Linear Regression

Art Tay

Gradient Derivation

a)

$$
\begin{aligned}
\frac{\partial}{\partial \beta_0}\left[\frac{1}{2m}\sum_{i=0}^{m}(\beta_0+\beta_1 x^{(i)}-y^{(i)})^2\right] &= \frac{\partial}{\partial \beta_0}\left[\frac{1}{2m}\sum_{i=0}^{m}(\beta_0+\beta_1 x^{(i)}-y^{(i)})^2\right] \\
&= \frac{1}{2m}\sum_{i=0}^{m}\left[2(\beta_0+\beta_1 x^{(i)}-y^{(i)})\cdot 1\right] \\
&= \frac{1}{m}\sum_{i=0}^{m}\beta_0+\beta_1\frac{1}{m}\sum_{i=0}^{m}x^{(i)}-\frac{1}{m}\sum_{i=0}^{m}y^{(i)} \\
&= \beta_0+\beta_1\bar{x}-\bar{y}
\end{aligned}
$$

b)

$$
\begin{aligned}
\frac{\partial}{\partial \beta_1}\left[\frac{1}{2m}\sum_{i=0}^{m}(\beta_0+\beta_1 x^{(i)}-y^{(i)})^2\right] &= \frac{1}{2m}\sum_{i=0}^{m}\frac{\partial}{\partial \beta_1}\left[(\beta_0+\beta_1 x^{(i)}-y^{(i)})^2\right] \\
&= \frac{1}{2m}\sum_{i=0}^{m}\left[2x^{(i)}(\beta_0+\beta_1 x^{(i)}-y^{(i)})\right] \\
&= \frac{1}{m}\sum_{i=0}^{m}\left[\beta_0 x^{(i)}+\beta_1 {x^{(i)}}^2-x^{(i)}y^{(i)}\right] \\
&= \beta_0\frac{1}{m}\sum_{i=0}^{m}x^{(i)}+\beta_1\frac{1}{m}\sum_{i=0}^{m}{x^{(i)}}^2-\frac{1}{m}\sum_{i=0}^{m}x^{(i)}y^{(i)} \\
&= \beta_0\bar{x}+\beta_1\bar{x^2}-\bar{xy}
\end{aligned}
$$

Linear Regression by Gradient Decent

```r
# Generates linear data with normal residuals
set.seed(123)
x <- rnorm(n = 30)

epsilon <- rnorm(n = 30)

y <- 5*x + 1 + epsilon
```

```r
summary(lm(y ~ x))
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6085 -0.5056 -0.2152  0.6932  2.0118
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.1720     0.1534   7.639 2.54e-08 ***
## x             4.8660     0.1589  30.629  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8393 on 28 degrees of freedom
## Multiple R-squared:  0.971,  Adjusted R-squared:   0.97
## F-statistic: 938.1 on 1 and 28 DF,  p-value: < 2.2e-16
```

```r
# Calculates the mean squared error for a simple linear regression model.
# @param x - a vector of the explainatory variable.
# @param y - a vector of the response variable.
# @param beta_0 - the intercept value for the current SLR model.
# @param beta_1 - the slope value for the current SLR model.
# @return - sum total mean squared error (y_hat - y)^2
slr_mse <- function(x, y, beta_0, beta_1){
    cost <- ((beta_1 * x + beta_0) - y)^2
    return(sum(cost))
}
```

```r
# Calculates the slope and intercept values for SLR
# or simple linear regression.
# @param x - a vector of the explanatory variable.
# @param y - a vector of the response variable.
# @param alpha - the learning rate.
# @return betas - a vector containing the calculated betas.
slr_gradient_desc <- function(x, y, alpha){

    # Summary statistic calculations.
    # Helps to calculate the gradient faster.
    x_bar <- mean(x)
    y_bar <- mean(y)
    xy_bar <- mean(x*y)
    x_sqbar <- mean(x^2)

    # initial guess for beta_0 and beta_1.
    beta_0 <- y_bar
    beta_1 <- 0

    # A counter to determine is the error is unchanging.
    # This is the Loop-Control-Variable (LCV).
    count_same <- 0

    # Iterate 100 times or until the cost remains unchanged for 10 iterations.
```

```r
    for(i in 1:1000){

        # Stop the loop if the LCV >= 10.
        if(count_same >= 10){
            break
        }

        # Cost prior to beta adjustment.
        cost_start <- slr_mse(x, y, beta_0, beta_1)

        # Calculate gradient values.
        g_0 <- beta_0 + (beta_1 * x_bar) - y_bar
        g_1 <- (beta_0 * x_bar) + (beta_1 * x_sqbar) - xy_bar

        # Update betas.
        beta_0 <- beta_0 - (alpha * g_0)
        beta_1 <- beta_1 - (alpha * g_1)

        # If the cost is unchanged add 1 to the LCV.
        if(cost_start == slr_mse(x, y, beta_0, beta_1)){
            count_same <- count_same + 1
        }
    }

    return(c(beta_0 = round(beta_0, 4),
             beta_1 = round(beta_1, 4),
             iterations = i))
}
```

```r
slr_gradient_desc(x, y, alpha = 0.1)
```

```
##     beta_0     beta_1 iterations
##      1.172      4.866    256.000
```

Linear Model on Airbnb Data

Introduction

Airbnb, Inc. is a software company that operates a marketplace for people to rent their properties on an ad hoc basis (Wikipedia). The specific dataset addressed in this report comes from Inside Airbnb, "a mission driven project that provides data ... about Airbnb's impact on residential communities." (Website). The data set contains 12 predictive variables: 4 categorical and 8 numeric. All of the roughly 40,000 listings in this particular dataset were located in New York City (Tay, 2022).

The ultimate goal of this analysis was to develope the most predictive linear model of a particular listings price.

Methodology

The data was initial split into training and testing data (70% and 30% of the raw data respectively). The Training data was analyzed and cleaned separately to avoid data leakage that might bias the test error rate. The ID variables were removed as predictor and the factor variables were check for empty strings. All empty strings in factor variables were recoded as `NA` as were any `0` values in `price`, `minimum_nights`, `longitude`, and `latitude`. `NA` values for the review type variables were all associated with properties that had never been reviewed, so `NA` values in `reviews_per_month` were assigned 0. Furthermore, the year was extracted from `last_review` and a level of `none` was added for any empty cells. This was put into a new feature and called `last_review_year`.

Linear models are very sensitive to non-normality in the response variable. While there are no explicit parametric assumptions about the distributions of the predictor, skewness can increase model error.

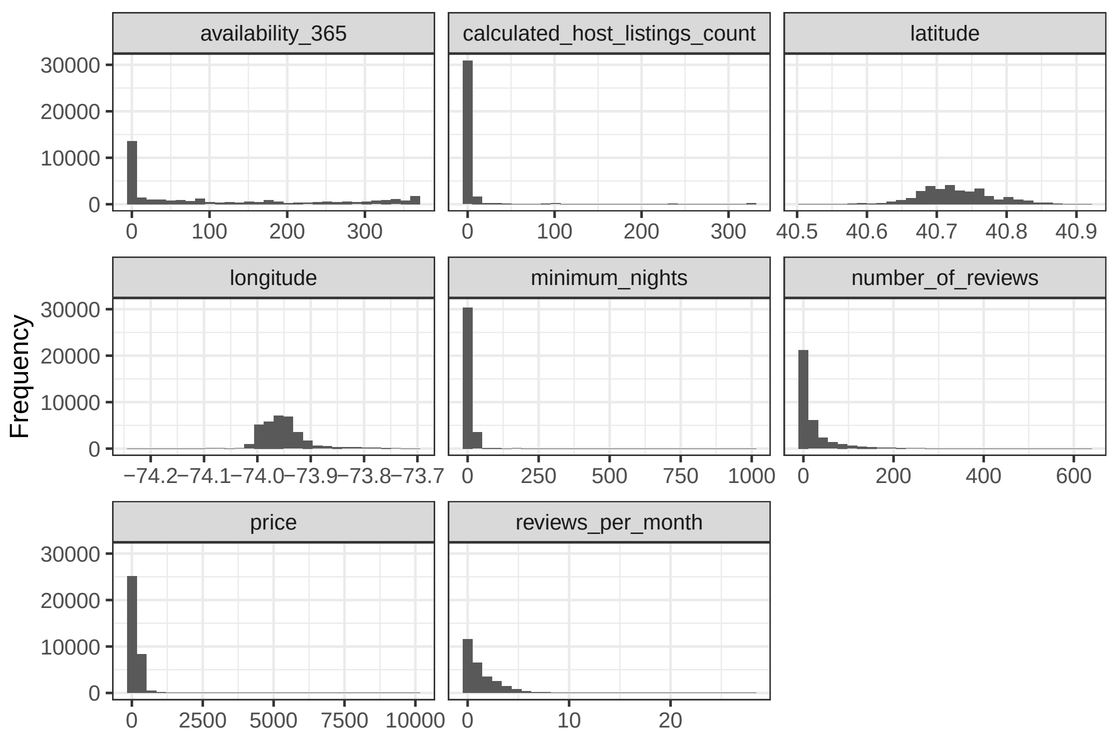


Figure 1: Histogram of raw numeric features before any transformations. Most distribution are clearly right skewed

Therefore, the response variable, `price`, was log transformed, and all other numeric predictors were transformed via the Yeos-Johnson transformation. Furthermore all numeric predictor were centered and scaled. Centering can address issues related to multicollinearity and scaling is a requirement of most imputation algorithms. All nominal predictor were transformed into $n-1$ dummy variables and 11 missing price values were imputed using k-nearest-neighbors with $k = 10$. Afterwards, near-zero-variance and highly correlated predictors were dropped from the pool. Given the large sample size this step was not strictly necessary; however, it can address overfitting and multicollinearity.

Another important assumption of linear regression is a linear relationship between all predictors and y.

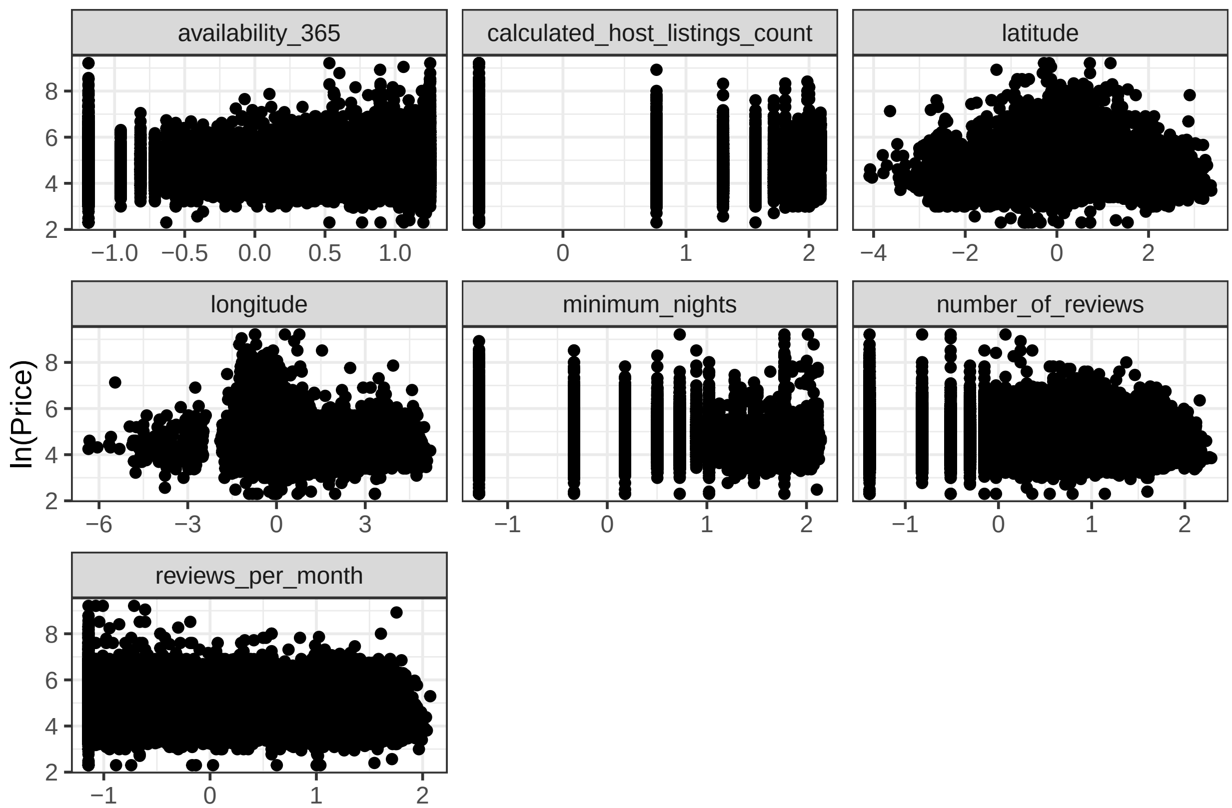


Figure 2: Scatter plot of transformed numeric predictors against log(price). Note that the x-axis is center and scaled relative to the predictor.

None of the predictors share a strong linear relationship with price; however, there are methods for including nonlinear relationship into a linear regression model. Frank Harrell in his book *Regression Modeling Strategies*, recommends treating all numeric predictors are natural cubic splines to capture non-linear relationships. Natural cubic splines are piecewise third degree piecewise polynomial transformations of predictors with added smoothness restrictions. Than can be past to the linear model via 3* basis functions per variable (marked by _ns#). * Not all variables produced 3 uncorrelated basis functions. High correlated predictors were dropped again after the spline transformation which left some predictors defined by 2 basis functions. The non-linear effect is still captured.

Results

The final model had 32 βs with almost all of them being statistically significant.

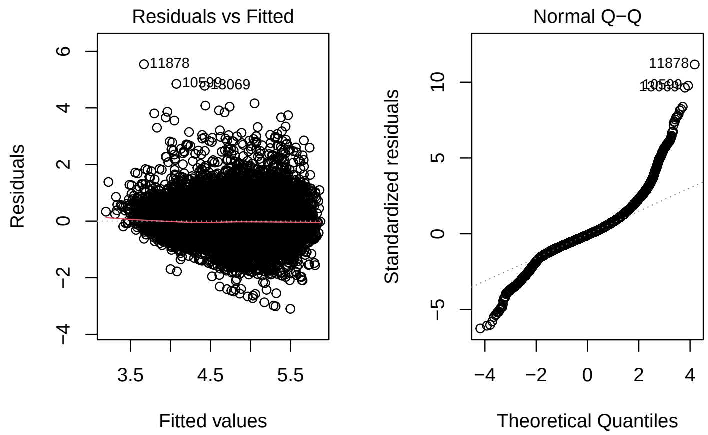


Figure 3: Diagnostic plots from the fit multiple linear regression model. The assumptions of homoscedasticity and normality are clearly violated

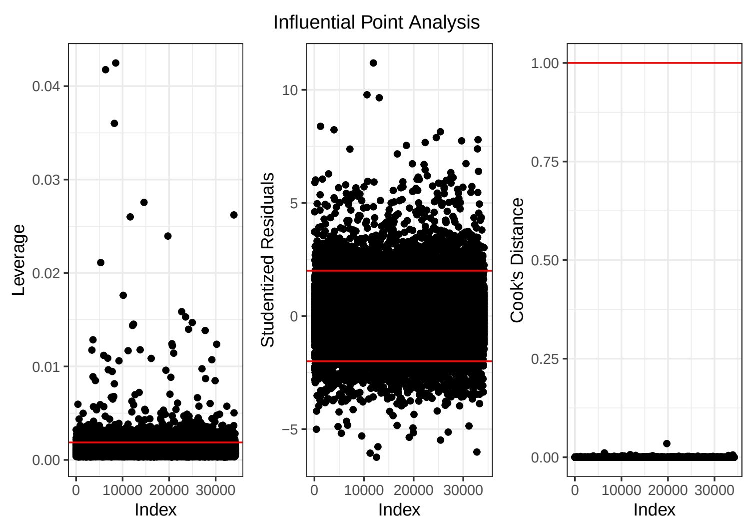


Figure 4: Analysis of potential outliers from the MLR model fit. Although there are some high leverage points, none of them are extremely influential in the model base on Cook's distance. Removing points is unlikely to improve the model fit.

Unfortunately, because the model assumption are not met, no statistical inference can be draw from the model; however, the prediction may still be accurate.

Conclusions

Table 1: Training and Testing Model Fit Metrics

	Training Data	Testing Data
Adjusted R^2	0.490	0.494
Ratio - Mean	1.000	0.999
Ratio - Standard Deviation	0.104	0.104

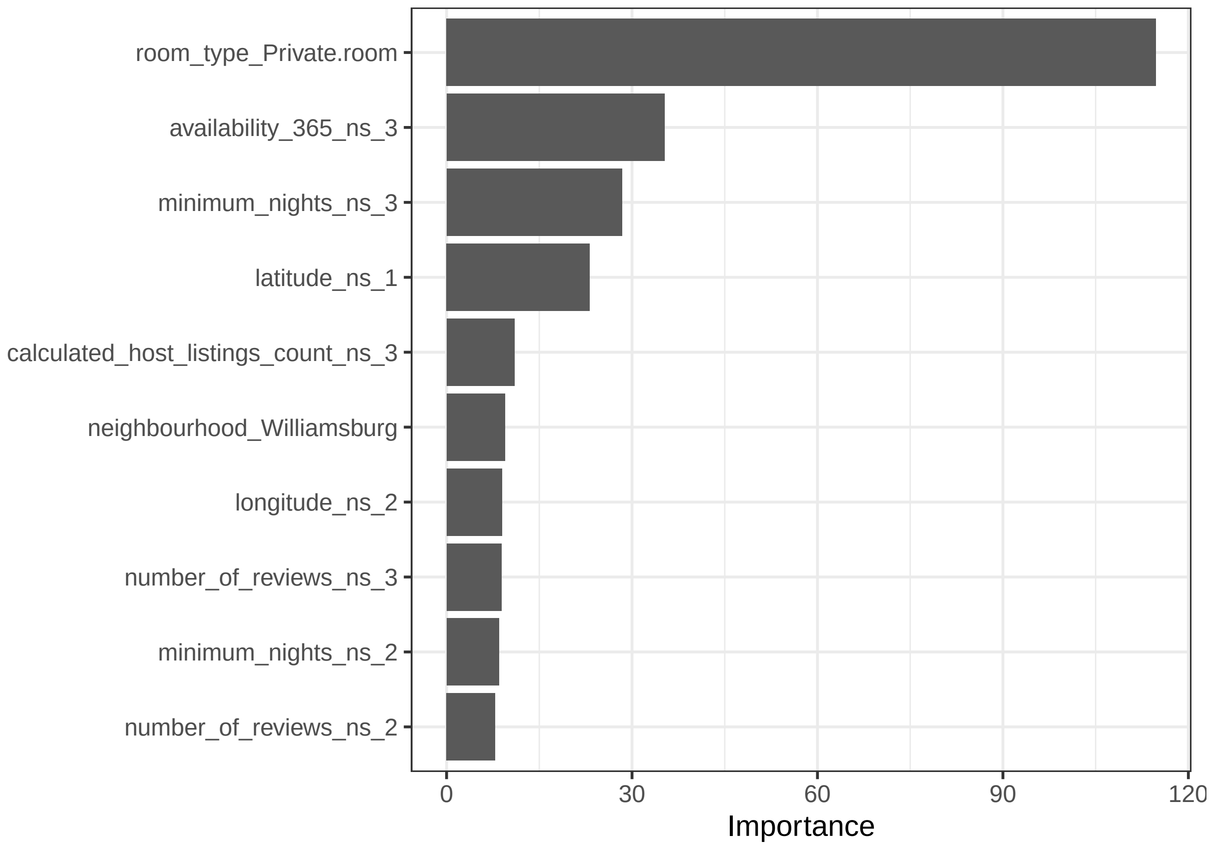


Figure 5: Top predictor variables by importance.

Based on adjusted R^2, the linear model explains roughly 50% of the variation in price for both the training and testing data. There is no significant drop in model metrics on the test data, which indicates that the model is not over fit. The ratio metrics indicate that the majority of predictions are within $\pm 10\%$ of the true price with almost all predictions within $\pm 20\%$. Given the extreme non-linearity of the data and the unequal variance of the residuals, the model prediction are usable. Additional predictive power may be achieved by transitioning to a tree based model; however, the data appears not to explain the vast majority of the variation in prices across listings.

Code for Linear Model

```
# Libraries
library(tidyverse)
library(tidymodels)
tidymodels_prefer()
```

Modeling Building

```
# Import raw data.
data_raw <- read.csv(file = "AB_NYC_2019.csv", stringsAsFactors = T,
                     header = T)
```

```
# 70/30 train/test data split
# Data was split prior to cleaning to prevent data leakage.
set.seed(123)
data_split <- initial_split(data_raw, prop = 0.7)

train_data <- training(data_split)
test_data <- testing(data_split)
```

```
# Define a cleaning recipe from the data.
# Assigns price the role of response.
# Recipe is used to create a pipline that can works with new raw data
# assuming that the format matches the training data.
cleaning_recipe <- recipe(price ~ ., data = train_data)
```

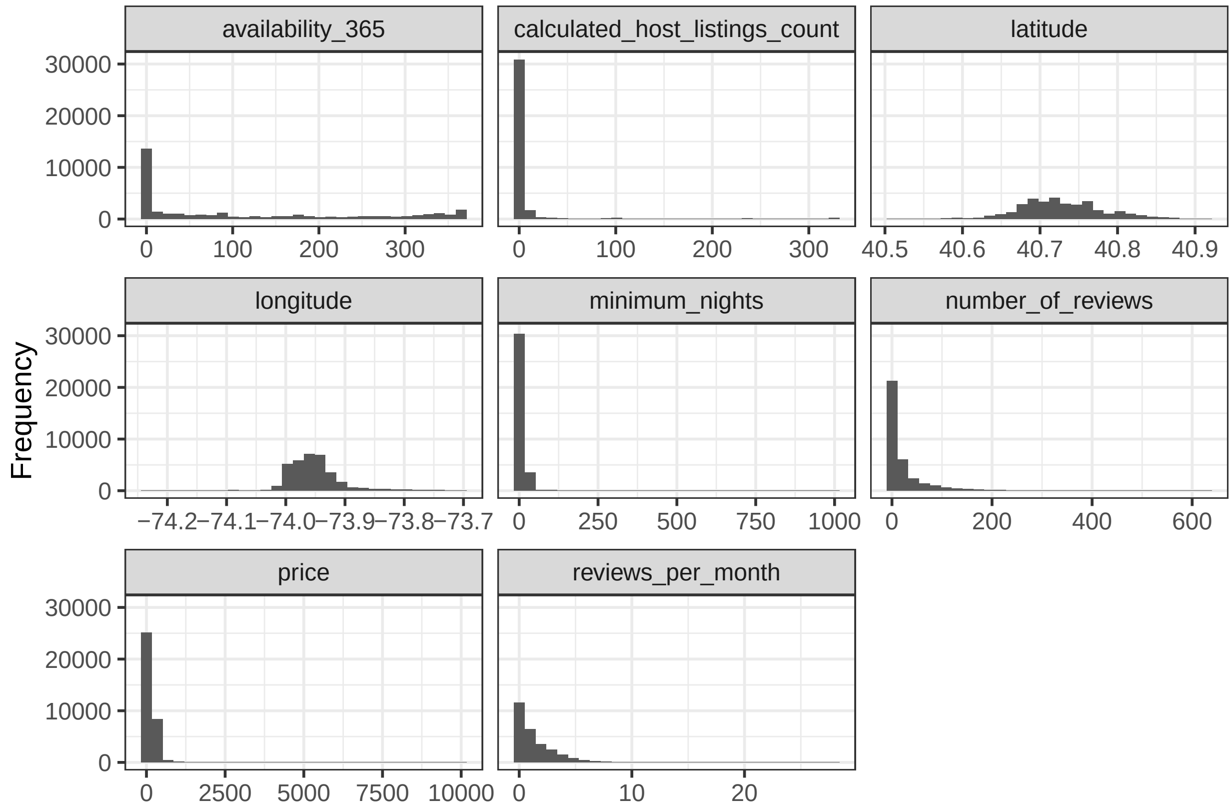
```
cleaning_recipe <- cleaning_recipe %>% step_rm(id, host_id, name, host_name)
```

```
cleaning_recipe <- cleaning_recipe %>%
    step_mutate(
        # numeric recodes
        latitude = ifelse(latitude == 0, NA, latitude),
        longitude = ifelse(longitude == 0, NA, longitude),
        minimum_nights = ifelse(minimum_nights == 0, NA, minimum_nights),

        #Factor recodes
        # It was done using replace to maintain factor coding
        room_type = replace(room_type,
            room_type == "" | room_type == " ", NA),
        neighbourhood_group = replace(neighbourhood_group,
            neighbourhood_group == "" | neighbourhood_group == " ",  NA),
        neighbourhood = replace(neighbourhood,
            neighbourhood == "" | neighbourhood == " ", NA),
        last_review = replace(last_review,
            last_review == "" | room_type == " ", NA)
    )
```

```r
# Skip = T tell the fit to ignore the step if it can't be done.
# Necessary because we assume the test_data does not have the response.
cleaning_recipe <- cleaning_recipe %>%
                   step_mutate(price = ifelse(price == 0, NA, price), skip = T)
```

```r
# Geodist (?)
cleaning_recipe <- cleaning_recipe %>%
    step_mutate(

            # Change NA reviews per month to be 0.
            reviews_per_month = ifelse(is.na(reviews_per_month),
                0, reviews_per_month),

            # Create a new variable last_review_year to reduce dimensionality.
            last_review_year = substring(last_review, 1, 4),

            # Modify NA last_review to be a new level "none".
            last_review_year = replace(last_review_year,
            is.na(last_review_year), "none"),

            # Cast to be factor.
            last_review_year = as.factor(last_review_year),

            # Remove old last review variable
            last_review = NULL
    )
```

```r
plot_skew <- train_data %>%
             select(where(is.numeric)) %>%
             select(-c(id, host_id)) %>%
             pivot_longer(cols = everything()) %>%
             ggplot(aes(value)) +
             geom_histogram() +
             facet_wrap(~name, scales = "free_x") +
             theme_bw() +
             labs(x = "", y = "Frequency")

plot_skew
```




```
save(plot_skew, file = "Figures/plot_skew.rds")
```

```
cleaning_recipe <- cleaning_recipe %>%
                   step_YeoJohnson(all_numeric_predictors()) %>%
                   # price was logged instead of YeoJohnson to maintain
                   # reversibility and interpretability.
                   step_log(price, skip = T)
```

```
cleaning_recipe <- cleaning_recipe %>%
                   step_normalize(all_numeric_predictors()) %>%
                   step_dummy(all_nominal_predictors()) %>%
                   step_impute_knn(everything(), neighbors = 10,
                                   impute_with = imp_vars(everything()),
                                   skip = T)
```

```
cleaning_recipe <- cleaning_recipe %>%
                   # removes predictor with less than 10% unique values and
                   # a greater than 95/5 ratio of most prevalent to next most
                   step_nzv(all_predictors()) %>%
                   step_corr(all_numeric_predictors())
```

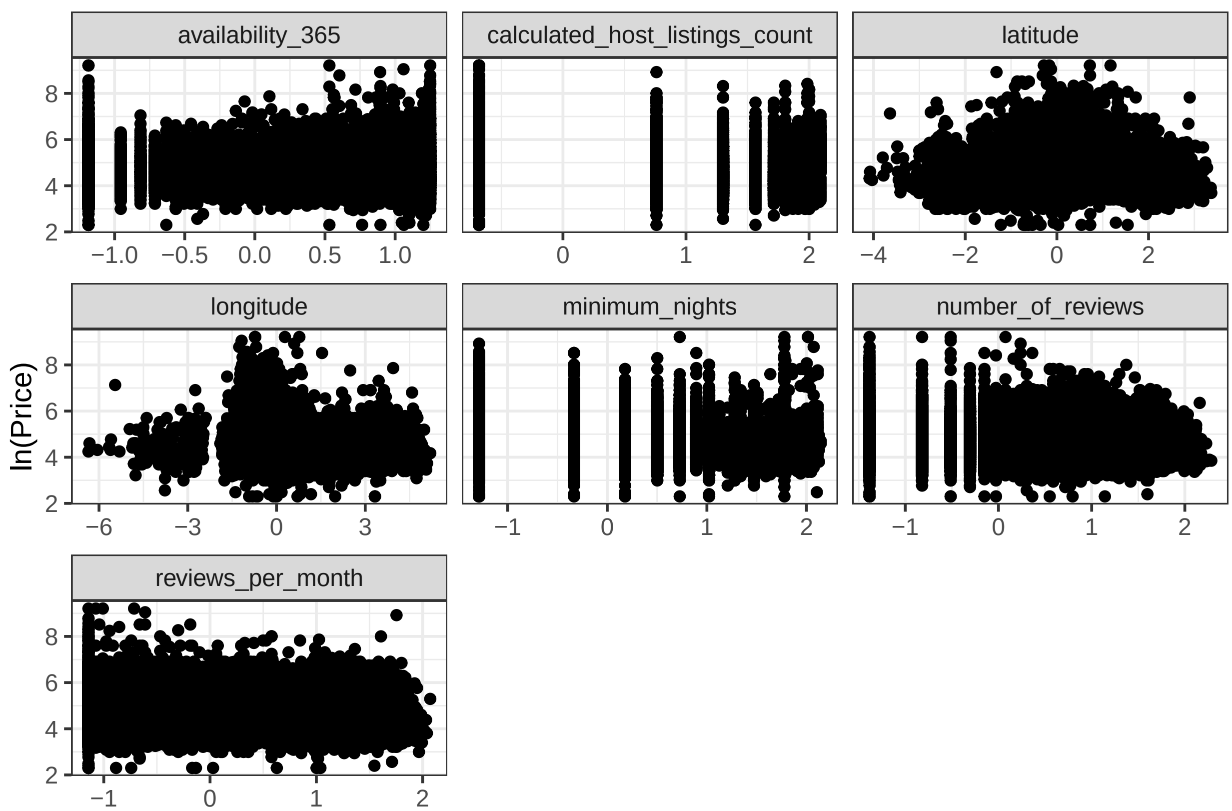
```
cleaning_recipe <- cleaning_recipe %>% prep(retain = T, verbose = T)
```

```
## oper 1 step rm [training]
```

```
## oper 2 step mutate [training]
## oper 3 step mutate [training]
## oper 4 step mutate [training]
## oper 5 step YeoJohnson [training]
## oper 6 step log [training]
## oper 7 step normalize [training]
## oper 8 step dummy [training]
## oper 9 step impute knn [training]
## oper 10 step nzv [training]
## oper 11 step corr [training]
## The retained training set is ~ 5.5 Mb  in memory.
```

```r
train_scatter <- bake(cleaning_recipe, new_data = NULL)
```

```r
# Matrix Scatterplot
plot_scatter <- train_scatter %>%
                select(-starts_with(c("neighbourhood",
                                      "room_type", "last_review"))) %>%
                pivot_longer(cols = -price) %>%
                ggplot(aes(x = value, y = price)) +
                geom_point() +
                facet_wrap(~name, scales = "free_x") +
                theme_bw() +
                labs(x = "", y = "ln(Price)")

plot_scatter
```



```
save(plot_scatter, file = "Figures/plot_scatter.rds")
```

```
cleaning_recipe <- cleaning_recipe %>%
                   # makes sure not to accidentally spline your response variable
                   step_ns(-starts_with(c("neighbourhood",
                                "room_type", "last_review", "price")),
                           deg_free = 3) %>%
                   # called again to remove overdetermined splines
                   step_corr(all_numeric_predictors())
```

```
mlr_mod <- linear_reg() %>% set_engine("lm")

mlr_wflow <- workflow() %>%
             add_model(mlr_mod) %>%
             add_recipe(cleaning_recipe)

mlr_wflow
```

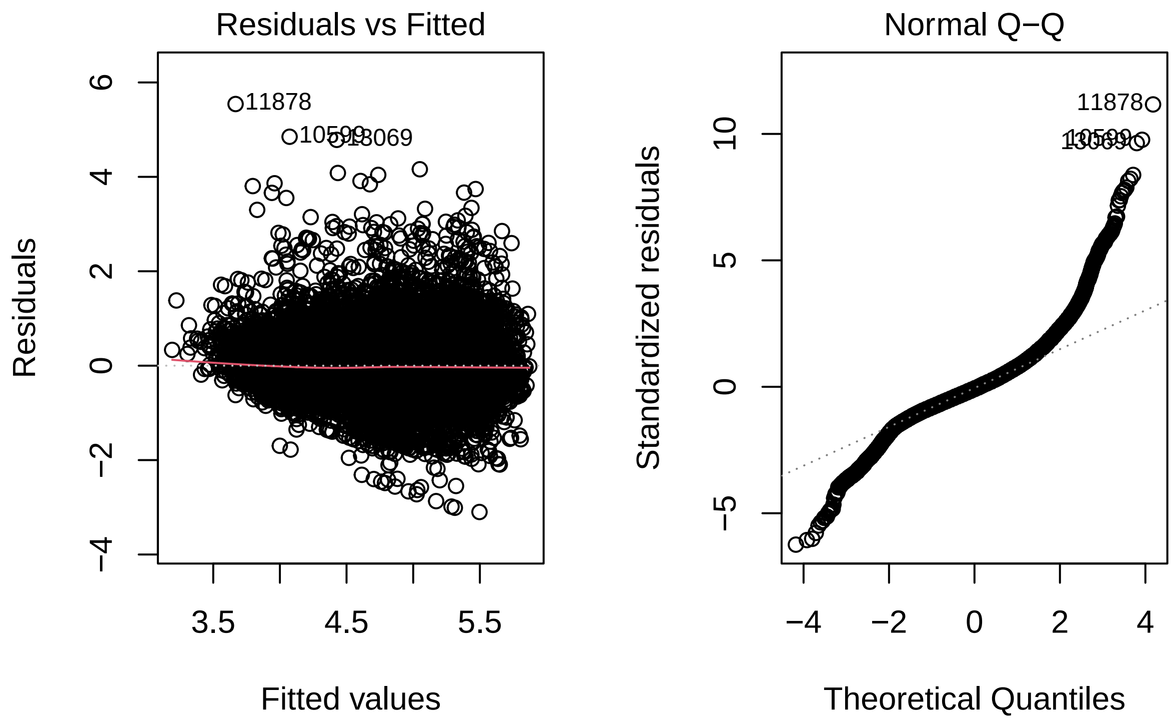
```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 13 Recipe Steps
##
## * step_rm()
## * step_mutate()
## * step_mutate()
## * step_mutate()
## * step_YeoJohnson()
## * step_log()
## * step_normalize()
## * step_dummy()
## * step_impute_knn()
## * step_nzv()
## * ...
## * and 3 more steps.
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

```
mlr_fit <- mlr_wflow %>%
           fit(data = train_data)

mlr_fit_lm <- mlr_fit %>% extract_fit_engine()

summary(mlr_fit_lm)
```

```
##
## Call:
```

```
## stats::lm(formula = ..y ~ ., data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0993 -0.2811 -0.0383  0.2333  5.5426
##
## Coefficients:
##                                      Estimate Std. Error  t value Pr(>|t|)
## (Intercept)                          3.598496   0.102484   35.113  < 2e-16 ***
## neighbourhood_group_Brooklyn         0.017221   0.025884    0.665 0.505863
## neighbourhood_group_Manhattan        0.158845   0.022715    6.993 2.74e-12 ***
## neighbourhood_group_Queens          -0.117120   0.025326   -4.624 3.77e-06 ***
## neighbourhood_Bedford.Stuyvesant    -0.036826   0.012519   -2.942 0.003267 **
## neighbourhood_Bushwick              -0.052567   0.015845   -3.318 0.000909 ***
## neighbourhood_Harlem                -0.101567   0.014018   -7.246 4.40e-13 ***
## neighbourhood_Williamsburg           0.118317   0.012558    9.422  < 2e-16 ***
## room_type_Private.room              -0.666184   0.005806 -114.750  < 2e-16 ***
## last_review_year_X2016              -0.098130   0.017791   -5.516 3.50e-08 ***
## last_review_year_X2017              -0.096721   0.018090   -5.347 9.02e-08 ***
## last_review_year_X2018              -0.095461   0.018332   -5.207 1.93e-07 ***
## last_review_year_X2019              -0.149531   0.019646   -7.611 2.78e-14 ***
## latitude_ns_1                        0.761940   0.032911   23.151  < 2e-16 ***
## latitude_ns_2                        0.818374   0.108689    7.529 5.22e-14 ***
## latitude_ns_3                       -0.257714   0.044796   -5.753 8.84e-09 ***
## longitude_ns_1                      -0.274553   0.065781   -4.174 3.00e-05 ***
## longitude_ns_2                       2.111494   0.236104    8.943  < 2e-16 ***
## longitude_ns_3                       0.343268   0.064950    5.285 1.26e-07 ***
## minimum_nights_ns_1                 -0.011806   0.013877   -0.851 0.394918
## minimum_nights_ns_2                 -0.124602   0.014705   -8.473  < 2e-16 ***
## minimum_nights_ns_3                 -0.378582   0.013341  -28.377  < 2e-16 ***
## number_of_reviews_ns_1              -0.085484   0.018736   -4.563 5.07e-06 ***
## number_of_reviews_ns_2              -0.313913   0.039953   -7.857 4.05e-15 ***
## number_of_reviews_ns_3              -0.194295   0.021876   -8.882  < 2e-16 ***
## reviews_per_month_ns_1               0.053901   0.020394    2.643 0.008221 **
## reviews_per_month_ns_2               0.156019   0.049282    3.166 0.001548 **
## reviews_per_month_ns_3              -0.024813   0.020468   -1.212 0.225410
## calculated_host_listings_count_ns_1  0.032294   0.010035    3.218 0.001291 **
## calculated_host_listings_count_ns_3 -0.125242   0.011395  -10.991  < 2e-16 ***
## availability_365_ns_1                0.051268   0.010436    4.913 9.03e-07 ***
## availability_365_ns_3                0.309121   0.008771   35.244  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4968 on 34194 degrees of freedom
## Multiple R-squared:  0.4903, Adjusted R-squared:  0.4898
## F-statistic:  1061 on 31 and 34194 DF,  p-value: < 2.2e-16
```

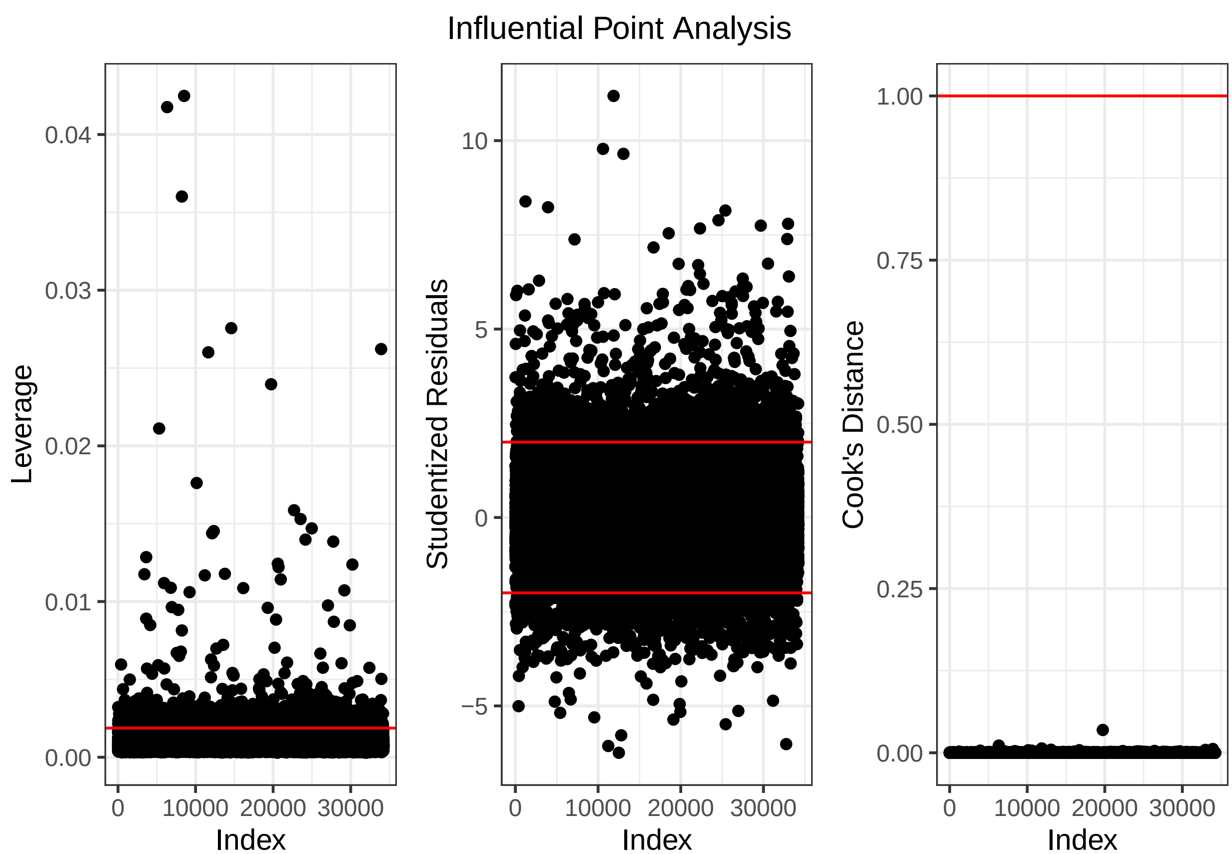
```r
par(mfrow = c(1,2))
plot(mlr_fit_lm, which = 1:2)
```




```r
save(mlr_fit_lm, file = "Figures/mlr_fit_lm.rds")
```

```r
library(MASS)
library(ggpubr)

##Influential Point Analysis ##

##Calculate Leverage, Studentized Residuals, and Cook's Distance.
Id <- 1:length(train_data$id)
Leverage <- hatvalues(mlr_fit_lm)
StudRes <- studres(mlr_fit_lm)
CookD <- cooks.distance(mlr_fit_lm)

inful_data <- cbind(Id, Leverage, StudRes, CookD)
inful_data <- as.data.frame(inful_data)

##Plots
##Leverage
lev <- ggplot(data = inful_data, aes(x = Id, y = Leverage)) + geom_point() +
            geom_hline(yintercept = 2 * length(mlr_fit_lm$coefficients) /
                length(inful_data$Id), col = "red") +
            labs(x = "Index") +
            theme_bw()

##Studentized Residuals
```

```r
studres <- ggplot(data = inful_data, aes(x = Id, y = StudRes)) + geom_point() +
                geom_hline(yintercept = 2, col = "red") +
                geom_hline(yintercept = -2, col = "red") +
                labs(y = "Studentized Residuals", x = "Index") +
                theme_bw()

##Cooks distance
cooks <- ggplot(data = inful_data, aes(x = Id, y = CookD)) + geom_point() +
             geom_hline(yintercept = 1, col = "red") +
             labs(y = "Cook's Distance", x = "Index") +
             theme_bw()

inful <- ggarrange(lev, studres, cooks, ncol = 3, nrow = 1)

inful <- annotate_figure(inful, top = text_grob("Influential Point Analysis"))

inful
```

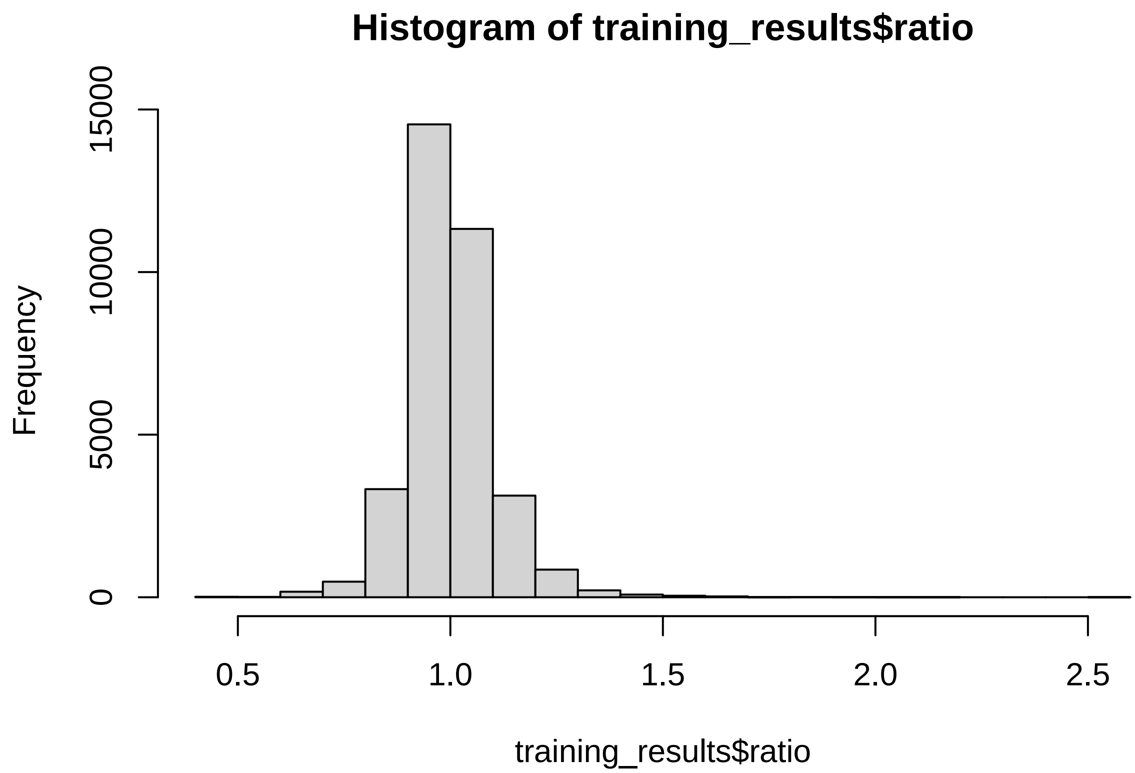



```r
save(inful, file = "Figures/inful.rds")
```

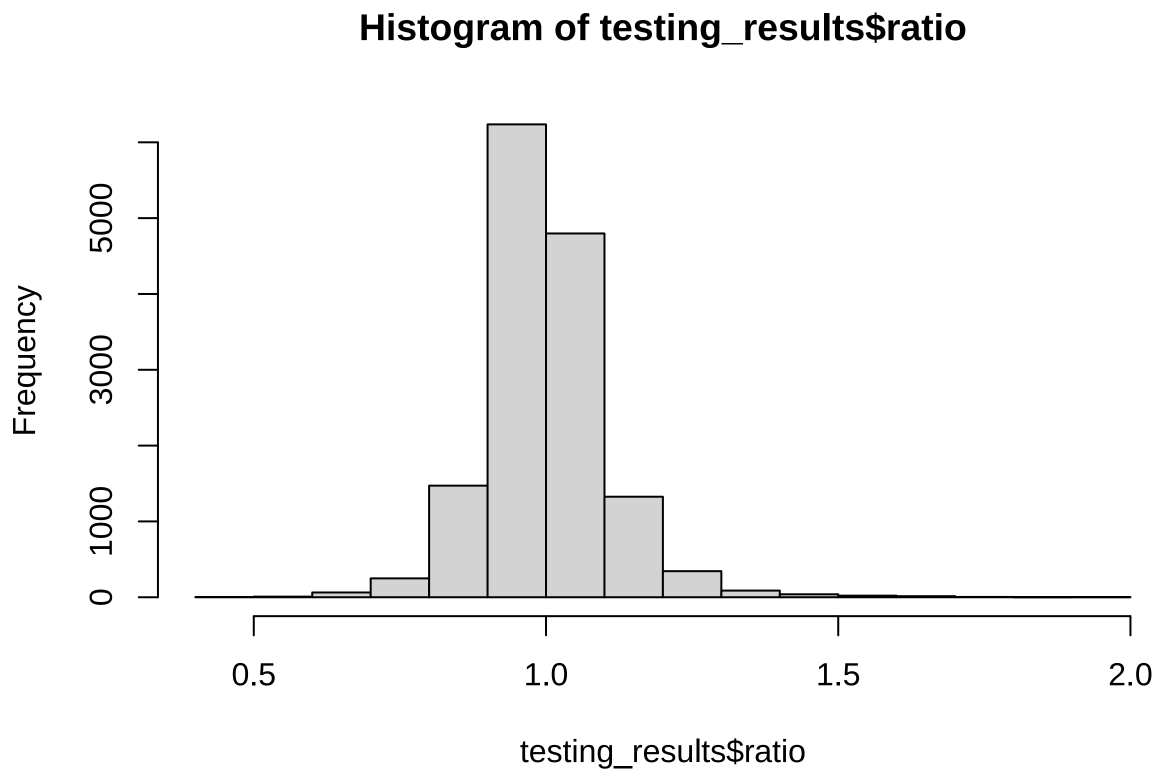
```r
# Group prediction results raw v. training error v. testing error.
# metric r = Predicted / Actually
# Looking for the mean closest to 1 with the smallest spread.

# Extract transformed pricing variables
```

```r
train_baked <- bake(prep(cleaning_recipe), new_data = NULL)

# Fresh = T tell the recipe to apply all the steps again
# training = test_data tell the recipe to treat the test_data
# as if it was the training data used in the preprocessing step
# this avoids the modeling problem of steps having to be skipped
# when the response is absent from the dataset.
test_baked <- bake(
    prep(cleaning_recipe, fresh = T, training = test_data, retain = T),
    new_data = NULL)

train_price <- train_baked %>% select(price)
test_price <- test_baked %>% select(price)
train_pred <- predict(mlr_fit, new_data = train_data)
test_pred <- predict(mlr_fit, new_data = test_data)

training_results <- cbind(train_price, train_pred)
testing_results <- cbind(test_price, test_pred)

training_results <- training_results %>%
                    mutate(ratio = price / .pred)

testing_results <- testing_results %>%
                    mutate(ratio = price / .pred)
```

```r
train_rss <- sum((training_results$price - training_results$.pred)^2)
test_rss <- sum((testing_results$price - testing_results$.pred)^2)

train_n <- nrow(training_results)
test_n <- nrow(testing_results)
model_p <- length(coef(mlr_fit_lm)) - 1 #subtract 1 to ignore the intercept

train_tss <- var(training_results$price) * train_n
test_tss <- var(testing_results$price) * test_n

train_rsq <- 1 - (train_rss / train_tss)
test_rsq <- 1 - (test_rss / test_tss)

train_adj_rsq <- 1 - (1 - train_rsq) * (train_n - 1) /
                 (train_n - model_p - 1)
test_adj_rsq <- 1 - (1 - test_rsq) * (test_n - 1) /
                 (test_n - model_p - 1)

hist(training_results$ratio)
```



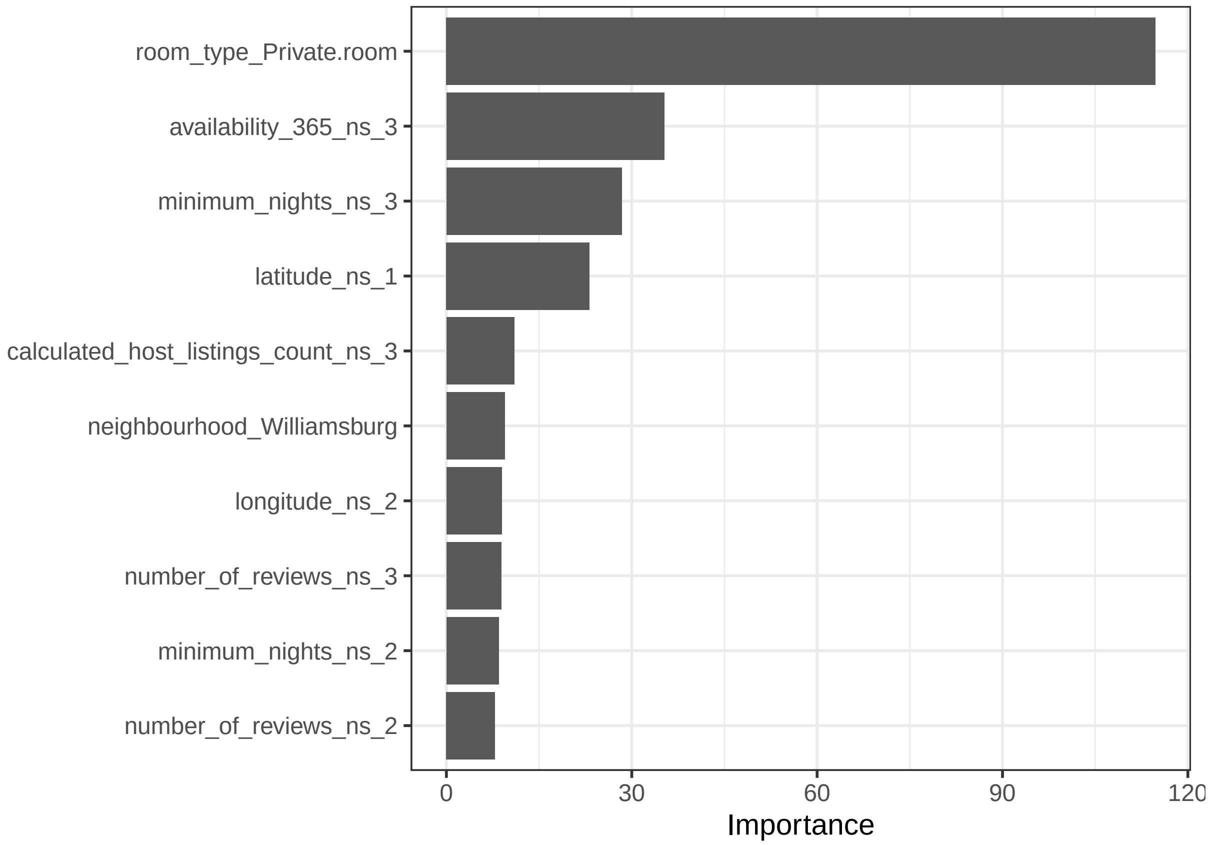

```
hist(testing_results$ratio)
```




```
train_ratio_mean <- mean(training_results$ratio)
train_ratio_sd <- sd(training_results$ratio)
test_ratio_mean <- mean(testing_results$ratio)
test_ratio_sd <- sd(testing_results$ratio)
```

```
#model metric table
library(kableExtra)
metric_table <- data.frame(train = c(train_adj_rsq, train_ratio_mean,
                                     train_ratio_sd),
                           test = c(test_adj_rsq, test_ratio_mean,
                                    test_ratio_sd))

colnames(metric_table) <- c("Training Data", "Testing Data")
rownames(metric_table) <- c("Adjusted $R^2$", "Ratio - Mean",
                            "Ratio - Standard Deviation")

caption <- "\\textbf{Table 1: }Training and Testing Model Fit Metrics"

metric_table <- metric_table %>% kbl(format = "latex", booktabs = T,
    longtable = T, escape = F, digits = 3, caption = caption) %>%
    kable_styling(full_width = F)

save(metric_table, file = "Figures/metric_table.rds")
```

```r
library(vip)
vip_plot  <- mlr_fit %>%
             extract_fit_parsnip %>%
             vip() + theme_bw()

vip_plot
```




```r
save(vip_plot, file = "Figures/vip_plot.rds")
```