Seattle Airbnb Homestays Price Prediction

Karthik Garimella, Sandeep Alfred
MA5790 Predictive Modelling

Abstract

Seattle is a city surrounded by water, mountains and evergreen forests which is also home to the Space Needle[5]. Airbnb homestays are available all around the world with each bringing their unique interpretation of a cozy, comfortable and congenial place. Homestays in Seattle would be an enticing option for people visiting and the prices of homestays matters for the experience it might provide. Airbnb is a public company open about their data which lends a helping hand in predicting the price of a homestay. The main goal of the study is to predict the prices of Airbnb homestays in Seattle to understand how volatile the prices could be and determine if the models trained on the open data set. This study helps in providing the distribution of prices in a high-cost of living area and the ability to settle on a reasonable price for new homestays in the region. This study will focus on how the original data looks like, how it was transformed for model building with both linear and non-linear models being considered. The dataset would be split in train and test with cross validation resampling. The best models trained on the training sample will be run on the test set which will assist in selecting the best model and also the most important predictors useful in deducing the price of an Airbnb homestay in Seattle, USA.

Table of Contents

Abstract: 1
1. Background..... 3
2. Variable Introduction and Definitions..... 4
3. Preprocessing of the predictors..... 6
    a. Correlations..... 6
    b. Transformations..... 7
4. PCA 11
5. Splitting of the Data 11
6. Model Fitting 11
7. Summary 15
References 15
Appendix 1: Supportive documentation 16
Appendix 2: R Code 25

1. Background

Airbnb is an American company which operates in the short and long term homestays of experiences all over the world. Each homestay brings their own unique style with some homes having a pizzazz look, some traditional, some historic. Airbnb is an abbreviation of its original name: “**Airbed and Breakfast”**[1]**.** It acts as a broker and charges a commission from each booking. Airbnb was founded in 2008 by Brian Chesky, Nathan Blecharczyk, and Joe Gebbia. It is the best-known company for short-term housing[2][3]. Washington state has restrictions on how hosts must obtain licenses and cannot rent more than two units[4]. This could impact on how the prices could are set for the homestays. Predicting prices could help other potential Airbnb hosts to land on a reasonable price for their homestays in Seattle. Figure 1 shows the geospatial locations of the airbnb homestays used in this project.

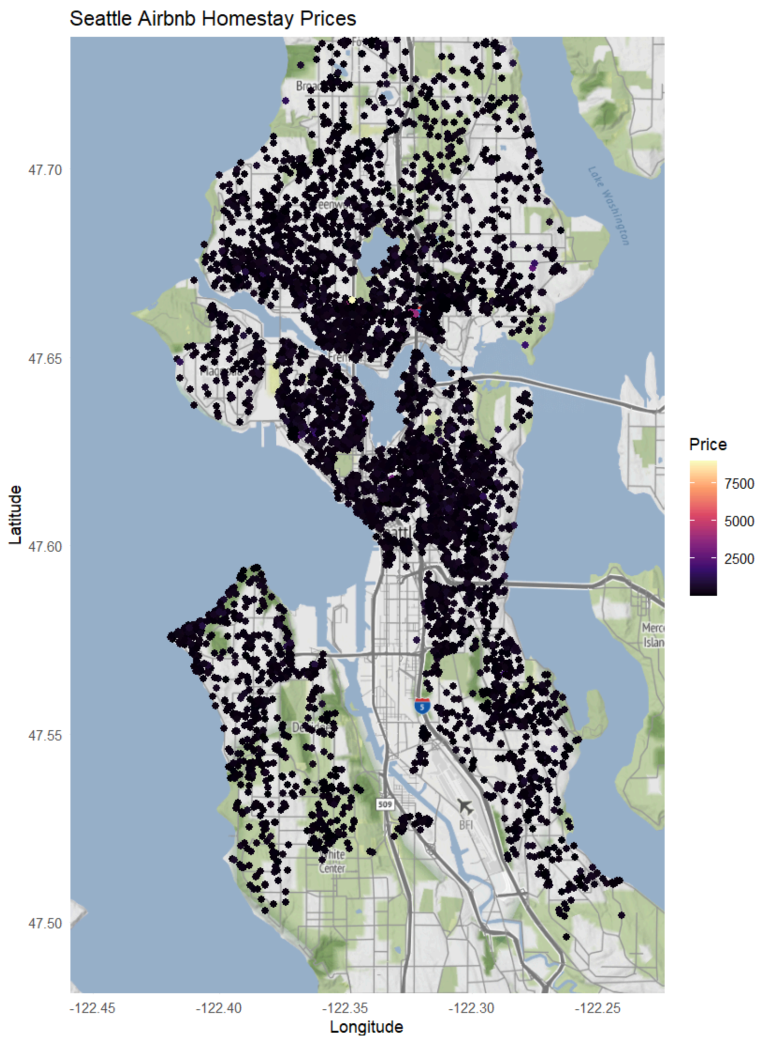


Figure 1. GeoSpatial Map of Airbnb Homestays in Seattle

2. Variable Introduction and Definitions:

The data has been retrieved from the Airbnb open dataset available for use by the public[6]. The Seattle dataset consists of 6442 samples and 77 predictors. There are a few samples where the response variable does not exist, i.e., the price of the Airbnb homestay is unavailable. After removing the Null response samples from the dataset, there are 6011 samples available for the data preprocessing. The dataset also consists of predictors which are either irrelevant or not feasible to convert the predictor as a dummy variable or a numerical value. These predictors consist of information about when the data was scraped, description of the Airbnb homestay, url for the homestay and host profile picture. After removing such predictors, we are left with 28 predictors with 14 numerical, 6 categorical, 5 logical and 3 date predictors in our dataset. The 28 predictors are described in table 1.

Field	**Type**	**Description**
host_since	date	The date the host/user was created. For hosts that are Airbnb guests this could be the date they registered as a guest.
host_response_time	text	Time taken for the host to respond
host_response_rate	numeric	That rate at which a host responds.
host_acceptance_rate	numeric	That rate at which a host accepts booking requests.
host_is_superhost	boolean [t=true; f=false]	If the host is the super host or not.
host_listings_count	text	The number of listings the host has (per Airbnb unknown calculations)
host_verifications	numeric	Number of verified listings of the host
host_has_profile_pic	boolean [t=true; f=false]	Does the host have a profile picture?
host_identity_verified	boolean [t=true; f=false]	Is the host verified by Airbnb?
latitude	numeric	Uses the World Geodetic System (WGS84) projection for latitude and longitude.
longitude	numeric	Uses the World Geodetic System (WGS84)

		projection for latitude and longitude.
property_type	text	Self selected property type. Hotels and Bed and Breakfasts are described as such by their hosts in this field
accommodates	integer	The maximum capacity of the listing
bathrooms	numeric	The number of bathrooms in the listing
bedrooms	integer	The number of bedrooms
beds	integer	The number of bed(s)
price	currency	Daily price in local currency. NOTE: the $ sign is a technical artifact of the export, please ignore it
minimum_nights	integer	Minimum number of night stay for the listing (calendar rules may be different)
maximum_nights	integer	Maximum number of night stay for the listing (calendar rules may be different)
has_availability	boolean	[t=true; f=false]
availability_30	integer	The availability of the listing 30 days in the future as determined by the calendar. Note a listing may not be available because it has been booked by a guest or blocked by the host.
number_of_reviews	integer	The number of reviews the listing has
first_review	date	The date of the first/oldest review
last_review	date	The date of the last/newest review
review_scores_value		Rating of the homestay
instant_bookable	boolean	[t=true; f=false]. Whether the guest can automatically book the listing without the host requiring to accept their booking request. An indicator of a commercial listing.
calculated_host_listings_count	integer	The number of listings the host has in the current scrape, in the city/region geography.
reviews_per_month	numeric	The average number of reviews per month the listing has over the lifetime of the listing.

Table 1. Variable Descriptions[6]

3. Preprocessing of the predictors

The dataset we have currently consists of 6011 samples and 28 predictors. The logical columns are converted to True/False, i.e., 0/1. The numerical columns are kept as and the date predictors are converted to a date data type with day, month and year being split into separate columns. The month predictors only have 12 unique values which are converted into a factor data type considering it as a categorical variable. The same applies to the predictor bedrooms which comprises only 10 unique values. Regarding all the categorical variables, they are converted into dummy variables which increases our predictor size to 75. The degenerate columns are removed from the set of predictors which leaves us with 54 predictors.

Figure 2. shows there are missing values in some of the predictor samples which models cannot handle. A KNN imputation with k = 5 is performed on the dataset which will use the nearest neighbors to impute the missing values.

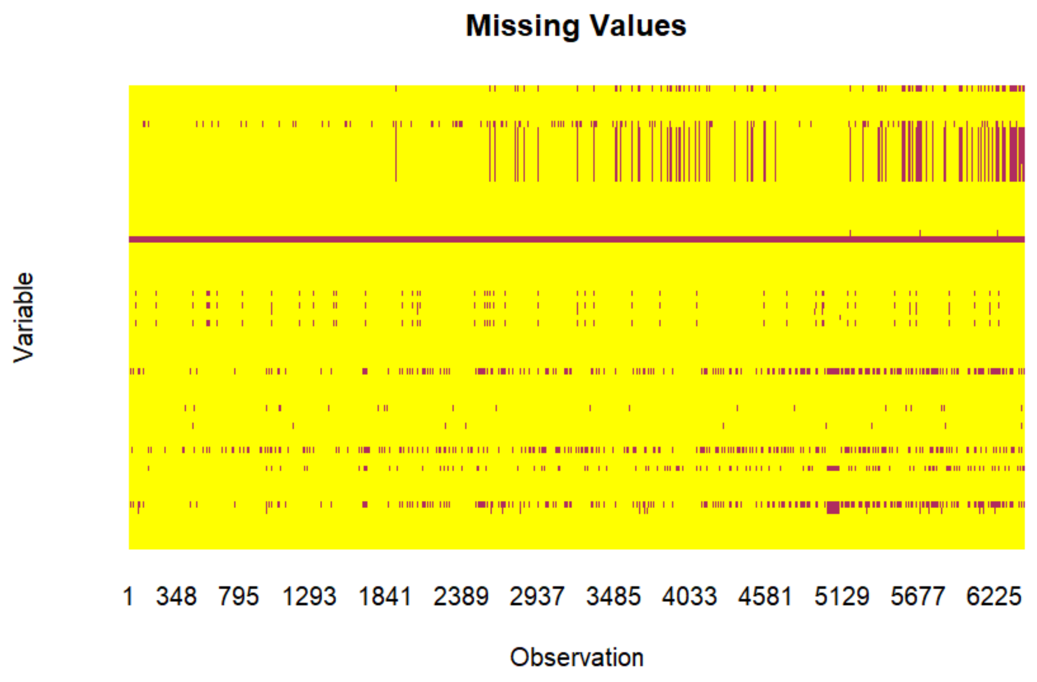


Figure 2. Missing Values Heatmap

a. Correlations

A correlation plot for the remaining variables is shown in figure 3. The relationship between each predictor might give a glimpse on which predictors are highly correlated that will aid in removing them. The legend indicates which predictors are correlated with a blue spot exhibiting a high positive correlation and a red spot representing a negative correlation with darker the shade of the color meaning a higher correlation. A threshold of 0.80 removes 3 predictors which reduces the predictor size to 51. The dataset without the highly correlated predictors was only used with models which cannot handle highly correlated variables.

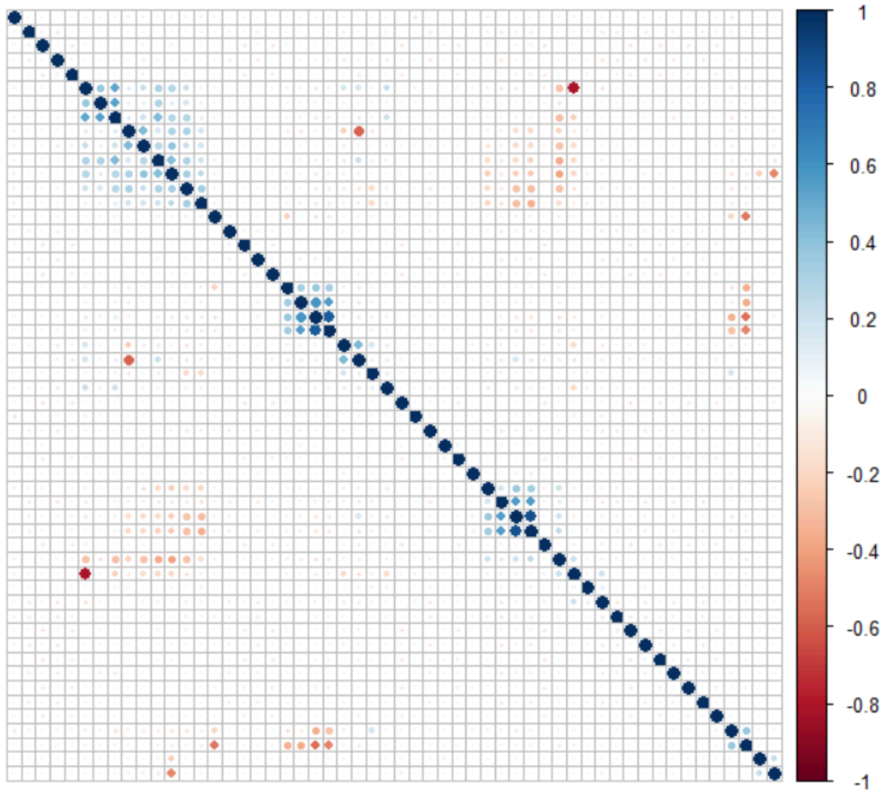


Figure 3. Correlation between Predictors

b. Transformations

The data is first preprocessed by centering and scaling. Before splitting the data for model fitting, the distribution of the predictors is analyzed. Most of the predictors are highly skewed (i.e., a skewed distribution of the predictor data, shown in figure 4) which warrants a transformation. A YeoJohnson transformation is performed to normalize the data. The YeoJohnson transformation can handle predictors with negative values, hence it was preferred in the place of BoxCox. Since our response variable was also highly skewed, we transformed the response variable using YeoJohnson transformation. Figure 5 shows how much the data distribution has improved after the transformation was applied. Figure 6 shows how the response variable distribution has changed after the transformation.

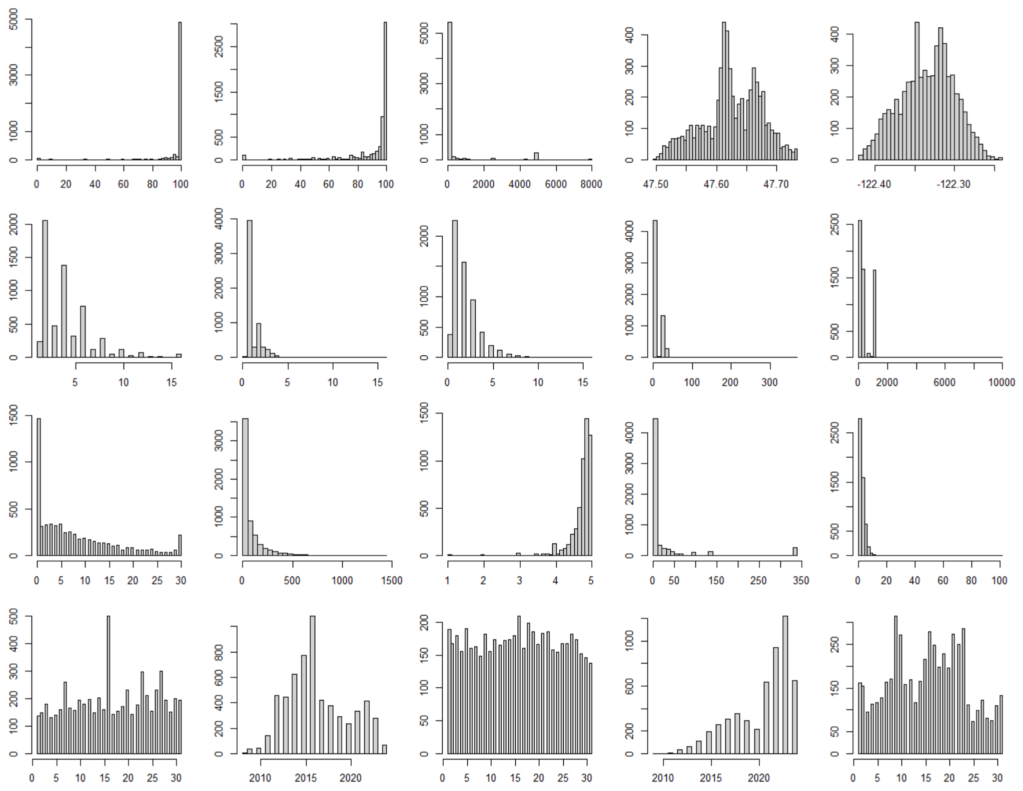


Figure 4. Before YeoJohnson Transformation

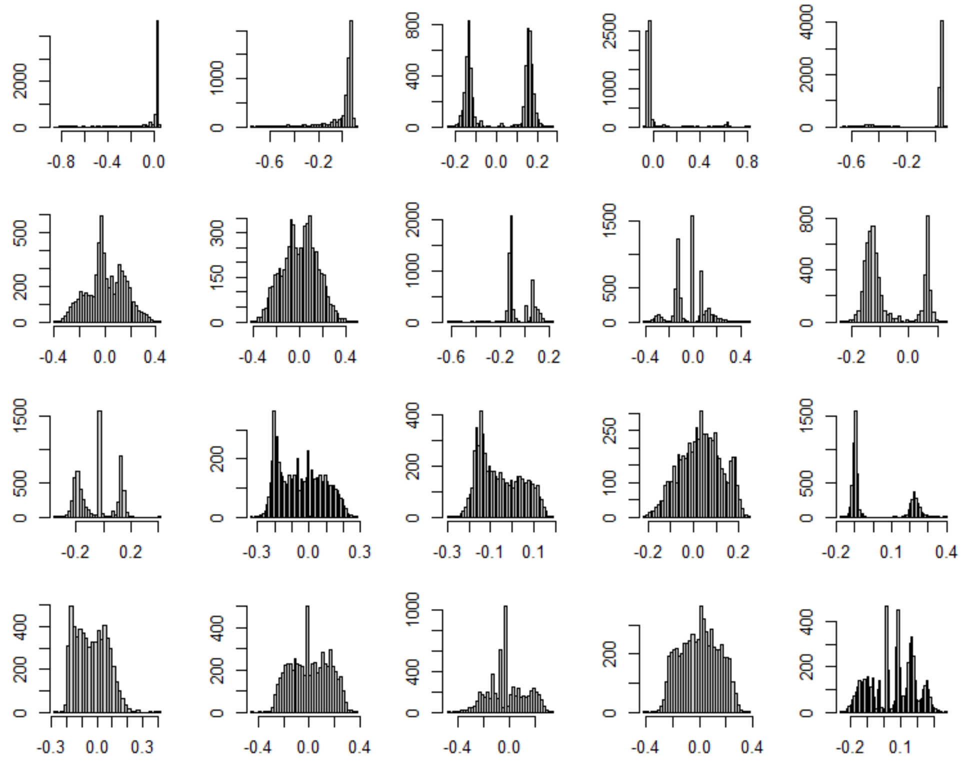


Figure 5. After YeoJohnson Transformation

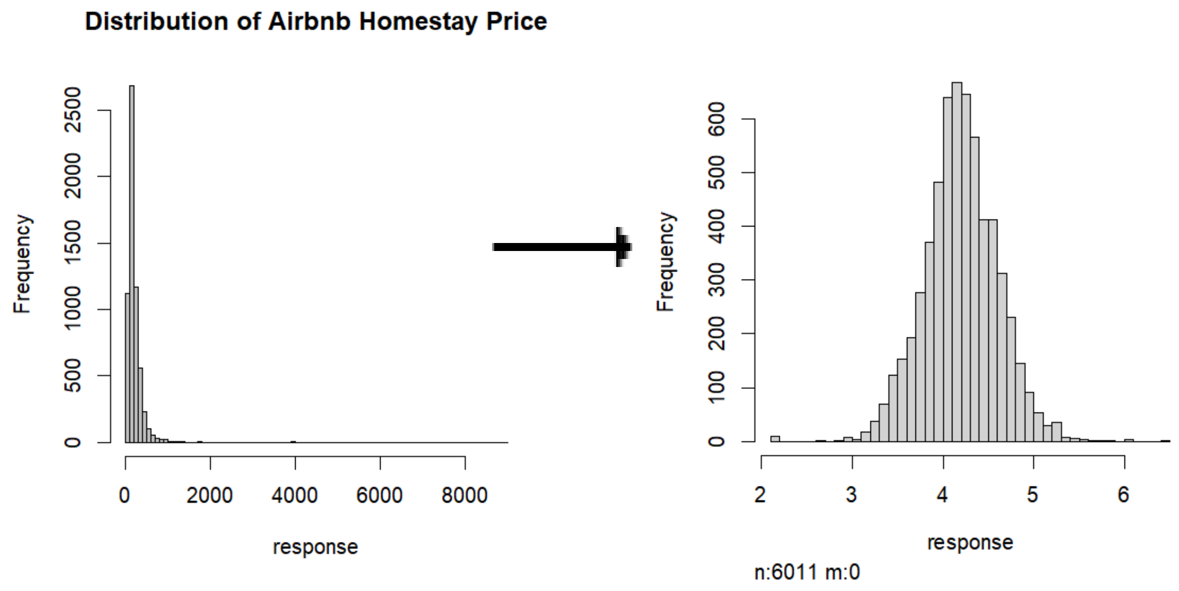


Figure 6. Response variable transformation

The existence of outliers also degrades the performance of the few models that the data is being trained on which requires a spatial sign transformation. The spatial sign transformation helps in minimizing the effect of outliers which might skew the model.

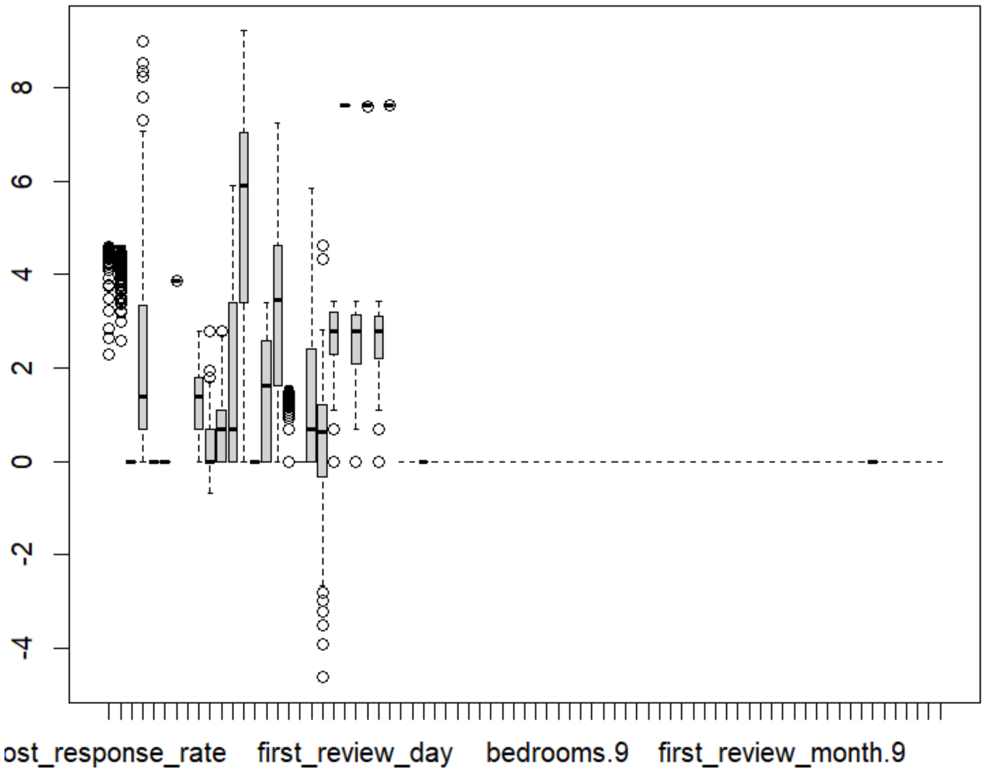


Figure 7. Before spatialSign Transformation

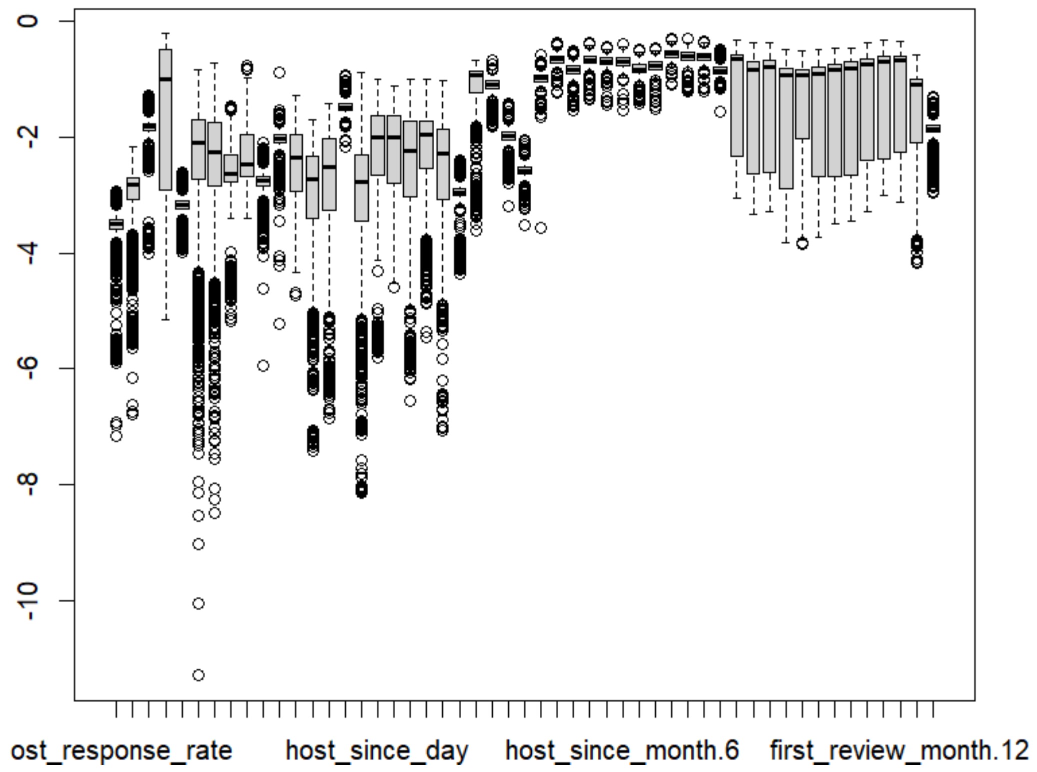
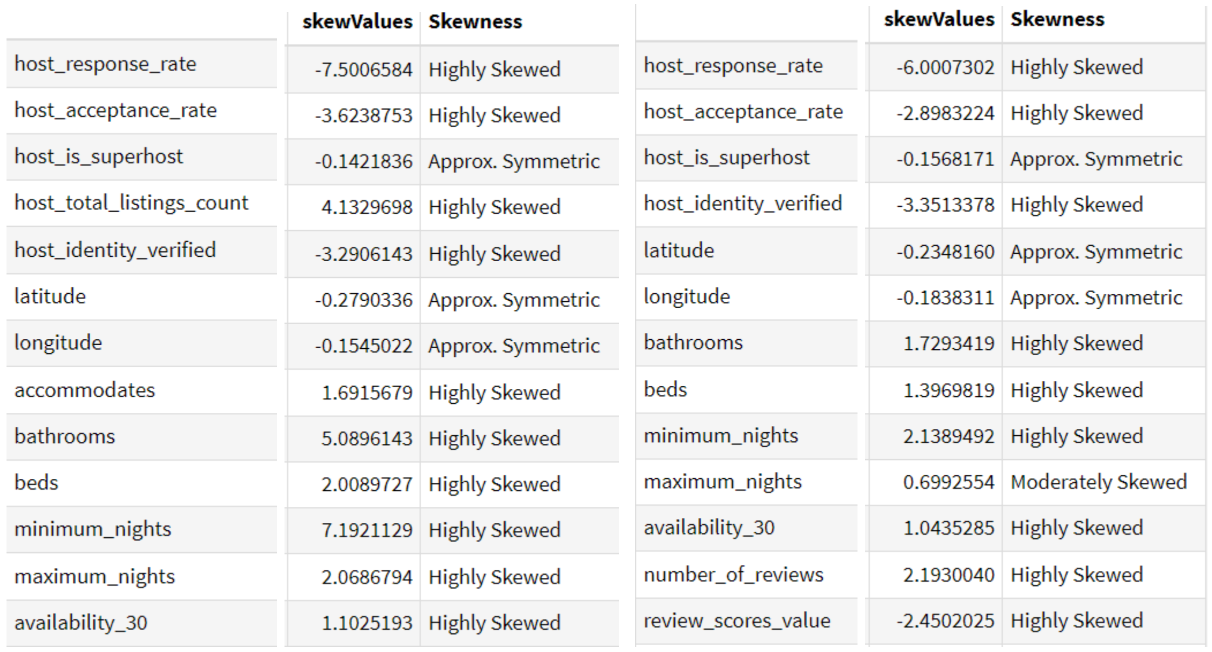


Figure 8. After spatialSign Transformation

After performing the above transformations, the skewness value of each predictor is analyzed to examine how the transformations affect the predictor distribution, figures 7 and 8 show the improvement in outliers after spatial sign transformation.



	skewValues	Skewness
host_response_rate	-7.5006584	Highly Skewed
host_acceptance_rate	-3.6238753	Highly Skewed
host_is_superhost	-0.1421836	Approx. Symmetric
host_total_listings_count	4.1329698	Highly Skewed
host_identity_verified	-3.2906143	Highly Skewed
latitude	-0.2790336	Approx. Symmetric
longitude	-0.1545022	Approx. Symmetric
accommodates	1.6915679	Highly Skewed
bathrooms	5.0896143	Highly Skewed
beds	2.0089727	Highly Skewed
minimum_nights	7.1921129	Highly Skewed
maximum_nights	2.0686794	Highly Skewed
availability_30	1.1025193	Highly Skewed

	skewValues	Skewness
host_response_rate	-6.0007302	Highly Skewed
host_acceptance_rate	-2.8983224	Highly Skewed
host_is_superhost	-0.1568171	Approx. Symmetric
host_identity_verified	-3.3513378	Highly Skewed
latitude	-0.2348160	Approx. Symmetric
longitude	-0.1838311	Approx. Symmetric
bathrooms	1.7293419	Highly Skewed
beds	1.3969819	Highly Skewed
minimum_nights	2.1389492	Highly Skewed
maximum_nights	0.6992554	Moderately Skewed
availability_30	1.0435285	Highly Skewed
number_of_reviews	2.1930040	Highly Skewed
review_scores_value	-2.4502025	Highly Skewed

Figure 9. Skewness Before and After Transformations.

4. PCA

We explored Principal component analysis (PCA) which is a linear dimensionality reduction technique in order to find the best combination of predictors for modelling. This was employed with the idea to see if PCA components perform well compared to transformed data. The predictors were centered and scaled with YeoJohnson transformation applied, after which PCA was performed on these predictors. PCA needed 42 components to capture 95% variance in the data. Figure 10 shows the cumulative variance explained.

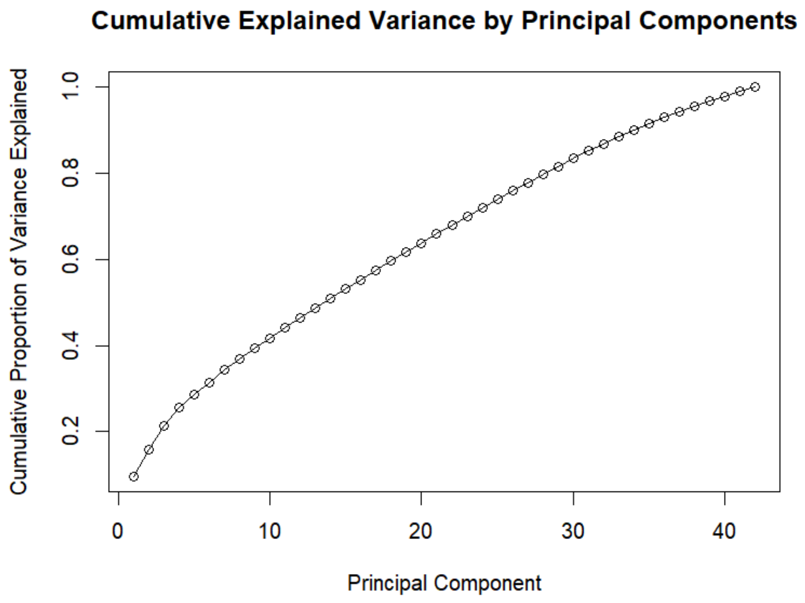


Figure 10. Cumulative Explained Variance by Principal Components

5. Splitting of the Data

The response variable is continuous, i.e., price. The data is split randomly into an 80/20 ratio with 80% of the data being used for training and 20% for testing. The train dataset consists of 4811 samples and the test set consists of 1200 samples. A 10-fold cross validation resampling method is utilized. Given this is a regression problem, RMSE was used as the metric to choose the best parameters.

6. Model Fitting

A range of linear and non-linear regression models are used to train on the training data. The models are tested on the testing data with RMSE as our primary metric along with R^2 to evaluate the model. The models are tested with multiple hyperparameters which will ensure the data is tested on different combinations to avoid underfitting or overfitting. The model should be able to generalize well on the unseen test data to obtain the best evaluation metrics. The table 2 showcases how the linear and non-linear regression models performed on the Airbnb Seattle dataset. The parameter tuning of each model is made available in the appendix 1.

Model	**Best Tuning Parameter**	**Training RMSE**	**Training R2**	**Testing RMSE**	**Testing R2**
OLR	Intercept = TRUE	0.2836184	0.5210050	0.2811907	0.5032191
Ridge	Lambda = 0	0.3111092	0.5218327	0.2811907	0.5032191
Lasso	Fraction = 0.89	0.4045804	0.5241353	0.2806709	0.5040488
Enet	Fraction = 0.9 , Lambda = 0	0.3882141	0.5234125	0.2807076	0.5040030
PLS	Components = 17	0.3037154	0.5224680	0.2812087	0.5031594
KNN	k = 25	0.3317944	0.4357944	0.2937164	0.4880424
MARS	nPrune = 31 , Degree = 3	0.4471146	0.6575060	0.2457432	0.6224974
NN	**Size = 10 , Decay = 0.1 , Bag = FALSE**	**0.2857756**	**0.6591239**	**0.2292414**	**0.6691850**
SVM	Sigma = 0.1 , C = 4	0.4096800	NA	0.3029597	0.4752846

Table 2. Transformed Data Summary.

The same models were now trained and tested with PCA data to see if there is any development in the performance of the models. The summary in table 3 shows how well PCA components have performed.

Model	Best Tuning Parameter	Training RMSE	Training R2	Testing RMSE	Testing R2
OLR	Intercept = TRUE	0.2738678	0.552975	0.2739719	0.5269756
Ridge	Lambda = 0.07142857	0.2735751	0.5548037	0.2739863	0.5268789
Lasso	Fraction = 0.9478947	0.4036720	0.5559061	0.2735700	0.5276964
Enet	Fraction = 0.95 , Lambda = 0	0.3823332	0.5567087	0.273582	0.5276680
PLS	Components = 6	0.2928899	0.5549160	0.2739776	0.5269389
KNN	k = 9	0.2908845	0.5288828	0.2644643	0.5665334
MARS	nPrune = 40 , Degree = 3	0.4093491	0.5558973	0.2523942	0.5996409
NN	Size = 10 , Decay = 2 , Bag = FALSE	0.4586250	0.6261046	0.2441472	0.6252260
SVM	**Sigma = 0.0089101 , C = 4**	**0.2807831**	**0.6368321**	**0.2420689**	**0.6307107**

Table 3. PCA Summary.

The best model overall is Neural Networks with the lowest training RMSE of 0.2857756 and the highest R2 score of 0.6591239. The testing RMSE is 0.2292414 and the R2 metric is 0.6691850. Neural Networks perform the best among all the models throughout all the training and testing RMSE and R2 evaluation metrics. **The PCA version of SVM performed close to neural networks, making it the second best model.** Also, the PCA version and the transformed data performed closely similar, yet, the transformed data performed better overall.

Table 4 shows the performance of the best models on the training and testing sets. With testing RMSE of 0.22 Neural Networks performs the best while explaining 0.66 variance in the data. The PCA version of SVM closely follows neural networks with 0.24 testing RMSE and 0.63 Rsquared.

Model	**Best Tuning Parameter**	**Training RMSE**	**Training R2**	**Testing RMSE**	**Testing R2**
NN	Size = 10 , Decay = 0.1 , Bag = FALSE	0.2857756	0.6591239	0.2292414	0.6691850
SVM	Sigma = 0.0089101 , C = 4	0.2807831	0.6368321	0.2420689	0.6307107

Table 4. Top Best Models performing on the Testing set.

The most important predictors for the best model are given below:

```
only 20 most important variables shown (out of 51)


                    Overall
beds                100.000
bathrooms            73.010
bedrooms.1           69.488
room_typePrivate.room  61.933
bedrooms.3           40.595
minimum_nights       14.548
bedrooms.2           13.263
availability_30      10.688
latitude             10.410
instant_bookable      8.723
```

```
host_is_superhost       6.937
host_acceptance_rate    6.204
first_review_year       5.693
first_review_month.7    4.993
last_review_year        4.617
maximum_nights          4.320
number_of_reviews       3.550
last_review_month.6     3.483
reviews_per_month       3.233
first_review_month.8    3.215
```

7. Summary

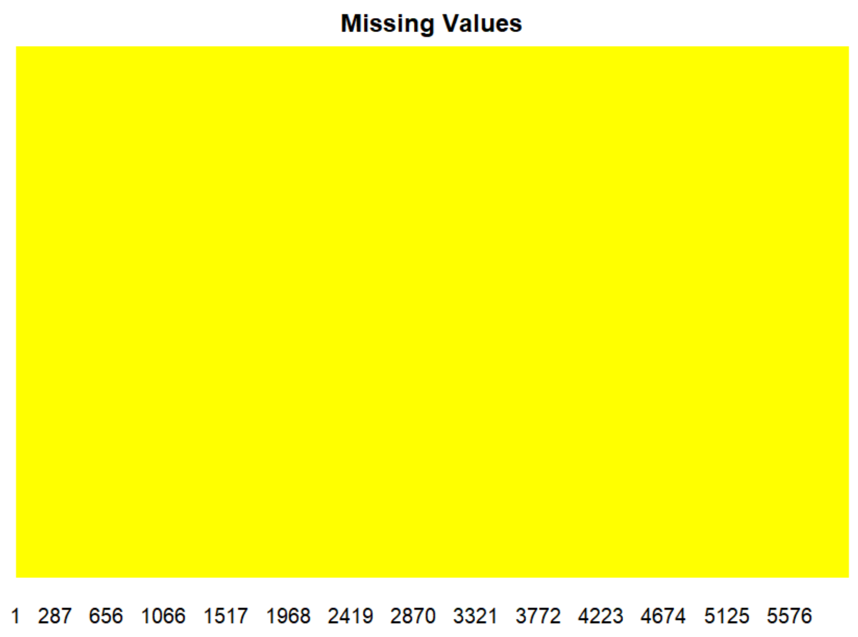
The best model is Neural Networks with the hyperparameters of size = 10 , decay = 0.1 , bag = FALSE to predict the price of an Airbnb homestay in Seattle. The RMSE for the model on the test dataset is 0.2292414 and an R2 of 0.6691850. The RMSE score of the model is low which indicates that model predictions are remarkably close to the ground truth. The R2 score also implies that the model is not effectively capturing the variance of the response variable which indicates a moderate fit on the training data. The conclusion is that the Seattle Airbnb dataset does not contain good enough predictors for predicting the response variable price. Transforming the response model was the key to achieving the impressive RMSE scores displayed by the models.

References

[1] "Why Is It Called Airbnb? The Origin Story and Its Impact Today".
[2] Cadwalladr, Carole (September 16, 2013). "Airbnb: the travel revolution in our spare rooms". The Observer. Archived from the original on February 23, 2023. Retrieved May 11, 2023 – via The Guardian.
[3] Thompson, Ben (July 1, 2015). "Airbnb and the Internet Revolution". Stratechery. Archived from the original on March 6, 2023. Retrieved May 11, 2023.
[4] Austermuhle, Martin (January 5, 2022). "D.C. To Start Restricting And Regulating Airbnb And Other Short-Term Rentals". WAMU. Archived from the original on January 5, 2022.
[5] Google Search Results
[6] Inside Airbnb

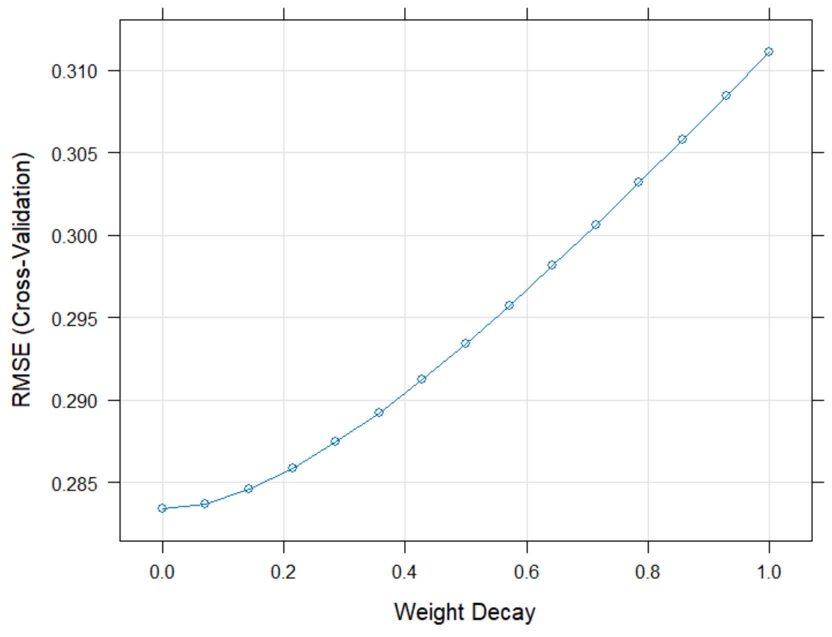
Appendix 1. Supportive documentation for preprocessing and parameter tuning.

A] Missing value in predictors after kNN imputation

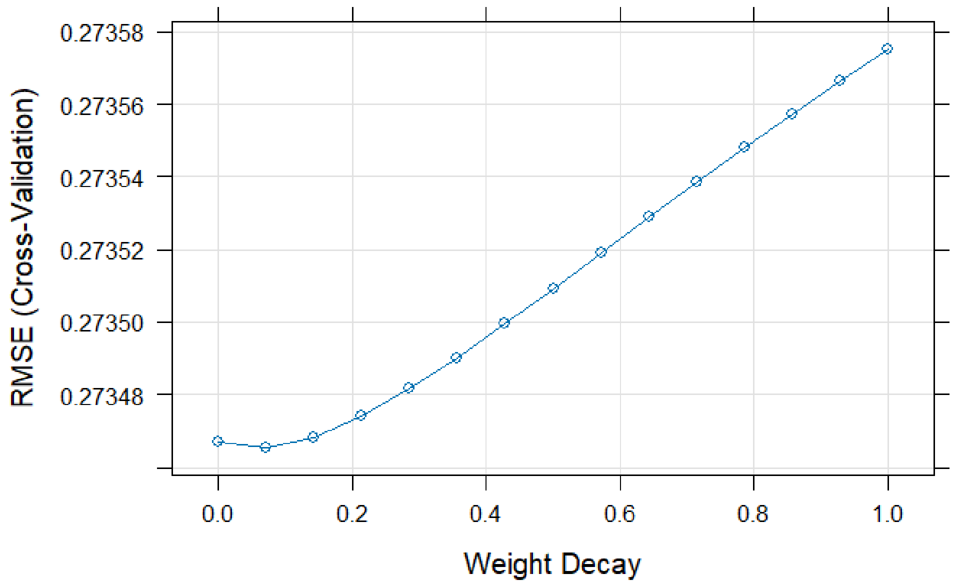


B] Linear Models:

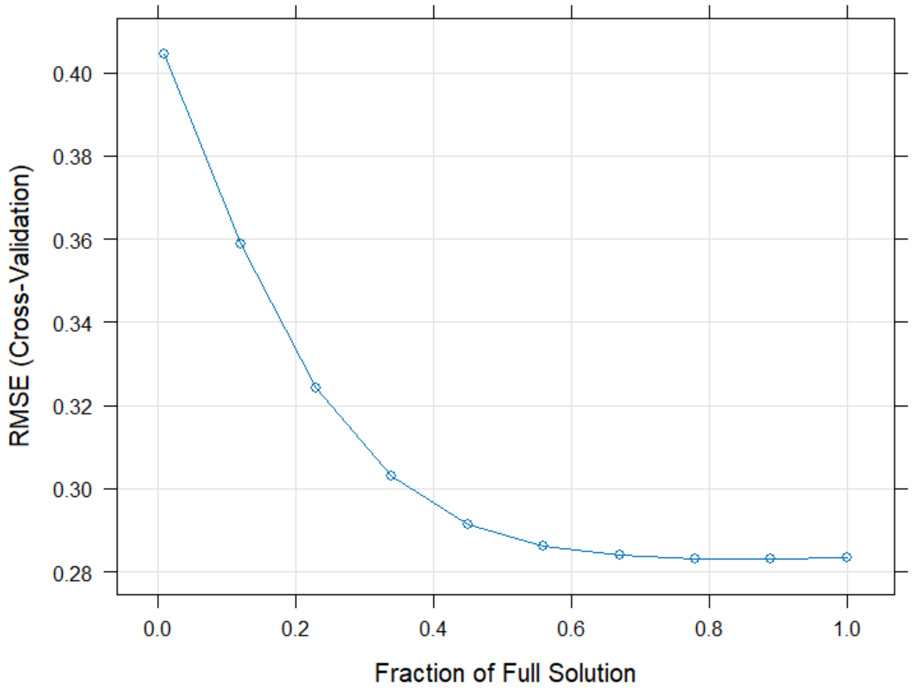
Ridge Regression:



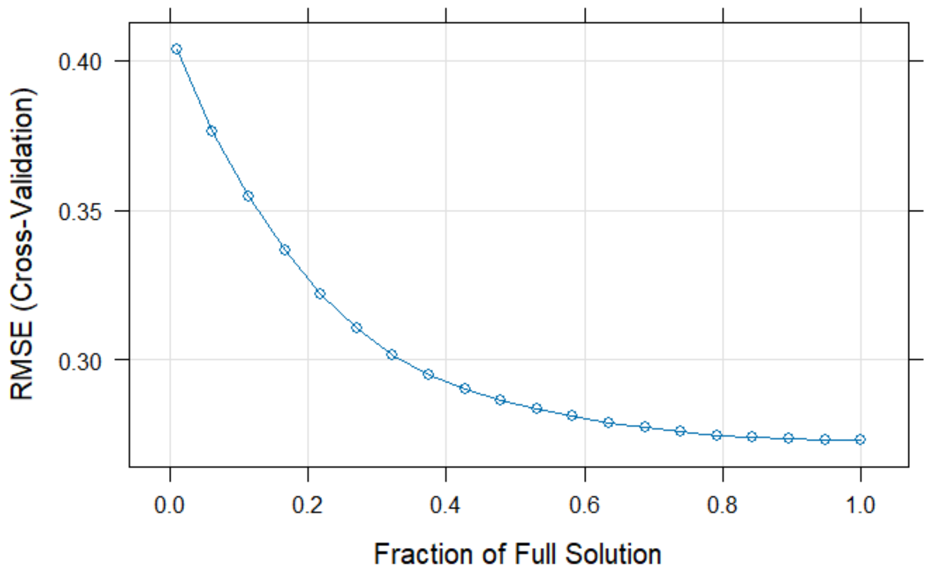
Ridge Regression PCA:



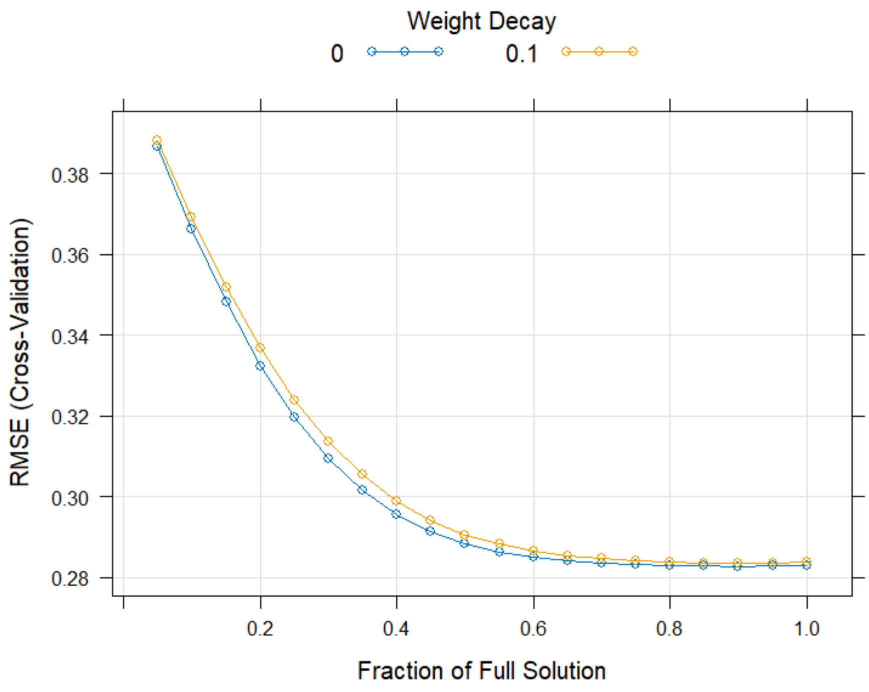
Lasso Regression:



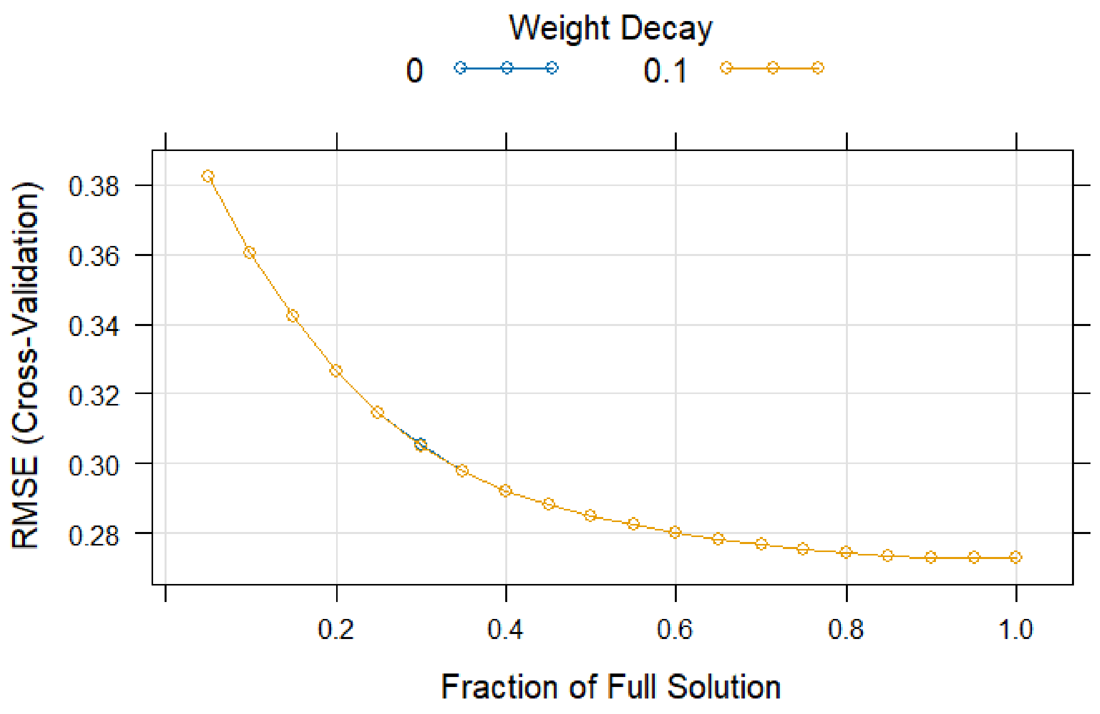
Lasso PCA:



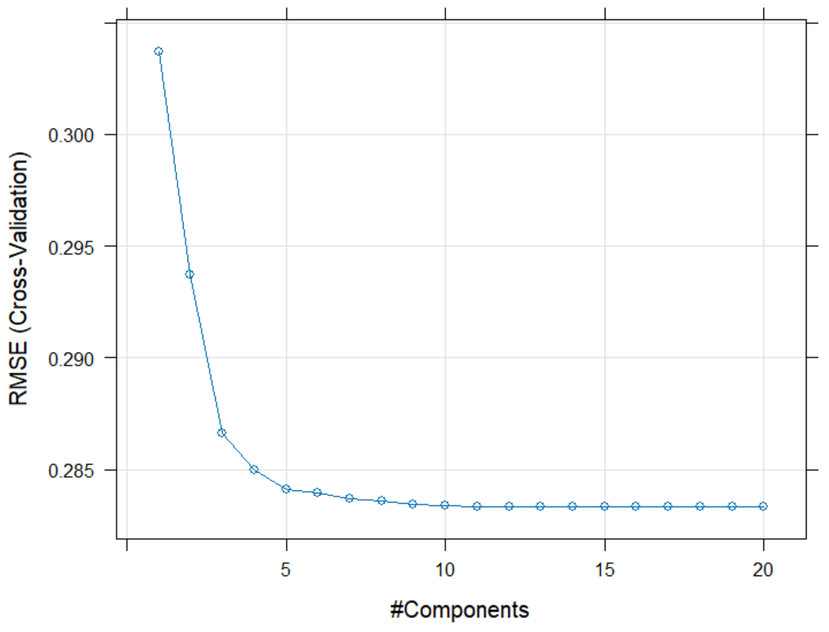
Elastic Net Regression:



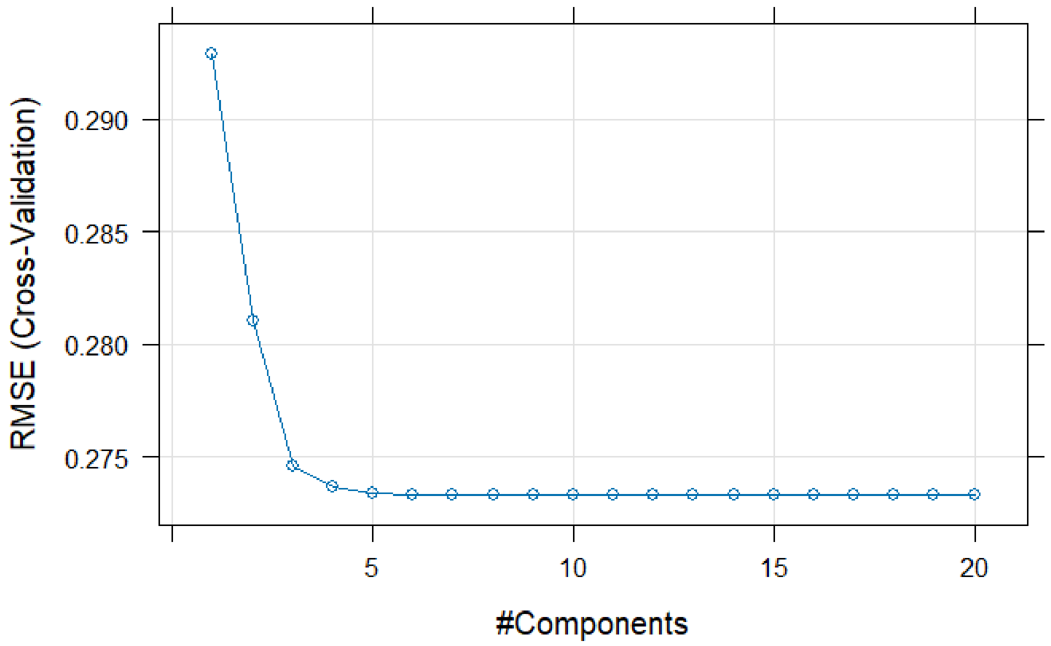
Elastic Net Regression PCA:



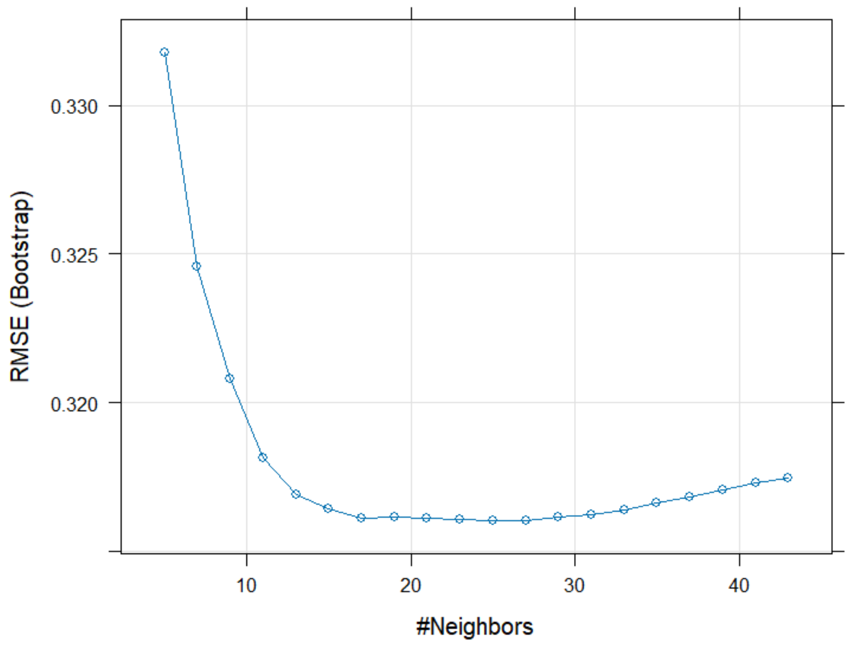
Partial Least Squares Regression:



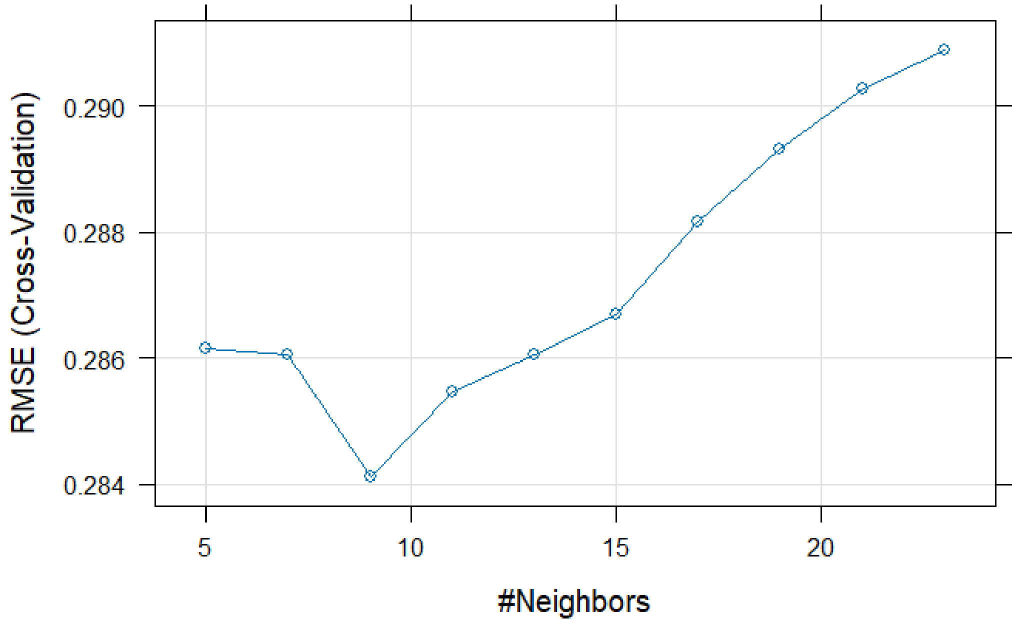
Partial Least Squares PCA:



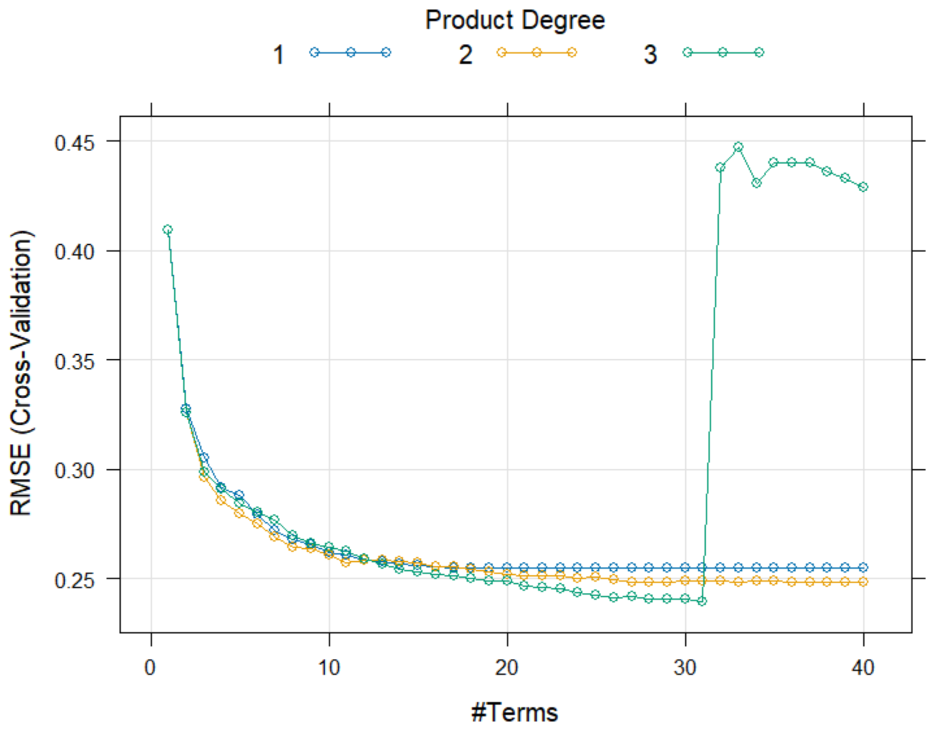
Non-Linear Models:
kNN:



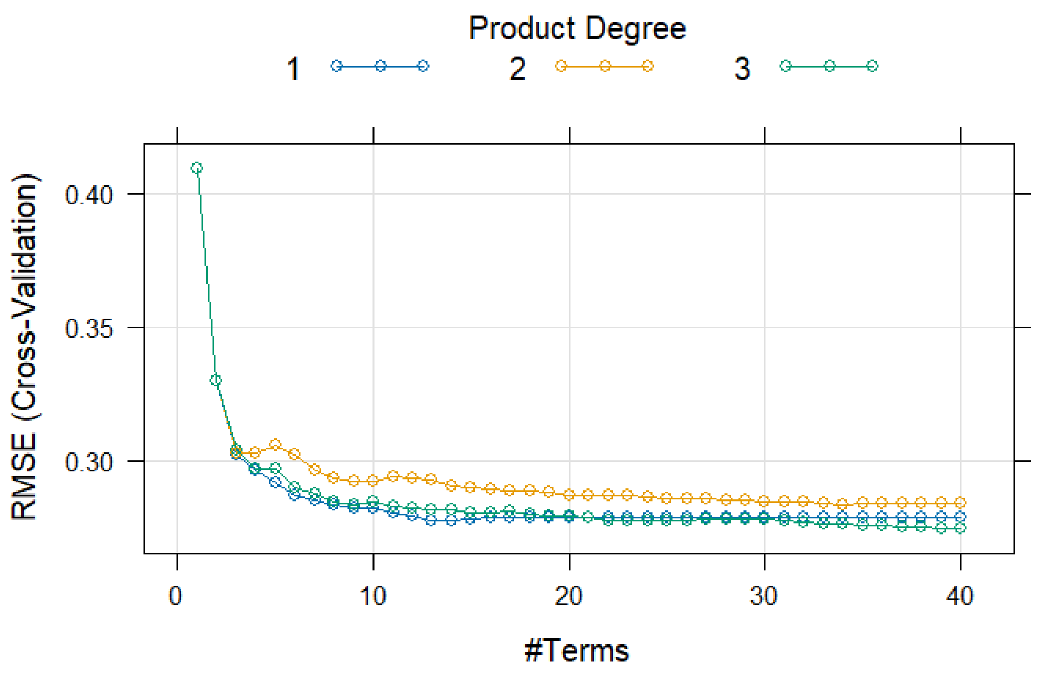
KNN PCA:


RMSE (Cross-Validation)
0.284
0.286
0.288
0.290
5
10
15
20
#Neighbors

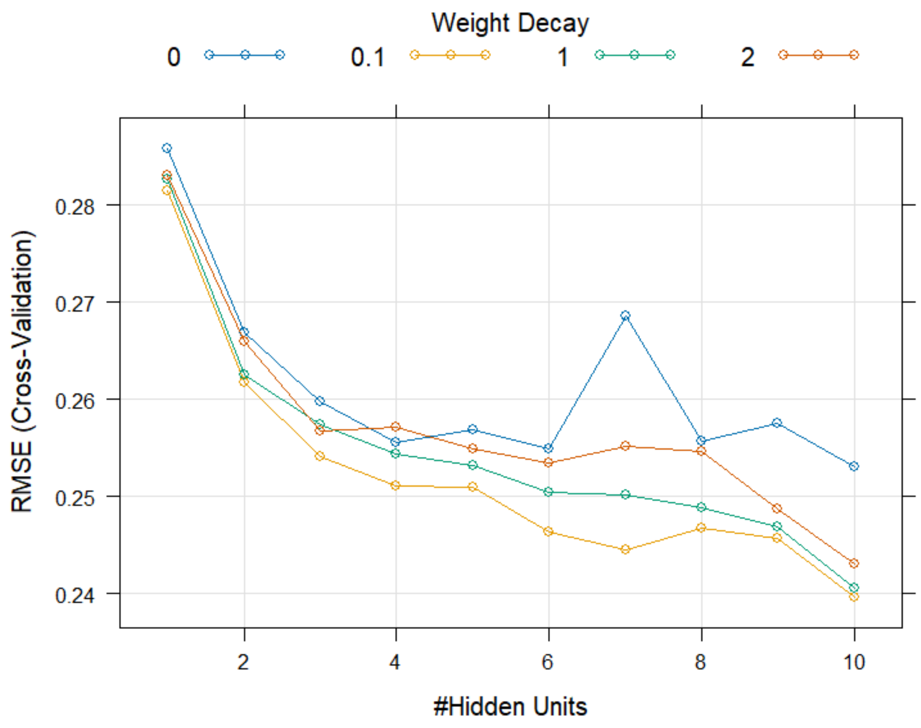
MARS:


Product Degree
1
2
3
RMSE (Cross-Validation)
0.25
0.30
0.35
0.40
0.45
0
10
20
30
40
#Terms

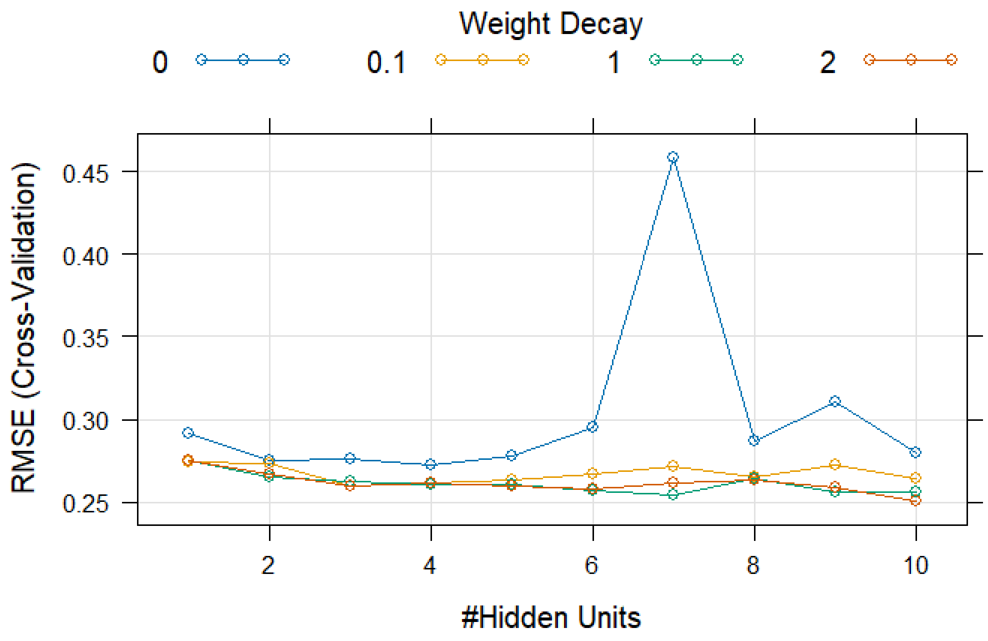
MARS PCA:



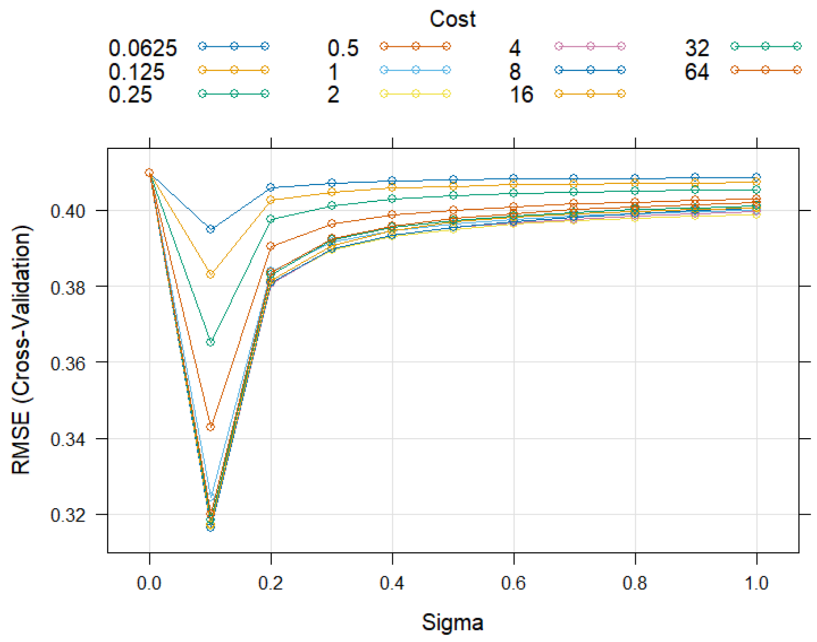
Neural Networks:



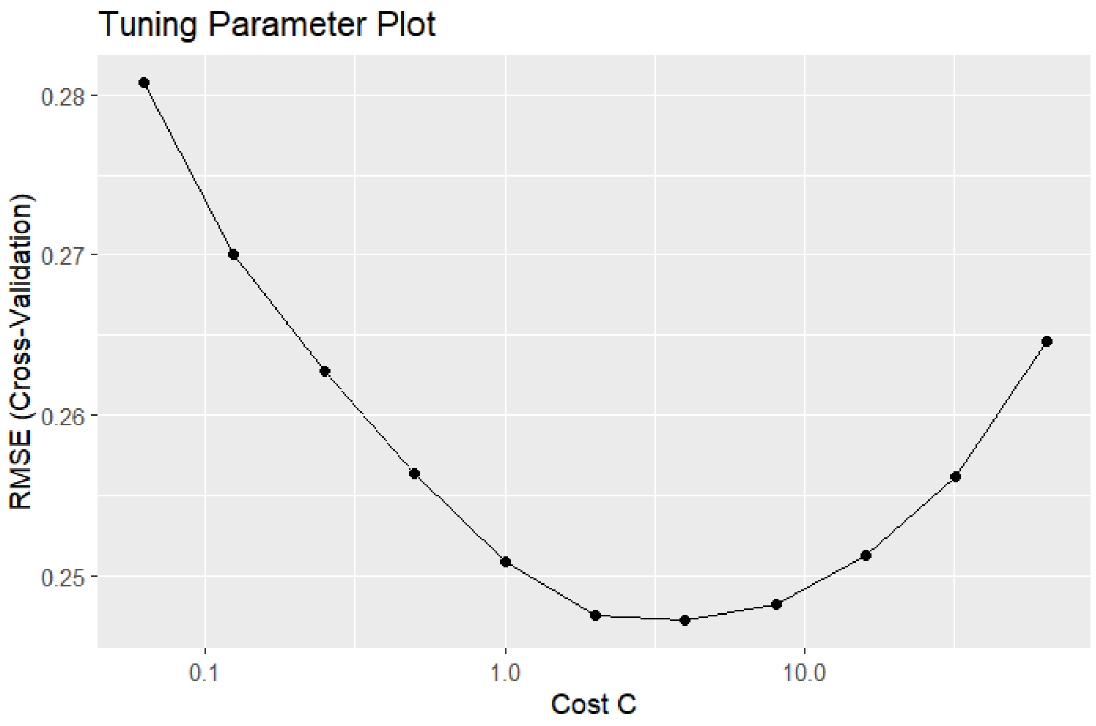
Neural Networks PCA:



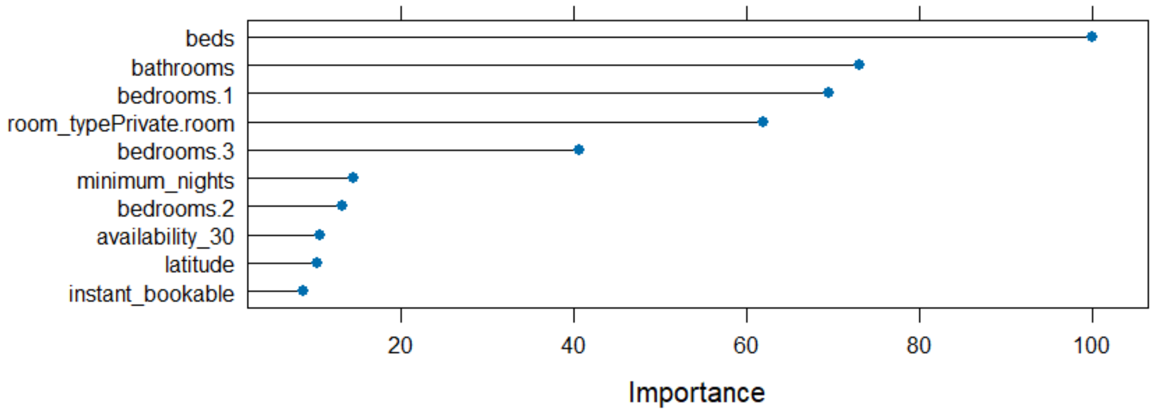
SVM:



SVM PCA:



Importance Score for predictors in Neural Network model



Appendix 2: R Code

```
# Libraries

library(readr)
library(dplyr)
library(knitr)
library(kableExtra)
library(tidyr)
library(caret)
library(corrplot)
library(e1071)
library(AppliedPredictiveModeling)
library(tidyverse)
library(Hmisc)
library(mlbench)
library(e1071)
library(caret)
library(lubridate)
library(ggplot2)

# Folder Path
folder_path <- "E:\\Coursework\\predictive\\project\\data"

gz_files <- list.files(folder_path, pattern = "\\.gz$", full.names = TRUE)

city_from_folder <- basename(folder_path) # Extracts the last part of the folder path
(which should be the city name)

data_list <- list()

for (file in gz_files) {
  file_name <- basename(file)
  city_from_file <- gsub("\\.gz$", "", file_name)
  city_state_split <- strsplit(city_from_file, ", ")[[1]]

  print(city_state_split[2])

  city_from_file <- city_state_split[1]
  state_from_file <- ifelse(length(city_from_file) > 1, city_from_file[2], NA)

  data <- readr::read_csv(file)

  data <- data %>%
    separate(host_location, into = c("city", "state"), sep = ", ", remove = FALSE)

  data$city <- city_from_file[1]
  data$state <- city_state_split[2]
  # sampled_data <- data %>% slice_sample(n = 100, replace = TRUE)

  data_list[[file]] <- data

}

# add city, state
# add columns for date, month and year
#
cleaned_data_list <- list()

for (file in names(data_list)) {
  data_list[[file]] <- data_list[[file]] %>%
    dplyr::mutate(neighbourhood_cleansed = as.character(neighbourhood_cleansed))

  cleaned_data_list[[file]] <- data_list[[file]]
}


final_df <- dplyr::bind_rows(cleaned_data_list)


# Dataframe format
airbnb_df <- data.frame(final_df)
airbnb_df <- airbnb_df %>% filter(state == "Washington")
dim(airbnb_df)

# Response and Predictor variables
response <- airbnb_df$price
predictors <- subset(airbnb_df, select = -price)


response <- gsub("[$,]", "", response)

predictors <- predictors[which(!is.na(response)), ]
response <- as.numeric(response[!is.na(response)])
response <- data.frame(response)
responseProcess <- preProcess(response, method = c("YeoJohnson"))

response <- predict(responseProcess, response)

hist(response, breaks = 100, col = 'grey', main = "Distribution of Airbnb Homestay
Price")

boxplot(log(response), main = "Boxplot of Airbnb Homestay Price",
        xlab = "Price", ylab = "Price (Log Scale)")
```

```
# Cont or Cat columns
continuous_columns <- names(predictors)[sapply(predictors, is.numeric)]
categorical_columns <- names(predictors)[sapply(predictors, function(col)
is.factor(col) | is.character(col ))]
logical_columns <- names(predictors)[sapply(predictors, function(col)
is.logical(col))]
date_columns <- names(predictors)[sapply(predictors, function(col) inherits(col,
"Date"))]

length(continuous_columns)
length(categorical_columns)

length(logical_columns)
length(date_columns)

length(predictors)


barplot(table(predictors$host_response_time   ))

barplot(table(predictors$room_type))


barplot(table(predictors$state), horiz = TRUE)

# Predictors with description that can be removed as it cannot be 1-hot-encoded

predictors <- subset(predictors, select = -c(id, listing_url, scrape_id,
last_scraped, source,
                                             name,
description,neighborhood_overview,picture_url,
                                             host_id, host_location,
host_verifications,
                                             neighbourhood, neighbourhood_cleansed,
                                             neighbourhood_group_cleansed,
calendar_updated,
                                             bathrooms_text, calendar_last_scraped,
                                             host_url, host_name, host_about,
license,
                                             host_thumbnail_url, host_picture_url,
                                             host_neighbourhood,property_type,
                                             amenities, host_listings_count,

minimum_minimum_nights,maximum_minimum_nights,

minimum_maximum_nights,maximum_maximum_nights,
minimum_nights_avg_ntm,maximum_nights_avg_ntm,

availability_60,availability_90,availability_365,
                                             number_of_reviews_ltm,
number_of_reviews_l30d,

review_scores_rating,review_scores_accuracy,
                                             review_scores_cleanliness,
review_scores_checkin,

review_scores_communication,review_scores_location,

calculated_host_listings_count_entire_homes,

calculated_host_listings_count_private_rooms,

calculated_host_listings_count_shared_rooms
))



## Converting the columns into Date format


predictors$host_since <- as.Date(predictors$host_since)
predictors$first_review <- as.Date(predictors$first_review)
predictors$last_review <- as.Date(predictors$last_review)

# creating is_weekend_or_not column using date/month/year
# predictors$weekend <- wday(predictors$date, label = TRUE) %in% c("Sat", "Sun")
# not needed as host_since, first_review and last_review does
# not make sense to have is_weekend or not

# Creating columns for day, month and year
predictors$host_since_day <- day(predictors$host_since)
predictors$host_since_month <- month(predictors$host_since)
predictors$host_since_year <- year(predictors$host_since)

predictors$first_review_day <- day(predictors$first_review)
predictors$first_review_month <- month(predictors$first_review)
predictors$first_review_year <- year(predictors$first_review)

predictors$last_review_day <- day(predictors$last_review)
predictors$last_review_month <- month(predictors$last_review)
predictors$last_review_year <- year(predictors$last_review)

# Removing date format columns
```

```
predictors <- subset(predictors, select = -c(host_since,first_review, last_review))


# Changing True/False variables to 1/0

logical_to_num <- function(col){
  ifelse(col, 1, 0)
}

predictors$host_is_superhost <- logical_to_num(predictors$host_is_superhost)
predictors$host_has_profile_pic <- logical_to_num(predictors$host_has_profile_pic)
predictors$host_identity_verified <-
logical_to_num(predictors$host_identity_verified)
predictors$has_availability <- logical_to_num(predictors$has_availability)
predictors$instant_bookable <- logical_to_num(predictors$instant_bookable)

predictors <- predictors %>%
  mutate(across(where(is.character), ~ na_if(., "N/A")))


# Remove percentage in columns
predictors$host_response_rate <- as.numeric(gsub("%", "",
predictors$host_response_rate))
predictors$host_acceptance_rate <- as.numeric(gsub("%", "",
predictors$host_acceptance_rate))


predictors$bedrooms <- as.factor(predictors$bedrooms)
predictors$host_since_month <- as.factor(predictors$host_since_month)
predictors$first_review_month <- as.factor(predictors$first_review_month)
predictors$last_review_month <- as.factor(predictors$last_review_month)


# Dummy variables for categorical data

predictors <- subset(predictors, select = -city)

dummy_var <-
dummyVars("~host_response_time+room_type+bedrooms+host_since_month+first_review_month
+last_review_month",
                       data=predictors, fullRank=TRUE)

add_dummy <- data.frame(predict(dummy_var, newdata=predictors))


predictors <- cbind(predictors, add_dummy)
predictors <- subset(predictors, select = -c(
  state, host_response_time,room_type,bedrooms,
  host_since_month, first_review_month,last_review_month))

#  knnimpute (preProcess function)
system.time({
  Im <- preProcess(predictors, method = c("knnImpute"))
  predictors_im <- predict(Im, predictors)
})

write.csv(predictors_im, "E:\\Coursework\\predictive\\project\\preds_wash.csv",
row.names = FALSE)
predictors_im <- read.csv("E:\\Coursework\\predictive\\project\\preds_wash.csv")

# Skewness
skewValues <- apply(predictors_im, 2, skewness)

skew_or_not <- function(x) {
  if (x > 1 || x < -1){
    return('Highly Skewed')
  }
  else if ((x > 0.5 && x < 1) || (x < -0.5 && x > -1)){
    return('Moderately Skewed')
  }
  else{
    return('Approx. Symmetric')
  }
}

skewValues <- data.frame(skewValues)



skewValues$Skewness <- sapply(!is.na(skewValues$skewValues), skew_or_not)

kable(skewValues, caption = "Skewness Analysis Table", format = "markdown")

kable(skewValues, format = "html", table.attr = "class='table table-bordered'") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))

# skew_df



# Near zero variance predictors
zero_var <- nearZeroVar(predictors_im)
predictors_im <- predictors_im[, -zero_var]
```

```r
# Correlation
pred_corr <- cor(predictors_im)

par(mfrow=c(1, 1), mar=c(2, 2, 2, 2))  # Modify margins as needed
windows(width = 12, height = 10)
# Create the correlation plot
corrplot(pred_corr, order = 'hclust',
         col = NULL, bg = "white",
         tl.col = "transparent",
         addgrid.col = "grey")


dim(pred_corr)

highCorr <- findCorrelation(pred_corr, cutoff = .8)

filter_pred <- predictors_im[, -highCorr]

# YeoJohnson Transformation

pred_yj <- preProcess(filter_pred, method = c("YeoJohnson"))

pred_trans <- predict(pred_yj, filter_pred)

pred_yjnzv <- preProcess(predictors_im, method = c("YeoJohnson"))

pred_transnzv <- predict(pred_yjnzv, predictors_im)


# SpatialSignTransformation


predictors_spatial_sign <- spatialSign(pred_trans)

predictors_spatial_signnzv <- spatialSign(pred_transnzv)


# Training and Testin Data Split


set.seed(37)

dataPartition <- createDataPartition(response[, 1], p = 0.8, list = FALSE)

trainx <- predictors_spatial_sign[dataPartition, ]
trainy <- response[dataPartition]

testx <- predictors_spatial_sign[-dataPartition, ]
```

```
testy <- response[-dataPartition]

trainx_nzv <- predictors_spatial_signnzv[dataPartition]
trainy_nzv <- response[dataPartition]

testx_nzv <- predictors_spatial_signnzv[-dataPartition, ]
testy_nzv <- response[-dataPartition]


ctrl <- trainControl(method = "cv", number = 10,
                     verboseIter = TRUE,
                     allowParallel = TRUE)

# Linear Models

# Ordinary Linear Regression

trainOLR <- train(trainx, trainy,
                  method = "lm",
                  # preProcess = c("center", "scale"),
                  metric = "RMSE",
                  trControl = ctrl)

trainOLR


testOLR <- predict(trainOLR, testx)

defaultSummary(data.frame(obs = testy, pred = testOLR))


# Ridge Regression


ridgeGrid <- data.frame(.lambda = seq(0, 1, length = 15))

trainRidge <- train(trainx, trainy, method = "ridge", tuneGrid = ridgeGrid,
                    # preProcess = c("center", "scale", "BoxCox", "spatialSign"),
                    metric = "RMSE",
                    trControl = ctrl)

trainRidge
plot(trainRidge)

testRidge <- predict(trainRidge, testx)

defaultSummary(data.frame(obs = testy, pred = testRidge))
```

```
# Lasso Model

lassoGrid <- expand.grid(fraction = seq(0.01, 1, length = 10))
trainLasso <- train(trainx, trainy, method = "lasso", tuneGrid = lassoGrid,
                    # preProcess = c("center", "scale", "BoxCox", "spatialSign") ,
                    metric = "RMSE",
                    trControl = ctrl)


trainLasso
plot(trainLasso)

testLasso <- predict(trainLasso, testx)

defaultSummary(data.frame(obs = testy, pred = testLasso))

# Elastic Net Model

enetGrid <- expand.grid(.lambda = c(0, 0.1, .1), .fraction = seq(.05, 1, length =
20))
trainEnet <- train(trainx, trainy, method = "enet", tuneGrid = enetGrid,
                   # preProcess = c("center", "scale", "BoxCox", "spatialSign") ,
                   metric = "RMSE",
                   trControl = ctrl)

trainEnet
plot(trainEnet)


testEnet <- predict(trainEnet, testx)

defaultSummary(data.frame(obs = testy, pred = testEnet))

# PLS


trainPLS <- train(trainx, trainy, method = "pls", tuneLength = 20,
                  trControl = ctrl, metric = "RMSE")
trainPLS

plot(trainPLS)

testPLS <- predict(trainPLS, testx)

defaultSummary(data.frame(obs = testy, pred = testPLS))
```

```
# Non-Linear Models


# KNN
system.time({
  trainKNN <- train(x = trainx, y = trainy,
                    method = "knn",
                    # preProcess = c("center", "scale", "BoxCox", "spatialSign"),
                    metric = "RMSE",
                    tuneLength = 20)
})
trainKNN

plot(trainKNN)

testKNN <- predict(trainKNN, newdata = testx)

postResample(pred = testy, obs = testKNN)

# MARS
system.time({
  marsGrid <- expand.grid(.degree = 1:3, .nprune = 1:40)

  trainMARS <- train(trainx, trainy,
                     method = "earth",
                     tuneGrid = marsGrid,
                     # preProcess = c("center", "scaling", "BoxCox", "spatialSign"),
                     trControl = ctrl,
                     metric = "RMSE")
})
trainMARS
plot(trainMARS)

testMARS <- predict(trainMARS, newdata = testx)

postResample(pred = testMARS, obs = testy)

# Neural Network
library(doParallel)
num_cores <- parallel::detectCores()

# Create a cluster
cl <- makeCluster(num_cores - 2)  # Use one less core to avoid freezing your system

# Register the parallel backend
registerDoParallel(cl)
```

```
nnetGrid <- expand.grid(.decay = c(0, .1, 1, 2),
                        .size = c(1:10),
                        .bag = FALSE)
system.time({
  trainNN <- train(x = trainx, y = trainy,
                   method = "avNNet",
                   tuneGrid = nnetGrid,
                   trControl = ctrl,
                   metric = "RMSE",
                   preProc = c("center", "scale"),
                   linout = TRUE,
                   trace = FALSE,
                   MaxNWts = 10 * (ncol(trainx) + 1) + 10 + 1,
                   maxit = 200)
})
trainNN

plot(trainNN)

testNN <- predict(trainNN, newdata = testx)

postResample(pred = testNN, obs = testy)

# SVM
# Define a grid of hyperparameters for SVM
library(kernlab)
sigmaRangeReduced <- sigest(as.matrix(trainx))
svmGrid <- expand.grid(
  .sigma = sigmaRangeReduced[1],
  .C = 2^(seq(-4, 6))
)

# parallel processing
num_cores <- parallel::detectCores()

# Create a cluster
cl <- makeCluster(num_cores - 2)  # Use one less core to avoid freezing your system

# Register the parallel backend
registerDoParallel(cl)

# Train the SVM model
system.time({
  trainSVM <- train(
    x = trainx,
    y = trainy,
    method = "svmRadial",             # SVM with RBF kernel
    metric = "RMSE",
    tuneGrid = svmGrid,               # Candidate parameter grid
    # preProcess = c("center", "scale", "BoxCox", "spatialSign"),
    tuneLength = 14,
    trControl = ctrl                  # Control parameters (e.g., cross-validation)
  )
})
# Summarize the results
summary(trainSVM)

# Plot the tuning results
plot(trainSVM)

# Make predictions on the test set
testSVM <- predict(trainSVM, newdata = testx)

# Evaluate model performance
postResample(pred = testSVM, obs = testy)




##### Exploring PCA------------------------------------------------------------:

dim(pred_transnzv)


pca_preds <- preProcess(pred_transnzv, method = c("pca"))
pca_predicted_pred <- predict(pca_preds, pred_transnzv)


hist(pca_predicted_pred)

# PCA Plotting
num_components <- ncol(pca_predicted_pred)

# We can retrieve the PCA model matrix to calculate the variances
pca_model_matrix <- as.matrix(pca_predicted_pred)

# Calculate the variance for each principal component
variance_explained <- apply(pca_model_matrix, 2, var)

# Normalize to get the proportion of variance explained
explained_variance <- variance_explained / sum(variance_explained)


par(mfrow=c(1,1))
# Plot cumulative explained variance
```

```
plot(cumsum(explained_variance), type = "o",
     xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     main = "Cumulative Explained Variance by Principal Components")

# Only for PCA Plotting without matrix multiplication
pca_result <- prcomp(pred_transnzv, center = TRUE, scale. = TRUE)
summary(pca_result)  # Check variance explained
sdev <- pca_result$sdev  # Get standard deviations

par(mfrow = c(1,1))

plot(cumsum(pca_result$sdev^2 / sum(pca_result[1:10]$sdev^2)), type="o",
     xlab = "PCA", ylab = "Variance", main = 'Variance explained by PCAs')


pca_result <- prcomp(pred_transnzv, center = TRUE, scale. = TRUE)
summary(pca_result)  # Check variance explained
sdev <- pca_result$sdev  # Get standard deviations



# Plot for PCA scatter for first 10 PCAs
plot(pca_predicted_pred[1:5])

# Set up the plotting area to have 2 rows and 3 columns (for 6 plots)
par(mfrow = c(5,5))

# Loop to plot the first 6 plots
for (i in 1:25) {
  plot(pca_predicted_pred[[i]], main = paste("Plot", i))
}



# Training and Testing Data Split PCA ###############


set.seed(37)

response <- response$response

dataPartition <- createDataPartition(response, p = 0.8, list = FALSE)

trainx_pca <- pca_predicted_pred[dataPartition, ]
trainy_pca <- response[dataPartition]

testx_pca <- pca_predicted_pred[-dataPartition, ]
```

```
testy_pca <- response[-dataPartition]


ctrl <- trainControl(method = "cv", number = 10,
                     verboseIter = TRUE,
                     allowParallel = TRUE)

# Linear Models

# Ordinary Linear Regression

trainOLR_pc <- train(trainx_pca, trainy_pca,
                  method = "lm",
                  metric = "RMSE",
                  # preProcess = c("center", "scale"),
                  trControl = ctrl)

trainOLR_pc


testOLR_pc <- predict(trainOLR_pc, testx_pca)

defaultSummary(data.frame(obs = testy_pca, pred = testOLR_pc))


# PLS
trainpls_pc <- train(trainx_pca, trainy_pca, method="pls",tuneLength = 20,
trControl=ctrl,
                  metric = "RMSE")

trainpls_pc

plot(trainpls_pc)
testpls_pc <- predict(trainpls_pc, testx_pca)

defaultSummary(data.frame(obs = testy_pca, pred = testpls_pc))

# Ridge Regression


ridgeGrid <- data.frame(.lambda = seq(0, 1, length = 15))

trainRidge_pc <- train(trainx_pca, trainy_pca, method = "ridge", tuneGrid =
ridgeGrid,
                    metric = "RMSE",
                    # preProcess = c("center", "scale", "BoxCox", "spatialSign"),
                    trControl = ctrl)
```

```
trainRidge_pc
plot(trainRidge_pc)

testRidge_pc <- predict(trainRidge_pc, testx_pca)

defaultSummary(data.frame(obs = testy_pca, pred = testRidge_pc))


# Lasso Model

lassoGrid <- expand.grid(fraction = seq(0.01, 1, length = 20))
trainLasso_pc <- train(trainx_pca, trainy_pca, method = "lasso", tuneGrid = lassoGrid
,
                    metric = "RMSE",
                    trControl = ctrl)


trainLasso_pc
plot(trainLasso_pc)

testLasso_pc <- predict(trainLasso_pc, testx_pca)

defaultSummary(data.frame(obs = testy_pca, pred = testLasso_pc))

# Elastic Net Model

enetGrid <- expand.grid(.lambda = c(0, 0.1, .1), .fraction = seq(.05, 1, length =
20))
trainEnet_pc <- train(trainx_pca, trainy_pca, method = "enet", tuneGrid = enetGrid,
                   metric = "RMSE",
                   trControl = ctrl)

trainEnet_pc
plot(trainEnet_pc)


testEnet_pc <- predict(trainEnet_pc, testx_pca)

defaultSummary(data.frame(obs = testy_pca, pred = testEnet_pc))




# Non-Linear Models


# KNN
system.time({
  trainKNN_pc <- train(x = trainx_pca, y = trainy_pca,
                    method = "knn",
                    metric = "RMSE",
                    # preProcess = c("center", "scale", "BoxCox", "spatialSign"),
                    tuneLength = 10,
                    trControl=ctrl)
})
trainKNN_pc

plot(trainKNN_pc)

testKNN_pc <- predict(trainKNN_pc, newdata = testx_pca)

postResample(pred = testy_pca, obs = testKNN_pc)

# MARS
system.time({
  marsGrid <- expand.grid(.degree = 1:3, .nprune = 1:40)

  trainMARS_pc <- train(trainx_pca, trainy_pca,
                     method = "earth",
                     tuneGrid = marsGrid,
                     metric = "RMSE",
                     # preProcess = c("center", "scaling", "BoxCox", "spatialSign"),
                     trControl = ctrl)
})
trainMARS_pc
plot(trainMARS_pc)

testMARS_pc <- predict(trainMARS_pc, newdata = testx_pca)

postResample(pred = testMARS_pc, obs = testy_pca)

# Neural Network
library(doParallel)
num_cores <- parallel::detectCores()

# Create a cluster
cl <- makeCluster(num_cores - 2)  # Use one less core to avoid freezing your system

# Register the parallel backend
registerDoParallel(cl)

nnetGrid <- expand.grid(.decay = c(0, .1, 1, 2),
                       .size = c(1:10))
system.time({
  trainNN_pc <- train(x = trainx_pca, y = trainy_pca,
                   method = "nnet",
                   tuneGrid = nnetGrid,
                   trControl = ctrl,
                   metric = "RMSE",
                   # preProc = c("center", "scale", "BoxCox", "spatialSign"),
                   linout = TRUE,
                   trace = FALSE,
                   MaxNWts = 10 * (ncol(trainx_pca) + 1) + 10 + 1,
                   maxit = 200)
})
trainNN_pc

plot(trainNN_pc)

testNN_pc <- predict(trainNN_pc, newdata = testx_pca)

postResample(pred = testNN_pc, obs = testy_pca)

# SVM
# Define a grid of hyperparameters for SVM
library(kernlab)
sigmaRangeReduced <- sigest(as.matrix(trainx_pca))
svmGrid <- expand.grid(
  .sigma = sigmaRangeReduced[1],
  .C = 2^(seq(-4, 6))
)

# parallel processing
num_cores <- parallel::detectCores()

# Create a cluster
cl <- makeCluster(num_cores - 2)  # Use one less core to avoid freezing your system

# Register the parallel backend
registerDoParallel(cl)

# Train the SVM model
system.time({
  trainSVM_pc <- train(
    x = trainx_pca,
    y = trainy_pca,
    method = "svmRadial",             # SVM with RBF kernel
    metric = "RMSE",
    tuneGrid = svmGrid,               # Candidate parameter grid
    # preProcess = c("center", "scale", "BoxCox", "spatialSign"),
    tuneLength = 14,
    trControl = ctrl                  # Control parameters (e.g., cross-validation)
  )
})
```

```
# Summarize the results
trainSVM_pc

# Plot the tuning results
plot(trainSVM_pc)

plot(trainSVM_pc, main='Tuning Parameter Plot')
ggplot(trainSVM_pc$results,aes(x = C, y = RMSE)) +
  geom_line() +
  geom_point() +
  scale_x_log10() +
  labs(title = "Tuning Parameter Plot",
       x = "Cost C",
       y = "RMSE (Cross-Validation)")

# Make predictions on the test set
testSVM_pc <- predict(trainSVM_pc, newdata = testx_pca)

# Evaluate model performance
postResample(pred = testSVM_pc, obs = testy_pca)


####### PCA TABLE

results_pc <- data.frame(
  Model = c("OLR", "Ridge","PLS", "Lasso", "Enet", "KNN", "MARS", "NN", "SVM"),
  Best_Tuning_Parameter = c(
    paste("Intercept = ", trainOLR_pc$bestTune$intercept), # OLR
    paste("Lambda = ", trainRidge_pc$bestTune$lambda),  # Ridge
    paste("Components = ", trainpls_pc$bestTune$ncomp),
    paste("Fraction = ", trainLasso_pc$bestTune$fraction),  # Lasso
    paste("Fraction = ", trainEnet_pc$bestTune$fraction, ", Lambda = ",
trainEnet_pc$bestTune$lambda),  # Enet
    paste("k = ", trainKNN_pc$bestTune$k), # KNN
    paste("nPrune = ", trainMARS_pc$bestTune$nprune, ", Degree = ",
trainMARS_pc$bestTune$degree), # MARS
    paste("size = ", trainNN_pc$bestTune$size, ", decay = ",
trainNN_pc$bestTune$decay,
          ", bag = FALSE", trainNN_pc$bestTune$bag), # NN
    paste("Sigma = ", trainSVM_pc$bestTune$sigma, ", C = ", trainSVM_pc$bestTune$C)
  ),
  Training_RMSE = c(
    trainOLR_pc$results$RMSE[which.max(trainOLR_pc$results$RMSE)],
    trainRidge_pc$results$RMSE[which.max(trainRidge_pc$results$RMSE)],
    trainpls_pc$results$RMSE[which.max(trainpls_pc$results$RMSE)],
    trainLasso_pc$results$RMSE[which.max(trainLasso_pc$results$RMSE)],
    trainEnet_pc$results$RMSE[which.max(trainEnet_pc$results$RMSE)],
    trainKNN_pc$results$RMSE[which.max(trainKNN_pc$results$RMSE)],
    trainMARS_pc$results$RMSE[which.max(trainMARS_pc$results$RMSE)],
    trainNN_pc$results$RMSE[which.max(trainNN_pc$results$RMSE)],
    trainSVM_pc$results$RMSE[which.max(trainSVM_pc$results$RMSE)]
  ),
  Testing_RMSE = c(
    unname(postResample(pred = testOLR_pc, obs = testy_pca)[1]),
    unname(postResample(pred = testRidge_pc, obs = testy_pca)[1]),
    unname(postResample(pred = testpls_pc, obs = testy_pca)[1]),
    unname(postResample(pred = testLasso_pc, obs = testy_pca)[1]),
    unname(postResample(pred = testEnet_pc, obs = testy_pca)[1]),
    unname(postResample(pred = testKNN_pc, obs = testy_pca)[1]),
    unname(postResample(pred = testMARS_pc, obs = testy_pca)[1]),
    unname(postResample(pred = testNN_pc, obs = testy_pca)[1]),
    unname(postResample(pred = testSVM_pc, obs = testy_pca)[1])
  ),
  Training_R2 = c(
    trainOLR_pc$results$Rsquared[which.max(trainOLR_pc$results$Rsquared)],
    trainRidge_pc$results$Rsquared[which.max(trainRidge_pc$results$Rsquared)],
    trainpls_pc$results$Rsquared[which.max(trainpls_pc$results$Rsquared)],
    trainLasso_pc$results$Rsquared[which.max(trainLasso_pc$results$Rsquared)],
    trainEnet_pc$results$Rsquared[which.max(trainEnet_pc$results$Rsquared)],
    trainKNN_pc$results$Rsquared[which.max(trainKNN_pc$results$Rsquared)],
    trainMARS_pc$results$Rsquared[which.max(trainMARS_pc$results$Rsquared)],
    trainNN_pc$results$Rsquared[which.max(trainNN_pc$results$Rsquared)],
    trainSVM_pc$results$Rsquared[which.max(trainSVM_pc$results$Rsquared)]
  ),
  Testing_R2 = c(
    unname(postResample(pred = testOLR_pc, obs = testy_pca)[2]),
    unname(postResample(pred = testRidge_pc, obs = testy_pca)[2]),
    unname(postResample(pred = testpls_pc, obs = testy_pca)[2]),
    unname(postResample(pred = testLasso_pc, obs = testy_pca)[2]),
    unname(postResample(pred = testEnet_pc, obs = testy_pca)[2]),
    unname(postResample(pred = testKNN_pc, obs = testy_pca)[2]),
    unname(postResample(pred = testMARS_pc, obs = testy_pca)[2]),
    unname(postResample(pred = testNN_pc, obs = testy_pca)[2]),
    unname(postResample(pred = testSVM_pc, obs = testy_pca)[2])
  )
)
```

```
kable(results_pc,
      col.names = c("Model", "Best Tuning Parameter", "Training RMSE", "Testing
RMSE", "Training R2", "Testing R2"),
      caption = "Model Performance Summary") %>%
  kable_styling(full_width = FALSE, position = "left") %>%
  row_spec(0, bold = TRUE, background = "#f2f2f2") %>%  # Header row
  column_spec(1, bold = TRUE)
```

```
# Table Summary



results <- data.frame(
  Model = c("OLR", "Ridge", "Lasso", "Enet", "PLS", "KNN", "MARS", "NN", "SVM"),
  Best_Tuning_Parameter = c(
    paste("Intercept = ", trainOLR$bestTune$intercept), # OLR
    paste("Lambda = ", trainRidge$bestTune$lambda),  # Ridge
    paste("Fraction = ", trainLasso$bestTune$fraction),  # Lasso
    paste("Fraction = ", trainEnet$bestTune$fraction, ", Lambda = ",
trainEnet$bestTune$lambda),  # Enet
    paste("ncomp = ", trainPLS$bestTune$ncomp), # PLS
    paste("k = ", trainKNN$bestTune$k), # KNN
    paste("nPrune = ", trainMARS$bestTune$nprune, ", Degree = ",
trainMARS$bestTune$degree), # MARS
    paste("size = ", trainNN$bestTune$size, ", decay = ", trainNN$bestTune$decay,
          ", bag = ", trainNN$bestTune$bag), # NN
    paste("Sigma = ", trainSVM$bestTune$sigma, ", C = ", trainSVM$bestTune$C) # SVM
  ),
  Training_RMSE = c(
    trainOLR$results$RMSE[which.max(trainOLR$results$RMSE)],
    trainRidge$results$RMSE[which.max(trainRidge$results$RMSE)],
    trainLasso$results$RMSE[which.max(trainLasso$results$RMSE)],
    trainEnet$results$RMSE[which.max(trainEnet$results$RMSE)],
    trainPLS$results$RMSE[which.max(trainPLS$results$RMSE)],
    trainKNN$results$RMSE[which.max(trainKNN$results$RMSE)],
    trainMARS$results$RMSE[which.max(trainMARS$results$RMSE)],
    trainNN$results$RMSE[which.max(trainNN$results$RMSE)],
    trainSVM$results$RMSE[which.max(trainSVM$results$RMSE)]
  ),
  Testing_RMSE = c(
    unname(postResample(pred = testOLR, obs = testy)[1]),
    unname(postResample(pred = testRidge, obs = testy)[1]),
    unname(postResample(pred = testLasso, obs = testy)[1]),
    unname(postResample(pred = testEnet, obs = testy)[1]),
    unname(postResample(pred = testPLS, obs = testy)[1]),
    unname(postResample(pred = testKNN, obs = testy)[1]),
    unname(postResample(pred = testMARS, obs = testy)[1]),
    unname(postResample(pred = testNN, obs = testy)[1]),
    unname(postResample(pred = testSVM, obs = testy)[1])
  ),
  Training_R2 = c(
    trainOLR$results$Rsquared[which.max(trainOLR$results$Rsquared)],
    trainRidge$results$Rsquared[which.max(trainRidge$results$Rsquared)],
    trainLasso$results$Rsquared[which.max(trainLasso$results$Rsquared)],
    trainEnet$results$Rsquared[which.max(trainEnet$results$Rsquared)],
    trainPLS$results$Rsquared[which.max(trainPLS$results$Rsquared)],
    trainKNN$results$Rsquared[which.max(trainKNN$results$Rsquared)],
    trainMARS$results$Rsquared[which.max(trainMARS$results$Rsquared)],
    trainNN$results$Rsquared[which.max(trainNN$results$Rsquared)],
    trainSVM$results$Rsquared[which.max(trainSVM$results$RMSE)]
  ),
  Testing_R2 = c(
    unname(postResample(pred = testOLR, obs = testy)[2]),
    unname(postResample(pred = testRidge, obs = testy)[2]),
    unname(postResample(pred = testLasso, obs = testy)[2]),
    unname(postResample(pred = testEnet, obs = testy)[2]),
    unname(postResample(pred = testPLS, obs = testy)[2]),
    unname(postResample(pred = testKNN, obs = testy)[2]),
    unname(postResample(pred = testMARS, obs = testy)[2]),
    unname(postResample(pred = testNN, obs = testy)[2]),
    unname(postResample(pred = testSVM, obs = testy)[2])
  )
)



kable(results,
      col.names = c("Model", "Best Tuning Parameter", "Training RMSE", "Testing
RMSE", "Training R2", "Testing R2"),
      caption = "Model Performance Summary") %>%
  kable_styling(full_width = FALSE, position = "left") %>%
  row_spec(0, bold = TRUE, background = "#f2f2f2") %>%  # Header row
  column_spec(1, bold = TRUE)
```