

Kaggle Challenge Report

Kaggle Team: YYDS

Lanxin LI Meng XIA Hanqi YANG

1. Exploratory Data Analysis (Detailed analysis in Appendix 1: Exploratory Data Analysis (EDA))

Checking Missing values: 'Payment_Method' and 'Referral_Source' accounted for approximately 15% of the total dataset. Therefore, in feature engineering, we filled categorical variables with 'unknown' instead of the mode.

The data showed significant class imbalance, requiring consideration of balancing through model parameters or SMOTE during feature engineering.

Categorical variables had inconsistencies in type and format, such as case mismatches and confusion between 0 and o (e.g., afterNoon, afterno0n). We took some operations in Feature Engineering to ensure data consistency.

2. Feature Engineering

2.1 Main Workflow

2.1.1 Handling Missing Values

For numerical columns: Missing values were filled using the mean of the respective columns.

For categorical columns: We manually defined the categorical and numerical columns. Some columns were originally of the 'float' data type, others were originally of the 'object'. Based on the issues identified earlier in the EDA, we divided the categorical variables into two categories: Original String Columns and Converted String Columns.

Original String Columns: These were the original categorical variables. All strings were converted to lowercase, and any instances of '0' were replaced with 'o'.

Converted String Columns: These were originally numeric variables that needed to be converted into categorical string types. The data in these columns were transformed into string format.

Additionally, implicit missing values in categorical columns (e.g., '', 'nan', 'N/A', '<NA>', 'None') were replaced with pd.NA as the default missing value representation. The missing value status was then re-checked to ensure no unintended changes occurred during the conversion.

For features with a high percentage of missing values, such as 'Payment_Method' and 'Referral_Source', we used the value 'unknown' to fill the missing values, creating a separate category. For other categorical variables, missing values were filled with the mode of the respective columns.

2.1.2 One-Hot Encoding and Normalization of Numerical Variables

Categorical variables were transformed into numerical representations using one-hot encoding, enabling them to be processed by machine learning models. The resulting cleaned datasets are shown in the Appendix (Figure 1 & Figure 2).

Numerical variables in the numerical columns were normalized to eliminate the influence of data scale on model training.

2.2 New Feature Construction

We used a comparative approach to evaluate the impact of a newly added feature. First, we modeled with the original feature list, then with the addition of the new feature to assess its contribution.

The new feature, $\text{Effective_Price} = \text{Price} * (1 - \text{Discount} / 100)$, combines price and discount to reflect the customer's actual expenditure more accurately. This feature is highly relevant to purchasing behavior, as it highlights how discounts adjust the payment amount.

In the XGBoost model (see Appendix, Figure 3 & Figure 4), Effective_Price achieved an F-score of 51, showing its importance in predicting Purchase. Meanwhile, Price (F-score: 40) and Discount (F-score: 51) also retained high importance, demonstrating that Effective_Price effectively captures the interaction between them and enhances the model's understanding of purchasing behavior.

2.3 Feature Selection

2.3.1 Statistical Selection

For numerical features: We performed a correlation analysis and found that the correlation coefficients between variables were all less than 0.05. This indicates that including all numerical variables in the model is unlikely to result in multicollinearity issues.

For categorical features: We initially used the Chi-Square test to filter categorical variables with a significance level below 0.05. However, we observed that modeling with the filtered features resulted in a significantly lower F1-score compared to using all categorical features in the model.

2.3.2 Analysis Based on Selected Models

With 6,500 training samples and 33 features (34 after adding the new feature):

Logistic Regression: Handles this feature scale well, especially with regularization, so it is not a high-dimensional issue.

XGBoost: Automatically selects relevant features based on split gain, minimizing the impact of redundant features.

Random Forest: Randomly selects features during tree splitting, making it robust to high-dimensional data, with minimal performance impact from redundancy.

Since each feature has its business significance (see Appendix 2.3) and based on the above analysis, we input all features directly into the three models we chose to use and ultimately calculate feature importance.

3. Model tuning and comparison

3.1 XGBoost

3.1.1 XGBoost without new feature

We used GridSearchCV for comprehensive hyperparameter tuning. Using F1-score as the evaluation metric, the best parameters were identified through 5-fold cross-validation: 'colsample_bytree': 0.8, 'gamma': 0.2, 'learning_rate': 0.1, 'max_depth': 3, 'min_child_weight': 1, 'n_estimators': 50, 'reg_alpha': 0, 'reg_lambda': 5, 'scale_pos_weight': 4.

The XGBoost 'scale_pos_weight' parameter was used to adjust the class imbalance problem.

Regularization parameters (reg_alpha and reg_lambda) were introduced in parameter tuning, and the tree depth was restricted (max_depth=3) to prevent excessive model complexity. A maximum of 500 iterations was set, and early stopping (set to 10 rounds) was applied to prevent overfitting while saving training time.

The final results were F1 Score: 0.3517, Class 0 Precision: 0.92, Recall: 0.86, Class 1 Precision: 0.3, Recall: 0.43. Kaggle Score: 0.41365.

3.1.2 XGBoost with new feature 'Effective_Price'

Using the same parameter tuning and overfitting control settings as in 3.1.1, the final results were F1 Score: 0.3746, Class 0 Precision: 0.92, Recall: 0.87, Class 1 Precision: 0.32, Recall: 0.45. This indicates the model's limited performance in balancing precision and recall. The model still tends to favor the majority class, and further optimization is needed for the precision and recall of the minority class. Kaggle Score: 0.41282.

3.2 Logistic Regression

3.2.1 Logistic Regression + SMOTE + without New Feature

First, we applied SMOTE to oversample the data, balancing the number of samples between class 0 and class 1, each with 5,744 samples. This prevents the model from favoring one class over the other.

We used GridSearchCV with 5-fold cross-validation to tune hyperparameters, focusing on optimizing F1-score. The optimal parameters were: C: 10, class_weight: None, penalty: 'l2', solver: 'liblinear'.

To prevent overfitting, we used L2 regularization to limit model complexity, applied 5-fold cross-validation for performance stability, and set max_iter=1000 to ensure full convergence.

Finally, we got Class 0 Precision: 0.89, Recall: 0.98. Class 1 Precision: 0.98, Recall: 0.87. Cross-Validated F1-score: 0.9277 ± 0.0041 . Kaggle Score: 0.27702. The significant gap between the F1-score and the Kaggle score is attributed to overfitting caused by SMOTE. Despite our attempts to prevent overfitting, oversampling prior to training likely resulted in data leakage.

3.2.2 Logistic Regression + SMOTE + New Feature

To evaluate the impact of the new feature effective price, we used a comparative experimental method. After GridSearchCV, the optimal parameters were: C: 10, class_weight: 'balanced', penalty: 'l2', solver: 'lbfgs'.

Finally, we got Class 0 Precision: 0.89, Recall: 0.99. Class 1 Precision: 0.98, Recall: 0.88. Cross-Validated F1-score: 0.9289 ± 0.0049 . Kaggle Score: 0.28169. Although the model showed slight improvement in minority class detection (class 1), the gain was minimal. Feature importance analysis revealed that effective price ranked lowest, contributing little to F1-score (see Appendix 3.2.2 Figure 5). Additionally, the overfitting issue persisted.

3.2.3 Logistic Regression + No SMOTE + without New Feature

To avoid overfitting caused by SMOTE, we used the built-in parameter class_weight='balanced' in logistic regression. Hyperparameter tuning via GridSearchCV yielded the following optimal parameters: C: 10, class_weight: 'balanced', penalty: 'l2', solver: 'liblinear'.

Finally, we got Class 0 Precision: 0.94, Recall: 0.71. Class 1 Precision: 0.24, Recall: 0.66. Cross-Validated F1-score: 0.3829 ± 0.0163 . Kaggle Score: 0.38534.

3.2.4 Logistic Regression + No SMOTE + New Feature

Using the same comparative method, we assessed the impact of the new feature effective price on logistic regression performance. After tuning, the optimal parameters were: C: 10, class_weight: 'balanced', penalty: 'l2', solver: 'liblinear'.

Finally, we got Cross-Validated F1-score: 0.3820 ± 0.0166 . Kaggle Score: 0.38862. While training set performance slightly decreased, the Kaggle score improved marginally compared to the model without effective price. Feature importance analysis showed that the ranking of effective price improved slightly but still had limited impact on F1-score (see Appendix 3.2.4 Figure 6).

3.3 Random Forest

3.3.1 Random Forest + SMOTE

The initial Random Forest model used predefined hyperparameters (max_depth=10, n_estimators=300, min_samples_split=5, min_samples_leaf=2, max_features='sqrt') and balanced class weights. SMOTE was applied to oversample the minority class, increasing its proportion to approximately 15%. Numerical features were standardized, while categorical features were passed directly to the model.

This model achieved a weighted F1 score of 0.896 on the validation set, demonstrating strong overall performance. The precision and recall for the majority class (0) were 0.90, while the minority class (1) achieved a recall of 0.89. However, the Kaggle score for this model was 0.40816, indicating reasonable alignment with validation performance but also a tendency to overfit the SMOTE-adjusted data. Adjusting the classification threshold to 0.35 could be a potential step to further improve minority class recall and reduce misclassification of majority samples.

3.3.2 Random Forest with Stratified K-Fold, SMOTE, and Hyperparameter Optimization

To further improve performance, Randomized Search was employed to optimize hyperparameters across 50 iterations. The best settings identified were max_depth=15, n_estimators=500, min_samples_split=10, min_samples_leaf=2, and max_features='sqrt'. Using these parameters, Stratified K-Fold cross-validation was applied to ensure consistent class distribution across folds while using SMOTE within each fold to address class imbalance.

This approach achieved a mean weighted F1 score of 0.7786 across folds, with a standard deviation of 0.0121, reflecting stable performance. However, the minority class F1 score remained low, indicating the model struggled to fully capture minority class patterns. The Kaggle score for this approach dropped to 0.37533, lower than the initial Random Forest model, suggesting overfitting to synthetic patterns introduced by SMOTE. Although class proportions improved (70.94% for 0, 29.06% for 1), the low minority class recall highlights limitations in learning minority class features effectively.

Appendix

1.Exploratory Data Analysis (EDA)

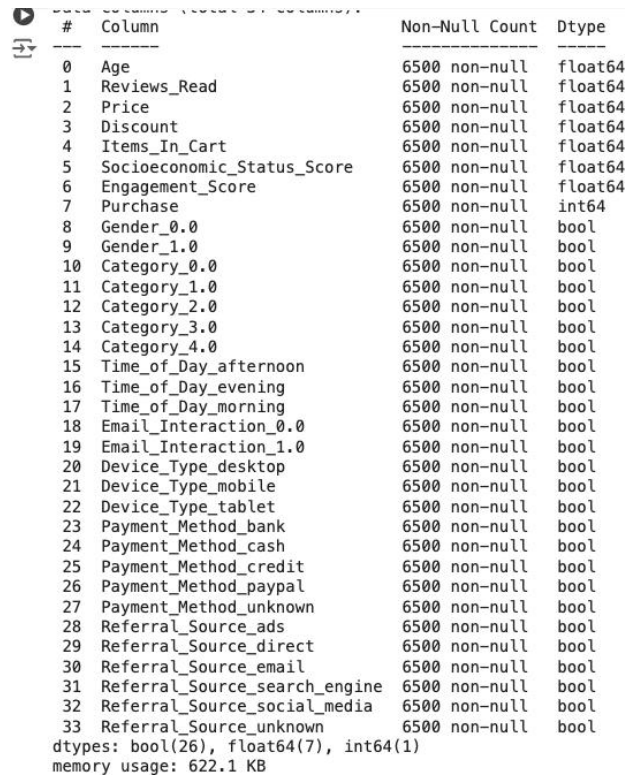
We first examined the missing values in the 'train_dataset' and 'eval_dataset'. We found that all feature variables in the datasets contained missing values. Specifically, in the training dataset, the 'Age' column had 970 missing values; the 'Payment_Method' column had the highest number of missing values at 1009; and the 'Referral_Source' column had 997 missing values. The missing values in these three features accounted for approximately 15% of the total dataset. Therefore, in the subsequent missing value handling process, we decided to fill the categorical variables 'Payment_Method' and 'Referral_Source' with an 'unknown' category rather than using the mode. Notably, the target variable 'Purchase' had no missing values.

We also checked whether the target variable data was imbalanced. We found that class '0' accounted for 88.27%, while class '1' accounted for 11.63%, resulting in a class ratio of approximately 7.6:1. This indicates a significant class imbalance. Therefore, during the model training phase, we considered either using the built-in parameters of the model to handle imbalanced data or applying SMOTE for data preprocessing.

We analyzed the distribution of numerical variables based on the 'Purchase'. We observed that the features 'Price', 'Discount', and 'Reviews_Read' exhibited noticeable differences between 'Purchase=0' and 'Purchase=1', while other features did not show significant differences. This observation supported our subsequent idea to construct new features, focusing on further exploring the information in 'Price' and 'Discount'.

We manually defined the categorical and numerical columns. Among the categorical columns, the variables 'Gender', 'Category', and 'Email_Interaction' were originally of the 'float' data type. Other variables were originally of the 'object' data type, but inconsistencies in character formats were present in both datasets, such as case mismatches and confusion between '0' and 'o'(e.g., 'afterNoon', 'afterno0n'). We addressed these issues with specific processing steps during feature engineering.

2.1.2 One-Hot Encoding and Normalization of Numerical Variables



#	Column	Non-Null Count	Dtype
0	Age	6500 non-null	float64
1	Reviews_Read	6500 non-null	float64
2	Price	6500 non-null	float64
3	Discount	6500 non-null	float64
4	Items_In_Cart	6500 non-null	float64
5	Socioeconomic_Status_Score	6500 non-null	float64
6	Engagement_Score	6500 non-null	float64
7	Purchase	6500 non-null	int64
8	Gender_0.0	6500 non-null	bool
9	Gender_1.0	6500 non-null	bool
10	Category_0.0	6500 non-null	bool
11	Category_1.0	6500 non-null	bool
12	Category_2.0	6500 non-null	bool
13	Category_3.0	6500 non-null	bool
14	Category_4.0	6500 non-null	bool
15	Time_of_Day_afternoon	6500 non-null	bool
16	Time_of_Day_evening	6500 non-null	bool
17	Time_of_Day_morning	6500 non-null	bool
18	Email_Interaction_0.0	6500 non-null	bool
19	Email_Interaction_1.0	6500 non-null	bool
20	Device_Type_desktop	6500 non-null	bool
21	Device_Type_mobile	6500 non-null	bool
22	Device_Type_tablet	6500 non-null	bool
23	Payment_Method_bank	6500 non-null	bool
24	Payment_Method_cash	6500 non-null	bool
25	Payment_Method_credit	6500 non-null	bool
26	Payment_Method_paypal	6500 non-null	bool
27	Payment_Method_unknown	6500 non-null	bool
28	Referral_Source_ads	6500 non-null	bool
29	Referral_Source_direct	6500 non-null	bool
30	Referral_Source_email	6500 non-null	bool
31	Referral_Source_search_engine	6500 non-null	bool
32	Referral_Source_social_media	6500 non-null	bool
33	Referral_Source_unknown	6500 non-null	bool

dtypes: bool(26), float64(7), int64(1)
memory usage: 622.1 KB

Figure 1 train dataset

#	Column	Non-Null Count	Dtype
0	Age	3500 non-null	float64
1	Reviews_Read	3500 non-null	float64
2	Price	3500 non-null	float64
3	Discount	3500 non-null	float64
4	Items_In_Cart	3500 non-null	float64
5	Socioeconomic_Status_Score	3500 non-null	float64
6	Engagement_Score	3500 non-null	float64
7	Gender_0.0	3500 non-null	bool
8	Gender_1.0	3500 non-null	bool
9	Category_0.0	3500 non-null	bool
10	Category_1.0	3500 non-null	bool
11	Category_2.0	3500 non-null	bool
12	Category_3.0	3500 non-null	bool
13	Category_4.0	3500 non-null	bool
14	Time_of_Day_afternoon	3500 non-null	bool
15	Time_of_Day_evening	3500 non-null	bool
16	Time_of_Day_morning	3500 non-null	bool
17	Email_Interaction_0.0	3500 non-null	bool
18	Email_Interaction_1.0	3500 non-null	bool
19	Device_Type_desktop	3500 non-null	bool
20	Device_Type_mobile	3500 non-null	bool
21	Device_Type_tablet	3500 non-null	bool
22	Payment_Method_bank	3500 non-null	bool
23	Payment_Method_cash	3500 non-null	bool
24	Payment_Method_credit	3500 non-null	bool
25	Payment_Method_paypal	3500 non-null	bool
26	Payment_Method_unknown	3500 non-null	bool
27	Referral_Source_ads	3500 non-null	bool
28	Referral_Source_direct	3500 non-null	bool
29	Referral_Source_email	3500 non-null	bool
30	Referral_Source_search_engine	3500 non-null	bool
31	Referral_Source_social_media	3500 non-null	bool
32	Referral_Source_unknown	3500 non-null	bool

dtypes: bool(26), float64(7)
memory usage: 307.6 KB

Figure 2 eval dataset

2.2 New Feature Construction

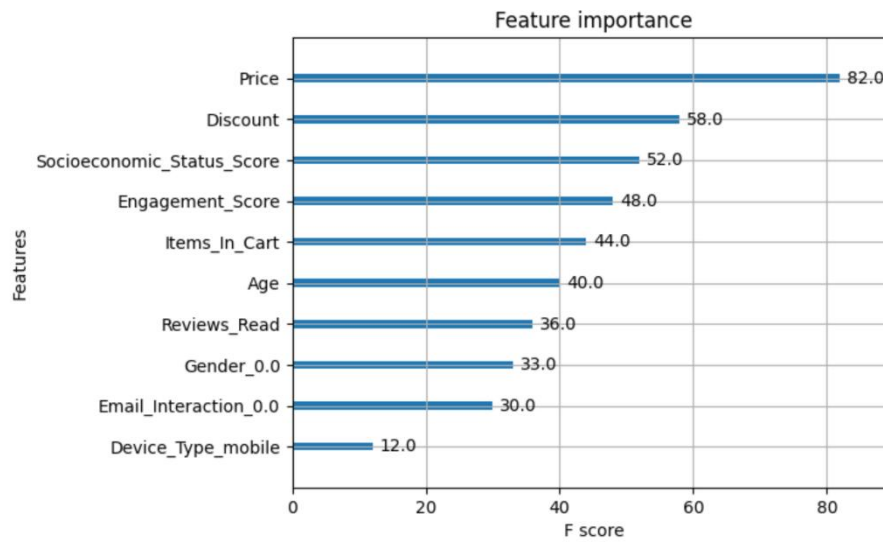


Figure 3 XGBoost without new feature

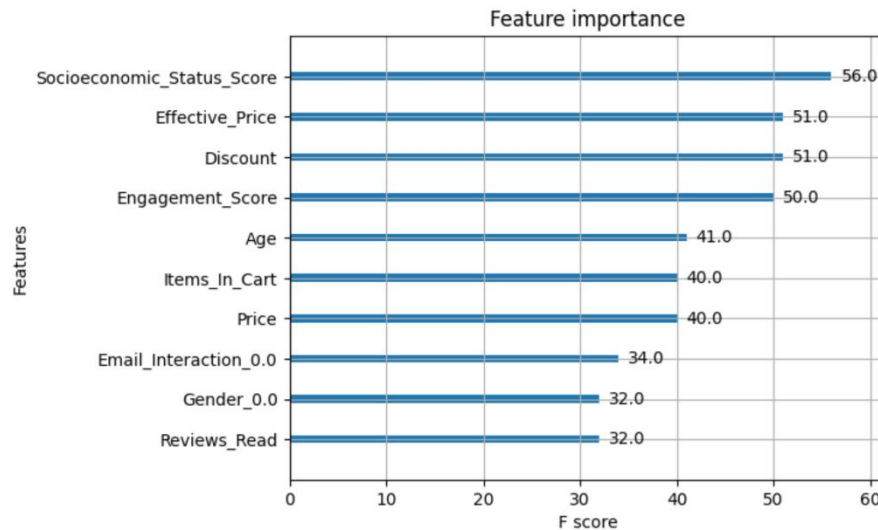


Figure 4 XGBoost with new feature: Effective_Price

2.3 Feature Selection

Business Significance Analysis

Age, Gender, Socioeconomic_Status_Score: Analyze demographics' impact on purchases.

Reviews_Read, Time_of_Day: Understand browsing behavior and timing effects.

Price, Discount, Category: Assess product factors on purchase decisions.

Items_In_Cart: Reflect intent through cart quantity.

Email_Interaction, Engagement_Score, Device_Type: Measure engagement and device preferences, providing a multi-dimensional information basis for predicting user purchasing behavior.

Payment_Method, Referral_Source: Identify payment and discovery preferences.

3.2.2 Logistic Regression + SMOTE + New Feature

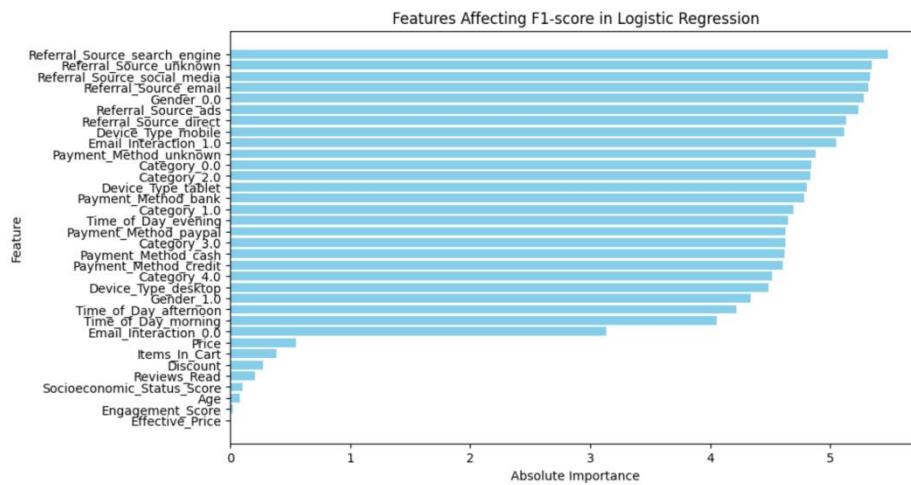


Figure 5 Logistic Regression + SMOTE + New Feature

3.2.4 Logistic Regression + No SMOTE + New Feature

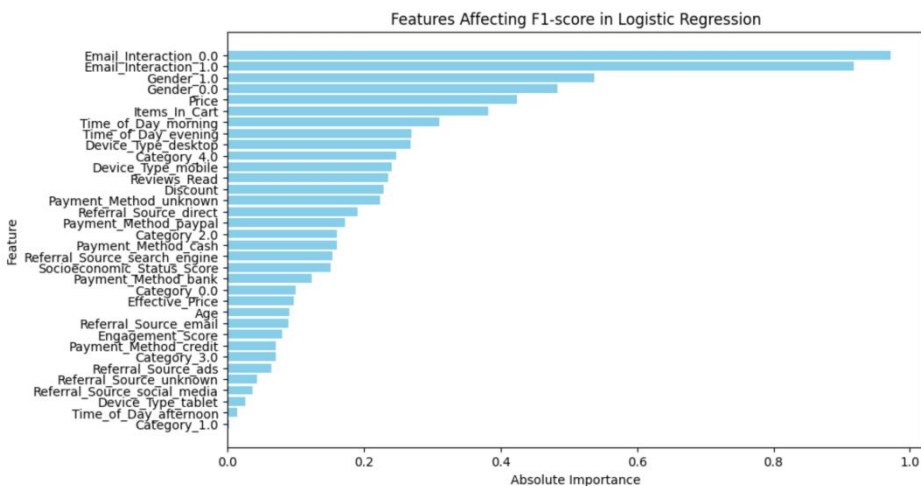


Figure 6 Logistic Regression + No SMOTE + New Feature