Modeling

In this section, we fit the model. Before doing that, however, it is essential to select the appropriate features and the ideal model to fit based on the problem that we are trying to solve for.

Data Loading

We install and load the required packages, followed by reading the model-ready csv file. Further, we convert certain categorical columns so that R considers them as a factor instead of an integer

```r
list.of.packages <- c("caret", "glmnet", "tidyverse", "randomForest")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)

library('tidyverse')
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library('caret')
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library('glmnet')
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
```

```
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8
```

```r
library('randomForest')
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
set.seed(123) # Setting a seed for reproducibility

data <- read.csv("model_data_clean.csv")

data$STATEFIP <- as.factor(data$STATEFIP)
data$SEX <- as.factor(data$SEX)
data$RACE <- as.factor(data$RACE)
data$OCC_CODE <- as.factor(data$OCC_CODE)
data$IND_CODE <- as.factor(data$IND_CODE)
data$CITIZEN <- as.factor(data$CITIZEN)
```

Feature Selection

Once we have the data loaded in, we perform feature selection. In our case, we use the Lasso Regression method to understand which features are not important. Lasso Regression automatically performs feature selection by shrinking some coefficients of features to zero. We ensure that Dummy Variables are created for each categorical feature before running lasso regression and

```r
# Split input data to two datasets, one for Log transformed income and other for raw income values
data1 <- data %>% select(-INCTOT)
data2 <- data %>% select(-log_INCTOT)

# Create Train and Test data for both these datasets
splitIndex1 <- createDataPartition(data$log_INCTOT, p = 0.8, list = FALSE)
train1 <- data1[ splitIndex1,]
test1 <- data1[-splitIndex1,]

splitIndex2 <- createDataPartition(data$INCTOT, p = 0.8, list = FALSE)
train2 <- data2[ splitIndex2,]
test2 <- data2[-splitIndex2,]
```

```r
# Define X and Y for all Cases as well as create Dummy Variables for Categorical Features
x1 <- train1[, c("MET_CODE", "STATEFIP", "AGE", "SEX", "RACE", "CITIZEN", "OCC_CODE", "IND_CODE", "EDUC
x1 <- model.matrix(~., data = x1)
y1 <- train1$log_INCTOT

x2 <- train2[, c("MET_CODE", "STATEFIP", "AGE", "SEX", "RACE", "CITIZEN", "OCC_CODE", "IND_CODE", "EDUC
x2 <- model.matrix(~., data = x2)
y2 <- train2$INCTOT

# Fit LASSO Model to perform Feature Selection
fit1 <- cv.glmnet(x1, y1, family="gaussian")
coef_selected1 <- coef(fit1, s="lambda.min")
print(coef_selected1)
```

```
## 150 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  9.2807169424
## (Intercept)  .
## MET_CODE     0.0002106139
## STATEFIP4    0.0216144081
## STATEFIP5   -0.0748942153
## STATEFIP6    0.0709584246
## STATEFIP8    0.0522905384
## STATEFIP9    0.1198153903
## STATEFIP10  -0.0125434876
## STATEFIP11   0.2041834733
## STATEFIP12  -0.0521119321
## STATEFIP13   .
## STATEFIP15   .
## STATEFIP16   .
## STATEFIP17   0.0546076128
## STATEFIP18  -0.0137392152
## STATEFIP19   .
## STATEFIP20   .
## STATEFIP21   .
## STATEFIP22  -0.0708479894
## STATEFIP23   .
## STATEFIP24   0.1135998593
## STATEFIP25   0.0794252899
## STATEFIP26   0.0325740766
## STATEFIP27   0.1220598901
## STATEFIP28  -0.2235868097
## STATEFIP29   0.0335774926
## STATEFIP30   0.0334916688
## STATEFIP31   0.0546510618
## STATEFIP32  -0.0528894415
## STATEFIP33   0.0983096685
## STATEFIP34   0.0756824216
## STATEFIP35  -0.0808104030
## STATEFIP36   0.0191823776
## STATEFIP37  -0.0608422978
## STATEFIP38   0.0170593128
## STATEFIP39  -0.0087523331
```

```
## STATEFIP40  -0.1395860391
## STATEFIP41   0.0472878282
## STATEFIP42   0.0230028410
## STATEFIP44   0.0901328362
## STATEFIP45  -0.0679741416
## STATEFIP46  -0.0078002832
## STATEFIP47  -0.0286410382
## STATEFIP48  -0.0298732158
## STATEFIP49   0.0064683061
## STATEFIP50   0.0570714998
## STATEFIP51   0.0119824237
## STATEFIP53   0.1495437072
## STATEFIP54  -0.0354807478
## STATEFIP55  -0.0464689074
## AGE          0.0125996277
## SEX2        -0.2021993291
## RACE200     -0.0993347789
## RACE300     -0.0433241449
## RACE651      0.0105318883
## RACE652     -0.0202073839
## RACE801     -0.1649758062
## RACE802     -0.0439386953
## RACE803      0.0871572906
## RACE804     -0.1438827856
## RACE805     -0.0578753491
## RACE806      .
## RACE807      0.0463344508
## RACE808      0.2953449839
## RACE809      .
## RACE810      .
## RACE811     -0.4123873847
## RACE812      .
## RACE813      0.1028154325
## RACE816      .
## RACE817      .
## RACE819      .
## RACE820     -0.4220689865
## RACE830      0.4049830609
## CITIZEN2    -0.0455321162
## CITIZEN3    -0.0528105584
## CITIZEN4    -0.0676636027
## CITIZEN5    -0.1181049689
## OCC_CODE2   -0.1099284828
## OCC_CODE3    0.0064373151
## OCC_CODE4    .
## OCC_CODE5   -0.2929857386
## OCC_CODE6   -0.3432532355
## OCC_CODE7    0.0808994485
## OCC_CODE8   -0.3453067802
## OCC_CODE9   -0.2562148565
## OCC_CODE10  -0.0330667331
## OCC_CODE11  -0.4721650465
## OCC_CODE12  -0.2309211254
## OCC_CODE13  -0.5307815869
```

```
## OCC_CODE14  -0.5133977186
## OCC_CODE15  -0.5315947433
## OCC_CODE16  -0.2240631495
## OCC_CODE17  -0.3828792459
## OCC_CODE18  -0.5984824745
## OCC_CODE19  -0.3422765678
## OCC_CODE20  -0.2530499926
## OCC_CODE21  -0.4345813200
## OCC_CODE22  -0.4648495646
## IND_CODE2   -0.0262041453
## IND_CODE3    0.2415106961
## IND_CODE4    .
## IND_CODE5    0.0073255674
## IND_CODE6   -0.0222780101
## IND_CODE7    0.1127708032
## IND_CODE8    0.1193878366
## IND_CODE9    0.0740564236
## IND_CODE10   0.1008904694
## IND_CODE11   0.0724777560
## IND_CODE12  -0.0750241164
## IND_CODE13   .
## IND_CODE14  -0.0084171698
## IND_CODE15   0.0195359418
## IND_CODE16  -0.0215698409
## IND_CODE17   0.0605987354
## IND_CODE18   0.1978994495
## IND_CODE19   0.1568444089
## IND_CODE20  -0.1500394527
## IND_CODE21   0.0418165903
## IND_CODE22  -0.1383033009
## IND_CODE23   0.0339773784
## IND_CODE24   0.1658700038
## IND_CODE25   0.2326363404
## IND_CODE26   0.1157002295
## IND_CODE27   0.2473913601
## IND_CODE29   0.1180650986
## IND_CODE30   0.1591036055
## IND_CODE31   .
## IND_CODE32   0.1449572940
## IND_CODE33   0.0182512544
## IND_CODE35   .
## IND_CODE36   0.1173787436
## IND_CODE37   0.0615662743
## IND_CODE38  -0.1146523215
## IND_CODE39   0.0453147356
## IND_CODE40  -0.1382562182
## IND_CODE41  -0.0247423349
## IND_CODE42  -0.0127539997
## IND_CODE43  -0.2179079674
## IND_CODE44  -0.1376543066
## IND_CODE45  -0.1722616666
## IND_CODE46  -0.2016343020
## IND_CODE47  -0.1929582806
## IND_CODE48  -0.0595795954
```

```
## IND_CODE49  -0.1597056513
## IND_CODE50  -0.4007666283
## IND_CODE51   0.0612021354
## EDUC         0.0102533112
## UHRSWORKT    0.0127088746
```

```r
selected_features1 <- which(abs(coef_selected1[-1]) != 0)
x1 <- x1[, c(selected_features1)]

fit2 <- cv.glmnet(x2, y2, family="gaussian")
coef_selected2 <- coef(fit2, s="lambda.min")
print(coef_selected2)
```

```
## 150 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) -72335.56862
## (Intercept)       .
## MET_CODE         13.14425
## STATEFIP4     -3149.25163
## STATEFIP5     -7833.29925
## STATEFIP6      7558.07778
## STATEFIP8      4840.22909
## STATEFIP9     15565.73566
## STATEFIP10    -4055.27650
## STATEFIP11    24807.76377
## STATEFIP12    -3840.68002
## STATEFIP13    -1750.69126
## STATEFIP15    -7530.04360
## STATEFIP16    -3089.21432
## STATEFIP17     6264.59053
## STATEFIP18      -19.27852
## STATEFIP19     3257.22509
## STATEFIP20    14266.47022
## STATEFIP21    -8293.07978
## STATEFIP22   -11476.56451
## STATEFIP23    -4613.52814
## STATEFIP24     5011.13591
## STATEFIP25    10845.28359
## STATEFIP26    -1622.72011
## STATEFIP27     9289.69418
## STATEFIP28   -10861.97211
## STATEFIP29     -604.05765
## STATEFIP30    -1325.95319
## STATEFIP31       68.75162
## STATEFIP32    -1101.60360
## STATEFIP33     6199.61171
## STATEFIP34     5377.30318
## STATEFIP35   -11283.89953
## STATEFIP36     3837.99909
## STATEFIP37        .
## STATEFIP38    -2498.56761
## STATEFIP39    -5702.47764
## STATEFIP40    -8410.62941
## STATEFIP41     1822.22046
```

```
## STATEFIP42      986.88247
## STATEFIP44     -149.81111
## STATEFIP45    -1245.82129
## STATEFIP46   -10612.00376
## STATEFIP47    -4493.44880
## STATEFIP48          .
## STATEFIP49     6025.84581
## STATEFIP50          .
## STATEFIP51     7477.23693
## STATEFIP53    14970.12287
## STATEFIP54    -3771.85411
## STATEFIP55   -10411.09815
## AGE             962.66543
## SEX2         -20213.23861
## RACE200      -10417.36106
## RACE300       -7646.52559
## RACE651        2492.45361
## RACE652        -299.77396
## RACE801      -13080.50503
## RACE802       -6779.57646
## RACE803        7090.01295
## RACE804       -6007.08736
## RACE805      -16577.92414
## RACE806             .
## RACE807             .
## RACE808       38813.19558
## RACE809             .
## RACE810      -14383.95198
## RACE811       -2225.21152
## RACE812      -22096.46447
## RACE813       18442.67728
## RACE816       51183.31993
## RACE817             .
## RACE819       -9368.36183
## RACE820             .
## RACE830       21677.22388
## CITIZEN2      -2965.84639
## CITIZEN3      -5120.66085
## CITIZEN4      -3333.93102
## CITIZEN5      -1953.37690
## OCC_CODE2    -18640.04632
## OCC_CODE3     -6597.11377
## OCC_CODE4    -10242.76033
## OCC_CODE5    -29200.33324
## OCC_CODE6    -38005.57111
## OCC_CODE7     29604.21219
## OCC_CODE8    -27300.59931
## OCC_CODE9    -28945.91911
## OCC_CODE10          .
## OCC_CODE11   -38686.84748
## OCC_CODE12   -23246.34229
## OCC_CODE13   -28493.12788
## OCC_CODE14   -35288.16424
## OCC_CODE15   -29673.53172
```

```
## OCC_CODE16  -17577.41805
## OCC_CODE17  -36705.57867
## OCC_CODE18  -35749.23419
## OCC_CODE19  -34103.18469
## OCC_CODE20  -31028.16491
## OCC_CODE21  -40811.70681
## OCC_CODE22  -40139.23704
## IND_CODE2          .
## IND_CODE3    19951.44600
## IND_CODE4          .
## IND_CODE5    -3802.70671
## IND_CODE6     -985.06232
## IND_CODE7     2463.10926
## IND_CODE8    10194.90853
## IND_CODE9    11354.08121
## IND_CODE10    7171.98742
## IND_CODE11         .
## IND_CODE12   -3958.03045
## IND_CODE13         .
## IND_CODE14   -1169.13523
## IND_CODE15         .
## IND_CODE16    -306.01295
## IND_CODE17   -3109.07517
## IND_CODE18   10854.92275
## IND_CODE19   16296.64865
## IND_CODE20    -228.31144
## IND_CODE21    2312.08853
## IND_CODE22   -5443.31725
## IND_CODE23         .
## IND_CODE24   10816.27804
## IND_CODE25   21280.76995
## IND_CODE26    7172.37658
## IND_CODE27   36266.10622
## IND_CODE29    7768.93033
## IND_CODE30   13045.86588
## IND_CODE31         .
## IND_CODE32   32192.97425
## IND_CODE33    1585.75712
## IND_CODE35   -8937.67855
## IND_CODE36   18535.19710
## IND_CODE37    2227.88321
## IND_CODE38   -5770.59769
## IND_CODE39         .
## IND_CODE40  -19374.16274
## IND_CODE41    -609.95811
## IND_CODE42         .
## IND_CODE43  -14525.36936
## IND_CODE44  -11749.00279
## IND_CODE45  -11157.42876
## IND_CODE46  -18050.25001
## IND_CODE47  -10013.98928
## IND_CODE48   -4546.46899
## IND_CODE49  -16577.46122
## IND_CODE50  -15640.73855
```

```
## IND_CODE51         .
## EDUC             862.96870
## UHRSWORKT       1477.65553
```

```r
selected_features2 <- which(abs(coef_selected2[-1]) != 0)
x2 <- x2[, c(selected_features2)]
```

Model Building

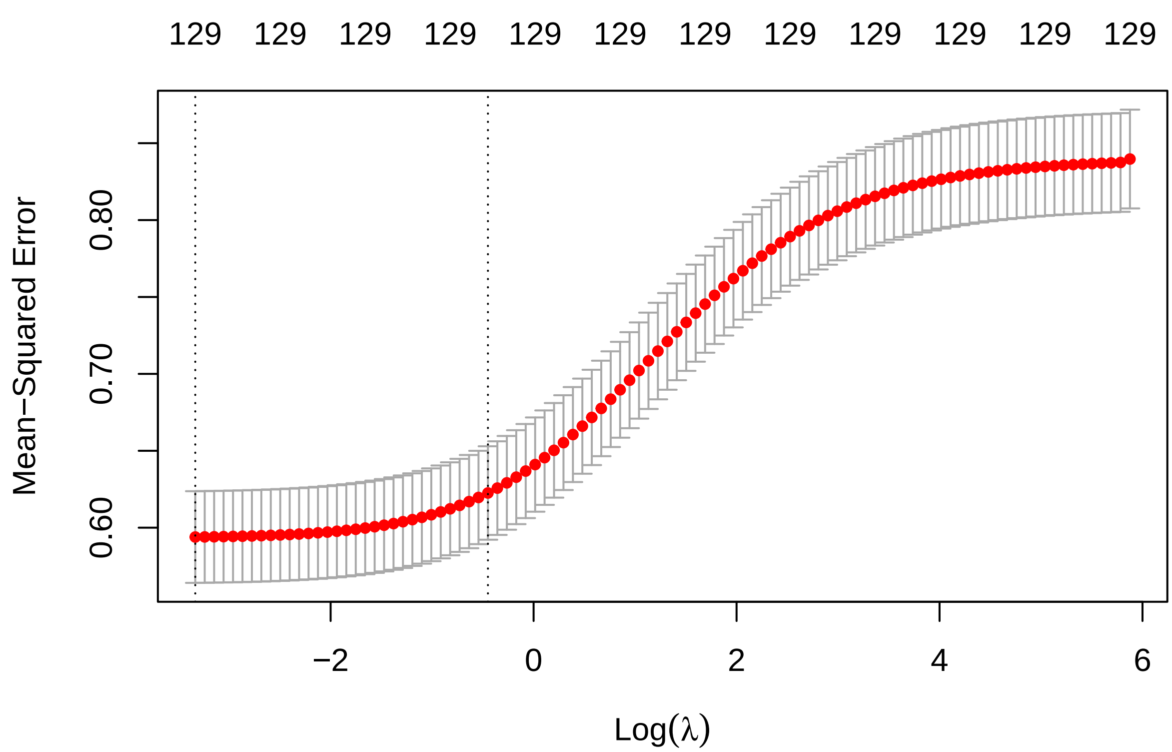
For income prediction, we are using RIDGE Regression for prediction. Further, we are considering 2 models, one with the RAW Income variable INCTOT and the second with the LOG transformed Income variable log_INCTOT. By using Ridge regression on both, we try to compare the performance and try to understand which is a better model for our usecase,

Model 1:

Here, we model for the Log Transformed model

```r
cv_ridge <- cv.glmnet(as.matrix(x1), y1, alpha = 0, type.measure = "mse", nfolds = 10)

# Plot the CV results to find the optimal lambda
plot(cv_ridge)
```



```r
cv_error <- cv_ridge$cvm

train_error <- mean(cv_error)
validation_error <- min(cv_error)  # This is the minimum error, which corresponds to the validation err

# Best lambda from CV
best_lambda <- cv_ridge$lambda.min
print(paste("Optimal lambda:", best_lambda))
```

```
## [1] "Optimal lambda: 0.0356564821437246"
```

```r
# Fit the model using the best lambda
final_ridge_model <- glmnet(x1, y1, alpha = 0, lambda = best_lambda, nfolds = 10, standardize = TRUE)

# Predict on Test Data
x_test1 <- test1[, c("MET_CODE", "STATEFIP", "AGE", "SEX", "RACE", "CITIZEN", "OCC_CODE", "IND_CODE", "I
x_test1 <- model.matrix(~., data = x_test1)
x_test1 <- x_test1[, c(selected_features1)]
y_test1 <- test1$log_INCTOT

predictions <- predict(final_ridge_model, newx = x_test1)
rsq <- 1 - sum((y_test1 - predictions)^2) / sum((y_test1 - mean(y_test1))^2)
mse_test <- mean((y_test1 - predictions)^2)
rmse_test <- sqrt(mse_test)
mae_test <- mean(abs(y_test1 - predictions))

# Print the results
print(paste("Train MSE:", train_error))
```

```
## [1] "Train MSE: 0.715235630941903"
```

```r
print(paste("Validation MSE:", validation_error))
```

```
## [1] "Validation MSE: 0.593914882875985"
```

```r
print(paste("R-squared: ", rsq))
```

```
## [1] "R-squared:  0.267658312012786"
```

```r
print(paste("Test MSE: ", mse_test))
```

```
## [1] "Test MSE:  0.669002228799646"
```

```r
print(paste("Test RMSE: ", rmse_test))
```

```
## [1] "Test RMSE:  0.817925564339229"
```

```r
print(paste("Test MAE: ", mae_test))
```

```
## [1] "Test MAE:  0.469130391910932"
```

Interpretation:

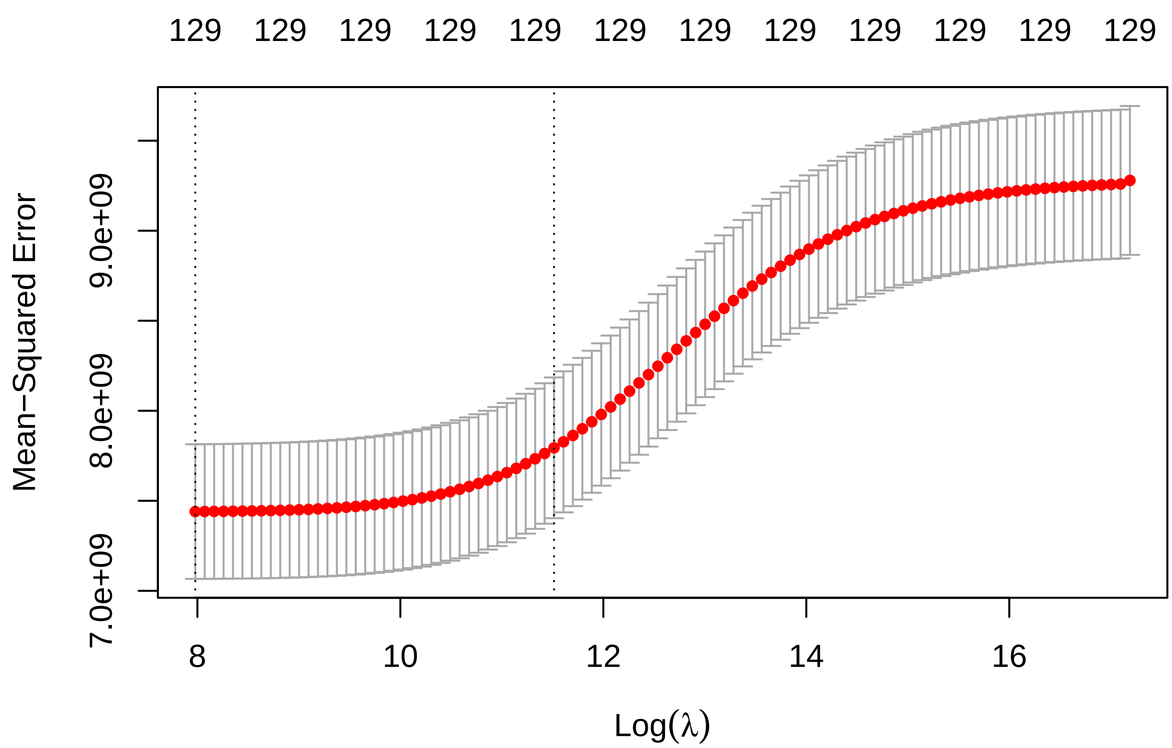
From the results above, we observe that we have an R^2 value of approximately 0.3 and moderate Train, Validation and Test Error. We can interpret that, on average, when using LOG_INCTOT, our model would be off by ~0.7 units when predicted the Wage of an individual given all other features.

Model 2:

Here we model for the Raw INCTOT variable

```r
cv_ridge2 <- cv.glmnet(as.matrix(x2), y2, alpha = 0, type.measure = "mse", nfolds = 10)

# Plot the CV results to find the optimal lambda
plot(cv_ridge2)
```




```r
cv_error2 <- cv_ridge2$cvm

train_error2 <- mean(cv_error2)
validation_error2 <- min(cv_error2)

# Best lambda from CV
best_lambda2 <- cv_ridge2$lambda.min
print(paste("Optimal lambda:", best_lambda2))
```

```
## [1] "Optimal lambda: 2918.94391295606"
```

```r
# Fit the model using the best lambda
final_ridge_model2 <- glmnet(x2, y2, alpha = 0, lambda = best_lambda2, nfolds = 10, standardize = TRUE)

# Predict on Test Data
x_test2 <- test2[, c("MET_CODE", "STATEFIP", "AGE", "SEX", "RACE", "CITIZEN", "OCC_CODE", "IND_CODE", "I
x_test2 <- model.matrix(~., data = x_test2)
x_test2 <- x_test2[, c(selected_features2)]
y_test2 <- test2$INCTOT

predictions2 <- predict(final_ridge_model2, newx = x_test2)
rsq2 <- 1 - sum((y_test2 - predictions2)^2) / sum((y_test2 - mean(y_test2))^2)
mse_test2 <- mean((y_test2 - predictions2)^2)
rmse_test2 <- sqrt(mse_test2)
mae_test2 <- mean(abs(y_test2 - predictions2))

# Print the results
print(paste("Train MSE:", train_error2))
```

```
## [1] "Train MSE: 8310636937.84522"
```

```r
print(paste("Validation MSE:", validation_error2))
```

```
## [1] "Validation MSE: 7440173342.40493"
```

```r
print(paste("R-squared: ", rsq2))
```

```
## [1] "R-squared:  0.20109606465489"
```

```r
print(paste("Test MSE: ", mse_test2))
```

```
## [1] "Test MSE:  7032242512.64366"
```

```r
print(paste("Test RMSE: ", rmse_test2))
```

```
## [1] "Test RMSE:  83858.4671493801"
```

```r
print(paste("Test MAE: ", mae_test2))
```

```
## [1] "Test MAE:  40548.9635521988"
```

Interpretation:

From the results above, we observe that we have an R^2 value of approximately 0.2 and extremely high Train, Validation and Test Error, so much so that they don't quite make sense in the context of the wages that we have in the dataset.

CONCLUSION

Based on above results, it is evident that our Log transformed model performs much better as compared to the raw INCTOT feature for our given dataset.