Classifying Chronic Kidney Disease Using a Multivariate Binary Logistic Regression Model

Aanish Pradhan

July 21, 2022

1 Abstract

Chronic Kidney Disease (CKD) is one of the leading causes of death in the United States. In 2020, renal diseases accounted for approximately 53,000 deaths. It is estimated that approximately 15% of adults in the U.S. have CKD. Furthermore, of the 15% of adults that have CKD, 90% of them are not aware that they have the condition.[1] CKD is a challenging condition for physicians to diagnose and often requires a combination of laboratory testing and physical examinations to be able to diagnose a patient. We attempted to construct a statistical classification model that could determine whether or not a patient has CKD from a set of various predictors that could be obtained from laboratory blood test results or a physical examination. Our approach consisted of constructing and training several multivariate binary logistic regression models using various feature selection algorithms. The resulting models were benchmarked on a batch of testing data set aside earlier. The optimal model was chosen based on which model demonstrated the most favorable results in a confusion matrix. All models were able to obtain an accuracy of over 93%.

2 Introduction

Chronic Kidney Disease is an umbrella phrase, used to refer to a multitude of chronic, degenerative (i.e. loss of kidney function over time) renal disorders.[8] Diagnosing CKD is problematic because its symptoms do not present until later in life when the condition has seriously progressed. For this reason, CKD is often called a "silent killer".[5] Furthermore, CKD presents with similar symptoms as acute kidney injury as well as completely unrelated conditions. For example, microscopic hematuria (presence of red blood cells in the urine) and proteinuria (presence of protein in the urine) are characteristic symptoms of kidney disease. However, they are also observed in individuals after strenuous exercise such as long-distance running and weightlifting. Physicians have to use a combination of metrics collected over time from full-body physical examinations and laboratory blood test results as well as their intuition to diagnose CKD in patients.

3 Methods

Our approach consists of three phases: an initial setup, modeling and, lastly, testing.

3.1 Setup

In our setup phase, we will acquire, clean, format and conduct some initial exploratory data analysis (EDA).

3.1.1 Data Collection

Our data will come from a dataset housed in the University of California-Irvine's Machine Learning Repository available here.[2] The data itself was collected in a study conducted over the span of two months at the Alagappa University Health Care Centre in Tamilnadu, India. No other details were given regarding collection methods.

```r
# Read original dataset
originalData <- read.csv("Data/chronic_kidney_disease_full.csv", header = TRUE)
colnames(originalData)
```

```
##  [1] "id"       "X.age."   "X.bp."    "X.sg."    "X.al."    "X.su."
##  [7] "X.rbc."   "X.pc."    "X.pcc."   "X.ba."    "X.bgr."   "X.bu."
## [13] "X.sc."    "X.sod."   "X.pot."   "X.hemo."  "X.pcv."   "X.wbcc."
## [19] "X.rbcc."  "X.htn."   "X.dm."    "X.cad."   "X.appet." "X.pe."
## [25] "X.ane."   "X.class."
```

The dataset contains a multitude of features such as age, blood pressure, serum creatinine and other biometrics that are obtained from full-body physical exams and laboratory blood tests.

3.1.2 Data Wrangling

```r
head(originalData, n = 5)
```

```
##   id X.age. X.bp. X.sg. X.al. X.su. X.rbc.    X.pc.     X.pcc.      X.ba.
## 1  1     48    80 1.020     1     0      ?   normal notpresent notpresent
## 2  2      7    50 1.020     4     0      ?   normal notpresent notpresent
## 3  3     62    80 1.010     2     3 normal   normal notpresent notpresent
## 4  4     48    70 1.005     4     0 normal abnormal    present notpresent
## 5  5     51    80 1.010     2     0 normal   normal notpresent notpresent
##   X.bgr. X.bu. X.sc. X.sod. X.pot. X.hemo. X.pcv. X.wbcc. X.rbcc. X.htn. X.dm.
## 1    121    36   1.2      ?      ?    15.4     44    7800     5.2    yes   yes
## 2      ?    18   0.8      ?      ?    11.3     38    6000       ?     no    no
## 3    423    53   1.8      ?      ?     9.6     31    7500       ?     no   yes
## 4    117    56   3.8    111    2.5    11.2     32    6700     3.9    yes    no
## 5    106    26   1.4      ?      ?    11.6     35    7300     4.6     no    no
##   X.cad. X.appet. X.pe. X.ane. X.class.
## 1     no     good    no     no      ckd
## 2     no     good    no     no      ckd
## 3     no     poor    no    yes      ckd
## 4     no     poor   yes    yes      ckd
## 5     no     good    no     no      ckd
```

We observe some extraneous characters contaminating the dataset. We will replace extraneous characters, whitespace and blankspace with "NA" values.

```r
# Replace extraneous characters, whitespace and blankspace with NA values
replacedData <- read.csv("Data/chronic_kidney_disease_full.csv",
    header = TRUE, na.strings = c("", " ", "?"))
head(replacedData, n = 1)
```

```
##   id X.age. X.bp. X.sg. X.al. X.su. X.rbc.  X.pc.     X.pcc.      X.ba. X.bgr.
## 1  1     48    80  1.02     1     0   <NA> normal notpresent notpresent    121
##   X.bu. X.sc. X.sod. X.pot. X.hemo. X.pcv. X.wbcc. X.rbcc. X.htn. X.dm. X.cad.
## 1    36   1.2     NA     NA    15.4     44    7800     5.2    yes   yes     no
##   X.appet. X.pe. X.ane. X.class.
## 1     good    no     no      ckd
```

3.1.3 Data Cleaning

```r
any(is.na(replacedData))
```

```
## [1] TRUE
```

Our dataset contains observations with missing values. We will discard these observations.

```
# Omit observations with NA entries
cleanedData <- na.omit(replacedData)
any(is.na(cleanedData))
```

```
## [1] FALSE
```

3.1.4 Data Formatting

```
str(cleanedData)
```

```
## 'data.frame':    157 obs. of  26 variables:
##  $ id      : chr  "4" "10" "12" "15" ...
##  $ X.age.  : int  48 53 63 68 61 48 69 73 73 46 ...
##  $ X.bp.   : int  70 90 70 80 80 80 70 70 80 60 ...
##  $ X.sg.   : num  1 1.02 1.01 1.01 1.01 ...
##  $ X.al.   : int  4 2 3 3 2 4 3 0 2 1 ...
##  $ X.su.   : int  0 0 0 2 0 0 4 0 0 0 ...
##  $ X.rbc.  : chr  "normal" "abnormal" "abnormal" "normal" ...
##  $ X.pc.   : chr  "abnormal" "abnormal" "abnormal" "abnormal" ...
##  $ X.pcc.  : chr  "present" "present" "present" "present" ...
##  $ X.ba.   : chr  "notpresent" "notpresent" "notpresent" "present" ...
##  $ X.bgr.  : int  117 70 380 157 173 95 264 70 253 163 ...
##  $ X.bu.   : num  56 107 60 90 148 163 87 32 142 92 ...
##  $ X.sc.   : num  3.8 7.2 2.7 4.1 3.9 7.7 2.7 0.9 4.6 3.3 ...
##  $ X.sod.  : num  111 114 131 130 135 136 130 125 138 141 ...
##  $ X.pot.  : num  2.5 3.7 4.2 6.4 5.2 3.8 4 4 5.8 4 ...
##  $ X.hemo. : num  11.2 9.5 10.8 5.6 7.7 9.8 12.5 10 10.5 9.8 ...
##  $ X.pcv.  : int  32 29 32 16 24 32 37 29 33 28 ...
##  $ X.wbcc. : int  6700 12100 4500 11000 9200 6900 9600 18900 7200 14600 ...
##  $ X.rbcc. : num  3.9 3.7 3.8 2.6 3.2 3.4 4.1 3.5 4.3 3.2 ...
##  $ X.htn.  : chr  "yes" "yes" "yes" "yes" ...
##  $ X.dm.   : chr  "no" "yes" "yes" "yes" ...
##  $ X.cad.  : chr  "no" "no" "no" "yes" ...
##  $ X.appet.: chr  "poor" "poor" "poor" "poor" ...
##  $ X.pe.   : chr  "yes" "no" "yes" "yes" ...
##  $ X.ane.  : chr  "yes" "yes" "no" "no" ...
##  $ X.class.: chr  "ckd" "ckd" "ckd" "ckd" ...
##  - attr(*, "na.action")= 'omit' Named int [1:244] 1 2 3 5 6 7 8 9 11 13 ...
##   ..- attr(*, "names")= chr [1:244] "1" "2" "3" "5" ...
```

Some of our features were read in with the wrong type. We will correct the type of the features.

```
# Correct the variable types
formattedData <- cleanedData
formattedData$id <- as.integer(formattedData$id)
formattedData$X.sg. <- as.factor(formattedData$X.sg.)
formattedData$X.al. <- as.factor(formattedData$X.al.)
formattedData$X.su. <- as.factor(formattedData$X.su.)
formattedData$X.rbc. <- as.factor(formattedData$X.rbc.)
formattedData$X.pc. <- as.factor(formattedData$X.pc.)
formattedData$X.pcc. <- as.factor(formattedData$X.pcc.)
formattedData$X.ba. <- as.factor(formattedData$X.ba.)
formattedData$X.htn. <- as.factor(formattedData$X.htn.)
formattedData$X.dm. <- as.factor(formattedData$X.dm.)
```

```
formattedData$X.cad. <- as.factor(formattedData$X.cad.)
formattedData$X.appet. <- as.factor(formattedData$X.appet.)
formattedData$X.pe. <- as.factor(formattedData$X.pe.)
formattedData$X.ane. <- as.factor(formattedData$X.ane.)
formattedData$X.class. <- as.factor(formattedData$X.class.)
str(formattedData)
```

```
## 'data.frame':    157 obs. of  26 variables:
##  $ id      : int  4 10 12 15 21 23 28 49 59 72 ...
##  $ X.age.  : int  48 53 63 68 61 48 69 73 73 46 ...
##  $ X.bp.   : int  70 90 70 80 80 80 70 70 80 60 ...
##  $ X.sg.   : Factor w/ 5 levels "1.005","1.01",..: 1 4 2 2 3 5 2 1 4 2 ...
##  $ X.al.   : Factor w/ 5 levels "0","1","2","3",..: 5 3 4 4 3 5 4 1 3 2 ...
##  $ X.su.   : Factor w/ 6 levels "0","1","2","3",..: 1 1 1 3 1 1 5 1 1 1 ...
##  $ X.rbc.  : Factor w/ 2 levels "abnormal","normal": 2 1 1 2 1 2 2 2 1 2 ...
##  $ X.pc.   : Factor w/ 2 levels "abnormal","normal": 1 1 1 1 1 1 1 2 1 2 ...
##  $ X.pcc.  : Factor w/ 2 levels "notpresent","present": 2 2 2 2 1 1 1 1 1 1 ...
##  $ X.ba.   : Factor w/ 2 levels "notpresent","present": 1 1 1 2 1 1 1 1 1 1 ...
##  $ X.bgr.  : int  117 70 380 157 173 95 264 70 253 163 ...
##  $ X.bu.   : num  56 107 60 90 148 163 87 32 142 92 ...
##  $ X.sc.   : num  3.8 7.2 2.7 4.1 3.9 7.7 2.7 0.9 4.6 3.3 ...
##  $ X.sod.  : num  111 114 131 130 135 136 130 125 138 141 ...
##  $ X.pot.  : num  2.5 3.7 4.2 6.4 5.2 3.8 4 4 5.8 4 ...
##  $ X.hemo. : num  11.2 9.5 10.8 5.6 7.7 9.8 12.5 10 10.5 9.8 ...
##  $ X.pcv.  : int  32 29 32 16 24 32 37 29 33 28 ...
##  $ X.wbcc. : int  6700 12100 4500 11000 9200 6900 9600 18900 7200 14600 ...
##  $ X.rbcc. : num  3.9 3.7 3.8 2.6 3.2 3.4 4.1 3.5 4.3 3.2 ...
##  $ X.htn.  : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ X.dm.   : Factor w/ 2 levels "no","yes": 1 2 2 2 2 1 2 2 2 2 ...
##  $ X.cad.  : Factor w/ 2 levels "no","yes": 1 1 1 2 2 1 2 1 2 1 ...
##  $ X.appet.: Factor w/ 2 levels "good","poor": 2 2 2 2 2 1 1 1 1 1 ...
##  $ X.pe.   : Factor w/ 2 levels "no","yes": 2 1 2 2 2 1 2 2 1 1 ...
##  $ X.ane.  : Factor w/ 2 levels "no","yes": 2 2 1 1 2 2 1 1 1 1 ...
##  $ X.class.: Factor w/ 2 levels "ckd","notckd": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "na.action")= 'omit' Named int [1:244] 1 2 3 5 6 7 8 9 11 13 ...
##   ..- attr(*, "names")= chr [1:244] "1" "2" "3" "5" ...
```

With our dataset cleaned, we can proceed with exploratory data analysis.

3.1.5 Exploratory Data Analysis

We can begin EDA with a 5-number summary of the features in our prepared data.

```
summary(formattedData)
```

```
##        id            X.age.          X.bp.          X.sg.     X.al.    X.su.
##  Min.   :  4.0   Min.   : 6.0   Min.   : 50.00   1.005: 3   0:115   0:139
##  1st Qu.:243.0   1st Qu.:39.0   1st Qu.: 60.00   1.01 :23   1:  3   1:  6
##  Median :299.0   Median :50.0   Median : 80.00   1.015:10   2:  9   2:  6
##  Mean   :275.2   Mean   :49.4   Mean   : 74.08   1.02 :60   3: 15   3:  3
##  3rd Qu.:356.0   3rd Qu.:60.0   3rd Qu.: 80.00   1.025:61   4: 15   4:  2
##  Max.   :400.0   Max.   :83.0   Max.   :110.00                      5:  1
##       X.rbc.         X.pc.           X.pcc.          X.ba.
##  abnormal: 18   abnormal: 29   notpresent:143   notpresent:145
##  normal  :139   normal  :128   present   : 14   present   : 12
```

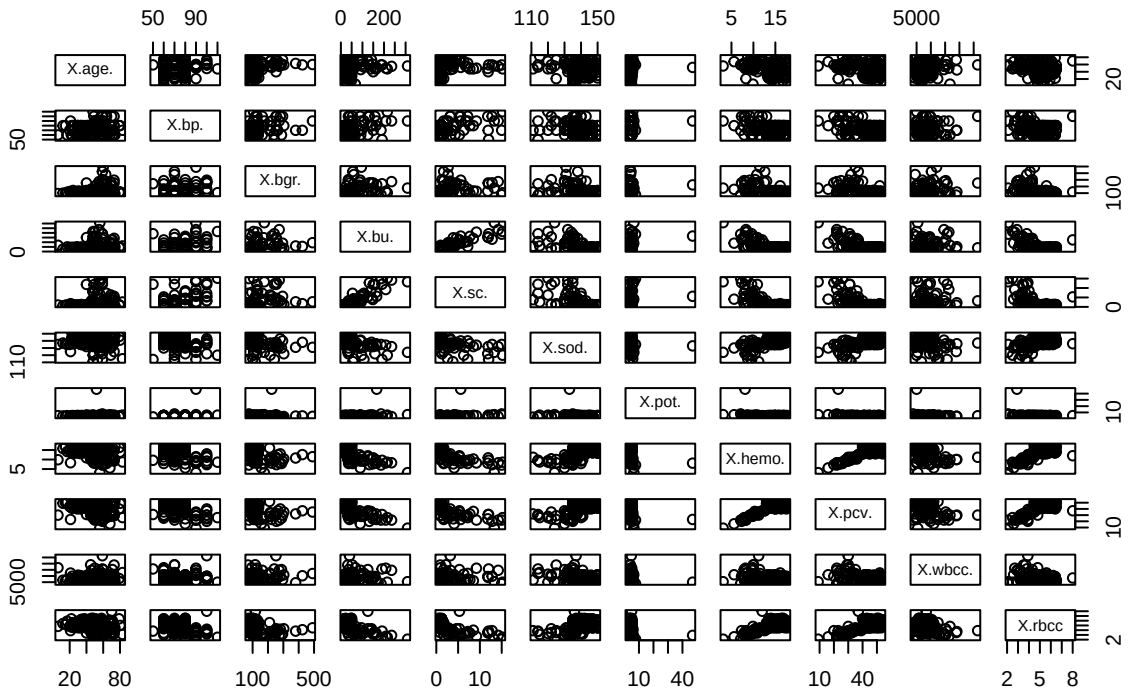
```
##
##
##
##
##      X.bgr.          X.bu.            X.sc.            X.sod.
##  Min.   : 70.0   Min.   : 10.00   Min.   : 0.400   Min.   :111.0
##  1st Qu.: 97.0   1st Qu.: 26.00   1st Qu.: 0.700   1st Qu.:135.0
##  Median :117.0   Median : 39.00   Median : 1.100   Median :139.0
##  Mean   :131.5   Mean   : 52.61   Mean   : 2.197   Mean   :138.8
##  3rd Qu.:132.0   3rd Qu.: 50.00   3rd Qu.: 1.700   3rd Qu.:144.0
##  Max.   :490.0   Max.   :309.00   Max.   :15.200   Max.   :150.0
##      X.pot.          X.hemo.          X.pcv.         X.wbcc.
##  Min.   : 2.500   Min.   : 3.10   Min.   : 9.00   Min.   : 3800
##  1st Qu.: 3.700   1st Qu.:12.60   1st Qu.:37.00   1st Qu.: 6500
##  Median : 4.500   Median :14.30   Median :44.00   Median : 7800
##  Mean   : 4.644   Mean   :13.69   Mean   :41.89   Mean   : 8464
##  3rd Qu.: 4.900   3rd Qu.:15.80   3rd Qu.:48.00   3rd Qu.: 9700
##  Max.   :47.000   Max.   :17.80   Max.   :54.00   Max.   :26400
##     X.rbcc.      X.htn.    X.dm.     X.cad.    X.appet.   X.pe.     X.ane.
##  Min.   :2.100   no :123   no :129   no :146   good:138   no :137   no :141
##  1st Qu.:4.500   yes: 34   yes: 28   yes: 11   poor: 19   yes: 20   yes: 16
##  Median :5.000
##  Mean   :4.892
##  3rd Qu.:5.600
##  Max.   :8.000
##    X.class.
##  ckd   : 43
##  notckd:114
##
##
##
##
```

We have several continuous numerical variables. We will examine correlation between the features.

```
# Extract continuous numerical variables
correlationMatrix <- data.frame(formattedData$X.age., formattedData$X.bp.,
    formattedData$X.bgr., formattedData$X.bu., formattedData$X.sc.,
        formattedData$X.sod., formattedData$X.pot., formattedData$X.hemo.,
            formattedData$X.pcv., formattedData$X.wbcc., formattedData$X.rbcc.)

# Abbreviate names
colnames(correlationMatrix) <- c("X.age.", "X.bp.", "X.bgr.", "X.bu.", "X.sc.",
    "X.sod.", "X.pot.", "X.hemo.", "X.pcv.", "X.wbcc.", "X.rbcc")

# Correlation plot
plot(correlationMatrix, main = "Scatterplot of Correlation Matrix")
```



The correlation matrix scatterplot shows some features are correlated with each other. We can quantify the correlation by examining the Pearson correlation coefficients computed from the correlation matrix.

```
# Compute Pearson correlation coefficients and round r-values
round(cor(correlationMatrix, method = "pearson"), digits = 2)
```

```
##         X.age. X.bp. X.bgr. X.bu. X.sc. X.sod. X.pot. X.hemo. X.pcv. X.wbcc.
## X.age.    1.00  0.08   0.31  0.19  0.20  -0.11   0.01   -0.25  -0.24   0.15
## X.bp.     0.08  1.00   0.19  0.32  0.39  -0.22   0.13   -0.28  -0.35   0.01
## X.bgr.    0.31  0.19   1.00  0.33  0.33  -0.28   0.10   -0.43  -0.44   0.21
## X.bu.     0.19  0.32   0.33  1.00  0.90  -0.49   0.25   -0.71  -0.71   0.13
## X.sc.     0.20  0.39   0.33  0.90  1.00  -0.53   0.14   -0.72  -0.73   0.13
## X.sod.   -0.11 -0.22  -0.28 -0.49 -0.53   1.00  -0.05    0.58   0.57  -0.18
## X.pot.    0.01  0.13   0.10  0.25  0.14  -0.05   1.00   -0.19  -0.21  -0.11
## X.hemo.  -0.25 -0.28  -0.43 -0.71 -0.72   0.58  -0.19    1.00   0.86  -0.34
## X.pcv.   -0.24 -0.35  -0.44 -0.71 -0.73   0.57  -0.21    0.86   1.00  -0.35
## X.wbcc.   0.15  0.01   0.21  0.13  0.13  -0.18  -0.11   -0.34  -0.35   1.00
## X.rbcc   -0.24 -0.23  -0.42 -0.62 -0.64   0.47  -0.19    0.74   0.74  -0.27
##         X.rbcc
## X.age.   -0.24
## X.bp.    -0.23
## X.bgr.   -0.42
## X.bu.    -0.62
## X.sc.    -0.64
## X.sod.    0.47
## X.pot.   -0.19
## X.hemo.   0.74
## X.pcv.    0.74
## X.wbcc.  -0.27
## X.rbcc    1.00
```

Some variables appear to have strong, linear relationships with other variables. This indicates that we could observe issues with multicollinearity in our models. Using a cutoff of $r = \pm 0.7$, we can identify which variables are highly correlated with others.

```
abs(cor(correlationMatrix, method = "pearson")) > 0.7
```

```
##         X.age. X.bp. X.bgr. X.bu. X.sc. X.sod. X.pot. X.hemo. X.pcv. X.wbcc.
## X.age.    TRUE FALSE  FALSE FALSE FALSE  FALSE  FALSE   FALSE  FALSE   FALSE
## X.bp.    FALSE  TRUE  FALSE FALSE FALSE  FALSE  FALSE   FALSE  FALSE   FALSE
## X.bgr.   FALSE FALSE   TRUE FALSE FALSE  FALSE  FALSE   FALSE  FALSE   FALSE
## X.bu.    FALSE FALSE  FALSE  TRUE  TRUE  FALSE  FALSE    TRUE   TRUE   FALSE
## X.sc.    FALSE FALSE  FALSE  TRUE  TRUE  FALSE  FALSE    TRUE   TRUE   FALSE
## X.sod.   FALSE FALSE  FALSE FALSE FALSE   TRUE  FALSE   FALSE  FALSE   FALSE
## X.pot.   FALSE FALSE  FALSE FALSE FALSE  FALSE   TRUE   FALSE  FALSE   FALSE
## X.hemo.  FALSE FALSE  FALSE  TRUE  TRUE  FALSE  FALSE    TRUE   TRUE   FALSE
## X.pcv.   FALSE FALSE  FALSE  TRUE  TRUE  FALSE  FALSE    TRUE   TRUE   FALSE
## X.wbcc.  FALSE FALSE  FALSE FALSE FALSE  FALSE  FALSE   FALSE  FALSE    TRUE
## X.rbcc   FALSE FALSE  FALSE FALSE FALSE  FALSE  FALSE    TRUE   TRUE   FALSE
##         X.rbcc
## X.age.   FALSE
## X.bp.    FALSE
## X.bgr.   FALSE
## X.bu.    FALSE
## X.sc.    FALSE
## X.sod.   FALSE
## X.pot.   FALSE
## X.hemo.   TRUE
## X.pcv.    TRUE
## X.wbcc.  FALSE
## X.rbcc    TRUE
```

We observe that `X.bu`, `X.hemo.`, and `X.rbcc.` exhibit multicollinearity with several other variables.

3.2 Modeling

In our modeling phase, we will perform a training and test dataset split, construct our models using various feature selection algorithms and perform various model diagnostics.

3.2.1 Train-Test Data Split

Our variable of interest is `X.class.`.

```
table(formattedData$X.class.)
```

```
##
##    ckd notckd
##     43    114
```

We will randomize the rows of our dataset and perform a 50-50 train-test data split. The training data will contain 22 "ckd"-classified observations 57 "notckd"-classified observations. The testing data will contain 21 "ckd"-classified observations and 57 "notckd"-classified observations. In order to maintain reproducibility, we will use a sample seed of "42".

```
# Generate a row-randomized dataset
set.seed(42)
randomizedData <- formattedData[sample(nrow(formattedData)), ]
```

From our randomized dataset, we can copy the first 22 "ckd"-classified observations into the training dataset and the subsequent 21 "notckd"-classified observations into the testing dataset.

```
# Construct testing dataset
testingData <- randomizedData # Dataset is constructed by Complement Rule

# Construct training dataset
trainingData <- testingData[-c(1:157), ] # Copy column names & preserve type

for (i in 1:length(testingData$id)) # Extract first 22 CKD observations
{
    if ((testingData[i, ]$X.class. == "ckd") &
        (sum(trainingData$X.class. == "ckd") < 22))
    {
        trainingData[nrow(trainingData) + 1, ] <- testingData[i, ]
        testingData <- testingData[-c(i), ]
    }
}

for (i in 1:length(testingData$id)) # Extract first 57 non-CKD observations
{
    if ((testingData[i, ]$X.class. == "notckd") &
        (sum(trainingData$X.class. == "notckd") < 57))
    {
        trainingData[nrow(trainingData) + 1, ] <- testingData[i, ]
        testingData <- testingData[-c(i), ]
    }
}

rm(i) # Clears the counter variable from the environment

# Reorder datasets
trainingData <- trainingData[order(trainingData$id), ]
testingData <- testingData[order(testingData$id), ]
```

With our training and testing datasets in place, we can proceed with building a model.

3.2.2 Feature Selection

We will construct our models using the Forward Selection, Backward Elimination, Sequential Selection (Bidirectional Elimination), Least Absolute Shrinkage and Selection Operator (LASSO) and Ridge Regression algorithms.

3.2.2.1 Forward Selection Algorithm To run the Forward Selection algorithm, we will construct a Null (intercept-only) model and a Full (all regressors) model. The algorithm will iteratively add regressors to the Null Model until it is no longer optimal to do so.

```
# Generate the Null Model
nullModel <- glm(X.class. ~ 1, data = trainingData, family = "binomial")

# Generate the Full Model
fullModel <- glm(X.class. ~ X.age. + X.bp. + X.bgr. + X.bu. + X.sod. + X.pot. +
    X.hemo. + X.pcv. + X.wbcc. + X.rbcc., data = trainingData,
        family = "binomial")
```

```r
# Run Forward Selection algorithm
forwardModel <- step(nullModel, direction = "forward",
    scope = list(upper = fullModel, lower = ~1), trace = 0)
summary(forwardModel)
```

```
##
## Call:
## glm(formula = X.class. ~ X.hemo. + X.bgr., family = "binomial",
##     data = trainingData)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -1.027e-04  -2.100e-08   2.100e-08   2.100e-08   1.587e-04
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -589.583 131340.614  -0.004    0.996
## X.hemo.         82.107  18155.300   0.005    0.996
## X.bgr.          -3.579    797.403  -0.004    0.996
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9.3459e+01  on 78  degrees of freedom
## Residual deviance: 4.4040e-08  on 76  degrees of freedom
## AIC: 6
##
## Number of Fisher Scoring iterations: 25
```

3.2.2.2 Backward Elimination Algorithm The Backward Elimination algorithm will iteratively remove the least statistically significant regressor from the Full Model until it is no longer optimal to do so.

```r
# Run the Backward Elimination algorithm
backwardModel <- step(fullModel, direction = "backward", trace = 0)
summary(backwardModel)
```

```
##
## Call:
## glm(formula = X.class. ~ X.bgr. + X.bu. + X.wbcc., family = "binomial",
##     data = trainingData)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -4.379e-05  -2.100e-08   2.100e-08   2.100e-08   5.280e-05
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.721e+02  1.714e+05   0.002    0.999
## X.bgr.      -6.095e-01  6.607e+02  -0.001    0.999
## X.bu.       -1.083e+00  9.808e+02  -0.001    0.999
## X.wbcc.     -1.295e-02  9.460e+00  -0.001    0.999
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 9.3459e+01  on 78  degrees of freedom
## Residual deviance: 5.0222e-09  on 75  degrees of freedom
## AIC: 8
##
## Number of Fisher Scoring iterations: 25
```

3.2.2.3 Sequential Selection Algorithm The Sequential Selection algorithm will iteratively either add or remove a regressor at each iteration until it is no longer optimal to do so.

```
# Run Sequential Selection algorithm
sequentialModel <- step(nullModel, direction = "both",
    scope = formula(fullModel), trace = 0)
summary(sequentialModel)
```

```
##
## Call:
## glm(formula = X.class. ~ X.hemo. + X.bgr., family = "binomial",
##     data = trainingData)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -1.027e-04  -2.100e-08   2.100e-08   2.100e-08   1.587e-04
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -589.583 131340.614  -0.004    0.996
## X.hemo.         82.107  18155.300   0.005    0.996
## X.bgr.          -3.579    797.403  -0.004    0.996
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9.3459e+01  on 78  degrees of freedom
## Residual deviance: 4.4040e-08  on 76  degrees of freedom
## AIC: 6
##
## Number of Fisher Scoring iterations: 25
```

The Sequential Selection algorithm yields the same linear model as the Forward Model, thus we can ignore this algorithm's output.

3.2.2.4 LASSO Regression Algorithm To run the LASSO Regression algorithm, we will utilize the glmnet package.[4] We will perform a k-fold cross-validation to find a value of λ that minimizes the Mean Squared Error (MSE). The algorithm will optimize a loss function that takes into account the sum of the absolute value of the regressors' coefficients. By doing so, it imposes a penalty on the optimization, causing the regressor coefficients to "shrink" towards zero, thereby minimizing the number of regressors required in the model.

```
# Load the glmnet package
library(glmnet)
```

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

```
## Loaded glmnet 4.1-4
```

```
# k-fold cross-validation
lassoY <- trainingData$X.class.
lassoX <- data.matrix(trainingData[, colnames(trainingData)[2:25]])
```

```
lassoCVModel <- cv.glmnet(lassoX, lassoY, alpha = 1, family = "binomial")
lassoBestLambda <- lassoCVModel$lambda.min
lassoBestLambda
```

```
## [1] 0.001499344
```

```
# Run LASSO Regression algorithm
lassoModel <- glmnet(lassoX, lassoY, alpha = 1, lambda = lassoBestLambda,
    family = "binomial")
coef(lassoModel)
```

```
## 25 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept) -1.2003909140
## X.age.       .
## X.bp.        .
## X.sg.        0.6097143183
## X.al.       -2.3346253391
## X.su.        .
## X.rbc.       0.6037182638
## X.pc.        0.4913849358
## X.pcc.       .
## X.ba.        .
## X.bgr.      -0.0022338519
## X.bu.        .
## X.sc.        .
## X.sod.       0.0181487875
## X.pot.       .
## X.hemo.      0.3466943614
## X.pcv.       0.0612109681
## X.wbcc.     -0.0002524382
## X.rbcc.      .
## X.htn.      -0.2641867169
## X.dm.       -2.9176751064
## X.cad.       .
## X.appet.     .
## X.pe.        .
## X.ane.       .
```

3.2.2.5 Ridge Regression Algorithm We will perform a k-fold cross-validation to find a value of λ that minimizes the MSE. Similar to the LASSO Regression algorithm, the Ridge Regression algorithm will minimize a loss function that accounts for the coefficients of the regressors. However, the loss function for the Ridge Regression algorithm involves the sum of the squares of the coefficients regressors as opposed to the sum of the absolute values.

```
# k-fold cross-validation
ridgeY <- trainingData$X.class.
ridgeX <- data.matrix(trainingData[, colnames(trainingData)[2:25]])
ridgeCVModel <- cv.glmnet(ridgeX, ridgeY, alpha = 0, family = "binomial")
ridgeBestLambda <- ridgeCVModel$lambda.min
ridgeBestLambda
```

```
## [1] 0.03982389
```

```
# Run Ridge Regression algorithm
ridgeModel <- glmnet(ridgeX, ridgeY, alpha = 0, lambda = ridgeBestLambda,
```

```
    family = "binomial")
coef(ridgeModel)
```

```
## 25 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -3.228125454
## X.age.      -0.014244369
## X.bp.       -0.008794322
## X.sg.        0.366075803
## X.al.       -0.518088940
## X.su.       -0.392029998
## X.rbc.       0.982154317
## X.pc.        0.604612792
## X.pcc.      -0.826046303
## X.ba.       -0.135234334
## X.bgr.      -0.004193946
## X.bu.       -0.005220384
## X.sc.       -0.069136878
## X.sod.       0.041029502
## X.pot.       0.028868176
## X.hemo.      0.118697286
## X.pcv.       0.033245605
## X.wbcc.     -0.000128641
## X.rbcc.      0.124670032
## X.htn.      -0.888780335
## X.dm.       -0.663355748
## X.cad.      -0.248975776
## X.appet.    -0.337182104
## X.pe.       -0.280448159
## X.ane.      -0.221139458
```

3.2.3 Model Checking

Before validating our models, we must check our assumptions.

1. **Binary Response**. Our dependent variable must be a categorical nominal variable with two levels.

```
table(formattedData$X.class.)
```

```
##
##    ckd notckd
##     43    114
```

Our assumption is met.

2. **Independence of Observations**. Our observations need to be independent from one another. Intuitively, one patient being diagnosed with CKD does not conceivably influence whether or not another patient is diagnosed with CKD. The inverse of this statement also is reasonably (i.e., a patient being diagnosed as healthy (without CKD) does not influence another patient being diagnosed as healthy). Our assumption is met.

3. **Multicollinearity**. The regressors of our models should not exhibit high amounts of multicollinearity between each other. Because the Forward and Backwards Models were generated from non-penalizing regression methods, we will explicitly check for multicollinearity using Variance Inflation Factors (VIF) which can be computed from the `vif()` function in the `car` package.[3]

```r
# Load car package
library(car)
```

```
## Loading required package: carData
```

```r
# Multicollinearity Detection
vif(forwardModel)
```

```
##  X.hemo.   X.bgr.
## 78.68015 78.68015
```

```r
vif(backwardModel)
```

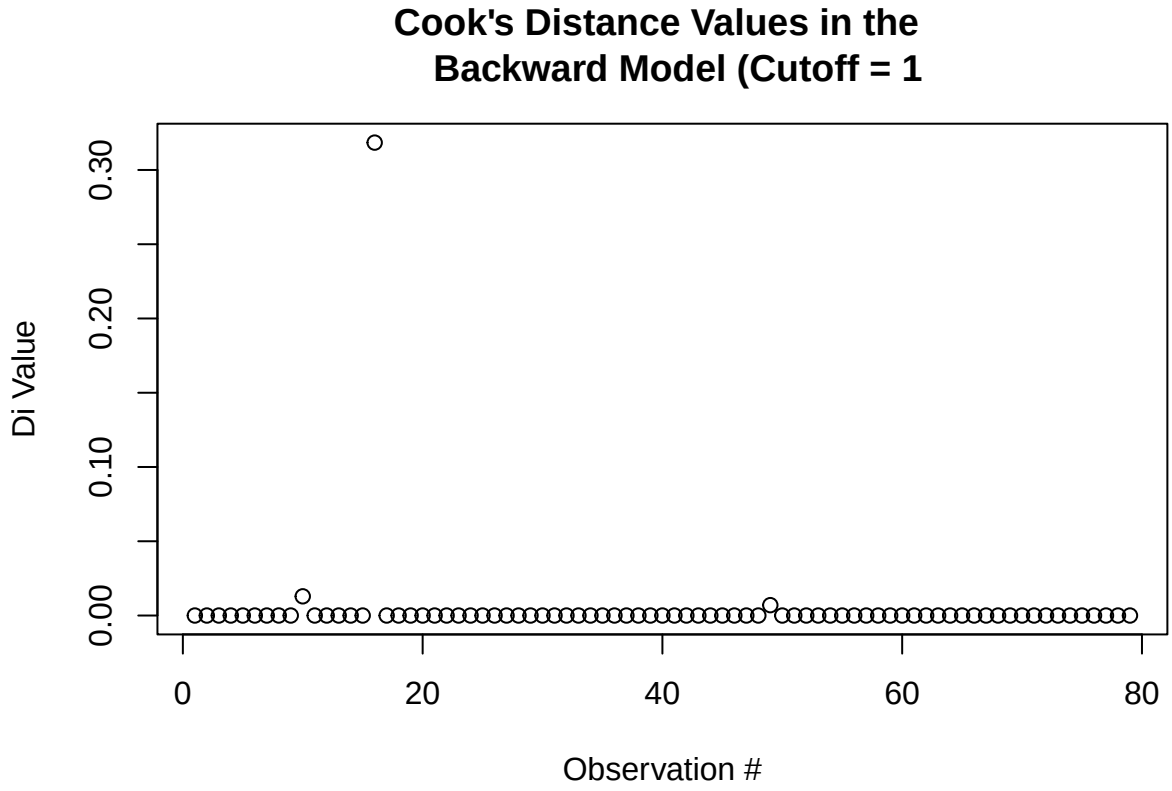
```
##   X.bgr.    X.bu.  X.wbcc.
## 2.134539 1.192654 2.089410
```

The Forward Model exhibits VIF scores over 10 for both regressors, indicating a serious multicollinearity issue with the model. The Backward Model exhibits VIF scores under 5 for each regressor, indicating a acceptable amount of correlation between the regressors. We will abandon the Forward Model and retain the other models.

4. **Outliers**. Our model should not contain any extreme outliers or high influence points (HIP). We will check for outliers and HIPs using Cook's Distance and discard those observations from both models if they are found.

```r
# Cook's Distance Plot Backward Model
plot(cooks.distance(backwardModel), main = "Cook's Distance Values in the
     Backward Model (Cutoff = 1", xlab = "Observation #", ylab = "Di Value")
```

```
## Warning in title(...): font width unknown for character 0x9
```



There are no observations with a D_i value greater than 1. Therefore, there are no HIPs in this model. Our

assumption is met.

5. **Linearity Between Logit of Response and Regressor**. For each regressor in our model, there needs to be a linear relationship between the logit of the response and the explanatory variable. We will check for linearity by examining a scatterplot of the log-odds versus the regressor.

```
# Linearity Check for the Backward Model
backwardModelLogOdds <- backwardModel$linear.predictors
cor(trainingData$X.bgr., backwardModelLogOdds)
```

```
## [1] -0.6116955
```

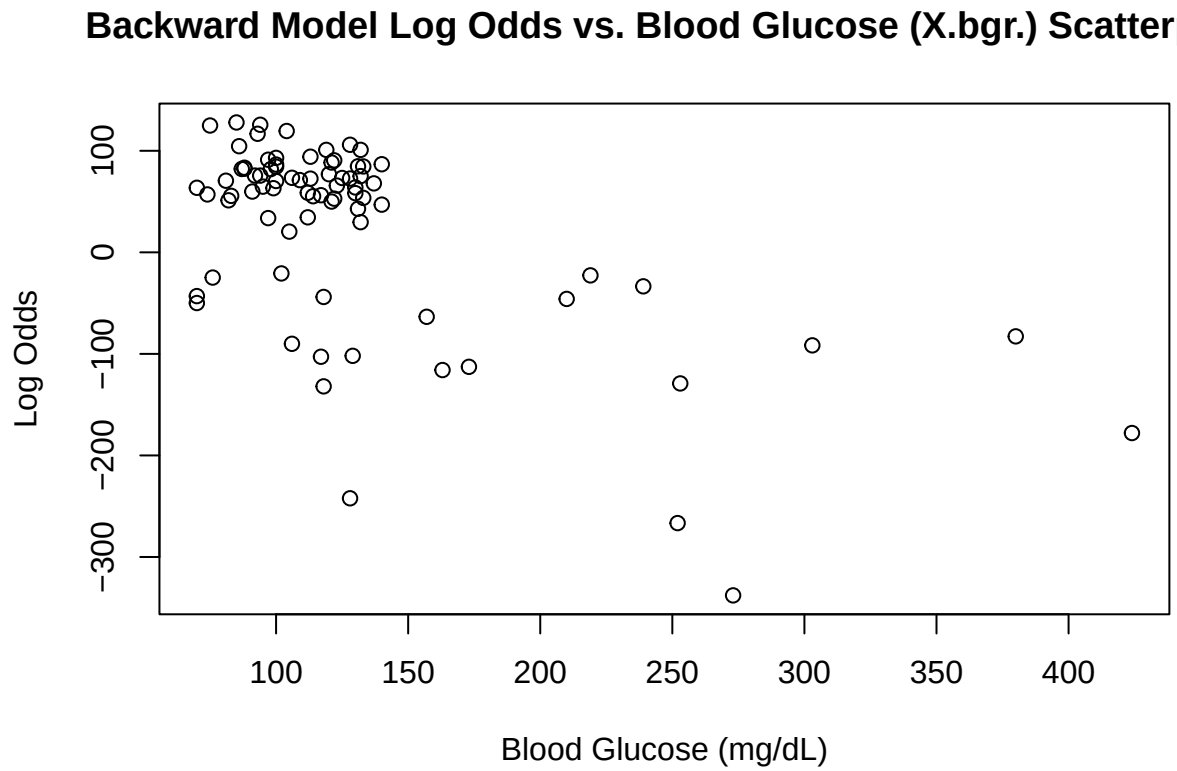
```
cor(trainingData$X.bu., backwardModelLogOdds)
```

```
## [1] -0.7385213
```

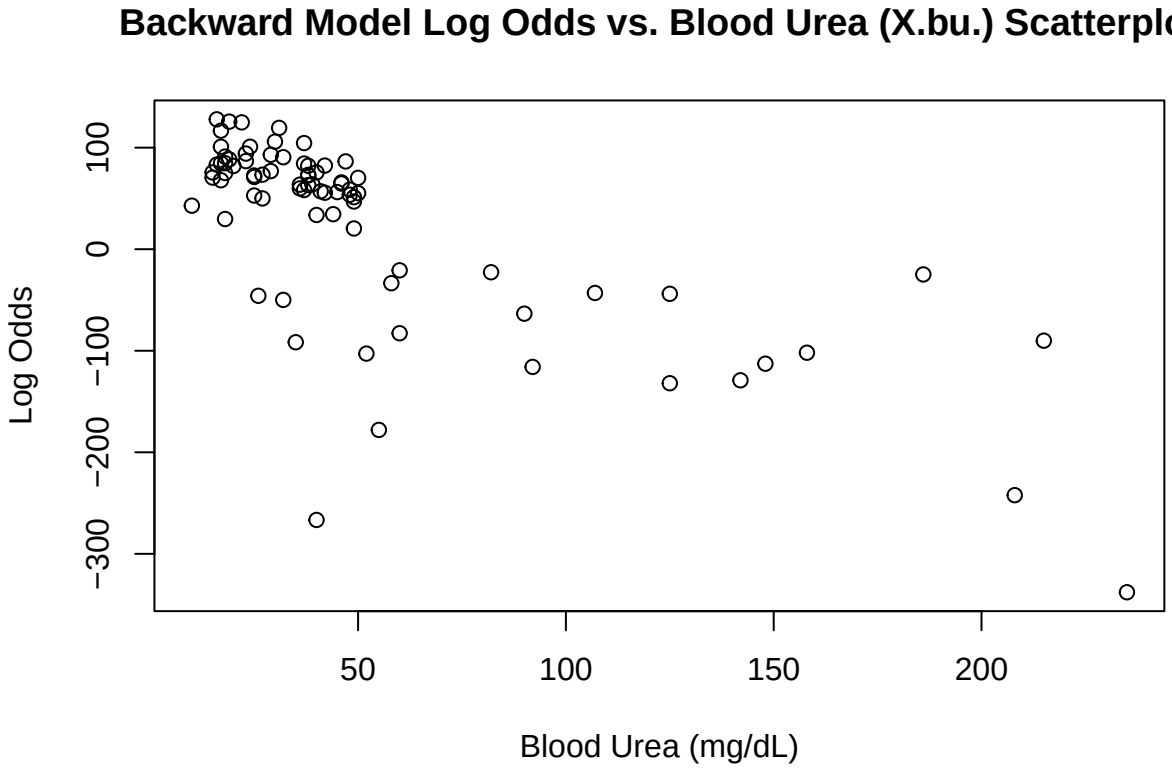
```
cor(trainingData$X.wbcc., backwardModelLogOdds)
```

```
## [1] -0.6763626
```

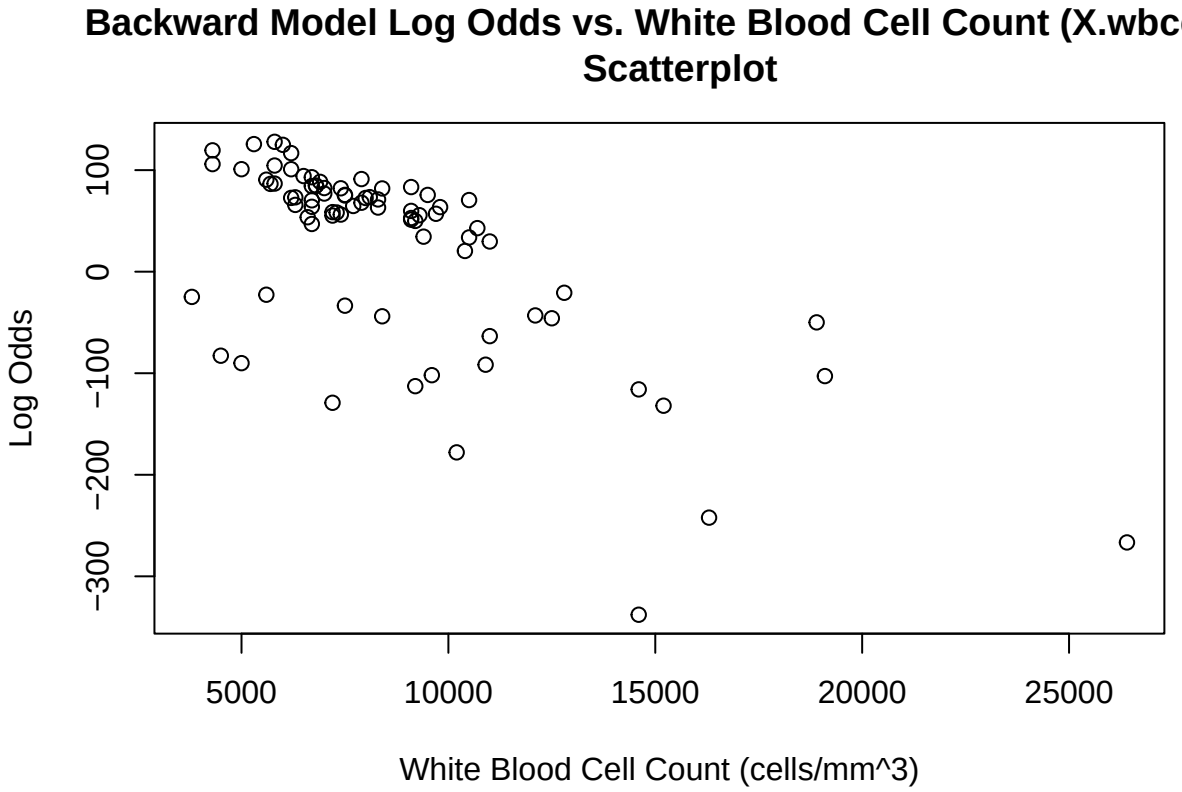
```
plot(trainingData$X.bgr., backwardModelLogOdds,
     main = "Backward Model Log Odds vs. Blood Glucose (X.bgr.) Scatterplot",
        xlab = "Blood Glucose (mg/dL)", ylab = "Log Odds")
```




```
plot(trainingData$X.bu., backwardModelLogOdds,
     main = "Backward Model Log Odds vs. Blood Urea (X.bu.) Scatterplot",
        xlab = "Blood Urea (mg/dL)", ylab = "Log Odds")
```




```
plot(trainingData$X.wbcc., backwardModelLogOdds,
     main = "Backward Model Log Odds vs. White Blood Cell Count (X.wbcc.)
        Scatterplot", xlab = "White Blood Cell Count (cells/mm^3)",
            ylab = "Log Odds")
```



The Backward model's log odds have a strong negative linear relationship with the blood glucose and blood urea regressors. The model's log odds have a moderate negative linear relationship with the white blood cell count regressor. Our assumption is met for this model.

6. **Sample Size**. We require $\frac{(10 \times k)}{P(x)}$ number of observations where k is the number of regressors and $P(x)$ is the expected probability of the least frequent outcome in the dataset.

```
# Sample Size Check

table(trainingData$X.class.)
```

```
##
##    ckd notckd
##     22     57
```

```
(10 * 4) / (22 / 79) # 4 regressors, 22 / 79 CKD-classified observations
```

```
## [1] 143.6364
```

Our training dataset contains only 79 observations. Thus our assumption will not be met. We will proceed anyways as our models have passed all other critera for logistic regression.

3.3 Validation

Using our testing dataset, we will collect statistics such as sensitivity, specificity, positive predictive value and negative predictive value on the classification models.

3.3.1 Backward Model

We will utilize the \texttt{%>%} (pipe) operator from the `dplyr` package.

```
# Load the dplyr package
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:car':
##
##     recode
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Run the Backward Model on the test dataset
backwardModelProbabilities <- backwardModel %>% predict(testingData,
    type = "response")
backwardModelPredictedClasses <- ifelse(backwardModelProbabilities < 0.5,
    "ckd", "notckd")
```

3.3.2 LASSO Model

```
# Testing data matrix
testDataMatrix <- data.matrix(testingData[, 2:25])
```

```
# Run the LASSO Model on the test dataset
lassoModelPredictedProbabilities <- predict(lassoModel, s = lassoBestLambda,
    newx = testDataMatrix, type = "response")
lassoModelPredictedClasses <- ifelse(lassoModelPredictedProbabilities < 0.5,
    "ckd", "notckd")
```

3.3.3 Ridge Regression Model

```
ridgeModelPredictedProbabilities <- predict(ridgeModel, s = ridgeBestLambda,
    newx = testDataMatrix, type = "response")
ridgeModelPredictedClasses <- ifelse(ridgeModelPredictedProbabilities < 0.5,
    "ckd", "notckd")
```

4 Results

We will use the caret package to generate a confusion matrix for each of the models to collect sensitivity, specificity, positive predictive value (PPV) and negative predictive value (NPV).[6]

```
# Load the caret package
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

4.1 Backward Model

```
confusionMatrix(table(backwardModelPredictedClasses, testingData$X.class.))
```

```
## Confusion Matrix and Statistics
##
##
## backwardModelPredictedClasses ckd notckd
##                         ckd     16      0
##                         notckd   5     57
##
##                Accuracy : 0.9359
##                  95% CI : (0.8567, 0.9789)
##     No Information Rate : 0.7308
##     P-Value [Acc > NIR] : 4.121e-06
##
##                   Kappa : 0.8238
##
##  Mcnemar's Test P-Value : 0.07364
##
##             Sensitivity : 0.7619
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9194
##              Prevalence : 0.2692
##          Detection Rate : 0.2051
##    Detection Prevalence : 0.2051
##       Balanced Accuracy : 0.8810
```

```
##
##        'Positive' Class : ckd
##
```

The Backward Model has an accuracy of 93.59%. It is able to correctly return a "ckd" classification, given that a patient does actually have CKD, 76.19% of the time and correctly return a "notckd" classification, given that a patient does not actually have CKD, 100% of the time. A patient has a 100% chance of having CKD given that the model returns a "ckd" classification for their biometrics and a 91.94% chance of not having CKD given that the model returns a "notckd" classification for their biometrics.

4.2 LASSO Model

```
confusionMatrix(table(lassoModelPredictedClasses, testingData$X.class.))
```

```
## Confusion Matrix and Statistics
##
##
## lassoModelPredictedClasses ckd notckd
##                      ckd     21      0
##                      notckd   0     57
##
##                Accuracy : 1
##                  95% CI : (0.9538, 1)
##     No Information Rate : 0.7308
##     P-Value [Acc > NIR] : 2.371e-11
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.2692
##          Detection Rate : 0.2692
##    Detection Prevalence : 0.2692
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : ckd
##
```

The LASSO Model has an accuracy of 100%. It is able to correctly return a "ckd" classification, given that a patient does actually have CKD, 100% of the time and correctly return a "notckd" classification, given that a patient does not actually have CKD, 100% of the time. A patient has a 100% chance of having CKD given that the model returns a "ckd" classification for their biometrics and a 100% chance of not having CKD given that the model returns a "notckd" classification for their biometrics.

4.3 Ridge Regression Model

```
confusionMatrix(table(ridgeModelPredictedClasses, testingData$X.class.))
```

```
## Confusion Matrix and Statistics
##
##
```

```
## ridgeModelPredictedClasses ckd notckd
##                      ckd     21      0
##                      notckd   0     57
##
##                Accuracy : 1
##                  95% CI : (0.9538, 1)
##     No Information Rate : 0.7308
##     P-Value [Acc > NIR] : 2.371e-11
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.2692
##          Detection Rate : 0.2692
##    Detection Prevalence : 0.2692
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : ckd
##
```

The Ridge Regresion Model has an accuracy of 100%. It is able to correctly return a "ckd" classification, given that a patient does actually have CKD, 100% of the time and correctly return a "notckd" classification, given that a patient does not actually have CKD, 100% of the time. A patient has a 100% chance of having CKD given that the model returns a "ckd" classification for their biometrics and a 100% chance of not having CKD given that the model returns a "notckd" classification for their biometrics.

5 Discussion & Conclusion

The most optimal model is the model that maximizes its accuracy, sensitivity, specificity, PPV and NPV. Based on this criteria, the LASSO and Ridge Regresion Models are the best models for classifying whether or not a patient has CKD. Additionally, the Backward Model is a strong model, demonstrating high accuracy, specificity, PPV and NPV. In the future, this analysis should be repeated using larger sample sizes. One potential re-approach to this study could involve the use of Generative Adversarial Neural Networks (GAN). GANs allow for the creation of new, but similar and useful data from random noise which can be used to handle issues involving low sample sizes.[7]

References

[1] *Chronic Kidney Disease in the United States, 2021*. 2021. URL: https://www.cdc.gov/kidneydisease/publications-resources/ckd-national-facts.html.

[2] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[3] John Fox and Sanford Weisberg. *An R Companion to Applied Regression*. 2019. URL: https://socialsciences.mcmaster.ca/jfox/Books/Companion/.

[4] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1 (2010), pp. 1–22. DOI: 10.18637/jss.v033.i01. URL: https://www.jstatsoft.org/v33/i01/.

[5] Nelson P. Kopyt. "Chronic Kidney Disease: The New Silent Killer". In: *Journal of Osteopathic Medicine* 106.3 (2006), pp. 133–136. DOI: doi:10.7556/jaoa.2006.106.3.133.

[6] Max Kuhn. "Building Predictive Models in R Using the caret Package". In: *Journal of Statistical Software* 28.i05 (2008). DOI: http://hdl.handle.net/10..

[7] Pawan Saxena. *Synthetic Data Generation Using Conditional-GAN*. Towards Data Science. 2021. URL: https://towardsdatascience.com/synthetic-data-generation-using-conditional-gan-45f91542ec6b.

[8] Elisabetta Versino and Giorgina Barbara Piccoli. "Chronic kidney disease: The complex history of the organization of long-term care and bioethics. Why now, more than ever, action is needed". In: *Int. J. Environ. Res. Public Health* 16.5 (2019).