

Model Selection - readmission_30d

Eduardo Yuki Yada

Global parameters

```
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 (952 bytes)
## . outcome_column = character 1= readmission_30d
## . k = double 1= 10
## . grid_size = double 1= 20
## . repeats = double 1= 2
## . RUN_ALL_MODELS = logical 1= TRUE
```

Minutes to run: 0

Imports

```
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: 0.036

Loading data

```
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: 0.007

```
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. education_level
## 02. underlying_heart_disease
## 03. heart_disease
## 04. nyha_basal
## 05. prior_mi
## 06. heart_failure
## 07. transplant
## 08. endocardites
## 09. hemodialysis
## 10. comorbidities_count
## 11. procedure_type_1
## 12. reop_type_1
## 13. procedure_type_new

```

14. cied_final_1
15. cied_final_group_1
16. admission_pre_t0_count
17. admission_pre_t0_180d
18. icu_t0
19. dialysis_t0
20. admission_t0_emergency
21. aco
22. antiarritmico
23. betabloqueador
24. ieca_bra
25. dva
26. digoxina
27. estatina
28. diuretico
29. vasodilatador
30. insuf_cardiaca
31. espironolactona
32. bloq_calcio
33. antiplaquetario_ev
34. insulina
35. anticonvulsivante
36. psicofarmacos
37. antifungico
38. antiviral
39. classe_meds_qtde
40. meds_cardiovasc_qtde
41. meds_antimicrobianos
42. ventilacao_mecanica
43. cec
44. transplante_cardiaco
45. outros_proced_cirurgicos
46. icp
47. intervencao_cv
48. cateterismo
49. eletrofisiologia
50. cateter_venoso_central
51. proced_invasivos_qtde
52. cve_desf
53. transfusao
54. equipe_multiprof
55. holter
56. metodos_graficos_qtde
57. laboratorio
58. cultura
59. analises_clinicas_qtde
60. citologia
61. biopsia
62. histopatologia_qtde
63. angio_rm
64. angio_tc
65. cintilografia
66. ecocardiograma
67. endoscopia
68. flebografia
69. pet_ct
70. ultrassom
71. tomografia
72. ressonancia
73. exames_imagem_qtde
74. bic

```
## 75. mpp
## 76. 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: 0.01

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",
             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)
```

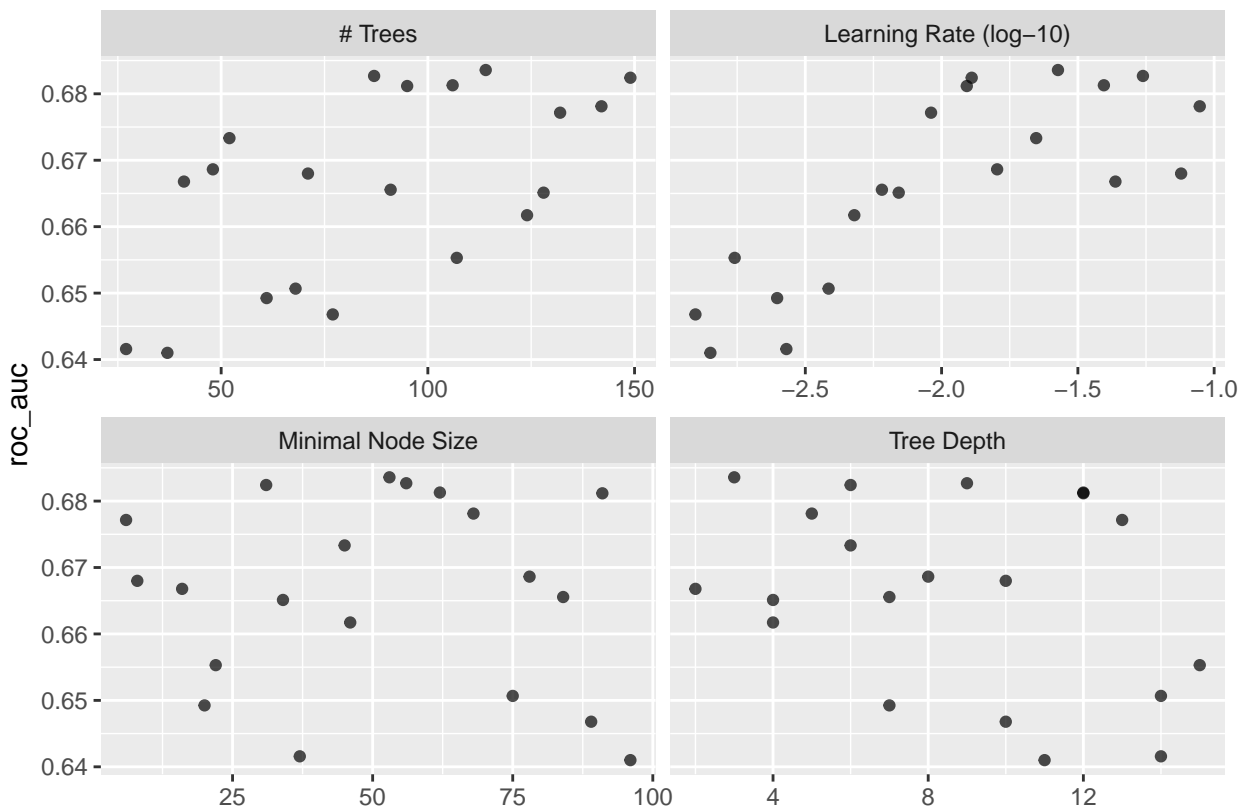
```
xgboost_tune %>%
  show_best("roc_auc")
```

```
## # 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	114	53	3	0.0267	roc_auc	binary	0.684	10	0.0101	Preprocessor1_Model11
## 2	87	56	9	0.0548	roc_auc	binary	0.683	10	0.00927	Preprocessor1_Model12
## 3	149	31	6	0.0129	roc_auc	binary	0.682	10	0.00657	Preprocessor1_Model106
## 4	106	62	12	0.0394	roc_auc	binary	0.681	10	0.00902	Preprocessor1_Model13
## 5	95	91	12	0.0124	roc_auc	binary	0.681	10	0.00801	Preprocessor1_Model19

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

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

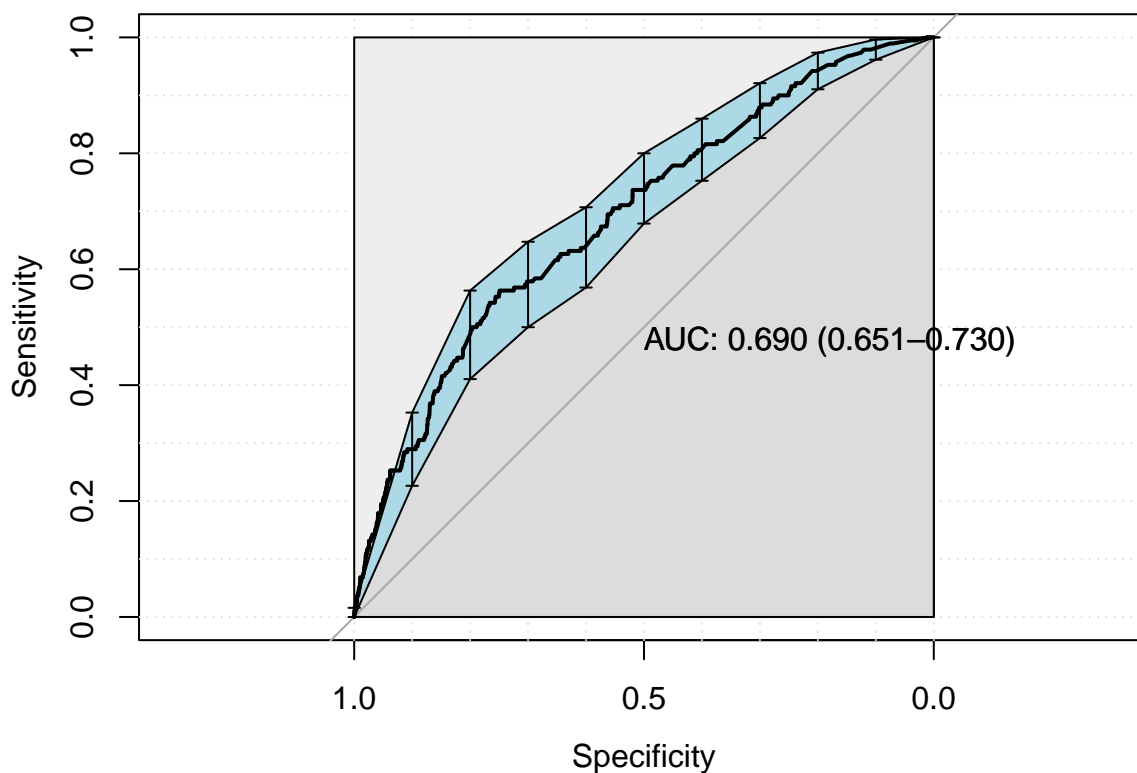


```
final_xgboost_workflow <-
  xgboost_workflow %>%
  finalize_workflow(best_xgboost)

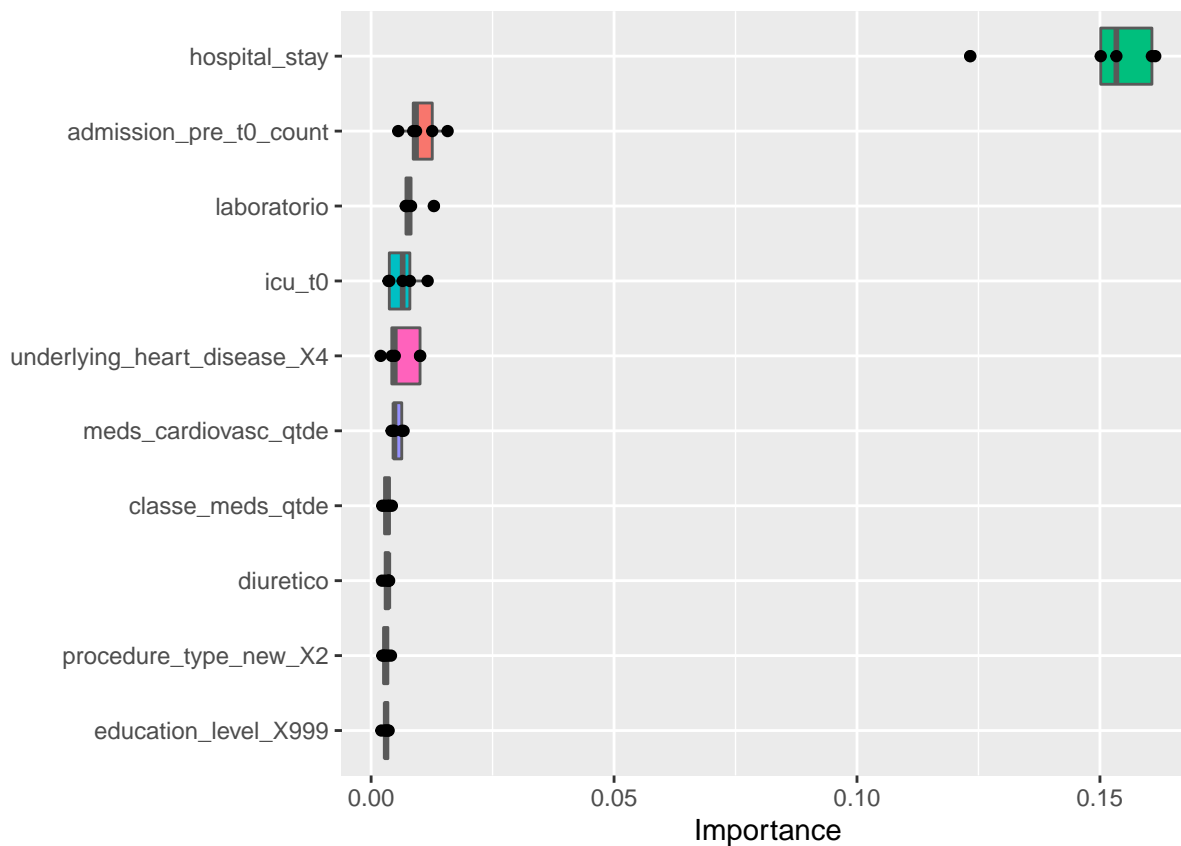
last_xgboost_fit <-
  final_xgboost_workflow %>%
  last_fit(df_split)

final_xgboost_fit <- extract_workflow(last_xgboost_fit)

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



```
## [1] "Optimal Threshold: 0.07"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 3402   83
##    1 1138  107
##
##              Accuracy : 0.7419
##              95% CI   : (0.7291, 0.7543)
##    No Information Rate : 0.9598
##    P-Value [Acc > NIR] : 1
##
##              Kappa   : 0.0854
##
## Mcnemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.74934
##              Specificity : 0.56316
##              Pos Pred Value : 0.97618
##              Neg Pred Value : 0.08594
##              Prevalence : 0.95983
##              Detection Rate : 0.71924
##              Detection Prevalence : 0.73679
##              Balanced Accuracy : 0.65625
##
##              'Positive' Class : 0
##
extract_vip(final_xgboost_fit, pred_wrapper = predict,
            reference_class = "0")
```



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

Minutes to run: 4.275

Boosted Tree (LightGBM)

```
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)
```

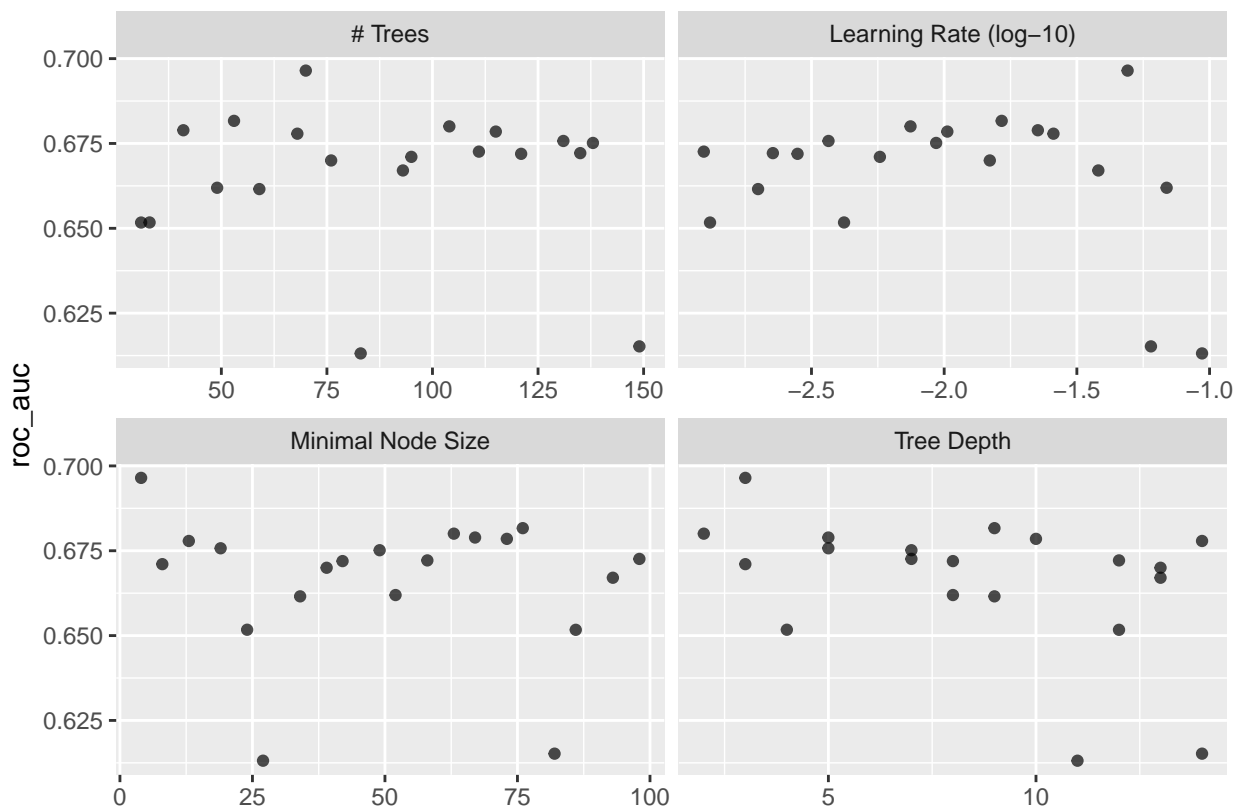
```
lightgbm_tune %>%  
  show_best("roc_auc")
```

```
## # 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    70     4        3    0.0491 roc_auc binary  0.696    10 0.0104 Preprocessor1_Model101  
## 2    53    76        9    0.0165 roc_auc binary  0.682    10 0.00882 Preprocessor1_Model116  
## 3   104    63        2    0.00747 roc_auc binary  0.680    10 0.00676 Preprocessor1_Model113  
## 4    41    67        5    0.0225 roc_auc binary  0.679    10 0.00945 Preprocessor1_Model114  
## 5   115    73       10    0.0103 roc_auc binary  0.678    10 0.00933 Preprocessor1_Model115
```

```
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 <-
```



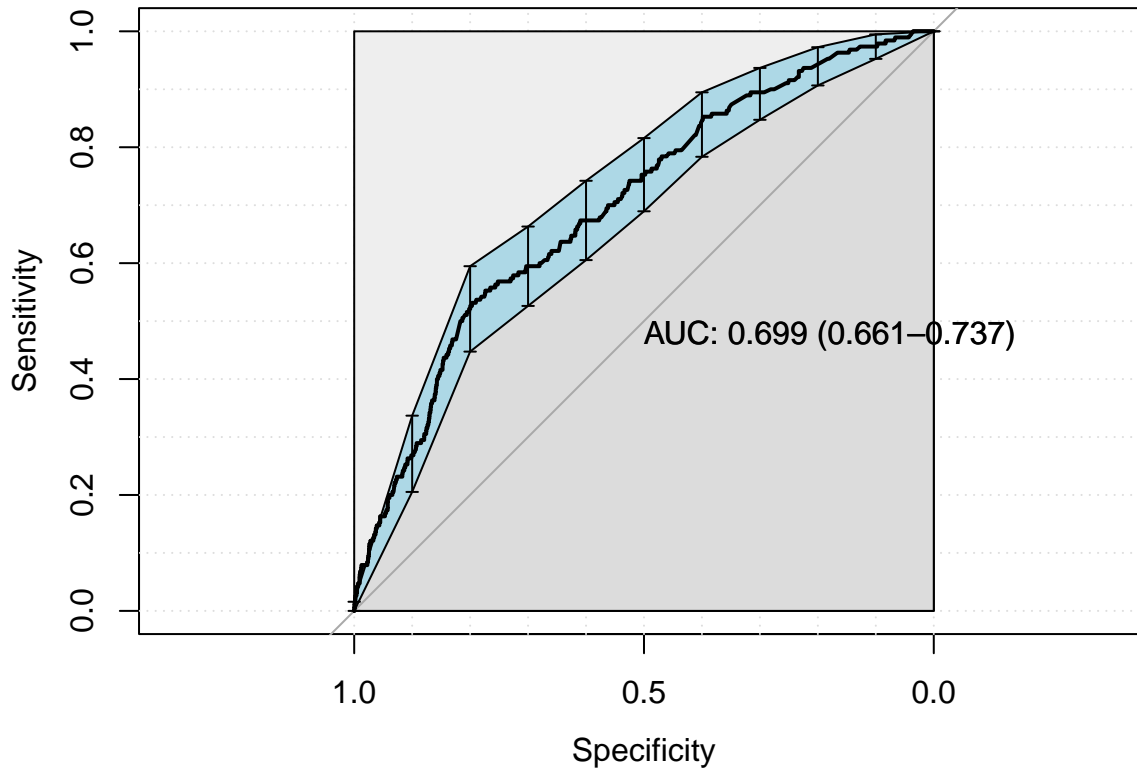
```

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: 0.05"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
## 0 3617   89
## 1  923  101
##
##              Accuracy : 0.786
##              95% CI   : (0.7741, 0.7977)
##    No Information Rate : 0.9598
##    P-Value [Acc > NIR] : 1
##
##              Kappa   : 0.1058
##
##  McNemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.79670
##              Specificity : 0.53158
##              Pos Pred Value : 0.97598
##              Neg Pred Value : 0.09863
##              Prevalence   : 0.95983
##              Detection Rate : 0.76469
##              Detection Prevalence : 0.78351
##              Balanced Accuracy : 0.66414
##

```

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

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

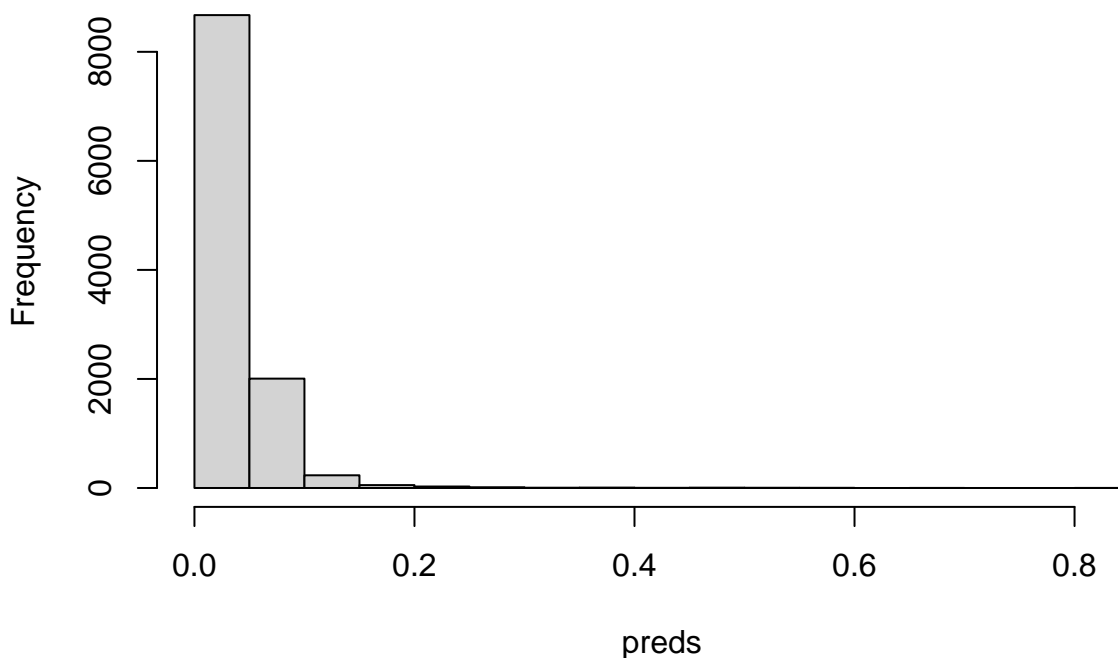
```
Hmisc::list.tree(lightgbm_parameters)
```

```
## lightgbm_parameters = list 4 (736 bytes)
## . trees = integer 1= 70
## . min_n = integer 1= 4
## . tree_depth = integer 1= 3
## . learn_rate = double 1= 0.049118
```

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

Minutes to run: 3.897

Histogram of preds



Minutes to run:

0.005

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())
```

```

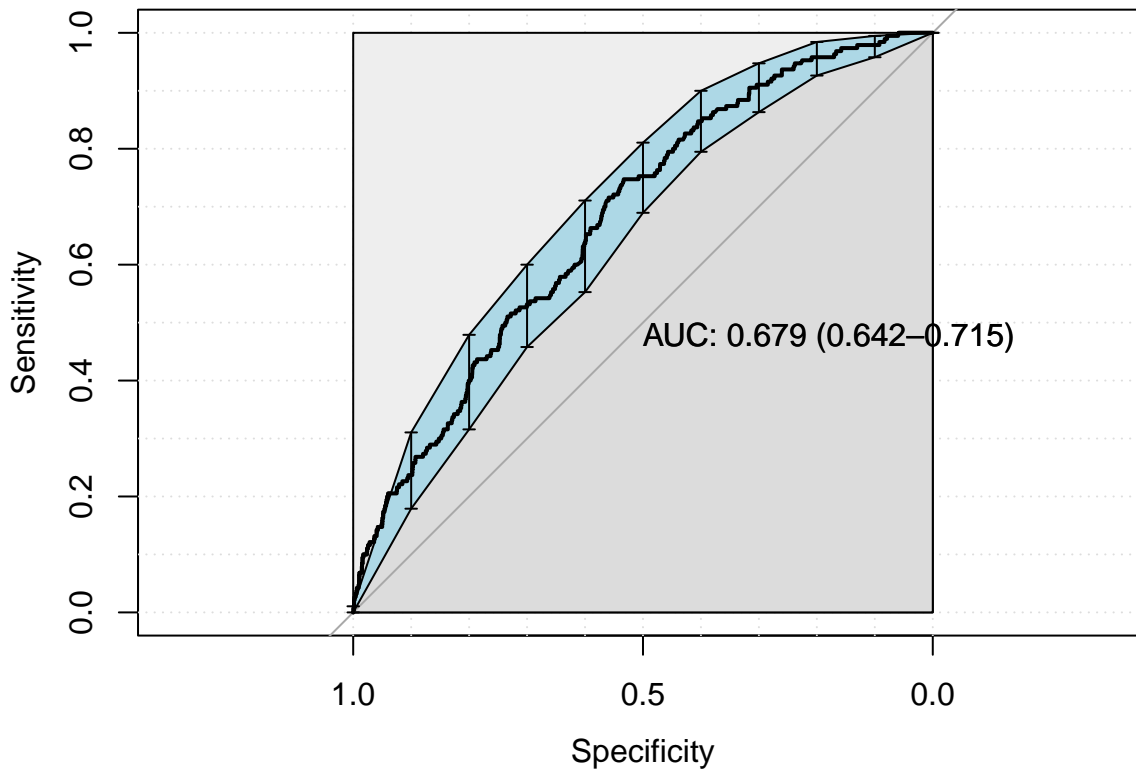
glmnet_spec <-
  logistic_reg(penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

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)

```



```

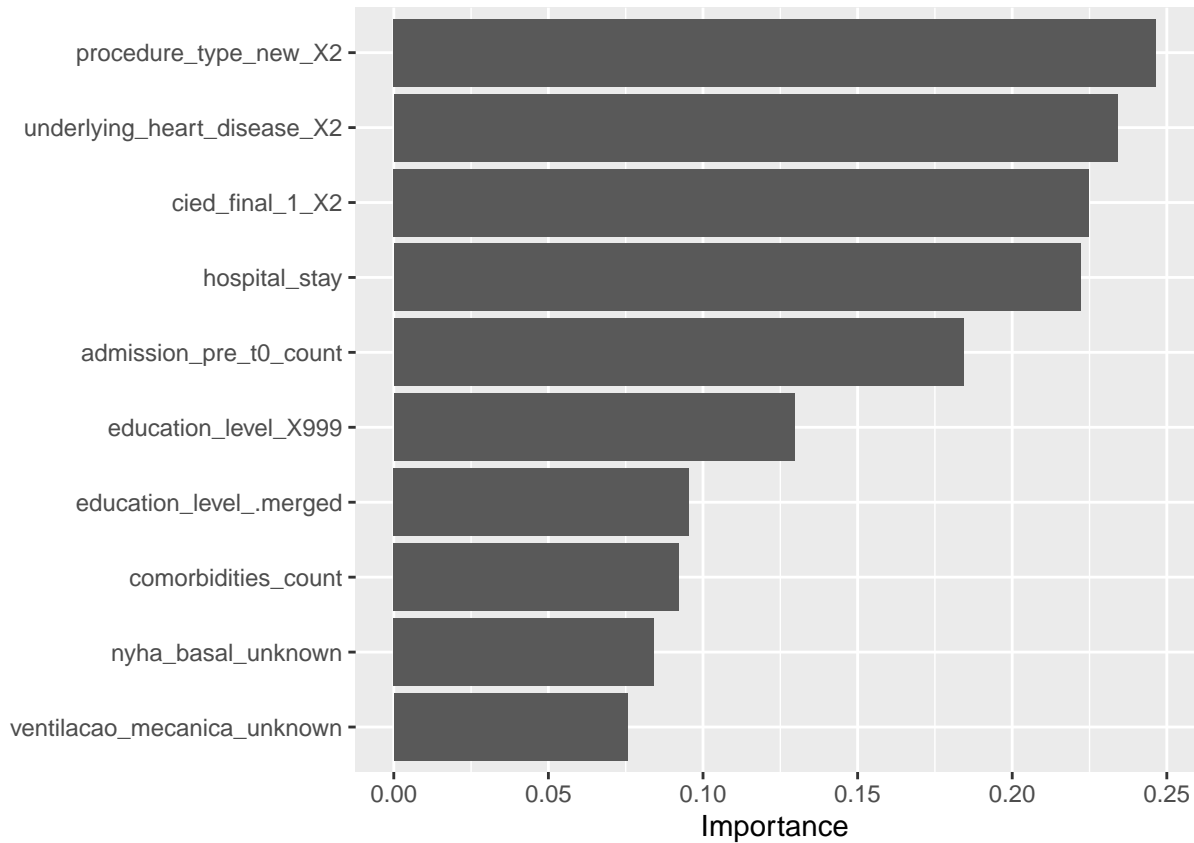
## [1] "Optimal Threshold: 0.03"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
## 0 2421   48
## 1 2119  142
##
##               Accuracy : 0.5419
##               95% CI   : (0.5275, 0.5561)
##      No Information Rate : 0.9598
##      P-Value [Acc > NIR] : 1
##
##               Kappa   : 0.0451
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Sensitivity : 0.5333

```

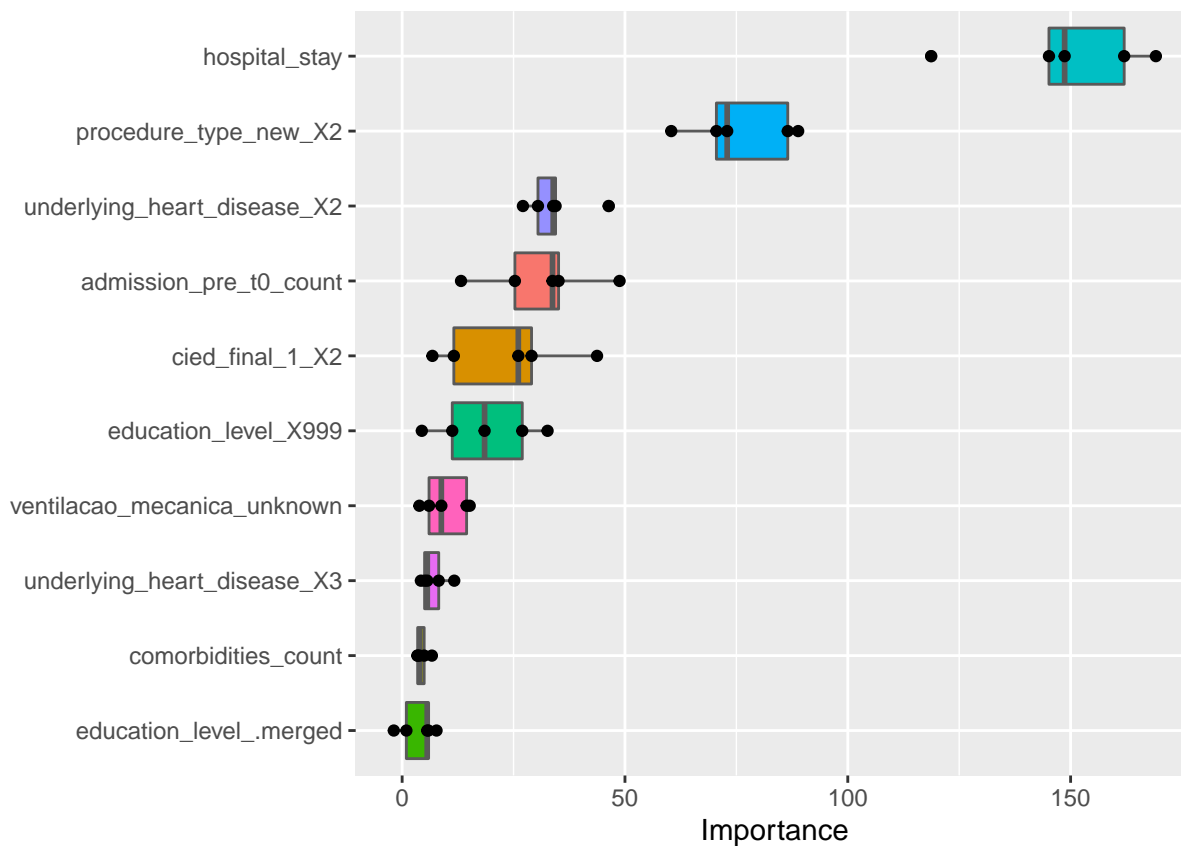
```
##          Specificity : 0.7474
##          Pos Pred Value : 0.9806
##          Neg Pred Value : 0.0628
##          Prevalence : 0.9598
##          Detection Rate : 0.5118
##          Detection Prevalence : 0.5220
##          Balanced Accuracy : 0.6403
##
##          'Positive' Class : 0
##
```

```
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:

1.949

Decision Tree

```
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)

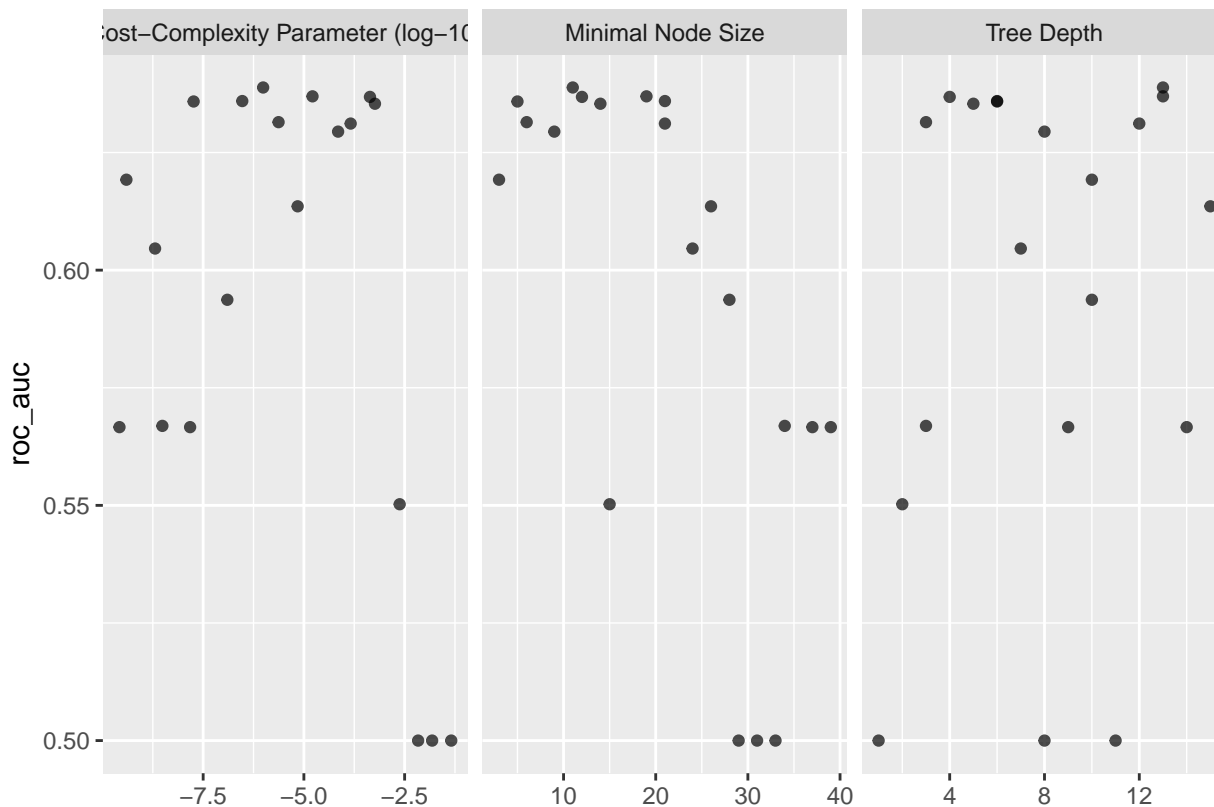
tree_workflow <-
  workflow() %>%
  add_recipe(tree_recipe) %>%
  add_model(tree_spec)

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  0.00000000202         7    24 accuracy binary    0.962   10 0.00218 Preprocessor1_Model101
## 2  0.00000000202         7    24 roc_auc  binary    0.605   10 0.0186  Preprocessor1_Model101
## 3  0.00000000307         3    34 accuracy binary    0.962   10 0.00213 Preprocessor1_Model102
## 4  0.00000000307         3    34 roc_auc  binary    0.567   10 0.0231  Preprocessor1_Model102
## 5  0.000000235          3     6 accuracy binary    0.962   10 0.00211 Preprocessor1_Model103
## 6  0.000000235          3     6 roc_auc  binary    0.631   10 0.00812 Preprocessor1_Model103
## 7  0.000000126         10    28 accuracy binary    0.961   10 0.00250 Preprocessor1_Model104
## 8  0.000000126         10    28 roc_auc  binary    0.594   10 0.0219  Preprocessor1_Model104
## 9  0.00239              2    15 accuracy binary    0.962   10 0.00217 Preprocessor1_Model105
## 10 0.00239              2    15 roc_auc  binary    0.550   10 0.0209  Preprocessor1_Model105
## # ... with 30 more rows
## # i Use 'print(n = ...)' to see 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  0.000000974         13    11 roc_auc binary    0.639   10 0.00943 Preprocessor1_Model106
## 2  0.0000164          13    19 roc_auc binary    0.637   10 0.00875 Preprocessor1_Model110
## 3  0.000437           4    12 roc_auc binary    0.637   10 0.00844 Preprocessor1_Model117
## 4  0.000000296         6    21 roc_auc binary    0.636   10 0.00761 Preprocessor1_Model118
## 5  0.0000000183         6     5 roc_auc binary    0.636   10 0.00886 Preprocessor1_Model120
```

```

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

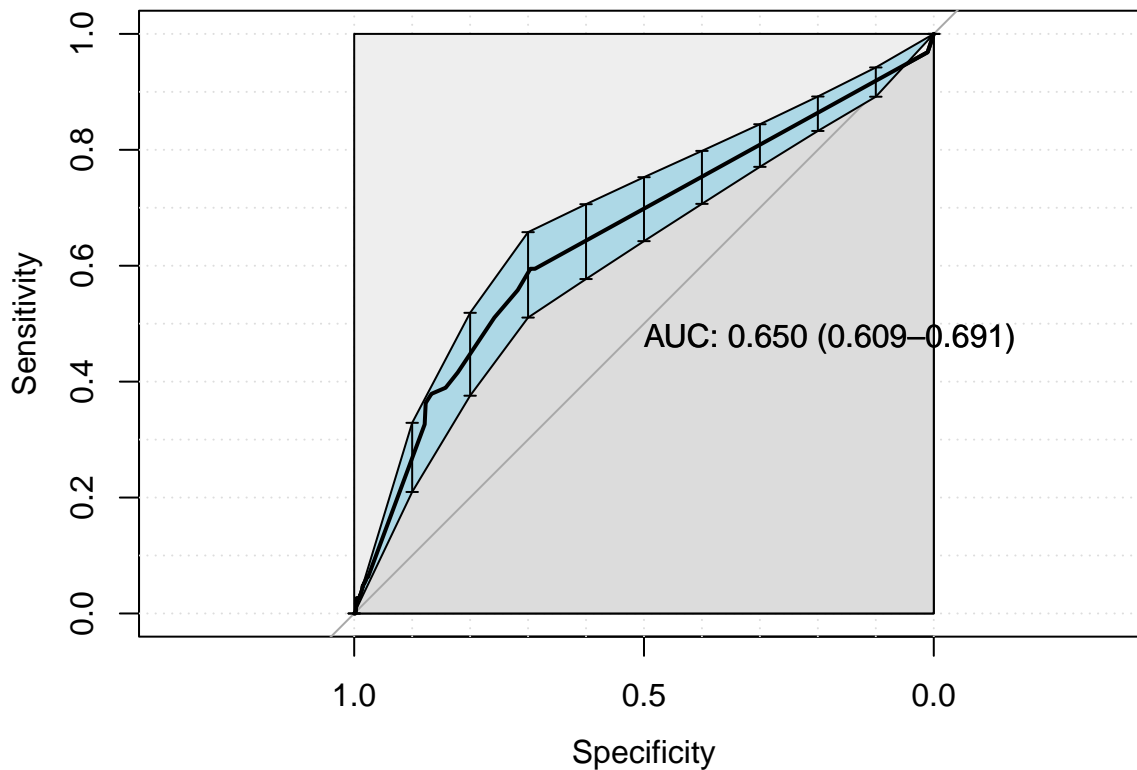
final_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_tree)

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)

```



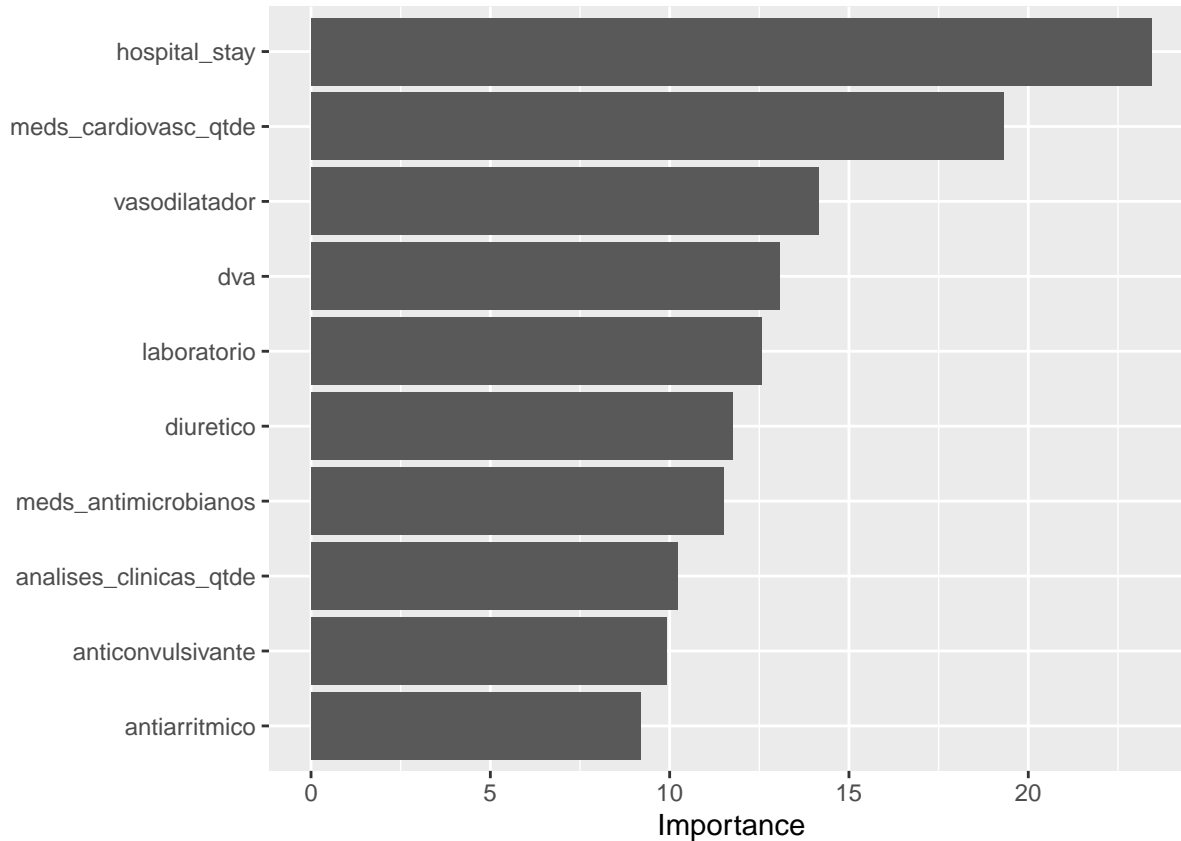
```

## [1] "Optimal Threshold: 0.03"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
## 0 3160   77
## 1 1380  113
##
##              Accuracy : 0.692
##              95% CI : (0.6786, 0.7051)
##    No Information Rate : 0.9598
##    P-Value [Acc > NIR] : 1
##
##              Kappa : 0.0679
##
##  McNemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.69604

```

```
##          Specificity : 0.59474
##          Pos Pred Value : 0.97621
##          Neg Pred Value : 0.07569
##          Prevalence : 0.95983
##          Detection Rate : 0.66808
##          Detection Prevalence : 0.68436
##          Balanced Accuracy : 0.64539
##
##          '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: 4.448

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)

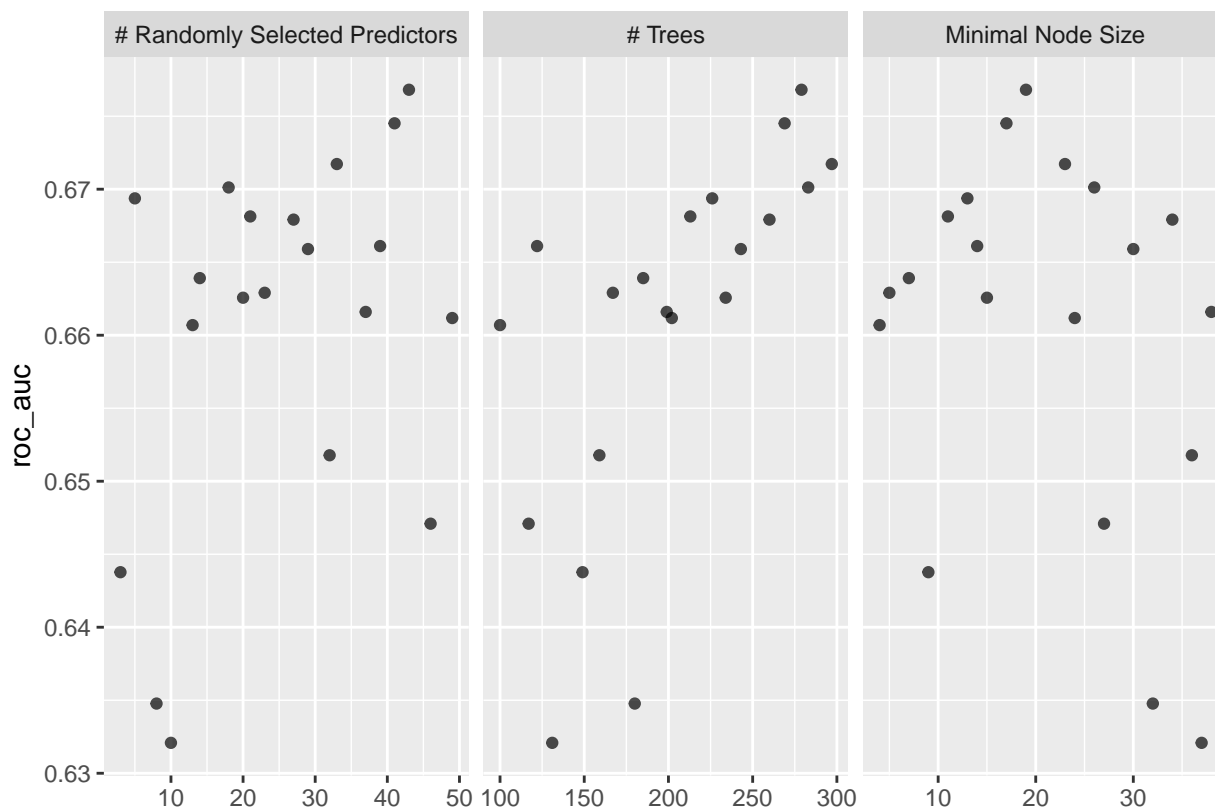
rf_tune <-
  rf_workflow %>%
  tune_grid(resamples = df_folds,
            grid = rf_grid)

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     23   167     5 accuracy binary    0.963    10 0.00223 Preprocessor1_Model01
## 2     23   167     5 roc_auc  binary    0.663    10 0.0151  Preprocessor1_Model01
## 3     14   185     7 accuracy binary    0.963    10 0.00225 Preprocessor1_Model02
## 4     14   185     7 roc_auc  binary    0.664    10 0.0107  Preprocessor1_Model02
## 5     27   260    34 accuracy binary    0.963    10 0.00225 Preprocessor1_Model03
## 6     27   260    34 roc_auc  binary    0.668    10 0.0105  Preprocessor1_Model03
## 7     20   234    15 accuracy binary    0.963    10 0.00222 Preprocessor1_Model04
## 8     20   234    15 roc_auc  binary    0.663    10 0.0105  Preprocessor1_Model04
## 9      8   180    32 accuracy binary    0.963    10 0.00219 Preprocessor1_Model05
## 10     8   180    32 roc_auc  binary    0.635    10 0.0112  Preprocessor1_Model05
## # ... with 30 more rows
## # i Use 'print(n = ...)' to see 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    43   279    19 roc_auc binary  0.677    10  0.0108 Preprocessor1_Model107
## 2    41   269    17 roc_auc binary  0.675    10  0.0137 Preprocessor1_Model114
## 3    33   297    23 roc_auc binary  0.672    10  0.0115 Preprocessor1_Model115
## 4    18   283    26 roc_auc binary  0.670    10  0.0108 Preprocessor1_Model106
## 5     5   226    13 roc_auc binary  0.669    10  0.0110 Preprocessor1_Model120
```

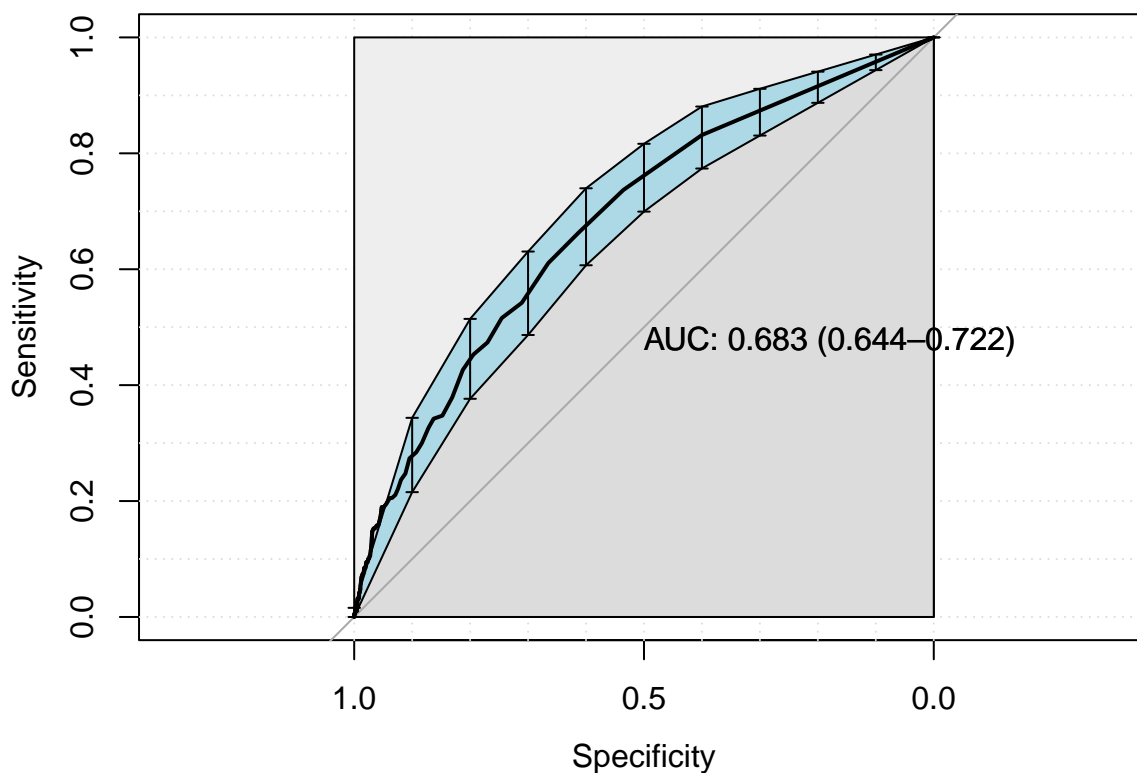
```
best_rf <- rf_tune %>%
  select_best("roc_auc")
```

```
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)
```

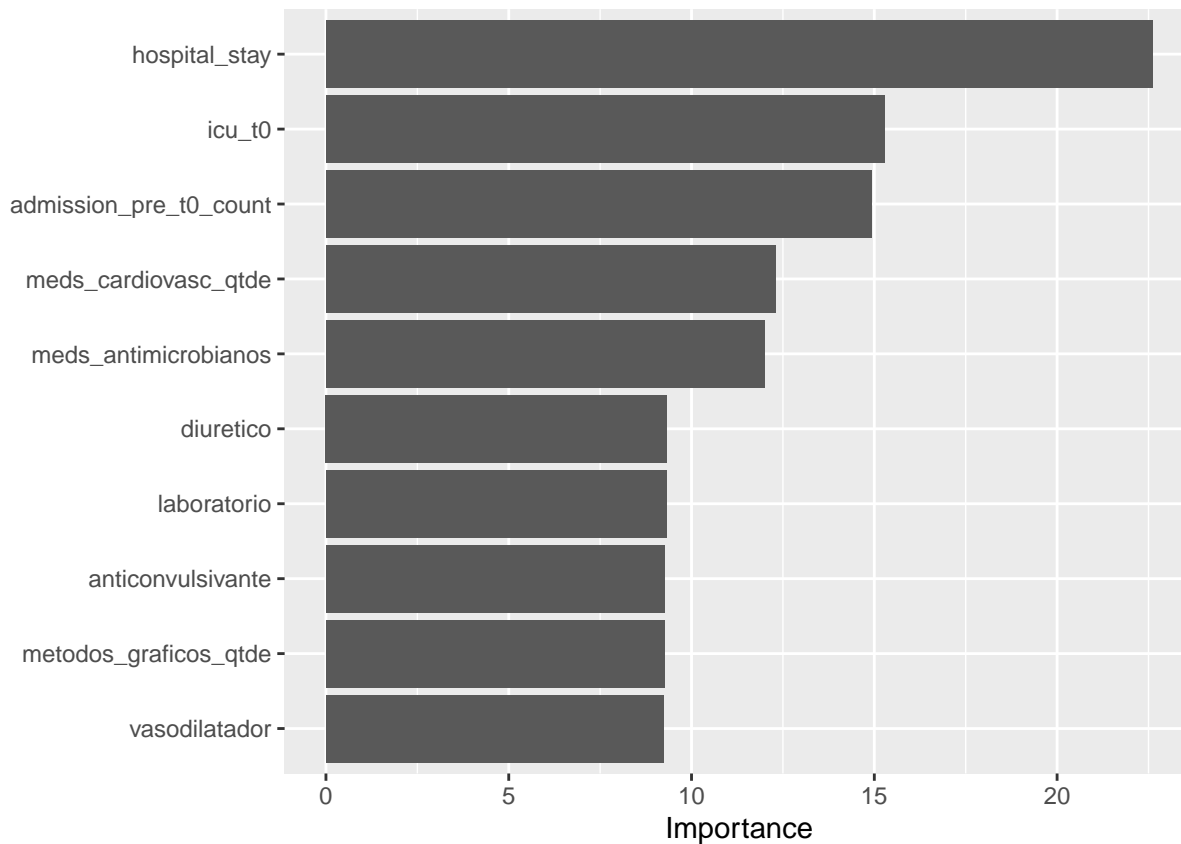
```
rf_auc <- validation(final_rf_fit, df_test)
```



```
## [1] "Optimal Threshold: 0.01"
## Confusion Matrix and Statistics
##
##      reference
## data    0    1
##    0 2782   64
##    1 1758  126
##
##              Accuracy : 0.6148
##              95% CI   : (0.6008, 0.6287)
##    No Information Rate : 0.9598
##    P-Value [Acc > NIR] : 1
##
##              Kappa   : 0.0523
##
##  McNemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.61278
##              Specificity : 0.66316
##              Pos Pred Value : 0.97751
##              Neg Pred Value : 0.06688
##              Prevalence : 0.95983
##              Detection Rate : 0.58816
##              Detection Prevalence : 0.60169
##              Balanced Accuracy : 0.63797
##
##              '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')
```



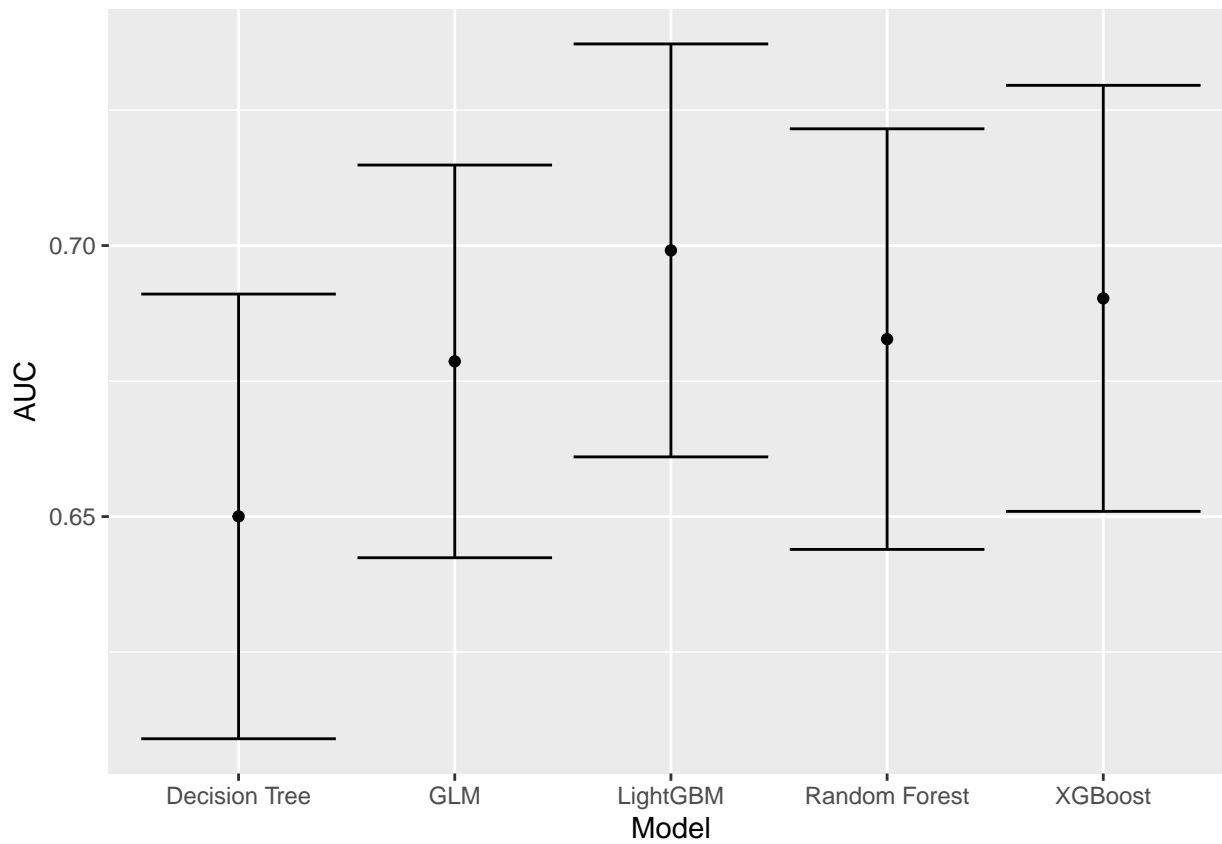
```
# extract_vip(final_rf_fit, pred_wrapper = predict,
#             reference_class = "1", use_matrix = FALSE,
#             method = 'permute')
```

Minutes to run: 71.58

Models Comparison

```
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()
```



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

Minutes to run: 0.002