GroupC_HM3

Trabucco, Suklan, Billo, Tavano

2025-01-08

Contents

FSDS - Chapter 4

Ex 4.24

> *Refer to the vegetarian survey result in Exercise 4.6, with $n = 25$ and no vegetarians.*
>
> *(a) Find the Bayesian estimate of π using a beta prior distribution with $\alpha = \beta$ equal (i) 0.5, (ii) 1.0, (iii) 10.0. Explain how the choice of prior distribution affects the posterior mean estimate.*

In order to solve this point, we compute the formula for the posterior distribution:

$$Beta(\alpha + x, \beta + n - x)$$

Thus, the posterior mean will be: $\frac{\alpha+x}{\alpha+\beta+n}$:

```
n <- 25
x <- 0

posterior_mean <- function(alpha, beta, x, n) {
  (alpha + x) / (alpha + beta + n)
}
alpha_beta_values <- c(0.5, 1.0, 10.0)

results <- sapply(alpha_beta_values, function(a) posterior_mean(a, a, x, n))
names(results) <- paste0("alpha=beta=", alpha_beta_values)
results
```

```
## alpha=beta=0.5   alpha=beta=1  alpha=beta=10
##     0.01923077     0.03703704     0.22222222
```

The posterior mean estimate is strongly affected by the choice of the prior distribution and its parameters: here we see that the mean estimate of the distribution π changes according to its parameters, as there is no x from the posterior to influence it in this case. In this setting where the target group is non-sampled (there are 0 vegetarians).

> *(b) If you were planning how to take a larger survey from the same population, explain how you can use the posterior results of the previous survey with $n = 25$* based on the prior with $\alpha = \beta = 1$ to form the prior distribution to use with the new survey results.

We can use the posterior results from the previous survey as a prior:

$$Beta(\alpha_{post} = 1 + x,\ \beta_{post} = 1 + n - x) = Beta(1, 26)$$

For the next survey, we can use this distribution as the new prior.

If in the new survey we will get x' vegetarians out of n' observations, the new posterior distribution will be:

$$Beta(\alpha' = \alpha_{post} + x',\ \beta' = \beta_{post} + n' - x')$$

Thus, the posterior mean of this distribution will be: $\frac{\alpha'}{\alpha'+\beta'}$

Ex 4.62

> *For the bootstrap method, explain the similarity and difference between the true sampling distribution of $\hat{\theta}$ and the empirically-generated bootstrap distribution in terms of its center and its spread.*

Solution

The bootstrap method is a resampling technique used to estimate the sampling distribution of a statistic, such as $\hat{\theta}$ by drawing repeated samples (with replacement) from the observed data.

The true sampling distribution of $\hat{\theta}$ represents the variability of the statistics if we repeatedly sampled from the actual population, centering around the true parameter θ, assuming that $\hat{\theta}$ is unbiased and with spread reflecting the true population's variability.

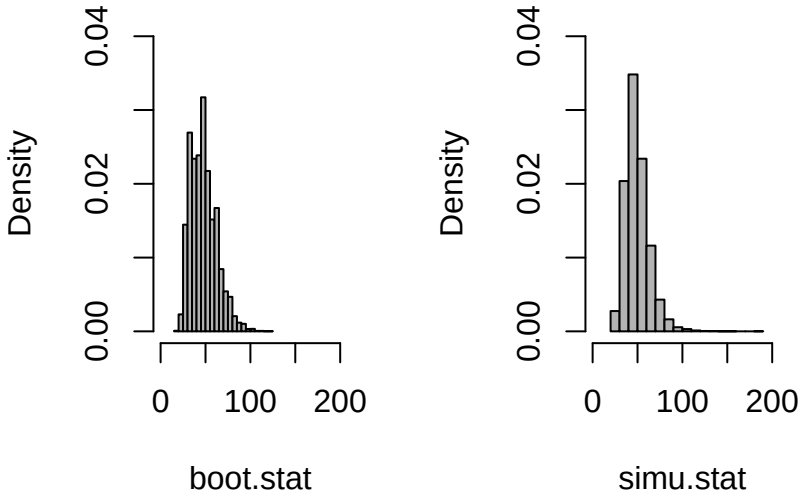
On the other hand, the bootstrap distribution is constructed by resampling from the observed data, as stated before, centering around the observed sample estimate $\hat{\theta}$ rather than θ and approximating the spread of the true sampling distribution based on the variability within the sample.

While the bootstrap distribution closely approximates the true distribution in large samples, difference can be noted in small samples or when the sample is not representative of the population, potentially leading to discrepancies in both centered and spread.

```
par(mfrow = c(1, 2))
y <- c(1, 4, 6, 12, 13, 14, 18, 19, 20, 22, 23, 24, 26, 31, 34,
37, 46, 47, 56, 61, 63, 65, 70, 97, 385)
n <- length(y); set.seed(1989);
B <- 10^4
boot.sample <- matrix(NA, nrow = B, ncol = n)
boot.sample[1,] <- sample(y, n, replace = TRUE)
boot.sample[2,] <- sample(y, n, replace = TRUE)
boot.sample[1, ] # sample output
```

```
##  [1] 22 20  4 34 70 13 24 70 13 63 18 12 46  6 23 19 46 63  6 20 12  4 65 31 31
```

```
# bootstrap mean estimates
for(i in 1:B) {
    boot.sample[i,] <- sample(y, n, replace = TRUE)
}
boot.stat <- rowMeans(boot.sample)
hist(boot.stat, main="", breaks=20, prob=TRUE, col=gray(0.7),
xlim=c(0, 200), ylim=c(0, 0.04))

# actual mean estimates
B <- 10^4;
simu.sample <- matrix(NA, nrow = B, ncol = n)
for(i in 1:B) simu.sample[i,] <- mean(30 * exp(rnorm(n)))
simu.stat <- rowMeans(simu.sample)
hist(simu.stat, main="", breaks=20, prob=TRUE, col=gray(0.7),
xlim=c(0, 200), ylim=c(0, 0.04))
```




```
c(mean(boot.sample), mean(simu.sample))
```

```
## [1] 47.86009 49.66223
```

The two parameters are close but there still is a significant difference.

FSDS - Chapter 8

Ex 8.4

> *Refer to Exercise 8.1. Construct a classification tree, and prune strongly until the tree uses a single explanatory variable. Which crabs were predicted to have satellites? How does the proportion of correct predictions compare with the more complex tree in Figure 8.2?*

Solution

```
library(rpart)
suppressWarnings(library(rpart.plot))
crabs = read.table("https://stat4ds.rwth-aachen.de/data/Crabs.dat", header = TRUE)
str(crabs)
```

```
## 'data.frame':    173 obs. of  7 variables:
##  $ crab  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ sat   : int  8 0 9 0 4 0 0 0 0 0 ...
##  $ y     : int  1 0 1 0 1 0 0 0 0 0 ...
##  $ weight: num  3.05 1.55 2.3 2.1 2.6 2.1 2.35 1.9 1.95 2.15 ...
##  $ width : num  28.3 22.5 26 24.8 26 23.8 26.5 24.7 23.7 25.6 ...
##  $ color : int  2 3 1 3 3 2 1 3 2 3 ...
##  $ spine : int  3 3 1 3 3 3 1 2 1 3 ...
```

```
summary(crabs)
```

```
##       crab          sat               y              weight          width      
##  Min.   :  1   Min.   : 0.000   Min.   :0.0000   Min.   :1.200   Min.   :21.0  
##  1st Qu.: 44   1st Qu.: 0.000   1st Qu.:0.0000   1st Qu.:2.000   1st Qu.:24.9  
##  Median : 87   Median : 2.000   Median :1.0000   Median :2.350   Median :26.1  
##  Mean   : 87   Mean   : 2.919   Mean   :0.6416   Mean   :2.437   Mean   :26.3  
##  3rd Qu.:130   3rd Qu.: 5.000   3rd Qu.:1.0000   3rd Qu.:2.850   3rd Qu.:27.7  
##  Max.   :173   Max.   :15.000   Max.   :1.0000   Max.   :5.200   Max.   :33.5  
##      color           spine      
##  Min.   :1.000   Min.   :1.000  
##  1st Qu.:2.000   1st Qu.:2.000  
##  Median :2.000   Median :3.000  
##  Mean   :2.439   Mean   :2.486  
##  3rd Qu.:3.000   3rd Qu.:3.000  
##  Max.   :4.000   Max.   :3.000
```

```
# using factor "Yes/no" as substitute of y.
crabs$satellites <- as.factor(ifelse(crabs$sat > 0, "Yes", "No"))

#building the regression tree
set.seed(123)
crabs_tree <- rpart(satellites ~ weight+color, data = crabs, method = "class")
crabs_tree
```

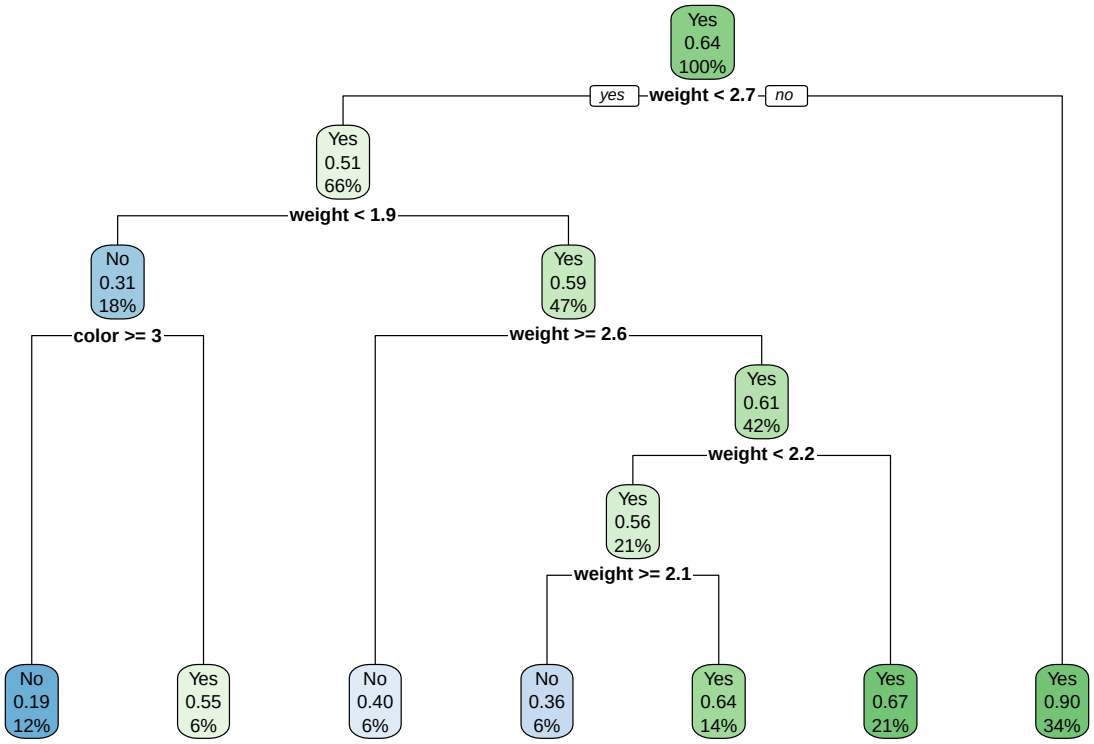
```
## n= 173
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 173 62 Yes (0.3583815 0.6416185)
##    2) weight< 2.675 114 56 Yes (0.4912281 0.5087719)
##      4) weight< 1.925 32 10 No (0.6875000 0.3125000)
##        8) color>=2.5 21  4 No (0.8095238 0.1904762) *
```

```
##        9) color< 2.5 11  5 Yes (0.4545455 0.5454545) *
##      5) weight>=1.925 82 34 Yes (0.4146341 0.5853659)
##       10) weight>=2.575 10  4 No (0.6000000 0.4000000) *
##       11) weight< 2.575 72 28 Yes (0.3888889 0.6111111)
##         22) weight< 2.2375 36 16 Yes (0.4444444 0.5555556)
##           44) weight>=2.125 11  4 No (0.6363636 0.3636364) *
##           45) weight< 2.125 25  9 Yes (0.3600000 0.6400000) *
##         23) weight>=2.2375 36 12 Yes (0.3333333 0.6666667) *
##    3) weight>=2.675 59  6 Yes (0.1016949 0.8983051) *
```

```
# using printcp function to detail CP, nsplit, rel error, xerror and xstd
printcp(crabs_tree)
```

```
##
## Classification tree:
## rpart(formula = satellites ~ weight + color, data = crabs, method = "class")
##
## Variables actually used in tree construction:
## [1] color  weight
##
## Root node error: 62/173 = 0.35838
##
## n= 173
##
##         CP nsplit rel error  xerror     xstd
## 1 0.096774      0   1.00000 1.00000 0.101728
## 2 0.032258      2   0.80645 0.80645 0.096166
## 3 0.024194      3   0.77419 1.00000 0.101728
## 4 0.016129      5   0.72581 1.01613 0.102083
## 5 0.010000      6   0.70968 1.01613 0.102083
```

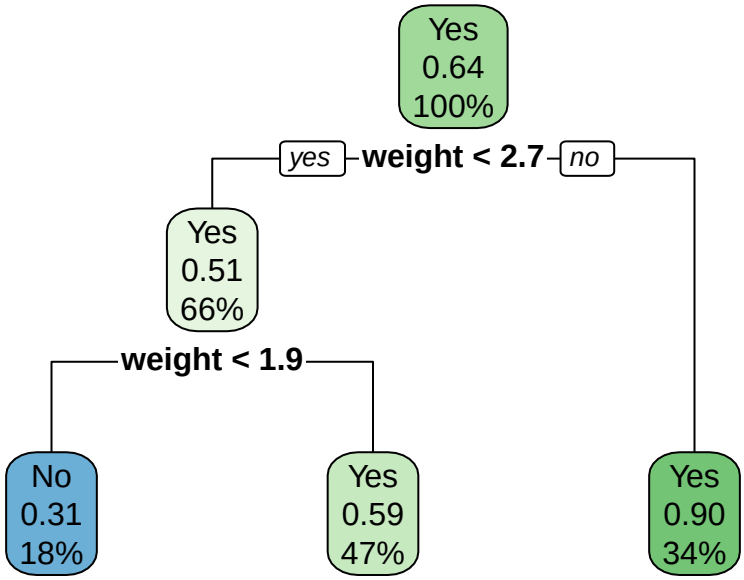
```
rpart.plot(crabs_tree)
```




```
# pruning into a single variable (weight)
tree_pruned <- prune(crabs_tree, cp =0.05)
printcp(tree_pruned)
```

```
##
## Classification tree:
## rpart(formula = satellites ~ weight + color, data = crabs, method = "class")
##
## Variables actually used in tree construction:
## [1] weight
##
## Root node error: 62/173 = 0.35838
##
## n= 173
##
##         CP nsplit rel error  xerror     xstd
## 1 0.096774      0   1.00000 1.00000 0.101728
## 2 0.050000      2   0.80645 0.80645 0.096166
```

```
rpart.plot(tree_pruned)
```




```
#making predictions
predicted <- predict(tree_pruned, type = "class")
comparison_table <- table(predicted, crabs$satellites)
comparison_table
```

```
##
## predicted  No Yes
##       No   22  10
##       Yes  40 101
```

```
acc = sum(diag(comparison_table))/sum(comparison_table)
acc
```

```
## [1] 0.7109827
```

LAB

Ex LAB

Suppose you receive $n = 15$ phone calls in a day, and you want to build a model to assess their average length. Your likelihood for each call length is $y_i \sim \mathcal{E}sp(\lambda)$. Now, you have to choose the prior $\pi(\lambda)$. Please, tell which of these priors is adequate to describe the problem, and provide a short motivation for each of them:

1. $\pi(\lambda)$ = *Beta(4,2);*
2. $\pi(\lambda)$ = *Normal(1,2);*
3. $\pi(\lambda)$ = *Gamma(4,2);*

Now, compute your posterior as $\pi(\lambda|y) \propto L(\lambda; y)\pi(\lambda)$ for the selected prior. If your first choice was correct, you will be able to compute it analytically.

Solution

Priors Evaluation:

1. $\pi(\lambda)$ = Beta(4,2):

Beta distribution is common used for probabilities, with $p \in [0,1]$. In this case the parameter of an exponential distribution λ is not restricted in [0,1] but it can take any positive value. Thus, using a Beta prior for this problem is not appropriate.

2. $\pi(\lambda)$ = Normal(1,2):

Similarly with the previous point, Normal distribution can take both positive and negative values, which is not consistent with λ adopted by the Exponential distribution. For this reason, Normal(1,2) distribution is not suitable.

3. $\pi(\lambda)$ = Gamma(4,2):

This is the most suitable prior because the Gamma distribution is defined on $\lambda > 0$ and it is **conjugate** to the Exponential likelihood. Using a Gamma prior will allow us to compute the posterior analytically.

Posterior Computation with $\pi(\lambda)$ = Gamma(4,2)

First, let's compute the likelihood:

$$L(\lambda; y) = \prod_{i=1}^{n} \lambda \; e^{-\lambda y_i} = \lambda^n \; e^{-\lambda \; \sum_{i=1}^{n} y_i}$$

Then, compute the Gamma prior with $\alpha = 4$ and $\beta = 2$:

$$\pi(\lambda) = \frac{\beta}{\Gamma(\alpha)} \; \lambda^{\alpha-1} \; e^{-\beta\lambda}$$

Now, combine the terms to obtain the posterior distribution:

$$\pi(\lambda|y) \propto L(\lambda; y)\pi(\lambda)$$
$$\pi(\lambda|y) \propto \lambda^n \; e^{-\lambda \; \sum_{i=1}^{n} y_i} \; \lambda^{\alpha-1} \; e^{-\beta\lambda}$$
$$\pi(\lambda|y) \propto \lambda^{n+\alpha-1} \; e^{-\lambda(\sum_{i=1}^{n} y_i + \beta)}$$

Finally, identify the parameters α' and β' of the posterior Gamma distribution:

$$\pi(\lambda|y) = Gamma(\alpha', \beta')$$

With $\alpha' = n + \alpha = 15 + 4 = 19$ and $\beta' = \sum_{i=1}^{n} y_i \; + \; \beta$.

ISLR - Chapter 6

Ex 6.9

In this exercise, we will predict the number of applications received using the other variables in the College data set.

```
college_data <- read.csv("college.csv", header = TRUE)
summary(college_data)
```

```
##       X                Private               Apps           Accept
##  Length:777         Length:777         Min.   :   81   Min.   :   72
##  Class :character   Class :character   1st Qu.:  776   1st Qu.:  604
##  Mode  :character   Mode  :character   Median : 1558   Median : 1110
##                                        Mean   : 3002   Mean   : 2019
##                                        3rd Qu.: 3624   3rd Qu.: 2424
##                                        Max.   :48094   Max.   :26330
##      Enroll       Top10perc       Top25perc      F.Undergrad
##  Min.   :  35   Min.   : 1.00   Min.   :  9.0   Min.   :  139
##  1st Qu.: 242   1st Qu.:15.00   1st Qu.: 41.0   1st Qu.:  992
##  Median : 434   Median :23.00   Median : 54.0   Median : 1707
##  Mean   : 780   Mean   :27.56   Mean   : 55.8   Mean   : 3700
##  3rd Qu.: 902   3rd Qu.:35.00   3rd Qu.: 69.0   3rd Qu.: 4005
##  Max.   :6392   Max.   :96.00   Max.   :100.0   Max.   :31643
##   P.Undergrad         Outstate       Room.Board       Books
##  Min.   :    1.0   Min.   : 2340   Min.   :1780   Min.   :  96.0
##  1st Qu.:   95.0   1st Qu.: 7320   1st Qu.:3597   1st Qu.: 470.0
##  Median :  353.0   Median : 9990   Median :4200   Median : 500.0
##  Mean   :  855.3   Mean   :10441   Mean   :4358   Mean   : 549.4
##  3rd Qu.:  967.0   3rd Qu.:12925   3rd Qu.:5050   3rd Qu.: 600.0
##  Max.   :21836.0   Max.   :21700   Max.   :8124   Max.   :2340.0
##     Personal         PhD            Terminal       S.F.Ratio
##  Min.   : 250   Min.   :  8.00   Min.   : 24.0   Min.   : 2.50
##  1st Qu.: 850   1st Qu.: 62.00   1st Qu.: 71.0   1st Qu.:11.50
##  Median :1200   Median : 75.00   Median : 82.0   Median :13.60
##  Mean   :1341   Mean   : 72.66   Mean   : 79.7   Mean   :14.09
##  3rd Qu.:1700   3rd Qu.: 85.00   3rd Qu.: 92.0   3rd Qu.:16.50
##  Max.   :6800   Max.   :103.00   Max.   :100.0   Max.   :39.80
##   perc.alumni        Expend        Grad.Rate
##  Min.   : 0.00   Min.   : 3186   Min.   : 10.00
##  1st Qu.:13.00   1st Qu.: 6751   1st Qu.: 53.00
##  Median :21.00   Median : 8377   Median : 65.00
##  Mean   :22.74   Mean   : 9660   Mean   : 65.46
##  3rd Qu.:31.00   3rd Qu.:10830   3rd Qu.: 78.00
##  Max.   :64.00   Max.   :56233   Max.   :118.00
```

```
str(college_data)
```

```
## 'data.frame':    777 obs. of  19 variables:
##  $ X          : chr  "Abilene Christian University" "Adelphi University" "Adrian College" "Agnes Sco
##  $ Private    : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ Apps       : int  1660 2186 1428 417 193 587 353 1899 1038 582 ...
##  $ Accept     : int  1232 1924 1097 349 146 479 340 1720 839 498 ...
##  $ Enroll     : int  721 512 336 137 55 158 103 489 227 172 ...
##  $ Top10perc  : int  23 16 22 60 16 38 17 37 30 21 ...
##  $ Top25perc  : int  52 29 50 89 44 62 45 68 63 44 ...
```

```
##  $ F.Undergrad: int  2885 2683 1036 510 249 678 416 1594 973 799 ...
##  $ P.Undergrad: int  537 1227 99 63 869 41 230 32 306 78 ...
##  $ Outstate   : int  7440 12280 11250 12960 7560 13500 13290 13868 15595 10468 ...
##  $ Room.Board : int  3300 6450 3750 5450 4120 3335 5720 4826 4400 3380 ...
##  $ Books      : int  450 750 400 450 800 500 500 450 300 660 ...
##  $ Personal   : int  2200 1500 1165 875 1500 675 1500 850 500 1800 ...
##  $ PhD        : int  70 29 53 92 76 67 90 89 79 40 ...
##  $ Terminal   : int  78 30 66 97 72 73 93 100 84 41 ...
##  $ S.F.Ratio  : num  18.1 12.2 12.9 7.7 11.9 9.4 11.5 13.7 11.3 11.5 ...
##  $ perc.alumni: int  12 16 30 37 2 11 26 37 23 15 ...
##  $ Expend     : int  7041 10527 8735 19016 10922 9727 8861 11487 11644 8991 ...
##  $ Grad.Rate  : int  60 56 54 59 15 55 63 73 80 52 ...
```

```
college_data$Private <- ifelse(college_data$Private == "Yes", 1, 0)
```

(a) Split the data set into a training set and a test set.

```
set.seed(123)
n_rows <- nrow(college_data)

train_idx = sample(1:n_rows, size=0.7*n_rows)

train_set <- college_data[train_idx, ]
test_set <- college_data[-train_idx, ]

n_rows
```

```
## [1] 777
```

```
dim(train_set)
```

```
## [1] 543  19
```

```
dim(test_set)
```

```
## [1] 234  19
```

(b) Fit a linear model using least squares on the training set, and report the test error obtained.

```
# Full Model
full_model <- lm(Apps ~ ., data = train_set[, -1])
summary(full_model)
```

```
##
## Call:
## lm(formula = Apps ~ ., data = train_set[, -1])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3097.8  -455.8   -46.5   343.8  6452.5
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -310.17331  481.30075  -0.644 0.519566
## Private     -681.96465  164.08211  -4.156 3.78e-05 ***
## Accept         1.22130    0.05921  20.626  < 2e-16 ***
## Enroll         0.08046    0.21794   0.369 0.712155
## Top10perc     49.33503    6.18296   7.979 9.31e-15 ***
```

```
## Top25perc    -16.11744    5.02717  -3.206 0.001428 **
## F.Undergrad    0.02284    0.03985   0.573 0.566831
## P.Undergrad    0.03541    0.03529   1.003 0.316139
## Outstate      -0.05446    0.02132  -2.555 0.010910 *
## Room.Board     0.18967    0.05275   3.596 0.000354 ***
## Books          0.21366    0.28099   0.760 0.447381
## Personal      -0.03685    0.07279  -0.506 0.612876
## PhD           -6.00401    5.34580  -1.123 0.261897
## Terminal      -5.01712    5.77787  -0.868 0.385609
## S.F.Ratio     -2.18927   14.83898  -0.148 0.882766
## perc.alumni   -8.01836    4.67330  -1.716 0.086792 .
## Expend         0.07614    0.01340   5.681 2.23e-08 ***
## Grad.Rate     10.63461    3.38228   3.144 0.001760 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 992.3 on 525 degrees of freedom
## Multiple R-squared:  0.9175, Adjusted R-squared:  0.9148
## F-statistic: 343.2 on 17 and 525 DF,  p-value: < 2.2e-16
```

As expected, many of the variables contained in the dataset do not seem to be useful in predicting the number of applications received and we end up in more of an *overfitting* situation as we can observe from the MSE:

```
# Prediction on test set
test_predictions <- predict(full_model, newdata = test_set)

# (Mean Squared Error)
test_error <- mean((test_set$Apps - test_predictions)^2)

# Output
cat("Mean Squared Error on test set (full model):", test_error, "\n")
```

```
## Mean Squared Error on test set (full model): 1734841
```

> *(c) Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.*

A ridge regression model will let us *shrink* some of the coefficients (associated to different variables of the model):

```
library(glmnet)
```

```
## Warning: il pacchetto 'glmnet' è stato creato con R versione 4.3.3
```

```
## Caricamento del pacchetto richiesto: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
x <- model.matrix(~ . - Apps, data = college_data)[, -1]
y <- college_data$Apps

set.seed(123)
idx <- sample(1:nrow(x), size = 0.75 * nrow(x))  # 75% train, 25% test
train_x <- x[idx, ]
test_x <- x[-idx, ]
train_y <- y[idx]
test_y <- y[-idx]
```

```r
# (10-fold)-cross-validation
set.seed(123)
cv_ridge <- cv.glmnet(train_x, train_y, alpha = 0)  # alpha = 0 for ridge regression (l2 norm)
best_lambda <- cv_ridge$lambda.min
cat("Optimal Lambda:", best_lambda, "\n")
```

```
## Optimal Lambda: 31883
```

```r
ridge_model <- glmnet(train_x, train_y, alpha = 0, lambda = best_lambda)
ridge_predictions <- predict(ridge_model, s = best_lambda, newx = test_x)

# MSE:
ridge_test_error <- mean((test_y - ridge_predictions)^2)
cat("Test Error (MSE):", ridge_test_error, "\n")
```

```
## Test Error (MSE): 16411119
```

(d) Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```r
set.seed(123)

cv_lasso <- cv.glmnet(train_x, train_y, alpha = 1)  # alpha = 1 for Lasso regression (l1 norm instead o
best_lambda <- cv_lasso$lambda.min
cat("Optimal Lambda (Lasso):", best_lambda, "\n")
```

```
## Optimal Lambda (Lasso): 122.8621
```

```r
lasso_model <- glmnet(train_x, train_y, alpha = 1, lambda = best_lambda)
lasso_predictions <- predict(lasso_model, s = best_lambda, newx = test_x)

lasso_test_error <- mean((test_y - lasso_predictions)^2)
cat("Test Error (MSE - Lasso):", lasso_test_error, "\n")
```

```
## Test Error (MSE - Lasso): 1936503
```

```r
# We want to extract the coefficients and count non-zero ones:
lasso_coefficients <- coef(lasso_model, s = best_lambda)
non_zero_coefficients <- sum(lasso_coefficients != 0) - 1
cat("Number of Non-Zero Coefficients is", non_zero_coefficients, "out of", ncol(x), "possible ones\n")
```

```
## Number of Non-Zero Coefficients is 21 out of 793 possible ones
```

Given that LASSO also performs **feature selection** (thanks to the L1-norm properties) we have 21 non-zero coefficients left in our model.

ISLR - Chapter 7

Ex 7.9

This question uses the variables **`dis`** *(the weighted mean of distances to five Boston employment centers) and* **`nox`** *(nitrogen oxides concentration in parts per 10 million) from the Boston data. We will treat* **`dis`** *as the predictor and* **`nox`** *as the response.*

(a) Use the **`poly()`** *function to fit a cubic polynomial regression to predict* **`nox`** *using* **`dis`**. *Report the regression output, and plot the resulting data and polynomial fits.*

```
library(MASS)
```

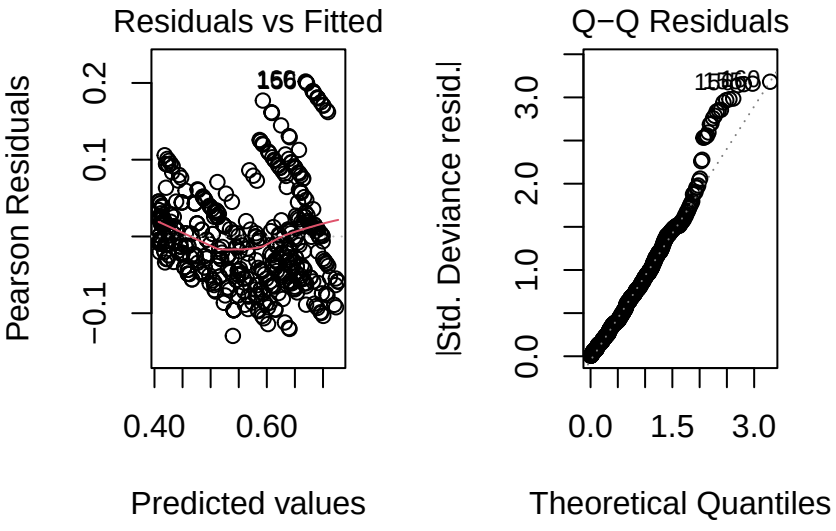
```
##
## Caricamento pacchetto: 'MASS'
```

```
## Il seguente oggetto è mascherato _da_ '.GlobalEnv':
##
##     crabs
```

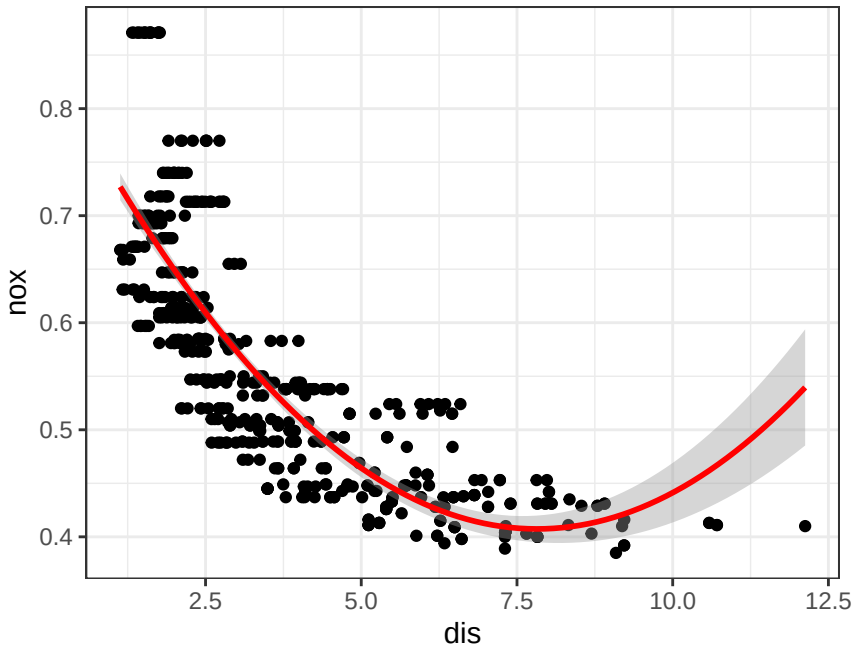
```
library(splines)
library(glmnet)
library(ggplot2)
```

```
## Warning: il pacchetto 'ggplot2' è stato creato con R versione 4.3.3
```

```
boston <- Boston

fitpoly <- glm(nox ~ poly(dis, 2, raw = TRUE), data = boston)
par(mfrow = c(1,2))
plot(fitpoly,  which = c(1,2))
```

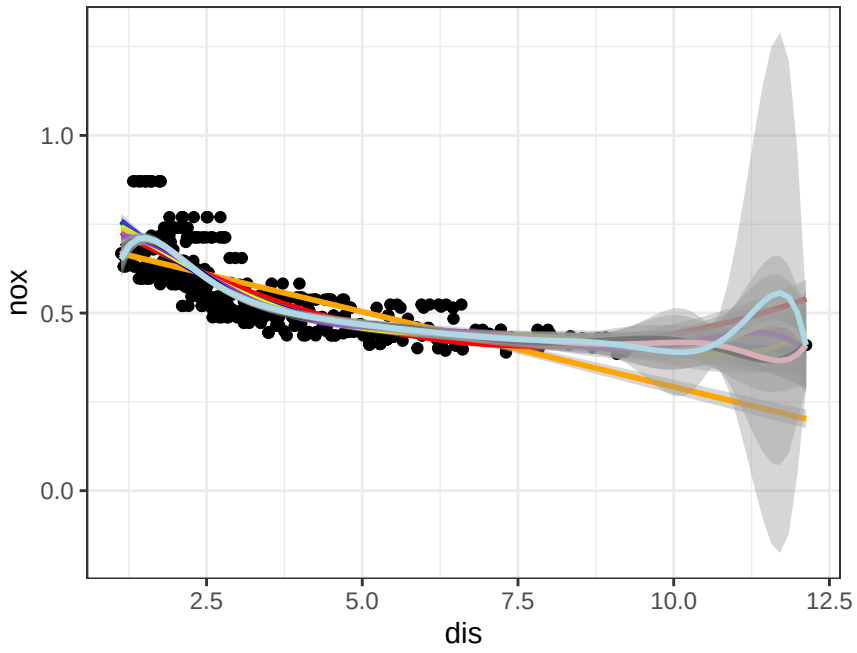



```
ggplot(boston, aes(dis, nox)) +
  geom_point() + theme_bw() +
  stat_smooth(method = lm, formula = y ~ poly(x, 2, raw = TRUE), col = "red")
```

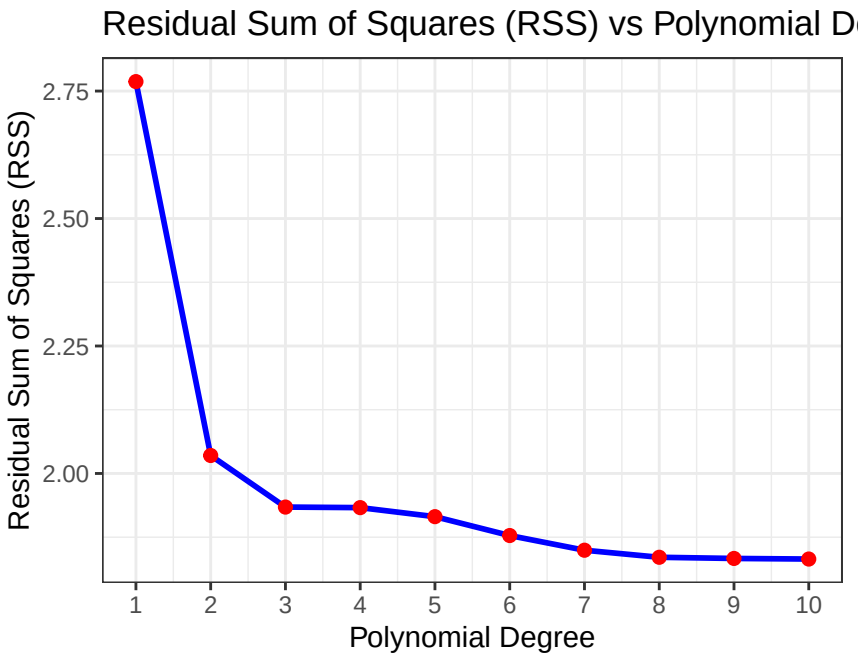


(b) Plot the polynomial fits for a range of different polynomial degrees (say, from 1 to 10), and report the associated residual sum of squares.

```
ggplot(boston, aes(dis, nox)) +
  geom_point() + theme_bw() +
  stat_smooth(method = lm, formula = y ~ poly(x, 1, raw = TRUE), col = "orange")+
  stat_smooth(method = lm, formula = y ~ poly(x, 2, raw = TRUE), col = "red")+
  stat_smooth(method = lm, formula = y ~ poly(x, 3, raw = TRUE), col = "green")+
  stat_smooth(method = lm, formula = y ~ poly(x, 4, raw = TRUE), col = "blue")+
  stat_smooth(method = lm, formula = y ~ poly(x, 5, raw = TRUE), col = "yellow")+
  stat_smooth(method = lm, formula = y ~ poly(x, 6, raw = TRUE), col = "purple")+
  stat_smooth(method = lm, formula = y ~ poly(x, 7, raw = TRUE), col = "black")+
  stat_smooth(method = lm, formula = y ~ poly(x, 8, raw = TRUE), col = "grey")+
  stat_smooth(method = lm, formula = y ~ poly(x, 9, raw = TRUE), col = "pink")+
  stat_smooth(method = lm, formula = y ~ poly(x, 10, raw = TRUE), col = "lightblue")
```




```r
# Fit polynomial models and compute RSS
rss_data <- data.frame(Degree = 1:10, RSS = NA)

for (degree in rss_data$Degree) {
  model <- lm(nox ~ poly(dis, degree, raw = TRUE), data = boston)
  residuals <- model$residuals
  rss_data$RSS[rss_data$Degree == degree] <- sum(residuals^2)
}

# Plot RSS vs. Polynomial Degree
suppressWarnings(

  ggplot(rss_data, aes(x = Degree, y = RSS)) +
  geom_line(size = 1, color = "blue") +
  geom_point(size = 2, color = "red") +
  theme_bw() +
  scale_x_continuous(breaks = 1:10) +
  labs(
    title = "Residual Sum of Squares (RSS) vs Polynomial Degree",
    x = "Polynomial Degree",
    y = "Residual Sum of Squares (RSS)"
  )

)
```



Obviously, in such a setting, the more complicated model has the lowest error, but it ends up just *fitting* the data, without providing useful information. As we can observe from the MSE vs Degree plot, we could already select a polynomial of degree 3, given that is already has a very low MSE and an even higher degree does not seem to provide a much better solution.

Another way to test this is by using cross-validation:

> *(c) Perform cross-validation or another approach to select the optimal degree for the polynomial, and explain your results.*

```
suppressPackageStartupMessages(suppressWarnings({
  library(caret)
  library(lattice)
  library(dplyr)
}))
```

```
library(caret)
library(dplyr)

set.seed(123)

# First we split the data
idx <- sample(1:nrow(boston), size = 0.75 * nrow(boston)) # 75% train, 25% test
train <- boston[idx,]
test <- boston[-idx,]

metrics <- data.frame(matrix(ncol = 5, nrow = 10))
colnames(metrics) <- c('R2', 'AdjustedR2', 'RMSE', 'MAE', 'AIC')

for (i in 1:10){
  model <- lm(nox ~ poly(dis, i, raw = TRUE), data = train)

  # Use the model we obtained to make predictions on the test set:
  predictions <- model %>% predict(test)
```

```
  # and examine R-squared, RMSE, and MAE of predictions:
  metrics[i, "R2"] <- (R2(predictions, test$nox))
  metrics[i, "RMSE"] <- (RMSE(predictions, test$nox))
  metrics[i, "MAE"] <- (MAE(predictions, test$nox))
  n <- nrow(test)
  k <- i
  adj_r2 <- 1 - ((1 - metrics[i, "R2"]) * (n - 1) / (n - k - 1))
  metrics[i, "AdjustedR2"] <- adj_r2
  metrics[i, "AIC"] <- AIC(model)


}
# select the best model(s)
indexr2 <- which.max(metrics[, "R2"])
indexrmse <- which.min(metrics[, "RMSE"])
indexmae <-  which.min(metrics[, "MAE"])
indexr2a <- which.max(metrics[, "AdjustedR2"])
indexaic <- which.min(metrics[, "AIC"])


results <- data.frame(
  Metric = c("R2", "RMSE", "MAE", "Adjusted R2", "AIC"),
  BestModelIndex = c(indexr2, indexrmse, indexmae, indexr2a, indexaic)
)

print(results)
```

```
##        Metric BestModelIndex
## 1          R2              9
## 2        RMSE              9
## 3         MAE              8
## 4 Adjusted R2              8
## 5         AIC              8
```

By considering this five different tools to measure the performance of our model, we can observe that even using measures that penalize model complexity (such as adjusted R^2 and AIC) we end up selecting a very high degree polynomial.

(d) Use the `bs()` *function to fit a regression spline to predict* **`nox`** *using* `dis`*. Report the output for the fit using four degrees of freedom. How did you choose the knots? Plot the resulting fit.*

```
library(splines)

# Fit a regression spline with 4 degrees of freedom
spline_model <- lm(nox ~ bs(dis, df = 4), data = train)
summary(spline_model)
```

```
##
## Call:
## lm(formula = nox ~ bs(dis, df = 4), data = train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.122284 -0.037910 -0.008286  0.021964  0.197748
##
```

```
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        0.72749    0.01538  47.292  < 2e-16 ***
## bs(dis, df = 4)1 -0.05794    0.02389  -2.426   0.0157 *
## bs(dis, df = 4)2 -0.44939    0.02644 -16.996  < 2e-16 ***
## bs(dis, df = 4)3 -0.20478    0.04644  -4.410 1.35e-05 ***
## bs(dis, df = 4)4 -0.36939    0.04613  -8.007 1.50e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06123 on 374 degrees of freedom
## Multiple R-squared:  0.719,  Adjusted R-squared:  0.716
## F-statistic: 239.2 on 4 and 374 DF,  p-value: < 2.2e-16
```

By default, when using the `bs()` function to fit a regression splines the knots are placed equidistanced on the quantiles of the predictor variable (which, in this case, is `dis`) and in this case:

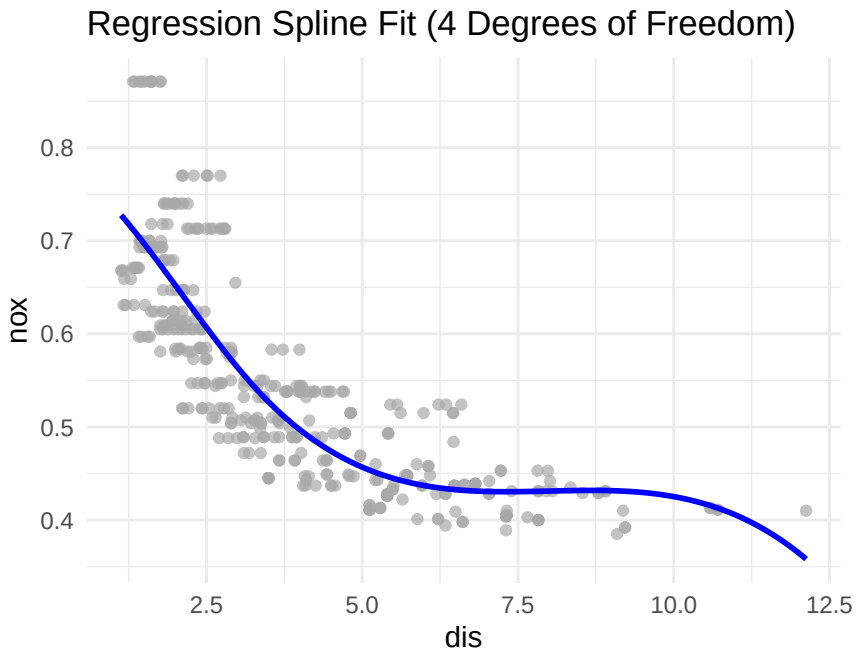
```
knots <- attr(bs(train$dis, df = 4), "knots")
boundary_knots <- attr(bs(train$dis, df = 4), "Boundary.knots")
knots; boundary_knots
```

```
## [1] 3.3317
```

```
## [1]  1.1296 12.1265
```

We then plot our results:

```
# Generate predictions for plotting
dis_values <- seq(min(train$dis), max(train$dis), length.out = 100)
predictions <- predict(spline_model, newdata = data.frame(dis = dis_values))

# Plot the data and spline fit
suppressWarnings(
ggplot(data = train, aes(x = dis, y = nox)) +
  geom_point(color = "darkgray", alpha = 0.7) +
  geom_line(data = data.frame(dis = dis_values, nox = predictions),
            aes(x = dis, y = nox), color = "blue", size = 1) +
  labs(title = "Regression Spline Fit (4 Degrees of Freedom)",
       x = "dis", y = "nox") +
  theme_minimal())
```



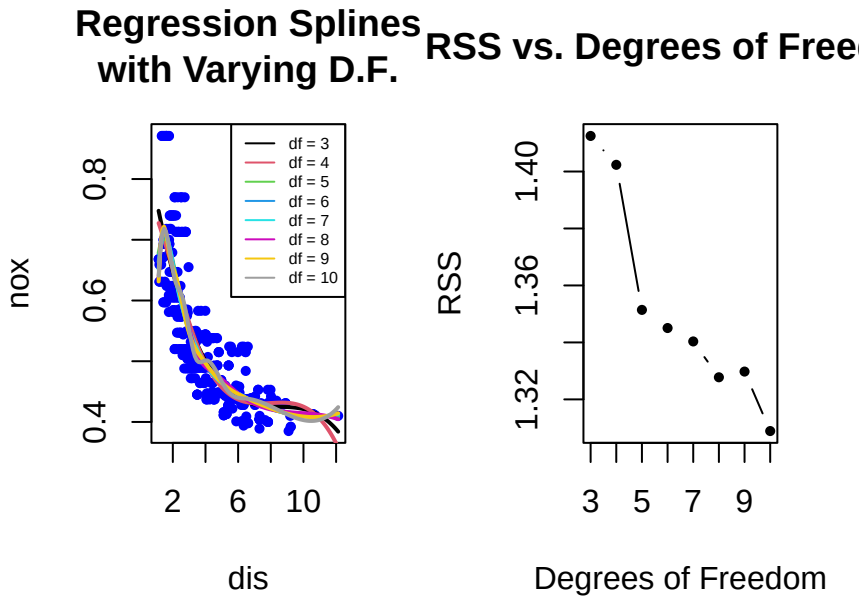
(e) Now fit a regression spline for a range of degrees of freedom, and plot the resulting fits and report the resulting RSS. Describe the results obtained.

```
par(mfrow = c(1,2))
df_range <- 3:10

rss_results <- data.frame(DegreesOfFreedom = df_range, RSS = numeric(length(df_range)))

# Plot the fits
plot(train$dis, train$nox, main = "Regression Splines\nwith Varying D.F.",
     xlab = "dis", ylab = "nox", pch = 16, col = "blue", cex=0.7)
legend("topright", legend = paste("df =", df_range), col = 1:length(df_range), lty = 1, cex = 0.5)

for (i in seq_along(df_range)) {
  # Fit the regression spline with i degrees of freedom
  spline_model <- lm(nox ~ bs(dis, df = df_range[i]), data = train)

  # Calculate RSS
  rss_results$RSS[i] <- sum(residuals(spline_model)^2)

  # Generate predictions for plotting
  dis_values <- seq(min(train$dis), max(train$dis), length.out = 100)
  predictions <- predict(spline_model, newdata = data.frame(dis = dis_values))

  # Add the fit to the plot
  lines(dis_values, predictions, col = i, lwd = 2)
}

# Print the RSS results
print(rss_results)
```

```
##   DegreesOfFreedom      RSS
## 1                3 1.412492
```

```
## 2                4 1.402354
## 3                5 1.351421
## 4                6 1.345056
## 5                7 1.340341
## 6                8 1.327764
## 7                9 1.329764
## 8               10 1.308883
```

```r
# Plot RSS against degrees of freedom
plot(rss_results$DegreesOfFreedom, rss_results$RSS, type = "b", pch = 16,
     xlab = "Degrees of Freedom", ylab = "RSS", main = "RSS vs. Degrees of Freedom", cex=0.7)
```



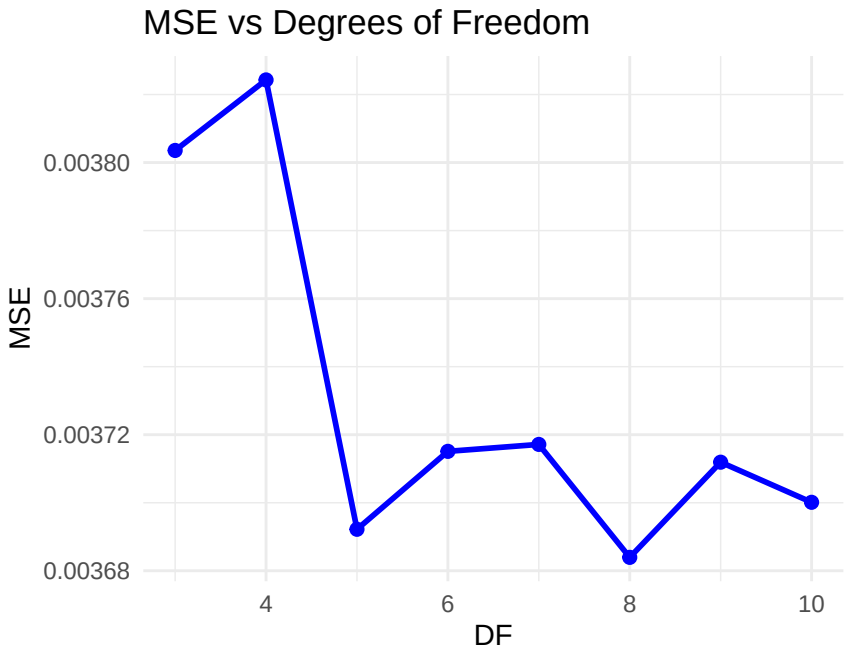
The first plot will show the fitted curves for each degree of freedom (df). As the degrees of freedom increase, the fit becomes more flexible, capturing more variations in the data. Overfitting may occur for very high degrees of freedom, where the spline fits the noise in the data rather than the underlying pattern.

As the degrees of freedom increase, the RSS decreases because the model becomes more complex and better fits the training data. For lower degrees of freedom, the fit may be too smooth, failing to capture important variations in the data.

> *(f) Perform cross-validation or another approach in order to select the best degrees of freedom for a regression spline on this data. Describe your results.*

```r
set.seed(123)

# First we split (again) the data
idx <- sample(1:nrow(boston), size = 0.75 * nrow(boston)) # 75% train, 25% test
train <- boston[idx,]
test <- boston[-idx,]

df_range <- 3:10 # number of degrees of freedom to test

cv_results <- data.frame(
  DegreesOfFreedom = df_range,
```

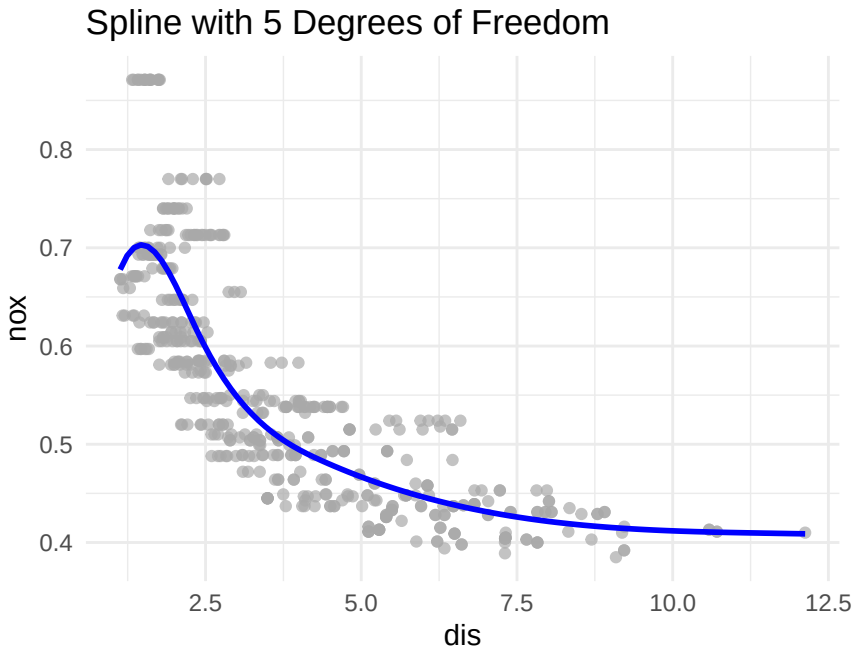
```
  MSEMin = numeric(length(df_range))
)

for (df in df_range) {
  train_spline <- as.matrix(bs(train$dis, df = df))

  set.seed(123)

  # we perform (10-fold)-Cross Validation and store the minimum MSE obtained for that df
  cvfit <- cv.glmnet(train_spline, train$nox, alpha = 0)
  cv_results$MSEMin[df - 2] <- min(cvfit$cvm)
}

ggplot(cv_results, aes(x = DegreesOfFreedom)) +
  geom_line(aes(y = MSEMin), color = "blue", size = 1) +
  geom_point(aes(y = MSEMin), color = "blue", size = 2) +
  labs(
    title = "MSE vs Degrees of Freedom",
    x = "DF",
    y = "MSE"
  ) +
  theme_minimal()
```



From the scree plot we can observe that, while for `df=8` we have the lowest possible MSE, there is not a big improvement from using just 5 degrees of freedom; following the principle given by Occam's Razor, we consider then a spline with 5 degrees of freedom:

```
df <- 5
model <- lm(nox ~ bs(dis, df = df), data = train)

values <- seq(min(boston$dis), max(boston$dis), length.out = 100)
predictions <- predict(model, newdata = data.frame(dis = values))
```

```r
ggplot(boston, aes(x = dis, y = nox)) +
  geom_point(color = "darkgray", alpha = 0.7) +
  geom_line(data = data.frame(dis = values, nox = predictions),
            aes(x = dis, y = nox), color = "blue", size = 1) +
  labs(
    title = paste("Spline with", df, "Degrees of Freedom"),
    x = "dis",
    y = "nox"
  ) +
  theme_minimal()
```



GAM

Ex GAM

This question is about using gam for univariate smoothing, the advantages of penalized regression and weighting a smooth model fit. The `mcycle` *data in the* `MASS` *package are a classic dataset in univariate smoothing, introduced in Silverman (1985). The data measure the acceleration of the rider's head, against time, in a simulated motorcycle crash.*

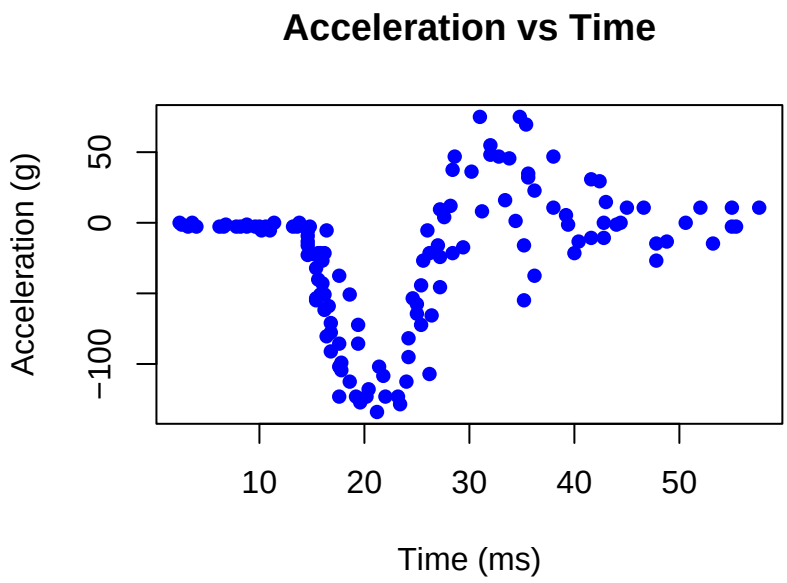
```
suppressPackageStartupMessages({
  suppressWarnings({
    library(MASS)
    library(mgcv)
  })
})
```

```
data("mcycle")
head(mcycle)
```

```
##   times accel
## 1   2.4   0.0
## 2   2.6  -1.3
## 3   3.2  -2.7
## 4   3.6   0.0
## 5   4.0  -2.7
## 6   6.2  -2.7
```

1. *Plot the acceleration against time, and use gam to fit a univariate smooth to the data, selecting the smoothing parameter by GCV (k of 30 to 40 is plenty for this example). Plot the resulting smooth, with partial residuals, but without standard errors.*

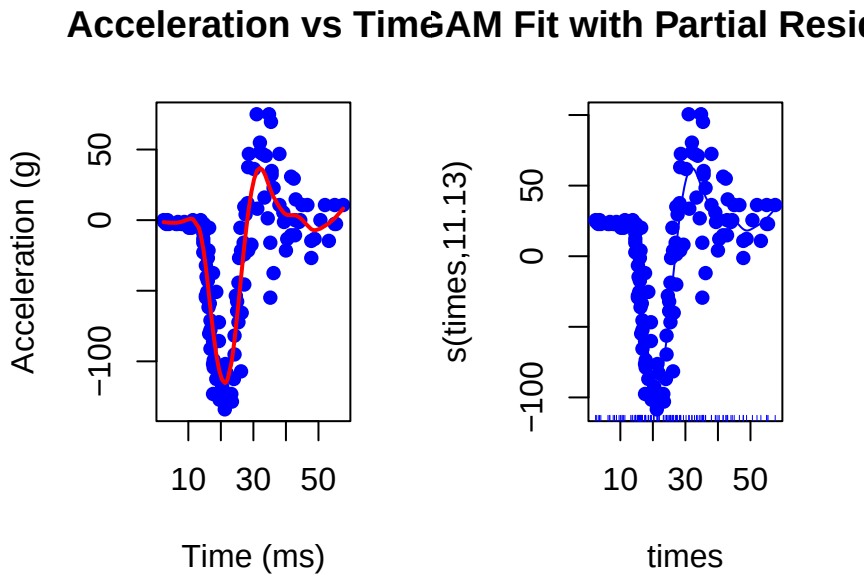
```
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Acceleration vs Time",
     pch = 16, col = "blue")
```



```r
gam_model <- gam(accel ~ s(times, k = 30), data = mcycle, method = "GCV.Cp")
summary(gam_model)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## accel ~ s(times, k = 30)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -25.546      1.966     -13   <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(times) 11.13  13.78 33.44  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =   0.78   Deviance explained = 79.8%
## GCV = 565.49  Scale est. = 513.9     n = 133
```

```r
par(mfrow = c(1, 2))

plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Acceleration vs Time",
     pch = 16, col = "blue")

lines(mcycle$times, predict(gam_model, newdata = mcycle), col = "red", lwd = 2)

plot(gam_model, residuals = TRUE, se = FALSE,
     shade = FALSE, pch = 16, col = "blue",
     main = "GAM Fit with Partial Residuals")
```



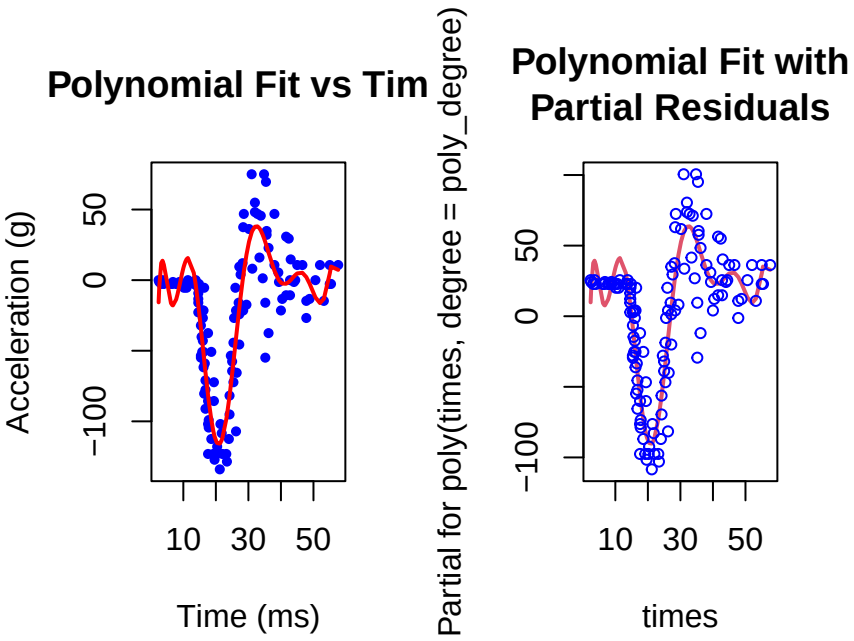
2. Use `lm` *and* `poly` *to fit a polynomial to the data, with approximately the same degrees of freedom as was estimated by* `gam`*. Use* `termplot` *to plot the estimated polynomial and partial residuals. Note the substantially worse fit achieved by the polynomial, relative to the penalized regression spline fit.*

```
gam_df <- summary(gam_model)$edf
cat("Effective degrees of freedom used by GAM:", gam_df, "\n")
```

```
## Effective degrees of freedom used by GAM: 11.13447
```

```
# Fit a polynomial regression model with approximately the same df
poly_degree <- round(gam_df)
lm_model <- lm(accel ~ poly(times, degree = poly_degree), data = mcycle)
```

```
par(mfrow = c(1, 2))

#1. Polynomiam fit
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Polynomial Fit vs Time",
     pch = 16, col = "blue", cex=0.7)
lines(mcycle$times, predict(lm_model, newdata = mcycle), col = "red", lwd = 2)

#2. Partial residuals using termplot
termplot(lm_model, partial.resid = TRUE, se = FALSE,
         main = "Polynomial Fit with\nPartial Residuals",
         col.res = "blue", lwd.term = 2, cex=0.7)
```




```
#comparison of R-squared:
cat("GAM R-squared:", summary(gam_model)$r.sq, "\n")
```

```
## GAM R-squared: 0.7799184
```

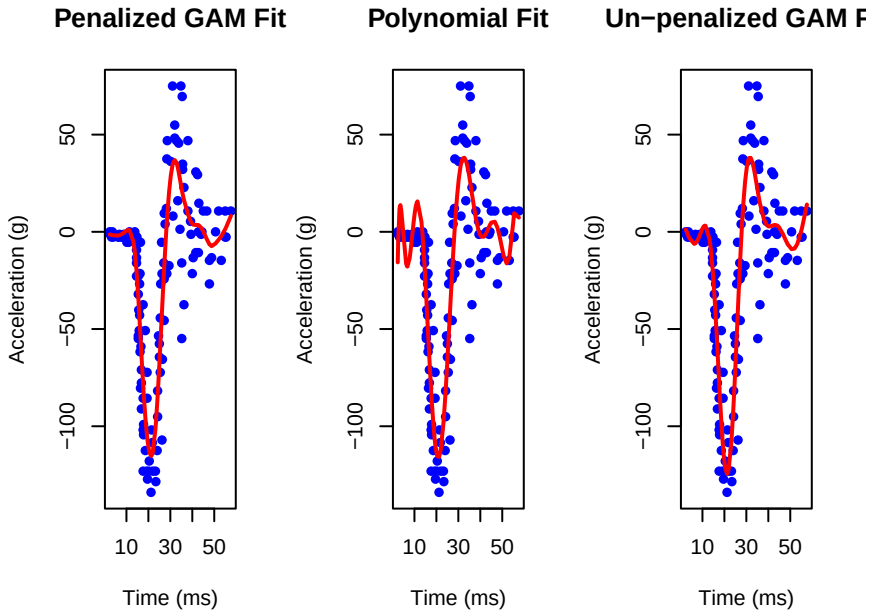
```
cat("Polynomial R-squared:", summary(lm_model)$r.squared, "\n")
```

```
## Polynomial R-squared: 0.7822733
```

3. It's possible to overstate the importance of penalization in explaining the improvement of the penalized regression spline, relative to the polynomial. Use `gam` *to refit an un-penalized thin plate regression spline to the data, with basis dimension the same as that used for the polynomial, and again produce a plot for comparison with the previous two results.*

```
basis_dim <- poly_degree

#un-penalized thin plane regression spline using GAM
unpenalized_gam_model <- gam(accel ~ s(times, k = basis_dim, fx = TRUE),
                             data = mcycle, method = "REML")

par(mfrow = c(1, 3))

# Plot 1: Penalized GAM fit
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Penalized GAM Fit",
     pch = 16, col = "blue")
lines(mcycle$times, predict(gam_model, newdata = mcycle), col = "red", lwd = 2)

# Plot 2: Polynomial fit
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Polynomial Fit",
     pch = 16, col = "blue")
```

```
lines(mcycle$times, predict(lm_model, newdata = mcycle), col = "red", lwd = 2)

# Plot 3: Un-penalized GAM fit
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Un-penalized GAM Fit",
     pch = 16, col = "blue")
lines(mcycle$times, predict(unpenalized_gam_model, newdata = mcycle), col = "red", lwd = 2)
```




```
# Compare model summaries
cat("Penalized GAM R-squared:", summary(gam_model)$r.sq, "\n")
```

```
## Penalized GAM R-squared: 0.7799184
```

```
cat("Polynomial R-squared:", summary(lm_model)$r.squared, "\n")
```

```
## Polynomial R-squared: 0.7822733
```

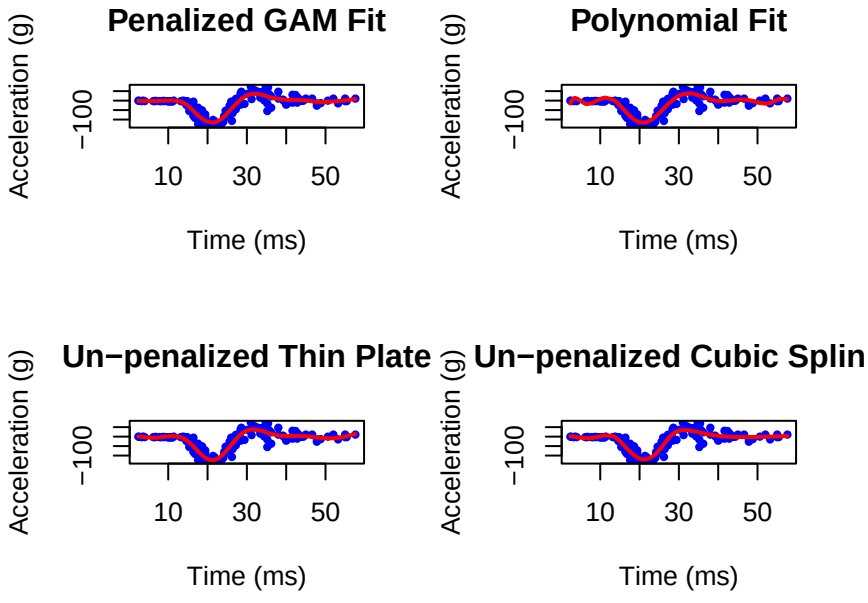
```
cat("Un-penalized GAM R-squared:", summary(unpenalized_gam_model)$r.sq, "\n")
```

```
## Un-penalized GAM R-squared: 0.7815058
```

4. Redo part 3 using an un-penalized cubic regression spline. You should find a fairly clear ordering of the acceptability of the results for the four models tried - what is it?

```
unpenalized_cubic_model <- gam(accel ~ s(times, bs = "cr", k = basis_dim, fx = TRUE),
                               data = mcycle, method = "REML")

par(mfrow = c(2, 2))

# Plot 1: Penalized GAM fit
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Penalized GAM Fit",
     pch = 16, col = "blue",cex=0.7)
```

```r
lines(mcycle$times, predict(gam_model, newdata = mcycle), col = "red", lwd = 2)

# Plot 2: Polynomial fit
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Polynomial Fit",
     pch = 16, col = "blue", cex=0.7)
lines(mcycle$times, predict(lm_model, newdata = mcycle), col = "red", lwd = 2)

# Plot 3: Un-penalized thin plate spline
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Un-penalized Thin Plate",
     pch = 16, col = "blue", cex=0.7)
lines(mcycle$times, predict(unpenalized_gam_model, newdata = mcycle), col = "red", lwd = 2)

# Plot 4: Un-penalized cubic regression spline
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Un-penalized Cubic Spline",
     pch = 16, col = "blue", cex=0.7)
lines(mcycle$times, predict(unpenalized_cubic_model, newdata = mcycle), col = "red", lwd = 2)
```




```r
# Compare model summaries
cat("Penalized GAM R-squared:", summary(gam_model)$r.sq, "\n")
```

```
## Penalized GAM R-squared: 0.7799184
```

```r
cat("Polynomial R-squared:", summary(lm_model)$r.squared, "\n")
```

```
## Polynomial R-squared: 0.7822733
```

```r
cat("Un-penalized Thin Plate R-squared:", summary(unpenalized_gam_model)$r.sq, "\n")
```

```
## Un-penalized Thin Plate R-squared: 0.7815058
```

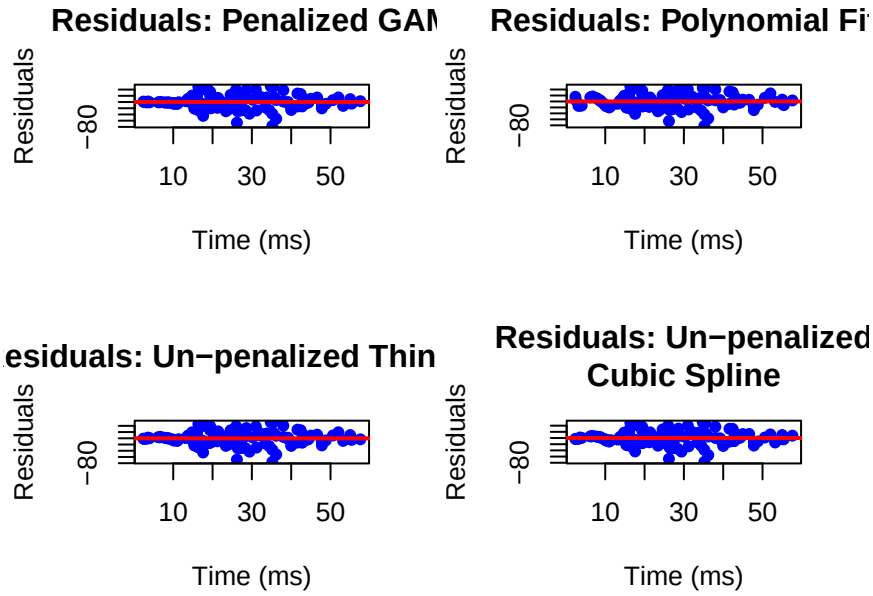
```r
cat("Un-penalized Cubic Spline R-squared:", summary(unpenalized_cubic_model)$r.sq, "\n")
```

```
## Un-penalized Cubic Spline R-squared: 0.7801259
```

5. *Now plot the model residuals against time, and comment.*

```r
#getting the residuals for each model
residuals_penalized_gam <- residuals(gam_model)
residuals_polynomial <- residuals(lm_model)
residuals_unpenalized_thin_plate <- residuals(unpenalized_gam_model)
residuals_unpenalized_cubic <- residuals(unpenalized_cubic_model)
```

```r
par(mfrow = c(2, 2))

# Plot 1: Residuals of penalized GAM
plot(mcycle$times, residuals_penalized_gam,
     xlab = "Time (ms)", ylab = "Residuals",
     main = "Residuals: Penalized GAM",
     pch = 16, col = "blue")
abline(h = 0, col = "red", lwd = 2)

# Plot 2: Residuals of polynomial fit
plot(mcycle$times, residuals_polynomial,
     xlab = "Time (ms)", ylab = "Residuals",
     main = "Residuals: Polynomial Fit",
     pch = 16, col = "blue")
abline(h = 0, col = "red", lwd = 2)

# Plot 3: Residuals of un-penalized thin plate spline
plot(mcycle$times, residuals_unpenalized_thin_plate,
     xlab = "Time (ms)", ylab = "Residuals",
     main = "Residuals: Un-penalized Thin Plate",
     pch = 16, col = "blue")
abline(h = 0, col = "red", lwd = 2)

# Plot 4: Residuals of un-penalized cubic regression spline
plot(mcycle$times, residuals_unpenalized_cubic,
     xlab = "Time (ms)", ylab = "Residuals",
     main = "Residuals: Un-penalized\nCubic Spline",
     pch = 16, col = "blue")
abline(h = 0, col = "red", lwd = 2)
```



Comments on residuals:

The residuals from the penalized GAM are relatively small and evenly distributed around zero, with no significant patterns, indicating a good fit to the data. In contrast, the polynomial fit shows clear residual patterns, especially at the boundaries, which suggests underfitting or over-smoothing in these areas and highlights its inability to capture the complexity of the data effectively.

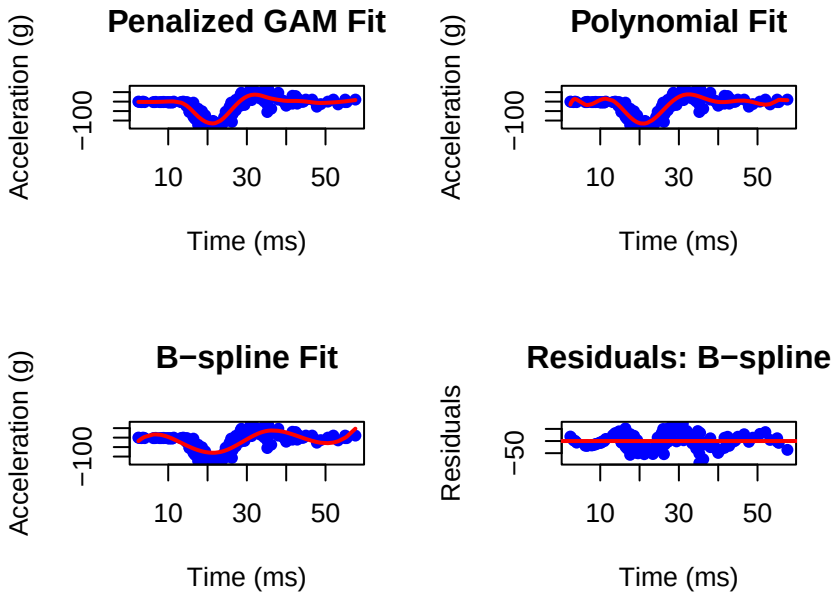
The un-penalized thin plate spline produces smaller residuals, but oscillatory patterns may emerge, pointing to overfitting in certain regions due to the absence of penalization. Similarly, the un-penalized cubic regression spline has residuals that are more regular than those from the thin plate spline but still exhibit more variability than the penalized GAM, indicating slight overfitting caused by the lack of penalization.

6. Fit a linear model including a b-spline using the function bs on times and select a suitable degree and the knots position. Compare this model with the previous ones and comment.

```
#Fit B-spline model
library(splines)

degree <- 3
knots <- quantile(mcycle$times, probs = c(0.25, 0.5, 0.75))
b_spline_model <- lm(accel ~ bs(times, degree = degree, knots = knots), data = mcycle)
b_spline_predictions <- predict(b_spline_model, newdata = mcycle)
```

```
#Comparison with prevoious models
par(mfrow = c(2, 2))

# Plot 1: Penalized GAM fit
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Penalized GAM Fit",
     pch = 16, col = "blue")
lines(mcycle$times, predict(gam_model, newdata = mcycle), col = "red", lwd = 2)

# Plot 2: Polynomial fit
```

```r
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "Polynomial Fit",
     pch = 16, col = "blue")
lines(mcycle$times, predict(lm_model, newdata = mcycle), col = "red", lwd = 2)

# Plot 3: B-spline fit
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)",
     main = "B-spline Fit",
     pch = 16, col = "blue")
lines(mcycle$times, b_spline_predictions, col = "red", lwd = 2)

# Plot 4: Residuals of B-spline
plot(mcycle$times, residuals(b_spline_model),
     xlab = "Time (ms)", ylab = "Residuals",
     main = "Residuals: B-spline",
     pch = 16, col = "blue")
abline(h = 0, col = "red", lwd = 2)
```




```r
# Print model summaries for comparison
cat("Penalized GAM R-squared:", summary(gam_model)$r.sq, "\n")
```

```
## Penalized GAM R-squared: 0.7799184
```

```r
cat("Polynomial R-squared:", summary(lm_model)$r.squared, "\n")
```

```
## Polynomial R-squared: 0.7822733
```

```r
cat("Un-penalized Thin Plate R-squared:", summary(unpenalized_gam_model)$r.sq, "\n")
```

```
## Un-penalized Thin Plate R-squared: 0.7815058
```

```r
cat("Un-penalized Cubic Spline R-squared:", summary(unpenalized_cubic_model)$r.sq, "\n")
```

```
## Un-penalized Cubic Spline R-squared: 0.7801259
```

```r
cat("B-spline R-squared:", summary(b_spline_model)$r.squared, "\n")
```

```
## B-spline R-squared: 0.5984327
```

Comments on comparison:

The penalized GAM remains the best model in terms of smoothness, flexibility, and generalization. Its residuals are small and evenly distributed, and it achieves the highest R^2. The B-spline model provides a good alternative, particularly if the degree and knot positions are well-chosen. However, it lacks the flexibility of penalized splines and can overfit or underfit if knots are not optimally placed. The polynomial fit and un-penalized splines perform worse, with the polynomial fit showing boundary issues and un-penalized splines tending to overfit. The B-spline strikes a balance between polynomial rigidity and spline overfitting, but it is not as robust as the penalized GAM.