Biostat 203B Homework 5

Due Mar 20 @ 11:59PM

AUTHOR
Julie Lee, 806409381

Loading all necessary packages

```
library(rsample)
library(tidymodels)
```

```
── Attaching packages ─────────────────────────────────────── tidymodels 1.2.0 ──

✔ broom        1.0.7     ✔ recipes      1.1.0
✔ dials        1.3.0     ✔ tibble       3.2.1
✔ dplyr        1.1.4     ✔ tidyr        1.3.1
✔ ggplot2      3.5.1     ✔ tune         1.2.1
✔ infer        1.0.7     ✔ workflows    1.1.4
✔ modeldata    1.4.0     ✔ workflowsets 1.1.0
✔ parsnip      1.2.1     ✔ yardstick    1.3.1
✔ purrr        1.0.2

── Conflicts ────────────────────────────────────────── tidymodels_conflicts() ──
✖ purrr::discard() masks scales::discard()
✖ dplyr::filter()  masks stats::filter()
✖ dplyr::lag()     masks stats::lag()
✖ recipes::step()  masks stats::step()
• Learn how to get started at https://www.tidymodels.org/start/
```

```
library(xgboost)
```

```
Attaching package: 'xgboost'

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

    slice
```

```
library(kernlab)
```

```
Attaching package: 'kernlab'

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

    cross
```

```
The following object is masked from 'package:ggplot2':

    alpha

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

    alpha
```

```
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
```

```
library(xgboost)
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
```

```
library(pROC)
```

```
Type 'citation("pROC")' for a citation.


Attaching package: 'pROC'

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

    cov, smooth, var
```

```
library(corrr)
library(ggfocus)
library(caret)
```

```
Loading required package: lattice


Attaching package: 'caret'

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

    precision, recall, sensitivity, specificity

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

    lift
```

```
library(e1071)
```

```

Attaching package: 'e1071'

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

    tune

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

    tune

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

    permutations
```

```
library(randomForest)
```

```
randomForest 4.7-1.2

Type rfNews() to see new features/changes/bug fixes.


Attaching package: 'randomForest'

The following object is masked from 'package:ggplot2':

    margin

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

```
    combine
```

```
library(ranger)
```

```
Attaching package: 'ranger'

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

    importance
```

```
library(SuperLearner)
```

```
Loading required package: nnls

Loading required package: gam

Loading required package: splines

Loaded gam 1.22-5

Super Learner

Version: 2.0-29

Package created on 2024-02-06
```

```
library(caretEnsemble)
library(dplyr)
library(purrr)
library(ggplot2)
library(gt)
```

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.

1. Data preprocessing and feature engineering.

```
mimiciv_icu_cohort <- readRDS("../hw4/mimiciv_shiny/mimic_icu_cohort.rds")
```

Data Preprocessing and Feature Engineering Explanation:

We began our data preprocessing by creating a new binary outcome variable, los_long, which is set to 1 (TRUE) if a patient's length of stay (los) is greater than or equal to two days and 0 (FALSE) otherwise. The dataset was then filtered to remove any patients with missing los values. To prepare the features for modeling, we selected 18 variables, including patient demographics (gender, age at ICU admission, marital status, and race), ICU admission details (first care unit), the last recorded laboratory measurements before ICU admission, and the first recorded vital signs during the ICU stay. To handle missing data, we applied mode imputation (replacing missing categorical values with the most frequent category) for variables such as gender, marital status, race, and first care unit. For numerical variables, we applied median imputation, filling in missing values with the median of each respective column. For consistency and to optimize model performance, all continuous variables were standardized (mean = 0, standard deviation = 1). Finally, categorical variables were converted into factors, ensuring compatibility with machine learning models. To maintain data integrity and avoid data leakage, we only used information available at the time of ICU admission. Additionally, we included the variables subject_id, hadm_id, and stay_id in the model but excluded them as predictive features in later steps.

```
mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  filter(!is.na(los)) %>%
  mutate(
    los_long = as.factor(as.integer(los >= 2))
  ) %>%
  select(
    subject_id, hadm_id, stay_id, los_long, gender, age_at_intime,
    marital_status, race, first_careunit, heart_rate,
    non_invasive_blood_pressure_systolic,
    non_invasive_blood_pressure_diastolic, respiratory_rate,
    temperature_fahrenheit, bicarbonate, glucose, potassium, sodium,
    chloride, creatinine, hematocrit, wbc
  ) %>%

  mutate(
    across(
      c(gender, marital_status, race, first_careunit),
      ~ replace_na(.x, names(sort(table(.x), decreasing = TRUE))[1])
    ),
    across(c(gender, marital_status, race, first_careunit), factor)
  ) %>%

  mutate(
    across(
      c(age_at_intime, heart_rate, non_invasive_blood_pressure_systolic,
        non_invasive_blood_pressure_diastolic, respiratory_rate,
        temperature_fahrenheit, bicarbonate, glucose, potassium, sodium,
        chloride, creatinine, hematocrit, wbc),
      ~ if_else(is.na(.x), median(.x, na.rm = TRUE), .x)
    )
  ) %>%

  mutate(
```

```
    across(
      c(age_at_intime, heart_rate, non_invasive_blood_pressure_systolic,
        non_invasive_blood_pressure_diastolic, respiratory_rate,
        temperature_fahrenheit, bicarbonate, glucose, potassium, sodium,
        chloride, creatinine, hematocrit, wbc),
      ~ as.numeric(scale(.x))
    )
  )

mimiciv_icu_cohort
```

```
# A tibble: 94,444 × 22
   subject_id hadm_id stay_id los_long gender age_at_intime marital_status race
        <int>   <int>   <int> <fct>    <fct>          <dbl> <fct>          <fct>
 1   10000032  2.91e7  3.96e7 0        F             -0.764 WIDOWED        WHITE
 2   10000690  2.59e7  3.71e7 1        F              1.27  WIDOWED        WHITE
 3   10000980  2.69e7  3.98e7 0        F              0.670 MARRIED        BLAC…
 4   10001217  2.46e7  3.71e7 0        F             -0.585 MARRIED        WHITE
 5   10001217  2.77e7  3.46e7 0        F             -0.585 MARRIED        WHITE
 6   10001725  2.56e7  3.12e7 0        F             -1.12  MARRIED        WHITE
 7   10001843  2.61e7  3.97e7 0        M              0.670 SINGLE         WHITE
 8   10001884  2.62e7  3.75e7 1        F              0.730 MARRIED        BLAC…
 9   10002013  2.36e7  3.91e7 0        F             -0.465 SINGLE         OTHER
10   10002114  2.78e7  3.47e7 1        M             -0.525 MARRIED        UNKN…
# ℹ 94,434 more rows
# ℹ 14 more variables: first_careunit <fct>, heart_rate <dbl>,
#   non_invasive_blood_pressure_systolic <dbl>,
#   non_invasive_blood_pressure_diastolic <dbl>, respiratory_rate <dbl>,
#   temperature_fahrenheit <dbl>, bicarbonate <dbl>, glucose <dbl>,
#   potassium <dbl>, sodium <dbl>, chloride <dbl>, creatinine <dbl>,
#   hematocrit <dbl>, wbc <dbl>
```

More Preprocessing: Identifying the Least significant Categorical Variable in the Dataset

We further preprocess our dataset by applying the Chi-Square test with Monte Carlo simulation to assess the statistical significance of categorical predictors in relation to ICU length of stay (los_long). We implement a specific function to compute the Chi-Square test with 100,000 Monte Carlo simulations. This test is conducted on the categorical variables gender, marital status, race, and first care unit, with the resulting p-values stored in a dataframe. Based on the analysis, we find that all categorical predictors (marital_status, race, first_careunit, and gender) exhibit statistical significance, and as a result, we decide to retain all of these predictors in our final dataset.

```
chi_square_test <- function(predictor, target) {
  tbl <- table(predictor, target)
  test <- chisq.test(tbl, simulate.p.value = TRUE, B = 100000)
  return(test$p.value)
}

categorical_vars <- mimiciv_icu_cohort %>%
  select(gender, marital_status, race, first_careunit)
```

```
chi_square_results <- map_dbl(
  categorical_vars,
  chi_square_test,
  target = mimiciv_icu_cohort$los_long
)

chi_square_results <- data.frame(
  predictor = names(chi_square_results),
  p_value = chi_square_results
)

chi_square_results <- chi_square_results %>%
  arrange(p_value)

print(chi_square_results)
```

```
                   predictor     p_value
marital_status marital_status 9.99990e-06
race                   race 9.99990e-06
first_careunit first_careunit 9.99990e-06
gender               gender 5.99994e-05
```

More Preprocessing: Identifying the Least Significant Categorical Variables in the Data Set

To further preprocess our data, we perform an independent t-test to assess the statistical significance of numerical predictors in relation to ICU length of stay (los_long). A function is defined to conduct two-sample t-tests, comparing the distributions of each numerical variable between patients with ICU stays of less than two days (los_long = 0) and those with longer stays (los_long = 1). The test is applied to numerical features, including age at ICU admission, heart rate, blood pressure, respiratory rate, temperature, and various lab measurements. Based on the p-values obtained, we identify glucose, potassium, bicarbonate, and diastolic blood pressure as not statistically significant ($p > 0.05$). Consequently, these four variables are removed from the final mimiciv_icu_cohort dataset to retain only the most relevant numerical predictors for further modeling. The first few rows of the final dataset that will be utilized for machine learning is shown below. Note that there are a total of 14 predictors used in our final model. We will now conduct 3 different machine learning approaches on this dataset.

```
t_test <- function(predictor, target) {
  group1 <- predictor[target == 0]
  group2 <- predictor[target == 1]
  test <- t.test(group1, group2)
  return(test$p.value)
}

numerical_vars <- mimiciv_icu_cohort %>%
  select(
    age_at_intime, heart_rate, non_invasive_blood_pressure_systolic,
    non_invasive_blood_pressure_diastolic, respiratory_rate,
    temperature_fahrenheit, bicarbonate, glucose, potassium, sodium,
    chloride, creatinine, hematocrit, wbc
```

```
  )

t_test_results <- map_dbl(
  numerical_vars,
  t_test,
  target = as.numeric(as.character(mimiciv_icu_cohort$los_long))
)

t_test_results <- data.frame(
  predictor = names(t_test_results),
  p_value = unname(t_test_results)
)


t_test_results <- t_test_results %>%
  arrange(p_value)

print(t_test_results)
```

```
                              predictor       p_value
1                       respiratory_rate 2.310112e-109
2                             heart_rate 9.842488e-102
3                             hematocrit  7.580349e-49
4                                    wbc  9.265108e-35
5                           age_at_intime  1.332422e-31
6                                chloride  1.956335e-16
7                              creatinine  1.537788e-11
8                                  sodium  2.148440e-04
9                  temperature_fahrenheit  3.933961e-02
10  non_invasive_blood_pressure_systolic  3.991551e-02
11                                glucose  2.007153e-01
12                              potassium  2.336844e-01
13                            bicarbonate  4.173911e-01
14 non_invasive_blood_pressure_diastolic  9.255369e-01
```

```
mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  select(
    -glucose, -potassium, -bicarbonate,
    -non_invasive_blood_pressure_diastolic
  )

head(mimiciv_icu_cohort)
```

```
# A tibble: 6 × 18
  subject_id  hadm_id stay_id los_long gender age_at_intime marital_status race
       <int>    <int>   <int> <fct>    <fct>          <dbl> <fct>          <fct>
1   10000032 29079034  3.96e7 0        F             -0.764 WIDOWED        WHITE
2   10000690 25860671  3.71e7 1        F              1.27  WIDOWED        WHITE
3   10000980 26913865  3.98e7 0        F              0.670 MARRIED        BLAC…
4   10001217 24597018  3.71e7 0        F             -0.585 MARRIED        WHITE
```

```
5   10001217 27703517  3.46e7 0        F             -0.585 MARRIED        WHITE
6   10001725 25563031  3.12e7 0        F             -1.12  MARRIED        WHITE
# ℹ 10 more variables: first_careunit <fct>, heart_rate <dbl>,
#   non_invasive_blood_pressure_systolic <dbl>, respiratory_rate <dbl>,
#   temperature_fahrenheit <dbl>, sodium <dbl>, chloride <dbl>,
#   creatinine <dbl>, hematocrit <dbl>, wbc <dbl>
```

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

The first few rows of the train_data and test_data are shown below. In addition, we are able to confirm that there are 47221 entries in the traning set while there are 47223 entries in the testing set.

```
set.seed(203)

mimiciv_icu_cohort <- mimiciv_icu_cohort %>%
  arrange(subject_id, hadm_id, stay_id)

data_split <- initial_split(mimiciv_icu_cohort, prop = 0.5, strata = los_long)

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

train_data <- train_data %>%
  select(-subject_id, -hadm_id, -stay_id)

test_data <- test_data %>%
  select(-subject_id, -hadm_id, -stay_id)

head(train_data)
```

```
# A tibble: 6 × 15
  los_long gender age_at_intime marital_status race    first_careunit heart_rate
  <fct>    <fct>          <dbl> <fct>          <fct>   <fct>               <dbl>
1 0        F              0.670 MARRIED        BLACK/… Medical Inten…    -0.593
2 0        F             -0.585 MARRIED        WHITE   Surgical Inte…    -0.109
3 0        F             -0.585 MARRIED        WHITE   Surgical Inte…    -0.431
4 0        M              0.670 SINGLE         WHITE   Medical/Surgi…     1.75
5 0        M             -0.525 MARRIED        UNABLE… Cardiac Vascu…     0.133
6 0        F             -0.824 SINGLE         BLACK/… Medical Inten…    -0.0606
# ℹ 8 more variables: non_invasive_blood_pressure_systolic <dbl>,
#   respiratory_rate <dbl>, temperature_fahrenheit <dbl>, sodium <dbl>,
#   chloride <dbl>, creatinine <dbl>, hematocrit <dbl>, wbc <dbl>
```

```
head(test_data)
```

```
# A tibble: 6 × 15
  los_long gender age_at_intime marital_status race    first_careunit heart_rate
  <fct>    <fct>          <dbl> <fct>          <fct>   <fct>               <dbl>
```

```
1 0        F             -0.764 WIDOWED        WHITE   Medical Inten…      0.133
2 0        F             -1.12  MARRIED        WHITE   Medical/Surgi…     -0.109
3 0        F             -0.465 SINGLE         OTHER   Cardiac Vascu…     -0.399
4 1        M             -0.525 MARRIED        UNKNOWN Coronary Care…      1.08
5 0        F              1.09  MARRIED        WHITE   Medical Inten…      1.25
6 1        F              1.03  MARRIED        WHITE   Medical/Surgi…      0.133
# ℹ 8 more variables: non_invasive_blood_pressure_systolic <dbl>,
#   respiratory_rate <dbl>, temperature_fahrenheit <dbl>, sodium <dbl>,
#   chloride <dbl>, creatinine <dbl>, hematocrit <dbl>, wbc <dbl>
```

```
nrow(train_data)
```

```
[1] 47221
```

```
nrow(test_data)
```

```
[1] 47223
```

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

Machine Learning Approach 1: Logistic Regression with e-net regularization

Analysis: Our first machine learning approach employs logistic regression with elastic net (ENet) regularization to predict ICU length of stay, optimizing hyperparameters through cross-validation. We started with more data preprocessing suitable for running a logistic regression model by converting categorical variables into dummy variables and splitting the dataset into training and testing sets. The model was then trained over a grid of alpha (L1 ratio) and lambda (regularization strength) values, with the best hyperparameters (alpha = 0.375, lambda = 0.0013) selected based on the lowest cross-validation error (1.3478). It is important to note that elastic net regularization combines L1 (Lasso) for feature selection and L2 (Ridge) to reduce multicollinearity.

In addition, hyperparameter tuning was conducted using a grid search strategy over a predefined range of alpha and lambda values. The alpha parameter was varied from 0 to 1 in 25 evenly spaced increments. The lambda parameter was varied across a logarithmic scale from 10^{-4} to 10^{1} (0.0001 to 10) to control the strength of regularization. We also used 5-fold cross-validation during tuning, where the dataset was divided into five subsets, and the model was trained and validated iteratively on different splits. The combination of alpha and lambda that produced the lowest cross-validation error was selected as the optimal set of hyperparameters as shown in the Kaggle table.

Results: The confusion matrix reveals a moderate prediction imbalance, with the model correctly classifying 15,582 short ICU stays but misclassifying 11,452 as long stays, while among actual long stays, 11,717 were correctly identified and 8,472 were misclassified as short stays. The test accuracy of 57.8% and AUC score of 0.606 indicate marginal predictive performance, suggesting the model struggles to distinguish between short and long ICU stays effectively - we will be exploring other machine learning approaches shortly. The ROC curve further illustrates weak discrimination, with the model performing only slightly better than random guessing. We are able to deduce

that logistic regression may not be the best machine learning approach because the linear assumption of logistic regression makes it less effective for capturing complex interactions and also due to its sensitivity to outliers.

```
set.seed(203)
train_sampled <- train_data %>% sample_frac(1.0)

train_x <- model.matrix(los_long ~ . - 1, data = train_sampled)
train_y <- as.numeric(as.character(train_sampled$los_long))

test_x <- model.matrix(los_long ~ . - 1, data = test_data)
test_y <- as.numeric(as.character(test_data$los_long))

registerDoParallel(cores = parallel::detectCores() - 1)
set.seed(123)

alpha_values <- seq(0, 1, length.out = 25)
lambda_values <- 10^seq(-4, 1, length.out = 10)

tuning_grid <- expand.grid(alpha = alpha_values, lambda = lambda_values)
cv_results <- list()


for (i in 1:nrow(tuning_grid)) {
  alpha_i <- tuning_grid$alpha[i]
  lambda_i <- tuning_grid$lambda[i]

  cv_fit <- cv.glmnet(
    x = train_x, y = train_y, family = "binomial",
    alpha = alpha_i, lambda = lambda_values,
    nfolds = 5, parallel = TRUE
  )

  cv_results[[i]] <- list(
    alpha = alpha_i, lambda = cv_fit$lambda.min,
    error = min(cv_fit$cvm)
  )
}

cv_results_df <- do.call(rbind, lapply(cv_results, as.data.frame))

best_params <- cv_results_df[which.min(cv_results_df$error), ]
best_alpha <- best_params$alpha
best_lambda <- best_params$lambda

best_params %>%
  gt() %>%
  tab_header(title = "Best Hyperparameters for Logistic Regression") %>%
  fmt_number(columns = c(alpha, lambda, error), decimals = 4) %>%
  cols_label(
    alpha = "Best Alpha",
```

```
    lambda = "Best Lambda",
    error = "CV Error"
  )
```

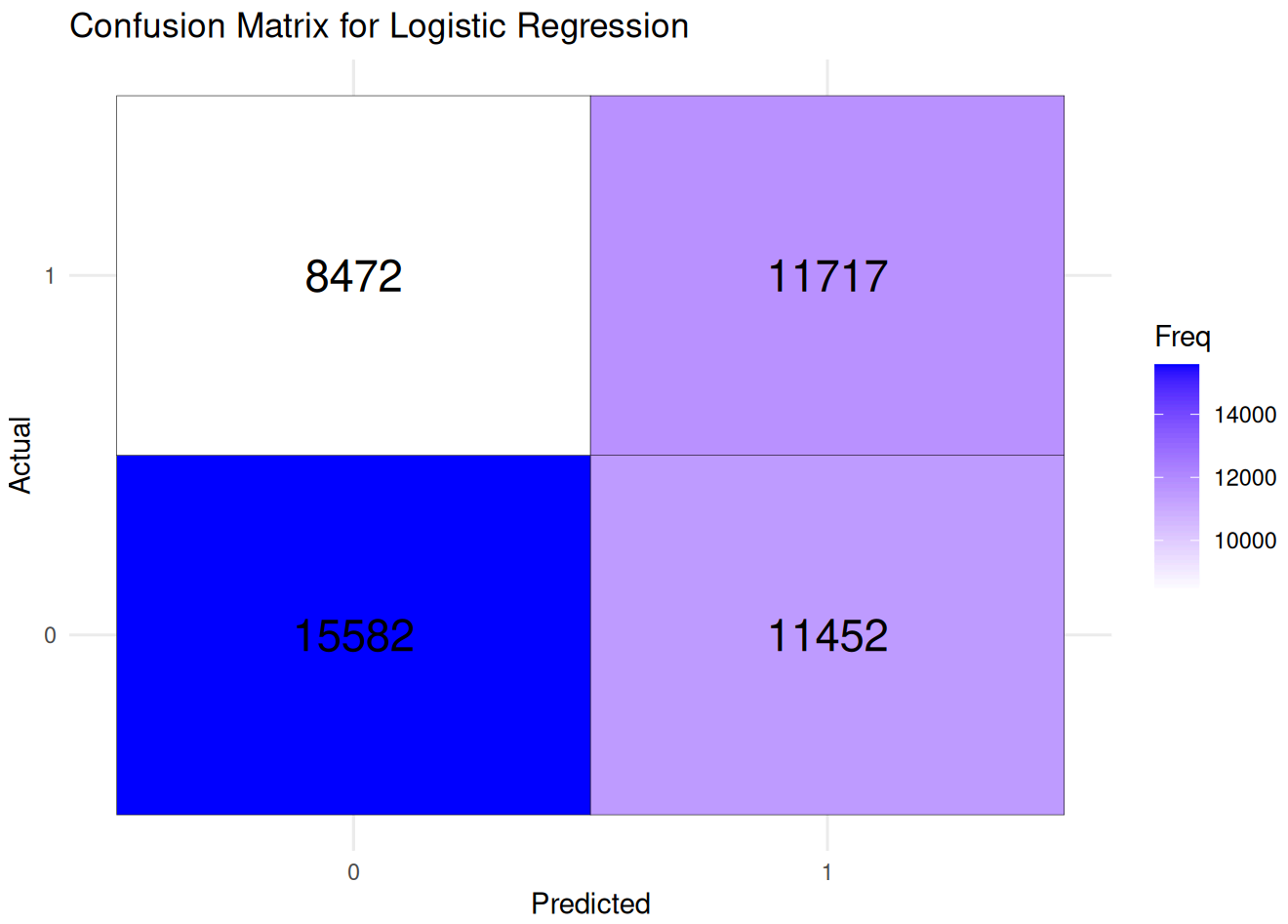
Best Hyperparameters for Logistic Regression		
Best Alpha	Best Lambda	CV Error
0.3750	0.0013	1.3478

```
final_model <- glmnet(
  x = train_x, y = train_y, family = "binomial",
  alpha = best_alpha, lambda = best_lambda
)

test_predictions <- predict(final_model, newx = test_x, type = "response")
test_pred_labels <- ifelse(test_predictions > 0.5, 1, 0)

conf_matrix <- table(Predicted = test_pred_labels, Actual = test_y)
accuracy_logit <- sum(diag(conf_matrix)) / sum(conf_matrix)

roc_curve <- roc(test_y, test_predictions)
```

```
Setting levels: control = 0, case = 1

Warning in roc.default(test_y, test_predictions): Deprecated use a matrix as
predictor. Unexpected results may be produced, please pass a numeric vector.

Setting direction: controls < cases
```

```
auc_score_logit <- auc(roc_curve)

conf_matrix_df <- as.data.frame.matrix(conf_matrix)

conf_matrix_long <- conf_matrix_df %>%
  rownames_to_column(var = "Actual") %>%
  pivot_longer(cols = -Actual, names_to = "Predicted", values_to = "Freq")

ggplot(conf_matrix_long, aes(x = Predicted, y = Actual, fill = Freq)) +
  geom_tile(color = "black") +
  scale_fill_gradient(low = "white", high = "blue") +
  geom_text(aes(label = Freq), color = "black", size = 6) +
  labs(
    title = "Confusion Matrix for Logistic Regression",
    x = "Predicted",
    y = "Actual"
```

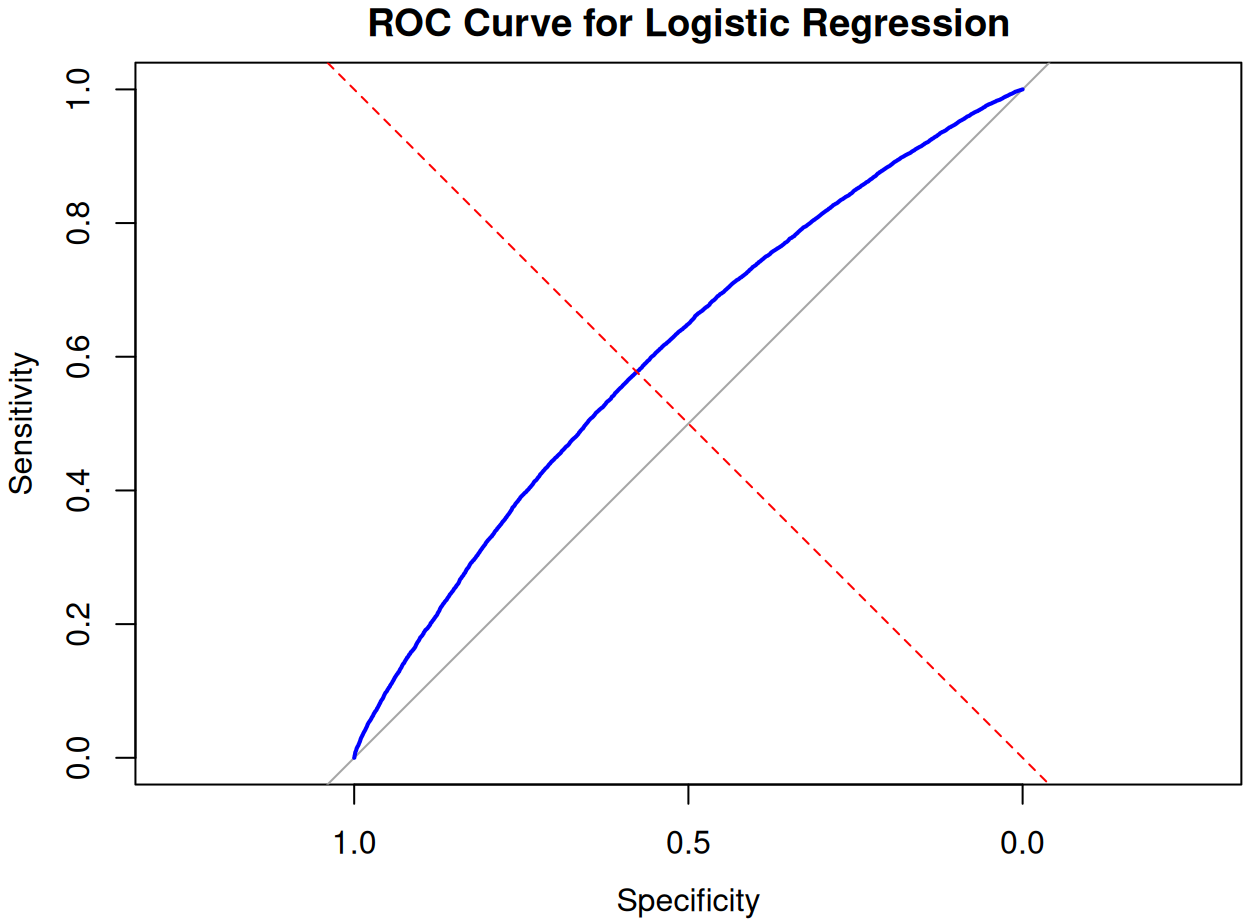
```
  ) +
  theme_minimal()
```




```
results_table <- data.frame(
  Metric = c("Test Accuracy", "AUC Score"),
  Value = c(round(accuracy_logit, 3), round(auc_score_logit, 3))
)

results_table %>%
  gt() %>%
  tab_header(title = "Model Performance Metrics") %>%
  fmt_number(columns = Value, decimals = 3)
```

Model Performance Metrics	
Metric	Value
Test Accuracy	0.578
AUC Score	0.606

```
plot(roc_curve, col = "blue", main = "ROC Curve for Logistic Regression")
abline(a = 0, b = 1, lty = 2, col = "red")
```



Machine Learning Approach 2: XG Boosting Classification Model

Analysis: Our second machine learning approach implements an XGBoost classification model to predict ICU length of stay, utilizing hyperparameter tuning and cross-validation for optimization. Again, we preprocess our mimiciv_icu_cohort dataset where categorical variables are converted into dummy variables, and the response variable (los_long) is formatted as a binary numeric variable (0 or 1). We then transform our data into XGBoost's DMatrix format to enhance computational efficiency. We implement hyperparameter tuning is conducted through a grid search, testing key parameters such as learning rate (eta), tree depth, minimum child weight, subsampling, and column sampling across 21 randomly sampled combinations. Finally, each hyperparameter combination is evaluated with a 5-fold cross-validation strategy, which identifies the best model based on the highest AUC score and optimal iteration count, with early stopping is applied to prevent overfitting.

Results: The results from the XGBoost model showcase a moderate predictive performance in classifying ICU length of stay, showing improved performance in classification compared to logistic regression with elastic net (ENet) regularization. The optimal hyperparameters, selected through cross-validation, include a learning rate of 0.155, a maximum tree depth of 5, and full feature and instance sampling (subsample = 1, colsample_bytree = 1).

The model achieved an AUC score of 0.641, demonstrating its ability to moderately differentiate between short and long ICU stays—an improvement over logistic regression, which had a lower AUC. However, the test accuracy of 60.1% suggests relatively low classification precision, which was likely influenced by class imbalances. We are able to see that there were also significant misclassifications done as seen in the confusion matrix, with 14,419 false positives and 13,974 false negatives. The ROC curve further supports these findings, indicating that while XGBoost outperforms random guessing and logistic regression, its discriminative power remains limited - we will use our last machine learning approach which is random forest to see if it will improve our model.

```
set.seed(203)

train_sampled <- train_data %>% sample_frac(1.0)
train_x <- model.matrix(los_long ~ . - 1, data = train_sampled)
test_x <- model.matrix(los_long ~ . - 1, data = test_data)

train_y <- as.numeric(as.character(train_sampled$los_long))
test_y <- as.numeric(as.character(test_data$los_long))

dtrain <- xgb.DMatrix(data = train_x, label = train_y)
dtest <- xgb.DMatrix(data = test_x, label = test_y)

registerDoParallel(cores = parallel::detectCores() - 1)

set.seed(123)
tuning_grid <- expand.grid(
  eta = seq(0.01, 0.3, length.out = 3),
  max_depth = c(3, 5, 7),
  min_child_weight = c(1, 3, 5),
  subsample = c(0.6, 0.8, 1),
  colsample_bytree = c(0.6, 0.8, 1)
)

tuning_grid <- tuning_grid[sample(1:nrow(tuning_grid), 21), ]

cv_results <- list()

for (i in 1:nrow(tuning_grid)) {
  params <- list(
    booster = "gbtree",
    objective = "binary:logistic",
    eval_metric = "auc",
    eta = tuning_grid$eta[i],
    max_depth = tuning_grid$max_depth[i],
    min_child_weight = tuning_grid$min_child_weight[i],
    subsample = tuning_grid$subsample[i],
    colsample_bytree = tuning_grid$colsample_bytree[i]
  )

  cv_model <- xgb.cv(params = params, data = dtrain, nrounds = 100,
                     nfold = 5, early_stopping_rounds = 10, maximize = TRUE,
```

```
                      verbose = FALSE, parallel = TRUE)

  best_iteration <- cv_model$best_iteration
  best_auc <- max(cv_model$evaluation_log$test_auc_mean)

  cv_results[[i]] <- c(tuning_grid[i, ], best_iteration = best_iteration,
                       AUC = best_auc)
}

cv_results_df <- do.call(rbind, lapply(cv_results, as.data.frame))

best_params <- cv_results_df[which.max(cv_results_df$AUC), ]

best_params %>%
  gt() %>%
  tab_header(title = "Best Hyperparameters for XGBoost") %>%
  fmt_number(columns = c(eta, max_depth, min_child_weight, subsample,
                         colsample_bytree,
                         best_iteration, AUC), decimals = 4) %>%
  cols_label(
    eta = "Learning Rate",
    max_depth = "Max Depth",
    min_child_weight = "Min Child Weight",
    subsample = "Subsample Ratio",
    colsample_bytree = "Column Sample by Tree",
    best_iteration = "Best Iteration",
    AUC = "Best CV AUC"
  )
```

Best Hyperparameters for XGBoost

Learning Rate	Max Depth	Min Child Weight	Subsample Ratio	Column Sample by Tree	Best Iteration	Best CV AUC
0.1550	5.0000	1.0000	1.0000	1.0000	64.0000	0.6471

```
final_params <- list(
  booster = "gbtree",
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = best_params$eta,
  max_depth = best_params$max_depth,
  min_child_weight = best_params$min_child_weight,
  subsample = best_params$subsample,
  colsample_bytree = best_params$colsample_bytree
)

final_model <- xgb.train(params = final_params, data = dtrain,
```

```
                      nrounds = best_params$best_iteration,
                      watchlist = list(train = dtrain, test = dtest),
                      early_stopping_rounds = 10, verbose = FALSE)


test_predictions <- predict(final_model, newdata = dtest)

roc_curve <- roc(test_y, test_predictions)
```

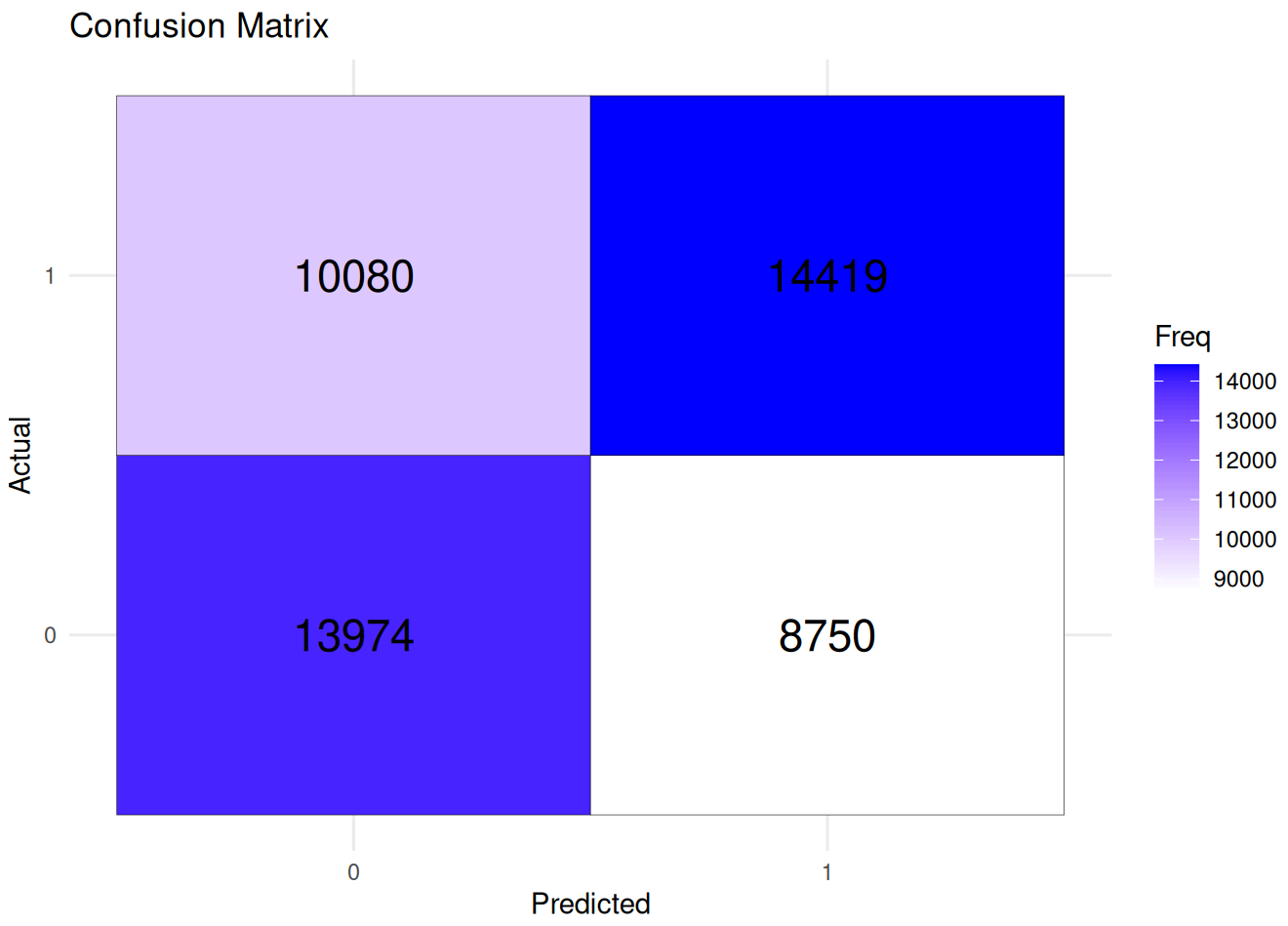
```
Setting levels: control = 0, case = 1

Setting direction: controls < cases
```

```
best_threshold <- coords(roc_curve, "best", ret = "threshold")
best_threshold <- as.numeric(best_threshold[1])

test_pred_labels <- ifelse(test_predictions > best_threshold, 1, 0)

conf_matrix <- table(Predicted = test_pred_labels, Actual = test_y)
accuracy_xgb <- sum(diag(conf_matrix)) / sum(conf_matrix)
auc_xbg <- auc(roc_curve)

results_table <- data.frame(
  Metric = c("Test Accuracy", "AUC Score"),
  Value = c(round(accuracy_xgb, 3), round(auc_xbg, 3))
)

results_table %>%
  gt() %>%
  tab_header(title = "Model Performance Metrics") %>%
  fmt_number(columns = Value, decimals = 3)
```

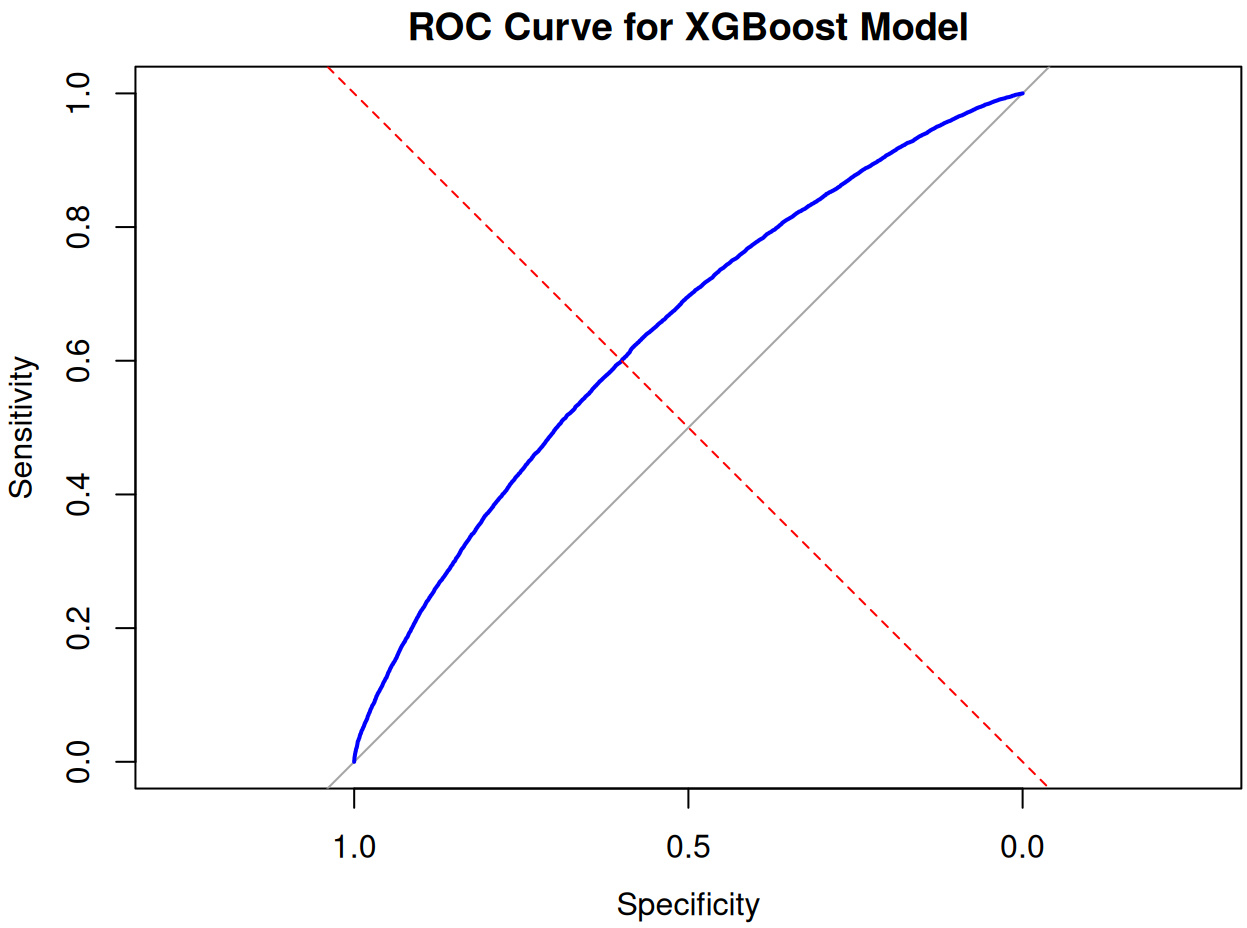
Model Performance Metrics	
Metric	Value
Test Accuracy	0.601
AUC Score	0.641

```
conf_matrix_df <- as.data.frame.matrix(conf_matrix)

conf_matrix_long <- conf_matrix_df %>%
  rownames_to_column(var = "Actual") %>%
  pivot_longer(cols = -Actual, names_to = "Predicted", values_to = "Freq")

ggplot(conf_matrix_long, aes(x = Predicted, y = Actual, fill = Freq)) +
  geom_tile(color = "black") +
```

```
  scale_fill_gradient(low = "white", high = "blue") +
  geom_text(aes(label = Freq), color = "black", size = 6) +
  labs(
    title = "Confusion Matrix",
    x = "Predicted",
    y = "Actual"
  ) +
  theme_minimal()
```




```
plot(roc_curve, col = "blue", main = "ROC Curve for XGBoost Model")
abline(a = 0, b = 1, lty = 2, col = "red")
```



Machine Learning Approach 3: Random Forest

Analysis: Our third and last machine learning approach implements a Random Forest classification model using the Ranger package to predict ICU length of stay (LOS) based on patient data. First, we preprocessed the dataset by converting categorical variables into factors and ensuring that the target variable (los_long) was encoded as a binary factor with two levels: “Class0” (short stay) and “Class1” (long stay). To optimize model performance, we conducted hyperparameter tuning using a grid search approach. The hyperparameters tuned included: mtry (number of variables randomly sampled at each split), splitrule (gini or extratrees for decision tree splitting), and min.node.size (minimum node size for splits). A 3-fold cross-validation strategy was used, with AUC (Area Under the Curve) as the evaluation metric. The best combination of hyperparameters was then selected to train the final Random Forest model, consisting of 200 trees, using all available CPU cores for parallel computation. After training, the model was evaluated on the test set, generating probability predictions for each sample. The optimal classification threshold was determined using Youden’s Index from the ROC curve to maximize sensitivity and specificity.

Results: We are able to see from the results that the Random Forest model achieved a test accuracy of 59.4%, indicating moderate predictive performance. The AUC score of 0.631 suggests that the model has a reasonable ability to differentiate between short and long ICU stays, though it is not able to distinguish the 2 classes of los_long very well.The confusion matrix highlights significant misclassification issues, with 14,280 false negatives (actual short stays misclassified as long) and 13,769 false positives (actual long stays misclassified as short). This

suggests that the model struggles with correctly classifying both classes, potentially due to imbalanced data or overlapping feature distributions. The ROC curve further supports this observation, demonstrating a moderate trade-off between sensitivity and specificity. Overall, while the Random Forest model shows some predictive power, further improvements are necessary and we are able to deduce that the XG Boosting model is the best approach to classify los_long.

```
set.seed(203)
train_sampled <- train_data %>% sample_frac(1.0)
train_sampled <- train_sampled %>%
  mutate(across(where(is.character), as.factor))
test_data <- test_data %>%
  mutate(across(where(is.character), as.factor))

train_sampled$los_long <- factor(
  train_sampled$los_long, levels = c(0, 1), labels = c("Class0", "Class1")
)

test_data$los_long <- factor(
  test_data$los_long, levels = c(0, 1), labels = c("Class0", "Class1")
)

set.seed(123)

tuning_grid <- expand.grid(
  mtry = seq(2, floor(log2(ncol(train_sampled) - 1)), by = 2),
  splitrule = c("gini", "extratrees"),
  min.node.size = c(1, 5, 10)
)

registerDoParallel(cores = parallel::detectCores() - 1)

cv_control <- trainControl(
  method = "cv", number = 3,
  summaryFunction = twoClassSummary,
  classProbs = TRUE,
  allowParallel = TRUE
)

rf_model <- train(
  los_long ~ ., data = train_sampled,
  method = "ranger",
  trControl = cv_control,
  tuneGrid = tuning_grid,
  metric = "ROC"
)

final_rf <- ranger(
  los_long ~ ., data = train_sampled,
  mtry = rf_model$bestTune$mtry,
  splitrule = rf_model$bestTune$splitrule,
```

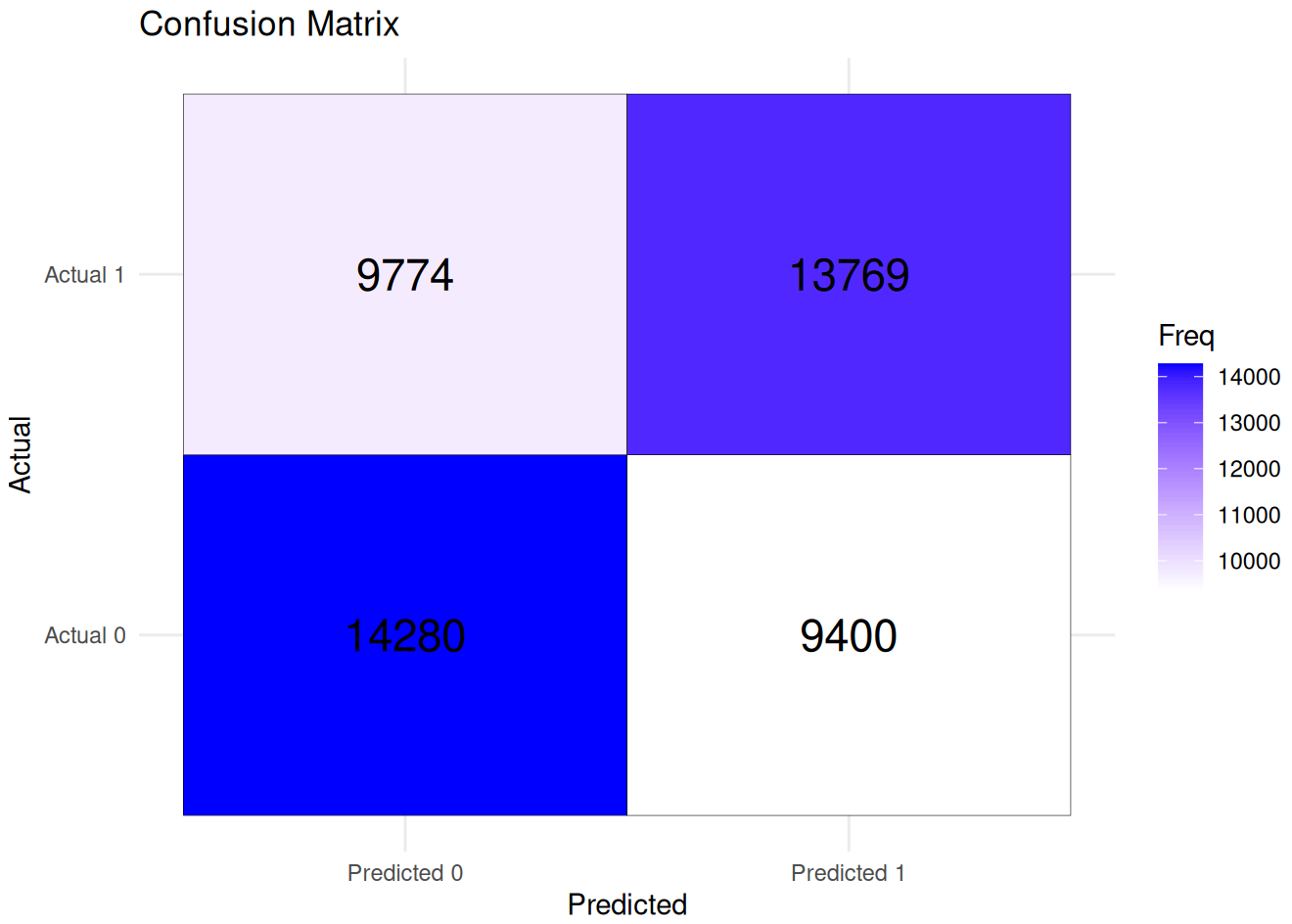
```
  min.node.size = rf_model$bestTune$min.node.size,
  num.trees = 200,
  importance = "impurity",
  probability = TRUE,
  num.threads = parallel::detectCores() - 1
)

rf_pred <- predict(final_rf, test_data, type = "response")

if ("predictions" %in% names(rf_pred)) {
  test_predictions <- rf_pred$predictions
  if (is.matrix(test_predictions)) {
    test_predictions <- test_predictions[, "Class1", drop = TRUE]
  }
}

roc_curve <- roc(test_data$los_long, test_predictions)
```

```
Setting levels: control = Class0, case = Class1

Setting direction: controls < cases
```

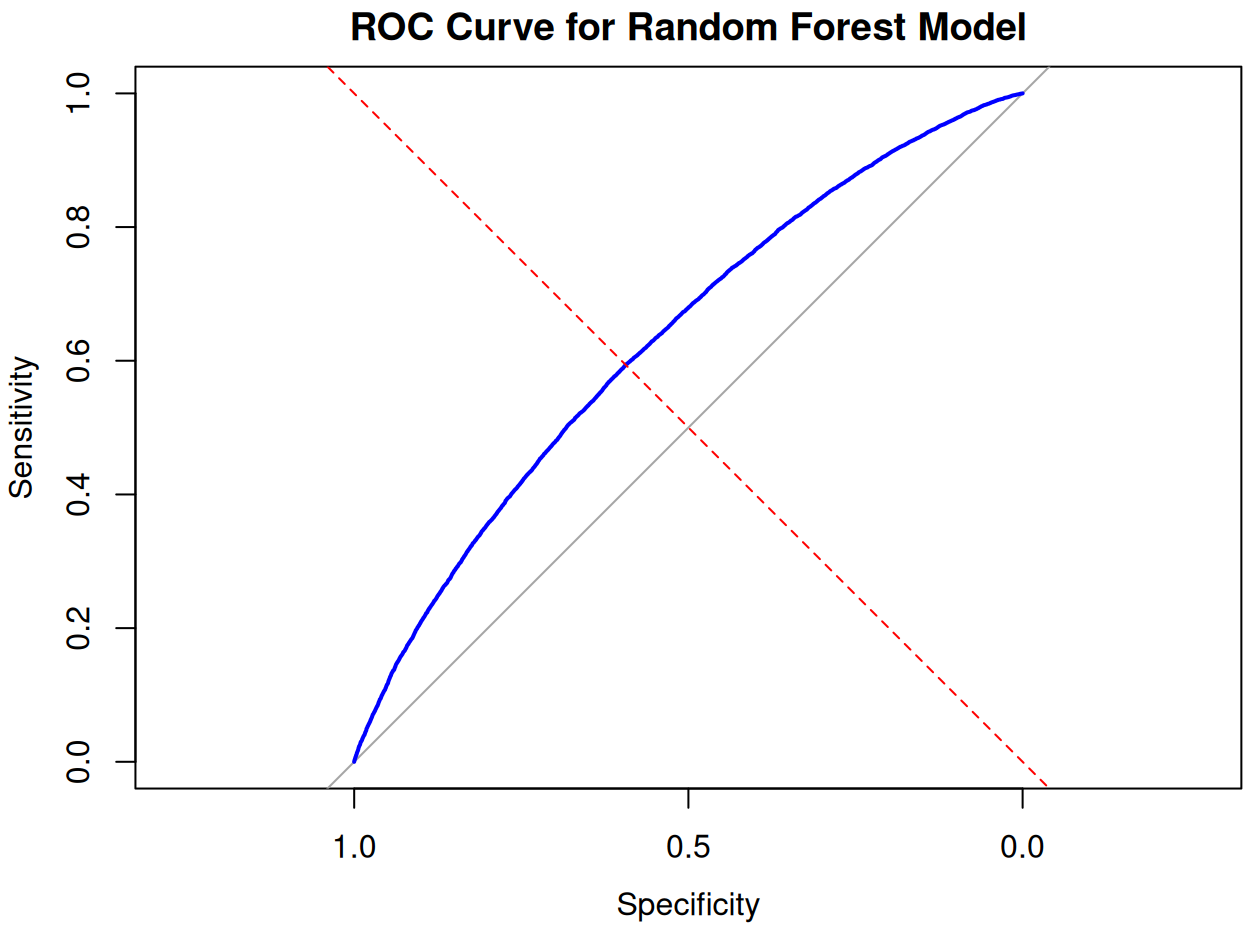
```
best_threshold <- coords(roc_curve, "best", ret = "threshold")
best_threshold <- as.numeric(best_threshold[1])

test_pred_labels <- ifelse(
  test_predictions > best_threshold, "Class1", "Class0"
)

test_pred_labels <- factor(test_pred_labels, levels = c("Class0", "Class1"))
test_data$los_long <- factor(test_data$los_long, levels = c("Class0", "Class1"))

conf_matrix <- table(
  Predicted = test_pred_labels, Actual = test_data$los_long
)

accuracy_rf <- sum(diag(conf_matrix)) / sum(conf_matrix)
auc_rf <- auc(roc_curve)

results_table <- data.frame(
  Metric = c("Test Accuracy", "AUC Score"),
  Value = c(round(accuracy_rf, 3), round(auc_rf, 3))
)

results_table %>%
  gt() %>%
  tab_header(title = "Model Performance Metrics") %>%
  fmt_number(columns = Value, decimals = 3)
```

Model Performance Metrics	
Metric	Value
Test Accuracy	0.594
AUC Score	0.631

```
conf_matrix_df <- as.data.frame.matrix(conf_matrix)

colnames(conf_matrix_df) <- c("Predicted 0", "Predicted 1")
rownames(conf_matrix_df) <- c("Actual 0", "Actual 1")

conf_matrix_long <- conf_matrix_df %>%
  rownames_to_column(var = "Actual") %>%
  pivot_longer(cols = -Actual, names_to = "Predicted", values_to = "Freq")

ggplot(conf_matrix_long, aes(x = Predicted, y = Actual, fill = Freq)) +
  geom_tile(color = "black") +
  scale_fill_gradient(low = "white", high = "blue") +
  geom_text(aes(label = Freq), color = "black", size = 6) +
  labs(
    title = "Confusion Matrix",
    x = "Predicted",
    y = "Actual"
  ) +
  theme_minimal()
```




```
plot(roc_curve, col = "blue", main = "ROC Curve for Random Forest Model")
abline(a = 0, b = 1, lty = 2, col = "red")
```



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

Results

The results indicate that XGBoost outperforms both Logistic Regression with Elastic Net regularization and Random Forest in predicting ICU length of stay, achieving the highest test accuracy (60.1%) and AUC score (0.641). Logistic Regression, while interpretable and efficient, performed the worst (57.8% accuracy, 0.606 AUC), likely due to its linear nature, which struggles to capture complex relationships among features. However, its simplicity allows for clear interpretation of feature importance. We are able to see that the Random Forest machine learning approach improved upon logistic regression (59.4% accuracy, 0.631 AUC), but its reliance on averaging multiple decision trees can make individual predictions less interpretable. XGBoost was the most suitable approach because it efficiently captures non-linear interactions, handles missing data well (which is prevalent in real data sets such as the MIMIC data), and optimizes performance using boosting techniques. Unlike Random Forest, which builds trees independently, XGBoost sequentially corrects errors, leading to a potentially better model. However, it is more computationally intensive and requires careful hyperparameter tuning. Overall, XGBoost provides the best trade-off between predictive power and model complexity, making it the most effective approach for this task.

```
model_results <- list(
  "Logistic Regression (ENet)" = list(
```

```
    accuracy = accuracy_logit,
    auc = auc_score_logit
  ),
  "XGBoost" = list(
    accuracy = accuracy_xgb,
    auc = auc_xbg
  ),
  "Random Forest" = list(
    accuracy = accuracy_rf,
    auc = auc_rf
  )
)

model_comparison <- do.call(
  rbind,
  lapply(names(model_results), function(model) {
    data.frame(
      Model = model,
      Test_Accuracy = model_results[[model]]$accuracy,
      AUC_Score = model_results[[model]]$auc
    )
  })
)

model_comparison %>%
  gt() %>%
  tab_header(title = "Model Performance Comparison") %>%
  fmt_number(columns = c(Test_Accuracy, AUC_Score), decimals = 3) %>%
  cols_label(
    Model = "Machine Learning Model",
    Test_Accuracy = "Test Accuracy",
    AUC_Score = "AUC Score"
  )
```

Model Performance Comparison		
Machine Learning Model	Test Accuracy	AUC Score
Logistic Regression (ENet)	0.578	0.606
XGBoost	0.601	0.641
Random Forest	0.594	0.631

10 Most Important Features

The table presents the top 10 most important features influencing the Random Forest model's predictions for ICU length of stay. The most significant feature is systolic non-invasive blood pressure (importance score: 2,346.97), indicating its strong predictive value. Other important features include heart rate (2,264.77), hematocrit levels

(2,228.56), and white blood cell (WBC) count (2,192.23), all of which play essential roles in patient stability and health status. Additionally, age at ICU admission (2,132.81) and temperature (2,070.43) contribute significantly to predictions, suggesting that both physiological and demographic factors influence ICU stay duration. Other important features include respiratory rate (1,966.11), creatinine (1,633.25), chloride (1,599.86), and sodium (1,571.78).

```
feature_importance <- as.data.frame(importance(final_rf))
feature_importance$Feature <- rownames(feature_importance)

colnames(feature_importance)[1] <- "Importance_Score"

feature_importance <- feature_importance %>%
  arrange(desc(Importance_Score)) %>%
  head(10)

feature_importance %>%
  gt() %>%
  tab_header(title = "Top 10 Most Important Features") %>%
  fmt_number(columns = Importance_Score, decimals = 4) %>%
  cols_label(
    Feature = "Feature Name",
    Importance_Score = "Feature Importance Score"
  )
```

Top 10 Most Important Features

Feature Importance Score	Feature Name
2,346.9698	non_invasive_blood_pressure_systolic
2,264.7725	heart_rate
2,228.5572	hematocrit
2,192.2270	wbc
2,132.8077	age_at_intime
2,070.4293	temperature_fahrenheit
1,966.1080	respiratory_rate
1,633.2516	creatinine
1,599.8614	chloride
1,571.7751	sodium