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?

```
diab <- read_csv("diabetic.csv", show_col_types = FALSE)
head(diab, n = 5)
```

```
## # A tibble: 5 × 8
##   acceptable_quality ma_detectio…¹ ma_de…² exuda…³ exuda…⁴ macul…⁵ optic…⁶ label
##                <dbl>         <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> <dbl>
## 1                  1            22      14    49.9 0.00392   0.487   0.100     0
## 2                  1            24      13    57.7 0.00390   0.521   0.144     0
## 3                  1            62      33    55.8 0.00774   0.531   0.129     1
## 4                  1            55      31    40.5 0.00153   0.483   0.115     0
## 5                  1            44      27    18.0 0         0.476   0.124     1
## # … with abbreviated variable names ¹ma_detection_0.5, ²ma_detection_1.0,
## #   ³exudates_0.5, ⁴exudates_1.0, ⁵macula_distance, ⁶optic_disc_diameter
```

```
nrow(diab)
```

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

There are 1151 rows in the 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?

```
diab1 <- diab %>% filter(acceptable_quality == 1) %>% select(-acceptable_quality)
nrow(diab1)
```

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

There are now 1147 rows.

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.

```
# Scaling independent variables
temp <- as_tibble(scale(diab1))

# Replacing columns with scaled ones
diab1$ma_detection_0.5 <- temp$ma_detection_0.5
diab1$ma_detection_1.0 <- temp$ma_detection_1.0
diab1$exudates_0.5 <- temp$exudates_0.5
diab1$exudates_1.0 <- temp$exudates_1.0
diab1$macula_distance <- temp$macula_distance
diab1$optic_disc_diameter <- temp$optic_disc_diameter
rm(temp)
```

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.

```
# Splitting dataset
df_train <- diab1[1:917,]
df_test <- diab1[918:1147,]

# Creating independent variable splits
X_train <- df_train %>% select(-label)
X_test <- df_test %>% select(-label)

# Creating dependent variable splits
y_train <- df_train %>% select(label)
y_test <- df_test %>% select(label)
```

---

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.

```
knn_model <- knn(X_train, X_test, y_train$label, sqrt(nrow(X_train)))
str(knn_model)
```

```
##  Factor w/ 2 levels "0","1": 1 1 2 1 1 2 1 1 1 2 ...
```

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)

```
cross <- CrossTable(y_test$label, knn_model, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  230
##
##
##              | knn_model
## y_test$label |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |        81 |        33 |       114 |
##              |     0.711 |     0.289 |     0.496 |
##              |     0.604 |     0.344 |           |
##              |     0.352 |     0.143 |           |
## -------------|-----------|-----------|-----------|
##            1 |        53 |        63 |       116 |
##              |     0.457 |     0.543 |     0.504 |
##              |     0.396 |     0.656 |           |
##              |     0.230 |     0.274 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       134 |        96 |       230 |
##              |     0.583 |     0.417 |           |
## -------------|-----------|-----------|-----------|
##
##
```

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.

Accuracy:

```
(cross$t[1] + cross$t[4]) / nrow(y_test)
```

```
## [1] 0.626087
```

Sensitivity:

```
cross$prop.row[4]
```

```
## [1] 0.5431034
```

Error Rate:

```
(cross$t[2] + cross$t[3]) / nrow(y_test)
```

```
## [1] 0.373913
```

Precision:

```
cross$prop.col[4]
```

```
## [1] 0.65625
```

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.

```
# Creating Folds
set.seed(123)
folds <- createFolds(diab1$ma_detection_0.5, k = 5)

# Printing length of each fold
length(folds$Fold1)
```

```
## [1] 228
```

```
length(folds$Fold2)
```

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

```
length(folds$Fold3)
```

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

```
length(folds$Fold4)
```

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

```
length(folds$Fold5)
```

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

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?

```
# Separating folds
fold_list <- list(folds$Fold1, folds$Fold2, folds$Fold3, folds$Fold4, folds$Fold5)

# Empty vector that will hold error rates for each model
knn_errors <- c()

for (fold in fold_list) {
  # Splitting training and testing sets
  k_x_train <- diab1 %>% select(-label) %>%
    filter(!as.integer(rownames(diab1)) %in% unlist(fold))
  k_x_test <- diab1 %>% select(-label) %>%
    filter(as.integer(rownames(diab1)) %in% unlist(fold))
  k_y_train <- diab1 %>% select(label) %>%
    filter(!as.integer(rownames(diab1)) %in% unlist(fold))
  k_y_test <- diab1 %>% select(label) %>%
    filter(as.integer(rownames(diab1)) %in% unlist(fold))

  # Training model
  set.seed(123)
  model <- knn(k_x_train, k_x_test, k_y_train$label, 33)

  # Calculating error rate for model and adding it to vector of error rates
  cross <- CrossTable(k_y_test$label, model, prop.chisq = FALSE)
  error <- (cross$t[2] + cross$t[3]) / nrow(k_y_test)
  knn_errors <- append(knn_errors, error)
}
```

```
# Return average error rate
mean(knn_errors)
```

```
## [1] 0.3688013
```

The mean error rate across the 5 folds is similar to the error rate calculated in question 7.

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.

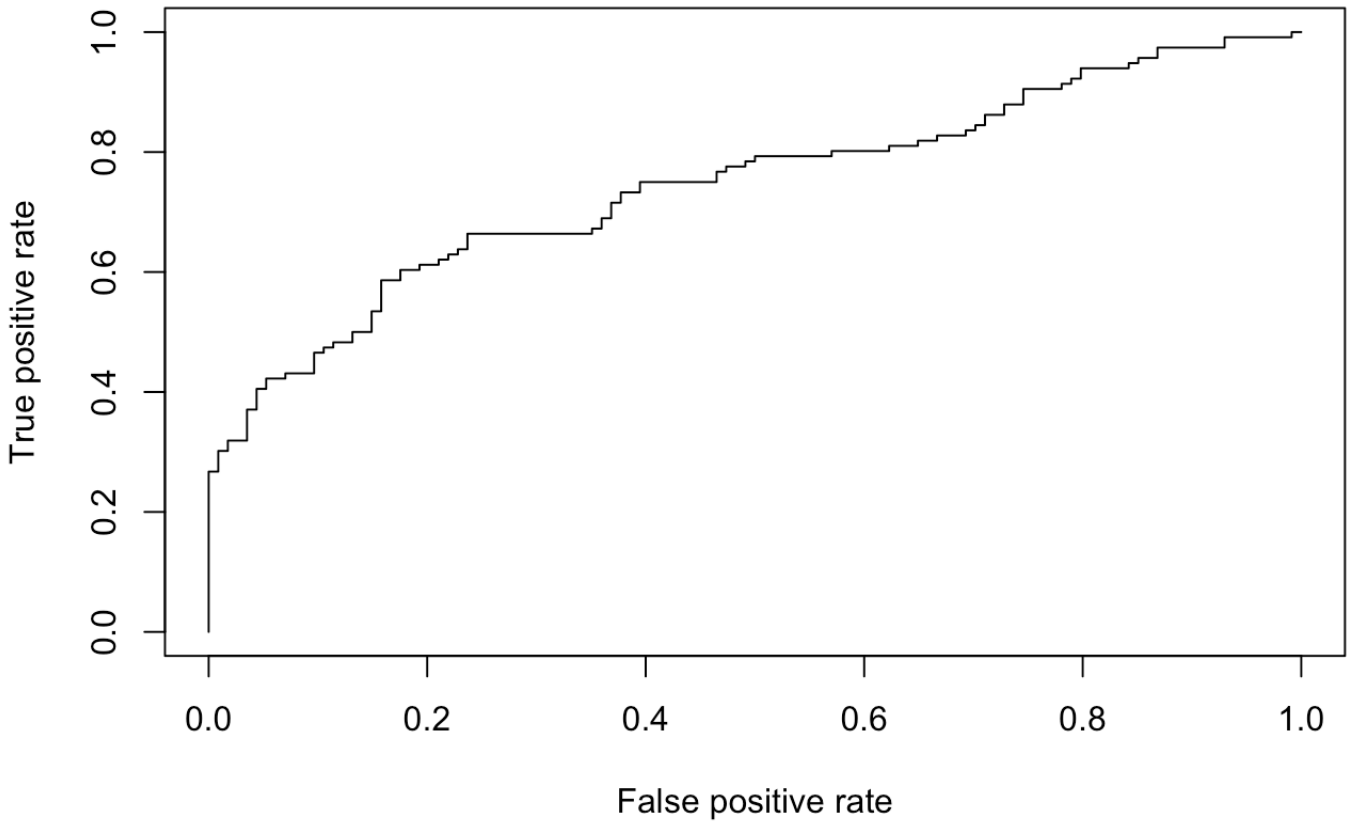
- For more information, please refer to https://www.rdocumentation.org/packages/MASS/versions/7.3-53/topics/lda (https://www.rdocumentation.org/packages/MASS/versions/7.3-53/topics/lda)

```
lda_model <- lda(X_train, y_train$label)
```

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?

```
# Creating prediction object
pred_model <- predict(object = lda_model, newdata = X_test)
pred <- prediction(pred_model$x, y_test)

# ROC curve using performance
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```




```
# Calculating and printing area under the ROC curve
perf_auc <- performance(pred, measure = "auc")
print(perf_auc@y.values)
```

```
## [[1]]
## [1] 0.7453872
```

```
# Calculating and printing error rate of model
1 / nrow(y_test) * sum(pred_model$class != y_test$label)
```

```
## [1] 0.3
```

Compared to the kNN results, the LDA has a lower error rate. It is possible that the decision boundary for class is linear or near linear in this situation, where we would expect LDA to outperform kNN. This could potentially explain the improved accuracy in the LDA model. kNN takes a non-parametric approach to the decision boundary, so we would expect it to outperform LDA if the decision boundary was non-linear.

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)

```
# Read in data and explore
life <- read_csv("life_expectancy_dataset.csv", show_col_types = FALSE)
head(life, n = 5)
```

```
## # A tibble: 5 × 22
##   Country    Year Status Life.…¹ Adult…² infan…³ Alcohol perce…⁴ Hepat…⁵ Measles
##   <chr>     <dbl> <chr>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Afghanis…  2014 Devel…    59.9     271      64    0.01    73.5      62     492
## 2 Afghanis…  2013 Devel…    59.9     268      66    0.01    73.2      64     430
## 3 Afghanis…  2004 Devel…    57       293      87    0.02    15.3      67     466
## 4 Afghanis…  2003 Devel…    56.7     295      87    0.01    11.1      65     798
## 5 Albania    2015 Devel…    77.8      74       0    4.6    365.       99       0
## # … with 12 more variables: 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>, and abbreviated
## #   variable names ¹Life.expectancy, ²Adult.Mortality, ³infant.deaths,
## #   ⁴percentage.expenditure, ⁵Hepatitis.B
```

```
str(life)
```

```
## spc_tbl_ [1,357 × 22] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Country                        : chr [1:1357] "Afghanistan" "Afghanistan" "Afgh
anistan" "Afghanistan" ...
##  $ Year                           : num [1:1357] 2014 2013 2004 2003 2015 ...
##  $ Status                         : chr [1:1357] "Developing" "Developing" "Develo
ping" "Developing" ...
##  $ Life.expectancy                : num [1:1357] 59.9 59.9 57 56.7 77.8 77.5 77.2
76.9 76.6 76.2 ...
##  $ 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 54.3 ...
##  $ 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 0.76
1 0.759 0.752 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>
```

```
# Clean and mutate data
life1 <- life %>% mutate(Developed = ifelse(Status == "Developed", 1, 0)) %>%
  dplyr::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)
life1 <- life1 %>% mutate(Split = sample.split(pull(life1, 1), SplitRatio = 0.8))
train.set <- life1 %>% filter(Split == TRUE)
test.set <- life1 %>% filter(Split == 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.set)
y.train <- train.set$Life.expectancy
x.test <- model.matrix(Life.expectancy ~., test.set)
y.test <- test.set$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.

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

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?

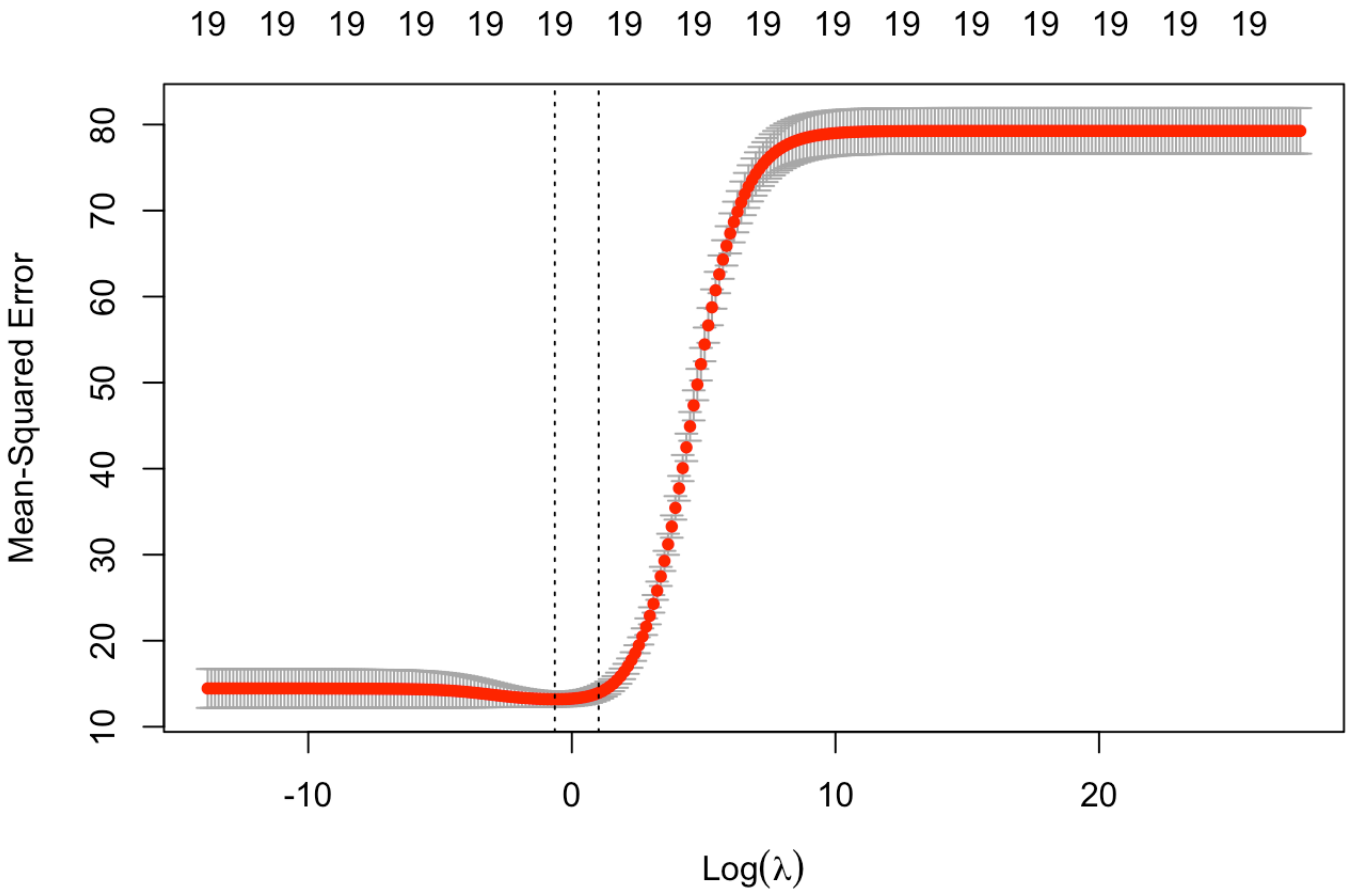
- Note: Make sure you set.seed(2022)
- Hint: accessing "lambda.min" outputs the value of λ that gives the minimum mean cross-validated error.
- For more information, see https://www.rdocumentation.org/packages/glmnet/versions/3.0- (https://www.rdocumentation.org/packages/glmnet/versions/3.0-) 2/topics/cv.glmnet
- Add a plot of the result from calling cv.glmnet(). What does this plot tell you?

```
# Perform CV
set.seed(2022)
cv_ridge <- cv.glmnet(x.train, y.train, alpha = 0, lambda = lambdas)

# Print lambda with minimal MSE
cv_ridge$lambda.min
```

```
## [1] 0.5236771
```

```
# Plot CV results
plot(cv_ridge)
```



The lambda value with minimal MSE is 0.5236771. The plot examines the relationship between the log transformed lambda values and the Mean-Squared Error of the model. We are looking for the lowest point in the curve, which corresponds to the optimal lambda value which we extracted. The log value of this lambda did the best job at minimizing the error during the cross validation process.

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)

```
# Generating predicted values
set.seed(2022)
ridge1 <- glmnet(x.train, y.train, alpha = 0, lambda = cv_ridge$lambda.min)
y.pred.r <- predict.glmnet(ridge1, x.test)

# Printing MSE
MSE_ridge <- mean((y.test - y.pred.r)^2)
MSE_ridge
```

```
## [1] 13.42056
```

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.

```
# Printing RSS
RSS_ridge <- sum((y.test - y.pred.r)^2)
RSS_ridge
```

```
## [1] 3529.607
```

```
# Printing R-squared statistic
TSS_ridge <- sum((y.test - mean(y.test))^2)
Rsquared_ridge <- 1 - RSS_ridge/TSS_ridge
Rsquared_ridge
```

```
## [1] 0.7712096
```

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

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

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)

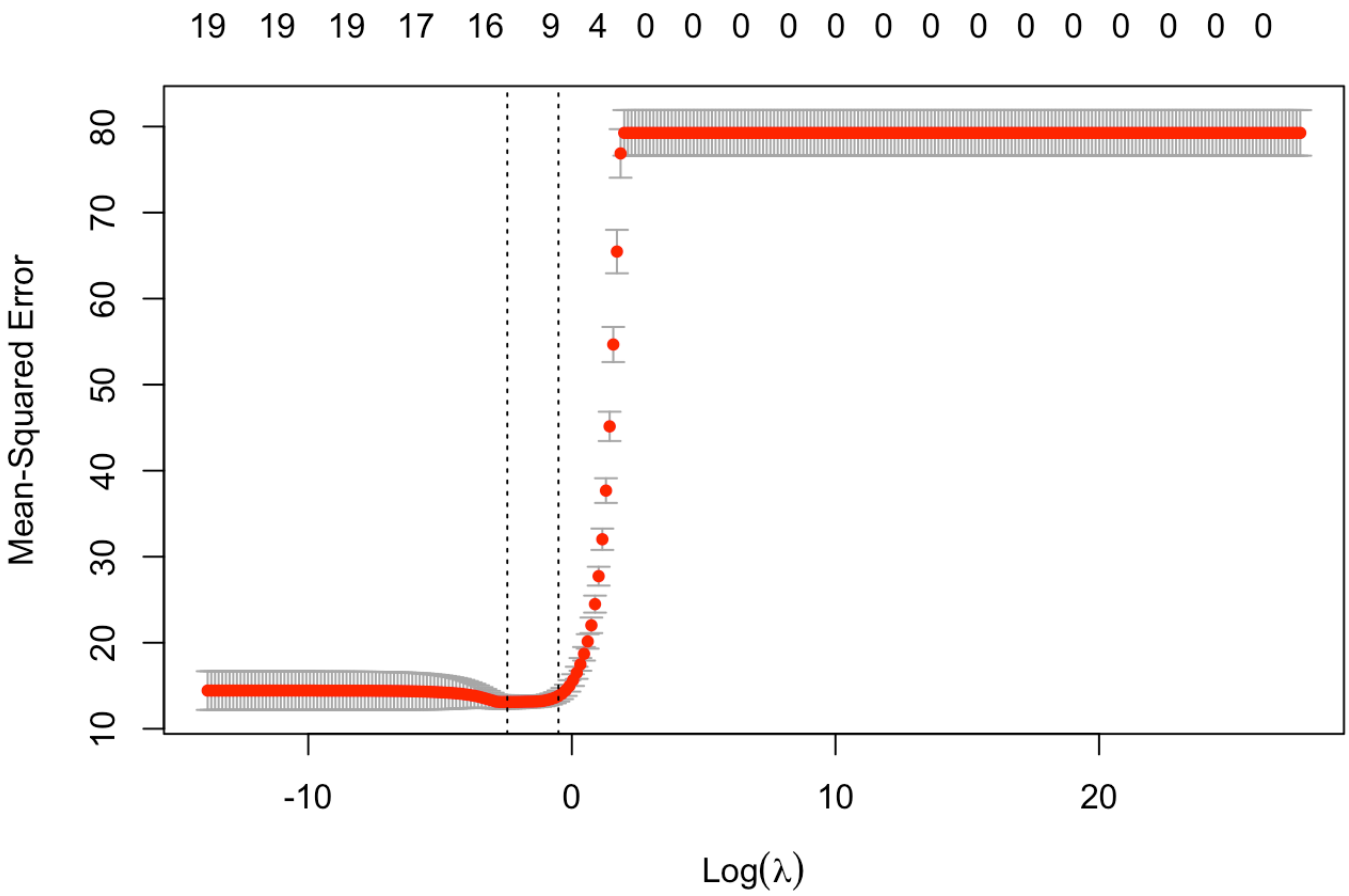
- Add a plot of the result from calling cv.glmnet(). What does this plot tell you?

```
# Perform CV
set.seed(2022)
cv_lasso <- cv.glmnet(x.train, y.train, alpha = 1, lambda = lambdas)

# Print lambda with minimal MSE
cv_lasso$lambda.min
```

```
## [1] 0.08638828
```

```
# Plot CV results
plot(cv_lasso)
```



The lambda value with minimal MSE is 0.08638828. The plot examines the relationship between the log transformed lambda values and the Mean-Squared Error of the model. We are looking for the lowest point in the curve, which corresponds to the optimal lambda value which we extracted. The log value of this lambda did the best job at minimizing the error during the cross validation process.

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)

```
# Generating predicted values
set.seed(2022)
lasso1 <- glmnet(x.train, y.train, alpha = 1, lambda = cv_lasso$lambda.min)
y.pred.l <- predict.glmnet(lasso1, x.test)

# Printing MSE
MSE_lasso <- mean((y.test - y.pred.l)^2)
MSE_lasso
```

```
## [1] 13.5178
```

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.

```
# Printing RSS
RSS_lasso <- sum((y.test - y.pred.l)^2)
RSS_lasso
```

```
## [1] 3555.181
```

```
# Printing R-squared statistic
TSS_lasso <- sum((y.test - mean(y.test))^2)
Rsquared_lasso <- 1 - RSS_lasso/TSS_lasso
Rsquared_lasso
```

```
## [1] 0.7695519
```

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 Ridge regression model explained slightly more of the variance in the dependent variable than the LASSO regression model. It explained 0.0016577 more variance overall. However, it is important to keep in mind that Ridge regression kept all of the predictors in the model. LASSO allows coefficients to go to zero, thus eliminating predictors from the model. This could make the LASSO model a bit more interpretable, as it performs its own variable selection.