UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

BUDT 758T Final Project Report Spring 2024 GROUP - 15

Section 1: Team member names and contributions

1. **Anand Vishnu Kaipanchery:** Anand explored data thoroughly, using techniques like data understanding and graphical developments. He also integrated supplementary data to enrich our analysis. Anand's report section provided detailed interpretations of the graphs and the additional data's impact.

2. **Aradhya Singh Bisht:** Aradhya led data preparation, meticulously cleaning and organizing the dataset. He helped in model development, employing techniques like Random Forest and Ridge model. Aradhya's attention to detail ensured accurate models, especially regarding numeric and categorical variables. In the report, she explained these processes and presented key findings, enriching understanding of our project.

3. **Aryan Asawa:** Aryan took charge of data preparation, demonstrating his adeptness in managing the process effectively. Focusing on text columns he performed sentiment analysis and text mining, to delve into understanding emotions from text, and ensuring they accurately captured the nuances of sentiment analysis. Beyond data preparation, further he developed Boosting and Logistic Regression models. Aryan played a crucial role in fine-tuning the models for optimal performance. His collaboration with team members also involved integrating advanced features into the models, enhancing their predictive capabilities further, and developing the report further.

4. **Siddhant Soymon:** Siddhant worked on understanding the data really well. He made graphs and charts to show how things changed over time or with different factors. Siddhant also helped write the report, explaining what the graphs meant and what we learned from them.

5. **Simran Gallani:** Simran played a role in data preparation, where she meticulously handled both numeric and categorical variables. Employing techniques like Bagging model and Lasso regression, Simran adeptly engaged in feature engineering and model building, resulting in robust predictive models.Her work throughout the data

preparation phase was instrumental in the project's success. In the report, Simran elaborated on these processes and their outcomes, providing comprehensive insights into our methodologies and results.

Section 2: Business Understanding

Airbnb is a platform where people can rent out their homes, rooms, or apartments to guests. Our model helps predict if a listing will get a lot of bookings or not. Using data from over 50 different features, our goal is to help Airbnb hosts and other businesses understand what makes a listing popular. Here's how this model could be useful for different people and what actions they might take with the insights it provides.

Who Can Use This Model?

- **Airbnb Hosts**: People who rent out their spaces on Airbnb can use this model to find out what makes a listing more likely to get booked. This can help them improve their listings to attract more guests.
- **Property Managers**: Companies that manage many Airbnb properties can use the model to understand which listings are likely to be most successful, allowing them to focus on those properties.
- **Airbnb Corporate**: The Airbnb company can use the model to guide hosts on how to improve their listings and to make the platform better for guests.
- **Real Estate Investors**: Investors who want to buy properties to rent out on Airbnb can use the model to decide which types of properties are likely to be most profitable.

What Business Actions Can Be Taken?

- **Improving Listings:** Hosts can use the model's results to find out what makes a listing more attractive. For example, if quick responses to guest questions lead to more bookings, hosts can aim to be faster in replying.
- **Enhancing the Guest Experience**: The model might show that certain amenities or services lead to better reviews. Hosts can use this information to improve their listings and get more bookings.
- **Pricing Strategies**: If the model indicates that a specific price range leads to more bookings, hosts can adjust their prices accordingly to get more business.
- **Marketing:** With the information from the model, Airbnb hosts can focus on advertising their most attractive features, making their listings more appealing to potential guests.
- **Investing in the Right Properties**: The model can help investors decide which locations or types of properties are most likely to attract guests and generate revenue.

How Does This Model Add Value?

- **Increased Revenue for Hosts**: By making their listings more appealing, hosts can increase their booking rates and earn more money.

- **Improved Experience on Airbnb**: If hosts use the model to improve their listings, guests will have a better experience, leading to more bookings for Airbnb as a whole.
- **Lower Costs for Hosts:** Hosts can use the model to avoid spending money on things that don't increase bookings.
- **Guided Business Growth for Airbnb:** With insights from the model, Airbnb can focus on areas that will make the platform more successful and grow its business.

Overall, this model helps people who use Airbnb to make better decisions about their listings, improve their services, and ultimately earn more money. It can also guide Airbnb in its business strategy to offer a better experience to both hosts and guests.

Section 3: Data Understanding and Data Preparation

This section should include:

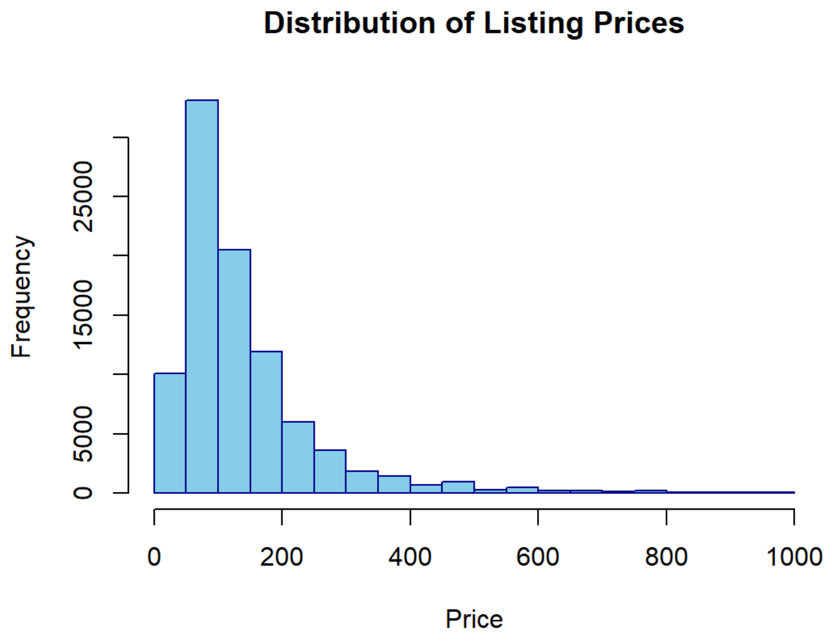
1) A table of the following format (for example):

ID	Feature Name	Brief Description	R Code Line Numbers (Train Data)	R Code Line Numbers (Test Data)
1	summary	Original feature from dataset	285-319	671-690
2	house_rules	Original feature from dataset	321-350	640-667
3	host_since	Original feature from dataset	38-39	397-398
4	host_response_rate	Original feature from dataset	40	399
5	host_response_time	Original feature from dataset	41	400
6	host_acceptanced	Original feature from dataset	44	403
7	host_neighbourhood	Original feature from dataset	45	404
8	host_listings_count	Original feature from dataset	173-177	533-537
9	host_total_listings_count	Original feature from dataset	46	405
10	host_verifications	Original feature from dataset	221-233	553-567
11	neighborhood	Original feature from dataset	47	406
12	city	Original feature from dataset	48	407
13	state	Original feature from dataset	49	408
14	zipcode	Original feature from dataset	363-383	700-720
15	country_code	Original feature from dataset	50	409
16	property_type	Original feature from dataset	52	411
17	room_type	Original feature from dataset	53	412
18	bed_type	Original feature from dataset	54	413
19	amenities	Original feature from dataset	264-271	599-606
20	cleaning_fee	Original feature from dataset	55	414

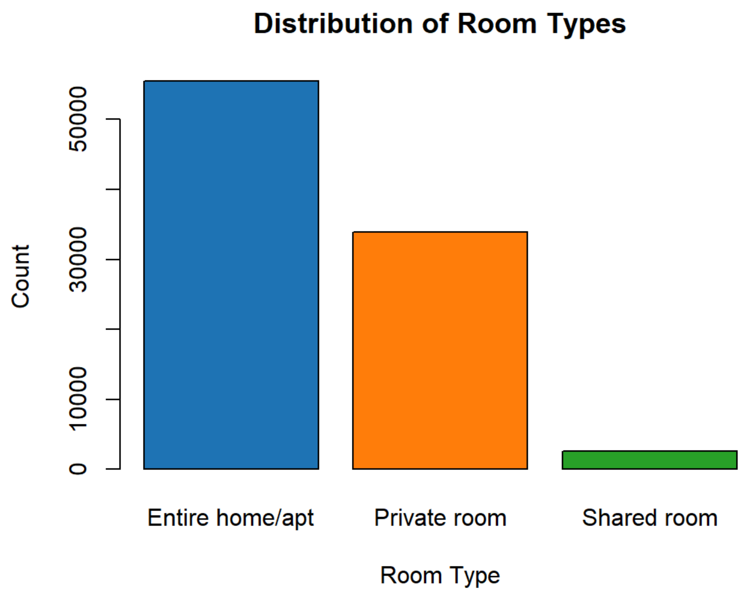
21	security_deposit	Original feature from dataset	56	415
22	guests_included	Original feature from dataset	57	416
23	extra_people	Original feature from dataset	58	417
24	minimum_nights	Original feature from dataset	59	418
25	maximum_nights	Original feature from dataset	60	419
26	availability	Calculated field to check the popularity of a property	61	420
27	first_review	Original feature from dataset	62	421
28	license	Original feature from dataset	63	422
29	jurisdiction_name	Original feature from dataset	249-260	587-598
30	cancellation_policy	Original feature from dataset	64	423
31	market	Original feature from dataset	68-80	428-440
32	smart_location	Original feature from dataset	82-98	442-458
33	accommodates	Original feature from dataset	102	462
34	bathrooms	Original feature from dataset	103	463
35	bedrooms	Original feature from dataset	104	464
36	beds	Original feature from dataset	105	465
37	bed_bath_ratio	Calculated field that measures the ratio of number of beds to the number of bathrooms	109-111	469-471
38	major_category	Classification of property_type field into larger groups	152-154	512-514
39	price	Original feature from dataset	158	518
40	price_per_person	Calculated field that measures price per person	166-167	522-523
41	room_type	Original feature from dataset	53	412
42	weekly_price	Original feature from dataset	167	527
43	monthly_price	Original feature from dataset	168	528
44	has_cleaning_fee	Field that checks if cleaning fee is applicable	169	529
45	has_security_deposit	Field that checks if security deposit is applicable	170	530
46	id	Field that uses row number	180	543
47	median_ppp_ind	Calculates the median price per person	184	547
48	ppp_ind	Classifies the properties based on price per person	188	551
49	features	Original feature from dataset	194-202	613-621
50	verification_methods	Original feature from dataset	228-229	560-561
51	verif_count	Number of different methods of verification	231-233	565-569

52	comp_price	Classifies the properties based on weekly price	238-240	574-576
53	min_booking_fee	Calculated field that measures the minimum fee to be paid to book the property	242-243	578-579
54	beds_accommodates	Calculated field that measures the ratio of number of accommodates to the number of beds	245-247	581-583
55	population	Population within the zipcode where the property is located	385-394	722-731
56	density	Population density within the zipcode where the property is located	385-394	722-731
57	dist2_medium_airport	Average distance from the zipcode location to the closest medium or large airport	385-394	722-731

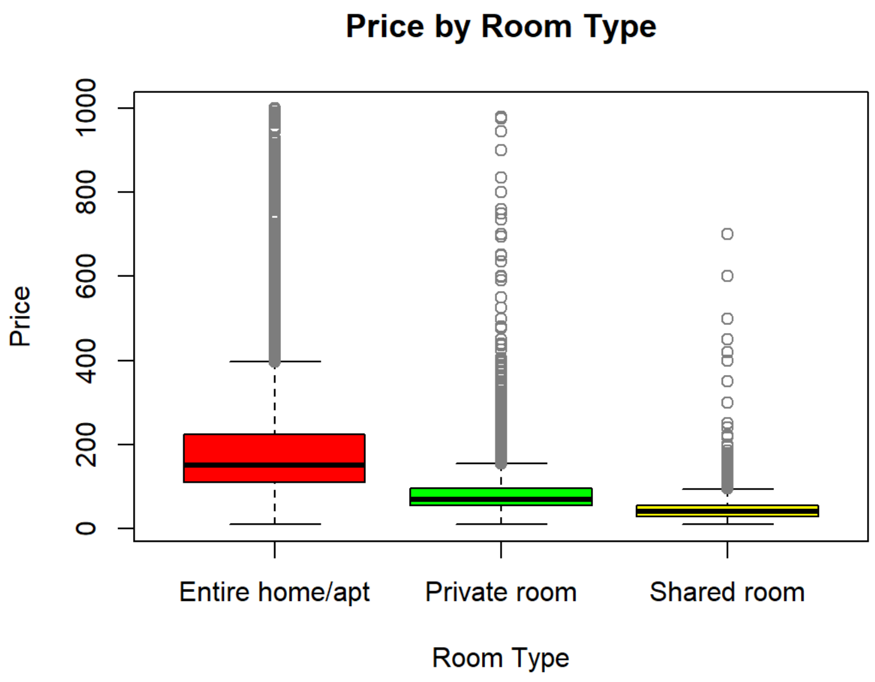
2) Graphs or tables demonstrating useful or interesting insights regarding features in the dataset.



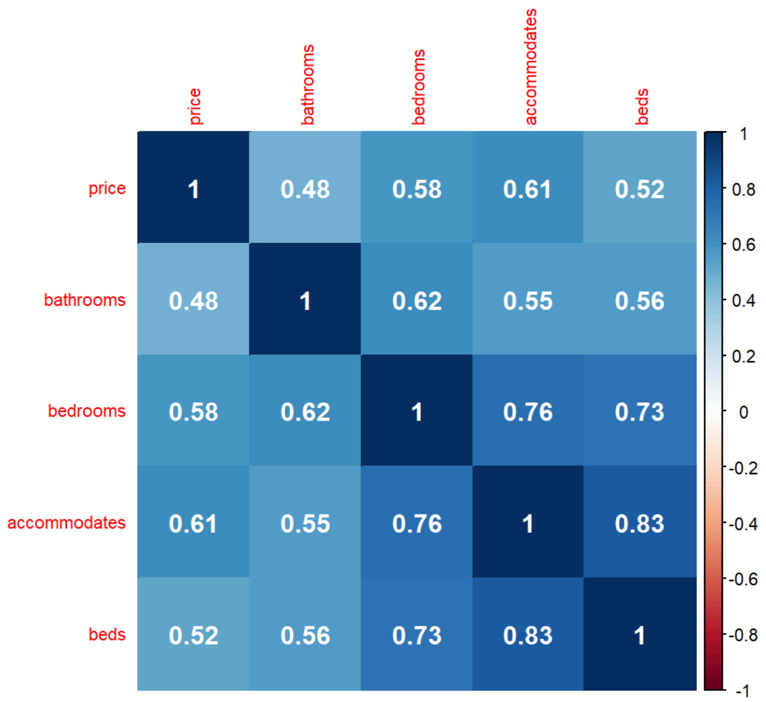
The Airbnb housing dataset has the largest count of property listings under $200.



The Airbnb housing dataset has the largest count of property listings for the entire/home apt category.

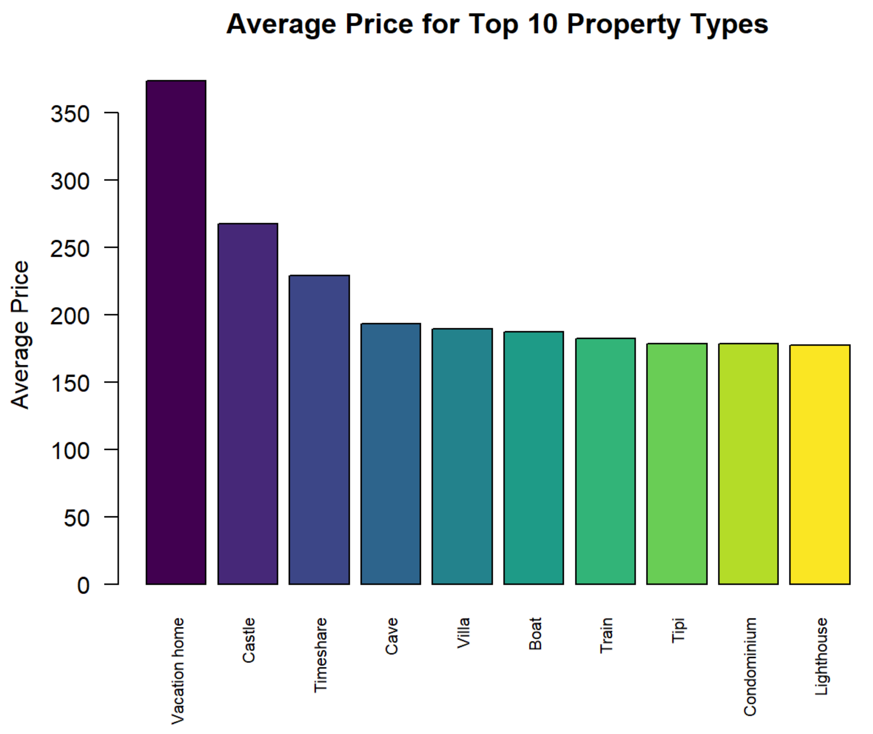


The property listings in the entire homes/apartments category tend to be priced higher than private rooms and shared rooms on average, with a wider range of prices and more significant variation among high-priced listings. Private rooms are generally less expensive than entire homes/apartments but more costly than shared rooms. Shared rooms have the lowest average prices with minimal variation except for some higher-priced outliers.

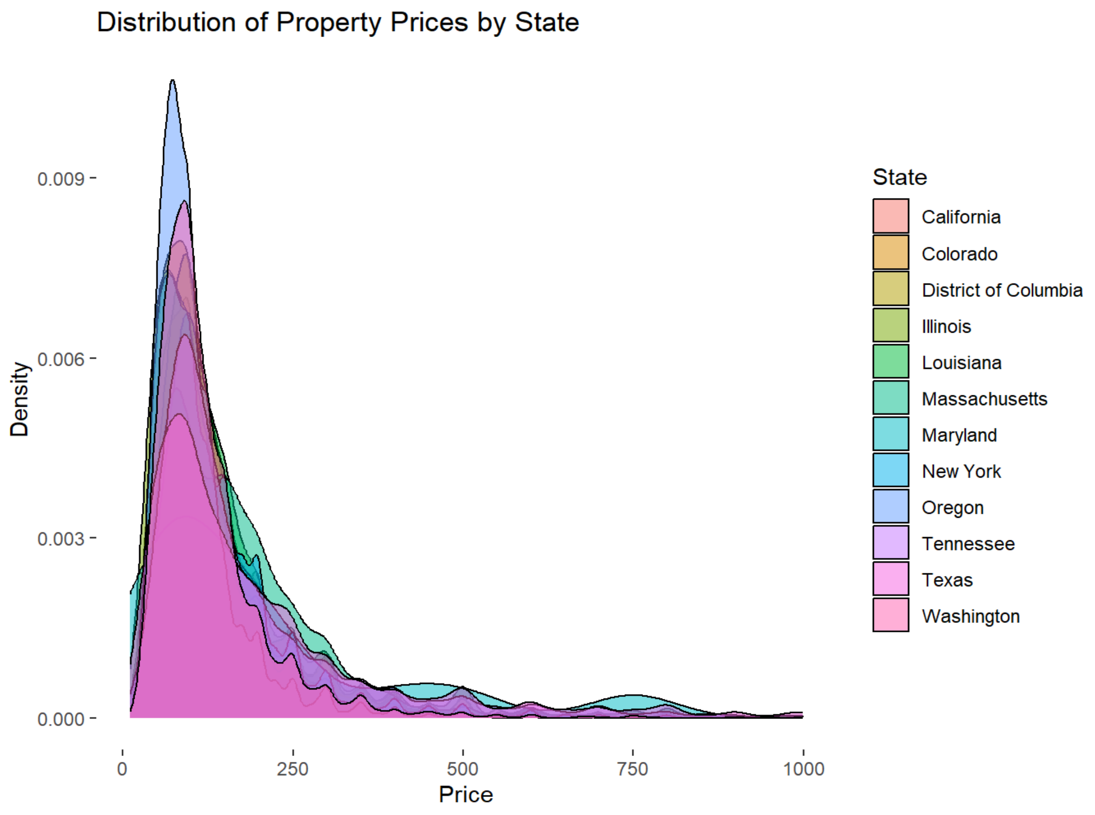


This plot indicates that there is a tendency for an increase in one feature to be associated with an increase in others, such as:

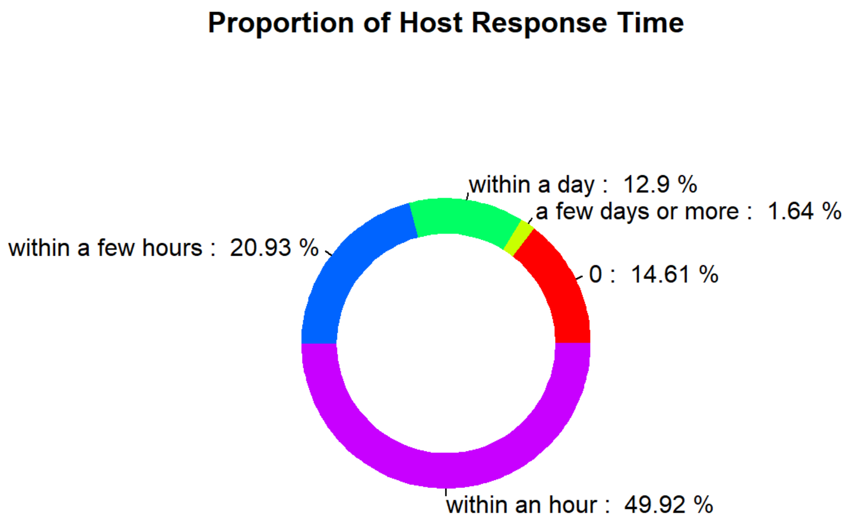
- Larger accommodations tend to have more bathrooms, bedrooms, and beds.
- Higher prices are linked to properties with more amenities (bathrooms, bedrooms, accommodates, and beds).



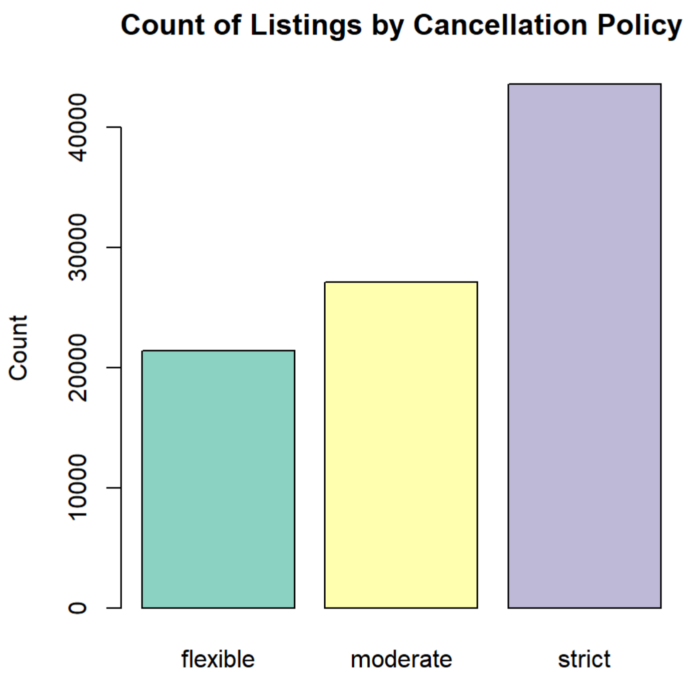
Vacation homes have the highest average price among the listed property types, followed by castle and timeshare.



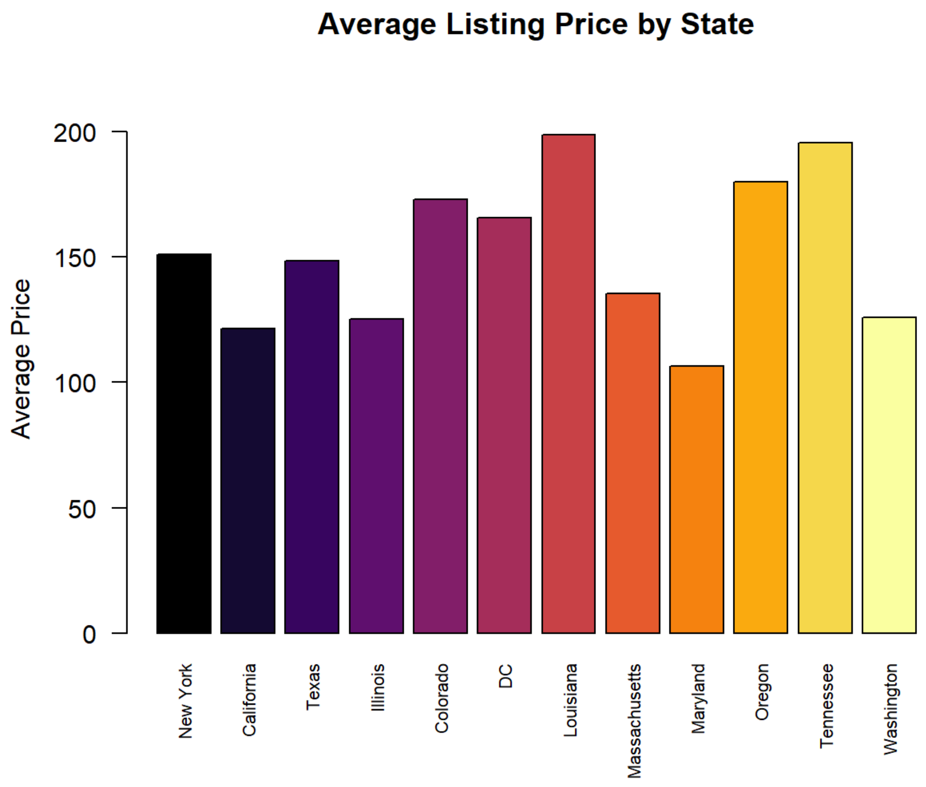
Most states exhibit property price distributions that peak below $250. This suggests a concentration of lower-priced properties across the country. Oregon has a much higher density in the $0 to $250 price range. In the $500 to $1000 price range, New York stands out with a much higher density.



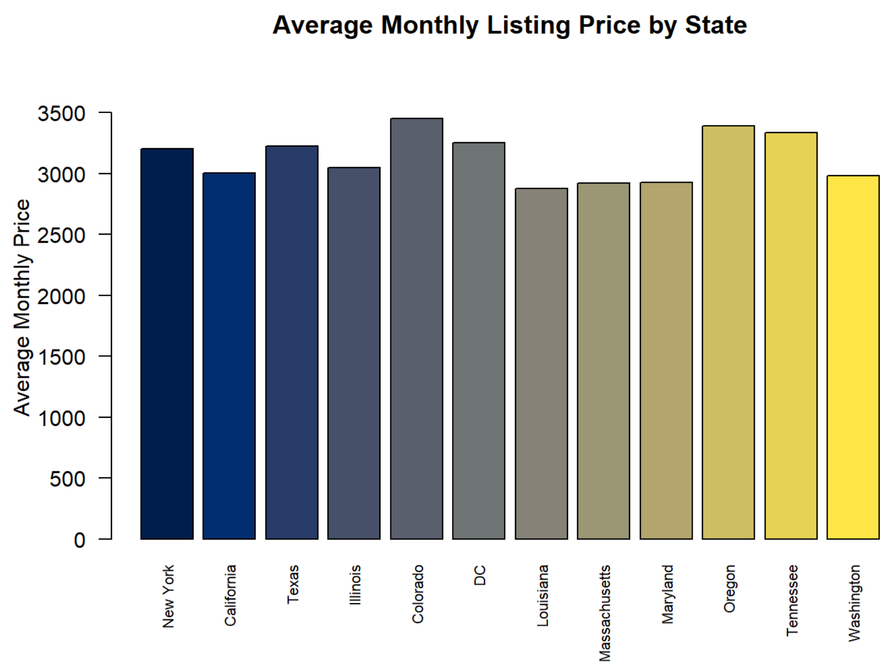
Most hosts respond promptly within an hour (49.92%), while a smaller percentage reply within a few hours (20.93%). Some hosts take longer, responding within a day (12.9%), and a small portion does not respond (14.61%). Very few hosts take several days or more to reply (1.64%).



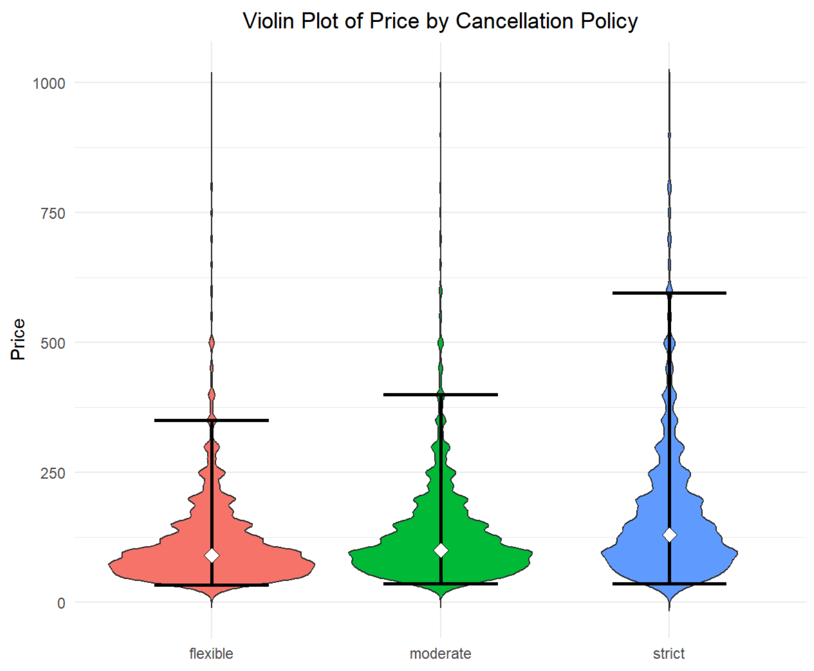
The strict cancellation policy has the highest count of approximately 40,000 listings among the three categories. Listings with moderate policies follow, with around 25,000 listings. The flexible policy category has approximately 20,000 listings. This suggests that hosts often prefer strict cancellation policies for booking a room in their listings.



Louisiana has the highest average listing price among the displayed states. Tennessee follows closely behind Louisiana. Oregon and Colorado also have relatively high average listing prices, while Washington has the lowest average listing price among all the states.



Colorado has the highest average monthly listing price among the states shown, followed by Oregon and Tennessee. Louisiana has a significant drop in average price compared to DC.



For the flexible cancellation policy (depicted in red), the majority of listings cluster at lower price points, indicating a prevalence of affordable options. However, there are outliers with notably higher prices.

In comparison, the moderate cancellation policy (shown in green) exhibits slightly higher prices on average, with a similar concentration at lower price points. However, there are thinner tails in the distribution, suggesting a greater presence of higher-priced listings.

Meanwhile, the strict cancellation policy (represented by the blue violin) displays a wider range of price points, with some listings commanding higher prices. The elongated shape indicates a more dispersed distribution of prices across the range of options.

3) We cleaned the amenities column which was comma separated and had multiple same values using the "gsub" method. Further we performed a split on the column based on "," and the features were further made dummies. This process was performed to incorporate the variable in the model to check if it boosts the performance.

Section 4: Evaluation and Modeling

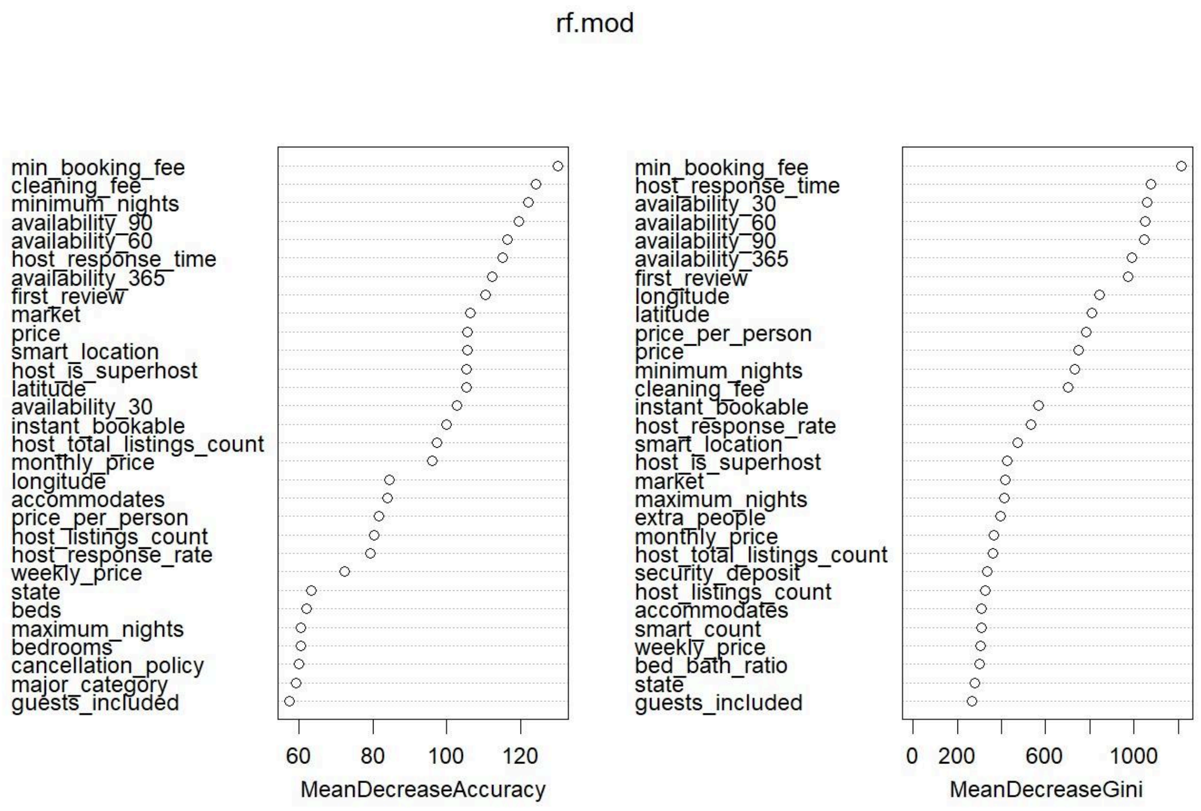
Random Forest (Winning Model)

Random Forest stands is an ensemble learning method, wherein each constituent tree is trained on a random subset of the available training data and subsequently makes predictions independently. The collective prediction of the Random Forest is then deduced by amalgamating the predictions from all individual trees. Essentially, within the family tree of a Random Forest lies a decision tree. This model derives its randomness from two primary sources:
i) Bootstrap Sampling: Each tree undergoes training on a bootstrapped sample of the training data, thereby ensuring diversity among the trees through repetitions and exclusions of observations.
ii) Feature Randomness: At every split in each tree, only a random subset of features is considered, thus aiding in the decorrelation of the trees and bolstering the model's robustness.

We chose Random Forest as our winning model. We found the best model after a lot of testing. It worked really well on the validation data, showing it could predict accurately. Also, it did a great job overall in handling complex data.

In our model, we utilized a comprehensive set of variables to predict the high booking rate. The model incorporated a total of **57** variables, each contributing uniquely to the prediction process. These variables encompassed a wide array of factors, including host-related metrics such as response time, response rate, and listings count, as well as property-specific attributes like price, availability, and room type. In our model, the most important features were: minimum booking fee, cleaning fee, minimum nights, availability for 90 days, availability for 60 days, host response rate, availability for 365 days, first review, market, price, smart location, and whether the host is a superhost. These features could have been identified as important because they directly influence key aspects of the guest experience and booking process. Minimum booking and cleaning fees affect the upfront cost perception, impacting guest affordability and willingness to book. The minimum nights requirement reflects booking flexibility, stay durations. High availability for specific periods indicates popularity and attractiveness, driving increased booking interest. Host responsiveness and positive reviews signal reliability and quality, fostering trust and encouraging bookings. Superhost status further reinforces a host's reputation for exceptional hospitality, attracting more guests. Pricing competitiveness and strategic location enhance a property's appeal, ensuring it stands out in the market and meets guest expectations effectively. Understanding these factors helps hosts optimize their listings to maximize booking rates and overall success.

The best set of features used to implement this model successfully are depicted in the graph below:



In the implementation phase, the randomForest library in R was employed, with specified hyperparameters such as mtry (the number of variables randomly sampled as candidates at each split) set to 6 and ntree (the number of trees in the forest) set to 2000. Performance evaluation ensued utilizing accuracy metrics, where the dataset was bifurcated into training and validation sets at a ratio of 70% and 30%, respectively. Code block for our Random Forest Model in R file: **[755-807]**

We left out some features because they weren't helping our model. For example, we tried using amenities, but they made the model less accurate overall. Similarly, we cleaned up the Jurisdiction count, but adding it didn't improve the model, so we took it out. This helped us focus on what really mattered in our data, making our predictions better.

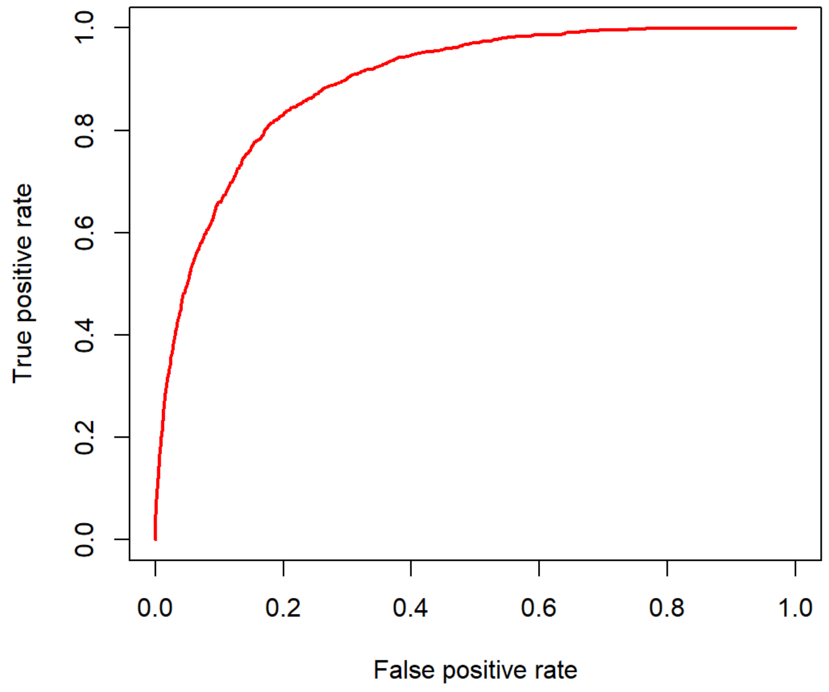
We tried lots of different settings to find the best one for our model. After testing them all thoroughly, we found that setting mtry to 6 and ntree to 2000 worked the best. So, we decided to go with that because it gave us the most accurate results compared to the other choices.

External Dataset: Incorporating an external dataset at the zip code level, which includes population statistics, population density, and distances to the nearest airports, proved to be a pivotal enhancement for our models. Ensuring perfect alignment between the zip code data from both datasets, we integrated these features into our winning model, and other models too.
Despite the positive impact of features from Airbnb dataset , the inclusion of this supplementary data led to a remarkable improvement. This enhancement notably boosted the model's AUC metric from 89.5 to 89.899, signifying a substantial advancement in predictive accuracy. The impact of this improvement was reflected in our contest standings, propelling us to a commendable third place finish.

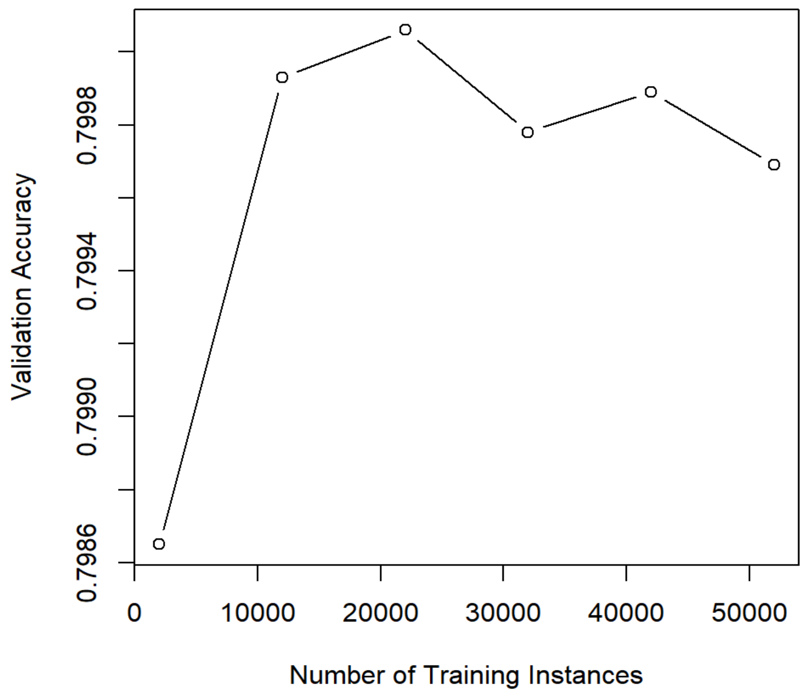
In our final model, we achieved impressive performance metrics. The AUC value, which measures the model's ability to distinguish between positive and negative classes, was **89.899**. This means our model was very good at telling the difference between properties with high booking rates and those with lower rates. Additionally, the accuracy of our model, representing the proportion of correctly predicted outcomes, was **85.41.** This high accuracy showed that our model made a lot of correct predictions. Overall, these strong performance metrics validate the reliability and effectiveness of our model in predicting high booking rates based on the selected variables which was the reason why we chose Random Forest as our winning model.

Fitting curves and ROC curves were developed to understand and represent the data better. The fitting curve showed how accurate our model was as we trained it with more data. This helped us see if our model was working well or not. The ROC curve showed how good our model was at telling positive and negative outcomes apart. You can see these curves in the attached images.

Fitting Curve:



Learning Curve:



Boosting Model:

Boosting is a powerful technique in machine learning aimed at enhancing the predictive performance of models through a collaborative and iterative process. It operates by training a series of weak learners—models that may perform slightly better than random chance—sequentially, with each subsequent model focusing on the examples that previous ones struggled with. By emphasizing the challenging instances and adjusting the prediction criteria at each step, boosting effectively combines the individual strengths of these weak learners into a strong ensemble model.

The model was trained using the gbm() function from the gbm package in R. This function, available within the gbm package, facilitated the training process and allowed for the implementation of the boosting method in our analysis. In our model, we utilized a comprehensive set of variables to predict the high booking rate. Code Block in R: **[809-857]**

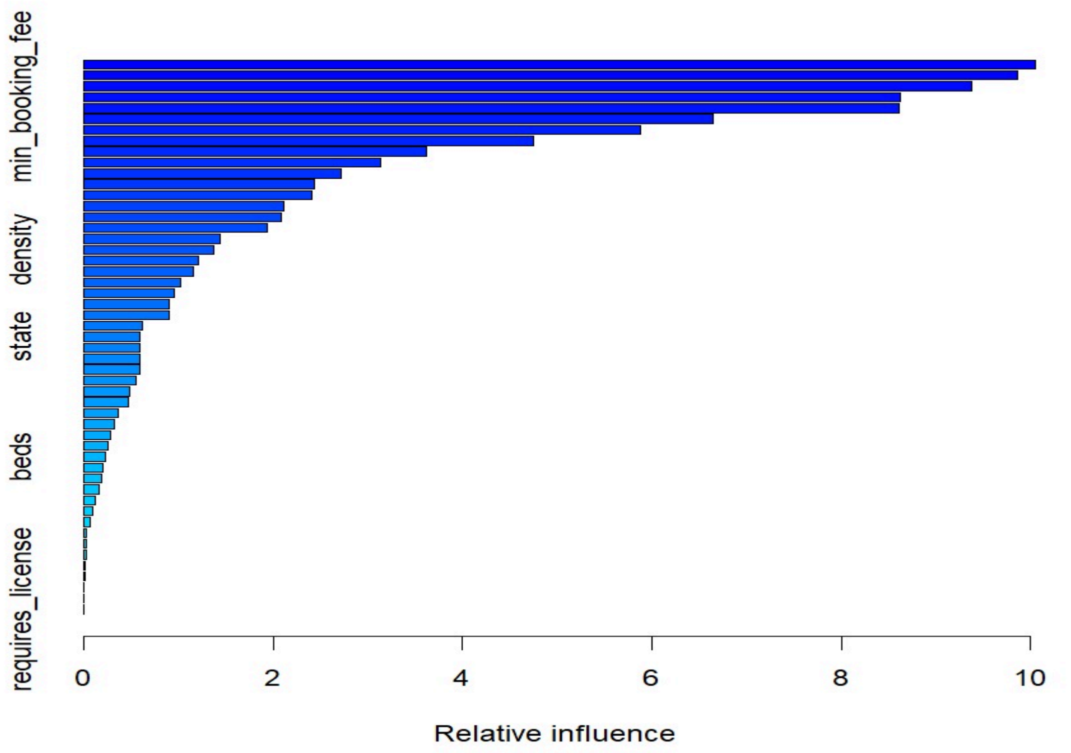
For our boosting model, the estimated training and generalization performance metrics are as follows:

- **AUC (Area Under the Curve):** 0.885
- **Accuracy:** 0.850114

These values represent the model's performance in terms of its ability to discriminate between classes and its overall correctness in predictions, respectively.

To estimate the generalization performance for our model, we employed a straightforward approach known as a simple split. Here's how it worked: we divided our dataset into two parts. Seventy percent of the data was used for training the model, while the remaining 30% was reserved for validation purposes. In total, our training dataset comprised 64,446 rows, while the validation dataset consisted of 27,621 rows. This method allowed us to assess how well our model performs on new, unseen data, providing valuable insights into its real-world predictive capabilities.

The following graph represents the best features from the model which were used to predict the high booking rate of airbnb properties:



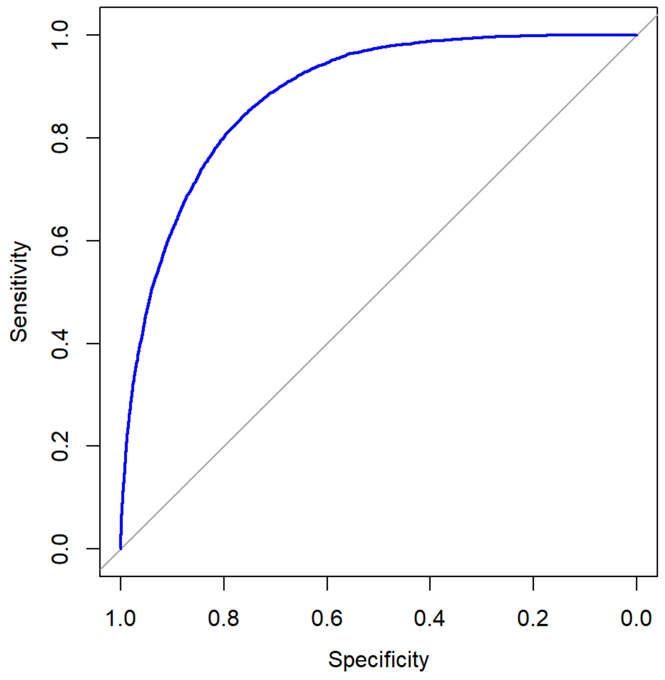
For tuning the model, we focused on adjusting two key hyperparameters:

1. **n.trees:** This parameter controls the number of trees (weak learners) in the ensemble. Initially, we experimented with different values to find the optimal balance between model performance and computational complexity. We tried setting n.trees to 2000 but observed no significant improvement in model performance. Ultimately, the value of n.trees that yielded the best performance was 1000.
2. **interaction.depth:** This hyperparameter governs the maximum depth of interaction between features in each tree. We explored various values to strike a balance between capturing complex relationships and avoiding overfitting. After testing different depths, we found that an interaction depth of 3 resulted in the best model performance.

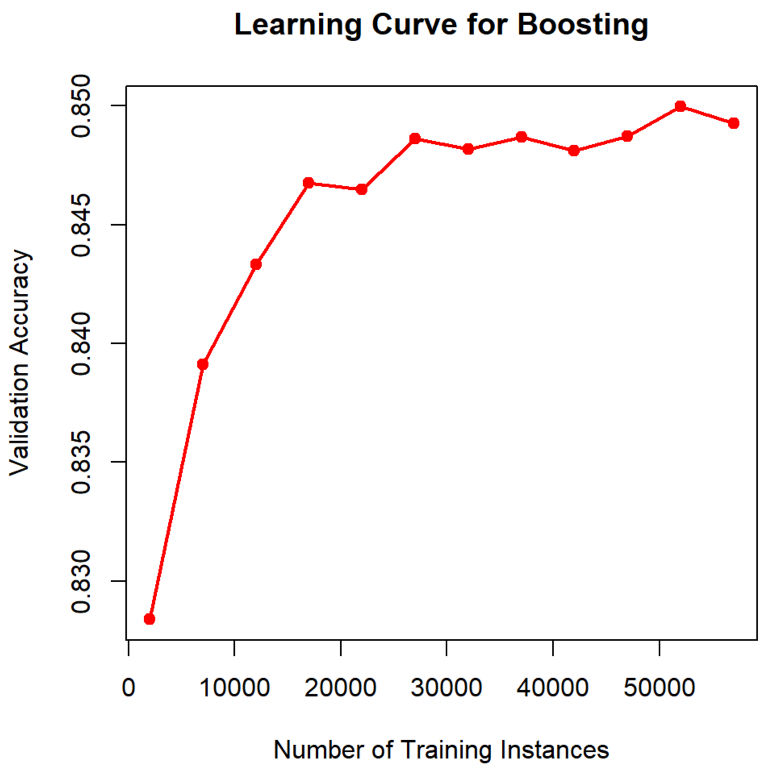
By carefully tuning these hyperparameters, we optimized the model to achieve the desired balance between predictive accuracy and generalization. This iterative process ensures that the model effectively captures relevant patterns in the data while avoiding overfitting.

Fitting curves and ROC curves were developed to understand and represent the data better. The fitting curve showed how accurate our model was as we trained it with more data. This helped us see if our model was working well or not. The ROC curve showed how good our model was at telling positive and negative outcomes apart. You can see these curves in the attached images.

Fitting Curve:



Learning Curve:



Bagging Model:

Bagging, short for Bootstrap Aggregating, is a machine learning technique aimed at improving the accuracy and robustness of models. It achieves this by creating multiple subsets of the original dataset through bootstrapping and training a separate model on each subset. During prediction, the outputs of these models are aggregated, typically by averaging for regression tasks or taking a majority vote for classification tasks. By leveraging the diverse perspectives captured by the ensemble of models, bagging helps to reduce variance, mitigate overfitting, and enhance the model's ability to generalize to new, unseen data. In essence, bagging acts as a collaborative approach where multiple models work together to provide more reliable predictions, akin to pooling insights from a diverse group of experts to make informed decisions.

For implementing the bagging method in R, we utilized the randomForest library. Within this library, we employed the randomForest function to create the bag.mod object. This function took several parameters: modelformula_bag, representing the model formula; data_train, which denoted the training data.
Code block in R**: [857-906]**

For our bagging model, the estimated training and generalization performance metrics are as follows:

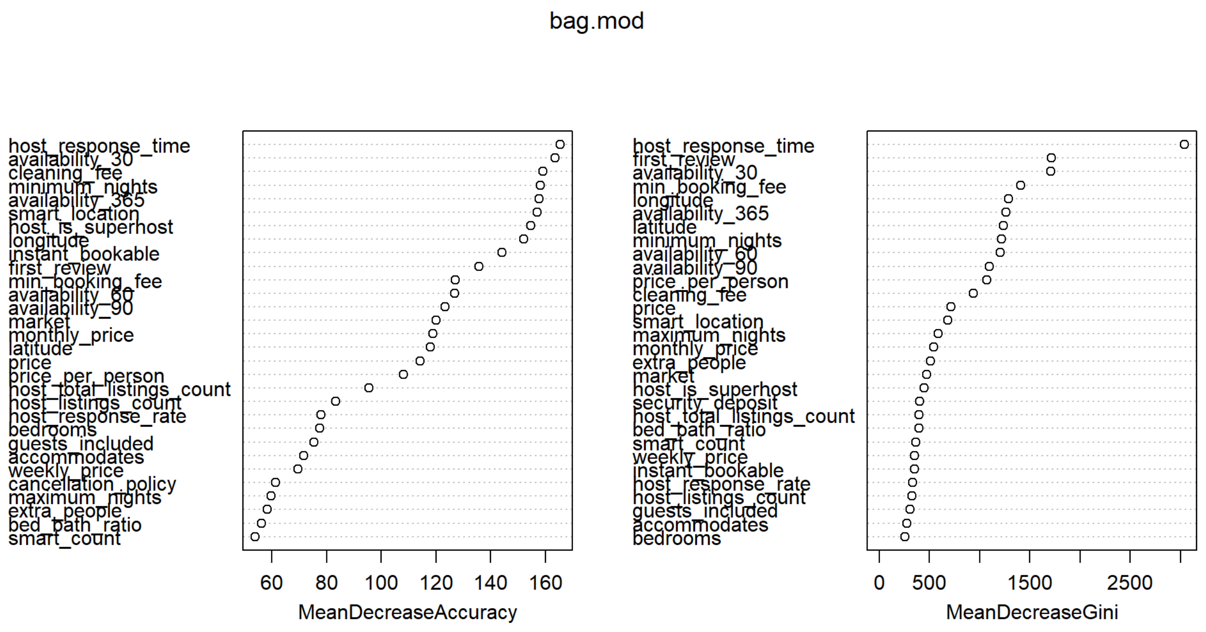
- **AUC (Area Under the Curve):** 0.8938069
- **Accuracy:** 0.8606276

These values represent the model's performance in terms of its ability to discriminate between classes and its overall correctness in predictions, respectively.

To assess the generalization performance of our bagging model, we employed a straightforward approach: a simple train/validation split. This methodology involved partitioning the dataset into two subsets, with 70% of the data allocated for training and the remaining 30% for validation. Specifically, the training dataset comprised 64,446 rows, while the validation dataset consisted of 27,621 rows. This validation setup allowed us to evaluate the model's performance on unseen data, providing insights into its ability to generalize to new observations. By utilizing this approach, we obtained valuable information about the model's effectiveness in real-world scenarios, enhancing our understanding of its predictive capabilities.

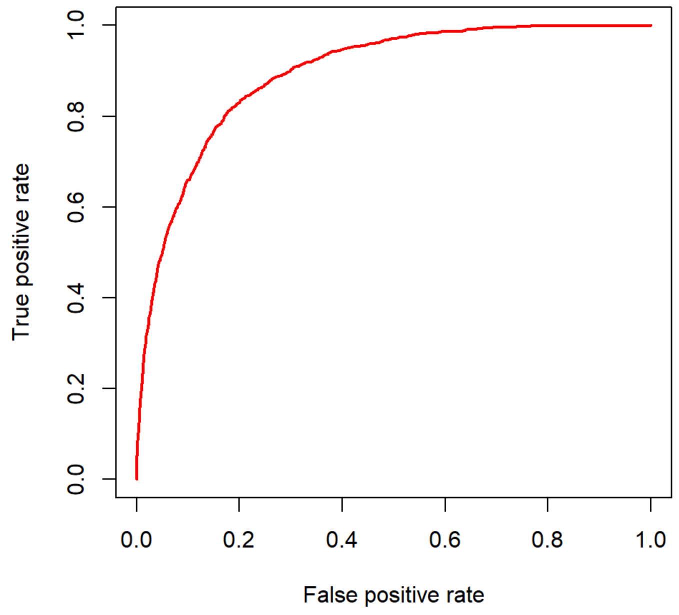
In our bagging model, we focused on tuning specific hyperparameters to optimize its performance. The primary hyperparameter we adjusted was ntree, which determines the number of trees to grow in the forest. By default, ntree is set to 500, but we experimented with alternative values, specifically 500 and 1000. Additionally, we fine-tuned another parameter, mtry, representing the number of features used in the model formula, which was fixed at 51. After implementing the bagging model with ntree values of 500 and 1000, we evaluated its performance. Our analysis revealed that the model achieved its best performance when ntree was set to 1000. This systematic tuning process allowed us to identify the optimal configuration for our bagging model, ensuring its effectiveness in capturing the underlying patterns in the data.

Importance plot:



Fitting curves and ROC curves were developed to understand and represent the data better. The fitting curve showed how accurate our model was as we trained it with more data. This helped us see if our model was working well or not. The ROC curve showed how good our model was at telling positive and negative outcomes apart. You can see these curves in the attached images.

Fitting Curve:



Logistic Regression

Logistic regression is a statistical method used for binary classification tasks, where the goal is to predict the probability that an observation belongs to one of two categories. It works by fitting a logistic curve to the data, which maps the relationship between the independent variables and the probability of the outcome. Unlike linear regression, which predicts continuous outcomes, logistic regression predicts probabilities using a logistic function, ensuring that the output falls between 0 and 1. This method is widely used in various fields such as healthcare, finance, and marketing for tasks like predicting whether a customer will churn, diagnosing diseases, or detecting spam emails. Its simplicity, interpretability, and effectiveness make logistic regression a popular choice for binary classification problems.

For implementing the logistic regression model in our analysis, we utilized the glm() function from the base R package. This function, which stands for Generalized Linear Models, provides a versatile framework for fitting various types of regression models, including logistic regression. By leveraging the glm() function, we were able to construct and train our logistic regression model directly within the R environment. This approach not only simplified the implementation process but also ensured compatibility and reliability, as the function is a fundamental component of the base R package.Code block in R : **[906-960]**

For our logistic regression model, the estimated training and generalization performance metrics are as follows:

- **AUC (Area Under the Curve):** 0.754
- **Accuracy:** 79.79%

These values represent the model's performance in terms of its ability to discriminate between classes and its overall correctness in predictions, respectively.

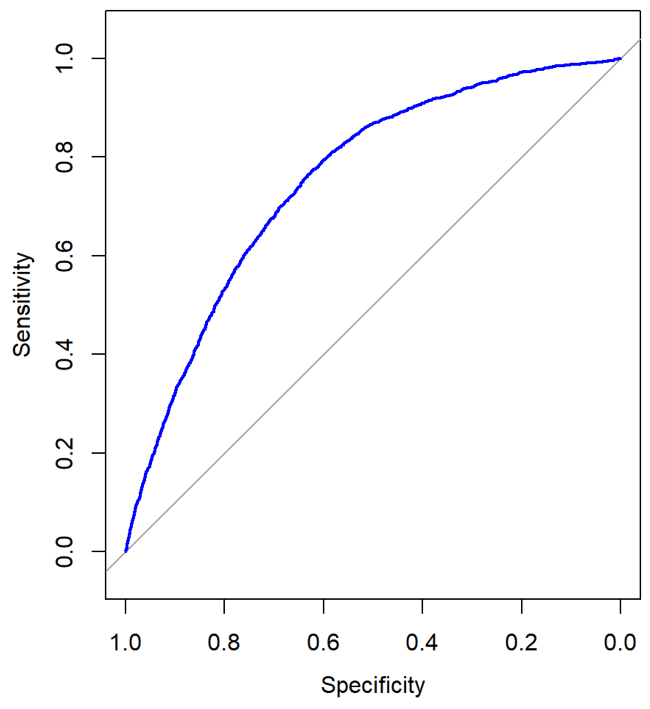
To evaluate the generalization performance of our logistic regression model, we employed a 10-fold cross-validation methodology. This involved dividing the dataset into ten equal parts or folds. Subsequently, the model was trained nine times on different combinations of nine folds, with each fold serving as the validation set once. By iteratively training and testing the model across multiple data subsets, we obtained a comprehensive assessment of its performance. This approach ensured that the model's effectiveness was evaluated on diverse portions of the data, helping to mitigate biases and provide robust insights into its predictive capabilities. Overall, 10-fold cross-validation served as a rigorous and reliable method for estimating the generalization performance of our logistic regression model.

The following features were used for logistic model:
accommodates , bathrooms , bedrooms , beds , price , guests_included , extra_people , minimum_nights , maximum_nights , availability_30 , availability_60 , availability_90 , availability_365 , room_type , cancellation_policy , bed_type , weekly_price , monthly_price

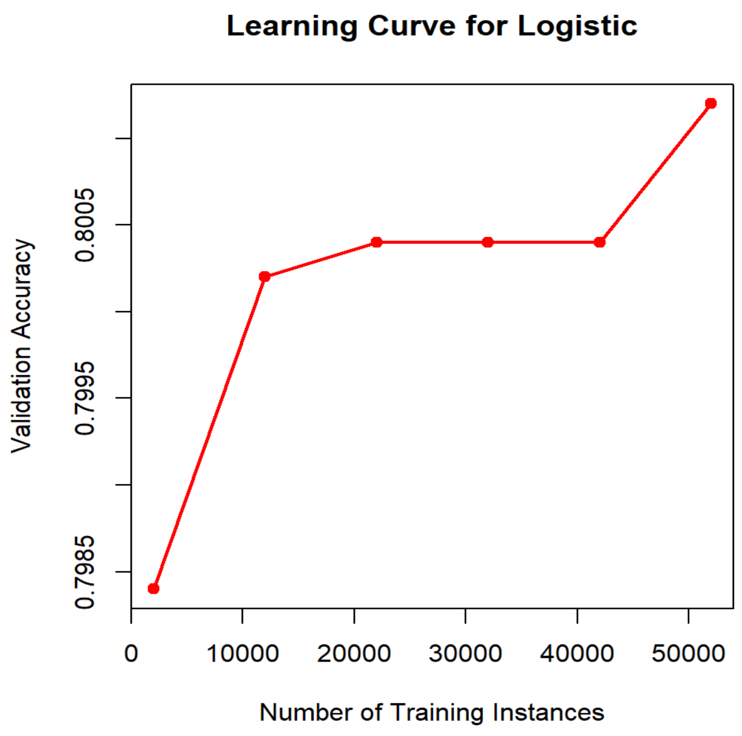
In our logistic regression model, we opted not to tune any hyperparameters, instead choosing to assess the model's performance without any adjustments. This decision was made to evaluate how the model performs under its default settings and to understand its baseline performance without additional fine-tuning. By abstaining from hyperparameter tuning, we aimed to gain insights into the model's inherent capabilities and to assess its performance in its most basic form. This approach allowed us to establish a benchmark against which any future optimizations or adjustments could be compared, providing valuable insights into the model's behavior and effectiveness in its original state.

Fitting curves and ROC curves were developed to understand and represent the data better. The fitting curve showed how accurate our model was as we trained it with more data. This helped us see if our model was working well or not. The ROC curve showed how good our model was at telling positive and negative outcomes apart. The curves are attached:

Fitting Curve:



Learning Curve:



Ridge Regression:

Ridge regression is a method used to enhance the performance of linear models like logistic regression. It tackles overfitting, a common problem where the model learns too much from the training data and struggles to generalize to new data. In ridge, a penalty term is added to the model's cost function, controlled by a parameter called lambda. This penalty discourages the model from relying too heavily on any single feature, effectively shrinking the coefficients towards zero. By doing so, ridge regression encourages simpler models that are less sensitive to small fluctuations in the data, resulting in more reliable predictions on unseen data. It's like finding a balance between fitting the training data well and avoiding too much complexity, ensuring the model works well in real-world scenarios.

For fitting the logistic regression model with a ridge penalty, we utilized the glmnet R library. This library provides efficient implementations of regularization methods, including ridge regression, which is particularly useful for preventing overfitting in linear models like logistic regression. In our implementation, we set the alpha parameter to 0 to specifically implement the ridge penalty. By leveraging glmnet, we were able to incorporate the ridge penalty seamlessly into our logistic regression model, enhancing its stability and generalization performance. This approach ensures that our model strikes the right balance between capturing important patterns in the data and avoiding excessive complexity, thereby improving its reliability in real-world applications. Code block in R: **[963-1032]**

For our Ridge Regression model, the estimated training and generalization performance metrics were as follows:

- **AUC (Area Under the Curve):** 0.8182
- **Accuracy:** 80.23%

These values represent the model's performance in terms of its ability to discriminate between classes and its overall correctness in predictions, respectively.

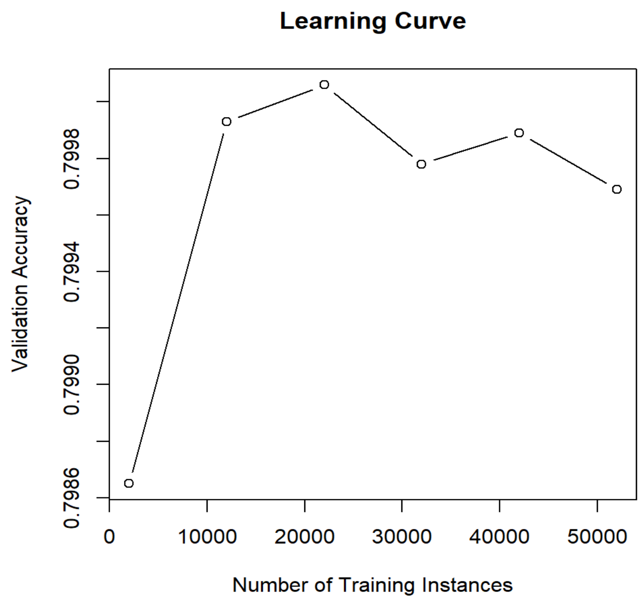
To estimate the generalization performance of our logistic regression model with ridge penalty, we employed a 10-fold cross-validation approach. This method involved dividing the dataset into ten equal parts or folds. Subsequently, the model was trained and evaluated ten times, with each fold serving as the validation set once while the remaining folds were used for training. This process allowed us to assess the model's performance across various data subsets and mitigate biases that may arise from a single train/test split. Additionally, after identifying the best-performing parameters through cross-validation, we further evaluated the model's performance using the chosen parameters. By leveraging cross-validation and validating the model on the best parameters, we obtained a comprehensive understanding of its generalization capabilities and ensured robustness in our analysis.

The following features were used for ridge regression model:
host_response_rate , host_listings_count , host_total_listings_count ,price_per_person , state , market , smart_location , room_type , accommodates , bathrooms , bedrooms , beds , bed_type , price , weekly_price , monthly_price , security_deposit , cleaning_fee , guests_included , extra_people , minimum_nights , maximum_nights, availability_30 ,availability_60 , availability_90 , availability_365 , host_acceptanced , availability , market_count , smart_count , major_category , latitude , longitude , price_per_person , has_cleaning_fee , has_security_deposit , host_listings , bed_bath_ratio , require_guest_profile_picture , requires_license , require_guest_phone_verification , host_is_superhost , instant_bookable , host_identity_verified , is_location_exact , host_has_profile_pic , sent_score_houserules , sent_score_summary , min_booking_fee

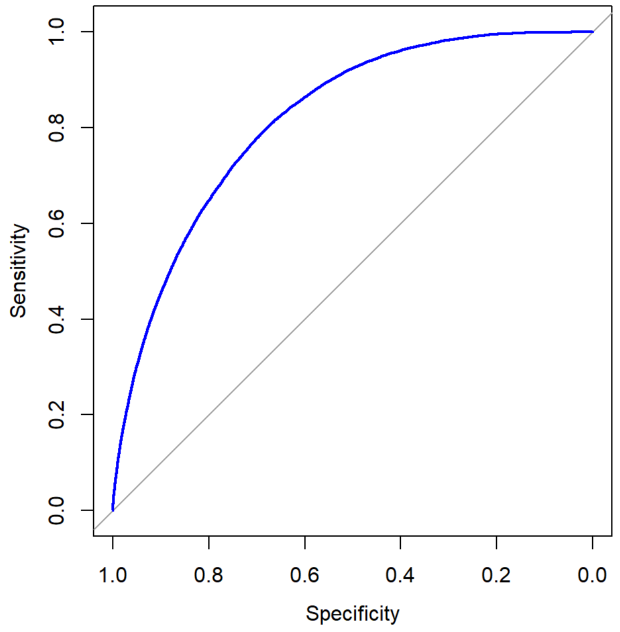
In our logistic regression model with ridge penalty, we focused on tuning the regularization parameter, lambda, to optimize model performance. Lambda controls the strength of the penalty applied to the coefficients during ridge regression, influencing the model's ability to prevent overfitting. We experimented with different values of lambda to find the optimal balance between bias and variance in our model. By systematically testing various values of lambda, we aimed to identify the configuration that maximized the model's generalization performance. This process allowed us to fine-tune the regularization strength and improve the model's ability to make accurate predictions on unseen data. Overall, tuning the lambda parameter played a crucial role in enhancing the effectiveness and reliability of our logistic regression model with ridge penalty.

Fitting curves and ROC curves were developed to understand and represent the data better. The fitting curve showed how accurate our model was as we trained it with more data. This helped us see if our model was working well or not. The ROC curve showed how good our model was at telling positive and negative outcomes apart. The curves are attached:

Learning Curve:



Fitting Curve:



Lasso Regression:

Lasso, short for Least Absolute Shrinkage and Selection Operator, is a method used to refine the performance of linear models, such as logistic regression, in statistics and machine learning. It works by incorporating a penalty term into the model's cost function, penalizing coefficients with large magnitudes. Unlike some other techniques, lasso can drive some coefficients all the way to zero, effectively removing less important features from the model. This feature selection capability makes lasso particularly valuable for dealing with datasets with many features, as it helps identify and prioritize the most relevant ones while disregarding the rest. By encouraging simpler models with fewer features, lasso aids in preventing overfitting and enhancing the model's ability to make accurate predictions on new data.

In implementing the logistic regression model with lasso penalty, we relied on the glmnet R library. This library provides efficient implementations of regularization methods, including lasso regression, which is instrumental in improving the stability and performance of linear models like logistic regression. By setting the alpha parameter to 1, we specifically implemented the lasso penalty, which encourages sparsity in the coefficient estimates, effectively performing feature selection. Utilizing glmnet allowed us to seamlessly incorporate the lasso penalty into our logistic regression model, enhancing its predictive accuracy and preventing overfitting. This approach ensures that our model is robust and capable of making reliable predictions on new data, thereby contributing to more informed decision-making in practical applications. Code block in R: **[1034-1106]**

For our LassoRegression model, the estimated training and generalization performance metrics were as follows:

- **AUC (Area Under the Curve):** 0.8283
- **Accuracy:** 81.34%

These values represent the model's performance in terms of its ability to discriminate between classes and its overall correctness in predictions, respectively.

To assess the generalization performance of our logistic regression model with lasso penalty, we employed a 10-fold cross-validation approach. This methodology involved splitting the dataset into ten equal parts, or folds. Subsequently, the model was trained and evaluated ten times, with each fold serving as the validation set once while the remaining folds were used for training. This process allowed us to comprehensively assess the model's performance across various data subsets and mitigate biases that may arise from a single train/test split. Additionally, after identifying the best-performing parameters through cross-validation, we further evaluated the model's performance using the chosen parameters. Leveraging cross-validation in this manner ensured robustness in our analysis and provided reliable insights into the model's generalization capabilities.

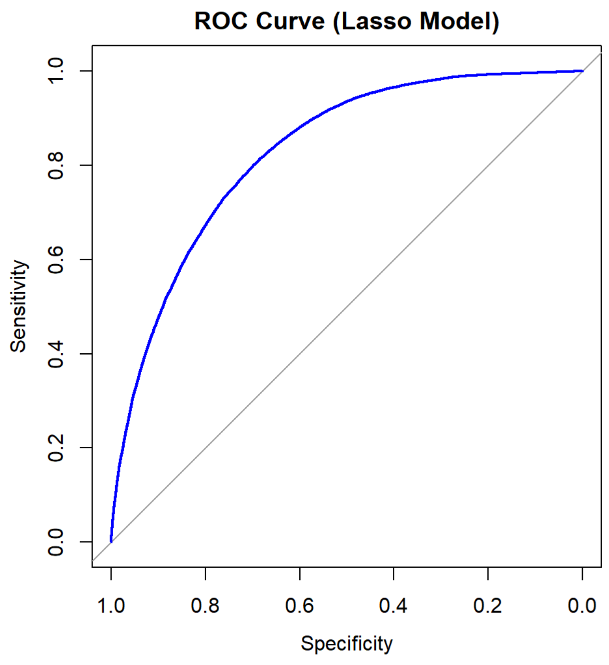
The following features were used for lasso regression model:
host_response_rate , host_listings_count , host_total_listings_count ,price_per_person , state , market , smart_location , room_type , accommodates , bathrooms , bedrooms , beds , bed_type , price , weekly_price , monthly_price , security_deposit , cleaning_fee , guests_included , extra_people , minimum_nights , maximum_nights, availability_30 ,availability_60 , availability_90 , availability_365 , host_acceptanced , availability , market_count , smart_count , major_category , latitude , longitude , price_per_person , has_cleaning_fee ,

has_security_deposit , host_listings , bed_bath_ratio , require_guest_profile_picture , requires_license , require_guest_phone_verification , host_is_superhost , instant_bookable , host_identity_verified , is_location_exact , host_has_profile_pic , sent_score_houserules , sent_score_summary , min_booking_fee

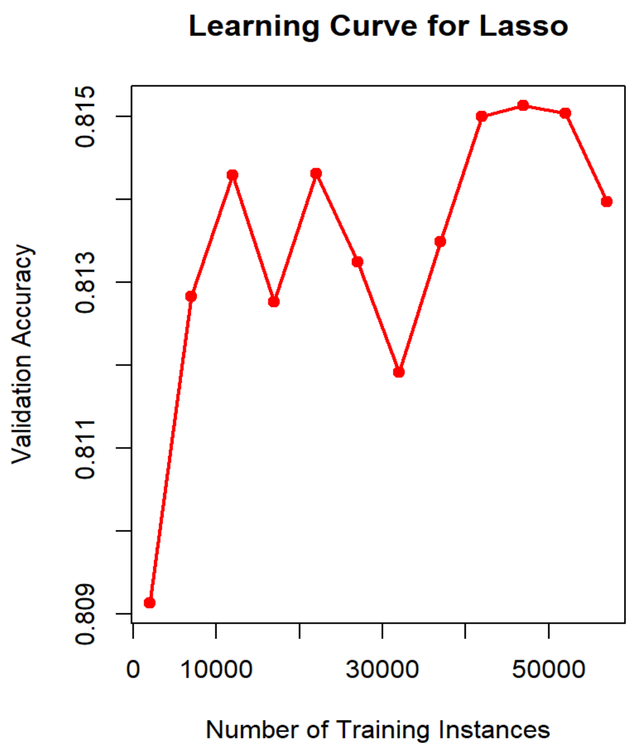
In optimizing our logistic regression model with lasso penalty, we set the regularization parameter, lambda, to a fixed value of 1. This choice influenced the strength of the lasso penalty applied to the coefficients, impacting the model's ability to prevent overfitting and perform feature selection. While our approach may have been simpler compared to more extensive hyperparameter optimization techniques, the selection of lambda still played a crucial role in shaping the model's behavior and performance. By setting lambda to 1, we aimed to strike a balance between regularization strength and model simplicity, leveraging the benefits of lasso regression. Overall, this decision contributed to the effectiveness and robustness of our logistic regression model with lasso penalty.

Fitting curves and ROC curves were developed to understand and represent the data better. The fitting curve showed how accurate our model was as we trained it with more data. This helped us see if our model was working well or not. The ROC curve showed how good our model was at telling positive and negative outcomes apart. You can see these curves in the attached images.

Fitting Curve:



Learning Curve:



Section 5: Reflection/Takeaways

As we look back on our project, we've identified some areas where we did well and others where we faced big challenges. Here's a detailed rundown of what we think we did right, what we struggled with, and what we'd do differently if we had the chance to start over.

What Our Team Did Well

One of the best things about our team was how well we communicated. We made sure everyone felt comfortable speaking up and sharing their ideas. We were always there for each other, making sure to lend a helping hand and provide clear explanations to everyone, ensuring that everyone stayed informed and understood what was going on. Plus, we knew each other's strengths and divided the work accordingly, so everyone felt happy with what they were doing. This teamwork made our project run smoothly and made our work better.

Main Challenges We Faced

The toughest part of our project was cleaning up the data.The dataset we had was quite complicated, with lots of mistakes and missing information. It felt like trying to solve a puzzle with many pieces missing or in the wrong place. Despite this, we didn't give up. We spent a lot of time carefully fixing errors and making sure the data was accurate. We had to spend a lot of time trying to fix everything, and sometimes it felt like we were going in circles. It was frustrating and took a lot of effort from all of us.

What We'd Do Differently
If we could start over from scratch, the first thing we'd do is clean the data right from the beginning. We made the mistake of trying to build models before we finished cleaning up the data, and it caused problems later on. So, next time, we'd tackle the data cleaning first to avoid headaches down the road.

If We Had More Time
With more time, we'd focus on making our models even better. We'd spend extra time figuring out which features are the most important for making accurate predictions. That way, our models would be even more reliable and helpful.

Advice for Future Teams
For the next group doing this project, we have some advice:

- Take the time to clean the data properly from the start. It might seem boring, but it's really important for making good predictions.
- Make sure everyone feels included and comfortable sharing their ideas. Teamwork is key to success.
- Don't rush through things. Take your time to do things right, even if it means the project takes a bit longer.
- Don't be afraid to ask for help if you're stuck. We're all in this together!

In summary, our project had its ups and downs, but we learned a lot along the way. Good communication and teamwork helped us overcome challenges, but we know now that cleaning the data is super important. Hopefully, our experiences can help the next group do even better.