Starbucks Capstone Challenge

Introduction

This data set contains simulated data that mimics customer behavior on the Starbucks rewards mobile app. Once every few days, Starbucks sends out an offer to users of the mobile app. An offer can be merely an advertisement for a drink or an actual offer such as a discount or BOGO (buy one get one free). **Some users might not receive any offer during certain weeks.**

Not all users receive the same offer, and that is the challenge to solve with this data set.

Your task is to combine transaction, demographic and offer data to determine which demographic groups respond best to which offer type. This data set is a simplified version of the real Starbucks app because the underlying simulator only has one product whereas Starbucks actually sells dozens of products.

Every offer has a validity period before the offer expires. As an example, a BOGO offer might be valid for only 5 days. **You'll see in the data set that informational offers have a validity period even though these ads are merely providing information about a product;** for example, if an informational offer has 7 days of validity, you can assume the customer is feeling the influence of the offer for 7 days after receiving the advertisement.

You'll be given transactional data showing user purchases made on the app including the timestamp of purchase and the amount of money spent on a purchase. This transactional data also has a record for each offer that a user **receives** as well as a record for when a user actually **views** the offer. There are also records for when a user **completes** an offer.

Keep in mind as well that **someone using the app might make a purchase through the app without having received an offer or seen an offer.**

Example

To give an example, a user could receive a discount offer buy 10 dollars get 2 off on Monday. The offer is valid for 10 days from receipt. If the customer accumulates at least 10 dollars in purchases during the validity period, the customer completes the offer.

However, there are a few things to watch out for in this data set. Customers do not opt into the offers that they receive; in other words, **a user can receive an offer, never actually view the offer, and still complete the offer.** For example, a user might receive the "buy 10 dollars get 2 dollars off offer", but the user never opens the offer during the 10 day validity period. The customer spends 15 dollars during those ten days. There will be an offer completion record in the data set; however, the customer was not influenced by the offer because the customer never viewed the offer.

Cleaning

This makes data cleaning especially important and tricky.

You'll also want to take into account that some demographic groups will make purchases even if they don't receive an offer. From a business perspective, if a customer is going to make a 10 dollar purchase without an offer anyway, you wouldn't want to send a buy 10 dollars get 2 dollars off offer. **You'll want to try to assess what a certain demographic group will buy when not receiving any offers.**

Final Advice

Because this is a capstone project, you are free to analyze the data any way you see fit. For example, you could build a **machine learning model that predicts how much someone will spend based on demographics and offer type**. Or you could build **a model that predicts whether or not someone will respond to an offer**. Or, you don't need to build a machine learning model at all. You could develop a set of **heuristics that determine what offer you should send to each customer** (ie 75 percent of women customers who were 35 years old responded to offer A vs 40 percent from the same demographic to offer B, so send offer A).

Data Sets

The data is contained in three files:

- portfolio.json - containing offer ids and meta data about each offer (duration, type, etc.)
- profile.json - demographic data for each customer
- transcript.json - records for transactions, offers received, offers viewed, and offers completed

Here is the schema and explanation of each variable in the files:

portfolio.json

- id (string) - offer id
- offer_type (string) - type of offer ie BOGO, discount, informational
- difficulty (int) - minimum required spend to complete an offer
- reward (int) - reward given for completing an offer
- duration (int) -
- channels (list of strings)

profile.json

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

transcript.json

- event (str) - record description (ie transaction, offer received, offer viewed, etc.)
- person (str) - customer id
- time (int) - time in hours. The data begins at time t=0
- value - (dict of strings) - either an offer id or transaction amount depending on the record

I. Business Understanding

Summarizing the above Introduction, we are going to:

- Combine transaction, demographic and offer data to **analyze** which demographic groups respond(i.e.view&complete) best to which offer type;
- Build a supervised learning model(specifically, a classification model) that predicts whether or not someone will respond to an offer

II. Data Understanding and Data Engineering

In [ ]:
```
import pandas as pd
import numpy as np
import math
import json
%matplotlib inline

# read in the json files
portfolio = pd.read_json('data/portfolio.json', orient='records', lines=True)
profile = pd.read_json('data/profile.json', orient='records', lines=True)
transcript = pd.read_json('data/transcript.json', orient='records', lines=True)
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
import seaborn as sns
```

1. Portfolio Data Preprocessing

In [ ]:
```
#create dummy columns
portfolio=portfolio.join(portfolio['channels'].str.join('|').str.get_dummies().add_prefix('channel_'))
```

In [ ]:
```
portfolio=portfolio.drop('channels',axis=1)
```

In [ ]:
```
#10 kinds of offers
portfolio
```

2. Profile Visualizations

In [ ]:
```
profile.sample(5)
```

In [ ]:
```
#Age Visualization
count_by_agegroup=profile.groupby(pd.cut(profile['age'], np.arange(0, 118+5, 5)))['id'].count()
```

In [ ]:
```
sns.set()
plt.figure(figsize=(12,4))
plt.bar(np.arange(0, 118, 5),count_by_agegroup,width=3.5, align='edge', color = (0.5, 0.1, 0.5, 0.6 ))
plt.xticks(np.arange(0, 118, 5))
plt.xlabel('Age')
plt.ylabel('Persons Counts')
plt.title('The Persons Number of Different Age')
plt.show()
```

In [ ]:
```
count_by_incomegroup=profile.groupby(pd.cut(profile['income'], np.arange(20000, 120000+10000, 10000)))['id'].count()
```

In [ ]:
```
count_by_incomegroup.plot.pie(figsize=(6, 6))
```

In [ ]:
```
def make_autopct(values):
    def my_autopct(pct):
        total = sum(values)
        val = int(round(pct*total/100.0))
        return '{p:.2f}%  ({v:d})'.format(p=pct,v=val)
    return my_autopct

fig, ax = plt.subplots(figsize=(8, 8), subplot_kw=dict(aspect="equal"))

labels=['(20000, 30000]','(30000, 40000]','(40000, 50000]','(50000, 60000]','(60000,
 70000]',\
       '(70000, 80000]','(80000, 90000]','(90000, 100000]','(100000, 110000]','(11000
0, 120000]']

porcent = 100.*count_by_incomegroup/count_by_incomegroup.sum()

patches, texts, autotexts = ax.pie(count_by_incomegroup, autopct=make_autopct(count_b
y_incomegroup),startangle=90,counterclock=False)

labels = ['{0} - {1:1.2f} %'.format(i,j) for i,j in zip(labels, porcent)]

'''
sort_legend = True
if sort_legend:
    patches, labels, dummy =  zip(*sorted(zip(patches, labels, count_by_incomegroup),
                                          key=lambda x: x[2],
                                          reverse=True))
'''
ax.legend(patches, labels,
          title='The Persons Number of Different Income',
          loc="center left",
          bbox_to_anchor=(1, 0, 0.5, 1))
plt.title('The Persons Number of Different Income')
#plt.legend(patches, labels, loc="lower right", bbox_transform=plt.gcf().transFigure)
#plt.subplots_adjust(left=0.0, bottom=0.1, right=0.45)
```

Comments: We can see that most people have income between 50000 and 80000.

3. Transcript Overview

In [ ]:
```
transcript.head()
```

In [ ]:
```
transcript[transcript.event=='offer received'].head(5)
```

In [ ]:
```
transcript[transcript.event=='offer viewed'].head(5)
```

In [ ]:
```
#Those purchased and have received an offer
transcript[transcript.event=='offer completed'].head(5)
```

In [ ]:
```
transcript[transcript.event=='offer completed'].iloc[0].value
```

In [ ]:
```
#Those purchased without an offer
transcript[transcript.event=='transaction'].head(5)
```

4. Data Engineering

Tables to be Generated:

- **transcript_new**: the transcript with offer id column name consolidated ((dup)rows are persons, columns are event, time, offer id, amount, reward)
- **person_and_offer**: transcript_new joins portfolio((dup) adding offer info to transcript_new)
- **person_offer_demographic**: person_and_offer joins parts of profile((dup) adding personal info to person_and_offer)
- **offer**: (nodup)rows are persons, columns are counts of offers received, completed, viewed&completed, noviewed&completed, received bogo,received discount, received informational, v&c_bogo, v&c_discount, v&c_discount, v&c_informational, etc.
- **offer_record**: (dup)rows are persons, columns are v&c offer id and time.
- **offer_norec_comp**: (nodup)rows are persons, columns are counts of offers completed, completed bogo, completed discount, completed informational, received&completed, noreceived&completed, nr&c_bogo, nr&c_discount, nr&c_informational, etc.
- **transaction_gen**: (nodup)rows are persons, columns are total amount of transactions(including those not completed), amount of transactions related to viewed&completed offers, amount of transactions related to noviewed&completed offers, etc.

In [ ]:
```
#convert dict into dummy
transcript=pd.concat([transcript.drop(['value'], axis=1), transcript['value'].apply(p
d.Series)], axis=1)
```

In [ ]:
```
transcript.head()
```

Below are preprocessing code having run and saved as csv, **no need to run again.**

In [ ]:
```
#it appears that those offer received and viewed have offer id in "offer id", while t
he offer completedc atogory have offer id in "offer_id".
#should consolidate these two columns

transcript['consolidate_offer_id']=0
df_1=transcript[transcript.event=='offer received']
df_2=transcript[transcript.event=='offer viewed']
df_3=transcript[transcript.event=='offer completed']
df_4=transcript[transcript.event=='transaction']
df_1['consolidate_offer_id']=df_1['offer id']
df_2['consolidate_offer_id']=df_2['offer id']
df_3['consolidate_offer_id']=df_3['offer_id']
df_4['consolidate_offer_id']=df_4['offer_id']
transcript_new=pd.concat([df_1,df_2,df_3,df_4],axis=0)
```

In [ ]:
```
#Save this data to read next time
transcript_new.to_csv('data/transcript_new_2.csv')
```

In [ ]:
```
from tqdm import trange
```

Read the saved output:

In [ ]:
```
#transcript=pd.read_csv('data/transcript_new.csv')
transcript_new=pd.read_csv('data/transcript_new_2.csv')
```

In [ ]:
```
del transcript_new['Unnamed: 0']
```

In [ ]:
```
del transcript_new['offer id']
```

In [ ]:
```
del transcript_new['offer_id']
```

In [ ]:
```
transcript_new.sample(5)
```

In [ ]:
```
#Visualizing those completed the offer
amount=transcript_new[transcript_new.event=='offer completed']['amount']
```

In [ ]:
```
amount.isnull().mean()
```

Comments: It seems that "offer completed" only has the offer id & reward; to see how it relates to transaction, it should join the "transactions"; or to see how is the difficulty, it should join the Portfolio table.

In [ ]:
```
offer_completed=transcript_new[transcript_new.event=='offer completed']
```

In [ ]:
```
transaction=transcript_new[transcript_new.event=='transaction']
```

In [ ]:
```
#hour to day
transaction['time']=transaction['time']/24
```

In [ ]:
```
transaction.rename(columns={'time':'transaction_time'}, inplace=True)
```

In [ ]:
```
portfolio.rename(columns={'id':'consolidate_offer_id'}, inplace=True)
```

In [ ]:
```
#map transcript with portfolio to see details of the offer
person_and_offer=transcript_new.merge(portfolio,on='consolidate_offer_id')
```

In [ ]:
```
del person_and_offer['reward_x']
```

In [ ]:
```
person_and_offer.rename(columns={'reward_y':'reward'}, inplace=True)
```

In [ ]:
```
person_and_offer.columns
```

In [ ]:
```
#reorder the columns
person_and_offer=person_and_offer[['event', 'person', 'consolidate_offer_id','offer_t
ype','difficulty','amount','reward',\
                                   'duration', 'time','channel_email',\
                                       'channel_mobile', 'channel_social', 'channel_w
eb']]
```

In [ ]:
```
person_and_offer.rename(columns={'time':'offer_time'}, inplace=True)
```

In [ ]:
```
#turn to days
person_and_offer['offer_time']=person_and_offer['offer_time']/24
```

In [ ]:
```
person_and_offer.amount.isnull().mean()
```

In [ ]:
```
#all nan in 'amount', hence can delete
del person_and_offer['amount']
```

In [ ]:
```
transaction.reward.isnull().mean()
```

In [ ]:
```
#all nan in 'reward', hence can delete
del transaction['reward']
```

In [ ]:
```
del transaction['event']
```

In [ ]:
```
transaction.consolidate_offer_id.isnull().mean()
```

In [ ]:
```
#all nan in 'consolidate_offer_id', hence can delete
del transaction['consolidate_offer_id']
```

In [ ]:
```
transaction.groupby('person').count().head()
```

In [ ]:
```
np.unique(person_and_offer.event)
```

In [ ]:
```
person_and_offer.reward.isnull().mean()
```

In [ ]:
```
#sort by person and offer_time, which may help for the following transaction merge
person_and_offer=person_and_offer.sort_values(['person','offer_time'])
```

In [ ]:
```
person_and_offer[person_and_offer.person=='003d66b6608740288d6cc97a6903f4f0']
```

In [ ]:
```
transaction[transaction.person=='003d66b6608740288d6cc97a6903f4f0']
```

Comment: The above cell shows that **person_and_offer** dataframe contains the tracking of the same offer.

For example,
offer "5a8bc65990b245e5a138643cd4eb9837" is received and viewed;
offer "0b1e1539f2cc45b7b9fa7c272da2e1d7" is received and completed.

Also,
one may received a same offer **more than once**.

In [ ]:
```
profile.rename(columns={'id':'person'}, inplace=True)
```

In [ ]:
```
#merge person&offer with profile
person_offer_demographic=person_and_offer.merge(profile[['person','age','gender','inc
ome']],on='person')
```

In [ ]:
```
person_and_offer.columns
```

In [ ]:
```
#reorder columns
person_offer_demographic=person_offer_demographic[['event', 'person', 'age','gender',
'income','consolidate_offer_id', 'offer_type', 'difficulty',
       'reward', 'duration', 'offer_time', 'channel_email', 'channel_mobile',
       'channel_social', 'channel_web']]
```

In [ ]:
```
person_offer_demographic.head()
```

In [ ]:
```
#Save and reuse
person_offer_demographic.to_csv('data/person_offer_demographic.csv')
```

In [ ]:
```
person_offer_demographic=pd.read_csv('data/person_offer_demographic.csv')
del person_offer_demographic['Unnamed: 0']
```

Comments: In the above parts, we link person, offer and demographic information together.

However, it is tricky to figure out **how many completed offers were from offers that the person viewed beforehand** because some of the offers were completed first and then viewed afterwards; Hence, we have to separate 'viewed and completed' offers from 'completed and then viewed' ones.

Also, we are going to calculate the transactions associated with viewed and completed vs. completed but not viewed offers because for each transaction, we don't know if it was associated with a completed offer or not.

We first conduct data engineering on offer information.

In [ ]:
```
#Loop through the unique persons one by one in the transcript. Each time:

#1.Extract all records for this person(using transcript['person id']==person id)

#2.Extract this person's 'offer received' records

#3.Loop through the received offers: For every loop, store the received offer time as
"start", and "start" plus "duration" is the end of
#the offer valid time; A. find whether there are "offer viewed" in the time from "sta
rt" to "end", offer id equals this offer id;B. find
#whether there are "offer completed" in the time from "start" to "end", offer id equa
ls this offer id; if A & B are satisfied, mark viewed
#&completed; if only B is satisfied, mark noviewed&completed; Use a list to keep trac
k of viewed&completed offers person id, offer id, time

#4.After the loop in step 3, count the number of received, received bogo&discount&inf
ormational, completed, viewed&completed, noviewed&
#completed, viewed&completed bogo&discount&informational.
```

In [ ]:

```
def offer_analyzer(person_df,person,idx,offer,offer_record):
    '''
    This function takes in a person's events from transcript df and generate a df that includes the number and type
    of offer the person received, completed, viewed and completed as well as not viewed but completed. The function
    also generates a df of all viewed and completed offers by all users for future use.

    inputs:
        1. person_df - all events of this person
        2. person - the person's id
        3. idx - index to keep track of for generating 'viewed & completed offer data frame'
        4. offer - empty offer df
        5. offer_record - empty offer_record df(if records viewed&completed offers)


    outputs:
        1. [final] - a list of all output variables including:
           receive -- counts of offer received
           comp -- counts of offer completed
           view_comp -- counts of offer viewed and completed
           noview_comp -- counts of offer completed without viewing
           bogo -- counts of viewed & completed bogo offer
           discount -- counts of viewed & completed discount offer
           informational -- counts of viewed & completed informational offer and
           10 columns for 10 types of offers-- how many times this specific offer was viewed & completed
        2. idx - updated index to keep track of for generating 'viewed & completed offer dataframe'
    '''

    # select all offers the person received
    offers = person_df[person_df.event=='offer received']

    # start counting
    comp = 0                    # completed offers
    view_comp = 0               # completed after view
    noview_comp = 0             # completed without view
    view_comp_offer_list = []           # keep track of offer id
    view_comp_offer_type_list = []      # keep track of offer type

    # loop through received offers and check if each offer was completed and/or viewed
    # 'start->end' is the time window in which an offer can be viewed & completed
    for i in range(len(offers)):
        id = offers.iloc[i]['consolidate_offer_id'] # offer id
        start = offers.iloc[i]['offer_time'] # time when this offer was received
        end = offers.iloc[i]['duration'] + start # end-point of this offer

        # now check if this offer was viewed and/or completed
        viewed = 'offer viewed' in list(person_df[(person_df.offer_time>=start)&(person_df.offer_time<=end)&\
                                                  (person_df.consolidate_offer_id==id)]['event'])
        completed = 'offer completed' in list(person_df[(person_df.offer_time>=start)&(person_df.offer_time<=end)&\
                                                        (person_df.consolidate_offer_id==id)]['event'])

        if completed:
            comp +=1

            if viewed:
```

```
                view_comp +=1
                view_comp_offer_list.append(id)
                view_comp_offer_type_list.append(person_df[person_df.consolidate_offe
r_id==id]['offer_type'].iloc[0])
                idx_time = person_df[(person_df.offer_time>=start)&(person_df.offer_t
ime<=end)&(person_df.consolidate_offer_id==id)&\
                                     (person_df.event=='offer completed')].iloc[0]['o
ffer_time']
                offer_record.iloc[idx] = [person,id,idx_time] # keep track of viewed
 & completed offers in this df
                idx +=1
            else: noview_comp +=1

    receive = len(offers)
    receive_bogo = list(offers.offer_type).count('bogo')
    receive_discount = list(offers.offer_type).count('discount')
    receive_informational = list(offers.offer_type).count('informational')
    bogo = view_comp_offer_type_list.count('bogo')
    discount = view_comp_offer_type_list.count('discount')
    informational = view_comp_offer_type_list.count('informational')

    # count how many times each type of offer was viewed & completed
    # loop through 10 different offers
    counts = []
    for off in list(portfolio.consolidate_offer_id):
        counts.append(view_comp_offer_list.count(off))

    final = [receive,receive_bogo,receive_discount,receive_informational,comp,view_co
mp,noview_comp,bogo,discount,informational] + counts

    return final,idx,offer,offer_record
```

In [ ]:

```
def get_offer_df(transcript):
    '''
    This function generates/modify 'offer' dataframe containing all offer history of
 all users,
    and 'offer_record' dataframe containing all viewed & completed offers from all us
ers.

    inputs:
        transcript - events df

    outputs:
        1. offer - a dataframe containing all offer summaries of users; columns repre
sent:
           [offers received; completed; viewed & completed; completed without viewin
g; viewed & completed bogo offer;
           viewed & completed discount offer; counts for each offer id (viewed and co
mpleted counts)]
        2. offer_record - a dataframe containing records (offer id, time and person i
d) for all
           viewed & completed offers
    '''



    # create empty offer and offer_record dataframes
    offer_record = pd.DataFrame(columns=['person','id','time'],index=range(len(transc
ript)))
    offer = pd.DataFrame(columns = ['receive','rec_bogo','rec_discount','rec_informat
ional','comp','view_comp',
                                    'noview_comp','bogo','discount','informational',
                                'ae264e3637204a6fb9bb56bc8210ddfd','4d5c57ea9a6940dd89
1ad53e9dbe8da0',
                                 '3f207df678b143eea3cee63160fa8bed','9b98b8c7a33c4b65b
9aebfe6a799e6d9',
                                '0b1e1539f2cc45b7b9fa7c272da2e1d7','2298d6c36e964ae4a3
e7e9706d1fb8c2',
                                'fafdcd668e3743c1bb461111dcafc2a4','5a8bc65990b245e5a1
38643cd4eb9837',
                                'f19421c1d4aa40978ebb69ca19b0e20d','2906b810c7d4411798
c6938adc9daaa5']
                                ,index=list(transcript.person.unique()))

    persons = list(transcript.person.unique())
    idx = 0

    # loop through all users
    for i in trange(len(persons)):
        person=persons[i]
        person_df = transcript[transcript.person==person]

        # use above function to parse offers of a user, and save the result in offer
 df
        final, idx, offer, offer_record = offer_analyzer(person_df,person,idx,offer,o
ffer_record)
        offer.loc[person] = final

    offer = offer.reset_index()

    return offer, offer_record
```

In [ ]:
```
# run function and get the dataframes modified!
offer, offer_record = get_offer_df(person_offer_demographic)

# take a look at the offer df
offer.sample(5)
```

In [ ]:
```
#Save and Reuse
#offer.to_csv('data/offer.csv')
offer_record.to_csv('data/offer_record.csv')
offer=pd.read_csv('data/offer.csv')
offer_record=pd.read_csv('data/offer_record.csv')
del offer['Unnamed: 0']
del offer_record['Unnamed: 0']
offer.rename(columns={'index':'person','bogo':'vc_bogo','discount':'vc_discount','inf
ormational':'vc_informational'}, inplace=True)
```

In [ ]:
```
offer.head(10)
```

Below, we attempt to see whether there are offers completed without being received.

Actually, we don't need to consider time comparison for this time, since if the offer is marked "completed", it **must** be completed in the valid time given!

In [ ]:

```
def offer_analyzer_norec_comp(person_df,person,idx,offer,offer_record):
    '''
    This function takes in a person's events from transcript df and generate a df tha
t includes the number and type
    of offer the person completed, received, received and completed as well as not re
ceived but completed. The function
    also generates a df of all not received but completed offers by all users for fut
ure use.

    inputs:
        1. person_df - all events of this person
        2. person - the person's id
        3. idx - index to keep track of for generating 'not received & completed offe
r dataframe'
        4. offer - empty offer df
        5. offer_record - empty offer_record df(if records not received&completed off
ers)


    outputs:
        1. [final] - a list of all output variables including:
           receive -- counts of offer received
           comp -- counts of offer completed
           view_comp -- counts of offer viewed and completed
           noview_comp -- counts of offer completed without viewing
           bogo -- counts of viewed & completed bogo offer
           discount -- counts of viewed & completed discount offer
           informational -- counts of viewed & completed informational offer and
           10 columns for 10 types of offers-- how many times this specific offer was
viewed & completed
        2. idx - updated index to keep track of for generating 'viewed & completed of
fer dataframe'
    '''

    # select all offers the person received
    comp_offers = person_df[person_df.event=='offer completed']

    # start counting
    rec_comp = 0                    # received&completed offers
    norec_comp = 0             # completed without received
    norec_comp_offer_list = []             # keep track of offer id
    norec_comp_offer_type_list = []        # keep track of offer type

    # loop through received offers and check if each offer was received/not recieved
    for i in range(len(comp_offers)):
        id = comp_offers.iloc[i]['consolidate_offer_id'] # offer id
        comp_time = comp_offers.iloc[i]['offer_time'] # time when this offer was comp
leted

        # now check if this offer was received/not recieved
        received = 'offer received' in list(person_df[(person_df.consolidate_offer_id
==id)]['event'])

        if received:
            rec_comp +=1

        else:
            norec_comp +=1
            norec_comp_offer_list.append(id)
            norec_comp_offer_list.append(person_df[person_df.consolidate_offer_id==id
]['offer_type'].iloc[0])
            idx_time = person_df[(person_df.consolidate_offer_id==id)&\
                                 (person_df.event=='offer completed')].iloc[0]['offer
_time']
            offer_record.iloc[idx] = [person,id,idx_time] # keep track of noreceived
```

```
 & completed offers in this df
            idx +=1

    complete = len(comp_offers)
    complete_bogo = list(comp_offers.offer_type).count('bogo')
    complete_discount = list(comp_offers.offer_type).count('discount')
    complete_informational = list(comp_offers.offer_type).count('informational')
    norec_comp_bogo = norec_comp_offer_type_list.count('bogo')
    norec_comp_discount = norec_comp_offer_type_list.count('discount')
    norec_comp_informational = norec_comp_offer_type_list.count('informational')

    # count how many times each type of offer was viewed & completed
    # loop through 10 different offers
    counts = []
    for off in list(portfolio.consolidate_offer_id):
        counts.append(norec_comp_offer_list.count(off))

    final = [complete,complete_bogo,complete_discount,complete_informational,rec_comp
,norec_comp,norec_comp_bogo,norec_comp_discount,norec_comp_informational] + counts

    return final,idx,offer,offer_record
```

In [ ]:

```
def get_offer_df_norec_comp(transcript):
    '''
    This function generates/modify 'offer' dataframe containing all offer history of
 all users,
    and 'offer_record' dataframe containing all noreceived & completed offers from al
l users.

    inputs:
        transcript - events df

    outputs:
        1. offer - a dataframe containing all offer summaries of users; columns repre
sent:
           [offers received; completed; viewed & completed; completed without viewin
g; viewed & completed bogo offer;
           viewed & completed discount offer; counts for each offer id (viewed and co
mpleted counts)]
        2. offer_record - a dataframe containing records (offer id, time and person i
d) for all
           viewed & completed offers
    '''



    # create empty offer and offer_record dataframes
    offer_record = pd.DataFrame(columns=['person','id','time'],index=range(len(transc
ript)))
    offer = pd.DataFrame(columns = ['complete','complete_bogo','complete_discount','c
omplete_informational','rec_comp','norec_comp',
                                    'norec_comp_bogo','norec_comp_discount','norec_co
mp_informational',
                                '4d5c57ea9a6940dd89
1ad53e9dbe8da0',
                                '9b98b8c7a33c4b65b
9aebfe6a799e6d9',
                                '2298d6c36e964ae4a3
e7e9706d1fb8c2',
                                '5a8bc65990b245e5a1
38643cd4eb9837',
                                '2906b810c7d4411798
c6938adc9daaa5']
                                ,index=list(transcript.person.unique()))

    persons = list(transcript.person.unique())
    idx = 0

    # loop through all users
    for i in trange(len(persons)):
        person=persons[i]
        person_df = transcript[transcript.person==person]

        # use above function to parse offers of a user, and save the result in offer
 df
        final, idx, offer, offer_record = offer_analyzer_norec_comp(person_df,person,
idx,offer,offer_record)
        offer.loc[person] = final

    offer = offer.reset_index()

    return offer, offer_record
```

In [ ]:
```
from tqdm import trange
# run function and get the dataframes modified!
offer_norec_comp, offer_record_norec_comp = get_offer_df_norec_comp(person_offer_demo
graphic)

# take a look at the offer df
offer_norec_comp.sample(5)
```

In [ ]:
```
offer_norec_comp.to_csv('data/offer_norec_comp.csv')
offer_record_norec_comp.to_csv('data/offer_record_norec_comp.csv')
```

In [ ]:
```
offer_norec_comp=pd.read_csv('data/offer_norec_comp.csv')
offer_record_norec_comp=pd.read_csv('data/offer_record_norec_comp.csv')
del offer_norec_comp['Unnamed: 0']
```

In [ ]:
```
offer_record_norec_comp.isnull().mean()
```

In [ ]:
```
offer_norec_comp['norec_comp'].mean()
```

Comments: It appears that no such offers(completed&noreceived) are in our records.

In [ ]:
```
[1,4,5]+[1,2,3]
```

In [ ]:
```
print(offer_record.shape)
offer_record.isnull().mean()#less than 0.17 in the offer received, offer viewed and o
ffer completed records are viewed&completed
```

In [ ]:
```
offer[offer['person']=='e1e614f30e9c45478d1c5aa8fe3c6dbb']
```

In [ ]:
```
offer.shape
```

In [ ]:
```
offer.head()
```

In [ ]:
```
#viewed&completed offer record
offer_record.head()
```

Comments: now the df "offer" can show clearly for each person,
1.how many offers received, within which how many received bogo&discount&informational;
2.how many offers completed, within which how many are viewed before completed and how many are completed before viewed;
3.within view&completed, how many are bogo&discount&informational;
4.within view&completed, the counts of each offer in the 10 choices.

We then conduct data engineering on transaction information.

In [ ]:
```
person_offer_demographic['offer_time'].sample()
```

In [ ]:
```
transaction['transaction_time'].sample()
```

In [ ]:
```
offer_record['time'][:100].sample()
```

Core logic of codes below: If an offer is completed, then the transaction time should be the same as that of completed offer.
Hence we first extract the transaction time, then see if any completed offer time of this person matches ->transaction associated with viewed and completed offers;
Else->transaction associated with completed but not viewed offers

In [ ]:
```
from tqdm import trange
```

In [ ]:
```
#Loop through the unique persons one by one in the transaction data. Each time:

#1.Extract all transaction records for this person(using transaction['person id']==pe
rson id)

#2.Sum the transaction amount of this person and mark as "total"

#3.Loop through the transaction records: For every loop, store the transaction time a
nd amount; find in the transcript(non-transaction
#records) satisfying "offer completed" and the same time as transaction time; Judge w
hether the offer(s) falls in the viewed&completed list;
#if YES, add the transaction amount to viewed&completed transaction amount; if NO, ad
d it to noviewed&completed transaction amount;
```

In [ ]:

```
def transaction_calculator(trans_ori,person,person_df,offer_record,transaction):
    '''
    This function takes in a person's events and calculate the total transaction, tra
nsaction associated
    with viewed & completed offers and transaction associated with not viewed but com
pleted offers.

    inputs:
        1. trans_ori - transaction dataframe
        2. person - person id
        3. person_df - all events of the person
        4. offer_record - record of all viewed & completed offers, will be used to as
sess if a certain amount of
           transaction is associated with viewed & completed offer

    outputs:
        1. [final] - a list including all output variables:
           total - total transaction made by this person
           view_comp - transaction associated with viewed and completed offers
           noview_comp - transaction associated with completed but not viewed offers
    '''

    # calculate total transaction made by this person
    trans = trans_ori[trans_ori['person']==person][['person','transaction_time','amou
nt']] # all transactions
    total = trans['amount'].sum()

    # start calculating
    view_comp = 0
    noview_comp = 0
    view = False
    comp = False

    # loop through transactions to see if they are associated with viewed & completed
offer
    for i in range(len(trans)):
        time = trans.iloc[i]['transaction_time'] # time of this transaction
        amount = trans.iloc[i]['amount'] # amount of this transaction

        # check if there's any completed offer(s) at this transaction time
        comp_off = person_df[(person_df.offer_time==time)&(person_df.event=='offer co
mpleted')] # completed offers df

        if len(comp_off) > 0:
            comp = True

            # check if the completed offer was viewed
            # if more than one offers were completed simultaneously, check if ANY of
 them were viewed
            for j in range(len(comp_off)):
                if ((offer_record.person==comp_off.iloc[j]['person'])&(offer_record.i
d==comp_off.iloc[j]['consolidate_offer_id'])&
                (offer_record.time==comp_off.iloc[j]['offer_time'])).any(): #offer re
cord is a df of viewed&completed offers
                    view = True

        # update transactions for viewed & completed offers as well as not viewed but
completed offers
        if comp and view:
            view_comp += amount
        else:
            noview_comp += amount

        # reset the value for next transaction
        view = False
```

```
        comp = False

    final = [total,view_comp,noview_comp]

    return final,transaction
```

In [ ]:
```
def get_transaction_df(trans_ori,transcript):
    '''
    This function generates a transaction dataframe containing all purchase behavior
 of all users.

    inputs:
        1. trans_ori - transaction dataframe
        2. transcript - events df
        3. transaction - empty transaction df

    outputs:
        1. transaction_df - a dataframe contains all purchase behavior of all users i
ncluding total transaction
           amount, transaction associated with viewed & compelted offers as well as t
ransaction associated with
           not viewed but completed offers

    '''

    # create an empty transaction dataframe
    transaction = pd.DataFrame(columns = ['total','view_complete_tran','noview_comple
te_tran'], index=list(transcript.person.unique()))

    # loop through all users in transcript (which is also the overall users in profil
e)
    persons = list(transcript.person.unique())

    for i in trange(len(persons)):
        person=persons[i]
        person_df = transcript[transcript.person==person]

        # use above function to parse transaction and save the result in transaction
 df
        final,transaction = transaction_calculator(trans_ori,person,person_df,offer_r
ecord,transaction)
        transaction.loc[person] = final

    transaction = transaction.reset_index()

    return transaction
```

In [ ]:
```
# run the function and get the transaction df modified!
transaction_gen = get_transaction_df(transaction,person_offer_demographic)

# take a look at the transaction df
transaction_gen.sample(5)
```

In [ ]:
```
#transaction_gen.to_csv('data/transaction_gen.csv')
```

In [ ]:
```
transaction_gen=pd.read_csv('data/transaction_gen.csv')
del transaction_gen['Unnamed: 0']
```

In [ ]:
```
transaction_gen.head()
```

5. Customer Behavior Analysis

5.1 Customer Response Demographic Analysis

Combine customer information&offer&transaction altogether:

In [ ]:
```
import datetime
```

In [ ]:
```
person_all_information=transaction_gen.merge(profile,on='person')
person_all_information=person_all_information.merge(offer,on='person')
person_all_information['became_member_on'] = person_all_information.became_member_on.
apply(lambda x: datetime.datetime.strptime(str(x),'%Y%m%d').date())
person_all_information.became_member_on=person_all_information.became_member_on.apply
(lambda x: x.toordinal())
```

In [ ]:
```
person_all_information.to_csv('data/person_all_information.csv')
```

In [ ]:
```
person_all_information=pd.read_csv('data/person_all_information.csv')
del person_all_information['Unnamed: 0']
```

In [ ]:
```
person_all_information.columns
```

In [ ]:
```
person_all_information=person_all_information[['person','age','gender','income','beca
me_member_on','total','view_complete_tran','noview_complete_tran','receive','rec_bog
o','rec_discount','rec_informational','comp','view_comp',
                                               'noview_comp','vc_bogo','vc_discount','vc_informa
tional',
                                    'ae264e3637204a6fb9bb56bc8210ddfd','4d5c57ea9a6940dd89
1ad53e9dbe8da0',
                                     '3f207df678b143eea3cee63160fa8bed','9b98b8c7a33c4b65b
9aebfe6a799e6d9',
                                    '0b1e1539f2cc45b7b9fa7c272da2e1d7','2298d6c36e964ae4a3
e7e9706d1fb8c2',
                                    'fafdcd668e3743c1bb461111dcafc2a4','5a8bc65990b245e5a1
38643cd4eb9837',
                                    'f19421c1d4aa40978ebb69ca19b0e20d','2906b810c7d4411798
c6938adc9daaa5']]
```

In [ ]:
```
person_all_information.head()
```

In [ ]:
```
person_all_information['responded'] = person_all_information.view_comp.apply(lambda x
: 'T' if x!=0 else 'F')
```

In [ ]:
```
person_all_information['gender']=person_all_information.gender.apply(lambda x: 1 if x
=='M' else (0 if x=='F' else 2))
```

In [ ]:
```
import warnings
warnings.filterwarnings('ignore') # turn off warning on missing values
sns.pairplot(person_all_information[['age','gender','income','became_member_on','view
_complete_tran','responded']].fillna(0),hue='responded',hue_order=['T','F'],plot_kws=
dict(alpha=0.2),dropna=True);
```

Comments: From the above graph, we can clearly see that the response has positive relations with age, income and became_member_on(concluded from the last row of the graph).

Also, we can see there are orange lines in the 3 plots(age, income, became_member_on) of the last row. Those customers might be those who newly registered the mobile app and have low income.

From the gender-gender plot, we can see that female tend to respond to offer compared with male and those input "Other" or did not input.

Furthermore, those who didn't fully provide their personal information tend not to respond to the offer.

In [ ]:
```
portfolio
```

In [ ]:
```
person_and_offer.head()
```

In [ ]:
```
person_all_information.head()
```

5.2 Analysis Based on Offer Information

Next, we make analysis based on the offer information, e.g. offer type, channel, etc.

Offer Type

In [ ]:
```
complete_rate=person_all_information.comp.sum()/person_all_information.receive.sum()
complete_rate
```

In [ ]:
```
bogo_complete_rate=offer_norec_comp.complete_bogo.sum()/person_all_information.rec_bo
go.sum()
bogo_complete_rate
```

In [ ]:
```
discount_complete_rate=offer_norec_comp.complete_discount.sum()/person_all_informatio
n.rec_discount.sum()
discount_complete_rate
```

In [ ]:
```
informational_complete_rate=offer_norec_comp.complete_informational.sum()/person_all_
information.rec_informational.sum()
informational_complete_rate
```

In [ ]:
```
vc_rate=person_all_information.view_comp.sum()/person_all_information.receive.sum()
vc_rate
```

In [ ]:
```
bogo_vc_rate=person_all_information.vc_bogo.sum()/person_all_information.rec_bogo.sum
()
bogo_vc_rate
```

In [ ]:
```
discount_vc_rate=person_all_information.vc_discount.sum()/person_all_information.rec_
bogo.sum()
discount_vc_rate
```

In [ ]:
```
informational_vc_rate=person_all_information.vc_informational.sum()/person_all_inform
ation.rec_bogo.sum()
informational_vc_rate
```

From above we can see that the complete rate is about 44%, and it appears that discount offer is more preferrable than bogo and informational. Especially, for informational offer, the complete rate is 0, which shows that this type of offer might be simply providing information and hardlly inspires people to buy the products.

Channel

In [ ]:
```
email_rec=person_and_offer[person_and_offer.event=='offer received'].channel_email.sum()
email_comp=person_and_offer[person_and_offer.event=='offer completed'].channel_email.sum()
email_comp/email_rec
```

In [ ]:
```
mobile_rec=person_and_offer[person_and_offer.event=='offer received'].channel_mobile.sum()
mobile_comp=person_and_offer[person_and_offer.event=='offer completed'].channel_mobile.sum()
mobile_comp/mobile_rec
```

In [ ]:
```
social_rec=person_and_offer[person_and_offer.event=='offer received'].channel_social.sum()
social_comp=person_and_offer[person_and_offer.event=='offer completed'].channel_social.sum()
social_comp/social_rec
```

In [ ]:
```
web_rec=person_and_offer[person_and_offer.event=='offer received'].channel_web.sum()
web_comp=person_and_offer[person_and_offer.event=='offer completed'].channel_web.sum()
web_comp/web_rec
```

From above, it appears that web might be the most efficient channel, which contributing the highest complete rate.

III. Data Modeling

In this part, we are going to build a machine learning model that predicts whether or not someone will respond to an offer.

From above analysis, we know that age, gender, income, membership date, offer type can affect whether a customer respond to an offer. Hence, we make them as explanatory variable and make 'responded' as binary response variable.

In [ ]:
```
#Import all useful packages
import datetime
#from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score,f1_score
```

In [ ]:
```
#make the time data available to extract year&month
profile['became_member_on'] = profile.became_member_on.apply(lambda x: datetime.datet
ime.strptime(str(x),'%Y%m%d').date())
```

Note that, below we are going to mark viewed&completed as responded instead of mark "completed" as responded, because the latter is actually a random behavior without **knowing** he/she has such an offer.

In [ ]:
```
def transform_and_tts(profile,offer):

    '''
    This function takes in profile and offer dataframes and returns training and test datasets for ML.

    inputs:
        1. profile - profile dataframe
        2. offer - offer dataframe

    outputs:
        1. X_train, X_test, y_train, y_test - input data for training and test, target label for training and test
        2. age_interval, income_interval - interval index for age and income variables
    '''


    # transform features and label
    prof = profile.copy()
    prof = prof[prof.person.isin(list(offer[offer.receive==0]['person']))==False] # exclude people never received an offer
    prof['member_year'] = prof.became_member_on.apply(lambda x: x.year)
    prof['member_month'] = prof.became_member_on.apply(lambda x: x.month)
    prof.drop('became_member_on',axis=1,inplace=True)

    # create 'offer' and 'label' columns: offer col has two values (bogo or discount) and label col shows whether
    # the user responded to the offer or not
    bogo = offer[offer.rec_bogo!=0][['person','vc_bogo']]
    bogo['label'] = bogo.vc_bogo.apply(lambda x: 0 if x==0 else 1)
    bogo.drop('vc_bogo',axis=1,inplace=True)
    bogo = prof.merge(bogo,on='person').drop('person',axis=1)
    bogo['offer'] = 'bogo'

    discount = offer[offer.rec_discount!=0][['person','vc_discount']]
    discount['label'] = discount.vc_discount.apply(lambda x: 0 if x==0 else 1)
    discount.drop('vc_discount',axis=1,inplace=True)
    discount = prof.merge(discount,on='person').drop('person',axis=1)
    discount['offer'] = 'discount'

    # concat bogo and discount df
    df = pd.concat([bogo,discount])
    df.age.replace(118,np.NaN,inplace=True)
    df=df.dropna()#drop nan for convenience

    # creat dummy variables
    df = pd.get_dummies(df,columns=['gender','member_year','member_month','offer'],dummy_na=True)

    # assign X and y
    X = df.drop('label',axis=1)
    y = df['label']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    return X_train, X_test, y_train, y_test
```

In [ ]:
```
X_train, X_test, y_train, y_test = transform_and_tts(profile,offer)
```

In [ ]:
```
X_train.head()
```

In [ ]:
```
y_train.value_counts()
```

In [ ]:
```
y_test.value_counts()
```

From above, we can see that the labels are **imbalanced**, approximately 2:1 for Positive vs. Negative labels. Hence, we should use F1_score instead of accuracy, since F1_score is a metric to balance recall&precision and to deal with imbalanced labels.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

In [ ]:
```
# Let's check out which classifier will work best in our case

classifiers = [SVC(),
               DecisionTreeClassifier(),
               RandomForestClassifier(),
               GaussianNB(),
               AdaBoostClassifier()]

X_train, X_test, y_train, y_test = transform_and_tts(profile,offer)

#Scale the X_train & X_test to [0,1]
#scaler=MinMaxScaler()

#X_train_scaled = scaler.fit_transform(X_train)
#X_test_scaled = scaler.fit_transform(X_test)

performance = pd.DataFrame(columns=["Classifier", "F1_Score"])

acc_dict = {}

for clf in classifiers:
    name = clf.__class__.__name__
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = f1_score(y_test, y_pred)
    if name not in acc_dict:
        acc_dict[name] = acc
    else:
        acc_dict[name] += acc

for clf in acc_dict:
    performance_record = pd.DataFrame([[clf, acc_dict[clf]]], columns=["Classifier",
"F1_Score"])
    performance = performance.append(performance_record)

plt.xlabel('F1_Score')
plt.title('Classifier F1_Score')

sns.barplot(x='F1_Score', y='Classifier', data=performance);
```

It appears that AdaBoost is the best classifier. Therefore, we conduct GridSearch on it to find the best parameters:

In [ ]:
```
param_grid = {"base_estimator__criterion" : ["gini", "entropy"],
              "base_estimator__splitter" :   ["best", "random"],
              "n_estimators": [5, 10, 20,50],
              "learning_rate":[0.001, 0.01, 0.1, 1],
              'base_estimator__max_depth':[1,2,3,4]
             }


DTC = DecisionTreeClassifier(random_state=42)

ADA = AdaBoostClassifier(base_estimator = DTC)

grid_ada = GridSearchCV(estimator=ADA,param_grid=param_grid,scoring='f1',cv=5)
grid_ada.fit(X_train,y_train)

print('Training F1_score is:', grid_ada.score(X_train,y_train))
print('Test F1_score is:', grid_ada.score(X_test,y_test))
```

In [ ]:
```
from sklearn.externals import joblib
joblib.dump(grid_ada.best_estimator_, 'filename4.pkl')
```

In [ ]:
```
from sklearn.externals import joblib
model=joblib.load('filename4.pkl')
model
```

Next, we want to make a function such that we take in customer info, offer type, etc, then we can transform all these information into format which can be taken by the classifier like the test data, then predict whether the customer respond or not based on the raw profile&offer information:

In [ ]:
```
X_test.columns
```

In [ ]:
```
def predict_engine(model,customer,offer_type):
    '''
    This function takes in a customers info and offer type and transform it to the same format as test data, then returns a
    predicted result(respond or not).

    inputs:
    1. model - the best classifier
    2. customer - customer's info, the same format as original profile df
    3. offer_type - 'bogo' or 'discount'

    outputs:
    prediction - whether the customer would respond to given offer or not

    '''


    # First, let's check whether the customer provided demographic info or not
    if customer['age']==118 or customer.isnull().any():
        flag = False
    else:
        flag = True


    # Creat new customer df and tranform datetime col
    cols = transform_and_tts(profile,offer)[1].columns#takes the X_test
    customer_df = pd.DataFrame(columns=cols,index=[0])
    year = customer['became_member_on'].year
    month = customer['became_member_on'].month

    customer_df['member_year_' + str(str(float(year)))] = 1
    customer_df['member_month_' + str(str(float(month)))] = 1

    if flag:
        # transform profile info if provided

        gender = customer['gender']
        customer_df['gender_' + str(gender)] = 1
        customer_df['age']=customer['age']
        customer_df['income']=customer['income']
        customer_df['offer_' + offer_type] = 1

    else:
        customer_df['gender_nan'] = 1
        customer_df['offer_' + offer_type] = 1


    customer_df.fillna(0,inplace=True)

    pred = model.predict(customer_df)[0]

    if pred==0:
        print('Not respond!')
    else:
        print('Respond!')

    return customer_df
```

In [ ]:
```
predict_engine(model, profile.sample(1).iloc[0], 'bogo')
```

In [ ]:
```
predict_engine(model, profile.sample(1).iloc[0], 'discount')
```

IV. Evaluation of Results

To conclude, in this notebook, we:

- Cleanse the offer data such that we can separate the completed offer data into A.viewed&completed offer and B.noviewed&completed offer. The reason to do this is that even though the customers made the transaction and completed the offer, he/she may not be aware of the offer. In other words, his/her action may not be offer-oriented, hence should not be counted as **responded** to the offer;
- Cleanse the offer data such that we can separate the completed offer data into A.received&completed offer and B.noreceived&completed offer to see whether there is anyone who completed the offer without actually receiving the offer. However, it appears that there is no record for **noreceived&completed** offer in our case/data;
- Compare the transaction time and the offer completed time in order to determine whether the transaction is associated with a completed offer(more specifically, viewed&completed offer or noviewed&completed offer or other offer), because in the raw data, there are only person, amounts and time given, without telling us whether it is related to any offer;
- Next, we have demographic, offer type and channel analysis on the complete rate;
- Finally, we build a machine learning model to enable the prediction of the customer's response given a customer with age, gender, income, offer type and other information.

Possible Enhancement in the Future:

- For convenience, I drop all the NaNs; in the future, other skills can be used, such as using mean, median, etc; or just simply keep it as a value, since in the future, there will still be part of customers who won't fill in their information;
- This is a classification model; Alternatively, regression model can be built to predict how much someone will spend(i.e. transaction) based on demographics and offer type;
- A web app can be built such that when inputting the customer information, the prediction of response/transaction amount can be output.

In [ ]: