Homework 4

In this homework, the objectives are to

1. Implement a k-Nearest Neighbors Classifier on a real world dataset
2. Implement cross validation with k-Nearest Neighbors Classifier
3. Implement a linear discriminant analysis classifier on a real world dataset
4. Implement Ridge and LASSO Regressions

Assignments will only be accepted in electronic format in RMarkdown (.rmd) files and knitted .html files. **5 points will be deducted for every assignment submission that does not include either the RMarkdown file or the knitted html file.** Your code should be adequately commented to clearly explain the steps you used to produce the analyses. RMarkdown homework files should be uploaded to Sakai with the naming convention date_lastname_firstname_HW[X].Rmd. For example, my first homework assignment would be named 20220830_Dunn_Jessilyn_HW1.Rmd. **It is important to note that 5 points will be deducted for every assignment that is named improperly.** Please add your answer to each question directly after the question prompt in the homework .Rmd file template provided below.

```
library(tidyverse)
library(ggplot2)
library(lubridate)
library(patchwork)
library(gridExtra)
library(psych)
library(corrplot)
library(ggfortify)
library(factoextra)
library(class) #knn
library(gmodels) # CrossTable()
library(caret) # creatFolds()
library(caTools) #sample.split()
library(ROCR) # prediction(), performance()
library(glmnet)
set.seed(123)
```

Dataset

Diabetic retinopathy https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set (https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set)

Terminologies:

Diabetic retinopathy: is a diabetes complication that affects eyes. It’s caused by damage to the blood vessels of the light-sensitive tissue at the back of the eye (retina). At first, diabetic retinopathy may cause no symptoms or only mild vision problems.

Microaneurysms (MA): Microaneurysms are the earliest clinically visible changes of diabetic retinopathy. They are localised capillary dilatations which are usually saccular (round). They appear as small red dots which are often in clusters, but may occur in isolation.

Exudate: a mass of cells and fluid that has seeped out of blood vessels or an organ, especially common as a result of inflammation.

Macula: The macula is the central area of the retina and is of particular interest to retina specialists. Remember that the retina is the light sensitive tissue which lines the inside of the eye. The macula is the functional center of the retina. It gives us the ability to see “20/20” and provides the best color vision.

Optic Disc: The optic disc or optic nerve head is the point of exit for ganglion cell axons leaving the eye. Because there are no rods or cones overlying the optic disc, it corresponds to a small blind spot in each eye. The ganglion cell axons form the optic nerve after they leave the eye.

Data Visualization and Preprocessing (14 points)

1. Load the CSV file titled “diabetic.csv” and print the first 5 rows using head() function. How many rows are there in the entire dataset?

```
diabetic_df <- read_csv("Data/diabetic.csv")
head(diabetic_df, 5)
```

```
## # A tibble: 5 x 8
##   acceptable_quality ma_detection_0.5 ma_detection_1.0 exudates_0.5 exudates_1.0
##                <dbl>            <dbl>            <dbl>        <dbl>        <dbl>
## 1                  1               22               14         49.9      0.00392
## 2                  1               24               13         57.7      0.00390
## 3                  1               62               33         55.8      0.00774
## 4                  1               55               31         40.5      0.00153
## 5                  1               44               27         18.0      0
## # ... with 3 more variables: macula_distance <dbl>, optic_disc_diameter <dbl>,
## #   label <dbl>
```

```
nrow(diabetic_df)
```

```
## [1] 1151
```

There are 1151 rows in the entire dataset.

2. The following are explanations of the columns included in this dataset:

- acceptable_quality: whether this observation has acceptable quality; 1 = acceptable; 0 = not acceptable
- ma_detection_0.5: detected macula area at 0.5 confidence
- ma_detection_1.0: detected macula area at 1.0 confidence
- exudates_0.5: detected exudates at 0.5 confidence, normalized by dividing the number of lesions with the diameter of the ROI to compensate different image sizes

- exudates_1.0: detected exudates at 1.0 confidence, normalized by dividing the number of lesions with the diameter of the ROI to compensate different image sizes
- macula_dist: the euclidean distance of the center of the macula and the center of the optic disc to provide important information regarding the patient's condition, normalized with the diameter of the ROI.
- optic_disc_diameter: the diameter of the optic disc
- label/dependent variable: 1 = contains signs of Diabetic Retinopathy (DR); 0 = no signs of DR

Filter and save a new dataframe that contains only observations of acceptable quality and then delete the acceptable_quality column. How many rows are left?

```
levels(as.factor(diabetic_df$acceptable_quality))
```

```
## [1] "0" "1"
```

```
acceptable <- diabetic_df %>%
  filter(acceptable_quality == 1) %>%
  subset(select = -c(acceptable_quality))
acceptable %>% nrow()
```

```
## [1] 1147
```

There are 1147 rows left.

3. Use scale() to standardize the independent variables in this dataset. Structure a new dataframe that has all the standardized independent variables as well as the binary label column. Hint: you can use the as_tibble() function to nicely format the standardized columns into a dataframe.

```
acc_scale <- acceptable %>%
  subset(select = -c(label)) %>%
  scale() %>%
  data.frame() %>%
  cbind(acceptable %>% select(label))
```

4. For simplicity, we will arbitrarily split our dataset into an 80:20 ratio for the training and testing datasets, respectively. Split your standardized dataset into two separate data frames – i.e. the first 80% of rows for training and the remaining 20% for testing. Name your dataframes appropriately (e.g. df_train and df_test). Then extract four new dataframes called X_train, X_test, which contain only the independent variables, and y_train, y_test, which contain only the labels.

```
acc_scale$id <- 1:nrow(acc_scale)

train <- acc_scale %>% dplyr::sample_frac(0.80)
test  <- dplyr::anti_join(acc_scale, train, by = 'id')

X_train <- train %>% subset(select = -c(label, id)) %>% as.matrix()
X_test <- test %>% subset(select = -c(label, id))

y_train <- train %>% select(label) %>% as.matrix()
y_test <- test %>% select(label) %>% as.matrix()
```

---

kNN (15 points)

5. Generate a knn() model where k is the square root of the number of observations in the training set, which is a typical starting choice for k.

- Learn its syntax from https://www.rdocumentation.org/packages/class/versions/7.3-(https://www.rdocumentation.org/packages/class/versions/7.3-) 15/topics/knn.
- Note: Your training and test sets should only contain numeric values.
- Note: The labels for the training dataset should be passed separately.
- It should be clear to you that the output of this function is a list of the predicted values for the test set you passed.

```
my_k <- round(sqrt(nrow(X_train)))
my_knn <- knn(train = X_train, test = X_test, cl = y_train, k = my_k)

my_knn # these are my predicted values for the test df
```

```
##   [1] 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 0 0 1 0 0 1 1 0 1 1 1 1 0 0 1
##  [38] 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 1 1 0
##  [75] 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1
## [112] 0 0 0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0
## [149] 0 0 1 1 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0
## [186] 1 0 0 0 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 0 1 1 0 1
## [223] 0 1 1 0 0 0 1
## Levels: 0 1
```

6. Create a confusion matrix of the prediction results using CrossTable().

- Set prop.chisq = FALSE.
- Learn its syntax from https://www.rdocumentation.org/packages/gmodels/versions/2.18.1/topics/CrossTable (https://www.rdocumentation.org/packages/gmodels/versions/2.18.1/topics/CrossTable)

```
my_crosstab <- CrossTable(x = y_test, y = my_knn, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  229
##
##
##              | my_knn
##       y_test |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |        85 |        22 |       107 |
##              |     0.794 |     0.206 |     0.467 |
##              |     0.620 |     0.239 |           |
##              |     0.371 |     0.096 |           |
## -------------|-----------|-----------|-----------|
##            1 |        52 |        70 |       122 |
##              |     0.426 |     0.574 |     0.533 |
##              |     0.380 |     0.761 |           |
##              |     0.227 |     0.306 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       137 |        92 |       229 |
##              |     0.598 |     0.402 |           |
## -------------|-----------|-----------|-----------|
##
##
```

```
my_crosstab
```

```
## $t
##    y
## x    0  1
##   0 85 22
##   1 52 70
##
## $prop.row
##    y
## x           0         1
##   0 0.7943925 0.2056075
##   1 0.4262295 0.5737705
##
## $prop.col
##    y
## x           0         1
##   0 0.6204380 0.2391304
##   1 0.3795620 0.7608696
##
## $prop.tbl
##    y
## x            0          1
##   0 0.37117904 0.09606987
##   1 0.22707424 0.30567686
```

7. Calculate and print accuracy, sensitivity, error rate, and precision. You may choose either to use the information from the printed confusion matrix or to calculate using the equations from lecture slides. However, make sure you print and annotate them clearly for full credit.

```
errorRate_7 <- mean(my_knn != y_test)
errorRate_7
```

```
## [1] 0.3231441
```

```
accuracy = 1-errorRate_7
```

```
conf_matrix<-table(my_knn,y_test)
sens <- sensitivity(conf_matrix)
```

```
prec <- precision(conf_matrix)
```

The accuracy is: 0.6768559

The sensitivity is 0.7943925

The error rate is 0.3231441

The precision is 0.620438

Cross Validation with kNN (10 points)

8. In order to try k -fold cross validation, use createFolds() to divide our standardized dataset into 5 groups. Print how many items each of the 5 groups contain.

- Note: There are two k values here that can have different values: one for kNN and the other for k-fold CV. We know this is confusing and wish that “k” was not the most common variable name for these methods!
- Note: createFolds() function samples randomly. Include set.seed(123) before your createFolds() function so that you will reproduce the same results every time. For more information, see http://rfunction.com/archives/62 (http://rfunction.com/archives/62). The number 123 is arbitrarily chosen for this homework.

```
set.seed(123)
my_folds <- createFolds(acc_scale$label, k = 5)

my_folds$Fold1 %>% length()
```

```
## [1] 229
```

```
my_folds$Fold2 %>% length()
```

```
## [1] 230
```

```
my_folds$Fold3 %>% length()
```

```
## [1] 230
```

```
my_folds$Fold4 %>% length()
```

```
## [1] 229
```

```
my_folds$Fold5 %>% length()
```

```
## [1] 229
```

9. Train kNN models with k = 33 (here, k is referring to kNN) for each of the 5 CV groups, compute their error rates, and print the average of the 5 error rates.Compare the average error rate with the error rate calculated in question 7, what is your observation?

```
errorRates <- vector()
for (i in 1:5){
  fold <- my_folds[[i]]
  kf_test <- acc_scale %>% filter(id %in% fold)
  kf_train <- anti_join(acc_scale, kf_test, by = "id")

  kf_x_train <- kf_train %>% subset(select = -c(label, id)) %>% as.matrix()
  kf_x_test <- kf_test %>% subset(select = -c(label, id))

  kf_y_train <- kf_train %>% select(label) %>% as.matrix()
  kf_y_test <- kf_test %>% select(label) %>% as.matrix()

  my_kf_k <- 33
  my_kf_knn <- knn(train = kf_x_train, test = kf_x_test, cl = kf_y_train, k = my_kf_k)

  errorRates[i] <- mean(my_kf_knn != kf_y_test)
}
mean_error_kf <- mean(errorRates)
mean_error_kf
```

```
## [1] 0.3652933
```

The error rates are different, the averaged error rate is 0.3652933 and the error rate in question 7 is 0.3231441.

Linear Discriminant Analysis (10 points)

```
library(MASS) # for LDA
```

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

```
## The following object is masked from 'package:patchwork':
##
##     area
```

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

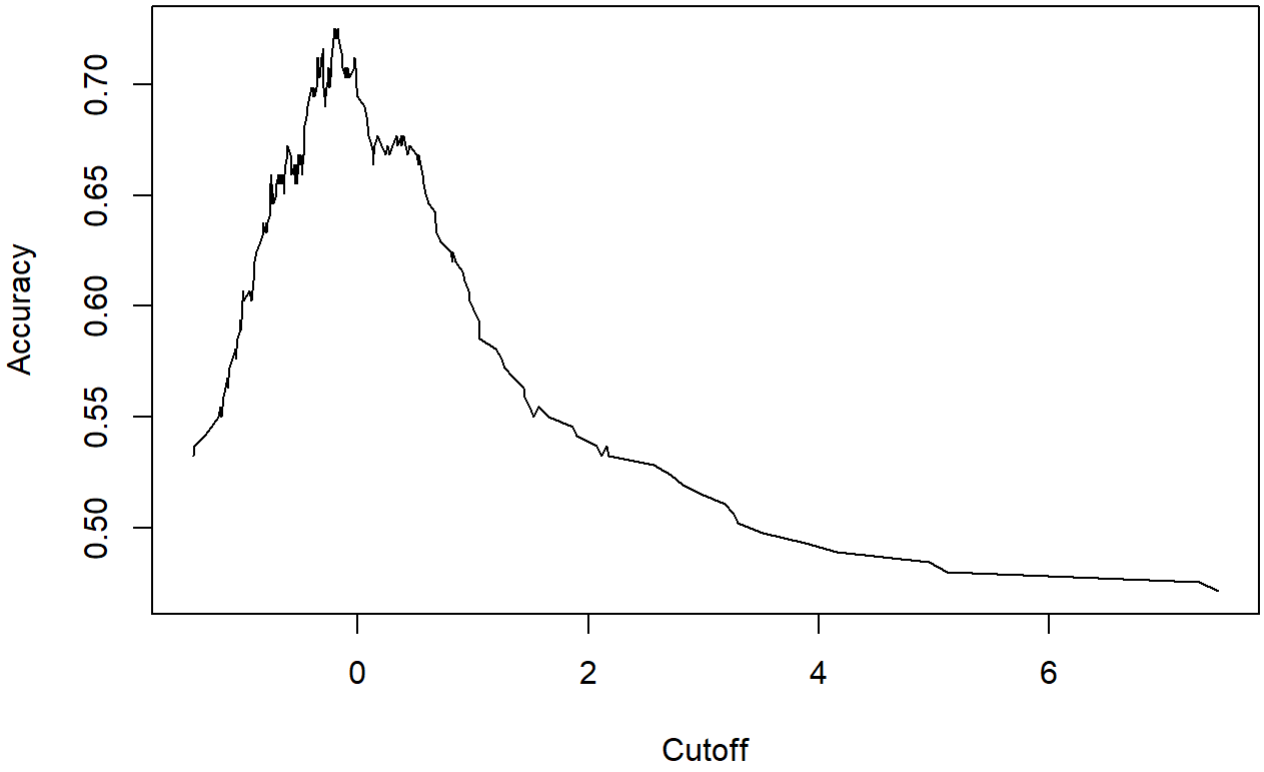
10. Train a linear discriminant analysis model on the training dataset using the lda() function.

- For more information, please refer to https://www.rdocumentation.org/packages/MASS/versions/7.3-53/topics/lda (https://www.rdocumentation.org/packages/MASS/versions/7.3-53/topics/lda)

```
my_lda <- lda(label ~ .,train %>% subset(select = -c(id)))
```

11. Evaluate LDA by plotting the ROC curve using prediction() and performance() from the ROCR package. Calculate and print the area under the ROC curve using performance().Interpret the results and compare it with the kNN results, which one has better performance in making predictions and why?

```
pred = predict(my_lda,test,type="response")
pred2 = prediction(pred$x, test$label)
perf = performance(pred2, "acc")
plot(perf)
```



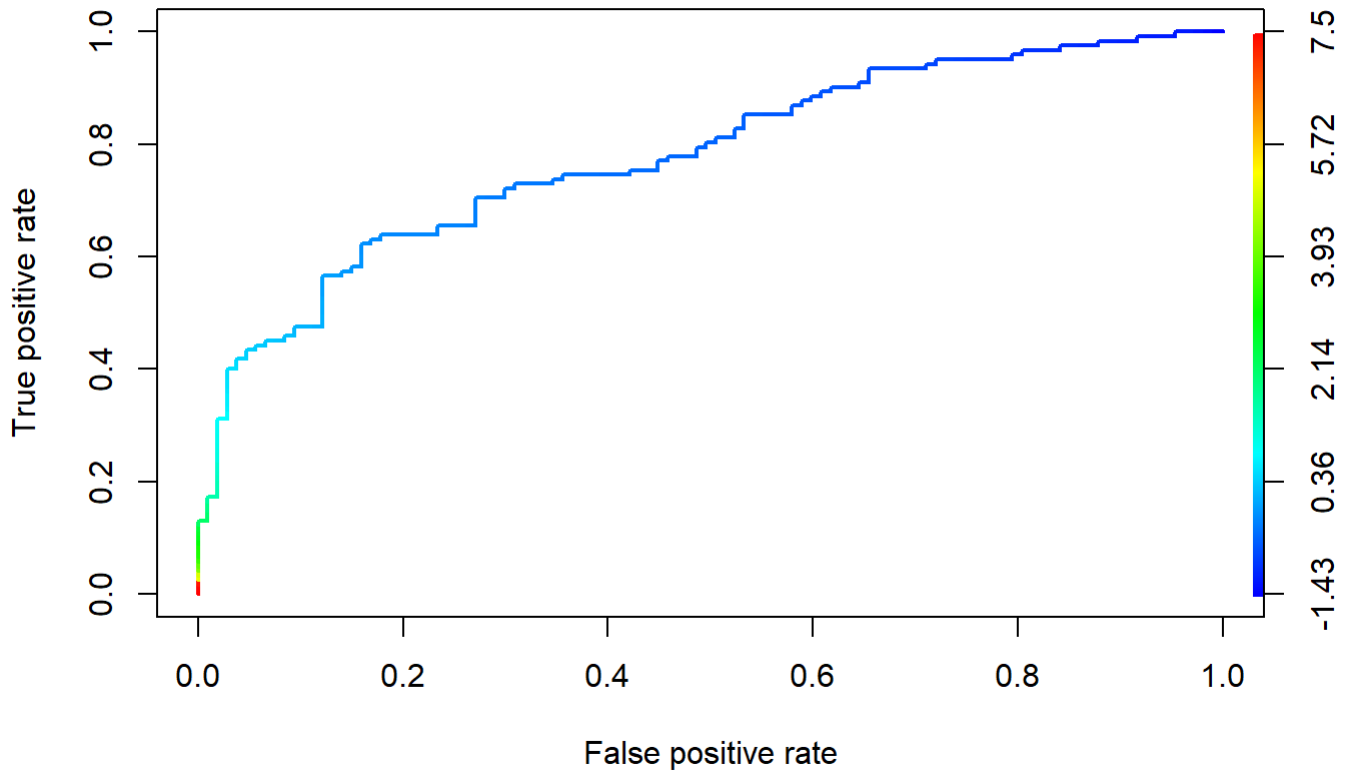

```
perf
```

```
## A performance instance
##   'Cutoff' vs. 'Accuracy' (alpha: 'none')
##   with 230 data points
```

```
auroc_obj <- performance(pred2, "auc")
my_auc <- auroc_obj@y.values[[1]]
my_auc
```

```
## [1] 0.7796078
```

```
roc = performance(pred2,"tpr","fpr")
plot(roc, colorize = T, lwd = 2)
```



I would say that LDA here is better, because the area under the ROC curve calculated is 0.7796078 and the error rate for the knn is larger than 1 - AUC. Though they are not exact opposites of each other, they can be somewhat related to each other in this way.

New Data Used Below

Load the dataset titled “life_expectancy_dataset.csv”. Attached on the Sakai page for this homework is an excel document explaining what the variables mean in this dataset. Print the first 5 rows of the imported dataset and take an initial glance at the structure of this data using the str() function. Mutate the dataframe so that there is a new column titled *developed* where integer 1 means that the country of this row is developed and 0 otherwise. Save a dataframe object with all columns except for *Country*, *Year*, and *Status*. (2 points)

```
life <- read_csv("Data/life_expectancy_dataset.csv")
```

```
## Rows: 1357 Columns: 22-- Column specification --------------------------------------------
---------
## Delimiter: ","
## chr  (2): Country, Status
## dbl (20): Year, Life.expectancy, Adult.Mortality, infant.deaths, Alcohol, pe...
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(life, 5)
```

```
## # A tibble: 5 x 22
##   Country      Year Status Life.expectancy Adult.Mortality infant.deaths Alcohol
##   <chr>       <dbl> <chr>            <dbl>           <dbl>         <dbl>   <dbl>
## 1 Afghanistan  2014 Devel~            59.9             271            64    0.01
## 2 Afghanistan  2013 Devel~            59.9             268            66    0.01
## 3 Afghanistan  2004 Devel~            57               293            87    0.02
## 4 Afghanistan  2003 Devel~            56.7             295            87    0.01
## 5 Albania      2015 Devel~            77.8              74             0    4.6
## # ... with 15 more variables: percentage.expenditure <dbl>, Hepatitis.B <dbl>,
## #   Measles <dbl>, BMI <dbl>, under.five.deaths <dbl>, Polio <dbl>,
## #   Total.expenditure <dbl>, Diphtheria <dbl>, HIV.AIDS <dbl>, GDP <dbl>,
## #   Population <dbl>, thinness..1.19.years <dbl>, thinness.5.9.years <dbl>,
## #   Income.composition.of.resources <dbl>, Schooling <dbl>
```

```
str(life)
```

```
## spec_tbl_df [1,357 x 22] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Country                        : chr [1:1357] "Afghanistan" "Afghanistan" "Afghanistan" "A
fghanistan" ...
##  $ Year                           : num [1:1357] 2014 2013 2004 2003 2015 ...
##  $ Status                         : chr [1:1357] "Developing" "Developing" "Developing" "Deve
loping" ...
##  $ Life.expectancy                : num [1:1357] 59.9 59.9 57 56.7 77.8 77.5 77.2 76.9 76.6 7
6.2 ...
##  $ Adult.Mortality                : num [1:1357] 271 268 293 295 74 8 84 86 88 91 ...
##  $ infant.deaths                  : num [1:1357] 64 66 87 87 0 0 0 0 0 1 ...
##  $ Alcohol                        : num [1:1357] 0.01 0.01 0.02 0.01 4.6 4.51 4.76 5.14 5.37
5.28 ...
##  $ percentage.expenditure         : num [1:1357] 73.5 73.2 15.3 11.1 365 ...
##  $ Hepatitis.B                    : num [1:1357] 62 64 67 65 99 98 99 99 99 99 ...
##  $ Measles                        : num [1:1357] 492 430 466 798 0 0 0 9 28 10 ...
##  $ BMI                            : num [1:1357] 18.6 18.1 13.8 13.4 58 57.2 56.5 55.8 55.1 5
4.3 ...
##  $ under.five.deaths              : num [1:1357] 86 89 120 122 0 1 1 1 1 1 ...
##  $ Polio                          : num [1:1357] 58 62 5 41 99 98 99 99 99 99 ...
##  $ Total.expenditure              : num [1:1357] 8.18 8.13 8.79 8.82 6 5.88 5.66 5.59 5.71 5.
34 ...
##  $ Diphtheria                     : num [1:1357] 62 64 5 41 99 98 99 99 99 99 ...
##  $ HIV.AIDS                       : num [1:1357] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
##  $ GDP                            : num [1:1357] 613 632 219 199 3954 ...
##  $ Population                     : num [1:1357] 327582 31731688 24118979 2364851 28873 ...
##  $ thinness..1.19.years           : num [1:1357] 17.5 17.7 19.5 19.7 1.2 1.2 1.3 1.3 1.4 1.4
...
##  $ thinness.5.9.years             : num [1:1357] 17.5 17.7 19.7 19.9 1.3 1.3 1.4 1.4 1.5 1.5
...
##  $ Income.composition.of.resources: num [1:1357] 0.476 0.47 0.381 0.373 0.762 0.761 0.759 0.7
52 0.738 0.725 ...
##  $ Schooling                      : num [1:1357] 10 9.9 6.8 6.5 14.2 14.2 14.2 14.2 13.3 12.5
...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Country = col_character(),
##   ..   Year = col_double(),
##   ..   Status = col_character(),
##   ..   Life.expectancy = col_double(),
##   ..   Adult.Mortality = col_double(),
##   ..   infant.deaths = col_double(),
##   ..   Alcohol = col_double(),
##   ..   percentage.expenditure = col_double(),
##   ..   Hepatitis.B = col_double(),
##   ..   Measles = col_double(),
##   ..   BMI = col_double(),
##   ..   under.five.deaths = col_double(),
##   ..   Polio = col_double(),
##   ..   Total.expenditure = col_double(),
##   ..   Diphtheria = col_double(),
##   ..   HIV.AIDS = col_double(),
##   ..   GDP = col_double(),
```

```
##   ..   Population = col_double(),
##   ..   thinness..1.19.years = col_double(),
##   ..   thinness.5.9.years = col_double(),
##   ..   Income.composition.of.resources = col_double(),
##   ..   Schooling = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
life2 <- life %>%
  mutate(developed = as.factor(case_when(
    Status == "Developed" ~ 1,
    TRUE ~ 0
  ))) %>%
  subset(select = -c(Country, Year, Status))
```

12. Now use sample.split() from the “caTools” package to split the data into 80:20 = train:test sets (80% of the data will be used for training, and 20% will be used to test the model). Set the seed of the random number generator for the random assignment of each observation to either the train or test set using set.seed(2022). (3 points)

```
set.seed(2022)
life2$my_splits <- sample.split(life2$developed, SplitRatio = 4/5)
train_q2 <- life2 %>% filter(my_splits == TRUE)
test_q2 <- life2 %>% filter(my_splits == FALSE)
```

We will use the glmnet() function from the glmnet package. Whereas all of the regression functions we have used so far, such as glm(), lm(), and regsubsets(), shared common syntax, glmnet() has a slightly different syntax. So to be able to use this function we will first pre-process our data. To do this, run the following lines of code to generate matrices of the testing and training datasets.

```
x.train <- model.matrix(Life.expectancy ~., train_q2)
y.train <- train_q2$Life.expectancy
x.test <- model.matrix(Life.expectancy ~., test_q2)
y.test <- test_q2$Life.expectancy
```

Ridge Regression (14 points)

Ridge regression seeks coefficient estimates that fit the data well by minimizing the residual sum of squares (RSS). This regularization is done by adding an extra term (the penalty term) to the original cost function: $RSS + \lambda \sum_{j=1}^{p} \beta_j^2$. Selecting a good value for λ is critical. We will first create an array of λ values we will test out.

```
lambdas <- 10^seq(12, -6, length = 300)
```

13. Build a ridge regression model using glmnet() using the training data and the labels that you built in question 12.

- For glmnet syntax information, refer to: https://www.rdocumentation.org/packages/glmnet/versions/3.0- (https://www.rdocumentation.org/packages/glmnet/versions/3.0-) 2/topics/glmnet

- Note: You need to set alpha = 0 to indicate you want to run ridge regression.

```
ridge1 <- glmnet(x.train, y.train, alpha = 0, lambda = lambdas)
ridge1
```

```
##
## Call:  glmnet(x = x.train, y = y.train, alpha = 0, lambda = lambdas)
##
##     Df  %Dev    Lambda
## 1   19  0.00 1.000e+12
## 2   19  0.00 8.706e+11
## 3   19  0.00 7.579e+11
## 4   19  0.00 6.598e+11
## 5   19  0.00 5.744e+11
## 6   19  0.00 5.000e+11
## 7   19  0.00 4.353e+11
## 8   19  0.00 3.790e+11
## 9   19  0.00 3.299e+11
## 10  19  0.00 2.872e+11
## 11  19  0.00 2.500e+11
## 12  19  0.00 2.177e+11
## 13  19  0.00 1.895e+11
## 14  19  0.00 1.650e+11
## 15  19  0.00 1.436e+11
## 16  19  0.00 1.250e+11
## 17  19  0.00 1.088e+11
## 18  19  0.00 9.475e+10
## 19  19  0.00 8.249e+10
## 20  19  0.00 7.181e+10
## 21  19  0.00 6.252e+10
## 22  19  0.00 5.442e+10
## 23  19  0.00 4.738e+10
## 24  19  0.00 4.125e+10
## 25  19  0.00 3.591e+10
## 26  19  0.00 3.126e+10
## 27  19  0.00 2.721e+10
## 28  19  0.00 2.369e+10
## 29  19  0.00 2.062e+10
## 30  19  0.00 1.795e+10
## 31  19  0.00 1.563e+10
## 32  19  0.00 1.361e+10
## 33  19  0.00 1.185e+10
## 34  19  0.00 1.031e+10
## 35  19  0.00 8.978e+09
## 36  19  0.00 7.816e+09
## 37  19  0.00 6.804e+09
## 38  19  0.00 5.923e+09
## 39  19  0.00 5.157e+09
## 40  19  0.00 4.489e+09
## 41  19  0.00 3.908e+09
## 42  19  0.00 3.402e+09
## 43  19  0.00 2.962e+09
## 44  19  0.00 2.579e+09
## 45  19  0.00 2.245e+09
## 46  19  0.00 1.954e+09
## 47  19  0.00 1.701e+09
## 48  19  0.00 1.481e+09
```

```
## 49  19  0.00 1.289e+09
## 50  19  0.00 1.122e+09
## 51  19  0.00 9.772e+08
## 52  19  0.00 8.507e+08
## 53  19  0.00 7.406e+08
## 54  19  0.00 6.447e+08
## 55  19  0.00 5.613e+08
## 56  19  0.00 4.886e+08
## 57  19  0.00 4.254e+08
## 58  19  0.00 3.703e+08
## 59  19  0.00 3.224e+08
## 60  19  0.00 2.806e+08
## 61  19  0.00 2.443e+08
## 62  19  0.00 2.127e+08
## 63  19  0.00 1.852e+08
## 64  19  0.00 1.612e+08
## 65  19  0.00 1.403e+08
## 66  19  0.00 1.222e+08
## 67  19  0.00 1.064e+08
## 68  19  0.00 9.259e+07
## 69  19  0.00 8.060e+07
## 70  19  0.00 7.017e+07
## 71  19  0.00 6.109e+07
## 72  19  0.00 5.318e+07
## 73  19  0.00 4.630e+07
## 74  19  0.00 4.030e+07
## 75  19  0.00 3.509e+07
## 76  19  0.00 3.055e+07
## 77  19  0.00 2.659e+07
## 78  19  0.00 2.315e+07
## 79  19  0.00 2.015e+07
## 80  19  0.00 1.754e+07
## 81  19  0.00 1.527e+07
## 82  19  0.00 1.330e+07
## 83  19  0.00 1.158e+07
## 84  19  0.00 1.008e+07
## 85  19  0.00 8.773e+06
## 86  19  0.00 7.637e+06
## 87  19  0.00 6.649e+06
## 88  19  0.00 5.788e+06
## 89  19  0.00 5.039e+06
## 90  19  0.00 4.387e+06
## 91  19  0.00 3.819e+06
## 92  19  0.00 3.325e+06
## 93  19  0.00 2.894e+06
## 94  19  0.00 2.520e+06
## 95  19  0.00 2.193e+06
## 96  19  0.00 1.910e+06
## 97  19  0.00 1.662e+06
## 98  19  0.01 1.447e+06
## 99  19  0.01 1.260e+06
## 100 19  0.01 1.097e+06
```

```
## 101 19  0.01 9.548e+05
## 102 19  0.01 8.313e+05
## 103 19  0.01 7.237e+05
## 104 19  0.01 6.300e+05
## 105 19  0.01 5.484e+05
## 106 19  0.02 4.775e+05
## 107 19  0.02 4.157e+05
## 108 19  0.02 3.618e+05
## 109 19  0.02 3.150e+05
## 110 19  0.03 2.742e+05
## 111 19  0.03 2.387e+05
## 112 19  0.04 2.078e+05
## 113 19  0.04 1.809e+05
## 114 19  0.05 1.575e+05
## 115 19  0.05 1.371e+05
## 116 19  0.06 1.194e+05
## 117 19  0.07 1.039e+05
## 118 19  0.08 9.047e+04
## 119 19  0.10 7.876e+04
## 120 19  0.11 6.857e+04
## 121 19  0.13 5.969e+04
## 122 19  0.14 5.197e+04
## 123 19  0.17 4.524e+04
## 124 19  0.19 3.938e+04
## 125 19  0.22 3.429e+04
## 126 19  0.25 2.985e+04
## 127 19  0.29 2.598e+04
## 128 19  0.33 2.262e+04
## 129 19  0.38 1.969e+04
## 130 19  0.44 1.714e+04
## 131 19  0.50 1.492e+04
## 132 19  0.58 1.299e+04
## 133 19  0.66 1.131e+04
## 134 19  0.76 9.847e+03
## 135 19  0.87 8.573e+03
## 136 19  1.00 7.463e+03
## 137 19  1.14 6.497e+03
## 138 19  1.31 5.656e+03
## 139 19  1.50 4.924e+03
## 140 19  1.72 4.287e+03
## 141 19  1.97 3.732e+03
## 142 19  2.26 3.249e+03
## 143 19  2.59 2.828e+03
## 144 19  2.96 2.462e+03
## 145 19  3.38 2.143e+03
## 146 19  3.86 1.866e+03
## 147 19  4.41 1.624e+03
## 148 19  5.03 1.414e+03
## 149 19  5.74 1.231e+03
## 150 19  6.53 1.072e+03
## 151 19  7.43 9.330e+02
## 152 19  8.43 8.120e+02
```

```
## 153 19  9.56 7.070e+02
## 154 19 10.82 6.160e+02
## 155 19 12.22 5.360e+02
## 156 19 13.77 4.660e+02
## 157 19 15.48 4.060e+02
## 158 19 17.36 3.540e+02
## 159 19 19.40 3.080e+02
## 160 19 21.61 2.680e+02
## 161 19 23.99 2.330e+02
## 162 19 26.53 2.030e+02
## 163 19 29.21 1.770e+02
## 164 19 32.03 1.540e+02
## 165 19 34.96 1.340e+02
## 166 19 37.96 1.170e+02
## 167 19 41.03 1.020e+02
## 168 19 44.11 8.800e+01
## 169 19 47.18 7.700e+01
## 170 19 50.20 6.700e+01
## 171 19 53.15 5.800e+01
## 172 19 55.99 5.100e+01
## 173 19 58.70 4.400e+01
## 174 19 61.27 3.800e+01
## 175 19 63.67 3.400e+01
## 176 19 65.91 2.900e+01
## 177 19 67.98 2.500e+01
## 178 19 69.87 2.200e+01
## 179 19 71.61 1.900e+01
## 180 19 73.18 1.700e+01
## 181 19 74.60 1.500e+01
## 182 19 75.87 1.300e+01
## 183 19 77.01 1.100e+01
## 184 19 78.02 1.000e+01
## 185 19 78.92 8.000e+00
## 186 19 79.70 7.000e+00
## 187 19 80.39 6.000e+00
## 188 19 80.98 6.000e+00
## 189 19 81.49 5.000e+00
## 190 19 81.93 4.000e+00
## 191 19 82.30 4.000e+00
## 192 19 82.61 3.000e+00
## 193 19 82.88 3.000e+00
## 194 19 83.10 2.000e+00
## 195 19 83.28 2.000e+00
## 196 19 83.43 2.000e+00
## 197 19 83.55 2.000e+00
## 198 19 83.66 1.000e+00
## 199 19 83.74 1.000e+00
## 200 19 83.82 1.000e+00
## 201 19 83.88 1.000e+00
## 202 19 83.93 1.000e+00
## 203 19 83.98 1.000e+00
## 204 19 84.02 1.000e+00
```

```
## 205 19 84.06 1.000e+00
## 206 19 84.09 0.000e+00
## 207 19 84.13 0.000e+00
## 208 19 84.16 0.000e+00
## 209 19 84.19 0.000e+00
## 210 19 84.23 0.000e+00
## 211 19 84.26 0.000e+00
## 212 19 84.29 0.000e+00
## 213 19 84.33 0.000e+00
## 214 19 84.36 0.000e+00
## 215 19 84.39 0.000e+00
## 216 19 84.43 0.000e+00
## 217 19 84.46 0.000e+00
## 218 19 84.49 0.000e+00
## 219 19 84.52 0.000e+00
## 220 19 84.55 0.000e+00
## 221 19 84.57 0.000e+00
## 222 19 84.59 0.000e+00
## 223 19 84.61 0.000e+00
## 224 19 84.63 0.000e+00
## 225 19 84.65 0.000e+00
## 226 19 84.66 0.000e+00
## 227 19 84.67 0.000e+00
## 228 19 84.68 0.000e+00
## 229 19 84.69 0.000e+00
## 230 19 84.70 0.000e+00
## 231 19 84.71 0.000e+00
## 232 19 84.71 0.000e+00
## 233 19 84.71 0.000e+00
## 234 19 84.72 0.000e+00
## 235 19 84.72 0.000e+00
## 236 19 84.72 0.000e+00
## 237 19 84.72 0.000e+00
## 238 19 84.73 0.000e+00
## 239 19 84.73 0.000e+00
## 240 19 84.73 0.000e+00
## 241 19 84.73 0.000e+00
## 242 19 84.73 0.000e+00
## 243 19 84.73 0.000e+00
## 244 19 84.73 0.000e+00
## 245 19 84.73 0.000e+00
## 246 19 84.73 0.000e+00
## 247 19 84.73 0.000e+00
## 248 19 84.73 0.000e+00
## 249 19 84.73 0.000e+00
## 250 19 84.73 0.000e+00
## 251 19 84.73 0.000e+00
## 252 19 84.73 0.000e+00
## 253 19 84.73 0.000e+00
## 254 19 84.73 0.000e+00
## 255 19 84.73 0.000e+00
## 256 19 84.73 0.000e+00
```

```
## 257 19 84.73 0.000e+00
## 258 19 84.73 0.000e+00
## 259 19 84.73 0.000e+00
## 260 19 84.73 0.000e+00
## 261 19 84.73 0.000e+00
## 262 19 84.73 0.000e+00
## 263 19 84.73 0.000e+00
## 264 19 84.73 0.000e+00
## 265 19 84.73 0.000e+00
## 266 19 84.73 0.000e+00
## 267 19 84.73 0.000e+00
## 268 19 84.73 0.000e+00
## 269 19 84.73 0.000e+00
## 270 19 84.73 0.000e+00
## 271 19 84.73 0.000e+00
## 272 19 84.73 0.000e+00
## 273 19 84.73 0.000e+00
## 274 19 84.73 0.000e+00
## 275 19 84.73 0.000e+00
## 276 19 84.73 0.000e+00
## 277 19 84.73 0.000e+00
## 278 19 84.73 0.000e+00
## 279 19 84.73 0.000e+00
## 280 19 84.73 0.000e+00
## 281 19 84.73 0.000e+00
## 282 19 84.73 0.000e+00
## 283 19 84.73 0.000e+00
## 284 19 84.73 0.000e+00
## 285 19 84.73 0.000e+00
## 286 19 84.73 0.000e+00
## 287 19 84.73 0.000e+00
## 288 19 84.73 0.000e+00
## 289 19 84.73 0.000e+00
## 290 19 84.73 0.000e+00
## 291 19 84.73 0.000e+00
## 292 19 84.73 0.000e+00
## 293 19 84.73 0.000e+00
## 294 19 84.73 0.000e+00
## 295 19 84.73 0.000e+00
## 296 19 84.73 0.000e+00
## 297 19 84.73 0.000e+00
## 298 19 84.73 0.000e+00
## 299 19 84.73 0.000e+00
## 300 19 84.73 0.000e+00
```

14. The glmnet package has a built-in cross validation function. Use cv.glmnet() to run cross-validated on ridge regression so that you can choose the optimal value of lambda. What is the λ value that gives rise to the ridge regression model with the minimal mean squared error (MSE), which we will define to be the best model for our purposes?

- Note: Make sure you set.seed(2022)
- Hint: accessing “lambda.min” outputs the value of λ that gives the minimum mean cross-validated error.

- For more information, see https://www.rdocumentation.org/packages/glmnet/versions/3.0-(https://www.rdocumentation.org/packages/glmnet/versions/3.0-) 2/topics/cv.glmnet
- Add a plot of the result from calling cv.glmnet(). What does this plot tell you?

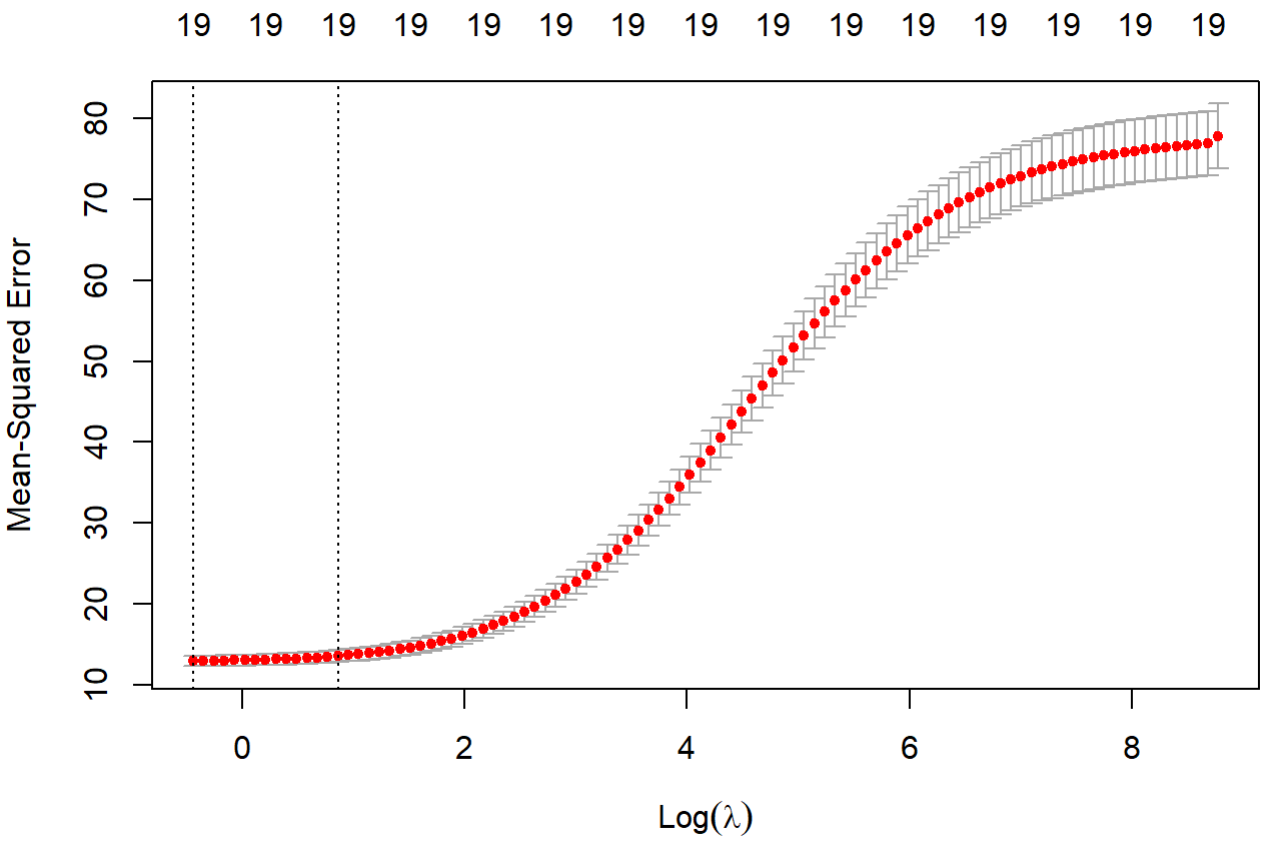
```
set.seed(2022)
ridge_cv <- cv.glmnet(x.train, y.train, alpha = 0)
ridge_cv
```

```
##
## Call:  cv.glmnet(x = x.train, y = y.train, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure     SE Nonzero
## min  0.645   100   12.95 0.6349      19
## 1se  2.373    86   13.55 0.7434      19
```

```
ridge_cv$lambda.min
```

```
## [1] 0.6450061
```

```
plot(ridge_cv)
```



the lambda value that gives rise to the model with the minimum SE is 100

This plot tells me that the lowest value for $log(\lambda)$ is approximately between [-.5,1.5].

15. Use predict() from the glmnet package to test your model. Make sure you use the λ derived from Question 15 and the test set. Calculate and print the mean squared error (MSE).

- For more information on the syntax, see https://www.rdocumentation.org/packages/glmnet/versions/1.1-1/topics/predict.glmnet (https://www.rdocumentation.org/packages/glmnet/versions/1.1-1/topics/predict.glmnet)

```
glm_preds <- glmnet::predict.glmnet(ridge1, x.test, s = ridge_cv$lambda.min)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
my_mse <- mean((glm_preds - y.test)^2)
my_mse
```

```
## [1] 16.7299
```

the MSE is 16.7299027

16. Calculate and print the sum of squared residuals (or RSS) and the R-squared statistic for the test set, using the predicted values from the best ridge regression model.

```
SSR <- sum((glm_preds - y.test)^2)
SST <- sum((glm_preds - mean(y.test))^2)
rsq <- SSR/SST
SSR
```

```
## [1] 4533.804
```

```
rsq
```

```
## [1] 0.2605656
```

LASSO (17 points)

17. Like ridge regression, lasso also seeks coefficient estimates that fit the data well by minimizing the residual sum of squares (RSS). This regularization is done by adding an extra term to the original cost function: $RSS + \lambda \sum_{j=1}^{p} |\beta_j|$ Selecting a good value for λ is critical for lasso as well.

First, build a lasso model using glmnet() using training data and labels from question 12.

- For its syntax information: https://www.rdocumentation.org/packages/glmnet/versions/3.0-2/topics/glmnet (https://www.rdocumentation.org/packages/glmnet/versions/3.0-2/topics/glmnet)

- Note: You need to set alpha = 1 to indicate you want to run lasso.
- Note: You should use the same lambdas array as you used previously

```
lasso1 <- glmnet(x.train, y.train, alpha = 1, lambda = lambdas)
lasso1
```

```
##
## Call:  glmnet(x = x.train, y = y.train, alpha = 1, lambda = lambdas)
##
##     Df  %Dev    Lambda
## 1    0  0.00 1.000e+12
## 2    0  0.00 8.706e+11
## 3    0  0.00 7.579e+11
## 4    0  0.00 6.598e+11
## 5    0  0.00 5.744e+11
## 6    0  0.00 5.000e+11
## 7    0  0.00 4.353e+11
## 8    0  0.00 3.790e+11
## 9    0  0.00 3.299e+11
## 10   0  0.00 2.872e+11
## 11   0  0.00 2.500e+11
## 12   0  0.00 2.177e+11
## 13   0  0.00 1.895e+11
## 14   0  0.00 1.650e+11
## 15   0  0.00 1.436e+11
## 16   0  0.00 1.250e+11
## 17   0  0.00 1.088e+11
## 18   0  0.00 9.475e+10
## 19   0  0.00 8.249e+10
## 20   0  0.00 7.181e+10
## 21   0  0.00 6.252e+10
## 22   0  0.00 5.442e+10
## 23   0  0.00 4.738e+10
## 24   0  0.00 4.125e+10
## 25   0  0.00 3.591e+10
## 26   0  0.00 3.126e+10
## 27   0  0.00 2.721e+10
## 28   0  0.00 2.369e+10
## 29   0  0.00 2.062e+10
## 30   0  0.00 1.795e+10
## 31   0  0.00 1.563e+10
## 32   0  0.00 1.361e+10
## 33   0  0.00 1.185e+10
## 34   0  0.00 1.031e+10
## 35   0  0.00 8.978e+09
## 36   0  0.00 7.816e+09
## 37   0  0.00 6.804e+09
## 38   0  0.00 5.923e+09
## 39   0  0.00 5.157e+09
## 40   0  0.00 4.489e+09
## 41   0  0.00 3.908e+09
## 42   0  0.00 3.402e+09
## 43   0  0.00 2.962e+09
## 44   0  0.00 2.579e+09
## 45   0  0.00 2.245e+09
## 46   0  0.00 1.954e+09
## 47   0  0.00 1.701e+09
## 48   0  0.00 1.481e+09
```

```
## 49   0  0.00 1.289e+09
## 50   0  0.00 1.122e+09
## 51   0  0.00 9.772e+08
## 52   0  0.00 8.507e+08
## 53   0  0.00 7.406e+08
## 54   0  0.00 6.447e+08
## 55   0  0.00 5.613e+08
## 56   0  0.00 4.886e+08
## 57   0  0.00 4.254e+08
## 58   0  0.00 3.703e+08
## 59   0  0.00 3.224e+08
## 60   0  0.00 2.806e+08
## 61   0  0.00 2.443e+08
## 62   0  0.00 2.127e+08
## 63   0  0.00 1.852e+08
## 64   0  0.00 1.612e+08
## 65   0  0.00 1.403e+08
## 66   0  0.00 1.222e+08
## 67   0  0.00 1.064e+08
## 68   0  0.00 9.259e+07
## 69   0  0.00 8.060e+07
## 70   0  0.00 7.017e+07
## 71   0  0.00 6.109e+07
## 72   0  0.00 5.318e+07
## 73   0  0.00 4.630e+07
## 74   0  0.00 4.030e+07
## 75   0  0.00 3.509e+07
## 76   0  0.00 3.055e+07
## 77   0  0.00 2.659e+07
## 78   0  0.00 2.315e+07
## 79   0  0.00 2.015e+07
## 80   0  0.00 1.754e+07
## 81   0  0.00 1.527e+07
## 82   0  0.00 1.330e+07
## 83   0  0.00 1.158e+07
## 84   0  0.00 1.008e+07
## 85   0  0.00 8.773e+06
## 86   0  0.00 7.637e+06
## 87   0  0.00 6.649e+06
## 88   0  0.00 5.788e+06
## 89   0  0.00 5.039e+06
## 90   0  0.00 4.387e+06
## 91   0  0.00 3.819e+06
## 92   0  0.00 3.325e+06
## 93   0  0.00 2.894e+06
## 94   0  0.00 2.520e+06
## 95   0  0.00 2.193e+06
## 96   0  0.00 1.910e+06
## 97   0  0.00 1.662e+06
## 98   0  0.00 1.447e+06
## 99   0  0.00 1.260e+06
## 100  0  0.00 1.097e+06
```

```
## 101  0  0.00 9.548e+05
## 102  0  0.00 8.313e+05
## 103  0  0.00 7.237e+05
## 104  0  0.00 6.300e+05
## 105  0  0.00 5.484e+05
## 106  0  0.00 4.775e+05
## 107  0  0.00 4.157e+05
## 108  0  0.00 3.618e+05
## 109  0  0.00 3.150e+05
## 110  0  0.00 2.742e+05
## 111  0  0.00 2.387e+05
## 112  0  0.00 2.078e+05
## 113  0  0.00 1.809e+05
## 114  0  0.00 1.575e+05
## 115  0  0.00 1.371e+05
## 116  0  0.00 1.194e+05
## 117  0  0.00 1.039e+05
## 118  0  0.00 9.047e+04
## 119  0  0.00 7.876e+04
## 120  0  0.00 6.857e+04
## 121  0  0.00 5.969e+04
## 122  0  0.00 5.197e+04
## 123  0  0.00 4.524e+04
## 124  0  0.00 3.938e+04
## 125  0  0.00 3.429e+04
## 126  0  0.00 2.985e+04
## 127  0  0.00 2.598e+04
## 128  0  0.00 2.262e+04
## 129  0  0.00 1.969e+04
## 130  0  0.00 1.714e+04
## 131  0  0.00 1.492e+04
## 132  0  0.00 1.299e+04
## 133  0  0.00 1.131e+04
## 134  0  0.00 9.847e+03
## 135  0  0.00 8.573e+03
## 136  0  0.00 7.463e+03
## 137  0  0.00 6.497e+03
## 138  0  0.00 5.656e+03
## 139  0  0.00 4.924e+03
## 140  0  0.00 4.287e+03
## 141  0  0.00 3.732e+03
## 142  0  0.00 3.249e+03
## 143  0  0.00 2.828e+03
## 144  0  0.00 2.462e+03
## 145  0  0.00 2.143e+03
## 146  0  0.00 1.866e+03
## 147  0  0.00 1.624e+03
## 148  0  0.00 1.414e+03
## 149  0  0.00 1.231e+03
## 150  0  0.00 1.072e+03
## 151  0  0.00 9.330e+02
## 152  0  0.00 8.120e+02
```

```
## 153  0  0.00 7.070e+02
## 154  0  0.00 6.160e+02
## 155  0  0.00 5.360e+02
## 156  0  0.00 4.660e+02
## 157  0  0.00 4.060e+02
## 158  0  0.00 3.540e+02
## 159  0  0.00 3.080e+02
## 160  0  0.00 2.680e+02
## 161  0  0.00 2.330e+02
## 162  0  0.00 2.030e+02
## 163  0  0.00 1.770e+02
## 164  0  0.00 1.540e+02
## 165  0  0.00 1.340e+02
## 166  0  0.00 1.170e+02
## 167  0  0.00 1.020e+02
## 168  0  0.00 8.800e+01
## 169  0  0.00 7.700e+01
## 170  0  0.00 6.700e+01
## 171  0  0.00 5.800e+01
## 172  0  0.00 5.100e+01
## 173  0  0.00 4.400e+01
## 174  0  0.00 3.800e+01
## 175  0  0.00 3.400e+01
## 176  0  0.00 2.900e+01
## 177  0  0.00 2.500e+01
## 178  0  0.00 2.200e+01
## 179  0  0.00 1.900e+01
## 180  0  0.00 1.700e+01
## 181  0  0.00 1.500e+01
## 182  0  0.00 1.300e+01
## 183  0  0.00 1.100e+01
## 184  0  0.00 1.000e+01
## 185  0  0.00 8.000e+00
## 186  0  0.00 7.000e+00
## 187  1  1.67 6.000e+00
## 188  2 16.88 6.000e+00
## 189  3 30.42 5.000e+00
## 190  4 41.62 4.000e+00
## 191  4 51.20 4.000e+00
## 192  4 58.48 3.000e+00
## 193  4 63.99 3.000e+00
## 194  4 68.16 2.000e+00
## 195  5 71.45 2.000e+00
## 196  5 74.17 2.000e+00
## 197  5 76.23 2.000e+00
## 198  5 77.79 1.000e+00
## 199  7 79.04 1.000e+00
## 200  7 80.16 1.000e+00
## 201  7 81.00 1.000e+00
## 202  7 81.63 1.000e+00
## 203  7 82.11 1.000e+00
## 204  9 82.48 1.000e+00
```

```
## 205  9 82.78 1.000e+00
## 206  9 83.01 0.000e+00
## 207  9 83.18 0.000e+00
## 208 10 83.32 0.000e+00
## 209 13 83.45 0.000e+00
## 210 14 83.56 0.000e+00
## 211 14 83.67 0.000e+00
## 212 14 83.75 0.000e+00
## 213 14 83.81 0.000e+00
## 214 14 83.86 0.000e+00
## 215 14 83.90 0.000e+00
## 216 14 83.92 0.000e+00
## 217 14 83.94 0.000e+00
## 218 14 83.96 0.000e+00
## 219 14 83.97 0.000e+00
## 220 16 83.98 0.000e+00
## 221 16 84.00 0.000e+00
## 222 17 84.01 0.000e+00
## 223 18 84.14 0.000e+00
## 224 18 84.28 0.000e+00
## 225 18 84.38 0.000e+00
## 226 18 84.47 0.000e+00
## 227 18 84.53 0.000e+00
## 228 18 84.57 0.000e+00
## 229 18 84.61 0.000e+00
## 230 18 84.64 0.000e+00
## 231 18 84.66 0.000e+00
## 232 18 84.67 0.000e+00
## 233 18 84.69 0.000e+00
## 234 18 84.70 0.000e+00
## 235 18 84.70 0.000e+00
## 236 18 84.71 0.000e+00
## 237 18 84.71 0.000e+00
## 238 18 84.72 0.000e+00
## 239 18 84.72 0.000e+00
## 240 18 84.72 0.000e+00
## 241 18 84.72 0.000e+00
## 242 18 84.73 0.000e+00
## 243 19 84.73 0.000e+00
## 244 19 84.73 0.000e+00
## 245 19 84.73 0.000e+00
## 246 19 84.73 0.000e+00
## 247 19 84.73 0.000e+00
## 248 19 84.73 0.000e+00
## 249 19 84.73 0.000e+00
## 250 19 84.73 0.000e+00
## 251 19 84.73 0.000e+00
## 252 19 84.73 0.000e+00
## 253 19 84.73 0.000e+00
## 254 19 84.73 0.000e+00
## 255 19 84.73 0.000e+00
## 256 19 84.73 0.000e+00
```

```
## 257 19 84.73 0.000e+00
## 258 19 84.73 0.000e+00
## 259 19 84.73 0.000e+00
## 260 19 84.73 0.000e+00
## 261 19 84.73 0.000e+00
## 262 19 84.73 0.000e+00
## 263 19 84.73 0.000e+00
## 264 19 84.73 0.000e+00
## 265 19 84.73 0.000e+00
## 266 19 84.73 0.000e+00
## 267 19 84.73 0.000e+00
## 268 19 84.73 0.000e+00
## 269 19 84.73 0.000e+00
## 270 19 84.73 0.000e+00
## 271 19 84.73 0.000e+00
## 272 19 84.73 0.000e+00
## 273 19 84.73 0.000e+00
## 274 19 84.73 0.000e+00
## 275 19 84.73 0.000e+00
## 276 19 84.73 0.000e+00
## 277 19 84.73 0.000e+00
## 278 19 84.73 0.000e+00
## 279 19 84.73 0.000e+00
## 280 19 84.73 0.000e+00
## 281 19 84.73 0.000e+00
## 282 19 84.73 0.000e+00
## 283 19 84.73 0.000e+00
## 284 19 84.73 0.000e+00
## 285 19 84.73 0.000e+00
## 286 19 84.73 0.000e+00
## 287 19 84.73 0.000e+00
## 288 19 84.73 0.000e+00
## 289 19 84.73 0.000e+00
## 290 19 84.73 0.000e+00
## 291 19 84.73 0.000e+00
## 292 19 84.73 0.000e+00
## 293 19 84.73 0.000e+00
## 294 19 84.73 0.000e+00
## 295 19 84.73 0.000e+00
## 296 19 84.73 0.000e+00
## 297 19 84.73 0.000e+00
## 298 19 84.73 0.000e+00
## 299 19 84.73 0.000e+00
## 300 19 84.73 0.000e+00
```

18. Use cv.glmnet() to run cross validation on lasso and determine the lambda that minimizes the MSE (which we will consider here to mean the best performing model). What is the λ value that gives rise to the best performing lasso model?

- Note: Make sure you set.seed(2022)
- Hint: $lambda$.min outputs the value of λ that gives the minimum mean cross-validated error (MSE).

- For more information, see https://www.rdocumentation.org/packages/glmnet/versions/3.0-2/topics/cv.glmnet (https://www.rdocumentation.org/packages/glmnet/versions/3.0-2/topics/cv.glmnet)
- Add a plot of the result from calling cv.glmnet(). What does this plot tell you?

```
set.seed(2022)
lasso_cv <- cv.glmnet(x.train, y.train, alpha = 1)
lasso_cv
```

```
##
## Call:  cv.glmnet(x = x.train, y = y.train, alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure     SE Nonzero
## min 0.00314    83   12.41 0.4633      18
## 1se 0.03867    56   12.84 0.6068      18
```

19. Use predict() from the glmnet package to test your model. Make sure you use the λ derived from Question 17 and the test set. Calculate and print the test mean squared error (MSE).

- For more information on the syntax, see https://www.rdocumentation.org/packages/glmnet/versions/1.1-1/topics/predict.glmnet (https://www.rdocumentation.org/packages/glmnet/versions/1.1-1/topics/predict.glmnet)

```
glm_preds2 <- glmnet::predict.glmnet(lasso1, x.test, s = lasso_cv$lambda.min)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
my_mse2 <- mean((glm_preds2 - y.test)^2)
my_mse2
```

```
## [1] 19.50385
```

20. Calculate and print the sum of squared residuals (i.e. RSS) and the R-squared statistic for the test set, using the predicted values from the best lasso model.

```
SSR2 <- sum((glm_preds2 - y.test)^2)
SST2 <- sum((glm_preds2 - mean(y.test))^2)
rsq2 <- SSR2/SST2
SSR2
```

```
## [1] 5285.544
```

```
rsq2
```

```
## [1] 0.2703072
```

21. We have implemented and tested both Ridge and LASSO models to predict life expectancy. What are your conclusions? Which model worked better? Provide quantitative metrics to support your reasoning when applicable.

The models have similar evaluation metrics: Ridge $R^2 = 0.2605656$ and $MSE = 16.7299027$ while Lasso had values of: $R^2 = 0.2703072$ and $MSE = 19.5038508$, though lasso allows us to remove variables from the model thus simplifying the model, the MSE in the ridge model is smaller than the MSE in the lasso model, while the R^2 values are nearly identical.