Data Science Capstone Project

Terry Strickland
Tuesday, 13June2023

Table of Contents

Project Overview

We were provided with three distinct datasets from Starbucks, each containing information on promotional offers sent to customers, recorded transactions related to those offers, and demographic data about the customers.

The data set contains simulated information that mimics how customers use the Starbucks Rewards mobile app. Starbucks sends out an offer to mobile app users every few days. An offer might be a simple advertisement for a beverage or a discount or BOGO (buy one get one free) offer. Some users may not receive any offers during certain weeks, and not all users receive the same offer.

Project Goal

The goal is to combine transaction (transcript.json), demographic and offer data to determine which demographic groups respond best to which offer type.

Data Overview

There are three files:

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

Descriptions for each files:

portfolio.json

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

profile.json

- age (int) - age of the customer
  - (numeric) missing value encoded as 118
- became_member_on (int) - date when customer created an app account
- gender (str) - gender of the customer (note some entries contain 'O' for other)
- id (str) - customer id
- income (float) - customer's income

transcript.json

event (str) - record description (ie transaction, offer received, offer viewed, etc.)

person (str) - customer id
time (int) - time in hours since the start of the test. The data begins at time t=0
value - (dict of strings) - either an offer id or transaction amount depending on the record
- offer id: (string/hash) not associated with any "transaction"
- amount: (numeric) money spent in "transaction"
- reward: (numeric) money gained from "offer completed"

Data Exploration

Portfolio.json

Has 10 rows and 6 columns where each row has a unique 'id' value

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

We have a total of:

- 4 unique BOGO (buy one get one) offers
- 4 unique discount offers
- 2 unique informational offers

```
offer_type
bogo             4
discount         4
informational    2
```

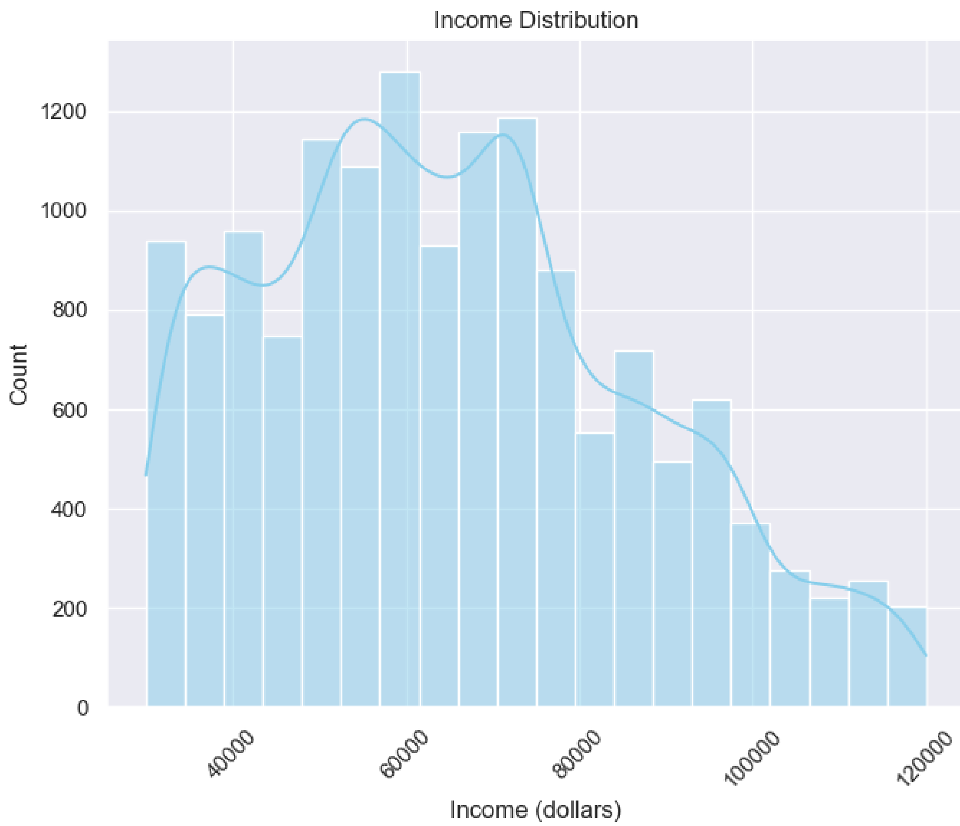
No data processing is required for this data set.

Profile.json

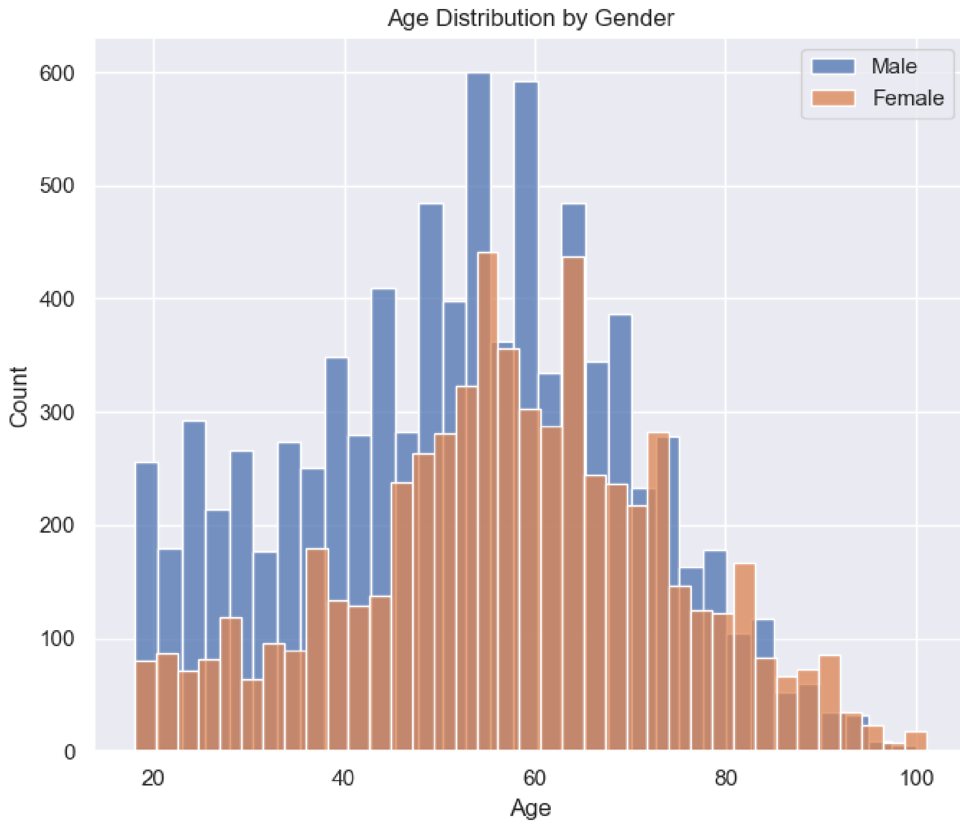
Key observations:

- Has 17000 rows and 5 columns
- Each row has a unique 'id' value that represents a customer
- The 'age' column with a value of 118 is considered as missing value
- The rows on the 'age' column is 118, the value for 'gender' and 'income' is None and NaN respectively
- A quick check reveals that there are 2175 rows with missing values on columns: 'age', 'gender' and 'income'

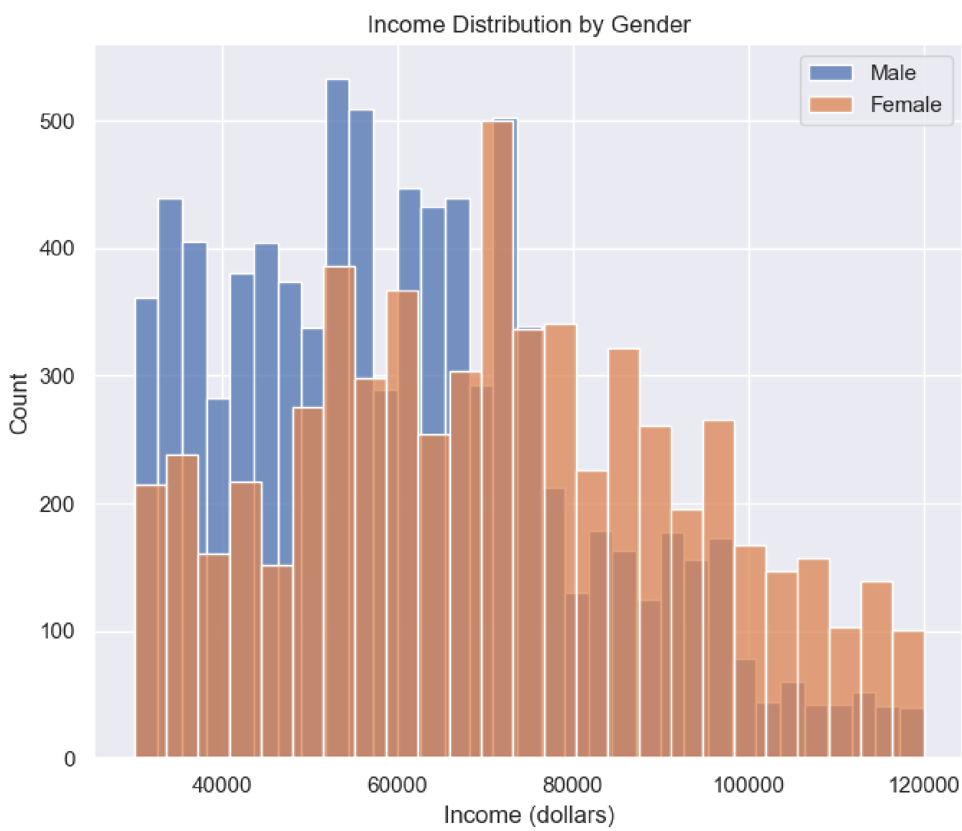
Most customers' income is around $58,000



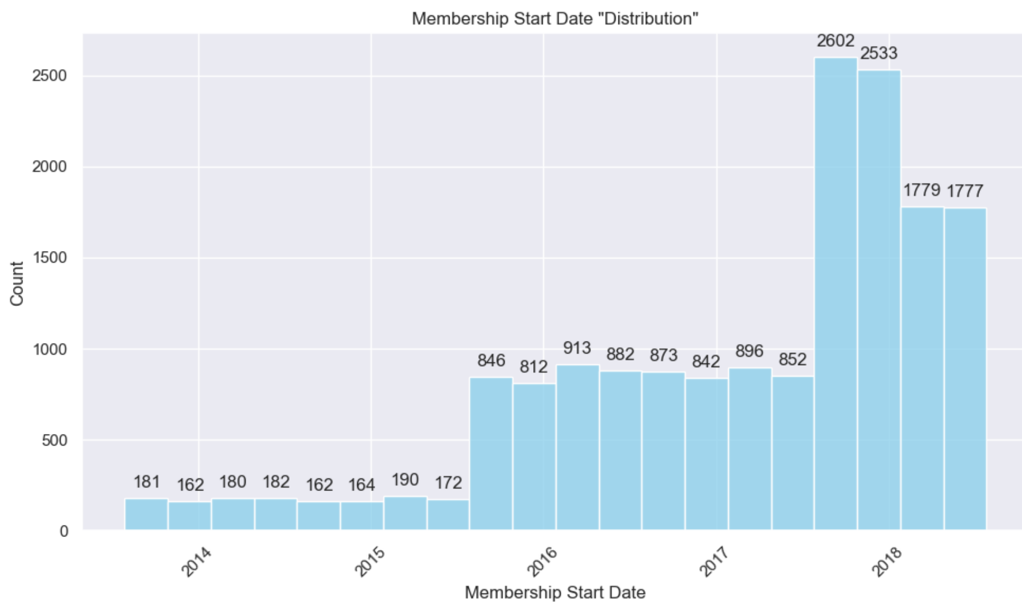
Most customers are between 50 and 70 years old, with more male customers being older than female customers



Female customers have higher income



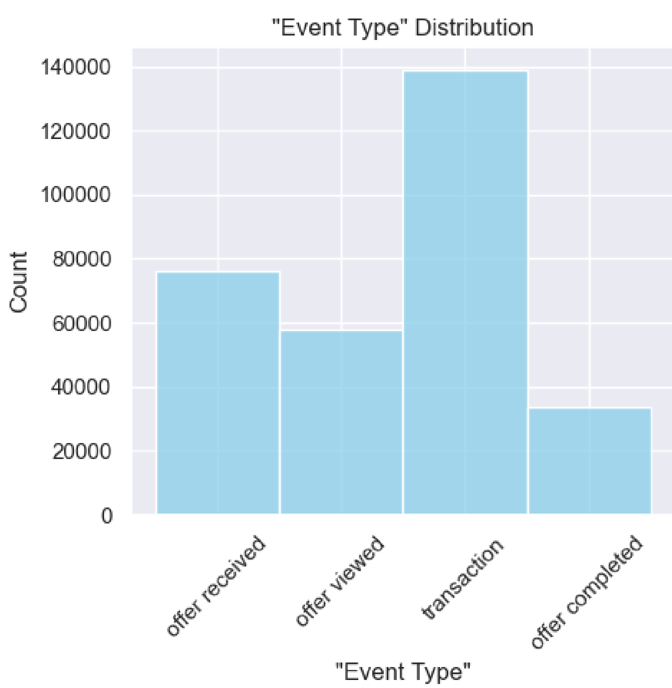
A lot of customers started membership in 2017 and 2018



Transcript.json

There are a total of 30,6534 rows and when we separate these rows by 'event type' we get:

- 76277 for 'offer received'
- 57725 for 'offer viewed'
- 138953 for 'transaction'
- 33579 for 'offer completed'



Data Processing (transcript.json)

Step 1

- we extract the 'data' from the 'value' column and create a new column called 'offer_id'
- 4 new columns are created: 'offer id', 'amount', 'offer_id', 'reward'
- we then 'move' data from 'offer_id' and 'offer id' column to a new column 'id_temp'
- drop the 'offer_id' and 'offer id' column
- rename the 'id_temp' column back to 'offer_id'
- we now have a 'offer_id' column that contains the 'id' from the 'value' column
- the final data frame is called 'transcript'

```
transcript.head()
```

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

Step 2

- we now merge the 'transcript' and 'portfolio' data frame together
- why? we're 'linking' the 'events' in 'transcript' data frame to the 'offer_type' and 'id' in the 'portfolio' data frame
- this will create our 'target feature' (y) column called 'effective_offer'

- dropping the 'reward_x' and 'reward_y' column (we don't need them)
- make a new data frame 'transcript2' and sort the 'time' column in ascending order. this will give us the 'flow' of the events

Defining the 'effective_offer'

- we define an 'effective_offer' as:
  - 'offer viewed' -> 'transaction'

  the 'effective_offer' is '1' if the above condition is met

- we define 'not effective_offer' as:
  - 'offer received' -> 'offer viewed'
  - 'offer received' -> 'offer completed'

  the 'effective_offer' is '0' if the above condition is met

- we again merge 'transcript2' and 'profile' data frame together
- and drop the columns that we don't need

```python
transcript2.head(3)
```

Python

	person	event	value	time	amount	offer_id	channels	difficulty	duration	offer_type
55972	0009655768c64bdeb2e877511632db8f	offer received	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	168	NaN	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]	0.0	3.0	informational
77705	0009655768c64bdeb2e877511632db8f	offer viewed	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	192	NaN	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]	0.0	3.0	informational
89291	0009655768c64bdeb2e877511632db8f	transaction	{'amount': 22.16}	228	22.16	NaN	NaN	NaN	NaN	NaN

Step 3

- using 'transcript2' data frame, we create 2 new data frames:
  - bogo_only1
  - discount_only1

These data frame will have only the 'BOGO' (buy one get one) and 'discount' 'offer_type', the 'offer_type' column

- we then filter these 2 data frames and look for the 'effective_offer'
- and then 'link' the 'effective_offer' to the 'person' column
- this will tell us which customer completed the 'BOGO' and 'discount' offer

bogo_only1

```python
bogo_only1.head(2) # bogo offers only
```

Python

person	event	value	time	amount	offer_id	reward	channels	difficulty	offer_type	duration	pre_offer_id	completed_offer
4bdeb2e877511632db8f	offer received	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}	408	NaN	f19421c1d4aa40978ebb69ca19b0e20d	5.0	[web, email, mobile, social]	5.0	bogo	5.0	NaN	NaN
4bdeb2e877511632db8f	offer completed	{'offer_id': 'f19421c1d4aa40978ebb69ca19b0e20d...	414	NaN	f19421c1d4aa40978ebb69ca19b0e20d	5.0	[web, email, mobile, social]	5.0	bogo	5.0	NaN	0.0

discount_only1

```python
discount_only1.head(2) # discount offers only
```

✓ 0.0s

	person	event	value	time	amount	offer_id	reward	channels	difficulty	offer_type	duration	pre_offer
9	0009655768c64bdeb2e877511632db8f	offer received	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	504	NaN	fafdcd668e3743c1bb461111dcafc2a4	2.0	[web, email, mobile, social]	10.0	discount	10.0	N
11	0009655768c64bdeb2e877511632db8f	offer completed	{'offer_id': 'fafdcd668e3743c1bb461111dcafc2a4...	528	NaN	fafdcd668e3743c1bb461111dcafc2a4	2.0	[web, email, mobile, social]	10.0	discount	10.0	N

Step 4

- add the 'effective_offer' column to the 'offers_bogo' and 'offers_discount' data frame
- these 2 data frames have 'person', 'offer_id', 'effective_offer' columns that tells us which customer uses which 'offer' and was it 'effective' or not

```
offers_bogo
```

	person	offer_id	effective_offer
0	0011e0d4e6b944f998e987f904e8c1e5	9b98b8c7a33c4b65b9aebfe6a799e6d9	1
1	0020c2b971eb4e9188eac86d93036a77	4d5c57ea9a6940dd891ad53e9dbe8da0	1
2	0020ccbbb6d84e358d3414a3ff76cffd	9b98b8c7a33c4b65b9aebfe6a799e6d9	1
3	0020ccbbb6d84e358d3414a3ff76cffd	f19421c1d4aa40978ebb69ca19b0e20d	1
4	004b041fbfe44859945daa2c7f79ee64	f19421c1d4aa40978ebb69ca19b0e20d	1

Engineering a 'feature' for the machine learning ‘model’

- the 'became_member_on' will be the column that we will 'engineer'
- by converting the current format of YYYYMMDD to 'the number of days' they are a member, counting from december 31st 2018

```
offers_bogo.head()
```

person	offer_id	effective_offer	gender	age	income	membership_days
0011e0d4e6b944f998e987f904e8c1e5	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	O	40	57000.0	356
0020c2b971eb4e9188eac86d93036a77	4d5c57ea9a6940dd891ad53e9dbe8da0	1	F	59	90000.0	1032
0020ccbbb6d84e358d3414a3ff76cffd	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	F	24	60000.0	780
0020ccbbb6d84e358d3414a3ff76cffd	f19421c1d4aa40978ebb69ca19b0e20d	1	F	24	60000.0	780
004b041fbfe44859945daa2c7f79ee64	f19421c1d4aa40978ebb69ca19b0e20d	1	F	55	74000.0	237

Data Processing for 'offers_bogo' data frame

- we again merge the 'offers_bogo' data frame with the 'profile' data frame
- convert the 'channels' column into a 'one hot encoding' format
  - you will end up with 4 new columns: 'web', 'email', 'mobile', 'social'
  - with values of 0 or 1
- convert the 'gender' column into a 'one hot encoding' format
  - you will end up with 3 new columns: 'F', 'M', 'O'
  - with values of 0 or 1

```
offers_bogo.head()
```

person	offer_id	effective_offer	age	income	membership_days	reward	difficulty	duration	offer_type	web	email	social	mobile	gen
987f904e8c1e5	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	40	57000.0	356	5	5	7	bogo	1.0	1.0	0.0	1.0	
c86d93036a77	4d5c57ea9a6940dd891ad53e9dbe8da0	1	59	90000.0	1032	10	10	5	bogo	1.0	1.0	1.0	1.0	
3414a3ff76cffd	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	24	60000.0	780	5	5	7	bogo	1.0	1.0	0.0	1.0	
3414a3ff76cffd	f19421c1d4aa40978ebb69ca19b0e20d	1	24	60000.0	780	5	5	5	bogo	1.0	1.0	1.0	1.0	
daa2c7f79ee64	f19421c1d4aa40978ebb69ca19b0e20d	1	55	74000.0	237	5	5	5	bogo	1.0	1.0	1.0	1.0	

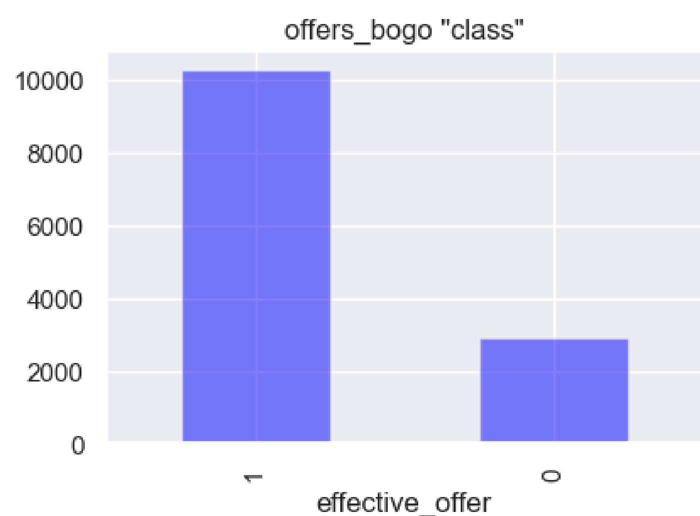
Data Processing for 'offers_discount' data frame

- we first need to 'address' the NaN values in the 'gender' and 'income' column by dropping them (the NaN values)
- we then repeat the same process as 'offers_bogo' data frame
- but we wrote a function 'convert_process2()' to accomplish the task

```
offers_discount.head()
```

person	offer_id	effective_offer	age	income	membership_days	reward	difficulty	duration	offer_type	web	email	so
987f904e8c1e5	0b1e1539f2cc45b7b9fa7c272da2e1d7	1	40	57000.0	356	5	20	10	discount	1.0	1.0	
987f904e8c1e5	2298d6c36e964ae4a3e7e9706d1fb8c2	1	40	57000.0	356	3	7	7	discount	1.0	1.0	
ac86d93036a77	fafdcd668e3743c1bb461111dcafc2a4	1	59	90000.0	1032	2	10	10	discount	1.0	1.0	
3414a3ff76cffd	2298d6c36e964ae4a3e7e9706d1fb8c2	1	24	60000.0	780	3	7	7	discount	1.0	1.0	
5cc97a6903f4f0	fafdcd668e3743c1bb461111dcafc2a4	1	26	73000.0	558	2	10	10	discount	1.0	1.0	

Justification for Selecting the 'weighted average F1 score,' as the 'metric' for the 'model'



as you can see, we have a 'class imbalance' problem for both dataframes:
- offers_bogo
- offers_discount

both have the 'class' of 1 (effective offer) as the 'majority class'

Thus we will use the 'weighted average F1 score,' as the 'metric' for the 'model' because this 'problem' is considered to be an 'imbalanced classification'.
Using 'weighted average F1 score' helps to give a more representative score of the model's performance\ across all classes, by giving more importance to the classes with more instances, and If the model doesn't do well in predicting a class that has only a small number of instances, the 'weighted average F1-score' won't be dramatically reduced.

Implementation (selecting the type of 'classification algorithm' to use)

This is a (yes/no) 'binary classification' problem where the 'outcome' is either:
- 'effective offer' == 1
- 'effective offer' == 0

We will use 'Random Forest Classifier' (RFC) as the 'classification algorithm' because:

- Feature importance: Random Forest offers built-in 'feature importance' which will be use to select the 'important feature'.

- Ease of use: Random Forest requires less preprocessing of data. It can handle missing values and categorical data, and is not affected by the scaling of features.

- Handling overfitting: By using multiple 'trees', compare to using a single 'decision trees'.

We will also by using 'K nearest neighbors' (KNN) as a 'benchmark model' to compare with the 'RFC' model:

- Performance with imbalanced data: KNN may not perform well on 'imbalanced' data because predictions are based on the nearest neighbors, thus prediction may be biased towards the majority class in case of imbalanced data.

- Sensitive to irrelevant features: KNN treats every 'feature' with the same 'importance' and does not perform well if the dataset has irrelevant 'features'.

Building the machine learning 'model'

We will build 3 machine learning 'models'

- KNN (K Nearest Neighbors), for benchmarking against the other 2 models below
- Random Forest Classifier, for the 'offers_bogo' data frame
- Random Forest Classifier, for the 'offers_discount' data frame

All the 'models' with start with a randomly selected 'hyperparameters' and then we will use 'GridSearchCV' to find the best 'hyperparameters' for each 'model'

```
# create a KNN model
compareModel1 = KNeighborsClassifier(
    n_neighbors=3,  # choose the number of neighbors
    weights='uniform',  # weight function used in prediction
    algorithm='auto',  # algorithm used to compute the nearest neighbors
    leaf_size=30,  # leaf size passed to BallTree or KDTree
    p=2,  # power parameter for the Minkowski metric
    metric='minkowski',  # the distance metric to use for the tree
    metric_params=None,  # additional keyword arguments for the metric fun
    n_jobs=None  # the number of parallel jobs to run for neighbors search
)

bogoModel1 = RandomForestClassifier(
    random_state=2,  # sets the 'seed' for the random number generator, so
    max_depth=11,  # limits how many splits each tree can have, which help
    min_samples_split=10,  # a 'node' in the tree will only split if it ha
    n_estimators=20,  # sets the number of 'trees in the forest'. More tre
    min_samples_leaf=20  # each "leaf" at the end of a tree must have at l
)
```

Our (total 13) 'features' (X) will be:
- 'age', 'income', 'membership_days', 'reward', 'difficulty', 'duration', 'web', 'email', 'mobile', 'social', 'gender_F', 'gender_M', 'gender_O'

Our 'target feature' (y) will be:
- 'effective_offer'

Results (before 'hyperparameter tuning' with GridSearch)

The KNN model (compareModel1) has:
- training accuracy of 0.8745
- testing accuracy of 0.7700

The Random Forest Classifier model (bogoModel1) has:
- training accuracy of 0.8263
- testing accuracy of 0.8160

The Random Forest Classifier model (discountModel1) has:
- training accuracy of 0.8959
- testing accuracy of 0.8322

We can see that the Random Forest Classifier model for both 'offers_bogo' and 'offers_discount' data frame has a higher accuracy than the KNN model.

	KNeighborsClassifier_bogoModel1	RandomForestClassifier_bogoModel1	KNeighborsClassifier_discountModel1	RandomForestClassifier_discountModel1
train_time	0.025999	0.107413	0.024514	0.118890
pred_time	0.534127	0.026462	0.467992	0.025282
training_score	0.874501	0.826298	0.895924	0.869608
testing_score	0.769962	0.815970	0.832212	0.869410

Searching for the best 'hyperparameters' using 'GridSearch'

We will use 'GridSearch' for both dataframes:

- 'offers_bogo'
- 'offers_discount'

with the following 'parameter grid' to search through:

param_grid={
'max_depth' : [5,10,15],
'n_estimators': [25,30,40],
'min_samples_split': [2, 10, 20],
'min_samples_leaf': [2, 10, 15, 20],
}

```
param_grid={
            'max_depth' : [5,10,15],
            'n_estimators': [25,30,40],
            'min_samples_split': [2, 10, 20],
            'min_samples_leaf': [2, 10, 15, 20],
            }
```

Results (after 'hyperparameter tuning' with GridSearch)

After running 'GridSearch', we found the best 'hyperparameters' for 'offers_bogo' dataframe to be:

'max_depth': 15,
'min_samples_leaf': 15,
'min_samples_split': 2,
'n_estimators': 40

and 'offers_discount' dataframe to be:

'max_depth': 10,
'min_samples_leaf': 20,
'min_samples_split': 2,
'n_estimators': 25

Results (after making 2 new 'models' with the best 'hyperparameters' from GridSearch)

'bogo_model2'
Training accuracy: 0.8745
Test accuracy: 0.7700

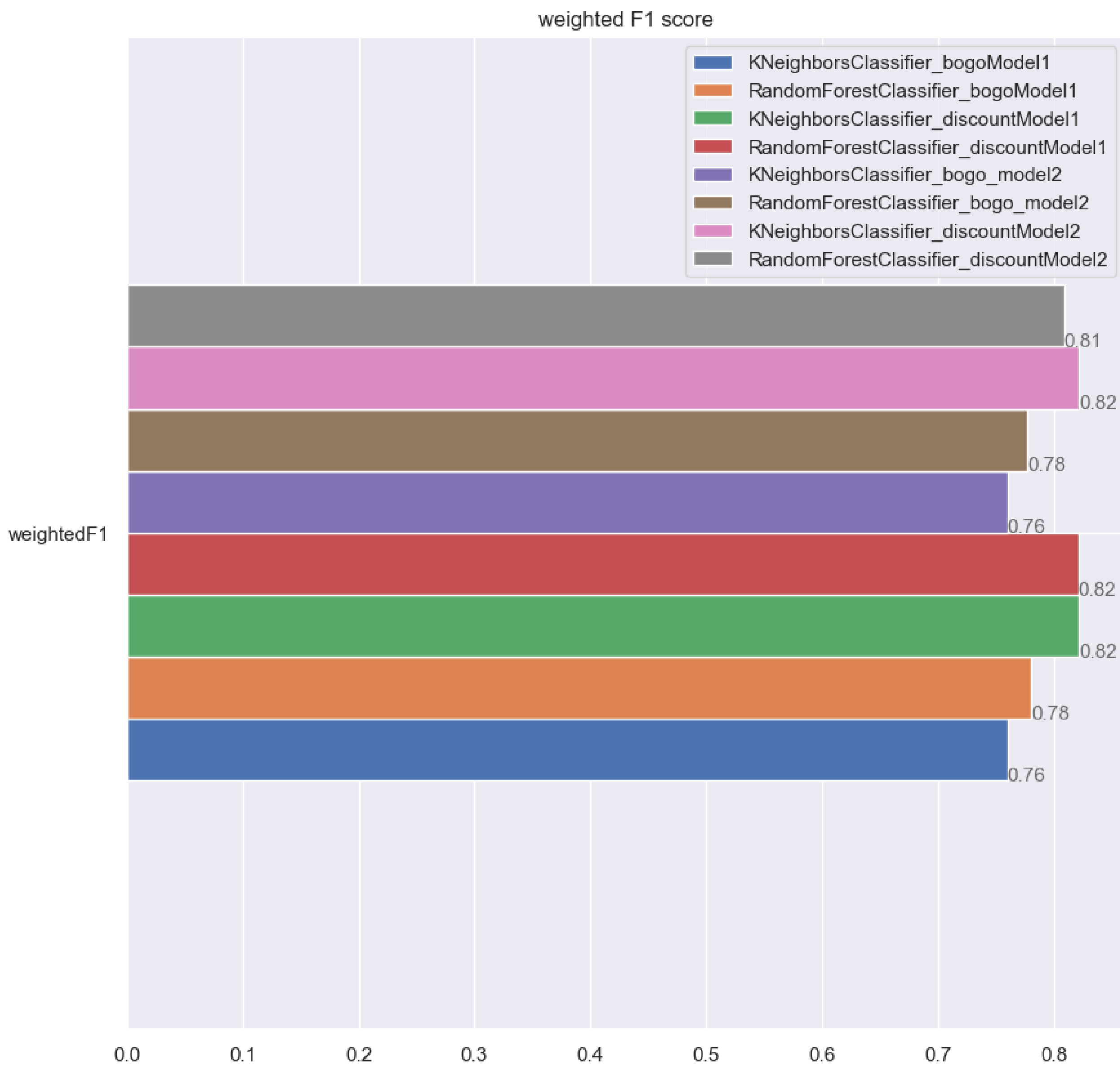
'discount_model2'
Training accuracy: 0.8959
Test accuracy: 0.8322

The first 2 models:
- 'bogo_model1'
- 'discount_model1'

(before 'hyperparameter tuning') has a higher accuracy than the 2 models after 'hyperparameter tuning' with 'GridSearch'

	train_time	pred_time	training_score	testing_score
RandomForestClassifier_bogoModel1	0.107413	0.026462	0.826298	0.81597
RandomForestClassifier_discountModel1	0.118890	0.025282	0.869608	0.86941

Comparing the 'models' with high 'weighted average F1 score' and the 'baseline model'



Why did KNN have a better 'weighted average F1 score' than RFC?

for 'offers_discount' dataset, and it is also an 'imbalanced' dataset

- KNeighborsClassifier_discountModel2 with 0.821472
- RandomForestClassifier_discountModel2 with 0.808719

The improved score is minimal, (0.821472 - 0.808719) / 0.808719 * 100 = 1.57%

Reason might be because:

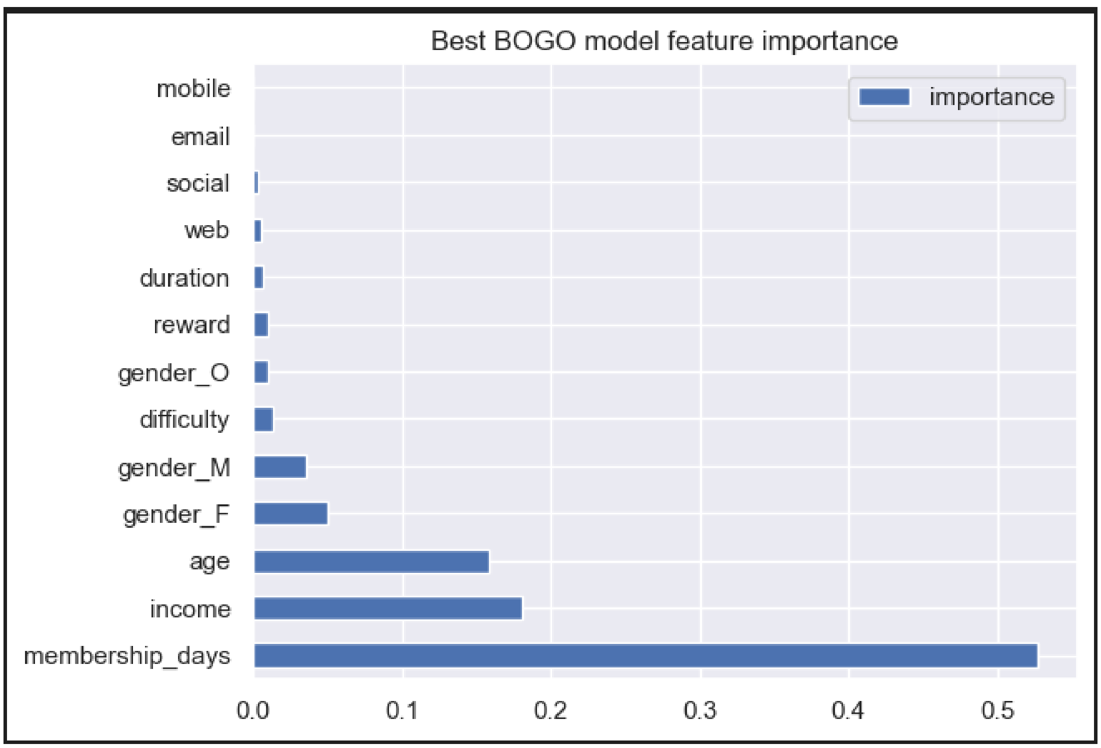
- Random Forest generally 'assumes' that 'features' have some sort of hierarchical structure
- KNN is a non-parametric model, meaning it makes no assumptions about the underlying data distribution.

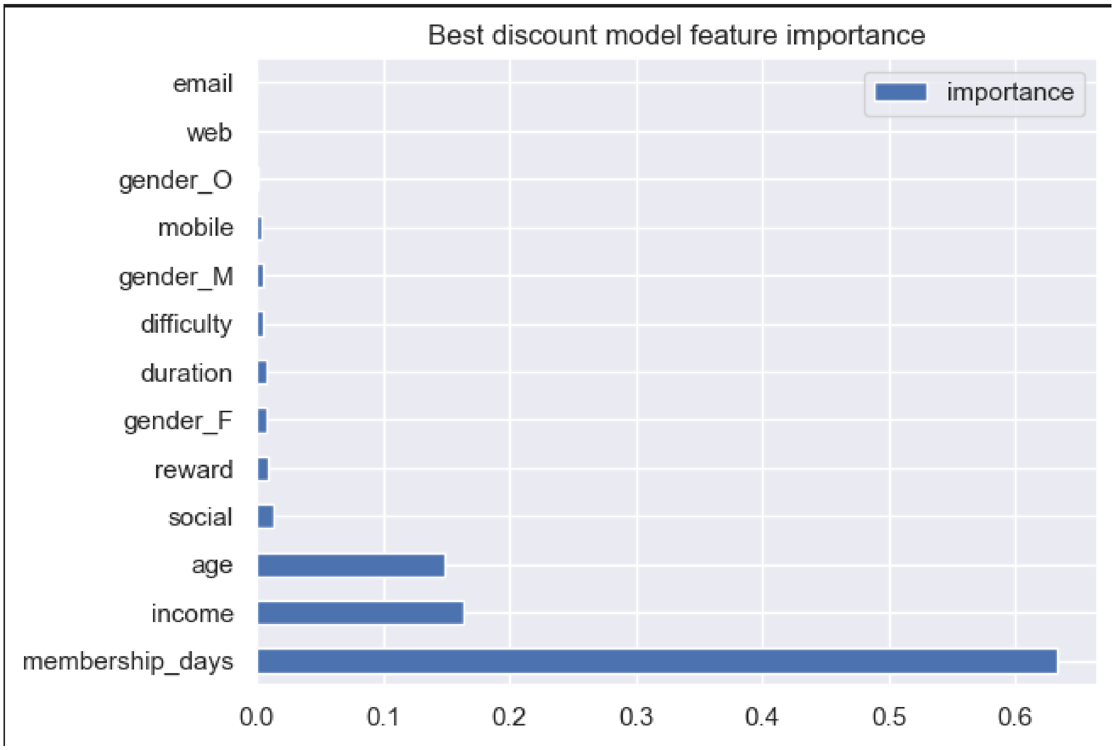
Finding the 'features' that have impact on the 'effective_offer' for both 'offers_bogo' and 'offers_discount' dataframe

Looking at the bogoModel1.feature_importances_ and discountModel1.feature_importances_ we can see that the 'features' that have the most impact on the 'effective_offer' are:

- membership_days
- income
- age

Where 'membership_days' has the most impact on the 'effective_offer' for both 'offers_bogo' and 'offers_discount' dataframe





'Complication' (checking for 'Data Leakage' with the machine learning 'model')

Feature importance: If any of the 'features' are 'very predictive', it's possible they are causing leakage.

From the 2 plots above, membership_day seems to be 'overwhelmingly' 'important'
compared to the other 'features'. This could be a sign of 'data leakage'. The keyword here is 'could be'.

Conclusion

Possible Improvements for KNN 'Model'

It was unexpected that KNN would perform better (slightly) than RFC for the 'discount' (or any) dataset,

but here are some possible improvements that can be made to the 'model' to improve the 'weighted average F1 score':

- Tuning the number of neighbors (K) for KNN 'model':

- Run the model with various K values and pick the one which performs best.

- 'Dimensionality Reduction'

- Big fancy term for reducing the number of 'features' in the dataset using 'Principal Component Analysis' (PCA)

Possible Improvements for 'Random Forest Model'

- 'Tuning Hyper-parameter' using 'Grid Search', this was done in this project.

- 'Feature Importance'

- 'Random Forest' provides a tool to measure the importance of each 'feature' in the prediction process (done above).

- Any 'feature' with small 'importance score' could be removed from the dataset and we can check if the 'weighted average F1 score' will improve, but I will not be doing that.

Reflection

Apply machine learning 'models' to 'classify' customer responses to different type of offers (promotions). We used two different models - K-Nearest Neighbors (KNN) and Random Forest - across various types of offers and datasets.

The analysis began with preprocessing 3 different datasets, which involved cleaning, encoding (one-hot-encoding), and splitting the data into training and testing sets. We then 'trained' our 'models' and evaluated them using the main metrics of 'weighted average F1 score'.

All 3 'models' produced a 'weighted average F1' score above 80%. The dataset being imbalanced and somehow KNN has better or the same 'weighted average F1 score' with the Random Forest model, which is supposed to perform better than KNN.

We noted areas for improvement. Hyperparameter tuning, feature scaling for KNN, dimensionality reduction, and analysis of 'feature importance' for 'Random Forest' were some of the things suggested to enhance the models' performance.