HW3_dbagchi2

Diptendra Nath Bagchi (dbagchi2@illinois.edu)

02 March, 2020

Question 1 [70 Points] Lasso and Coordinate Descent

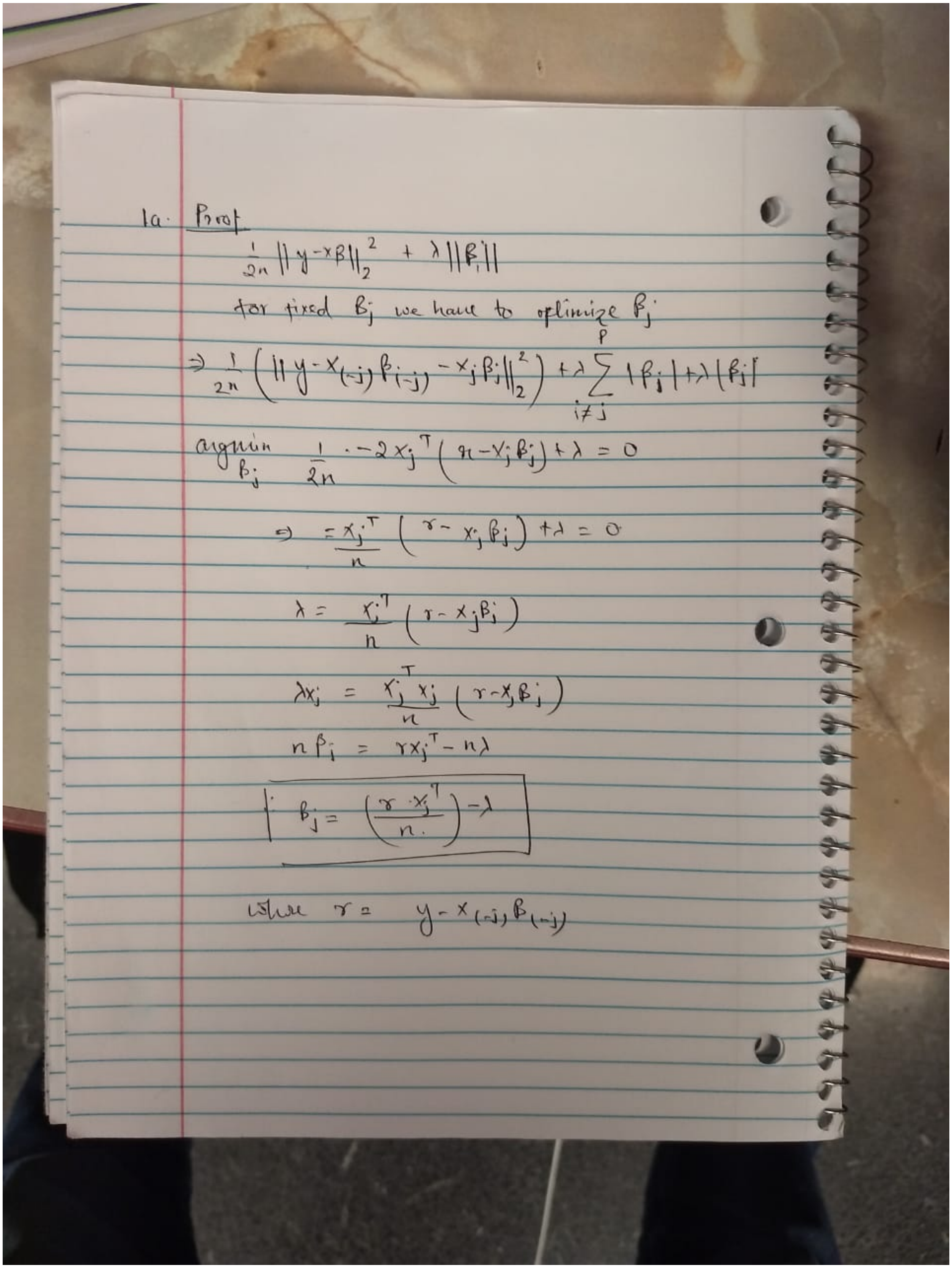
a. [15 points]

```
soft_th = function(b, lambda) {
  if (b > lambda) {
    soft_th_val = b - lambda
  } else if (b < -lambda) {
    soft_th_val = b + lambda
  } else {
    soft_th_val = 0
  }
  return (soft_th_val)
}
```

b. [15 points]

Let's write one iteration of the coordinate descent algorithm that goes through variables 1 to p and update their parameter values. First, we will generate the following data:

```
set.seed(1)
n = 200
p = 500

# generate data
V = matrix(0.2, p, p)
diag(V) = 1
X_org = as.matrix(mvrnorm(n, mu = rep(0, p), Sigma = V))
true_b = c(runif(10, -1, 1), rep(0, p-10))
y_org = X_org %*% true_b + rnorm(n)

# Center and Scale the origial X matix


x_normalized = apply(X = X_org, 2,
                     function(x) (x - mean(x)) / sqrt(sum((x - mean(x)) ^ 2) / n))

mean_x = apply(X = X_org, 2, mean)
sd_x = apply(X = X_org, 2, function(x) sqrt(sum((x - mean(x)) ^ 2) / n))

# x transpose x to see if the diagonal values are n or not
a = t(x_normalized) %*% x_normalized

# Center the outcome variable, here y_org
y_center = as.matrix(apply(X = y_org, 2,
                 function(x)  (x - mean(x))), ncol = 1)
mean_y = apply(X = y_org, 2, mean)
```



1a. Proof

$$\frac{1}{2n}\|y-X\beta\|_2^2 + \lambda\|\beta_1\|$$

for fixed β_j we have to optimize β_j

$$\Rightarrow \frac{1}{2n}\left(\|y - X_{(-j)}\beta_{(-j)} - x_j\beta_j\|_2^2\right) + \lambda\sum_{i\neq j}^{p}|\beta_i| + \lambda|\beta_j|$$

$$\operatorname{argmin}_{\beta_j} \quad \frac{1}{2n}\cdot -2x_j^T(r - x_j\beta_j) + \lambda = 0$$

$$\Rightarrow \frac{-x_j^T}{n}(r - x_j\beta_j) + \lambda = 0$$

$$\lambda = \frac{x_j^T}{n}(r - x_j\beta_j)$$

$$\lambda x_j = \frac{x_j^T x_j}{n}(r - x_j\beta_j)$$

$$n\beta_j = r x_j^T - n\lambda$$

$$\boxed{\beta_j = \left(\frac{r\cdot x_j^T}{n}\right) - \lambda}$$

where $r = y - X_{(-j)}\beta_{(-j)}$

Figure 1: Derivation

```r
coordinate_descent = function(x, y, lambda, b1 = rep(0, ncol(x))) {
  # Initialize beta values
  for (j in 1:ncol(x)) {
    if (j == 1) {
      r = y - x[, -j, drop = FALSE] %*% b1[-j]
    } else {
      r = r + (x[,j, drop = FALSE] * b1[j])  - (x[,j - 1 , drop = FALSE] * b1[j - 1])
    }
    one_beta = (t(r) %*% x[, j]) / (t(x[, j, drop = FALSE]) %*% x[, j])
    b1[j] = soft_th(one_beta, lambda = lambda)
  }
  return(b1)
}

betas_one_itr = coordinate_descent(x_normalized, y_center, 0.4)
print(betas_one_itr[1:10])
```

```
##  [1] 0.18330176 0.08744669 0.17079989 0.24813098 0.00000000 0.17953579
##  [7] 0.00000000 0.10104141 0.00000000 0.19245049
```

c. [**15 points**]

```r
myLasso = function(x, y, lambda, tol = 1e-5, maxitr = 100, post_beta = rep(0, ncol(x))) {
  if (!is.matrix(x)) stop("Covariate Matrix (x) is not a matrix")
  if (nrow(x) != length(y)) stop("Number of observations are diffent in x and y")
  for (itr in 1:maxitr) {
    pre_beta = post_beta
    post_beta = coordinate_descent(x = x, y = y, lambda = lambda, b1 = pre_beta)
    if (sum(abs(pre_beta - post_beta)) < tol)
          break;
  }
  return(post_beta)
}
scaled_betas = myLasso(x_normalized, y_center, 0.4)
print(scaled_betas[1:10])
```

```
##  [1] 0.06928864 0.00000000 0.09019810 0.18169060 0.00000000 0.16057299
##  [7] 0.00000000 0.07634841 0.00000000 0.24045067
```

d. [**10 points**]

Recover the parameter estimates on the original scale of the data. Report the nonzero parameter estimates. Check your results with the glmnet package on the first 11 parameters (intercept and the ten nonzero ones).

```r
intercept = mean_y
rescaled_betas = (scaled_betas / sd_x)
rescaled_betas = c(intercept, rescaled_betas)
print(rescaled_betas[1:11])
```

```
##  [1] 0.04398545 0.07146135 0.00000000 0.09304371 0.17396105 0.00000000
##  [7] 0.16405477 0.00000000 0.07067697 0.00000000 0.22918336
```

```r
glmnet_lasso = glmnet(x = X_org, y = y_org, alpha = 1, lambda = 0.4)
rescaled_betas_glmnet = coef(glmnet_lasso)
diff_of_myLasso_glmnet = sum(abs(rescaled_betas[1:11] - rescaled_betas_glmnet[1:11]))
```

```r
print(diff_of_myLasso_glmnet)
```

```
## [1] 0.01138849
```

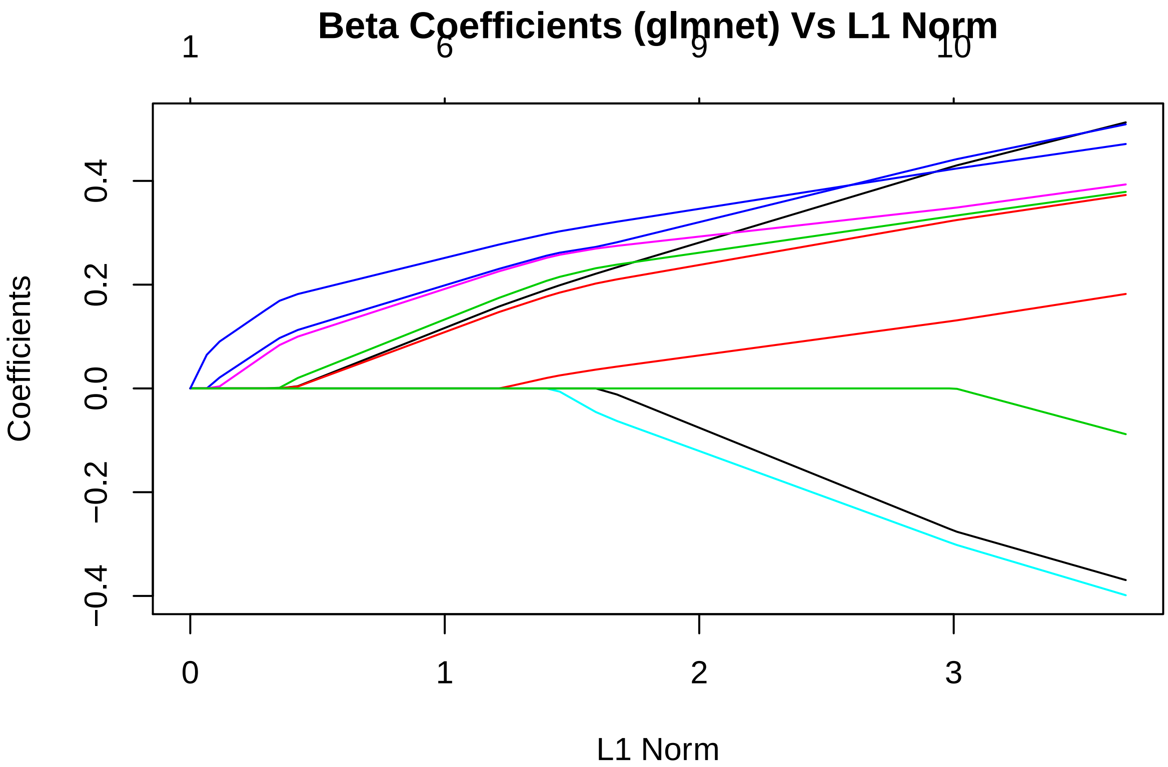
The L1 norm of my solution with the glmnet solution is 0.0113885

The L1 norm is around 0.01, which clearly shows that the solution of the first 11 terms including the intercept is really close to the glmnet solution. This explains that the function that we wrote replicates the glmnet solution. Even the Professor mentioned it during the class that if we follow step by step, we should be able to replicated the solution from the glmnet package.

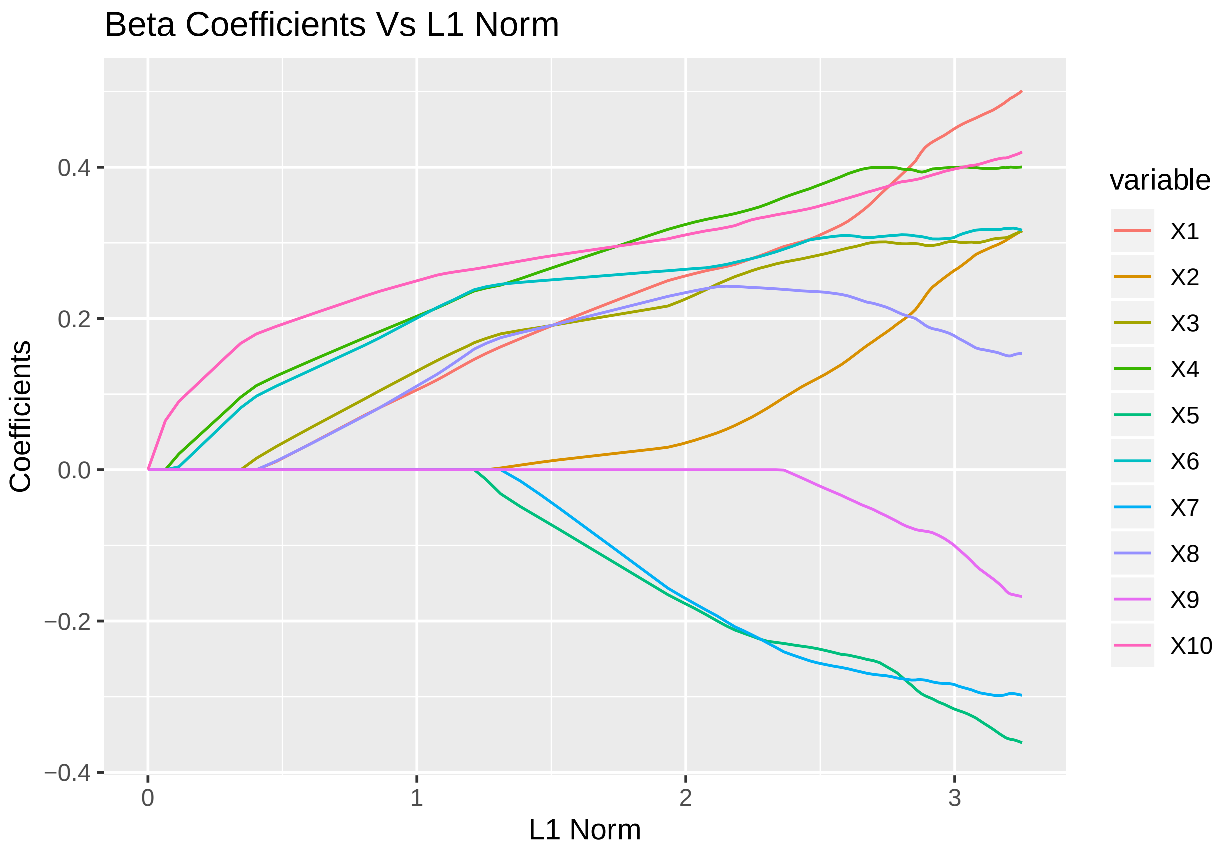
e. [15 points] Let's modify our Lasso code to perform path-wise coordinate descent.

```r
lmax = 0.765074054
lambda_all = exp(seq(log(lmax), log(lmax*0.01), length.out = 100))
df = data.frame(matrix(nrow = 1, ncol = 501))
#lambda_all = 0.4
pre_beta_lambda = rep(0, ncol(x_normalized))
for (i in lambda_all) {
  post_beta_lambda = myLasso(x = x_normalized,
                             y = y_center,
                             lambda = i,
                             post_beta = pre_beta_lambda)


  df_1 = data.frame(i, t(post_beta_lambda))
  colnames(df_1) = colnames(df)
  df = rbind(df, df_1)
  pre_beta_lambda = post_beta_lambda
  }

colnames(df)[1] = "lambdas"
df = df[-1, ]

lasso_with_all_lamdbas = data.frame("lambdas" = df[, "lambdas"], mean_y, df[, -1])

temp = lasso_with_all_lamdbas # I dont need this. this is just to recheck the solution

lasso_with_all_lamdbas[, 3:ncol(lasso_with_all_lamdbas)] =
  t(t(lasso_with_all_lamdbas[, 3:ncol(lasso_with_all_lamdbas)]) / sd_x)
```

```r
glmnet_lasso_all_lambdas = glmnet(x = X_org[, 1:10], y = y_org, alpha = 1, lambda = lambda_all)
coef_all_lambdas = t(data.matrix(coef(glmnet_lasso_all_lambdas)))
l1_norm_all_lambdas = apply(lasso_with_all_lamdbas[, 3:12],
                            1, function(x) sum(abs(x)))
plot(glmnet_lasso_all_lambdas, main = "Beta Coefficients (glmnet) Vs L1 Norm")
```




```
data_for_plotting = data.frame("l1_norm" = l1_norm_all_lambdas, lasso_with_all_lamdbas[, 3:12])
colnames(data_for_plotting) = c("l1_norm", "X1", "X2", "X3", "X4", "X5",
                                "X6", "X7", "X8", "X9", "X10")
#data_for_plotting = round(data_for_plotting, 2)
# Plot the variables with df
df_plot = melt(data = data_for_plotting, id.vars = "l1_norm")
ggplot(df_plot, aes(x = l1_norm, y = value, colour = variable)) +
  #geom_smooth(se=FALSE) +
  geom_line() +
  #xlim(c(-0.001, 0.04)) +
  ggtitle("Beta Coefficients Vs L1 Norm") +
  labs(x = "L1 Norm", y = "Coefficients")
```



If we look at the two plot- one by using 10 columns in `glmnet` and the other one using the `MyLasso` function that we wrote, the coefficient values looks very similar. This suggests that the function written by us is sort of replicating the `glmnet` package. And also, we get to know the way lasso is implemented in the package itself.

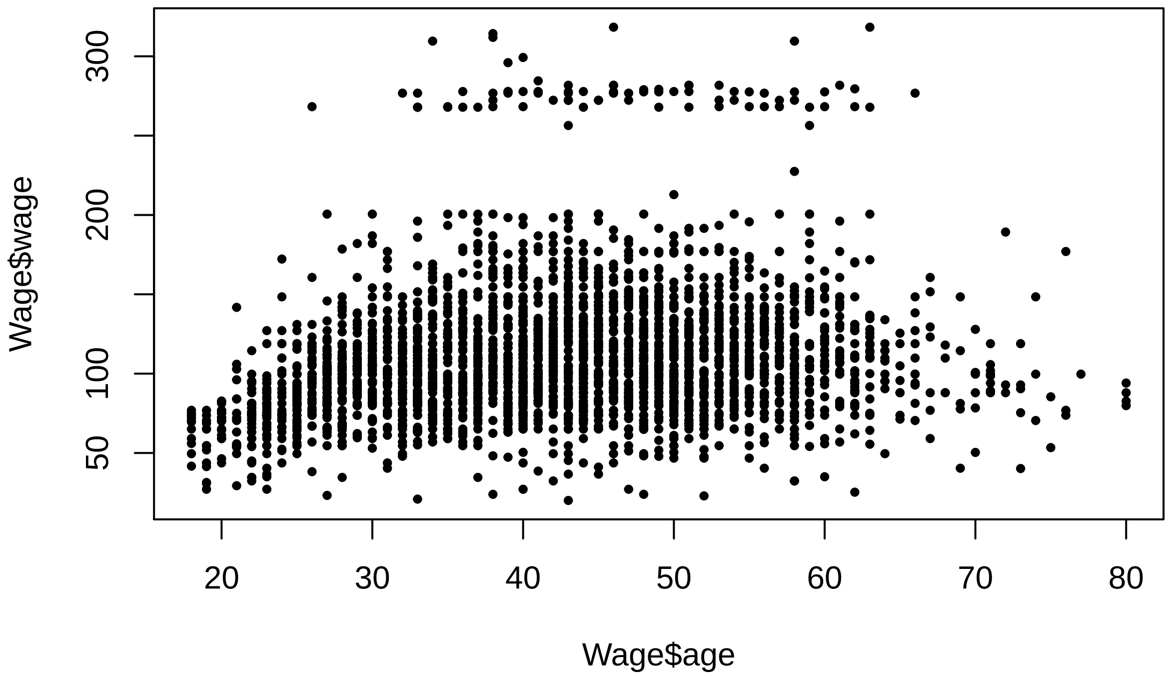
Question 2 [30 Points] Splines

We will fit and compare different spline models to the `Wage` dataset form the `ISLR` package.

a. [20 points]

Let's consider several different spline methods to model Wage using age. For each method (except the smoothing spline), report the degrees of freedom. To test your model, use the first 300 observations of the data as the training data and the rest as testing data. Use the mean squared error as the metric.

```
data(Wage)
plot(Wage$age, Wage$wage, pch = 19, cex = 0.5)
```

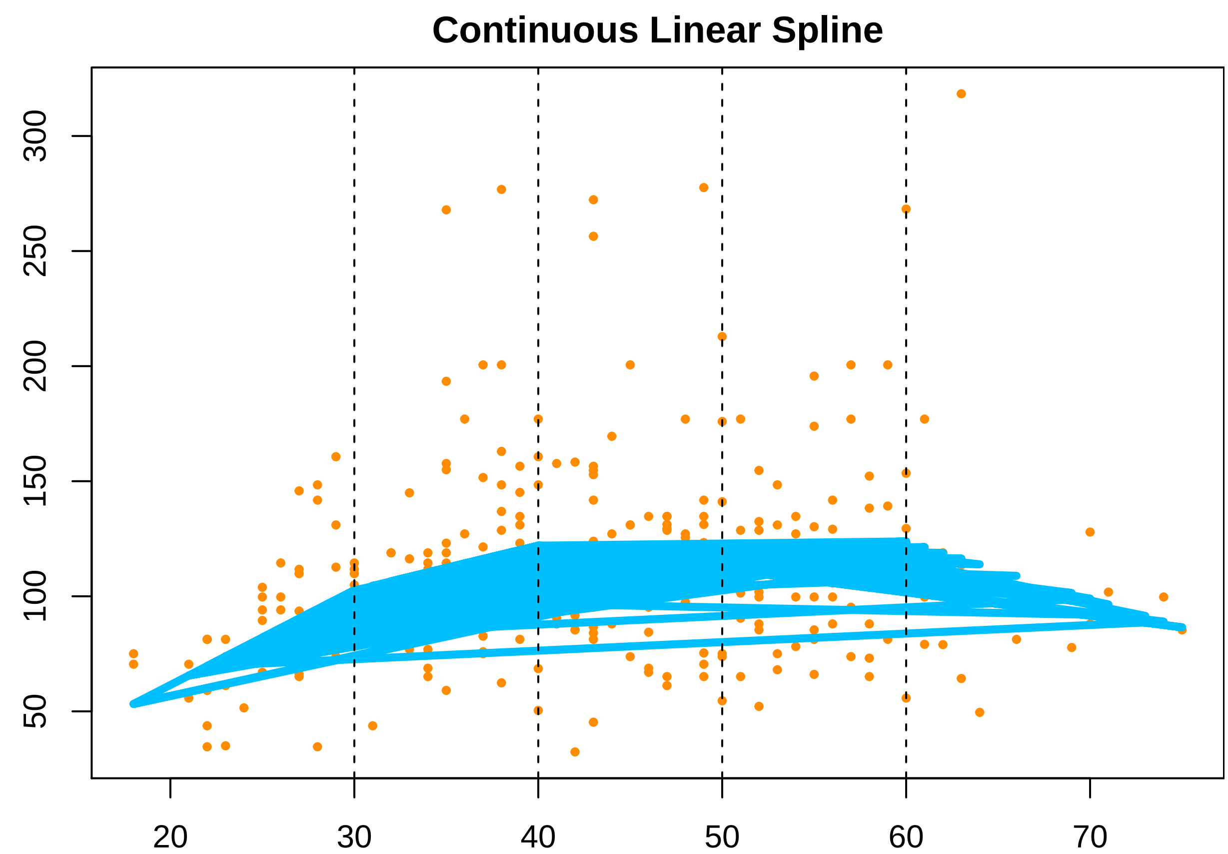



```r
random_sampling = function(Wage, mybasis, shuffle=FALSE) {
  if(shuffle == FALSE) {
    wage_trn = head(x = Wage, 300)
    wage_tst = tail(x = Wage, 2700)
    mybasis_trn = head(mybasis, 300)
    mybasis_tst = tail(mybasis, 2700)
    return(list(wage_trn, wage_tst, mybasis_trn, mybasis_tst))
  }
  ids = sample(nrow(Wage), 300)
  wage_trn = Wage[ids, ]
  wage_tst = Wage[-ids, ]
  mybasis_trn = mybasis[ids, ]
  mybasis_tst = mybasis[-ids, ]
  return(list(wage_trn, wage_tst, mybasis_trn, mybasis_tst))
}


calc_mse = function(actual, predicted) {
  return(mean((actual - predicted) ^ 2))
}
```

a1. Write your own code (you cannot use bs() or similar functions) to implement a continuous piecewise linear spline fitting. Pick 4 knots using your own judgment.

```r
# We have to use 4 knots
my_knots = c(30, 40, 50, 60)
# Function to calculate the basis for continuous splines
pos = function(x) { x * (x > 0) }
# Creating basis for the continuos linear
mybasis = cbind("x_0" = 1,
                "x_1" = Wage$age,
                "x_2" = pos(Wage$age - my_knots[1]),
```

```r
                "x_3" = pos(Wage$age - my_knots[2]),
                "x_4" = pos(Wage$age - my_knots[3]),
                "x_5" = pos(Wage$age - my_knots[4])
                )
# Sampling for the calculation of the mse
wage = random_sampling(Wage, mybasis = mybasis)

# Linear fit using our basis function
lmfit <- lm(wage[[1]]$wage ~ . -1, data = data.frame(wage[[3]]))

# Plot the graph
par(mar = c(2,3,2,0))
plot(wage[[1]]$age, wage[[1]]$wage, pch = 19, cex = 0.5,
     col = "darkorange", xlab = "Age", ylab = "Wage")
lines(wage[[1]]$age, lmfit$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
abline(v = my_knots, lty = 2)
title("Continuous Linear Spline")
```

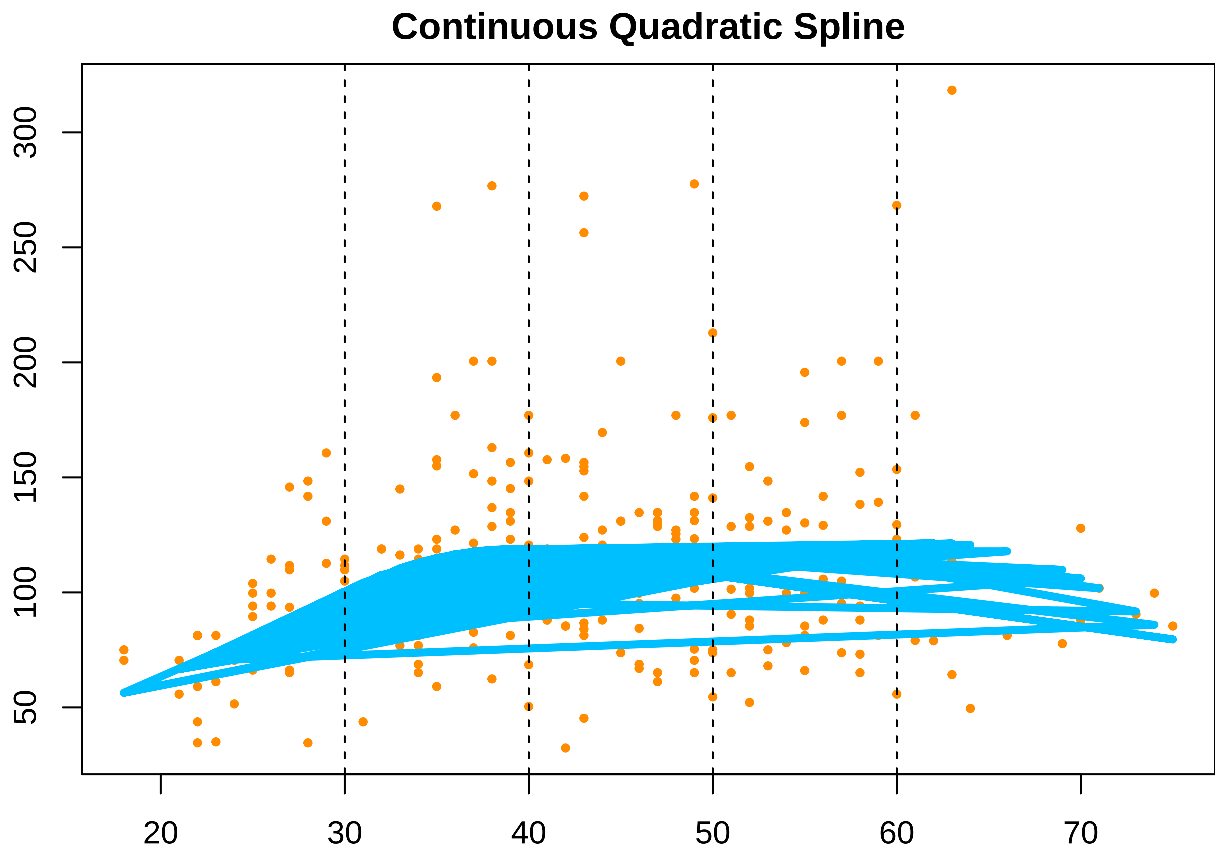



```r
mse_cls = calc_mse(wage[[2]]$wage, predict(object = lmfit, newdata = data.frame(wage[[4]])))
df_cls = ncol(mybasis)
```

a2. Write your own code to implement a quadratic spline fitting. Your spline should have a continuous first derivative. Pick 4 knots using your own judgment.

```r
# We have to use 4 knots
my_knots = c(30, 40, 50, 60)
```

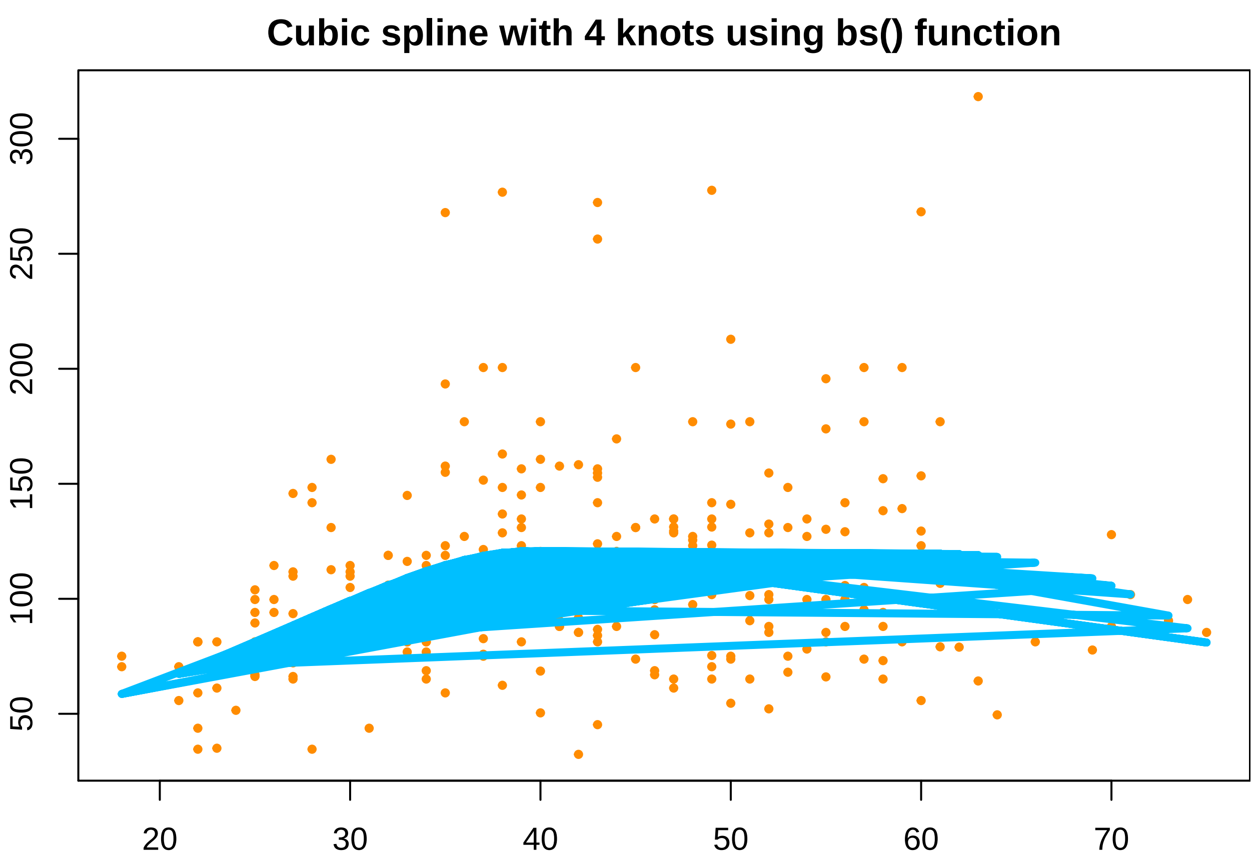
```r
# Creating basis for the Quadratic Spline
mybasis = cbind("x_0" = 1,
                "x_1" = Wage$age,
                "x_11" = Wage$age ^ 2,
                "x_22" = pos(Wage$age - my_knots[1]) ^ 2,
                "x_33" = pos(Wage$age - my_knots[2]) ^ 2,
                "x_44" = pos(Wage$age - my_knots[3]) ^ 2,
                "x_55" = pos(Wage$age - my_knots[4]) ^ 2
                )

# Sample the data into train and tst data for calculating mse
wage = random_sampling(Wage, mybasis = mybasis)

# Fit the data into new basis functions
lmfit <- lm(wage[[1]]$wage ~ . -1, data = data.frame(wage[[3]]))

# Plot the training data
par(mar = c(2,3,2,0))
plot(wage[[1]]$age, wage[[1]]$wage, pch = 19, cex = 0.5,
     col = "darkorange", xlab = "Age", ylab = "Wage")
lines(wage[[1]]$age, lmfit$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
abline(v = my_knots, lty = 2)
title("Continuous Quadratic Spline")
```




```r
mse_cqs = calc_mse(wage[[2]]$wage, predict(object = lmfit, newdata = data.frame(wage[[4]])))
df_cqs = ncol(mybasis)
```

2c. Use existing functions (bs() or similar) to implement a cubic spline with 4 knots. Use their default knots.

```
# Fit a cubic spline with 4 knots (knots = df - degree -1, if there is intercept)
lmfit <- lm(wage ~ splines::bs(age, degree = 3, df = 8, intercept = TRUE), data = wage[[1]])

# Plot the cubic splice with 4 knots - selected automatically by bs()
par(mar = c(2,2,2,0))
plot(wage[[1]]$age, wage[[1]]$wage, pch = 19, cex = 0.5,
     col = "darkorange", xlab = "Age", ylab = "Wage")
lines(wage[[1]]$age, lmfit$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
title("Cubic spline with 4 knots using bs() function")
```

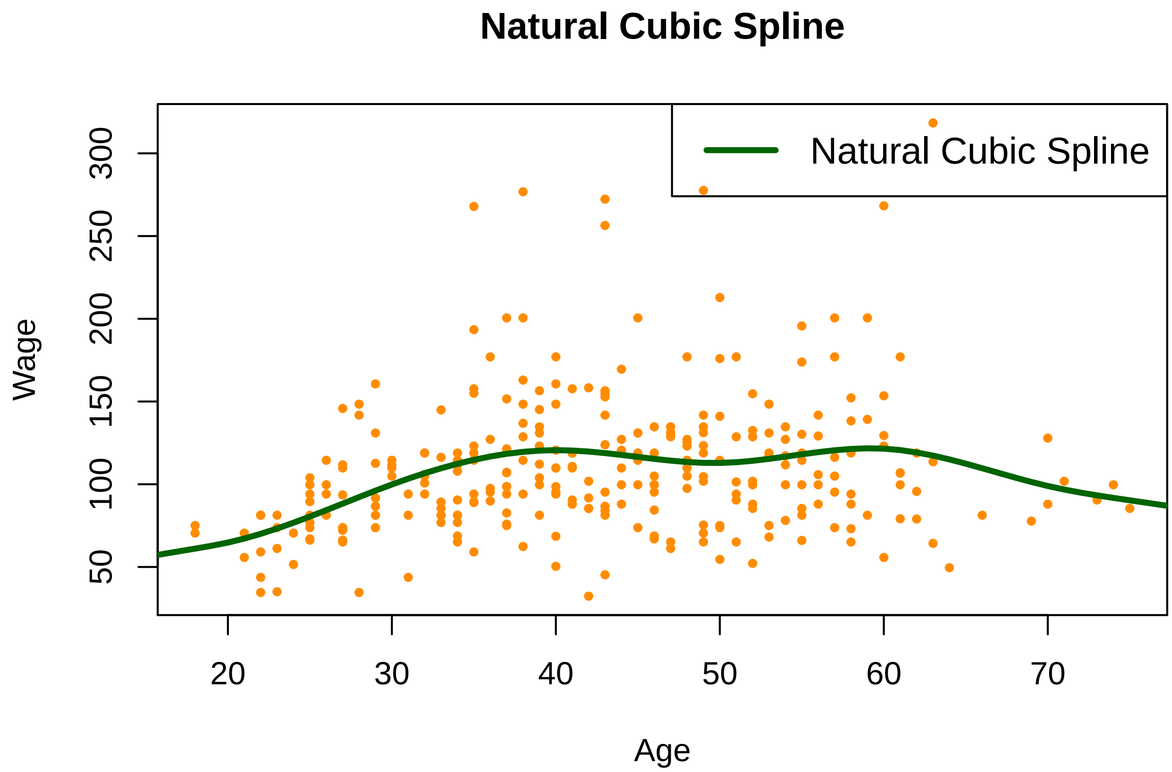



```
mse_ccs = calc_mse(wage[[2]]$wage, predict(object = lmfit, newdata = wage[[2]]))
df_ccs = 4 + length(my_knots)
```

2d. Use existing functions to implement a natural cubic spline with 6 knots. Choose your own knots.

```
# Fit a natural cubic splines with 6 own knots
fit_ns = lm(wage ~ splines::ns(age, knots = c(20, 30, 40, 50, 60, 70)), data = wage[[1]])
# Plot the data for the NCS with 6 knots
plot(wage[[1]]$age, wage[[1]]$wage, pch = 19, cex = 0.5,
     col = "darkorange", xlab = "Age", ylab = "Wage")
lines(seq(0, 100), predict(fit_ns, data.frame("age"= seq(0, 100))),
      col="darkgreen", lty=1, lwd = 3)
legend("topright", c("Natural Cubic Spline"),
```

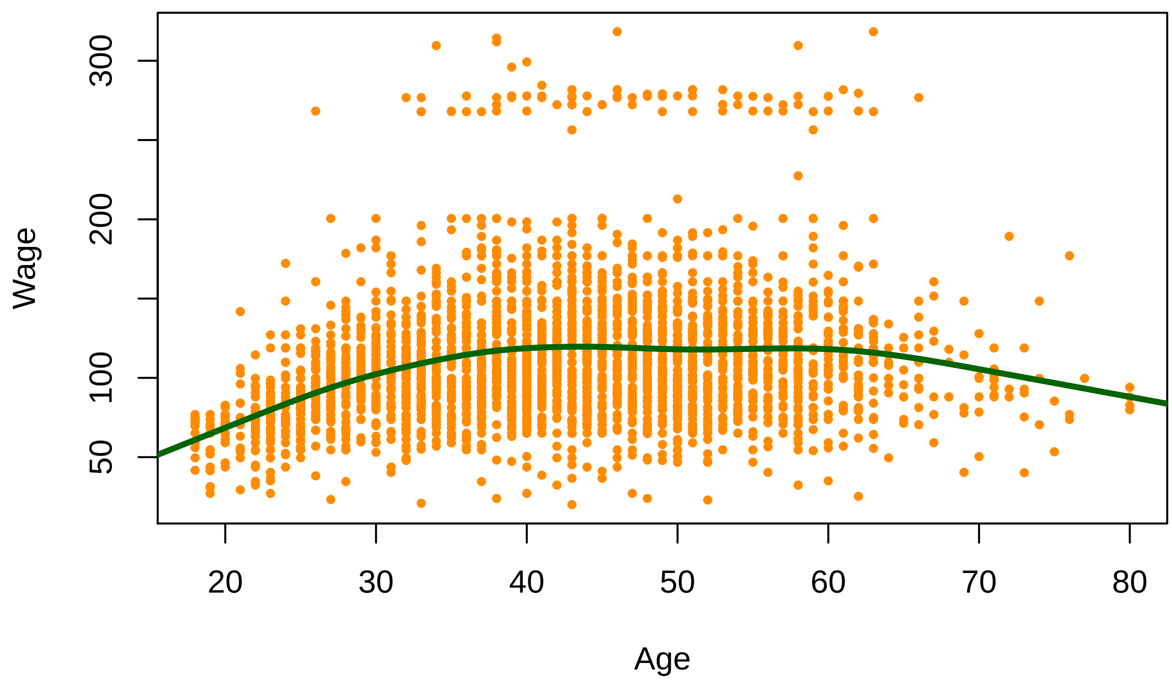
```
       col = c("darkgreen"),
       lty = 1,
       lwd = 3,
       cex = 1.2)
title("Natural Cubic Spline")
```




```
# Calculating the mse for NCS
mse_ncs = calc_mse(wage[[2]]$wage, predict(object = fit_ns, newdata = wage[[2]]))
df_ncs = 6
```

2e. Use existing functions to implement a smoothing spline. Use the built-in generalized cross-validation method to select the best tuning parameter.

```
# Fit a smoothing spline using smooth.spline function in stats package
fit_smooth = smooth.spline(Wage$age, Wage$wage)
# Plot the smoothing spline
plot(Wage$age, Wage$wage, pch = 19, cex = 0.5, col = "darkorange",
      xlab = "Age", ylab = "Wage")
lines(seq(0, 100), predict(fit_smooth, seq(0, 100))$y, col="darkgreen", lty=1, lwd = 3)
```




```
df_ss = fit_smooth$df
```

```
# combining all the data to make an html table
dof_mse_df = tibble("Model" = c("Linear Spline", "Quadratic Spline", "Cubic Spline",
                                "Natural Cubic Spline", "Smooth Spline"),
                    "Degrees of Freedom" = c(df_cls, df_cqs, df_ccs, df_ncs, df_ss),
                    "Mean Squared Error" = c(mse_cls, mse_cqs, mse_ccs, mse_ncs, NA))
kable(dof_mse_df, digits = 2, row.names = FALSE, align = "c") %>%
  kable_styling("striped", full_width = FALSE) %>%
  add_header_above(c("Degrees of Freedom and MSE" = 3)) %>%
  column_spec(column = 1, bold = TRUE)
```

Degrees of Freedom and MSE		
Model	Degrees of Freedom	Mean Squared Error
Linear Spline	6.00	1593.14
Quadratic Spline	7.00	1598.12
Cubic Spline	8.00	1599.63
Natural Cubic Spline	6.00	1595.62
Smooth Spline	6.47	NA

b. [10 points]

After writing these models, evaluate their performances by repeatedly doing a train-test split of the data. Randomly sample 300 observations as the training data and the rest as testing data. Repeat this process 200 times. Use the mean squared error as the metric.

b1. Record and report the mean, median, and standard deviation of the errors for each method. Also, provide an informative boxplot that displays the error distribution for all models side-by-side.

```
rownames(Wage) = 1:nrow(Wage)
linear_spline = function() {
```

```r
  # We have to use 4 knots
  my_knots = c(30, 40, 50, 60)
  # Function to calculate the basis for continuous splines
  pos = function(x) { x * (x > 0) }
  # Creating basis for the continuos linear
  mybasis = cbind("x_0" = 1,
                  "x_1" = Wage$age,
                  "x_2" = pos(Wage$age - my_knots[1]),
                  "x_3" = pos(Wage$age - my_knots[2]),
                  "x_4" = pos(Wage$age - my_knots[3]),
                  "x_5" = pos(Wage$age - my_knots[4])
                  )
  # Sampling for the calculation of the mse
  wage = random_sampling(Wage, mybasis = mybasis, shuffle = TRUE)

  # Linear fit using our basis function
  lmfit <- lm(wage[[1]]$wage ~ . -1, data = data.frame(wage[[3]]))
  mse_cls = calc_mse(wage[[2]]$wage, predict(object = lmfit, newdata = data.frame(wage[[4]])))
  return(mse_cls)
  }

mse_cls_200 = replicate(n = 200, expr = linear_spline(), simplify = "vector")
```

```r
quadratic_spline = function() {
  # We have to use 4 knots
  my_knots = c(30, 40, 50, 60)
  # Creating basis for the Quadratic Spline
  mybasis = cbind("x_0" = 1,
                  "x_1" = Wage$age,
                  "x_11" = Wage$age ^ 2,
                  "x_22" = pos(Wage$age - my_knots[1]) ^ 2,
                  "x_33" = pos(Wage$age - my_knots[2]) ^ 2,
                  "x_44" = pos(Wage$age - my_knots[3]) ^ 2,
                  "x_55" = pos(Wage$age - my_knots[4]) ^ 2
                  )

  # Sample the data into train and tst data for calculating mse
  wage = random_sampling(Wage, mybasis = mybasis, shuffle = TRUE)

  # Fit the data into new basis functions
  lmfit <- lm(wage[[1]]$wage ~ . -1, data = data.frame(wage[[3]]))
  mse_cqs = calc_mse(wage[[2]]$wage, predict(object = lmfit, newdata = data.frame(wage[[4]])))
  return(mse_cqs)

  }

mse_cqs_200 = replicate(n = 200, expr = quadratic_spline(), simplify = "vector")
```

```r
cubic_spline = function() {
  wage = random_sampling(Wage, mybasis = mybasis, shuffle = TRUE)
  # Fit a cubic spline with 4 knots (knots = df - degree -1, if there is intercept)
  lmfit <- lm(wage ~ splines::bs(age, degree = 3, df = 8, intercept = TRUE), data = wage[[1]])
  mse_ccs = calc_mse(wage[[2]]$wage, predict(object = lmfit, newdata = wage[[2]]))
  return(mse_ccs)
```

```r
}
mse_ccs_200 = replicate(n = 200, expr = cubic_spline(), simplify = "vector")
```

```r
nature_cubic_spline = function() {
  #set.seed(12345)
  wage = random_sampling(Wage, mybasis = mybasis, shuffle = TRUE)
  my_knots = runif(6, min(wage[[1]]$age), max(wage[[1]]$age))
  fit_ns = lm(wage ~ splines::ns(age, knots = my_knots),
                      data = wage[[1]])
  mse_ncs = calc_mse(wage[[2]]$wage, predict(object = fit_ns, newdata = wage[[2]]))
  return(mse_ncs)
}
mse_ncs_200 = replicate(n = 200, expr = nature_cubic_spline(), simplify = "vector")
```

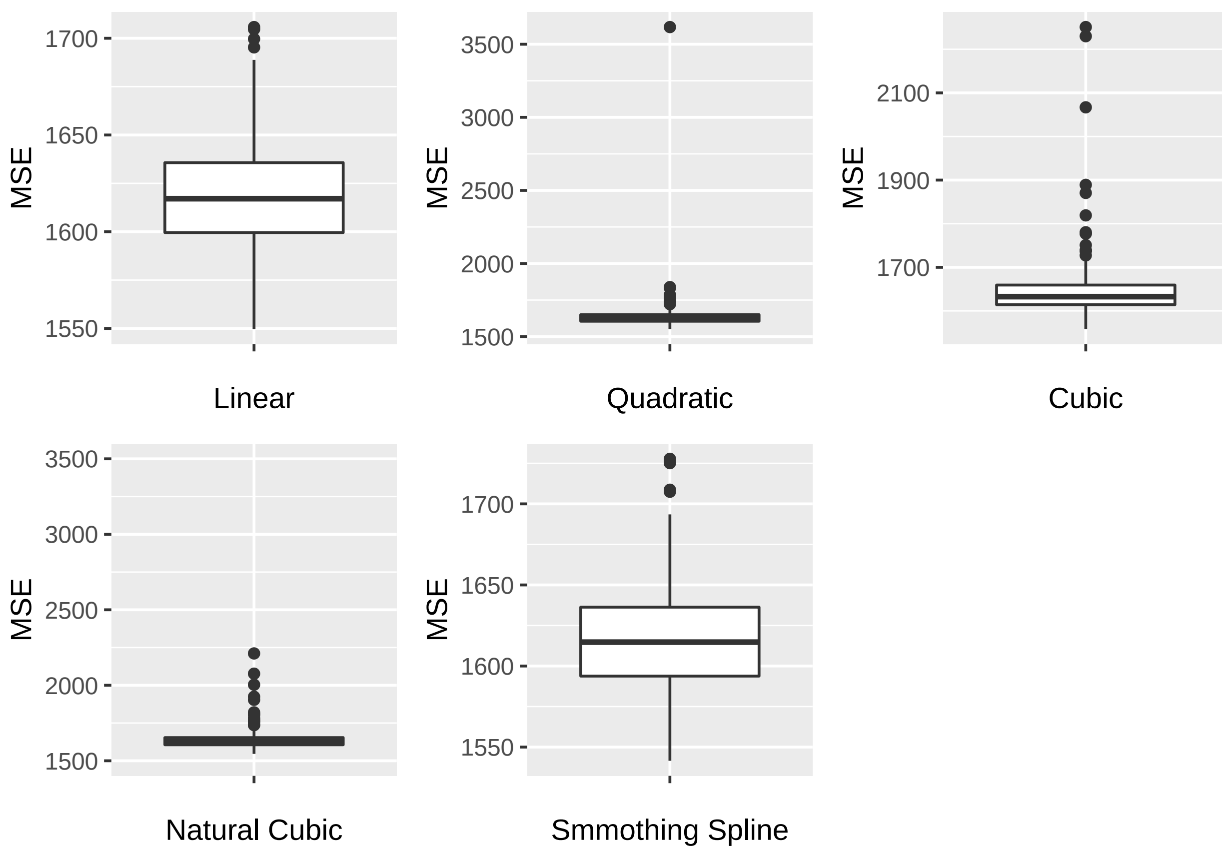
```r
smooth_spline = function() {
  wage = random_sampling(Wage, mybasis = mybasis, shuffle = TRUE)
  # Fit a smoothing spline using smooth.spline function in stats package
  fit_smooth = smooth.spline(wage[[1]]$age, wage[[1]]$wage)
  mse_ss = calc_mse(wage[[2]]$wage,
                    predict(object = fit_smooth, wage[[2]]$age)$y)

  }
mse_ss_200 = replicate(n = 200, expr = smooth_spline(), simplify = "vector")
```

```r
mean_cls = mean(mse_cls_200)
mean_cqs = mean(mse_cqs_200)
mean_ccs = mean(mse_ccs_200)
mean_ncs = mean(mse_ncs_200)
mean_ss = mean(mse_ss_200)

med_cls = median(mse_cls_200)
med_cqs = median(mse_cqs_200)
med_ccs = median(mse_ccs_200)
med_ncs = median(mse_ncs_200)
med_ss = median(mse_ss_200)

sd_cls = sd(mse_cls_200)
sd_cqs = sd(mse_cqs_200)
sd_ccs = sd(mse_ccs_200)
sd_ncs = sd(mse_ncs_200)
sd_ss = sd(mse_ss_200)

dof_mse_df = tibble("Model" = c("Linear Spline", "Quadratic Spline", "Cubic Spline",
                                "Natural Cubic Spline", "Smooth Spline"),
                    "Mean" = c(mean_cls, mean_cqs, mean_ccs, mean_ncs, mean_ss),
                    "Median" = c(med_cls, med_cqs, med_ccs, med_ncs, med_ss),
                    "Standard Deviation" = c(sd_cls, sd_cqs, sd_ccs, sd_ncs, sd_ss))
kable(dof_mse_df, digits = 2, row.names = FALSE, align = "c") %>%
  kable_styling("striped", full_width = FALSE) %>%
  add_header_above(c("Mean Median and Standard Deviation" = 4)) %>%
  column_spec(column = 1, bold = TRUE)
```

Mean Median and Standard Deviation			
Model	Mean	Median	Standard Deviation
Linear Spline	1619.45	1617.06	30.48
Quadratic Spline	1643.37	1628.56	147.18
Cubic Spline	1650.62	1632.79	82.59
Natural Cubic Spline	1717.31	1627.07	887.39
Smooth Spline	1616.52	1614.72	33.17

Base on the mean + 1 SE rule, I will choose the Linear Spline as the (mean + sd) is less than the (mean + sd) of smoothing spline.

```
df = data.frame("Linear Spline" = mse_cls_200,
                "Quadratic Spline" = mse_cqs_200,
                "Cubic Spline" = mse_ccs_200,
                "Natural Cubic Spline" = mse_ncs_200,
                "Smooth Spline" = mse_ss_200
                )
p1 = ggplot(data = df, aes(x = "", y = df$Linear.Spline)) + geom_boxplot() + labs(x = "Linear", y = "MSE
p2 = ggplot(data = df, aes(x = "", y = df$Quadratic.Spline)) + geom_boxplot() + labs(x = "Quadratic", y
p3 = ggplot(data = df, aes(x = "", y = df$Cubic.Spline)) + geom_boxplot()+ labs(x = "Cubic", y = "MSE")
p4 = ggplot(data = df, aes(x = "", y = df$Natural.Cubic.Spline)) + geom_boxplot() + labs(x = "Natural Cu
p5 = ggplot(data = df, aes(x = "", y = df$Smooth.Spline)) + geom_boxplot() + labs(x = "Smmothing Spline
gridExtra::grid.arrange(p1, p2, p3, p4, p5, ncol = 3)
```



b2. For each method, provide a discussion of their (theoretical) advantages and disadvantages. Based on the empirical results, what method would you prefer?

Adavantages and Disadvantages of different spline methods

1. Linear Spline

Advantages - It is linear at the boundaries as their is no other terms

Disadvantages

- It does not capture the variation in the data as it is linear
- If the knots are not chosen correctly, then the accuracy comes down

2. Cubic and Quadratic Splines

Advantages

- Because of the more flexibility of the model (by including higher terms) it captures the data much better than the linear spline.

Disadvantages

- Because of the higher terms, the value at the boundary can be highly variable and can be out of the ranges
- Similar to the linear spline, the knot selection can have huge impact on the fit

3. Natural Cubic Spline

Advantages

- One of the biggest advantages is that because of the additonla conditions on cubic splines, the value does not overshoot at the boundaries

Disadvantages

- Similar to the linear spline, the knot selection can have huge impact on the fit

4. Smooth Spline

Advantages

- The biggest advantage is that, we do not have to select the knots manually. It considers all the data points as a knot

Disadvantages

- The disadvantage is that it has to fit and complex model and has to estimate all the parameters(n) which makes it a complex model