Capstone Report

Contents

1. Define the problem, investigate potential solutions and performance metrics

Problem

Predict a customer response to different promotion/offer activities. We should be able to predict a customer spending with all the information starbucks has regarding the customer. If the customer are willing to buy without any promotion, it will not make sense for starbucks to give reduce cost for that customer. However, if the customer initially do not plan to buy, but make a move after seeing the promotion, it will be a win for starbucks

Promotion/offer: Either buy one get one free(BOGO) or a fixed discount

Potential Solutions

Since our aim is to predict a customer success rate to different promotion. We must have a definition of success. For a promotion to be successful to a customer, the customer must meet the following three criteria.

- Receiving the promotion material (regardless of method, mail, text, social media,etc)
- Viewing the promotion material
- Be enticed by the promotion material and complete the offer that's given

Note that fulfilling the second point will result in the first point being fulfilled. Given the above three criteria, I have proposed a measure whether a specific customer is receptive to a promotion by the following metric.

Success of a specific person = (viewing rate of promotion materials >70%) AND (completion rate of promotion >70%)

We will look at demographic data given (age, membership_start_date, annual income, gender) and try to predict the success of each promotion event using the above metric

Performance Metrics

Before we start on performance metrics, we want to know the success rate for promotional materials with current data that we have.

For customer who was given BOGO promotion, the success rate is 33.187%

For customer who was given the discount promotion, the success rate is 33.52%

```
In [290]: df['bogo_success'] = (df.bogo_view_rate>70) & (df.bogo_com_rate>70)
          df['dis_success'] = (df.dis_view_rate>70) & (df.dis_com_rate>70)

In [315]: print(df.bogo_success.sum())
          df.bogo_success.mean()

          4848

Out[315]: 0.33187294633077763

In [314]: print(df.dis_success.sum())
          print(df.dis_success.mean())

          4898
          0.335295728368
```

We will evaluate the performance of our models by using F1 score. F1 score conveys the balance between precision and recall. F1 score is defined as the following

The traditional F-measure or balanced F-score (**F_1 score**) is the harmonic mean of precision and recall:

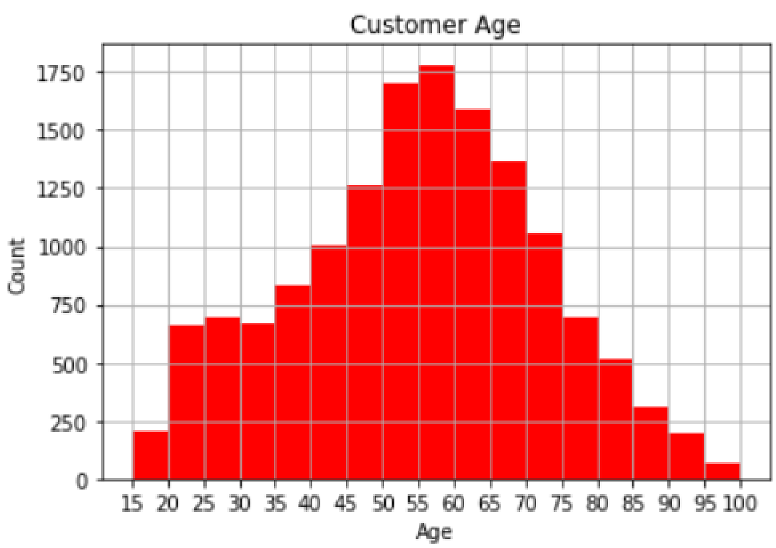
$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

We seek to maximize F1 score for our model.

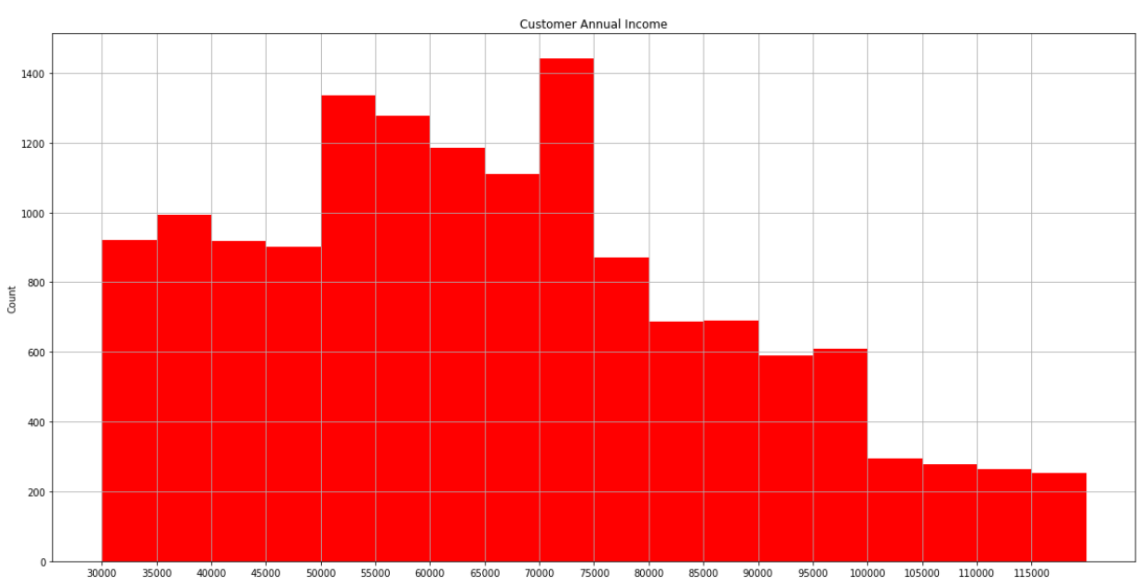
2. Analyze the problem through visualization and data exploration

We first visualize data on the demographic of Starbuck's Members.

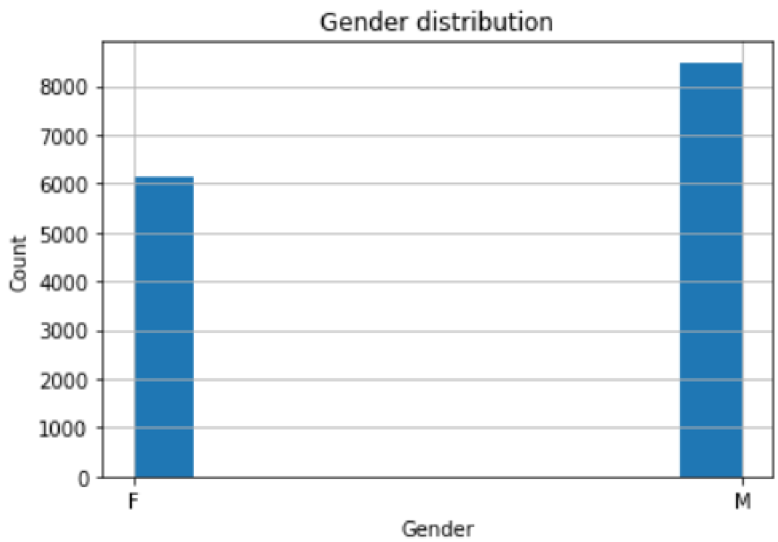
1. Age: Surprisingly most of the customers are middle aged people from 55-60. We can see that the graph bellow shows the customer based is approximately normal.



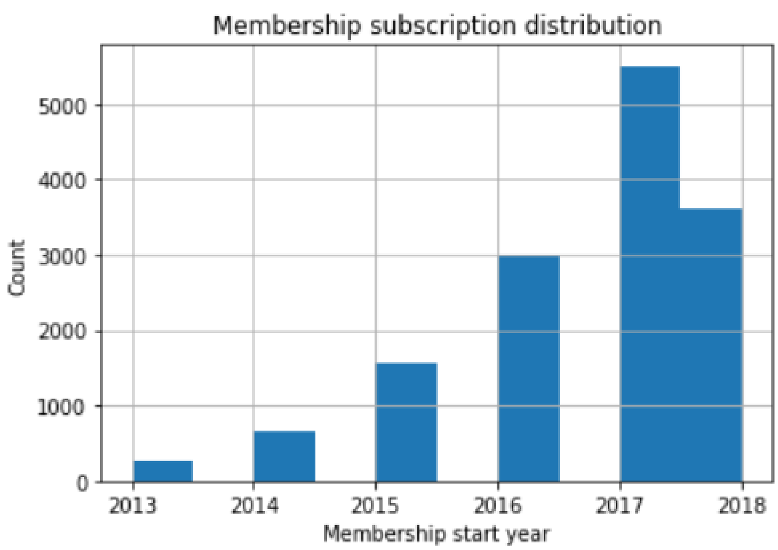
2. Annual Income: Most of the people who drinks starbucks are not excessively rich. They are earning 75000 and below. We can see the 75 percentile for income is 80,000



3. Gender: Surprisingly, there are more male members compared to female members.

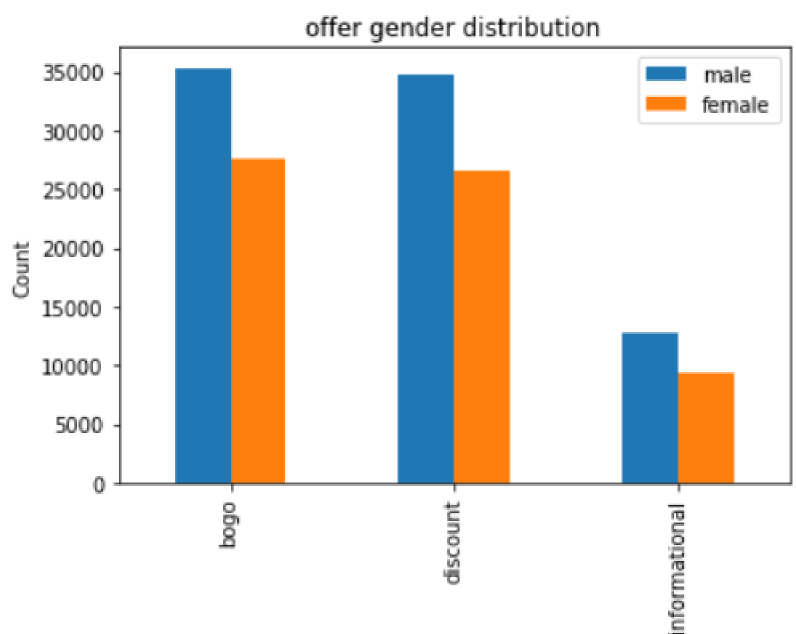


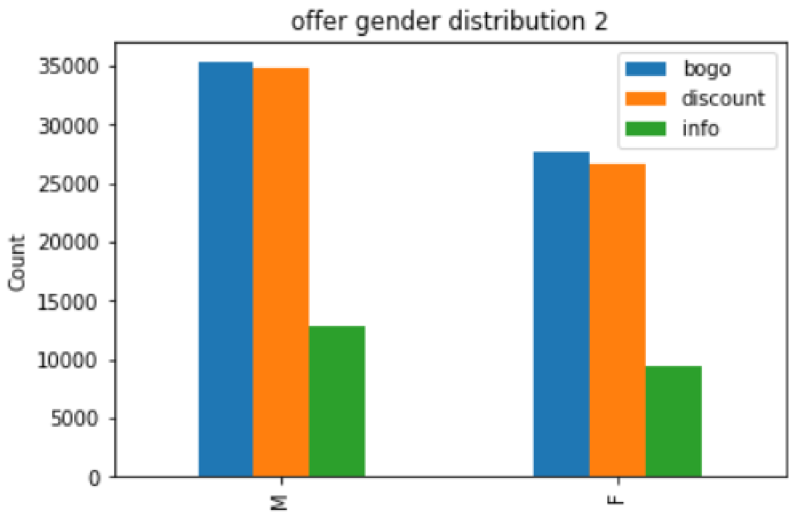
4. Became_member_on: We have two charts here, plotted year on year and month on month. Do note that the earliest data point is 29 July 2013 and latest data point is 26 July 2018. This mean that for year on year data, 2013 and 2018 data point might be underestimated. We can see there's a year on year increase in membership. (2018 only has 7/12 of the membership for that year). For monthly data point, there's no significant visible seasonal trend, except membership subscription is slightly more from August to January.



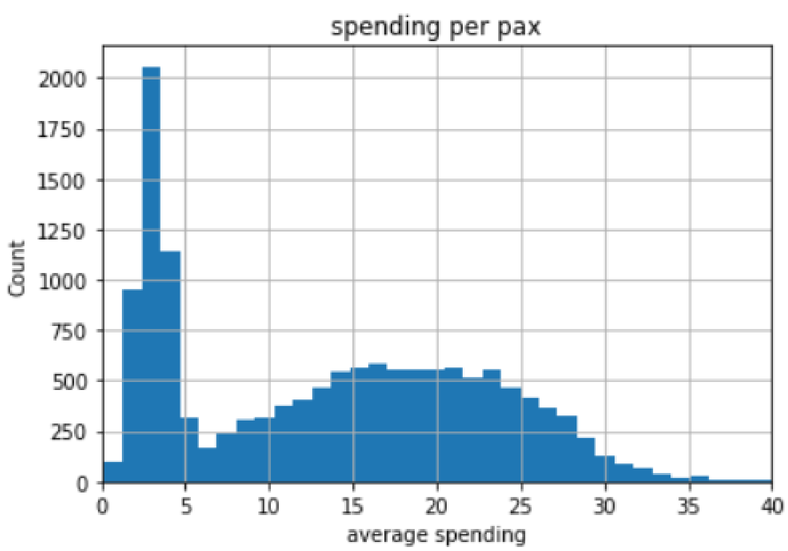
In the following visualization below, we will combined members' transaction data with the member details. We do a left join using transaction data as main table and customer detail as secondary table. More about data transformation in the next section.

5. Investigating whether the data fair towards each gender, or is it heavily scaled. We see that the distribution is similar in both male and female.

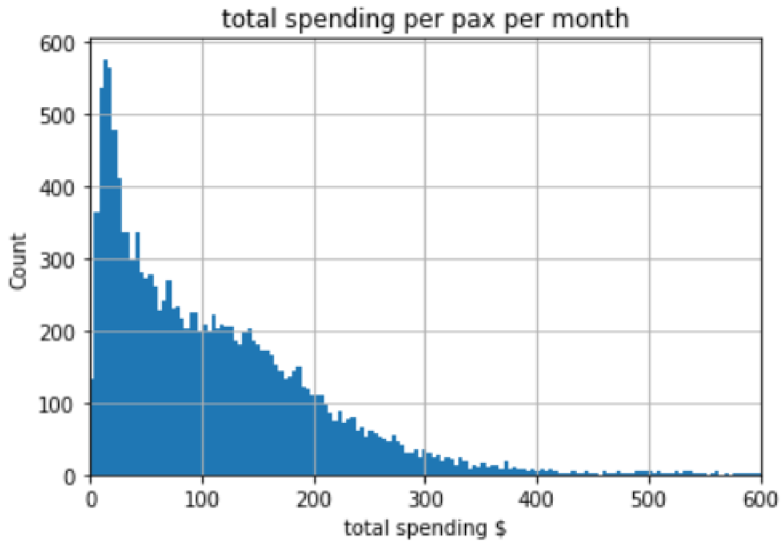




6. Average spending of each person: This looks like a bi-modal model, at around $3 and $18. This make sense as the people who spend there are most likely segregated into just a coffee break or a complete meal and a coffee.



7. Total spending for each person: Note that this dataset is over a one month period, so the following is graph of distribution of total spending per month for each pax. The distribution below looks like an exponential distribution. Few people spend more than $500 at starbucks a month(likely the people who dines there frequently) and it slowly decays.



3. Preprocessing of data

The structure of this section will be split into three steps.

1. Preprocessing of each individual tables
2. Combining three tables into 1
3. Further processing after combination
4. Preprocessing for ML algorithms

1a) Preprocessing of Portfolio.json

Table is read in as offers. 10 rows x 6 cols

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

Since the data is small, we can print the whole data out to take a look at it.

In [206]:
```
print(offers.shape)
print(offers.columns)
offers
```

```
(10, 6)
Index(['channels', 'difficulty', 'duration', 'id', 'offer_type', 'reward'], dtype='object')
```

Out[206]:

	channels	difficulty	duration	id	offer_type	reward
0	[email, mobile, social]	10	7	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10
1	[web, email, mobile, social]	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10
2	[web, email, mobile]	0	4	3f207df678b143eea3cee63160fa8bed	informational	0
3	[web, email, mobile]	5	7	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5
4	[web, email]	20	10	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5
5	[web, email, mobile, social]	7	7	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	3
6	[web, email, mobile, social]	10	10	fafdcd668e3743c1bb461111dcafc2a4	discount	2
7	[email, mobile, social]	0	3	5a8bc65990b245e5a138643cd4eb9837	informational	0
8	[web, email, mobile, social]	5	5	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5
9	[web, email, mobile]	10	7	2906b810c7d4411798c6938adc9daaa5	discount	2

Under the channels columns, a list is hard to manipulate and work with. We will perform a one hot encoding of it and drop the channels columns

```
from sklearn.preprocessing import MultiLabelBinarizer
mlb = MultiLabelBinarizer()
offers = offers.join(pd.DataFrame(mlb.fit_transform(offers.pop('channels')),
                          columns=mlb.classes_,
                          index=offers.index))
```

In [210]:
```
offers
```

Out[210]:

	difficulty	duration	id	offer_type	offer_reward	email	mobile	social	web
0	10	7	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10	1	1	1	0
1	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10	1	1	1	1
2	0	4	3f207df678b143eea3cee63160fa8bed	informational	0	1	1	0	1

1b)Pre-processing of profile.json

Table is read in as customers, 17000 rows X 5 columns

In [211]:
```
print(customers.shape)
print(customers.columns)
customers.head()
```

```
(17000, 5)
Index(['age', 'became_member_on', 'gender', 'id', 'income'], dtype='object')
```

Out[211]:

	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN

The first manipulation here is making became_member_on into a datetime column.

```
In [213]: customers['became_member_on']=pd.to_datetime(customers.became_member_on,format = "%Y%m%d")
```

Next manipulation is to see amount of NaN in the data

```
In [212]: customers.isna().sum()

Out[212]: age                    0
          became_member_on       0
          gender              2175
          id                     0
          income              2175
          dtype: int64
```

Interestingly, the amount of NaN in gender and income matches. Let's see whether is this a coincidence or a None gender is equivalent to a NaN income.

```
In [214]: customers.isna().sum(axis=1).value_counts()

Out[214]: 0    14825
          2     2175
          dtype: int64
```

Summing the NaN row wise, we realized that either the rows have 0 NaN or it has 2 NaN. We conclude **that None gender is equivalent to a NaN income**. We go ahead and drop the NaN rows as only 12.8% of the rows are affected

```
In [215]: 2175/(14825+2175) #12% of data is private

Out[215]: 0.12794117647058822
```

Under the description of the tables given, it is said that if the age is unknown, it will be replaced with a 118. We will see the amount of data affected by this erroneous age data.

```
In [217]: customers.age[customers.age>100].value_counts()

Out[217]: 118    2175
          101       5
          Name: age, dtype: int64

In [218]: 118/17000*100 ## 0.694%, not a very likely number, we will drop off these anomally.

Out[218]: 0.6941176470588235
```

Approximately 0.694%. We will go ahead and drop these rows too.

```
In [15]: customers = customers[customers.age<118].reset_index(drop=True)
```

We also noticed that some of the genders are “O”. As this is not a popular gender type, and it represent a small fraction of the dataset, <1.5%. we will drop off these rows too.

```
In [18]: customers.gender.value_counts()

Out[18]: M    8484
         F    6129
         O     212
         Name: gender, dtype: int64

In [101]: 212/(8484+6129+212)

Out[101]: 0.014300168634064081

In [19]: customers = customers[customers.gender != 'O']
```

1c) Pre-processing of transcript.json

We can see there’s 306534 rows X 4 columns

```
In [20]: print(actions.shape)
         print(actions.columns)
         actions.tail()

         (306534, 4)
         Index(['event', 'person', 'time', 'value'], dtype='object')

Out[20]:
```

	event	person	time	value
306529	transaction	b3a1272bc9904337b331bf348c3e8c17	714	{'amount': 1.5899999999999999}
306530	transaction	68213b08d99a4ae1b0dcb72aebd9aa35	714	{'amount': 9.53}
306531	transaction	a00058cf10334a308c68e7631c529907	714	{'amount': 3.61}
306532	transaction	76ddbd6576844afe811f1a3c0fbb5bec	714	{'amount': 3.5300000000000002}
306533	transaction	c02b10e8752c4d8e9b73f918558531f7	714	{'amount': 4.05}

We first convert the time here to the day number. As the offers table duration is quoted in days.

```
In [21]: actions['time'] = np.ceil(actions.time/24)
```

We do note that the value columns is a dictionary that is nasty to manipulate. We will unpack the dictionary into columns by using json_normalize

```
In [24]: from pandas.io.json import json_normalize
         temp = json_normalize(actions['value']) ## note that "off
         temp.head()
```

Out[24]:

	amount	offer id	offer_id	reward
0	NaN	9b98b8c7a33c4b65b9aebfe6a799e6d9	NaN	NaN
1	NaN	0b1e1539f2cc45b7b9fa7c272da2e1d7	NaN	NaN
2	NaN	2906b810c7d4411798c6938adc9daaa5	NaN	NaN
3	NaN	fafdcd668e3743c1bb461111dcafc2a4	NaN	NaN
4	NaN	4d5c57ea9a6940dd891ad53e9dbe8da0	NaN	NaN

We realized there are two similar columns, “offer id” and “offer_id”. Before we attempt to combine them, we must ensure that no data is loss. Meaning that we can cannot have both column having information at the same time. We do a row sum of NaN on those 2 columns, if the row sum of NaN is 0, it implies that we might have information loss by combining them.

```
In [25]: temp[['offer id','offer_id']].isna().sum(axis=1).value_counts()

Out[25]: 1    167581
         2    138953
         dtype: int64
```

Thankfully at most only one of them is occupied. We will combine the two columns and delete one of them.

We combine offer id and offer_id and delete offer id

```
In [26]: temp['offer_id']= temp['offer_id'].fillna(temp['offer id'])

In [27]: del temp['offer id']
```

We will join back this temp dataframe into the main events data frame. Note that we need to merge them from the side instead of from the bottom. We will delete the value column as the information has been extracted out.

joining temp back to main table

concat rowwise

```
In [29]: events = pd.concat([actions,temp],axis=1)
```

```
In [30]: del events['value']
```

```
In [31]: events.tail()
```

Out[31]:

	event	person	time	amount	offer_id	reward
306529	transaction	b3a1272bc9904337b331bf348c3e8c17	30.0	1.59	NaN	NaN
306530	transaction	68213b08d99a4ae1b0dcb72aebd9aa35	30.0	9.53	NaN	NaN
306531	transaction	a00058cf10334a308c68e7631c529907	30.0	3.61	NaN	NaN
306532	transaction	76ddbd6576844afe811f1a3c0fbb5bec	30.0	3.53	NaN	NaN
306533	transaction	c02b10e8752c4d8e9b73f918558531f7	30.0	4.05	NaN	NaN

2) Combining all three tables into 1

Just to recap the columns of the three tables are

```
In [102]: events.columns
Out[102]: Index(['event', 'person', 'time', 'amount', 'offer_id', 'reward'], dtype='object')
```

```
In [44]: customers.columns
Out[44]: Index(['age', 'became_member_on', 'gender', 'id', 'income'], dtype='object')
```

```
In [45]: offers.columns
Out[45]: Index(['difficulty', 'duration', 'id', 'offer_type', 'offer_reward', 'email',
               'mobile', 'social', 'web'],
              dtype='object')
```

We will first merge events and customers together. To decide on the merge type, we first must note that column “person” in events and column “id” in customers are together. The former is not unique as each person can have a lot of events, while the latter is unique. It is a many to one relationship.

So in this case, we will do a left join on those two columns and drop the “id” column as it is repeated information from “person”

```
In [46]: df = pd.merge(events,customers,left_on = 'person',right_on='id',how ='left')
         df.drop(columns='id',inplace=True)
```

We will next merge the combined dataframe(df) “offer_id” column and offers “id” column together.

Again, this is a many to one relationship, so we will do a left join with df as the main table

```
In [47]: df = pd.merge(df,offers,left_on = 'offer_id',right_on='id',how='left')
         df.drop(columns='id',inplace=True)
```

We have seen from above that age column in customer table does not have any NaN. So if we do a left join of events and customers on “person” column, if there is a customer id that is found on main table but not on customer table. Age will be NaN in the merged column.

We can drop any rows with age that has NaN as we do not have the customer record for that specific customer.

```
In [48]: df.dropna(subset=['age'],inplace=True)
```

We split df into two tables, a transaction table and non_transaction table. (to facilitate easier data analysis of transactions)

```
In [55]: transaction_df = df[df.event=='transaction']
         df = df[df.event!='transaction']
```

3) Further processing after combining tables

We will manipulate the non-transaction table such that each row will represent a single customer. We need details on the number of promotion each customer received, number of promotion each customer viewed, and number of promotion each customer completed. Where promotion =[‘bogo’,’discount’]

We will do a groupy, by “person” “offer_type” and “event”, do a count and unstack(from series to a multi-Index dataframe) them

```
In [65]: test = df.copy()

In [66]: test = test.groupby(['person','offer_type','event'])['index'].count().unstack(level=1).unstack()

In [67]: test.head()
```

Out[67]:

offer_type	bogo			discount			informational		
event	offer completed	offer received	offer viewed	offer completed	offer received	offer viewed	offer completed	offer received	offer viewed
person									
0009655768c64bdeb2e877511632db8f	1.0	1.0	1.0	2.0	2.0	1.0	NaN	2.0	2.0
0020c2b971eb4e9188eac86d93036a77	1.0	2.0	1.0	2.0	2.0	1.0	NaN	1.0	1.0
0020ccbbb6d84e358d3414a3ff76cffd	2.0	2.0	2.0	1.0	1.0	1.0	NaN	1.0	1.0
003d66b6608740288d6cc97a6903f4f0	NaN	NaN	NaN	3.0	3.0	2.0	NaN	2.0	2.0
00426fe3ffde4c6b9cb9ad6d077a13ea	NaN	NaN	NaN	1.0	4.0	1.0	NaN	1.0	1.0

Next we will flatten multi-index column to a single index column, extract relevant columns

```
In [68]: test.columns
Out[68]: MultiIndex(levels=[['bogo', 'discount', 'informational'], ['offer completed', 'offer received', 'offer viewed']],
                    labels=[[0, 0, 0, 1, 1, 1, 2, 2, 2], [0, 1, 2, 0, 1, 2, 0, 1, 2]],
                    names=['offer_type', 'event'])
```

```
In [69]: test.columns = ['_'.join(col) for col in test.columns]
```

```
In [70]: test.drop(columns= ['informational_offer completed',"informational_offer received","informational_offer viewed"],inplace=True)
```

```
In [71]: test.head()
```

Out[71]:

person	bogo_offer completed	bogo_offer received	bogo_offer viewed	discount_offer completed	discount_offer received	discount_offer viewed
0009655768c64bdeb2e877511632db8f	1.0	1.0	1.0	2.0	2.0	1.0
0020c2b971eb4e9188eac86d93036a77	1.0	2.0	1.0	2.0	2.0	1.0
0020ccbbb6d84e358d3414a3ff76cffd	2.0	2.0	2.0	1.0	1.0	1.0
003d66b6608740288d6cc97a6903f4f0	NaN	NaN	NaN	3.0	3.0	2.0

For the main df, we only need the age, membership_join_date, gender and annual_income for each customer. Once we extracted that, we will merge df and test together on "person"

```
In [73]: df = df.groupby('person').agg({'age':"first","became_member_on":"first","gender":"first","income":"first"})
```

```
In [74]: df = pd.merge(df,test, on = 'person',how = 'left')
```

Instead of nominal value of number of offer viewed and completed, it is better to see it as a percentage of number of offers received. We will also change the membership date to just the year.

```
In [76]: df['bogo_view_rate'] = df['bogo_offer viewed']/df['bogo_offer received']*100
         df['bogo_com_rate'] = df['bogo_offer completed']/df['bogo_offer received']*100
         df['dis_view_rate'] = df['discount_offer viewed']/df['discount_offer received']*100
         df['dis_com_rate'] = df['discount_offer completed']/df['discount_offer received']*100
```

```
In [77]: df.drop(columns= ['bogo_offer viewed','bogo_offer completed','discount_offer viewed','discount_offer completed'],inplace=True)
```

```
In [78]: df.columns = ['age', 'member_start', 'gender', 'income', 'bogo_offer_rec',
               'dis_offer_rec', 'bogo_view_rate', 'bogo_com_rate',
               'dis_view_rate', 'dis_com_rate']
```

```
In [79]: df['mem_year']=df.member_start.dt.year
```

We will now create the labels for our dataset and calculate the success rate for each promotion on this dataset. Success rate definition is given in the first section of this document

Creating labels

We define an offer(bogo or discount) as successful if viewrate>70% and success rate is more than 70%

In [80]:
```
df['bogo_success'] = (df.bogo_view_rate>70) & (df.bogo_com_rate>70)
df['dis_success'] = (df.dis_view_rate>70) & (df.dis_com_rate>70)
```

In [81]:
```
print(df.bogo_success.sum())
df.bogo_success.mean()
```

```
4848
```

Out[81]:
```
0.33187294633077763
```

In [82]:
```
print(df.dis_success.sum())
print(df.dis_success.mean())
```

```
4898
0.335295728368
```

Benchmark

The current benchmark for bogo_success is 33.187%
the current benchmark for dis_success is 33.529%

Do note that only 33.18% of bogo were successful or 4848 of bogo promotion were successful and 33.52%/4898 of discount promotion were successful

4) Preprocessing for ML algorithms

We first select from rows from df and call the new dataframe "ready".

In [84]:
```
ready=df.reset_index()[['age','mem_year','gender','income','bogo_offer_rec','dis_offer_rec','bogo_success','dis_success']]
```

In [85]:
```
ready.head()
```

Out[85]:

	age	mem_year	gender	income	bogo_offer_rec	dis_offer_rec	bogo_success	dis_success
0	33.0	2017	M	72000.0	1.0	2.0	True	False
1	59.0	2016	F	90000.0	2.0	2.0	False	False
2	24.0	2016	F	60000.0	2.0	1.0	True	True
3	26.0	2017	F	73000.0	NaN	3.0	False	False
4	19.0	2016	F	65000.0	NaN	4.0	False	False

We will do a one hot encoding for categorical data(gender) and change the ensure the rest of dataframe is of right type

One hot encoding for categorical data

```
In [86]: ready['isMale']=ready['gender'].map({"M":1,'F':0})
         # ready = pd.concat([ready,pd.get_dummies(data=ready.mem_year, drop_first=True)],axis=1)
```

```
In [87]: ready[['age','bogo_success','dis_success']] = ready[['age','bogo_success','dis_success']].astype('int')
```

```
In [88]: ready.dtypes
```

```
Out[88]: age                  int64
         mem_year             int64
         gender              object
         income             float64
         bogo_offer_rec     float64
         dis_offer_rec      float64
         bogo_success         int64
         dis_success          int64
         isMale               int64
         dtype: object
```

We will next split the data into 80% training and 20% testing

```
In [90]: from sklearn import linear_model
         from sklearn.model_selection import train_test_split
         bogo_X = np.squeeze(ready[ready.bogo_offer_rec>0][['age','mem_year','isMale','income']].values)
         dis_X = np.squeeze(ready[ready.dis_offer_rec>0][['age','mem_year','isMale','income']].values)
         bogo_Y = np.squeeze(ready[ready.bogo_offer_rec>0][['bogo_success']].values)
         dis_Y = np.squeeze(ready[ready.dis_offer_rec>0][['dis_success']].values)

         bogo_X_train, bogo_X_test, bogo_Y_train, bogo_Y_test = train_test_split(bogo_X, bogo_Y, test_size = .20, random_state = 9)
         dis_X_train, dis_X_test, dis_Y_train, dis_Y_test = train_test_split(dis_X, dis_Y, test_size = .20, random_state = 9)
```

4. Implement Algorithms and metrics of choice

Next, we will run through the data with following machine learning algorithm

1. Logit Regression
2. K- Nearest Neighbour
3. Gaussian Naives Bayes
4. Support vector classification
5. Adaptive Boosting

Evaluation Metrics is defined in section 1. We will be using F1 score.

5. Results on performance of models used

Model 1: Logistic Regression

```
bogo_1 = linear_model.LogisticRegression()
dis_1 = linear_model.LogisticRegression()
bogo_1.fit(bogo_X_train, bogo_Y_train)
dis_1.fit(dis_X_train, dis_Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
print('Bogo train F1_score {}'.format( bogo_1.score(bogo_X_train,bogo_Y_train)))
print('Bogo Test F1_score {}'.format(bogo_1.score(bogo_X_test,bogo_Y_test)))
print('Discount train F1_score {}'.format(dis_1.score(dis_X_train,dis_Y_train)))
print('Discount test Test F1_score {}'.format(dis_1.score(dis_X_test,dis_Y_test)))
```

```
Bogo train F1_score 0.6175387596899224
Bogo Test F1_score 0.6178294573643411
Discount train F1_score 0.6088478366553233
Discount test Test F1_score 0.5905909797822706
```

Model 2: KNN

```
from sklearn.neighbors import KNeighborsClassifier
bogo_2 = KNeighborsClassifier()
dis_2 = KNeighborsClassifier()
bogo_2.fit(bogo_X_train, bogo_Y_train)
dis_2.fit(dis_X_train, dis_Y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

```
print('Bogo train F1_score {}'.format( bogo_2.score(bogo_X_train,bogo_Y_train)))
print('Bogo Test F1_score {}'.format(bogo_2.score(bogo_X_test,bogo_Y_test)))
print('Discount train F1_score {}'.format(dis_2.score(dis_X_train,dis_Y_train)))
print('Discount test Test F1_score {}'.format(dis_2.score(dis_X_test,dis_Y_test)))
```

```
Bogo train F1_score 0.7410852713178294
Bogo Test F1_score 0.5906976744186047
Discount train F1_score 0.7271754982984929
Discount test Test F1_score 0.5987558320373251
```

Model 3: GaussianNB

```
from sklearn.naive_bayes import GaussianNB
bogo_3 = GaussianNB()
dis_3 = GaussianNB()
bogo_3.fit(bogo_X_train, bogo_Y_train)
dis_3.fit(dis_X_train, dis_Y_train)
```

```
GaussianNB(priors=None)
```

```
print('Bogo train F1_score {}'.format( bogo_3.score(bogo_X_train,bogo_Y_train)))
print('Bogo Test F1_score {}'.format(bogo_3.score(bogo_X_test,bogo_Y_test)))
print('Discount train F1_score {}'.format(dis_3.score(dis_X_train,dis_Y_train)))
print('Discount test Test F1_score {}'.format(dis_3.score(dis_X_test,dis_Y_test)))
```

```
Bogo train F1_score 0.6400193798449613
Bogo Test F1_score 0.6372093023255814
Discount train F1_score 0.6235294117647059
Discount test Test F1_score 0.6298600311041991
```

Model 4: SVC

```
from sklearn.svm import SVC
bogo_4 = SVC()
dis_4 = SVC()
bogo_4.fit(bogo_X_train, bogo_Y_train)
dis_4.fit(dis_X_train, dis_Y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

```
print('Bogo train F1_score {}'.format(bogo_4.score(bogo_X_train,bogo_Y_train)))
print('Bogo Test F1_score {}'.format(bogo_4.score(bogo_X_test,bogo_Y_test)))
print('Discount train F1_score {}'.format(dis_4.score(dis_X_train,dis_Y_train)))
print('Discount test Test F1_score {}'.format(dis_4.score(dis_X_test,dis_Y_test)))
```

```
Bogo train F1_score 0.811531007751938
Bogo Test F1_score 0.6065891472868217
Discount train F1_score 0.8073894020418084
Discount test Test F1_score 0.604199066874028
```

Model 5: Ada Boost

```
from sklearn.ensemble import AdaBoostClassifier
bogo_5 = AdaBoostClassifier()
dis_5 = AdaBoostClassifier()
bogo_5.fit(bogo_X_train, bogo_Y_train)
dis_5.fit(dis_X_train, dis_Y_train)
```

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None)
```

```
print('Bogo train F1_score {}'.format(bogo_5.score(bogo_X_train,bogo_Y_train)))
print('Bogo Test F1_score {}'.format(bogo_5.score(bogo_X_test,bogo_Y_test)))
print('Discount train F1_score {}'.format(dis_5.score(dis_X_train,dis_Y_train)))
print('Discount test Test F1_score {}'.format(dis_5.score(dis_X_test,dis_Y_test)))
```

```
Bogo train F1_score 0.6736434108527132
Bogo Test F1_score 0.6593023255813953
Discount train F1_score 0.6516285853184249
Discount test Test F1_score 0.6590202177293935
```

6. Conclusions

Most models give a testing F1 score of above 60%, a performance that is twice of benchmark. bogo_5,dis_5 , Adaboost has the best performance of around 66%.

To predict which offers to give a specific starbucks member. We will insert the member demographic into bogo_5 and dis_5. If both returns 1, member is receptive to both discount and BOGO. If either returns 1, the member is receptive to the specific promotion. Else the member is likely not to be receptive to both promotions.