HW03

Minyoung Do

2/4/2020

Egalitarianism and income

```r
gss_test <- read_csv("gss_test.csv")
gss_train <- read_csv("gss_train.csv")
```

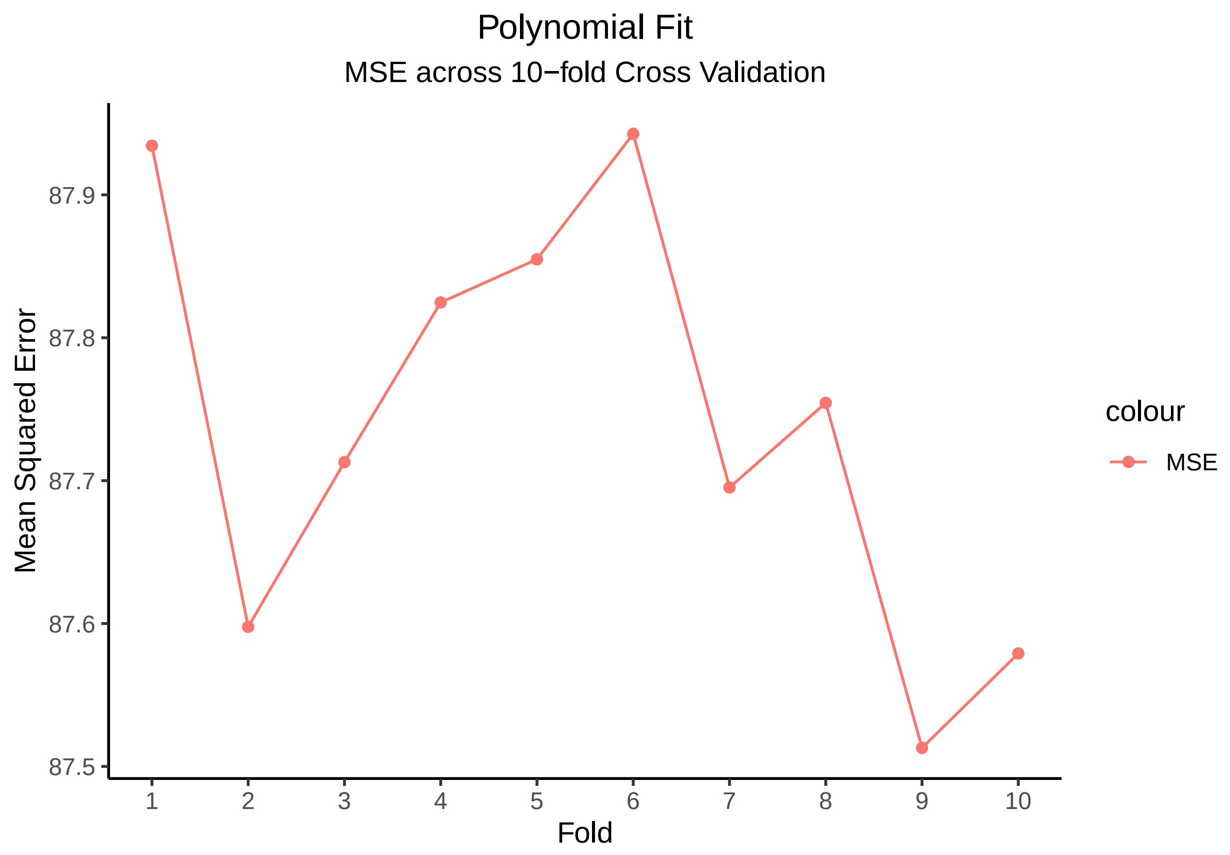
(20 points) Perform polynomial regression to predict egalit_scale as a function of income06. Use and plot 10-fold cross-validation to select the optimal degree d for the polynomial based on the MSE. Plot the resulting polynomial fit to the data, and also graph the average marginal effect (AME) of income06 across its potential values. Be sure to provide substantive interpretation of the results.

```r
# referenced this post on Stack Overflow: https://stackoverflow.com/questions/43686539/using-c

k <- 10
fold <- sample(k, nrow(gss_train), replace = TRUE)

## For each span from 1 to 10 we can calculate the CV test error:
mse <- numeric(k)
span <- seq(1, 10, by = 1)
cv <- numeric(length(span))

for (j in seq_along(span))
{
  for (i in seq_len(k))
  {
    take <- fold == i
    foldi <- gss_train[take, ]
    foldOther <- gss_train[!take, ]
    f <- glm(egalit_scale ~ poly(income06, span[j]), data=foldOther)
    pred <- predict(f, foldi)
    mse[i] <- mean((pred - foldi$egalit_scale)^2, na.rm = TRUE)
  }
  cv[j]<- mean(mse)
  result <- cbind(span, cv) %>%
    as.tibble() %>%
    rename(Fold = "span", MeanMSE = "cv")
}
```

```r
result %>%
  kable() %>%
  kable_styling(full_width = F)
```

Fold	MeanMSE
1	87.93442
2	87.59767
3	87.71297
4	87.82472
5	87.85491
6	87.94272
7	87.69522
8	87.75443
9	87.51300
10	87.57905

```r
ggplot(result, aes(Fold, MeanMSE, color = "MSE")) +
  geom_point() +
  geom_line() +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5)) +
  scale_x_continuous(breaks=c(1:10)) +
  labs(title = "Polynomial Fit",
       subtitle = "MSE across 10-fold Cross Validation",
       x = "Fold",
       y = "Mean Squared Error")
```

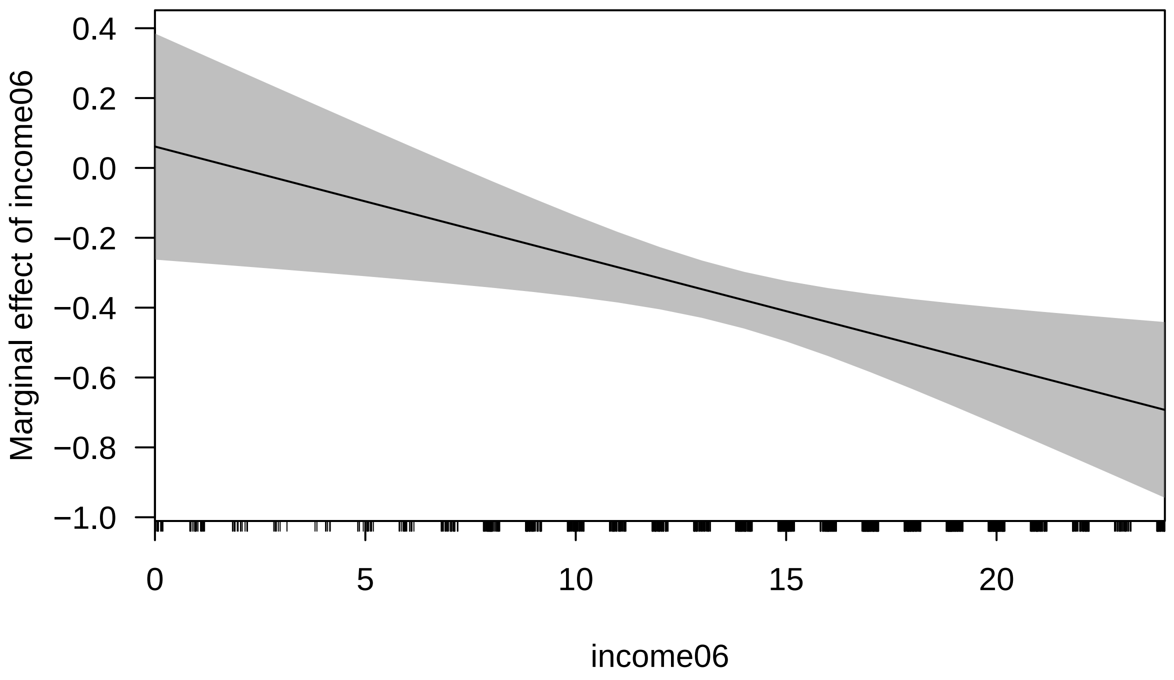


As seen in the graph, the second order polynomial fits the data best with the mean MSE value of 88.01.

```r
poly_ame <- lm(egalit_scale ~ income06 + I(income06^2), data = gss_train)
margins(poly_ame)
```

```
##  income06
##   -0.4303
```

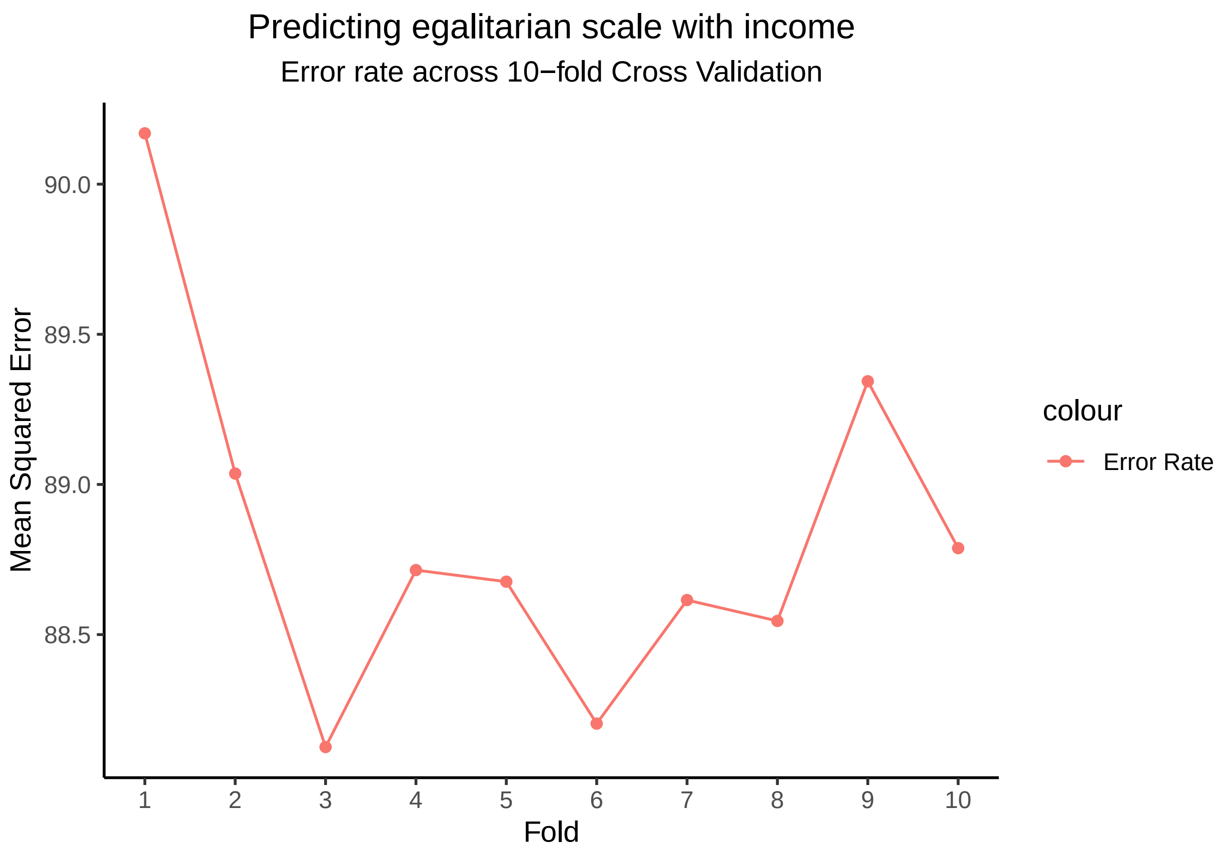
```r
cplot(object = poly_ame, x = "income06", what = "effect")
```



This graph shows the marginal effect of `income06` in a negative line, which indicates that the marginal effect of `income06` decreases as the income level increases. The average marginal effect is -0.43, which corresponds with the graph with a negative slope.

(20 points) Fit a step function to predict egalit_scale as a function of income06, and perform 10-fold cross-validation to choose the optimal number of cuts. Plot the fit and interpret the results.

```
# referenced this Stack Overflow post: https://stackoverflow.com/questions/42190337/cross-vali

step_fun <- vector(mode = "numeric", length = 10)

for (i in 1:10) {
  cuts <- levels(cut(gss_train$income06, i + 1))
  breaks <- unique(c(as.numeric(sub("\\((.+),.*", "\\1", cuts)),
                  as.numeric(sub("[^,]*,([^]]*)\\]", "\\1", cuts))))
  interval <- glm(egalit_scale ~ cut(income06, unique(breaks)), data = gss_train)
  step_fun[i] <- cv.glm(gss_train, interval, K = 10)$delta[1]
  error_result <- step_fun %>%
    as.tibble() %>%
    mutate(id = seq_len(n()))
}

error_result %>%
  ggplot() +
  geom_point(aes(id, value, color = "Error Rate")) +
  geom_line(aes(id, value, color = "Error Rate")) +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5)) +
  scale_x_continuous(breaks=c(1:10)) +
  labs(title = "Predicting egalitarian scale with income",
       subtitle = "Error rate across 10-fold Cross Validation",
       x = "Fold",
       y = "Mean Squared Error")
```



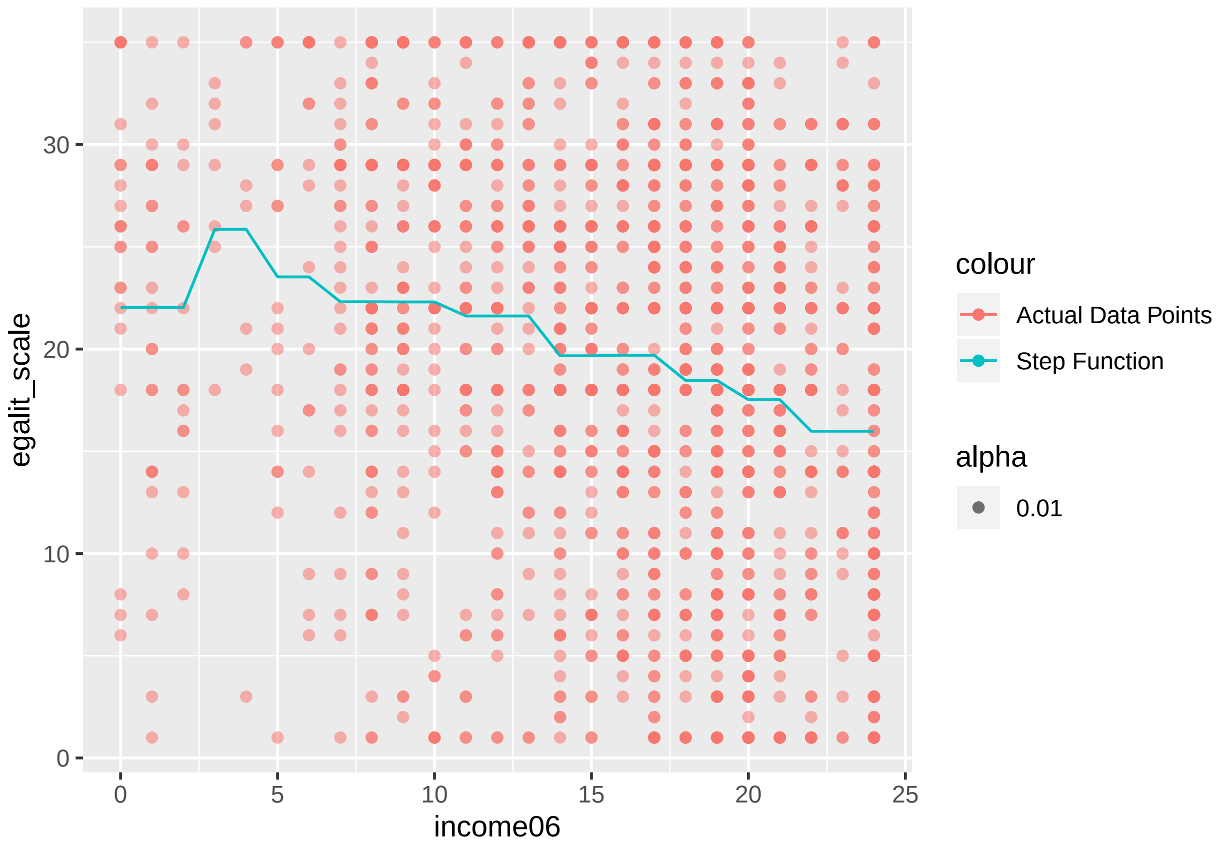
```
min(error_result[,1])
```

```
## [1] 88.12559
```

The graph suggests that the 6 cuts are the optimal number for the fit, with the error rate of 88.28 out of 10 models. Let's take a closer look at this model:

```
step_cut <- levels(cut(gss_train$income06, i + 1))
step_breaks <- unique(c(as.numeric(sub("\\((.+),.*", "\\1", cuts)),
                        as.numeric(sub("[^,]*,([^]]*)\\]", "\\1", cuts))))
step_fit <- glm(egalit_scale ~ cut(income06, unique(breaks)), data = gss_train)

step_pred <- predict(step_fit, gss_train)
```

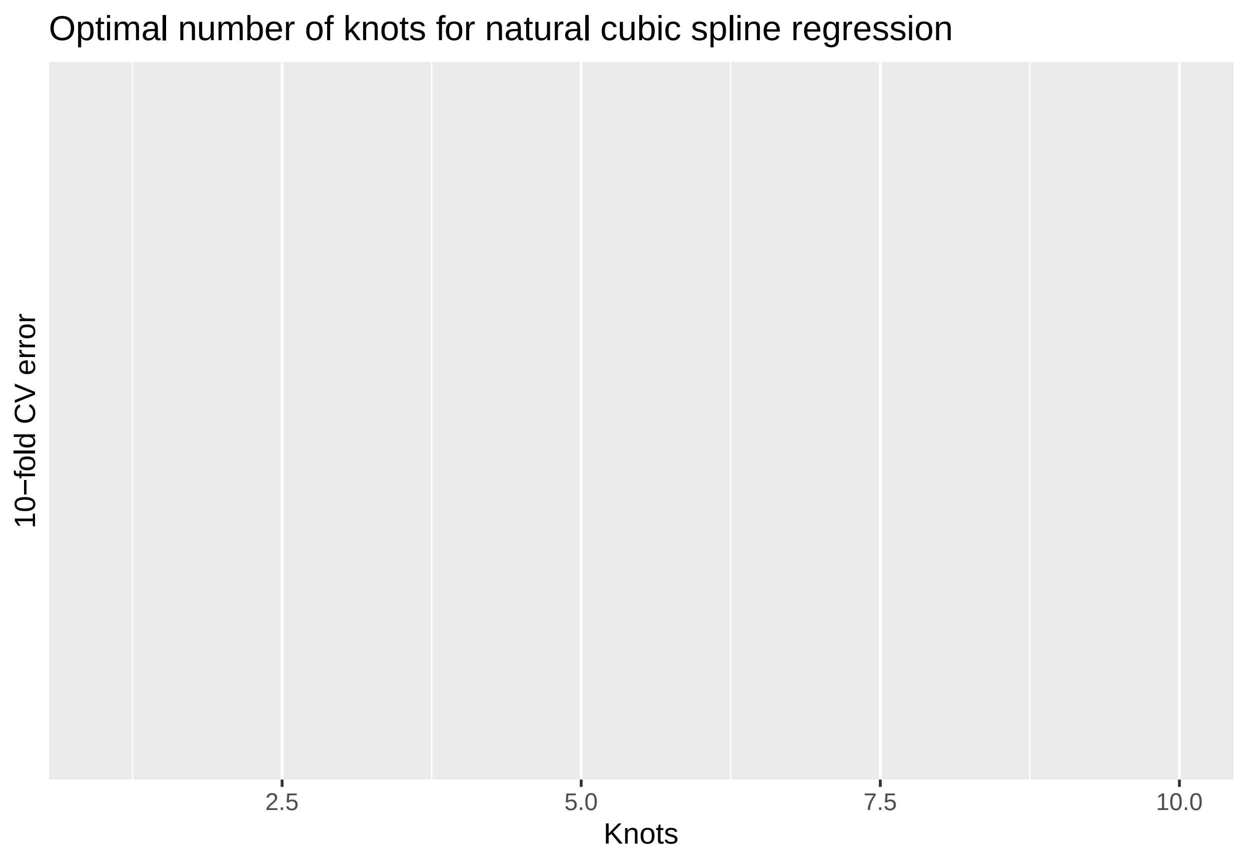
```r
step_df <- tibble(step_pred, gss_train$income06) %>%
  rename(income = "gss_train$income06")

ggplot() +
  geom_point(data = gss_train, aes(income06, egalit_scale, alpha = 0.01, color = "Actual Data P
  geom_line(data = step_df, aes(income, step_pred, color = "Step Function"))
```



The line graph shows how many bins there are for the `income06` variable. There are 9 bins total, and there is a decreasing pattern overall as the income increases.

(20 points) Fit a natural regression spline to predict egalit_scale as a function of income06. Use 10-fold cross-validation to select the optimal number of degrees of freedom, and present the results of the optimal model.

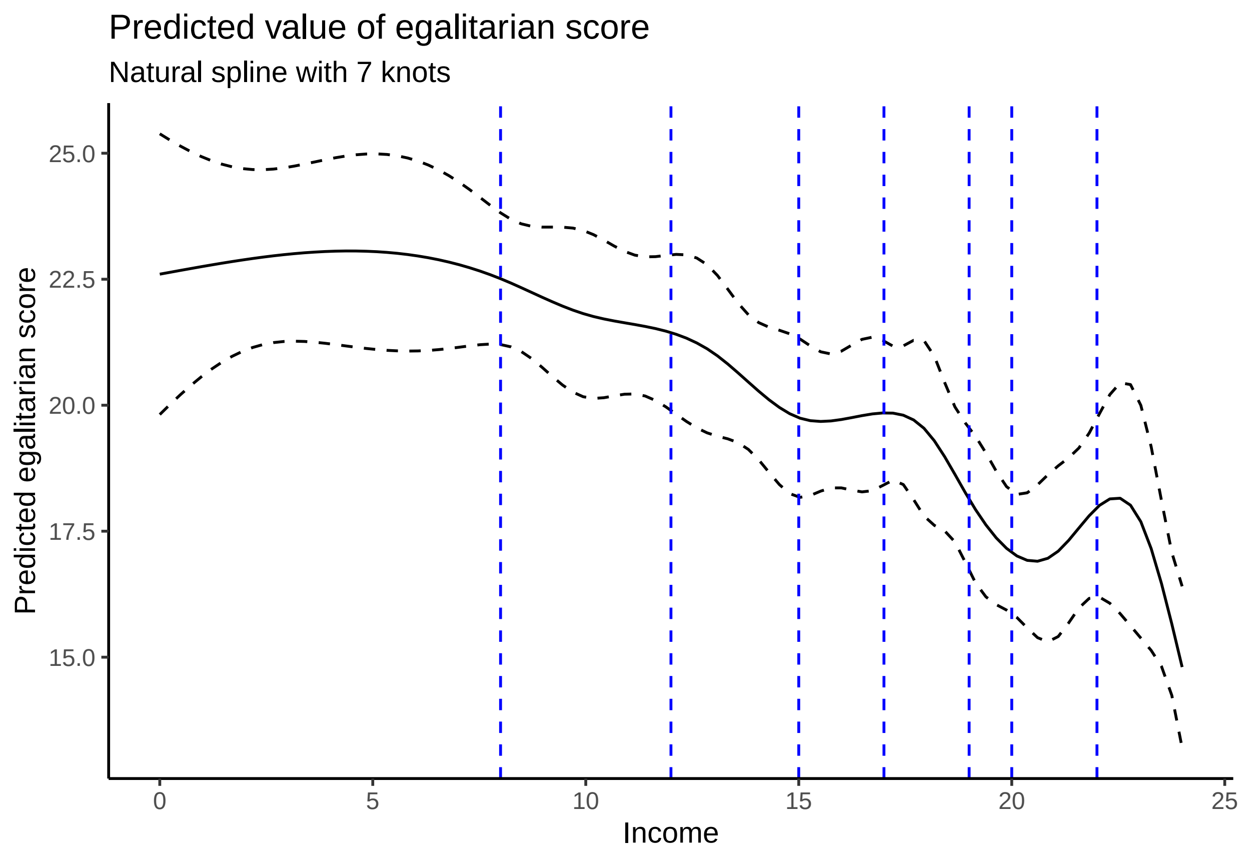
```r
# function to simplify things
egalit_spline <- function(splits, df = NULL){
  # estimate the model on each fold
  model <- glm(egalit_scale ~ ns(income06, df = df),
               data = analysis(splits))

  model_acc <- augment(model, newdata = assessment(splits)) %>%
    mse(egalit_scale, estimate = .fitted)
  mean(model_acc$.estimate)
}

knots_fun <- function(splits, knots){
  egalit_spline(splits, df = knots + 3)
}

# estimate CV error for knots in 0:10
results <- vfold_cv(gss_train, v = 10)

tidyr::expand(results, id, knots = 1:10) %>%
  left_join(results) %>%
  mutate(acc = map(splits, knots, tune_over_knots)) %>%
  group_by(knots) %>%
  summarize(acc = mean(acc)) %>%
  mutate(err = 1 - acc) %>%
  ggplot(aes(knots, err)) +
  geom_point() +
  geom_line() +
  scale_y_continuous(labels = scales::percent) +
  labs(title = "Optimal number of knots for natural cubic spline regression",
       x = "Knots",
       y = "10-fold CV error")
```



```r
df_glm <- glm(egalit_scale ~ ns(income06, df = 10), data = gss_train) %>%
  cplot("income06", what = "prediction", n = 100, draw = FALSE) %>%
  as.tibble()
```

```
##         xvals    yvals    upper    lower
## 1  0.0000000 22.60034 25.38579 19.81488
## 2  0.2424242 22.63784 25.25913 20.01656
## 3  0.4848485 22.67513 25.14081 20.20944
## 4  0.7272727 22.71197 25.03245 20.39149
## 5  0.9696970 22.74814 24.93568 20.56061
## 6  1.2121212 22.78343 24.85218 20.71468
## 7  1.4545455 22.81761 24.78349 20.85173
## 8  1.6969697 22.85046 24.73086 20.97006
## 9  1.9393939 22.88175 24.69505 21.06845
## 10 2.1818182 22.91127 24.67619 21.14636
## 11 2.4242424 22.93879 24.67354 21.20405
## 12 2.6666667 22.96410 24.68556 21.24264
## 13 2.9090909 22.98696 24.70994 21.26399
## 14 3.1515152 23.00717 24.74380 21.27054
## 15 3.3939394 23.02448 24.78395 21.26502
## 16 3.6363636 23.03870 24.82712 21.25027
## 17 3.8787879 23.04958 24.87012 21.22904
## 18 4.1212121 23.05692 24.90995 21.20388
## 19 4.3636364 23.06048 24.94389 21.17707
## 20 4.6060606 23.06005 24.96950 21.15060
```

```r
df_glm %>%
  ggplot(aes(x = xvals)) +
  geom_line(aes(y = yvals)) +
  geom_line(aes(y = upper), linetype = 2) +
  geom_line(aes(y = lower), linetype = 2) +
  theme_classic() +
  geom_vline(xintercept = attr(bs(gss_train$income06, df = 10), "knots"),
             linetype = 2, color = "blue") +
  labs(title = "Predicted value of egalitarian score",
       subtitle = "Natural spline with 7 knots",
       x = "Income",
       y = "Predicted egalitarian score")
```



We can observe a similar pattern found in the polynomial regression and step-function-applied model, i.e., the decreasing pattern of egalitarian scale with increasing income levels. This natural regression model has 7 knots total, whereas the step function model has 9 bins. Despite the smaller number of knots compared to 9 bins, this regression line represents the data better with a smoother line.

Egalitarianism and everything

(20 points total) Estimate the following models using all the available predictors (be sure to perform appropriate data pre-processing (e.g., feature standardization) and hyperparameter tuning (e.g. lambda for PCR/PLS, lambda and alpha for elastic net). Also use 10-fold cross-validation for each model to estimate the model's performance using MSE):

a. (5 points) Linear regression

```
ten_fold <- trainControl(method = "cv", number = 10)

gss_lm <- train(egalit_scale ~ ., data = gss_train,
                method = "lm", trControl = ten_fold)
```

```r
pred_train <- predict(gss_lm, gss_test)
Metrics::mse(gss_test$egalit_scale, pred_train)
```

```
## [1] 63.21363
```

b. (5 points) Elastic net regression

```r
gss_train_x <- model.matrix(egalit_scale ~ ., gss_train)[, -1]
gss_train_y <- gss_train$egalit_scale

gss_test_x <- model.matrix(egalit_scale ~ ., gss_test)[, -1]
gss_test_y <- gss_test$egalit_scale

# Now, more efficient grid search for varying alpha
fold_id <- sample(1:10, size = length(gss_train_y), replace = TRUE) # maintain the same folds

# search across a range of alphas
tuning_grid <- tibble::tibble(
  alpha      = seq(0, 1, by = .1),
  mse_min    = NA,
  mse_1se    = NA,
  lambda_min = NA,
  lambda_1se = NA
)

for(i in seq_along(tuning_grid$alpha)) {
  # fit CV model for each alpha value
  fit <- cv.glmnet(gss_train_x,
                  gss_train_y,
                  alpha = tuning_grid$alpha[i],
                  foldid = fold_id)

  # extract MSE and lambda values
  tuning_grid$mse_min[i]    <- fit$cvm[fit$lambda == fit$lambda.min]
  tuning_grid$mse_1se[i]    <- fit$cvm[fit$lambda == fit$lambda.1se]
  tuning_grid$lambda_min[i] <- fit$lambda.min
  tuning_grid$lambda_1se[i] <- fit$lambda.1se
}

# minimum MSE
min(tuning_grid[,2])
```

```
## [1] 60.09246
```

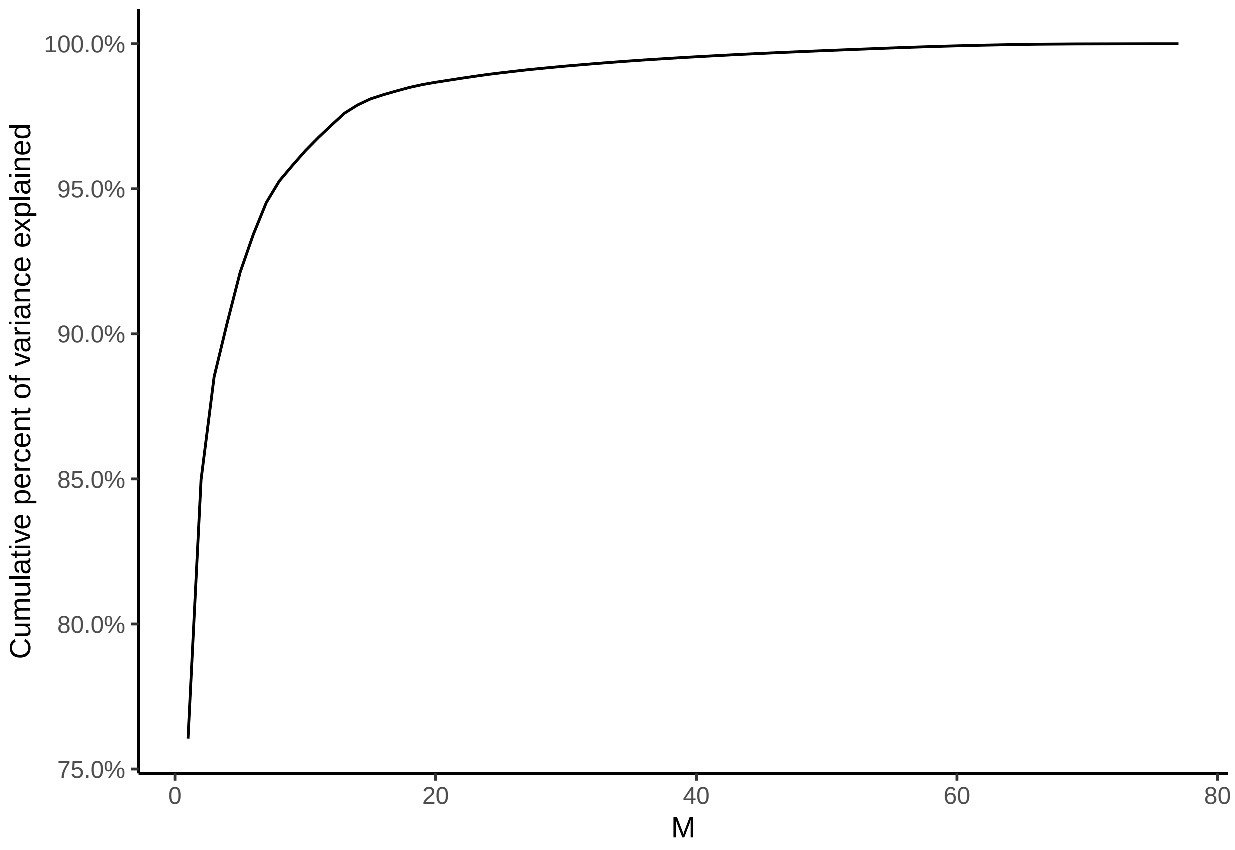
```r
# minimum lambda
min(tuning_grid[,4])
```

```
## [1] 0.2127845
```

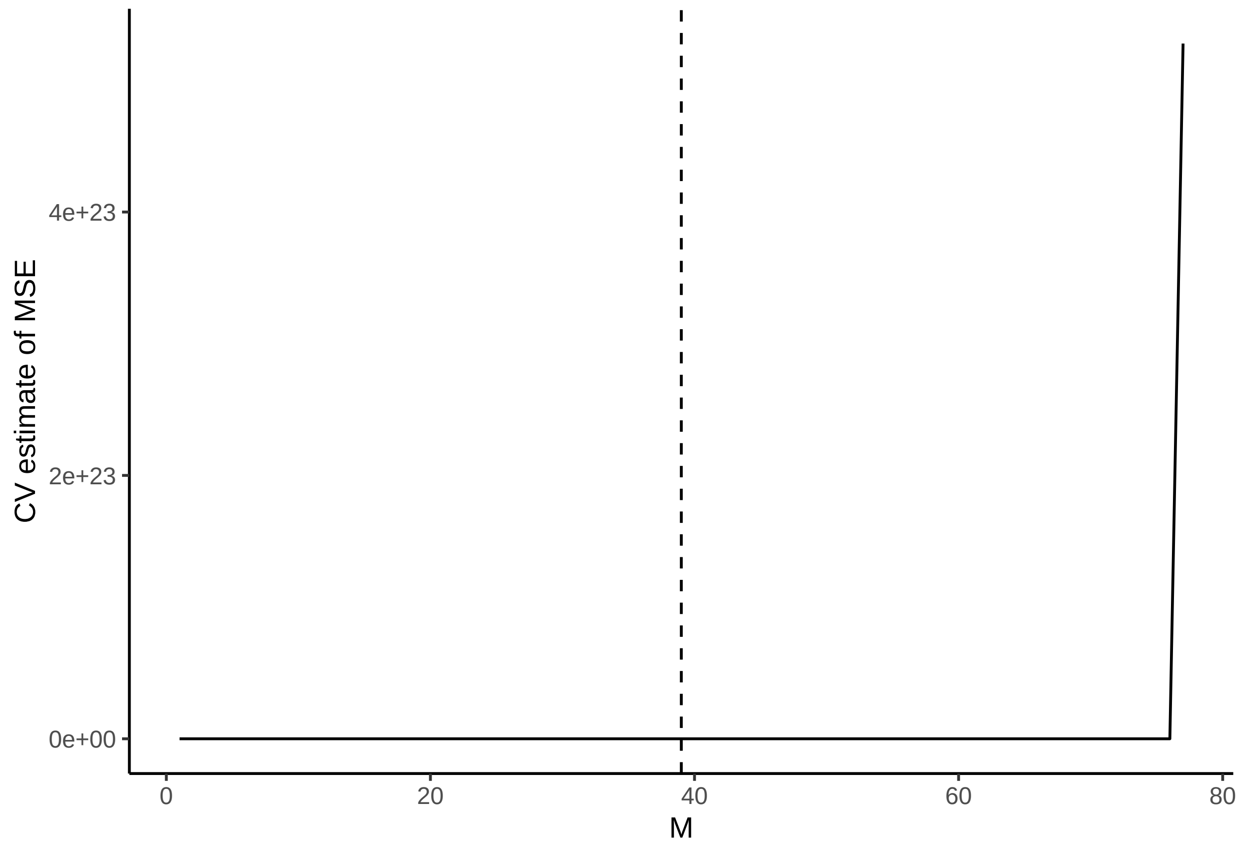
```r
# combination of alpha = 1 & lambda = 0.2127845
coef_en <- coef(fit, alpha = 1)
```

c. (5 points) Principal component regression

```r
# running pcr
gss_pcr <- pcr(egalit_scale ~ .,
                # just grabbing numeric features
                data = select_if(gss_train, is.numeric),
                center = TRUE,
                validation = "CV",
                segments = 10)

# extracting gss_exp, mse, pc, and cum_exp
gss_pcr_stats <- tibble(
  gss_exp = loadings(gss_pcr) %>%
    attr("explvar"),
  mse = as.vector(MSEP(gss_pcr, estimate = "CV", intercept = FALSE)$val)
) %>%
  mutate(pc = row_number(),
         cum_exp = cumsum(gss_exp) / 100)

# percent of variance explained by each model
ggplot(gss_pcr_stats, aes(pc, cum_exp)) +
  geom_line() +
  scale_y_continuous(labels = scales::percent) +
  theme_classic() +
  labs(x = expression(M),
       y = "Cumulative percent of variance explained")
```



```r
# MSE
ggplot(gss_pcr_stats, aes(pc, mse)) +
  geom_line() +
  geom_vline(xintercept = which.min(gss_pcr_stats$mse), linetype = 2) +
  theme_classic() +
  labs(x = expression(M),
       y = "CV estimate of MSE")
```



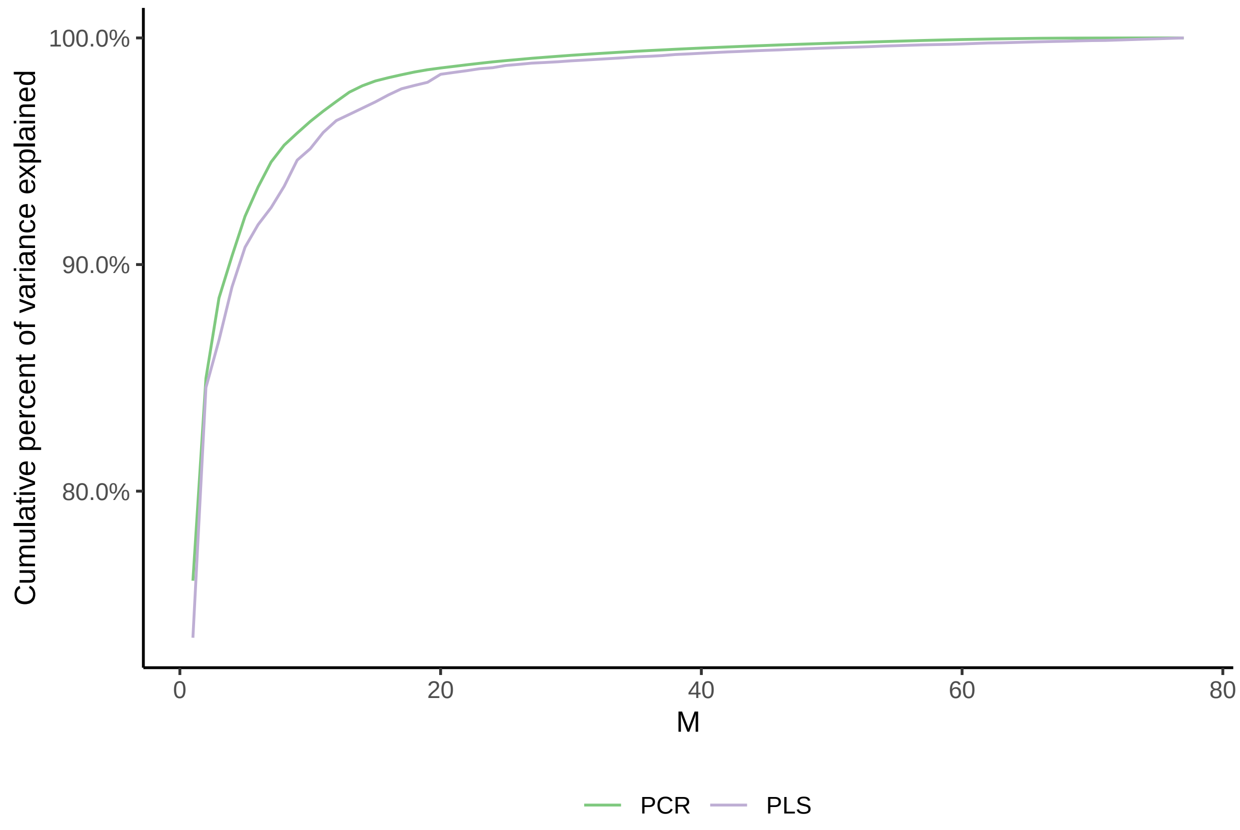
This tuning process tells us that the MSE and proportion of variance doesn't improve anymore after the number of principal components passes a certain point: the optimal number is 40.

d. (5 points) Partial least squares regression

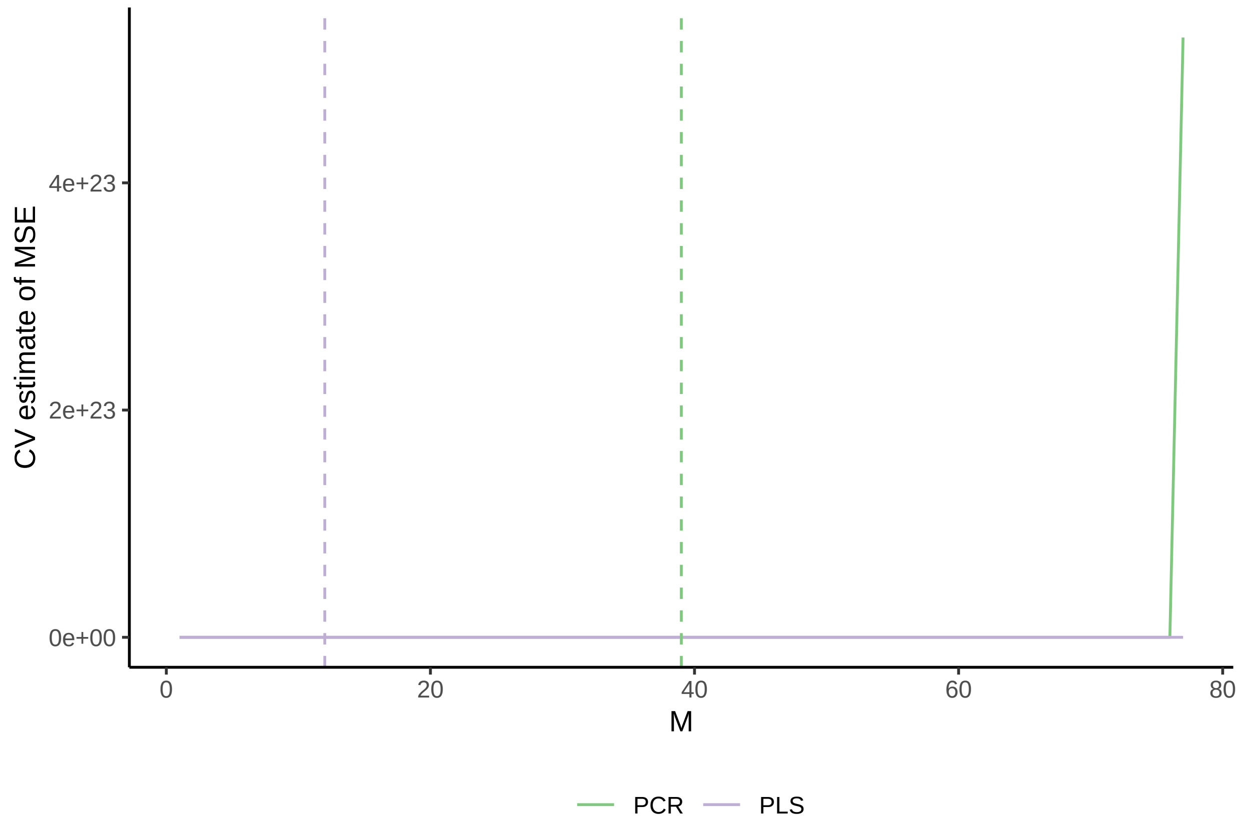
```
# running pls
gss_pls <- plsr(egalit_scale ~ .,
                data = select_if(gss_train, is.numeric),
                center = TRUE,
                validation = "CV",
                segments = 10)

# extracting numbers we need
gss_pls_stats <- tibble(
  pct_exp = loadings(gss_pls) %>% attr("explvar"),
  mse = as.vector(MSEP(gss_pls, estimate = "CV", intercept = FALSE)$val)
) %>%
  mutate(pc = row_number(),
         cum_exp = cumsum(pct_exp) / 100)
```

(20 points) For each final tuned version of each model fit, evaluate feature importance by generating feature interaction plots. Upon visual presentation, be sure to discuss the substantive results for these models and in comparison to each other (e.g., talk about feature importance, conditional effects, how these are ranked differently across different models, etc.).

```r
gss_stats <- bind_rows(PCR = gss_pcr_stats,
                       PLS = gss_pls_stats,
                       .id = "model")

# percent of variance explained by M
ggplot(gss_stats, aes(pc, cum_exp, color = model)) +
  geom_line() +
  scale_y_continuous(labels = scales::percent) +
  scale_color_brewer(type = "qual") +
  theme_classic() +
  labs(x = expression(M),
       y = "Cumulative percent of variance explained",
       color = NULL) +
  theme(legend.position = "bottom")
```



```r
# PCR explains more variance with less coefficients

# MSE comparison
ggplot(gss_stats, aes(pc, mse, color = model)) +
  geom_line() +
  geom_vline(data = tribble(
    ~pc, ~model,
    which.min(gss_pls_stats$mse), "PLS",
    which.min(gss_pcr_stats$mse), "PCR"
  ),
  aes(xintercept = pc, color = model), linetype = 2, show.legend = FALSE) +
  scale_color_brewer(type = "qual") +
  theme_classic() +
  labs(x = expression(M),
       y = "CV estimate of MSE",
       color = NULL) +
  theme(legend.position = "bottom")
```



The graph above compares the last two models, PCR and PLS. They both perform similarly, but in between 0 and 20, PCR explains the variance of the data a little better, reaching the maximum percent of variance a little before PLS. However, in the MSE plot, PLS is slightly better as its best fit is at 20, while PCR has the best fit when $M = 40$. Both simple linear regression and elastic net has way higher MSE values. In this analysis, PCR and PLS have a better prediction accuracy of the data than linear regression and elastic net.