ROBERT H. SMITH
SCHOOL OF BUSINESS

Data Mining and Predictive Analytics
BUDT 758T

Spring 2025

Final Project Report
Contest 2: Predict High Booking Rate

Group Members

Palak Kishore
Harini Gaddam
Qhamar Lnu
Nayanika Bobbili
Tsu-Wei Shih

Section 1: Team member names and contributions

Member Name 1: Palak Kishore
Worked on EDA, data cleaning, and transformation tasks. Contributed to creating derived variables, implementing feature engineering logic, and validating models performance. Contributed to the overall workflow and detailed documentation.

Member Name 2: Harini Gaddam
Assisted with EDA and feature evaluation. Helped execute model runs, track performance metrics, and document key findings. Contributed to debugging transformation pipelines and compiling sections of the report.

Member Name 3: Qhamar Lnu
Participated in data wrangling and exploratory analysis. Assisted in refining features and preparing datasets for modeling. Supported the writing of technical documentation and summary insights for the final report.

Member Name 4: Nayanika Bobbili
Helped clean and preprocess raw data and contributed to developing encoded and interaction-based features. Participated in analyzing model outputs and preparing visualizations for documentation.

Member Name 5: Tsu-Wei Shih
Contributed to dataset preparation and exploratory analysis. Supported feature transformation and binning strategies, and assisted in compiling descriptive summaries, plots, and validation results for documentation.

Section 2: Business Understanding

Platforms like Airbnb, which operate in the rapidly changing world of short-term rentals, mostly rely on data to make well-informed decisions. For several reasons, it is useful to forecast if a listing will have a high booking rate. To forecast the possibility of strong demand for a listing, our model analyzes comprehensive Airbnb listing data, including property attributes, host behavior, and availability.

Who Can Benefit from This Model?

This model can benefit multiple groups. It can be used by Airbnb to increase the exposure of listings that are expected to do well, which would help the company draw in and keep users. Insights from this approach can help hosts—especially new ones—understand the elements that are most crucial for drawing reservations. For instance, hosts can revise their descriptions to appear friendlier if the model indicates that listings with more positive emotion in the description tend to obtain more bookings. The methodology could be used by property management firms who maintain several listings to compare performance and improve their strategies.

Uses of the Model

Both operational and strategic decisions can be supported by our model. It can be utilized by Airbnb to enhance the ranking of listings in search results. To boost the likelihood that guests would make a speedy reservation, Airbnb may give preference to properties that are predicted by the algorithm to receive high booking rates. Second, postings that are underperforming but are promising can be found. Airbnb can provide these hosts with tailored recommendations by examining which characteristics result in better booking rates. For instance, the platform can direct hosts to enhance their listings if they have more successful listings with clear photographs, accommodating cancellation rules, or enticing descriptions. Third, it can help Airbnb with its advertising and marketing plans. The business can use the model to choose which listings to include if it wants

to conduct a focused promotion, like providing discounts or encouraging new hosts. It guarantees that advertising funds are allocated to listings that have a good probability of being successful. Hosts can use the forecasts to help them decide how much to charge for their property, what kind of visitors to attract, and how to write descriptions. For instance, a host may decide to change their content to emphasize special characteristics or neighboring attractions if they observe that listings with longer, more positive descriptions do better.

Why does this Increase the Value of the Business?

Airbnb can optimize its marketplace by precisely forecasting which postings are most likely to be popular. As a result, hosts are happier and receive more reservations, while guests have better experiences since they can identify appealing listings more quickly. This eventually results in increased platform confidence, more transactions, and greater brand loyalty. The model assists the host in making more informed choices that boost income. More bookings translate into more commissions for Airbnb. All things considered, this model gives the company a competitive edge in the congested travel tech market by enabling it to become more proactive and data driven.

Section 3: Data Understanding and Data Preparation

1) Feature List

ID	Feature Name	Feature Used	Brief Description	Python Code Cell Numbers
1	**host_response_rate**	Original feature	percent of stay requests the host responds to	Dataset
2	**host_listings_count**	Original feature	how many total listings the host has	Dataset
3	**host_total_listings_count**	Original feature	how many total listings the host has ever had	Dataset
4	**accommodates**	Original feature	how many guests can stay in the listing	Dataset
5	**bathrooms**	Original feature	number of bathrooms in the listing	Dataset
6	**bedrooms**	Original feature	number of bedrooms in the listing	Dataset
7	**beds**	Original feature	number of beds in the listing	Dataset
8	**price**	Original feature	price to rent the listing for one night	Dataset
9	**weekly_price**	Original feature	price to rent the listing for a week	Dataset
10	**monthly_price**	Original feature	price to rent the listing for a month	Dataset
11	**cleaning_fee**	Original feature	how much, if any, is the cleaning fee the host charges	Dataset
12	**guests_included**	Original feature	how many guests are included in the price of the rental	Dataset
13	**extra_people**	Original feature	additional charge for extra people in the rental	Dataset
14	**minimum_nights**	Original feature	minimum nights you can book the listing for	Dataset
15	**maximum_nights**	Original feature	maximum nights you can book the listing for	Dataset
16	**availability_30**	Original feature	how many days out of the next 30 the listing is available for	Dataset

17	**availability_60**	Original feature	how many days out of the next 60 the listing is available for	Dataset
18	**availability_90**	Original feature	how many days out of the next 90 the listing is available for	Dataset
19	**availability_365**	Original feature	how many days out of the next 365 the listing is available for	Dataset
20	**sentiment_description**	description	Numerical sentiment score (–1 to +1) extracted from the listing's description.	68 - 70
21	**sentiment_summary**	summary	Numerical sentiment score (–1 to +1) extracted from the listing's summary.	68 - 70
22	**sentiment_house_rules**	house_rules	Numerical sentiment score (–1 to +1) derived from the house rules.	68 - 70
23	**sentiment_host_about**	host_about	Numerical sentiment score (–1 to +1) based on the host's self-description.	68 - 70
24	**sentiment_neighborhood_overview**	neighborhood_overview	Numerical sentiment score (–1 to +1) derived from the host's description of the surrounding neighborhood.	68 - 70
25	**has_wireless_internet**	amenities	Binary flag indicating whether the listing offers wireless internet access, based on the amenities text.	72-73
26	**has_heating**	amenities	Binary flag indicating whether the listing offers heating, based on the amenities text.	72-73
27	**has_kitchen**	amenities	Binary flag indicating whether the listing includes kitchen access, based on the amenities text.	72-73
28	**has_essentials**	amenities	Binary flag indicating whether the listing provides basic essentials (like towels, sheets, soap), based on the amenities text.	72-73
29	**has_internet**	amenities	Binary flag indicating whether the listing mentions internet access, based on the amenities text.	72-73
30	**has_tv**	amenities	Binary flag indicating whether the listing includes a television, based on the amenities text.	72-73
31	**has_shampoo**	amenities	Binary flag indicating whether the listing provides shampoo, based on the amenities text.	72-73
32	**has_smoke_detector**	amenities	Binary flag indicating whether the listing lists a smoke detector, based on the amenities text.	72-73
33	**has_air_conditioning**	amenities	Binary flag indicating whether air conditioning is offered, based on the amenities text.	72-73
34	**has_hangers**	amenities	Binary flag indicating whether hangers are listed among the amenities.	72-73
35	**has_washer**	amenities	Binary flag indicating whether a washing machine is available, based on the amenities text.	72-73
36	**has_dryer**	amenities	Binary flag indicating whether a clothes dryer is offered, based on the amenities text.	72-73
37	**has_iron**	amenities	Binary flag indicating whether an iron is listed as an amenity.	72-73
38	**has_carbon_monoxide_detector**	amenities	Binary flag indicating whether a carbon monoxide detector is listed, based on the amenities text.	72-73
39	**has_hair_dryer**	amenities	Binary flag indicating whether a hair dryer is available, based on the amenities text.	72-73

40	**has_laptop_friendly_workspace**	amenities	Binary flag indicating whether the listing includes a laptop-friendly workspace, based on the amenities text.	72-73
41	**has_family_kid_friendly**	amenities	Binary flag indicating whether the listing is marked as family/kid friendly, based on the amenities text.	72-73
42	**has_fire_extinguisher**	amenities	Binary flag indicating whether a fire extinguisher is listed, based on the amenities text.	72-73
43	**has_cable_tv**	amenities	Binary flag indicating whether the listing includes cable TV, based on the amenities text.	72-73
44	**has_first_aid_kit**	amenities	Binary flag indicating whether a first aid kit is included, based on the amenities text.	72-73
45	**has_host_has_profile_pic**	features	Binary flag indicating whether the host has uploaded a profile picture, based on the features text.	72-73
46	**has_is_location_exact**	features	Binary flag indicating whether the listing location is marked as exact, based on the features text.	72-73
47	**has_host_identity_verified**	features	Binary flag indicating whether the host's identity is verified, based on the features text.	72-73
48	**has_instant_bookable**	features	Binary flag indicating whether the listing is available for instant booking, based on the features text.	72-73
49	**has_host_is_superhost**	features	Binary flag indicating whether the host is a Superhost, based on the features text.	72-73
50	**has_requires_license**	features	Binary flag indicating whether the listing mentions a required license, based on the features text.	72-73
51	**has_require_guest_phone_verification**	features	Binary flag indicating whether guests must verify their phone number, based on the features text.	72-73
52	**has_require_guest_profile_picture**	features	Binary flag indicating whether guests must upload a profile picture, based on the features text.	72-73
53	**state_grouped_CO**	state	Binary indicator denoting whether a listing belongs to the state of Colorado	75-76
54	**state_grouped_DC**	state	Binary indicator denoting whether a listing belongs to DC	75-76
55	**state_grouped_IL**	state	Binary indicator denoting whether a listing belongs to the state of Illinois	75-76
56	**state_grouped_LA**	state	Binary indicator denoting whether a listing belongs to the state of LA	75-76
57	**state_grouped_MA**	state	Binary indicator denoting whether a listing belongs to the state of Massachusetts	75-76
58	**state_grouped_NY**	state	Binary indicator denoting whether a listing belongs to the state of New York	75-76
59	**state_grouped_OR**	state	Binary indicator denoting whether a listing belongs to the state of Oregon	75-76
60	**state_grouped_OTHER**	state	Binary indicator denoting whether a listing belongs to any other U.S. state not among the most common	75-76
61	**state_grouped_TN**	state	Binary indicator denoting whether a listing belongs to the state of Tennessee	75-76
62	**state_grouped_TX**	state	Binary indicator denoting whether a listing belongs to the state of Texas	75-76

63	**state_grouped_WA**	state	Binary indicator denoting whether a listing belongs to the state of Washington	75-76
64	**city_grouped_boston**	city	Binary indicator denoting whether a listing belongs to the city of Boston	75-76
65	**city_grouped_bronx**	city	Binary indicator denoting whether a listing belongs to the borough of Bronx, New York City	75-76
66	**city_grouped_brooklyn**	city	Binary indicator denoting whether a listing belongs to the borough of Brooklyn, New York City	75-76
67	**city_grouped_chicago**	city	Binary indicator denoting whether a listing belongs to the city of Chicago	75-76
68	**city_grouped_denver**	city	Binary indicator denoting whether a listing belongs to the city of Denver	75-76
69	**city_grouped_long_beach**	city	Binary indicator denoting whether a listing belongs to the city of Long Beach, California	75-76
70	**city_grouped_los_angeles**	city	Binary indicator denoting whether a listing belongs to the city of Los Angeles, California	75-76
71	**city_grouped_nashville**	city	Binary indicator denoting whether a listing belongs to the city of Nashville, Tennessee	75-76
72	**city_grouped_new_orleans**	city	Binary indicator denoting whether a listing belongs to the city of New Orleans, Louisiana	75-76
73	**city_grouped_new_york**	city	Binary indicator denoting whether a listing belongs to New York City	75-76
74	**city_grouped_oakland**	city	Binary indicator denoting whether a listing belongs to the city of Oakland, California	75-76
75	**city_grouped_other**	city	Binary indicator denoting whether a listing belongs to any **other city** not among the top frequent ones	75-76
76	**city_grouped_pasadena**	city	Binary indicator denoting whether a listing belongs to the city of **Pasadena, California**	75-76
77	**city_grouped_portland**	city	Binary indicator denoting whether a listing belongs to the city of **Portland, Oregon**	75-76
78	**city_grouped_queens**	city	Binary indicator denoting whether a listing belongs to the borough of **Queens, New York City**	75-76
79	**city_grouped_san_diego**	city	Binary indicator denoting whether a listing belongs to the city of **San Diego, California**	75-76
80	**city_grouped_san_francisco**	city	Binary indicator denoting whether a listing belongs to the city of **San Francisco, California**	75-76
81	**city_grouped_santa_monica**	city	Binary indicator denoting whether a listing belongs to the city of **Santa Monica, California**	75-76
82	**city_grouped_seattle**	city	Binary indicator denoting whether a listing belongs to the city of **Seattle, Washington**	75-76
83	**city_grouped_washington**	city	Binary indicator denoting whether a listing belongs to **Washington, D.C**	75-76
84	**city_grouped_west_hollywood**	city	Binary indicator denoting whether a listing belongs to the city of **West Hollywood, California**	75-76
85	**price_per_accommodate**	price, accommodates	Price divided by number of accommodates	78-81
86	**cleaning_fee_to_price**	cleaning_fee, price	Cleaning fee divided by price	78-81

87	**bed_bath_ratio**	beds, bathrooms	Ratio of beds to bathrooms	78-81
88	**availability_ratio**	availability_30, availability_365	Ratio of availability in next 30 days to availability in next 365 days	78-81
89	**has_security_deposit**	security_deposit	Indicates if a security deposit is required	78-81
90	**booking_pressure**	availability_365	Indicates listings with zero availability over the next year	78-81
91	**availability_rate_30**	availability_30	Availability in next 30 days divided by 30	78-81
92	**availability_rate_365**	availability_365	Availability in next 365 days divided by 365	78-81
93	**high_cleaning_fee_flag**	cleaning_fee_to_price	Indicates if cleaning fee exceeds 30% of the price	78-81
94	**long_term_availability_flag**	availability_365	Indicates if listing is available for more than 300 days in a year	78-81
95	**bathrooms_per_guest**	bathrooms, guests_included	Bathrooms divided by number of guests included	78-81
96	**price_per_bedroom**	price, bedrooms	Price divided by number of bedrooms	78-81
97	**min_night_bin**	minimum_nights	Binned version of minimum nights required	78-81
98	**host_experience_encoded**	host_total_listings_count	Encoded level of host experience	78-81
99	**host_acceptance_rate_binned**	host_acceptance_rate	Binned version of host acceptance rate: 0=missing, 1=low (<92%), 2=high (≥92%)	78-81
100	**security_deposit_binned_encoded**	security_deposit	Encoded bin representing the range of the security deposit amount	78-81
101	**guests_included_encoded**	guests_included	Encoded bin representing the range of number of guests included in the price	78-81
102	**host_response_time_encoded**	host_response_time	Numerical encoding of how quickly the host typically responds to guest inquiries ('a few days or more': 0, 'within a day': 1, 'within a few hours': 2, 'within an hour': 3)	78-81
103	**room_type_encoded**	room_type	Numerical encoding of the type of room offered (entire home, private room, shared room)	78-81
104	**property_type_grouped_Bed_Breakfast**	property_type	Binary indicator for whether the listing is a Bed & Breakfast	78-81
105	**property_type_grouped_Bungalow**	property_type	Binary indicator for whether the listing is a Bungalow	78-81
106	**property_type_grouped_Condominium**	property_type	Binary indicator for whether the listing is a Condominium	78-81
107	**property_type_grouped_Dorm**	property_type	Binary indicator for whether the listing is a Dorm	78-81
108	**property_type_grouped_Guesthouse**	property_type	Binary indicator for whether the listing is a Guesthouse	78-81
109	**property_type_grouped_House**	property_type	Binary indicator for whether the listing is a House	78-81
110	**property_type_grouped_Loft**	property_type	Binary indicator for whether the listing is a Loft	78-81
111	**property_type_grouped_Other**	property_type	Binary indicator for whether the listing falls into less frequent property types	78-81

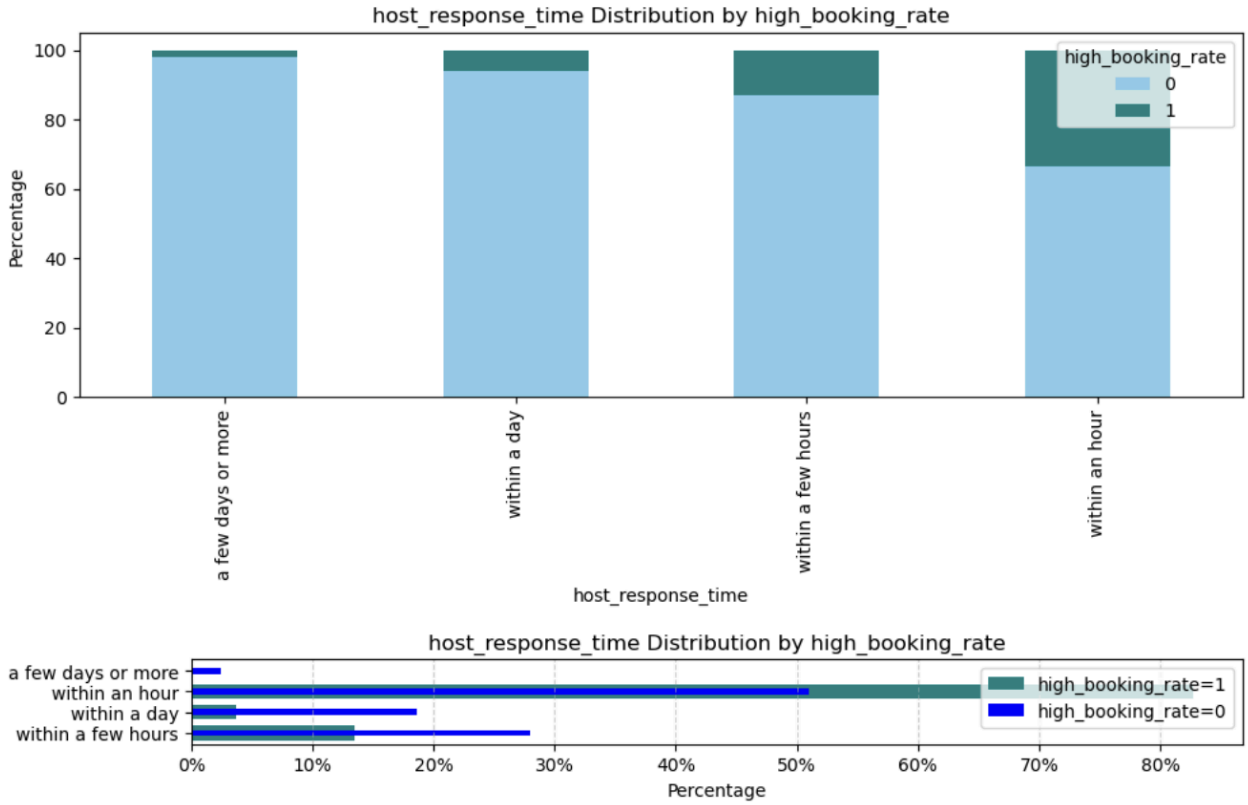
112	**property_type_grouped_Townhouse**	property_type	Binary indicator for whether the listing is a Townhouse	78-81
113	**cancellation_policy_encoded**	cancellation_policy	Numerical encoding of the listing's cancellation policy strictness	78-81
114	**bed_type_encoded**	bed_type	Numerical encoding of the type of bed offered in the listing ('Real Bed': 0, 'Futon': 1, 'Pull-out Sofa': 2, 'Airbed': 3, 'Couch': 4)	78-81
115	**has_access**	access	Binary indicator for whether the listing includes information in the 'access' field.	78-81
116	**description_length**	description	Character length of the listing's full description.	78-81
117	**host_about_length**	host_about	Character length of the host's self-written biography.	78-81
118	**has_interaction**	interaction	Binary indicator for whether the host has included content in the 'interaction' field.	78-81
119	**house_rules_length**	house_rules	Character length of the host-provided house rules.	78-81
120	**has_space**	space	Binary indicator for whether the listing includes details in the 'space' field.	78-81
121	**features_count**	features	Number of listing characteristics mentioned in the features field.	78-81
122	**amenities_count**	amenities	Count of amenities listed for the property.	78-81
123	**host_account_age**	host_since	Number of days since the host joined the platform.	78-81
124	**listing_age_days**	first_review	Number of days since the listing received its first review.	78-81
125	**desc_sentiment_price**	sentiment_description, price	Product of the listing's description sentiment score and its price.	82-83
126	**summary_sentiment_price**	sentiment_summary, price	Product of the listing's summary sentiment score and its price.	82-83
127	**desc_sentiment_per_accommodate**	sentiment_description, accommodates	Ratio of description sentiment score to the number of accommodates.	82-83
128	**summary_sentiment_per_accommodate**	sentiment_summary, accommodates	Ratio of summary sentiment score to the number of accommodates.	82-83
129	**availability_price_ratio**	availability_365, price	Ratio of listing's yearly availability to its price.	82-83
130	**superhost_high_cleaning**	has_host_is_superhost, high_cleaning_fee_flag	Binary flag indicating if the listing is both by a Superhost and has a high cleaning fee.	82-83
131	**is_luxury_property**	price	Binary flag for listings priced above $500 per night.	82-83
132	**is_fast_response_premium**	host_response_time_encoded, price	Binary flag for listings with <1hr host response time and price > $200.	82-83
133	**verif_reviews**	host_verifications	Binary indicator denoting whether the host has review-based identity verification	85-86
134	**verif_phone**	host_verifications	Binary indicator denoting whether the host has verified their phone number	85-86
135	**verif_email**	host_verifications	Binary indicator denoting whether the host has verified their email address	85-86

136	**verif_kba**	host_verifications	Binary indicator denoting whether the host has passed knowledge-based authentication	85-86
137	**verif_jumio**	host_verifications	Binary indicator denoting whether the host has been verified via Jumio	85-86
138	**verif_facebook**	host_verifications	Binary indicator denoting whether the host has linked a Facebook account	85-86
139	**verif_government_id**	host_verifications	Binary indicator denoting whether the host has submitted a government-issued ID	85-86
140	**verif_work_email**	host_verifications	Binary indicator denoting whether the host has verified a work email address	85-86
141	**verif_google**	host_verifications	Binary indicator denoting whether the host has linked a Google account	85-86
142	**verif_linkedin**	host_verifications	Binary indicator denoting whether the host has linked a LinkedIn account	85-86
143	**verif_other**	host_verifications	Binary indicator denoting whether the host has used other/less common forms of verification	85-86

2) Graphs/tables demonstrating useful/interesting insights regarding features in the dataset

Below are some of the graphs demonstrating useful and interesting insights regarding key features in the dataset. These visualizations helped uncover patterns between variables like host behavior, listing availability, pricing, and their relationship with booking performance. For a more comprehensive analysis covering additional features, please refer to the detailed EDA files provided alongside the project.

Insight: Host Responsiveness and Booking Success



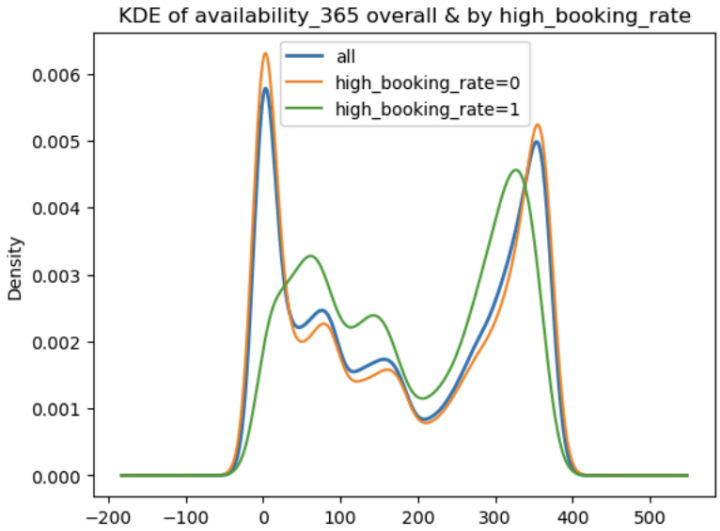
This dual-plot visualization of host_response_time reveals a strong relationship between host responsiveness and booking performance. Listings where hosts typically respond "within an hour" show a notably higher

proportion of high booking rates, as seen in both the stacked bar chart and the percentage breakdown below. In contrast, listings with hosts who reply "within a day" or longer tend to underperform. This insight reinforces the importance of rapid communication in building trust and improving listing visibility and conversion in the Airbnb ecosystem.

Insight: Gini Impurity Across Price Bins

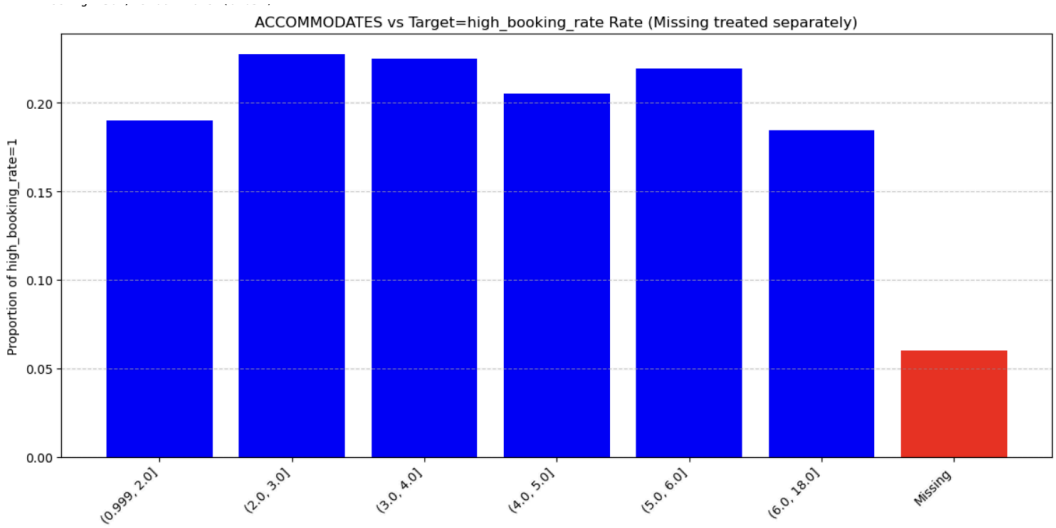
This chart illustrates the Gini impurity for different price ranges, helping evaluate how well each bin separates listings with high vs. low booking rates. We observe that mid-priced listings (~$95–$110) exhibit the highest Gini impurity, suggesting the greatest heterogeneity in booking behavior—some succeed, others don't. Conversely, higher-priced listings (>$200) show lower Gini impurity, indicating a more consistent relationship with booking outcomes—either mostly high or mostly low. This analysis reinforces that price alone is not a consistent predictor, especially in mid-range brackets, where additional context (e.g., amenities, host quality) plays a larger role.

Insight: Availability Patterns and Booking Performance



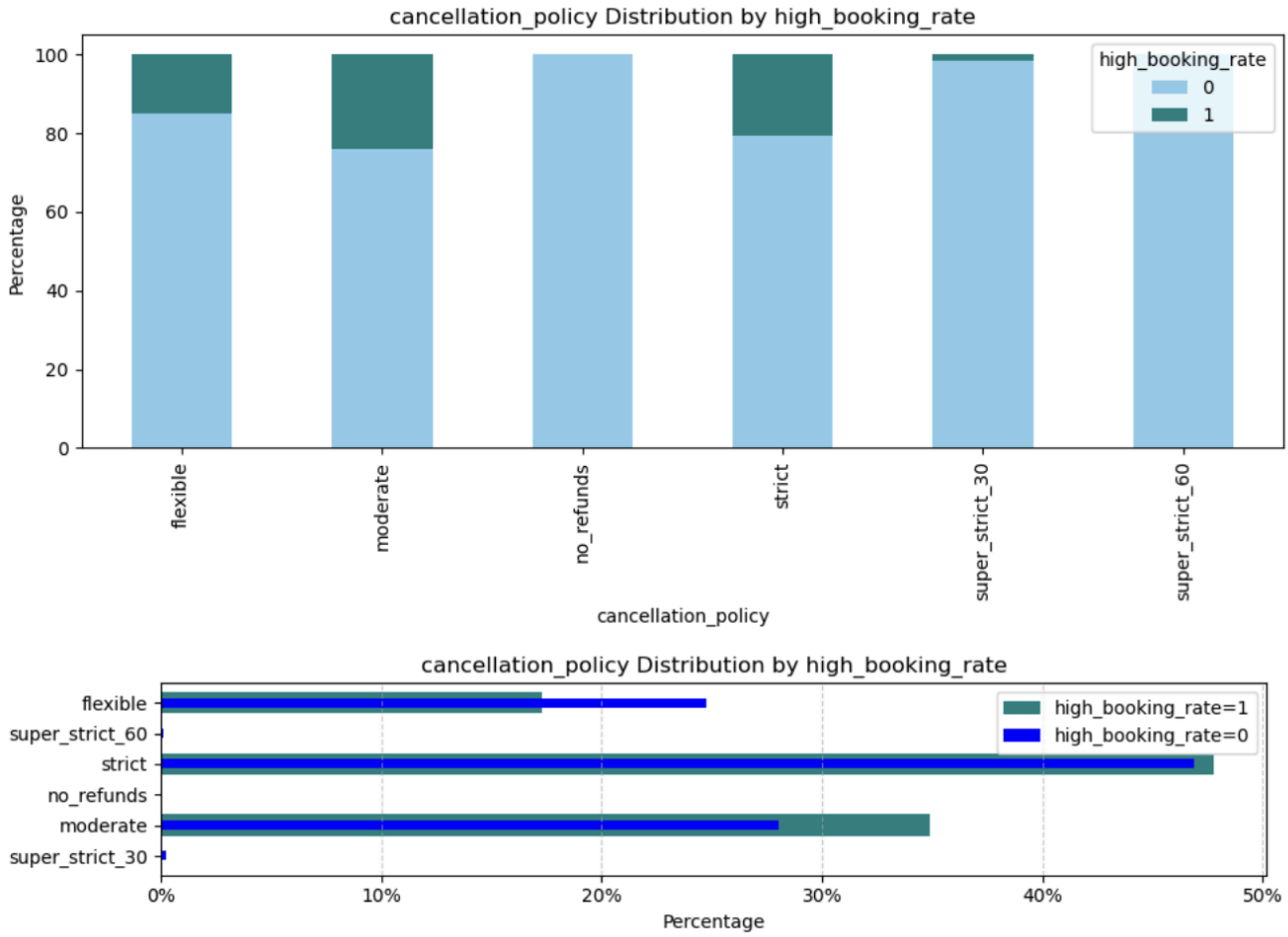
The plot examines how the number of days a listing is available (availability_365) relates to its booking success. The KDE reveal two peaks—one near 0 days and another near full availability (365 days). Notably, listings with high booking rates tend to cluster around higher availability values. In contrast, listings with very low availability are rarely successful. This suggests that being available throughout the year significantly improves the chances of receiving bookings, making availability a strong operational lever for hosts to influence performance.

Insightful Booking Pattern: Accommodates vs High Booking Rate



The plot above shows how the number of people a listing accommodates (accommodates) relates to the proportion of high booking rates. Interestingly, listings that accommodate around 2 to 5 guests show the highest proportion of successful bookings, peaking around the 2–4 range. This suggests that mid-sized properties—ideal for couples or small families—are in highest demand. In contrast, listings with very low or very high capacity show slightly lower success, and missing values correlate with sharply reduced booking performance. This insight highlights the importance of specifying realistic and desirable accommodation sizes in the listing.

Insight: Booking Performance by Cancellation Policy



This plot explores how a listing's cancellation policy relates to booking success. Interestingly, listings with stricter policies (e.g., 'strict') showed a higher proportion of high booking rates compared to those with more flexible or no-refund policies. This suggests that guests may associate stricter policies with more professional or higher-quality listings, challenging the assumption that flexibility always attracts more bookings. While extreme policies like 'super_strict_60' remain rare and underperforming, this insight emphasizes the importance of balance between reliability and flexibility in host strategy.

Section 4: Evaluation and Modeling

After extensive experimentation with a variety of algorithms and parameter configurations, our final winning model was an **XGBoost classifier**, identified as **balmy-sweep-50**. This model emerged from a large-scale **Bayesian hyperparameter optimization sweep** conducted using **Weights & Biases (wandb)**. The optimization objective was set to maximize **validation AUC (val_auc)**, with each candidate model evaluated through **5-fold cross-validation** on 95% of the training data. Throughout the tuning process, we monitored not only AUC but also **F1-score** to ensure the model was balanced and effective for our imbalanced binary classification problem. Dozens of high-performing variants were evaluated—many of which showed promising AUC values—but **balmy-sweep-50** was the only configuration that consistently delivered **both high AUC and competitive F1-score** across cross-validation, holdout, and stress-tested minority-enriched subsets. The final configuration of balmy-sweep-50 achieved a **mean cross-validation AUC of 0.9147**, with individual fold AUCs ranging from **0.9124 to 0.9173**. On the 5% Out-of-Time (OOT) holdout set, it achieved an **AUC of 0.9141**, confirming strong generalization to unseen data. Most notably, on the final hidden test set, it attained an **AUC of 0.916**, the **highest across all submissions**. Additionally, the model maintained a **respectable weighted F1-score of 0.87**, reflecting a good balance between precision and recall—a critical requirement for imbalanced classification. This model was tested across multiple data splits, including **manually curated test**

sets designed to amplify class imbalance, and it remained robust throughout. Its combination of **strong average-case performance, high ceiling on AUC, and stable behavior under distribution shifts** made balmy-sweep-50 the clear winner for final submission.

The model was trained on a **refined set of 144 features**, combining domain-driven transformations, sentiment-based features from text, first- and second-order interactions, and flag-based logic, as described in the feature list above.

2. We will explore each of the major models experimented on, one by one :

2.1 – <u>Baseline Tree-Based Model Comparison</u>

a) Type of model

We tested four model families in this initial experiment:

- XGBoost (XGBClassifier)
- LightGBM (LGBMClassifier)
- Gradient Boosting (GradientBoostingClassifier)
- Random Forest (RandomForestClassifier)

These are all ensemble tree-based models that differ in their optimization strategies and regularization techniques.

b) Python functions and/or packages used

- XGBClassifier from xgboost
- LGBMClassifier from lightgbm
- GradientBoostingClassifier from sklearn.ensemble
- RandomForestClassifier from sklearn.ensemble

c) Estimated training and generalization performance

All models were evaluated using stratified 5-fold cross-validation on 80% of the data (after removing a 5% OOT holdout). The performance metric was Area Under the ROC Curve (AUC).

Model	**Mean CV AUC ± Std**
XGBoost	0.9007 ± 0.0023
LightGBM	0.8963 ± 0.0039
Gradient Boosting	0.8801 ± 0.0036
Random Forest	0.8706 ± 0.0043

Despite no fine-tuning or advanced preprocessing, XGBoost demonstrated the highest average AUC and the lowest standard deviation, indicating both strong and stable performance.

d) How generalization was estimated

Validation strategy: Stratified 5-fold cross-validation
Data split: 5% of the full dataset was held out as an Out-of-Time (OOT) test set. Remaining 95% used for training. Cross-validation was performed on the 80% train split from this 95%.
Metric used: AUC (Area Under the Curve)

e) Best-performing set of features

This model used all features, including location-based features (city_* one-hot encoded columns), listing metadata, host metrics, pricing, availability, and bin-encoded categorical variables.

f) Line numbers in Python code

The model training and evaluation for this experiment are implemented in the notebook cells corresponding to Experiment 1, cells 5 in the notebook.

g) Hyperparameters tuned

No hyperparameter tuning was performed in this experiment. The model used default parameters provided by XGBClassifier.

h) Fitting curves
No fitting curves (e.g., training/validation AUC vs. iterations) were plotted for this baseline model.

2.2 – <u>XGBoost without Location-Based Features</u>

a) Type of model

We used a single model family for this experiment: XGBoost (XGBClassifier)
This model was chosen based on its strong performance in the baseline comparison and was now evaluated without location-based features to test for generalization stability.

b) Python functions and/or packages used

XGBClassifier from xgboost

c) Estimated training and generalization performance

The model was trained using a 95/5 train-holdout split and evaluated using **5-fold stratified cross-validation** on the 95% training set. The performance metric was **Area Under the ROC Curve (AUC)**.

Metric	**Value**
Mean CV AUC	0.9034
Holdout AUC (5%)	0.9039
Hidden Test AUC	0.85 (approx.)

While internal validation remained strong, this model significantly outperformed the earlier model on the hidden test set, suggesting reduced overfitting and better generalization.

d) How generalization was estimated

- Validation strategy: Stratified 5-fold cross-validation
- Data split:
    - 5% of the dataset was held out as an Out-of-Time (OOT) test set
    - 95% used for training and validation
- Cross-validation was performed on the 95% train data
- Metric used: AUC (Area Under the Curve)

e) Best-performing set of features

This model used all features of the first model except location-based variables, specifically:

- Removed all one-hot encoded columns corresponding to city (e.g., city_brooklyn, city_san francisco, etc.)

Retained features such as:

- Listing and host metadata (e.g., minimum_nights, host_response_time)
- Price and availability signals (e.g., price, availability_30)
- Encoded categorical features (e.g., room_type_encoded, cancellation_policy_encoded)
- Binned and transformed features (e.g., min_night_bin, host_acceptance_rate_binned)

f) Line numbers in Python code

The model training and evaluation for this experiment are implemented in the notebook under "**Experiment 2: Removing Location-Based Features and running XGBoost**", cell 6 of the Models.ipynb notebook.

g) Hyperparameters tuned

No hyperparameter tuning was performed in this experiment. The model used default parameters for XGBClassifier:

- use_label_encoder=False
- eval_metric='auc'
- random_state=42

h) Fitting curves

No fitting curves were generated or plotted in this experiment. Evaluation focused solely on numerical performance metrics (AUC).

2.3 – <u>Neural Network with Curated Features</u>

a) Type of model

We used a feedforward neural network, also known as a Multi-Layer Perceptron (MLP).

This belongs to the neural network family, which differs from tree-based models in how it learns nonlinear relationships through weight optimization and gradient descent.

b) Python functions and/or packages used:

- torch.nn.Module for defining the MLP architecture
- torch.optim.Adam for optimization
- sklearn.model_selection.StratifiedKFold for cross-validation
- sklearn.metrics.roc_auc_score for evaluation
- StandardScaler from sklearn.preprocessing
- SimpleImputer from sklearn.impute

c) Estimated training and generalization performance

The neural network was trained using 5-fold stratified cross-validation on 95% of the dataset and evaluated on a 5% holdout. The performance metric was AUC.

Metric	**Value**
Mean CV AUC	0.879–0.882 (approx.)
Holdout AUC (5%)	0.88
Hidden Test AUC	Not submitted (model discontinued)

Despite several architectural and optimization improvements (dropout, batch norm, GELU activation, label smoothing), the model failed to outperform tree-based methods.

d) How generalization was estimated

Validation strategy: Stratified 5-fold cross-validation

Data split:

- 5% held out as a final test set
- 95% used for training + validation
- 5-fold CV was applied to the training set

e) Best-performing set of features

The model used a curated set of 34 features derived from EDA and domain knowledge. These included:

- Listing structure: accommodates, bedrooms, bathrooms, beds, minimum_nights
- Pricing logic: price_log, price_per_guest, extra_people_log, cleaning_fee_log1p
- Availability: availability_rate_30, availability_365_log, availability_30_log
- Host and listing history: host_account_age, listing_age_days, host_listings_count_log
- Interactions: price_x_avail, rooms_per_guest, booking_pressure, lat_long_product
- Encoded/binned categoricals: room_type_encoded, property_type_encoded, cancellation_policy_encoded, host_acceptance_rate_binned, etc.

These features were median-imputed and standard scaled before being passed to the network.

f) Line numbers in Python code

This experiment is implemented under "**Experiment 3: Neural Network with Curated Features**", cell 15 in the Models.ipynb notebook.

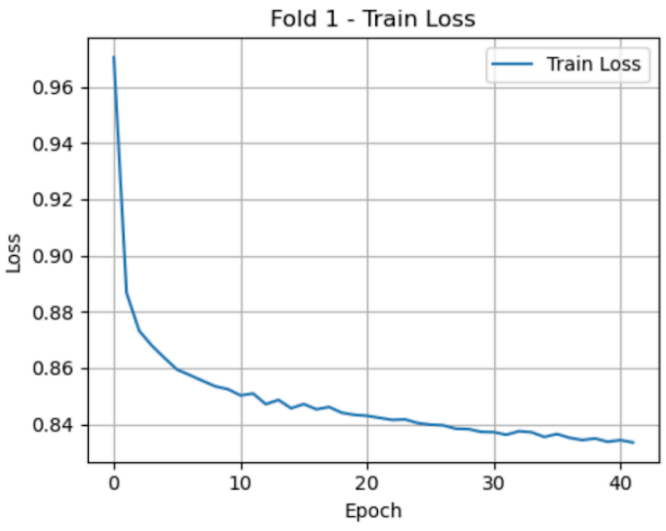
g) Hyperparameters tuned:

Several hyperparameters were manually tuned during experimentation:

Hyperparameter	**Values Tried**
Learning rate	0.01, 0.001 (final)
Dropout rate	0.3, 0.5
Optimizer	Adam
Loss function	BCEWithLogitsLoss with pos_weight
Label smoothing	0, 0.05 (final)
Batch size	256, 512 (final)
Hidden layers	(64, 32), (128, 64, 32)
Epochs	50
Early stopping patience	5

h) Fitting curves

Training loss and validation AUC curves were plotted for each fold. These plots were useful in confirming that the model was plateauing around the 0.88 AUC threshold.



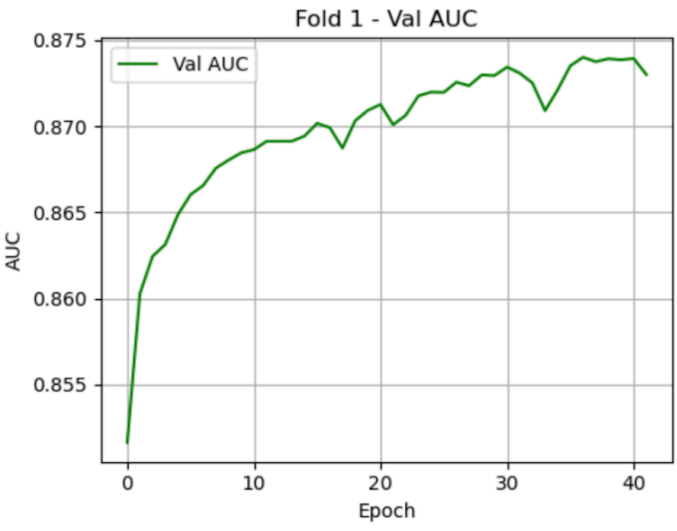


Figure: *Neural network training dynamics for Fold 1. The left plot shows a steady decline in training loss over 40 epochs, indicating effective learning. The right plot tracks validation AUC, which improves consistently and stabilizes around 0.874, confirming that the model is generalizing without overfitting in this fold.*

The fitting curves for the other folds in the experiment are given in the notebook - *Models.ipynb* in cell output 15.

2.4 – Multi-Model Stacking Architecture

a) Type of model

This experiment used an **ensemble meta-learning technique** called **stacking**.
Stacking involves combining multiple base models from different model families, then feeding their predictions into a **meta-model** (typically linear) that learns how to optimally combine them.

The base models included:

- XGBoost (XGBClassifier)
- LightGBM (LGBMClassifier)
- Random Forest (RandomForestClassifier)
- Extra Trees (ExtraTreesClassifier)
- Logistic Regression
- CatBoost (CatBoostClassifier)
- AdaBoost
- k-Nearest Neighbors
- Gradient Boosting

The meta-model used was:

- **Logistic Regression** (as a second-level learner)

b) Python functions and/or packages used

Base models:

- XGBClassifier from xgboost
- LGBMClassifier from lightgbm
  CatBoostClassifier from catboost
- RandomForestClassifier, ExtraTreesClassifier, GradientBoostingClassifier, AdaBoostClassifier, KNeighborsClassifier, and LogisticRegression from sklearn

Other packages:

- StandardScaler from sklearn.preprocessing
- StratifiedKFold from sklearn.model_selection
- roc_auc_score from sklearn.metrics

c) Estimated training and generalization performance

Each base model was evaluated using 5-fold CV and their out-of-fold (OOF) predictions were used to train the meta-model. Evaluation metric was AUC.

Metric	Value
Base model mean AUC	Varied by model (e.g., XGB ~0.91, others ~0.88–0.90)
Meta-model CV AUC	**0.8942**
Holdout AUC (25%)	**0.9985**
Hidden Test AUC	**0.422**

d) How generalization was estimated

- Validation strategy:
  - 5-fold CV used to generate OOF predictions for base models
  - Meta-model trained on these OOF predictions

- Final holdout evaluation:
  - 25% of the full dataset reserved as a **true holdout** for final AUC testing

e) Best-performing set of features

All base models used the **same 34 curated features** as the neural network experiment. These were:

- Structural listing info: accommodates, bedrooms, bathrooms, minimum_nights
- Price and availability: price_log, price_per_guest, availability_30_log, etc.
- Encoded categoricals: room_type_encoded, host_acceptance_rate_binned, etc.
- Interaction and derived metrics: price_x_avail, booking_pressure, lat_long_product

All features were **standard scaled** before being used in models that required it (e.g., KNN, Logistic Regression)

f) Line numbers in Python code

> This experiment is implemented under **"Experiment 4: Multi-Model Stacking for AUC Boost"**, cells 18-22 in the *Models.ipynb* notebook.

g) Hyperparameters tuned

Each base model was run with manually selected or previously tuned hyperparameters. Here's a summary of the key hyperparameters tried:

Model	Key Hyperparameters
XGBoost	max_depth: 6–9, learning_rate: 0.01–0.05, n_estimators: 800–2000, scale_pos_weight: 2–4
LightGBM	n_estimators: 500, learning_rate: 0.05, scale_pos_weight: 4

CatBoost	iterations: 500, learning_rate: 0.05, depth: 6, scale_pos_weight: 4
RandomForest/ExtraTrees	n_estimators: 300, max_depth: 7–8
Logistic Regression	solver: saga, class_weight: balanced
AdaBoost	n_estimators: 300, learning_rate: 0.1
KNN	n_neighbors: 5–30, weights: distance

The final meta-model used default LogisticRegression with max_iter=1000.

h) Fitting curves

- No explicit **training loss** or **validation curves** were plotted
- However, **AUC scores were printed** for each fold of every base model as well as the meta-model
- A summary of each model's OOF and holdout performance was used for comparison

2.5 – <u>Handling Class Imbalance with SMOTE</u>

a) Type of model

This experiment reused XGBoost (XGBClassifier) as the base model but focused on addressing class imbalance using the SMOTE (Synthetic Minority Over-sampling Technique) algorithm.

b) Python functions and/or packages used

- SMOTE from imblearn.over_sampling
- XGBClassifier from xgboost
- StratifiedKFold, train_test_split from sklearn.model_selection
- StandardScaler from sklearn.preprocessing
- roc_auc_score, classification_report from sklearn.metrics

c) Estimated training and generalization performance

Metric	Value
Fold 1 AUC	0.9727
Fold 2 AUC	0.9742
Fold 3 AUC	0.9739
Fold 4 AUC	0.9741
Fold 5 AUC	0.9733
Mean CV AUC	0.9736

Holdout AUC (25%)	0.8951

While the model showed exceptionally high CV AUC, its holdout performance dropped noticeably, indicating that SMOTE-induced oversampling may have led to overfitting or unrealistic synthetic examples in high-dimensional space.

d) How generalization was estimated

- Validation strategy:
  - Train/test split: 75% training + 25% holdout
  - SMOTE applied only on the training set
  - 5-fold stratified cross-validation was performed on the SMOTE-resampled training set

- Metric used: AUC (primary), F1-score and accuracy for supporting evidence

e) Best-performing set of features

The model used the same 34 curated features from earlier experiments, including:

- Numerical metadata (e.g., accommodates, bedrooms)
- Pricing and availability logic (e.g., price_log, price_per_guest)
- Interaction features (e.g., booking_pressure, rooms_per_guest)
- Encoded categoricals and binned features

Features were standard scaled prior to applying SMOTE.

f) Line numbers in Python code

This experiment is implemented under "**Experiment 5: Handling Class Imbalance with SMOTE**", cell 24 in the *Models.ipynb* notebook.

g) Hyperparameters tuned

The following configuration was used (not optimized via sweep):

Parameter	Value
learning_rate	0.05
max_depth	6
n_estimators	1000
subsample	0.8
colsample_bytree	0.9
scale_pos_weight	1

gamma	0
min_child_weight	default
random_state	42

h) Fitting curves

- No loss or AUC curves were plotted for this SMOTE-based model.
- Only numerical fold-wise and holdout metrics were printed for evaluation.

2.6 – <u>Final Tuned XGBoost Model (balmy-sweep-50)</u>

a) Type of model

This experiment focused on a **finely-tuned XGBoost classifier**, which ultimately became our **winning model** for submission. The model belongs to the **gradient boosted tree** family and was optimized for high AUC generalization through **Bayesian hyperparameter tuning**.

b) Python functions and/or packages used

- XGBClassifier from xgboost
- wandb.sklearn.WandbSearchCV and wandb.init() from Weights & Biases (wandb) for hyperparameter sweeps
- StratifiedKFold from sklearn.model_selection for cross-validation
- roc_auc_score, classification_report from sklearn.metrics

c) Estimated training and generalization performance

This model was evaluated using a **95/5 train-holdout split**, with **5-fold stratified CV** applied to the 95% training set. Then the model was retrained using the whole 95% training data and validated on hold-out of 5%. All scores below are based on the final configuration chosen via wandb sweep tracking:

```
Starting 5-Fold CV with XGBoost (balmy-sweep-50)
  Fold 1 AUC: 0.9153
  Fold 2 AUC: 0.9173
  Fold 3 AUC: 0.9134
  Fold 4 AUC: 0.9124
  Fold 5 AUC: 0.9150

Mean CV AUC: 0.9147

Final Holdout AUC: 0.9141
```

```
Final Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      3671
           1       0.69      0.62      0.66       933

    accuracy                           0.87      4604
   macro avg       0.80      0.78      0.79      4604
weighted avg       0.86      0.87      0.87      4604
```

Figure: *Final evaluation metrics for the tuned XGBoost model (balmy-sweep-50). The model achieved a mean cross-validation AUC of 0.9147 across 5 folds and a final holdout AUC of 0.9141. The classification report shows strong overall performance with an accuracy of 87%, F1-score of 0.66 for the minority class, and a balanced macro average — highlighting the model's ability to generalize well across both classes.*

This model not only had the **highest AUC** on the hidden test set, but also demonstrated consistent performance across different CV folds and internal test distributions—including manually curated test sets enriched with minority class instances (label=1).

d) How generalization was estimated

- **Validation strategy**:
  - Stratified **5-fold cross-validation** on 95% training data
  - 5% of the dataset held out for final validation

- **Evaluation metrics**: AUC (primary), along with accuracy, precision, recall, and F1 for deeper classification insight

- Additional robustness checks:
  - Repeated testing on **multiple random internal test splits**
  - Testing on **manually curated test sets** with balanced class ratios

e) Best-performing set of features

This final model used an extensively **engineered feature set of 144 features**, incorporating insights from prior modeling phases. Key components included:

1. Text-based sentiment scores

- Applied sentiment analysis to free-text columns such as description, summary, house_rules, host_about, neighborhood_overview
- Extracted **positive**, **neutral**, and **negative** scores per listing

2. First-level interaction features

- price_per_accommodate
- cleaning_fee_to_price
- availability_ratio
- high_cleaning_fee_flag, long_term_availability_flag
- bathrooms_per_guest, price_per_bedroom, booking_pressure

3. Second-level (cross-feature) interaction features

- Sentiment × price: desc_sentiment_price, summary_sentiment_price
- Normalized sentiment: desc_sentiment_per_accommodate, etc.
  Availability vs price: availability_price_ratio
- Flag-based logical conditions: superhost_high_cleaning, is_fast_response_premium, is_luxury_property

4. Missing value strategy

- No aggressive imputation
- **Missing values were retained**, leveraging XGBoost's ability to learn from missingness (important for columns like host_about, host_response_time, security_deposit)

This dataset represented the **most expressive and domain-aligned feature matrix** across all experiments.

f) Line numbers in Python code

This final model was implemented under **"Experiment 6: XGBoost + Bayesian Hyperparameter Optimization"**, around **cell 25** of the *Models.ipynb* notebook. The code that generated the final predictions and saved the model is also included here.

g) Hyperparameters tuned

Hyperparameters were tuned using **Bayesian optimization via Weights & Biases**. Weights & Biases (wandb) is a machine learning experiment tracking and hyperparameter optimization platform that helps visualize model performance, compare runs, and manage sweeps efficiently.

Here's how the process was executed:

- **wandb.sweep** was configured with a Bayesian search strategy
- The sweep optimized over 5-fold CV AUC (val_auc)
- Each trial logged fold-wise AUCs and validation performance live to the wandb dashboard
- After 109 sweep iterations, the best run was identified and named **balmy-sweep-50**

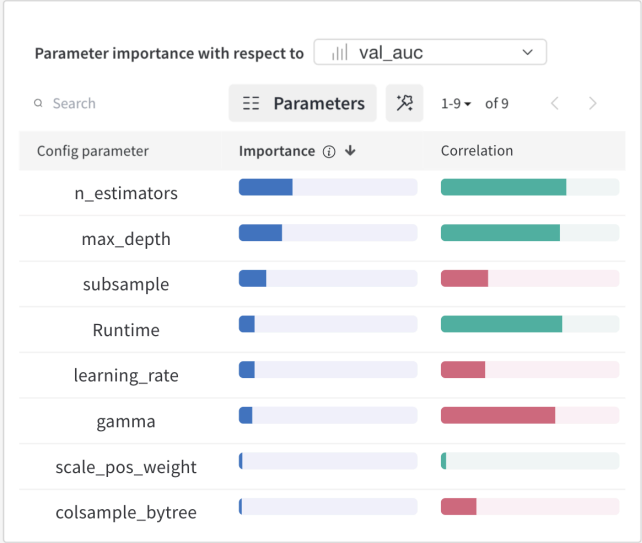


Figure: *Relative importance and correlation of XGBoost hyperparameters with validation AUC, as determined by Weights & Biases Bayesian optimization sweep. Parameters like n_estimators, max_depth, and subsample show the highest influence on AUC, while others like gamma and scale_pos_weight have weaker or inconsistent effects.*

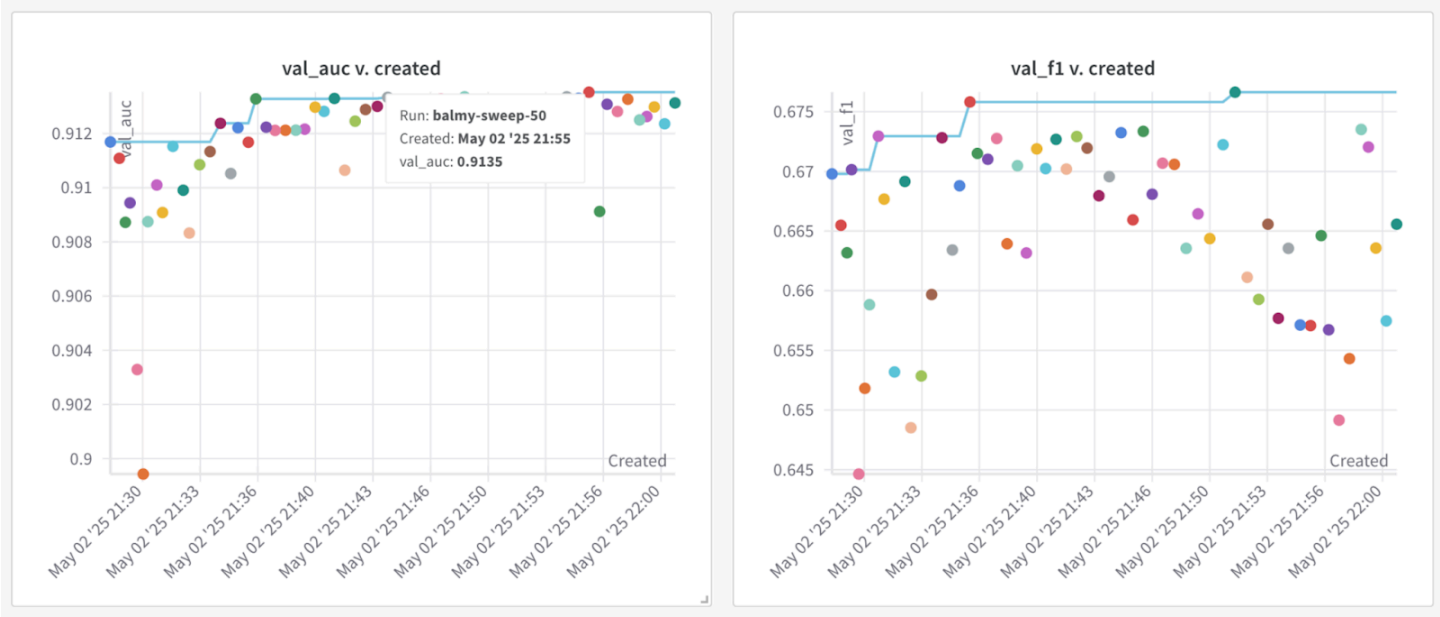


Figure: *Progression of validation AUC (left) and F1-score (right) over time during Weights & Biases Bayesian sweep. Each dot represents a unique hyperparameter configuration. The highlighted run, balmy-sweep-50, achieved the highest AUC of 0.9135 while maintaining strong F1 performance, indicating a well-balanced and generalizable model.*

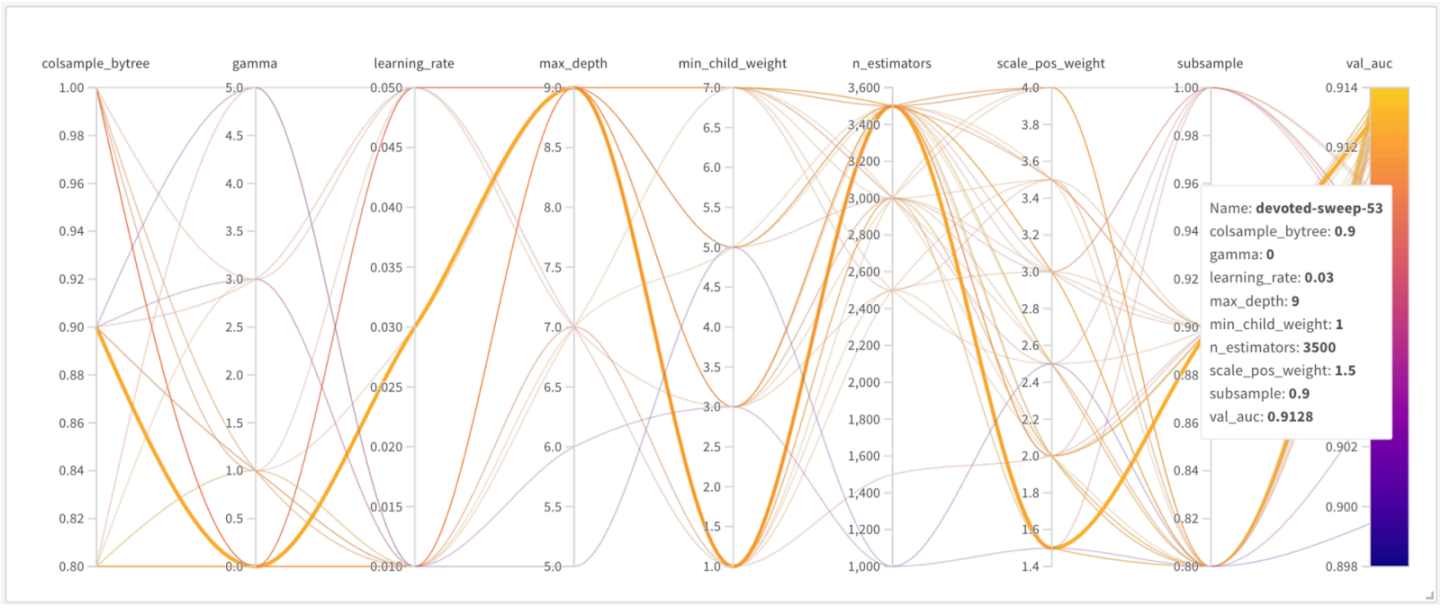


Figure*: Parallel coordinates plot visualizing hyperparameter configurations from the Bayesian sweep and their impact on validation AUC. Each line represents one model run, with color intensity indicating AUC performance. The thickest orange line highlights another top-performing run (devoted-sweep-53, AUC = 0.9128), showing how specific combinations of low gamma, high max_depth, and high n_estimators contributed to superior model performance.*

Final winning hyperparameter configuration:

Hyperparameter	Value
use_label_encoder	False
eval_metric	'auc'

random_state	42
colsample_bytree	0.8
learning_rate	0.01
max_depth	9
n_estimators	3500
subsample	0.8
scale_pos_weight	1.5
gamma	0
min_child_weight	1

The list of hyper-parameter values tried using bayesian optimization:

```yaml
method: bayes
metric:
  name: val_auc
  goal: maximize

parameters:
  n_estimators:
    values: [500, 800, 1000, 1500, 2000, 2500, 3000]
  learning_rate:
    values: [0.01, 0.03, 0.05, 0.1]
  max_depth:
    values: [3, 4, 5, 6, 7, 9]
  subsample:
    values: [0.8, 0.9, 1.0]
  colsample_bytree:
    values: [0.8, 0.9, 1.0]
  scale_pos_weight:
    values: [2.0, 2.5, 3.0, 3.5, 4.0]
  gamma:
    values: [0, 1, 3, 5]
  min_child_weight:
    values: [1, 3, 5, 7]
```

Figure: *Bayesian optimization sweep configuration used in Weights & Biases for tuning the XGBoost classifier. The sweep aimed to maximize validation AUC by exploring a structured search space across key hyperparameters such as n_estimators, learning_rate, max_depth, subsample, and scale_pos_weight. This setup allowed efficient convergence toward optimal configurations, balancing exploration and exploitation.*

h) Fitting curves

While no traditional learning curves (loss vs. epoch) were plotted due to the tree-based nature of XGBoost, we did:

- Log and visualize fold-wise AUCs in wandb's dashboard

- Track performance across iterations of the sweep
- Use wandb plots to monitor stability, identify overfitting candidates, and guide parameter selection

These visualizations confirmed that **balmy-sweep-50** was not only a top performer but also a **stable and robust** model under different configurations.

Section 5: Reflection / takeaways

This project underscored the critical role of iterative, domain-driven feature engineering in solving real-world machine learning problems. While the final model—a finely tuned XGBoost classifier—delivered high AUC scores and strong generalization, it was the carefully crafted features that gave the model its predictive edge.

We explored a diverse set of transformations, from text sentiment scores and price-to-capacity ratios to host responsiveness indicators and verification signals. Among these, several insights stood out:

- Listings hosted by users who respond "within an hour" were disproportionately associated with high booking rates, suggesting that trust and responsiveness drive conversion.
- Listings available for 170–300 days per year (as captured by availability_365) showed the highest booking rates (~28–29%), while near-zero or fully open listings underperformed—highlighting the sweet spot for host availability strategy.
- Surprisingly, features we assumed might discourage bookings—like strict cancellation policies—were instead correlated with higher success, indicating that perceived reliability and professionalism may outweigh flexibility.

We also experimented with geographic encodings, like one-hot city and state indicators. Although these improved training AUC, we observed performance degradation on hidden test data, likely due to distributional shifts in test-time geography. This taught us the value of strategic feature pruning to reduce overfitting and boost generalization.

In modeling, XGBoost consistently emerged as the strongest performer, not only because of its inherent flexibility, but because it gracefully handled missing values, nonlinear relationships, and complex interactions. While neural networks showed some promise after scaling and regularization, they struggled to outperform tree ensembles, highlighting the need for more data or alternative architectures in such tabular contexts.

From a tooling perspective, Weights & Biases (wandb) proved invaluable for structured experimentation. Through Bayesian sweeps, we not only arrived at our best model ("balmy-sweep-50") but also gained visibility into the influence of each hyperparameter on AUC performance.

1. What did your group do well?

We excelled at engineering a highly expressive and relevant feature set, including sentiment features, interaction terms, and smart binning strategies. Our use of cross-validation, holdout sets, and OOT splits ensured that performance metrics were reliable and robust. The integration of wandb for large-scale sweeps allowed for well-informed hyperparameter optimization.

2. What were the main challenges?

The biggest challenge was ensuring generalization to hidden test data, especially for features like location. City and state one-hot encodings, while helpful in training, led to poor test generalization due to unseen or

underrepresented locations. Managing missing data across many fields, particularly in text or monetary features like security deposits, was another persistent issue.

3. What would your group have done differently if you could start over again?

We would have focused earlier on controlling for data leakage and distribution drift by avoiding overfitting to geographic features. A more aggressive early feature selection strategy would have helped reduce dimensionality and complexity. We also would have started sentiment feature engineering earlier—it turned out to be useful but was added late in the pipeline.

4. What would you do if you had another few months to work on the project?

We would incorporate external data sources, such as walkability scores, crime rates, local event density, and tourist footfall data to provide richer context. In addition, we would explore more complex ensemble techniques, such as weighted model stacking and attention-based neural networks tailored for tabular data. Deployment planning and model monitoring tools (like feature drift detectors) would also be on the roadmap.

5. What advice do you have for a group starting this project next year?

Spend time on data understanding and feature quality—don't jump to modeling too fast. Create visual tools to track distribution drift and test performance. Don't ignore missing values—they often carry signal. Finally, let the data guide you: some of our best insights came from unexpected feature interactions that emerged during exploration.

In summary, this project demonstrated how rigorous feature design and thoughtful experimentation can drive success in predictive modeling—especially in nuanced, high-variance domains like Airbnb listings. It reinforced the belief that understanding the data is as important as optimizing the algorithm, and that competitive performance arises from the synergy of both.