Homework 3

Daniel Dulaney

September 18, 2020

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------------------------- tidymodels 0.1.1 --
```

```
## v broom     0.7.0      v recipes   0.1.13
## v dials     0.0.8      v rsample   0.0.7
## v infer     0.5.3      v tune      0.1.1
## v modeldata 0.0.2      v workflows 0.1.3
## v parsnip   0.1.3      v yardstick 0.0.7
```

```
## -- Conflicts ---------------------------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
```

```r
library(leaps)
library(ISLR)
library(gam)
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loaded gam 1.20
```

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

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

```
knitr::opts_chunk$set(warning = FALSE)
```

Chapter 6

(8)

(a)

```
x <- rnorm(100)
e <- rnorm(100, mean = 0, sd = .01)
```

(b)

```
x2 <- x^2
x3 <- x^3

beta_0 <- 5
beta_1 <- 8
beta_2 <- 2
beta_3 <- 10

y <- beta_0 + (beta_1* x) + (beta_2 * x2) + (beta_3 * x3) + e

# so y is related to x, x^2, and x^3 but not to the additional x^i's that'll be included in (c). should be that o
nly these 3 x's are important variables moving forward
```

(c)

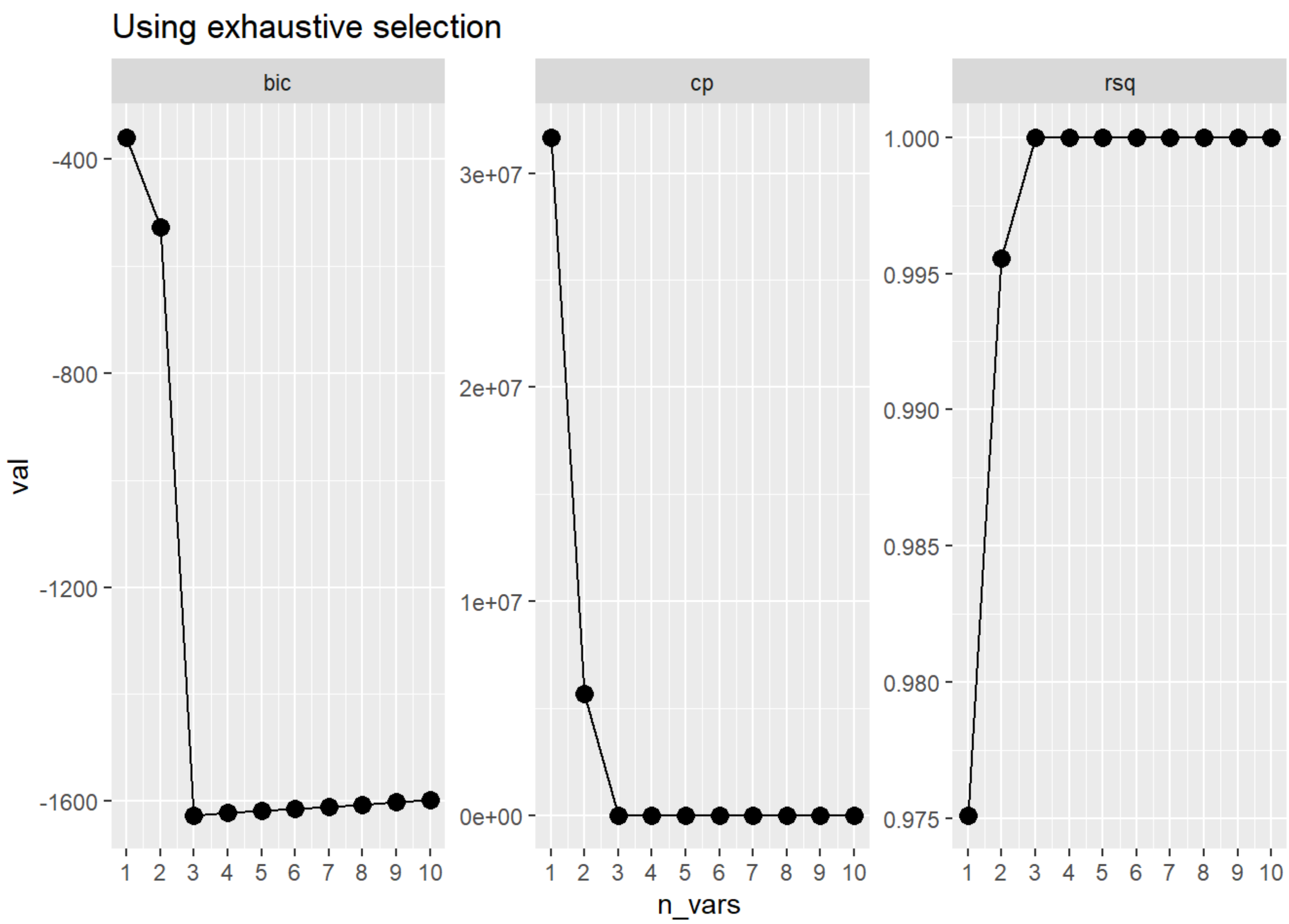
```
df <- cbind(x, x2, x3, e, y) %>%
  as.data.frame()
```

```
# function that takes in regsubsets() output, then plots metrics over various # variable values
plot_subset_metrics <- function(model, max_subset, method) {

  cp <- model$cp
  bic <- model$bic
  rsq <- model$rsq

  metrics <- cbind(n_vars = 1:max_subset, cp, bic, rsq) %>%
    as_tibble()

  metrics %>%
    pivot_longer(cols = cp:rsq, names_to = "metric", values_to = "val") %>%
    ggplot(aes(n_vars, val)) +
    geom_path() +
    geom_point(size = 3) +
    facet_wrap(~ metric, scales = "free") +
    scale_x_continuous(breaks = 1:max_subset) +
    labs(title = str_c("Using", method, "selection", sep = " "))

}
```

```
df_x10 <- poly(df$x, 10, raw = TRUE) %>%
  as_tibble() %>%
  rename_with(function(num) str_c("x", num)) %>%
  cbind(y)
```

```
exhaustive_mod <- regsubsets(y ~ .,
                       data = df_x10,
                       nvmax = 10,
```

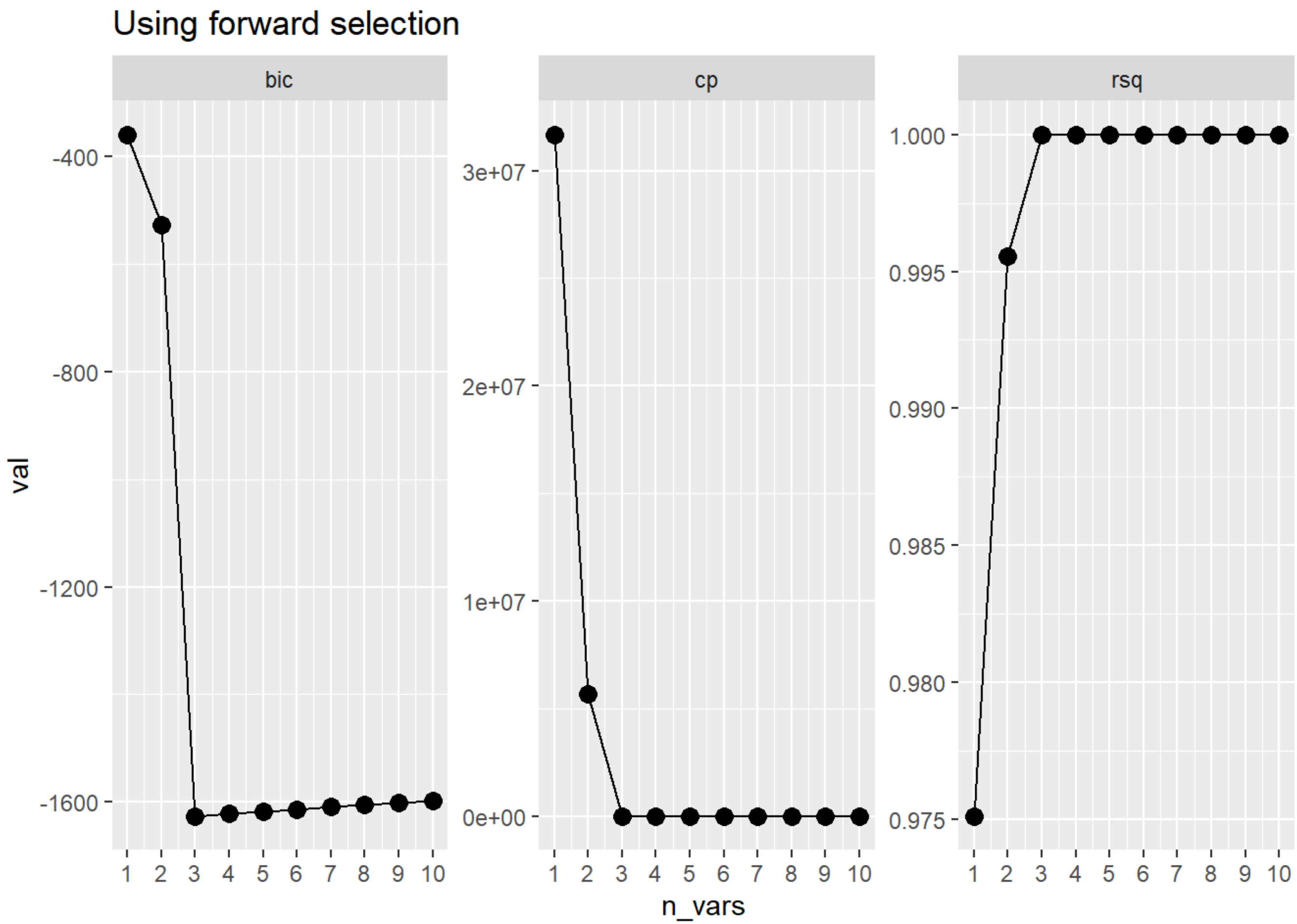
```
                          method = "exhaustive") %>%
  summary()

plot_subset_metrics(exhaustive_mod, max_subset = 10, method = "exhaustive")
```



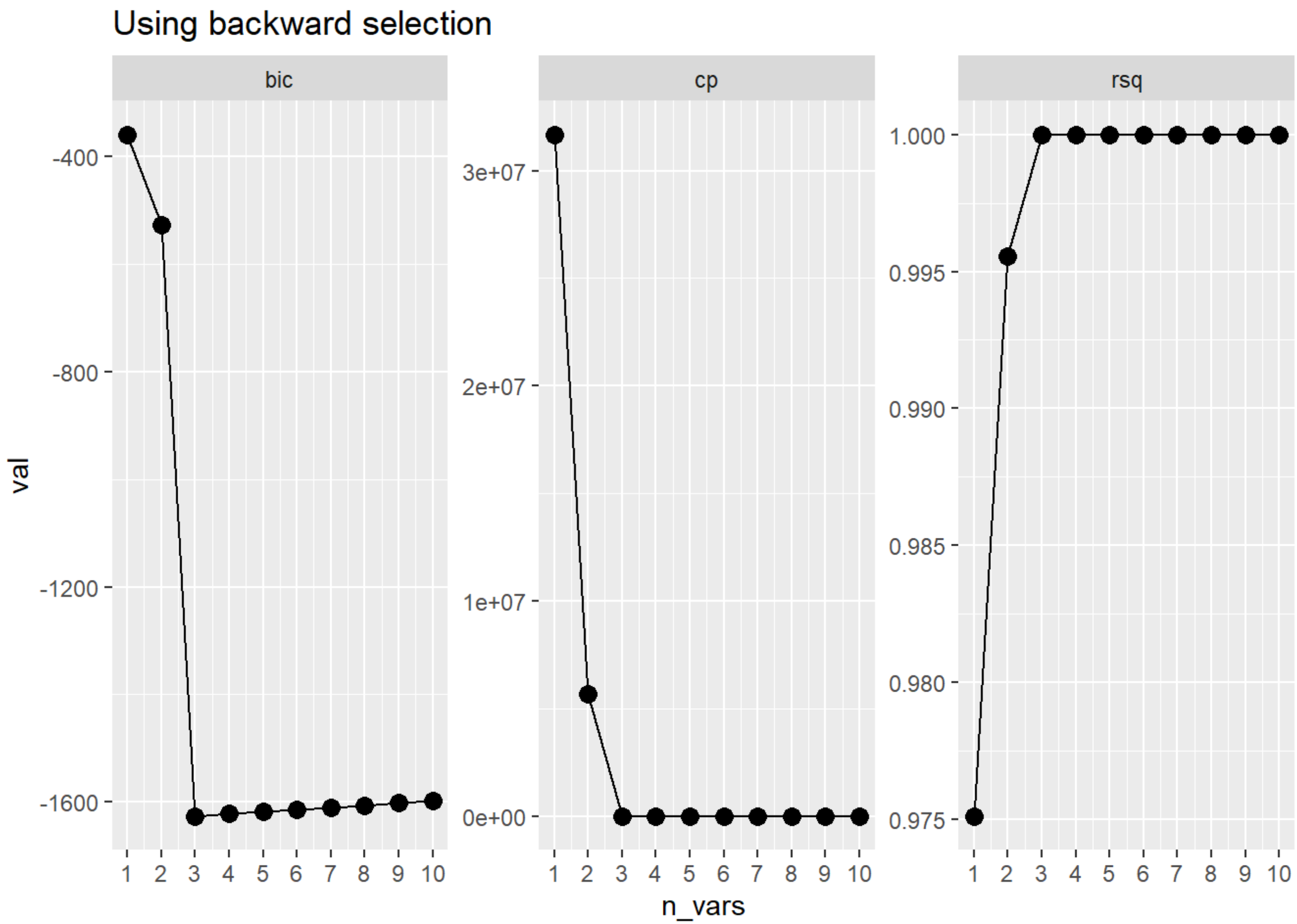
The best model appears to be the 3-variable one, as should be the case!

(d)

```r
forward_mod <- regsubsets(y ~ .,
                          data = df_x10,
                          nvmax = 10,
                          method = "forward") %>%
  summary()

plot_subset_metrics(forward_mod, max_subset = 10, method = "forward")
```



```
backward_mod <- regsubsets(y ~ .,
                           data = df_x10,
                           nvmax = 10,
                           method = "backward") %>%
  summary()

plot_subset_metrics(backward_mod, max_subset = 10, method = "backward")
```



Answer is the same. 3-variable model is best, as it should be!

(e)

```
x10_rec <- recipe(y ~ ., data = df_x10) %>%
  step_normalize(all_numeric())

wf <- workflow() %>%
  add_recipe(x10_rec)
```

```
# initialize a LASSO model with penalty (lambda) to be tuned through cross-valiation
tune_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")

# create a grid of lambda values to test
lambda_grid <- grid_regular(penalty(), levels = 50)

lambda_grid
```

```
## # A tibble: 50 x 1
##     penalty
##       <dbl>
##  1 1.00e-10
##  2 1.60e-10
##  3 2.56e-10
##  4 4.09e-10
##  5 6.55e-10
##  6 1.05e- 9
##  7 1.68e- 9
##  8 2.68e- 9
##  9 4.29e- 9
## 10 6.87e- 9
## # ... with 40 more rows
```

```
# create CV folds
x10_folds <- vfold_cv(df_x10)
```

```r
lasso_grid <- tune_grid(wf %>% add_model(tune_spec),
                        resamples = x10_folds,
                        grid = lambda_grid)
```

```
## ! Fold01: internal: A correlation computation is required, but `estimate` is const...
```

```
## ! Fold02: internal: A correlation computation is required, but `estimate` is const...
```

```
## ! Fold03: internal: A correlation computation is required, but `estimate` is const...
```

```
## ! Fold04: internal: A correlation computation is required, but `estimate` is const...
```

```
## ! Fold05: internal: A correlation computation is required, but `estimate` is const...
```

```
## ! Fold06: internal: A correlation computation is required, but `estimate` is const...
```

```
## ! Fold07: internal: A correlation computation is required, but `estimate` is const...
```

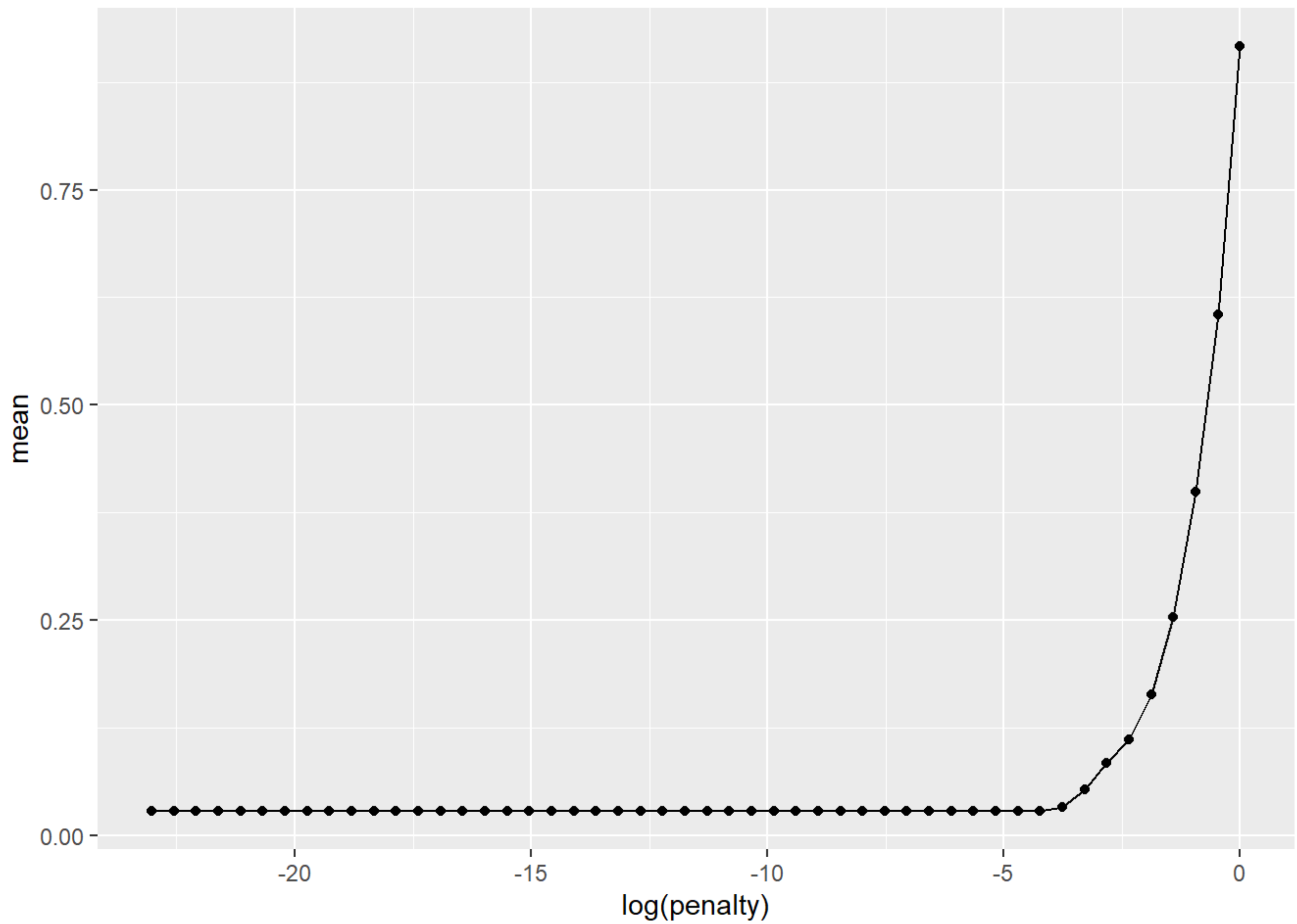
```
## ! Fold08: internal: A correlation computation is required, but `estimate` is const...
```

```
## ! Fold09: internal: A correlation computation is required, but `estimate` is const...
```

```
## ! Fold10: internal: A correlation computation is required, but `estimate` is const...
```

```r
# plot cross-validated RMSE over many values of lambda (log-scale to improve visibility)
lasso_grid %>%
  collect_metrics() %>%
  filter(.metric == "rmse") %>%
  ggplot(aes(log(penalty), mean)) +
```

```
  geom_path() +
  geom_point()
```




```
# which penalty performed best?
best_lambda <- lasso_grid %>%
  collect_metrics() %>%
  filter(.metric == "rmse") %>%
  arrange(mean) %>%
  slice(1) %>%
```

```
  pull(penalty)

best_lambda
```

```
## [1] 1e-10
```

The smallest penalty in this tuning parameter grid ends up being the best choice of λ

```
# fit final model w/ lambda ~ 0 and get coefficients
final_mod <- linear_reg(penalty = best_lambda, mixture = 1) %>%
  set_engine("glmnet") %>%
  fit(y ~ ., data = df_x10)
```

```
final_mod %>%
  tidy()
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.0-2
```

```
## # A tibble: 11 x 3
##    term        estimate      penalty
##    <chr>          <dbl>        <dbl>
##  1 (Intercept)     5.60 0.0000000001
##  2 x1              7.90 0.0000000001
```

```
##   3 x2             1.37 0.0000000001
##   4 x3             9.76 0.0000000001
##   5 x4             0    0.0000000001
##   6 x5             0    0.0000000001
##   7 x6             0    0.0000000001
##   8 x7             0    0.0000000001
##   9 x8             0    0.0000000001
##  10 x9             0    0.0000000001
##  11 x10            0    0.0000000001
```

Only the intercept and `x1` - `x3` coefficients are meaningfully above 0.

(f)

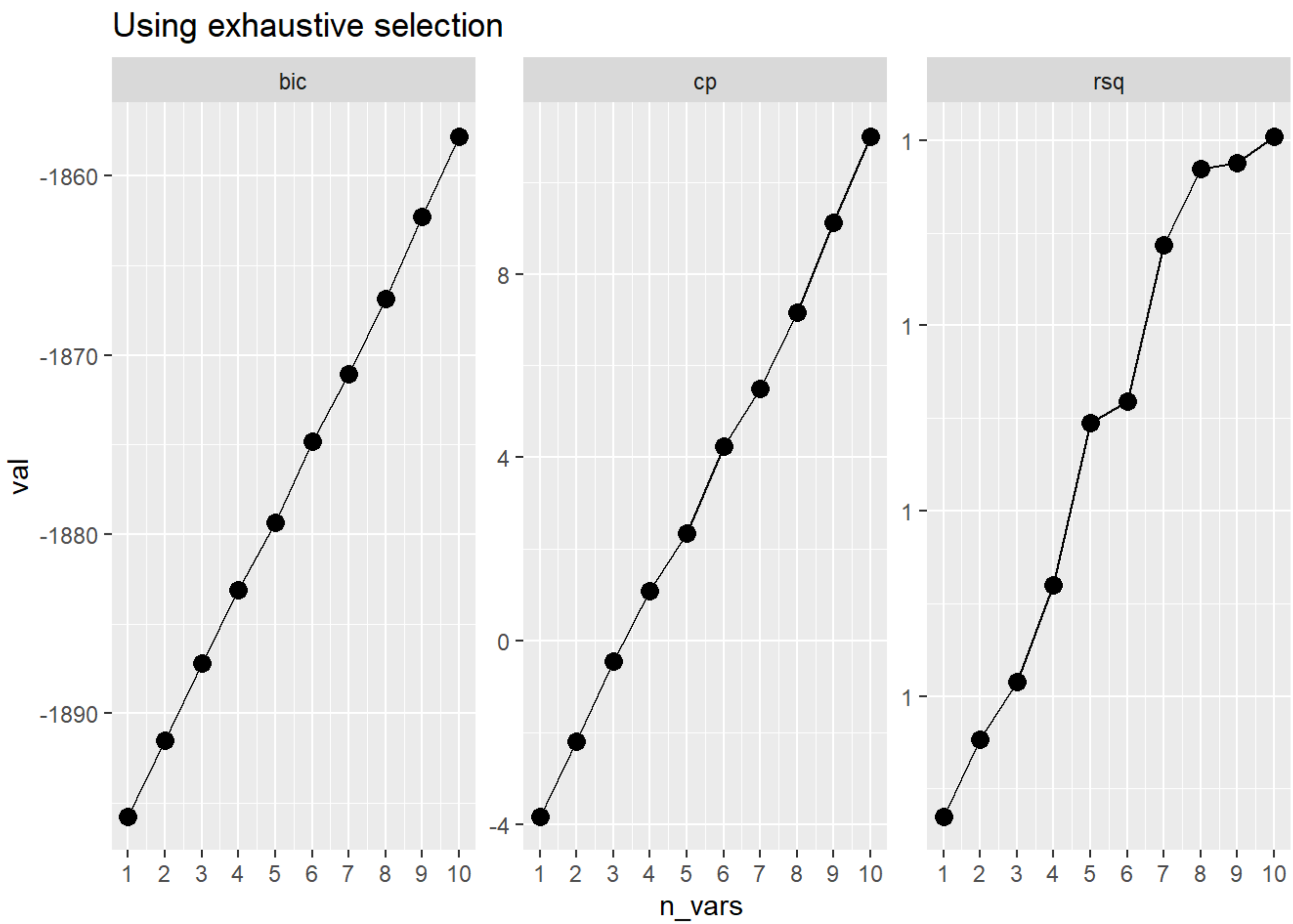
```
beta_7 <- 2

y_x7 <- beta_0 + (beta_7 * df_x10$x7) + e

df_x10_y7 <- df_x10 %>%
  cbind(y_x7) %>%
  dplyr::select(-y)
```

```
exhaustive_x7_mod <- regsubsets(y_x7 ~ .,
                     data = df_x10_y7,
                     nvmax = 10,
                     method = "exhaustive") %>%
  summary()

exhaustive_x7_mod
```

```
## Subset selection object
## Call: regsubsets.formula(y_x7 ~ ., data = df_x10_y7, nvmax = 10, method = "exhaustive")
## 10 Variables  (and intercept)
##     Forced in Forced out
## x1      FALSE      FALSE
```

```
## x2       FALSE      FALSE
## x3       FALSE      FALSE
## x4       FALSE      FALSE
## x5       FALSE      FALSE
## x6       FALSE      FALSE
## x7       FALSE      FALSE
## x8       FALSE      FALSE
## x9       FALSE      FALSE
## x10      FALSE      FALSE
## 1 subsets of each size up to 10
## Selection Algorithm: exhaustive
##           x1  x2  x3  x4  x5  x6  x7  x8  x9  x10
## 1  ( 1 )  " " " " " " " " " " " " "*" " " " " " "
## 2  ( 1 )  " " " " " " " " " " " " "*" " " " " "*"
## 3  ( 1 )  " " " " " " " " "*" " " "*" " " " " "*"
## 4  ( 1 )  " " "*" " " "*" " " " " "*" " " " " "*"
## 5  ( 1 )  " " "*" " " " " "*" " " "*" "*" " " "*"
## 6  ( 1 )  "*" " " "*" " " "*" " " "*" "*" "*" " "
## 7  ( 1 )  "*" "*" "*" "*" "*" " " "*" " " "*" " "
## 8  ( 1 )  "*" "*" "*" " " "*" " " "*" "*" "*" "*"
## 9  ( 1 )  "*" "*" "*" " " "*" "*" "*" "*" "*" "*"
## 10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

```
plot_subset_metrics(exhaustive_x7_mod, max_subset = 10, method = "exhaustive")
```



Best subset selection clearly indicates 1-variable model is best, with `x7` being the variable included. This makes sense!

```
# fit final model w/ lambda ~ 0 and get coefficients
final_mod_y7 <- linear_reg(penalty = best_lambda, mixture = 1) %>%
  set_engine("glmnet") %>%
  fit(y_x7 ~ ., data = df_x10_y7)
```

```
final_mod_y7 %>%
  tidy()
```

```
## # A tibble: 11 x 3
##    term        estimate      penalty
##    <chr>          <dbl>        <dbl>
##  1 (Intercept)     4.81 0.0000000001
##  2 x1              0    0.0000000001
##  3 x2              0    0.0000000001
##  4 x3              0    0.0000000001
##  5 x4              0    0.0000000001
##  6 x5              0    0.0000000001
##  7 x6              0    0.0000000001
##  8 x7              1.94 0.0000000001
##  9 x8              0    0.0000000001
## 10 x9              0    0.0000000001
## 11 x10             0    0.0000000001
```

Only significant variable is `x7`, as expected.

(9)

(a)

```
college <- College

college_split <- initial_split(college)
college_train <- training(college_split)
college_test <- testing(college_split)
```

```
college_rec <- recipe(Apps ~ ., data = college_train) %>%
  step_normalize(all_numeric())
```

(b)

```
lm_fit <- lm(Apps ~ ., data = college_train)
```

```
lm_preds <- predict(lm_fit, college_test)

mean((lm_preds - college_test$Apps)^2, na.rm = T)
```

```
## [1] 1569543
```

Test RMSE = 1,010,467

(c)

```
# train_mat <- model.matrix(Apps ~ ., data = college_train %>% select_if(is.numeric))
# test_mat <- model.matrix(Apps ~ ., data = college_test)
#
# lambda_grid <- seq(0, 1, .01)
#
# ridge_fit <- cv.glmnet(train_mat,
#                        dplyr::select(college_train, Apps),
#                        alpha = 0,
#                        lambda = lambda_grid,
#                        thresh = .000001)
```

```
# lambda_grid <- 10^seq(10, -2, length = 100)
#
# ridge_fit <- glmnet(college_train %>% dplyr::select(-Apps) %>% as.matrix(),
#                     college_train$Apps %>% as.matrix(),
#                     lambda = lambda_grid)
```

```
ridge_fit <- linear_reg(penalty = .1, mixture = 0) %>%
  set_engine("glmnet") %>%
  fit(Apps ~ ., data = college_train)
```

```r
ridge_preds <- predict(ridge_fit, college_test)

mean((ridge_preds$.pred - college_test$Apps)^2, na.rm = T)
```

```
## [1] 3091428
```

MSE is 1,017,567

(d)

```r
# lasso_fit <- cv.glmnet(train_mat,
#                         college_train %>%
#                           rownames_to_column() %>%
#                           dplyr::select(Apps),
#                         alpha = 1,
#                         lambda = lambda_grid,
#                         threshold(.00001))
```

Running into unexpected error with `cv.glmnet()` in (c) and (d). Will not be cross-validating to choose the hyperparameters, will just fit at some defaults.

```r
lasso_fit <- linear_reg(penalty = .1, mixture = 1) %>%
  set_engine("glmnet") %>%
  fit(Apps ~ ., data = college_train)

lasso_preds <- predict(lasso_fit, college_test)

mean((lasso_preds$.pred - college_test$Apps)^2, na.rm = T)
```

```
## [1] 1595294
```

MSE is 1,000,474

(e)

```r
library(pls)
```

```
##
## Attaching package: 'pls'
```

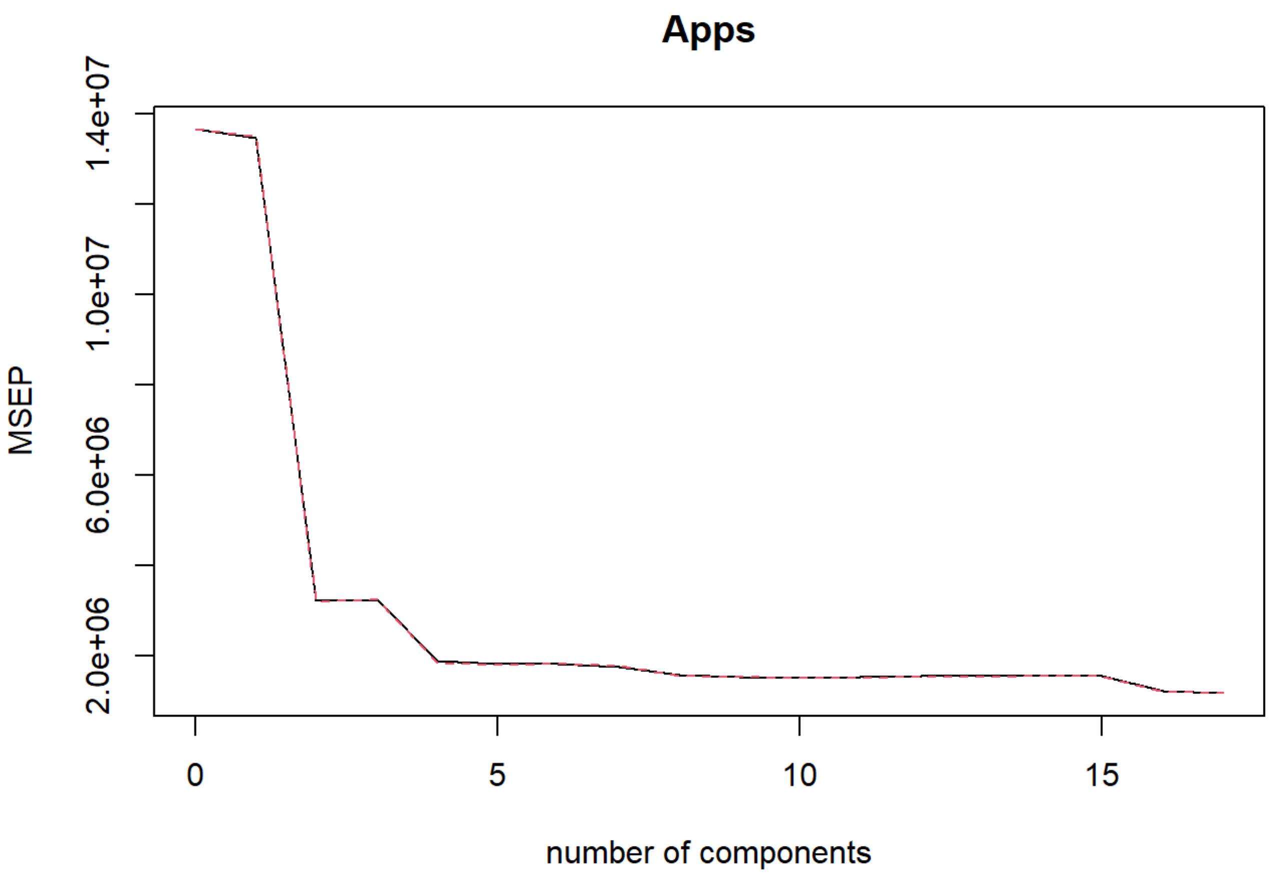
```
## The following object is masked from 'package:stats':
##
##     loadings
```

```r
pcr_fit <- pcr(Apps ~ ., data = college_train, scale = TRUE, validation = "CV")

summary(pcr_fit)
```

```
## Data:    X dimension: 583 17
##  Y dimension: 583 1
## Fit method: svdpc
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            3697     3671     1795     1802     1379     1350     1354
## adjCV         3697     3672     1793     1802     1361     1344     1350
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1330     1261     1240      1234      1237      1247      1248
## adjCV     1336     1250     1236      1230      1234      1243      1243
##        14 comps  15 comps  16 comps  17 comps
## CV         1248      1250      1103      1090
## adjCV      1244      1246      1098      1085
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       31.83    57.55    64.75    70.55    75.86    80.73    84.38    87.84
## Apps     1.61    76.78    76.94    87.01    87.30    87.48    87.84    89.35
```

```
##        9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X        90.91     93.24     95.34     97.19     98.22     98.95     99.46
## Apps     89.72     89.84     89.85     89.86     89.93     89.94     90.00
##        16 comps  17 comps
## X         99.83     100.0
## Apps      92.33      92.8
```

```
validationplot(pcr_fit, val.type = "MSEP")
```



```
pcr_preds <- predict(pcr_fit, college_test)

mean((pcr_preds - college_test$Apps)^2)
```

```
## [1] 5168693
```

Can see a high M (close to normal OLS) looks best.

The MSE is 2,400,575

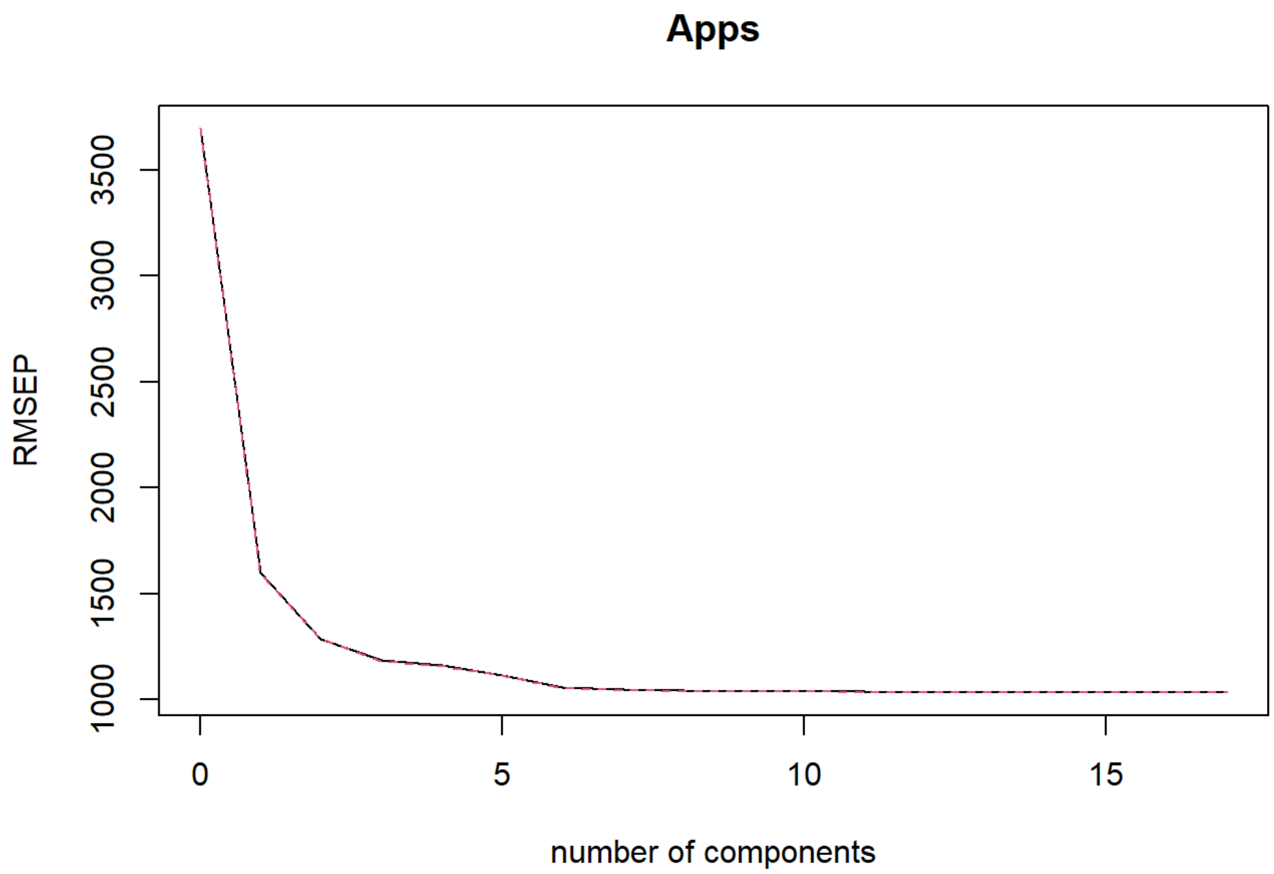
(f)

```
pls_fit <- plsr(Apps ~ ., data = college_train, scale = TRUE, validation = "CV")

summary(pls_fit)
```

```
## Data:    X dimension: 583 17
##  Y dimension: 583 1
## Fit method: kernelpls
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            3697     1597     1286     1185     1162     1114     1058
## adjCV         3697     1595     1287     1183     1158     1114     1055
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1047     1044     1040      1039      1039      1036      1036
## adjCV     1044     1041     1038      1037      1037      1034      1034
##        14 comps  15 comps  16 comps  17 comps
## CV         1036      1037      1037      1037
## adjCV      1034      1034      1034      1034
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
```

```
## X        25.76    38.62    63.03    66.63    70.17    74.30    77.69    81.12
## Apps     81.66    88.45    90.12    90.97    91.97    92.58    92.68    92.72
##        9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X        83.32     86.46     89.17     91.32     93.82     95.83     97.31
## Apps     92.75     92.77     92.79     92.80     92.80     92.80     92.80
##        16 comps  17 comps
## X         98.64     100.0
## Apps      92.80      92.8
```

```
validationplot(pls_fit)
```




```
pls_preds <- predict(pls_fit, college_test)

mean((pls_preds - college_test$Apps)^2)
```

```
## [1] 2275662
```

The MSE is 1,175,241

(g)

The best model according to MSE is the lasso in part (d).

Chapter 7

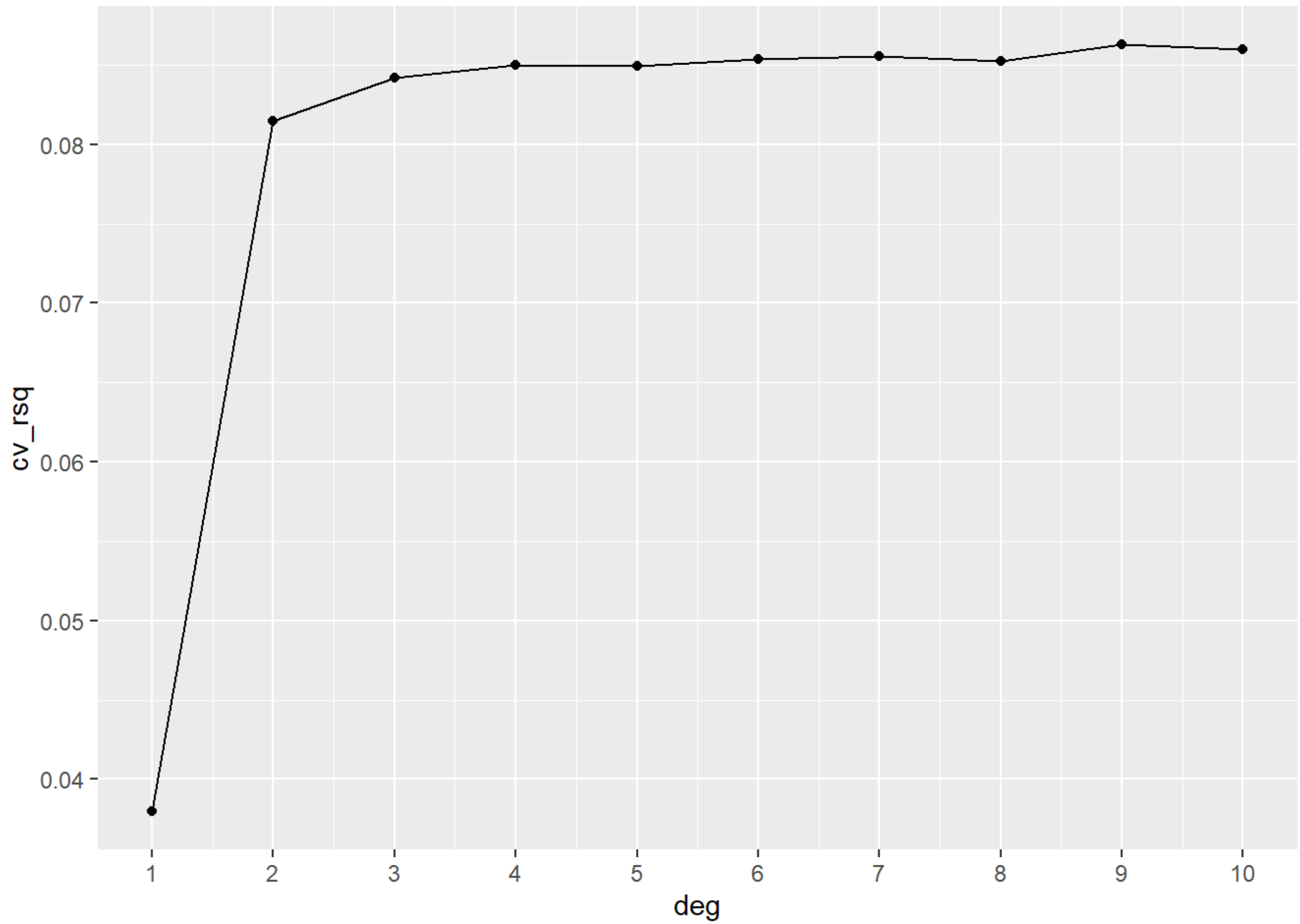
(6)

(a)

```
wage <- ISLR::Wage

# create CV folds
wage_folds <- vfold_cv(wage)
```

```
# function that returns 10-fold CV'ed RMSE of wage ~ age model at a specific degree d
cv_rsq_at_d <- function(deg) {
  cv_rsq <- wage_folds$splits %>%
  # over each fold, calculate RMSE of wage ~ age model for that fold, using a specific poly(age, i) value
  map_dbl(function(x) {
    df <- x
    fit <- lm(wage ~ poly(age, deg), data = df)
    sum <- summary(fit)
    sum$adj.r.squared
  })

  mean(cv_rsq)
}

df_deg <- tibble(deg = 1:10,
                 cv_rsq = c(cv_rsq_at_d(deg = 1),
                            cv_rsq_at_d(deg = 2),
```

```
                          cv_rsq_at_d(deg = 3),
                          cv_rsq_at_d(deg = 4),
                          cv_rsq_at_d(deg = 5),
                          cv_rsq_at_d(deg = 6),
                          cv_rsq_at_d(deg = 7),
                          cv_rsq_at_d(deg = 8),
                          cv_rsq_at_d(deg = 9),
                          cv_rsq_at_d(deg = 10)))

df_deg %>%
  ggplot(aes(deg, cv_rsq)) +
  geom_path() +
  geom_point() +
  scale_x_continuous(breaks = 1:10)
```



Should choose d = 9 (at least out of the first 10 degrees) since it has the highest R_a^2 from this test.

```
fit_d1 <- lm(wage ~ poly(age, 1), data = wage)
fit_d2 <- lm(wage ~ poly(age, 2), data = wage)
fit_d3 <- lm(wage ~ poly(age, 3), data = wage)
fit_d9 <- lm(wage ~ poly(age, 9), data = wage)

anova(fit_d1, fit_d2)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, 1)
## Model 2: wage ~ poly(age, 2)
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1   2998 5022216
## 2   2997 4793430  1    228786 143.04 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(fit_d1, fit_d3)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, 1)
## Model 2: wage ~ poly(age, 3)
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1   2998 5022216
## 2   2996 4777674  2    244542 76.674 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

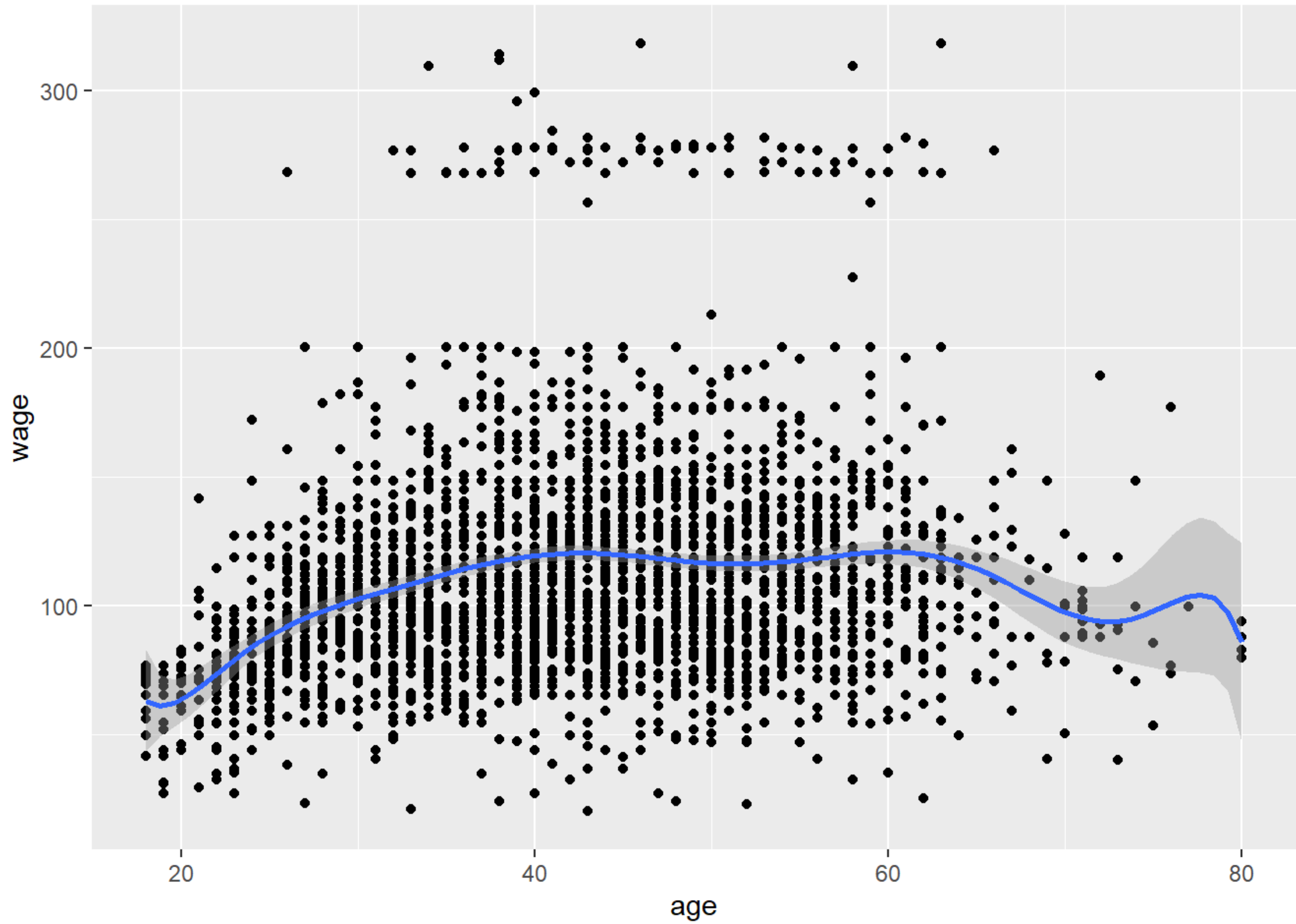
```
anova(fit_d3, fit_d9)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, 3)
## Model 2: wage ~ poly(age, 9)
##   Res.Df     RSS Df Sum of Sq     F  Pr(>F)
## 1   2996 4777674
## 2   2990 4756703  6     20971 2.197 0.04053 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This seems to match what a few ANOVA's look like, at least in terms of requiring more than d = 1 and d = 9 being significant compared to smaller degrees like 3.

Final fit plot:

```
wage %>%
  ggplot(aes(age, wage)) +
  geom_point() +
  stat_smooth(method = "lm", formula = y ~ poly(x, 9))
```



(b)

```r
cv_rsq_at_ncuts <- function(ncuts) {
  cv_rsq <- wage_folds$splits %>%
  # over each fold, calculate RMSE of wage ~ age model for that fold, using a specific poly(age, i) value
  map_dbl(function(x) {
    df <- x
    fit <- lm(wage ~ cut(age, ncuts), data = df)
    sum <- summary(fit)
    sum$adj.r.squared
  })

  mean(cv_rsq)
}

cv_rsq_at_ncuts(ncuts = 2)
```

```
## [1] 0.004858072
```

```r
cv_rsq_at_ncuts(ncuts = 3)
```

```
## [1] 0.03464812
```

```r
cv_rsq_at_ncuts(ncuts = 4)
```

```
## [1] 0.06158809
```

```r
cv_rsq_at_ncuts(ncuts = 5)
```

```
## [1] 0.06428999
```

```r
cv_rsq_at_ncuts(ncuts = 6)
```

```
## [1] 0.06932384
```

```r
cv_rsq_at_ncuts(ncuts = 7)
```

```
## [1] 0.07639686
```

```r
cv_rsq_at_ncuts(ncuts = 8)
```

```
## [1] 0.08254497
```

```r
cv_rsq_at_ncuts(ncuts = 9)
```

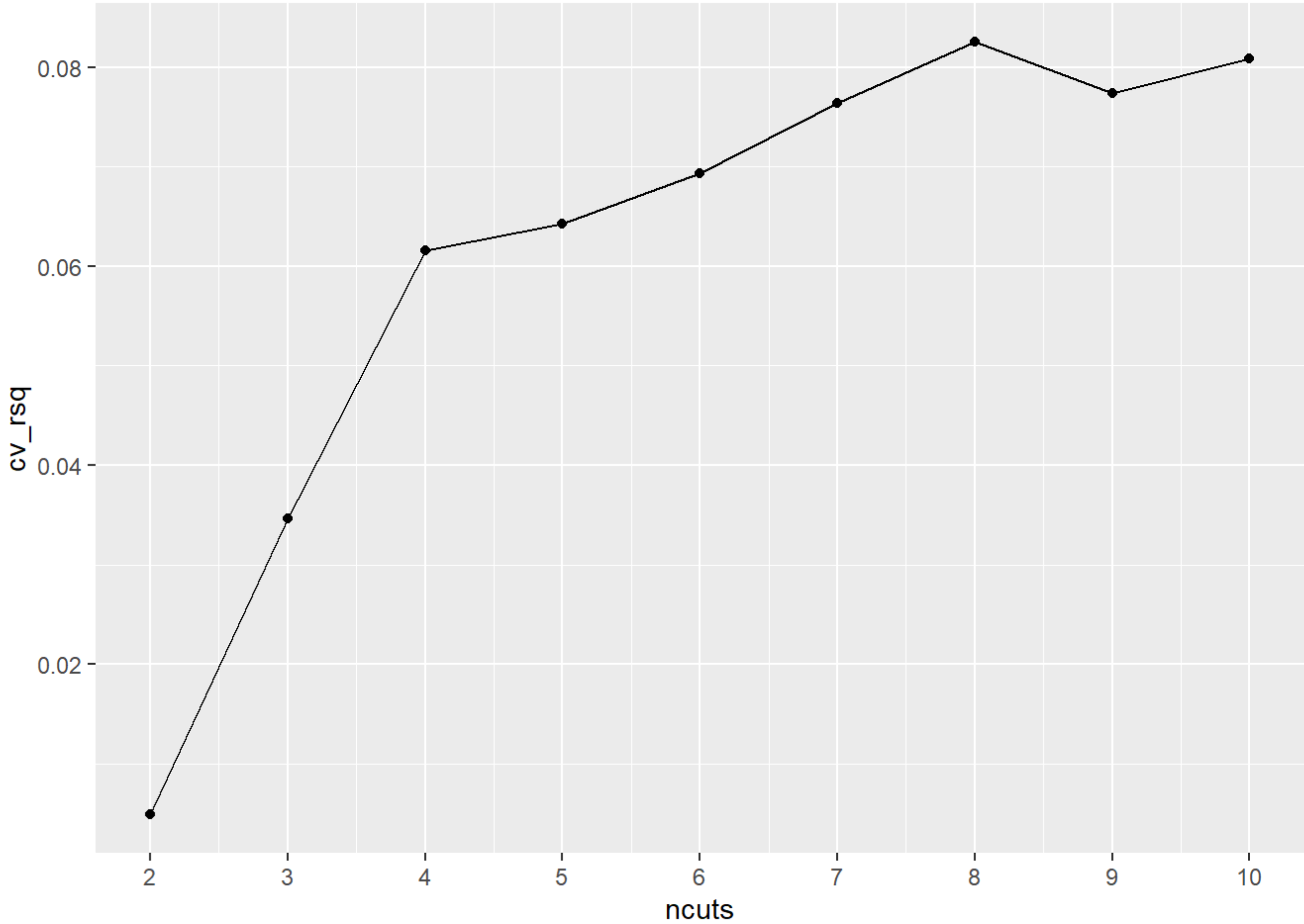
```
## [1] 0.0773816
```

```r
cv_rsq_at_ncuts(ncuts = 10)
```

```
## [1] 0.08083422
```

```r
df_ncuts <- tibble(ncuts = 2:10,
                   cv_rsq = c(cv_rsq_at_ncuts(ncuts = 2),
                              cv_rsq_at_ncuts(ncuts = 3),
                              cv_rsq_at_ncuts(ncuts = 4),
                              cv_rsq_at_ncuts(ncuts = 5),
                              cv_rsq_at_ncuts(ncuts = 6),
                              cv_rsq_at_ncuts(ncuts = 7),
                              cv_rsq_at_ncuts(ncuts = 8),
                              cv_rsq_at_ncuts(ncuts = 9),
```

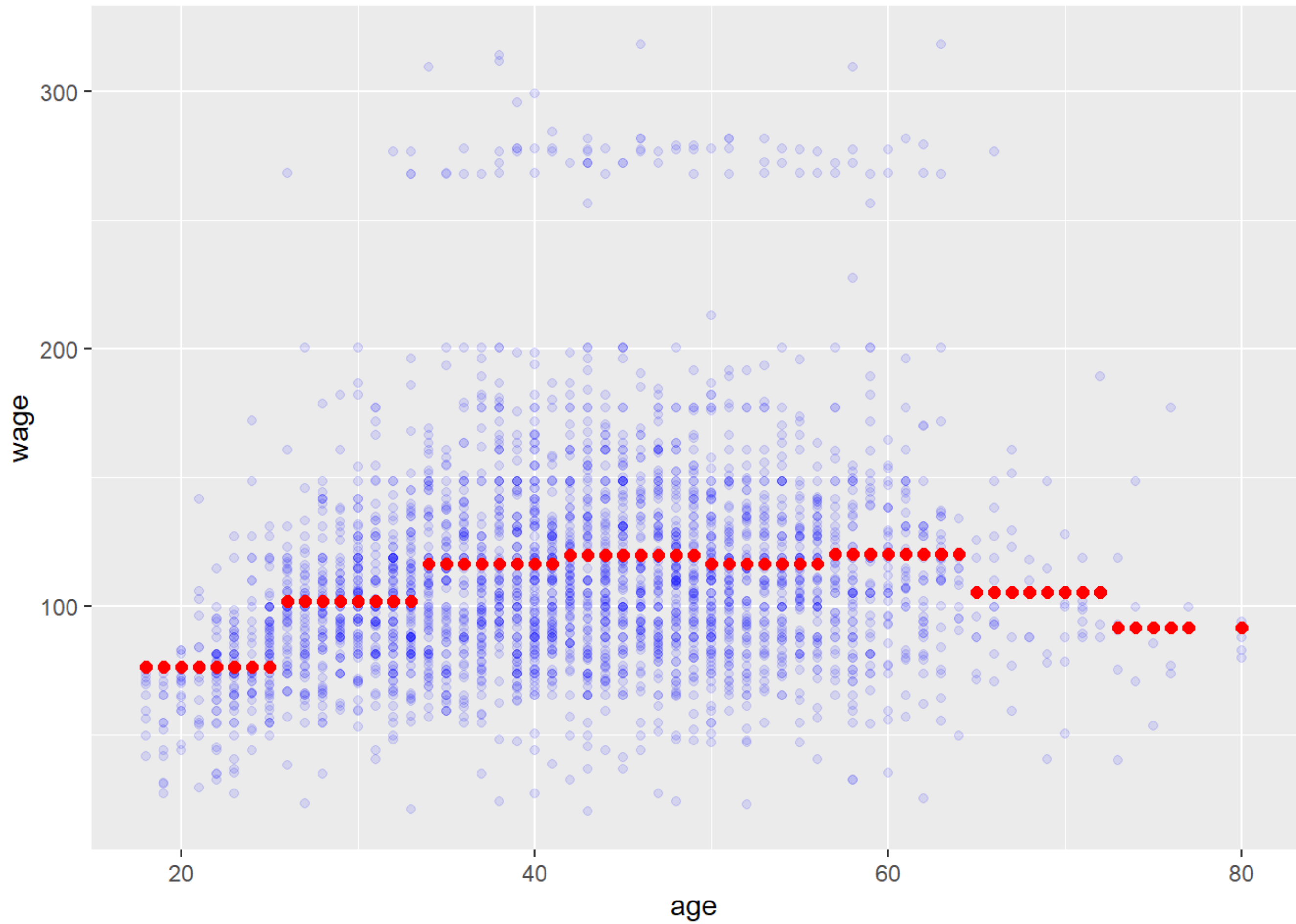
```
                             cv_rsq_at_ncuts(ncuts = 10)))

df_ncuts %>%
  ggplot(aes(ncuts, cv_rsq)) +
  geom_path() +
  geom_point() +
  scale_x_continuous(breaks = 1:10)
```



Looks like 8 cuts has the best R_a^2

Final fit plot:

```
fit <- lm(wage ~ cut(age, 8), data = wage)

preds <- predict(fit, wage)

wage %>%
  mutate(pred_wage = preds) %>%
  ggplot() +
  geom_point(aes(age, wage), color = "blue", alpha = .1) +
  geom_point(aes(age, pred_wage), color = "red", size = 2)
```



(9)

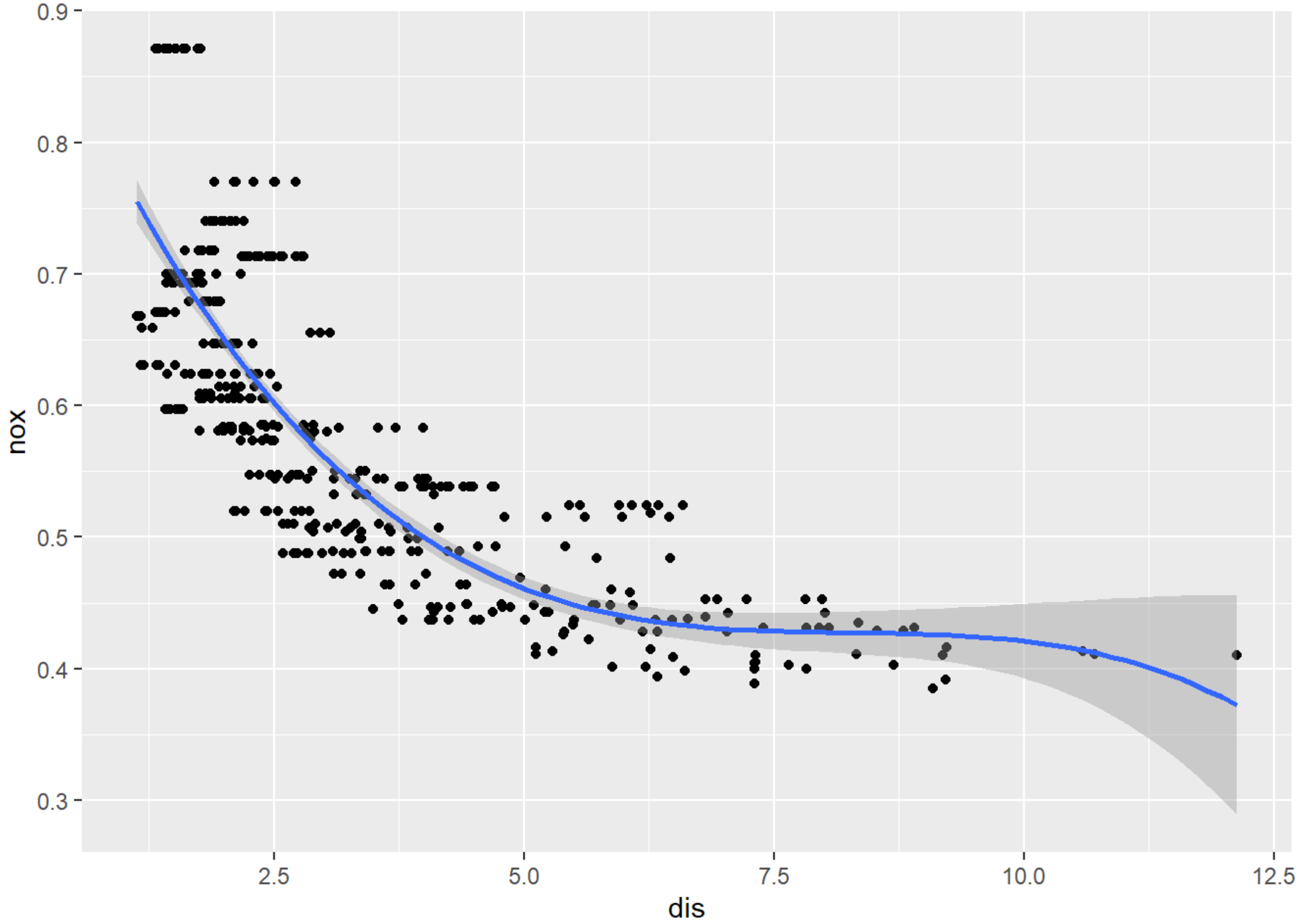
(a)

```
boston <- MASS::Boston

fit_cub <- lm(nox ~ poly(dis, 3), data = boston)
```

```r
summary(fit_cub)
```

```
##
## Call:
## lm(formula = nox ~ poly(dis, 3), data = boston)
##
## 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
```

```r
boston %>%
  ggplot(aes(dis, nox)) +
  geom_point() +
  stat_smooth(method = "lm", formula = y ~ poly(x, 3))
```

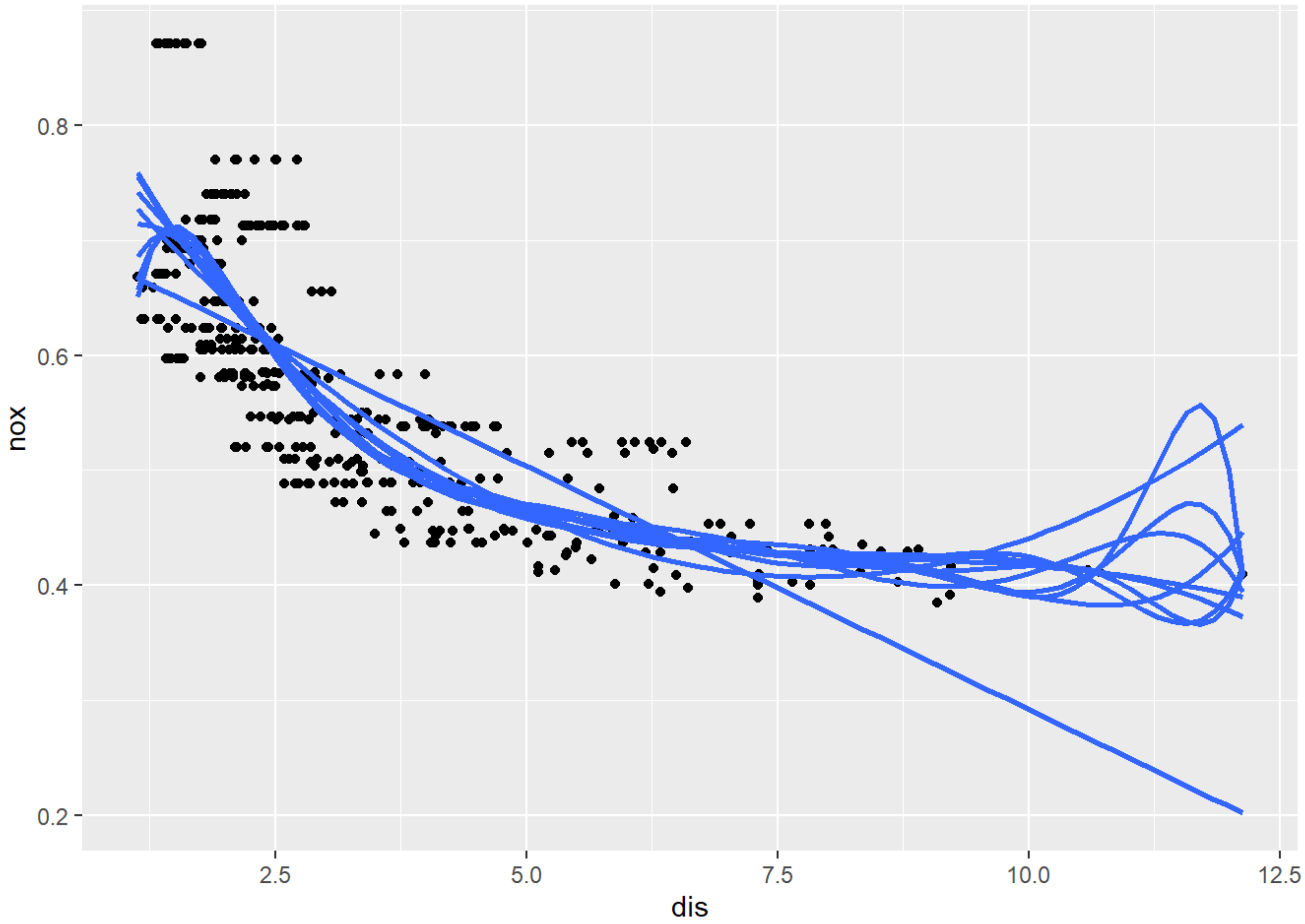


Output and plot above-- all degrees significant, fit looks good.

(b)

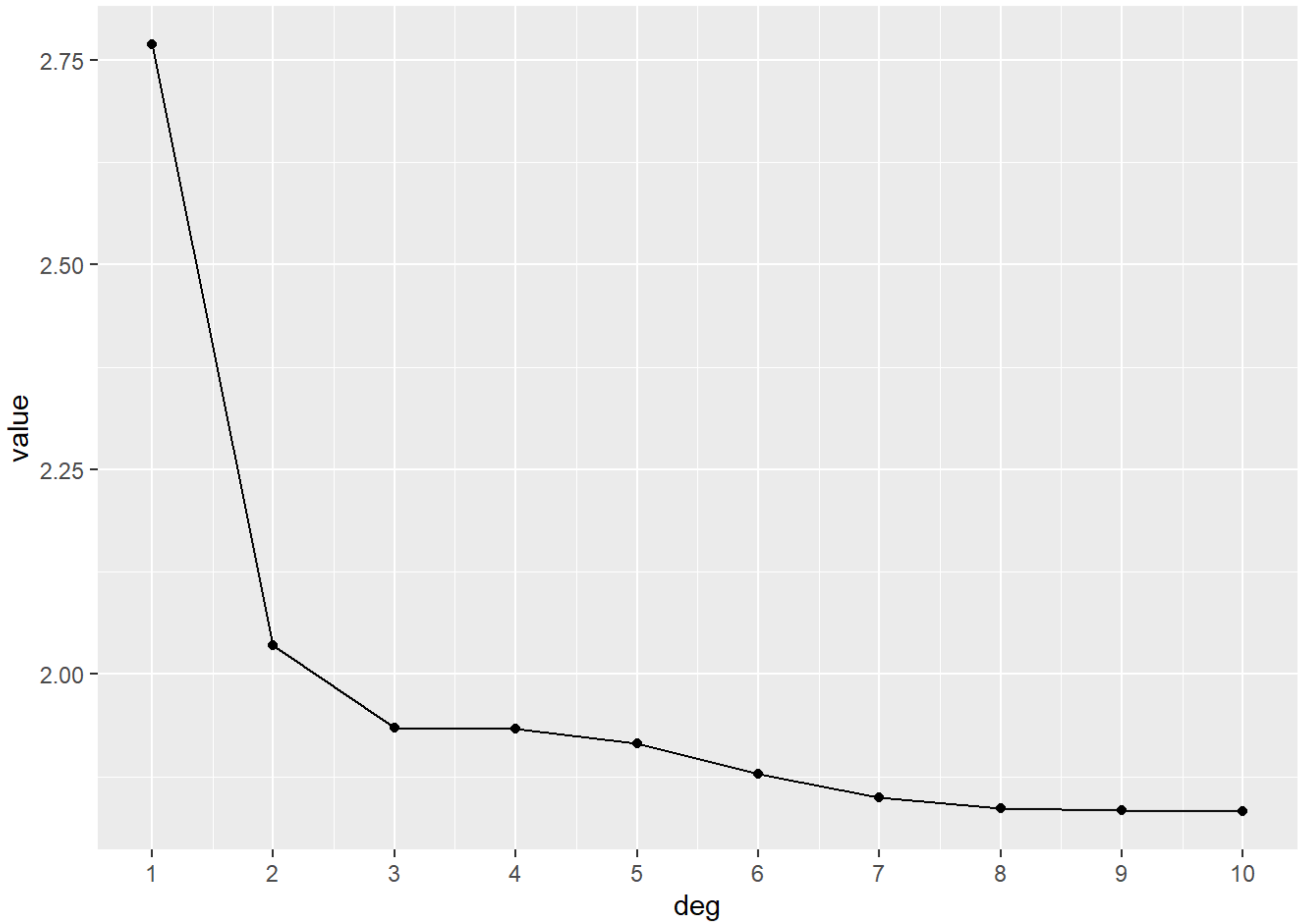
```
boston %>%
  ggplot(aes(dis, nox)) +
  geom_point() +
  stat_smooth(method = "lm", formula = y ~ poly(x, 1), se = F, alpha = .5) +
  stat_smooth(method = "lm", formula = y ~ poly(x, 2), se = F, alpha = .5) +
```

```
  stat_smooth(method = "lm", formula = y ~ poly(x, 3), se = F, alpha = .5) +
  stat_smooth(method = "lm", formula = y ~ poly(x, 4), se = F, alpha = .5) +
  stat_smooth(method = "lm", formula = y ~ poly(x, 5), se = F, alpha = .5) +
  stat_smooth(method = "lm", formula = y ~ poly(x, 6), se = F, alpha = .5) +
  stat_smooth(method = "lm", formula = y ~ poly(x, 7), se = F, alpha = .5) +
  stat_smooth(method = "lm", formula = y ~ poly(x, 8), se = F, alpha = .5) +
  stat_smooth(method = "lm", formula = y ~ poly(x, 9), se = F, alpha = .5) +
  stat_smooth(method = "lm", formula = y ~ poly(x, 10), se = F)
```



Many with very similar, odd high-degree fits.

```r
rss <- c()

for (i in 1:10) {
  fit <- lm(nox ~ poly(dis, i), boston)
  rss[i] <- sum(fit$residuals^2)
}

rss %>%
  as_tibble() %>%
  cbind(deg = 1:10) %>%
  ggplot(aes(deg, value)) +
  geom_path() +
  geom_point() +
  scale_x_continuous(breaks = 1:10)
```



(c)

I'll used bootstrap datasets and test a degree on each one, deciding like that

```
boston_folds <- vfold_cv(boston)

# boston_folds %>%
#   map(function(df) {
```

```
#     fit <- lm(nox ~ poly(dis, i), data = df)
#   })
```

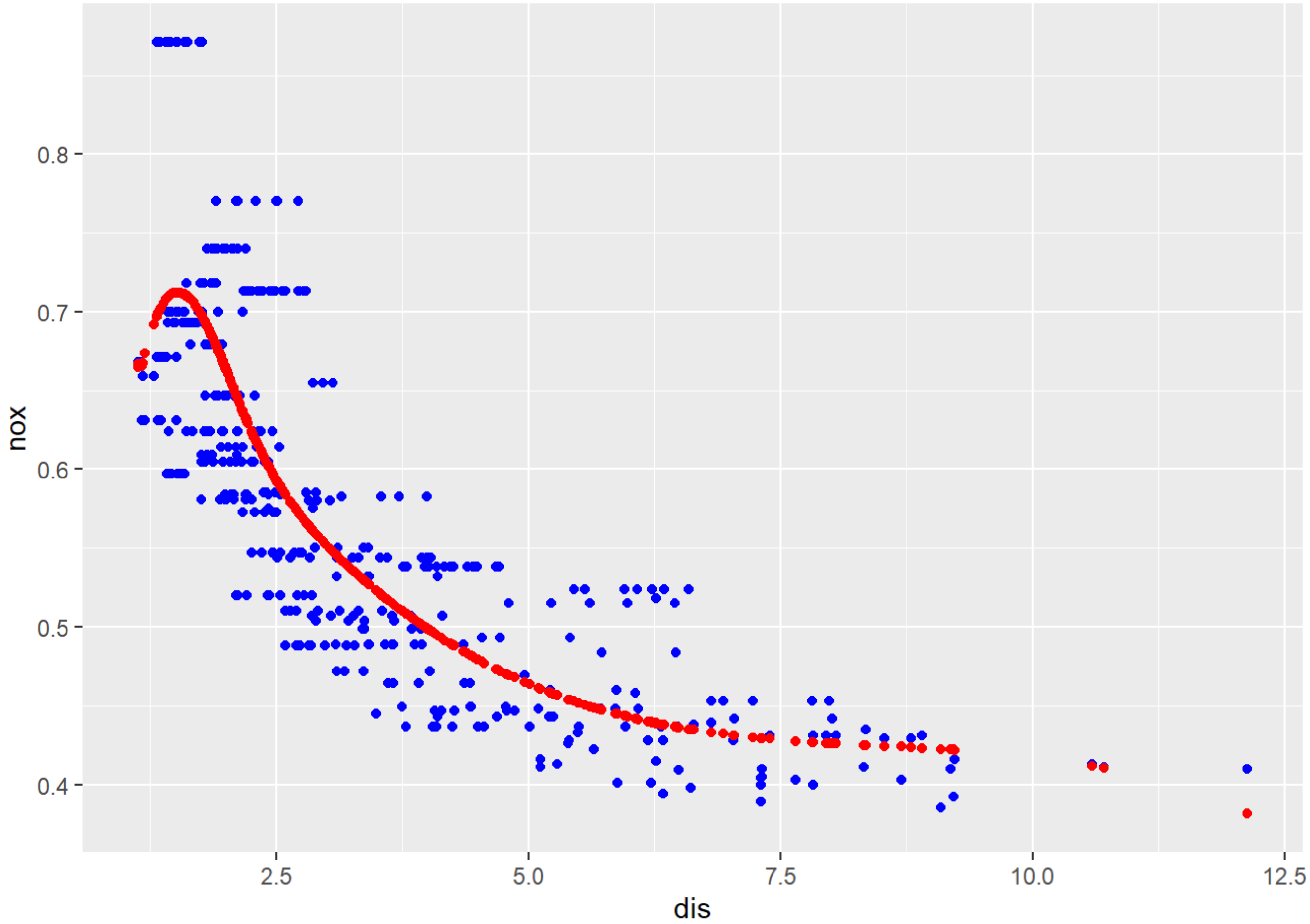
(d)

```
bs_fit <- lm(nox ~ bs(dis, knots = c(1.2, 2.2, 3.2)), data = boston)

summary(bs_fit)
```

```
##
## Call:
## lm(formula = nox ~ bs(dis, knots = c(1.2, 2.2, 3.2)), data = boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.12689 -0.03784 -0.01103  0.02309  0.19751
##
## Coefficients:
##                                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)                        0.666905   0.049590  13.448  < 2e-16 ***
## bs(dis, knots = c(1.2, 2.2, 3.2))1 -0.006928   0.057745  -0.120   0.9046
## bs(dis, knots = c(1.2, 2.2, 3.2))2  0.098264   0.051036   1.925   0.0547 .
## bs(dis, knots = c(1.2, 2.2, 3.2))3 -0.061642   0.050796  -1.214   0.2255
## bs(dis, knots = c(1.2, 2.2, 3.2))4 -0.313991   0.054398  -5.772 1.38e-08 ***
## bs(dis, knots = c(1.2, 2.2, 3.2))5 -0.198266   0.063512  -3.122   0.0019 **
## bs(dis, knots = c(1.2, 2.2, 3.2))6 -0.285601   0.066682  -4.283 2.21e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06084 on 499 degrees of freedom
## Multiple R-squared:  0.7276, Adjusted R-squared:  0.7243
## F-statistic: 222.1 on 6 and 499 DF,  p-value: < 2.2e-16
```

I chose the knots based an approximate 25th, 50th, and 75th percentiles of `dis`. Output shows mostly significant

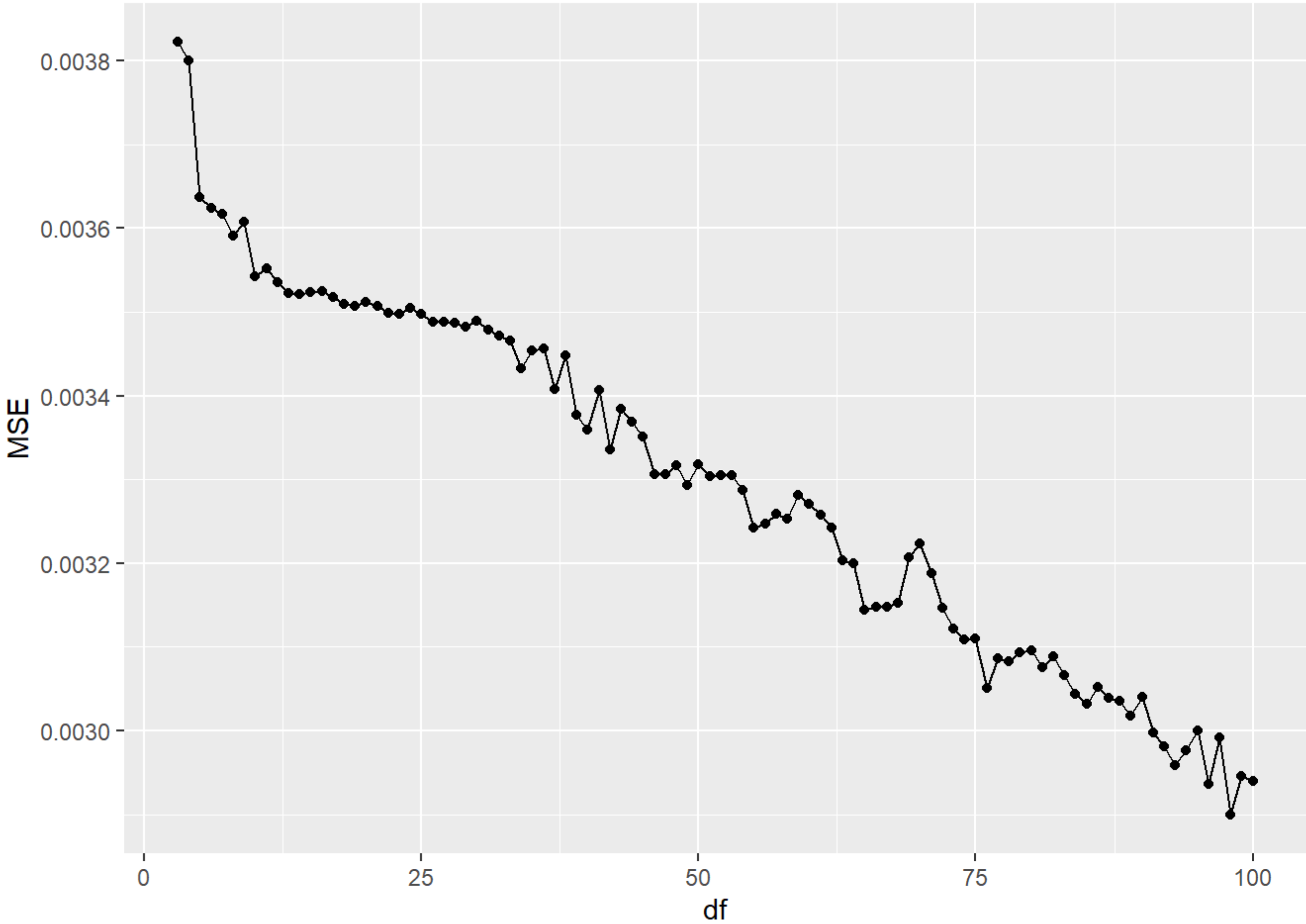
```
preds <- predict(bs_fit, boston)

boston %>%
  cbind(pred_nox = preds) %>%
  ggplot() +
  geom_point(aes(dis, nox), color = "blue") +
  geom_point(aes(dis, pred_nox), color = "red", size = 1.5)
```



Fit looks very good!

(e)

```
df_err <- c()

for (i in 3:100) {
  bs_fit <- lm(nox ~ bs(dis, df = i), data = boston)

  preds <- predict(bs_fit, boston)

  df_err[i] <- mean((preds - boston$nox)^2)
}

df_err %>%
  as_tibble() %>%
  drop_na() %>%
  mutate(df = row_number() + 2) %>%
  ggplot(aes(df, value)) +
  geom_path() +
  geom_point(size = 1.5) +
  labs(y = "MSE")
```



Consistently lower MSE as degrees of freedom increases.

(f)

To choose the best df value, I will split up the `Boston` dataset into a training and test set, fit a model at each to the training set, and test on the testing set.

```
boston_split <- initial_split(boston)
boston_train <- training(boston_split)
```

```
boston_test <- testing(boston_split)

train_err <- c()
test_err <- c()

for (i in 3:100) {
  bs_fit <- lm(nox ~ bs(dis, df = i), data = boston_train)

  preds_train <- predict(bs_fit, boston_train)
  preds_test <- predict(bs_fit, boston_test)

  train_err[i] <- mean((preds_train - boston$nox)^2)
  test_err[i] <- mean((preds_test - boston$nox)^2)
}

train_err
```

```
##   [1]         NA         NA 0.02673048 0.02674319 0.02698140 0.02700020
##   [7] 0.02700532 0.02701539 0.02701333 0.02698099 0.02700237 0.02700528
##  [13] 0.02702920 0.02701629 0.02701474 0.02700907 0.02704557 0.02704747
##  [19] 0.02707516 0.02712012 0.02711976 0.02712396 0.02711964 0.02711666
##  [25] 0.02714984 0.02712042 0.02713934 0.02709688 0.02714625 0.02716642
##  [31] 0.02713955 0.02721150 0.02716206 0.02720568 0.02728346 0.02720061
##  [37] 0.02735224 0.02729540 0.02722252 0.02736771 0.02729942 0.02732434
##  [43] 0.02738182 0.02733057 0.02731069 0.02740184 0.02732204 0.02738418
##  [49] 0.02734771 0.02740090 0.02738727 0.02743525 0.02739627 0.02744459
##  [55] 0.02741692 0.02735885 0.02743833 0.02741494 0.02745146 0.02765554
##  [61] 0.02769282 0.02773922 0.02764566 0.02754380 0.02754252 0.02761453
##  [67] 0.02767418 0.02761749 0.02762807 0.02775467 0.02787488 0.02780321
##  [73] 0.02774660 0.02778750 0.02788759 0.02780248 0.02779225 0.02787408
##  [79] 0.02791440 0.02785383 0.02780754 0.02793008 0.02789761 0.02797780
##  [85] 0.02790075 0.02810913 0.02790284 0.02796516 0.02801737 0.02797756
##  [91] 0.02804054 0.02791585 0.02804689 0.02804975 0.02805679 0.02810319
##  [97] 0.02808097 0.02819891 0.02813027 0.02804503
```

```
test_err
```

```
##   [1]         NA         NA 0.02388870 0.02384091 0.02369150 0.02368087
##   [7] 0.02368156 0.02364542 0.02365322 0.02375756 0.02373288 0.02369146
##  [13] 0.02373942 0.02388518 0.02394620 0.02394544 0.02392056 0.02385942
##  [19] 0.02394629 0.02404068 0.02408200 0.02410226 0.02406578 0.02403610
##  [25] 0.02411958 0.02406950 0.02411092 0.02405107 0.02411218 0.02422758
##  [31] 0.02414097 0.02416923 0.02412054 0.02430509 0.02433055 0.02419860
##  [37] 0.02435822 0.02417112 0.02427864 0.02422960 0.02410771 0.02429497
##  [43] 0.02433210 0.02415979 0.02428059 0.02409811 0.02419687 0.02424022
##  [49] 0.02422597 0.02418520 0.02417214 0.02427551 0.02420412 0.02420131
##  [55] 0.02431883 0.02430948 0.02419067 0.02408221 0.02398191 0.02395865
##  [61] 0.02424113 0.02416681 0.02423405 0.02441855 0.02444478 0.02451811
##  [67] 0.02445144 0.02387488 0.02374839 0.02393232 0.02404287 0.02424714
##  [73] 0.02423101 0.02404580 0.02426564 0.02434843 0.02435961 0.02415198
##  [79] 0.02408608 0.02399700 0.02413893 0.02421070 0.02394535 0.02395394
##  [85] 0.02388030 0.02427400 0.02438021 0.02420350 0.02402296 0.02402339
##  [91] 0.02418820 0.02416326 0.02430492 0.02412363 0.02386833 0.02417224
##  [97] 0.02414540 0.02417245 0.02407301 0.02392687
```

Using this method, the degrees of freedom doesn't seem to affect MSE so much.

(10)

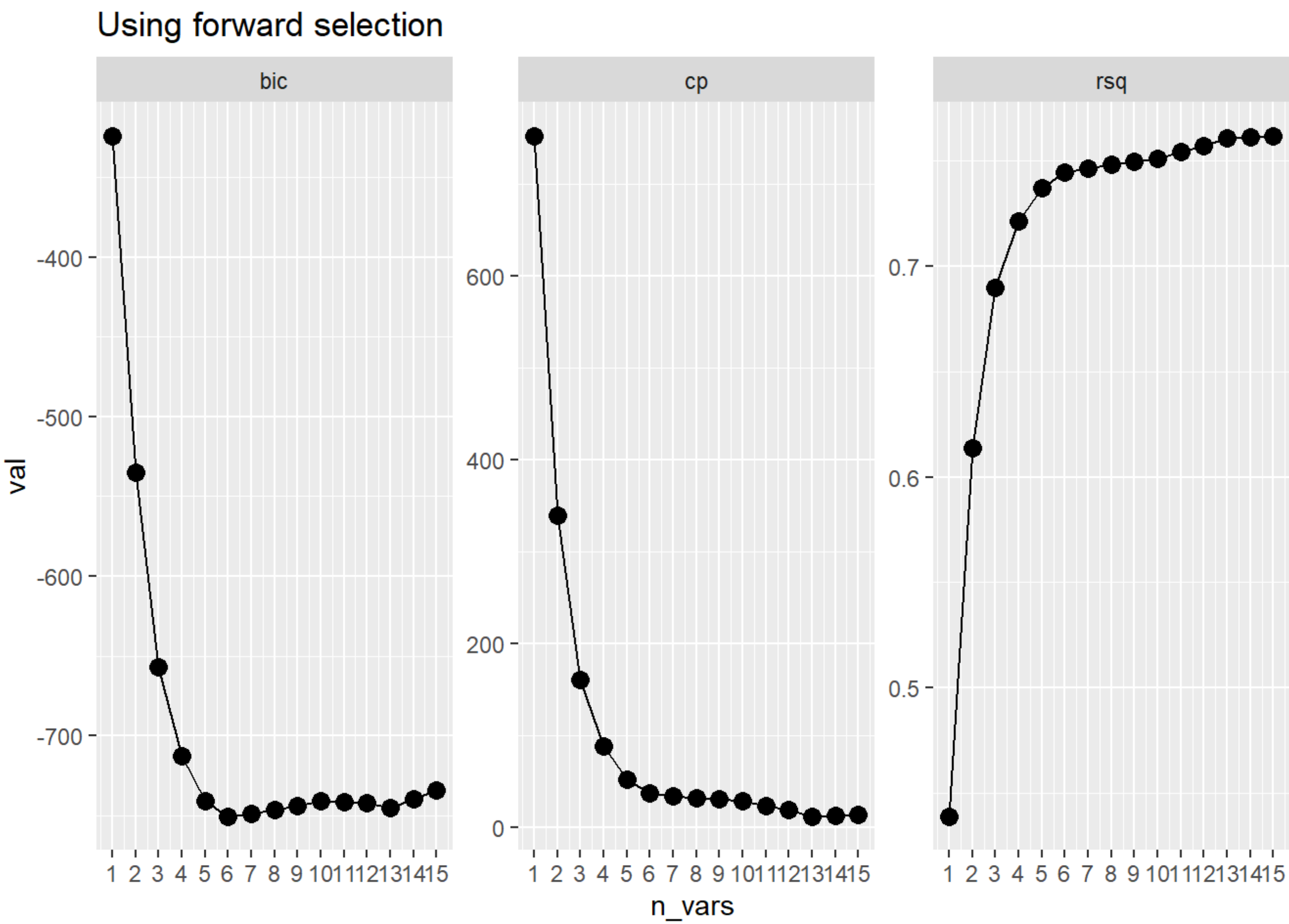
(a)

Using the same `college_split`, `college_train`, and `college_test` objects created earlier

```
college <- College

college_split <- initial_split(college)
college_train <- training(college_split)
college_test <- testing(college_split)
```

```
college_forward <- regsubsets(Outstate ~ .,
                              data = college_train,
```

```
                               nvmax = 15,
                               method = "forward") %>%
  summary()

# using function defined earlier to plot metrics over various subsets so we can decide a proper model size
plot_subset_metrics(model = college_forward,
                    max_subset = 15,
                    method = "forward")
```



The 12-variable model seems to be a good fit according to BIC and Mallow's Cp

```r
# selecting variables from the best 12-variable subset model
cols_to_keep <- college_forward$outmat %>%
  as_tibble() %>%
  dplyr::select(1:3, 5, 7, 9, 11, 13:17) %>%
  colnames()

cols_to_keep[1] <- "Private"

college_train_p12 <- college_train %>%
  dplyr::select(cols_to_keep, Outstate)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(cols_to_keep)` instead of `cols_to_keep` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```r
college_test_p12 <- college_test %>%
  dplyr::select(cols_to_keep, Outstate)
```

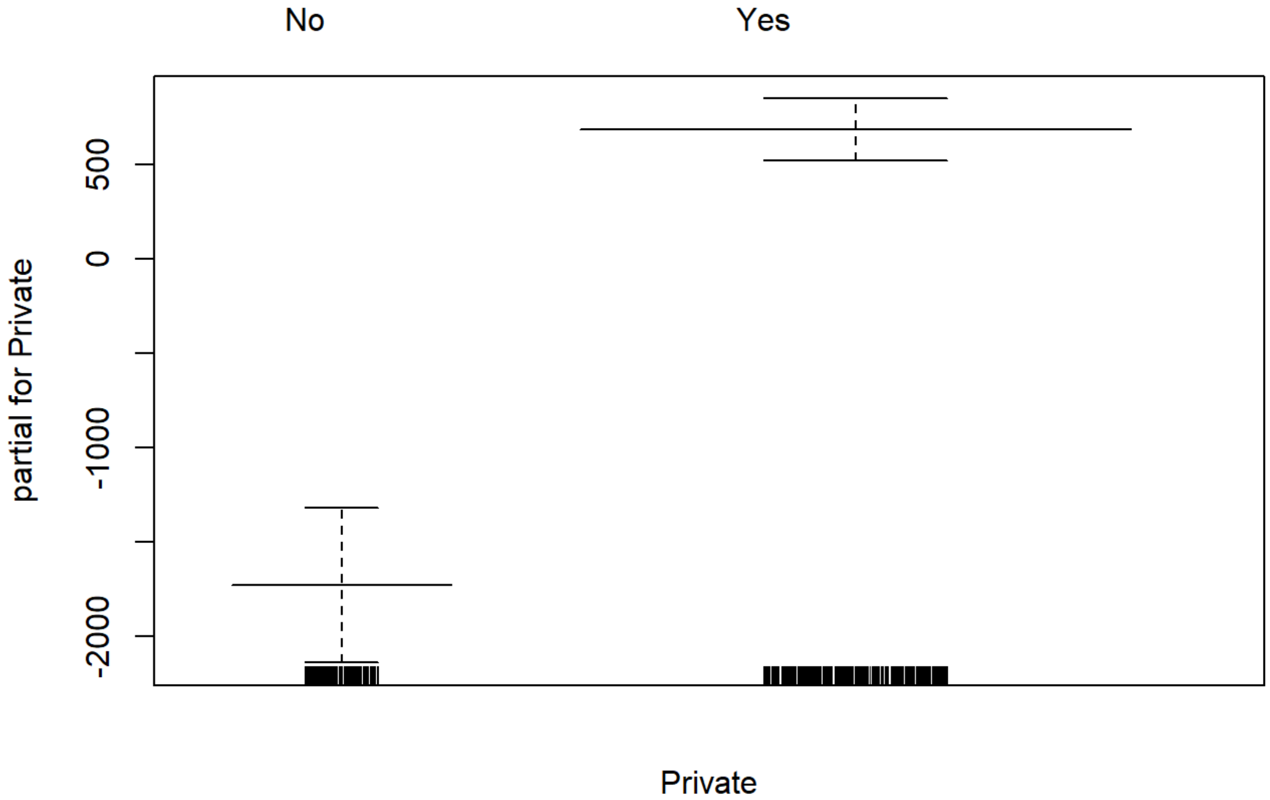
(b)

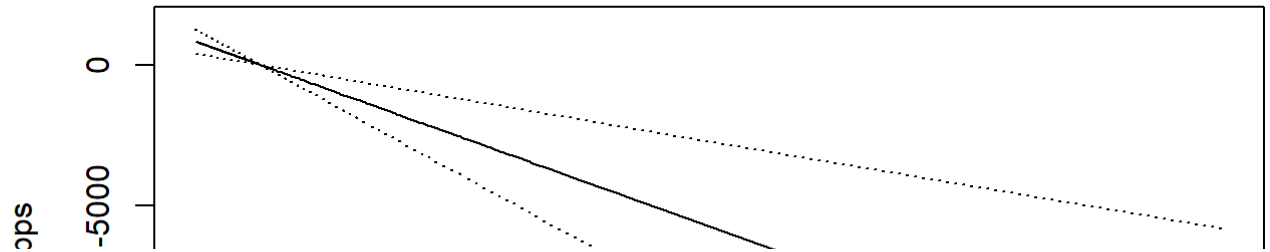
```r
gam_fit <- gam(Outstate ~ ., data = college_train_p12)

# see a few basic metrics
train_preds <- predict(gam_fit, college_train_p12)

college_train_p12 %>%
  cbind(pred_Outstate = train_preds) %>%
  metrics(truth = Outstate, estimate = pred_Outstate)
```

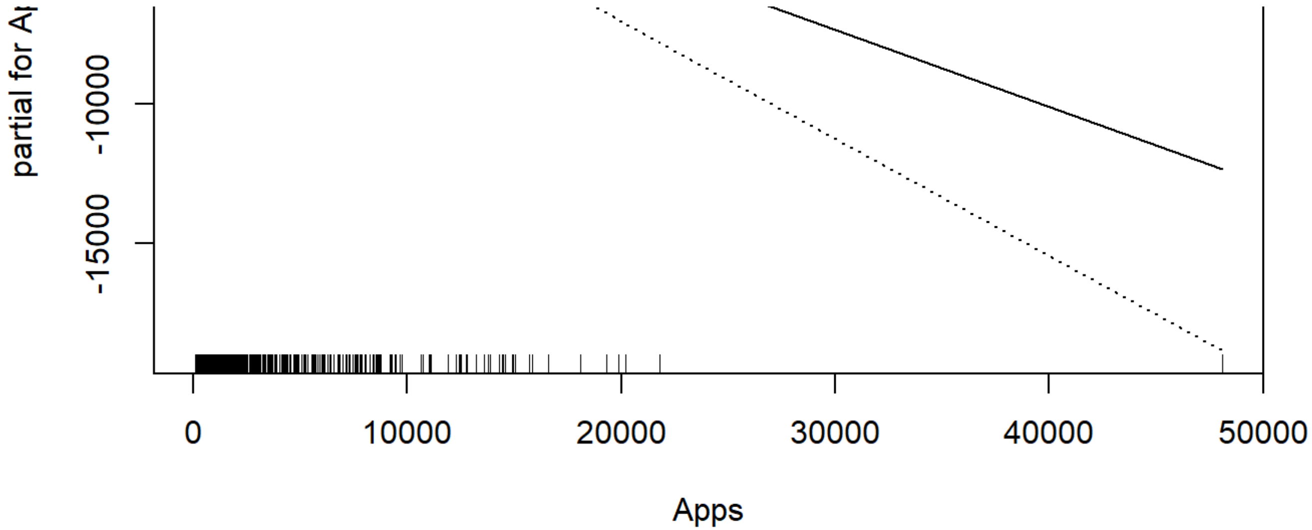
```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
```

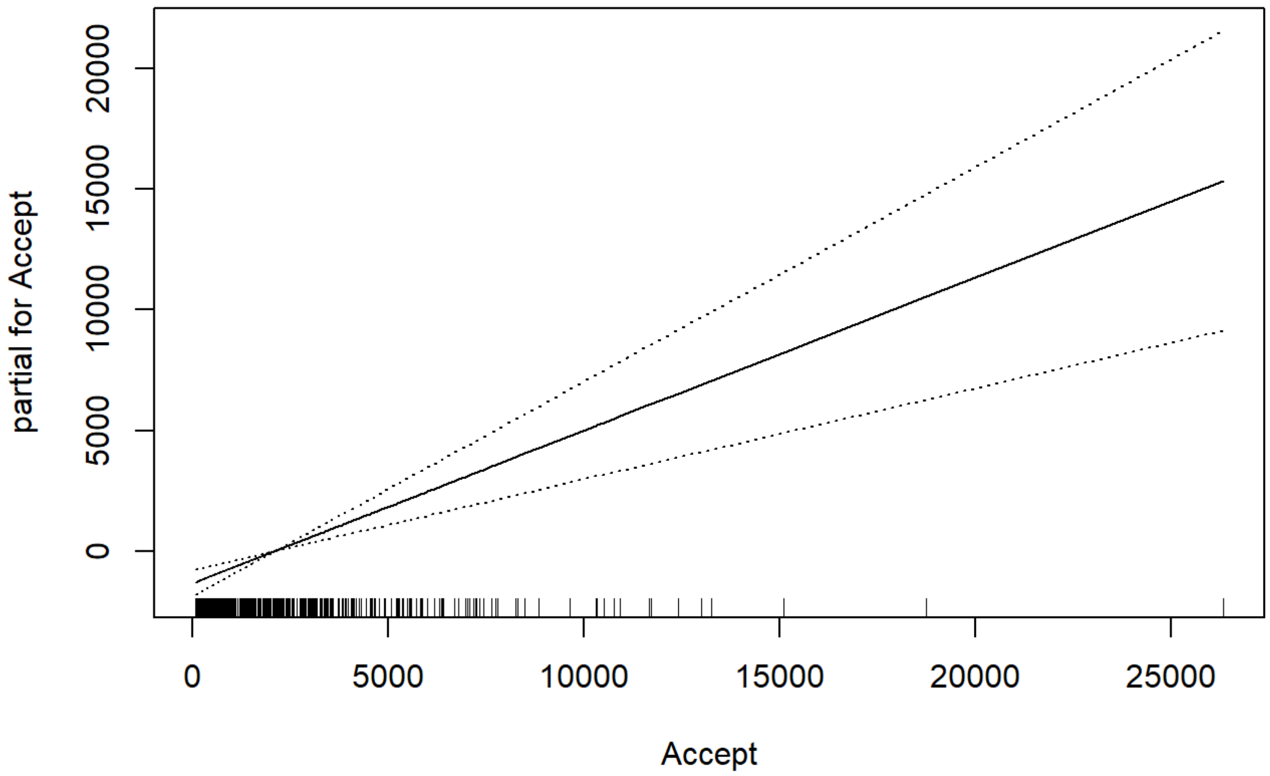
```
## 1 rmse    standard    1944.
## 2 rsq     standard       0.760
## 3 mae     standard    1530.
```

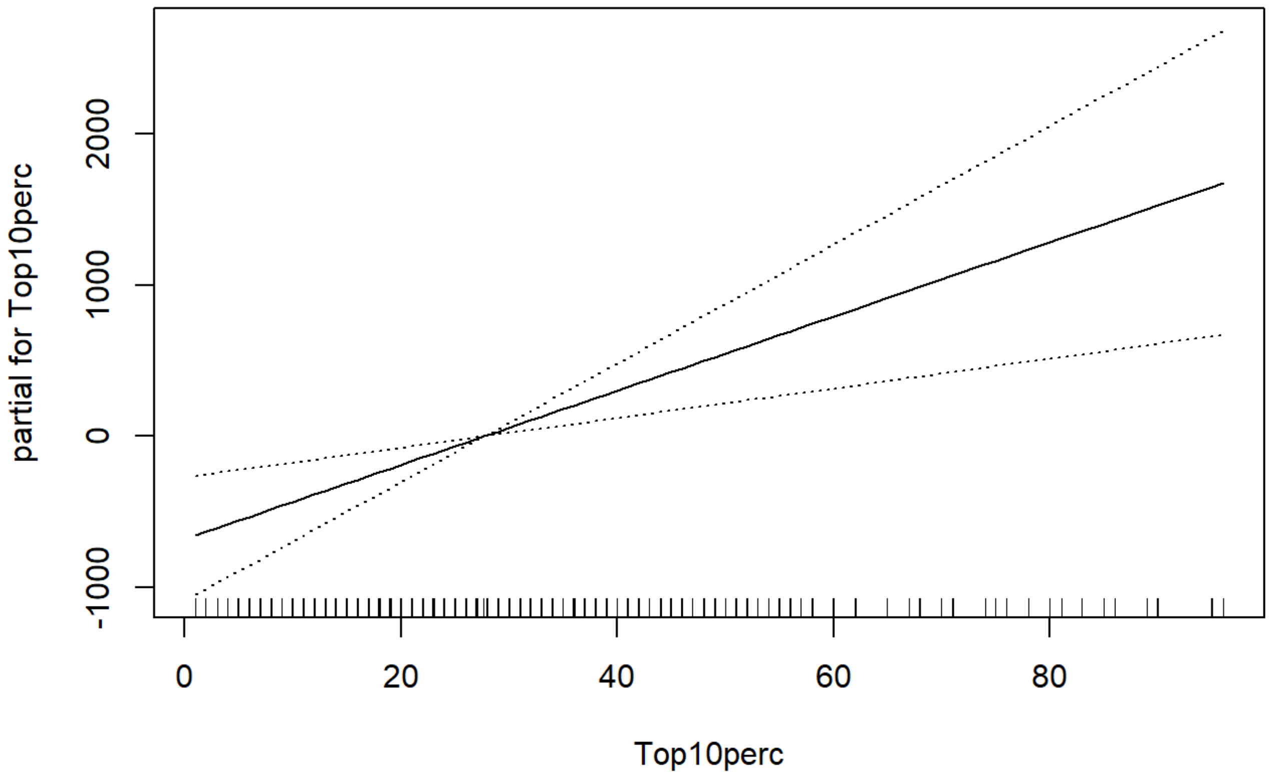
```
plot(gam_fit, se = TRUE)
```


No
Yes
partial for Private
500
0
-1000
-2000
Private

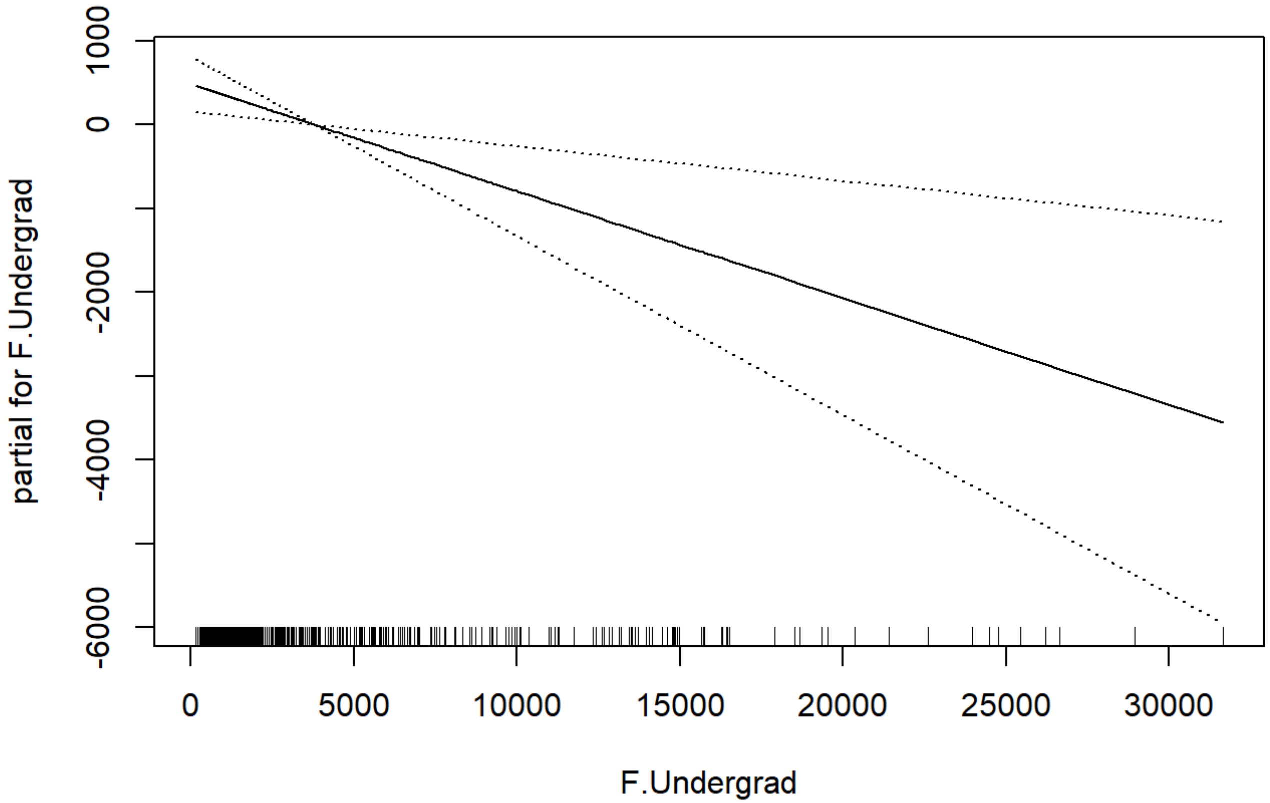

0
-5000

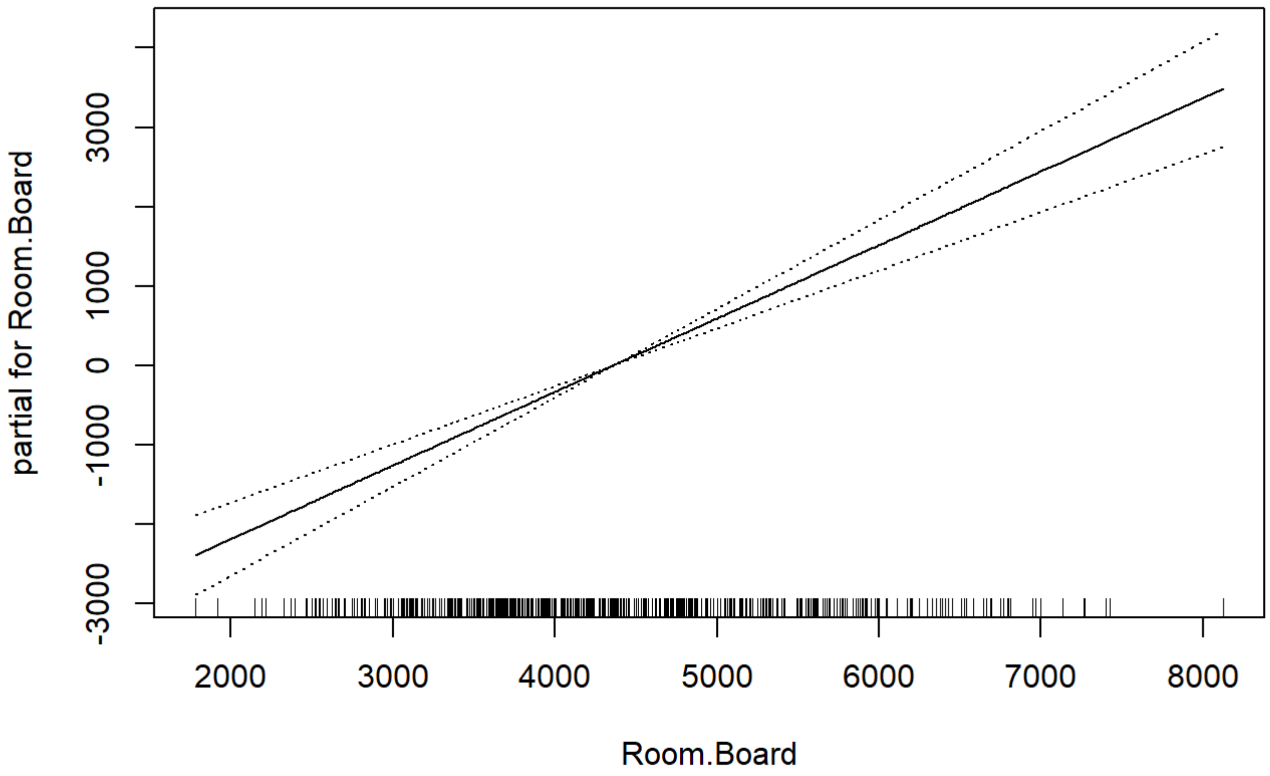

partial for Ap
-10000
-15000
0
10000
20000
30000
40000
50000
Apps


partial for Accept
0
5000
10000
15000
20000
0
5000
10000
15000
20000
25000
Accept

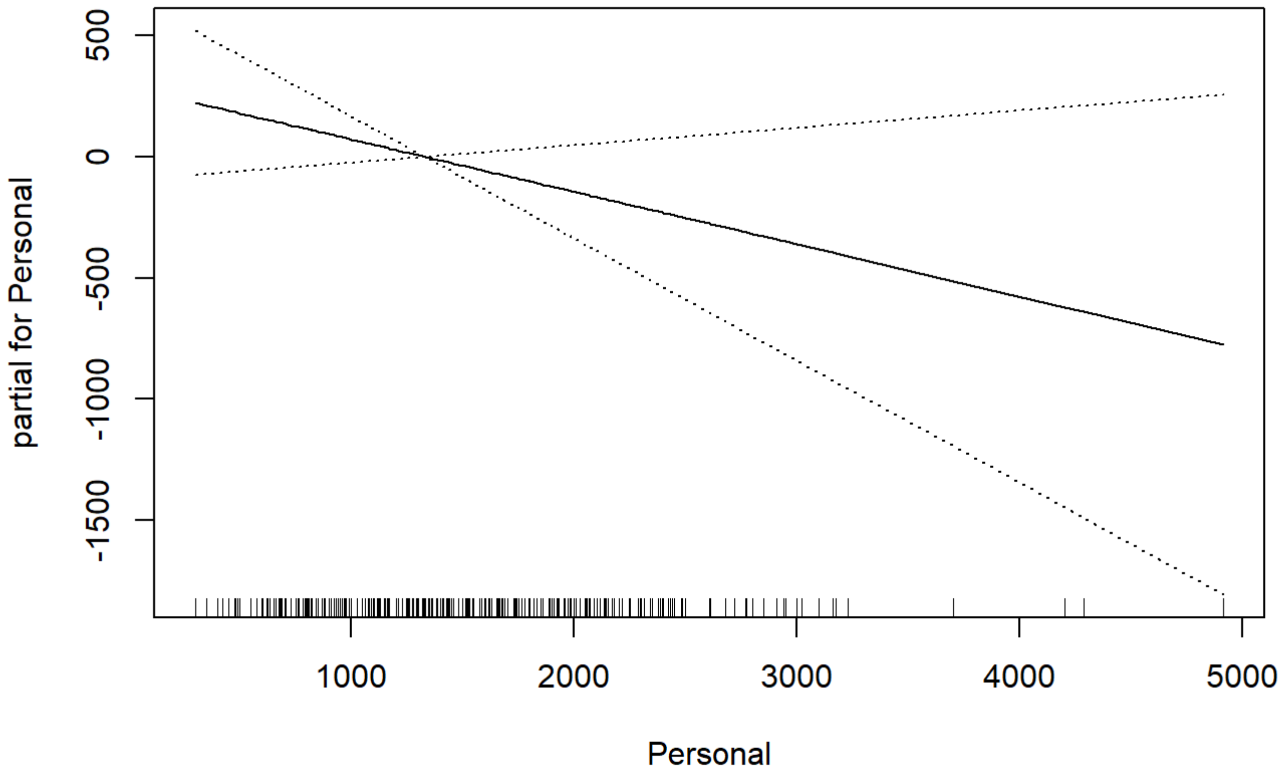


partial for Top10perc
-1000
0
1000
2000
0
20
40
60
80
Top10perc

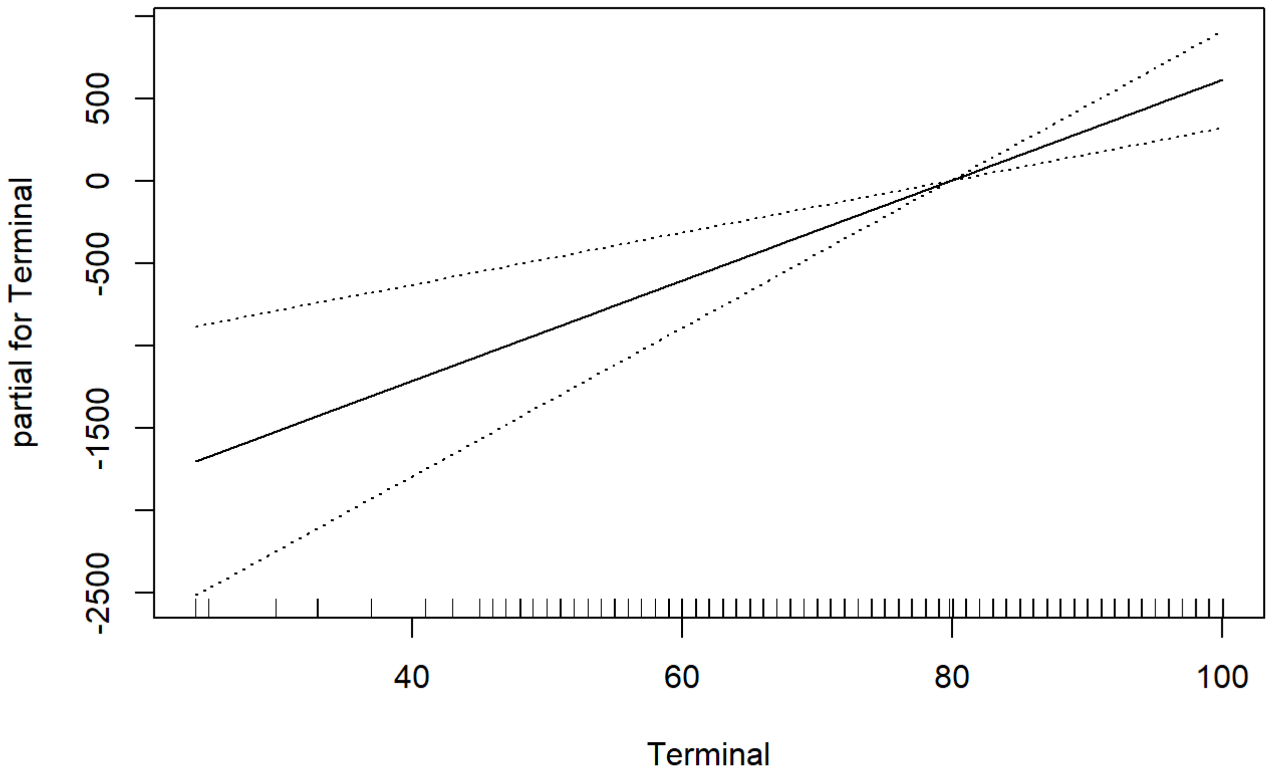

partial for F.Undergrad
1000
0
-2000
-4000
-6000
0
5000
10000
15000
20000
25000
30000
F.Undergrad

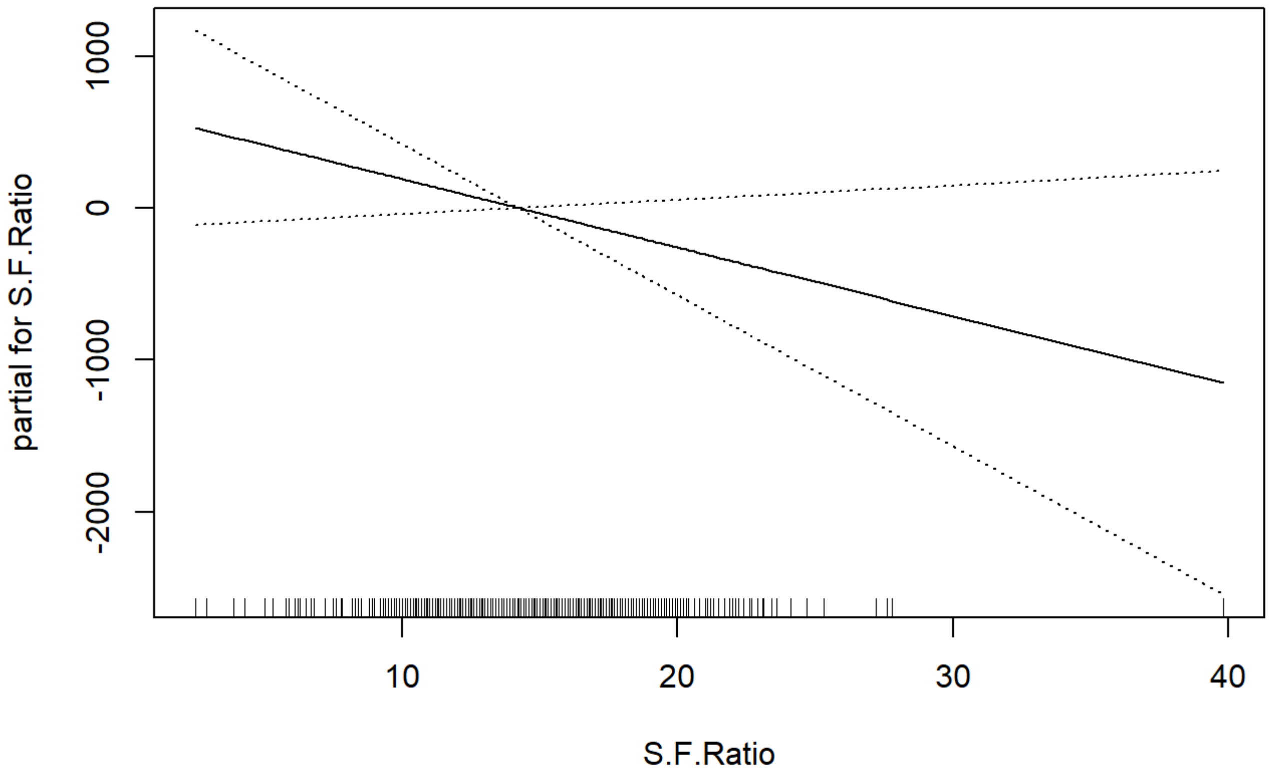


partial for Room.Board
-3000
-1000
0
1000
3000
2000
3000
4000
5000
6000
7000
8000
Room.Board

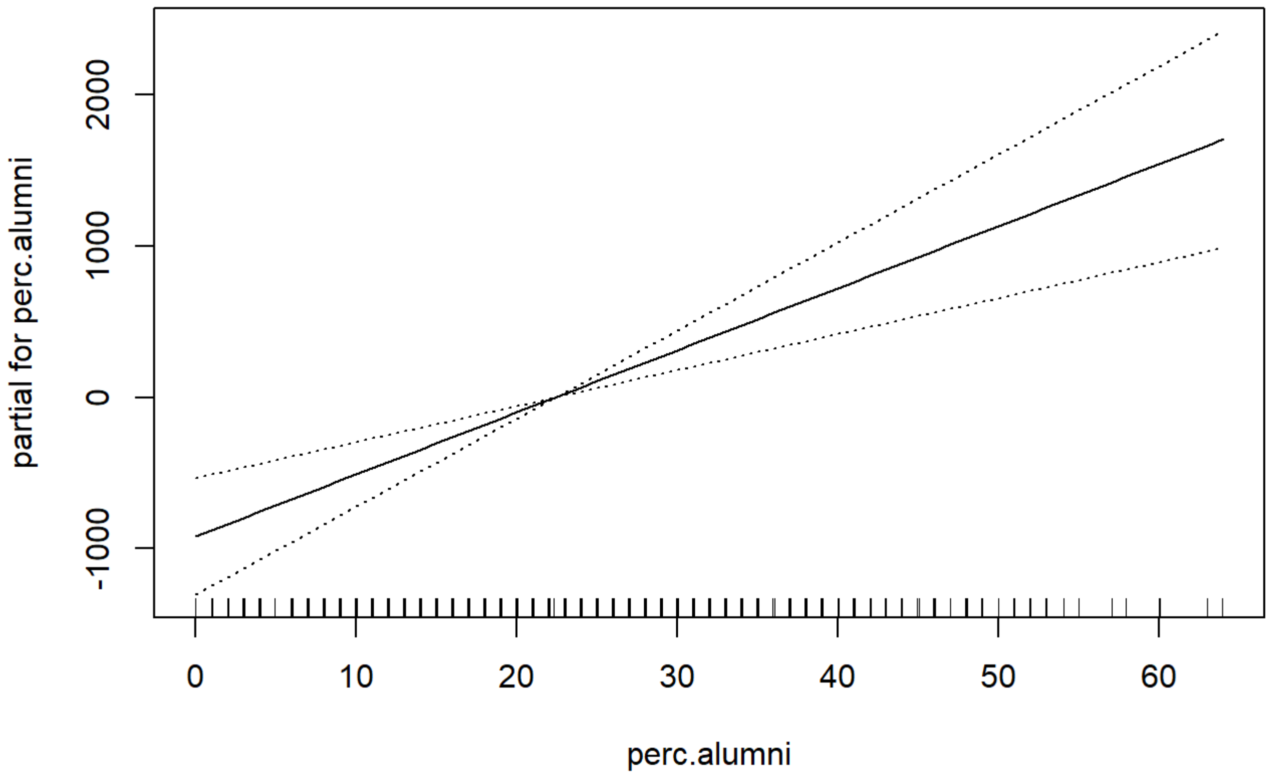


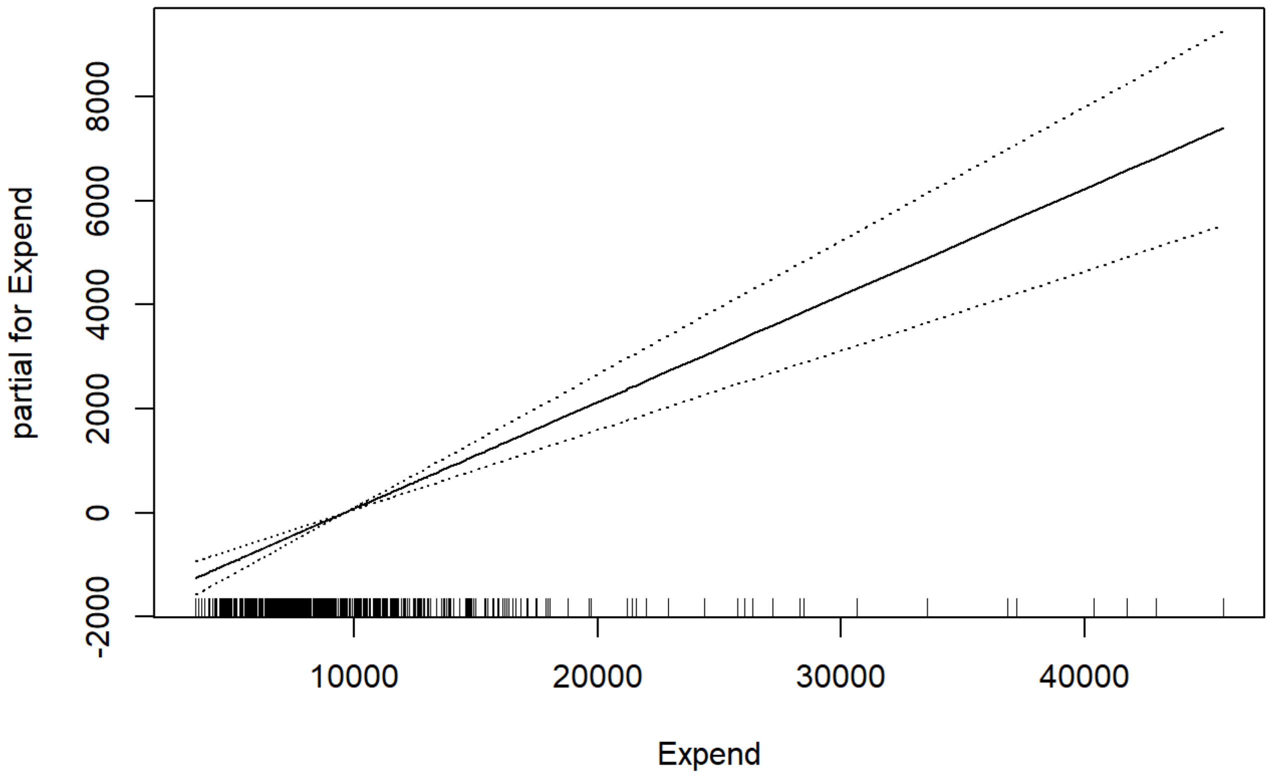
500
0
-500
-1000
-1500
partial for Personal
1000
2000
3000
4000
5000
Personal


partial for Terminal
500
0
-500
-1500
-2500
40
60
80
100
Terminal

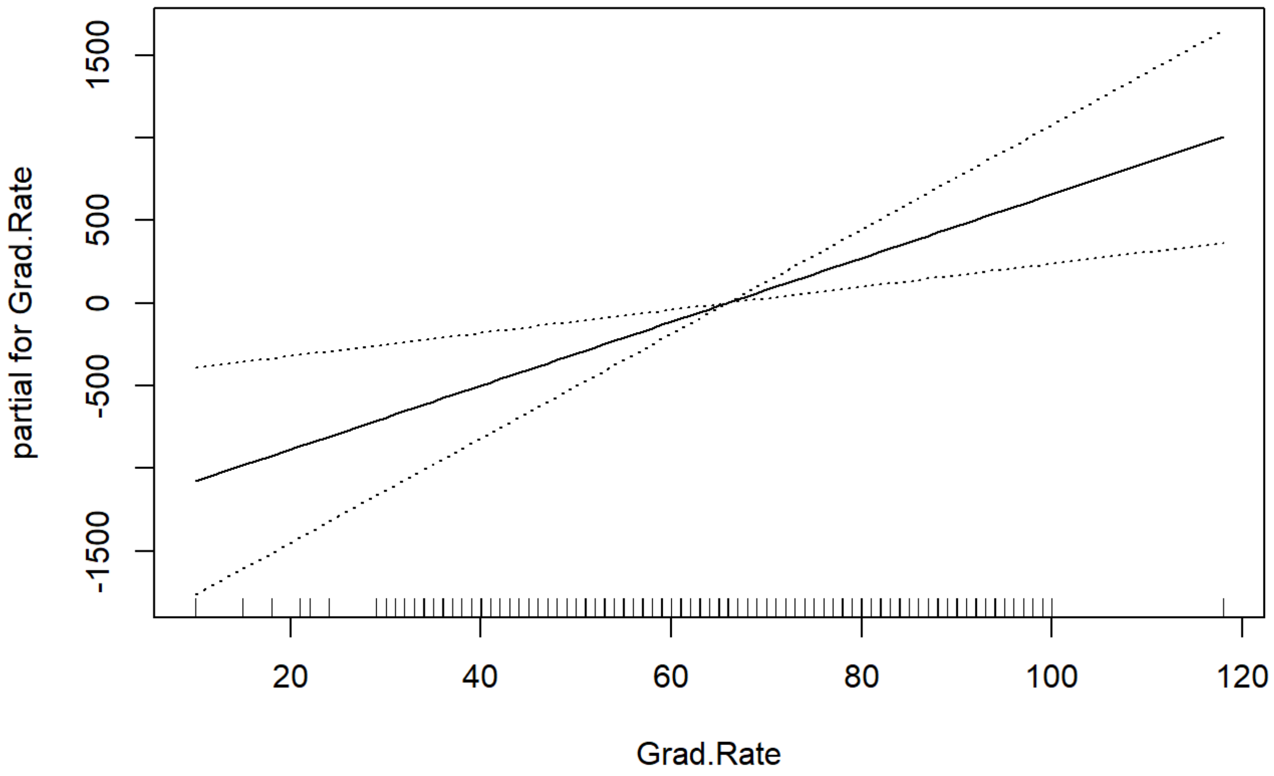


partial for S.F.Ratio
1000
0
-1000
-2000
10
20
30
40
S.F.Ratio


partial for perc.alumni
-1000
0
1000
2000
0
10
20
30
40
50
60
perc.alumni



-2000
0
2000
4000
6000
8000
partial for Expend
10000
20000
30000
40000
Expend



A good training R^2 of .78

(c)

```
test_preds <- predict(gam_fit, college_test_p12)

college_test_p12 %>%
  cbind(pred_Outstate = test_preds) %>%
  metrics(truth = Outstate, estimate = pred_Outstate)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    1959.
## 2 rsq     standard       0.781
## 3 mae     standard    1530.
```

The model performs slightly worse, as expected, on the test data. Training RMSE and R^2 were 2168 and .75, while on training were 1867 and 77.

(d)

The plots that go variable-by-variable when running `plot(gam_fit, se = TRUE)` are all linear, indicating a lack of non-linear relationship with the response.