EC349 Assignment

Donghua (Alex) Zheng

March 2025

Main section

This study compares econometric and machine learning (ML) models in forecasting Airbnb listing prices in London.

The Inside Airbnb dataset originally contained 95,144 observations and 75 variables. After cleaning, it had 60,358 observations and 18 variables (Table 1).

Table 1: Descriptions of variables from the cleaned dataset

Variable Name	**Description**
price	Price of Airbnb listing in USD (removed extreme outliers above $1,000)
room_type	Type of room
property_type	Type of property
accommodates	Number of people the property accommodates
bathrooms	Number of bathrooms available
bedrooms	Number of bedrooms available
beds	Number of beds available
neighbourhood_cleansed	Neighbourhood where the property is located
latitude	Latitude coordinate of the property
longitude	Longitude coordinate of the property
host_is_superhost	Whether the host is a superhost
calculated_host_listings_count	Number of listings by the host
host_has_profile_pic	Whether the host has a profile picture
host_identity_verified	Whether the host's identity is verified
number_of_reviews	Total number of reviews received by the listing
availability_30	Number of days the listing is available in the next 30 days
availability_365	Number of days the listing is available in the next 365 days
host_years_of_experience	Number of years since the host joined Airbnb (calculated from "host_since")

I first selected a subset of the dataset ("room_type"="Entire home/apt"), as the majority of the listings were for an entire home or apartment [page A-5], to investigate whether neighbourhood could affect pricing. I computed the mean price for each neighbourhood and plotted a bar chart [page A-6] in descending order by price. The results showed that neighbourhood could indeed influence prices, with Westminster, Kensington and Chelsea being the most expensive areas in London, averaging close to $300 (after removing extreme outliers), whilst Croydon and Sutton were the cheapest areas.

Next, I plotted scatter plots of selected variables against price [page A-7] and found that variables such as "host_has_profile_pic" had limited impact on prices. In addition, I created a scatter mapbox plot [page A-9] to display the exact locations of listings and their prices on a map (interactive version here), which showed that most listings were concentrated in central London. Furthermore, I generated a 3D scatter plot [page A-11] showing the number of people a listing accommodates, the number of bathrooms, and the price (interactive version here), and it showed that accommodation capacity was mostly fixed at even numbers (2–10 guests), with no clear correlation between price and capacity, and most listings had fewer than five bathrooms, though some outliers exceeded 10.

Intuitively, "room_type", "property_type", "accommodates", "bathrooms", "bedrooms", "beds" are likely to affect the size of the listing, and geographic locations can impact convenience, particularly in terms of proximity to tourist attractions as Airbnb are mainly used by travellers according to the Airbnb report. In addition, "host_is_superhost", "calculated_host_listings_count" may reflect host's level of experience and "host_has_profile_pic", "host_identity_verified" and "number_of_reviews" could indicate trustworthiness. Furthermore, availability in the next 30/365 days may signal the property's popularity. Finally, "host_years_of_experience" directly influences both the host's experience and perceived trustworthiness.

Based on analyses and intuitions explained above, any variables that could potentially impact listing prices were selected despite some with low visible correlations in the scatter plots, as summarised in Table 1.

After one-hot encoding the categorical variables and removing features with zero variance, there were 28 features in the dataset. Data was then split into training, validation, and test sets by 70%, 20%, and 10%, respectively. Given a relatively large dataset, allocating 10% to testing would be sufficient to evaluate the model's performance and thus leaving a larger portion (20%) for validation to further improve the model.

I then performed clustering only on the training set to prevent data leakage. The training set was standardised and the dimensionality was reduced using Principal Component Analysis (PCA) while retaining around 95% of the variance. The WCSS plot set the optimal number of clusters (k) at 12 using the elbow method. A scatter plot of the clusters [page A-15] showed no clear separation between clusters, except that cluster 12 contained most of the outliers. Moreover, overlaps among clusters were visible. To further investigate the clusters, box plots [page A-16] of the target variable ("price") were created for each cluster. These plots confirmed considerable overlaps in interquartile ranges across clusters and showed large numbers of outliers within each cluster.

To assess whether clustering could improve model performance, labels were assigned to the training set. The k-means model (trained solely on the training set) was then used to predict cluster labels for the validation and test sets. However, results showed that including cluster labels only marginally improved the Ridge regression whilst worsened both Random Forest and Feedforward Neural Network models regarding the validation and the test Mean Absolute Error. Therefore, cluster labels were excluded from training any subsequent models here.

In this study, Ridge regression, LASSO regression, Random Forest, and Feedforward Neural Network (FNN) models were selected. Ridge and LASSO regression models can address multicollinearity through their regularisation term (lambda) while maintaining interpretability, albeit slightly lower than that of standard Linear Regression. They also offer simplicity and computational efficiency. On the other hand, Random Forest and FNN are more complex models that can handle outliers and capture non-linear patterns effectively, thus offering more flexibility, but are more difficult to interpret. Boosting was excluded due to its higher computational cost especially for a large dataset.

Before training any models, a baseline model was created using the mean price from the training set as the prediction for both the validation and test sets. The mean price was 171.07, the validation Mean Absolute Error (MAE) was 97.23, and the test MAE was 97.93.

The Ridge regression had a validation MAE of 61.82 and a test MAE of 61.88. Optimal lambda of 8.097 was chosen through cross-validation by minimising the average Mean Squared Error (MSE). The LASSO regression with an optimal lambda of 0.069 had 62.23 for both the validation and test MAEs. Ridge and LASSO coefficients [page A-19, page A-23] showed Westminster or Kensington locations strongly influenced pricing, whilst features like listings count, number of reviews, and availability over the next 365 days had minimal influence.

To optimise the performance of Random Forest, I used Bayesian Optimisation to fine-tune mtry (number of variables tried at each split) and ntree (number of trees in the model), which was more efficient than Grid Search. That is because Bayesian Optimisation strategically explores hyperparameter ranges without exhaustively evaluating all combinations. The optimised values for mtry and ntree were 13 and 443. The Random Forest model was then trained on the training set, resulting in a validation MAE of 45.79 and a test MAE of 45.40, making it the best performing model so far.

According to the feature importance plot in page A-29, geographic locations, availability in the next 30 days, whether the property was in Westminster, host experience, and the number of bathrooms were the most important features in this model. In contrast, features such as whether the host had a profile picture or whether the property was a condo contributed little.

Finally, a three-layer FNN model was implemented. The training set was first scaled, and then the mean and standard deviation of the training set were used to scale the validation and test sets to avoid data leakage. The number of neurons in the three layers was 64, 32 and 1. Dropout rate at 0.3 was applied to each hidden layer to prevent overfitting. In addition, the Adam optimiser was used, and early stopping with a patience value of 10 epochs was included in the model. It was trained for 100 epochs with a batch size of 32. Results showed that the validation MAE was 54.24, and the test MAE was 54.62, making it the second best performing model.

In conclusion, all four models outperformed the baseline, with the ML models (Random Forest and FNN) achieving higher forecasting accuracy than econometric models with regularisation techniques (Ridge and LASSO regressions) in predicting London Airbnb listing prices. That is because Random Forest and FNN can handle outliers and capture complex nonlinear patterns in the data. The flexibility of the FNN model allowed it to outperform Ridge and LASSO regressions despite not being fine-tuned (due to limited compute). Nonetheless, Ridge and LASSO offered more interpretability in explaining the

importance of each feature. Table 2 below summarises their performances.

Table 2: Model Results

Model	**Validation MAE**	**Test MAE**
Baseline Model	97.23	97.93
Ridge Regression	61.82	61.88
Lasso Regression	62.23	62.23
Random Forest	45.79	45.40
Feedforward Neural Network (FNN)	52.94	53.62

Appendix

> American spellings were predominantly used in this appendix to maintain consistency with the coding language.

Data cleansing

Run this on terminal to create a new virtual environment 'ec349_env':

```
conda create --name ec349_env python=3.9
conda activate ec349_env
conda install tensorflow keras numpy pandas matplotlib seaborn
```

```
# Import Python libraries
library(reticulate)
use_condaenv("ec349_env", required = TRUE)
library(tensorflow)
library(keras)
library(plotly)
```

```
# Import R packages
# install.packages("pacman")
library(pacman)
pacman::p_load(pacman, dplyr, ggplot2, plotly, tidyr, readr, plotly,
               caret, glmnet, randomForest, ParBayesianOptimization,
               progressr)

# Global progress bar
handlers("txtprogressbar")
```

```r
# Import the dataset
data <- read_csv("listings.csv")
```

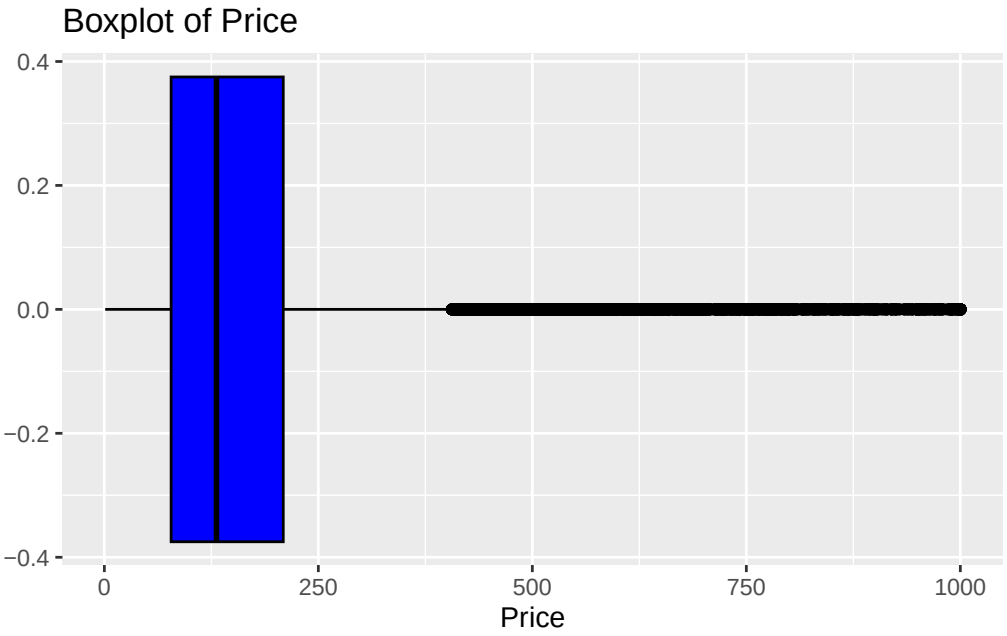
```r
# Keep relevant columns
data_cleaned <- data %>% select(c(price,

                                  room_type, property_type, accommodates,
                                  bathrooms, bedrooms, beds, amenities,

                                  neighbourhood_cleansed,
                                  latitude, longitude, review_scores_location,

                                  host_is_superhost, host_since,
                                  host_response_rate,
                                  calculated_host_listings_count,
                                  host_has_profile_pic,
                                  host_identity_verified,

                                  review_scores_rating, number_of_reviews,

                                  availability_30, availability_365))

# Replace blank values with NA
data_cleaned[data_cleaned == ""] <- NA

# Drop missing values in the 'price' column
data_cleaned <- data_cleaned %>% drop_na(price)

# # Check columns numeric
# sapply(data_cleaned, is.numeric)

# Remove '$' sign in the 'price' column
data_cleaned$price <- as.numeric(gsub("[$,]", "", data_cleaned$price))

# Replace "N/A" with NA in 'host_response_rate'
data_cleaned$host_response_rate[data_cleaned$host_response_rate == "N/A"] <- NA

# Remove the "%" symbol and convert to decimals in 'host_response_rate'
data_cleaned$host_response_rate <-
  as.numeric(gsub("%", "", data_cleaned$host_response_rate))/100

# Convert TRUE/FALSE into 1 or 0
```

```r
data_cleaned <- data_cleaned %>%
  mutate_at(vars(host_is_superhost, host_has_profile_pic,
                 host_identity_verified), as.numeric)

# Convert the 'host_since' column to date format
data_cleaned$host_since <- as.Date(data_cleaned$host_since)

# Add 'host_years_of_experience' column and drop the 'host_since' column
data_cleaned$host_years_of_experience <-
  as.numeric(difftime("2024-12-11",
                      data_cleaned$host_since,
                      units = "days")) / 365
data_cleaned <- data_cleaned %>% select(-host_since)

# Drop 'review_scores_rating', 'review_scores_location'
# and 'host_response_rate' due to large numbers of missing values
data_cleaned <- data_cleaned %>% select(-c(review_scores_rating,
                                           review_scores_location,
                                           host_response_rate))

# Drop extreme outliers
data_cleaned <- data_cleaned %>% filter(price <= 1000)

# Replace "[]" with NA in the 'amenities' column
data_cleaned$amenities[data_cleaned$amenities == "[]"] <- NA

# # Check for the number of missing values in each column
# colSums(is.na(data_cleaned))

# # Check the number of unique values in 'amenities'
# n_distinct(data_cleaned$amenities)

# Drop the 'amenities' column
data_cleaned <- data_cleaned %>% select(-amenities)

# Drop missing values
data_cleaned <- data_cleaned %>% drop_na()
```

Data analysis

Inspect the dataset further

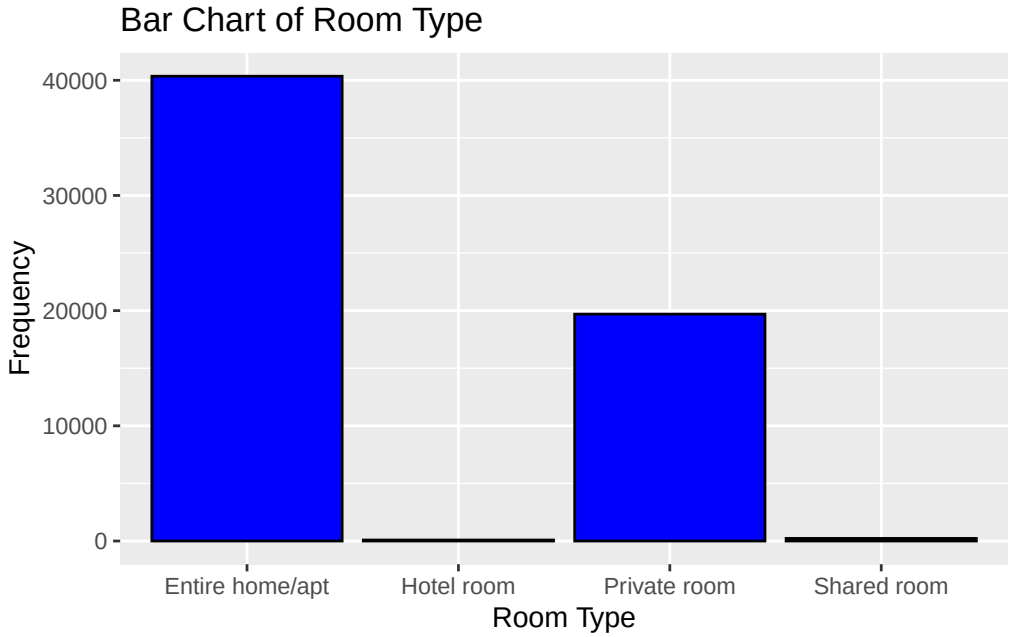
```
# Boxplot for 'price'
ggplot(data_cleaned, aes(x = price)) +
  geom_boxplot(fill = "blue", color = "black") +
  labs(title = "Boxplot of Price", x = "Price")
```




```
# Frequency of values in 'room_type'
table(data_cleaned$room_type)
```

```
Entire home/apt      Hotel room    Private room     Shared room
          40353              85           19694             226
```

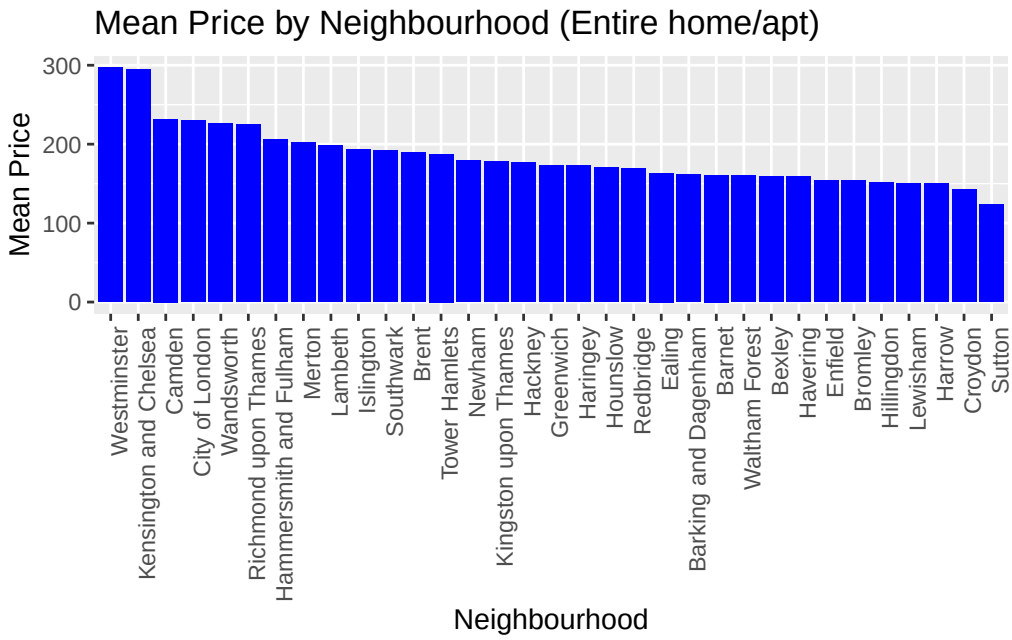
```
# Plot bar chart for 'room_type'
ggplot(data_cleaned, aes(x = room_type)) +
  geom_bar(fill = "blue", color = "black") +
  labs(title = "Bar Chart of Room Type", x = "Room Type", y = "Frequency")
```




```
# Unique values in 'property_type'
unique(data_cleaned$property_type)

# Frequency of values in 'property_type'
table(data_cleaned$property_type)

# Unique values in 'neighbourhood_cleansed'
unique(data_cleaned$neighbourhood_cleansed)

# Frequency of values in 'neighbourhood_cleansed'
table(data_cleaned$neighbourhood_cleansed)

# Check other variables for value frequencies
table(data_cleaned$host_is_superhost)
table(data_cleaned$host_has_profile_pic)
table(data_cleaned$host_identity_verified)
```

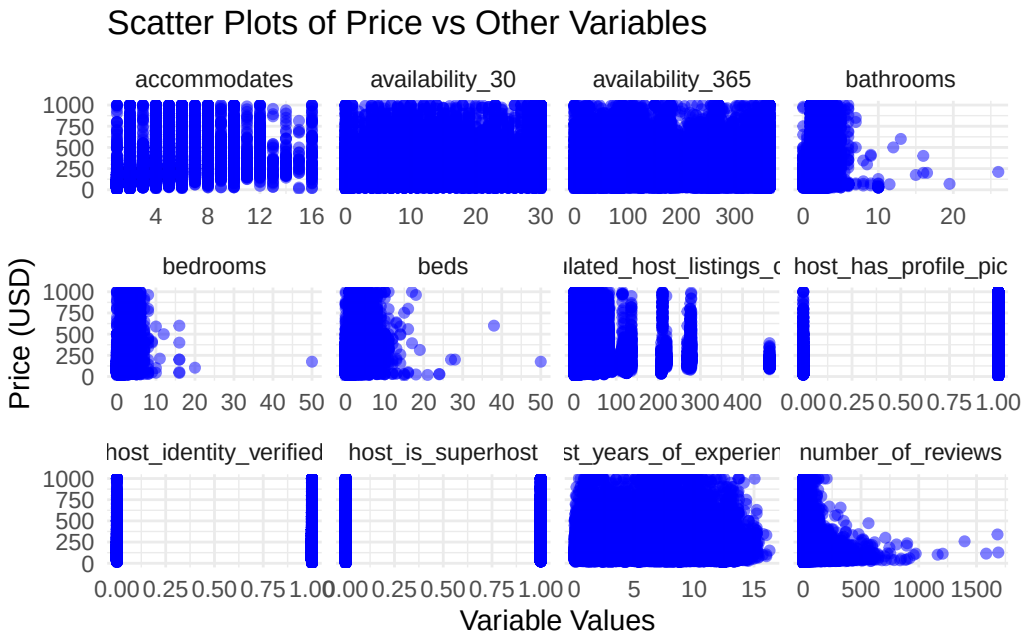
Select a subset of the dataset for further analysis ('room_type' == Entire home/apt)

```r
data_Entire_home_apt <- data_cleaned %>% filter(room_type == "Entire home/apt")

# Mean price by 'neighbourhood_cleansed' for Entire home/apt
mean_price_by_location <- data_Entire_home_apt %>%
  group_by(neighbourhood_cleansed) %>%
  summarize(mean_price = mean(price, na.rm = TRUE)) %>%
  arrange(desc(mean_price))

# Plot the mean price by neighbourhood location
ggplot(mean_price_by_location, aes(x = reorder(neighbourhood_cleansed,
                                               -mean_price),
                                   y = mean_price)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(title = "Mean Price by Neighbourhood (Entire home/apt)",
       x = "Neighbourhood",
       y = "Mean Price") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



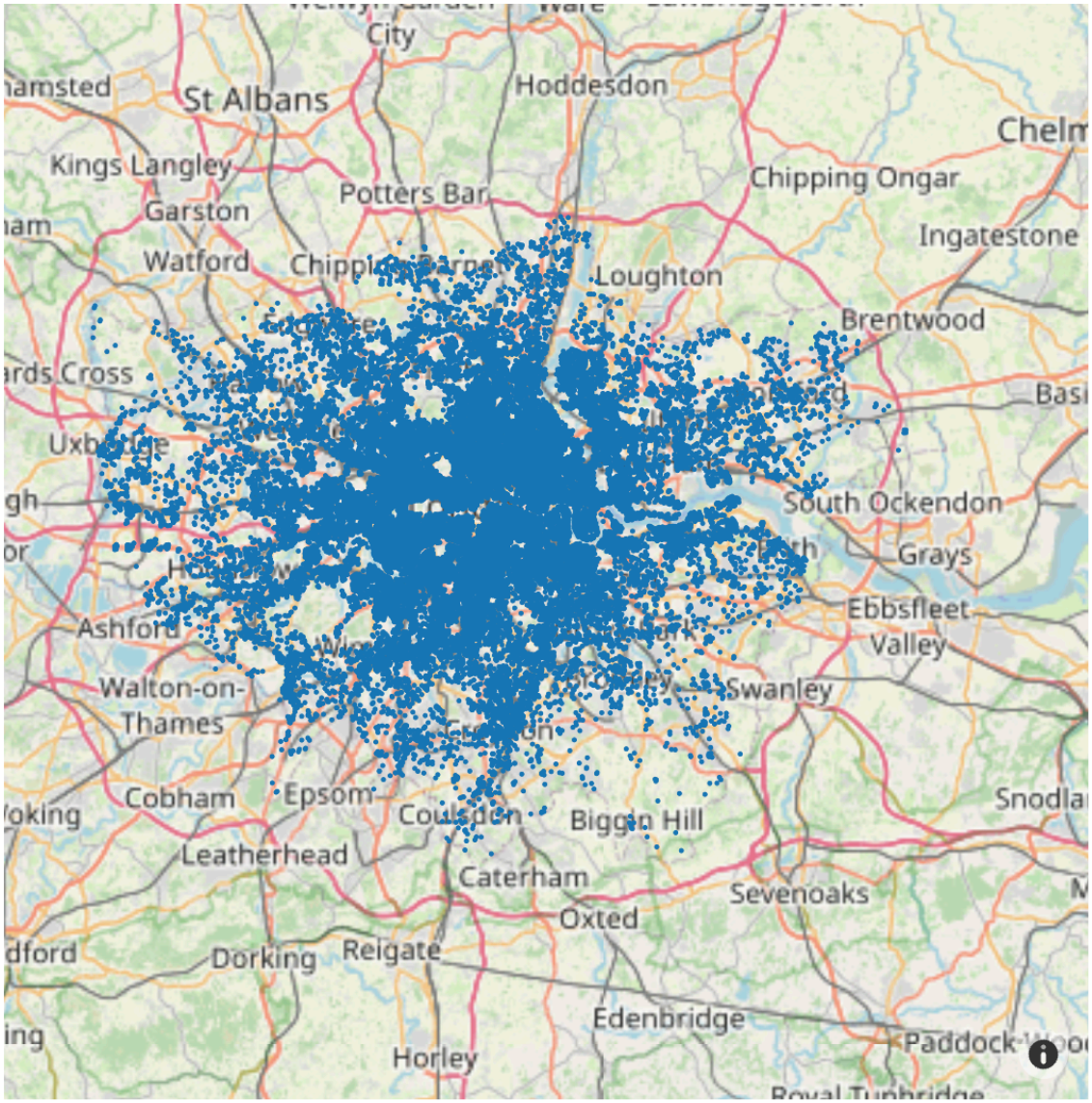
Data visualization

```
# Select relevant columns
data_subset <- data_cleaned %>%
  select(price, accommodates, bathrooms, bedrooms, beds, host_is_superhost,
         calculated_host_listings_count, host_has_profile_pic,
         host_identity_verified,
         number_of_reviews, availability_30, availability_365,
         host_years_of_experience)

# Reshape data into long format
data_subset_long <- pivot_longer(data_subset, cols = -price,
                                 names_to = "Variable", values_to = "Value")

# Create scatter plots with faceting
ggplot(data_subset_long, aes(x = Value, y = price)) +
  geom_point(alpha = 0.5, color = "blue") +
  facet_wrap(~Variable, scales = "free_x") + # Separate plots for each variable
  labs(title = "Scatter Plots of Price vs Other Variables",
       x = "Variable Values",
       y = "Price (USD)") +
  theme_minimal()
```



Scatter mapbox plot for the prices

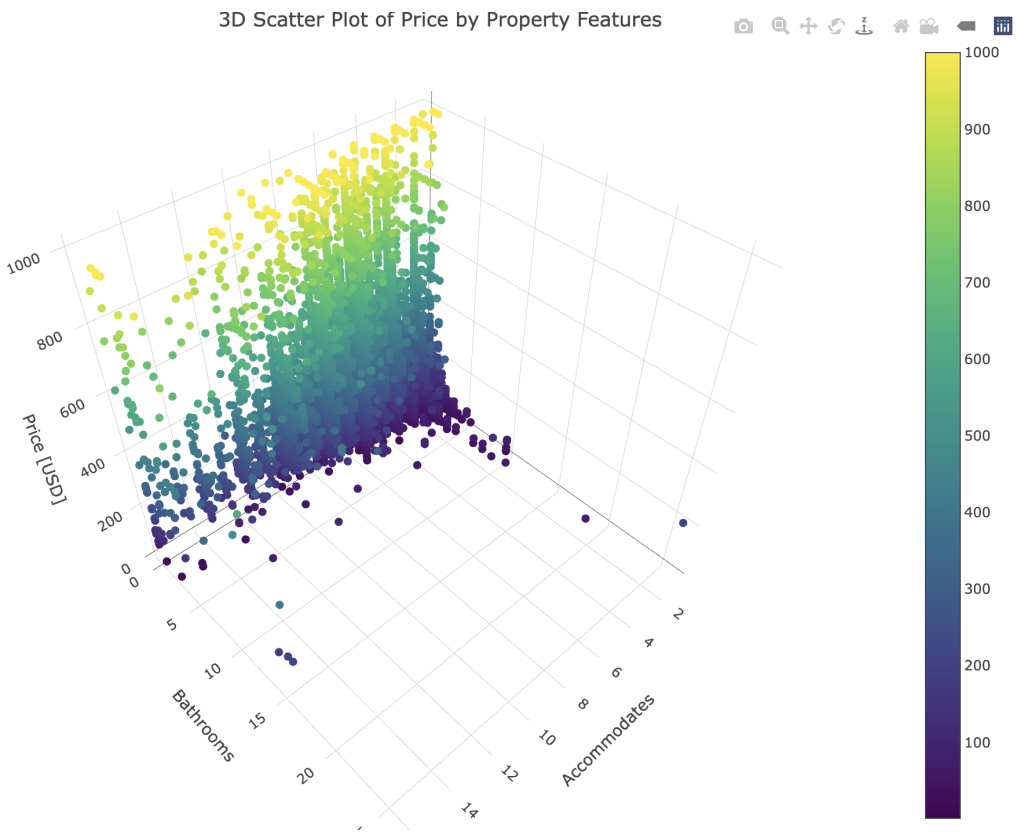
```
fig_map <- plot_ly(
  data = data_cleaned,
  lat = ~latitude,
  lon = ~longitude,
  type = "scattermapbox",
  mode = "markers",
  marker = list(size = 8),
  hoverinfo = "text",
  text = ~paste("Price: $", price),
  width = 600,
  height = 600
)

# Customize the layout with mapbox style and center
fig_map <- fig_map %>%
  layout(
    mapbox = list(
      style = "open-street-map",
      center = list(lat = 51.45, lon = 0), # Center on London
      zoom = 8 # Adjust zoom level
    )
  )
fig_map
```



3D scatter plot with accommodates, bathrooms, and price

```
fig_3d <- plot_ly(
  data = data_cleaned,
  x = ~accommodates,
  y = ~bathrooms,
  z = ~price,
  type = "scatter3d",
```

```
  mode = "markers",
  marker = list(size = 4, color = ~price, colorscale = "Viridis",
                showscale = TRUE),
  text = ~paste("Price: $", price, "<br>Accommodates: ", accommodates,
                "<br>Bathrooms: ", bathrooms),
  hoverinfo = "text"
)

# Customize layout
fig_3d <- fig_3d %>%
  layout(
    title = "3D Scatter Plot of Price by Property Features",
    scene = list(
      xaxis = list(title = "Accommodates"),
      yaxis = list(title = "Bathrooms"),
      zaxis = list(title = "Price [USD]")
    )
  )
fig_3d
```



Modeling

Prepare data

```
set.seed(42) # For reproducibility

# Define train, val, test sizes
train_size <- 0.7
val_size <- 0.2
test_size <- 0.1

n <- nrow(data_cleaned) # Total number of rows
indices <- sample(seq_len(n)) # Shuffle row indices

# Compute split indices
```

```
train_indices <- indices[1:floor(train_size * n)]
val_indices <-
  indices[(floor(train_size * n) + 1):(floor((train_size + val_size) * n))]
test_indices <- indices[(floor((train_size + val_size) * n) + 1):n]
```

OneHotEncoding & Train, Val, Test Split

```
# One-hot encoding
dummy <- dummyVars(" ~ .", data = data_cleaned, levelsOnly = FALSE)
data_encoded <- data.frame(predict(dummy, newdata = data_cleaned))

# Remove predictors with zero variance
zero_variance_cols <- nearZeroVar(data_encoded)
data_encoded <- data_encoded[, -zero_variance_cols]

# Split data into train, validation, and test sets
target <- 'price'
y <- data_encoded %>% pull(target)
X <- data_encoded %>% select(-target)

X_train <- X[train_indices, , drop = FALSE]
y_train <- y[train_indices]

X_val <- X[val_indices, , drop = FALSE]
y_val <- y[val_indices]

X_test <- X[test_indices, , drop = FALSE]
y_test <- y[test_indices]

# Print dimensions
cat("X_train shape:", dim(X_train), "\n")
```

```
X_train shape: 42250 27
```

```
cat("y_train shape:", length(y_train), "\n")
```

```
y_train shape: 42250
```

```r
cat("X_val shape:", dim(X_val), "\n")
```

```
X_val shape: 12072 27
```

```r
cat("y_val shape:", length(y_val), "\n")
```

```
y_val shape: 12072
```

```r
cat("X_test shape:", dim(X_test), "\n")
```

```
X_test shape: 6036 27
```

```r
cat("y_test shape:", length(y_test), "\n")
```

```
y_test shape: 6036
```

Cluster analysis on the training set (PCA & k-means)

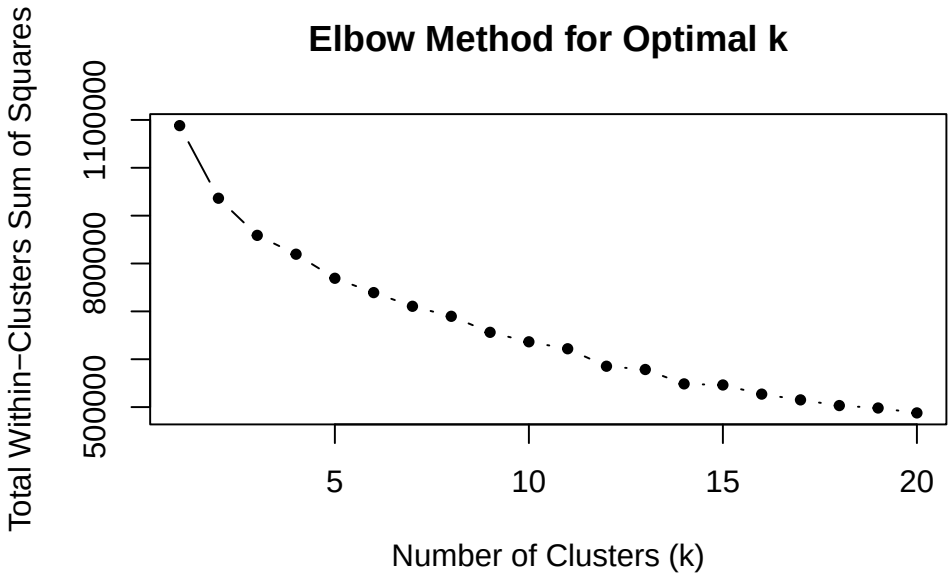
```r
# Standardization (only on training set)
X_train_scaled <- scale(X_train)

# Check dimensions
dim(X_train_scaled)
```

```
[1] 42250    27
```

```r
# Reduce dimensionality (PCA) before clustering (keep ~95% variance)
pca_result <- prcomp(X_train_scaled)
num_components <-
  which(cumsum(pca_result$sdev^2) / sum(pca_result$sdev^2) >= 0.95)[1]
X_train_pca <- pca_result$x[, 1:num_components]

# Check dimensions
dim(X_train_pca)
```

```
[1] 42250    21
```

```r
# Determine optimal number of clusters (elbow method)

# Compute Within-Cluster Sum of Squares (WCSS) for k = 1 to 20 (on training data)
wss <- numeric(20)

for (k in 1:20) {
  km_model <- kmeans(X_train_pca, centers = k,
                     nstart = 10,
                     iter.max = 300,
                     algorithm = "Lloyd")
  wss[k] <- km_model$tot.withinss
}

# Plot WCSS vs. Number of Clusters
plot(1:20, wss, type = "b", pch = 20,
     xlab = "Number of Clusters (k)",
     ylab = "Total Within-Clusters Sum of Squares",
     main = "Elbow Method for Optimal k")
```



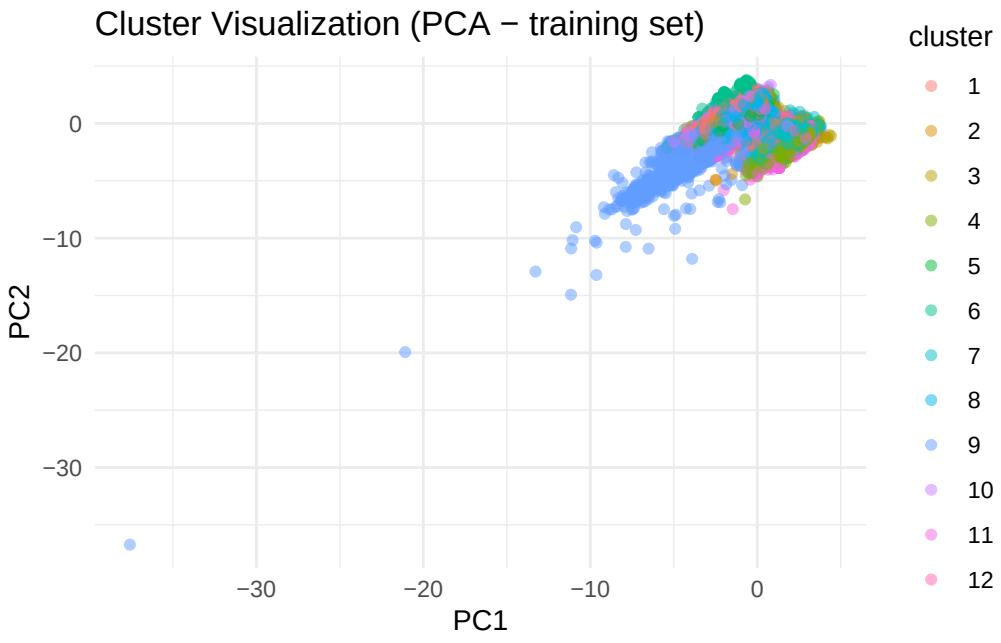

```r
# Optimal k value
k_optimal <- 12

# Run k-means on training data
set.seed(42)
km_model <- kmeans(X_train_pca, centers = k_optimal,
```

```
                    nstart = 25, iter.max = 300, algorithm = "Lloyd")

# Evaluate clusters
table(km_model$cluster)
```

```
   1    2    3    4    5    6    7    8    9   10   11   12
3379 2106  921 4483 3224  705 6386 10611 3770 2481 2229 1955
```

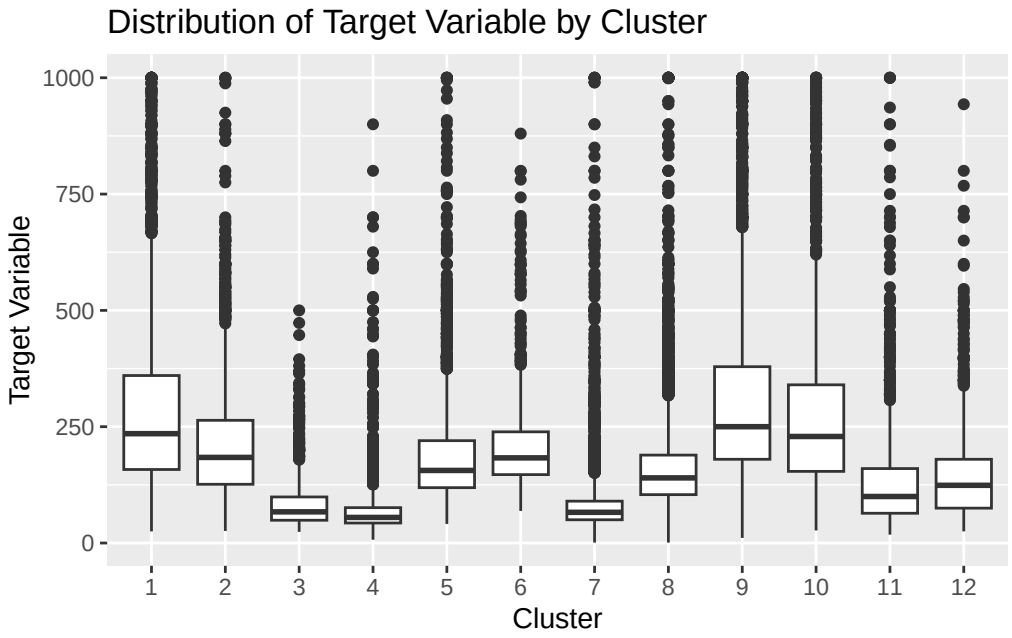
Visualize the clusters

```
# Ensure X_train_pca is a dataframe and add cluster labels
X_train_pca_df <- as.data.frame(X_train_pca)
X_train_pca_df$cluster <- factor(km_model$cluster)

# Create a scatter plot of clusters using PC1 and PC2 (training set only)
ggplot(data = X_train_pca_df, aes(x = PC1, y = PC2, color = cluster)) +
  geom_point(alpha = 0.5) +
  labs(title = "Cluster Visualization (PCA - training set)",
       x = "PC1", y = "PC2") +
  theme_minimal()
```



Visualize the distribution of the target variable across clusters

```
ggplot(data = data.frame(cluster = km_model$cluster,
                         y_train = y_train),
       aes(x = factor(cluster), y = y_train)) +
  geom_boxplot() +
  labs(title = "Distribution of Target Variable by Cluster",
       x = "Cluster",
       y = "Target Variable")
```




```
# Assign labels to the original training set
X_train$cluster <- factor(km_model$cluster)
```

Use the k-mean model to assign labels to the val and test sets

- The clusters were tested and found to lower model accuracy; therefore, they were not implemented in this project

```
# Define predict_kmeans function
predict_kmeans <- function(km_model, newdata) {
  apply(newdata, 1, function(row) {
    which.min(colSums((t(km_model$centers) - row)^2))
```

```
  })
}

# Standardization
X_val_scaled <- scale(X_val, center = attr(X_train_scaled, "scaled:center"),
scale = attr(X_train_scaled, "scaled:scale"))
X_test_scaled <- scale(X_test, center = attr(X_train_scaled, "scaled:center"),
scale = attr(X_train_scaled, "scaled:scale"))

# Apply PCA
X_val_pca <- predict(pca_result, newdata = X_val_scaled)
X_test_pca <- predict(pca_result, newdata = X_test_scaled)

# Assign cluster labels to the original val and test sets
X_val$cluster <- factor(predict_kmeans(km_model, newdata = X_val_pca))
X_test$cluster <- factor(predict_kmeans(km_model, newdata = X_test_pca))


# Convert the cluster column into one-hot encoded columns
X_train_cluster_onehot <- model.matrix(~ cluster - 1, data = X_train)
X_val_cluster_onehot <- model.matrix(~ cluster - 1, data = X_val)
X_test_cluster_onehot <- model.matrix(~ cluster - 1, data = X_test)

# Combine one-hot encoded cluster columns with the rest of the features
X_train <- cbind(X_train[, !colnames(X_train) %in% "cluster"], X_train_cluster_onehot)
X_val <- cbind(X_val[, !colnames(X_val) %in% "cluster"], X_val_cluster_onehot)
X_test <- cbind(X_test[, !colnames(X_test) %in% "cluster"], X_test_cluster_onehot)
```

Baseline model

```
# Mean of the target column
y_mean <- mean(y_train)

# Forecast validation and test sets using mean price
val_forecast_baseline <- rep(y_mean, length(y_val))
test_forecast_baseline <- rep(y_mean, length(y_test))

# Mean Absolute Error (MAE) for validation and test sets
val_mae_baseline <- mean(abs(y_val - val_forecast_baseline))
test_mae_baseline <- mean(abs(y_test - test_forecast_baseline))
```

```r
# Print results
cat("Mean price from training set:", y_mean, "\n")
```

```
Mean price from training set: 171.0696
```

```r
cat("Validation MAE (Baseline):", val_mae_baseline, "\n")
```

```
Validation MAE (Baseline): 97.2308
```

```r
cat("Test MAE (Baseline):", test_mae_baseline, "\n")
```

```
Test MAE (Baseline): 97.93063
```

Ridge regression

```r
X_train <- model.matrix(~ ., data = X_train)[, -1]
X_val <- model.matrix(~ ., data = X_val)[, -1]
X_test <- model.matrix(~ ., data = X_test)[, -1]

# Fit the model
model_ridge <- glmnet(X_train, y_train, alpha = 0)

# # Plot coefficient shrinkage across lambda values
# plot(model_ridge, xvar = "lambda")

# Cross-validation for optimal lambda
cv_ridge <- cv.glmnet(X_train, y_train, alpha = 0)

# # Plot cross-validation results
# plot(cv_ridge)

# Optimal lambda value
lambda_optimal <- cv_ridge$lambda.min
cat("Optimal lambda:", lambda_optimal, "\n")
```

```
Optimal lambda: 8.096857
```

```r
# Refit model with optimal lambda
model_ridge_optimal <- glmnet(X_train, y_train, alpha = 0,
                              lambda = lambda_optimal)

# Evaluation
val_predictions_ridge <- predict(model_ridge_optimal,
                                 s = lambda_optimal, newx = X_val)
test_predictions_ridge <- predict(model_ridge_optimal,
                                  s = lambda_optimal, newx = X_test)

# MAE for validation and test sets
val_mae_ridge <- mean(abs(y_val - val_predictions_ridge))
test_mae_ridge <- mean(abs(y_test - test_predictions_ridge))
cat("Validation MAE (Ridge Regression):", val_mae_ridge, "\n")
```

```
Validation MAE (Ridge Regression): 61.82322
```

```r
cat("Validation MAE (Ridge Regression):", test_mae_ridge, "\n")
```

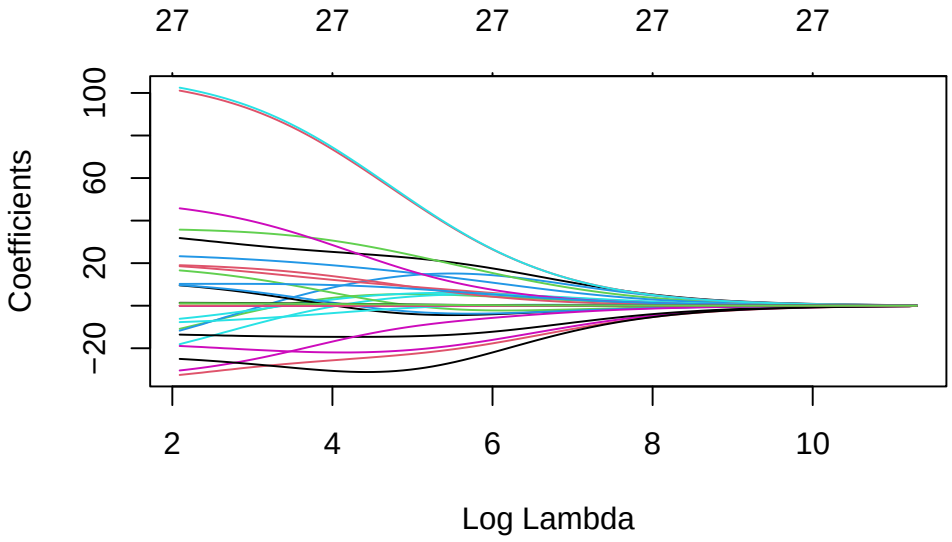
```
Validation MAE (Ridge Regression): 61.87696
```

```r
# Extract and print coefficients at the optimal lambda
ridge_coefficients <- coef(model_ridge_optimal, s = lambda_optimal)
print(ridge_coefficients)
```

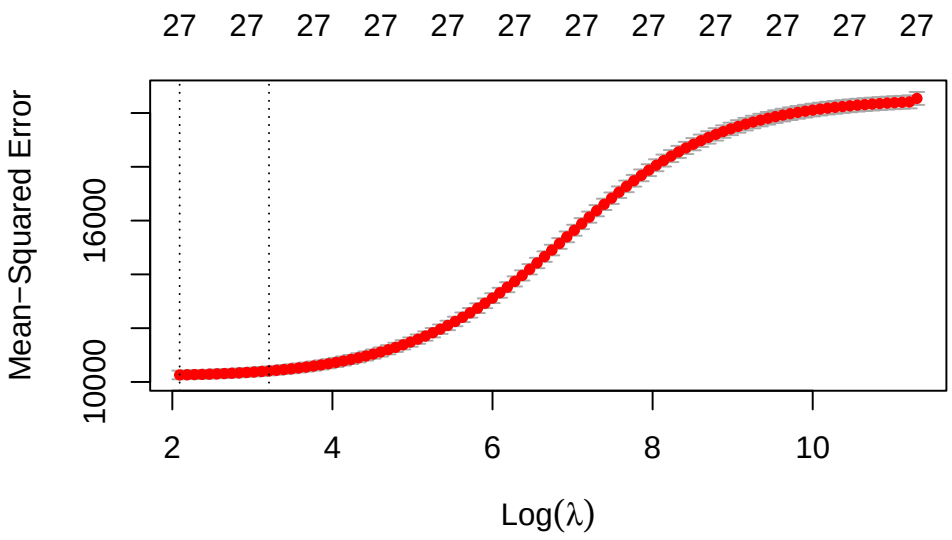
```
28 x 1 sparse Matrix of class "dgCMatrix"
                                                 s1
(Intercept)                                1541.21173120
room_typeEntire.home.apt                     31.95881140
room_typePrivate.room                       -33.00260378
property_typeEntire.condo                   -11.22378266
property_typeEntire.home                    -12.05419864
property_typeEntire.rental.unit             -18.36824804
property_typePrivate.room.in.home           -18.67532760
property_typePrivate.room.in.rental.unit    -13.33505889
accommodates                                 18.54042555
bathrooms                                    35.73566105
bedrooms                                     23.29282072
beds                                         -6.15773483
neighbourhood_cleansedCamden                 45.77611850
```

```
neighbourhood_cleansedHackney                    9.55690328
neighbourhood_cleansedKensington.and.Chelsea   101.08657408
neighbourhood_cleansedSouthwark                 16.59814374
neighbourhood_cleansedTower.Hamlets              9.67934514
neighbourhood_cleansedWestminster              102.49048741
latitude                                       -30.43962445
longitude                                      -24.99861935
host_is_superhost                               19.04118766
calculated_host_listings_count                   0.05730955
host_has_profile_pic                            10.23947474
host_identity_verified                          -7.71433595
number_of_reviews                               -0.12811219
availability_30                                  1.35976606
availability_365                                 0.02944394
host_years_of_experience                         1.17301908
```

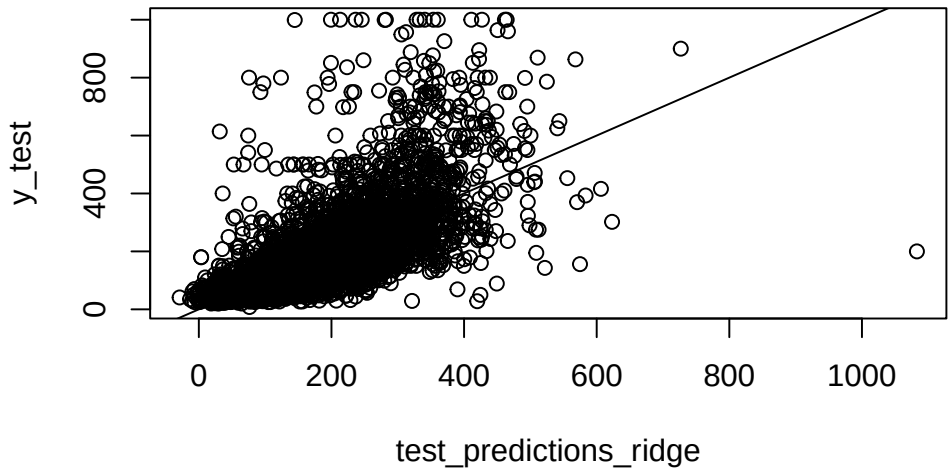
```r
# Plot coefficient shrinkage across lambda values
plot(model_ridge, xvar = "lambda")
```




```r
# Plot cross-validation results
plot(cv_ridge)
```




```
# Plot predictions vs actual values (test set)
plot(test_predictions_ridge, y_test)
abline(0, 1)
```



LASSO regression

```
# Fit the model
model_lasso <- glmnet(X_train, y_train, alpha = 1)  # Set alpha = 1 for LASSO

# # Plot coefficient shrinkage across lambda values
```

```r
# plot(model_lasso, xvar = "lambda")

# Cross-validation for optimal lambda
cv_lasso <- cv.glmnet(X_train, y_train, alpha = 1)

# # Plot cross-validation results
# plot(cv_lasso)

# Optimal lambda value
lambda_optimal <- cv_lasso$lambda.min
cat("Optimal lambda:", lambda_optimal, "\n")
```

```
Optimal lambda: 0.06880332
```

```r
# Refit model with optimal lambda
model_lasso_optimal <- glmnet(X_train, y_train, alpha = 1,
                              lambda = lambda_optimal)

# Evaluation
val_predictions_lasso <- predict(model_lasso_optimal,
                                 s = lambda_optimal, newx = X_val)
test_predictions_lasso <- predict(model_lasso_optimal,
                                  s = lambda_optimal, newx = X_test)

# MAE for validation and test sets
val_mae_lasso <- mean(abs(y_val - val_predictions_lasso))
test_mae_lasso <- mean(abs(y_test - test_predictions_lasso))
cat("Validation MAE (Lasso Regression):", val_mae_lasso, "\n")
```

```
Validation MAE (Lasso Regression): 62.22935
```

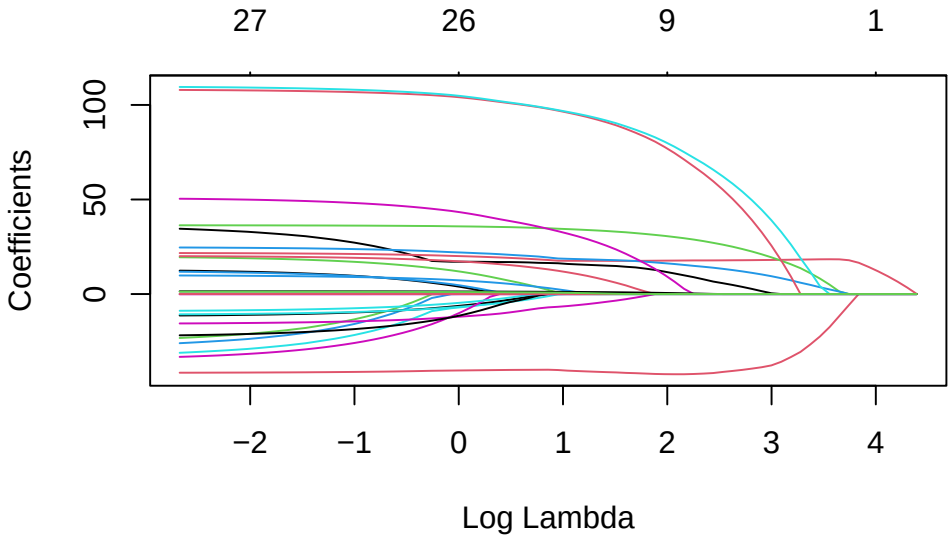
```r
cat("Test MAE (Lasso Regression):", test_mae_lasso, "\n")
```

```
Test MAE (Lasso Regression): 62.23003
```

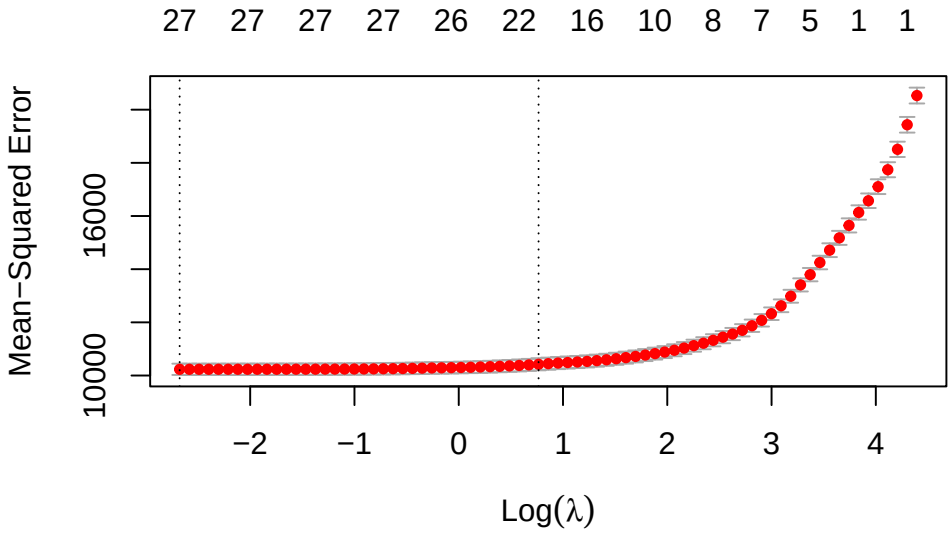
```r
# Extract and print coefficients at the optimal lambda
lasso_coefficients <- coef(model_lasso_optimal, s = lambda_optimal)
print(lasso_coefficients)
```

```
28 x 1 sparse Matrix of class "dgCMatrix"
                                                       s1
(Intercept)                                  1679.96614161
room_typeEntire.home.apt                       38.36592362
room_typePrivate.room                         -38.64107250
property_typeEntire.condo                     -23.99875480
property_typeEntire.home                      -26.73958161
property_typeEntire.rental.unit               -31.79175694
property_typePrivate.room.in.home             -15.42730626
property_typePrivate.room.in.rental.unit      -11.20963791
accommodates                                   21.65166544
bathrooms                                      36.36149405
bedrooms                                       24.62167675
beds                                          -10.68783702
neighbourhood_cleansedCamden                   50.41864825
neighbourhood_cleansedHackney                  12.37254486
neighbourhood_cleansedKensington.and.Chelsea  107.92630703
neighbourhood_cleansedSouthwark                19.48169721
neighbourhood_cleansedTower.Hamlets            11.79692830
neighbourhood_cleansedWestminster             109.51627554
latitude                                      -33.16729877
longitude                                     -21.78119553
host_is_superhost                              20.02446445
calculated_host_listings_count                  0.05258249
host_has_profile_pic                            9.82373939
host_identity_verified                         -8.85338667
number_of_reviews                              -0.13549564
availability_30                                 1.46792213
availability_365                                0.02770526
host_years_of_experience                        1.26458077
```

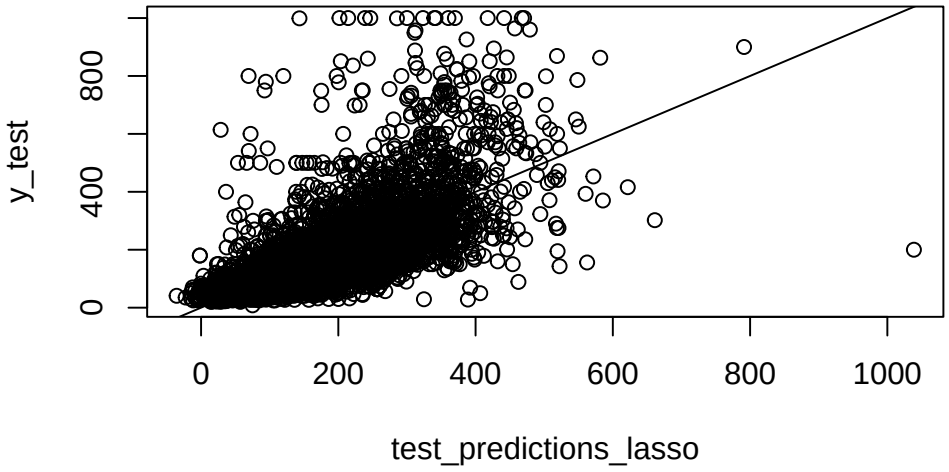
```r
# Plot coefficient shrinkage across lambda values
plot(model_lasso, xvar = "lambda")
```




```
# Plot cross-validation results
plot(cv_lasso)
```




```
# Plot predictions vs actual values (test set)
plot(test_predictions_lasso, y_test)
abline(0, 1)
```



Random forest

Hyperparameters tuning with Bayesian Optimization

```
# Define a function for optimization
optimization_rf <- function(mtry, ntree) {
  set.seed(42)

  # Train Random Forest with specified parameters
  model <- randomForest(
    x = X_train,
    y = y_train,
    mtry = floor(mtry), # mtry must be an integer
    ntree = floor(ntree), # ntree must be an integer
    importance = TRUE
  )

  # Predict on validation set
  val_predictions <- predict(model, newdata = X_val)

  # Calculate Mean Absolute Error (MAE)
  val_mae <- mean(abs(y_val - val_predictions))

  # Return negative MAE because Bayesian Optimization minimizes the objective
  return(list(Score = -val_mae))
}
```

```
# Define bounds for hyperparameters
bounds <- list(
  mtry = c(2, ncol(X_train)), # Range of mtry values
  ntree = c(100, 500)         # Range of ntree values
)


# Run Bayesian Optimization with progress tracking
with_progress({
  p <- progressor(along = seq_len(10)) # Define progress steps (10 iterations)

  opt_results <- bayesOpt(
    FUN = function(mtry, ntree) {
      p("Optimizing...") # Signal progress

      optimization_rf(mtry, ntree) # Call optimization function
    },
    bounds = bounds,
    initPoints = 5, # Number of random points to start with
    iters.n = 10,   # Number of iterations to refine the search
    acq = "ei"      # Acquisition function (expected improvement)
  )
})
print(opt_results)
# Extract best hyperparameters
best_params <- getBestPars(opt_results)
cat("Best Parameters:\n")
print(best_params)
```

Optimization results:

- mtry: 12.74439
- ntree: 443.0612

Random forest with tuned hyperparameters

```
# Training
model_rf <- randomForest(x = X_train, y = y_train, ntree = 443,
                         mtry = 13, importance = TRUE)
print(model_rf)
```

```
Call:
 randomForest(x = X_train, y = y_train, ntree = 443, mtry = 13,      importance = TRUE)
               Type of random forest: regression
                     Number of trees: 443
No. of variables tried at each split: 13

          Mean of squared residuals: 6520.634
                    % Var explained: 68.26
```

```r
# Evaluate variables
importance(model_rf) # Display importance metrics
```

	%IncMSE	IncNodePurity
room_typeEntire.home.apt	20.839613	30440948
room_typePrivate.room	31.800301	40226831
property_typeEntire.condo	8.878866	2557198
property_typeEntire.home	18.845825	2700874
property_typeEntire.rental.unit	19.375446	6428430
property_typePrivate.room.in.home	16.180416	1638205
property_typePrivate.room.in.rental.unit	25.089165	1489877
accommodates	56.077406	92707454
bathrooms	65.969427	89979962
bedrooms	48.700199	143381637
beds	40.571067	16642721
neighbourhood_cleansedCamden	38.209354	5128595
neighbourhood_cleansedHackney	9.109848	1127346
neighbourhood_cleansedKensington.and.Chelsea	53.420282	27290635
neighbourhood_cleansedSouthwark	10.102535	1223539
neighbourhood_cleansedTower.Hamlets	11.781277	1570543
neighbourhood_cleansedWestminster	78.455881	38470536
latitude	85.065089	66476848
longitude	111.120564	81186196
host_is_superhost	43.814337	6012616
calculated_host_listings_count	69.171663	34707058
host_has_profile_pic	4.646641	1964393
host_identity_verified	20.414257	4682627
number_of_reviews	43.122689	23367864
availability_30	89.974825	36821815
availability_365	57.526413	38449876
host_years_of_experience	63.141798	40957412

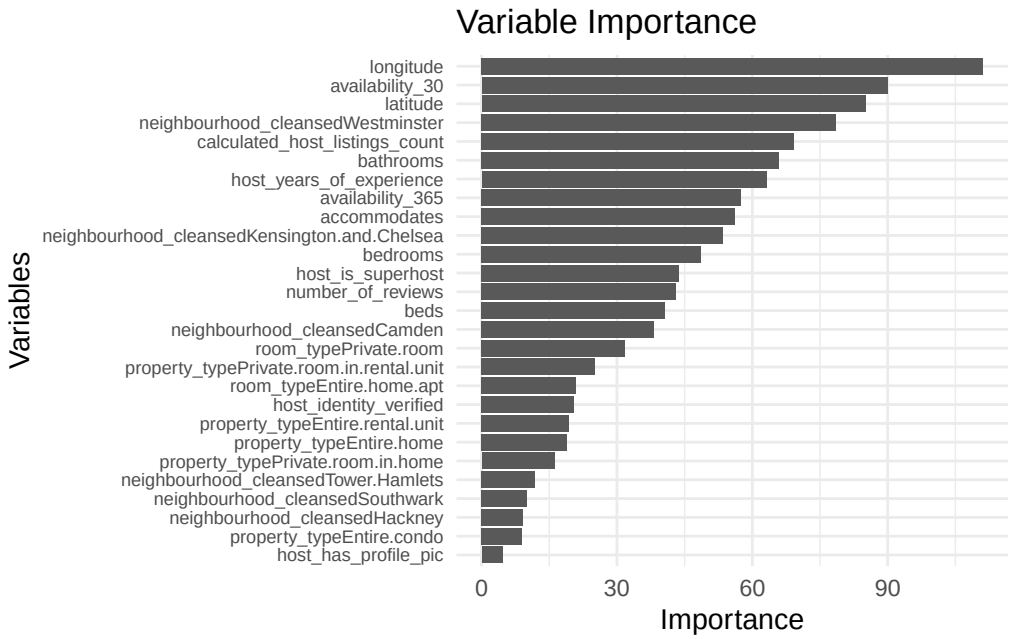
```r
# Predict and evaluate model performance
val_predictions_rf <- predict(model_rf, newdata = X_val)
test_predictions_rf <- predict(model_rf, newdata = X_test)

# MAE for validation and test sets
val_mae_rf <- mean(abs(y_val - val_predictions_rf))
test_mae_rf <- mean(abs(y_test - test_predictions_rf))
cat("Validation MAE (Random Forest):", val_mae_rf, "\n")
```

```
Validation MAE (Random Forest): 45.79155
```

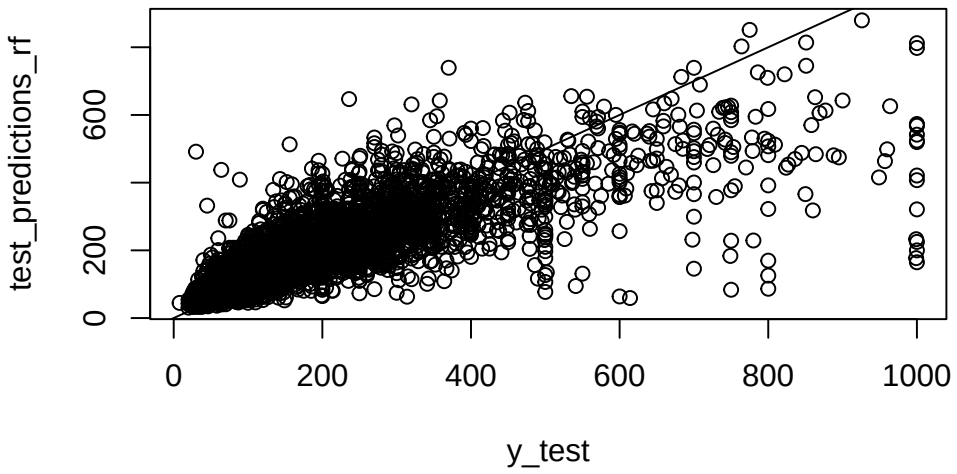
```r
cat("Test MAE (Random Forest):", test_mae_rf, "\n")
```

```
Test MAE (Random Forest): 45.39509
```

```r
# Plot variable importance based on %IncMSE

importance <- as.data.frame(varImp(model_rf)) # Extract importance values
importance$Variable <- rownames(importance) # Add variable names

# Sort by importance and plot top 10 variables
ggplot(importance, aes(x = reorder(Variable, Overall), y = Overall)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  theme(axis.text.y = element_text(size = 7)) +
  labs(x = "Variables", y = "Importance", title = "Variable Importance")
```




```r
# Plot actual vs predicted values
plot(y_test, test_predictions_rf)
abline(0, 1) # Add a diagonal line for reference
```

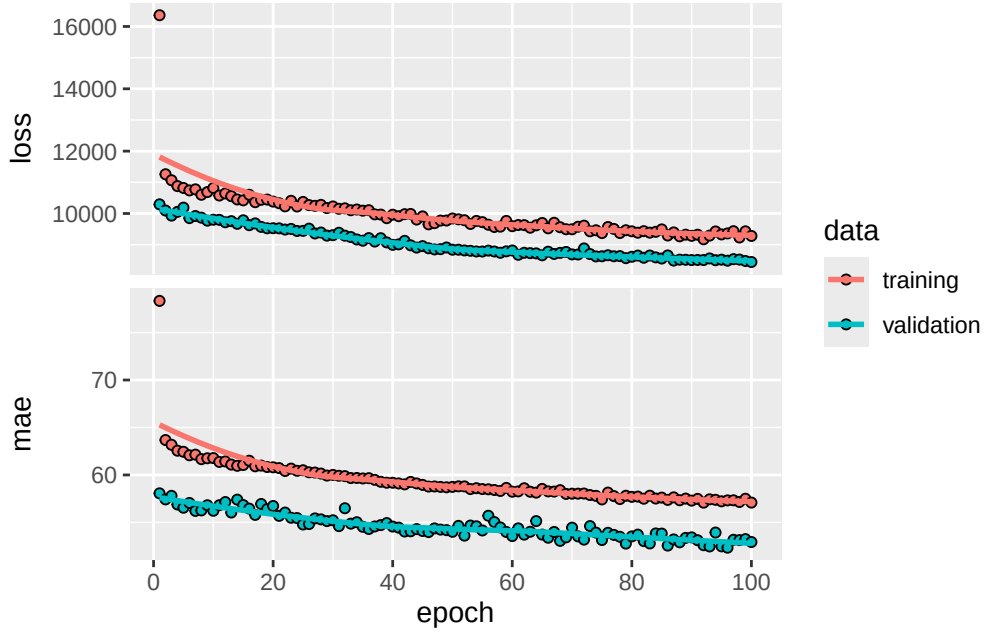


Feedforward neural network

```r
# Standardization
X_train <- scale(X_train)

# Use training set's mean and standard deviation for scaling validation and test sets
X_val <- scale(X_val, center = attr(X_train, "scaled:center"),
               scale = attr(X_train, "scaled:scale"))
X_test <- scale(X_test, center = attr(X_train, "scaled:center"),
                scale = attr(X_train, "scaled:scale"))

tf$random$set_seed(42) # Set seed for tf

# Define the model with dropout layers
model_fnn <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = ncol(X_train)) %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1)

# Compile the model
model_fnn %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae")
)

# Early stopping
early_stopping <- callback_early_stopping(
  monitor = "val_loss",
  patience = 10,
  restore_best_weights = TRUE
)

# Train the model
history <- model_fnn %>% fit(
  x = as.matrix(X_train),
  y = y_train,
  validation_data = list(as.matrix(X_val), y_val),
  epochs = 100,
  batch_size = 32,
  verbose = 0,
```

```
  callbacks = list(early_stopping)
)
```

```
# Plot training history
plot(history)
```




```
# Identify the chosen epoch with the lowest validation loss
best_epoch <- which.min(history$metrics$val_loss) # Index of minimum validation loss

cat("Best epoch (with lowest val_loss):", best_epoch, "\n")
```

```
Best epoch (with lowest val_loss): 100
```

```
cat("Validation loss at best epoch:", min(history$metrics$val_loss), "\n")
```

```
Validation loss at best epoch: 8443.774
```

```
# Evaluate
model_fnn %>% evaluate(as.matrix(X_val), y_val)
```

```
378/378 - 0s - loss: 8443.7744 - mae: 52.9353 - 64ms/epoch - 170us/step
```

```
      loss        mae
8443.77441   52.93534
```

```
model_fnn %>% evaluate(as.matrix(X_test), y_test)
```

```
189/189 - 0s - loss: 8331.1123 - mae: 53.6154 - 36ms/epoch - 190us/step
```

```
      loss        mae
8331.11230   53.61544
```