Biostat 203B Homework 5

Due Mar 20 @ 11:59PM

Amaan Jogia-Sattar, 206324648

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(tidymodels)
```

```
-- Attaching packages -------------------------------------- tidymodels 1.3.0 --
v broom        1.0.7     v rsample      1.2.1
v dials        1.4.0     v tune         1.3.0
v infer        1.0.7     v workflows    1.2.0
v modeldata    1.4.0     v workflowsets 1.1.0
v parsnip      1.3.1     v yardstick    1.3.2
v recipes      1.1.1
-- Conflicts ----------------------------------------- tidymodels_conflicts() --
x scales::discard() masks purrr::discard()
x dplyr::filter()   masks stats::filter()
x recipes::fixed()  masks stringr::fixed()
x dplyr::lag()      masks stats::lag()
x yardstick::spec() masks readr::spec()
x recipes::step()   masks stats::step()
```

```r
library(GGally)
```

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

```r
library(gtsummary)
library(naniar)
library(lubridate)
library(glmnet)
```

```
Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-8
```

```r
library(vip)
```

```
Attaching package: 'vip'

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

    vi
```

```r
library(ranger)
library(doParallel)
```

```
Loading required package: foreach

Attaching package: 'foreach'

The following objects are masked from 'package:purrr':
```

```
    accumulate, when

Loading required package: iterators
Loading required package: parallel
```

```r
library(xgboost)
```

```


Attaching package: 'xgboost'

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

    slice
```

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

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.

First, we need to load in the data and preprocess it. We will use the `mimic_icu_cohort_rds` file we created in Homework 4. We do not have to copy mimic_icu_cohort.rds into. Instead, we can use `../hw4/mimiciv_shiny/mimic_icu_cohort.rds`.

```r
# Load the data
mimiciv_icu_cohort <- readRDS("../hw4/mimiciv_shiny/mimic_icu_cohort.rds")
```

We can now go ahead with preprocessing. Let's take a look at our dataset:

```r
head(mimiciv_icu_cohort)
```

```
# A tibble: 6 x 41
  subject_id  hadm_id  stay_id first_careunit  last_careunit intime
       <int>    <int>    <int> <chr>           <chr>         <dttm>
1   10000032 29079034 39553978 Medical Intens~ Medical Inte~ 2180-07-23 14:00:00
2   10000690 25860671 37081114 Medical Intens~ Medical Inte~ 2150-11-02 19:37:00
3   10000980 26913865 39765666 Medical Intens~ Medical Inte~ 2189-06-27 08:42:00
4   10001217 24597018 37067082 Surgical Inten~ Surgical Int~ 2157-11-20 19:18:02
5   10001217 27703517 34592300 Surgical Inten~ Surgical Int~ 2157-12-19 15:42:24
6   10001725 25563031 31205490 Medical/Surgic~ Medical/Surg~ 2110-04-11 15:52:22
# i 35 more variables: outtime <dttm>, los <dbl>, admittime <dttm>,
#   dischtime <dttm>, deathtime <dttm>, admission_type <chr>,
#   admit_provider_id <chr>, admission_location <chr>,
#   discharge_location <chr>, insurance <chr>, language <chr>,
#   marital_status <chr>, race <chr>, edregtime <dttm>, edouttime <dttm>,
#   hospital_expire_flag <int>, gender <chr>, anchor_age <int>,
#   anchor_year <int>, anchor_year_group <chr>, dod <date>, glucose <dbl>, ...
```

```r
str(mimiciv_icu_cohort)
```

```
tibble [94,458 x 41] (S3: tbl_df/tbl/data.frame)
 $ subject_id          : int [1:94458] 10000032 10000690 10000980 10001217 10001217 1000
 $ hadm_id             : int [1:94458] 29079034 25860671 26913865 24597018 27703517 2556
 $ stay_id             : int [1:94458] 39553978 37081114 39765666 37067082 34592300 3120
 $ first_careunit      : chr [1:94458] "Medical Intensive Care Unit (MICU)" "Medical Inte
 $ last_careunit       : chr [1:94458] "Medical Intensive Care Unit (MICU)" "Medical Inte
 $ intime              : POSIXct[1:94458], format: "2180-07-23 14:00:00" "2150-11-02 19:
 $ outtime             : POSIXct[1:94458], format: "2180-07-23 23:50:47" "2150-11-06 17:
 $ los                 : num [1:94458] 0.41 3.893 0.498 1.118 0.948 ...
 $ admittime           : POSIXct[1:94458], format: "2180-07-23 12:35:00" "2150-11-02 18:
 $ dischtime           : POSIXct[1:94458], format: "2180-07-25 17:55:00" "2150-11-12 13:
 $ deathtime           : POSIXct[1:94458], format: NA NA ...
 $ admission_type      : chr [1:94458] "EW EMER." "EW EMER." "EW EMER." "EW EMER." ...
 $ admit_provider_id   : chr [1:94458] "P060TX" "P26QQ4" "P060TX" "P3610N" ...
 $ admission_location  : chr [1:94458] "EMERGENCY ROOM" "EMERGENCY ROOM" "EMERGENCY ROOM
 $ discharge_location  : chr [1:94458] "HOME" "REHAB" "HOME HEALTH CARE" "HOME HEALTH CA
 $ insurance           : chr [1:94458] "Medicaid" "Medicare" "Medicare" "Private" ...
 $ language            : chr [1:94458] "English" "English" "English" "Other" ...
 $ marital_status      : chr [1:94458] "WIDOWED" "WIDOWED" "MARRIED" "MARRIED" ...
 $ race                : chr [1:94458] "WHITE" "WHITE" "BLACK/AFRICAN AMERICAN" "WHITE"
```

```
 $ edregtime               : POSIXct[1:94458], format: "2180-07-23 05:54:00" "2150-11-02 11:4
 $ edouttime               : POSIXct[1:94458], format: "2180-07-23 14:00:00" "2150-11-02 19:3
 $ hospital_expire_flag    : int [1:94458] 0 0 0 0 0 0 1 1 0 0 ...
 $ gender                  : chr [1:94458] "F" "F" "F" "F" ...
 $ anchor_age              : int [1:94458] 52 86 73 55 55 46 73 68 53 56 ...
 $ anchor_year             : int [1:94458] 2180 2150 2186 2157 2157 2110 2131 2122 2156 2162
 $ anchor_year_group       : chr [1:94458] "2014 - 2016" "2008 - 2010" "2008 - 2010" "2011 -
 $ dod                     : Date[1:94458], format: "2180-09-09" "2152-01-30" ...
 $ glucose                 : num [1:94458] 102 85 89 112 87 NA 131 141 288 95 ...
 $ potassium               : num [1:94458] 6.7 4.8 3.9 4.2 4.1 4.1 3.9 4.5 3.5 6.5 ...
 $ sodium                  : num [1:94458] 126 137 144 142 142 139 138 130 137 125 ...
 $ chloride                : num [1:94458] 95 100 109 108 104 98 97 88 102 NA ...
 $ creatinine              : num [1:94458] 0.7 1 2.3 0.6 0.5 NA 1.3 1.1 0.9 3.1 ...
 $ wbc_count               : num [1:94458] 6.9 7.1 5.3 15.7 5.4 NA 10.4 12.2 7.2 16.8 ...
 $ bicarbonate             : num [1:94458] 25 26 21 22 30 NA 28 30 24 18 ...
 $ hematocrit              : num [1:94458] 41.1 36.1 27.3 38.1 37.4 NA 31.4 39.7 34.9 34.3 .
 $ Noninvasive BP Diastolic: num [1:94458] 48 56.5 102 90 93.3 ...
 $ Respiratory Rate        : num [1:94458] 24 24.3 23.5 18 14 ...
 $ Noninvasive BP Systolic : num [1:94458] 84 106 154 151 156 ...
 $ Heart Rate              : num [1:94458] 91 78 76 86 79.3 ...
 $ Temperature_F           : num [1:94458] 98.7 97.7 98 98.5 97.6 97.7 97.9 98.1 97.2 97.9 .
 $ age_intime              : int [1:94458] 52 86 76 55 55 46 76 77 57 56 ...
```

We first adapt our preprocessing code from HW4:

```
mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  mutate(
    first_careunit = fct_lump_n(first_careunit,
                                n = 4,
                                other_level = "Other"),
    last_careunit = fct_lump_n(last_careunit,
                               n = 4,
                               other_level = "Other"),
    admission_type = fct_lump_n(admission_type,
                                n = 4,
                                other_level = "Other"),
    admission_location = fct_lump_n(admission_location,
                                    n = 3,
                                    other_level = "Other"),
    discharge_location = fct_lump_n(discharge_location,
                                    n = 4,
                                    other_level = "Other")
```

```
  ) %>%
  # Ensure race is a factor so we can work with its levels
  mutate(race = factor(race)) %>%
  {
    # Capture the current levels of race
    race_levels <- levels(.$race)
    mutate(., race = fct_collapse(race,
      ASIAN    = race_levels[grep("ASIAN",
                                  race_levels)],
      BLACK    = race_levels[grep("BLACK",
                                  race_levels)],
      HISPANIC = race_levels[grep("HISPANIC",
                                  race_levels)],
      WHITE    = race_levels[grep("WHITE",
                                  race_levels)],
      OTHER    = setdiff(race_levels,
                         c(race_levels[grep("ASIAN",
                                            race_levels)],
                           race_levels[grep("BLACK",
                                            race_levels)],
                           race_levels[grep("HISPANIC",
                                            race_levels)],
                           race_levels[grep("WHITE",
                                            race_levels)]))
    ))
  }

mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  mutate(
    insurance = as.factor(insurance),
    language = as.factor(language),
    marital_status = as.factor(marital_status),
    gender = as.factor(gender)
  )

mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  mutate(los_long = los >= 2) %>%
  mutate(los_long = as.factor(los_long))

mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  filter(!is.na(los_long))
```

```r
mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  select(
    subject_id,
    hadm_id,
    stay_id,
    intime,
    first_careunit,
    los_long,
    admission_type,
    admission_location,
    insurance,
    language,
    marital_status,
    race,
    gender,
    chloride,
    creatinine,
    sodium,
    potassium,
    glucose,
    hematocrit,
    wbc_count,
    bicarbonate,
    `Noninvasive BP Systolic`,
    `Noninvasive BP Diastolic`,
    `Respiratory Rate`,
    `Temperature_F`,
    `Heart Rate`,
    age_intime
  )

mimiciv_icu_cohort
```

```
# A tibble: 94,444 x 27
   subject_id  hadm_id  stay_id intime              first_careunit      los_long
        <int>    <int>    <int> <dttm>              <fct>               <fct>
 1   10000032 29079034 39553978 2180-07-23 14:00:00 Medical Intensive ~ FALSE
 2   10000690 25860671 37081114 2150-11-02 19:37:00 Medical Intensive ~ TRUE
 3   10000980 26913865 39765666 2189-06-27 08:42:00 Medical Intensive ~ FALSE
 4   10001217 24597018 37067082 2157-11-20 19:18:02 Surgical Intensive~ FALSE
 5   10001217 27703517 34592300 2157-12-19 15:42:24 Surgical Intensive~ FALSE
```

```
 6   10001725 25563031 31205490 2110-04-11 15:52:22 Medical/Surgical I~ FALSE
 7   10001843 26133978 39698942 2134-12-05 18:50:03 Medical/Surgical I~ FALSE
 8   10001884 26184834 37510196 2131-01-11 04:20:05 Medical Intensive ~ TRUE
 9   10002013 23581541 39060235 2160-05-18 10:00:53 Cardiac Vascular I~ FALSE
10   10002114 27793700 34672098 2162-02-17 23:30:00 Other               TRUE
# i 94,434 more rows
# i 21 more variables: admission_type <fct>, admission_location <fct>,
#   insurance <fct>, language <fct>, marital_status <fct>, race <fct>,
#   gender <fct>, chloride <dbl>, creatinine <dbl>, sodium <dbl>,
#   potassium <dbl>, glucose <dbl>, hematocrit <dbl>, wbc_count <dbl>,
#   bicarbonate <dbl>, `Noninvasive BP Systolic` <dbl>,
#   `Noninvasive BP Diastolic` <dbl>, `Respiratory Rate` <dbl>, ...
```

Double-checking how our variables are stored:

```r
str(mimiciv_icu_cohort)
```

```
tibble [94,444 x 27] (S3: tbl_df/tbl/data.frame)
 $ subject_id              : int [1:94444] 10000032 10000690 10000980 10001217 10001217 1000
 $ hadm_id                 : int [1:94444] 29079034 25860671 26913865 24597018 27703517 2556
 $ stay_id                 : int [1:94444] 39553978 37081114 39765666 37067082 34592300 3120
 $ intime                  : POSIXct[1:94444], format: "2180-07-23 14:00:00" "2150-11-02 19:
 $ first_careunit          : Factor w/ 5 levels "Cardiac Vascular Intensive Care Unit (CVICU
 $ los_long                : Factor w/ 2 levels "FALSE","TRUE": 1 2 1 1 1 1 1 2 1 2 ...
 $ admission_type          : Factor w/ 5 levels "EW EMER.","OBSERVATION ADMIT",..: 1 1 1 1 5
 $ admission_location      : Factor w/ 4 levels "EMERGENCY ROOM",..: 1 1 1 1 2 4 3 1 2 2 ...
 $ insurance               : Factor w/ 5 levels "Medicaid","Medicare",..: 1 2 2 5 5 5 2 2 2 
 $ language                : Factor w/ 25 levels "American Sign Language",..: 7 7 7 17 17 7 
 $ marital_status          : Factor w/ 4 levels "DIVORCED","MARRIED",..: 4 4 2 2 2 2 3 2 3 N
 $ race                    : Factor w/ 5 levels "OTHER","ASIAN",..: 5 5 3 5 5 5 5 3 1 1 ...
 $ gender                  : Factor w/ 2 levels "F","M": 1 1 1 1 1 1 2 1 1 2 ...
 $ chloride                : num [1:94444] 95 100 109 108 104 98 97 88 102 NA ...
 $ creatinine              : num [1:94444] 0.7 1 2.3 0.6 0.5 NA 1.3 1.1 0.9 3.1 ...
 $ sodium                  : num [1:94444] 126 137 144 142 142 139 138 130 137 125 ...
 $ potassium               : num [1:94444] 6.7 4.8 3.9 4.2 4.1 4.1 3.9 4.5 3.5 6.5 ...
 $ glucose                 : num [1:94444] 102 85 89 112 87 NA 131 141 288 95 ...
 $ hematocrit              : num [1:94444] 41.1 36.1 27.3 38.1 37.4 NA 31.4 39.7 34.9 34.3 .
 $ wbc_count               : num [1:94444] 6.9 7.1 5.3 15.7 5.4 NA 10.4 12.2 7.2 16.8 ...
 $ bicarbonate             : num [1:94444] 25 26 21 22 30 NA 28 30 24 18 ...
 $ Noninvasive BP Systolic : num [1:94444] 84 106 154 151 156 ...
 $ Noninvasive BP Diastolic: num [1:94444] 48 56.5 102 90 93.3 ...
 $ Respiratory Rate        : num [1:94444] 24 24.3 23.5 18 14 ...
```

```
 $ Temperature_F           : num [1:94444] 98.7 97.7 98 98.5 97.6 97.7 97.9 98.1 97.2 97.9 .
 $ Heart Rate              : num [1:94444] 91 78 76 86 79.3 ...
 $ age_intime              : int [1:94444] 52 86 76 55 55 46 76 77 57 56 ...
```

Now, we can check for missing values across our dataset:

```r
miss_var_summary(mimiciv_icu_cohort)
```

```
# A tibble: 27 x 3
   variable       n_miss pct_miss
   <chr>           <int>    <num>
 1 glucose         11654    12.3 
 2 bicarbonate     11549    12.2 
 3 potassium       11387    12.1 
 4 chloride        11351    12.0 
 5 sodium          11330    12.0 
 6 creatinine       8027     8.50
 7 marital_status   7756     8.21
 8 wbc_count        6850     7.25
 9 hematocrit       6751     7.15
10 Temperature_F    1675     1.77
# i 17 more rows
```

We observe that lab results make up the majority of missing values. We will impute numeric variables with the median and categorical variables with the mode.Moreover, we can convert categorical variables into dummy (one-hot) encoded variables. We will also normalize our numeric predictors using centering and scaling. We will use the 'tidymodels' package to do this. For 'intime', we can extract the hour of admission and represent it cyclically using trigonometric transformations (sine and cosine). This approach is commonly utilized for encoding 24-hour time in machine learning models, and is further explained in https://ianlondon.github.io/posts/encoding-cyclical-features-24-hour-time/.

Here is our prepared recipe for preprocessing our data.

```r
icu_recipe<- recipe(los_long ~ ., data = mimiciv_icu_cohort) %>%
  update_role(subject_id, hadm_id, stay_id, new_role = "ID") %>%
  # Extract the hour from intime
  step_mutate(admission_hour = hour(intime)) %>%
  # Create cyclic features for the hour
  step_mutate(
    hour_sin = sin(2 * pi * admission_hour / 24),
    hour_cos = cos(2 * pi * admission_hour / 24)
```

```
  ) %>%
  # Remove the original intime and raw admission_hour if not needed
  step_rm(intime, admission_hour) %>%
  # Remaining steps: imputation, dummy coding, normalization
  step_impute_median(all_numeric_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())

summary(icu_recipe)
```

```
# A tibble: 27 x 4
   variable           type      role      source
   <chr>              <list>    <chr>     <chr>
 1 subject_id         <chr [2]> ID        original
 2 hadm_id            <chr [2]> ID        original
 3 stay_id            <chr [2]> ID        original
 4 intime             <chr [1]> predictor original
 5 first_careunit     <chr [3]> predictor original
 6 admission_type     <chr [3]> predictor original
 7 admission_location <chr [3]> predictor original
 8 insurance          <chr [3]> predictor original
 9 language           <chr [3]> predictor original
10 marital_status     <chr [3]> predictor original
# i 17 more rows
```

MODEL 1: LOGISTIC REGRESSION WITH ENET REGULARIZATION

Our first model is as follows:

```
logit_mod <-
  logistic_reg(
    penalty = tune(),
    mixture = tune()
  ) |>
  set_engine("glmnet", standardize = FALSE) |>
  print()
```

```
Logistic Regression Model Specification (classification)
```

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

Engine-Specific Arguments:
  standardize = FALSE

Computational engine: glmnet
```

Now, we can do our initial split of the data.

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.

```
# #| eval: false
set.seed(203)

# sort
mimiciv_icu_cohort <- mimiciv_icu_cohort |>
  arrange(subject_id, hadm_id, stay_id)

data_split <- initial_split(
  mimiciv_icu_cohort,
  # stratify by los_long
  strata = "los_long",
  prop = 0.5
  )
```

Extracting our training and testing sets:

```
train_data <- training(data_split)
test_data <- testing(data_split)
```

Now, we combine our recipe and logistic regression model into a workflow:

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

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

-- Preprocessor ----------------------------------------------------------------
8 Recipe Steps

* step_mutate()
* step_mutate()
* step_rm()
* 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
```

Now, we can tune our hyperparameters:

```r
# Define the tuning grid
param_grid <- grid_regular(
  penalty(range = c(-6, 3)),
  mixture(),
  levels = c(100, 5)
) |> print()
```

```
# A tibble: 500 x 2
      penalty mixture
        <dbl>   <dbl>
 1 0.000001         0
 2 0.00000123       0
```

```
 3 0.00000152       0
 4 0.00000187       0
 5 0.00000231       0
 6 0.00000285       0
 7 0.00000351       0
 8 0.00000433       0
 9 0.00000534       0
10 0.00000658       0
# i 490 more rows
```

Next, we set cross-validation partitioning, creating 5 folds:

```r
set.seed(203)
folds <- vfold_cv(train_data, v = 5, strata = los_long)
```

Having our workflow and tuning grid, we run the grid search:

```r
logit_fit <- logit_wf |>
  tune_grid(
    resamples = folds,
    grid = param_grid,
    metrics = metric_set(roc_auc, accuracy)
  )
```

We can inspect the results:

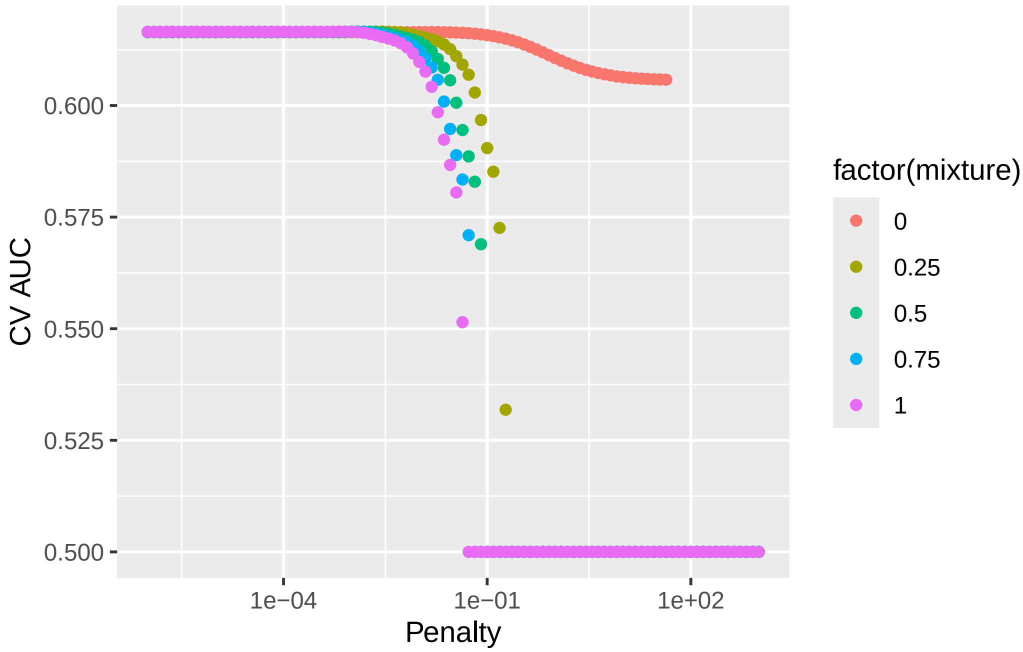
```r
logit_fit
```

```
# Tuning results
# 5-fold cross-validation using stratification
# A tibble: 5 x 4
  splits               id    .metrics             .notes
  <list>               <chr> <list>               <list>
1 <split [37776/9445]> Fold1 <tibble [1,000 x 6]> <tibble [0 x 3]>
2 <split [37776/9445]> Fold2 <tibble [1,000 x 6]> <tibble [0 x 3]>
3 <split [37776/9445]> Fold3 <tibble [1,000 x 6]> <tibble [0 x 3]>
4 <split [37778/9443]> Fold4 <tibble [1,000 x 6]> <tibble [0 x 3]>
5 <split [37778/9443]> Fold5 <tibble [1,000 x 6]> <tibble [0 x 3]>
```

And we can visualize results:

```r
logit_fit |>
  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 x 8
      penalty mixture .metric  .estimator  mean     n std_err
        <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl>
 1 0.000001         0 accuracy binary     0.584     5 0.00103
 2 0.000001         0 roc_auc  binary     0.616     5 0.00151
 3 0.00000123       0 accuracy binary     0.584     5 0.00103
 4 0.00000123       0 roc_auc  binary     0.616     5 0.00151
 5 0.00000152       0 accuracy binary     0.584     5 0.00103
 6 0.00000152       0 roc_auc  binary     0.616     5 0.00151
 7 0.00000187       0 accuracy binary     0.584     5 0.00103
 8 0.00000187       0 roc_auc  binary     0.616     5 0.00151
 9 0.00000231       0 accuracy binary     0.584     5 0.00103
10 0.00000231       0 roc_auc  binary     0.616     5 0.00151
   .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
```



This plot will show us how performance (CV AUC) varies with different penalty and mixture settings.

Next, we can review the best-performing models and select the top one:

```
# Show the top 5 models based on ROC AUC
logit_fit |>
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 8
   penalty mixture .metric .estimator  mean     n std_err .config
     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1 0.00285     0.25 roc_auc binary     0.617     5 0.00162 Preprocessor1_Model139
2 0.00123     0.5  roc_auc binary     0.617     5 0.00159 Preprocessor1_Model235
3 0.000658    1    roc_auc binary     0.617     5 0.00160 Preprocessor1_Model432
4 0.00231     0.25 roc_auc binary     0.617     5 0.00159 Preprocessor1_Model138
5 0.000811    0.75 roc_auc binary     0.617     5 0.00159 Preprocessor1_Model333
```

```
# Select the best model
best_logit <- logit_fit |>
  select_best(metric = "roc_auc")
best_logit
```

```
# A tibble: 1 x 3
  penalty mixture .config
    <dbl>   <dbl> <chr>
1 0.00285    0.25 Preprocessor1_Model139
```

We finalize our workflow:

```r
final_logit_wf <- logit_wf |>
  finalize_workflow(best_logit)
final_logit_wf
```

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

-- Preprocessor ----------------------------------------------------------------
8 Recipe Steps

* step_mutate()
* step_mutate()
* step_rm()
* step_impute_median()
* step_impute_mode()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = 0.0028480358684358
  mixture = 0.25

Engine-Specific Arguments:
  standardize = FALSE

Computational engine: glmnet
```

Now, we can fit the final model on the entire trainingg set and evaluate it on the test set using the `last_fit()` function:

```r
final_logit_fit <- final_logit_wf |>
  last_fit(data_split)
final_logit_fit
```

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

```r
# Collect test metrics
final_logit_fit |>
  collect_metrics()
```

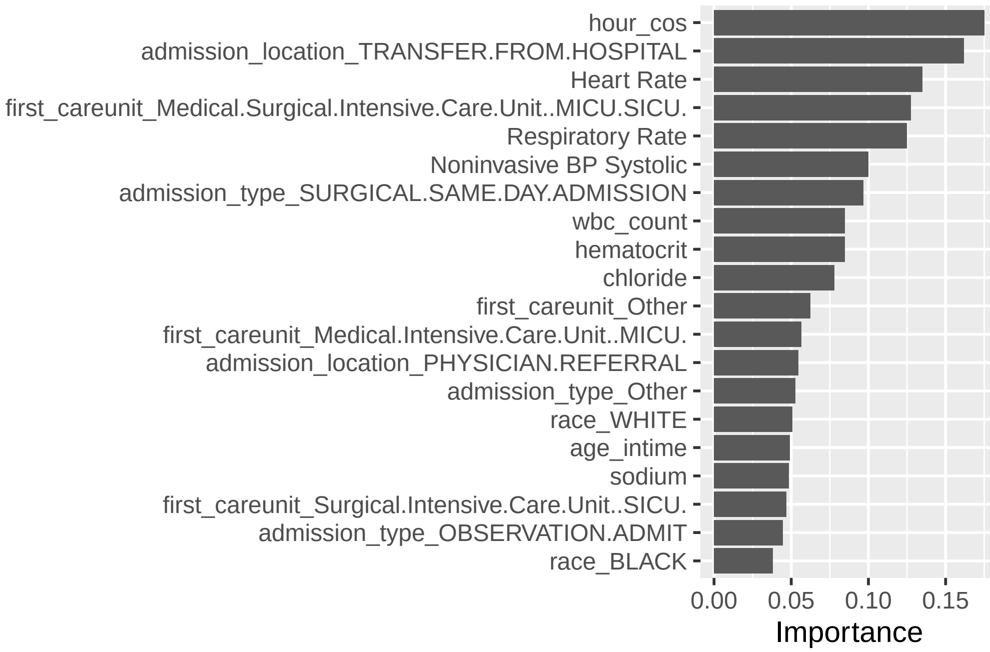
```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.582 Preprocessor1_Model1
2 roc_auc     binary         0.614 Preprocessor1_Model1
3 brier_class binary         0.240 Preprocessor1_Model1
```

For our Logistic Regression Model (with ENet Regularization), we observe an AUC of 0.614202 and an accuracy of 0.5820469.

Examining feature importance:

```r
final_logit_model <- final_logit_fit %>%
  extract_fit_parsnip()

# Create a VIP plot:
vip(final_logit_model, num_features = 20)
```



It appears that the cosine component of hour of admission `hour_cos`, indicator for admission location `admission_location_TRANSFER.FROM_HOSPITAL`, `Heart Rate`, indicator for first care unit `first_careunit_Medical.Surgical.Intensive.Care.Unit..MICU.SICU.`, `Respiratory Rate`, and `Noninvasive BP Systolic` were among the most important features.

MODEL 2: RANDOM FOREST We began this process using a coarser tuning grid to identify a promising region of the parameter space. We then refined our grid search within this region. We utilized parallel processing to expedite the tuning process. First, we define our random forest model:

```
rf_mod <-
  rand_forest(
    mode = "classification",
    mtry = tune(),     # number of predictors randomly sampled at each split
    trees = tune()     # number of trees in the ensemble
  ) %>%
  set_engine("ranger", importance = "impurity")

rf_mod
```

```
Random Forest Model Specification (classification)
```

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

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

Next, we combine our recipe and random forest model into a workflow:

```r
rf_wf <- workflow() |>
  add_recipe(icu_recipe) |>
  add_model(rf_mod) |>
  print()
```

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

-- Preprocessor ----------------------------------------------------------------
8 Recipe Steps

* step_mutate()
* step_mutate()
* step_rm()
* 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
```

Next, we define our tuning grid:

```r
param_grid_rf <- grid_regular(
  trees(range = c(100L, 250L)),
  mtry(range = c(1L, 3L)),
  levels = c(3, 3)
) %>% print()
```

```
# A tibble: 9 x 2
  trees  mtry
  <int> <int>
1   100     1
2   175     1
3   250     1
4   100     2
5   175     2
6   250     2
7   100     3
8   175     3
9   250     3
```

Next, we set cross-validation partitioning, creating 5 folds:

```r
set.seed(203)
rf_folds <- vfold_cv(train_data, v = 5, strata = los_long)
rf_folds
```

```
#  5-fold cross-validation using stratification
# A tibble: 5 x 2
  splits               id
  <list>               <chr>
1 <split [37776/9445]> Fold1
2 <split [37776/9445]> Fold2
3 <split [37776/9445]> Fold3
4 <split [37778/9443]> Fold4
5 <split [37778/9443]> Fold5
```

Set up parallel processing: use all cores minus one.

```r
cl <- makeCluster(detectCores() - 1)
registerDoParallel(cl)

clusterEvalQ(cl, {
  library(tidyverse)
  library(tidymodels)
  library(GGally)
  library(gtsummary)
  library(naniar)
  library(lubridate)
  library(glmnet)
  library(vip)
  library(ranger)
})
```

```
[[1]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[2]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[3]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
```

```
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[4]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[5]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[6]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[7]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"
```

```
[[8]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[9]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[10]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[11]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"
```

Having our workflow and tuning grid, we run the grid search:

```r
rf_fit_coarse <- rf_wf |>
  tune_grid(
    resamples = rf_folds,
    grid = param_grid_rf,
    metrics = metric_set(roc_auc, accuracy)
  )
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

```r
rf_fit_coarse
```

```
# Tuning results
# 5-fold cross-validation using stratification
# A tibble: 5 x 4
  splits               id    .metrics          .notes
  <list>               <chr> <list>            <list>
1 <split [37776/9445]> Fold1 <tibble [18 x 6]> <tibble [0 x 3]>
2 <split [37776/9445]> Fold2 <tibble [18 x 6]> <tibble [0 x 3]>
3 <split [37776/9445]> Fold3 <tibble [18 x 6]> <tibble [0 x 3]>
4 <split [37778/9443]> Fold4 <tibble [18 x 6]> <tibble [0 x 3]>
5 <split [37778/9443]> Fold5 <tibble [18 x 6]> <tibble [0 x 3]>
```

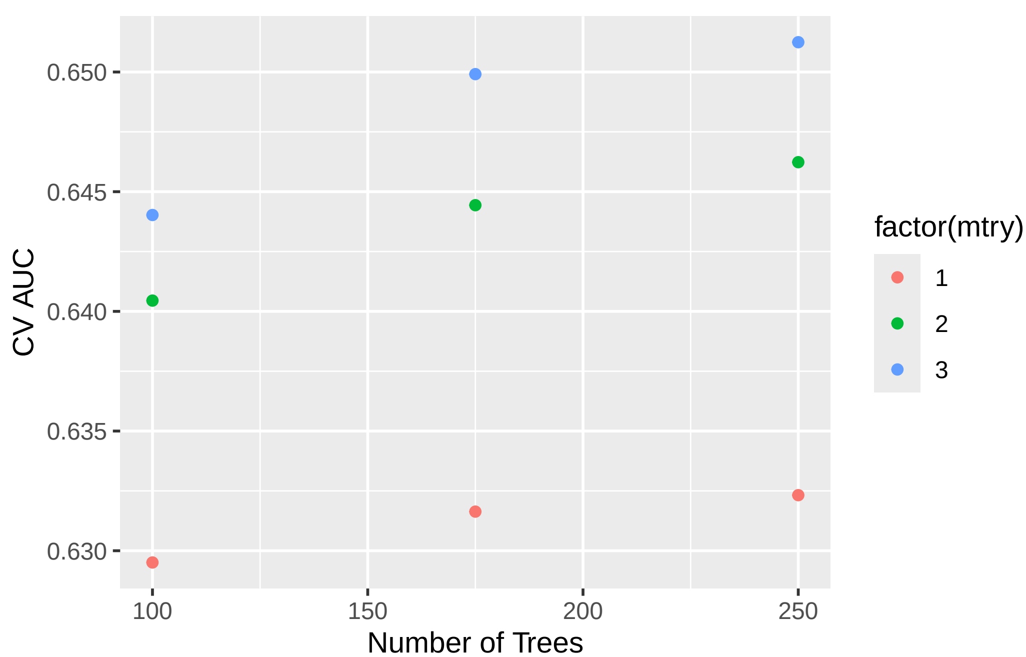
Stop the parallel cluster after tuning:

```r
stopCluster(cl)
```

Visualizing the results:

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

```
# A tibble: 18 x 8
    mtry trees .metric  .estimator  mean     n std_err .config
   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1     1   100 accuracy binary     0.585     5 0.00368 Preprocessor1_Model1
 2     1   100 roc_auc  binary     0.630     5 0.00359 Preprocessor1_Model1
 3     1   175 accuracy binary     0.583     5 0.00222 Preprocessor1_Model2
 4     1   175 roc_auc  binary     0.632     5 0.00233 Preprocessor1_Model2
 5     1   250 accuracy binary     0.584     5 0.00286 Preprocessor1_Model3
 6     1   250 roc_auc  binary     0.632     5 0.00293 Preprocessor1_Model3
 7     2   100 accuracy binary     0.601     5 0.00156 Preprocessor1_Model4
 8     2   100 roc_auc  binary     0.640     5 0.00293 Preprocessor1_Model4
 9     2   175 accuracy binary     0.603     5 0.00309 Preprocessor1_Model5
10     2   175 roc_auc  binary     0.644     5 0.00290 Preprocessor1_Model5
11     2   250 accuracy binary     0.604     5 0.00267 Preprocessor1_Model6
12     2   250 roc_auc  binary     0.646     5 0.00294 Preprocessor1_Model6
13     3   100 accuracy binary     0.603     5 0.00275 Preprocessor1_Model7
14     3   100 roc_auc  binary     0.644     5 0.00303 Preprocessor1_Model7
15     3   175 accuracy binary     0.609     5 0.00201 Preprocessor1_Model8
16     3   175 roc_auc  binary     0.650     5 0.00252 Preprocessor1_Model8
17     3   250 accuracy binary     0.609     5 0.00213 Preprocessor1_Model9
18     3   250 roc_auc  binary     0.651     5 0.00255 Preprocessor1_Model9
```



Next, we can review the best-performing models and select the top one:

```r
rf_fit_coarse %>%
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 8
   mtry trees .metric .estimator  mean     n std_err .config
  <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1     3   250 roc_auc binary     0.651     5 0.00255 Preprocessor1_Model9
2     3   175 roc_auc binary     0.650     5 0.00252 Preprocessor1_Model8
3     2   250 roc_auc binary     0.646     5 0.00294 Preprocessor1_Model6
4     2   175 roc_auc binary     0.644     5 0.00290 Preprocessor1_Model5
5     3   100 roc_auc binary     0.644     5 0.00303 Preprocessor1_Model7
```

```r
best_rf <- rf_fit_coarse %>%
  select_best(metric = "roc_auc")
best_rf
```

```
# A tibble: 1 x 3
   mtry trees .config
  <int> <int> <chr>
1     3   250 Preprocessor1_Model9
```

We finalize our workflow:

```r
final_rf_wf <- rf_wf %>%
  finalize_workflow(best_rf)
final_rf_wf
```

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

-- Preprocessor ---------------------------------------------------------------
8 Recipe Steps

* step_mutate()
* step_mutate()
* step_rm()
* step_impute_median()
* step_impute_mode()
* step_dummy()
```

```
* step_zv()
* step_normalize()

-- Model ---------------------------------------------------------------------
Random Forest Model Specification (classification)

Main Arguments:
  mtry = 3
  trees = 250

Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

Now, we can fit the final model on the entire training set and evaluate it on the test set using the `last_fit()` function:

```r
final_rf_fit <- final_rf_wf %>%
  last_fit(data_split)
final_rf_fit
```

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

Collect test metrics:

```r
final_rf_fit %>%
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.602 Preprocessor1_Model1
2 roc_auc     binary         0.644 Preprocessor1_Model1
3 brier_class binary         0.236 Preprocessor1_Model1
```

```r
# Show all metrics for each combination in your grid
rf_fit_coarse %>%
  collect_metrics() %>%
  print(width = Inf)
```

```
# A tibble: 18 x 8
    mtry trees .metric  .estimator  mean     n std_err .config
   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1     1   100 accuracy binary     0.585     5 0.00368 Preprocessor1_Model1
 2     1   100 roc_auc  binary     0.630     5 0.00359 Preprocessor1_Model1
 3     1   175 accuracy binary     0.583     5 0.00222 Preprocessor1_Model2
 4     1   175 roc_auc  binary     0.632     5 0.00233 Preprocessor1_Model2
 5     1   250 accuracy binary     0.584     5 0.00286 Preprocessor1_Model3
 6     1   250 roc_auc  binary     0.632     5 0.00293 Preprocessor1_Model3
 7     2   100 accuracy binary     0.601     5 0.00156 Preprocessor1_Model4
 8     2   100 roc_auc  binary     0.640     5 0.00293 Preprocessor1_Model4
 9     2   175 accuracy binary     0.603     5 0.00309 Preprocessor1_Model5
10     2   175 roc_auc  binary     0.644     5 0.00290 Preprocessor1_Model5
11     2   250 accuracy binary     0.604     5 0.00267 Preprocessor1_Model6
12     2   250 roc_auc  binary     0.646     5 0.00294 Preprocessor1_Model6
13     3   100 accuracy binary     0.603     5 0.00275 Preprocessor1_Model7
14     3   100 roc_auc  binary     0.644     5 0.00303 Preprocessor1_Model7
15     3   175 accuracy binary     0.609     5 0.00201 Preprocessor1_Model8
16     3   175 roc_auc  binary     0.650     5 0.00252 Preprocessor1_Model8
17     3   250 accuracy binary     0.609     5 0.00213 Preprocessor1_Model9
18     3   250 roc_auc  binary     0.651     5 0.00255 Preprocessor1_Model9
```

```r
# Display the top 5 models based on ROC AUC
rf_fit_coarse %>%
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 8
   mtry trees .metric .estimator  mean     n std_err .config
  <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1     3   250 roc_auc binary     0.651     5 0.00255 Preprocessor1_Model9
2     3   175 roc_auc binary     0.650     5 0.00252 Preprocessor1_Model8
3     2   250 roc_auc binary     0.646     5 0.00294 Preprocessor1_Model6
4     2   175 roc_auc binary     0.644     5 0.00290 Preprocessor1_Model5
5     3   100 roc_auc binary     0.644     5 0.00303 Preprocessor1_Model7
```

Now that we have run the coarse grid search, we can refine our grid search within the promising region of the parameter space. We will use the best-performing model from the coarse grid search as a starting point.

```r
# Select the best parameters based on ROC AUC from your coarse grid tuning
best_rf <- rf_fit_coarse %>%
  select_best(metric = "roc_auc")
print(best_rf)
```

```
# A tibble: 1 x 3
   mtry trees .config
  <int> <int> <chr>
1     3   250 Preprocessor1_Model9
```

```r
# Finalize your workflow using the best hyperparameters (mtry = 3 and trees = 250)
final_rf_wf <- rf_wf %>%
  finalize_workflow(best_rf)
print(final_rf_wf)
```

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

-- Preprocessor ----------------------------------------------------------------
8 Recipe Steps

* step_mutate()
* step_mutate()
* step_rm()
* step_impute_median()
* step_impute_mode()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Random Forest Model Specification (classification)

Main Arguments:
  mtry = 3
  trees = 250
```

```
Engine-Specific Arguments:
  importance = impurity

Computational engine: ranger
```

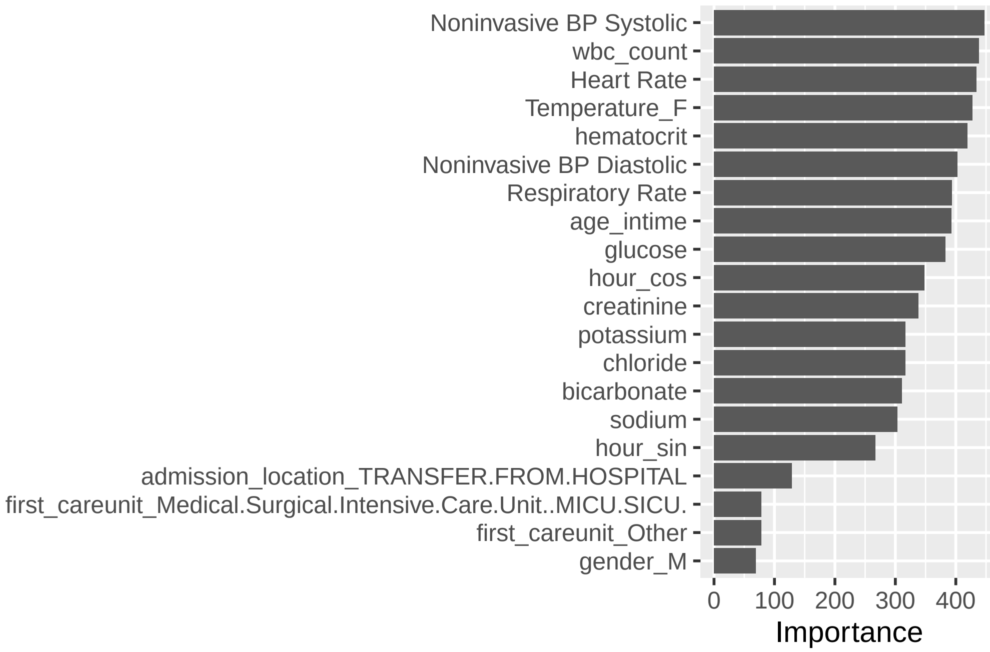
```r
# Fit the final random forest model on the entire training set and evaluate on the test set
final_rf_fit <- final_rf_wf %>%
  last_fit(data_split)
print(final_rf_fit)
```

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

```r
# Collect and display test set metrics (ROC AUC and Accuracy)
final_rf_metrics <- final_rf_fit %>%
  collect_metrics()
print(final_rf_metrics)
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.602 Preprocessor1_Model1
2 roc_auc     binary         0.645 Preprocessor1_Model1
3 brier_class binary         0.236 Preprocessor1_Model1
```

```r
# Optionally, extract the fitted model and generate a variable importance plot
final_rf_model <- final_rf_fit %>% extract_fit_parsnip()
vip(final_rf_model, num_features = 20)
```



For our Random Forest Model, we observe an AUC of 0.64527210 and an accuracy of 0.6017195. Examining feature importance based on 'Impurity', we observe that there is a group of features with relatively large importance values, followed by a drop-off in the variable importance plot. Namely, we observe that `Noninvasive BP Systolic`, `wbc_count`, `Heart Rate`, `Temperature_F`, `hematocrit`, `Noninvasive BP Diastolic`, `Respiratory Rate`, `age_intime`, `glucose`, `hour_cos`, `creatinine`, `potassium`, `chloride`, `bicarbonate`, `sodium`, and the sin component of hour of admission `hour_sin` were among the most important features. Interestingly, it appears that clinical vital signs and lab results were among the most important features in the Random Forest Model. This is a departure from what we observed in terms of variable importance in the Logistic Regression Model, where some of the categorical features recorded at the time of admission were also in the upper section of the variable importance plot.

MODEL 3: BOOSTING We proceed to fit a boosting model. First, we will define our boosting model with tunable hyperparameters:

```
# Define the XGBoost model with tunable parameters
xgb_mod <- boost_tree(
  mode = "classification",
  trees      = tune(),   # total number of trees
  tree_depth = tune(),   # maximum tree depth
  learn_rate = tune()    # learning rate
) %>%
```

```
  set_engine("xgboost")
xgb_mod
```

```
Boosted Tree Model Specification (classification)

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

Computational engine: xgboost
```

Next, we combine our recipe and boosting model into a workflow:

```
xgb_wf <- workflow() |>
  add_recipe(icu_recipe) |>
  add_model(xgb_mod) |>
  print()
```

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

-- Preprocessor ------------------------------------------------------------
8 Recipe Steps

* step_mutate()
* step_mutate()
* step_rm()
* step_impute_median()
* step_impute_mode()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -------------------------------------------------------------------
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = tune()
  tree_depth = tune()
```

```
  learn_rate = tune()

Computational engine: xgboost
```

Next, we define our tuning grid:

```
param_grid_xgb <- grid_regular(
  trees(range = c(100L, 500L)),
  tree_depth(range = c(1L, 3L)),
  learn_rate(range = c(-5, 2), trans = log10_trans()),
  levels = c(3, 3, 3)) %>%
  print()
```

```
# A tibble: 27 x 3
   trees tree_depth learn_rate
   <int>      <int>      <dbl>
 1   100          1    0.00001
 2   300          1    0.00001
 3   500          1    0.00001
 4   100          2    0.00001
 5   300          2    0.00001
 6   500          2    0.00001
 7   100          3    0.00001
 8   300          3    0.00001
 9   500          3    0.00001
10   100          1    0.0316
# i 17 more rows
```

Next, we set cross-validation partitioning, creating 5 folds:

```
set.seed(203)
xgb_folds <- vfold_cv(train_data, v = 5, strata = los_long)
xgb_folds
```

```
#  5-fold cross-validation using stratification
# A tibble: 5 x 2
  splits               id
  <list>               <chr>
1 <split [37776/9445]> Fold1
2 <split [37776/9445]> Fold2
3 <split [37776/9445]> Fold3
```

```
4 <split [37778/9443]> Fold4
5 <split [37778/9443]> Fold5
```

Next, we tune our model using the grid search. We will use parallel processing to expedite the tuning process.

```r
cl <- makeCluster(detectCores() - 1)
registerDoParallel(cl)
clusterEvalQ(cl, {
  library(tidyverse)
  library(tidymodels)
  library(GGally)
  library(gtsummary)
  library(naniar)
  library(lubridate)
  library(glmnet)
  library(vip)
  library(ranger)
})
```

```
[[1]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[2]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[3]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
```

```
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[4]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[5]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[6]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[7]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
```

```
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[8]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[9]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[10]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[11]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
```

```
[36] "methods"      "base"
```

```r
xgb_fit <- xgb_wf |>
  tune_grid(
    resamples = xgb_folds,
    grid = param_grid_xgb,
    metrics = metric_set(roc_auc, accuracy)
  )
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

```r
xgb_fit
```

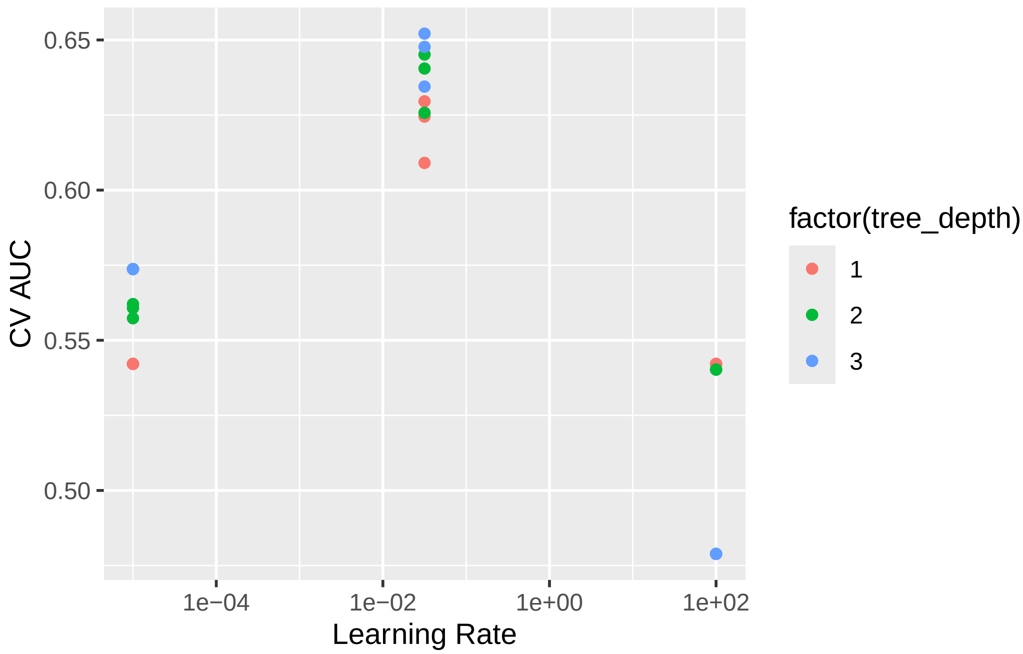
```
# Tuning results
# 5-fold cross-validation using stratification
# A tibble: 5 x 4
  splits               id    .metrics          .notes
  <list>               <chr> <list>            <list>
1 <split [37776/9445]> Fold1 <tibble [54 x 7]> <tibble [0 x 3]>
2 <split [37776/9445]> Fold2 <tibble [54 x 7]> <tibble [0 x 3]>
3 <split [37776/9445]> Fold3 <tibble [54 x 7]> <tibble [0 x 3]>
4 <split [37778/9443]> Fold4 <tibble [54 x 7]> <tibble [0 x 3]>
5 <split [37778/9443]> Fold5 <tibble [54 x 7]> <tibble [0 x 3]>
```

Now, we can terminate our parallel processing:

```r
stopCluster(cl)
```

Visualizing the results:

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



Next, we can review the best-performing models and select the top one:

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

```
# A tibble: 5 x 9
  trees tree_depth learn_rate .metric .estimator  mean     n std_err .config
  <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1   500          3     0.0316 roc_auc binary     0.652     5 0.00271 Preprocess~
2   300          3     0.0316 roc_auc binary     0.648     5 0.00283 Preprocess~
3   500          2     0.0316 roc_auc binary     0.645     5 0.00316 Preprocess~
4   300          2     0.0316 roc_auc binary     0.640     5 0.00339 Preprocess~
5   100          3     0.0316 roc_auc binary     0.634     5 0.00341 Preprocess~
```

Next, we select the best tuning parameters based on ROC AUC:

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

```
# A tibble: 1 x 4
  trees tree_depth learn_rate .config
```

```
  <int>      <int>      <dbl> <chr>
1   500          3     0.0316 Preprocessor1_Model24
```

We finalize our workflow:

```
final_xgb_wf <- xgb_wf %>%
  finalize_workflow(best_xgb)
final_xgb_wf
```

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

-- Preprocessor ----------------------------------------------------------------
8 Recipe Steps

* step_mutate()
* step_mutate()
* step_rm()
* step_impute_median()
* step_impute_mode()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 500
  tree_depth = 3
  learn_rate = 0.0316227766016838

Computational engine: xgboost
```

Now, we can fit the final model on the entire training set and evaluate it on the test set using the `last_fit()` function:

```
final_xgb_fit <- final_xgb_wf %>%
  last_fit(data_split)
final_xgb_fit
```

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

Collect test metrics:

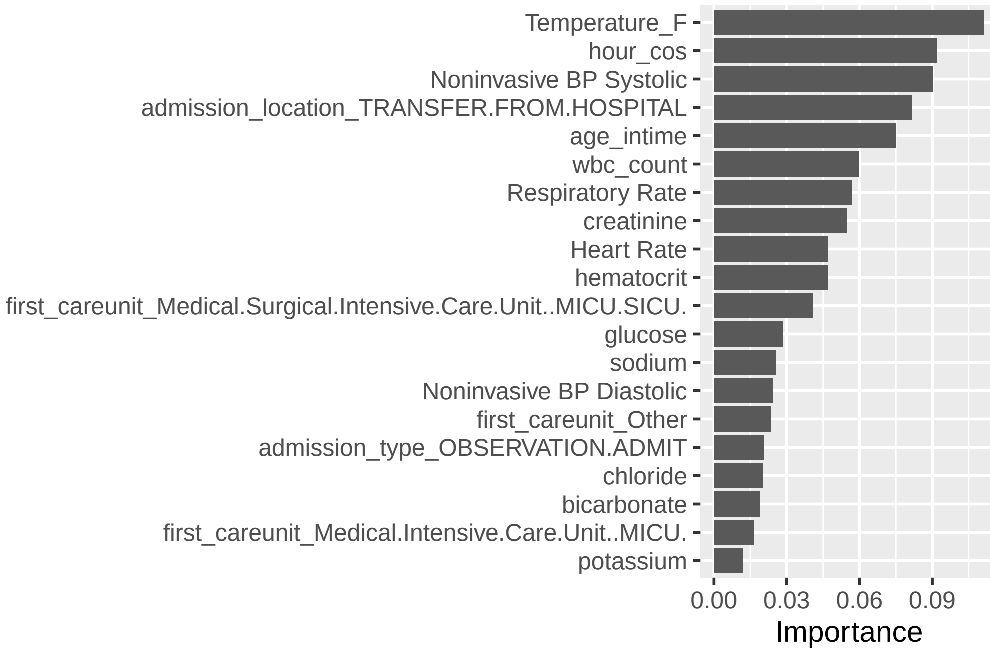
```
final_xgb_fit %>%
  collect_metrics() %>%
  print()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.605 Preprocessor1_Model1
2 roc_auc     binary         0.648 Preprocessor1_Model1
3 brier_class binary         0.233 Preprocessor1_Model1
```

Examining feature importance:

```
final_xgb_model <- final_xgb_fit %>%
  extract_fit_parsnip()

vip(final_xgb_model, num_features = 20)
```



For our Boosting Model, we observe an AUC of 0.6480967 and an accuracy of 0.6053618. Examining feature importance, we observe that there isn't necessarily a sharp 'drop-off' in the variable importance plot, so we will identify the top 10 features based on importance. Namely, we observe that `Temperature_F`, the hour component of admission time `hour_cos`, `Noninvasive BP Systolic`, the indicator for admission location `admission_location_TRANSFER.FROM_HOSPITAL`, `age_intime`, `wbc_count`, `Respiratory Rate`, `creatinine`, `Heart Rate`, and `hematocrit` were among the most important features.

We have now created our three non-stacked models, and we can summarize their performance and feature importance. We can extract the top ten features from each model:

```
# For Logistic Regression:
final_logit_model <- final_logit_fit %>% extract_fit_parsnip()
top10_logit <- vi(final_logit_model) %>%
  arrange(desc(Importance)) %>%
  slice_head(n = 10)
cat("Top 10 features for Logistic Regression Model:\n")
```

```
Top 10 features for Logistic Regression Model:
```

```
print(top10_logit)
```

```
# A tibble: 10 x 3
   Variable                                                     Importance Sign
   <chr>                                                             <dbl> <chr>
 1 hour_cos                                                         0.175  NEG
 2 admission_location_TRANSFER.FROM.HOSPITAL                        0.162  POS
 3 Heart Rate                                                       0.135  POS
 4 first_careunit_Medical.Surgical.Intensive.Care.Unit..MICU.S~     0.128  NEG
 5 Respiratory Rate                                                 0.125  POS
 6 Noninvasive BP Systolic                                          0.100  NEG
 7 admission_type_SURGICAL.SAME.DAY.ADMISSION                       0.0967 NEG
 8 wbc_count                                                        0.0846 POS
 9 hematocrit                                                       0.0846 NEG
10 chloride                                                         0.0779 NEG
```

```r
# For Random Forest:
final_rf_model <- final_rf_fit %>% extract_fit_parsnip()
top10_rf <- vi(final_rf_model) %>%
  arrange(desc(Importance)) %>%
  slice_head(n = 10)
cat("\nTop 10 features for Random Forest Model:\n")
```

```
Top 10 features for Random Forest Model:
```

```r
print(top10_rf)
```

```
# A tibble: 10 x 2
   Variable                Importance
   <chr>                        <dbl>
 1 Noninvasive BP Systolic       447.
 2 wbc_count                     438.
 3 Heart Rate                    434.
 4 Temperature_F                 427.
 5 hematocrit                    419.
 6 Noninvasive BP Diastolic      403.
 7 Respiratory Rate              394.
 8 age_intime                    393.
 9 glucose                       383.
10 hour_cos                      348.
```

```r
# For XGBoost:
final_xgb_model <- final_xgb_fit %>% extract_fit_parsnip()
top10_xgb <- vi(final_xgb_model) %>%
  arrange(desc(Importance)) %>%
  slice_head(n = 10)
cat("\nTop 10 features for XGBoost Model:\n")
```

```
Top 10 features for XGBoost Model:
```

```r
print(top10_xgb)
```

```
# A tibble: 10 x 2
   Variable                                    Importance
   <chr>                                            <dbl>
 1 Temperature_F                                   0.111
 2 hour_cos                                        0.0920
 3 Noninvasive BP Systolic                         0.0902
 4 admission_location_TRANSFER.FROM.HOSPITAL       0.0815
 5 age_intime                                      0.0749
 6 wbc_count                                       0.0596
 7 Respiratory Rate                                0.0567
 8 creatinine                                      0.0548
 9 Heart Rate                                      0.0472
10 hematocrit                                      0.0469
```

It would be interesting to see if there are any features that were commonly identified as important across all three models:

```r
# Compute the intersection of the top 10 variable names from each model
common_features <- Reduce(intersect, list(top10_logit$Variable,
                                          top10_rf$Variable,
                                          top10_xgb$Variable))
cat("Common features across all top 10 lists:\n")
```

```
Common features across all top 10 lists:
```

```r
print(common_features)
```

```
[1] "hour_cos"                 "Heart Rate"
[3] "Respiratory Rate"         "Noninvasive BP Systolic"
[5] "wbc_count"                "hematocrit"
```

We observe that across our Logistic Regression Model with Elastic Net Regularization, our Random Forest Model, and our Boosting Model, these features consistently appeared in the top ten in importance: `hour_cos`, `Heart Rate`, `Respiratory Rate`, `Noninvasive BP Systolic`, `wbc_count`, and `hematocrit`.

Now, we can chronicle the observed performance on the test set from each of these models, in terms of AUC and Accuracy:

```r
logit_metrics <- final_logit_fit %>%
  collect_metrics() %>%
  mutate(Model = "Logistic Regression")

rf_metrics <- final_rf_fit %>%
  collect_metrics() %>%
  mutate(Model = "Random Forest")

xgb_metrics <- final_xgb_fit %>%
  collect_metrics() %>%
  mutate(Model = "XGBoost")

# Combine the metrics and filter for roc_auc and accuracy, using .estimate as the metric valu
performance_metrics <- bind_rows(logit_metrics, rf_metrics, xgb_metrics) %>%
  filter(.metric %in% c("roc_auc", "accuracy")) %>%
  select(Model, .metric, .estimate) %>%
  pivot_wider(names_from = .metric, values_from = .estimate)

performance_metrics
```

```
# A tibble: 3 x 3
  Model               accuracy roc_auc
  <chr>                  <dbl>   <dbl>
1 Logistic Regression    0.582   0.614
2 Random Forest          0.602   0.645
3 XGBoost                0.605   0.648
```

We observe that the Random Forest and XGBoost models had similar overall performance on the test set, while logistic regression performed slightly worse in terms of these two performance metrics. There is certainly a tradeoff between model accuracy and interpretability to be explored, and the decision to report one model over another should be made with careful consideration of intention and audience. For instance, a statistician prioritizing interpretability such that they can communicate generalizable results to clinician colleagues may opt for logistic regression in this instance, despite its relatively lower performance. Meanwhile, a machine learning engineer hoping to develop a prediction algorithm for length of ICU stay based on clinical features may opt to maximize their performance at the expense of interpretability, while still maintaining generalizable results. An individual participating in a hackathon or statistical modeling competition who simply aims to achieve maximal performance on the unseen test data may fully prioritize performance at the expense of gleaning insights from the model, possibly resorting to more "black-box" complex algorithmic approaches that leverage high computational power. Ultimately, our models all predicted length of ICU stay within a similar "neighborhood" of accuracy, and it will be interesting to compare their individual performance with that of a stacked model.

Now, we create our stacked model. To reduce computational complexity and runtime, we will reduce the number of folds we utilize for each individual model, as well as the size of the grid.

```
# Instead of 100 penalty levels, let's only do 5
# For mixture, let's do 3 levels
param_grid_logit_small <- grid_regular(
  penalty(range = c(-4, 0)),  # narrower range
  mixture(range = c(0, 1)),   # full range for mixture
  levels = c(5, 3)
)
```

```
set.seed(203)
folds2 <- vfold_cv(train_data, v = 2, strata = los_long)
cl <- makeCluster(parallel::detectCores() - 1)
registerDoParallel(cl)
clusterEvalQ(cl, {
  library(tidyverse)
  library(tidymodels)
  library(GGally)
  library(gtsummary)
  library(naniar)
  library(lubridate)
  library(glmnet)
  library(vip)
  library(ranger)
})
```

```
[[1]]
```

```
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[2]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[3]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[4]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[5]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
```

```
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[6]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[7]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[8]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[9]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
```

```
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[10]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[11]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"
```

```
# Use control_stack_grid() if you plan to use these results in stacking
# DO NOT have a parallel backend running here
# No cluster setup needed


logit_fit2 <- logit_wf %>%
  tune_grid(
    resamples = folds2,
    grid = param_grid_logit_small,
    metrics = metric_set(roc_auc, accuracy),
    control = control_stack_grid()
  )
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

```
i The workflow being saved contains a recipe, which is 14.87 Mb in i memory. If
this was not intentional, please set the control setting i `save_workflow =
FALSE`.
```

```
stopCluster(cl)
```

We can also do our Random Forest with Reduced Folds and grid size:

```
param_grid_rf_small <- grid_regular(
  trees(range = c(100L, 200L)),  # from 100 to 200 trees
  mtry(range = c(1L, 3L)),       # from 1 to 3 for mtry
  levels = c(3, 3)               # 3 levels each → 3 × 3 = 9 combos
)
param_grid_rf_small
```

```
# A tibble: 9 x 2
  trees  mtry
  <int> <int>
1   100     1
2   150     1
3   200     1
4   100     2
5   150     2
6   200     2
7   100     3
8   150     3
9   200     3
```

```
set.seed(203)
rf_folds2 <- vfold_cv(train_data, v = 2, strata = los_long)

cl <- makeCluster(parallel::detectCores() - 1)
registerDoParallel(cl)
clusterEvalQ(cl, {
  library(tidyverse)
  library(tidymodels)
  library(GGally)
  library(gtsummary)
  library(naniar)
  library(lubridate)
  library(glmnet)
```

```
  library(vip)
  library(ranger)
})
```

```
[[1]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[2]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[3]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[4]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
```

```
[36] "methods"      "base"

[[5]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[6]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[7]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[8]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[9]]
```

```
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[10]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[11]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"
```

```r
rf_fit_small <- rf_wf %>%
  tune_grid(
    resamples = rf_folds2,
    grid = param_grid_rf_small,
    metrics = metric_set(roc_auc, accuracy),
    control = control_stack_grid()
  )
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

```
i The workflow being saved contains a recipe, which is 14.87 Mb in i memory. If
this was not intentional, please set the control setting i `save_workflow =
FALSE`.
```

```
stopCluster(cl)

rf_fit_small
```

```
# Tuning results
# 2-fold cross-validation using stratification
# A tibble: 2 x 5
  splits                 id    .metrics          .notes           .predictions
  <list>                 <chr> <list>            <list>           <list>
1 <split [23610/23611]> Fold1 <tibble [18 x 6]> <tibble [0 x 3]> <tibble>
2 <split [23611/23610]> Fold2 <tibble [18 x 6]> <tibble [0 x 3]> <tibble>
```

as well as XGBoost with Reduced Folds and grid size:

```
param_grid_xgb_small <- grid_regular(
  trees(range = c(100L, 300L)),      # from 100 to 300 trees, 3 levels
  tree_depth(range = c(1L, 3L)),     # from depth 1 to 3, 2 or 3 levels
  learn_rate(range = c(-3, -2), trans = log10_trans()),
  # e.g., 10^-3 to 10^-2 → 2 levels
  levels = c(3, 2, 2)    # 3 × 2 × 2 = 12 total combos
)
param_grid_xgb_small
```

```
# A tibble: 12 x 3
   trees tree_depth learn_rate
   <int>      <int>      <dbl>
 1   100          1      0.001
 2   200          1      0.001
 3   300          1      0.001
 4   100          3      0.001
 5   200          3      0.001
 6   300          3      0.001
 7   100          1      0.01
 8   200          1      0.01
 9   300          1      0.01
10   100          3      0.01
11   200          3      0.01
12   300          3      0.01
```

```r
set.seed(203)
xgb_folds2 <- vfold_cv(train_data, v = 2, strata = los_long)

cl <- makeCluster(parallel::detectCores() - 1)
registerDoParallel(cl)
clusterEvalQ(cl, {
  library(tidyverse)
  library(tidymodels)
  library(GGally)
  library(gtsummary)
  library(naniar)
  library(lubridate)
  library(glmnet)
  library(vip)
  library(ranger)
})
```

```
[[1]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[2]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[3]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
```

```
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[4]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[5]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[6]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[7]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
```

```
[36] "methods"      "base"

[[8]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[9]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[10]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[11]]
 [1] "ranger"       "vip"          "glmnet"       "Matrix"       "naniar"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"
```

```r
xgb_fit_small <- xgb_wf %>%
  tune_grid(
    resamples = xgb_folds2,
    grid = param_grid_xgb_small,
    metrics = metric_set(roc_auc, accuracy),
    control = control_stack_grid()
  )
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

```
i The workflow being saved contains a recipe, which is 14.87 Mb in i memory. If
this was not intentional, please set the control setting i `save_workflow =
FALSE`.
```

```r
stopCluster(cl)

xgb_fit_small
```

```
# Tuning results
# 2-fold cross-validation using stratification
# A tibble: 2 x 5
  splits                id    .metrics          .notes           .predictions
  <list>                <chr> <list>            <list>           <list>
1 <split [23610/23611]> Fold1 <tibble [24 x 7]> <tibble [0 x 3]> <tibble>
2 <split [23611/23610]> Fold2 <tibble [24 x 7]> <tibble [0 x 3]> <tibble>
```

```r
# Combine all resample metrics data frames
metrics_combined <- map_dfr(logit_fit2$.metrics, ~ .x)

# Count the distinct .config values
num_configs_filtered <- metrics_combined %>%
  distinct(.config) %>%
  nrow()

cat("Number of candidate configurations in logit_fit_filtered:",
    num_configs_filtered, "\n")
```

```
Number of candidate configurations in logit_fit_filtered: 15
```

Lastly, we will create the stacked model:

```r
cl <- makeCluster(detectCores() - 1)
registerDoParallel(cl)
clusterEvalQ(cl, {
  library(tidyverse)
  library(tidymodels)
  library(GGally)
  library(gtsummary)
  library(glmnet)
  library(vip)
  library(ranger)
  library(stacks)
})
```

```
[[1]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"      
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"   
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"   
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"  
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"       
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"   
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"    
[36] "methods"      "base"        

[[2]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"      
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"   
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"   
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"  
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"       
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"   
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"    
[36] "methods"      "base"        

[[3]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"      
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"   
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"   
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"  
```

```
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[4]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[5]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[6]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[7]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
```

```
[36] "methods"      "base"

[[8]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[9]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[10]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"

[[11]]
 [1] "stacks"       "ranger"       "vip"          "glmnet"       "Matrix"
 [6] "gtsummary"    "GGally"       "yardstick"    "workflowsets" "workflows"
[11] "tune"         "rsample"      "recipes"      "parsnip"      "modeldata"
[16] "infer"        "dials"        "scales"       "broom"        "tidymodels"
[21] "lubridate"    "forcats"      "stringr"      "dplyr"        "purrr"
[26] "readr"        "tidyr"        "tibble"       "ggplot2"      "tidyverse"
[31] "stats"        "graphics"     "grDevices"    "utils"        "datasets"
[36] "methods"      "base"
```

Here, we set the penalty term based on autoplot() after fitting the stacked model to our data.

```r
# Build the stacking ensemble using the candidate tuning results from your three models.
icu_model_stack <- stacks() %>%
  # Add the candidates from each model. (These objects should have been tuned with control =
  add_candidates(logit_fit2) %>%
  add_candidates(rf_fit_small) %>%
  add_candidates(xgb_fit_small) %>%
  # Blend predictions: this fits a meta-model that combines the candidates.
  blend_predictions(
    penalty = 1e-2,
    metrics = metric_set(roc_auc)
  ) %>%
  # Fit the ensemble members (only those with nonzero stacking coefficients)
  fit_members()
```

```
Warning: Predictions from 8 candidates were identical to those from existing candidates
and were removed from the data stack.
```

```
Warning: The `...` are not used in this function but one or more arguments were
passed: 'metrics'
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

```r
# Display the stacked model object
icu_model_stack
```

```
-- A stacked ensemble model -------------------------------------
```

```
Out of 32 possible candidate members, the ensemble retained 5.
```

```
Penalty: 0.01.
```

```
Mixture: 1.
```

```
The 5 highest weighted member classes are:
```

```
# A tibble: 5 x 3
  member                          type        weight
  <chr>                           <chr>        <dbl>
1 .pred_TRUE_rf_fit_small_1_9   rand_forest 3.74
2 .pred_TRUE_rf_fit_small_1_8   rand_forest 1.70
3 .pred_TRUE_rf_fit_small_1_7   rand_forest 0.885
4 .pred_TRUE_rf_fit_small_1_6   rand_forest 0.866
5 .pred_TRUE_xgb_fit_small_1_12 boost_tree  0.0276
```

```
stopCluster(cl)
```

Now that we have our model, we fit it on the training data and evaluate it on the test set, just as we did for our individual models:

```
# Get probability predictions from the stacked model
stacked_preds <- predict(icu_model_stack, new_data = test_data,
                         type = "prob") %>%
  bind_cols(test_data)

# Convert probabilities into predicted classes (threshold at 0.5)
stacked_preds <- stacked_preds %>%
  mutate(.pred_class = if_else(.pred_TRUE >= 0.5, "TRUE", "FALSE") %>%
           factor(levels = c("FALSE", "TRUE")))
# Compute ROC AUC
stacked_auc <- stacked_preds %>%
  roc_auc(truth = los_long, .pred_TRUE) %>%
  mutate(Model = "Stacked Ensemble")

# Compute Accuracy
stacked_accuracy <- stacked_preds %>%
  accuracy(truth = los_long, estimate = .pred_class) %>%
  mutate(Model = "Stacked Ensemble")
```

Now we incorporate the stacked model performance into our performance metrics:

```
stacked_wide <- bind_rows(stacked_auc, stacked_accuracy) %>%
  # pivot them so we get columns "accuracy" and "roc_auc"
  tidyr::pivot_wider(
    names_from = .metric,
    values_from = .estimate
  )
final_model_performance <- bind_rows(
```

```
  performance_metrics,
  stacked_wide
)

final_model_performance
```

```
# A tibble: 4 x 4
  Model               accuracy roc_auc .estimator
  <chr>                  <dbl>   <dbl> <chr>
1 Logistic Regression    0.582   0.614 <NA>
2 Random Forest          0.602   0.645 <NA>
3 XGBoost                0.605   0.648 <NA>
4 Stacked Ensemble       0.606   0.353 binary
```

We observe an AUC of 0.3523246 (our lowest by a considerable margin for all models), and an accuracy on par with our other models at 0.6068865. The likely reason for this low AUC is the lack of diversity in our candidate configurations. Due to exorbitant runtimes and issues with R crashes, I opted to limit grid size as well as cross validation. As a result, We miss out on potentially crucial diversity in our model selection process. Additionally, our ensemble model may have discarded more diverse base models based on the training set that would have generalized well to the test set. Lastly, it is possible that our stacked model overfit the data, and it is an unlikely but existent possibility that there are systematic differences between the test and training sets. I intend on diagnosing these issues over time to iterate upon the stacked model and achieve a more satisfactory AUC. Still, the discrepancy in AUC values is undoubtedly insightful and motivating for future ML-focused endeavors.

Overall, the three individual models performed similarly on training and test data, with XGBoost marginally coming out on top in both metrics. The stacked ensemble model did not seem to add any predictive accuracy to our task, and in fact seems to have compromised ability to perform well at our classification task. Accuracy is just barely larger than that of XGBoost, and AUC is by far the lowest. This is an important consideration for us, considering that AUC is robust to class imbalances. It would be interesting to determine whether model misclassification appears to be random or systematic in some way. I plan to explore these items further in the coming weeks.