**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.

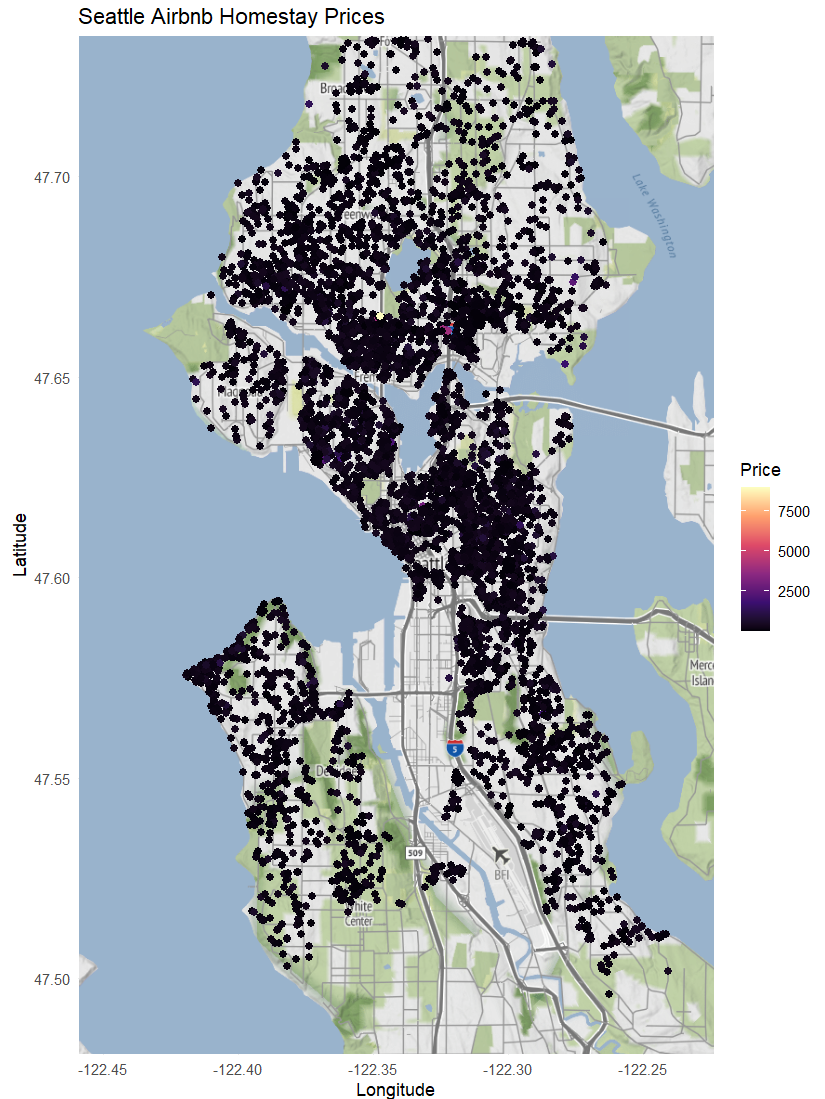


Figure 1. GeoSpatial Map of Airbnb Homestays in Seattle

**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.

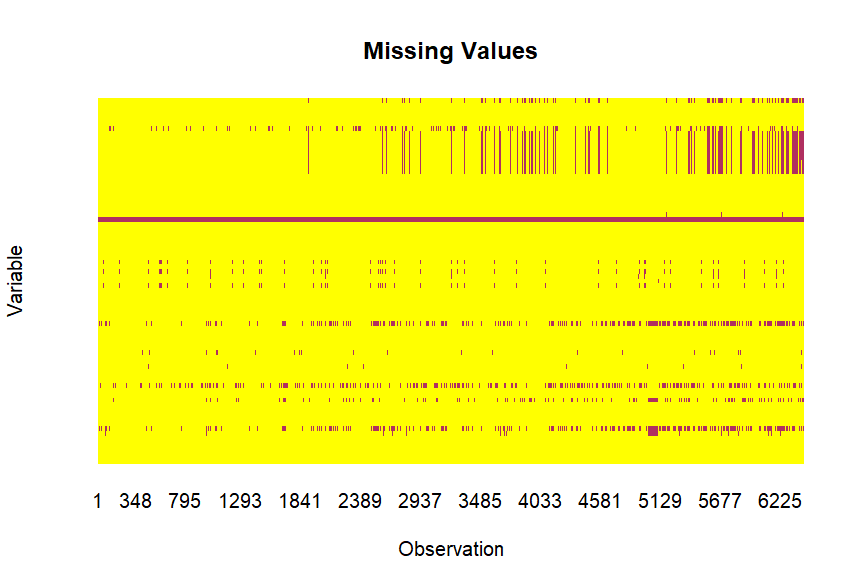


Figure 2. Missing Values Heatmap

**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.

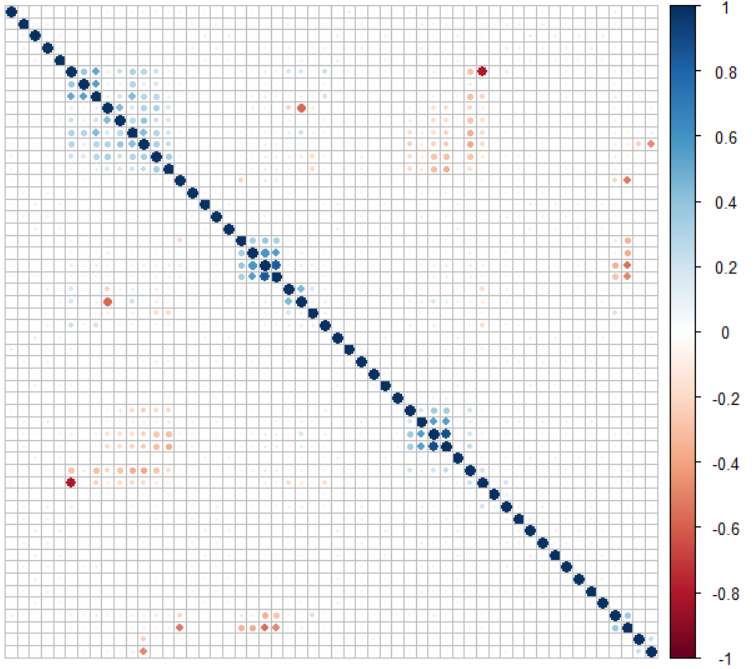


Figure 3. Correlation between Predictors

**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.

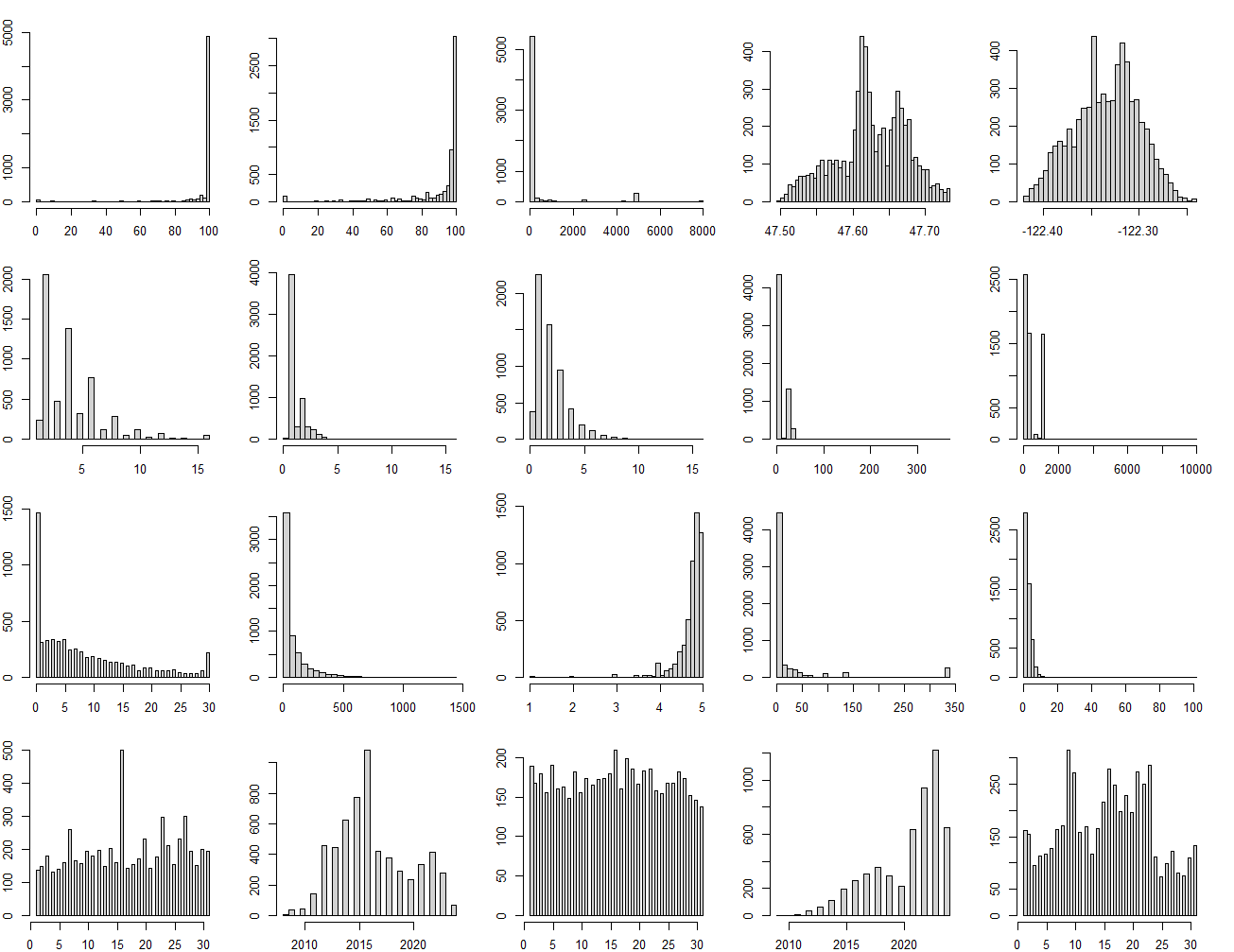


Figure 4. Before YeoJohnson Transformation

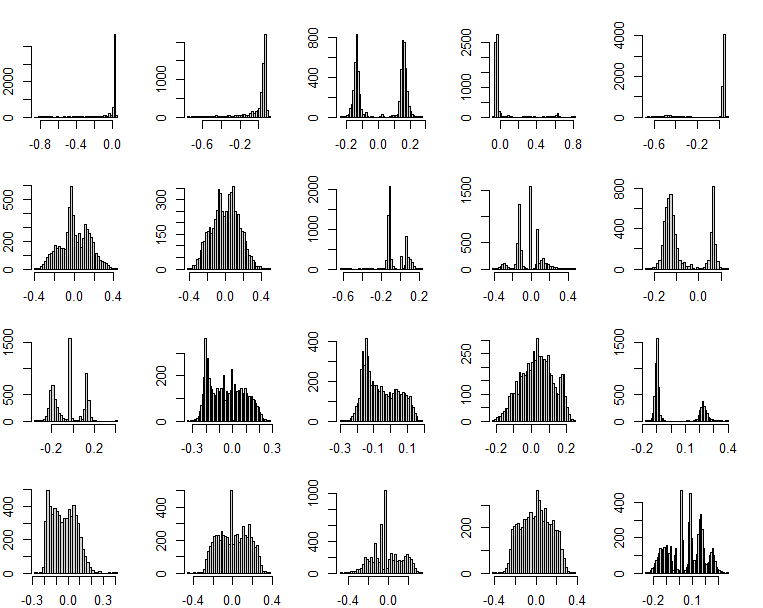


Figure 5. After YeoJohnson Transformation

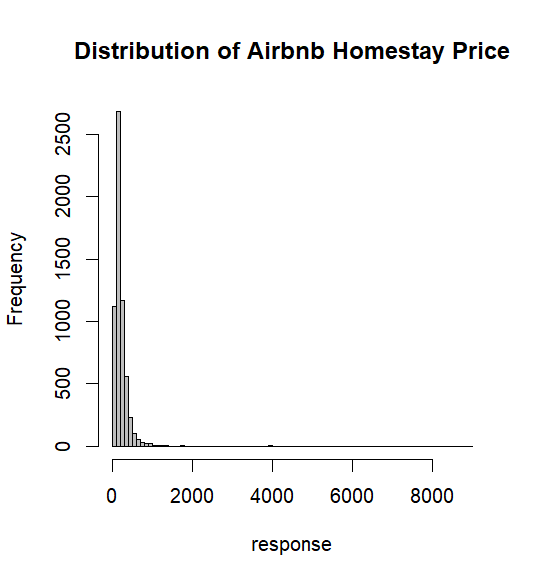
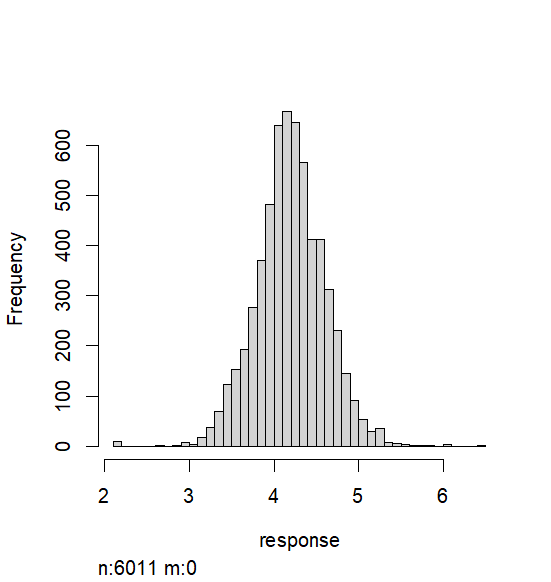
 

Figure 6. Response variable transformation

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.

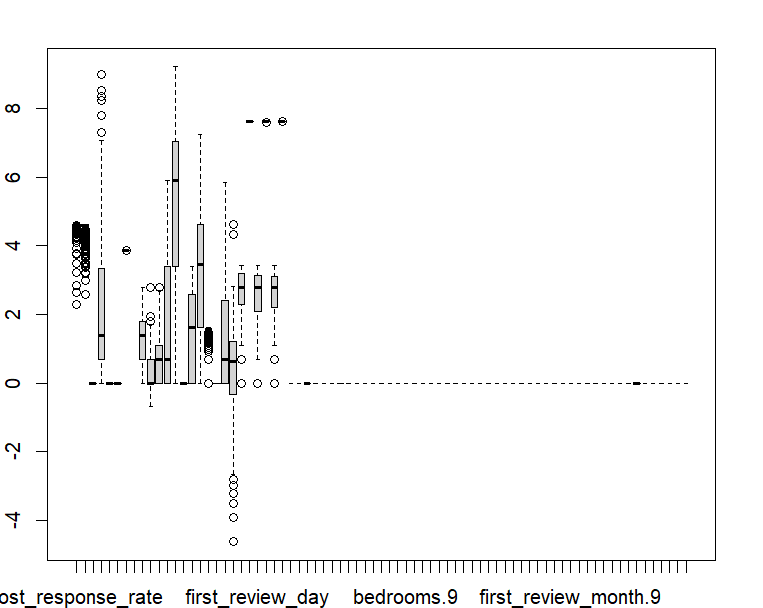


Figure 7. Before spatialSign Transformation

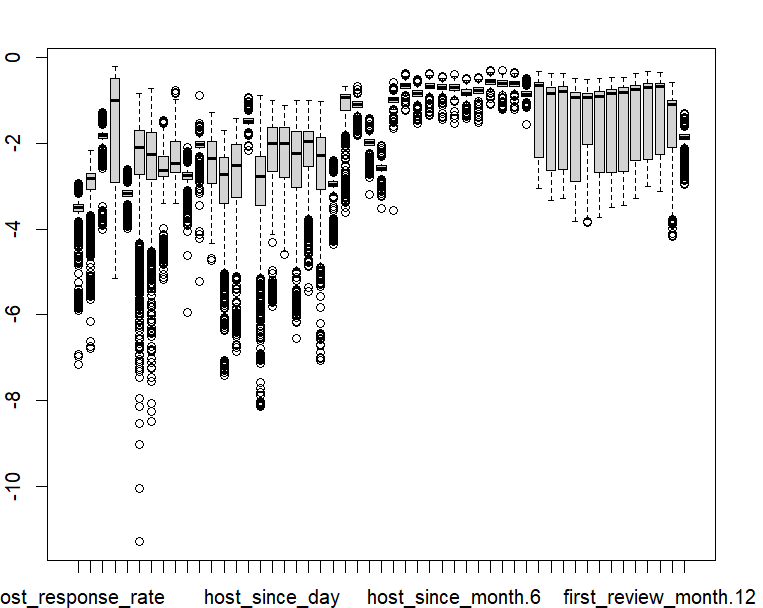


Figure 8. After spatialSign Transformation

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.

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.

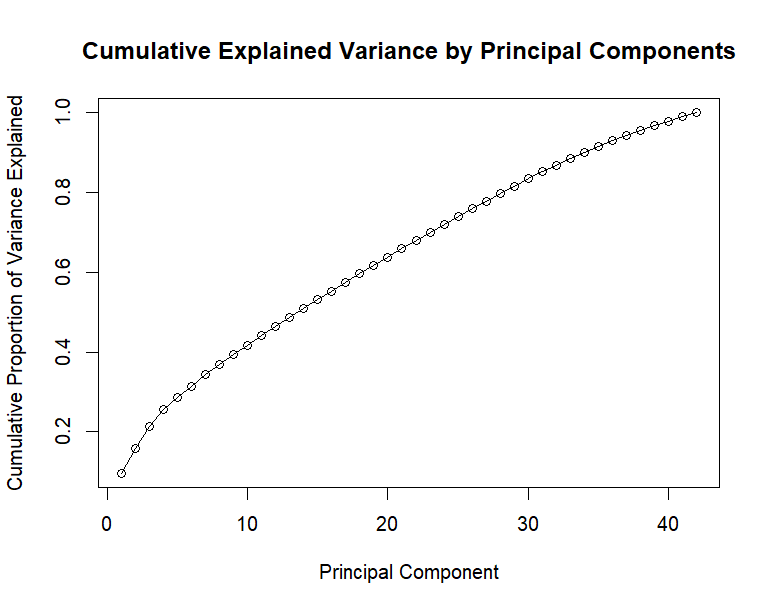


Figure 10. Cumulative Explained Variance by Principal Components

**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 R2 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.

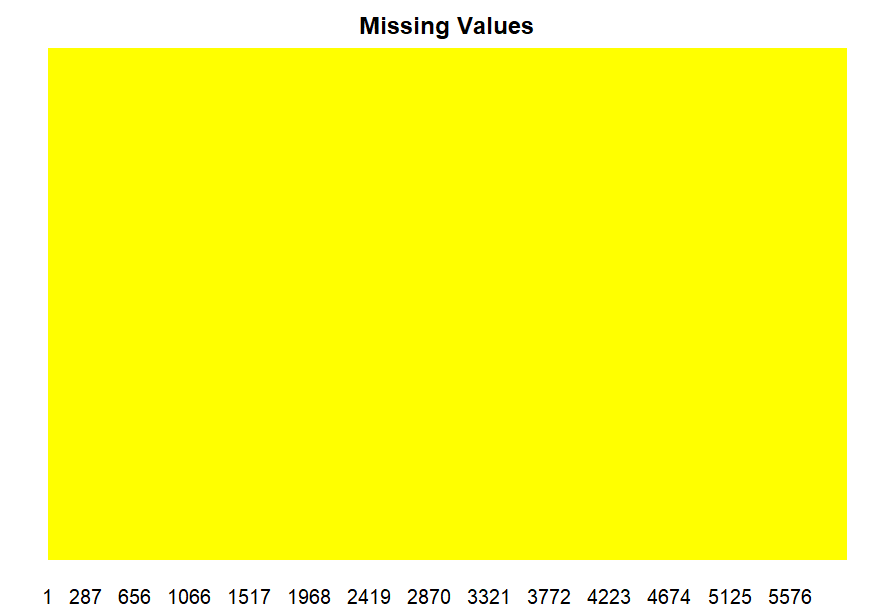
[4] Austermuhle, Martin (January 5, 2022). "D.C. To Start Restricting And Regulating Airbnb And Other Short-Term Rentals". WAMU. Archived from the original on January 5, 2022.

[5] Google Search Results

[6] [Inside Airbnb](https://insideairbnb.com/get-the-data/)

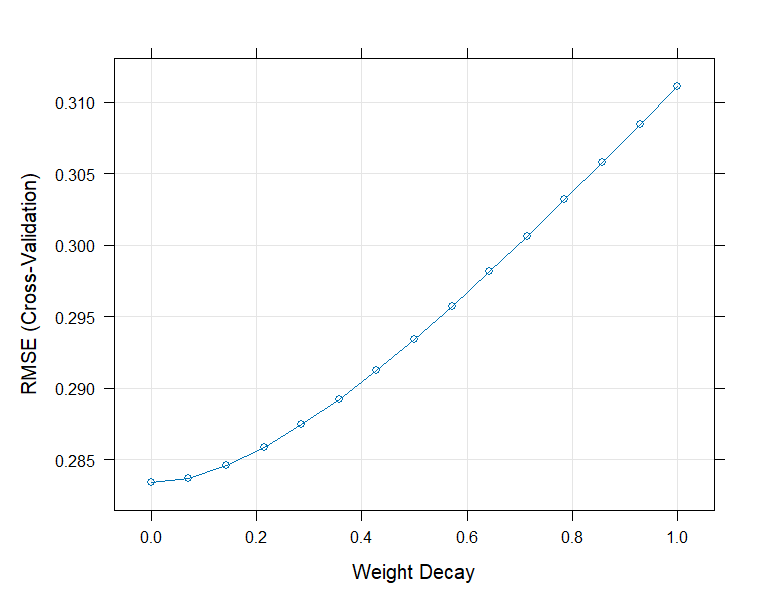
**Appendix 1. Supportive documentation for preprocessing and parameter tuning.**

**A] Missing value in predictors after kNN imputation**

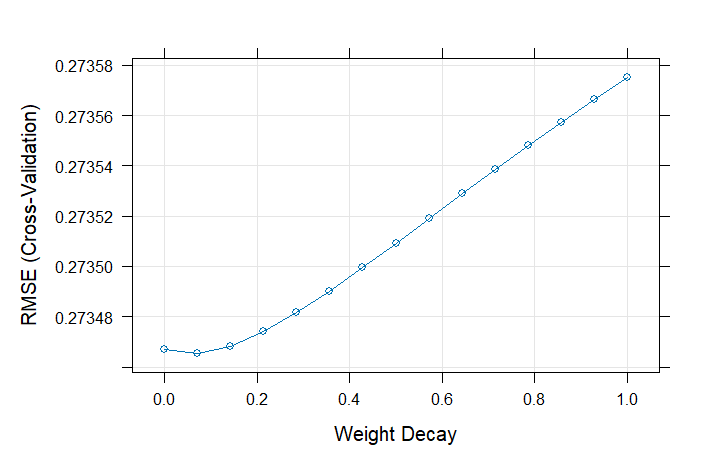


**B] Linear Models:**

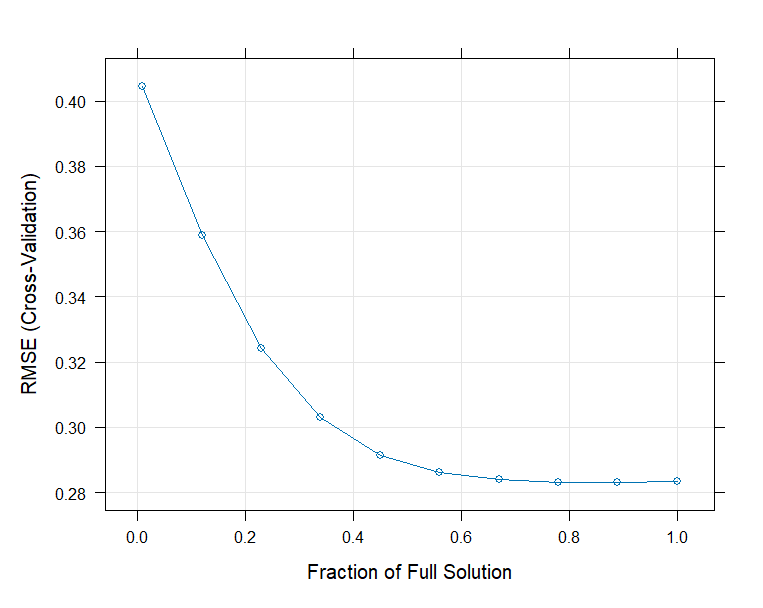
**Ridge Regression:**

****

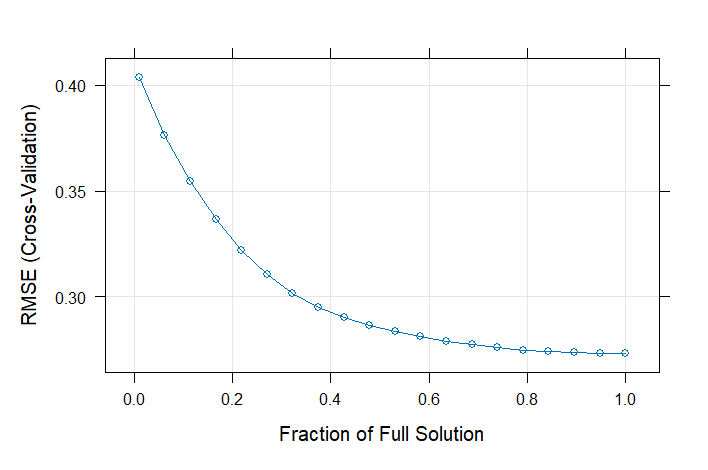
**Ridge Regression PCA:**

****

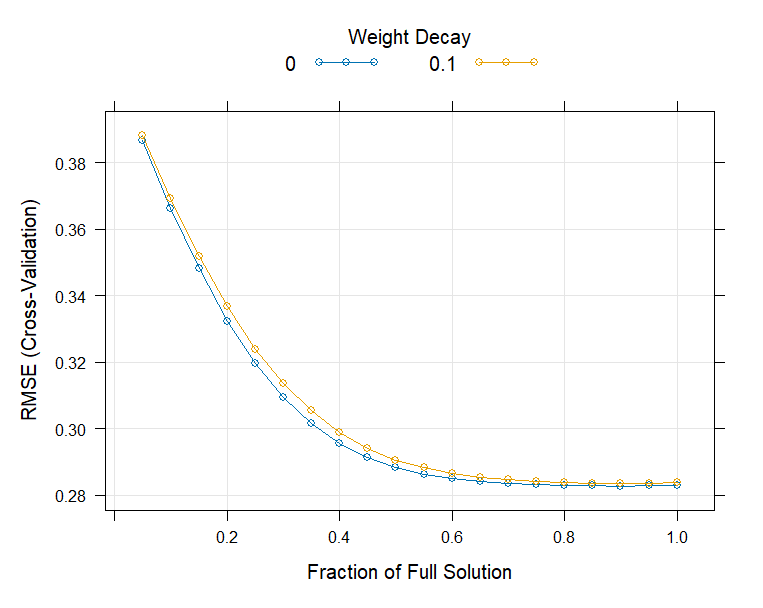
**Lasso Regression:**

****

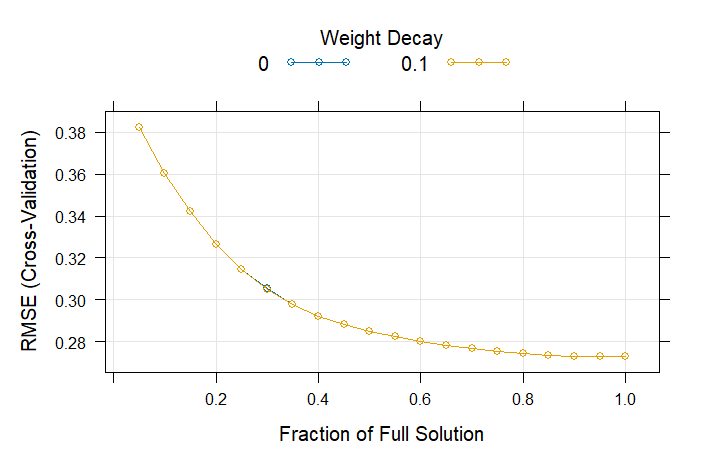
**Lasso PCA:**

****

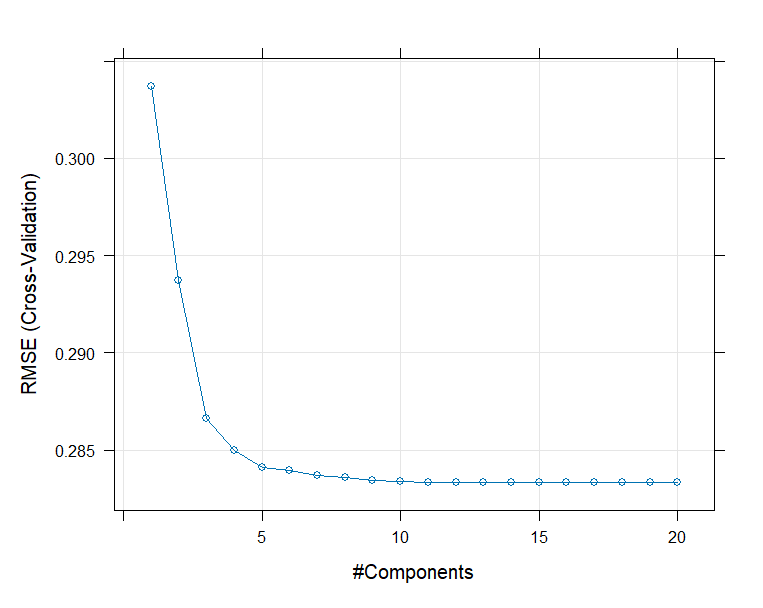
**Elastic Net Regression:**

****

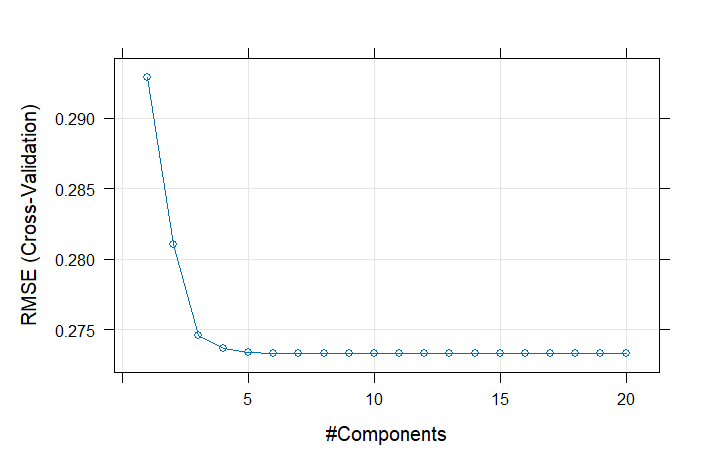
**Elastic Net Regression PCA:**

****

**Partial Least Squares Regression:**

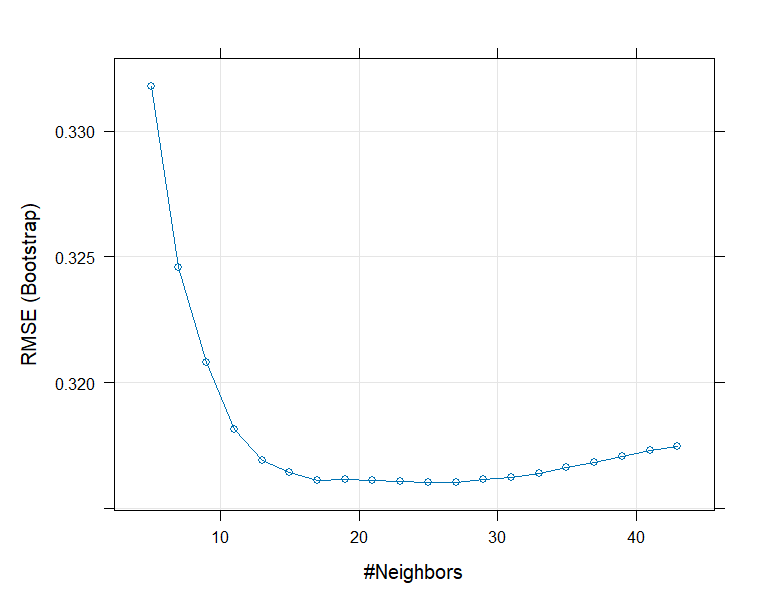
****

**Partial Least Squares PCA:**

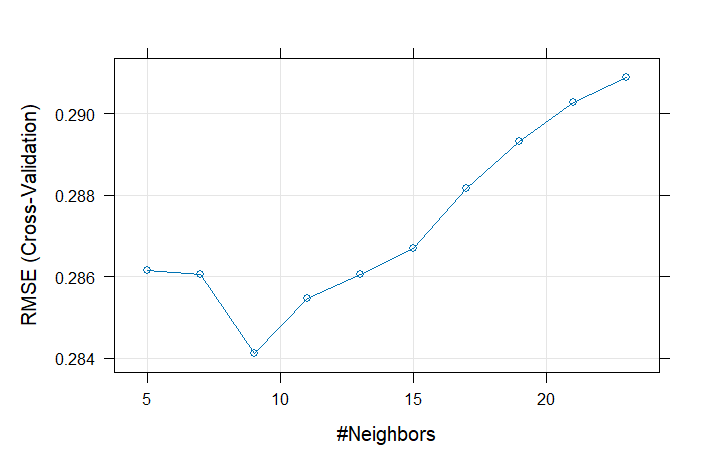
****

**Non-Linear Models:**

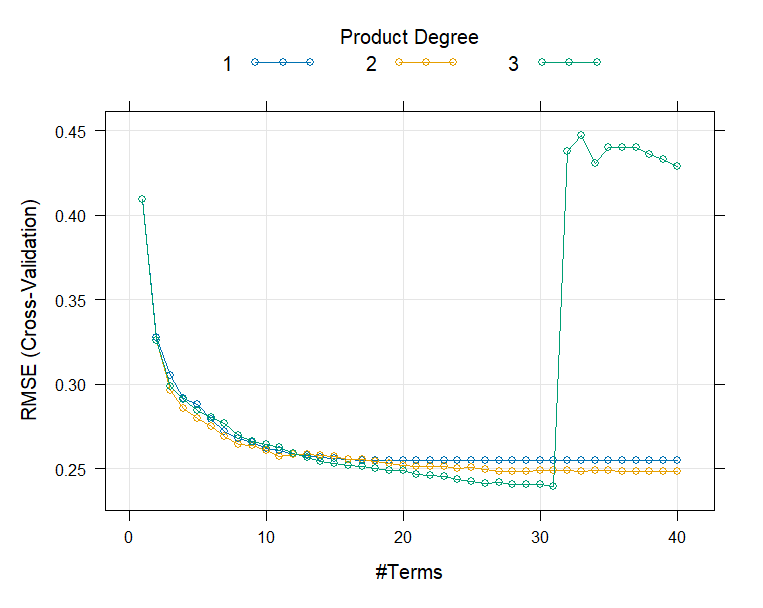
**kNN:**

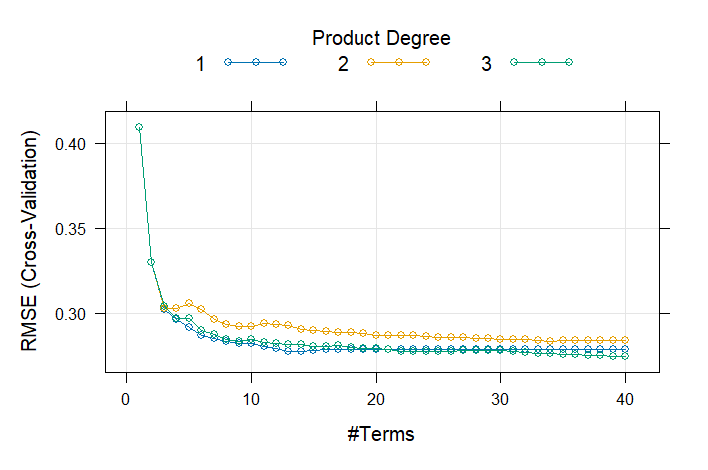
****

**KNN PCA:**

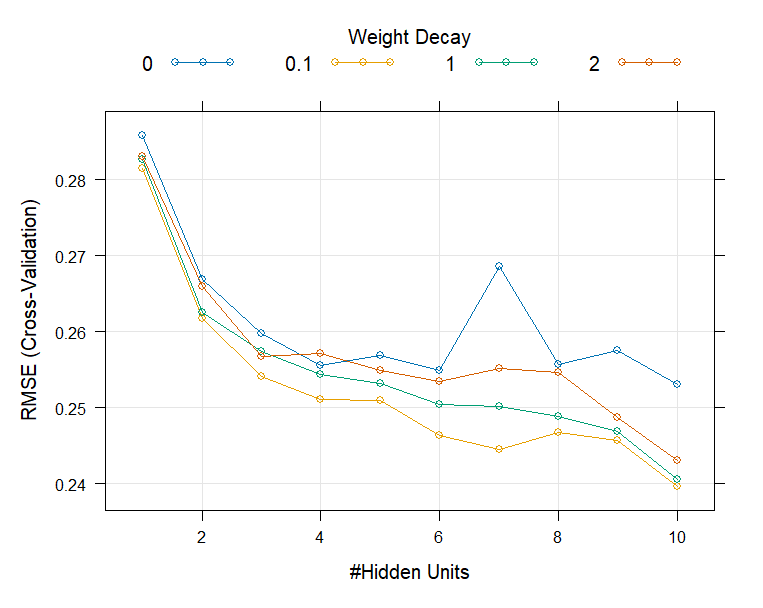
****

**MARS:**

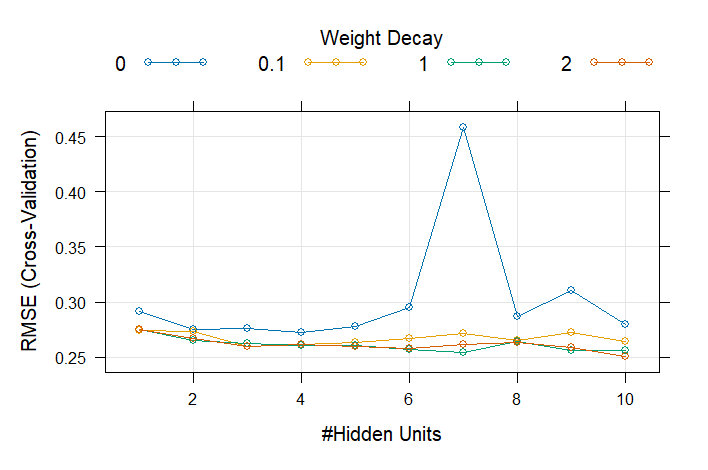
****

**MARS PCA:**

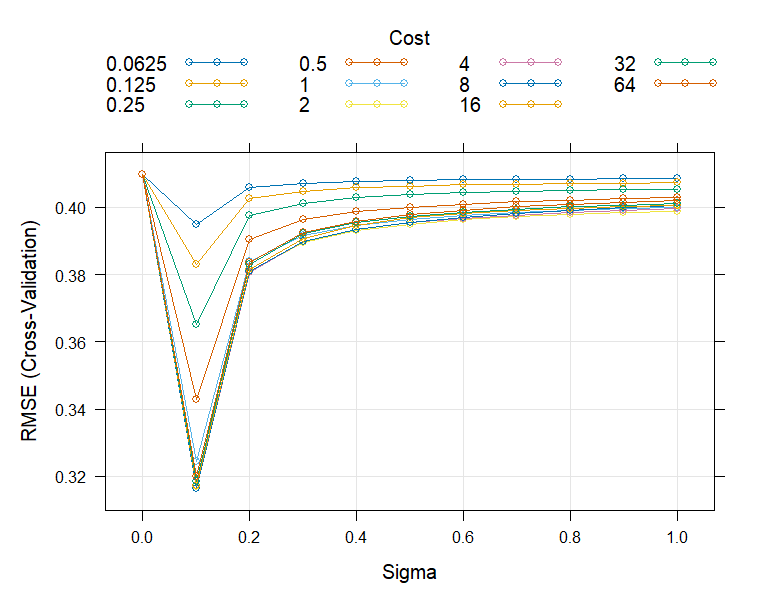
**Neural Networks:**

****

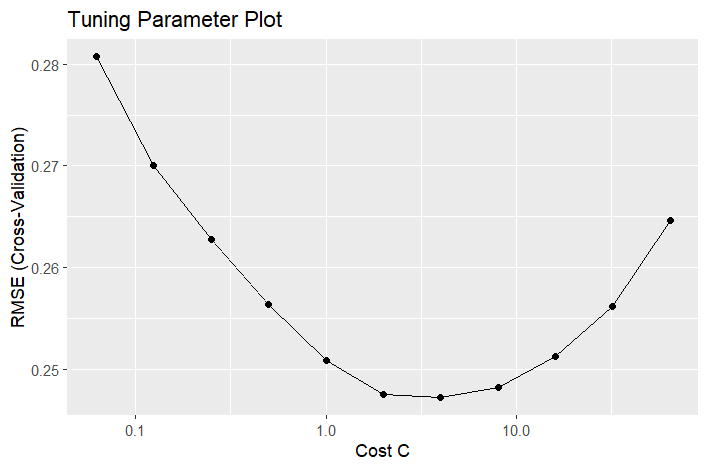
**Neural Networks PCA:**

****

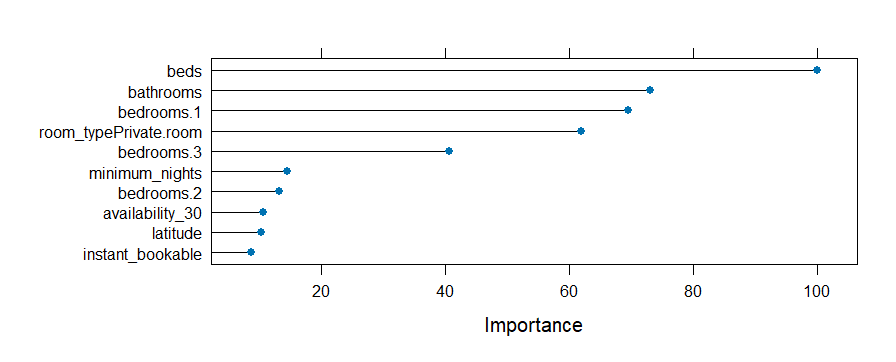
**SVM:**

****

**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]

# 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)