# elastic net

2024-11-20

two dataset: “final_full_rmna_OneHotEncoded_housing.csv” (no outliers removed) “final_rmna_OneHotEncoded_housing
(price outliers removed)

## (1)dataset with all complete cases (no missing values)

```
library(readr)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
housing_data <- read_csv("/Users/chensihan/Documents/GitHub/BIOSTAT625/Data Exploration/final_full_rmna
```

```
## Rows: 60994 Columns: 44
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (44): Transaction_Other, Transaction_Resale, Transaction_New Property, T...
##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

```
set.seed(34)
```

```
attach(housing_data)
#transform price into log_price
housing_data$log_price=log(housing_data$`Price (in rupees)`)
X <- housing_data[, -which(names(housing_data) == "Price (in rupees)" | names(housing_data) == "log_pri
Y <- housing_data$log_price
X_matrix <- as.matrix(X)
Y_matrix <- as.matrix(Y)
```

```
# Initialize storage
alphas <- seq(0, 1, by = 0.01)
elastic_net <- list()  # A list to store models
best_lambda <- numeric(length(alphas))  # Numeric vector for best lambda values
predicted_prices <- list()  # A list to store predictions
results <- list()  # A list to store data frames with actual and predicted values
```

```r
rmse <- numeric(length(alphas))   # Numeric vector for RMSE values
mae <- numeric(length(alphas))   # Numeric vector for MAE values
mape=numeric(length(alphas))  # Numeric vector for MAPE values

for (i in seq_along(alphas)) {
  alpha=alphas[i]
  # Fit elastic net model with alpha
  elastic_net[[i]] <- cv.glmnet(X_matrix, Y_matrix, alpha = alpha)

  # Store the best lambda value (we choose lambda where MSE is smallest) #lambda.min
  best_lambda[i] <- elastic_net[[i]]$lambda.1se

  # Predict on the test set using the best lambda
  predicted_prices[[i]] <- predict(elastic_net[[i]], newx = X_matrix, s = "lambda.1se")

  # Combine actual and predicted values
  results[[i]] <- data.frame(Actual = Y_matrix, Predicted = as.vector(predicted_prices[[i]]))
  # Compute RMSE, MAE, MAPE
  rmse[i] <- sqrt(mean((results[[i]]$Actual - results[[i]]$Predicted)^2))
  mae[i] <- mean(abs(results[[i]]$Actual - results[[i]]$Predicted))
  mape[i] <- mean(abs((results[[i]]$Actual - results[[i]]$Predicted) / results[[i]]$Actual)) * 100
}
```

```r
# Results for RMSE, MAE and MAPE
print(rmse)
```

```
##   [1] 0.3855821 0.3888134 0.3854056 0.3856297 0.3855306 0.3852394 0.3886506
##   [8] 0.3852936 0.3854536 0.3854855 0.3857370 0.3867147 0.3857147 0.3856222
##  [15] 0.3859868 0.3864778 0.3867286 0.3855543 0.3890328 0.3861460 0.3860869
##  [22] 0.3870655 0.3862760 0.3865587 0.3872847 0.3861427 0.3895947 0.3858166
##  [29] 0.3860433 0.3863103 0.3862801 0.3900663 0.3859447 0.3865318 0.3865059
##  [36] 0.3861595 0.3864577 0.3858569 0.3864166 0.3863979 0.3863803 0.3860596
##  [43] 0.3881044 0.3870724 0.3860216 0.3893912 0.3900949 0.3862796 0.3869934
##  [50] 0.3866049 0.3862468 0.3859528 0.3862280 0.3865580 0.3869211 0.3869109
##  [57] 0.3865293 0.3865206 0.3884375 0.3865041 0.3864964 0.3864889 0.3868500
##  [64] 0.3868426 0.3868355 0.3896223 0.3864541 0.3868157 0.3864424 0.3864368
##  [71] 0.3867978 0.3867922 0.3861035 0.3871941 0.3864113 0.3864066 0.3871771
##  [78] 0.3867624 0.3871666 0.3882120 0.3863854 0.3871521 0.3876344 0.3867377
##  [85] 0.3876242 0.3871345 0.3909929 0.3900970 0.3876053 0.3867167 0.3871142
##  [92] 0.3875919 0.3875877 0.3871036 0.3871003 0.3900210 0.3863342 0.3863316
##  [99] 0.3875653 0.3899880 0.3875580
```

```r
which.min(abs(rmse))
```

```
## [1] 6
```

```r
print(mae)
```

```
##   [1] 0.2833366 0.2876325 0.2832731 0.2836855 0.2836181 0.2831885 0.2884449
##   [8] 0.2834210 0.2837441 0.2838287 0.2842460 0.2858316 0.2843000 0.2841965
##  [15] 0.2847922 0.2856338 0.2860648 0.2842069 0.2898366 0.2852135 0.2851336
```

```
##  [22] 0.2867887 0.2855057 0.2859912 0.2872173 0.2853217 0.2907912 0.2848085
##  [29] 0.2851840 0.2856661 0.2856240 0.2915335 0.2850601 0.2860763 0.2860414
##  [36] 0.2854547 0.2859739 0.2849574 0.2859160 0.2858893 0.2858639 0.2853126
##  [43] 0.2886414 0.2869944 0.2852562 0.2905738 0.2916623 0.2857240 0.2868923
##  [50] 0.2862769 0.2856767 0.2851714 0.2856472 0.2862045 0.2867950 0.2867809
##  [57] 0.2861623 0.2861502 0.2891273 0.2861273 0.2861165 0.2861060 0.2866953
##  [64] 0.2866846 0.2866742 0.2909468 0.2860551 0.2866451 0.2860380 0.2860298
##  [71] 0.2866186 0.2866102 0.2854592 0.2872361 0.2859917 0.2859846 0.2872119
##  [78] 0.2865649 0.2871969 0.2887208 0.2859520 0.2871758 0.2878824 0.2865271
##  [85] 0.2878680 0.2871502 0.2928018 0.2915646 0.2878396 0.2864962 0.2871178
##  [92] 0.2878148 0.2878069 0.2871019 0.2870969 0.2914284 0.2858706 0.2858664
##  [99] 0.2877643 0.2913683 0.2877503
```

```
which.min(abs(mae))
```

```
## [1] 6
```

```
print(mape)
```

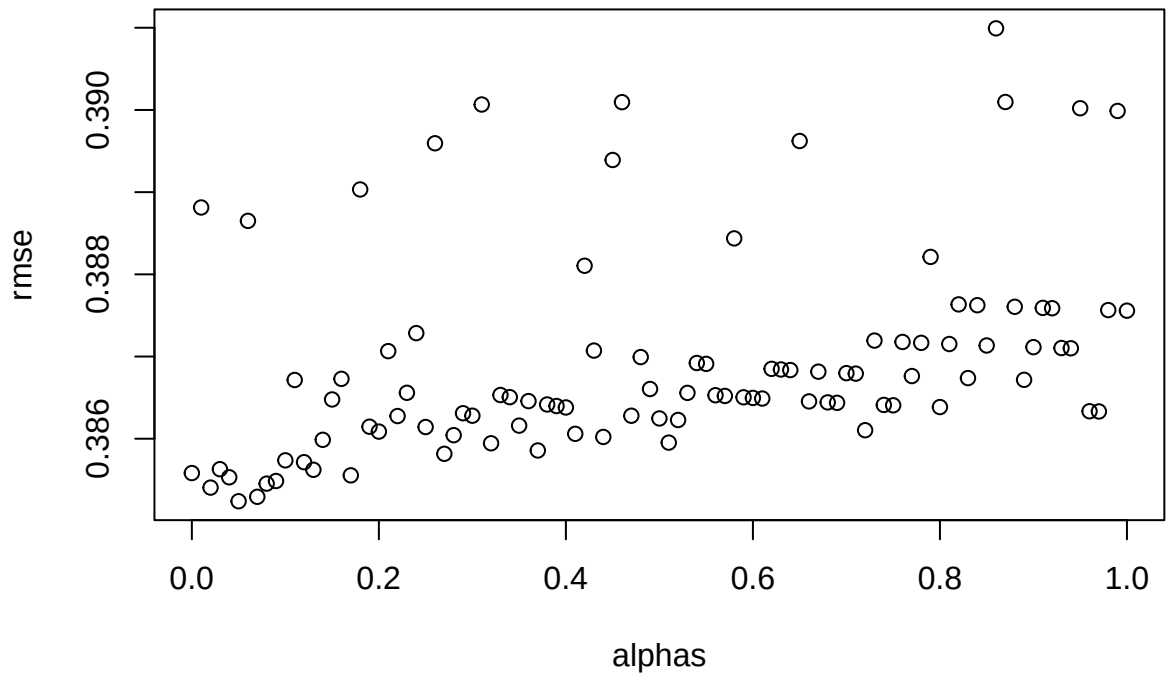
```
##   [1] 3.209803 3.257968 3.209123 3.213756 3.213000 3.208178 3.267229 3.210788
##   [9] 3.214401 3.215347 3.220016 3.237867 3.220692 3.219570 3.226301 3.235789
##  [17] 3.240661 3.219811 3.283270 3.231166 3.230289 3.248960 3.234518 3.240006
##  [25] 3.253879 3.232500 3.294342 3.226757 3.230993 3.236444 3.235983 3.302865
##  [33] 3.229650 3.241121 3.240739 3.234134 3.240002 3.228549 3.239370 3.239078
##  [41] 3.238801 3.232586 3.270349 3.251658 3.231972 3.292220 3.304281 3.237282
##  [49] 3.250544 3.243546 3.236768 3.231064 3.236445 3.242755 3.249482 3.249329
##  [57] 3.242297 3.242165 3.276064 3.241918 3.241801 3.241687 3.248396 3.248280
##  [65] 3.248167 3.296291 3.241137 3.247851 3.240951 3.240864 3.247562 3.247472
##  [73] 3.234415 3.254627 3.240451 3.240375 3.254364 3.246979 3.254201 3.271556
##  [81] 3.240022 3.253972 3.262052 3.246570 3.261895 3.253695 3.316739 3.303044
##  [89] 3.261586 3.246235 3.253342 3.261312 3.261224 3.253170 3.253115 3.301515
##  [97] 3.239143 3.239098 3.260757 3.300841 3.260602
```

```
which.min(abs(mape))
```

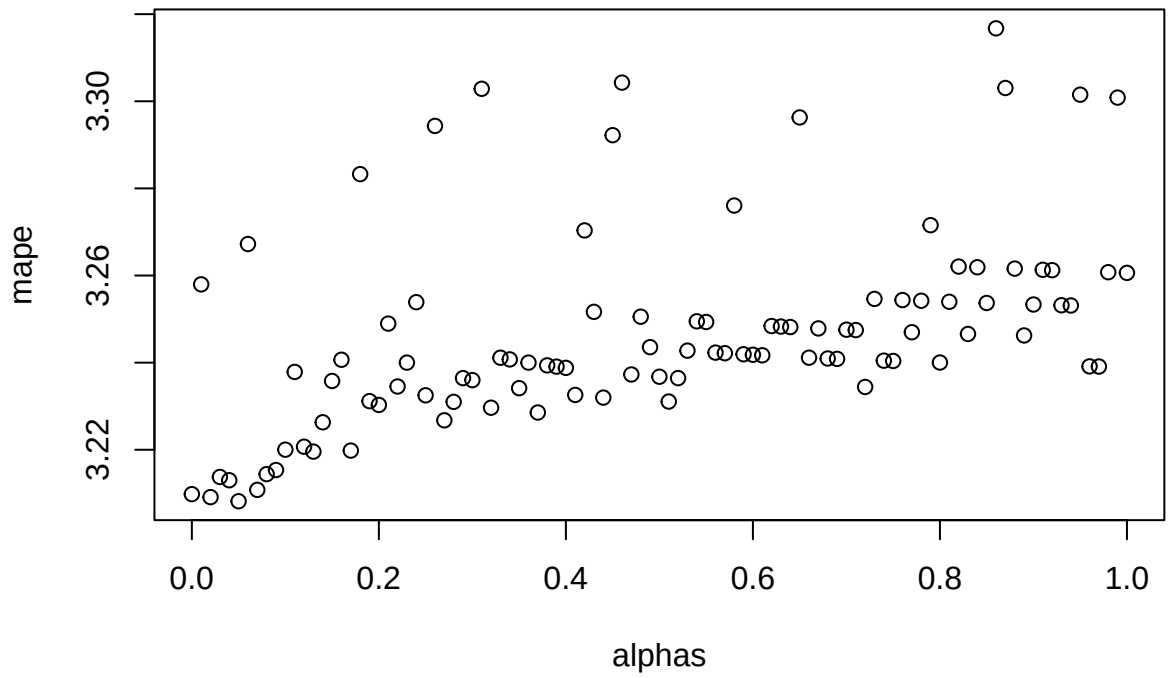
```
## [1] 6
```

### best alpha is 0.05

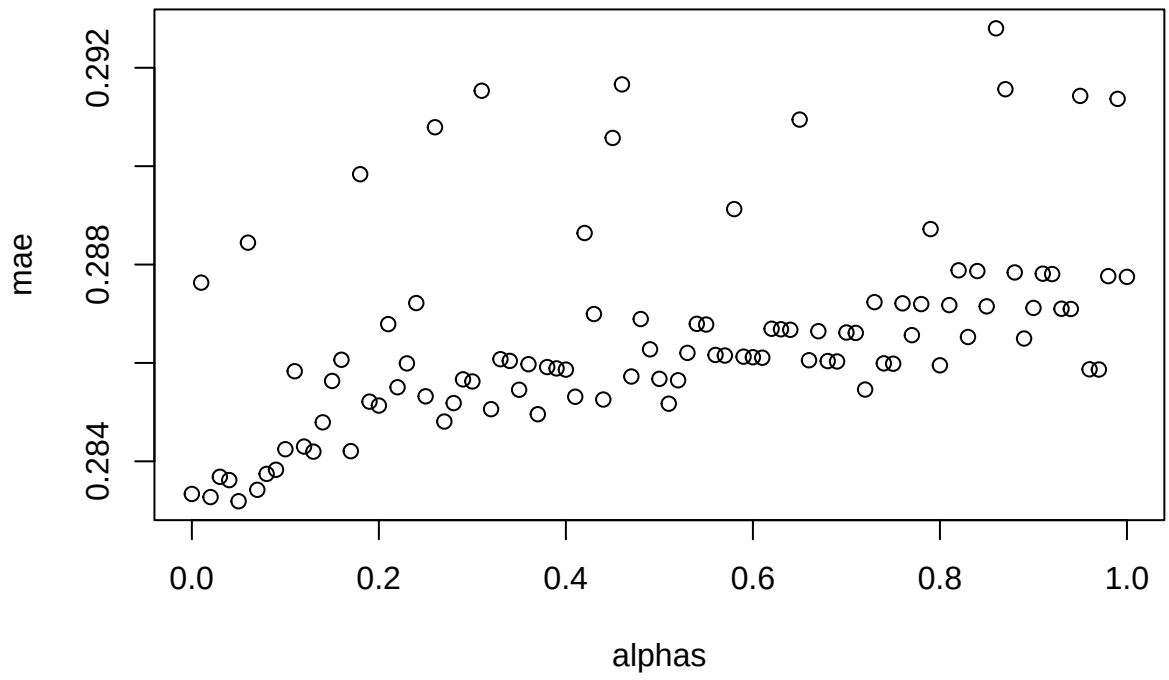
```
plot(alphas,rmse)
```




```
plot(alphas,mape)
```




```
plot(alphas, mae)
```

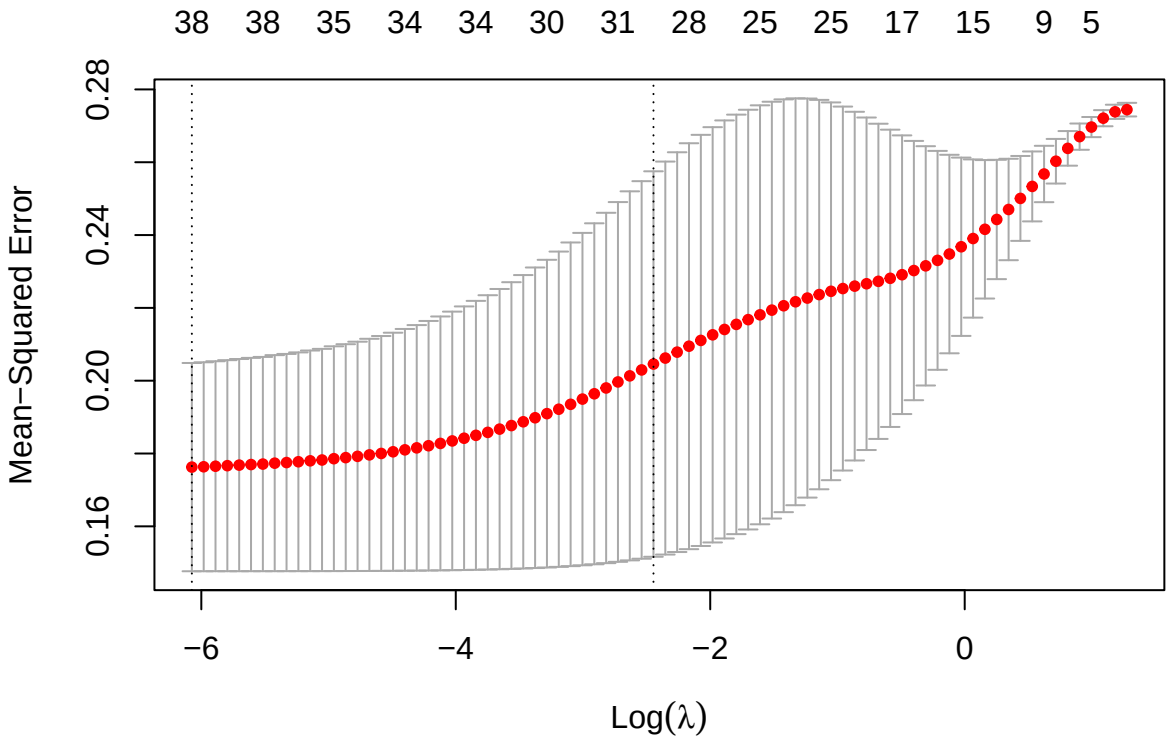



```
alphas[6]
```

```
## [1] 0.05
```

**we choose lambda.1se = 0.08663284**

```
fit= cv.glmnet(X_matrix, Y_matrix, alpha = alphas[6])
plot(fit)
```




```
# two methods to choose lambda
fit$lambda.1se
```

```
## [1] 0.08663284
```

```
fit$lambda.min
```

```
## [1] 0.002301044
```

s comes from lambda.1se

```
# Compute RMSE, MAE, MAPE and R_squared in training dataset
predicted_prices<- predict(fit, newx = X_matrix,s=0.08663284)

  # Combine actual and predicted values
results<- data.frame(Actual = Y_matrix, Predicted = as.vector(predicted_prices))

  rmse<- sqrt(mean((results$Actual - results$Predicted)^2))
  mae<- mean(abs(results$Actual - results$Predicted))
  mape<- mean(abs((results$Actual - results$Predicted) / results$Actual)) * 100
  rmse
```

```
## [1] 0.389574
```

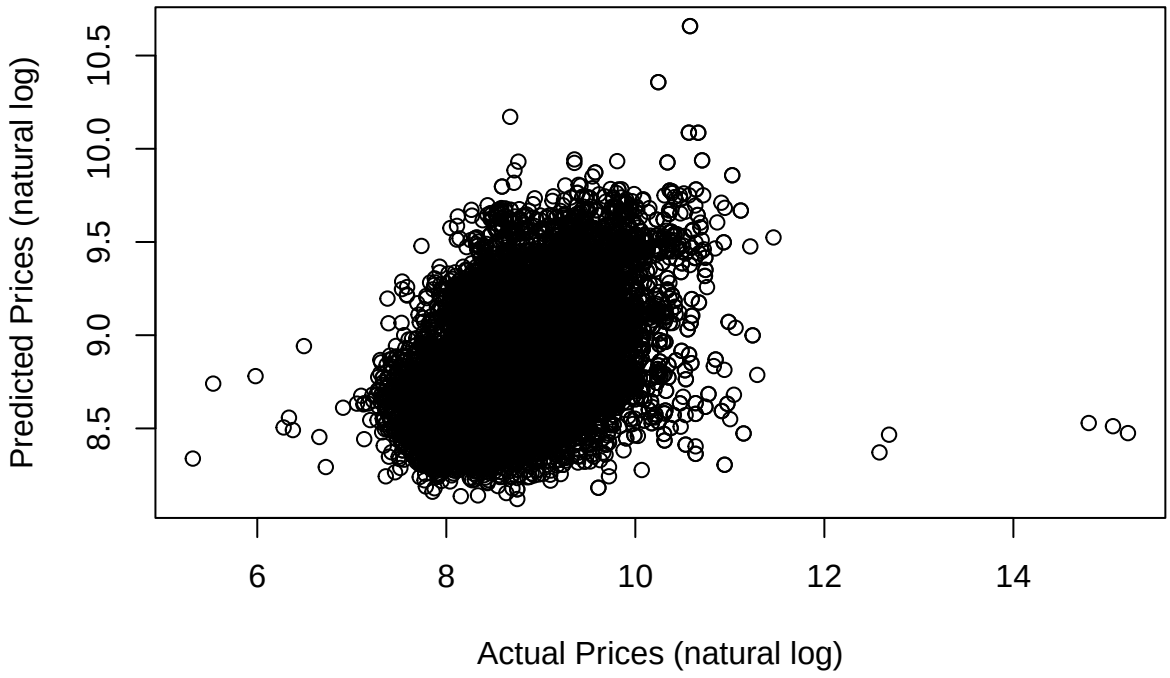
```
  mae
```

```
## [1] 0.2895238
```

```
  mape
```

```
## [1] 3.27942
```

```
plot(Y_matrix, predicted_prices, xlab = "Actual Prices (natural log)", ylab = "Predicted Prices (natural
```

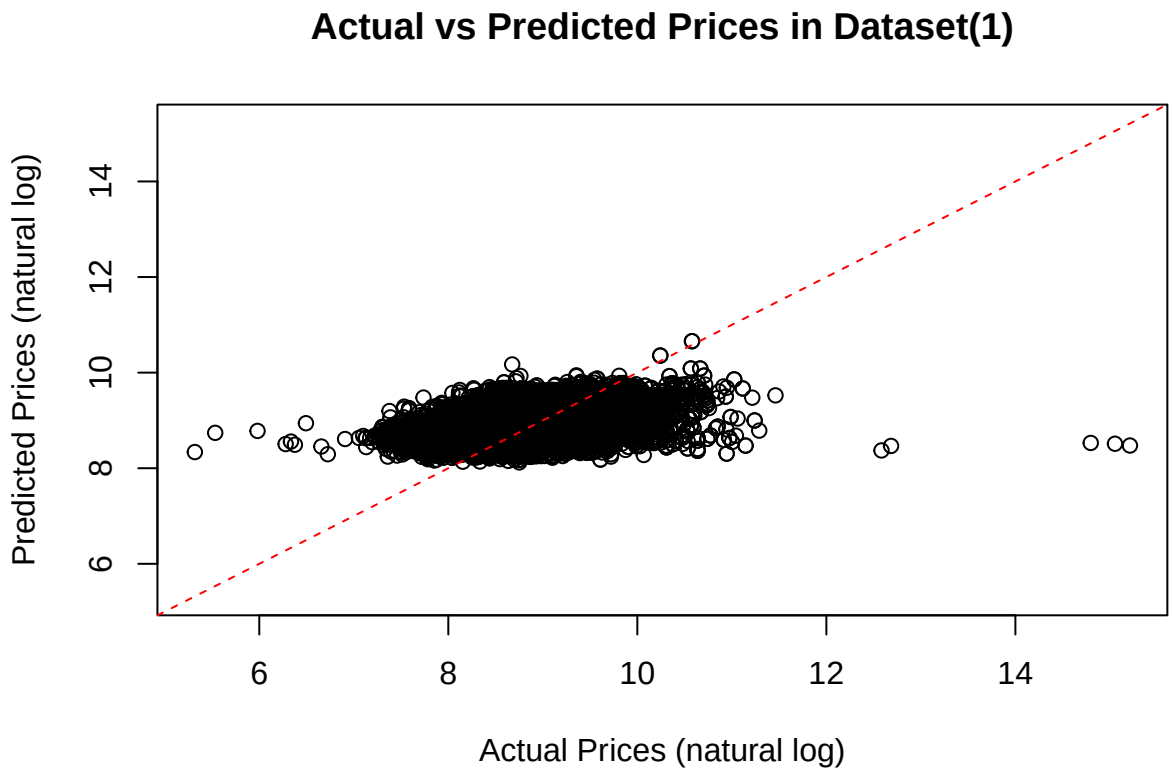



```
y_mean <- mean(Y_matrix)
SS_total <- sum((Y_matrix - y_mean)^2)
SS_residual <- sum((Y_matrix - predicted_prices)^2)
R_squared <- 1 - (SS_residual / SS_total)
print(R_squared)
```

```
## [1] 0.4465968
```

```
# Determine the range of both axes
range_values <- range(c(Y_matrix, predicted_prices))

# Create the plot with the same scale for x and y axes
plot(
```

```
  Y_matrix, predicted_prices,
  xlab = "Actual Prices (natural log)",
  ylab = "Predicted Prices (natural log)",
  xlim = range_values,
  ylim = range_values,
  main = "Actual vs Predicted Prices in Dataset(1)"
)

# Optionally, add a 45-degree line for reference
abline(0, 1, col = "red", lty = 2)  # A red dashed line
```




# (2)a refined dataset excluding both missing values and outliers

```
housing_data2 <- read_csv("/Users/chensihan/Documents/GitHub/BIOSTAT625/Data Exploration/final_rmna_One
```

```
## Rows: 60988 Columns: 44
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (44): Transaction_Other, Transaction_Resale, Transaction_New Property, T...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
attach(housing_data2)
```

```
## The following objects are masked from housing_data:
##
##     Balcony_Category_=1, Balcony_Category_=2, Balcony_Category_=3,
##     Balcony_Category_=4, Balcony_Category_>=5, Bathroom_Category_=1,
##     Bathroom_Category_=2, Bathroom_Category_=3, Bathroom_Category_=4,
##     Bathroom_Category_>=5, Car_Parking_Category_covered,
##     Car_Parking_Category_open, Car_Parking_Category_unknown,
##     clustering_1, clustering_2, clustering_3, clustering_4, Current
##     Floor.c, facing_East, facing_North, facing_North - East,
##     facing_North - West, facing_South, facing_South - East,
##     facing_South -West, facing_West, Furnishing_Furnished,
##     Furnishing_Semi-Furnished, Furnishing_Unfurnished, Garden/Park,
##     Main Road, Ownership_Category_Co-operative Society,
##     Ownership_Category_Freehold, Ownership_Category_Leasehold,
##     Ownership_Category_Other, Ownership_Category_Power Of Attorney,
##     Pool, Price (in rupees), Total Floors.c, Transaction_New Property,
##     Transaction_Other, Transaction_Rent/Lease, Transaction_Resale
```

```
#transform price into log_price
housing_data2$log_price=log(housing_data2$`Price (in rupees)`)
X2 <- housing_data2[, -which(names(housing_data2) == "Price (in rupees)" | names(housing_data2) == "log
Y2 <- housing_data2$log_price
X2_matrix <- as.matrix(X2)
Y2_matrix <- as.matrix(Y2)
```

```
# Initialize storage
alphas2 <- seq(0, 1, by = 0.01)
elastic_net2 <- list()  # A list to store models
best_lambda2 <- numeric(length(alphas2))  # Numeric vector for best lambda values
predicted_prices2 <- list()  # A list to store predictions
results2 <- list()  # A list to store data frames with actual and predicted values
rmse2 <- numeric(length(alphas2))   # Numeric vector for RMSE values
mae2 <- numeric(length(alphas2))   # Numeric vector for MAE values
mape2=numeric(length(alphas2))  # Numeric vector for MAPE values

for (i in seq_along(alphas2)) {
  alpha2=alphas2[i]

  elastic_net2[[i]] <- cv.glmnet(X2_matrix, Y2_matrix, alpha = alpha2)

  # Store the best lambda value (we choose lambda where MSE is smallest)
  best_lambda2[i] <- elastic_net2[[i]]$lambda.1se

  # Predict on the test set using the best lambda
  predicted_prices2[[i]] <- predict(elastic_net2[[i]], newx = X2_matrix, s = "lambda.1se")

  # Combine actual and predicted values
  results2[[i]] <- data.frame(Actual = Y2_matrix, Predicted = as.vector(predicted_prices2[[i]]))
  # Compute RMSE, MAE, MAPE
  rmse2[i] <- sqrt(mean((results2[[i]]$Actual - results2[[i]]$Predicted)^2))
  mae2[i] <- mean(abs(results2[[i]]$Actual - results2[[i]]$Predicted))
  mape2[i] <- mean(abs((results2[[i]]$Actual - results2[[i]]$Predicted) / results2[[i]]$Actual)) * 100
}
```

```r
# Results for RMSE, MAE and MAPE
print(rmse2)
```

```
##   [1] 0.3821987 0.3817843 0.3815171 0.3818539 0.3821323 0.3815062 0.3857840
##   [8] 0.3822441 0.3818566 0.3846416 0.3821449 0.3820215 0.3821215 0.3815530
##  [15] 0.3823979 0.3823083 0.3824910 0.3817799 0.3854926 0.3825618 0.3828105
##  [22] 0.3827476 0.3819437 0.3863798 0.3818856 0.3822848 0.3818384 0.3818171
##  [29] 0.3824546 0.3830761 0.3826934 0.3865346 0.3823532 0.3823331 0.3829201
##  [36] 0.3820636 0.3832387 0.3828494 0.3825149 0.3820119 0.3831534 0.3822097
##  [43] 0.3819801 0.3821878 0.3827300 0.3839189 0.3827036 0.3826913 0.3826797
##  [50] 0.3826687 0.3826581 0.3872467 0.3823512 0.3826293 0.3829605 0.3829507
##  [57] 0.3833177 0.3870647 0.3829238 0.3832901 0.3832816 0.3825684 0.3828927
##  [64] 0.3828856 0.3825501 0.3832438 0.3825389 0.3828583 0.3832242 0.3828466
##  [71] 0.3836335 0.3867673 0.3820179 0.3841160 0.3818230 0.3824972 0.3831808
##  [78] 0.3828068 0.3820005 0.3831672 0.3866360 0.3827903 0.3831546 0.3831507
##  [85] 0.3827791 0.3831432 0.3831396 0.3827688 0.3835401 0.3831293 0.3827592
##  [92] 0.3835287 0.3840115 0.3827504 0.3831140 0.3827448 0.3827422 0.3824319
##  [99] 0.3827370 0.3831004 0.3834994
```

```r
which.min(abs(rmse2))
```

```
## [1] 6
```

```r
print(mae2)
```

```
##   [1] 0.2831234 0.2827070 0.2824052 0.2829435 0.2834486 0.2825132 0.2888955
##   [8] 0.2837998 0.2832729 0.2876597 0.2837756 0.2836342 0.2838291 0.2829140
##  [15] 0.2843173 0.2842180 0.2845493 0.2834341 0.2893896 0.2847615 0.2852026
##  [22] 0.2851207 0.2838094 0.2907667 0.2837385 0.2844010 0.2836783 0.2836493
##  [29] 0.2847282 0.2857796 0.2851637 0.2910849 0.2845899 0.2845595 0.2855740
##  [36] 0.2841399 0.2860966 0.2854770 0.2849197 0.2840735 0.2859874 0.2844124
##  [43] 0.2840298 0.2843870 0.2853119 0.2871731 0.2852773 0.2852603 0.2852441
##  [50] 0.2852286 0.2852137 0.2922066 0.2846829 0.2851725 0.2857192 0.2857050
##  [57] 0.2862962 0.2919351 0.2856674 0.2862581 0.2862462 0.2850831 0.2856241
##  [64] 0.2856142 0.2850553 0.2861924 0.2850382 0.2855744 0.2861640 0.2855575
##  [71] 0.2867912 0.2914543 0.2841617 0.2874823 0.2838026 0.2849726 0.2860993
##  [78] 0.2854988 0.2841363 0.2860786 0.2912328 0.2854739 0.2860599 0.2860543
##  [85] 0.2854568 0.2860434 0.2860382 0.2854410 0.2866565 0.2860233 0.2854262
##  [92] 0.2866398 0.2873194 0.2854124 0.2860009 0.2854037 0.2853995 0.2848652
##  [99] 0.2853913 0.2859807 0.2865958
```

```r
which.min(abs(mae2))
```

```
## [1] 3
```

```r
print(mape2)
```

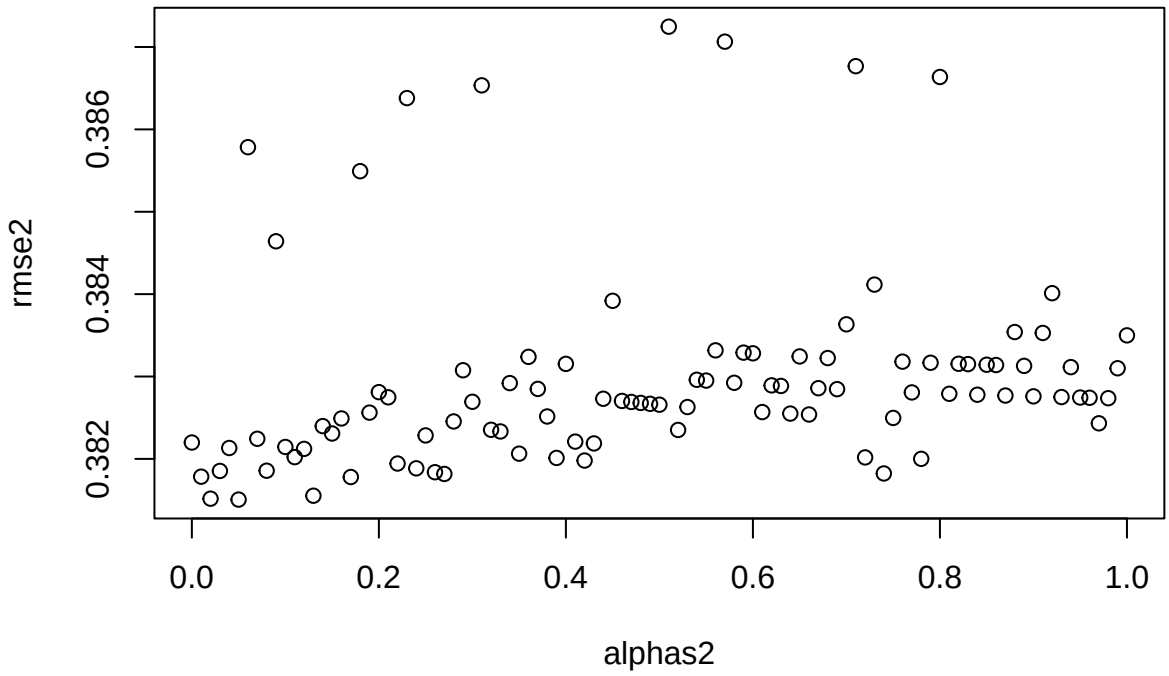
```
##   [1] 3.209195 3.204544 3.201165 3.207201 3.212855 3.202374 3.274119 3.216778
##   [9] 3.210883 3.260174 3.216505 3.214952 3.217177 3.206950 3.222734 3.221653
##  [17] 3.225422 3.212898 3.280047 3.227896 3.232882 3.231981 3.217240 3.295774
```

```
##  [25] 3.216476 3.223954 3.215830 3.215516 3.227681 3.239547 3.232619 3.299619
##  [33] 3.226176 3.225842 3.237295 3.221144 3.243239 3.236236 3.229957 3.220432
##  [41] 3.242043 3.224263 3.219960 3.223994 3.234436 3.255586 3.234064 3.233880
##  [49] 3.233704 3.233536 3.233374 3.312015 3.227386 3.232927 3.239115 3.238961
##  [57] 3.245696 3.308975 3.238554 3.245281 3.245151 3.231959 3.238086 3.237979
##  [65] 3.231659 3.244566 3.231474 3.237547 3.244257 3.237364 3.251426 3.303581
##  [73] 3.221590 3.259332 3.217535 3.230763 3.243555 3.236731 3.221319 3.243329
##  [81] 3.301097 3.236462 3.243127 3.243066 3.236278 3.242949 3.242893 3.236107
##  [89] 3.249962 3.242732 3.235947 3.249781 3.257548 3.235799 3.242490 3.235705
##  [97] 3.235659 3.229605 3.235571 3.242272 3.249306
```

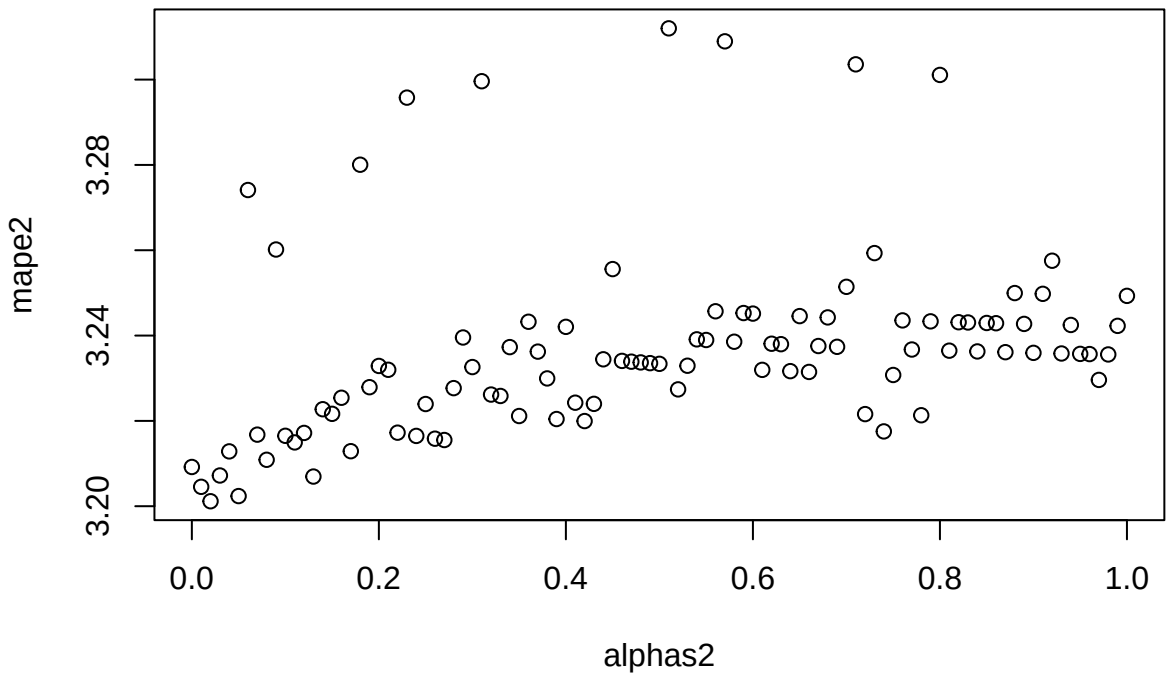
```
which.min(abs(mape2))
```

```
## [1] 3
```

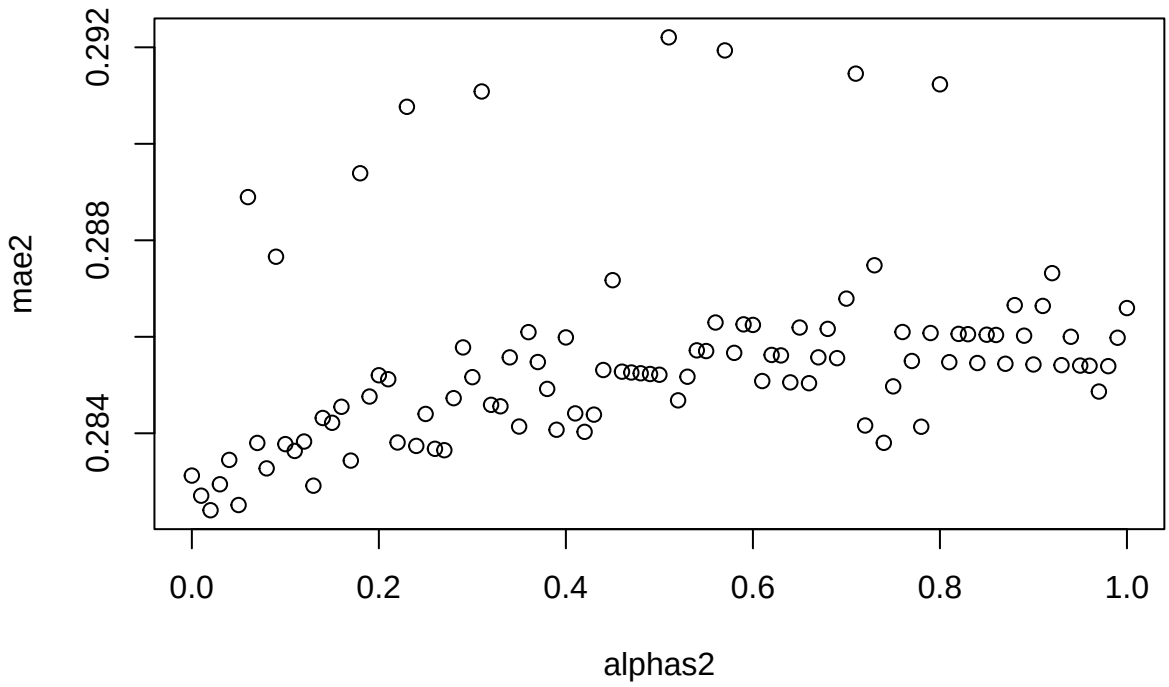
```
plot(alphas2,rmse2)
```




```
plot(alphas2,mape2)
```

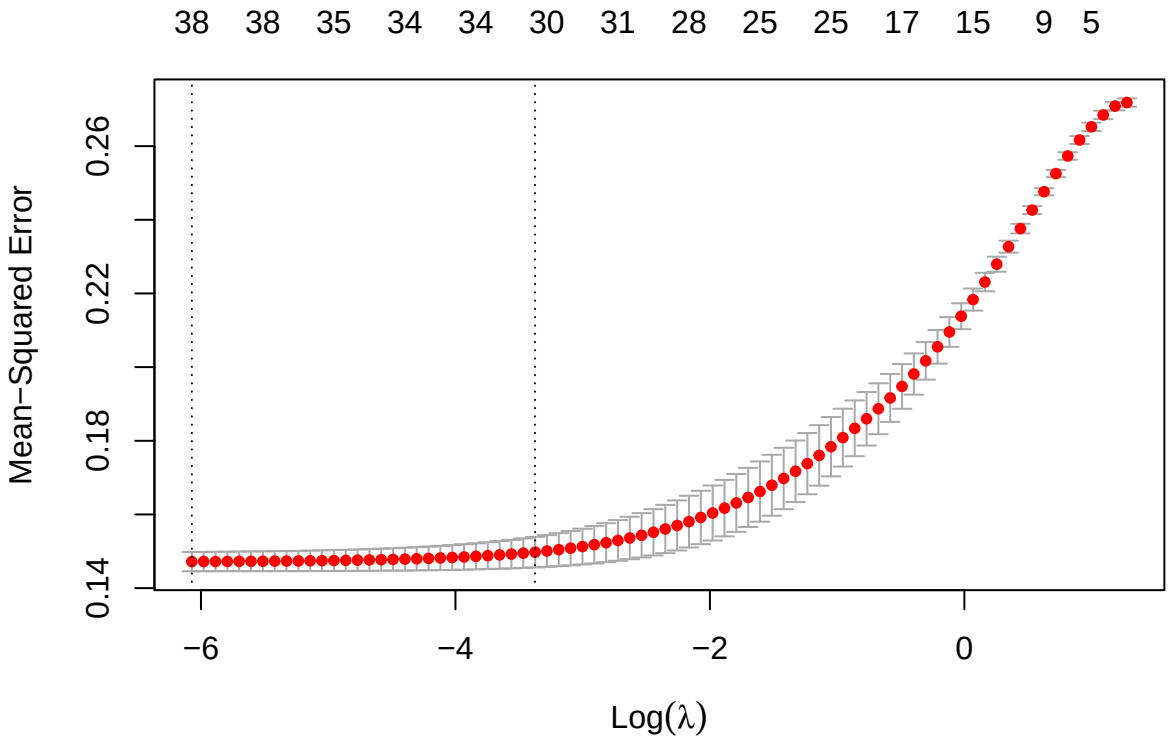



```
plot(alphas2, mae2)
```




## best alpha is 0.05

```
fit2= cv.glmnet(X2_matrix, Y2_matrix, alpha = 0.05)
plot(fit2)
```




```
# two methods to choose lambda
fit2$lambda.1se
```

```
## [1] 0.0342591
```

```
fit2$lambda.min
```

```
## [1] 0.002307059
```

### we choose lambda.1se = 0.0342591

s comes from lambda.1se

```
# Compute RMSE, MAE, MAPE and R_squared in training dataset
predicted_prices2<- predict(fit2, newx = X2_matrix,s=0.0342591)

  # Combine actual and predicted values
results2<- data.frame(Actual = Y2_matrix, Predicted = as.vector(predicted_prices2))

  rmse2<- sqrt(mean((results2$Actual - results2$Predicted)^2))
  mae2<- mean(abs(results2$Actual - results2$Predicted))
  mape2<- mean(abs((results2$Actual - results2$Predicted) / results2$Actual)) * 100
  rmse2
```

```
## [1] 0.38196
```

```
  mae2
```

```
## [1] 0.2832698
```

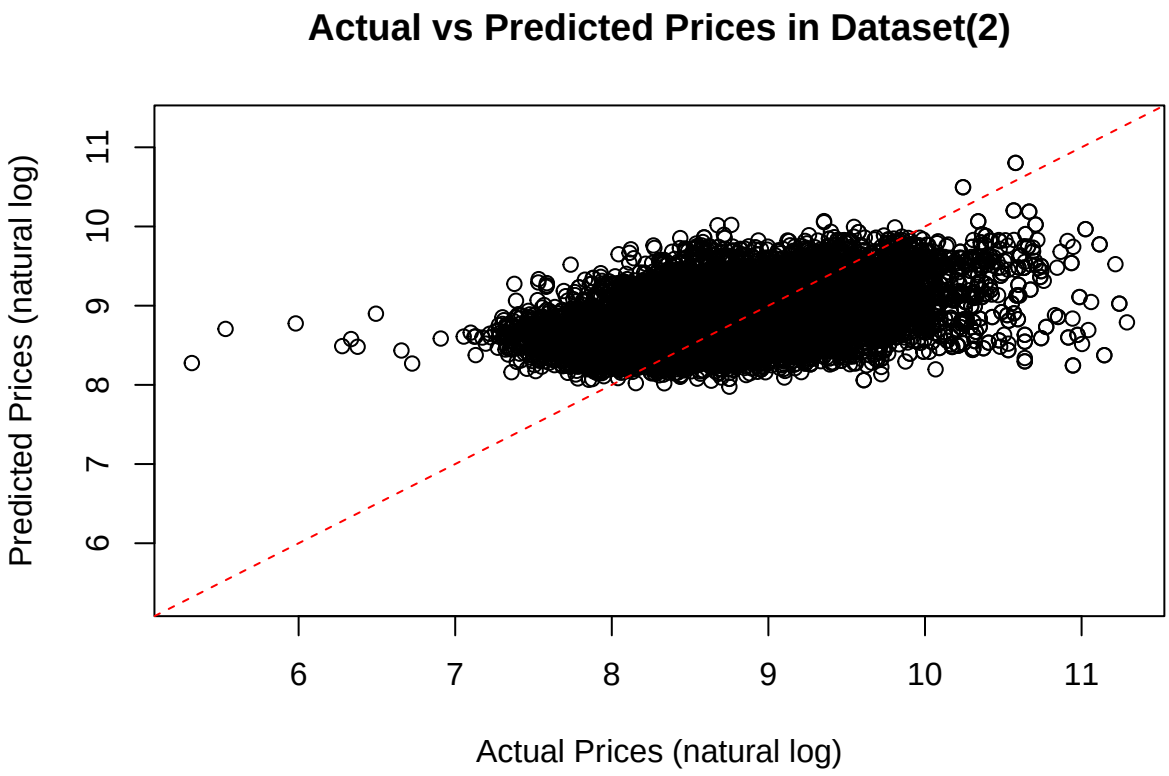
```
  mape2
```

```
## [1] 3.210852
```

```
y2_mean <- mean(Y2_matrix)
SS_total2 <- sum((Y2_matrix - y2_mean)^2)
SS_residual2 <- sum((Y2_matrix - predicted_prices2)^2)
R_squared2 <- 1 - (SS_residual2 / SS_total2)
print(R_squared2)
```

```
## [1] 0.4633214
```

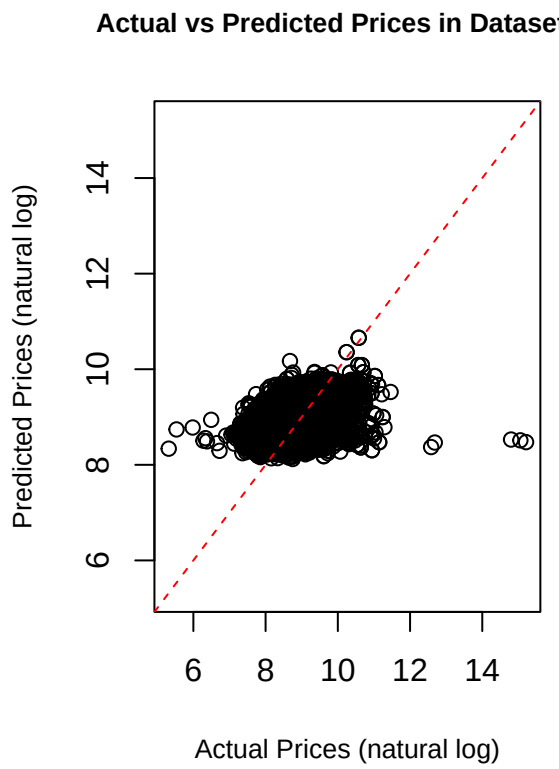
```
# Determine the range of both axes
range_values <- range(c(Y2_matrix, predicted_prices2))

# Create the plot with the same scale for x and y axes
plot(
  Y2_matrix, predicted_prices2,
  xlab = "Actual Prices (natural log)",
  ylab = "Predicted Prices (natural log)",
  xlim = range_values,
  ylim = range_values,
  main = "Actual vs Predicted Prices in Dataset(2)"
)

# Optionally, add a 45-degree line for reference
abline(0, 1, col = "red", lty = 2)  # A red dashed line
```

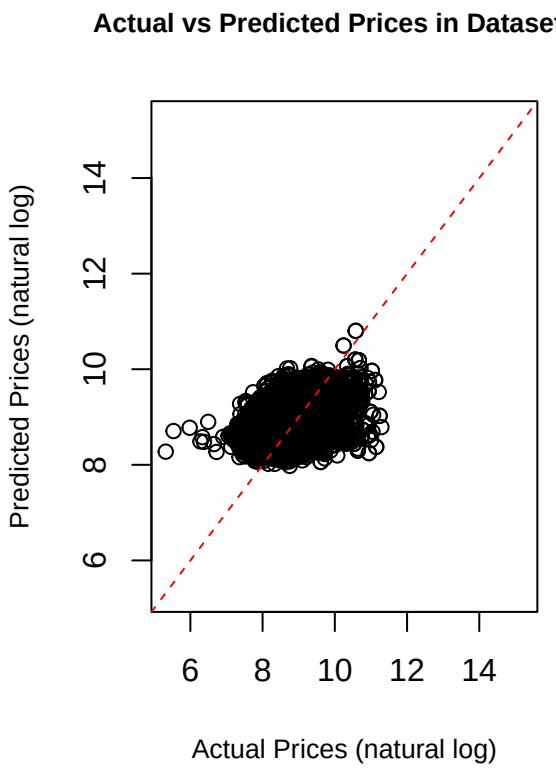



# Actual vs Predicted Prices Plot

```r
# Set layout to display 2 plots side by side
par(mfrow = c(1, 2))  # 1 row, 2 columns

# Plot 1
range_values <- range(c(Y_matrix, predicted_prices))
plot(
  Y_matrix, predicted_prices,
  xlab = "Actual Prices (natural log)",
  ylab = "Predicted Prices (natural log)",
  xlim = range_values,
  ylim = range_values,
  cex.lab = 0.8,
  cex.main = 0.8,
  main = "Actual vs Predicted Prices in Dataset(1)"
)
abline(0, 1, col = "red", lty = 2)  # A red dashed line

# Plot 2

plot(
  Y2_matrix, predicted_prices2,
  xlab = "Actual Prices (natural log)",
  ylab = "Predicted Prices (natural log)",
  xlim = range_values,
  ylim = range_values,
```

```
  cex.lab = 0.8,
  cex.main = 0.8,
  main = "Actual vs Predicted Prices in Dataset(2)"
)
abline(0, 1, col = "red", lty = 2)  # A red dashed line
```




```
# Reset layout to default (1 plot)
par(mfrow = c(1, 1))
```

```
library(knitr)

# Create a data frame
performance_metrics <- data.frame(
  `Performance Metric` = c("RMSE", "MAE", "MAPE", "R-square"),
  `Dataset$(1)$` = c(0.390, 0.290, 3.279, 0.447),
  `Dataset$(2)$` = c(0.382, 0.283, 3.211, 0.463)
)

# Display the table using knitr::kable with booktabs formatting
kable(performance_metrics, format = "latex", booktabs = TRUE, caption = "Model Performance Metrics for
```

Table 1: Model Performance Metrics for Dataset 1 and Dataset 2

| Performance.Metric | Dataset..1.. | Dataset..2.. |
|---|---|---|
| RMSE | 0.390 | 0.382 |
| MAE | 0.290 | 0.283 |
| MAPE | 3.279 | 3.211 |
| R-square | 0.447 | 0.463 |