UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

BUDT 758T Final Project Report Spring 2025

Section 1: Team member names and contributions

Name	Contribution
Maitreyee Tiwari	Performed initial exploratory data analysis and handled data cleaning tasks. Also worked on logistic regression model
Abimanyu Krishnamoorthy	Engineered and transformed features, including creation of derived variables. Also worked on Logistic regression model.
Rohan Vasudevan	Trained and implemented the random forest classification model. Also worked on XGboost model.
Pranav Bharatwaj Manikantan	Worked on XGboost model. Tuned model hyperparameters, evaluated multiple thresholds, and selected the final model.
Aditya Kaushik	Created performance plots, fitting curves, supported feature analysis, and assembled the final report.

Section 2: Business Understanding

Airbnb is a global online platform that matches those who wish to rent out their homes with those who need to stay somewhere. Hosts post a range of properties from shared rooms to entire houses and guests leave public reviews that build the credibility and future success of each property. Ratings are a significant feedback mechanism, one that influences how visible a listing is on the website, its ability to book, and the host's potential earnings.

A perfect rating score (100%) indicates that a listing has consistently delivered outstanding experiences across all categories like cleanliness, accuracy, communication, location, check-in, and value. It is a badge of honor for hosts. It is a guest trust indicator. Highly rated listings are believed to be prioritized by Airbnb's own algorithms in search rankings, which can significantly influence booking rates and long-term revenues for hosts.

Our initial objective is to build a predictive model that can predict and label whether a listing will receive a perfect rating score based solely on its features and historical performance. We participated in Contest 1 of the data mining competition BUDT758T, and the objective was to optimize and maximize such predictions' True Positive Rate (TPR) so that False Positive Rate (FPR) is below 10%.

Who can utilize this model?

1. **Individual Hosts and Property Managers**
   Hosts can forecast in advance how probable their listing is to earn a perfect rating even before checking-in guests. They will be able to use these findings to work on their listings. This includes lowering the price, adding amenities, improving response time, etc. to improve guest satisfaction. It can be a performance dashboard for managers of several properties.

2. **Airbnb's Internal Analytics and Quality Assurance Teams**
   The model can help the platform highlight the listings with the highest likelihood of a perfect guest experience. It can flag listings likely to underperform for host guidance, compliance audits, or targeted improvements. It can help guest search ranking models by introducing a quality prediction parameter. It also shows what factors people are most concerned with when choosing a listing to rent.

What could be done with model output?

For low predicted probability of perfect ratings:

- Hosts could be notified to optimize key factors (e.g., price, minimum stay, speed of communication). Airbnb could suppress short-term visibility or notify the host with quality improvement recommendations.
- Listings could be offered with supporting packages like professional photographs or intelligent pricing plans.

For top-predicted score listings:

- Airbnb can make them more prominent in search results.
- Hosts can be encouraged to list more properties.
- They can be used as best-practice case studies or benchmarks.

Business value of this predictive model

Improved guest satisfaction: By pointing guests toward likely high-performing listings and enabling hosts to correct early in a self-driven manner.
Higher app/platform credibility: Airbnb can avoid customer churn by actively identifying low-quality listings.
Operational efficiency: Airbnb can reduce manual audits and deploy quality assurance efforts more efficiently.
Revenue boost: Rated-up listings have higher booking volumes so the model helps hosts

optimize for such success drivers.

This type of model will educate hosts about what is working and what is not. It also educates them to be data driven. Such a model encourages both hosts and Airbnb to make informed, strategic decisions on listings and get rewarded. Hosts are given actionable insights into their prospect performance, while Airbnb safeguards guests' experience and optimizes revenue, thereby living up to its brand promise of quality and trust.

Section 3: Data Understanding and Data Preparation

Overall, we used 20 features in our final models, however, we feature engineered/cleaned 14 fields, as well as impute and clean 28 numeric fields. Totally we processed 42 fields in the dataset.
The table below highlights the final feature list we used for our models. We processed multiple other fields as well, but did not use them due to high missingness, requirement for imputing values, or too many categories, not justifying their usage.

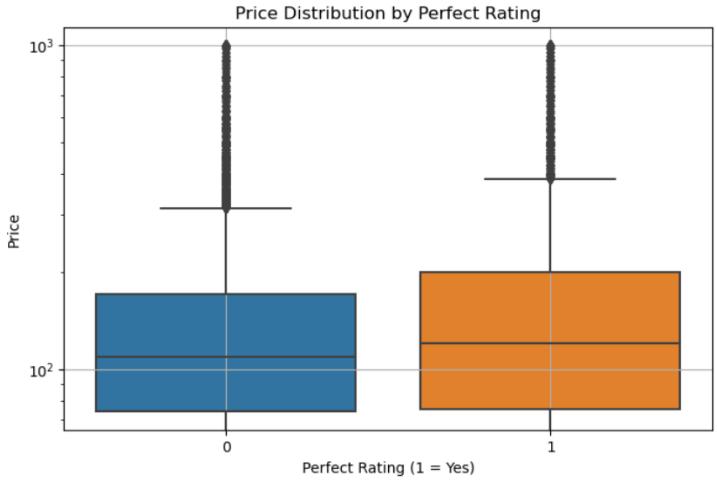
ID	Feature Name	Description	Python Code Line Numbers
1	accommodates	Original feature from dataset	20, 33, 382
2	availability_30	Original feature from dataset	33, 382
3	availability_365	Original feature from dataset	33, 382
4	bathrooms	Original feature from dataset	33, 382
5	bed_category	Derived from bed_type (consolidated into 2 groups)	139–144
6	cancellation_policy	Original, but simplified into fewer categories	134–136
7	has_cleaning_fee	Derived: binary flag from cleaning_fee	160–164
8	host_response_time	Original, but mapped to ordinal numeric scale	177–181
9	host_response_rate	Original, cleaned and scaled to decimal	33, 95–96
10	host_acceptance_rate	Original, cleaned and scaled to decimal	33, 98–100
11	host_listings_count	Original feature from dataset	33, 382
12	host_tenure_days	Derived: combining host start date and review age	171–176
13	minimum_nights	Original feature from dataset	33, 382
14	price	Original feature from dataset	33, 382
15	monthly_price	Original feature from dataset	33, 382
16	host_verifications	Original feature from dataset	382
17	property_category	Derived: consolidated from property_type	125–131
18	room_type	Original feature from dataset	382

19	has_security_deposit	Derived: binary flag from security_deposit	167–169
20	days_from_first_review	Derived: time since listing's first review	33, 165–166

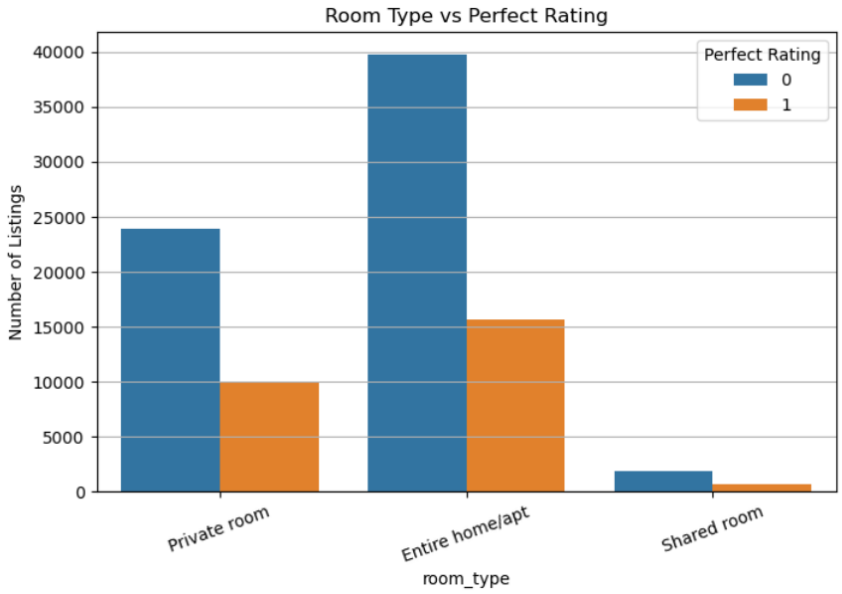
Feature Insights:

We also plotted graphs that help interpret relationships with the target (perfect_rating_score) or reveal useful patterns in the data.

- This boxplot visualizes the distribution of listing prices for properties with and without a perfect rating. While both groups show a wide price range, listings with perfect ratings tend to have slightly higher median prices, suggesting that price alone does not negatively impact high ratings.

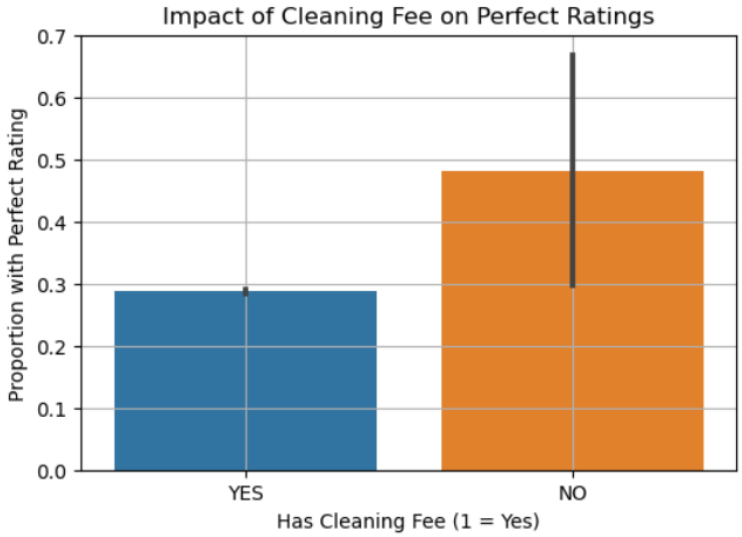


- This grouped bar chart compares the number of listings with and without perfect ratings across different room types. Listings with entire homes or apartments are not only the most common but also more likely to receive perfect ratings compared to private or shared rooms.

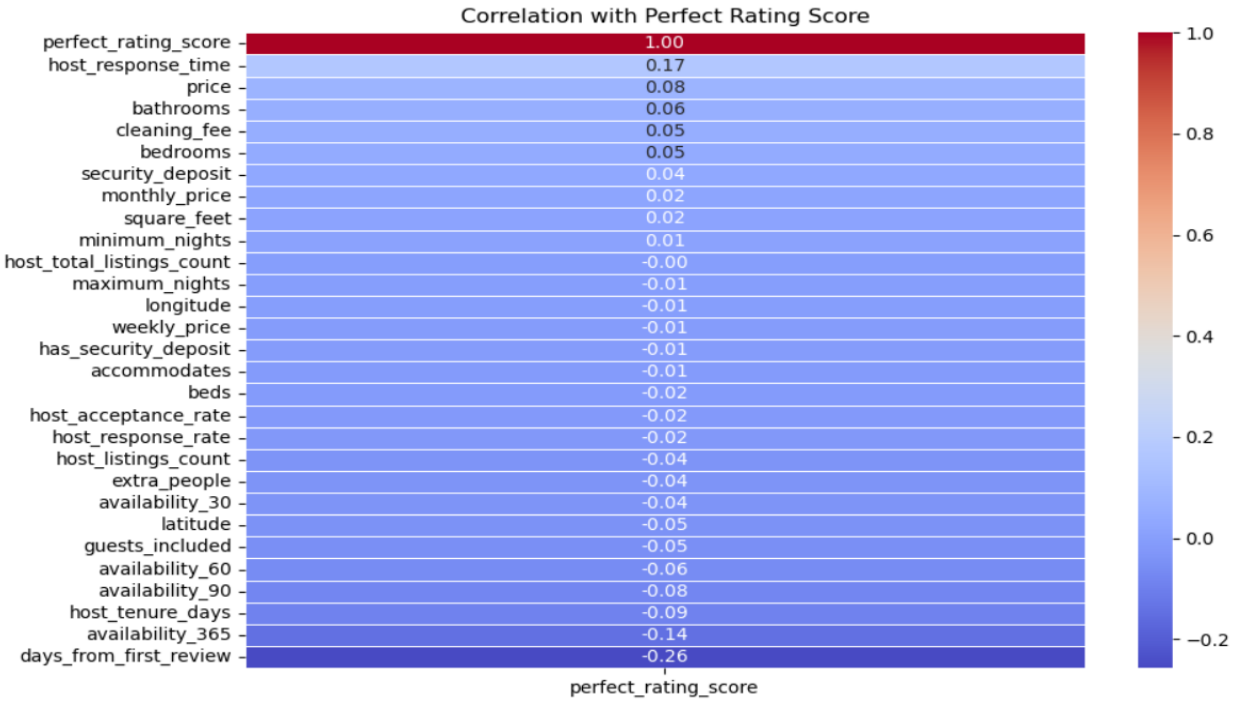


- This stacked bar chart shows the proportion of perfect ratings across different cancellation policies. Listings with flexible policies have a noticeably higher share of perfect ratings compared to moderate and strict policies, suggesting that guests prefer flexibility.

- This bar chart compares the proportion of perfect ratings between listings that charge a cleaning fee and those that don't. Listings without a cleaning fee receive significantly more perfect ratings, suggesting that additional charges may negatively impact guest satisfaction.



- This heatmap displays the correlation of numeric features with the perfect_rating_score. Most features show weak correlation, but host_response_time has the strongest positive association, while days_from_first_review is moderately negatively correlated, suggesting that newer listings may receive more perfect ratings.



Data Preparation Summary:

- Null values in numerical columns were replaced with mean.
- Categorical columns were replaced with "Not available" and then one-hot encoded.

- Features like host_response_rate, host_acceptance_rate were cleaned and normalized to numeric format.
- Categorical consolidation was done for property_type and bed_type.
- New binary flags: has_cleaning_fee, has_security_deposit were added.
- A derived feature host_tenure_days was added to estimate host experience.

Section 4: Evaluation and Modeling

1. **Winning Model:**

   The winning model was an XGBoost classifier trained on fields_considered_v2, which included 20 engineered features. After tuning multiple hyperparameters and evaluating different classification thresholds, we selected a threshold of 0.78, which provided an ideal tradeoff between TPR (0.1874) and FPR (0.0392) on the test set, below the 10% FPR constraint and also with enough room to account for uncertainty on the external data set.

- **Final model hyperparameters:**
  n_estimators: 300
  max_depth: 3
  min_child_weight: 7
  learning_rate: 0.2
  scale_pos_weight: computed from train data

- **Code Line Numbers:**
  Model training: Lines 618–677
  Threshold evaluation and generalization performance: Lines 678–723
  Final predictions: Lines 1038–1054

2. Models Used:

i. Logistic Regression:

a) Model Family : Logistic Regression

b) Packages Used:

- sklearn.linear_model.LogisticRegression: For training the logistic regression classifier
- sklearn.metrics: confusion_matrix, classification_report, accuracy_score, roc_curve
- pandas: For data manipulation and feature creation
- numpy: For numerical operations
- sklearn.preprocessing.OneHotEncoder: For encoding categorical variables
- sklearn.impute.SimpleImputer: For handling missing values
- scipy.sparse: csr_matrix, hstack for combining encoded and numeric matrices

- sklearn.model_selection.train_test_split: For stratified data splitting
- matplotlib.pyplot: For fitting and ROC curve plotting

c) Training and Generalization Performance:

We trained the logistic regression model using a classification threshold of 0.5. On the test set, the model achieved an accuracy of 0.7263, True Positive Rate (TPR) of 0.1969 and a False Positive Rate (FPR) 0.0604. This FPR is not good enough to accommodate uncertainty in the external dataset. These results indicate that while the model satisfied the FPR constraint of 10%, it failed to capture enough positive instances. The recall of ~19.7% suggests that the model struggled to identify listings that achieved a perfect rating, highlighting its limitations for this prediction task.

d) How did we estimate it?

We used a stratified 70/30 train-test split from the labeled dataset to evaluate generalization performance. The stratification preserved class balance. We varied the classification threshold between 0.5 and 0.8 and observed recall and precision metrics. Evaluation was based on fixed-threshold performance on the held-out test set.

e) Features:

The final feature set we used was:

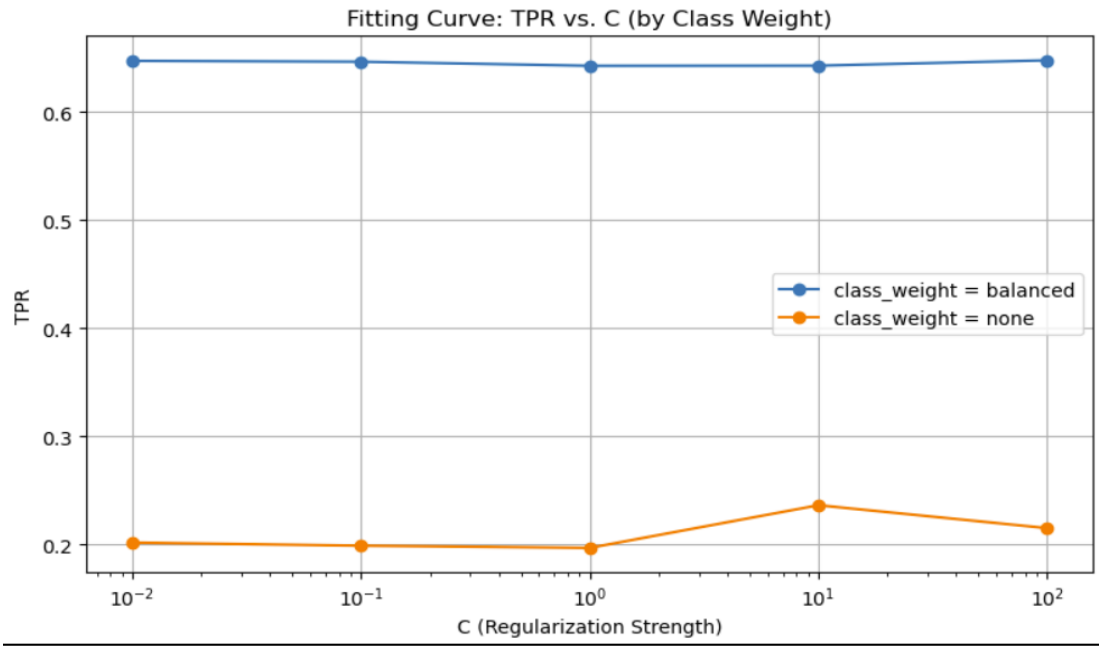
```
[
   'accommodates',
   'availability_30',
   'availability_365',
   'bathrooms',
   'bed_category',
   'cancellation_policy',
   'has_cleaning_fee',
   'host_response_time',
   'host_response_rate',
   'host_acceptance_rate',
   'host_listings_count',
   'host_tenure_days',
   'minimum_nights',
   'price',
   'monthly_price',
   'host_verifications',
   'property_category',
   'room_type',
   'has_security_deposit',
   'days_from_first_review',
   'perfect_rating_score'
]
```

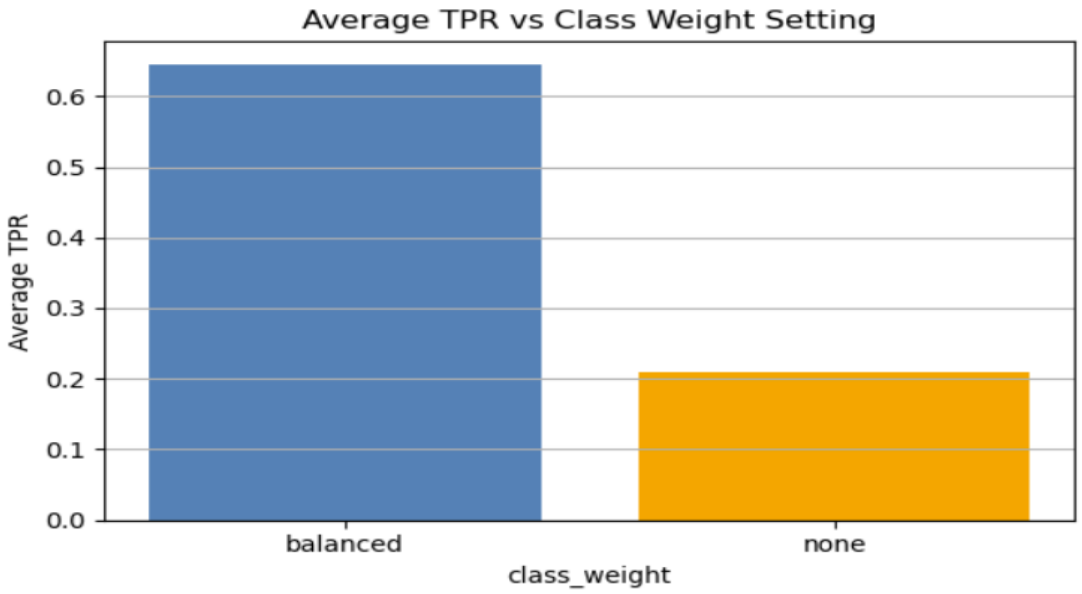
f) **Code Line Numbers**:
Model training: Lines 331–372
Threshold evaluation and Generalization performance: Lines 373–421

g) Hyperparameters:
We tuned two hyperparameters: C (inverse of regularization strength) and class_weight (to handle class imbalance)
We tested five values of C: [0.01, 0.1, 1, 10, 100] with both class_weight=None and class_weight='balanced', using a fixed threshold of 0.5.

h) Fitting Curves:

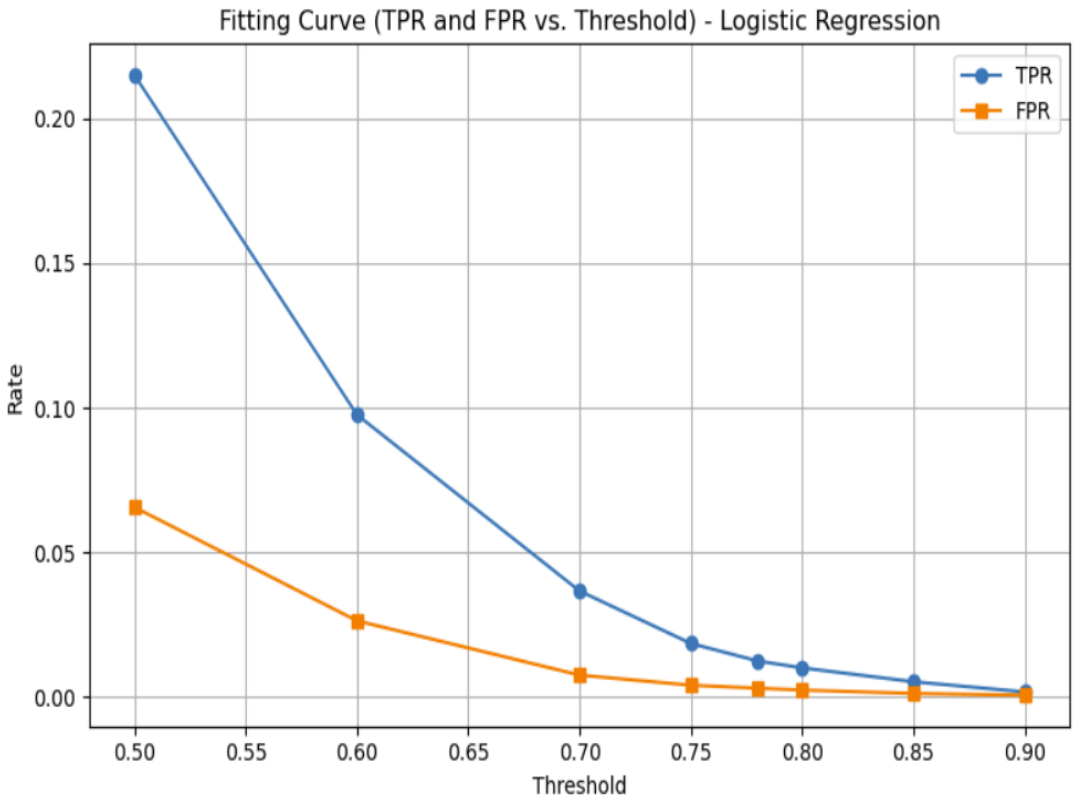
- A fitting curve was plotted to show the difference between the class_weight values as well as the C values. These results show that without balancing, TPR is consistently low but FPR remains within the contest constraint. With class balancing, TPR increases substantially but FPR exceeds 0.32. We also visualized the average TPR by class weight. On average, class_weight='balanced' achieved TPR = 0.6459 and class_weight=None achieved TPR = 0.2078. This shows that class_weight improves recall, but the false positive rate also skyrockets in this use case.





C	class_weight	TPR	FPR	Accuracy
0.01	none	0.2018	0.0615	0.7269
0.01	balanced	0.6471	0.3218	0.6692
0.10	none	0.1989	0.0588	0.7280
0.10	balanced	0.6463	0.3195	0.6707
1.00	none	0.1969	0.0604	0.7263
1.00	balanced	0.6426	0.3205	0.6689
10.00	none	0.2363	0.0710	0.7300
10.00	balanced	0.6427	0.3208	0.6687
100.00	none	0.2151	0.0656	0.7278
100.00	balanced	0.6476	0.3212	0.6699

- A fitting curve was plotted to analyse TPR ad FPR vs threshold values. As we can see, the logistic regression model at 0.5 threshold provides a decent TPR and FPR tradeoff, however the 6% FPR is still too close to account for uncertainty in the external data set.



ii. Random Forest

a) Model Family : Random Forest (Ensemble Decision Trees)

b) Packages Used:

- sklearn.ensemble.RandomForestClassifier: For model training
- sklearn.metrics: For performance evaluation
- sklearn.model_selection.train_test_split: For splitting train/test sets
- sklearn.preprocessing.OneHotEncoder: For categorical encoding
- scipy.sparse: For combining numeric and encoded features
- pandas, numpy: For data manipulation and math

c) **Training and Generalization Performance:**
The Random Forest with 100 trees (n_estimators=100) achieved an accuracy of 0.7337, TPR (Recall): 0.2968, FPR: 0.0902. This FPR is not good enough to accommodate uncertainty in the external dataset. The recall is close to 30% which is higher than the logistic regression model.

d) **How did we estimate it?**

We used a stratified 70/30 train-test split on the labeled dataset to evaluate generalization. Class proportions were preserved between the train and test sets using stratify=y to ensure fair evaluation.

e) Features:

The final feature set we used was:

```
[
  'accommodates',
  'availability_30',
  'availability_365',
  'bathrooms',
  'bed_category',
  'cancellation_policy',
  'has_cleaning_fee',
  'host_response_time',
  'host_response_rate',
  'host_acceptance_rate',
  'host_listings_count',
  'host_tenure_days',
  'minimum_nights',
  'price',
  'monthly_price',
  'host_verifications',
  'property_category',
  'room_type',
  'has_security_deposit',
  'days_from_first_review',
  'perfect_rating_score'
]
```

f) Code Line Numbers:

Model training: Lines 530–579
Threshold evaluation and Generalization performance: Lines 580–617

g) Hyperparameters:

We tuned two hyperparameters: n_estimators and max_depth.

The values we tried were:

- n_estimators: [100, 200, 300]
- max_depth: [None, 10, 20]

This resulted in 9 total combinations. Each model was evaluated using a fixed classification threshold of 0.5 by calculating confusion matrices and classification

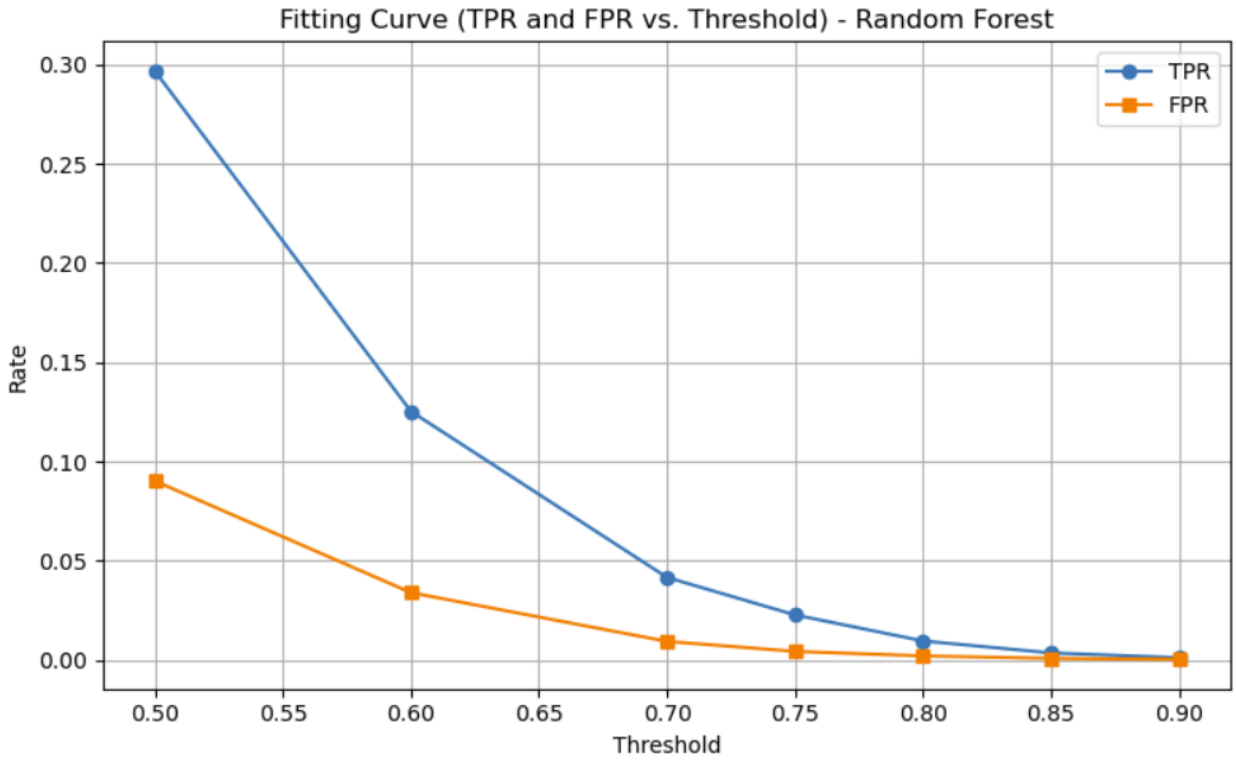
metrics (TPR, FPR, and Accuracy). We found that models with max_depth=None consistently achieved the highest true positive rate (TPR) while keeping the false positive rate (FPR) under the required 10% threshold. Increasing n_estimators beyond 100 offered only marginal improvement, so we retained the simpler configuration.

h) Fitting Curves:

- To evaluate threshold sensitivity, we created a fitting curve by varying the classification threshold from 0.5 to 0.9. At each threshold, we calculated the true positive rate (TPR) and false positive rate (FPR) using the confusion matrix.

Threshold	TPR	FPR	Threshold	TPR
0.50	0.2968	0.0902	0.50	0.2968
0.60	0.1253	0.0339	0.60	0.1253
0.70	0.0416	0.0093	0.70	0.0416
0.75	0.0227	0.0043	0.75	0.0227
0.80	0.0096	0.0020	0.80	0.0096
0.85	0.0035	0.0008	0.85	0.0035
0.90	0.0011	0.0003	0.90	0.0011

- The curve shows that as the threshold increases, both TPR and FPR decline rapidly. While higher thresholds reduce false positives significantly, they also eliminate true positives, which are critical for this task. The baseline threshold of 0.50 achieves the highest TPR (0.2968) while maintaining FPR within the contest constraint (0.0902). Based on this tradeoff, we retained a threshold of 0.50 for our final Random Forest predictions.



iii. <u>XGBoost</u>

a) Model Family : XGBoost

b) Packages Used:

- xgboost: For training the XGBoost classification model (XGBClassifier)
- sklearn.ensemble.RandomForestClassifier: For initial baseline comparison
- sklearn.metrics: confusion_matrix, classification_report, accuracy_score, roc_curve
- pandas: For data manipulation and feature creation
- numpy: For numerical operations
- sklearn.preprocessing: OneHotEncoder, StandardScaler (loaded but not used)
- sklearn.impute.SimpleImputer: For missing value handling
- scipy.sparse: csr_matrix, hstack for combining encoded and numeric feature matrices
- sklearn.model_selection.train_test_split: For creating stratified train/test splits
- matplotlib.pyplot: For ROC curve visualization
- itertools: For generating combinations during hyperparameter tuning

c) **Training and Generalization Performance:**
Our XGBoost model achieved a training set TPR of 0.1874 and FPR of 0.0392 using a classification threshold of 0.78. On the contest test set, our group (Group 2) achieved a TPR of 0.3487 and FPR of 0.0991, meeting the FPR constraint of 10%.

d) **How did we estimate it?**

Performance was estimated using a stratified 70/30 train-test split from the labeled data. We ensured that class distribution was preserved and tested generalization by varying the threshold from 0.3 to 0.8 to examine stability and required metric values.

e) **Features:**
The best-performing set of features is defined in fields_considered_v2, which includes 20 features such as host_tenure_days, has_cleaning_fee, price, property_category, and transformed categorical indicators like bed_category and cancellation_policy. Additional features like square_feet and weekly_price were considered earlier but removed due to high missingness or redundancy.

The final feature set we used was:

```
[
  'accommodates',
  'availability_30',
  'availability_365',
  'bathrooms',
  'bed_category',
  'cancellation_policy',
  'has_cleaning_fee',
  'host_response_time',
  'host_response_rate',
  'host_acceptance_rate',
  'host_listings_count',
  'host_tenure_days',
  'minimum_nights',
  'price',
  'monthly_price',
  'host_verifications',
  'property_category',
  'room_type',
  'has_security_deposit',
  'days_from_first_review',
  'perfect_rating_score'
]
```

f) **Code Line Numbers:**
Model training: Lines 618–677
Threshold evaluation and generalization performance: Lines 678–723

g) Hyperparameters:
We tuned four hyperparameters: n_estimators, max_depth, learning_rate, and min_child_weight.

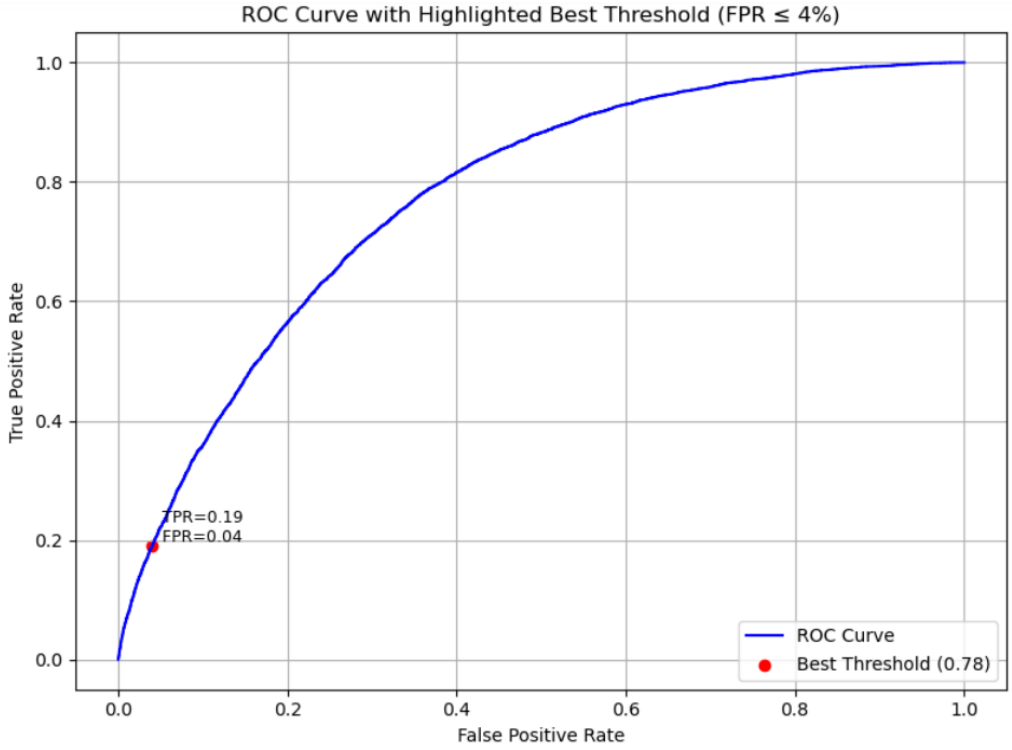
The values we tried were:

- n_estimators: [100, 300]
- max_depth: [3, 5]
- learning_rate: [0.1, 0.2]
- min_child_weight: [1, 3]

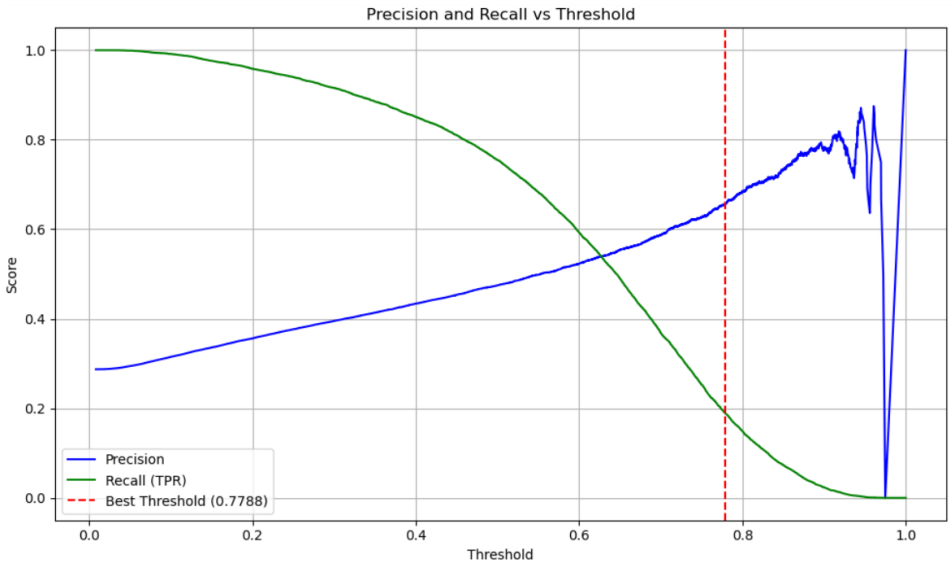
This resulted in 16 total combinations. Each model was evaluated at five thresholds (0.3 to 0.7) using confusion matrices and classification metrics.

h) **Fitting Curves:**

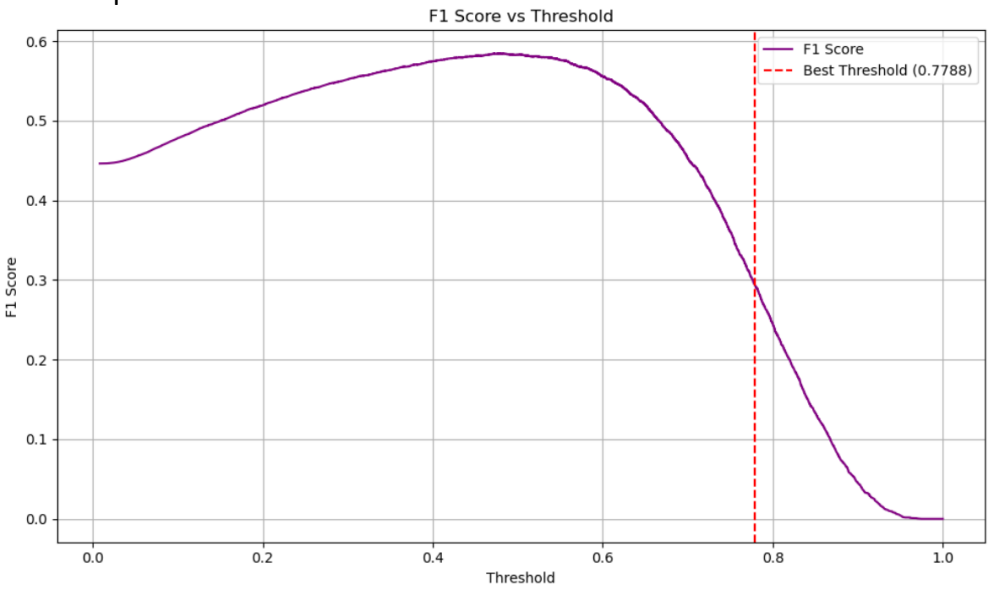
- A fitting curve was generated in the form of a threshold vs. TPR/FPR table to assess how changing thresholds impacts tradeoffs. Additionally, we created a ROC curve using sklearn.metrics.roc_curve to visualize TPR/FPR performance and identify the optimal cutoff that keeps FPR under 0.04. These plots guided our decision to use threshold = 0.78 for final predictions.



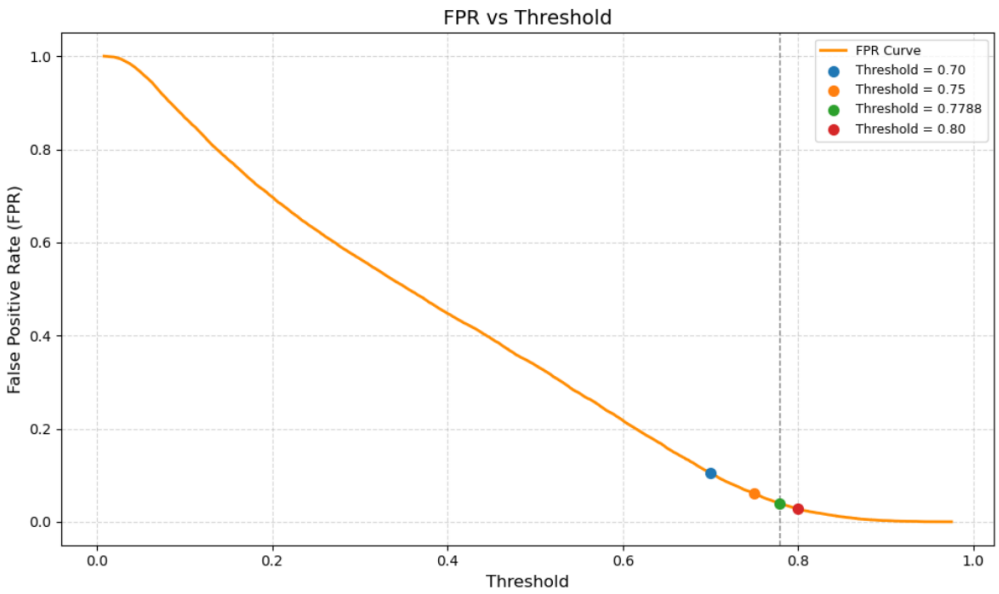
- This graph shows how precision and recall (true positive rate) vary across different classification thresholds. As the threshold increases, precision improves while recall declines, reflecting the typical trade-off between identifying true positives and avoiding false positives. The red dashed line indicates the optimal threshold (0.7788) selected based on the FPR constraint.



- This graph displays the variation of F1 Score across different classification thresholds. The F1 Score peaks at a mid-range threshold before declining, indicating the point where precision and recall are optimally balanced. The red dashed line highlights the chosen threshold (0.7788), optimized for controlling the false positive rate.



- This graph shows how the false positive rate (FPR) decreases as the classification threshold increases. The plotted points highlight FPR values at key thresholds (0.70, 0.75, 0.7788, and 0.80), showing that higher thresholds effectively reduce false positives, with 0.7788 offering a low FPR under 4%.



3. **<u>Learning Curves:</u>**
    - The main learning curve from the models was to understand what each hyperparameter does and how it affects the model. Tuning hyperparameters ultimately proved to be the solution to hitting the required TPR and FPR targets.
    - Initially, we saw poor performance from the logistic regression model, where the TPR was extremely low when we tried to reduce FPR. This was a learning curve because we had to decide whether it is worth tweaking the features here or adding interaction terms.
    - Similarly, with the random forest model, it is not the best equipped to handle our dataset. For a variety of reasons, the XGBoost turned out to be the best model because it handles imbalanced datasets better, and boosting helps predict perfect_rating_scores better if misclassified initially.
    - Lastly, the trade-off between TPR and FPR proved to be a huge learning curve. We essentially hit a trade-off point, beyond which we cannot optimize the model anymore.

We evaluated model performance by graphing fitting curves for all algorithms to observe how variation in threshold and hyperparameter tuning influenced key indicators such as true positive rate (TPR) and false positive rate (FPR).

Logistic Regression

We used thresholds from 0.5 to 0.9 and observed a big fall in TPR as well as FPR with a rise in the threshold. With the default 0.5 threshold, the model had TPR = 0.1969 and FPR = 0.0604. We also tuned regularization parameter C and attempted class_weight='balanced'. While balancing elevated TPR significantly (up to 0.64), it gave us an FPR over 0.32, which violated the rules of the contest. Logistic could not capture feature interactions needed to produce high recall with low FPR.

Random Forest
Random Forest initially came up with a good balance of TPR = 0.2968 and FPR = 0.0902 at threshold = 0.5. Fitting curves showed that increasing the threshold decreased both metrics dramatically. Tuning n_estimators and max_depth suggested that limiting depth led to underfitting, with trees larger than 100 offering minimal use. Threshold curves validated that our decision in using default configuration with n_estimators=100 and max_depth=None was optimal since it offered the best balance under competition conditions.

XGBoost
XGBoost was the most adaptable and the highest performing model when it came to handling FPR. Through the optimization of four hyperparameters (n_estimators, max_depth, learning_rate, and min_child_weight), we have identified settings that minimized FPR to at least 0.0392 without sacrificing TPR = 0.1874 on the validation set. Test results from the external set confirmed a TPR of 0.3487 and FPR of 0.0991 to achieve the required criteria. The threshold and ROC curves also showed that 0.78 was the best cutoff. Among all the models, XGBoost provided the best balance of generalizability, control over FPR, and recall performance.

Section 5: Reflection/takeaways

1) **What did your group do well?**
   Our team worked very well in preprocessing and cleaning the dataset, especially reducing categories and handling missing values. The construction of feature-rich features like host_tenure_days, has_security_deposit, and categorical encoding transformations was beneficial in optimizing model performance.

2) **What were the main challenges?**
   The hardest task was maintaining FPR below 10% on an unseen external test set and, at the same time, optimizing TPR. This required careful tuning of the threshold and performance simulation. It was particularly hard to estimate the effect of each feature on false positives in the absence of test labels.

3) **What would your group have done differently if you could start the project over again?**
   If we repeated it, we'd invest more time in examining feature importance and seeing which variables were most responsible for driving FPR. Tools like SHAP for model interpretability could then be used to convey model behavior and inform better feature selection.

4) **What would you do if you had another few months to work on the project?**
   - Attempt more advanced modeling approaches (e.g., LightGBM, stacking)
   - Plot learning curves to examine overfitting
   - Construct better tools to estimate external test performance

5) **What advice do you have for a group starting this project next year?**
   Recommendations for future teams:
   Be very careful with feature selection and engineering. Little mistakes can ruin your FPR.
   Always tune parameters that will give the best result and learn from each evaluation result on the external data to improve the model.