Profitable Promotional Strategy for Starbucks

kartikey shaurya Follow
May 24 · 10 min read

Introduction

The case we discuss here is a real-life marketing strategy study based on a simulated data set that mimics customer behavior on the Starbucks rewards mobile app.

The goal is to combine transaction, demographic and offer data to analyze which demographic groups respond best to which offer type; Also, we will build a supervised learning model(specifically, a classification problem) that predicts whether or not someone will respond to an offer.

The article contains mainly four parts that are respectively :-

1. Data Exploration and visualization
2. Data Preprocessing
3. Customer Behavior Analysis
4. Data Modeling

Part 1. Data Exploration and Visualization

We will first have a glimpse at the 3 json data sets given:

1. For **portfolio** data, since the values in column “channels” is in a list, and in the list are one or more values in “email”, “mobile”, “social” and “web”, hence we made it as dummy variables using the code below:

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

a dummy variable

and we will get this :

	reward	difficulty	duration	offer_type	id	channel_email	channel_mobile	channel_social	channel_web
0	10	10	7	bogo	ae264e3637204a6fb9bb56bc8210ddfd	1	1	1	0

1	10	10	5	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0	1	1	1	1
2	0	0	4	informational	3f207df678b143eea3cee63160fa8bed	1	1	0	1
3	5	5	7	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	1	0	1
4	5	20	10	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7	1	0	0	1
5	3	7	7	discount	2298d6c36e964ae4a3e7e9706d1fb8c2	1	1	1	1
6	2	10	10	discount	fafdcd668e3743c1bb461111dcafc2a4	1	1	1	1
7	0	0	3	informational	5a8bc65990b245e5a138643cd4eb9837	1	1	1	0
8	5	5	5	bogo	f19421c1d4aa40978ebb69ca19b0e20d	1	1	1	1
9	2	10	7	discount	2906b810c7d4411798c6938adc9daaa5	1	1	0	1

Raw data of the portfolio

it clearly shows there are 10 choices of offers, and their difficulty(i.e. the threshold to get the reward), duration, offer id, offer type, reward, channel.

2. Next, we take a look at **profile** data:

```
profile.sample(5)
```

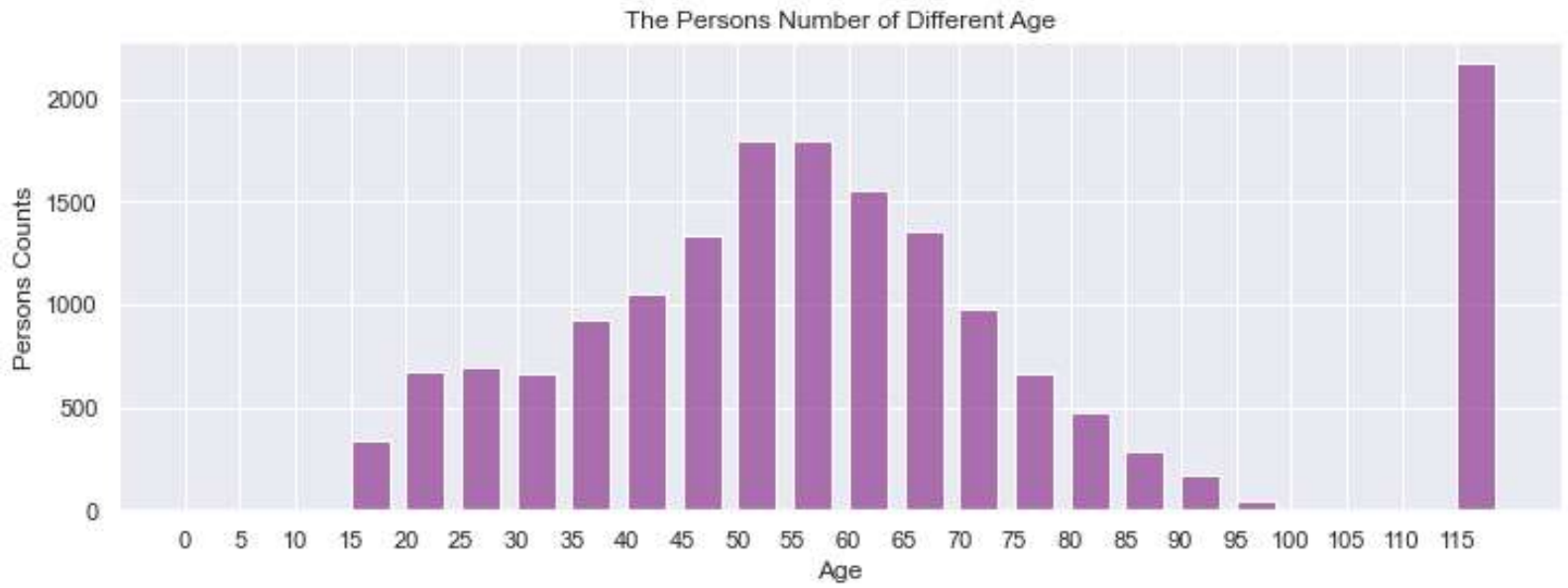
	gender	age	id	became_member_on	income
542	F	62	a453d70e85fd493f9584570a3dae58a8	20180707	117000.0
3455	M	55	6a79b214080d4afbaa10e9d0fc5a5e90	20150820	87000.0
16453	None	118	484cd5288cd2460aab5d3d3ddf8646ad	20171203	NaN
14712	M	52	74d216ad116c4ea198245bf22ff0d26b	20180428	62000.0
4849	F	42	ade281e47ea7413da6c43d28a74bd871	20170108	33000.0

we can see from the data set that the age, the date when the customer became a member(in yy/mm/dd format), gender(M, F, O or NaN), person id, and income are listed.

It is a good way to visualize the age & income distribution of our customer base.

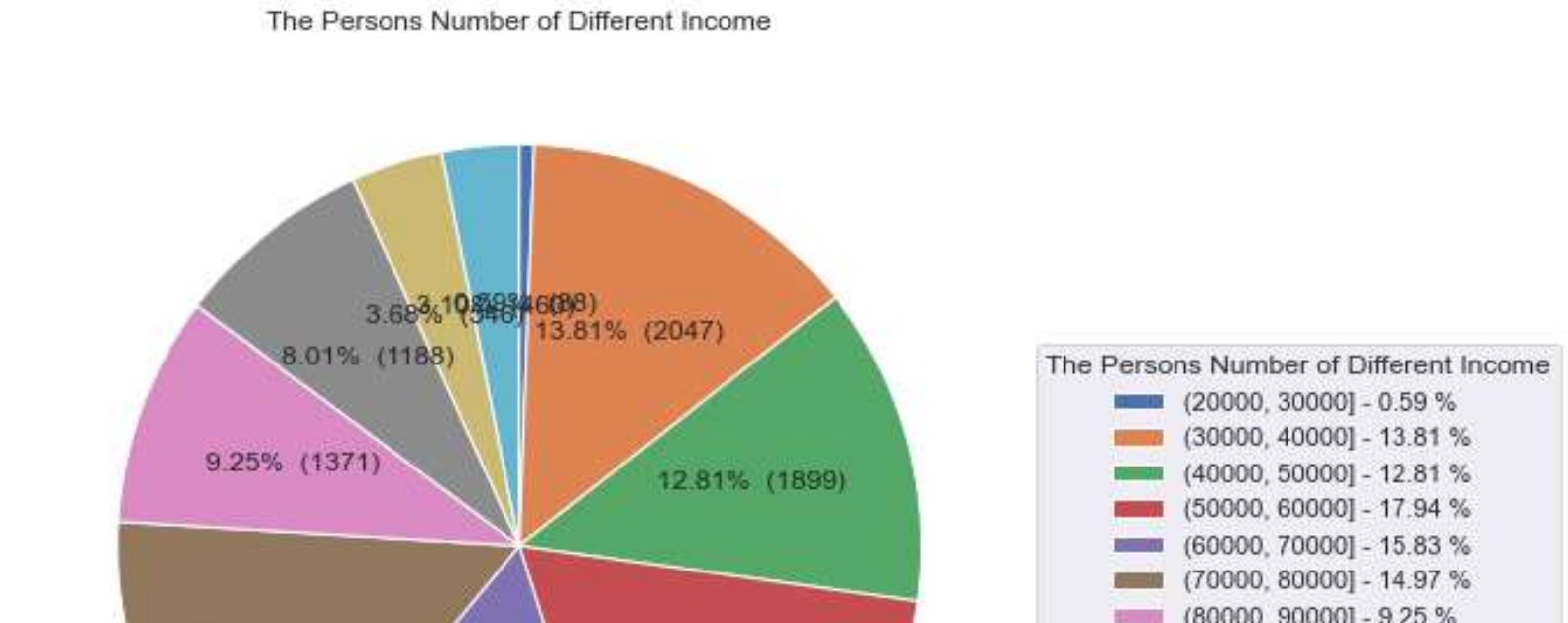
For Age, it is better to group the customers into a group for every 5 years interval using the code below:

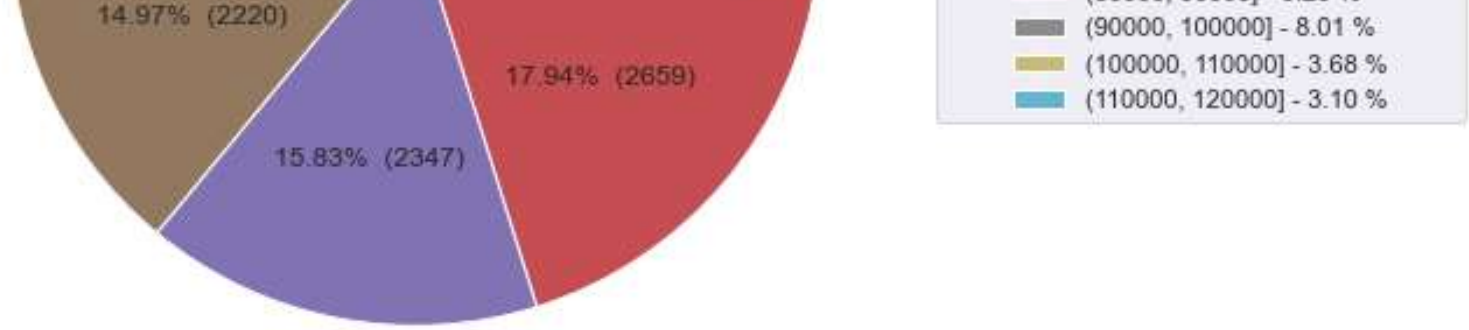
```
count_by_agegroup=profile.groupby(pd.cut(profile['age'], np.arange(0,
118+5, 5)))['id'].count()

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()
```



since those who did not enter their age are marked as 118 years old, we can see from the chart that the amount is over 2000 for this group. And apart from this, most of the customers fall in the age between 50 and 60 years old.

For Income, we drew a pie chart below:





person Number by income group

it appears that most customers have an income between 50000 and 80000.

3. We then explore the **transcript** data set:

For “offer received” and “offer viewed”, the value only contains its offer id:

```
transcript[transcript.event=='offer received'].head(5)
transcript[transcript.event=='offer viewed'].head(5)
transcript[transcript.event=='offer completed'].head(5)
transcript[transcript.event=='offer completed'].iloc[0].value
transcript[transcript.event=='transaction'].head(5)
```

	person	event	value	time
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
1	a03223e636434f42ac4c3df47e8bac43	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	0
3	8ec6ce2a7e7949b1bf142def7d0e0586	offer received	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	0
4	68617ca6246f4fbc85e91a2a49552598	offer received	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	0

	person	event	value	time
12650	389bc3fa690240e798340f5a15918d5c	offer viewed	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}	0
12651	d1ede868e29245ea91818a903fec04c6	offer viewed	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	0
12652	102e9454054946fda62242d2e176fdce	offer viewed	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	0
12653	02c083884c7d45b39cc68e1314fec56c	offer viewed	{'offer id': 'ae264e3637204a6fb9bb56bc8210ddfd'}	0
12655	be8a5d1981a2458d90b255ddc7e0d174	offer viewed	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	0

offer viewed

In the transcript, transaction data is also included, with the amount attached:

	person	event	value	time
12654	02c083884c7d45b39cc68e1314fec56c	transaction	{'amount': 0.8300000000000001}	0
12657	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	transaction	{'amount': 34.56}	0
12659	54890f68699049c2a04d415abc25e717	transaction	{'amount': 13.23}	0
12670	b2f1cd155b864803ad8334cdf13c4bd2	transaction	{'amount': 19.51}	0
12671	fe97aa22dd3e48c8b143116a8403dd52	transaction	{'amount': 18.97}	0

however, it *does not show any offers associated with the transaction*, therefore, we need to have data preprocessing procedure to solve this problem and facilitate our analysis in the following sections.

Part II. Data Preprocessing

1. We used the code below to make it clear of the time, reward, and transaction amount(can be NaN) for each record in the transcript data set:

```
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)
```

code to preprocess the Transcript

the output will be

```
transcript_new.sample(5)
```

	person	event	time	consolidate_offer_id	amount	reward
200672	702b17d2b3754788b4008985343633d8	transaction	204	NaN	20.47	NaN
194206	79b174cbc5004c84a5ade3eab1aa409a	transaction	168	NaN	1.71	NaN
227766	7eac0ad97cc04bb2b23dc6139fe12821	transaction	360	NaN	2.58	NaN
106234	0d15aeb66e044f26b1a50c678cef7fe4	offer viewed	408	2906b810c7d4411798c6938adc9daaa5	NaN	NaN

304588	201e187acb924b2c8c73f28b18d03892	transaction	702	NaN	6.48	NaN

2. Based on the above data frame, we want to see more clearly on the offer information as well as the related customer's information, hence we combined it with the portfolio and profile:

```
person_offer_demographic.head()
```

	event	person	consolidate_offer_id	offer_type	difficulty	reward	duration	offer_time	channel_email	channel_mobile	channel_social	channel_web
153521	offer received	003d66b6608740288d6cc97a6903f4f0	5a8bc65990b245e5a138643cd4eb9837	informational	0	0	3	0.0	1	1	1	0
161698	offer viewed	003d66b6608740288d6cc97a6903f4f0	5a8bc65990b245e5a138643cd4eb9837	informational	0	0	3	1.5	1	1	1	0
47308	offer received	003d66b6608740288d6cc97a6903f4f0	fafdcd668e3743c1bb461111dcafc2a4	discount	10	2	10	7.0	1	1	1	1
55848	offer viewed	003d66b6608740288d6cc97a6903f4f0	fafdcd668e3743c1bb461111dcafc2a4	discount	10	2	10	12.5	1	1	1	1
126280	offer received	003d66b6608740288d6cc97a6903f4f0	3f207df678b143eea3cee63160fa8bed	informational	0	0	4	14.0	1	1	0	1
133015	offer viewed	003d66b6608740288d6cc97a6903f4f0	3f207df678b143eea3cee63160fa8bed	informational	0	0	4	15.5	1	1	0	1
62834	offer completed	003d66b6608740288d6cc97a6903f4f0	fafdcd668e3743c1bb461111dcafc2a4	discount	10	2	10	16.0	1	1	1	1
50246	offer received	003d66b6608740288d6cc97a6903f4f0	fafdcd668e3743c1bb461111dcafc2a4	discount	10	2	10	17.0	1	1	1	1
57617	offer viewed	003d66b6608740288d6cc97a6903f4f0	fafdcd668e3743c1bb461111dcafc2a4	discount	10	2	10	17.5	1	1	1	1
22514	offer received	003d66b6608740288d6cc97a6903f4f0	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	20	5	10	21.0	1	0	0	1
63930	offer completed	003d66b6608740288d6cc97a6903f4f0	fafdcd668e3743c1bb461111dcafc2a4	discount	10	2	10	21.0	1	1	1	1
29874	offer completed	003d66b6608740288d6cc97a6903f4f0	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	20	5	10	29.0	1	0	0	1

Preprocessed Transcript with the person and other information.

3. Note that 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 afterward. 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 it should not be counted as **responded** to the offer.

Here is the pseudo-code for separating viewed&completed vs. noviewed&completed offers:

```
#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 "start" to "end", offer id equals this offer id;B. find
#whether there are "offer completed" in the time from "start" to "end", offer id equals this offer id; if A & B are satisfied, mark viewed
#&completed; if only B is satisfied, mark noviewed&completed; Use a list to keep track of viewed&completed offers person id, offer id, time

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

the pseudo code

then we got the data set below showing the number of received offer as well as number

of viewed&completed offer totally and for each offer type

4. Similar to step 3, I attempted to separate "received and completed" vs. "completed before received" offers to see whether there are offers completed without being received. However, the result shows no such records, so I don't put the process in this article.

5. Next, as discussed above, in the transcript transaction data, there are the only person, amounts, and time given, without telling us whether it is related to any offer. Hence, we need to differentiate total transaction, transaction-related to viewed&completed offers and transaction-related to noviewed&completed offers.

and the result is as follows:

	index	total	view_complete_tran	noview_complete_tran
1621	1922e62c02ec4664979eaeb8653d6fc5	61.62	34.63	26.99
1409	15953663abf54181adf1fd4c39442b85	112.04	39.99	72.05
10894	a27a36c99868424aad34d110ee7aafe7	22.61	6.62	15.99
8818	84be43a1f66b4abfa2d57321036e2559	275.44	67.47	207.97
2210	22afc7c1694847c6aedc0d4d55dce9a7	13.61	0	13.61

Part III. Customer Behavior Analysis

We first combined profile with data set generated in step 3&5 above, and got customer information&offer&transaction in one table:

```
person_all_information.head()
```

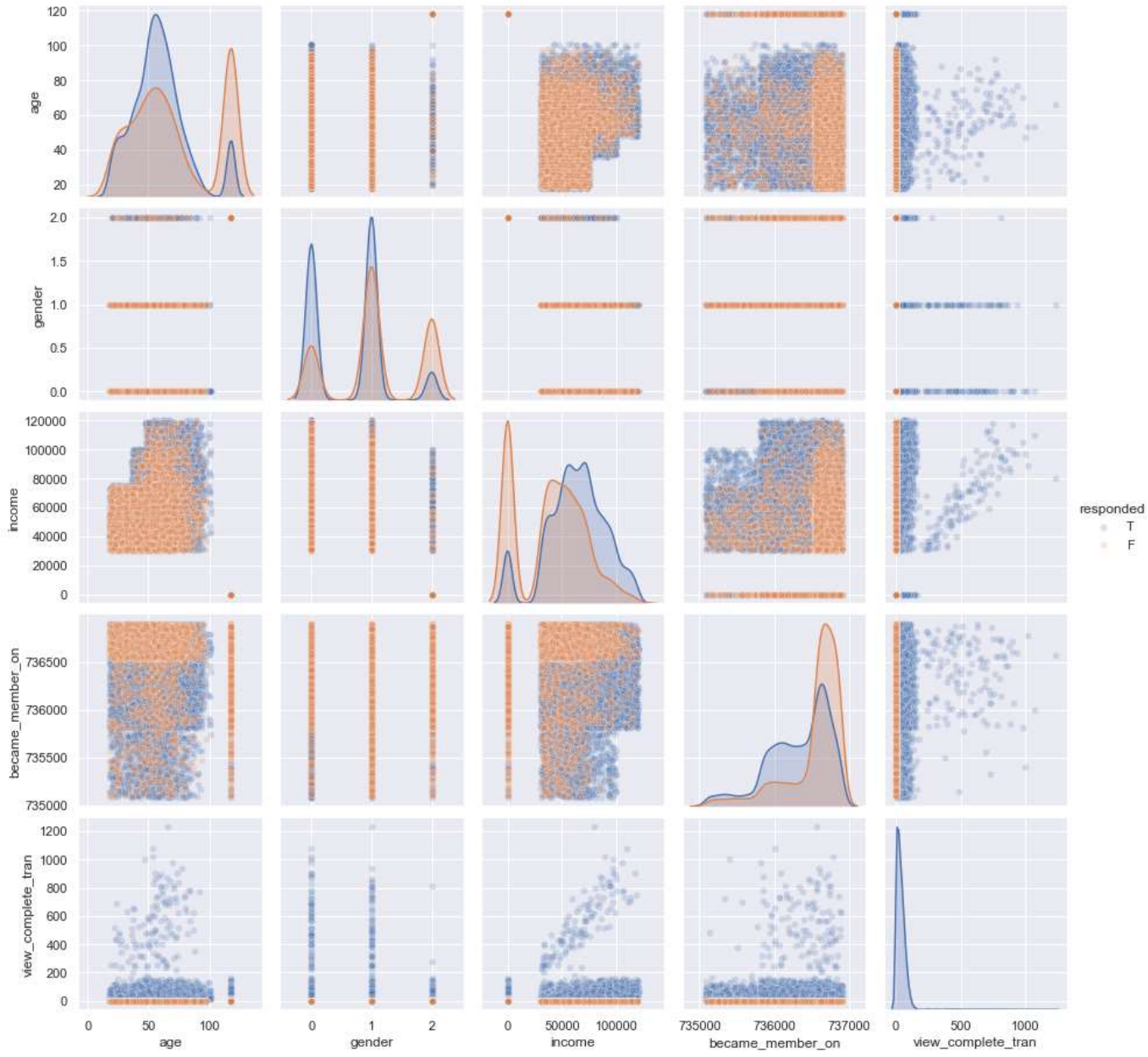
	person	age	gender	income	became_member_on	total	view_complete_tran	noview_complete_tran	receive	rec_bogo	...
0	0009655768c64bdeb2e877511632db8f	33	M	72000.0	736440	127.60	22.68	104.92	5	1	...
1	00116118485d4dfda04fdbaba9a87b5c	118	NaN	NaN	736809	4.09	0.00	4.09	2	2	...
2	0011e0d4e6b944f998e987f904e8c1e5	40	O	57000.0	736703	79.46	33.98	45.48	5	1	...
3	0020c2b971eb4e9188eac86d93036a77	59	F	90000.0	736027	196.86	34.87	161.99	5	2	...
4	0020ccbbb6d84e358d3414a3ff76cffd	24	F	60000.0	736279	154.05	36.50	117.55	4	2	...

5 rows × 28 columns

For visualization purpose, we applied the processes below to make a new variable "responded" to be "T" when viewed&completed offer amount is non-zero, otherwise "F"; And made gender variable to be 0,1,2 in order to be shown to the axis:

```
person_all_information['responded'] =
person_all_information.view_comp.apply(lambda x: 'T' if x!=0 else
'F')
person_all_information['gender']=person_all_information.gender.apply(
lambda x: 1 if x=='M' else (0 if x=='F' else 2))
```

1. We then drew pair-plot for variables ‘age’,’gender’,’ income’,’became_member_on’,’view_complete_tran’,’ responded’ and fill any NaN with 0:



Pair-plot for multiple variable

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 females tend to respond to offer compared with males 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.

2. Next, we make an analysis based on the offer type:

Offer Type

```
complete_rate=person_all_information.comp.sum()/person_all_information.receive.sum()
complete_rate
```

```
0.4408957415565345
```

```
bogo_complete_rate=offer_norec_comp.complete_bogo.sum()/person_all_information.rec_bogo.sum()
bogo_complete_rate
```

```
0.5137882414663737
```

```
discount_complete_rate=offer_norec_comp.complete_discount.sum()/person_all_information.rec_discount.sum()
discount_complete_rate
```

```
0.5864248059984938
```

```
informational_complete_rate=offer_norec_comp.complete_informational.sum()/person_all_information.rec_informational.sum()
informational_complete_rate
```

```
0.0
```

```
vc_rate=person_all_information.view_comp.sum()/person_all_information.receive.sum()
vc_rate
```

```
0.3657174323473883
```

```
bogo_vc_rate=person_all_information.vc_bogo.sum()/person_all_information.rec_bogo.sum()
bogo_vc_rate
```

```
0.43843656753123256
```

```
discount_vc_rate=person_all_information.vc_discount.sum()/person_all_information.rec_bogo.sum()
discount_vc_rate
```

```
0.47621077483031116
```

```
informational_vc_rate=person_all_information.vc_informational.sum()/person_all_information.rec_bogo.sum()
informational_vc_rate
```

```
0.0
```

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

3. Further, we developed the heuristics based on offer channels:

```
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
```

```
0.4402244451145168
```

```
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
```

```
0.4395778979434185
```

```
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
```

```
0.4769389050631526
```

```
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
```

```
0.49000836051868
```

From above, it appears that the web might be the most efficient channel, which contributes to the highest completion rate.

Part IV. 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.

1. From the above analysis, we know that age, gender, income, membership date, offer type can affect whether a customer responds to an offer.

Since the offer record generated in Part II, step 3 is customer unique and only contains the amount of each type completed/received/viewed&completed offers, and each customer may receive more than one offer, we need to preprocess the data again such that we have train/test data sets without person id, only with age, income, gender, membership date, offer type, etc, as our "X", and "responded" as our "y".

Also, we note that membership date appears in the format of yymmdd, this should be either turned into ordinal variables or extract its year, month, date to capture its real sequence. Otherwise, if you directly use this yymmdd number, we will have strange results, e.g. For the pairs 20170831 vs. 20170901, and 20170901 vs. 20170902, they are both 1-day-distant pairs, however, the number difference is not the same. Here, we chose to use extracting year&month of membership date.

Furthermore, the "gender" values are in M, F, O, and NaN, while "responded" values are in BOGO, discount, and informational, which both are categorical values. However, most machine learning algorithm requires the input to be numeric. Hence, these two variables should be turned into dummy variables.

For a limited time, I just simply dropped all the NaNs and conducted the actions above(including train_test_split) and got the train/test data sets we want

	age	income	gender_F	gender_M	gender_O	gender_nan	member_year_2013.0	member_year_2014.0	member_year_2015.0
6050	87.0	97000.0	1	0	0	0	0	0	0
556	59.0	83000.0	1	0	0	0	0	0	0
10245	47.0	36000.0	0	1	0	0	0	0	0
12685	58.0	97000.0	1	0	0	0	0	0	0
14741	36.0	50000.0	1	0	0	0	0	1	0

5 rows × 29 columns

s

Observing the labels, we also found that they are **imbalanced**, approximately 2:1 for Positive vs. Negative labels:

```
y_train.value_counts()
```

```
1    11712
0     6574
Name: label, dtype: int64
```

```
y_test.value_counts()
```

```
1    5034
0    2804
Name: label, dtype: int64
```

f

Hence, we should use an F1 score instead of accuracy, since the F1 score is a metric to balance recall&precision and to deal with imbalanced labels.

Recall the definition of F1 score:

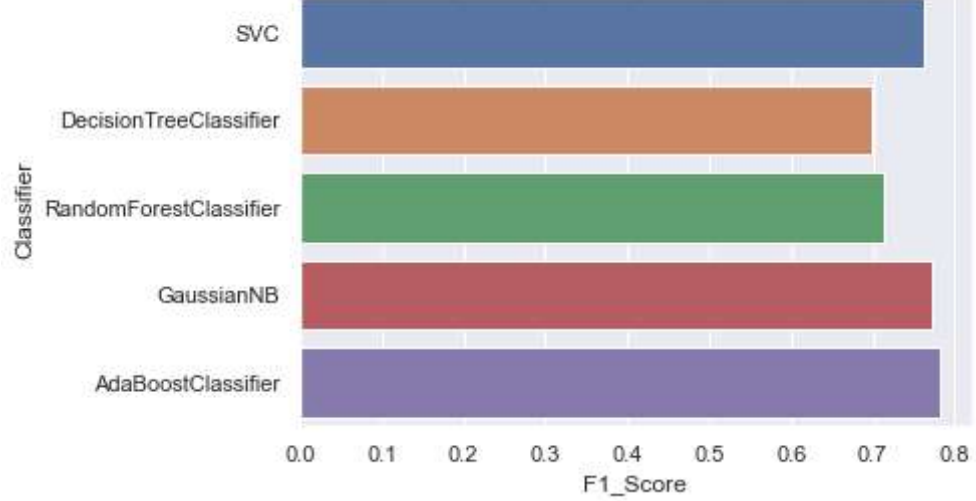
$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

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

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

Then we scaled the X_train & X_test to [0,1] and fit the training data into several classifiers(with its default mode) and got the F1 score(ranges from 0 to 1) for each model as shown below:

Classifier F1_Score



We find that in default mode, AdaBoost performed the best. Hence, we can conduct Gridsearch on it and find the best parameters:

```
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))
```

```
Training F1_score is: 0.7881944444444445
Test F1_score is: 0.7857584592278469
```

Saving the best parameters:

```
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=
3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=42,
            splitter='best'),
          learning_rate=0.1, n_estimators=10, random_state=None)
```

2. Next, we made a prediction engine such that we take in customer info, offer type, etc, then we can transform all these information into a 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, rather than simply seeing the test set performance without actually having any real-life usage.

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

Respond!

	age	income	gender_F	gender_M	gender_O	gender_nan	member_year_2013.0	member_year_2014.0	member_year_2015.0	member_year_2016.0	...	member_month_7.0	member_month_8.0
0	53	83000.0	1	0	0	0	0	0	0	0	...	1	0

1 rows × 29 columns

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

Respond!

	age	income	gender_F	gender_M	gender_O	gender_nan	member_year_2013.0	member_year_2014.0	member_year_2015.0	member_year_2016.0	...	member_month_7.0	member_month_8.0
0	50	34000.0	1	0	0	0	0	0	0	0	...	0	1

1 rows × 29 columns

After building the prediction engine, we test it using a random sample of the raw profile, and chose one type of offer('BOGO',' discount' or 'informational') and found it actually works. At first glance of the data above(since the function not only replies you "respond" or not, it also returns the customer info for you to check whether this response makes sense.), the first customer has a high income, and the gender is female(in the above analysis, high income and female tends to respond to the offer), while the second person is male with lower income without responding to the offer. Also, these responses may also result from the fact that the 'discount' offer is more preferable than 'BOGO'. This result is to some extent acceptable, although more samples should be tested to validate such a conclusion.

Conclusion

To conclude, in this project, 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 offers), 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, the 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.

at last i can not able to properly give all the code hence go through this repo.

- https://github.com/kartikeyshaurya/Capstone__Starbuks

Machine Learning Data Science