```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.3     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.5.0     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
options(scipen = 999)
zurich <- read_csv("zurich_listings_699 (1).csv")
```

```
## Rows: 2534 Columns: 75
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (30): listing_url, last_scraped, source, name, description, neighborhood...
## dbl (37): id, scrape_id, host_id, host_listings_count, host_total_listings_c...
## lgl  (8): host_is_superhost, host_has_profile_pic, host_identity_verified, b...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
b <- unique(zurich$amenities)
zurich1 <- zurich[,-c(7,36,38,40,50,69)]
test_cases <- complete.cases(zurich1)
l <- sum(test_cases)
percentage <- (l/nrow(zurich1))*100
cat("percentage  and l", percentage, l)
```

```
## percentage  and l 0.394633 10
```

```
library(naniar)
missing_var <- miss_var_summary(zurich1)
zurich1$review_scores_value[is.na(zurich1$review_scores_value)] <- 0
bins= c(-Inf,0,4,5)
zurich1$review_value <- cut(zurich1$review_scores_value, breaks= bins, labels= c("No Rev
iews", "Poor Reviews","Good Reviews" ))
zurich1$review_scores_accuracy[is.na(zurich1$review_scores_accuracy)] <- 0
#summary(zurich1$review_scores_accuracy)
bins= c(-Inf,0,4,5)
zurich1$review_accuracy <- cut(zurich1$review_scores_accuracy, breaks= bins, labels= c
("No Reviews", "Poor Reviews","Good Reviews" ))
zurich1$review_scores_rating[is.na(zurich1$review_scores_rating)] <- 0
bins= c(-Inf,0,4,5)
zurich1$review_rating <- cut(zurich1$review_scores_rating, breaks= bins, labels= c("No R
eviews", "Poor Reviews","Good Reviews" ))
zurich1$review_scores_cleanliness [is.na(zurich1$review_scores_cleanliness )] <- 0
bins= c(-Inf,0,4,5)
zurich1$review_cleanliness  <- cut(zurich1$review_scores_cleanliness , breaks= bins, lab
els= c("No Reviews", "Poor Reviews","Good Reviews" ))
zurich1$review_scores_checkin [is.na(zurich1$review_scores_checkin )] <- 0
bins= c(-Inf,0,4,5)
zurich1$review_checkin  <- cut(zurich1$review_scores_checkin , breaks= bins, labels= c
("No Reviews", "Poor Reviews","Good Reviews" ))
zurich1$review_scores_communication [is.na(zurich1$review_scores_communication )] <- 0
bins= c(-Inf,0,4,5)
zurich1$review_communication  <- cut(zurich1$review_scores_communication , breaks= bins,
labels= c("No Reviews", "Poor Reviews","Good Reviews" ))
zurich1$review_scores_location  [is.na(zurich1$review_scores_location  )] <- 0
bins= c(-Inf,0,4,5)
zurich1$review_location   <- cut(zurich1$review_scores_location  , breaks= bins, labels=
c("No Reviews", "Poor Reviews","Good Reviews" ))
zurich1 <- zurich1[,-c(57,58,59,60,61,62,62)]
zurich2 <- zurich1 %>%
          mutate(NumPrice=as.numeric(gsub("[$,]","",zurich1$price))) %>%
          mutate(baths=case_when(
            grepl("(half).*",  zurich1$bathrooms_text, ignore.case = TRUE) ~0.5,
            TRUE ~ as.numeric(gsub("[^0-9.]+","",zurich1$bathrooms_text))
          ))
zurich2$baths <- ifelse(is.na(zurich2$baths), 1,zurich2$baths)
u_room_type<- unique(zurich2$room_type)
u_property_type <- unique(zurich2$property_type)
test_u_property_type <- zurich2 %>% filter(property_type=="Casa particular")
test_u_room_type <- zurich2 %>% filter(room_type=="Hotel room")
test_u_beds <- unique(zurich2$beds)
c<- mode(test_u_beds)
zurich2$beds <- ifelse(is.na(zurich2$beds) & zurich2$room_type == "Shared room", zurich2
$accommodates,
                       ifelse(is.na(zurich2$beds) & zurich2$room_type %in% c("Private ro
om", "Entire home/apt") & zurich2$accommodates %in% 1:2, 1,
                              ifelse(is.na(zurich2$beds) & zurich2$room_type %in% c("Pri
vate room", "Entire home/apt") & zurich2$accommodates %in% 3:8, ceiling(zurich2$accommod
ates/2),
```

```
                        zurich2$beds)))
zurich_FE <- zurich2 %>% mutate(guestsPerBath= zurich2$accommodates/zurich2$baths) %>% m
utate(guestsPerBed = zurich2$accommodates/zurich2$beds)
zurich2_price_nonas<- zurich2 %>% filter(!is.na(NumPrice))
zurich2_price_nas<- zurich2 %>% filter(is.na(NumPrice))
zurich2_beds_nonas <- zurich2 %>% filter(is.na(beds))
price_imputing_mlr_model <- lm(NumPrice~neighbourhood_cleansed + neighbourhood_group_cle
ansed  + room_type +accommodates+beds, zurich2_price_nonas)
#{step_mlr <- step(price_imputing_mlr_model,  method= "backward")}
impute_price_preds <- predict(price_imputing_mlr_model,zurich2_price_nas)
zurich_test <- zurich_FE %>% group_by(neighbourhood_group_cleansed,neighbourhood_cleanse
d, property_type,room_type, beds) %>% arrange(neighbourhood_group_cleansed,neighbourhood
_cleansed, property_type,room_type, beds)

zurich_imputed <- zurich_test %>% mutate(NumPrice= ifelse(is.na(NumPrice), mean(NumPric
e, na.rm= TRUE), NumPrice))
zurich_imputed <- zurich_imputed[,-c(35,37)]
test_values <- unique(zurich1$host_neighbourhood)
seerow<- zurich1[465,]
cleansed<- unique(zurich1$neighbourhood_cleansed)
grp_cleansed <- unique(zurich1$neighbourhood_group_cleansed)
zurich_baths <- zurich2 %>% filter(is.na(baths))
zurich2$baths <- as.numeric(gsub("(half).*", "0.5", zurich2$baths, ignore.case = TRUE))
```

```
                                                               Classification Tree
```

```
# Load necessary libraries
library(dplyr)
library(rpart)
library(rpart.plot)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(ggplot2)
library(tidyverse)
library(naniar)
library(dplyr)
library(arules)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
##
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
library(arulesViz)
library(caret)
library(rpart)
library(rpart.plot)

# Step 1: Replace "N/A" with "no response"
zurich_imputed$host_response_time <- ifelse(zurich_imputed$host_response_time == "N/A",
"no response", zurich_imputed$host_response_time)

zurich_imputed$host_is_superhost <- as.factor(zurich_imputed$host_is_superhost)
zurich_imputed$host_identity_verified <- as.factor(zurich_imputed$host_identity_verifie
d)
zurich_imputed$instant_bookable <- as.factor(zurich_imputed$instant_bookable)
zurich_imputed$room_type <- as.factor(zurich_imputed$room_type)

# Step 2: Selecting relevant columns
data_for_model <- zurich_imputed %>%
  select( host_is_superhost, host_identity_verified, instant_bookable,room_type)
```

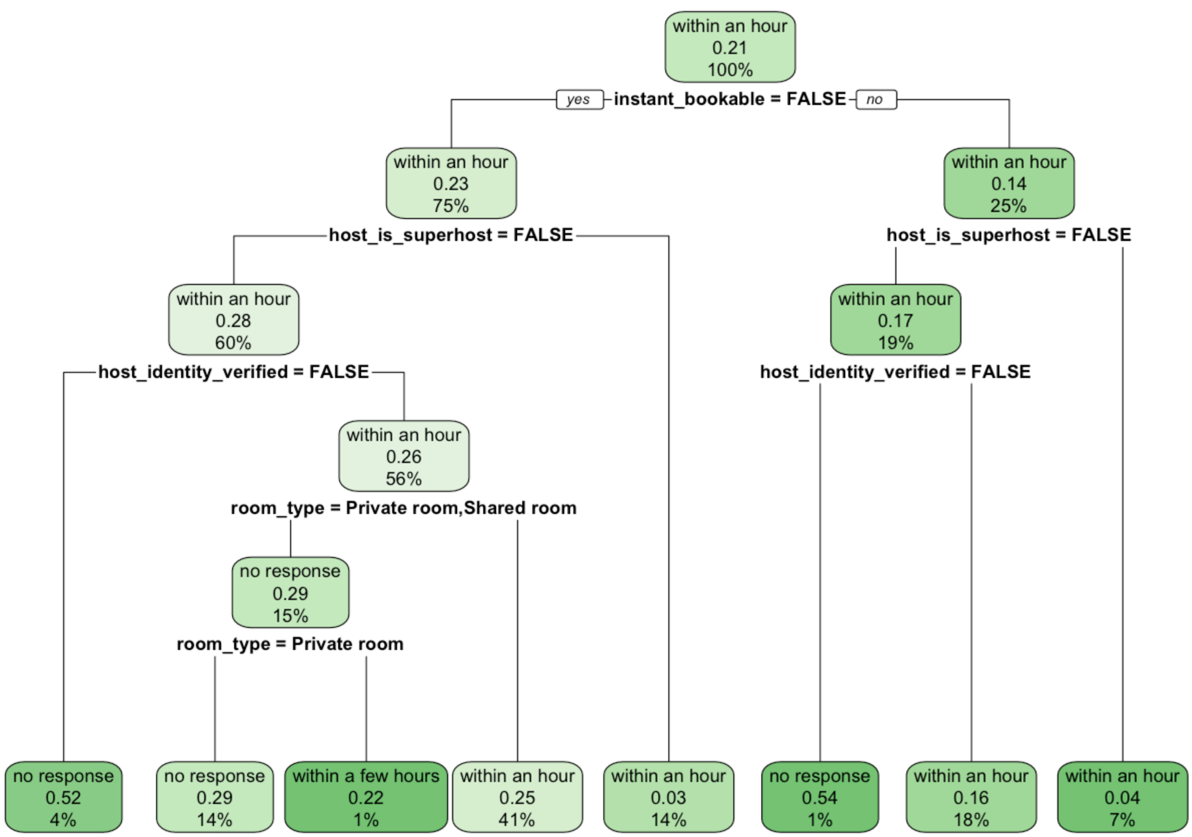
```
## Adding missing grouping variables: `neighbourhood_group_cleansed`,
## `neighbourhood_cleansed`, `property_type`, `beds`
```

```
# Splitting the dataset into training and validation sets
set.seed(123)  # for reproducibility
sample <- createDataPartition(zurich_imputed$host_response_time, p=0.6, list=FALSE)
train.df <- zurich_imputed[sample,]
valid.df <- zurich_imputed[-sample,]


# Building the classification tree on the training data with simplified parameters
tree_model <- rpart(host_response_time ~  instant_bookable+ host_is_superhost+ host_iden
tity_verified+room_type , data = train.df, method = "class", control = rpart.control(cp
= 0.0))

# Plotting the tree with simpler visual settings
rpart.plot(tree_model, extra = 106 , box.palette = "Greens")
```

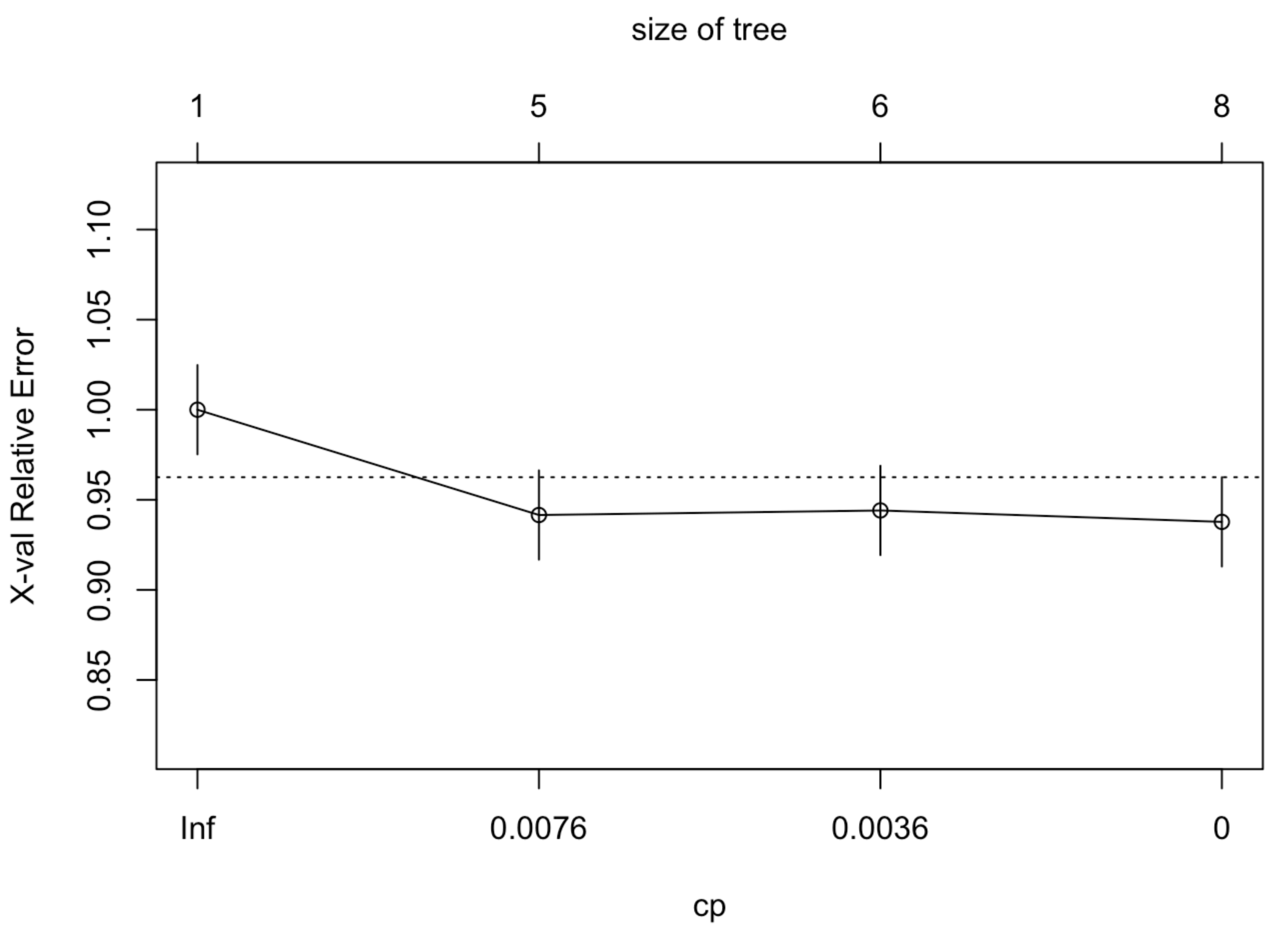
```
## Warning: extra=106 but the response has 5 levels (only the 2nd level is
## displayed)
```




```
# Print the complexity parameter table
printcp(tree_model)
```

```
##
## Classification tree:
## rpart(formula = host_response_time ~ instant_bookable + host_is_superhost +
##     host_identity_verified + room_type, data = train.df, method = "class",
##     control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
## [1] host_identity_verified host_is_superhost      instant_bookable
## [4] room_type
##
## Root node error: 787/1523 = 0.51674
##
## n= 1523
##
##          CP nsplit rel error  xerror     xstd
## 1 0.0114358      0   1.00000 1.00000 0.024780
## 2 0.0050826      4   0.94155 0.94155 0.024785
## 3 0.0025413      5   0.93647 0.94409 0.024787
## 4 0.0000000      7   0.93139 0.93774 0.024782
```

```
# Plot the complexity parameter against cross-validation error
plotcp(tree_model)
```



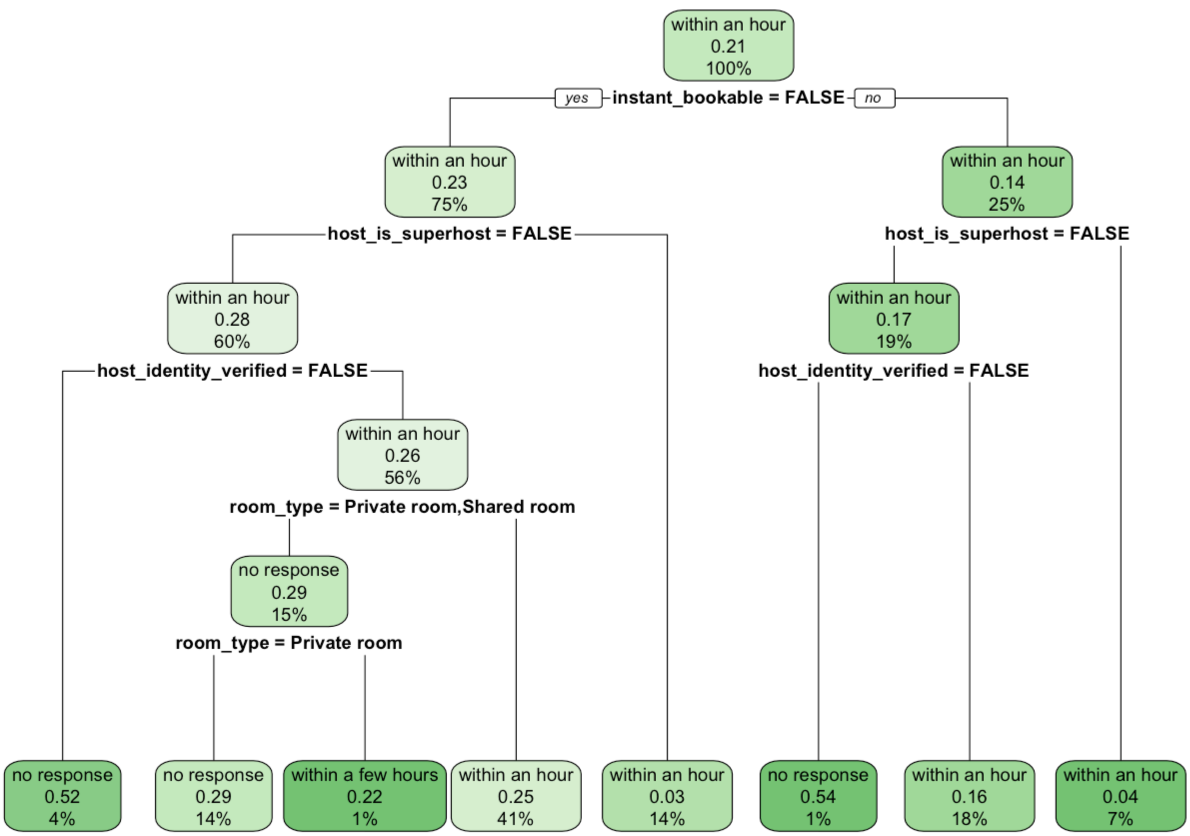
```
# Use the xerror (cross-validated error) to find the optimal cp value
optimal_cp <- tree_model$cptable[which.min(tree_model$cptable[, "xerror"]), "CP"]
optimal_cp
```

```
## [1] 0
```

```
# Now prune the tree using the optimal cp value
pruned_tree <- prune(tree_model, cp = 0.001)

# Plot the pruned tree
rpart.plot(pruned_tree, extra = 106, box.palette = "Greens")
```

```
## Warning: extra=106 but the response has 5 levels (only the 2nd level is
## displayed)
```




```
library(caret)
# Optionally, you can evaluate the pruned tree's performance on the validation set
predictions <- predict(pruned_tree, valid.df, type = "class")

print(confusionMatrix)
```

```
## function (data, ...)
## {
##     UseMethod("confusionMatrix")
## }
## <bytecode: 0x12f6e6e60>
## <environment: namespace:caret>
```

```
# Convert both the predicted and true class labels to factors
predictions_factor <- factor(predictions, levels = unique(c(predictions, valid.df$host_r
esponse_time)))
reference_factor <- factor(valid.df$host_response_time, levels = unique(c(predictions, v
alid.df$host_response_time)))

# Now use these factors in the confusionMatrix function
confusionMatrix(data = predictions_factor, reference = reference_factor)
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction            5   2   4 within a few hours no response within an hour
##   5                   0   0   0                  0           0              0
##   2                   0   0   0                  0           0              0
##   4                   0   0   0                  0           0              0
##   within a few hours  0   0   0                  1           0              0
##   no response         0   0   0                 46          60             37
##   within an hour      0   0   0                 97         150            453
##   a few days or more  0   0   0                  0           0              0
##   within a day        0   0   0                  0           0              0
##                     Reference
## Prediction           a few days or more within a day
##   5                                   0            0
##   2                                   0            0
##   4                                   0            0
##   within a few hours                  0            1
##   no response                        15           41
##   within an hour                     21           89
##   a few days or more                  0            0
##   within a day                        0            0
##
## Overall Statistics
##
##                Accuracy : 0.5084
##                  95% CI : (0.4771, 0.5397)
##     No Information Rate : 0.4847
##     P-Value [Acc > NIR] : 0.06962
##
##                   Kappa : 0.1383
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 5 Class: 2 Class: 4 Class: within a few hours
## Sensitivity                NA       NA       NA                 0.0069444
## Specificity                 1        1        1                 0.9988466
## Pos Pred Value             NA       NA       NA                 0.5000000
## Neg Pred Value             NA       NA       NA                 0.8582755
## Prevalence                  0        0        0                 0.1424332
## Detection Rate              0        0        0                 0.0009891
## Detection Prevalence        0        0        0                 0.0019782
## Balanced Accuracy          NA       NA       NA                 0.5028955
##                      Class: no response Class: within an hour
## Sensitivity                     0.28571                0.9245
## Specificity                     0.82647                0.3148
## Pos Pred Value                  0.30151                0.5593
## Neg Pred Value                  0.81527                0.8159
## Prevalence                      0.20772                0.4847
## Detection Rate                  0.05935                0.4481
```

```
## Detection Prevalence           0.19683              0.8012
## Balanced Accuracy              0.55609              0.6196
##                        Class: a few days or more Class: within a day
## Sensitivity                             0.00000          0.0000
## Specificity                             1.00000          1.0000
## Pos Pred Value                              NaN             NaN
## Neg Pred Value                          0.96439          0.8704
## Prevalence                              0.03561          0.1296
## Detection Rate                          0.00000          0.0000
## Detection Prevalence                    0.00000          0.0000
## Balanced Accuracy                       0.50000          0.5000
```

```
predictions <- predict(pruned_tree, valid.df, type = "class")
confusionMatrix <- table(Predicted = predictions, Actual = valid.df$host_response_time)
print(confusionMatrix)
```

```
##                       Actual
## Predicted              a few days or more no response within a day
##   a few days or more                    0           0            0
##   no response                          15          60           41
##   within a day                          0           0            0
##   within a few hours                    0           0            1
##   within an hour                       21         150           89
##                       Actual
## Predicted              within a few hours within an hour
##   a few days or more                    0              0
##   no response                          46             37
##   within a day                          0              0
##   within a few hours                    1              0
##   within an hour                       97            453
```

We started by preparing our dataset, where we transformed key variables into factors and replaced missing values labeled “N/A” with “no response”. To streamline our model and reduce its complexity, we chose to focus on specific variables that we believed would significantly impact Airbnb host response times. These included whether the host is a superhost, their identity verification status, if the listing is instantly bookable, and the type of room offered. By limiting the number of variables, we aimed to simplify the tree structure and make our model easier to interpret. Using the rpart package in R, we constructed a classification tree centered around these chosen features. To ensure our model was neither underfitting nor overfitting, we employed cross-validation methods to pinpoint the ideal complexity parameter (cp). This helped us prune our tree effectively, maintaining only the most significant branches and ensuring our model was robust yet straightforward.

Insights:

From the visual analysis of the initial tree, it was clear that certain features, like superhost status and room type, played a pivotal role in predicting how quickly a host would respond. The cross-validation process led us to the optimal cp value, which we used to prune our tree to enhance its generalizability to new data. After refining our model, we tested its performance on a validation set. The results, illustrated through the confusion matrix, revealed that our model excelled in predicting responses “within an hour” but faced challenges with less common categories such as “a few days or more”. This discrepancy suggested potential areas for further data collection or model adjustment to better capture these rare outcomes.

We also uncovered the intricate factors that influence host responsiveness on Airbnb. Our choice to focus on select variables was not only strategic in reducing the tree's complexity but also effective in drawing meaningful conclusions from the model. This was not just about building a predictive model but also about gaining deeper insights that could benefit both hosts, by helping them understand factors that lead to faster response times, and guests, by setting more accurate expectations. This project was a valuable learning experience in applying various techniques to real-world data.