Project Report: Starbucks Capstone Project

Arya Suneesh

April 17, 2023

1 Introduction

In this project, we analyzed the purchase behavior of customers to identify the most popular offers based on net expense and average transaction value. We also explored how the popularity of offers varies among customers based on their demographics such as age, income, and gender.

2 Data

We used a dataset of customer transactions containing information about offers received, viewed, and completed by the customers. The dataset also included information about the customers' demographics such as age, income, and gender.

The data used for this project is contained in three files: `portfolio.json`, `profile.json`, and `transcript.json`.

`portfolio.json` contains offer ids and metadata about each offer. It includes the following variables:

- `id` (string) - offer id
- `offer_type` (string) - type of offer, such as BOGO, discount, or informational
- `difficulty` (int) - minimum required spend to complete an offer
- `reward` (int) - reward given for completing an offer
- `duration` (int) - time for offer to be open, in days
- `channels` (list of strings) - channels used to distribute the offer

`profile.json` contains demographic data for each customer. It includes the following variables:

- `age` (int) - age of the customer
- `became_member_on` (int) - date when the customer created an app account
- `gender` (string) - gender of the customer (some entries contain 'O' for other rather than 'M' or 'F')
- `id` (string) - customer id
- `income` (float) - customer's income

`transcript.json` contains records for transactions, offers received, offers viewed, and offers completed. It includes the following variables:

- `event` (string) - record description (such as transaction, offer received, offer viewed, etc.)

- `person` (string) - customer id
- `time` (int) - time in hours since the start of the test. The data begins at time $t = 0$
- `value` (dictionary of strings) - either an offer id or transaction amount depending on the record

3 Data Preprocessing

We first cleaned the data by removing duplicate records, filling missing values, and converting data types as needed. We also derived additional columns such as net expense and income groups based on the original data.

- The `clean_portfolio()` function takes the portfolio dataframe as an input and returns a cleaned version of it. The function creates dummy columns for the channels column and renames the id column to offer_id.
- The `clean_profile()` function takes the profile dataframe as an input and returns a cleaned version of it. The function transforms the became_member_on column from an integer to a datetime format. It creates a new valid column that identifies whether a customer's age is valid or not (invalid ages are denoted as 118 in the dataset). The function also renames the id column to customer_id and creates dummy variables for the gender column.
- The `clean_transcript()` function takes the transcript dataframe as an input and returns a cleaned version of it. The function first splits

the event column into several dummy columns, with each column representing a unique event. Then, it extracts the offer_id and amount data from the value column and creates new columns for them. Finally, the function changes the name of the person column to customer_id.

The cleaned dataframes are then merged together and stored in the `df` variable. `offer_id` is a dictionary that maps the original offer ids to a simplified form. The new offer ids are stored in the offer$_idcolumnofthe$`df`$dataframe$.

4 Data Analysis

In the data analysis portion of the project, various functions perform various data analysis tasks on a merged dataframe called `df`, which contains information on customer demographics, transaction history, and offer interactions. The code creates various functions to extract and organize data related to customer offer interactions and store it in a dictionary called `cust_dict`.

The first part of the code creates bar plots of the number of offers by offer type and offer ID, respectively, using the `groupby` method of the `df` dataframe.

The helper function used are listed below:

The `get_offer_cust` function takes two arguments: `df` (the merged dataframe) and `offer_type` (optional, default is None). It returns a dictionary with keys for 'received', 'viewed', 'completed', and 'reward' (if applicable) for each offer

type. The function filters the dataframe based on the `offer_type` parameter and aggregates the data for each customer using the `groupby` method.

The `get_offer_id_cust` function takes two arguments: `df` and `offer_id`. It returns a dictionary with keys for 'B1_received', 'B1_viewed', 'B1_completed', 'B1_reward', etc. for the specified offer_id. The function filters the dataframe based on the `offer_id` parameter and aggregates the data for each customer using the `groupby` method.

The `round_age` function takes an age value and rounds it to the nearest multiple of 10, between 15 and 105.

The `round_income` function takes an income value and rounds it down to the nearest multiple of 10,000, between 30,000 and 120,000.

The `cust_dict` dictionary is initialized with total transaction data for each customer, including the sum of all their expenses and the number of transactions they made. The `get_offer_cust` function is then called for each offer type, and the resulting dictionaries are added to `cust_dict` using the `update` method. Finally, the `get_offer_id_cust` function is called for each offer ID, and the resulting dictionaries are also added to `cust_dict` using the `update` method.

The `get_average_expense(customers, offer)` function calculates the average expense for customers who received, viewed, and completed the offer.

It first calls the `get_offer_stat(customers, 'total_expense', offer)` function to obtain the total expenses for customers who received, viewed, and completed the offer. It then calls the `get_offer_stat(customers, 'total_transactions', offer)` function to obtain the total number of transactions for customers who received, viewed, and completed the offer. Finally, it divides the total expenses by the total number of transactions to obtain the average expense for customers who received, viewed, and completed the offer. If the offer is an informational offer, the function does not calculate the average expense for customers who completed the offer.

The `get_average_reward(customers, offer)` function calculates the average reward for customers who completed the offer. It first selects the completed offers from the customers dataframe by calling `completed = customers[(customers.valid == 1)&(customers[cpd_col] > 0)]`, where `cpd_col =` *'_completed'*`.format(offer)`. It then returns the average reward for completed offers by calling `completed[rwd_col].mean()`, where `rwd_col =` *'_reward'*`.format(offer)`. If the offer is not a discount or bogo offer, the function returns None for the average reward.

The function `plot_offer_expense` takes in two arguments:

- `customers`: A Pandas DataFrame that contains customer transaction and offer data
- `offer`: A string that specifies the type of offer to analyze ('bogo', 'discount', 'informational', or a specific offer ID such as 'B1', 'D2', etc.)

The function generates a 2x1 grid of histograms to visualize the transaction data for the specified offer type. The first plot shows the total transaction amount for customers who received, viewed, and completed the offer. The second plot shows the average transaction amount for customers who received, viewed, and completed the offer.

The histograms are plotted using Matplotlib, with 100 bins for the total transaction plot and 50 bins for the average transaction plot. The histograms for the different groups (received, viewed, completed) are plotted with different colors and an alpha value of 0.5. The x-axis limits for the two plots are set to 0 and 600 for the total transaction plot and 0 and 50 for the average transaction plot.

Note that for 'informational' offers and specific offer IDs ('I1', 'I2', 'B1', 'D1', etc.), the "completed" group is not applicable, so it is not plotted.

The `plot_offer_expense_by` function creates a six-panel plot of different statistics related to customer behavior for a given offer. The function takes in two arguments: a Pandas DataFrame `customers` containing customer data and a string `offer` containing the name of the offer to analyze.

The function first creates six empty dictionaries to store statistics for each of three customer demographics (age_group, income_group, and gender) and for three different metrics (net_expense, received, and viewed). The `get_offer_stat_by` function is then used to calculate the relevant statistics for each demographic

and metric, storing the results in the dictionaries. The `get_average_expense_by` function is also used to calculate the average transaction value for each demographic.

The six subplots of the resulting plot each show different combinations of these statistics for each demographic and metric. The first two subplots show line plots of net expense (the sum of all transactions) for each demographic and metric, with separate lines for customers who received the offer and those who viewed it. For offers that can be completed, a third line is shown for customers who completed the offer.

The third subplot shows a stacked bar chart of net expense for each gender, with separate bars for customers who received, viewed, and completed the offer.

The next three subplots show the same statistics as the first two, but for average transaction value instead of net expense.

The final subplot shows a stacked bar chart of average transaction value for each gender.

5 Recommendation System - Without Using Demographics

We defined functions to create a recommendation system for popular offers based on customer purchase behavior without using demographic data such as age, income, or gender.

The `get_net_expense` function calculated the net expense of customers who viewed and completed a specific offer and made at least 5 transactions. The `get_most_popular_offers` function sorted a list of offers by net expense and returned the top n offers along with their net expense.

The `get_most_popular_offers_filtered` function filtered the customer data by specified demographics if provided and then called the `get_most_popular_offers` function to return the top n offers based on the filtered data.

The code also included calls to plot the net expense of specific offers, such as B2, B1, and D1.

The conclusion reached was that the completed offers customers data for the recommended offer (B3) provides both the maximum net expense and average transaction value. Hence, B3 offers pertaining to the buy-one-get-one (BOGO) offers should be pushed more to the user.

6 Recommendation System - Using Demographics

Here, we analyzed the popularity of different offers among customers based on their net expenses and filtered the results based on specific customer demographics like gender, age, and income.

The `get_net_expense()` function calculated the net expense of customers who viewed or completed a specific offer and had a minimum number of total transactions. The `get_most_popular_offers()` function sorted the offers based on their net expenses and returned the top n offers along with their corresponding net expenses.

The `get_most_popular_offers_filtered()` function filtered the customers based on their demographic information and returned the most popular offers among the filtered customers.

The code then used the `get_offer_stat_by()` and `get_average_expense_by()` functions to calculate the net expense and average transaction value of the 'D1' offer for different gender groups. Finally, the code generated two bar plots to visualize the net expenses and average transaction values of the 'D1' offer for different gender groups.

We were able to deduce that women were more inclined to go through with the D1 offers (discount). Hence, such discount promotions should be targeted

more towards women.

7 Conclusion

Overall, we successfully created a recommendation system for popular offers based on customer purchase behavior, and also explored how the popularity of offers varies among customers based on their demographics. The insights generated from this analysis can be used by businesses to tailor their marketing strategies to different customer groups and optimize their offers to increase sales and revenue.

Further analysis may be done while exploring the different demographics. This project showcases only one analysis, pertaining to the gender of the user.