Machine Learning for Predicting Airbnb Listings in New York

41708, 42244, 40681

###Introduction

For many people, price is the most important factor which determines where they stay. Be it the flat they rent, the home they own, or the Airbnb they spent their holiday in. Price is always the first thing to consider, but there are also preconditions and preferences that are individual to the person: for example, how many bedrooms might one need to fit their family? Or: how close do they want to be to public transport? For anyone renting a property, then, there are a multitude of factors at play, and it is up to the individual to decide what they value most.

Airbnb is an online marketplace, where hosts can rent out their properties to travelers or for long term stays. The host can name their price and, if that price is attractive, a guest can immediately book any available days. This creates a system where, theoretically, listings are efficiently priced with respect to the whole market, including other rental properties and hotels. However, it may take considerable time for a host to find the sweet spot of frequent bookings and good profit. For the guest, the problem is one of utility: to find the listing at the best price that satisfies all of their needs, and has any extra characteristics that they are looking for. Again, this will take time - there are thousands of listings in any given city, and weighing each of these against hotels and hostels conjures the image of a mildly incompetent pastry chef, who just can't get their measurements right.

In this study, we propose a neat simplification of these two tasks, that of the host, and that of the guest. Through machine learning methods, we aim to create meaningful models for the price of Airbnb listings, which can be used to ascertain the reasonable listing price of a property/room, given the current market. By factoring in essential variables like length of stay, as well as nice-to-haves such as amenities, we will create models that can provide a baseline nightly price for staying in an Airbnb; but also, inform guests on the marginal price increases of their nice-to-haves. Additionally, they should give hosts a useful starting point to price their listing, such that they are immediately competitive in the market.

Machine learning diverges into several subsections for predicting and finding relationships in data. For our purpose, we employ supervised learning, specifically regression, with listing ($ per night) price as the outcome. By varying the complexity, dimensionality and drivers of regression, we hope to generate both models with strong predictive power and, crucially, models which are easy to understand for potential users. Thus, we will fit models using: OLS, decision trees, gradient descent, ridge regression and a random forest. The discussion of each model will focus around motivation, tuning, findings and limitations.

##Dataset

Our data comes from Inside Airbnb (https://insideairbnb.com/get-the-data/), specifically listing data on New York City, which was updated on the 1st of May 2025 (though most listings were last scraped from Airbnb's official website at the start of March). It contains the title and description of 37,434 active Airbnb listings, alongside information on the host (no. listings, time on the site), and 40 useful listing characteristics (bedrooms, property/room type, neighbourhood, maximum length of stay, reviews, etc.). By manipulating, and exploring this data, we should gain valuable insights which will motivate later modelling choices. After all models have been fit, we consider the scope and limitations of our project overall, what we learned from our modelling efforts and, finally, conclusions. So, with that, our (somewhat lengthy) section on data cleaning and manipulation begins.

###Data manipulation

Throughout our modelling, we used the package 'tidymodels' to produce clean, easy to follow workflows. Functions from tidymodels were the principal tools used in modelling.

As discussed, our dataset consists of 37,434 observations of Airbnb listings in New York City. In the chunk below, we load the data, create missing levels and turn price, our outcome variable, into a number. If we immediately run na.omit on this dataframe, we find that only 10% of observations are complete cases, but, thankfully, there are many redundant and useless predictors which have a high number of values missing.

```
#bnb_data <- read.csv(file.choose())
bnb_data <- read.csv("listings.csv")
bnb_data <- as.data.frame(bnb_data)
#summary(bnb_data)

# create NA: only 4128 complete cases after removing columns full of NAs
bnb_data[bnb_data == ""] <- NA
bnb_data[bnb_data == "N/A"] <- NA

# make price a number (remove $ sign)
bnb_data$price <- as.numeric(gsub("[$,]", "", bnb_data$price))
```

Next, we checked correlations within the data, with the (admittedly unnecessary) 'cor.checker' function. No variable was highly correlated with price, with the highest absolute values being .4 for estimated revenue in the last year and number of guests the listing takes. However, several predictors were similar to, or the same as, each other. For example, review scores generally had >0.7 correlation with each other. Looking at 'full_correlations', we can find more groups, like the three host listings count predictors.

```
cor.checker <- function(predictors = c(names(bnb_data)), data = bnb_data) {
  to_check <- c()

  # correlation can be checked if column is numeric, with some variance
  for (col in predictors) {
    if (is.numeric(data[[col]])) {
      if(var(data[[col]], na.rm = TRUE) > 0) {
         to_check <- c(to_check, col)
    } }
  }

  cor(data[, to_check], use = "complete.obs")
}

# keep rating, location, checkin based on <0.7 correlation with each other
cor.checker(c("review_scores_rating", "review_scores_accuracy",
         "review_scores_cleanliness", "review_scores_checkin",
         "review_scores_communication", "review_scores_location",
         "review_scores_value"))
```

```
##                            review_scores_rating review_scores_accuracy
## review_scores_rating                  1.0000000              0.8015071
## review_scores_accuracy                0.8015071              1.0000000
## review_scores_cleanliness             0.7236020              0.6750966
## review_scores_checkin                 0.6680185              0.6661002
## review_scores_communication           0.7365299              0.7048916
## review_scores_location                0.5642527              0.5394525
## review_scores_value                   0.8095958              0.7691530
```

```
##                             review_scores_cleanliness review_scores_checkin
## review_scores_rating                        0.7236020             0.6680185
## review_scores_accuracy                      0.6750966             0.6661002
## review_scores_cleanliness                   1.0000000             0.5514071
## review_scores_checkin                       0.5514071             1.0000000
## review_scores_communication                 0.5492820             0.6834782
## review_scores_location                      0.4420918             0.5072147
## review_scores_value                         0.6528169             0.6173821
##                             review_scores_communication review_scores_location
## review_scores_rating                          0.7365299              0.5642527
## review_scores_accuracy                        0.7048916              0.5394525
## review_scores_cleanliness                     0.5492820              0.4420918
## review_scores_checkin                         0.6834782              0.5072147
## review_scores_communication                   1.0000000              0.5016628
## review_scores_location                        0.5016628              1.0000000
## review_scores_value                           0.6911021              0.5587767
##                             review_scores_value
## review_scores_rating                  0.8095958
## review_scores_accuracy                0.7691530
## review_scores_cleanliness             0.6528169
## review_scores_checkin                 0.6173821
## review_scores_communication           0.6911021
## review_scores_location                0.5587767
## review_scores_value                   1.0000000
```

```
#cor.checker(c("estimated_revenue_l365d", "price"))
#cor.checker(c("beds", "bedrooms"))

# use to check correlations with price (mostly low, bed/bathrooms .3, est_revenue .4, accomodates .4)
full_correlations <- cor.checker(c(names(bnb_data)))
```

We will return to deal with the correlated variables after creating the ridge regression dataset. For now, we remove predictors, and observations, which will not help towards any modelling effort. These include observations with no associated price; URL & nominal predictors and empty columns like number of reviews in the last year. Other removed columns, similarly, did not contain enough information. For example, 'host_locations' had many levels, but almost all of them were variations of "New York" or "United States". Indeed, any location outside of NY might only appear once, making estimating coefficients for these levels unproductive.

```
# trimming the data
bnb_data <- bnb_data %>%
  filter(!is.na(price)) %>% # observations with a price

  # URL & data generation related columns
  select(-ends_with("url")) %>%
  select(-contains("scrape")) %>%
  select(-c(id, source, calendar_updated)) %>%

  # don't contain enough information to be used reliably
  select(-c(bathrooms_text, license, has_availability,     # some baths are listed as private or shared
                                                           #while some are just the number
            number_of_reviews_l30d, number_of_reviews_ly,  # <10% have at least one review in the
                                                           #last month, ly is empty
            host_identity_verified, host_has_profile_pic,
```

```
            host_location, host_verifications,             # locations are US/NY, verification all true
            host_listings_count, neighbourhood))
```

#Variable names.

```
# some renaming for consistency / convenience
bnb_data <- bnb_data %>%
  rename(
    neighborhood_about = neighborhood_overview,
    host_neighborhood = host_neighbourhood,
    neighborhood = neighbourhood_cleansed,
    borough = neighbourhood_group_cleansed,
    review_count = number_of_reviews,
    review_count_ly = number_of_reviews_ltm, # in the last year
    est_occupancy_ly = estimated_occupancy_l365d,
    est_revenue_ly = estimated_revenue_l365d,
    stars = review_scores_rating,
    stars_checkin = review_scores_checkin,
    stars_location = review_scores_location,
    host_listings_count = calculated_host_listings_count
    )
```

Another minor issue with the dataset is the lack of normalisation in column types. To this end, we define columns as factors, TRUE/FALSE or date as required. Further, in an initial effort to incorporate date predictors, we set them 'as.numeric', which converts a date to the number of days it was since January 1st 1970.

```
# a couple of quick fixes (removing %: 100% -> 100)
for (col in c("host_response_rate", "host_acceptance_rate")) {
  bnb_data[[col]] <- as.numeric(gsub("[,%]", "", bnb_data[[col]]))
}

factor_preds <- c("host_response_time", "host_neighborhood",
                  "neighborhood", "borough", "property_type",
                  "room_type", "bathrooms", "bedrooms")

boolean_preds <- c("host_is_superhost", "instant_bookable")

date_preds <- c("host_since", "first_review", "last_review")

# update column types
bnb_data <- bnb_data %>%
  mutate(
    across(all_of(factor_preds), ~ as.factor(.)), # add factors
    across(all_of(boolean_preds), ~ ifelse(. == "t", TRUE, FALSE)), # true/false
    across(all_of(date_preds), ~ as.numeric(as.Date(., format = "%Y-%m-%d")))
  ) # dates to numbers for regression

#summary(bnb_data)
```

Below, we find a key feature of our preprocessing. By removing observations which have no value for 'stars', which is the average review score, we define a validation step: a listing must have at least one review to be

considered in our models. Despite the vast majority of listings coming from verified hosts, we argue that any listing without one review could be operating outside of the market system, in that it may be priced irrationally. This is not a perfect fix, in fact, one could argue that some listings without a review are the most modern, up-to-date representations of the market, if they are newly listed. Nonetheless, we prefer to consider those listings which have spent some time up on Airbnb, such that they're price could have been adjusted to their theoretical 'optimal' level.

Also included in this chunk is the removal of those columns with the most missing values (>3000). The three 'host_response' predictors were also removed: they each had 2000-3000 missing values and, following testing within the linear model, had fairly insignificant effects on the outcome. Thus, we remove them in an effort to consider more observations. A possible improvement on this process would be imputing some missing values, but given their insignificance, we elected not to go down this path.

```
# listings with one rating
bnb_data <- bnb_data %>%
  filter(!is.na(stars))

na_counts <- colSums(is.na(bnb_data))
na_head <- sort(na_counts, decreasing = TRUE)[1:15]
na_head
```

```
##                host_about       neighborhood_about         host_neighborhood
##                      6320                     6087                      3372
##        host_response_time       host_response_rate     host_acceptance_rate
##                      2950                     2950                      2049
##         host_is_superhost              description                      beds
##                       296                      244                        55
##                  bedrooms                host_name                host_since
##                        23                       17                        17
## host_total_listings_count                bathrooms           stars_location
##                        17                        5                         1
```

```
# high NA
bnb_data <- bnb_data %>%
  select(-c(neighborhood_about, host_about)) %>% # rm high na (>3000) columns
  select(-c(host_neighborhood)) %>%              # LM shows none of the medium na columns are essential
  select(-c(host_response_rate, host_response_time, host_acceptance_rate))

full_correlations <- cor.checker() # no notable change
```

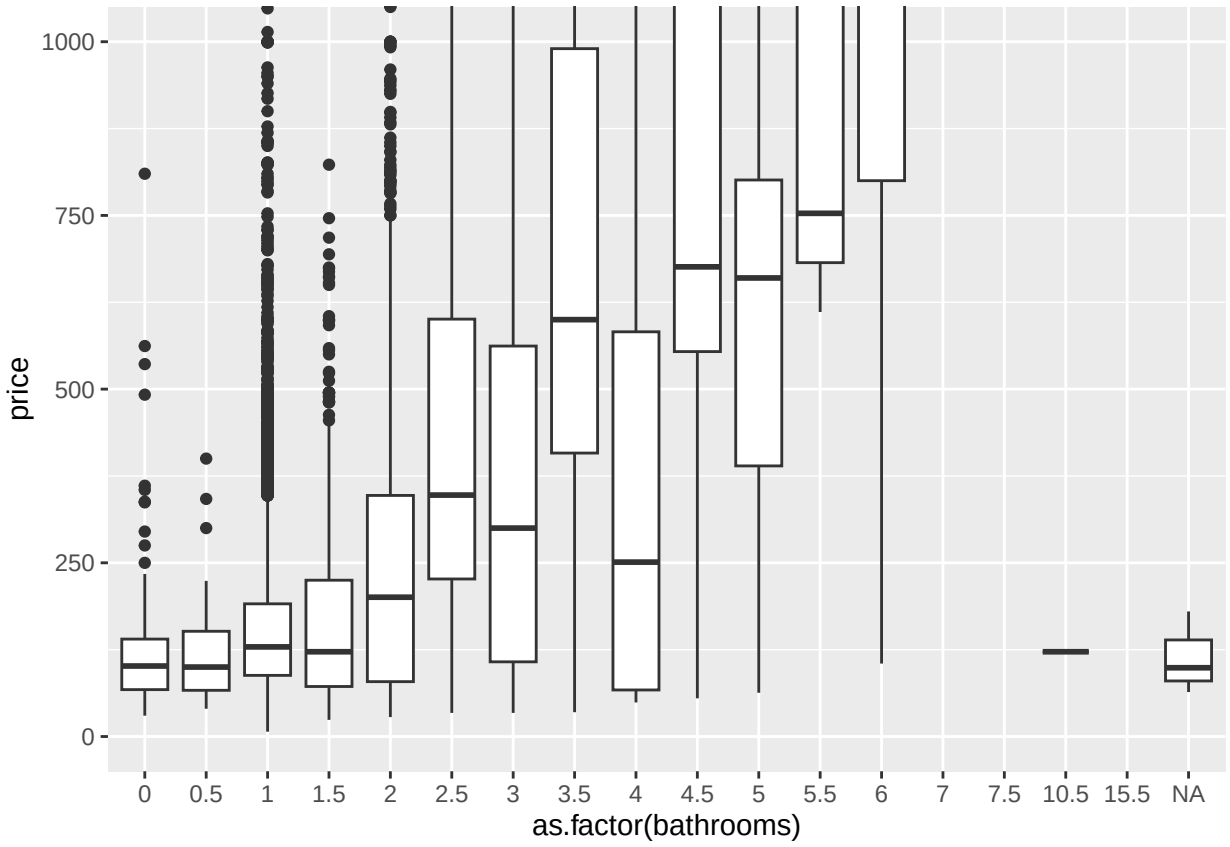
Now, a couple of plot functions, which will be used further during our EDA. For now, observe that the predictor name can be used directly in the function, and that dataset, outcome and axis height can be selected. For 'scatter.plot' a smoothed line of best fit can be added.

```
# predictor is what you plot, yaxis is the height of the axis
box.plot <- function(predictor, data = bnb_data2, yaxis = 1000, outcome = price) {
  ggplot(data, aes(x = as.factor({{ predictor }}), y = {{ outcome }})) +
    # {{.}} lets you put the predictor in without quotes
    geom_boxplot() +
    coord_cartesian(ylim = c(0,yaxis))
}

# loess if you want a LOBF
```

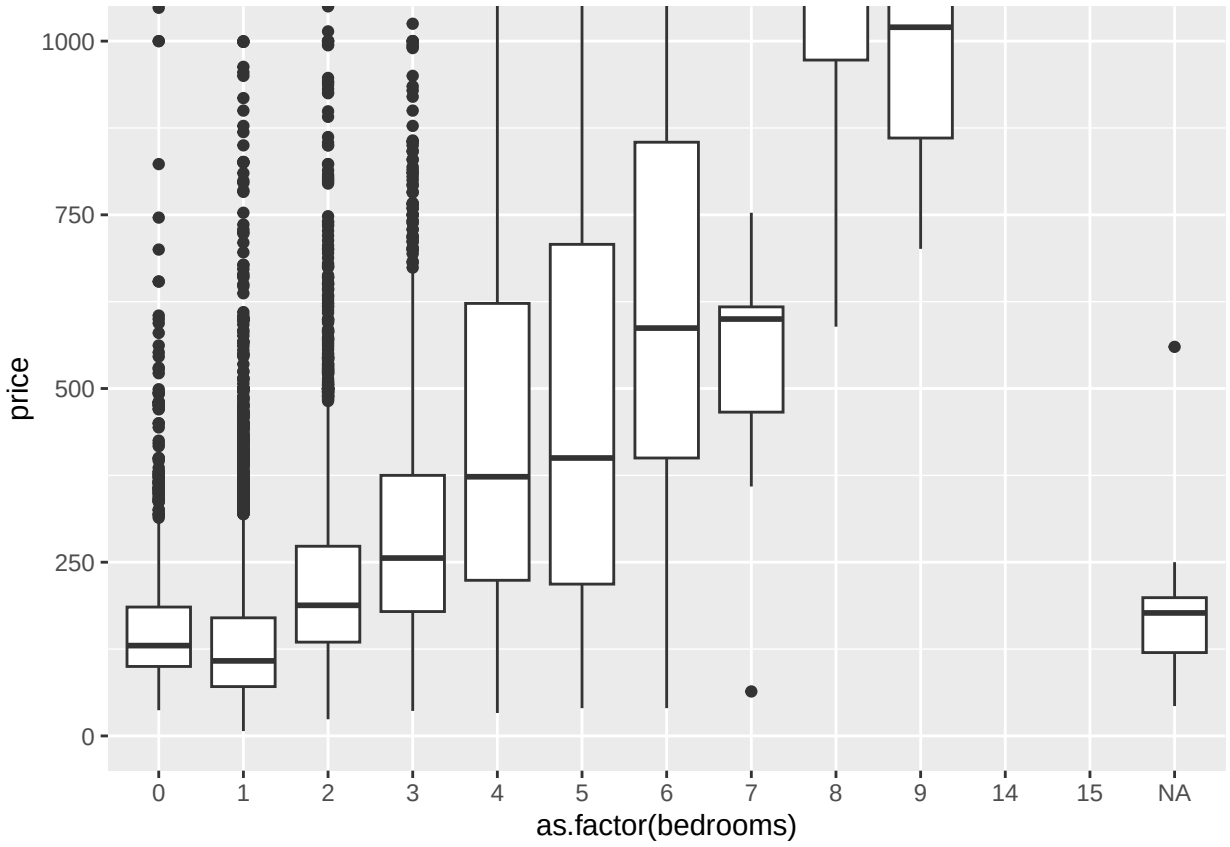
```
scatter.plot <- function(predictor, data = bnb_data2, yaxis = 1500,
                         loess = FALSE, outcome = price) {
  plot <- ggplot(data, aes(x = {{ predictor }}, y = {{ outcome }})) +
    geom_point() +
    coord_cartesian(ylim = c(0, yaxis)) +
    theme_minimal()

  # if loess = TRUE
  if (loess) {
    plot <- plot + geom_smooth(method = "loess", colour = "blue")
  }

  return(plot)
}
```

Observing the boxplots for bedrooms and bathrooms, we cluster levels together, to create a more easily interpreted predictor. For instance: bathrooms '0', '0.5', '1', and '1.5' all have median prices around 120, so we create the new level '<2' Additionally, using the level '4+' combats the lack of listings with 4 or more bed/bathrooms. Only 350/15000 observations have 4+ bedrooms, and around 100 have 5+. A similar process was followed for 'accommodates'. We also merged 'borough', since S.I. and Bronx were very similar in our regression. Importantly, these changes were implemented after the EDA, but it is necessary to include them in 'bnb_data'.

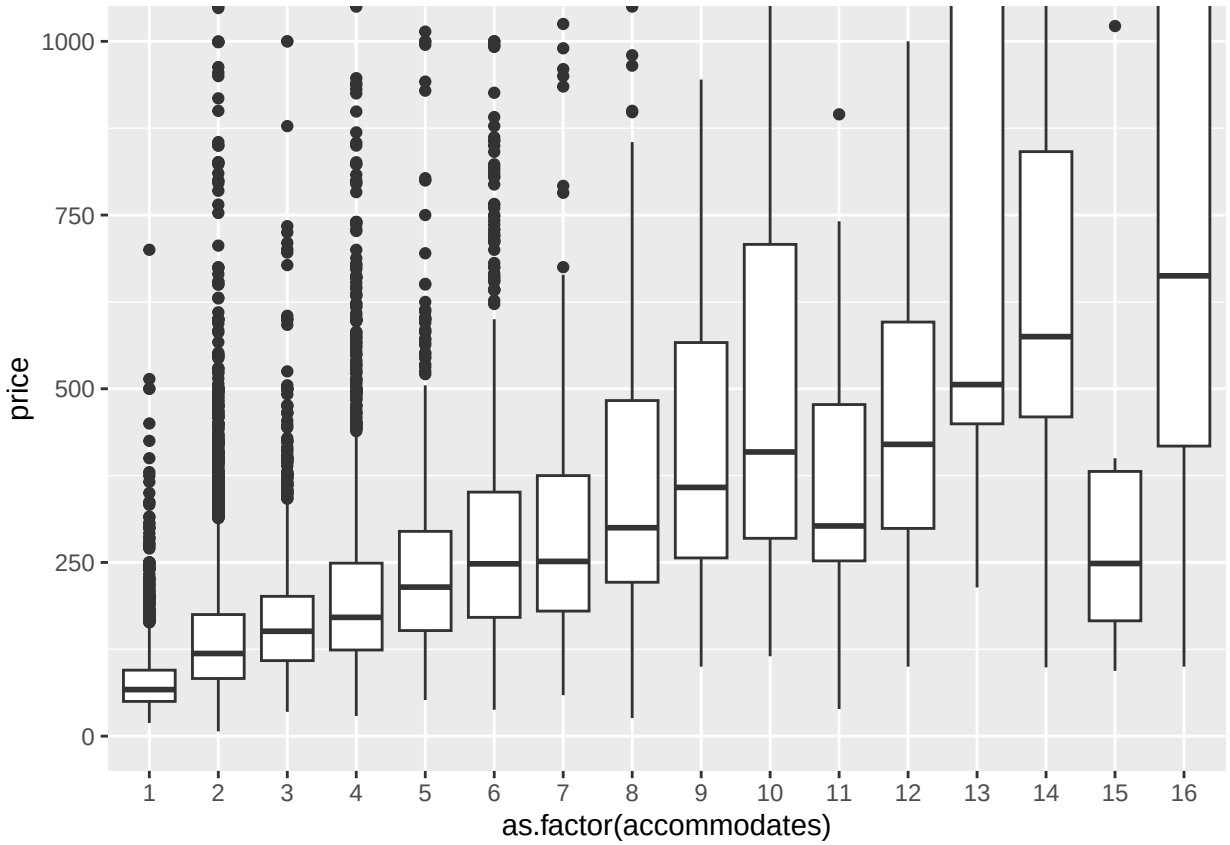
```
box.plot(bathrooms, data = bnb_data, yaxis = 1000)
```



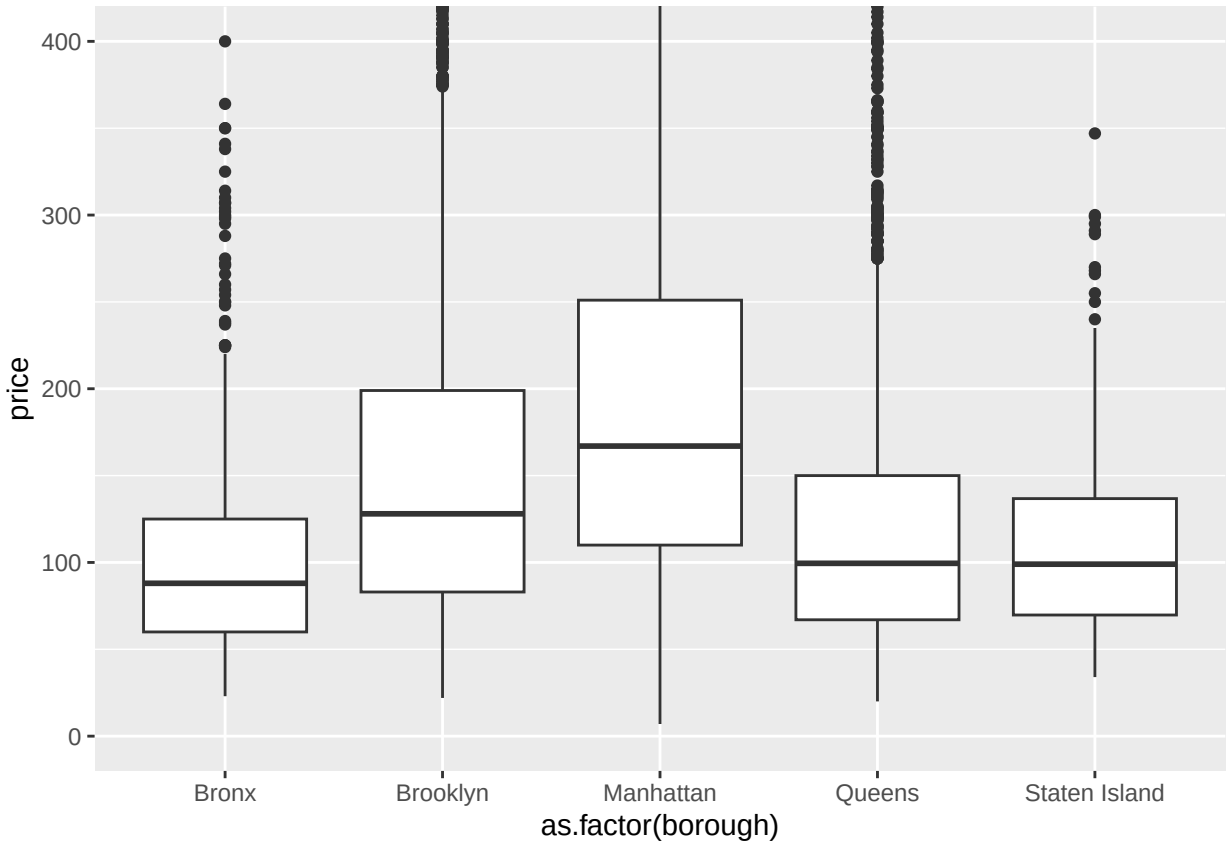
```
box.plot(bedrooms, data = bnb_data, yaxis = 1000)
```




```
box.plot(accommodates, data = bnb_data, yaxis = 1000)
```




```
box.plot(borough, data = bnb_data, yaxi = 400)
```




```r
# we will round half bathrooms (no shower) down as is common practice
bnb_data$bathrooms <- as.numeric(as.character(bnb_data$bathrooms))
bnb_data$bedrooms <- as.numeric(as.character(bnb_data$bedrooms))

# categorize bathrooms (vast majority '<2' == 1)
bnb_data$bathrooms <- cut(
  bnb_data$bathrooms,
  breaks = c(-Inf, 2, 2.6, 3.6, Inf),
  labels = c("<2", "2", "3", "4+"),      # few observations 4+
  right = FALSE                           #i.e. (-inf, 2) ; [2, 2.6) ; [...) ; etc.
)

# categorize bedrooms (some had 0 bedrooms listed, often for entire homes)
bnb_data$bedrooms <- cut(
  bnb_data$bedrooms,
  breaks = c(-Inf, 2, 3, 4, Inf),
  labels = c("<2", "2", "3", "4+"),
  right = FALSE                         #i.e. (-inf, 1) ; etc.
)

# categorize accomodates similarly to bed/bathrooms
bnb_data$accommodates <- cut(
  bnb_data$accommodates,
  breaks = c(-Inf, 2, 5, 7, 9, Inf),
  labels = c("1", "2-4", "5-6", "7-8", "9+"),
  right = FALSE
```

```
)

# merge borough
levels(bnb_data$borough) <- ifelse(
  levels(bnb_data$borough) %in% c("Bronx", "Staten Island"),
  "Bronx/S.I.",
  levels(bnb_data$borough)
)
```

'Amenities' is a column which contains a list of strings - ["Refrigerator", "Dishes and silverware", "etc."]. Of course, an observation which contains several strings cannot be passed into regression, so we aimed to extract a meaningful subset of amenities that many hosts advertised in their listings. For this, the amenities function (having coded 'create_descriptors', I find that there was surely an easier way, but the function works). A summary of its steps follows: decide on a 'cutoff', how many instances of an amenity must there be before a related column is created. Format values by removing brackets and quotes, and separate the list by commas. Create new columns for each item that appeared in 'amenities' and standardise their names, then find columns which do not have 'cutoff' true values and remove them. Finally, create 'amenity_count', which, for each observation, is the number of new columns with TRUE, i.e. how many relevant amenities that listings has. The most common amenities were 'smoke_alarm' (95%) & 'wifi' (90%), an example of a less common amenity is 'free_street_parking' (30%).

We generate 'bnb_data2' here, by creating 10 new boolean amenity columns.

```
# Observations are lists of strings with 'amenities'. Code will split these into
# Boolean variables, then they will be inspected for usefulness.

# function to create new amenity columns
create_amenities <- function(cutoff = 8000, data = bnb_data) {
  before <- ncol(data)

  # format amenities as plain cs list
  data <- data %>%
    mutate(
      amenities_list = str_replace_all(amenities, "\\[|\\]", ""),    # rm any square brackets
      amenities_list = str_replace_all(amenities_list, '\\"', ""),  # rm quotes
      amenities_list = strsplit(amenities_list, ",\\s*") # separate based on commas
    )

# code from
#https://stackoverflow.com/questions/70383248/how-to-seperate-a-list-column-to-multiple-logical-columns
#url ends '-in-r'
  data <- data %>%
    unnest(amenities_list) %>%
    pivot_wider(names_from = amenities_list,
                values_from = amenities_list, # create columns for each amenity
                values_fn = list(amenities_list = function(x) length(x) > 0),
                values_fill = FALSE)
  # if the amenity is not present for an observation, it's value becomes FALSE

  colnames(data) <- gsub(" ", "_", tolower(colnames(data)))
  # standardise names: Washing machine -> washing_machine
  bad_amenities <- c()

  # can adjust cutoff however you want, for OLS I don't want too many new predictors
```

```
  for (col in colnames(data)) {
  if (is.logical(data[[col]])) {
    if (sum(data[[col]] == TRUE, na.rm = TRUE) < cutoff) {
      bad_amenities <- c(bad_amenities, col)
      }
    }
  }

  data <- data[, !(colnames(data) %in% bad_amenities)]

  colnames(data) <- make.unique(colnames(data))
  data <- data[, !str_detect(colnames(data), "\\.1")] # remove duplicate columns (insignificant)

  data <- data %>%               # check number of TRUE across new columns
    mutate(
      amenity_count = rowSums(
        across(((before + 1):ncol(data)))
        )
      )

  return(data)
}

# amenities included in over 11500 listings, creates 10 new predictors for OLS
bnb_data2 <- create_amenities(11500)
```

Another column with string values is 'description'. By determining commonly used words, we generated another set of TRUE/FALSE variables for our data. 'create_descriptors' is similar to 'create_amenities' in its effect. The key differences are: manually setting the number of new columns, deciding a minimum length for a word to be considered (set to 5 so that there isn't a column for... 'that'). Furthermore, we remove all characters that aren't letters. Then, create a row for each word in each observation, group them by word and create columns for the top 'count' most common. An important point is that we did not parse the 'name' column, since, on inspection, it largely includes information contained in other predictors.

The new 'minutes_in_description' column was created, offering a useful way to determine whether the property is close to anything of note, such as a subway or tourist attraction.

```
# Observations are just strings, we can split these more easily

# function to create 'count' new common word columns
create_descriptors <- function(count = 10, length_bound = 5, data = bnb_data,
                               cols = c("description")) {

  for (col in cols) {
  all_words <- data %>%   # !!sym(.)means the same as [[.]], but in dplyr
    select(!!sym(col)) %>%
    mutate(
      strings = tolower(!!sym(col)),
      strings = str_replace_all(strings, "[^a-z\\s]", "")  # rm any non a-z (s is spaces) characters
    ) %>%
    separate_rows(strings, sep = "\\s+") %>% # create rows e.g. 'flat' or 'welcome'
    filter(nchar(strings) >= length_bound) %>% # take out words like "and"
    group_by(strings) %>%
    summarise(freq = n(), .groups = "drop") %>% # each word now has an associated total count
```

```
    arrange(desc(freq)) %>%
    slice_head(n = count) # and choose the 'count' most frequently appearing

    top_words <- all_words$strings

    for (word in top_words) {
      col_name <- paste0(word, "_in_", col)
      data[[col_name]] <- str_detect(
        tolower(data[[col]]),
        paste0("\\b", word, "\\b") # has to be the full word
      )
    }
  }

  return(data)
}

bnb_data2 <- create_descriptors(data = bnb_data2)
```

#LM Setup / refining bnb_data2 Here, we remove the predictors which were flagged by analysis of 'cor.checker' and 'full_correlations'. These are mostly predictors related to listing count, minimum/maximum nights and review scores.

```
bnb_data2 <- bnb_data2 %>%
  # too similar to other predictors
  select(-c(latitude, longitude,    # not enough info for simple, used in ridge
            minimum_minimum_nights, # minimum_minimum is the lowest that minimum_nights has ever been
            maximum_maximum_nights, maximum_minimum_nights,
            minimum_maximum_nights, minimum_nights_avg_ntm,
            maximum_nights_avg_ntm, availability_60,
            availability_eoy, review_scores_accuracy,
            review_scores_cleanliness, review_scores_communication, # these were highly correlated
            review_scores_value, calculated_host_listings_count_entire_homes,
            calculated_host_listings_count_private_rooms,
            calculated_host_listings_count_shared_rooms))
```

'one_listing' is a crucial function for our data processing. We decided to subsample the data by host, such that our models consider only one random listing from each host. This was done in an effort to promote independence between observations. Of course, no two observations in this dataset are completely independent, they are all operating in the same Airbnb marketplace in the same city. Even so, we argue that listings from the same host will have notably similar characteristics, so models could overfit to a certain host's style. Indeed, Blueground has some 1194 listings in the base dataset, with the majority being, specifically, entire apartments in Manhattan. There are multiple hosts with over 100 listings, which gives further reason for our decision to slice the data by host.

```
# use this once you have your full final data
one_listing <- function(data) {
  data <- data %>%
   group_by(host_id) %>%
   slice_sample(n = 1)# random sample of one observation per host_id
}
```

#Missing Values (data2 for lm) So, finally, we opt for complete cases (which now only lowers the total by 500 observations to 15000) and use 'one_listing'.

```r
bnb_data2 <- na.omit(bnb_data2) # complete cases, 15000 total observations

bnb_data2 <- one_listing(bnb_data2) # sample one listing for each host (~9000 total observations)

summary(bnb_data2)
```

```
##      name           description          host_id           host_name
##  Length:8785        Length:8785        Min.   :     1678   Length:8785
##  Class :character   Class :character   1st Qu.: 15448039   Class :character
##  Mode  :character   Mode  :character   Median : 91156145   Mode  :character
##                                        Mean   :180242874
##                                        3rd Qu.:344545165
##                                        Max.   :676667930
##
##    host_since    host_total_listings_count             neighborhood
##  Min.   :14102   Min.   :   1.00           Bedford-Stuyvesant: 705
##  1st Qu.:16208   1st Qu.:   1.00           Harlem            : 485
##  Median :17036   Median :   2.00           Williamsburg      : 440
##  Mean   :17272   Mean   :   9.38           Crown Heights     : 325
##  3rd Qu.:18376   3rd Qu.:   4.00           Bushwick          : 323
##  Max.   :20121   Max.   :9055.00           Midtown           : 223
##                                            (Other)           :6284
##        borough                          property_type            room_type
##  Bronx/S.I.: 615   Entire rental unit         :3289   Entire home/apt:5132
##  Brooklyn  :3786   Private room in rental unit:1963   Hotel room     :  12
##  Manhattan :2726   Private room in home       : 800   Private room   :3595
##  Queens    :1658   Entire home                : 643   Shared room    :  46
##                    Entire condo               : 438
##                    Private room in townhouse  : 245
##                    (Other)                    :1407
##  accommodates bathrooms bedrooms       beds         amenities
##  1  :1171     <2:7738   <2:6210   Min.   : 0.000   Length:8785
##  2-4:6270     2 : 907   2 :1728   1st Qu.: 1.000   Class :character
##  5-6: 912     3 : 100   3 : 644   Median : 1.000   Mode  :character
##  7-8: 289     4+:  40   4+: 203   Mean   : 1.765
##  9+ : 143                         3rd Qu.: 2.000
##                                   Max.   :42.000
##
##      price       minimum_nights   maximum_nights   availability_30
##  Min.   :   19   Min.   :  1.00   Min.   :   1.0   Min.   : 0.0
##  1st Qu.:   91   1st Qu.: 30.00   1st Qu.:  60.0   1st Qu.: 1.0
##  Median :  139   Median : 30.00   Median : 365.0   Median :20.0
##  Mean   :  188   Mean   : 25.44   Mean   : 433.6   Mean   :16.5
##  3rd Qu.:  212   3rd Qu.: 30.00   3rd Qu.:1125.0   3rd Qu.:29.0
##  Max.   :10271   Max.   :719.00   Max.   :3000.0   Max.   :30.0
##
##  availability_90 availability_365  review_count     review_count_ly
##  Min.   : 0.00   Min.   :  0.0    Min.   :   1.00   Min.   :   0.000
##  1st Qu.:26.00   1st Qu.:113.0    1st Qu.:   6.00   1st Qu.:   0.000
##  Median :62.00   Median :214.0    Median :  26.00   Median :   1.000
##  Mean   :55.77   Mean   :217.5    Mean   :  58.22   Mean   :   9.048
##  3rd Qu.:89.00   3rd Qu.:335.0    3rd Qu.:  74.00   3rd Qu.:   5.000
##  Max.   :90.00   Max.   :365.0    Max.   :2749.00   Max.   :1784.000
```

```
##
##  est_occupancy_ly est_revenue_ly    first_review    last_review
##  Min.   :  0.0    Min.   :     0   Min.   :14389   Min.   :15492
##  1st Qu.:  0.0    1st Qu.:     0   1st Qu.:17776   1st Qu.:19615
##  Median : 60.0    Median :  8400   Median :19082   Median :19982
##  Mean   :102.9    Mean   : 17878   Mean   :18606   Mean   :19766
##  3rd Qu.:240.0    3rd Qu.: 26180   3rd Qu.:19510   3rd Qu.:20096
##  Max.   :255.0    Max.   :702014   Max.   :20147   Max.   :20149
##
##      stars       stars_checkin   stars_location  host_listings_count
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :   1.000
##  1st Qu.:4.720   1st Qu.:4.840   1st Qu.:4.650   1st Qu.:   1.000
##  Median :4.880   Median :4.940   Median :4.820   Median :   1.000
##  Mean   :4.783   Mean   :4.864   Mean   :4.749   Mean   :   2.453
##  3rd Qu.:5.000   3rd Qu.:5.000   3rd Qu.:4.970   3rd Qu.:   2.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :1194.000
##
##  reviews_per_month dishes_and_silverware    wifi          kitchen
##  Min.   :  0.010   Mode :logical         Mode :logical   Mode :logical
##  1st Qu.:  0.240   FALSE:1616            FALSE:1000      FALSE:855
##  Median :  0.700   TRUE :7169            TRUE :7785      TRUE :7930
##  Mean   :  1.264
##  3rd Qu.:  1.750
##  Max.   :117.980
##
##  hair_dryer      essentials         iron         hot_water
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:1964      FALSE:1303      FALSE:2122      FALSE:1305
##  TRUE :6821      TRUE :7482      TRUE :6663      TRUE :7480
##
##
##
##
##   hangers        carbon_monoxide_alarm smoke_alarm     amenity_count
##  Mode :logical   Mode :logical         Mode :logical   Min.   :0.000
##  FALSE:1645      FALSE:1145            FALSE:385       1st Qu.:6.000
##  TRUE :7140      TRUE :7640            TRUE :8400      Median :7.000
##                                                        Mean   :6.779
##                                                        3rd Qu.:8.000
##                                                        Max.   :8.000
##
##  apartment_in_description located_in_description bedroom_in_description
##  Mode :logical            Mode :logical          Mode :logical
##  FALSE:5175               FALSE:5838             FALSE:5987
##  TRUE :3610               TRUE :2947             TRUE :2798
##
##
##
##
##  private_in_description kitchen_in_description manhattan_in_description
##  Mode :logical          Mode :logical          Mode :logical
##  FALSE:6442             FALSE:6405             FALSE:6180
##  TRUE :2343             TRUE :2380             TRUE :2605
##
```

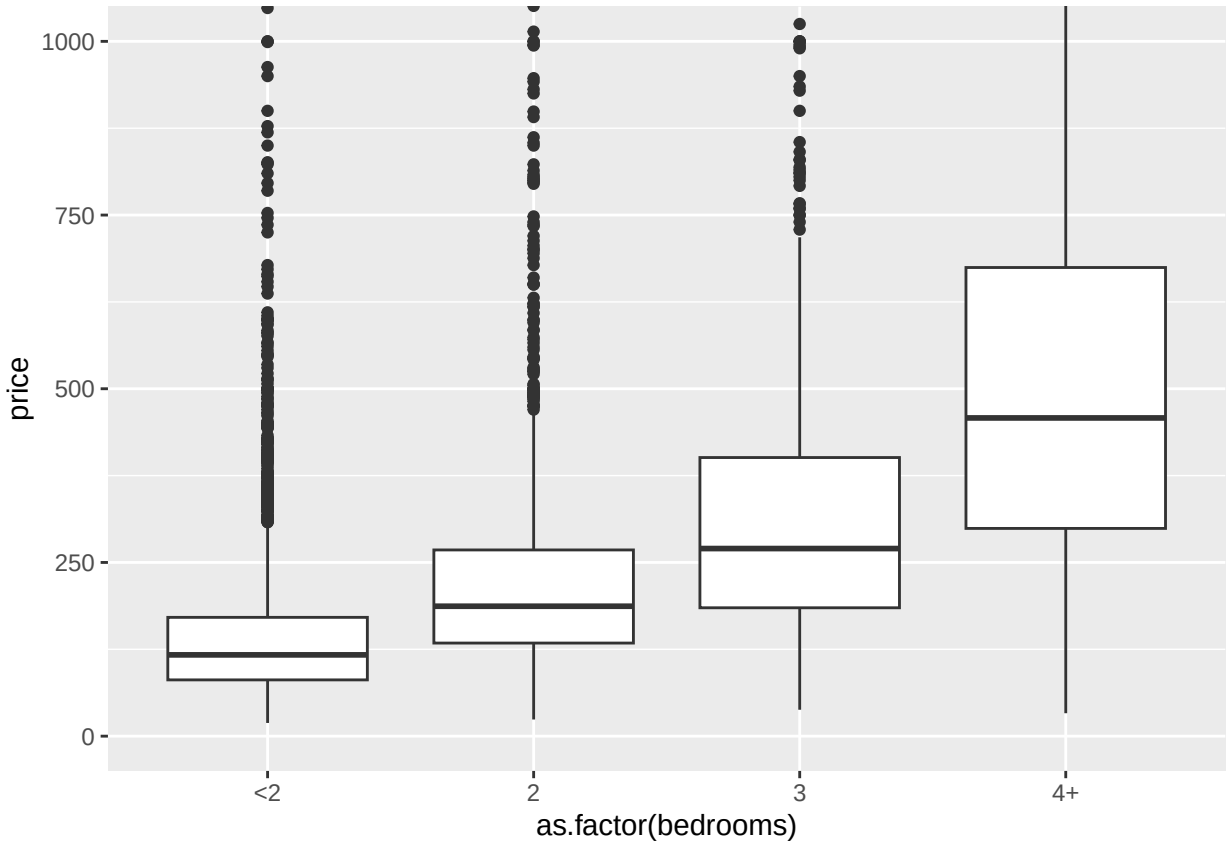
```
##
##
##
##  minutes_in_description restaurants_in_description subway_in_description
##  Mode :logical          Mode :logical               Mode :logical
##  FALSE:7129             FALSE:6299                  FALSE:6914
##  TRUE :1656             TRUE :2486                  TRUE :1871
##
##
##
##
##  space_in_description
##  Mode :logical
##  FALSE:6841
##  TRUE :1944
##
##
##
##
```

###EDA

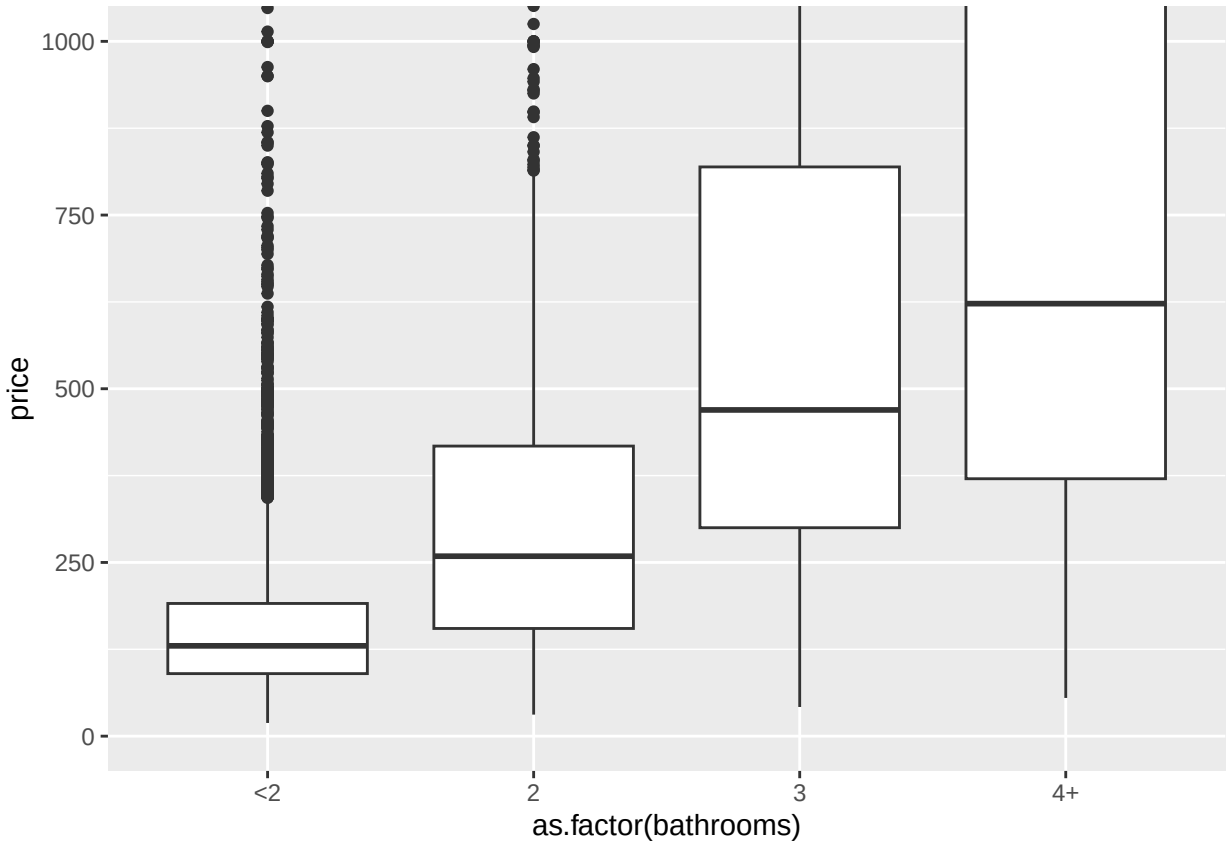
In this section, we include interesting plots from our EDA, looking at the basic effects of variables, to inform our simple models (OLS & Simple Tree), as well as lay the groundwork for further modelling.

First, we see the boxplots for the three predictors with new levels. Clearly, there is a strong effect of increasing bed/bathrooms, as well 'accommodates'. A listing with 2 bedrooms has an average price of $190 per night, while a three bedroom listing jumps to over $250. Average increases by a factor of 5, from 125 to 625, when comparing between a listing with <2 bathrooms, and one with 4+. 'Accommodates' and 'bathrooms' have similar, relatively linear effects, though they also have many outlying observations.

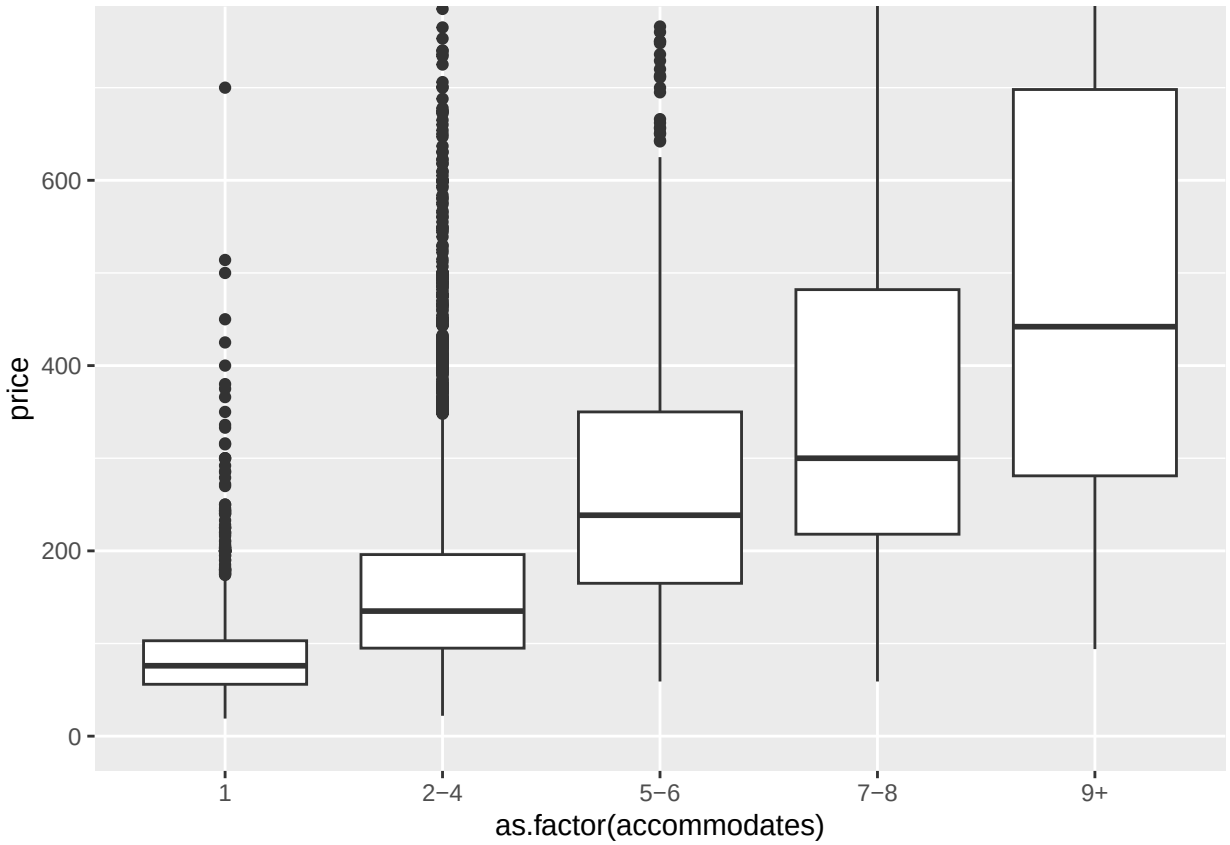
```
box.plot(bedrooms, yaxis = 1000)
```




```
box.plot(bathrooms, yaxis = 1000)
```



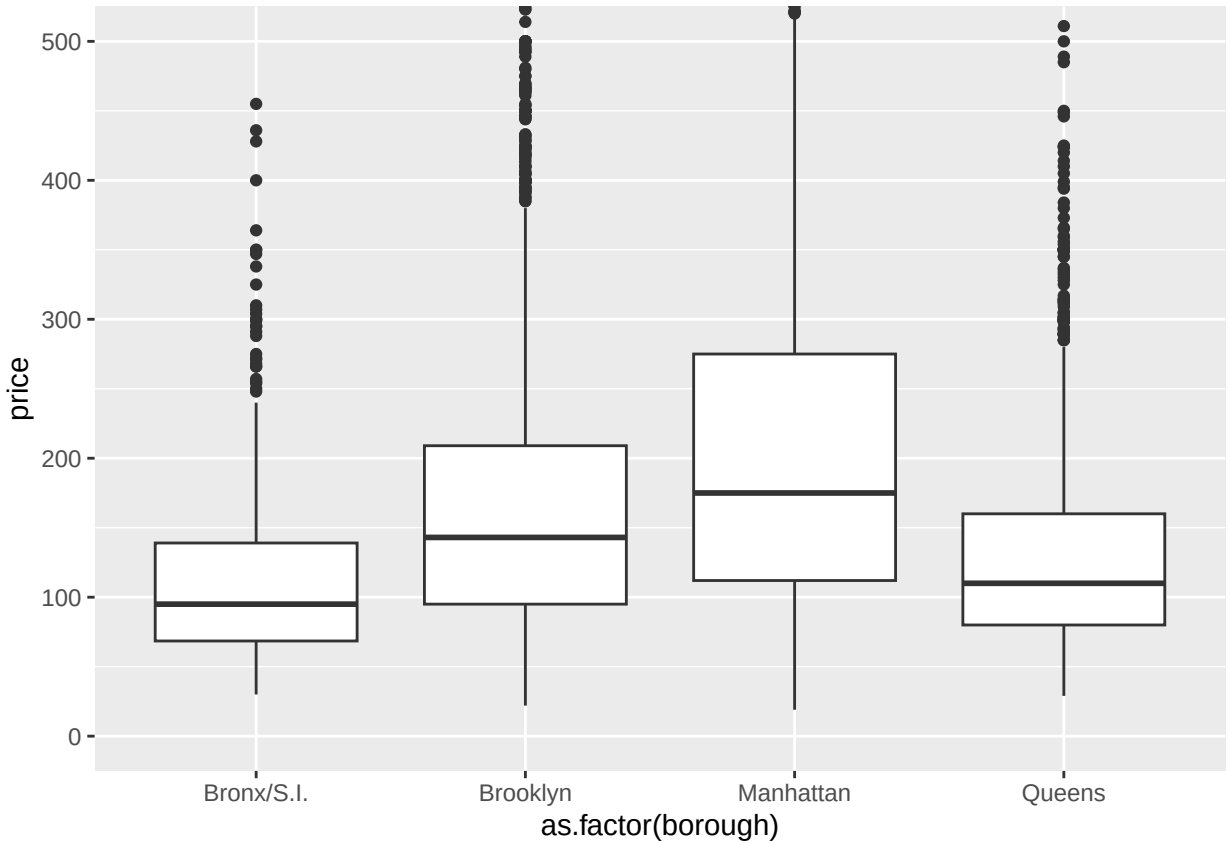

```
box.plot(accommodates, yaxis = 750)
```



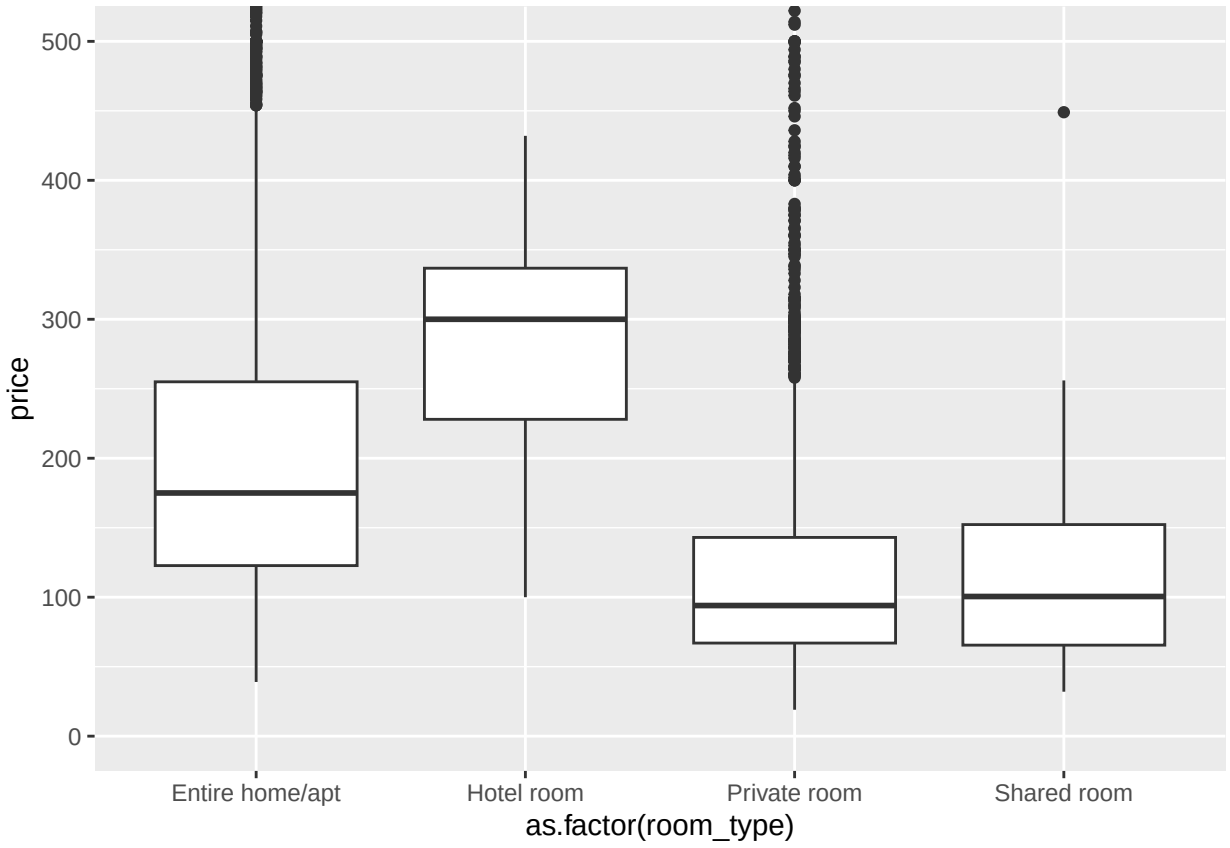
Two more plots that show clear relationships are that of 'borough' and 'room_type'. The average price of Manhattan listings is almost double that of those in the Bronx - $175 vs. $90 a night. Similarly to the 'room_type' plot, we clearly see that the levels which include many listings have low average prices. For instance 'Private room's are 40% of the listings in 'bnb_data2', and they have an average price of <$100 per night (a counter-example is shared rooms, which, with only 50 listings, have an average price of $100, but this is trivial).

With this in mind, we plot histograms to determine the distribution of price. We find that the data is heavily skewed. If we don't limit the x axis, it runs off to 10,000, despite almost all observations having price <500. Checking log(price), we find it is much more nicely distributed, in an approximate bell curve.

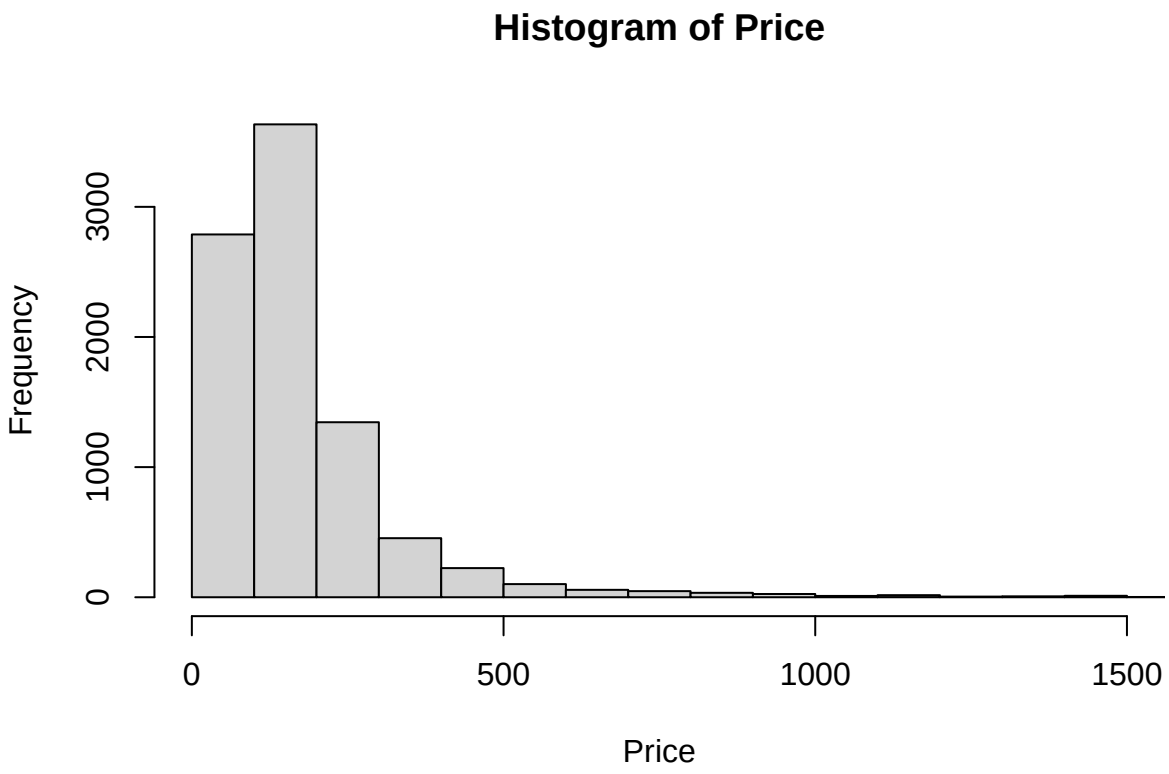
```
# clear relationship
box.plot(borough, yaxis = 500)
```



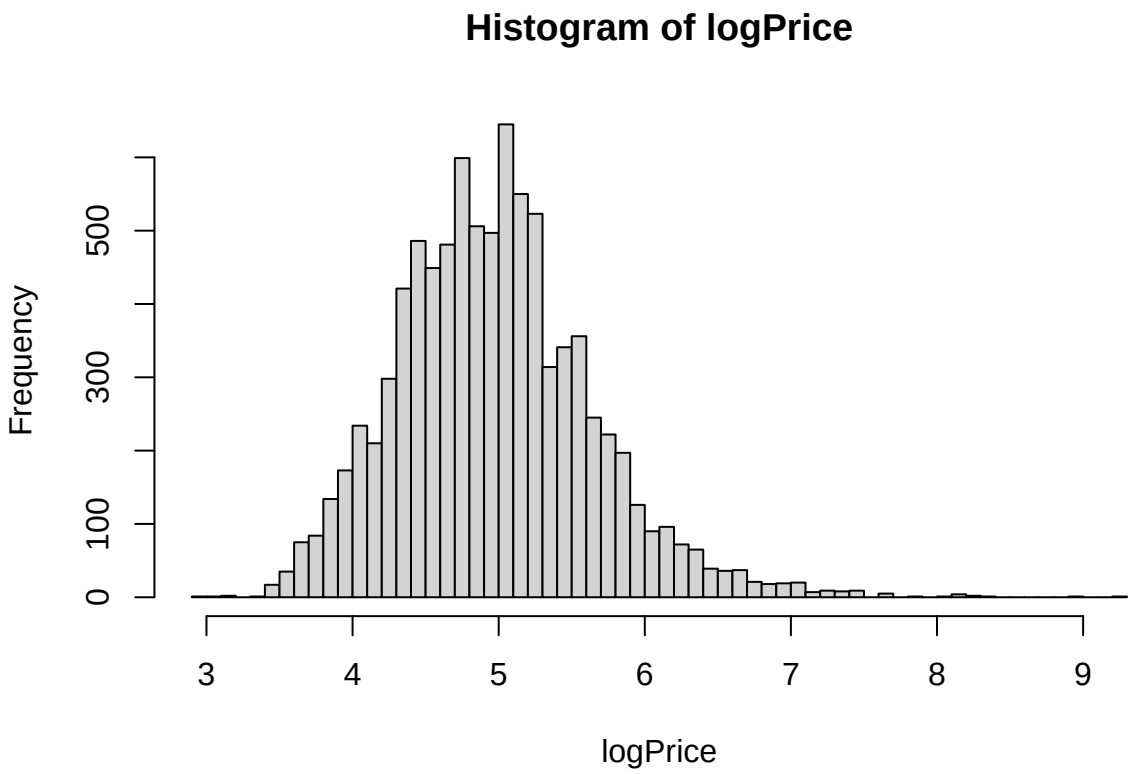

```
box.plot(room_type, yaxis = 500)
```




```
# histograms of price distribution
hist(bnb_data2$price,
     main = "Histogram of Price",
     xlab = "Price", breaks = 100, xlim = c(0,1500))
```



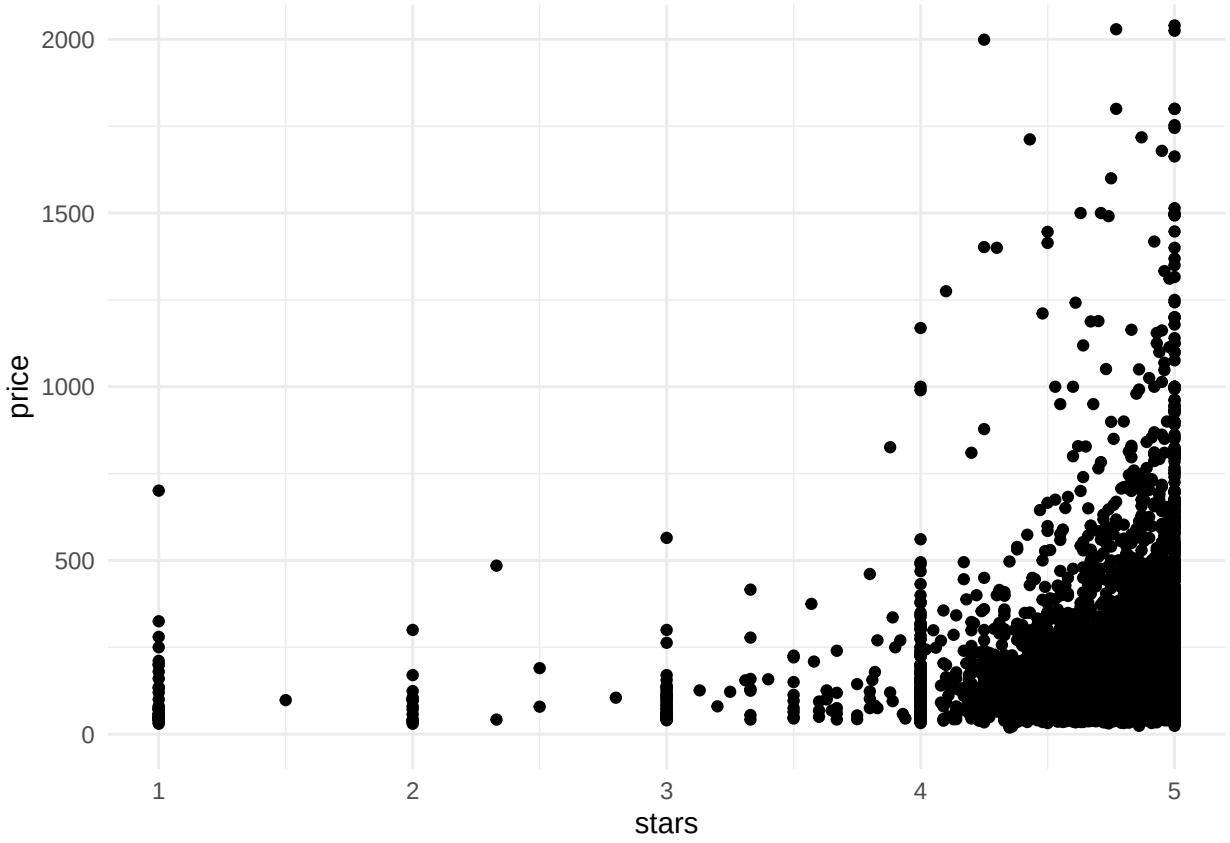

```
hist(log(bnb_data2$price),
     main = "Histogram of logPrice",
     xlab = "logPrice", breaks = 50)
```



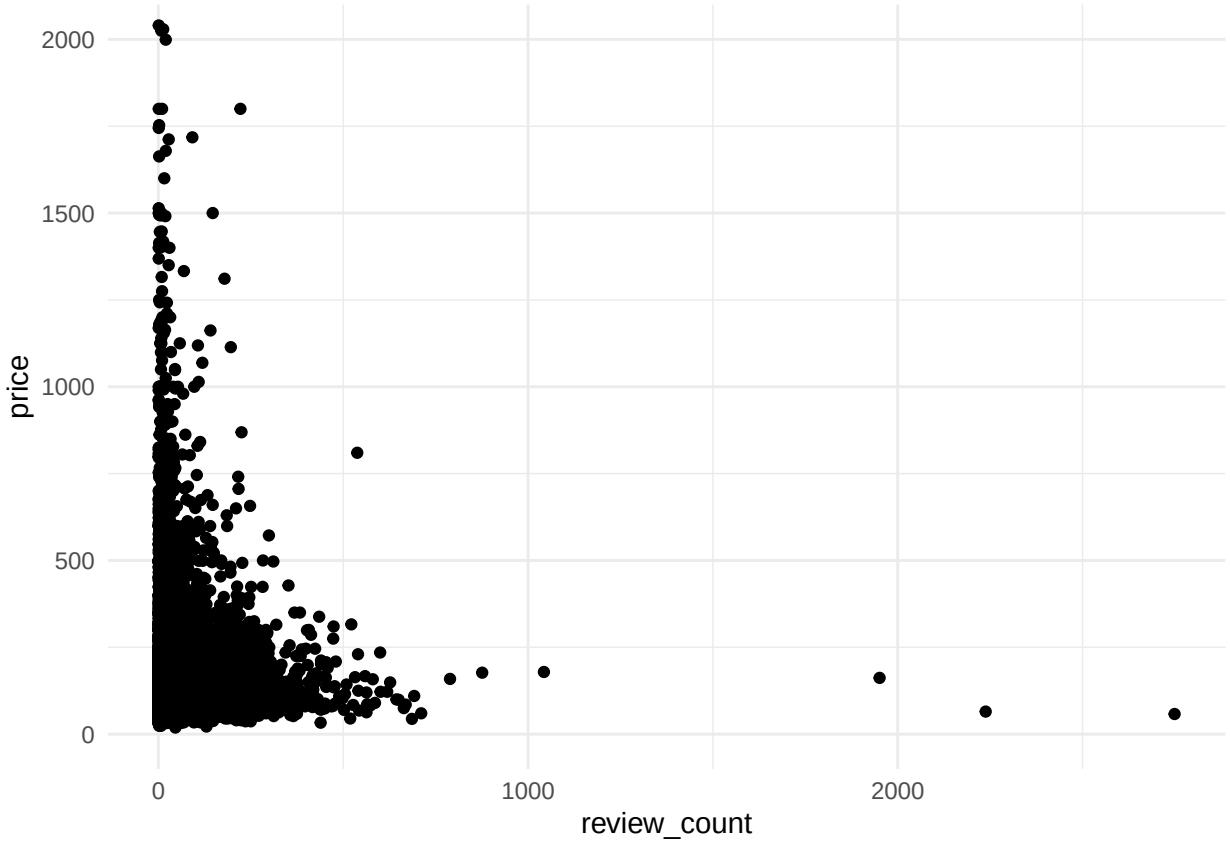
We found many other less clearly defined relationships. As an example, 'stars' - average rating /5 - shows strong positive correlation for the most expensive properties, but it is somewhat difficult to glean any further information from the plot, since so many values are >4.

An informative plot is that of 'est_revenue_ly'. Looking at its shape, one can notice that estimated revenue is linearly related to price, where the slope can vary. We will exercise caution in including 'est_revenue_ly' in modelling, as it is very likely calculated from price.

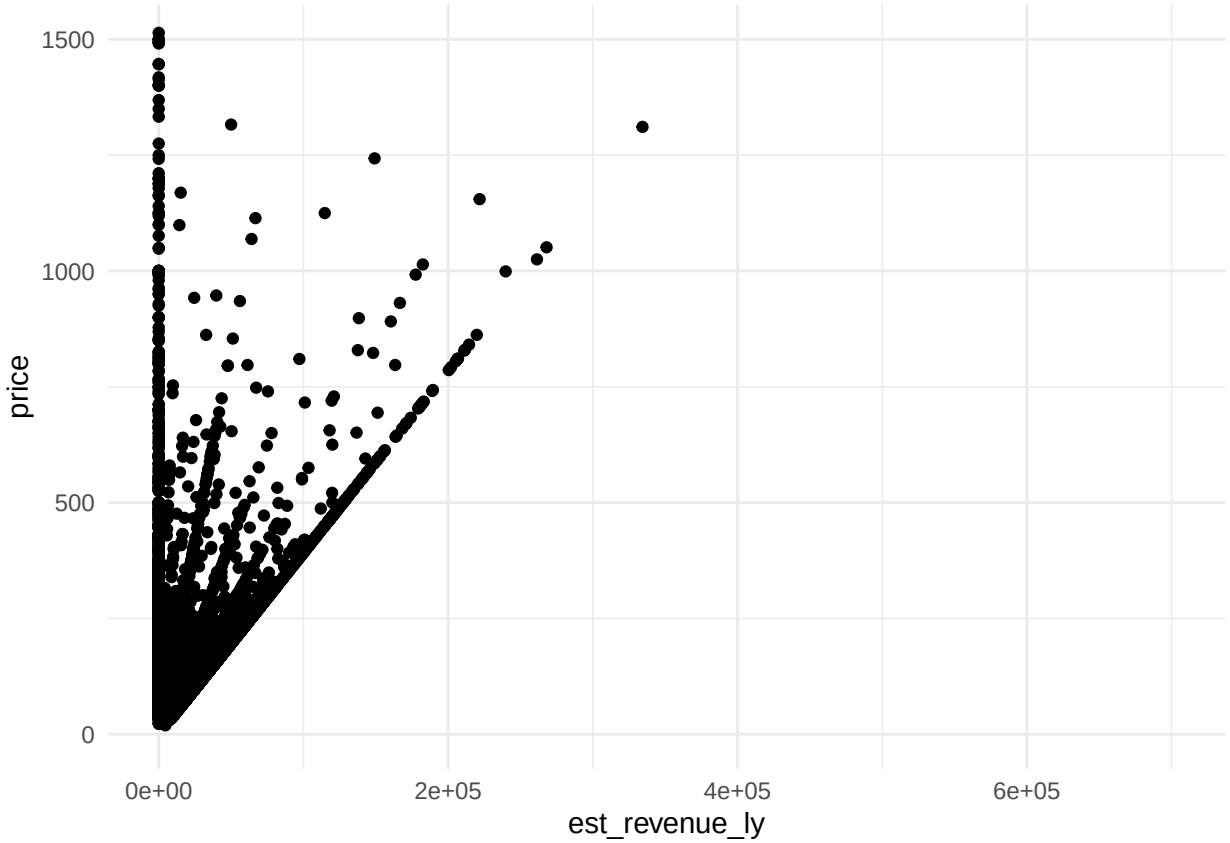
```
scatter.plot(stars, yaxis = 2000)
```




```
# some relationship
scatter.plot(review_count, yaxis = 2000)
```

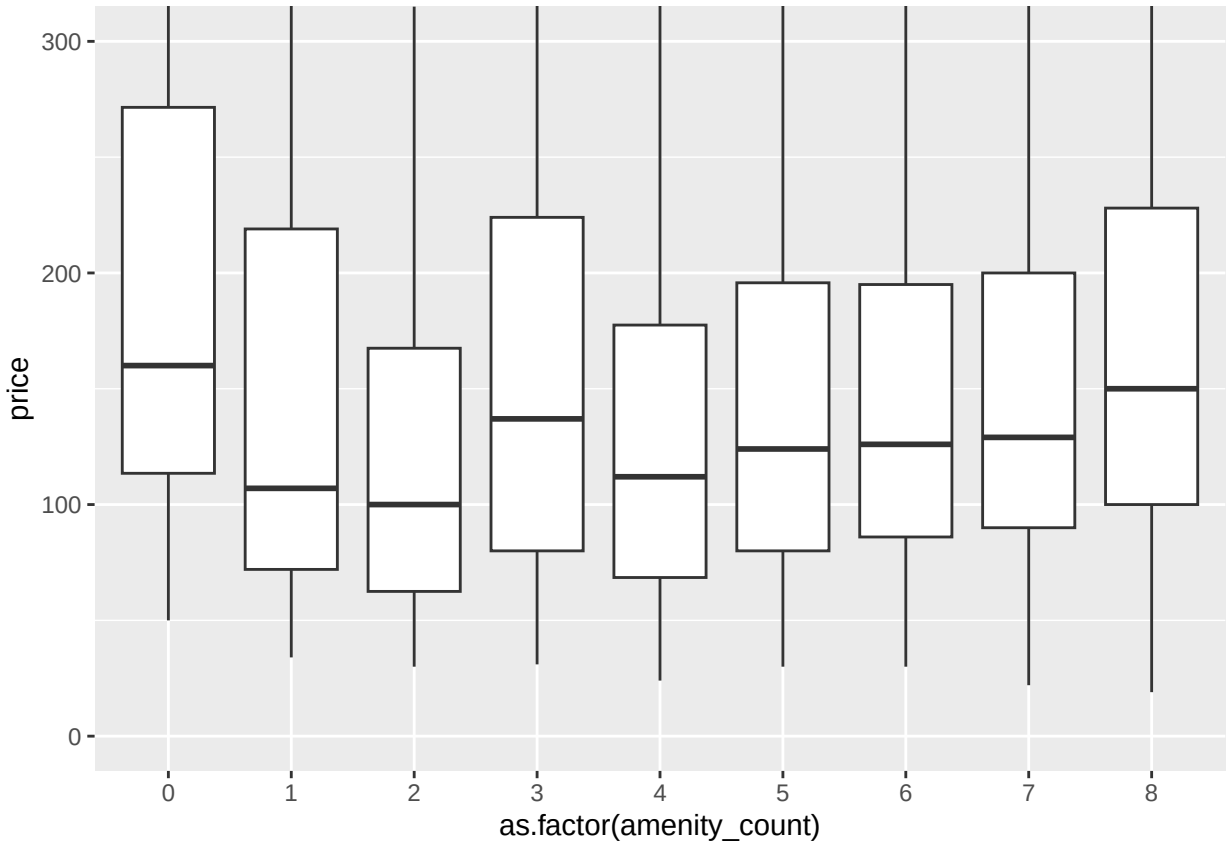



```
# cool plot
scatter.plot(est_revenue_ly)
```

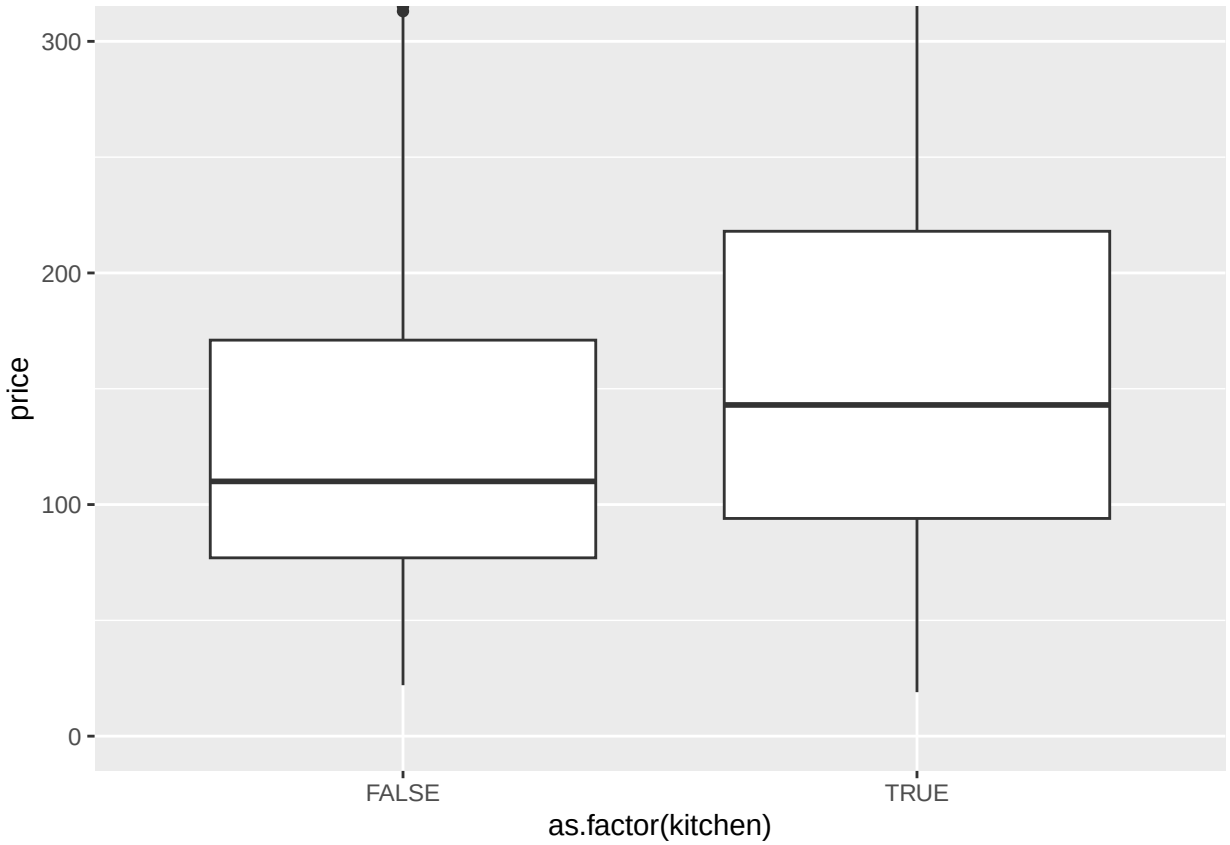


Next, a couple of examples of amenities/descriptors, we see the mention of kitchen as an amenity corresponds to a ~$30 jump in listing price, while listings with "minutes" in their description actually have a $20 lower average price than those without.

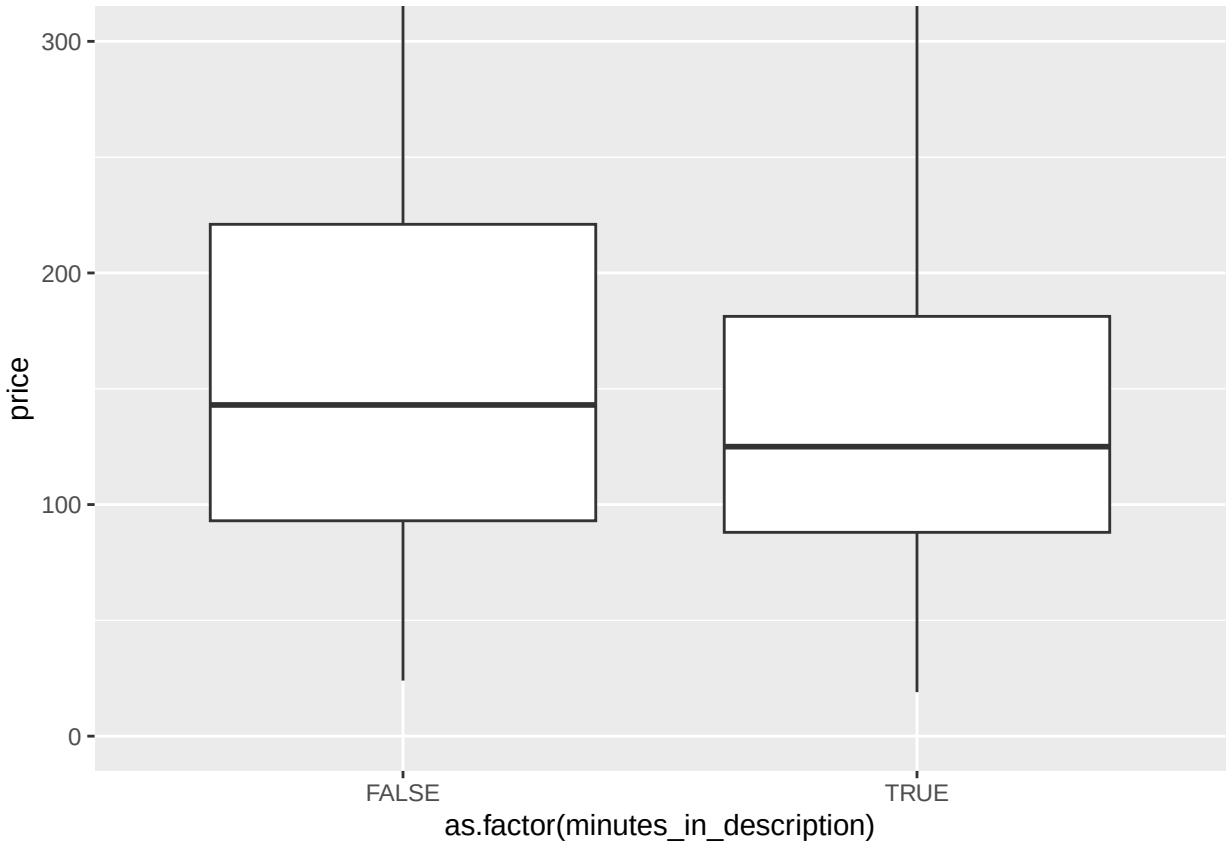
```
box.plot(amenity_count, yaxis = 300)
```




```
# example of amenities
box.plot(kitchen, yaxis = 300)
```

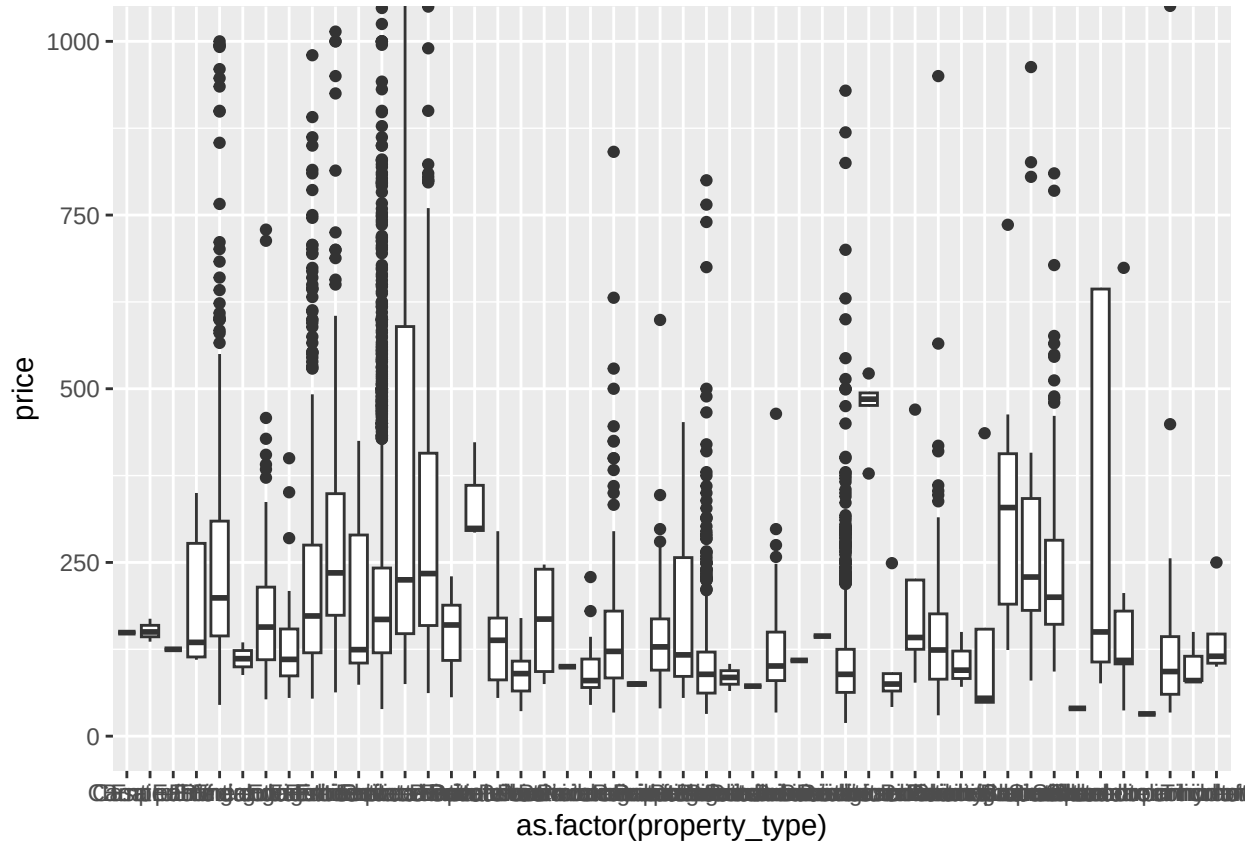



```
# examples of descriptors
box.plot(minutes_in_description, yaxis = 300)
```

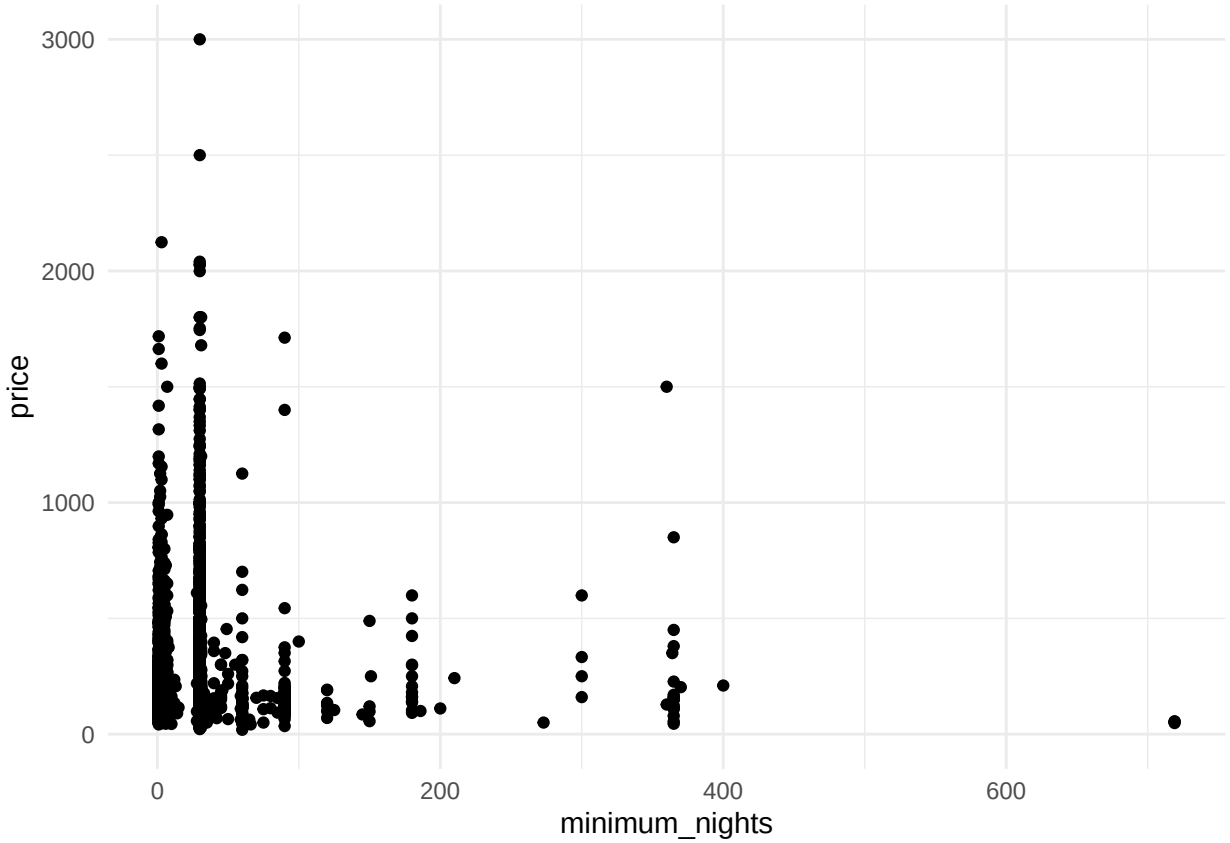


Much of the rest of our EDA looked like the plots generated in the next chunk - somewhat uninformative. Though, with the plots detailed above, and a few others, we have a good idea of what direction to head with this data.

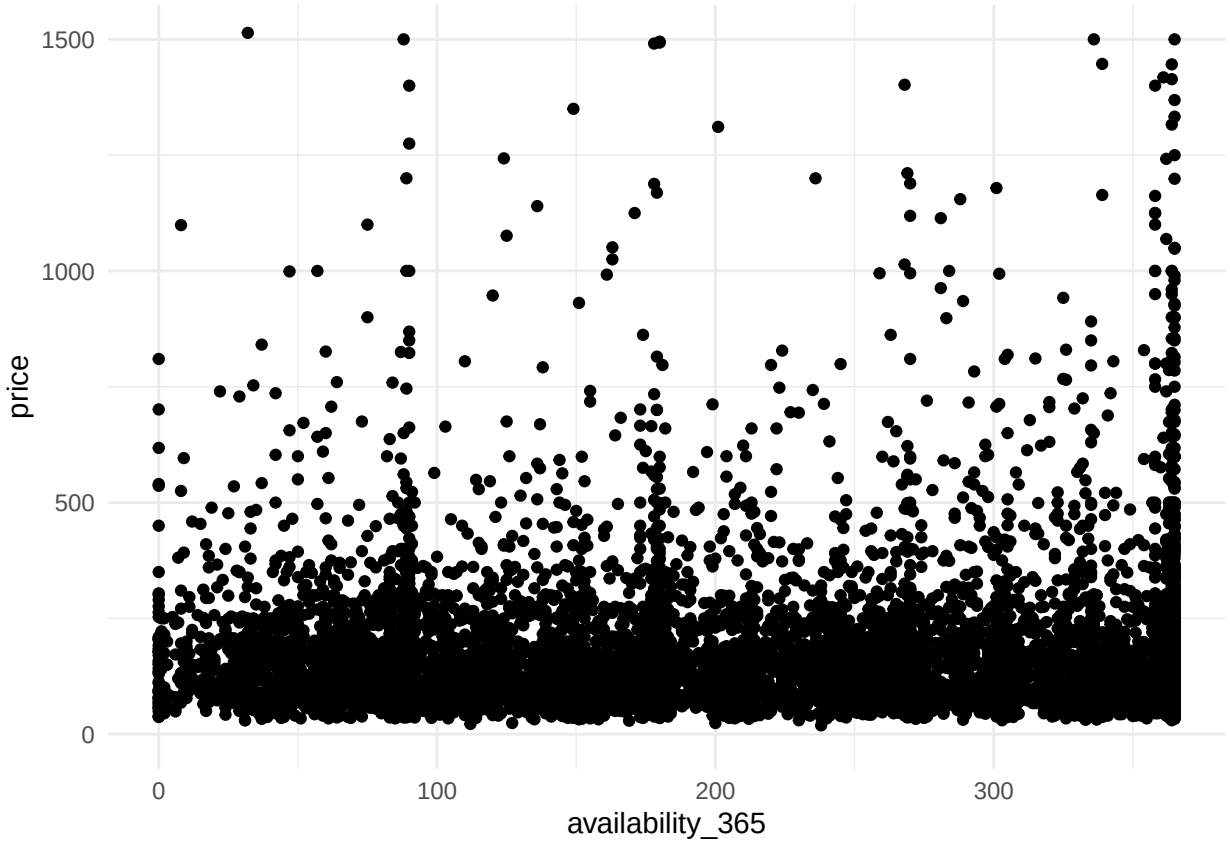
```
# huge number of factors
box.plot(property_type) # could definitely remove
```




```
# not much
scatter.plot(minimum_nights, yaxis = 3000)
```




```
scatter.plot(availability_365)
```



###Models

By exploring the data, we developed a solid understanding of the distribution of the outcome and the individual effects of each predictor. Now, we will apply this analysis to model fitting:

##Linear Regression (baseline)

Selected as an easily interpreted baseline, the LM developed here serves as a foundation for further modelling. k-NN was tested, but the added complexity yielded no great benefit.

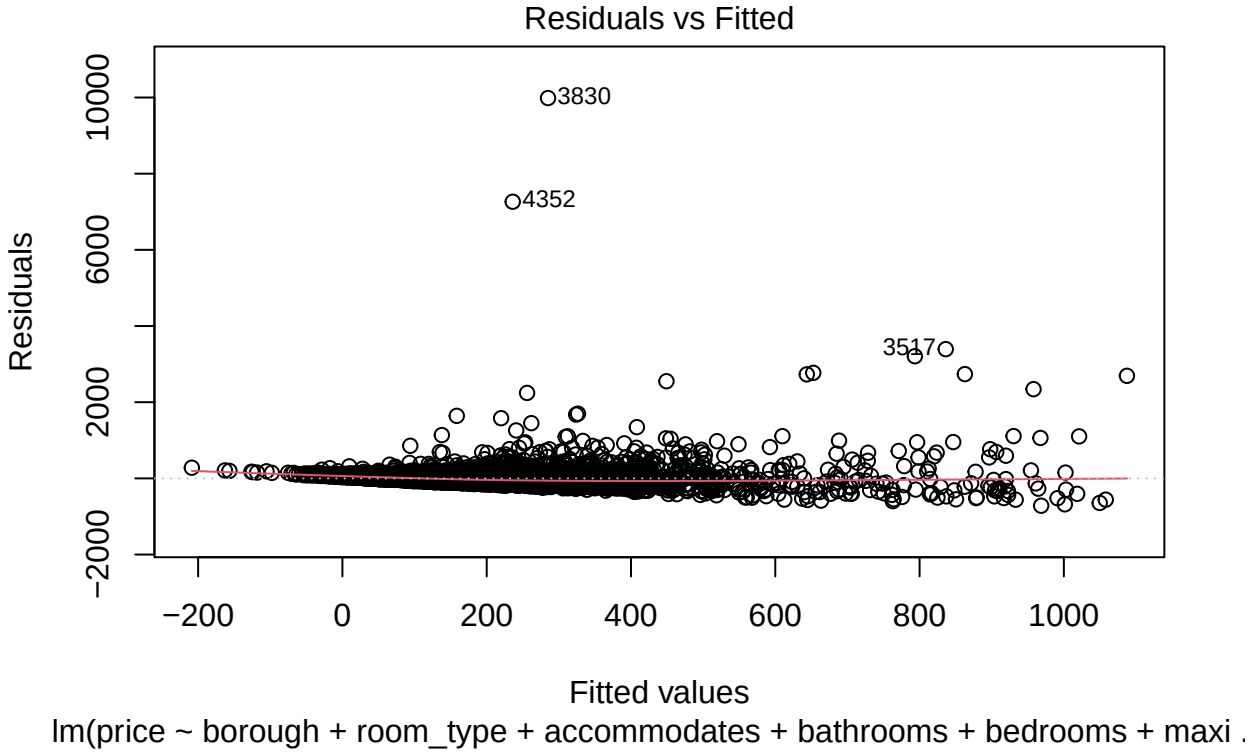
The starting point 'lm_base' includes 30 of the 52 columns in our data 'bnb_data2'. Several string type columns were not considered, such as 'name' and 'description', further, seeing as this is our most interpretable model, categorical variables with many levels (like 'neighborhood') were removed. Knowing the relationship between 'est_revenue_ly' and price, we also removed this predictor, as well as quickly culling some of the new columns from 'create_...'. Also of note: 'beds' was dropped in favour of 'bedrooms'.

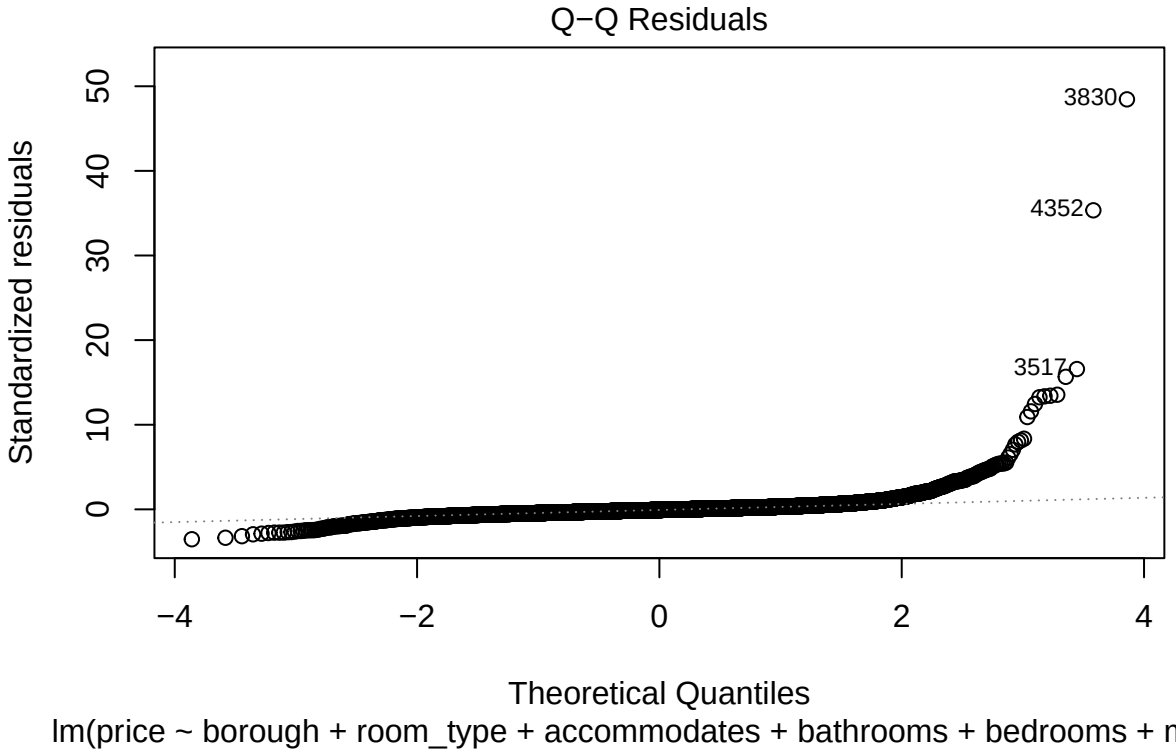
Viewing the Q-Q plot for the base LM is revealing, we saw that a log transform is necessary to model listings with the most positive residual (usually highest true values). Looking at the most extreme values, we found that a few of the very highest priced listings were incredibly badly predicted (see 3830, which is $10,000 a night, std. resid = 50). Even with the outcome logged, this issue persists, so we decided to remove the 20 observations with the highest prices (>=1800). An intuitive justification for this fix is that, if one is paying for, or listing, a property on Airbnb for over $1800 a night, there should be a clear reason why the price is so high. Our models aim to assist hosts in pricing their listings, or customers in aligning their price range with characteristics of a property they might stay in. In the case of, say, observation 3517: a '4 bedroom Luxury Penthouse ...' which has '... 24/7 security and VIP concierge ...' for $4000 a night, both the host, and client, are almost operating in a different market from the thousands of listings priced below $1800 a night. Thus removing these listings improves both the reliability and the general relevance of modelling.

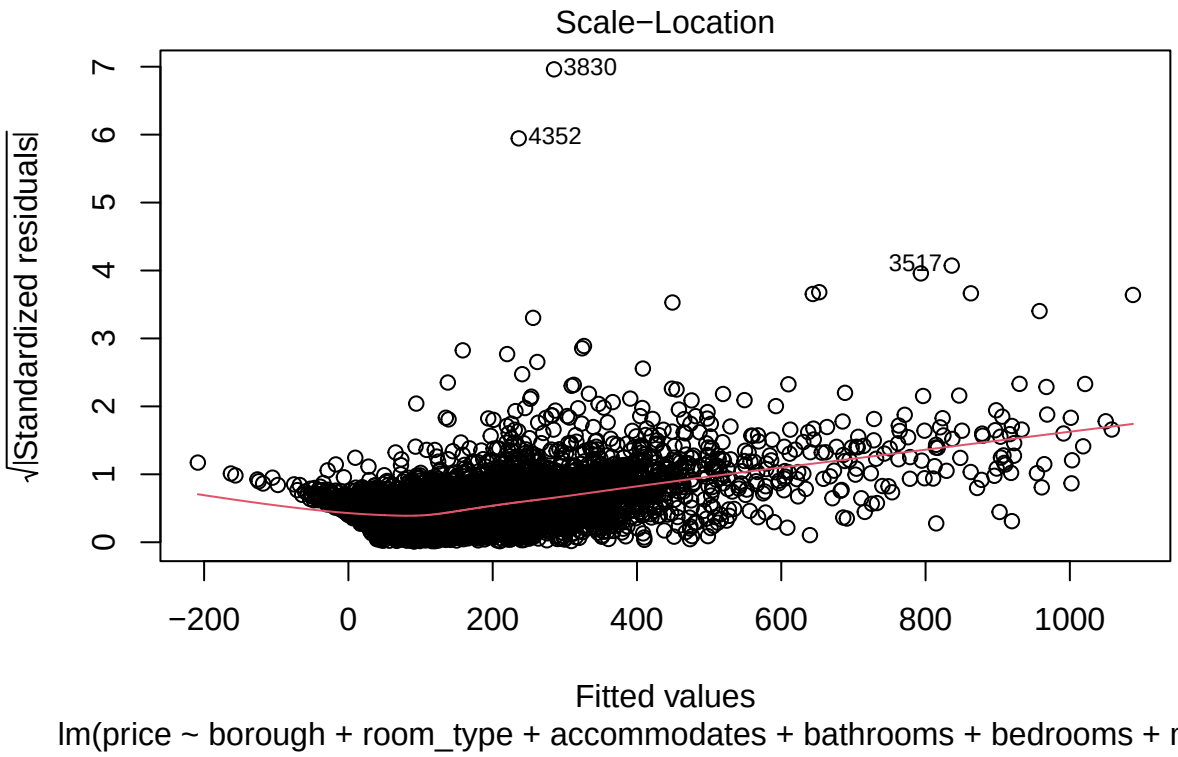
Improving upon our base case, then, we filtered for price, logged the outcome, and performed backwards elimination - iteratively removing the least significant predictor until all were significant at at least the 10%

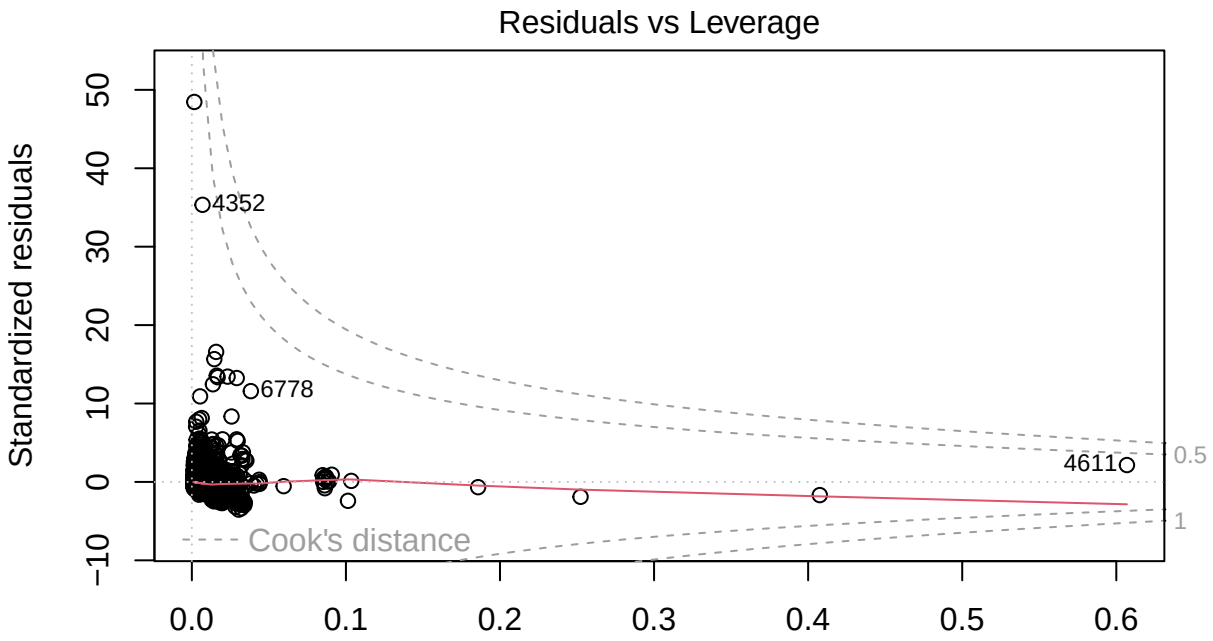
level. Already, the residual plot & Q-Q demonstrated that residuals are far more normally, independently distributed.

```
#colnames(bnb_data2)

# base model I started with (doesn't include all predictors)
lm_base <- lm(price ~ borough + room_type + accommodates + bathrooms + bedrooms + maximum_nights +
                      availability_30 + availability_90  + review_count_ly + stars +
                      stars_checkin + stars_location + host_listings_count + reviews_per_month +
                      dishes_and_silverware + wifi + kitchen + hair_dryer + essentials +
                      iron + hot_water + hangers + carbon_monoxide_alarm + smoke_alarm +
                      minutes_in_description + restaurants_in_description + subway_in_description +
                      amenity_count + host_since + first_review + last_review,
                    data = bnb_data2)

#summary(lm_base)
plot(lm_base)
```




Q−Q Residuals
Standardized residuals
0
10
20
30
40
50
3830
4352
3517
−4
−2
0
2
4
Theoretical Quantiles
lm(price ~ borough + room_type + accommodates + bathrooms + bedrooms + maxi .


Scale−Location
√|Standardized residuals|
3830
4352
3517
0
1
2
3
4
5
6
7
−200
0
200
400
600
800
1000
Fitted values
lm(price ~ borough + room_type + accommodates + bathrooms + bedrooms + maxi .




```
# intermediate (log(outcome) -> backwards elimination)
# given residual plots, we remove the highest price listings, as these seem hard to estimate
bnb_data2 <- bnb_data2 %>%
  filter(price < 1800)

lm_inter <- lm(log(price) ~ borough + room_type + accommodates + bathrooms +
                        bedrooms + maximum_nights + availability_30 +
                        stars + stars_checkin + stars_location +
                        reviews_per_month + wifi + hair_dryer + hot_water +
                        minutes_in_description +
                        host_since + first_review + last_review,
                      data = bnb_data2)

summary(lm_inter)
```

```
##
## Call:
## lm(formula = log(price) ~ borough + room_type + accommodates +
##     bathrooms + bedrooms + maximum_nights + availability_30 +
##     stars + stars_checkin + stars_location + reviews_per_month +
##     wifi + hair_dryer + hot_water + minutes_in_description +
##     host_since + first_review + last_review, data = bnb_data2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -1.91075 -0.29672 -0.01517  0.26957  2.59633
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   3.543e+00  2.166e-01  16.355  < 2e-16 ***
## boroughBrooklyn               2.578e-01  1.990e-02  12.957  < 2e-16 ***
## boroughManhattan              4.880e-01  2.068e-02  23.600  < 2e-16 ***
## boroughQueens                 5.900e-02  2.143e-02   2.753 0.005916 **
## room_typeHotel room           2.732e-01  1.308e-01   2.089 0.036730 *
## room_typePrivate room        -3.383e-01  1.166e-02 -29.010  < 2e-16 ***
## room_typeShared room         -4.343e-01  6.777e-02  -6.409 1.54e-10 ***
## accommodates2-4               3.444e-01  1.554e-02  22.154  < 2e-16 ***
## accommodates5-6               5.413e-01  2.507e-02  21.594  < 2e-16 ***
## accommodates7-8               6.411e-01  3.654e-02  17.545  < 2e-16 ***
## accommodates9+                7.287e-01  4.920e-02  14.810  < 2e-16 ***
## bathrooms2                    2.284e-01  1.790e-02  12.761  < 2e-16 ***
## bathrooms3                    4.086e-01  5.020e-02   8.141 4.47e-16 ***
## bathrooms4+                   5.214e-01  8.067e-02   6.462 1.09e-10 ***
## bedrooms2                     2.318e-01  1.394e-02  16.634  < 2e-16 ***
## bedrooms3                     3.544e-01  2.543e-02  13.935  < 2e-16 ***
## bedrooms4+                    4.846e-01  4.385e-02  11.050  < 2e-16 ***
## maximum_nights               -6.045e-05  1.145e-05  -5.280 1.33e-07 ***
## availability_30               8.129e-03  4.092e-04  19.863  < 2e-16 ***
## stars                         1.244e-01  2.261e-02   5.501 3.89e-08 ***
## stars_checkin                -1.216e-01  2.560e-02  -4.749 2.08e-06 ***
## stars_location                2.459e-01  1.934e-02  12.716  < 2e-16 ***
## reviews_per_month             1.128e-02  2.298e-03   4.910 9.28e-07 ***
## wifiTRUE                     -7.282e-02  1.525e-02  -4.775 1.82e-06 ***
## hair_dryerTRUE                1.354e-01  1.282e-02  10.561  < 2e-16 ***
## hot_waterTRUE                -5.428e-02  1.510e-02  -3.595 0.000326 ***
## minutes_in_descriptionTRUE -5.085e-02  1.248e-02  -4.075 4.65e-05 ***
## host_since                   -2.202e-05  4.276e-06  -5.149 2.67e-07 ***
## first_review                  3.818e-05  5.029e-06   7.592 3.49e-14 ***
## last_review                  -3.884e-05  1.042e-05  -3.727 0.000195 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4505 on 8736 degrees of freedom
## Multiple R-squared:  0.5221, Adjusted R-squared:  0.5205
## F-statistic: 329.1 on 29 and 8736 DF,  p-value: < 2.2e-16
```

```
#plot(lm_inter)
```

Moving towards our final model, we removed several more predictors, aiming for the model to strike a balance between predictive power, parsimony & interpretability:

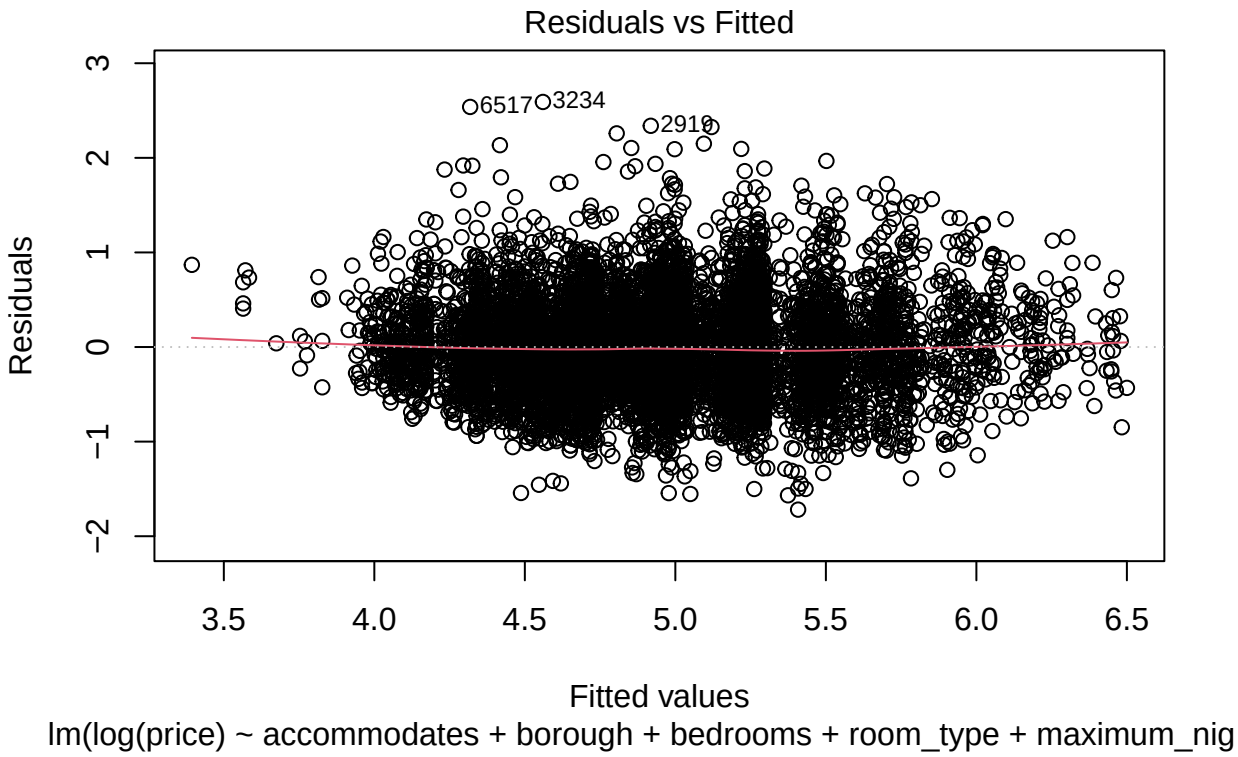
Predictor removed (reason): bathrooms, est_revenue_ly (correlations) ; host_response_rate (too frequently '100') ; reviews_per_month (leverage / infrequent high values) ; hair_dryer, hot_water, wifi, restaurants_in_description, minutes_in_description, host_acceptance_rate, stars_checkin, stars_location, availability_30 (for simplicity / small effect on outcome)

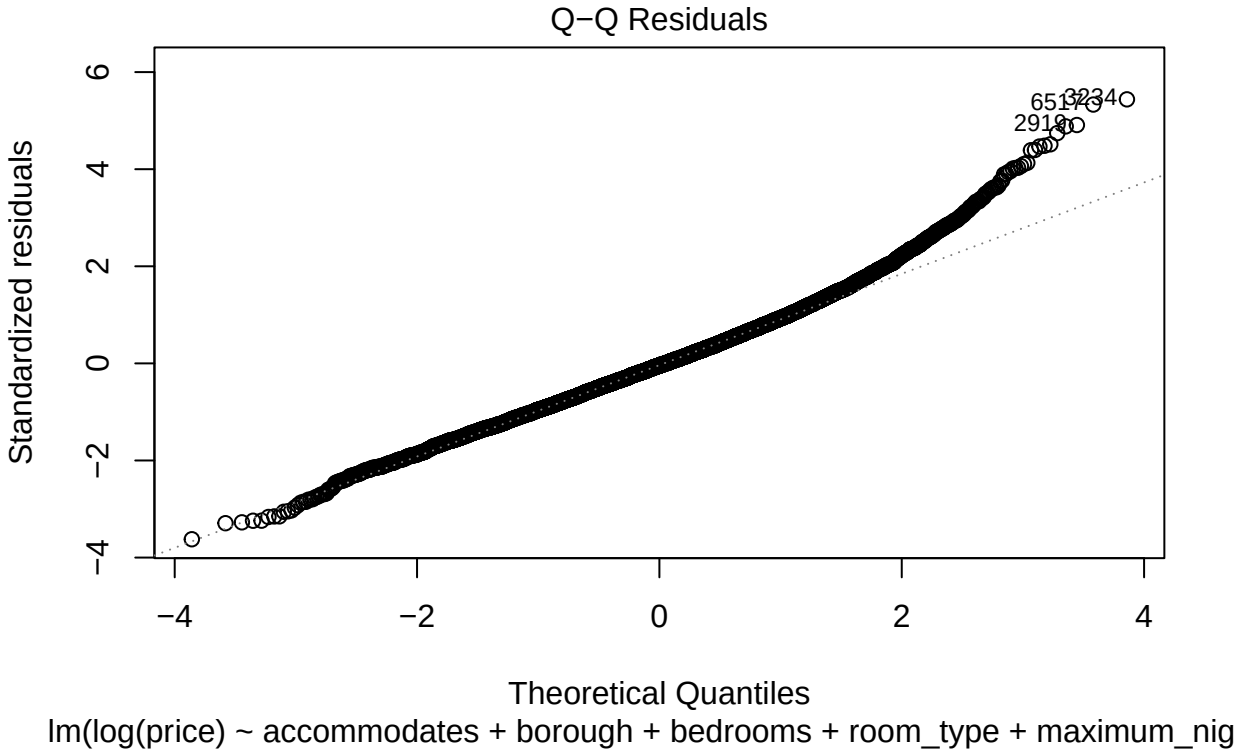
#What I ended up with, chosen for easy interpretation, log necessary for solid modelling

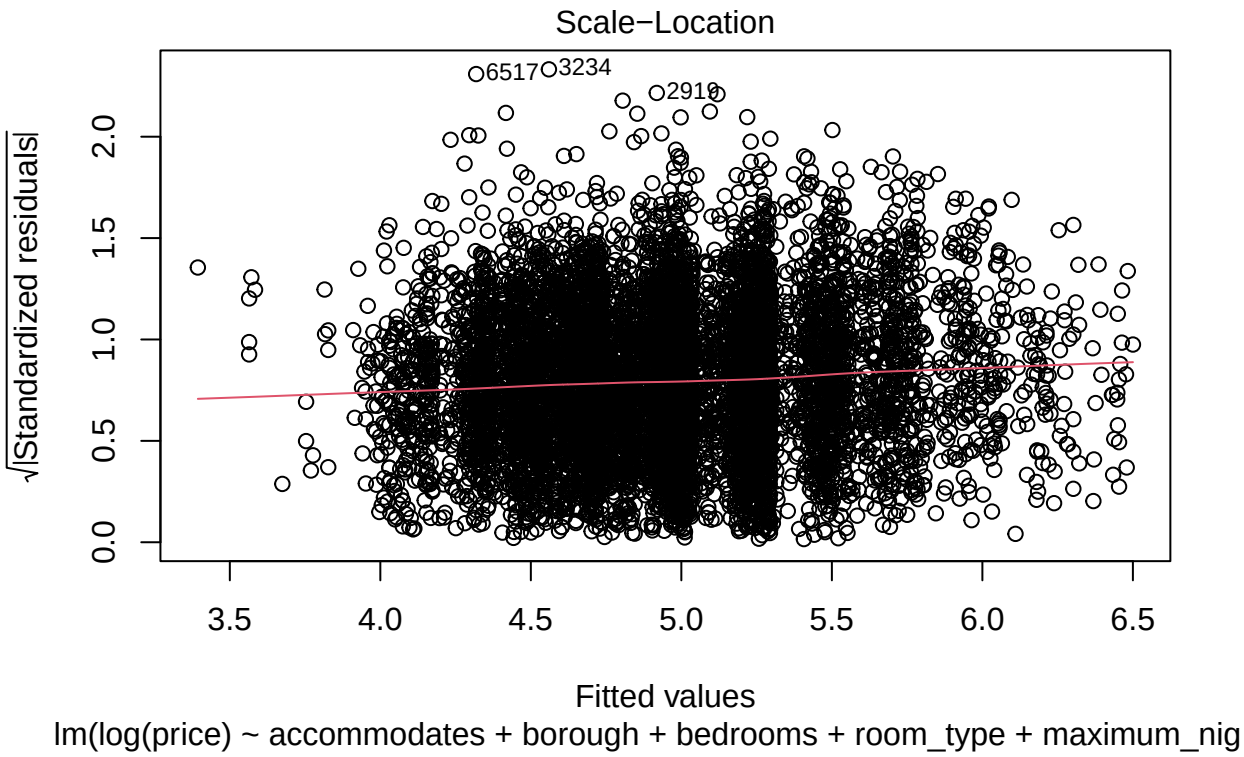
```r
bnb_data2 <- bnb_data2 %>%
  filter(price < 1800)

lm_final <- lm(log(price) ~ accommodates + borough + bedrooms +
                   room_type + maximum_nights + stars,
                   data = bnb_data2)

summary(lm_final)
```

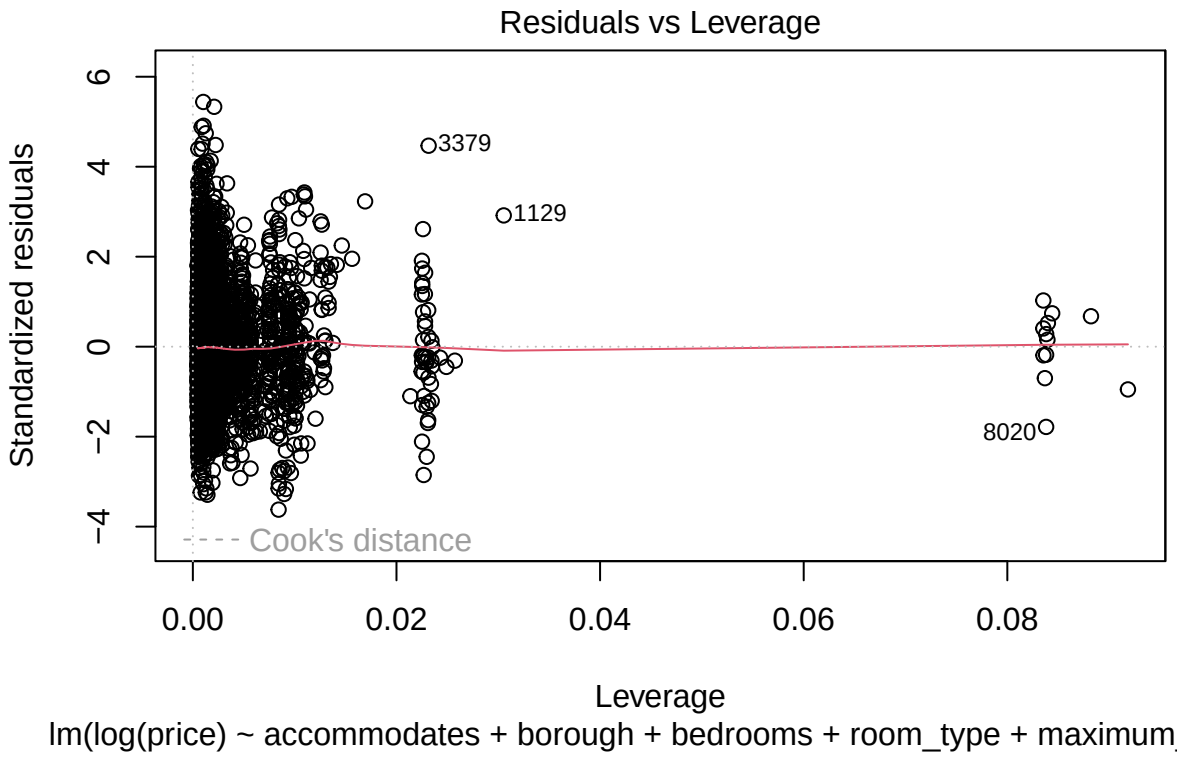
```
##
## Call:
## lm(formula = log(price) ~ accommodates + borough + bedrooms +
##     room_type + maximum_nights + stars, data = bnb_data2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7189 -0.3192 -0.0211  0.2851  2.5907
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               3.423e+00  7.203e-02  47.520  < 2e-16 ***
## accommodates2-4           3.756e-01  1.624e-02  23.125  < 2e-16 ***
## accommodates5-6           6.015e-01  2.617e-02  22.984  < 2e-16 ***
## accommodates7-8           7.105e-01  3.823e-02  18.583  < 2e-16 ***
## accommodates9+            8.718e-01  5.115e-02  17.044  < 2e-16 ***
## boroughBrooklyn           2.654e-01  2.085e-02  12.728  < 2e-16 ***
## boroughManhattan          5.282e-01  2.146e-02  24.615  < 2e-16 ***
## boroughQueens             7.896e-02  2.257e-02   3.498 0.000471 ***
## bedrooms2                 2.639e-01  1.448e-02  18.229  < 2e-16 ***
## bedrooms3                 4.553e-01  2.584e-02  17.617  < 2e-16 ***
## bedrooms4+                6.925e-01  4.297e-02  16.117  < 2e-16 ***
## room_typeHotel room       3.823e-01  1.380e-01   2.769 0.005627 **
## room_typePrivate room    -2.966e-01  1.199e-02 -24.733  < 2e-16 ***
## room_typeShared room     -3.699e-01  7.157e-02  -5.168 2.41e-07 ***
## maximum_nights           -5.828e-05  1.189e-05  -4.901 9.69e-07 ***
## stars                     1.939e-01  1.392e-02  13.923  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4765 on 8750 degrees of freedom
## Multiple R-squared:  0.4644, Adjusted R-squared:  0.4635
## F-statistic: 505.9 on 15 and 8750 DF,  p-value: < 2.2e-16
```

```r
plot(lm_final)
```


Residuals vs Fitted
Residuals
Fitted values
lm(log(price) ~ accommodates + borough + bedrooms + room_type + maximum_nig
6517
3234
2919
3
2
1
0
−1
−2
3.5
4.0
4.5
5.0
5.5
6.0
6.5


Q−Q Residuals
Standardized residuals
Theoretical Quantiles
lm(log(price) ~ accommodates + borough + bedrooms + room_type + maximum_nig
3234
6517
2918
−4
−2
0
2
4
6


Scale−Location
6517
3234
2919
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
Fitted values
lm(log(price) ~ accommodates + borough + bedrooms + room_type + maximum_nig



The result was our baseline model, linear regression. It includes four categorical, and two continuous predictors and has a Rˆ2 value of 0.4644, which is a very solid baseline for such variable data. Looking at the regression coefficients, all are significant at the 0.1% level.

Further examination demonstrates, as expected, that increasing the number of people that the listing can accommodate, the number of bedrooms, or the rating, increases the estimated price per night. A one night increase in maximum nights, however, lowers log(price) by 0.00005828 or price by ~0.005%. A similar listing going from the Bronx -> Manhattan increases log(price) by 0.53 -> price by over 53%. The intercept, or price with all variables at baseline values, is 3.423, giving a price of eˆ3.423 = $31 per night. In contrast, a 3 bedroom, 4 accommodate, Brooklyn home/apt with 30 night max stay, and 4.5 stars will have an estimated price of $219 a night, by taking eˆ(3.423 + 0.3756 + ... + 0.1939 * 4.5).

Viewing the results of plotting our LM, we see adequate dispersion of residuals, though some bunching can be seen at levels like 5.0 and 5.25. This is certainly a result of the substantial influence that categorical predictors have in the model, but, to create an interpretable and useful model, such a dependence is almost required with this dataset, since the most impactful predictors are categorical. The Q-Q is fair, with the lower tail actually being quite well predicted, but we still have a clear issue with the upper tail, even in this reduced set. A final concern are the several high leverage (0.08) points, we refit the model without these and saw no significant change, suggesting that the final fit is robust.

Limitations of this model are easily found: (intentional) simplicity - e.g. no interactions; poor fit for high price listings. Additionally, by using 'stars' and 'maximum_nights', we sacrificed improved predictions for interpretation, since these predictors are the most useful to potential users of our models. Finally, standard LM limitations apply, such as assumption of linear effects, though in this specific case, that is far less of an issue, given our dependence on categorical predictors. Assumption of normal residuals does apply.

##Decision Tree (interpretable)

As our second interpretable model, we elected to fit a simple tree model, citing its highly interpretable nature.

The formula is a direct copy of the 'lm_base' formula, since a tree model will naturally select which predictors give the best fit. So the main choice in tuning this model was the level of complexity that we allowed. As 'cost_complexity' increases from 0.0001 to 0.1, RMSE constantly increases, so the least pruned (most complex) tree is always selected. Thus we must pick a lower limit to promote simplicity. Therefore, we decided on 0.05 as the complexity cost which created an effective, yet interpretable model. Also, we split the data 80/20 s.t. we have OOS observations to test on.

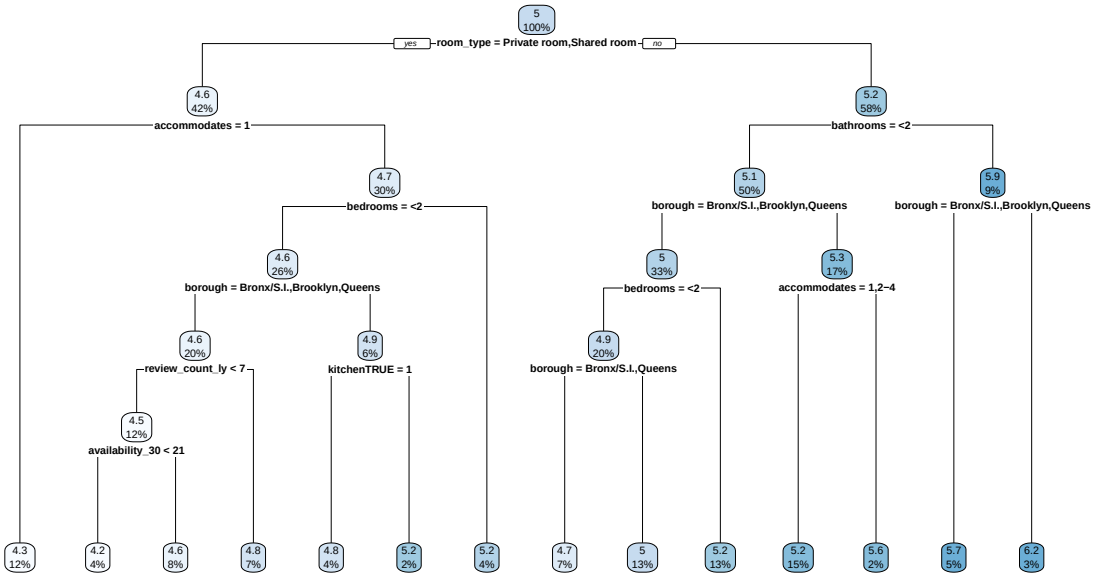
```
#library(rpart.plot)
#library(vip)

set.seed(1)
tsplit <- initial_split(bnb_data2, prop = 0.8)  # will use test to plot predictions
ttrain <- training(tsplit)
ttest  <- testing(tsplit)

tree_workflow <- workflow() %>%
  add_formula(log(price) ~ borough + room_type + accommodates + bathrooms + bedrooms + maximum_nights +
                   availability_30 + availability_90  + review_count_ly + stars +
                    stars_checkin + stars_location + host_listings_count + reviews_per_month +
                    dishes_and_silverware + wifi + kitchen + hair_dryer + essentials +
                    iron + hot_water + hangers + carbon_monoxide_alarm + smoke_alarm +
                    minutes_in_description + restaurants_in_description + subway_in_description +
                    amenity_count + host_since + first_review + last_review) %>%
  add_model(decision_tree(cost_complexity = tune()) %>%   # tune with cv
              set_engine("rpart", model = TRUE) %>%
              set_mode("regression"))

tuned_tree <- tune_grid(
  tree_workflow,
  resamples = vfold_cv(ttrain, v = 10),
  grid = grid_regular(
    cost_complexity(range = c(0.005, 0.1), trans = NULL), levels = 15),
  # no transformation s.t. complexity is not logged
  metrics = metric_set(rmse, rsq)
)

# show how rmse & rsq changes with complexity
tuned_tree %>%
  collect_metrics() %>%
  select(cost_complexity, .metric, mean) %>%
  pivot_wider(                          # one row per complexity cost
    names_from = .metric,
    values_from = mean
  ) %>%
  arrange(cost_complexity)
```

```
## # A tibble: 15 x 3
##    cost_complexity  rmse   rsq
##              <dbl> <dbl> <dbl>
##  1          0.005  0.475 0.461
##  2          0.0118 0.493 0.419
##  3          0.0186 0.513 0.372
##  4          0.0254 0.520 0.354
##  5          0.0321 0.520 0.354
```

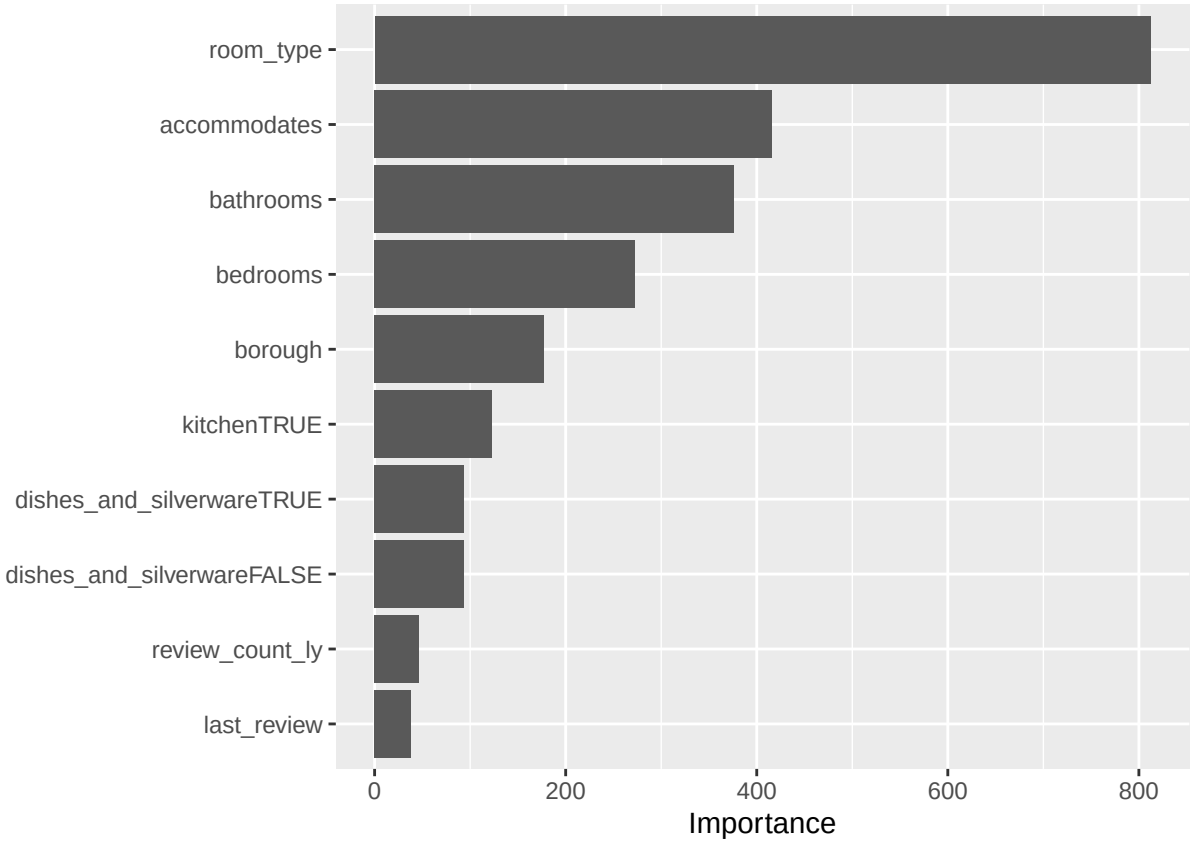
```
##  6          0.0389 0.533 0.321
##  7          0.0457 0.533 0.321
##  8          0.0525 0.533 0.321
##  9          0.0593 0.533 0.321
## 10          0.0661 0.533 0.321
## 11          0.0729 0.533 0.321
## 12          0.0796 0.533 0.321
## 13          0.0864 0.533 0.321
## 14          0.0932 0.533 0.321
## 15          0.1    0.556 0.262
```

Thus, our second interpretable model, a simple tree model. It has up to 6 branches, and utilises similar predictors to the LM, with a few important differences. As mentioned, 'stars' and 'maximum_nights' are not the best variables for prediction, here they are replaced by 'review_count_ly' (no. reviews in the last year), 'availability_30' (no. days available to book in the next 30 days) and, interestingly, 'kitchen', which is an amenity column. These have a reduced role, though, as bed/bathrooms, 'accommodates', borough and room type take precedence. The R^2 is 0.4601, slightly lower than baseline, though this can be increased past .5 by reducing the penalty on complexity. Average RMSE is 0.4750 for our log(price) outcome.

```
# extract the best tree
best_tree <- select_best(tuned_tree, metric = "rmse")
final_tree <- finalize_workflow(tree_workflow, best_tree)
tree_fit <- fit(final_tree, data = bnb_data2)
tree_model <- extract_fit_parsnip(tree_fit)

# plot the tree
rpart.plot(tree_model$fit)
```



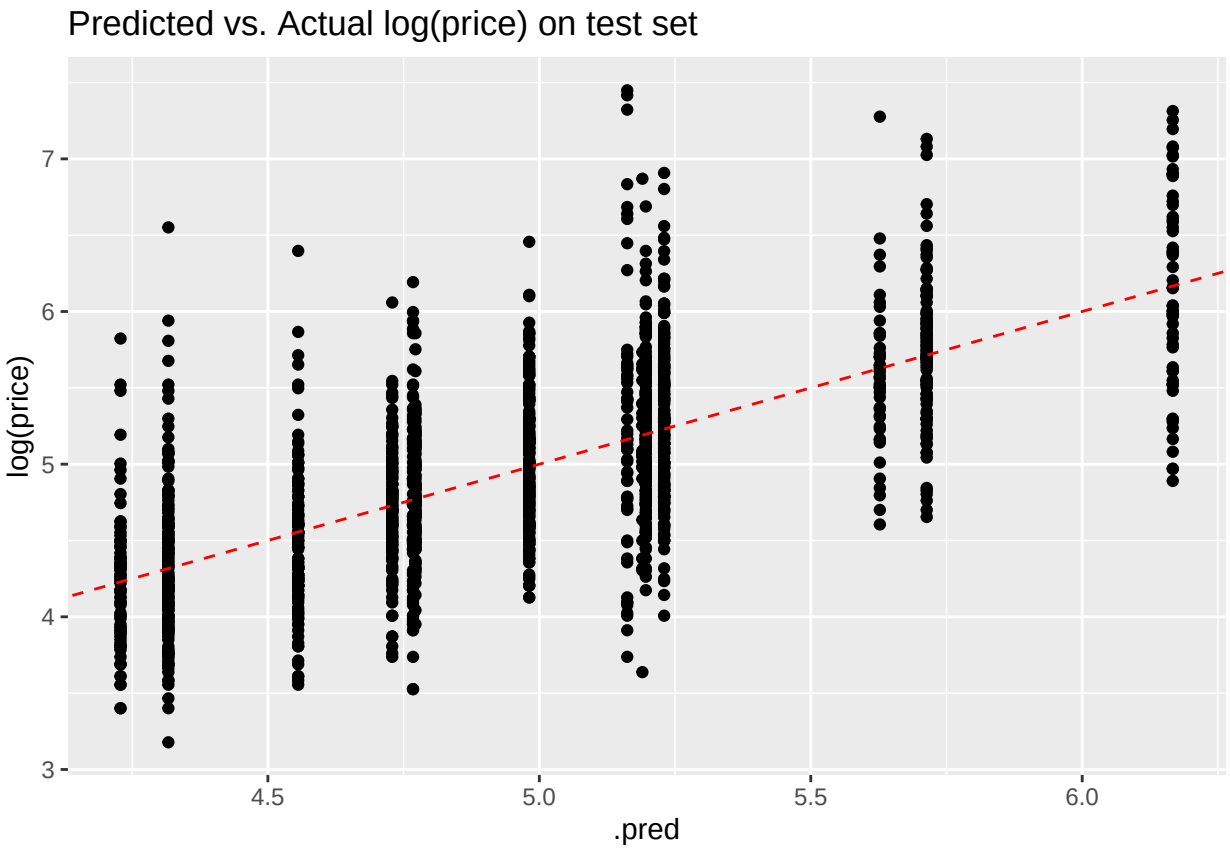

```
# variable importance
vip(tree_model)
```




```
# predict log(price) on test set
ttest_preds <- predict(tree_fit, new_data = ttest) %>%
  bind_cols(ttest %>% select(price))
```

```
## Adding missing grouping variables: `host_id`
```

```
# plot predicted vs. actual
ggplot(ttest_preds, aes(x = .pred, y = log(price))) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Predicted vs. Actual log(price) on test set")
```



Regarding the outcomes, we see that predictions range from 4.2 to 6.2, with 58% lying between 4.8 and 5.2 inclusive. The total range corresponds to prices eˆ4.2 = $67 and $493. We find generally parallel effects to those in the LM: for instance, private and shared rooms immediately lock a listing out of the highest prices (LM coefficients are -0.34 and -0.41). Also the separation between Manhattan and the other boroughs of New York. A listing in Manhattan increases estimated log(price) by at least 0.3. In fact, all the tree needs to know is that a listing is an 'Entire home/apt' (or hotel) with two or more bathrooms, in Manhattan, for a price of $493.

On the left hand side, we find more original results. A review count of fewer than 7 in the last year leads to listing price predictions $67 or $99, where a listing with 7+ reviews is estimated at $122. This 67/99 branch is based on booking availability in the next month, with less availability (<21 days) actually leading to the lower price. Finally, looking at predictions on the test set, we see a roughly linear trend, with s.e. (assuming normal residuals) of ~.6 at each outcome, indicating a reasonable, if somewhat loose, fit on the data.

Through this brief summary, and your own analysis, it can be fairly argued that interpretation of a log outcome is easier with this model, compared to the LM. Without coefficients to deal with, we can directly exponentiate any number on the tree to view the corresponding price. Even so, there are limitations to this model. Similarly to the LM, the induced simplicity reduced predictive power, by pruning away potentially useful predictors. Additionally, the determined range of $67 - $493 doesn't include around 14% of observations, meaning that listings with rare characteristics will not be well predicted.

##Gradient Descent ()

For our third model, we use gradient descent to minimize the difference between predicted and actual prices by iteratively adjusting coefficients to reduce error. We selected a subset of predictors relevant to log(price): maximum_nights, accommodates, bedrooms, bathrooms, review_count_ly, stars, borough, and room_type. This formula is similar to that of the LM, with a few useful predictors added on. We transformed categorical variables to dummy variables and standardized numeric variables. Matrix X includes an intercept term to ensure the model could fit a baseline price level.

```r
model_data <- bnb_data2 %>%
  ungroup() %>%   # drop host_id
  select(maximum_nights, borough, room_type, bathrooms, bedrooms,
         accommodates, review_count_ly, stars)

# log(price) as target var
y <- log(bnb_data2$price)

X <- model.matrix(~ ., data = model_data)[, -1]
numeric_cols <- c("accommodates", "review_count_ly", "stars", "maximum_nights")

# Confirm columns exist in X
numeric_cols_in_X <- numeric_cols[numeric_cols %in% colnames(X)]
#print(numeric_cols_in_X)

# Scale only those found
X[, numeric_cols_in_X] <- scale(X[, numeric_cols_in_X])
X_scaled <- cbind(1, X)  # manually add intercept
colnames(X_scaled)[1] <- "intercept"

#cat(colnames(X_scaled), sep = "\n")
```

The model begins by initializing all coefficient values (theta) to 0. In each interaction, the model computes the predicted log prices by multiplying matrix X by the current coefficient vector, and calculating the error between the prediction and the log(price) value. Using this error, the gradient is computed which indicates how much and in which direction each coefficient should change to reduce loss. The coefficients are then updated, and the process repeats itself. At each step, we calculate MSE. After training, predictions are converted back to the original price scale and the RMSE is calculated.

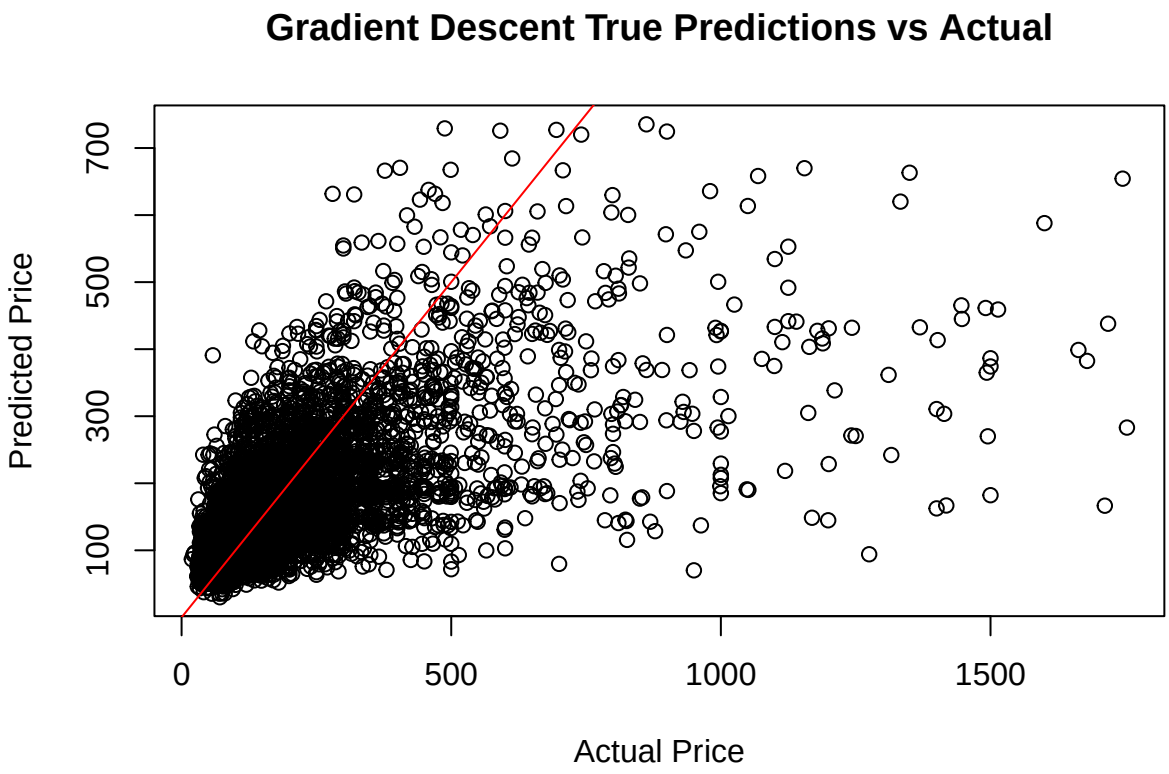
```r
gradient_descent <- function(X, y, learning_rate = 0.01, n_iter = 1000) {
  n <- nrow(X)
  p <- ncol(X)

  # Initialize weights to zero
  theta <- rep(0, p)

  loss_history <- numeric(n_iter)

  for (i in 1:n_iter) {
    y_pred <- X %*% theta
    error <- y_pred - y
    gradient <- (2 / n) * (t(X) %*% error)
    theta <- theta - learning_rate * gradient
    loss_history[i] <- mean(error^2)
  }

  # Predictions
  y_pred <- X %*% theta

  # RMSE- log space
  rmse_log <- sqrt(mean((y - y_pred)^2))
  cat("RMSE (log space):", rmse_log, "\n")
```

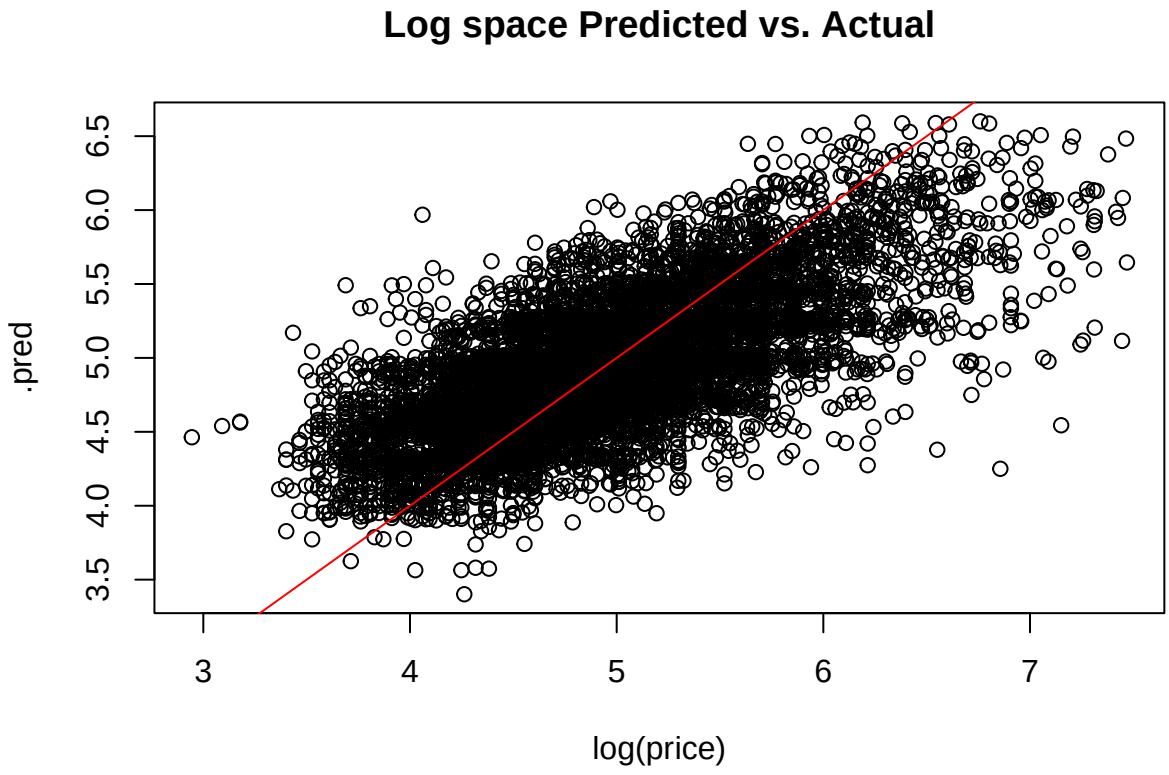
```
  ss_error <- sum((y - y_pred)^2)
  ss_x <- sum((y - mean(y))^2)
  r2_log <- 1 - ss_error / ss_x
  cat("R^2 (log space):", r2_log, "\n")

  # RMSE- price space
  rmse_price <- sqrt(mean((exp(y) - exp(y_pred))^2))
  cat("RMSE (price space):", rmse_price, "\n")
  #preds typically off by $107

  # Plot actual vs. predicted
  plot(exp(y), exp(y_pred), xlab = "Actual Price", ylab = "Predicted Price",
      main = "Gradient Descent True Predictions vs Actual")
  abline(0, 1, col = "red")
  plot(y, y_pred, xlab = "log(price)", ylab = ".pred", main = "Log space Predicted vs. Actual")
  abline(0, 1, col = "red")

  return(list(coefficients = theta, loss_history = loss_history))
}
```

Results

```
# Fit model
gd_result <- gradient_descent(X_scaled, y, learning_rate = 0.02, n_iter = 4000)
```

```
## RMSE (log space): 0.4709805
## R^2 (log space): 0.4759179
## RMSE (price space): 126.5773
```


Gradient Descent True Predictions vs Actual
Predicted Price
100
300
500
700
0
500
1000
1500
Actual Price




```
# Print the coefficients
cat("Gradient Descent Coefficients:\n")
```

```
## Gradient Descent Coefficients:
```

```
coef_names <- colnames(X_scaled)
names(gd_result$coefficients) <- coef_names
print(round(gd_result$coefficients, 4))
```

```
##                            [,1]
## intercept                4.2503
## maximum_nights          -0.0212
## boroughBrooklyn          0.3247
## boroughManhattan         0.5873
## boroughQueens            0.1484
## room_typeHotel room      0.0800
## room_typePrivate room   -0.3002
## room_typeShared room    -0.1836
## bathrooms2               0.2105
## bathrooms3               0.3550
## bathrooms4+              0.2884
## bedrooms2                0.2439
## bedrooms3                0.3933
## bedrooms4+               0.5838
## accommodates2-4          0.3849
```

```
## accommodates5-6        0.5797
## accommodates7-8        0.6698
## accommodates9+         0.7617
## review_count_ly        0.0252
## stars                  0.0686
## attr(,"names")
##  [1] "intercept"              "maximum_nights"         "boroughBrooklyn"
##  [4] "boroughManhattan"       "boroughQueens"          "room_typeHotel room"
##  [7] "room_typePrivate room" "room_typeShared room"  "bathrooms2"
## [10] "bathrooms3"             "bathrooms4+"            "bedrooms2"
## [13] "bedrooms3"              "bedrooms4+"             "accommodates2-4"
## [16] "accommodates5-6"        "accommodates7-8"        "accommodates9+"
## [19] "review_count_ly"        "stars"
```

We find that the RMSE (log space) is 0.471, which is a measure of the error in predicting log(price). The RMSE of price space is 126.58, which means that the model's predictions are, on average, off by about $126. Considering the median price is ~$139 and max is ~$1,753, this is a moderate error. This suggests that the model performs well on the majority of Airbnb listings, but struggles on outliers. Looking at the predictions vs. actual plot, we verify this notion: predictions are fair for listings below $500, but have terrible accuracy above this point. Therefore, we can say that our model is adequate for prediction of 96% of listings.

For coefficients, boroughManhattan = 0.5873, is significantly greater than other boroughs like Brooklyn and Staten Island, which makes sense given the area's demand. Room_type Private room = -0.3002, which understandably shows that single rooms carry lower prices than entire homes. Bathrooms, bedrooms, and accommodates all increase log(price) logically; more space leads to higher price. Stars = 0.0686, confirms that higher review ratings lead to higher prices.

##Ridge (high dim, predictive?)

We used ridge regression to handle the high-dimensional task, adding complexity for greater predictive accuracy. One way we added dimensions was by augmented dataset with lower cutoff for amenities (40 amenities) and including 30 common words from 'description'. We used complete cases, with one listing per host. Then, after setting boolean columns to 0-1, and removing uninformative predictors, we are almost ready to attempt the regression.

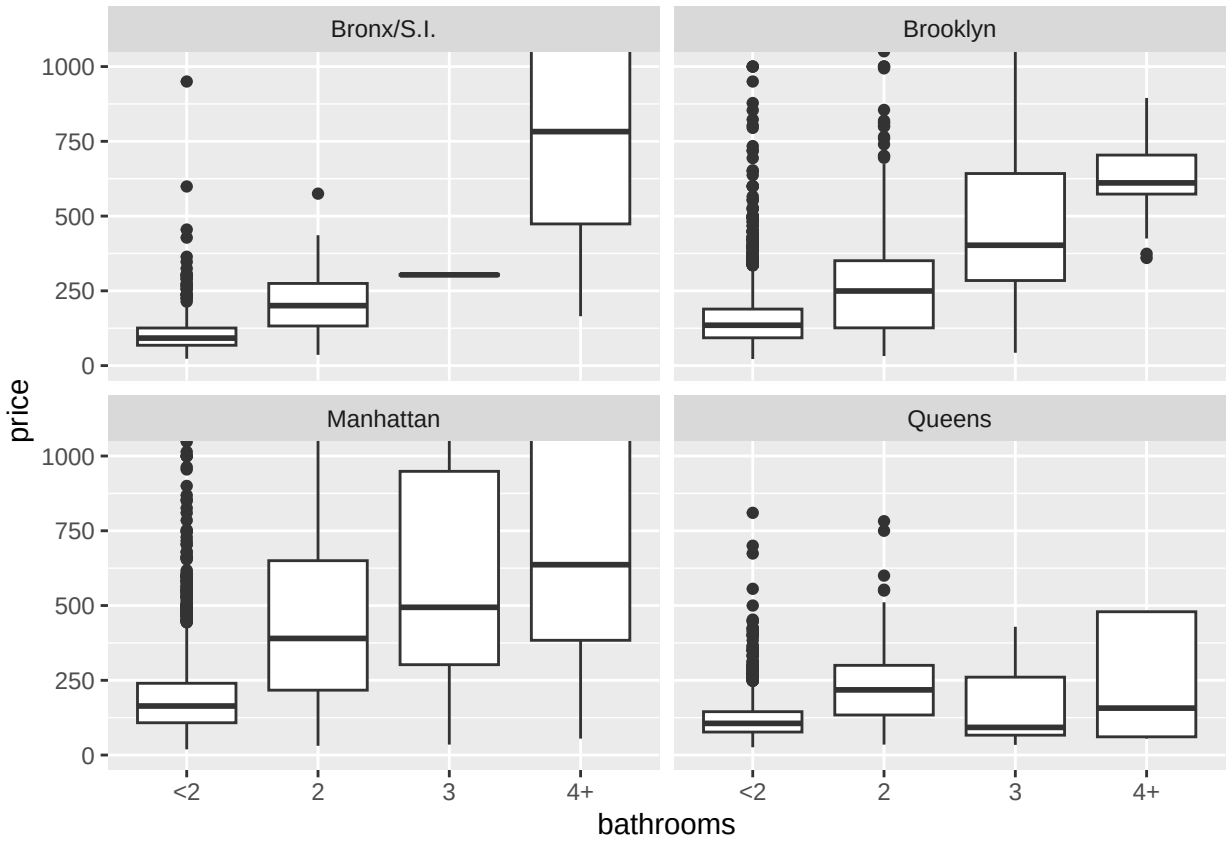
```
# amenities with >5000 occurences
bnb_data3 <- create_amenities(5000)
#Flagged keywords from NLP
bnb_data3 <- create_descriptors(count = 30, data = bnb_data3)

#colSums(is.na(bnb_data3))

bnb_data3 <- na.omit(bnb_data3)
bnb_data3 <- one_listing(bnb_data3)

# removing strings
bnb_data3 <- bnb_data3 %>%
  select(where(~ is.numeric(.) | is.factor(.) | is.logical(.))) %>%
  mutate(across(where(is.logical), as.numeric),
         log_price = log(price)) %>%  # 1/0 instead of TRUE/FALSE
  select(-c(est_revenue_ly, host_id, longitude, latitude))
```

```
## Adding missing grouping variables: `host_id`
```

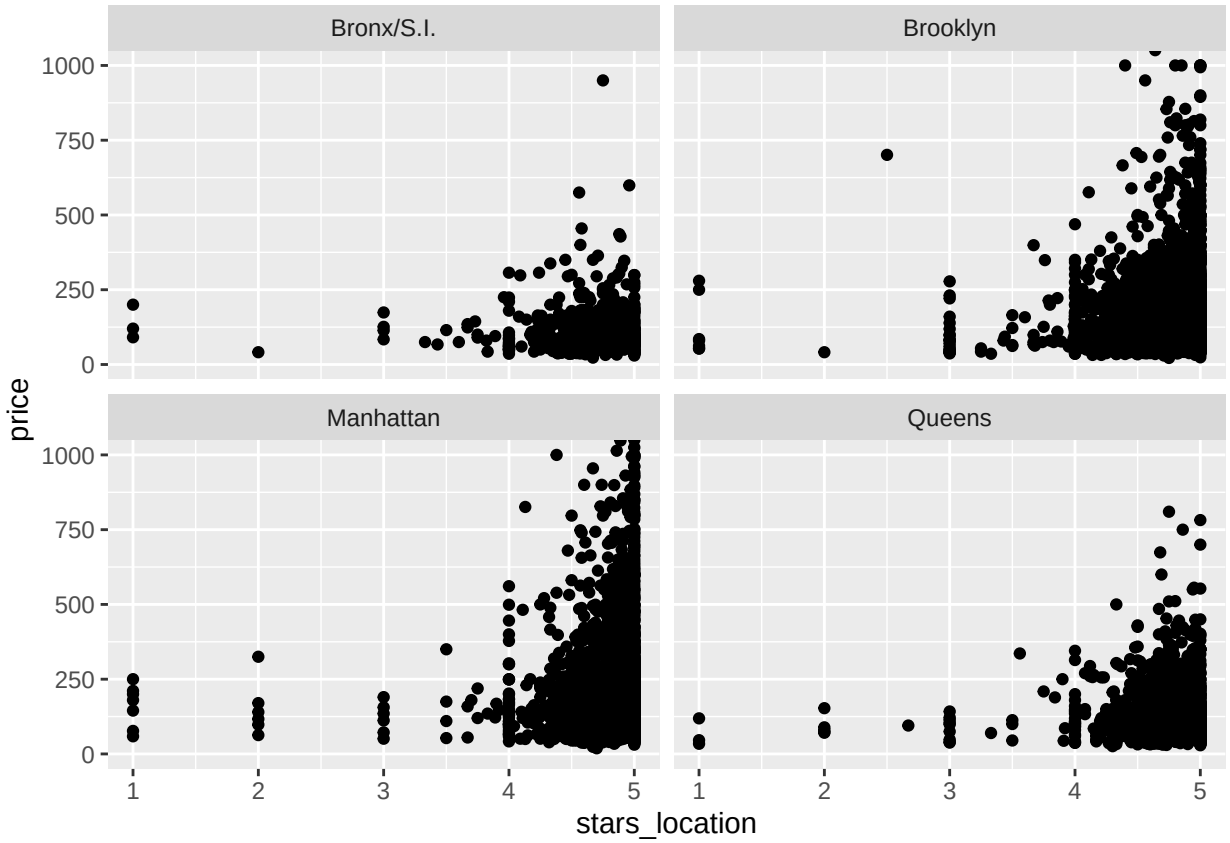
```
# estimated is generated from price; host_id, long, lat nominal

bnb_data3 <- bnb_data3 %>%  # top 20 highest price as with bnb_data2
  filter(price < 1800)

#summary(bnb_data3)
```

Before that though, we also add more dimensions by exploring interactions, including room_type x borough, bathrooms x room_type, bathrooms x borough, and stars_locations x borough. Looking at how borough changes the effect of the number of bathrooms on price, we find that the effect has no clear pattern in Queens, where median price oscillates between $125 and $225. In contrast, in Manhattan, median price goes from ~$175 (<2 bathrooms) to $625 (4+). Here, then, is one instance of interaction.

```
# these plots compare effect of 'x' on price across levels of the predictor in 'facet_wrap'
# bathrooms by room_type seems reasonable, though little (or no) data for some combinations
ggplot(bnb_data3, aes(x = bathrooms, y = price)) +
  geom_boxplot() +
  coord_cartesian(ylim = c(0, 1000)) +
  facet_wrap(~ borough)
```




```
ggplot(bnb_data3, aes(x = stars_location, y = price)) +
  geom_point() +
  coord_cartesian(ylim = c(0, 1000)) +
  facet_wrap(~ borough)
```



Before starting, we split the data into training and test, mainly to combat overfitting and visualise how prediction performed outside the training set. The model encoded categorical predictors (step_dummy), removed low-variance predictors (step_zv), and standardized numeric predictors (step_normalize). To find the optimal penalty value, we performed grid search over 30 penalty levels using tune_grid() with 10-fold cross validation. After careful analysis, we decided on four interactions between predictors, which can be viewed in the code.

```
set.seed(1)
bnb_data3 <- bnb_data3 %>%
  select(-price)

split <- initial_split(bnb_data3, prop = 0.8)              # 20% test data
train_data <- training(split)
test_data  <- testing(split)

ridge_workflow <- workflow() %>%
  add_model(linear_reg(penalty = tune(), mixture = 0) %>%
              set_engine("glmnet")) %>%
  add_recipe(recipe(log_price ~ ., data = train_data) %>%      # considering all predictors
               step_dummy(all_nominal_predictors()) %>%
               step_interact(terms = ~ starts_with("room_type"):starts_with("borough") +
                         starts_with("bathrooms"):starts_with("room_type") +
                         starts_with("bathrooms"):starts_with("borough") +
                         starts_with("stars_location"):starts_with("borough")) %>%
               step_zv(all_predictors()) %>%                 # remove low (no) variance columns
               step_normalize(all_numeric_predictors()))
```

```r
set.seed(1)
cv_folds <- vfold_cv(train_data, v = 10)

set.seed(1)
tuned_ridge <- tune_grid(
  ridge_workflow,
  resamples = cv_folds,    # 10-fold cv
  grid = grid_regular(penalty(), levels = 30),
  control = control_grid(save_pred = TRUE)
)

best_ridge <- select_best(tuned_ridge, metric = "rmse")
final_ridge <- finalize_workflow(ridge_workflow, best_ridge)
ridge_fit <- fit(final_ridge, data = train_data)
```

Metrics

```r
train_data <- train_data %>%
  mutate(price = exp(log_price))
test_data <- test_data %>%
  mutate(price = exp(log_price))

test_results <- predict(ridge_fit, test_data) %>%
  bind_cols(test_data) %>% metrics(truth = log_price, estimate = .pred)

test_resultsexp <- predict(ridge_fit, test_data) %>%
  bind_cols(test_data) %>%
  mutate(.predexp = exp(.pred)) %>%
  metrics(truth = price, estimate = .predexp)

test_results
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.406
## 2 rsq     standard       0.614
## 3 mae     standard       0.319
```

```r
test_resultsexp
```

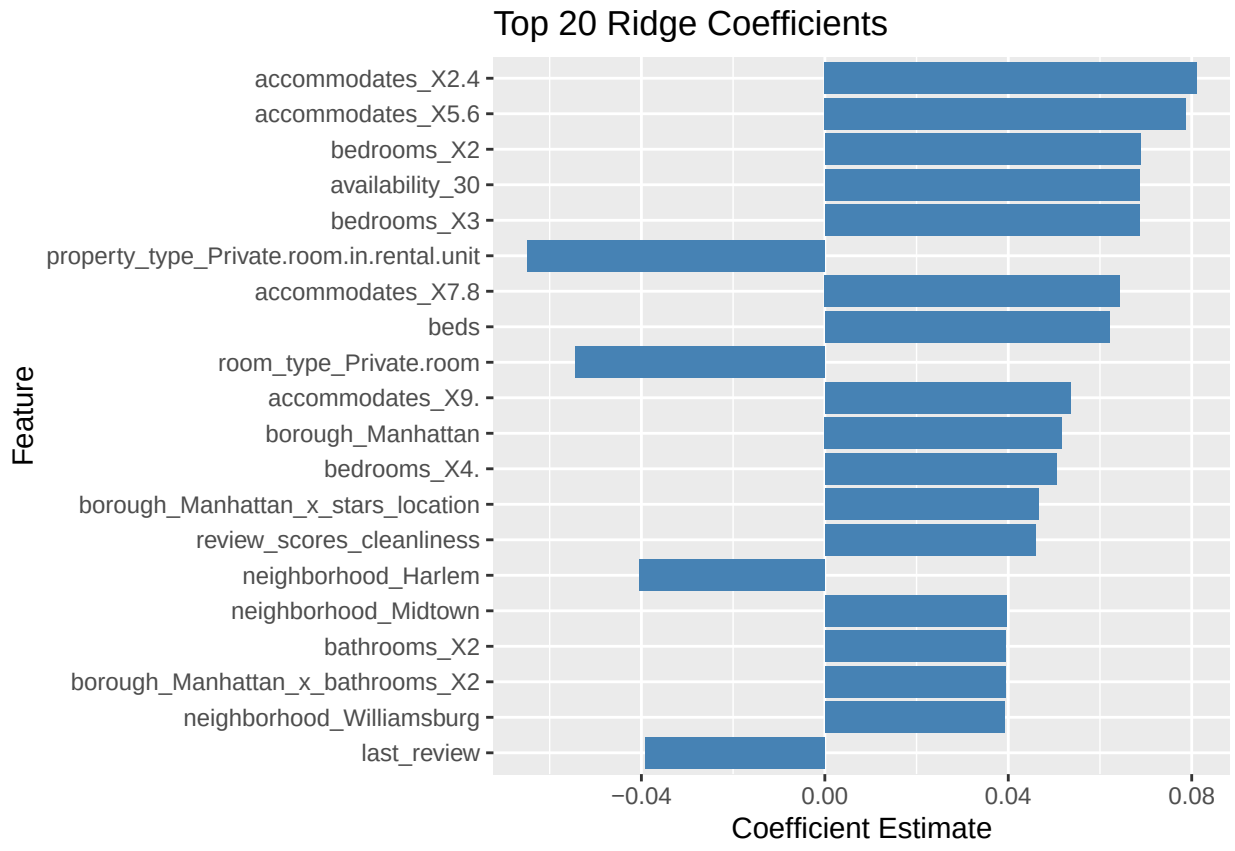
```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard     105.
## 2 rsq     standard       0.540
## 3 mae     standard      59.2
```

For some reason I can't figure out, the knitted file outputs different results, so I'll comment briefly on them first, then what you will find if you run the script. R^2 is 0.614 for log(price) outcome, and 0.540 for price outcome.

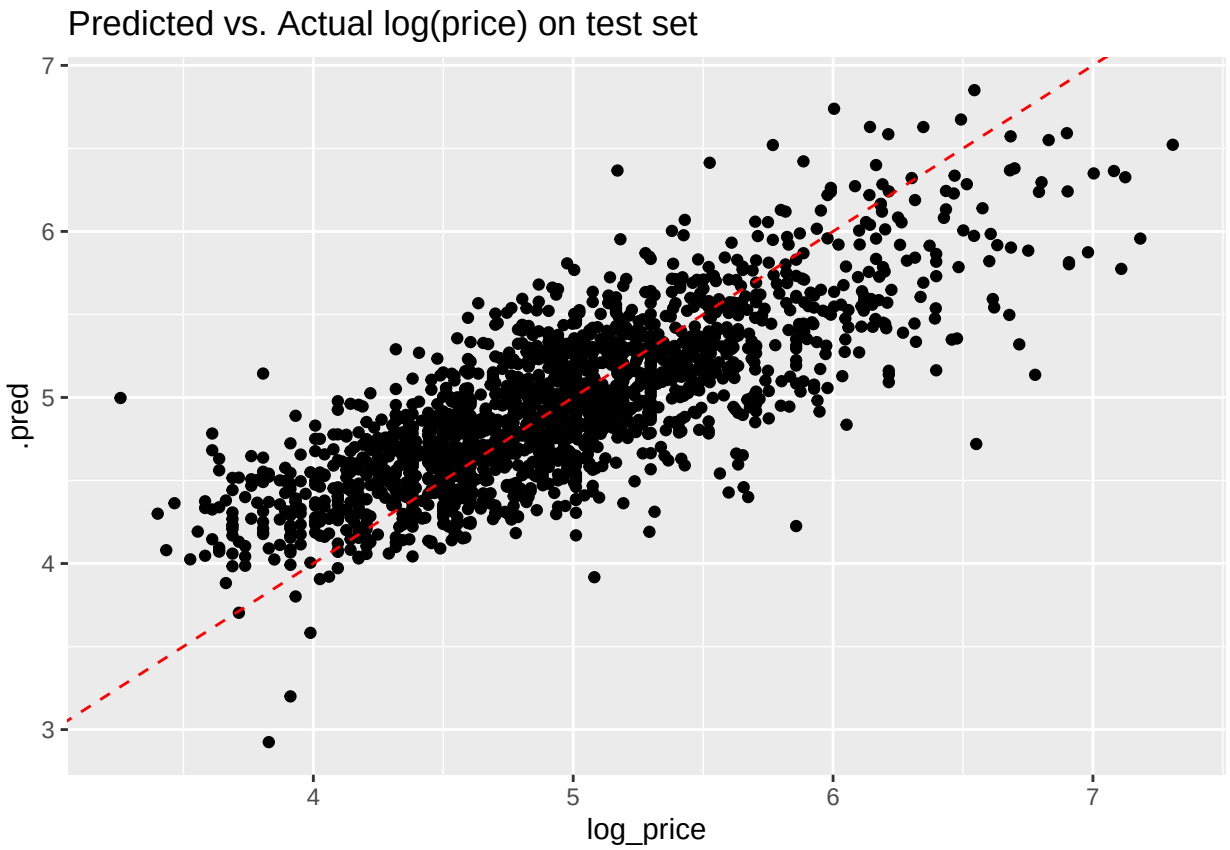
Test data R^2 for this model is 0.607, which is our best yet by far. Of course this does not tell the whole story - even if we only consider this metric, R^2 changes with the training/test sets (we observed 0.54-0.63 when running unseeded). Also, if we view metrics using the true values (i.e. exp(log(price))), R^2 falls to 0.540.

Ridge plots

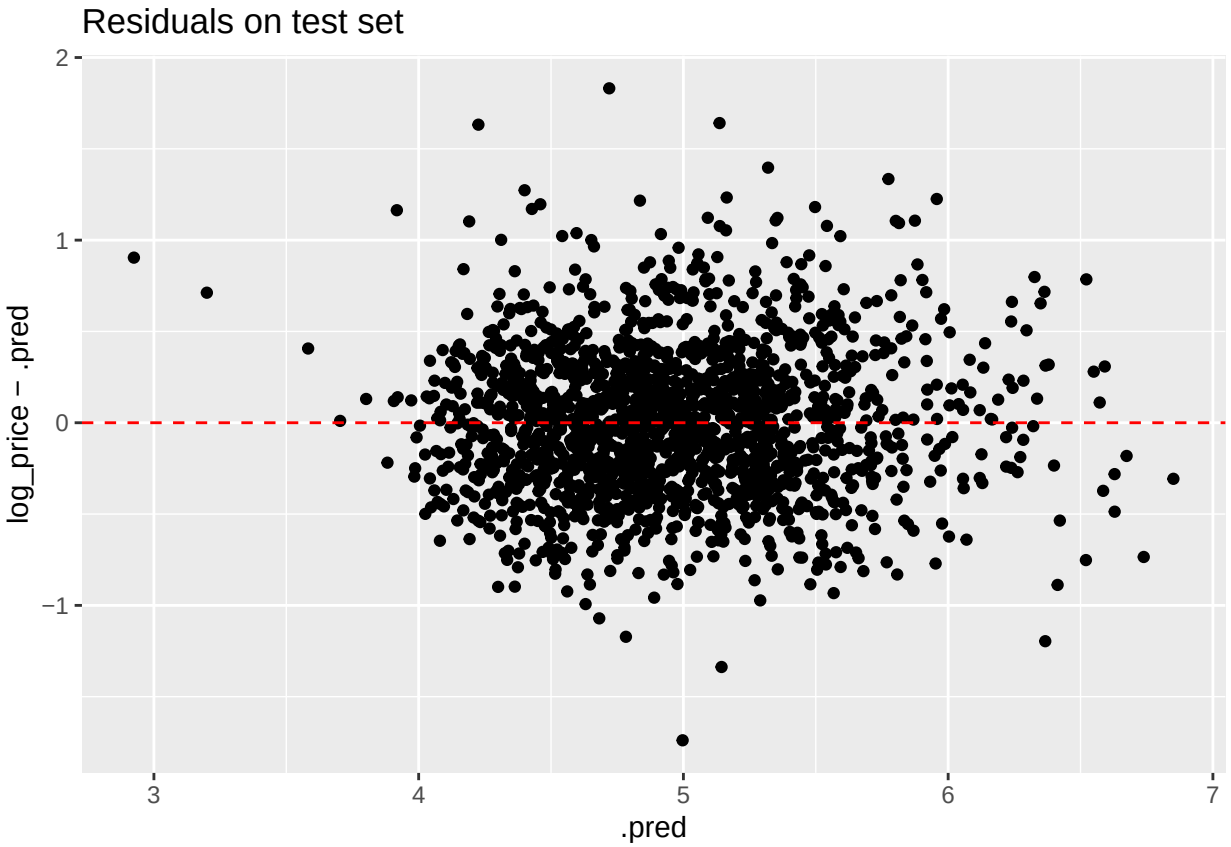
```
library(broom)

# Extract the fitted model object from the workflow
ridge_model <- extract_fit_parsnip(ridge_fit)

# Use `tidy()` to get a data frame of coefficients
coef_df <- tidy(ridge_model)

coef_df %>%
  filter(term != "(Intercept)") %>%
  mutate(abs_estimate = abs(estimate)) %>%
  arrange(desc(abs_estimate)) %>%
  slice_head(n = 20) %>%
  ggplot(aes(x = reorder(term, abs_estimate), y = estimate)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Top 20 Ridge Coefficients",
       x = "Feature",
       y = "Coefficient Estimate")
```



```r
# predict log(price) on test set
test_results <- predict(ridge_fit, test_data) %>%
  bind_cols(test_data)

# plot predicted / actual
ggplot(test_results, aes(x = log_price, y = .pred)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
  labs(title = "Predicted vs. Actual log(price) on test set")
```




```r
# and a residual plot
ggplot(test_results, aes(x = .pred, y = log_price - .pred)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  labs(title = "Residuals on test set")
```



(Knitted file) Residual plot well distributed, we see one quite negative residual, but non with magnitude >2. Predicted actual looks alright, we can see how uncertain the model become for high price properties. Top coefficients are 'accommodates', 'bedrooms'. Otherwise similar to script:

Viewing the residual plot, we see that for the vast majority of listings, the model is a good fit. When running unseeded, however, we found that a couple of residuals usually have value > 2, we argue that these are outliers with unique characteristics / selling points that perhaps our model does not factor in. There is also sometimes a residual below -2, which is more concerning, since we haven't established a precedent for large overestimates.

Viewing coefficients, we find that the most impactful predictor is 'beds', the coefficient is 0.06, which individually increases predicted price by ~6% for each bed in the house. Other predictors we might expect also appear here, 'accommodates', 'availability_30' (days available to book in the next month), 'bedrooms' and the ever present Manhattan. The most negative coefficient is assigned to the dummy variable, 'Private.room.in.rental.unit' with a value of -0.055. No interaction terms appears in the top 20, though we frequently saw 'stars_location' * 'boroughManhattan'. The intuition is clear, as Manhattan is considered a tourist hub, as well as having the most prime real estate. Finally, we see a couple of 'neighborhood' levels appearing, this predictor was excluded from the LM for simplicity.

Predicted vs. actual demonstrates, as ever, that the gradient of prediction w.r.t. price is too low: at prices below e^4 = \$55, we predict too high, while for listings above e^6 = \$400, predictions are almost all too low. A further improvement to this regression would be incorporating further domain knowledge into interactions, since we mainly plotted interactions for predictors that were influential in the LM & tree. Also, the latitude and longitude predictors could be used to measure attributes like distance to tourist attractions, but that is beyond the scope of this project.

##Random Forest Tree (predictive? Seasonality & Inflation)

Because the data is based mainly on listings (with the requirement of 1 review being set), some of the apartments can be presented as outliers, extremely luxurious apartments that can affect square-error. Despite

implementing logistic regression to reduce the impact, implementing random forest can capture the non-linearities and interactions without needing to manually feature engineer, focusing mainly on accuracy rather than interpretability.To guide feature pruning or engineering, we use variable importance out-of-the-box via the ranger engine, which can be used for a higher efficiency. Source:https://parsnip.tidymodels.org/reference/details_rand_forest_ranger.html

```
# use same train/test split from ridge/bnb_data3
set.seed(1)
bnb_data3 <- bnb_data3 %>%
  mutate(price = exp(log_price))

rf_recipe <- recipe(log_price ~ borough + room_type + bathrooms + maximum_nights + last_review + review
                    data = train_data)

rf_specs <- rand_forest(
  mtry =  tune(),
  trees = 1000,
  min_n = tune()
) %>%
  set_engine("ranger",importance = "impurity"
) %>%
  set_mode("regression") #Setting the mode for the ranger engine

rf_workflow <- workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_specs) #The tidymodels workflow bundles the preprocessing step,
#model specification, and tuning information to avoid manually piping the data

set.seed(1)
cv_folds <- vfold_cv(train_data, v = 10) # better default for heavier models, which is the case here

#We consider several combinations, and cross-validate each to tune our random tree model
rf_grid <- expand.grid(
  mtry = 6,
  min_n = c(5,10,15)
)

set.seed(1)
rf_res <- tune_grid(
  rf_workflow,
  resamples = cv_folds,
  grid = rf_grid,
  metrics = metric_set(rmse,mae,rsq) #Output comparable metrics to base model, average the metrics
)
```

```
## Warning: package 'ranger' was built under R version 4.4.3
```

```
#Selecting the best performance
best_pars <- select_best(rf_res, metric = "rmse")
final_rf <- finalize_workflow(rf_workflow,best_pars) %>%
  fit(data = train_data)

#Evaluating the set
```

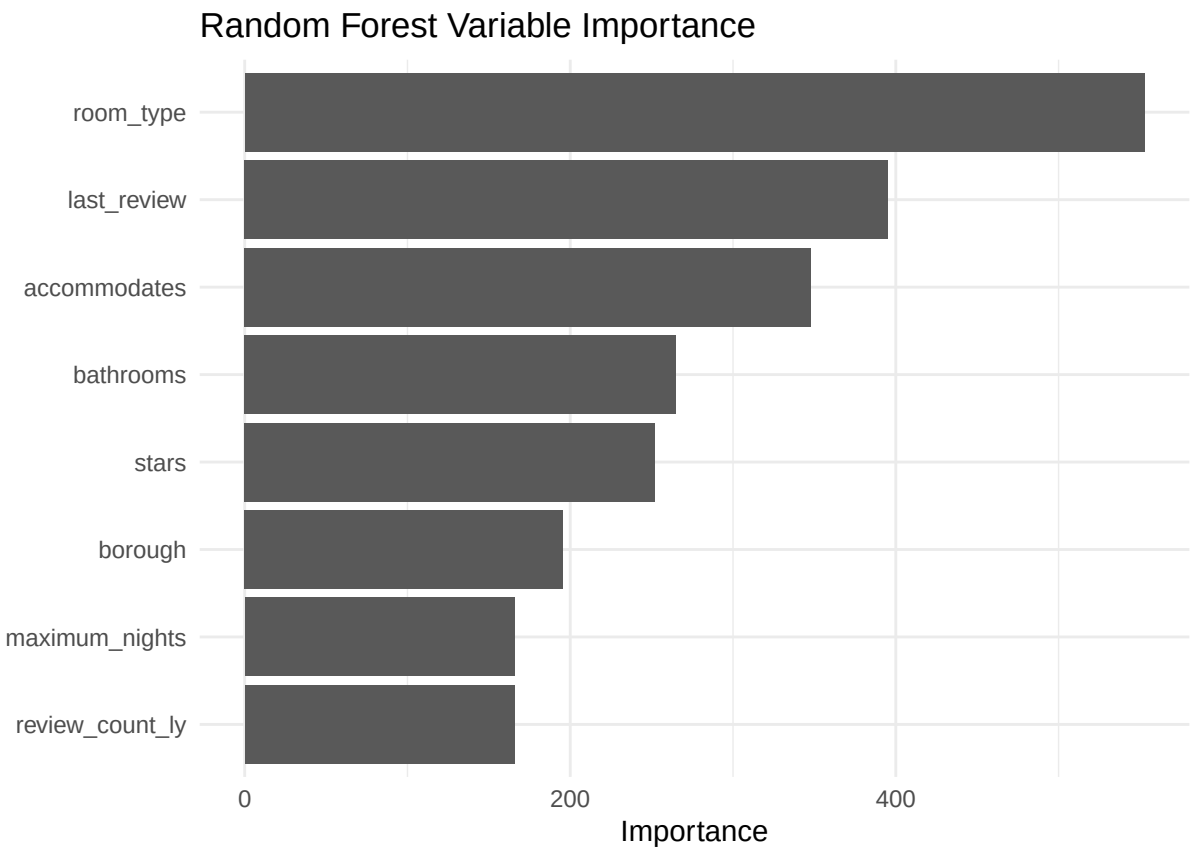
```r
pred_log <- predict(final_rf, test_data)
pred <- exp(pred_log$.pred)

# logged outcome metrics
tibble(Metrics = "logged",
       RMSE = rmse_vec(test_data$log_price, pred_log$.pred),
       MAE = mae_vec(test_data$log_price, pred_log$.pred),
       R2 = rsq_vec(test_data$log_price, pred_log$.pred))
```

```
## # A tibble: 1 x 4
##   Metrics  RMSE   MAE    R2
##   <chr>   <dbl> <dbl> <dbl>
## 1 logged  0.457 0.354 0.510
```

```r
# true value metrics
test_metrics <- tibble( #Bundle the test metrics
  Metrics = "true",
  RMSE = rmse_vec(test_data$price, pred),
  MAE = mae_vec(test_data$price, pred),
  R2 = rsq_vec(test_data$price, pred)
)
test_metrics
```

```
## # A tibble: 1 x 4
##   Metrics  RMSE   MAE    R2
##   <chr>   <dbl> <dbl> <dbl>
## 1 true     117.  65.9 0.418
```

```r
final_rf %>%
  extract_fit_parsnip() %>%
  vip(geom = "col") +
  labs(title = "Random Forest Variable Importance") + theme_minimal()
```




```
#Based on variable importance, attempted to remove the availability_30 and borough variables,
#but R2 reduced significantly, so we remain with the original inclusion of these variables.
```

We create a new model, while considering factors that could influence the dataset such as inflation and season. This is a heuristic approach, as intuitively, the Airbnb rental volume worldwide is not distributed equally throughout the year. However, since data is limited to New York City, this assumption might not influence the data much. For inflation, the oldest data point is in 2008, so inflation could contribute to the factor of price. Exploring these first, and then applying them to the same random forest tree may increase the accuracy of the model.Since inflation will change the data around 50%, we use "stops", so altering data from 2010-2015, 2015-2020, and leave 2020-2025 unaffected.Source for the CPI Inflation calculator is: https://www.bls.gov/data/inflation_calculator.html. Since the properties have different dates, we use the last_review column, as the price is the current one, meaning that using the first_review column data could be wrong for a property that is still active for more than 15 years.

(Knitted file) R^2 are 0.510 and 0.418 for logged and true outcome respectively. Better than LM.

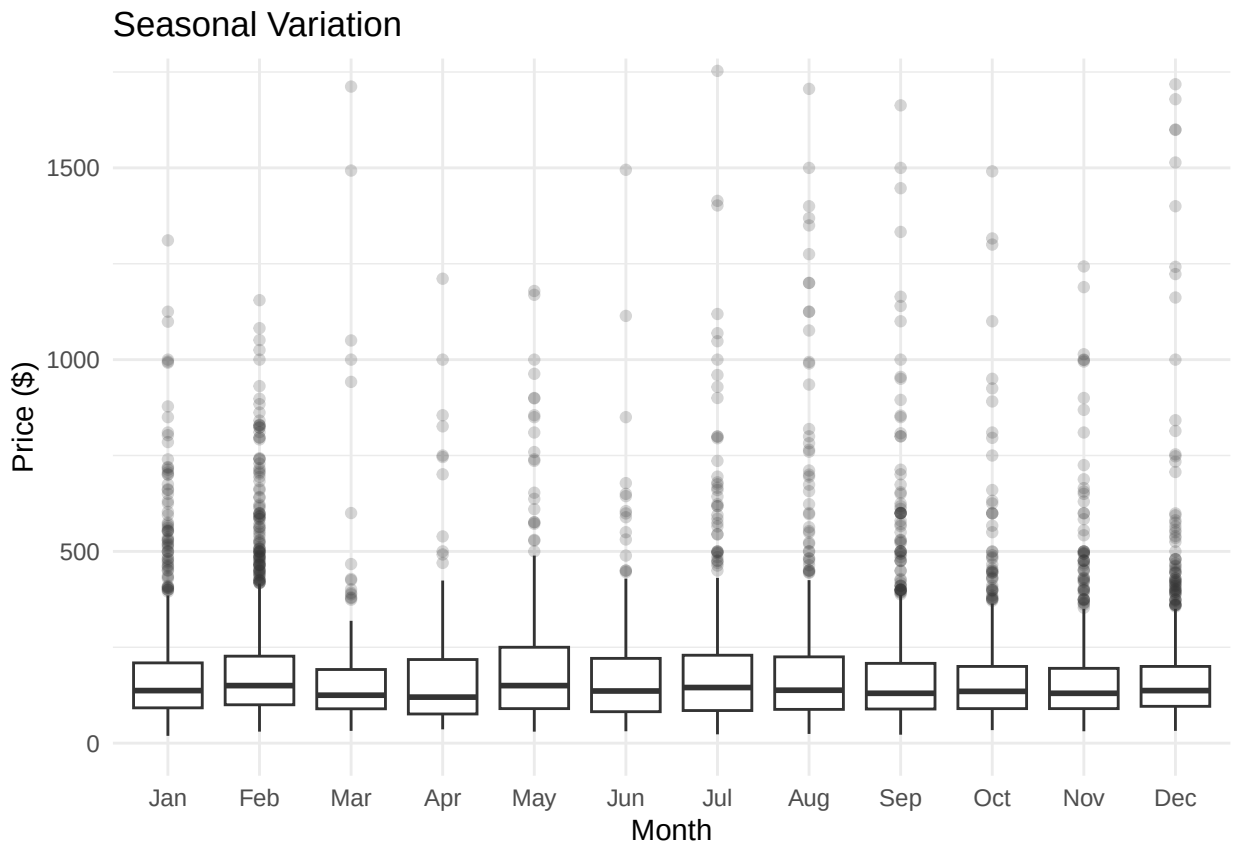
For the model, we use grids to evaluate the best depth of the tree. Additionally, after tuning with several values, additionally, we evaluated that because of the subset of data being used, a 10-fold implementation was an approach that would not exceed the required computational power. This model performed substantially better than our linear baseline, with a log outcome R^2 of 0.526. The most important variable was 'room_type' (similar to decision tree) and second, interestingly, was 'last_review'. Indicating the impact of having an active listing.

```
bnb_time <- bnb_data3 %>%
  filter(!is.na(last_review)) %>%
  mutate(
```

```
    last_review = as.Date(last_review),
    year = year(last_review),
    month = month(last_review, label=TRUE,abbr=TRUE)
  )

bnb_time %>%
  group_by(year) %>%
  summarise(avg_price = mean(price,na.rm = TRUE)) %>%
  ggplot(aes(x=year,y=avg_price)) +
  geom_line(size=1) +
  geom_point(size = 2) +
  labs(
    title= "Average Nightly Price by Year",
    x= "Year",
    y= "Price ($)"
    ) + theme_minimal ()
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```r
bnb_time %>%
  ggplot(aes(x = month, y = price)) +
    geom_boxplot(outlier.alpha = 0.2) +
    labs(
      title = "Seasonal Variation",
      x = "Month",
      y = "Price ($)"
    ) +
    coord_cartesian(ylim = c(0,1700)) +
    theme_minimal()
```



From the plots, it can first be observed that in August and September, we can see that some prices can be relatively higher than others. We can see a trend in April and May where the boxplots have a lower median than other months. From the average nightly price per year, it is not a linear pattern, however, there are two historical trends that affected both the Airbnb rentals and hotel services. Firstly, in 2013, the US went through a housing crisis, affecting all sectors of real estate. Similarly, during the Covid pandemic, there was a massive impact in the hotel and rental sectors. However, the overall hypothesis that there was going to be an upward trend because of inflation was not present, however it must still be conidered. Airbnb is founded in 2008, and in 2012, it might have still be considered a luxury service. However, as time went on and supply for Airbnb increased, the larger flow of properties being rented could be attributed to a lower price. For this reason, the variance in price can occur by several factors we dont have control of (rental protests in the US, tourism rates, competitors, the overall performance of Airbnb, etc).Hence, we account for inflation and for the months where there are the price increases because of seasonal changes.

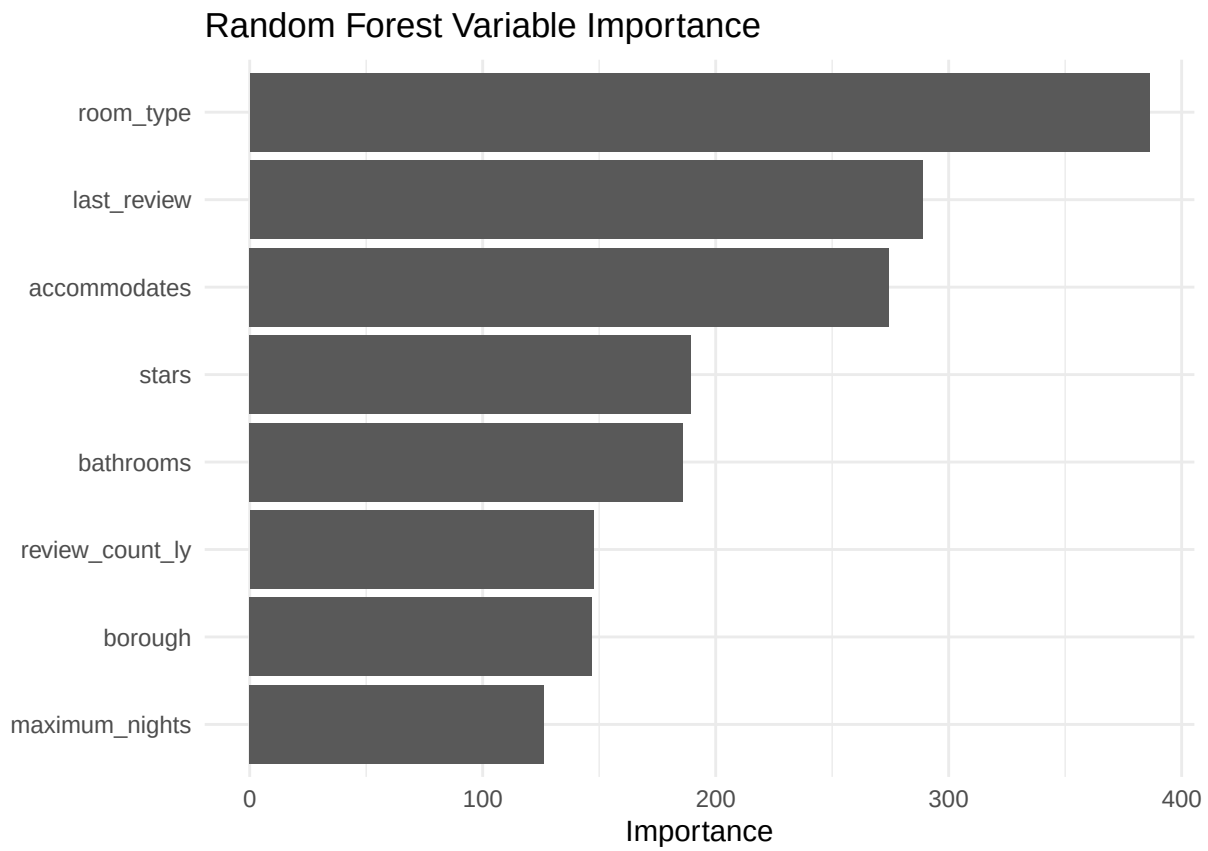
From exploring the data even further, we realized that the data points previous to 2017 are equal to 5, meaning that these are listings can be considered as inserting bias to our model. If we decide to include these listings, then the last_review column (composed of date) would predict the data point on an extremely

small subset of the data, not making the model generalizable for these years. We also decided to drop 2020, because of the influence of the pandemic in Airbnb listings. Additionally, we decided to remove the months of April, August, and September. Firstly, April is removed as it heavily underperforms against other months, with the opposite occurring for the months of August and September. Hence, by dropping these cases, we arrive at a model where the listings are most closely related to the "average" performance of an Airbnb listing, rather than their positive or negative effect through seasonal patterns.

Furthermore, we decide to implement inflation as a way to control price changing.This transformation implements the years after 2017 until 2020 to be multiplied by 1.3, closely resembling the real purchasing power that the dollar has in 2025. With these changes being implemented, we expect our model to perform better because of the reduced uncertainty presented by the aforementioned external factors.

```
set.seed(1)
bnb_seasonal <- bnb_data3 %>%
  filter(!is.na(last_review)) %>%
  mutate(
    last_review = as.Date(last_review),
    year = year(last_review),
    month = month(last_review, label=TRUE,abbr=TRUE)
  )

bnb_data3_seasonal <- bnb_seasonal %>%
  filter(
    !month %in% c("Apr", "Aug", "Sep"),
    year >= 2017,
    !year %in% c(2020)
  )

bnb_data3_inflation <- bnb_data3_seasonal %>%
  mutate(
    price_inflated = if_else(year >= 2017 & year <= 2020,
                             price *1.3,
                             price),
    log_price = log(price_inflated),
    last_review = as.numeric(last_review)
  )

split <- initial_split(bnb_data3_inflation, prop = 0.8)
train_data <-training(split)
test_data <- testing(split)

final_rf_recipe <- recipe(log_price ~ borough + room_type + bathrooms + maximum_nights +
                            last_review + review_count_ly + accommodates + stars,
                  data = train_data)

final_rf_specs <- rand_forest(
  mtry =  tune(),
  trees = 1000,
  min_n = tune()
) %>%
  set_engine("ranger",importance = "impurity"
) %>%
  set_mode("regression")
```

```r
final_rf_workflow <- workflow() %>%
  add_recipe(final_rf_recipe) %>%
  add_model(final_rf_specs)

set.seed(1)
cv_folds <- vfold_cv(train_data, v = 10)

rf_grid <- expand.grid(
  mtry = 6,
  min_n = c(5,10,15)
)

set.seed(1)
rf_res <- tune_grid(
  final_rf_workflow,
  resamples = cv_folds,
  grid = rf_grid,
  metrics = metric_set(rmse,mae,rsq)
)

best_pars <- select_best(rf_res, metric = "rmse")
inflation_final_rf <- finalize_workflow(final_rf_workflow, best_pars) %>%
  fit(data = train_data)

pred_log <- predict(inflation_final_rf, test_data)
pred <- exp(pred_log$.pred)

# true value metrics
final_test_metrics <- tibble(
  Metrics = "true",
  RMSE = rmse_vec(test_data$price, pred),
  MAE = mae_vec(test_data$price, pred),
  R2 = rsq_vec(test_data$price, pred)
)

inflation_final_rf %>%
  extract_fit_parsnip() %>%
  vip(geom = "col") +
  labs(title = "Random Forest Variable Importance") + theme_minimal()
```




```
# log outcome metrics
tibble(Metrics = "logged",
       RMSE = rmse_vec(test_data$log_price, pred_log$.pred),
       MAE = mae_vec(test_data$log_price, pred_log$.pred),
       R2 = rsq_vec(test_data$log_price, pred_log$.pred))
```

```
## # A tibble: 1 x 4
##   Metrics  RMSE   MAE    R2
##   <chr>   <dbl> <dbl> <dbl>
## 1 logged  0.462 0.355 0.482
```

```
final_test_metrics
```

```
## # A tibble: 1 x 4
##   Metrics  RMSE   MAE    R2
##   <chr>   <dbl> <dbl> <dbl>
## 1 true     106.  62.0 0.418
```

(Knitted file) R^2 got worse for log outcome (0.510 -> 0.482), and stayed the same for true outcome (0.418), when compared to the intial random forest. Not a great showing for our friends inflation and seasonality. But perhaps this highlights the difficulty of defining time in this dataset, since all we have to go off of is 'last_review'. I'm certain that results would be better could be more sensibly approximated. Though I know time series data is not wholly appropriate for the methods learned in this course.

After these transformations, we present the final random forest model, comparing it to the baseline random forest model that did not take seasonality or inflation into account. The R^2 for the log outcome decreased

slightly to 0.512; however, for the true values, it increased to 0.447, meaning that the variability in the data can be explained by a greater percentage than in our older random forest model. By controlling these variables, we are increasing the prediction accuracy in our model, however these also serve as indicators that there is a presence of external values that are not integrated into the dataset, and that could influence the results of our models.

```
# use multiple cores if you'd like (random forests take a few minutes to run)
#library(doParallel)
#registerDoParallel(makeCluster(6))
```

###Limitations

As it has been seen through the modifications conducted to the models, the presence of external factors can heavily influence an Airbnb listing. However, despite having access to several predictors, these can not fully explain variability in the price of the listings, because of the influence of factors not accounted for in the data. As Airbnb is a rental service, it is often affected by the overall trends that would affect both real-estate and hotel industries. For this reason, the underperformance in some models can be explained by these factors. For example, before accounting for the year, it was explored how the Covid-19 pandemic and the housing market crash both influence the pricing for these years. While both of these are more general known, more specific trends involving the location of the listings, which in this case is exclusively New York, can occur and affect the variance of price in an unpredictable way. One example is what is happening in Spain, where the Spanish government is calling for the removal of 66,000 property listings because of a breach in tourism rules, according to the BBC, and while this event did not affect our dataset, it leaves an important precedent to a potential hidden impact on Airbnb listings when not acknowledging these factors. These are the types of factors that may have affected Airbnb listings in New York in the past decade, that the models can not account for.

Another limitation for our data is the nature of time-series data, and the potential of the error terms being correlated. As Airbnb depends heavily on tourism, the months in which New York has a higher volume of tourists would impact the Airbnb listings. This seasonal expectation comes from demand, and how often Airbnb hosts would raise the price in certain months as they expect for people to pay more for an apartment in summer as opposed to winter. Additionally, this factor could also be influenced by the day of the week, which was not analyzed because of the scope of the dataset, and because of the "last_review" not being the best value to evaluate a day of the week, because of the column representing the last date of a review, rather than the date of booking.

Source: http://bbc.com/news/articles/c3wdd8lg581o

###Conclusions:

In conclusion, having fit five different machine learning models, we find that our dataset was not sufficient to consistently predict the price of Airbnb listings in New York. The main issue was in predicting the highest priced rentals. However, we find that the models generated were certainly useful for the majority of Airbnb users (as 96% of listings are <$500 a night); both those who are having difficulty deciding where to stay, and those unsure of the price to list their property at.

We propose ridge regression as the method that yielded the best predictions. Through 10-fold cross validation and an 80/20 training/test split, we identified the best possible model, without overfitting. Looking at the metrics, the R^2 was 0.540 on true values of price and 0.607 on the log outcome, our best results by quite a margin. Ridge was also our high-dimensional model.

The simplest models, LM & decision tree, saw a tradeoff between predictive power and interpretation. While the log outcome makes the coefficients of our baseline model more difficult to understand, the decision tree is truly interpretable, as one only has to exponentiate the outcome. Thus, these combine to provide potential users with a way to easily figure out the price that they might have to pay for the type of property they want to rent (or of their property). Of course, a user could add predictors to the baseline LM to better understand their effect on price.

We also employed gradient descent as an alternative method, it was by plotting true (exponentiated) predictions and actual price here that we began to see the scale of underestimation for properties priced above $500 a night. The final (log outcome) R^2 of this model was 0.476.

Finally, a more exploratory model was our random forest. We attempted this in the hopes of predicting top shelf listings more effectively. By including factors like the seasonality of listing prices and inflation, we hoped to better encapsulate changes in the market value of properties. In terms of pure prediction, this was our second strongest model, with a final (log outcome) R^2 of 0.512, however, this performance was slightly worse than running a random forest directly on 'bnb_data3', which had R^2 0.526

For each model, predictors were carefully selected to give the best fit. The LM and Basic Tree placed 'accommodates', 'borough', 'bedrooms', 'bathrooms' and 'room_type' as the most important predictors. For every subsequent model, we went in with these as our baseline, and tuned from there. Other predictors selected for interpretability were 'stars' and 'maximum_nights', as we believe these would be useful to Airbnb users. 'Review_count_ly' and 'availability_30' also appeared with some constancy, and for the ridge model, where 'neighborhood' and 'property_type' could be efficiently incorporated, they were a useful variables.

Our project has potential to be greatly extended. Predicting prices is a difficult game, and there are a myriad of factors to consider. Our random forest utilised seasonality and inflation, on top of the basic data. Given time, it is tempting to explore: greater use of latitude and longitude data, comparison to other cities and a more accurate use of time (since this data had only 'last_review', 'first_review', 'host_since') are the first opportunities that come to mind.

After a thorough exploration of the data, we employed five different machine learning methods to predict the price of New York Airbnb listings. Our most effective models were ridge and LM/decision tree, for predictions and interpretation respectively. Gradient descent played a key role, helping us to identify our project's biggest weakness - the poor fit for highly priced properties. Through random forest we began to examine additional factors and, thus, all models proved vital in creating a well-rounded project. We were not able to fully meet our primary goal, as predictions for high priced listings were simply inadequate across all models. However, if we limit our reach, discounting just 4% of the most expensive properties, we were able to provide solid predictions for the price of Airbnb listings in New York, in models that could be used by the average Airbnb user; and, for the average user, we argue that this 96% of listings proves most important.