Starbucks Capstone Challenge

Domain Background

Companies involved in selling products either online or through physical stores must critically think about **customer engagement** and **customer spend over time (a.k.a. customer lifetime value)**. The Starbucks capstone project falls under this umbrella. Starbucks provides great coffee drinks to its customers, and the company continues to evolve as the market shifts.

The Starbucks capstone project tackles a challenge that many companies and brands face-**customer acquisition, customer retention, and revenue maximization**. Essentially, revenue growth YoY, acquiring new customers, and retaining existing customers are a few important pillars in every company's marketing efforts.

More specifically, Starbucks' use of offers to provide discounts and provide more information about the brand is one way to engage their customers positively. The company can create a layered strategy to maximize customer acquisition, customer retention, and revenue growth. As a result, this approach of engaging customers has a universal appeal, and other companies can reproduce the application of this strategy.

Problem Statement

- **Goal:**
  - Create demographic cohorts to determine what group of customers will respond best to which offer type or set of offer types using the available raw transaction, profile, and offers data.
- **Insights:**
  - Number of customers who have completed offers is relatively high with 77% of all converted customers have used offers.
  - 97.5% of customers in profile have converted at least once. Conversion isn't the problem, but benefits of offers might be looked at deeper. Does it increase spend?
  - Although 77% of customers have at least one offer completed, the percentage of transactions with offer completed stands at about 22%.
  - Average transaction size with offer completed is overall higher than transactions with no offer completed.
  - Takeaway: A lot of customers have been exposed and used offers at least once at some point, but only about quarter of transactions are associated with offer-completed.
- **Opportunity:**
  - Although less than a quarter of the transactions have a completed offer or completed offers, data shows that completed offers generate higher average transaction size. At the same time, money is given away through BOGO and discount offers to entice customers to spend. Over the long run, the strategy contributes to a larger lifetime value of the customer and more than make up for the rewards given to customers.
  - Therefore, with the right balance (determining customer cohorts) and targeting specific offers to more appropriate groups driven by data can achieve a higher ROI. In this case, we are looking a longer customer retention and higher customer lifetime value.
- **Approach:**
  - **Combine** transaction, demographic and offer data to determine which demographic groups respond best to which offer type.
  - **Build** a model that predicts whether or not someone will respond to an offer.
  - **Build** a model that predicts how much someone will spend based on demographics and offer type.
  - **Develop** a set of heuristics that determine what offer you should send to each customer.

Datasets and Inputs

Datasets

Offers Funnel Analysis EDA [Dataset 1]

- **Script:** offers_funnel_analysis.py
- **Data Output:** /data/starbucks_offers_funnel_analysis.csv.gz
- Using raw transcript data, a offer funnel conversion view was created, where offer received being the top layer.
- This aggregated funnel view provides important insights into the overall offers' performance and how customers are engaging with them. This analysis shows that there are two BOGO standouts (rows 2 & 9) as well as two discount standouts (rows 6 & 7). One of the informational offers has much higher view rate (row 8) than the other.
- This custom dataset and the insights help develop deeper intuition into the Starbucks' data. The insights will help in engineering feature in the model building phase of the project.

reward	channels	difficulty	duration	offer_type	offer_id	viewed_rate	completion_rate	avg_spend_per_customer	avg_transactions_per_customer
10	[email, mobile, social]	10	168	bogo	ae264e3637204a6fb9bb56bc8210ddfd	87.70%	48.16%	$27.73	1.14
10	[web, email, mobile, social]	10	120	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0	96.11%	43.87%	$27.48	1.14
0	[web, email, mobile]	0	96	informational	3f207df678b143eea3cee63160fa8bed	54.40%	0.00%	$0.00	0.00
5	[web, email, mobile]	5	168	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9	54.33%	56.71%	$20.59	1.14
5	[web, email]	20	240	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7	34.73%	44.60%	$29.52	1.12
3	[web, email, mobile, social]	7	168	discount	2298d6c36e964ae4a3e7e9706d1fb8c2	95.96%	67.43%	$20.48	1.16
2	[web, email, mobile, social]	10	240	discount	fafdcd668e3743c1bb461111dcafc2a4	96.45%	69.99%	$21.61	1.16
0	[email, mobile, social]	0	72	informational	5a8bc65990b245e5a138643cd4eb9837	87.78%	0.00%	$0.00	0.00
5	[web, email, mobile, social]	5	120	bogo	f19421c1d4aa40978ebb69ca19b0e20d	95.95%	56.74%	$22.34	1.14
2	[web, email, mobile]	10	168	discount	2906b810c7d4411798c6938adc9daaa5	53.96%	52.63%	$23.76	1.14

Transaction Engagement Data [Dataset 2]

- **Script:** transaction_engagement.py
- **Data Output:** /data/starbucks_transaction_engagement.csv.gz
- A custom transaction-level dataset was needed to deep dive into the impacts of offers on customers' purchasing behavior and spending.
- In order to build out this view, the transcript dataframe had to be flattened so that every transaction would have all the related offer activities attached to the transaction as column attributes. I generated transaction_id's using uuid.uuid().hex as well as creating record_id's to group multiple offers completed per transaction into one trackable identifier (record_id).
- With this view, slicing and dicing the data and extracting insights to create new features become much easier and efficient.
- Although only 22% of transactions have offer completed, when comparing transaction average order size (AOS) with offer completed versus without offer completed, you see a noticeable lift in the transaction AOS with offer completed.
- The trade-off for customer spending more to meet the difficulty threshold is getting the reward dollar amount and, in the short run, the impact on net revenue might not be impactful; however, in the long run, offers will potentially drive higher retention and higher spend during the lifetime of the customer.
- Furthermore, by leveraging data-driven calculations and insights, the possibility of driving up the number of transactions with offer completed is very much achievable.
- Average transaction size with offer completed is 2x the average transaction size without offer completed. At a glance, this looks fairly significant. **($22.47 / $10.70) = 2.1x**

Bin Name	Transaction Amount	Number of Transactions	Average Transaction Size
Transactions with Offer Completed	$687,925	30,617	$22.47
Transactions without Offer Completed	$1,158,828	108,336	$10.70
Lift	--	--	2.1x

Inputs

Customer-level Aggregation and Feature Engineering

- Using the **transaction engagement dataset** coupled with **custom data transformation class**, I was able to put together all the relevant features needed to complete this project.
  - **Transaction engagement data**: transaction_engagement.py
  - **Custom data transformation class functions**: /base_transforms/base_transforms_df.py
  - **Feature engineering notebook:** Starbucks_Capstone_Feature Engineering_Notebook.ipynb
- The dataset is at the customer_id level. In order to prevent data leakage, the customer_id's were split into X_train and X_test with 77% to 33% split. Feature aggregation occurred using each dataset while any feature engineering requiring th calculation of median and creating relevant groupings, for example, the X_train dataset was used and applied to the X_test dataset.
- The end result of the preprocessing step created two output files.
  - **X_train:** /data/train_starbucks.csv.gz
    - Shape: (11390, 53)
    - Dataset includes customer_id and target variable (transaction_aos)
  - **X_test:** /data/test_starbucks.csv.gz
    - Shape: (5610, 53)
    - Dataset includes customer_id and target variable (transaction_aos)

Feature List

Feature Name	Feature Type	Data Type	Description
customer_id	uuid	string	Unique identifier per customer
gender	binary	string	Describes the sex of the customer
age	continuous	integer	Numeric age of the customer
age_quantile_range	interval	string	5 age quantile ranges (e.g., [$0-$20k])

Feature Name	Feature Type	Data Type	Description
age_quantile_label	interval	string	5 age quantile labels (e.g., 0-20Q)
date_registered	date	date	Date when customer became a Starbucks member with format YYYY-MM-DD
days_registered	continuous	integer	Represents how long a customer has been a member in days from the max available registered date in the train dataset
days_reg_quantile_range	interval	string	10 days registered quantile ranges
days_reg_quantile_label	interval	string	10 days registered quantile labels
income	continuous	float	Numeric value representing the customer's annual income
income_quantile_range	interval	string	5 income quantile ranges
income_quantile_label	interval	string	5 income quantile labels
transaction_amount	continuous	float	Represents total summed amount of customer's transactions
transaction_cnt	continuous	float	Represents total number of transactions per customer
transaction_aos	continuous	float	Represents the average transaction size (transaction_amount / transaction_cnt) per customer
gender_NA	binary	integer	Binary flag indicates customer's gender is missing
income_NA	binary	integer	Binary flag indicates customer's income is missing
age_NA	binary	integer	Binary flag indicates customer's age is missing
num_offer_received	continuous	integer	Total number of offers received by a customer
num_bogo_offer_received	continuous	integer	Total number of BOGO offers received by a customer
num_info_offer_received	continuous	integer	Total number of informational offers received by a customer
num_discount_offer_received	continuous	integer	Total number of discount offers received by a customer
num_offer_viewed	continuous	integer	Total number of offers viewed by a customer
num_offer_completed	continuous	integer	Total number of offers completed by a customer
num_offer_completed_viewed	continuous	integer	Total number of offers completed where offers were viewed by the customer
num_offer_completed_not_viewed	continuous	integer	Total number of offers completed where offers were not viewed by the customer
num_transactions_no_oc	continuous	integer	Total number of transactions without any completed offers per customer
num_transactions_oc_direct	continuous	integer	Number of transactions with completed offers where the transaction amount was equal or greater than the offer difficulty treshold amount
percent_oc_direct_transactions	continuous	float	Percentage of transactions with completed offers where the transaction amount was equal or greather than the offer difficulty threshold amount
num_transactions_oc_indirect	continuous	integer	Number of transactions with completed offers where the transaction amount was less than the offer difficulty threshold amount, which is an indication of cummulated spends meeting the difficulty threshold over time
avg_offered_received_freq	continuous	float	On average, how often does the customer receive an offer (in hours)
info_view_rate	continuous	float	Number of informational offers viewed divided by number of informational offers received
offer_view_rate	continuous	float	Number of offers viewed divided by number of offers received
offer_completion_rate	continuous	float	Number of offers completed divided by number of offers received minus number of informational offers received
total_reward_amount	continuous	float	Total reward amount the customer has acquired by completing offers
avg_reward_per_oc_transaction	continuous	float	Among all transactions with completed offers, calculate the average reward acquired per transaction
transaction_oc_amount	continuous	float	Total transaction amount with completed offers per customer
transaction_aos_oc	continuous	float	Average order size with completed offer transactions
transaction_no_oc_amount	continuous	float	Total transaction amount with transactions without completed offers
transaction_aos_no_oc	continuous	float	Average order size of transactions without completed offers
num_bogo_offer_viewed	continuous	integer	Number of BOGO offers viewed per customer
num_info_offer_viewed	continuous	integer	Number of informational offers viewed per customer
num_discount_offer_viewed	continuous	integer	Number of discount offers viewed per customer
num_bogo_offer_completed	continuous	integer	Number of BOGO offers completed per customer
num_discount_offer_completed	continuous	integer	Number of discount offers completed per customer
median_offer_duration	continuous	float	Calculate the median offer duration based on all the offers received per customer
avg_offer_completion_time	continuous	float	sum(offer_completed_time - offer_received_time) / count(offer_completed); Unit is in hours
avg_hrs_bw_transactions	continuous	float	Calculate the average time in hours between transactions if the customer has more than one transaction
num_oc_ch_web	continuous	integer	Number of completed offers where it was advertised through the web channel
num_oc_ch_social	continuous	integer	Number of completed offers where it was advertised through the social channel
num_oc_ch_mobile	continuous	integer	Number of completed offers where it was advertised through the mobile channel
num_oc_ch_email	continuous	integer	Number of completed offers where it was advertised through the email channel

- During the feature selection process, important features will be identified and those features will be used to assist in creating relevant customer cohorts.

Solution Statement

Goal

- Develop a framework to create customer cohorts where each cohort will be defined by a set of important features, and custom ranking formula will generate a list of ranked offers most engaged by the cohort.
  1. More specifically, the **feature engineering stage** converts raw data into features.
  2. These features are plugged into a **classification model** and reduced to features with predicting if a customer would use an offer or not.
  3. Next, the **regression model** will be trained to predict average customer spend per transaction and average spends will be binned into five categories (Low, Medium-Low, Medium, Medium-High, High), where this will be combined with selected features from above to create cohorts.
  4. Finally, a **custom ranking formula - Maximum Contribution Estimation (MCE) -** is used to rank which offers are most contributing for each cohort.

Approach

1. Feature Engineering: Combine raw data to create datasets for analysis and feature engineering.

- Two main datasets created to focus on extracting insights, and simplify complex aggregation and feature engineering steps.
- Datasets:
  - Transaction-level dataset: **transaction_engagement.py**
  - Customer-level dataset: **Starbucks_Capstone_Feature Engineering_Notebook.ipynb**
    - Train set: /data/train_starbucks.csv.gz
    - Test set: /data/test_starbucks.csv.gz

2. Feature Importance: Supervised learning classification model used to determine important features associated with customers using offers. Target is binary - used offers (1) or did not used offers (0). Various supervised classification models will be explored and select the most appropriate model for this challenge.

- Datasets: **train_starbucks.csv.gz**, **test_starbucks.csv.gz**
- Although predictions about who will use the offers are valuable, this will not be the main goal. Even if the model identifies who will use the offers or not, it makes more sense to target every customer with different offer types.
- The main objective is discovering most important features, in which these selected features will be used to define the customer cohorts.
- Because there are different scenarios where offers are completed, it's essential that a strict set of rules are created to define what it means to have completed offers. We do not want weak signals associate with offer completions.
- **Offer completion criteria (Defining the target variable):**
  - If **offer_view_rate >= 50%** and **offer_completion_rate >= 50%**, then assign instance with 1 (has_offer_completed).
  - If the classification model does not perform well against test dataset, the criteria will become stricter with these additional conditions:
    - **num_transactions_oc_direct** > **num_transactions_oc_indirect**
    - **num_offer_completed_viewed** > **num_offer_completed_not_viewed**
- By following this criteria, we are strengthening the signal correlating with offer completions. Remember that 77% of customers have used offers, but only 22% of transactions have completed offers. As a result, by following the criteria, we are carving out the customers from the 77% only the ones that have truly engaged with offers in a meaningful way.
- Without a robust criteria to define the target dependent variable, the training dataset would have 77% of the instances with majority class of 1, where each instance would have varying signals correlating with what we are trying to predict. Also, we are balancing the imbalanced train dataset as well as creating a meaningful target variable.
- **Evaluation metrics:** ROC-AUC, PR AUC, precision, recall, F1 score, G-mean, and accuracy.

3. Regression Model: Predict spend per customer and create bins to bucket them into low, med-low, medium, med-high, and high category.

- Datasets: **train_starbucks.csv.gz**, **test_starbucks.csv.gz**
- Once important features are identified through the classification model, the nex step is to create a regression model to predict customer average spend per transaction. The same datasets used for classification will be modified to train a regression model.
- The goal of the regression model is to predict customer's spend based on existing customers and place those predicted spends into bins. **I am proposing five bins, Low, Medium-Low, Medium, Medium-High, and High.**

- This categorical variable/parameter will be coupled with important features (identified from the Feature Importance step) to define more granular cohorts.
- **Evaluation metrics:** mean square error (MSE) or root mean square error (RMSE)

4. Custom Ranking Formula (I'm calling this method the Maximum Rank Contribution Estimation (MRCE)): The feature importance work and regression model outputs will be combined to create customer cohorts. Next, a custom ranking formula will rank which offer types are relatively larger contributor to a particular cohort. Contribution is defined in the form of "rank contribution score" that I will define later. Basically, higher the rank contribution score per offer, higher the contribution or engagement to the cohort.

- In this stage, important features will be used to group the customers.
- Furthermore, the predicted spends will be used per customer_id and averaged based on the defined features.
- The ranking formula will rank which offer types are most impactful and ranked from high to low using rank constirbution scores.
- **Evaluation metrics:** Rank contribution scores and taking the maximum from the list of rank contribution scores to identify the number 1 ranked offer.

Benchmark Model

Basline Analysis

- There is no clear benchmark model in the wild that can be used to understand Starbucks' offers challenge. However, based on the problem statement, we could use the existing raw data to create baseline metrics to quantify the impact of offers on customers' spend behavior. One way to assess the impact is the following:
  - Measure if the average order size per customer is significantly different between customers who converted with offers (treatment) and customers who converted withtout offers (baseline). What does this show us? This will show if offers have an impact on their spends and if the increase or decrease in spend is significant.
- To generate naive baseline models, I'll be using zero rule algorithm to measure the baseline accuracy of the classification model and measure the baseline mean-square-error value for the regression model.

Approach

- **2-sample t-test:** Measure the statistical difference between average spend using two-sample t-test for statistical significance. (Assume conditions are met: normal, independent, and random condition.)
  - These perspectives should give us a baseline view if offers actually work from a statistical standpoint as well as a directional standpoint. Although when the data is split in such a fashion, it might not reflect random sampling process. However, with a big enough sample in place I'm assuming the split samples are relatively random and adequate representation of the population base.
  - The splits will measure if average order size is incrementally higher when offers are used. At the same time, rewards received will be summed and quantified in the analysis below. This is the trade-off for Starbucks - giving free money away in the short term when difficulty thresholds are met with mix of customers who are really engaged with offers and customers who are not. Essentially, by giving away free stuff, the bet is that, in the long run, overall average spend and duration of spend will be higher and longer.
- **Classification baseline model using zero rule algorithm:** Basically, in the train set, calculate the majority class either "1" or "0", and use the majority class to predict all the instances in the test set. With the predicted values, accuracy will be calculated against the target variable in the test set.
- **Regression baseline model using zero rule algorithm:** In this case, the target variable is continuous. The regression will be trained to predict customer's average order size (AOS) or average transaction size. To create the naive baseline model for regression, the average transaction_aos will be calculated using the train set and then the error value will be calculated against the test set. Finally, the error values will be used to calculate the mean-square-error. This will be the regression baseline metric and the baseline evaluation metric.

In [130]:
```
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

%matplotlib inline
import os
import pandas as pd
import numpy as np
import scipy.stats as stats
pd.set_option('display.max_columns', None)
```

Two-Sample t-Test

In [193]:
```
# read in the train data
def summary_metrics():
    train_file = 'train_starbucks.csv.gz'
    train_file_gzip = os.path.join(os.getcwd(), 'data', train_file)
    train = pd.read_csv(train_file_gzip, compression='gzip')

    # create offers only train data based on "offer completion criteria"
    train_offers = train[(train['offer_view_rate'] >= 0.5) &
                         (train['offer_completion_rate'] >= 0.5)]
                         # (train['num_offer_completed_viewed'] >= train['num_offer_completed_not_viewed'])]
                         # (train['num_transactions_oc_direct'] > train['num_transactions_oc_indirect'])]
    train_offers['transaction_amount_minus_reward'] = train_offers['transaction_amount'] - train_offers['total_reward_amount']
    train_offers['transaction_aos_minus_reward'] = (1.0 * train_offers['transaction_amount_minus_reward']) / train_offers['transaction_cnt']

    # create no offers only data
    train_offers_filter = pd.DataFrame({'customer_id': train_offers.customer_id})
    train_offers_filter['is_engaged'] = 1
    train_no_offers = train.merge(train_offers_filter, how='left', on='customer_id')
    train_no_offers['is_engaged'].fillna(0, inplace=True)
    train_no_offers = train_no_offers[train_no_offers['is_engaged']==0]
    train_no_offers.drop(columns=['is_engaged'], inplace=True)
    train_no_offers['transaction_amount_minus_reward'] = train_no_offers['transaction_amount'] - train_no_offers['total_reward_amount']
    train_no_offers['transaction_aos_minus_reward'] = (1.0 * train_no_offers['transaction_amount_minus_reward']) / train_no_offers['transaction_cnt']
    train_no_offers['transaction_aos_minus_reward'].fillna(0, inplace=True)

    # create summary metrics
    df_summary = pd.DataFrame({'category': ['Customers with offers', 'Customers without offers'],
                  'sample_size': [train_offers.shape[0], train_no_offers.shape[0]],
                  'transaction_amount': [train_offers.transaction_amount.sum(), train_no_offers.transaction_amount.sum()],
                  'transaction_count': [train_offers.transaction_cnt.sum(), train_no_offers.transaction_cnt.sum()],
                  'total_reward_amount': [train_offers.total_reward_amount.sum(), train_no_offers.total_reward_amount.sum()],
                  'transaction_aos': [train_offers.transaction_aos.mean(), train_no_offers.transaction_aos.mean()],
                  'transaction_aos_minus_reward': [train_offers.transaction_aos_minus_reward.mean(),
                                                   train_no_offers.transaction_aos_minus_reward.mean()]
                 }).set_index('category')

    return df_summary

df_summary = summary_metrics()
df_summary
```

Out[193]:

category	sample_size	transaction_amount	transaction_count	total_reward_amount	transaction_aos	transaction_aos_minus_reward
Customers with offers	6345	1048447.98	61780.0	95667.0	19.689899	17.720517
Customers without offers	5045	192292.27	31443.0	14443.0	7.242906	6.640530

- In this summary view, transaction_aos and transaction_aos_minus_reward are used to assess if providing offers generates overall larger transaction average order size (aos) versus not having offers.
- The conditions, offer_view_rate >= 05 and offer_completion_rate >= 0.5, removes customers who are most likely benefitting from offers who are not aware of them or not interested in using them but still meet the spend threshold and thus automatically receiving the rewards.
- This analysis shows that customers who are already spending relatively high (transaction_aos_mius_reward) are the ones who are most engaged with offers and most likely to use offers during transactions.

In [192]:
```
# Conduct two-sample t-test to determine if transaction_aos values are statistically different
    # as well as transaction_aos_minus_reward values.
# We are assuming all conditions are met for this test:
    # (1) Sampling distribution is normally distributed
    # (2) Simple ransome sampling
    # (3) Samples are independent

def t_test_stats():
    offers = np.array(train_offers.transaction_aos)
    no_offers = np.array(train_no_offers.transaction_aos)
    t_stat_1, p_val_1 = stats.ttest_ind(offers, no_offers, equal_var=False)

    offers_minus_reward = np.array(train_offers.transaction_aos_minus_reward)
    no_offers_minus_reward = np.array(train_no_offers.transaction_aos_minus_reward)
    t_stat_2, p_val_2 = stats.ttest_ind(offers_minus_reward, no_offers_minus_reward, equal_var=False)

    df_statistics = pd.DataFrame({'2-sample t-test': ['transaction_aos', 'transaction_aos_minus_reward'],
                                  't-statistics': [t_stat_1, t_stat_2],
                                  'p-value': [p_val_1, p_val_2]
                                 }).set_index('2-sample t-test')
    return df_statistics

df_stats = t_test_stats()
df_stats
```

Out[192]:

2-sample t-test	t-statistics	p-value
transaction_aos	39.664380	0.000000e+00
transaction_aos_minus_reward	36.305992	1.003924e-271

transaction_aos:

- Null Hypothesis: transaction_aos_offers = transaction_aos_no_offers
- Alternative Hypothesis: transaction_aos_offers != transaction_aos_no_offers
- Conclusion: p-value is near zero and thus we reject the null. This directionally suggests that customers who engage with offers in a meaningful way generate larger transaction average order size.

transaction_aos_minus_reward:

- Null Hypothesis: transaction_aos_minus_reward_offers = transaction_aos_minus_reward_no_offers
- Alternative Hypothesis: transaction_aos_minus_reward_offers != transaction_aos_minus_reward_no_offers
- Conclusion: p-value is near zero and thus we reject the null. This directionally suggests that customers who engage with offers in a meaningful way generate larger transaction average order size minus the reward.

Zero Rule Algorithm: Classification & Regression

In [212]:
```
def get_train_test_data():
    train_file = 'train_starbucks.csv.gz'
    test_file = 'train_starbucks.csv.gz'
    train_file_gzip = os.path.join(os.getcwd(), 'data', train_file)
    train = pd.read_csv(train_file_gzip, compression='gzip')
    test_file_gzip = os.path.join(os.getcwd(), 'data', test_file)
    test = pd.read_csv(test_file_gzip, compression='gzip')
    return train, test

def identify_engaged_customers(df):
    is_engaged = []
    customer_list = df.customer_id.to_list()
    for id_ in customer_list:
        row = df[df['customer_id']==id_]
        offer_view_rate = row.offer_view_rate.values[0]
        offer_completion_rate = row.offer_completion_rate.values[0]
        if (offer_view_rate >= 0.5) & (offer_completion_rate >= 0.5):
            is_engaged.append(id_)
    return is_engaged

# zero rule algorithm for classification
def zeror_algo_clf(train, test):
    outputs = train.is_engaged.to_list()
    prediction = max(set(outputs), key=outputs.count)
    predicted = [prediction for _ in range(len(test))]
    return predicted

# zero rule algorithm for regression
def zeror_algo_regression(train, test):
    outputs = train.transaction_aos.to_list()
    prediction = sum(outputs) / float(len(outputs))
    predicted = [prediction for _ in range(len(test))]
    return predicted
```

In [205]:
```
%%time
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error

def naive_baseline_clf_model():
    train, test = get_train_test_data()
    train_is_engaged = identify_engaged_customers(train)
    train['is_engaged'] = train['customer_id'].apply(lambda x: 1 if x in train_is_engaged else 0)
    test_is_engaged = identify_engaged_customers(test)
    test['is_engaged'] = test['customer_id'].apply(lambda x: 1 if x in test_is_engaged else 0)

    predicted_clf = zeror_algo_clf(train, test)
    actual_clf = test.is_engaged.to_list()

    tn, fp, fn, tp = confusion_matrix(actual_clf, predicted_clf).ravel()
    accuracy_score = round(1.0 * (tp + tn) / (tn + fp + fn + tp), 4)

    clf_baseline = pd.DataFrame({'tn': [tn], 'fp': [fp],
                                 'fn': [fn], 'tp': [tp],
                                 '(tp + tn)': [tp+tn],
                                 '(tn + fp + fn + tp)': [(tn + fp + fn + tp)],
                                 'accuracy_score': [accuracy_score]})
    return clf_baseline

df_clf_baseline = naive_baseline_clf_model()
df_clf_baseline
```

```
CPU times: user 29.5 s, sys: 228 ms, total: 29.7 s
Wall time: 29.6 s
```

Out[205]:

	tn	fp	fn	tp	(tp + tn)	(tn + fp + fn + tp)	accuracy_score
0	0	5045	0	6345	6345	11390	0.5571

Based on the baseline accuracy score, it tells us that a customer converting with offers is a fair coin toss. Therefore, we want to make an improvement with relevant features and aim to maximize this score as well as other metrics.

In [215]:
```
from sklearn.metrics import mean_squared_error

def naive_baseline_regression_model():
    train, test = get_train_test_data()
    predicted_transaction_aos = zeror_algo_regression(train, test)
    actual_transaction_aos = test.transaction_aos.to_list()
    mse = mean_squared_error(actual_transaction_aos, predicted_transaction_aos)
    return mse

baseline_mse_val = naive_baseline_regression_model()
df_baseline_mse = pd.DataFrame({'Baseline Mean-Squared-Error': [baseline_mse_val]})
df_baseline_mse
```

Out[215]:

	Baseline Mean-Squared-Error
0	348.484547

The baseline MSE value is quite high and the regression model is extremely underfitting. Additional tuning and minimizing the loss function (MSE) using gradient descent, for example, is needed to reduce the error to create an effective generalized linear model. The baseline MSE gives a solid starting point to start making improvements to the model.

Evaluation Metrics

Classification Eval Metric

- The primary metric that I'll be using to evaluate the classification model is ROC-AUC. The goal is to get ROC-AUC as close to "1" as possible. In other words, creating the optimal ROC curve where the "area under curve" is closest to "1".
- I'll also calculate other evaluation metrics, such as accuracy, precision, recall, and F1-score. However, ROC-AUC might be the overall leader among these metrics. Accuracy is easily influenced by imbalanced dataset. And precision and recall can be manipulated by arbitrarily moving the decision threshold.
- F1-score would be a solid secondary metric if the aim is to maximize both precision and recall while balancing the trade-off between the two metrics.

Regresssion Eval Metric

- For the regression model, the primary metric to evaluate its performance would mean squared error (MSE). This is the cost or loss function, where the goal is to minimize this value through gradient descent or use the normal equation.
- Because the goal is predicting a continuous dependent variable, it will be important to also check that the error terms are normally distributed. I'll examine histograms and normal probability plots to confirm.

Maximum Rank Contribution Estimation Eval Metric

- The concept of maximum rank contribution estimation (MRCE) is my own creation to provide a standardized approach to ranking relevant offers for each cohort.
- Within the MRCE framework, offer difficulty, offer duration, viewed rate, and completion rate are attached to each offer in each cohort.
- Using these values, I'll create a metric called "rank contribution score (RCS)". There will a list of RCS values for each cohort, and the maximum RCS value is selected to represent the most engaged offer for the cohort. I'm calling this process the "maximum rank contribution estimation (MRCE)".
- Rank Contribution Score calculation:
  - Normalized difficulty score (NDS): difficulty / duration
  - Engagement score (ES): viewed rate * completion rate
  - Rank contribution score (RCS): ES * (1 - NDS) where (1 - NDS) equals to what I'm calling "penalty".
    - penalty = (1 - NDS)
    - RCS = ES * penalty
  - Maximum rank contribution estimation: MAX[RCS1, RCS2, RCS3, ... , RCSi]

Project Design