Assignment 2

Khoi Mai Tieu

Contents

Problem 1. Regression

```
data <- read.csv("qsar_aquatic_toxicity.csv", sep = ";", header = FALSE)

# Since the raw data does not have column names, we will assign them manually
names(data) <- c(
    "TPSA",
    "SAacc",
    "H050",
    "MLOGP",
    "RDCHI",
    "GATS1p",
    "nN",
    "C040",
    "LC50"
)

head(data)
```

```
##     TPSA   SAacc H050 MLOGP RDCHI GATS1p nN C040  LC50
## 1   0.00   0.000    0 2.419 1.225  0.667  0    0 3.740
## 2   0.00   0.000    0 2.638 1.401  0.632  0    0 4.330
## 3   9.23  11.000    0 5.799 2.930  0.486  0    0 7.019
## 4   9.23  11.000    0 5.453 2.887  0.495  0    0 6.723
## 5   9.23  11.000    0 4.068 2.758  0.695  0    0 5.979
## 6 215.34 327.629    3 0.189 4.677  1.333  0    4 6.064
```

a. Data splitting

We split the data into a training and a test set, with approximately 2/3 and 1/3 of the observations, respectively.

```
# Use 2/3 of dataset as training set and remaining 1/3 as testing set
set.seed(123)
sample <- sample.split(data$LC50, SplitRatio = 2/3)
train  <- subset(data, sample == TRUE)
test   <- subset(data, sample == FALSE)
```

```
cat("Dimension of Training Set:", paste(dim(train), collapse = "x"), "\nDimension of Test Set:", paste(
```

```
## Dimension of Training Set: 364x9
## Dimension of Test Set: 182x9
```

(i) Original Model

First, we will fit a linear regression model on the training data using all the predictors.

```
# To make sure we use the same split in (i) and (ii)
train_i = train
test_i = test
```

The initial linear regression model shows significant predictors including TPSA, SAacc, MLOGP, RDCHI, GATS1p, and nN based on p-values less than 0.05. However, H050 and C040 do not appear to have a significant impact.

```
# Fit linear regression model on training data
model <- lm(LC50 ~ ., data=train_i)

summary(model)
```

```
##
## Call:
## lm(formula = LC50 ~ ., data = train_i)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8548 -0.8166 -0.1830  0.6771  4.8867
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.629264   0.312580   8.411 1.00e-15 ***
## TPSA         0.027092   0.003336   8.121 7.74e-15 ***
## SAacc       -0.015959   0.002652  -6.017 4.42e-09 ***
## H050        -0.003879   0.076369  -0.051 0.959522
## MLOGP        0.400783   0.081760   4.902 1.45e-06 ***
## RDCHI        0.654990   0.177787   3.684 0.000265 ***
## GATS1p      -0.589994   0.195299  -3.021 0.002702 **
## nN          -0.199466   0.059602  -3.347 0.000906 ***
## C040        -0.046002   0.091165  -0.505 0.614156
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.219 on 355 degrees of freedom
## Multiple R-squared:  0.5024, Adjusted R-squared:  0.4912
## F-statistic:  44.8 on 8 and 355 DF,  p-value: < 2.2e-16
```

We will predict the LC50 values on the training and test datasets to evaluate the model using these metrics:

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R-squared

```
# Predict on training and test datasets
pred_train <- predict(model, newdata=train_i)
pred_test <- predict(model, newdata=test_i)
```

```r
# Adding predictions columns to the datasets
train_i$predicted_LC50 <- pred_train
test_i$predicted_LC50 <- pred_test
```

```r
# Evaluate model: calculate MSE, RMSE, and R-squared for training and test sets
mse_train <- mean((train_i$LC50 - train_i$predicted_LC50)^2)
rmse_train <- sqrt(mse_train)
r2_train <- 1 - (sum((train_i$LC50 - train_i$predicted_LC50)^2) /
            sum((train_i$LC50 - mean(train_i$LC50))^2))

mse_test <- mean((test_i$LC50 - test_i$predicted_LC50)^2)
rmse_test <- sqrt(mse_test)
r2_test <- 1 - (sum((test_i$LC50 - test_i$predicted_LC50)^2) /
            sum((test_i$LC50 - mean(test_i$LC50))^2))
```

We have the following result for the model on the training and test set as follows. We see that the training and test set metrics are reasonably close, with R-squared values indicating that approximately 50% of the variance is explained by the model in the training set and around 43% in the test set, suggesting the model generalizes fairly well.

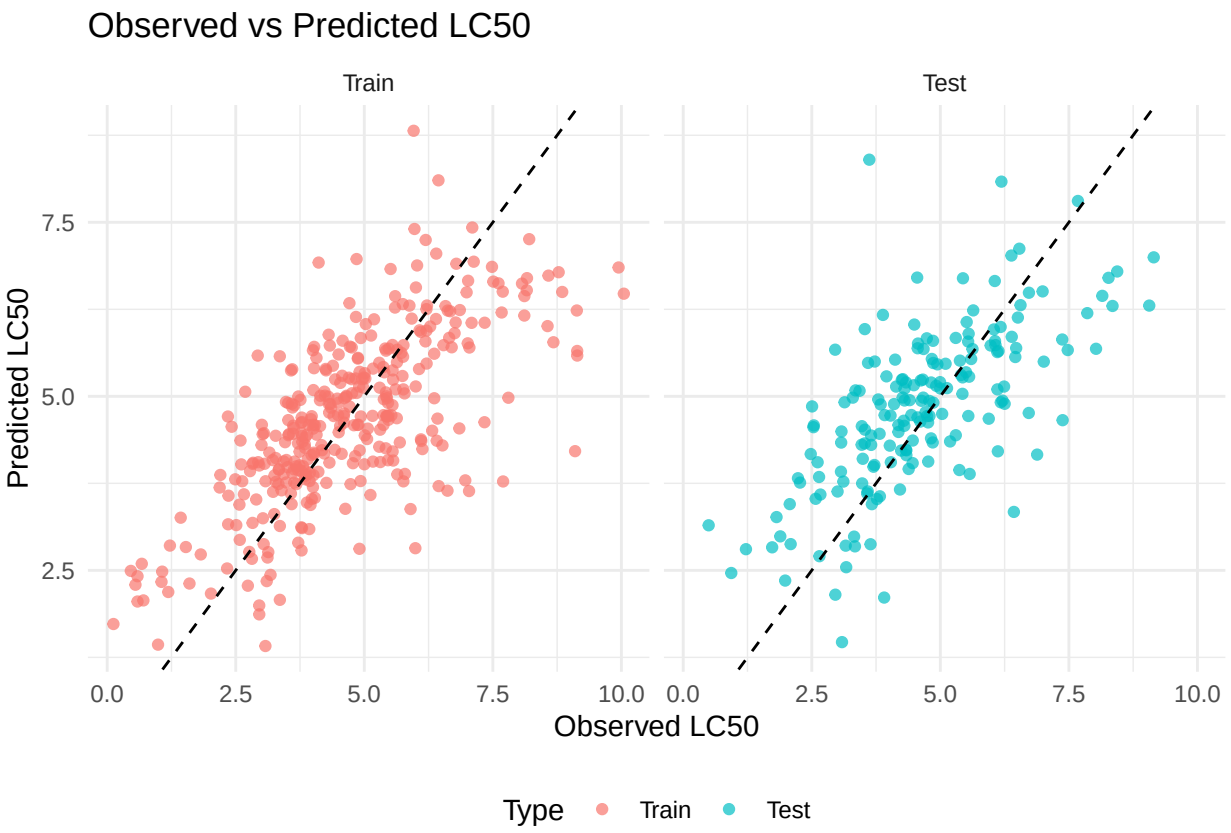
```r
cat(paste0(
  "Training Metrics:\n",
  "MSE (Train): ", mse_train, "\n",
  "RMSE (Train): ", rmse_train, "\n",
  "R-squared (Train): ", r2_train, "\n\n",

  "Test Metrics:\n",
  "MSE (Test): ", mse_test, "\n",
  "RMSE (Test): ", rmse_test, "\n",
  "R-squared (Test): ", r2_test, "\n"
))
```

```
## Training Metrics:
## MSE (Train): 1.44990640082018
## RMSE (Train): 1.20412059230801
## R-squared (Train): 0.502397645581479
##
## Test Metrics:
## MSE (Test): 1.40224882922927
## RMSE (Test): 1.18416587910194
## R-squared (Test): 0.433587696937759
```

Plotting the observed vs predicted LC50 values for the training and test sets, we can see that the model generally performs well, with most points falling close to the dashed line (y=x) indicating perfect predictions.

```r
# Combine data for plotting
train_i$Type <- 'Train'
test_i$Type <- 'Test'
combined_data <- rbind(train_i, test_i)

combined_data$Type <- factor(combined_data$Type, levels = c('Train', 'Test'))
```

```r
# Plotting observed vs predicted LC50 values
ggplot(combined_data, aes(x = LC50, y = predicted_LC50, color = Type)) +
  geom_point(alpha = 0.7) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(title = "Observed vs Predicted LC50", x = "Observed LC50", y = "Predicted LC50") +
  theme_minimal() +
  facet_wrap(~Type) +
  theme(legend.position = "bottom")
```



(ii). Dummy encoding

We can see that in our data set, there are 3 count variables (H050, nN, C040) that represent the number of specific atoms in the chemical compounds. We will transform these variables using a 0/1 dummy encoding, where 0 represents the absence of the specific atom, and 1 represents the presence of the specific atoms. In this case, I suppose that the model will perform a litter bit worse than the original model because we lose some information by transforming the count variables into binary variables. On the other hand, it may help to reduce overfitting because it simplifies the model.

```r
# To make sure we use the same split in (i) and (ii)
train_ii = train
test_ii = test
```

```r
# Transform 3 count variables (H050, nN, C040) into 0/1 in train and test datasets
```

```r
train_ii$H050 <- ifelse(train_ii$H050 > 0, 1, 0)
train_ii$nN <- ifelse(train_ii$nN > 0, 1, 0)
train_ii$C040 <- ifelse(train_ii$C040 > 0, 1, 0)

test_ii$H050 <- ifelse(test_ii$H050 > 0, 1, 0)
test_ii$nN <- ifelse(test_ii$nN > 0, 1, 0)
test_ii$C040 <- ifelse(test_ii$C040 > 0, 1, 0)
```

```r
head(train_ii)
```

```
##      TPSA   SAacc H050 MLOGP RDCHI GATS1p nN C040  LC50
## 1    0.00   0.000    0 2.419 1.225  0.667  0    0 3.740
## 3    9.23  11.000    0 5.799 2.930  0.486  0    0 7.019
## 6  215.34 327.629    1 0.189 4.677  1.333  0    1 6.064
## 7    9.23  11.000    0 2.723 2.321  1.165  0    0 7.337
## 9    0.00   0.000    0 2.067 1.800  1.250  0    0 3.941
## 10   0.00   0.000    0 2.746 1.667  1.400  0    0 3.809
```

After transforming the count variables into binary variables, we will fit a linear regression model on the training data using all the predictors.

```r
# Fit linear regression model on transformed training data
model_transform_dummy <- lm(LC50 ~ ., data = train_ii)

summary(model_transform_dummy)
```

```
##
## Call:
## lm(formula = LC50 ~ ., data = train_ii)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0873 -0.8306 -0.1303  0.6571  5.0526
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.693545   0.315949   8.525 4.44e-16 ***
## TPSA         0.023267   0.003365   6.914 2.20e-11 ***
## SAacc       -0.014731   0.002407  -6.121 2.46e-09 ***
## H050        -0.090233   0.161558  -0.559  0.57684
## MLOGP        0.436885   0.082632   5.287 2.18e-07 ***
## RDCHI        0.553158   0.180181   3.070  0.00231 **
## GATS1p      -0.539057   0.190296  -2.833  0.00488 **
## nN           0.018072   0.156479   0.115  0.90812
## C040        -0.124928   0.169094  -0.739  0.46051
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.239 on 355 degrees of freedom
## Multiple R-squared:  0.4864, Adjusted R-squared:  0.4749
## F-statistic: 42.03 on 8 and 355 DF,  p-value: < 2.2e-16
```

```r
# Predict on training and test datasets
pred_train_transform_dummy <- predict(model, newdata=train_ii)
pred_test_transform_dummy <- predict(model, newdata=test_ii)
```

```r
# Adding predictions columns to the datasets
train_ii$predicted_LC50 <- pred_train_transform_dummy
test_ii$predicted_LC50 <- pred_test_transform_dummy
```

Based on the results, we see that R^2 values dropped from 43% in the original model to around 38% after the transformation. For MSE and RMSE in test set, it increased slightly from 1.18 to 1.53 and 1.18 to 1.23, respectively.

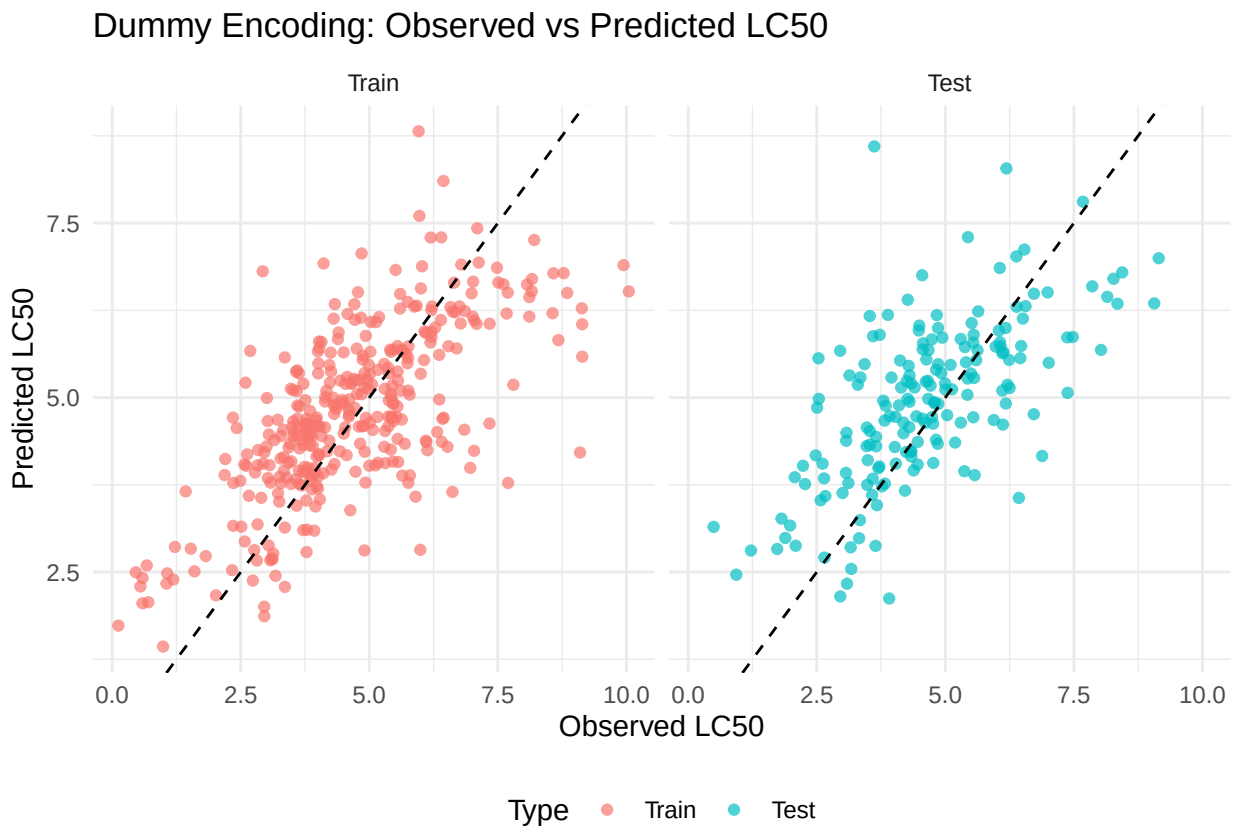
```r
# Evaluate model: calculate MSE, RMSE, and R-squared for training and test sets
mse_train_transform_dummy <- mean((train_ii$LC50 - train_ii$predicted_LC50)^2)
rmse_train_transform_dummy <- sqrt(mse_train_transform_dummy)
r2_train_transform_dummy <- 1 - (sum((train_ii$LC50 - train_ii$predicted_LC50)^2) / sum((train_ii$LC50 -

mse_test_transform_dummy <- mean((test_ii$LC50 - test_ii$predicted_LC50)^2)
rmse_test_transform_dummy <- sqrt(mse_test_transform_dummy)
r2_test_transform_dummy <- 1 - (sum((test_ii$LC50 - test_ii$predicted_LC50)^2) / sum((test_ii$LC50 - mea
```

```r
cat(paste0(
  "Training Metrics:\n",
  "MSE (Train): ", mse_train_transform_dummy, "\n",
  "RMSE (Train): ", rmse_train_transform_dummy, "\n",
  "R-squared (Train): ", r2_train_transform_dummy, "\n\n",

  "Test Metrics:\n",
  "MSE (Test): ", mse_test_transform_dummy, "\n",
  "RMSE (Test): ", rmse_test_transform_dummy, "\n",
  "R-squared (Test): ", r2_test_transform_dummy, "\n"
))
```

```
## Training Metrics:
## MSE (Train): 1.53935201233042
## RMSE (Train): 1.24070625545711
## R-squared (Train): 0.471700252387877
##
## Test Metrics:
## MSE (Test): 1.53043849004967
## RMSE (Test): 1.23710892408457
## R-squared (Test): 0.381807870490008
```

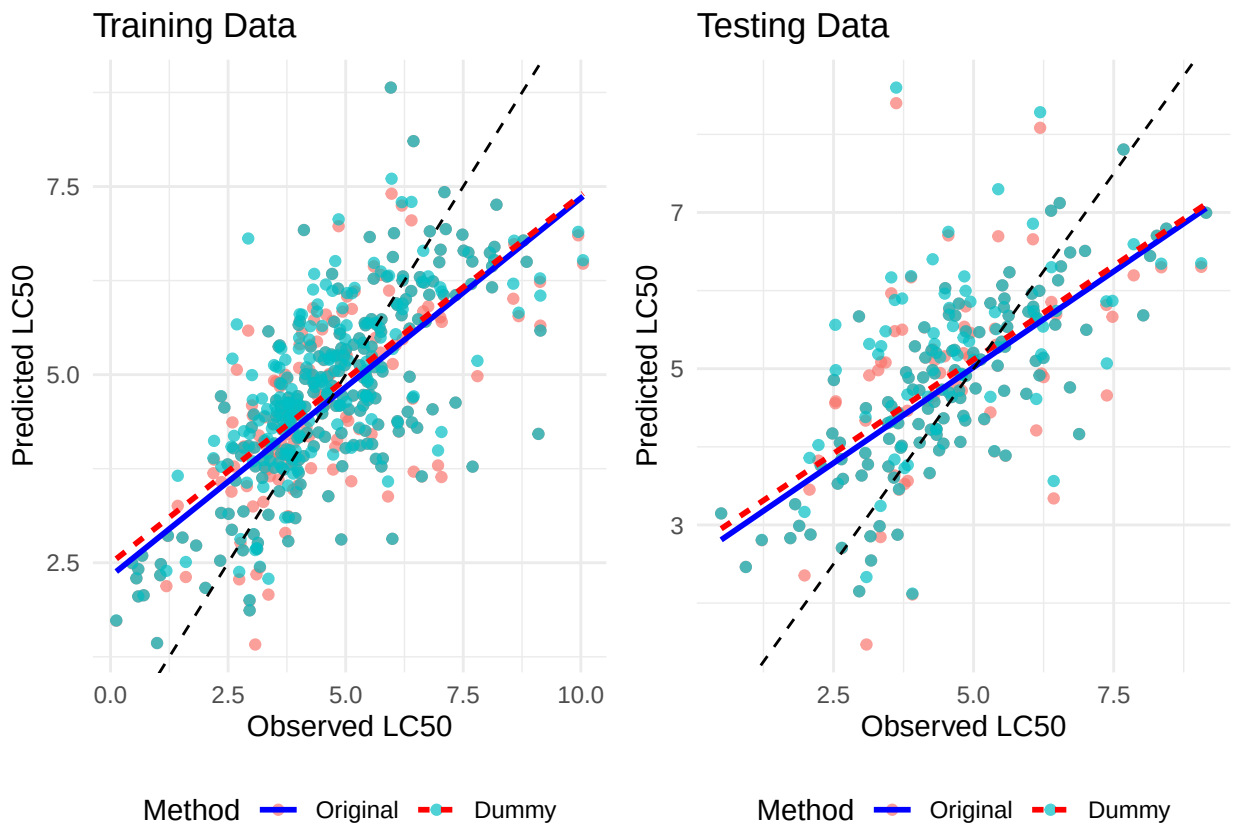
```r
# Combine data for plotting
train_ii$Type <- 'Train'
test_ii$Type <- 'Test'
combined_data <- rbind(train_ii, test_ii)

combined_data$Type <- factor(combined_data$Type, levels = c('Train', 'Test'))

# Plotting observed vs predicted LC50 values
ggplot(combined_data, aes(x = LC50, y = predicted_LC50, color = Type)) +
```

```
  geom_point(alpha = 0.7) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(title = "Dummy Encoding: Observed vs Predicted LC50",
       x = "Observed LC50",
       y = "Predicted LC50") +
  theme_minimal() +
  facet_wrap(~Type) +
  theme(legend.position = "bottom")
```




```
# Prepare combined data
train_combined <- train_i[, c("LC50", "predicted_LC50")]
train_combined$Method <- 'Original'
train_combined$Type <- 'Train'
train_ii_combined <- train_ii[, c("LC50", "predicted_LC50")]
train_ii_combined$Method <- 'Dummy'
train_ii_combined$Type <- 'Train'
train_combined_all <- rbind(train_combined, train_ii_combined)
test_combined <- test_i[, c("LC50", "predicted_LC50")]
test_combined$Method <- 'Original'
test_combined$Type <- 'Test'
test_ii_combined <- test_ii[, c("LC50", "predicted_LC50")]
test_ii_combined$Method <- 'Dummy'
test_ii_combined$Type <- 'Test'
test_combined_all <- rbind(test_combined, test_ii_combined)
```

```r
# Convert 'Method' and 'Type' to factors
train_combined_all$Method <- factor(train_combined_all$Method, levels = c('Original', 'Dummy'))
test_combined_all$Method <- factor(test_combined_all$Method, levels = c('Original', 'Dummy'))

# Function to draw regression lines
add_regression_lines <- function(df, original_model, dummy_model) {
ggplot(df, aes(x = LC50, y = predicted_LC50, color = Method)) +
geom_point(alpha = 0.7) +
geom_smooth(method = "lm", formula = y ~ x, se = FALSE,
aes(linetype = Method),
data = df[df$Method == 'Original', ],
color = 'blue') +
geom_smooth(method = "lm", formula = y ~ x, se = FALSE,
aes(linetype = Method),
data = df[df$Method == 'Dummy', ],
color = 'red') +
geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
labs(x = "Observed LC50", y = "Predicted LC50", title = df$Type[1]) +
theme_minimal() +
theme(legend.position = "bottom")
}
# Plot training data with both regression lines
train_plot <- add_regression_lines(train_combined_all, model, model_transform_dummy)
train_plot <- train_plot + labs(title = "Training Data")
# Plot testing data with both regression lines
test_plot <- add_regression_lines(test_combined_all, model, model_transform_dummy)
test_plot <- test_plot + labs(title = "Testing Data")
# Display plots side by side
grid.arrange(train_plot, test_plot, ncol = 2)
```



In initial conclusion in one time spliting, the original model (without dummy encoding) provides a better fit to both the training and testing data, as evidenced by its closer alignment with the ideal prediction line and lower dispersion in the test data.

This is likely because it retains the continuous information in the count variables, which adds more nuance to the model's predictions. So, in part b, we will draw a more reliable conclusion by repeating the procedure 200 times and comparing the average test errors.

b. Repeating the procedure 200 times

Procedure - Randomly spiting training vs test set (2/3 vs 1/3). - Fit the models with 2 options (i) Original model and (ii) Dummy encoding. - Record the test errors (MSE/RMSE/R^2).

```
# Initialize vectors to store test errors
mse_test_errors_i <- numeric(200)
rmse_test_errors_i <- numeric(200)
r2_test_errors_i <- numeric(200)
mse_test_errors_ii <- numeric(200)
rmse_test_errors_ii <- numeric(200)
r2_test_errors_ii <- numeric(200)

# Repeat the procedure 200 times
set.seed(2)
for (i in 1:200) {
  # Split the data
  sample <- sample.split(data$LC50, SplitRatio = 2/3)
```

```
  train <- subset(data, sample == TRUE)
  test <- subset(data, sample == FALSE)

  # Option (i): Original model
  model <- lm(LC50 ~ ., data=train)
  pred_test_i <- predict(model, newdata=test)
  mse_test_i <- mean((test$LC50 - pred_test_i)^2)
  rmse_test_i <- sqrt(mse_test_i)
  r2_test_i <- 1 - (sum((test$LC50 - pred_test_i)^2) / sum((test$LC50 - mean(test$LC50))^2))

  # Option (ii): Dummy encoding
  train$H050 <- ifelse(train$H050 > 0, 1, 0)
  train$nN <- ifelse(train$nN > 0, 1, 0)
  train$C040 <- ifelse(train$C040 > 0, 1, 0)

  test$H050 <- ifelse(test$H050 > 0, 1, 0)
  test$nN <- ifelse(test$nN > 0, 1, 0)
  test$C040 <- ifelse(test$C040 > 0, 1, 0)

  model_ii <- lm(LC50 ~ ., data = train)
  pred_test_ii <- predict(model_ii, newdata = test)
  mse_test_ii <- mean((test$LC50 - pred_test_ii)^2)
  rmse_test_ii <- sqrt(mse_test_ii)
  r2_test_ii <- 1 - (sum((test$LC50 - pred_test_ii)^2) / sum((test$LC50 - mean(test$LC50))^2))

  # Record the test errors
  mse_test_errors_i[i] <- mse_test_i
  rmse_test_errors_i[i] <- rmse_test_i
  r2_test_errors_i[i] <- r2_test_i

  mse_test_errors_ii[i] <- mse_test_ii
  rmse_test_errors_ii[i] <- rmse_test_ii
  r2_test_errors_ii[i] <- r2_test_ii
}
```

There are a few key reasons for repeating the procedure 200 times:

- To reduce the influence of random data splits
- To provide a more reliable estimate of the model's performance

```
# Calculate and print average test errors
average_test_error_i <- mean(mse_test_errors_i)
average_rmse_error_i <- mean(rmse_test_errors_i)
average_r2_error_i <- mean(r2_test_errors_i)

average_test_error_ii <- mean(mse_test_errors_ii)
average_rmse_error_ii <- mean(rmse_test_errors_ii)
average_r2_error_ii <- mean(r2_test_errors_ii)

cat(paste0(
  "Average Test Errors (Original Model):\n",
  "MSE: ", average_test_error_i, "\n",
  "RMSE: ", average_rmse_error_i, "\n",
```

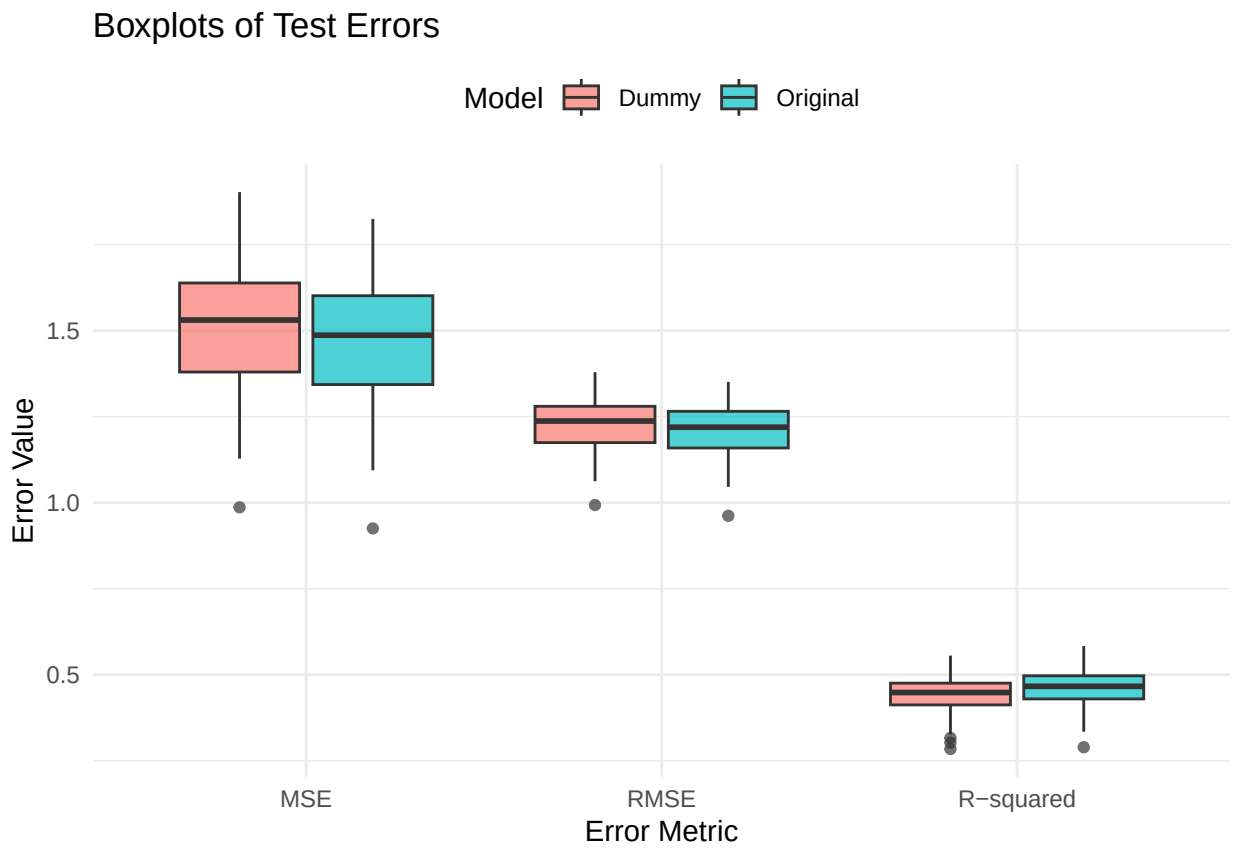
```
  "R-squared: ", average_r2_error_i, "\n\n",

  "Average Test Errors (Dummy Model):\n",
  "MSE: ", average_test_error_ii, "\n",
  "RMSE: ", average_rmse_error_ii, "\n",
  "R-squared: ", average_r2_error_ii, "\n"
))
```

```
## Average Test Errors (Original Model):
## MSE: 1.47708146772242
## RMSE: 1.21330895603276
## R-squared: 0.460485255063669
##
## Average Test Errors (Dummy Model):
## MSE: 1.52752950559007
## RMSE: 1.23398478875802
## R-squared: 0.442128799570138
```

The original model consistently achieves lower MSE and RMSE than the dummy-encoded model, as indicated by the density distributions. The peak of the distribution for the original model is shifted left compared to the dummy model, meaning the original model typically has smaller test errors. The dummy-encoded model shows slightly larger and more spread-out test errors, indicating poorer performance.

```
# Create data frames for plotting
errors_df_mse <- data.frame(
  Error = c(mse_test_errors_i, mse_test_errors_ii),
  Metric = 'MSE',
  Model = factor(rep(c("Original", "Dummy"), each = 200))
)
errors_df_rmse <- data.frame(
  Error = c(rmse_test_errors_i, rmse_test_errors_ii),
  Metric = 'RMSE',
  Model = factor(rep(c("Original", "Dummy"), each = 200))
)
errors_df_r2 <- data.frame(
  Error = c(r2_test_errors_i, r2_test_errors_ii),
  Metric = 'R-squared',
  Model = factor(rep(c("Original", "Dummy"), each = 200))
)
errors_df <- rbind(errors_df_mse, errors_df_rmse, errors_df_r2)

# Ensure the 'Metric' factor has the correct level order
errors_df$Metric <- factor(errors_df$Metric, levels = c('MSE', 'RMSE', 'R-squared'))

# Plot the empirical distributions of the test errors
ggplot(errors_df, aes(x = Error, fill = Model)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~ Metric, scales = "free") +
  labs(title = "Empirical Distributions of Test Errors", x = "Test Error", y = "Density") +
  theme_minimal()
```


```
# Plot the empirical distributions of the test errors using boxplots
ggplot(errors_df, aes(x = Metric, y = Error, fill = Model)) +
  geom_boxplot(alpha = 0.7) +
  labs(title = "Boxplots of Test Errors", x = "Error Metric", y = "Error Value") +
  theme_minimal() +
  theme(legend.position = "top")
```



In conclusion, the higher test errors and greater variability of the dummy-encoded model (option ii) occur because it sacrifices important information present in the original continuous variables. Repeating the process confirms that the original model (option i) is generally superior.

c. Variable selection procedures

```r
# Split the data into training (2/3) and test (1/3) sets
set.seed(123)
sample <- sample.split(data$LC50, SplitRatio = 2/3)
train <- subset(data, sample == TRUE)
test <- subset(data, sample == FALSE)

# Set up full and null model
full.model <- lm(LC50 ~ ., data = train)
null.model <- lm(LC50 ~ 1, data = train)

# Set up target and number of variables
y <- train$LC50
num_vars <- ncol(train) - 1  # exclude the response variable column
```

Forward Selection

Forward Selection is a stepwise regression method that starts with an empty model and adds predictors one by one based on a criterion (e.g., AIC, BIC) until no more predictors can be added.

- Key variables that were consistently selected include MLOGP, TPSA, SAacc, nN, RDCHI, and GATS1p.
- Both AIC and BIC agreed on the significance of these variables. However, BIC, being stricter, might lead to simpler models in other datasets, but in this case, the results remained the same across both criteria.

```
# With AIC
model.forward.aic <- stepAIC(null.model, scope = list(lower = null.model, upper = full.model), directio
summary(model.forward.aic)
```

```
##
## Call:
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + RDCHI + GATS1p,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8194 -0.8018 -0.1737  0.6654  4.8981
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.653540   0.285743   9.286  < 2e-16 ***
## MLOGP        0.404067   0.078544   5.144 4.44e-07 ***
## TPSA         0.027138   0.003284   8.265 2.78e-15 ***
## SAacc       -0.016185   0.002177  -7.435 7.84e-13 ***
## nN          -0.201305   0.058114  -3.464 0.000597 ***
## RDCHI        0.639082   0.174662   3.659 0.000291 ***
## GATS1p      -0.589921   0.183821  -3.209 0.001452 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.216 on 357 degrees of freedom
## Multiple R-squared:  0.502,  Adjusted R-squared:  0.4937
## F-statistic: 59.98 on 6 and 357 DF,  p-value: < 2.2e-16
```

```
# With BIC
# If we set it to k = log(n), the function considers the BIC.

model.forward.bic <- stepAIC(null.model, scope = list(lower = null.model, upper = full.model), directio
summary(model.forward.bic)
```

```
##
## Call:
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + RDCHI + GATS1p,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8194 -0.8018 -0.1737  0.6654  4.8981
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.653540   0.285743   9.286  < 2e-16 ***
```

```
## MLOGP        0.404067   0.078544   5.144 4.44e-07 ***
## TPSA         0.027138   0.003284   8.265 2.78e-15 ***
## SAacc       -0.016185   0.002177  -7.435 7.84e-13 ***
## nN          -0.201305   0.058114  -3.464 0.000597 ***
## RDCHI        0.639082   0.174662   3.659 0.000291 ***
## GATS1p      -0.589921   0.183821  -3.209 0.001452 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.216 on 357 degrees of freedom
## Multiple R-squared:  0.502,  Adjusted R-squared:  0.4937
## F-statistic: 59.98 on 6 and 357 DF,  p-value: < 2.2e-16
```

Backward Elimination

Backward Elimination is a stepwise regression method that starts with the full model and removes predictors one by one based on a criterion (e.g., AIC, BIC) until no more predictors can be removed.

- Similar to forward selection, backward elimination with both AIC and BIC resulted in a model that includes MLOGP, TPSA, SAacc, nN, RDCHI, and GATS1p.
- The consistency between forward and backward selection indicates that these predictors are strong, regardless of the method or criterion (AIC vs. BIC) used.

```r
# With AIC
model.backward.aic <- stepAIC(full.model, direction = 'backward', trace = FALSE)
summary(model.backward.aic)
```

```
##
## Call:
## lm(formula = LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8194 -0.8018 -0.1737  0.6654  4.8981
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.653540   0.285743   9.286  < 2e-16 ***
## TPSA         0.027138   0.003284   8.265 2.78e-15 ***
## SAacc       -0.016185   0.002177  -7.435 7.84e-13 ***
## MLOGP        0.404067   0.078544   5.144 4.44e-07 ***
## RDCHI        0.639082   0.174662   3.659 0.000291 ***
## GATS1p      -0.589921   0.183821  -3.209 0.001452 **
## nN          -0.201305   0.058114  -3.464 0.000597 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.216 on 357 degrees of freedom
## Multiple R-squared:  0.502,  Adjusted R-squared:  0.4937
## F-statistic: 59.98 on 6 and 357 DF,  p-value: < 2.2e-16
```

```r
# With BIC
model.backward.bic <- stepAIC(full.model, direction = 'backward', k = log(nrow(train)), trace = FALSE)
summary(model.backward.bic)
```

```
##
## Call:
## lm(formula = LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8194 -0.8018 -0.1737  0.6654  4.8981
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.653540   0.285743   9.286  < 2e-16 ***
## TPSA         0.027138   0.003284   8.265 2.78e-15 ***
## SAacc       -0.016185   0.002177  -7.435 7.84e-13 ***
## MLOGP        0.404067   0.078544   5.144 4.44e-07 ***
## RDCHI        0.639082   0.174662   3.659 0.000291 ***
## GATS1p      -0.589921   0.183821  -3.209 0.001452 **
## nN          -0.201305   0.058114  -3.464 0.000597 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.216 on 357 degrees of freedom
## Multiple R-squared:  0.502,  Adjusted R-squared:  0.4937
## F-statistic: 59.98 on 6 and 357 DF,  p-value: < 2.2e-16
```

Stepwise Selection

Stepwise Selection is a combination of forward and backward selection, where predictors are added or removed based on a criterion (e.g., AIC, BIC) until no more changes can be made.

- The stepwise selection, which combines both forward and backward methods, also identified the same set of variables: MLOGP, TPSA, SAacc, nN, RDCHI, and GATS1p.
- There is no significant difference between the AIC and BIC results in this specific case.

```r
# With AIC
model.stepwise.aic <- stepAIC(null.model, scope = list(lower = null.model, upper = full.model), directi
summary(model.stepwise.aic)
```

```
##
## Call:
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + RDCHI + GATS1p,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8194 -0.8018 -0.1737  0.6654  4.8981
##
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.653540   0.285743   9.286  < 2e-16 ***
## MLOGP        0.404067   0.078544   5.144 4.44e-07 ***
## TPSA         0.027138   0.003284   8.265 2.78e-15 ***
## SAacc       -0.016185   0.002177  -7.435 7.84e-13 ***
## nN          -0.201305   0.058114  -3.464 0.000597 ***
## RDCHI        0.639082   0.174662   3.659 0.000291 ***
## GATS1p      -0.589921   0.183821  -3.209 0.001452 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.216 on 357 degrees of freedom
## Multiple R-squared:  0.502,  Adjusted R-squared:  0.4937
## F-statistic: 59.98 on 6 and 357 DF,  p-value: < 2.2e-16
```

```
# With BIC
model.stepwise.bic <- stepAIC(null.model, scope = list(lower = null.model, upper = full.model), directi
summary(model.stepwise.bic)
```

```
##
## Call:
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + RDCHI + GATS1p,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8194 -0.8018 -0.1737  0.6654  4.8981
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.653540   0.285743   9.286  < 2e-16 ***
## MLOGP        0.404067   0.078544   5.144 4.44e-07 ***
## TPSA         0.027138   0.003284   8.265 2.78e-15 ***
## SAacc       -0.016185   0.002177  -7.435 7.84e-13 ***
## nN          -0.201305   0.058114  -3.464 0.000597 ***
## RDCHI        0.639082   0.174662   3.659 0.000291 ***
## GATS1p      -0.589921   0.183821  -3.209 0.001452 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.216 on 357 degrees of freedom
## Multiple R-squared:  0.502,  Adjusted R-squared:  0.4937
## F-statistic: 59.98 on 6 and 357 DF,  p-value: < 2.2e-16
```

Model Comparison

```
# Predict on the test set using all models
test$pred_backward_aic <- predict(model.backward.aic, newdata = test)
test$pred_forward_aic <- predict(model.forward.aic, newdata = test)
test$pred_stepwise_aic <- predict(model.stepwise.aic, newdata = test)
test$pred_backward_bic <- predict(model.backward.bic, newdata = test)
test$pred_forward_bic <- predict(model.forward.bic, newdata = test)
```

```r
test$pred_stepwise_bic <- predict(model.stepwise.bic, newdata = test)

# Calculate MSE, RMSE, and R-squared for each model
mse <- function(actual, predicted) mean((actual - predicted)^2)
rmse <- function(actual, predicted) sqrt(mse(actual, predicted))
r2 <- function(actual, predicted) 1 - (sum((actual - predicted)^2) / sum((actual - mean(actual))^2))

metrics <- data.frame(
  Model = c("Backward AIC", "Forward AIC", "Stepwise AIC", "Backward BIC", "Forward BIC", "Stepwise BIC
  MSE = c(
    mse(test$LC50, test$pred_backward_aic),
    mse(test$LC50, test$pred_forward_aic),
    mse(test$LC50, test$pred_stepwise_aic),
    mse(test$LC50, test$pred_backward_bic),
    mse(test$LC50, test$pred_forward_bic),
    mse(test$LC50, test$pred_stepwise_bic)
  ),
  RMSE = c(
    rmse(test$LC50, test$pred_backward_aic),
    rmse(test$LC50, test$pred_forward_aic),
    rmse(test$LC50, test$pred_stepwise_aic),
    rmse(test$LC50, test$pred_backward_bic),
    rmse(test$LC50, test$pred_forward_bic),
    rmse(test$LC50, test$pred_stepwise_bic)
  ),
  R2 = c(
    r2(test$LC50, test$pred_backward_aic),
    r2(test$LC50, test$pred_forward_aic),
    r2(test$LC50, test$pred_stepwise_aic),
    r2(test$LC50, test$pred_backward_bic),
    r2(test$LC50, test$pred_forward_bic),
    r2(test$LC50, test$pred_stepwise_bic)
  )
)
print(metrics)
```

```
##          Model      MSE     RMSE        R2
## 1 Backward AIC 1.398176 1.182445 0.4352328
## 2  Forward AIC 1.398176 1.182445 0.4352328
## 3 Stepwise AIC 1.398176 1.182445 0.4352328
## 4 Backward BIC 1.398176 1.182445 0.4352328
## 5  Forward BIC 1.398176 1.182445 0.4352328
## 6 Stepwise BIC 1.398176 1.182445 0.4352328
```

In conclusion, the variable selection procedures (forward, backward, and stepwise) consistently identified the same set of predictors: MLOGP, TPSA, SAacc, nN, RDCHI, and GATS1p.

d. Ridge Regression

```r
# Split the data into training (2/3) and test (1/3) sets
set.seed(123)
```

```r
sample <- sample.split(data$LC50, SplitRatio = 2/3)
train <- subset(data, sample == TRUE)
test <- subset(data, sample == FALSE)

# Set up the training and test data
x_train <- as.matrix(train[, -9])
y_train <- train$LC50
x_test <- as.matrix(test[, -9])
y_test <- test$LC50
```

Cross Validation

```r
# Reference: https://bookdown.org/ssjackson300/Machine-Learning-Lecture-Notes/choosing-lambda.html

# Define a grid of lambda values
lambda_grid <- 10^seq(10, -2, length = 100)

# Perform cross-validation for ridge regression
cv_ridge <- cv.glmnet(x_train,
                      y_train,
                      alpha = 0,
                      lambda = lambda_grid,
                      standardize = TRUE
                      )
best_lambda_cv <- cv_ridge$lambda.min
print(paste("Best Lambda from Cross-Validation:", best_lambda_cv))
```

```
## [1] "Best Lambda from Cross-Validation: 0.0174752840000768"
```

```r
# Predict and evaluate on test data
ridge_pred_cv <- predict(cv_ridge, s = best_lambda_cv, newx = x_test)
mse_cv <- mean((ridge_pred_cv - y_test)^2)
rmse_cv <- sqrt(mse_cv)
r2_cv <- 1 - (sum((ridge_pred_cv - y_test)^2) / sum((y_test - mean(y_test))^2))

cat(paste0(
  "MSE: ", mse_cv, "\n",
  "RMSE (Test): ", rmse_cv, "\n",
  "R-squared (Test): ", r2_cv, "\n"
))
```

```
## MSE: 1.39984196703498
## RMSE (Test): 1.18314917361885
## R-squared (Test): 0.434559904102569
```

Bootstrap Procedure

```r
# Reference: https://pages.stat.wisc.edu/~kdlevin/teaching/Fall2022/STAT340/lecs/L13_bootstrap.html

# Define ridge regression function for bootstrap
ridge_bootstrap <- function(data, lambda, B = 100) {
  n <- nrow(data)  # number of observations
  boot_mses <- numeric(B)

  for (i in 1:B) {
    resample_indices <- sample(1:n, n, replace = TRUE)

    # resampled_data <- fin_pairs[resample_indices,] fin_pairs = [X, Y]
    resampled_data <- data[resample_indices, ]

    x_boot <- as.matrix(resampled_data[, -9])
    y_boot <- resampled_data$LC50

    # Apply ridge regression and predict in this resampling data set
    ridge_model <- glmnet(x_boot, y_boot, alpha = 0, lambda = lambda, standardize = TRUE)
    boot_pred <- predict(ridge_model, s = lambda, newx = as.matrix(data[, -9]))
    boot_mses[i] <- mean((boot_pred - data$LC50)^2)
  }

  return(mean(boot_mses))
}

# Perform bootstrap for ridge regression
set.seed(1)
boot_results <- sapply(lambda_grid, function(lambda) {
  ridge_bootstrap(train, lambda, B = 100)
})

# Find the optimal lambda
best_lambda_bootstrap <- lambda_grid[which.min(boot_results)]
print(paste("Best Lambda from Bootstrap:", best_lambda_bootstrap))
```

```
## [1] "Best Lambda from Bootstrap: 0.0132194114846603"
```

```r
# Predict and evaluate on test data
ridge_pred_bootstrap <- predict(cv_ridge, s = best_lambda_bootstrap, newx = x_test)
mse_bootstrap <- mean((ridge_pred_bootstrap - y_test)^2)
rmse_bootstrap <- sqrt(mse_bootstrap)
r2_bootstrap <- 1 - (sum((ridge_pred_bootstrap - y_test)^2) / sum((y_test - mean(y_test))^2))

cat(paste0(
  "MSE: ", mse_bootstrap, "\n",
  "RMSE (Test): ", rmse_bootstrap, "\n",
  "R-squared (Test): ", r2_bootstrap, "\n"
))
```

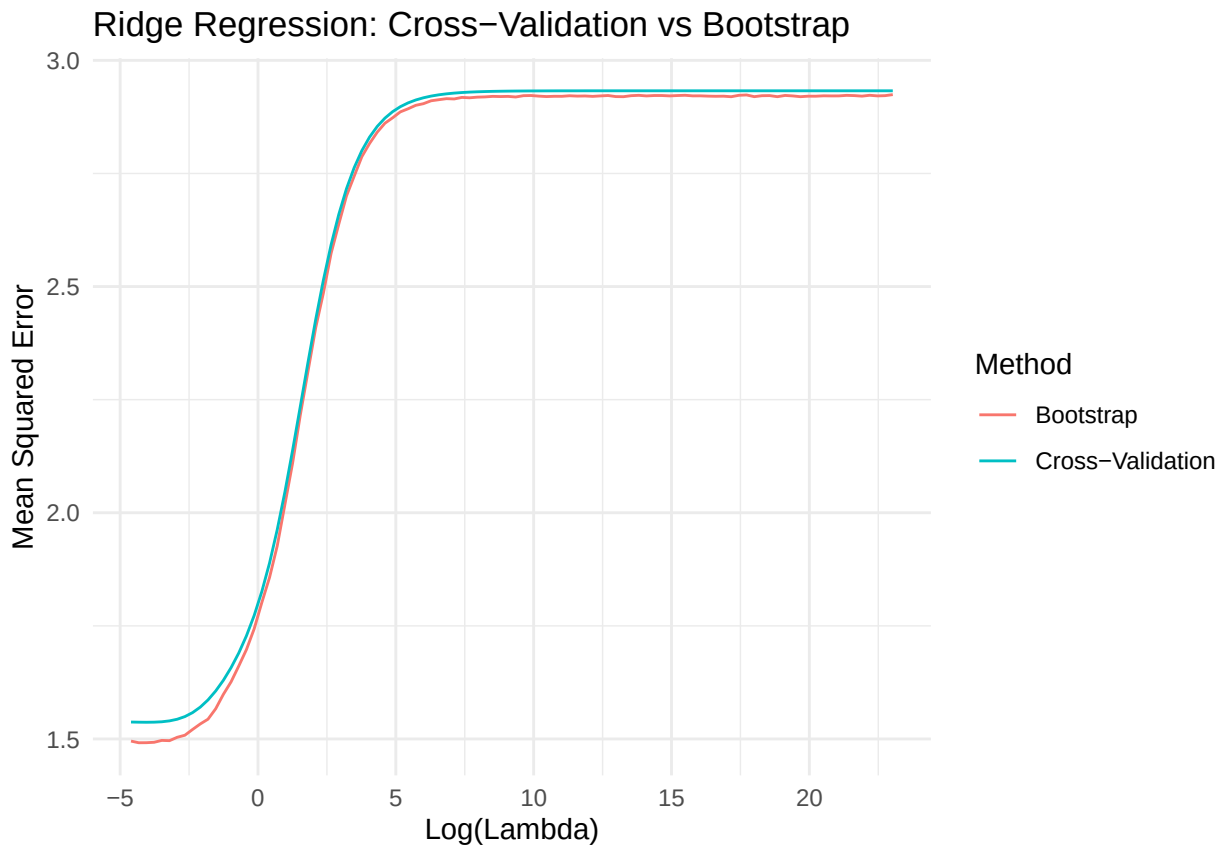
```
## MSE: 1.39995494236231
## RMSE (Test): 1.18319691613962
## R-squared (Test): 0.434514269822825
```

Cross Validation vs Bootstrap Comparision

Using both cross-validation and bootstrap procedures, the optimal complexity parameter (λ) was found to be approximately 0.017 using cross-validation and 0.013 using bootstrap. The performance of the ridge regression was very similar under both methods, with test set MSE around 1.40 and R-squared about 0.43, slightly better than the dummy encoded linear regression but on par with the original linear model.

In the plot below, we observed that the both cross-validation and bootstrap should provide similar λ values, but cross-validation typically has less variance in error estimates compared to bootstrap.

```
# Create comparison data frame
comparison_df <- data.frame(
  Lambda = rep(lambda_grid, 2),
  MSE = c(cv_ridge$cvm, boot_results),
  Method = rep(c("Cross-Validation", "Bootstrap"), each = length(lambda_grid))
)

# Plot the results
ggplot(comparison_df, aes(x = log(Lambda), y = MSE, color = Method)) +
  geom_line() +
  labs(title = "Ridge Regression: Cross-Validation vs Bootstrap",
       x = "Log(Lambda)",
       y = "Mean Squared Error") +
  theme_minimal()
```



e. Generalised Additive Model (GAM)

```
summary(train)
```

```
##       TPSA            SAacc             H050             MLOGP
##  Min.   :  0.00   Min.   :  0.00   Min.   : 0.0000   Min.   :-5.199
##  1st Qu.: 16.05   1st Qu.: 13.13   1st Qu.: 0.0000   1st Qu.: 1.139
##  Median : 40.46   Median : 42.92   Median : 0.0000   Median : 2.226
##  Mean   : 48.02   Mean   : 58.75   Mean   : 0.9313   Mean   : 2.273
##  3rd Qu.: 70.14   3rd Qu.: 78.20   3rd Qu.: 1.0000   3rd Qu.: 3.455
##  Max.   :336.43   Max.   :551.10   Max.   :16.0000   Max.   : 9.148
##      RDCHI           GATS1p             nN              C040
##  Min.   :1.000   Min.   :0.2880   Min.   : 0.000   Min.   : 0.0000
##  1st Qu.:1.946   1st Qu.:0.7578   1st Qu.: 0.000   1st Qu.: 0.0000
##  Median :2.329   Median :1.0485   Median : 1.000   Median : 0.0000
##  Mean   :2.469   Mean   :1.0682   Mean   : 1.025   Mean   : 0.3654
##  3rd Qu.:2.913   3rd Qu.:1.2902   3rd Qu.: 2.000   3rd Qu.: 0.0000
##  Max.   :6.439   Max.   :2.3530   Max.   :11.000   Max.   :11.0000
##       LC50
##  Min.   : 0.122
##  1st Qu.: 3.603
##  Median : 4.516
##  Mean   : 4.666
##  3rd Qu.: 5.637
##  Max.   :10.047
```

Lower Complexity (k = 4)

```
# Fit GAM with smoothing splines (lower complexity)
gam_model_1 <- gam(LC50 ~ s(TPSA, k = 4) + s(SAacc, k = 4) + s(H050, k = 4) +
                     s(MLOGP, k = 4) + s(RDCHI, k = 4) + s(GATS1p, k = 4) +
                     s(nN, k = 4) + s(C040, k = 4), data = train)

# Summarize models
summary(gam_model_1)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## LC50 ~ s(TPSA, k = 4) + s(SAacc, k = 4) + s(H050, k = 4) + s(MLOGP,
##     k = 4) + s(RDCHI, k = 4) + s(GATS1p, k = 4) + s(nN, k = 4) +
##     s(C040, k = 4)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.66605    0.06325   73.77   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Approximate significance of smooth terms:
##             edf Ref.df      F  p-value
## s(TPSA)   2.070  2.398 29.935  < 2e-16 ***
## s(SAacc)  2.653  2.839 13.121 1.57e-07 ***
## s(H050)   1.000  1.000  0.243 0.622164
## s(MLOGP)  1.000  1.000 26.936 6.42e-07 ***
## s(RDCHI)  1.000  1.000 11.284 0.000867 ***
## s(GATS1p) 1.000  1.000  8.847 0.003138 **
## s(nN)     1.000  1.000  8.832 0.003164 **
## s(C040)   1.000  1.000  0.216 0.642396
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.502   Deviance explained = 51.6%
## GCV = 1.5049  Scale est. = 1.4564    n = 364
```

```
gam_pred_1 <- predict(gam_model_1, newdata = test)
gam_mse_1 <- mean((gam_pred_1 - y_test)^2)
gam_rmse_1 <- sqrt(gam_mse_1)
gam_r2_1 <- 1 - (sum((gam_pred_1 - y_test)^2) / sum((y_test - mean(y_test))^2))

cat(paste0(
  "MSE (Test): ", gam_mse_1, "\n",
  "RMSE (Test): ", gam_rmse_1, "\n",
  "R-squared (Test): ", gam_r2_1, "\n"
))
```

```
## MSE (Test): 1.40582163542459
## RMSE (Test): 1.18567349444296
## R-squared (Test): 0.432144528404967
```

Higher Complexity (k = 6)

```
# Fit GAM with smoothing splines (higher complexity)
gam_model_2 <- gam(LC50 ~ s(TPSA, k = 6) + s(SAacc, k = 6) + s(H050, k = 6) +
                     s(MLOGP, k = 6) + s(RDCHI, k = 6) + s(GATS1p, k = 6) +
                     s(nN, k = 6) + s(C040, k = 6), data = train)

# Summarize models
summary(gam_model_2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## LC50 ~ s(TPSA, k = 6) + s(SAacc, k = 6) + s(H050, k = 6) + s(MLOGP,
##     k = 6) + s(RDCHI, k = 6) + s(GATS1p, k = 6) + s(nN, k = 6) +
##     s(C040, k = 6)
##
```

```
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.66605    0.06113   76.33   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##             edf Ref.df      F  p-value
## s(TPSA)   4.026  4.425 18.481  < 2e-16 ***
## s(SAacc)  3.775  4.260  9.769 5.29e-07 ***
## s(H050)   1.000  1.000  1.095  0.29606
## s(MLOGP)  4.643  4.865  7.208 2.61e-06 ***
## s(RDCHI)  1.000  1.000  8.520  0.00375 **
## s(GATS1p) 1.871  2.363  3.643  0.01718 *
## s(nN)     3.721  4.295  3.602  0.00821 **
## s(C040)   1.000  1.000  2.129  0.14545
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.534   Deviance explained = 56.1%
## GCV = 1.4478  Scale est. = 1.3601    n = 364
```

```r
gam_pred_2 <- predict(gam_model_2, newdata = test)
gam_mse_2 <- mean((gam_pred_2 - y_test)^2)
gam_rmse_2 <- sqrt(gam_mse_2)
gam_r2_2 <- 1 - (sum((gam_pred_2 - y_test)^2) / sum((y_test - mean(y_test))^2))

cat(paste0(
  "MSE: (Test):", gam_mse_2, "\n",
  "RMSE (Test): ", gam_rmse_2, "\n",
  "R-squared (Test): ", gam_r2_2, "\n"
))
```

```
## MSE: (Test):1.3585368150292
## RMSE (Test): 1.16556287476446
## R-squared (Test): 0.451244351105306
```

In conclusion, fitting GAM models with different levels of smoothing complexity (k=4 and k=6) showed that increasing complexity slightly improved the fit, with R^2 improving from 0.43 to 0.45 and RMSE decreasing from 1.19 to 1.17. However, the benefit of increasing complexity is relatively minor, suggesting that moderate smoothing (k=4) suffices for this problem without introducing too much overfitting.

f. Regression Tree with Cost-Complexity Pruning

The complexity parameter (CP) table below shows the relative error, cross-validated error, and standard deviation for each tree size, helping to determine the best trade-off between bias and variance. The final pruned tree has about 26 splits, meaning that it captures sufficient detail to provide accurate predictions without being overly complex.

```r
# Fit a regression tree model
tree_model <- rpart(LC50 ~ .,
                    data = train,
```

```
                    method = "anova",
                    control = rpart.control(cp = 0.001))
printcp(tree_model)  # Display the cost complexity pruning table
```

```
##
## Regression tree:
## rpart(formula = LC50 ~ ., data = train, method = "anova", control = rpart.control(cp = 0.001))
##
## Variables actually used in tree construction:
## [1] C040   GATS1p H050   MLOGP  RDCHI  SAacc  TPSA
##
## Root node error: 1060.6/364 = 2.9138
##
## n= 364
##
##           CP nsplit rel error  xerror     xstd
## 1  0.2198608      0   1.00000 1.00582 0.083377
## 2  0.1015513      1   0.78014 0.85037 0.073595
## 3  0.0470255      2   0.67859 0.73790 0.061890
## 4  0.0384187      3   0.63156 0.67470 0.059262
## 5  0.0282925      6   0.51631 0.66571 0.059021
## 6  0.0225187      7   0.48801 0.66047 0.058312
## 7  0.0132815      8   0.46550 0.65154 0.056470
## 8  0.0109658     10   0.43893 0.60906 0.054135
## 9  0.0094706     11   0.42797 0.60140 0.053811
## 10 0.0086901     14   0.39955 0.59657 0.052692
## 11 0.0066615     15   0.39086 0.61671 0.053956
## 12 0.0063800     16   0.38420 0.63467 0.055231
## 13 0.0039482     17   0.37782 0.63634 0.058081
## 14 0.0038386     18   0.37387 0.62956 0.057522
## 15 0.0031083     19   0.37004 0.63425 0.057628
## 16 0.0027054     20   0.36693 0.63142 0.057532
## 17 0.0021546     21   0.36422 0.63076 0.058458
## 18 0.0019759     22   0.36207 0.62731 0.058474
## 19 0.0014224     23   0.36009 0.62838 0.058471
## 20 0.0014058     24   0.35867 0.63024 0.058550
## 21 0.0010000     26   0.35586 0.62934 0.058560
```

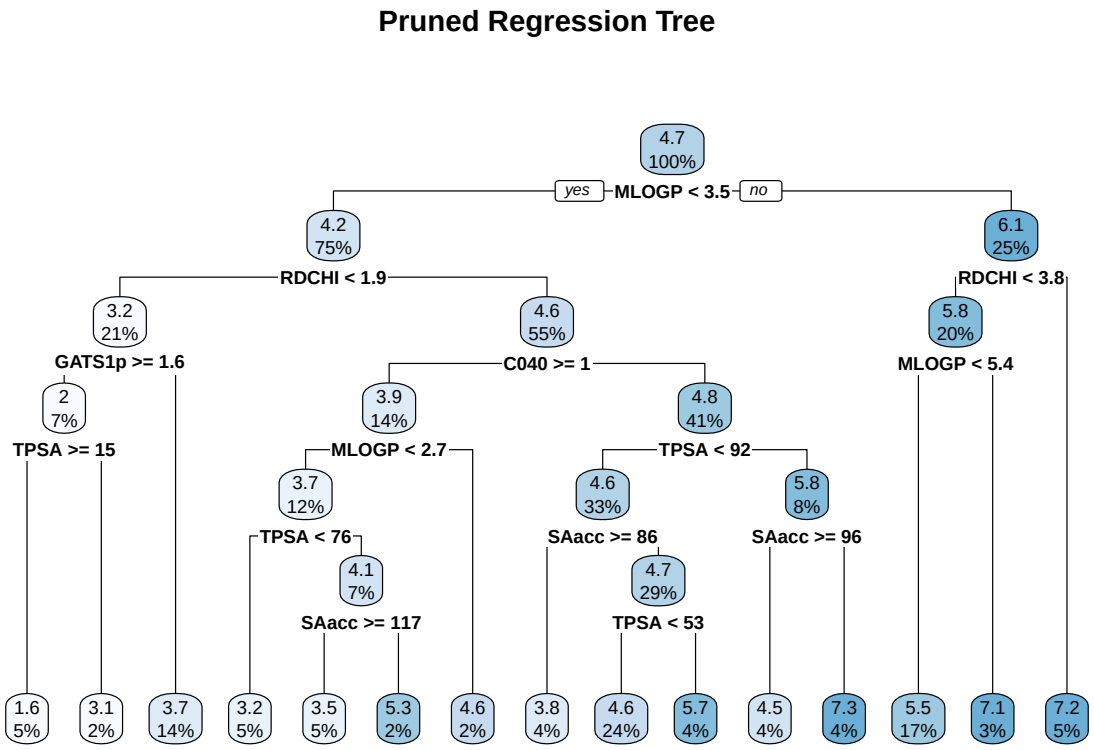
```
# Prune the tree
optimal_cp <- tree_model$cptable[which.min(tree_model$cptable[,"xerror"]), "CP"]
pruned_tree <- prune(tree_model, cp = optimal_cp)
```

The root node splits based on MLOGP (lipophilicity), which is the most important predictor for determining the LC50 value. The tree continues to split based on other descriptors, such as:

- RDCHI (topological index), which also be the first split in the tree
- C040 (the number of certain carbon atoms)
- TPSA (polar surface area)

The terminal nodes represent the predicted LC50 values for the corresponding groups of observations.

```r
# Visualize the tree
rpart.plot(pruned_tree, main = "Pruned Regression Tree")
```




```r
# Predict and evaluate on test data
tree_pred <- predict(pruned_tree, newdata = test)
tree_mse <- mean((tree_pred - y_test)^2)
tree_rmse <- sqrt(tree_mse)
tree_r2 <- 1 - (sum((tree_pred - y_test)^2) / sum((y_test - mean(y_test))^2))

cat(paste0(
  "MSE (Test): ", tree_mse, "\n",
  "RMSE (Test): ", tree_rmse, "\n",
  "R-squared (Test): ", tree_r2, "\n"
))
```

```
## MSE (Test): 1.55285972999018
## RMSE (Test): 1.24613792574906
## R-squared (Test): 0.372751228125619
```

However, the performance of the tree, with an R-squared of around 0.37 on the test set, suggests that the tree structure, while interpretable, may not capture the relationships in the data as effectively as other methods.

g. Compare all the models implemented

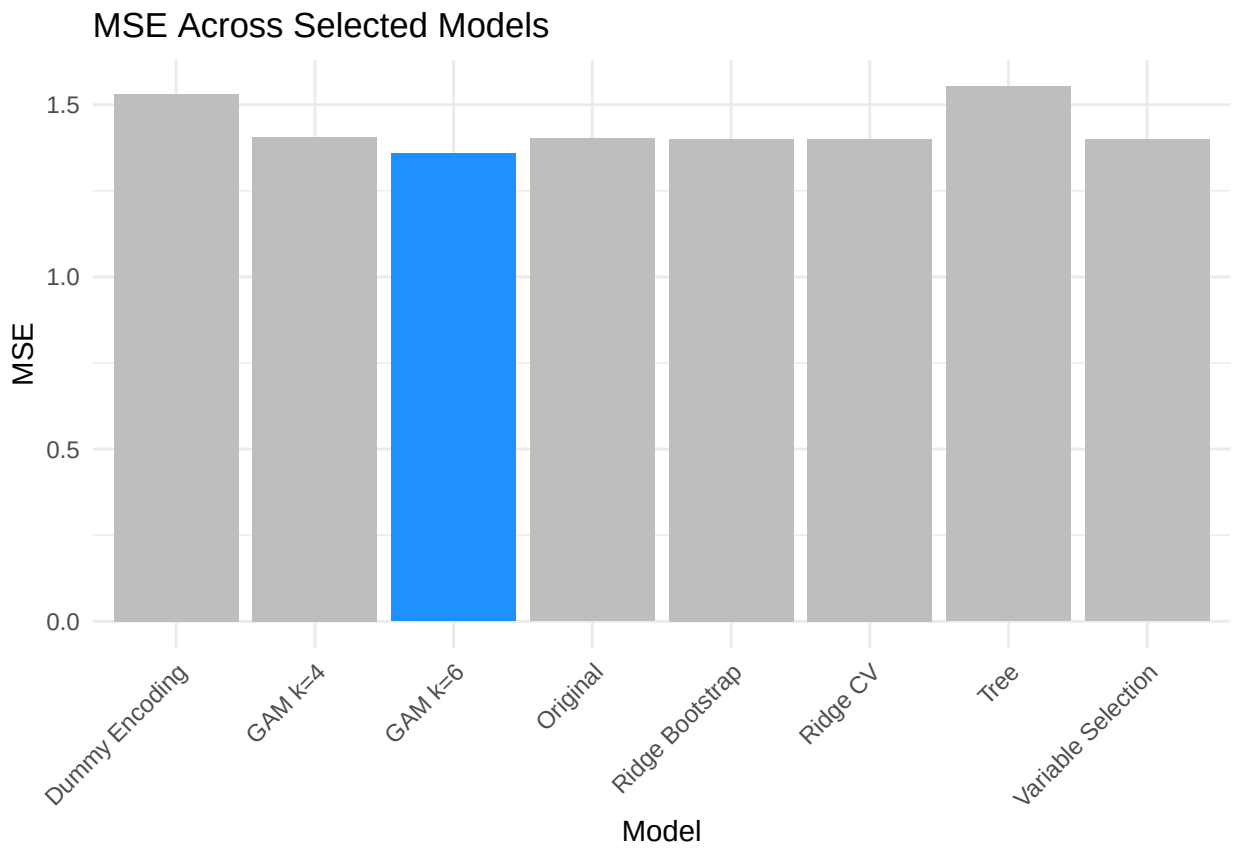
- Linear Regression (Original)

- Linear Regression model by Dummy Encoding Method (Dummy Encoding)
- Ridge Regression with Cross Validation (Ridge CV)
- Ridge Regression with Bootstrapping (Ridge Bootstrap)
- Generalized Additive Model with lower complexity (GAM k=4)
- Generalized Additive Model with higher complexity (GAM k=6)
- Regression Tree with Cost-Complexity Pruning (Tree)

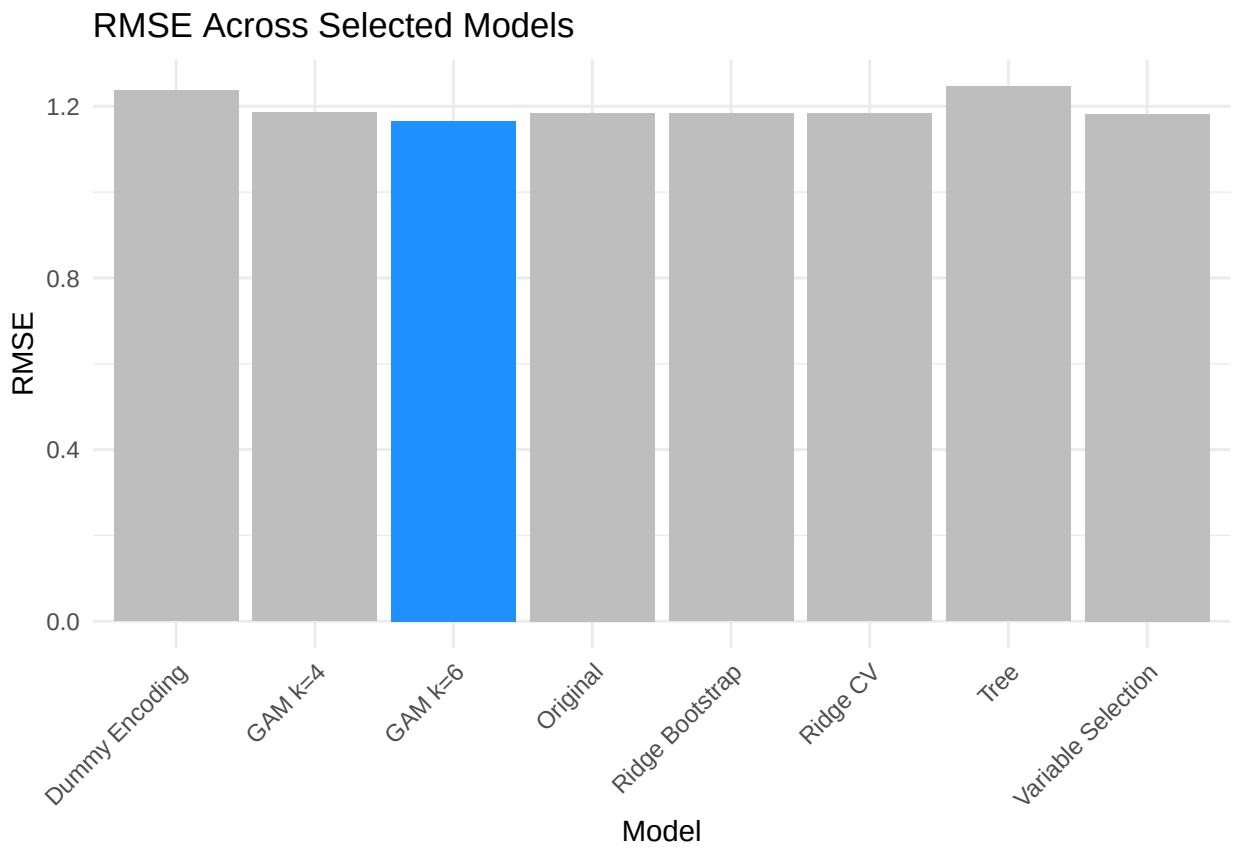
```
# Create a comprehensive metrics data frame
all_models_metrics <- data.frame(
  Model = c("Original", "Dummy Encoding", "Variable Selection", "Ridge CV", "Ridge Bootstrap", "GAM k=4
  MSE = c(mse_test, mse_test_transform_dummy, metrics$MSE[metrics$Model == "Forward AIC"], mse_cv, mse_
          gam_mse_2, tree_mse),
  RMSE = c(rmse_test, rmse_test_transform_dummy, metrics$RMSE[metrics$Model == "Forward AIC"], rmse_cv,
           gam_rmse_2, tree_rmse),
  R2 = c(r2_test, r2_test_transform_dummy, metrics$R2[metrics$Model == "Forward AIC"], r2_cv, r2_bootst
         gam_r2_2, tree_r2)
)

# Print the all models metrics
print(all_models_metrics)
```

```
##                Model      MSE     RMSE        R2
## 1           Original 1.402249 1.184166 0.4335877
## 2     Dummy Encoding 1.530438 1.237109 0.3818079
## 3 Variable Selection 1.398176 1.182445 0.4352328
## 4           Ridge CV 1.399842 1.183149 0.4345599
## 5    Ridge Bootstrap 1.399955 1.183197 0.4345143
## 6            GAM k=4 1.405822 1.185673 0.4321445
## 7            GAM k=6 1.358537 1.165563 0.4512444
## 8               Tree 1.552860 1.246138 0.3727512
```

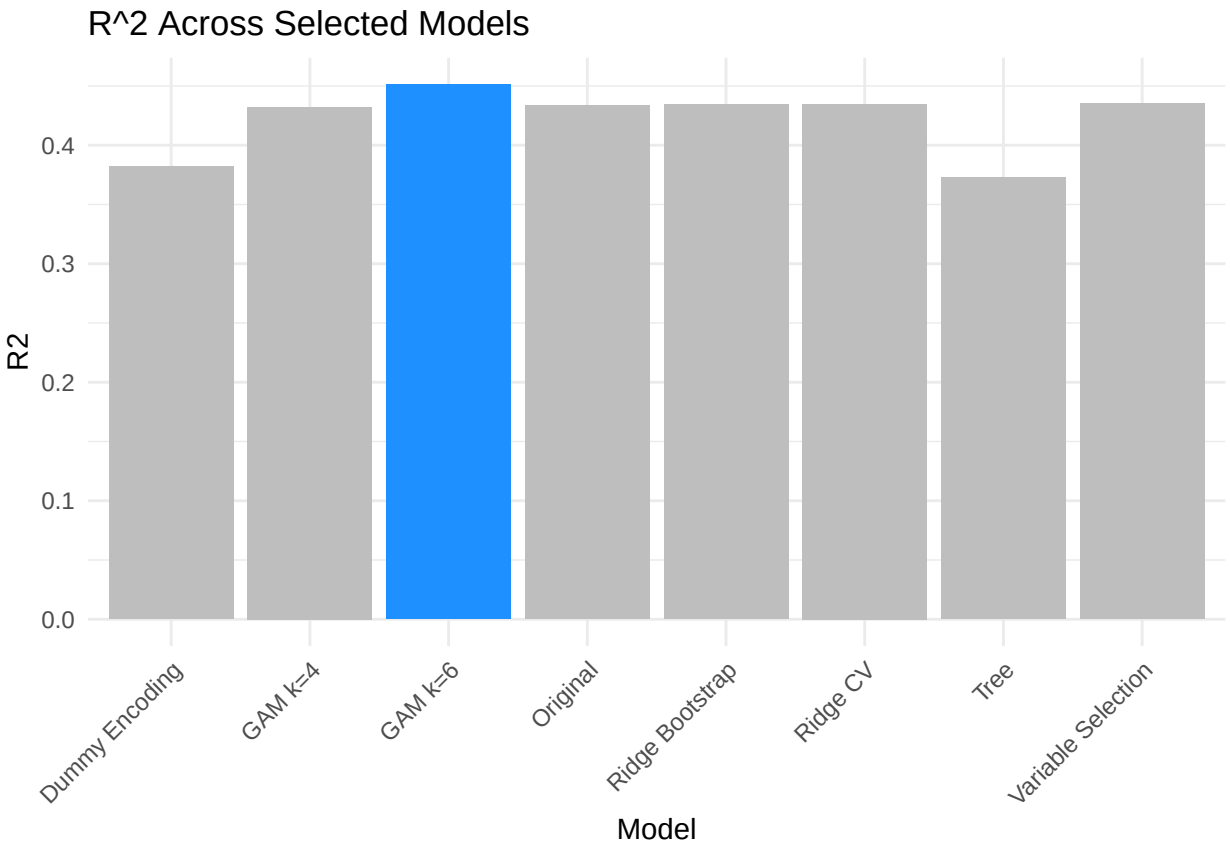
```
## Visualization of Model Comparisons

# Identify the model with the minimum MSE, RMSE and maximum R-squared
best_mse_model <- all_models_metrics$Model[which.min(all_models_metrics$MSE)]
best_rmse_model <- all_models_metrics$Model[which.min(all_models_metrics$RMSE)]
best_r2_model <- all_models_metrics$Model[which.max(all_models_metrics$R2)]

# Plot MSE across selected models
ggplot(all_models_metrics, aes(x = Model, y = MSE, fill = Model == best_mse_model)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("gray", "dodgerblue"), guide = "none") +
  theme_minimal() +
  labs(title = "MSE Across Selected Models", x = "Model", y = "MSE", fill = "Best Model") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```r
# Plot RMSE across selected models
ggplot(all_models_metrics, aes(x = Model, y = RMSE, fill = Model == best_mse_model)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("gray", "dodgerblue"), guide = "none") +
  theme_minimal() +
  labs(title = "RMSE Across Selected Models", x = "Model", y = "RMSE", fill = "Best Model") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```r
# Plot RMSE across selected models
ggplot(all_models_metrics, aes(x = Model, y = R2, fill = Model == best_mse_model)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("gray", "dodgerblue"), guide = "none") +
  theme_minimal() +
  labs(title = "R^2 Across Selected Models", x = "Model", y = "R2", fill = "Best Model") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



- The GAM (k=6) model provides the best performance, achieving the lowest error and highest R2. This indicates that adding flexibility through smoothing splines allows for capturing more intricate patterns in the data.
- It is closely followed by Ridge Regression (both cross-validation and bootstrap), which also performs well, showing that regularization improves generalization without overfitting.
- Variable Selection and the original linear model also perform solidly, while the dummy-encoded model under performs due to oversimplification, and the regression tree shows the weakest results.

Problem 2. Classification

```
library(mlbench)
data("PimaIndiansDiabetes2")
```

```
data <- PimaIndiansDiabetes2
head(data)
```

```
##   pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1        6     148       72      35      NA 33.6    0.627  50      pos
## 2        1      85       66      29      NA 26.6    0.351  31      neg
## 3        8     183       64      NA      NA 23.3    0.672  32      pos
## 4        1      89       66      23      94 28.1    0.167  21      neg
```

```
## 5        0     137       40      35     168 43.1    2.288  33      pos
## 6        5     116       74      NA      NA 25.6    0.201  30      neg
```

```
# Checking missing value
sapply(data, function(x) sum(is.na(x)))
```

```
## pregnant  glucose pressure  triceps  insulin     mass pedigree      age
##        0        5       35      227      374       11        0        0
## diabetes
##        0
```

```
# Remove rows with missing values
data <- na.omit(data)
head(data)
```

```
##    pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 4         1      89       66      23      94 28.1    0.167  21      neg
## 5         0     137       40      35     168 43.1    2.288  33      pos
## 7         3      78       50      32      88 31.0    0.248  26      pos
## 9         2     197       70      45     543 30.5    0.158  53      pos
## 14        1     189       60      23     846 30.1    0.398  59      pos
## 15        5     166       72      19     175 25.8    0.587  51      pos
```

```
summary(data)
```

```
##     pregnant         glucose         pressure         triceps
##  Min.   : 0.000   Min.   : 56.0   Min.   : 24.00   Min.   : 7.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.:21.00
##  Median : 2.000   Median :119.0   Median : 70.00   Median :29.00
##  Mean   : 3.301   Mean   :122.6   Mean   : 70.66   Mean   :29.15
##  3rd Qu.: 5.000   3rd Qu.:143.0   3rd Qu.: 78.00   3rd Qu.:37.00
##  Max.   :17.000   Max.   :198.0   Max.   :110.00   Max.   :63.00
##     insulin            mass          pedigree           age        diabetes
##  Min.   : 14.00   Min.   :18.20   Min.   :0.0850   Min.   :21.00   neg:262
##  1st Qu.: 76.75   1st Qu.:28.40   1st Qu.:0.2697   1st Qu.:23.00   pos:130
##  Median :125.50   Median :33.20   Median :0.4495   Median :27.00
##  Mean   :156.06   Mean   :33.09   Mean   :0.5230   Mean   :30.86
##  3rd Qu.:190.00   3rd Qu.:37.10   3rd Qu.:0.6870   3rd Qu.:36.00
##  Max.   :846.00   Max.   :67.10   Max.   :2.4200   Max.   :81.00
```

```
# Checking how balance is with the dependent variable
prop.table(table(data$diabetes))
```

```
##
##       neg       pos
## 0.6683673 0.3316327
```

Randomly split the dataset into a training set (approximately 2/3 of the sample size) and a test set, such that the class distributions (i.e. the empirical distribution of diabetes) is similar in the two sets.

```r
set.seed(123)

sample <- sample.split(data$diabetes, SplitRatio = 2/3)
train <- subset(data, sample == TRUE)
test <- subset(data, sample == FALSE)
```

```r
# Class distribution in the training set
prop.table(table(train$diabetes))
```

```
##
##       neg       pos
## 0.6679389 0.3320611
```

```r
# Class distribution in the testing set
prop.table(table(test$diabetes))
```

```
##
##       neg       pos
## 0.6692308 0.3307692
```

```r
cat("Dimension of Training Set:", paste(dim(train), collapse = "x"), "\nDimension of Test Set:", paste(
```

```
## Dimension of Training Set: 262x9
## Dimension of Test Set: 130x9
```

a. k-NN classifier

```r
X_train <- train[, -ncol(train)]
y_train <- train$diabetes

X_test <- test[, -ncol(test)]
y_test <- test$diabetes
```

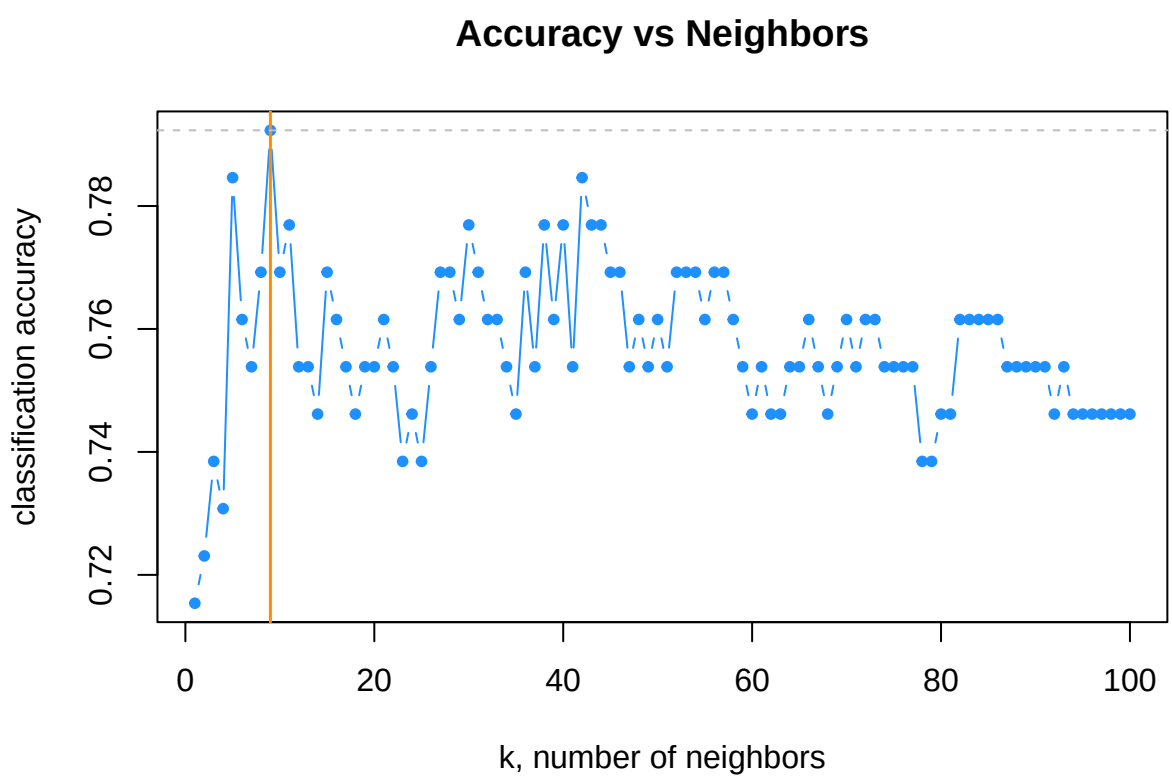
```r
accuracy = function(actual, predicted) {
  mean(actual == predicted)
}

set.seed(42)
k_to_try = 1:100
acc_k = rep(0, length(k_to_try))

# Loop over values of k
for (i in seq_along(k_to_try)) {
  pred <- knn(
    train = scale(X_train),
    test = scale(X_test),
    cl = y_train,
    k = k_to_try[i]
  )
  acc_k[i] <- accuracy(y_test, pred)
}
```

```r
ex_seq = seq(from = 1, to = 100, by = 5)
ex_storage = rep(x = 0, times = length(ex_seq))
for(i in seq_along(ex_seq)) {
  ex_storage[i] = mean(rnorm(n = 10, mean = ex_seq[i], sd = 1))
}

ex_storage
```

```
##  [1]  1.547297  5.836543 10.821920 15.636096 20.979785 26.018394 31.539077
##  [8] 35.782125 41.251106 45.912805 51.229248 55.801428 60.205034 66.210242
## [15] 70.797793 75.754132 81.101802 86.093609 90.743936 96.187940
```

We see that $k = 9$ are tied for the highest accuracy. Also notice that, as k increases, eventually the accuracy approaches the test prevalence.

```r
# Reference: https://daviddalpiaz.github.io/r4sl/k-nearest-neighbors.html

# plot accuracy vs choice of k
plot(acc_k, type = "b", col = "dodgerblue", cex = 1, pch = 20,
     xlab = "k, number of neighbors", ylab = "classification accuracy",
     main = "Accuracy vs Neighbors")
# add lines indicating k with best accuracy
abline(v = which(acc_k == max(acc_k)), col = "darkorange", lwd = 1.5)
# add line for max accuracy seen
abline(h = max(acc_k), col = "grey", lty = 2)
# add line for prevalence in test set
abline(h = mean(y_test == "No"), col = "grey", lty = 2)
```




```
cat("Number of optimal k: ", paste(max(which(acc_k == max(acc_k)))),
    "\nAccuracy: ", paste(max(acc_k)))
```

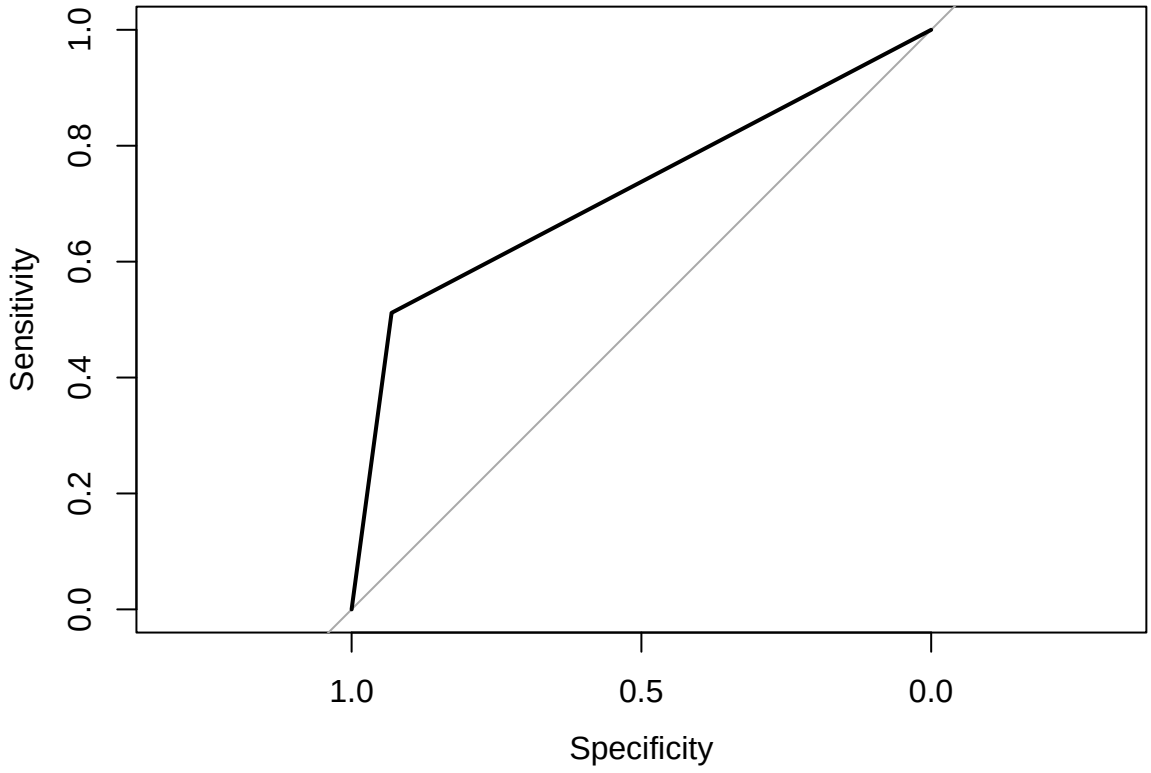
```
## Number of optimal k:  9
## Accuracy:  0.792307692307692
```

As the plot below, we can see that k-NN classifier has moderate discriminative ability. This performance aligns with the AUC of 0.721, which reflects that k-NN is a reasonably effective classifier.

```
cat("k-NN\nAUC:", auc_knn,
    "\nAccuracy:", knn_accuracy)
```

```
## k-NN
## AUC: 0.7213312
## Accuracy: 0.7923077
```

```
# Plot ROC
plot(roc_curve_knn)
```

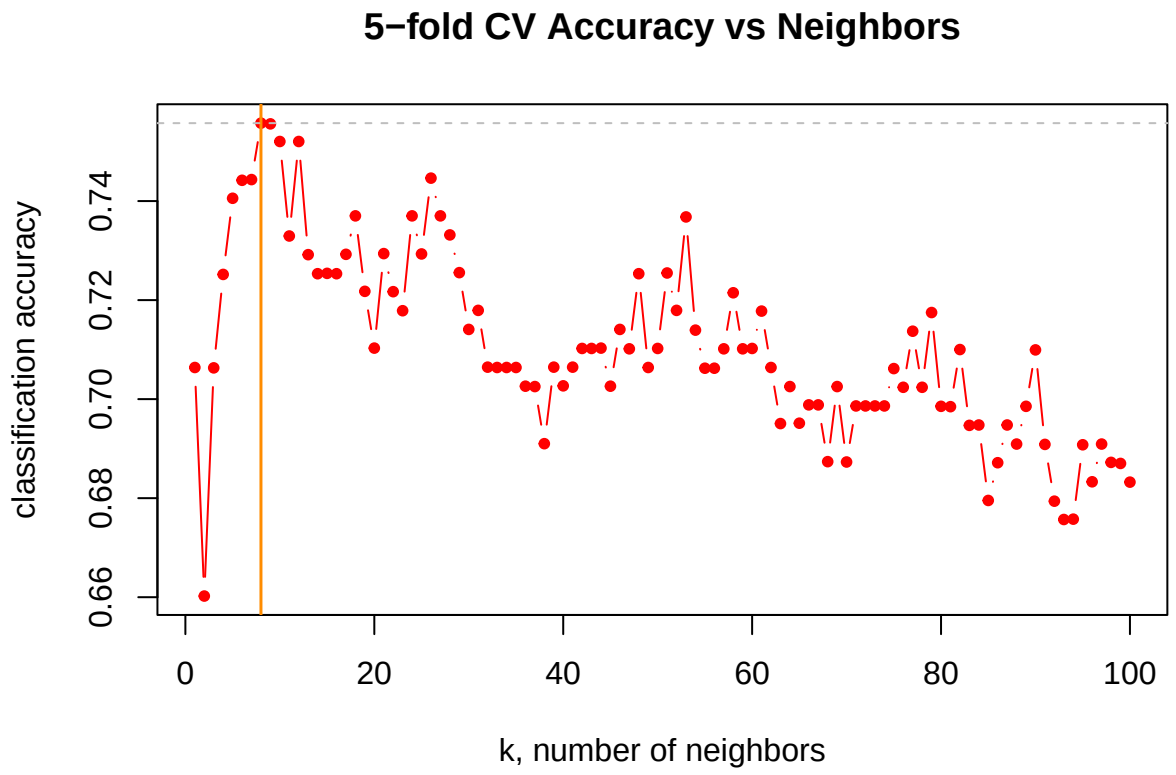


Using 5-fold

```
# 5-fold cross-validation using caret
train_control <- trainControl(method = "cv", number = 5)
train_knn <- train(diabetes ~ .,
                   data = train,
                   method = "knn",
                   trControl = train_control,
                   tuneGrid = expand.grid(k = k_to_try))

## plot(train_knn)

cv_results <- train_knn$results

head(cv_results)
```

```
##   k  Accuracy     Kappa AccuracySD   KappaSD
## 1 1 0.7063861 0.3512222 0.05274457 0.1132618
## 2 2 0.6602322 0.2180471 0.05024751 0.1199193
## 3 3 0.7063135 0.3304328 0.05489854 0.1044619
## 4 4 0.7251814 0.3627845 0.05461723 0.1114380
## 5 5 0.7405660 0.3922546 0.07858248 0.1721106
## 6 6 0.7441945 0.3957179 0.08260038 0.2122957
```

```r
# Plot the 5-fold CV accuracy vs choice of k

plot(cv_results$k,
     cv_results$Accuracy,
     type = "b",
     col = "red",
     cex = 1,
     pch = 20,
     xlab = "k, number of neighbors", ylab = "classification accuracy",
     main = "5-fold CV Accuracy vs Neighbors")

# Add lines indicating k with best accuracy
abline(v = cv_results$k[which.max(cv_results$Accuracy)],
       col = "darkorange",
       lwd = 1.5)
# Add line for max accuracy seen
abline(h = max(cv_results$Accuracy), col = "grey", lty = 2)
```



Applying 5-fold CV, k = 8 is the optimal number of neighbors for the k-NN model. And the model achieved an accuracy of 75.57% on the test set.

```r
cat("k-NN (5 fold CV)",
    "\nNumber of optimal k: ",
    paste(cv_results$k[which.max(cv_results$Accuracy)]),
    "\nAccuracy: ",
    paste(max(cv_results$Accuracy)))
```

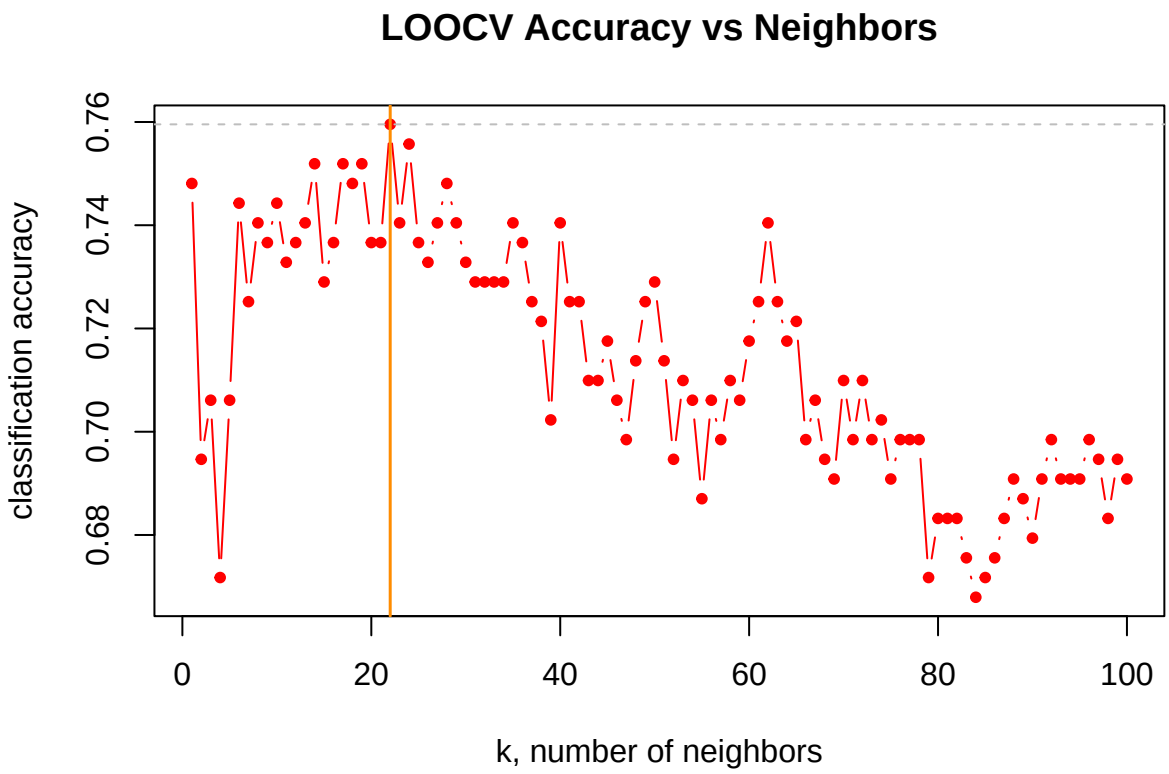
```
## k-NN (5 fold CV)
## Number of optimal k:  8
## Accuracy:  0.755732946298984
```

Using leave-one-out cross-validation

```
# leave-one-out cross-validation using caret
train_control_loocv <- trainControl(method = "LOOCV")
train_knn_loocv <- train(diabetes ~ .,
                   data = train,
                   method = "knn",
                   trControl = train_control_loocv,
                   tuneGrid = expand.grid(k = k_to_try))

cv_results_loocv <- train_knn_loocv$results

head(cv_results_loocv)
```

```
##   k  Accuracy     Kappa
## 1 1 0.7480916 0.4353817
## 2 2 0.6946565 0.3347299
## 3 3 0.7061069 0.3197788
## 4 4 0.6717557 0.2334490
## 5 5 0.7061069 0.3157181
## 6 6 0.7442748 0.4116109
```

```
# Plot the LOOCV accuracy vs choice of k

plot(cv_results_loocv$k,
     cv_results_loocv$Accuracy,
     type = "b",
     col = "red",
     cex = 1,
     pch = 20,
     xlab = "k, number of neighbors",
     ylab = "classification accuracy",
     main = "LOOCV Accuracy vs Neighbors")

# Add lines indicating k with best accuracy
abline(v = cv_results_loocv$k[which.max(cv_results_loocv$Accuracy)],
       col = "darkorange",
       lwd = 1.5)

# Add line for max accuracy seen
abline(h = max(cv_results_loocv$Accuracy), col = "grey", lty = 2)
```



With k = 22, the model achieved an accuracy of 75.95%, slightly higher than the 5-fold CV result.

```
cat("k-NN (LOOCV)",
    "\nNumber of optimal k: ",
    paste(cv_results_loocv$k[which.max(cv_results_loocv$Accuracy)]),
    "\nAccuracy: ",
    paste(max(cv_results_loocv$Accuracy)))
```

```
## k-NN (LOOCV)
## Number of optimal k:  22
## Accuracy:  0.759541984732824
```

In conclusion, LOOCV tends to select a larger k = 22 and archives slightly better accuracy (75.95%) compared to 5-fold CV with k = 8 (75.57%). This suggests that the higher k reduces the risk of overfitting by averaging predictions over a larger neighborhood of points, but 5-fold CV remains a practical alternative with nearly comparable results.

b. Generalized Additive Model (GAM)

```
# reference: <https://osf.io/wgc4f/wiki/mgcv:%20model%20selection/>

# Fit a GAM with automatic smoothness selection
gam_model <- gam(
```

```
    diabetes ~
      s(glucose) +
      s(pressure) +
      s(insulin) +
      s(mass) +
      s(pedigree) +
      s(age) +
      s(pregnant),
    data = train,
    family = binomial(link = 'logit'),
    select = TRUE,
    method= "REML")
```

In the `summary(model)` output, the Approximate significance of smooth terms table shows an estimated degrees of freedom (edf) and Chi square score (Chi.sq) close to zero, with a p-value > 0.05.

```
summary(gam_model)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## diabetes ~ s(glucose) + s(pressure) + s(insulin) + s(mass) +
##     s(pedigree) + s(age) + s(pregnant)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.1305     0.1918  -5.895 3.75e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                   edf Ref.df Chi.sq  p-value
## s(glucose)  9.761e-01      9 36.884  < 2e-16 ***
## s(pressure) 1.297e-05      9  0.000 0.832820
## s(insulin)  5.976e-06      9  0.000 0.697822
## s(mass)     1.889e+00      9 10.598 0.001791 **
## s(pedigree) 8.059e-01      9  3.887 0.027410 *
## s(age)      2.127e+00      9 12.199 0.000871 ***
## s(pregnant) 1.798e+00      9  4.837 0.047140 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.421   Deviance explained = 39.2%
## -REML = 114.77  Scale est. = 1         n = 262
```

We can see that there are `glucose`, `mass`, `pedigree`, `age` and `pregnant` are significant predictors of diabetes risk in this GAM model (based on the p-value is greater than 0.05). `pressure` and `insulin` have p-values > 0.05, suggesting they do not contribute meaningfully to the model and could be removed to simplify it.

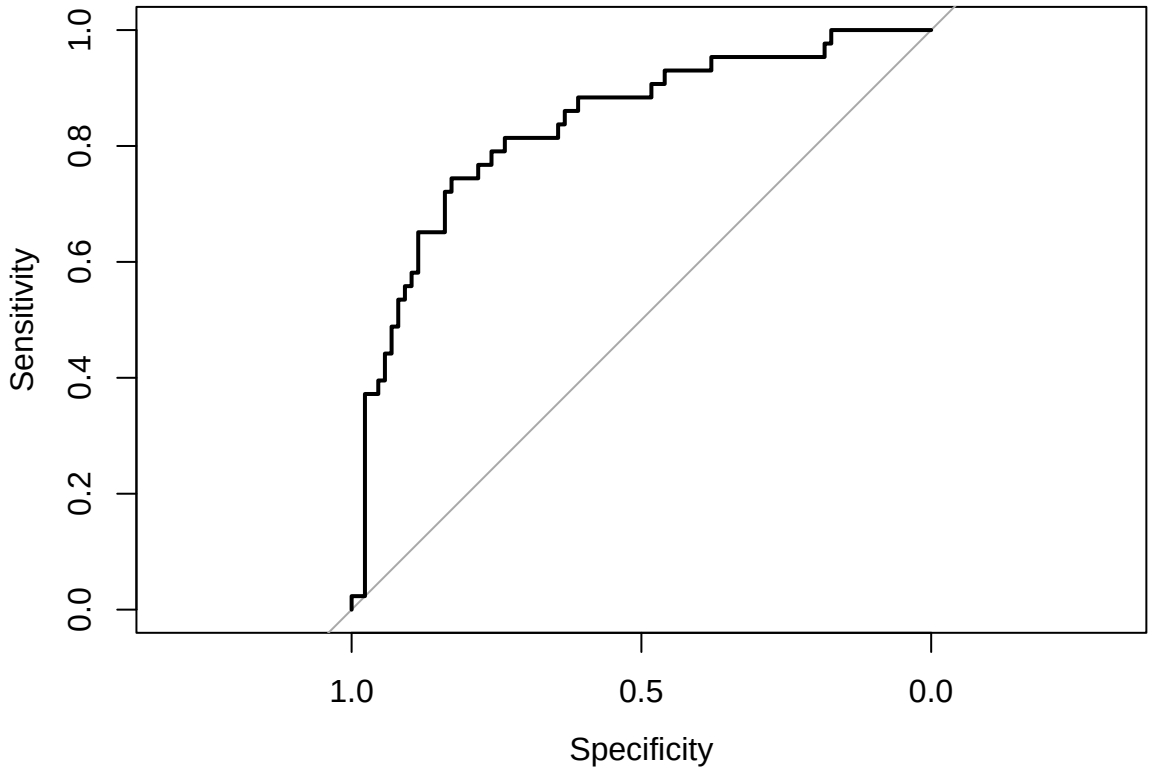
```r
cat("GAM\nTest Accuracy:", gam_accuracy)
```

```
## GAM
## Test Accuracy: 0.7769231
```

```r
cat("\nAUC:", auc_gam, "\n")
```

```
##
## AUC: 0.833467
```

```r
# Plot ROC
plot(roc_curve_gam)
```



In conclusion, The GAM model with spline smoothing achieves a test accuracy of 77.69%. The AUC for GAM is 0.833, indicating that it has strong performance in distinguishing between diabetic and non-diabetic individuals.

c. Tree-based methods

Setting up the k-fold cross validation k = 10 cross-validation folds for all tree based method.

```r
# reference: https://quantdev.ssri.psu.edu/sites/qdev/files/09_EnsembleMethods_2017_1127.html
```

```r
set.seed(1234)

# Setting up cross-validation
cvcontrol <- trainControl(method="repeatedcv",
                          number = 10,
                          allowParallel=TRUE)
```

(i) Classification tree

```r
train.tree <- train(as.factor(diabetes) ~ .,
                    data=train,
                    method="ctree",
                    trControl=cvcontrol,
                    tuneLength = 10)
train.tree
```

```
## Conditional Inference Tree
##
## 262 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 235, 236, 235, 236, 235, 237, ...
## Resampling results across tuning parameters:
##
##   mincriterion  Accuracy   Kappa
##   0.0100000     0.7931567  0.5294069
##   0.1188889     0.7855954  0.4963222
##   0.2277778     0.7781880  0.4816163
##   0.3366667     0.7780456  0.4835965
##   0.4455556     0.7780456  0.4835965
##   0.5544444     0.7817493  0.4930560
##   0.6633333     0.7817493  0.4930560
##   0.7722222     0.7853105  0.4941199
##   0.8811111     0.7664957  0.4422783
##   0.9900000     0.7371852  0.3930795
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mincriterion = 0.01.
```

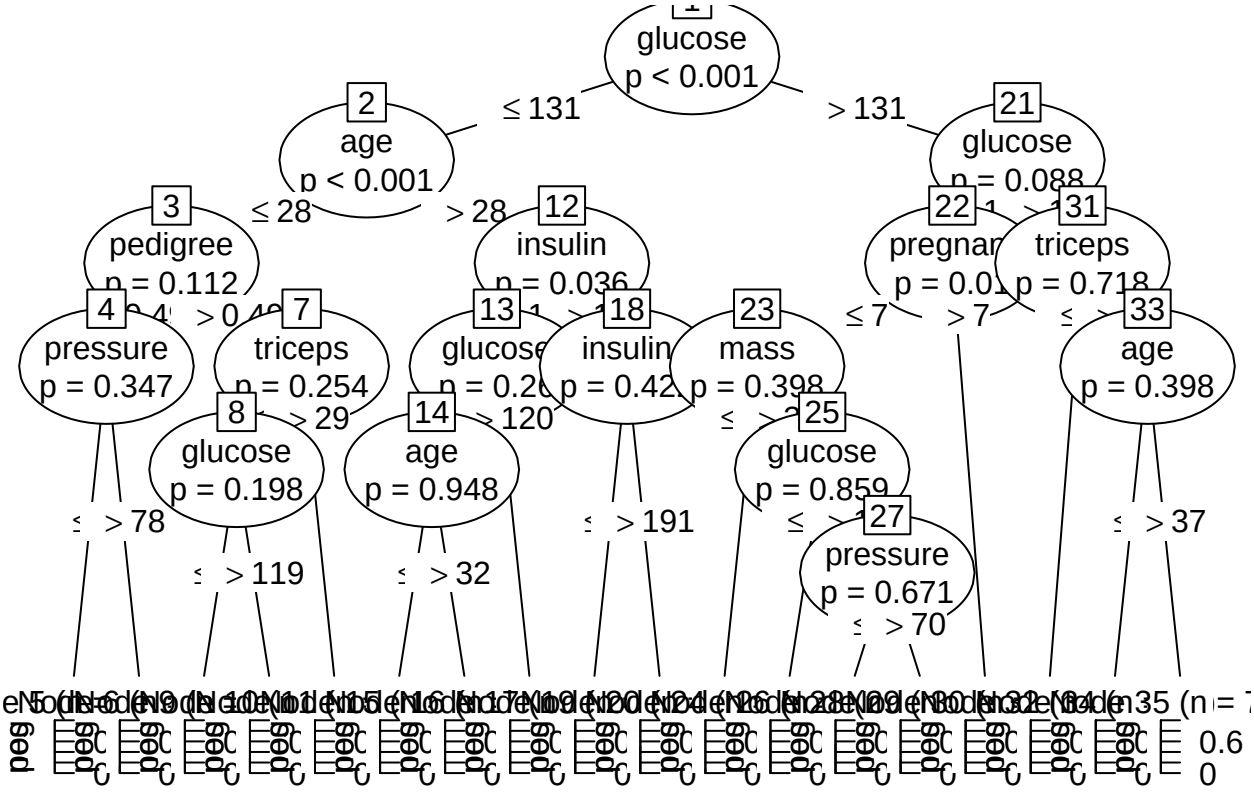
We can see that the optimal mincriterion value selected was 0.01, which provided the highest accuracy of 79.32% and a Kappa of 0.5294 during cross-validation.

- The root node splits based on glucose (with a p-value < 0.001). If glucose is $<= 131$, the model then considers age (with a split at age $<= 28$). This indicates that glucose is the first major factor, followed by age for further classification.
- Other significant splits include conditions based on insulin, pedigree, prenants and mass for certain branches.

- The end nodes represent the final classification of each branch, either “pos” or “neg” for diabetes, along with the proportion of samples in each class.

This aligns with known risk factors for diabetes, where higher glucose levels, older age, higher BMI, and pregnancy history (for gestational diabetes) are associated with a higher risk.

```
plot(train.tree$finalModel)
```



On the training set, the classification tree provides a high level of interpretability, accurately classifying diabetes status with an accuracy of 86.64% and balanced sensitivity and specificity.

```
# obtaining class predictions for training
tree.classTrain <-  predict(train.tree, type="raw")

# Check accuracy and confusion matrix for training set
confusionMatrix(train$diabetes, tree.classTrain)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg 164  11
##        pos  24  63
##
##                Accuracy : 0.8664
##                  95% CI : (0.8191, 0.9052)
```

```
##     No Information Rate : 0.7176
##     P-Value [Acc > NIR] : 7.327e-09
##
##                   Kappa : 0.6871
##
##  Mcnemar's Test P-Value : 0.04252
##
##             Sensitivity : 0.8723
##             Specificity : 0.8514
##          Pos Pred Value : 0.9371
##          Neg Pred Value : 0.7241
##              Prevalence : 0.7176
##          Detection Rate : 0.6260
##    Detection Prevalence : 0.6679
##       Balanced Accuracy : 0.8618
##
##        'Positive' Class : neg
##
```

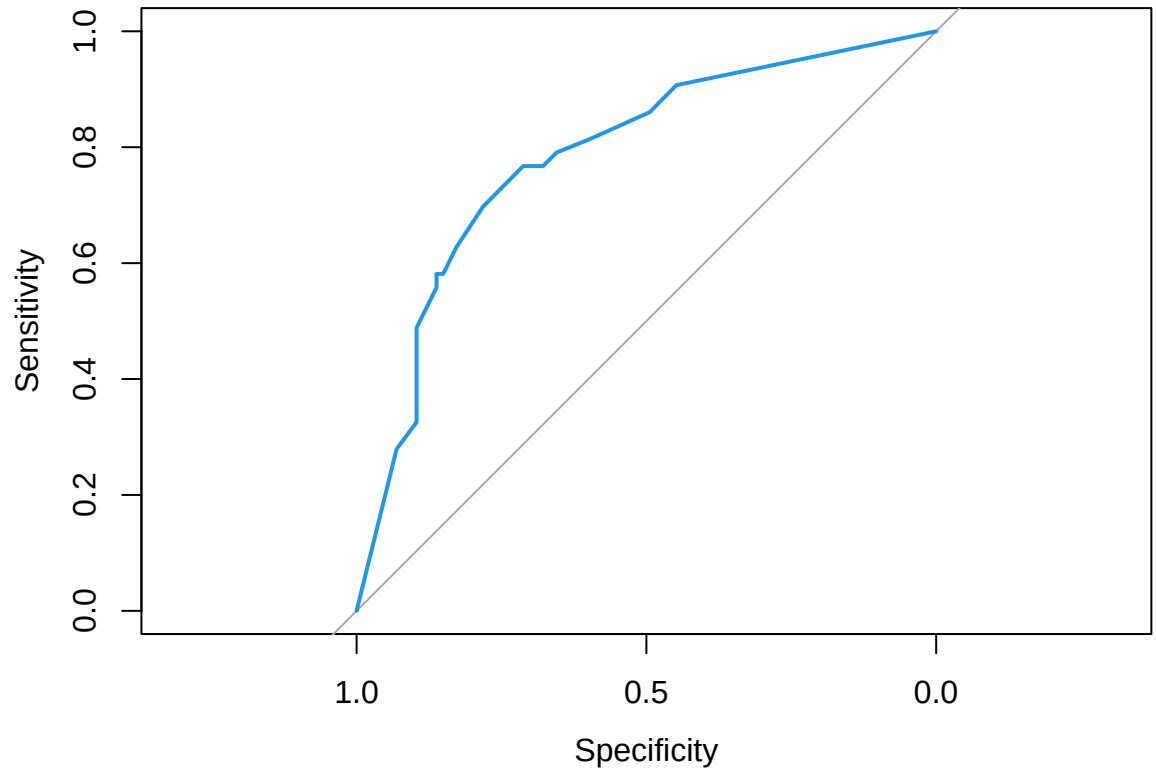
On the test set, the model's performance slightly declined, with accuracy dropping to 76.15% and Kappa reducing to 0.4451. Sensitivity remained relatively high at 80.43%, but specificity decreased to 65.79%.

```
# Obtaining class predictions for test set
tree.classTest <-  predict(train.tree,
                           newdata = test,
                           type="raw")

# Check accuracy and confusion matrix for training set
confusionMatrix(test$diabetes, tree.classTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg  74  13
##        pos  18  25
##
##                Accuracy : 0.7615
##                  95% CI : (0.6789, 0.8319)
##     No Information Rate : 0.7077
##     P-Value [Acc > NIR] : 0.1034
##
##                   Kappa : 0.4451
##
##  Mcnemar's Test P-Value : 0.4725
##
##             Sensitivity : 0.8043
##             Specificity : 0.6579
##          Pos Pred Value : 0.8506
##          Neg Pred Value : 0.5814
##              Prevalence : 0.7077
##          Detection Rate : 0.5692
##    Detection Prevalence : 0.6692
```

```
##        Balanced Accuracy : 0.7311
##
##         'Positive' Class : neg
##
```

```
#plot the ROC curve
plot(rocCurve.tree,col=c(4))
```




```
cat("Classification Tree",
    "\nAccuracy:", tree_accuracy,
    "\nAUC:",auc(rocCurve.tree)
    )
```

```
## Classification Tree
## Accuracy: 0.7615385
## AUC: 0.7848169
```

(ii) Ensemble of bagged trees

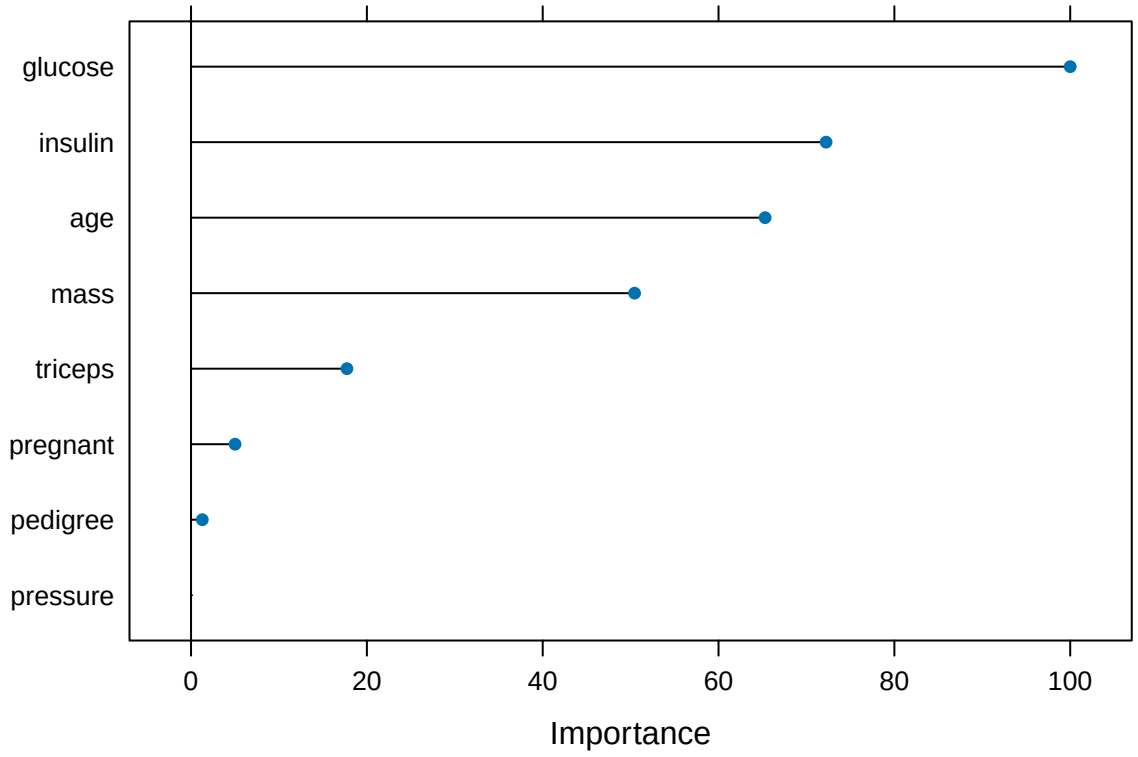
```
train.bagg <- train(as.factor(diabetes) ~ .,
                    data=train,
                    method="treebag",
                    trControl=cvcontrol,
```

```
                  importance=TRUE)

train.bagg
```

```
## Bagged CART
##
## 262 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 235, 237, 235, 235, 236, 236, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7817721  0.4906951
```

The importance plot shows that glucose and insulin are the top predictors, followed by age and mass (BMI). These variables align well with known diabetes risk factors, reinforcing the model's reliance on medically relevant predictors. Less important variables include triceps, pregnant, pedigree, and pressure, which have minimal impact on the model's decisions.

```
plot(varImp(train.bagg))
```



```r
#obtaining class predictions
bagg.classTrain <-  predict(train.bagg, type="raw")

confusionMatrix(train$diabetes, bagg.classTrain)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg 175   0
##        pos   0  87
##
##                Accuracy : 1
##                  95% CI : (0.986, 1)
##     No Information Rate : 0.6679
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6679
##          Detection Rate : 0.6679
##    Detection Prevalence : 0.6679
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : neg
##
```

To evaluate the accuracy of the Bagged Trees we can look at the confusion matrix for the training data. It indicates an accuracy of 100%, which may result in overfitting, particularly when evaluated on new data.

```r
bagg.classTest <-  predict(train.bagg,
                           newdata = test,
                           type="raw")

confusionMatrix(test$diabetes, bagg.classTest)
```

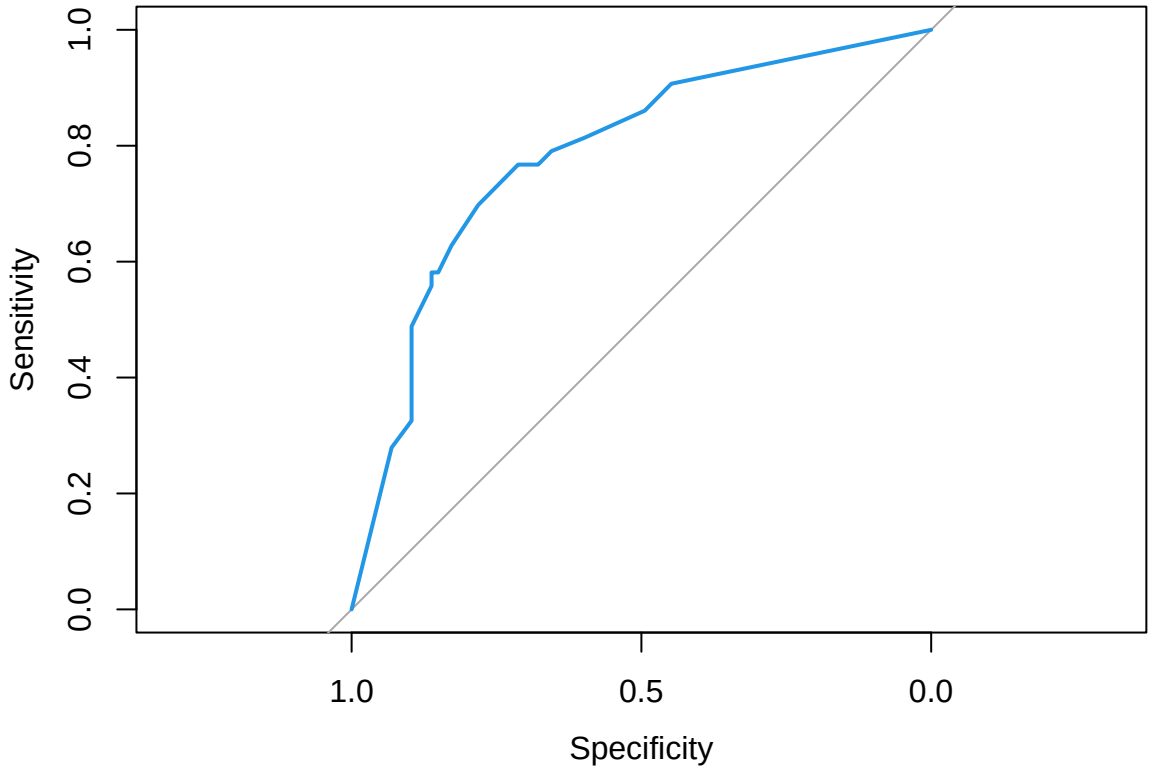
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg  74  13
##        pos  17  26
##
##                Accuracy : 0.7692
##                  95% CI : (0.6872, 0.8386)
##     No Information Rate : 0.7
```

```
##     P-Value [Acc > NIR] : 0.04933
##
##                   Kappa : 0.4662
##
##  Mcnemar's Test P-Value : 0.58388
##
##             Sensitivity : 0.8132
##             Specificity : 0.6667
##          Pos Pred Value : 0.8506
##          Neg Pred Value : 0.6047
##              Prevalence : 0.7000
##          Detection Rate : 0.5692
##    Detection Prevalence : 0.6692
##       Balanced Accuracy : 0.7399
##
##        'Positive' Class : neg
##
```

Applying for the testing data, the accuracy is 76.92% and Kappa is 0.4662 showing moderate agreement on the test set and slightly better than the classification tree's Kappa score, which was 0.4451

The ROC curve for the bagged trees shows a smoother and better-performing curve than the single classification tree. AUC is bagged tree is 0.8311, which is higher than the AUC of the single classification tree (0.7848). This suggests that the ensemble model has a stronger ability to discriminate between diabetic and non-diabetic cases across different thresholds.

```
# plot the ROC curve
plot(rocCurve.tree,col=c(4))
```




```
cat("Classification Tree",
    "\nAccuracy:", bagg_accuracy,
    "\nAUC:",auc(rocCurve.bagg)
    )
```

```
## Classification Tree
## Accuracy: 0.7692308
## AUC: 0.8310612
```

(iii) Random Forest

Applying Random Forest, which builds multiple decision trees and averages their predictions to create a robust classifier. As shown in the summary table below, the best hyperparameter `mtry` (the number of predictors to sample for each split) was found to be 5, yielding the highest accuracy (80.14%) and Kappa (0.5417) among the tested values.

```
#Using treebag
train.rf <- train(as.factor(diabetes) ~ .,
                  data=train,
                  method="rf",
                  trControl=cvcontrol,
                  importance=TRUE)

train.rf
```

```
## Random Forest
##
## 262 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 235, 235, 235, 236, 235, 236, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7938917  0.5193219
##   5     0.8014416  0.5417510
##   8     0.7898917  0.5141065
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

On the training set, the accuracy is 100%, with both Sensitivity and Specificity also at 100%, and a Kappa value of 1.0. This indicates that the model perfectly fits the training data, a common characteristic of Random Forest models when sufficient trees are used.

```
#obtaining class predictions
rf.classTrain <-  predict(train.rf, type="raw")

confusionMatrix(train$diabetes, rf.classTrain)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg 175   0
##        pos   0  87
##
##                Accuracy : 1
##                  95% CI : (0.986, 1)
##     No Information Rate : 0.6679
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6679
##          Detection Rate : 0.6679
##    Detection Prevalence : 0.6679
##       Balanced Accuracy : 1.0000
##
```

```
##        'Positive' Class : neg
##
```

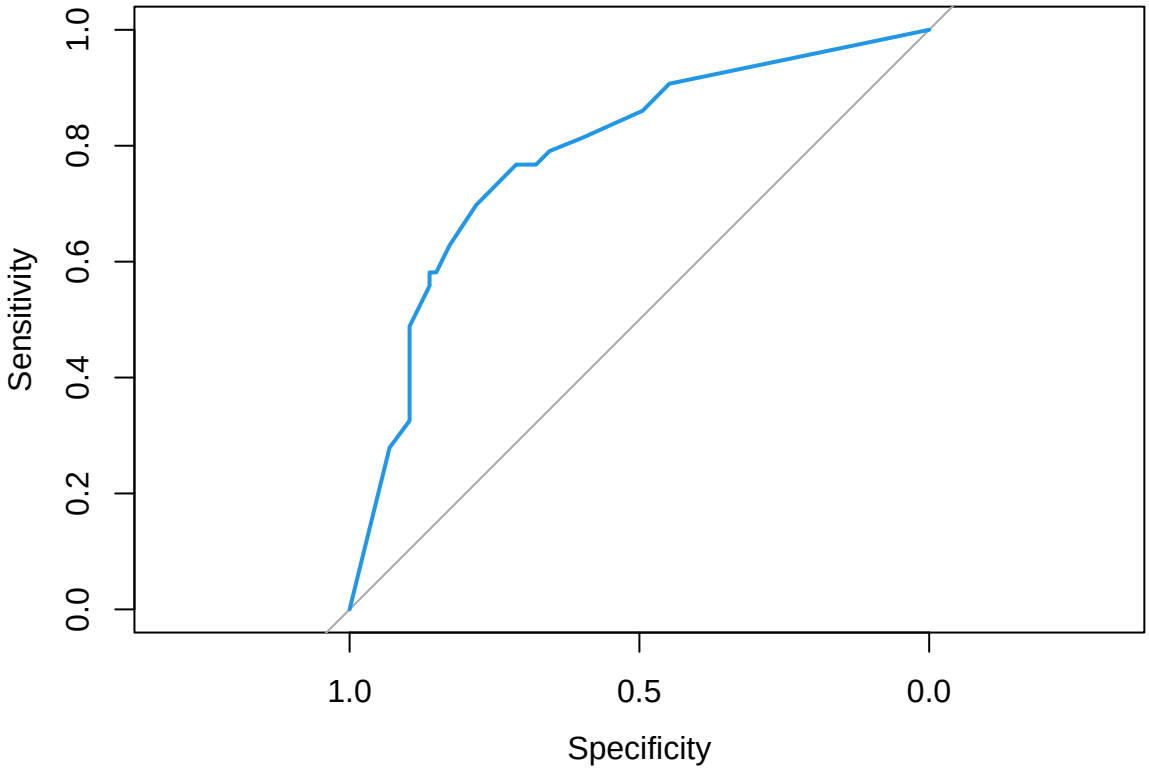
On the test set, the accuracy is 76.92%, consistent with earlier models such as bagged trees and the single decision tree. The sensitivity value is 82.02%, indicating that the model is effective at identifying non-diabetic cases. However, the specificity is 65.85%, suggesting that it is less effective at detecting diabetic cases, resulting in some false negatives.

```
rf.classTest <-  predict(train.rf,
                         newdata = test,
                          type="raw")

confusionMatrix(test$diabetes, rf.classTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg  73  14
##        pos  16  27
##
##                Accuracy : 0.7692
##                  95% CI : (0.6872, 0.8386)
##     No Information Rate : 0.6846
##     P-Value [Acc > NIR] : 0.02154
##
##                   Kappa : 0.4725
##
##  Mcnemar's Test P-Value : 0.85513
##
##             Sensitivity : 0.8202
##             Specificity : 0.6585
##          Pos Pred Value : 0.8391
##          Neg Pred Value : 0.6279
##              Prevalence : 0.6846
##          Detection Rate : 0.5615
##    Detection Prevalence : 0.6692
##       Balanced Accuracy : 0.7394
##
##        'Positive' Class : neg
##
```

The AUC is 0.8292, which is slightly higher than the AUC of the bagged trees (0.8311) and the single decision tree (0.7848). This suggests that the Random Forest model demonstrates strong discriminatory power.

```
# plot the ROC curve
plot(rocCurve.tree,col=c(4))
```




```
cat("Classification Tree",
    "\nAccuracy:", rf_accuracy,
    "\nAUC:", auc(rocCurve.rf)
    )
```

```
## Classification Tree
## Accuracy: 0.7692308
## AUC: 0.8291901
```

In conclusion, the Random Forest shows the best Kappa and AUC among the models, indicating its slight advantage in discriminating between classes on the test set.All models achieve similar test set accuracy, but the Random Forest has a small improvement in sensitivity. Specificity is consistent across models, showing that all models have limitations in correctly identifying diabetic cases.

d. Neural Network

In this problem, I am using a simple neural network. To prepare the data for the neural network, we need to:

- Feature Scaling: Both the training and test sets were scaled to standardize the input features. This enhances training effectiveness and helps the model converge more quickly. It's particularly crucial in neural networks, as it ensures that each feature has a similar range, preventing any single feature from dominating the others.

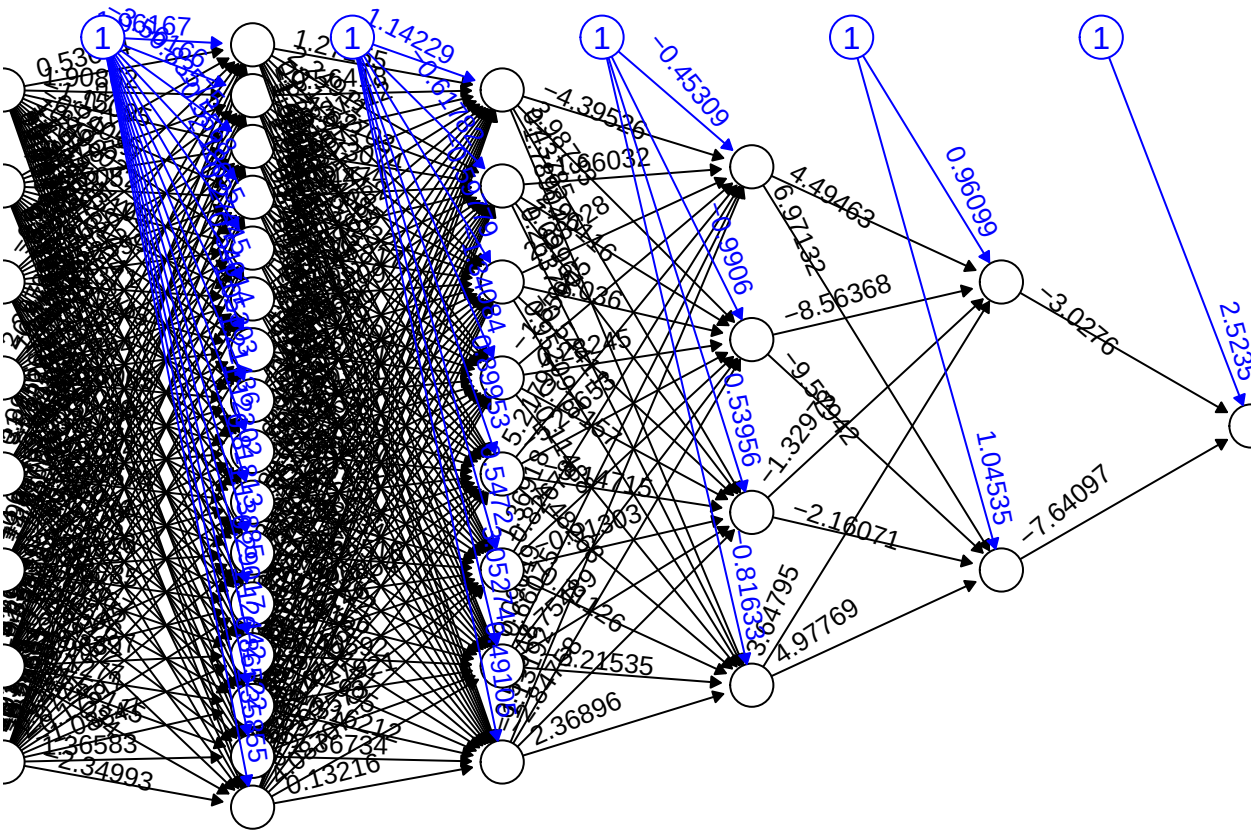
- Binary Conversion of Target Variable: The target variable (diabetes) was transformed into a binary format of 0 and 1, where 1 indicates a positive diagnosis (diabetic) and 0 indicates a negative diagnosis (non-diabetic).

```
# reference: https://www.datacamp.com/tutorial/neural-network-models-r

# Extract features and labels
X_train_nn <- as.data.frame(train[, -ncol(train)])
y_train_nn <- as.numeric(train$diabetes == "pos")  # Convert to binary 0/1
X_test_nn <- as.data.frame(test[, -ncol(test)])
y_test_nn <- as.numeric(test$diabetes == "pos")

# Scale the features
X_train_nn_scaled <- scale(X_train_nn)
X_test_nn_scaled <- scale(X_test_nn)

# Combine scaled features and labels for training
train_combined <- cbind(X_train_nn_scaled, diabetes = y_train_nn)
```

I selected the network size by starting with a larger number of neurons in the initial layers and progressively decreasing the number of neurons in each subsequent layer. Thus, there are four hidden layers containing 16, 8, 4, and 2 neurons, respectively. This configuration is often referred to as a funnel-shaped network, which is commonly used for binary classification tasks.

```
nn_model = neuralnet(
    diabetes~.,
    data=train_combined,
    hidden=c(16,8,4,2),
    linear.output = FALSE
)
```

```
plot(nn_model,rep = "best")
```



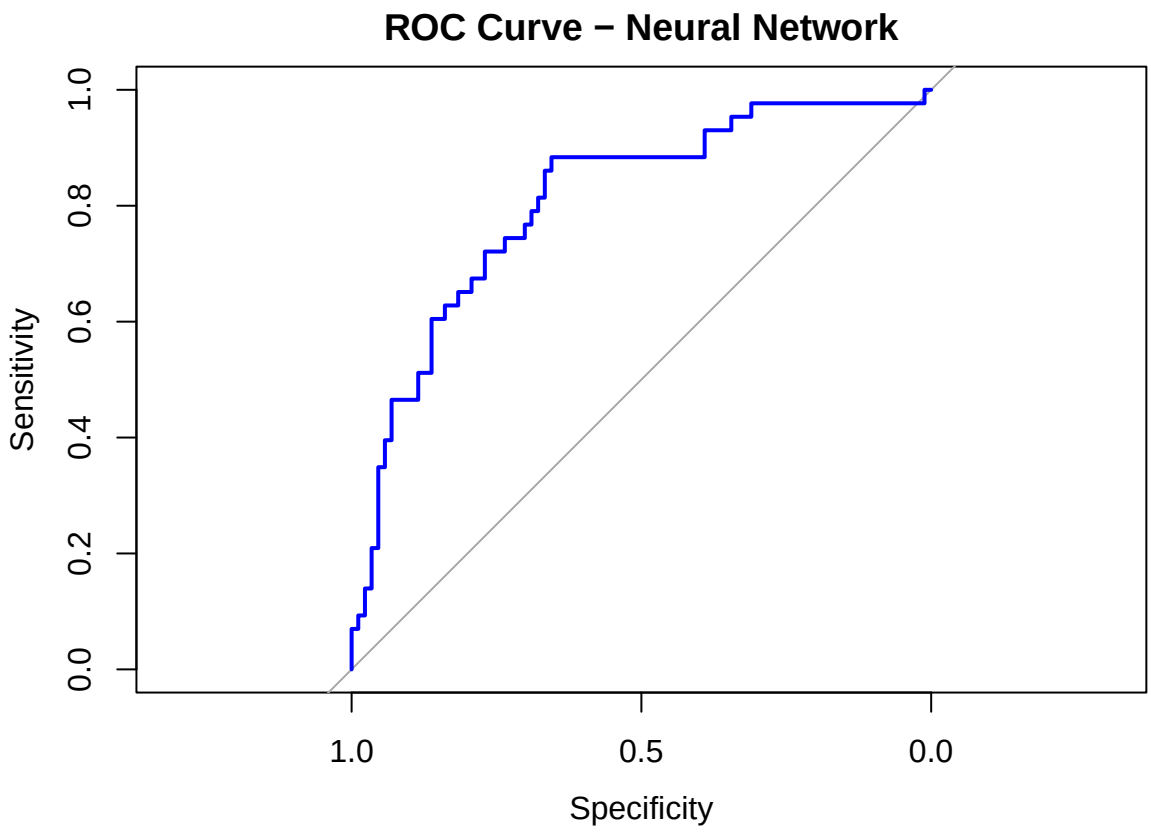

```
# Predict probabilities on test set
nn_pred <- compute(nn_model, X_test_nn_scaled)
nn_pred_prob <- nn_pred$net.result

# Convert probabilities to binary predictions
nn_pred_class <- ifelse(nn_pred_prob > 0.5, 1, 0)
```

```
# Calculate and print confusion matrix
conf_matrix <- table(Predicted = nn_pred_class, Actual = as.numeric(test$diabetes == "pos"))
print(conf_matrix)
```

```
##          Actual
## Predicted  0  1
##         0 72 16
##         1 15 27
```

In conclusion, the Neural Network achieved 76.15% accuracy and an AUC of 0.8067 on the test set, which is comparable to the performance of the Random Forest. The architecture was selected with a funnel-shaped design to capture complex relationships and reduce data to simpler representations in the final layers.

```
# Plot the ROC curve
plot(roc_curve_nn, col = "blue", main = "ROC Curve - Neural Network")
```




```
cat("Neural Network",
    "\nAccuracy:", nn_accuracy,
    "\nAUC:", auc_nn
)
```

```
## Neural Network
## Accuracy: 0.7615385
## AUC: 0.8067362
```

e. Comparision

I would choose Random Forest as the preferred method because (i) it achieves a strong balance between high accuracy and AUC, making it one of the top performers in both metrics, and (ii) it offers feature importance scores, allowing us to identify key predictors (such as glucose and insulin). This interpretability is especially valuable in medical or health-related data, where understanding influential factors is crucial for informed decision-making.

```
# Create a summary table for accuracy and AUC
accuracy_results <- data.frame(
  Model = c("KNN", "GAM", "Classification Tree", "Bagged Trees", "Random Forest", "Neural Network"),
  Accuracy = c(knn_accuracy, gam_accuracy, tree_accuracy, bagg_accuracy, rf_accuracy, nn_accuracy),
  AUC = c(auc_knn, auc_gam, auc(rocCurve.tree), auc(rocCurve.bagg), auc(rocCurve.rf), auc_nn)
)
```

```r
# Print the summary table
print(accuracy_results)
```

```
##                 Model  Accuracy       AUC
## 1                 KNN 0.7923077 0.7213312
## 2                 GAM 0.7769231 0.8334670
## 3 Classification Tree 0.7615385 0.7848169
## 4        Bagged Trees 0.7692308 0.8310612
## 5       Random Forest 0.7692308 0.8291901
## 6      Neural Network 0.7615385 0.8067362
```

```r
# Plot ROC curves for ALL models
plot(rocCurve.tree, col = 4, main = "ROC Curve Comparison", lwd = 2)
lines(rocCurve.bagg, col = 6, lwd = 2)  # Add ROC curve for bagged trees
lines(rocCurve.rf, col = 1, lwd = 2)    # Add ROC curve for random forest
lines(roc_curve_nn, col = "blue", lwd = 2)  # Add ROC curve for neural network
lines(roc_curve_knn, col = "green", lwd = 2)  # Add ROC curve for k-NN
lines(roc_curve_gam, col = "purple", lwd = 2)  # Add ROC curve for GAM

# Add a legend to the plot
legend("bottomright", legend = c("Classification Tree", "Bagged Trees", "Random Forest", "Neural Networ
       col = c(4, 6, 1, "blue", "green", "purple"), lwd = 2, bty = "n")
```