Final Project Code

Helen Liang, Ivy Zhao, Xiaotong Zhao

05/02/2024

Predicting Heart Disease

The `heart` dataset (source: UCI Machine Learning Repository) contains 920 observations and 76 variables. However, we're only using the following 14 of the 76 variables in our study:

- `age`: patient age (in years)
- `sex`: gender of patient; 0 = male, 1 = female
- `cp`: chest pain type; 1 = typical angina, 2 = atypical angina, 3 = non-anginal pain, 4 = asymptomatic
- `trestbps`: resting blood pressure (in mmHg)
- `chol`: serum cholesterol (in mg/dl)
- `fbs`: fasting blood sugar > 120 mg/dl; 0 = false, 1 = true
- `restecg`: resting electrocardiographic results; 0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Ester's criteria
- `thalach`: maximum heart race achieved
- `exang`: exercise included angina; 0 = no, 1 = yes
- `oldpeak`: ST depression induced by exercise relative to rest
- `slope`: the slope of the peak exercise ST segment; 1 = upsloping, 2 = flat, 3 = downsloping
- `ca`: number of major vessels (0-3) colored by fluoroscopy
- `thal`: thalassemia; 3 = normal, 6 = fixed defect, 7 = reversible defect
- `num`: diagosis of heart disease; 0 = no heart disease, 1 = have heart disease

Load, Merge, and Recode Data

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
cleveland <- read.csv("processed.cleveland.csv")
hungarian <- read.csv("processed.hungarian.csv")
switzerland <- read.csv("processed.switzerland.csv")
va <- read.csv("processed.va.csv")

nrow(cleveland)
```

```
## [1] 303
```

```r
nrow(hungarian)
```

```
## [1] 294
```

```r
nrow(switzerland)
```

```
## [1] 123
```

```r
nrow(va)
```

```
## [1] 200
```

```r
heart <- rbind(cleveland, hungarian, switzerland, va)

# check for missing values
heart[heart == "?"] <- NA
sum(is.na(heart))
```

```
## [1] 1759
```

```r
sapply(heart, function(x) sum(is.na(x)))
```

```
##      age      sex       cp trestbps     chol      fbs  restecg  thalach
##        0        0        0       59       30       90        2       55
##    exang  oldpeak    slope       ca     thal      num
##       55       62      309      611      486        0
```

```r
nrow(heart)
```

```
## [1] 920
```

```r
ncol(heart)
```

```
## [1] 14
```

Deal with NA's

```r
#convert integers to numeric, standardization and replace na with median
heart$age <- as.numeric(heart$age)
heart$age <- scale(heart$age)
heart$age[is.na(heart$age)] <- median(heart$age, na.rm = TRUE)

heart$trestbps <- as.numeric(heart$trestbps)
heart$trestbps <- scale(heart$trestbps)
heart$trestbps[is.na(heart$trestbps)] <- median(heart$trestbps, na.rm = TRUE)

heart$chol <- as.numeric(heart$chol)
heart$chol <- scale(heart$chol)
heart$chol[is.na(heart$chol)] <- median(heart$chol, na.rm = TRUE)

heart$thalach <- as.numeric(heart$thalach)
heart$thalach <- scale(heart$thalach)
heart$thalach[is.na(heart$thalach)] <- median(heart$thalach, na.rm = TRUE)

heart$oldpeak <- as.numeric(heart$oldpeak)
heart$oldpeak <- scale(heart$oldpeak)
heart$oldpeak[is.na(heart$oldpeak)] <- median(heart$oldpeak, na.rm = TRUE)

#replace NA values in the character variables into mode
heart$fbs <- ifelse(is.na(heart$fbs), names(which.max(table(heart$fbs))), heart$fbs)

heart$restecg <- ifelse(is.na(heart$restecg), names(which.max(table(heart$restecg))), heart$restecg)

heart$exang <- ifelse(is.na(heart$exang), names(which.max(table(heart$exang))), heart$exang)

heart$slope <- ifelse(is.na(heart$slope), names(which.max(table(heart$slope))), heart$slope)

#remove ca & thal, as more than half of their observations are mising values.
heart$ca <- NULL
heart$thal <- NULL

#check for missing values
sum(is.na(heart))
```

```
## [1] 0
```

```r
sapply(heart, function(x) sum(is.na(x)))
```

```
##      age      sex       cp trestbps     chol      fbs  restecg  thalach
##        0        0        0        0        0        0        0        0
##    exang  oldpeak    slope      num
##        0        0        0        0
```

```r
nrow(heart)
```

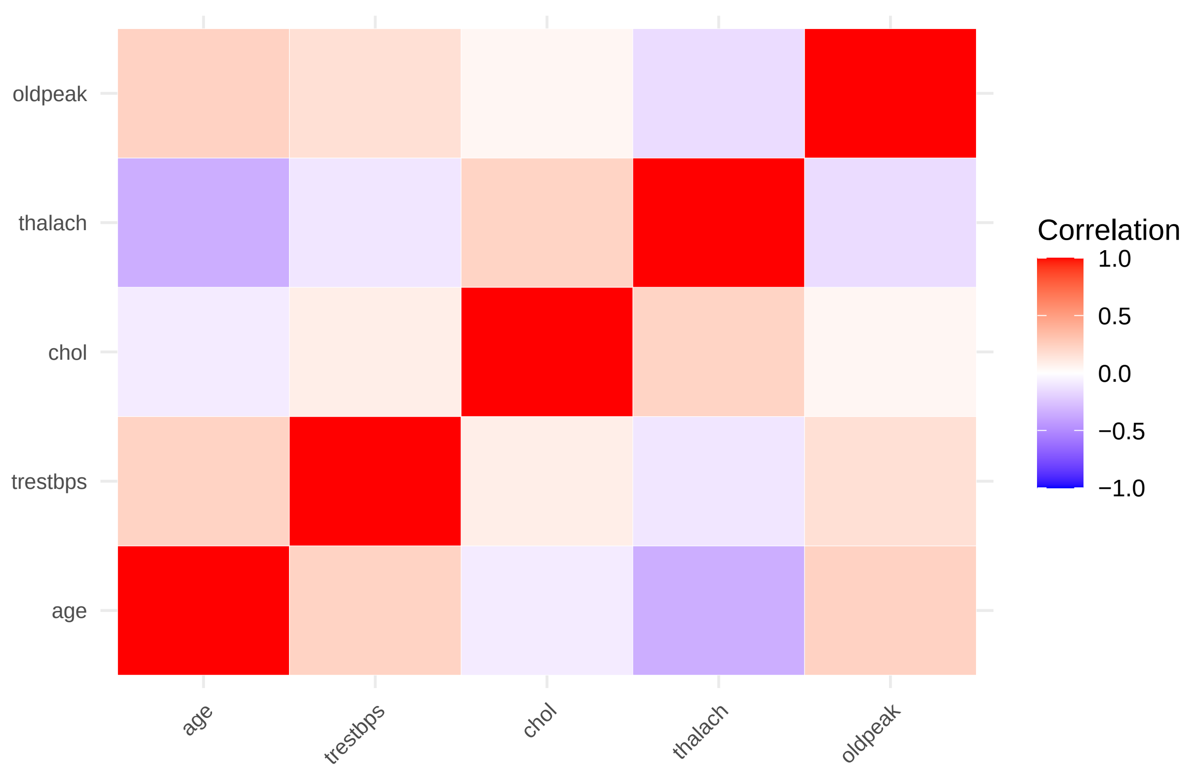
```
## [1] 920
```

```r
ncol(heart)
```

```
## [1] 12
```

Correlation Heatmap

```r
library(reshape2)

cor_matrix <- cor(heart[, c("age", "trestbps", "chol", "thalach", "oldpeak")])
melted_cor_matrix <- melt(cor_matrix)

library(ggplot2)

ggplot(melted_cor_matrix, aes(Var1, Var2, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint = 0, limit = c(-1,1),
                       space = "Lab", name="Correlation") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, vjust = 1, size = 8, hjust = 1),
    axis.text.y = element_text(size = 8)
  ) +
  labs(x = "", y = "", title = "")
```



#Feature Selection

```r
library(leaps)
# Forward Stepwise Selection with Adjusted R^2
forward_fit <- regsubsets(num ~ ., data = heart, method = "forward", nvmax = 11)
forward_sum <- summary(forward_fit)
best_ind_for <- which.max(forward_sum$adjr2)
best_model_forward <- coef(forward_fit, best_ind_for)
best_model_forward
```

```
## (Intercept)         age         sex          cp        chol        fbs1
##  -0.3114297   0.1610103   0.3459368   0.2403301  -0.2036175   0.2374377
##    restecg1    restecg2     thalach      exang1     oldpeak      slope2
##   0.1548254   0.2555852  -0.1119386   0.1838280   0.3246255   0.1088429
```

```r
# Backward Stepwise Selection with Cp
backward_fit <- regsubsets(num ~ ., data = heart, method = "backward", nvmax = 11)
backward_sum <- summary(backward_fit)
best_ind_back <- which.min(backward_sum$cp)
best_model_backward <- coef(backward_fit, best_ind_back)
best_model_backward
```

```
## (Intercept)         age         sex          cp        chol        fbs1
##  -0.3114297   0.1610103   0.3459368   0.2403301  -0.2036175   0.2374377
##    restecg1    restecg2     thalach      exang1     oldpeak      slope2
##   0.1548254   0.2555852  -0.1119386   0.1838280   0.3246255   0.1088429
```

```r
# age, sex, cp, chol, fbs1,  restecg, thalach, exang, oldpeak, slope
```

```r
#convert character variables into factors
heart$sex <- factor(heart$sex,
                    levels = c(0, 1),
                    labels = c("male", "female"))

heart$cp <- factor(heart$cp,
                    levels = c(1, 2, 3, 4),
                    labels = c("typical angina", "atypial angina", "non-anginal pain", "asymptomatic"))

heart$fbs <- factor(heart$fbs,
                    levels = c(0, 1),
                    labels = c("false", "true"))

heart$restecg <- factor(heart$restecg,
                    levels = c(0, 1, 2),
                    labels = c("normal", "abonormal", "left ventricular hypertrophy"))

heart$exang <- factor(heart$exang,
                    levels = c(0, 1),
                    labels = c("no", "yes"))

heart$slope <- factor(heart$slope,
                    levels = c(1, 2, 3),
```

```r
                     labels = c("upsloping", "flat", "downsloping"))

heart$num <- as.integer(heart$num > 0)
heart$num <- factor(heart$num,
                     levels = c(0, 1),
                     labels = c("no", "yes"))
#check for unbalanced label
summary(heart$num)
```

```
##  no yes
## 411 509
```

Initial Data Preparation

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v stringr   1.5.0
## v lubridate 1.9.2     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.0
## v readr     2.1.4
## -- 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 become errors
```

```r
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
set.seed(123)

#select the predictors from feature selection
heart <- heart %>%
         select(age, sex, cp, chol, fbs, restecg, thalach, exang, oldpeak,slope,num)
head(heart)
```

```
##           age    sex              cp       chol   fbs
## 1  1.0068379 female  typical angina 0.30573583  true
## 2  1.4312553 female    asymptomatic 0.78415804 false
## 3  1.4312553 female    asymptomatic 0.26962849 false
## 4 -1.7518749 female non-anginal pain 0.45919201 false
```

```
## 5 -1.3274576   male   atypial angina 0.04395764 false
## 6  0.2641075 female   atypial angina 0.33281633 false
##                        restecg  thalach exang     oldpeak       slope num
## 1 left ventricular hypertrophy  0.480375    no  1.30239913 downsloping  no
## 2 left ventricular hypertrophy -1.139603   yes  0.56927894        flat yes
## 3 left ventricular hypertrophy -0.329614   yes  1.57731921        flat yes
## 4                       normal  1.907499    no  2.40207943 downsloping  no
## 5 left ventricular hypertrophy  1.328935    no  0.47763891   upsloping  no
## 6                       normal  1.560360    no -0.07220123   upsloping  no
```

```
tr_ind <- sample(1:nrow(heart), 0.8 * nrow(heart))
heart_train <- heart[tr_ind, ]
heart_test <- heart[-tr_ind, ]
```

We will use the following machine learning models for heart disease diagnosis:
1. Logistic Regression
2. K-Nearest Neighbors (KNN)
3. Random Forest
4. Gradient Boosting
5. Support Vector Machines (SVM)

First, we will fit models using the above machine learning techniques and compute the metrics (see below) for validating model performances. Then we will use k-folds cross-validation (CV) to improve on all models.

Metrics for validating model performances:
1. Compute training & testing errors
2.
$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

$$\text{sensitivity (TPR)} = \frac{TP}{TP + FN}$$

$$\text{specificity (FPR)} = \frac{TN}{TN + FP}$$

3. Calculate AUC for each model
4. Plot AUCROC for all models on the same plot and compare

Logistic Regression Model

```r
library(caret)

set.seed(123)

logistic_model <- glm(num ~., data = heart_train, family = "binomial")

#training error
predict_train_prob <- predict(logistic_model, type = "response")
predict_train_label <- ifelse(predict_train_prob > 0.5, "yes", "no")
train_error <- mean(predict_train_label != heart_train$num)
print(train_error)
```

```
## [1] 0.1779891
```

```r
#training error:0.1779891

#testing error
predict_test_prob <- predict(logistic_model, newdata = heart_test, type = "response")
predict_test_label <- ifelse(predict_test_prob > 0.5, "yes", "no")
test_error <- mean(predict_test_label != heart_test$num)
print(test_error)
```

```
## [1] 0.2065217
```

```r
#testing error: 0.2065217

#confusion matrix & accuracy
predictions <- factor(predict_test_label, levels = c("no", "yes"))
y_test <- factor(heart_test$num, levels = c("no", "yes"))

confusion_matrix <- confusionMatrix(predictions, y_test,mode = "everything")
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  51  16
##        yes 22  95
##
##                Accuracy : 0.7935
##                  95% CI : (0.7277, 0.8495)
##     No Information Rate : 0.6033
##     P-Value [Acc > NIR] : 2.936e-08
##
##                   Kappa : 0.5624
##
##  Mcnemar's Test P-Value : 0.4173
##
```

```
##             Sensitivity : 0.6986
##             Specificity : 0.8559
##          Pos Pred Value : 0.7612
##          Neg Pred Value : 0.8120
##               Precision : 0.7612
##                  Recall : 0.6986
##                      F1 : 0.7286
##              Prevalence : 0.3967
##          Detection Rate : 0.2772
##    Detection Prevalence : 0.3641
##       Balanced Accuracy : 0.7772
##
##        'Positive' Class : no
##
```

```r
#Accuracy :  0.7935
#F1 : 0.7286

library(pROC)
```

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

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
#aucroc
roc_logistic <- roc(heart_test$num, predict_test_prob)
```

```
## Setting levels: control = no, case = yes
```

```
## Setting direction: controls < cases
```

```r
auc_logistic <- auc(roc_logistic)
print(auc_logistic)
```

```
## Area under the curve: 0.8841
```

```r
#Area under the curve: 0.8841
```

Logistic regression Model using Elastic Net and cross-validation for regulation.

```r
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(caret)
set.seed(123)

X_train <- heart_train[, -which(names(heart_train) == "num")]
y_train <- heart_train$num
X_test <- heart_test[, -which(names(heart_test) == "num")]
y_test <- heart_test$num

# Set up the trainControl for 5-fold cross-validation
ctrl <- trainControl(method = "cv", number = 5)

# Define the tuning grid for alpha and lambda
grid <- expand.grid(alpha = seq(0, 1, by = 0.1), lambda = seq(0.05, 0.1, by = 0.002))

#Perform 5-fold cross-validation to tune hyperparameters
logi_reg_model <- train(x = X_test,y = y_test,method = "glmnet",trControl = ctrl,
                        tuneGrid = grid,metric = "Accuracy")
```

```
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
```

```
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
```

```
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
```

```
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
```

```
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
```

```
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
```

```
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion

## Warning in storage.mode(xd) <- "double": NAs introduced by coercion
```

```r
print(logi_reg_model)
```

```
## glmnet
##
## 184 samples
##  10 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 147, 148, 146, 148, 147
## Resampling results across tuning parameters:
##
##   alpha  lambda  Accuracy   Kappa
##   0.0    0.050   0.7174964  0.4022094
##   0.0    0.052   0.7174964  0.4022094
##   0.0    0.054   0.7174964  0.4022094
##   0.0    0.056   0.7174964  0.4022094
##   0.0    0.058   0.7174964  0.4022094
##   0.0    0.060   0.7174964  0.4022094
##   0.0    0.062   0.7174964  0.4022094
##   0.0    0.064   0.7120910  0.3890896
##   0.0    0.066   0.7120910  0.3890896
##   0.0    0.068   0.7229018  0.4085980
##   0.0    0.070   0.7229018  0.4085980
##   0.0    0.072   0.7229018  0.4085980
##   0.0    0.074   0.7176387  0.3957991
##   0.0    0.076   0.7176387  0.3957991
##   0.0    0.078   0.7230441  0.4058739
##   0.0    0.080   0.7230441  0.4058739
##   0.0    0.082   0.7230441  0.4058739
##   0.0    0.084   0.7285997  0.4154945
##   0.0    0.086   0.7285997  0.4154945
##   0.0    0.088   0.7285997  0.4154945
##   0.0    0.090   0.7285997  0.4154945
##   0.0    0.092   0.7285997  0.4154945
##   0.0    0.094   0.7285997  0.4154945
##   0.0    0.096   0.7233365  0.4023806
##   0.0    0.098   0.7179311  0.3892608
##   0.0    0.100   0.7179311  0.3892608
##   0.1    0.050   0.7174964  0.4022094
```

```
##    0.1    0.052   0.7174964  0.4022094
##    0.1    0.054   0.7174964  0.4022094
##    0.1    0.056   0.7174964  0.4022094
##    0.1    0.058   0.7120910  0.3890896
##    0.1    0.060   0.7120910  0.3890896
##    0.1    0.062   0.7120910  0.3890896
##    0.1    0.064   0.7174964  0.3987392
##    0.1    0.066   0.7229018  0.4085980
##    0.1    0.068   0.7229018  0.4085980
##    0.1    0.070   0.7229018  0.4085980
##    0.1    0.072   0.7176387  0.3957991
##    0.1    0.074   0.7176387  0.3957991
##    0.1    0.076   0.7176387  0.3957991
##    0.1    0.078   0.7176387  0.3957991
##    0.1    0.080   0.7230441  0.4058739
##    0.1    0.082   0.7230441  0.4058739
##    0.1    0.084   0.7285997  0.4154945
##    0.1    0.086   0.7179311  0.3889670
##    0.1    0.088   0.7179311  0.3889670
##    0.1    0.090   0.7125257  0.3758472
##    0.1    0.092   0.7125257  0.3758472
##    0.1    0.094   0.7125257  0.3758472
##    0.1    0.096   0.7125257  0.3758472
##    0.1    0.098   0.7125257  0.3758472
##    0.1    0.100   0.7069701  0.3614993
##    0.2    0.050   0.7174964  0.4022094
##    0.2    0.052   0.7174964  0.4022094
##    0.2    0.054   0.7120910  0.3890896
##    0.2    0.056   0.7120910  0.3890896
##    0.2    0.058   0.7174964  0.3987392
##    0.2    0.060   0.7174964  0.3987392
##    0.2    0.062   0.7229018  0.4085980
##    0.2    0.064   0.7229018  0.4085980
##    0.2    0.066   0.7229018  0.4085980
##    0.2    0.068   0.7229018  0.4085980
##    0.2    0.070   0.7229018  0.4085980
##    0.2    0.072   0.7176387  0.3957991
##    0.2    0.074   0.7122333  0.3823855
##    0.2    0.076   0.7122333  0.3823855
##    0.2    0.078   0.7069701  0.3692716
##    0.2    0.080   0.7123755  0.3793464
##    0.2    0.082   0.7123755  0.3793464
##    0.2    0.084   0.7125257  0.3758472
##    0.2    0.086   0.7069701  0.3614993
##    0.2    0.088   0.7069701  0.3614993
##    0.2    0.090   0.7069701  0.3614993
##    0.2    0.092   0.7017070  0.3480587
##    0.2    0.094   0.7017070  0.3480587
##    0.2    0.096   0.7017070  0.3480587
##    0.2    0.098   0.7017070  0.3480587
##    0.2    0.100   0.7017070  0.3480587
##    0.3    0.050   0.7230520  0.4118299
##    0.3    0.052   0.7176466  0.3987102
##    0.3    0.054   0.7230520  0.4083598
```

```
##   0.3    0.056   0.7230520  0.4083598
##   0.3    0.058   0.7230520  0.4083598
##   0.3    0.060   0.7284574  0.4182186
##   0.3    0.062   0.7284574  0.4182186
##   0.3    0.064   0.7230520  0.4048049
##   0.3    0.066   0.7230520  0.4048049
##   0.3    0.068   0.7230520  0.4048049
##   0.3    0.070   0.7177888  0.3920061
##   0.3    0.072   0.7125257  0.3788922
##   0.3    0.074   0.7069701  0.3645444
##   0.3    0.076   0.7069701  0.3645444
##   0.3    0.078   0.7015647  0.3514579
##   0.3    0.080   0.7015647  0.3514579
##   0.3    0.082   0.7072625  0.3579198
##   0.3    0.084   0.7072625  0.3579198
##   0.3    0.086   0.7072625  0.3579198
##   0.3    0.088   0.7126679  0.3681504
##   0.3    0.090   0.7126679  0.3681504
##   0.3    0.092   0.7126679  0.3681504
##   0.3    0.094   0.7126679  0.3681504
##   0.3    0.096   0.7126679  0.3681504
##   0.3    0.098   0.7126679  0.3681504
##   0.3    0.100   0.7126679  0.3681504
##   0.4    0.050   0.7230520  0.4083598
##   0.4    0.052   0.7230520  0.4083598
##   0.4    0.054   0.7230520  0.4083598
##   0.4    0.056   0.7230520  0.4083598
##   0.4    0.058   0.7230520  0.4048049
##   0.4    0.060   0.7230520  0.4048049
##   0.4    0.062   0.7230520  0.4048049
##   0.4    0.064   0.7230520  0.4048049
##   0.4    0.066   0.7174964  0.3904571
##   0.4    0.068   0.7069701  0.3645444
##   0.4    0.070   0.7069701  0.3645444
##   0.4    0.072   0.7015647  0.3514579
##   0.4    0.074   0.6963016  0.3380173
##   0.4    0.076   0.7017070  0.3482479
##   0.4    0.078   0.7017070  0.3482479
##   0.4    0.080   0.7072625  0.3581090
##   0.4    0.082   0.7126679  0.3681504
##   0.4    0.084   0.7126679  0.3681504
##   0.4    0.086   0.7126679  0.3681504
##   0.4    0.088   0.7126679  0.3681504
##   0.4    0.090   0.7126679  0.3681504
##   0.4    0.092   0.7126679  0.3681504
##   0.4    0.094   0.7074048  0.3543706
##   0.4    0.096   0.7074048  0.3543706
##   0.4    0.098   0.7074048  0.3543706
##   0.4    0.100   0.7129603  0.3644813
##   0.5    0.050   0.7230520  0.4083598
##   0.5    0.052   0.7230520  0.4048049
##   0.5    0.054   0.7230520  0.4048049
##   0.5    0.056   0.7230520  0.4048049
##   0.5    0.058   0.7230520  0.4048049
```

```
## 0.5 0.060 0.7174964 0.3904571
## 0.5 0.062 0.7174964 0.3904571
## 0.5 0.064 0.7122333 0.3776582
## 0.5 0.066 0.7123755 0.3747751
## 0.5 0.068 0.7017070 0.3482479
## 0.5 0.070 0.7017070 0.3482479
## 0.5 0.072 0.7017070 0.3482479
## 0.5 0.074 0.7071124 0.3582893
## 0.5 0.076 0.7071124 0.3582893
## 0.5 0.078 0.7071124 0.3582893
## 0.5 0.080 0.7126679 0.3681504
## 0.5 0.082 0.7126679 0.3681504
## 0.5 0.084 0.7126679 0.3681504
## 0.5 0.086 0.7074048 0.3543706
## 0.5 0.088 0.7129603 0.3644813
## 0.5 0.090 0.7129603 0.3644813
## 0.5 0.092 0.7129603 0.3644813
## 0.5 0.094 0.7129603 0.3644813
## 0.5 0.096 0.7129603 0.3644813
## 0.5 0.098 0.7129603 0.3644813
## 0.5 0.100 0.7129603 0.3644813
## 0.6 0.050 0.7230520 0.4048049
## 0.6 0.052 0.7230520 0.4048049
## 0.6 0.054 0.7174964 0.3904571
## 0.6 0.056 0.7174964 0.3904571
## 0.6 0.058 0.7174964 0.3904571
## 0.6 0.060 0.7123755 0.3747751
## 0.6 0.062 0.7123755 0.3747751
## 0.6 0.064 0.7069701 0.3616886
## 0.6 0.066 0.7071124 0.3582893
## 0.6 0.068 0.7071124 0.3582893
## 0.6 0.070 0.7071124 0.3582893
## 0.6 0.072 0.7071124 0.3582893
## 0.6 0.074 0.7071124 0.3582893
## 0.6 0.076 0.7071124 0.3582893
## 0.6 0.078 0.7126679 0.3681504
## 0.6 0.080 0.7129603 0.3644813
## 0.6 0.082 0.7129603 0.3644813
## 0.6 0.084 0.7129603 0.3644813
## 0.6 0.086 0.7129603 0.3644813
## 0.6 0.088 0.7129603 0.3644813
## 0.6 0.090 0.7129603 0.3644813
## 0.6 0.092 0.7183657 0.3747477
## 0.6 0.094 0.7183657 0.3747477
## 0.6 0.096 0.7131026 0.3606156
## 0.6 0.098 0.7075470 0.3458457
## 0.6 0.100 0.7075470 0.3458457
## 0.7 0.050 0.7174964 0.3904571
## 0.7 0.052 0.7174964 0.3904571
## 0.7 0.054 0.7229018 0.4006877
## 0.7 0.056 0.7123755 0.3747751
## 0.7 0.058 0.7123755 0.3747751
## 0.7 0.060 0.7123755 0.3717300
## 0.7 0.062 0.7068200 0.3621094
```

```
##   0.7   0.064   0.7015568  0.3486687
##   0.7   0.066   0.7015568  0.3486687
##   0.7   0.068   0.7015568  0.3486687
##   0.7   0.070   0.7015568  0.3486687
##   0.7   0.072   0.7071124  0.3582893
##   0.7   0.074   0.7074048  0.3543706
##   0.7   0.076   0.7129603  0.3644813
##   0.7   0.078   0.7129603  0.3644813
##   0.7   0.080   0.7129603  0.3644813
##   0.7   0.082   0.7129603  0.3644813
##   0.7   0.084   0.7128102  0.3599778
##   0.7   0.086   0.7075470  0.3458457
##   0.7   0.088   0.7075470  0.3458457
##   0.7   0.090   0.7075470  0.3458457
##   0.7   0.092   0.7075470  0.3458457
##   0.7   0.094   0.7021416  0.3323964
##   0.7   0.096   0.7021416  0.3323964
##   0.7   0.098   0.7021416  0.3323964
##   0.7   0.100   0.7018492  0.3297870
##   0.8   0.050   0.7173463  0.3910671
##   0.8   0.052   0.7120831  0.3782683
##   0.8   0.054   0.7068200  0.3651545
##   0.8   0.056   0.7068200  0.3621094
##   0.8   0.058   0.7068200  0.3621094
##   0.8   0.060   0.7068200  0.3621094
##   0.8   0.062   0.7068200  0.3621094
##   0.8   0.064   0.7015568  0.3486687
##   0.8   0.066   0.7071124  0.3582893
##   0.8   0.068   0.7071124  0.3582893
##   0.8   0.070   0.7018492  0.3445095
##   0.8   0.072   0.7018492  0.3445095
##   0.8   0.074   0.7018492  0.3396007
##   0.8   0.076   0.7019915  0.3357350
##   0.8   0.078   0.7019915  0.3357350
##   0.8   0.080   0.7019915  0.3357350
##   0.8   0.082   0.7019915  0.3357350
##   0.8   0.084   0.6965861  0.3222857
##   0.8   0.086   0.6965861  0.3222857
##   0.8   0.088   0.7018492  0.3323832
##   0.8   0.090   0.7074048  0.3424939
##   0.8   0.092   0.7019994  0.3287364
##   0.8   0.094   0.6908883  0.3008188
##   0.8   0.096   0.6853327  0.2851472
##   0.8   0.098   0.6853327  0.2851472
##   0.8   0.100   0.6799273  0.2713897
##   0.9   0.050   0.7120831  0.3782683
##   0.9   0.052   0.7068200  0.3621094
##   0.9   0.054   0.7068200  0.3621094
##   0.9   0.056   0.7068200  0.3621094
##   0.9   0.058   0.7068200  0.3621094
##   0.9   0.060   0.7123755  0.3717300
##   0.9   0.062   0.7071124  0.3582893
##   0.9   0.064   0.7071124  0.3582893
##   0.9   0.066   0.6962937  0.3297396
```

```
##   0.9    0.068    0.6962937  0.3297396
##   0.9    0.070    0.6964359  0.3258739
##   0.9    0.072    0.7019915  0.3357350
##   0.9    0.074    0.7019915  0.3357350
##   0.9    0.076    0.7018492  0.3323832
##   0.9    0.078    0.7018492  0.3323832
##   0.9    0.080    0.7018492  0.3323832
##   0.9    0.082    0.7074048  0.3424939
##   0.9    0.084    0.7019994  0.3287364
##   0.9    0.086    0.6964438  0.3135257
##   0.9    0.088    0.6853327  0.2851472
##   0.9    0.090    0.6853327  0.2851472
##   0.9    0.092    0.6799273  0.2713897
##   0.9    0.094    0.6799273  0.2677402
##   0.9    0.096    0.6797771  0.2650874
##   0.9    0.098    0.6743717  0.2509683
##   0.9    0.100    0.6743717  0.2509683
##   1.0    0.050    0.7068200  0.3621094
##   1.0    0.052    0.7068200  0.3621094
##   1.0    0.054    0.7068200  0.3621094
##   1.0    0.056    0.7123755  0.3717300
##   1.0    0.058    0.7123755  0.3717300
##   1.0    0.060    0.7015568  0.3435195
##   1.0    0.062    0.6962937  0.3297396
##   1.0    0.064    0.6964359  0.3258739
##   1.0    0.066    0.6964359  0.3258739
##   1.0    0.068    0.7016991  0.3359714
##   1.0    0.070    0.7018492  0.3323832
##   1.0    0.072    0.7018492  0.3323832
##   1.0    0.074    0.7074048  0.3424939
##   1.0    0.076    0.7074048  0.3424939
##   1.0    0.078    0.7019994  0.3287364
##   1.0    0.080    0.6908883  0.2978541
##   1.0    0.082    0.6908883  0.2978541
##   1.0    0.084    0.6853327  0.2851472
##   1.0    0.086    0.6799273  0.2713897
##   1.0    0.088    0.6799273  0.2677402
##   1.0    0.090    0.6743717  0.2547033
##   1.0    0.092    0.6743717  0.2509683
##   1.0    0.094    0.6688162  0.2348144
##   1.0    0.096    0.6796270  0.2563308
##   1.0    0.098    0.6737790  0.2395763
##   1.0    0.100    0.6682235  0.2214410
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0 and lambda = 0.094.
```

```r
# The final values used for the model were alpha = 0 and lambda = 0.05.
logi_reg_model$bestTune
```

```
##    alpha lambda
## 23     0  0.094
```

```r
# Evaluate the model
predictions_logiregu <- predict(logi_reg_model, X_test)
```

```
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```r
confusion_matrix_logiregu <- confusionMatrix(predictions_logiregu, y_test, mode = "everything")
print(confusion_matrix_logiregu)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  40  17
##        yes 33  94
##
##                Accuracy : 0.7283
##                  95% CI : (0.6579, 0.7911)
##     No Information Rate : 0.6033
##     P-Value [Acc > NIR] : 0.0002639
##
##                   Kappa : 0.4102
##
##  Mcnemar's Test P-Value : 0.0338949
##
##             Sensitivity : 0.5479
##             Specificity : 0.8468
##          Pos Pred Value : 0.7018
##          Neg Pred Value : 0.7402
##               Precision : 0.7018
##                  Recall : 0.5479
##                      F1 : 0.6154
##              Prevalence : 0.3967
##          Detection Rate : 0.2174
##    Detection Prevalence : 0.3098
##       Balanced Accuracy : 0.6974
##
##        'Positive' Class : no
##
```

```r
# Accuracy : 0.7283
#F1 : 0.6667

# training error
train_predict_logiregu <- predict(logi_reg_model, X_train)
```

```
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```r
train_error_logiregu <- mean(train_predict_logiregu != y_train)
train_error_logiregu
```

```
## [1] 0.2472826
```

```
# training error :0.2486413

# test error
test_predict_logiregu <- predict(logi_reg_model, X_test)
```

```
## Warning in cbind2(1, newx) %*% nbeta: NAs introduced by coercion
```

```
test_error_logiregu <- mean(test_predict_logiregu != y_test)
test_error_logiregu
```

```
## [1] 0.2717391
```

```
# test error : 0.2717391

#roc auc
library(pROC)
roc_logiregu <- roc(y_test, as.numeric(test_predict_logiregu), levels = rev(levels(y_test)))
```

```
## Setting direction: controls > cases
```

```
auc_logiregu <- auc(roc_logiregu)
auc_logiregu
```

```
## Area under the curve: 0.6974
```

```
#Area under the curve: 0.7208
```

K-Nearest Neighbors (KNN) Model

```
library(class)

set.seed(123)

k_seq <- seq(from = 1, to = 50, by = 1)

train_error <- numeric(length(k_seq))
test_error <- numeric(length(k_seq))
train_accuracy <- numeric(length(k_seq))
test_accuracy <- numeric(length(k_seq))
train_confusion <- vector("list", length(k_seq))
test_confusion <- vector("list", length(k_seq))


for(i in seq_along(k_seq)){
  k <- k_seq[i]

  knn_model <- knn3(num ~., data = heart_train, k = k)
```

```r
  train_predictions <- predict(knn_model, newdata = heart_train, type = "class")
  train_confusion[[i]] <- table(predicted = train_predictions, actual = heart_train$num)
  train_error[i] <- mean(train_predictions != heart_train$num)
  train_accuracy[i] <- 1 - train_error[i]

  test_predictions <- predict(knn_model, newdata = heart_test, type = "class")
  test_confusion[[i]] <- table(predicted = test_predictions, actual = heart_test$num)
  test_error[i] <- mean(test_predictions != heart_test$num)
  test_accuracy[i] <- 1 - test_error[i]
}

knn_df<-data.frame(train_error,test_error)

#optimal k is 12,15,16,20,22 (using test error)
# combined with training error, optimal k is 12

#training error
print(train_error[12])
```

```
## [1] 0.1671196
```

```r
#training error: 0.1671196

#testing error
print(test_error[12])
```

```
## [1] 0.1956522
```

```r
#testing error: 0.1956522

#confusion matrix & accuracy
check_k <- 12
train_confusion[[check_k]]
```

```
##          actual
## predicted  no yes
##       no  266  51
##       yes  72 347
```

```r
test_confusion[[check_k]]
```

```
##          actual
## predicted no yes
##       no  54  17
##       yes 19  94
```

```r
test_accuracy[check_k]
```

```
## [1] 0.8043478
```

```r
# accuracy: 0.8043478

#aucroc
set.seed(123)
knn_model <- knn3(num ~., data = heart_train, k = 12)
test_predictions <- predict(knn_model, newdata = heart_test, type = "class")
roc_knn <- roc(heart_test$num, as.numeric(test_predictions))
```

```
## Setting levels: control = no, case = yes
```

```
## Setting direction: controls < cases
```

```r
auc_knn <- auc(roc_knn)
print(auc_knn)
```

```
## Area under the curve: 0.8001
```

```r
#Area under the curve: 0.8001
```

Using K-folds to Improve K-Nearest Neighbors (KNN) Model

```r
library(caret)
library(class)

set.seed(123)

train_control <- trainControl(method = "cv", number = 5,
                              savePredictions = "final", classProbs = TRUE)
tune_grid <- expand.grid(k = 1:50)
knn_model_tuned <- train(num ~., data = heart_train, method = "knn",
                         trControl = train_control, preProcess = "scale",  tuneGrid = tune_grid)
print(knn_model_tuned)
```

```
## k-Nearest Neighbors
##
## 736 samples
##  10 predictor
##   2 classes: 'no', 'yes'
##
## Pre-processing: scaled (14)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 589, 589, 589, 589, 588
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.7363302  0.4706913
##    2  0.7363762  0.4718558
##    3  0.7661978  0.5301525
##    4  0.7635227  0.5252189
##    5  0.7784703  0.5544382
##    6  0.7825336  0.5618082
##    7  0.7852546  0.5670339
##    8  0.7865968  0.5698539
##    9  0.7825244  0.5616686
##   10  0.7865876  0.5700207
##   11  0.7906876  0.5783425
##   12  0.7879757  0.5738630
##   13  0.7893179  0.5759920
##   14  0.7879298  0.5735267
##   15  0.7920482  0.5813318
##   16  0.7974812  0.5923096
##   17  0.8056536  0.6086280
##   18  0.8083839  0.6144393
##   19  0.8083747  0.6143145
##   20  0.7974903  0.5917245
##   21  0.8029233  0.6034834
##   22  0.8029417  0.6027094
##   23  0.8083563  0.6139710
##   24  0.8042839  0.6057632
##   25  0.8029141  0.6031357
##   26  0.8069866  0.6108219
```

```
##   27  0.8042839  0.6052284
##   28  0.8056444  0.6078968
##   29  0.8083471  0.6134877
##   30  0.8110682  0.6190382
##   31  0.8110866  0.6187693
##   32  0.8097444  0.6159249
##   33  0.8097352  0.6157438
##   34  0.8070142  0.6102012
##   35  0.8110958  0.6180391
##   36  0.8069958  0.6096246
##   37  0.8070050  0.6096623
##   38  0.8029417  0.6013685
##   39  0.8056536  0.6067819
##   40  0.8015720  0.5989314
##   41  0.7988601  0.5930085
##   42  0.8002022  0.5959217
##   43  0.8002114  0.5954540
##   44  0.8002114  0.5956708
##   45  0.8029325  0.6009237
##   46  0.7988601  0.5927660
##   47  0.7988601  0.5927660
##   48  0.8015812  0.5982546
##   49  0.7975087  0.5901745
##   50  0.7961390  0.5876471
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 35.
```

```r
# Optimal K = 35

#training error
predict_train <- predict(knn_model_tuned, newdata = heart_train, type = "raw")
confusion_matrix <- confusionMatrix(factor(predict_train, levels = c("no", "yes")),
                                    factor(heart_train$num, levels = c("no", "yes")))
train_error <- 1 - (confusion_matrix$table[1,1] + confusion_matrix$table[2,2]) /
  sum(confusion_matrix$table)
print(train_error)
```

```
## [1] 0.1752717
```

```r
#training error: 0.1752717

#testing error
predict_test <- predict(knn_model_tuned, newdata = heart_test)
confusion_matrix <- confusionMatrix(predict_test, heart_test[, ncol(heart_test)])
test_error <- 1 - (confusion_matrix$table[1,1] + confusion_matrix$table[2,2]) /
  sum(confusion_matrix$table)
print(test_error)
```

```
## [1] 0.1847826
```

```r
#testing error:0.1847826

#confusion matrix & accuracy
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  54  15
##        yes 19  96
##
##                Accuracy : 0.8152
##                  95% CI : (0.7515, 0.8685)
##     No Information Rate : 0.6033
##     P-Value [Acc > NIR] : 5.296e-10
##
##                   Kappa : 0.6103
##
##  Mcnemar's Test P-Value : 0.6069
##
##             Sensitivity : 0.7397
##             Specificity : 0.8649
##          Pos Pred Value : 0.7826
##          Neg Pred Value : 0.8348
##              Prevalence : 0.3967
##          Detection Rate : 0.2935
##    Detection Prevalence : 0.3750
##       Balanced Accuracy : 0.8023
##
##        'Positive' Class : no
##
```

```r
# when k =26,  0.8152

#aucroc
roc_knn_tuned <- roc(heart_test$num, as.numeric(predict_test))
```

```
## Setting levels: control = no, case = yes
```

```
## Setting direction: controls < cases
```

```r
auc_knn_tuned <- auc(roc_knn_tuned)
print(auc_knn_tuned)
```

```
## Area under the curve: 0.8023
```

```r
#Area under the curve: 0.8023
```

Random Forest

```r
library(randomForest)
```

```
## randomForest 4.7-1.1

## 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
```

```r
library(e1071)
library(caret)

set.seed(123)

rf.hd <- randomForest(num ~., data = heart_train,importance=TRUE)

#training error
predict.train <- predict(rf.hd, newdata = heart_train)
train_error <- mean(predict.train != heart_train$num)
print(train_error) #training error = 0.001358696
```

```
## [1] 0.001358696
```

```r
#testing error
predict.test <- predict(rf.hd,newdata = heart_test)
test_error <- mean(predict.test != heart_test$num)
print(test_error) # testing error = 0.1793478
```

```
## [1] 0.1793478
```

```r
#confusion matrix & accuracy
x_test <- heart_test[, -which(names(heart_test) == "num")]
y_test <- heart_test$num
predictions <- predict(rf.hd, x_test)
confusion_matrix <- confusionMatrix(predictions, y_test, mode = "everything")
print(confusion_matrix) #accuracy = 0.8207, F1 = 0.7660
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  53  13
##        yes 20  98
##
##                Accuracy : 0.8207
##                  95% CI : (0.7575, 0.8732)
##     No Information Rate : 0.6033
##     P-Value [Acc > NIR] : 1.781e-10
##
##                   Kappa : 0.6191
##
##  Mcnemar's Test P-Value : 0.2963
##
##             Sensitivity : 0.7260
##             Specificity : 0.8829
##          Pos Pred Value : 0.8030
##          Neg Pred Value : 0.8305
##               Precision : 0.8030
##                  Recall : 0.7260
##                      F1 : 0.7626
##              Prevalence : 0.3967
##          Detection Rate : 0.2880
##    Detection Prevalence : 0.3587
##       Balanced Accuracy : 0.8045
##
##        'Positive' Class : no
##
```

```r
#aucroc
library(pROC)
predictions.prob <- predict(rf.hd, x_test, type = "prob")
roc_rf <- roc(response = y_test, predictor = predictions.prob[,2])
```

```
## Setting levels: control = no, case = yes
```

```
## Setting direction: controls < cases
```

```r
auc_rf <- auc(roc_rf)
print(auc_rf) #auc = 0.8803
```

```
## Area under the curve: 0.8803
```

Random Forest Model with K-Folds

```r
library(caret)

set.seed(123)

train_control <- trainControl(method = "cv", number = 10, search = "grid")
tune_grid <- expand.grid(mtry = c(2, 4, 6, 8))
rf_tuned <- train(num ~ ., data = heart_train, method = "rf",
                  metric = "Accuracy", trControl = train_control, tuneGrid = tune_grid)
#the final value used for the model was mtry = 2.

#training error
predict.train <- predict(rf_tuned, newdata = heart_train)
train_error <- mean(predict.train != heart_train$num)
print(train_error) #training error =  0.07608696
```

```
## [1] 0.07608696
```

```r
#testing error
predict.test <- predict(rf_tuned,newdata = heart_test)
test_error <- mean(predict.test != heart_test$num)
print(test_error) #testing error = 0.1847826
```

```
## [1] 0.1847826
```

```r
#confusion matrix & accuracy
predictions_tuned <- predict(rf_tuned, x_test)
confusion_matrix_tuned <- confusionMatrix(predictions_tuned, y_test, mode = "everything")
print(confusion_matrix_tuned) #accuracy = , F1 = 0.7445
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  54  15
##        yes 19  96
##
##                Accuracy : 0.8152
##                  95% CI : (0.7515, 0.8685)
##     No Information Rate : 0.6033
##     P-Value [Acc > NIR] : 5.296e-10
##
##                   Kappa : 0.6103
##
##  Mcnemar's Test P-Value : 0.6069
##
##             Sensitivity : 0.7397
##             Specificity : 0.8649
##          Pos Pred Value : 0.7826
##          Neg Pred Value : 0.8348
```

```
##               Precision : 0.7826
##                  Recall : 0.7397
##                      F1 : 0.7606
##              Prevalence : 0.3967
##          Detection Rate : 0.2935
##    Detection Prevalence : 0.3750
##       Balanced Accuracy : 0.8023
##
##        'Positive' Class : no
##
```

```r
#aucroc
library(pROC)
predictions.prob <- predict(rf_tuned, x_test, type = "prob")
roc_rf_tuned <- roc(response = y_test, predictor = predictions.prob[,2])
```

```
## Setting levels: control = no, case = yes
```

```
## Setting direction: controls < cases
```

```r
auc_rf_tuned <- auc(roc_rf_tuned)
print(auc_rf_tuned) #auc = 0.8956
```

```
## Area under the curve: 0.8956
```

Gradient Boosting Model

```
library(gbm)
```

```
## Loaded gbm 2.1.9
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```
library(caret)

set.seed(123)

unique(heart_train$num)
```

```
## [1] no  yes
## Levels: no yes
```

```
heart_train$num <- ifelse(heart_train$num == "no", 0, 1)
unique(heart_test$num)
```

```
## [1] no  yes
## Levels: no yes
```

```
heart_test$num <- ifelse(heart_test$num == "no", 0, 1)

boost.hd <- gbm(num ~ ., data = heart_train, distribution = "bernoulli", n.trees = 1000, interaction.de

#training error
predict_train <- predict(boost.hd, n.trees = 1000, type = "response", newdata = heart_train)
predicted_train_classes <- ifelse(predict_train > 0.5, 1, 0)
train_error <- mean(predicted_train_classes != heart_train$num)
print(train_error) #training error = 0.1358696
```

```
## [1] 0.1358696
```

```
#testing error
predict_test <- predict(boost.hd, n.trees = 1000, type = "response", newdata = heart_test)
predicted_test_classes <- ifelse(predict_test > 0.5, 1, 0)
test_error <- mean(predicted_test_classes != heart_test$num)
print(test_error) #testing error = 0.1902174
```

```
## [1] 0.1902174
```

```
#confusion matrix & accuracy
predictions_test <- factor(predicted_test_classes, levels = c(0, 1), labels = c("no", "yes"))
confusion_matrix_test <- confusionMatrix(predictions_test, factor(heart_test$num, levels = c(0, 1), lab
print(confusion_matrix_test) #0.8098, F1 = 0.7586
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  55  17
##        yes 18  94
##
##                Accuracy : 0.8098
##                  95% CI : (0.7455, 0.8638)
##     No Information Rate : 0.6033
##     P-Value [Acc > NIR] : 1.52e-09
##
##                   Kappa : 0.6017
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.7534
##             Specificity : 0.8468
##          Pos Pred Value : 0.7639
##          Neg Pred Value : 0.8393
##               Precision : 0.7639
##                  Recall : 0.7534
##                      F1 : 0.7586
##              Prevalence : 0.3967
##          Detection Rate : 0.2989
##    Detection Prevalence : 0.3913
##       Balanced Accuracy : 0.8001
##
##        'Positive' Class : no
##
```

```r
#aucroc
library(pROC)
predictions.prob <- predict(boost.hd, newdata = x_test, type = "response")
```

```
## Using 154 trees...
```

```r
y_test <- factor(y_test, levels = c("no", "yes"))
roc_gbm <- roc(response = y_test, predictor = predictions.prob)
```

```
## Setting levels: control = no, case = yes
```

```
## Setting direction: controls < cases
```

```r
auc_gbm <- auc(roc_gbm)
print(auc_gbm) #auc = 0.8999
```

```
## Area under the curve: 0.8999
```

Using K-folds to Improve Gradient Boosting Model

```r
library(caret)
library(pROC)

set.seed(123)

train_control <- trainControl(method = "cv", number = 10, search = "grid")
tune_grid <- expand.grid(n.trees = c(150), interaction.depth = c(1, 3, 5),
                         shrinkage = c(0.01), n.minobsinnode = c(5, 10, 15))

heart_train$num <- as.factor(ifelse(heart_train$num == 0, "no", "yes"))
boost_tuned <- train(num ~ ., data = heart_train, method = "gbm",
                     metric = "Accuracy", trControl = train_control, tuneGrid = tune_grid,
                     verbose = FALSE)

#training error
predict_train <- predict(boost_tuned, n.trees = 1000, type = "prob", newdata = heart_train)
predicted_train_classes <- ifelse(predict_train[, 2] > 0.5, 1, 0)
train_error <- mean(predicted_train_classes != heart_test$num)
print(train_error) #training error = 0.4959239
```

```
## [1] 0.4959239
```

```r
#testing error
predict_test <- predict(boost_tuned, n.trees = 1000, type = "prob", newdata = heart_test)
predicted_test_classes <- ifelse(predict_test[, 2] > 0.5, 1, 0)
test_error <- mean(predicted_test_classes != heart_test$num)
print(test_error) #testing error = 0.173913
```

```
## [1] 0.173913
```

```r
#confusion matrix & accuracy
predictions_test <- factor(predicted_test_classes, levels = c(0, 1), labels = c("no", "yes"))
conf_matrix_test <- confusionMatrix(predictions_test, y_test, mode = "everything")
print(conf_matrix_test) #accuracy = 0.8261, F1 : 0.7681
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no  53  12
##        yes 20  99
##
##                Accuracy : 0.8261
##                  95% CI : (0.7634, 0.8779)
##     No Information Rate : 0.6033
##     P-Value [Acc > NIR] : 5.78e-11
##
##                   Kappa : 0.6297
##
```

```
##  Mcnemar's Test P-Value : 0.2159
##
##             Sensitivity : 0.7260
##             Specificity : 0.8919
##          Pos Pred Value : 0.8154
##          Neg Pred Value : 0.8319
##               Precision : 0.8154
##                  Recall : 0.7260
##                      F1 : 0.7681
##              Prevalence : 0.3967
##          Detection Rate : 0.2880
##    Detection Prevalence : 0.3533
##       Balanced Accuracy : 0.8090
##
##        'Positive' Class : no
##
```

```r
#aucroc
library(pROC)
predictions.prob <- predict(boost_tuned, x_test, type = "prob")
roc_gbm_tuned <- roc(response = y_test, predictor = predictions.prob[,2])
```

```
## Setting levels: control = no, case = yes
```

```
## Setting direction: controls < cases
```

```r
auc_gbm_tuned <- auc(roc_gbm_tuned)
print(auc_gbm_tuned) #auc = 0.9025
```

```
## Area under the curve: 0.9025
```

Support Vector Machines (SVM) Model

```r
library(e1071)
library(caret)
set.seed(123)

# Train the SVM model, Use radial kernel
svm_model <- svm(num ~ ., data = heart_train, kernel = "radial")
print(svm_model)
```

```
##
## Call:
## svm(formula = num ~ ., data = heart_train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  358
```

```r
# Extract sigma and cost from the model
svm_model$cost
```

```
## [1] 1
```

```r
gamma <- svm_model$gamma
gamma
```

```
## [1] 0.06666667
```

```r
# Evaluate the model
predictions_svm <- predict(svm_model, newdata=heart_test,levels = levels(heart_test$num))
predictions_svm <- factor(predictions_svm, levels = c("no", "yes"))
y_test <- factor(heart_test$num, levels = c("no", "yes"))
confusion_matrix_svm <- confusionMatrix(predictions_svm, y_test, mode = "everything")
print(confusion_matrix_svm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no   0   0
##        yes  0   0
##
##                Accuracy : NaN
##                  95% CI : (NA, NA)
##     No Information Rate : NA
##     P-Value [Acc > NIR] : NA
```

```
##
##                  Kappa : NaN
##
##  Mcnemar's Test P-Value : NA
##
##            Sensitivity :  NA
##            Specificity :  NA
##         Pos Pred Value :  NA
##         Neg Pred Value :  NA
##              Precision :  NA
##                 Recall :  NA
##                     F1 :  NA
##             Prevalence : NaN
##         Detection Rate : NaN
##   Detection Prevalence : NaN
##      Balanced Accuracy :  NA
##
##       'Positive' Class : no
##
```

```r
# Accuracy : 0.8043
#F1 : 0.7353

# training error
train_predict_svm <- predict(svm_model, heart_train)
train_error_svm <- mean(train_predict_svm != heart_train$num)
train_error_svm
```

```
## [1] 0.1576087
```

```r
# training error :0.1576087

# test error
test_predict_svm <- predict(svm_model, heart_test)
test_error_svm <- mean(test_predict_svm != heart_test$num)
test_error_svm
```

```
## [1] 1
```

```r
# test error : 0.1956522

#roc auc
library(pROC)
roc_svm <- roc(heart_test$num, as.numeric(test_predict_svm))
```

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

```
## Setting direction: controls < cases
```

```r
auc_svm <- auc(roc_svm)
auc_svm
```

```
## Area under the curve: 0.7839
```

```
#Area under the curve: 0.7839
```

Using K-folds to Improve Support Vector Machines (SVM) Model

```
# Set up the grid of hyperparameters to tune
sigma_values <- seq(0,0.1, by = 0.002)
cost_values <- c(0.5, 1, 1.5)
tune_grid_svm <- expand.grid(sigma = sigma_values, C = cost_values)


# Set up the training control for K-fold cross-validation
ctrl_svm <- trainControl(method = "cv",number = 5)

# Perform hyperparameter tuning using K-fold cross-validation
svm_model_tuned <- train(num ~ .,data = heart_train, method = "svmRadial", trControl = ctrl_svm,tuneGri
# Print the tuned SVM model
print(svm_model_tuned)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 736 samples
##  10 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 589, 589, 589, 589, 588
## Resampling results across tuning parameters:
##
##   sigma  C    Accuracy   Kappa
##   0.000  0.5  0.5407612  0.0000000
##   0.000  1.0  0.5407612  0.0000000
##   0.000  1.5  0.5407612  0.0000000
##   0.002  0.5  0.7893455  0.5743973
##   0.002  1.0  0.7975087  0.5910029
##   0.002  1.5  0.7974995  0.5906716
##   0.004  0.5  0.7975087  0.5910029
##   0.004  1.0  0.7988509  0.5931324
##   0.004  1.5  0.8029325  0.6013323
##   0.006  0.5  0.7947785  0.5852026
##   0.006  1.0  0.8015720  0.5986679
##   0.006  1.5  0.8015812  0.5992111
##   0.008  0.5  0.7988601  0.5931175
##   0.008  1.0  0.8002206  0.5961881
##   0.008  1.5  0.8097352  0.6156824
##   0.010  0.5  0.7974995  0.5902530
##   0.010  1.0  0.8015812  0.5990547
##   0.010  1.5  0.8110866  0.6184658
##   0.012  0.5  0.8029417  0.6009161
##   0.012  1.0  0.8083747  0.6126917
##   0.012  1.5  0.8110958  0.6182579
##   0.014  0.5  0.8029417  0.6010919
##   0.014  1.0  0.8083747  0.6128297
##   0.014  1.5  0.8110774  0.6178356
```

```
##   0.016  0.5  0.8015812  0.5985867
##   0.016  1.0  0.8110958  0.6181027
##   0.016  1.5  0.8124379  0.6209230
##   0.018  0.5  0.8015812  0.5984429
##   0.018  1.0  0.8110866  0.6180498
##   0.018  1.5  0.8137893  0.6234790
##   0.020  0.5  0.8029417  0.6012711
##   0.020  1.0  0.8124471  0.6211632
##   0.020  1.5  0.8110682  0.6177809
##   0.022  0.5  0.8043023  0.6039423
##   0.022  1.0  0.8165104  0.6290171
##   0.022  1.5  0.8110682  0.6179681
##   0.024  0.5  0.8056628  0.6070367
##   0.024  1.0  0.8151590  0.6262396
##   0.024  1.5  0.8083471  0.6124692
##   0.026  0.5  0.8056628  0.6069988
##   0.026  1.0  0.8124379  0.6205485
##   0.026  1.5  0.8083563  0.6123533
##   0.028  0.5  0.8083839  0.6123212
##   0.028  1.0  0.8070050  0.6097826
##   0.028  1.5  0.8083655  0.6124147
##   0.030  0.5  0.8097352  0.6151171
##   0.030  1.0  0.8097261  0.6153232
##   0.030  1.5  0.8110774  0.6181602
##   0.032  0.5  0.8110958  0.6179839
##   0.032  1.0  0.8070050  0.6099904
##   0.032  1.5  0.8083655  0.6126361
##   0.034  0.5  0.8083839  0.6126870
##   0.034  1.0  0.8042931  0.6043271
##   0.034  1.5  0.8097261  0.6152811
##   0.036  0.5  0.8110958  0.6181984
##   0.036  1.0  0.8042931  0.6043271
##   0.036  1.5  0.8097261  0.6152811
##   0.038  0.5  0.8056536  0.6067736
##   0.038  1.0  0.8056444  0.6069845
##   0.038  1.5  0.8110866  0.6181106
##   0.040  0.5  0.8070050  0.6094309
##   0.040  1.0  0.8042931  0.6044540
##   0.040  1.5  0.8097261  0.6154502
##   0.042  0.5  0.8070050  0.6094053
##   0.042  1.0  0.8043023  0.6044361
##   0.042  1.5  0.8083655  0.6127678
##   0.044  0.5  0.8070050  0.6094317
##   0.044  1.0  0.8070233  0.6099172
##   0.044  1.5  0.8110866  0.6182869
##   0.046  0.5  0.8029325  0.6011603
##   0.046  1.0  0.8083839  0.6125624
##   0.046  1.5  0.8138077  0.6233904
##   0.048  0.5  0.8029325  0.6011603
##   0.048  1.0  0.8083839  0.6123979
##   0.048  1.5  0.8124563  0.6202812
##   0.050  0.5  0.8029325  0.6011773
##   0.050  1.0  0.8083839  0.6123979
##   0.050  1.5  0.8110958  0.6174278
```

```
## 0.052 0.5 0.8029325 0.6011773
## 0.052 1.0 0.8070233 0.6095311
## 0.052 1.5 0.8110958 0.6174278
## 0.054 0.5 0.8015720 0.5983246
## 0.054 1.0 0.8070233 0.6093283
## 0.054 1.5 0.8097261 0.6145898
## 0.056 0.5 0.8002114 0.5954440
## 0.056 1.0 0.8083747 0.6121242
## 0.056 1.5 0.8097352 0.6148064
## 0.058 0.5 0.8015720 0.5978769
## 0.058 1.0 0.8056536 0.6066634
## 0.058 1.5 0.8083747 0.6119710
## 0.060 0.5 0.8015720 0.5978769
## 0.060 1.0 0.8070142 0.6093024
## 0.060 1.5 0.8070142 0.6091299
## 0.062 0.5 0.8015720 0.5978769
## 0.062 1.0 0.8056536 0.6066258
## 0.062 1.5 0.8056628 0.6066570
## 0.064 0.5 0.8042839 0.6031906
## 0.064 1.0 0.8056536 0.6062970
## 0.064 1.5 0.8056628 0.6066570
## 0.066 0.5 0.8029325 0.6006256
## 0.066 1.0 0.8015812 0.5981536
## 0.066 1.5 0.8056628 0.6066570
## 0.068 0.5 0.8042931 0.6033103
## 0.068 1.0 0.8029417 0.6008413
## 0.068 1.5 0.8043023 0.6042267
## 0.070 0.5 0.8042931 0.6033103
## 0.070 1.0 0.8029417 0.6008413
## 0.070 1.5 0.8056628 0.6068854
## 0.072 0.5 0.8015720 0.5980202
## 0.072 1.0 0.8043023 0.6034860
## 0.072 1.5 0.8070233 0.6097517
## 0.074 0.5 0.8015720 0.5980202
## 0.074 1.0 0.8043023 0.6034860
## 0.074 1.5 0.8070233 0.6099784
## 0.076 0.5 0.8015720 0.5980202
## 0.076 1.0 0.8043023 0.6034860
## 0.076 1.5 0.8070233 0.6099784
## 0.078 0.5 0.8015720 0.5980202
## 0.078 1.0 0.8056628 0.6061793
## 0.078 1.5 0.8056720 0.6071890
## 0.080 0.5 0.8029325 0.6007024
## 0.080 1.0 0.8043023 0.6036789
## 0.080 1.5 0.8056628 0.6069812
## 0.082 0.5 0.8002206 0.5950578
## 0.082 1.0 0.8043023 0.6036789
## 0.082 1.5 0.8043023 0.6041222
## 0.084 0.5 0.8002206 0.5952613
## 0.084 1.0 0.8043023 0.6038386
## 0.084 1.5 0.8002206 0.5958941
## 0.086 0.5 0.7988601 0.5926222
## 0.086 1.0 0.8056536 0.6064740
## 0.086 1.5 0.7988601 0.5932357
```

```
##   0.088  0.5  0.7988601  0.5926222
##   0.088  1.0  0.8042931  0.6036207
##   0.088  1.5  0.7988601  0.5932489
##   0.090  0.5  0.7974995  0.5897805
##   0.090  1.0  0.8042931  0.6036207
##   0.090  1.5  0.7974995  0.5905838
##   0.092  0.5  0.8002206  0.5950651
##   0.092  1.0  0.8042931  0.6036207
##   0.092  1.5  0.7961390  0.5879383
##   0.094  0.5  0.8002206  0.5950651
##   0.094  1.0  0.8029325  0.6007678
##   0.094  1.5  0.7961390  0.5879383
##   0.096  0.5  0.7988693  0.5924023
##   0.096  1.0  0.8029325  0.6010029
##   0.096  1.5  0.7974995  0.5906034
##   0.098  0.5  0.7988693  0.5924023
##   0.098  1.0  0.8015720  0.5983096
##   0.098  1.5  0.7974995  0.5906034
##   0.100  0.5  0.7975087  0.5895547
##   0.100  1.0  0.8042839  0.6036132
##   0.100  1.5  0.7988601  0.5934507
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.022 and C = 1.
```

```r
#The final values used for the model were sigma = 0.032 and C = 1.5..

# Make predictions on the test set using the tuned model
predictions_svm_tuned <- predict(svm_model_tuned, heart_test)

# Evaluate the tuned model

predictions_svm2 <- predict(svm_model_tuned, newdata=heart_test,levels = levels(heart_test$num))
predictions_svm2 <- factor(predictions_svm_tuned, levels = c("no", "yes"))
y_test <- factor(heart_test$num, levels = c("no", "yes"))
confusion_matrix_svm2 <- confusionMatrix(predictions_svm2, y_test, mode = "everything")
print(confusion_matrix_svm2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no yes
##        no   0   0
##        yes  0   0
##
##                Accuracy : NaN
##                  95% CI : (NA, NA)
##     No Information Rate : NA
##     P-Value [Acc > NIR] : NA
##
##                   Kappa : NaN
##
##  Mcnemar's Test P-Value : NA
##
```

```
##             Sensitivity :  NA
##             Specificity :  NA
##          Pos Pred Value :  NA
##          Neg Pred Value :  NA
##               Precision :  NA
##                  Recall :  NA
##                      F1 :  NA
##              Prevalence : NaN
##          Detection Rate : NaN
##    Detection Prevalence : NaN
##       Balanced Accuracy :  NA
##
##        'Positive' Class : no
##
```

```r
# Accuracy :  0.8043
#F1 : 0.7465
#The final values used for the model were sigma = 0.022 and C = 1.

# training error
train_predict_svm2 <- predict(svm_model_tuned, heart_train)
train_error_svm2 <- mean(train_predict_svm2 != heart_train$num)
train_error_svm2
```

```
## [1] 0.1576087
```

```r
# training error: 0.1576087

# test error
test_predict_svm2 <- predict(svm_model_tuned, heart_test)
test_error_svm2 <- mean(test_predict_svm2 != heart_test$num)
test_error_svm2
```

```
## [1] 1
```

```r
# test error : 0.1902174

predict_test_svm2 <- as.numeric(predict(svm_model_tuned, heart_test, probability = TRUE))
roc_svm2 <- roc(heart_test$num, predict_test_svm2)
```

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

```
## Setting direction: controls < cases
```

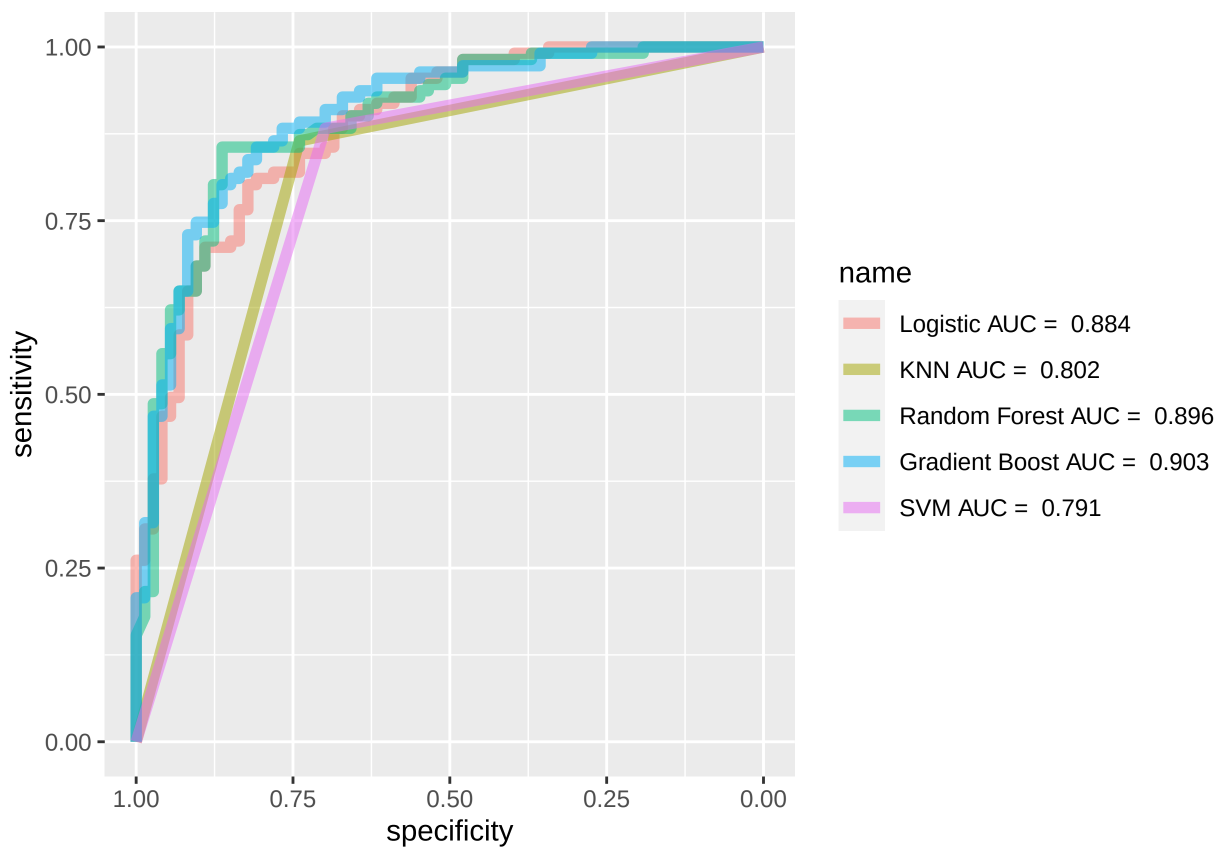
```r
auc_svm2 <- auc(roc_svm2)
auc_svm2
```

```
## Area under the curve: 0.7908
```

```
#Area under the curve:  0.7817
```

Comparison of AURROC for Final Models

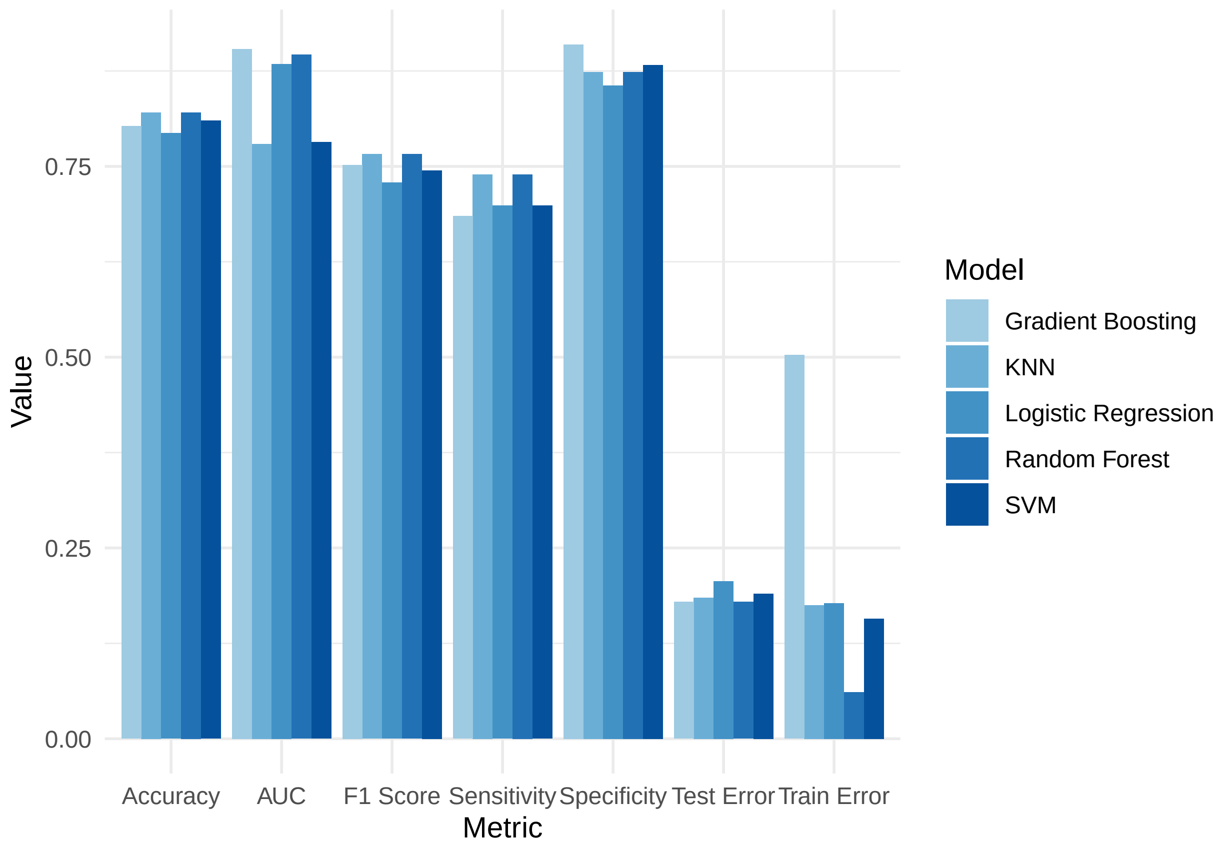
```
library(pROC)


rocobjs <- list(Logistic = roc_logistic, KNN = roc_knn_tuned, RandomForest = roc_rf_tuned, GradientBoos
methods_auc <- paste(c("Logistic", "KNN", "Random Forest", "Gradient Boost", "SVM"),
                     "AUC = ",
                     round(c(auc_logistic, auc_knn_tuned, auc_rf_tuned, auc_gbm_tuned,auc_svm2), 3))

ggroc(rocobjs, size = 2, alpha = 0.5) +
  scale_color_discrete(labels = methods_auc)
```



Bar Charts Comparing Final Models

```
library(ggplot2)

#initial models
logistic_metrics <- c("Train Error", "Test Error", "Accuracy", "Sensitivity", "Specificity", "F1 Score"
logistic_values <- c(0.1779, 0.2065, 0.7935, 0.6986, 0.8559, 0.7286, 0.8841)
```

```
knn_metrics <- c("Train Error", "Test Error", "Accuracy", "Sensitivity", "Specificity", "F1 Score", "AUC
knn_values <- c(0.1752, 0.1847, 0.8207, 0.7397, 0.8739, 0.7660, 0.7796)

rf_metrics <- c("Train Error", "Test Error", "Accuracy", "Sensitivity", "Specificity", "F1 Score", "AUC
rf_values <- c(0.0611, 0.1793, 0.8207, 0.7397, 0.8739, 0.7660, 0.8963)

gbm_metrics <- c("Train Error", "Test Error", "Accuracy", "Sensitivity", "Specificity", "F1 Score", "AUC
gbm_values <- c(0.5027, 0.1793, 0.8027, 0.6849, 0.9099, 0.7519, 0.904)

svm_metrics <- c("Train Error", "Test Error", "Accuracy", "Sensitivity", "Specificity", "F1 Score", "AUC
svm_values <- c(0.1576, 0.1902, 0.8098, 0.6986, 0.8829, 0.7445, 0.7817)

data <- data.frame(
  Model = rep(c("Logistic Regression", "KNN", "Random Forest", "Gradient Boosting", "SVM"), each = 7),
  Metric = rep(logistic_metrics, times = 5),
  Value = c(logistic_values, knn_values, rf_values, gbm_values, svm_values)
)

colors <- c("#9ecae2", "#6aaed6", "#4292c6", "#2271b5", "#05519c")
ggplot(data, aes(x = Metric, y = Value, fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_manual(values = colors) +
  theme_minimal() +
  labs(x = "Metric", y = "Value")
```



"'