assignment_2

2024-10-16

Contents

Problem 1. Regression

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

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

```r
# Use 70% of dataset as training set and remaining 30% as testing set
set.seed(123)
sample <- sample.split(data$LC50, SplitRatio = 2/3)
train  <- subset(data, sample == TRUE)
test   <- subset(data, sample == FALSE)
```

```r
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) Orginal Model

Model each of them directly as a linear effect

```r
train_i = train
test_i = test
```

```r
# 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
```

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

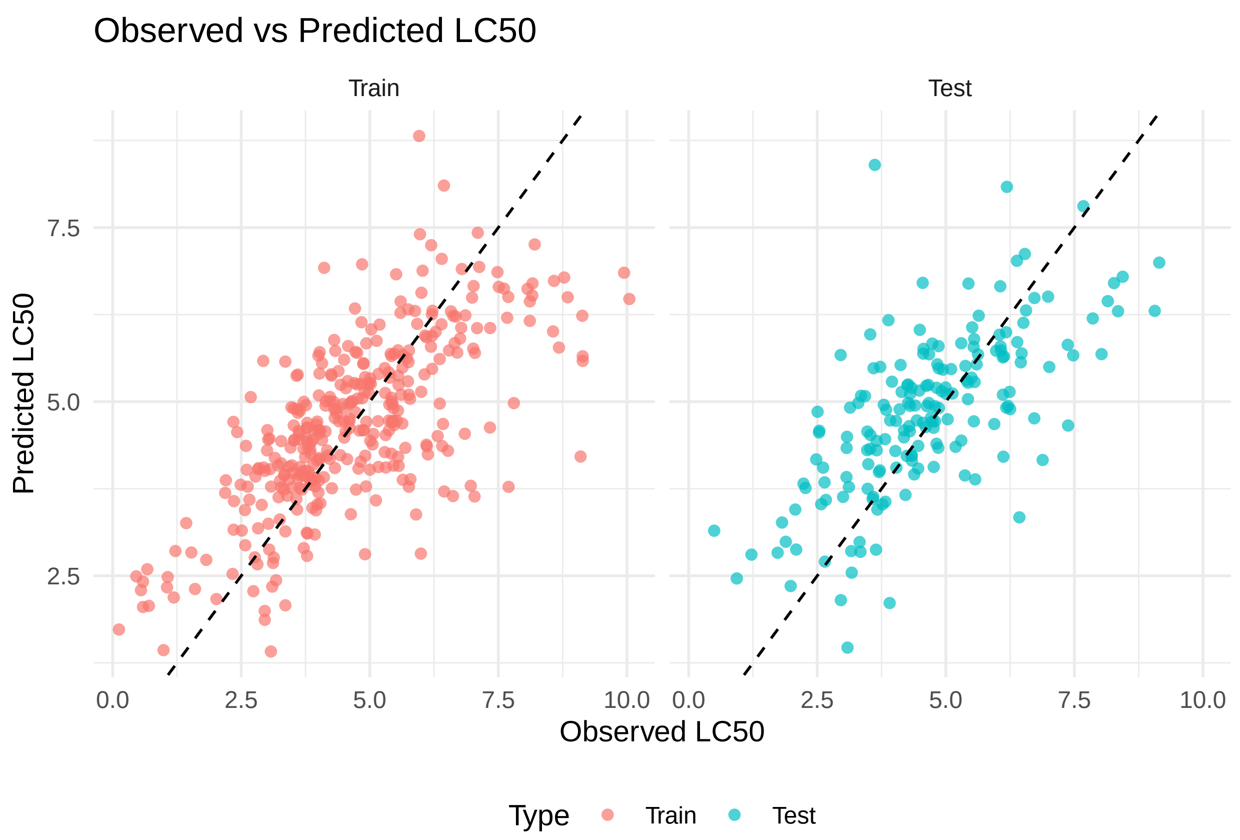
```r
# Adding predictions columns to the datasets
train_i$predicted_LC50 <- pred_train
test_i$predicted_LC50 <- pred_test
```

```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

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)
```

```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
```

```
# 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'))

# 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

Transform 3 count variables (H050, nN, C040) using a 0/1 dummy encoding where 0 represents absence of the specific atom and 1 represents presence of the specific atoms.

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

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

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)
```

```
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
```

```
# 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
```

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

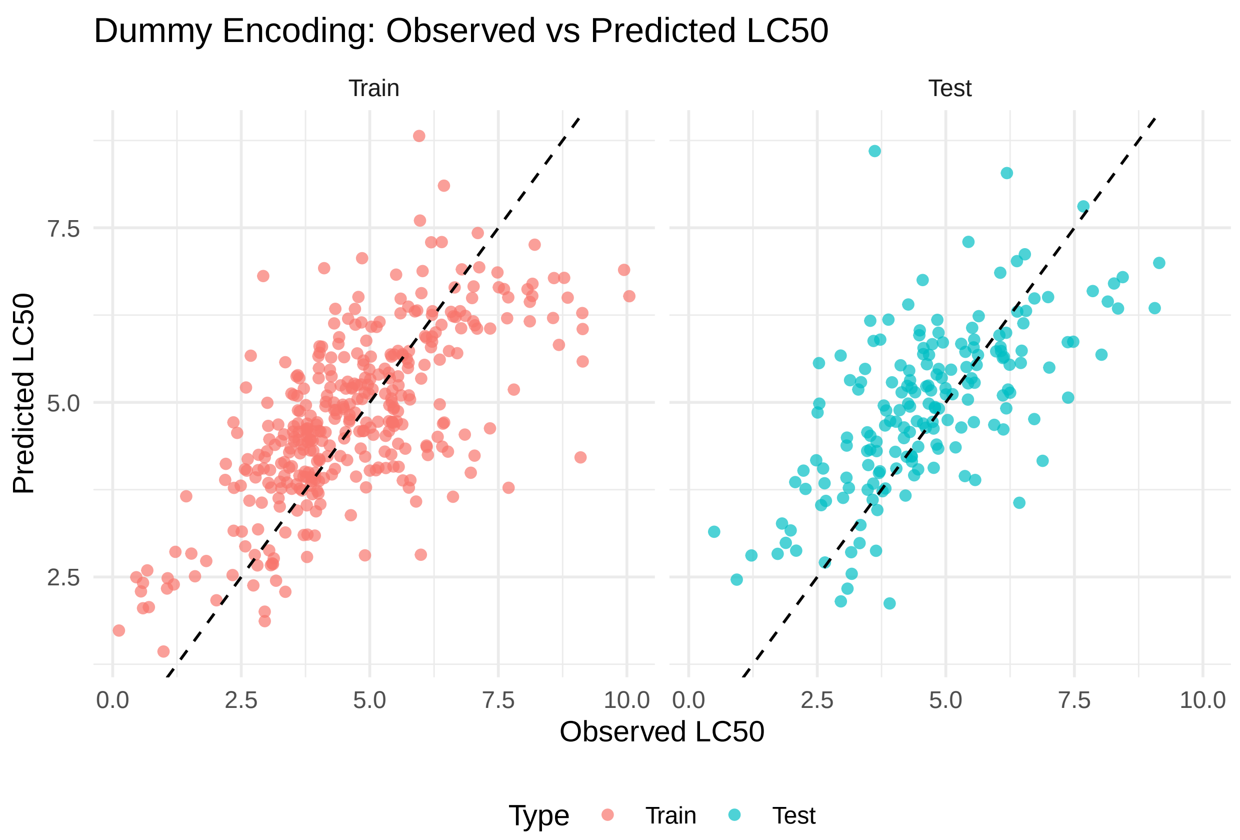
```
# Adding predictions columns to the datasets
train_ii$predicted_LC50 <- pred_train_transform_dummy
test_ii$predicted_LC50 <- pred_test_transform_dummy
```

```
# 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
```

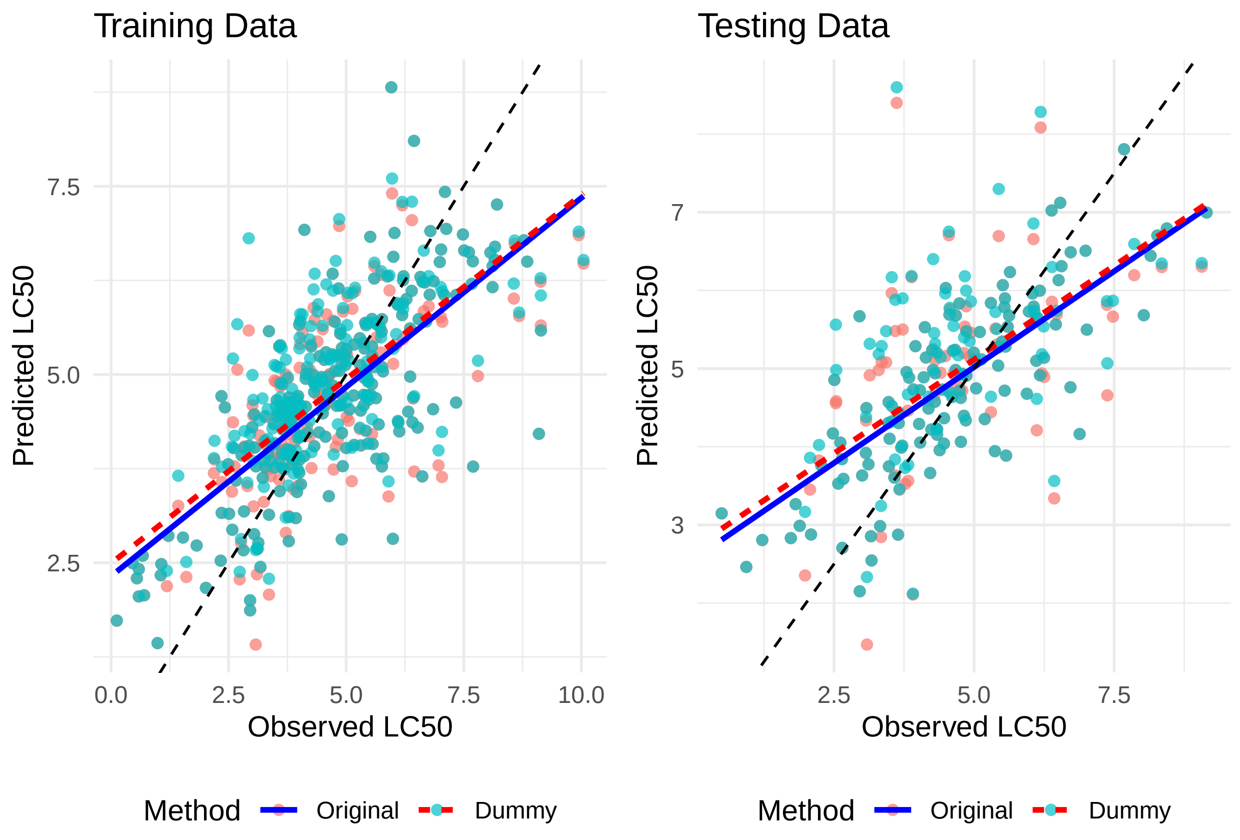
```r
cat(paste0(
  "Training Metrics:\n",
  "MSE (Train): ", mse_train_transform_dummy, "\n",
  "RMSE (Train): ", rmse_train_transform_dummy, "\n",
  "R-squared (Train): ", r2_train_transform_dummy, "\n\n",

  "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")
```



```r
# 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)
# 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)
```



b. Repeating the procedure 200 times

Procedure

- Randomly spiting training/test (70%/30%).
- Fit the models with 2 options (i) Original model and (ii) Dummy encoding.

- Record the test errors (MSE/RMSE/R^2).

```r
# 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
}
```

Make a plot that illustrates the empirical distributions of the test error for each modelling option and compare the average test error. What is the point of repeating the experiment in this way before drawing any conclusions? Try to explain why one often obtains, like in this case, a worse result by using option (ii).

Initials insight:

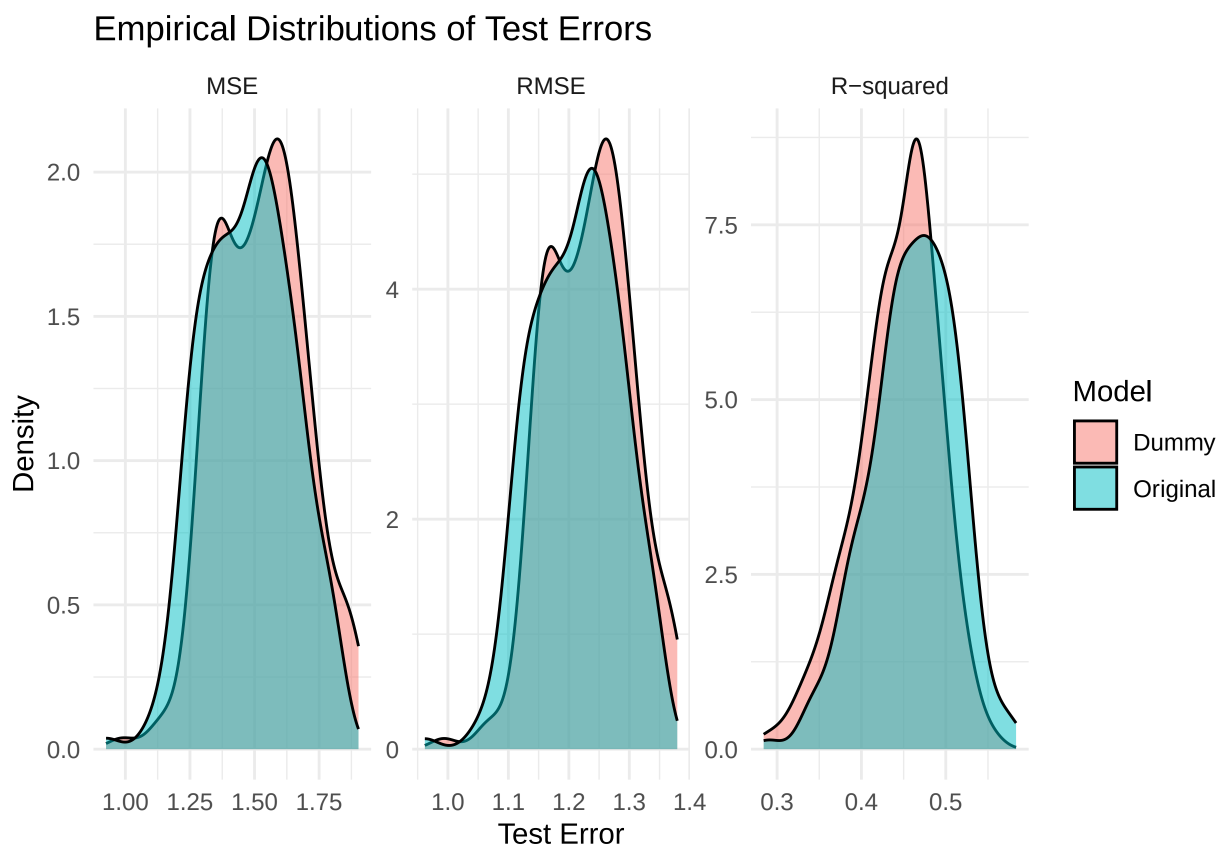
- Method 1: performs better in term of MSE
- Method 2: better in reduce over fitting

```r
# 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"
))
```

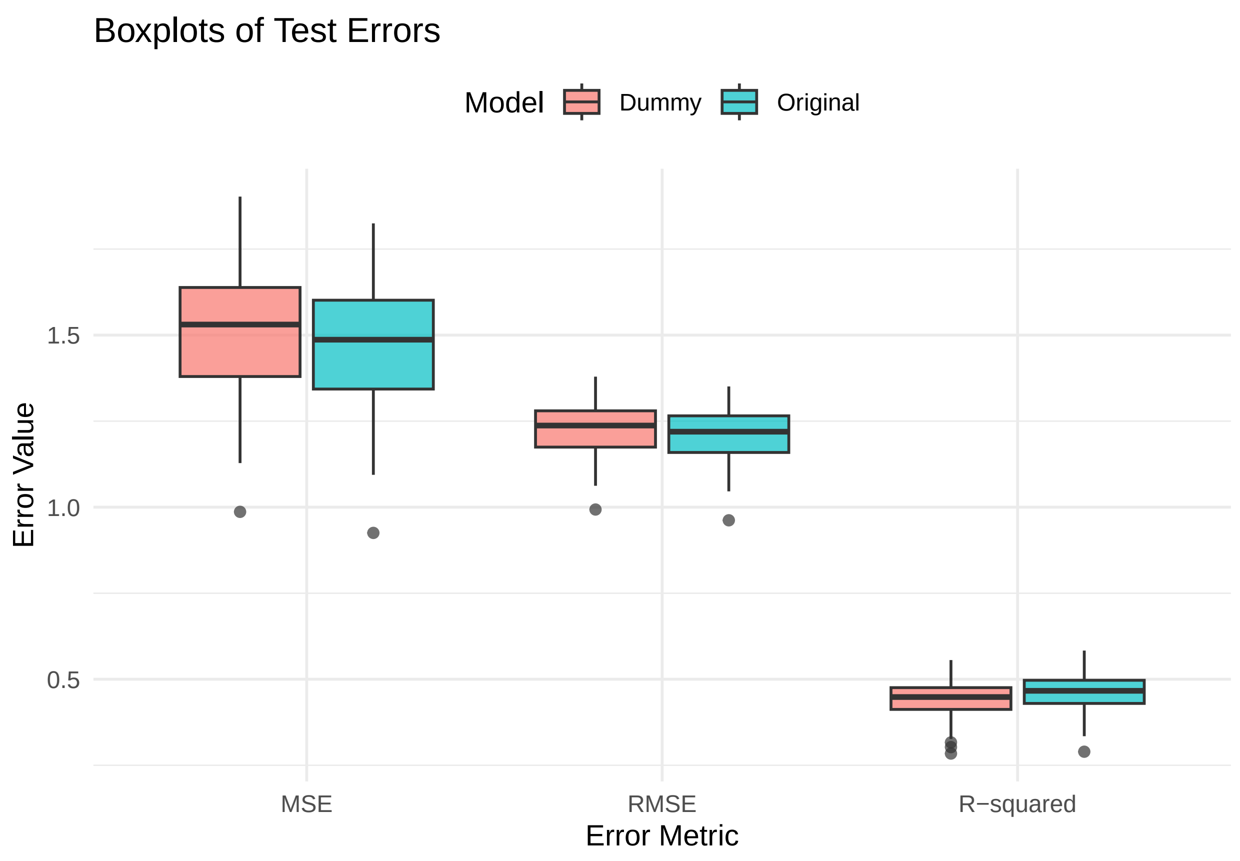
```
## Average Test Errors (Original Model):
## MSE: 1.47708146772242
## RMSE: 1.21330895603276
## R-squared: 0.460485255063669
##
## Average Test Errors (Dummy Model):
## MSE: 1.52752950559007
## RMSE: 1.23398478875802
## R-squared: 0.442128799570138
```

```r
# 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
```

```r
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()
```



```r
# 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")
```



c. Variable selection procedures

(at least backward elimination and forward selection) with different stopping criteria (at least AIC and BIC) and compare the results. Do you obtain the same model?

```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

```
# 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

```
# 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
```

```
# 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

```
# 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$pred_forward_bic <- predict(model.forward.bic, newdata = test)
test$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),
```

```r
    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
```

d. Ridge Regression

```r
set.seed(123)
sample <- sample.split(data$LC50, SplitRatio = 2/3)
train <- subset(data, sample == TRUE)
test <- subset(data, sample == FALSE)

x_train <- as.matrix(train[, -9])
y_train <- train$LC50
x_test <- as.matrix(test[, -9])
y_test <- test$LC50
```

Cross Validation

```r
# Define a grid of candidate 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)
print(paste("Test MSE for Ridge Regression (Cross-Validation):", mse_cv))
```

```
## [1] "Test MSE for Ridge Regression (Cross-Validation): 1.39984196703498"
```

Bootstrap Procedure

```r
# Define ridge regression function for bootstrap
ridge_bootstrap <- function(dataset, indices, lambda) {
  d <- dataset[indices, ]  # subset data
  x_boot <- as.matrix(d[, -9])
  y_boot <- d$LC50
  ridge_model <- glmnet(x_boot, y_boot, alpha = 0, lambda = lambda, standardize = TRUE)
  return(ridge_model)
}

# Bootstrap procedure
set.seed(123)
num_bootstraps <- 100
boot_results <- sapply(lambda_grid, function(lambda) {
  boot_mses <- replicate(num_bootstraps, {
    sample_indices <- sample(1:nrow(train), replace = TRUE)
    ridge_model <- ridge_bootstrap(train, sample_indices, lambda)
    boot_pred <- predict(ridge_model, s = lambda, newx = x_test)
    mean((boot_pred - y_test)^2)
  })
  mean(boot_mses)
})

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

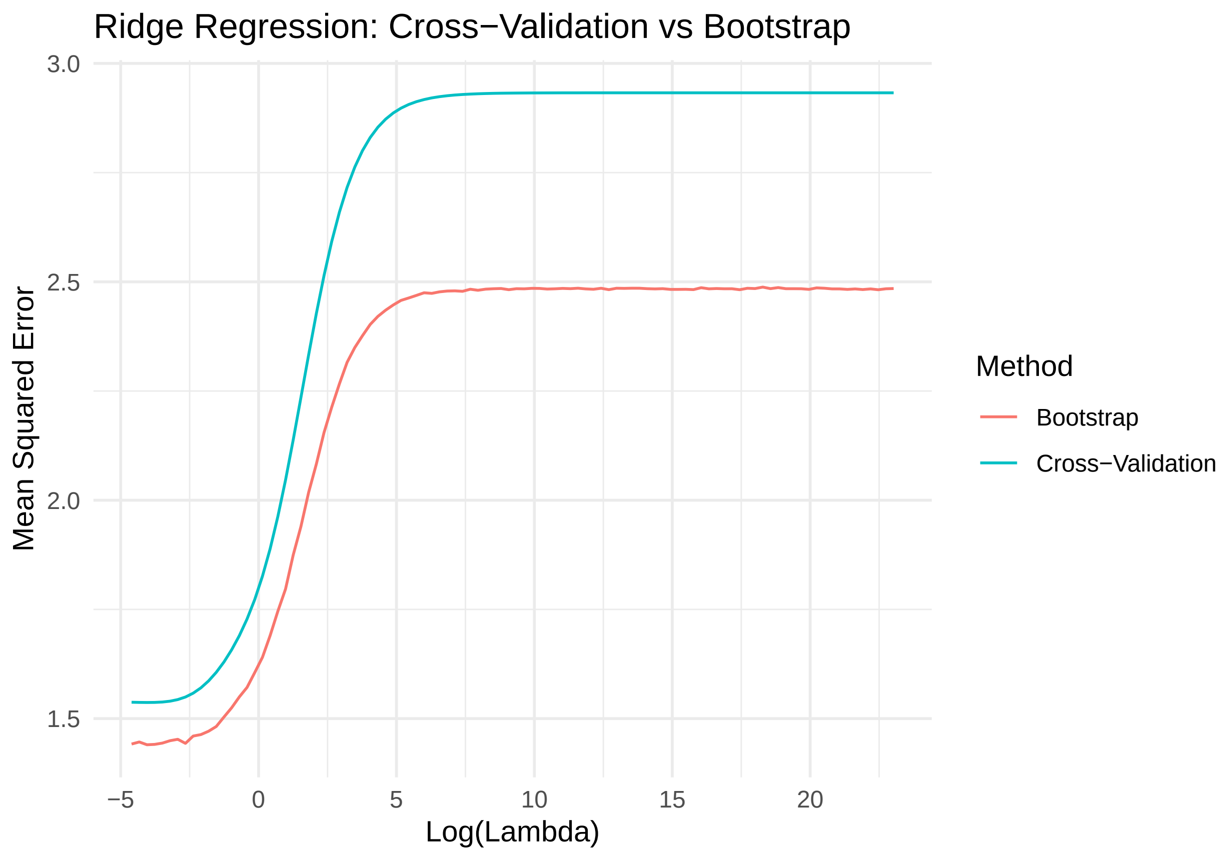
```
## [1] "Best Lambda from Bootstrap: 0.0174752840000768"
```

```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)
print(paste("Test MSE for Ridge Regression (Bootstrap):", mse_bootstrap))
```

```
## [1] "Test MSE for Ridge Regression (Bootstrap): 1.39984196703498"
```

Cross Validation vs Bootstrap Comparision

```r
# 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 E
  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
```

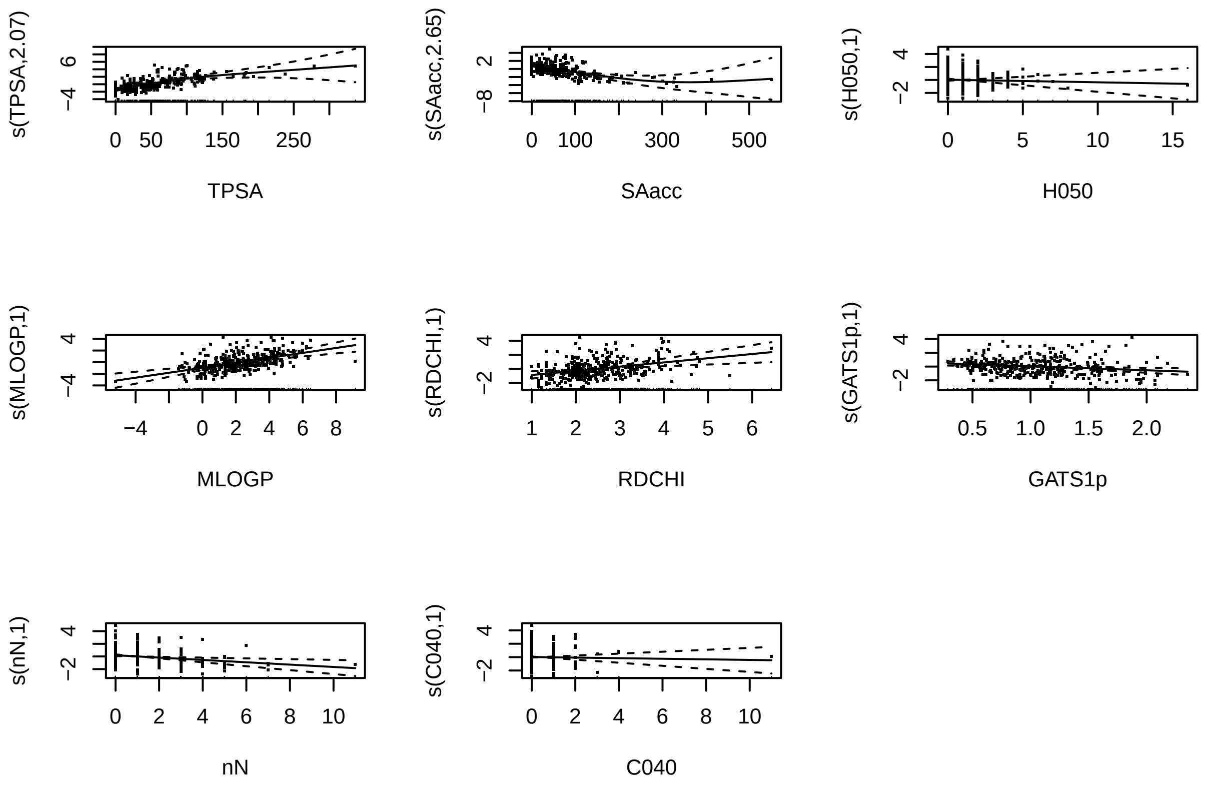
Lower Complexity (k = 4)

```
# 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)
```

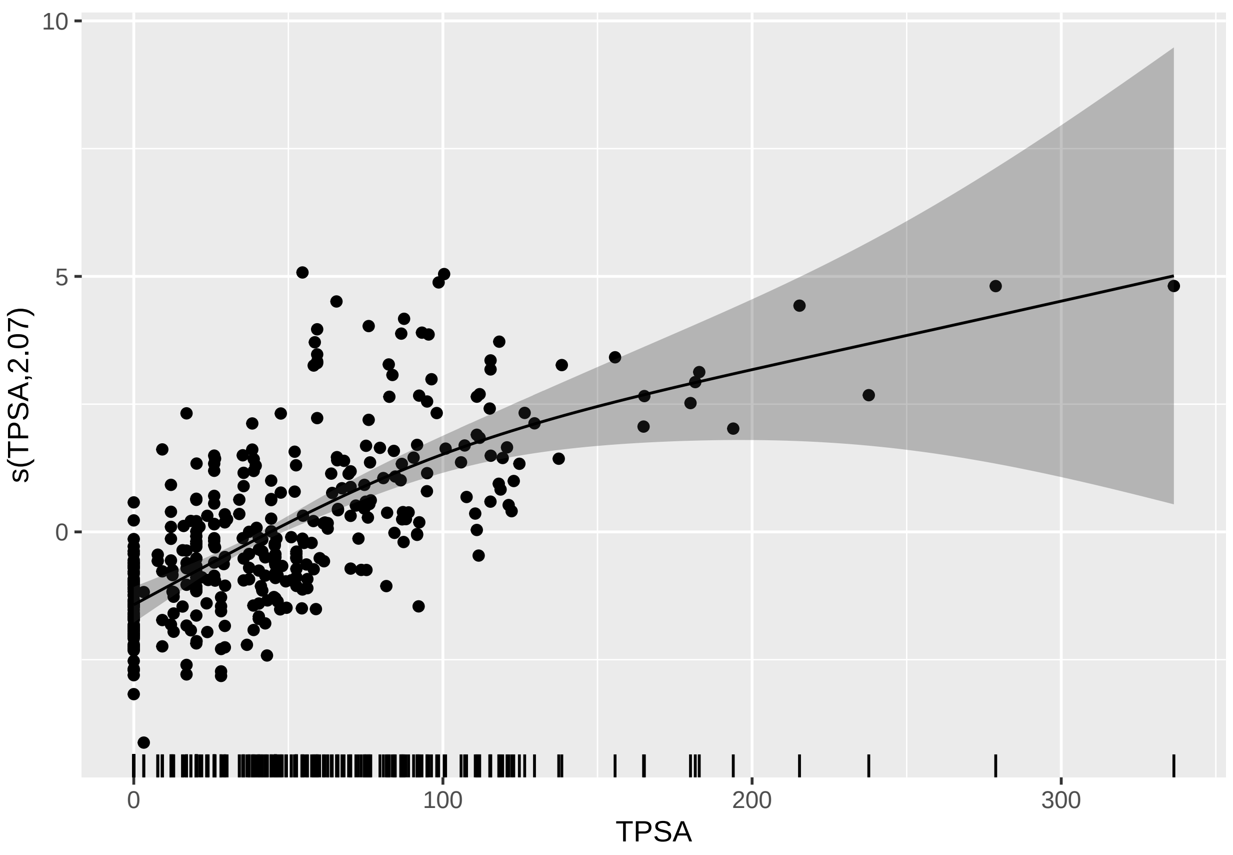
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## LC50 ~ s(TPSA, k = 4) + s(SAacc, k = 4) + s(H050, k = 4) + s(MLOGP,
##     k = 4) + s(RDCHI, k = 4) + s(GATS1p, k = 4) + s(nN, k = 4) +
##     s(C040, k = 4)
##
## Parametric coefficients:
##             Estimate Std. Error t value 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
```

```
# reference: https://stackoverflow.com/questions/67077306/plotting-output-of-gam-model

p_obj <- plot(gam_model_1, residuals = TRUE, pages = 1, scale = 0)
```



```
p_obj <- p_obj[[1]] # just one smooth so select the first component
sm_df <- as.data.frame(p_obj[c("x", "se", "fit")])
data_df <- as.data.frame(p_obj[c("raw", "p.resid")])

## plot
ggplot(sm_df, aes(x = x, y = fit)) +
  geom_rug(data = data_df, mapping = aes(x = raw, y = NULL),
           sides = "b") +
  geom_point(data = data_df, mapping = aes(x = raw, y = p.resid)) +
  geom_ribbon(aes(ymin = fit - se, ymax = fit + se, y = NULL),
              alpha = 0.3) +
  geom_line() +
  labs(x = p_obj$xlab, y = p_obj$ylab)
```



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

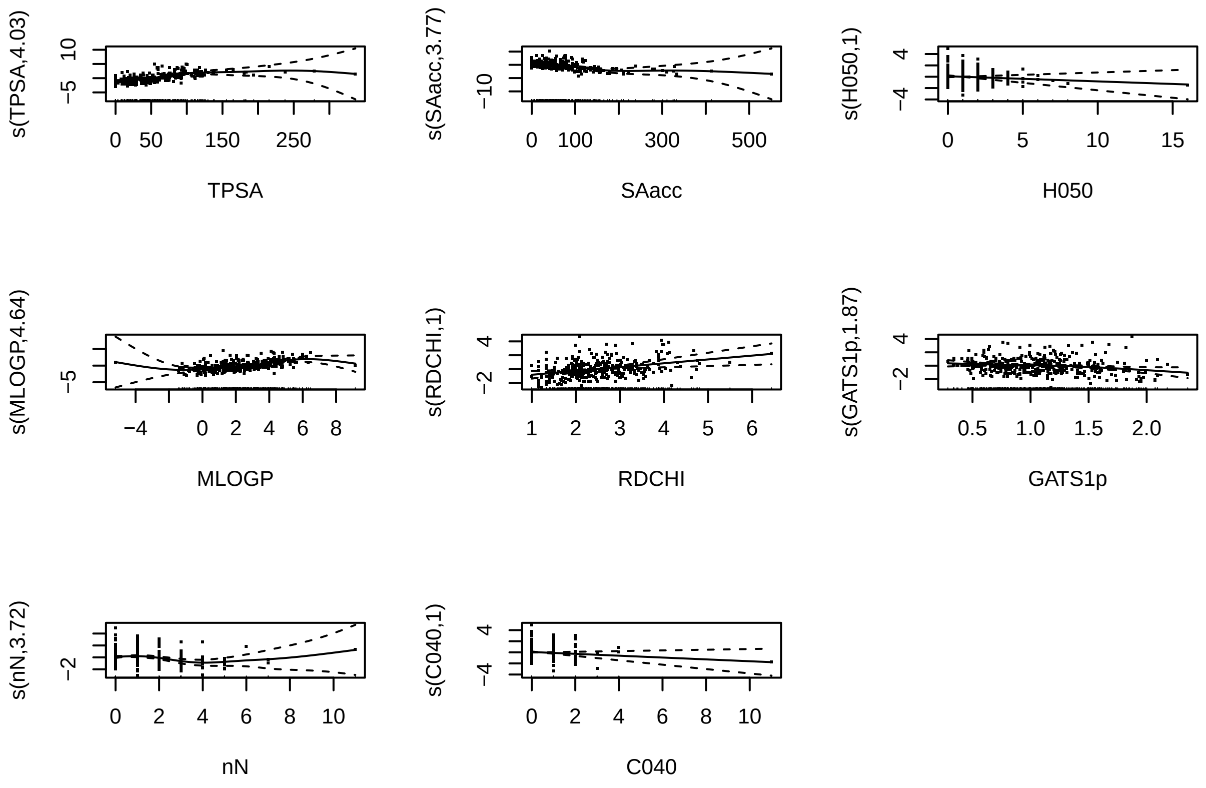
```
## MSE: 1.40582163542459
## RMSE (Test): 1.18567349444296
## R-squared (Test): 0.432144528404967
```

Higher Complexity (k = 6)

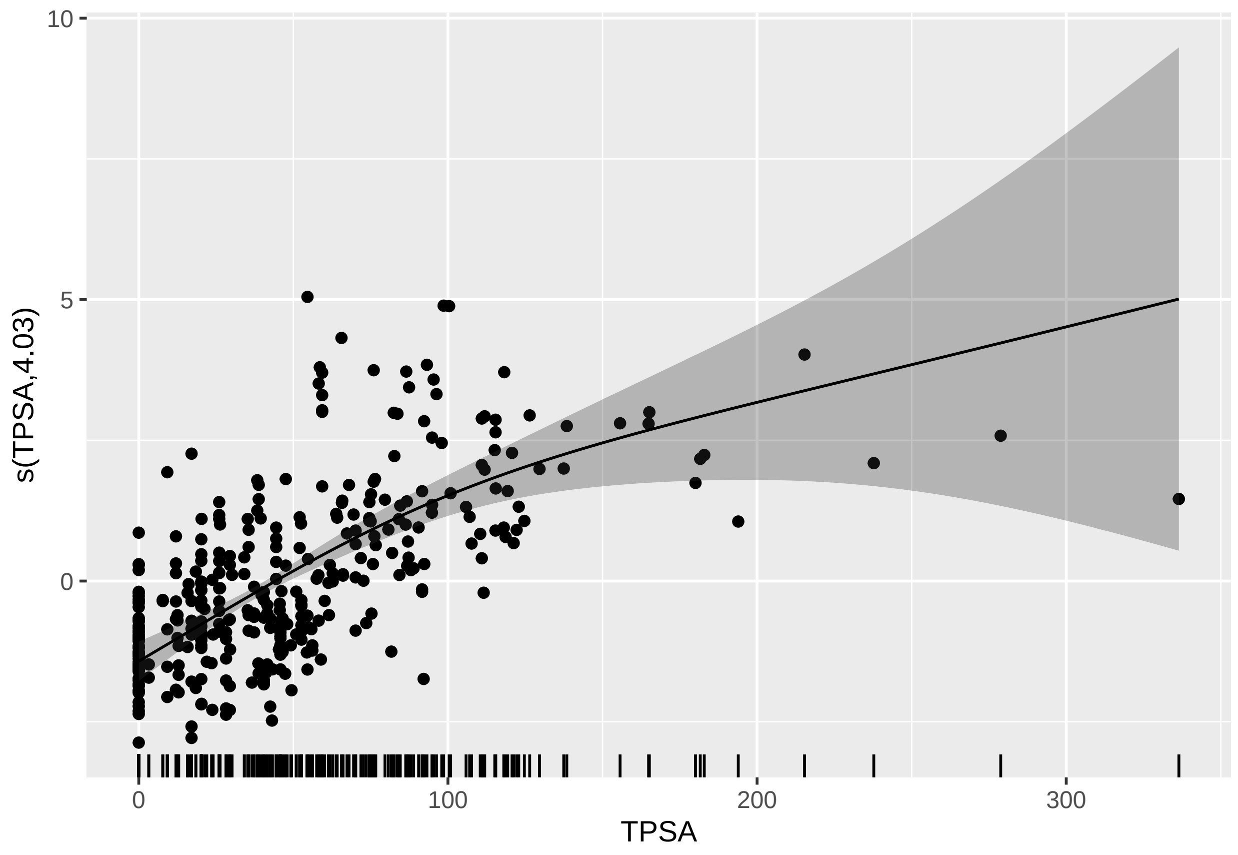
```r
# Fit GAM with smoothing splines (higher complexity)
gam_model_2 <- gam(LC50 ~ s(TPSA, k = 6) + s(SAacc, k = 6) + s(H050, k = 6) +
                       s(MLOGP, k = 6) + s(RDCHI, k = 6) + s(GATS1p, k = 6) +
                       s(nN, k = 6) + s(C040, k = 6), data = train)

# 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
p_obj_2 <- plot(gam_model_2, residuals = TRUE, pages = 1, scale = 0)
```



```r
p_obj_2 <- p_obj_2[[1]] # just one smooth so select the first component
sm_df_2 <- as.data.frame(p_obj_2[c("x", "se", "fit")])
data_df_2 <- as.data.frame(p_obj_2[c("raw", "p.resid")])

## plot
ggplot(sm_df, aes(x = x, y = fit)) +
  geom_rug(data = data_df_2, mapping = aes(x = raw, y = NULL),
           sides = "b") +
  geom_point(data = data_df_2, mapping = aes(x = raw, y = p.resid)) +
  geom_ribbon(aes(ymin = fit - se, ymax = fit + se, y = NULL),
              alpha = 0.3) +
  geom_line() +
  labs(x = p_obj_2$xlab, y = p_obj_2$ylab)
```



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

```
## MSE: 1.35853681502292
## RMSE (Test): 1.16556287476446
## R-squared (Test): 0.451244351105306
```

f. Regression Tree with Cost-Complexity Pruning

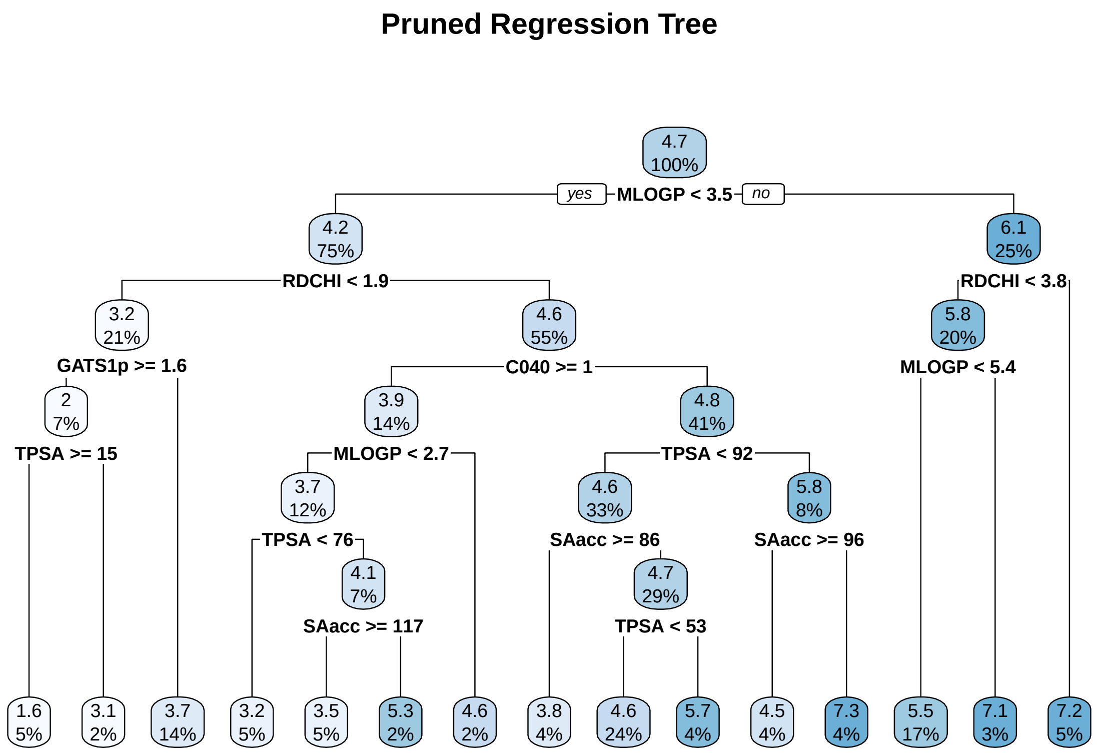
```r
# Fit a regression tree model
tree_model <- rpart(LC50 ~ ., data = train, method = "anova", control = rpart.control(cp = 0.001))
printcp(tree_model)  # Display the cost complexity pruning table
```

```
##
## 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 1.00554 0.083153
## 2  0.1015513      1   0.78014 0.88779 0.077326
## 3  0.0470255      2   0.67859 0.76571 0.069177
## 4  0.0384187      3   0.63156 0.79172 0.079065
## 5  0.0282925      6   0.51631 0.76715 0.080684
## 6  0.0225187      7   0.48801 0.73563 0.077249
## 7  0.0132815      8   0.46550 0.69366 0.074173
## 8  0.0109658     10   0.43893 0.68226 0.068890
## 9  0.0094706     11   0.42797 0.68746 0.068921
## 10 0.0086901     14   0.39955 0.67795 0.068397
## 11 0.0066615     15   0.39086 0.69035 0.069010
## 12 0.0063800     16   0.38420 0.69365 0.069430
## 13 0.0039482     17   0.37782 0.69517 0.070256
## 14 0.0038386     18   0.37387 0.71356 0.070894
## 15 0.0031083     19   0.37004 0.71325 0.070945
## 16 0.0027054     20   0.36693 0.70726 0.070949
## 17 0.0021546     21   0.36422 0.71199 0.071055
## 18 0.0019759     22   0.36207 0.71482 0.071172
## 19 0.0014224     23   0.36009 0.71665 0.071368
## 20 0.0014058     24   0.35867 0.71705 0.071359
## 21 0.0010000     26   0.35586 0.71731 0.071354
```

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

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

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

g. Compare all the models implemented

[TBD]

Problem 2. Classification

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

```r
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
```

```r
# 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
```

```r
# 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
```

```r
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
```

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

```
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:boot':
##
##     melanoma
```

```r
library(FNN)
```

```
## Warning: package 'FNN' was built under R version 4.3.3

##
## Attaching package: 'FNN'

## The following objects are masked from 'package:class':
##
##     knn, knn.cv
```

```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)
}
```

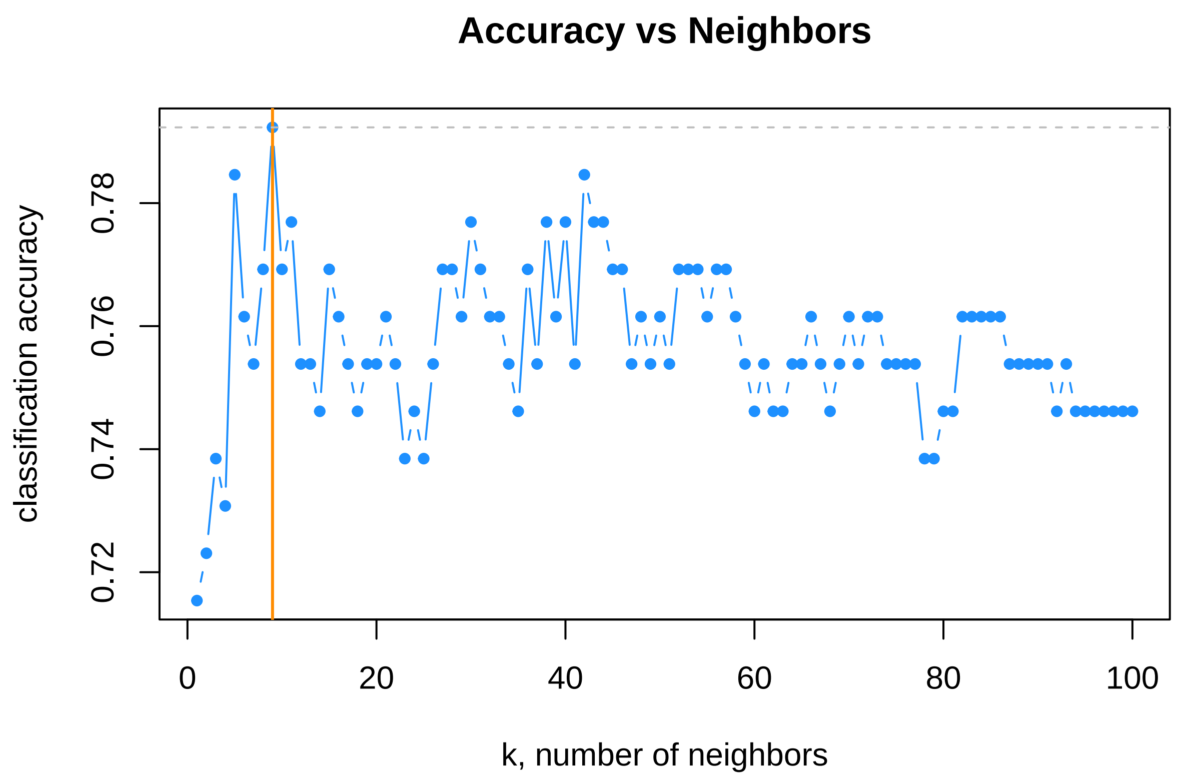
```r
ex_seq = seq(from = 1, to = 100, by = 5)
seq_along(ex_seq)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```r
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
```

```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)
```



```r
max(acc_k)
```

```
## [1] 0.7923077
```

```r
max(which(acc_k == max(acc_k)))
```

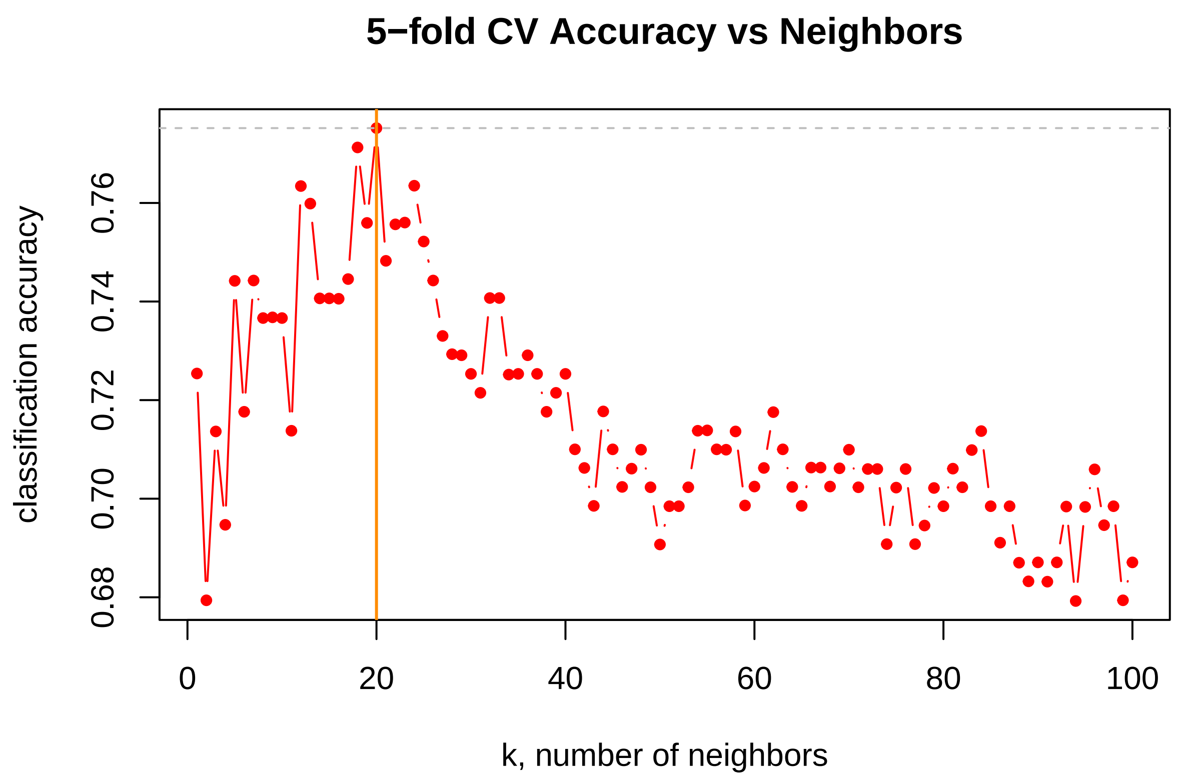
```
## [1] 9
```

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

cv_results
```

```
##      k  Accuracy     Kappa AccuracySD    KappaSD
## 1    1 0.7253991 0.3853807 0.04199871 0.10102890
## 2    2 0.6793904 0.2899952 0.06339646 0.15037051
## 3    3 0.7136430 0.3261247 0.05242032 0.13848651
## 4    4 0.6947025 0.2954201 0.05140217 0.12331509
## 5    5 0.7441945 0.4223798 0.05806649 0.13038485
## 6    6 0.7176343 0.3517014 0.06971643 0.14191727
## 7    7 0.7442671 0.4098717 0.06911187 0.15210421
## 8    8 0.7366473 0.3686896 0.04121555 0.11771477
## 9    9 0.7367925 0.3708664 0.04597802 0.11763412
## 10  10 0.7366473 0.3667508 0.01549733 0.05872508
## 11  11 0.7137881 0.3069454 0.03473601 0.09676656
## 12  12 0.7634253 0.4151053 0.03131835 0.10991421
## 13  13 0.7598694 0.4105388 0.04354015 0.13644072
## 14  14 0.7406386 0.3681274 0.04682131 0.15315245
## 15  15 0.7406386 0.3699683 0.04449355 0.13681483
## 16  16 0.7405660 0.3716355 0.05425469 0.15022554
## 17  17 0.7445573 0.3654783 0.03702528 0.12190268
## 18  18 0.7712627 0.4314293 0.03842609 0.11623144
## 19  19 0.7559507 0.4023795 0.03190620 0.09235988
## 20  20 0.7751814 0.4522215 0.06040716 0.15260923
## 21  21 0.7482583 0.3853247 0.03526171 0.09597292
## 22  22 0.7556604 0.4020951 0.02905546 0.05268034
## 23  23 0.7560232 0.4025624 0.05117950 0.12085406
## 24  24 0.7634978 0.4154198 0.02382290 0.05932417
## 25  25 0.7521771 0.3865091 0.04051856 0.11388513
## 26  26 0.7442671 0.3708857 0.04808438 0.10520664
## 27  27 0.7330189 0.3400789 0.03355715 0.08195350
## 28  28 0.7293179 0.3274359 0.05077428 0.13931525
## 29  29 0.7291001 0.3282595 0.02705274 0.06791059
## 30  30 0.7253266 0.3311189 0.03845812 0.08178613
## 31  31 0.7214804 0.3333429 0.04308894 0.07915469
## 32  32 0.7407112 0.3735370 0.04799202 0.12512801
## 33  33 0.7407112 0.3757549 0.04988132 0.11812590
## 34  34 0.7251814 0.3310971 0.02910413 0.03160359
## 35  35 0.7253266 0.3357028 0.04888516 0.10246613
## 36  36 0.7291001 0.3456244 0.03319147 0.07254548
## 37   37 0.7253266 0.3326820 0.03330472 0.09165372
## 38   38 0.7176343 0.3082879 0.02368017 0.08546077
## 39   39 0.7214804 0.3209288 0.02758646 0.08475561
## 40   40 0.7253266 0.3331346 0.03330472 0.09264985
## 41   41 0.7100145 0.2908339 0.03312575 0.09285197
## 42   42 0.7062409 0.2849974 0.04688748 0.11672756
## 43   43 0.6985486 0.2613262 0.04065432 0.10638433
## 44   44 0.7177068 0.3188721 0.03244469 0.09470211
## 45   45 0.7100145 0.2921299 0.04038952 0.10732635
## 46   46 0.7023948 0.2806634 0.03601259 0.11189724
## 47   47 0.7060958 0.2836002 0.03429122 0.10452421
## 48   48 0.7099419 0.2871439 0.03383670 0.09334582
## 49   49 0.7023222 0.2684573 0.02498443 0.07019557
## 50   50 0.6907112 0.2339495 0.02642621 0.06585375
## 51   51 0.6984761 0.2540067 0.02468944 0.09774892
## 52   52 0.6984761 0.2545123 0.03673164 0.11169192
## 53   53 0.7023222 0.2685101 0.02095974 0.10545927
## 54   54 0.7137881 0.2972683 0.02606730 0.08946744
## 55   55 0.7138607 0.2955564 0.02112554 0.09763439
## 56   56 0.7100145 0.2824568 0.01891218 0.10452692
## 57   57 0.7099419 0.2950449 0.03665594 0.09263199
## 58   58 0.7136430 0.3059461 0.03676335 0.09250440
## 59   59 0.6986212 0.2687081 0.04815839 0.12983469
## 60   60 0.7024673 0.2813490 0.05198376 0.13411870
## 61   61 0.7062409 0.2882363 0.04688748 0.12221321
## 62   62 0.7175617 0.3138726 0.03129796 0.08627802
## 63   63 0.7100145 0.2953418 0.04285978 0.11083655
## 64   64 0.7023948 0.2734939 0.03867353 0.09898091
## 65   65 0.6985486 0.2658145 0.03334462 0.09364268
## 66   66 0.7063135 0.2870065 0.05195500 0.15030023
## 67   67 0.7063135 0.2905570 0.04212797 0.13482275
## 68   68 0.7024673 0.2808363 0.04461293 0.12588210
## 69   69 0.7061684 0.2888208 0.02467397 0.08059471
## 70   70 0.7099419 0.3012442 0.02457667 0.06644135
## 71   71 0.7023222 0.2912501 0.03163789 0.08509673
## 72   72 0.7060232 0.3041397 0.03002241 0.05774592
## 73   73 0.7060232 0.3081720 0.03002241 0.05836819
## 74   74 0.6907837 0.2671480 0.03210858 0.07251566
## 75   75 0.7022496 0.3007584 0.02948166 0.06775997
## 76   76 0.7060232 0.3036828 0.04264961 0.09628330
## 77   77 0.6907837 0.2718546 0.03210858 0.06955760
## 78   78 0.6945573 0.2713882 0.03908075 0.07287679
## 79   79 0.7021771 0.2888170 0.02715451 0.04504920
## 80   80 0.6984761 0.2807089 0.03129522 0.07830808
## 81   81 0.7060958 0.3082247 0.03213174 0.06841935
## 82   82 0.7023222 0.2866907 0.03433592 0.09311945
## 83   83 0.7098694 0.2976172 0.04171736 0.09859163
## 84   84 0.7137155 0.3153341 0.04883733 0.10277407
## 85   85 0.6984761 0.2674806 0.03129522 0.07683945
## 86   86 0.6910740 0.2420152 0.04751437 0.13697009
## 87   87 0.6984761 0.2591730 0.03381666 0.08071863
## 88   88 0.6870102 0.2236917 0.02888390 0.06837402
## 89   89 0.6832366 0.2173979 0.03907115 0.08676745
## 90   90 0.6870827 0.2292425 0.05429144 0.12726271
## 91    91 0.6831640 0.2125598 0.03996187 0.08545398
## 92    92 0.6870827 0.2050847 0.03112607 0.11355339
## 93    93 0.6984035 0.2396666 0.02922015 0.07125958
## 94    94 0.6792453 0.1948207 0.04497226 0.09625943
## 95    95 0.6983309 0.2417438 0.04489652 0.08503586
## 96    96 0.7059507 0.2647771 0.04108373 0.08135582
## 97    97 0.6946299 0.2168359 0.04091836 0.10605373
## 98    98 0.6984761 0.2302913 0.03422301 0.07781011
## 99    99 0.6793904 0.1718094 0.03673039 0.10534485
## 100 100 0.6870827 0.1822333 0.03903288 0.09966853
```

```
# 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")
#legend("bottomright", legend = "5-fold CV Accuracy", col = "red", pch = 19, lty = 1)

# 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)
```



```
max(cv_results$Accuracy)
```

```
## [1] 0.7751814
```

```r
cv_results$k[which.max(cv_results$Accuracy)]
```

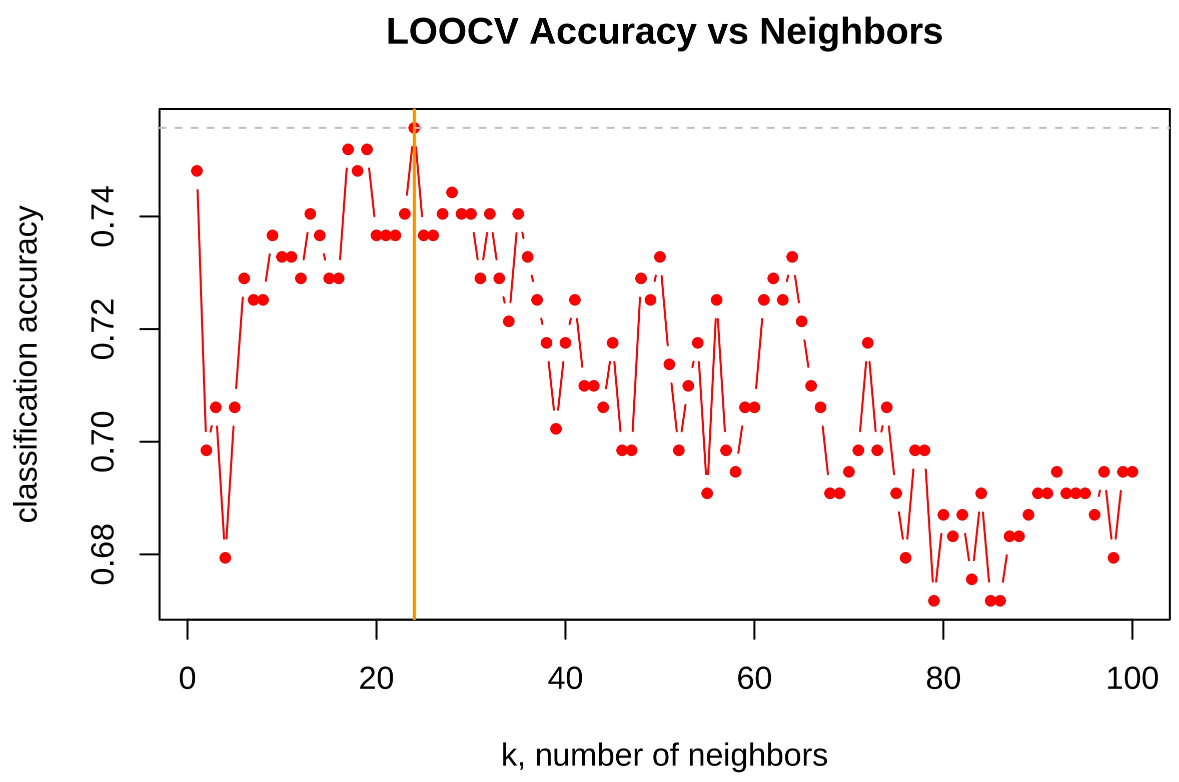
```
## [1] 20
```

Using leave-one-out cross-validation

```r
# 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

cv_results_loocv
```

```
##       k  Accuracy     Kappa
## 1     1 0.7480916 0.4353817
## 2     2 0.6984733 0.3337411
## 3     3 0.7061069 0.3197788
## 4     4 0.6793893 0.2644877
## 5     5 0.7061069 0.3157181
## 6     6 0.7290076 0.3764832
## 7     7 0.7251908 0.3620561
## 8     8 0.7251908 0.3658307
## 9     9 0.7366412 0.3831297
## 10   10 0.7328244 0.3685008
## 11   11 0.7328244 0.3646505
## 12   12 0.7290076 0.3496259
## 13   13 0.7404580 0.3751841
## 14   14 0.7366412 0.3640329
## 15   15 0.7290076 0.3415221
## 16   16 0.7290076 0.3415221
## 17   17 0.7519084 0.3933889
## 18   18 0.7480916 0.3859811
## 19   19 0.7519084 0.3856865
## 20   20 0.7366412 0.3519966
## 21   21 0.7366412 0.3478825
## 22   22 0.7366412 0.3437160
## 23   23 0.7404580 0.3552870
## 24   24 0.7557252 0.3893219
## 25   25 0.7366412 0.3394958
## 26   26 0.7366412 0.3394958
## 27   27 0.7404580 0.3384330
## 28   28 0.7442748 0.3586408
## 29   29 0.7404580 0.3427285
## 30   30 0.7404580 0.3427285
## 31   31 0.7290076 0.3159520
## 32   32 0.7404580 0.3511545
## 33   33 0.7290076 0.3159520
## 34   34 0.7213740 0.2966831
## 35   35 0.7404580 0.3384330
## 36   36 0.7328244 0.3277619
## 37   37 0.7251908 0.3085551
## 38   38 0.7175573 0.3028407
## 39   39 0.7022901 0.2651564
## 40   40 0.7175573 0.3028407
## 41   41 0.7251908 0.3173627
## 42   42 0.7099237 0.2701415
## 43   43 0.7099237 0.2557375
## 44   44 0.7061069 0.2533126
## 45   45 0.7175573 0.2847340
## 46   46 0.6984733 0.2437706
## 47   47 0.6984733 0.2437706
## 48   48 0.7290076 0.3289806
## 49   49 0.7251908 0.3301612
## 50   50 0.7328244 0.3446723
## 51   51 0.7137405 0.3000641
## 52   52 0.6984733 0.2580830
## 53   53 0.7099237 0.2973393
## 54   54 0.7175573 0.3200533
## 55   55 0.6908397 0.2393003
## 56   56 0.7251908 0.3085551
## 57   57 0.6984733 0.2533728
## 58   58 0.6946565 0.2366523
## 59   59 0.7061069 0.2629156
## 60   60 0.7061069 0.2629156
## 61   61 0.7251908 0.3040655
## 62   62 0.7290076 0.3159520
## 63   63 0.7251908 0.3040655
## 64   64 0.7328244 0.3277619
## 65   65 0.7213740 0.3012057
## 66   66 0.7099237 0.2794384
## 67   67 0.7061069 0.2722747
## 68   68 0.6908397 0.2295796
## 69   69 0.6908397 0.2295796
## 70   70 0.6946565 0.2366523
## 71   71 0.6984733 0.2627342
## 72   72 0.7175573 0.3115546
## 73   73 0.6984733 0.2673274
## 74   74 0.7061069 0.2989784
## 75   75 0.6908397 0.2625617
## 76   76 0.6793893 0.2329035
## 77   77 0.6984733 0.2894610
## 78   78 0.6984733 0.2807700
## 79   79 0.6717557 0.2194277
## 80   80 0.6870229 0.2557334
## 81   81 0.6832061 0.2489466
## 82   82 0.6870229 0.2734528
## 83   83 0.6755725 0.2308489
## 84   84 0.6908397 0.2801710
## 85   85 0.6717557 0.2288315
## 86   86 0.6717557 0.2288315
## 87   87 0.6832061 0.2579677
## 88   88 0.6832061 0.2623974
## 89   89 0.6870229 0.2691025
## 90   90 0.6908397 0.2844426
## 91   91 0.6908397 0.2844426
## 92   92 0.6946565 0.2953674
## 93   93 0.6908397 0.2844426
## 94   94 0.6908397 0.2801710
## 95   95 0.6908397 0.2801710
## 96   96 0.6870229 0.2646998
## 97   97 0.6946565 0.2826340
## 98   98 0.6793893 0.2467657
## 99   99 0.6946565 0.2826340
## 100 100 0.6946565 0.2782866
```

```
# 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)
```



```
max(cv_results_loocv$Accuracy)
```

```
## [1] 0.7557252
```

```
cv_results$k[which.max(cv_results_loocv$Accuracy)]
```

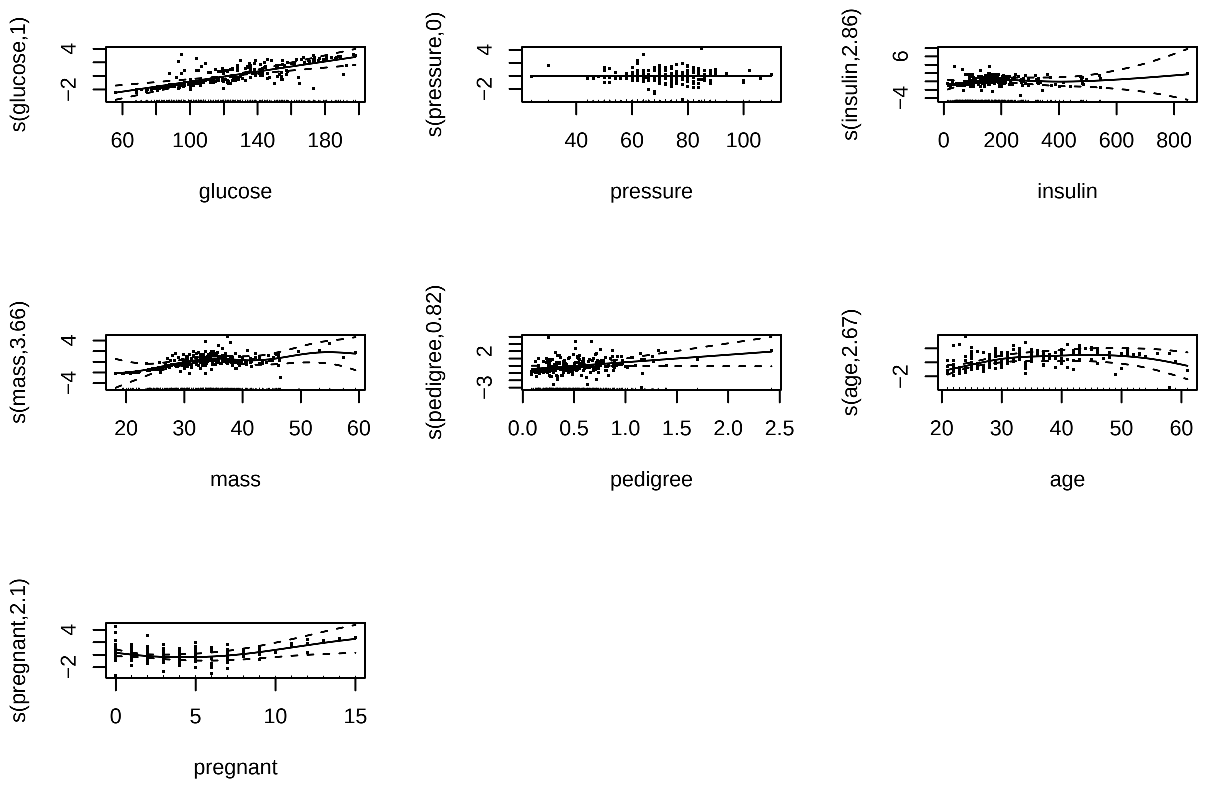
```
## [1] 24
```

b. Generalised Additive Model (GAM)

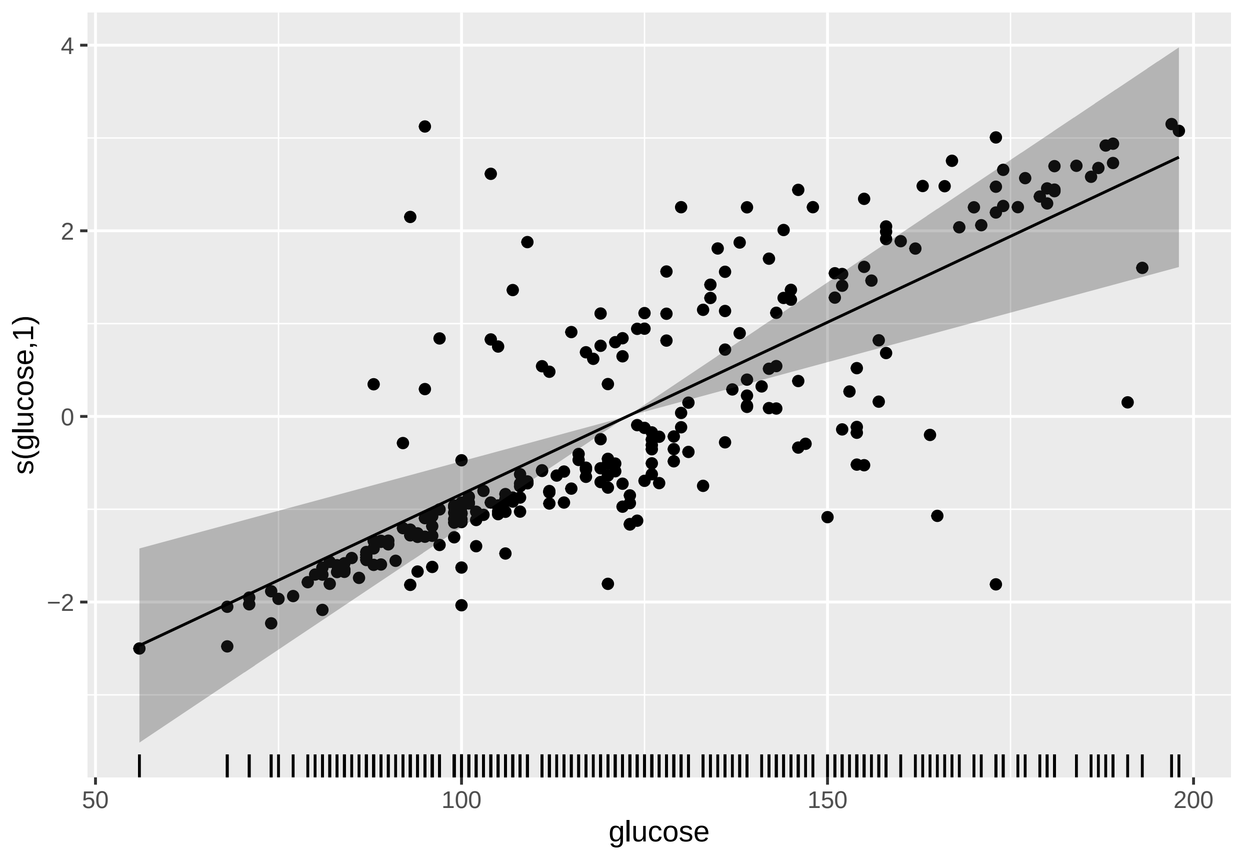
```
# Fit a GAM with automatic smoothness selection
gam_model <- gam(
      diabetes ~ s(glucose) + s(pressure) + s(insulin) + s(mass) + s(pedigree) + s(age) + s(pregnant),
      data = train,
      family = binomial(link = 'logit'),
      select = TRUE)

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)  -1.2771     0.2191  -5.828 5.61e-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)  9.998e-01      9 22.282 1.59e-06 ***
## s(pressure) 9.611e-07      9  0.000  0.85638
## s(insulin)  2.863e+00      9  3.283  0.31288
## s(mass)     3.662e+00      9 11.415  0.00922 **
## s(pedigree) 8.237e-01      9  3.732  0.03187 *
## s(age)      2.670e+00      9 11.742  0.00254 **
## s(pregnant) 2.103e+00      9  6.358  0.02744 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.445   Deviance explained = 42.8%
## UBRE = -0.16466  Scale est. = 1         n = 262
```

```r
p_obj <- plot(gam_model, residuals = TRUE, pages = 1, scale = 0)
```



```r
p_obj <- p_obj[[1]] # just one smooth so select the first component
sm_df <- as.data.frame(p_obj[c("x", "se", "fit")])
data_df <- as.data.frame(p_obj[c("raw", "p.resid")])

## plot
ggplot(sm_df, aes(x = x, y = fit)) +
  geom_rug(data = data_df, mapping = aes(x = raw, y = NULL),
           sides = "b") +
  geom_point(data = data_df, mapping = aes(x = raw, y = p.resid)) +
  geom_ribbon(aes(ymin = fit - se, ymax = fit + se, y = NULL),
              alpha = 0.3) +
  geom_line() +
  labs(x = p_obj$xlab, y = p_obj$ylab)
```



[TBD] add variable selection step

c. Tree-based methods

Setting up the k-fold cross validation k = 10 cross-validation folds. Reference: https://quantdev.ssri.psu.edu/sites/qdev/files/09_EnsembleMethods_2017_1127.html

```
library(rpart)  #for trees
library(rpart.plot)             # Enhanced tree plots
library(RColorBrewer)       # Color selection for fancy tree plot
library(party)                  # Alternative decision tree algorithm
```

```
## Warning: package 'party' was built under R version 4.3.3

## Loading required package: grid

## Loading required package: mvtnorm

## Warning: package 'mvtnorm' was built under R version 4.3.3

## Loading required package: modeltools

## Loading required package: stats4
## Loading required package: strucchange

## Warning: package 'strucchange' was built under R version 4.3.3

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

## Warning: package 'sandwich' was built under R version 4.3.3
```

```r
library(partykit)                # Convert rpart object to BinaryTree
```

```
## Warning: package 'partykit' was built under R version 4.3.3

## Loading required package: libcoin

##
## Attaching package: 'partykit'

## The following objects are masked from 'package:party':
##
##     cforest, ctree, ctree_control, edge_simple, mob, mob_control,
##     node_barplot, node_bivplot, node_boxplot, node_inner, node_surv,
##     node_terminal, varimp
```

```r
library(pROC)    #for ROC curves
```

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

##
## Attaching package: 'pROC'

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

```r
set.seed(1234)

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

(i) Classification tree

[TBD]

```
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.
```

(ii) Ensemble of bagged trees

[TBD]

(iii) Random Forest

[TBD]