Starbucks Capstone Project

I. Definition

Project Overview

Note: In this project, an offer is represented by four attributes concatenated together, as shown below. As a result, a unique offer is represented by type, reward, difficulty, and duration. In the train and test dataset, this offer representation is found under offer_type_v2 column. Throughout the project, when it refers to offer or offer_type, it is referring to this representation.

Offer/Offer Type: type-reward-difficulty-duration
bogo-5-5-168
bogo-10-10-168
bogo-5-5-120
bogo-10-10-120
discount-5-20-240
discount-2-10-240
discount-2-10-168
discount-3-7-168

The goal of the project is to predict which offer has the highest probability of completion for each customer. There are eight offer types, four BOGO and four discount offers, excluding informational ones. In order to predict at this granular level, eight separate classification models are developed to produce eight predictions for each customer based on aggregated lifetime features.

For a batch of targeted customers, eight predictions are generated for every customer. Among the positive predictions, the offer with the highest prediction probability is selected as the winner. If there is a tie, the first winning offer in the array is selected. If there are only negative predictions, the customer will receive a randomly selected offer from top four performing offers.

Problem Statement

Goal:

- Starbucks is using offers to **drive higher customer engagement and spends**. The data shows that average transaction size with completed offers is higher than transactions without completed offers. The goal is to **maximize the potential of offers** by creating predictive models to target customers with offers having the highest probability of success. More specifically, eight supervised classification models are developed each representing a specific offer type. Two informational offers are excluded because there are nothing to redeem or complete. Informational offers with social channel has a high view rate (approximately 90%) and thus this can be used to send messages to customers separately from BOGO and discount offers.

Opportunity:

- The opportunity can expressed by a single data point - only 22% of all transactions have completed offers. Furthermore, 77% of customers have at least one completed offer and most customers have received offers. Therefore, the opportunity is to increase 22% to something much higher so that more transactions have completed offers leading to higher spends and higher engagement.
- 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 and identifying offers with higher probability of conversion per customer, we can achieve a higher ROI. Moreover, this will result in longer customer retention and higher customer lifetime value.

Approach:

- First, to make analyzing and aggregating data faster and less repetitive, two data marts are created.
  1. Offer funnel view: This view flattens the transcript data using the offer received events as the base. In terms of SQL, the offer received events would be the left table and all other events, such as viewed and completed offer events, would be joined to the left table where the matches are made using join keys.
     - `datamart_offer_funnel_view.py`
  2. Transaction engagement view: This dataset is the most important due to the fact that the transaction events are used as the base. All offer events are attributed and mapped back to the transactions where completed offers are attached. This allows fast way to understand which offers received are being viewed and completed.
     - `datamart_transaction_engagement.py`
- Next, the customer_id's are used to split the data into train (67%) and test (33%) sets. Using the train set, two feature engineering/transformation layers are created. The first layer creates aggregated/lifetime metrics at the customer level. The second layer further transforms the features so that they can be used as input features for random forest classification models. All features are listed below.
  - `preprocessor_feat_engine_layer_1.py`
  - `preprocessor_feat_engine_layer_2.py`

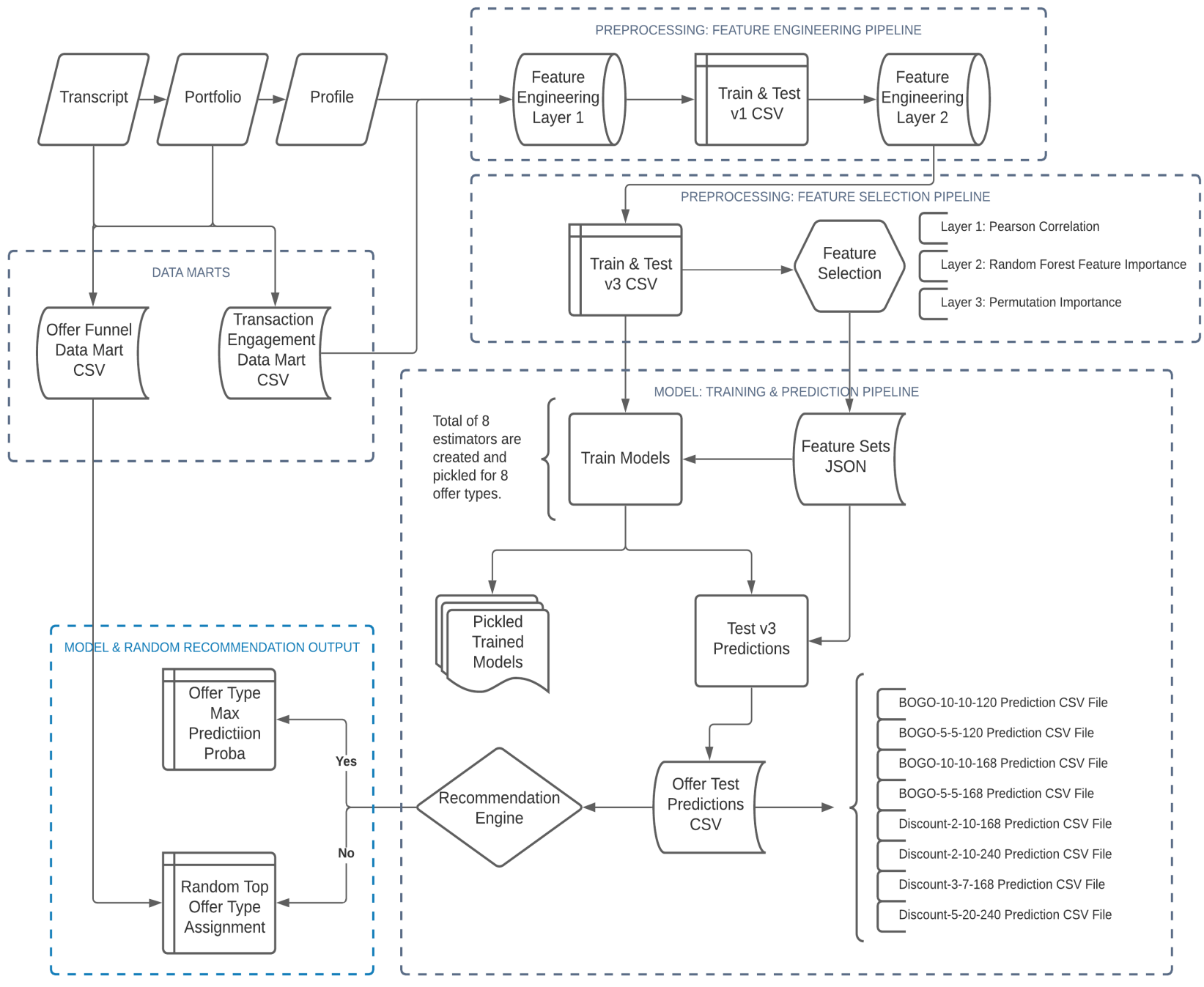
Feature Name	Feature Type	Data Type	Description
customer_id	uuid	string	Unique identifier per customer
gender	binary	string	Describes the sex of the customer
age	continuous	integer	Numeric age of the customer
age_quantile_range	interval	string	5 age quantile ranges (e.g., [$0-$20k])
age_quantile_label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

- At this stage, the train set is split by offer_type (v2), which creates eight train datasets. Before the final models are created, Pearson correlation, random forest feature importance, and permutation importance techniques are used to select most important features to train the final models. At the end of this process, a list of important feature names are saved for each offer type and saved as JSON file.
  - `preprocessor_feat_select_pipeline.py`
  - `trained_models/training_feature_sets.json`

```
{
"bogo-5-5-168": [
"num_bogo_offer_completed",
"total_reward_amount",
"offer_completion_rate",
"offer_received_time",
"avg_reward_per_oc_transaction",
"median_offer_duration",
"num_transactions_oc_direct",
"num_offer_completed",
"percent_oc_direct_transactions",
"transaction_aos_oc",
"num_oc_ch_social",
"avg_offer_completion_time",
"transaction_aos_no_oc",
"transaction_no_oc_amount",
"num_offer_completed_not_viewed",
"transaction_amount",
"num_bogo_offer_received",
"transaction_cnt"
], ...
}
```

- Using the train v3 dataset coupled with important features identified during feature selection, eight random forest classification models are created and pickled (serialized) so that it can be used later to make predictions. The test v3 dataset is used to make predictions for each offer type, and the results are saved.
  - `model_train_predict_pipeline.py`
- Once the test prediction files are saved with prediction probabilities, the recommendation logic will determine which offer will be the winner for each customer. For each customer, eight predictions are generated. Among all predictions, only the positive predictions are isolated. Next, the one with the highest prediction probability is chosen as the winner. If the customer has only negative predictions, a random offer will be chosen among top four performing offers.
  - `model_make_recommedation_pipeline.py`

Full Pipeline Design:



Metrics

Background:

In building a binary classification model, there is a cost associated with certain misclassifications. For example, if you are making a diagnosis if a patient has a disease or not, then you want to minimize false negative results. This means that the patient has the disease, but the diagnosis comes back negative. In this case, it's better to receive a false positive result than a false negative one due to the impact on human life. Therefore, the cost of false negatives are much higher. As a result, building a cost-sensitive model and finding the best tradeoff in making predictions become essential.

The same line of thinking is applicable to Starbucks challenge, although not as serious as disease diagnosis. The goal is to make customer use more offers. It will be helpful to think through the cost associated with misclassifications, false positives and false negatives, which in turn helps put more weight on certain model diagnostic metrics.

Therefore, we want to avoid false postives and false negatives. However, because we are aiming to convert as many customers as possible, false negatives would be relatively better than false positives. Model said a customer would not convert, but converted. This would be overall bad for model performance and even worse if these come up more frequently, but from a marketing perspective it's another offer completion and thus a win. For model performance, what we want here is minimize both false positives and false negatives, and maximize true positives.

Model Diagnostic Metrics

- **Precision:** This will be one of the leading metrics used to measure the performance of random forest classifiers. Precision is important because it measures how well the model successfully predicts true positives among all predicted positives [TP/(TP + FP)]. For this project, we want to maximize precision. In doing so, this helps minimize false positives.
- **Recall:** Recall is another leading metric we want to maximize. Recall is important because it tells us how many actual positives (true positives and false negatives) were predicted correctly in train and test. In the background section, I talked about minimizing false negatives and thus maximizing recall. This is where recall comes into the performance picture [TP/(TP + FN)]. Recall is also known as true positve rate and sensitivity.
- **F1 Score:** This will be the **main metric** used to measure model performance success. This metric is a harmonic mean calculated using precision and recall [(2 * precision * recall)/(precision + recall)]. Therefore, maximizing the F1 score maximizes precision and recall as well, and essentially finds the best balance between the two metrics. This is known as the precision-recall tradeoff and this can be represented visually by the precision-recall curve, which I'll produce later on. As precision increases, recall decreases, and vice versa.
    - **Decision Threshold:** We can further try to maximize the F1 score by tuning the decision threshold, which often is set at 0.5. By moving the decision threshold up or down, we can further improve the F1 score. This method can optimize either precision or recall, but in this case we are optimizing for both using the F1 score. I'll share my analysis later on.
- **ROC AUC:** This is another leading metric and will provide valuable insights into model performance. AUC is the area under the ROC curve and the value usually has a range between 0.5 and 1.0. The ROC curve plots false positive rate on the x-axis and the true positive rate (recall) on the y-axis. This can help locate the best operating point on the curve. For this project, we want to drive the AUC as closest to 1.0 as possible. This means that we are maximizing true positive rate while minimizing false positive rate [FP/(FP + TN)]. False positive rate also can be calculated by (1 - specificity), where specificity is true negative rate [TN/(TN + FP)] and captures the proportion of true negatives among all actual negatives.
- **Accuracy:** Accuracy will be used as a secondary metric and a metric to gauge performance at a high level. This measures how well model performs overall. Accuracy collects all correct positves and negatives and divide by total predictions [(TP + TN)/(TP + TN + FP + FN)].

Marketing Campaign Success Metrics

- **Offer Completion Rate, Offer View Rate, and Average Spends per Customer:** From a machine learning perspective, the models might not be performing well; however, if the completion rate is higher than before, this could potential be a win for marketing. The models are getting some of the predictions right and completion rate might tick up. Same logic applies to view rate and average spends per customer. If the opposite is true, then we have a big problem and back to the drawing boards.
- **What about the customers who received negative predictions?** From a marketing standpoint, you want to maximize success and even if some customers are predicted to reject offers you want to target them with something in order to drive incremental gain. One strategy could be to let the model predict away and see if the predicted negatives are truly negatives. Over time, the team could create a heuristic approach to target these customers and gradually convert them if possible. For this project, I am randomly selecting one of the four top performing offers and targeting customers who received negative predictions.

II. Analysis

Data Exploration

- **Data Marts:** I extensively explored the raw data, which includes transcript.json, profile.json, and portfolio.json files. In order to expedite my exploratory analysis, I created two views, an offer funnel view (offer data mart) and a transaction engagement view (transaction data mart). I'm calling these data marts because they allow quick aggregations of offers and transactions. I also introduce new identifiers to properly, for example, define transaction groupings (row_id's) when multiple offers were attached to a single transaction. There are total of four identifiers, row_id, transaction_id, customer_id, and offer_id. For the transaction data mart, I attributed all received offers, viewed offers, and completed offers to the appropriate transaction_id. I built a custom attribution logic to attach viewed offers and received offers. The attribution logic to attach the correct received offers work about 98% of the time. I noticed a slight issue, but for this exercise this will be good enough.
    - `datamart_offers_funnel_view.py`
    - `datamart_transaction_engagement.py`
- **Feature Transformation & Engineering:** I introduce two layers of feature transformation and engineering. I also created a class to quickly output base dataframes. Using the three provided raw JSON files, rows with missing values and outliers were replaced with corresponding median values. The entire list of features can be found under "Problem Statement - Approach". For more details, refer to these files.
    - `base_transforms/base_transforms_df.py`
    - `preprocessor_feat_engine_layer_1.py`
    - `preprocessor_feat_engine_layer_2.py`

- **Insights:**
  - Number of customers who have completed offers is relatively high, where 77% of all transacted customers have used offers.
  - 97.5% of customers in the profile data have at least one transaction regardless of completing an offer or not. Overall, conversion isn't the problem. The challenge is using offers to amplify every aspect of the business. Does it increase spend? Does it increase retention?
  - Although 77% of customers have at least one offer completed, the percentage of transactions with offer completed is only 22%.
  - Average transaction size with offer completed is overall higher than transactions with no offer completed.

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

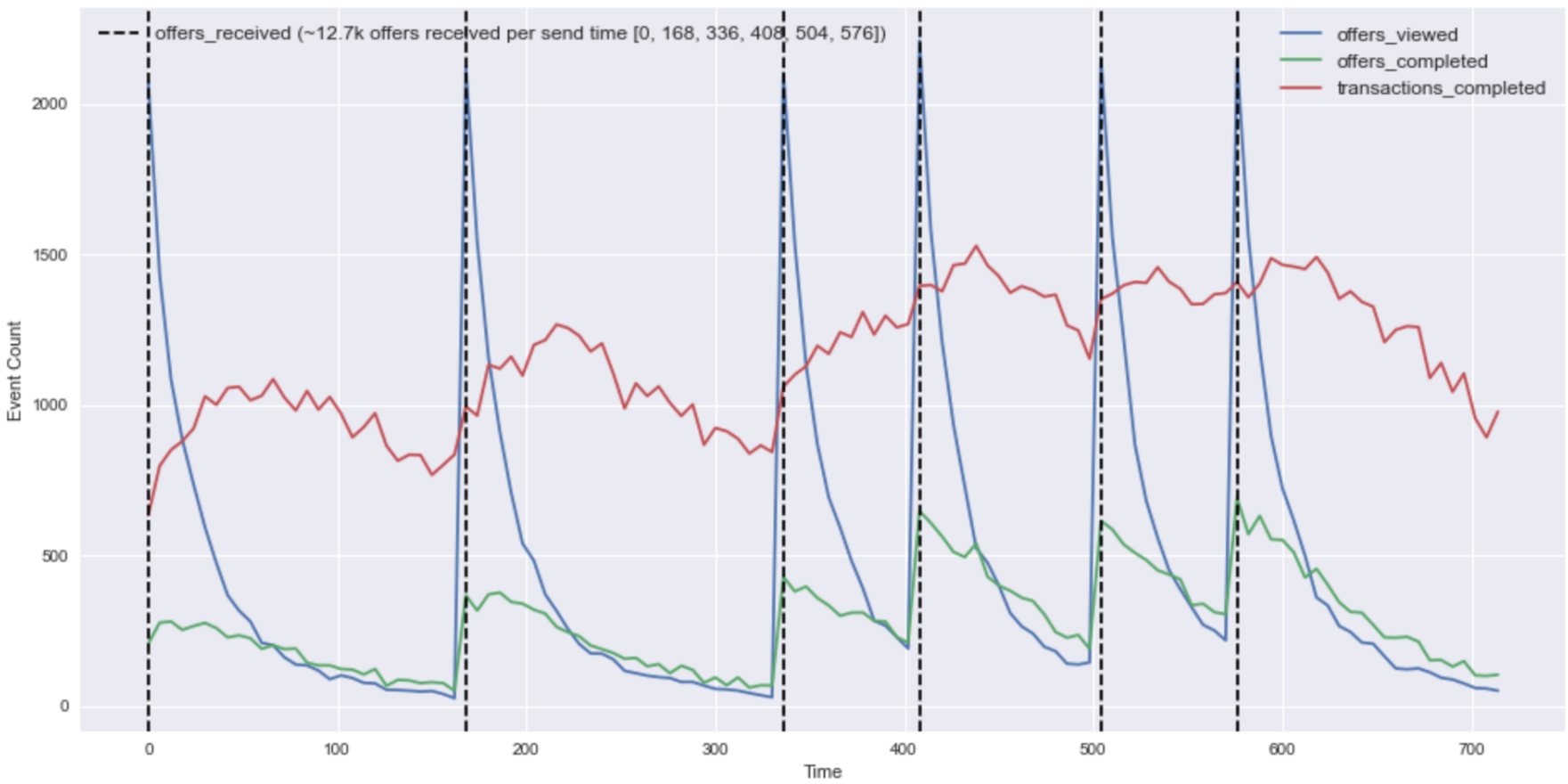
2-sample t-test	t-statistics	p-value
transaction_aos	39.664380	0.000000e+00
transaction_aos_minus_reward	36.305992	1.003924e-271

  - When closely examining the offer funnel, it becomes clear that all offers with social channel perform the best. Thus, leveraging social media becomes an important component in increasing view rate and completion rate.

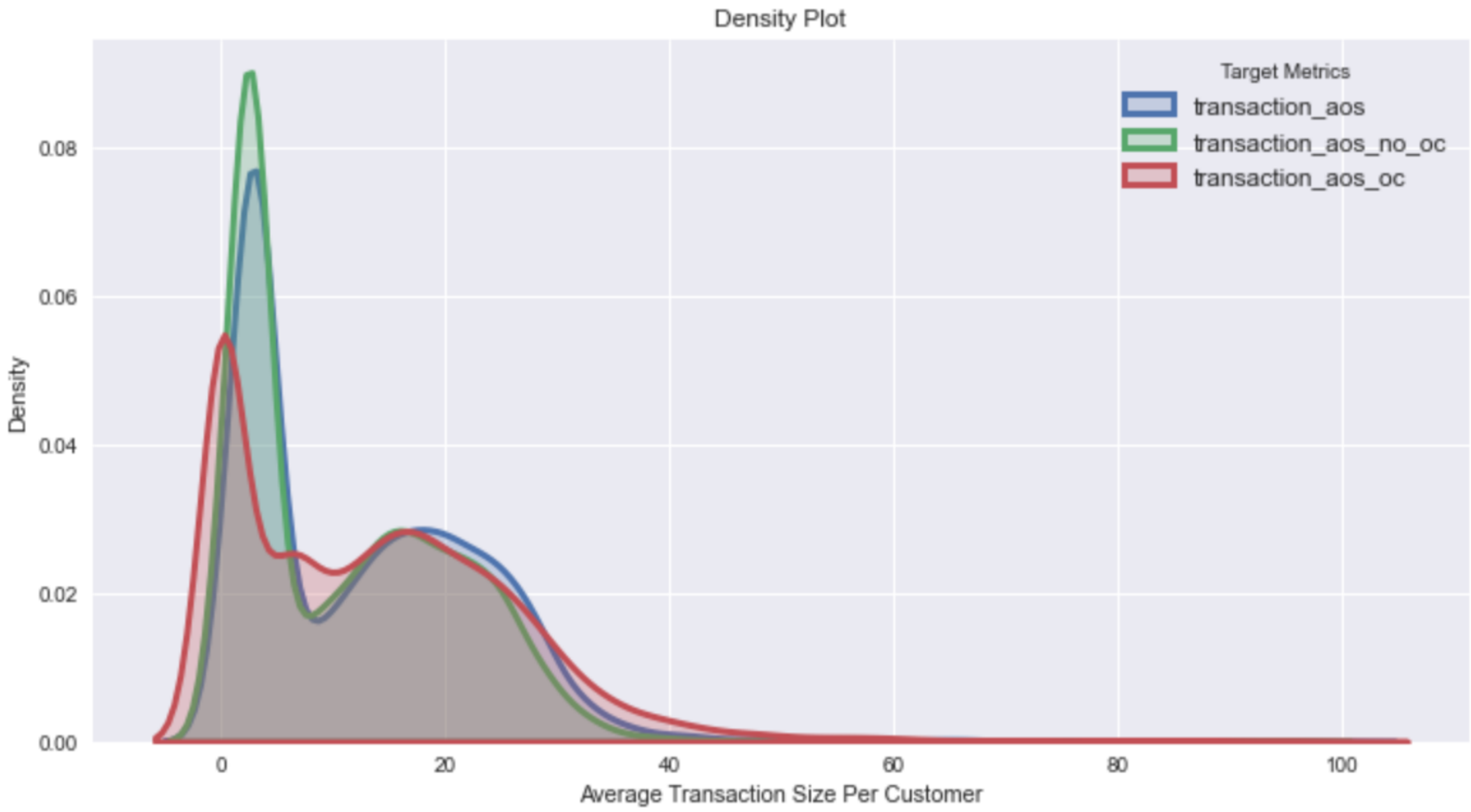
reward	channels	difficulty	duration	offer_type	offer_id	viewed_rate	completion_rate	engagement_score (view_rate * completion_rate)	avg_spend_per_customer	avg_transactions_per_customer
10	[web, email, mobile, social]	10	120	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0	96.11%	43.87%	0.4216	$27.48	1.14
5	[web, email, mobile, social]	5	120	bogo	f19421c1d4aa40978ebb69ca19b0e20d	95.95%	56.74%	0.5444	$22.34	1.14
10	[email, mobile, social]	10	168	bogo	ae264e3637204a6fb9bb56bc8210ddfd	87.70%	48.16%	0.4223	$27.73	1.14
5	[web, email, mobile]	5	168	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9	54.33%	56.71%	0.3081	$20.59	1.14
2	[web, email, mobile, social]	10	240	discount	fafdcd668e3743c1bb461111dcafc2a4	96.45%	69.99%	0.6750	$21.61	1.16
3	[web, email, mobile, social]	7	168	discount	2298d6c36e964ae4a3e7e9706d1fb8c2	95.96%	67.43%	0.6471	$20.48	1.16
2	[web, email, mobile]	10	168	discount	2906b810c7d4411798c6938adc9daaa5	53.96%	52.63%	0.2840	$23.76	1.14
5	[web, email]	20	240	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7	34.73%	44.60%	0.1549	$29.52	1.12
0	[email, mobile, social]	0	72	informational	5a8bc65990b245e5a138643cd4eb9837	87.78%	0.00%	0.8778	$0.00	0.00
0	[web, email, mobile]	0	96	informational	3f207df678b143eea3cee63160fa8bed	54.40%	0.00%	0.5440	$0.00	0.00

Exploratory Visualization

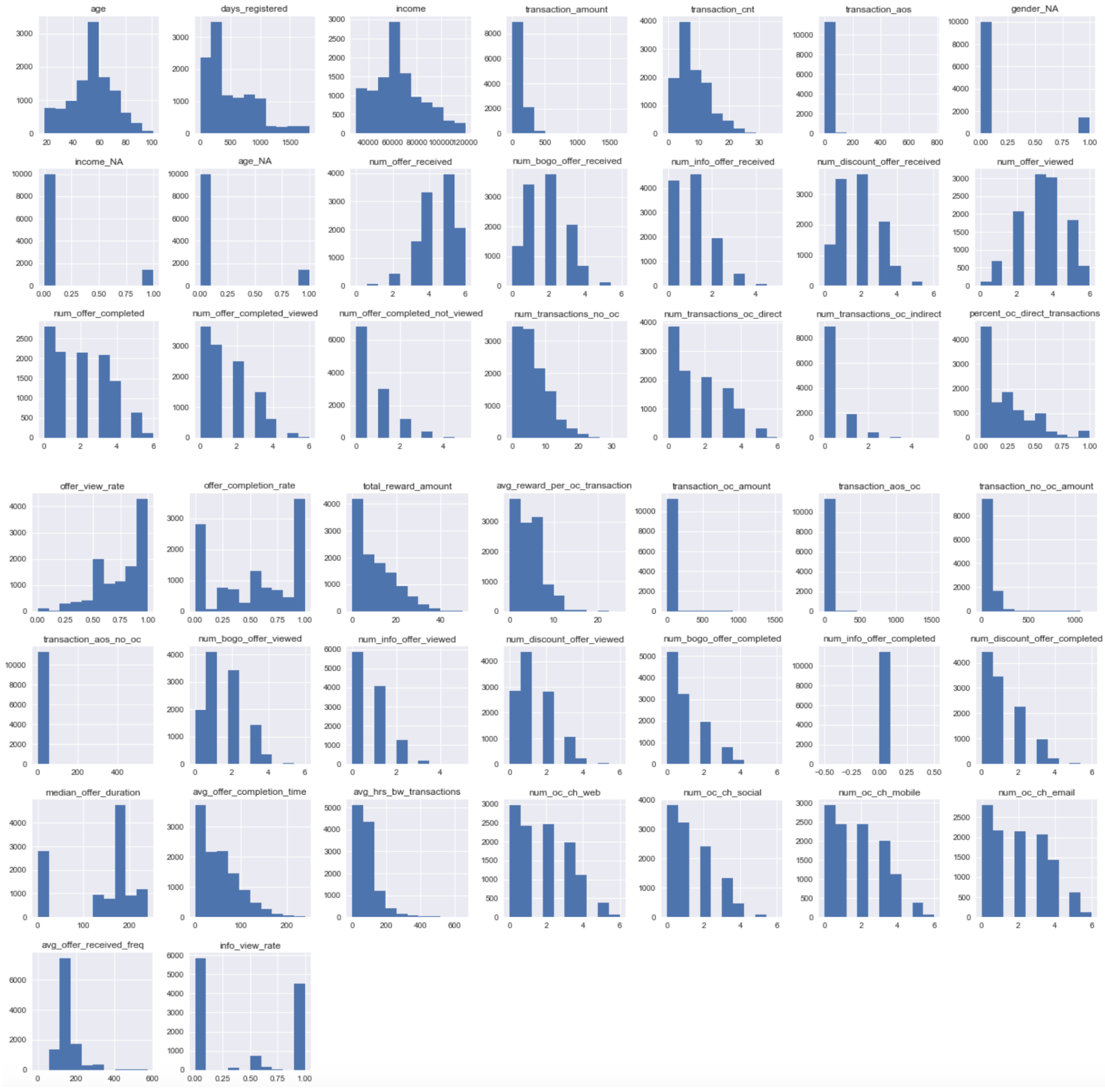
- **Time series chart** can be powerful in showing big trends over time. This chart shows offer received, viewed, and completed events as well as number of transaction events over time (in hours). We can clearly see the peaks and troughs. For each offer send date, we see an immediate jump in all events. We also see a gradual increase in transactions, and a loose correlation between offer sends and transactions can be observed. This could be a weekly spend cycle, but we also observe increase in transaction events in some offer send dates.



- **Density plots** of average transaction size with or without offer completed can show us if offers are making a difference in driving higher spends and helping topline (gross revenue). We do see this happening in the plots.
  - transaction_aos: overall transaction average order size
  - transaction_aos_no_oc: transaction average order size without offer completed
  - transaction_aos_oc: transaction average order size with offer completed



- **Feature histograms** help quickly understand variable distribution in training data and become an input in providing insights into how we would go about transforming the features as well as engineering new ones.



Algorithms and Techniques

- **Algorithm:** Random forest classifier was chosen to predict offer conversion. I decided on this classifier due to its flexibility and robustness. It is an ensemble model and generally produces solid predictions. It uses impurity-based approach to feature importance. As a result, random forest classifier can be used for both feature selection and building a binary classification model.
  - **Hyperparameter tuning:** Random forest classifier also has its set of hyperparameters, usually related to leaf and node splits. These can be tuned using randomized search or grid search to further improve the model. For the model tuning step, I experimented with randomized search to tune hyperparameters. The tuning process did not significantly improve the model. Therefore, tuning the hyperparameters won't be necessary for ths project.
  - **Decision threshold tuning:** Another opportunity to further tune the model is determining the best decision treshold, which is usually set at 0.5. Based on which misclassifications to minimize, the decision threshold can be manipulated to locate the new optimal point and potentially improve the model. I went ahead and conducted the decision threshold analysis, which will be shared later. The model threholds were already pretty close to the optimal threholds and decided not to make changes.
- **Feature Selection:** I applied what felt right for this process. First, I wanted to remove correlated features. Second, I wanted to use both random forest feature importance and permutation importance to determine important features for each offer type. I created an arbitrary threshold for each importance method, and overlapping features made up the final feature set. After correlated features were removed, I trained eight random forest models using eight different training sets (each representing offer type) and extracted the important features. I ran each training set through permutation importance function as well. Next, I identified overlapping features. These feature sets were serialized as JSON to be used later when creating model estimators.
- **Model Estimators:** Leveraging the serialized JSON feature sets and training v3 data, I created eight estimators and pickled each to be used later for making test predictions. There is not much class imbalance in the datasets so hyperparamter and decision threshold tuning steps were not absolutely necessary. However, I went through the exercise of diving deeper into randomized search to tune hyperparameters and determining the optimal threshold for each estimator, as mentioned above. The results did not incrementally improve the models in a significant way.

Benchmark

- **Zero Rule Algorithm Classification:** ROC_AUC and accuracy provide a consistent picture of no-skill model per offer type. The zero rule algo picks the majority class and generates the confusion matrix, which in turn creates the diagnostic metrics. In some cases, the positive is the majority class while in some cases the negative is the majority class, which is reflected in the baseline precision and recall values.

In [74]:

```
import pandas as pd
from sklearn.metrics import confusion_matrix, roc_auc_score
import warnings
warnings.filterwarnings('ignore')

# zero rule algorithm for classification
def zeror_algo_clf(offer_type, train, test):
    outputs = train[train.offer_type_v2==offer_type].offer_completed.tolist()
    prediction = max(set(outputs), key=outputs.count)
    n_test = test[test.offer_type_v2==offer_type].shape[0]
    predicted = [prediction for _ in range(n_test)]
    return predicted

def naive_baseline_model(train, test):
    grouped_metrics = {}
    offer_types = train.offer_type_v2.unique()
    for offer_type in offer_types:
        offer_type_metrics = []
        y_baseline_pred = zeror_algo_clf(offer_type, train, test)
        y_actual_test = test[test.offer_type_v2==offer_type].offer_completed.to_list()
        tn, fp, fn, tp = confusion_matrix(y_actual_test, y_baseline_pred).ravel()
        precision = round((1.0 * tp)/(tp + fp), 4)
        recall = round((1.0 * tp)/(tp + fn), 4)
        f1_score = round((2 * precision * recall)/(precision + recall), 4)
        roc_auc = roc_auc_score(y_actual_test, y_baseline_pred)
        accuracy_score = round(1.0 * (tp + tn) / (tn + fp + fn + tp), 4)
        offer_type_metrics.append(precision)
        offer_type_metrics.append(recall)
        offer_type_metrics.append(f1_score)
        offer_type_metrics.append(roc_auc)
        offer_type_metrics.append(accuracy_score)
        grouped_metrics[offer_type] = offer_type_metrics
    column_names = ['baseline_precision', 'baseline_recall', 'baseline_f1_score',
                    'baseline_roc_auc', 'baseline_accuracy']
    df = pd.DataFrame.from_dict(grouped_metrics, orient = 'index',
                                columns = column_names).reset_index().rename(columns={'index': 'offer_type'})
    return df
```

In [75]:
```
# execute naive_baseline_model function
X_train_v3 = pd.read_csv('data/train_v3_starbucks.csv.gz', compression = 'gzip')
X_train_v3 = X_train_v3[X_train_v3.offer_type != 'informational']
X_test_v3 = pd.read_csv('data/test_v3_starbucks.csv.gz', compression = 'gzip')
X_test_v3 = X_test_v3[X_test_v3.offer_type != 'informational']
df_baseline = naive_baseline_model(X_train_v3, X_test_v3).fillna(0)
df_baseline.to_csv('output/diagnostic_metrics/baseline_metrics.csv.gz', index = False, compression = 'gzip'
df_baseline
```

Out[75]:

	offer_type	baseline_precision	baseline_recall	baseline_f1_score	baseline_roc_auc	baseline_accuracy
0	bogo-5-5-168	0.5518	1.0	0.7112	0.5	0.5518
1	bogo-10-10-168	0.0000	0.0	0.0000	0.5	0.5058
2	bogo-5-5-120	0.5642	1.0	0.7214	0.5	0.5642
3	discount-5-20-240	0.0000	0.0	0.0000	0.5	0.5678
4	discount-2-10-240	0.6839	1.0	0.8123	0.5	0.6839
5	bogo-10-10-120	0.0000	0.0	0.0000	0.5	0.5676
6	discount-2-10-168	0.5014	1.0	0.6679	0.5	0.5014
7	discount-3-7-168	0.6517	1.0	0.7891	0.5	0.6517

III. Methodology

Data Preprocessing

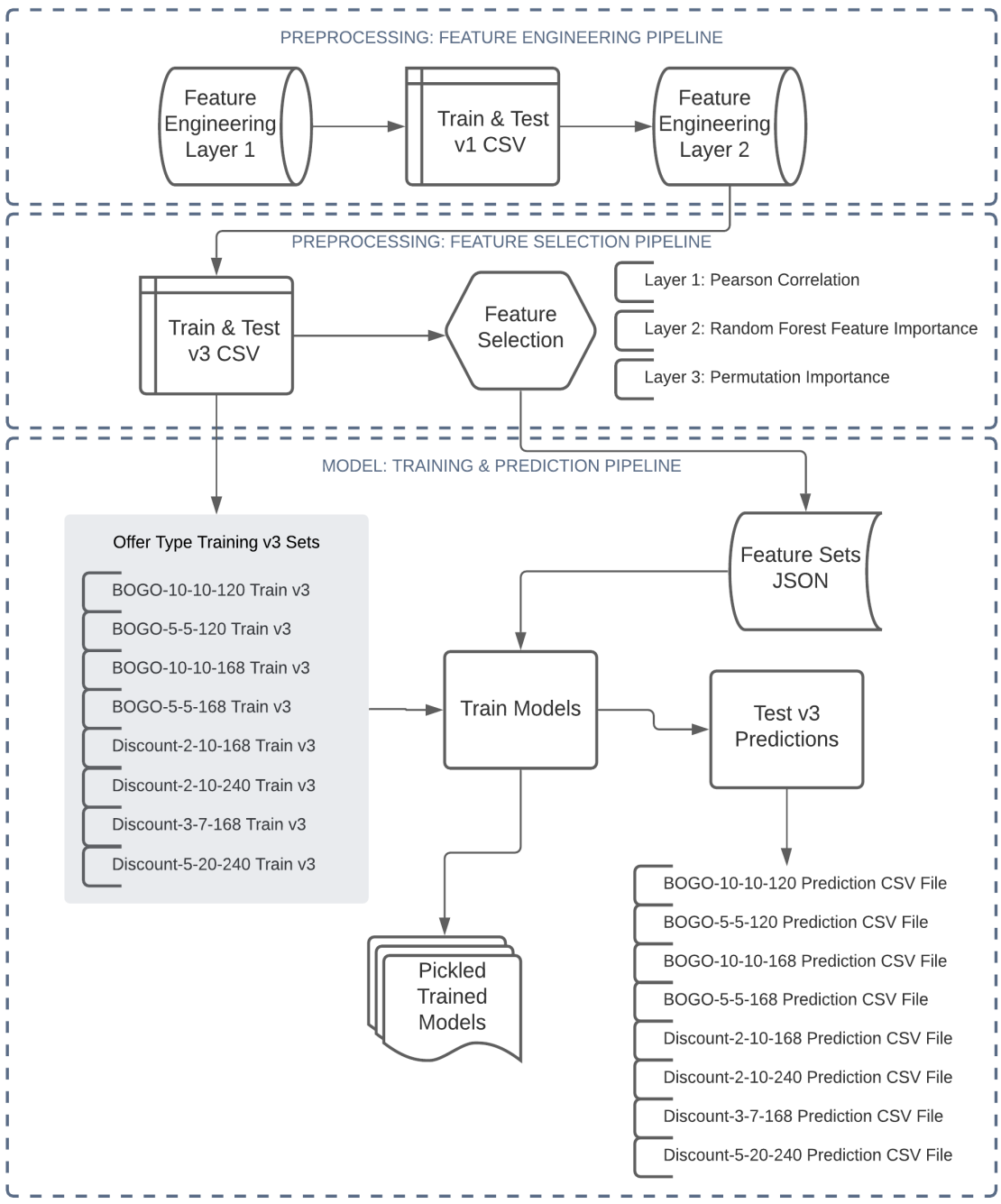
- **Transaction Engagement:** `datamart_transaction_engagement.py`
  - Attribution of all offer events to corresponding transactions.
  - This view allows what offers are being completed with transactions as well as which offers have been receive and viewed at the transaction level.
- **Feature Engineering Layer 1:** `precessor_feat_engine_layer_1.py`
  - **missing_outlier_imputer** - Imputes missing values and outliers with corresponding median values. Rows were not dropped due to small number of occurrences. Age outliers were imputed with median age and any data points that looked like outliers were kept because these data points can occur in the real world.
  - **age_quantile_transformer** - Placing customer's age into one of five quantile bins.
  - **income_quantile_transformer** - Placing customer's income into one of five quantile bins.
  - **date_registered_transformer** - Member date is changed to days (available max date - member's date) and each customer's member date is placed in one of ten quantile bins.
  - **offer_received_transformer** - For each customer, create number of offers received for each offer name (BOGO, discount, and informational).
  - **offer_viewed_transformer** - For each customer, create number of offers viewed for each offer name (BOGO, discount, and informational).
  - **offer_completed_transformer** - For each customer, aggregate total number of offers completed, number of offers completed that were viewed, and number of offers completed that were not viewed.
  - **offer_completed_by_offer_type** - For each customer, aggregate number of offers completed by offer name (BOGO, discount, and informational).
  - **transaction_count_transformer** -
    - Create num_transactions_no_oc (no completed offer attached),
    - num_transactions_oc_direct (number of transactions where offer threshold was met in a single transaction),
    - num_transactions_oc_indirect (number of transactions where offer threshold was met over time), and
    - percent_oc_direct_transactions (percent of transactions where offers were completed in the single transaction).
  - **offer_ratio_calculations** - Calculate offer_view_rate, offer_completion_rate, and info_view_rate.
  - **offer_reward_transformer** - Calculate total_reward_amount and avg_reward_per_oc_transaction (average reward for transaction with offer completed).
  - **transaction_amount_transformer** - For each customer, create transaction_oc_amount (transaction amount with offers completed), transaction_aos_oc (average transaction size with offers completed), transaction_no_oc_amount (transaction amount without offers completed), and transaction_aos_no_oc (average transaction size without any offers completed).
  - **median_offer_duration_calc** - For all the offers received per customer, calculate the median duration of all offers received.
  - **avg_offer_completion_time_calc** - Calculate avg_offer_completion_time for each customer if possible. sum(offer_completed_time - offer_received_time) / count(offer_completed)
  - **avg_hrs_bw_trans** - Function iterates over each customer_id and creates a list of all transaction_time per transaction_id, and calculates the difference between current and previous transaction, and then averages the spreads.
  - **offer_channel_counter** - For each customer, create num_oc_ch_social, num_oc_ch_web, num_oc_ch_mobile, and num_oc_ch_email. Count of offer completed attached to channels.
  - **avg_offer_received_frequency** - Calculates the average offer received frequency. Basically, taking all the offers received per customer, and taking the difference from current to previous until the very first one. Once you have time differences, you take the average to get an understand how often the customers received offers.
- **Feature Engineering Layer 2:** `precessor_feat_engine_layer_2.py`
  - **drop_constant_features** - Identify constant features and drop them if any. `config.py`
  - **drop_generic_features** - Drop manually identified generic features. `config.py`
  - **gender_one_hot_encoder** - Gender converted with one hot encoding.

    - **ordinal_feature_encoder** - 'age_quantile_label', 'days_reg_quantile_label', and 'income_quantile_label' features converted to ordinal features. `config.py`
    - **drop_redundant_features** - Drop manually identified redundant features. `config.py`
    - **create_train_test_v1** - Staging step for train and test set.
    - **create_portfolio_expanded** - Staging step.
    - **create_received_expanded** - Staging step.
    - **create_train_test_v2** - Staging step.
    - **create_train_test_v3** - Final train and test set.
- **Feature Selection:** `precessor_feat_select_pipeline.py`
    - **feature_importance_ranking** - Random forest feature importance ranking.
    - **permutation_importance_ranking** - Scikit-learn permutation importance ranking for features.
    - **feature_selection_per_offer_type** - Full feature selection function with removing correlated features and implementing two ranking functions from above.
    - **serialize_feature_sets** - Serializes the feature selection function output for each offer type for later use.

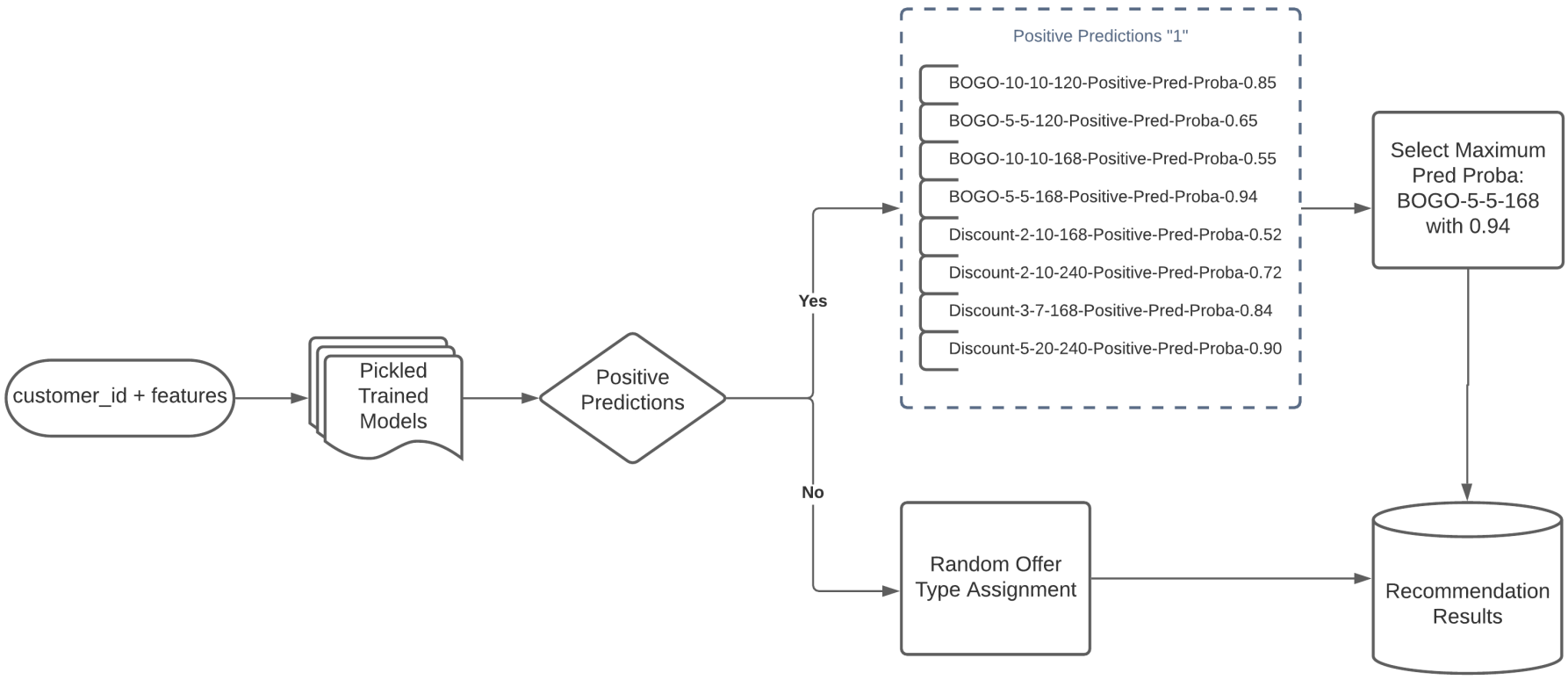
Implementation

- **Overview:** Once all the scripts are created, the implementation is straightforward. In the design flow diagram below, the key step is the "Model: Training & Prediction Pipeline". In this step, the train v3 set is split into eight training datasets for each offer type. Each training set is used to train a random forest model and the serialized feature set JSON is used as input to map important features to each training data. Furthermore, the pickled models are used to make predictions on the testing set. Finally, the recommendation logic assigns a winning offer per customer.
    - `preprocessor_feat_engine_layer_1.py`
        - data/train_starbucks.csv.gz (train v1)
        - data/test_starbucks.csv.gz (test v1)
    - `precessor_feat_engine_layer_2.py`
        - data/train_v3_starbucks.csv.gz
        - data/test_v3_starbucks.csv.gz
    - `precessor_feat_select_pipeline.py`
        - trained_models/training_feature_sets.json
    - `model_train_predict_pipeline.py`
        - trained_models/
            - bogo-5-5-120_model.pickle
            - bogo-5-5-168_model.pickle
            - bogo-10-10-120_model.pickle
            - bogo-10-10-168_model.pickle
            - discount-2-10-168_model.pickle
            - discount-2-10-240_model.pickle
            - discount-3-7-168_model.pickle
            - discount-5-20-240_model.pickle
        - output/predictions/
            - bogo-5-5-120_test_predictions.csv.gz
            - bogo-5-5-168_test_predictions.csv.gz
            - bogo-10-10-120_test_predictions.csv.gz
            - bogo-10-10-168_test_predictions.csv.gz
            - discount-2-10-168_test_predictions.csv.gz
            - discount-2-10-240_test_predictions.csv.gz
            - discount-3-7-168_test_predictions.csv.gz
            - discount-5-20-240_test_predictions.csv.gz
    - `model_make_recommendation_pipeline.py`
        - output/recommendations/
            - model_recommendations.csv.gz
            - random_recommendations.csv.gz
    - `output_model_stats.py`
        - output/diagnostic_metrics/
            - train_model_stats.csv.gz
            - test_model_stats.csv.gz
            - test_optimal_f1_thresholds.csv.gz
            - test_roc_curves.png
            - test_pr_curves.png

- **Train & Predict Deep Dive:** The outcome at this stage is eight pickled random forest estimators and eight test prediction CSV files. Using the pickled models and saved test predictions, I can easily generate diagnostic metrics and apply the recommendation logic to produce offer recommendations.



- **Recommendation Logic Deep Dive:** Using the testing set, we make predictions with the assumption these are the customers we will be targeting in the next offer batch send. Essentially, each customer with corresponding features is plugged into eight models to generate positive or negative predictions. Among eight predictions per customer, the positive ones are isolated. Among the positive predictions, the maximum prediction probability is determined and the corresponding offer type is the winner. If customer has only negative predictions, the customer will receive a randomly chosen offer type from the top four performing offers.



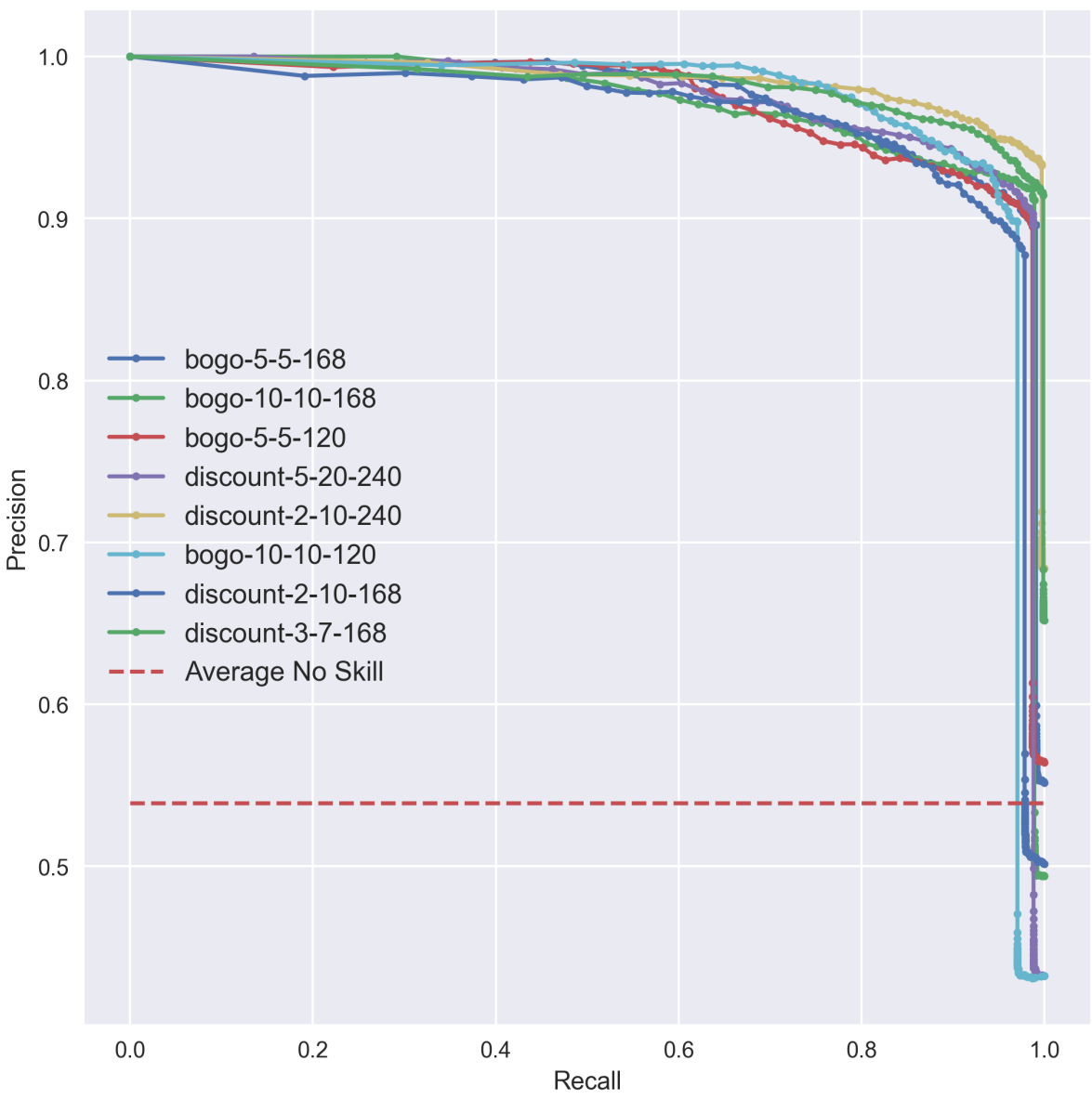
Refinement

- **Randomized Search Hyperparameter Tuning:** For each random forest model, I used randomized search to tune model hyperparameters. Based on how I set up the randomized search, the run time was about 10 minutes per offer type model. I assessed the model improvement and the gains were not huge for the amount of effort. As a result, I made the decision of using the default hyperparameters.

In [174]:
```
%%time
# param_distributions
param_grid = {'criterion': ['gini', 'entropy'],
              'n_estimators': [100, 200, 300, 500, 800, 1000, 1200],
              'max_features': ['auto', 'log2'],
              'max_depth': [3, 5, 7, 9, 15, 25, 30],
              'min_samples_split': [2, 3, 4, 5, 7, 10, 15],
              'min_samples_leaf': [1, 2, 5, 10, 12, 14]}

# create base model to tune
rf = RandomForestClassifier(random_state=0)

# randomized search of parameters using 3 fold cross validation,
search = RandomizedSearchCV(estimator=rf,
                            param_distributions=param_grid,
                            n_iter=200,
                            cv=5,
                            verbose=3,
                            random_state=0,
                            n_jobs=-1)

# fit model
rf_rsearch.fit(X_train_d520240[feature_set_v2], y_train_d520240)
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
CPU times: user 8.01 s, sys: 432 ms, total: 8.44 s
Wall time: 10min 28s
```

- **Decision Threshold Optimization:** `output_model_stats.py`
  - Because one of the main success metrics is the F1 score, I wanted to see if the decision threshold could be further tuned and optimized by improving the F1 score. I took a closer look at precision and recall by creating the precision-recall curves using the testing set.
  - Can we move the decision threshold to make better predictions on the test set? In order to achieve this, what we want to do is maximize the F1 score with the understanding that there is a tradeoff between precision and recall. By locating the maximum F1 score, we are locating the maximum operating point on the precision-recall curve and the decision threshold associated with this point can be mapped. This logic is captured under the `output_model_stats.create_and_save_precision_recall_curve` function.



  - The output of the function shows the best decision threshold for each model. When making predictions with new data or test data, these new decision thresholds can be used to determine positive or negative outcome per customer per model. My decision was not to modify the thresholds for future predictions because the difference between default and modified thresholds were not too different. The gains were small, but any other future improvements could incorporate these changes.

In [56]:
```
1 pd.read_csv('output/diagnostic_metrics/test_optimal_f1_thresholds.csv.gz', compression = 'gzip')
```

Out[56]:

	offer_type	optimal_decision_threshold	optimal_f1_score	optimal_precision	optimal_recall
0	bogo-5-5-168	0.51	0.941058	0.896046	0.990832
1	bogo-10-10-168	0.54	0.951044	0.917949	0.986614
2	bogo-5-5-120	0.54	0.940105	0.901882	0.981712
3	discount-5-20-240	0.57	0.943446	0.906487	0.983547
4	discount-2-10-240	0.57	0.964458	0.937017	0.993556
5	bogo-10-10-120	0.61	0.935721	0.931444	0.940037
6	discount-2-10-168	0.55	0.926541	0.887553	0.969112
7	discount-3-7-168	0.57	0.955477	0.918873	0.995119

IV. Results

Model Evaluation and Validation

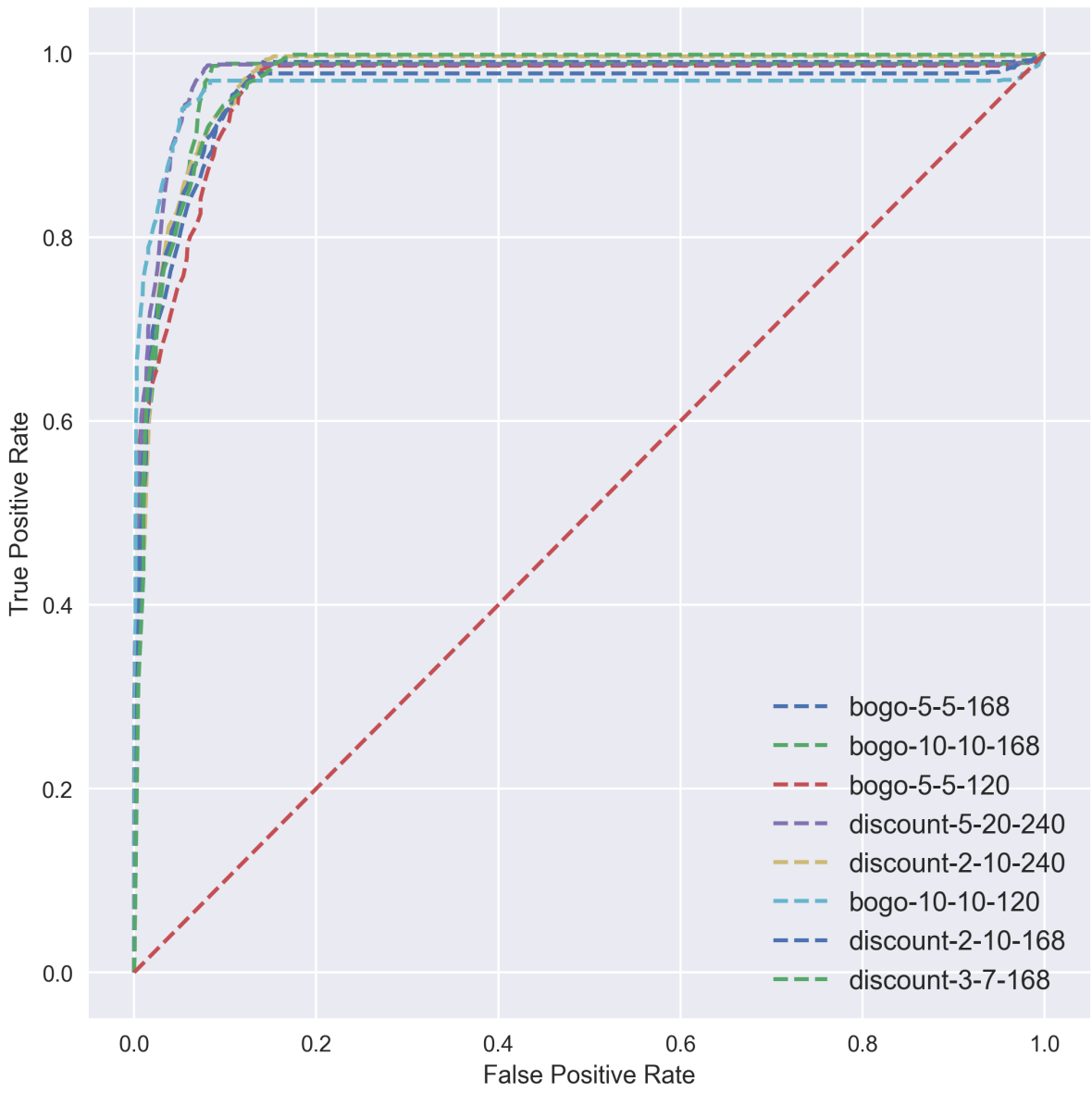
- **Training Model Evaluation:** After eight different random forest models were trained, I used the split training sets, X_train_v3 and y_train_v3, to generate cross-validation (with kfold = 6) diagnostic metrics. This process provides a more realistic picture of model performance before using the test set. Overall, the results look good.
  - Precision and recall metrics are calculated using `sklearn.model_selection.cross_val_predict` function, and the results are looking solid across the board.
  - F1 scores and ROC AUC's are all in the mid to high 90% range.
  - If all customers are able to have the features generated during the feature engineering process, then we should be able to plug these customers' attributes into the models and generate reliable predictions.

In [57]:
```
1 pd.read_csv('output/diagnostic_metrics/train_model_stats.csv.gz', compression = 'gzip')
```

Out[57]:

	offer_type	accuracy	accuracy_cv_score	accuracy_cv_stddev	precision_score	recall_score	f1_score	roc_auc_score (cross_val_score)
0	bogo-5-5-168	1.000000	0.932251	0.009086	0.900600	0.989255	0.942848	0.976598
1	bogo-10-10-168	1.000000	0.950275	0.007625	0.917391	0.981395	0.948315	0.985364
2	bogo-5-5-120	1.000000	0.918026	0.010330	0.882226	0.985838	0.931158	0.968386
3	discount-5-20-240	1.000000	0.940359	0.007936	0.898438	0.972841	0.934160	0.982780
4	bogo-10-10-120	1.000000	0.941017	0.011880	0.902900	0.968877	0.934726	0.986862
5	discount-2-10-168	1.000000	0.929023	0.011651	0.897180	0.977520	0.935628	0.975838
6	discount-3-7-168	1.000000	0.936785	0.008056	0.917950	0.995393	0.955104	0.974016
7	discount-2-10-240	0.999805	0.950156	0.004508	0.936311	0.995784	0.965132	0.977149

- **Test Set Evaluation:** The test set produced solid numbers across the board. In examining the F1 scores and ROC AUC more closely, these numbers were indicating that the model can produce predictions with high precision and high recall. The ROC curves also indicate that the model performance across all eight models were very strong.
  - The current F1 scores hold up very well against the optimized F1 scores from an earlier exercise. In other words, the decision threshold tuning is not necessary for this project.
  - In terms of how sensitive the models are to the input values, it's good to have a general understanding of how this could impact the models going forward. Leveraging the work I did earlier with permutation importance, we can extrapolate a general framework to measure model sensitivity. In a nutshell, permutation feature importance measures the decrease in model performance when a single feature is randomly shuffled. The output is a list of features with their predictive contribution to the model. In another words, the shuffled feature allows the measurement of decrease in model score and thus quantifying the impact on model performance.
  - The current models will start to degrade if features found in trained_models/trainging_feature_sets.json change over time. Therefore, monitoring the feature distribution over time will be helpful, including outlier monitoring if possible.



In [71]:
```
import json
feature_sets = json.load(open('trained_models/training_feature_sets.json'))
feature_sets['bogo-5-5-168']
```

Out[71]:
```
['num_bogo_offer_completed',
 'total_reward_amount',
 'offer_completion_rate',
 'offer_received_time',
 'avg_reward_per_oc_transaction',
 'median_offer_duration',
 'num_transactions_oc_direct',
 'num_offer_completed',
 'percent_oc_direct_transactions',
 'transaction_aos_oc',
 'num_oc_ch_social',
 'avg_offer_completion_time',
 'transaction_aos_no_oc',
 'transaction_no_oc_amount',
 'num_offer_completed_not_viewed',
 'transaction_amount',
 'num_bogo_offer_received',
 'transaction_cnt']
```

Model Diagnostic Metrics for Test Set

In [58]:
```
pd.read_csv('output/diagnostic_metrics/test_model_stats.csv.gz', compression = 'gzip')
```

Out[58]:

	offer_type	accuracy	precision	recall	f1	roc_auc	tn	fp	fn	tp
0	bogo-5-5-168	0.931518	0.896046	0.990832	0.941058	0.975653	989	163	13	1405
1	bogo-10-10-168	0.947082	0.911466	0.988976	0.948640	0.979838	1178	122	14	1256
2	bogo-5-5-120	0.927363	0.895156	0.986832	0.938761	0.971952	898	158	18	1349
3	discount-5-20-240	0.945081	0.895609	0.988117	0.939591	0.985684	1311	126	13	1081
4	discount-2-10-240	0.948718	0.932603	0.997071	0.963760	0.975934	666	123	5	1702
5	bogo-10-10-120	0.939370	0.897611	0.970480	0.932624	0.987849	1303	120	32	1052
6	discount-2-10-168	0.920635	0.877424	0.978378	0.925155	0.975082	1111	177	28	1267
7	discount-3-7-168	0.938370	0.914525	0.998780	0.954797	0.974480	723	153	2	1637

Optimized Decision Thresholds using Precision-Recall Curve

In [59]:
```
pd.read_csv('output/diagnostic_metrics/test_optimal_f1_thresholds.csv.gz', compression = 'gzip')
```

Out[59]:

	offer_type	optimal_decision_threshold	optimal_f1_score	optimal_precision	optimal_recall
0	bogo-5-5-168	0.51	0.941058	0.896046	0.990832
1	bogo-10-10-168	0.54	0.951044	0.917949	0.986614
2	bogo-5-5-120	0.54	0.940105	0.901882	0.981712
3	discount-5-20-240	0.57	0.943446	0.906487	0.983547
4	discount-2-10-240	0.57	0.964458	0.937017	0.993556
5	bogo-10-10-120	0.61	0.935721	0.931444	0.940037
6	discount-2-10-168	0.55	0.926541	0.887553	0.969112
7	discount-3-7-168	0.57	0.955477	0.918873	0.995119

Justification

The test results show that the models significantly improved performance compared to their corresponding benchmark/baseline metrics. Before the important features were identified and the random forest classifiers were trained, the baseline metrics show that it was almost a fair coin toss and close to random. In the context of the ROC curve, the baseline ROC AUC is represented as the straight diagonal line, which represents the no-skill model. In comparison, all the trained models' ROC AUC scores are close to 1.0. The trained models are all predicting at 90%+ across all leading metrics.

In [72]:
```
pd.read_csv('output/diagnostic_metrics/baseline_metrics.csv.gz', compression = 'gzip')
```

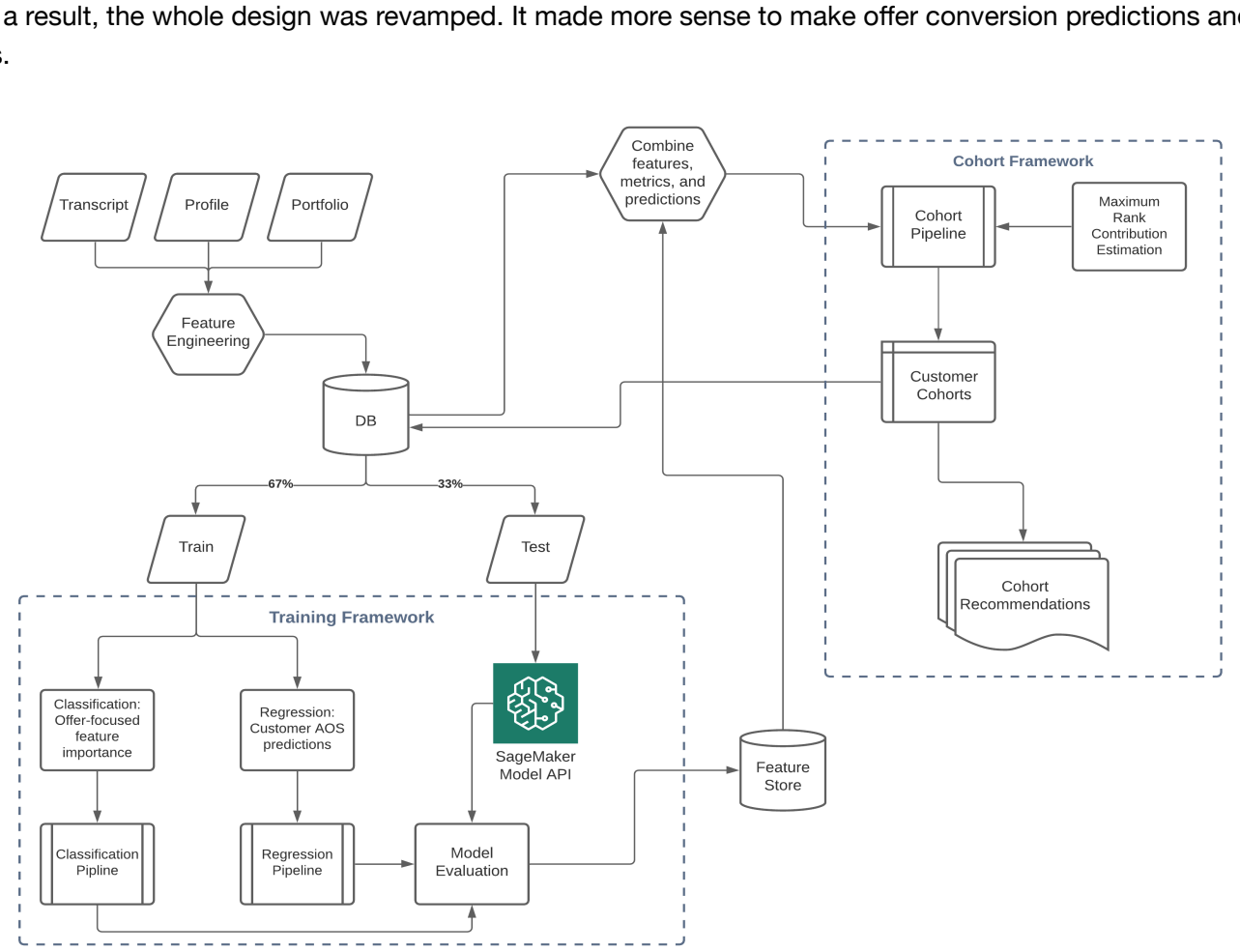
Out[72]:

	offer_type	baseline_precision	baseline_recall	baseline_f1_score	baseline_roc_auc	baseline_accuracy
0	bogo-5-5-168	0.5518	1.0	0.7112	0.5	0.5518
1	bogo-10-10-168	0.0000	0.0	0.0000	0.5	0.5058
2	bogo-5-5-120	0.5642	1.0	0.7214	0.5	0.5642
3	discount-5-20-240	0.0000	0.0	0.0000	0.5	0.5678
4	discount-2-10-240	0.6839	1.0	0.8123	0.5	0.6839
5	bogo-10-10-120	0.0000	0.0	0.0000	0.5	0.5676
6	discount-2-10-168	0.5014	1.0	0.6679	0.5	0.5014
7	discount-3-7-168	0.6517	1.0	0.7891	0.5	0.6517

V. Conclusion

Free-Form Visualization

- This pipeline design, as shown below, was initially what I proposed, but during the build process I soon realized things had to change. The linear regression model that I had initially try to build produced bad results. At the same time, my initial approach of creating cohorts was much harder than expected. As a result, the whole design was revamped. It made more sense to make offer conversion predictions and targeting customers with specific offers.



Reflection

- First, data marts were created to make analysis and aggregations faster. Next, feature engineering layers were created to build, transform, and engineer features. Feature selection layer was built to remove correlated features and retain important ones through random forest feature importance and permutation importance. Random forest classifiers were trained and pickled for each offer type. The test sets were used to measure model performance, and test predictions were serialized and saved. The recommendation logic, which takes the maximum prediction probability, selected the winning offer type. For customers who received only negative predictions, they received a randomly selected offer type.
- One of the challenges was building the attribution logic to attribute offer events to transactions. I took on the challenge to build this out because, in my mind, having this aggregated view would expedite every part of this project.
- Another challenge was thinking through how to approach the problem. The process of developing a framework to tackle the problem took time.
- The feature engineering step was time consuming and coming up with features also took a long time. This stage required critical and creative thinking. At the end of the day, you want good features with good predictive power.
- Feature selection process required a lot of thinking as well. There is no standard process for this, but I came up with an approach that was well suited for this project.

Improvement

- The script run times could be improved by parallelizing the execution process with Dask.
- The full pipeline could be more standardized and redundant code could be removed as well.
- Other feature selection techniques could be leveraged to improve feature selection process.
- The attribution logic used in the transaction_engagement script could be improved to capture edge cases.
- Other model algorithms could be explored to improve overall model performance.
- Number of model estimators could be reduced by grouping similar offer types together.