```r
# install.packages("scales")              # dependency of plot_glmnet
source("functions/plot_glmnet.R")
```

```r
theft_train = read_csv("../data/clean/theft_train.csv")
```

```
## Rows: 1836 Columns: 69

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): county, state
## dbl (67): fips, pertrump, permale, med_age, nevermarried, widowed, fromdifst...

##
## 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.
```

```r
theft_test = read_csv("../data/clean/theft_test.csv")
```

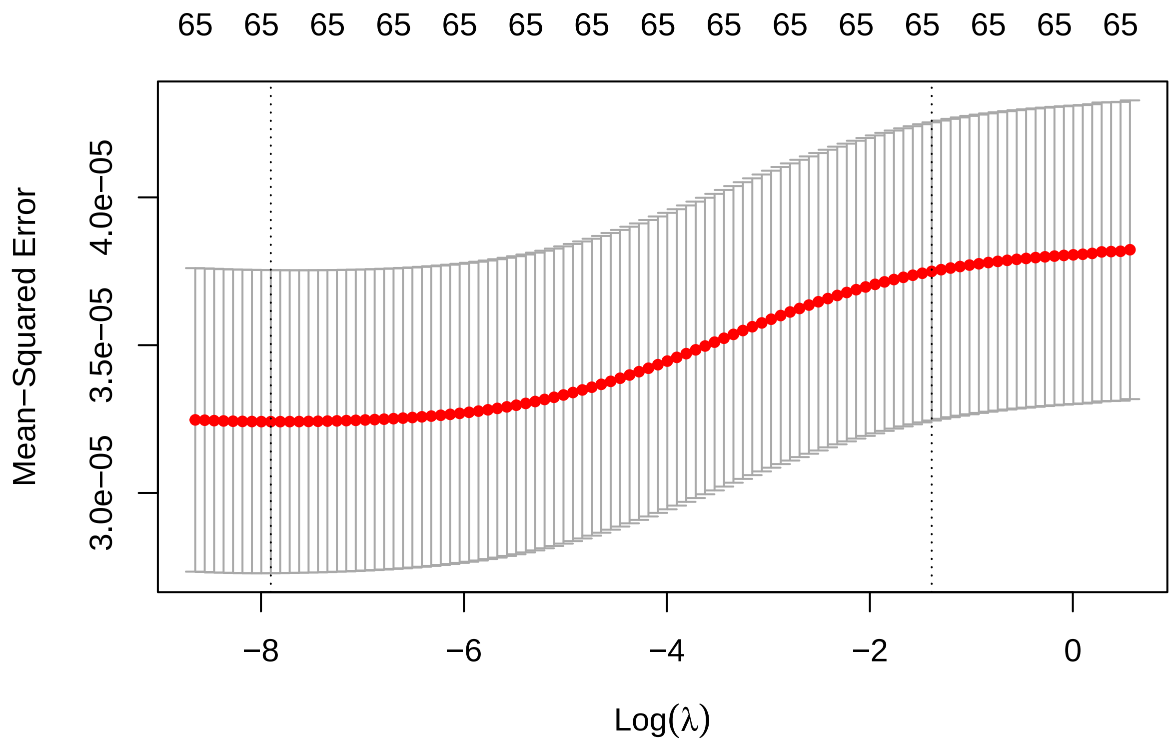
```
## Rows: 457 Columns: 69

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): county, state
## dbl (67): fips, pertrump, permale, med_age, nevermarried, widowed, fromdifst...

##
## 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.
```

Regression Based Methods

Ridge

```r
set.seed(471) # set seed for reproducibility
ridge_fit = cv.glmnet(theftrate ~ .-fips -state -county,  # formula notation, as usual
                      alpha = 0,                 # alpha = 0 for ridge
                      nfolds = 10,               # number of folds
                      data = theft_train)    # data to run ridge on
```

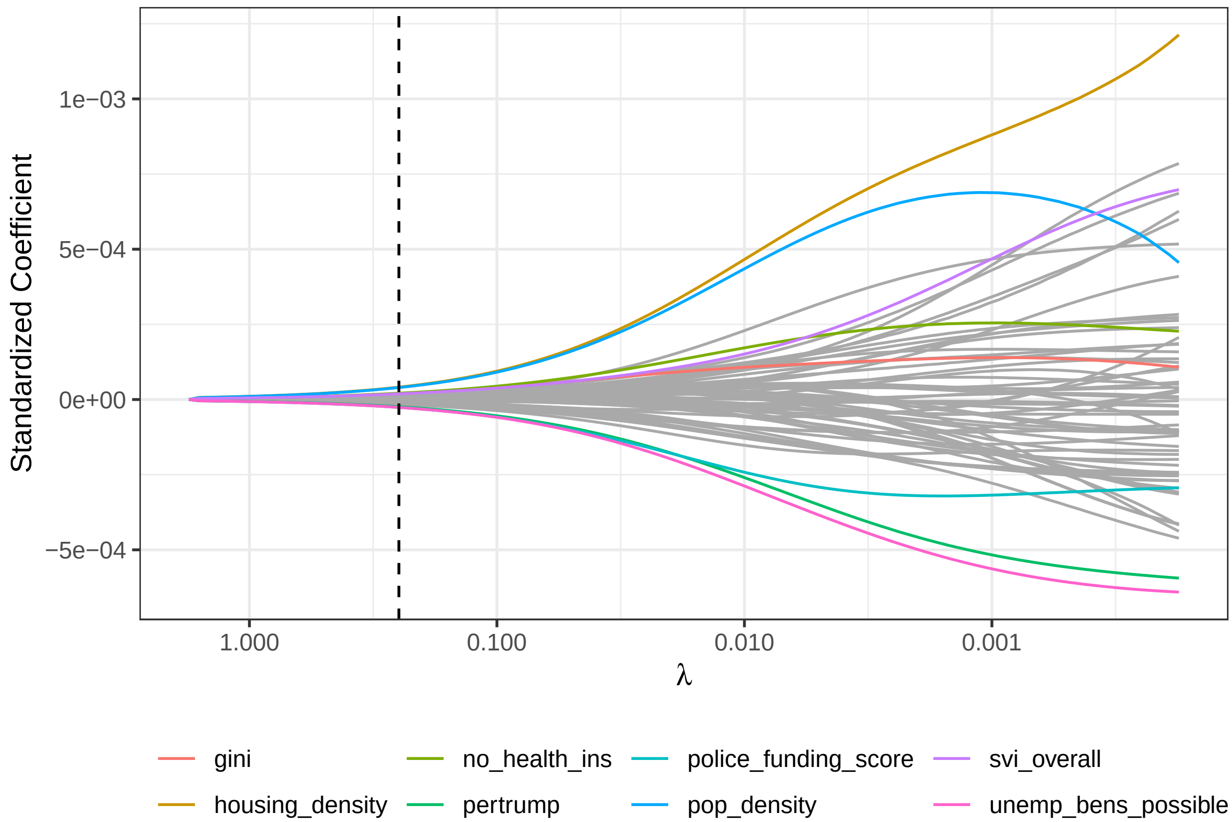
```r
plot(ridge_fit)
```




```r
plot_glmnet(ridge_fit, theft_train, features_to_plot = 8)
```

Table 1: Standardized coefficients for features in the Ridge model based on the one-standard-error rule.

Feature	Coefficient
housing_density	4.110e-05
pop_density	3.977e-05
unemp_bens_possible	-2.637e-05
police_funding_score	-2.621e-05
pertrump	-2.419e-05
no_health_ins	2.090e-05
gini	1.931e-05
svi_overall	1.887e-05
pct_child_in_pov	1.877e-05
Marriedcouplefamily	-1.689e-05




```
extract_std_coefs(ridge_fit, theft_train) %>%
  filter(coefficient != 0) %>% arrange(desc(abs(coefficient))) %>% head(10) %>%
  kable(format = "latex", row.names = NA,
        booktabs = TRUE, digits = 8,
        col.names = c("Feature", "Coefficient"),
        caption = "Standardized coefficients for features in the Ridge
        model based on the one-standard-error rule.") %>%
  kable_styling(position = "center")
```

lasso

```
set.seed(471) # set seed before cross-validation for reproducibility

lasso_fit = cv.glmnet(theftrate ~. -state -county -fips , alpha = 1, nfolds = 10, data = theft_train)
```

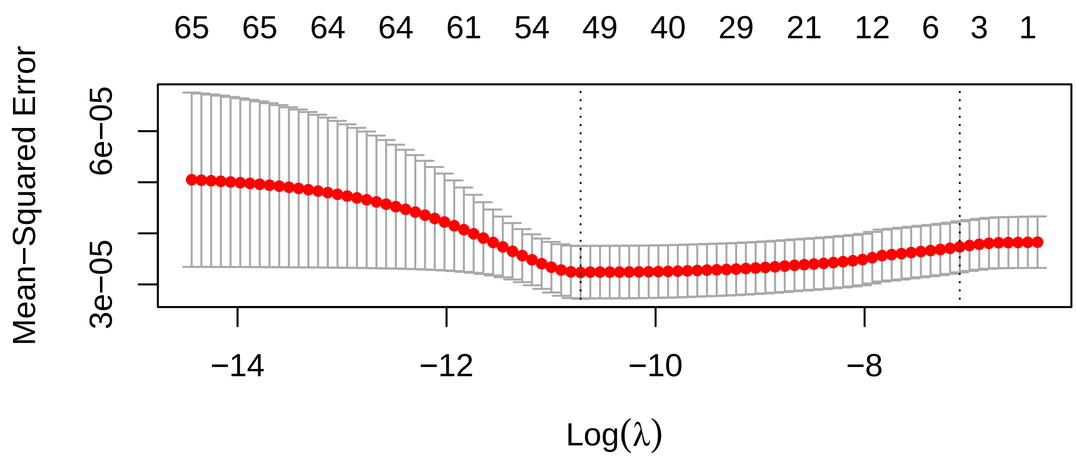


Figure 1: This is the CV plot for the 10-fold cross-validated lasso regression model on the training data.

```
lambda_lasso = lasso_fit$lambda.1se
sprintf("The value of lambda based on the one-standard-error rule: %f",
        lambda_lasso)
```

```
## [1] "The value of lambda based on the one-standard-error rule: 0.000834"
```

In Figure @ref(fig:lasso-cv-plot), we have the CV plot for a 10-fold cross-validated lasso regression model to the training data. (We can also note that corresponding to the right vertical dashed line on the plot (on the log scale), the value of lambda selected according to the one-standard-error rule is about 0.009.)

ii. How many features (excluding the intercept) are selected if lambda is chosen according to the one-standard-error rule?

```
num_features = lasso_fit$nzero[lasso_fit$lambda == lasso_fit$lambda.1se]
sprintf("The number of features (excluding intercept) selected (1se): %i",
        num_features)
```

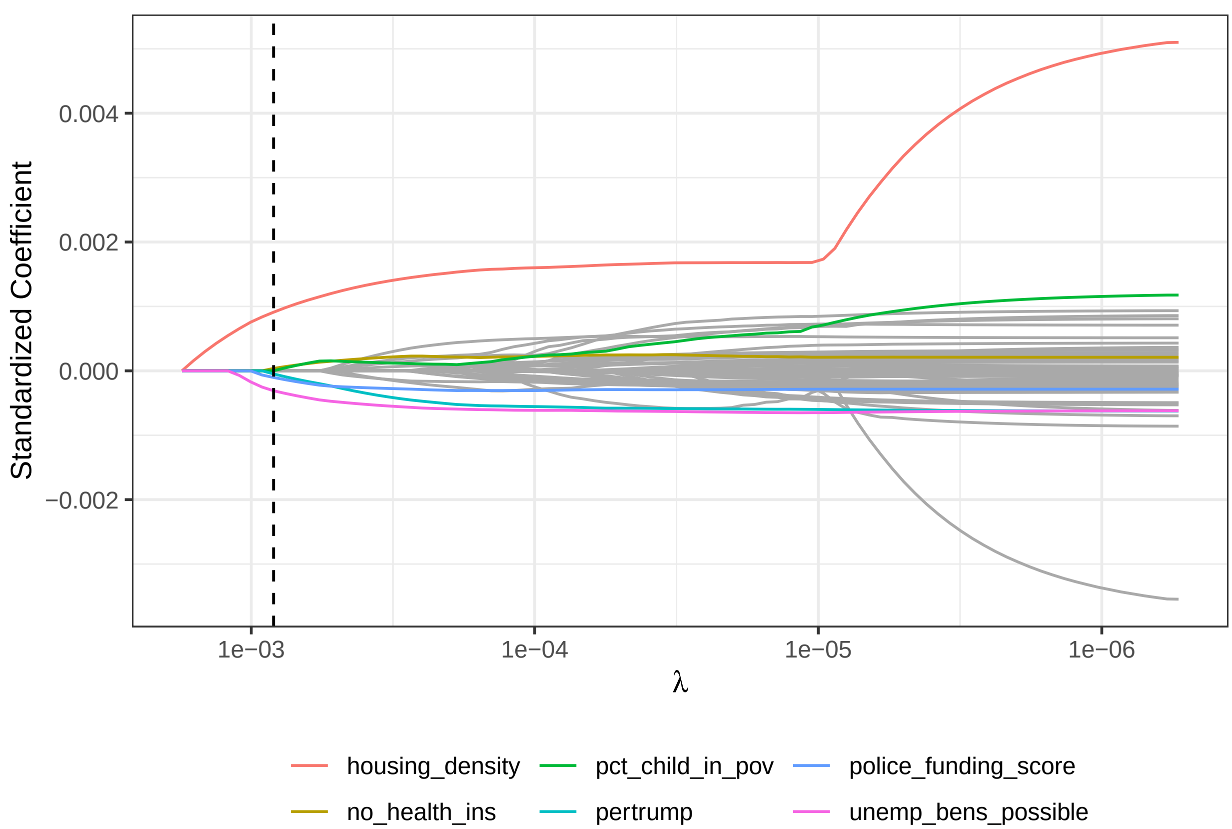
```
## [1] "The number of features (excluding intercept) selected (1se): 6"
```

Table 2: Standardized coefficients for features in the Lasso model based on the one-standard-error rule.

Feature	Coefficient
housing_density	0.00091026
no_health_ins	0.00003732
pct_child_in_pov	0.00000162
pertrump	-0.00004780
police_funding_score	-0.00010290
unemp_bens_possible	-0.00030893

```
extract_std_coefs(lasso_fit, theft_train) %>%
  filter(coefficient != 0) %>% arrange(desc(coefficient)) %>%
  kable(format = "latex", row.names = NA,
        booktabs = TRUE, digits = 8,
        col.names = c("Feature", "Coefficient"),
        caption = "Standardized coefficients for features in the Lasso
        model based on the one-standard-error rule.") %>%
  kable_styling(position = "center")
```

```
plot_glmnet(lasso_fit, theft_train)
```



Elastic net

Next, let's run an elastic net regression. We can do this via the `cva.glmnet()` function:

```
set.seed(471)
elnet_fit = cva.glmnet(theftrate ~ .-fips -county -state, # formula notation, as usual
                       nfolds = 10,               # number of folds
                       data = theft_train)    # data to run on
```

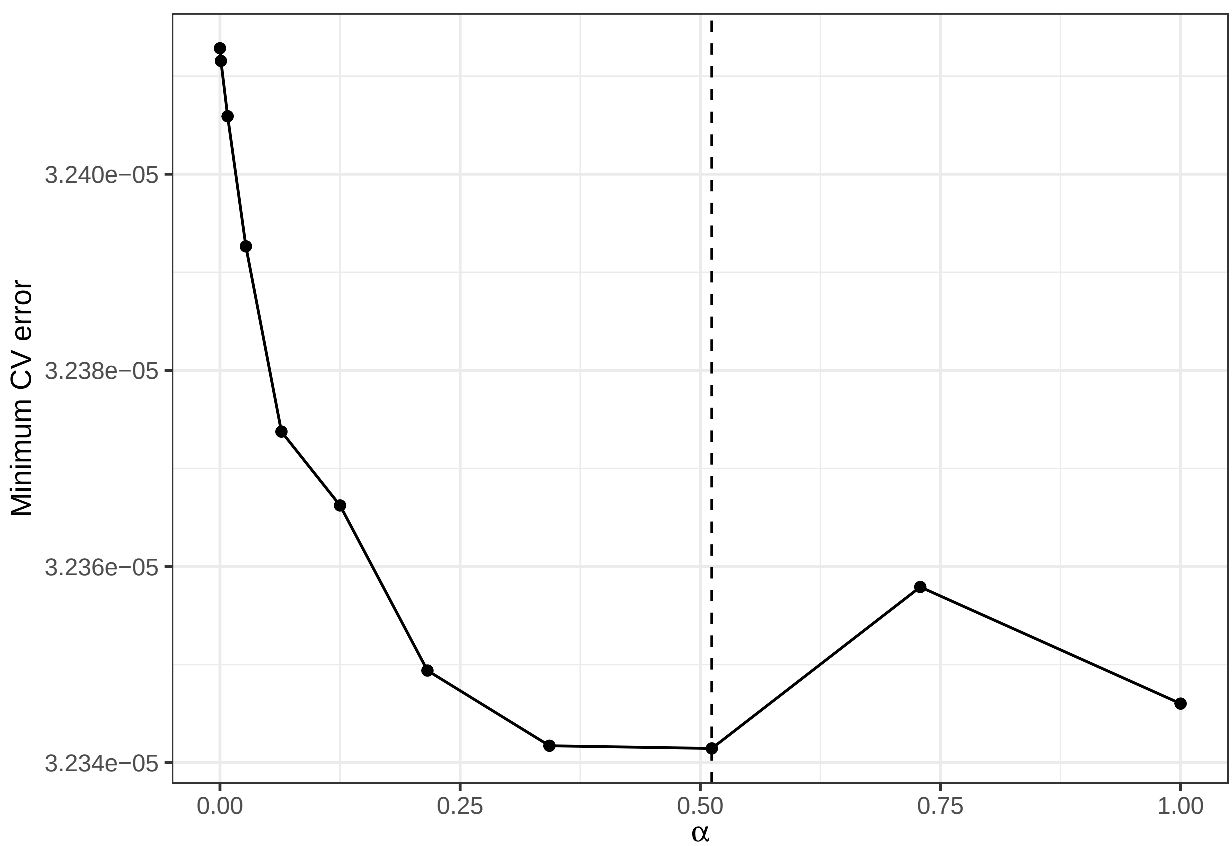
The following are the values of `alpha` that were used:

```
elnet_fit$alpha
```

```
##  [1] 0.000 0.001 0.008 0.027 0.064 0.125 0.216 0.343 0.512 0.729 1.000
```

We can plot the minimum CV error for each value of alpha using the helper function `plot_cva_glmnet()` from `plot_glmnet.R`:

```
plot_cva_glmnet(elnet_fit)
```



We can then extract the `cv.glmnet` fit object based on the optimal `alpha` using `extract_best_elnet` from `plot_glmnet.R`:

```r
elnet_fit_best = extract_best_elnet(elnet_fit)
```

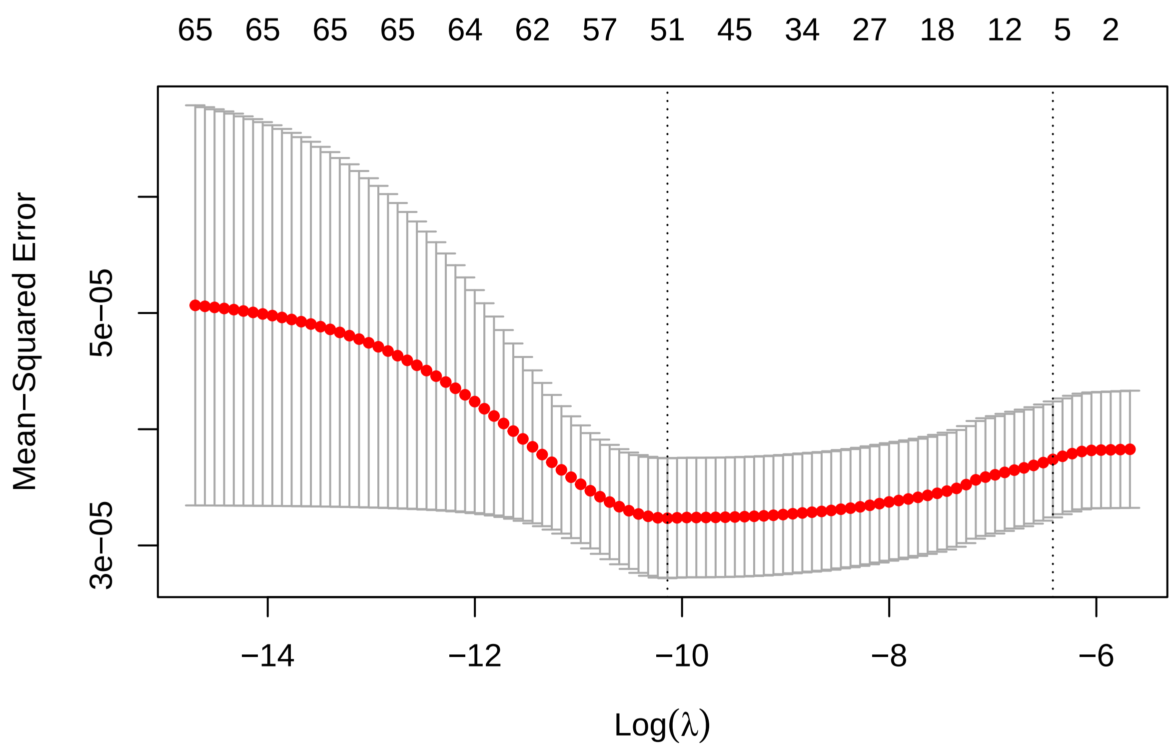
The `elnet_fit_best` object is a usual `glmnet` fit object, with an additional field called `alpha` specifying which value of `alpha` was used:

```r
elnet_fit_best$alpha
```

```
## [1] 0.512
```

We can make a CV plot to select `lambda` as usual:

```r
plot(elnet_fit_best)
```

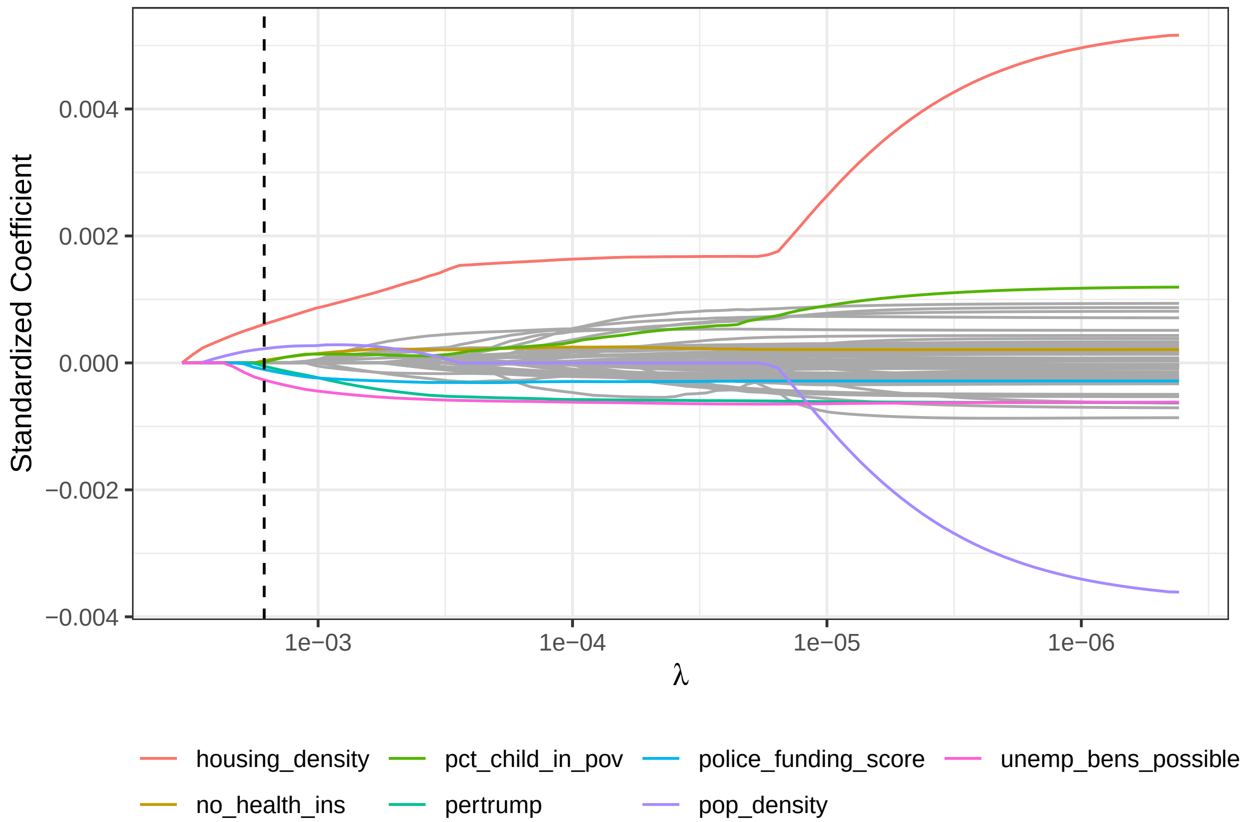


And we can make a trace plot for this optimal value of `alpha`:

```r
plot_glmnet(elnet_fit_best, theft_train)
```

Table 3: Standardized coefficients for features in the Lasso model based on the one-standard-error rule.

Feature	Coefficient
housing_density	0.00060887
pop_density	0.00022307
no_health_ins	0.00003622
pct_child_in_pov	0.00001214
pertrump	-0.00005131
police_funding_score	-0.00010466
unemp_bens_possible	-0.00027003

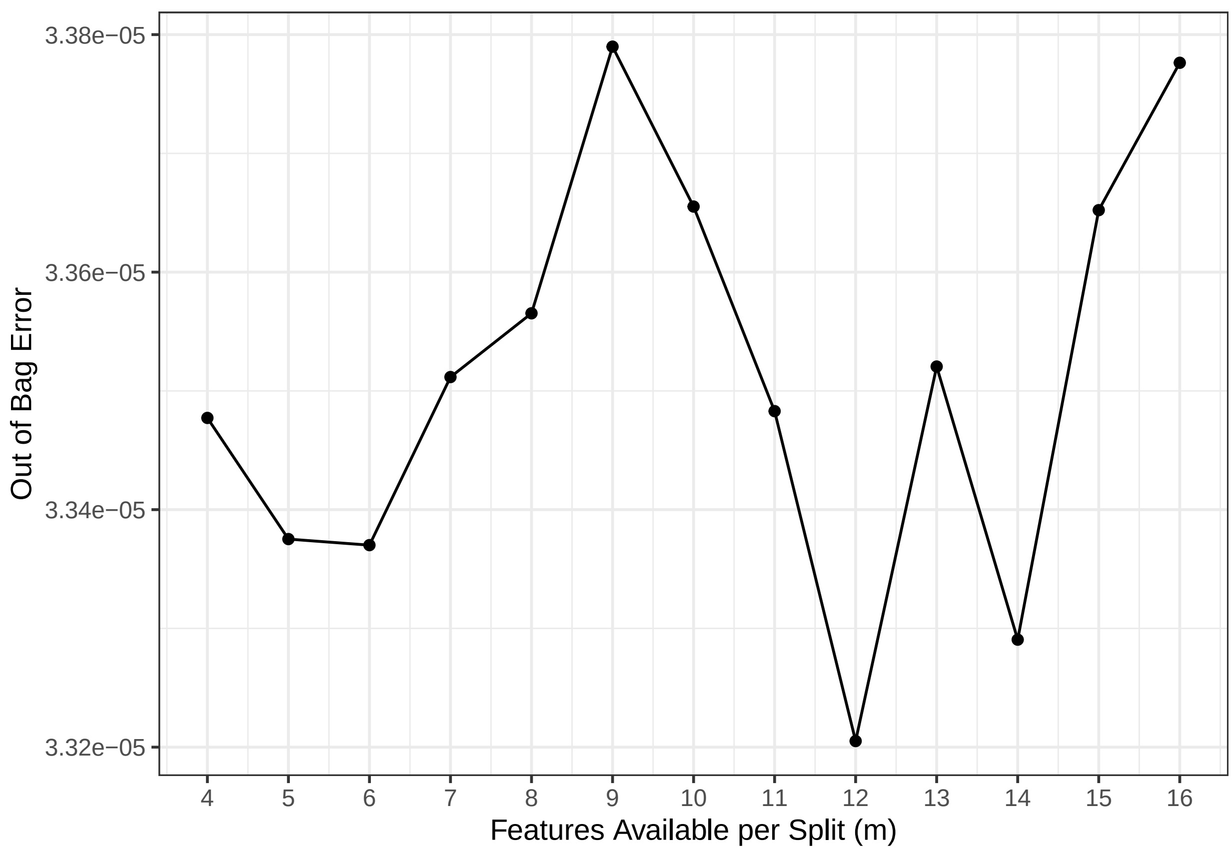



```r
extract_std_coefs(elnet_fit_best, theft_train) %>%
  filter(coefficient != 0) %>% arrange(desc(coefficient)) %>%
  kable(format = "latex", row.names = NA,
        booktabs = TRUE, digits = 8,
        col.names = c("Feature", "Coefficient"),
        caption = "Standardized coefficients for features in the Lasso
        model based on the one-standard-error rule.") %>%
  kable_styling(position = "center")
```

Tree based methods

random forest

```
set.seed(471) # set seed for reproducibility

mvalues = seq(4,16, by = 1)
oob_errors = numeric(length(mvalues))
ntree = 500
for(idx in 1:length(mvalues)){
  m = mvalues[idx]
  rf_fit = randomForest(theftrate ~. -fips -state - county, mtry = m, data = theft_train)
  oob_errors[idx] = rf_fit$mse[ntree]
}
tibble(m = mvalues, oob_err = oob_errors) %>%
  ggplot(aes(x = m, y = oob_err)) +
  geom_line() + geom_point() +
  scale_x_continuous(breaks = mvalues) + labs(y = "Out of Bag Error", x = "Features Available per Split
  theme_bw()
```



A quick-and-dirty way to tune a random forest is to try out a few different values of `mtry`:

```
rf_5 = randomForest(theftrate ~. -fips -state - county, mtry = 5, data = theft_train)
rf_6 = randomForest(theftrate ~. -fips -state - county, mtry = 6, data = theft_train)
```

```
rf_12 = randomForest(theftrate ~. -fips -state - county, mtry = 12, data = theft_train)
rf_14 = randomForest(theftrate ~. -fips -state - county, mtry = 14, data = theft_train)
```

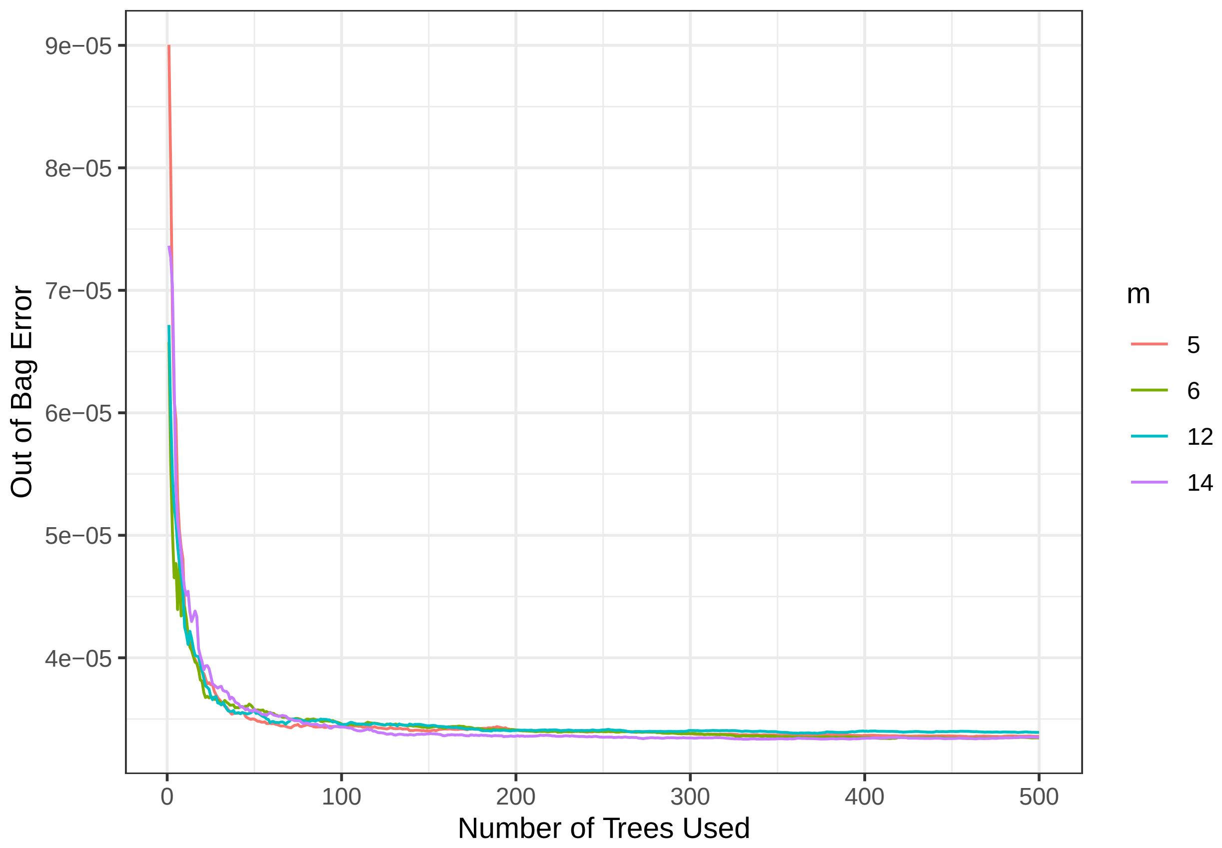
We can extract the OOB errors from each of these objects by using the `mse` field:

```
oob_errors2 = bind_rows(
  tibble(ntree = 1:500, oob_err = rf_5$mse, m = 5),
  tibble(ntree = 1:500, oob_err = rf_6$mse, m = 6),
  tibble(ntree = 1:500, oob_err = rf_12$mse, m = 12),
  tibble(ntree = 1:500, oob_err = rf_14$mse, m = 14),
)
oob_errors2
```

```
## # A tibble: 2,000 x 3
##    ntree   oob_err     m
##    <int>     <dbl> <dbl>
##  1     1 0.0000900     5
##  2     2 0.0000801     5
##  3     3 0.0000682     5
##  4     4 0.0000610     5
##  5     5 0.0000595     5
##  6     6 0.0000531     5
##  7     7 0.0000504     5
##  8     8 0.0000490     5
##  9     9 0.0000481     5
## 10    10 0.0000441     5
## # ... with 1,990 more rows
```

We can then plot these as follows:

```
oob_errors2 %>%
  ggplot(aes(x = ntree, y = oob_err, colour = factor(m))) +
  geom_line() + labs(y = "Out of Bag Error", x = "Number of Trees Used", color = "m") + theme_bw()
```




```
set.seed(471) # set seed for reproducibility
rf_12 = randomForest(theftrate ~ .-fips -state -county, mtry = 12, data = theft_train, importance = TRUI
```

```
rf_12$importance
```

```
##                              %IncMSE IncNodePurity
## pertrump                3.268471e-06  0.0021423476
## permale                 5.791992e-07  0.0010647745
## med_age                 4.448595e-07  0.0007421022
## nevermarried            3.311118e-07  0.0008855598
## widowed                 1.486440e-07  0.0006215402
## fromdifstate            2.251692e-07  0.0006548981
## fromabroad              5.055631e-08  0.0004622736
## divorced                1.220735e-06  0.0009135617
## foodstamp               4.599561e-07  0.0007782685
## Marriedcouplefamily     9.882575e-07  0.0006096238
## single_mom              2.859487e-07  0.0010440935
## inschool                7.440281e-07  0.0007791918
## inundergrad             1.607562e-07  0.0006266991
## ingradprofesh           4.481941e-07  0.0007847604
## lessthan_hs             1.285329e-06  0.0019144692
## bachplus                2.514062e-06  0.0024429584
## med_income              7.520443e-07  0.0006356728
## gini                    1.705314e-07  0.0006974698
## singledad               7.064211e-08  0.0004742167
```

```
## withkids                        8.160782e-07  0.0013242636
## med_2bed                        9.932610e-07  0.0007409655
## foreignborn                     7.705772e-07  0.0005623356
## unemployed_rate                 7.912557e-07  0.0005909172
## employed_rate                   1.167479e-06  0.0007045857
## no_health_ins                   1.617277e-06  0.0018726872
## dis5to17                        3.538471e-07  0.0006403859
## dis18to34                      -2.418496e-08  0.0006541598
## dis35to64                       9.246326e-07  0.0008900862
## pop_density                     1.667041e-06  0.0019541441
## housing_density                 1.625485e-06  0.0024304063
## pct_all_in_pov                  7.094815e-07  0.0011744358
## police_violence_score           1.915837e-07  0.0007978518
## police_accountability_score     4.150848e-07  0.0006533829
## approach_to_policing_score      3.089152e-07  0.0007761506
## police_funding_score            6.074883e-07  0.0017992451
## mean_hha_score                  2.057019e-07  0.0002055870
## svi_overall                     1.842966e-06  0.0008742754
## res_seg_nonwhite_white          4.743153e-07  0.0008105365
## pct_child_in_pov                1.605542e-06  0.0012193034
## sev_hou_cost_burden             3.016869e-07  0.0007918913
## sev_hou_prob                    8.478450e-07  0.0010582980
## poor_fair_health                4.343328e-07  0.0016873722
## spend_per_capita                2.621906e-06  0.0010974209
## unemp_bens_possible             2.569956e-06  0.0025678906
## saversperhouses                 5.221259e-07  0.0007441583
## ForeignBornEuropePct            5.306777e-07  0.0007536977
## ForeignBornMexPct               1.845339e-07  0.0005367608
## ForeignBornCaribPct             4.486352e-07  0.0006101083
## ForeignBornCentralSouthAmPct    2.041437e-07  0.0005759721
## ForeignBornAsiaPct              8.934119e-07  0.0007867626
## ForeignBornAfricaPct            2.130971e-07  0.0005704192
## AvgHHSize                       8.468149e-07  0.0007666426
## PopChangeRate1819               5.920033e-07  0.0010119685
## NonEnglishHHNum                 8.031597e-07  0.0006855051
## PctEmpChange1920                5.079400e-07  0.0007690777
## PctEmpConstruction              4.281844e-07  0.0026076394
## PctEmpMining                    3.037590e-07  0.0007866685
## PctEmpTrade                     7.251240e-08  0.0006374936
## PctEmpTrans                    -1.192790e-08  0.0008641203
## PctEmpInformation               4.914684e-07  0.0016163770
## PctEmpFIRE                      3.280471e-07  0.0014551722
## Deep_Pov_Children               5.235189e-07  0.0005797682
## PerCapitaInc                    1.156042e-06  0.0013849461
## Deep_Pov_All                    4.015176e-07  0.0005207421
## incar_rate                      1.186708e-07  0.0005256170
```

```
varImpPlot(rf_12, n.var = 10)
```

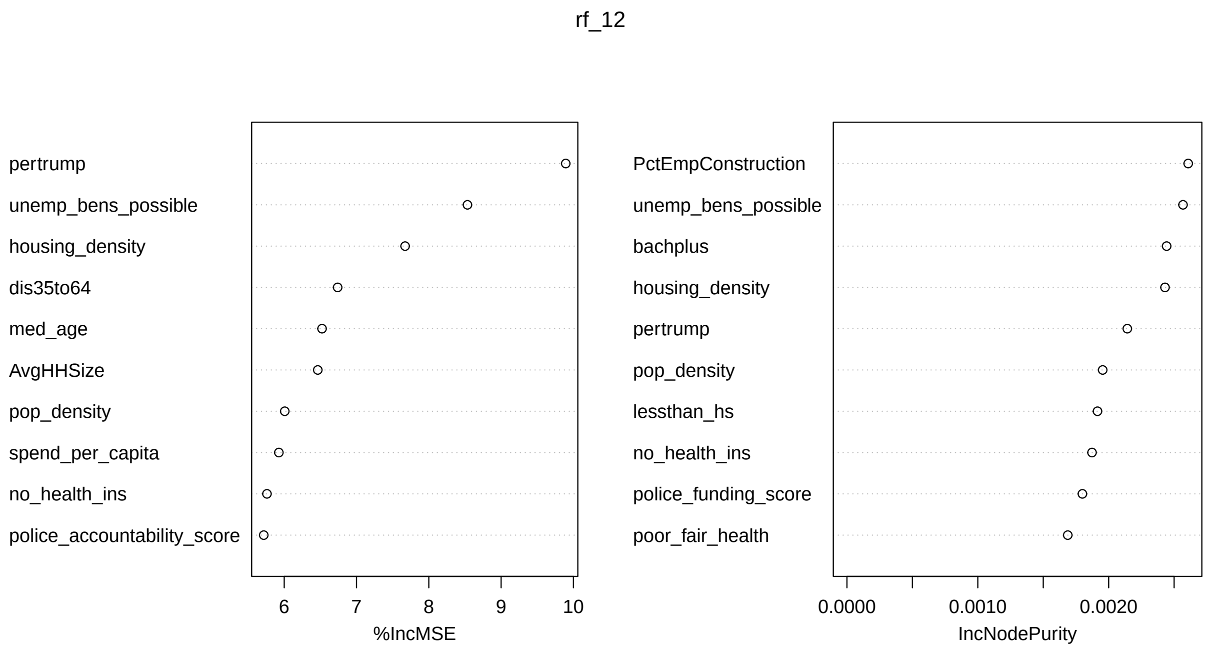


Figure 2: Variable Importance Plot for the optimal random forest model.

Boosting

Model tuning (4 points)

i. (2 points) Fit boosted tree models with interaction depths 1, 2, and 3. For each, use a shrinkage factor of 0.1, 1000 trees, and 5-fold cross-validation.

Solution:

```
set.seed(471) # for reproducibility (DO NOT CHANGE)
# TODO: Fit random forest with interaction depth 1

gbm_1 = gbm(theftrate ~ . -fips -state -county,
              distribution = "gaussian",
              n.trees = 1000,
              interaction.depth = 1,
              shrinkage = 0.1,
              cv.folds = 5,
              data = theft_train)

set.seed(471) # for reproducibility (DO NOT CHANGE)
# TODO: Fit random forest with interaction depth 2

gbm_2 = gbm(theftrate ~ . -fips -state -county,
              distribution = "gaussian",
              n.trees = 1000,
              interaction.depth = 2,
```

```
             shrinkage = 0.1,
             cv.folds = 5,
             data = theft_train)

set.seed(471) # for reproducibility (DO NOT CHANGE)
# TODO: Fit random forest with interaction depth 3


gbm_3 = gbm(theftrate ~ . -fips -state -county,
             distribution = "gaussian",
             n.trees = 1000,
             interaction.depth = 3,
             shrinkage = 0.1,
             cv.folds = 5,
             data = theft_train)
```

```
ntrees = 1000
cv_errors = bind_rows(
  tibble(ntree = 1:ntrees, cv_err = gbm_1$cv.error, Depth = 1),
  tibble(ntree = 1:ntrees, cv_err = gbm_2$cv.error, Depth = 2),
  tibble(ntree = 1:ntrees, cv_err = gbm_3$cv.error, Depth = 3)
) %>% mutate(Depth = factor(Depth))

# plot CV errors

mins = cv_errors %>% group_by(Depth) %>% summarise(min_err = min(cv_err))

gbm.perf(gbm_3, plot.it = FALSE)
```

```
## [1] 43
```

```
cv_errors %>%
  ggplot(aes(x = ntree, y = cv_err, colour = Depth)) +
  geom_line() + theme_bw() +
  geom_hline(aes(yintercept = min_err, color = Depth),
             data = mins, linetype = "dashed") +
  labs(y = "CV Error", x = "Trees") + scale_y_log10()
```

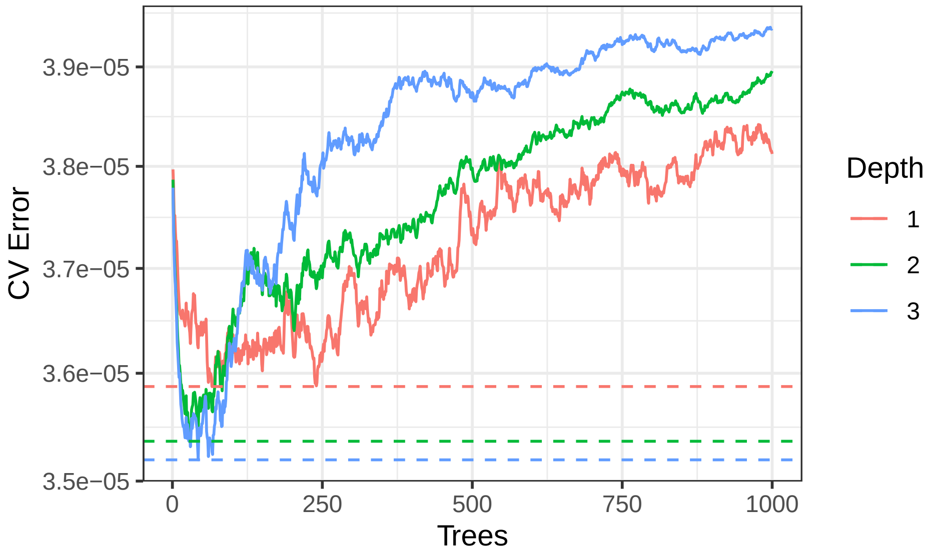


Figure 3: CV Error by Trees and Interaction Depth (with min error for each depth dashed)

Solution:

```
gbm_fit_optimal = gbm_3
optimal_num_trees = gbm.perf(gbm_3, plot.it = FALSE)
summary(gbm_3, n.trees = optimal_num_trees, plotit = FALSE) %>% tibble() %>%
  head(12) %>%
  kable(format = "latex", row.names = NA,
        booktabs = TRUE,
        digits = 3,
        col.names = c("Variable", "Relative influence"),
        caption = "These are the first ten rows of the relative influence
        table for the optimal boosting model above.") %>%
  kable_styling(position = "center") %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 4: These are the first ten rows of the relative influence table for the optimal boosting model above.

Variable	Relative influence
bachplus	22.800
PctEmpFIRE	9.707
unemp_bens_possible	9.269
housing_density	4.708
pertrump	4.603
poor_fair_health	3.635
pop_density	3.632
police_funding_score	3.370
withkids	3.279
dis35to64	3.241
lessthan_hs	3.088
no_health_ins	2.566

ii. (4 points) Produce partial dependence plots for the top three features based on relative influence. Comment on the nature of the relationship with the response and whether it makes sense.

Solution:

```
housing = plot(gbm_3, i.var = "housing_density",
     n.trees = optimal_num_trees)
```

```
FIRE = plot(gbm_3, i.var = "PctEmpFIRE",
     n.trees = optimal_num_trees)
```

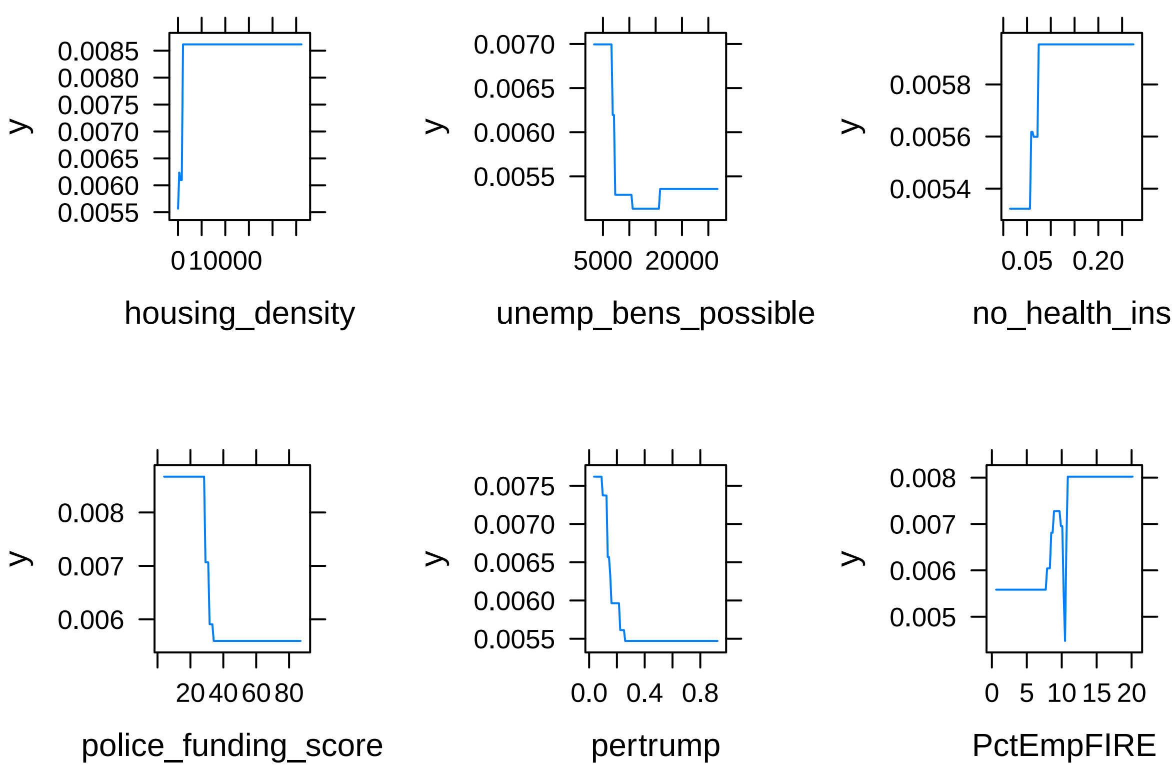
```
trump = plot(gbm_3, i.var = "pertrump", n.trees =
       optimal_num_trees)
```

```
bens = plot(gbm_3, i.var = "unemp_bens_possible", n.trees =
       optimal_num_trees)
```

```
police = plot(gbm_3, i.var = "police_funding_score", n.trees =
       optimal_num_trees)
```

```
healthins = plot(gbm_3, i.var = "no_health_ins", n.trees =
       optimal_num_trees)
```

```
plot_grid(nrow = 2, housing, bens, healthins, police, trump, FIRE)
```



```r
set.seed(471)

# ridge prediction error
ridge_predictions = predict(ridge_fit,
                            newdata = theft_test,
                            s = "lambda.1se") %>% as.numeric()

ridge_RMSE = sqrt(mean((ridge_predictions-theft_test$theftrate)^2))



# lasso prediction error

lasso_predictions = predict(lasso_fit,
                            newdata = theft_test,
                            s = "lambda.1se") %>%
  as.numeric()

lasso_RMSE = sqrt(mean((lasso_predictions-theft_test$theftrate)^2))




# elnet prediction error

elnet_predictions = predict(elnet_fit,
                            alpha = elnet_fit_best$alpha,
                            newdata = theft_test,
                            s = "lambda.1se") %>%
  as.numeric()



elnet_RMSE = sqrt(mean((elnet_predictions-theft_test$theftrate)^2))



# intercept-only prediction error
training_mean_response = mean(theft_test$theftrate)
constant_RMSE = sqrt(mean((training_mean_response-theft_test$theftrate)^2))



#RF
rf_predictions = predict(rf_12, newdata = theft_test)

rf_RMSE = sqrt(mean((rf_predictions-theft_test$theftrate)^2))

#Boosting
gbm_predictions = predict(gbm_3, n.trees = optimal_num_trees,
                          newdata = theft_test)

gbm_RMSE = sqrt(mean((gbm_predictions-theft_test$theftrate)^2))
```

Table 5: Root-mean-squared prediction errors by model type, and intercept-only models.

Model	Test RMSE
Ridge	0.00641267
Lasso	0.00637219
Intercept-only	0.00645320
Elastic_Net	0.00637443
Random_Forest	0.00630238
Boosting	0.00630166

```
# print nice table


tibble(Ridge = ridge_RMSE, Lasso = lasso_RMSE, `Intercept-only` = constant_RMSE,
       Elastic_Net = elnet_RMSE, Random_Forest = rf_RMSE, Boosting = gbm_RMSE) %>% pivot_longer(everythi
   kable(format = "latex", row.names = NA,
       booktabs = TRUE, digits = 8,
       caption = "Root-mean-squared prediction errors by model type,
       and intercept-only models.") %>%
 kable_styling(position = "center")
```

```
mean(theft_test$theftrate)
```

```
## [1] 0.005804112
```