Udacity Machine Learning Engineer Nanodegree Capstone Project

Optimizing Offer Impact in the Starbucks Reward Program: A Machine Learning Approach

Khoa Pham

November 2023

Definition

Domain Background & Project Overview

The influence of personalized marketing is undeniable in today's retail sector, significantly impacting customer retention and satisfaction. The Starbucks Rewards mobile app is one such avenue where personalized marketing is executed, providing offers to customers to enhance sales and customer loyalty. Historically, companies like Starbucks have employed various strategies to entice customer purchases, including the use of special offers and discounts (Smith, 2016). However, not all customers respond to all offers, necessitating a more targeted marketing approach. This project aims to explore this domain, leveraging data science and machine learning to predict customer behavior in response to offers, thereby optimizing the marketing strategy.

The goal of my capstone project is to understand the customer behavior through machine learning models and decide which features of customer will respond to promotion offers that empowers a more targeted marketing approach to launch promotion plan.

Multiple classification models are built and evaluated to predict the probability that the customer will complete the promotion offer. As a result, it will maximize the promotion campaign efficiency later in the future.

Problem Statement

The primary challenge Starbucks faces is determining which offers should be sent to which customers to maximize the likelihood of a positive response. The current strategy involves sending out offers indiscriminately, leading to resource wastage and potential customer dissatisfaction. The problem is quantifiable (increase in offer response rate or purchase amount) and replicable (can be tested with different offers and customer segments).

The proposed solution is to develop a machine learning model that predicts the likelihood of a customer responding to an offer. This predictive model will consider various customer demographic factors and the offer's details, providing Starbucks with a data-driven method to target offers more effectively. The model's effectiveness will be quantifiable (accuracy of predictions) and replicable (can be re-implemented as new data becomes available).

The proposed solution involves the development of a predictive model on AWS SageMaker Studio. Utilizing SageMaker's advanced capabilities and broad algorithmic support, we'll develop a model that predicts customer responses to offers based on their demographic data and offer details. This cloud-based approach enables scalable model training with increased computational efficiency.

We will start with baseline models such as Logistic Regression to establish benchmark performance. Progressing to more sophisticated algorithms, we will experiment with ensemble methods like Random Forests and Gradient Boosting Machines, known for their higher accuracy due to aggregated learning. All these models will be implemented within the AWS SageMaker Studio, leveraging its built-in algorithms and Jupyter notebook-based interface for iterative development. Additionally, all the algorithms will be hyper tunning to get better results on each model initially. This task will heavily consume computing resources on Sagemaker Studio.

Evaluation Metrics

The primary metric for evaluating the effectiveness of our model will be the F1-score, as it balances precision and recall, providing a more comprehensive performance measure when classes are imbalanced (i.e., when the number of offers completed vs. not completed are unequal). Additionally, the accuracy and AUC-ROC curve will be considered to provide a holistic view of the model's performance.

F1-Score

The F1-score is a harmonic mean of precision and recall, which provides a balance between the two:

- **Precision** is the ratio of true positives (correct positive predictions) to all positive predictions, including false positives. It answers the question: "Of all the instances the model labeled positive, how many are actually positive?"
- **Recall**, also known as sensitivity, is the ratio of true positives to the sum of true positives and false negatives. It answers: "Of all the actual positive instances, how many did the model correctly identify?"

The F1-score combines these into a single metric by calculating their harmonic mean, which is a way to penalize extreme values more heavily. The F1-score ranges from 0 to 1, where 1 is perfect precision and recall, and 0 is the worst. It is particularly useful when you have classes that are imbalanced because it does not get inflated by the large number of true negatives.

Accuracy

Accuracy is the most intuitive performance measure, and it is simply a ratio of correctly predicted observation to the total observations. It is the total number of correct predictions divided by the total number of predictions made for a dataset.

While accuracy can be useful, it can also be misleading when dealing with imbalanced classes. If one class is much larger than the other, a model can have high accuracy by simply predicting the majority class all the time, without capturing the minority class effectively.

AUC-ROC Score
The Area Under the Receiver Operating Characteristic (ROC) Curve (AUC-ROC) is a performance measurement for classification problems at various threshold settings. The ROC is a probability curve that plots the true positive rate (recall) against the false positive rate at various threshold values and essentially separates the 'signal' from the 'noise'. The AUC is the area under the ROC curve; it reduces the ROC curve to a single value, which represents the likelihood that a model will rank a randomly chosen positive instance higher than a randomly chosen negative one.

AUC-ROC is a useful metric when dealing with imbalanced classes because:

- It considers all possible thresholds for a given classifier, not just the one resulting from the default cut-off of 0.5.
- It is not biased towards the majority class.

By considering all three metrics, you get a more rounded picture of your model's performance. The F1-score can tell you how well your model identifies the minority class, accuracy gives you a quick summary of overall correctness, and AUC-ROC tells you how well your model distinguishes between classes.

Analysis

Data Exploration & Exploratory Visualizations

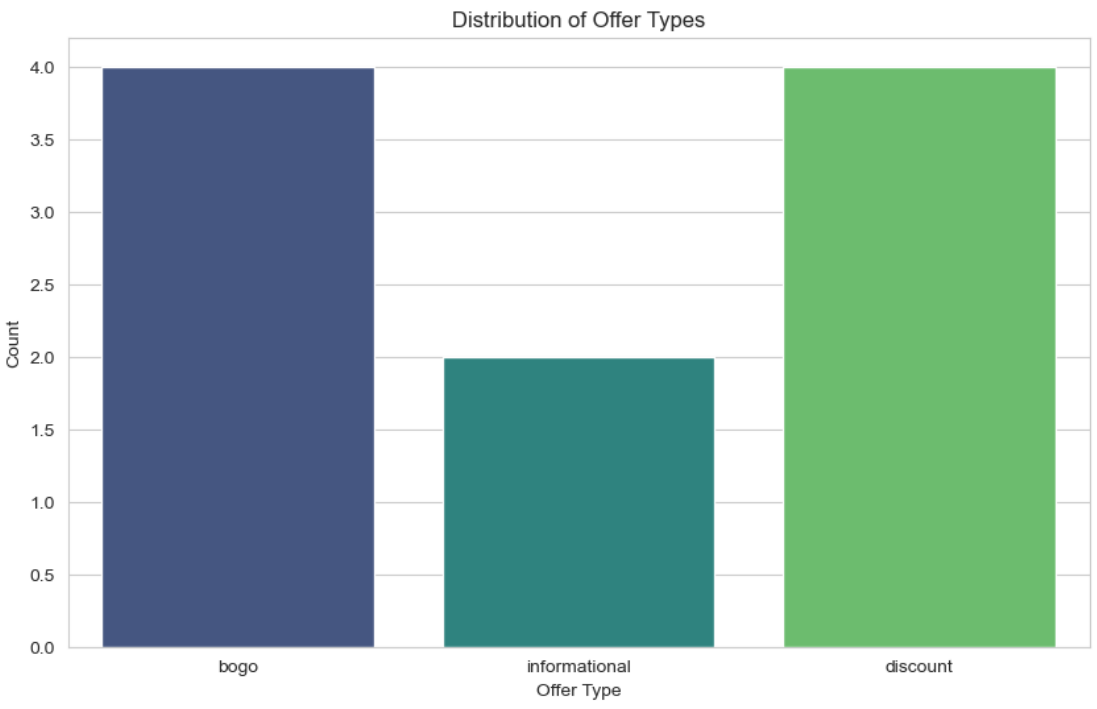
The data is contained in three files:

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

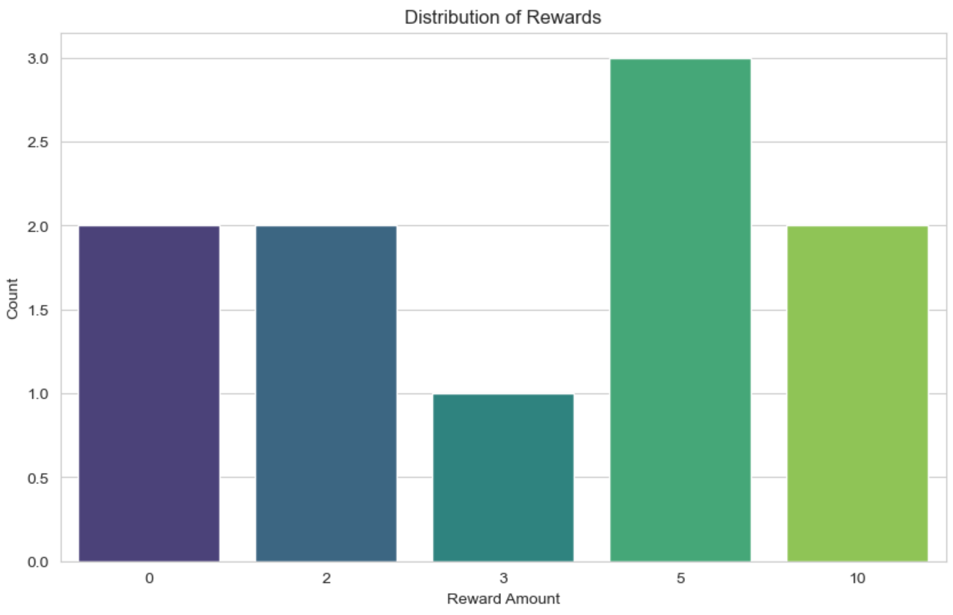
Here is the schema and explanation of each variable in the files:

portfolio.json

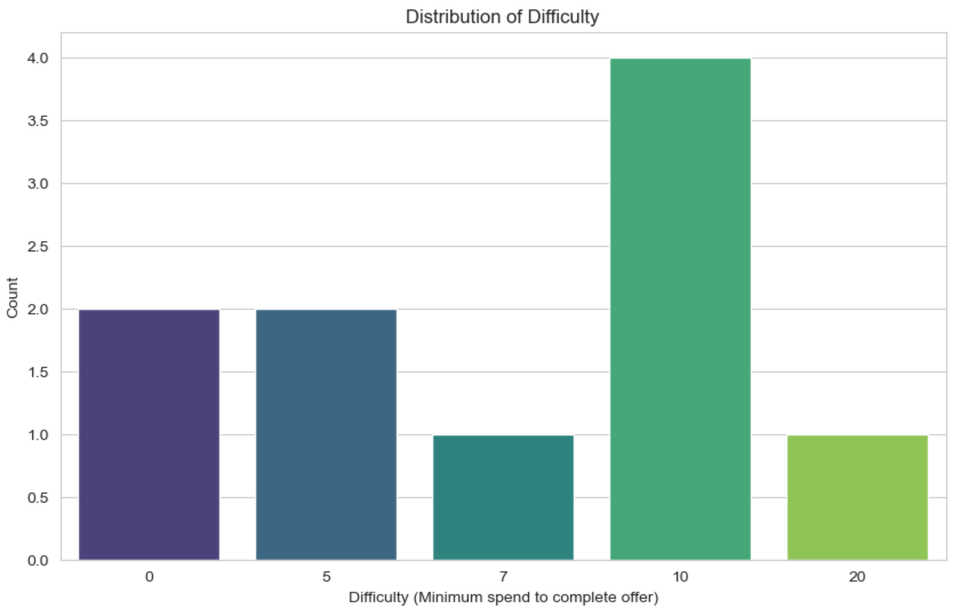
- id (string) - offer id
- offer_type (string) - type of offer ie BOGO, discount, informational
- difficulty (int) - minimum required spend to complete an offer
- reward (int) - reward given for completing an offer
- duration (int) - time for offer to be open, in days
- channels (list of strings)

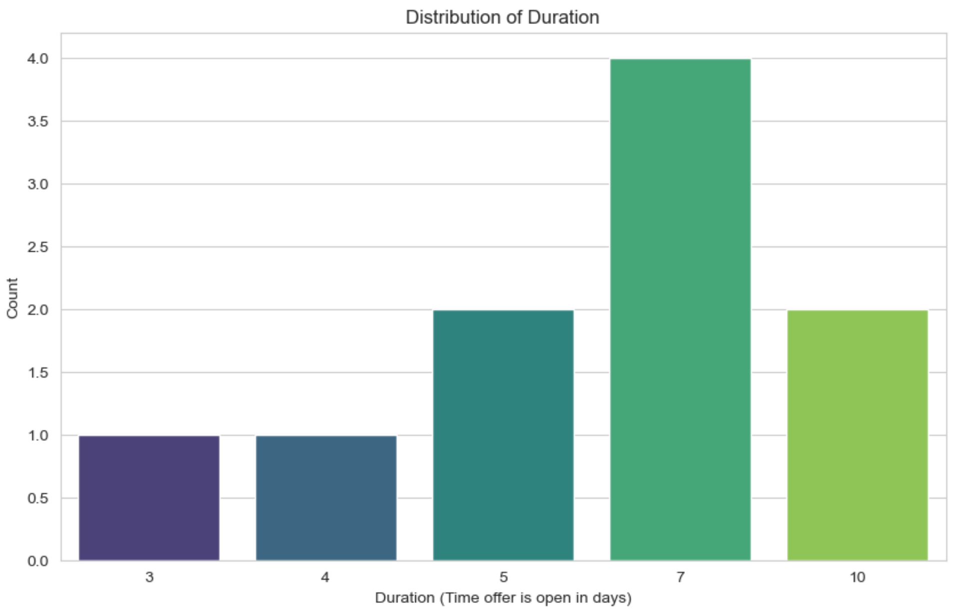


Distribution of Offer Types
Count
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
bogo
informational
discount
Offer Type



Distribution of Rewards
Count
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
3
5
10
Reward Amount





The visualizations provide insights into the portfolio dataset:

1. Distribution of Offer Types:

The dataset contains three types of offers: BOGO (Buy One Get One), Discount, and Informational. Both BOGO and Discount offers have similar counts, while Informational offers are fewer in number.

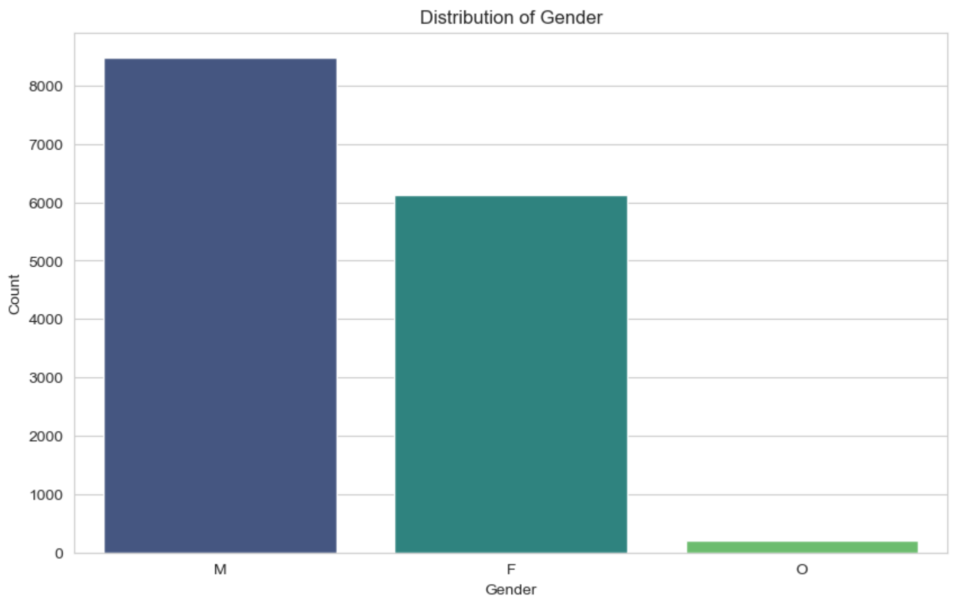
2. Distribution of Rewards: The rewards range from 0 to 10 units. There are offers with no rewards, which are likely the Informational offers. The most common reward amounts are 5 and 10 units.

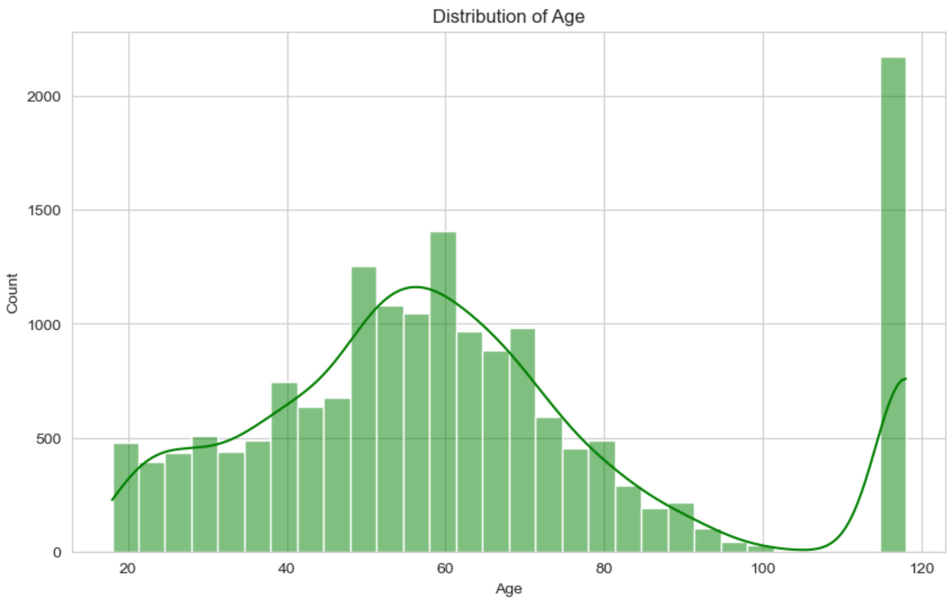
3. Distribution of Difficulty: The difficulty (or minimum expenditure to complete an offer) ranges from 0 to 20 units. As with rewards, offers with a difficulty of 0 are likely the Informational offers. The most common difficulties are set at 10 and 5 units.

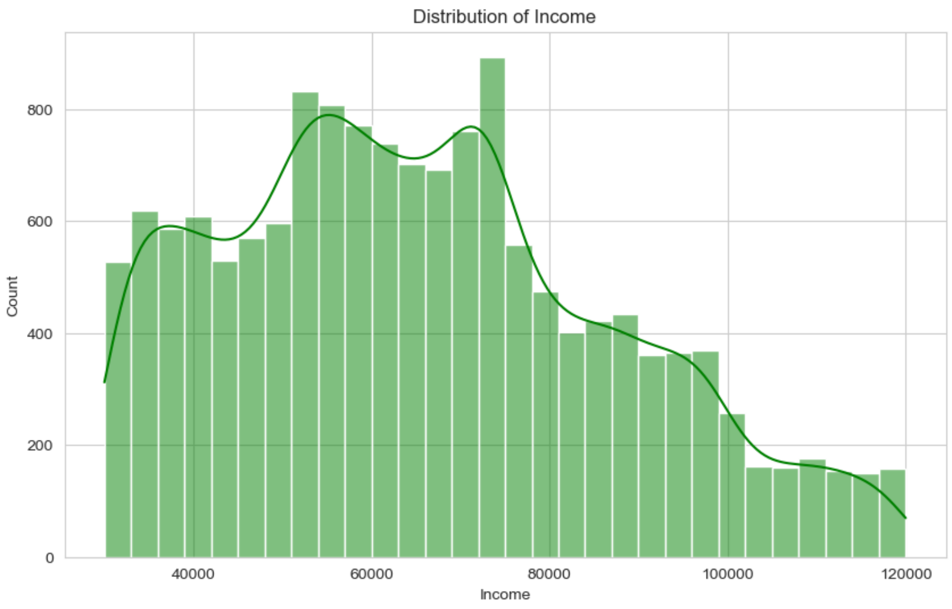
4. Distribution of Duration: Offer durations range from 4 to 10 days. The most common duration is 7 days, followed by 10 days.

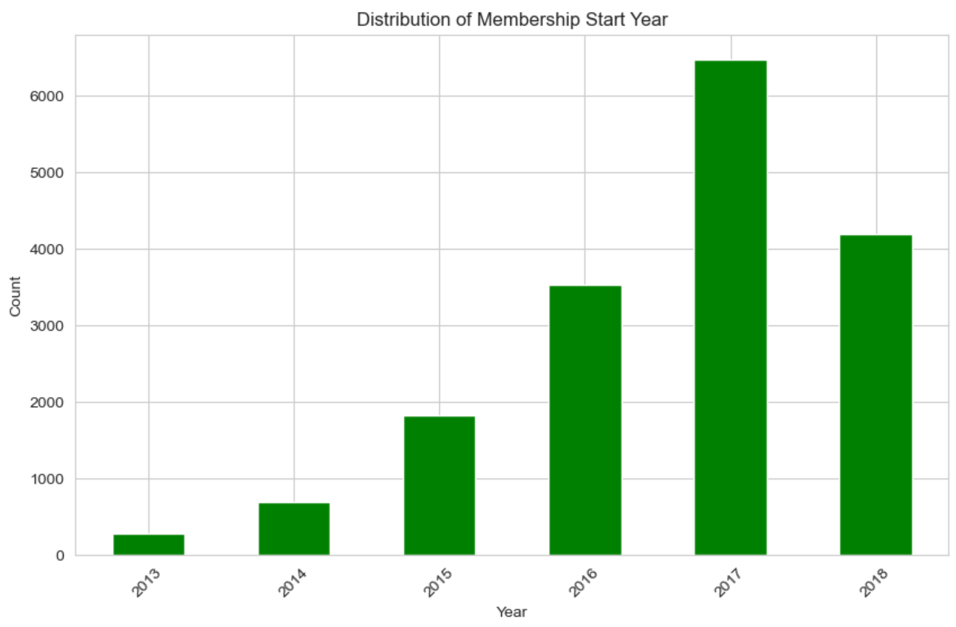
profile.json

- 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




Distribution of Age
Count
0
500
1000
1500
2000
20
40
60
80
100
120
Age


Distribution of Income
Count
0
200
400
600
800
40000
60000
80000
100000
120000
Income



Here are the insights from the profile dataset visualizations:

1. Distribution of Gender: Most customers in the dataset are male, followed by female and other. There's a significant portion of customers with unspecified genders (2175 records) (represented as None).

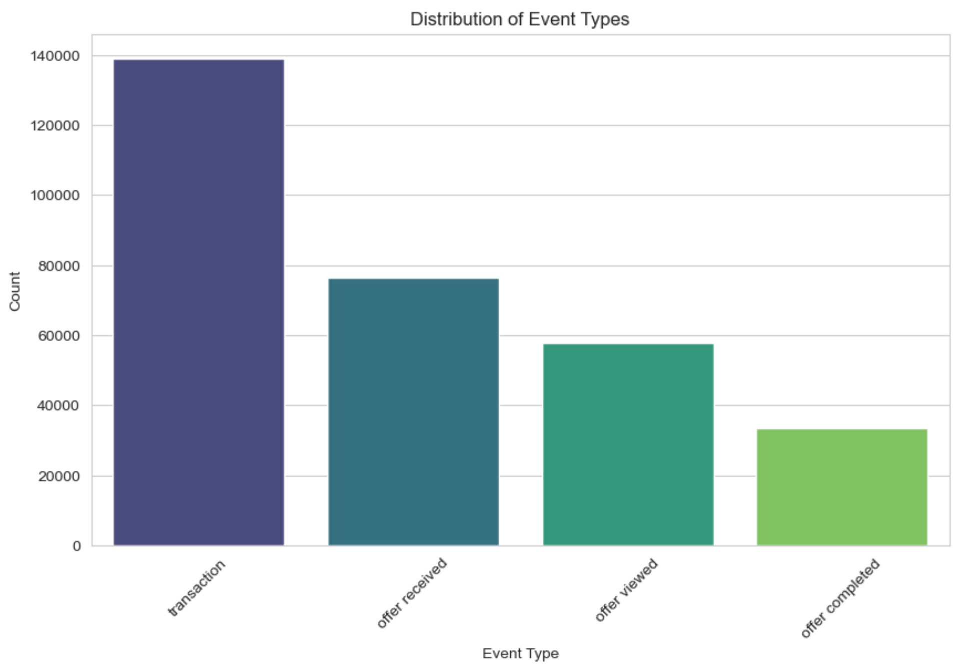
2. Distribution of Age: The age distribution has a wide range, with a peak around the early 60s. There's a noticeable spike at age 118, which appears to be a placeholder or default value for unspecified ages. This will need to be addressed during data cleaning.

3. Distribution of Income: Income distribution appears to be multimodal, with peaks around 30k, 50k and 70k USD. The number of null records of income is aligned with gender column. This could be handled in the cleaning phase.

4. Distribution of Membership Start Year: A significant number of customers joined in 2017, followed by 2018. Membership numbers have been growing since 2013.

transcript.json

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



From the transcript dataset visualizations and exploration:

1. Distribution of Event Types: The dataset contains four types of events: offer received, offer viewed, transaction, and offer completed. The most common event is transaction, followed by offer received, offer viewed, and offer completed.

2. Exploration of the 'value' Column: The value column contains dictionaries with different keys. We've identified four unique keys: offer id, amount, offer_id, and reward. The presence of both offer id and offer_id suggests that there might be some inconsistency in the data, which we'll need to address during data cleaning.

These datasets will be combined and cleaned to create a comprehensive view of customer-offer interactions, forming the foundation for predictive modeling.

Algorithms and Techniques

The project performs many machine learning models to conquer the problem of targeting marketing to whom would be responding to the promotion offer from Starbucks. First, the model is **Logistic Regression** for binary classification as a benchmark model. This result model would be a baseline to continue to develop more advanced models like **Random Forest Classification** and **XGboost Classification.** In addition, the project utilizes the RandomSearchCV for hyper tunning the results.

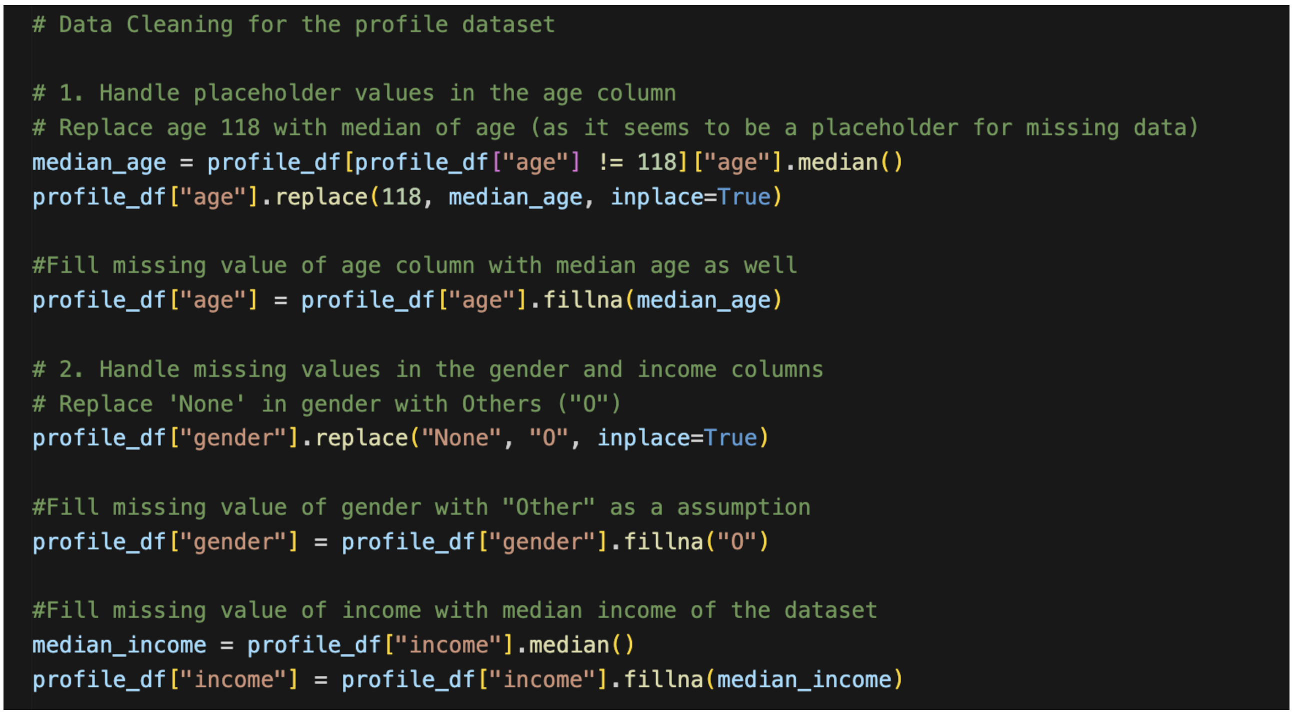
Benchmark

The Logistic Regression classification model is a benchmark model. This model is simple and wide range application in scikit-learn machine learning library. Furthermore, the project will evaluate the model with other advanced machine learning model and apply hypertunning to achieve better results.

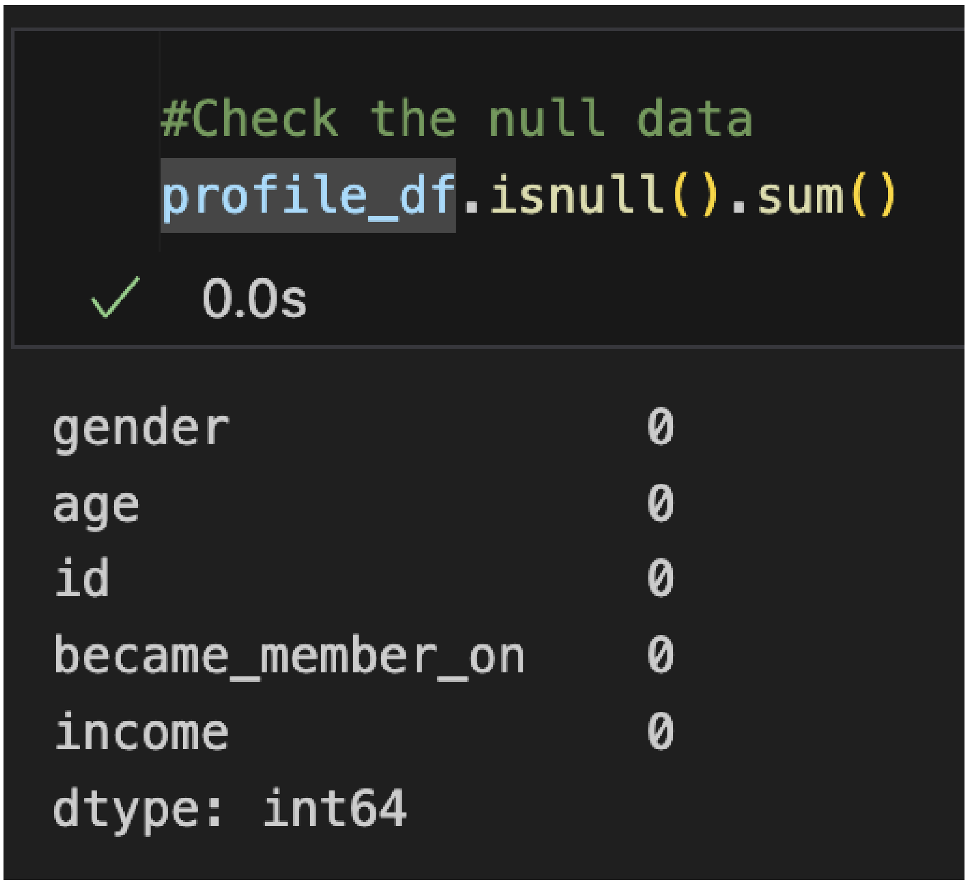
Methodology

Data Cleaning & Preprocessing

- Handle missing values and placeholder values.
- Address inconsistencies, such as the offer id and offer_id keys in the transcript dataset.
- Transform categorical variables.



```python
# Data Cleaning for the profile dataset

# 1. Handle placeholder values in the age column
# Replace age 118 with median of age (as it seems to be a placeholder for missing data)
median_age = profile_df[profile_df["age"] != 118]["age"].median()
profile_df["age"].replace(118, median_age, inplace=True)

#Fill missing value of age column with median age as well
profile_df["age"] = profile_df["age"].fillna(median_age)

# 2. Handle missing values in the gender and income columns
# Replace 'None' in gender with Others ("O")
profile_df["gender"].replace("None", "O", inplace=True)

#Fill missing value of gender with "Other" as a assumption
profile_df["gender"] = profile_df["gender"].fillna("O")

#Fill missing value of income with median income of the dataset
median_income = profile_df["income"].median()
profile_df["income"] = profile_df["income"].fillna(median_income)
```

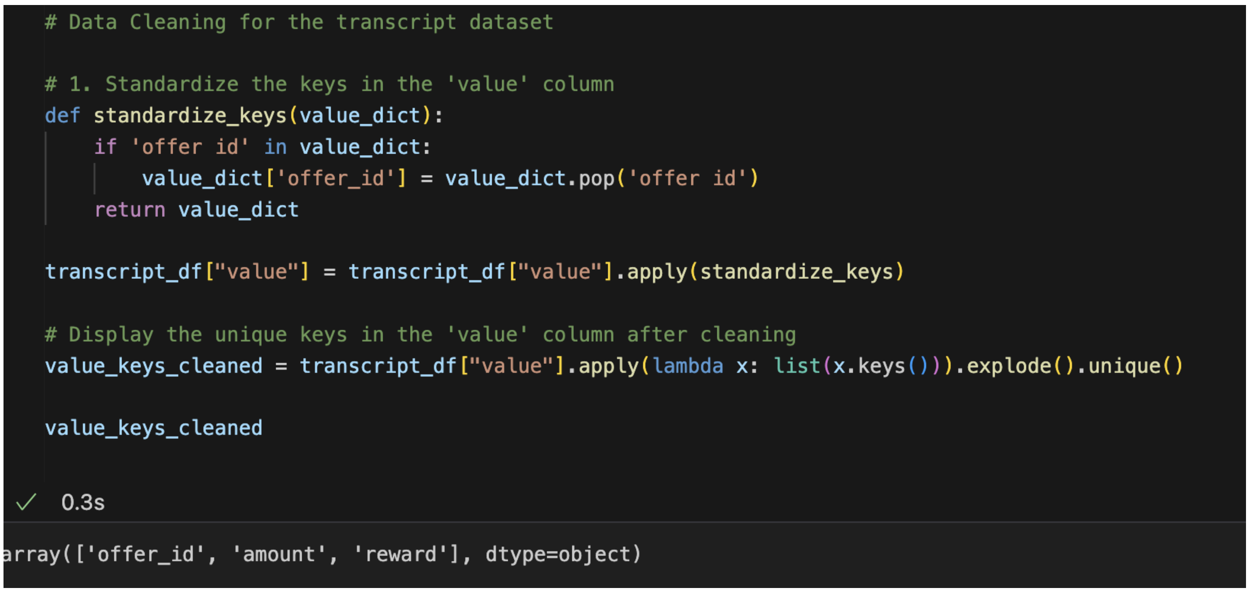


```
#Check the null data
profile_df.isnull().sum()
```

✓ 0.0s

```
gender              0
age                 0
id                  0
became_member_on    0
income              0
dtype: int64
```

Next, we'll address the inconsistencies in the transcript dataset, specifically the value column having both offer id and offer_id keys.



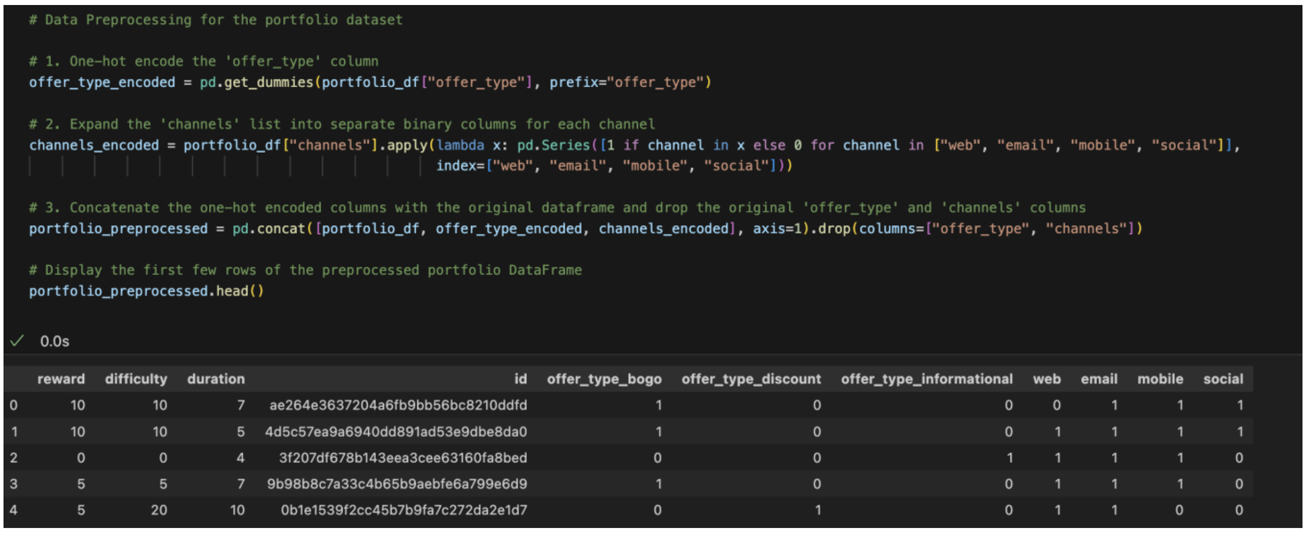
```
# Data Cleaning for the transcript dataset

# 1. Standardize the keys in the 'value' column
def standardize_keys(value_dict):
    if 'offer id' in value_dict:
        value_dict['offer_id'] = value_dict.pop('offer id')
    return value_dict

transcript_df["value"] = transcript_df["value"].apply(standardize_keys)

# Display the unique keys in the 'value' column after cleaning
value_keys_cleaned = transcript_df["value"].apply(lambda x: list(x.keys())).explode().unique()

value_keys_cleaned
```

✓ 0.3s

```
array(['offer_id', 'amount', 'reward'], dtype=object)
```

The inconsistencies in the transcript dataset's value column have been addressed. The keys are now standardized to offer_id, amount, and reward.

Next, we'll preprocess the portfolio dataset, mainly focusing on transforming the categorical variable offer_type using one-hot encoding and expanding the channels list into separate binary columns for each channel.

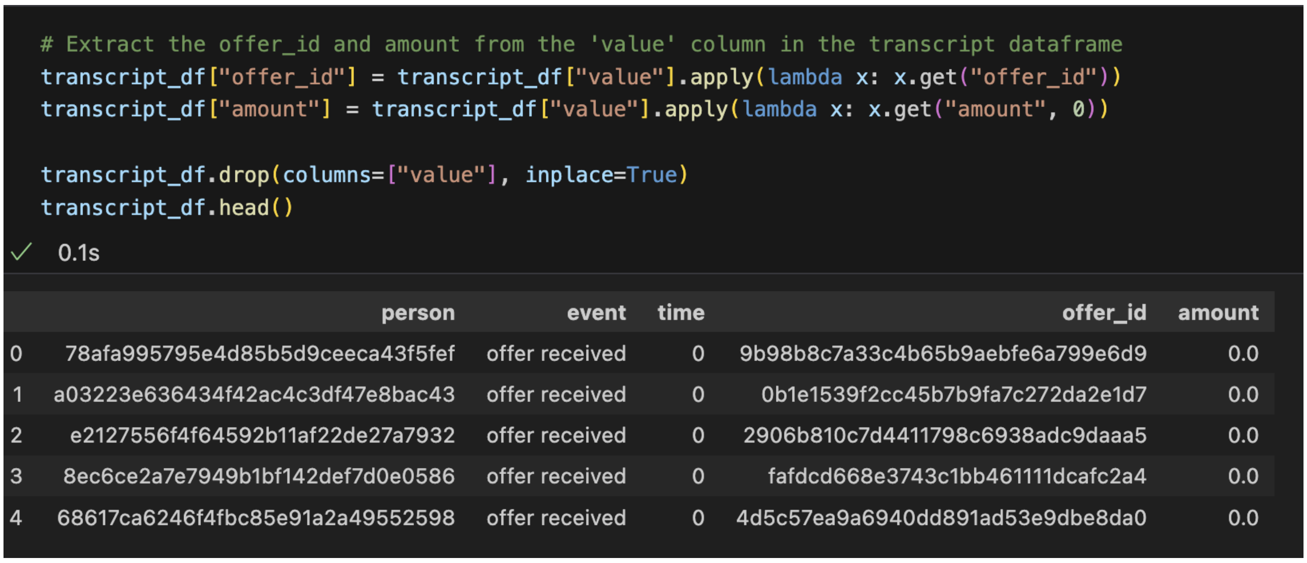


```python
# Data Preprocessing for the portfolio dataset

# 1. One-hot encode the 'offer_type' column
offer_type_encoded = pd.get_dummies(portfolio_df["offer_type"], prefix="offer_type")

# 2. Expand the 'channels' list into separate binary columns for each channel
channels_encoded = portfolio_df["channels"].apply(lambda x: pd.Series([1 if channel in x else 0 for channel in ["web", "email", "mobile", "social"]],
                                                                       index=["web", "email", "mobile", "social"]))

# 3. Concatenate the one-hot encoded columns with the original dataframe and drop the original 'offer_type' and 'channels' columns
portfolio_preprocessed = pd.concat([portfolio_df, offer_type_encoded, channels_encoded], axis=1).drop(columns=["offer_type", "channels"])

# Display the first few rows of the preprocessed portfolio DataFrame
portfolio_preprocessed.head()
```

```
✓  0.0s

   reward  difficulty  duration                                id  offer_type_bogo  offer_type_discount  offer_type_informational  web  email  mobile  social
0      10          10         7  ae264e3637204a6fb9bb56bc8210ddfd                1                    0                         0    0      1       1       1
1      10          10         5  4d5c57ea9a6940dd891ad53e9dbe8da0                1                    0                         0    1      1       1       1
2       0           0         4  3f207df678b143eea3cee63160fa8bed                0                    0                         1    1      1       1       0
3       5           5         7  9b98b8c7a33c4b65b9aebfe6a799e6d9                1                    0                         0    1      1       1       0
4       5          20        10  0b1e1539f2cc45b7b9fa7c272da2e1d7                0                    1                         0    1      1       0       0
```

The offer_type column has been one-hot encoded, resulting in separate columns for each offer type (BOGO, discount, and informational).

The channels list column has been expanded into separate binary columns for each channel (web, email, mobile, and social).



```python
# Extract the offer_id and amount from the 'value' column in the transcript dataframe
transcript_df["offer_id"] = transcript_df["value"].apply(lambda x: x.get("offer_id"))
transcript_df["amount"] = transcript_df["value"].apply(lambda x: x.get("amount", 0))

transcript_df.drop(columns=["value"], inplace=True)
transcript_df.head()
```

```
✓  0.1s

                             person           event  time                          offer_id  amount
0  78afa995795e4d85b5d9ceeca43f5fef  offer received     0  9b98b8c7a33c4b65b9aebfe6a799e6d9     0.0
1  a03223e636434f42ac4c3df47e8bac43  offer received     0  0b1e1539f2cc45b7b9fa7c272da2e1d7     0.0
2  e2127556f4f64592b11af22de27a7932  offer received     0  2906b810c7d4411798c6938adc9daaa5     0.0
3  8ec6ce2a7e7949b1bf142def7d0e0586  offer received     0  fafdcd668e3743c1bb461111dcafc2a4     0.0
4  68617ca6246f4fbc85e91a2a49552598  offer received     0  4d5c57ea9a6940dd891ad53e9dbe8da0     0.0
```

Feature Engineering and Selection

- Combine the datasets to create a comprehensive view of customer-offer interactions.
- Generate new features that may be valuable for predictive modeling.
- Select relevant features for modeling.

The datasets have been successfully merged into a unified dataframe, capturing customer-offer interactions. The merged dataframe contains:

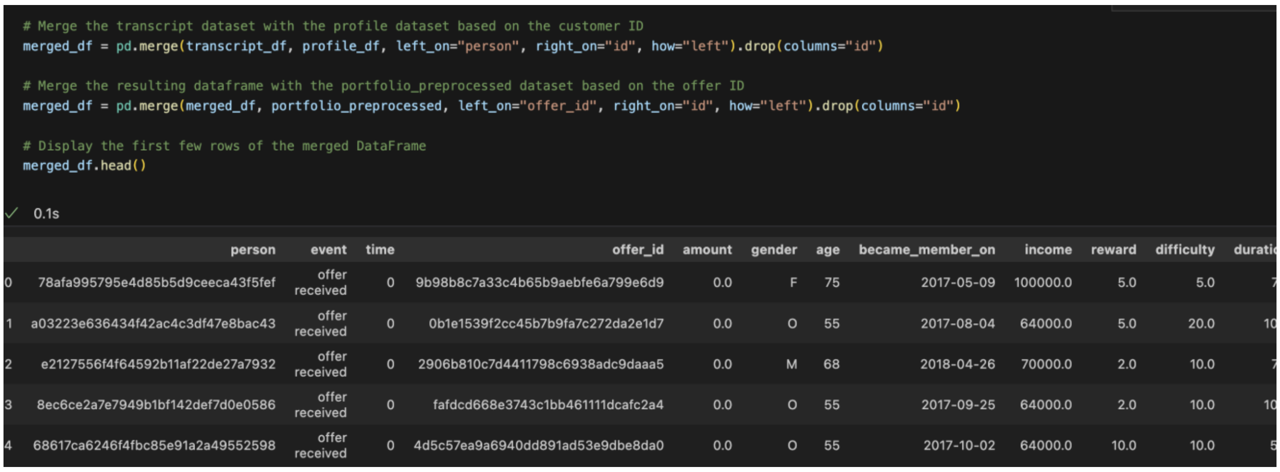
- Customer details like gender, age, income, and membership start date.

- Transcript events like event, time, amount, and offer_id.

- Offer details like reward, difficulty, duration, offer type, and channels.

- Membership Duration: The duration (in days) since the customer became a member up to the current date.

- Event Type Encoding: The event column has been transformed into separate binary columns for each event type (event_offer completed, event_offer received, event_offer viewed, and event_transaction).

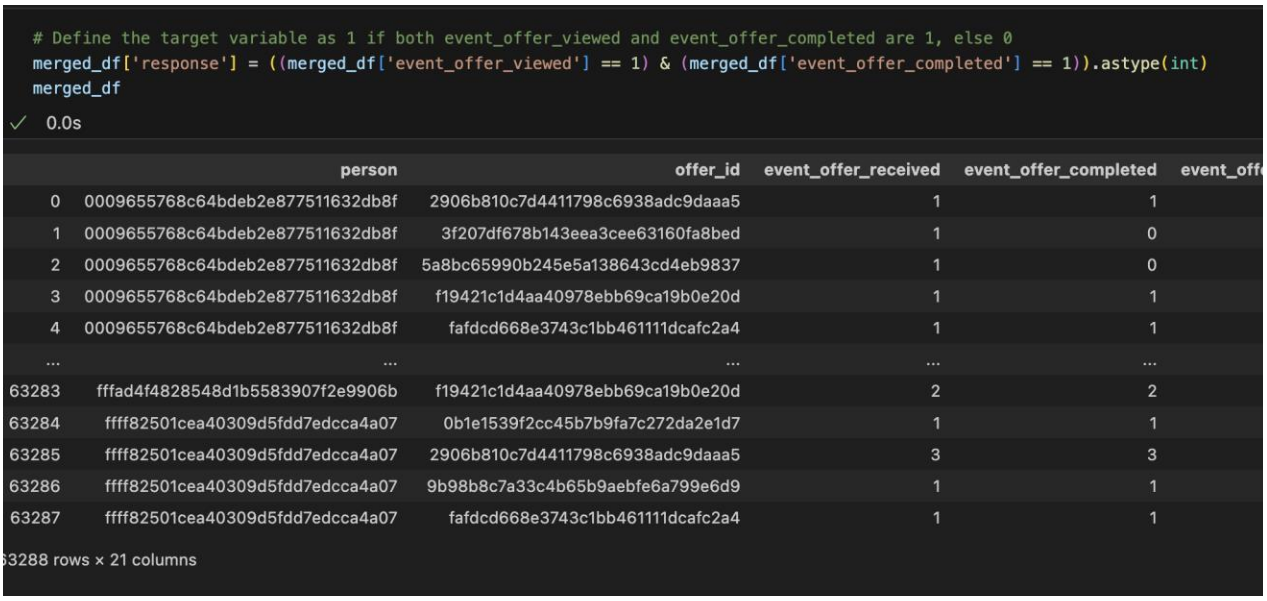


```python
# Merge the transcript dataset with the profile dataset based on the customer ID
merged_df = pd.merge(transcript_df, profile_df, left_on="person", right_on="id", how="left").drop(columns="id")

# Merge the resulting dataframe with the portfolio_preprocessed dataset based on the offer ID
merged_df = pd.merge(merged_df, portfolio_preprocessed, left_on="offer_id", right_on="id", how="left").drop(columns="id")

# Display the first few rows of the merged DataFrame
merged_df.head()
```

✓ 0.1s

	person	event	time	offer_id	amount	gender	age	became_member_on	income	reward	difficulty	durati
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	0	9b98b8c7a33c4b65b9aebfe6a799e6d9	0.0	F	75	2017-05-09	100000.0	5.0	5.0	7
1	a03223e636434f42ac4c3df47e8bac43	offer received	0	0b1e1539f2cc45b7b9fa7c272da2e1d7	0.0	O	55	2017-08-04	64000.0	5.0	20.0	10
2	e2127556f4f64592b11af22de27a7932	offer received	0	2906b810c7d4411798c6938adc9daaa5	0.0	M	68	2018-04-26	70000.0	2.0	10.0	7
3	8ec6ce2a7e7949b1bf142def7d0e0586	offer received	0	fafdcd668e3743c1bb461111dcafc2a4	0.0	O	55	2017-09-25	64000.0	2.0	10.0	10
4	68617ca6246f4fbc85e91a2a49552598	offer received	0	4d5c57ea9a6940dd891ad53e9dbe8da0	0.0	O	55	2017-10-02	64000.0	10.0	10.0	5

Define the Target variable either the cusotmer respone to the promotion or not as below .



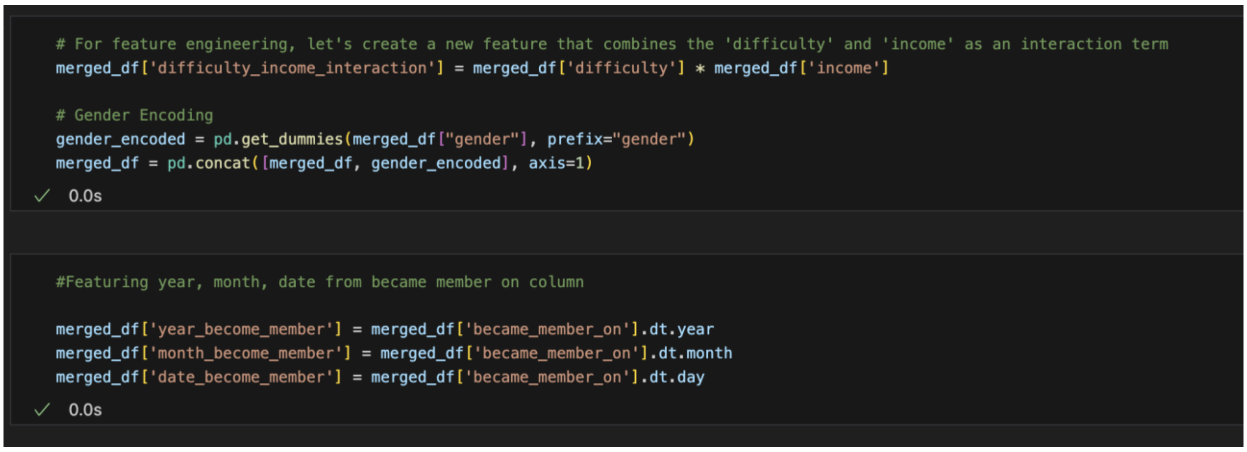
```python
# Define the target variable as 1 if both event_offer_viewed and event_offer_completed are 1, else 0
merged_df['response'] = ((merged_df['event_offer_viewed'] == 1) & (merged_df['event_offer_completed'] == 1)).astype(int)
merged_df
```

✓ 0.0s

	person	offer_id	event_offer_received	event_offer_completed	event_off
0	0009655768c64bdeb2e877511632db8f	2906b810c7d4411798c6938adc9daaa5	1	1	
1	0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed	1	0	
2	0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837	1	0	
3	0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d	1	1	
4	0009655768c64bdeb2e877511632db8f	fafdcd668e3743c1bb461111dcafc2a4	1	1	
...	...	...	...	...	
63283	fffad4f4828548d1b5583907f2e9906b	f19421c1d4aa40978ebb69ca19b0e20d	2	2	
63284	ffff82501cea40309d5fdd7edcca4a07	0b1e1539f2cc45b7b9fa7c272da2e1d7	1	1	
63285	ffff82501cea40309d5fdd7edcca4a07	2906b810c7d4411798c6938adc9daaa5	3	3	
63286	ffff82501cea40309d5fdd7edcca4a07	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	1	
63287	ffff82501cea40309d5fdd7edcca4a07	fafdcd668e3743c1bb461111dcafc2a4	1	1	

3288 rows × 21 columns

Add some more feature: interaction feature by combinng “difficulty" & “income”, gender encoding and enriching more features on “became_member_on".



```
# For feature engineering, let's create a new feature that combines the 'difficulty' and 'income' as an interaction term
merged_df['difficulty_income_interaction'] = merged_df['difficulty'] * merged_df['income']

# Gender Encoding
gender_encoded = pd.get_dummies(merged_df["gender"], prefix="gender")
merged_df = pd.concat([merged_df, gender_encoded], axis=1)
```
✓ 0.0s

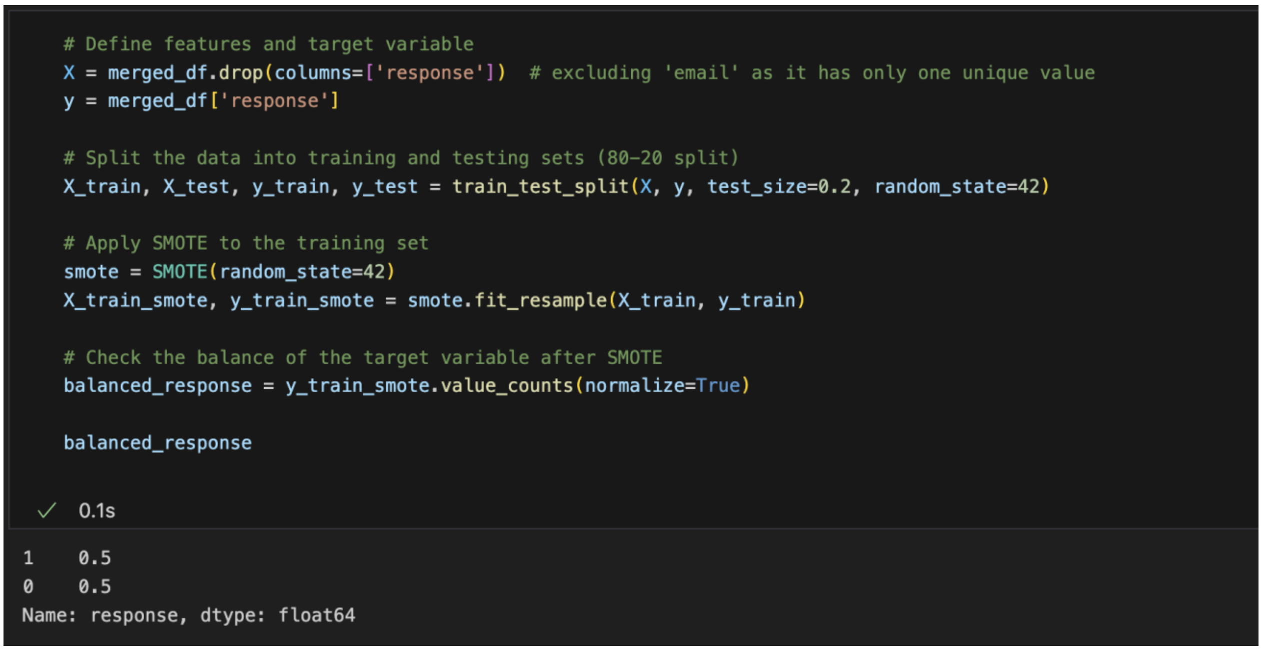
```
#Featuring year, month, date from became member on column

merged_df['year_become_member'] = merged_df['became_member_on'].dt.year
merged_df['month_become_member'] = merged_df['became_member_on'].dt.month
merged_df['date_become_member'] = merged_df['became_member_on'].dt.day
```
✓ 0.0s

Implementation & Refinement

Here are the steps we'll follow:

- **Data Splitting**: We'll split the dataset into training and testing sets.
- **Balancing the Dataset**: We'll apply SMOTE only to the training set to avoid introducing bias in the model evaluation.
- **Model Training**: We'll train a baseline model to start, such as Logistic Regression, to gauge the initial performance. Then continue advancing more models with hyperparameter tunning.
- **Evaluation**: We'll evaluate the model using the F1-score as the primary metric and also consider accuracy, and the AUC-ROC score.

Let's proceed with the data splitting. We will use a typical train-test split of 80-20 for this purpose. To address the imbalance in the target variable, we will use a technique called Synthetic Minority Over-sampling Technique (SMOTE). SMOTE generates synthetic samples for the minority class (in this case, positive responses) to achieve balance between the classes.



```
# Define features and target variable
X = merged_df.drop(columns=['response'])  # excluding 'email' as it has only one unique value
y = merged_df['response']

# Split the data into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE to the training set
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Check the balance of the target variable after SMOTE
balanced_response = y_train_smote.value_counts(normalize=True)

balanced_response
```

✓ 0.1s

```
1    0.5
0    0.5
Name: response, dtype: float64
```

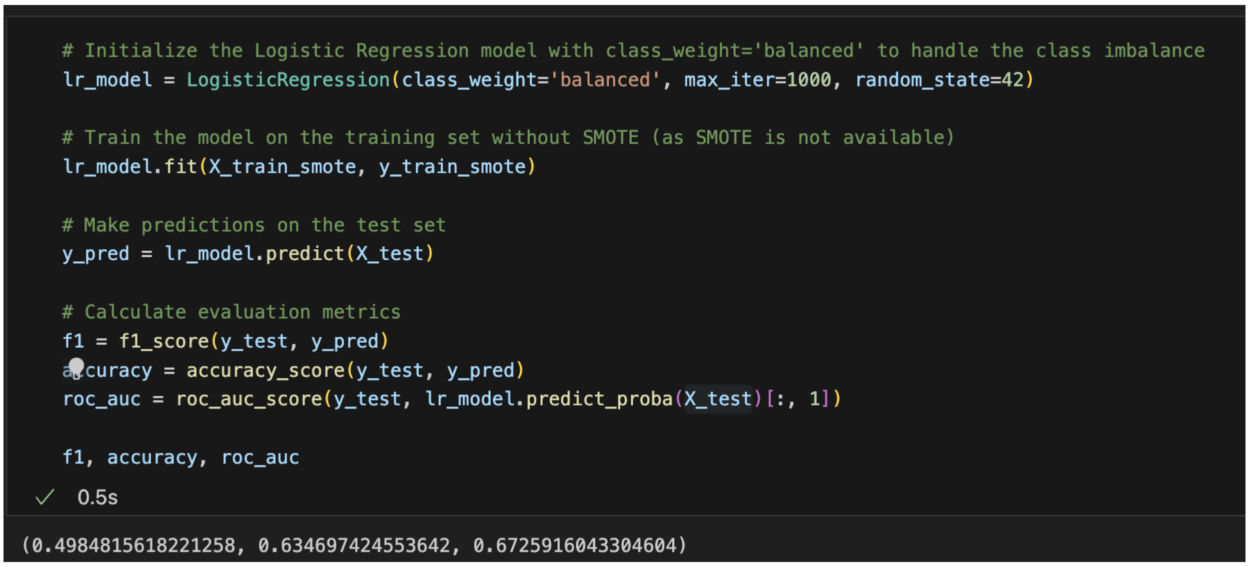
Logistic Regression Models and results:

F1: 0.4984815618221258

Accuracy: 0.634697424553642

ROC-AUC Score: 0.6725916043304604

The F1-score, a balance between precision and recall, is a moderate 0.498, indicating room for improvement in the model's performance. The accuracy is approximately 63.46%, which is not very high, considering that accuracy can be misleading when classes are imbalanced. The ROC-AUC score, which measures the model's ability to distinguish between the classes, is 0.672, suggesting that the model's predictive power is better than random guessing but still not excellent.



```
# Initialize the Logistic Regression model with class_weight='balanced' to handle the class imbalance
lr_model = LogisticRegression(class_weight='balanced', max_iter=1000, random_state=42)

# Train the model on the training set without SMOTE (as SMOTE is not available)
lr_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test set
y_pred = lr_model.predict(X_test)

# Calculate evaluation metrics
f1 = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, lr_model.predict_proba(X_test)[:, 1])

f1, accuracy, roc_auc
```

✓ 0.5s

```
(0.4984815618221258, 0.634697424553642, 0.6725916043304604)
```

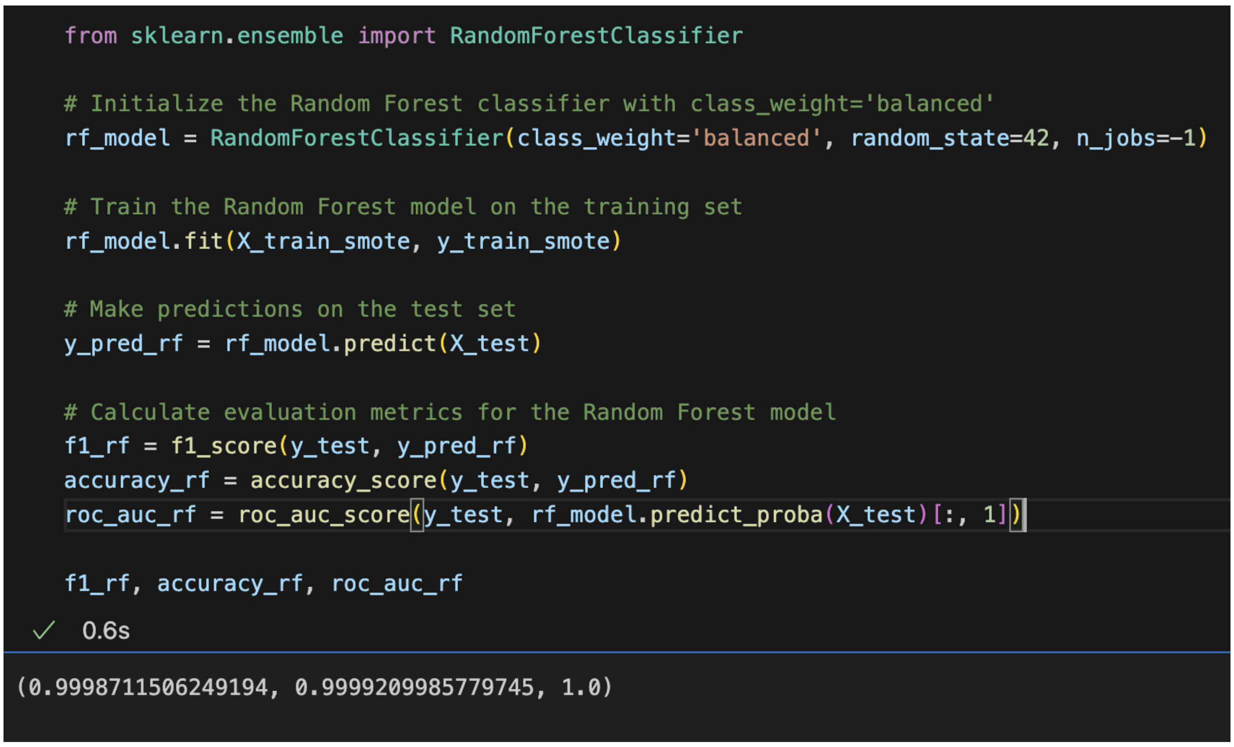
Random Forrest Classification Model and results:

F1: 0.9998711506249194

Accuracy: 0.9999209985779745

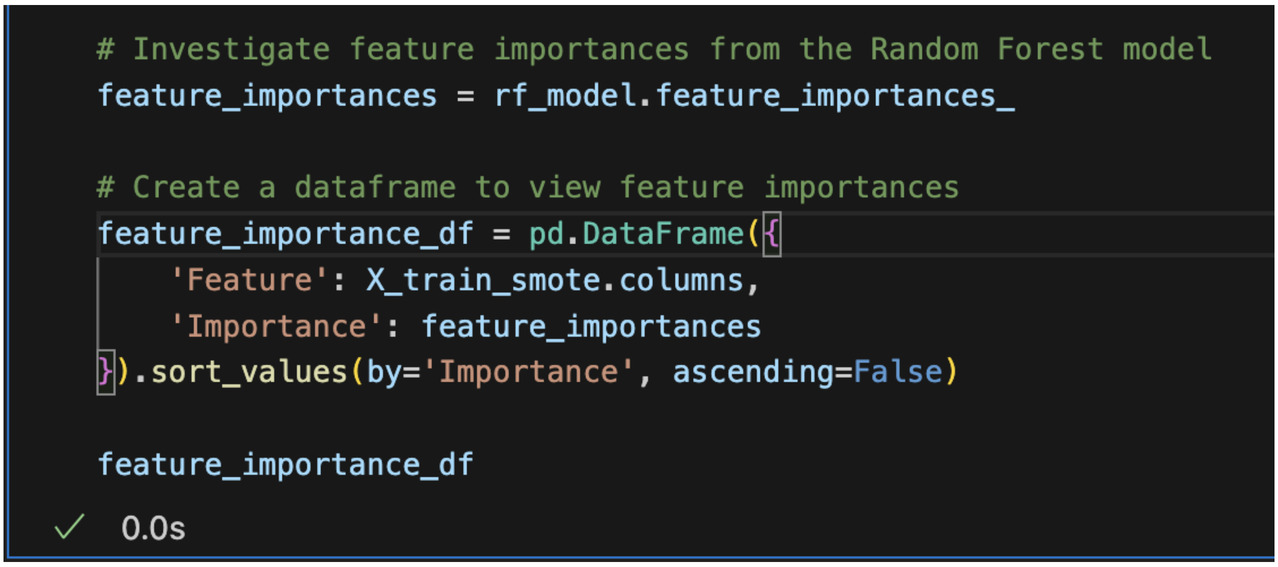
ROC-AUC Score: 1.0

These results are unusually perfect and suggest that there might be data leakage or overfitting. Data leakage occurs when information from outside the training dataset is used to create the model, leading to overly optimistic performance estimates. Overfitting happens when the model learns the training data too well, including noise and outliers, which does not generalize well to unseen data.



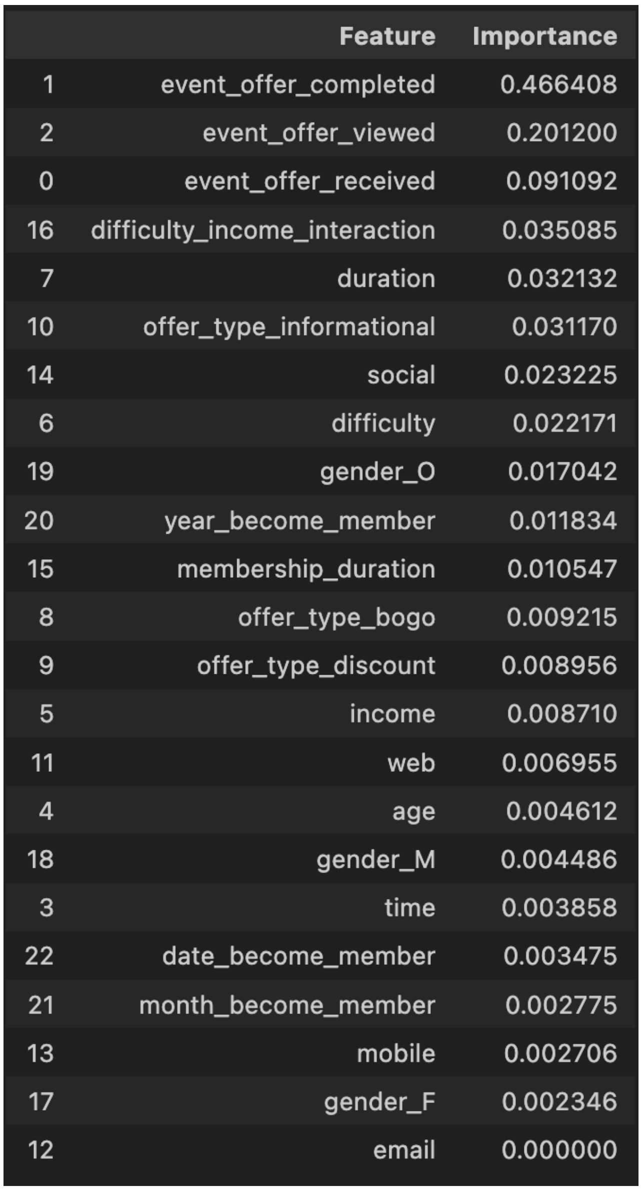
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest classifier with class_weight='balanced'
rf_model = RandomForestClassifier(class_weight='balanced', random_state=42, n_jobs=-1)

# Train the Random Forest model on the training set
rf_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Calculate evaluation metrics for the Random Forest model
f1_rf = f1_score(y_test, y_pred_rf)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
roc_auc_rf = roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1])

f1_rf, accuracy_rf, roc_auc_rf
✓ 0.6s

(0.9998711506249194, 0.9999209985779745, 1.0)
```

Let's investigate the features for potential data leakage and ensure they're all features that would be available at the time of making a prediction in a real-world setting. We'll also look at the feature importances from the Random Forest model to see if any feature stands out as overly influential.

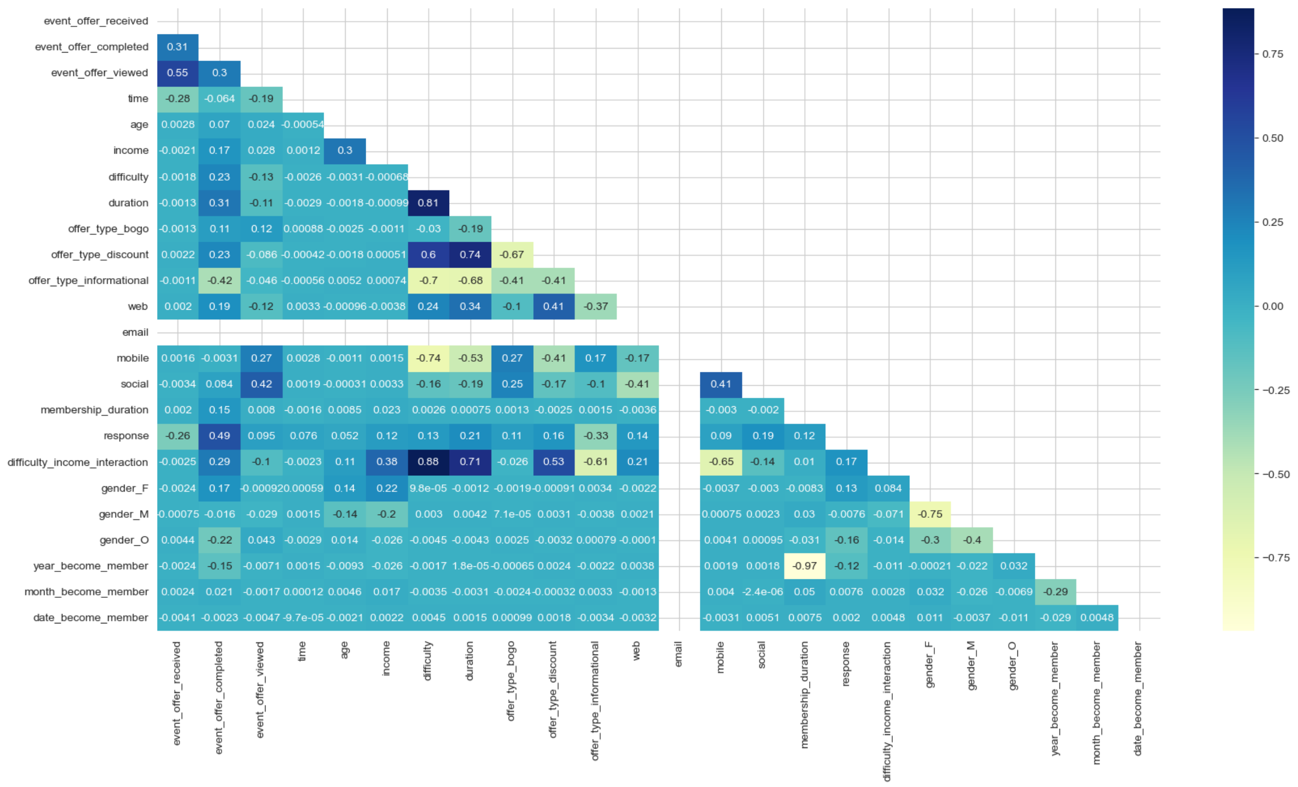


```python
# Investigate feature importances from the Random Forest model
feature_importances = rf_model.feature_importances_

# Create a dataframe to view feature importances
feature_importance_df = pd.DataFrame({
    'Feature': X_train_smote.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

feature_importance_df
```

✓ 0.0s



	Feature	Importance
1	event_offer_completed	0.466408
2	event_offer_viewed	0.201200
0	event_offer_received	0.091092
16	difficulty_income_interaction	0.035085
7	duration	0.032132
10	offer_type_informational	0.031170
14	social	0.023225
6	difficulty	0.022171
19	gender_O	0.017042
20	year_become_member	0.011834
15	membership_duration	0.010547
8	offer_type_bogo	0.009215
9	offer_type_discount	0.008956
5	income	0.008710
11	web	0.006955
4	age	0.004612
18	gender_M	0.004486
3	time	0.003858
22	date_become_member	0.003475
21	month_become_member	0.002775
13	mobile	0.002706
17	gender_F	0.002346
12	email	0.000000

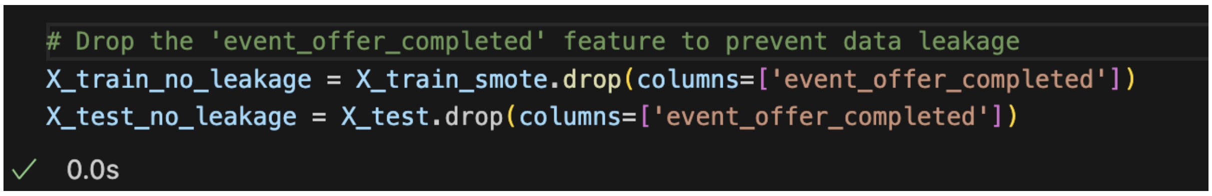


The feature importance from the Random Forest model (0.46) & the correlation chart value (0.49) indicate that the features **event_offer_completed** is the most important in predicting the target variable (**response**). This is a strong indicator of data leakage because this feature are directly related to the target we're predicting – whether a customer responds to an offer. Specifically:

- **event_offer_completed**: This feature indicates whether an offer has been completed, which directly correlates with the target variable.

This feature should not be used an input to the model because it essentially gives away the answer the model is trying to predict. The high importance of this feature is why the model performance is unrealistically perfect.

We need to remove it and retrain the model without it. The revised model will then provide a more realistic evaluation of its performance. Let's proceed with that.
With the removal of the **event_offer_completed** feature to prevent data leakage.

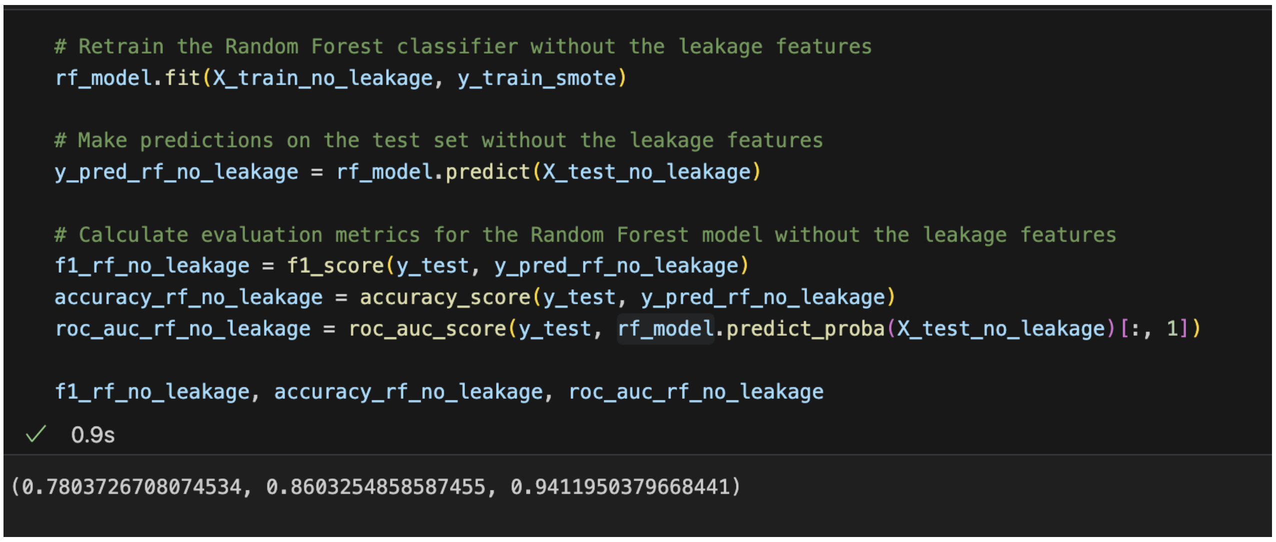


```
# Drop the 'event_offer_completed' feature to prevent data leakage
X_train_no_leakage = X_train_smote.drop(columns=['event_offer_completed'])
X_test_no_leakage = X_test.drop(columns=['event_offer_completed'])
✓ 0.0s
```

The Random Forest classifier's performance is more realistic:

F1-score: 0.7803726708074534
Accuracy: 0.8603254858587455
ROC-AUC Score: 0.9411950379668441

These results are indicative of a model that is performing well, especially considering the ROC-AUC score, which shows a good ability to differentiate between the classes. The F1-score and accuracy have decreased compared to the previous perfect scores, which is expected after removing the features that were causing data leakage.



```
# Retrain the Random Forest classifier without the leakage features
rf_model.fit(X_train_no_leakage, y_train_smote)

# Make predictions on the test set without the leakage features
y_pred_rf_no_leakage = rf_model.predict(X_test_no_leakage)

# Calculate evaluation metrics for the Random Forest model without the leakage features
f1_rf_no_leakage = f1_score(y_test, y_pred_rf_no_leakage)
accuracy_rf_no_leakage = accuracy_score(y_test, y_pred_rf_no_leakage)
roc_auc_rf_no_leakage = roc_auc_score(y_test, rf_model.predict_proba(X_test_no_leakage)[:, 1])

f1_rf_no_leakage, accuracy_rf_no_leakage, roc_auc_rf_no_leakage
```

✓ 0.9s

```
(0.7803726708074534, 0.8603254858587455, 0.9411950379668441)
```

To further enhance the model's performance, we can consider hyperparameter tuning with **RandomizedSearchCV**. This involves adjusting the settings of the model to find the optimal combination for the best performance.

Here is the Random Forrest Classifier results after tunning:

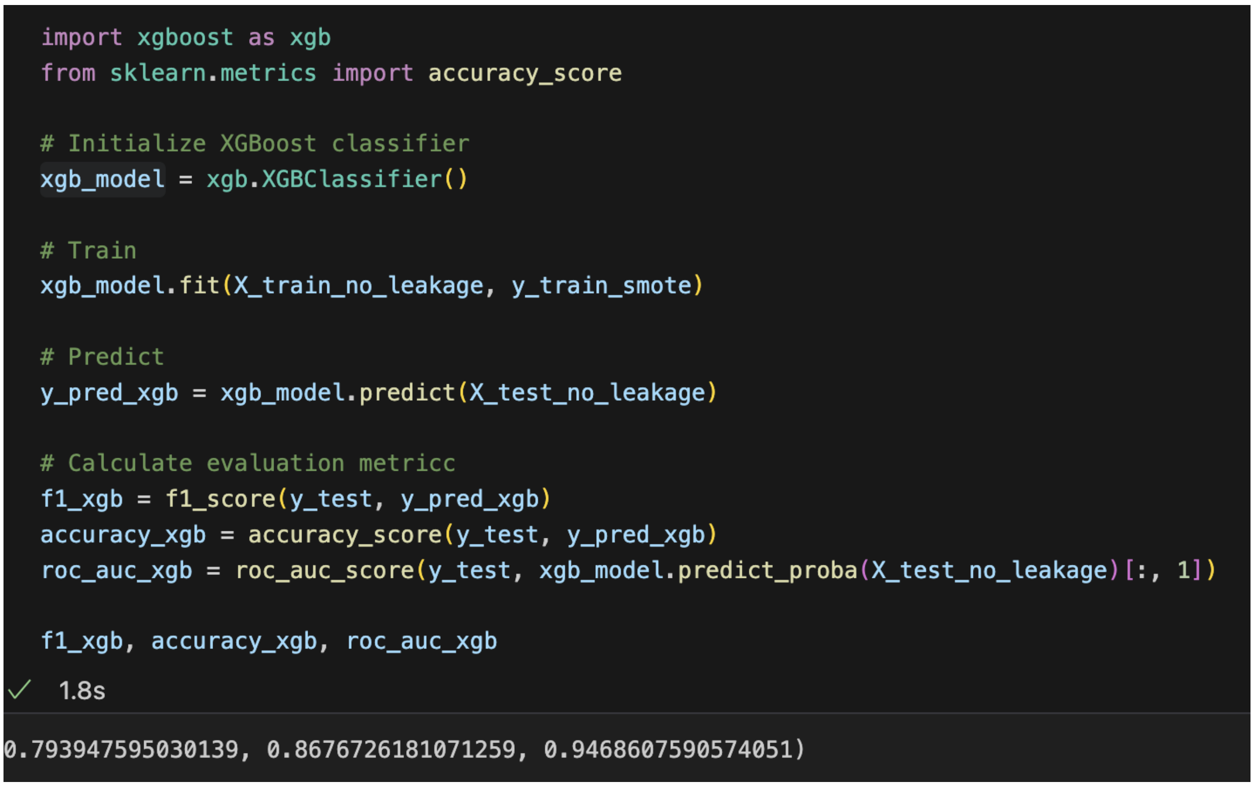
Best params: ({'n_estimators': 100, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 15}

F1-score: 0. 799810919404396
Accuracy: 0. 8661715910886396
ROC-AUC Score: 0. 9473598411110391

There is a bit increased in result from 0.780 to 0.799 for F1 score, 0.860 to 0.866 for Accuracy and ROC-AUC score from 0.941 to 0.947.

We are doing the same approach by applying **XGBoost classification model** and here is the result:

F1-score: 0.793947595030139
Accuracy: 0.8676726181071259
ROC-AUC Score: 0.9468607590574051



```
import xgboost as xgb
from sklearn.metrics import accuracy_score

# Initialize XGBoost classifier
xgb_model = xgb.XGBClassifier()

# Train
xgb_model.fit(X_train_no_leakage, y_train_smote)

# Predict
y_pred_xgb = xgb_model.predict(X_test_no_leakage)

# Calculate evaluation metricc
f1_xgb = f1_score(y_test, y_pred_xgb)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
roc_auc_xgb = roc_auc_score(y_test, xgb_model.predict_proba(X_test_no_leakage)[:, 1])

f1_xgb, accuracy_xgb, roc_auc_xgb
```

```
✓ 1.8s
```

```
0.793947595030139, 0.8676726181071259, 0.9468607590574051)
```

XGBoost model result after hyperparameter tunning with RandomSearchCV:

Fitting 3 folds for each of 25 candidates, totaling 75 fits

Best params: ({'n_estimators': 100, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 15}

F1-score: 0.799810919404396
Accuracy: 0.8661715910886396
ROC-AUC Score: 0.9473598411110391

Results

Here is the summary result of different machine learning models.

Model	F1	Accuracy	ROC-AUC Score
Logistic Regression	0.4985	0.6347	0.6726
Random Forrest	0.9999	0.9999	1.0000
Random Forrest (no data leakage)	0.7804	0.8603	0.9412
Tunned Random Forrest	**0.7998**	0.8662	**0.9474**
XGBoost	0.7939	**0.8677**	0.9469
Tunned XGBoost	**0.7998**	0.8662	**0.9474**

The results show a comparison of several classification models on a dataset, with metrics including F1 score, Accuracy, and ROC-AUC score.

- **Logistic Regression**: This model has the lowest F1 score (0.4985), accuracy (0.6347), and ROC-AUC score (0.6726) among all models tested, indicating it may not be well-suited to the dataset or might require further parameter tuning or feature engineering to improve its performance.
- **Random Forest:** The perfect scores (F1 score and accuracy of 0.9999 and ROC-AUC score of 1.0000) are indicative of data leakage, where the model had access to information during training that it wouldn't have during a real-world prediction scenario. This leads to an unrealistically high estimation of performance.
- **Random Forest (no data leakage)**: Once data leakage is addressed, the performance metrics drop significantly but still show strong performance with an F1 score of 0.7804, accuracy of 0.8603, and ROC-AUC score of 0.9412. This indicates that the Random Forest model is robust and performs well on the dataset without overfitting.
- **Tuned Random Forest**: After tuning, there's a slight improvement across all metrics compared to the Random Forest model without data leakage. This suggests that parameter tuning helped the model to better generalize.
- **XGBoost**: The XGBoost model's performance is competitive, with an F1 score of 0.7939, accuracy of 0.8677, and ROC-AUC score of 0.9469. It performs slightly better than the non-tuned Random Forest model, indicating it's a good candidate for this problem.
- **Tuned XGBoost**: The tuned XGBoost matches the performance of the tuned Random Forest in terms of F1 score and accuracy but has a marginally lower ROC-AUC score.

This suggests that both tuned models are strong contenders and tuning has effectively improved their performance.

In conclusion, tuning the models has resulted in improved performance, with the Random Forest and XGBoost models showing strong potential for this classification problem after tuning. Logistic Regression appears to be the weakest model for this dataset, and the initial perfect scores for Random Forest indicate a need to ensure that data leakage is prevented to maintain model validity.

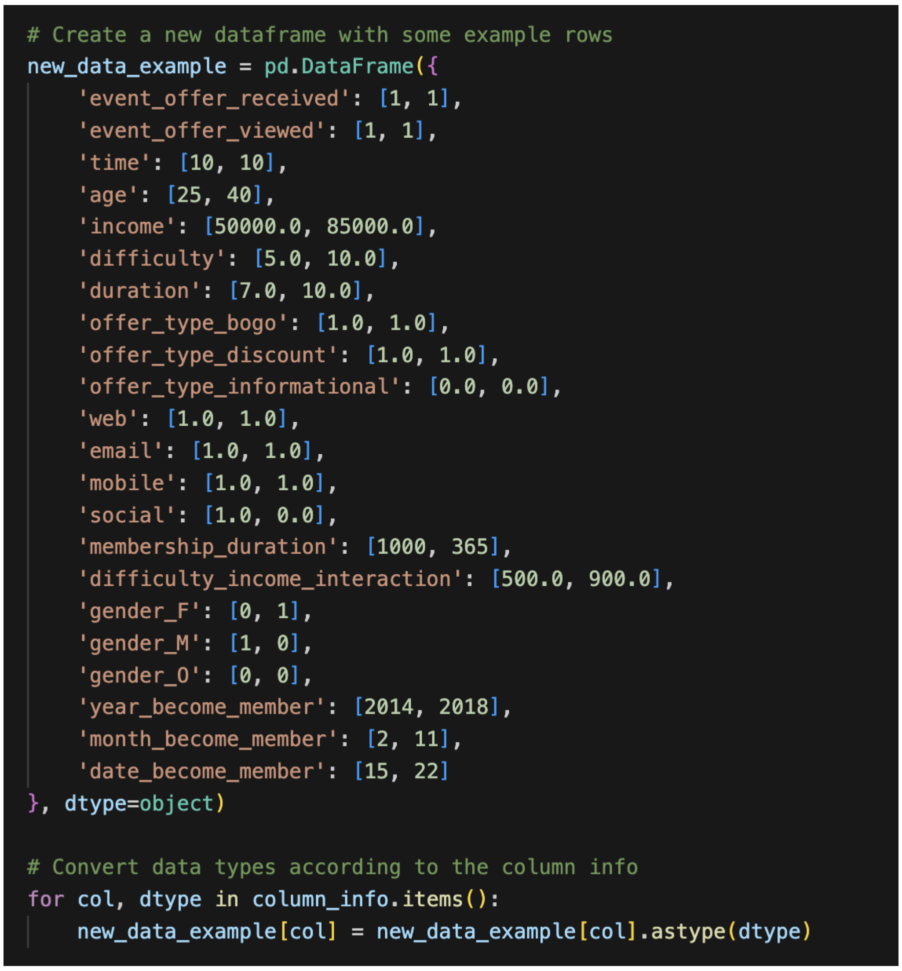
Conclusion

The project successfully demonstrated the use of machine learning to predict customer response to promotional offers. The tuned Random Forest and XGBoost models emerged as the most promising, showing potential for deployment in a real-world scenario. Logistic Regression was found to be less effective for this dataset. The importance of avoiding data leakage was highlighted, as it could lead to overestimation of a model's performance.

The project's outcome provides a solid foundation for Starbucks to implement a data-driven marketing strategy, where promotional offers are tailored based on predictive insights, potentially leading to increased efficiency in marketing resource allocation and enhanced customer satisfaction.

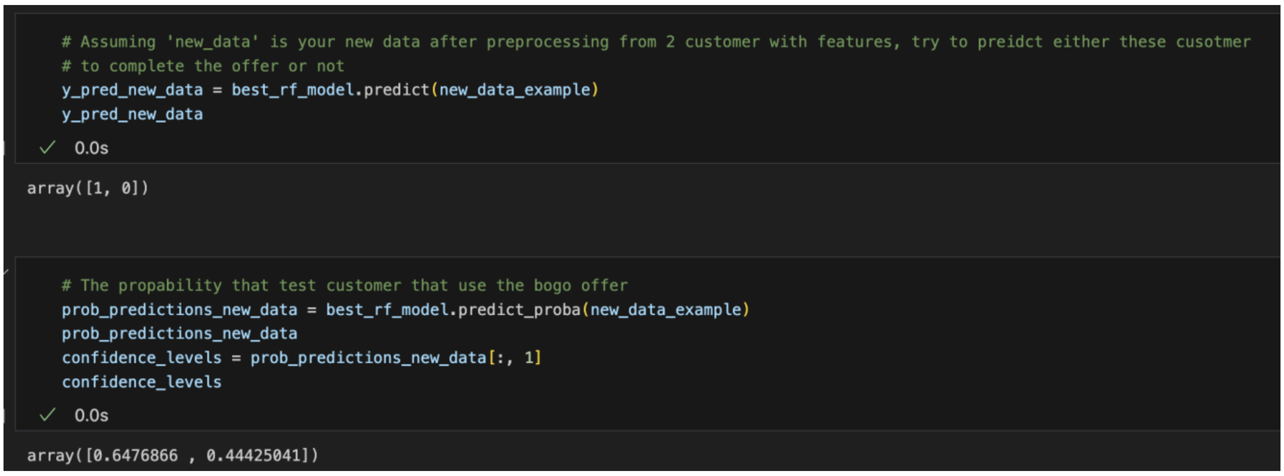
Appendix

Try to apply a random customer information to predict the probability of responding offers



```python
# Create a new dataframe with some example rows
new_data_example = pd.DataFrame({
    'event_offer_received': [1, 1],
    'event_offer_viewed': [1, 1],
    'time': [10, 10],
    'age': [25, 40],
    'income': [50000.0, 85000.0],
    'difficulty': [5.0, 10.0],
    'duration': [7.0, 10.0],
    'offer_type_bogo': [1.0, 1.0],
    'offer_type_discount': [1.0, 1.0],
    'offer_type_informational': [0.0, 0.0],
    'web': [1.0, 1.0],
    'email': [1.0, 1.0],
    'mobile': [1.0, 1.0],
    'social': [1.0, 0.0],
    'membership_duration': [1000, 365],
    'difficulty_income_interaction': [500.0, 900.0],
    'gender_F': [0, 1],
    'gender_M': [1, 0],
    'gender_O': [0, 0],
    'year_become_member': [2014, 2018],
    'month_become_member': [2, 11],
    'date_become_member': [15, 22]
}, dtype=object)

# Convert data types according to the column info
for col, dtype in column_info.items():
    new_data_example[col] = new_data_example[col].astype(dtype)
```

We use the best tuned random forest to make a prediction. The first customer has 64.76 % to response to the offer compared to the second one has 44.42%. Therefore, model predicts “1" as response to the first one and “0" as non-response to the second one.



```python
# Assuming 'new_data' is your new data after preprocessing from 2 customer with features, try to preidct either these cusotmer
# to complete the offer or not
y_pred_new_data = best_rf_model.predict(new_data_example)
y_pred_new_data
```

✓ 0.0s

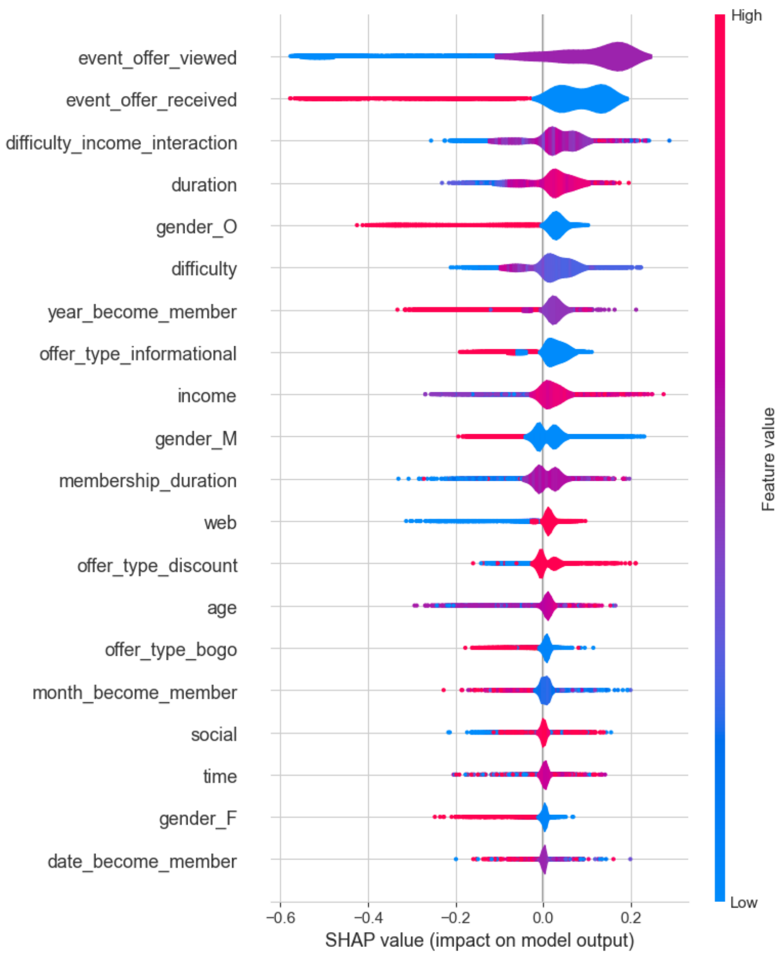
```
array([1, 0])
```

```python
# The propability that test customer that use the bogo offer
prob_predictions_new_data = best_rf_model.predict_proba(new_data_example)
prob_predictions_new_data
confidence_levels = prob_predictions_new_data[:, 1]
confidence_levels
```

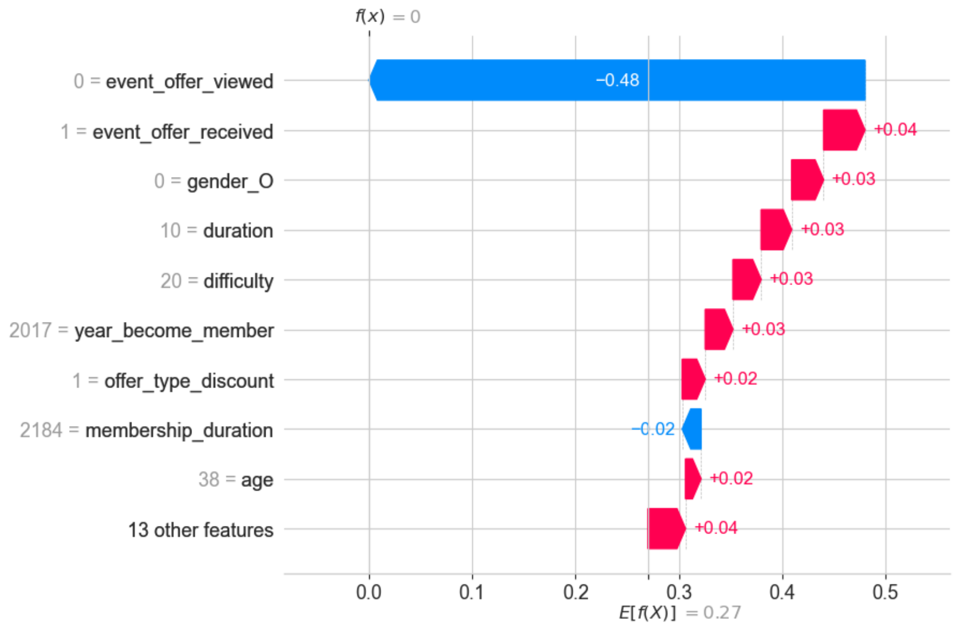
✓ 0.0s

```
array([0.6476866 , 0.44425041])
```

SHAP (SHapley Additive exPlanations) summary plot



Waterfall SHAP plotted in one record



f(x) = 0
0 = event_offer_viewed
1 = event_offer_received
0 = gender_O
10 = duration
20 = difficulty
2017 = year_become_member
1 = offer_type_discount
2184 = membership_duration
38 = age
13 other features
−0.48
+0.04
+0.03
+0.03
+0.03
+0.03
+0.02
−0.02
+0.02
+0.04
0.0
0.1
0.2
0.3
0.4
0.5
E[f(X)] = 0.27