Stat 432 HW 06

Name: Ahmadreza Eslaminia, netID: ae15

Summer 2024

Include the R code for this HW.

```r
knitr::opts_chunk$set(echo = TRUE)
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.3.3
```

```r
library(GGally)
```

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

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```r
library(tibble)
```

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

```r
library(dplyr)
```

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

```r
library(knitr)
```

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

```r
library(kableExtra)
```

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

```r
library(caret)
```

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

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.3.3
```

```r
library(gam)
```

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

```r
library(splines)
library(boot)
library(mgcv)

#add more libraries as needed.
```

Question 1 (SVM)

You're given 9 observations in $p = 2$ dimensions. For each observations, there is an associated class label (y).

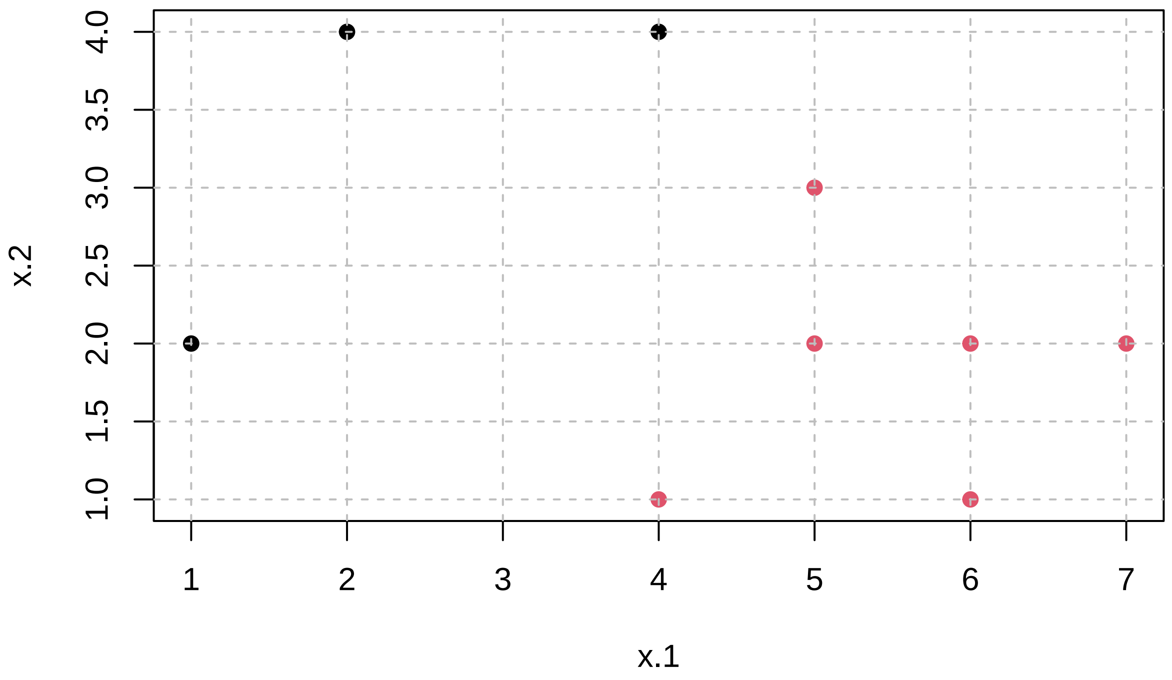
```r
x.1=c(1,2,4,4,5,5,6,6,7)
x.2=c(2,4,4,1,3,2,1,2,2)
y=factor(c(rep(1,3),rep(2,6)))
my.data=data.frame(x.1,x.2,y)
print(my.data)
```

```
##   x.1 x.2 y
## 1   1   2 1
## 2   2   4 1
## 3   4   4 1
## 4   4   1 2
## 5   5   3 2
## 6   5   2 2
## 7   6   1 2
## 8   6   2 2
## 9   7   2 2
```

```r
attach(my.data)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     x.1, x.2, y
```

```r
plot(x.2~x.1,col=y,pch=19,asp=1)
grid(nx = NULL, ny = NULL,
     lty = 2,      # Grid line type
     col = "gray", # Grid line color
     lwd = 1)
```



Answer following questions without using svm algorithm function in R.

(a) Find the optimal separating hyperplane define by the equation $-1 + X_1 + \beta_2 X_2 = 0$. Find β_2.

To find β_2, we use the support vectors (5, 3) and (4, 4) which seem to be the nearest to the line of separation. (candidate support vectors)

For the point $(5, 3)$:

$$-1 + 5 + \beta_2 \cdot 3 = 1 \implies 4 + 3\beta_2 = 1 \implies 3\beta_2 = -3 \implies \beta_2 = -1$$

For the point $(4, 4)$:

$$-1 + 4 + \beta_2 \cdot 4 = -1 \implies 3 + 4\beta_2 = -1 \implies 4\beta_2 = -4 \implies \beta_2 = -1$$

Therefore, the optimal β_2 is -1.

(b) Find all support vectors.

The support vectors are:

$$(5, 3) \quad \text{and} \quad (4, 4)$$

(c) If we add a new observation (x.1=1, x.2=4, y=1), would this affect the maximal margin classifier?

Adding the new observation would not affect the maximal margin classifier, as this point lies further from the bondary within the class boundries of $y = 1$.

Now, use svm algorithm in R to answer the following question. *Use the option `scale=FALSE` for this question.*

(d) If we add a new observation (x.1=1, x.2=4, y=1), would this affect the maximal margin classifier? Add the observation to your dataset and see if it makes meaningful change.

```r
svm.model <- svm(y ~ ., data = my.data, scale = FALSE, kernel = "linear")
summary(svm.model)
```

```
##
## Call:
## svm(formula = y ~ ., data = my.data, kernel = "linear", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  2
##
##  ( 1 1 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

```r
svm.model$SV
```

```
##   x.1 x.2
## 3   4   4
## 5   5   3
```

```r
new.data <- rbind(my.data, data.frame(x.1 = 1, x.2 = 4, y = factor(1, levels = levels(y))))

# added new data
svm.model.new <- svm(y ~ ., data = new.data, scale = FALSE, kernel = "linear")
summary(svm.model.new)
```

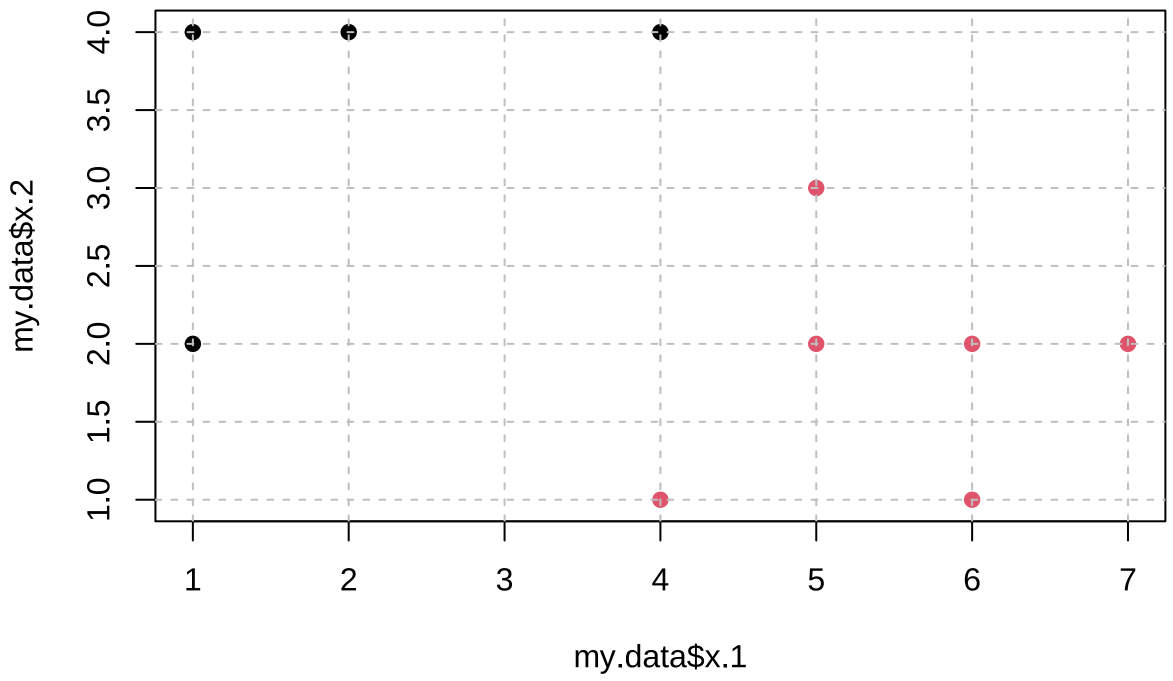
```
##
## Call:
## svm(formula = y ~ ., data = new.data, kernel = "linear", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  2
##
##  ( 1 1 )
##
##
## Number of Classes:  2
##
```

```
## Levels:
##  1 2
```

```
svm.model.new$SV
```

```
##   x.1 x.2
## 3   4   4
## 5   5   3
```

```
plot(my.data$x.1, my.data$x.2, col = my.data$y, pch = 19, asp = 1)
points(1, 4, col = "black", pch = 19)
grid(nx = NULL, ny = NULL, lty = 2, col = "gray", lwd = 1)
```



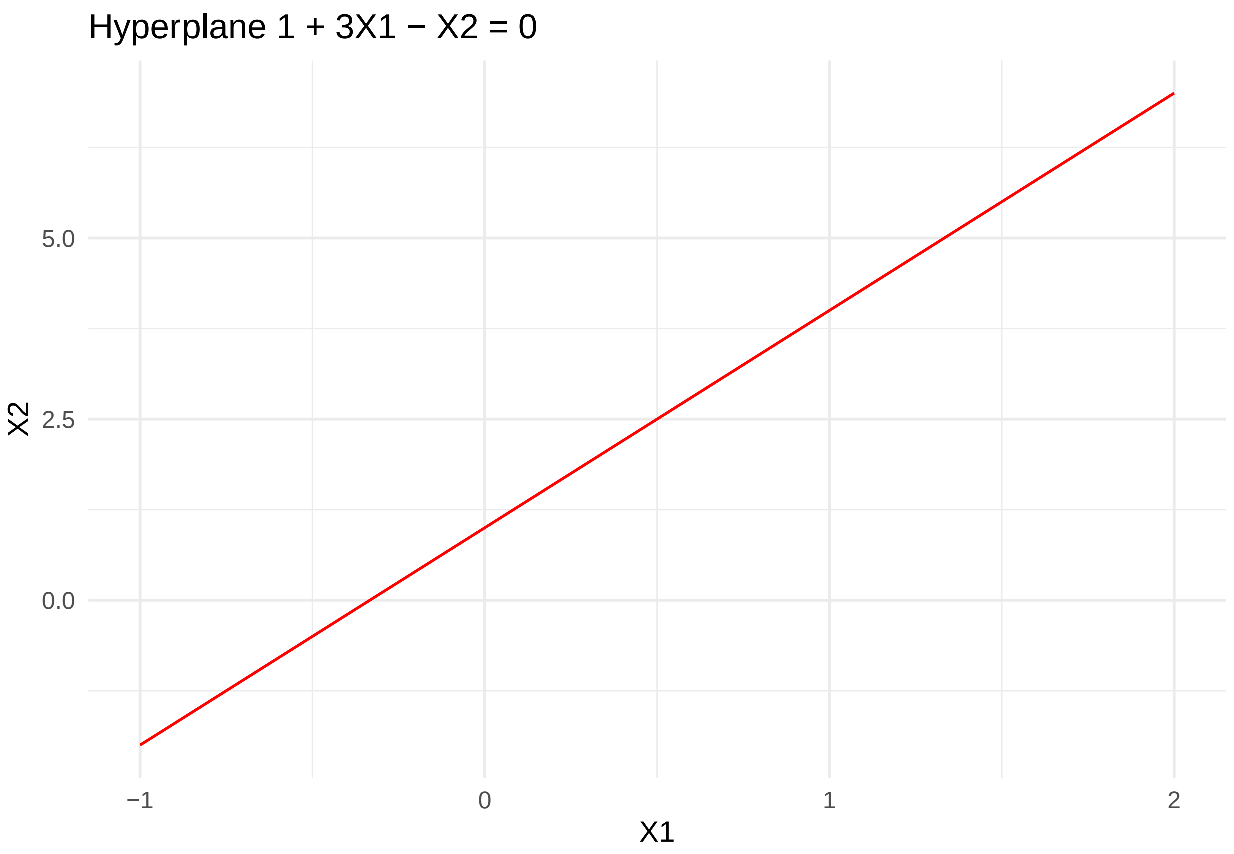
As it can be seen in the support vectors, before and after adding the new number, the addition do not have any effect on the SVM boundaries since the support vectors that are the nearest points to the decision line is important and as it is obvious int he plot, the added point do not change the nearest point.

Question 2 (SVM)

(a) Sketch the hyperplane $1 + 3X_1 - X_2 = 0$.

```
# Define the hyperplane function
hyperplane <- function(x1) {
  return(1 + 3 * x1)
}

# Generate a sequence of x1 values
x1_seq <- seq(-1, 2, length.out = 100)
x2_seq <- hyperplane(x1_seq)
```

```r
# Plot the hyperplane
ggplot() +
  geom_line(aes(x = x1_seq, y = x2_seq), color = 'red') +
  labs(title = "Hyperplane 1 + 3X1 - X2 = 0",
       x = "X1",
       y = "X2") +
  theme_minimal()
```



(b) For the given observations

```r
set.seed(4)
x1=sample(1:10,5)
x2=sample(5:15,5)
print(data.frame(x1,x2))
```

```
##   x1 x2
## 1  8  7
## 2  3 10
## 3  9  9
## 4  7  6
## 5  4 15
```

indicate the set of points for which $1+3X_1-X_2>0$, as well as the set of points for which $1+3X_1-X_2<0$.

Evaluate the function $1+3X_1-X_2$ for each point:

1. For the point $(8,7)$:
$$1 + 3(8) - 7 = 18 \implies 18 > 0$$
2. For the point $(3,10)$:
$$1 + 3(3) - 10 = 0 \implies 0 = 0$$
(This point is exactly on the hyperplane)
3. For the point $(9,9)$:
$$1 + 3(9) - 9 = 19 \implies 19 > 0$$
4. For the point $(7,6)$:
$$1 + 3(7) - 6 = 16 \implies 16 > 0$$
5. For the point $(4,15)$:
$$1 + 3(4) - 15 = -2 \implies -2 < 0$$

Question 3 (SVM)

We will use data found in (wisc-trn.csv) and (wisc-tst.csv) [check box folder for files] which contain train and test data respectively. 'This is a modification of the Breast Cancer Wisconsin (Diagnostic) dataset from the UCI Machine Learning Repository. Only the first 10 feature variables have been provided. (And these are all you should use.)

You should consider coercing the response (`class` variable) to be a factor variable.

(a) Fit a support vector classifier to the training data using cost = 0.01, with `class` as the response and the other variables as predictors. Use the `summary()` function to produce summary statistics, and describe the results obtained. Report training and test error rates.

```r
train_data <- read.csv('wisc-trn.csv')
test_data <- read.csv('wisc-tst.csv')

# factorizing
train_data$class <- as.factor(train_data$class)
test_data$class <- as.factor(test_data$class)


svm_model <- svm(class ~ ., data = train_data, cost = 0.01, scale = FALSE)

summary(svm_model)
```

```
##
## Call:
## svm(formula = class ~ ., data = train_data, cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  440
##
```

```
##  ( 172 268 )
##
##
## Number of Classes:  2
##
## Levels:
##  B M
```

```
train_pred <- predict(svm_model, train_data)


train_error <- mean(train_pred != train_data$class)
print(paste("Training Error Rate:", train_error))
```

```
## [1] "Training Error Rate: 0.366737739872068"
```

```
test_pred <- predict(svm_model, test_data)

test_error <- mean(test_pred != test_data$class)
print(paste("Test Error Rate:", test_error))
```

```
## [1] "Test Error Rate: 0.4"
```

(b) Use the `tune()` function to select an optimal `cost`. Consider values in range 0.01 to 10. If necessary, you can change the cost grid. Compute training and test error rates using this new value for `cost`.

```
# Tuneing
tune_result <- tune(svm, class ~ ., data = train_data, ranges = list(cost = seq(0.01, 10, by = 0.1)), s


best_model <- tune_result$best.model


summary(best_model)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = class ~ ., data = train_data, ranges = list(cost = seq(0.01,
##     10, by = 0.1)), scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1.91
##
## Number of Support Vectors:  440
##
##  ( 171 269 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  B M
```

```
train_pred_best <- predict(best_model, train_data)

train_error_best <- mean(train_pred_best != train_data$class)
print(paste("Training Error Rate (Best Model):", train_error_best))
```

```
## [1] "Training Error Rate (Best Model): 0.00426439232409382"
```

```
# test predict
test_pred_best <- predict(best_model, test_data)

test_error_best <- mean(test_pred_best != test_data$class)
print(paste("Test Error Rate (Best Model):", test_error_best))
```

```
## [1] "Test Error Rate (Best Model): 0.28"
```

(c) Repeat (b) using SVM with a radial kernel. Use default value for `gamma`.

```
# Tuneing
tune_result_radial <- tune(svm, class ~ ., data = train_data, kernel = "radial", ranges = list(cost = s

best_model_radial <- tune_result_radial$best.model

summary(best_model_radial)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = class ~ ., data = train_data, ranges = list(cost = seq(0.01,
##     10, by = 0.1)), kernel = "radial", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  9.81
##
## Number of Support Vectors:  437
##
##  ( 171 266 )
##
##
## Number of Classes:  2
##
## Levels:
##  B M
```

```r
train_pred_radial <- predict(best_model_radial, train_data)

train_error_radial <- mean(train_pred_radial != train_data$class)
print(paste("Training Error rate (Raddial Kernel):", train_error_radial))
```

```
## [1] "Training Error rate (Raddial Kernel): 0"
```

```r
test_pred_radial <- predict(best_model_radial, test_data)

test_error_radial <- mean(test_pred_radial != test_data$class)
print(paste("Test Error rate (Raddial Kernel):", test_error_radial))
```

```
## [1] "Test Error rate (Raddial Kernel): 0.27"
```

(d) Repeat (b) using SVM with a polynomial kernel. Set `degree=2`. Hint: you can use `tune` function with `kernel='polynomial'` option. Students can also use other functions/options.

```r
cost_range <- seq(0.01, 10, by = 0.1)

# Tune the SVM model with a polynomial kernel (degree=2)
set.seed(123)
tune_result_poly <- tune(svm, class ~ ., data = train_data,
                         kernel = "polynomial", degree = 2,
                         ranges = list(cost = cost_range))

# Print the best model
best_model_poly <- tune_result_poly$best.model
summary(best_model_poly)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = class ~ ., data = train_data, ranges = list(cost = cost_range),
##     kernel = "polynomial", degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  9.51
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  227
##
##  ( 111 116 )
##
##
## Number of Classes:  2
##
## Levels:
##  B M
```

```r
# Evaluate the model on the training data
train_predictions <- predict(best_model_poly, train_data)
train_conf_matrix <- confusionMatrix(train_predictions, train_data$class)
train_conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   B   M
##          B 295  67
##          M   2 105
##
##                Accuracy : 0.8529
##                  95% CI : (0.8175, 0.8837)
##     No Information Rate : 0.6333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6559
##
##  Mcnemar's Test P-Value : 1.312e-14
##
##             Sensitivity : 0.9933
##             Specificity : 0.6105
##          Pos Pred Value : 0.8149
##          Neg Pred Value : 0.9813
##              Prevalence : 0.6333
##          Detection Rate : 0.6290
##    Detection Prevalence : 0.7719
##       Balanced Accuracy : 0.8019
##
##        'Positive' Class : B
##
```

```r
# Evaluate the model on the test data
test_predictions <- predict(best_model_poly, test_data)
test_conf_matrix <- confusionMatrix(test_predictions, test_data$class)
test_conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 57 24
##          M  3 16
##
##                Accuracy : 0.73
##                  95% CI : (0.632, 0.8139)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.0046004
##
##                   Kappa : 0.3836
##
##  Mcnemar's Test P-Value : 0.0001186
```

```
##
##             Sensitivity : 0.9500
##             Specificity : 0.4000
##          Pos Pred Value : 0.7037
##          Neg Pred Value : 0.8421
##              Prevalence : 0.6000
##          Detection Rate : 0.5700
##    Detection Prevalence : 0.8100
##       Balanced Accuracy : 0.6750
##
##        'Positive' Class : B
##
```

```r
# Print the error rates
train_error_rate <- 1 - train_conf_matrix$overall["Accuracy"]
test_error_rate <- 1 - test_conf_matrix$overall["Accuracy"]

train_error_rate
```

```
##  Accuracy
## 0.1471215
```

```r
test_error_rate
```

```
## Accuracy
##     0.27
```

(e) Based on your analysis on (a) through (d) which approach seems to give the best results?

According to the previous results, the SVM with radial kernel and C= 3.01 and Test error of 0.27 and the SVM with poly degree of two and also C=9.31 has the same 0.27 erorr rate so both are the best models.

Question 4 (GAM: Poly)

In this question, we will conduct regression using `Boston` data from the `ISLR2` package.

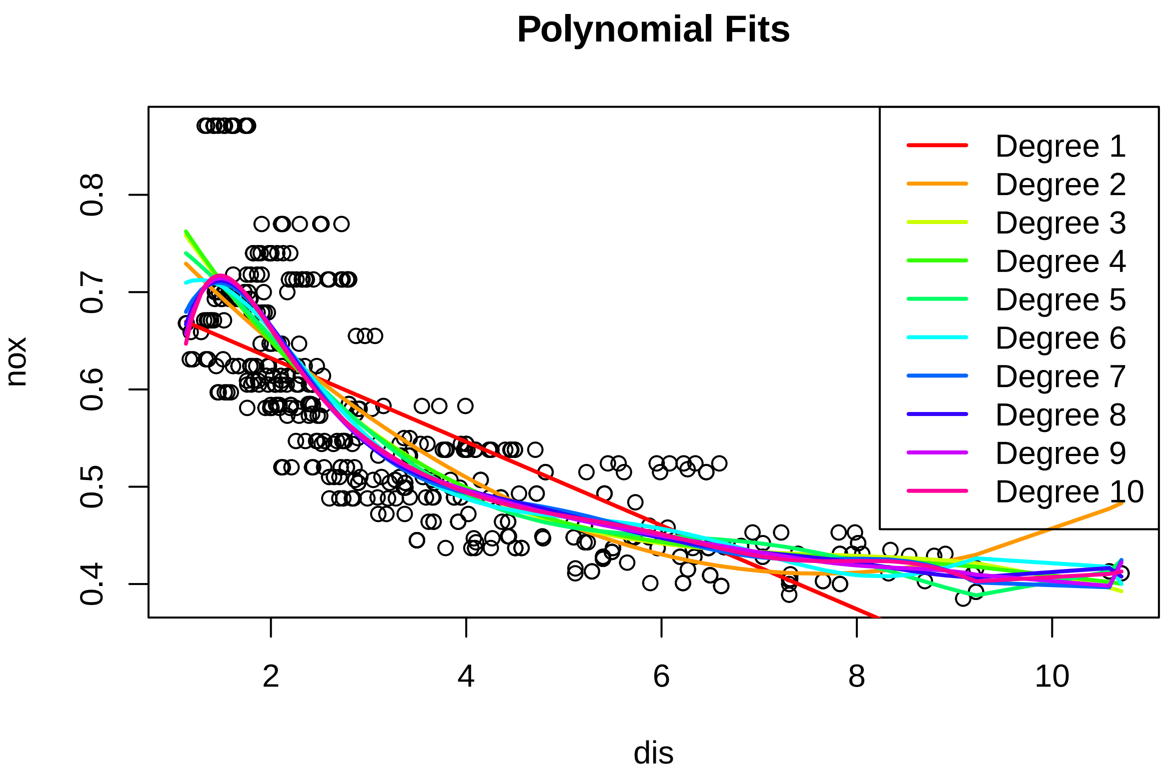
```r
set.seed(432)
trn.idx=sample(1:nrow(ISLR2::Boston),450)
tst.boston=ISLR2::Boston[-trn.idx,]
trn.boston=ISLR2::Boston[trn.idx,]
```

`nox` variable is your response variable. `dis` variable is your (only) predictor variable for this question.

(a) Use the `poly()` function to fit polynomials of degree 1 to 10. Plot polynomial fits.

```r
fits <- list()
for (i in 1:10) {
  fits[[i]] <- lm(nox ~ poly(dis, i), data = trn.boston)
}
```

```r
plot(trn.boston$dis, trn.boston$nox, main = "Polynomial Fits", xlab = "dis", ylab = "nox")

colors <- rainbow(10)
for (i in 1:10) {
  lines(sort(trn.boston$dis), predict(fits[[i]], newdata = data.frame(dis = sort(trn.boston$dis))), col
}

legend("topright", legend = paste("Degree", 1:10), col = colors, lwd = 2)
```



(b) Using anova function to select the optimal degree for the polynomial, and explain your results. You do not have to pick one model, you can suggest multiple models or just explain why certain models are not ideal.

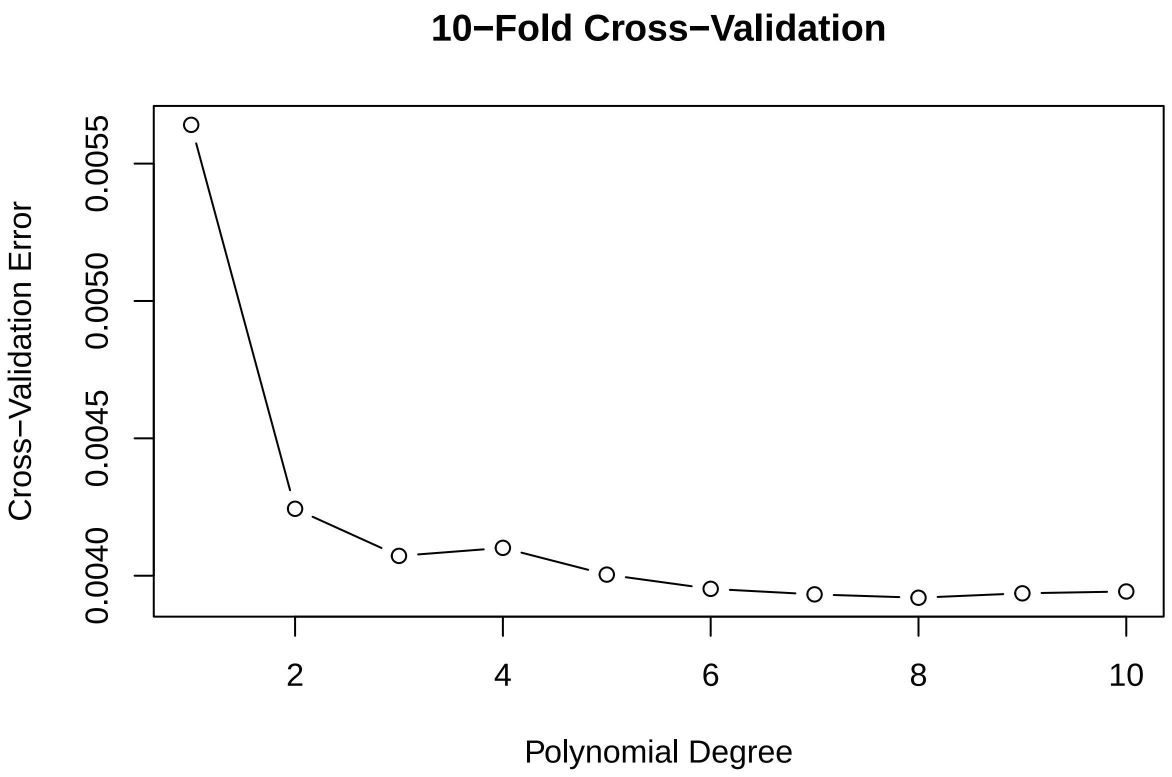
```r
anova_results <- anova(fits[[1]], fits[[2]], fits[[3]], fits[[4]], fits[[5]], fits[[6]], fits[[7]], fit
print(anova_results)
```

```
## Analysis of Variance Table
##
## Model  1: nox ~ poly(dis, i)
## Model  2: nox ~ poly(dis, i)
## Model  3: nox ~ poly(dis, i)
## Model  4: nox ~ poly(dis, i)
## Model  5: nox ~ poly(dis, i)
## Model  6: nox ~ poly(dis, i)
## Model  7: nox ~ poly(dis, i)
## Model  8: nox ~ poly(dis, i)
## Model  9: nox ~ poly(dis, i)
## Model 10: nox ~ poly(dis, i)
##    Res.Df    RSS Df Sum of Sq        F    Pr(>F)
## 1     448 2.5206
## 2     447 1.8919  1   0.62867 161.2454 < 2.2e-16 ***
## 3     446 1.8042  1   0.08770  22.4949 2.853e-06 ***
## 4     445 1.8032  1   0.00104   0.2671  0.605542
## 5     444 1.7806  1   0.02262   5.8011  0.016427 *
## 6     443 1.7470  1   0.03351   8.5953  0.003546 **
## 7     442 1.7221  1   0.02491   6.3889  0.011834 *
## 8     441 1.7172  1   0.00491   1.2592  0.262407
## 9     440 1.7144  1   0.00286   0.7347  0.391816
## 10    439 1.7116  1   0.00276   0.7074  0.400780
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Looking at the ANOVA table, we can see how well different polynomial models fit the data. Models with degrees 1 and 2 do a good job, showing a big improvement over a simple model. Adding a cubic term (degree 3) doesn't help much, so it's probably not worth the extra complxity. Models with degrees 4, 5, and 6 all show siginificant improvements, meaning they capture the relationship betwen nox and dis better. However, going beyond degree 6 (degrees 7 to 10) doesn't make things better and might even overfit the data. So, degrees 4, 5, or 6 seem to be the best choics as they balance fit and complexity well.

(c) Using 10-fold Cross-validation, select the optimal degree for the polynomial and explain your reason.

```r
cv_error <- rep(0, 10)

# poly degrees 1 to 10
for (i in 1:10) {
  glm_fit <- glm(nox ~ poly(dis, i), data = trn.boston)
  cv_result <- cv.glm(trn.boston, glm_fit, K = 10)
  cv_error[i] <- cv_result$delta[1]
}

plot(1:10, cv_error, type = "b", xlab = "Polynomial Degree", ylab = "Cross-Validation Error", main = "1(
```



Based on the plot, the best polynomal degree for predicting nox using dis is 3. The plot shows that the cross-validation error drops sharp when moving from degree 1 to degree 3, indicating a big improvement in the model's performance. Beyond degree 2, the cross-validation error doesn't decrease significantly and remains fairly stable, suggesting that higher-degree polynomials don't add much valuee and might even overfit the data. So, a polynomial of degree 3 strikes the right balance by providing a good fit without unnecesary complexity.

Question 5 (GAM)

In this question, we will conduct regression using `Boston` data from the `ISLR2` package.

```r
set.seed(432)
trn.idx=sample(1:nrow(ISLR2::Boston),450)
tst.boston=ISLR2::Boston[-trn.idx,]
trn.boston=ISLR2::Boston[trn.idx,]
```

- `nox` variable is your response variable.
- You may choose your own predictor variables.

Come up with at least 3 generalized additive models, using variables of your own choice. Compare your models (on training data) and analyze which model works best.

```r
gam1 <- gam(nox ~ s(dis) + s(rm) + s(age), data = trn.boston)
gam2 <- gam(nox ~ s(dis) + s(lstat) + s(indus), data = trn.boston)
gam3 <- gam(nox ~ s(dis) + s(crim) + s(tax), data = trn.boston)

# Summarizeing
summary(gam1)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## nox ~ s(dis) + s(rm) + s(age)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.557144   0.002851   195.4   <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(dis) 4.694  5.777 53.127  < 2e-16 ***
## s(rm)  5.748  6.946  3.026  0.00409 **
## s(age) 1.000  1.000 16.635 5.44e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.732   Deviance explained = 73.9%
## GCV = 0.0037606  Scale est. = 0.0036566  n = 450
```

```
summary(gam2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## nox ~ s(dis) + s(lstat) + s(indus)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.557144   0.002401   232.1   <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(dis)   3.612  4.555 33.822  < 2e-16 ***
## s(lstat) 6.900  7.994  3.871 0.000206 ***
## s(indus) 8.120  8.762 21.794  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =   0.81   Deviance explained = 81.8%
## GCV = 0.0027117  Scale est. = 0.0025934  n = 450
```

```
summary(gam3)
```

```
##
```

```
## Family: gaussian
## Link function: identity
##
## Formula:
## nox ~ s(dis) + s(crim) + s(tax)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.557144   0.002339   238.2   <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(dis)  6.579  7.723 35.090  <2e-16 ***
## s(crim) 8.680  8.961 20.822  <2e-16 ***
## s(tax)  6.060  7.127  2.132  0.0426 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =   0.82   Deviance explained = 82.8%
## GCV = 0.0025911  Scale est. = 0.0024626  n = 450
```

```r
predict_gam1 <- predict(gam1, trn.boston)
predict_gam2 <- predict(gam2, trn.boston)
predict_gam3 <- predict(gam3, trn.boston)

rmse <- function(actual, predicted) {
  sqrt(mean((actual - predicted)^2))
}

rmse_gam1 <- rmse(trn.boston$nox, predict_gam1)
rmse_gam2 <- rmse(trn.boston$nox, predict_gam2)
rmse_gam3 <- rmse(trn.boston$nox, predict_gam3)

rmse_gam1
```

```
## [1] 0.0596281
```

```r
rmse_gam2
```

```
## [1] 0.04980199
```

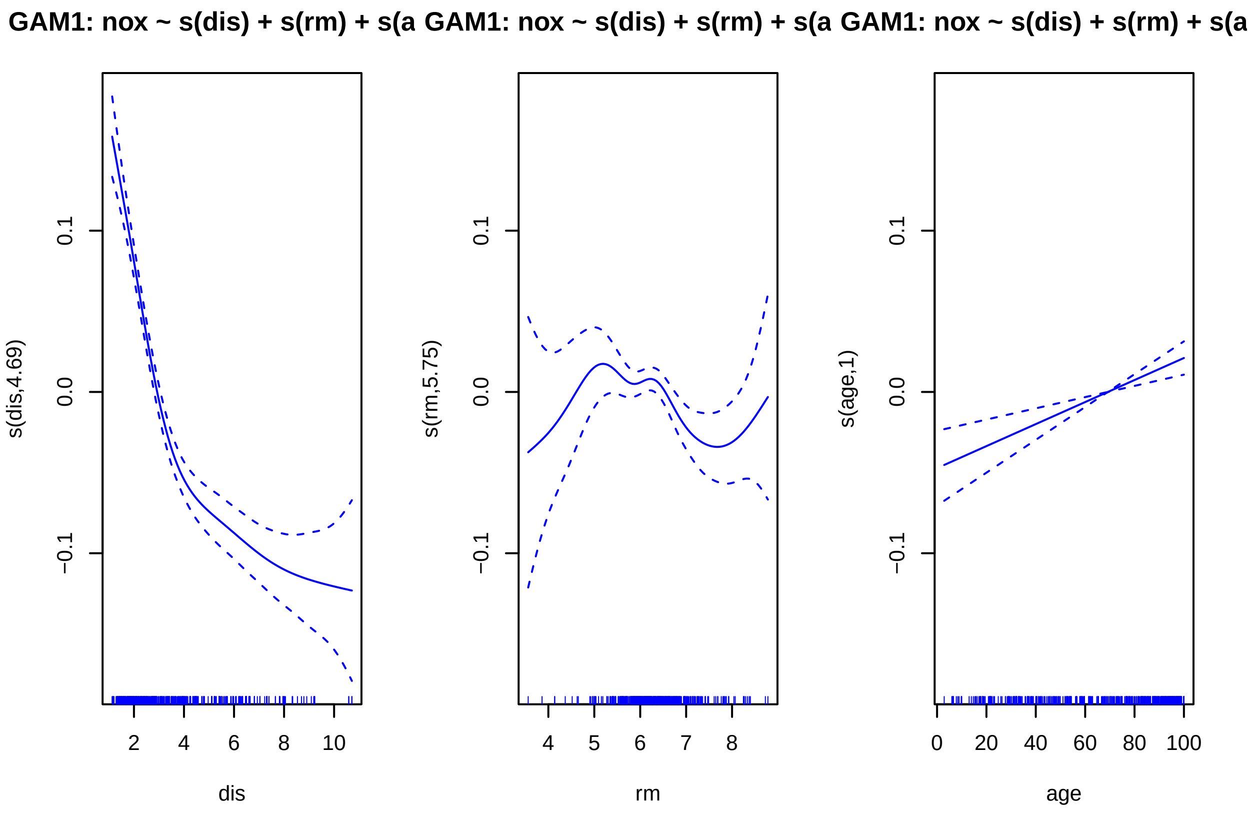
```r
rmse_gam3
```

```
## [1] 0.04837809
```

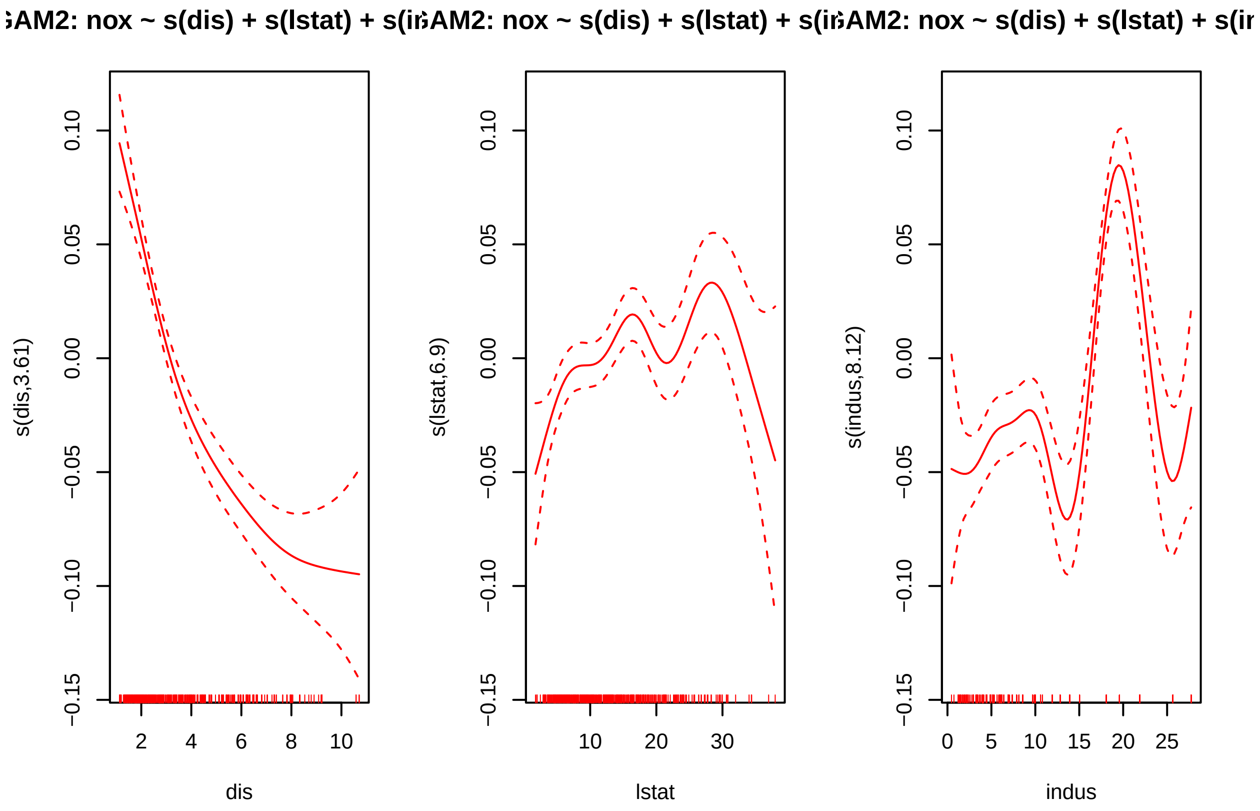
```r
rmse_values <- data.frame(
  Model = c("GAM1", "GAM2", "GAM3"),
  RMSE = c(rmse_gam1, rmse_gam2, rmse_gam3)
)

print(rmse_values)
```

```
##    Model        RMSE
## 1  GAM1 0.05962810
## 2  GAM2 0.04980199
## 3  GAM3 0.04837809
```

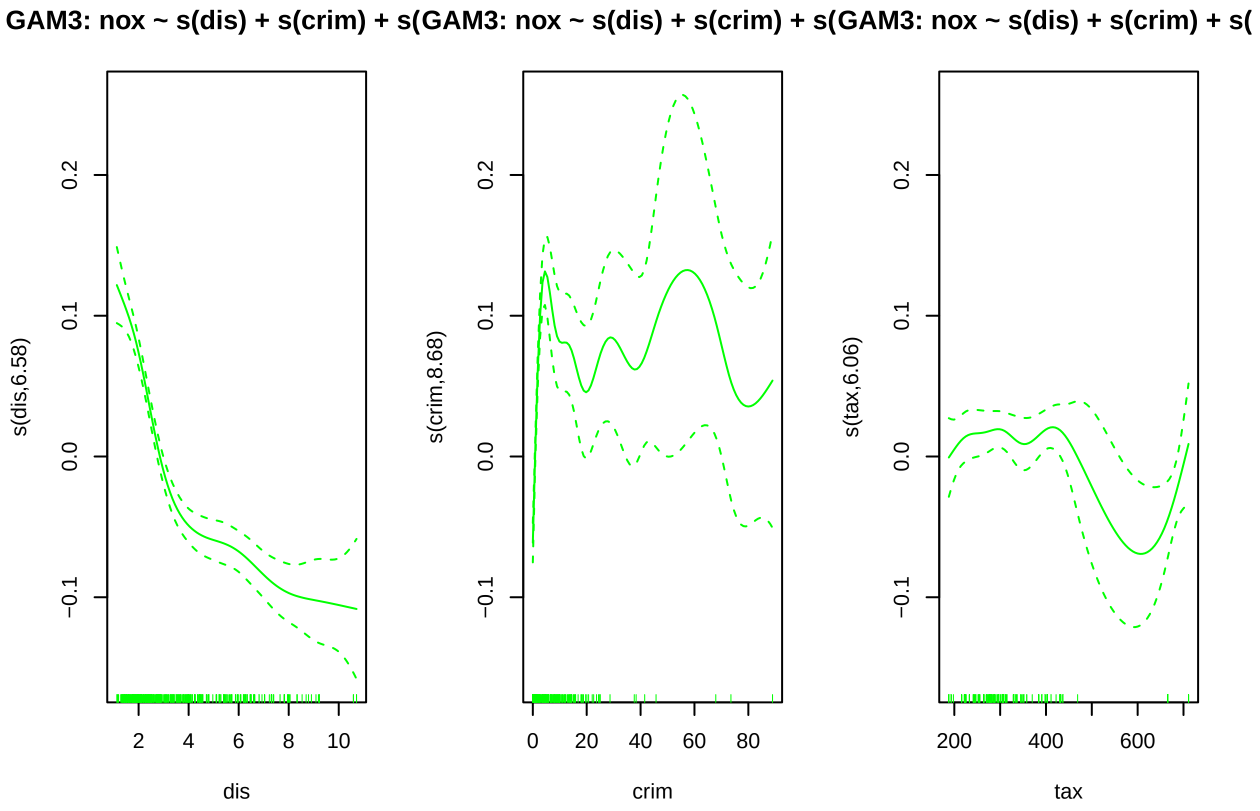
```
# Plotting
par(mfrow = c(1, 3))
plot(gam1, se = TRUE, col = "blue", main = "GAM1: nox ~ s(dis) + s(rm) + s(age)")
```



```
plot(gam2, se = TRUE, col = "red", main = "GAM2: nox ~ s(dis) + s(lstat) + s(indus)")
```



```r
plot(gam3, se = TRUE, col = "green", main = "GAM3: nox ~ s(dis) + s(crim) + s(tax)")
```



Based on the results, GAM3, which uses dis, crim, and tax as predictors, perfrms the best for predicting nox. It has the lowest RMSE of 0.0484, indicating it fits the data more accurately than the other models. The plots show that dis has a strong negative non-linear relationship with nox, while crim and tax also show significant non-linear effects. Overal, GAM3's predctors capture the variations in nox more efectively, making it the best model among the three.