Optimizing App Offers - Starbucks

Diego Rosa

Github Profile LinkedIn

Udacity Data Scientist Nanodegree Capstone Project

Notebook Summary

1 - Business Undestanding

> **Starbucks Corporation** is an American multinational chain of coffeehouses and roastery rese
> Washington.

> As the largest coffeehouse in the world, Starbucks is seen to be the main representation of th
> coffee culture.

Starbucks Article on wikepedia

One convenient way to pay in store, order ahead for pickpup or even get updated about new drinks
Rewards are built right in, so you'll collect Stars and start earning free drinks and food with every p

The data sets used in this project contains simulated data that mimics customer behavior on the S
few days, Starbucks sends out an offer to users of the mobile app. An offer can be merely an adve
as a discount or BOGO (buy one get one free). Some users might not receive any offer during certa

Goal:

Not all users receive the same offer, and that is the challenge this project aims to solve:

The task here is to combine **transaction**, **demographic** and **offer data** to determine **which demogra**
type. This data set is a simplified version of the real Starbucks app because the underlying simula
actually sells dozens of products.

More details:

Every offer has a validity period before the offer expires. As an example, a BOGO offer might be va
that informational offers have a validity period even though these ads are merely providing inform
informational offer has 7 days of validity, you can assume the customer is feeling the influence of
advertisement.

One'll be given transactional data showing user purchases made on the app including the timestar
spent on a purchase. This transactional data also has a record for each offer that a user receives a
views the offer. There are also records for when a user completes an offer.

It's worth to keep in mind as well that someone using the app might make a purchase through the
an offer.

2 - Data Understanding

Data Sets

Tha data is contained in three files:

- *portifolio.json* - containing offer ids and meta data about each offer (duration, type, etc);
- *profile.json* - demographic data for each user;
- *transcript.json* - records for transactions, offers recieved, offer viewed, and offers completed

Data Dictionary

portifolio.json

- id (string) - offer id;
- offer_type (string) - type of offer:
    - BOGO (Buy one get one free);
    - discount;
    - informational;
- difficulty (int) - minimun required spend to complete an offer;
- reward (int) - reward given for completing an offer
- channels (list of strings) - communication channels in which the offer might be sent;

profile.json

- age (int) - age of the costumer;
- became_member_on (int) - date when costumer created an app account;
- gender (str) - gender of the costumer
    - M : Male
    - F : Female
    - O : Others
- id (str) - costume id;

- income (float) - costumer's income;

transcript.json

- event (str) - record description:
  - offer received;
  - offer viewed;
  - transaction;
  - offer completed;
- person (str) - costumer id;
- time (int) - time in hours since start of the test. Data begins at time t=0;
- value (dict of strings) - either an offer id or transaction amount depending on the record;

```
# Import relevant packages:

#Data packages
import pandas as pd
import json

#Utility
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
from datetime import date

#Unsupervised Machine Learning
from sklearn.mixture import GaussianMixture
from mpl_toolkits.mplot3d import Axes3D

#Supervised machine learning
import sklearn as sk
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import GridSearchCV



#metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import make_scorer
from sklearn.metrics import average_precision_score
```

```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
# Import portifolio.json from google drive

link_portifolio = 'https://drive.google.com/open?id=1J2NRI-js0MhcnMEkdT9yLScmoA6GdIV

fluff, id = link_portifolio.split('=')
print (id) # Verify that we have everything after '='

downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('portifolio.json')
portifolio_df = pd.read_json('portifolio.json', orient='records', lines=True)
```

```
# Import profile.json from google drive

link_profile = 'https://drive.google.com/open?id=19FWNvSjVeFMExM3vqokdI0ZHkcWI4LcB'

fluff, id = link_profile.split('=')
print (id) # Verify that we have everything after '='

downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('profile.json')
profile_df = pd.read_json('profile.json', orient='records', lines=True)
```

```
# Import transcript.json from google drive

link_transcript = 'https://drive.google.com/open?id=1N8NsO1UDMDYjhTX2J-UTLH5c32E-CHR

fluff, id = link_transcript.split('=')
print (id) # Verify that we have everything after '='

downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('transcript.json')
transcript_df = pd.read_json('transcript.json', orient='records', lines=True)
```

```
# Checking the dimensions of each dataset:

#Portifolio
print("Portifolio Dataset \n Variables:\t{}\n Inputs:\t{}".format(portifolio_df.shap
print("\n")
#Profile
print("Profile Dataset \n Variables:\t{}\n Inputs:\t{}".format(profile_df.shape[1],
print("\n")
#Transcript
print("Transcript Dataset \n Variables:\t{}\n Inputs:\t{}".format(transcript_df.shap
```

```
portifolio_df.head()
```

```
profile_df.head()
```

```
transcript_df.head()
transcript_df.head()
```

3 - Exploratory Data Analysis (EDA)

At this section, we're going to explore the given data, in order to learn and get some insights about
each other:

3.1 - Offers

There were sent 10 different offers during the experiment:

```
portifolio_df.head(10)
```

Distribution of Offer Types

There are three types of offers:

- ***bogo*** (buy one get one) : Costumers recieve an discount (*reward*), if a certain amount (*difficu*
- ***discount*** : Immediate discount (*reward*) over the product's price (*difficulty*);
- ***informational*** : No discount directly applied, only infomational data regarding an specific pro

```
# Distribution of offer types [%]
offer_type_dist = (portifolio_df['offer_type'].value_counts() / portifolio_df.shape[
offer_type_dist
```

```
# Countplot of Offer Types
sns.set()

plt.figure(figsize = (18,6));
plt.title('Distribution of Offer Types');
plt.xlabel('Count [%]');
plt.ylabel('Offer Types');
sns.countplot(x='offer_type', data=portifolio_df, palette="GnBu_d");
```

```
#
portifolio_df.groupby('offer_type')['difficulty'].mean()
```

```
# Violin plot - Distribution per offer type
plt.figure(figsize = (18,6));
plt.title('Difficulty distribution per type of offer');
sns.violinplot(x ='offer_type', y ='difficulty', data = portifolio_df);
sns.violinplot(x ='offer_type', y ='difficulty', data = portifolio_df);
```

```
portifolio_df.groupby('offer_type')['reward'].mean()
```

```
# Violin plot - Reward per offer type
plt.figure(figsize = (18,6));
plt.title('Reward distribution per type of offer');
sns.violinplot(x ='offer_type', y ='reward', data = portifolio_df);
```

```
portifolio_df.groupby('offer_type')['duration'].mean()
```

```
portifolio_df.groupby('offer_type')['duration'].median()
```

```
# Violin plot - Duration per offer type
plt.figure(figsize = (18,6));
plt.title('Duration distribution per type of offer');
sns.violinplot(x ='offer_type', y ='duration', data = portifolio_df);
```

3.2 Clients

```
profile_df.head()
```

Missing Values

Only *gender* and *income* features have missing values in this dataset. Curiously, they have exactly
worth to keep investigating:

```
# Checking for missing values [%]
(profile_df.isnull().sum() / profile_df.shape[0]) * 100
```

Investigating a little bit deeper, it's possible to realize that all the registers in which the *gender* info
in which are lacking the *income* information. Furthermore, also the *age* feature for these registers
age value for these ones is arround 118 years old.

Hence, it makes sense to remove all this registers from our dataset:

```
# Replacing null values with 'N'
profile_df['gender'] = profile_df['gender'].fillna('N')

# Checking age by gender
profile_df.groupby('gender')['age'].mean()
```

```
# Checking income by gender
profile_df.groupby('gender')['income'].mean()
```

```
profile_df.drop(profile_df[profile_df['gender'] == 'N'].index, axis=0, inplace=True)
profile_df.drop(profile_df[profile_df['gender'] == 'N'].index, axis=0, inplace=True)

profile_df = profile_df.reset_index(drop=True)

profile_df.head()
```

Gender

```
# Gender distribution among the clients
gndr_cnt = (profile_df['gender'].value_counts() / profile_df.shape[0])* 100
gndr_cnt
```

```
# Countplot of User's gender

plt.figure(figsize = (18,6));
plt.title('Distribution of User´s gender');
plt.xlabel('Count [%]');
plt.ylabel('Gender');
ax_gndr = sns.countplot(x='gender', data=profile_df, palette="GnBu_d");
```

Age

```
# Distribution plot
plt.figure(figsize = (18,6));
plt.title('Distribution of user´s age');
sns.distplot(profile_df['age'], color='blue');
```

```
# Violin plot - Age x gender
plt.figure(figsize = (18,6));
plt.title('Age per gender');
sns.violinplot(x ='gender', y ='age', data = profile_df);
```

Became_member_on

In order to have a easier way to work with this information later on, I'm going to calculate how mar
App instead of looking at the subscription data.

For doing that, one needs only to subtract the subscription date from today's date:

$$MembershipDays : date_{today} - date_{subscriptio}$$

```
# Today's date
today = pd.to_datetime(date.today())
today

# Convert values to datetime format
profile_df['became_member_on'] = pd.to_datetime(profile_df['became_member_on'], form


# Calculate the number of days the user has been using the app

membership_days = []
for i in range(profile_df.shape[0]):
  membership_days.append((today - profile_df['became_member_on'][i]).days)
```

```
profile_df['membership_days'] = membership_days

# Distribution plot
plt.figure(figsize = (18,6));
plt.title('Distribution of membership days');
sns.distplot(profile_df['membership_days'], color='blue');
```

```
# Violin plot - Membership time x gender
plt.figure(figsize = (18,6));
plt.title('Membership time per gender');
sns.violinplot(x ='gender', y ='membership_days', data = profile_df);
```

```
# Scatter plot: age x membership time
plt.figure(figsize = (18,6));
plt.title('Age x Membership time [days]');
sns.scatterplot(x ='age', y ='membership_days', data = profile_df);
```

Income

```
# Distribution plot
plt.figure(figsize = (18,6));
plt.title('Distribution of member´s income');
sns.distplot(profile_df['income'], color='blue');
```

```
# Average income by gender
profile_df.groupby('gender')['income'].mean()
```

```
# Violin plot - Income x Gender
plt.figure(figsize = (18,6));
plt.title('Income per gender');
sns.violinplot(x ='gender', y ='income', data = profile_df);
```

```
# Scatter plot - Age x income
plt.figure(figsize = (18,6));
plt.title('Age x Income');
sns.scatterplot(x ='age', y ='income', data = profile_df);
```

```
# Scatter plot - Membership time x income
plt.figure(figsize = (18,6));
plt.title('Membership time x Income');
sns.scatterplot(x ='membership_days', y ='income', data = profile_df);
```

3.3 Transactions

Missing Values

```
(transcript_df.isnull().sum() / transcript_df.shape[0]) * 100
```

Dropping transactions from users which had missing information

As I've have dropped the users which had missing data from the *profile dataframe*, I also have to d
from the *transcript dataframe*.

```
# Get the list of users who have a complete register
complete_register_users = profile_df['id'].unique()
# Get the list of unique users in transcript_df
user_list = transcript_df['person'].unique()
# Check which ones to drop
user_drop_list = []
for user in user_list:
  if user not in complete_register_users:
    user_drop_list.append(user)

print('Number of users with complete registers: {}'.format(len(complete_register_use
print('Number of unique users in the transcript data frame: {}'.format(len(user_list
print('Number of users to drop:{}'.format(len(user_drop_list)))
print('Percent of values to drop: {:.2f} %'.format((transcript_df[transcript_df['per
```

```
len(transcript_df[transcript_df['person'].isin(user_drop_list)].index)
```

```
# Drop users
transcript_df.drop(transcript_df[transcript_df['person'].isin(user_drop_list)].index
transcript_df = transcript_df.reset_index(drop=True)
```

Time

This feature tells us how many hours have passed until the moment of the described transaction s

The maximum value for this feature is **714** hours, that tells us that the experiment ran for approxin

```
transcript_df['time'].max()
```

```
plt.figure(figsize = (18,6));
plt.title('Time Distribution');
sns.distplot(transcript_df['time'], color='blue', bins=50);
```

```
# Distribution plot per type of offer
fig, ((axis1, axis2),(axis3,axis4)) = plt.subplots(2,2,figsize=(18,12))
fig.suptitle('Time distribution per type of offer')
```

```
axis1.set_title('transactions')
axis2.set_title('offer received')
axis3.set_title('offer viewed')
axis4.set_title('offer completed')
sns.distplot(transcript_df[transcript_df['event'] == 'transaction']['time'], color='
sns.distplot(transcript_df[transcript_df['event'] == 'offer received']['time'], colo
sns.distplot(transcript_df[transcript_df['event'] == 'offer viewed']['time'], color=
sns.distplot(transcript_df[transcript_df['event'] == 'offer completed']['time'], col
```

Type of transaction

```
(transcript_df['event'].value_counts() / transcript_df.shape[0]) *100
```

```
# Countplot of Event types

plt.figure(figsize = (18,6));
plt.title('Distribution of Event Types');
sns.countplot(x='event', data=transcript_df, palette="GnBu_d");
```

Value

The *value* column holds the information of which offer is related to each user iteraction, or if it's a

The information was stored as a dictionary of strings, and as only the offer id or the transaction va extract these values from the dictionary add a new column called *description*.

As I'm not going to use it anymore, I'm dropping the *value* column.

```
# Getting transaction amounts and offer ids from the dictionary

description = []
for idx in range(transcript_df.shape[0]):
  val = transcript_df['value'][idx].values()
  idx_ = 0
  for i in val:
    if idx_ > 0:
      break
    else:
      description.append(i)
      idx_ = idx_ + 1

# Create a new column called description and drop value column
transcript_df['description'] = description
transcript_df.drop(columns = 'value', axis=1, inplace=True)
transcript_df.head(5)
```

Iteractions

I'm going to create a new dataframe, called *iteractions_df*, which is going to have the information a
offers. The dataset will have the following features:

- **user_id** (string) - customer identification;
- **offer_id** (string) - offer identification;
    - *note* : It's possible that the same offer has been sent to a customer more than once
- **completed** (int)
    - 1: Offer has been completed;
    - 0: Offer has not been completed;
    - *note* : Offers can be completed without being seen
- **viewed** (int)
    - 1: Customer has acknowledge the offer;
    - 0: Customaer hasn't ackowledge the offer;
- **influenced_spent** (int) - Proportion of the value spent by the influence of this offer compared
  experiment period;
    - *note: Transactions were considered influenced by the offer only if they were made after t*
- **offer type** (str) - Type of offer:
    - bogo (buy one get one)
    - discount
    - informational
- **reward_ratio** (float) - offer reward / offer difficulty

- *note: reward_ratio is considered 0 for informational offers*

```
# Defines the end of time period to analyze and if the offer was completed or not

def offer_final_time(per_user_df, idx_offer, repeat_idx, recieve_time_, duration):

  '''
  INPUT:
  per_user_df - dataframe subset containing only the customers of interest's informa
  idx_offer - offer_id
  repeat_idx - multiple offer with the same id for the same cliente count
  recieve_time_ - offer recieved time
  duration - offer duration

  OUTPUT:
  completed_time - list of completed times for this offer id
  completed_time_ - completed time for offer [repeat_idx] with id = idx_offer
  offer_completed - offer was completed or not [1 -> yes, 0 -> no]

  Function defines the end of time period to analyze and if the offer was completed

  '''

  completed_time = per_user_df[(per_user_df['description'] == idx_offer) & (per_user
  if len(completed_time) > 0:
    if (repeat_idx+1) > len(completed_time):
      completed_time_ = recieve_time_ + duration
    else:
      completed_time_ = completed_time[repeat_idx]
  else:
    completed_time_ = recieve_time_ + duration

  # Offer completed or not
  if (recieve_time_ + duration) != completed_time_:
    offer_completed = 1
  else:
    offer_completed = 0

  return completed_time, completed_time_, offer_completed


# Check if the offer has been seen or not

def offer_seen(per_user_df,idx_offer,repeat_idx,recieve_time,completed_time,complete

  '''
  INPUT:
  per_user_df - dataframe subset containing only the customers of interest's informa
  idx_offer - offer_id
  repeat_idx - multiple offer with the same id for the same cliente count
  recieve_time - list of recieved times for the offer id of interest (idx_offer)
```

```
    completed_time -  list of completed times for the offer id of interest (idx_offer)
    completed_time_ - completed time for this offer
    nr_viewed_offers - number of offers confirmed as seen

    OUTPUT:
    offer_viewed_ - offer has been seen or not [1 -> yes, 0 -> no]
    nr_viewed_offers - number of offers confirmed to be seen
    time_viewed_ - time the offer has been seen
    '''

    offer_viewed = per_user_df[((per_user_df['event'] == 'offer viewed') & (per_user_d
    offer_viewed_ = 0
    time_viewed_ = 0
    if len(offer_viewed) == len(recieve_time):
      offer_viewed_ = 1
      time_viewed_ = offer_viewed[repeat_idx]
    else:
      if len(offer_viewed) == 0:
        offer_viewed_ = 0
      else:
        if nr_viewed_offers < len(offer_viewed):
          for time_viewed in offer_viewed:
            if (time_viewed <=  completed_time_ ) & (time_viewed >=  recieve_time_ ):
              offer_viewed_ = 1
              time_viewed_ = time_viewed
              nr_viewed_offers = nr_viewed_offers +1
            else:
              offer_viewed_ = 0

    return offer_viewed_, nr_viewed_offers, time_viewed_


  # Value spent by the influence of the offer

  def amount_spent(per_user_per_offer_df, repeat_idx, completed_time, offer_viewed_, t

    '''
    per_user_per_offer_df - dataframe subset containing only the customers of interest
    repeat_idx - multiple offer with the same id for the same cliente count
    completed_time - list of completed times for this offer index (idx_offer)
    offer_viewed_ - offer has been seen or not
    time_viewed_ - time offer has been seen

    OUTPUT:

    influenced_value_spent - amount spent by the influence of the offer

    '''

    # Value spent by the influence of the offer
    influenced_value_spent = 0

    if (len(completed_time) > 1) & (repeat_idx<len(completed_time)) :
      if offer_viewed_ == 1:
        influenced_value_spent = per_user_per_offer_df[(per_user_per_offer_df['event']
                                 & (per_user_per_offer_df['time'] > completed_time[repe
                                 & (per_user_per_offer_df['time'] >= time_viewed_)]['de
    else:
      if offer_viewed_ == 1:
        influenced_value_spent = per_user_per_offer_df[(per_user_per_offer_df['event']
                                 & (per_user_per_offer_df['time'] >= time_viewed_)]['de


    return influenced_value_spent
```

```
portifolio_df.head()
```

```
# List of unique users in the transcript dataframe
unique_ids = transcript_df['person'].unique()

# Lists to store the features
user_list = []
offer_list = []
offer_completed_list = []
offer_viewed_list = []
influenced_spent_list = []
offer_type_list = []
reward_ratio_list = []

#Loop through all the unique customers
for idx_user in unique_ids:

  # Create a subset of transcript_df holding only the information of the customer of
  per_user_df = []
  per_user_df = transcript_df[transcript_df['person'] == idx_user]

  # Get unique values for offers to that user
  unique_offers = per_user_df[per_user_df['event'] != 'transaction']['description'].

  # Total value spent by this user during the experiment
  total_ever_spent = per_user_df[per_user_df['event'] == 'transaction']['description

  #Loop trhough all the offers this customer have interections with
```

```
      #Loop through all the offers this customer have interections with
      for idx_offer in unique_offers:

        # Get offer type
        offer_type = portifolio_df[portifolio_df['id'] == idx_offer]['offer_type'].value

        #Get offer duration
        duration = int(portifolio_df[portifolio_df['id'] == idx_offer]['duration'])*24

        #Get offer difficulty and reward
        difficulty = int(portifolio_df[portifolio_df['id'] == idx_offer]['difficulty'])
        reward = int(portifolio_df[portifolio_df['id'] == idx_offer]['reward'])

        # Define initial time (delta t)
        recieve_time = per_user_df[(per_user_df['description'] == idx_offer) & (per_user

        # Loop for multiple offers with the same id
        nr_viewed_offers = 0

        for repeat_idx in range(len(recieve_time)):
          recieve_time_ = recieve_time[repeat_idx]

          #Define final time of delta_t
          completed_time, completed_time_, offer_completed = offer_final_time(per_user_d

          # Create a dataframe subset on the time period of interest
          per_user_per_offer_df = per_user_df[((per_user_df['description'] == idx_offer)

          # Offer viewed or not
          offer_viewed_, nr_viewed_offers, time_viewed_ = offer_seen(per_user_df,idx_off

          # Value spent by the influence of the offer
          influenced_value_spent = amount_spent(per_user_per_offer_df, repeat_idx, compl

          # Append values to the lists
          user_list.append(idx_user)
          offer_list.append(idx_offer)
          offer_completed_list.append(offer_completed)
          offer_viewed_list.append(offer_viewed_)
          if total_ever_spent == 0:
            influenced_spent_list.append(0)
          else:
            influenced_spent_list.append(influenced_value_spent/total_ever_spent)
          offer_type_list.append(offer_type)
          if difficulty == 0:
            reward_ratio_list.append(0)
          else:
            reward_ratio_list.append(reward/difficulty)



    # Create a dataframe with the arrays obtained

    interactions_df = (pd.DataFrame(np.array([user_list,offer_list,offer_completed_list,o
    iteractions_df = (pd.DataFrame(np.array([user_list,offer_list,offer_completed_list,o
    iteractions_df.columns = (['user_id','offer_id','completed','viewed','influenced_spe

    iteractions_df.head()
```

```
# Convert both completed and viewed columns to integer or float
iteractions_df['completed'] = iteractions_df['completed'].astype(int)
iteractions_df['viewed'] = iteractions_df['viewed'].astype(int)
iteractions_df['influenced_spent'] = iteractions_df['influenced_spent'].astype(float
iteractions_df['reward_ratio'] = iteractions_df['reward_ratio'].astype(float)
```

```
# Percent of completed offers by type
(iteractions_df.groupby('offer_type')['completed'].mean())*100
```

```
# Percent of viewed offers by type
(iteractions_df.groupby('offer_type')['viewed'].mean())*100
```

```
iteractions_df[iteractions_df['viewed'] == 1].groupby('offer_type')['influenced_spen
```

```
(iteractions_df[iteractions_df['completed'] == 1].groupby('offer_type')['viewed'].me
```

4 - Customer Labeling

In order to being able to use a supervised machine learning algorithim to model which type of offe
user, one needs first to label customers somehow.

I've chosen to use the iteractions customers had with the offers sent to them during the experimer
going to enable me to create labels for effectiveness of that type of offer:

4.1 - Offer Metrics

Based on the data of customer x offer iteractions obtained above, now I'm going to calculate an of
measure how effective that kind of offer was to a particular user. The metric is given by the followi

$$OfferScore_{OfferType} = \pi_{completed_{offertype}} + [1 - (\pi_{completed_{offertype}} - \pi_{viewed_{offertype}})] +$$

where:

- Completed Ratio

$$\pi_{completed} = \frac{offers_{completed}}{offers_{sent}}$$

- Viewed Ratio

$$\pi_{viewed} = \frac{offers_{viewed}}{offers_{sent}}$$

- Influenced Value Spent

$$V_{influenced} = \frac{V_{offer_{seen}}}{V_{total}}$$

- Reward Ratio

$$\phi_{reward} = \frac{reward_{ratio_{completed}}}{reward_{ratio_{sent}}}$$

note: An offer is only considered completed for this formula if it has been acknowledge by the custo

note: Informational offers have reward ratio considered as 0, as they have no difficulty or reward;

```
iteractions_df[iteractions_df['user_id'] == '78afa995795e4d85b5d9ceeca43f5fef']
```

```
#Calculate metrics for each offer type by user
def offer_meterics(idx_user, df, offer_type):
  '''
  INPUT:
  idx_user - costumer offers are related to
  df - iteractions dataframe (iteractions_df)
  offer_type - type of offer to check (bogo, discount or informational)

  OUTPUT:
  offer_metric - score of metric type for each user

  function uses the following features to calculate the metric:

  offers_sent - number of 'offer_type' offers sent
  offers_viewed - number of 'offer_type' viewed / number of 'offer_type' offers sent
  offers_completed - number of 'offer_type' viewed and completed / number of 'offer_
  influenced_value - value spent by the influence of the offer / total value spent b
  offer_reward_ratio - offer reward value / offer difficulty value

  '''

  offers_sent = df[(df['user_id'] == idx_user) & (df['offer_type'] == offer_type)].s
  offers_viewed = df[(df['user_id'] == idx_user) & (df['offer_type'] == offer_type)
  if offer_type == 'informational':
    offers_completed = 0
  else:
    offers_completed = df[(df['user_id'] == idx_user) & (df['offer_type'] == offer_t
  influenced_value = df[(df['user_id'] == idx_user) & (df['offer_type'] == offer_typ
  offer_reward_ratio_completed = df[(df['user_id'] == idx_user) & (df['offer_type']
  offer_reward_ratio_sent = df[(df['user_id'] == idx_user) & (df['offer_type'] == of

  # Offer viewed / sent ratio (if offers were sent)
  if offers_sent != 0:
    offers_viewed = offers_viewed/offers_sent

  # Offer completed & viewed / viewed ration (if offers were viewed)
  if offers_viewed != 0:
    offers_completed = offers_completed / offers_sent

  # Offer reward_ratio
  if offer_reward_ratio_sent == 0:
    reward_ratio = 0
  else:
    reward_ratio = offer_reward_ratio_completed / offer_reward_ratio_sent

  # Calculate offer type score
  offer_metric = offers_completed + (1 - (offers_completed - offers_viewed)) + (1 +

  return offer_metric


# List of unique user ids
# List of unique user_ids
unique_ids = iteractions_df['user_id'].unique()

# List to store the metrics for each user
bogo_metrics_list = []
disc_metrics_list = []
info_metrics_list = []

# Get offer metrics to each user
for idx_user in unique_ids:

  #bogo
  bogo_metric = offer_meterics(idx_user, iteractions_df, 'bogo')

  #discount
  disc_metric = offer_meterics(idx_user, iteractions_df, 'discount')

  #informational
  info_metric = offer_meterics(idx_user, iteractions_df, 'informational')

  #append normalized values to the lists
  bogo_metrics_list.append(bogo_metric)
  disc_metrics_list.append(disc_metric)
  info_metrics_list.append(info_metric)



#Create the metrics_df
metrics_df = (pd.DataFrame([unique_ids,bogo_metrics_list,disc_metrics_list,info_metr
metrics_df.columns = (['user_id','bogo','disc','info'])


metrics_df.head(10)
```

In order to keep the same range for each offer type score, I'm going to normalize them by it's maxi

Hence, now each score is contained in the range [0 , 1] :

```
# Normalize each offer type score by it's maximum value (metric value [0,1])
metrics_df['bogo'] = metrics_df['bogo']/metrics_df['bogo'].max()
metrics_df['disc'] = metrics_df['disc']/metrics_df['disc'].max()
metrics_df['info'] = metrics_df['info']/metrics_df['info'].max()
```

4.2 - Labeling Criteria

We have now normalized metrics for all the three offer types in respect to each user, which enable
of offer as effective or not.

As a customer could well be influenced by more than one type of offer, I'm going for **soft label** stra

- *Soft label* - One customer may be labled in more than one offer type;

First of all, it's interesting to have a look on how these scores are distributed:

```
# Distribution plot per metric type
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Offer metrics distribution')
sns.distplot(metrics_df['bogo'], color='blue', ax=axis1 );
sns.distplot(metrics_df['disc'], color='blue', ax=axis2);
sns.distplot(metrics_df['info'], color='blue', ax=axis3);
```

4.2.1 - Gaussian Mixture Model

The first idea to classify our costumers, is to use an unsupervised machine learning algorithim, tha

One good option is the *Gaussian Mixture Model*.

The ideia is to identify **3 clusters** (one for each type of offer) in a 3D dimensional space - *a cube of respective offer type*, and identify to which cluster a respective costumer is more probable to belor

There's a really good article, written by **Oscar Contreras Carrasco**, which explains this algorithm fr

```
# Creating the predictors matrix X_gmm
X_gmm_df = metrics_df.copy()
X_gmm_df.drop('user_id', axis=1, inplace=True)
X_gmm = (X_gmm_df.values).astype(float)


# Fitting and Predicting the Gaussian Mixture Model
gmm = GaussianMixture(n_components=3)
gmm.fit(X_gmm)
proba_lists = gmm.predict_proba(X_gmm)


#Plotting the results as an 3D figure
colored_arrays = np.matrix(proba_lists)
colored_tuples = [tuple(i.tolist()[0]) for i in colored_arrays]
fig = plt.figure(1, figsize=(7,7))
ax = Axes3D(fig, rect=[0, 0, 0.95, 1], elev=48, azim=134)
ax.scatter(X_gmm[:,2], X_gmm[:,0], X_gmm[:,1], c=colored_tuples, edgecolor="k", s=50
ax.set_xlabel("informational offers")
ax.set_ylabel("bogo offers")
ax.set_zlabel("discount offers")
#ax.plot([], [], 'o', c=self.clusters[i].color, label='Cluster' + str(i+1))
plt.title("Gaussian Mixture Model", fontsize=14);
```

```
# Matrix of probabilities
proba_lists
```

```
# Predictors Matrix X_gmm
X_gmm
```

No fancy tests were needed to conclude that this algorithm have found some pattern, but it isn't ex

The 3 identified clusters do not represent specifically the effectiviness of an offer type for a partic

It's actually difficult to identify exactly what each cluster represents, so I'm going to run another cla for.

4.2.2 Quantile Classification

In order to have a clear classification by the effectiviness of offers, where it's possible for a user tc as I have stated before, one good option is to use a quantile classification approach.

Shortly, if the metric for an specific offer type is among the top **Q**% of the top scores for that metri considered effective for that particular customer.

In order to define this threshold, one uses the **quantile(1-Q)**.

However, it's important to bear in mind that for low values of **Q** it is quite probable that some costu I'm going to use this classification labels on a supervised machine learning algorithm ahead, and r charactheristics are the most determinant for the effectiviness of an offer type, having a lot of use would like avoid.

In the other hand, high values of **Q** would lead my classification to be less effective, so there's a tra

```
def calculate_threshold (df, q):

  '''
  INPUT
```

```
    df - metrics_df
    q - top % metrics

    OUTPUT
    bogo_threshold - threshold value for quantile(1-q) for bogo offers
    disc_threshold - threshold value for quantile(1-q) for bogo offers
    info_threshold - threshold value for quantile(1-q) for bogo offers
    '''

    bogo_threshold = df['bogo'].quantile(1-q)
    disc_threshold = df['disc'].quantile(1-q)
    info_threshold = df['info'].quantile(1-q)

    return bogo_threshold, disc_threshold, info_threshold


  def label_classifier(df, bogo_threshold, disc_threshold, info_threshold):

    bogo_labels = ((df['bogo'].values > bogo_threshold)).astype(np.int_)
    disc_labels = ((df['disc'].values > disc_threshold)).astype(np.int_)
    info_labels = ((df['info'].values > info_threshold)).astype(np.int_)


    return bogo_labels, disc_labels, info_labels


  quantile_threshold = 0.4
  nr_bogo_labels = []
  nr_disc_labels = []
  nr_info_labels = []
  threshold_list = []

  for i in range(100):

    bogo_threshold, disc_threshold, info_threshold = calculate_threshold (metrics_df,
    bogo_labels, disc_labels, info_labels = label_classifier(metrics_df, bogo_threshol

    nr_bogo_labels.append(bogo_labels.sum())
    nr_disc_labels.append(disc_labels.sum())
    nr_info_labels.append(info_labels.sum())
    threshold_list.append(100 -((1-quantile_threshold)*100))
    quantile_threshold = quantile_threshold + 0.005




  plt.figure(figsize = (18,6));
  plt.title(' Number of labeled customers per offer category x Quantile Threshold');
  plt.ylabel('Labeled Customers')
  plt.xlabel(' Top Q % scores ')
  sns.scatterplot(x = threshold_list, y = nr_bogo_labels);
  sns.scatterplot(x = threshold_list, y = nr_disc_labels);
  sns.scatterplot(x = threshold_list, y = nr_info_labels);
  plt.legend(labels=['bogo','discount','informational']);
```

```
# Calculate thresholds for Q = 0.5
quantile_threshold = 0.516
bogo_threshold, disc_threshold, info_threshold = calculate_threshold (metrics_df, qu
print('Threshold BOGO offers: {}'.format(bogo_threshold))
print('Threshold Discount offers: {}'.format(disc_threshold))
print('Threshold Informational offers: {}'.format(info_threshold))
```

```
# Distribution plot per metric type + Quantile threshold
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Offer metrics distribution | Quantile threshold = 0.516')
sns.distplot(metrics_df['bogo'], color='blue', ax=axis1 );
axis1.axvline(bogo_threshold, c='red');
sns.distplot(metrics_df['disc'], color='blue', ax=axis2);
axis2.axvline(disc_threshold, c='red');
sns.distplot(metrics_df['info'], color='blue', ax=axis3);
axis3.axvline(info_threshold , c='red');
```

The graphs above shows us that informational offers had it's score equals to zero to approximatel
choose the maximum value of **Q** that considers only scores higher than zero to all the three metric

Hence,

Q = 51.6

```
# Customer classification using the chosen threshold

bogo_labels, disc_labels, info_labels = label_classifier(metrics_df, bogo_threshold,
```

4.2.3 Label_df

As an output of the last steps, we have a new dataframe, called label_df, which is going to be used
learning algorithm on the next step:

This new dataframe has the following columns:

- **id** - customer id
- **label_bogo** -indicates that bogo offers were efficient to the customer
  - 1 - Offer type has been efficient
  - 0 - Offer type has not been efficient
- **label_disc** - indicates that discount offers were efficient to the customer
  - 1 - Offer type has been efficient
  - 0 - Offer type has not been efficient
- **label_info** - indicates that informational offers were efficient to the customer
  - 1 - Offer type has been efficient
  - 0 - Offer type has not been efficient
- **prob_cluster_1** - probability of user to belong to cluster #1 (Gaussian Mixture Model)
- **prob_cluster_2** - probability of user to belong to cluster #2 (Gaussian Mixture Model)
- **prob_cluster_3** - probability of user to belong to cluster #3 (Gaussian Mixture Model)
- **labels_nr** - number of offer types that have been efficient with this user

```
#Create the labels_df (labels + gmm probabilities)
label_df = (pd.DataFrame([unique_ids,bogo_labels, disc_labels, info_labels,proba_lis
label_df.columns = (['id','label_bogo','label_disc','label_info','prob_cluster_1','p
label_df['labels_nr'] = label_df['label_bogo'] + label_df['label_disc'] + label_df['

label_df.head()
```

```
# Percent of customers that haven't been label to category
drop_rate = (label_df[label_df['labels_nr'] == 0].shape[0] / label_df.shape[0])*100
print(' {:.2f} % of customers were not labeled'.format(drop_rate))
```

As I stated before, I'm going to drop these customers from the dataset, as we are looking to be abl
labels.

```
# Drop customers that haven't been labeled
label_df.drop(label_df[label_df['labels_nr'] == 0].index, axis=0, inplace=True)

# Drop labels_nr column, as it's no longer necessary
label_df.drop(columns='labels_nr', axis=1, inplace=True)

# Merge label_df with profile_df by the primary key 'id'
customer_df = label_df.merge(profile_df, how='inner')
# 'Became_member_on' column is no longer necessary as we have 'membership_days'
customer_df.drop(columns='became_member_on', axis=1, inplace=True)

customer_df.head()
```

Looking for explicit patterns

I'm going to check if there are some explicit patterns on the data when we look only at customers

```
# Cluster 1 distribution per user group
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Cluster 1')
axis1.title.set_text('label_bogo = 1')
axis2.title.set_text('label_disc = 1')
axis3.title.set_text('label_info = 1')
sns.distplot(customer_df[customer_df['label_bogo'] == 1]['prob_cluster_1'], color='b
sns.distplot(customer_df[customer_df['label_disc'] == 1]['prob_cluster_1'], color='b
sns.distplot(customer_df[customer_df['label_info'] == 1]['prob_cluster_1'], color='b
```

```
# Cluster 2 distribution per user group
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Cluster 2')
axis1.title.set_text('label_bogo = 1')
axis2.title.set_text('label_disc = 1')
axis3.title.set_text('label_info = 1')
sns.distplot(customer_df[customer_df['label_bogo'] == 1]['prob_cluster_2'], color='b
sns.distplot(customer_df[customer_df['label_disc'] == 1]['prob_cluster_2'], color='b
sns.distplot(customer_df[customer_df['label_info'] == 1]['prob_cluster_2'], color='b
# Cluster 3 distribution per user group
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Cluster 3')
axis1.title.set_text('label_bogo = 1')
axis2.title.set_text('label_disc = 1')
axis3.title.set_text('label_info = 1')
sns.distplot(customer_df[customer_df['label_bogo'] == 1]['prob_cluster3'], color='bl
sns.distplot(customer_df[customer_df['label_disc'] == 1]['prob_cluster3'], color='bl
sns.distplot(customer_df[customer_df['label_info'] == 1]['prob_cluster3'], color='bl
```

```
# age distribution per user group
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Age')
axis1.title.set_text('label_bogo = 1')
axis2.title.set_text('label_disc = 1')
axis3.title.set_text('label_info = 1')
sns.distplot(customer_df[customer_df['label_bogo'] == 1]['age'], color='blue', ax=ax
sns.distplot(customer_df[customer_df['label_disc'] == 1]['age'], color='blue', ax=ax
sns.distplot(customer_df[customer_df['label_info'] == 1]['age'], color='blue', ax=ax
# income distribution per user group
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Income')
axis1.title.set_text('label_bogo = 1')
axis2.title.set_text('label_disc = 1')
axis3.title.set_text('label_info = 1')
sns.distplot(customer_df[customer_df['label_bogo'] == 1]['income'], color='blue', ax
sns.distplot(customer_df[customer_df['label_disc'] == 1]['income'], color='blue', ax
sns.distplot(customer_df[customer_df['label_info'] == 1]['income'], color='blue', ax
```

```
# membership days distribution per user group
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Income')
axis1.title.set_text('label_bogo = 1')
axis2.title.set_text('label_disc = 1')
axis3.title.set_text('label_info = 1')
sns.distplot(customer_df[customer_df['label_bogo'] == 1]['membership_days'], color='
sns.distplot(customer_df[customer_df['label_disc'] == 1]['membership_days'], color='
sns.distplot(customer_df[customer_df['label_info'] == 1]['membership_days'], color='
# gender distribution per user group
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Gender')
axis1.title.set_text('label_bogo = 1')
axis2.title.set_text('label_disc = 1')
axis3.title.set_text('label_info = 1')
sns.countplot(x='gender', data=customer_df[customer_df['label_bogo'] == 1], palette=
sns.countplot(x='gender', data=customer_df[customer_df['label_disc'] == 1], palette=
sns.countplot(x='gender', data=customer_df[customer_df['label_info'] == 1], palette=
```

No explicit patterns were identified with an visual analysis.

Hence, the next step is trying to find some patterns using social and demographic data with a sup

5 - Data Pre-Processing

Now that transaction, demographic and offer data are combined into a single dataframe (*custome* demographic groups respond best to each offer type, which is the main goal of this project:

```
customer_df.head()
```

5.1 Feature Engineering

We have as demographic and social information the following features:

- Age
- Income
- Membership Days
- Gender

All of them are numerical features, but *Gender*, which is already split into three categories (M for m

I'm going to convert all the three numerical features into categorical features, according to the foll

- **Age**
  - Young Customers
    - age < 30
  - Adult Customers
    - age >= 30 and age < 60
  - Senior Customers
    - age > 60
- **Income**
  - Income Level 1
    - income < 60.000
  - Income Level 2
    - income >= 60.000 and income < 100.000
  - Income Level 3
    - income > 100.000
- **Membership Days**
  - New Customers
    - membership_days < 1000
  - Regular Customers
    - membership_days >= 1000 and membership_days < 1740
  - Legacy Customers
    - income > 1740

```
# Customer Age Distribution and Category Thresholds
plt.figure(figsize = (18,6));
plt.title('Customer Income Distribution');
sns.distplot(customer_df['age'], color='blue', bins=50);
plt.axvline(30, c='red');
plt.axvline(60, c='red');
```

```
# Converert Age data to categories
customer_df['age'] = customer_df['age'].apply(lambda x:'age_group_1' if x < 30 else
                                              'age_group_2' if (x>=30 and x< 60) els
                                              'age_group_3')
```

```
# Customer Income Distribution and Category Thresholds
plt.figure(figsize = (18,6));
plt.title('Customer Income Distribution');
sns.distplot(customer_df['income'], color='blue', bins=50);
plt.axvline(60000, c='red');
plt.axvline(100000, c='red');
```

```
# Converert Age data to categories
customer_df['income'] = customer_df['income'].apply(lambda x:'income_group_1' if x <
                                              'income_group_2' if (x>=60000 and x< 1
                                              'income_group_3')
```

```
# Customer Membership Time Distribution and Category Thresholds
plt.figure(figsize = (18,6));
plt.title('Customer Income Distribution');
sns.distplot(customer_df['membership_days'], color='blue', bins=50);
plt.axvline(1000, c='red');
plt.axvline(1740, c='red');
```

```
# Converert Membership Time data to categories
customer_df['membership_days'] = customer_df['membership_days'].apply(lambda x:'memb
                                              'membership_group_2' if (x>=1000 and x
                                              'membership_group_3')
```

Target and Predictor Variables

Here I'm going to define my target and predictor variables, which I'm going to use in a supervised r

- Target Variable - *y*
  - label_bogo;
  - label_disc;
  - label_info;
- Predictor Variable - *X*
  - gender
  - age
  - income
  - membership_days

```
# Define y
y = customer_df[['label_bogo', 'label_disc', 'label_info']].values.astype(int)
y
```

```
# Define X
X = customer_df[['gender','age','income','membership_days']]
```

Pre Processing X Categorial Features

In order to prepare the data for the further steps, I'm going to create dummy variables to represent

As each feature has three different categories, two dummie variables shall be generated for each f

```
# Creating dummy variables
gender_dummies = pd.get_dummies(X['gender'], prefix='gender',drop_first=True)
age_dummies = pd.get_dummies(X['age'],drop_first=True)
income_dummies = pd.get_dummies(X['income'],drop_first=True)
membership_dummies = pd.get_dummies(X['membership_days'],drop_first=True)

# concatenate dummie variables into the dataframe
X = pd.concat([X, gender_dummies, age_dummies, income_dummies, membership_dummies],
# drop gender column, as it's no longer necessary
X.drop(['gender','age','income','membership_days'], axis=1, inplace=True)
```

```
# Check predictor variable X
X.head()
```

```
#Matrix X
X.values
```

6 - Modeling

Now that our data is prepared, one is able to start building the supervised machine learning model

As the target variable *y* is an array of size 3, where each position representing the label of each off
customers into this categories based on the social and demographic data contained in the predict

As there are three types of offers, and custumers may be labeled in more than one category, this is
problem.

Before going deeper in the model itself, it's necessary to split our data into training, validation and

Training and Test sets

```
# Split data into traning and test sets
np.random.seed(2)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_stat
```

I've chosen to use the **OneVsRest** classification strategy, using **Linear SVC** as estimator.

On the top of that, I'm using the **MultiOutputClassifier** in order to enable the model to make multi-l

Model hyperparameters will be optimized using **GridSearchCV**, according to the metrics defined in

- Accuracy
- Average_Precision (chosen as the metric for refit)

```
#Define the Machile Learning Pipeline
pipeline = Pipeline([
        ('clf',MultiOutputClassifier(OneVsRestClassifier(LinearSVC(), n_jobs=1)))
                        ])
# Define hyperparameters to be optimized in the GridSearchCV
```

```
parameters = {
    'clf__estimator__estimator__loss': ('squared_hinge', 'hinge'),
    'clf__estimator__estimator__multi_class': ('ovr','crammer_singer'),
    'clf__estimator__estimator__max_iter': (500,1000,2000,5000)
             }

# Define scoring metrics for the GridSearchCV optimization
scoring = {
    'accuracy' : make_scorer(accuracy_score),
    'average_precision' : make_scorer(average_precision_score)
        }

# Optimize and Fit
cv = GridSearchCV(pipeline, param_grid=parameters, verbose=2, cv=5, scoring=scoring,
cv.fit(X_train, y_train)
```

I'm going to check which were the optimized parameters found using the GridSearchCV method:

```
# Optimized Hyperparameters
optimized_model = cv.best_estimator_
print(optimized_model)
```

7 - Model Evaluation

After fitting our training data to the classification model, and having the hyperparameters optimize
check how good the model is by identifying which offer type is more effective based demographic

For that, I'm going to make predictions using the test dataset (X_test), and compare it to the respe

As I've chosen to label customers using a quantile approach, our categories are well balanced. He
the classification report is **precision**:

```
# Predictions using the test dataset
y_hat = optimized_model.predict(X_test)

# Model evaluation
```

```
print(classification_report(y_test, y_hat, target_names=['bogo','discount','infomati
```

8 - Result Discussion and Conclusions

As shown on the classification report above, our model hasn't an outstanding performance classif
based on demographic information.

As the classification step is completely dependent of the labeling step, here are some thoughts on
to achive better results:

8.1 Offer Scores

The equation used to calculate the offers score was chosen to have zero as minimum value. In otl
unresponded or uncompleted offers. This simple fact resulted in a huge concentration of custome
metrics, and our labeling criteria could not extract much information from this.

The output is that less customers were able to be labeled, as the minimum score (were most of cu
categories) has to be ignored in any score selection criteria.

```
# Distribution plot per metric type + Quantile threshold
fig, (axis1, axis2,axis3) = plt.subplots(1,3,figsize=(18,4))
fig.suptitle('Score Distribution - Concentration at Score = Zero')
sns.distplot(metrics_df['bogo'], color='blue', ax=axis1 );
sns.distplot(metrics_df['disc'], color='blue', ax=axis2);
sns.distplot(metrics_df['info'], color='blue', ax=axis3);
```

Adding a penalty factor to the metrics equation would help to distinguish this customers by unders
and then, adding much more information to the labeling criteria step.

8.2 Iteractions Data

On the other hand, the offer metrics are 100% dependent of the iteractions between customers an

Despite the fact that metrics should had a penalty factor, this strategy should perform better and b

As one have observed during the EDA Section, this experiment have gathered one month of data. A
would help the offer scores distribution to be more spread, and then being easier to chose thresho

8.3 Conclusions

The results achieved in this project were not enough for deploying the model into the Starbucks ap
method used here with different **offer score equations** and using data from **wider time frames**.