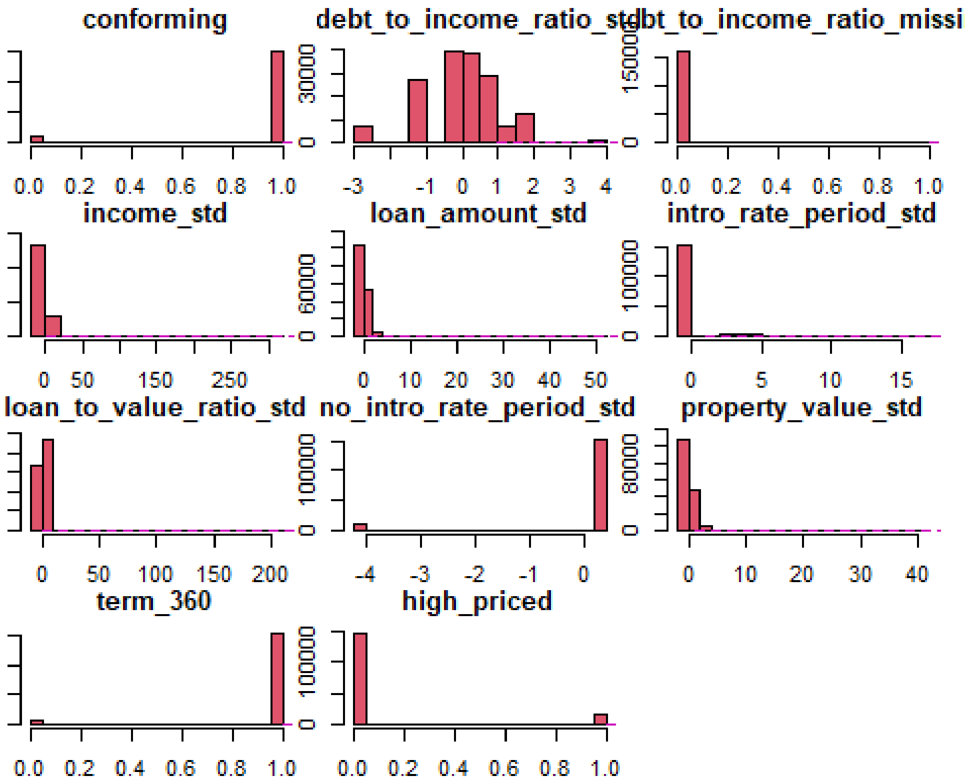
RML_Assignment 1

Tivon Johnson
Minhye Kim
Zach Vila
Qunzhe Ding

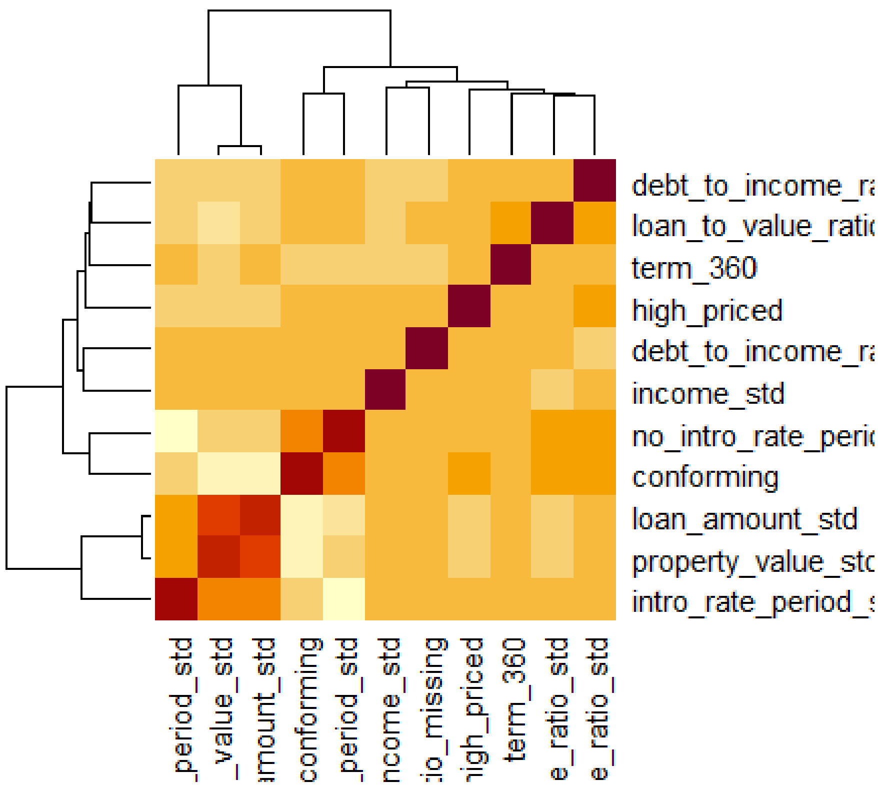
5/28/2021

```
hmda_train <- read.csv("hmda_train_preprocessed.csv", header = TRUE)
hmda_test <- read.csv("hmda_test_preprocessed.csv", header = TRUE)

names(hmda_train)

##  [1] "row_id"                         "black"
##  [3] "asian"                          "white"
##  [5] "amind"                          "hipac"
##  [7] "hispanic"                       "non_hispanic"
##  [9] "male"                           "female"
## [11] "agegte62"                       "agelt62"
## [13] "term_360"                       "conforming"
## [15] "debt_to_income_ratio_missing" "loan_amount_std"
## [17] "loan_to_value_ratio_std"        "no_intro_rate_period_std"
## [19] "intro_rate_period_std"          "property_value_std"
## [21] "income_std"                     "debt_to_income_ratio_std"
## [23] "high_priced"

#Create new data frame containing the pertinent variables.
attach(hmda_train)
new_data <- data.frame(conforming, debt_to_income_ratio_std, debt_to_income_r
atio_missing, income_std, loan_amount_std, intro_rate_period_std,
                                  loan_to_value_ratio_std, no_intro_rate_peri
od_std, property_value_std, term_360, high_priced)

attach(new_data)

## The following objects are masked from hmda_train:
##
##     conforming, debt_to_income_ratio_missing, debt_to_income_ratio_std,
##     high_priced, income_std, intro_rate_period_std, loan_amount_std,
##     loan_to_value_ratio_std, no_intro_rate_period_std,
##     property_value_std, term_360

library(ggplot2)
library(plyr)
library(psych)

##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
multi.hist(new_data, dcol=6, bcol=2, freq=TRUE)
```




```
cor <- cor(new_data)
heatmap(cor)
```




```
corPlot(new_data, numbers=FALSE)

str(new_data)

## 'data.frame':    160338 obs. of  11 variables:
##  $ conforming                    : int  1 1 1 1 1 1 1 1 1 0 ...
##  $ debt_to_income_ratio_std      : num  0.855 -0.425 0.123 -0.425 0.763 ...
##  $ debt_to_income_ratio_missing: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ income_std                    : num  -0.0403 -0.0181 -0.0323 -0.0181 -0.0
382 ...
##  $ loan_amount_std               : num  -0.5144 -0.1186 -0.7782 -0.0747 -0.6
023 ...
##  $ intro_rate_period_std         : num  -0.215 -0.215 4.611 -0.215 -0.215 ..
.
##  $ loan_to_value_ratio_std       : num  0.334 0.269 0.229 -1.15 0.553 ...
##  $ no_intro_rate_period_std      : num  0.244 0.244 -4.092 0.244 0.244 ...
##  $ property_value_std            : num  -0.536 -0.228 -0.721 0.358 -0.628 ..
.
##  $ term_360                      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ high_priced                   : int  0 0 0 0 0 0 0 0 0 0 ...

#Convert binary variables to factor
new_data$high_priced <- as.factor(new_data$high_priced)
new_data$term_360 <- as.factor(new_data$term_360)
new_data$debt_to_income_ratio_missing <- as.factor(new_data$debt_to_income_ra
tio_missing)
new_data$conforming <- as.factor(new_data$conforming)
```

```r
library(caret)

## Warning: package 'caret' was built under R version 4.0.3

## Loading required package: lattice

library(glmnet)

## Warning: package 'glmnet' was built under R version 4.0.4

## Loading required package: Matrix

## Warning: package 'Matrix' was built under R version 4.0.5

## Loaded glmnet 4.1-1

library(lattice)
#Split new data into 70/30 training and test set.
set.seed(1)
trainIndex <- createDataPartition(new_data$high_priced, p=0.7, list=F)
new_train <- new_data[trainIndex, ]
new_test <- new_data[-trainIndex, ]

#Set up 5-fold cross-validation method
set.seed(3)
cv_5 <- trainControl(method="cv", number=5)

#Fit elastic net with tuning grid
train_elnet <- train(high_priced~., data=new_train, method = "glmnet", trCont
rol = cv_5)

train_elnet

## glmnet
##
## 112238 samples
##     10 predictor
##      2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 89790, 89791, 89791, 89790, 89790
## Resampling results across tuning parameters:
##
##   alpha  lambda        Accuracy   Kappa
##   0.10   8.136002e-05  0.9026177  0.0020423738
##   0.10   8.136002e-04  0.9027424  0.0019985048
##   0.10   8.136002e-03  0.9032413  0.0010799528
##   0.55   8.136002e-05  0.9025553  0.0019176604
##   0.55   8.136002e-04  0.9028048  0.0019764199
##   0.55   8.136002e-03  0.9032324  0.0006176217
##   1.00   8.136002e-05  0.9025286  0.0017178086
```

```
##   1.00   8.136002e-04  0.9028849  0.0021365892
##   1.00   8.136002e-03  0.9032235  0.0001545347
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.008136
002.
```

```
#Highest accuracy is 0.90324 at alpha 0.1

#Expand feature space with larger tuning grid
#train_elnet_int <- train(high_priced~.^2, data=new_train, method = "glmnet",
trControl = cv_5, tuneLength = 10)

#Function to get best result
#get_best_result <- function(caret_fit) {
  #best <- which(rownames(caret_fit$results) == rownames(caret_fit$bestTune))
  #best_result <- caret_fit$results[best, ]
  #rownames(best_result) = NULL
  #best_result
#}

#get_best_result(train_elnet_int)

#Function to calculate accuracy
calc_acc <- function(actual, predicted){
  mean(actual == predicted)
}

#Calculate accuracy
calc_acc(actual = new_test$high_priced, predicted = predict(train_elnet, newd
ata = new_test))
```

```
## [1] 0.9032432
```

```
library(h2o)
```

```
## Warning: package 'h2o' was built under R version 4.0.5
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'

## The following objects are masked from 'package:stats':
##
##     cor, sd, var

## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc

h2o.init()

##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         19 minutes 7 seconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.32.1.3
##     H2O cluster version age:    11 days
##     H2O cluster name:           H2O_started_from_R_Admin_ikc867
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.79 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, TargetE
ncoder, Core V4
##     R Version:                  R version 4.0.2 (2020-06-22)

#Create feature names
y <- "high_priced"
x <- setdiff(names(new_train), y)

#Turn training set into h2o object
train.h2o <- as.h2o(new_train)

## Warning in use.package("data.table"): data.table cannot be used without R
## package bit64 version 0.9.7 or higher. Please upgrade to take advangage of
## data.table speedups.

##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

```r
#Training basic GBM model with default parameters
h2o.fit1 <- h2o.gbm(x = x,
                    y = y,
                    training_frame = train.h2o,
                    nfolds = 5)
```

```
##   |
|                                                                      |   0%
|
|=                                                                     |   1%
|
|======                                                                |   8%
|
|======================                                                |  32%
|
|==========================                                            |  38%
|
|=============================                                         |  42%
|
|===============================================                       |  67%
|
|=================================================                     |  70%
|
|===================================================                   |  72%
|
|====================================================                  |  75%
|
|=======================================================               |  79%
|
|==========================================================            |  83%
|
|=============================================================         |  87%
|
|===============================================================       |  90%
|
|=================================================================     |  93%
|
|===================================================================== |  98%
|
|======================================================================| 100%
```

```r
#Model results
h2o.fit1
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: gbm
## Model ID:  GBM_model_R_1622429036451_1410
## Model Summary:
```

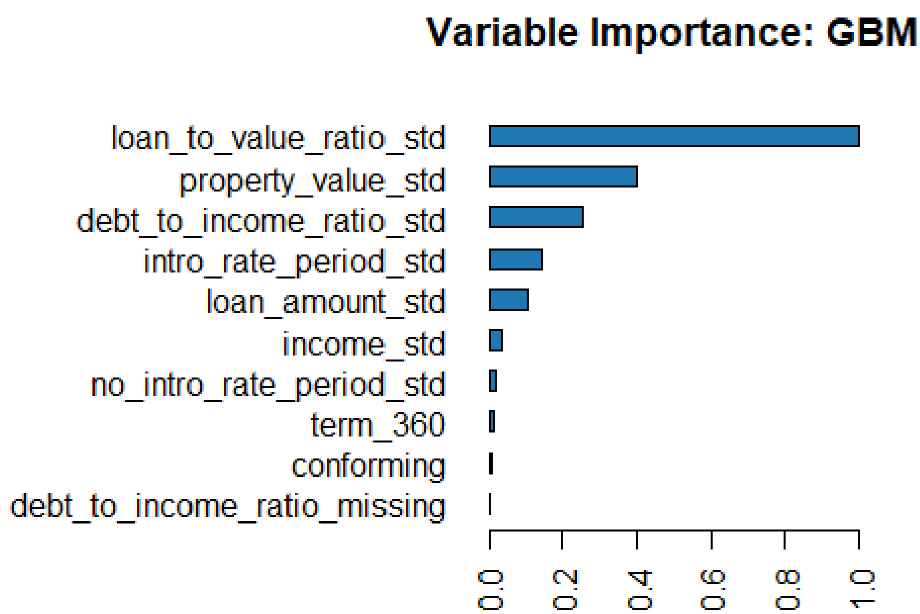
```
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1              50                       50               21901         5
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1         5    5.00000         25         32    30.18000
##
##
## H2OBinomialMetrics: gbm
## ** Reported on training data. **
##
## MSE:  0.07523272
## RMSE:  0.2742858
## LogLoss:  0.2506421
## Mean Per-Class Error:  0.2784423
## AUC:  0.8296986
## AUCPR:  0.3119291
## Gini:  0.6593971
## R^2:  0.1387537
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal thre
shold:
##            0     1    Error            Rate
## 0      83175 18209 0.179604  =18209/101384
## 1       4095  6759 0.377280    =4095/10854
## Totals 87270 24968 0.198721  =22304/112238
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold         value idx
## 1                       max f1  0.188555      0.377366 187
## 2                       max f2  0.111298      0.541478 262
## 3                 max f0point5  0.259006      0.329960 121
## 4                 max accuracy  0.420162      0.904266  33
## 5                max precision  0.833293      1.000000   0
## 6                   max recall  0.005673      1.000000 399
## 7              max specificity  0.833293      1.000000   0
## 8             max absolute_mcc  0.140438      0.328062 233
## 9   max min_per_class_accuracy  0.154566      0.751981 220
## 10 max mean_per_class_accuracy  0.095350      0.765245 276
## 11                     max tns  0.833293 101384.000000   0
## 12                     max fns  0.833293  10853.000000   0
## 13                     max fps  0.005673 101384.000000 399
## 14                     max tps  0.005673  10854.000000 399
## 15                     max tnr  0.833293      1.000000   0
## 16                     max fnr  0.833293      0.999908   0
## 17                     max fpr  0.005673      1.000000 399
## 18                     max tpr  0.005673      1.000000 399
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.ga
insLift(<model>, valid=<T/F>, xval=<T/F>)`
##
## H2OBinomialMetrics: gbm
```

```
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined
holdout predictions) **
##
## MSE:  0.07610695
## RMSE:  0.2758749
## LogLoss:  0.2540164
## Mean Per-Class Error:  0.2821288
## AUC:  0.8221344
## AUCPR:  0.2897182
## Gini:  0.6442688
## R^2:  0.1287457
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal thre
shold:
##            0     1    Error           Rate
## 0      82222 19162 0.189004  =19162/101384
## 1       4073  6781 0.375253    =4073/10854
## Totals 86295 25943 0.207015  =23235/112238
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold         value idx
## 1                       max f1  0.182744      0.368563 199
## 2                       max f2  0.100692      0.535909 276
## 3                 max f0point5  0.254271      0.317754 131
## 4                 max accuracy  0.477167      0.903633  28
## 5                max precision  0.854327      1.000000   0
## 6                   max recall  0.005457      1.000000 399
## 7              max specificity  0.854327      1.000000   0
## 8             max absolute_mcc  0.136470      0.318991 245
## 9   max min_per_class_accuracy  0.150206      0.745473 232
## 10 max mean_per_class_accuracy  0.100692      0.760137 276
## 11                     max tns  0.854327 101384.000000   0
## 12                     max fns  0.854327  10853.000000   0
## 13                     max fps  0.005457 101384.000000 399
## 14                     max tps  0.005457  10854.000000 399
## 15                     max tnr  0.854327      1.000000   0
## 16                     max fnr  0.854327      0.999908   0
## 17                     max fpr  0.005457      1.000000 399
## 18                     max tpr  0.005457      1.000000 399
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.ga
insLift(<model>, valid=<T/F>, xval=<T/F>)`
## Cross-Validation Metrics Summary:
##                               mean           sd cv_1_valid  cv_2_valid
## accuracy                 0.7894789  0.010866088 0.77214056   0.7953426
## auc                     0.82226807  0.004802158 0.81496036  0.82605195
## err                     0.21052107  0.010866088 0.22785941  0.20465735
## err_count                   4725.2    232.05754     5102.0      4614.0
## f0point5                0.29558888 0.0055195773 0.28726012  0.29427335
```

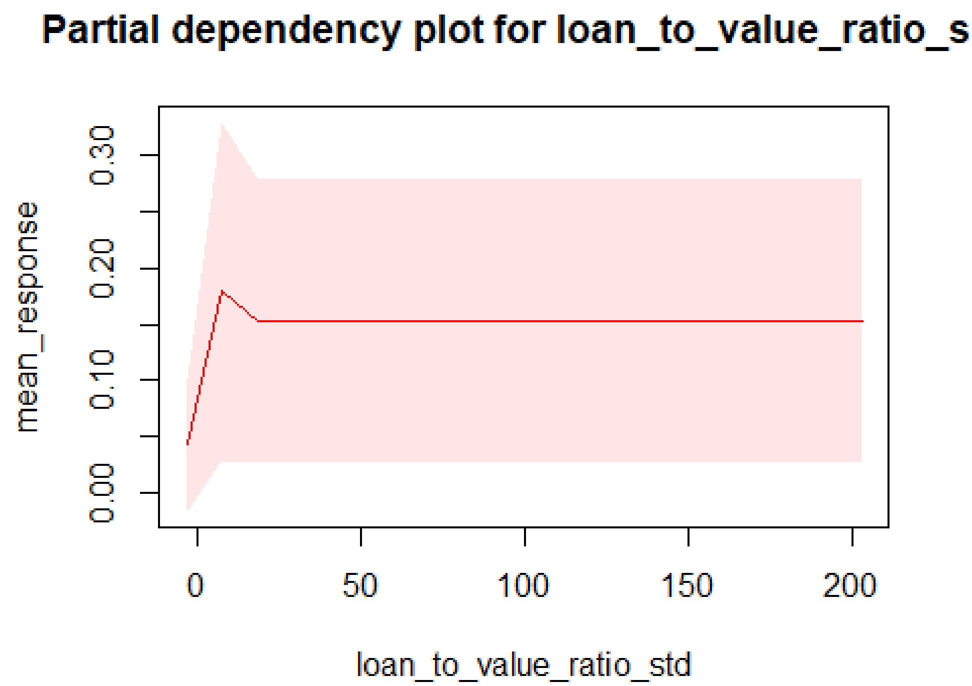
```
## f1                       0.37004244 0.0036663387 0.36652595  0.36638287
## f2                       0.49493015 0.0102767525  0.5062076   0.4853027
## lift_top_group             4.568642   0.37506232  4.1362686    4.770164
## logloss                  0.25401807 0.0030391607  0.2578716  0.25045457
## max_per_class_error       0.3607638   0.02901928  0.3213793   0.3806871
## mcc                      0.30914396 0.0045944043 0.31014138  0.30382285
## mean_per_class_accuracy   0.7223964  0.007301601 0.73041147   0.7166252
## mean_per_class_error     0.27760363  0.007301601 0.26958856   0.2833748
## mse                     0.076107666  9.033316E-4  0.0771715 0.074965365
## pr_auc                     0.290607  0.009188015 0.27514684  0.29937336
## precision                0.26065308 0.0062621064  0.2510631  0.26014042
## r2                       0.12874511 0.0051593487 0.12006736  0.13248512
## recall                    0.6392362   0.02901928  0.6786207   0.6193129
## rmse                      0.2758723 0.0016371785 0.27779758  0.27379805
## specificity              0.80555654  0.015034443 0.78220224  0.81393754
##                          cv_3_valid  cv_4_valid cv_5_valid
## accuracy                  0.7854873  0.79814756 0.79627657
## auc                       0.8267376   0.8203637 0.82322675
## err                      0.21451274  0.20185243 0.20372345
## err_count                    4792.0      4533.0     4585.0
## f0point5                 0.29586166  0.29843378 0.30211547
## f1                       0.37326705  0.36980397  0.3742323
## f2                        0.5055264  0.48604006 0.49157405
## lift_top_group             5.041891    4.245223  4.6496644
## logloss                  0.25151825  0.25452825  0.2557177
## max_per_class_error      0.33812615  0.38511327 0.37851316
## mcc                      0.31588987   0.3061047 0.30976096
## mean_per_class_accuracy   0.7302829  0.71628344  0.7183789
## mean_per_class_error      0.2697171   0.2837166  0.2816211
## mse                     0.075652964 0.075885095 0.07686341
## pr_auc                    0.2933567  0.29077986 0.29437825
## precision                0.25992715   0.2644135 0.26772115
## r2                       0.13240124  0.12816203 0.13060981
## recall                    0.6618738  0.61488676 0.62148684
## rmse                     0.27505085   0.2754725  0.2772425
## specificity                0.798692   0.8176801 0.81527096
```

```
#Train model with 5000 trees
#h2o.fit2 <- h2o.gbm(
  #x = x,
  #y = y,
  #training_frame = train.h2o,
  #nfolds = 5,
  #ntrees = 5000,
  #stopping_rounds = 10, #stop training after 10 consecutive trees have no im
provement in cv error
  #stopping_tolerance = 0,
  #seed = 123
#)
```

```r
#Model stopped after 2533 trees
#h2o.fit2@parameters$ntrees

#h2o.auc(h2o.fit2, xval = TRUE)
# AUC shows no significant improvement over first model.

#Plot variable importance
h2o.varimp_plot(h2o.fit1, num_of_features = 10)
```




```r
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "loan_to_value_ratio_std")

##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

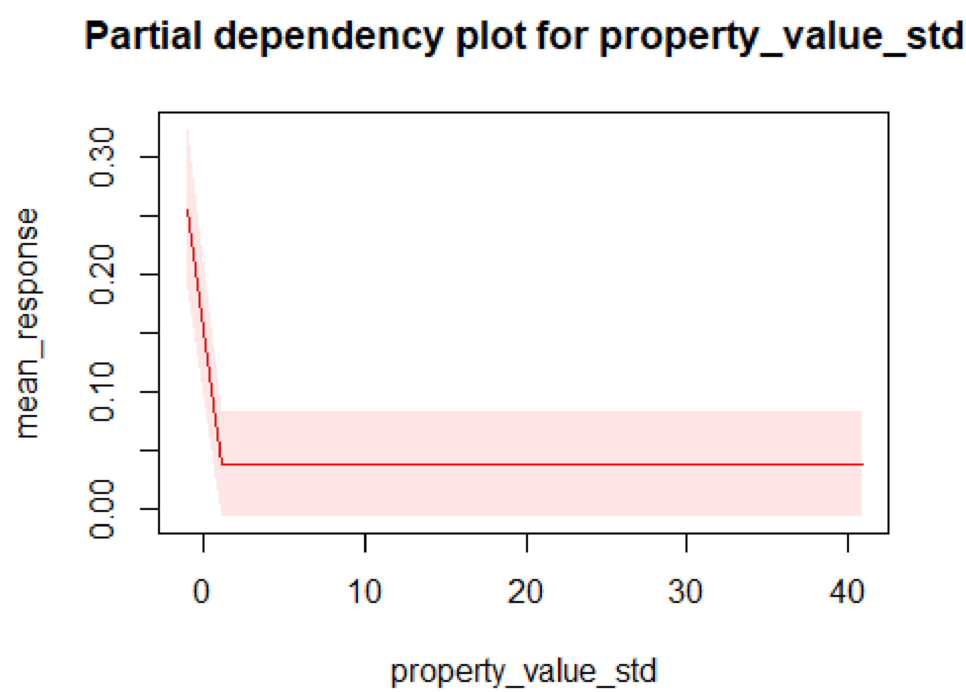



```
## PartialDependence: Partial dependency plot for loan_to_value_ratio_std
##    loan_to_value_ratio_std mean_response stddev_response
## 1                -3.478615      0.042561        0.058386
## 2                 7.375801      0.179413        0.149913
## 3                18.230216      0.152956        0.126701
## 4                29.084632      0.152956        0.126701
## 5                39.939048      0.152956        0.126701
## 6                50.793464      0.152956        0.126701
## 7                61.647879      0.152956        0.126701
## 8                72.502295      0.152956        0.126701
## 9                83.356711      0.152956        0.126701
## 10               94.211126      0.152956        0.126701
## 11              105.065542      0.152956        0.126701
## 12              115.919958      0.152956        0.126701
## 13              126.774374      0.152956        0.126701
## 14              137.628789      0.152956        0.126701
## 15              148.483205      0.152956        0.126701
## 16              159.337621      0.152956        0.126701
## 17              170.192036      0.152956        0.126701
## 18              181.046452      0.152956        0.126701
## 19              191.900868      0.152956        0.126701
## 20              202.755284      0.152956        0.126701
##    std_error_mean_response
## 1                 0.000174
## 2                 0.000447
## 3                 0.000378
## 4                 0.000378
```

```
## 5                      0.000378
## 6                      0.000378
## 7                      0.000378
## 8                      0.000378
## 9                      0.000378
## 10                     0.000378
## 11                     0.000378
## 12                     0.000378
## 13                     0.000378
## 14                     0.000378
## 15                     0.000378
## 16                     0.000378
## 17                     0.000378
## 18                     0.000378
## 19                     0.000378
## 20                     0.000378
```

```
#Plot partial dependence for property_value_std
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "property_value_std")
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

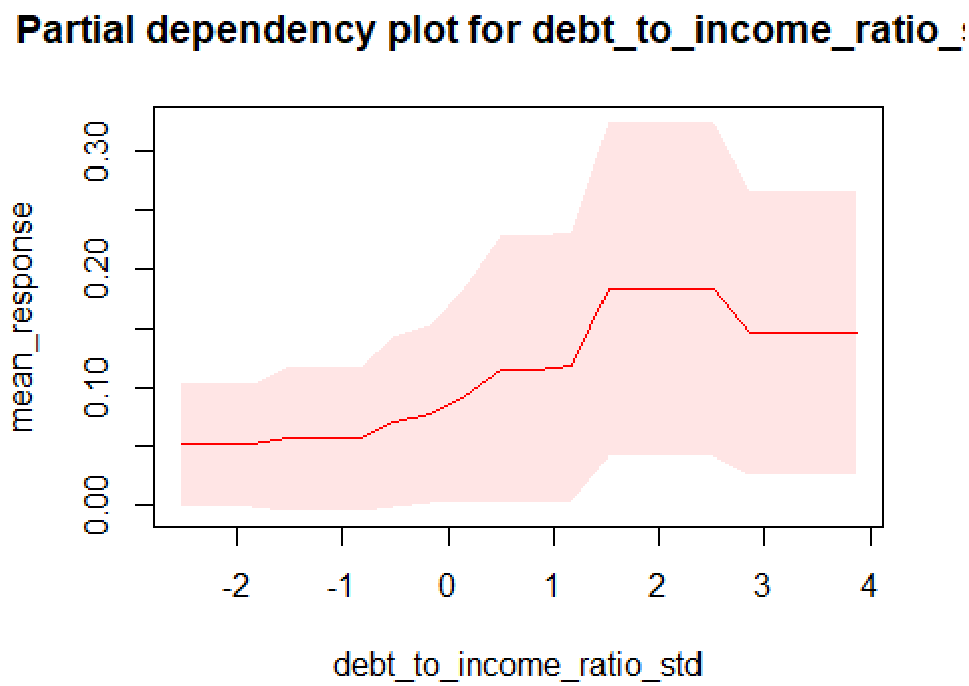



```
## PartialDependence: Partial dependency plot for property_value_std
##    property_value_std mean_response stddev_response std_error_mean_respons
```

```
e
## 1           -1.090958      0.255675        0.068498                  0.00020
4
## 2            1.116162      0.038323        0.045015                  0.00013
4
## 3            3.323283      0.038150        0.044321                  0.00013
2
## 4            5.530404      0.038098        0.044222                  0.00013
2
## 5            7.737525      0.038098        0.044222                  0.00013
2
## 6            9.944646      0.038074        0.044179                  0.00013
2
## 7           12.151767      0.038074        0.044179                  0.00013
2
## 8           14.358888      0.038074        0.044179                  0.00013
2
## 9           16.566008      0.038074        0.044179                  0.00013
2
## 10          18.773129      0.038074        0.044179                  0.00013
2
## 11          20.980250      0.038074        0.044179                  0.00013
2
## 12          23.187371      0.038074        0.044179                  0.00013
2
## 13          25.394492      0.038074        0.044179                  0.00013
2
## 14          27.601613      0.038074        0.044179                  0.00013
2
## 15          29.808733      0.038074        0.044179                  0.00013
2
## 16          32.015854      0.038074        0.044179                  0.00013
2
## 17          34.222975      0.038074        0.044179                  0.00013
2
## 18          36.430096      0.038074        0.044179                  0.00013
2
## 19          38.637217      0.038074        0.044179                  0.00013
2
## 20          40.844338      0.038074        0.044179                  0.00013
2
```

```
#Plot partial dependence for debt_to_income_ratio_std
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "debt_to_income_ratio_std"
)
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

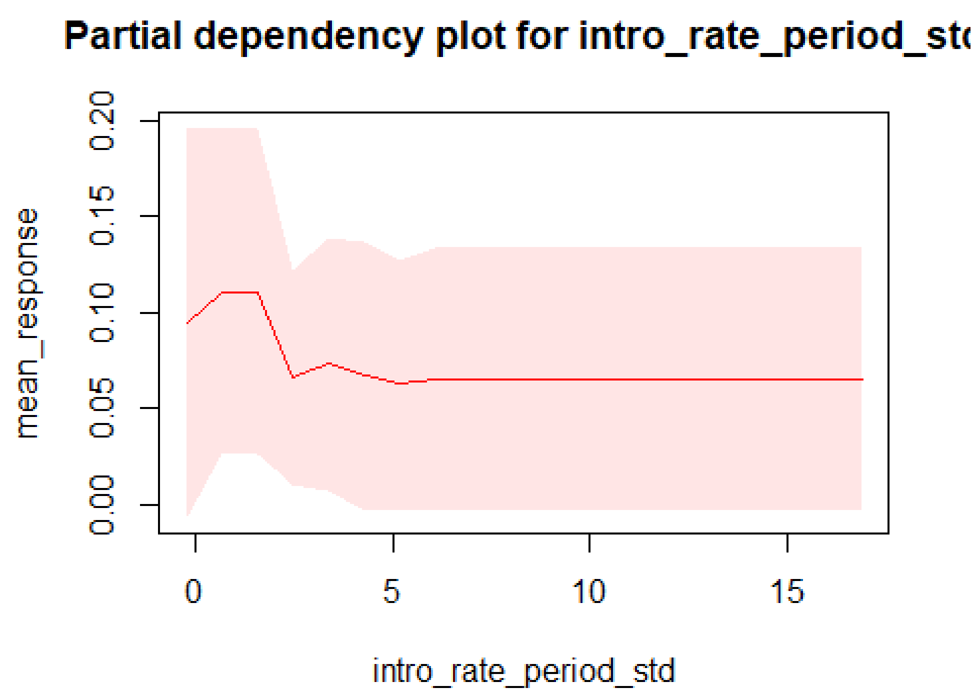



```
## PartialDependence: Partial dependency plot for debt_to_income_ratio_std
##    debt_to_income_ratio_std mean_response stddev_response
## 1                 -2.527547      0.050752        0.052628
## 2                 -2.190775      0.050752        0.052628
## 3                 -1.854004      0.050752        0.052628
## 4                 -1.517232      0.056119        0.061167
## 5                 -1.180461      0.056119        0.061167
## 6                 -0.843690      0.056119        0.061167
## 7                 -0.506918      0.069838        0.071677
## 8                 -0.170147      0.076971        0.075162
## 9                  0.166625      0.093299        0.092250
## 10                 0.503396      0.114812        0.113338
## 11                 0.840168      0.114664        0.113087
## 12                 1.176939      0.117609        0.114516
## 13                 1.513710      0.182935        0.141383
## 14                 1.850482      0.182935        0.141383
## 15                 2.187253      0.182935        0.141383
## 16                 2.524025      0.182935        0.141383
## 17                 2.860796      0.145769        0.120734
## 18                 3.197568      0.145769        0.120734
## 19                 3.534339      0.145769        0.120734
## 20                 3.871110      0.145769        0.120734
##    std_error_mean_response
## 1                 0.000157
## 2                 0.000157
## 3                 0.000157
## 4                 0.000183
```

```
## 5                     0.000183
## 6                     0.000183
## 7                     0.000214
## 8                     0.000224
## 9                     0.000275
## 10                    0.000338
## 11                    0.000338
## 12                    0.000342
## 13                    0.000422
## 14                    0.000422
## 15                    0.000422
## 16                    0.000422
## 17                    0.000360
## 18                    0.000360
## 19                    0.000360
## 20                    0.000360
```

```
#Plot partial dependence for property_value_std
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "intro_rate_period_std")
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

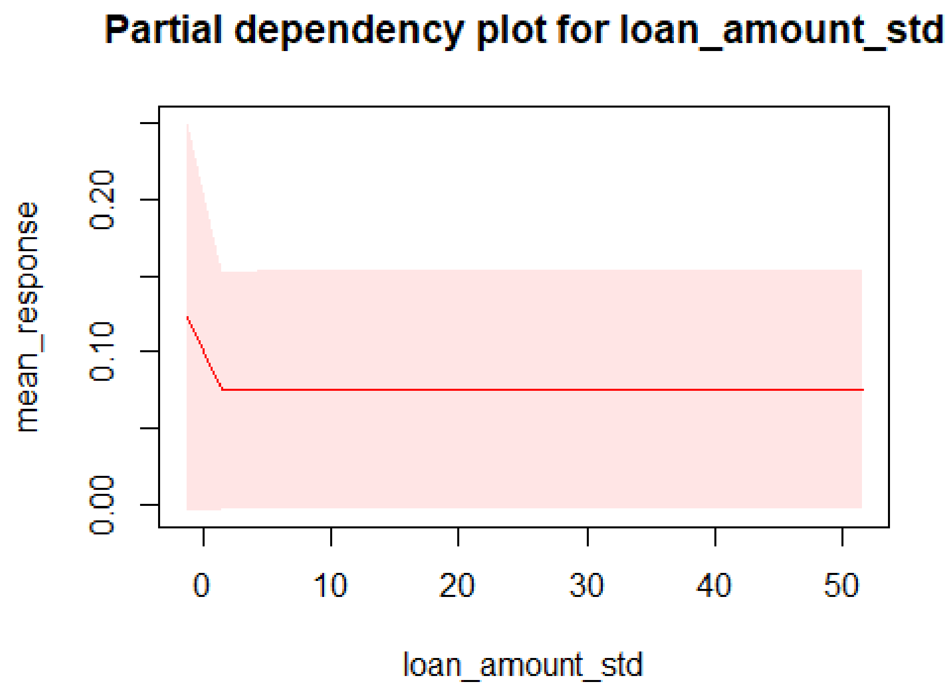



```
## PartialDependence: Partial dependency plot for intro_rate_period_std
##    intro_rate_period_std mean_response stddev_response std_error_mean_resp
```

```
onse
## 1              -0.215304      0.094275        0.100926                       0.00
0301
## 2               0.684309      0.110769        0.084960                       0.00
0254
## 3               1.583922      0.110725        0.084989                       0.00
0254
## 4               2.483536      0.066179        0.056031                       0.00
0167
## 5               3.383149      0.073119        0.065802                       0.00
0196
## 6               4.282762      0.066653        0.070457                       0.00
0210
## 7               5.182376      0.062585        0.064926                       0.00
0194
## 8               6.081989      0.065132        0.068516                       0.00
0205
## 9               6.981602      0.065132        0.068516                       0.00
0205
## 10              7.881216      0.065132        0.068516                       0.00
0205
## 11              8.780829      0.065132        0.068516                       0.00
0205
## 12              9.680442      0.065132        0.068516                       0.00
0205
## 13             10.580056      0.065132        0.068516                       0.00
0205
## 14             11.479669      0.065132        0.068516                       0.00
0205
## 15             12.379282      0.065132        0.068516                       0.00
0205
## 16             13.278896      0.065132        0.068516                       0.00
0205
## 17             14.178509      0.065132        0.068516                       0.00
0205
## 18             15.078122      0.065132        0.068516                       0.00
0205
## 19             15.977736      0.065132        0.068516                       0.00
0205
## 20             16.877349      0.065132        0.068516                       0.00
0205
```

```
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "loan_amount_std")
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

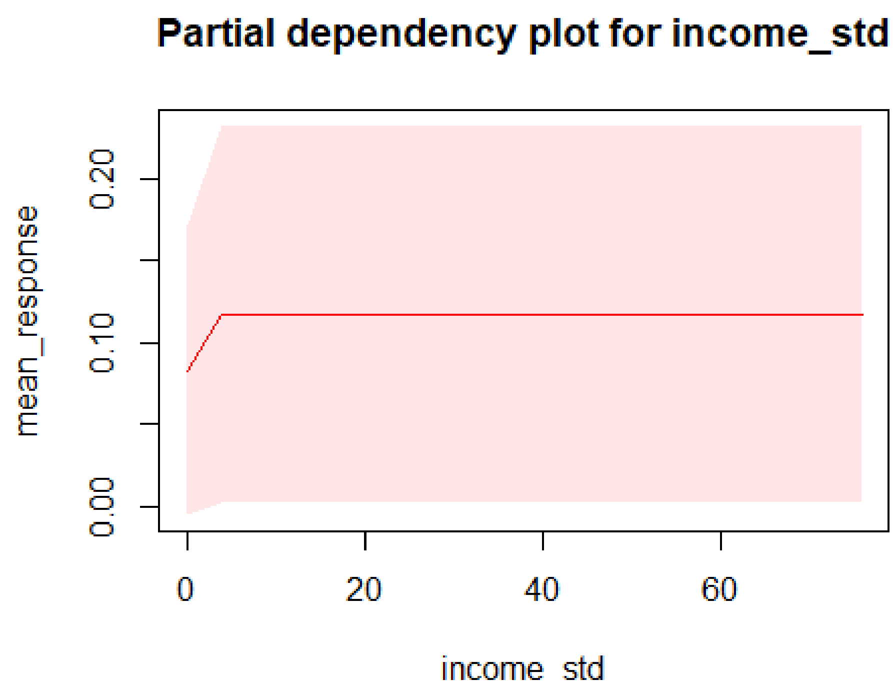



```
## PartialDependence: Partial dependency plot for loan_amount_std
##    loan_amount_std mean_response stddev_response std_error_mean_response
## 1        -1.261924      0.123158        0.126631                0.000378
## 2         1.515279      0.074907        0.076726                0.000229
## 3         4.292482      0.075522        0.077970                0.000233
## 4         7.069684      0.075522        0.077970                0.000233
## 5         9.846887      0.075522        0.077970                0.000233
## 6        12.624089      0.075522        0.077970                0.000233
## 7        15.401292      0.075522        0.077970                0.000233
## 8        18.178494      0.075522        0.077970                0.000233
## 9        20.955697      0.075522        0.077970                0.000233
## 10       23.732900      0.075522        0.077970                0.000233
## 11       26.510102      0.075522        0.077970                0.000233
## 12       29.287305      0.075522        0.077970                0.000233
## 13       32.064507      0.075522        0.077970                0.000233
## 14       34.841710      0.075522        0.077970                0.000233
## 15       37.618913      0.075522        0.077970                0.000233
## 16       40.396115      0.075522        0.077970                0.000233
## 17       43.173318      0.075522        0.077970                0.000233
## 18       45.950520      0.075522        0.077970                0.000233
## 19       48.727723      0.075522        0.077970                0.000233
## 20       51.504926      0.075522        0.077970                0.000233
```

```
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "income_std")
```

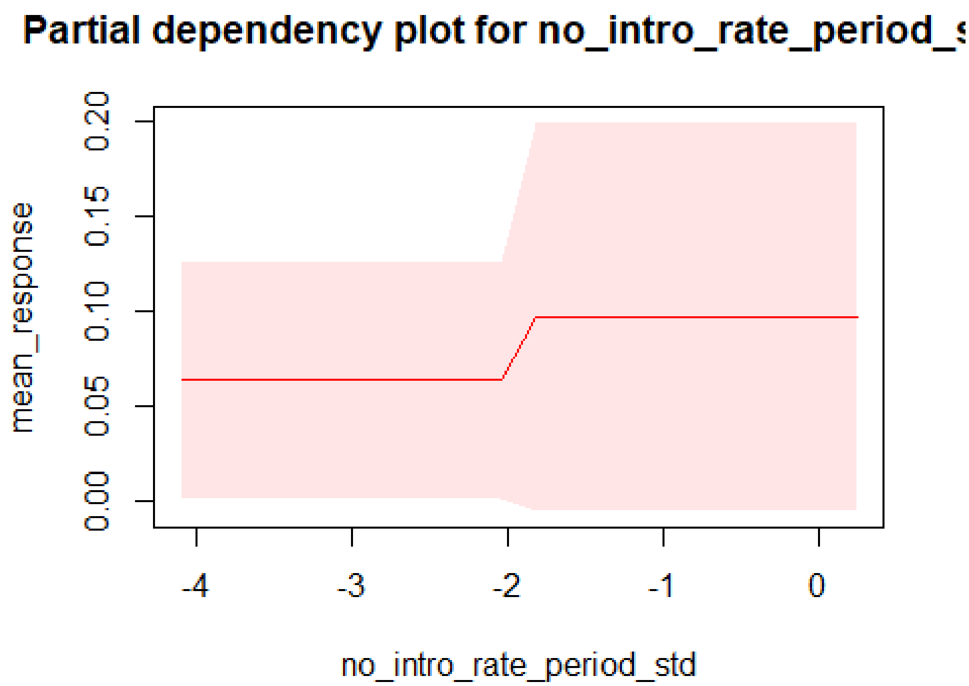
```
##   |
|                                                                      |   0%
```

```
|
|=====================================================================| 100%
```




```
## PartialDependence: Partial dependency plot for income_std
##    income_std mean_response stddev_response std_error_mean_response
## 1   -0.097823      0.083190        0.088220                0.000263
## 2    3.898190      0.117357        0.114860                0.000343
## 3    7.894204      0.117357        0.114860                0.000343
## 4   11.890218      0.117357        0.114860                0.000343
## 5   15.886231      0.117357        0.114860                0.000343
## 6   19.882245      0.117357        0.114860                0.000343
## 7   23.878258      0.117357        0.114860                0.000343
## 8   27.874272      0.117357        0.114860                0.000343
## 9   31.870286      0.117357        0.114860                0.000343
## 10  35.866299      0.117357        0.114860                0.000343
## 11  39.862313      0.117357        0.114860                0.000343
## 12  43.858326      0.117357        0.114860                0.000343
## 13  47.854340      0.117357        0.114860                0.000343
## 14  51.850354      0.117357        0.114860                0.000343
## 15  55.846367      0.117357        0.114860                0.000343
## 16  59.842381      0.117357        0.114860                0.000343
## 17  63.838394      0.117357        0.114860                0.000343
## 18  67.834408      0.117357        0.114860                0.000343
## 19  71.830422      0.117357        0.114860                0.000343
## 20  75.826435      0.117357        0.114860                0.000343
```

```r
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "no_intro_rate_period_std"
)

##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

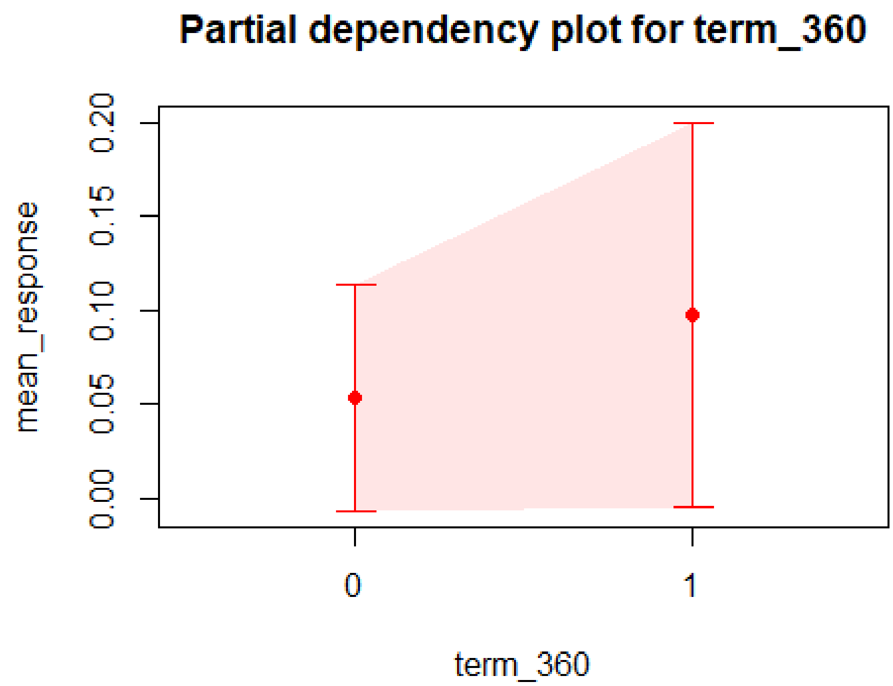



```
## PartialDependence: Partial dependency plot for no_intro_rate_period_std
##    no_intro_rate_period_std mean_response stddev_response
## 1                 -4.091747      0.063678        0.062087
## 2                 -3.863529      0.063678        0.062087
## 3                 -3.635311      0.063678        0.062087
## 4                 -3.407093      0.063678        0.062087
## 5                 -3.178875      0.063678        0.062087
## 6                 -2.950657      0.063678        0.062087
## 7                 -2.722439      0.063678        0.062087
## 8                 -2.494221      0.063678        0.062087
## 9                 -2.266003      0.063678        0.062087
## 10                -2.037785      0.063678        0.062087
## 11                -1.809567      0.097133        0.102361
## 12                -1.581349      0.097133        0.102361
## 13                -1.353131      0.097133        0.102361
## 14                -1.124913      0.097133        0.102361
## 15                -0.896695      0.097133        0.102361
## 16                -0.668477      0.097133        0.102361
## 17                -0.440259      0.097133        0.102361
```

```
## 18                -0.212042      0.097133        0.102361
## 19                 0.016176      0.097133        0.102361
## 20                 0.244394      0.097133        0.102361
##    std_error_mean_response
## 1                 0.000185
## 2                 0.000185
## 3                 0.000185
## 4                 0.000185
## 5                 0.000185
## 6                 0.000185
## 7                 0.000185
## 8                 0.000185
## 9                 0.000185
## 10                0.000185
## 11                0.000306
## 12                0.000306
## 13                0.000306
## 14                0.000306
## 15                0.000306
## 16                0.000306
## 17                0.000306
## 18                0.000306
## 19                0.000306
## 20                0.000306
```

```r
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "term_360")
```

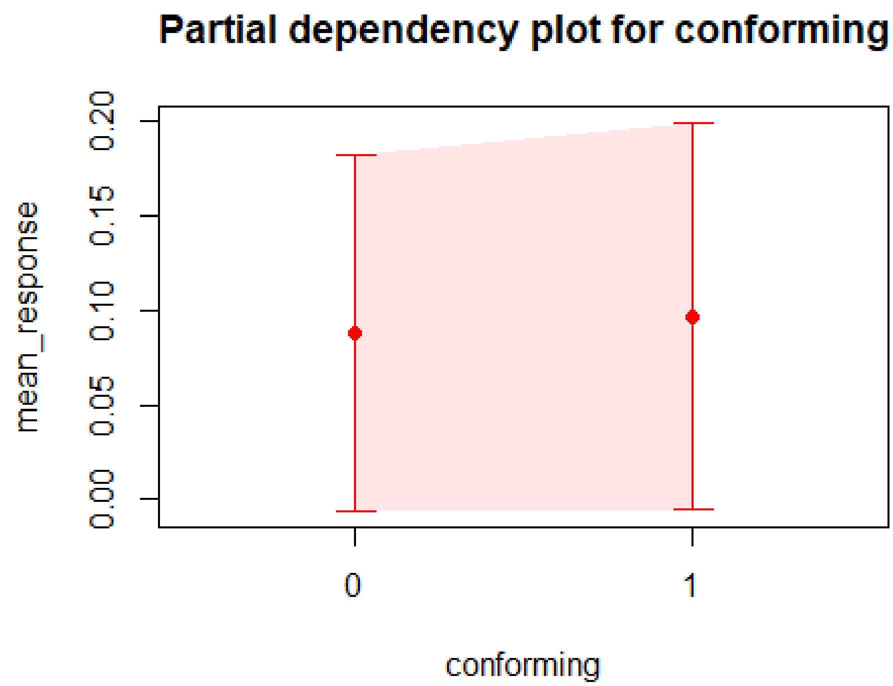
```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```




```
## PartialDependence: Partial dependency plot for term_360
##   term_360 mean_response stddev_response std_error_mean_response
## 1        0      0.053270        0.060098                0.000179
## 2        1      0.097434        0.102601                0.000306
```

```r
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "conforming")
```

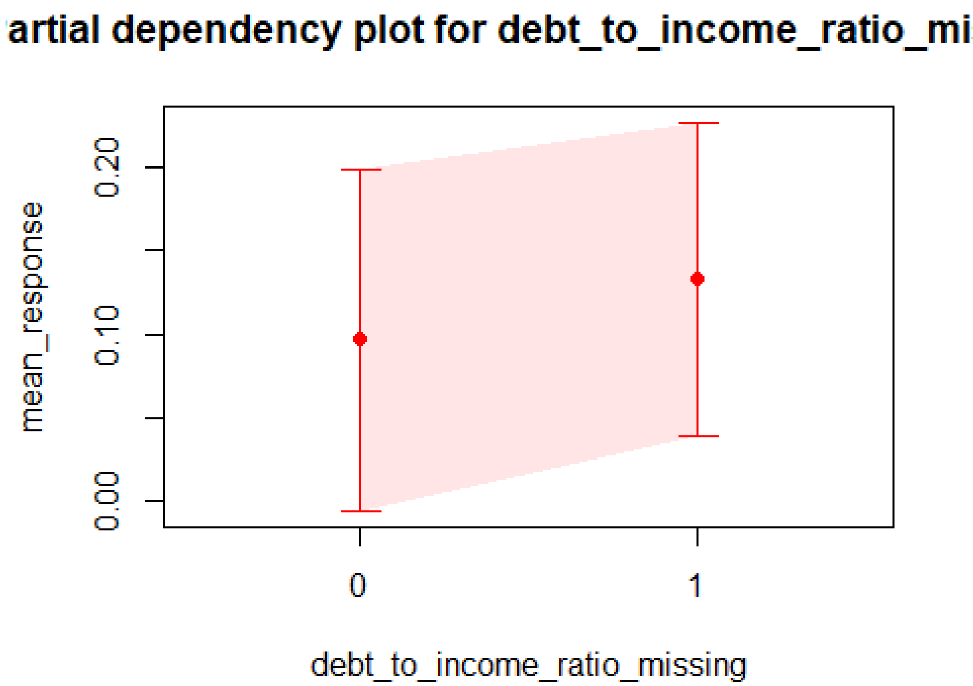
```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```




```
## PartialDependence: Partial dependency plot for conforming
##   conforming mean_response stddev_response std_error_mean_response
## 1          0      0.088092        0.094029                0.000281
## 2          1      0.097053        0.102435                0.000306
```

```
h2o.partialPlot(h2o.fit1, data = train.h2o, cols = "debt_to_income_ratio_miss
ing")
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```




```
## PartialDependence: Partial dependency plot for debt_to_income_ratio_missin
g
##   debt_to_income_ratio_missing mean_response stddev_response
## 1                            0      0.096768        0.102254
## 2                            1      0.132865        0.093492
##   std_error_mean_response
## 1                0.000305
## 2                0.000279
```

```
#Convert test set to h2o object
test.h2o <- as.h2o(new_test)
```

```
## Warning in use.package("data.table"): data.table cannot be used without R
## package bit64 version 0.9.7 or higher. Please upgrade to take advangage of
## data.table speedups.
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

```
#Evaluate performance on new data
h2o.performance(model = h2o.fit1, newdata = test.h2o)
```

```
## H2OBinomialMetrics: gbm
##
## MSE:  0.07640013
## RMSE:  0.2764057
```

```
## LogLoss:  0.2554305
## Mean Per-Class Error:  0.2843115
## AUC:  0.8181167
## AUCPR:  0.2875207
## Gini:  0.6362335
## R^2:  0.125302
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal thre
shold:
##             0     1    Error           Rate
## 0      34923  8526 0.196230   =8526/43449
## 1       1732  2919 0.372393    =1732/4651
## Totals 36655 11445 0.213264  =10258/48100
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold        value idx
## 1                       max f1  0.180082     0.362699 192
## 2                       max f2  0.107392     0.533937 262
## 3                 max f0point5  0.275193     0.307637 102
## 4                 max accuracy  0.420364     0.903701  28
## 5                max precision  0.521580     0.718750   9
## 6                   max recall  0.006484     1.000000 398
## 7              max specificity  0.621394     0.999977   0
## 8             max absolute_mcc  0.129899     0.316077 242
## 9   max min_per_class_accuracy  0.151591     0.742206 223
## 10 max mean_per_class_accuracy  0.096220     0.757929 271
## 11                     max tns  0.621394 43448.000000   0
## 12                     max fns  0.621394  4649.000000   0
## 13                     max fps  0.005735 43449.000000 399
## 14                     max tps  0.006484  4651.000000 398
## 15                     max tnr  0.621394     0.999977   0
## 16                     max fnr  0.621394     0.999570   0
## 17                     max fpr  0.005735     1.000000 399
## 18                     max tpr  0.006484     1.000000 398
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.ga
insLift(<model>, valid=<T/F>, xval=<T/F>)`
```

```
#Predict with h2o predict
h2o.predict(h2o.fit1, newdata = test.h2o)
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

```
##   predict        p0         p1
## 1       0 0.9569403 0.04305967
## 2       0 0.9335652 0.06643483
## 3       0 0.9806166 0.01938338
```

```
## 4       1 0.6000476 0.39995245
## 5       0 0.8198561 0.18014389
## 6       1 0.7909985 0.20900155
##
## [48100 rows x 3 columns]
```

```r
#Predict values with predict
predict(h2o.fit1, test.h2o)
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```



```
##   predict        p0         p1
## 1       0 0.9569403 0.04305967
## 2       0 0.9335652 0.06643483
## 3       0 0.9806166 0.01938338
## 4       1 0.6000476 0.39995245
## 5       0 0.8198561 0.18014389
## 6       1 0.7909985 0.20900155
##
## [48100 rows x 3 columns]
```