Econ 573: Problem Set 4

Juan M. Alvarez

Part I

Exercise 3

a. First, we divide the dataset into k folds of approximately equal size. Then, for each $k = 1, \ldots, K$, we repeat the following process: we leave out the k^{th} fold as the validation set and train the model on the remaining $k - 1$ folds. We then compute the mean squared error (MSE) on the held-out k^{th} fold. This process results in k estimates of the test error: $\text{MSE}_1, \ldots, \text{MSE}_K$. The final k-fold cross-validation estimate is obtained by averaging these values.

b.

(i)The test error estimates that result from the k-fold CV approach typically have much lower variability than the results from the validation set approach. Additionally, because the validation set approach only uses half of the observations in the data set when fitting the learning method, the test error estimates for the validation approach have substantially more bias than those from the k-fold CV approach.

(ii) The main advantage of the k-fold CV approach over the LOOCV approach is that the latter is computational expensive as it has to fit the method n times. Since some statistical learning methods have computationally intensive fitting procedures, LOOCV could pose computational problems. In contrast, k-fold cross-validation can be applied to almost any statistical learning method. So, as long as $k < n$ the k-fold CV approach has a computational advantage to LOOCV. Another advantage of this approach is that it often gives more accurate estimates of the test error rate than the LOOCV approach. On the other hand, since each training method contains $n - 1$ observations, the LOOCV approach gives almost unbiased estimates of the test error, while k-fold CV does have some bias to it. However, because the LOOCV approach trains it's models with almost identical sets of observations, the outputs are highly correlated with each other. Such that the estimates from LOOCV has a higher variance than the k-fold CV approach.

Exercise 5

```
default_data <- read.csv("C:\\Users\\mateo\\OneDrive\\Escritorio\\Default.csv")

default_data$default <- as.factor(default_data$default)
default_data$student <- as.factor(default_data$student)

View(default_data)

set.seed(1)
```

a.

```
# Fitting a logistic regression model that uses income and balance to predict default.
default_predict <- glm(default ~ income + balance, data = default_data, family = "binomial")

summary(default_predict)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = default_data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

b.

```
# i. Splitting the set into a training set and a validation set.

n <- nrow(default_data)

train_indices <- sample(1:n, size = n/2)
```

```
#ii. Fitting a multiple logistic regression model using only the training observations.

training_default <- glm(default ~ income + balance, data = default_data, subset = train_indices,
family = "binomial")
```

```
names(default_data)
```

```
## [1] "default" "student" "balance" "income"
```

```
# iii.
probs <- predict(training_default, newdata = default_data[-train_indices, ], type = "response")
predictions <- ifelse(probs > 0.5, "Yes", "No")
```

```
# iv.
actual <- default_data$default[-train_indices]

test_error <- mean(predictions != actual)
test_error
```

```
## [1] 0.0254
```

c.

```
# Function to compute validation error given a seed

get_validation_error <- function(seed) {

  set.seed(seed)
  n <- nrow(default_data)
  train_indices <- sample(1:n, size = n/2)

  model <- glm(default ~ income + balance, data = default_data, subset = train_indices, family =
"binomial")

  validation_set <- default_data[-train_indices, ]
  probs <- predict(model, newdata = validation_set, type = "response")
  predictions <- ifelse(probs > 0.5, "Yes", "No")
  actual <- validation_set$default

  mean(predictions != actual)
}
```

```
error2 <- get_validation_error(2)
error3 <- get_validation_error(3)
error4 <- get_validation_error(4)

c(error2, error3, error4)
```

```
## [1] 0.0238 0.0264 0.0256
```

```
mean(c(error2, error3, error4))
```

```
## [1] 0.02526667
```

The test error rates for the three iterations were approximately 2.38%, 2.64%, and 2.56%, giving an average error of 2.53%. These results indicate that the model performs consistently well across different random splits of the data.

d.

```
set.seed(1)
n <- nrow(default_data)
train_indices <- sample(1:n, size = n/2)

model_student <- glm(default ~ income + balance + student, data = default_data, subset = train_i
ndices, family = "binomial")

validation_set <- default_data[-train_indices, ]

probs_student <- predict(model_student, newdata = validation_set, type = "response")

predictions_student <- ifelse(probs_student > 0.5, "Yes", "No")

actual_student <- validation_set$default

test_error_student <- mean(predictions_student != actual_student)

test_error_student
```

```
## [1] 0.026
```

Compared to our other model, which didn't include the predictor "student" in the regression and had a text error of approximately 2,53%, this model shows a slightly higher test error. In conclusion, including student in the model didn't reduce the test error, but rather slightly increase it. This suggests that whether or not a person is a student doesn't provide additional predictive value for default when income and balance are included as predictors in the model.

Exercise 6

```
set.seed(1)

glm_fit <- glm(default ~ income + balance, data = default_data, family = "binomial")

summary(glm_fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = default_data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

The estimated standard for the income coefficient is $4.985e-06$, while for the balance coefficient it is $2.274e-04$.

b.

```
library(boot)
```

```
## Warning: package 'boot' was built under R version 4.4.3
```

```
boot.fn <- function(data, index) {
  fit <- glm(default ~ income + balance, data = data, subset = index, family = "binomial")
  return(coef(fit)[c("income", "balance")])
}
```

c.

```
set.seed(1)

boot_results <- boot(data = default_data, statistic = boot.fn, R = 1000)

boot_results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = default_data, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##         original       bias     std. error
## t1* 2.080898e-05 1.680317e-07 4.866284e-06
## t2* 5.647103e-03 1.855765e-05 2.298949e-04
```

d.

```
glm_se <- summary(glm_fit)$coefficients[c("income", "balance"), "Std. Error"]

bootstrap_se <- boot_results$t0  # point estimates
bootstrap_sd <- apply(boot_results$t, 2, sd)  # standard errors

se_comparison <- data.frame(
  Coefficient = c("Income", "Balance"),
  SE_GLM = glm_se,
  SE_Bootstrap = bootstrap_sd
)
se_comparison
```

	Coefficient <chr>	**SE_GLM** <dbl>	**SE_Bootstrap** <dbl>
income	Income	4.985167e-06	4.866284e-06
balance	Balance	2.273731e-04	2.298949e-04

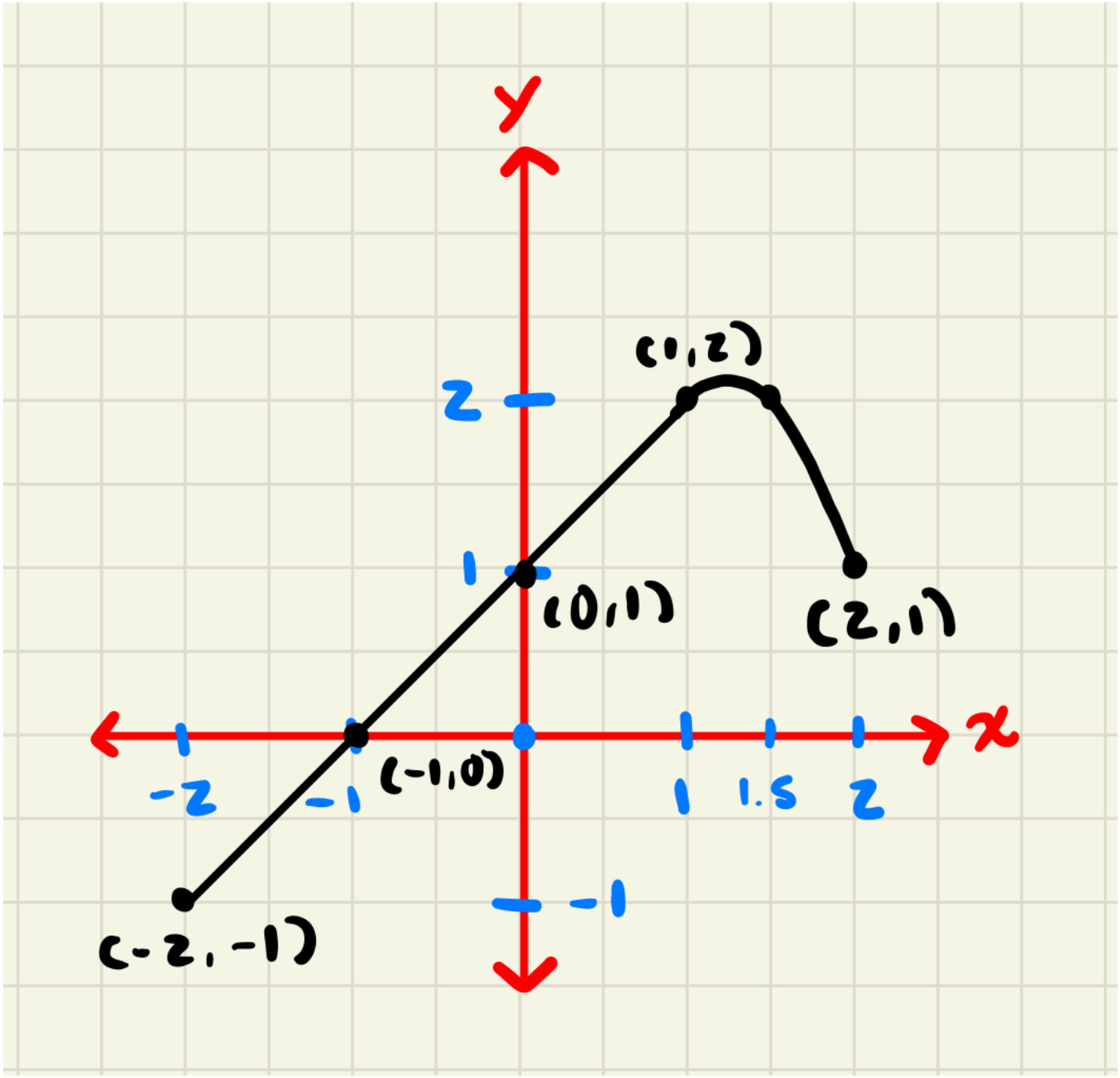
2 rows

The bootstrap estimates are $4.866284e-06$ for Income, and $2.2989949e-04$. These values are very close to the standard errors reported by glm(), suggesting that the model-based estimates are accurate and the logistic regression assumptions hold well for this data.

Part II

Exercise 3

```
knitr::include_graphics("C:/Users/mateo/Downloads/1F5BB6F7-EF21-46F1-9213-BA13C52B8B7C.jpeg")
```



Exercise 9

```
boston_data <- read.csv("C:/Users/mateo/OneDrive - University of North Carolina at Chapel Hill/C
ourses/Spring 2025/Econ 573/Data/ALL+CSV+FILES+-+2nd+Edition+-+corrected/ALL CSV FILES - 2nd Edi
tion/Boston.csv")
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.4.3
```

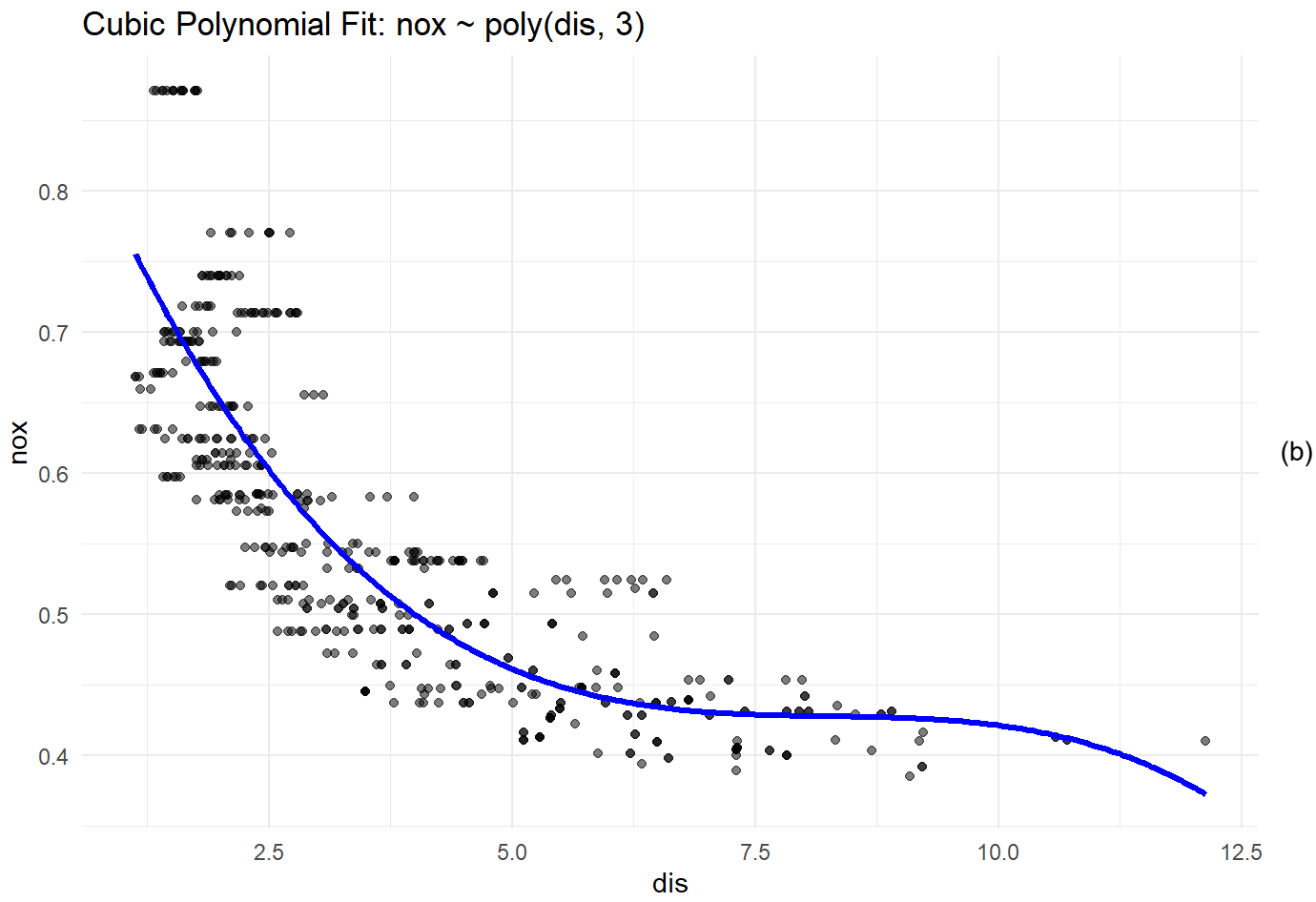
a.

```
fit_poly3 <- lm(nox ~ poly(dis, 3), data = boston_data)

summary(fit_poly3)
```

```
##
## Call:
## lm(formula = nox ~ poly(dis, 3), data = boston_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.121130 -0.040619 -0.009738  0.023385  0.194904
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.554695   0.002759 201.021  < 2e-16 ***
## poly(dis, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
## poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
## poly(dis, 3)3 -0.318049   0.062071  -5.124 4.27e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06207 on 502 degrees of freedom
## Multiple R-squared:  0.7148, Adjusted R-squared:  0.7131
## F-statistic: 419.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

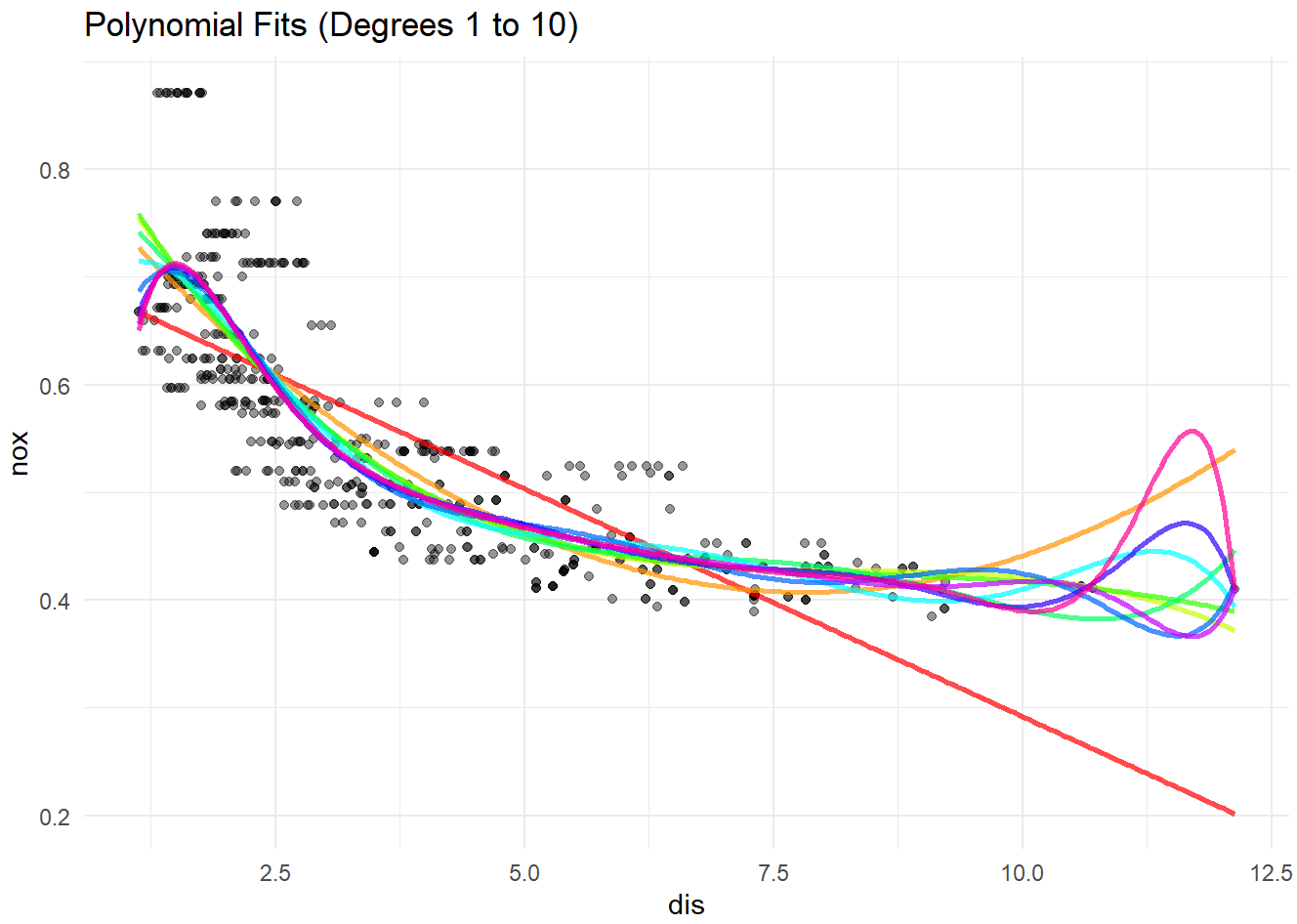
```
dis_grid <- seq(min(boston_data$dis), max(boston_data$dis), length.out = 300)
predictions <- predict(fit_poly3, newdata = data.frame(dis = dis_grid))

pred_df <- data.frame(dis = dis_grid, nox = predictions)

ggplot(boston_data, aes(x = dis, y = nox)) +
  geom_point(alpha = 0.5) +
  geom_line(data = pred_df, aes(x = dis, y = nox), color = "blue", size = 1.2) +
  labs(title = "Cubic Polynomial Fit: nox ~ poly(dis, 3)",
       x = "dis", y = "nox") +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



(b)

```
library(ggplot2)

rss_values <- numeric(10)

dis_grid <- seq(min(boston_data$dis), max(boston_data$dis), length.out = 300)

plot_base <- ggplot(boston_data, aes(x = dis, y = nox)) +
  geom_point(alpha = 0.4) +
  labs(x = "dis", y = "nox", title = "Polynomial Fits (Degrees 1 to 10)") +
  theme_minimal()

for (d in 1:10) {
  fit <- lm(nox ~ poly(dis, d), data = boston_data)
  predictions <- predict(fit, newdata = data.frame(dis = dis_grid))
  rss <- sum((predict(fit) - boston_data$nox)^2)
  rss_values[d] <- rss

  pred_df <- data.frame(dis = dis_grid, nox = predictions)
  plot_base <- plot_base +
    geom_line(data = pred_df, aes(x = dis, y = nox),
              color = rainbow(10)[d], size = 1, alpha = 0.7)
}

print(plot_base)
```




```
rss_table <- data.frame(Degree = 1:10, RSS = round(rss_values, 5))
print(rss_table)
```

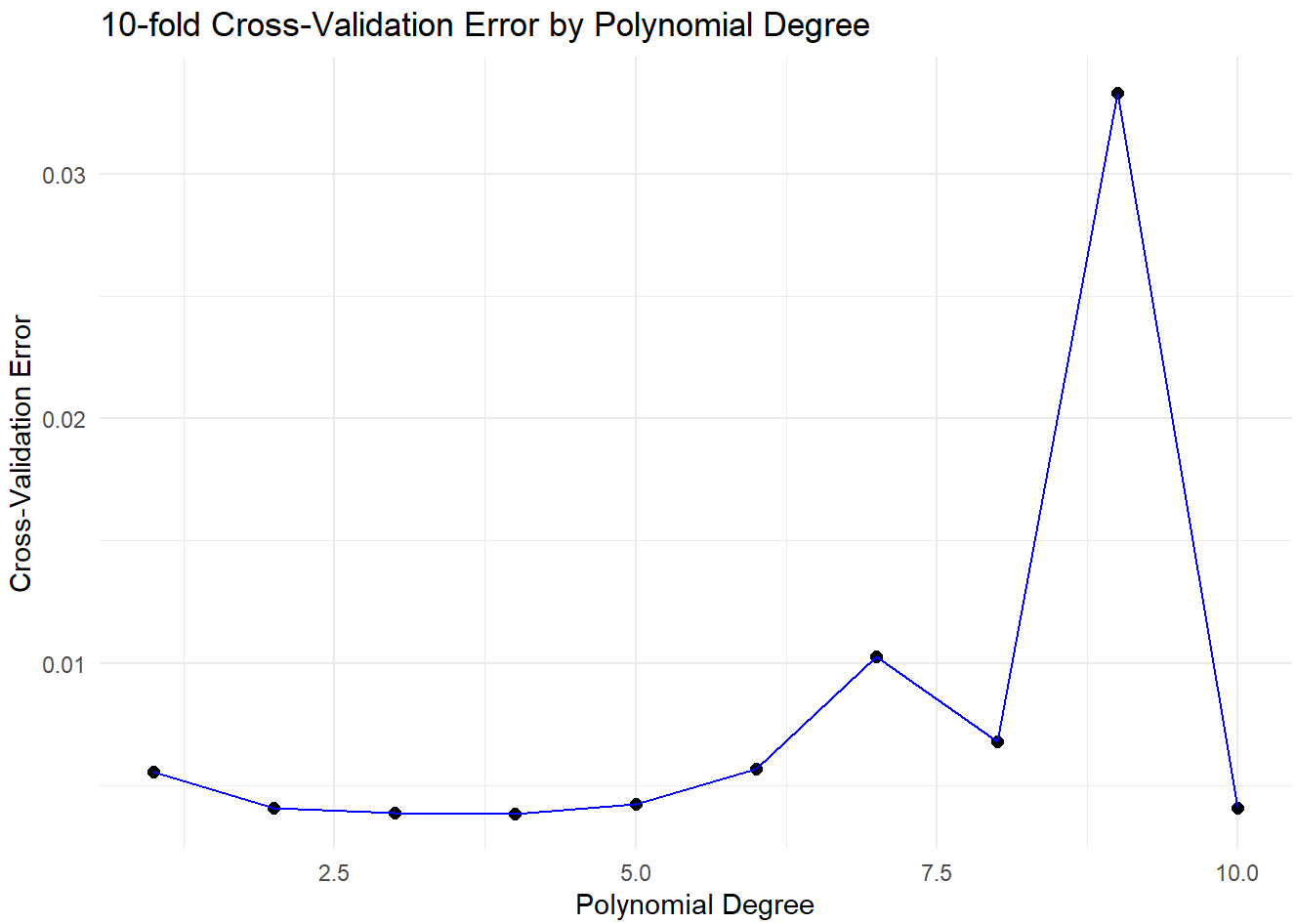
```
##    Degree     RSS
## 1       1 2.76856
## 2       2 2.03526
## 3       3 1.93411
## 4       4 1.93298
## 5       5 1.91529
## 6       6 1.87826
## 7       7 1.84948
## 8       8 1.83563
## 9       9 1.83333
## 10     10 1.83217
```

c.

```r
library(boot)

cv_errors <- rep(NA, 10)

set.seed(1)
for (d in 1:10) {
  fit <- glm(nox ~ poly(dis, d), data = boston_data)
  cv_result <- cv.glm(boston_data, fit, K = 10)
  cv_errors[d] <- cv_result$delta[1]
}

cv_results <- data.frame(Degree = 1:10, CV_Error = round(cv_errors, 5))
print(cv_results)
```

```
##    Degree CV_Error
## 1       1  0.00556
## 2       2  0.00409
## 3       3  0.00388
## 4       4  0.00386
## 5       5  0.00424
## 6       6  0.00569
## 7       7  0.01028
## 8       8  0.00681
## 9       9  0.03331
## 10     10  0.00408
```

```r
library(ggplot2)
ggplot(cv_results, aes(x = Degree, y = CV_Error)) +
  geom_point(size = 2) +
  geom_line(group = 1, color = "blue") +
  labs(title = "10-fold Cross-Validation Error by Polynomial Degree",
       x = "Polynomial Degree", y = "Cross-Validation Error") +
  theme_minimal()
```



The results show that the cross-validation error decreases initially and stabilizes around degrees 2 to 4, indicating good model fit.

Higher-degree polynomials (especially degree 9) show a sharp increase in error, suggesting overfitting. Therefore, based on the cross-validation results, a degree-3 polynomial provides a good balance between model complexity and predictive performance.

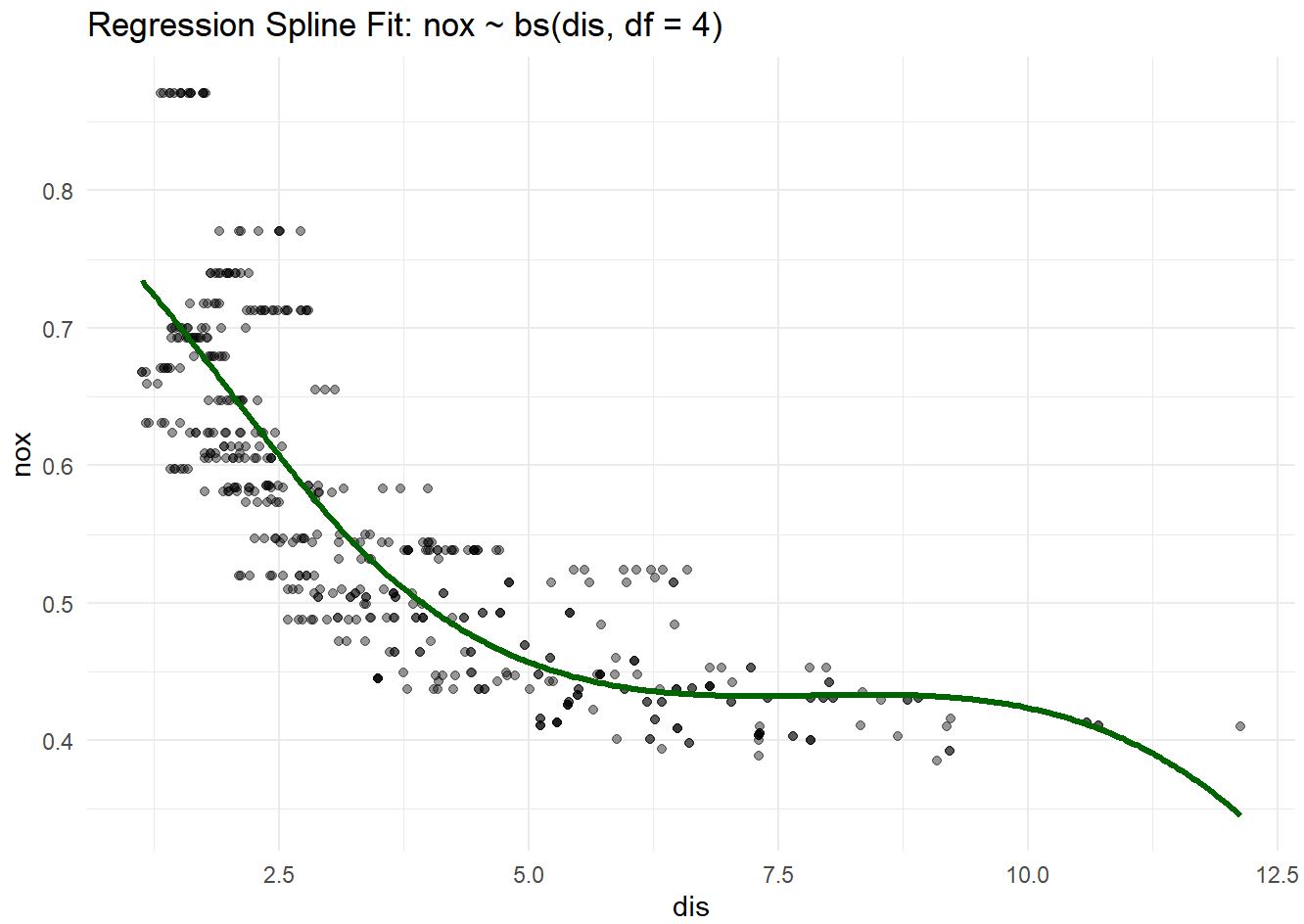
d.

```
library(splines)

spline_fit <- lm(nox ~ bs(dis, df = 4), data = boston_data)

summary(spline_fit)
```

```
##
## Call:
## lm(formula = nox ~ bs(dis, df = 4), data = boston_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.124622 -0.039259 -0.008514  0.020850  0.193891
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.73447    0.01460  50.306  < 2e-16 ***
## bs(dis, df = 4)1 -0.05810    0.02186  -2.658  0.00812 **
## bs(dis, df = 4)2 -0.46356    0.02366 -19.596  < 2e-16 ***
## bs(dis, df = 4)3 -0.19979    0.04311  -4.634 4.58e-06 ***
## bs(dis, df = 4)4 -0.38881    0.04551  -8.544  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06195 on 501 degrees of freedom
## Multiple R-squared:  0.7164, Adjusted R-squared:  0.7142
## F-statistic: 316.5 on 4 and 501 DF,  p-value: < 2.2e-16
```

The bs() function chooses evenly spaced quantile-based knots by default when we set the degrees of freedom to 4.

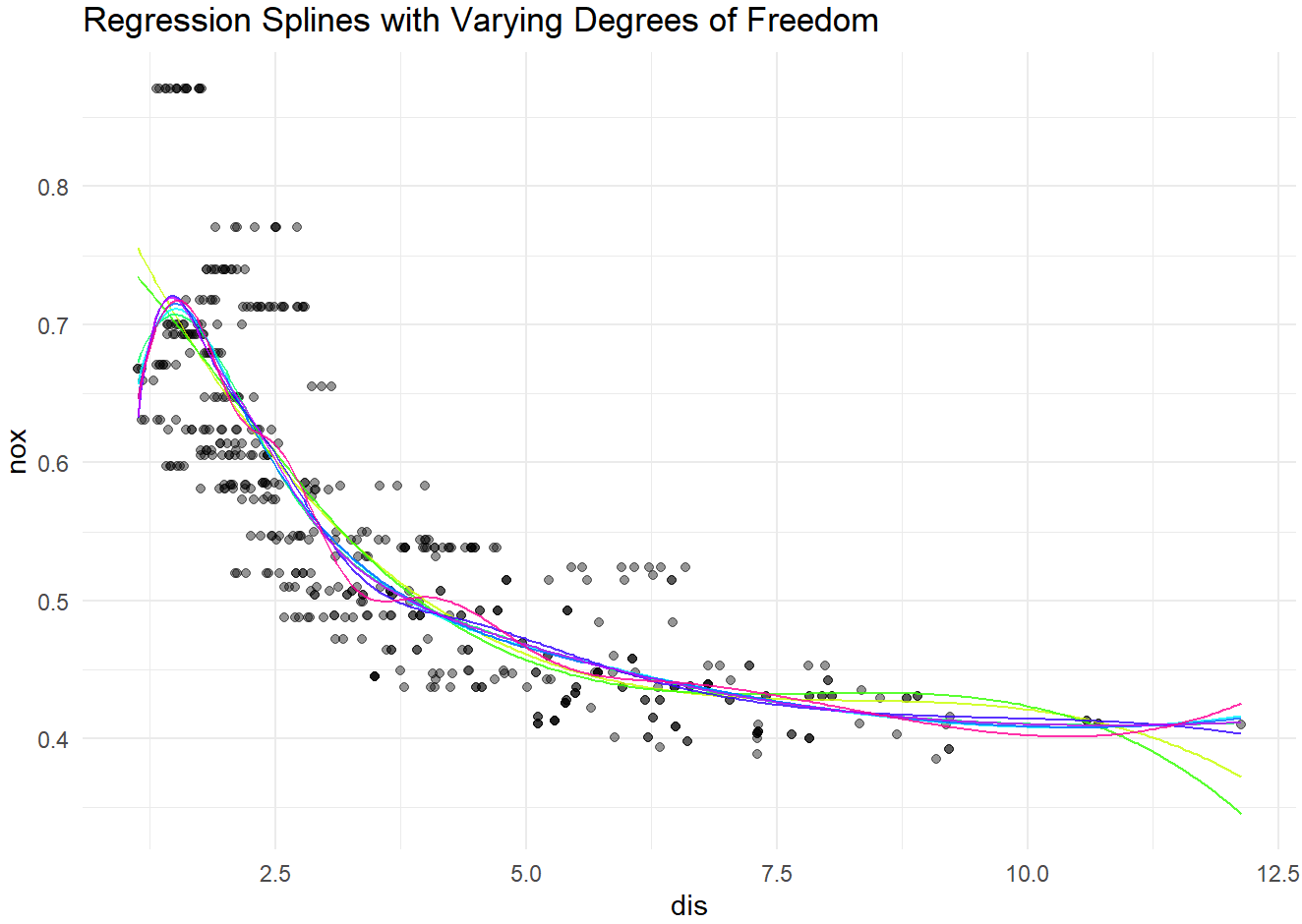
```
dis_grid <- seq(min(boston_data$dis), max(boston_data$dis), length.out = 300)
predictions <- predict(spline_fit, newdata = data.frame(dis = dis_grid))
pred_df <- data.frame(dis = dis_grid, nox = predictions)

ggplot(boston_data, aes(x = dis, y = nox)) +
  geom_point(alpha = 0.4) +
  geom_line(data = pred_df, aes(x = dis, y = nox), color = "darkgreen", size = 1.2) +
  labs(title = "Regression Spline Fit: nox ~ bs(dis, df = 4)",
       x = "dis", y = "nox") +
  theme_minimal()
```



The spline placed knots at quantiles of dis, adapting the shape of the curve to the data. The resulting regression spline shows a clear non-linear decline in nox with increasing dis, fitting the data more closely than a simple linear or low-degree polynomial model.

e.

```
dis_grid <- seq(min(boston_data$dis), max(boston_data$dis), length.out = 300)
rss_spline <- numeric()

plot_base <- ggplot(boston_data, aes(x = dis, y = nox)) +
  geom_point(alpha = 0.4) +
  labs(x = "dis", y = "nox", title = "Regression Splines with Varying Degrees of Freedom") +
  theme_minimal()

for (df_val in 3:10) {
  fit <- lm(nox ~ bs(dis, df = df_val), data = boston_data)

  y_hat <- predict(fit, newdata = boston_data)
  rss <- sum((boston_data$nox - y_hat)^2)
  rss_spline[df_val] <- rss

  pred_grid <- predict(fit, newdata = data.frame(dis = dis_grid))
  pred_df <- data.frame(dis = dis_grid, nox = pred_grid)

  plot_base <- plot_base +
    geom_line(data = pred_df, aes(x = dis, y = nox), color = rainbow(10)[df_val], alpha = 0.8)
}

print(plot_base)
```



```
rss_df <- data.frame(Degrees_of_Freedom = 3:10, RSS = round(rss_spline[3:10], 5))
print(rss_df)
```

```
##   Degrees_of_Freedom     RSS
## 1                  3 1.93411
## 2                  4 1.92277
## 3                  5 1.84017
## 4                  6 1.83397
## 5                  7 1.82988
## 6                  8 1.81700
## 7                  9 1.82565
## 8                 10 1.79253
```

The lowest RSS occurred at df = 10, though improvements diminished after around df = 6–7, suggesting potential diminishing returns. Visual inspection confirms that spline fits with df ≥ 6 track the curvature of the data well without overfitting.

f.

```
library(boot)

cv_errors_spline <- numeric()

set.seed(1)
for (df_val in 3:10) {
  fit <- glm(nox ~ bs(dis, df = df_val), data = boston_data)
  cv_result <- cv.glm(boston_data, fit, K = 10)
  cv_errors_spline[df_val] <- cv_result$delta[1]
}
```

```
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.1296,
## : some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.1296,
## : some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = 3.0993, Boundary.knots = c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = 3.0993, Boundary.knots = c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = 3.3603, Boundary.knots = c(1.1296, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = 3.3603, Boundary.knots = c(1.1296, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(2.38876666666667, 4.3257),
## Boundary.knots = c(1.137, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(2.38876666666667, 4.3257),
## Boundary.knots = c(1.137, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(2.3088, 4.09726666666667),
## Boundary.knots = c(1.1296, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(2.3088, 4.09726666666667),
## Boundary.knots = c(1.1296, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(2.087875, 3.19095, 5.1167),
## Boundary.knots = c(1.137, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(2.087875, 3.19095, 5.1167),
## Boundary.knots = c(1.137, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.92404, 2.55946, 3.6715, 5.4917:
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(1.92404, 2.55946, 3.6715, 5.4917:
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.94984, 2.59774, 3.81326, 5.4917:
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(1.94984, 2.59774, 3.81326, 5.4917:
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.86636666666667, 2.4212, 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(1.86636666666667, 2.4212, 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.82085, 2.36386666666667, 3.2157, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(1.82085, 2.36386666666667, 3.2157, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.79078571428571, 2.16972857142857, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(1.79078571428571, 2.16972857142857, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.7912, 2.1705, 2.7344, 3.6519, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(1.7912, 2.1705, 2.7344, 3.6519, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.757275, 2.1084, 2.53475, 3.2157, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(1.757275, 2.1084, 2.53475, 3.2157, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.76375, 2.10525, 2.522775, 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(1.76375, 2.10525, 2.522775, 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
cv_df <- data.frame(Degrees_of_Freedom = 3:10, CV_Error = round(cv_errors_spline[3:10], 5))
print(cv_df)
```

```
##   Degrees_of_Freedom CV_Error
## 1                  3  0.00390
## 2                  4  0.00389
## 3                  5  0.00372
## 4                  6  0.00370
## 5                  7  0.00374
## 6                  8  0.00371
## 7                  9  0.00373
## 8                 10  0.00372
```

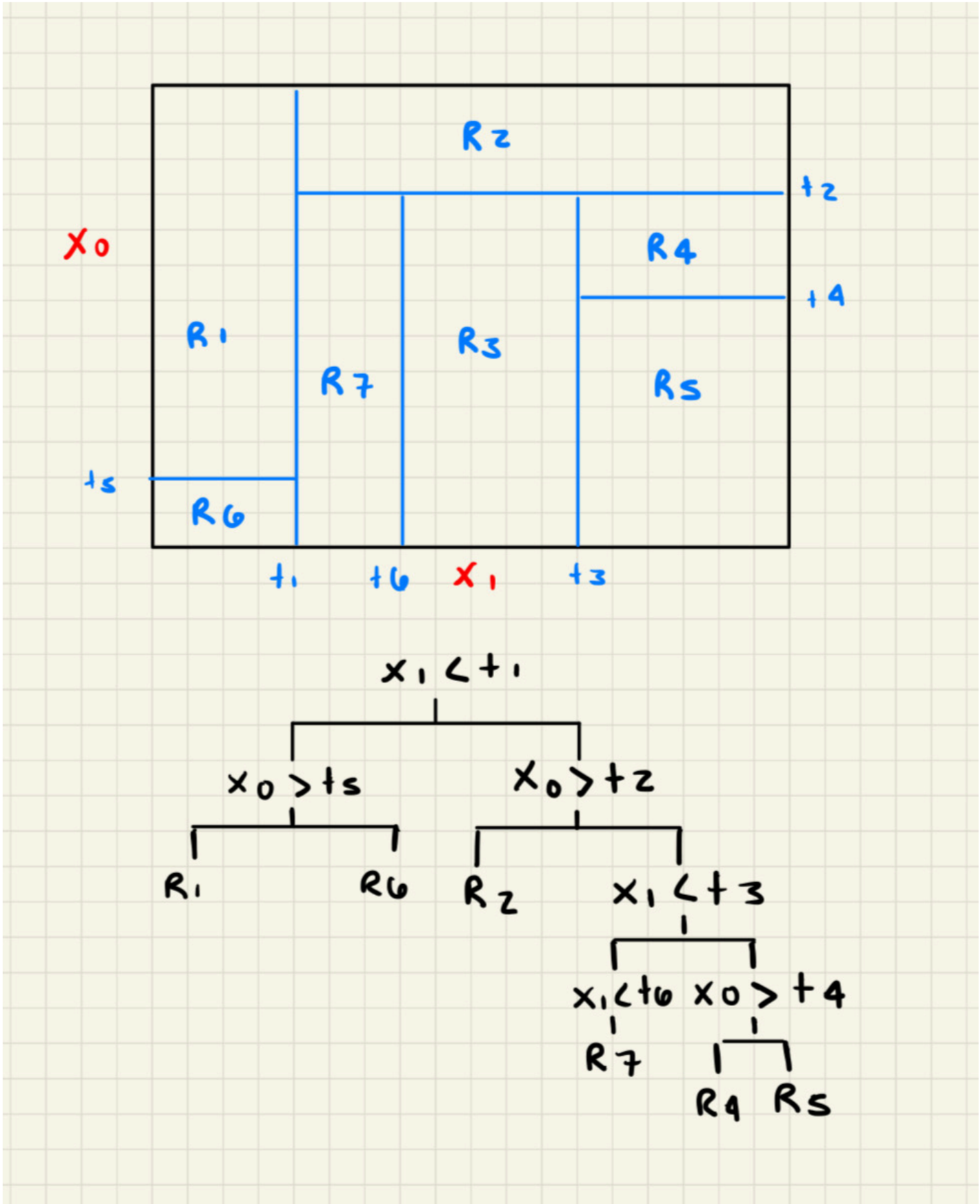
```
library(ggplot2)
ggplot(cv_df, aes(x = Degrees_of_Freedom, y = CV_Error)) +
  geom_point(size = 2) +
  geom_line(group = 1, color = "purple") +
  labs(title = "10-fold Cross-Validation for Regression Splines",
       x = "Degrees of Freedom", y = "Cross-Validation Error") +
  theme_minimal()
```

Based on these results, a regression spline with 6 degrees of freedom provides the best balance between model complexity and predictive performance. This model is flexible enough to capture the non-linear trend in the data without overfitting.

Part III

Exercise 1

```
knitr::include_graphics("C:/Users/mateo/Downloads/IMG_0763.jpeg")
```


R2
t2
X0
R4
t4
R1
R3
R7
R5
t5
R6
t1
t6
X1
t3
X1 < t1
X0 > t5
X0 > t2
R1
R6
R2
X1 < t3
X1 < t6
X0 > t4
R7
R4
R5

Exercise 5

In this example, we have ten bootstrapped probability estimates for the class being Red. Using the majority vote approach, we classify each estimate individually: if the probability is greater than 0.5, we predict Red; otherwise, Green. Since 6 out of the 10 estimates are above 0.5, the majority vote leads to a final classification of Red. In contrast, the average probability approach computes the mean of all 10 probabilities, which equals 0.45. Because this is less than 0.5, the final classification under the average probability method is Green.

Exercise 10

```
hitters_data <- read.csv("C:/Users/mateo/OneDrive - University of North Carolina at Chapel Hill/Courses/Spring 2025/Econ 573/Data/ALL+CSV+FILES+-+2nd+Edition+-+corrected/ALL CSV FILES - 2nd Edition/Hitters.csv")
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.4.3
```

```
##
## Adjuntando el paquete: 'dplyr'
```

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

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

a.

```
hitters_clean <- hitters_data %>%
  filter(!is.na(Salary)) %>%
  mutate(Salary = log(Salary))

summary(hitters_clean$Salary)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.212   5.247   6.052   5.927   6.620   7.808
```

b.

```
train_data <- hitters_clean[1:200, ]
test_data  <- hitters_clean[-(1:200), ]


dim(train_data)
```

```
## [1] 200  20
```

```
dim(test_data)
```

```
## [1] 63 20
```

c.

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.4.3
```

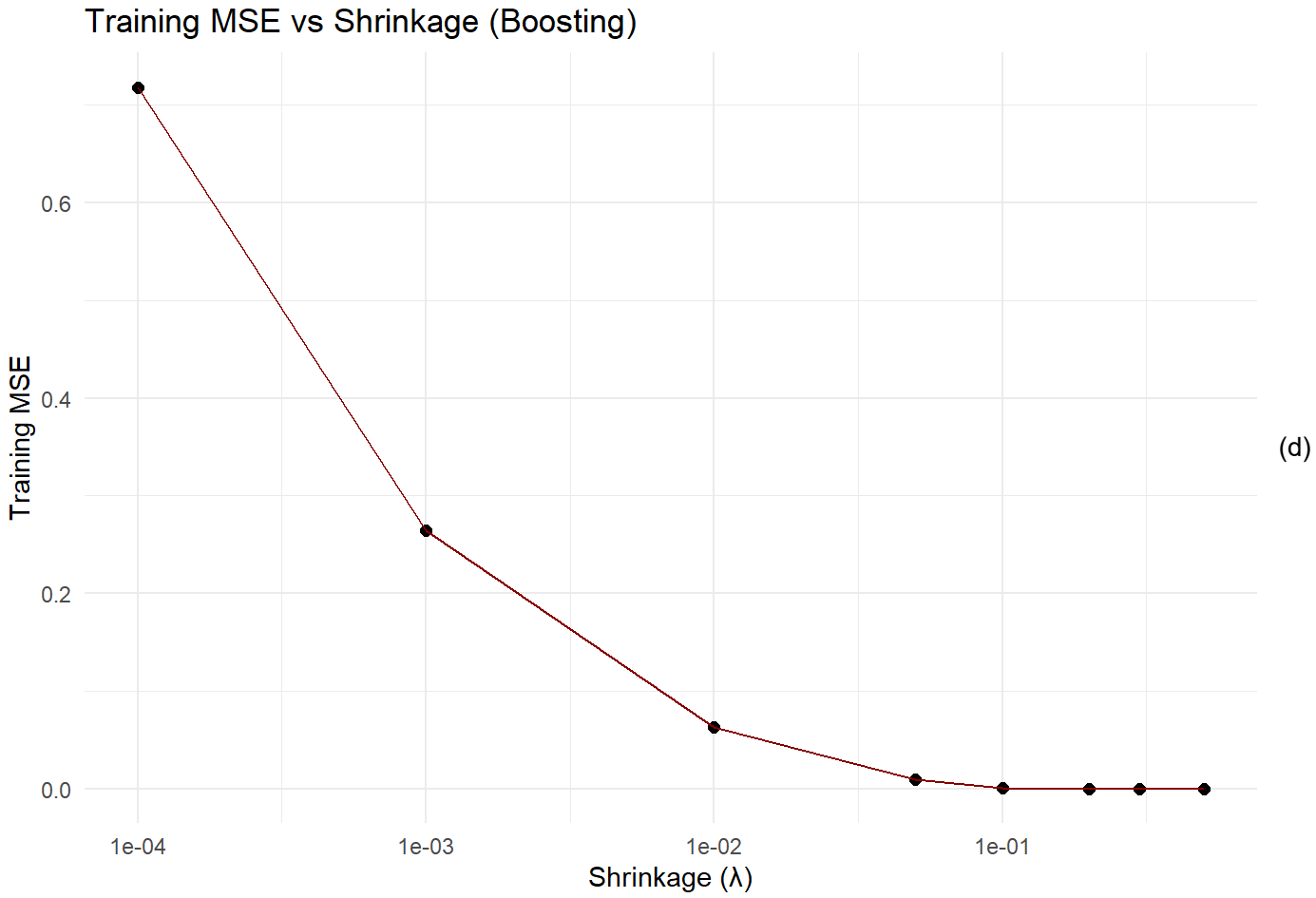
```
## Loaded gbm 2.2.2
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://g
ithub.com/gbm-developers/gbm3
```

```
shrink_vals <- c(0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5)

set.seed(1)

train_mse <- numeric(length(shrink_vals))

train_data <- train_data %>%
  mutate(across(where(is.character), as.factor))

test_data <- test_data %>%
  mutate(across(where(is.character), as.factor))
```

```
for (i in seq_along(shrink_vals)) {
  shrink <- shrink_vals[i]

  boost_model <- gbm(Salary ~ ., data = train_data, distribution = "gaussian",
                     n.trees = 1000, shrinkage = shrink, interaction.depth = 4, verbose = FALSE)

  preds <- predict(boost_model, newdata = train_data, n.trees = 1000)

  train_mse[i] <- mean((train_data$Salary - preds)^2)
}
```

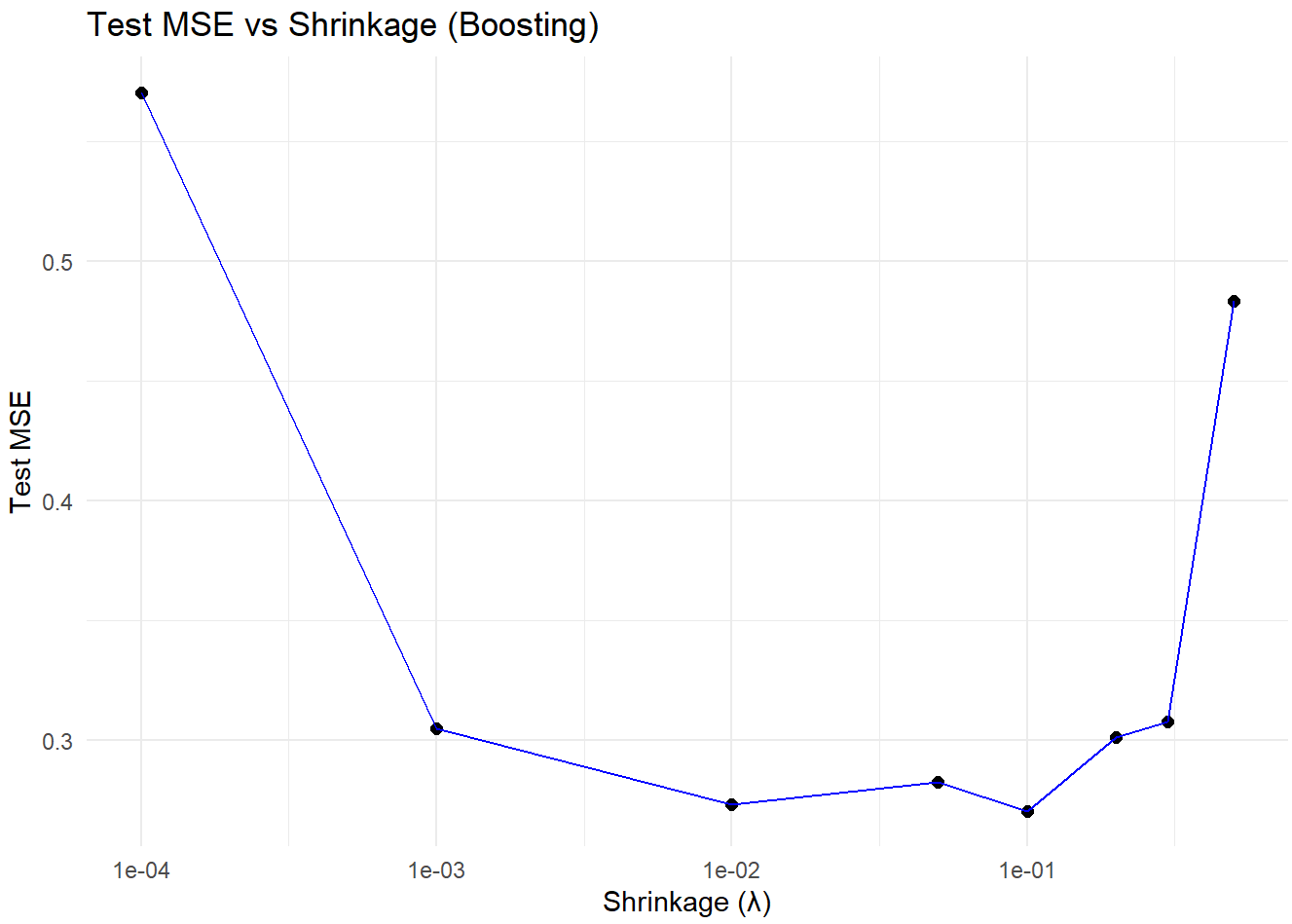
```
mse_df <- data.frame(Shrinkage = shrink_vals, Training_MSE = train_mse)

library(ggplot2)
ggplot(mse_df, aes(x = Shrinkage, y = Training_MSE)) +
  geom_point(size = 2) +
  geom_line(group = 1, color = "darkred") +
  scale_x_log10() +  # log scale helps with viewing small values
  labs(title = "Training MSE vs Shrinkage (Boosting)",
       x = "Shrinkage (λ)", y = "Training MSE") +
  theme_minimal()
```



(d)

```
shrink_vals <- c(0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5)

test_mse <- numeric(length(shrink_vals))

set.seed(1)
for (i in seq_along(shrink_vals)) {
  shrink <- shrink_vals[i]

  boost_model <- gbm(Salary ~ ., data = train_data, distribution = "gaussian",
                     n.trees = 1000, shrinkage = shrink, interaction.depth = 4, verbose = FALSE)

  preds <- predict(boost_model, newdata = test_data, n.trees = 1000)

  test_mse[i] <- mean((test_data$Salary - preds)^2)
}
```

```
mse_df_test <- data.frame(Shrinkage = shrink_vals, Test_MSE = test_mse)

ggplot(mse_df_test, aes(x = Shrinkage, y = Test_MSE)) +
  geom_point(size = 2) +
  geom_line(group = 1, color = "blue") +
  scale_x_log10() +
  labs(title = "Test MSE vs Shrinkage (Boosting)",
       x = "Shrinkage (λ)", y = "Test MSE") +
  theme_minimal()
```



e.

```
# OLS

ols_model <- lm(Salary ~ ., data = train_data)

ols_preds <- predict(ols_model, newdata = test_data)

ols_mse <- mean((test_data$Salary - ols_preds)^2)
ols_mse
```

```
## [1] 0.4917959
```

```
# Ridge Regression

library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.4.3
```

```
## Cargando paquete requerido: Matrix
```

```
## Warning: package 'Matrix' was built under R version 4.4.3
```

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

```
x_train <- model.matrix(Salary ~ ., data = train_data)[, -1]
y_train <- train_data$Salary
x_test <- model.matrix(Salary ~ ., data = test_data)[, -1]
y_test <- test_data$Salary

set.seed(1)
ridge_cv <- cv.glmnet(x_train, y_train, alpha = 0)
ridge_preds <- predict(ridge_cv, s = ridge_cv$lambda.min, newx = x_test)

ridge_mse <- mean((y_test - ridge_preds)^2)
ridge_mse
```

```
## [1] 0.4570283
```

```
# Lasso Regression

set.seed(1)
lasso_cv <- cv.glmnet(x_train, y_train, alpha = 1)
lasso_preds <- predict(lasso_cv, s = lasso_cv$lambda.min, newx = x_test)

lasso_mse <- mean((y_test - lasso_preds)^2)
lasso_mse
```

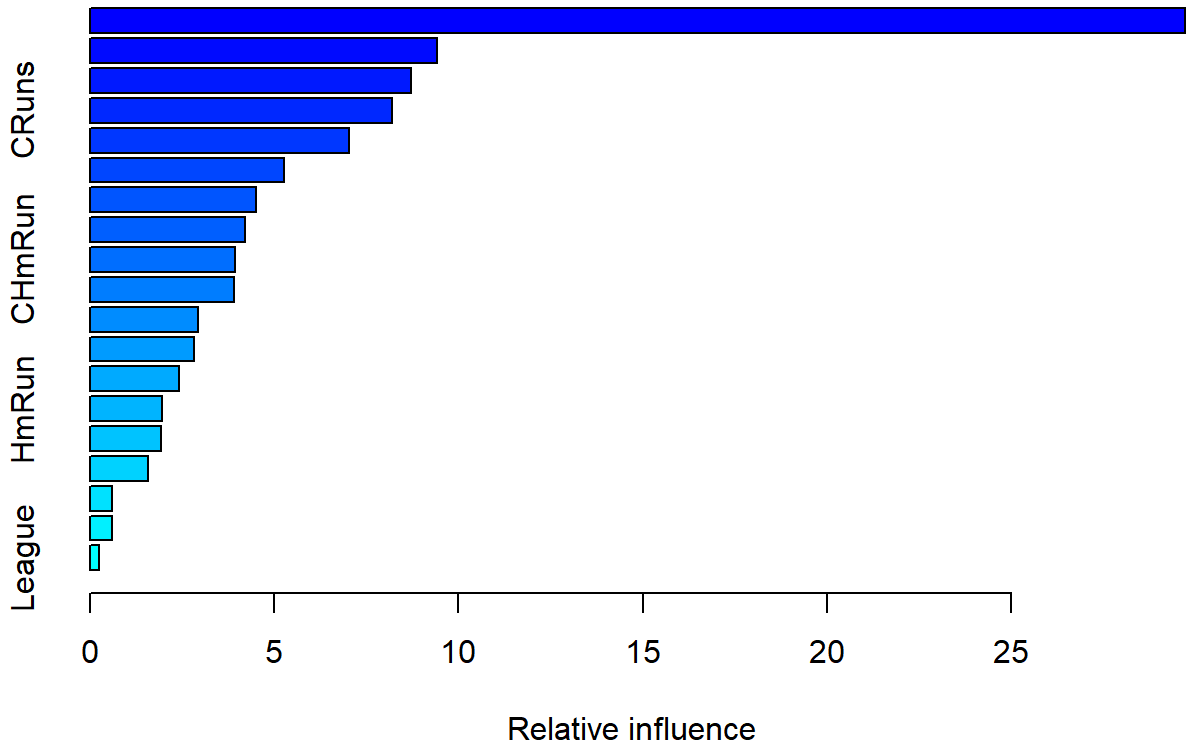
```
## [1] 0.4706337
```

Boosting, using 1,000 trees and optimal shrinkage, achieved the lowest test MSE (~0.285), outperforming all other methods.

Linear regression had the highest MSE (0.492), while ridge and lasso slightly improved performance (0.457 and 0.471, respectively). These results highlight boosting's strength in capturing complex, non-linear relationships in the data.

f.

```
set.seed(1)
best_boost <- gbm(Salary ~ ., data = train_data, distribution = "gaussian",
                  n.trees = 1000, shrinkage = 0.01, interaction.depth = 4, verbose = FALSE)

summary(best_boost)
```



	var <chr>	rel.inf <dbl>
CAtBat	CAtBat	29.7209630
CWalks	CWalks	9.4190467
CRBI	CRBI	8.7135364
CRuns	CRuns	8.1951668
CHits	CHits	7.0338392
Years	Years	5.2718953
Walks	Walks	4.5053805
PutOuts	PutOuts	4.2107275
CHmRun	CHmRun	3.9470337
AtBat	AtBat	3.9059906

1-10 of 19 rows Previous **1** 2 Next

Based on the boosted model, the most important predictor of salary is Cumulative At-Bats (CAtBat), which had a relative influence of approximately 30%. Other top predictors include Cumulative Walks (CWalks), Cumulative RBIs (CRBI), and Cumulative Runs (CRuns). These variables, which reflect a player's long-term performance, had

substantially higher influence than categorical variables like League, Division, or NewLeague, suggesting that accumulated performance statistics are strong drivers of salary in the model.

g.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.4.3
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Adjuntando el paquete: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
p <- ncol(train_data) - 1

set.seed(1)
bag_model <- randomForest(Salary ~ ., data = train_data, mtry = p, importance = TRUE)

bag_preds <- predict(bag_model, newdata = test_data)

bag_mse <- mean((test_data$Salary - bag_preds)^2)
bag_mse
```

```
## [1] 0.2301184
```

This resulted in a test MSE of 0.230, the lowest among all methods we tested.

Compared to boosting (MSE ≈ 0.285), ridge (0.457), lasso (0.471), and OLS (0.492), bagging demonstrated superior performance in predicting log-transformed salary. This suggests that averaging over many full-depth trees effectively reduces variance while maintaining low bias for this dataset.