HW2_dbagchi2

Diptendra Nath Bagchi (dbagchi2@illinois.edu)

17 February, 2020

```r
data("BostonHousing2")
BostonHousing2[, "chas"] = as.integer(as.character(BostonHousing2[, "chas"]))
```

Question 1 [30 Points] - Linear Model Selection

We will use the Boston Housing data (`BostonHousing2` from the mlbench package) for this question. If you do not use R, you can download a .csv file from the course website. We will remove variables `medv`, `town` and `tract` from the data and use `cmedv` as the outcome. First, you need to standardize all variables marginally (only the covariates, not the outcome) to mean 0 and sample variation 1. Answer the following questions by performing linear regression and model selection.

```r
# To check if there is any NA values present in the columns
apply(BostonHousing2, 2, function(x) sum(is.na(x)))
```

```
##    town   tract     lon     lat    medv   cmedv    crim      zn   indus
##       0       0       0       0       0       0       0       0       0
##    chas     nox      rm     age     dis     rad     tax ptratio       b
##       0       0       0       0       0       0       0       0       0
##   lstat
##       0
```

```r
#str(BostonHousing2)
```

```r
BostonHousing2 = BostonHousing2[,
                                !(colnames(BostonHousing2) %in% c("medv", "town", "tract"))]
# Scale function does the same manipulation

#Keep an original data frame
BostonHousing2_original = BostonHousing2

BostonHousing2[, !(colnames(BostonHousing2) %in% c("cmedv"))] = as.data.frame(apply(
  BostonHousing2[, !(colnames(BostonHousing2) %in% c("cmedv"))], 2,
  function(x) {(x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)}))
bstn_hsn2 = BostonHousing2
```

a. [5 Points] Perform a linear regression and obtain the ordinary least square estimators and their variances.

```r
# Linear Model
bstn_lmod = lm(formula = cmedv ~ ., data = bstn_hsn2)
#kable(as.data.frame(summary(bstn_lmod)))
model_summary = summary(bstn_lmod)
lm_table = model_summary$coefficients
beta_estimate = as.data.frame(lm_table[, "Estimate"])
beta_estimate = data.frame("Variable" = rownames(beta_estimate), beta_estimate)
colnames(beta_estimate)[2] = "OLS Estimates"
beta_sd = as.data.frame(lm_table[, "Std. Error"])
```

Table 1: Table: Ordinary Least Squares Estimates and Variance

OLS Estimates & Variance		
Variable	OLS Estimates	Variance-Betas
(Intercept)	22.53	0.04
age	0.07	0.14
b	0.84	0.06
chas	0.65	0.05
crim	-0.90	0.08
dis	-2.95	0.19
indus	0.10	0.18
lat	0.28	0.05
lon	-0.30	0.06
lstat	-3.84	0.13
nox	-1.83	0.22
ptratio	-1.90	0.09
rad	2.67	0.34
rm	2.64	0.09
tax	-2.17	0.39
zn	1.09	0.10

```r
beta_sd = data.frame("Variable" = rownames(beta_sd), beta_sd)
beta_sd[, 2] = beta_sd[, 2] ^ 2
colnames(beta_sd)[2] = "Variance-Betas"
beta_estimate_var = merge(x = beta_estimate,y = beta_sd, by = ("Variable"))
beta_estimate_var$Variable = as.character(beta_estimate_var$Variable)
# Block code to create the HTML table for the df - beta_estimate_var
kable(beta_estimate_var, digits = 2,
      caption = "Table: Ordinary Least Squares Estimates and Variance") %>%
  kable_styling("striped", full_width = FALSE) %>%
  add_header_above(c("OLS Estimates & Variance" = 3)) %>%
  column_spec(column = 1, bold = TRUE)
```

b. [5 Points] Starting from the full model, use stepwise regression with backward and BIC criterion to select the best model. Which variables are removed from the full model?

```r
bstn_step_wise_back = step(object = bstn_lmod,
                           direction = "backward",
                           trace = 0,
                           criterion = "BIC")
variables_removed = beta_estimate_var[
    !(beta_estimate_var[,
                        "Variable"] %in% rownames(
                          summary(bstn_step_wise_back)$coefficients)), "Variable"]
variables_removed = as.data.frame(variables_removed)
colnames(variables_removed) = "Variables removed from the full model"
kable(variables_removed, digits = 2, align = "c",
      caption = "Table: Variables removed from the full model using stepwise regression") %>%
  kable_styling("striped", full_width = FALSE) %>%
  column_spec(column = 1, bold = TRUE)
```

Table 2: Table: Variables removed from the full model using stepwise regression

Variables removed from the full model
age
indus
lat
lon

Table 3: Table: Best model for each size (p)

Best Model for each size																		
Size	cp	bic	adjr2	lon	lat	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat
1	367.53	-390.29	0.55															*
2	189.06	-501.76	0.64								*							*
3	116.46	-553.52	0.68								*					*		*
4	94.91	-567.07	0.69								*		*			*		*
5	61.51	-592.10	0.71							*	*		*			*		*
6	48.55	-599.69	0.72						*	*	*		*			*		*
7	38.18	-605.18	0.72						*	*	*		*			*	*	*
8	31.28	-607.55	0.73				*		*	*	*		*			*	*	*
9	27.81	-606.68	0.73						*	*	*		*	*	*	*	*	*
10	19.03	-611.18	0.73			*	*			*	*		*	*	*	*	*	*
11	10.68	-615.48	0.74			*	*		*	*	*		*	*	*	*	*	*
12	11.55	-610.41	0.74		*	*	*		*	*	*		*	*	*	*	*	*
13	12.10	-605.68	0.74	*	*	*	*		*	*	*		*	*	*	*	*	*
14	14.03	-599.52	0.74	*	*	*	*	*	*	*	*		*	*	*	*	*	*
15	16.00	-593.33	0.74	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

c. [5 Points] Starting from this full model, use the best subset selection and list the best model of each model size.

```r
bstn_best_subset_leaps = regsubsets(cmedv ~ ., data = bstn_hsn2, nvmax = 15)
all_combn_table = with(summary(bstn_best_subset_leaps), data.frame(cp, bic, adjr2, outmat))
all_combn_table = data.frame("Size" = 1:15, all_combn_table)
kable(all_combn_table, digits = 2, row.names = FALSE,
      caption = "Table: Best model for each size (p)") %>%
  kable_styling("striped", full_width = FALSE, latex_options = "scale_down") %>%
  add_header_above(c("Best Model for each size" = 19)) %>%
  column_spec(column = 1, bold = TRUE)
```

d. [5 Points] Use the BIC criterion to select the best model from part c). Which variables are removed from the full model?

```r
bstn_subset_table = with(summary(bstn_best_subset_leaps), data.frame(bic, outmat))
bstn_best_mod_bic = bstn_subset_table[which.min(bstn_subset_table$bic), ]
bstn_best_mod_bic[bstn_best_mod_bic == " "] = NA
variables_removed_leaps = as.data.frame(
  colnames(bstn_best_mod_bic)[colSums(is.na(bstn_best_mod_bic)) > 0])
colnames(variables_removed_leaps) = "Variables removed - Leaps Method"
kable(variables_removed_leaps, digits = 2, align = "c",
      caption =
        "Table: Variables removed from the full model using best subset regression") %>%
  kable_styling("striped", full_width = FALSE) %>%
  column_spec(column = 1, bold = TRUE)
```

Table 4: Table: Variables removed from the full model using best subset regression

Variables removed - Leaps Method
lon
lat
indus
age

e. [10 Points] Our solution is obtained based on the scaled X. Can you recover the original OLS estimates based on the original data? For this question, you can use information from the original design matrix. However, you cannot refit the linear regression. Provide a rigorous mathematical derivation and also validate that by comparing it to the lm() function on the original data.

```r
standardized_betas = beta_estimate_var[-1, c("Variable", "OLS Estimates")]
sd_original_data = apply(BostonHousing2_original[, !(colnames(BostonHousing2_original) %in% c("cmedv"))
sd_original_data = data.frame("Variable" = names(sd_original_data), sd_original_data)
estimate_sd_df = merge(standardized_betas, sd_original_data, by = "Variable")
estimate_sd_df[, "Original Betas"] =
  estimate_sd_df$`OLS Estimates` / estimate_sd_df$sd_original_data
colnames(estimate_sd_df) = c("Variable", "Standardized Betas",
                             "Standard Deviation", "Un-Standardized Betas")

# To validate the output
bstn_orig_lmod = lm(formula = cmedv ~ ., data = BostonHousing2_original)
bstn_orig_sum = summary(bstn_orig_lmod)
df = data.frame(bstn_orig_sum$coefficients[, "Estimate"])
df = data.frame("Variable" = rownames(df), df)
df = df[-1, ]
colnames(df) = c("Variable", "Un-Standardized Beta's (Using lm)")

# Merge the df with the values from the lm model
estimate_sd_df = merge(x = estimate_sd_df, y = df, by = "Variable")

# Convert data frame in to HTML table
kable(estimate_sd_df, digits = 2, row.names = FALSE, align = "c") %>%
  kable_styling("striped", full_width = FALSE) %>%
  add_header_above(c("Standardized and Un-Standardized Betas" = 5)) %>%
  column_spec(column = 1, bold = TRUE)
```

Standardized and Un-Standardized Betas				
Variable	Standardized Betas	Standard Deviation	Un-Standardized Betas	Un-Standardized Beta's (Using lm)
age	0.07	28.15	0.00	0.00
b	0.84	91.29	0.01	0.01
chas	0.65	0.25	2.58	2.58
crim	-0.90	8.60	-0.10	-0.10
dis	-2.95	2.11	-1.40	-1.40
indus	0.10	6.86	0.02	0.02
lat	0.28	0.06	4.50	4.50
lon	-0.30	0.08	-3.94	-3.94
lstat	-3.84	7.14	-0.54	-0.54
nox	-1.83	0.12	-15.82	-15.82
ptratio	-1.90	2.16	-0.88	-0.88
rad	2.67	8.71	0.31	0.31
rm	2.64	0.70	3.75	3.75
tax	-2.17	168.54	-0.01	-0.01
zn	1.09	23.32	0.05	0.05

Last two columns indeed show that the values calculated by the mathematical formula given below give the exact same value from the linear model (lm function) directly.

Relation between Standardized and Un-Standardized Betas

$$E(Y) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p = \beta_0 + \sum_{i=1}^{p} \beta_i X_i$$

Standardizing to

$$X_i = \sigma_i Z_i + \mu_i$$

$$E(Y) = \beta_0 + \sum_{i=1}^{p} \beta_i(\sigma_i Z_i + \mu_i) = \left(\beta_0 + \sum_{i=1}^{p} \beta_i \mu_i\right) + \sum_{i=1}^{p} (\beta_i \sigma_i) Z_i = \beta_0^* + \sum_{i=1}^{p} \beta_i^* Z_i$$

Hence, $\beta_i^* = \sigma_i \beta_i$

Where,

- X_i : Un-standardized value of X
- σ_i : Standard Deviation of X
- Z_i : Standardized value of X
- μ_i : Mean of X
- β_i : Beta coefficients of the un-standardized X
- β_0 : Beta coefficient of the intercept term

Reference

Question 2 [70 Points] - Ridge Regression and Scaling Issues

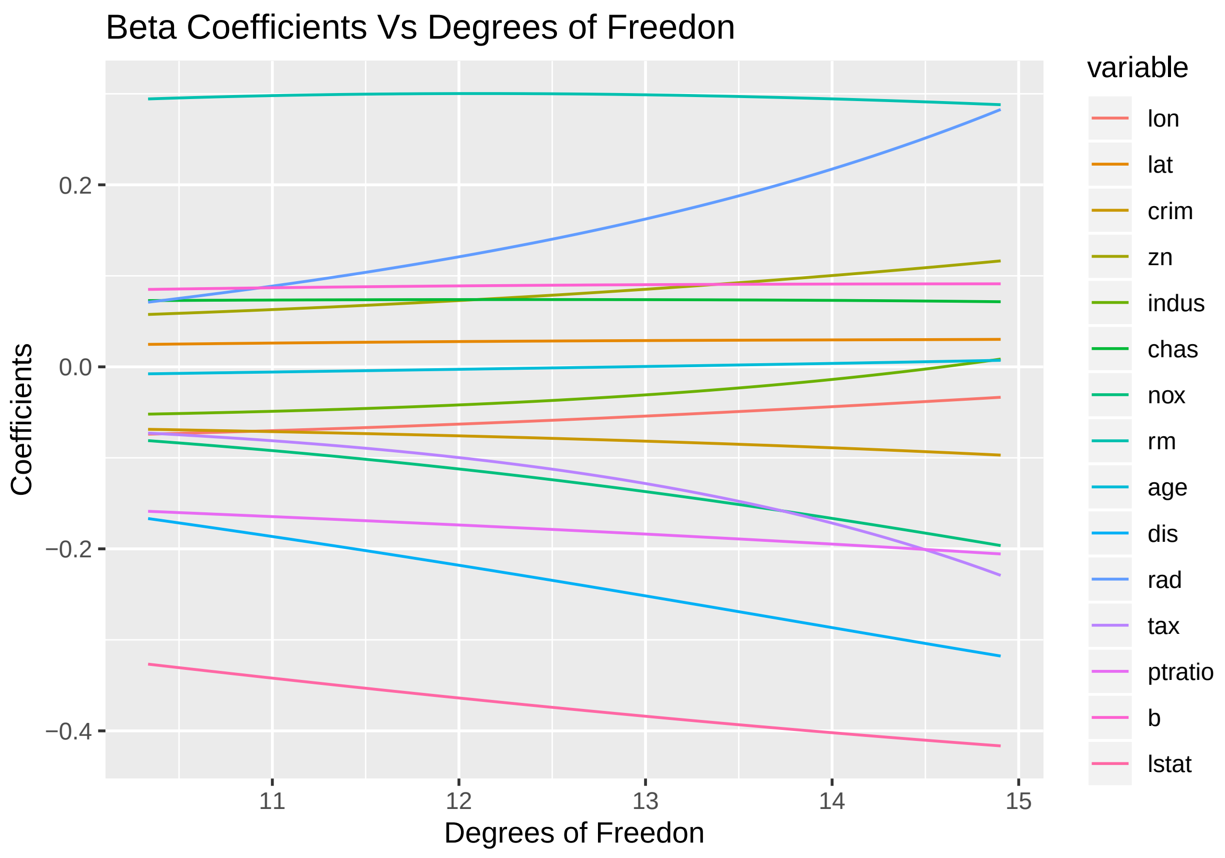
For this question, you can ONLY use the base package. We will use the dataset from Question 1 a). However, you should further standardize the outcome variable cmedv to mean 0 and sample variance 1. Hence, no intercept term is needed when you fit the model.

a. [30 points] First, fit a ridge regression with your own code, with the objective function

You should consider a grid of 100 penalty lambda values. Your choice of lambda grid can be flexible. However, they should be appropriate for the scale of the objective function. Calculate the degrees of freedom and the leave-one-out cross-validation (computationally efficient version) based on your choice of the penalty grid. Report details of your model fitting result. In particular, you should produce a plot similar to page 25 in the lecture note Penalized

```r
BostonHousing2 = BostonHousing2_original
BostonHousing2 = as.data.frame(apply(BostonHousing2, 2,
  function(x) {(x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)}))
y = data.matrix(subset(x = BostonHousing2, select = c(cmedv)))
x = data.matrix(subset(x = BostonHousing2, select = -c(cmedv)))
ridge = function(x, y, lambda){
  p = ncol(x)
  I = diag(p)
  beta_hat = solve((t(x) %*% x) + (lambda * I)) %*% t(x) %*% y
  dof = sum(diag((x %*% solve(t(x) %*% x + lambda * I) %*% t(x))))
  hat_matrix = x %*% solve((t(x) %*% x) + (lambda * I)) %*% t(x)
  y_hat = x %*% beta_hat
  cvv = rep(0, nrow(x))
  for (i in 1:nrow(x)) {
    cvv[i] = ((y[i] - y_hat[i]) / (1 - hat_matrix[i, i])) ^ 2
  }
  loocv = sum(cvv)
  ridge_df = cbind("df" = dof, data.frame(t(beta_hat)), "loocv" = loocv)
  return(ridge_df)
}
lambda = 1:100
ridge_grid_search = lapply(X = lambda, ridge, x = x, y = y)
ridge_grid_search_df = Reduce(rbind, ridge_grid_search)

# Plot the variables with df
ridge_grid_search_melt = melt(data = ridge_grid_search_df[, -ncol(ridge_grid_search_df)],
                              id.vars = "df")
ggplot(ridge_grid_search_melt, aes(x = df, y = value, colour = variable)) + geom_line() +
  ggtitle("Beta Coefficients Vs Degrees of Freedon") +
  labs(x = "Degrees of Freedon", y = "Coefficients")
```



b. [25 points] Following the setting of part a), with lambda = 10, recover the original solution based on the unstandardized data (with intercept). Again, you cannot refit the model on the original data. Provide a rigorous mathematical derivation. In this case, is your solution the same as a ridge model (with your previous code) fitted on the original data? Make sure that you check the model fitting results from either model. What is the difference? Please list all possible reasons that may cause this difference.

Mathematical Derivation

Standardized betas witho no intercept formula:

$$\frac{y_i - \mu_y}{\sigma_y} = \sum_{i=1}^{n} \frac{(x_i - \mu_i)}{\sigma_i}\beta_i$$

$$y_i - \mu_y = \sum_{i=1}^{n} \sigma_y \frac{(x_i - \mu_i)}{\sigma_i}\beta_i$$

$$y_i - \mu_y = \sum_{i=1}^{n} \sigma_y \frac{x_i}{\sigma_i}\beta_i - \sum_{i=1}^{n} \sigma_y \frac{\mu_i}{\sigma_i}\beta_i$$

$$y_i = \sum_{i=1}^{n} (\frac{\sigma_y \beta_i}{\sigma_i}) x_i - (\sum_{i=1}^{n} \sigma_y \frac{\mu_i}{\sigma_i}\beta_i + \mu_y)$$

Where,

- y_i : Un-standardized value of y
- μ_y : Mean of the dependent variable y
- σ_y : Standard deviation of the dependent variable y
- x_i : Un-standardized value of x
- μ_i : Mean of the in-dependent variable x for each covariate
- σ_i : Standard deviation of the in-dependent variable x for each covariate
- The first term tells us the way to calculate the unstandardized $\beta's$ from the standardized $\beta's$.
- The second term in the bracket gives us a way to calculate the intercept from the standardized $\beta's$

```r
new_y = data.matrix(subset(x = BostonHousing2_original, select = c(cmedv)))
new_x = data.matrix(subset(x = BostonHousing2_original, select = -c(cmedv)))
new_x = data.matrix(data.frame("intercept" = rep(1, nrow(x)), new_x))
lambda = 10
ridge_grid_search_10 = lapply(X = lambda, ridge, x = new_x, y = new_y)
ridge_grid_search_df_10 = Reduce(rbind, ridge_grid_search_10)
original_ridge_df = ridge_grid_search_df_10[, -c(1, ncol(ridge_grid_search_df_10))]
original_ridge_df = t(original_ridge_df)
original_ridge_df = data.frame("Variable" = row.names(original_ridge_df),
                               original_ridge_df)
```

```r
# Standardized value of the coefficients for lambda = 10
beta_std_10 = ridge_grid_search_df[10, -c(1, ncol(ridge_grid_search_df))]
sigma_y = sd(new_y)
sigma_x = apply(new_x[, -c(1)], 2, sd)
mean_x = apply(new_x[, -c(1)], 2, mean)
intercept = mean(new_y) - sum((((beta_std_10 * sigma_y) * mean_x) / sigma_x))
beta_un_std_ridge = (beta_std_10 * sigma_y) / sigma_x
beta_un_std_ridge = cbind.data.frame("intercept" = intercept, beta_un_std_ridge)
beta_un_std_ridge = t(beta_un_std_ridge)
beta_un_std_ridge = data.frame("Variable" = row.names(beta_un_std_ridge),
                               beta_un_std_ridge)
colnames(beta_un_std_ridge)[2] = "Un-Standardized Beta's (using formula)"
```

```r
ridge_calc_and_orig = merge(x = beta_un_std_ridge, y = original_ridge_df, by = "Variable", sort = FALSE
colnames(ridge_calc_and_orig)[3] = "Un-Standardized Beta's (Using code)"
kable(ridge_calc_and_orig, digits = 2, row.names = FALSE, align = "c") %>%
  kable_styling("striped", full_width = FALSE) %>%
  add_header_above(c("Ridge: Un-Standardized Betas (Using formula & Code)" = 3)) %>%
  column_spec(column = 1, bold = TRUE)
```

Ridge: Un-Standardized Betas (Using formula & Code)		
Variable	Un-Standardized Beta's (using formula)	Un-Standardized Beta's (Using code)
intercept	-520.42	-0.02
lon	-5.14	-0.31
lat	4.41	0.13
crim	-0.10	-0.10
zn	0.04	0.05
indus	-0.01	-0.04
chas	2.63	1.95
nox	-13.56	-2.36
rm	3.84	3.67
age	0.00	-0.01
dis	-1.27	-1.27
rad	0.24	0.28
tax	-0.01	-0.01
ptratio	-0.83	-0.77
b	0.01	0.01
lstat	-0.52	-0.57

Difference :

From the above table, we can see there is a significant difference in some of the beta coefficients. The reasons are as follows:

1. Due to the inclusion of the intercept term, the penalty of the ridge regression (without intercept) in not exactly same as the former. Hence, the difference.
2. Because β_0 - intercept, is not included using above equations. Hence, using the backtracking method to get the original coefficients, we are not taking into account of the effect by the intercept.

c. [15 points] A researcher is interested in only penalizing a subset of the variables, and leave the rest of them unpenalized. In particular, the categorical variables zn, chas, rad should not be penalized. You should use the data in part 2), which does not concern the intercept. Following the derivation during our lecture:

- Write down the objective function of this new model
- Derive the theoretical solution of this model and implement it with lambda = 10
- Check your model fitting results and report the residual sum of squares

Proof of regularization for selective variables

The Objective function of the new model is given below,

$$RSS + \sum_{j=0}^{n} \lambda_j \beta_j^2 = \sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \sum_{j=1}^{p} \lambda_j \beta_j^2$$

Where some of the $\lambda_j = 0$

In matrix form, it can be written as given below,

$$(Y - X\beta)^T (Y - X\beta) + \lambda \beta^T \beta$$

Taking the derivation and setting it to zero gives,

$$-2X^T(Y - X\beta) + 2\lambda\beta \ = 0$$

Hence, the solution for β becomes,

$$\hat{\beta} \ = (X^TX + \lambda)^{-1}X^TY$$

- p : Number of the covariates / variables
- n : Number of the observations
- Y : Vector of the dependent variable (n * 1)
- X : Matrix of the covariates (n * p)
- β : Vector of betas (p * 1)
- λ : Diagonal matrix of $\lambda's$ and $0's$ depending on the variable to regularize

```r
beta_std_10 = ridge_grid_search_df[10, -c(1, ncol(ridge_grid_search_df))]
beta_std_10[,] = 1
beta_std_10[, colnames(beta_std_10) %in% c("zn", "chas", "rad")] = 0
beta_std_10 = data.matrix(beta_std_10)
D = diag(as.vector(beta_std_10))
# Getting the old data
BostonHousing2 = BostonHousing2_original
BostonHousing2 = as.data.frame(apply(BostonHousing2, 2,
  function(x) {(x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)}))
y = data.matrix(subset(x = BostonHousing2, select = c(cmedv)))
x = data.matrix(subset(x = BostonHousing2, select = -c(cmedv)))

ridge = function(x, y, lambda, D){
  p = ncol(x)
  beta_hat = solve((t(x) %*% x) + (lambda * D)) %*% t(x) %*% y
  #dof = sum(diag((x %*% solve(t(x) %*% x + lambda * I) %*% t(x))))
  #hat_matrix = x %*% solve((t(x) %*% x) + (lambda * I)) %*% t(x)
  y_hat = x %*% beta_hat
  rss = sum((y - y_hat) ^ 2)
  ridge_df = cbind(data.frame(t(beta_hat)), "rss" = rss)
  return(ridge_df)
}
lambda = 10
ridge_rss = ridge(x, y, lambda = lambda, D)

# Model fitting results
df = t(ridge_rss[, -c(ncol(ridge_rss))])
df = data.frame("Variable" = row.names(df),
                            df)
colnames(df)[2] = "Beta Coefficients"
kable(df, digits = 2, row.names = FALSE, align = "c") %>%
  kable_styling("striped", full_width = FALSE) %>%
  add_header_above(c("Ridge Coefficients: Only selected variables" = 2)) %>%
  column_spec(column = 1, bold = TRUE)
```

Ridge Coefficients: Only selected variables	
Variable	Beta Coefficients
lon	-0.04
lat	0.03
crim	-0.09
zn	0.11
indus	0.00
chas	0.07
nox	-0.17
rm	0.29
age	0.01
dis	-0.29
rad	0.26
tax	-0.21
ptratio	-0.20
b	0.09
lstat	-0.41

The Residual Sum of Squares (RSS) for the model is 128.6341323
