Starbucks Rewards Recommender

Introduction

As a coffee lover, I always check out the Starbucks mobile app for offers and discounts for my next coffee purchase. I was greatly motivated to see datasets containing transaction, demographic information and offers that mimics customer behavior on Starbucks rewards mobile app. I decided to take this as a challenge for my Machine Learning Nanodegree program, explore the datasets, build a supervised model and deploy the same to predict influencing offers for Starbucks coffee lovers.

Domain background

A recommendation system falls under the broad umbrella of information filtering system. The primary goal or application of a such systems is to suggest an attractive list of product recommendations for commercial purposes. A company can use these suggested recommendations and advertise its products to targeted audience. What makes this problem challenging is the plethora of industries that require such recommendation services. Each domain has its own set of consumers and product types to experiment with. On a positive note, there is one common question that needs an answer across the board for any industry i.e. How to attract new customers and keep existing customers to buy products and services? In this project, I would like to explore this question from the perspective of a famous Coffee company, “Starbucks”.

Problem statement

We will answer the query of how to make Starbucks customers happy and keep them interested in purchasing its products with the help of the rewards app that the coffee franchise offers to its customers. The goal is to make the rewards app provide different offers that interests its customers and keeps them buying more coffee and related products from Starbucks. One thing to remember is that the app can not only recommend offers to existing customers but also draw new customers based on the machine learning model which can recommends offers to new customers who have similar demographic attributes as existing customers.

The specific question that I wanted to answer in this project is,

What specific offers should we recommend to customers with certain demographic attributes. Meaning, a very specific query we might want to answer could be,

What offers will attract customers < 40 years of age and yearly income > 70K?

I will explore at least two or more queries like this in this project.

I also expect the model to predict one or more offers we want to recommend our customers with a given set of attributes. Also, before training the model using the provided datasets, I would want to know if a previously recommended offer actually influenced the customers on their next purchase. We will see later how an indicator attribute, something like influencing_offer can be useful for our analysis.

Datasets and Inputs

The datasets for this project are contained in three files. These files were provided as part of the Starbucks project workspace on Udacity's website. I have referenced a short video in the reference section showing the motivation of this capstone project as shared by a Data Scientist at Starbucks.

The data names, shapes, schema and explanation of each variable in these files are below:

1. **portfolio.json** - containing offer ids and meta data about each offer (duration, type, etc.)

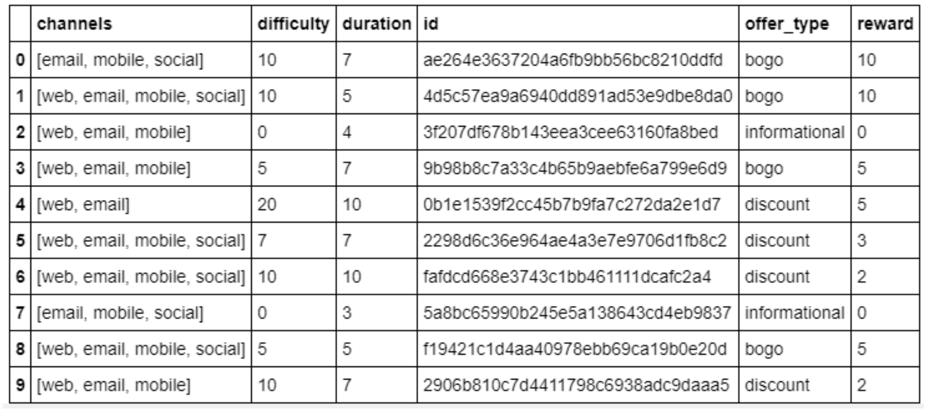
Columns:
id (string) - offer id
offer_type (string) - type of offer i.e. BOGO, discount, informational
difficulty (int) - minimum required spend to complete an offer
reward (int) - reward given for completing an offer
duration (int) - time for offer to be open, in days
channels - List of strings

Numeric – difficulty, reward, duration

Text – id, offer_type, channels

Rows: 10

As mentioned, the portfolio data has details of offer metadata. The tabular representation of this data is shown next. This is just metadata so we need not worry about balanced and unbalanced nature of this table.



	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

2. **profile.json** - demographic data for each customer

Columns:

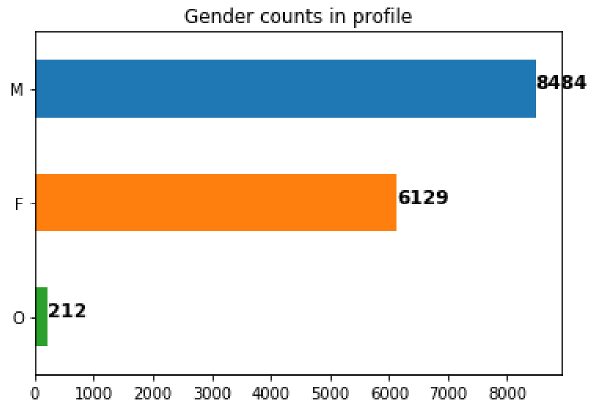
age (int) - age of the customer
became_member_on (int) - date when customer created an app account
gender (str) - gender of the customer (note some entries contain 'O' for other rather than M or F)
id (str) - customer id
income (float) - customer's income

Numeric – *age, id and income*

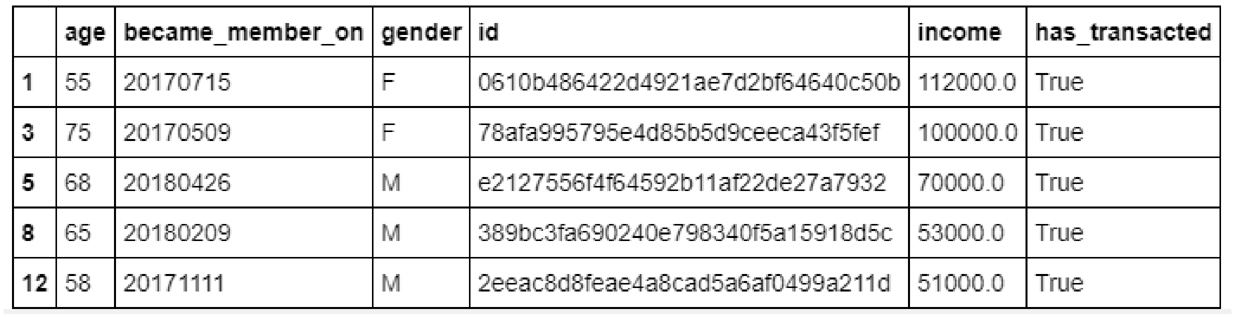
Text – *gender*

Rows: 17,000

The dataset is little imbalanced as there are more males and females than other gender types. We could either sample this data to work on subset cases targeting specific gender types and also augment the dataset with some more cases for the other gender category so as to balance for few cases with the other gender category.



The first few rows of the profile dataset are shown below,



	age	became_member_on	gender	id	income	has_transacted
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0	True
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0	True
5	68	20180426	M	e2127556f4f64592b11af22de27a7932	70000.0	True
8	65	20180209	M	389bc3fa690240e798340f5a15918d5c	53000.0	True
12	58	20171111	M	2eeac8d8feae4a8cad5a6af0499a211d	51000.0	True

3. **transcript.json** - records for transactions, offers received, offers viewed, and offers completed

Columns:
event (string) - record description (i.e. transaction, offer received, offer viewed, etc.)
person (string) - 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

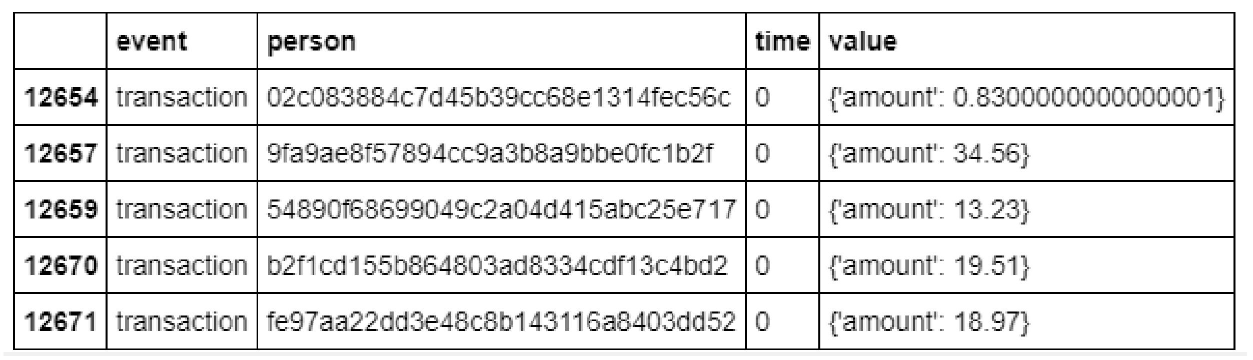
Numeric – *time (this is time in hours since test started)*

Text – event, person, value

Rows: 306,534

The transactions data is balanced as such as it does not involve any gender columns like in the profile data above. But we surely want to normalize the transactions if we want to query and find any influencing offers based on gender with particular age and annual income range.

The first few rows of the transcript dataset are shown below,



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

Project design

The solution was designed on Jupyter notebooks that reads the previously mentioned JSON datasets into Pandas Dataframes, transforms the same to create a normalized data table of customer profiles based on the transactions seen so far on the Starbucks mobile rewards app.

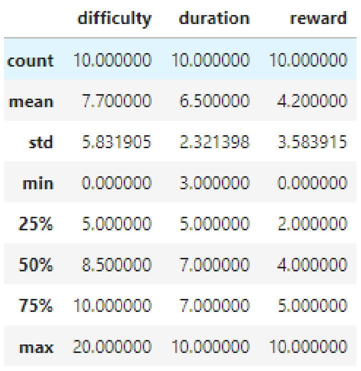
In this project, I have followed Data mining and Machine learning approaches to explore, clean and transform the provided datasets and then build out a baseline and challenger models and finally interpreted the results of these models based on metrics like Confusion matrix, AUC and F1 scores.

Data Exploration

A best way to look at the details of the features in various datasets is by using the describe function available in the Pandas dataframes.

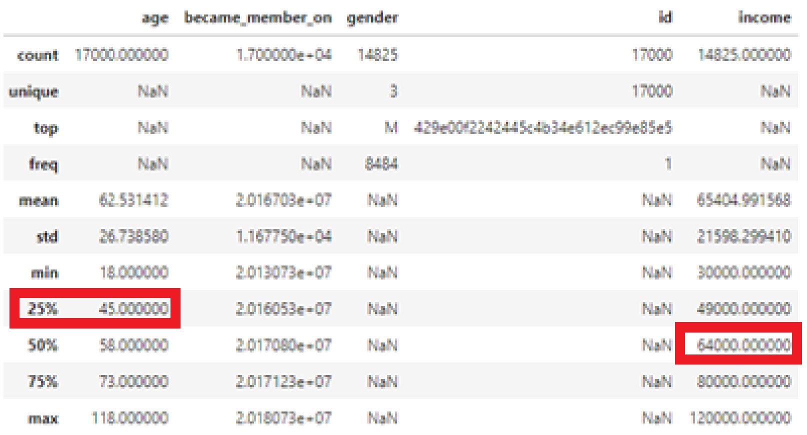
1. The data description of each datasets are shown below,

   a. Portfolio



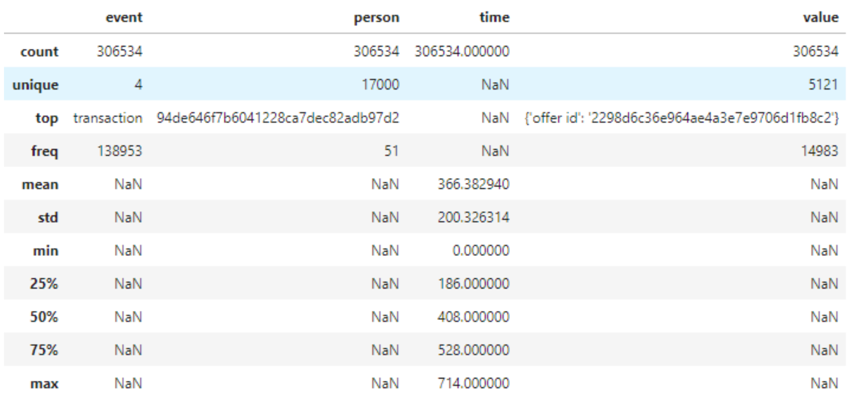
	difficulty	duration	reward
count	10.000000	10.000000	10.000000
mean	7.700000	6.500000	4.200000
std	5.831905	2.321398	3.583915
min	0.000000	3.000000	0.000000
25%	5.000000	5.000000	2.000000
50%	8.500000	7.000000	4.000000
75%	10.000000	7.000000	5.000000
max	20.000000	10.000000	10.000000

   b. Profile



	age	became_member_on	gender	id	income
count	17000.000000	1.700000e+04	14825	17000	14825.000000
unique	NaN	NaN	3	17000	NaN
top	NaN	NaN	M	429e00f2242445c4b34e612ec99e85e5	NaN
freq	NaN	NaN	8484	1	NaN
mean	62.531412	2.016703e+07	NaN	NaN	65404.991568
std	26.738580	1.167750e+04	NaN	NaN	21598.299410
min	18.000000	2.013073e+07	NaN	NaN	30000.000000
25%	45.000000	2.016053e+07	NaN	NaN	49000.000000
50%	58.000000	2.017080e+07	NaN	NaN	64000.000000
75%	73.000000	2.017123e+07	NaN	NaN	80000.000000
max	118.000000	2.018073e+07	NaN	NaN	120000.000000

   c. Transcript

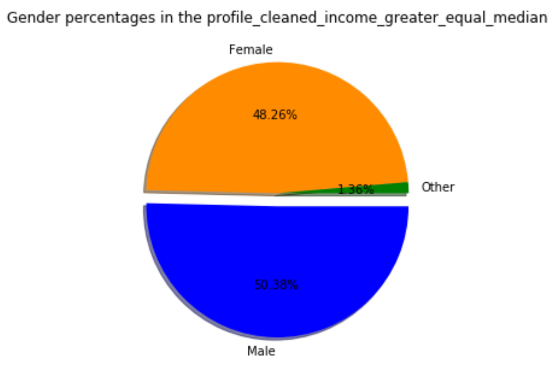


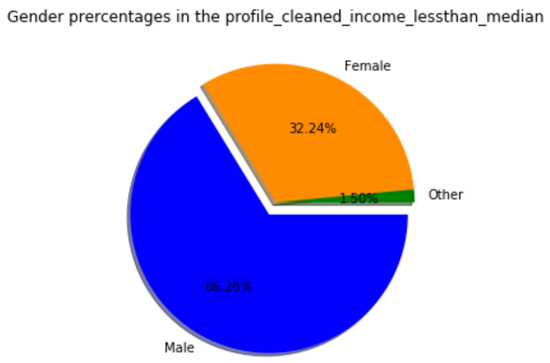
	event	person	time	value
count	306534	306534	306534.000000	306534
unique	4	17000	NaN	5121
top	transaction	94de646f7b6041228ca7dec82adb97d2	NaN	{'offer id': '2298d6c36e964ae4a3e7e9706d1fb8c2'}
freq	138953	51	NaN	14983
mean	NaN	NaN	366.382940	NaN
std	NaN	NaN	200.326314	NaN
min	NaN	NaN	0.000000	NaN
25%	NaN	NaN	186.000000	NaN
50%	NaN	NaN	408.000000	NaN
75%	NaN	NaN	528.000000	NaN
max	NaN	NaN	714.000000	NaN

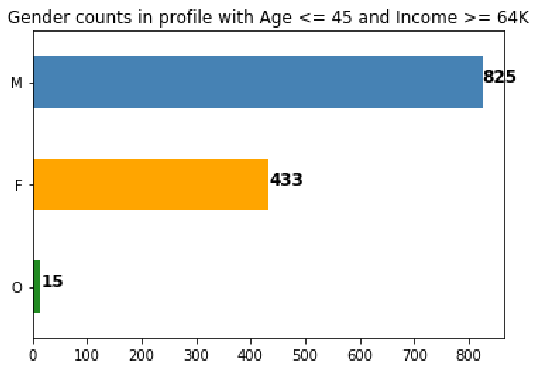
As we can see from the above Profile table description. The median income of the customers in the profile dataset is 64K. Also, the first Inter Quartile range of age is 45. We will use these values to query the datasets and explore some more details from it to prepare clean and transform the profile data into a shape we can then use to analyze.

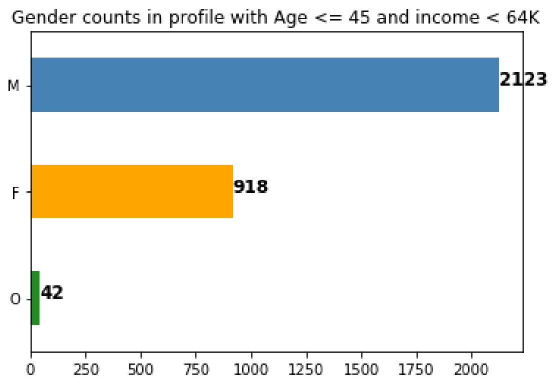
Descriptive data analysis

I first performed some simple descriptive analysis on gender counts and percentages based on the Q1 age limit of 45 and median income of 65K. Pictures showing Pi charts and bar graphs of these analysis are shown next.







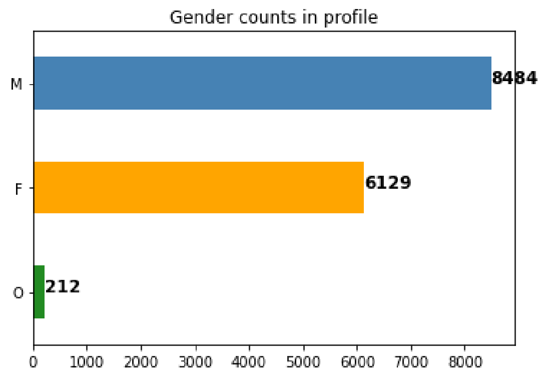


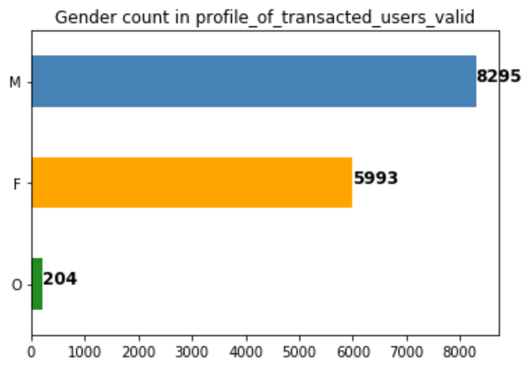
Data Preparation and Cleaning

I removed invalid values from the profile data and further dropped rows that made less sense for the analysis. For example, there were rows of customers with age 118. These customers were probably not alive or maybe invalid value entered for age. We can exclude such rows from our analysis.

Using the transcript dataset, I had to find the users who had one or more transactions and others who haven't done any transactions at all. Based on this information we can further subset our profile dataset into two categories i.e. profiles who are actively transacting using the app and others who aren't. We will train a model using the transacted users and can later deploy it on users who have not transacted to predict offers for them.

The picture below shows a breakdown of valid users i.e. profiles after dropping NA values and ages with values 118.





1. Males in original Profile data = 8484; Males in profile_of_transacted_users_valid data = **8295**
    a. Proportion of "Males" from valid records who have not transacted = 1 - (8295/8484) = 1 - 0.9777 = **0.022 (Approx. 2.2%)**
2. Females in original Profile data = 6129; Females in profile_of_transacted_users_valid data = **5993**
    a. Proportion of "Females" from valid records who have not transacted = 1 - (5993/6129) = 1 - 0.9778 = **0.022 (Approx. 2.2%)**
3. Others in original Profile data = 212; Others in profile_of_transacted_users_valid data = **204**
    a. Proportion of "Others" from valid records who have not transacted = 1 - (204/212) = 1 - 0.9622 = **0.0377 (Approx. 3.8%)**

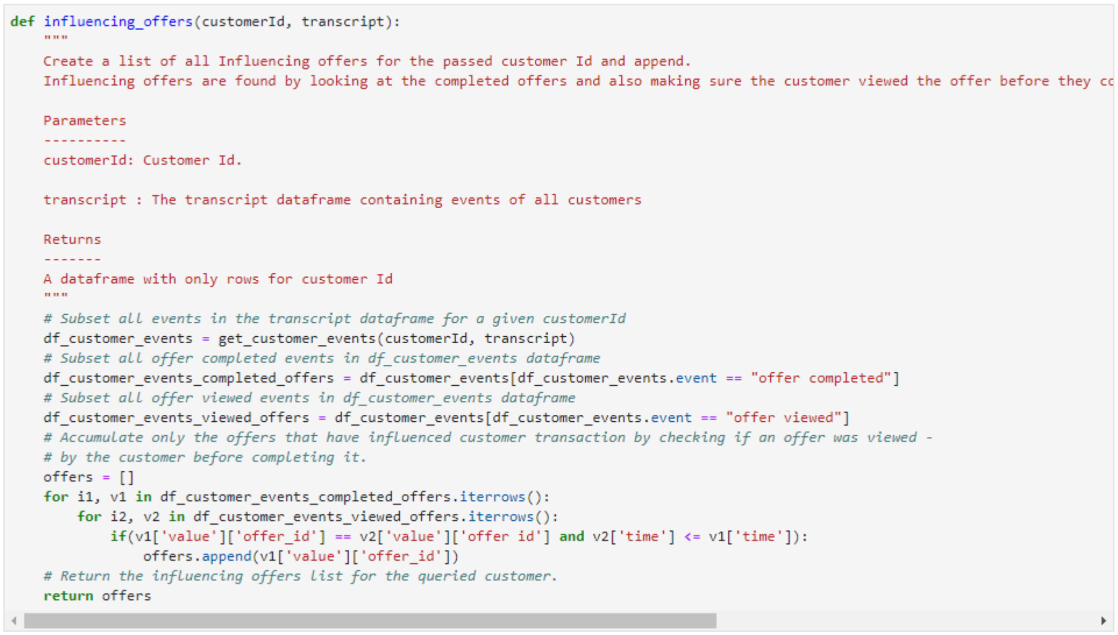
Data Transformation

After cleaning and preparing profile data of just the transacted users we will then create a column of influential offers for all transacted users. For this I have written a function called influencing_offers that takes in the transcript dataset and works on it for each user.

Influencing offers are found by looking at the completed offers and also making sure the customer viewed the offer before they completed the offer based on the key offer_id in the transcript dataframe.

Note: A tricky part here was that in the "value" column of transcript dataframe, the dictionary keys were "offer id" i.e. with a space in the middle, for offer received and offfer viewed entries, but was "offer_id" i.e. with an underscore in the middle, for offer completed entries. So, we have to be mindful of these keys.

The function is below,



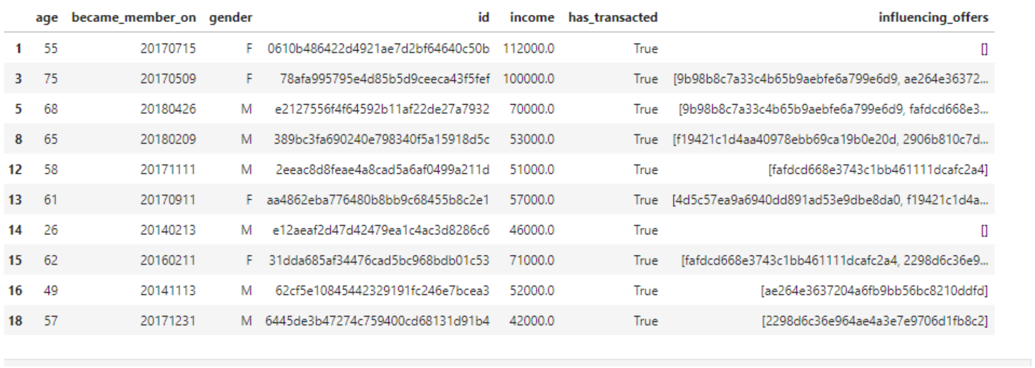
```python
def influencing_offers(customerId, transcript):
    """
    Create a list of all Influencing offers for the passed customer Id and append.
    Influencing offers are found by looking at the completed offers and also making sure the customer viewed the offer before they c

    Parameters
    ----------
    customerId: Customer Id.

    transcript : The transcript dataframe containing events of all customers

    Returns
    -------
    A dataframe with only rows for customer Id
    """
    # Subset all events in the transcript dataframe for a given customerId
    df_customer_events = get_customer_events(customerId, transcript)
    # Subset all offer completed events in df_customer_events dataframe
    df_customer_events_completed_offers = df_customer_events[df_customer_events.event == "offer completed"]
    # Subset all offer viewed events in df_customer_events dataframe
    df_customer_events_viewed_offers = df_customer_events[df_customer_events.event == "offer viewed"]
    # Accumulate only the offers that have influenced customer transaction by checking if an offer was viewed -
    # by the customer before completing it.
    offers = []
    for i1, v1 in df_customer_events_completed_offers.iterrows():
        for i2, v2 in df_customer_events_viewed_offers.iterrows():
            if(v1['value']['offer_id'] == v2['value']['offer id'] and v2['time'] <= v1['time']):
                offers.append(v1['value']['offer_id'])
    # Return the influencing offers list for the queried customer.
    return offers
```

1. Add influencing_offers column for each transacted user in profile dataset,

   Using the above function we transform the profile dataframe of transacted users by adding a new column called influencing_offers to this dataframe. Meaning, we are building a column called "influencing_offers" for each user by applying a lambda function that deduces the influencing offers by looking at the list of all events in the transcript dataframe for each customer and returning a list of influencing offers.
   Note: For some users there might not be an influencing offer at all. As mentioned before an influencing offer is one where the customer viewed the offer before completing it. The Python call looks like below,

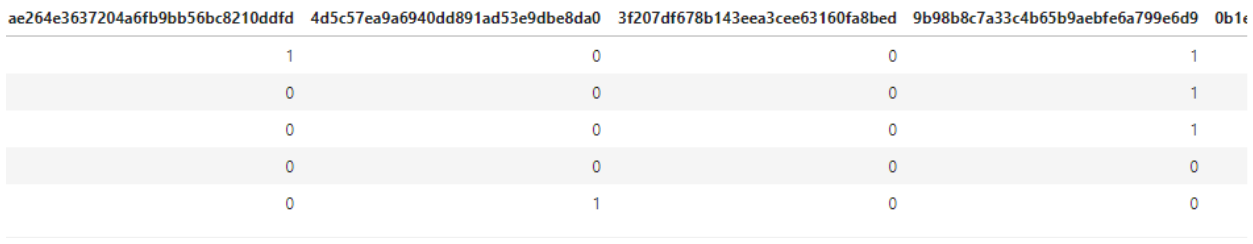
   profile_of_transacted_users_valid["influencing_offers"] = profile_of_transacted_users_valid.apply(lambda x:influencing_offers(x.id, transcript),axis=1)

   A screenshot of profile of transacted dataframe with the included influential_offer column is below,



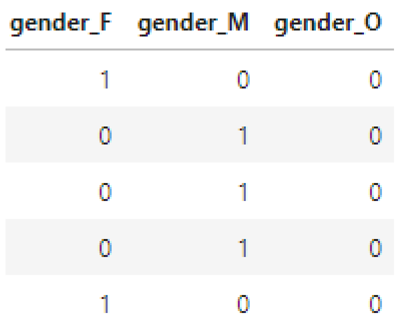
	age	became_member_on	gender	id	income	has_transacted	influencing_offers
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0	True	[]
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0	True	[9b98b8c7a33c4b65b9aebfe6a799e6d9, ae264e36372...
5	68	20180426	M	e2127556f4f64592b11af22de27a7932	70000.0	True	[9b98b8c7a33c4b65b9aebfe6a799e6d9, fafdcd668e3...
8	65	20180209	M	389bc3fa690240e798340f5a15918d5c	53000.0	True	[f19421c1d4aa40978ebb69ca19b0e20d, 2906b810c7d...
12	58	20171111	M	2eeac8d8feae4a8cad5a6af0499a211d	51000.0	True	[fafdcd668e3743c1bb461111dcafc2a4]
13	61	20170911	F	aa4862eba776480b8bb9c68455b8c2e1	57000.0	True	[4d5c57ea9a6940dd891ad53e9dbe8da0, f19421c1d4a...
14	26	20140213	M	e12aeaf2d47d42479ea1c4ac3d8286c6	46000.0	True	[]
15	62	20160211	F	31dda685af34476cad5bc968bdb01c53	71000.0	True	[fafdcd668e3743c1bb461111dcafc2a4, 2298d6c36e9...
16	49	20141113	M	62cf5e10845442329191fc246e7bcea3	52000.0	True	[ae264e3637204a6fb9bb56bc8210ddfd]
18	57	20171231	M	6445de3b47274c759400cd68131d91b4	42000.0	True	[2298d6c36e964ae4a3e7e9706d1fb8c2]

2. One-Hot encode the influencing offers column into columns of all offer ids with a 1 if the customer was influenced by the offer and a 0 if they were not influenced by the offer**.**



ae264e3637204a6fb9bb56bc8210ddfd	4d5c57ea9a6940dd891ad53e9dbe8da0	3f207df678b143eea3cee63160fa8bed	9b98b8c7a33c4b65b9aebfe6a799e6d9	0b1
1	0	0	1	
0	0	0	1	
0	0	0	1	
0	0	0	0	
0	1	0	0	

3. One-Hot encode gender column



gender_F	gender_M	gender_O
1	0	0
0	1	0
0	1	0
0	1	0
1	0	0

Model Building and hyperparameter tuning

Baseline models

As a first baseline model, I built a simple logistic Regression classifier with the profile dataset that was prepared as explained in the Data Transformation section above. I had a train-test split of 75%-25% for this model. This model was performing very poorly and was only predicting 0's for offer types. The logistic regression fit was learning mostly the 0's from this dataset as there is slight imbalance in the number of offers that influenced users when compared to number of offers that did not influence users.

I then went ahead and built out a basic Random Forest tree model. This model performed better than the logistic regression model. We still want to improve the overall accuracy of our basic Random Forest model by tuning the hyperparameters like,

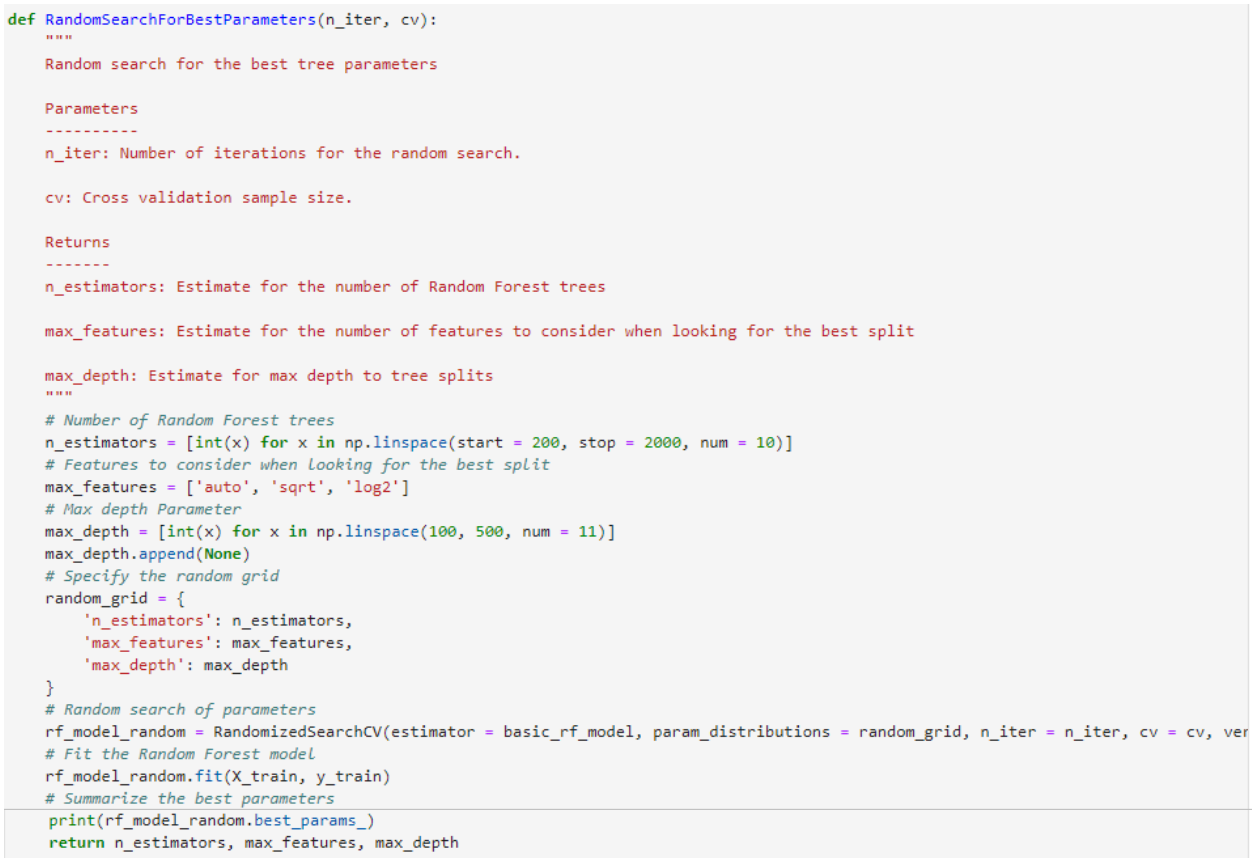
1. Number of Random Forest trees

2. The number of features to consider when looking for the best split

3. Depth of the built trees

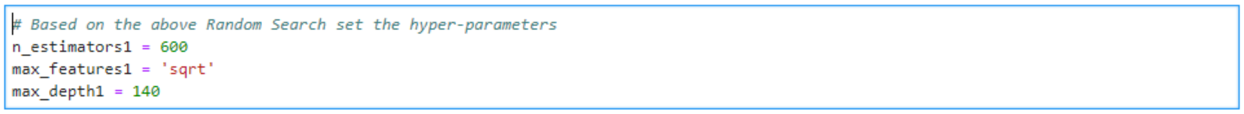
Tuned Model

I wrote a function RandomSearchForBestParameters (shown below) that was handy for tuning the hyperparameters of the base Random Forest model.



```
def RandomSearchForBestParameters(n_iter, cv):
    """
    Random search for the best tree parameters

    Parameters
    ----------
    n_iter: Number of iterations for the random search.

    cv: Cross validation sample size.

    Returns
    -------
    n_estimators: Estimate for the number of Random Forest trees

    max_features: Estimate for the number of features to consider when looking for the best split

    max_depth: Estimate for max depth to tree splits
    """
    # Number of Random Forest trees
    n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
    # Features to consider when looking for the best split
    max_features = ['auto', 'sqrt', 'log2']
    # Max depth Parameter
    max_depth = [int(x) for x in np.linspace(100, 500, num = 11)]
    max_depth.append(None)
    # Specify the random grid
    random_grid = {
        'n_estimators': n_estimators,
        'max_features': max_features,
        'max_depth': max_depth
    }
    # Random search of parameters
    rf_model_random = RandomizedSearchCV(estimator = basic_rf_model, param_distributions = random_grid, n_iter = n_iter, cv = cv, ver
    # Fit the Random Forest model
    rf_model_random.fit(X_train, y_train)
    # Summarize the best parameters
    print(rf_model_random.best_params_)
    return n_estimators, max_features, max_depth
```

After using the above function, I narrowed down the required hyperparameters to the following values,



```
# Based on the above Random Search set the hyper-parameters
n_estimators1 = 600
max_features1 = 'sqrt'
max_depth1 = 140
```

Finally, the tuned model with better accuracy i.e. one with better precision and recall was the Champion Challenger that can be used for deployment.

Results Interpretation

Evaluation metrics

I used the following evaluation metrics for the models.

1. Confusion Matrix — The confusion matrix provides a great place to start model evaluation for classification models. In essence we look at the proportions of predicted and observed classes and build out a matrix showing True Positives, True Negatives, False Positives and False Negatives. From this we can then compute our model's precision and recall. A precise model with a high recall rate is always better to have. Understanding the confusion matrix is a much bigger topic. All I will say here is, we can trust our classification model to recommend an offer with high confidence if it had a high precision and a better recall rate.

2. Area Under the Curve (AOC) — This metric will also fit in well as a performance metric for our classifier. In short, the AUC-ROC curve is a performance measure that tells us how better our model is in distinguishing the different categories or classes that we are exploring using the model. The higher the AUC the better is the model.

3. F1 score — This can be interpreted as the weighted average of the precision and recall. The traditional or balanced F-score (F1 score) is the harmonic mean of precision and recall, where an F1 score reaches its best value at 1 and worst at 0.

I have summarized the results of the Random Forest mode that was built with hyper parameters found using the function we discussed above. Please note the summary below shows the results for one offer id *fafdcd668e3743c1bb461111dcafc2a4 : discount*

(6.) Model Summary for offer id - fafdcd668e3743c1bb461111dcafc2a4 : discount

---

```
--------------------------
   Confusion Matrix
--------------------------
[[1915  457]
 [1013  245]]


---------------------------------
   Classification Summary
---------------------------------
              precision    recall  f1-score   support

           0       0.65      0.81      0.72      2372
           1       0.35      0.19      0.25      1258

   micro avg       0.60      0.60      0.60      3630
   macro avg       0.50      0.50      0.49      3630
weighted avg       0.55      0.60      0.56      3630



--------------
```

```
   AUC
--------------
[0.47987411 0.51307665 0.51917315 0.48960705 0.50500194 0.52295414
 0.53691392 0.48218289 0.48763755 0.48422565]


-------------------
   Mean AUC
-------------------
Mean AUC for Random Forest Model:  0.502064702387952
```

Sumary of basic_rf_model for the same discount offer above was,

1. Precision for class 0 i.e. offer did not influence customer is 0.65
2. Precision for class 1 i.e. offer influenced customer is 0.37
3. Recall for class 0 i.e. offer did not influence customer is 0.74
4. Recall for class 1 i.e. offer influenced customer is 0.28
5. AUC is 0.5021
6. From the confusion matrix, out of 616 offers that influenced customers our model seems to be returning only correct predictions for 381 cases. This can be improved.
7. The F1-scores reported were 0.69 for non-influential cases and 0.34 for influential cases for this offer.

Our improved model with the optimized hyper parameters produced the following result,

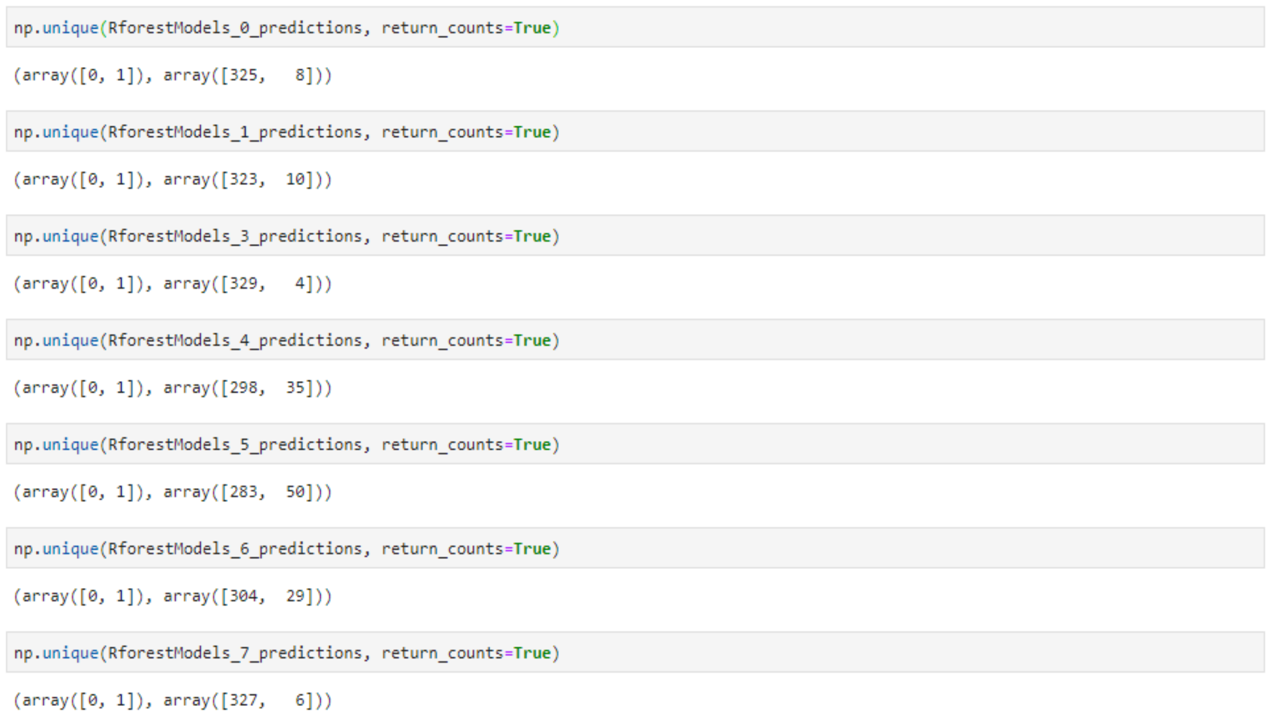
1. Precision for class 0 i.e. offer did not influence customer is 0.65
2. Precision for class 1 i.e. offer influenced customer is 0.37
3. Recall for class 0 i.e. offer did not influence customer is 0.81
4. Recall for class 1 i.e. offer influenced customer is 0.19
5. AUC is 0.4965
6. From the confusion matrix, out of 707 offers that influenced customers our model seems to be returning only correct predictions for 457 cases. There was a slight improvement here.
7. The F1-scores reported were 0.72 for non-influential cases and 0.25 for influential cases for this offer. There was slight improvement predicting the non-influential offers here.

Model Deployment

I then took these build out Random Forest models and deployed them on cases that did not have make any transactions. The results looked like below,

```
RforestModels_0_predictions = RforestModels[0].predict(profile_to_deploy)

RforestModels_1_predictions = RforestModels[1].predict(profile_to_deploy)
```

```
RforestModels_2_predictions = RforestModels[2].predict(profile_to_deploy)

RforestModels_3_predictions = RforestModels[3].predict(profile_to_deploy)

RforestModels_4_predictions = RforestModels[4].predict(profile_to_deploy)

RforestModels_5_predictions = RforestModels[5].predict(profile_to_deploy)

RforestModels_6_predictions = RforestModels[6].predict(profile_to_deploy)

RforestModels_7_predictions = RforestModels[7].predict(profile_to_deploy)
```



```
np.unique(RforestModels_0_predictions, return_counts=True)

(array([0, 1]), array([325,   8]))

np.unique(RforestModels_1_predictions, return_counts=True)

(array([0, 1]), array([323,  10]))

np.unique(RforestModels_3_predictions, return_counts=True)

(array([0, 1]), array([329,   4]))

np.unique(RforestModels_4_predictions, return_counts=True)

(array([0, 1]), array([298,  35]))

np.unique(RforestModels_5_predictions, return_counts=True)

(array([0, 1]), array([283,  50]))

np.unique(RforestModels_6_predictions, return_counts=True)

(array([0, 1]), array([304,  29]))

np.unique(RforestModels_7_predictions, return_counts=True)

(array([0, 1]), array([327,   6]))
```

From the above deployment results, we see that for each offers we are mostly getting 0's indicating no recommended offers and only few 1's for customers to recommend. This can be attributed to the fact that our Random Forest model is still in its early stages and needs more data to improve its precision and recall.

In such cases, we should not completely rely on this one model. It would be better if we can use some hybrid approach of using this model and also performing a targeted offer search based on the demographic information. We will call that as Heuristic approach. This approach is recorded in the targeted_offers.py file here, Heuristic Approach for recommending targeted offers

Heuristic Approach

To address the problem of poor recall rate of influenced customers and to provide a little more robust approach to recommending offers to all customers, we will focus on another simple heuristic approach to target only the subset of user profiles using the demographic information like age, gender and income.

Once we subset these user profiles, we will find the influential offers for these customers like we did before in the previous section. Finally, we will count the influencing offers for all the influenced customers based on each offer type. Once we have a count of each offers, we can return the top two majority votes across each offers.

We now have a heuristic model that can recommend offers to the rest of the samples in the same population who were either not influenced or did not transact at all.

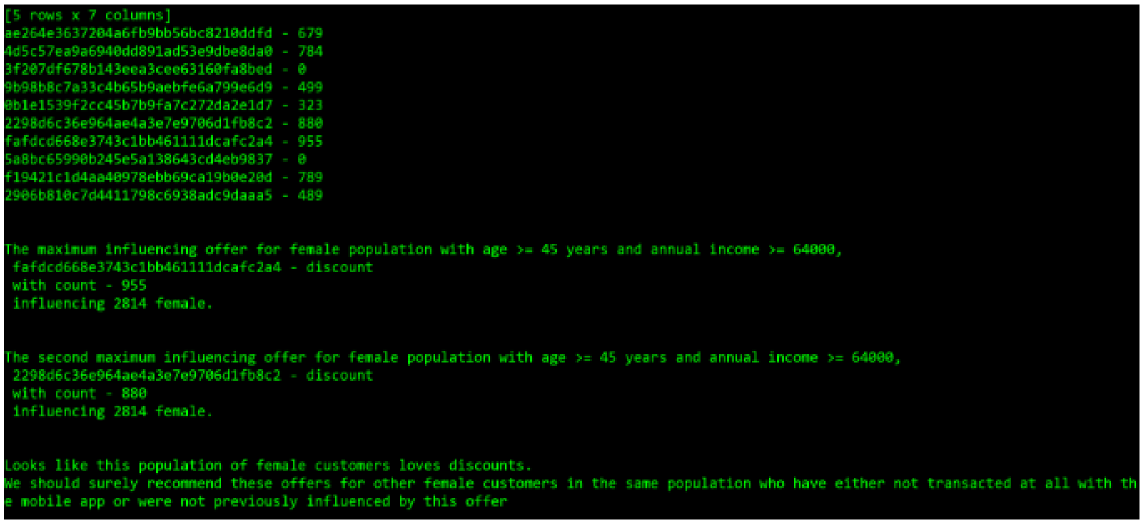
This type of analysis has its own benefits and pitfall,

1. The benefit being we can recommend the offer liked by majority of the sample population to the rest in the same population. That way all the customers with the same demographic attributes get recommended.

2. The pitfall being, these recommendations are targeting only a subset of the population and not the whole customer base of the mobile app.

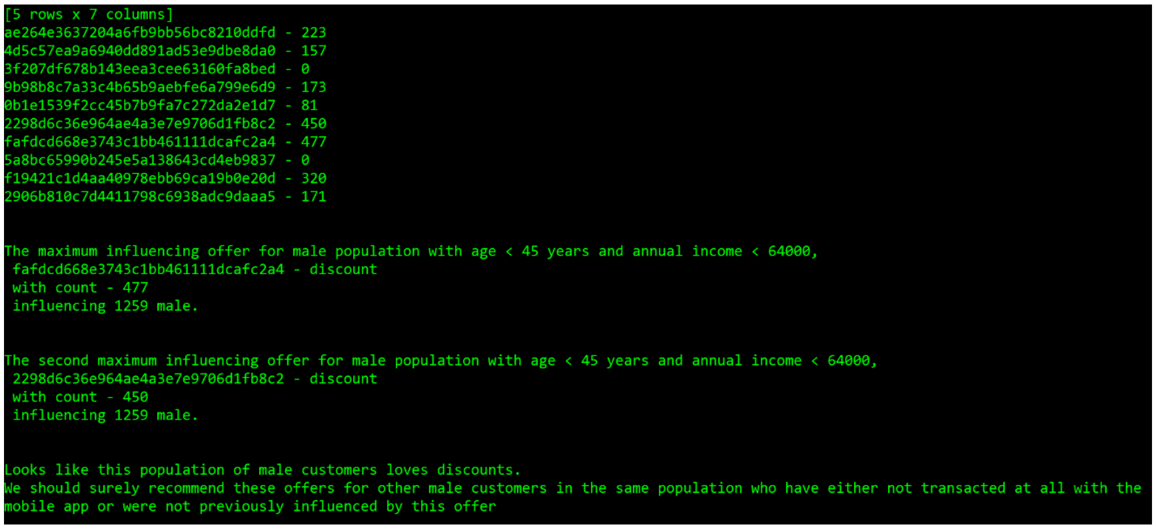
I performed such targeted offers for three subsets of customer population

1. Female customers with age >= 45 years and yearly income >= 64000 USD – *These customers were influenced by discount offers*



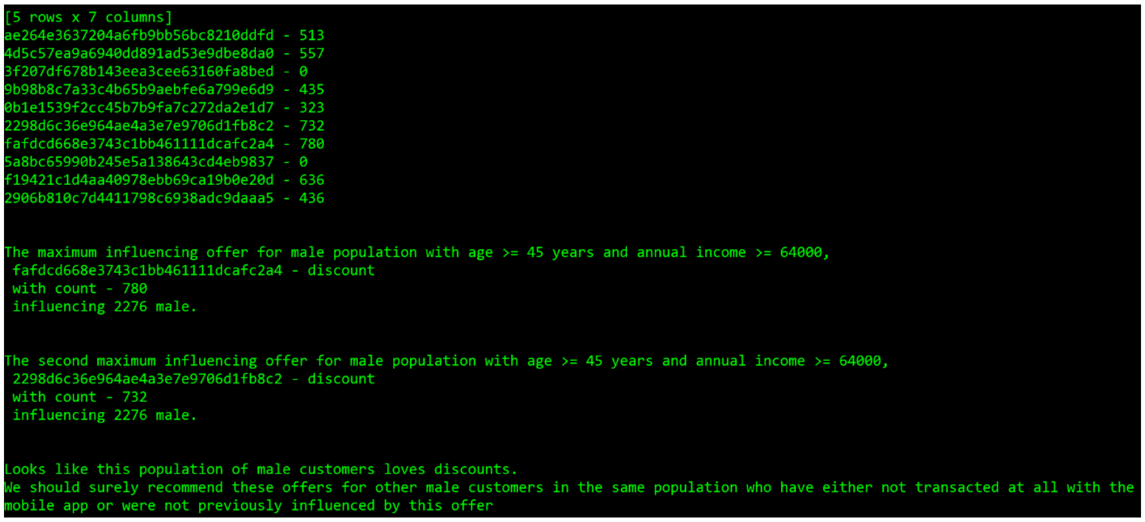
```
[5 rows x 7 columns]
ae264e3637204a6fb9bb56bc8210ddfd - 679
4d5c57ea9a6940dd891ad53e9dbe8da0 - 784
3f207df678b143eea3cee63160fa8bed - 0
9b98b8c7a33c4b65b9aebfe6a799e6d9 - 499
0b1e1539f2cc45b7b9fa7c272da2e1d7 - 323
2298d6c36e964ae4a3e7e9706d1fb8c2 - 880
fafdcd668e3743c1bb461111dcafc2a4 - 955
5a8bc65990b245e5a138643cd4eb9837 - 0
f19421c1d4aa40978ebb69ca19b0e20d - 789
2906b810c7d4411798c6938adc9daaa5 - 489


The maximum influencing offer for female population with age >= 45 years and annual income >= 64000,
 fafdcd668e3743c1bb461111dcafc2a4 - discount
 with count - 955
 influencing 2814 female.


The second maximum influencing offer for female population with age >= 45 years and annual income >= 64000,
 2298d6c36e964ae4a3e7e9706d1fb8c2 - discount
 with count - 880
 influencing 2814 female.


Looks like this population of female customers loves discounts.
We should surely recommend these offers for other female customers in the same population who have either not transacted at all with th
e mobile app or were not previously influenced by this offer
```

2. Male customers with age < 45 years and yearly income >= 64000 USD – *These customers were also influenced by discount offers*



```
[5 rows x 7 columns]
ae264e3637204a6fb9bb56bc8210ddfd - 223
4d5c57ea9a6940dd891ad53e9dbe8da0 - 157
3f207df678b143eea3cee63160fa8bed - 0
9b98b8c7a33c4b65b9aebfe6a799e6d9 - 173
0b1e1539f2cc45b7b9fa7c272da2e1d7 - 81
2298d6c36e964ae4a3e7e9706d1fb8c2 - 450
fafdcd668e3743c1bb461111dcafc2a4 - 477
5a8bc65990b245e5a138643cd4eb9837 - 0
f19421c1d4aa40978ebb69ca19b0e20d - 320
2906b810c7d4411798c6938adc9daaa5 - 171


The maximum influencing offer for male population with age < 45 years and annual income < 64000,
 fafdcd668e3743c1bb461111dcafc2a4 - discount
 with count - 477
 influencing 1259 male.


The second maximum influencing offer for male population with age < 45 years and annual income < 64000,
 2298d6c36e964ae4a3e7e9706d1fb8c2 - discount
 with count - 450
 influencing 1259 male.


Looks like this population of male customers loves discounts.
We should surely recommend these offers for other male customers in the same population who have either not transacted at all with the
mobile app or were not previously influenced by this offer
```

3. Male customers with age >= 45 years and yearly income >= 64000 USD – *These customers were also influenced by discount offers*



```
[5 rows x 7 columns]
ae264e3637204a6fb9bb56bc8210ddfd - 513
4d5c57ea9a6940dd891ad53e9dbe8da0 - 557
3f207df678b143eea3cee63160fa8bed - 0
9b98b8c7a33c4b65b9aebfe6a799e6d9 - 435
0b1e1539f2cc45b7b9fa7c272da2e1d7 - 323
2298d6c36e964ae4a3e7e9706d1fb8c2 - 732
fafdcd668e3743c1bb461111dcafc2a4 - 780
5a8bc65990b245e5a138643cd4eb9837 - 0
f19421c1d4aa40978ebb69ca19b0e20d - 636
2906b810c7d4411798c6938adc9daaa5 - 436


The maximum influencing offer for male population with age >= 45 years and annual income >= 64000,
 fafdcd668e3743c1bb461111dcafc2a4 - discount
 with count - 780
 influencing 2276 male.


The second maximum influencing offer for male population with age >= 45 years and annual income >= 64000,
 2298d6c36e964ae4a3e7e9706d1fb8c2 - discount
 with count - 732
 influencing 2276 male.


Looks like this population of male customers loves discounts.
We should surely recommend these offers for other male customers in the same population who have either not transacted at all with the
mobile app or were not previously influenced by this offer
```

Our heuristic analysis shows that all three targeted population were influenced by the two discount offers with the following ids,

1. fafdcd668e3743c1bb461111dcafc2a4

2. 2298d6c36e964ae4a3e7e9706d1fb8c2

Conclusion

In this project, I wanted to use data mining and machine learning techniques to explore the given datasets, build a predictive analytics model and finally deploy it on test samples to generate recommendations of influencing offers to Starbucks customers. I have also used Amazon Sagemaker notebook instance to train and test base models. Tuned the base model and got slightly better performance than the base models.

Customer recommendation is a difficult process. Given the fact there are many competitions from the outside world, it is tough to stay on top of the business. In this project we looked at two different approaches to solve an offer recommendation problem.

Even though the predictive model building approach has a solid foundation, it would be difficult in situations where we need to provide personalized recommendation for each customer. We need more data showing customer influences to build robust models that can predict offers in all cases.

The heuristic method was better in its approach to recommend offers to all customers for our sampled data. The problem here was it was targeted recommendation and might not influence the entire customer base using the mobile app. Also, the only metric we are using here is the count of influenced customers (or we can call it votes of offer influences). This might at times not be useful if there are not many influenced users at all in the population and we still end up recommending offers to customers.

Improvement

For the tree-based approach we found that our recall rates were not very high especially when we look at the influenced customers. Values like 0.28 and 0.19 were seen in the summary table for few of the offer types. This was using our 600 estimators with a maximum depth of 140. Even a basic random forest model we had built out earlier had similar recall rates for influenced users. We surely need to focus on such poor recall problems as it could be blocker for our model performance during deployment.

To address this,

1. We could try changing our modeling strategy a bit to focus on a demographic related record to build out our Random Forest models with these records and see if our model precision and recall improves any further.

2. Another strategy we want to use is further tuning our Random Forest model's hyper-parameters. We can try out a Random Grid Search mechanism to tune our hyper-parameters like the number of tree estimators, maximum features and maximum

depth of tree for say 500 iterations. This will be a very long process but in the end we will end up with some optimum set of hyper parameters we want to use which will surely improve our model performance.

3. Finally, we should explore our data to see how many influenced customers are available for each offer types. In cases where there are many 0's in the indicator variables than 1's we could end up getting poor recall rates for 1's and better recall rates for 0's thus averaging out our recall rates to 0.77 (as was the case with our basic_rf_model). So, it is better to collect data with some transactions that could potentially involve more influenced customer records. Also as mentioned in 1 above, a better sampling of the data could also help for better models fitted on such samples. So, we could also focus on sampling the dataset based on demographic rules and building better models using those samples.

Finale

With all said, in case of customer recommendations, it is always better to choose what type of marketing we need for our products. If needed, we should try out multiple modelling approaches and not be afraid to investigate even a collaborative modelling approach that uses both the model based and heuristic approaches and recommend offers to all customers using the mobile app.

A comprehensive analysis of the two approaches, python program and Jupyter notebook code are available here, Starbucks-Capstone-Project-GitHub-Repo

References

1. https://en.wikipedia.org/wiki/Recommender_system
2. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
4. https://barnraisersllc.com/2018/10/01/data-mining-process-essential-steps/
5. https://classroom.udacity.com/nanodegrees/nd009t/parts/2f120d8a-e90a-4bc0-9f4e-43c71c504879/modules/2c37ba18-d9dc-4a94-abb9-066216ccace1/lessons/4f0118c0-20fc-482a-81d6-b27507355985/concepts/480e9dc2-4726-4582-81d7-3b8e6a863450