ACTL3142: Statistical Machine Learning for Risk and Actuarial Applications

# Airbnb Sydney Price Analysis
Assignment Part 2

Audrey Chang
Z5627566

**1.1 Creating the HAS Variable**

The proportion of listings with HAS = 1 is 0.3115. 31.15% of listings are classified as high scoring based on user reviews.

**2.1 Data Preparation Cleaning**

- Removed irrelevant columns: bathrooms_text, host_since.
- Created host_experience_years = 2025 -  host_since.
- Create high_availability: Binary feature based on the median of availability_60.
- Imputed missing values: host_response_time-"Unknown"; host_response_rate and host_acceptance_rate-median values; host_is_superhost-most frequent value (0, not a superhost)

**2.2 Feature Engineering**

- **Region**: Derived from neighbourhood using domain-specific groupings (East, West, North, South).
- Property Group: Categorized property_type into logical groups (e.g., Apartment/Condo, Hotel/Hostel, Guest Accommodation).
- **High Availability**: Binary feature based on whether availability_60 is above the median.
- **Interaction Terms:**
  - host_experience_years × host_is_superhost: Indicates consistent quality over time.
  - price × availability_60: Captures trade-offs between price and availability.
  - room_type × property_group: Captures contextual differences in guest expectations.
- Log transformation: Applied to price to handle skewness.
- Re-leveled categorical variables: Adjusted reference levels for interpretability.

**2.3 Exploratory Analysis and Variable Selection**

- Removed avg_score and individual review scores to prevent data leakage.
- Assessed predictive strength of numerical and binary predictors with HAS using point-biserial and phi correlations, respectively. Top features:(See Appendix for correlation)
  - Positively correlated: host_is_superhost, price, longitude, host_experience_years, latitude
  - Negatively correlated: calculated_host_listings_count, instant_bookable, availability_60
- Visualized categorical predictor relationships with HAS using stacked bar plots. (See appendix)

**2.4 Train-Test Split**

- Random 75%/25% split for training and testing.

**2.5 Model Comparison and Justification(See Appendix for Model Comparison and ROC curves and AUC scores)**

**2.6 Model Selection and Justification(See Appendix for model summary output.)**

The logistic regression model was selected as the final model for the following reasons:

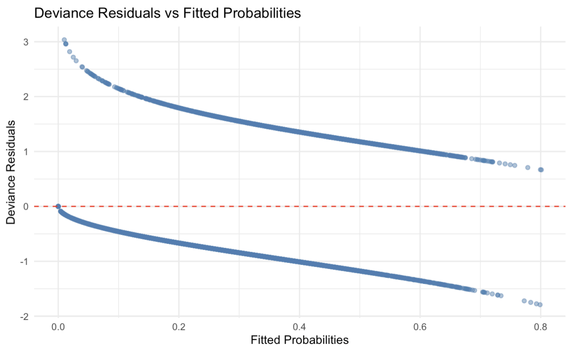
| **Criterion** | **Justification** |
|---|---|
| Balanced Predictive Power | Achieved the highest AUC among all models with balanced accuracy, recall & F1-score. |
| High Interpretability | Provides transparent coefficients, ideal for stakeholder interpretation. |
| Strategic Feature Inclusion | Includes key predictors like host_is_superhost, log_price, and meaningful interactions. |
| Justifiable Feature Selection | Selected using stepwise selection of lowest AIC(6421.5) and EDA insights, ensuring model parsimony and relevance. |
| Best Recall Performance | Recall of 60.1%, the highest among all models, ensures accurate detection of high-quality listings. |
| Most Competitive Overall | Although Ridge had slightly higher accuracy, its recall was significantly lower. |

**3.1 Model Evaluation**

- **Model Fit:** The model's Null Deviance (7145.5) and Residual Deviance (6371.5) show a substantial reduction, indicating a good fit. The AIC (6421.5) suggests the model is relatively efficient. Most predictors are statistically significant, reinforcing the model's strength.

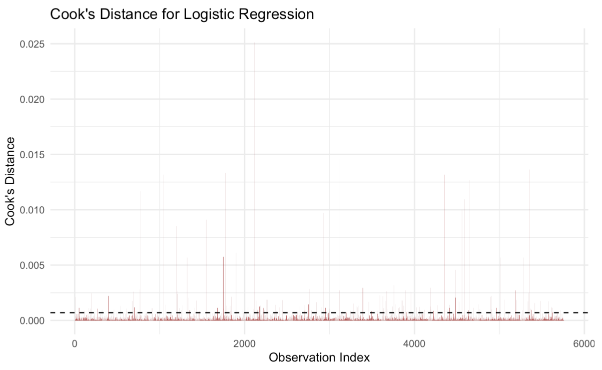
- **Model Coefficients**: Most of the model's predictors are statistically significant and their coefficients are directionally sensible, meaning they align with expectations based on domain knowledge.

### 3.1.1 Diagnostics & Validation



- **Confusion Matrix:** Shows balanced classification across both HAS classes, with no major imbalance.
- **ROC Curve & AUC**: AUC = 0.722 indicates solid discrimination power between HAS = 1 and HAS = 0.
- **Model Coefficients**: Most predictors are statistically significant and directionally sensible (details in Q4).

### 3.1.2 Residual & Deviance Diagnostics



- Deviance residuals show slight skew above zero and mild heteroscedasticity, especially at low predicted probabilities.
- A few residuals exceed ±2, suggesting moderate outliers.
- No strong curvature or funnel shape in plots, supporting adequate model fit.

### 3.1.3 Influence Diagnostics:

  - Cook's Distance values are mostly below 0.025, with no major leverage issues.
  - Minor spikes (around index 2000 and beyond) are non-distorting.

### 3.1.4 Potential Limitations

| **Issue** | **Explanation** |
|---|---|
| Linearity Assumption | Logistic regression assumes log-odds are linearly related to predictors. This may oversimplify effects (e.g., price or geography). |
| Interaction Overfitting | Sparse categories in some room-type × property-group interactions could lead to overfitting. |
| Recall at 60.1% | Although balanced, the model still misses some HAS=1 listings |
| Limited Flexibility | Logistic regression may underperform compared to nonlinear models like Random Forest or XGBoost in capturing complex patterns. |
| Business Impact of False Negatives | With a recall of 60.1%, ~40% of truly high-rated listings are missed. For Airbnb hosts and investors, these false negatives may lead to lost visibility, underpricing, and fewer bookings — making recall improvement a key business priority. |

## 4.1 Insights for a Sydney Investor with HAS=0 Listings

| **Recommendation** | **Model Evidence** | **Strategic Action** |
|---|---|---|
| Become a Superhost | host_is_superhost, +0.5361, p < 2e-16 | Focus on improving host responsiveness, enhancing reviews, and avoiding cancellations to elevate the profile. |
| Align Price with Quality | log_price, +0.7267, p < 2e-16 | Invest in premium property features and avoid underpricing when offering high-quality value. |
| Avoid Overextending | calculated_host_listings_count, -0.0181, p < 2e-16 | Limit the number of listings to maintain high-quality, personalized service for guests. |
| Limit Excessive Availability | high_availability, -0.3465, p < 0.001 | Consider offering exclusive or time-limited booking windows to create a sense of exclusivity. |

| | | |
|---|---|---|
| Be Cautious with Instant Book | instant_bookable, -0.4813, p < 0.001 | Allow for guest communication before booking whenever possible to ensure both parties' expectations are clear. |
| Favor South Sydney | regionSouth, +0.3523, p = 0.0267 | Target regions in South Sydney, where listings show a positive regional effect. |
| Consider Eastern Suburbs | longitude, +0.8272, p = 0.049 | Focus on properties near beaches or the CBD (e.g., Bondi), as these areas tend to generate higher ratings. |

**Room Type × Property Group Insights**

**-Top Performing Combinations**

Private rooms generally perform well when paired with home-like or distinctive property types:

- Private room × Guest Accommodation (+1.192, p < 0.001), House/Townhouse/Villa (+1.008, p < 0.001), and Unique Stays (+1.279, p < 2e-16)score highly—guests value privacy, homeliness, and unique experiences.
- Private room × Apartment/Condo (+1.027, p = 0.0029) also performs strongly, likely due to affordability and convenience.

Entire home options also see success when the property offers charm or hospitality focus:

- Entire home × Guest Accommodation (+0.8555, p < 0.001) and Cottage/Cabin/Bungalow (+0.7475, p = 0.002) attract guests seeking either professional hosting or rustic, private retreats.

**-Underperforming Combination**

- Private room × Hotel/Hostel (−1.887, p = 0.0015) performs poorly—likely because guests booking private rooms expect more personalization than standard hotel settings provide.

**5.1 Data Preprocessing & Feature Engineering, used the cleaned dataset from Q2. Key steps included:**

- Log Transformation on response: Applied to price to reduce skewness; used log_price in all models.
- Review Score Proxy: Kept avg_score for review-related insights.
- Engineered Features
  - **Activity Duration**: days_between_reviews = last - first review.
  - **PCA**: Applied to accommodates, beds, bedrooms, and bathrooms to reduce dimensionality and multicollinearity; retained 1st PC (PC_capacity). (Based on correlation plot)

**5.2 Feature Selection (See Appendix for Random Forest's %IncMSE plot)**

- Used Random Forest's %IncMSE metric to assess variable importance and identify predictive strength.
- Performed cutoff sensitivity analysis using top 10, 15, and 20 features to evaluate trade-offs.
- Selected top 20 predictors to reduce dimensionality and multicollinearity → df_reduced used in all models

**5.3 Model Development(See Appendix for model summary)**

Split: 60% train, 20% validation, 20% test.

**5.3.1 Random Forest**

Captured non-linear effects and variable interactions. Hyperparameter grid search included:

- mtry: 10, 15, 20 and ntree: 100, 300, 500

Trained models on training data; best combination selected using validation RMSE and MSE.

**5.3.2 Lasso Regression**

- Used alpha = 1. Design matrix encoded categorical vars.
- 10-fold CV on train set to choose lambda.min (0.000197037).
- Final model trained on full train set. Provided sparse feature selection.

**5.3.3 Generalized Linear Model**

- Gaussian with identity link on log_price.
- Transparent baseline. Residual Deviance = 595.22, AIC = 3720.2.

**5.3.4 XGBoost**

- Strong tabular data performance. Used:
  - eta = 0.1, max_depth = 6, objective = reg:squarederror

- Early stopping CV to select best rounds. Final model trained accordingly.
- Feature Importance: Key contributors include PC_capacity, longitude, room/property interactions, latitude, etc

**5.3.5 Gradient Boosted Machines**

- The log-transformed target was back-transformed prior to modeling.
- A model was trained using 1,000 trees, a learning rate (shrinkage) = 0.01, and depth = 4.
- 5-fold cross-validation was employed internally to determine the optimal number of boosting iterations.

**5.4 Model Evaluation Reported RMSE on original price scale (exp(log_price)) (See appendix)**

**5.5 Final Model Justification, Final Choice: XGBoost**

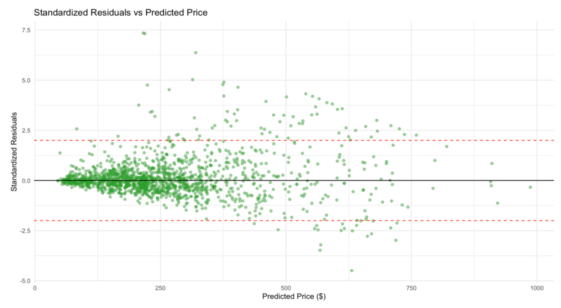
- Predictive performance: XGBoost had the lowest test MSE, our primary criterion.
- Flexibility: It captures non-linear relationships and high-order interactions automatically.
- Feature importance: Though interpretation is not required, it provides insight into what vars drive predictions.
- Robustness: Through regularization and early stopping, overfitting was mitigated.

**6.1 Strengths of the Final Model**

- High Predictive Accuracy: XGBoost achieved the lowest RMSE on both validation and test sets, excelling in capturing non-linear data relationships.
- No Overfitting: Consistent validation ($107.8) and test ($107.71) RMSEs indicate low overfitting risk.
- Robust Feature Engineering: Applied log transformation, PCA on capacity variables, and domain-specific features.

**6.2 Model Diagnostics & Validation:**

- Residual Analysis: Standardized residuals scatter around zero, indicating no major model misspecification
- 5-Fold Cross-Validation with Early Stopping: Prevented overfitting.
- Mild Heteroscedasticity: Observed for high-priced listings (> $500), expected in pricing contexts.



**6.3 Model Limitations**

- Black-Box Nature: XGBoost lacks transparency, limiting its interpretability for stakeholders needing explainable models.
- Heteroscedasticity: Increased residual variance at higher price points reduces precision for luxury listings.
- Lack of Temporal Features: Seasonal and event-based pricing patterns were not captured, likely affecting real-world Airbnb pricing.
- No Segmentation: The model assumes a global pricing strategy, missing potential segmentation (e.g., business vs. leisure, luxury vs. budget).

**7.1 Improve the Best Model**

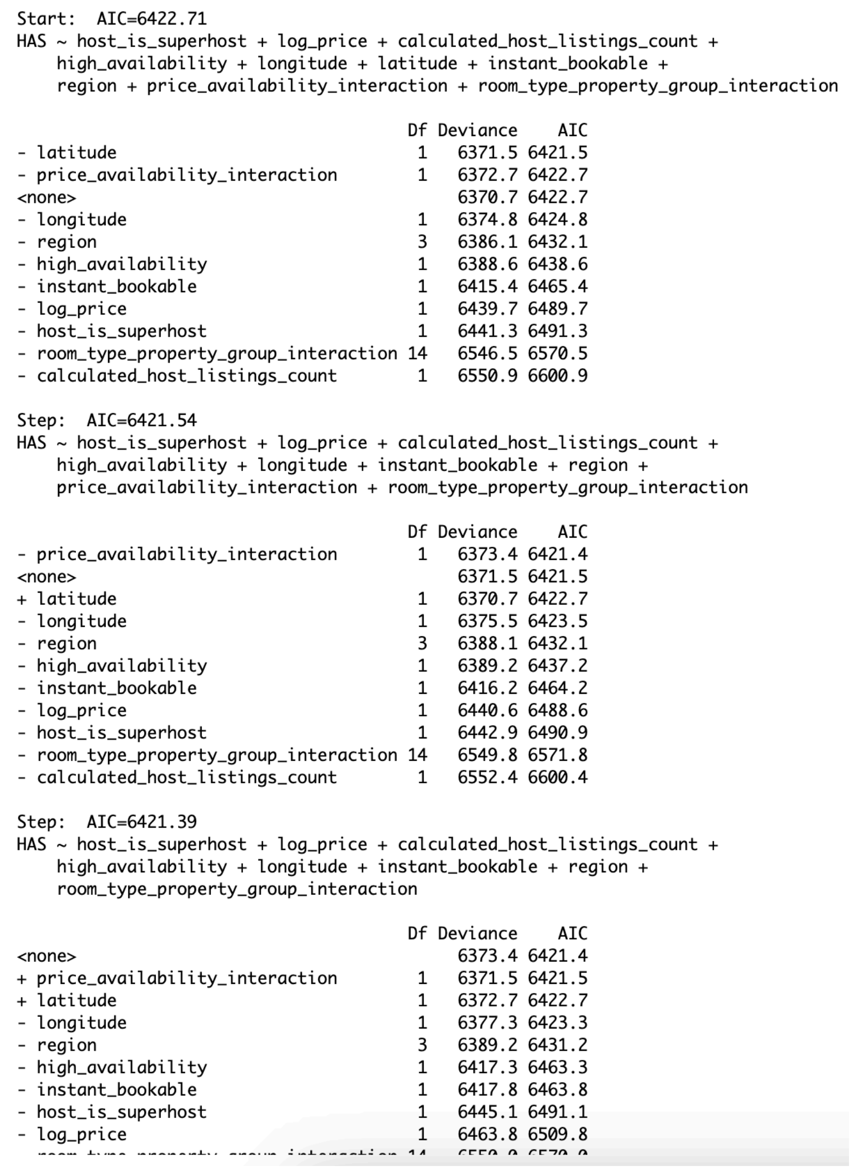
- **Expand Cross-Validation and Hyperparameter Tuning**: Use 10-fold cross-validation and a wider grid to improve stability and unlock better parameter combinations. Bayesian optimization could enhance tuning efficiency.
- **Implement Ensemble Stacking**: Combine XGBoost with Random Forest or GBM to leverage each model's strengths, reducing bias and variance for better performance.
- **Enhance Feature Selection and Dimensionality Reduction**: Extend PCA and L1 regularization to reduce overfitting by eliminating weak or collinear predictors.
- **Engineer Nonlinear Transformations**: Add splines or polynomial features for continuous predictors (e.g., latitude, host_experience_years) to improve signal representation and interpretability.
- **Deepen Residual Analysis and Subgroup Diagnostics**: Explore residuals by subgroups (e.g., region, room type) and error distribution plots to identify misfit segments and guide further feature engineering.
- **Incorporate Clustering for Latent Segments**: Use clustering (e.g., K-means, hierarchical) to define submarkets, allowing the model to adapt to distinct market segments for improved predictions.

**8.1** Final XGBoost model and preprocessing pipeline were applied to the Q8 test set to generate predictions.

**Appendix**

Model Summaries
2.5

Logistic Regression Stepwise Selection



```
Start:  AIC=6422.71
HAS ~ host_is_superhost + log_price + calculated_host_listings_count + 
    high_availability + longitude + latitude + instant_bookable + 
    region + price_availability_interaction + room_type_property_group_interaction

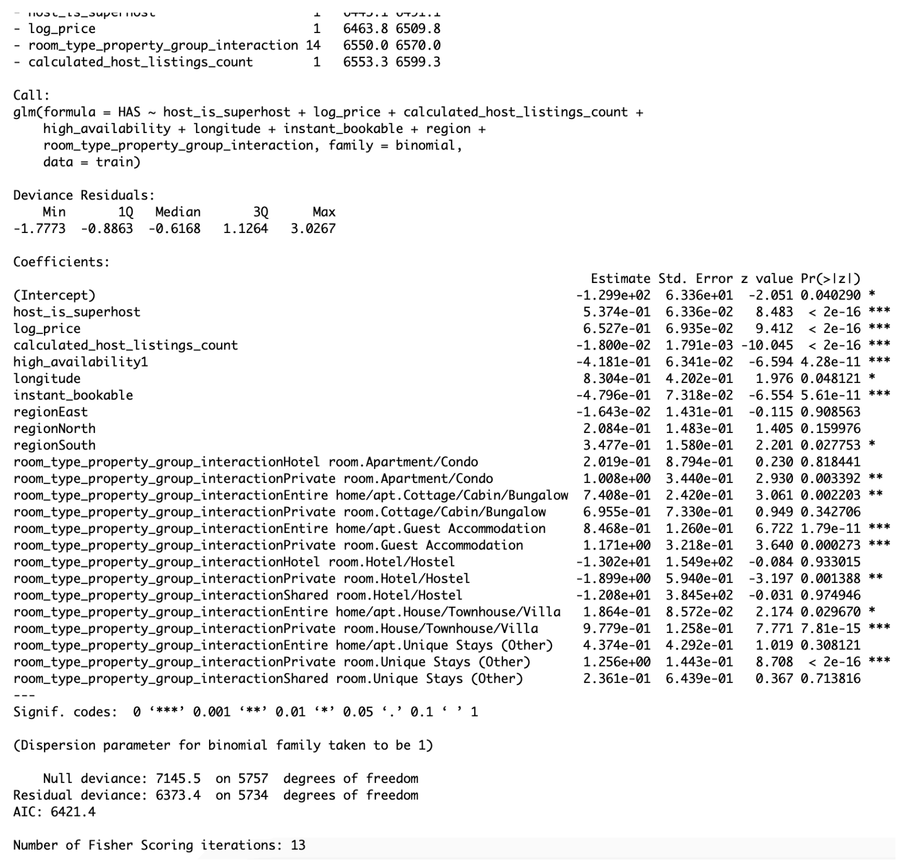
                                       Df Deviance    AIC
- latitude                              1   6371.5 6421.5
- price_availability_interaction        1   6372.7 6422.7
<none>                                      6370.7 6422.7
- longitude                             1   6374.8 6424.8
- region                                3   6386.1 6432.1
- high_availability                     1   6388.6 6438.6
- instant_bookable                      1   6415.4 6465.4
- log_price                             1   6439.7 6489.7
- host_is_superhost                     1   6441.3 6491.3
- room_type_property_group_interaction 14   6546.5 6570.5
- calculated_host_listings_count        1   6550.9 6600.9

Step:  AIC=6421.54
HAS ~ host_is_superhost + log_price + calculated_host_listings_count + 
    high_availability + longitude + instant_bookable + region + 
    price_availability_interaction + room_type_property_group_interaction

                                       Df Deviance    AIC
- price_availability_interaction        1   6373.4 6421.4
<none>                                      6371.5 6421.5
+ latitude                              1   6370.7 6422.7
- longitude                             1   6375.5 6423.5
- region                                3   6388.1 6432.1
- high_availability                     1   6389.2 6437.2
- instant_bookable                      1   6416.2 6464.2
- log_price                             1   6440.6 6488.6
- host_is_superhost                     1   6442.9 6490.9
- room_type_property_group_interaction 14   6549.8 6571.8
- calculated_host_listings_count        1   6552.4 6600.4

Step:  AIC=6421.39
HAS ~ host_is_superhost + log_price + calculated_host_listings_count + 
    high_availability + longitude + instant_bookable + region + 
    room_type_property_group_interaction

                                       Df Deviance    AIC
<none>                                      6373.4 6421.4
+ price_availability_interaction        1   6371.5 6421.5
+ latitude                              1   6372.7 6422.7
- longitude                             1   6377.3 6423.3
- region                                3   6389.2 6431.2
- high_availability                     1   6417.3 6463.3
- instant_bookable                      1   6417.8 6463.8
- host_is_superhost                     1   6445.1 6491.1
- log_price                             1   6463.8 6509.8
- room_type_property_group_interaction 14   6550.0 6570.0
```



```
- host_is_superhost                     1   6445.1 6491.1
- log_price                             1   6463.8 6509.8
- room_type_property_group_interaction 14   6550.0 6570.0
- calculated_host_listings_count        1   6553.3 6599.3

Call:
glm(formula = HAS ~ host_is_superhost + log_price + calculated_host_listings_count + 
    high_availability + longitude + instant_bookable + region + 
    room_type_property_group_interaction, family = binomial, 
    data = train)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-1.7773  -0.8863  -0.6168   1.1264   3.0267  

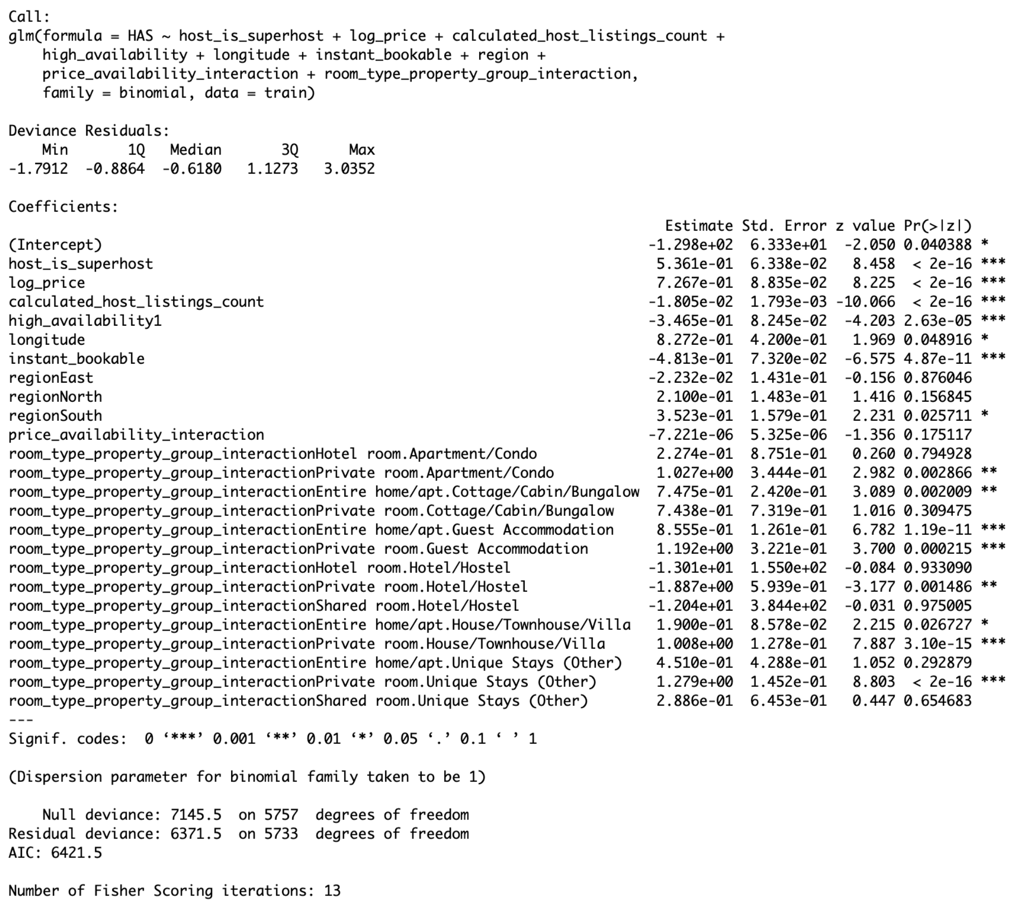
Coefficients:
                                                                          Estimate Std. Error z value Pr(>|z|)    
(Intercept)                                                             -1.299e+02  6.336e+01  -2.051 0.040290 *  
host_is_superhost                                                        5.374e-01  6.336e-02   8.483  < 2e-16 ***
log_price                                                                6.527e-01  6.935e-02   9.412  < 2e-16 ***
calculated_host_listings_count                                          -1.800e-02  1.791e-03 -10.045  < 2e-16 ***
high_availability1                                                      -4.181e-01  6.341e-02  -6.594 4.28e-11 ***
longitude                                                                8.304e-01  4.202e-01   1.976 0.048121 *  
instant_bookable                                                        -4.796e-01  7.318e-02  -6.554 5.61e-11 ***
regionEast                                                              -1.643e-02  1.431e-01  -0.115 0.908563    
regionNorth                                                              2.084e-01  1.483e-01   1.405 0.159976    
regionSouth                                                              3.477e-01  1.580e-01   2.201 0.027753 *  
room_type_property_group_interactionHotel room.Apartment/Condo           2.019e-01  8.794e-01   0.230 0.818441    
room_type_property_group_interactionPrivate room.Apartment/Condo         1.008e+00  3.440e-01   2.930 0.003392 ** 
room_type_property_group_interactionEntire home/apt.Cottage/Cabin/Bungalow  7.408e-01  2.420e-01   3.061 0.002203 ** 
room_type_property_group_interactionPrivate room.Cottage/Cabin/Bungalow   6.955e-01  7.330e-01   0.949 0.342706    
room_type_property_group_interactionEntire home/apt.Guest Accommodation   8.468e-01  1.260e-01   6.722 1.79e-11 ***
room_type_property_group_interactionPrivate room.Guest Accommodation     1.171e+00  3.218e-01   3.640 0.000273 ***
room_type_property_group_interactionHotel room.Hotel/Hostel             -1.302e+01  1.549e+02  -0.084 0.933015    
room_type_property_group_interactionPrivate room.Hotel/Hostel           -1.899e+00  5.940e-01  -3.197 0.001388 ** 
room_type_property_group_interactionShared room.Hotel/Hostel            -1.208e+01  3.845e+02  -0.031 0.974946    
room_type_property_group_interactionEntire home/apt.House/Townhouse/Villa  1.864e-01  8.572e-02   2.174 0.029670 *  
room_type_property_group_interactionPrivate room.House/Townhouse/Villa   9.779e-01  1.258e-01   7.771 7.81e-15 ***
room_type_property_group_interactionEntire home/apt.Unique Stays (Other)  4.374e-01  4.292e-01   1.019 0.308121    
room_type_property_group_interactionPrivate room.Unique Stays (Other)    1.256e+00  1.443e-01   8.708  < 2e-16 ***
room_type_property_group_interactionShared room.Unique Stays (Other)     2.361e-01  6.439e-01   0.367 0.713816    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 7145.5  on 5757  degrees of freedom
Residual deviance: 6373.4  on 5734  degrees of freedom
AIC: 6421.4

Number of Fisher Scoring iterations: 13
```

## Final Logistic regression summary



```
Call:
glm(formula = HAS ~ host_is_superhost + log_price + calculated_host_listings_count + 
    high_availability + longitude + instant_bookable + region + 
    price_availability_interaction + room_type_property_group_interaction, 
    family = binomial, data = train)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-1.7912  -0.8864  -0.6180   1.1273   3.0352  

Coefficients:
                                                                          Estimate Std. Error z value Pr(>|z|)    
(Intercept)                                                             -1.298e+02  6.333e+01  -2.050 0.040388 *  
host_is_superhost                                                        5.361e-01  6.338e-02   8.458  < 2e-16 ***
log_price                                                                7.267e-01  8.835e-02   8.225  < 2e-16 ***
calculated_host_listings_count                                          -1.805e-02  1.793e-03 -10.066  < 2e-16 ***
high_availability1                                                      -3.465e-01  8.245e-02  -4.203 2.63e-05 ***
longitude                                                                8.272e-01  4.200e-01   1.969 0.048916 *  
instant_bookable                                                        -4.813e-01  7.320e-02  -6.575 4.87e-11 ***
regionEast                                                              -2.232e-02  1.431e-01  -0.156 0.876046    
regionNorth                                                              2.100e-01  1.483e-01   1.416 0.156845    
regionSouth                                                              3.523e-01  1.579e-01   2.231 0.025711 *  
price_availability_interaction                                          -7.221e-06  5.325e-06  -1.356 0.175117    
room_type_property_group_interactionHotel room.Apartment/Condo           2.274e-01  8.751e-01   0.260 0.794928    
room_type_property_group_interactionPrivate room.Apartment/Condo         1.027e+00  3.444e-01   2.982 0.002866 ** 
room_type_property_group_interactionEntire home/apt.Cottage/Cabin/Bungalow  7.475e-01  2.420e-01   3.089 0.002009 ** 
room_type_property_group_interactionPrivate room.Cottage/Cabin/Bungalow   7.438e-01  7.319e-01   1.016 0.309475    
room_type_property_group_interactionEntire home/apt.Guest Accommodation   8.555e-01  1.261e-01   6.782 1.19e-11 ***
room_type_property_group_interactionPrivate room.Guest Accommodation     1.192e+00  3.221e-01   3.700 0.000215 ***
room_type_property_group_interactionHotel room.Hotel/Hostel             -1.301e+01  1.550e+02  -0.084 0.933090    
room_type_property_group_interactionPrivate room.Hotel/Hostel           -1.887e+00  5.939e-01  -3.177 0.001486 ** 
room_type_property_group_interactionShared room.Hotel/Hostel            -1.204e+01  3.844e+02  -0.031 0.975005    
room_type_property_group_interactionEntire home/apt.House/Townhouse/Villa  1.900e-01  8.578e-02   2.215 0.026727 *  
room_type_property_group_interactionPrivate room.House/Townhouse/Villa   1.008e+00  1.278e-01   7.887 3.10e-15 ***
room_type_property_group_interactionEntire home/apt.Unique Stays (Other)  4.510e-01  4.288e-01   1.052 0.292879    
room_type_property_group_interactionPrivate room.Unique Stays (Other)    1.279e+00  1.452e-01   8.803  < 2e-16 ***
room_type_property_group_interactionShared room.Unique Stays (Other)     2.886e-01  6.453e-01   0.447 0.654683    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

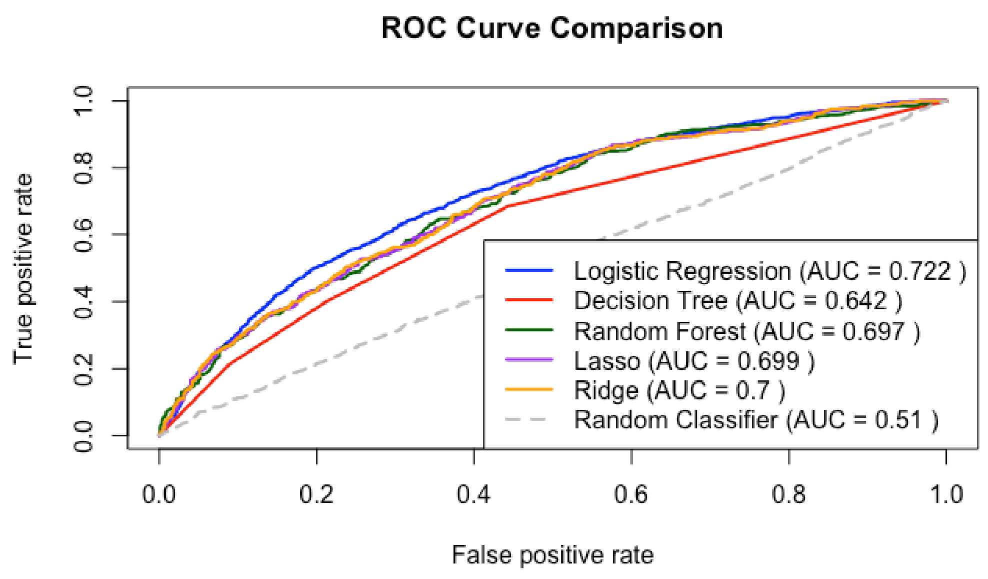
    Null deviance: 7145.5  on 5757  degrees of freedom
Residual deviance: 6371.5  on 5733  degrees of freedom
AIC: 6421.5

Number of Fisher Scoring iterations: 13
```

**2.5 Model Comparison**

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** | **AUC** | **Interpretability** |
|---|---|---|---|---|---|---|
| **Logistic Regression** | **0.713** | **0.601** | **0.601** | **0.601** | **0.722** | **High** |
| **Decision Tree** | 0.694 | 0.519 | 0.519 | 0.519 | 0.642 | Very High |
| **Random Forest** | 0.697 | 0.518 | 0.368 | 0.430 | 0.697 | Lower (black box) |
| **Lasso Regression** | 0.712 | 0.583 | 0.258 | 0.358 | 0.699 | High (sparse) |
| **Ridge Regression** | 0.717 | 0.609 | 0.249 | 0.353 | 0.700 | High (shrinkage) |

**ROC curves and AUC scores for all models**



**2.6 Model Summary (Logistic Regression)**

Model Fit

| Metric | Value |
|---|---|
| Null Deviance | 7145.5 |
| Residual Deviance | 6371.5 |
| AIC | 6421.5 |

**5.4 Model comparison**

| Model | Val MSE | Val RMSE ($) | Test MSE | Test RMSE | Comments |
|---|---|---|---|---|---|
| XGBoost | 11,621.14 | 107.80 | 11,602.35 | 107.71 | Best predictive performance |
| GBM | 12,125.03 | 110.11 | 12,487.32 | 111.75 | Strong performance, slightly behind XGB |
| Random Forest | 12,180.51 | 110.37 | 12,829.36 | 113.27 | Competitive model |
| GLM | 14,763.91 | 121.51 | 15,917.98 | 126.17 | Simple interpretable baseline |
| Lasso Reg. | 14,780.49 | 121.58 | 15,921.67 | 126.18 | Regularized, but weaker prediction |

**Code**

~~~
```{r}
# Load necessary libraries
library(randomForest)
library(MASS)
library(readxl)
library(glmnet)
library(caret)
library(tidyverse)
library(ggplot2)
library(corrplot)
library(dplyr)
library(psych)
library(ROCR)
library(rpart)
library(splines)
library(xgboost)
library(Matrix)
library(gbm)
library(writexl)
```

## Q1

```{r}
df <- read_excel("/Users/audreychang/Desktop/ACTL_ML/Textbook_R/HW/Assignment2/AirbnbSydneyV2.xlsx")
```

### Create avg_score from the 7 review score variables

```{r}
df$avg_score <- rowMeans(df[, c("review_scores_rating", "review_scores_accuracy",
                                "review_scores_cleanliness", "review_scores_checkin",
                                "review_scores_communication", "review_scores_location",
~~~

~~~~
                    "review_scores_value")], na.rm = TRUE)
```

```{r}
# Create binary HAS variable: 1 if avg_score > 4.9, else 0
df$HAS <- as.numeric(df$avg_score > 4.9)

```


```{r}
prop_HAS <- mean(df$HAS)
cat("Proportion of listings with HAS=1:", prop_HAS, "\n")
```

## Q2

### Data Preprocessing and Feature Engineering

### Clean and transform data

```{r}
# Convert character indicators to numeric
df <- df %>%
  mutate(
    host_is_superhost = ifelse(host_is_superhost == "t", 1, 0),
    host_identity_verified = ifelse(host_identity_verified == "t", 1, 0),
    host_has_profile_pic = ifelse(host_has_profile_pic == "t", 1, 0),
    instant_bookable = ifelse(instant_bookable == "t", 1, 0),
    host_experience_years = 2025 - lubridate::year(host_since),
    host_listings_count = as.numeric(host_listings_count),
    host_total_listings_count = as.numeric(host_total_listings_count),
    host_response_rate = as.numeric(host_response_rate),
    host_acceptance_rate = as.numeric(host_acceptance_rate)
  )

```


```{r}
# Handle missing values
# Replace "N/A" strings with actual NA values
na_string_cols <- sapply(df, function(x) {
  if (is.character(x) || is.factor(x)) {
    any(x == "N/A", na.rm = TRUE)
  } else {
    FALSE
  }
})
cols_to_fix <- names(na_string_cols[na_string_cols == TRUE])
~~~~

~~~
df[cols_to_fix] <- lapply(df[cols_to_fix], function(x) na_if(x, "N/A"))

# Check NA values in each column
na_count <- sapply(df, function(x) sum(is.na(x)))
na_count[na_count > 0]  # Show columns with missing values

# Calculate proportion of missing values
na_proportion <- sapply(df, function(x) mean(is.na(x)))

# Remove columns not needed
df <- df[, !(names(df) %in% c("bathrooms_text", "host_since"))]

# Replace missing values in categorical variables
df$host_response_time[is.na(df$host_response_time)] <- "Unknown"

# Impute host_response_rate with median
median_response_rate <- median(df$host_response_rate, na.rm = TRUE)
df$host_response_rate[is.na(df$host_response_rate)] <- median_response_rate

# Impute host_acceptance_rate with median
median_acceptance_rate <- median(df$host_acceptance_rate, na.rm = TRUE)
df$host_acceptance_rate[is.na(df$host_acceptance_rate)] <- median_acceptance_rate

# Impute host_is_superhost with 0 (Not Superhost) most common type
# Since it's already numeric (0 = not superhost, 1 = superhost)
df$host_is_superhost[is.na(df$host_is_superhost)] <- 0
```

### Feature Engineering: Create new variables

```{r}
# Create region categories
east <- c("Randwick", "Waverley", "Woollahra", "Sydney", "Marrickville",
        "Canada Bay", "Botany Bay", "Leichhardt", "Ashfield", "Burwood")
west <- c("Penrith", "Blacktown", "Fairfield", "Liverpool", "Campbelltown",
        "Auburn", "Bankstown", "Holroyd", "Strathfield", "Parramatta", "Canterbury")
north <- c("Ku-Ring-Gai", "North Sydney", "Willoughby", "Lane Cove", "Hornsby",
         "Ryde", "Mosman", "Hunters Hill", "The Hills Shire", "Warringah", "Manly", "Pittwater")
south <- c("Sutherland Shire", "Rockdale", "Hurstville", "Bankstown", "Camden", "City Of Kogarah")

# Create region classification
df$region <- case_when(
  df$neighbourhood %in% east ~ "East",
  df$neighbourhood %in% west ~ "West",
  df$neighbourhood %in% north ~ "North",
  df$neighbourhood %in% south ~ "South",
  TRUE ~ "Other"
)
~~~

```
# Categorize property types into broad groups
df$property_group <- ifelse(df$property_type %in% c(
   "Entire rental unit", "Entire condo", "Entire serviced apartment",
   "Entire loft", "Room in serviced apartment", "Room in aparthotel",
   "Private room in serviced apartment", "Private room in condo"),
    "Apartment/Condo",
      ifelse(df$property_type %in% c(
      "Entire home", "Entire vacation home", "Entire villa",
      "Entire townhouse", "Private room in home", "Private room in townhouse",
      "Private room in villa"),
       "House/Townhouse/Villa",
            ifelse(df$property_type %in% c(
         "Entire guesthouse", "Entire guest suite",
         "Private room in guesthouse", "Private room in guest suite",
         "Private room in bed and breakfast"),
          "Guest Accommodation",
                  ifelse(df$property_type %in% c(
            "Entire cottage", "Entire cabin", "Entire bungalow",
            "Private room in cabin", "Private room in cottage",
            "Private room in bungalow", "Private room in chalet", "Tiny home"),
             "Cottage/Cabin/Bungalow",
                        ifelse(df$property_type %in% c(
               "Room in hotel", "Room in boutique hotel", "Shared room in boutique hotel",
               "Shared room in hostel", "Private room in hostel",
               "Shared room in hotel", "Shared room in condo"),
                "Hotel/Hostel",
                              "Unique Stays (Other)"
            )
         )
      )
   ))

# Create high_availability indicator
df$high_availability <- ifelse(df$availability_60 > median(df$availability_60), 1, 0)

# Drop original variables after categorization
df <- df[, !(names(df) %in% c("neighbourhood", "property_type"))]




# Convert categorical variables to factors
df$room_type <- as.factor(df$room_type)
df$property_group <- as.factor(df$property_group)
df$region <- as.factor(df$region)
df$high_availability <- as.factor(df$high_availability)
```

~~~
```

### Exploratory Data Analysis

```{r}
# Create a clean dataset excluding review-based variables
df_clean <- df
columns_to_exclude <- c("review_scores_rating",
                    "review_scores_accuracy",
                    "review_scores_cleanliness",
                    "review_scores_checkin",
                    "review_scores_communication",
                    "review_scores_location",
                    "review_scores_value",
                    "avg_score")
df_clean <- df_clean[, !(names(df_clean) %in% columns_to_exclude)]

# Ensure HAS is numeric for correlation analysis
df_clean$HAS <- as.numeric(df_clean$HAS)

# Calculate correlation with HAS for numeric variables
numeric_vars <- df_clean[, sapply(df_clean, is.numeric)]
numeric_vars <- numeric_vars[, names(numeric_vars) != "HAS"]

cor_results <- sapply(numeric_vars, function(x) {
  if (is.numeric(x)) {
    cor(df_clean$HAS, x, use = "complete.obs")
  } else {
    NA
  }
})

# Remove NA results
cor_results <- cor_results[!is.na(cor_results)]

# Create correlation summary dataframe
cor_df <- data.frame(
  Variable = names(cor_results),
  CorrelationWithHAS = cor_results
) %>% arrange(desc(abs(CorrelationWithHAS)))

print(cor_df)

# Visualize correlations
ggplot(cor_df, aes(x = reorder(Variable, CorrelationWithHAS), y = CorrelationWithHAS)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = round(CorrelationWithHAS, 3)),
            hjust = ifelse(cor_df$CorrelationWithHAS >= 0, -0.1, 1.1),
~~~

```
          size = 3.5) +
  coord_flip() +
  labs(title = "Point-Biserial Correlation with HAS",
       x = "Variable",
       y = "Correlation") +
  theme(axis.text.x = element_text(face = "bold"),
       axis.text.y = element_text(face = "bold"),
       plot.title = element_text(hjust = 0.5, face = "bold"))
```

```{r}
# Convert HAS to factor for visualizations and modeling
df$HAS <- factor(df$HAS)

# Visualize categorical variables
categorical_vars <- c("region", "property_group", "host_response_time", "room_type", "high_availability")

# Loop through each categorical variable and create a visualization
for (var in categorical_vars) {
  # Create a summary table
  summary_table <- df %>%
    group_by(HAS, !!sym(var)) %>%
    summarise(count = n(), .groups = "drop")

  # Plot the data
  p <- ggplot(summary_table, aes_string(x = var, y = "count", fill = "HAS")) +
    geom_bar(stat = "identity", position = "stack") +
    labs(title = paste("Distribution of", var, "by HAS"),
         x = var, y = "Count") +
    theme_minimal() +
    scale_fill_manual(values = c("0" = "blue", "1" = "red"))  # Match fill to factor levels

  # Rotate x-axis labels for crowded variables
  if (var %in% c("property_group", "region", "host_response_time", "high_availability")) {
    p <- p + theme(axis.text.x = element_text(angle = 45, hjust = 1))
  }

  print(p)
}

```

### Feature Transformation and Interaction based on EDA

```{r}
# create transformations and interactions based on insights from EDA

# Set reference levels for categorical variables
```

~~~
# First check what levels exist
region_levels <- levels(df$region)
print(region_levels)

# Relevel with appropriate error handling
if("West" %in% region_levels) {
  df$region <- relevel(df$region, ref = "West")
} else {
  cat("Warning: 'West' not found in region levels. Available levels:", paste(region_levels, collapse=", "), "\n")
  # Use first level if "West" isn't available
  df$region <- relevel(df$region, ref = region_levels[1])
}

df$property_group <- relevel(df$property_group, ref = "Apartment/Condo")
df$room_type <- relevel(df$room_type, ref = "Entire home/apt")

# log transformation to handle skewness
df$log_price <- log(df$price + 1)


# Create interactions based on EDA insights
df$host_exp_superhost_interaction <- df$host_experience_years * df$host_is_superhost
df$price_availability_interaction <- df$price * df$availability_60
df$room_type_property_group_interaction <- interaction(df$room_type, df$property_group)

```

## Q2,3,4

### Model Building

```{r}
# First make sure HAS is a factor for classification
df$HAS <- factor(df$HAS)

# Split data into training and test sets
set.seed(123)
split <- sample(1:nrow(df), 0.75 * nrow(df))
train <- df[split, ]
test <- df[-split, ]

# Handle missing values in training data
# For numeric columns, replace NA with median
train <- train %>%
  mutate(across(where(is.numeric), ~ifelse(is.na(.), median(., na.rm = TRUE), .)))

# For categorical variables, replace NA with most common category
for(col in names(train)[sapply(train, is.factor)]) {
~~~

~~~~
  if(any(is.na(train[[col]]))) {
    most_common <- names(sort(table(train[[col]]), decreasing = TRUE))[1]
    train[[col]][is.na(train[[col]])] <- most_common
  }
}

# Apply same transformations to test set
test <- test %>%
  mutate(across(where(is.numeric), ~ifelse(is.na(.), median(., na.rm = TRUE), .)))

for(col in names(test)[sapply(test, is.factor)]) {
  if(any(is.na(test[[col]]))) {
    most_common <- names(sort(table(train[[col]]), decreasing = TRUE))[1]  # Use distribution from training data
    test[[col]][is.na(test[[col]])] <- most_common
  }
}

# Extract response variables
y_train <- train$HAS
y_test <- test$HAS
```

Train-Test Split

```{r}
# Split
set.seed(123)
split <- sample(1:nrow(df), 0.75 * nrow(df))
train <- df[split, ]
test <- df[-split, ]

# If filtering, do it here
train <- na.omit(train)
test <- na.omit(test)

# Then extract y
y_train <- train$HAS
y_test <- test$HAS

```

Logistic Regression (Baseline)

1.Stepwise Selection

```{r}
# Start withfull model
logit_full <- glm(HAS ~ host_is_superhost + log_price + calculated_host_listings_count +
~~~~

```
                  high_availability + longitude + latitude + instant_bookable +
                  region + price_availability_interaction + room_type_property_group_interaction,
                 family = binomial, data = train)

# Run stepwise selection
model_stepwise <- stepAIC(logit_full, direction = "both")
summary(model_stepwise)

```

~~~
```{r}
# Final Logistic regression model
logit_revised <- glm(HAS ~ host_is_superhost + log_price + calculated_host_listings_count +
                    high_availability + longitude + instant_bookable +
                    region + price_availability_interaction +
                    room_type_property_group_interaction,
                   family = binomial, data = train)

# View the summary of the updated model
summary(logit_revised)

~~~

~~~
```{r}
# Get predicted probabilities for the logistic model
pred_probs <- predict(logit_revised, type = "response")

# Convert probabilities to binary labels using a threshold of 0.5
pred_labels <- ifelse(pred_probs > 0.5, 1, 0)

# Actual labels (assuming your dependent variable is 'HAS')
true_labels <- train$HAS

# Confusion Matrix
cm <- table(Predicted = pred_labels, Actual = true_labels)
print(cm)

# Accuracy
accuracy <- sum(diag(cm)) / sum(cm)
cat("Accuracy:", accuracy, "\n")

# Precision
precision <- cm[2, 2] / sum(cm[2, ])  # TP / (TP + FP)
cat("Precision:", precision, "\n")

# Recall (Sensitivity)
recall <- cm[2, 2] / sum(cm[2, ])  # TP / (TP + FN)
cat("Recall:", recall, "\n")
~~~

~~~
# F1-score
f1 <- 2 * (precision * recall) / (precision + recall)
cat("F1-score:", f1, "\n")

# ROC Curve & AUC
pred_logit <- prediction(pred_probs, true_labels)
perf <- performance(pred_logit, measure = "tpr", x.measure = "fpr")
plot(perf, main = "ROC Curve", col = "blue", lwd = 2)
roc_auc <- performance(pred_logit, measure = "auc")@y.values[[1]]
abline(a = 0, b = 1, col = "gray", lty = 2)
cat("AUC:", roc_auc, "\n")
```
~~~

Decision Tree

~~~
```{r}
# Train the Decision Tree model
dt_model <- rpart(HAS ~ host_is_superhost + log_price + calculated_host_listings_count +
                   high_availability + longitude + instant_bookable +
                   region + price_availability_interaction +
                   room_type_property_group_interaction,
                  data = train, method = "class")

# Make predictions on the test set
pred_probs_dt <- predict(dt_model, test, type = "prob")[, 2]  # Get probabilities for the positive class
pred_labels_dt <- ifelse(pred_probs_dt > 0.5, 1, 0)  # Convert probabilities to binary labels

# Actual labels for the test set
true_labels_dt <- test$HAS

# Evaluate the model
# Confusion Matrix
cm_dt <- table(Predicted = pred_labels_dt, Actual = true_labels_dt)
print(cm_dt)

# Accuracy
accuracy_dt <- sum(diag(cm_dt)) / sum(cm_dt)
cat("Accuracy:", accuracy_dt, "\n")

# Precision
precision_dt <- cm_dt[2, 2] / sum(cm_dt[2, ])  # TP / (TP + FP)
cat("Precision:", precision_dt, "\n")

# Recall (Sensitivity)
recall_dt <- cm_dt[2, 2] / sum(cm_dt[2, ])  # TP / (TP + FN)
cat("Recall:", recall_dt, "\n")
~~~

~~~~
# F1-score
f1_dt <- 2 * (precision_dt * recall_dt) / (precision_dt + recall_dt)
cat("F1-score:", f1_dt, "\n")

# ROC Curve & AUC
pred_dt <- prediction(pred_probs_dt, true_labels_dt)
perf_dt <- performance(pred_dt, measure = "tpr", x.measure = "fpr")
plot(perf_dt, main = "ROC Curve - Decision Tree", col = "red", lwd = 2)
roc_auc_dt <- performance(pred_dt, measure = "auc")@y.values[[1]]
abline(a = 0, b = 1, col = "gray", lty = 2)
cat("AUC:", roc_auc_dt, "\n")
```

### Random Forest Model

```{r}
# Train Random Forest
set.seed(123)
rf_model <- randomForest(HAS ~ host_is_superhost + log_price + calculated_host_listings_count +
                           high_availability + longitude + instant_bookable + region +
                           price_availability_interaction + room_type_property_group_interaction,
                         data = train, importance = TRUE, ntree = 500)

# Predict probabilities and classes on the test set
rf_probs <- predict(rf_model, newdata = test, type = "prob")[, 2]  # probabilities for class 1
rf_preds <- ifelse(rf_probs > 0.5, 1, 0)

# Convert numeric predictions to factors for confusionMatrix function
rf_preds_factor <- factor(rf_preds, levels = c(0, 1))
test_HAS_factor <- factor(as.numeric(as.character(test$HAS)), levels = c(0, 1))

# Confusion Matrix & Basic Metrics
conf_rf <- confusionMatrix(rf_preds_factor, test_HAS_factor, positive = "1")
print(conf_rf)

# Extract evaluation metrics
accuracy_rf <- conf_rf$overall["Accuracy"]
precision_rf <- conf_rf$byClass["Pos Pred Value"]
recall_rf <- conf_rf$byClass["Sensitivity"]
f1_rf <- conf_rf$byClass["F1"]

cat("\n--- Evaluation Metrics: Random Forest ---\n")
cat("Accuracy :", round(accuracy_rf, 4), "\n")
cat("Precision:", round(precision_rf, 4), "\n")
cat("Recall   :", round(recall_rf, 4), "\n")
cat("F1-score :", round(f1_rf, 4), "\n")

# ROC Curve & AUC
~~~~

```
pred_rf <- prediction(rf_probs, as.numeric(as.character(test$HAS)))
perf_rf <- performance(pred_rf, "tpr", "fpr")
roc_auc_rf <- performance(pred_rf, "auc")@y.values[[1]]

# Plot ROC
plot(perf_rf, main = "ROC Curve - Random Forest", col = "darkgreen", lwd = 2)
abline(a = 0, b = 1, col = "gray", lty = 2)
cat("AUC      :", round(roc_auc_rf, 4), "\n")

# Variable importance plot
varImpPlot(rf_model, main = "Variable Importance - Random Forest")
```

Regularized Logistic Regression (Lasso/Ridge)

Prepare Data

```{r}
# --- Prepare data ---
# Build model matrix from predictors
x_train <- model.matrix(HAS ~ host_is_superhost + log_price + calculated_host_listings_count +
                        high_availability + longitude + instant_bookable +
                        region + price_availability_interaction +
                        room_type_property_group_interaction, data = train)[, -1]  # drop intercept

x_test <- model.matrix(HAS ~ host_is_superhost + log_price + calculated_host_listings_count +
                       high_availability + longitude + instant_bookable +
                       region + price_availability_interaction +
                       room_type_property_group_interaction, data = test)[, -1]  # drop intercept

```

Lasso Logistic Regression (L1)

```{r}
# Lasso (alpha = 1)
lasso_model <- cv.glmnet(x_train, y_train, alpha = 1, family = "binomial", type.measure = "class")

# Predict on test set
lasso_probs <- predict(lasso_model, newx = x_test, s = "lambda.min", type = "response")
lasso_preds <- ifelse(lasso_probs > 0.5, 1, 0)

# Evaluation
conf_lasso <- confusionMatrix(as.factor(lasso_preds), as.factor(y_test), positive = "1")
print(conf_lasso)

# Metrics
accuracy_lasso <- conf_lasso$overall["Accuracy"]
```

~~~
precision_lasso <- conf_lasso$byClass["Precision"]
recall_lasso <- conf_lasso$byClass["Recall"]
f1_lasso <- conf_lasso$byClass["F1"]

cat("\n--- Lasso Logistic Regression ---\n")
cat("Accuracy :", round(accuracy_lasso, 4), "\n")
cat("Precision:", round(precision_lasso, 4), "\n")
cat("Recall   :", round(recall_lasso, 4), "\n")
cat("F1-score :", round(f1_lasso, 4), "\n")

```
~~~

~~~
```{r}
# Check levels of the target variable
levels(y_train)
levels(y_test)
```
~~~

Ridge Logistic Regression (L2)

~~~
```{r}
# Ridge (alpha = 0)
ridge_model <- cv.glmnet(x_train, y_train, alpha = 0, family = "binomial", type.measure = "class")

# Predict on test set
ridge_probs <- predict(ridge_model, newx = x_test, s = "lambda.min", type = "response")
ridge_preds <- ifelse(ridge_probs > 0.5, 1, 0)

# Evaluation
conf_ridge <- confusionMatrix(as.factor(ridge_preds), as.factor(y_test), positive = "1")
print(conf_ridge)

# Metrics
accuracy_ridge <- conf_ridge$overall["Accuracy"]
precision_ridge <- conf_ridge$byClass["Precision"]
recall_ridge <- conf_ridge$byClass["Recall"]
f1_ridge <- conf_ridge$byClass["F1"]

cat("\n--- Ridge Logistic Regression ---\n")
cat("Accuracy :", round(accuracy_ridge, 4), "\n")
cat("Precision:", round(precision_ridge, 4), "\n")
cat("Recall   :", round(recall_ridge, 4), "\n")
cat("F1-score :", round(f1_ridge, 4), "\n")

```
~~~

ROC Curve & AUC for both

```{r}
# Lasso
pred_lasso <- prediction(lasso_probs, y_test)
perf_lasso <- performance(pred_lasso, "tpr", "fpr")
auc_lasso <- performance(pred_lasso, "auc")@y.values[[1]]

# Ridge
pred_ridge <- prediction(ridge_probs, y_test)
perf_ridge <- performance(pred_ridge, "tpr", "fpr")
auc_ridge <- performance(pred_ridge, "auc")@y.values[[1]]

# Plot ROC
plot(perf_lasso, main = "ROC Curve - Regularized Logistic Regression", col = "purple", lwd = 2)
plot(perf_ridge, add = TRUE, col = "orange", lwd = 2)
legend("bottomright",
       legend = c(
         paste("Lasso (AUC =", round(auc_lasso, 3), ")"),
         paste("Ridge (AUC =", round(auc_ridge, 3), ")")
       ),
       col = c("purple", "orange"), lwd = 2)

cat("Lasso AUC:", auc_lasso, "\n")
cat("Ridge AUC:", auc_ridge, "\n")

```

ROC comparison

```{r}
# --- Logistic Regression ROC ---
pred_logit <- prediction(pred_probs, true_labels)
perf_logit <- performance(pred_logit, measure = "tpr", x.measure = "fpr")
roc_auc_logit <- performance(pred_logit, measure = "auc")@y.values[[1]]

# --- Decision Tree ROC ---
pred_dt <- prediction(pred_probs_dt, true_labels_dt)
perf_dt <- performance(pred_dt, measure = "tpr", x.measure = "fpr")
roc_auc_dt <- performance(pred_dt, measure = "auc")@y.values[[1]]

# --- Random Forest ROC ---
pred_rf <- prediction(rf_probs, test$HAS)
perf_rf <- performance(pred_rf, measure = "tpr", x.measure = "fpr")
roc_auc_rf <- performance(pred_rf, measure = "auc")@y.values[[1]]

# --- Lasso ROC ---
pred_lasso <- prediction(lasso_probs, y_test)
perf_lasso <- performance(pred_lasso, "tpr", "fpr")
roc_auc_lasso <- performance(pred_lasso, "auc")@y.values[[1]]
```

```
# --- Ridge ROC ---
pred_ridge <- prediction(ridge_probs, y_test)
perf_ridge <- performance(pred_ridge, "tpr", "fpr")
roc_auc_ridge <- performance(pred_ridge, "auc")@y.values[[1]]

# --- Random Classifier (baseline) ---
set.seed(42)
random_probs <- runif(length(true_labels))  # generate random probabilities
pred_random <- prediction(random_probs, true_labels)
perf_random <- performance(pred_random, "tpr", "fpr")
roc_auc_random <- performance(pred_random, "auc")@y.values[[1]]

# --- Plot ROC curves together ---
plot(perf_logit, main = "ROC Curve Comparison", col = "blue", lwd = 2)
plot(perf_dt, add = TRUE, col = "red", lwd = 2)
plot(perf_rf, add = TRUE, col = "darkgreen", lwd = 2)
plot(perf_lasso, add = TRUE, col = "purple", lwd = 2)
plot(perf_ridge, add = TRUE, col = "orange", lwd = 2)
plot(perf_random, add = TRUE, col = "gray", lwd = 2, lty = 2)

# --- Add the legend ---
legend("bottomright",
       legend = c(
         paste("Logistic Regression (AUC =", round(roc_auc_logit, 3), ")"),
         paste("Decision Tree (AUC =", round(roc_auc_dt, 3), ")"),
         paste("Random Forest (AUC =", round(roc_auc_rf, 3), ")"),
         paste("Lasso (AUC =", round(roc_auc_lasso, 3), ")"),
         paste("Ridge (AUC =", round(roc_auc_ridge, 3), ")"),
         paste("Random Classifier (AUC =", round(roc_auc_random, 3), ")")
       ),
       col = c("blue", "red", "darkgreen", "purple", "orange", "gray"),
       lty = c(1, 1, 1, 1, 1, 2),
       lwd = 2)

# --- AUC output ---
cat("Logistic Regression AUC :", roc_auc_logit, "\n")
cat("Decision Tree AUC       :", roc_auc_dt, "\n")
cat("Random Forest AUC       :", roc_auc_rf, "\n")
cat("Lasso Logistic AUC      :", roc_auc_lasso, "\n")
cat("Ridge Logistic AUC      :", roc_auc_ridge, "\n")
cat("Random Classifier AUC   :", roc_auc_random, "\n")

```

Q3

```{r}
```

~~~
# --- Residual and Deviance Diagnostics for Final Logistic Regression Model ---

# Pearson residuals
pearson_resid <- residuals(logit_revised, type = "pearson")

# Deviance residuals
deviance_resid <- residuals(logit_revised, type = "deviance")

# Fitted values (predicted probabilities)
fitted_probs <- fitted(logit_revised)

# --- Plot: Deviance Residuals vs Fitted Probabilities ---

ggplot(data = data.frame(Fitted = fitted_probs, Deviance = deviance_resid), aes(x = Fitted, y = Deviance)) +
  geom_point(alpha = 0.5, color = "steelblue") +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  labs(
    title = "Deviance Residuals vs Fitted Probabilities",
    x = "Fitted Probabilities",
    y = "Deviance Residuals"
  ) +
  theme_minimal()

```


```{r}
# --- Influence Diagnostics ---
library(ggplot2)

# Cook's Distance
cooks_d <- cooks.distance(logit_revised)

# Leverage (hat) values
hat_values <- hatvalues(logit_revised)

# Combine diagnostics
influence_df <- data.frame(
  obs = 1:length(cooks_d),
  cook = cooks_d,
  hat = hat_values,
  deviance = residuals(logit_revised, type = "deviance")
)

# --- Plot Cook's Distance ---
ggplot(influence_df, aes(x = obs, y = cook)) +
  geom_bar(stat = "identity", fill = "firebrick", alpha = 0.7) +
  geom_hline(yintercept = 4 / nrow(influence_df), linetype = "dashed", color = "black") +
  labs(
~~~

~~~
    title = "Cook's Distance for Logistic Regression",
    x = "Observation Index",
    y = "Cook's Distance"
  ) +
  theme_minimal()

```

## Q5

```{r}
# Preprocessing and Feature Engineering
# --------------------------------------------------

# Drop review-related variables Keep avg_score to be the proxy of all review-score-related variables
review_vars <- c(
  "review_scores_rating",
  "review_scores_accuracy",
  "review_scores_cleanliness",
  "review_scores_checkin",
  "review_scores_communication",
  "review_scores_location",
  "review_scores_value"
)

# Create df_clean2 by removing review variables
df_clean2 <- df[, !(names(df) %in% review_vars)]

# Remove other unusable columns
df_clean2 <- df_clean2[, !(names(df_clean2) %in% c(
  "price",  # Remove original price as I'll use log_price
  "price_availability_interaction",
  "high_availability"
))]

# Feature Engineering: Calculate days between reviews
# This might introduce data leakage for new listings - only use if available at prediction time
df_clean2$days_between_reviews <- as.numeric(difftime(
  as.Date(df_clean2$last_review),
  as.Date(df_clean2$first_review),
  units = "days"
))

# Remove original date columns after feature extraction
df_clean2 <- df_clean2[, !(names(df_clean2) %in% c("first_review", "last_review"))]

# Handle missing values
# Replace "N/A" strings with actual NA
~~~

~~~r
na_string_cols <- sapply(df_clean2, function(x) {
  if (is.character(x) || is.factor(x)) {
    any(x == "N/A", na.rm = TRUE)
  } else {
    FALSE
  }
})

cols_to_fix <- names(na_string_cols[na_string_cols == TRUE])
df_clean2[cols_to_fix] <- lapply(df_clean2[cols_to_fix], function(x) na_if(x, "N/A"))

# Check for remaining NA values
na_count <- sapply(df_clean2, function(x) sum(is.na(x)))
na_count[na_count > 0]  # Show only columns with missing values
```

```{r}
# Select all numeric predictors for correlation analysis
numeric_vars <- names(df_clean2)[sapply(df_clean2, is.numeric)]
df_numeric <- df_clean2[numeric_vars]

# Calculate correlation with log_price
cor_with_price <- sapply(numeric_vars, function(x) {
  if(x != "log_price") {
    cor(df_clean2[[x]], df_clean2$log_price, use = "complete.obs")
  } else {
    NA
  }
})

# Create and sort correlation dataframe
cor_df <- data.frame(
  Predictor = names(cor_with_price),
  Correlation = cor_with_price
) %>%
  filter(!is.na(Correlation)) %>%  # Remove NA values
  arrange(desc(abs(Correlation)))  # Sort by absolute correlation for both pos/neg correlations

# Print the sorted correlations
print(cor_df)


# Plot correlation
ggplot(cor_df, aes(x = reorder(Predictor, Correlation), y = Correlation)) +
  geom_bar(stat = "identity", fill = "skyblue", color = "black") +
  coord_flip() +
  theme_minimal(base_size = 15) +
  labs(
~~~

~~~
    title = "Correlation of Numeric Predictors with Airbnb Log_Price",
    x = "Predictor",
    y = "Correlation with Log_Price"
  ) +
  theme(
    plot.title = element_text(size = 18, face = "bold", hjust = 0.5),
    axis.title = element_text(size = 14, face = "bold"),
    axis.text = element_text(size = 12, face = "bold")
  ) +
  geom_text(
    aes(label = round(Correlation, 2)),
    hjust = ifelse(cor_df$Correlation > 0, -0.2, 1.2),
    size = 4,
    fontface = "bold"
  )

```
~~~

Prepossessing data

**PCA on correlated subset**:

~~~
```{r}
# Select highly correlated "capacity" variables
capacity_vars <- df_clean2 %>% select(accommodates, beds, bedrooms, bathrooms)

# Perform PCA
pca_capacity <- prcomp(capacity_vars, center = TRUE, scale. = TRUE)

# Add first principal component back to data
df_clean2$PC_capacity <- pca_capacity$x[, 1]
#Drop original correlated variables
df_clean2 <- df_clean2 %>% select(-accommodates, -beds, -bedrooms, -bathrooms)
```
~~~

~~~
```{r}
colnames(df_clean2)
#Remove rows with missing values
df_clean2 <- na.omit(df_clean2)
```
~~~

~~~
```{r}
# Fit initial Random Forest model to get variable importance
rf_model <- randomForest(log_price ~ ., data = df_clean2, ntree = 500, importance = TRUE)

# Extract variable importance
importance_df <- as.data.frame(importance(rf_model))
importance_df$Variable <- rownames(importance_df)
~~~

~~~
# Sort by %IncMSE (most important at the top)
importance_sorted <- importance_df %>%
  arrange(desc(`%IncMSE`))

# Print the top variables by importance
print(importance_sorted)

# Plot variable importance
varImpPlot(rf_model, main = "Variable Importance (%IncMSE)")

```


```{r}
#Try several cutoffs:
mse_results <- c()
for (n in c(10, 15, 20)) {
  top_n_vars <- importance_sorted$Variable[1:n]
  df_sub <- df_clean2 %>% select(all_of(top_n_vars), log_price)

  # Fit model (Random Forest)
  rf_temp <- randomForest(log_price ~ ., data = df_sub, ntree = 500)
  pred_temp <- predict(rf_temp)
  mse <- mean((df_sub$log_price - pred_temp)^2)
  mse_results <- c(mse_results, mse)
}
print(mse_results)

```


**Create the reduced dataset** (`df_reduced`) using the selected top 20 predictors.

```{r}
# Extract top 20 predictor variable names from importance
top_20_vars <- importance_sorted$Variable[1:20]

# Filter top 20 variables to include only those that exist in df_clean2
top_20_vars<- top_20_vars[top_20_vars %in% names(df_clean2)]

# Create reduced dataset with the top predictors and target variable (log_price)
df_reduced <- df_clean2 %>% select(all_of(top_20_vars), log_price)

# View column names in the reduced dataset
cat("Variables in df_reduced:\n")
print(colnames(df_reduced))

```
~~~

### **Split this new reduced dataset**

-   **60/20/20 train-validation-test split**

```{r}
# First extract the test dataset:
set.seed(123)
n <- nrow(df_reduced)
test_index <- sample(1:n, size = 0.2 * n)
test_reduced <- df_reduced[test_index, ]
train_val <- df_reduced[-test_index, ]

# Then split the rest into validation and training sets:
val_index <- sample(1:nrow(train_val), size = 0.25 * nrow(train_val))# 25% of 80% = 20%
valid_reduced <- train_val[val_index, ]
train_reduced <- train_val[-val_index, ]
```

### 2. Random Forest (on log_price)

Tune Hyperparameters (mtry and ntree) Using the Validation Set

```{r}
# RMSE evaluation function
evaluate_model <- function(pred, actual) {
  mse <- mean((actual - pred)^2)
  rmse <- sqrt(mse)
  return(list(MSE = mse, RMSE = rmse))
}
```

Fit the Final Random Forest on the Full Training Set with Best Parameters

```{r}
# Manual tuning of mtry and ntree
mtry_values <- c(10, 15, 20)
ntree_values <- c(100, 300, 500)
tune_results <- expand.grid(mtry = mtry_values, ntree = ntree_values)
tune_results$MSE <- NA
tune_results$RMSE <- NA

for (i in 1:nrow(tune_results)) {
  rf_temp <- randomForest(log_price ~ ., data = train_reduced,
                          mtry = tune_results$mtry[i],
                          ntree = tune_results$ntree[i])
  pred_val <- predict(rf_temp, newdata = valid_reduced)
  eval <- evaluate_model(exp(pred_val), exp(valid_reduced$log_price))
  tune_results$MSE[i] <- eval$MSE
```

~~~
  tune_results$RMSE[i] <- eval$RMSE
}

tune_results <- tune_results[order(tune_results$RMSE), ]
print(tune_results)
best_combo <- tune_results[1, ]
```

```{r}
# Final RF model using best tuning parameters
rf_final <- randomForest(log_price ~ ., data = train_reduced,
                         mtry = best_combo$mtry,
                         ntree = best_combo$ntree)

# Validation set predictions and evaluation
pred_val_final <- predict(rf_final, newdata = valid_reduced)
eval_val_final <- evaluate_model(exp(pred_val_final), exp(valid_reduced$log_price))
val_mse <- mean((exp(pred_val_final) - exp(valid_reduced$log_price))^2)

cat("Validation RMSE: $", round(eval_val_final$RMSE, 2), "\n")
cat("Validation MSE:", round(val_mse, 2), "\n\n")

# Test set predictions and evaluation
pred_test_final <- predict(rf_final, newdata = test_reduced)
eval_test_final <- evaluate_model(exp(pred_test_final), exp(test_reduced$log_price))
test_mse <- mean((exp(pred_test_final) - exp(test_reduced$log_price))^2)

cat("Test RMSE: $", round(eval_test_final$RMSE, 2), "\n")
cat("Test MSE:", round(test_mse, 2), "\n")

```

**Lasso Regression (on log_price)**

Lasso Regression (`alpha = 1`)

```{r}
# Build model matrix once from full reduced dataset
x_full <- model.matrix(log_price ~ ., data = df_reduced)[, -1]
y_full <- df_reduced$log_price

# Perform consistent 60/20/20 split on full data
set.seed(123)
n <- nrow(df_reduced)
test_index <- sample(1:n, size = 0.2 * n)
train_val_index <- setdiff(1:n, test_index)

val_index <- sample(train_val_index, size = 0.25 * length(train_val_index))
~~~

~~~
train_index <- setdiff(train_val_index, val_index)

# Subset model matrix and target vectors
x_train <- x_full[train_index, ]
x_valid <- x_full[val_index, ]
x_test  <- x_full[test_index, ]

y_train <- y_full[train_index]
y_valid <- y_full[val_index]
y_test  <- y_full[test_index]

```

```{r}
# Cross-validation for best lambda
set.seed(123)
lasso_cv <- cv.glmnet(x_train, y_train, alpha = 1, standardize = TRUE)

# Plot CV curve
plot(lasso_cv)

# Fit final model
best_lambda_lasso <- lasso_cv$lambda.min
cat("Best lambda (Lasso):", best_lambda_lasso, "\n")

lasso_model <- glmnet(x_train, y_train, alpha = 1, lambda = best_lambda_lasso)

# Predict
pred_valid_lasso <- predict(lasso_model, s = best_lambda_lasso, newx = x_valid)
pred_test_lasso <- predict(lasso_model, s = best_lambda_lasso, newx = x_test)

# Evaluate
eval_val_lasso <- evaluate_model(exp(pred_valid_lasso), exp(y_valid))
eval_test_lasso <- evaluate_model(exp(pred_test_lasso), exp(y_test))

# Display both MSE and RMSE
cat("Lasso Validation MSE:", round(eval_val_lasso$MSE, 2), "\n")
cat("Lasso Validation RMSE: $", round(eval_val_lasso$RMSE, 2), "\n\n")

cat("Lasso Test MSE:", round(eval_test_lasso$MSE, 2), "\n")
cat("Lasso Test RMSE: $", round(eval_test_lasso$RMSE, 2), "\n")

# Selected predictors
lasso_coefs <- coef(lasso_model)
selected_predictors <- rownames(lasso_coefs)[which(lasso_coefs != 0)]
cat("\nSelected predictors by Lasso:\n")
print(selected_predictors)
~~~

~~~
```

## **GLM (Gaussian with identity link)**

prepossessing\

```{r}
# Convert percentage-like variables to numeric
percentage_vars <- c("host_acceptance_rate", "host_response_rate")

convert_percentages_to_numeric <- function(df) {
  for (var in percentage_vars) {
    if (var %in% colnames(df)) {
      df[[var]] <- as.numeric(as.character(df[[var]]))
    }
  }
  return(df)
}

train_reduced <- convert_percentages_to_numeric(train_reduced)
valid_reduced <- convert_percentages_to_numeric(valid_reduced)
test_reduced  <- convert_percentages_to_numeric(test_reduced)

# Remove factors with only one level
drop_single_level_factors <- function(df) {
  keep <- sapply(df, function(col) !(is.factor(col) && length(unique(col)) == 1))
  df[, keep]
}

train_reduced <- drop_single_level_factors(train_reduced)

# Align validation/test with training columns
valid_reduced <- valid_reduced[, colnames(train_reduced)]
test_reduced  <- test_reduced[, colnames(train_reduced)]

# Drop NAs
train_reduced <- na.omit(train_reduced)
valid_reduced <- na.omit(valid_reduced)
test_reduced  <- na.omit(test_reduced)


```

```{r}
# Fit GLM on log_price
glm_log_model <- glm(log_price ~ ., data = train_reduced, family = gaussian(link = "identity"))

# Predict on validation and test
~~~

~~~
pred_log_val <- predict(glm_log_model, newdata = valid_reduced)
pred_log_test <- predict(glm_log_model, newdata = test_reduced)

# Actual and predicted prices (back-transform)
actual_val_price <- exp(valid_reduced$log_price)
actual_test_price <- exp(test_reduced$log_price)

pred_val_price <- exp(pred_log_val)
pred_test_price <- exp(pred_log_test)

# MSE & RMSE
mse_price_val <- mean((actual_val_price - pred_val_price)^2)
mse_price_test <- mean((actual_test_price - pred_test_price)^2)

rmse_price_val <- sqrt(mse_price_val)
rmse_price_test <- sqrt(mse_price_test)

# Output summary results
glm_metrics <- list(
  Validation_MSE = round(mse_price_val, 2),
  Validation_RMSE = round(rmse_price_val, 2),
  Test_MSE = round(mse_price_test, 2),
  Test_RMSE = round(rmse_price_test, 2)
)

print(glm_metrics)



```

```{r}
summary(glm_log_model)
```

**XGBoost**

Prepare Data (Convert to DMatrix)

```{r}
# Drop target from features
X_train <- model.matrix(log_price ~ . -1, data = train_reduced)
y_train <- train_reduced$log_price

X_valid <- model.matrix(log_price ~ . -1, data = valid_reduced)
y_valid <- valid_reduced$log_price

X_test  <- model.matrix(log_price ~ . -1, data = test_reduced)
~~~

~~~
y_test  <- test_reduced$log_price

# Convert to DMatrix
dtrain <- xgb.DMatrix(data = X_train, label = y_train)
dvalid <- xgb.DMatrix(data = X_valid, label = y_valid)
dtest  <- xgb.DMatrix(data = X_test, label = y_test)

```
~~~

Define Parameters and Train

~~~
```{r}
# Hyperparameters
params <- list(
  objective = "reg:squarederror",
  eval_metric = "rmse",
  eta = 0.1,
  max_depth = 6
)

# Cross-validation to determine best nrounds
cv <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 500,
  nfold = 5,
  early_stopping_rounds = 10,
  verbose = 0
)

best_nrounds <- cv$best_iteration

# Final model training using best_nrounds
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = best_nrounds,
  watchlist = list(train = dtrain, eval = dvalid),
  early_stopping_rounds = 10,
  print_every_n = 20
)


# Save the model for later use (optional)
xgb.save(xgb_model, "final_xgb_model.model")
# Reload the model if needed
# xgb_model <- xgb.load("final_xgb_model.model")
```
~~~

Predict on Validation & Test Sets

~~~
```{r}
# Predict log prices
pred_valid_xgb <- predict(xgb_model, newdata = dvalid)
pred_test_xgb  <- predict(xgb_model, newdata = dtest)

# Convert back to price
pred_valid_price <- exp(pred_valid_xgb)
pred_test_price  <- exp(pred_test_xgb)

actual_valid_price <- exp(y_valid)
actual_test_price  <- exp(y_test)

# Calculate RMSE and MSE
rmse_xgb_val <- sqrt(mean((actual_valid_price - pred_valid_price)^2))
rmse_xgb_test <- sqrt(mean((actual_test_price - pred_test_price)^2))

mse_xgb_val <- mean((actual_valid_price - pred_valid_price)^2)
mse_xgb_test <- mean((actual_test_price - pred_test_price)^2)

# Output results
cat("XGBoost Validation RMSE: $", round(rmse_xgb_val, 2), "\n")
cat("XGBoost Validation MSE:  ", round(mse_xgb_val, 4), "\n\n")

cat("XGBoost Test RMSE: $", round(rmse_xgb_test, 2), "\n")
cat("XGBoost Test MSE:  ", round(mse_xgb_test, 4), "\n")


```
~~~

Feature Importance Plot

~~~
```{r}
# Feature importance plot
importance_matrix <- xgb.importance(model = xgb_model)
xgb.plot.importance(importance_matrix, top_n = 20)
```
~~~

Gradient Boosted Trees (GBM)

~~~
```{r}
# Prepare the data
# gbm requires the response to be in the original (not log) scale
train_gbm <- train_reduced
valid_gbm <- valid_reduced
test_gbm  <- test_reduced
~~~

```
train_gbm$price <- exp(train_gbm$log_price)
valid_gbm$price <- exp(valid_gbm$log_price)
test_gbm$price  <- exp(test_gbm$log_price)

# Remove the log_price column
train_gbm$log_price <- NULL
valid_gbm$log_price <- NULL
test_gbm$log_price  <- NULL

# Fit the GBM model

set.seed(123)
gbm_model <- gbm(
  formula = price ~ .,
  distribution = "gaussian",
  data = train_gbm,
  n.trees = 1000,
  interaction.depth = 4,
  shrinkage = 0.01,
  n.minobsinnode = 10,
  cv.folds = 5,
  verbose = FALSE
)

# Determine the best number of trees
best_iter <- gbm.perf(gbm_model, method = "cv")

# Predict
pred_valid_gbm <- predict(gbm_model, newdata = valid_gbm, n.trees = best_iter)
pred_test_gbm  <- predict(gbm_model, newdata = test_gbm, n.trees = best_iter)

actual_valid_gbm <- valid_gbm$price
actual_test_gbm  <- test_gbm$price

# Evaluate
rmse_gbm_val <- sqrt(mean((actual_valid_gbm - pred_valid_gbm)^2))
rmse_gbm_test <- sqrt(mean((actual_test_gbm - pred_test_gbm)^2))

mse_gbm_val <- mean((actual_valid_gbm - pred_valid_gbm)^2)
mse_gbm_test <- mean((actual_test_gbm - pred_test_gbm)^2)

cat("GBM Validation RMSE: $", round(rmse_gbm_val, 2), "\n")
cat("GBM Validation MSE:  ", round(mse_gbm_val, 4), "\n\n")

cat("GBM Test RMSE: $", round(rmse_gbm_test, 2), "\n")
cat("GBM Test MSE:  ", round(mse_gbm_test, 4), "\n")
```

~~~
```

## Q6

Diagnostics & Validation

```{r}
# Residuals on test set (actual - predicted)
residuals_test <- actual_test_price - pred_test_price

# Plot residuals vs predicted prices
library(ggplot2)

ggplot(data = data.frame(predicted = pred_test_price, residuals = residuals_test),
       aes(x = predicted, y = residuals)) +
  geom_point(alpha = 0.5, color = "#1f77b4") +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  labs(
    title = "Residuals vs Predicted Price (Test Set)",
    x = "Predicted Price ($)",
    y = "Residuals ($)"
  ) +
  theme_minimal()

```

```{r}
# Standardized residuals on the test set
standardized_residuals <- residuals_test / sd(residuals_test)

# Plot standardized residuals to spot potential outliers
library(ggplot2)

ggplot(data = data.frame(predicted = pred_test_price, std_resid = standardized_residuals),
       aes(x = predicted, y = std_resid)) +
  geom_point(alpha = 0.5, color = "#2ca02c") +
  geom_hline(yintercept = c(-2, 0, 2), linetype = c("dashed", "solid", "dashed"), color = c("red", "black", "red")) +
  labs(
    title = "Standardized Residuals vs Predicted Price",
    x = "Predicted Price ($)",
    y = "Standardized Residuals"
  ) +
  theme_minimal()
```

Q7 in the report

Question 8: Final prediction on test_data using trained XGBoost model
~~~

~~~
```{r}
test_data <- read_excel("/Users/audreychang/Desktop/ACTL_ML/Textbook_R/HW/Assignment2/AirbnbTest (1).xlsx")
```
~~~

Preprocess Test Data

~~~
```{r}
#days_between_reviews
test_data$days_between_reviews <- as.numeric(as.Date(test_data$last_review) - as.Date(test_data$first_review))
#drop first_review, last_review
test_data <- subset(test_data, select = -c(first_review, last_review))

#only run sucessfully when u have the original file

```
~~~

~~~
```{r}
#host_experience_years
test_data$host_experience_years <- 2025 - as.numeric(format(as.Date(test_data$host_since), "%Y"))
#drop host_since
test_data <- subset(test_data, select = -c(host_since))
```
~~~

~~~
```{r}
#avg_score
test_data$avg_score <- rowMeans(test_data[, c("review_scores_rating", "review_scores_accuracy",
"review_scores_cleanliness", "review_scores_checkin","review_scores_communication", "review_scores_location",
"review_scores_value")], na.rm = TRUE)
#drop review_related variables
test_data <- test_data %>%
  select(-review_scores_rating,
         -review_scores_accuracy,
         -review_scores_cleanliness,
         -review_scores_checkin,
         -review_scores_communication,
         -review_scores_location,
         -review_scores_value)
```
~~~

~~~
```{r}
#Convert 't'/'f' to binary (0/1)
binary_vars <- c("host_is_superhost", "host_identity_verified", "host_has_profile_pic", "instant_bookable")
for (var in binary_vars) {
  test_data[[var]] <- ifelse(test_data[[var]] == "t", 1, 0)
}
```
~~~

~~~
```{r}
#numeric conversions
test_data <- test_data %>%
  mutate(
    host_listings_count = as.numeric(host_listings_count),
    host_total_listings_count = as.numeric(host_total_listings_count),
    host_response_rate = as.numeric(host_response_rate),
    host_acceptance_rate = as.numeric(host_acceptance_rate)
  )

```

Handling NA

```{r}
#Identify columns in test_data where "N/A" appears (as a string)
na_string_cols_test <- sapply(test_data, function(x) {
  if (is.character(x) || is.factor(x)) {
    any(x == "N/A", na.rm = TRUE)
  } else {
    FALSE
  }
})

# List of column names that contain "N/A" strings
cols_to_fix_test <- names(na_string_cols_test[na_string_cols_test == TRUE])

#Replace "N/A" with real NA only in those columns
test_data[cols_to_fix_test] <- lapply(test_data[cols_to_fix_test], function(x) na_if(x, "N/A"))


#Count NA values per column
na_count_test <- sapply(test_data, function(x) sum(is.na(x)))
na_count_test <- na_count_test[na_count_test > 0]  # Only show columns with missing
na_count_test
```

remove unused column

```{r}
test_data <- test_data %>%
  select(-bathrooms_text)
```

```{r}
# Convert percentage columns to numeric
test_data <- test_data %>%
  mutate(
~~~

~~~
    host_response_rate = as.numeric(host_response_rate),
    host_acceptance_rate = as.numeric(host_acceptance_rate)
  )

# Impute missing values in host_response_rate with its median
test_data$host_response_rate[is.na(test_data$host_response_rate)] <- median(test_data$host_response_rate, na.rm =
TRUE)

# Impute missing values in host_acceptance_rate with its median
test_data$host_acceptance_rate[is.na(test_data$host_acceptance_rate)] <- median(test_data$host_acceptance_rate, na.rm
= TRUE)

```

```{r}
# Replace missing values for host_response_rate and host_acceptance_rate host_response_time, host_is_superhost with
"Unknown"

test_data$host_response_time[is.na(test_data$host_response_time)] <- "Unknown"
test_data$host_is_superhost[is.na(test_data$host_is_superhost)] <- "Unknown"
```

```{r}
# Check if there are any NA values in the entire test_data
anyNA(test_data)
```

```{r}
#Region Classification (based on neighbourhood)
test_data$region <- case_when(
  test_data$neighbourhood %in% east ~ "East",
  test_data$neighbourhood %in% west ~ "West",
  test_data$neighbourhood %in% north ~ "North",
  test_data$neighbourhood %in% south ~ "South",
  TRUE ~ "Other"
)

```


```{r}
#Property Group Categorization
test_data$property_group <- ifelse(test_data$property_type %in% c(
  "Entire rental unit", "Entire condo", "Entire serviced apartment",
  "Entire loft", "Room in serviced apartment", "Room in aparthotel",
  "Private room in serviced apartment", "Private room in condo"),
  "Apartment/Condo",

  ifelse(test_data$property_type %in% c(
~~~

~~~~
    "Entire home", "Entire vacation home", "Entire villa",
    "Entire townhouse", "Private room in home", "Private room in townhouse",
    "Private room in villa"),
    "House/Townhouse/Villa",

    ifelse(test_data$property_type %in% c(
      "Entire guesthouse", "Entire guest suite",
      "Private room in guesthouse", "Private room in guest suite",
      "Private room in bed and breakfast"),
      "Guest Accommodation",

      ifelse(test_data$property_type %in% c(
        "Entire cottage", "Entire cabin", "Entire bungalow",
        "Private room in cabin", "Private room in cottage",
        "Private room in bungalow", "Private room in chalet", "Tiny home"),
        "Cottage/Cabin/Bungalow",

        ifelse(test_data$property_type %in% c(
          "Room in hotel", "Room in boutique hotel", "Shared room in boutique hotel",
          "Shared room in hostel", "Private room in hostel",
          "Shared room in hotel", "Shared room in condo"),
          "Hotel/Hostel",

          "Unique Stays (Other)"
        )
      )
    )
  )
)

```

```{r}
df$region <- relevel(as.factor(df$region), ref = "West")  # Set West as the reference category
df$property_group <- relevel(as.factor(df$property_group), ref = "Apartment/Condo")  # Relevel property_group
df$room_type <- relevel(as.factor(df$room_type), ref = "Entire home/apt")  # Relevel room_type

```

```{r}
#room_type × property_group interaction
test_data$room_type_property_group_interaction <- interaction(test_data$room_type, test_data$property_group)
```

```{r}
#PCA
# Select the same capacity variables from the test set
capacity_vars_test <- test_data %>% select(accommodates, beds, bedrooms, bathrooms)
~~~~

```
# Apply the PCA transformation using the training PCA object
# Use predict() to project the test data onto the training PCA space
pca_capacity_test <- predict(pca_capacity, newdata = capacity_vars_test)

# Add the first principal component to the test data
test_data$PC_capacity <- pca_capacity_test[, 1]

# Drop the original capacity variables from the test set
test_data <- test_data %>% select(-accommodates, -beds, -bedrooms, -bathrooms)

```

Apply the Model to the Test Set

```{r}
colnames(test_data)
```

3\. Make predictions using the trained model

```{r}
# Assuming you have the list of top predictor variables from Question 5
test_data_reduced <- test_data %>% select(all_of(top_20_vars))  # Reduced dataset with top predictors

# Load the trained model from Question 5 (XGBoost in this case)
xgb_model <- xgb.load("final_xgb_model.model")

# Convert test data to DMatrix (for XGBoost)
dtest <- xgb.DMatrix(data = model.matrix(~ . -1, data = test_data_reduced))  # Ensure you remove the intercept term

# Predict log prices using the model
pred_log_price <- predict(xgb_model, newdata = dtest)

# Convert log price predictions back to the price scale (exponentiate)
test_data$price_prediction <- exp(pred_log_price)  # Update the price_prediction column

# Optionally, check the first few predictions
head(test_data$price_prediction)


# Save the predictions to an Excel file
write_xlsx(test_data, "/Users/audreychang/Desktop/ACTL_ML/Textbook_R/HW/Assignment2/AirbnbTest (1).xlsx")

```

## Generative AI usage

OpenAI.(2025). *ChatGPT* (March 27 version) [Large language model]. https://chat.openai.com/chat

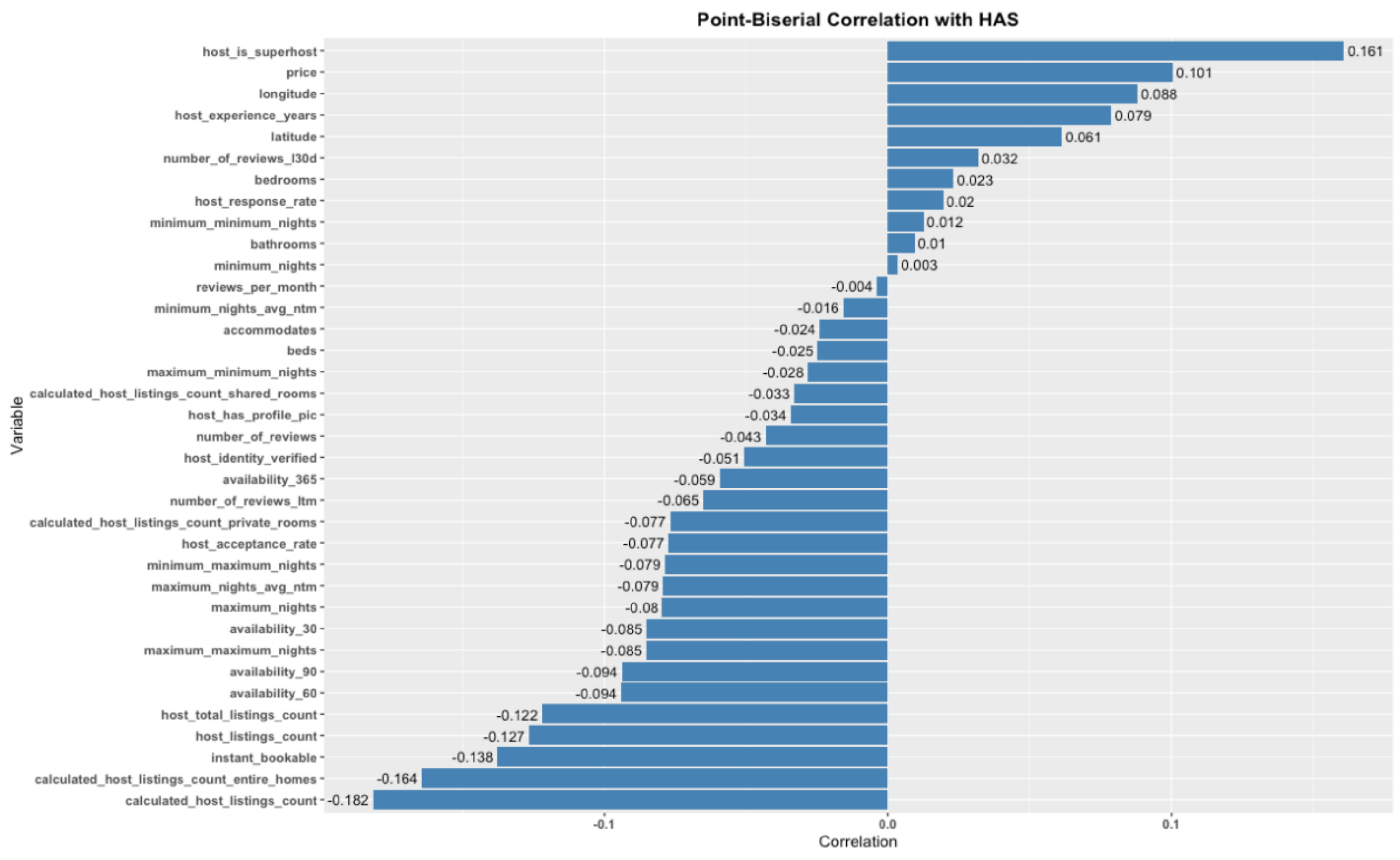
**Purpose of Use:**

ChatGPT was used to support various aspects of the assignment, including:

- **Editing and Refinement:**
    - Improved clarity, flow, and conciseness of report sections.
    - Reduced redundancy in model descriptions and interpretations.
- **Planning and Structure:**
    - Helped organize report sections logically.
    - Suggested concise formats for presenting model results and diagnostics.
- **Concept Clarification and Idea Generation:**
    - Explained statistical concepts such as:
        - Cook's Distance (influence diagnostics)
        - XGBoost mechanisms and hyperparameters
        - Gradient Boosted Trees and ensemble methods
        - Residual diagnostics in logistic regression and XGBoost
    - Suggested realistic improvement steps for the predictive model.
- **Prompt Examples Used:**
    - "Make this paragraph more concise but preserve meaning."
    - "Explain how to interpret Cook's Distance."
    - "What are the pros and cons of XGBoost in model interpretability?"
    - "Suggest ways to improve a regression model using advanced techniques."
    - "Refine this model limitation section to save words."

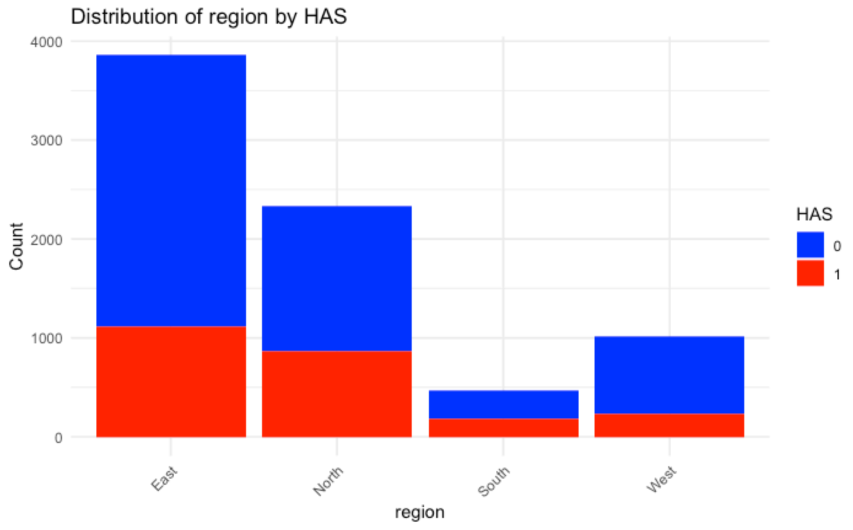
**Others:**

## 2.3 Variable Selection (Correlation plot)

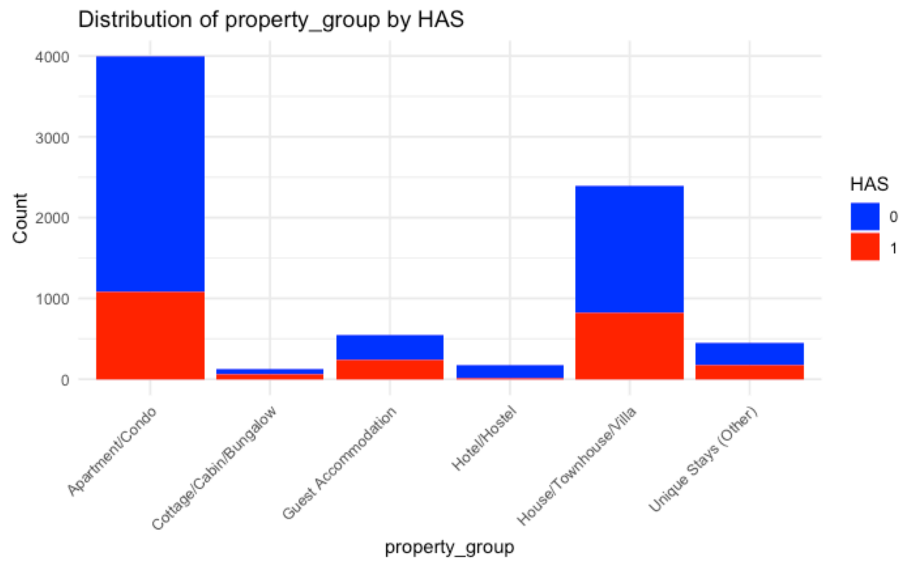


## 2.3 Data Visualization

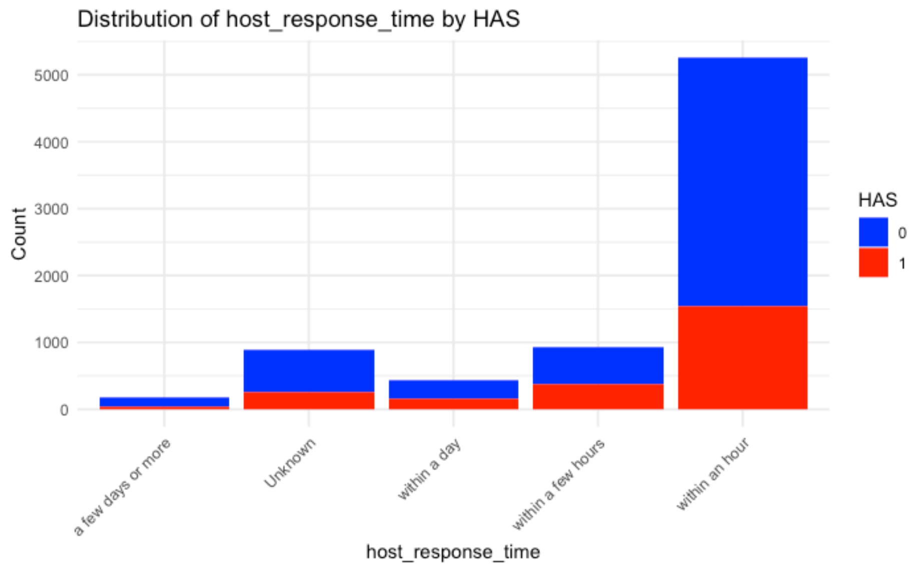
**"Region"**



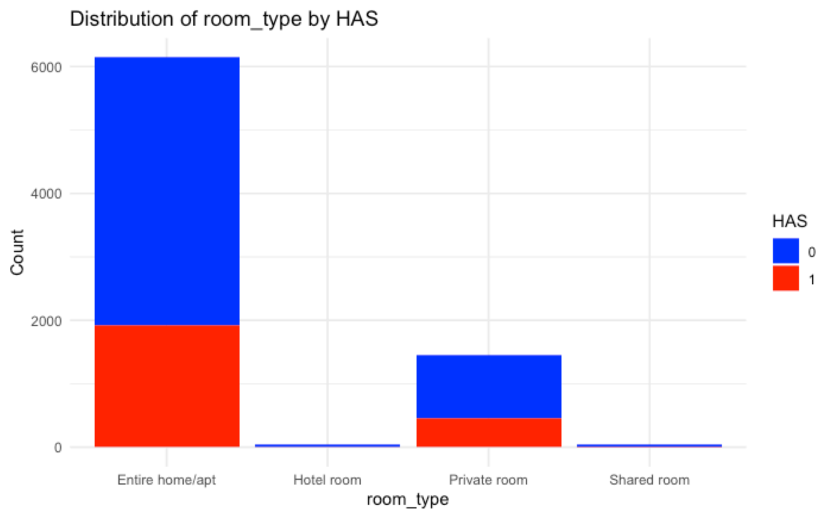
**"property_group"**



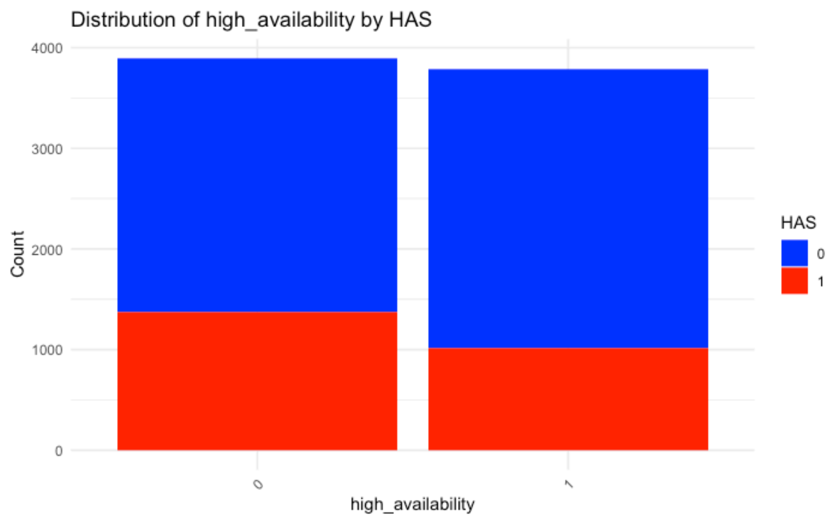
**"Host_response_time"**



**"room_type"**



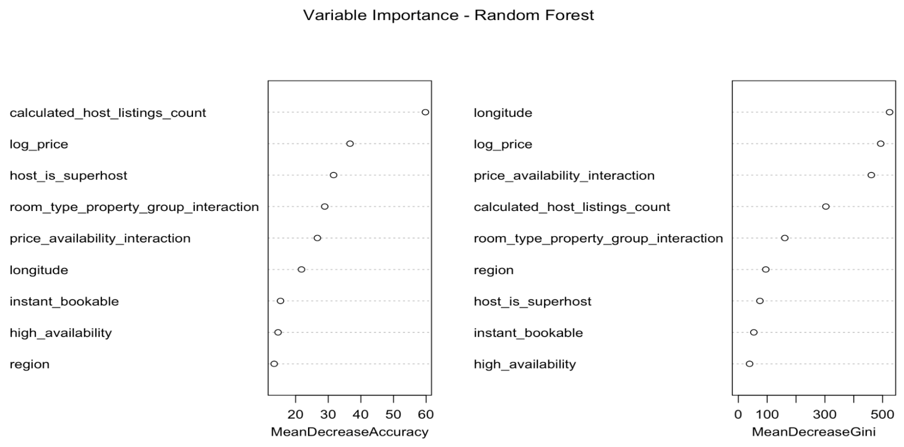
High availability



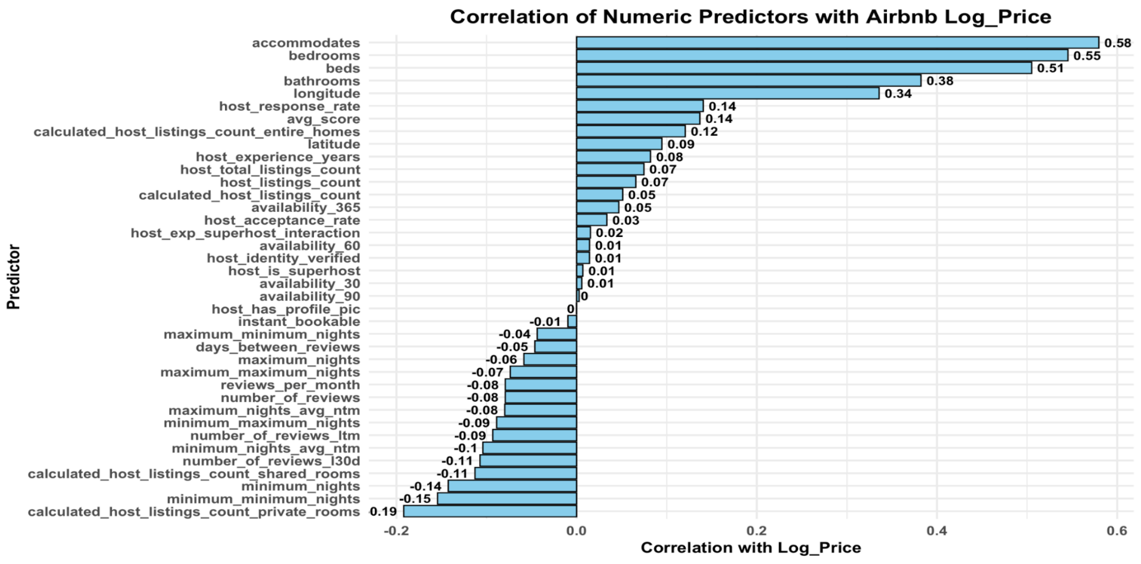
Predictors used in Q2

- host_is_superhost
- log(price
- longitude
- (host_experience_years
- latitude
- calculated_host_listings_count
- instant_bookable
- high_availability
- "Region"
- "property_group"
- "Host_response_time"
- "room_type"

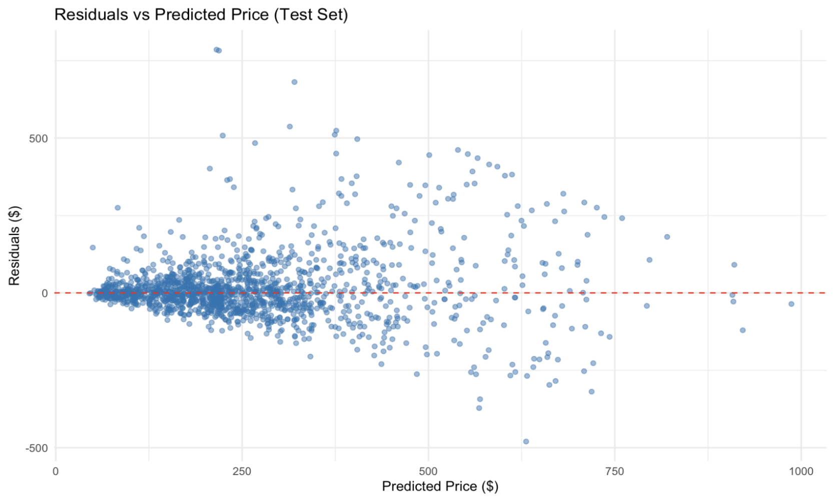
Random Forest Variable Importance(Classification)



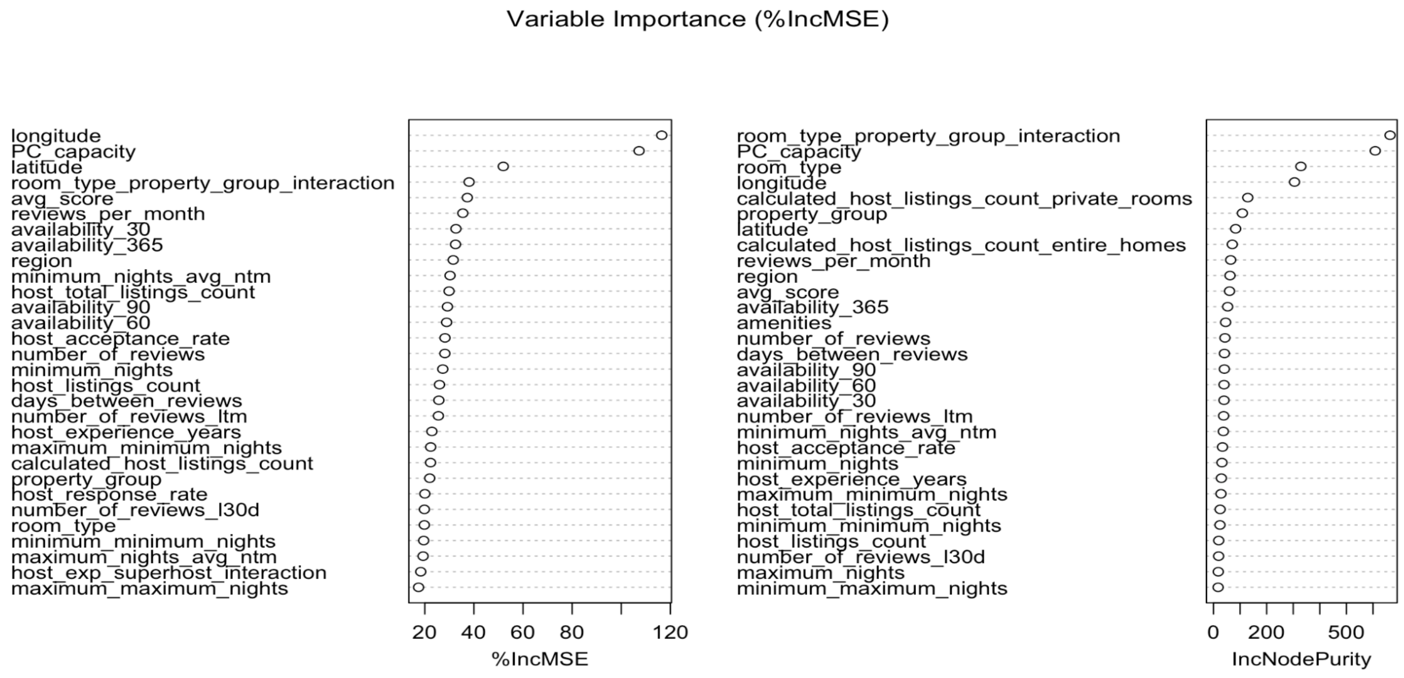
## 5.1 PCA



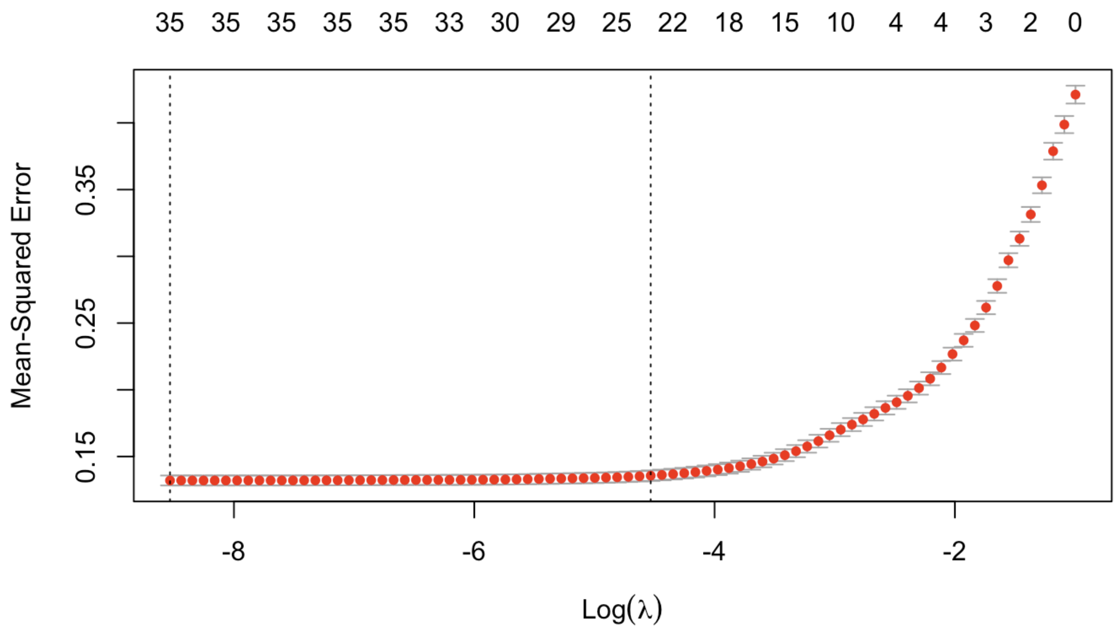
## 6.2 Raw Residual Analysis



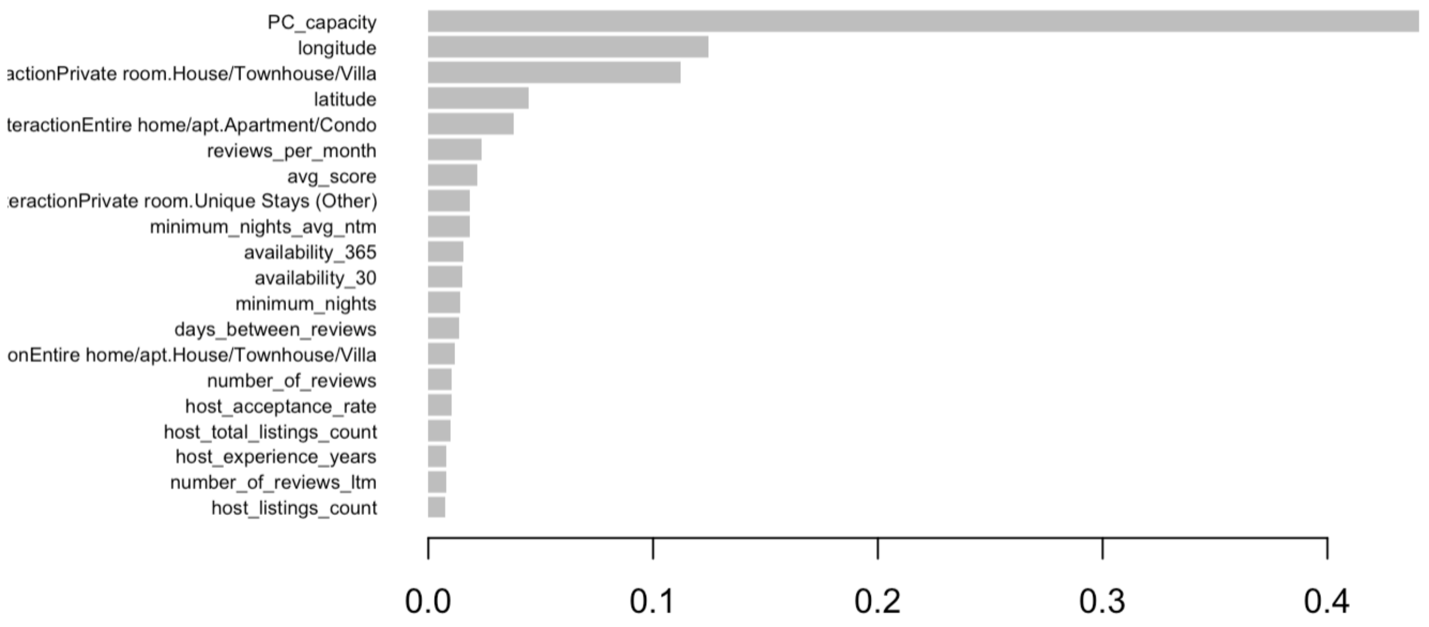
## 5.2 Feature Selection Random Forest's %IncMSE metric



**5.3.2 Lasso Regression** lambda.min (0.000197037)



**5.3.4 XGBoost Feature Importance Plot**


PC_capacity
longitude
actionPrivate room.House/Townhouse/Villa
latitude
teractionEntire home/apt.Apartment/Condo
reviews_per_month
avg_score
eractionPrivate room.Unique Stays (Other)
minimum_nights_avg_ntm
availability_365
availability_30
minimum_nights
days_between_reviews
onEntire home/apt.House/Townhouse/Villa
number_of_reviews
host_acceptance_rate
host_total_listings_count
host_experience_years
number_of_reviews_ltm
host_listings_count
0.0
0.1
0.2
0.3
0.4