Project

3

2025-04-22

Step I: Data Preparation & Exploration

Missing Values

read data

```
df <- read.csv("~/Desktop/BU-Master Courses/2nd Sem Classes/AD 699 A1 Data Mining /AD 69
9 Group Project/copenhagen_listings.csv", header=TRUE)
```

id

```
df <- df %>%
  mutate(
    id = as.character(id),
    scrape_id = as.character(scrape_id),
    host_id = as.character(host_id)
  )
str(df[, c("id", "scrape_id", "host_id")])
```

```
## 'data.frame':    21707 obs. of  3 variables:
##  $ id       : chr  "31094" "32379" "32841" "38499" ...
##  $ scrape_id: chr  "20241230011627" "20241230011627" "20241230011627" "2024123001162
7" ...
##  $ host_id  : chr  "129976" "140105" "142143" "122489" ...
```

We convert the id, scrape_id, and host_id columns in our dataset to character type because these variables represent identifiers rather than numeric values with mathematical meaning. After the conversion, we check the structure of these columns to confirm that the data types have been correctly updated.

date

```
df <- df %>%
  mutate(
    last_scraped = ymd(last_scraped),
    host_since = ymd(host_since),
    calendar_last_scraped = ymd(calendar_last_scraped),
    first_review = ymd(first_review),
    last_review = ymd(last_review)
  )
str(df[, c("last_scraped", "host_since", "calendar_last_scraped", "first_review", "last_
review")])
```

```
## 'data.frame':	21707 obs. of  5 variables:
##  $ last_scraped         : Date, format: "2024-12-30" "2024-12-30" ...
##  $ host_since           : Date, format: "2010-05-22" "2010-06-07" ...
##  $ calendar_last_scraped: Date, format: "2024-12-30" "2024-12-30" ...
##  $ first_review         : Date, format: "2010-08-16" "2010-08-23" ...
##  $ last_review          : Date, format: "2022-08-22" "2024-10-28" ...
```

We convert the last_scraped, host_since, calendar_last_scraped, first_review, and last_review columns to date format using the ymd() function because these variables represent specific points in time rather than plain text or numbers. We do this to ensure that any future operations involving dates, such as calculating durations or filtering listings by time periods, are handled correctly and efficiently.

num

```
df <- df %>%
  mutate(
    host_response_rate = as.numeric(gsub("%", "", host_response_rate)) / 100,
    host_acceptance_rate = as.numeric(gsub("%", "", host_acceptance_rate)) / 100,
    price = as.numeric(gsub("[$,]", "", price))
  )
```

```
## Warning: There were 2 warnings in `mutate()`.
## The first warning was:
## ℹ In argument: `host_response_rate = as.numeric(gsub("%", "",
##   host_response_rate))/100`.
## Caused by warning:
## ! NAs introduced by coercion
## ℹ Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
str(df[, c("host_response_rate", "host_acceptance_rate", "price")])
```

```
## 'data.frame':	21707 obs. of  3 variables:
##  $ host_response_rate  : num  NA 1 NA 1 1 NA NA NA 1 NA ...
##  $ host_acceptance_rate: num  NA 1 NA 1 0.65 0.89 0.92 1 0.57 0.5 ...
##  $ price               : num  NA NA NA 3000 2200 ...
```

We clean and convert the host_response_rate, host_acceptance_rate, and price columns to numeric values to prepare them for quantitative analysis. We do this by removing any non-numeric characters, such as percentage signs or currency symbols, and then converting the cleaned strings into numbers. For the response and acceptance rates, we further divide by 100 to express them as decimal form instead of percentages, making them easier to use in calculations or modeling.

bool

```
df <- df %>%
  mutate(
    host_is_superhost = ifelse(host_is_superhost == "t", TRUE,
                           ifelse(host_is_superhost == "f", FALSE, NA)),
    host_has_profile_pic = ifelse(host_has_profile_pic == "t", TRUE,
                              ifelse(host_has_profile_pic == "f", FALSE, NA)),
    host_identity_verified = ifelse(host_identity_verified == "t", TRUE,
                                ifelse(host_identity_verified == "f", FALSE, NA)),
    instant_bookable = ifelse(instant_bookable == "t", TRUE,
                          ifelse(instant_bookable == "f", FALSE, NA)),
    has_availability = ifelse(has_availability == "t", TRUE,
                          ifelse(has_availability == "f", FALSE, NA))
  )
str(df[, c("host_is_superhost", "host_has_profile_pic", "host_identity_verified", "insta
nt_bookable", "has_availability")])
```

```
## 'data.frame':    21707 obs. of  5 variables:
##  $ host_is_superhost     : logi  FALSE TRUE FALSE TRUE TRUE TRUE ...
##  $ host_has_profile_pic  : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
##  $ host_identity_verified: logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
##  $ instant_bookable      : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ has_availability      : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
```

We transform the host_is_superhost, host_has_profile_pic, host_identity_verified, instant_bookable, and has_availability columns into logical values by recoding entries labeled “t” as TRUE and “f” as FALSE, while leaving any other unexpected or missing entries as NA to preserve data integrity.

NA

```
df <- df %>%
  dplyr::select(-neighbourhood_group_cleansed, -calendar_updated, -license)
```

We remove the neighbourhood_group_cleansed, calendar_updated, and license columns from the dataset because we look at the dataset and realize that these columns are almost all NAs. So we think they are not useful for our intend analysis, and that removing them would reduce the clutter and focus only on those variables that were meaningful.

bathrooms, bathrooms_text

```
df <- df %>%
  mutate(
    #     bathrooms    NA    bathrooms_text     NA       NA
    bathrooms = ifelse(is.na(bathrooms) & is.na(bathrooms_text), NA,
                       ifelse(is.na(bathrooms), as.numeric(str_extract(bathrooms_text,
"\\d+\\.?\\d*")), bathrooms)),

    #     bathrooms_text    NA    bathrooms    NA       NA
    bathrooms_text = ifelse(is.na(bathrooms_text) & is.na(bathrooms), NA,
                            ifelse(is.na(bathrooms_text), paste(bathrooms, "bath", ifels
e(bathrooms > 1, "s", "")), bathrooms_text))
  )


unique(df$bathrooms_text)
```

```
##  [1] "1.5 baths"         "2 baths"           "1 bath"
##  [4] "1 shared bath"     "3 baths"           "2 shared baths"
##  [7] "2.5 baths"         ""                  "Half-bath"
## [10] "0 baths"           "1 private bath"    "Private half-bath"
## [13] "3.5 baths"         "1.5 shared baths"  "5 baths"
## [16] "Shared half-bath"  "0 shared baths"    "4 baths"
## [19] "2.5 shared baths"  "3 shared baths"    "8 baths"
```

```
df <- df %>%
  mutate(
    bathrooms = case_when(
      bathrooms_text %in% c("Half-bath", "Private half-bath", "Shared half-bath") ~ 0.5,
      TRUE ~ as.numeric(str_extract(bathrooms_text, "\\d+\\.?\\d*"))  #
    )
  )

str(df[, c("bathrooms", "bathrooms_text")])
```

```
## 'data.frame':    21707 obs. of  2 variables:
##  $ bathrooms     : num  1.5 2 1 1 2 1 1.5 1 1.5 1 ...
##  $ bathrooms_text: chr  "1.5 baths" "2 baths" "1 bath" "1 bath" ...
```

We clean and harmonize the bathrooms and bathrooms_text columns to ensure they are consistent. We do this by first filling missing bathrooms values using the numeric information extracted from bathrooms_text, and vice versa, ensuring that neither field is unnecessarily left blank if useful information is available. Then, we further refine the bathrooms column by assigning a value of 0.5 to listings described as having a “half-bath” and extracting numerical values from other bathroom descriptions. This approach improves the quality and completeness of our bathroom data while handling special cases like half-baths appropriately.

NA

```
df <- df %>%
  mutate(across(where(is.character), ~ ifelse(is.na(.) | . == "", "Unknown", .)))
df <- df %>%
  mutate(across(where(is.numeric), ~ ifelse(. == "", NA, .)))
```

We do this by replacing any NA or empty strings in character columns with the label “Unknown,” ensuring that missing text information is clearly identified rather than left blank. For numeric columns, we replace any empty strings with NAs, recognizing that numeric fields should either have a valid number or be explicitly missing.

The reason we don’t delete rows that contain NA values is because, for a row of data, just because he has NA in one of the columns doesn’t mean that all the columns are NA, and we can still analyze it using other columns instead of simply deleting an entire column. Similarly, we do not populate the NA values with mean, median, etc., as we believe that this affects the accuracy of the original data set. In subsequent analyses, we will extract the desired columns and drop the NA values from them, but not change the dataset itself.

```
# 1.
numeric_cols <- df %>% dplyr::select(where(is.numeric))
summary(numeric_cols)
```

```
##  host_response_rate host_acceptance_rate host_listings_count
##  Min.   :0.000      Min.   :0.000        Min.   :   1.000   
##  1st Qu.:1.000      1st Qu.:0.430        1st Qu.:   1.000   
##  Median :1.000      Median :0.750        Median :   1.000   
##  Mean   :0.887      Mean   :0.671        Mean   :   6.002   
##  3rd Qu.:1.000      3rd Qu.:1.000        3rd Qu.:   1.000   
##  Max.   :1.000      Max.   :1.000        Max.   :3284.000   
##  NA's   :10524      NA's   :4663         NA's   :2          
##  host_total_listings_count    latitude       longitude      accommodates   
##  Min.   :   1.0            Min.   :55.62   Min.   :12.45   Min.   : 1.000  
##  1st Qu.:   1.0            1st Qu.:55.67   1st Qu.:12.54   1st Qu.: 2.000  
##  Median :   1.0            Median :55.68   Median :12.55   Median : 3.000  
##  Mean   :  12.5            Mean   :55.68   Mean   :12.56   Mean   : 3.309  
##  3rd Qu.:   2.0            3rd Qu.:55.70   3rd Qu.:12.58   3rd Qu.: 4.000  
##  Max.   :8065.0            Max.   :55.73   Max.   :12.64   Max.   :16.000  
##  NA's   :2                                                                 
##    bathrooms        bedroom         beds           price       
##  Min.   :0.000   Min.   :0.00   Min.   : 0.00   Min.   :    72  
##  1st Qu.:1.000   1st Qu.:1.00   1st Qu.: 1.00   1st Qu.:   895  
##  Median :1.000   Median :1.00   Median : 1.00   Median :  1150  
##  Mean   :1.092   Mean   :1.61   Mean   : 1.82   Mean   :  1390  
##  3rd Qu.:1.000   3rd Qu.:2.00   3rd Qu.: 2.00   3rd Qu.:  1600  
##  Max.   :8.000   Max.   :8.00   Max.   :18.00   Max.   :100000  
##  NA's   :10      NA's   :712    NA's   :9223    NA's   :9236    
##  minimum_nights     maximum_nights   minimum_minimum_nights
##  Min.   :   1.000   Min.   :   1.0   Min.   :   1.000      
##  1st Qu.:   2.000   1st Qu.:  20.0   1st Qu.:   2.000      
##  Median :   3.000   Median : 150.0   Median :   3.000      
##  Mean   :   4.567   Mean   : 328.3   Mean   :   4.305      
##  3rd Qu.:   4.000   3rd Qu.: 365.0   3rd Qu.:   4.000      
##  Max.   :1111.000   Max.   :1125.0   Max.   :1111.000      
##                                                            
##  maximum_minimum_nights minimum_maximum_nights maximum_maximum_nights
##  Min.   :   1.000       Min.   :   1.0         Min.   :   1.0        
##  1st Qu.:   2.000       1st Qu.:  21.0         1st Qu.:  21.0        
##  Median :   3.000       Median : 365.0         Median : 365.0        
##  Mean   :   4.884       Mean   : 395.3         Mean   : 398.2        
##  3rd Qu.:   4.000       3rd Qu.: 730.0         3rd Qu.: 730.0        
##  Max.   :1111.000       Max.   :1125.0         Max.   :1125.0        
##                                                                      
##  minimum_nights_avg_ntm maximum_nights_avg_ntm availability_30  availability_60
##  Min.   :   1.000       Min.   :   1.0         Min.   : 0.000   Min.   : 0.00  
##  1st Qu.:   2.000       1st Qu.:  21.0         1st Qu.: 0.000   1st Qu.: 0.00  
##  Median :   3.000       Median : 365.0         Median : 0.000   Median : 0.00  
##  Mean   :   4.607       Mean   : 396.7         Mean   : 7.666   Mean   :16.88  
##  3rd Qu.:   4.000       3rd Qu.: 730.0         3rd Qu.:17.000   3rd Qu.:35.00  
##  Max.   :1111.000       Max.   :1125.0         Max.   :30.000   Max.   :60.00  
##                                                                                
##  availability_90 availability_365 number_of_reviews number_of_reviews_ltm
##  Min.   : 0.00   Min.   :  0.00   Min.   :   0.00   Min.   :  0.000      
##  1st Qu.: 0.00   1st Qu.:  0.00   1st Qu.:   2.00   1st Qu.:  0.000      
##  Median : 1.00   Median : 17.00   Median :   8.00   Median :  2.000      
##  Mean   :26.36   Mean   : 90.85   Mean   :  18.49   Mean   :  5.162  
##  3rd Qu.:58.00   3rd Qu.:166.00   3rd Qu.:  19.00   3rd Qu.:  6.000  
##  Max.   :90.00   Max.   :365.00   Max.   :2138.00   Max.   :569.000  
##                                                                        
##  number_of_reviews_l30d review_scores_rating review_scores_accuracy
##  Min.   : 0.0000        Min.   :0.000        Min.   :1.00          
##  1st Qu.: 0.0000        1st Qu.:4.760        1st Qu.:4.80          
##  Median : 0.0000        Median :4.920        Median :4.92          
##  Mean   : 0.2568        Mean   :4.835        Mean   :4.85          
##  3rd Qu.: 0.0000        3rd Qu.:5.000        3rd Qu.:5.00          
##  Max.   :87.0000        Max.   :5.000        Max.   :5.00          
##                         NA's   :2630         NA's   :2632          
##  review_scores_cleanliness review_scores_checkin review_scores_communication
##  Min.   :1.000             Min.   :1.00          Min.   :1.000             
##  1st Qu.:4.600             1st Qu.:4.83          1st Qu.:4.900             
##  Median :4.820             Median :4.96          Median :5.000             
##  Mean   :4.715             Mean   :4.88          Mean   :4.917             
##  3rd Qu.:5.000             3rd Qu.:5.00          3rd Qu.:5.000             
##  Max.   :5.000             Max.   :5.00          Max.   :5.000             
##  NA's   :2632              NA's   :2632          NA's   :2632              
##  review_scores_location review_scores_value calculated_host_listings_count
##  Min.   :1.000          Min.   :1.000       Min.   :  1.000               
##  1st Qu.:4.770          1st Qu.:4.630       1st Qu.:  1.000               
##  Median :4.910          Median :4.780       Median :  1.000               
##  Mean   :4.845          Mean   :4.726       Mean   :  4.297               
##  3rd Qu.:5.000          3rd Qu.:4.940       3rd Qu.:  1.000               
##  Max.   :5.000          Max.   :5.000       Max.   :223.000               
##  NA's   :2633           NA's   :2632                                      
##  calculated_host_listings_count_entire_homes
##  Min.   :  0.000                            
##  1st Qu.:  1.000                            
##  Median :  1.000                            
##  Mean   :  4.086                            
##  3rd Qu.:  1.000                            
##  Max.   :223.000                            
##                                             
##  calculated_host_listings_count_private_rooms
##  Min.   : 0.0000                             
##  1st Qu.: 0.0000                             
##  Median : 0.0000                             
##  Mean   : 0.2038                             
##  3rd Qu.: 0.0000                             
##  Max.   :16.0000                             
##                                              
##  calculated_host_listings_count_shared_rooms reviews_per_month
##  Min.   :0.000000                            Min.   : 0.0100  
##  1st Qu.:0.000000                            1st Qu.: 0.2000  
##  Median :0.000000                            Median : 0.4100  
##  Mean   :0.005574                            Mean   : 0.7131  
##  3rd Qu.:0.000000                            3rd Qu.: 0.8300  
##  Max.   :7.000000                            Max.   :82.6000  
##                                              NA's   :2630     
```

Finally, we looked at all of the value columns in order to confirm that there were no outliers or values in there that shouldn't be there. We did find some extreme values, such as a very high price ($100,000), or a very long booking time (1125 nights), and so on. But we analyzed these as reasonable, perhaps the landlord owns a great big villa, and the price of the villa is very high. Also including the very long living time, since some people may indeed wish to stay for a long time, which is also reasonable. We checked all the columns one by one and didn't find impossible values like negative prices, scores that are not in the normal range of 1-5, or latitude and longitude coordinates that are not in Copenhagen but in the Pacific Ocean or something like that, all the data inside the columns are reasonable in our opinion, some of them may be very extreme but not unexplainable.

Summary Statistics

```
summary_stats <- df %>%
  group_by(neighbourhood_cleansed) %>%
  summarise(
    listing_count = n(),  #
    avg_price = mean(price, na.rm = TRUE),  #
    median_price = median(price, na.rm = TRUE),  #
    avg_review_score = mean(review_scores_rating, na.rm = TRUE),  #
    superhost_rate = mean(host_is_superhost == TRUE, na.rm = TRUE),  #
    instant_bookable_rate = mean(instant_bookable == TRUE, na.rm = TRUE),  #
    avg_minimum_nights = mean(minimum_nights, na.rm = TRUE)  #
  ) %>%
  arrange(desc(listing_count))
summary_stats
```

neighbourhood_cleansed	listing_count	avg_price	median_price	avg_review_score	s
<chr>	<int>	<dbl>	<dbl>	<dbl>	
Nrrebro	3896	1203.0813	1073	4.843494	
Vesterbro-Kongens Enghave	3734	1390.5509	1186	4.850504	
Indre By	3073	1899.7602	1600	4.811310	
sterbro	2318	1426.4455	1196	4.843299	
Frederiksberg	2310	1349.4435	1157	4.850385	
Amager Vest	1930	1337.4215	1195	4.846440	
Amager st	1652	1315.9876	1000	4.831821	
Bispebjerg	1037	953.1906	850	4.806091	
Valby	848	1068.3452	903	4.799684	
Vanlse	519	1073.0174	880	4.811538	

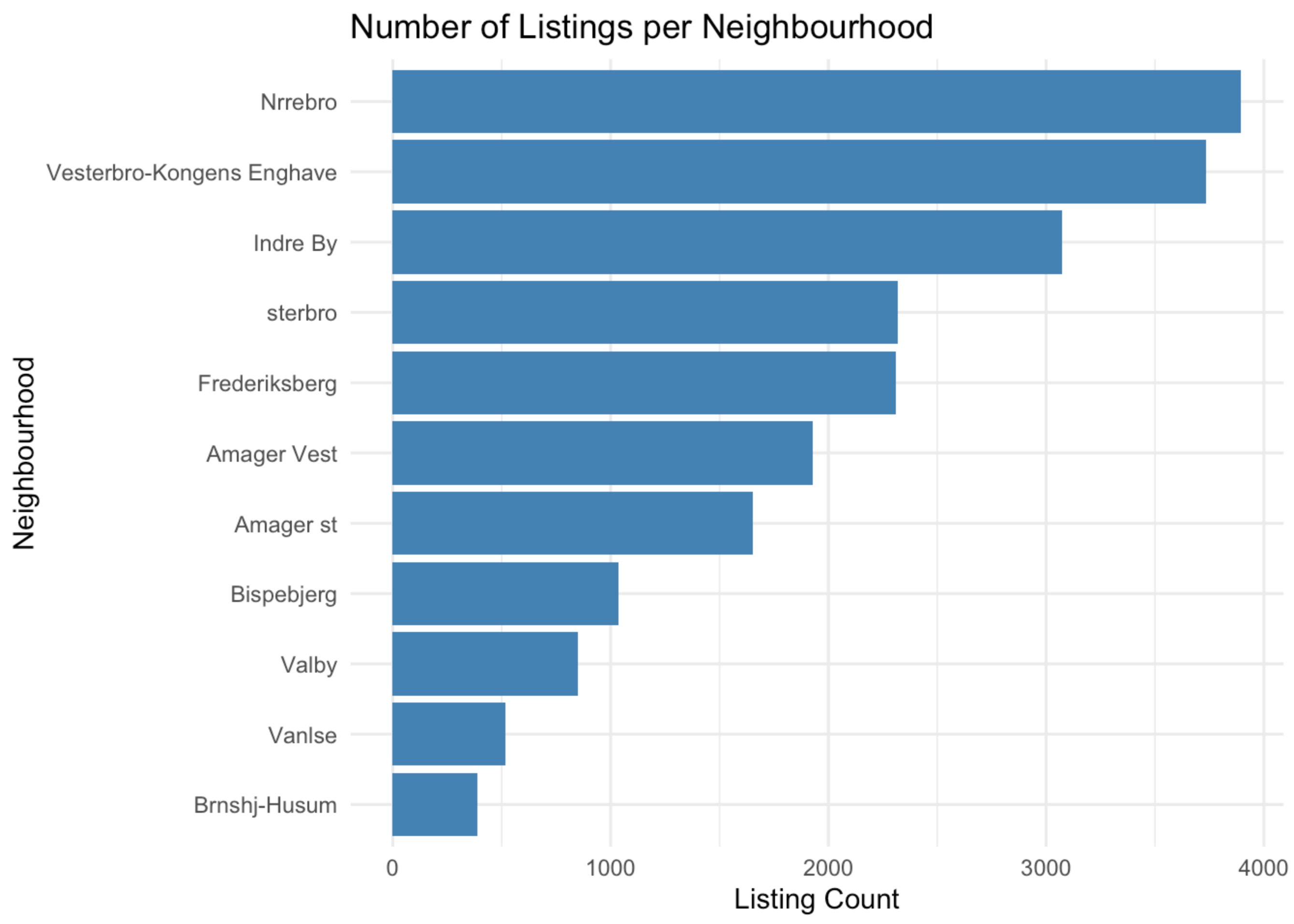
1-10 of 11 rows | 1-6 of 8 columns Previous **1** 2 Next

In our analysis of Copenhagen's Airbnb listings by neighborhood, we found that Nrrebro has the highest number of listings (3896), followed closely by Vesterbro-Kongens Enghave (3734) and Indre By (3073). This suggests that these areas are major hubs for short-term rentals. In terms of price, Indre By clearly stands out with an average listing price of DKK 1899.76 and a median price of DKK 1600, significantly higher than other neighborhoods. In contrast, Bispebjerg has one of the lowest average prices at DKK 953.19. This price gap indicates that staying in the city center (Indre By) comes at a premium, likely reflecting its proximity to major attractions.

Looking at host and booking characteristics, we noticed that Indre By also has the highest superhost rate (18.35%) and the highest instant bookable rate (15.36%), suggesting a more professional and tourist-ready host base compared to other areas like Nrrebro and sterbro, where the superhost rates are closer to 10%. Review scores were relatively high and consistent across neighborhoods, all clustering around 4.8 out of 5, indicating strong guest satisfaction city-wide. Finally, when considering minimum stay requirements, Brnshj-Husum had the longest average minimum stay at 5.54 nights, while Vesterbro-Kongens Enghave had the shortest at 4.2 nights, hinting at different hosting strategies depending on location—possibly with more residential areas encouraging longer stays.

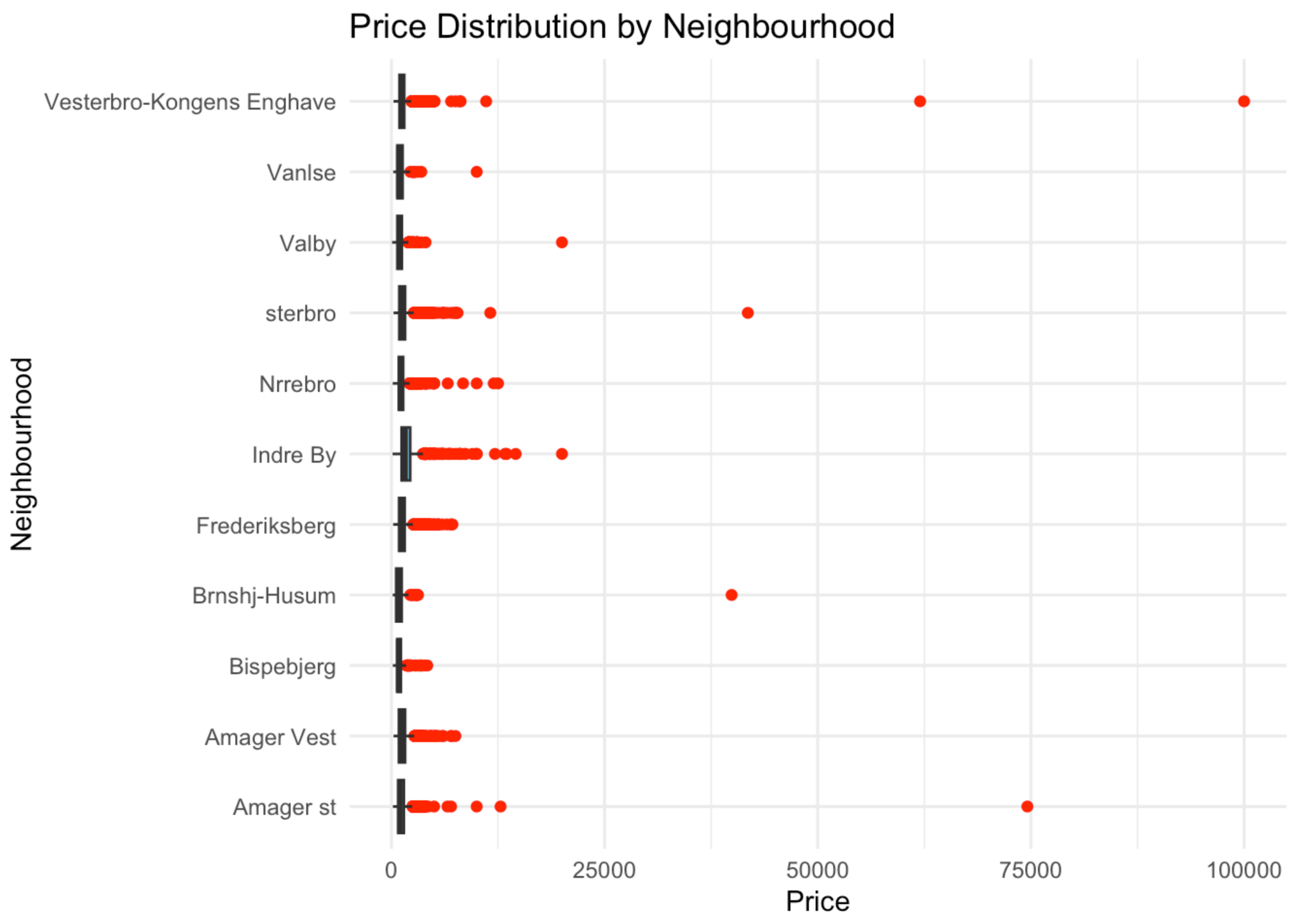
Data Visualization

```
ggplot(df %>% count(neighbourhood_cleansed),
       aes(x = reorder(neighbourhood_cleansed, n), y = n)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Number of Listings per Neighbourhood",
       x = "Neighbourhood", y = "Listing Count") +
  theme_minimal()
```

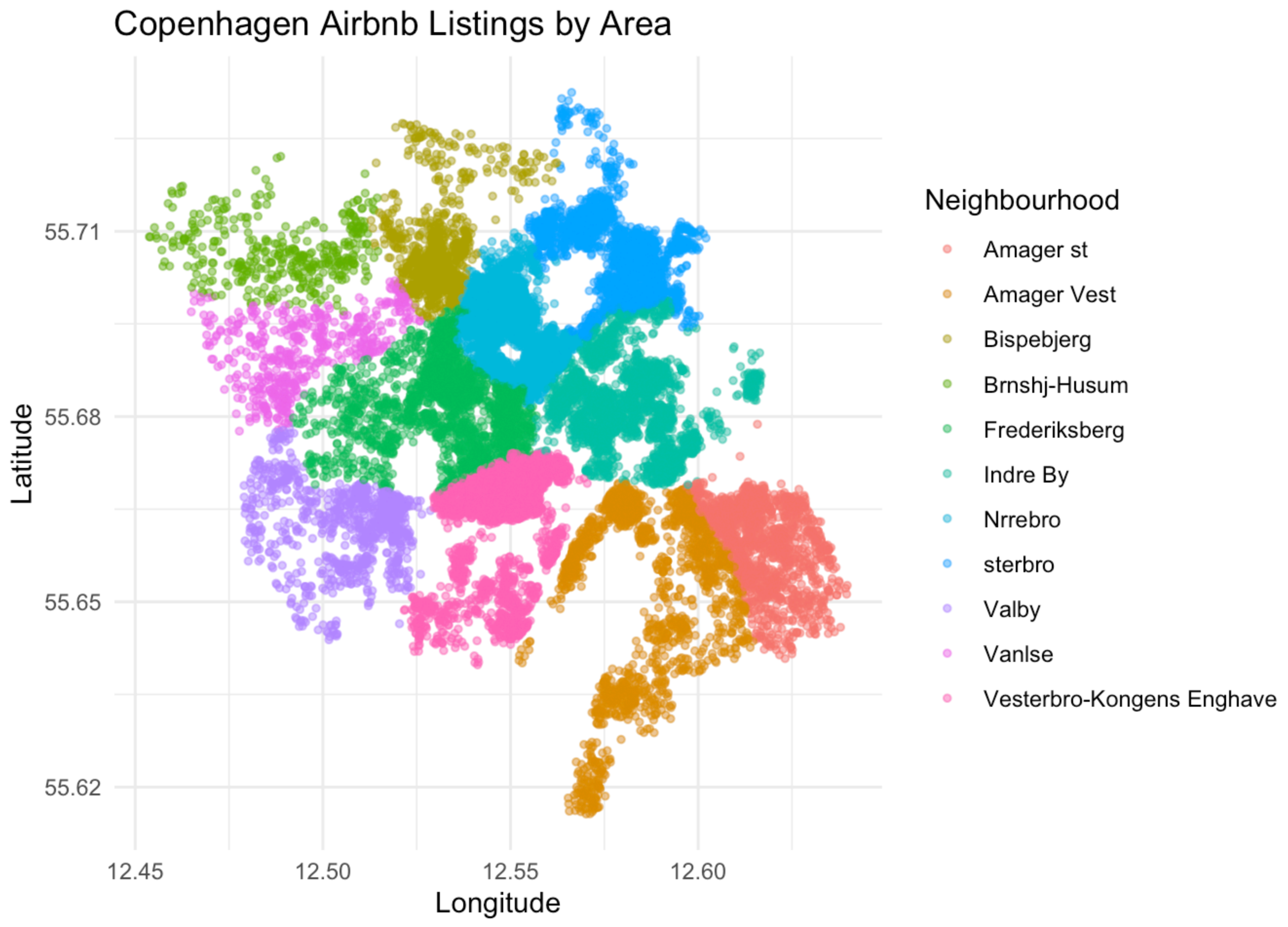


```
ggplot(df, aes(x = neighbourhood_cleansed, y = price)) +
  geom_boxplot(outlier.color = "red", fill = "skyblue") +
  coord_flip() +
  labs(title = "Price Distribution by Neighbourhood",
       x = "Neighbourhood", y = "Price") +
  theme_minimal()
```

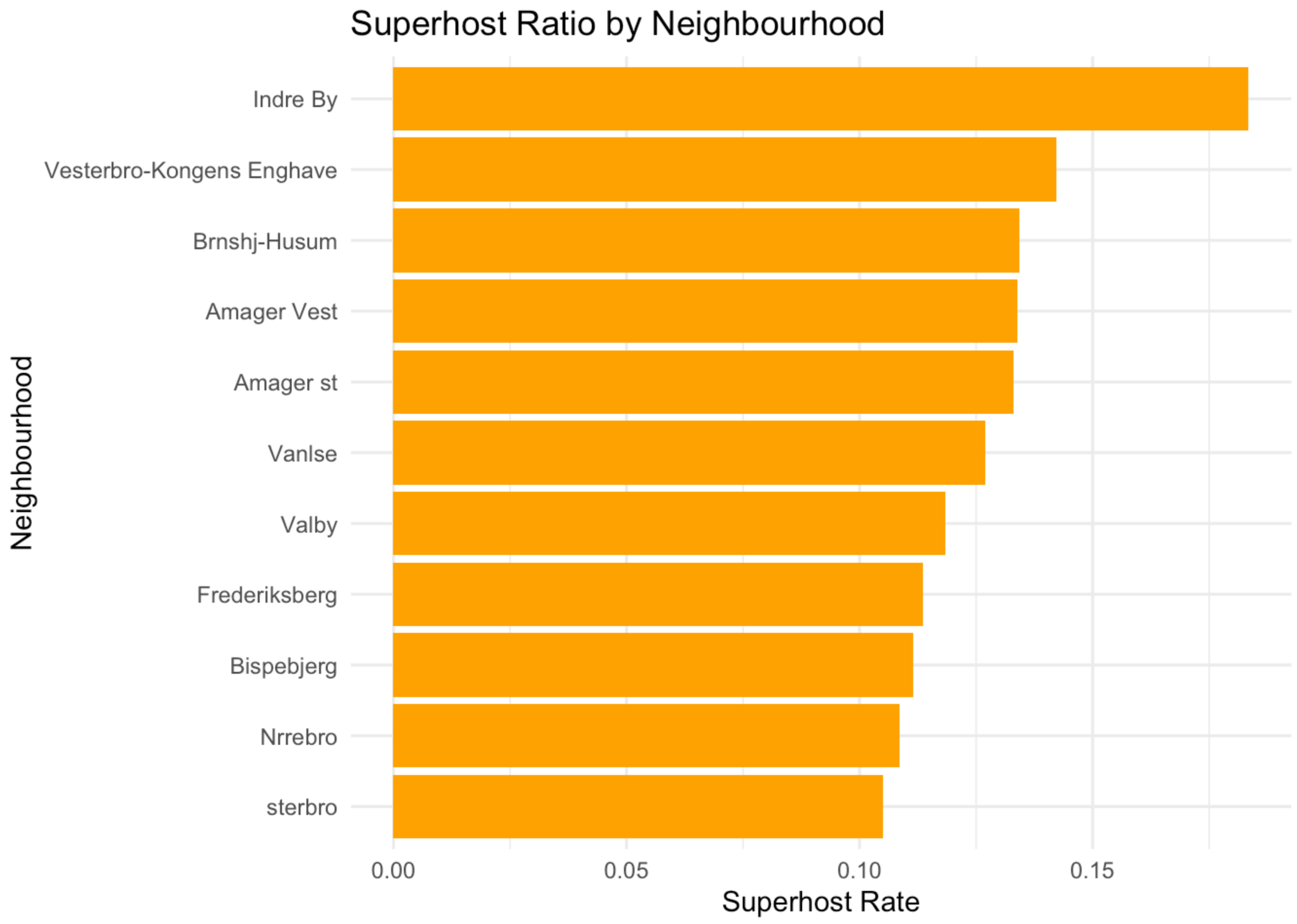
```
## Warning: Removed 9236 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```



```
ggplot(df, aes(x = longitude, y = latitude, color = neighbourhood_cleansed)) +
  geom_point(alpha = 0.5, size = 1) +
  theme_minimal() +
  labs(title = "Copenhagen Airbnb Listings by Area",
       x = "Longitude", y = "Latitude", color = "Neighbourhood")
```

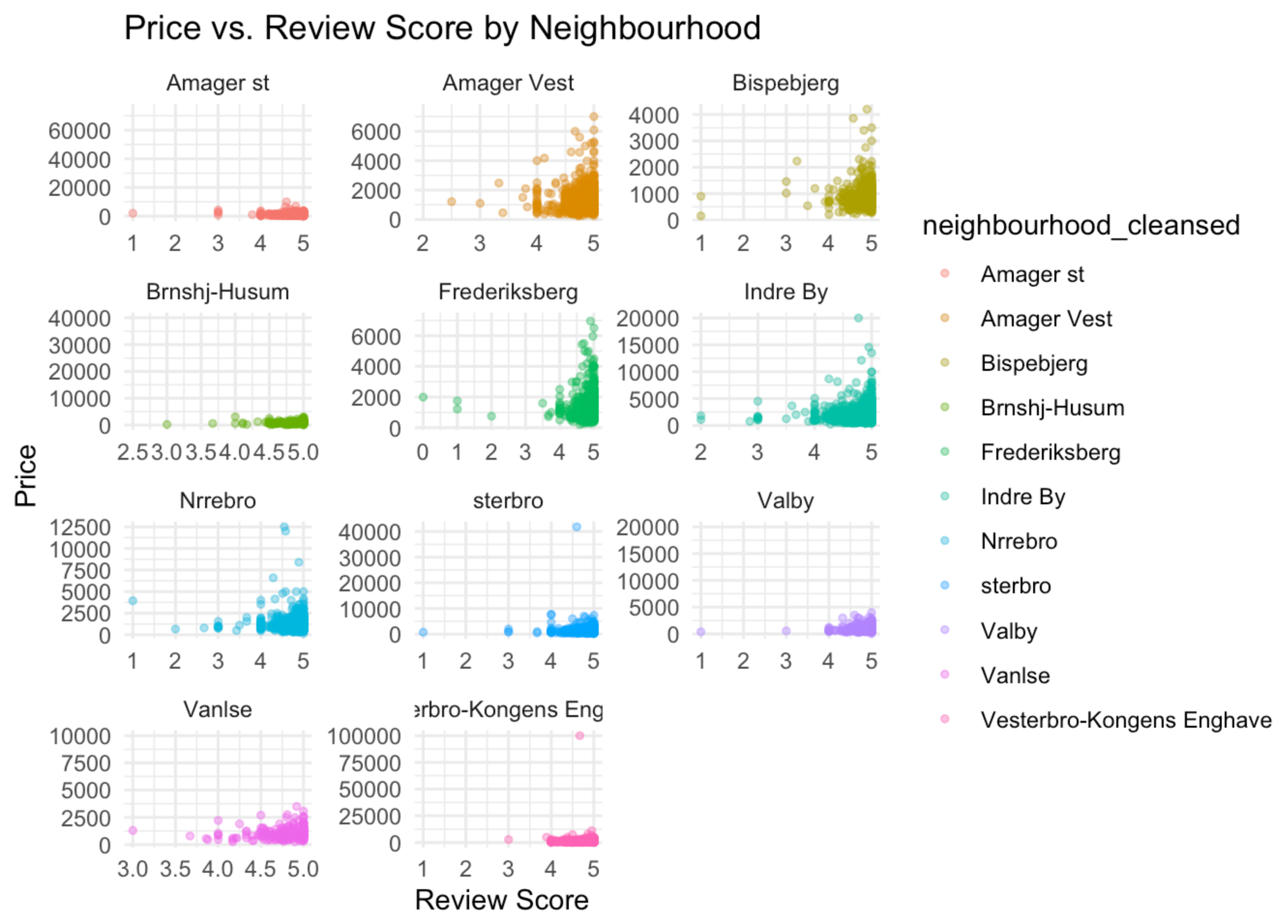


```
df %>%
  group_by(neighbourhood_cleansed) %>%
  summarise(superhost_rate = mean(host_is_superhost == TRUE, na.rm = TRUE)) %>%
  ggplot(aes(x = reorder(neighbourhood_cleansed, superhost_rate), y = superhost_rate)) +
  geom_col(fill = "orange") +
  coord_flip() +
  labs(title = "Superhost Ratio by Neighbourhood",
       x = "Neighbourhood", y = "Superhost Rate") +
  theme_minimal()
```



```
ggplot(df, aes(x = review_scores_rating, y = price, color = neighbourhood_cleansed)) +
  geom_point(alpha = 0.4, size = 1) +
  labs(title = "Price vs. Review Score by Neighbourhood",
       x = "Review Score", y = "Price") +
  theme_minimal() +
  facet_wrap(~ neighbourhood_cleansed, scales = "free", ncol = 3)
```

```
## Warning: Removed 10929 rows containing missing values or values outside the scale ran
ge
## (`geom_point()`).
```

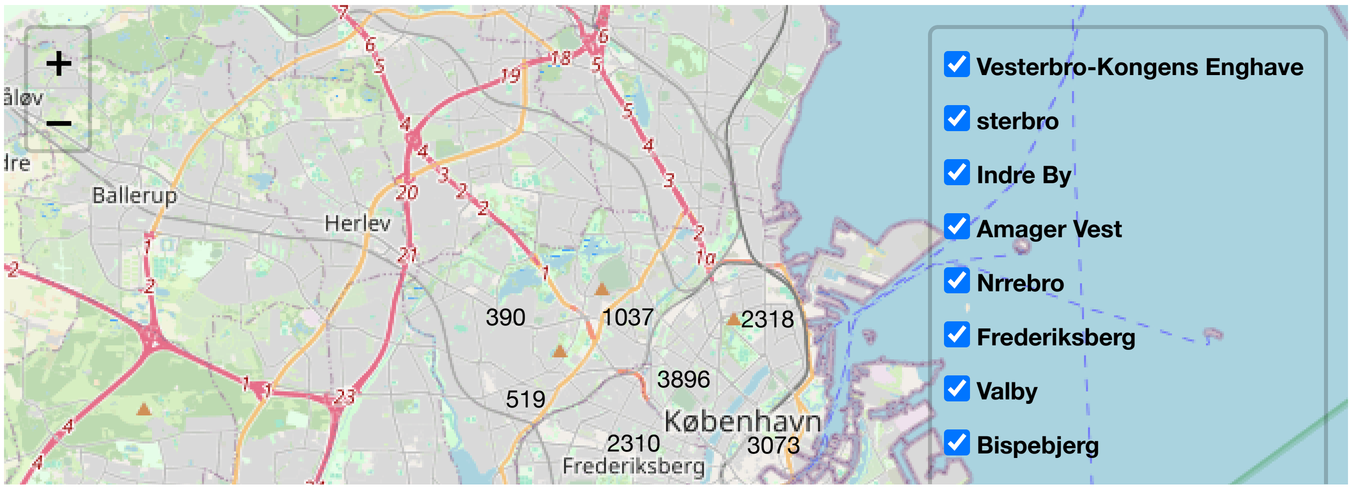


To dive into Copenhagen's Airbnb landscape, we crafted five distinct visualizations, each spotlighting a different facet of the market. Our first chart, a horizontal bar plot of listing counts, makes it immediately clear that Nrrebro and Vesterbro-Kongens Enghave dominate the scene with nearly 4000 listings apiece, while tucked-away neighborhoods like Brnshj-Husum and Vanlse each hover below 500. Next, a boxplot of prices reveals that although most areas share similar median price points, Vesterbro-Kongens Enghave boasts a handful of sky-high outliers pushing into the six-figure DKK range, which shows the evidence of luxury offerings interspersed among more budget-friendly options.

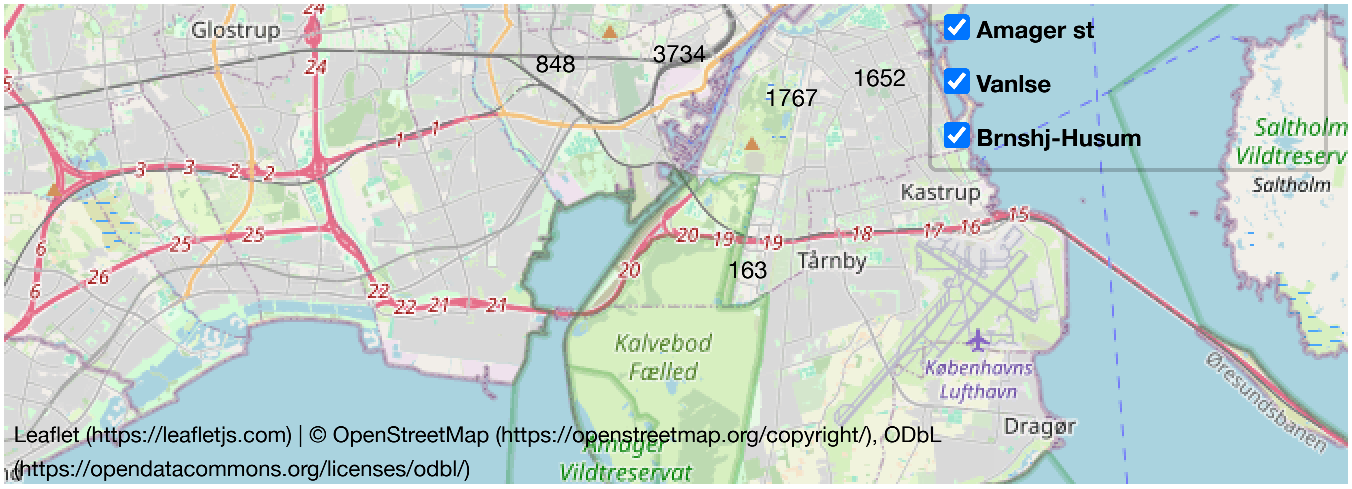
The geospatial scatter of longitude vs. latitude then brings the data to life on a map, showing tight clusters of listings around Indre By and its neighboring districts, and sparser dots as you move toward the city's edge. Shifting to host quality, our bar chart of superhost rates confirms that Indre By leads the pack with over 15% superhosts, whereas areas like sterbro sit closer to 10%, hinting at more polished operations in the heart of the city. Finally, facetted scatter plots of price against review score demonstrate a clear pattern, higher-rated listings generally command higher prices—but the strength of that relationship varies. In Vesterbro-Kongens Enghave, you'll find steep price tags even among average-scoring properties, while Vanlse shows a tighter, more predictable price–rating link, underlining how different neighborhoods cater to distinct guest expectations.

Mapping

```
#
areas <- unique(df$neighbourhood_cleansed)

#
pal <- colorFactor("Set3", domain = df$neighbourhood_cleansed)

#
m <- leaflet() %>%
  addTiles()

#
for (area in areas) {
  subdf <- df[df$neighbourhood_cleansed == area, ]
  m <- m %>%
    addCircleMarkers(
      data = subdf,
      lng = ~longitude, lat = ~latitude,
      color = ~pal(neighbourhood_cleansed),
      radius = 4,
      opacity = 0.7, fillOpacity = 0.7,
      popup = ~paste("Neighbourhood:", neighbourhood_cleansed, "<br>",
                     "Price:", price, "<br>",
                     "Review Score:", review_scores_rating),
      group = area,
      clusterOptions = markerClusterOptions()
    )
}

#
m <- m %>% addLayersControl(
  overlayGroups = areas,
  options = layersControlOptions(collapsed = FALSE)
)

#
m
```



✓ Vesterbro-Kongens Enghave
✓ sterbro
✓ Indre By
✓ Amager Vest
✓ Nrrebro
✓ Frederiksberg
✓ Valby
✓ Bispebjerg



Our interactive leaflet map immediately highlights how tightly Airbnb activity is concentrated in Copenhagen's core districts. When all layers are toggled on, you can see a dense "cloud" of points around Indre By, Vesterbro-Kongens Enghave, and Nrrebro. These neighborhoods host the vast majority of listings, sitting cheek-by-jowl around Nyhavn, the Tivoli Gardens, and the pedestrian streets of Strget. Just to the west, Frederiksberg and sterbro form smaller but still substantial clusters, reflecting their residential appeal and good transport links. In contrast, areas like Brnshj-Husum, Valby, and Vanlse appear as much lighter sprinklings, reminding us that Copenhagen's short-term rental market thins out as you cross into the quieter suburbs.

Zooming and toggling individual neighborhood layers reveals subtler patterns. Amager Vest and Amager st listings trace the coastline and metro line, suggesting a growing appetite for waterfront stays and easy airport access. Marker clusters in those zones often pop up around Bella Center and the new restad developments, pointing to business travelers or convention-goers. The color-coding also makes it obvious that superhost-heavy Indre By has a wide price spread, pale dots from budget rooms sit alongside darker, expensive listings, whereas outsider districts show a more uniform price band. Overall, the map teaches us not just where hosts are, but also hints at why they're there: proximity to attractions, transit corridors, and the balance between tourist and local people.

Wordcloud

```
#     neighbourhood_overview
text_data <- df$neighborhood_overview
text_data <- text_data[text_data != "Unknown"]

#
corpus <- Corpus(VectorSource(text_data))

#
corpus <- corpus %>%
  tm_map(content_transformer(tolower)) %>%
  tm_map(removePunctuation) %>%
  tm_map(removeNumbers) %>%
  tm_map(removeWords, stopwords("en")) %>%
  tm_map(stripWhitespace)
```

```
## Warning in tm_map.SimpleCorpus(., content_transformer(tolower)): transformation
## drops documents
```

```
## Warning in tm_map.SimpleCorpus(., removePunctuation): transformation drops
## documents
```

```
## Warning in tm_map.SimpleCorpus(., removeNumbers): transformation drops
## documents
```

```
## Warning in tm_map.SimpleCorpus(., removeWords, stopwords("en")): transformation
## drops documents
```

```
## Warning in tm_map.SimpleCorpus(., stripWhitespace): transformation drops
## documents
```

```
#
dtm <- TermDocumentMatrix(corpus)
m <- as.matrix(dtm)
word_freq <- sort(rowSums(m), decreasing = TRUE)

#
word_df <- data.frame(word = names(word_freq), freq = word_freq)

#
set.seed(699)
wordcloud(words = word_df$word, freq = word_df$freq,
          min.freq = 5,
          max.words = 200,
          random.order = FALSE,
          rot.per = 0.35,
          colors = brewer.pal(8, "Dark2"))
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## neighbourhood could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## copenhagens could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## famous could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## playgrounds could not be fit on page. It will not be plotted.
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## atmosphere could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## trendy could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## square could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## centre could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## short could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## perfect could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## lively could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## nature could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## offers could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## unique could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## home could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## bakery could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## wine could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## nyhavn could not be fit on page. It will not be plotted.
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## harbour could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## residential could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## access could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## friendly could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## large could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## opportunities could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## gardens could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## experience could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## attractions could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## buildings could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## harbor could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## transport could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## cultural could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## christianshavn could not be fit on page. It will not be plotted.
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## strøget could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## carlsberg could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## options could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## need could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## brygge could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## summer could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## cemetery could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## explore could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## swimming could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## diverse could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## supermarket could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## assistens could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## playground could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## across could not be fit on page. It will not be plotted.
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## delicious could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## transportation could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## near could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## along could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## stroll could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## royal could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## family could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## canals could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## houses could not be fit on page. It will not be plotted.
```

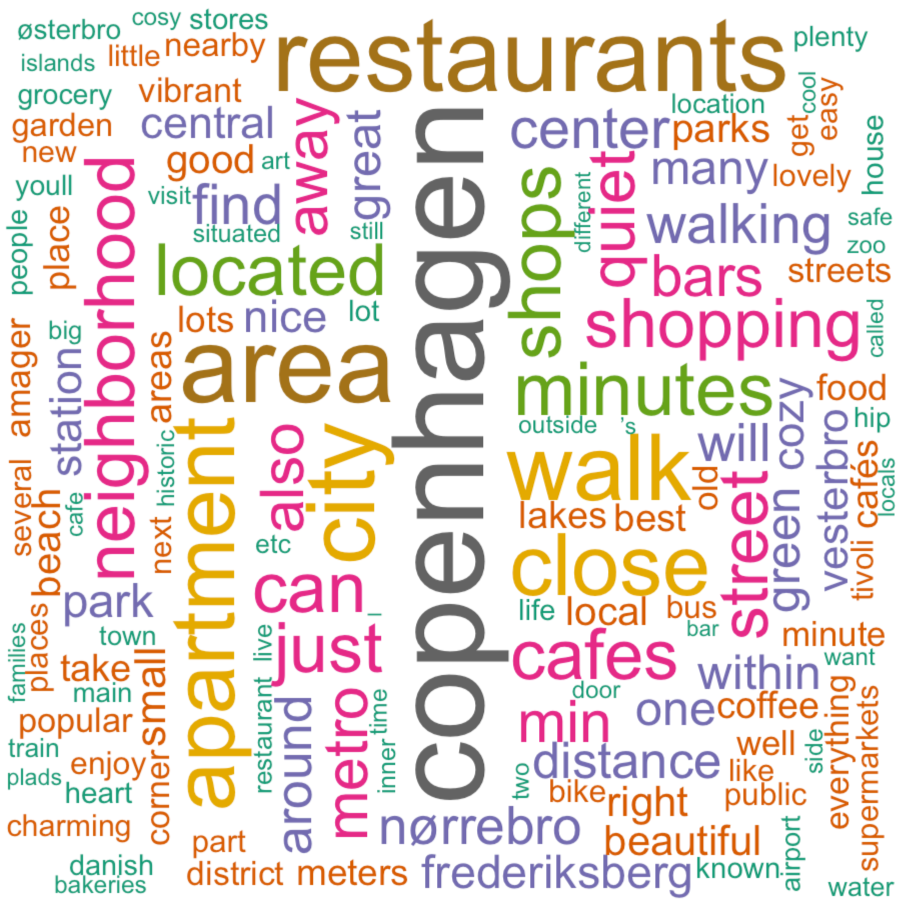
```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## middle could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## building could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## christiania could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## boutiques could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## fælledparken could not be fit on page. It will not be plotted.
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## jægersborggade could not be fit on page. It will not be plotted.
```



```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## architecture could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_df$word, freq = word_df$freq, min.freq = 5, :
## swim could not be fit on page. It will not be plotted.
```

Step II: Prediction

Multiple regression model

A. Modeling Process

We began by selecting variables grounded in both theory and data availability: host acceptance rate (a proxy for host responsiveness), neighborhood (to capture location premium), room type, and key listing characteristics (accommodates, bathrooms, bedrooms, minimum nights, and average review score). For some highly relevant variables, such as property type and room type, we only retain one of them. We transformed the outcome with a natural logarithm, log(price), to stabilize variance and improve linearity with predictors. Because the magnitude of the price is a bit too high compared to other numerical variables. To prepare the data, we dropped any rows with missing values among these fields and converted categorical variables to factors.

We checked pairwise correlations and variance to guard against multicollinearity. For example, we found that accommodates, bedrooms, and bathrooms were moderately correlated but each added distinct explanatory power, so we retained them. We also compared models with and without neighborhood fixed effects; including the ten neighborhood dummies raised adjusted R-squared by nearly 0.05, so we kept them to account for locational heterogeneity. We judged our final model's quality by overall fit (adjusted R-squared), the significance of individual coefficients matching our expectations, and residual diagnostics to ensure approximately normal errors.

```
# Select only the variables you listed, and drop rows with any NA
model_df <- df %>%
  dplyr::select(
    price,
    host_acceptance_rate,
    neighbourhood_cleansed,
    room_type,
    accommodates,
    bathrooms,
    bedrooms,
    minimum_nights,
    review_scores_rating
  ) %>%

  drop_na() %>%

  mutate(
    neighbourhood_cleansed = as.factor(neighbourhood_cleansed),
    room_type              = as.factor(room_type),
    log_price              = log(price)
  )


# Fit the multiple regression
model_price <- lm(log_price ~ . - price, data = model_df)

# View the summary
summary(model_price)
```

```
##
## Call:
## lm(formula = log_price ~ . - price, data = model_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87709 -0.19364 -0.02646  0.17296  2.54235
##
## Coefficients:
##                                                   Estimate Std. Error t value
## (Intercept)                                      5.7903781  0.0593842  97.507
## host_acceptance_rate                             0.0316928  0.0102218   3.100
## neighbourhood_cleansedAmager Vest                0.0797488  0.0153965   5.180
## neighbourhood_cleansedBispebjerg                -0.1114922  0.0187064  -5.960
## neighbourhood_cleansedBrnshj-Husum              -0.2905096  0.0280438 -10.359
## neighbourhood_cleansedFrederiksberg              0.1334874  0.0149293   8.941
## neighbourhood_cleansedIndre By                   0.3856567  0.0136777  28.196
## neighbourhood_cleansedNrrebro                    0.1003743  0.0137765   7.286
## neighbourhood_cleansedsterbro                    0.1515594  0.0150117  10.096
## neighbourhood_cleansedValby                     -0.1183802  0.0202857  -5.836
## neighbourhood_cleansedVanlse                    -0.1717469  0.0239928  -7.158
## neighbourhood_cleansedVesterbro-Kongens Enghave  0.1454514  0.0136350  10.667
## room_typeHotel room                             -0.5052774  0.1442227  -3.503
## room_typePrivate room                           -0.5538080  0.0115998 -47.743
## room_typeShared room                            -0.5841692  0.2274887  -2.568
## accommodates                                     0.0852252  0.0032628  26.120
## bathrooms                                        0.1205654  0.0115072  10.477
## bedrooms                                         0.1084345  0.0061636  17.593
## minimum_nights                                  -0.0023280  0.0003196  -7.284
## review_scores_rating                             0.1253444  0.0117744  10.646
##                                                 Pr(>|t|)
## (Intercept)                                      < 2e-16 ***
## host_acceptance_rate                            0.001937 **
## neighbourhood_cleansedAmager Vest               2.26e-07 ***
## neighbourhood_cleansedBispebjerg                2.60e-09 ***
## neighbourhood_cleansedBrnshj-Husum               < 2e-16 ***
## neighbourhood_cleansedFrederiksberg              < 2e-16 ***
## neighbourhood_cleansedIndre By                   < 2e-16 ***
## neighbourhood_cleansedNrrebro                   3.43e-13 ***
## neighbourhood_cleansedsterbro                    < 2e-16 ***
## neighbourhood_cleansedValby                     5.52e-09 ***
## neighbourhood_cleansedVanlse                    8.72e-13 ***
## neighbourhood_cleansedVesterbro-Kongens Enghave  < 2e-16 ***
## room_typeHotel room                             0.000461 ***
## room_typePrivate room                            < 2e-16 ***
## room_typeShared room                            0.010245 *
## accommodates                                     < 2e-16 ***
## bathrooms                                        < 2e-16 ***
## bedrooms                                         < 2e-16 ***
## minimum_nights                                  3.47e-13 ***
## review_scores_rating                             < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3215 on 10371 degrees of freedom
## Multiple R-squared:  0.573,  Adjusted R-squared:  0.5722
## F-statistic: 732.5 on 19 and 10371 DF,  p-value: < 2.2e-16
```

B. The regression equation predicts the logarithm of nightly price as a sum of an overall baseline plus adjustments for each feature. Concretely, the intercept (≈ 5.79) represents the log-price for an average entire home in the reference neighborhood (Amager Vest) with zero values on the numeric predictors. Each coefficient then shows how a one-unit increase in that predictor shifts log-price: for example, an extra bathroom adds about 0.121 to log-price (about 13 % higher price), each additional bedroom adds 0.108 (≈ 11 % more), and accommodating one more guest adds 0.085 (≈ 9 % more). Categorical effects work the same way on the log scale: being in Indre By adds 0.386 (≈ 47 % premium over Amager Vest), whereas listing a private room subtracts 0.554 (≈ 42 % discount relative to an entire home). Host acceptance rate and review score also boost price, higher responsiveness and better reviews translate into noticeably higher nightly rates.

C. Model Performance

With an adjusted R^2 of 0.572, our model explains about 57 % of the variation in nightly prices on the log scale—a respectable fit for cross-sectional listing data. The F-statistic is 732.5, with p-value < 2.2 × 10^-16 confirms that the set of predictors collectively contributes significant explanatory power beyond intercept-only. The residual standard error of 0.322 on log prices translates to a typical multiplicative error of exp(0.322) ≈ 1.38, meaning our model's predictions are, on average, within ±38 % of the true price.

However, there is still substantial unexplained variance, likely driven by unobserved factors such as interior quality, seasonality, or dynamic demand conditions. A more flexible modeling approach like tree-based ensembles might capture nonlinearity better. Nevertheless, our linear model offers clear, interpretable insights into how location, room type, and host/listing attributes shift expected price, providing a solid foundation for further refinement.

Step III: Classification

k-nearest neighbors

```
# 1.
df_knn <- df %>%
  filter(!is.na(amenities)) %>%
  mutate(
    has_wifi = factor(grepl("internet|wifi", amenities, ignore.case = TRUE),
                      levels = c(FALSE, TRUE))
  ) %>%
  select(has_wifi, price, accommodates, bathrooms, bedrooms, beds, review_scores_rating)
%>%
  drop_na()

# 2.
predictors <- scale(df_knn %>% select(-has_wifi))
knn_data <- bind_cols(has_wifi = df_knn$has_wifi, as.data.frame(predictors))

# 3.       /
set.seed(699)
idx <- createDataPartition(knn_data$has_wifi, p = 0.7, list = FALSE)
train <- knn_data[idx, ];  test <- knn_data[-idx, ]

train_x <- train %>% select(-has_wifi)
test_x  <- test  %>% select(-has_wifi)
train_y <- train$has_wifi
test_y  <- test$has_wifi

# 4.
k <- round(sqrt(nrow(train)))
pred_y <- knn(train = train_x, test = test_x, cl = train_y, k = k)

# 5.     pred_y
pred_y <- factor(pred_y, levels = levels(test_y))

# 6.
confusionMatrix(data = pred_y, reference = test_y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE     0    0
##      TRUE     74 3156
##
##                Accuracy : 0.9771
##                  95% CI : (0.9713, 0.982)
##     No Information Rate : 0.9771
##     P-Value [Acc > NIR] : 0.5309
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.00000
##             Specificity : 1.00000
##          Pos Pred Value :     NaN
##          Neg Pred Value : 0.97709
##              Prevalence : 0.02291
##          Detection Rate : 0.00000
##    Detection Prevalence : 0.00000
##       Balanced Accuracy : 0.50000
##
##        'Positive' Class : FALSE
##
```

For our first classification task, we chose Wi-Fi availability as the binary outcome, which is one of the most basic amenity that most guests expect.

We created the has_wifi indicator by searching the raw amenities text for “internet” or “wifi,” then encoded it as a factor with levels FALSE (no Wi-Fi) and TRUE (has Wi-Fi). To predict this indicator, we selected six numeric features that are both commonly reported and likely correlated with hosting quality such as nightly price, accommodates, bathrooms, bedrooms, beds, and overall review_scores_rating. We scaled all predictors to have zero mean and unit variance. Then we set our test seed at 699 ensure reproducibility and then split the data 70/30 into training and test sets.

We then built a confusion matrix after fitting knn() and predicting on the held-out 30%. We observed the model to achieve 97.71 % accuracy, but this exactly matches the naive benchmark of always predicting “Wi-Fi” is “TRUE” (since over 97 % of listings offer Wi-Fi). In effect, k-NN simply learned the overwhelming majority class; sensitivity is 0 % which means it never correctly predicts a non-Wi-Fi listing, and meanwhile specificity is 100 % which shows that it never falsely predicts missing Wi-Fi.

Thus, since nearly every listing offers Wi-Fi, our k-NN model simply predicts “Yes” for all cases and it matches the baseline of always choosing the majority, and yields no real improvement. Predicting less common amenities will require either features that capture rare cases or models specifically designed to handle imbalanced outcomes.

Naive Bayes

```r
set.seed(699)
# 1. Prepare & bin the target
df_nb <- df %>%
  filter(!is.na(review_scores_value)) %>%
  # equal-frequency binning into tertiles
  mutate(
    value_bin = ntile(review_scores_value, 3),
    value_bin = factor(value_bin,
                       levels = 1:3,
                       labels = c("Low", "Medium", "High"))
  ) %>%
  select(-review_scores_value)  # drop original

# 2. Select predictors (no other review_scores_*)
df_nb <- df_nb %>%
  select(
    value_bin,
    price,
    accommodates,
    bedrooms,
    bathrooms,
    beds,
    number_of_reviews,
    host_response_time,
    host_response_rate,
    host_acceptance_rate,
    host_is_superhost,
    host_identity_verified,
    neighbourhood_cleansed,
    room_type
  ) %>%
  drop_na() %>%
  mutate(
    host_response_time     = factor(host_response_time),
    host_is_superhost      = factor(host_is_superhost),
    host_identity_verified = factor(host_identity_verified),
    neighbourhood_cleansed = factor(neighbourhood_cleansed),
    room_type              = factor(room_type),
    log_price              = log(price)
  ) %>%
  select(-price)

# 4. Train/test split
train_idx <- createDataPartition(df_nb$value_bin, p = 0.7, list = FALSE)
train_nb <- df_nb[train_idx, ]
test_nb  <- df_nb[-train_idx, ]

# 5. Fit Naive Bayes
model_nb <- naiveBayes(value_bin ~ ., data = train_nb, laplace = 1)
```

```
# 6. Predict & evaluate
pred_nb <- predict(model_nb, test_nb)
conf_mat <- confusionMatrix(pred_nb, test_nb$value_bin)
print(conf_mat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Low Medium High
##     Low    134     87   42
##     Medium 145    301  110
##     High   546    432  584
##
## Overall Statistics
##
##                Accuracy : 0.428
##                  95% CI : (0.408, 0.4481)
##     No Information Rate : 0.3465
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.1569
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: Low Class: Medium Class: High
## Sensitivity             0.16242        0.3671      0.7935
## Specificity             0.91710        0.8366      0.4055
## Pos Pred Value          0.50951        0.5414      0.3739
## Neg Pred Value          0.67375        0.7156      0.8144
## Prevalence              0.34649        0.3444      0.3091
## Detection Rate          0.05628        0.1264      0.2453
## Detection Prevalence    0.11046        0.2335      0.6560
## Balanced Accuracy       0.53976        0.6019      0.5995
```

We first transformed each listing's review_scores_value into three equal-sized bins—Low, Medium, and High, so that our response classes were roughly balanced. To prevent any leakage from other satisfaction measures, we excluded all other review_scores_* columns. We then selected a mix of numeric and categorical predictors that seemed most likely to influence perceived value:

- Numeric: log(price), accommodates, bedrooms, bathrooms, beds, number_of_reviews, host_response_rate, host_acceptance_rate
- Categorical: host_response_time, host_is_superhost, host_identity_verified, neighbourhood_cleansed, room_type

After dropping rows with any missing values, then using the same method, we trained a Laplace-smoothed Naïve Bayes model on the 70% training set and checked how well it worked on the 30% test set. The overall accuracy of 42.8% significantly exceeds the no-information rate of 34.7% ($p < .001$), and Cohen's k = 0.157 indicates modest agreement beyond chance.

By class, the model is best at catching High-value stays (sensitivity = 0.79) but struggles with Low-value ones (sensitivity = 0.16), suggesting that features like location, host responsiveness, and price are stronger signals of exceptional value than of poor value.

```
# 1. Build a one-row data.frame matching train_nb's predictors
new_listing <- data.frame(
  host_response_rate = 0.8,
  host_response_time = factor("within an hour",
                   levels = levels(train_nb$host_response_time)),
  host_acceptance_rate = 0.9,
  neighbourhood_cleansed = factor("Indre By",
                                levels = levels(train_nb$neighbourhood_cleansed)),
  room_type = factor("Entire home/apt",
                   levels = levels(train_nb$room_type)),
  accommodates = 2,
  bathrooms    = 1,
  bedrooms     = 1,
  beds         = 1,
  minimum_nights = 2,
  number_of_reviews = 100,
  host_is_superhost = TRUE,
   host_identity_verified = TRUE,
  log_price    = log(1200)  # log of 1 200 DKK
)

# 2. Predict with the trained naive Bayes model
predicted_bin <- predict(model_nb, new_listing)

print(predicted_bin)
```

```
## [1] Medium
## Levels: Low Medium High
```

The model assigns our example listing to the “Medium” value bin, showing that guests with similar stays typically report neither poor nor exceptional value for the price they paid. In other words, this listing sits in the middle third of all Copenhagen rentals when it comes to perceived bang-for-the-buck.

Our hypothetical apartment includes a two-guest, one-bedroom unit in Indre By, priced at DKK 1200 per night, with a host acceptance rate of 90 %, an 80 % on-time response rate, superhost status, and 100 prior reviews, matches the profile of many “average” value stays. The model has learned that moderate pricing combined with solid service indicators and review counts tends to produce a middling perception of value.

To push the perception into the “High” value category, the host could consider either enhancing service (for example, faster or more consistent responses, additional amenities, or personalized touches) or tweaking the nightly rate downward.

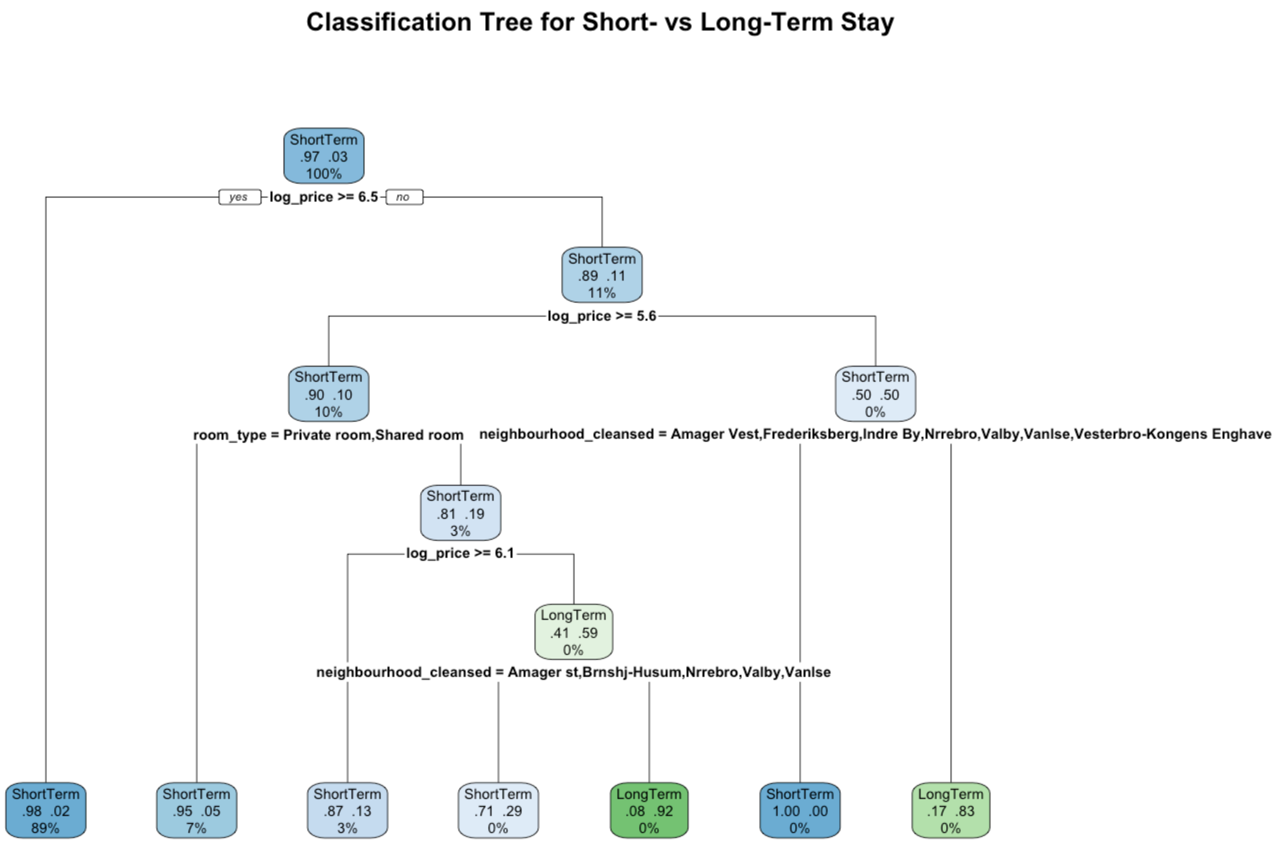
Classification Tree

```
df_tree <- df %>%
  #        minimum_nights
  filter(!is.na(minimum_nights)) %>%
  #          ≤ 30        > 30
  mutate(
    stay_type = factor(
      ifelse(minimum_nights <= 7, "ShortTerm", "LongTerm"),
      levels = c("ShortTerm","LongTerm")
    )
  ) %>%
  select(-minimum_nights,   #
         -maximum_nights,   #           /
         -minimum_minimum_nights,
         -maximum_minimum_nights,
         -minimum_maximum_nights,
         -maximum_maximum_nights)

df_tree <- df_tree %>%
  select(
    stay_type,
    price,
    accommodates,
    bedrooms,
    bathrooms,
    beds,
    number_of_reviews,
    host_response_time,
    host_response_rate,
    host_acceptance_rate,
    host_is_superhost,
    host_identity_verified,
    neighbourhood_cleansed,
    room_type,
    review_scores_rating
  ) %>%
  drop_na() %>%
  mutate(
    host_response_time     = factor(host_response_time),
    host_is_superhost      = factor(host_is_superhost),
    host_identity_verified = factor(host_identity_verified),
    neighbourhood_cleansed = factor(neighbourhood_cleansed),
    room_type              = factor(room_type),
    log_price              = log(price)
  ) %>%
  select(-price)

set.seed(699)

train_idx  <- createDataPartition(df_tree$stay_type, p = 0.7, list = FALSE)
train_tree <- df_tree[train_idx, ]
```

```
test_tree  <- df_tree[-train_idx, ]

# 4.1
tree_fit <- rpart(
  stay_type ~ .,
  data   = train_tree,
  method = "class",
  control = rpart.control(cp = 0.01)  #         cp
)

# 4.2
printcp(tree_fit)
```

```
##
## Classification tree:
## rpart(formula = stay_type ~ ., data = train_tree, method = "class",
##     control = rpart.control(cp = 0.01))
##
## Variables actually used in tree construction:
## [1] log_price              neighbourhood_cleansed room_type
##
## Root node error: 170/5560 = 0.030576
##
## n= 5560
##
##         CP nsplit rel error xerror     xstd
## 1 0.015686      0   1.00000 1.0000 0.075515
## 2 0.010000      6   0.88824 1.0529 0.077423
```

```
# 4.3
rpart.plot(tree_fit,
           type = 2,      #
           extra = 104,   #
           fallen.leaves = TRUE,
           main = "Classification Tree for Short- vs Long-Term Stay")
```



We began by turning each listing's minimum-nights requirement into a simple two-level outcome: "ShortTerm" (≤ 7 nights) versus "LongTerm" (> 7 nights), and then stripped out every other "nights" column so the tree couldn't peek at the raw minimum/maximum values.

After converting all textual fields (neighborhood, room type, host response time, etc.) into factors and taking the natural log of price, we split the data 70/30 (with seed 699) and fitted an rpart classification tree. We used cross-validation to scan complexity parameters (cp) from 0.01 downward, then pruned at the cp value (0.0157) that minimized the cross-validated error.

```
#
pred_tree <- predict(tree_fit, test_tree, type = "class")

#
conf_mat_tree <- confusionMatrix(pred_tree, test_tree$stay_type)
print(conf_mat_tree)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  ShortTerm LongTerm
##   ShortTerm      2306       65
##   LongTerm          3        7
##
##                Accuracy : 0.9714
##                  95% CI : (0.9639, 0.9778)
##     No Information Rate : 0.9698
##     P-Value [Acc > NIR] : 0.3434
##
##                   Kappa : 0.1646
##
##  Mcnemar's Test P-Value : 1.389e-13
##
##             Sensitivity : 0.99870
##             Specificity : 0.09722
##          Pos Pred Value : 0.97259
##          Neg Pred Value : 0.70000
##              Prevalence : 0.96976
##          Detection Rate : 0.96850
##    Detection Prevalence : 0.99580
##       Balanced Accuracy : 0.54796
##
##        'Positive' Class : ShortTerm
##
```

Our full tree (before pruning) consistently split first on log(price)—listings above about DKK 665/night were almost always short-term stays—then dove into room type (private/shared rooms tend to be short-term) and even specific neighborhoods for the cheapest entire-place rentals. But when we trimmed back to the “best” cp, the tree collapsed to a single node predicting “ShortTerm” for all listings. On our held-out 30% test set, it reported 97.1% overall accuracy, which is essentially identical to the 96.98% you’d get by always guessing “ShortTerm.” Sensitivity for short-term stays was nearly 100%, but specificity for long-term was under 10%, yielding a balanced-accuracy of just 55%.

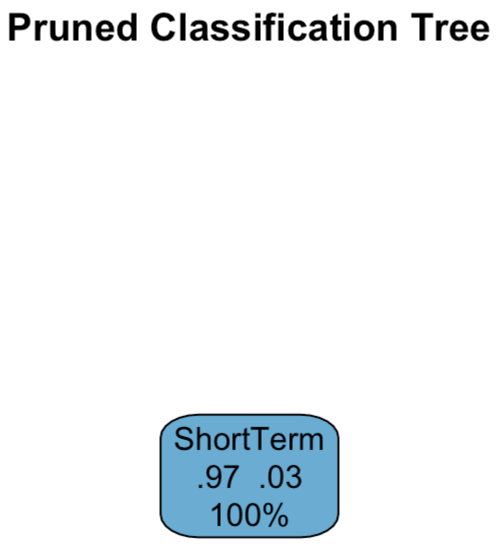
```
printcp(tree_fit)
```

```
##
## Classification tree:
## rpart(formula = stay_type ~ ., data = train_tree, method = "class",
##     control = rpart.control(cp = 0.01))
##
## Variables actually used in tree construction:
## [1] log_price              neighbourhood_cleansed room_type
##
## Root node error: 170/5560 = 0.030576
##
## n= 5560
##
##         CP nsplit rel error xerror     xstd
## 1 0.015686      0   1.00000 1.0000 0.075515
## 2 0.010000      6   0.88824 1.0529 0.077423
```

```
opt_index <- which.min(tree_fit$cptable[, "xerror"])
opt_cp    <- tree_fit$cptable[opt_index, "CP"]

cat("       CP      ", opt_cp, "\n")
```

```
##        CP       0.01568627
```

```
pruned_tree <- prune(tree_fit, cp = opt_cp)

rpart.plot(pruned_tree,
           type = 2,
           extra = 104,
           fallen.leaves = TRUE,
           main = "Pruned Classification Tree")
```



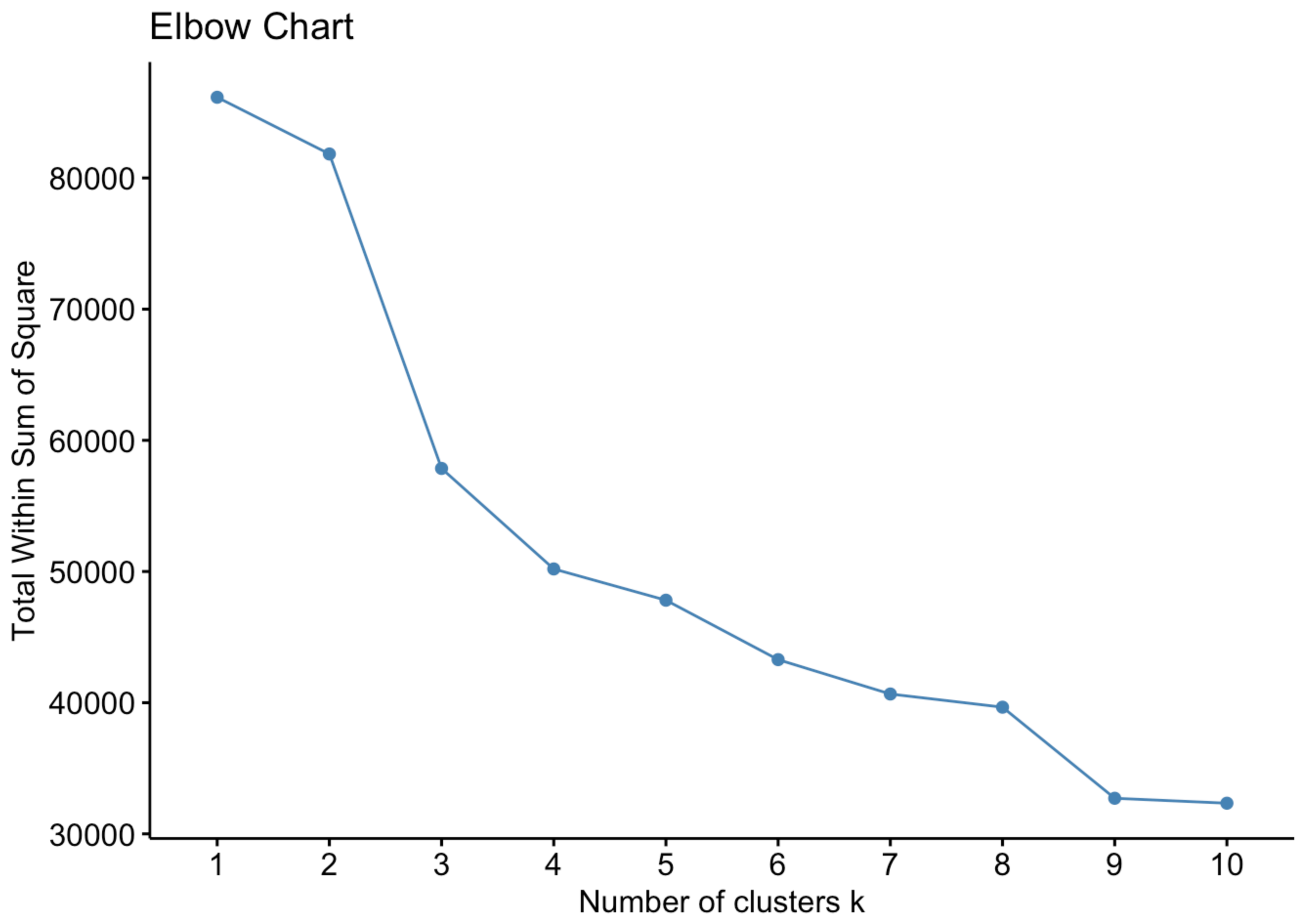
Looking at the result here, it is safe to say that Price is the strongest single signal for distinguishing short-versus long-term rentals. But with such a heavy class imbalance, 97% of the listings in our sample are short-term by our definition (≤7 nights), so the cost of making any further splits (on price, room type, etc.) outweighs the tiny gain in correctly identifying the 3% that are long-term. The pruned tree therefore collapses to the trivial majority-class classifier: always guess ShortTerm.

Step IV: Clustering

```
cluster_df <- df %>%
  select(
    price,
    accommodates,
    bedrooms,
    bathrooms,
    beds,
    minimum_nights,
    number_of_reviews,
    review_scores_rating
  ) %>%
  drop_na() %>%
  mutate(log_price = log(price)) %>%
  select(-price)

set.seed(699)

cluster_scaled <- scale(cluster_df)
```

We chose price, accommodates, bedrooms, bathrooms, beds, minimum_nights, number_of_reviews and review_scores_rating as our variables. As can be seen from the previously plotted box plots of prices, there are many outliers and these outliers have a large impact on the clustering. So we take logarithm of price before using K-means as a way to eliminate the effect of extreme values and use log_price instead of price. Finally, we normalize these variables.

```
fviz_nbclust(cluster_scaled, kmeans, method = "wss") +
  labs(title = "Elbow Chart")
```



Before formal clustering, we plotted an Elbow Chart for determining the value of k. From the Elbow Chart, it is evident that the Total Within Sum of Squares (WSS) decreases sharply when increasing the number of clusters from 2 to 3, indicating a significant improvement in clustering performance. After K=3, the rate of decrease in WSS slows down considerably, forming a clear “elbow” shape. Thus, we chose K=3.

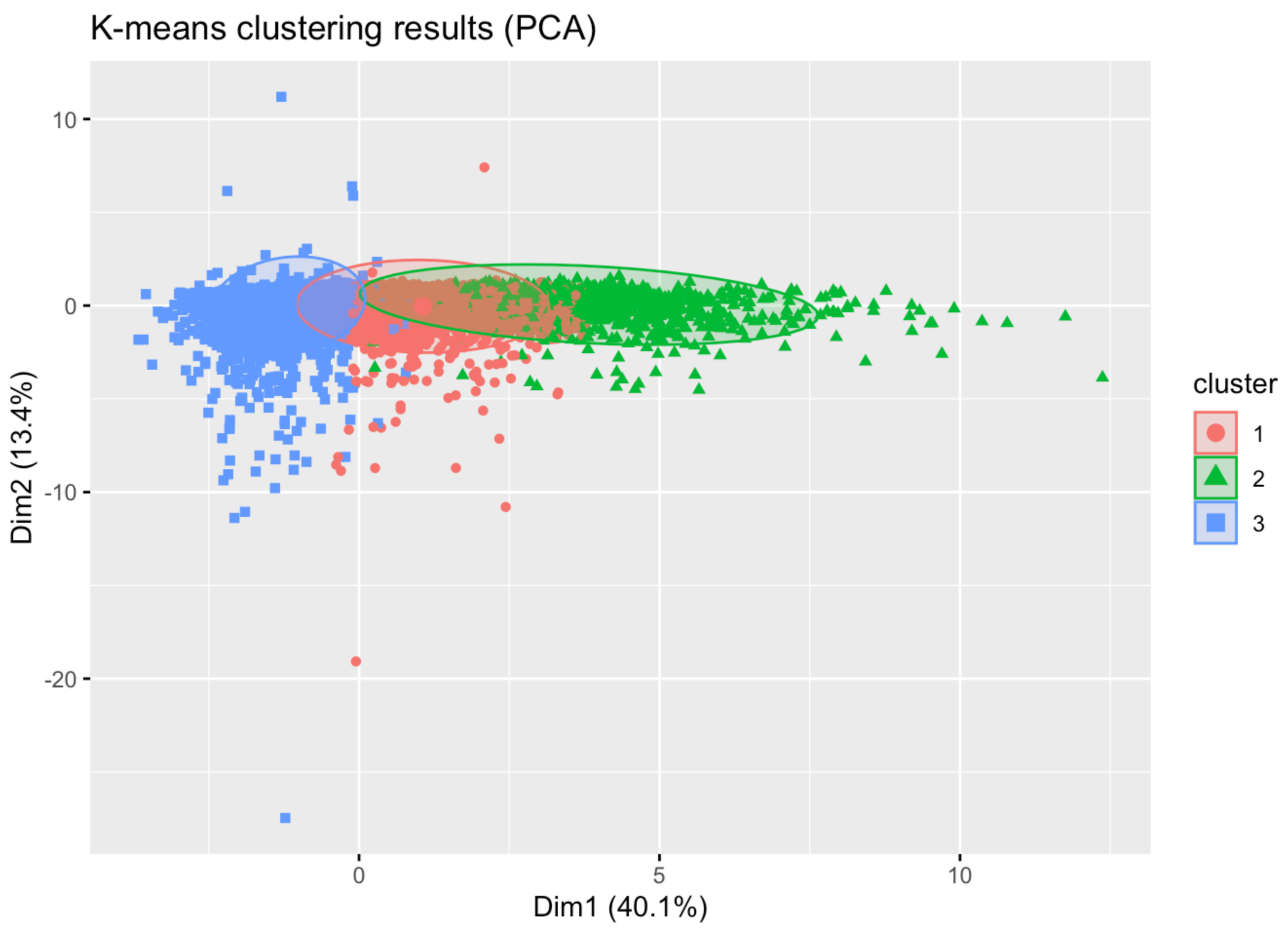
```
# K-means
k <- 3
km_res <- kmeans(cluster_scaled, centers = k, nstart = 25)

#
df_kmeans <- cluster_df %>%
  mutate(cluster = factor(km_res$cluster))

#     PCA
pca_res <- prcomp(cluster_scaled, center = TRUE, scale. = TRUE)

#
summary(pca_res)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     1.7919 1.0366 1.0010 0.9789 0.85960 0.75520 0.50726
## Proportion of Variance 0.4014 0.1343 0.1252 0.1198 0.09236 0.07129 0.03216
## Cumulative Proportion  0.4014 0.5357 0.6609 0.7807 0.87309 0.94438 0.97654
##                            PC8
## Standard deviation     0.43319
## Proportion of Variance 0.02346
## Cumulative Proportion  1.00000
```

```
#           PCA        Loadings
pca_res$rotation
```

```
##                               PC1          PC2          PC3          PC4
## accommodates          0.507477016 -0.056555401  0.005444997  0.032102303
## bedrooms              0.505924467 -0.003840606  0.018676434  0.009694798
## bathrooms             0.332042566  0.013938186  0.037714295 -0.076076410
## beds                  0.472196680 -0.121007124  0.024537613 -0.028626208
## minimum_nights       -0.007183963  0.188720317  0.944326442 -0.258826774
## number_of_reviews    -0.003891318 -0.709987534 -0.052700959 -0.670657431
## review_scores_rating  0.008939206  0.635981610 -0.313573320 -0.687803113
## log_price             0.391329632  0.194369426 -0.068840635  0.049219530
##                              PC5         PC6          PC7         PC8
## accommodates          0.16219921  0.12401630  0.373086585 -0.74654279
## bedrooms              0.03551632  0.21977773  0.521542961  0.64962464
## bathrooms            -0.91323135 -0.18199382 -0.107456877 -0.06068976
## beds                  0.16997399  0.42873060 -0.737244871  0.06881088
## minimum_nights        0.06729556 -0.02999241  0.002821848 -0.01237357
## number_of_reviews     0.07094956 -0.18706944  0.048756743  0.03062257
## review_scores_rating  0.03002787  0.14664147  0.006070546 -0.04004855
## log_price             0.31477476 -0.81362632 -0.176918981  0.09771787
```

```
#              PCA
fviz_cluster(km_res, data = cluster_scaled,
             geom = "point", ellipse.type = "norm",
             main = "K-means clustering results (PCA)")
```



```
#            cluster
df_kmeans %>%
  group_by(cluster) %>%
  summarise(
    sample_num = n(),
    avg_price = exp(mean(log_price, na.rm = TRUE)),  #
    avg_score = mean(review_scores_rating, na.rm = TRUE),
    avg_accommodates = mean(accommodates, na.rm = TRUE),
    avg_bedrooms = mean(bedrooms, na.rm = TRUE),
    avg_bathrooms = mean(bathrooms, na.rm = TRUE),
    avg_beds = mean(beds, na.rm = TRUE),
    avg_minimum_nights = mean(minimum_nights, na.rm = TRUE),
    avg_num_reviews = mean(number_of_reviews, na.rm = TRUE)
  )
```

cluster	sample_n...	avg_price	avg_score	avg_accommodat...	avg_bedroo...	avg_bathrooms	av
<fct>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
1	3409	1563.2319	4.817436	4.494573	2.166618	1.022441	2.
2	1025	2153.0175	4.855268	6.237073	3.267317	1.842927	3.
3	6336	933.3401	4.831037	2.298769	1.020991	1.013573	1.

3 rows | 1-8 of 10 columns

We use k=3 to divide the data into 3 clusters using k-means and add the cluster labels back to the dataframe. By analyzing the explained variance of the principal components and the Loadings matrix, we use PCA for dimensionality reduction to visually interpret the three clusters.

From the explanation of the principal components, we can see that PC1 explains 40.14% of the variance and PC2 explains 13.43%. Together, they account for most of the variability in the data. The variables that contribute significantly to PC1 (with large absolute coefficients) include accommodates, bedrooms, beds, log_price, and bathrooms, indicating that PC1 mainly represents the size and price level of the properties. The variables that contribute significantly to PC2 include number_of_reviews and review_scores_rating, suggesting that PC2 mainly represents the popularity and rating of the properties.

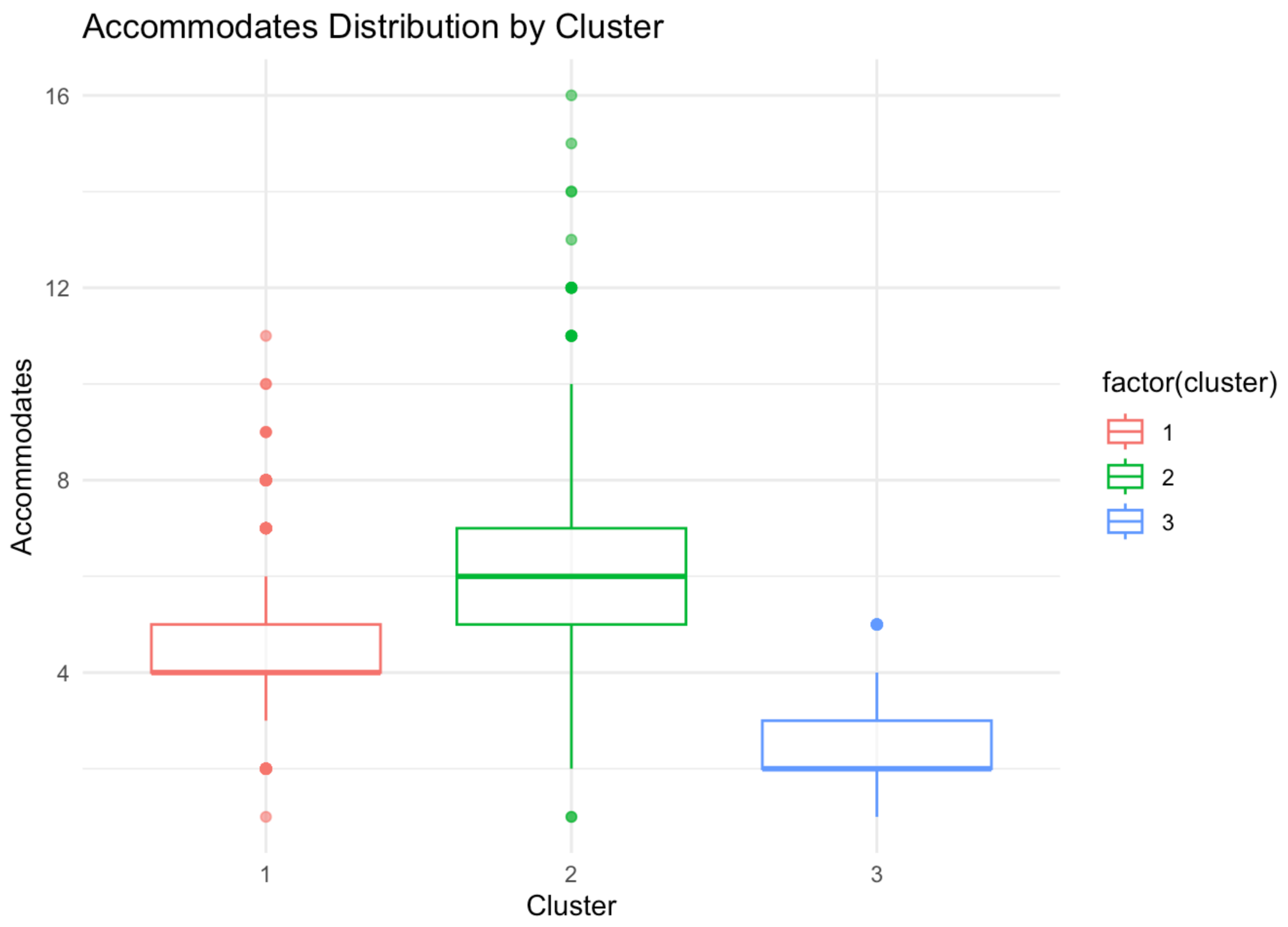
Therefore, the three clusters can be described as follows:

Cluster 1 (red): Medium-sized properties with moderate ratings and size.

Cluster 2 (green): Large/high-price properties with significant individual differences.

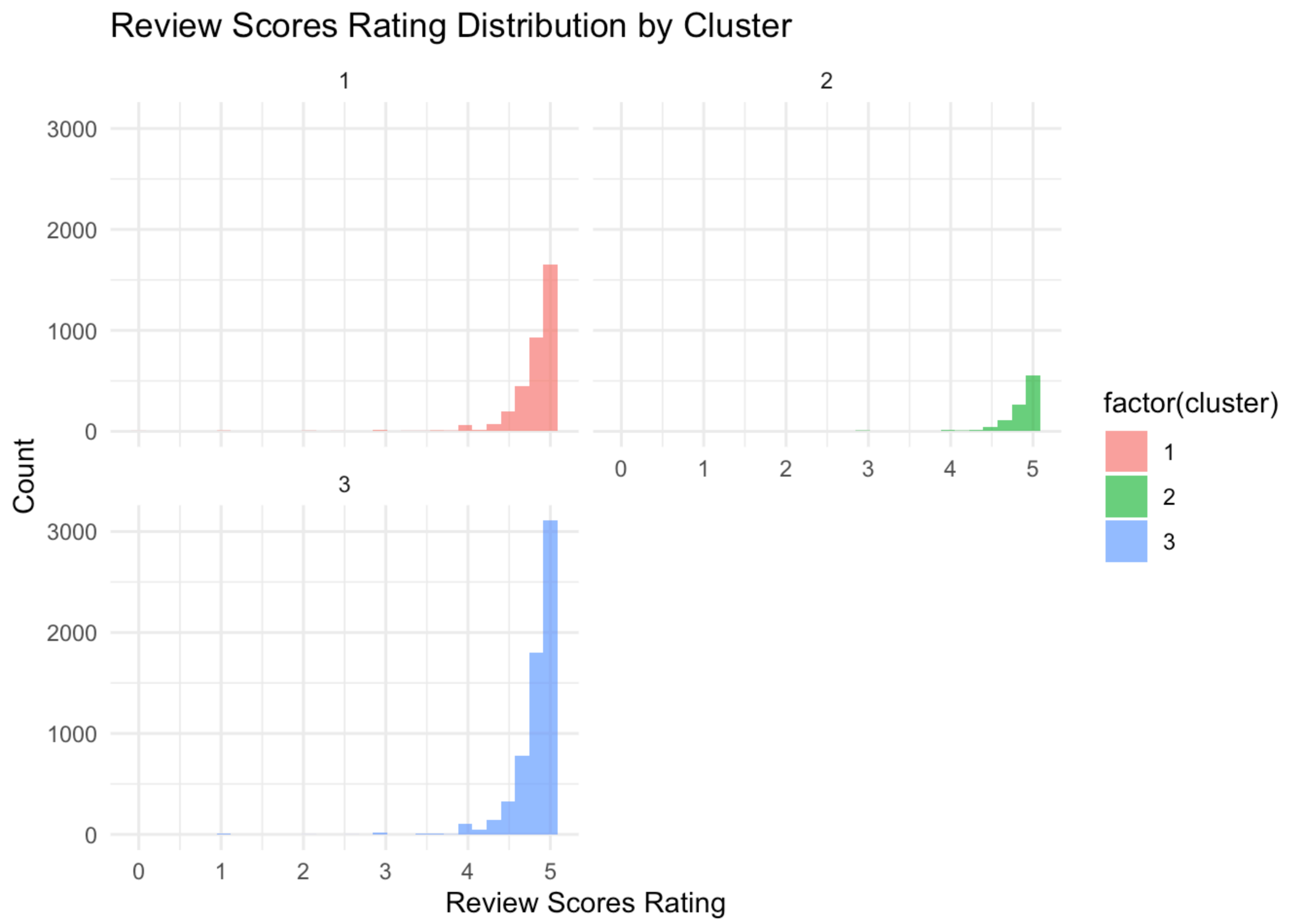
Cluster 3 (blue): Small/low-price properties, very concentrated.

```
ggplot(df_kmeans, aes(x = factor(cluster), y = accommodates, color = factor(cluster))) +
  geom_boxplot(alpha = 0.6) +
  labs(title = "Accommodates Distribution by Cluster", x = "Cluster", y = "Accommodate
s") +
  theme_minimal()
```



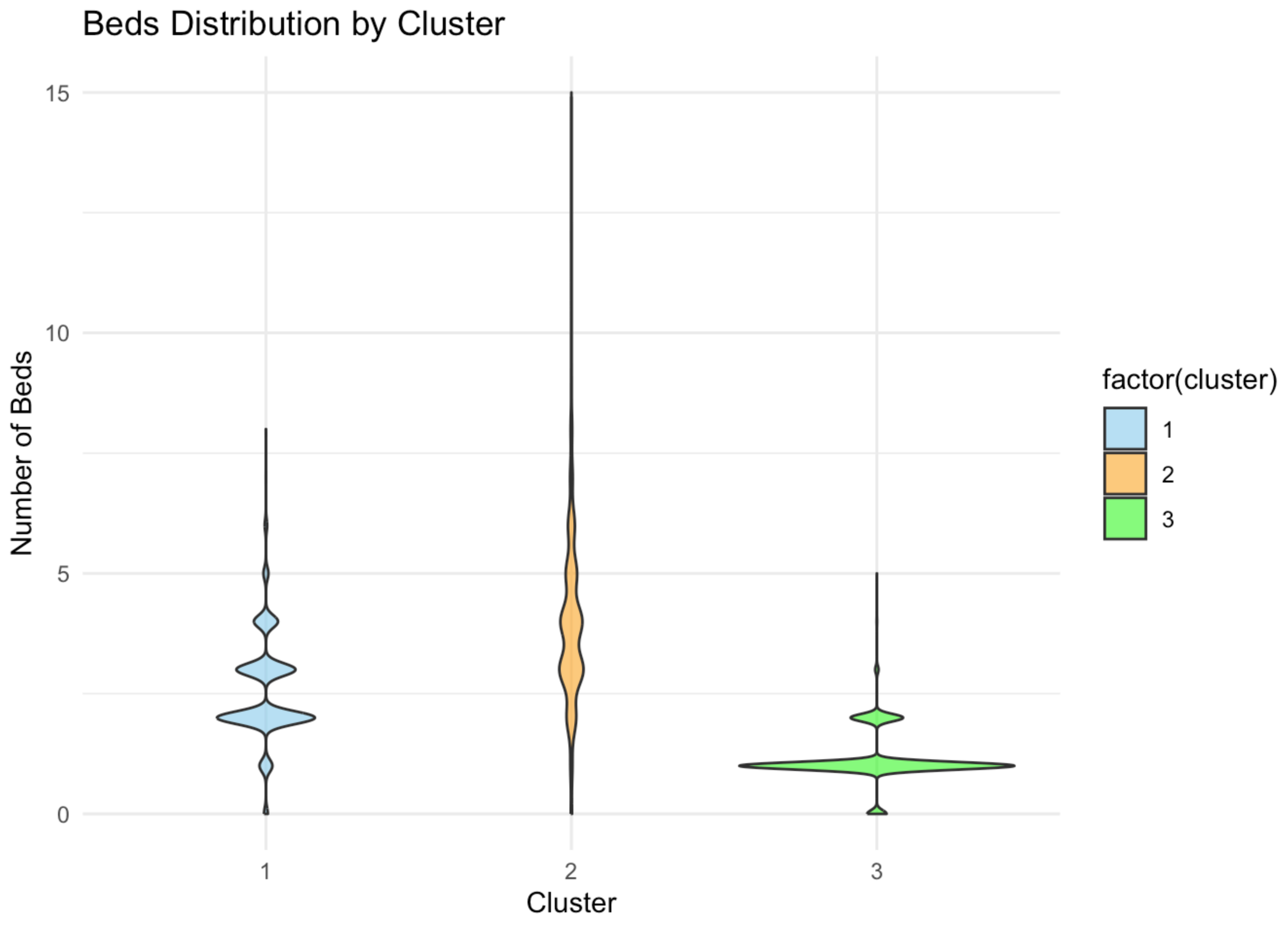
The box plot illustrates the distribution of accommodation capacities across three clusters. Cluster 2 shows the widest variation in the number of people accommodated (typically 4-14, with some up to 16), while cluster 1 generally accommodates 4-6 people and cluster 3 typically hosts 2-4 people.

```
#
ggplot(df_kmeans, aes(x = review_scores_rating, fill = factor(cluster))) +
  geom_histogram(bins = 30, position = "dodge", alpha = 0.7) +
  labs(title = "Review Scores Rating Distribution by Cluster", x = "Review Scores Ratin
g", y = "Count") +
  theme_minimal() +
  facet_wrap(~cluster, ncol = 2)
```



The histogram shows the distribution of review scores ratings across three clusters. All clusters have most reviews concentrated in the 4.5-5 range, but cluster 3 has the highest count of 5-star ratings, followed by cluster 1 and then cluster 2.

```
ggplot(df_kmeans, aes(x = factor(cluster), y = beds, fill = factor(cluster))) +
  geom_violin(alpha = 0.6) +
  labs(title = "Beds Distribution by Cluster", x = "Cluster", y = "Number of Beds") +
  theme_minimal() +
  scale_fill_manual(values = c("skyblue", "orange", "green"))
```



The violin plot shows the distribution of the number of beds across three clusters. Cluster 2 has the widest variation in bed numbers, ranging from near 0 to 15 with a mid-concentration of 4-6. Cluster 1 typically has 1-4 beds, while cluster 3 mostly has 1-2 beds. Cluster 3 shows the most consistent bed counts.

Conclusion

With the initial and raw dataset, we approached it by cleaning and preparing it, then we ran descriptive analyses and plotting visualizations, fit a log-linear price model, carried out three classification tasks (Wi-Fi, perceived value, stay length), and distilled listings into three clusters—always balancing clarity with rigor.

Throughout the whole process, we found that location commands a notable price premium, Inner City listings run roughly 40–50% above the citywide baseline, while unit characteristics (bedrooms, bathrooms, accommodates) each add another 10–15% on the log scale. Meanwhile, Wi-Fi turns out to be nearly universal (97% of listings), so simple models can't improve on the trivial "yes" prediction. Moreover, perceived-value bins (low/medium/high) proved modestly predictable, our Naive Bayes model achieved 43% accuracy versus a 35% baseline—suggesting that price, reviews, and host responsiveness drive "high value" marks. Finally, clustering distilled three clear market segments: high-end, review-rich core-city rentals; mid-range family/large-group homes; and budget stays on the periphery.

As a result, hosts and property managers can benchmark their own rates, amenities, and minimum-stay rules against neighborhood peers to fine-tune their positioning—whether they aim to undersell or command a premium. Similarly, guests gain clearer expectations around cost, location, and service quality when choosing between central and suburban stays. In addition, Airbnb itself could integrate these insights into search rankings (promote listings with high predicted value scores) or automated alerts (notify hosts whose response rates fall below neighborhood medians). Beyond that, urban planners and tourism boards can leverage our cluster maps and word-cloud overviews to identify under-served or over-served areas, thereby helping to balance short-term rental growth with broader affordable-housing goals.

Looking ahead, although our models explain a substantial share of observable variation, there remains untapped signal in interior features, seasonality effects, and dynamic demand curves. Therefore, future work could layer in time-series calendar data, image-based quality metrics, or third-party amenity indexes to sharpen predictions—particularly for rarer cases like non-Wi-Fi listings or long-term stays. Nonetheless, this project demonstrates that even straightforward, transparent techniques can yield actionable insights into a complex urban lodging market.