Model Selection - readmission_1year

Eduardo Yuki Yada

Global parameters

```r
k <- params$k # Number of folds for cross validation
grid_size <- params$grid_size # Number of parameter combination to tune on each model
repeats <- params$repeats
RUN_ALL_MODELS <- params$RUN_ALL_MODELS
Hmisc::list.tree(params)
```

```
##  params = list 5 (968 bytes)
## .  outcome_column = character 1= readmission_1year
## .  k = double 1= 10
## .  grid_size = double 1= 20
## .  repeats = double 1= 2
## .  RUN_ALL_MODELS = logical 1= TRUE
```

Minutes to run: 0

Imports

```r
library(tidyverse)
library(yaml)
library(tidymodels)
library(usemodels)
library(vip)
library(bonsai)
library(lightgbm)
library(caret)
library(pROC)

source("aux_functions.R")
predict <- stats::predict
```

Minutes to run: 0

Loading data

```r
load('dataset/processed_data.RData')
load('dataset/processed_dictionary.RData')

columns_list <- yaml.load_file("./auxiliar/columns_list.yaml")

outcome_column <- params$outcome_column
features_list <- params$features_list

df <- mutate(df, across(where(is.character), as.factor))
```

Minutes to run: 0.006

```r
dir.create(file.path("./auxiliar/model_selection/hyperparameters/"),
           showWarnings = FALSE,
           recursive = TRUE)

dir.create(file.path("./auxiliar/model_selection/performance/"),
           showWarnings = FALSE,
           recursive = TRUE)
```

Minutes to run: 0

Eligible features

```
cat_features_list = read_yaml(sprintf(
  "./auxiliar/significant_columns/categorical_%s.yaml",
  outcome_column
))

num_features_list = read_yaml(sprintf(
  "./auxiliar/significant_columns/numerical_%s.yaml",
  outcome_column
))

features_list = c(cat_features_list, num_features_list)
```

Minutes to run: 0

```
eligible_columns = df_names %>%
  filter(momento.aquisicao == 'Admissão t0') %>%
  .$variable.name

exception_columns = c('death_intraop', 'death_intraop_1', 'disch_outcomes_t0')

correlated_columns = c('year_procedure_1', # com year_adm_t0
                       'age_surgery_1', # com age
                       'admission_t0', # com admission_pre_t0_count
                       'atb', # com meds_antimicrobianos
                       'classe_meds_cardio_qtde', # com classe_meds_qtde
                       'suporte_hemod', # com proced_invasivos_qtde,
                       'radiografia', # com exames_imagem_qtde
                       'ecg' # com metodos_graficos_qtde
                       )

eligible_features = eligible_columns %>%
  base::intersect(c(columns_list$categorical_columns, columns_list$numerical_columns)) %>%
  setdiff(c(exception_columns, correlated_columns))

features = base::intersect(eligible_features, features_list)

gluedown::md_order(features, seq = TRUE, pad = TRUE)
```

```
## 01. sex
## 02. age
## 03. race
## 04. education_level
## 05. patient_state
## 06. underlying_heart_disease
## 07. heart_disease
## 08. nyha_basal
## 09. prior_mi
## 10. heart_failure
## 11. af
## 12. cardiac_arrest
## 13. transplant
## 14. valvopathy
## 15. endocardites
## 16. diabetes
## 17. renal_failure
## 18. hemodialysis
## 19. copd
## 20. comorbidities_count
## 21. procedure_type_1
## 22. reop_type_1
## 23. procedure_type_new
## 24. cied_final_1
## 25. cied_final_group_1
## 26. admission_pre_t0_count
## 27. admission_pre_t0_180d
## 28. year_adm_t0
## 29. icu_t0
## 30. dialysis_t0
## 31. admission_t0_emergency
## 32. aco
## 33. antiarritmico
## 34. betabloqueador
## 35. ieca_bra
## 36. dva
## 37. digoxina
## 38. estatina
## 39. diuretico
## 40. vasodilatador
## 41. insuf_cardiaca
## 42. espironolactona
## 43. bloq_calcio
## 44. antiplaquetario_ev
## 45. insulina
## 46. anticonvulsivante
## 47. psicofarmacos
## 48. antifungico
## 49. antiviral
## 50. antiretroviral
## 51. classe_meds_qtde
## 52. meds_cardiovasc_qtde
## 53. meds_antimicrobianos
## 54. ventilacao_mecanica
## 55. cec
## 56. transplante_cardiaco
## 57. cir_toracica
## 58. outros_proced_cirurgicos
## 59. icp
## 60. intervencao_cv
## 61. angioplastia
## 62. cateterismo
## 63. eletrofisiologia
## 64. cateter_venoso_central
## 65. proced_invasivos_qtde
## 66. cve_desf
## 67. transfusao
## 68. interconsulta
## 69. equipe_multiprof
## 70. holter
## 71. teste_esforco
## 72. espiro_ergoespiro
## 73. tilt_teste
## 74. metodos_graficos_qtde
## 75. laboratorio
## 76. cultura
## 77. analises_clinicas_qtde
## 78. citologia
## 79. biopsia
## 80. histopatologia_qtde
## 81. angio_rm
## 82. angio_tc
## 83. aortografia
## 84. arteriografia
## 85. cintilografia
## 86. ecocardiograma
## 87. endoscopia
## 88. flebografia
## 89. pet_ct
## 90. ultrassom
## 91. tomografia
## 92. ressonancia
## 93. exames_imagem_qtde
## 94. dieta_parenteral
## 95. bic
## 96. mpp
## 97. hospital_stay
```

Minutes to run: 0

Train test split (70%/30%)

```
set.seed(42)

if (outcome_column == 'readmission_30d') {
  df_split <- readRDS("./dataset/split_object.rds")
} else {
  df_split <- initial_split(df, prop = .7, strata = all_of(outcome_column))
}

df_train <- training(df_split) %>% dplyr::select(all_of(c(features, outcome_column)))
df_test <- testing(df_split) %>% dplyr::select(all_of(c(features, outcome_column)))

df_folds <- vfold_cv(df_train, v = k,
                     strata = all_of(outcome_column))
```

Minutes to run: 0.001

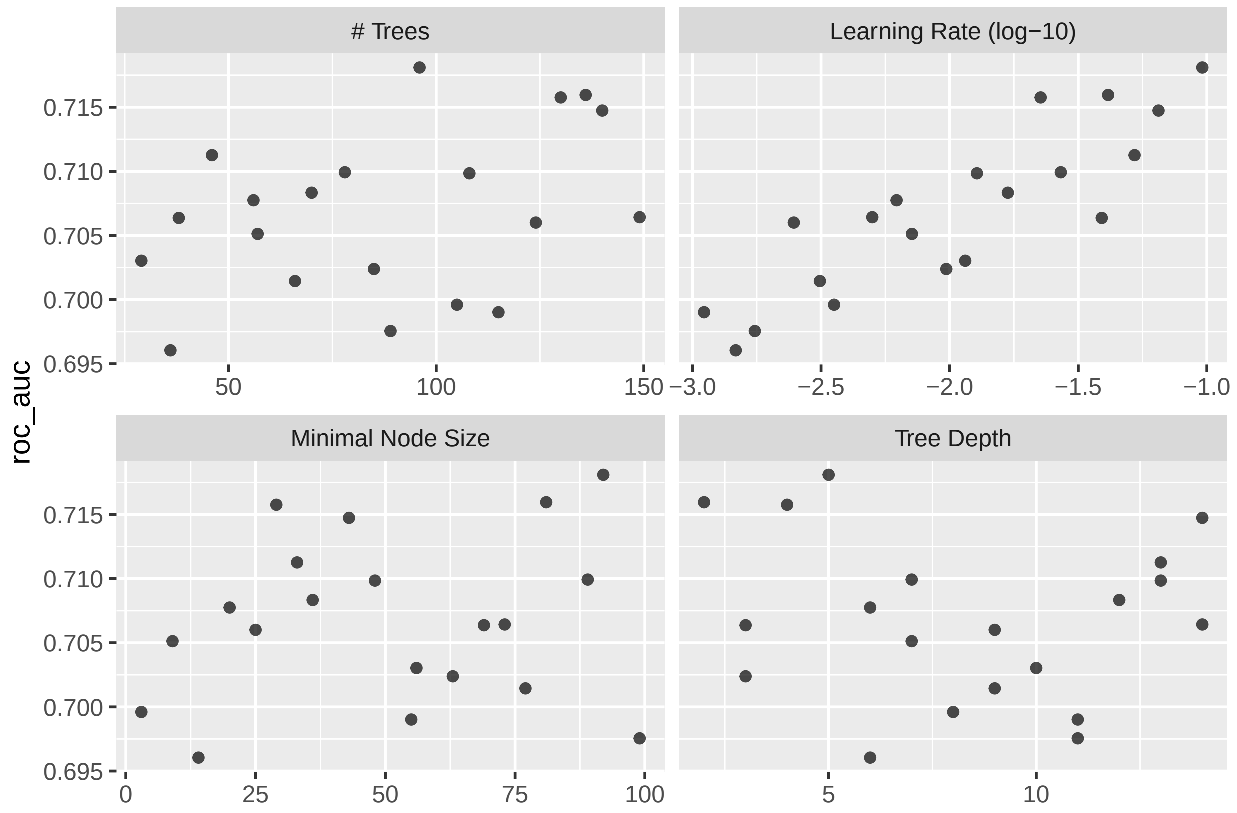
Boosted Tree (XGBoost)

```
xgboost_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
  step_dummy(all_nominal_predictors())

xgboost_spec <- boost_tree(
  trees = tune(),
  min_n = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
) %>%
  set_engine("xgboost",
```

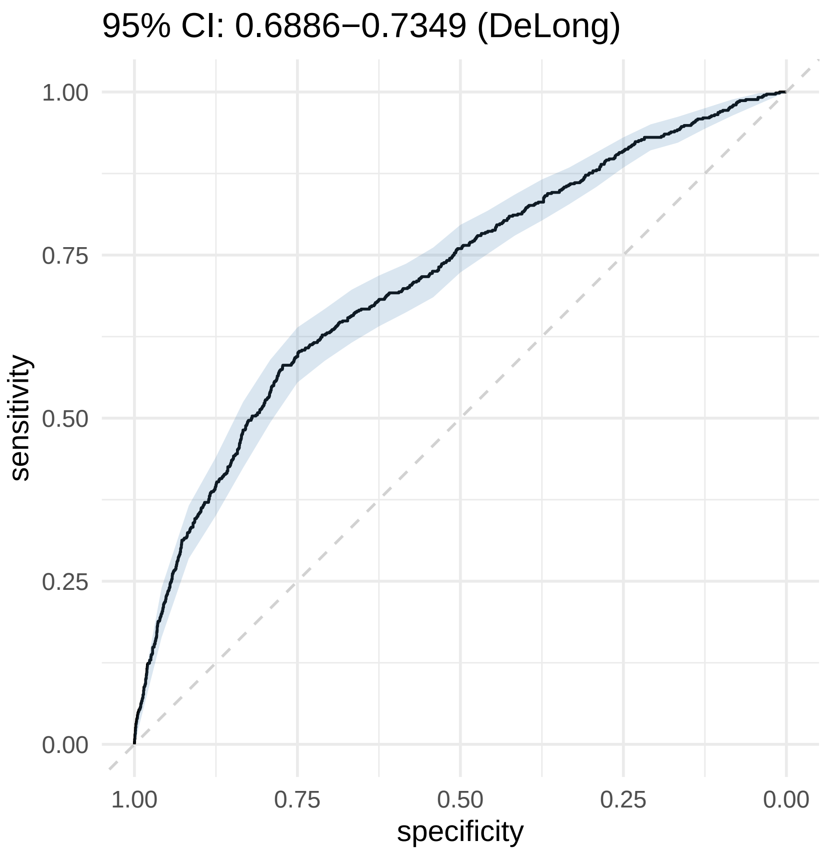
```r
             nthread = 8) %>%
  set_mode("classification")

xgboost_grid <- grid_latin_hypercube(
  trees(range = c(25L, 150L)),
  min_n(range = c(2L, 100L)),
  tree_depth(range = c(2L, 15L)),
  learn_rate(range = c(-3, -1), trans = log10_trans()),
  size = grid_size
)

xgboost_workflow <-
  workflow() %>%
  add_recipe(xgboost_recipe) %>%
  add_model(xgboost_spec)

xgboost_tune <-
  xgboost_workflow %>%
  tune_grid(resamples = df_folds,
            grid = xgboost_grid)

xgboost_tune %>%
  show_best("roc_auc")
```

```
## # A tibble: 5 x 10
##   trees min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    96    92          5     0.0960 roc_auc binary     0.718    10 0.00757 Prepro~
## 2   136    81          2     0.0413 roc_auc binary     0.716    10 0.00727 Prepro~
## 3   130    29          4     0.0226 roc_auc binary     0.716    10 0.00681 Prepro~
## 4   140    43         14     0.0649 roc_auc binary     0.715    10 0.00767 Prepro~
## 5    46    33         13     0.0524 roc_auc binary     0.711    10 0.00731 Prepro~
```

```r
best_xgboost <- xgboost_tune %>%
  select_best("roc_auc")

autoplot(xgboost_tune, metric = "roc_auc")
```

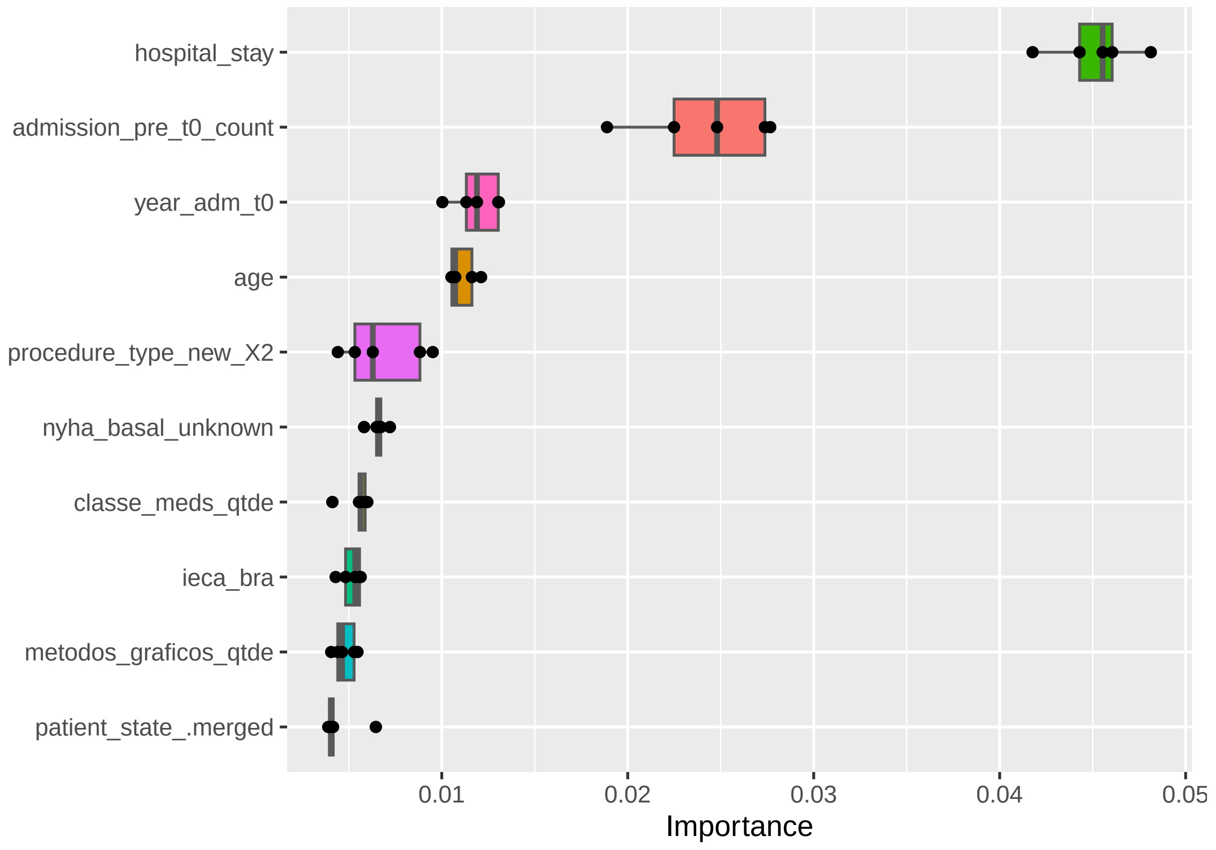


```r
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

final_xgboost_fit <- extract_workflow(last_xgboost_fit)

xgboost_auc <- validation(final_xgboost_fit, df_test)
```



```
## [1] "Optimal Threshold: 0.14"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 3188  253
##    1  939  351
##
##                Accuracy : 0.748
##                  95% CI : (0.7354, 0.7604)
##     No Information Rate : 0.8723
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2382
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7725
##             Specificity : 0.5811
##          Pos Pred Value : 0.9265
##          Neg Pred Value : 0.2721
##              Prevalence : 0.8723
##          Detection Rate : 0.6739
##    Detection Prevalence : 0.7273
##       Balanced Accuracy : 0.6768
##
##        'Positive' Class : 0
##
```

```
extract_vip(final_xgboost_fit, pred_wrapper = predict,
            reference_class = "0")
```



```r
xgboost_parameters <- xgboost_tune %>%
  show_best("roc_auc", n = 1) %>%
  select(-.metric, -.estimator, -.config, -mean, -n, -std_err) %>%
  as.list
```

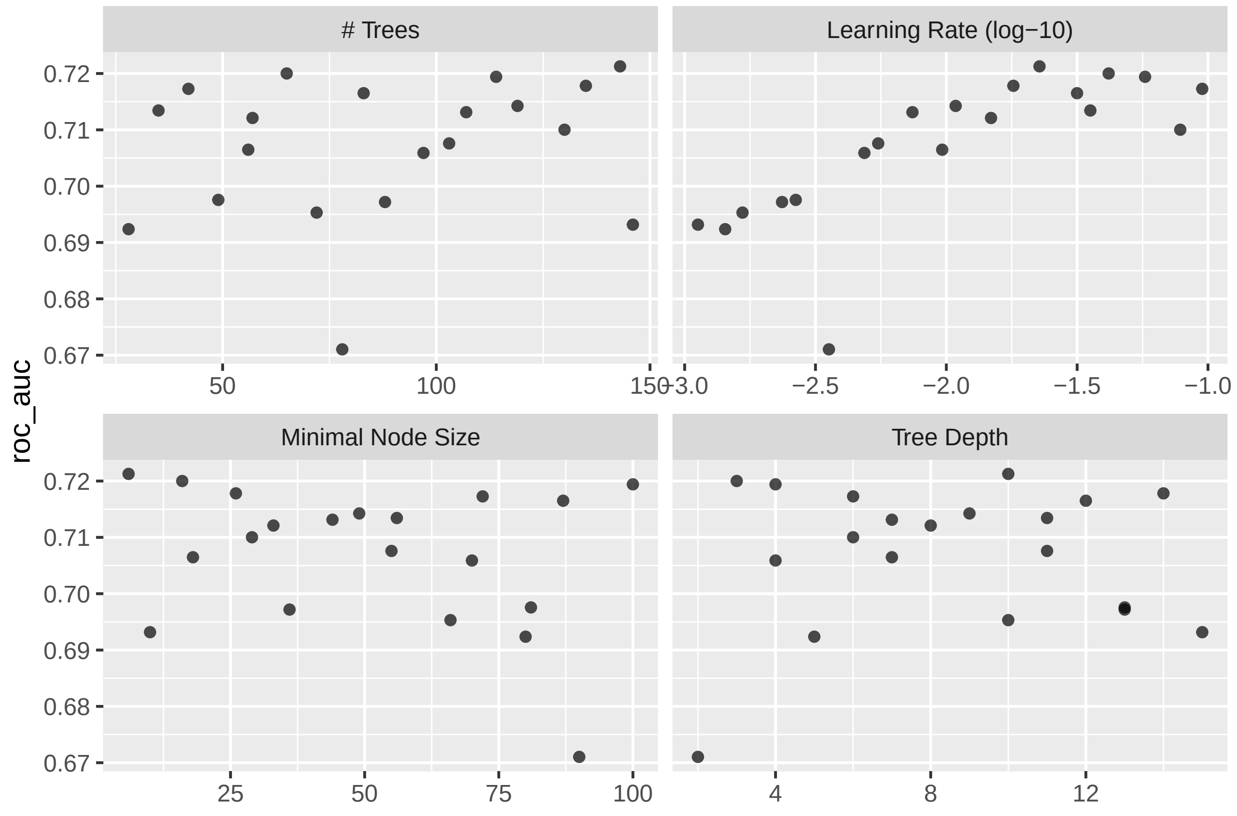
Minutes to run: 5.452

Boosted Tree (LightGBM)

```r
lightgbm_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
  step_dummy(all_nominal_predictors())

lightgbm_spec <- boost_tree(
  trees = tune(),
  min_n = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  sample_size = 1
) %>%
  set_engine("lightgbm",
             nthread = 8) %>%
  set_mode("classification")

lightgbm_grid <- grid_latin_hypercube(
  trees(range = c(25L, 150L)),
  min_n(range = c(2L, 100L)),
  tree_depth(range = c(2L, 15L)),
  learn_rate(range = c(-3, -1), trans = log10_trans()),
  size = grid_size
)

lightgbm_workflow <-
  workflow() %>%
  add_recipe(lightgbm_recipe) %>%
  add_model(lightgbm_spec)

lightgbm_tune <-
  lightgbm_workflow %>%
  tune_grid(resamples = df_folds,
            grid = lightgbm_grid)

lightgbm_tune %>%
  show_best("roc_auc")
```

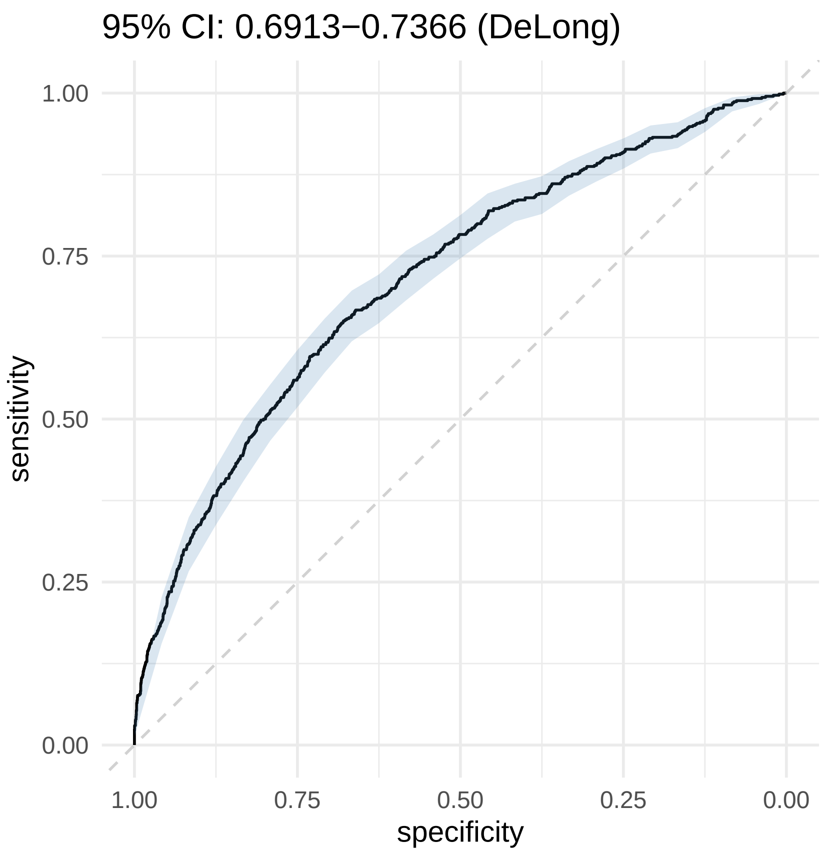
```
## # A tibble: 5 x 10
##   trees min_n tree_depth learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   143     6         10     0.0227 roc_auc binary     0.721    10 0.00741 Prepro~
## 2    65    16          3     0.0417 roc_auc binary     0.720    10 0.00666 Prepro~
## 3   114   100          4     0.0575 roc_auc binary     0.719    10 0.00703 Prepro~
## 4   135    26         14     0.0180 roc_auc binary     0.718    10 0.00751 Prepro~
## 5    42    72          6     0.0951 roc_auc binary     0.717    10 0.00744 Prepro~
```

```
best_lightgbm <- lightgbm_tune %>%
  select_best("roc_auc")

autoplot(lightgbm_tune, metric = "roc_auc")
```



```
final_lightgbm_workflow <-
  lightgbm_workflow %>%
  finalize_workflow(best_lightgbm)

last_lightgbm_fit <-
```

```
  final_lightgbm_workflow %>%
  last_fit(df_split)

final_lightgbm_fit <- extract_workflow(last_lightgbm_fit)

lightgbm_auc <- validation(final_lightgbm_fit, df_test)
```



```
## [1] "Optimal Threshold: 0.11"
## Confusion Matrix and Statistics
##
##       reference
## data    0    1
##    0 2804  211
##    1 1323  393
##
##                Accuracy : 0.6758
##                  95% CI : (0.6622, 0.6891)
##     No Information Rate : 0.8723
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1848
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6794
##             Specificity : 0.6507
##          Pos Pred Value : 0.9300
##          Neg Pred Value : 0.2290
##              Prevalence : 0.8723
##          Detection Rate : 0.5927
##    Detection Prevalence : 0.6373
##       Balanced Accuracy : 0.6650
##
```

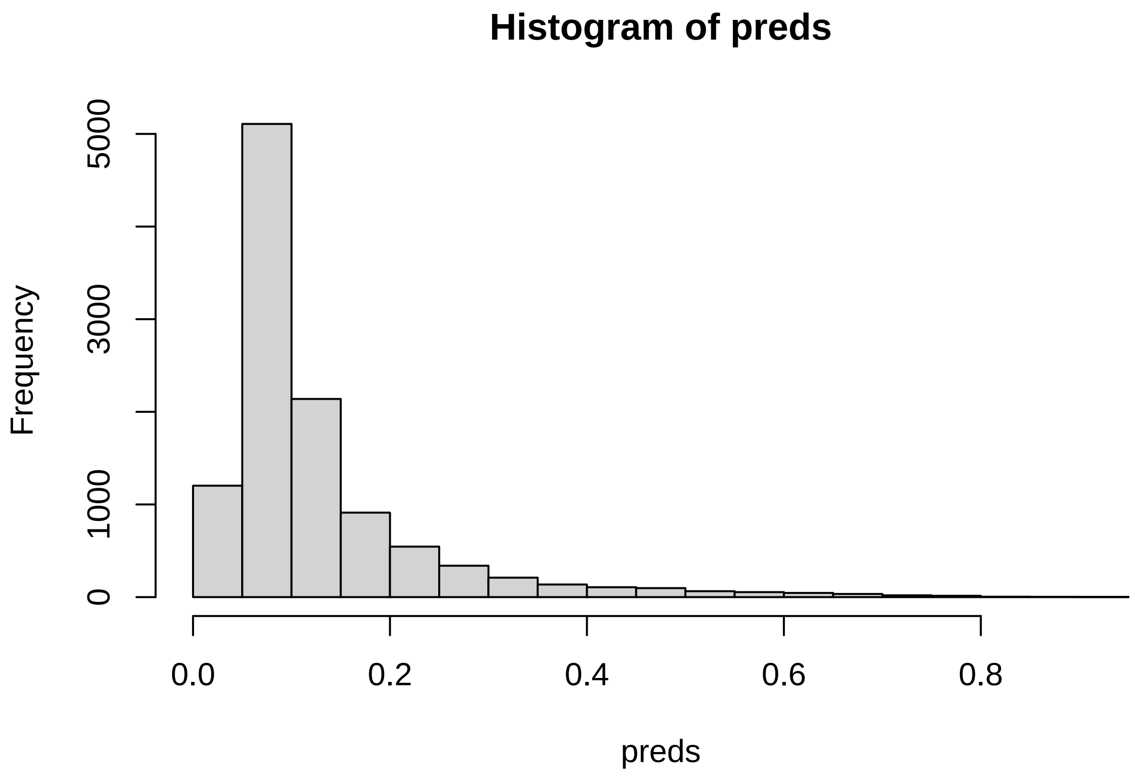
```
##        'Positive' Class : 0
##
```

```
lightgbm_parameters <- lightgbm_tune %>%
  show_best("roc_auc", n = 1) %>%
  select(-.metric, -.estimator, -.config, -mean, -n, -std_err) %>%
  as.list

Hmisc::list.tree(lightgbm_parameters)
```

```
##  lightgbm_parameters = list 4 (736 bytes)
## .  trees = integer 1= 143
## .  min_n = integer 1= 6
## .  tree_depth = integer 1= 10
## .  learn_rate = double 1= 0.022707
```

```
con <- file(sprintf('./auxiliar/model_selection/hyperparameters/%s.yaml', outcome_column), "w")
write_yaml(lightgbm_parameters, con)
close(con)
```

Minutes to run: 3.568

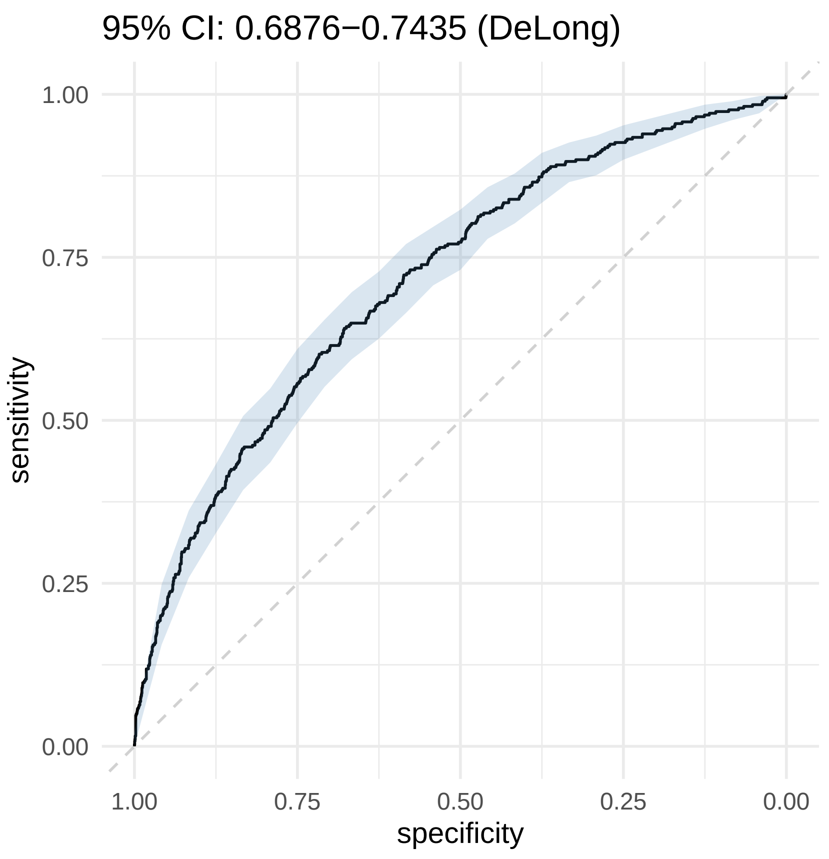


Minutes to run: 0.017

GLM

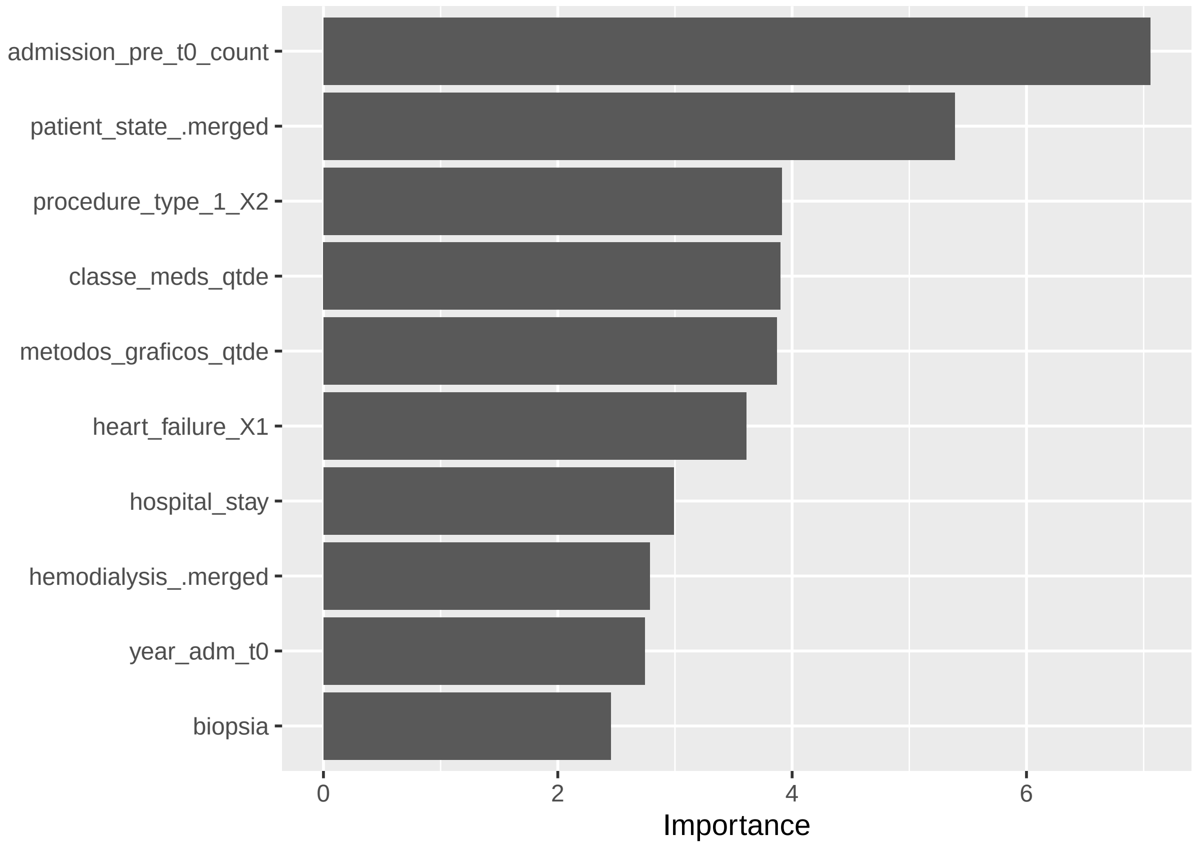
```
glmnet_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

```r
glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glm")

glmnet_workflow <-
  workflow() %>%
  add_recipe(glmnet_recipe) %>%
  add_model(glmnet_spec)

glm_fit <- glmnet_workflow %>%
  fit(df_train)

glmnet_auc <- validation(glm_fit, df_test)
```

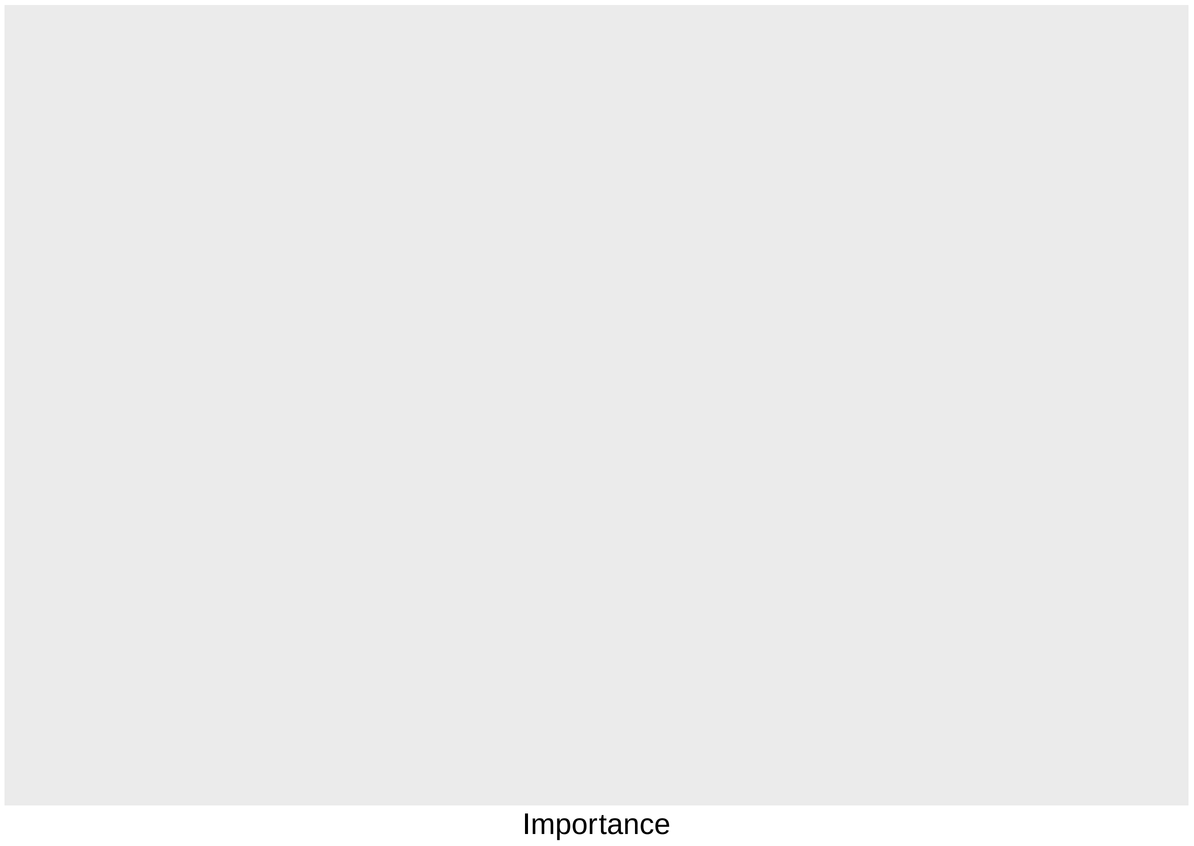


```
## [1] "Optimal Threshold: 0.11"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 1709  136
##    1  811  243
##
##                Accuracy : 0.6733
##                  95% CI : (0.6559, 0.6904)
##     No Information Rate : 0.8693
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1818
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6782
##              Specificity : 0.6412
##           Pos Pred Value : 0.9263
##           Neg Pred Value : 0.2306
##               Prevalence : 0.8693
##           Detection Rate : 0.5895
##     Detection Prevalence : 0.6364
##        Balanced Accuracy : 0.6597
##
##         'Positive' Class : 0
##
```

```
pfun_glmnet <- function(object, newdata) predict(object, newx = newdata)

extract_vip(glm_fit, pred_wrapper = pfun_glmnet,
            reference_class = "1", method = 'model')
```



```
extract_vip(glm_fit, pred_wrapper = pfun_glmnet,
            reference_class = "1", method = 'permute')
```



Minutes to run: 0.206

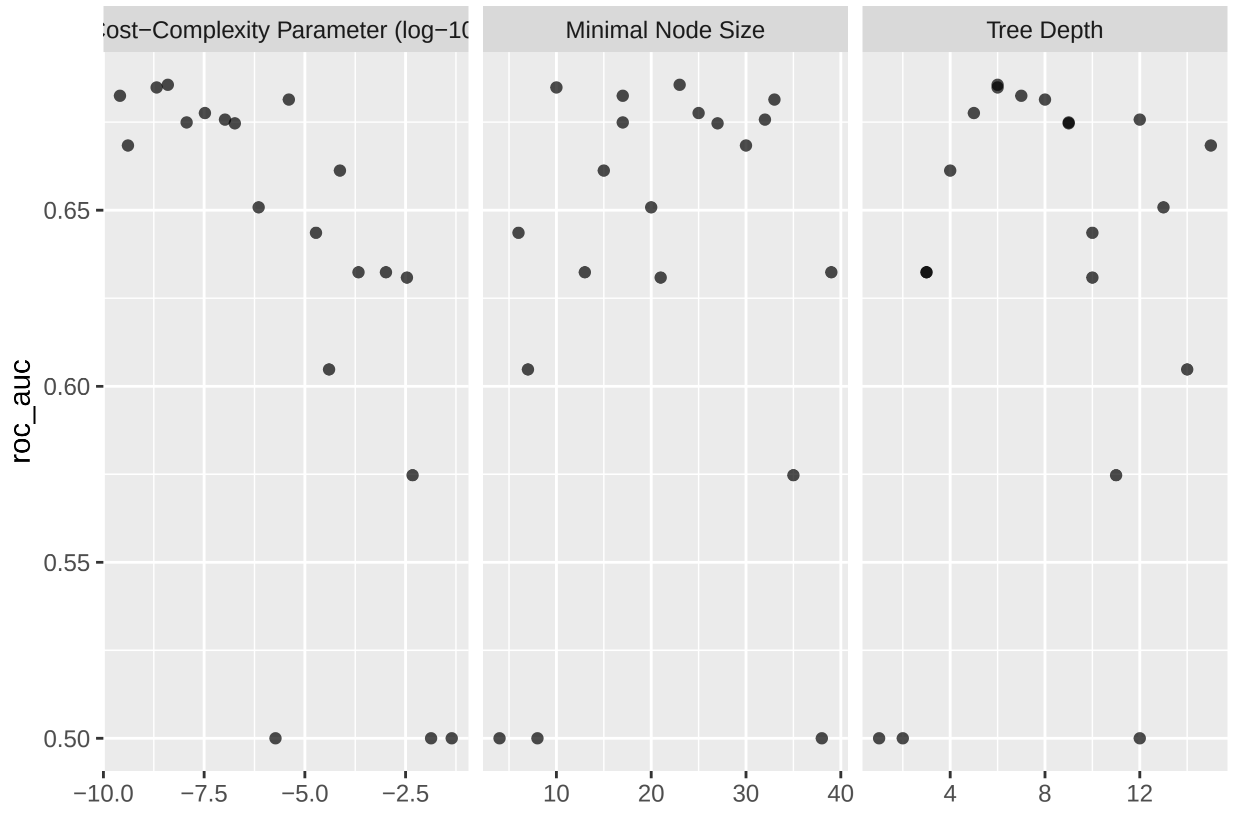
Decision Tree

```r
tree_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula, data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors())

tree_spec <-
  decision_tree(cost_complexity = tune(),
                tree_depth = tune(),
                min_n = tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

tree_grid <- grid_latin_hypercube(cost_complexity(),
                                  tree_depth(),
                                  min_n(),
                                  size = grid_size)

tree_workflow <-
  workflow() %>%
  add_recipe(tree_recipe) %>%
  add_model(tree_spec)

tree_tune <-
  tree_workflow %>%
  tune_grid(resamples = df_folds,
            grid = tree_grid)
```

```
tree_tune %>%
  collect_metrics()
```

```
## # A tibble: 40 x 9
##    cost_complexity tree_depth min_n .metric .estimator  mean     n std_err .config
##              <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1        2.57e-10          7    17 accura~ binary     0.866    10 1.56e-3 Prepro~
##  2        2.57e-10          7    17 roc_auc binary     0.682    10 5.49e-3 Prepro~
##  3        7.42e- 5          4    15 accura~ binary     0.871    10 8.72e-4 Prepro~
##  4        7.42e- 5          4    15 roc_auc binary     0.661    10 7.02e-3 Prepro~
##  5        3.42e- 3         10    21 accura~ binary     0.870    10 1.14e-3 Prepro~
##  6        3.42e- 3         10    21 roc_auc binary     0.631    10 4.48e-3 Prepro~
##  7        3.99e- 5         14     7 accura~ binary     0.840    10 2.67e-3 Prepro~
##  8        3.99e- 5         14     7 roc_auc binary     0.605    10 7.61e-3 Prepro~
##  9        1.89e- 5         10     6 accura~ binary     0.854    10 1.49e-3 Prepro~
## 10        1.89e- 5         10     6 roc_auc binary     0.644    10 8.08e-3 Prepro~
## # i 30 more rows
```

```
autoplot(tree_tune, metric = "roc_auc")
```

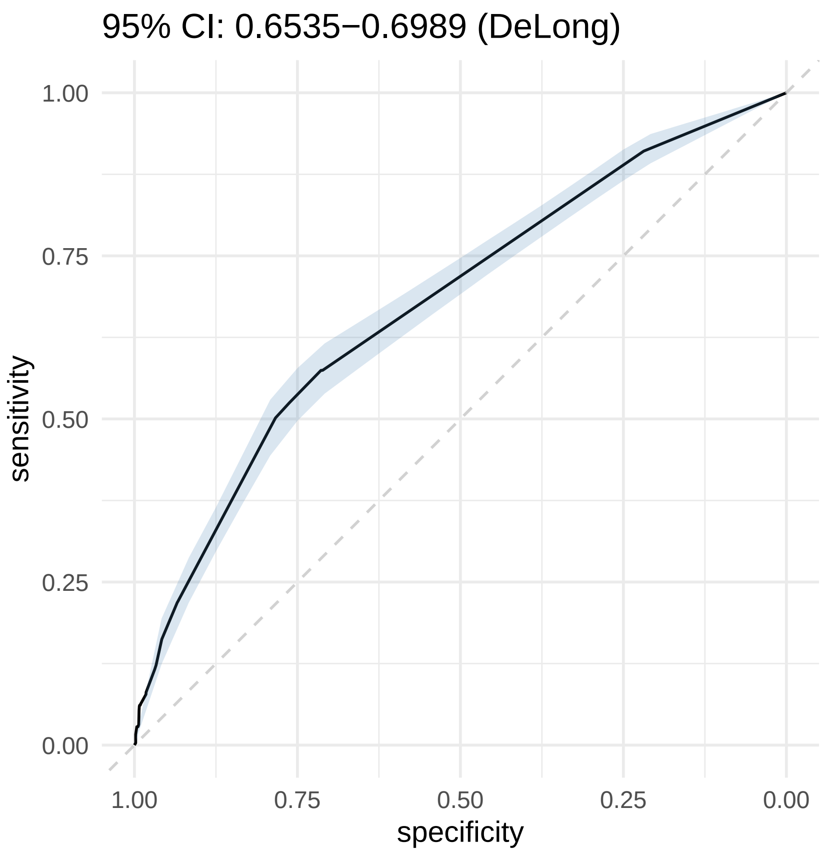


```
tree_tune %>%
  show_best("roc_auc")
```

```
## # A tibble: 5 x 9
##   cost_complexity tree_depth min_n .metric .estimator  mean     n std_err .config
##             <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        3.97e- 9          6    23 roc_auc binary     0.686    10 0.00538 Preproc~
## 2        2.09e- 9          6    10 roc_auc binary     0.685    10 0.00542 Preproc~
## 3        2.57e-10          7    17 roc_auc binary     0.682    10 0.00549 Preproc~
## 4        4.00e- 6          8    33 roc_auc binary     0.681    10 0.00453 Preproc~
## 5        3.30e- 8          5    25 roc_auc binary     0.678    10 0.00769 Preproc~
```

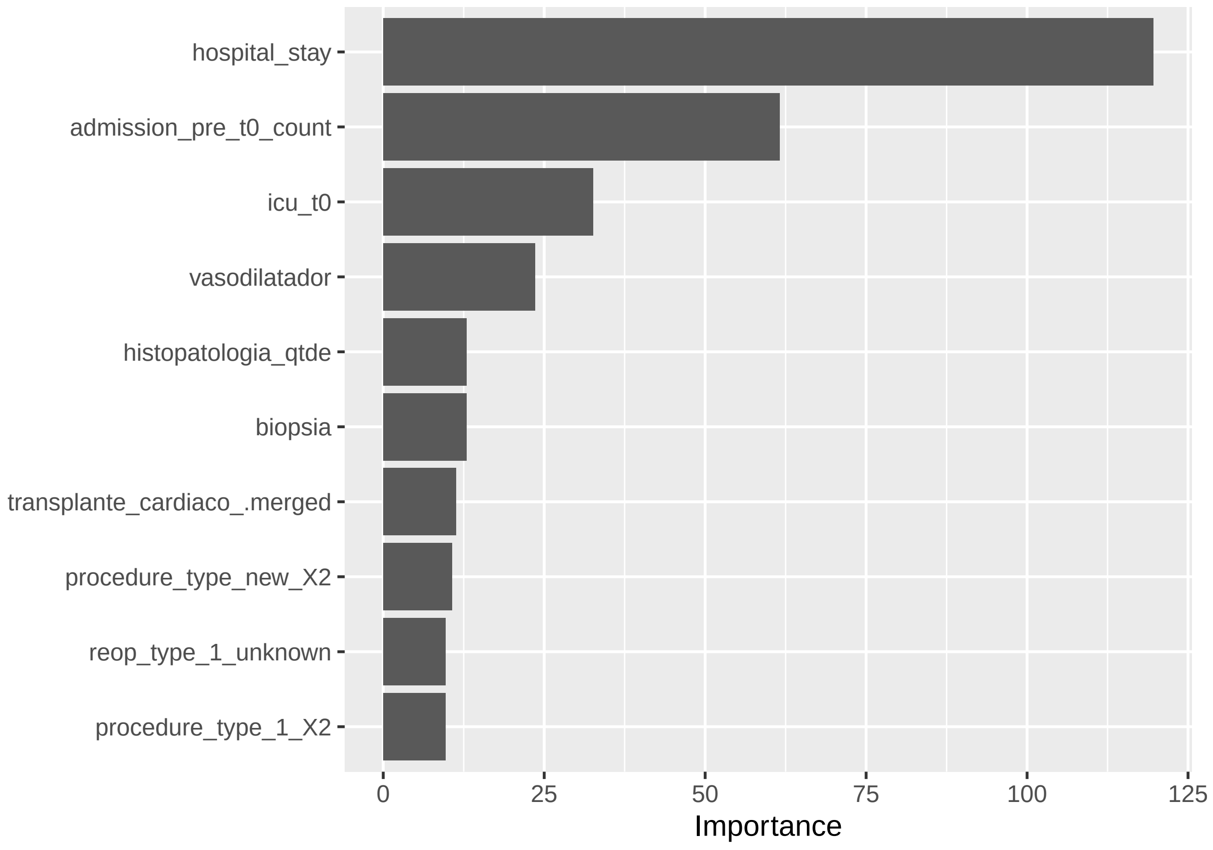
```
best_tree <- tree_tune %>%
  select_best("roc_auc")
```

```
final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

last_tree_fit <-
  final_tree_workflow %>%
  last_fit(df_split)

final_tree_fit <- extract_workflow(last_tree_fit)

tree_auc <- validation(final_tree_fit, df_test)
```



```
## [1] "Optimal Threshold: 0.11"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 2946  257
##    1 1181  347
##
##                 Accuracy : 0.696
##                   95% CI : (0.6827, 0.7091)
##      No Information Rate : 0.8723
##      P-Value [Acc > NIR] : 1
##
##                    Kappa : 0.1744
##
##   Mcnemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.7138
##              Specificity : 0.5745
##           Pos Pred Value : 0.9198
##           Neg Pred Value : 0.2271
##               Prevalence : 0.8723
##           Detection Rate : 0.6227
##     Detection Prevalence : 0.6770
##        Balanced Accuracy : 0.6442
##
##         'Positive' Class : 0
##
```

```
extract_vip(final_tree_fit, pred_wrapper = predict,
            reference_class = "0", use_matrix = FALSE,
            method = 'model')
```



```
# extract_vip(final_tree_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

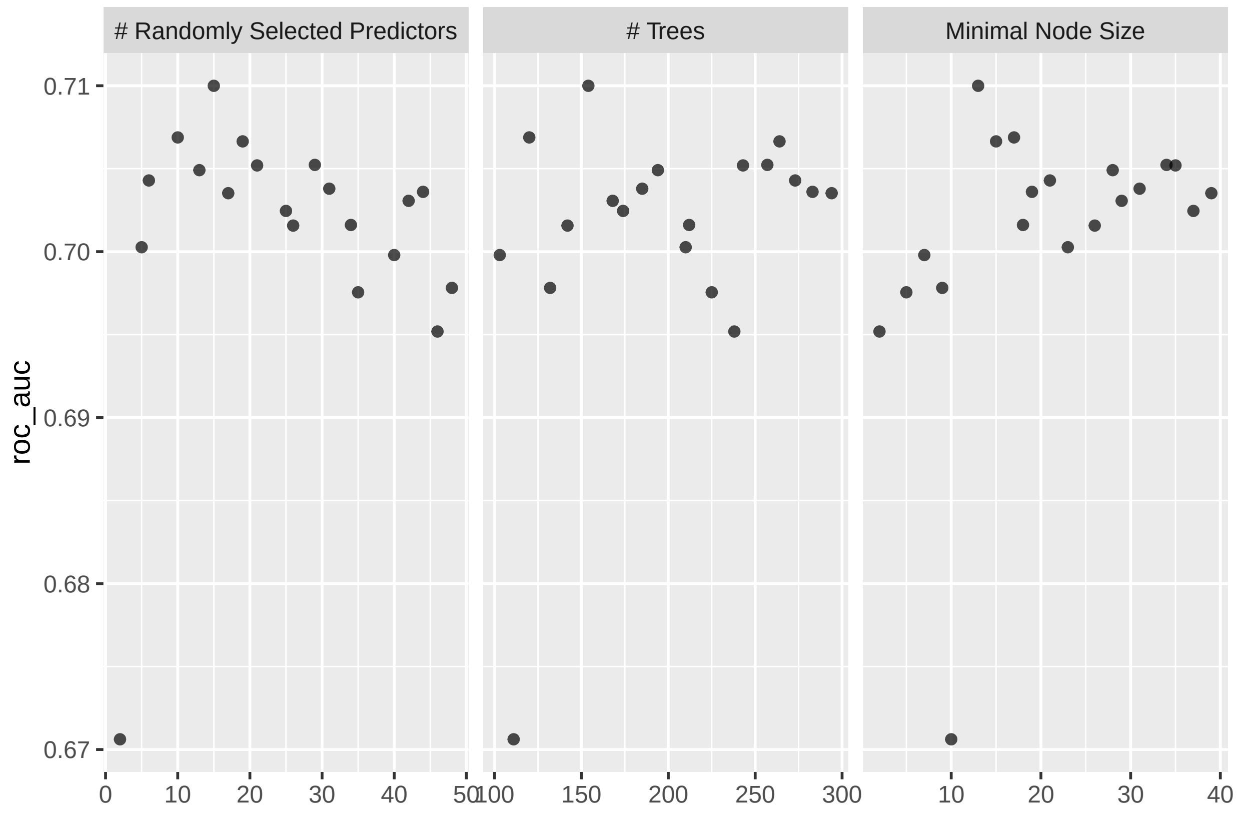
Minutes to run: 4.95

Random Forest

```
rf_recipe <-
  recipe(formula = sprintf("%s ~ .", outcome_column) %>% as.formula,
         data = df_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_other(all_nominal_predictors(), threshold = 0.05, other = ".merged") %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_impute_mean(all_numeric_predictors())

rf_spec <-
  rand_forest(mtry = tune(),
              trees = tune(),
              min_n = tune()) %>%
  set_mode("classification") %>%
  set_engine("randomForest",
             probability = TRUE,
             nthread = 8)

rf_grid <- grid_latin_hypercube(mtry(range = c(1L, 50L)),
                                trees(range = c(100L, 300L)),
                                min_n(),
                                size = grid_size)

rf_workflow <-
  workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_spec)

rf_tune <-
  rf_workflow %>%
  tune_grid(resamples = df_folds,
            grid = rf_grid)

rf_tune %>%
  collect_metrics()
```

```
## # A tibble: 40 x 9
##     mtry trees min_n .metric  .estimator  mean     n  std_err .config
##    <int> <int> <int> <chr>    <chr>      <dbl> <int>    <dbl> <chr>
##  1     2   111    10 accuracy binary     0.873    10 0.000406 Preprocessor1_Model~
##  2     2   111    10 roc_auc  binary     0.671    10 0.00754  Preprocessor1_Model~
##  3     5   210    23 accuracy binary     0.874    10 0.000752 Preprocessor1_Model~
##  4     5   210    23 roc_auc  binary     0.700    10 0.00692  Preprocessor1_Model~
##  5    35   225     5 accuracy binary     0.874    10 0.00138  Preprocessor1_Model~
##  6    35   225     5 roc_auc  binary     0.698    10 0.00806  Preprocessor1_Model~
##  7    19   264    15 accuracy binary     0.875    10 0.00122  Preprocessor1_Model~
##  8    19   264    15 roc_auc  binary     0.707    10 0.00787  Preprocessor1_Model~
##  9    42   168    29 accuracy binary     0.874    10 0.00110  Preprocessor1_Model~
## 10    42   168    29 roc_auc  binary     0.703    10 0.00781  Preprocessor1_Model~
## # i 30 more rows
```

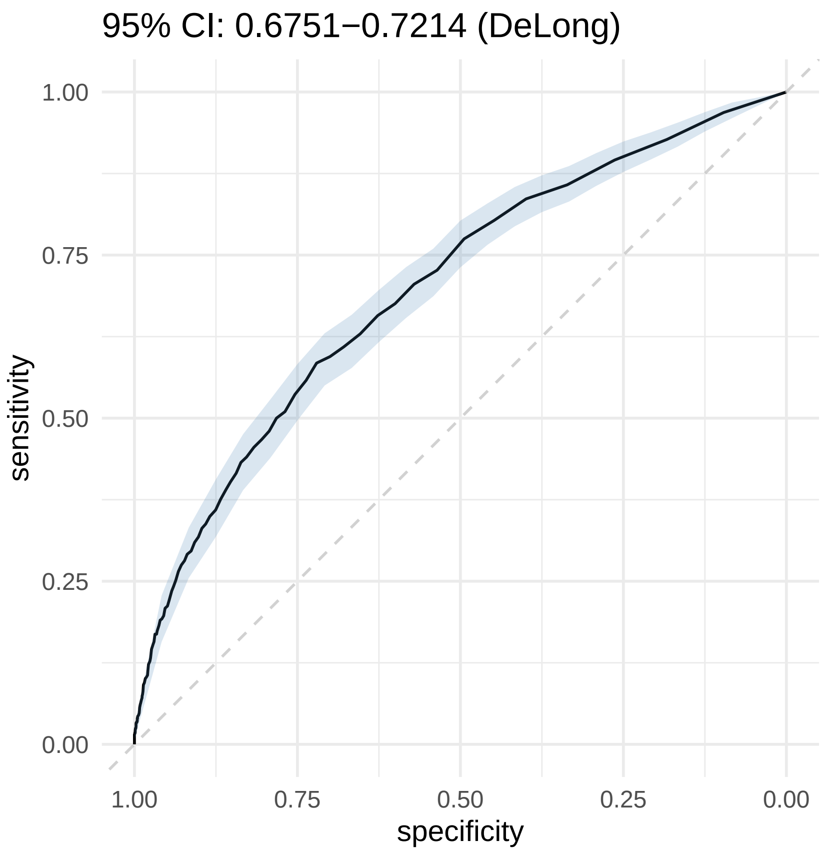
```
autoplot(rf_tune, metric = "roc_auc")
```



```
rf_tune %>%
  show_best("roc_auc")
```

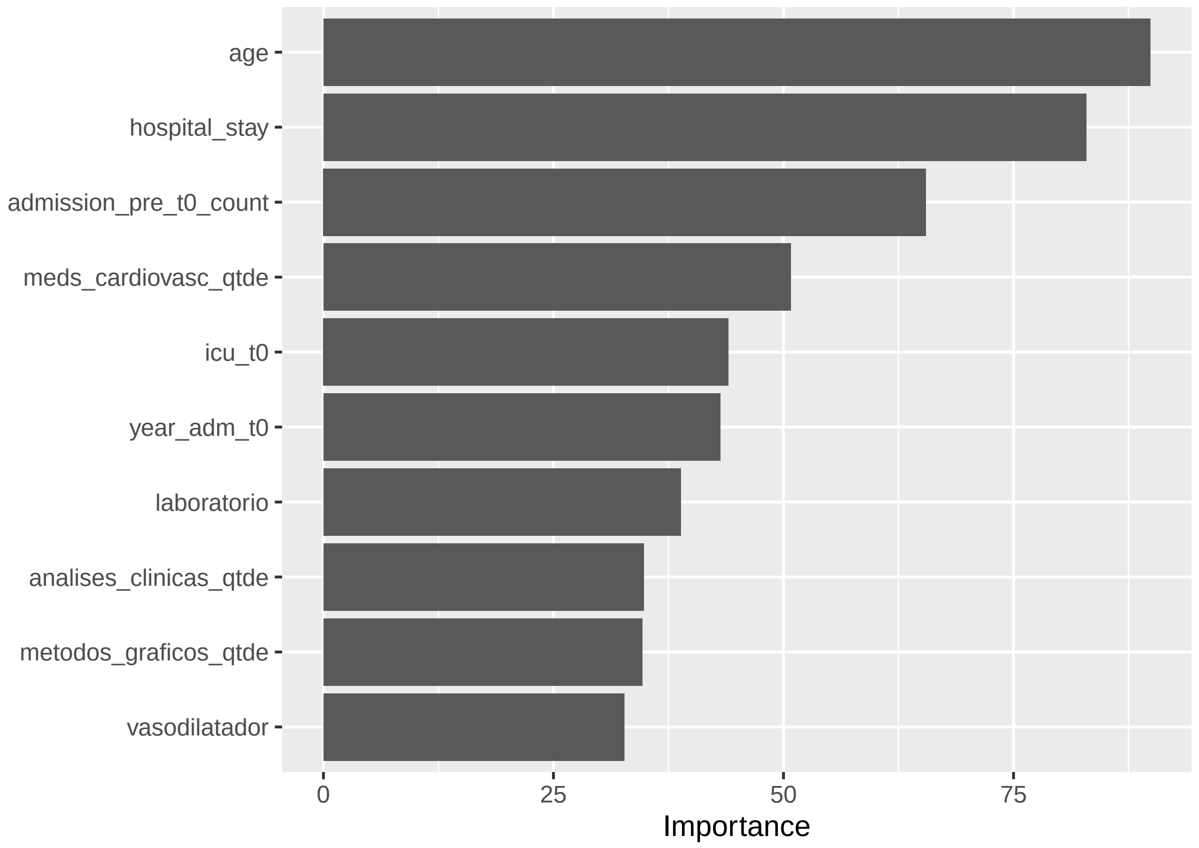
```
## # A tibble: 5 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    15   154    13 roc_auc binary     0.710    10 0.00870 Preprocessor1_Model17
## 2    10   120    17 roc_auc binary     0.707    10 0.00882 Preprocessor1_Model08
## 3    19   264    15 roc_auc binary     0.707    10 0.00787 Preprocessor1_Model04
## 4    29   257    34 roc_auc binary     0.705    10 0.00774 Preprocessor1_Model16
## 5    21   243    35 roc_auc binary     0.705    10 0.00796 Preprocessor1_Model06
```

```
best_rf <- rf_tune %>%
  select_best("roc_auc")

final_rf_workflow <-
  rf_workflow %>%
  finalize_workflow(best_rf)

last_rf_fit <-
  final_rf_workflow %>%
  last_fit(df_split)

final_rf_fit <- extract_workflow(last_rf_fit)

rf_auc <- validation(final_rf_fit, df_test)
```



```
## [1] "Optimal Threshold: 0.09"
## Confusion Matrix and Statistics
##
##     reference
## data    0    1
##    0 2974  251
##    1 1153  353
##
##                Accuracy : 0.7032
##                  95% CI : (0.69, 0.7162)
##     No Information Rate : 0.8723
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1863
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7206
##             Specificity : 0.5844
##          Pos Pred Value : 0.9222
##          Neg Pred Value : 0.2344
##              Prevalence : 0.8723
##          Detection Rate : 0.6286
##    Detection Prevalence : 0.6817
##       Balanced Accuracy : 0.6525
##
##        'Positive' Class : 0
##
```

```
pfun_rf <- function(object, newdata) predict(object, data = newdata)

extract_vip(final_rf_fit, pred_wrapper = predict,
            reference_class = "1", use_matrix = FALSE,
            method = 'model')
```

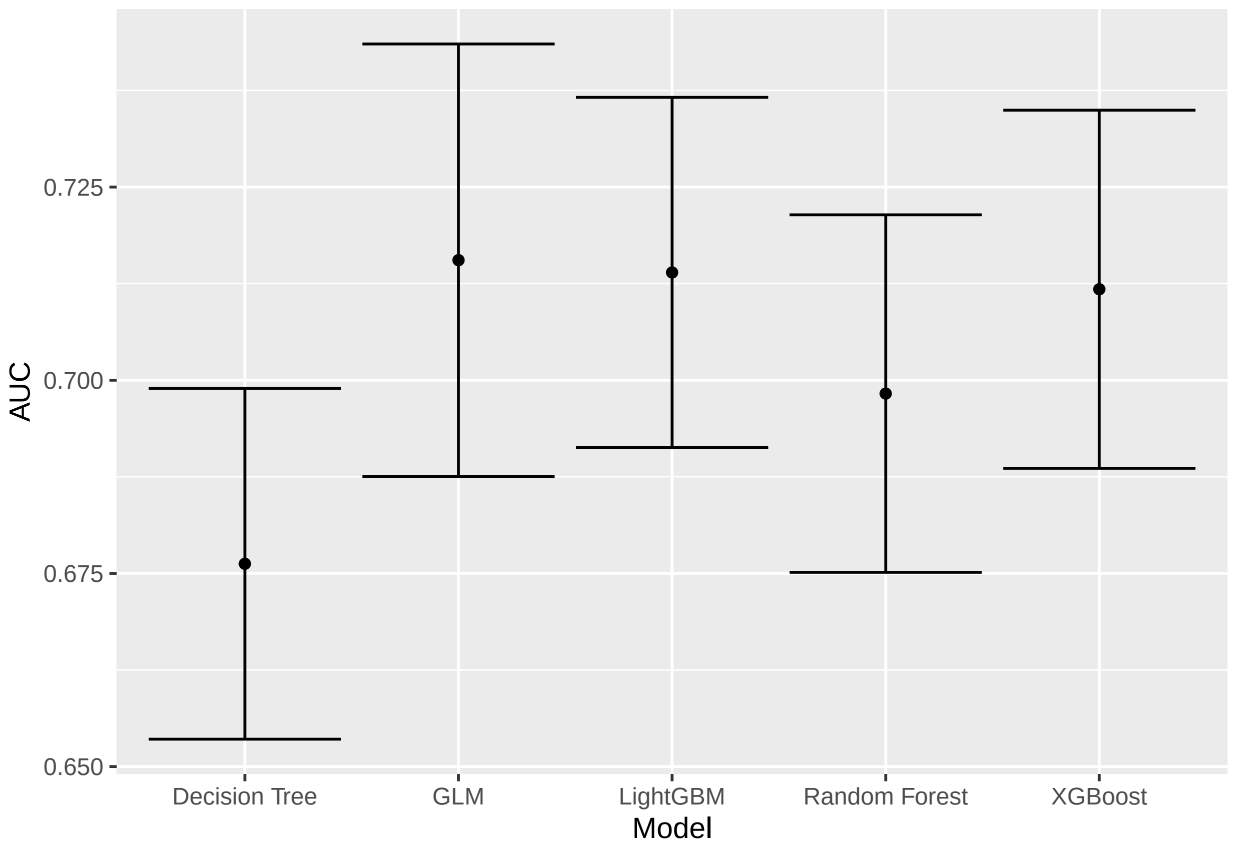


```r
# extract_vip(final_rf_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

Minutes to run: 67.47

Models Comparison

```r
if (RUN_ALL_MODELS) {
  df_auc <- tibble::tribble(
    ~Model, ~`AUC`, ~`Lower Limit`, ~`Upper Limit`,
    'XGBoost', as.numeric(xgboost_auc$auc), xgboost_auc$ci[1], xgboost_auc$ci[3],
    'LightGBM', as.numeric(lightgbm_auc$auc), lightgbm_auc$ci[1], lightgbm_auc$ci[3],
    'GLM', as.numeric(glmnet_auc$auc), glmnet_auc$ci[1], glmnet_auc$ci[3],
    'Decision Tree', as.numeric(tree_auc$auc), tree_auc$ci[1], tree_auc$ci[3],
    'Random Forest', as.numeric(rf_auc$auc), rf_auc$ci[1], rf_auc$ci[3]
  ) %>%
    mutate(Target = outcome_column)
} else {
  df_auc <- tibble::tribble(
    ~Model, ~`AUC`, ~`Lower Limit`, ~`Upper Limit`,
    'LightGBM', as.numeric(lightgbm_auc$auc), lightgbm_auc$ci[1], lightgbm_auc$ci[3]
  ) %>%
    mutate(Target = outcome_column)
}

df_auc %>%
  ggplot(aes(x = Model, y = AUC, ymin = `Lower Limit`, ymax = `Upper Limit`)) +
    geom_point() +
    geom_errorbar()
```



```r
write_csv(df_auc, sprintf("./auxiliar/model_selection/performance/%s.csv", outcome_column))
```

Minutes to run: 0.002