Machine Learning Nano Degree Capstone Submission – Starbucks – Ishan Pathak

1. Introduction

Starbucks is a well-known coffee chain which is looking for improving the effectiveness of the interaction which it has with its existing customers.

In the current context, Starbucks has rolled out three diverse types of offers to its customers:

- Discount Offer – Discount on next purchase above a certain amount
- BOGO – Buy One Get One Free for purchase worth a certain amount
- Informational – This has information on existing Starbucks Products

Not all the existing customers react to these offers similarly. Take for example, a regular Starbucks user – In all probability, the offers delivered to this customer would not have a significant impact on the purchase activity for this customer. On the other hand, a customer who is infrequent to Starbucks or using a substitute product in market (say Costa Coffee) could react favorably to some of these offers.

Domain

This project is derived from the field of Customer Relationship Management (CRM) in Starbucks. One main concern for CRM is to interact with not only current customers, but also earlier and potential customers, so that the company can effectively keep its business relationship with all customers.

Problem Statement

The Problem Statement for this Capstone Challenge is to analyze the dataset for Starbucks Customers and build a model that can predict whether a customer will respond to an offer. For Starbucks, this is an important question since there is a direct marketing cost associated sending out offers to customers. Hence, we would like to roll-out offers to customers where there is a high likelihood for the offer to be desirable. Also, if we are unable to decipher certain good offers to customers then that would represent a loss in expected Starbucks sales.

Solution Statement

In my current implementation, I have first classified all the offers rolled out to customers into one of the two categories:

- **Desirable**: An offer will be classified as desirable in any of the two situations:
  - If the customer receives the offer, then views the offer then conducts a transaction sufficient to complete the offer all within the offer validity period then the offer is desirable
  - If the customer receives the offer, then views the offer then conducts a transaction within the validity period which is insufficient to meet the offer difficulty can still be considered a desirable offer since we can conclude that the transaction was performed under the influence of the offer
- In all other cases, the offer will be classified as **undesirable**

In the implementation, we are even including the offers that could not be completed due to not meeting the minimum transaction thresholds (I.e., offer difficulty) but still after viewing these offers, the customer did buy certain products which can help us conclude that the purchase was done under the influence of the offer. In this case, rolling the offer is leading to an increase in Starbucks sales thereby making it favorable. The task for us at hand then becomes classification of offers into of the above two categories in the data dataset. Note that the analysis is done separately for different classes.

We use a benchmark model which is Logistic Regression to perform the classification task. Logistic Regression is a supervised learning algorithm which is used for categorical target variables. Logistic Regression is implemented through **sklearn** library in Python. In the next step, we implement the XGBoost Sage maker Algorithm and try to assess the relative performance across the two models. Finally, we chose a suitable model out of the two for each of the offer types.

Performance of the models is computed through the following evaluation metrics:

- Accuracy
- Precision
- Recall
- F1 Score

For our classification task we need to correctly classify the positives (since errors here would lead to an economic sales loss) as well as the negatives (since errors here would lead to excessive direct marketing costs). Accuracy is the total fraction of correct predictions. Precision tells us the extent to which the positive class predictions have been correct, and recall is the percentage of positive ground-truths correctly labelled as positive. F1 score is the harmonic mean of precision and recall.

Model selection for prediction is dependent on the business questions which we are trying to answer. False negatives predicted by the model would mean a future economic loss since the customer would have desirably responded to the offer by conducting a transaction. False positives predicted by the model would mean wasted marketing cost incurred in rolling the offer to the customer. Ideally, we would like to reduce the proportion of both false positives and false negatives. For business situations which require us to predict the negatives correctly, we use the accuracy score as the deciding criteria whereas for business situations which require us to predict the positives correctly, we use the re call rate as the deciding criteria. We will also use the confusion matrices to infer performance of models under different business con texts. In the following sections, I explain the Data Exploration and Analysis Section.

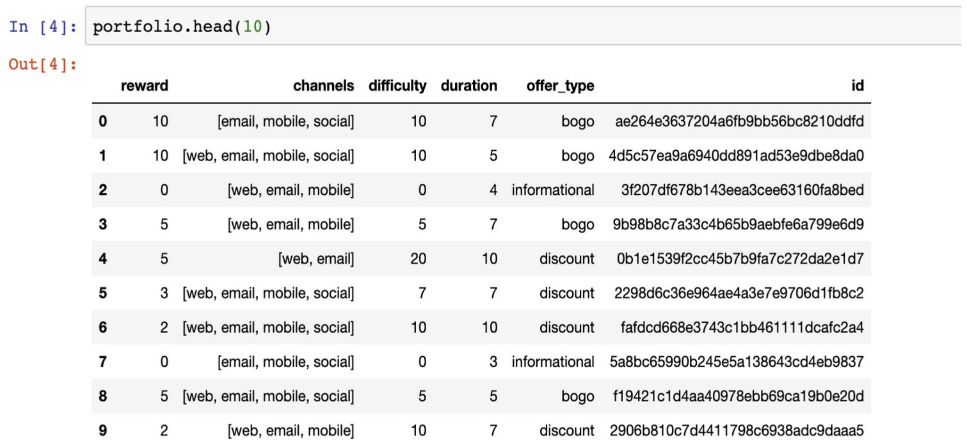
2. Datasets and Inputs

The datasets are contained in three files:

- **Portfolio.json** - containing the offer id and metadata about each offer
- **Profile.json** - containing the demographic details for each customer
- **Transcript.json** - containing the records for transactions, offers received, offers viewed and offers completed

We analyze each of these datasets and describe our findings below:

The portfolio dataset contains details on the communicated offers. It contains information on the channels through which the offers are rolled out as well as the difficulty associated with the offer and the reward post offer completion. Offer difficulty is defined as the minimum spending required to complete the offer. Note that highly difficult offers will have lower chances of completion.



```
In [4]: portfolio.head(10)
```

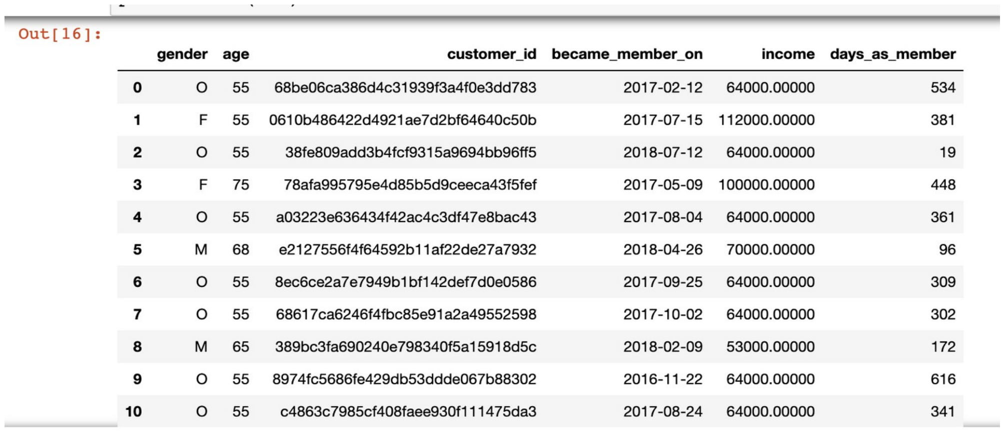
Out[4]:

	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

Fig 1: portfolio dataset

For the profile dataset we see that has many missing values in income field, gender field and age (where we see a high default value of 118). Additionally, all the missing values occur concurrently for all three fields over 2,175 entries. We replace these missing values with the median of the remaining dataset for income(64000), with the median age (55) of remaining dataset for age and create a separate field for gender namely "O".

We create an additional attribute named "**days_as_member** which corresponds to the number of days of membership for a customer. This attribute is computed using the number of days elapsed between the date on which membership happened and the latest date on which a customer became member rounded to the neared Month End Cob Date (July Month End 2018). After making these adjustments, below is a snapshot of the profile dataset.



Out[16]:

	gender	age	customer_id	became_member_on	income	days_as_member
0	O	55	68be06ca386d4c31939f3a4f0e3dd783	2017-02-12	64000.00000	534
1	F	55	0610b486422d4921ae7d2bf64640c50b	2017-07-15	112000.00000	381
2	O	55	38fe809add3b4fcf9315a9694bb96ff5	2018-07-12	64000.00000	19
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	2017-05-09	100000.00000	448
4	O	55	a03223e636434f42ac4c3df47e8bac43	2017-08-04	64000.00000	361
5	M	68	e2127556f4f64592b11af22de27a7932	2018-04-26	70000.00000	96
6	O	55	8ec6ce2a7e7949b1bf142def7d0e0586	2017-09-25	64000.00000	309
7	O	55	68617ca6246f4fbc85e91a2a49552598	2017-10-02	64000.00000	302
8	M	65	389bc3fa690240e798340f5a15918d5c	2018-02-09	53000.00000	172
9	O	55	8974fc5686fe429db53ddde067b88302	2016-11-22	64000.00000	616
10	O	55	c4863c7985cf408faee930f111475da3	2017-08-24	64000.00000	341

Fig 2: profile dataset after imputation process

We construct a scatterplot matrix for income and age of different customers, we conclude that generally the income increases with age as expected.

```
In [13]: pd.plotting.scatter_matrix(profile[['age','income']], figsize = (6,6))

Out[13]: array([[<AxesSubplot:xlabel='age', ylabel='age'>,
                 <AxesSubplot:xlabel='income', ylabel='age'>],
                [<AxesSubplot:xlabel='age', ylabel='income'>,
                 <AxesSubplot:xlabel='income', ylabel='income'>]], dtype=object)
```

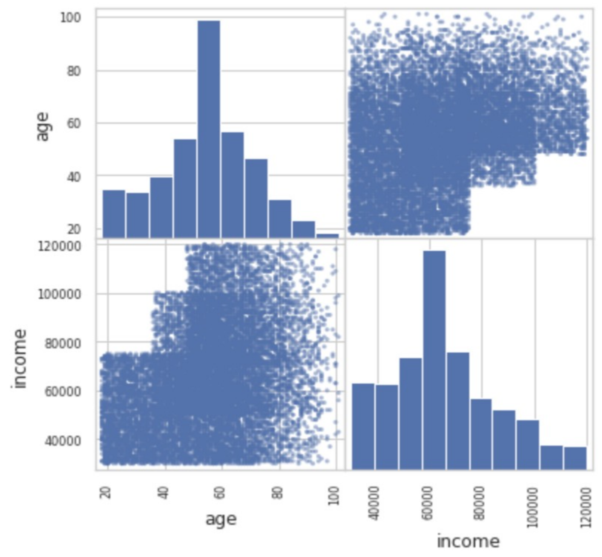
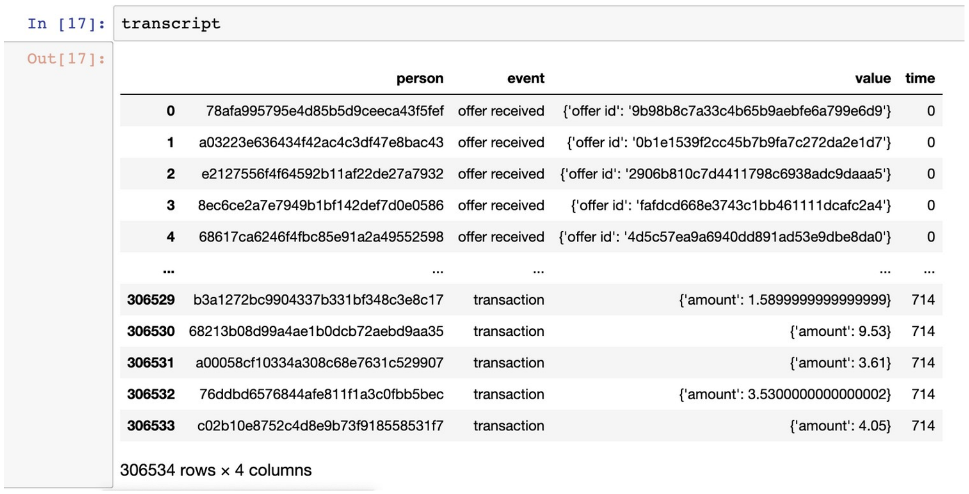


Fig 3: scatterplot for age and income for different customers

The Transcript dataset has information on the different events associated with customers over the Starbucks testing period. These events include receiving an offer, viewing an offer, conducting a transaction and completing an offer. The value column contains the information on "**offer_id**" against the transaction for offer related events (offer receive, offer view or offer complete) and the amount of transaction for transactions conducted. These two aspects are interrelated in the sense that conducted transactions lead to offer completion. Later we separate the transcript dataset into offer related and transaction related sub datasets.



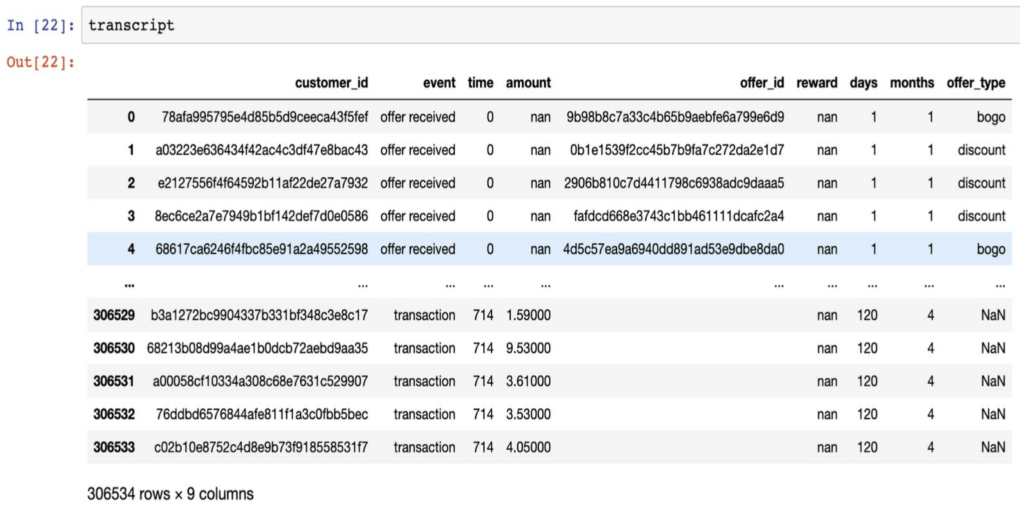
```
In [17]: transcript
Out[17]:
```

	person	event	value	time
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
1	a03223e636434f42ac4c3df47e8bac43	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	0
3	8ec6ce2a7e7949b1bf142def7d0e0586	offer received	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	0
4	68617ca6246f4fbc85e91a2a49552598	offer received	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	0
...	...	...	...	...
306529	b3a1272bc9904337b331bf348c3e8c17	transaction	{'amount': 1.5899999999999999}	714
306530	68213b08d99a4ae1b0dcb72aebd9aa35	transaction	{'amount': 9.53}	714
306531	a00058cf10334a308c68e7631c529907	transaction	{'amount': 3.61}	714
306532	76ddbd6576844afe811f1a3c0fbb5bec	transaction	{'amount': 3.5300000000000002}	714
306533	c02b10e8752c4d8e9b73f918558531f7	transaction	{'amount': 4.05}	714

306534 rows × 4 columns

Fig 4: Transcript dataset

We see that the time column starts from 0 and goes till 714 in increments of 6. Here we can conclude that testing happened for six hours daily each day for 120 days or four months. Hence, we create additional attributes like "days" and "months" which can be used later for studying the aggregate consumer behavior. We also flatten the value column of the transcript dataset. Finally, we pull the offer type against each offer from the portfolio dataset. Below is snapshot of the dataset after these transformations:



```
In [22]: transcript
Out[22]:
```

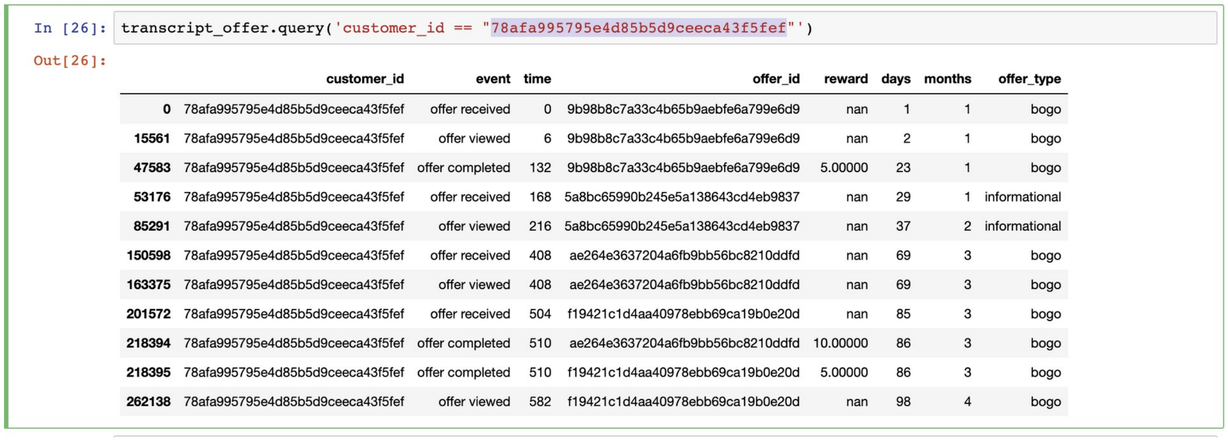
	customer_id	event	time	amount	offer_id	reward	days	months	offer_type
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	0	nan	9b98b8c7a33c4b65b9aebfe6a799e6d9	nan	1	1	bogo
1	a03223e636434f42ac4c3df47e8bac43	offer received	0	nan	0b1e1539f2cc45b7b9fa7c272da2e1d7	nan	1	1	discount
2	e2127556f4f64592b11af22de27a7932	offer received	0	nan	2906b810c7d4411798c6938adc9daaa5	nan	1	1	discount
3	8ec6ce2a7e7949b1bf142def7d0e0586	offer received	0	nan	fafdcd668e3743c1bb461111dcafc2a4	nan	1	1	discount
4	68617ca6246f4fbc85e91a2a49552598	offer received	0	nan	4d5c57ea9a6940dd891ad53e9dbe8da0	nan	1	1	bogo
...	...	...	...	...	...	...	...	...	...
306529	b3a1272bc9904337b331bf348c3e8c17	transaction	714	1.59000		nan	120	4	NaN
306530	68213b08d99a4ae1b0dcb72aebd9aa35	transaction	714	9.53000		nan	120	4	NaN
306531	a00058cf10334a308c68e7631c529907	transaction	714	3.61000		nan	120	4	NaN
306532	76ddbd6576844afe811f1a3c0fbb5bec	transaction	714	3.53000		nan	120	4	NaN
306533	c02b10e8752c4d8e9b73f918558531f7	transaction	714	4.05000		nan	120	4	NaN

306534 rows × 9 columns

Fig 5: Transformed Transcript dataset

Next, we separate the transcript dataset above based on event type. For event type equal to transaction, we call that **"transcript_transaction"** as it has the details on transaction conducted by the customers. Note that this dataset will not have the “**offer_id**” information. The other split is for offer type equal to offer received, offer viewed and offer completed.

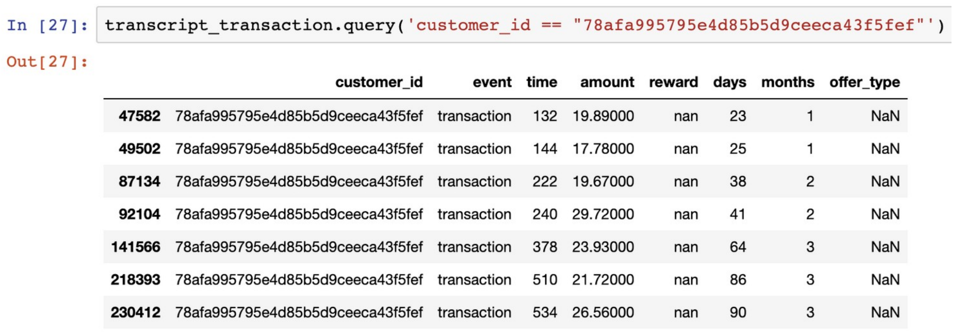
To illustrate how the two datasets are interrelated, we sample the details for a customer id “**78afa995795e4d85b5d9ceeca43f5fef**” (say Customer c).



```
In [26]: transcript_offer.query('customer_id == "78afa995795e4d85b5d9ceeca43f5fef"')
Out[26]:
```

	customer_id	event	time	offer_id	reward	days	months	offer_type
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	0	9b98b8c7a33c4b65b9aebfe6a799e6d9	nan	1	1	bogo
15561	78afa995795e4d85b5d9ceeca43f5fef	offer viewed	6	9b98b8c7a33c4b65b9aebfe6a799e6d9	nan	2	1	bogo
47583	78afa995795e4d85b5d9ceeca43f5fef	offer completed	132	9b98b8c7a33c4b65b9aebfe6a799e6d9	5.00000	23	1	bogo
53176	78afa995795e4d85b5d9ceeca43f5fef	offer received	168	5a8bc65990b245e5a138643cd4eb9837	nan	29	1	informational
85291	78afa995795e4d85b5d9ceeca43f5fef	offer viewed	216	5a8bc65990b245e5a138643cd4eb9837	nan	37	2	informational
150598	78afa995795e4d85b5d9ceeca43f5fef	offer received	408	ae264e3637204a6fb9bb56bc8210ddfd	nan	69	3	bogo
163375	78afa995795e4d85b5d9ceeca43f5fef	offer viewed	408	ae264e3637204a6fb9bb56bc8210ddfd	nan	69	3	bogo
201572	78afa995795e4d85b5d9ceeca43f5fef	offer received	504	f19421c1d4aa40978ebb69ca19b0e20d	nan	85	3	bogo
218394	78afa995795e4d85b5d9ceeca43f5fef	offer completed	510	ae264e3637204a6fb9bb56bc8210ddfd	10.00000	86	3	bogo
218395	78afa995795e4d85b5d9ceeca43f5fef	offer completed	510	f19421c1d4aa40978ebb69ca19b0e20d	5.00000	86	3	bogo
262138	78afa995795e4d85b5d9ceeca43f5fef	offer viewed	582	f19421c1d4aa40978ebb69ca19b0e20d	nan	98	4	bogo

Fig 6: Transcript Offer for Customer



```
In [27]: transcript_transaction.query('customer_id == "78afa995795e4d85b5d9ceeca43f5fef"')
Out[27]:
```

	customer_id	event	time	amount	reward	days	months	offer_type
47582	78afa995795e4d85b5d9ceeca43f5fef	transaction	132	19.89000	nan	23	1	NaN
49502	78afa995795e4d85b5d9ceeca43f5fef	transaction	144	17.78000	nan	25	1	NaN
87134	78afa995795e4d85b5d9ceeca43f5fef	transaction	222	19.67000	nan	38	2	NaN
92104	78afa995795e4d85b5d9ceeca43f5fef	transaction	240	29.72000	nan	41	2	NaN
141566	78afa995795e4d85b5d9ceeca43f5fef	transaction	378	23.93000	nan	64	3	NaN
218393	78afa995795e4d85b5d9ceeca43f5fef	transaction	510	21.72000	nan	86	3	NaN
230412	78afa995795e4d85b5d9ceeca43f5fef	transaction	534	26.56000	nan	90	3	NaN

Fig 7: Transcript Transaction for Customer

From the two datasets queries above, we conclude that customer **78afa995795e4d85b5d9ceeca43f5fef** received offer **9b98b8c7a33c4b65b9aebfe6a799e6d9** on day 1, viewed the offer on day 2 and conducted a transaction worth 19.89 on day 3 which met the offer difficulty. This led to offer completion on day 23 and a reward worth 5 units were credit to the customer. Note that this offer will be considered as desirable only if the offer is completed within the offer validity period.

3. Data Exploration

In this section, we perform exploration on the datasets above to analyze the below three broader themes:

- Customer activity over the test period
- Total customer sales over the test period
- Relation between customer demographics and offers rolled

- What is the relative number of offers received vs offer viewed vs offer completed for each of the three offer types

Customer activity over the test period

The below plots show the aggregate daily and monthly consumer activity in terms of number of customers buying Starbucks products over the test period and the sales amount associated with those purchases. We conclude from the plots below that the aggregate sale amounts and customer visits increased till the third month and dipped slightly for the fourth month. Interestingly, average sale amounts across the four months are evenly distributed implying that the average sale per transaction per customer was higher in the fourth month.

```
Text(0.5, 0.98, 'Total Sales Amount over Test Perios')
```

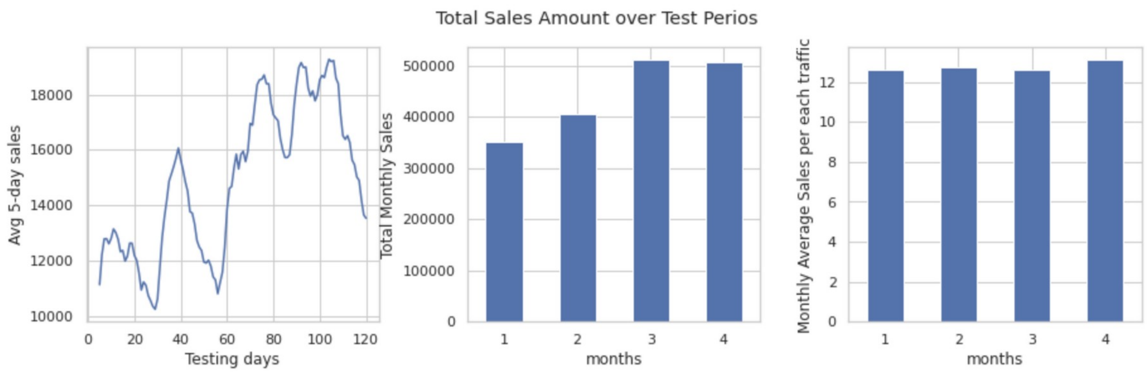


Fig 8: Customer Sale Trends at Starbucks

```
Text(0.5, 0.98, 'Total Sales Amount over Test Perios')
```

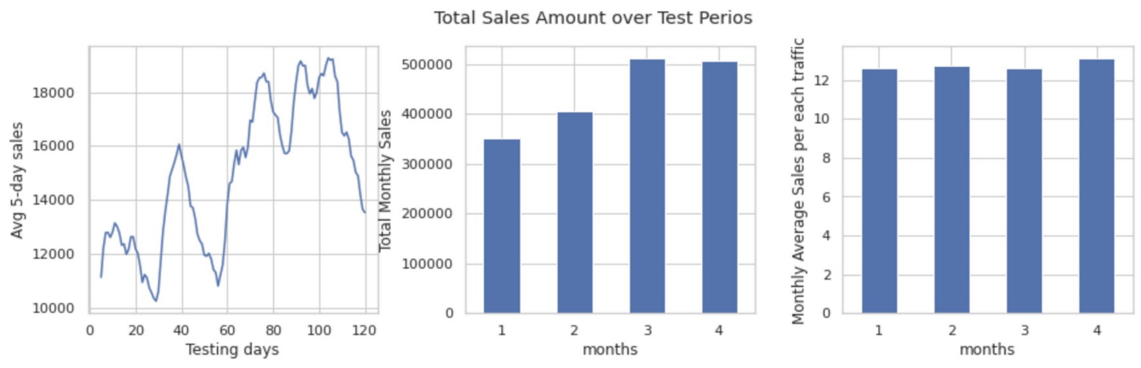


Fig 9: Aggregate Sales Amount Trends at Starbucks

Relation between customer demographics and offers rolled

Here we investigate if there is a certain bias in the way the offers have been rolled out. An example of this that only informational offers are rolled to customers in low-income groups whereas BOGO offers are rolled to customers in high income groups. Presence of such a bias will indicate that Starbucks is using some income-based heuristic in sending out offers to customers. Another bias could be in terms of gender.

```
Text(0.5, 1.0, 'Informational')
```

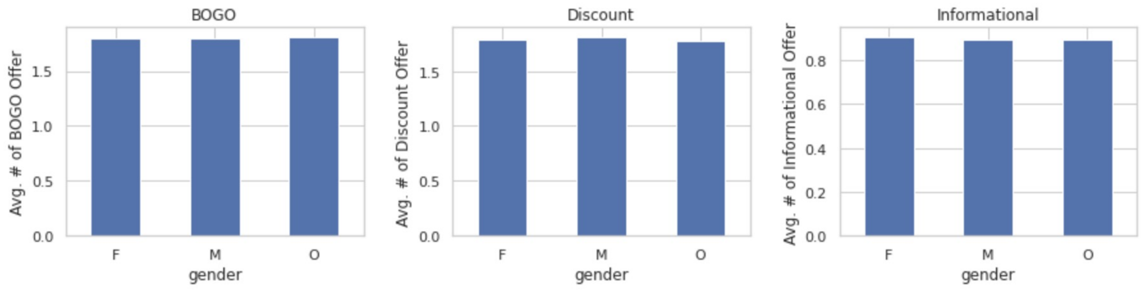
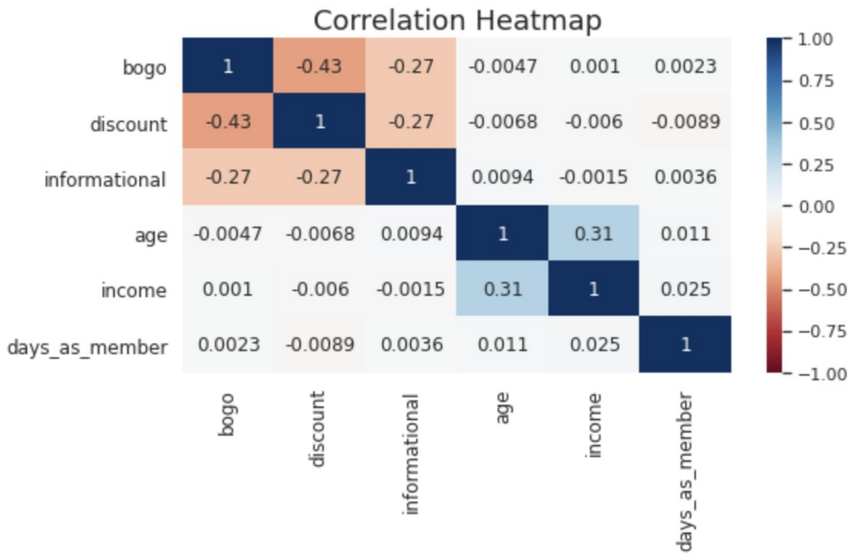


Fig 10: Average offers received for different gender types



	bogo	discount	informational	age	income	days_as_member
bogo	1	-0.43	-0.27	-0.0047	0.001	0.0023
discount	-0.43	1	-0.27	-0.0068	-0.006	-0.0089
informational	-0.27	-0.27	1	0.0094	-0.0015	0.0036
age	-0.0047	-0.0068	0.0094	1	0.31	0.011
income	0.001	-0.006	-0.0015	0.31	1	0.025
days_as_member	0.0023	-0.0089	0.0036	0.011	0.025	1

Fig 11: Correlation Matrix for offer demographics

From the two plots above, we conclude that the average number of offers rolled is similar for all the three gender types. Also, the number of offers rolled is uncorrelated with age, income and "**days_as_member**" attributes of the underlying population.

Offer Events Comparison

In this section, we investigate what kind of offers are viewed the most and what fraction of viewed offers successfully get completed.

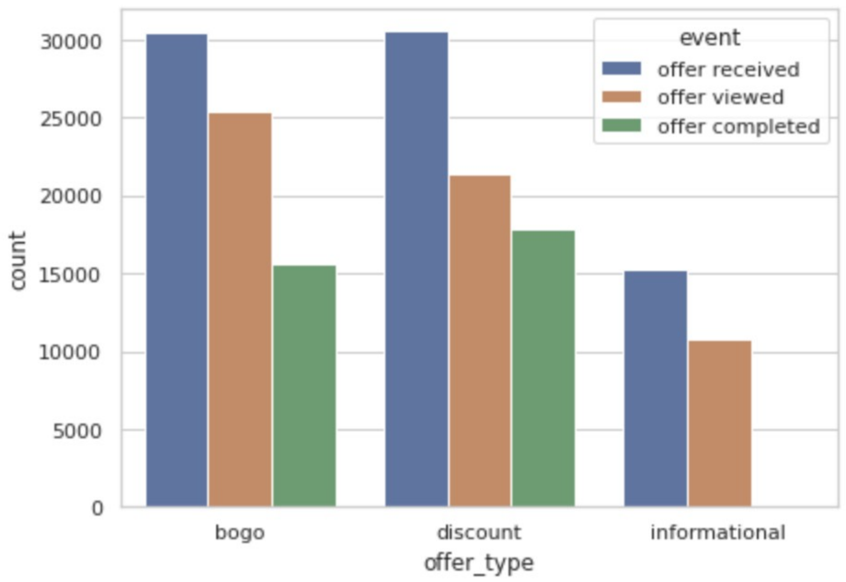


Fig 12: Completion Rates for Offer Types

Finally, from the two plots above, we conclude that BOGO and discount offers are rolled in same number for each of the four test months. The number of discount offers rolled are always lower than BOGO/informational offers.

A higher proportion of BOGO offers are viewed than discount offers. However, more fraction of discount offers is completed. This implies that more viewings of discount offer get converted into offer completion. Note that there is no information on offer completed for informational offers.

4. Feature Engineering

In this section, we create two new features in our dataset:

- Desirability of offers – to understand whether a particular offer rolled is desirable or not
- Recency, Frequency and Monetary columns – to quantify the customer value for different customers

Desirability of offers

In this section we classify offers into one of the two categories namely, “desirable” or “non-desirable”. Offers will be classified as “desirable” **only if** one of the following holds good:

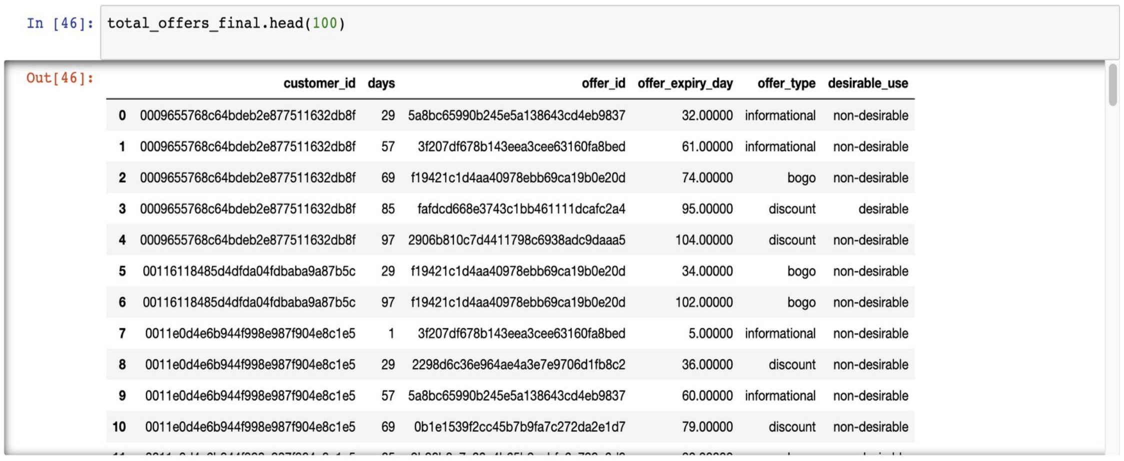
- If the customer receives the offer, then views the offer then conducts a transaction sufficient to complete the offer all within the offer validity period
- If the customer receives the offer, then views the offer then conducts a transaction within the validity period which is insufficient to meet the offer difficulty can still be considered a desirable offer since we can conclude that the transaction was performed under the influence of the offer

In our implementation, we are even including the offers that could not be completed due to not meeting the minimum transaction thresholds (I.e., offer difficulty) but still after viewing these offers, the customer did buy certain products which can help us conclude that the purchase was done under the influence of the offer. In this case, rolling the offer is leading to an increase in Starbucks sales thereby making it favorable. The task for us at hand then becomes classification of offers into of the above two categories in the data dataset. Note that the analysis is done separately for different classes.

Using the Transcript Dataset, we follow the below steps to obtain the desirable/non-desirable categories.

Step Number	Input	Transformation
1.	Transcript Dataset	Data Wrangling • Convert the Event Type in Binary dummies • Fetch the offer Information from Portfolio dataset • Compute the offer expiry day as offer received day plus offer validity period • Multiply the binary dummies for offer viewings, offer reception and offer completion by their respective dates Data Cleaning • Remove unwarranted columns like event, months, etc.
2.	Dataset from Step 1	Data Wrangling • Pivot the input dataset on "**customer_id**", "**offer_id**" and "**day**", the events to analyze will be the offer expiry day, offer and transaction events (i.e., if it is offer received, offer viewed, offer completed or transaction conducted) - This pivot tells us for each offer rolled on each date, what was the offer expiry date and customer activity?
3.	Dataset from Step 2	*Here we perform the decision if offer events happened prior to the offer expiry date.* We filter out the records from input in this step where the offer related events happen before or on the offer expiry date. Note that any record for which the offer related event happens after the offer expiry period is automatically filtered out in this step. The implementation of this condition is explained in the attached Jupyter Notebook. This filtered dataset is the output of this step.
4.	Dataset from Step 3	*Here we classify the records in input dataset as desirable/non-desirable* • We first pivot the input dataset on "**customer_id**", "**offer_id**" and "**offer expiry day**" • We then filter out the records for which "**offer_viewed**" and "**transaction**" columns are zero • This leaves us with offer records which were viewed by the customers, and for which the customers had conducted a transaction all within the offer expiry period – This group would constitute all the desirable offers according to our definition • Finally, we mark all these rows as desirable
5.	Dataset from Step 4	We merge the dataset from step 4 above with all the received offer record on "**customer_id**", "**offer_id**" and "**offer expiry day**" marking the desirable records on this level as "**desirable**" and others as "**non-desirable**"

Below we show a small representation of the output of this step and sample a record for our undestanding



```
In [46]: total_offers_final.head(100)
```

Out[46]:

	customer_id	days	offer_id	offer_expiry_day	offer_type	desirable_use
0	0009655768c64bdeb2e877511632db8f	29	5a8bc65990b245e5a138643cd4eb9837	32.00000	informational	non-desirable
1	0009655768c64bdeb2e877511632db8f	57	3f207df678b143eea3cee63160fa8bed	61.00000	informational	non-desirable
2	0009655768c64bdeb2e877511632db8f	69	f19421c1d4aa40978ebb69ca19b0e20d	74.00000	bogo	non-desirable
3	0009655768c64bdeb2e877511632db8f	85	fafdcd668e3743c1bb461111dcafc2a4	95.00000	discount	desirable
4	0009655768c64bdeb2e877511632db8f	97	2906b810c7d4411798c6938adc9daaa5	104.00000	discount	non-desirable
5	00116118485d4dfda04fdbaba9a87b5c	29	f19421c1d4aa40978ebb69ca19b0e20d	34.00000	bogo	non-desirable
6	00116118485d4dfda04fdbaba9a87b5c	97	f19421c1d4aa40978ebb69ca19b0e20d	102.00000	bogo	non-desirable
7	0011e0d4e6b944f998e987f904e8c1e5	1	3f207df678b143eea3cee63160fa8bed	5.00000	informational	non-desirable
8	0011e0d4e6b944f998e987f904e8c1e5	29	2298d6c36e964ae4a3e7e9706d1fb8c2	36.00000	discount	non-desirable
9	0011e0d4e6b944f998e987f904e8c1e5	57	5a8bc65990b245e5a138643cd4eb9837	60.00000	informational	non-desirable
10	0011e0d4e6b944f998e987f904e8c1e5	69	0b1e1539f2cc45b7b9fa7c272da2e1d7	79.00000	discount	non-desirable

Fig 13: Dataset with "**desirable_use**" feature

```
<AxesSubplot:xlabel='offer_type', ylabel='count'>
```

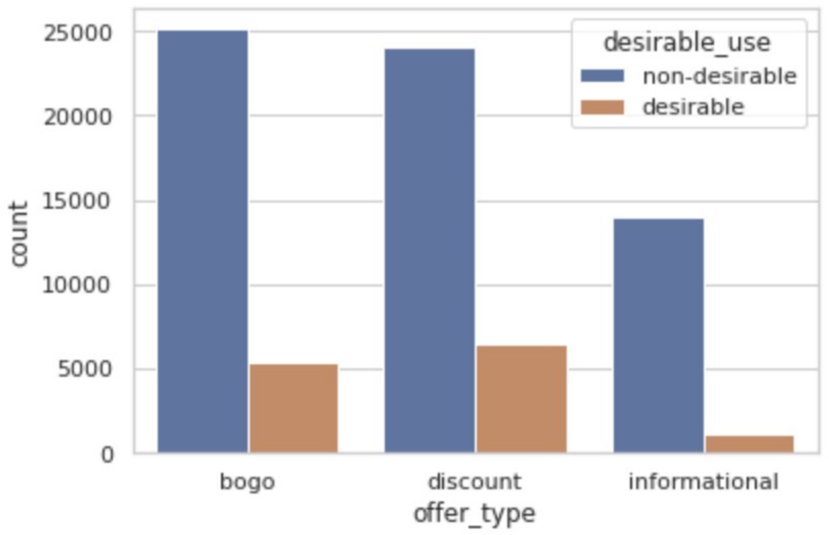


Fig 14: Desirable offers vs offer type

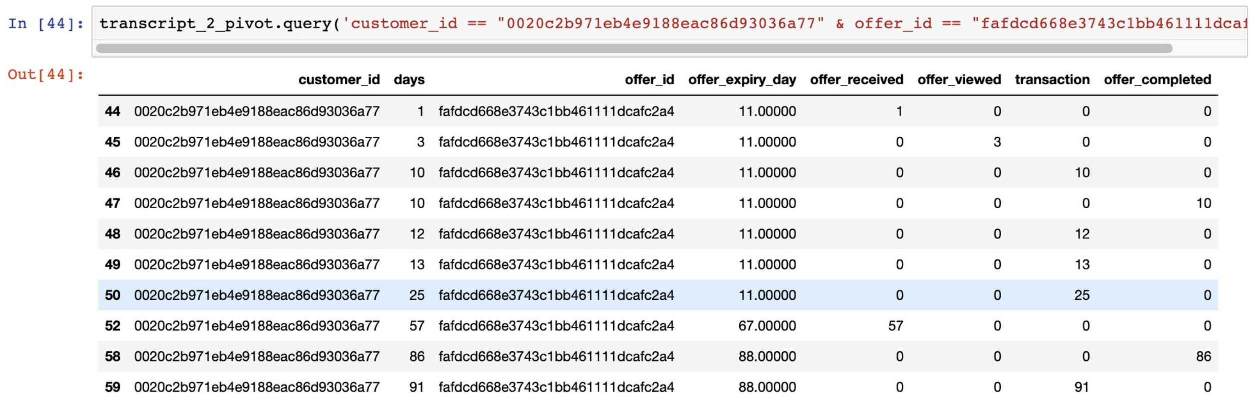
We see from the count plot above that the number of desirable offers is much lower than the total number of offers received for each of the three offer types. This is especially true for informational offers where we only see 7% of the informational offers rolled as desirable.

This could potentially dictate the need for data processing before model training for classification task. This is because the target class is imbalanced and splitting the test set from an imbalanced training dataset can only worsen the problem. This has been discussed in detail in the Modelling Section.

From the "**total_offer_final**" dataset shown above we sample a customer and offer id which has been marked desirable in our final dataset

customer_id: 0020c2b971eb4e9188eac86d93036a77

offer_id: fafdcd668e3743c1bb461111dcafc2a4



```
In [44]: transcript_2_pivot.query('customer_id == "0020c2b971eb4e9188eac86d93036a77" & offer_id == "fafdcd668e3743c1bb461111dcaf
```

Out[44]:

	customer_id	days	offer_id	offer_expiry_day	offer_received	offer_viewed	transaction	offer_completed
44	0020c2b971eb4e9188eac86d93036a77	1	fafdcd668e3743c1bb461111dcafc2a4	11.00000	1	0	0	0
45	0020c2b971eb4e9188eac86d93036a77	3	fafdcd668e3743c1bb461111dcafc2a4	11.00000	0	3	0	0
46	0020c2b971eb4e9188eac86d93036a77	10	fafdcd668e3743c1bb461111dcafc2a4	11.00000	0	0	10	0
47	0020c2b971eb4e9188eac86d93036a77	10	fafdcd668e3743c1bb461111dcafc2a4	11.00000	0	0	0	10
48	0020c2b971eb4e9188eac86d93036a77	12	fafdcd668e3743c1bb461111dcafc2a4	11.00000	0	0	12	0
49	0020c2b971eb4e9188eac86d93036a77	13	fafdcd668e3743c1bb461111dcafc2a4	11.00000	0	0	13	0
50	0020c2b971eb4e9188eac86d93036a77	25	fafdcd668e3743c1bb461111dcafc2a4	11.00000	0	0	25	0
52	0020c2b971eb4e9188eac86d93036a77	57	fafdcd668e3743c1bb461111dcafc2a4	67.00000	57	0	0	0
58	0020c2b971eb4e9188eac86d93036a77	86	fafdcd668e3743c1bb461111dcafc2a4	88.00000	0	0	0	86
59	0020c2b971eb4e9188eac86d93036a77	91	fafdcd668e3743c1bb461111dcafc2a4	88.00000	0	0	91	0

Fig 15: Summary of all events for the sampled customer and offer

From the example above, we see that customer **0020c2b971eb4e9188eac86d93036a77** received offer **fafdcd668e3743c1bb461111dcafc2a4** on day 1, viewed the offer on day 3 and conducted the transaction on day 10. The offer expiry is on day 11. As all the relevant events happened prior to offer expiry, we can classify this as a desirable event.

Quantification of Customer Value

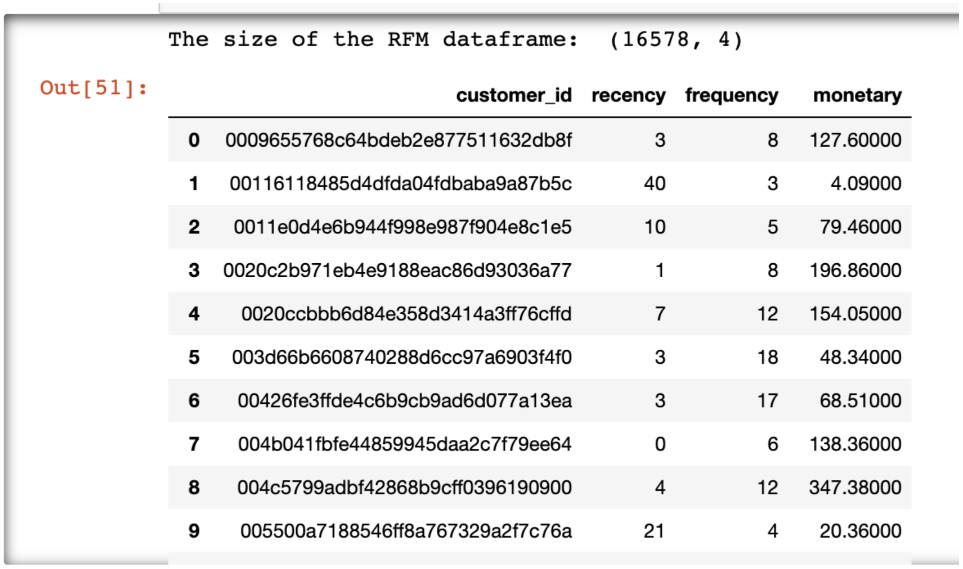
In this section, we explore the customer buying patterns using RFM Analysis. RFM refers to Recency, Frequency and Monetary values. This method analyses different customers on three major dimensions:

- How recently did the customer purchase?
- How often does the customer purchase?
- How much does the customer spend?

RFM method is commonly used for analyzing customer value. It is often used in database marketing in retail and professional service industries.

In our use-case, we insert three more attributes for each customer which helps answer the three questions outlined above

- Recency - total test duration (about 4 months) minus maximum date when a purchase happens
- Frequency is computed as the total number of times a transaction has been performed by the customer
- Monetary is computed as the aggregate sales amount done by the customer over the test period



```
The size of the RFM dataframe:  (16578, 4)
```

Out[51]:

	customer_id	recency	frequency	monetary
0	0009655768c64bdeb2e877511632db8f	3	8	127.60000
1	00116118485d4dfda04fdbaba9a87b5c	40	3	4.09000
2	0011e0d4e6b944f998e987f904e8c1e5	10	5	79.46000
3	0020c2b971eb4e9188eac86d93036a77	1	8	196.86000
4	0020ccbbb6d84e358d3414a3ff76cffd	7	12	154.05000
5	003d66b6608740288d6cc97a6903f4f0	3	18	48.34000
6	00426fe3ffde4c6b9cb9ad6d077a13ea	3	17	68.51000
7	004b041fbfe44859945daa2c7f79ee64	0	6	138.36000
8	004c5799adbf42868b9cff0396190900	4	12	347.38000
9	005500a7188546ff8a767329a2f7c76a	21	4	20.36000

Fig 16: Recency, Frequency and Monetary values per customer

Below is a correlation matrix for recency, frequency, and monetary values over the customer population, we see that recency is negatively correlated with frequency as expected since customers that have recently visited the shops (low recency) are likely to be frequent visitors (high frequency). Also, more frequent buyers are likely to spend higher amounts in aggregate thereby making frequency positively correlated with monetary.

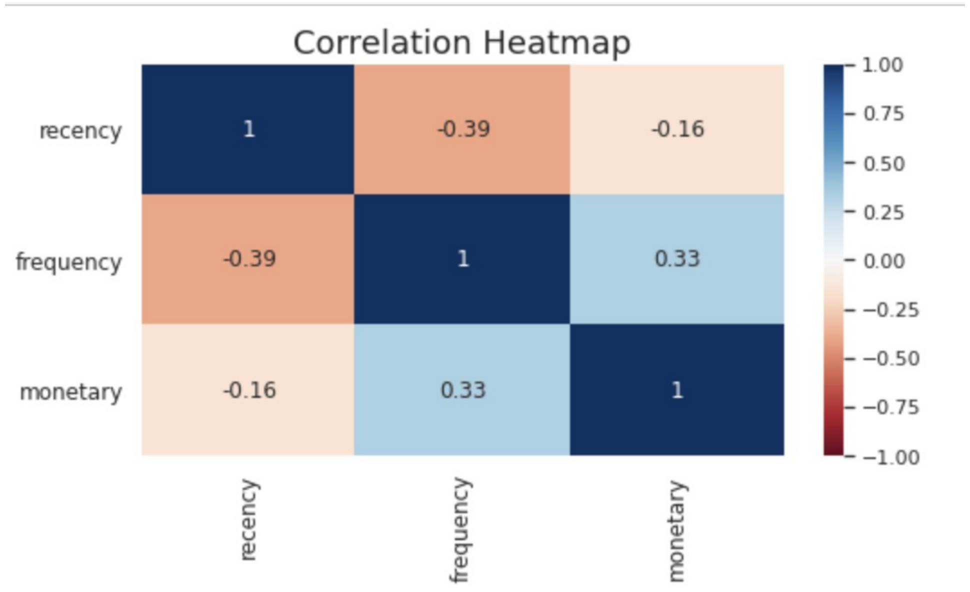


Fig 17: Correlation values for RFM Matrix

5. Model Set-up and Performance Evaluation

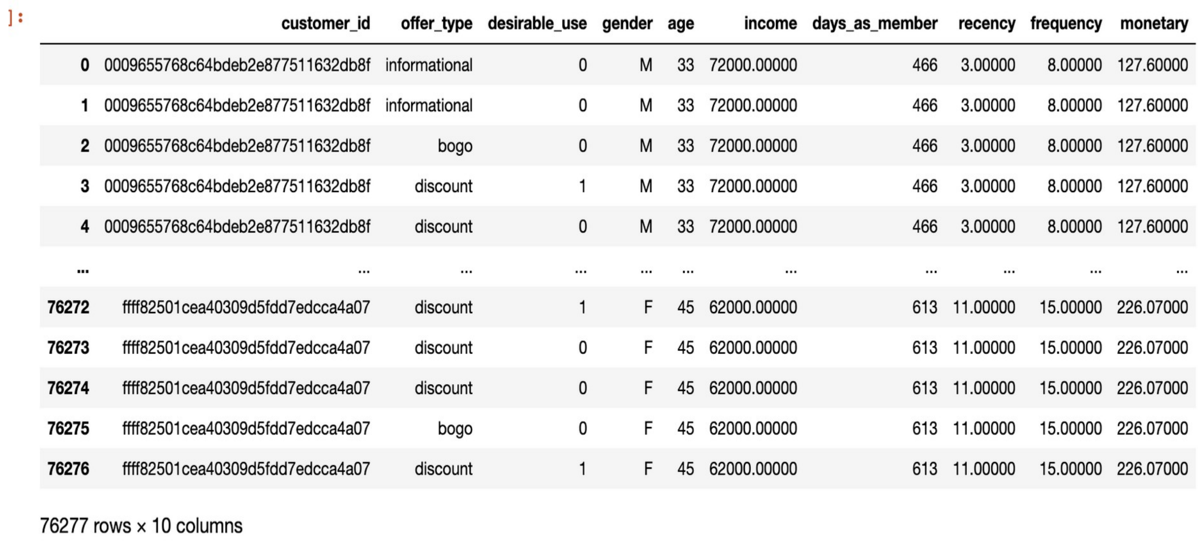
In this section, we perform the Modelling using the following key Steps

Step Number	Process
1	Splitting of Training Data into Train, Test and Validation Datasets
2	Setting up Benchmark Model (Logistic regression) for Classification Problem
3	Setting up of SageMaker XGBoost Model: ○ Hyperparameter Tuning ○ Model Training on Training Dataset ○ Batch Transform using predictor object on Test Dataset
4	Model performance evaluation and reasoning
5	Adjustment of Model Inputs e.g., removing class imbalance
6	Re-run of Steps 2, 3 and 4
7	Set-up and prediction using SageMaker Linear Learner Model
7	Model performance assessment and inference

Setting up of Input Data

We take the two sets of output data from Feature engineering section and merge on customer ID to create the final input dataset as shown below (the "**desirable_use**" column values have been one-hot encoded. Also, converted the gender attribute into binary dummies.

Below is a snapshot of the input dataset that we will use in rest of the classification task



]:

	customer_id	offer_type	desirable_use	gender	age	income	days_as_member	recency	frequency	monetary
0	0009655768c64bdeb2e877511632db8f	informational	0	M	33	72000.00000	466	3.00000	8.00000	127.60000
1	0009655768c64bdeb2e877511632db8f	informational	0	M	33	72000.00000	466	3.00000	8.00000	127.60000
2	0009655768c64bdeb2e877511632db8f	bogo	0	M	33	72000.00000	466	3.00000	8.00000	127.60000
3	0009655768c64bdeb2e877511632db8f	discount	1	M	33	72000.00000	466	3.00000	8.00000	127.60000
4	0009655768c64bdeb2e877511632db8f	discount	0	M	33	72000.00000	466	3.00000	8.00000	127.60000
...	...	...	...	...	...	...	...	...	...	...
76272	ffff82501cea40309d5fdd7edcca4a07	discount	1	F	45	62000.00000	613	11.00000	15.00000	226.07000
76273	ffff82501cea40309d5fdd7edcca4a07	discount	0	F	45	62000.00000	613	11.00000	15.00000	226.07000
76274	ffff82501cea40309d5fdd7edcca4a07	discount	0	F	45	62000.00000	613	11.00000	15.00000	226.07000
76275	ffff82501cea40309d5fdd7edcca4a07	bogo	0	F	45	62000.00000	613	11.00000	15.00000	226.07000
76276	ffff82501cea40309d5fdd7edcca4a07	discount	1	F	45	62000.00000	613	11.00000	15.00000	226.07000

76277 rows × 10 columns

Fig 18: Input Dataset to use in the classification task

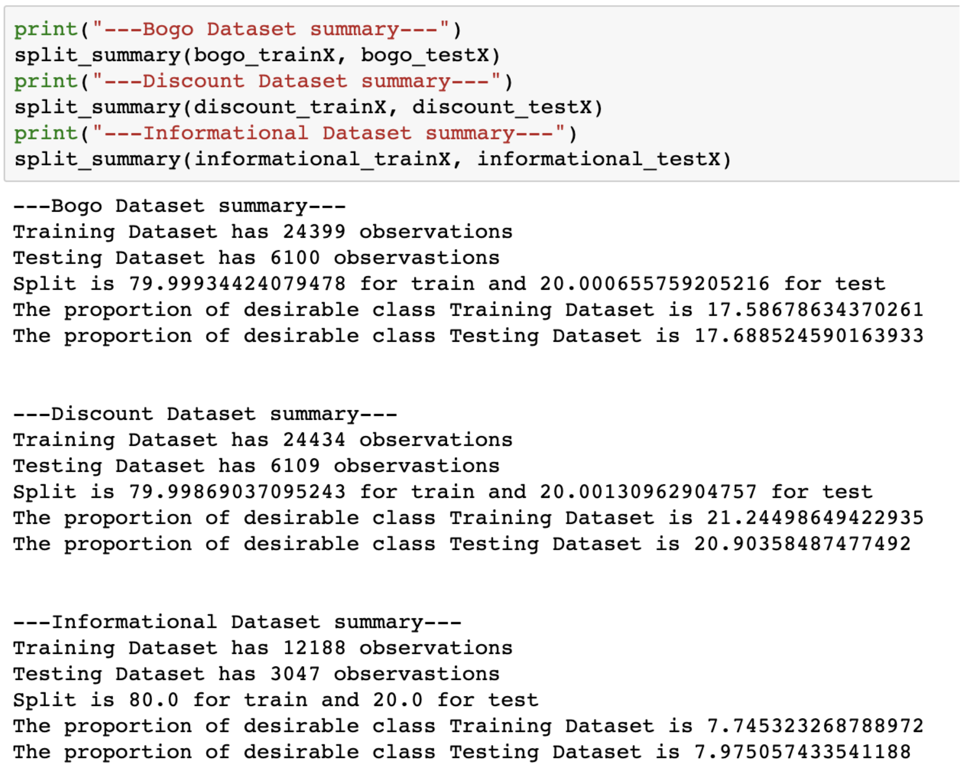
Note that the dataset above covers all three offer types but we will be making predictions separately for each offer type. Hence, we split the dataset above into three separate input datasets with offer_type equal to BOGO/discount/informational.

Step 1: Splitting of Training Data

We use the **sklearn** Library to perform the train test split functionality. The Test size is 20% and Train size is 80%. We train our models on Training Dataset and test it on the Test Dataset. Note that for XGBoost & Linear Learner Implementation, we further split the Training Dataset into 90% Training and 10% Validation Dataset.

Below is compositional summary for the split of Training and Testing Data. We see that both the Training and Testing Data is equivalently distributed for each offer type although there is a noticeable class imbalance in the sense that the percentage of positive class is significantly lower. The imbalance problem is especially significant in informational datasets.

Presence of imbalance trains the model to predict negative outcomes more frequently than positive outcomes, while this may not significantly hurt the accuracy of the model (since test dataset will also have significant negative outcomes) but could hurt the Recall metric since a less proportion of positives would be predicted correctly. We are likely to see a similar trend in Precision (although to a lesser extent).



```
print("---Bogo Dataset summary---")
split_summary(bogo_trainX, bogo_testX)
print("---Discount Dataset summary---")
split_summary(discount_trainX, discount_testX)
print("---Informational Dataset summary---")
split_summary(informational_trainX, informational_testX)
```

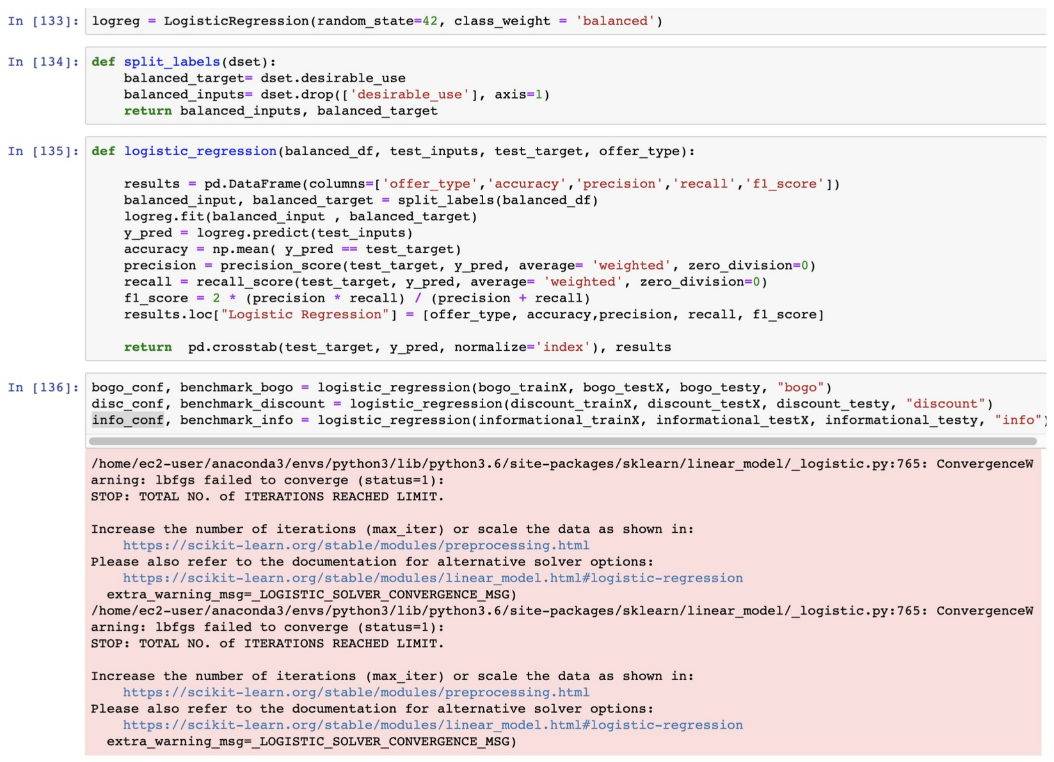
```
---Bogo Dataset summary---
Training Dataset has 24399 observations
Testing Dataset has 6100 observastions
Split is 79.99934424079478 for train and 20.000655759205216 for test
The proportion of desirable class Training Dataset is 17.58678634370261
The proportion of desirable class Testing Dataset is 17.688524590163933


---Discount Dataset summary---
Training Dataset has 24434 observations
Testing Dataset has 6109 observastions
Split is 79.99869037095243 for train and 20.00130962904757 for test
The proportion of desirable class Training Dataset is 21.24498649422935
The proportion of desirable class Testing Dataset is 20.90358487477492


---Informational Dataset summary---
Training Dataset has 12188 observations
Testing Dataset has 3047 observastions
Split is 80.0 for train and 20.0 for test
The proportion of desirable class Training Dataset is 7.745323268788972
The proportion of desirable class Testing Dataset is 7.975057433541188
```

Fig 19: 80%-20% Train Test Split

Step 2: Setting up of benchmark Logistic Regression Model

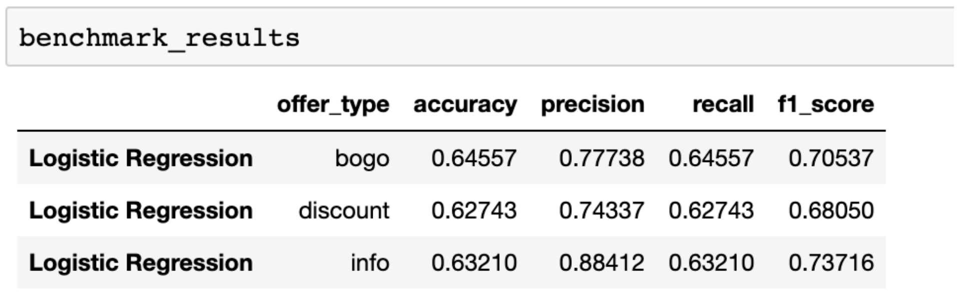
Logistic Regression is taken as a benchmark model for predicting the outcomes since this is a common starting point for a classification task. Logistic regression models the probabilities for classification problems with two outcomes. It is an extension of the linear regression model for classification problems.

Below is a snapshot of the Logistic Regression Model implemented and the results for each of the three offer types



```
In [133]: logreg = LogisticRegression(random_state=42, class_weight = 'balanced')

In [134]: def split_labels(dset):
              balanced_target= dset.desirable_use
              balanced_inputs= dset.drop(['desirable_use'], axis=1)
              return balanced_inputs, balanced_target

In [135]: def logistic_regression(balanced_df, test_inputs, test_target, offer_type):

              results = pd.DataFrame(columns=['offer_type','accuracy','precision','recall','f1_score'])
              balanced_input, balanced_target = split_labels(balanced_df)
              logreg.fit(balanced_input , balanced_target)
              y_pred = logreg.predict(test_inputs)
              accuracy = np.mean( y_pred == test_target)
              precision = precision_score(test_target, y_pred, average= 'weighted', zero_division=0)
              recall = recall_score(test_target, y_pred, average= 'weighted', zero_division=0)
              f1_score = 2 * (precision * recall) / (precision + recall)
              results.loc["Logistic Regression"] = [offer_type, accuracy,precision, recall, f1_score]

              return  pd.crosstab(test_target, y_pred, normalize='index'), results

In [136]: bogo_conf, benchmark_bogo = logistic_regression(bogo_trainX, bogo_testX, bogo_testy, "bogo")
          disc_conf, benchmark_discount = logistic_regression(discount_trainX, discount_testX, discount_testy, "discount")
          info_conf, benchmark_info = logistic_regression(informational_trainX, informational_testX, informational_testy, "info"
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/sklearn/linear_model/_logistic.py:765: ConvergenceW
arning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/sklearn/linear_model/_logistic.py:765: ConvergenceW
arning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

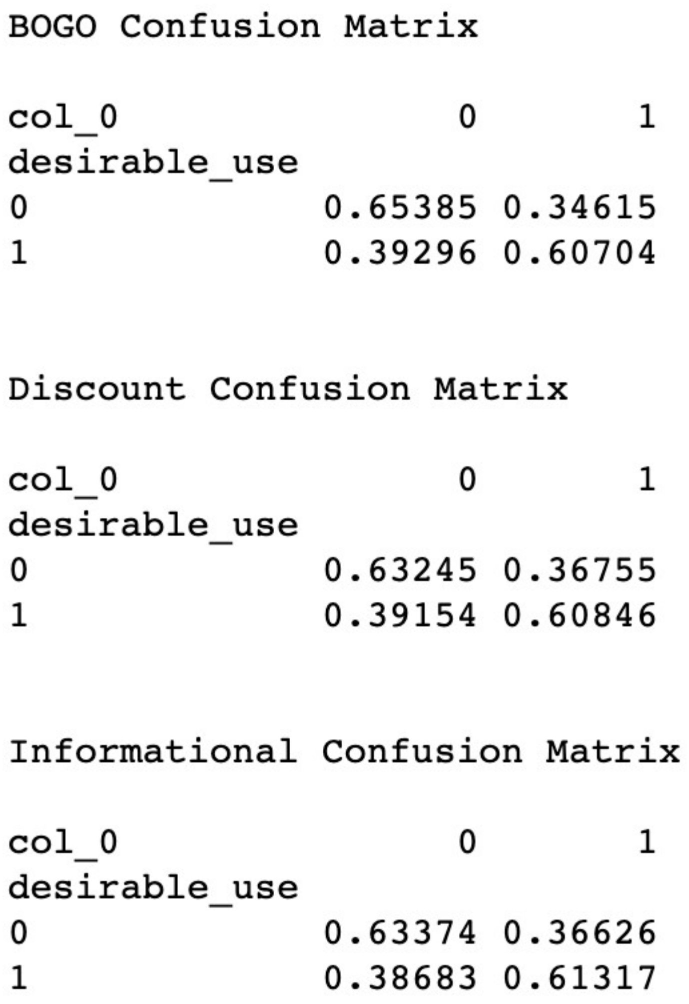


```
benchmark_results
```

	offer_type	accuracy	precision	recall	f1_score
Logistic Regression	bogo	0.64557	0.77738	0.64557	0.70537
Logistic Regression	discount	0.62743	0.74337	0.62743	0.68050
Logistic Regression	info	0.63210	0.88412	0.63210	0.73716

Fig 20: Benchmark Model Implementation Results

From the results above, we conclude that Benchmark Model performs well in terms of predicting the positive outcomes as suggested by a high recall score. Below we analyze the confusion matrices for each of the three offer type classes



```
BOGO Confusion Matrix

col_0                0       1
desirable_use
0              0.65385 0.34615
1              0.39296 0.60704


Discount Confusion Matrix

col_0                0       1
desirable_use
0              0.63245 0.36755
1              0.39154 0.60846


Informational Confusion Matrix

col_0                0       1
desirable_use
0              0.63374 0.36626
1              0.38683 0.61317
```

Fig 21: Confusion Matrix for Benchmark Model

On average, we see that the benchmark model can correctly identify 60% of the positive ground truth labels and 65% of the negative ground truth labels for BOGO Offers and close to 63% of the negative ground truth labels for the Discount and Informational Offer Types.

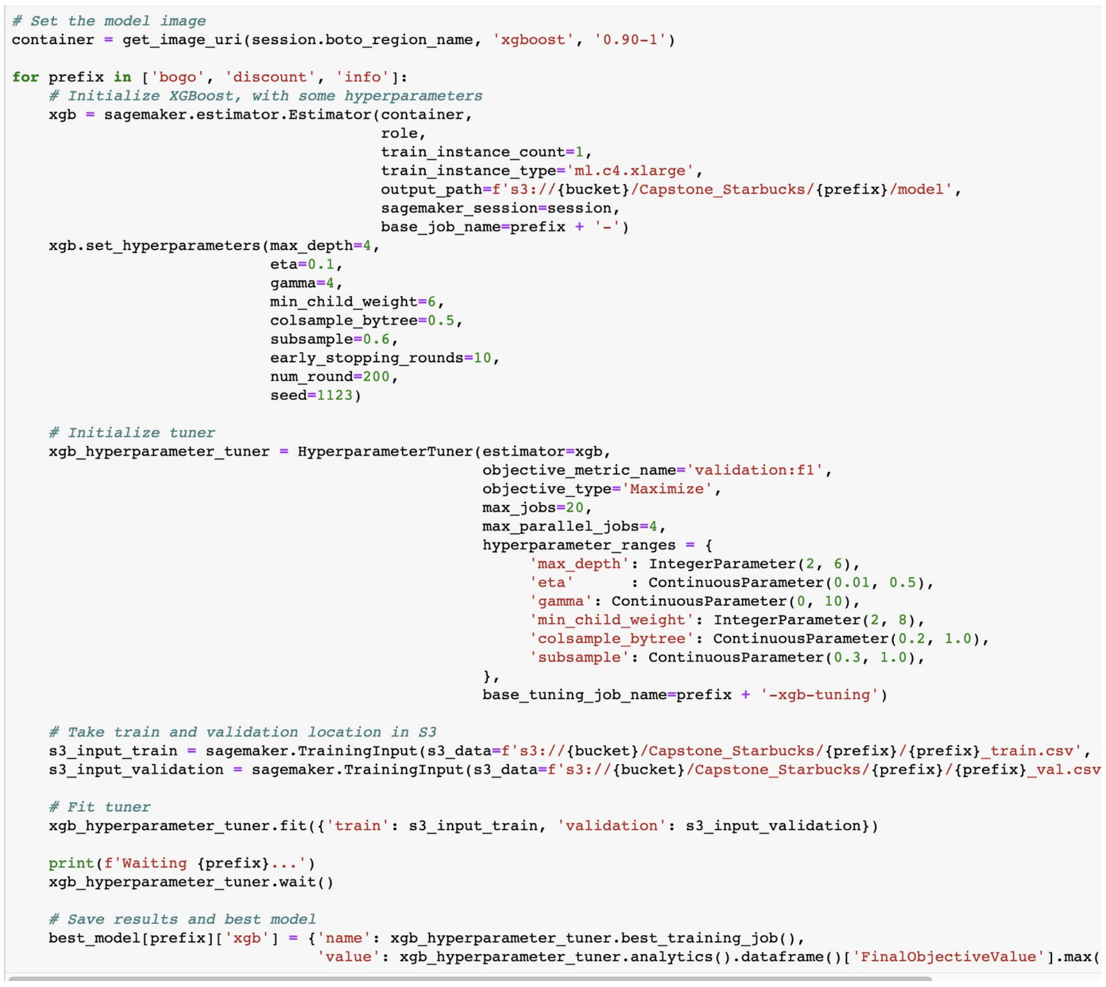
We conclude that model performs well on positive ground truths especially considering that this is imbalanced dataset with fewer positive labels. However, for more frequent negative ground truths (non-desirables) we see that the model performance is not as great since not more than 65% of the outcomes are correctly predicted. This sets up a nice benchmark for us to compare the remaining two models namely the SageMaker XGBoost and SageMaker Linear Learner.

<u>Step 3: Setting up of SageMaker XGBoost Model</u>

XGBoost Model has dominated predictions tasks recently for structured and Tabular Data. It has low model latency and high execution speed. It works based on Gradient Boosted Decision Tree Algorithm which generates predictions using an ensemble of weak prediction models like decision trees. Like other boosting algorithms, additive models are bult in a stage wise way.

We first begin the setup by Tuning the Hyperparameters for our model over the training and validation datasets. We first define a base estimator with default values for each parameter. Then we set a hyperparameter tuner object which specifies the range of hyperparameters to tune from. The decision criterion is set to maximization of F1 Score. This helps us chose hyperparameters with good balance of precision and recall yielding high F1 scores.

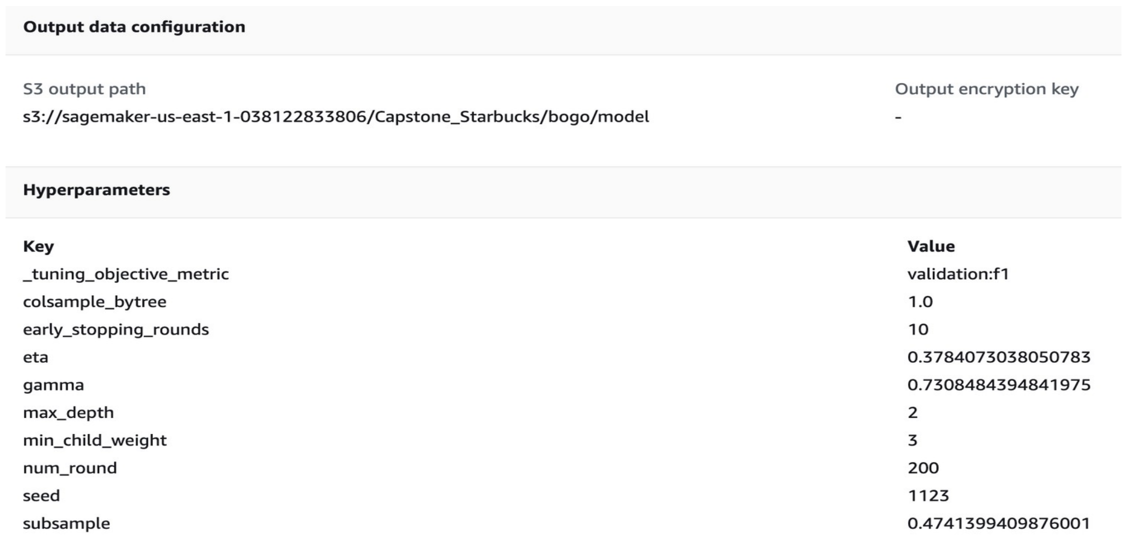
The tunable hyperparameters which we have chosen are “**max_depth**”, “**eta**”, “**gamma**”, “**min_child_weight**”,”**colsample_bytree**”, “**subsample**”



```
# Set the model image
container = get_image_uri(session.boto_region_name, 'xgboost', '0.90-1')

for prefix in ['bogo', 'discount', 'info']:
    # Initialize XGBoost, with some hyperparameters
    xgb = sagemaker.estimator.Estimator(container,
                                        role,
                                        train_instance_count=1,
                                        train_instance_type='ml.c4.xlarge',
                                        output_path=f's3://{bucket}/Capstone_Starbucks/{prefix}/model',
                                        sagemaker_session=session,
                                        base_job_name=prefix + '-')
    xgb.set_hyperparameters(max_depth=4,
                            eta=0.1,
                            gamma=4,
                            min_child_weight=6,
                            colsample_bytree=0.5,
                            subsample=0.6,
                            early_stopping_rounds=10,
                            num_round=200,
                            seed=1123)

    # Initialize tuner
    xgb_hyperparameter_tuner = HyperparameterTuner(estimator=xgb,
                                                   objective_metric_name='validation:f1',
                                                   objective_type='Maximize',
                                                   max_jobs=20,
                                                   max_parallel_jobs=4,
                                                   hyperparameter_ranges = {
                                                        'max_depth': IntegerParameter(2, 6),
                                                        'eta'      : ContinuousParameter(0.01, 0.5),
                                                        'gamma': ContinuousParameter(0, 10),
                                                        'min_child_weight': IntegerParameter(2, 8),
                                                        'colsample_bytree': ContinuousParameter(0.2, 1.0),
                                                        'subsample': ContinuousParameter(0.3, 1.0),
                                                   },
                                                   base_tuning_job_name=prefix + '-xgb-tuning')

    # Take train and validation location in S3
    s3_input_train = sagemaker.TrainingInput(s3_data=f's3://{bucket}/Capstone_Starbucks/{prefix}/{prefix}_train.csv',
    s3_input_validation = sagemaker.TrainingInput(s3_data=f's3://{bucket}/Capstone_Starbucks/{prefix}/{prefix}_val.csv

    # Fit tuner
    xgb_hyperparameter_tuner.fit({'train': s3_input_train, 'validation': s3_input_validation})

    print(f'Waiting {prefix}...')
    xgb_hyperparameter_tuner.wait()

    # Save results and best model
    best_model[prefix]['xgb'] = {'name': xgb_hyperparameter_tuner.best_training_job(),
                                 'value': xgb_hyperparameter_tuner.analytics().dataframe()['FinalObjectiveValue'].max(
```

Fig 22: Implementation of Hyperparameters

After the Hyperparameter Tuning we have the following optimal set of Hyperparameters for Training the model

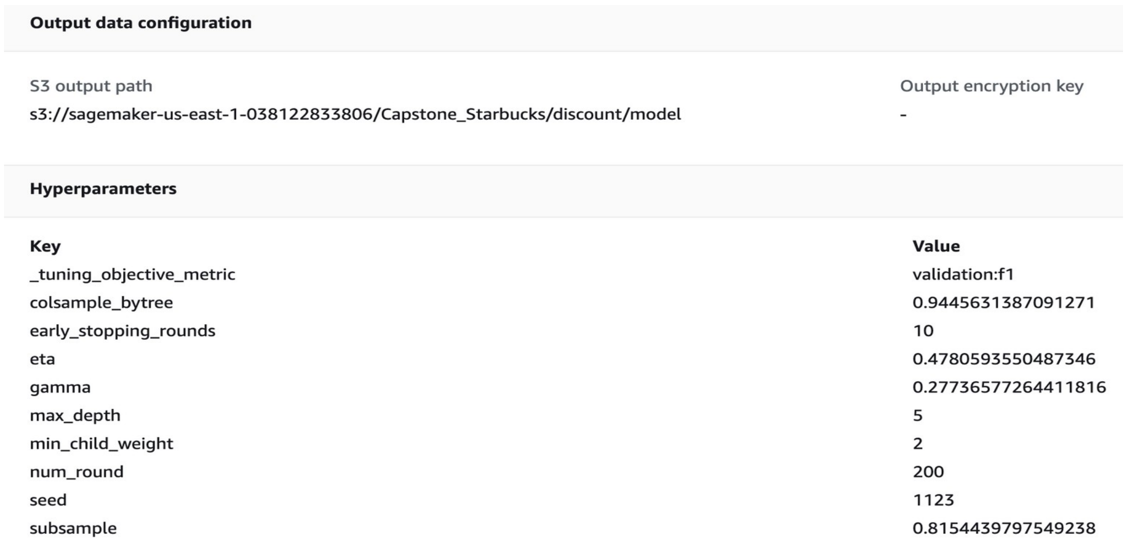


Output data configuration

S3 output path	Output encryption key
s3://sagemaker-us-east-1-038122833806/Capstone_Starbucks/bogo/model	-

Hyperparameters

Key	Value
_tuning_objective_metric	validation:f1
colsample_bytree	1.0
early_stopping_rounds	10
eta	0.3784073038050783
gamma	0.7308484394841975
max_depth	2
min_child_weight	3
num_round	200
seed	1123
subsample	0.4741399409876001

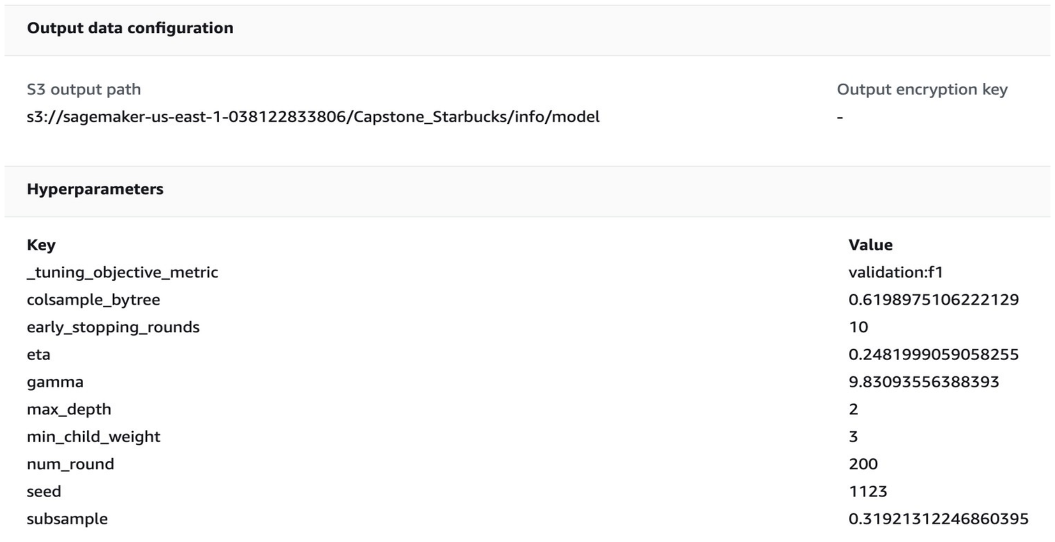


Output data configuration

S3 output path	Output encryption key
s3://sagemaker-us-east-1-038122833806/Capstone_Starbucks/discount/model	-

Hyperparameters

Key	Value
_tuning_objective_metric	validation:f1
colsample_bytree	0.9445631387091271
early_stopping_rounds	10
eta	0.4780593550487346
gamma	0.27736577264411816
max_depth	5
min_child_weight	2
num_round	200
seed	1123
subsample	0.8154439797549238



Output data configuration

S3 output path	Output encryption key
s3://sagemaker-us-east-1-038122833806/Capstone_Starbucks/info/model	-

Hyperparameters

Key	Value
_tuning_objective_metric	validation:f1
colsample_bytree	0.6198975106222129
early_stopping_rounds	10
eta	0.2481999059058255
gamma	9.83093556388393
max_depth	2
min_child_weight	3
num_round	200
seed	1123
subsample	0.31921312246860395

Fig 23: Optimal Hyperparameters for each offer type

Next, we train the XGBoost Model with these Hyperparameters for each offer-type and save the results to the s3 bucket. We then compare the results with ground truth values and compute the performance metrics and the confusion matrix for each of the predictions.

Confusion Matrix BOGO	Predicted Positive	Predicted Negative
True Positive	0	1
True Negative	0	1

Confusion Matrix Discount	Predicted Positive	Predicted Negative
True Positive	0.00861	0.99139
True Negative	0.00642	0.99358

Confusion Matrix Informational	Predicted Positive	Predicted Negative
True Positive	0	1
True Negative	0	1

Fig 24: Confusion Matrix for each offer type

Accuracy Metrics BOGO	Value
Accuracy	0.82311
Precision	0
Recall	0
F1	0

Accuracy Metrics Discount	Value
Accuracy	0.78769
Precision	0.26190
Recall	0.00861
F1	0.01668

Accuracy Metrics Informational	Value
Accuracy	0.92025
Precision	0
Recall	0
F1	0

Fig 25: Accuracy Metrics for each offer type

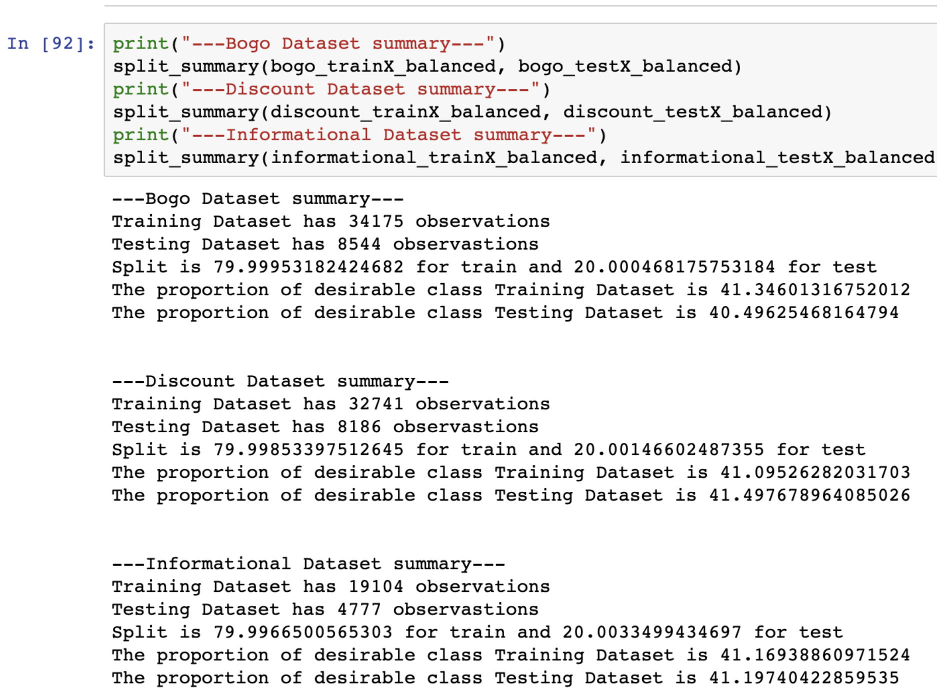
From the tables above, we conclude that the XGBoost Model developed has seriously low levels of precision and recall. This could be since we have used a highly imbalanced dataset for training and the model is biased toward classifying records as negative (this is certainly the case for BOGO and Informational offers where the model simply classifies all offers in Test Dataset as undesirable). We still end up with much higher accuracy scores in these cases since test dataset has majority of ground truth labels negative which the model is correctly able to classify.

Hence in the next step, we try to create a more balanced dataset and perform the same Benchmark Logistic Model re-run as well as XGBoost HPO + Model Training to generate the results

Step 4: Balancing the Input Dataset

In this section, we Balance the input dataset using oversampling using SMOTE Fucnctionalityin **sklearn**. As we do not have many records in the dataset (< 100000), we are not going for under sampling as it might result in missing important patterns for the model to learn during training.

Below we show the compositional distribution of the input dataset post the balancing exercise. We see that the balanced dataset has a fair 60%-40% distribution of positive and negative ground truth labels respectively.



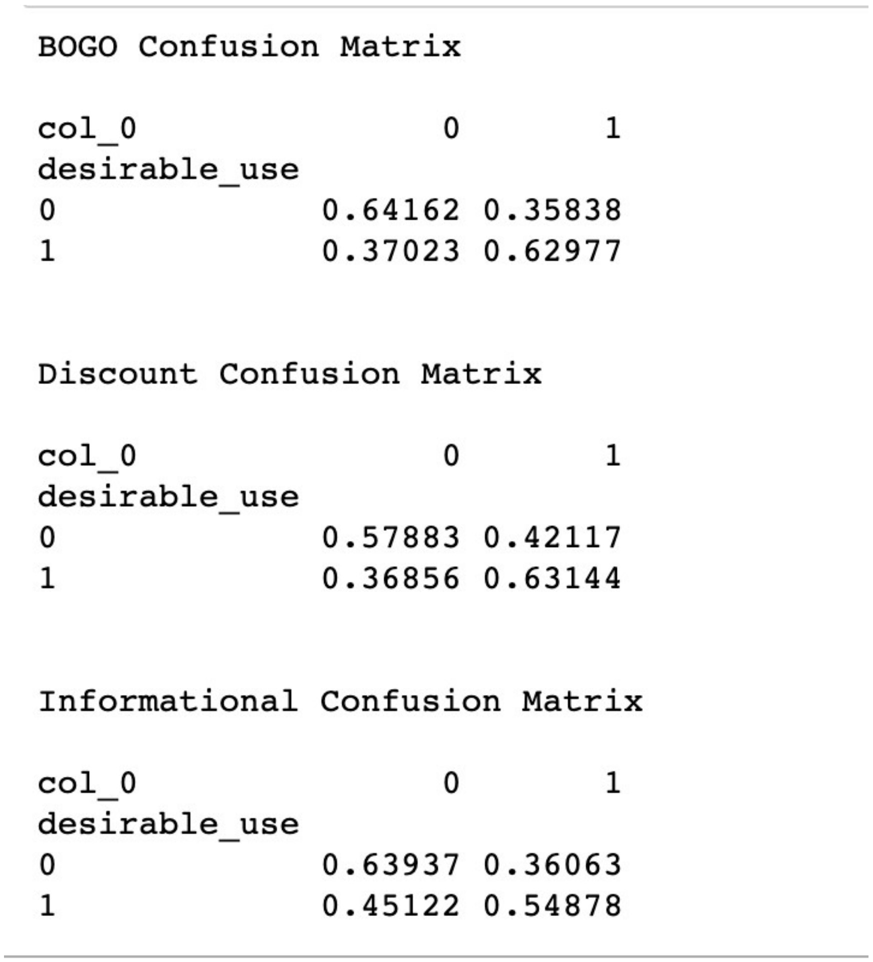
```
In [92]: print("---Bogo Dataset summary---")
         split_summary(bogo_trainX_balanced, bogo_testX_balanced)
         print("---Discount Dataset summary---")
         split_summary(discount_trainX_balanced, discount_testX_balanced)
         print("---Informational Dataset summary---")
         split_summary(informational_trainX_balanced, informational_testX_balanced
```

```
---Bogo Dataset summary---
Training Dataset has 34175 observations
Testing Dataset has 8544 observastions
Split is 79.99953182424682 for train and 20.000468175753184 for test
The proportion of desirable class Training Dataset is 41.34601316752012
The proportion of desirable class Testing Dataset is 40.49625468164794


---Discount Dataset summary---
Training Dataset has 32741 observations
Testing Dataset has 8186 observastions
Split is 79.99853397512645 for train and 20.00146602487355 for test
The proportion of desirable class Training Dataset is 41.09526282031703
The proportion of desirable class Testing Dataset is 41.497678964085026


---Informational Dataset summary---
Training Dataset has 19104 observations
Testing Dataset has 4777 observastions
Split is 79.9966500565303 for train and 20.0033499434697 for test
The proportion of desirable class Training Dataset is 41.16938860971524
The proportion of desirable class Testing Dataset is 41.19740422859535
```

Fig 26: Composition of ground truth labels post data-balancing

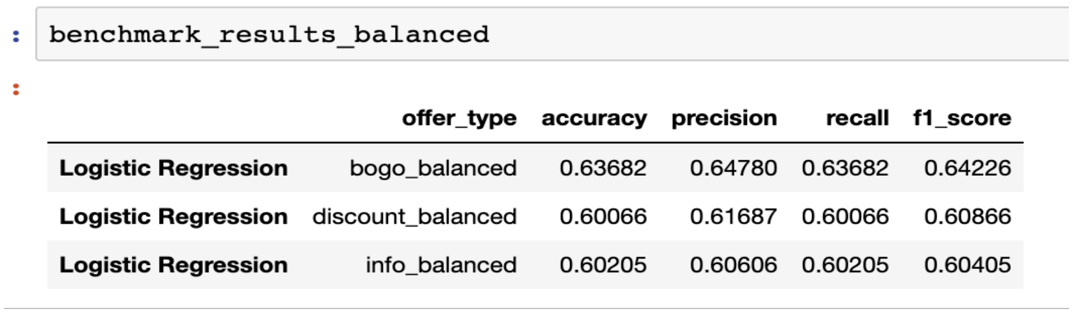
Step 5: Re-run of Benchmark and XGBoost Models

The re-run procedure is the same as outlined in steps 2, 3. Here we summarize the Benchmark and XGBoost model performance results.



```
BOGO Confusion Matrix

col_0                 0       1
desirable_use
0              0.64162 0.35838
1              0.37023 0.62977


Discount Confusion Matrix

col_0                 0       1
desirable_use
0              0.57883 0.42117
1              0.36856 0.63144


Informational Confusion Matrix

col_0                 0       1
desirable_use
0              0.63937 0.36063
1              0.45122 0.54878
```

Fig 27: Logistic Regression Confusion Matrix for Balanced Dataset



```
benchmark_results_balanced
```

	offer_type	accuracy	precision	recall	f1_score
Logistic Regression	bogo_balanced	0.63682	0.64780	0.63682	0.64226
Logistic Regression	discount_balanced	0.60066	0.61687	0.60066	0.60866
Logistic Regression	info_balanced	0.60205	0.60606	0.60205	0.60405

Fig 28: Logistic Regression performance on Balanced Dataset

Confusion Matrix BOGO	Predicted Positive	Predicted Negative
True Positive	0.41891	0.58109
True Negative	0.22824	0.77176

Confusion Matrix Discount	Predicted Positive	Predicted Negative
True Positive	0.63821	0.36179
True Negative	0.37562	0.62438

Confusion Matrix Informational	Predicted Positive	Predicted Negative
True Positive	0.92593	0.07407
True Negative	0.03424	0.96576

Fig 29: XGBoost Confusion Matrix on Balanced Dataset

Accuracy Metrics BOGO	Value
Accuracy	0.70934
Precision	0.28285
Recall	0.41891
F1	0.33769

Accuracy Metrics Discount	Value
Accuracy	0.62727
Precision	0.30989
Recall	0.63821
F1	0.41720

Accuracy Metrics Informational	Value
Accuracy	0.89465
Precision	0.15789
Recall	0.07407
F1	0.100084

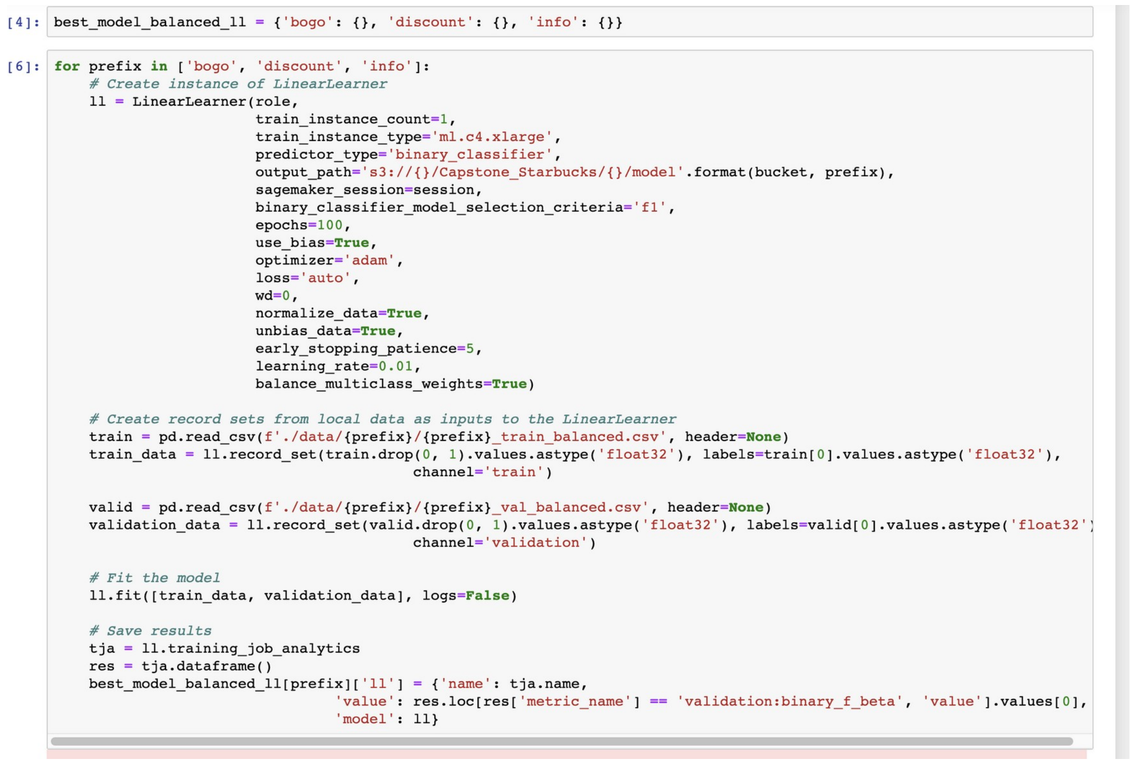
Fig 30: XGBoost Accuracy Metrics on Balanced Dataset

From the performance metrics above, we conclude that the XGBoost Model performs significantly well on the Balanced Dataset while there is a slight reduction in the performance of the Benchmark Logistic Regression Model.

We see for the Infromational Offer Types, the precision and recall values are lower since the model still is unable to classify the positive outcomes correctly. We try to set-up a Linear Model on the balanced dataset to see if we get any improvement in performance for our models.

<u>Step 5: Set-up SageMaker Linear Model on Balanced Dataset</u>

We do not perform Hyperparameter Tuning at this stage since we have very few hyperparameters to perfect. Below is a snapshot of the SageMaker Training script and the performance evaluation metrics.



```
[4]: best_model_balanced_ll = {'bogo': {}, 'discount': {}, 'info': {}}

[6]: for prefix in ['bogo', 'discount', 'info']:
         # Create instance of LinearLearner
         ll = LinearLearner(role,
                            train_instance_count=1,
                            train_instance_type='ml.c4.xlarge',
                            predictor_type='binary_classifier',
                            output_path='s3://{}/Capstone_Starbucks/{}/model'.format(bucket, prefix),
                            sagemaker_session=session,
                            binary_classifier_model_selection_criteria='f1',
                            epochs=100,
                            use_bias=True,
                            optimizer='adam',
                            loss='auto',
                            wd=0,
                            normalize_data=True,
                            unbias_data=True,
                            early_stopping_patience=5,
                            learning_rate=0.01,
                            balance_multiclass_weights=True)

         # Create record sets from local data as inputs to the LinearLearner
         train = pd.read_csv(f'./data/{prefix}/{prefix}_train_balanced.csv', header=None)
         train_data = ll.record_set(train.drop(0, 1).values.astype('float32'), labels=train[0].values.astype('float32'),
                                    channel='train')

         valid = pd.read_csv(f'./data/{prefix}/{prefix}_val_balanced.csv', header=None)
         validation_data = ll.record_set(valid.drop(0, 1).values.astype('float32'), labels=valid[0].values.astype('float32'
                                    channel='validation')

         # Fit the model
         ll.fit([train_data, validation_data], logs=False)

         # Save results
         tja = ll.training_job_analytics
         res = tja.dataframe()
         best_model_balanced_ll[prefix]['ll'] = {'name': tja.name,
                                   'value': res.loc[res['metric_name'] == 'validation:binary_f_beta', 'value'].values[0],
                                   'model': ll}
```

Fig 31: SageMaker Linear Learner Training script

Confusion Matrix BOGO	Predicted Positive	Predicted Negative
True Positive	0.74421	0.25579
True Negative	0.48277	0.51723

Confusion Matrix Discount	Predicted Positive	Predicted Negative
True Positive	0.64839	0.35161
True Negative	0.38969	0.61031

Confusion Matrix Informational	Predicted Positive	Predicted Negative
True Positive	0.59671	0.40329
True Negative	0.31990	0.68010

Fig 32: Linear Model Confusion Matrix on Balanced Dataset

Accuracy Metrics BOGO	Value
Accuracy	0.55738
Precision	0.24884
Recall	0.74421
F1	0.37297

Accuracy Metrics Discount	Value
Accuracy	0.61827

Precision	0.30542
Recall	0.64839
F1	0.41525

Accuracy Metrics Informational	Value
Accuracy	0.67345
Precision	0.13916
Recall	0.59671
F1	0.22568

Fig 33: Linear Model Accuracy Metrics on Balanced Dataset

Step 6: Inference and Conclusion

Model selection for prediction is dependent on the business questions which we are trying to answer. False negatives predicted by the model would mean a future economic loss since the customer would have desirably responded to the offer by conducting a transaction. False positives predicted by the model would mean wasted marketing cost incurred in rolling the offer to the customer. Ideally, we would like to reduce the proportion of both false positives and false negatives.

Consider situations where we require the model to predict the negatives correctly, business situations like these can happen in the following scenarios

- We need to make initial assessments of how well on average a group of customers new to Starbucks would respond to Starbucks offers before making investments in creating/releasing offers (a large proportion of predicted negatives might indicate a loss-making investment)

In these situations, we use the accuracy metrics for selecting the best model for each offer-type to be used in prediction

Offer-type	Selected Model
BOGO	XGBoost
Discount	XGBoost
Informational	XGBoost

Fig 34: Selected Models for Negative Prediction

We see from the accuracy metrics and confusion matrices above that the XGBoost Model on Balanced Dataset is able to correctly classify around 77%, 63% and 91% of the Negative offers respectively in each of the three offer-type categories namely BOGO, Discount and Informational. As the percentage of correct classifications are high, we can conclude that the models above can be used for the said business context use-case.

Consider situations where we need the model to predict positives correctly, business situations like these can happen if we want to send the offers to all the customers in a targeted population who will respond to offers positively.

In these situations, we can use the Recall scores to find the optimal models for each offer-type

Offer-type	Selected Model
BOGO	Linear Model
Discount	XGBoost
Informational	Linear Model

Fig 35: Selected Models for Positive Predictions

Linear Model is able to correctly classify around 74% of the True Positives, for XGBoost the number stands at 64% and for Informational offers, the Linear Model predicts around 60% of the True positives correctly. As the percentage of correct classifications are high, we can conclude that the models above can be used for the said business context use-case.

Hence, we can use the XGBoost Model on Informational offers and for other two offer-types we should use the models considering the broader business context of usage.

6. Conclusion

In this Project, I have modeled the purchasing patterns of customers to predict how well they react to different offers of each type. The broader process involved include cleaning, analyzing and processing three different datasets for offer-types followed by feature engineering to include the explanatory variables for purchasing patterns of the customers involved and classification of offers into desirable/non-desirable category. This transforms the problem into a classification task for which we set up different models like Logistic Regression, SageMaker XGBoost and SageMaker Linear Model for prediction. Finally, we make the conclusions on which model to use under different business contexts of usage.

We notice that we do not have a single model which can be used for all the three offer-types. This is mostly due to the fact that SageMaker Models do not perform very well on predicting the positives correctly. Classification of positives is generally, a difficult task since some customers might react to offers erratically thereby reducing the learning power of the algorithm. Gathering more input data about app usage and about the types of products purchased by different customers can help us in understanding the purchasing patterns better. We could also try other algorithms like Random Forest and SVM for relative comparison of performance with the models already implemented.

7. References

https://en.wikipedia.org/wiki/Gradient_descent

https://en.wikipedia.org/wiki/Logistic_regression

https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html

https://docs.aws.amazon.com/sagemaker/latest/dg/linear-learner.html

https://en.wikipedia.org/wiki/RFM_(market_research)