Group 13 - Final Project Code Documentation

Jessica Flynn, Shan Jiang, Quinton Neville, Jieqi Tu

12/16/2018

1. Data Cleaning and Exploration

```
# Import data
cancer_raw = readr::read_csv("./Data/Cancer_Registry.csv") %>%
  janitor::clean_names()

#Check Dim
dim(cancer_raw)
```

```
## [1] 3047   34
```

```
# Check the percentage of NA values for each column
percentage_NA <- sapply(cancer_raw[1:ncol(cancer_raw)], function(x) sum(length(which(is.na(x)))) / nrow
print("Percentage of Missing Variable Data:")
```

```
## [1] "Percentage of Missing Variable Data:"
```

```
percentage_NA[which(percentage_NA != 0)] %>% t()
```

```
##      pct_some_col18_24 pct_employed16_over pct_private_coverage_alone
## [1,]          0.749918          0.04988513                  0.1998687
```

```
#Pulling quartiles for study_per_cap categorical manipulation
study.quart <- with(cancer_raw, study_per_cap[study_per_cap > 0]) %>%
  quantile(., probs = c(0.25, 0.5, 0.75))

#Pull State Region for matching from base R data()
data(state)
state.df <- cbind(state.name, state.region) %>% as.tibble() %>% rename(state = state.name, region = sta

#Clean, Manipulate, Tidy
cancer.df <- cancer_raw %>%    #Remove Rows with > 20% missing
  dplyr::select(-pct_some_col18_24) %>%  #Remove for too many missing
  mutate(
    pct_non_white = pct_black + pct_asian + pct_other_race, #Creating white, non-white percentages vari
    state = str_split_fixed(geography, ", ", 2)[ ,2] %>% as.factor(), #pulling state variable and casti
    binned_inc_lb = str_split_fixed(binned_inc, ", ", 2)[ ,1] %>% parse_number(), #pulling numeric lowe
    binned_inc_ub = str_split_fixed(binned_inc, ", ", 2)[ ,2] %>% parse_number(), #pulling numeric uppe
    binned_inc_point = (binned_inc_lb + binned_inc_ub)/2, #computing point estimate from ub,lb (interva
    study_quantile = ifelse(study_per_cap == 0, "None",
                            ifelse(study_per_cap > 0 & study_per_cap <= study.quart[1], "Low",
                                   ifelse(study_per_cap > study.quart[1] & study_per_cap <= study.quart[
                                          ifelse(study_per_cap > study.quart[2] & study_per_cap <= study
                                                 "Very High")))),
    study_quantile = as.factor(study_quantile) %>% fct_relevel(., "None", "Low", "Moderate", "High", "V
    avg_deaths_yr_pop = avg_deaths_per_year/pop_est2015,  #incorporate two vars into one (multicollinea
    avg_ann_count_pop = avg_ann_count/pop_est2015 #incorporate two vars into one (multicollinearity)
  ) %>%
```

```
  left_join(., state.df) %>%
  mutate(region = ifelse(is.na(region), 2, region),
         region = as.factor(region)) %>%
  dplyr::select(-c(binned_inc, geography, state, study_per_cap))

#Variable Selection
cancer.df <- cancer.df %>%
  dplyr::select(pct_white, pct_hs25_over, pct_employed16_over, pct_private_coverage, pct_public_coverag
  mutate(median_age_all_gender = (median_age_male + median_age_female)/2,
         south = ifelse(region == "2", TRUE, FALSE)) %>%
  dplyr::select(-median_age_female, -median_age_male, -region)
```

Exploratory Data Analysis

Below is a summary table of all continuous variables, including the response.

	target_death_rate	pct_white	pct_hs25_over	pct_employed16_over	pct_private_coverage	pct_publ
Min	59.7000000	10.1991551	7.5000000	17.6000000	22.3000000	
1st_Q	161.3000000	77.2111471	30.5000000	48.6000000	57.1500000	
Median	178.1000000	90.0288804	35.3000000	54.5000000	65.0000000	
Mean	178.6389637	83.6106781	34.8162003	54.1526425	64.3052159	
3rd_Q	195.2000000	95.3648343	39.6000000	60.3000000	72.0000000	
Max	293.9000000	100.0000000	54.8000000	80.1000000	92.3000000	
SD	27.4649908	16.3529187	7.0171148	8.3150644	10.6458406	
CV	0.1537458	0.1955841	0.2015474	0.1535486	0.1655517	

Below is a count of the categories of the per capita number of cancer-related clinical trials per county.

study_quantile	Count
None	1931
Low	279
Moderate	279
High	279
Very High	279

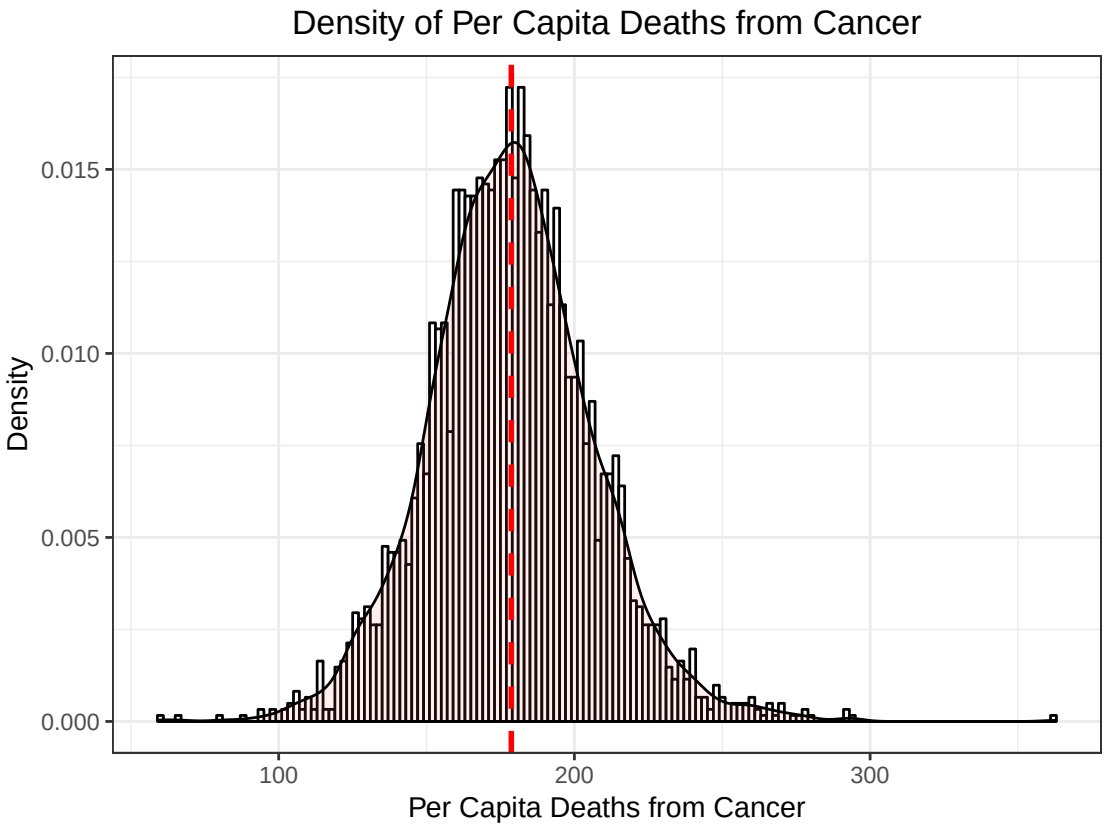
Below is the number and percentage of observations from the Southern United States and elsewhere.

south	Count	Percentage
FALSE	1664	0.5747841
TRUE	1383	0.4777202

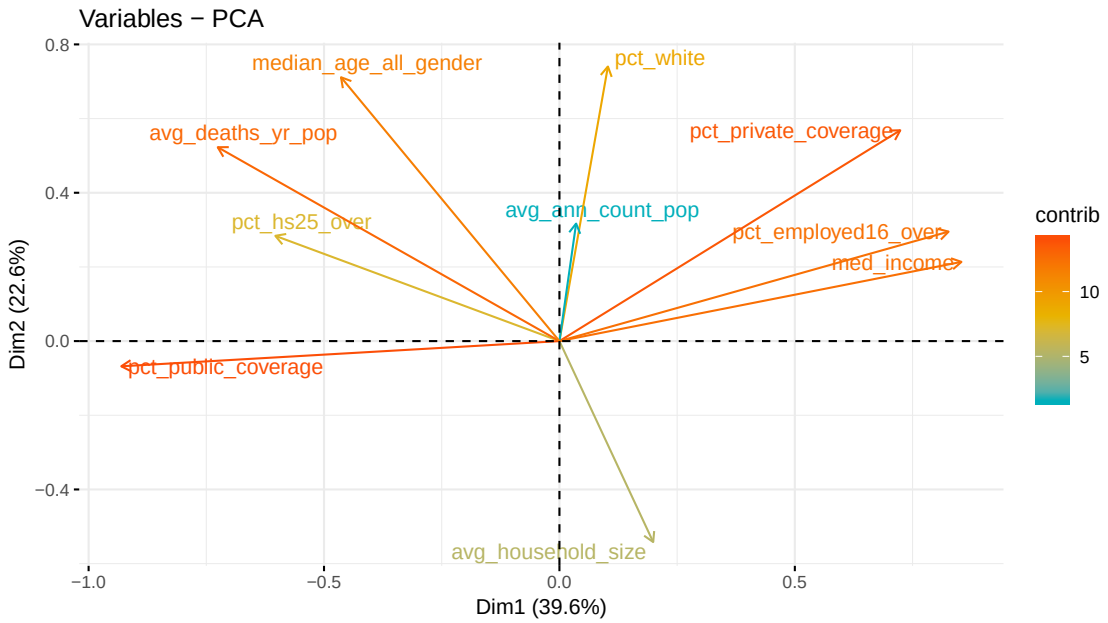
Below is the distribution of the per capita death counts from cancer by county, as it is roughly symmetric and approximately normal, this implies linear modeling may be an appropriate approach.

```
cancer_raw %>%
  ggplot(aes(x = target_death_rate)) +
    geom_histogram(aes(y = ..density..),
                   binwidth = 2, colour = "black", fill = "white") +
```

```
  geom_density(alpha = .1, fill = "#FF6666") +
  geom_vline(aes(xintercept = mean(target_death_rate, na.rm = T)),   # Ignore NA values for mean
             color = "red", linetype = "dashed", size = 1) + # Overlay with transparent density plot
labs(
  x = "Per Capita Deaths from Cancer",
  y = "Density",
  title = "Density of Per Capita Deaths from Cancer"
)
```



Below we have a projection of the predictor space onto the space spanned by the first two principal component vectors (describing the highest percentage of variability in the data).



Here we see that the variables we selected are generally of high magnitude and non-overlapping direction, implying that our covariates are explaining both different and important types of variance. This implies that we have chosen a relatively good set of predictors on which to build a linear model.

Inital Modeling Exploration

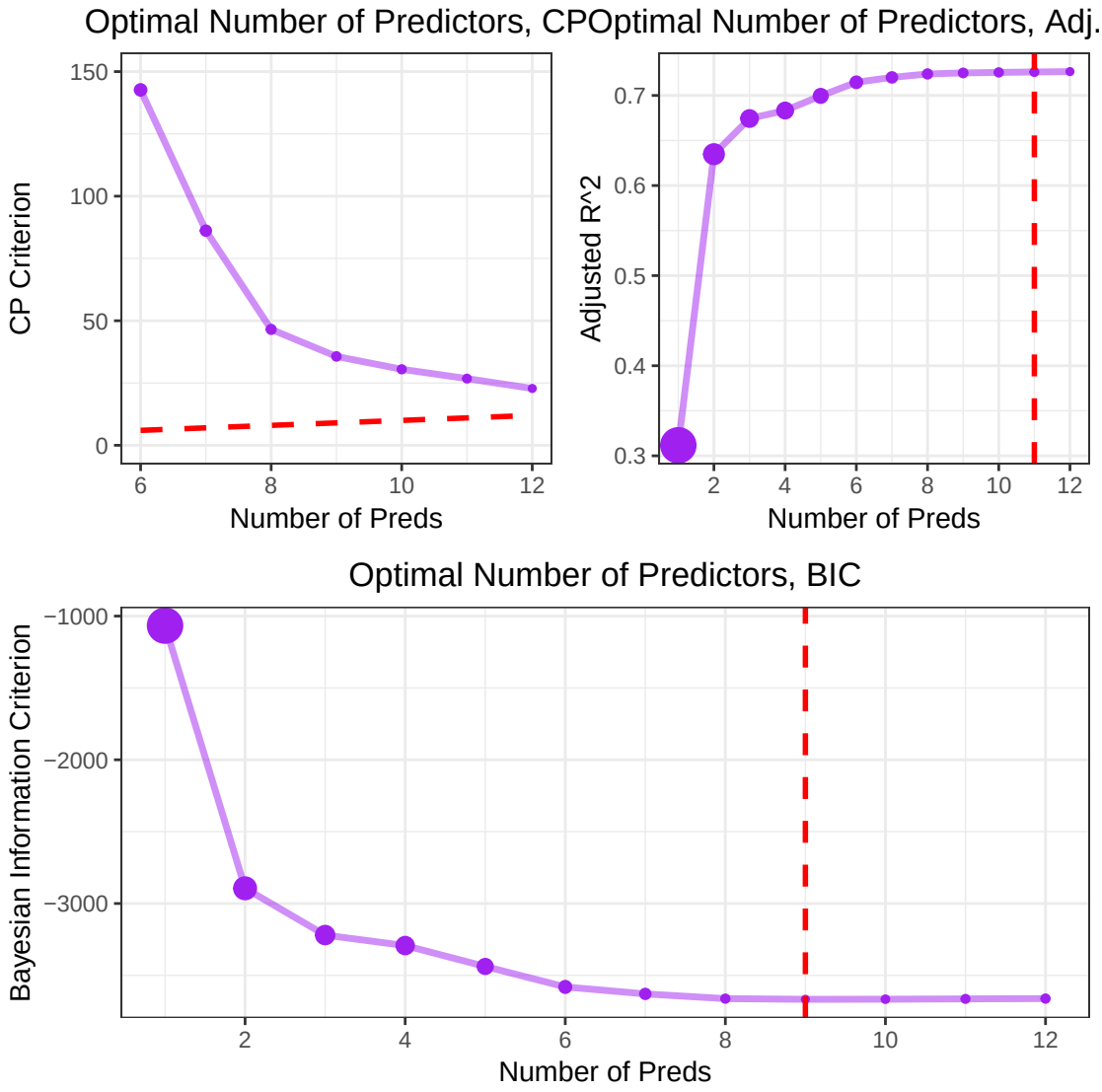
Visualize the approximately optimal number of predictors for a linear model (CP, adjR^2, BIC).

```
#Summary of models for each size (one model per size)
#Set max number of predictors
nvmax <- ncol(cancer.df) - 1
reg.subsets <- cancer.df %>%
  regsubsets(target_death_rate ~ ., data = ., nvmax = 12)
rs <- summary(reg.subsets)

reg.subsets$xnames
```

```
##  [1] "(Intercept)"             "pct_white"
##  [3] "pct_hs25_over"           "pct_employed16_over"
##  [5] "pct_private_coverage"    "pct_public_coverage"
##  [7] "avg_ann_count_pop"       "avg_household_size"
##  [9] "avg_deaths_yr_pop"       "med_income"
## [11] "study_quantileLow"       "study_quantileModerate"
## [13] "study_quantileHigh"      "study_quantileVery High"
## [15] "median_age_all_gender"   "southTRUE"
```

```
# Plots of Cp and Adj-R2 as functions of parameters
r.df <- tibble(
  preds = 1:nvmax,
  cp = rs$cp,
  adjr2 = rs$adjr2,
  bic = rs$bic,
  step = 1:nvmax
)
```

```r
#Grab max r^2 pred number
max.r2 <- with(r.df, which.max(adjr2))

#Grab min bic pred number
min.bic <- with(r.df, which.min(bic))

cp.plot <- r.df %>% ggplot(aes(x = preds, y = cp, size = cp)) +
  geom_point(colour = "purple") +
  geom_line(alpha = 0.5, colour = "purple", size = 1.25) +
 # geom_point(aes(x = preds, step),color = "black",size = 0.5) +
  geom_line(aes(x = preds, step), size = 1, linetype = 2, color = "red") +
  labs(
    x = "Number of Preds",
    y = "CP Criterion",
    title = "Optimal Number of Predictors, CP"
  ) + theme(legend.position = "none") +
  #scale_x_continuous(breaks = seq(0, 12, by = 2))
  xlim(c(6, 12)) + ylim(c(0, 150))

adjr2.plot <- r.df %>% ggplot(aes(x = preds, y = adjr2, size = 1 - adjr2)) +
  geom_point(colour = "purple") +
  geom_line(alpha = 0.5, colour = "purple", size = 1.25) +
  geom_vline(xintercept = 11, size = 1, linetype = 2, color = "red") +
  labs(
    x = "Number of Preds",
    y = "Adjusted R^2",
    title = "Optimal Number of Predictors, Adj.R^2"
  ) + theme(legend.position = "none") +
  scale_x_continuous(breaks = seq(0, 12, by = 2))

bic.plot <- r.df %>% ggplot(aes(x = preds, y = bic, size = bic)) +
  geom_point(colour = "purple") +
  geom_line(alpha = 0.5, colour = "purple", size = 1.25) +
  geom_vline(xintercept = min.bic, size = 1, linetype = 2, color = "red") +
  labs(
    x = "Number of Preds",
    y = "Bayesian Information Criterion",
    title = "Optimal Number of Predictors, BIC"
  ) + theme(legend.position = "none") +
  scale_x_continuous(breaks = seq(0, 12, by = 2))

(cp.plot + adjr2.plot) / bic.plot
```



Based on the plots above, CP criterion in the upper left with $p \leq CP$ constraint in red, that we reduced our original predictor too much to meet $p < Cp$ optimization. With respect to adjusted R^2, it appears as though we reach a converging maximum starting around 10 and negligible increase after, where additional predictors have diminishing marginal return. Lastly, BIC is more conservative (penalizing more strongly for more predictors) and meets a minimum at 9 predictors.

Feature Selection Lasso

Inputs are data (cancer.df), response (default target_death_rate), lambda (penalty, default = 1, higher penalty removes more coefficients), print(logical whether or not to print the results).

```
lasso.feature.removal <- function(data = cancer.df, response = "target_death_rate", lambda = 1, print =

  #Snag the index for matrices
  index <- which(names(data) == response)

  #Response
  Y <- as.matrix(data[, index])
```

```
  #Design matrix
  X <- data[ ,-index] %>%
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(.,data)
  X <- X[,-1]
  #Fit Model
  mod.lasso <- glmnet(X,Y,alpha = 1,intercept = T,lambda = lambda, family = "gaussian")
  #Print if true
  if (isTRUE(print)) {
  coef.lasso   <- coef(mod.lasso)[,1]
  remove.index <- which(coef.lasso == 0)
  keep.index   <- which(coef.lasso != 0)

    print(sprintf("Lambda = %f : Lasso method removed %i variables:", lambda, length(remove.index)))
    print(paste(names(remove.index)))
    print(sprintf("Lambda = %f : Lasso method selected %i variables:",lambda, length(keep.index)))
    print(paste(names(keep.index)))
    cat("\n")
  }
}

#Example Function for Feature Selection
map(.x = 0.2, ~lasso.feature.removal(na.omit(cancer.df), lambda = .x))
```

```
## [1] "Lambda = 0.200000 : Lasso method removed 2 variables:"
## [1] "pct_public_coverage" "avg_household_size"
## [1] "Lambda = 0.200000 : Lasso method selected 14 variables:"
##  [1] "(Intercept)"             "pct_white"
##  [3] "pct_hs25_over"           "pct_employed16_over"
##  [5] "pct_private_coverage"    "avg_ann_count_pop"
##  [7] "avg_deaths_yr_pop"       "med_income"
##  [9] "study_quantileLow"       "study_quantileModerate"
## [11] "study_quantileHigh"      "study_quantileVery High"
## [13] "median_age_all_gender"   "southTRUE"
```

```
## [[1]]
## NULL
```

2. Linear Model Selection

First we scale the predictor space.

```
cancer.df <- cancer.df %>%
  dplyr::select(target_death_rate, study_quantile, south, everything()) %>%
  na.omit(cancer.df)
scale.df <- cancer.df %>% dplyr::select(pct_white:median_age_all_gender) %>% scale() %>% as.tibble()
cancer.df <- bind_cols(cancer.df %>% dplyr::select(-c(pct_white:median_age_all_gender)), scale.df)
```

Then we write the machinery for K-fold cross validation in subset selecion (MSE, AIC, BIC criterion).

```r
sampleSize <- nrow(cancer.df)

mseCV <- function(data.df, kfolds = 10){
  folds <- sample(1:kfolds, sampleSize, rep = T)
  mse <- rep(0, kfolds)
  for (k in 1:kfolds) {
    train.df <- data.df[folds != k,]
    test.df <- data.df[folds == k,]

    lm.mod <- lm(target_death_rate ~ ., data = train.df)
    preds <- predict(lm.mod, newdata = test.df)
    mse[k] <- with(test.df, mean((target_death_rate - preds)^2))
  }
  mean(mse)
}

aicCV <- function(data.df, kfolds = 10){
  folds <- sample(1:kfolds, sampleSize, rep = T)
  aic <- rep(0, kfolds)
  for (k in 1:kfolds) {
    train.df <- data.df[folds != k,]

    lm.mod <- lm(target_death_rate ~ ., data = train.df)
    aic[k] <- AIC(lm.mod)
  }
  mean(aic)
}

bicCV <- function(data.df, kfolds = 10){
  folds <- sample(1:kfolds, sampleSize, rep = T)
  bic <- rep(0, kfolds)
  for (k in 1:kfolds) {
    train.df <- data.df[folds != k,]

    lm.mod <- lm(target_death_rate ~ ., data = train.df)
    bic[k] <- BIC(lm.mod)
  }
  mean(bic)
}
```

a. Backward Selection (p-value)

```r
mult.fit = lm(target_death_rate ~ ., data = cancer.clean)
summary(mult.fit)

# Remove the variable with the highest p-value: avg_household_size
step1 = update(mult.fit, . ~ . -avg_household_size)
summary(step1)

# All variables are significant with p-value less than 0.05

# store the result of backward procedure
```

```
model_backward = step1
```

```
## [1] "Variables Selected:"
```

```
## list(target_death_rate, study_quantile, south, pct_white, pct_hs25_over,
##     pct_employed16_over, pct_private_coverage, pct_public_coverage,
##     avg_ann_count_pop, avg_deaths_yr_pop, med_income, median_age_all_gender)
```

b. Forward Selection (p-value)

```
# check p-value for each predictor in SLR model
fit1 = lm(target_death_rate ~ pct_white, data = cancer.clean)
tidy(fit1)
fit2 = lm(target_death_rate ~ pct_hs25_over, data = cancer.clean)
tidy(fit2)
fit3 = lm(target_death_rate ~ pct_employed16_over, data = cancer.clean)
tidy(fit3)
fit4 = lm(target_death_rate ~ pct_private_coverage, data = cancer.clean)
tidy(fit4)
fit5 = lm(target_death_rate ~ pct_public_coverage, data = cancer.clean)
tidy(fit5)
fit6 = lm(target_death_rate ~ avg_ann_count_pop, data = cancer.clean)
tidy(fit6)
fit7 = lm(target_death_rate ~ avg_deaths_yr_pop, data = cancer.clean)
tidy(fit7)
fit8 = lm(target_death_rate ~ med_income, data = cancer.clean)
tidy(fit8)
fit9 = lm(target_death_rate ~ study_quantile, data = cancer.clean)
tidy(fit9)
fit10 = lm(target_death_rate ~ median_age_all_gender, data = cancer.clean)
tidy(fit10)
fit11 = lm(target_death_rate ~ avg_household_size, data = cancer.clean)
tidy(fit11)
fit12 = lm(target_death_rate ~ south, data = cancer.clean)
tidy(fit12)

# start with one predictor that has the lowest p-value: avg_deaths_yr_pop
step1 = lm(target_death_rate ~ avg_deaths_yr_pop, data = cancer.clean)
tidy(step1)

# enter the one with the lowest p-value in the rest
step2 = update(step1, .~. + med_income)
tidy(step2)
step3 = update(step2, .~. + pct_hs25_over)
tidy(step3)
step4 = update(step3, .~. + pct_public_coverage)
tidy(step4)
step5 = update(step4, .~. + pct_employed16_over)
tidy(step5)
step6 = update(step5, .~. + pct_private_coverage)
tidy(step6)
step7 = update(step6, .~. + south)
tidy(step7)
```

```
step8 = update(step7, .~. + pct_white)
tidy(step8)
step9 = update(step8, .~. + study_quantile)
tidy(step9)
step10 = update(step9, .~. + avg_ann_count_pop)
tidy(step10)
step11 = update(step10, .~. + avg_household_size)
tidy(step11)
step12 = update(step11, .~. + median_age_all_gender)
tidy(step12)

# store the result of forward procedure
model_forward = step12
```

```
## [1] "Variables Selected:"
```

```
## list(target_death_rate, avg_deaths_yr_pop, med_income, pct_hs25_over,
##     pct_public_coverage, pct_employed16_over, pct_private_coverage,
##     south, pct_white, study_quantile, avg_ann_count_pop, avg_household_size,
##     median_age_all_gender)
```

c. Stepwise Backwards (Leaps, AIC)

```
# use stepwise function to choose the best subsets
step(mult.fit, direction = "backward")

# store the result of stepwise procedure
model_stepwise = lm(target_death_rate ~ pct_white + pct_hs25_over +
    pct_employed16_over + pct_private_coverage + pct_public_coverage +
    avg_ann_count_pop + avg_deaths_yr_pop + med_income + study_quantile +
    median_age_all_gender + south, data = cancer.clean)
```

```
## [1] "Variables Selected:"
```

```
## list(target_death_rate, pct_white, pct_hs25_over, pct_employed16_over,
##     pct_private_coverage, pct_public_coverage, avg_ann_count_pop,
##     avg_deaths_yr_pop, med_income, study_quantile, median_age_all_gender,
##     south)
```

d. Forward Subset Selection with k-Fold CV (MSE, AIC, BIC criterion)

Function to run the subset algorithm, it will output the variable selection process so you visualize how it works. Comment out the set seed to get true variability in the subsets, only leave it in for reproducibility.

- The function `f.subset.mse(sub.cancer.df, maxPreds = 10, nfolds = 5)` takes in a data frame, max number of predictors you want the algorithm to run through (can't be larger than the number of predictors in the data frame), and the number of folds for the cross validation.
- The way the algorithm works is

1. Starts with an empty (null) set of current predictors in the data frame input

2. Starts with a full set of available predictors

3. Loops through all available preds and adds one at a time, splits the data into kfolds test/train (CV), fits lm() with current preds, stores criterion value
4. Selects the predictor whose CV MSE was smallest
5. Adds 'best' predictor to current predictors, removes it from available predictors

6. Stores that predictor set as 'best' and records the indices and CV criterion in a matrix
7. Repeats 3.- 5. until you have met the maximum number of predictors you specified
8. Returns a list with all 'best' predictor sets at each iteration, matrix of results, and prespecified max preds
9. Last few lines of code output the optimal predictor set

notes - the print() lines within the function allow you to visualize how the algorithm is working, these are commented out to reduce clutter (avoids reams of output)

```
#Forward Subset#
#set.seed(4) #Comment this out if you want to really get a different subset each time

#Function
#inputs:
#full data frame (whatever that ends up being)
#maximum number you want to look at subsets up to (no larger than p)
#Number of folds for the cross validation step, 10 is a little slow but better for final paper
#criterion to minimize -  either "mse", or "aic". Must put those in just as they appear with "

f.subset <- function(sub.cancer.df, maxPreds = 10, nfolds = 5, criterion = "mse") {

sub.cancer.df <- cancer.df
#number of possible predictors
## Allthe predictors (their indices).
allPreds <- 1:ncol(sub.cancer.df)
allPreds <- allPreds[-1] #substract the response

#current set of preds (starts empty), and available
currPreds <- c()
availPreds <- setdiff(allPreds,currPreds)
#Record the min errors
minError <- c()
#Initalize pred list and mse result matrix (just for storage)
pred.list <- list()
result.mat <- matrix(nrow = ncol(sub.cancer.df) - 1, ncol = 2)
#Initalize iteration
i <- 1
#Forward selection loop
while (length(currPreds) < maxPreds) {
  ##add predictor which decreases MSE (as determined by CV or
  ##Bootstrapping)
  ## The MSE/AIC computed as we add each of the available predictors
  allError <- c()
  for (id in availPreds) {
    data.df <- sub.cancer.df[,c(currPreds,id,1)] #Only the preds we want in the df
    if (criterion == "mse") {
      error <- mseCV(data.df, nfolds)  #cross validate mse for the model with the added pred
    } else if (criterion == "aic") {
```

```
      error <- aicCV(data.df, nfolds) #same but with AIC
    } else if (criterion == "bic") {
      error <- bicCV(data.df, nfolds) #same bic
    } else {
      print("Invalid criterion")
      stop("Wrong criterion input")
    }
    allError <- c(allError,error)  #Collect all the errors for every variable added
  }
  ##Find the min
  id <- which.min(allError)
  ##get the best predictor and lowest MSE/AIC
  bestPred <- availPreds[id]
  bestError <- min(allError)
  ##Add these into the collection
  currPreds <- c(currPreds,bestPred)
  minError <- c(minError,bestError)
  ##Take out the pred you just added from consideration
  availPreds <- setdiff(allPreds,currPreds)
  ## Print stuff out for debugging and attention-grabbing
  #print(sprintf("Iteration: %i Predictor Added: %s %s Value: %s",i, names(sub.cancer.df[,bestPred]), c
  #print(currPreds)
  result.mat[i,] <- c(bestPred,bestError)         #You can also comment out all print() later, it's jus
  pred.list[[i]] <- currPreds
  i <- i + 1
    }
  return(list(pred.list = pred.list, result.mat = result.mat, maxPreds = maxPreds)) #returns list of pr
}

#Select k-Folds
k <- 10

#Run Subset, call function, output is a list with predictor sets, reslut matrix and maxPreds
f.mse.list <- f.subset(cancer.df, maxPreds = ncol(cancer.df) - 1, nfolds = k, criterion = "mse")
f.aic.list <- f.subset(cancer.df, maxPreds = ncol(cancer.df) - 1, nfolds = k, criterion = "aic")
f.bic.list <- f.subset(cancer.df, maxPreds = ncol(cancer.df) - 1, nfolds = k, criterion = "bic")

#Show the 'best' final selection with minimal MSE (takes in a list object from the previous function, w
present.fs.result <- function(f.result.list, criterion) {
lm.fs.preds <- with(f.result.list, pred.list[[which.min(result.mat[,2])]]) #Pick out indices from best
fs.mse <- with(f.result.list, result.mat[which.min(result.mat[,2]), 2])
print(sprintf("The best predictor set of %s predictors, out of a max of %s, (%s = %s)",
              length(lm.fs.preds), f.result.list$maxPreds, criterion, round(fs.mse, 3)))
print(names(cancer.df[,c(lm.fs.preds)]))
}

#Show the final results and subset selections, with CV criterion stats
present.fs.result(f.mse.list, "MSE")
```

```
## [1] "The best predictor set of 12 predictors, out of a max of 12, (MSE = 207.523)"
##  [1] "avg_deaths_yr_pop"     "median_age_all_gender"
##  [3] "south"                 "pct_hs25_over"
##  [5] "med_income"            "pct_employed16_over"
##  [7] "pct_private_coverage"  "pct_white"
```

```
##  [9] "avg_ann_count_pop"     "study_quantile"
## [11] "pct_public_coverage"   "avg_household_size"
```

```
present.fs.result(f.aic.list, "AIC")
```

```
## [1] "The best predictor set of 11 predictors, out of a max of 12, (AIC = 21286.212)"
##  [1] "avg_deaths_yr_pop"     "median_age_all_gender"
##  [3] "south"                 "pct_hs25_over"
##  [5] "med_income"            "pct_employed16_over"
##  [7] "pct_private_coverage"  "avg_ann_count_pop"
##  [9] "study_quantile"        "pct_white"
## [11] "pct_public_coverage"
```

```
present.fs.result(f.bic.list, "BIC")
```

```
## [1] "The best predictor set of 9 predictors, out of a max of 12, (BIC = 21369.508)"
## [1] "avg_deaths_yr_pop"     "median_age_all_gender" "south"
## [4] "pct_hs25_over"         "med_income"            "pct_employed16_over"
## [7] "pct_private_coverage"  "avg_ann_count_pop"     "pct_white"
```

```
#Pull the indices for the selected models, for later comparison
fs.mse.preds <- with(f.mse.list, pred.list[[which.min(result.mat[,2])]])
fs.aic.preds <- with(f.aic.list, pred.list[[which.min(result.mat[,2])]])
fs.bic.preds <- with(f.bic.list,  pred.list[[which.min(result.mat[,2])]])
```

e. Backwards CV Subset Selection

- The way the algorithm works is

1. Starts with an exhaustive set of current predictors in the data frame input (all preds in the data frame)
2. Starts with a empty set of predictors removed

3. Loops through all current preds and removes one at a time, splits the data into kfolds test/train (CV), fits lm() with current preds, stores criterion value
4. Selects the set whose CV criterion was smallest
5. The optimal set had one of the predictors removed, selects that predictor as 'worst' removes it from current predictors

6. Stores that predictor set as worst and records the indices and CV MSE in a matrix for the model without that pred
7. Repeats 3.- 5. until you have met the minimum number of predictors you specified
8. Returns a list with all 'best' predictor sets at each iteration, matrix of results, and prespecified max preds
9. Last few lines of code output the optimal predictor set

notes - the print() lines within the function are so you can visualize how the algorithm is working, they can be commented out to reduce clutter (avoids reams of output)

```
#Backward Selection
#set.seed(4444)  #Set seed for reproducibility
################################################

#Num all possible preds
maxPreds <- ncol(cancer.df - 1)
```

```r
#Function
#Inputs:
#full data set (whatever that ends up being, ideally non-singular)
#minimum number of preds you want the algorithm to reduce subsets to
#Number of folds for CV step
#Criterion to minimize: "mse" or "aic"

b.subset <- function(sub.cancer.df, minPreds = 1, nfolds = 5, criterion = "mse"){

## Allthe predictors (their indices).
allPreds <- 1:ncol(sub.cancer.df)
allPreds <- allPreds[-1] #take out response

#current set of preds (starts full), and available
currPreds <- allPreds
availPreds <- allPreds
#Record the min errors
minError <- c()
#Set up storage objects
pred.list <- list()
result.mat <- matrix(nrow = ncol(sub.cancer.df) - 1, ncol = 2)
#initialize iteration
i <- 1
#Forward selection loop
while (length(currPreds) >= minPreds) {
  ##remove predictor which has CV lm with lowest MSE/AIC (as determined by CV)
  ## The MSE/AIC computed as we remove of the available predictors
  allError <- c()
  for (id in availPreds) {
    data.df <- sub.cancer.df[,c(currPreds[-id],1)]  #Take data frame without the predictor were removin
    if (criterion == "mse") {
      error <- mseCV(data.df, nfolds)  #calculate CV mse
    } else if (criterion == "aic") {
      error <- aicCV(data.df, nfolds)  #calculate CV aic
    } else if (criterion == "bic") {
      error <- bicCV(data.df, nfolds)  # CV bic
    } else {
      print("Invalid criterion")
      stop("Wrong criterion input")
    }
    allError <- c(allError,error)  #Collect all our criterions (error is misleading, can be mse or aic)
  }
  ##Find the min
  id <- which.min(allError)
  ##get the worst predictor and MSE
  worstPred <- availPreds[id]
  bestError <- min(allError)
  ##Remove these from the collection
  currPreds <- currPreds[-id]
  minError <- c(minError, bestError)
  availPreds <- currPreds
  ## Print stuff out for debugging and attention-grabbing
  #print(sprintf("Predictor Removed: %s  %s Value: %s",names(sub.cancer.df[,worstPred]), criterion, bes
```

```
  #print(currPreds)
  result.mat[i,] <- c(worstPred,bestError)   #All print() can be commented out at any time, just to vis
  pred.list[[i]] <- currPreds

  i <- i + 1
  }
  list(pred.list = pred.list, result.mat = result.mat)
}

#Set Min Preds and k-Folds
min.preds <- 5
k <- 10

#Call functions, output is a list of preds and a result matrix of criterion and predictor removed
b.mse.list <- b.subset(cancer.df, minPreds = min.preds, nfolds = k, criterion = "mse")
b.aic.list <- b.subset(cancer.df, minPreds = min.preds, nfolds = k, criterion = "aic")
b.bic.list <- b.subset(cancer.df, minPreds = min.preds, nfolds = k, criterion = "bic")

#Visualize function takes in a list object from the last function
present.bs.result <- function(result.list, criterion) {
lm.bs.preds <- with(result.list, pred.list[[which.min(result.mat[,2])]])
lm.bs.mse <- with(result.list, result.mat[which.min(result.mat[,2]), 2])
print(sprintf("The best predictor set of %s predictors, out of a max of %s, (%s = %s)",
              length(lm.bs.preds), maxPreds, criterion, round(lm.bs.mse, 3)))
names(cancer.df[,c(lm.bs.preds)])
}

#Present the final subsets which were selected
present.bs.result(b.mse.list, "MSE")
```

```
## [1] "The best predictor set of 11 predictors, out of a max of 13, (MSE = 207.237)"
```

```
##  [1] "study_quantile"        "south"
##  [3] "pct_white"             "pct_hs25_over"
##  [5] "pct_employed16_over"   "pct_private_coverage"
##  [7] "pct_public_coverage"   "avg_household_size"
##  [9] "avg_deaths_yr_pop"     "med_income"
## [11] "median_age_all_gender"
```

```
present.bs.result(b.aic.list, "AIC")
```

```
## [1] "The best predictor set of 11 predictors, out of a max of 13, (AIC = 21286.373)"
```

```
##  [1] "study_quantile"        "south"
##  [3] "pct_white"             "pct_hs25_over"
##  [5] "pct_employed16_over"   "pct_private_coverage"
##  [7] "pct_public_coverage"   "avg_household_size"
##  [9] "avg_deaths_yr_pop"     "med_income"
## [11] "median_age_all_gender"
```

```
present.bs.result(b.bic.list, "BIC")
```

```
## [1] "The best predictor set of 11 predictors, out of a max of 13, (BIC = 21380.117)"
```

```
##  [1] "study_quantile"        "south"
##  [3] "pct_white"             "pct_hs25_over"
##  [5] "pct_employed16_over"   "pct_private_coverage"
```

```
##  [7] "pct_public_coverage"   "avg_household_size"
##  [9] "avg_deaths_yr_pop"     "med_income"
## [11] "median_age_all_gender"
```

```
#Pull the indices for the selected models, for later comparison
bs.mse.preds <- with(b.mse.list, pred.list[[which.min(result.mat[,2])]])
bs.aic.preds <- with(b.aic.list, pred.list[[which.min(result.mat[,2])]])
bs.bic.preds <- with(b.bic.list,  pred.list[[which.min(result.mat[,2])]])
```

In general, backwards selection selects larger models than forwards. We hypothesize forward selection will perform better.

f. Ridge and Lasso

Need to build functions for glmnet to be compatible with tidyverse, and for ease of use.

```
#Initialize Sample Size
sampleSize <- nrow(cancer.df)

#Initialize lambda grids
lambda.grid.lasso <- 10^seq(-3,0,length = 100)
lambda.grid.ridge <- 10^seq(-3,0,length = 100)


#Lambda optimization Ridge
ridge.opt <- function(data, grid = lambda.grid, response = "target_death_rate"){
    #Snag the index for the matrices
  index <- which(names(data) == response)

  #Response
  Y <- as.matrix(data[ ,index])

  #Design matrix
  X <- data[ ,-index] %>%
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(., data)
  X <- X[,-1]

  #Lambda optimize
  cv.ridge <- cv.glmnet(X,Y,alpha = 0,intercept = T, lambda = grid, family = "gaussian")
  cv.ridge$lambda.min
}

#lambda.opt.ridge <- ridge.opt(cancer.df, grid = lambda.grid)

#Lambda optimization Lasso
lasso.opt <- function(data, grid = lambda.grid, response = "target_death_rate"){
  #Snag the index for matrices
  index <- which(names(data) == response)

  #Response
```

```
  Y <- as.matrix(data[, index])

  #Design matrix
  X <- data[ ,-index] %>%
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(.,data)
  X <- X[,-1]

  #Optimize Lambda
  cv.lasso <- cv.glmnet(X,Y,alpha = 1,intercept = T,lambda = grid, family = "gaussian")
  cv.lasso$lambda.min
}

#lambda.opt.lasso <- lasso.opt(cancer.df, grid = lambda.grid.lasso)

ridge <- function(data, response = "target_death_rate") {

  #Snag the index for the matrices
  index <- which(names(data) == response)

  #Response
  Y <- as.matrix(data[ ,index])

  #Design matrix
  X <- data[ ,-index] %>%
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(., data)
  X <- X[,-1]

  #Lambda optimize
  cv.ridge <- cv.glmnet(X,Y,alpha = 0,intercept = T, lambda = lambda.grid.ridge, family = "gaussian")
  lambda.opt.ridge <- cv.ridge$lambda.min

  #Return model
  return(glmnet(X,Y,alpha = 0,intercept = T,lambda = lambda.opt.ridge, family = "gaussian"))
}

lasso <- function(data, response = "target_death_rate", print = FALSE) {

  #Snag the index for matrices
  index <- which(names(data) == response)

  #Response
  Y <- as.matrix(data[, index])

  #Design matrix
  X <- data[ ,-index] %>%
```

```
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(.,data)
  X <- X[,-1]

  #Optimize Lambda
  cv.lasso <- cv.glmnet(X,Y,alpha = 1,intercept = T,lambda = lambda.grid.lasso, family = "gaussian")
  lambda.opt.lasso <- cv.lasso$lambda.min
  mod.lasso <- glmnet(X,Y,alpha = 1,intercept = T,lambda = lambda.opt.lasso, family = "gaussian")
  #Print if true
  if (isTRUE(print)) {
  coef.lasso <- coef(mod.lasso)[,1]
    print("The Lasso method selected the variables:")
    print(paste(names(which(coef.lasso != 0))))
  }
  return(mod.lasso)
}

glmnet.predict <- function(model, data, response = "target_death_rate") {
  #Snag the index for matrices
  index <- which(names(data) == response)
  #Design matrix
  X1 <- data[ ,-index] %>%
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(., data)
  X1 <- X1[,-1]
  return(predict(model, newx = X1, type = "response"))
}

glmnet.rmse <- function(model, data, response = "target_death_rate") {
  #Snag the index for matrices
  index <- which(names(data) == response)
  #Design matrix
  X1 <- data[ ,-index] %>%
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(., data)
  X1 <- X1[,-1]
  preds <- predict(model, newx = X1, type = "response")
  return(with(data, sqrt(mean((target_death_rate - preds)^2))))
}

glmnet.mse <- function(model, data, response = "target_death_rate") {
  #Snag the index for matrices
  index <- which(names(data) == response)
  #Design matrix
```

```
  X1 <- data[ ,-index] %>%
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(., data)
  X1 <- X1[,-1]
  preds <- predict(model, newx = X1, type = "response")
  return(with(data, mean((target_death_rate - preds)^2)))
}

lassoCV <- function(data.df = cancer.df, kfolds = 10, response = "target_death_rate"){
  index <- which(names(data.df) == response) #Grab response index for matrices
  folds <- sample(1:kfolds, sampleSize, rep = T) #Randomly sample indeices for kfolds
  error <- rep(0, kfolds) #initialize error vector
  for (k in 1:kfolds) {
    train.df <- data.df[folds != k,] #Split into test train
    test.df <- data.df[folds == k,]
    X1 <- test.df[ , -index] %>%
          names() %>%
          paste(., collapse = "+") %>%
          paste("~ ", .) %>%
          formula() %>%
          model.matrix(.,test.df)
    X1 <- X1[,-1]

    lasso.mod <- lasso(train.df, response)
       preds <- predict(lasso.mod, newx = X1, type = "response")
    error[k] <- with(test.df, mean((target_death_rate - preds)^2))
  }
  mean(error) #return kfold mean mse
}

ridgeCV <- function(data.df, kfolds = 10, response){
  index <- which(names(data.df) == response) #Grab response index for matrices
  folds <- sample(1:kfolds, sampleSize, rep = T) #Randomly sample indeices for kfolds
  error <- rep(0, kfolds) #initialize error vector
  for (k in 1:kfolds) {
    train.df <- data.df[folds != k,] #Split into test train
     test.df <- data.df[folds == k,]
    #Design test matrix
  X1 <- test.df[ , -index] %>%
  names() %>%
  paste(., collapse = "+") %>%
  paste("~ ", .) %>%
  formula() %>%
  model.matrix(.,test.df)
  #Remove Intercept
   X1 <- X1[ ,-1]
  #Fit model, predict on kth fold, record mse
   ridge.mod <- ridge(train.df, response)
       preds <- predict(ridge.mod, newx = X1, type = "response")
    error[k] <- with(test.df, mean((target_death_rate - preds)^2))
```

```
  }
  mean(error) #return kfold mean mse
}

#test - CV - all good
lassoCV(cancer.df, 10, "target_death_rate")
```

```
## [1] 207.7374
```

```
ridgeCV(cancer.df, 10, "target_death_rate")
```

```
## [1] 208.6434
```

3. Model Comparison

a. Iterative 10-Fold Cross Validation (all models)

```
#Number of CV iterations
nrep <- 300

#make sure target_death_rate is the first index in the data set (just way easier that way) District of
#State is a problem, need to make a region variable
set.seed(44)
cv.df <- cancer.df %>%
         crossv_mc(., n = nrep,  test = 0.1) %>%   #Create nrep 10-fold data frames, 10% ~ 10 fold
  mutate(train = map(train, as.tibble),
         test = map(test, as.tibble)) %>%                                                    #Fit Mode
  mutate(fs.mse.lm = map(train, ~lm(target_death_rate ~ ., data = .x[, c(fs.mse.preds, 1)])), #fs mse
         fs.aic.lm = map(train, ~lm(target_death_rate ~ ., data = .x[, c(fs.aic.preds, 1)])), #   aic
         fs.bic.lm = map(train, ~lm(target_death_rate ~ ., data = .x[, c(fs.bic.preds, 1)])), #   bic
         bs.mse.lm = map(train, ~lm(target_death_rate ~ ., data = .x[, c(bs.mse.preds, 1)])), #bs mse
         bs.aic.lm = map(train, ~lm(target_death_rate ~ ., data = .x[, c(bs.aic.preds, 1)])), #   aic
         bs.bic.lm = map(train, ~lm(target_death_rate ~ ., data = .x[, c(bs.bic.preds, 1)])), #   bic
         lasso.lm  = map(train, ~lasso(data = .x)), #lasso
         ridge.lm  = map(train, ~ridge(data = .x)), #ridge
         leaps.back  = map(train, ~lm(target_death_rate ~ pct_white + pct_hs25_over +  # Leap package b
          pct_employed16_over + pct_private_coverage + pct_public_coverage +
          avg_ann_count_pop + avg_deaths_yr_pop + med_income + study_quantile +
          median_age_all_gender + south, data = .x)),
         step.front  = map(train, ~lm(target_death_rate ~ pct_white + pct_hs25_over +  # F. step p-valu
          pct_employed16_over + pct_private_coverage + pct_public_coverage +
          avg_ann_count_pop + avg_deaths_yr_pop + med_income + study_quantile +
          median_age_all_gender + south + avg_household_size, data = .x))) %>%
  mutate(fs.mse.mse = map2_dbl(fs.mse.lm, test, ~rmse(model = .x, data = .y)),
         fs.aic.mse = map2_dbl(fs.aic.lm, test, ~rmse(model = .x, data = .y)),    #Calculate RMSE for e
         fs.bic.mse = map2_dbl(fs.bic.lm, test, ~rmse(model = .x, data = .y)),
         bs.mse.mse = map2_dbl(bs.mse.lm, test, ~rmse(model = .x, data = .y)),
         bs.aic.mse = map2_dbl(bs.aic.lm, test, ~rmse(model = .x, data = .y)),
         bs.bic.mse = map2_dbl(bs.bic.lm, test, ~rmse(model = .x, data = .y)),
         lasso.mse  = map2_dbl( lasso.lm, test, ~glmnet.rmse(model = .x, data = .y)),
         ridge.mse  = map2_dbl( lasso.lm, test, ~glmnet.rmse(model = .x, data = .y)),
         leaps.back.mse = map2_dbl(leaps.back, test, ~rmse(model = .x, data = .y)),
```

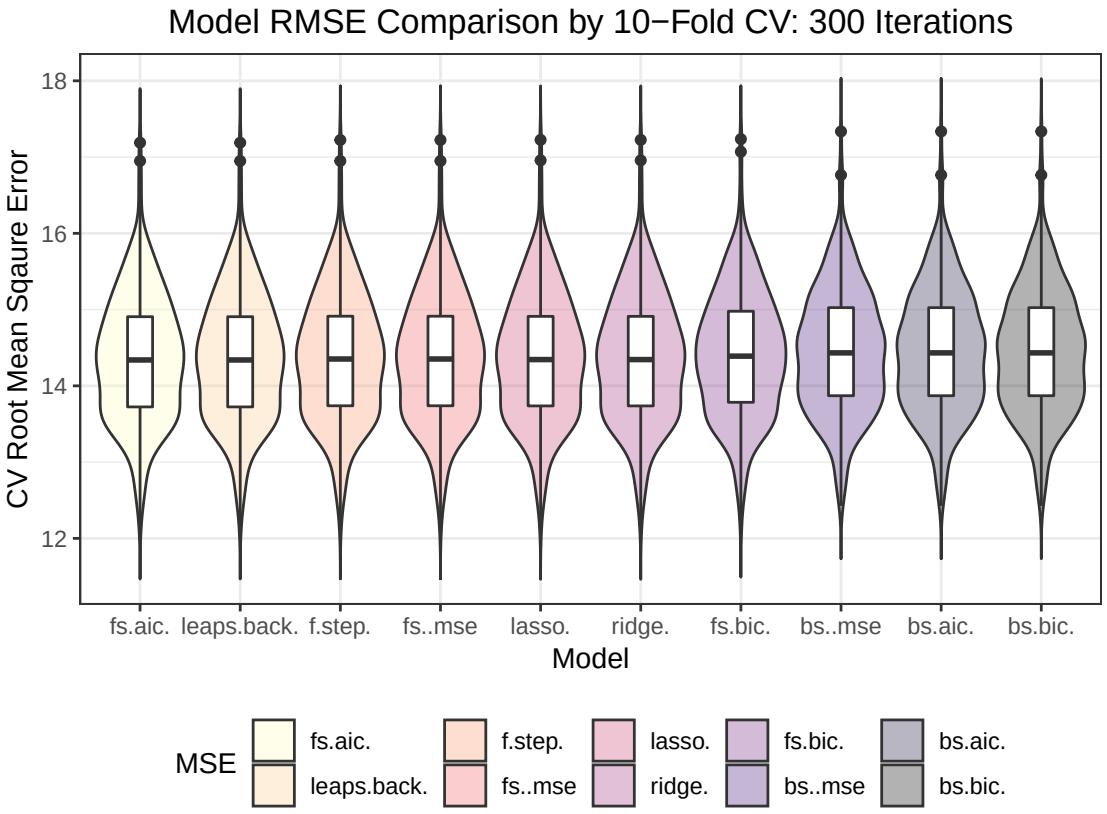
```
        f.step.mse = map2_dbl(step.front, test, ~rmse(model = .x, data = .y)))

table.result <- cv.df %>%
  dplyr::select(ends_with(".mse")) %>%
  gather(key = model, value = mse) %>%
  mutate(model = str_replace(model, ".mse", ""),
         model = fct_reorder(model, mse, .desc = FALSE, .fun = mean)) %>%
  group_by(model) %>%
  summarize(
    mean_rmse = mean(mse),
    median_rmse = median(mse),
    variance_rmse = sd(mse)^2,
    Q1 = quantile(mse, .25),
    Q3 = quantile(mse, .75)
  ) %>% ungroup()

table.result %>% knitr::kable(digits = 4)
```

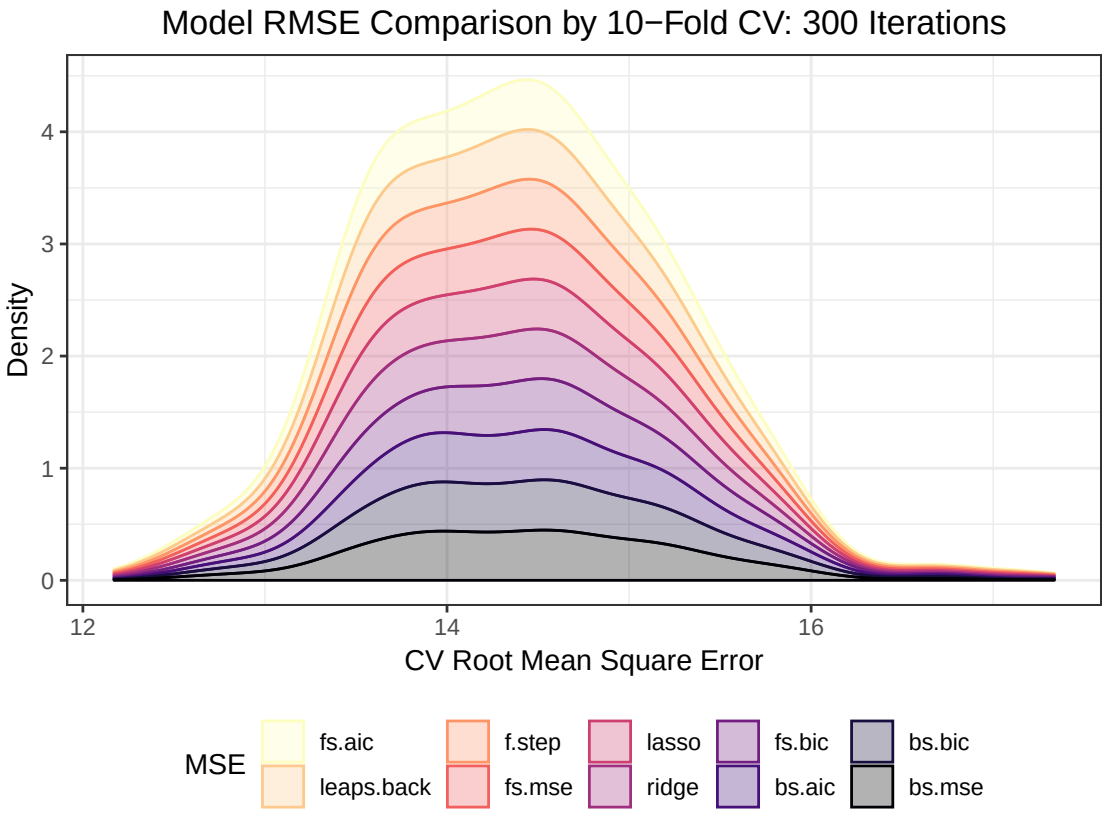
model	mean_rmse	median_rmse	variance_rmse	Q1	Q3
fs.aic	14.3561	14.3395	0.6741	13.7241	14.9068
leaps.back	14.3561	14.3395	0.6741	13.7241	14.9068
f.step	14.3628	14.3521	0.6734	13.7385	14.9132
fs.mse	14.3628	14.3521	0.6734	13.7385	14.9132
lasso	14.3631	14.3443	0.6728	13.7370	14.9113
ridge	14.3631	14.3443	0.6728	13.7370	14.9113
fs.bic	14.4102	14.3895	0.6587	13.7844	14.9780
bs.aic	14.4458	14.4323	0.6437	13.8707	15.0255
bs.bic	14.4458	14.4323	0.6437	13.8707	15.0255
bs.mse	14.4458	14.4323	0.6437	13.8707	15.0255

Visualize Model Comparison

```
#Violin box plots
violin.mse <- cv.df %>%
  dplyr::select(ends_with(".mse")) %>%
  gather(key = model, value = mse) %>%
  mutate(model = str_replace(model, "mse", ""),
         model = fct_reorder(model, mse, .desc = FALSE, .fun = mean)) %>%
  ggplot(aes(x = model, y = mse)) +
  geom_violin(aes(fill = model), trim = FALSE, alpha = 0.3) +
  geom_boxplot(width = 0.25) +
  labs(
    y = "CV Root Mean Sqaure Error",
    x = "Model",
    title = sprintf("Model RMSE Comparison by 10-Fold CV: %i Iterations", nrep)
  ) +
  viridis::scale_fill_viridis(
    option = "magma",
    name = "MSE",
    begin = 1,
    end = 0,
```

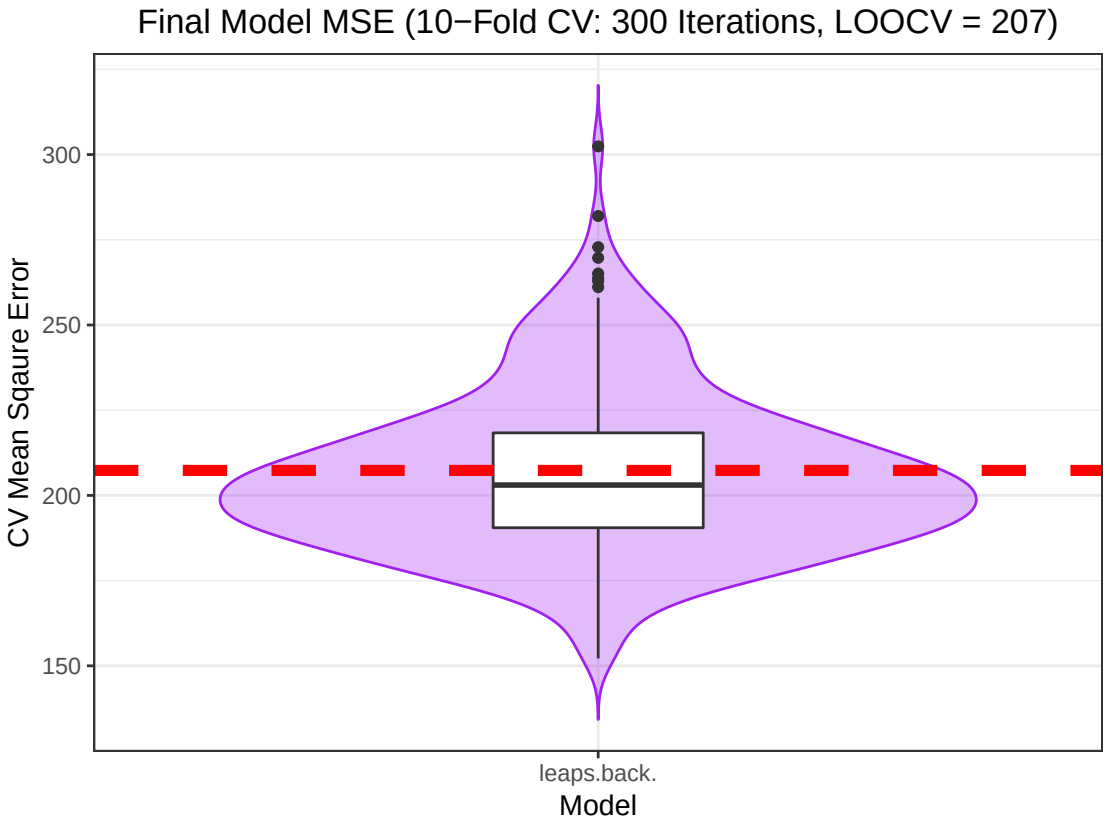
```
    discrete = TRUE)

density.mse <- cv.df %>%
   dplyr::select(ends_with(".mse")) %>%
  gather(key = model, value = mse) %>%
  mutate(model = str_replace(model, ".mse", ""),
         model = fct_reorder(model, mse, .desc = FALSE, .fun = mean)) %>%
  ggplot(aes(x = mse, colour = model, fill = model)) +
  geom_density(position = "stack", alpha = 0.3) +
  #geom_boxplot(width = 0.25) +
  labs(
    y = "Density",
    x = "CV Root Mean Square Error",
    title = sprintf("Model RMSE Comparison by 10-Fold CV: %i Iterations", nrep)
  ) +
  viridis::scale_fill_viridis(
    option = "magma",
    name = "MSE",
    begin = 1,
    end = 0,
    discrete = TRUE) +
    viridis::scale_colour_viridis(
    option = "magma",
    name = "MSE",
    begin = 1,
    end = 0,
    discrete = TRUE)
```

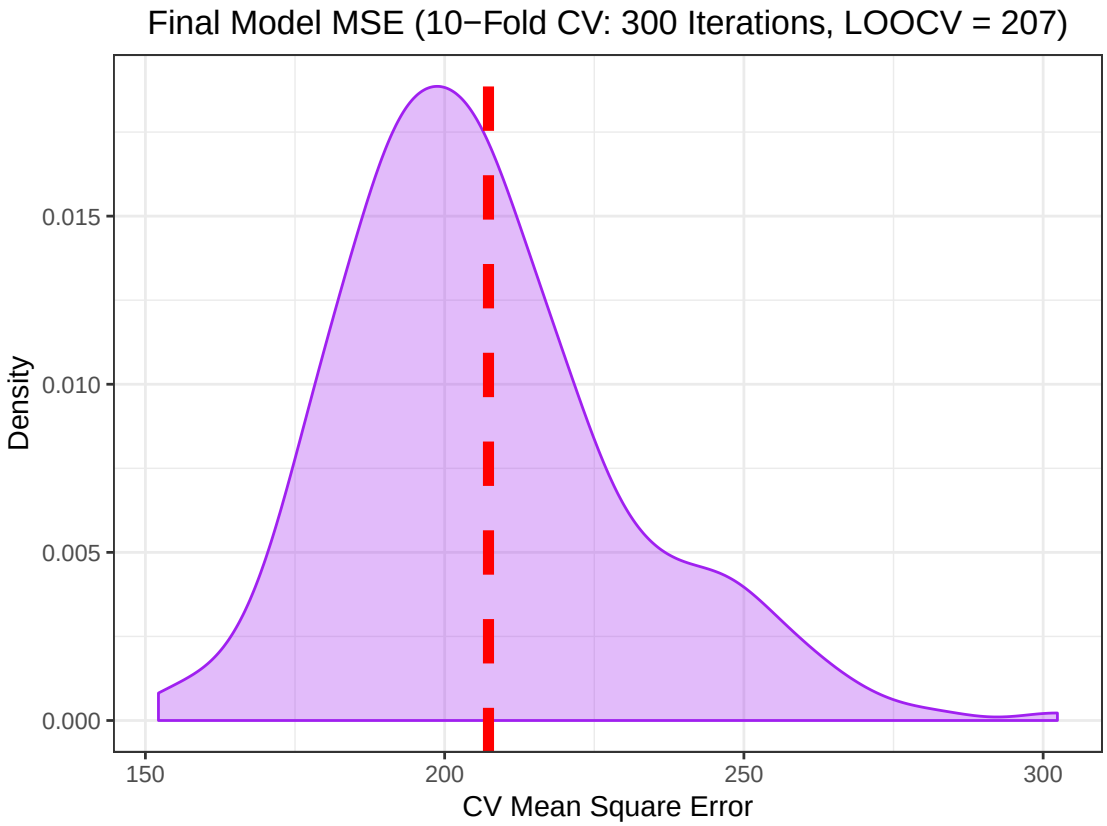

Model RMSE Comparison by 10−Fold CV: 300 Iterations
CV Root Mean Sqaure Error
18
16
14
12
fs.aic.
leaps.back.
f.step.
fs..mse
lasso.
ridge.
fs.bic.
bs..mse
bs.aic.
bs.bic.
Model
MSE
fs.aic.
leaps.back.
f.step.
fs..mse
lasso.
ridge.
fs.bic.
bs..mse
bs.aic.
bs.bic.


Model RMSE Comparison by 10−Fold CV: 300 Iterations
Density
4
3
2
1
0
12
14
16
CV Root Mean Square Error
MSE
fs.aic
leaps.back
f.step
fs.mse
lasso
ridge
fs.bic
bs.aic
bs.bic
bs.mse

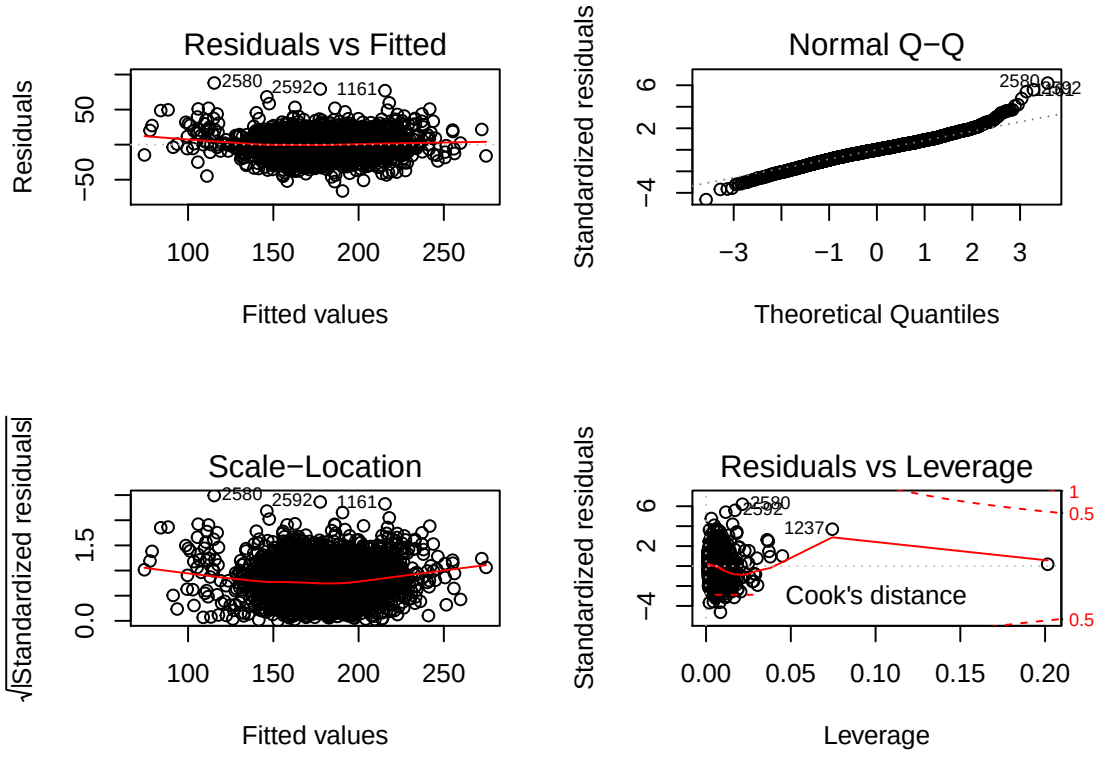
b. LOOCV Validation with PRESS Criterion

```
lm.loocv <- function(fit){
  h <- lm.influence(fit)$h
  mean((residuals(fit) / (1 - h))^2)
}

final.model <- lm(target_death_rate ~ pct_white + pct_hs25_over +  # Leap package back AIC ~ back p val
          pct_employed16_over + pct_private_coverage + pct_public_coverage +
          avg_ann_count_pop + avg_deaths_yr_pop + med_income + study_quantile +
          median_age_all_gender + south, data = cancer.df)

glm.final.model <- glm(target_death_rate ~ pct_white + pct_hs25_over +  # Leap package back AIC ~ back
          pct_employed16_over + pct_private_coverage + pct_public_coverage +
          avg_ann_count_pop + avg_deaths_yr_pop + med_income + study_quantile +
          median_age_all_gender + south, data = cancer.df, family = "gaussian")

#LOOCV
#cv.glm(cancer.df, glm.final.model)$delta #207

final.loocv <- lm.loocv(final.model) #207

#Number of CV iterations
nrep <- 300

#make sure target_death_rate is the first index in the data set (just way easier that way) District of
#State is a problem, need to make a region variable
set.seed(4)
cv.df.2 <- cancer.df %>%
         crossv_mc(., n = nrep,  test = 0.1) %>%   #Create nrep 10-fold data frames, 10% ~ 10 fold
  mutate(train = map(train, as.tibble),
         test = map(test, as.tibble)) %>%                                          #Fit Mode
  mutate(leaps.back  = map(train, ~lm(target_death_rate ~ pct_white + pct_hs25_over +  # Leap package b
          pct_employed16_over + pct_private_coverage + pct_public_coverage +
          avg_ann_count_pop + avg_deaths_yr_pop + med_income + study_quantile +
          median_age_all_gender + south, data = .x))) %>%
  mutate(leaps.back.mse = map2_dbl(leaps.back, test, ~mse(model = .x, data = .y)))
```


Final Model MSE (10−Fold CV: 300 Iterations, LOOCV = 207)
CV Mean Sqaure Error
300
250
200
150
leaps.back.
Model


Final Model MSE (10−Fold CV: 300 Iterations, LOOCV = 207)
Density
0.015
0.010
0.005
0.000
150
200
250
300
CV Mean Square Error

4. Final Linear Model Diagnostics




```
#Scaled Diagnostic, unappreciably different
par(mfrow = c(2,2))
plot(lm(target_death_rate ~ ., data = cancer.df[, c(fs.aic.preds, 1)]))
```

Appendix

Optional Imputation Algorithm for Variables < 20% missing (two variables)

- pct_employed16_over ~ 4%
- pct_private_coverage_alone ~ 20%

```
#Impute those missing less than 20%
#1. pct_employed16_over
#2. pct_private_coverage_alone

#Set up appropriate test and train for pct_employed16_over (removing other missing % variable and respo
train.df <- cancer.df %>% dplyr::select(-c(pct_private_coverage_alone, target_death_rate)) %>% filter(!
test.df <- cancer.df %>% dplyr::select(-c(pct_private_coverage_alone, target_death_rate)) %>% filter(is

#Function for imputation (after correct test, train set up), charstring must literally be the character
impute.lasso <- function(train.df, test.df, charstring){

  if ((charstring %in% names(train.df))) {
```

```
#pull variable index
index <- which(names(train.df) == charstring)

#Set up Matrices
#Create Design Matrix Train
X <- train.df[ , -index] %>%
  names() %>%
  paste(., collapse = "+") %>%
  paste("~ ", .) %>%
  formula() %>%
  model.matrix(.,train.df)

#Create Design Matrix Test
X1 <- test.df[, -index] %>%
  names() %>%
  paste(., collapse = "+") %>%
  paste("~ ", .) %>%
  formula() %>%
  model.matrix(., test.df)

#Remove Intercept
X <- X[,-1]
X1 <- X1[,-1]

#Create Response vector (as matrix)
Y <- train.df[, index] %>% as.matrix()

#Optimize lambda
lambda.grid <- 10^seq(-3,1,length = 100)

#CV n = 10
cv.lasso <- cv.glmnet(X, Y, alpha = 1, intercept = TRUE, lambda = lambda.grid, family = "gaussian")

#Grab optimal lambda
opt.lambda.lasso <- cv.lasso$lambda.min

#Run model
unemploy.lasso <- glmnet(X, Y, alpha = 1, intercept = TRUE, lambda = opt.lambda.lasso, family = "gaussi

#Return predictions
predict(unemploy.lasso, newx = X1)
  }else{
    stop("Error: Incorrect variable name")
  }
}

#Impute employed16_over_preds (first since it has less missing data ~4%)
employed16_over_preds <- impute.lasso(train.df = train.df, test.df, "pct_employed16_over")

#Set up appropriate test and train
train.df <- cancer.df %>% dplyr::select(-c(pct_employed16_over, target_death_rate)) %>% filter(!is.na(p
test.df <- cancer.df %>% dplyr::select(-c(pct_employed16_over, target_death_rate)) %>% filter(is.na(pct
```

```r
#Impute pct_private_coverage_alone (second since it has more missing data ~20%)
pct_private_coverage_alone_preds <- impute.lasso(train.df = train.df, test.df, "pct_private_coverage_al

#Replace Imputed values
cancer.df <- cancer.df %>%
  mutate(imp_pct_employed16_over = ifelse(is.na(pct_employed16_over),
                                          employed16_over_preds, pct_employed16_over),
         imp_pct_private_coverage_alone = ifelse(is.na(pct_private_coverage_alone),
                                          pct_private_coverage_alone_preds, pct_private_coverage_alone)
        )

#Looks good, so we will replace imputed variables in our final data set
cancer.df <- cancer.df %>%
  dplyr::select(-c(pct_employed16_over, pct_private_coverage_alone))
```