CSCI E-106:Assignment 11 Section RMD using State data

Ian Kelk

Contents


```
# Ensure required packages are installed, then load them
options(repos = c(CRAN = "https://cloud.r-project.org"))

req_pkgs <- c("faraway", "dplyr", "tibble", "tidyr", "ggplot2",
              "knitr", "leaps", "glmnet", "rpart", "rpart.plot", "broom",
              "car", "lmtest", "MASS")

to_install <- setdiff(req_pkgs, rownames(installed.packages()))
if (length(to_install)) {
  install.packages(to_install, dependencies = TRUE)
}

# Load quietly
invisible(lapply(req_pkgs, function(p) {
  suppressPackageStartupMessages(library(p, character.only = TRUE))
}))
```

Due Date: December 5, 2025 at 11:59 pm EST

Instructions

Students should submit their reports on Canvas. The report needs to clearly state what question is being solved, step-by-step walk-through solutions, and final answers clearly indicated. Please solve by hand where appropriate.

Please submit two files: (1) a R Markdown file (.Rmd extension) and (2) a PDF document, word, or html generated using knitr for the .Rmd file submitted in (1) where appropriate. Please, use RStudio Cloud for your solutions.

Problem 1

Refer to the Prostate cancer data set. Serum prostate-specific antigen (PSA) was determined in 97 men with advanced prostate cancer. PSA is a well-established screening test for prostate cancer and the oncologists wanted to examine the correlation between level of PSA and a number of clinical measures for men who were about to undergo radical prostatectomy.

The measures are cancer volume, prostate weight, patient age, the amount of benign prostatic hyperplasia, seminal vesicle invasion, capsular penetration, and Gleason score. (100 pts)

a-) Use 70% of the data for train data set and use remaining data (30% of data) for test data set (Please use set.seed(567)). (10 points)

use the train data set for the questions below

b-) Develop a best subset model for predicting PSA. Justify your choice of model. Assess the model performance and check the model assumptions.(10 pts)

c-) Develop a regression tree for predicting PSA. Assess the model performance and check the model assumptions. (15 pts)

d-) Use the lasso regression to predict the PSA level.(15 pts)

e-) Based on the performances of the model on the train data set, which model would you choose? (10 pts)

use the test data set for the questions below

f-) Assess the best subset model performance on the test data set. Is the model performance stable (10 pts).

g-) Assess the regression tree model performance on the test data set. Is the model performance stable (10 pts).

h-) Assess the Lasso regression model performance on the test data set. Is the model performance stable (10 pts).

i-) Based on the performances on the test data set, which model would you choose? (10pts)

Problem 1

Refer to the Prostate cancer data set. Serum prostate-specific a. . . y. The measures include cancer volume, prostate weight, patient

age, the amount of benign prostatic hyperplasia, and several oth. . . ors. The goal is to model PSA using these clinical measurements.

IMPORTANT: WE ARE NOT USING THE PROSTATE CANCER DATA FOR THIS SECTION RMD! WE ARE USING STATE DATA. DO NOT RELY ON THE OUTPUTS OF THIS FILE TO INFLUENCE YOUR ACTUAL HOMEWORK ANALYSIS!

For this solution we will use the built-in `state.x77` dataset from base R. This dataset records several numeric characteristics of the 50 U.S. states, including population, income, illiteracy rate, life expectancy, murder rate, high-school graduation rate, number of frost days, and land area. We will treat life expectancy (`LifeExp`) as the outcome in all models below.

```
# Use the built-in state.x77 dataset (50 U.S. states, 8 numeric variables)
states_df <- as.data.frame(state.x77)

# Give the columns convenient names
names(states_df) <- c(
  "Population",  # population (in thousands)
  "Income",      # per-capita income (USD)
  "Illiteracy",  # illiteracy rate (%)
  "LifeExp",     # life expectancy (years)
  "Murder",      # murder rate (per 100,000)
  "HSGrad",      # high school graduation rate (%)
  "Frost",       # mean number of days with minimum temperature below freezing
  "Area"         # land area (thousands of square miles)
)

# Inspect the data
glimpse(states_df)
```

```
## Rows: 50
## Columns: 8
## $ Population <dbl> 3615, 365, 2212, 2110, 21198, 2541, 3100, 579, 8277, 4931, ~
## $ Income     <dbl> 3624, 6315, 4530, 3378, 5114, 4884, 5348, 4809, 4815, 4091,~
## $ Illiteracy <dbl> 2.1, 1.5, 1.8, 1.9, 1.1, 0.7, 1.1, 0.9, 1.3, 2.0, 1.9, 0.6,~
## $ LifeExp    <dbl> 69.05, 69.31, 70.55, 70.66, 71.71, 72.06, 72.48, 70.06, 70.~
## $ Murder     <dbl> 15.1, 11.3, 7.8, 10.1, 10.3, 6.8, 3.1, 6.2, 10.7, 13.9, 6.2~
```

```
## $ HSGrad      <dbl> 41.3, 66.7, 58.1, 39.9, 62.6, 63.9, 56.0, 54.6, 52.6, 40.6,~
## $ Frost       <dbl> 20, 152, 15, 65, 20, 166, 139, 103, 11, 60, 0, 126, 127, 12~
## $ Area        <dbl> 50708, 566432, 113417, 51945, 156361, 103766, 4862, 1982, 5~
```

```r
# Response is LifeExp (life expectancy); predictors are the remaining columns
```

1(a) Train / Test Split (10 pts)

We now set the random seed, create a 70/30 train/test split stratified by `LifeExp`, and report the sizes of the resulting samples.

```r
set.seed(567)

# Stratified 70/30 split based on the outcome LifeExp
train_index <- caret::createDataPartition(
  y    = states_df$LifeExp,
  p    = 0.7,
  list = FALSE
)

states_train <- states_df[train_index, ] |> as.data.frame()
states_test  <- states_df[-train_index, ] |> as.data.frame()

# Training/test sizes and shares (should be approx 0.70 / 0.30)
train_test_sizes <- tibble(
  sample = c("Training", "Test"),
  n      = c(nrow(states_train), nrow(states_test))
) |>
  mutate(share = n / nrow(states_df))

knitr::kable(
  train_test_sizes,
  digits  = 3,
  caption = "Training and test sample sizes and shares for the U.S. states data"
)
```

Table 1: Training and test sample sizes and shares for the U.S. states data

sample	n	share
Training	38	0.76
Test	12	0.24

```r
# Outcome distribution (LifeExp) overall vs training vs test
LifeExp_summary <- bind_rows(
  overall = states_df,
  train   = states_train,
  test    = states_test,
  .id     = "sample"
) |>
  group_by(sample) |>
  summarise(
    n            = n(),
    mean_LifeExp = mean(LifeExp),
    sd_LifeExp   = sd(LifeExp),
    min_LifeExp  = min(LifeExp),
    max_LifeExp  = max(LifeExp),
    .groups      = "drop"
  )
```

```r
knitr::kable(
  LifeExp_summary,
  digits  = 3,
  caption = "Distribution of LifeExp for the overall U.S. states data and the training/test splits"
)
```

Table 2: Distribution of LifeExp for the overall U.S. states data and the training/test splits

sample	n	mean_LifeExp	sd_LifeExp	min_LifeExp	max_LifeExp
overall	50	70.879	1.342	67.96	73.60
test	12	70.872	1.170	69.05	72.58
train	38	70.881	1.407	67.96	73.60

Commentary (1a).

What you should discuss for the homework:

Briefly describe the train/test split for the U.S. states data, including the number of states in the training and test sets, and comment on whether the distribution of life expectancy (`LifeExp`) looks similar across the overall sample, training set, and test set.

Conclusion: The stratified split produced a training set of 38 states (about 76% of the data) and a test set of 12 states (about 24%), as shown in Table 1. Here we used `caret::createDataPartition()` with `LifeExp` as the stratification variable, which groups the outcome into bins and then samples roughly 70% from each bin. Because of this binning and integer rounding within each group, the overall training share is only approximately 70%, not exactly 35 out of 50 states. The summary statistics in Table 2 indicate that the distribution of life expectancy is very similar across all three groups: the overall mean of `LifeExp` is about 70.88 years (SD $\approx$ 1.34), the training mean is 70.88 (SD $\approx$ 1.41), and the test mean is 70.87 (SD $\approx$ 1.17). The minimum and maximum life expectancies in the train and test sets fall within the overall range, with only small differences at the extremes. The split preserves the distribution of life expectancy well and does not introduce any obvious shift between training and test samples, so it is better to accept this approximate 70/30 stratified split than to try to "tune" p to force an exact 70/30 at the cost of less balanced outcome distributions.

1(b) Best Subset Regression for Life Expectancy (10 pts)

Develop a best subset model for predicting life expectancy (`LifeExp`). Justify your choice of model. Assess the model performance and check the model assumptions.

A note on `regsubsets()` vs `step()`

There are several ways to select a subset of predictors in a regression model in R. Two common ones are:

- **Best subset selection** via `leaps::regsubsets()`
- **Stepwise selection** via `step()`

`regsubsets()` is designed for best subset regression. For each possible model size (1 predictor, 2 predictors, 3 predictors, etc.), it searches across essentially all subsets of the predictors and finds the best model of that size according to criteria like adjusted R^2, C_p, or BIC. With a small number of predictors (like we have here), this exhaustive search is computationally cheap and gives us a clear "best" model for each size. With only 7 predictors in the states example (and 8 in prostate for your homework), the total number of possible subsets is small:

7 predictors -> $2^7 = 128$ possible models

8 predictors -> $2^8 = 256$ possible models

By contrast, `step()` implements a stepwise procedure. It starts from a given model (null, full, or something in between) and then adds or removes one predictor at a time based on a criterion such as AIC. This is a greedy algorithm: it only explores models along a single path and does not guarantee that the final model is the best possible subset of a given size.

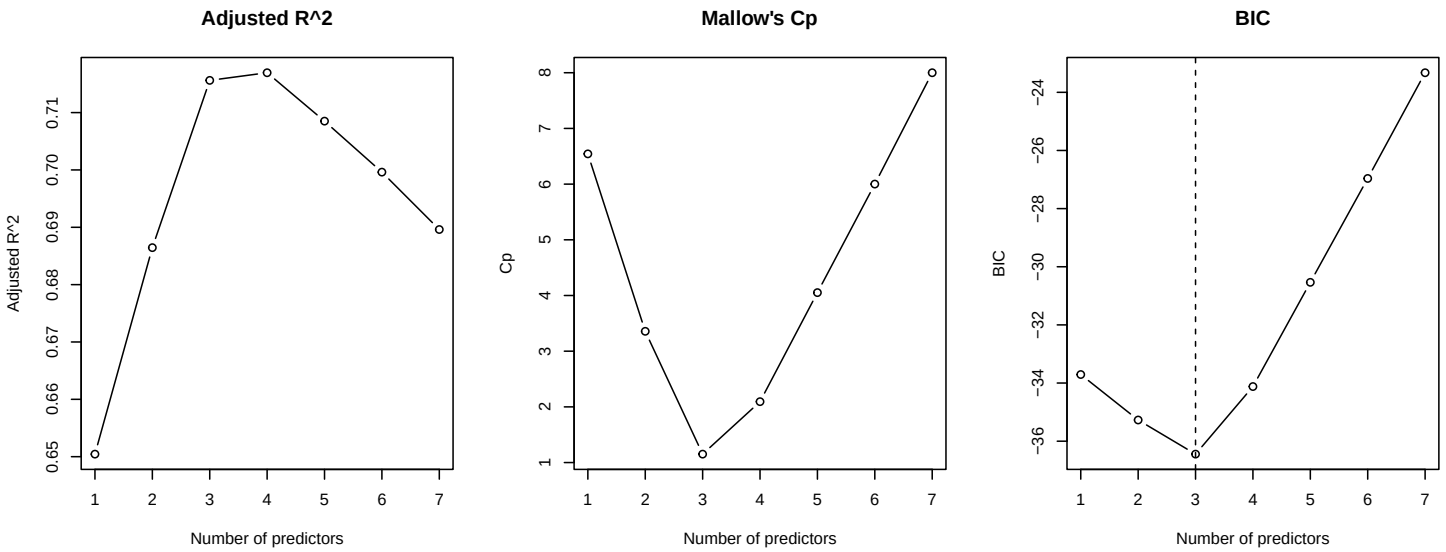
We explicitly want a best subset linear model and we only have a modest number of predictors, so we use `regsubsets()` to perform genuine best subset search and then choose among the candidate models using BIC.

```r
set.seed(567)

best_sub <- regsubsets(
  LifeExp ~ .,
  data   = states_train,
  nvmax  = ncol(states_train) - 1,  # up to all predictors
  method = "exhaustive"
)

best_sub_summary <- summary(best_sub)

# Extract key selection criteria
best_sub_tib <- tibble(
  nvar     = 1:length(best_sub_summary$cp),
  rss      = best_sub_summary$rss,
  adjr2    = best_sub_summary$adjr2,
  cp       = best_sub_summary$cp,
  bic      = best_sub_summary$bic
)

best_sub_tib
```

1(b)(i) Fit best subset models using `regsubsets()`

```
## # A tibble: 7 x 5
##    nvar   rss adjr2    cp   bic
##   <int> <dbl> <dbl> <dbl> <dbl>
## 1     1  24.9 0.650  6.54 -33.7
## 2     2  21.7 0.686  3.36 -35.3
## 3     3  19.1 0.716  1.15 -36.4
## 4     4  18.5 0.717  2.09 -34.1
## 5     5  18.5 0.709  4.05 -30.5
## 6     6  18.4 0.700  6.00 -27.0
## 7     7  18.4 0.690  8    -23.3
```

```r
par(mfrow = c(1, 3))

plot(best_sub_summary$adjr2, type = "b",
     xlab = "Number of predictors", ylab = "Adjusted R^2",
     main = "Adjusted R^2")

plot(best_sub_summary$cp, type = "b",
     xlab = "Number of predictors", ylab = "Cp",
     main = "Mallow's Cp")

plot(best_sub_summary$bic, type = "b",
     xlab = "Number of predictors", ylab = "BIC",
     main = "BIC")
abline(v = which.min(best_sub_summary$bic), lty = 2)
```




```
par(mfrow = c(1, 1))
```

A note on R^2, adjusted R^2, Mallow's C_p, and BIC

When we run `regsubsets()`, the summary gives us several statistics for each candidate model size (number of predictors):

- `adjr2` adjusted R^2
- `cp` Mallow's C_p
- `bic` Bayesian Information Criterion (BIC)

We then plot these (adjusted R^2, C_p, BIC) against the number of predictors to help choose a model.

- **R^2 and adjusted R^2**
  For a given linear model, R^2 measures the proportion of variability in the response explained by the predictors. It always increases (or stays the same) when you add more predictors, even if they are useless.
  Adjusted R^2 corrects for this by penalizing extra predictors:

$$R^2_{\text{adj}} = 1 - \frac{\text{RSS}/(n-p)}{\text{TSS}/(n-1)},$$

  where n is the sample size and p is the number of parameters (including the intercept). Adjusted R^2 can go **down** when you add a weak predictor, so we often look for the model with a **high** adjusted R^2 that doesn't require too many variables.

- **Mallow's C_p**
  Mallow's C_p is defined for least-squares regression models. For a model with p parameters and residual sum of squares RSS_p,

$$C_p = \frac{\text{RSS}_p}{\hat{\sigma}^2} - (n - 2p),$$

  where $\hat{\sigma}^2$ is an estimate of the error variance (often from the full model). Good models tend to have **small** C_p, and a common rule of thumb is to look for models where $C_p \approx p$.

- **BIC (Bayesian Information Criterion)**
  For a Gaussian linear model, BIC can be written (up to a constant) as

$$\text{BIC} = n \log\left(\frac{\text{RSS}}{n}\right) + k \log n + \text{constant},$$

  where k is the number of parameters. BIC rewards small residual error but penalizes larger models through the $k \log n$ term. When comparing models fit to the same data, we prefer the model with the **smallest BIC**. Compared to AIC, BIC uses a stronger penalty for extra predictors and therefore tends to favor more parsimonious models.

In this question, `regsubsets()` computes adjusted R^2, C_p, and BIC for the best model of each size. We examine the plots and then choose our final linear model by **minimizing BIC**, which usually gives a simple model that still fits the data well.

```r
# Example: choose the BIC-minimizing model
best_bic_size <- which.min(best_sub_summary$bic)
best_bic_size
```

1(b)(ii) Identify the chosen model and fit as `lm` because `regsubsets()` just returns information, not an actual model

```
## [1] 3
```

```r
# Identify which predictors are in that model
coef_best_bic <- coef(best_sub, best_bic_size)
coef_best_bic
```

```
##  (Intercept)       Murder       HSGrad        Frost
## 71.224826638 -0.297194062  0.045860560 -0.005821069
```

```r
# Build a formula string from those predictors
best_pred_names <- names(coef_best_bic)[-1]  # drop intercept
best_formula <- as.formula(
  paste("LifeExp ~", paste(best_pred_names, collapse = " + "))
)
best_formula
```

```
## LifeExp ~ Murder + HSGrad + Frost
```

```r
# Fit the chosen best-subset model as a standard lm
lm_best <- lm(best_formula, data = states_train)
summary(lm_best)
```

```
##
## Call:
## lm(formula = best_formula, data = states_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.60378 -0.49494  0.06292  0.41532  1.37901
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 71.224827   1.101027  64.689  < 2e-16 ***
## Murder      -0.297194   0.040185  -7.396 1.42e-08 ***
## HSGrad       0.045861   0.017401   2.635   0.0126 *
## Frost       -0.005821   0.002717  -2.142   0.0394 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7503 on 34 degrees of freedom
## Multiple R-squared:  0.7387, Adjusted R-squared:  0.7156
## F-statistic: 32.04 on 3 and 34 DF,  p-value: 5.1e-10
```

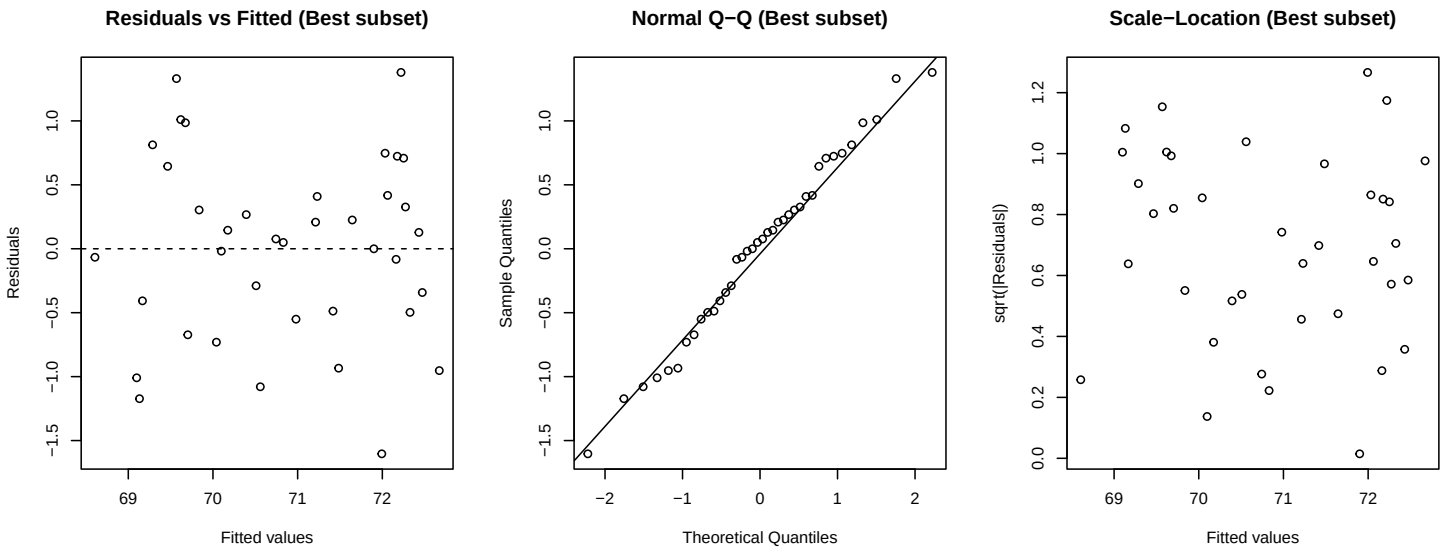
1(b)(iii) Assess model performance on training data We can look at training R^2, RMSE, and residual diagnostics.

```r
# Training performance metrics
train_aug_best <- augment(lm_best)

train_metrics_best <- train_aug_best %>%
  summarise(
    n       = n(),
    rmse    = sqrt(mean(.resid^2)),
    mae     = mean(abs(.resid)),
    rsq     = 1 - sum(.resid^2) / sum((LifeExp - mean(LifeExp))^2)
  )
```

```
train_metrics_best
```

```
## # A tibble: 1 x 4
##       n  rmse   mae   rsq
##   <int> <dbl> <dbl> <dbl>
## 1    38 0.710 0.574 0.739
```

```
par(mfrow = c(1, 3))

# Residuals vs fitted
plot(lm_best$fitted.values, resid(lm_best),
     xlab = "Fitted values", ylab = "Residuals",
     main = "Residuals vs Fitted (Best subset)")
abline(h = 0, lty = 2)

# Normal Q-Q plot
qqnorm(resid(lm_best), main = "Normal Q-Q (Best subset)")
qqline(resid(lm_best))

# Scale-Location plot
plot(lm_best$fitted.values, sqrt(abs(resid(lm_best))),
     xlab = "Fitted values", ylab = "sqrt(|Residuals|)",
     main = "Scale-Location (Best subset)")
```




```
par(mfrow = c(1, 1))
```

A note on `crPlots()` (component-plus-residual plots)

In addition to the usual residual diagnostics, we can also look at the linearity of each predictor's effect using `car::crPlots()`. This function produces component-plus-residual (partial residual) plots for each predictor in the model. For a given predictor X_j, it plots a transformed version of the residuals against X_j, conditional on the other predictors.

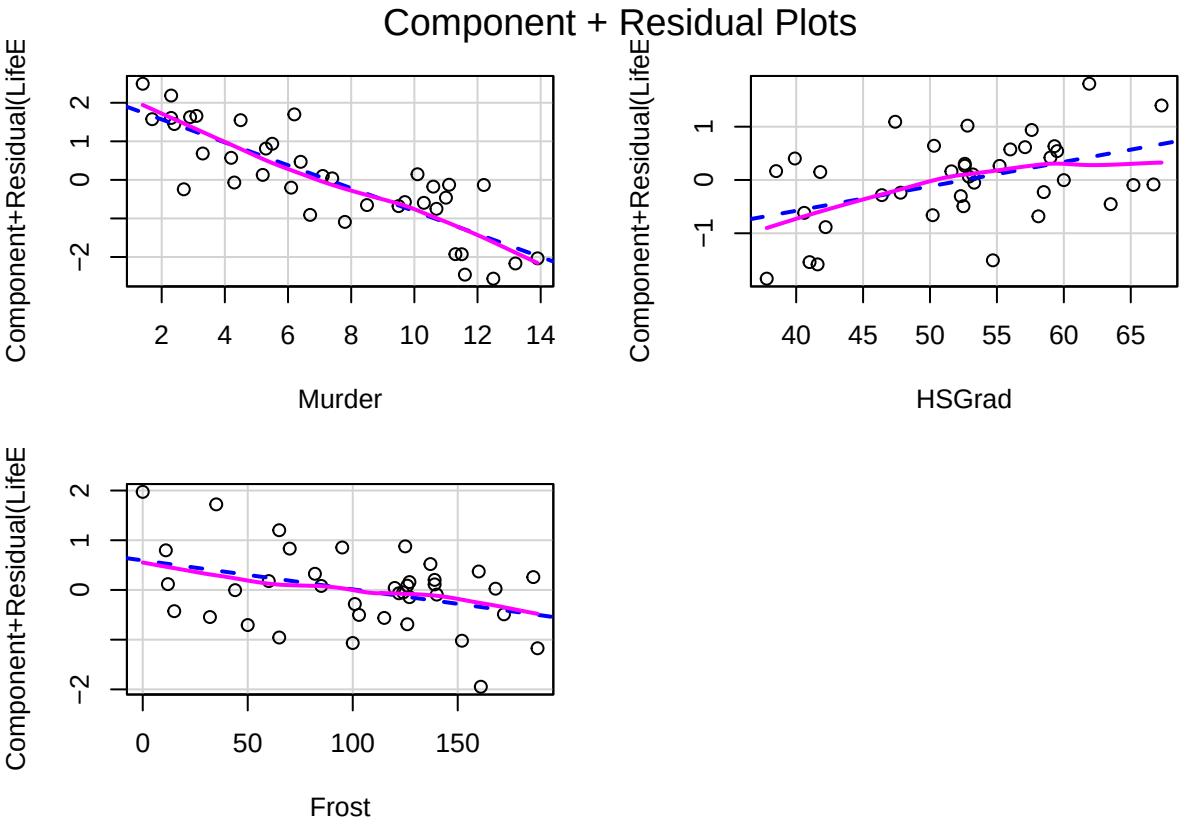
If the true relationship between X_j and the response is approximately linear (after accounting for the other variables), the points in the component-plus-residual plot should lie roughly along a straight line. Systematic curvature in these plots is a warning sign that the linearity assumption may not hold for that predictor and that a transformation or more flexible term might be needed, such as adding a squared term (e.g., `I(Murder^2)`), using a transformation like `log(Murder)` or `sqrt(Murder)`, or including an interaction term if the effect seems to depend on another variable.

```
# Breusch-Pagan test for heteroskedasticity
lmtest::bptest(lm_best)
```

```
##
##  studentized Breusch-Pagan test
##
```

```
## data:  lm_best
## BP = 2.0762, df = 3, p-value = 0.5567
```

```
# Component-plus-residual plots for linearity
car::crPlots(lm_best)
```



Commentary (1b).

What you should discuss for the homework:

- State which model you selected (e.g., "We chose the BIC-minimizing model with predictors . . . ").
- Report the key training metrics (e.g., RMSE, R^2) using the output above.
- Comment on the residual plots and tests:
  - Does the residual vs fitted plot show any strong patterns?
  - Does the QQ-plot suggest approximately normal residuals?
  - Do the Breusch–Pagan and component-plus-residual plots suggest any serious violations of linearity or constant variance?

Conclusion: Using `regsubsets()`, we selected the model that minimizes BIC, which is the 3-predictor model with `Murder`, `HSGrad`, and `Frost` as predictors. The fitted best-subset model is

$$\text{LifeExp}_i = \beta_0 + \beta_1\,\text{Murder}_i + \beta_2\,\text{HSGrad}_i + \beta_3\,\text{Frost}_i + \varepsilon_i.$$

with estimated coefficients indicating that higher murder rates and more frost days are associated with lower life expectancy, while higher high-school graduation rates are associated with higher life expectancy. All three predictors are statistically significant at the 5% level.

On the training data, this model achieves RMSE ≈ 0.710, MAE ≈ 0.574, and $R^2 \approx 0.739$, so it explains about 74% of the variability in life expectancy across the states. The residuals-versus-fitted plot shows a fairly random scatter around zero without a strong trend or funnel shape, and the normal Q–Q plot is close to a straight line with only mild deviations in the tails. The Breusch–Pagan test (BP ≈ 2.08, df $= 3$, $p \approx 0.56$) does not provide evidence of heteroskedasticity, and the component-plus-residual plots suggest approximately linear relationships between `LifeExp` and each of the selected predictors. The usual linear model assumptions appear reasonably well satisfied for this best-subset model.

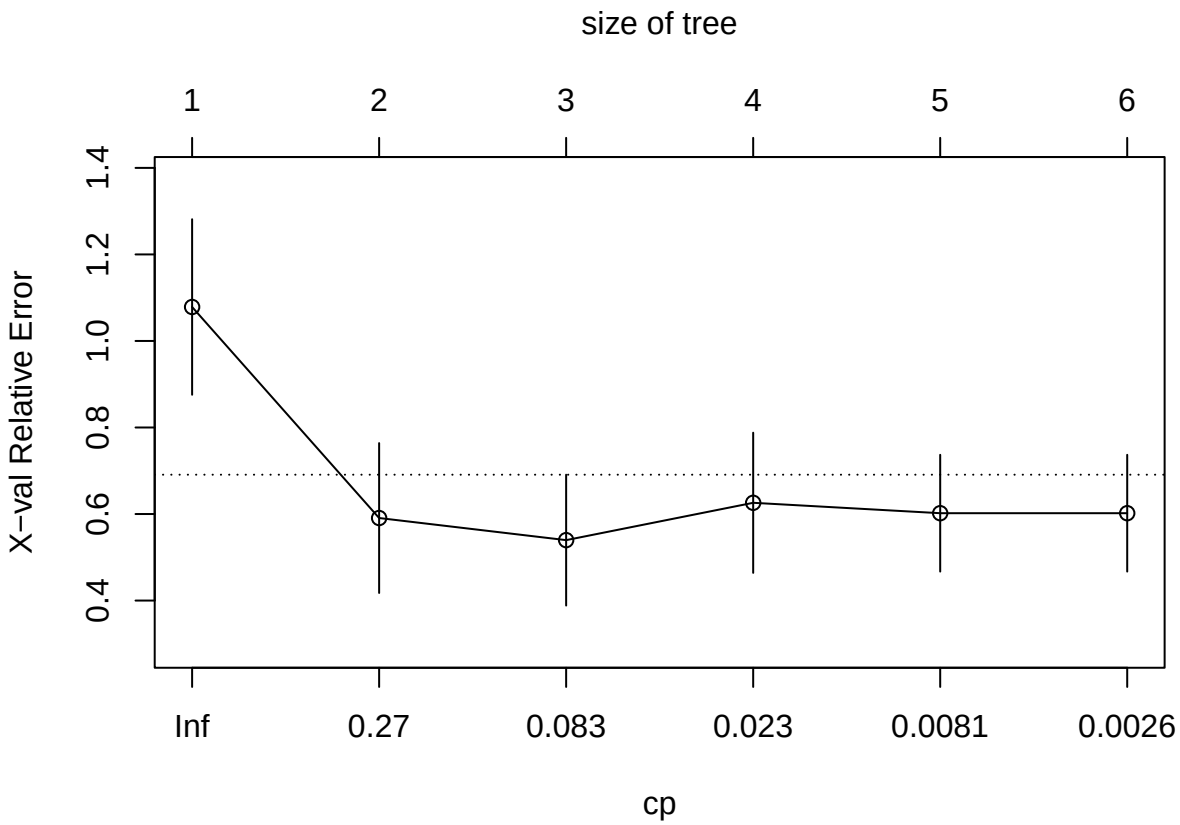
1(c) Regression Tree for Life Expectancy (15 pts)

Develop a regression tree for predicting life expectancy (`LifeExp`) and assess its performance and assumptions.

```
set.seed(567)

tree_life <- rpart(
  LifeExp ~ .,
  data   = states_train,
  method = "anova",
  control = rpart.control(
    cp        = 0.001, # small cp to allow a fairly large tree
    minsplit  = 10,
    xval      = 10
  )
)

printcp(tree_life)
```

```
##
## Regression tree:
## rpart(formula = LifeExp ~ ., data = states_train, method = "anova",
##     control = rpart.control(cp = 0.001, minsplit = 10, xval = 10))
##
## Variables actually used in tree construction:
## [1] HSGrad Murder
##
## Root node error: 73.24/38 = 1.9274
##
## n= 38
##
##          CP nsplit rel error  xerror    xstd
## 1 0.5980530      0   1.00000 1.07847 0.20288
## 2 0.1246102      1   0.40195 0.59072 0.17323
## 3 0.0551181      2   0.27734 0.53962 0.15125
## 4 0.0094602      3   0.22222 0.62594 0.16202
## 5 0.0068994      4   0.21276 0.60188 0.13496
## 6 0.0010000      5   0.20586 0.60188 0.13496
```

```
plotcp(tree_life)
```



We first let the tree grow fairly large, then use the cp table to prune it back to a smaller subtree. Pruning removes weaker splits, reducing both the depth and breadth of the tree, so the final model is simpler and typically generalizes better to new data.

The `plotcp()` output shows the cross-validated relative error (`xerror`) for candidate subtrees of different sizes. Each point corresponds to a pruned tree size (number of terminal nodes), and the vertical bars show +/-1 standard error. The lowest point gives the tree with the smallest cross-validated error; we use that cp value (`cp_min`) to prune the tree (minimum xerror rule). An alternative is the 1-SE rule, which chooses the simplest tree whose xerror is within one standard error of the minimum.

Use the complexity parameter (cp) table to prune to an appropriate subtree (e.g., cp that minimizes xerror or the 1-SE rule).

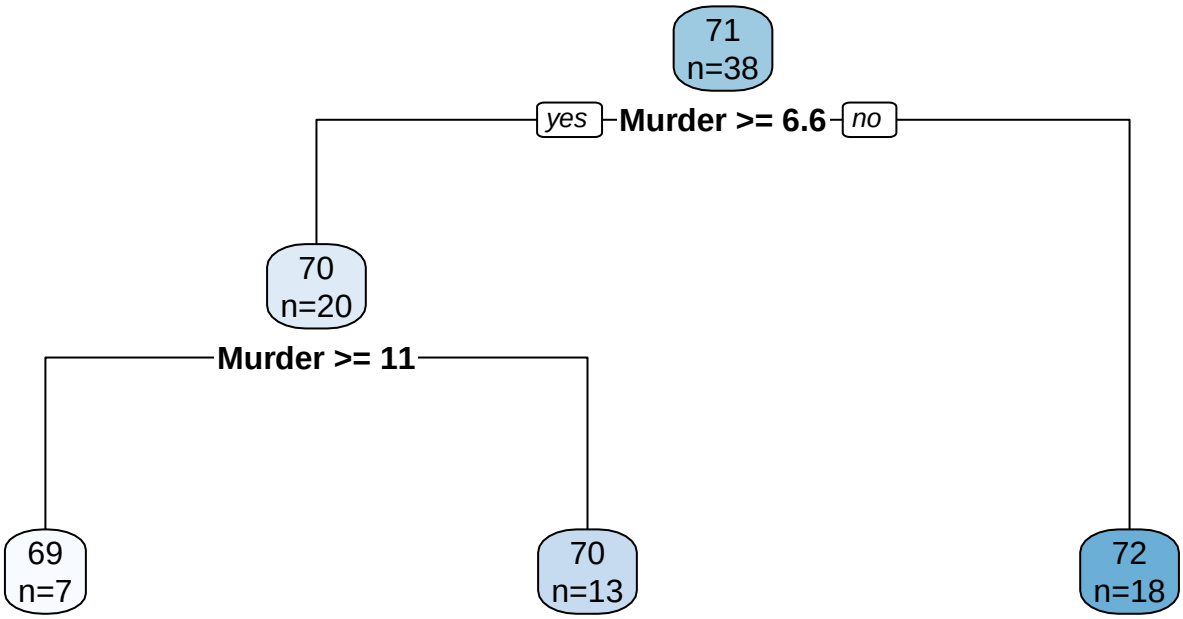
```
cp_min <- tree_life$cptable[which.min(tree_life$cptable[, "xerror"]), "CP"]
cp_min
```

```
## [1] 0.05511808
```

```
tree_pruned <- prune(tree_life, cp = cp_min)
tree_pruned
```

```
## n= 38
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 38 73.239590 70.88053
##   2) Murder>=6.55 20 16.811720 69.86200
##     4) Murder>=11.2 7  5.861886 68.94143 *
##     5) Murder< 11.2 13  1.823431 70.35769 *
##   3) Murder< 6.55 18 12.626710 72.01222 *
```

```
rpart.plot(
  tree_pruned,
  type  = 2,
  extra = 1,
  main  = "Regression tree for life expectancy (pruned)"
)
```



In each node, the large number is the predicted value of LifeExp (the mean life expectancy for states in that node, rounded), and the n = line shows how many training states fall into that node.

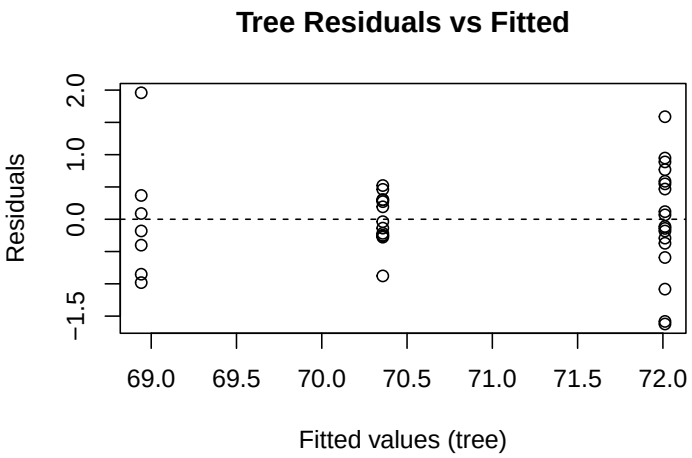
```
train_pred_tree <- predict(tree_pruned, newdata = states_train)

tree_train_metrics <- tibble(
  LifeExp_obs = states_train$LifeExp,
  LifeExp_hat = train_pred_tree,
  resid       = LifeExp_obs - LifeExp_hat
) %>%
  summarise(
    n    = n(),
    rmse = sqrt(mean(resid^2)),
    mae  = mean(abs(resid)),
    rsq  = 1 - sum(resid^2) / sum((LifeExp_obs - mean(LifeExp_obs))^2)
  )

tree_train_metrics
```

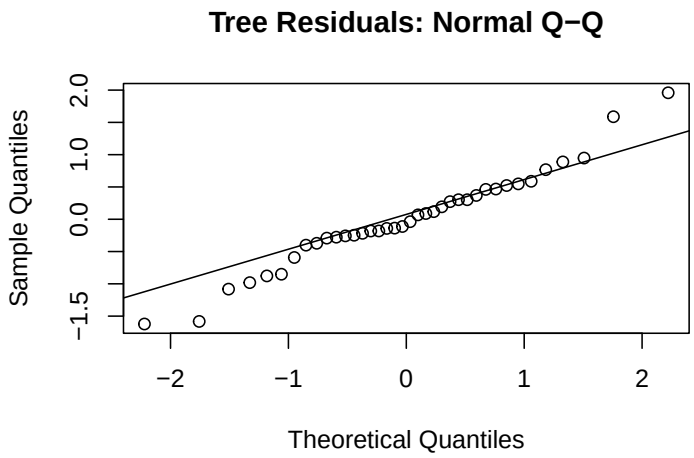
Tree performance on training data

```
## # A tibble: 1 x 4
##       n  rmse   mae   rsq
##   <int> <dbl> <dbl> <dbl>
## 1    38 0.731 0.550 0.723
```

```
par(mfrow = c(1, 2))

plot(train_pred_tree,
     states_train$LifeExp - train_pred_tree,
     xlab = "Fitted values (tree)",
     ylab = "Residuals",
     main = "Tree Residuals vs Fitted")
```

```r
abline(h = 0, lty = 2)

qqnorm(states_train$LifeExp - train_pred_tree,
       main = "Tree Residuals: Normal Q-Q")
qqline(states_train$LifeExp - train_pred_tree)
```






```r
par(mfrow = c(1, 1))
```

Commentary (1c).

What you should discuss for the homework:

- Describe the main splits of the tree (which predictors appear, in what order).
- Report the training RMSE and R^2.
- Comment on the residual plots: are there any strong patterns or non-normality signals?
- Briefly compare the tree's training performance and interpretability to the best subset model from 1(b).

Conclusion: The pruned regression tree for life expectancy first splits on `Murder` at about 6.6, separating higher-murder states (with lower average life expectancy) from lower-murder states (with higher average life expectancy). Within the higher-murder group, a further split at `Murder` ≈ 11 distinguishes a small group of very high-murder states with the lowest life expectancy (around 68.9 years) from states with intermediate murder rates (average life expectancy around 70.4 years). States with lower murder rates than 6.6 form a third leaf with the highest average life expectancy (about 72.0 years). The pruned tree summarizes life expectancy in terms of a few interpretable thresholds on murder rate.

On the training set, the pruned tree has RMSE ≈ 0.731, MAE ≈ 0.550, and $R^2 \approx 0.723$, which is very similar to the best-subset linear model in terms of RMSE and R^2. The residuals-versus-fitted plot shows residuals scattered symmetrically around zero with no strong pattern, and the Q–Q plot is close to linear with only modest tail deviations, suggesting no severe departures from normality. Thus, on the training data the tree provides a good fit with an easily interpretable set of murder-rate thresholds, though it does not dramatically outperform the best-subset linear model.

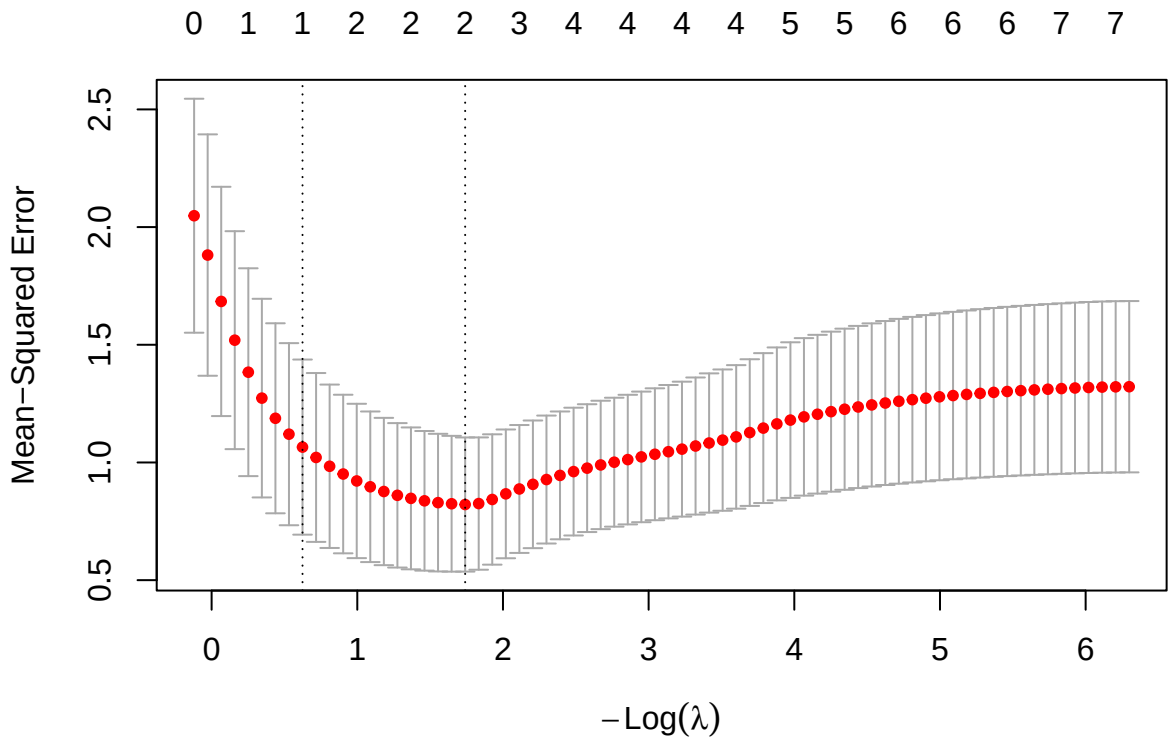
1(d) Lasso Regression for Life Expectancy (15 pts)

Use lasso regression to predict life expectancy (`LifeExp`). We will use `glmnet` with cross-validation to choose λ.

```r
# Create model matrices (drop intercept)
x_train <- model.matrix(LifeExp ~ ., data = states_train)[, -1]
y_train <- states_train$LifeExp

x_test  <- model.matrix(LifeExp ~ ., data = states_test)[, -1]
y_test  <- states_test$LifeExp
```

```r
set.seed(567)

cv_lasso <- cv.glmnet(
  x_train, y_train,
  alpha  = 1,        # lasso
  nfolds = 10
```

```
)

plot(cv_lasso)
```




```
lambda_min  <- cv_lasso$lambda.min
lambda_1se  <- cv_lasso$lambda.1se
lambda_min
```

```
## [1] 0.175443
```

```
lambda_1se
```

```
## [1] 0.5357777
```

Recall that in lasso regression, λ is the strength of the penalty on the size of the coefficients.

When λ is large, the penalty is strong: many coefficients are shrunk toward zero and some are forced exactly to zero (very simple, high-bias model). When λ is small, the penalty is weak: coefficients look more like ordinary least squares (more complex, low-bias model). Cross-validation is just helping us pick a good value of λ that balances fit vs. shrinkage.

`cv.glmnet()` performs K-fold cross-validation over a grid of λ values (here `nfolds = 10`). For each candidate λ:

1. The training data are split into 10 folds.
2. For each fold in turn, the model is fit on the other 9 folds and its prediction error is computed on the held-out fold.
3. These errors are averaged to get an estimate of the out-of-sample prediction error for that λ.

This gives us two useful choices:

- `lambda.min`: the value of λ with the smallest average cross-validated error. This corresponds to the model with the best estimated predictive performance among the grid of λ values.
- `lambda.1se`: the largest value of λ whose cross-validated error is within one standard error of the minimum. This typically gives a simpler, more heavily regularized model that performs almost as well as `lambda.min` but with more shrinkage and often fewer non-zero coefficients.

`lambda.min` chooses the model with the best estimated prediction error; `lambda.1se` chooses the simplest model whose error is statistically indistinguishable from that best one.

We will use `lambda.min` for a more aggressively fit model. (You can also examine `lambda.1se` for a sparser model.)

```
lasso_model <- glmnet(
  x_train, y_train,
```

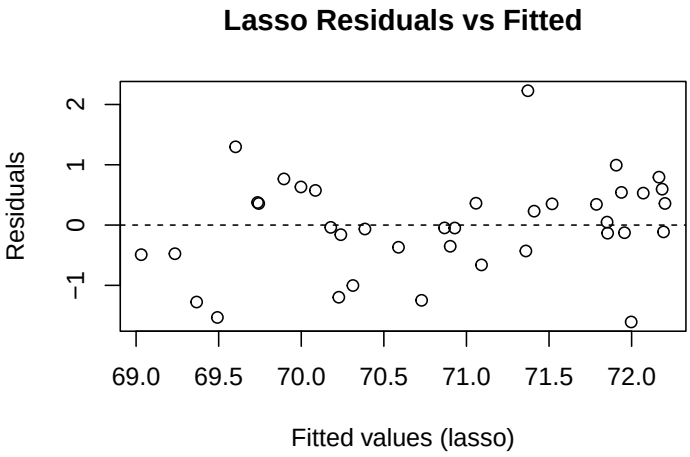
```
  alpha  = 1,
  lambda = lambda_min
)

coef(lasso_model)
```

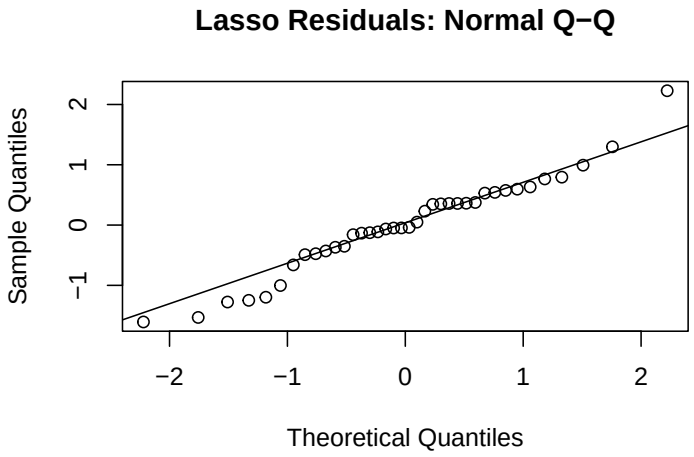
```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) 71.20307267
## Population   .
## Income       .
## Illiteracy   .
## Murder      -0.23213677
## HSGrad       0.02597882
## Frost        .
## Area         .
```

```
train_pred_lasso <- predict(lasso_model, newx = x_train)

lasso_train_metrics <- tibble(
  LifeExp_obs = y_train,
  LifeExp_hat = as.numeric(train_pred_lasso),
  resid       = LifeExp_obs - LifeExp_hat
) %>%
  summarise(
    n    = n(),
    rmse = sqrt(mean(resid^2)),
    mae  = mean(abs(resid)),
    rsq  = 1 - sum(resid^2) / sum((LifeExp_obs - mean(LifeExp_obs))^2)
  )

lasso_train_metrics
```

```
## # A tibble: 1 x 4
##       n  rmse   mae   rsq
##   <int> <dbl> <dbl> <dbl>
## 1    38 0.783 0.599 0.682
```

```
par(mfrow = c(1, 2))

plot(as.numeric(train_pred_lasso),
     y_train - as.numeric(train_pred_lasso),
     xlab = "Fitted values (lasso)",
     ylab = "Residuals",
     main = "Lasso Residuals vs Fitted")
abline(h = 0, lty = 2)

qqnorm(y_train - as.numeric(train_pred_lasso),
       main = "Lasso Residuals: Normal Q-Q")
qqline(y_train - as.numeric(train_pred_lasso))
```






```
par(mfrow = c(1, 1))
```

Commentary (1d).

What you should discuss for the homework:

- List which coefficients are non-zero at `lambda.min` (which predictors lasso keeps).
- Report the training RMSE, MAE, and R^2.
- Briefly compare lasso's training performance and sparsity to the best subset and tree models.

Conclusion: Cross-validated lasso selects $\lambda_{\min} \approx 0.175$. At this value, the lasso model retains only two predictors with non-zero coefficients: `Murder` and `HSGrad`, in addition to the intercept. In particular, higher murder rates are associated with lower life expectancy, while higher high-school graduation rates are associated with higher life expectancy; all other predictors are shrunk exactly to zero at $\lambda_{\min}$.

On the training data, the lasso model has RMSE ≈ 0.783, MAE ≈ 0.599, and $R^2 \approx 0.682$. This is somewhat worse than the best-subset linear model (which has RMSE ≈ 0.710 and $R^2 \approx 0.739$) and also slightly worse than the regression tree (RMSE ≈ 0.731, $R^2 \approx 0.723$). The residual plots for lasso look similar to those for the best-subset model: residuals are roughly centered around zero with no strong structure, and the Q–Q plot is close to a straight line. Lasso produces a very sparse model using only two predictors, but it does so at the cost of noticeably higher training error compared with the other models.

1(e) Compare models on training data (10 pts)

```
train_compare <- bind_rows(
  train_metrics_best  %>% mutate(model = "Best subset (lm)"),
  tree_train_metrics  %>% mutate(model = "Regression tree"),
  lasso_train_metrics %>% mutate(model = "Lasso")
) %>%
  dplyr::select(model, n, rmse, mae, rsq)

train_compare
```

```
## # A tibble: 3 x 5
##   model                n  rmse   mae   rsq
##   <chr>            <int> <dbl> <dbl> <dbl>
## 1 Best subset (lm)    38 0.710 0.574 0.739
## 2 Regression tree     38 0.731 0.550 0.723
## 3 Lasso               38 0.783 0.599 0.682
```

Commentary (1e).

What you should discuss for the homework:

- Compare the three models in terms of training RMSE, MAE, and R^2 using the table above.
- Comment on which model appears to fit the training data best and which is most parsimonious.
- Discuss any tradeoffs between fit and model complexity (e.g., tree vs linear models).

Conclusion: The training comparison table shows that the best-subset linear model has the strongest in-sample fit in terms of RMSE and R^2, with RMSE ≈ 0.710, MAE ≈ 0.574, and $R^2 \approx 0.739$. The regression tree is close behind (RMSE ≈ 0.731, MAE ≈ 0.550, $R^2 \approx 0.723$), while the lasso model has the weakest training fit (RMSE ≈ 0.783, MAE ≈ 0.599, $R^2 \approx 0.682$). All three models fit the training data reasonably well, but lasso clearly sacrifices some in-sample accuracy.

In terms of parsimony and complexity, the lasso model is actually the sparsest, using only `Murder` and `HSGrad` with non-zero coefficients. The best-subset model is also fairly simple, using three predictors (`Murder`, `HSGrad`, and `Frost`) in a linear form. The tree is most complex structurally, with multiple nodes and a piecewise-constant fit, even though each individual split is easy to interpret. Thus there is a tradeoff: best subset and the tree provide better training fit, while lasso yields the simplest coefficient pattern but with higher error. Which model is preferred will depend on how much we value parsimony relative to in-sample accuracy.

Use the test data set for the questions below

1(f) Best subset model on test data (10 pts)

```
test_pred_best <- predict(lm_best, newdata = states_test)

best_test_metrics <- tibble(
  LifeExp_obs = states_test$LifeExp,
  LifeExp_hat = test_pred_best,
  resid       = LifeExp_obs - LifeExp_hat
) %>%
  summarise(
    n    = n(),
    rmse = sqrt(mean(resid^2)),
    mae  = mean(abs(resid)),
    rsq  = 1 - sum(resid^2) / sum((LifeExp_obs - mean(LifeExp_obs))^2)
  )

best_test_metrics
```

```
## # A tibble: 1 x 4
##       n  rmse   mae   rsq
##   <int> <dbl> <dbl> <dbl>
## 1    12 0.742 0.686 0.562
```

Commentary (1f).

What you should discuss for the homework:

- Compare the test RMSE and R^2 to the training values for the best subset model.
- Comment on whether the performance seems stable (similar train vs test) or whether there is evidence of overfitting.

Conclusion: For the best-subset model, the test-set metrics are RMSE ≈ 0.742, MAE ≈ 0.686, and $R^2 \approx 0.562$ on the 12 hold-out states. On the training data, the same model had RMSE ≈ 0.710, MAE ≈ 0.574, and $R^2 \approx 0.739$. The test RMSE is modestly larger than the training RMSE and the test R^2 is somewhat lower, which is expected when we move from training to new data. The performance appears reasonably stable between train and test, with some loss of fit but no dramatic evidence of severe overfitting.

1(g) Regression tree on test data (10 pts)

Assess the regression tree model performance on the test data set. Is the model performance stable?

```
test_pred_tree <- predict(tree_pruned, newdata = states_test)

tree_test_metrics <- tibble(
  LifeExp_obs = states_test$LifeExp,
```

```
  LifeExp_hat = test_pred_tree,
  resid       = LifeExp_obs - LifeExp_hat
) %>%
  summarise(
    n    = n(),
    rmse = sqrt(mean(resid^2)),
    mae  = mean(abs(resid)),
    rsq  = 1 - sum(resid^2) / sum((LifeExp_obs - mean(LifeExp_obs))^2)
  )

tree_test_metrics
```

```
## # A tibble: 1 x 4
##       n  rmse   mae   rsq
##   <int> <dbl> <dbl> <dbl>
## 1    12  1.05 0.844 0.118
```

Commentary (1g).

What you should discuss for the homework:

- Compare the tree's test metrics to its training metrics (RMSE, MAE, R^2).
- Discuss whether the tree appears to overfit more or less than the best subset model.
- Comment on stability of performance and any notable differences.

Conclusion: For the regression tree, the test-set metrics are RMSE ≈ 1.05, MAE ≈ 0.844, and $R^2 \approx 0.118$ on the 12 test states. In contrast, on the training set the tree achieved RMSE ≈ 0.731, MAE ≈ 0.550, and $R^2 \approx 0.723$. Thus, RMSE increases substantially and MAE increases as well when we move from the training data to the test data, while R^2 drops from about 0.72 to roughly 0.12. This sizable deterioration in out-of-sample performance indicates that the tree is overfitting the training data: it captures patterns that do not generalize well to new states. Compared with the best-subset model, the tree is clearly less stable across train and test samples.

1(h) Lasso on test data (10 pts)

Assess the lasso regression model performance on the test data set. Is the model performance stable?

```
test_pred_lasso <- predict(lasso_model, newx = x_test)

lasso_test_metrics <- tibble(
  LifeExp_obs = y_test,
  LifeExp_hat = as.numeric(test_pred_lasso),
  resid       = LifeExp_obs - LifeExp_hat
) %>%
  summarise(
    n    = n(),
    rmse = sqrt(mean(resid^2)),
    mae  = mean(abs(resid)),
    rsq  = 1 - sum(resid^2) / sum((LifeExp_obs - mean(LifeExp_obs))^2)
  )

lasso_test_metrics
```

```
## # A tibble: 1 x 4
##       n  rmse   mae   rsq
##   <int> <dbl> <dbl> <dbl>
## 1    12 0.797 0.737 0.494
```

Commentary (1h).

What you should discuss for the homework:

- Compare the lasso test metrics to its training metrics.
- Comment on whether lasso seems to generalize well (similar train vs test) or shows signs of overfitting/underfitting.

- Briefly compare its test performance to the other models.

Conclusion: For the lasso model, the test-set metrics are RMSE ≈ 0.797, MAE ≈ 0.737, and $R^2 \approx 0.494$. On the training data, lasso had RMSE ≈ 0.783, MAE ≈ 0.599, and $R^2 \approx 0.682$. The test RMSE is slightly larger and the test R^2 is noticeably lower than the training values, indicating some loss of fit on new data, but the changes are moderate rather than extreme. This suggests that lasso generalizes reasonably well, though it does suffer more degradation than the best-subset model and much less than the regression tree. Relative to the other models, lasso's test performance is clearly better than the tree's but somewhat weaker than the best-subset model, which still has a lower test RMSE and higher test R^2.

1(i) Choose a model based on test performance (10 pts)

```
test_compare <- bind_rows(
  best_test_metrics  %>% mutate(model = "Best subset (lm)"),
  tree_test_metrics  %>% mutate(model = "Regression tree"),
  lasso_test_metrics %>% mutate(model = "Lasso")
) %>%
  dplyr::select(model, n, rmse, mae, rsq)

test_compare
```

```
## # A tibble: 3 x 5
##   model                n  rmse   mae   rsq
##   <chr>            <int> <dbl> <dbl> <dbl>
## 1 Best subset (lm)    12 0.742 0.686 0.562
## 2 Regression tree     12 1.05  0.844 0.118
## 3 Lasso               12 0.797 0.737 0.494
```

Commentary (1i).

What you should discuss for the homework:

- Compare the three models on the test set using RMSE, MAE, and R^2.
- Discuss the tradeoff between predictive accuracy, model interpretability, and complexity.
- State clearly which model you would choose **based on test performance** and explain your reasoning.

Conclusion: On the test set, the best-subset linear model has the strongest predictive performance: RMSE ≈ 0.742, MAE ≈ 0.686, and $R^2 \approx 0.562$. The lasso model is a clear second, with RMSE ≈ 0.797, MAE ≈ 0.737, and $R^2 \approx 0.494$, while the regression tree performs worst out of sample, with RMSE ≈ 1.05, MAE ≈ 0.844, and $R^2 \approx 0.118$.

Taking into account both predictive accuracy and model complexity, I would choose the best-subset linear model as the final model for predicting life expectancy. It achieves the lowest test RMSE and highest test R^2 while using only three interpretable predictors and showing relatively stable performance between training and test sets. Lasso is an attractive alternative that produces an extremely sparse model with only two predictors and reasonably good test performance, but it does not outperform the best-subset model. The regression tree, although interpretable in terms of threshold rules, appears to overfit the training data and has noticeably poorer accuracy on the test states.