ORIE 5355/INFO 5370 HW 3: Algorithmic Pricing

- Name:
- Net-id:
- Date:
- Late days used for this assignment:
- Total late days used (counting this assignment):
- People with whom you discussed this assignment:

After you finish the homework, please complete the following (short, anonymous) post-homework survey: https://forms.gle/N2hdk8B4r7TF1RDG6 and include the survey completion code below.

Question 0 [1 point]

Survey completion code:

We have marked questions in blue . Please put answers in black (do not change colors). You'll want to write text answers in "markdown" mode instead of code. In Jupyter notebook, you can go to Cell > Cell Type > Markdown, from the menu. Please carefully read the late days policy and grading procedure here.

Conceptual component [4 points]

Please complete the following pricing ethics scenario questionaire: https://forms.gle/dLq7mC32ft1NrhK69, and include the survey completion code below. **We will discuss these issues in class most likely on 10/5 (Exact date to be announced). You must complete the questionaire before the day of that class, even if you turn in the rest of the homework later. The questionaire will close the morning of the class that we discuss these issues.**

Survey completion code:

```
In [ ]:
```

Survey completion code: Based on the first letter of your first name, explain your answers to the following questions, in at most three sentences each.

First letter A-C: 1, 6, 11, 16

First letter D-H: 2, 7, 12, 17

First letter I-M: 3, 8, 13, 18

First letter N-S: 4, 9, 14, 19

First letter T-Z: 5, 10, 15, 20

Be prepared to discuss your answers to at least these questions in class (I might randomly call on people), but you should also be willing/able to discuss your answers to other questions.

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

Programming component

Helper code

In [ ]:

```
import numpy as np
import pandas as pd
import os, sys, math
import matplotlib.pyplot as plt
```

In [ ]:

```
df_train = pd.read_csv('HW3_data_train.csv')
test_demand_curve = pd.read_csv('test_demand.csv')
```

In [ ]:

```
df_train.head()
```

Out[ ]:

	Location	Income	Offered price	Purchased
0	Africa	10.38	3.16	False
1	Europe	26.33	3.47	True
2	Europe	24.06	3.78	True
3	Africa	16.18	3.74	False
4	Asia Pacific	13.73	4.75	False

In [ ]:

```
df_train.shape, test_demand_curve.shape
```

Out[ ]:

```
((4000, 4), (199, 5))
```

Problem 1: Demand estimation and pricing without

covariates

First, we will use the training data to construct estimates of the demand at each price without leveraging the covariates, and then use that estimated function to calculate optimal prices.

```
In [ ]:
```

1a) Naive method: empirical estimate of demand $d(p)$ at each price

Fill in the below function, that takes in a dataframe and the number of bins into which to separate the historical prices. The function should output a dataframe that has one row for each price bin, with two columns: the bin interval, and the estimated demand $d(p)$ (the fraction of potential customers who purchase at price p) in that bin.

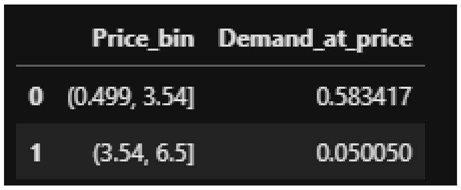
Use the following function to create bins: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.qcut.html

```
In [ ]: # Example with 10 bins:
df_train['bin_with_10_bins'] = pd.qcut(df_train['Offered price'], 10)
df_train.head()
```

Out[ ]:

	Location	Income	Offered price	Purchased	bin_with_10_bins
0	Africa	10.38	3.16	False	(2.95, 3.54]
1	Europe	26.33	3.47	True	(2.95, 3.54]
2	Europe	24.06	3.78	True	(3.54, 4.19]
3	Africa	16.18	3.74	False	(3.54, 4.19]
4	Asia Pacific	13.73	4.75	False	(4.19, 4.77]

For example, with 2 bins and passing in df_train to the function, you should see the following output:



	Price_bin	Demand_at_price
0	(0.499, 3.54]	0.583417
1	(3.54, 6.5]	0.050050

```
In [ ]: def create_empirical_estimate_demand_df(df, number_of_pricing_bins):
            pass
```

```
In [ ]:
```

Fill in the below function, that takes in a single price and your empirical df from the above function and outputs the prediction for the demand $d(p)$ at that price. For example, with 2 bins, at price = 3 the function should output 0.583417.

If the price is lower than the smallest bin, then use the value of the smallest bin. If it is higher than the highest bin, use the value of the highest bin.

In [ ]:
```
def get_prediction_empirical(empirical_df, price):
    pass
```

In [ ]:

In [ ]:
```
prices_to_predict = np.linspace(min(df_train['Offered price']), max(df_train['Offered
```

Plot in a single figure the outputs of your function as a line plot -- where the X axis corresponds to prices in `prices_to_predict` and the Y axis the predicted Demand at that price -- for the following three inputs to the function:

```
1. the dataframe is the first 100 rows of df_train, with 10 bins.

2. the dataframe is the first 500 rows of df_train, with 10 bins.

3. the dataframe is all the rows of df_train, with 10 bins.
```

In the same figure, include the "true" test-time demand curve, `test_demand_curve` -- plot the mid-point of each bin on the X axis, and the demand for that bin on the Y axis. So your plot will contain 4 curves in total.

In [ ]:

In [ ]:

Do the same plot, except now you're using 50 bins for each of the three data frames.

In [ ]:

Comment on your output in no more than 3 sentences. What is the effect of using more data and more bins?

1b) Demand estimation using logistic regression

First, Fill in the below function that fits a logistic regression to predict the probability of purchase at a price ($d(p)$). The logistic regression should just have two coefficients: one for the intercept, and one for the price. The function takes in a dataframe that you will use as your training data for your model, and should return your fitted model.

In [ ]:
```
def fit_logistic_regression_demand_just_on_price(df):
    pass
```

Fill in the below function, that takes in a single price and your trained model and outputs the prediction for the demand $d(p)$ at that price.

Note that you do not want to treat logistic regression as a binary classifier that outputs either 0 or 1. Rather, you want to get the probability of being a 1. You can extract this using the predict_proba(X) function.

In [ ]:
```python
def get_prediction_logistic(fitted_model, price):
    pass
```

For each of the three training dataframes as in part A, fit a model and get the predictions for each of the prices in `prices_to_predict` using your above function. Generate the same lineplot as above. Also include the "true" test-time demand curve, `test_demand_curve`.

In [ ]:
```python

```

Comment on your output in no more than 3 sentences. What is the effect of using logistic regression instead of the empirical distribution?

In [ ]:
```python

```

1c) Optimal pricing using your demand estimates

Fill in the following function that takes in two lists: a list of prices, and a list of predicted demand d(p) at that price. The function outputs the revenue maximizing price given the data and the corresponding revenue. You may use a "brute force" technique, that loops through all the possible prices and calculates the revenue using that price.

In [ ]:
```python
def get_revenue_maximizing_price_and_revenue(price_options, demand_predictions):
    pass
```

In [ ]:
```python

```

Print out the optimal price and the predicted optimal revenue from the predictions for your naive and logistic models, using 100 rows and all the data, each.

For example, we got the following (your numbers may differ slightly):

logistic, 100 points: 2.580402010050251 1.2143341610705582

naive, all points: 2.278894472361809 1.348157868550674

In [ ]:
```python

```

Now, we're going to use the "true" test-time demand curve, `test_demand_curve`. For each of the above predicted optimal prices, calculate the revenue resulting from that price used on the true demand curve. Also print out the true optimal price and corresponding revenue for that curve.

For example, we got:

true revenue using logistic 100 price: 0.9729384628058323

In [ ]:
```

```

How do your estimates compare to the actual revenue? Discuss in no more than 3 sentences.

In [ ]:
```

```

Problem 2: Demand estimation and pricing with covariates

Now, we are going to ask you to do personalized pricing, based on just a two binarized covariates.

First, take `df_train` and create a new column for "low" and "high" wealth, based on if the income level is above or below the median income level. Second, create a new column for Location: `1` if the location is either America, and `0` if the location is anything else.

For this section, we will use all the df_train data, as opposed to just the first few rows.

2a) Demand estimation

First, Fill in the below function that fits a logistic regression to predict the probability of purchase at a price ($d(p)$). The logistic regression should now have more coefficients than before: 1 for each covariate, and any interactions (including interactions between price and covariates) that you wish to add. If you add more interactions, you may wish to add regularization.

In [ ]:
```python
def fit_logistic_regression_demand_with_covariates(df):
    pass
```

Fill in the below function, that takes in a single price, covariates, and your trained model, and outputs the prediction for the demand $d(p)$ at that price. For example, one of the covariate inputs to the function can be `['NotAmerica', 'LowWealth']`.

In [ ]:
```python
def get_prediction_logistic(fitted_model, price, covariates):
    pass
```

In [ ]:
```python
test_demand_curve_America_HighWealth = pd.read_csv('test_demand_America_HighWealth.csv
test_demand_curve_NotAmerica_HighWealth = pd.read_csv('test_demand_NotAmerica_HighWeal
test_demand_curve_America_LowWealth = pd.read_csv('test_demand_America_LowWealth.csv')
test_demand_curve_NotAmerica_LowWealth = pd.read_csv('test_demand_NotAmerica_LowWealth
```

Fit a model and get the predictions for each of the prices in `prices_to_predict` using your above function and each unique covariate combination.

For example, `test_demand_NotAmerica_LowWealth`, we got:

Group Not America, Low Wealth: Optimal price 1.314070, Revenue 0.913284, True revenue 0.788442

For each covariate combination, generate the same lineplot as in 1a and 1b (separately for each covariate combination). Also include the "true" test-time demand curve for the appropriate covariate combination

In [ ]:

```
```

2b) Pricing

Now, use your code from 1c to output predicted optimal prices, predicted revenue, and and actual revenue using the test data curve, for each covariate combination.

In [ ]:

```
```

Suppose each of the 4 covariate combinations make up an equal part of the population. What would be the resulting revenue achieved at test time if you use the optimal price for each group (so you look at their covariates, and then give them the optimal price for that group).

In [ ]:

```
```

Comment on your outputs in no more than 3 sentences. What is the effect of using different prices for differerent covariate groups?

Problem 3: Pricing under capacity constraints

Now, we are going to build up to implementing the Bellman equation approach discussed in class, to price a single copy of an item to be sold over T time periods. For simplicity, we will use `test_demand_curve` as $d(p)$.

In [ ]:

```
price_options = list(test_demand_curve.Price)
demand_predictions = list(test_demand_curve.Demand_at_price)
```

3a) Implementing one step of the Bellman equation

Recall the "Bellman equation" taught in class. Suppose we have 1 copy of the item at time t. Then, my expected revenue given I price the item at p_t is:

$$V_t = d(p_t)p_t + (1 - d(p_t))V_{t+1}$$

Implement the following function that returns optimal price p_t and the resulting value V_t, given the demand curve and V_{t+1}.

For example, we find that the output of the following function call is: (2.083, 1.1922434210526316)

`get_single_step_revenue_maximizing_price_and_revenue(0, price_options, demand_predictions)`

```
In [ ]: def get_single_step_revenue_maximizing_price_and_revenue(Vtplus1, price_options, demar
            pass
```

```
In [ ]:
```

3b) Calculating prices over time

Implement the following function that returns a list (of length T) of optimal prices for each time period, and a expected revenue number for those prices.

Hint: your function should loop through each time step, *starting* at time $t = T - 1$ (the last time period, since the first time period is time $t = 0$). Each iteration of the loop should call the function from part 3a. Recall that we can define $V_T = 0$, since even if the item is unsold at time T, we have finished trying to sell it.

```
In [ ]: def get_prices_over_time_and_expected_revenue(prices, demand_predictions, T):
            pass
```

Plot a line plot for your optimal prices over time when $T = 100$ and $T = 10$. Also print out the expected revenue using these prices and for each T.

For example, when $T = 100$, we find that prices[0] = 5.822, prices[90] = 4.224, and that revenue = 5.2287

```
In [ ]:
```

3c) [Bonus, 3 points] Prices over time with multiple copies

Now, suppose that you have K copies of the item, that you must sell over a time period T. Implement the two-dimensional dynamic program as discussed in class. Plot a line plot where the X axis is time as in 3b, but now you have K lines where each line indicates the price at time T if you have K items left.

Hint: As in 3a and 3b, you may find it useful to first optimize the price $p_{t,k}$ given the values $V_{t+1,k}$, $V_{t+1,k-1}$. Then, have a 2nd function that loops through t, k in an appropriate order.

```
In [ ]:
```