Biostat 203B Homework 5

Due Mar 20 @ 11:59PM

AUTHOR
Zhiyuan Yu 906405523

Predicting ICU duration

Using the ICU cohort `mimiciv_icu_cohort.rds` you built in Homework 4, develop at least three machine learning approaches (logistic regression with enet regularization, random forest, boosting, SVM, MLP, etc) plus a model stacking approach for predicting whether a patient's ICU stay will be longer than 2 days. You should use the `los_long` variable as the outcome. You algorithms can use patient demographic information (gender, age at ICU `intime`, marital status, race), ICU admission information (first care unit), the last lab measurements before the ICU stay, and first vital measurements during ICU stay as features. You are welcome to use any feature engineering techniques you think are appropriate; but make sure to not use features that are not available at an ICU stay's `intime`. For instance, `last_careunit` cannot be used in your algorithms.

```r
library(gtsummary)
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.4     ✓ readr     2.1.5
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.5.1     ✓ tibble    3.2.1
✓ lubridate 1.9.4     ✓ tidyr     1.3.1
✓ purrr     1.0.4
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
errors
```

```r
library(tidymodels)
```

```
── Attaching packages ────────────────────────────────────── tidymodels 1.3.0 ──
✓ broom        1.0.7     ✓ rsample      1.2.1
✓ dials        1.4.0     ✓ tune         1.3.0
✓ infer        1.0.7     ✓ workflows    1.2.0
✓ modeldata    1.4.0     ✓ workflowsets 1.1.0
✓ parsnip      1.3.0     ✓ yardstick    1.3.2
✓ recipes      1.1.1
── Conflicts ───────────────────────────────────────── tidymodels_conflicts() ──
✗ scales::discard() masks purrr::discard()
✗ dplyr::filter()   masks stats::filter()
✗ recipes::fixed()  masks stringr::fixed()
✗ dplyr::lag()      masks stats::lag()
```

```
✖ yardstick::spec() masks readr::spec()
✖ recipes::step()   masks stats::step()
```

```r
library(dplyr)
library(haven)
library(recipes)
library(GGally)
```

```
Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2
```

```r
library(ranger)
library(xgboost)
```

```

Attaching package: 'xgboost'

The following object is masked from 'package:dplyr':

    slice
```

```r
library(stacks)
library(yardstick)
library(purrr)
library(vip)
```

```

Attaching package: 'vip'

The following object is masked from 'package:utils':

    vi
```

```r
library(parsnip)
library(tune)
library(dials)
library(purrr)
```

1. Data preprocessing and feature engineering.

Solution: I put the mimic_icu_cohort.rds file from HW4 to my current working directory of HW5. Step 1: Check for missing values

```r
mimic_icu_cohort <- read_rds("mimic_icu_cohort.rds") %>%
  filter(!is.na(los_long)) %>%
  select(-last_careunit, -dod, -discharge_location, -hospital_expire_flag, -los,
         -intime, -outtime_x, -admittime, -dischtime,
         -deathtime, -edregtime, -edouttime, -outtime_y, -admit_provider_id) %>%
  mutate(
    los_long = as.factor(los_long),
    insurance = as.factor(insurance),
    marital_status = as.factor(marital_status),
    language = as.factor(language)
  )

# Chech for missing values
colSums(is.na(mimic_icu_cohort))
```

```
                          subject_id                              hadm_id 
                                   0                                    0 
                             stay_id                       first_careunit 
                                   0                                    0 
                      admission_type                   admission_location 
                                   0                                    0 
                           insurance                             language 
                                1523                                  396 
                      marital_status                                 race 
                                7756                                    0 
                              gender                           anchor_age 
                                   0                                    0 
                         anchor_year                    anchor_year_group 
                                   0                                    0 
                         bicarbonate                             chloride 
                               11549                                11351 
                          creatinine                              glucose 
                                8027                                11654 
                           potassium                               sodium 
                               11387                                11330 
                          hematocrit                    white_blood_cells 
                                6751                                 6850 
                          heart_rate  systolic_non_invasive_blood_pressure 
                                  86                                 1370 
diastolic_non_invasive_blood_pressure               temperature_fahrenheit 
                                1375                                 1675 
                    respiratory_rate                           age_intime 
                                 198                                    0 
                            los_long 
                                   0 
```

Step 2: Take a peek at the data types

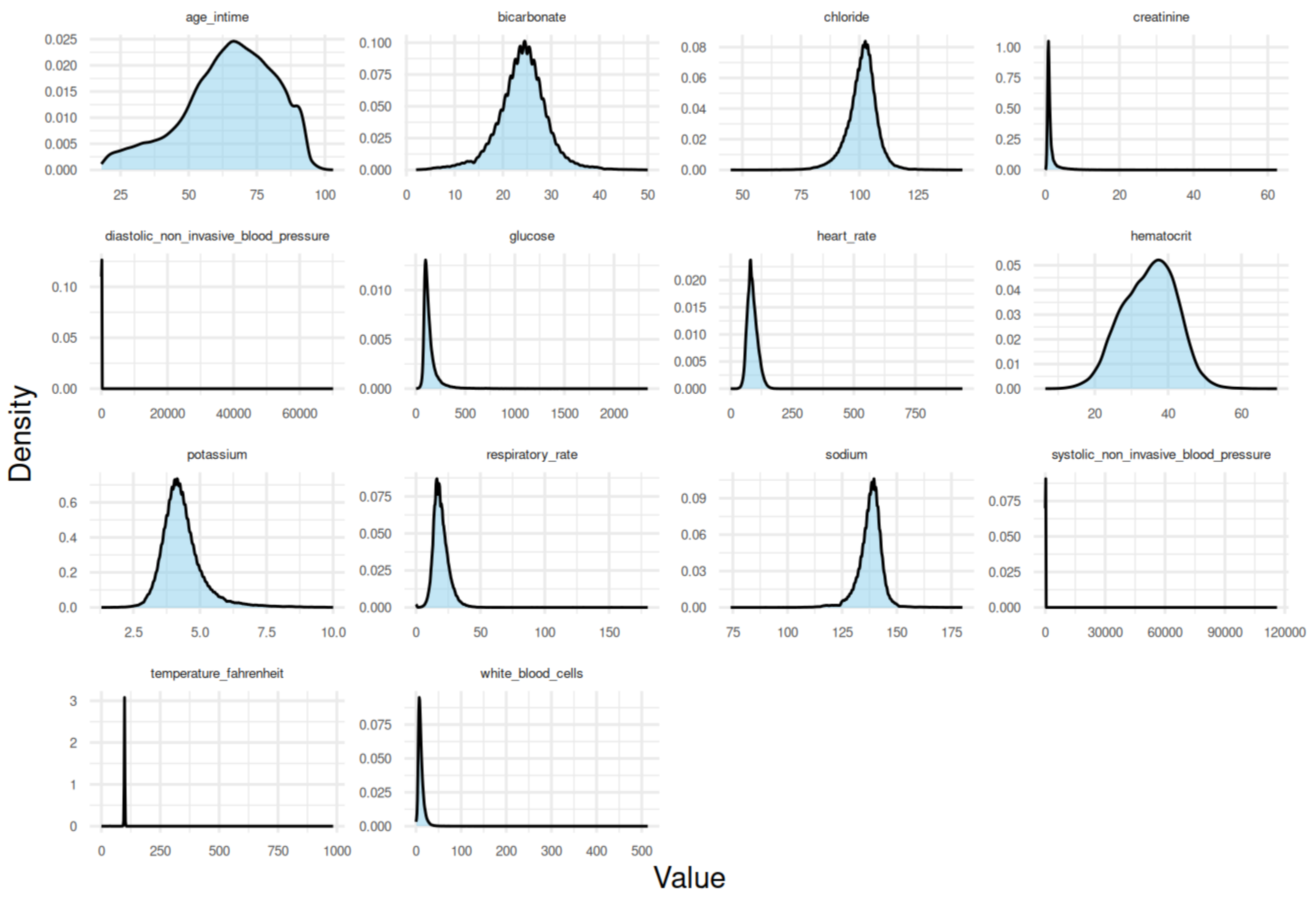
```
sapply(mimic_icu_cohort, class)
```

```
                          subject_id                              hadm_id 
                           "integer"                            "integer" 
                             stay_id                       first_careunit 
                           "integer"                             "factor" 
                      admission_type                   admission_location 
```

```
                             "factor"                              "factor" 
                            insurance                              language 
                             "factor"                              "factor" 
                       marital_status                                  race 
                             "factor"                              "factor" 
                               gender                            anchor_age 
                          "character"                             "integer" 
                          anchor_year                     anchor_year_group 
                            "integer"                           "character" 
                          bicarbonate                              chloride 
                            "numeric"                             "numeric" 
                           creatinine                               glucose 
                            "numeric"                             "numeric" 
                            potassium                                sodium 
                            "numeric"                             "numeric" 
                           hematocrit                     white_blood_cells 
                            "numeric"                             "numeric" 
                           heart_rate  systolic_non_invasive_blood_pressure 
                            "numeric"                             "numeric" 
diastolic_non_invasive_blood_pressure                temperature_fahrenheit 
                            "numeric"                             "numeric" 
                     respiratory_rate                            age_intime 
                            "numeric"                             "integer" 
                             los_long 
                             "factor" 
```

Step 3: Create histograms for the continuous variables to see their distributions

```r
# Define a vector with the names of the continuous variables
cont_vars <- c(
  "bicarbonate", "chloride", "creatinine", "glucose", "potassium", "sodium",
  "hematocrit", "white_blood_cells", "heart_rate",
  "systolic_non_invasive_blood_pressure",
  "diastolic_non_invasive_blood_pressure", "temperature_fahrenheit",
  "respiratory_rate", "age_intime"
)

# Reshape the dataset into long format
hist_data_alt <- mimic_icu_cohort %>%
  select(all_of(cont_vars)) %>%
  pivot_longer(
    cols = everything(),
    names_to = "variable",
    values_to = "value"
  )

# Plot the density distributions for each variable
ggplot(hist_data_alt, aes(x = value)) +
  geom_density(kernel = "cosine", fill = "skyblue", alpha = 0.5, na.rm = TRUE) +
  facet_wrap(~ variable, scales = "free") +
  labs(x = "Value", y = "Density") +
  theme_minimal() +
  theme(
    plot.margin = margin(12, 12, 12, 12, "pt"),
    strip.text = element_text(size = 5),
    axis.text.x = element_text(size = 5),
    axis.text.y = element_text(size = 5)
  )
```



It is clear that most of the data distributions are either left or right skewed in different levels, therefore I will use median instead of mean to impute the data.

2. Partition data into 50% training set and 50% test set. Stratify partitioning according to `los_long`. For grading purpose, sort the data by `subject_id`, `hadm_id`, and `stay_id` and use the seed `203` for the initial data split. Below is the sample code.

Solution: Step 1: split the data into training set and test set

```r
set.seed(203)

# sort the data by subject_id, hadm_id, and stay_id
mimic_icu_cohort <- mimic_icu_cohort |>
  arrange(subject_id, hadm_id, stay_id) |>
  select(-subject_id, -hadm_id, -stay_id)

# partition data into 50% training set and 50% test set
data_split <- initial_split(
  mimic_icu_cohort,
  strata = "los_long",
  prop = 0.5
  )

data_split
```

```
<Training/Testing/Total>
<47221/47223/94444>
```

```r
# check the training set
train_data <- training(data_split)
dim(train_data)
```

```
[1] 47221    26
```

```r
# check the training and testing set
test_data <- testing(data_split)
dim(test_data)
```

```
[1] 47223    26
```

Step 2: Preprocess the data

```r
recipe <- recipe(los_long ~ ., data = train_data) %>%
  step_impute_median(all_of(c("bicarbonate", "chloride", "creatinine",
                              "glucose", "potassium",
                              "sodium", "hematocrit", "white_blood_cells",
                              "heart_rate",
                              "systolic_non_invasive_blood_pressure",
                              "diastolic_non_invasive_blood_pressure",
                              "temperature_fahrenheit", "respiratory_rate",
                              "age_intime"))) %>%
  step_impute_mode(insurance, marital_status, language) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  print()
```

```
── Recipe ──────────────────────────────────────────────────────────────────────


── Inputs

Number of variables by role
```

```
outcome:    1
predictor: 25


── Operations

• Median imputation for: all_of(c("bicarbonate", "chloride", "creatinine",
  "glucose", "potassium", "sodium", "hematocrit", "white_blood_cells",
  "heart_rate", "systolic_non_invasive_blood_pressure",
  "diastolic_non_invasive_blood_pressure", "temperature_fahrenheit",
  "respiratory_rate", "age_intime"))

• Mode imputation for: insurance, marital_status, language

• Dummy variables from: all_nominal_predictors()

• Zero variance filter on: all_numeric_predictors()

• Centering and scaling for: all_numeric_predictors()
```

3. Train and tune the models using the training set.

First approach: Logistic regression with elastic net regularization

Step 1: Define recipe

```r
logit_recipe <- recipe
```

Step 1: Define the logistic regression model for classification and set the engine

```r
logit_model <- logistic_reg(
  penalty = tune(),
  mixture = tune()
) %>%
  set_engine("glmnet", standardize = FALSE)

print(logit_model)
```

```
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = tune()
  mixture = tune()

Engine-Specific Arguments:
  standardize = FALSE

Computational engine: glmnet
```

Step 2: Define workflow

```
logit_workflow <- workflow() %>%
  add_recipe(logit_recipe) %>%
  add_model(logit_model) %>%
  print()
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: logistic_reg()

── Preprocessor ────────────────────────────────────────────────────────────────
5 Recipe Steps

• step_impute_median()
• step_impute_mode()
• step_dummy()
• step_zv()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = tune()
  mixture = tune()

Engine-Specific Arguments:
  standardize = FALSE

Computational engine: glmnet
```

Step 3: Tune the grid and do cross-validation folds

```
logit_param_grid <- grid_regular(
  penalty(range = c(-4, 1)),
  mixture(),
  levels = c(100, 5)) %>%
  print()
```

```
# A tibble: 500 × 2
    penalty mixture
      <dbl>   <dbl>
 1 0.0001         0
 2 0.000112       0
 3 0.000126       0
 4 0.000142       0
 5 0.000159       0
 6 0.000179       0
 7 0.000201       0
 8 0.000226       0
 9 0.000254       0
10 0.000285       0
# ℹ 490 more rows
```

```
set.seed(203)

# define the number of folds for cross-validation is 5
logit_folds <- vfold_cv(train_data, v = 5)
logit_folds
```

```
#  5-fold cross-validation
# A tibble: 5 × 2
  splits               id   
  <list>               <chr>
1 <split [37776/9445]> Fold1
2 <split [37777/9444]> Fold2
3 <split [37777/9444]> Fold3
4 <split [37777/9444]> Fold4
5 <split [37777/9444]> Fold5
```

Step 4: Fit the corss-validated models and select the best model

```
suppressMessages(suppressWarnings({
  if (file.exists("logit_fit.rds")) {
    logit_fit <- read_rds("logit_fit.rds")
    logit_fit

  } else {
    (logit_fit <- logit_workflow |>
      tune_grid(
        resamples = logit_folds,
        grid = logit_param_grid,
        metrics = metric_set(roc_auc, accuracy)
      )) |>
    system.time()

    logit_fit |>
      write_rds("logit_fit.rds")

    logit_fit
  }
}))
```

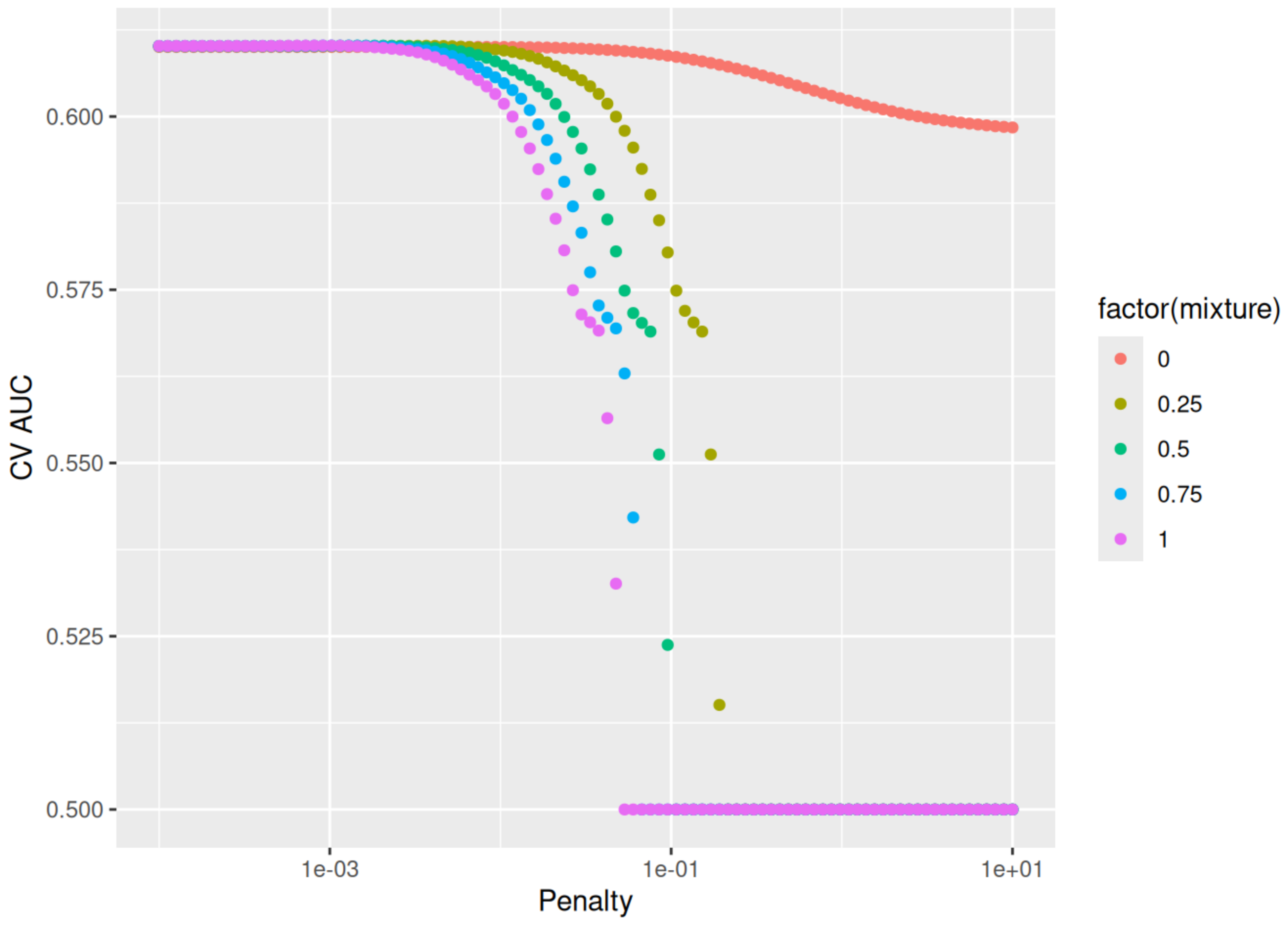
```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 × 5
  splits               id    .metrics             .notes           .predictions
  <list>               <chr> <list>               <list>           <list>
```

```
1 <split [37776/9445]> Fold1 <tibble [1,000 × 6]> <tibble [0 × 3]> <tibble>
2 <split [37777/9444]> Fold2 <tibble [1,000 × 6]> <tibble [0 × 3]> <tibble>
3 <split [37777/9444]> Fold3 <tibble [1,000 × 6]> <tibble [0 × 3]> <tibble>
4 <split [37777/9444]> Fold4 <tibble [1,000 × 6]> <tibble [0 × 3]> <tibble>
5 <split [37777/9444]> Fold5 <tibble [1,000 × 6]> <tibble [0 × 3]> <tibble>
```

Step 5: Visualize CV result

```
logit_fit |>
  # aggregate metrics from 5-fold cross-validation
  collect_metrics() |>
  print(width = Inf) |>
  filter(.metric == "roc_auc") |>
  ggplot(mapping = aes(x = penalty, y = mean,
                       color = factor(mixture))) +
  geom_point() +
  labs(x = "Penalty", y = "CV AUC") +
  scale_x_log10()
```

```
# A tibble: 1,000 × 8
    penalty mixture .metric  .estimator  mean     n std_err
      <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl>
 1 0.0001         0 accuracy binary     0.579     5 0.00200
 2 0.0001         0 roc_auc  binary     0.610     5 0.00149
 3 0.000112       0 accuracy binary     0.579     5 0.00200
 4 0.000112       0 roc_auc  binary     0.610     5 0.00149
 5 0.000126       0 accuracy binary     0.579     5 0.00200
 6 0.000126       0 roc_auc  binary     0.610     5 0.00149
 7 0.000142       0 accuracy binary     0.579     5 0.00200
 8 0.000142       0 roc_auc  binary     0.610     5 0.00149
 9 0.000159       0 accuracy binary     0.579     5 0.00200
10 0.000159       0 roc_auc  binary     0.610     5 0.00149
   .config
   <chr>
 1 Preprocessor1_Model001
 2 Preprocessor1_Model001
 3 Preprocessor1_Model002
 4 Preprocessor1_Model002
 5 Preprocessor1_Model003
 6 Preprocessor1_Model003
 7 Preprocessor1_Model004
 8 Preprocessor1_Model004
 9 Preprocessor1_Model005
10 Preprocessor1_Model005
# ℹ 990 more rows
```



Step 6: Show the top 5 models

```
logit_fit %>%
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 × 8
   penalty mixture .metric .estimator  mean     n std_err .config
     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1 0.000911       1 roc_auc binary     0.610     5 0.00137 Preprocessor1_Model420
2 0.000811       1 roc_auc binary     0.610     5 0.00139 Preprocessor1_Model419
3 0.00102        1 roc_auc binary     0.610     5 0.00135 Preprocessor1_Model421
4 0.000722       1 roc_auc binary     0.610     5 0.00141 Preprocessor1_Model418
5 0.000643       1 roc_auc binary     0.610     5 0.00143 Preprocessor1_Model417
```

Step 7: Select the best model

```
best_logit <- logit_fit |>
  select_best(metric = "roc_auc")
best_logit
```

```
# A tibble: 1 × 3
   penalty mixture .config
```

```
      <dbl>   <dbl> <chr>
1 0.000911       1 Preprocessor1_Model420
```

Step 8: Finalize model

```
# Final workflow
logit_final_workflow <- logit_workflow %>%
  finalize_workflow(best_logit)
logit_final_workflow
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: logistic_reg()

── Preprocessor ────────────────────────────────────────────────────────────────
5 Recipe Steps

• step_impute_median()
• step_impute_mode()
• step_dummy()
• step_zv()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = 0.000911162756115489
  mixture = 1

Engine-Specific Arguments:
  standardize = FALSE

Computational engine: glmnet
```

```
# Fit the whole traning set, then predict the test cases
logit_final_fit <- logit_final_workflow %>%
  last_fit(data_split)
```

```
New names:
New names:
• `anchor_year_group_X2011...2013` -> `anchor_year_group_X2011`
• `anchor_year_group_X2014...2016` -> `anchor_year_group_X2014`
• `anchor_year_group_X2017...2019` -> `anchor_year_group_X2017`
• `anchor_year_group_X2020...2022` -> `anchor_year_group_X2020`
```

```
logit_final_fit
```

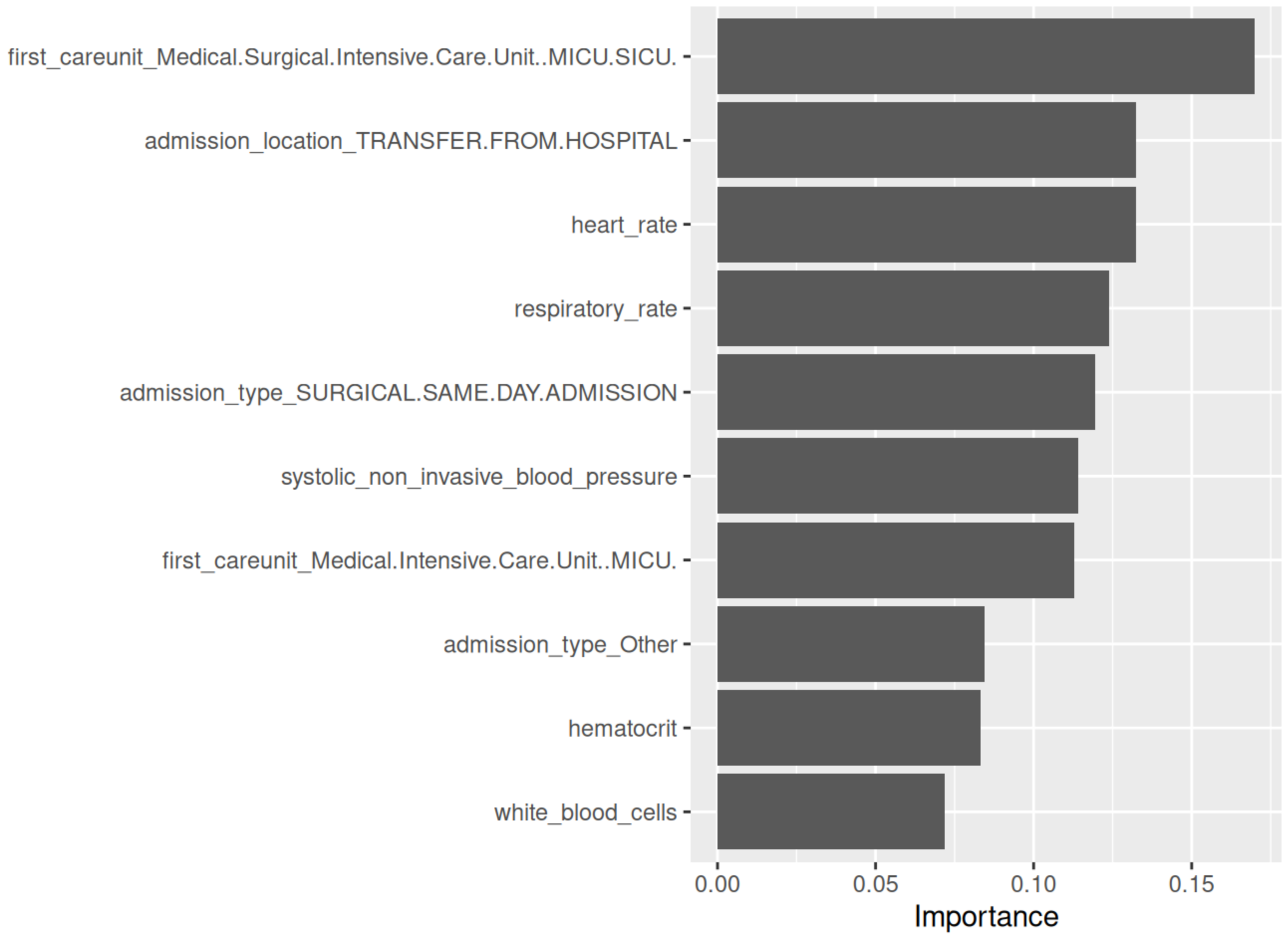
```
# Resampling results
# Manual resampling
# A tibble: 1 × 6
  splits                id              .metrics .notes   .predictions .workflow
  <list>                <chr>           <list>   <list>   <list>       <list>
1 <split [47221/47223]> train/test sp… <tibble> <tibble> <tibble>     <workflow>
```

```
# Test metrics
logit_final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 × 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.578 Preprocessor1_Model1
2 roc_auc     binary         0.608 Preprocessor1_Model1
3 brier_class binary         0.241 Preprocessor1_Model1
```

Step 9: Plot the variable importance

```
logit_final_fit %>%
  extract_fit_parsnip() %>%
  vip::vip() %>%
  print()
```



Summary of the logistic regression with elastic net regularization: Based on the final logistic regression model, its accuracy is 0.5782775, meaning that the model only has 57.8% rate of correct prediction on the test set. The model also has AUC of 0.60881425, indicating that the model can classify 60.9% of the test set correctly.

The 10 most important predictors are shown in the variable importance plot, and the most important predictor is the first care unit.

Second Approach: Random Forest

Step 1: Define recipe

```
# define the recipe
rf_recipe <- recipe
```

Step 2: Define Random Forest Model

```
rf_model <- rand_forest(
  mode = "classification",
  mtry = tune(),
  trees = tune()
) %>%
  set_engine("ranger", importance = "impurity")
rf_model
```

```
Random Forest Model Specification (classification)

Main Arguments:
  mtry = tune()
  trees = tune()

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

Step 3: Define workflow

```
rf_workflow <- workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_model) %>%
  print()
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: rand_forest()

── Preprocessor ────────────────────────────────────────────────────────────────
5 Recipe Steps
```

```
• step_impute_median()
• step_impute_mode()
• step_dummy()
• step_zv()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────
Random Forest Model Specification (classification)

Main Arguments:
  mtry = tune()
  trees = tune()

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

Step 4: Tuning grid

```r
rf_param_grid <- grid_regular(
  trees(range = c(100L, 500L)),
  mtry(range = c(2L, 5L)),
  levels = c(5, 5)
  )

rf_param_grid
```

```
# A tibble: 20 × 2
   trees  mtry
   <int> <int>
 1   100     2
 2   200     2
 3   300     2
 4   400     2
 5   500     2
 6   100     3
 7   200     3
 8   300     3
 9   400     3
10   500     3
11   100     4
12   200     4
13   300     4
14   400     4
15   500     4
16   100     5
17   200     5
18   300     5
19   400     5
20   500     5
```

Step 5: Cross-validation

```r
# Set cross-validation partitions
set.seed(203)

rf_folds <- vfold_cv(train_data, v = 5)
rf_folds
```

```
#  5-fold cross-validation
# A tibble: 5 × 2
  splits               id
  <list>               <chr>
1 <split [37776/9445]> Fold1
2 <split [37777/9444]> Fold2
3 <split [37777/9444]> Fold3
4 <split [37777/9444]> Fold4
5 <split [37777/9444]> Fold5
```

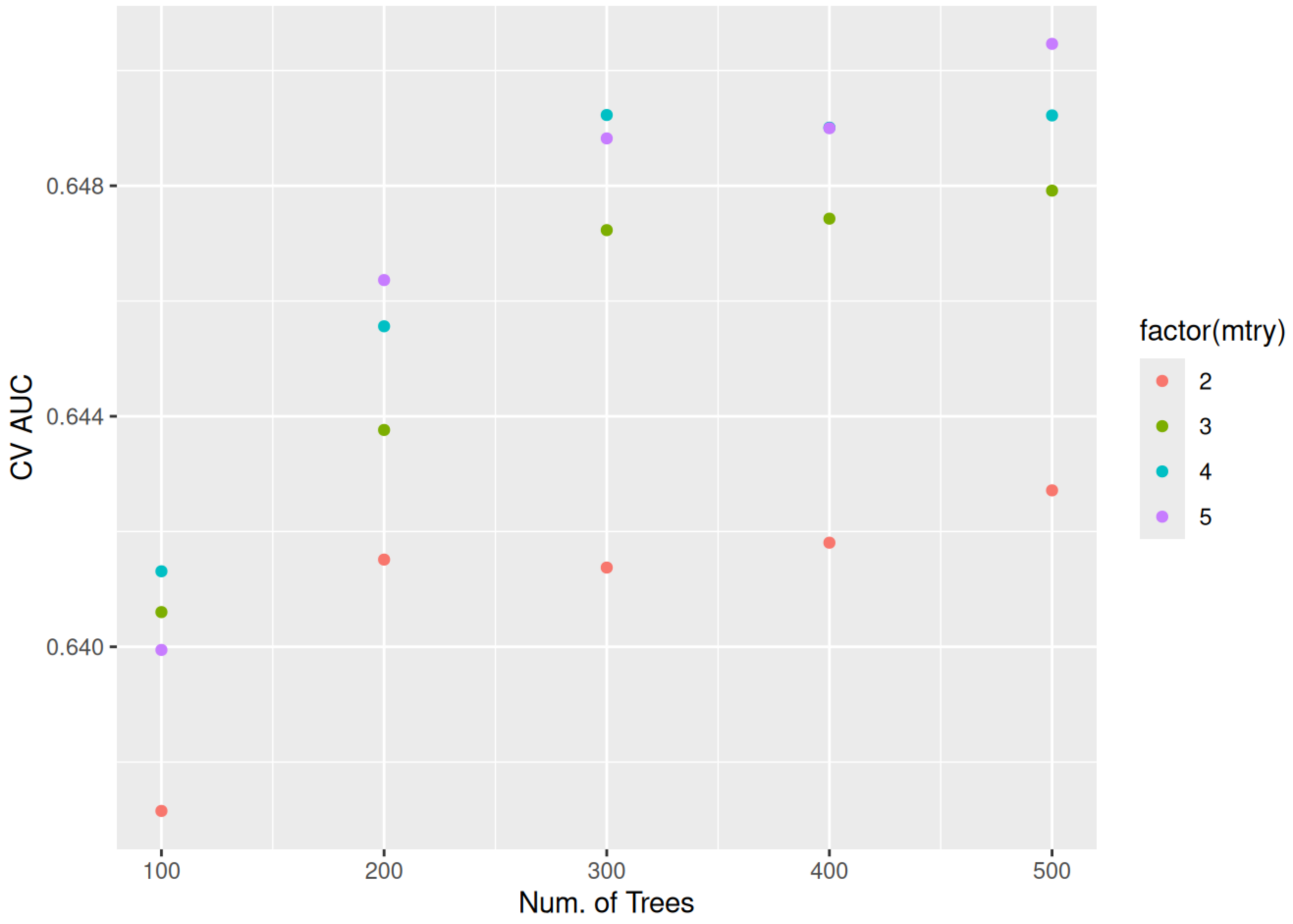
```r
# Fit cross-validation
if (file.exists("rf_fit.rds")) {
  rf_fit <- read_rds("rf_fit.rds")
  rf_fit

} else {
  (rf_fit <- rf_workflow %>%
    tune_grid(
      resamples = rf_folds,
      grid = rf_param_grid,
      metrics = metric_set(roc_auc, accuracy),
      control = control_stack_grid()
    )) %>%
  system.time()

  rf_fit %>%
    write_rds("rf_fit.rds")

  rf_fit
}
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 × 5
  splits               id    .metrics          .notes           .predictions
  <list>               <chr> <list>            <list>           <list>
1 <split [37776/9445]> Fold1 <tibble [40 × 6]> <tibble [0 × 3]> <tibble>
2 <split [37777/9444]> Fold2 <tibble [40 × 6]> <tibble [0 × 3]> <tibble>
3 <split [37777/9444]> Fold3 <tibble [40 × 6]> <tibble [0 × 3]> <tibble>
4 <split [37777/9444]> Fold4 <tibble [40 × 6]> <tibble [0 × 3]> <tibble>
5 <split [37777/9444]> Fold5 <tibble [40 × 6]> <tibble [0 × 3]> <tibble>
```

Step 6: Visualize CV results

```
rf_fit |>
  collect_metrics() %>%
  print(width = Inf) %>%
  filter(.metric == "roc_auc") %>%
  ggplot(mapping = aes(x = trees, y = mean,
                       color = factor(mtry))) +
  geom_point() +
  labs(x = "Num. of Trees", y = "CV AUC")
```

```
# A tibble: 40 × 8
    mtry trees .metric  .estimator  mean     n std_err .config
   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1     2   100 accuracy binary     0.597     5 0.00253 Preprocessor1_Model01
 2     2   100 roc_auc  binary     0.637     5 0.00280 Preprocessor1_Model01
 3     2   200 accuracy binary     0.602     5 0.00249 Preprocessor1_Model02
 4     2   200 roc_auc  binary     0.642     5 0.00235 Preprocessor1_Model02
 5     2   300 accuracy binary     0.599     5 0.00218 Preprocessor1_Model03
 6     2   300 roc_auc  binary     0.641     5 0.00195 Preprocessor1_Model03
 7     2   400 accuracy binary     0.600     5 0.00196 Preprocessor1_Model04
 8     2   400 roc_auc  binary     0.642     5 0.00220 Preprocessor1_Model04
 9     2   500 accuracy binary     0.601     5 0.00237 Preprocessor1_Model05
10     2   500 roc_auc  binary     0.643     5 0.00220 Preprocessor1_Model05
# ℹ 30 more rows
```



Step 7: Show the top 5 models

```
rf_fit |>
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 × 8
   mtry trees .metric .estimator  mean     n std_err .config
  <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1     5   500 roc_auc binary     0.650     5 0.00194 Preprocessor1_Model20
2     4   300 roc_auc binary     0.649     5 0.00158 Preprocessor1_Model13
3     4   500 roc_auc binary     0.649     5 0.00176 Preprocessor1_Model15
4     4   400 roc_auc binary     0.649     5 0.00194 Preprocessor1_Model14
5     5   400 roc_auc binary     0.649     5 0.00177 Preprocessor1_Model19
```

Step 8: Select the best model

```
best_rf <- rf_fit |>
  select_best(metric = "roc_auc")
best_rf
```

```
# A tibble: 1 × 3
   mtry trees .config
```

```
  <int> <int> <chr>
1     5   500 Preprocessor1_Model20
```

Step 9: Finalize the model

```
# Final workflow
rf_final_workflow <- rf_workflow %>%
  finalize_workflow(best_rf)
rf_final_workflow
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: rand_forest()

── Preprocessor ────────────────────────────────────────────────────────────────
5 Recipe Steps

• step_impute_median()
• step_impute_mode()
• step_dummy()
• step_zv()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────
Random Forest Model Specification (classification)

Main Arguments:
  mtry = 5
  trees = 500

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

```
# Fit the whole training set, then predict the test cases
if (file.exists("rf_final_fit.rds")) {
  rf_final_fit <- read_rds("rf_final_fit.rds")
  rf_final_fit

} else {

  # fit the final model on the whole training set
  rf_final_fit <- rf_final_workflow %>%
    last_fit(data_split)

  rf_final_fit %>%
    write_rds("rf_final_fit.rds")
  rf_final_fit
}
```

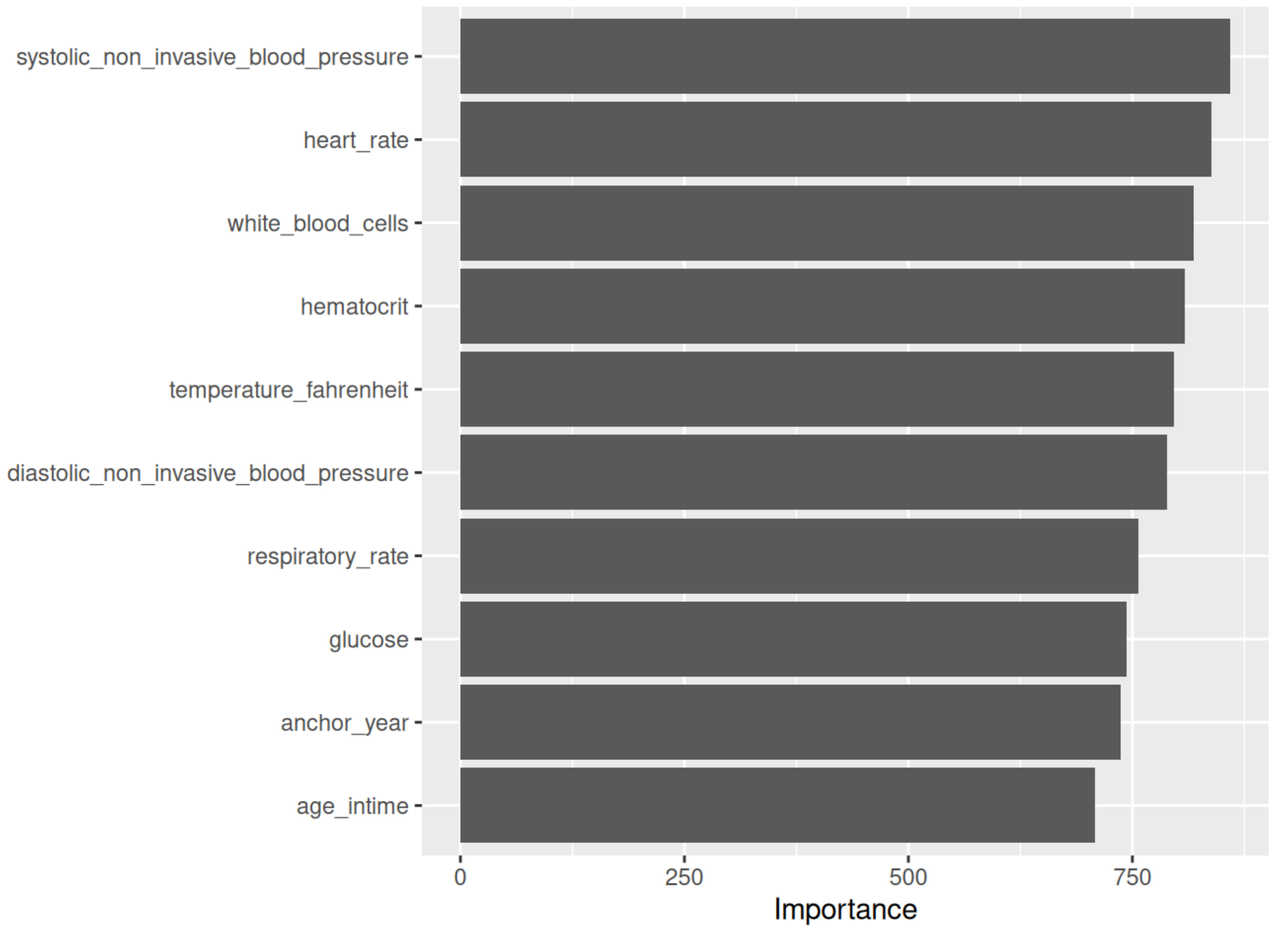
```
# Resampling results
# Manual resampling
# A tibble: 1 × 6
  splits                id              .metrics .notes   .predictions .workflow
  <list>                <chr>           <list>   <list>   <list>       <list>
1 <split [47221/47223]> train/test sp… <tibble> <tibble> <tibble>     <workflow>
```

```
# Test metrics
rf_final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 × 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.606 Preprocessor1_Model1
2 roc_auc     binary         0.647 Preprocessor1_Model1
3 brier_class binary         0.234 Preprocessor1_Model1
```

Step 10: Plot the variable importance

```
rf_final_fit %>%
  extract_fit_engine() %>%
  vip::vip() %>%
  print()
```



Summary of the random forest model: The model has accuracy of 0.6060818, meaning that the model can predict 60.6% of the test dataset correctly. The AUC of the model is 0.6474515, which means that the final random forest model can correctly classify 64.7% of the test dataset. At this point, we can see that the random forest model has better performance in both accuracy and AUC than the logistic model.

The 10 most important predictors are listed in the variable importance plot. The most important variable is systolic noninvasive blood pressure.

Third approach: Boosting (XGBoost)

Step 1: Define recipe

```
xgb_recipe <- recipe
```

Step 2: Define boosting model

```
xgb_model <- boost_tree(
  mode = "classification",
  trees = 1000,
  tree_depth = tune(),
  learn_rate = tune()
) %>%
  set_engine("xgboost")

xgb_model
```

```
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 1000
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

Step 3: Define workflow

```
xgb_workflow <- workflow() %>%
  add_recipe(xgb_recipe) %>%
  add_model(xgb_model)
xgb_workflow
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: boost_tree()

── Preprocessor ────────────────────────────────────────────────────────────────
5 Recipe Steps
```

```
• step_impute_median()
• step_impute_mode()
• step_dummy()
• step_zv()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 1000
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

Step 4: Tuning grid

```
xgb_param_grid <- grid_regular(
  tree_depth(range = c(1L, 3L)),
  learn_rate(range = c(-3, 0), trans = log10_trans()),
  levels = c(3, 3)
  )
xgb_param_grid
```

```
# A tibble: 9 × 2
  tree_depth learn_rate
       <int>      <dbl>
1          1     0.001 
2          2     0.001 
3          3     0.001 
4          1     0.0316
5          2     0.0316
6          3     0.0316
7          1     1     
8          2     1     
9          3     1     
```

Step 5: Cross-validation

```
# Set cross-validation partitions
set.seed(203)

xgb_folds <- vfold_cv(train_data, v = 5)
xgb_folds
```

```
#  5-fold cross-validation 
# A tibble: 5 × 2
  splits                 id   
  <list>                 <chr>
```

```
1 <split [37776/9445]> Fold1
2 <split [37777/9444]> Fold2
3 <split [37777/9444]> Fold3
4 <split [37777/9444]> Fold4
5 <split [37777/9444]> Fold5
```

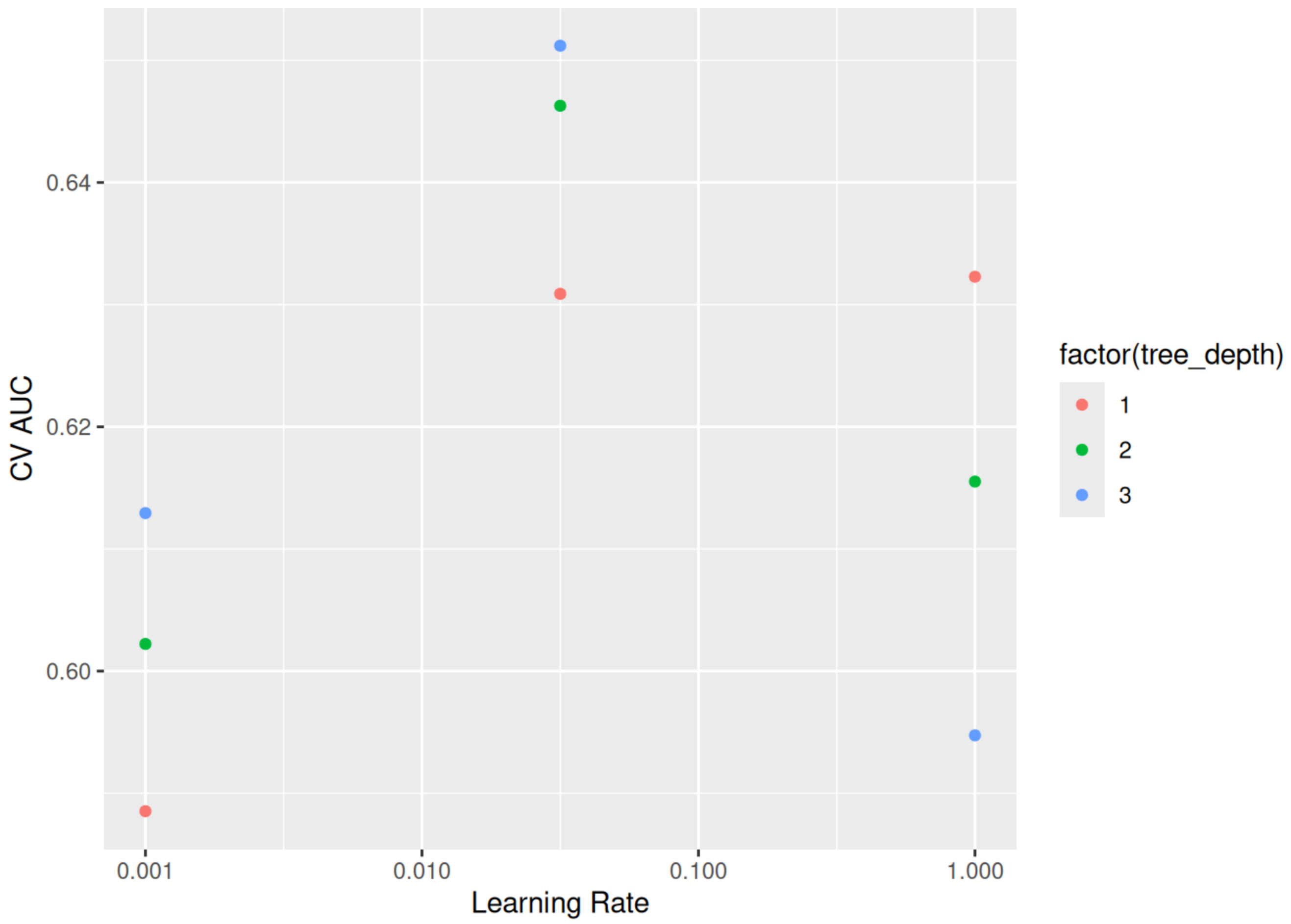
```
# Fit cross-validation
if (file.exists("xgb_fit.rds")) {
  xgb_fit <- read_rds("xgb_fit.rds")
  xgb_fit

} else {
  (xgb_fit <- xgb_workflow %>%
    tune_grid(
      resamples = xgb_folds,
      grid = xgb_param_grid,
      metrics = metric_set(roc_auc, accuracy),
      control = control_stack_grid()
      )) %>%
    system.time()

  xgb_fit %>%
    write_rds("xgb_fit.rds")

  xgb_fit
}
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 × 5
  splits               id    .metrics          .notes           .predictions
  <list>               <chr> <list>            <list>           <list>
1 <split [37776/9445]> Fold1 <tibble [18 × 6]> <tibble [0 × 3]> <tibble>
2 <split [37777/9444]> Fold2 <tibble [18 × 6]> <tibble [0 × 3]> <tibble>
3 <split [37777/9444]> Fold3 <tibble [18 × 6]> <tibble [0 × 3]> <tibble>
4 <split [37777/9444]> Fold4 <tibble [18 × 6]> <tibble [0 × 3]> <tibble>
5 <split [37777/9444]> Fold5 <tibble [18 × 6]> <tibble [0 × 3]> <tibble>
```

Step 6: Visualize CV results

```
xgb_fit |>
  collect_metrics() |>
  print(width = Inf) |>
  filter(.metric == "roc_auc") |>
  ggplot(mapping = aes(x = learn_rate, y = mean,
                       color = factor(tree_depth))) +
  geom_point() +
  labs(x = "Learning Rate", y = "CV AUC") +
  scale_x_log10()
```

```
# A tibble: 18 × 8
   tree_depth learn_rate .metric  .estimator  mean     n  std_err
        <int>      <dbl> <chr>    <chr>      <dbl> <int>    <dbl>
 1          1     0.001  accuracy binary     0.559     5 0.00245
 2          1     0.001  roc_auc  binary     0.589     5 0.00293
 3          2     0.001  accuracy binary     0.572     5 0.00289
 4          2     0.001  roc_auc  binary     0.602     5 0.00302
 5          3     0.001  accuracy binary     0.580     5 0.00236
 6          3     0.001  roc_auc  binary     0.613     5 0.00299
 7          1     0.0316 accuracy binary     0.592     5 0.00241
 8          1     0.0316 roc_auc  binary     0.631     5 0.00130
 9          2     0.0316 accuracy binary     0.605     5 0.000971
10          2     0.0316 roc_auc  binary     0.646     5 0.00115
11          3     0.0316 accuracy binary     0.608     5 0.000896
12          3     0.0316 roc_auc  binary     0.651     5 0.00139
13          1     1      accuracy binary     0.596     5 0.00197
14          1     1      roc_auc  binary     0.632     5 0.000888
15          2     1      accuracy binary     0.585     5 0.00166
16          2     1      roc_auc  binary     0.616     5 0.00202
17          3     1      accuracy binary     0.571     5 0.00148
18          3     1      roc_auc  binary     0.595     5 0.00129
   .config
   <chr>
 1 Preprocessor1_Model1
 2 Preprocessor1_Model1
 3 Preprocessor1_Model2
 4 Preprocessor1_Model2
 5 Preprocessor1_Model3
 6 Preprocessor1_Model3
 7 Preprocessor1_Model4
 8 Preprocessor1_Model4
 9 Preprocessor1_Model5
10 Preprocessor1_Model5
11 Preprocessor1_Model6
12 Preprocessor1_Model6
13 Preprocessor1_Model7
14 Preprocessor1_Model7
15 Preprocessor1_Model8
16 Preprocessor1_Model8
17 Preprocessor1_Model9
18 Preprocessor1_Model9
```



Step 7: Show the best 5 models

```
xgb_fit |>
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 × 8
  tree_depth learn_rate .metric .estimator  mean     n  std_err .config
       <int>      <dbl> <chr>   <chr>      <dbl> <int>    <dbl> <chr>
1          3     0.0316 roc_auc binary     0.651     5 0.00139  Preprocessor1_M…
2          2     0.0316 roc_auc binary     0.646     5 0.00115  Preprocessor1_M…
3          1     1      roc_auc binary     0.632     5 0.000888 Preprocessor1_M…
4          1     0.0316 roc_auc binary     0.631     5 0.00130  Preprocessor1_M…
5          2     1      roc_auc binary     0.616     5 0.00202  Preprocessor1_M…
```

Step 8: Select the best model

```
best_xgb <- xgb_fit |>
  select_best(metric = "roc_auc")
best_xgb
```

```
# A tibble: 1 × 3
  tree_depth learn_rate .config
```

```
      <int>      <dbl> <chr>
1         3     0.0316 Preprocessor1_Model6
```

Step 9: Finalize the model

```
# Final workflow
xgb_final_workflow <- xgb_workflow %>%
  finalize_workflow(best_xgb)
xgb_final_workflow
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: boost_tree()

── Preprocessor ────────────────────────────────────────────────────────────────
5 Recipe Steps

• step_impute_median()
• step_impute_mode()
• step_dummy()
• step_zv()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 1000
  tree_depth = 3
  learn_rate = 0.0316227766016838

Computational engine: xgboost
```

```
if (file.exists("xgb_final_fit.rds")) {
  xgb_final_fit <- read_rds("xgb_final_fit.rds")
  xgb_final_fit

} else {

  # fit the final model on the whole training set
  xgb_final_fit <- xgb_final_workflow %>%
    last_fit(data_split)

  xgb_final_fit %>%
    write_rds("xgb_final_fit.rds")

  xgb_final_fit
}
```

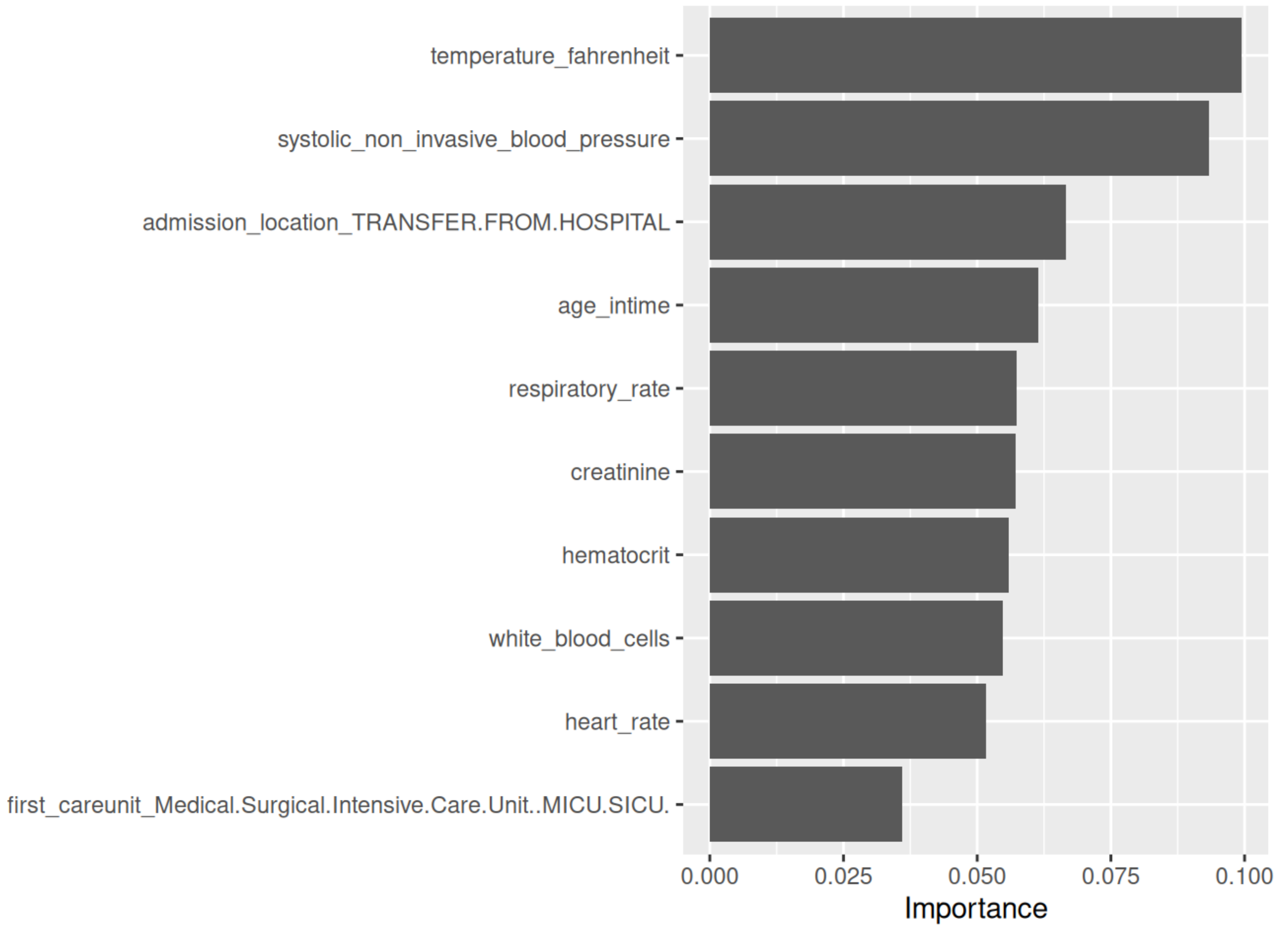
```
# Resampling results
# Manual resampling
# A tibble: 1 × 6
  splits                id              .metrics .notes   .predictions .workflow
  <list>                <chr>           <list>   <list>   <list>       <list>
1 <split [47221/47223]> train/test sp… <tibble> <tibble> <tibble>     <workflow>
```

```
# Test metrics
xgb_final_fit |>
  collect_metrics()
```

```
# A tibble: 3 × 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.604 Preprocessor1_Model1
2 roc_auc     binary         0.647 Preprocessor1_Model1
3 brier_class binary         0.233 Preprocessor1_Model1
```

Step 10: Plot the variable importance

```
xgb_final_fit %>%
  extract_fit_engine() %>%
  vip::vip() %>%
  print()
```



Summary of the XGBoosting model: The model has accuracy of 0.6036677, meaning that the model can predict 60.4% of the test dataset correctly. The AUC of the model is 0.6465419, which means that the final XGBoosting model can correctly classify 64.7% of the test dataset. At this point, we can see that both the random forest model and XGBoosting model have better performance in both accuracy and AUC than the logistic model. The performance of random forest model and XGBoosting model are close to each other, with random forest has slighlty higher accuracy and AUC.

The 10 most important predictors are listed in the variable importance plot. The most important variable in XGBoosting is temperature fahrenheit.

Model Stacking

Step 1: Set up the cross-validation folds to be shared by 3 models using in the stack model

```
set.seed(203)
folds <- vfold_cv(train_data, v = 3)
```

Step 2: Base models. Here we use logistic regression with elastic net regularization, Random Forest, and XGBoosting. To shorten the running time, I changed the grid of each model to smaller numbers to ensure the system won't crash.

```
# Logistic regression with elastic net regularization
logit_mod <-
  logistic_reg(
    penalty = tune(),
    mixture = tune()
  ) |>
  set_engine("glmnet", standardize = TRUE)
logit_mod
```

```
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = tune()
  mixture = tune()

Engine-Specific Arguments:
  standardize = TRUE

Computational engine: glmnet
```

```
logit_wf <- workflow() |>
  add_recipe(logit_recipe) |>
  add_model(logit_mod)
logit_wf
```

```
══ Workflow ══════════════════════════════════════════════════════════════════
Preprocessor: Recipe
```

```
Model: logistic_reg()

── Preprocessor ────────────────────────────────────────────────────────────────
5 Recipe Steps

• step_impute_median()
• step_impute_mode()
• step_dummy()
• step_zv()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = tune()
  mixture = tune()

Engine-Specific Arguments:
  standardize = TRUE

Computational engine: glmnet
```

```r
logit_stack_grid <- grid_regular(
  penalty(range = c(-6, 3)),
  mixture(),
  levels = c(5, 5)
  )

suppressMessages(suppressWarnings({
  if (file.exists("logit_stack.rds")) {
    logit_stack <- read_rds("logit_stack.rds")
    logit_stack

  } else {
    (logit_stack <- logit_wf |>
      tune_grid(
        resamples = folds,
        grid = logit_stack_grid,
        metrics = metric_set(roc_auc, accuracy),
        control = control_stack_grid()
      )) |>
    system.time()

    logit_stack |>
      write_rds("logit_stack.rds")

    logit_stack
  }
}))
```

```
# Tuning results
# 3-fold cross-validation
# A tibble: 3 × 5
  splits                id    .metrics          .notes           .predictions
  <list>                <chr> <list>            <list>           <list>
1 <split [31480/15741]> Fold1 <tibble [50 × 6]> <tibble [0 × 3]> <tibble>
2 <split [31481/15740]> Fold2 <tibble [50 × 6]> <tibble [0 × 3]> <tibble>
3 <split [31481/15740]> Fold3 <tibble [50 × 6]> <tibble [0 × 3]> <tibble>
```

```
# Random forest
rf_mod <-
  rand_forest(
    mode = "classification",
    mtry = tune(),
    trees = tune()
  ) |>
  set_engine("ranger")
rf_mod
```

```
Random Forest Model Specification (classification)

Main Arguments:
  mtry = tune()
  trees = tune()

Computational engine: ranger
```

```
rf_wf <- workflow() |>
  add_recipe(rf_recipe) |>
  add_model(rf_mod)
rf_wf
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: rand_forest()

── Preprocessor ────────────────────────────────────────────────────────────────
5 Recipe Steps

• step_impute_median()
• step_impute_mode()
• step_dummy()
• step_zv()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────
Random Forest Model Specification (classification)

Main Arguments:
  mtry = tune()
  trees = tune()

Computational engine: ranger
```

```
rf_stack_grid <- grid_regular(
  trees(range = c(200L, 500L)),
  mtry(range = c(1L, 5L)),
  levels = c(5, 2)
  )

if (file.exists("rf_stack.rds")) {
  rf_stack <- read_rds("rf_stack.rds")
  rf_stack

} else {
  (rf_stack <- rf_wf %>%
    tune_grid(
      resamples = folds,
      grid = rf_stack_grid,
      metrics = metric_set(roc_auc, accuracy),
      control = control_stack_grid()
    )) %>%
  system.time()

  rf_stack %>%
    write_rds("rf_stack.rds")

  rf_stack
}
```

```
# Tuning results
# 3-fold cross-validation
# A tibble: 3 × 5
  splits                id    .metrics          .notes           .predictions
  <list>                <chr> <list>            <list>           <list>
1 <split [31480/15741]> Fold1 <tibble [20 × 6]> <tibble [0 × 3]> <tibble>
2 <split [31481/15740]> Fold2 <tibble [20 × 6]> <tibble [0 × 3]> <tibble>
3 <split [31481/15740]> Fold3 <tibble [20 × 6]> <tibble [0 × 3]> <tibble>
```

```
# XGBoosting
gb_mod <-
  boost_tree(
    mode = "classification",
    trees = 1000,
    tree_depth = tune(),
    learn_rate = tune()
  ) |>
```

```r
  set_engine("xgboost")
gb_mod
```

```
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 1000
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

```r
gb_wf <- workflow() |>
  add_recipe(xgb_recipe) |>
  add_model(gb_mod)
gb_wf
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: boost_tree()

── Preprocessor ────────────────────────────────────────────────────────────────
5 Recipe Steps

• step_impute_median()
• step_impute_mode()
• step_dummy()
• step_zv()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 1000
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

```r
gb_stack_grid <- grid_regular(
  tree_depth(range = c(1L, 3L)),
  learn_rate(range = c(-3, 1), trans = log10_trans()),
  levels = c(3, 3)
  )
gb_stack_grid
```

```
# A tibble: 9 × 2
  tree_depth learn_rate
```

```
         <int>      <dbl>
1            1      0.001
2            2      0.001
3            3      0.001
4            1      0.1
5            2      0.1
6            3      0.1
7            1     10
8            2     10
9            3     10
```

```
if (file.exists("gb_stack.rds")) {
  gb_stack <- read_rds("gb_stack.rds")
  gb_stack

} else {
  (gb_stack <- gb_wf %>%
    tune_grid(
      resamples = folds,
      grid = gb_stack_grid,
      metrics = metric_set(roc_auc, accuracy),
      control = control_stack_grid()
      )) %>%
    system.time()

  gb_stack %>%
    write_rds("gb_stack.rds")

  gb_stack
}
```

```
# Tuning results
# 3-fold cross-validation
# A tibble: 3 × 5
  splits                id    .metrics          .notes           .predictions
  <list>                <chr> <list>            <list>           <list>
1 <split [31480/15741]> Fold1 <tibble [18 × 6]> <tibble [0 × 3]> <tibble>
2 <split [31481/15740]> Fold2 <tibble [18 × 6]> <tibble [0 × 3]> <tibble>
3 <split [31481/15740]> Fold3 <tibble [18 × 6]> <tibble [0 × 3]> <tibble>
```

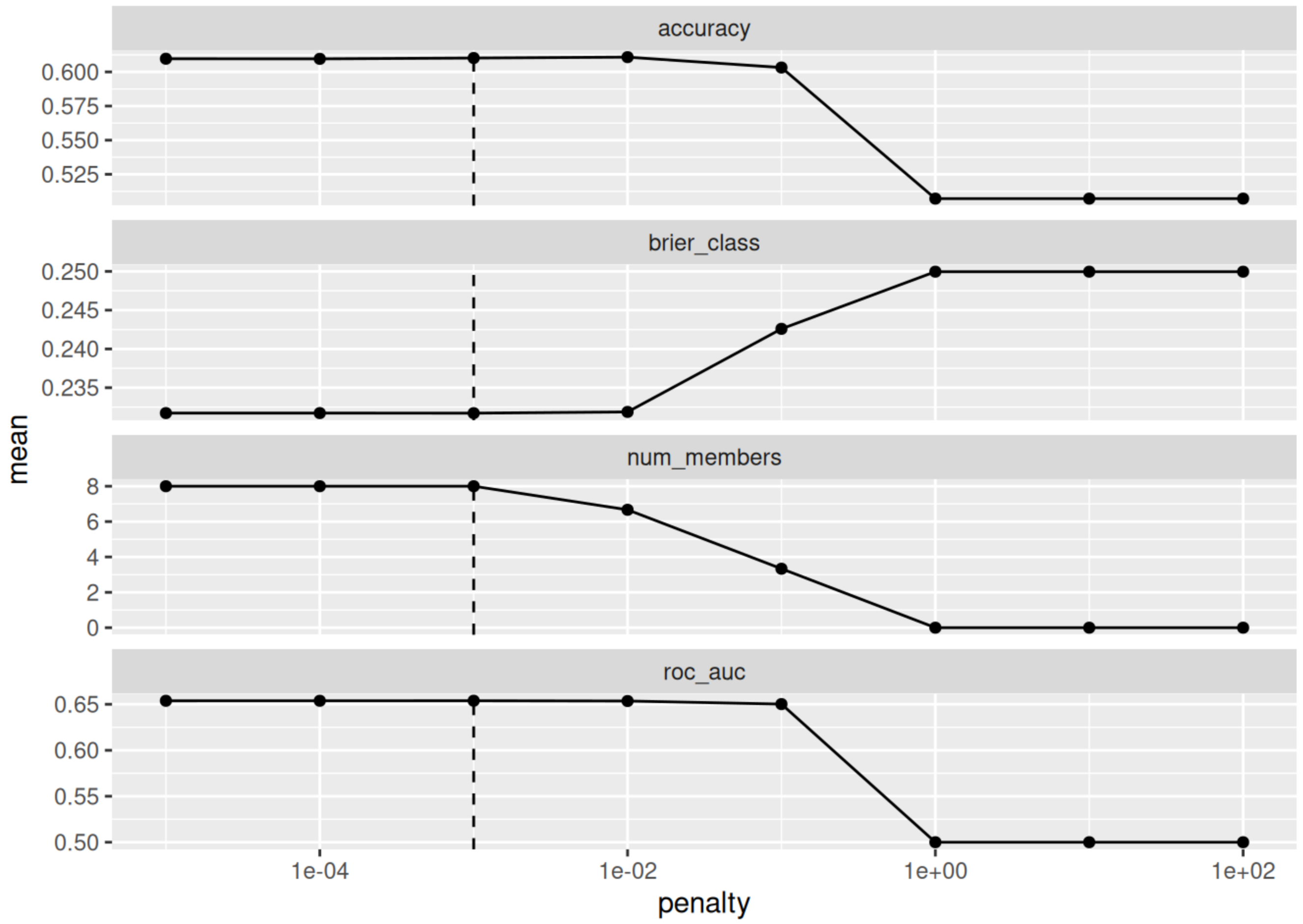
Step 3: Build the stacked ensemble

```
if (file.exists("stacks.rds")) {
  model_st <- read_rds("stacks.rds")
  model_st
} else {
  suppressWarnings({
  model_st <-
    stacks() |>

    # add candidate models
    add_candidates(logit_stack) |>
    add_candidates(rf_stack) |>
    add_candidates(gb_stack) |>

    # determine how to combine their predictions
    blend_predictions(
      penalty = 10^(-5:2),
      metrics = c("roc_auc"),

      # set the number of resamples to 3 to reduce computation time
      times = 3) |>

    # fit the candidates with nonzero stacking coefficients
    fit_members()

  model_st |> write_rds("stacks.rds")

  model_st
})
}
```

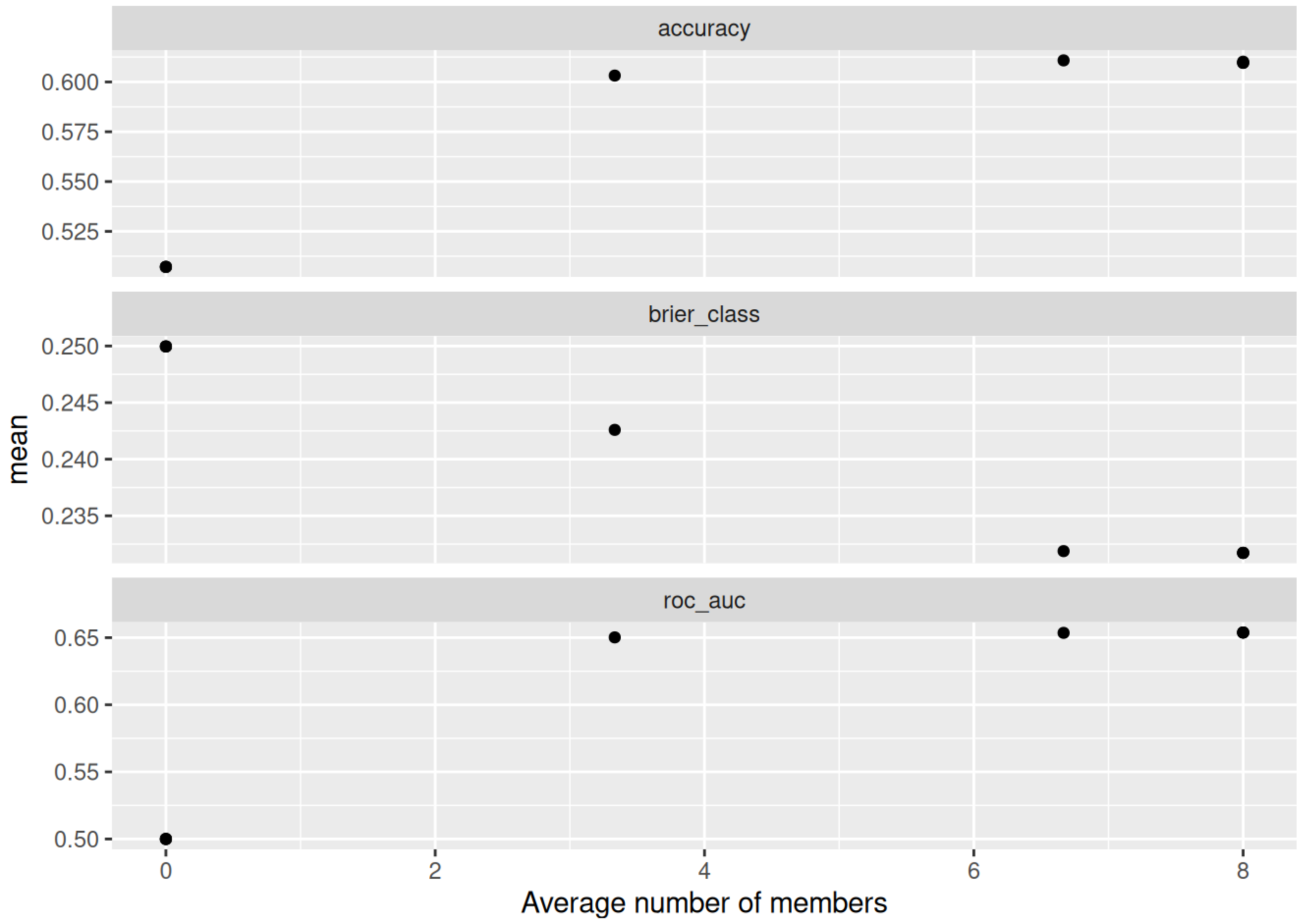
```
── A stacked ensemble model ─────────────────────────────────────


Out of 31 possible candidate members, the ensemble retained 9.

Penalty: 0.001.

Mixture: 1.


The 9 highest weighted member classes are:

# A tibble: 9 × 3
  member                    type        weight
  <chr>                     <chr>        <dbl>
1 .pred_TRUE_rf_stack_1_09 rand_forest 1.26
2 .pred_TRUE_rf_stack_1_06 rand_forest 1.01
3 .pred_TRUE_gb_stack_1_6  boost_tree  0.984
4 .pred_TRUE_rf_stack_1_07 rand_forest 0.816
5 .pred_TRUE_gb_stack_1_5  boost_tree  0.702
6 .pred_TRUE_rf_stack_1_10 rand_forest 0.585
7 .pred_TRUE_rf_stack_1_08 rand_forest 0.566
8 .pred_TRUE_gb_stack_1_8  boost_tree  0.0722
9 .pred_TRUE_gb_stack_1_7  boost_tree  0.0229
```

Step 4: Plot the stacked results

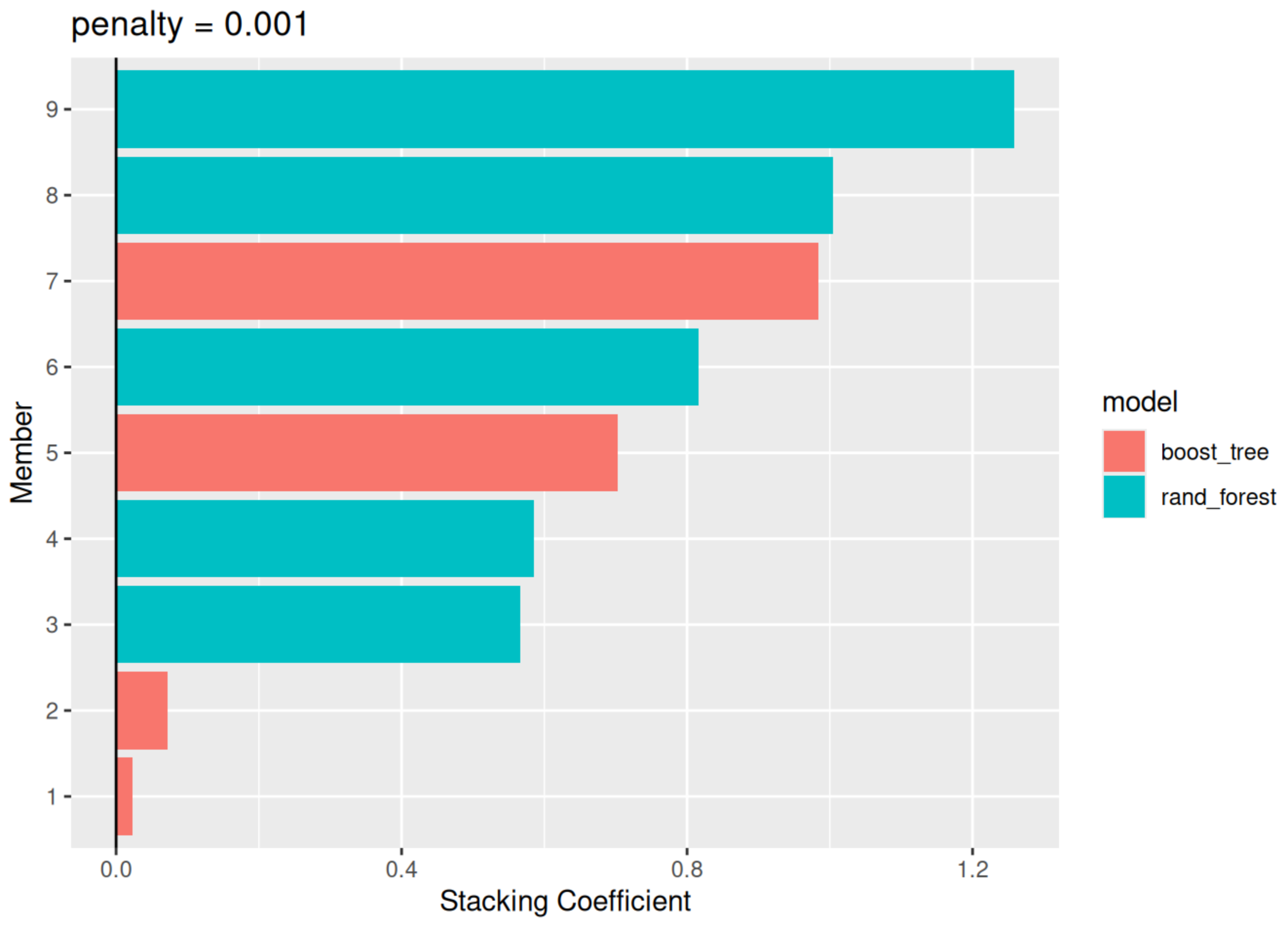
```
autoplot(model_st)
```



```r
# To show the relationship more directly
autoplot(model_st, type = "members")
```



```
# To see the top models
autoplot(model_st, type = "weights")
```



Step 5: To identify which model configurations were assigned what stacking coefficients

```
# Coefficients of random forest model
collect_parameters(model_st, "rf_stack")
```

```
# A tibble: 10 × 5
   member         mtry trees terms                     coef
   <chr>         <int> <int> <chr>                    <dbl>
 1 rf_stack_1_01     1   200 .pred_TRUE_rf_stack_1_01 0
 2 rf_stack_1_02     1   275 .pred_TRUE_rf_stack_1_02 0
 3 rf_stack_1_03     1   350 .pred_TRUE_rf_stack_1_03 0
 4 rf_stack_1_04     1   425 .pred_TRUE_rf_stack_1_04 0
 5 rf_stack_1_05     1   500 .pred_TRUE_rf_stack_1_05 0
 6 rf_stack_1_06     5   200 .pred_TRUE_rf_stack_1_06 1.01
 7 rf_stack_1_07     5   275 .pred_TRUE_rf_stack_1_07 0.816
 8 rf_stack_1_08     5   350 .pred_TRUE_rf_stack_1_08 0.566
 9 rf_stack_1_09     5   425 .pred_TRUE_rf_stack_1_09 1.26
10 rf_stack_1_10     5   500 .pred_TRUE_rf_stack_1_10 0.585
```

```
# Coefficients of XGBoosting model
collect_parameters(model_st, "gb_stack")
```

```
# A tibble: 9 × 5
  member       tree_depth learn_rate terms                      coef
  <chr>             <int>      <dbl> <chr>                     <dbl>
1 gb_stack_1_1          1      0.001 .pred_TRUE_gb_stack_1_1 0
2 gb_stack_1_2          2      0.001 .pred_TRUE_gb_stack_1_2 0
3 gb_stack_1_3          3      0.001 .pred_TRUE_gb_stack_1_3 0
4 gb_stack_1_4          1      0.1   .pred_TRUE_gb_stack_1_4 0
5 gb_stack_1_5          2      0.1   .pred_TRUE_gb_stack_1_5 0.702
6 gb_stack_1_6          3      0.1   .pred_TRUE_gb_stack_1_6 0.984
7 gb_stack_1_7          1     10     .pred_TRUE_gb_stack_1_7 0.0229
8 gb_stack_1_8          2     10     .pred_TRUE_gb_stack_1_8 0.0722
9 gb_stack_1_9          3     10     .pred_TRUE_gb_stack_1_9 0
```

```r
# Coefficients of Logistic regression model
collect_parameters(model_st, "logit_stack")
```

```
# A tibble: 12 × 5
   member             penalty mixture terms                        coef
   <chr>                <dbl>   <dbl> <chr>                       <dbl>
 1 logit_stack_1_01 0.000001    0    .pred_TRUE_logit_stack_1_01     0
 2 logit_stack_1_03 0.0316      0    .pred_TRUE_logit_stack_1_03     0
 3 logit_stack_1_04 5.62        0    .pred_TRUE_logit_stack_1_04     0
 4 logit_stack_1_06 0.000001    0.25 .pred_TRUE_logit_stack_1_06     0
 5 logit_stack_1_08 0.0316      0.25 .pred_TRUE_logit_stack_1_08     0
 6 logit_stack_1_09 5.62        0.25 .pred_TRUE_logit_stack_1_09     0
 7 logit_stack_1_11 0.000001    0.5  .pred_TRUE_logit_stack_1_11     0
 8 logit_stack_1_13 0.0316      0.5  .pred_TRUE_logit_stack_1_13     0
 9 logit_stack_1_16 0.000001    0.75 .pred_TRUE_logit_stack_1_16     0
10 logit_stack_1_18 0.0316      0.75 .pred_TRUE_logit_stack_1_18     0
11 logit_stack_1_21 0.000001    1    .pred_TRUE_logit_stack_1_21     0
12 logit_stack_1_23 0.0316      1    .pred_TRUE_logit_stack_1_23     0
```

Step 6: Finalization Apply the model on the test data and output the final classification

```r
if (file.exists("final_classification.rds")) {
  final_classification <- read_rds("final_classification.rds")

  final_classification

} else {
  final_classification <- test_data |>
    bind_cols(predict(model_st, test_data, type = "prob")) |>
    print(width = Inf)

  final_classification

  final_classification |>
    write_rds("final_classification.rds")
}
```

```
# A tibble: 47,223 × 28
   first_careunit           admission_type admission_location insurance language
   <fct>                    <fct>          <fct>              <fct>     <fct>
 1 Medical Intensive Care … EW EMER.       EMERGENCY ROOM     Medicaid  English
 2 Medical/Surgical Intens… EW EMER.       Other              Private   English
 3 Cardiac Vascular Intens… SURGICAL SAME… PHYSICIAN REFERRAL Medicare  English
 4 Coronary Care Unit (CCU) OBSERVATION A… PHYSICIAN REFERRAL Medicaid  English
 5 Medical Intensive Care … EW EMER.       EMERGENCY ROOM     Medicare  English
 6 Medical/Surgical Intens… EW EMER.       EMERGENCY ROOM     Medicare  English
 7 Medical Intensive Care … EW EMER.       EMERGENCY ROOM     Medicare  English
 8 Coronary Care Unit (CCU) URGENT         TRANSFER FROM HOS… Medicare  English
 9 Coronary Care Unit (CCU) EW EMER.       TRANSFER FROM HOS… Private   English
10 Cardiac Vascular Intens… OBSERVATION A… PHYSICIAN REFERRAL Private   English
# ℹ 47,213 more rows
# ℹ 23 more variables: marital_status <fct>, race <fct>, gender <chr>,
#   anchor_age <int>, anchor_year <int>, anchor_year_group <chr>,
#   bicarbonate <dbl>, chloride <dbl>, creatinine <dbl>, glucose <dbl>,
#   potassium <dbl>, sodium <dbl>, hematocrit <dbl>, white_blood_cells <dbl>,
#   heart_rate <dbl>, systolic_non_invasive_blood_pressure <dbl>,
#   diastolic_non_invasive_blood_pressure <dbl>, …
```

Compute the ROC AUC and accuracy of the final classification

```r
# ROC AUC
yardstick::roc_auc(
  final_classification,
  truth = los_long,
  contains(".pred_FALSE")
  )
```

```
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.653
```

```r
# Accuracy
final_classification <- final_classification %>%
  mutate(.pred_class = as.factor(
    ifelse(.pred_TRUE > .pred_FALSE, "TRUE", "FALSE")))

yardstick::accuracy(
  final_classification,
  truth = los_long,
  estimate = .pred_class
  )
```

```
# A tibble: 1 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.609
```

Use the members argument to generate predictions from each of the ensemble members

```
if (file.exists("mimic_pred.rds")) {
  mimic_pred <- read_rds("mimic_pred.rds")
} else {
  mimic_pred <-
    test_data |>
    select(los_long) |>
    bind_cols(
      predict(
        model_st,
        test_data,
        type = "class",
        members = TRUE
      )
    ) |>
    print(width = Inf)

  write_rds(mimic_pred, "mimic_pred.rds")
}

mimic_pred
```

```
# A tibble: 47,223 × 11
   los_long .pred_class .pred_class_rf_stack_1_06 .pred_class_rf_stack_1_07
   <fct>    <fct>       <fct>                     <fct>
 1 FALSE    TRUE        TRUE                      TRUE
 2 FALSE    FALSE       FALSE                     FALSE
 3 FALSE    FALSE       FALSE                     FALSE
 4 TRUE     TRUE        TRUE                      TRUE
 5 FALSE    TRUE        TRUE                      TRUE
 6 TRUE     TRUE        TRUE                      TRUE
 7 TRUE     FALSE       FALSE                     FALSE
 8 TRUE     TRUE        TRUE                      TRUE
 9 TRUE     FALSE       FALSE                     TRUE
10 FALSE    TRUE        TRUE                      TRUE
# ℹ 47,213 more rows
# ℹ 7 more variables: .pred_class_rf_stack_1_08 <fct>,
#   .pred_class_rf_stack_1_09 <fct>, .pred_class_rf_stack_1_10 <fct>,
#   .pred_class_gb_stack_1_7 <fct>, .pred_class_gb_stack_1_5 <fct>,
#   .pred_class_gb_stack_1_8 <fct>, .pred_class_gb_stack_1_6 <fct>
```

```
# Get the mean of the predicted classes for each model
map(
  colnames(mimic_pred),
  ~mean(mimic_pred$los_long == pull(mimic_pred, .x))
  ) |>
  set_names(colnames(mimic_pred)) |>
  as_tibble() |>
  pivot_longer(c(everything(), -los_long))
```

```
# A tibble: 10 × 3
   los_long name                        value
      <dbl> <chr>                       <dbl>
 1        1 .pred_class                 0.609
 2        1 .pred_class_rf_stack_1_06 0.603
 3        1 .pred_class_rf_stack_1_07 0.604
 4        1 .pred_class_rf_stack_1_08 0.606
 5        1 .pred_class_rf_stack_1_09 0.604
 6        1 .pred_class_rf_stack_1_10 0.605
 7        1 .pred_class_gb_stack_1_7  0.454
 8        1 .pred_class_gb_stack_1_5  0.603
 9        1 .pred_class_gb_stack_1_8  0.448
10        1 .pred_class_gb_stack_1_6  0.605
```

4. Compare model classification performance on the test set. Report both the area under ROC curve and accuracy for each machine learning algorithm and the model stacking. Interpret the results. What are the most important features in predicting long ICU stays? How do the models compare in terms of performance and interpretability?

Report the information of accuracy and AUC of each machine learning algorithm and the model stacking

```
# Logistic regression with elastic net regularization
logit_final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 × 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.578 Preprocessor1_Model1
2 roc_auc     binary         0.608 Preprocessor1_Model1
3 brier_class binary         0.241 Preprocessor1_Model1
```

```
# Random forest
rf_final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 × 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.606 Preprocessor1_Model1
2 roc_auc     binary         0.647 Preprocessor1_Model1
3 brier_class binary         0.234 Preprocessor1_Model1
```

```r
# XGBoosting
xgb_final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 × 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.604 Preprocessor1_Model1
2 roc_auc     binary         0.647 Preprocessor1_Model1
3 brier_class binary         0.233 Preprocessor1_Model1
```

```r
# Model stacking
## Accuracy
final_classification <- final_classification %>%
  mutate(.pred_class = as.factor(
    ifelse(.pred_TRUE > .pred_FALSE, "TRUE", "FALSE")))

yardstick::accuracy(
  final_classification,
  truth = los_long,
  estimate = .pred_class
  )
```

```
# A tibble: 1 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.609
```

```r
## ROC AUC
yardstick::roc_auc(
  final_classification,
  truth = los_long,
  contains(".pred_FALSE")
  )
```

```
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.653
```

It can be seen that the accuracy and AUC of the logistic regression model are 0.5782775 and 0.6081425 respectively, the accuracy and AUC of the random forest model are 0.6060818 and 0.6474515 respectively, and the accuracy and AUC of the XGBoosting model are 0.6036677 and 0.6465419 respectively. Based on the above results, we can see that for individual models, random forest has the highest accuracy and AUC and thus it is the best-performing model among the three machine learning algorithms, and it has the greatest weight in the stacked ensemble, followed by the XGBoosting model. In order to speed up the processing and prevent the system from crashing, I selected a smaller fold of cross-validation (3) than the ones for single models (5), and I also reduced the grids. Because of this reason, there are some duplicated candidates that are automatically removed by the system, and the stacked results shows no information on logistic model. But this should not be a problem since the logistic model is the least efficient one among the three models.

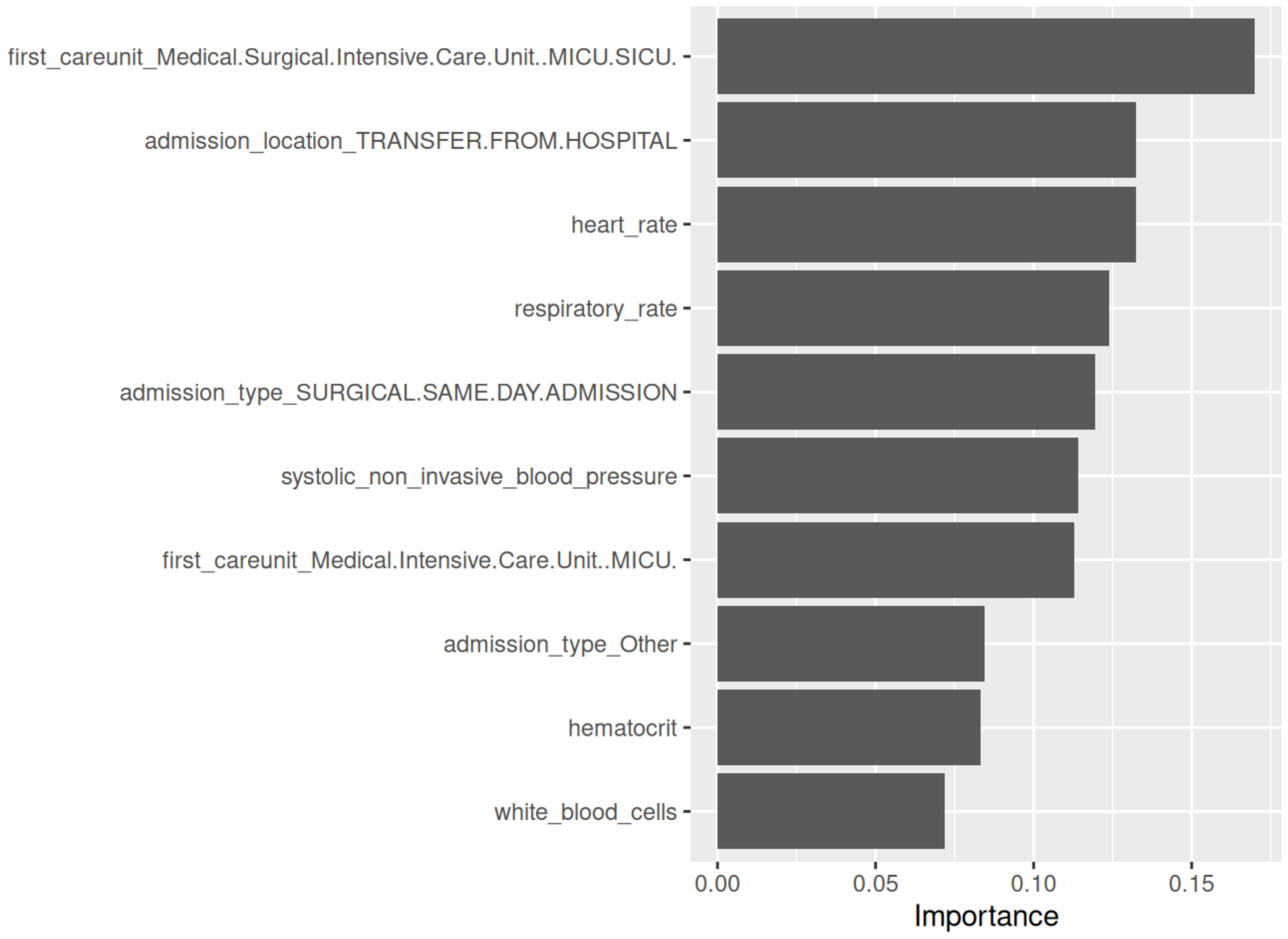
The accuracy and AUC of the final classification of the stacked model are 0.6091947 and 0.6530818 respectively, meaning that the probability of the model ranking a randomly chosen positive observation higher than a randomly chosen negative observation is 0.6530818 and the probability of correct prediction is 0.6091947.

Turning to the coefficients of each model within the stacked ensemble, the random forest model occupied the top 5 values and the XGBoosting model takes the remaining 4 positions. No information about the logistic model is shown because of the reason I mentioned earlier.

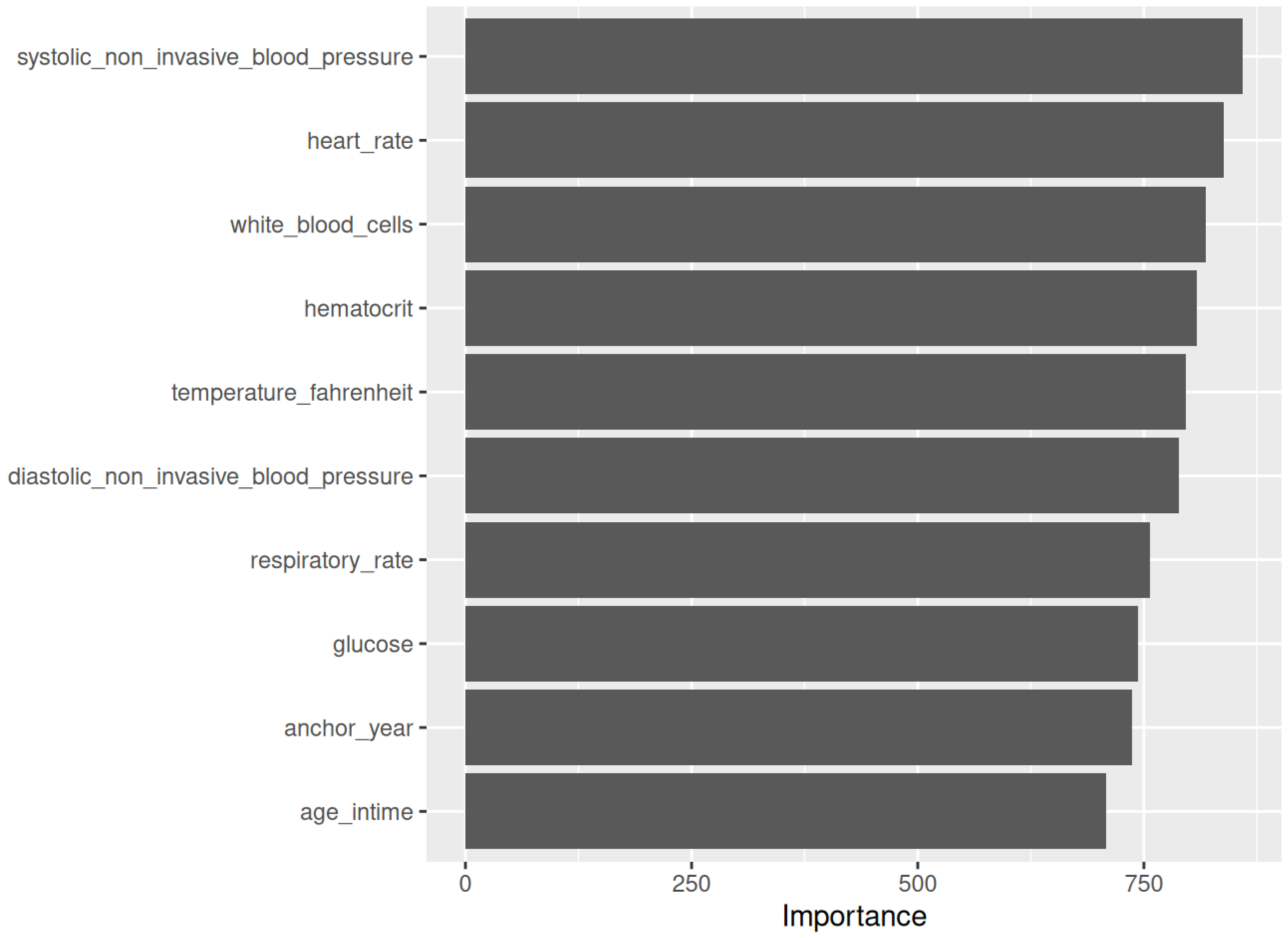
To summarize, the random forest model has the best performance in the prediction, followed by the XGBoosting model, and logitic model being the least efficient one.

Report the most important features in predicting long ICU stays

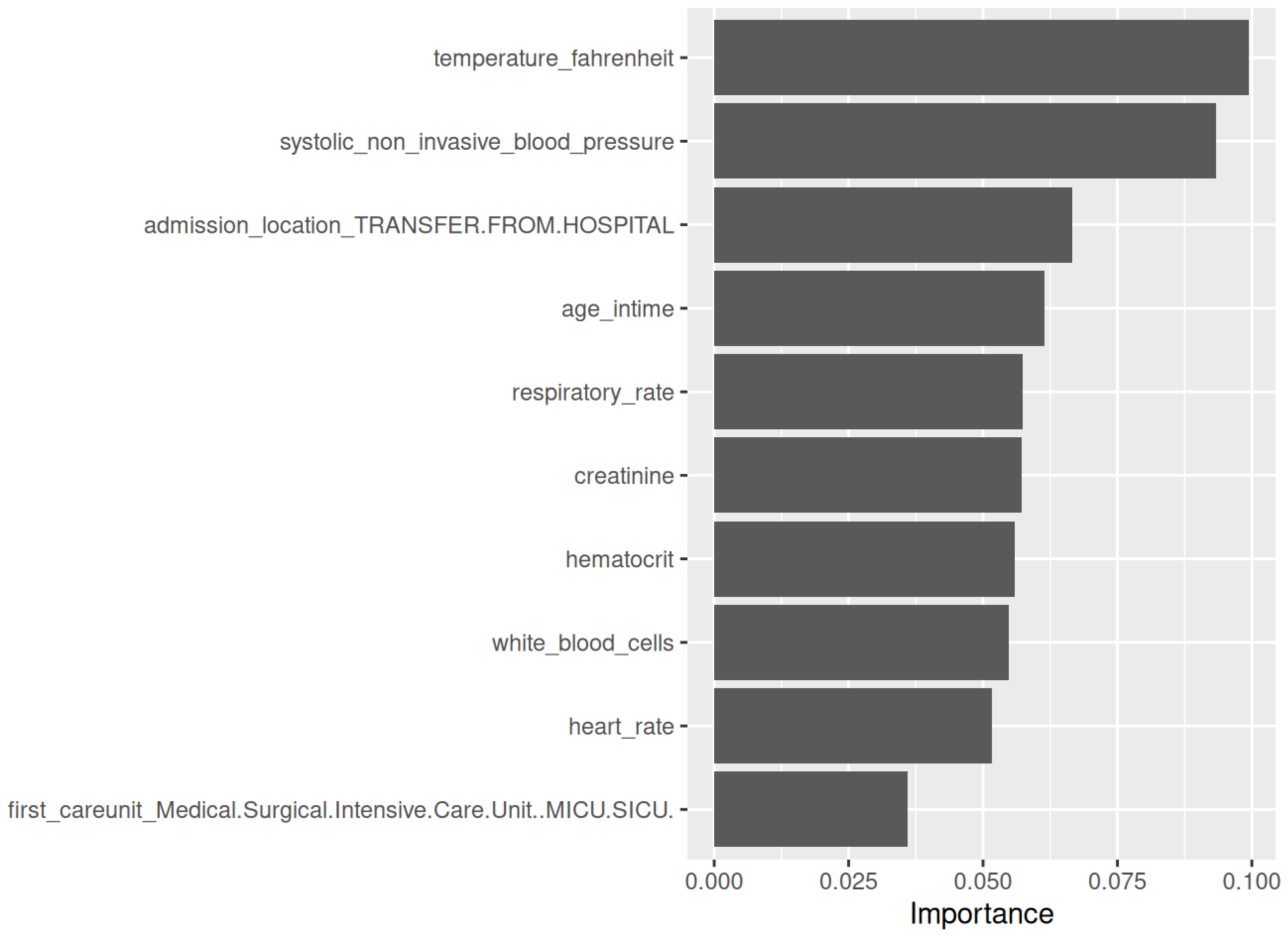
```r
# Logistic regression with elastic net regularization
logit_final_fit %>%
  extract_fit_parsnip() %>%
  vip::vip() %>%
  print()
```



```
# Random forest
rf_final_fit %>%
  extract_fit_engine() %>%
  vip::vip() %>%
  print()
```



```
# XGBoosting
xgb_final_fit %>%
  extract_fit_engine() %>%
  vip::vip() %>%
  print()
```



As shown in the above plots, the most important feature in predicting long ICU stays for logistic regression model is first careunit. For random forest model, the most important feature is systolic noninvasive blood pressure. If we look close enough, we will find that heart rate also plays a very important role in prediction. In fact, for this model, there are 7 variables (systolic noninvasive blood pressure, heart rate, white blood cells, hematocrit, temperature fahrenheit, diastolic noninvasive blood pressure, and respiratory rate) exit the importance of 750. For XGBoosting model, the most important variable is temperature fahrenheit. #### Summary of performance and interpretability In general, random forest model has the best performance, followed by XGBoosting model, and logistic regression model the least efficient. But in terms of intepretability, logistic regression model is the best.