9500648  315.85440   394%     -    0s
     0     0  235.21261    0   27   63.95006  235.21261   268%     -    0s
H    0     0                      95.0000000  235.21261   148%     -    0s
H    0     0                     104.1661054  235.21261   126%     -    0s
     0     0  215.89789    0   21  104.16611  215.89789   107%     -    0s
     0     0  215.69524    0   21  104.16611  215.69524   107%     -    0s
     0     0  208.16826    0   21  104.16611  208.16826   100%     -    0s
     0     0  207.37556    0   21  104.16611  207.37556  99.1%     -    0s
     0     0  204.82223    0   21  104.16611  204.82223  96.6%     -    0s
     0     0  204.29493    0   21  104.16611  204.29493  96.1%     -    0s
     0     0  204.10180    0   21  104.16611  204.10180  95.9%     -    0s
     0     0  204.08495    0   21  104.16611  204.08495  95.9%     -    0s
     0     0  203.26060    0   21  104.16611  203.26060  95.1%     -    0s
```

```
     0     0  202.87856    0   21  104.16611  202.87856  94.8%     -    0s
     0     0  202.77925    0   21  104.16611  202.77925  94.7%     -    0s
     0     0  202.46741    0   21  104.16611  202.46741  94.4%     -    0s
     0     0  197.21557    0   21  104.16611  197.21557  89.3%     -    0s
     0     0  145.42732    0   21  104.16611  145.42732  39.6%     -    0s
     0     0  125.26907    0   21  104.16611  125.26907  20.3%     -    0s
H    0     0                     106.3996659  125.26907  17.7%     -    0s
     0     0  120.31963    0   20  106.39967  120.31963  13.1%     -    0s
     0     0  118.78998    0   20  106.39967  118.78998  11.6%     -    0s
     0     0  118.67266    0   20  106.39967  118.67266  11.5%     -    0s
     0     0  118.65821    0   20  106.39967  118.65821  11.5%     -    0s
     0     0  118.00719    0   20  106.39967  118.00719  10.9%     -    0s
     0     0  117.83012    0   20  106.39967  117.83012  10.7%     -    0s
     0     0  117.78841    0   20  106.39967  117.78841  10.7%     -    0s
     0     0  117.74659    0   20  106.39967  117.74659  10.7%     -    0s
     0     0  117.74508    0   20  106.39967  117.74508  10.7%     -    0s
     0     0  117.71995    0   20  106.39967  117.71995  10.6%     -    0s
     0     0  117.70598    0   20  106.39967  117.70598  10.6%     -    0s
     0     0  117.62978    0   20  106.39967  117.62978  10.6%     -    0s
     0     0  117.53092    0   20  106.39967  117.53092  10.5%     -    0s
H    0     0                     106.9397071  117.53092  9.90%     -    0s
     0     2  117.36974    0   20  106.93971  117.36974  9.75%     -    0s
*  205     1              17     107.1935860  107.20863  0.01%   6.8    0s

Cutting planes:
  Gomory: 16
  Implied bound: 16
  MIR: 53
  Flow cover: 13

Explored 207 nodes (2067 simplex iterations) in 0.53 seconds (0.18 work units)
Thread count was 2 (of 2 available processors)

Solution count 8: 107.194 106.94 106.4 ... -0

Optimal solution found (tolerance 1.00e-04)
Best objective 1.071935860372e+02, best bound 1.071935860372e+02, gap 0.0000%
Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (linux64)

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 234 rows, 89 columns and 640 nonzeros
Model fingerprint: 0x022f6cf8
Variable types: 60 continuous, 29 integer (29 binary)
Coefficient statistics:
  Matrix range     [2e-01, 1e+08]
  Objective range  [5e-01, 5e-01]
```

```
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+08, 1e+08]
Found heuristic solution: objective -0.0000000
Presolve removed 58 rows and 0 columns
Presolve time: 0.00s
Presolved: 176 rows, 89 columns, 524 nonzeros
Variable types: 60 continuous, 29 integer (29 binary)

Root relaxation: objective 7.586619e+02, 127 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  758.66193    0   29   -0.00000  758.66193      -     -    0s
H    0     0                     111.1610128  758.66193   582%     -    0s
     0     0  482.41442    0   16  111.16101  482.41442   334%     -    0s
     0     0  474.68719    0   16  111.16101  474.68719   327%     -    0s
H    0     0                     120.1758892  474.68719   295%     -    0s
     0     0  376.31949    0   14  120.17589  376.31949   213%     -    0s
H    0     0                     123.0000000  376.31949   206%     -    0s
     0     0  308.93359    0   10  123.00000  308.93359   151%     -    0s
     0     0  307.02171    0   10  123.00000  307.02171   150%     -    0s
     0     0  297.22335    0   11  123.00000  297.22335   142%     -    0s
H    0     0                     124.1014720  297.22335   140%     -    0s
     0     0  296.31586    0   11  124.10147  296.31586   139%     -    0s
     0     0  294.87858    0   11  124.10147  294.87858   138%     -    0s
     0     0  294.56666    0   11  124.10147  294.56666   137%     -    0s
     0     0  294.32510    0   11  124.10147  294.32510   137%     -    0s
     0     0  294.02964    0   11  124.10147  294.02964   137%     -    0s
     0     0  293.89629    0   11  124.10147  293.89629   137%     -    0s
     0     0  293.66859    0   11  124.10147  293.66859   137%     -    0s
     0     0  285.72929    0   11  124.10147  285.72929   130%     -    0s
H    0     0                     130.9062348  285.72929   118%     -    0s
     0     0  274.42218    0   10  130.90623  274.42218   110%     -    0s
     0     0  274.24028    0   10  130.90623  274.24028   109%     -    0s
     0     0  274.16845    0   10  130.90623  274.16845   109%     -    0s
     0     0  259.80789    0   10  130.90623  259.80789  98.5%     -    0s
     0     0  258.62419    0   10  130.90623  258.62419  97.6%     -    0s
     0     0  258.47583    0   10  130.90623  258.47583  97.5%     -    0s
     0     0  258.46490    0   10  130.90623  258.46490  97.4%     -    0s
     0     0  248.73710    0   10  130.90623  248.73710  90.0%     -    0s
     0     0  248.07700    0   10  130.90623  248.07700  89.5%     -    0s
     0     0  247.91110    0   10  130.90623  247.91110  89.4%     -    0s
     0     0  247.88180    0   10  130.90623  247.88180  89.4%     -    0s
     0     0  243.33199    0   10  130.90623  243.33199  85.9%     -    0s
     0     0  243.06173    0   10  130.90623  243.06173  85.7%     -    0s
     0     0  243.00241    0   10  130.90623  243.00241  85.6%     -    0s
```

```
     0     0  242.41118    0   10  130.90623  242.41118  85.2%     -    0s
     0     0  205.43936    0   10  130.90623  205.43936  56.9%     -    0s
     0     0  142.00854    0    7  130.90623  142.00854  8.48%     -    0s
     0     0  136.52850    0    8  130.90623  136.52850  4.29%     -    0s
     0     0  135.91992    0    7  130.90623  135.91992  3.83%     -    0s
     0     0  135.25453    0    7  130.90623  135.25453  3.32%     -    0s
     0     0  132.59638    0    8  130.90623  132.59638  1.29%     -    0s
     0     0  132.56270    0    7  130.90623  132.56270  1.27%     -    0s
     0     0  131.45420    0    4  130.90623  131.45420  0.42%     -    0s
     0     0  131.44951    0    3  130.90623  131.44951  0.42%     -    0s
     0     0  131.28608    0    3  130.90623  131.28608  0.29%     -    0s
     0     0  131.22377    0    3  130.90623  131.22377  0.24%     -    0s
     0     0  131.22132    0    3  130.90623  131.22132  0.24%     -    0s
     0     0  131.21539    0    2  130.90623  131.21539  0.24%     -    0s
     0     0  131.17619    0    2  130.90623  131.17619  0.21%     -    0s
     0     0  131.17505    0    2  130.90623  131.17505  0.21%     -    0s
     0     0  131.17188    0    2  130.90623  131.17188  0.20%     -    0s

Cutting planes:
  MIR: 9
  Flow cover: 1
  Relax-and-lift: 3

Explored 1 nodes (664 simplex iterations) in 0.35 seconds (0.04 work units)
Thread count was 2 (of 2 available processors)

Solution count 6: 130.906 124.101 123 ... -0

Optimal solution found (tolerance 1.00e-04)
Best objective 1.309062347635e+02, best bound 1.309062347635e+02, gap 0.0000%
Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (linux64)

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 210 rows, 80 columns and 574 nonzeros
Model fingerprint: 0x1b0ed139
Variable types: 54 continuous, 26 integer (26 binary)
Coefficient statistics:
  Matrix range     [2e-03, 1e+08]
  Objective range  [5e-01, 5e-01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+08, 1e+08]
Found heuristic solution: objective -0.0000000
Presolve removed 52 rows and 0 columns
Presolve time: 0.00s
Presolved: 158 rows, 80 columns, 470 nonzeros
Variable types: 54 continuous, 26 integer (26 binary)
```

```
Root relaxation: objective 2.962427e+02, 110 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  296.24270    0   25   -0.00000  296.24270      -      -    0s
H    0     0                         2.3077931  296.24270      -      -    0s
H    0     0                        79.8638844  296.24270   271%      -    0s
     0     0  233.69817    0   25   79.86388  233.69817   193%      -    0s
H    0     0                       101.0000000  233.69817   131%      -    0s
H    0     0                       105.0000000  233.69817   123%      -    0s
H    0     0                       115.0000000  233.69817   103%      -    0s
H    0     0                       118.2282924  233.69817  97.7%      -    0s
     0     0  208.17982    0   18  118.22829  208.17982  76.1%      -    0s
     0     0  207.93340    0   19  118.22829  207.93340  75.9%      -    0s
     0     0  199.89157    0   18  118.22829  199.89157  69.1%      -    0s
H    0     0                       118.6499204  199.89157  68.5%      -    0s
     0     0  198.23064    0   18  118.64992  198.23064  67.1%      -    0s
     0     0  198.20837    0   18  118.64992  198.20837  67.1%      -    0s
     0     0  196.02527    0   18  118.64992  196.02527  65.2%      -    0s
     0     0  195.66601    0   18  118.64992  195.66601  64.9%      -    0s
     0     0  195.62677    0   18  118.64992  195.62677  64.9%      -    0s
     0     0  194.85027    0   18  118.64992  194.85027  64.2%      -    0s
     0     0  194.59151    0   18  118.64992  194.59151  64.0%      -    0s
     0     0  194.48594    0   18  118.64992  194.48594  63.9%      -    0s
     0     0  194.12413    0   18  118.64992  194.12413  63.6%      -    0s
     0     0  186.58410    0   18  118.64992  186.58410  57.3%      -    0s
     0     0  141.13871    0   17  118.64992  141.13871  19.0%      -    0s
     0     0  127.15925    0   17  118.64992  127.15925  7.17%      -    0s
     0     0  127.15848    0   17  118.64992  127.15848  7.17%      -    0s
     0     0  125.33436    0   15  118.64992  125.33436  5.63%      -    0s
     0     0  125.25193    0   15  118.64992  125.25193  5.56%      -    0s
     0     0  124.59925    0   14  118.64992  124.59925  5.01%      -    0s
     0     0  124.46880    0   14  118.64992  124.46880  4.90%      -    0s
     0     0  124.46206    0   14  118.64992  124.46206  4.90%      -    0s
     0     0  124.17254    0   11  118.64992  124.17254  4.65%      -    0s
     0     0  124.15674    0   10  118.64992  124.15674  4.64%      -    0s
     0     0  124.08031    0   10  118.64992  124.08031  4.58%      -    0s
     0     0  123.95976    0   10  118.64992  123.95976  4.48%      -    0s
     0     2  123.92428    0   10  118.64992  123.92428  4.45%      -    0s
H    6     1                       118.9355885  123.42589  3.78%    5.0    0s
H    7     2                       118.9734895  123.42589  3.74%    5.0    0s
*   34     7              10     119.0977065  122.27349  2.67%    3.4    0s
*   46     8              10     119.0998358  122.14813  2.56%    2.9    0s
H   54     7                       119.2988117  121.89326  2.17%    3.4    0s
*   62     9              10     119.7025378  121.73434  1.70%    3.3    0s
```

```
*   67     6              10     120.4125476  121.73434  1.10%   3.1    0s
*   73     5               9     120.6121022  121.61095  0.83%   3.0    0s
*   78     3               9     120.9342355  121.61095  0.56%   2.9    0s

Cutting planes:
  Gomory: 6
  Implied bound: 9
  MIR: 32
  Flow cover: 13
  Relax-and-lift: 8

Explored 83 nodes (785 simplex iterations) in 0.45 seconds (0.11 work units)
Thread count was 2 (of 2 available processors)

Solution count 10: 120.934 120.612 120.413 ... 118.65

Optimal solution found (tolerance 1.00e-04)
Best objective 1.209342354664e+02, best bound 1.209342354664e+02, gap 0.0000%
Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (linux64)

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 218 rows, 83 columns and 596 nonzeros
Model fingerprint: 0x1836b654
Variable types: 56 continuous, 27 integer (27 binary)
Coefficient statistics:
  Matrix range     [2e-01, 1e+08]
  Objective range  [5e-01, 5e-01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+08, 1e+08]
Found heuristic solution: objective -0.0000000
Presolve removed 54 rows and 0 columns
Presolve time: 0.00s
Presolved: 164 rows, 83 columns, 488 nonzeros
Variable types: 56 continuous, 27 integer (27 binary)

Root relaxation: objective 4.588530e+02, 114 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  458.85304    0   27   -0.00000  458.85304      -     -    0s
H    0     0                        73.4949998  458.85304   524%     -    0s
     0     0  299.48693    0   15   73.49500  299.48693   307%     -    0s
     0     0  299.48693    0   15   73.49500  299.48693   307%     -    0s
H    0     0                        84.4263910  299.48693   255%     -    0s
```

```
     0     0  224.79233    0   14   84.42639  224.79233   166%     -    0s
     0     0  223.70940    0   14   84.42639  223.70940   165%     -    0s
     0     0  218.91935    0   14   84.42639  218.91935   159%     -    0s
H    0     0                      86.0000000  218.91935   155%     -    0s
H    0     0                      91.0807283  218.91935   140%     -    0s
     0     0  206.90997    0   12   91.08073  206.90997   127%     -    0s
     0     0  206.60044    0   12   91.08073  206.60044   127%     -    0s
     0     0  204.72465    0   12   91.08073  204.72465   125%     -    0s
     0     0  204.65466    0   12   91.08073  204.65466   125%     -    0s
     0     0  204.49473    0   12   91.08073  204.49473   125%     -    0s
     0     0  186.87527    0   12   91.08073  186.87527   105%     -    0s
H    0     0                      92.0000000  186.87527   103%     -    0s
H    0     0                      95.6181553  186.87527  95.4%     -    0s
     0     0  131.52727    0   11   95.61816  131.52727  37.6%     -    0s
     0     0  112.68763    0   10   95.61816  112.68763  17.9%     -    0s
H    0     0                      97.3822410  112.68763  15.7%     -    0s
     0     0  106.02237    0   10   97.38224  106.02237  8.87%     -    0s
     0     0  105.96498    0   10   97.38224  105.96498  8.81%     -    0s
     0     0  105.60743    0   10   97.38224  105.60743  8.45%     -    0s
     0     0  105.40881    0   10   97.38224  105.40881  8.24%     -    0s
     0     0  105.25159    0   10   97.38224  105.25159  8.08%     -    0s
     0     0  105.25128    0   10   97.38224  105.25128  8.08%     -    0s
     0     0  104.97953    0   10   97.38224  104.97953  7.80%     -    0s
     0     0  104.94182    0   10   97.38224  104.94182  7.76%     -    0s
     0     0  104.93671    0   10   97.38224  104.93671  7.76%     -    0s
     0     0  104.92652    0   10   97.38224  104.92652  7.75%     -    0s
     0     0  104.52351    0   11   97.38224  104.52351  7.33%     -    0s
     0     2  104.36471    0   11   97.38224  104.36471  7.17%     -    0s

Cutting planes:
  Gomory: 4
  MIR: 10
  Flow cover: 7

Explored 58 nodes (833 simplex iterations) in 0.31 seconds (0.06 work units)
Thread count was 2 (of 2 available processors)

Solution count 8: 97.3822 95.6182 92 ... -0

Optimal solution found (tolerance 1.00e-04)
Best objective 9.738224102259e+01, best bound 9.738224102259e+01, gap 0.0000%
```

[ ]:
```
pd.DataFrame(p5_res)
```

[ ]:
```
  Dataset                                      Assortment with Unknown Type  \
0   data1  [1, 2, 3, 4, 5, 6, 7, 16, 18, 19, 20, 21, 22, 23, 24, 25, 27]
1   data2                                    [1, 2, 7, 8, 9, 10, 11, 22]
```

```
2   data3   [1, 2, 3, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 20, 24, 25]
3   data4                            [4, 5, 7, 11, 16, 19, 20, 21, 22, 27]

   Expected Revenue with Unknown Type  \
0                           107.193586
1                           130.906235
2                           120.934235
3                            97.382241

                                                      Assortment for Early Cust
\
0  [0, 1, 2, 3, 4, 5, 6, 9, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26]
1                                           [0, 1, 6, 7, 8, 9, 10, 21, 23, 25]
2          [0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 13, 14, 15, 16, 18, 19, 23, 24, 25]
3                                     [3, 4, 6, 8, 10, 15, 18, 19, 20, 21, 26]

   Expected Revenue for Early Cust  \
0                       103.464546
1                       127.139496
2                       117.514092
3                        94.855285

                                      Assortment for Late Cust  \
0     [0, 1, 2, 3, 4, 5, 6, 15, 17, 18, 19, 20, 21, 22, 23, 24]
1                                       [0, 1, 6, 7, 8, 10, 21]
2  [0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 13, 14, 15, 16, 19, 23, 24]
3                         [3, 4, 6, 10, 15, 18, 19, 20, 21, 26]

   Expected Revenue for Late Cust
0                      112.207643
1                      137.684999
2                      125.386044
3                      100.952388
```

- Comparison of the expected revenue of S (Unknown) and S1 under MNL model of type 1 (Early)

Based on the table above, optimal assortments in S1 consist of more hotel options compared to in S with expected revenue of S1 lower than expected revenue of S. Our hypothesis is this is due to early customers are more price-sensitive and prefer a wider option.

- Comparison of the expected revenue of S and S2 under MNL model of type 2

Based on the table above, optimal assortments in S2 consist of fewer hotel options compared to in S and S1 with expected revenue is higher than S (and S1). One hypothesis on this last-minute customers might be willing to pay more for convenience and prefer a more curated selection compared to early customer.

6 Problem 6: Mixture of MNL with Other Ways of Defining Customer Types

In previous problems, we define customer types based on whether the customer wants to make an early or a late reservation. Given dataset data.csv, there are other ways to define customer types, for example, whether the customer in the search query is looking for a Saturday night booking. Explore your own ways to define customer types and estimate the mixture of MNL model. Repeat Problem 5 for this new MMNL model.

6.1 MNL Model based on Saturday vs Non-Saturday Bookings

[ ]:
```
columns = [col for col in data.columns if col.startswith('p')]
assortment_sat = data[data['srch_saturday_night_bool'] == 1].
↪groupby('srch_id')[columns].apply(lambda g: g.values.tolist()).tolist()
assortment_sat = [np.array(assortment) for assortment in assortment_sat]
assortment_sat = [scaler.transform(assortment) for assortment in assortment_sat]
assortment_sat = [np.concatenate([np.ones((len(assortment), 1)), assortment],␣
↪axis=1) for assortment in assortment_sat]
bookings_sat = data[data['srch_saturday_night_bool'] == 1].
↪groupby('srch_id')['booking_bool'].apply(lambda g: g.values.tolist()).tolist()
bookings_sat = [np.array(booking) for booking in bookings_sat]
assortment_nonsat = data[data['srch_saturday_night_bool'] == 0].
↪groupby('srch_id')[columns].apply(lambda g: g.values.tolist()).tolist()
assortment_nonsat = [np.array(assortment) for assortment in assortment_nonsat]
assortment_nonsat = [scaler.transform(assortment) for assortment in␣
↪assortment_nonsat]
assortment_nonsat = [np.concatenate([np.ones((len(assortment), 1)), assortment],␣
↪axis=1) for assortment in assortment_nonsat]
bookings_nonsat = data[data['srch_saturday_night_bool'] == 0].
↪groupby('srch_id')['booking_bool'].apply(lambda g: g.values.tolist()).tolist()
bookings_nonsat = [np.array(booking) for booking in bookings_nonsat]
```

[ ]:
```
epsilon_1= len(data[data['srch_saturday_night_bool'] == 1]['srch_id'].unique()) /
↪ len(data['srch_id'].unique())
epsilon_2= len(data[data['srch_saturday_night_bool'] == 0]['srch_id'].unique()) /
↪ len(data['srch_id'].unique())
print(f"epsilon_1 =", epsilon_1)
print(f"epsilon_2 =", epsilon_2)
```

```
epsilon_1 = 0.5766100071821881
epsilon_2 = 0.4233899928178118
```

[ ]:
```
def sum_log_sat_nonsat(param5, param6, x5, x6, y5, y6, epsilon_5, epsilon_6):
  sum_logL1= (epsilon_5 * sum_log(param5, x5, y5)) + (epsilon_6 *␣
↪sum_log(param6,x6,y6))
  return sum_logL1
```

[ ]:
```
param_sat_initial = np.zeros(9)
param_nonsat_initial = np.zeros(9)
initial_guess = np.concatenate([param_sat_initial, param_nonsat_initial])
result1 = minimize(lambda params: sum_log_sat_nonsat(params[:9], params[9:],
 ↪assortment_sat, assortment_nonsat, bookings_sat, bookings_nonsat, epsilon_1,
 ↪epsilon_2), x0=initial_guess, method='Powell')
```

[ ]:
```
beta_hat_sat= result1.x[:9]
beta_hat_nonsat= result1.x[9:]
columns = ['coef_' + col for col in data.columns if col.startswith('p')]
df_p6 = pd.DataFrame([['Saturday'] + beta_hat_sat.tolist(), ['Non-Saturday'] +
 ↪beta_hat_nonsat.tolist()], columns=['Type', 'Intercept'] + columns)
df_p6
```

```
           Type  Intercept  coef_prop_starrating  coef_prop_review_score  \
0      Saturday  -1.755092              0.363025                0.108878
1  Non-Saturday  -1.737982              0.476415                0.105263

   coef_prop_brand_bool  coef_prop_location_score  \
0              0.098139                  0.024299
1              0.107620                  0.014078

   coef_prop_accesibility_score  coef_prop_log_historical_price  \
0                      0.036228                       -0.064094
1                      0.052390                       -0.074377

   coef_price_usd  coef_promotion_flag
0       -1.157790             0.167435
1       -1.563576             0.147735
```

For customers who are looking for Saturday Night Bookings, the most influential features with high and positive coefficient values are star rating, review score and promotions of the property leading to higher probability. Other features such as brand, location score and accessibility also have positive coefficient even though they may have a smaller impact. On the other hand, price as well as historical price significantly have negative effect resulting in a decrease in the probability of booking.

The most influential features for customers not looking for Saturday Night Bookings are similar to the previous type with star rating, promotion and brand of the property having higher probability and location score and accessibility having positive but smaller impact. However, the coefficient for price and historical price is negative and higher compared to the first type probably because customers looking for non Saturday bookings are more flexible with the days and make the decision based on the price.

[ ]:
```
def prob_no_purchase(betas, assortment):
    utility = (betas * assortment).sum(axis=1)
    preference = np.exp(utility)
    return 1 / (1+preference.sum(axis=0))
```

[ ]:

```
beta= [beta_hat_sat, beta_hat_nonsat]
epsilon= [epsilon_1, epsilon_2]

p6_res = {'Dataset': [], 'Assortment with Unknown Type': [], 'Expected Revenue␣
↪with Unknown Type':[],
          'Assortment for Saturday Bookings': [], 'P(No Purchase) for Saturday␣
↪Bookings': [], 'Expected Revenue for Saturday Boookings':[],
          'Assortment for Non-Saturday Bookings': [], 'P(No Purchase) for␣
↪Non-Saturday Bookings': [], 'Expected Revenue for Non-Saturday Bookings':[]}
for x in range(1, 5):
    data_x = pd.read_csv(f'data{x}.csv')
    p6_res['Dataset'].append(f'data{x}')

    #S
    opt_assortment1, rev1 = optimal_assortment(data_x, beta, theta, 2)
    p6_res['Assortment with Unknown Type'].append(opt_assortment1)
    p6_res['Expected Revenue with Unknown Type'].append(rev1)

    #S1 and S2
    data_x = data_x.sort_values(by='price_usd', ascending=False)
    data_x_norm= scaler.transform(data_x.values)
    data_x_scaled = np.concatenate([np.ones((len(data_x_norm), 1)),␣
↪data_x_norm], axis=1)
    best_assortment_sat, max_revenue_sat = None, -1
    best_assortment_nonsat, max_revenue_nonsat = None, -1

    for n in range(1, len(data_x_scaled)):
        prices = data_x['price_usd'].iloc[:n]
        rev_sat = revenue(data_x_scaled[:n], beta[0], prices)
        if rev_sat > max_revenue_sat:
            best_assortment_sat, max_revenue_sat = data_x.iloc[:n].index, rev_sat

        rev_nonsat = revenue(data_x_scaled[:n], beta[1], prices)
        if rev_nonsat > max_revenue_nonsat:
            best_assortment_nonsat, max_revenue_nonsat = data_x.iloc[:n].index,␣
↪rev_nonsat

    p6_res['Assortment for Saturday Bookings'].append(best_assortment_sat.
↪tolist())
    p6_res['P(No Purchase) for Saturday Bookings'].
↪append(prob_no_purchase(beta_hat_sat, data_x_scaled[best_assortment_sat]))
    p6_res['Assortment for Saturday Bookings'] = [sorted(x) for x in␣
↪p6_res['Assortment for Saturday Bookings']]
    p6_res['Expected Revenue for Saturday Boookings'].append(max_revenue_sat)
```

```
    p6_res['Assortment for Non-Saturday Bookings'].append(best_assortment_nonsat.
↪tolist())
    p6_res['P(No Purchase) for Non-Saturday Bookings'].
↪append(prob_no_purchase(beta_hat_nonsat, data_x_scaled[best_assortment_sat]))
    p6_res['Assortment for Non-Saturday Bookings'] = [sorted(x) for x in␣
↪p6_res['Assortment for Non-Saturday Bookings']]
    p6_res['Expected Revenue for Non-Saturday Bookings'].
↪append(max_revenue_nonsat)
```

[ ]:
```
pd.DataFrame(p6_res)
```

```
  Dataset                                      Assortment with Unknown Type  \
0   data1      [1, 2, 3, 4, 5, 6, 7, 16, 18, 19, 20, 21, 22, 23, 24, 25, 27]
1   data2                                 [1, 2, 7, 8, 9, 10, 11, 22, 24, 26]
2   data3  [1, 2, 3, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 24, 25]
3   data4                              [4, 5, 7, 9, 11, 16, 19, 20, 21, 22, 27]

   Expected Revenue with Unknown Type  \
0                          107.371426
1                          131.245381
2                          121.150493
3                           97.533480

                                         Assortment for Saturday Bookings  \
0  [0, 1, 2, 3, 4, 5, 6, 9, 12, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 26]
1                                     [0, 1, 6, 7, 8, 9, 10, 21, 23, 25]
2          [0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 13, 14, 15, 16, 18, 19, 23, 24]
3                              [3, 4, 6, 8, 10, 15, 18, 19, 20, 21, 26]

   P(No Purchase) for Saturday Bookings  \
0                              0.215389
1                              0.214357
2                              0.284000
3                              0.190383

   Expected Revenue for Saturday Boookings  \
0                                 106.824547
1                                 130.078292
2                                 121.600537
3                                  96.788086

                           Assortment for Non-Saturday Bookings  \
0     [0, 1, 2, 3, 4, 5, 6, 15, 17, 18, 19, 20, 21, 22, 23, 24, 26]
1                                  [0, 1, 6, 7, 8, 9, 10, 21, 23, 25]
2  [0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 13, 14, 15, 16, 18, 19, 23, 24]
3                          [3, 4, 6, 8, 10, 15, 18, 19, 20, 21, 26]
```

```
   P(No Purchase) for Non-Saturday Bookings  \
0                                  0.208732
1                                  0.186260
2                                  0.281082
3                                  0.158543

   Expected Revenue for Non-Saturday Bookings
0                                  108.063333
1                                  132.632618
2                                  120.615557
3                                   98.419477
```

- Comparison of the expected revenue of S (Unknown) and S1 under MNL model of type 1 (Saturday Booking)

Based on the table above, optimal assortments in S1 has lower expected revenue than compared to S. Even though on average the price for Saturday bookings is higher since the Probability for No Purchase being made for Saturday Bookings is higher compared to that for non Saturday Bookings, so less bookings are made probably due to the high prices leading to a lower revenue.

- Comparison of the expected revenue of S and S2 under MNL model of type 2

Based on the table above, optimal assortments in S2 consist of higher expected revenue compared to S (and S1). Similar to the previous case, since the Probability for No Purchase being made for Non - Saturday Booking is lower than compared to type 1, there are more chances of a booking being made, increasing expected revenue.

6.2 MNL for Customer Booking with Child vs Non-child

[ ]:
```
columns = [col for col in data.columns if col.startswith('p')]
assortments_child = data[data['srch_children_count'] != 0].
 ↪groupby('srch_id')[columns].apply(lambda g: g.values.tolist()).tolist()
assortments_child = [np.array(assortment) for assortment in assortments_child]
assortments_child = [scaler.transform(assortment) for assortment in␣
 ↪assortments_child]
assortments_child = [np.concatenate([np.ones((len(assortment), 1)), assortment],␣
 ↪axis=1) for assortment in assortments_child]
bookings_child = data[data['srch_children_count'] != 0].
 ↪groupby('srch_id')['booking_bool'].apply(lambda g: g.values.tolist()).tolist()
bookings_child = [np.array(booking) for booking in bookings_child]
assortments_nochild = data[data['srch_children_count'] == 0 ].
 ↪groupby('srch_id')[columns].apply(lambda g: g.values.tolist()).tolist()
assortments_nochild = [np.array(assortment) for assortment in␣
 ↪assortments_nochild]
assortments_nochild = [scaler.transform(assortment) for assortment in␣
 ↪assortments_nochild]
assortments_nochild = [np.concatenate([np.ones((len(assortment), 1)),␣
 ↪assortment], axis=1) for assortment in assortments_nochild]
```

```
bookings_nochild = data[data['srch_children_count'] == 0].
↪groupby('srch_id')['booking_bool'].apply(lambda g: g.values.tolist()).tolist()
bookings_nochild = [np.array(booking) for booking in bookings_nochild]
```

[ ]:
```
epsilon_5= len(data[data['srch_children_count'] == 0]['srch_id'].unique()) /␣
↪len(data['srch_id'].unique())
epsilon_6= len(data[data['srch_children_count'] != 0]['srch_id'].unique()) /␣
↪len(data['srch_id'].unique())
print(f"epsilon_5 =", epsilon_5)
print(f"epsilon_6 =", epsilon_6)
```

[ ]:
```
def sum_log_child(param7, param8, x7, x8, y7, y8, epsilon5, epsilon6):
  sum_logL= (epsilon5 * sum_log(param7, x7, y7)) + (epsilon6 *␣
↪sum_log(param8,x8,y8))
  return sum_logL
```

[ ]:
```
param_child_initial = np.zeros(9)
param_nochild_initial = np.zeros(9)
initial_guess = np.concatenate([param_child_initial, param_nochild_initial])
result3 = minimize(lambda params: sum_log_child(params[:9], params[9:],␣
↪assortments_child, assortments_nochild, bookings_child, bookings_nochild,␣
↪epsilon_5, epsilon_6), x0=initial_guess, method='Powell')
```

[ ]:
```
beta_hat_child= result3.x[:9]
beta_hat_nochild = result3.x[9:]
columns = ['coef_' + col for col in data.columns if col.startswith('p')]
df_p8 = pd.DataFrame([['With Child'] + beta_hat_child.tolist(), ['No Child'] +␣
↪beta_hat_nochild.tolist()], columns=['Type', 'Intercept'] + columns)
df_p8
```

```
         Type  Intercept  coef_prop_starrating  coef_prop_review_score  \
0  With Child  -1.630091              0.306199                0.104049
1    No Child  -1.789953              0.448788                0.108076

   coef_prop_brand_bool  coef_prop_location_score  \
0              0.134787                  0.025146
1              0.093688                  0.018319

   coef_prop_accesibility_score  coef_prop_log_historical_price  \
0                      0.040039                       -0.148323
1                      0.043815                       -0.038002

   coef_price_usd  coef_promotion_flag
0       -0.687487             0.171185
1       -1.560769             0.154869
```

For customers whose bookings are with child, the most influential features with high and postitive coefficient values are star rating, promotions and hotel brand of the property leading to higher probability. Other features such as review score, location score and accessibility also have positive coefficient even though they may have a smaller impact. On the other hand, price as well as historical price significantly have negative effect resulting in a decrease in the probablity of booking.

The most influential features for customers not looking for Saturday Night Bookings are similar with star rating, promotion, and review score having higher probaiblity and brand of the property, location score and accessibility having postitive but smaller impact. However, the coefficient for price and historical price is negative and higher probably because customers that have no child can be more flexible when it comes to choosing a hotel.

[ ]:
```
beta= [beta_hat_child, beta_hat_nochild]
epsilon= [epsilon_5, epsilon_6]

p8_res = {'Dataset': [], 'Assortment with Unknown Type': [], 'Expected Revenue␣
↪with Unknown Type':[],
          'Assortment with Child': [], 'Expected Revenue with Child':[],
          'Assortment with No Child': [], 'Expected Revenue with No Child':[]}
for x in range(1, 5):
    data_x = pd.read_csv(f'data{x}.csv')
    p8_res['Dataset'].append(f'data{x}')

    #S
    opt_assortment4, rev4= optimal_assortment(data_x, beta, theta, 2)
    p8_res['Assortment with Unknown Type'].append(opt_assortment4)
    p8_res['Expected Revenue with Unknown Type'].append(rev4)

    #S1 and S2
    data_x = data_x.sort_values(by='price_usd', ascending=False)
    data_x_norm= scaler.transform(data_x.values)
    data_x_scaled = np.concatenate([np.ones((len(data_x_norm), 1)),␣
↪data_x_norm], axis=1)
    best_assortment_child, max_revenue_child = None, -1
    best_assortment_nochild, max_revenue_nochild = None, -1

    for n in range(1, len(data_x_scaled)):
        prices = data_x['price_usd'].iloc[:n]
        rev_child = revenue(data_x_scaled[:n], beta[0], prices)
        if rev_child > max_revenue_child:
            best_assortment_child, max_revenue_child = data_x.iloc[:n].index,␣
↪rev_child

        rev_nochild = revenue(data_x_scaled[:n], beta[1], prices)
        if rev_nochild > max_revenue_nochild:
            best_assortment_nochild, max_revenue_nochild = data_x.iloc[:n].
↪index, rev_nochild

    p8_res['Assortment with Child'].append(best_assortment_child.tolist())
```

```
    p8_res['Assortment with Child'] = [sorted(x) for x in p8_res['Assortment␣
↪with Child']]
    p8_res['Expected Revenue with Child'].append(max_revenue_child)

    p8_res['Assortment with No Child'].append(best_assortment_nochild.tolist())
    p8_res['Assortment with No Child'] = [sorted(x) for x in p8_res['Assortment␣
↪with No Child']]
    p8_res['Expected Revenue with No Child'].append(max_revenue_nochild)
```

[ ]:
```
pd.DataFrame(p8_res)
```

```
  Dataset                                  Assortment with Unknown Type  \
0   data1  [1, 2, 3, 4, 5, 6, 7, 16, 18, 19, 20, 21, 22, 23, 24, 25, 27]
1   data2                                      [1, 2, 7, 8, 9, 11, 22]
2   data3          [1, 2, 3, 4, 5, 6, 8, 11, 14, 15, 16, 17, 20, 24, 25]
3   data4                          [4, 5, 7, 11, 16, 19, 20, 21, 22, 27]

   Expected Revenue with Unknown Type  \
0                          107.841956
1                          134.127574
2                          124.598607
3                           99.142871

                                        Assortment with Child  \
0  [0, 1, 2, 3, 4, 5, 6, 15, 17, 18, 19, 20, 21, 22, 23, 24, 26]
1                                      [0, 1, 6, 7, 8, 10, 21]
2          [0, 1, 2, 3, 4, 5, 7, 10, 13, 14, 15, 16, 19, 23, 24]
3                          [3, 4, 6, 10, 15, 18, 19, 20, 21, 26]

   Expected Revenue with Child  \
0                   108.600662
1                   141.752748
2                   130.301298
3                   102.382759

                                          Assortment with No Child  \
0  [0, 1, 2, 3, 4, 5, 6, 9, 12, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 26]
1                                      [0, 1, 6, 7, 8, 9, 10, 21, 23, 25]
2      [0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 13, 14, 15, 16, 18, 19, 23, 24, 25]
3                              [3, 4, 6, 8, 10, 15, 18, 19, 20, 21, 26]

   Expected Revenue with No Child
0                      106.963076
1                      128.111309
2                      118.779934
3                       95.723844
```

- Comparison of the expected revenue of S (Unknown) and S1 under MNL model of type 1

(With Child)

Based on the table above, optimal assortments in S1 has higher expected revenue than that of S. The most likely reason for this is because the hotel price goes up wuth adddition for children than without.

- Comparison of the expected revenue of S and S2 under MNL model of type 2 (N Child)

Based on the table above, optimal assortments in S2 consist of lower expected revenue compared to S (and S1). Similar to the previous case, since the price for without child is lower for any hotel, the expected revenue decreases.