Machine Learning Capstone Report

Rajashekar Chintalapati
July 11, 2020

1. Definition

1.1 Project Overview

This report that was done on dataset of Starbucks which contains simulated data that mimics customer behavior collected from Starbucks rewards mobile app. To increase sales and customer satisfaction, Starbucks often give promotions (like BOGO - Buy one Get one, rebate & reward programs) to customers, so that customer come back more often. By identifying customer purchase patterns, companies more precisely can target the users who purchase less frequently. By doing analysis like given an offer, we can find out user conversion rate.

1.2. Problem statement

The problem I choose to solve given an offer and demographic data, whether user will complete the offer or not. By using given data, we can find the buyer pattern, what factors are impacting the sales. This data was provided by Udacity and is a simplified version of real Starbucks app.

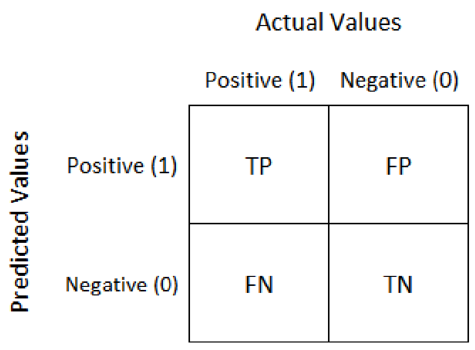
To accomplish above, I will use below strategy

1. Pre-process data
    1. Go through portfolio, profile and transcript data and find the distribution of each feature.
    2. Process demographic data in profile and process transactional data in transcript data.
    3. Find relation between each feature, do one hot encoding wherever necessary and drop any unnecessary features.
    4. Merge portfolio, profile and transcript data into one dataset and determine which feature is output class.
2. Split data into training and test data.
3. Create a benchmark with above data using Logistic Regression.
4. Train the model using Naive Bayes, sklearn ensemble methods - Adaboost, Bagging & Random Forest.
5. Validate above models prediction with test data by using metrics accuracy and f1 score.
6. Compare how above models metric score with benchmark score created with Logistic Regression.
7. Use Random Forest to determine feature importance.
8. Based on above data, check if given user demographic data, transaction details and offer details models are able to predict accurately.

1.3. Metrics

Will use accuracy and f1 score to evaluate each model score because accuracy tells us how many are correctly labeled and f1 score considers both precision and recall simultaneously (harmonic average between precision and recall), this gives a better measure on how many are not correctly classified. And will use feature importance using Random Forest model to define which features were important to decide whether will user will accept the offer or not.

Below is the confusion matrix - matrix contains True/False Positives/Negatives.



Confusion matrix - Predicted vs Actual

True Positive (TP) - Both Predicted and Actual are Positive

False Positive (FP) - Predicted Positive for Actual Negative (Type 1 error)

False Negative (FN) - Predicted Negative for Actual Positive (Type 2 error)

True Negative (TN) - Both Predicted and Actual are Negative

Accuracy tells us how many of them were labeled correctly, so this metric can be used to compare models. This is used when both TP & TN are important. In this example, if model predicts if user accepts both offer and will not accept offer correctly, that is the model we choose.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Above accuracy metric has limitations - lets take an example out 100 patients, 90 of them have cancer and if model predicted all 100 patients have cancer then accuracy comes to 90% but the model did not predict correctly even though accuracy percent is more. In order to avoid this type of cases, I will also use **F1 score** which is harmonic mean of **Precision** and **Recall.**

Precision is how many positives are predicted correctly out of all positives where as **Recall** is how many actual positives are predicted correctly.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

F-Beta score whee the β controls the degree to which precision is weighed into the F score, which allows precision and recall to be considered simultaneously. The most common value is 1, as this is where harmonic average is done between precision and recall.

$$f_\beta = (1 + \beta^2)\frac{Precision\ *\ Recall}{(\beta^2 * Precison) + Recall}$$

$$f_1 = 2 * \frac{Precision\ *\ Recall}{Precision\ +\ Recall}$$

For this project, I am considering both Accuracy and F1 score, as the target classes can be either balanced or imbalanced.

2. Analysis

2.1 Data Exploration

There are 3 datasets provided.

i) Portfolio Dataset - which contains offer ids and meta data about each offer (duration, type, etc.). This dataset has 10 records, out of them 4 has offer type as bogo, 4 has offer type has discount and 2 has offer type has informational.

- 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) - time for offer to be open, in days
- channels (list of strings)

In this dataset, I will be considering all above features, as this will help on identify which feature of offer has weight so that user completed the offer.

ii) Profile Dataset - which contains demographic data for each customer. This dataset has 17,100 records out of them 2175 records have NaN values, after removing those records, total came to 14925 records with 4 features (excluding id).

- 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

became_member_on has date, this was converted to days to calculate tenure.

iii) Transcript Dataset - which contains records for transactions, offers received, offers viewed, and offers completed. This data set has 306534 records and 4 features. I removed all informational offers as this transactions does not end up with offer complete. After removing I got 280468 records. Among that 138953 records are of transactional data which has user amount transactions.

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

From transactional data, amount spent till now, number of transactions done and time spent in test will be known.

There are total 16928 customers received offer, out of them 16578 customers made amount transactions. 16523 viewed the offer and 12774 customer completed the offer.

2.2 Exploratory Visualization

2.1.1 Portfolio dataset

In Portfolio dataset, there are 3 offer types

- 4 bogo (buy one get one)
- 4 discount (get some amount off or percent off)
- 2 informational

offer_type	reward	channels	difficulty	duration	id
bogo	4	4	4	4	4
discount	4	4	4	4	4
informational	2	2	2	2	2

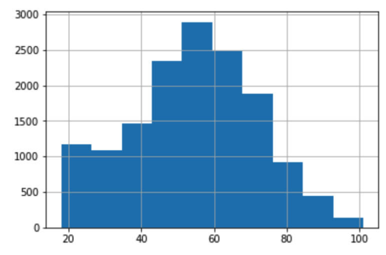
Group by on offer type

From transcript data I see that only bogo & discount promotions have offer completed, I will be dropping all informational data from transcript.

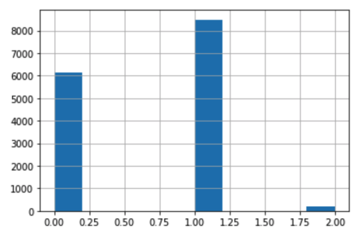
2.1.2 Profile Dataset

For the features like gender, age, income there are Nan values, there are 2175 records that are null, dropping all null values.

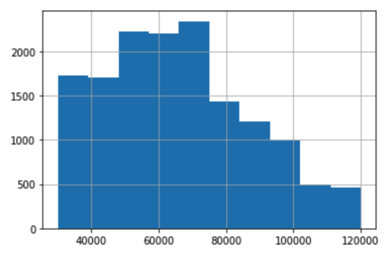
Below are the data distribution for Age, Gender and Income.



Age



Gender



Income

From above distributions, below can be concluded

- There are more users with age between 40 to 80
- There are more males than females.
- There are more users with income between 45K and 75K and number of users are less after 75K

From **became_member_on** we can find the tenure - how long the customer is with Starbucks.

Below is the distribution of became_member_on per year.

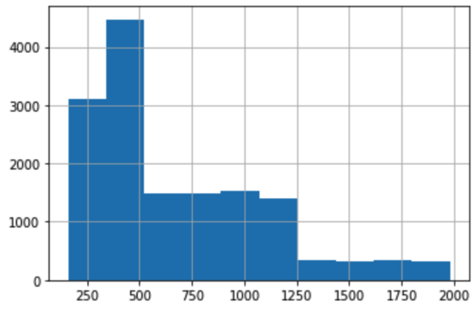
```
profile_conv['became_member_on'].apply(lambda x: str(x)[:4]).value_counts()
```

```
2017    5599
2018    3669
2016    3024
2015    1597
2014     662
2013     274
Name: became_member_on, dtype: int64
```

became_member_on per year distribution

became_member_on was given YYYYMMDD, this needs to be converted to date and get number of days, from above data we can see that we got users from 2013 to 2018. To find tenure, number of days user is with starbucks, I will use 2019-01-01 as the end date.

Below is the distribution of tenure. There are more number of users who are with starbucks from 125 to 500 days.



Tenure

2.1.3 Transcript Dataset

Transcript data has all the transactions made by user and also any rewards received as part of the promotion. The event values are given in dictionary. Since these events contain all transactions, offer received, offer viewed and offer completed, it was little difficult to extract those values into seperate features.

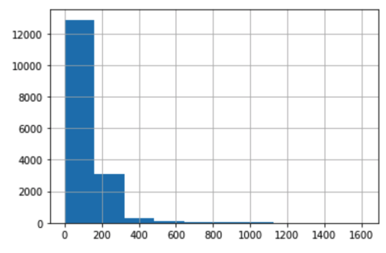
The event with type transaction has the amount, with this we can calculate how much user spent till now and number of transactions (visits) made to Starbucks. There is also time field which was in hours the user spent in this test.

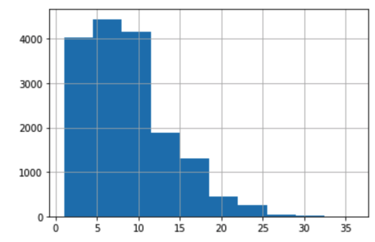
From Transaction data - Amount, number of transactions and time spent in test. From below we can see that

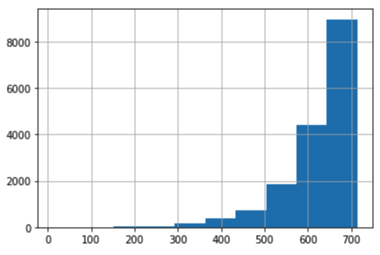
1) There are more number of users who spent at-least 180 when compared to people who spent more than 200 dollars

2) There are more number of users who made at-least 10 transactions

3) There are more number of users who spent more than 600 hours in this test.


Amount

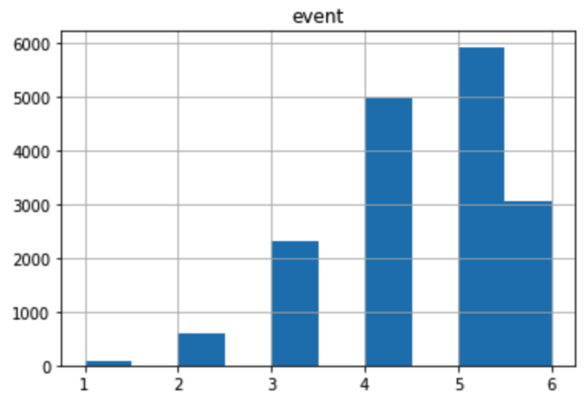

Number of transactions


Time spent in test

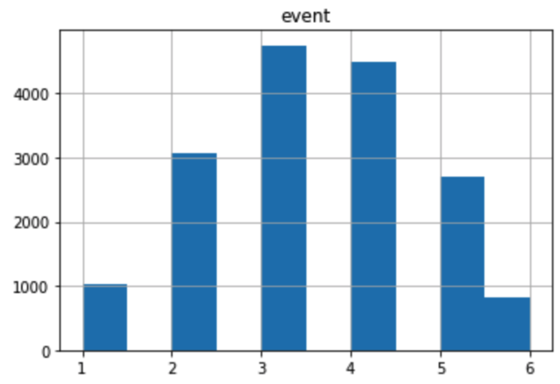
Transcript data also had offers received and offers viewed. Below is the distribution we see

1) There are more number of people who received offers more than 5 promotions.

2) There are more people who viewed 3 to 4 offers

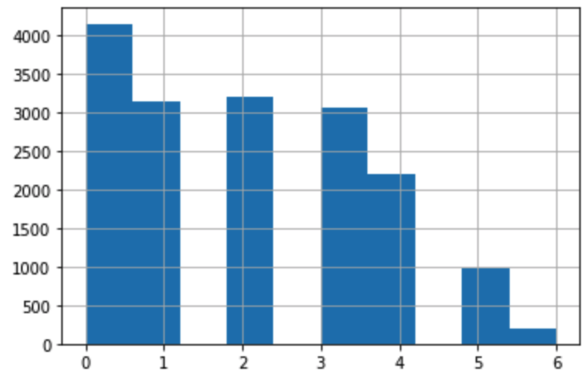


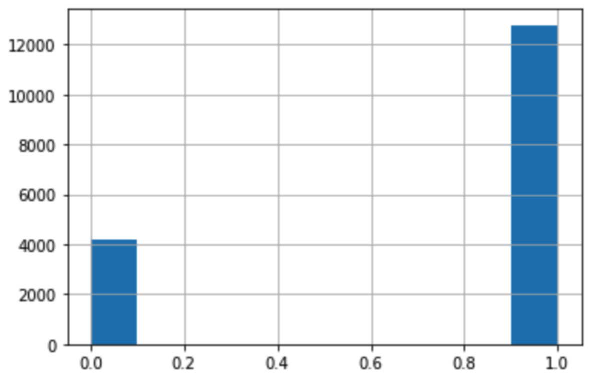
Offer received



Offer Viewed

Finally there is offer completed transactions, this tells whether user completed offer or not. I extracted number of offers completed, from which we can see that users where able to complete at-least 3 offers. There are at-least 4K users who have not completed single offer. There are at-least 12K users who completed the offer. This feature is the output class for the model.


Offer Completed


Offer success rate

2.3 Algorithms and Techniques

By going through data, first analyze data, process and prepare the data for training like removing NaN records and removing any unnecessary features, do one-hot encoding where ever necessary, once the data is ready with all required features, identify which feature is the output in this case it would be offer completed. And then split the data into train and test. And compare with benchmark results.

Since here we are trying to solve binary classification problem, where given demographic details and offer details, user can accept the offer or not. I would be using supervised learning algorithms and the most popular, fast and simple algorithm for classification is Naive Bayes which is based on Bayes theorem for calculating probabilities.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

$P(c/X) = P(x_1|c) \, * P(x_2|c) \, * ... * \, P(x_n|c) * P(c)$

where

$P(c|x)$ is posterior probability of class c given predictor(features).

$P(c)$ is the probability of class

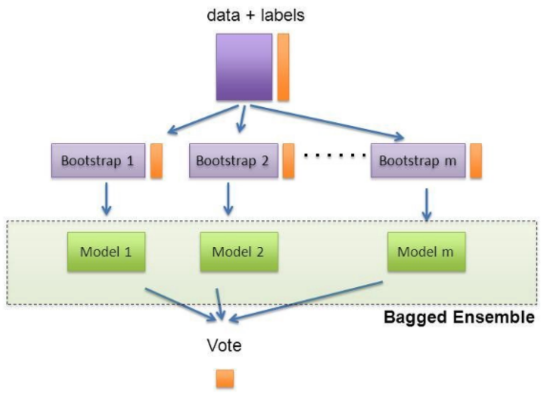
$P(x|c)$ is the likelihood which is the probability of predictor given class

$P(x)$ is the prior probability of predictor

Here given all profile features and offer features, we can quickly check what would be likelihood of customer accepting the offer. Naive Bayes does not need a lot of data to perform well. I want to check in this example whether Naive Bayes perform better or worse than logistic regression.

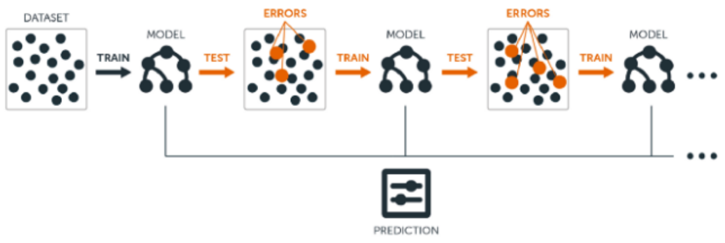
And also I want to use Ensemble methods, which usually uses predictions of different models to get better prediction. Majorly there are 2 methods, Bagging and Boosting. Ensemble methods minimize the errors that causes due to noise, bias and variance.

Bagging ensemble method creates random samples of training data and uses decision tree to build model for each sample, using majority the best is chosen as result. Random forest falls into bagging method, but it also randomly picks features in samples, thus it will further reduce variance.



Bagging

Boosting method adjusts weights after each iteration by giving higher weight to those that were poorly predicted by previous model. Thus it reduces the bias error.



Bossting

I will compare how ensemble methods perform on this classification when compared to logistic regression.

2.4 Benchmark

Using Logistic regression which is extremely efficient mechanism for calculating probabilities especially incases like binary classifications.

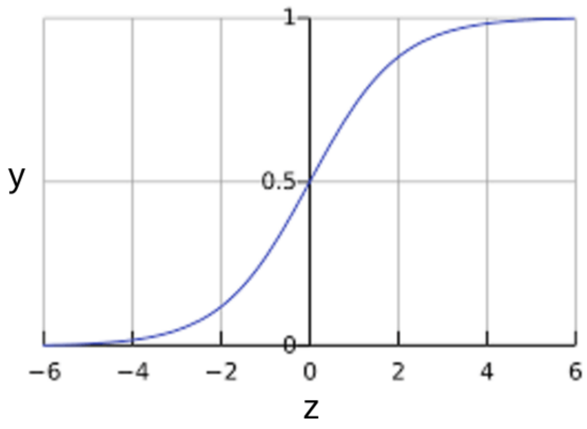
> In many cases, you'll map the logistic regression output into the solution to a binary classification problem, in which the goal is to correctly predict one of two possible labels (e.g., "spam" or "not spam") [3]

If z represents output of linear layer of a model trained with logistic regression, then sigmoid(z) will yield a value between 0 & 1.

$$y' = \frac{1}{1+e^{-z}}$$

y' is output of logistic regression model

z is $b + w_1x_1 + w_2x_2 + ... + w_Nx_N$



```
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(random_state=42).fit(X_train, y_train)
logistic_preds = logistic.predict(X_test)
print_metrics(y_test, logistic_preds, 'logistic')
```

Below is the accuracy and F1 score

```
Accuracy score for logistic : 1.0
F1 score logistic : 1.0
```

Logistic regression gave 100% as the benchmark.

3. Methodology

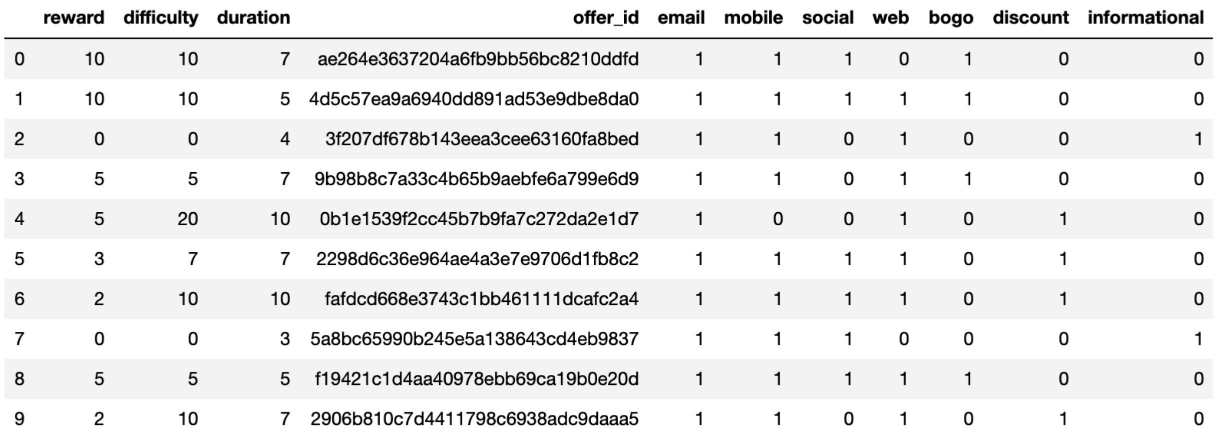
3.1 Data Preprocessing and Implementation

3.1.1 Portfolio dataset

For information offer the reward is 0, so this feature might not appeal to customers to convert, so will remove any informational offers at later point.

```
mlb = MultiLabelBinarizer()
channels_onehot = pd.DataFrame(mlb.fit_transform(portfolio['channels']),columns=mlb.classes_, index=portfolio.index)

offertype_onehot = pd.get_dummies(portfolio['offer_type'])
portfolio_conv = pd.concat((portfolio, channels_onehot, offertype_onehot), axis=1)

# Drop channels & offer_type
portfolio_conv.drop(['channels','offer_type'], axis = 1, inplace=True)

# rename to offer_id
portfolio_conv.rename(columns={'id': 'offer_id'}, inplace=True)
```

After applying one-hot encoding on channels and offer_type, below are the all the records after encoding. (After encoding the offer_type and channels features are removed).



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

Portfolio Dataset

3.1.2 Profile Dataset

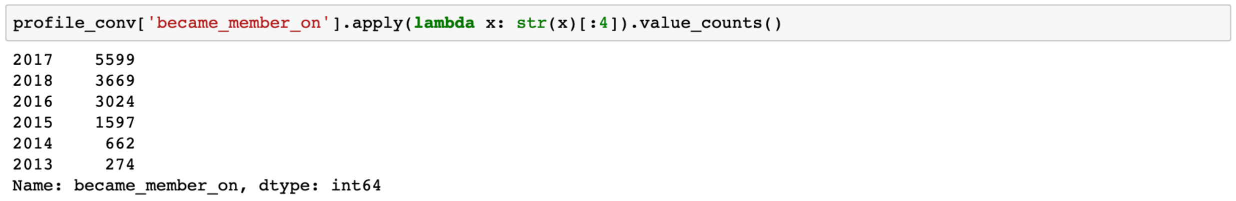
For the features like gender, age, income there are Nan values, there are 2175 records that are null, dropping all null values.

```
profile_conv = profile.dropna(axis=0)
```

From **became_member_on** we can find the tenure - how long the customer is with Starbucks.

Below is the distribution of became_member_on per year.

```
profile_conv['became_member_on'].apply(lambda x: str(x)[:4]).value_counts()
```



```
profile_conv['became_member_on'].apply(lambda x: str(x)[:4]).value_counts()
```

```
2017    5599
2018    3669
2016    3024
2015    1597
2014     662
2013     274
Name: became_member_on, dtype: int64
```

became_member_on per year distribution

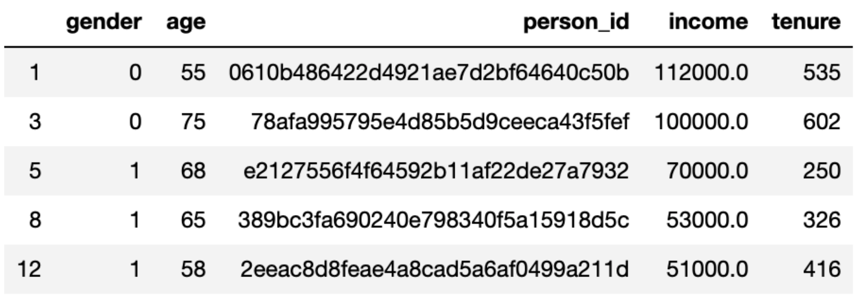
became_member_on was given YYYYMMDD, this needs to be converted to date and get number of days, from above data we can see that we got users from 2013 to 2018. To find tenure, number of days user is with starbucks, I will use 2019-01-01 as the end date.

```
def convertDateToDays(dateint):
    to_date = datetime.strptime(str(dateint),'%Y%m%d')
    from_date = datetime.strptime('20190101','%Y%m%d')
    duration = from_date - to_date
    return duration.days

# Convert to number of days
profile_conv['tenure'] = profile_conv['became_member_on'].apply(lambda x: convertDateToDays(x))
profile_conv['tenure'].hist()
```

Processing gender

```
profile_conv['gender'].replace({'F':0,'M':1,'O':2}, inplace=True)

# rename id to person_id
profile_conv.rename(columns={'id': 'person_id'}, inplace=True)
```

Final profile data, after processing data



	gender	age	person_id	income	tenure
1	0	55	0610b486422d4921ae7d2bf64640c50b	112000.0	535
3	0	75	78afa995795e4d85b5d9ceeca43f5fef	100000.0	602
5	1	68	e2127556f4f64592b11af22de27a7932	70000.0	250
8	1	65	389bc3fa690240e798340f5a15918d5c	53000.0	326
12	1	58	2eeac8d8feae4a8cad5a6af0499a211d	51000.0	416

Profile

3.1.3 Transcript Dataset

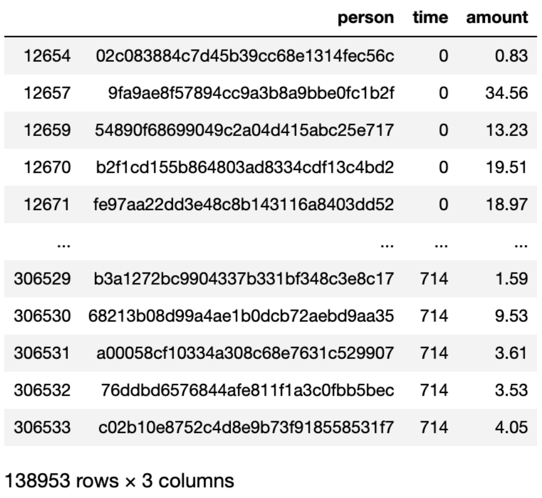
Removing information transactions as these records wont help in identifying whether user had taken offer or not. After removing these records the count came to 280468.

```
off_view_or_receive = transcript[(transcript['event'] == 'offer received') | (transcript['event'] == 'offer viewed')]

informational_offers = portfolio_conv[portfolio_conv['informational'] == 1]['offer_id'].values

inf_off_1 = off_view_or_receive[off_view_or_receive['value'] == {'offer id': informational_offers[0]}]
inf_off_2 = off_view_or_receive[off_view_or_receive['value'] == {'offer id': informational_offers[1]}]

transcript.drop(inf_off_1.index, inplace=True)
transcript.drop(inf_off_2.index, inplace=True)
```

3**.1.3.1 Processing transactional data**

Getting only transaction data. There are 138953 records.

```
transaction_data = transcript[transcript['event'] == 'transaction']
transaction_data['amount'] = transaction_data['value'].apply(lambda x: x['amount'])
transaction_data.drop(['event', 'value'], axis=1, inplace=True)
```



	person	time	amount
12654	02c083884c7d45b39cc68e1314fec56c	0	0.83
12657	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	0	34.56
12659	54890f68699049c2a04d415abc25e717	0	13.23
12670	b2f1cd155b864803ad8334cdf13c4bd2	0	19.51
12671	fe97aa22dd3e48c8b143116a8403dd52	0	18.97
...	...	...	...
306529	b3a1272bc9904337b331bf348c3e8c17	714	1.59
306530	68213b08d99a4ae1b0dcb72aebd9aa35	714	9.53
306531	a00058cf10334a308c68e7631c529907	714	3.61
306532	76ddbd6576844afe811f1a3c0fbb5bec	714	3.53
306533	c02b10e8752c4d8e9b73f918558531f7	714	4.05

138953 rows × 3 columns

User transactions

From above data, calculating how much amount was spent by each customer, number of transaction made, number of hours users was in this test.

```
# getting amount spent per customer
transaction_final = transaction_data.groupby('person').sum().drop('time', axis=1)
# getting number of transactions made by customer
transaction_final['no_of_trans'] = transaction_data.groupby('person').count().drop('time', axis=1)
# getting time spent in test by customer
transaction_final['time_spent_in_hrs'] = transaction_data.groupby('person').max().drop('amount', axis=1)

transaction_final.reset_index(level=0, inplace=True)
transaction_final.rename(columns={'person': 'person_id'}, inplace=True)
```

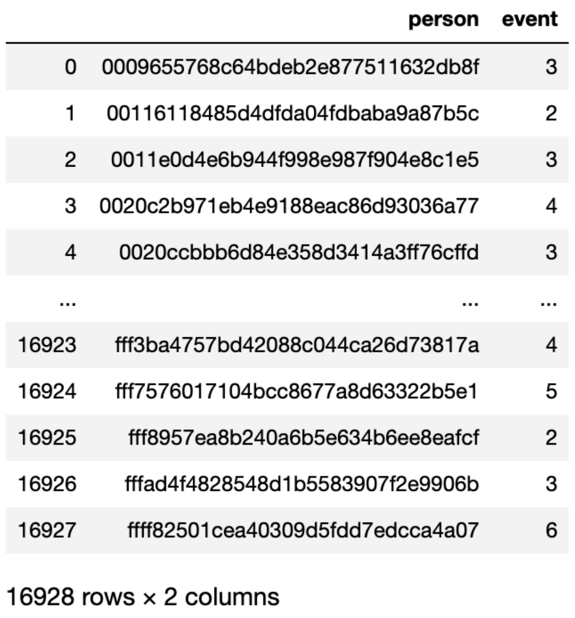
	person_id	amount	no_of_trans	time_spent_in_hrs
0	0009655768c64bdeb2e877511632db8f	127.60	8	696
1	00116118485d4dfda04fdbaba9a87b5c	4.09	3	474
2	0011e0d4e6b944f998e987f904e8c1e5	79.46	5	654
3	0020c2b971eb4e9188eac86d93036a77	196.86	8	708
4	0020ccbbb6d84e358d3414a3ff76cffd	154.05	12	672
...	...	...	...	...
16573	fff3ba4757bd42088c044ca26d73817a	580.98	11	552
16574	fff7576017104bcc8677a8d63322b5e1	29.94	6	696
16575	fff8957ea8b240a6b5e634b6ee8eafcf	12.15	5	576
16576	fffad4f4828548d1b5583907f2e9906b	88.83	12	678
16577	ffff82501cea40309d5fdd7edcca4a07	226.07	15	648

16578 rows × 4 columns

3.1.3.2 Processing offer received and viewed

Since offer completed will be the actual output event we are looking for, we can convert this to number of offers customer received and viewed.

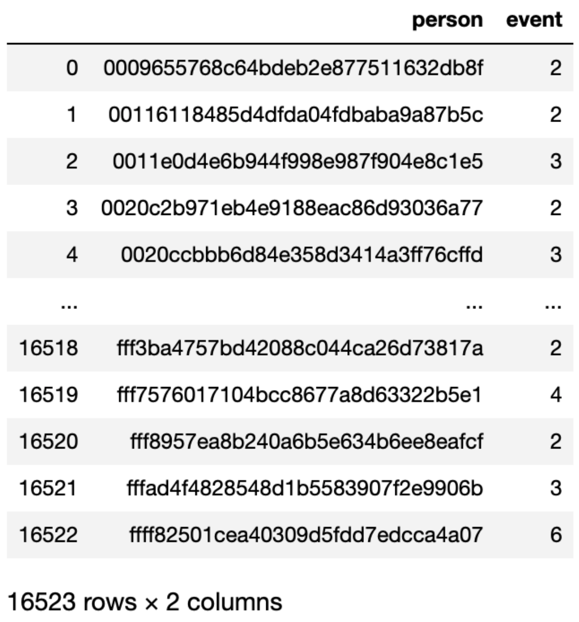
```
offer_received = pd.DataFrame(transcript[transcript['event'] == 'offer received'].groupby('person').count()['event'])
offer_received.reset_index(level=0, inplace=True)

offer_viewed = pd.DataFrame(transcript[transcript['event'] == 'offer viewed'].groupby('person').count()['event'])
offer_viewed.reset_index(level=0, inplace=True)
```



	person	event
0	0009655768c64bdeb2e877511632db8f	3
1	00116118485d4dfda04fdbaba9a87b5c	2
2	0011e0d4e6b944f998e987f904e8c1e5	3
3	0020c2b971eb4e9188eac86d93036a77	4
4	0020ccbbb6d84e358d3414a3ff76cffd	3
...	...	...
16923	fff3ba4757bd42088c044ca26d73817a	4
16924	fff7576017104bcc8677a8d63322b5e1	5
16925	fff8957ea8b240a6b5e634b6ee8eafcf	2
16926	fffad4f4828548d1b5583907f2e9906b	3
16927	ffff82501cea40309d5fdd7edcca4a07	6

16928 rows × 2 columns

Offer received



	person	event
0	0009655768c64bdeb2e877511632db8f	2
1	00116118485d4dfda04fdbaba9a87b5c	2
2	0011e0d4e6b944f998e987f904e8c1e5	3
3	0020c2b971eb4e9188eac86d93036a77	2
4	0020ccbbb6d84e358d3414a3ff76cffd	3
...	...	...
16518	fff3ba4757bd42088c044ca26d73817a	2
16519	fff7576017104bcc8677a8d63322b5e1	4
16520	fff8957ea8b240a6b5e634b6ee8eafcf	2
16521	fffad4f4828548d1b5583907f2e9906b	3
16522	ffff82501cea40309d5fdd7edcca4a07	6

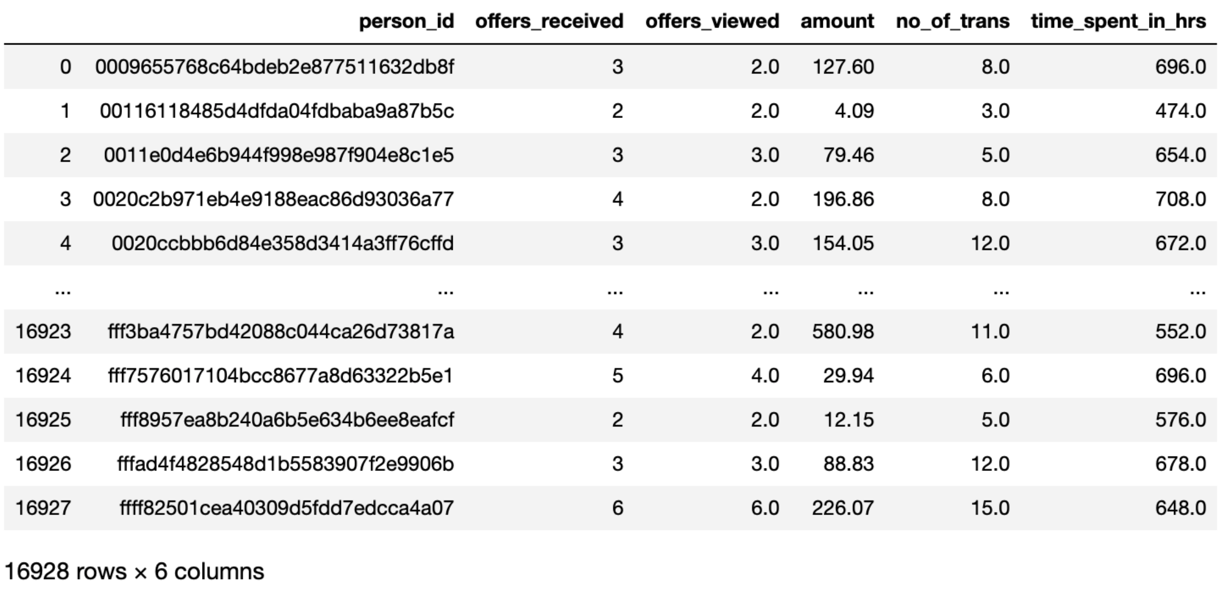
16523 rows × 2 columns

Offer received

Offer received is superset, all received offer, but only few people viewed or done transactions

- offer_received has 16928 records
- transaction_final has 16578 records
- offer_viewed has 16523

```
# renaming columns for merge
offer_received.rename(columns={'person': 'person_id', 'event': 'offers_received'}, inplace=True)
offer_viewed.rename(columns={'person': 'person_id', 'event': 'offers_viewed'}, inplace=True)

# merging offer_viewed to offer_received
offer_details = pd.merge(pd.merge(offer_received, offer_viewed, on='person_id', how="left"), \
         transaction_final, on='person_id', how="left")

# fill nan values with 0
offer_details = offer_details.fillna(0)
```



	person_id	offers_received	offers_viewed	amount	no_of_trans	time_spent_in_hrs
0	0009655768c64bdeb2e877511632db8f	3	2.0	127.60	8.0	696.0
1	00116118485d4dfda04fdbaba9a87b5c	2	2.0	4.09	3.0	474.0
2	0011e0d4e6b944f998e987f904e8c1e5	3	3.0	79.46	5.0	654.0
3	0020c2b971eb4e9188eac86d93036a77	4	2.0	196.86	8.0	708.0
4	0020ccbbb6d84e358d3414a3ff76cffd	3	3.0	154.05	12.0	672.0
...	...	...	...	...	...	...
16923	fff3ba4757bd42088c044ca26d73817a	4	2.0	580.98	11.0	552.0
16924	fff7576017104bcc8677a8d63322b5e1	5	4.0	29.94	6.0	696.0
16925	fff8957ea8b240a6b5e634b6ee8eafcf	2	2.0	12.15	5.0	576.0
16926	fffad4f4828548d1b5583907f2e9906b	3	3.0	88.83	12.0	678.0
16927	ffff82501cea40309d5fdd7edcca4a07	6	6.0	226.07	15.0	648.0

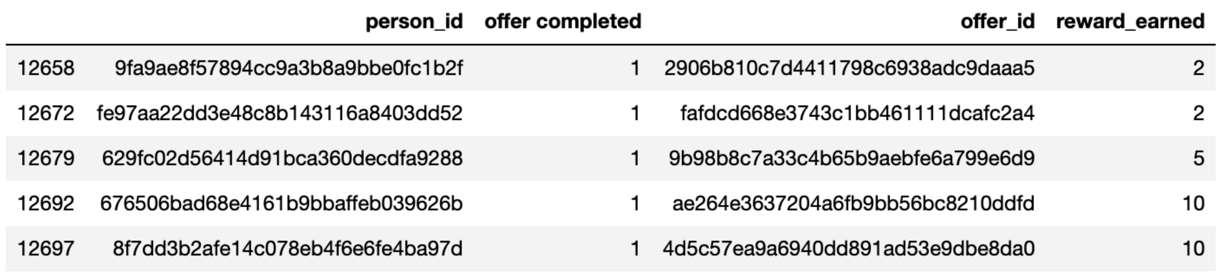
16928 rows × 6 columns

After merging transactional data, offer received and offer viewed

3.1.3.3 Processing event

After processing transaction, offer received and offer viewed, now we look into offer completed data. There are 16928 members got offer, 16523 viewed the offer, 12774 persons completed offer. All this offer completed data has dictionary object which has what offer was completed and what reward was given to the user.

```
# convert to one hot encoding
transcript_conv = pd.concat((transcript, pd.get_dummies(transcript['event'])), axis=1)

transcript_conv['offer'] = transcript_conv['value'].apply(lambda x: x['offer_id'])
transcript_conv['reward_earned'] = transcript_conv['value'].apply(lambda x: x['reward'])
transcript_conv['reward_earned'] = transcript_conv['reward_earned'].fillna(0)

# drop time, event, value columns
transcript_conv.drop(['time', 'event', 'value'], axis=1, inplace=True)

# rename to person_id and offer_id
transcript_conv.rename(columns={'person': 'person_id', 'offer': 'offer_id'}, inplace=True)
transcript_conv.head()
```



	person_id	offer completed	offer_id	reward_earned
12658	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	1	2906b810c7d4411798c6938adc9daaa5	2
12672	fe97aa22dd3e48c8b143116a8403dd52	1	fafdcd668e3743c1bb461111dcafc2a4	2
12679	629fc02d56414d91bca360decdfa9288	1	9b98b8c7a33c4b65b9aebfe6a799e6d9	5
12692	676506bad68e4161b9bbaffeb039626b	1	ae264e3637204a6fb9bb56bc8210ddfd	10
12697	8f7dd3b2afe14c078eb4f6e6fe4ba97d	1	4d5c57ea9a6940dd891ad53e9dbe8da0	10

After processing offer completed and rewards earned

3.1.3.4 Merging all datasets

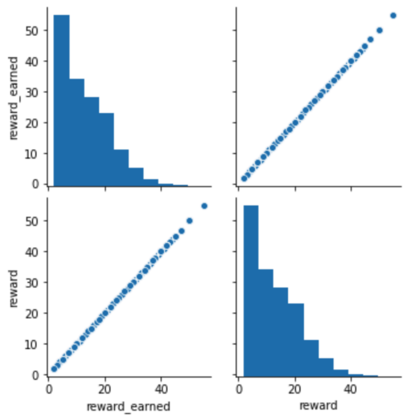
Merge portfolio and transcript data.

```
transcript_conv = pd.merge(transcript_conv, portfolio_conv, on='offer_id')
```

Grouping records so that we have only one entry for each user. Calculating count for columns to know how many emails, mobile, social, web, bogo, discount, offer completed happened for each user. And we also calcualted the sum of rewards earned, reward, difficulty and duration.

```
transcript_conv = pd.merge(transcript_conv.groupby('person_id').count()\
    [['email', 'mobile', 'social', 'web', 'bogo', 'discount', 'offer completed']].reset_index(level=0),\
        transcript_conv.groupby('person_id').sum()\
  [['reward_earned', 'reward', 'difficulty', 'duration']].reset_index(level=0), on='person_id')
```

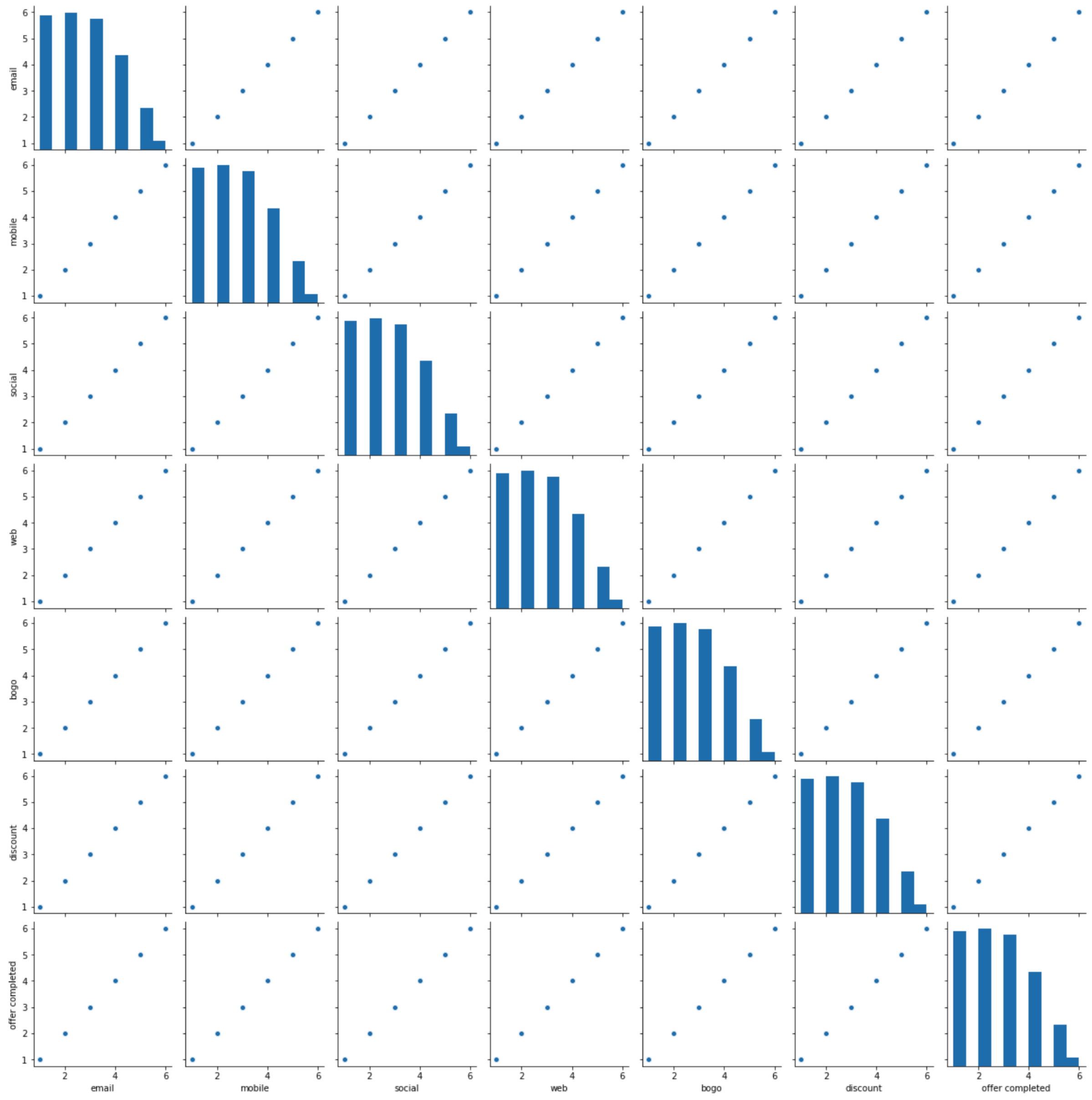
I see that rewards_earned & rewards are same so deleting rewards_earned.



reward earned vs reward given

Similarly number of email, mobile, social, web, bogo, discount are matching with offer completed. So having these features in train will lead to overfitting so removing email, mobile, social, web, bogo, discount.

```
: axes = sns.pairplot(transcript_conv[['email', 'mobile', 'social', 'web', 'bogo', 'discount', 'offer completed']])
```



Merge offer details and transcript details

```
final = pd.merge(offer_details, transcript_conv, on='person_id', how='left')
```

For those who have not completed the offer, fill with zero.

```
final['offer completed'] = final['offer completed'].fillna(0)
```

Now calculate output feature - offer success - 0 means user have not completed the offer, 1 means user completed the offer

```
final['offer_success'] = final['offer completed'].apply(lambda x: 1 if x>0 else 0)
```

Finally merge above final data with profile data

```
# Merging profile data to transcript_conv
final = pd.merge(final, profile_conv, on='person_id', how='left')
final = final.fillna(0)
# drop person_id & offer completed
final.drop(['person_id', 'offer completed'], axis=1, inplace=True)
```

The final data after processing all data, it has 16928 records with 12 features and 1 output feature

1. offer_received - number of offers received
2. offer_viewed - number of offers viewed
3. amount - Amount spent till now
4. no_of_trans - Number of transactions made
5. time_spent_in_hrs - Time spent in hours in test
6. reward - Total rewards received
7. difficulty - Total number of amount spend to complete the offer
8. duration - Total number of days offer was open
9. gender - Customer Gender
10. age - Customer age
11. income - Customer Income
12. tenure - Number of days user is with Starbucks<br>
    Output feature -
13. offer_success - User successfully completed the offer or not

3.2 Split data

Separating offer success data and splitting data 70% for training and 30% for testing.

```
X = final.drop('offer_success', axis=1)
y = pd.DataFrame(final['offer_success'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Finding success rate distribution in both train and test

```
# train success percent - 75%
y_train[y_train['offer_success'] == 1.0].count()/(y_train[y_train['offer_success'] == 0.0].count()\
                                                  +y_train[y_train['offer_success'] == 1.0].count())
```

```
offer_success    0.756435
dtype: float64
```

```
# test success percent - 75%
y_test[y_test['offer_success'] == 1.0].count()/(y_test[y_test['offer_success'] == 0.0].count()\
                                                +y_test[y_test['offer_success'] == 1.0].count())
```

```
offer_success    0.750345
dtype: float64
```

Above shows in both train and test, success rate distribution is 75%

4. Results

4.1 Model Evaluation and Validation

4.1.1 Training model

Using Naive Bayes

```
# Instantiate our model
naive_bayes = MultinomialNB()

# Fit our model to the training data
naive_bayes.fit(X_train, y_train)
```

Using sklearn ensemble

```
# Instantiate a BaggingClassifier with:
# 200 weak learners (n_estimators) and everything else a default values
bagging_mod = BaggingClassifier(n_estimators=200)

# Instantiate a RandomForestClassifier with:
# 200 weak learners (n_estimators) and everything else a default values
random_mod = RandomForestClassifier(n_estimators=200)

# Instantiate an a AdaBoostClassifier with:
# With 300 weak learners (n_estimators) and a learning_rate of 0.2
adaboost_mod = AdaBoostClassifier(n_estimators=300, learning_rate=0.2)

# Fit your BaggingClassifier to the training data
bagging_mod.fit(X_train, y_train)
random_mod.fit(X_train, y_train)
adaboost_mod.fit(X_train, y_train)
```

4.1.2 Testing model

Below is the method which calculates accuracy and f1 score

```
def print_metrics(y_true, preds, model_name=None):
    '''
    INPUT:
    y_true - the y values that are actually true in the dataset (NumPy array or pandas series)
    preds - the predictions for those values from some model (NumPy array or pandas series)
    model_name - (str - optional) a name associated with the model if you would like to add it to the print statements
    OUTPUT:
    None - prints the accuracy, precision, recall, and F1 score
    '''
    if model_name == None:
        print('Accuracy score: ', format(accuracy_score(y_true, preds)))
        print('F1 score: ', format(f1_score(y_true, preds)))
        print('\n\n')

    else:
        print('Accuracy score for ' + model_name + ' :' , format(accuracy_score(y_true, preds)))
        print('F1 score ' + model_name + ' :', format(f1_score(y_true, preds)))
        print('\n\n')
```

Predicting using Naive Bayes & Ensemble methods - bagging, random forest and adaboost.

```
# Predict using Naive bayes on the test data
predictions = naive_bayes.predict(X_test)

# Predict using BaggingClassifier on the test data
bag_preds = bagging_mod.predict(X_test)

# Predict using RandomForestClassifier on the test data
rf_preds = random_mod.predict(X_test)

# Predict using AdaBoostClassifier on the test data
ada_preds = adaboost_mod.predict(X_test)

# Print Bagging scores
print_metrics(y_test, bag_preds, 'bagging')

# Pring Random Forest scores
print_metrics(y_test, rf_preds, 'random_forest')

# Print AdaBosst scores
print_metrics(y_test, ada_preds, 'adaboost')

# Print Bayes Classifier scores
print_metrics(y_test, predictions, 'naive_bayes')
```

Naive Bayes gave 94% accuracy and F1 score as 96%

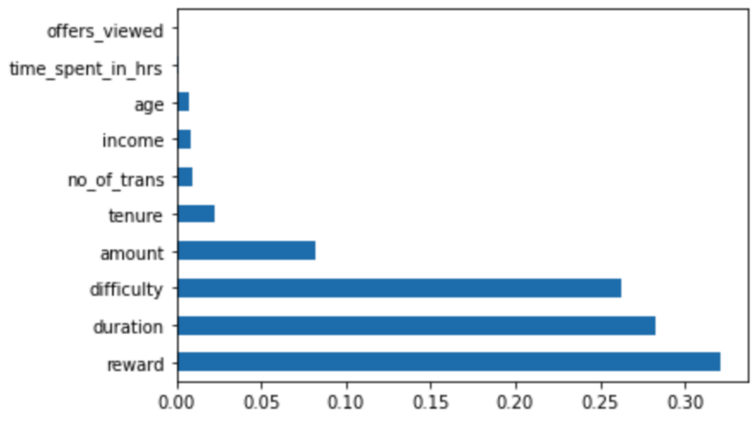
```
Accuracy score for naive_bayes : 0.9265603465249065
F1 score naive_bayes : 0.951602439340859
```

Where as ensemble methods - bagging, random forest & adaboost gave 100% on accuracy & F1 score.

```
Accuracy score for bagging : 1.0
F1 score bagging : 1.0

Accuracy score for random_forest : 1.0
F1 score random_forest : 1.0

Accuracy score for adaboost : 1.0
F1 score adaboost : 1.0
```

4.1.3 Feature importance using Random Forest

```
feature_importances = pd.DataFrame(random_mod.feature_importances_,
                                   index = X_train.columns,
                                    columns=['importance']).sort_values('importance',ascending=False)

(pd.Series(random_mod.feature_importances_, index=X_train.columns)
   .nlargest(10)
   .plot(kind='barh'))
```



4.2 Refinement

Tuning Naive Bayes using GridSearch. As ensemble methods matched benchmark model, no refinement needed.

Naive Bayes available hyper parameters - naive_bayes.get_params()

```
{'alpha': 1.0, 'class_prior': None, 'fit_prior': True}
```

Using GridSearchCV to find hyper-parameters to tune Naive Bayes model

```
grid_params = {
  'alpha': np.linspace(0.1, 1.5, 6),
  'fit_prior': [True, False],
}
clf = GridSearchCV(naive_bayes, grid_params)
clf.fit(X_train, y_train)
print("Best Score: ", clf.best_score_)
print("Best Params: ", clf.best_params_)
```

Output -

```
Best Score:  0.9436234199478852
Best Params:  {'alpha': 0.1, 'fit_prior': True}
```

After tuning with alpha parameter to 0.1, accuracy increased from 92% to 94%

4.4 Testing Robustness of models

```
# Testing robustness
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
for seed in range(5, 50, 5):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=seed)

    naive_bayes = MultinomialNB(alpha=0.1)
    bagging_mod = BaggingClassifier()
    random_mod = RandomForestClassifier()
    adaboost_mod = AdaBoostClassifier()

    naive_bayes.fit(X_train, y_train)
    bagging_mod.fit(X_train, y_train)
    random_mod.fit(X_train, y_train)
    adaboost_mod.fit(X_train, y_train)

    predictions = naive_bayes.predict(X_test)
    bag_preds = bagging_mod.predict(X_test)
    rf_preds = random_mod.predict(X_test)
    ada_preds = adaboost_mod.predict(X_test)

    print_metrics(y_test, bag_preds, 'bagging with seed {}'.format(seed))
    print_metrics(y_test, rf_preds, 'random_forest with seed {}'.format(seed))
    print_metrics(y_test, ada_preds, 'adaboost with seed {}'.format(seed))
    print_metrics(y_test, predictions, 'naive_bayes with seed {}'.format(seed))
```

Output -

```
Accuracy score for bagging with seed 5 : 1.0
F1 score bagging with seed 5 : 1.0

Accuracy score for random_forest with seed 5 : 1.0
F1 score random_forest with seed 5 : 1.0

Accuracy score for adaboost with seed 5 : 1.0
F1 score adaboost with seed 5 : 1.0

Accuracy score for naive_bayes with seed 5 : 0.9523528253593228
F1 score naive_bayes with seed 5 : 0.9688063934003609

Accuracy score for bagging with seed 10 : 1.0
F1 score bagging with seed 10 : 1.0

Accuracy score for random_forest with seed 10 : 1.0
F1 score random_forest with seed 10 : 1.0

Accuracy score for adaboost with seed 10 : 1.0
F1 score adaboost with seed 10 : 1.0

Accuracy score for naive_bayes with seed 10 : 0.9470368182713133
F1 score naive_bayes with seed 10 : 0.9654818426793276

Accuracy score for bagging with seed 15 : 1.0
F1 score bagging with seed 15 : 1.0

Accuracy score for random_forest with seed 15 : 1.0
F1 score random_forest with seed 15 : 1.0

Accuracy score for adaboost with seed 15 : 1.0
F1 score adaboost with seed 15 : 1.0

Accuracy score for naive_bayes with seed 15 : 0.9411301437290805
F1 score naive_bayes with seed 15 : 0.9616912235746317

Accuracy score for bagging with seed 20 : 1.0
F1 score bagging with seed 20 : 1.0

Accuracy score for random_forest with seed 20 : 1.0
F1 score random_forest with seed 20 : 1.0
```

```
Accuracy score for adaboost with seed 20 : 1.0
F1 score adaboost with seed 20 : 1.0

Accuracy score for naive_bayes with seed 20 : 0.9454617050600512
F1 score naive_bayes with seed 20 : 0.9643729903536977

Accuracy score for bagging with seed 25 : 1.0
F1 score bagging with seed 25 : 1.0

Accuracy score for random_forest with seed 25 : 1.0
F1 score random_forest with seed 25 : 1.0

Accuracy score for adaboost with seed 25 : 1.0
F1 score adaboost with seed 25 : 1.0

Accuracy score for naive_bayes with seed 25 : 0.9448710376058279
F1 score naive_bayes with seed 25 : 0.9642126789366053

Accuracy score for bagging with seed 30 : 1.0
F1 score bagging with seed 30 : 1.0

Accuracy score for random_forest with seed 30 : 1.0
F1 score random_forest with seed 30 : 1.0

Accuracy score for adaboost with seed 30 : 1.0
F1 score adaboost with seed 30 : 1.0

Accuracy score for naive_bayes with seed 30 : 0.9495963772396141
F1 score naive_bayes with seed 30 : 0.9671626475115443

Accuracy score for bagging with seed 35 : 1.0
F1 score bagging with seed 35 : 1.0

Accuracy score for random_forest with seed 35 : 1.0
F1 score random_forest with seed 35 : 1.0

Accuracy score for adaboost with seed 35 : 1.0
F1 score adaboost with seed 35 : 1.0

Accuracy score for naive_bayes with seed 35 : 0.9405394762748572
F1 score naive_bayes with seed 35 : 0.9610221992772328

Accuracy score for bagging with seed 40 : 1.0
F1 score bagging with seed 40 : 1.0

Accuracy score for random_forest with seed 40 : 1.0
F1 score random_forest with seed 40 : 1.0

Accuracy score for adaboost with seed 40 : 1.0
F1 score adaboost with seed 40 : 1.0

Accuracy score for naive_bayes with seed 40 : 0.939948808820634
F1 score naive_bayes with seed 40 : 0.9607211848036059

Accuracy score for bagging with seed 45 : 1.0
F1 score bagging with seed 45 : 1.0

Accuracy score for random_forest with seed 45 : 1.0
F1 score random_forest with seed 45 : 1.0

Accuracy score for adaboost with seed 45 : 1.0
F1 score adaboost with seed 45 : 1.0

Accuracy score for naive_bayes with seed 45 : 0.9419177003347116
F1 score naive_bayes with seed 45 : 0.9622230759380203
```

4.4 Justification

Given user demographic data like gender, age etc and promotion data like duration or reward, logistic regression model gave 100% as benchmark model.

When quickly checked with Naive Bayes, which initially gave 92% as accuracy but later on doing some refinement, accuracy came to 94%.

When used sklearn ensemble methods like Ada Boost, Bagging and Random Forest was able to predict 100% of the time. The model seems to perform well with different random seeds consistently.

By using feature importance, we can say that, below features played important role in offer to get complete

- reward - how much reward was given on promotion.
- duration - Total number of days offer was open.
- difficulty - minimum required spend to complete an offer.
- amount - How much user spent till now.
- tenure - How long user is with Starbucks.

5. References

1. https://scikit-learn.org/stable/modules/ensemble.html
2. https://scikit-learn.org/stable/modules/naive_bayes.html
3. https://developers.google.com/machine-learning/crash-course/logistic-regression/calculating-a-probability
4. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html
6. https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/
7. https://towardsdatascience.com/all-about-naive-bayes-8e13cef044cf