Performance Comparison of Machine Learning Algorithms to Predict Chronic Kidney Disease

DA5030

Remala,Subbaramireddy

2024-12-01, Fall 2024

Contents

Introduction

The project aims to leverage advanced Machine Learning techniques to predict the onset of CKD. Given the subtle progression of CKD, early detection is crucial for effective intervention and prevention of kidney failure. This project will compare various machine learning algorithms to assess their accuracy and efficiency in predicting CKD based on available clinical data, including lab tests, medical history, and demographic factors. By analyzing the performance of these algorithms, the project seeks to identify the most effective model for early diagnosis, helping clinicians make timely decisions in patient care.

Purpose of Analysis

Purpose of this analysis is to evaluate and compare the performance of various machine learning algorithms in predicting the onset of Chronic Kidney Disease (CKD) using clinical data such as laboratory test results, medical history, and demographic factors. By identifying the most accurate and efficient model for early CKD detection, the project aims to facilitate timely interventions and improve patient outcomes, thereby reducing the risk of kidney failure and its associated complications.

Dataset Overview

The dataset is designed for the prediction of Chronic Kidney Disease (CKD) and consists of 400 instances with 24 features plus a target variable (class), which indicates whether a patient has CKD or not. The features are a mix of 11 numeric and 14 nominal variables, encompassing demographic information, clinical test results, and medical history. These include attributes such as age, blood pressure, blood glucose, serum creatinine, hemoglobin, and binary indicators for conditions like hypertension, diabetes, and anemia. Additionally, features such as specific gravity, albumin, pus cell clumps, and bacteria are categorical or binary in nature. The dataset contains missing values, making data preprocessing essential. The target variable (class) is binary, representing CKD (ckd) or not CKD (notckd), and the data supports classification tasks, aiming to leverage clinical insights for early CKD detection and improved patient care(Rubini, Soundarapandian, and Eswaran 2015).

Dataset Source

The source of this dataset is available from the UCI Machine Learning Repository. It was contributed by Rubini K. and can be accessed via the following link: UCI Machine Learning Repository - Chronic Kidney Disease Dataset- https://archive.ics.uci.edu/dataset/336/chronic+kidney+disease .

1. Date Preprocessing and Exploration

```
# loading required libraries
library(httr)
library(farff)
```

```r
library(ggcorrplot)
library(caret)
library(e1071)
library(pROC)
library(glmnet)
library(nnet)
library(gridExtra)
library(FactoMineR)
library(vcd)
library(factoextra)
```

1.1 Data Acquisition

```r
# Define the Google Drive URL
url <- "https://drive.google.com/uc?export=download&id=15K4XjrVVsEoD4HMEvFz3vYQN2EMpXr_R"

# Send a request to the Google Drive URL
response <- GET(url)

# Saving the downloaded content to a temporary file
temp_file <- tempfile(fileext = ".arff")
writeBin(content(response, "raw"), temp_file)

# Read the ARFF file from the temporary location
ckd_data <- readARFF(temp_file)

# Reneaming the columns of the dataset for clarity
names(ckd_data) = c("age", "blood_pressure", "specific_gravity", "albumin", "sugar", "rbc",
                    "pus_cell", "pus_cell_clumps", "Bacteria", "blood_glucose_random", "blood_urea",
                    "serum_creatinine", "sodium", "potassium", "hemoglobin", "packed_cell_volume",
                    "wbc_count", "rbc_count", "hypertension", "diabetes_mellitus",
                    "cad", "appetite", "peda_edema", "anemia", "class")

# first few rows of the dataset
head(ckd_data)
```

```
##   age blood_pressure specific_gravity albumin sugar    rbc pus_cell
## 1  48             80           1.020       1     0   <NA>   normal
## 2   7             50           1.020       4     0   <NA>   normal
## 3  62             80           1.010       2     3 normal   normal
## 4  48             70           1.005       4     0 normal abnormal
## 5  51             80           1.010       2     0 normal   normal
## 6  60             90           1.015       3     0   <NA>     <NA>
##   pus_cell_clumps   Bacteria blood_glucose_random blood_urea serum_creatinine
## 1      notpresent notpresent                  121         36              1.2
## 2      notpresent notpresent                   NA         18              0.8
## 3      notpresent notpresent                  423         53              1.8
## 4         present notpresent                  117         56              3.8
## 5      notpresent notpresent                  106         26              1.4
## 6      notpresent notpresent                   74         25              1.1
##   sodium potassium hemoglobin packed_cell_volume wbc_count rbc_count
```

```
## 1     NA        NA       15.4                44      7800       5.2
## 2     NA        NA       11.3                38      6000        NA
## 3     NA        NA        9.6                31      7500        NA
## 4    111       2.5       11.2                32      6700       3.9
## 5     NA        NA       11.6                35      7300       4.6
## 6    142       3.2       12.2                39      7800       4.4
##   hypertension diabetes_mellitus cad appetite peda_edema anemia class
## 1          yes               yes  no     good         no     no   ckd
## 2           no                no  no     good         no     no   ckd
## 3           no               yes  no     poor         no    yes   ckd
## 4          yes                no  no     poor        yes    yes   ckd
## 5           no                no  no     good         no     no   ckd
## 6          yes               yes  no     good        yes     no   ckd
```

```
# summary of the dataset
summary(ckd_data)
```

```
##       age        blood_pressure   specific_gravity albumin     sugar
##  Min.   : 2.00   Min.   : 50.00   1.005:  7        0   :199   0   :290
##  1st Qu.:42.00   1st Qu.: 70.00   1.010: 84        1   : 44   1   : 13
##  Median :55.00   Median : 80.00   1.015: 75        2   : 43   2   : 18
##  Mean   :51.48   Mean   : 76.47   1.020:106        3   : 43   3   : 14
##  3rd Qu.:64.50   3rd Qu.: 80.00   1.025: 81        4   : 24   4   : 13
##  Max.   :90.00   Max.   :180.00   NA's : 47        5   :  1   5   :  3
##  NA's   :9       NA's   :12                        NA's: 46   NA's: 49
##        rbc          pus_cell     pus_cell_clumps       Bacteria
##  normal  :201   normal  :259   present   : 42    present   : 22
##  abnormal: 47   abnormal: 76   notpresent:354    notpresent:374
##  NA's    :152   NA's    : 65   NA's      :  4    NA's      :  4
##
##
##
##
##  blood_glucose_random   blood_urea     serum_creatinine     sodium
##  Min.   : 22          Min.   :  1.50   Min.   : 0.400   Min.   :  4.5
##  1st Qu.: 99          1st Qu.: 27.00   1st Qu.: 0.900   1st Qu.:135.0
##  Median :121          Median : 42.00   Median : 1.300   Median :138.0
##  Mean   :148          Mean   : 57.43   Mean   : 3.072   Mean   :137.5
##  3rd Qu.:163          3rd Qu.: 66.00   3rd Qu.: 2.800   3rd Qu.:142.0
##  Max.   :490          Max.   :391.00   Max.   :76.000   Max.   :163.0
##  NA's   :44           NA's   :19       NA's   :17       NA's   :87
##    potassium        hemoglobin    packed_cell_volume   wbc_count
##  Min.   : 2.500   Min.   : 3.10   Min.   : 9.00      Min.   : 2200
##  1st Qu.: 3.800   1st Qu.:10.30   1st Qu.:32.00      1st Qu.: 6500
##  Median : 4.400   Median :12.65   Median :40.00      Median : 8000
##  Mean   : 4.627   Mean   :12.53   Mean   :38.87      Mean   : 8414
##  3rd Qu.: 4.900   3rd Qu.:15.00   3rd Qu.:45.00      3rd Qu.: 9800
##  Max.   :47.000   Max.   :17.80   Max.   :54.00      Max.   :26400
##  NA's   :88       NA's   :52      NA's   :72         NA's   :108
##    rbc_count     hypertension diabetes_mellitus   cad      appetite
##  Min.   :2.100   yes :147     yes :135          yes : 34   good:316
##  1st Qu.:3.900   no  :251     no  :257          no  :362   poor: 82
##  Median :4.800   NA's:  2     NA's:  8          NA's:  4   NA's:  2
##  Mean   :4.707
```

```
##  3rd Qu.:5.400
##  Max.   :8.000
##  NA's   :131
##  peda_edema  anemia       class
##  yes : 76   yes : 60   ckd   :246
##  no  :322   no  :339   notckd:149
##  NA's:  2   NA's:  1   NA's  :  5
##
##
##
##
```

```
# checking the structure
str(ckd_data)
```

```
## 'data.frame':    400 obs. of  25 variables:
##  $ age                 : num  48 7 62 48 51 60 68 24 52 53 ...
##  $ blood_pressure      : num  80 50 80 70 80 90 70 NA 100 90 ...
##  $ specific_gravity    : Factor w/ 5 levels "1.005","1.010",..: 4 4 2 1 2 3 2 3 3 4 ...
##  $ albumin             : Factor w/ 6 levels "0","1","2","3",..: 2 5 3 5 3 4 1 3 4 3 ...
##  $ sugar               : Factor w/ 6 levels "0","1","2","3",..: 1 1 4 1 1 1 1 5 1 1 ...
##  $ rbc                 : Factor w/ 2 levels "normal","abnormal": NA NA 1 1 1 NA NA 1 1 2 ...
##  $ pus_cell            : Factor w/ 2 levels "normal","abnormal": 1 1 1 2 1 NA 1 2 2 2 ...
##  $ pus_cell_clumps     : Factor w/ 2 levels "present","notpresent": 2 2 2 1 2 2 2 2 1 1 ...
##  $ Bacteria            : Factor w/ 2 levels "present","notpresent": 2 2 2 2 2 2 2 2 2 2 ...
##  $ blood_glucose_random: num  121 NA 423 117 106 74 100 410 138 70 ...
##  $ blood_urea          : num  36 18 53 56 26 25 54 31 60 107 ...
##  $ serum_creatinine    : num  1.2 0.8 1.8 3.8 1.4 1.1 24 1.1 1.9 7.2 ...
##  $ sodium              : num  NA NA NA 111 NA 142 104 NA NA 114 ...
##  $ potassium           : num  NA NA NA 2.5 NA 3.2 4 NA NA 3.7 ...
##  $ hemoglobin          : num  15.4 11.3 9.6 11.2 11.6 12.2 12.4 12.4 10.8 9.5 ...
##  $ packed_cell_volume  : num  44 38 31 32 35 39 36 44 33 29 ...
##  $ wbc_count           : num  7800 6000 7500 6700 7300 7800 NA 6900 9600 12100 ...
##  $ rbc_count           : num  5.2 NA NA 3.9 4.6 4.4 NA 5 4 3.7 ...
##  $ hypertension        : Factor w/ 2 levels "yes","no": 1 2 2 1 2 1 2 2 1 1 ...
##  $ diabetes_mellitus   : Factor w/ 2 levels "yes","no": 1 2 1 2 2 1 2 1 1 1 ...
##  $ cad                 : Factor w/ 2 levels "yes","no": 2 2 2 2 2 2 2 2 2 2 ...
##  $ appetite            : Factor w/ 2 levels "good","poor": 1 1 2 2 1 1 1 1 1 2 ...
##  $ peda_edema          : Factor w/ 2 levels "yes","no": 2 2 2 1 2 1 2 1 2 2 ...
##  $ anemia              : Factor w/ 2 levels "yes","no": 2 2 1 1 2 2 2 2 1 1 ...
##  $ class               : Factor w/ 2 levels "ckd","notckd": 1 1 1 1 1 1 1 1 1 1 ...
```

1.2 Data Cleaning, and Immputation

```
# counting the number of missing (NA) values in each column of the dataset
colSums(is.na(ckd_data))
```

```
##                 age       blood_pressure     specific_gravity
##                   9                   12                   47
##             albumin                sugar                  rbc
##                  46                   49                  152
```

```
##             pus_cell      pus_cell_clumps             Bacteria
##                   65                    4                    4
## blood_glucose_random          blood_urea     serum_creatinine
##                   44                   19                   17
##               sodium            potassium           hemoglobin
##                   87                   88                   52
##   packed_cell_volume           wbc_count           rbc_count
##                   72                  108                  131
##         hypertension    diabetes_mellitus                  cad
##                    2                    8                    4
##             appetite          peda_edema               anemia
##                    2                    2                    1
##                class
##                    5
```

Handling missing values for numeric variables and all Binary class null values

```
# # Convert selected columns to numeric type (specific_gravity, albumin, sugar)
ckd_data[c("specific_gravity", "albumin", "sugar")] <- lapply(ckd_data[c("specific_gravity", "albumin",

# defining a vector of numeric variables that may have missing values
numeric_vars <- c("age", "blood_pressure","specific_gravity", "albumin", "sugar", "blood_glucose_random
                  "serum_creatinine","sodium","potassium", "hemoglobin", "packed_cell_volume", "wbc_cou

# looping through each numeric variable and fill missing values with the median
for (var in numeric_vars) {
  ckd_data[[var]][is.na(ckd_data[[var]])] <- median(ckd_data[[var]], na.rm = TRUE)
}
# counting the number of missing values (NA) in each column after filling
colSums(is.na(ckd_data))
```

```
##                  age       blood_pressure     specific_gravity
##                    0                    0                    0
##              albumin                sugar                  rbc
##                    0                    0                  152
##             pus_cell      pus_cell_clumps             Bacteria
##                   65                    4                    4
## blood_glucose_random          blood_urea     serum_creatinine
##                    0                    0                    0
##               sodium            potassium           hemoglobin
##                    0                    0                    0
##   packed_cell_volume           wbc_count           rbc_count
##                    0                    0                    0
##         hypertension    diabetes_mellitus                  cad
##                    2                    8                    4
##             appetite          peda_edema               anemia
##                    2                    2                    1
##                class
##                    5
```

Handling missing values for categorical variables except class variable

```
# # function to calculate the mode of a variable
get_mode <- function(x) {
  uniq_x <- unique(x)
  uniq_x[which.max(tabulate(match(x, uniq_x)))]
}

# Define the numeric variables
numeric_vars <- c("rbc", "pus_cell", "pus_cell_clumps", "Bacteria", "hypertension", "diabetes_mellitus"
                  "cad", "appetite", "peda_edema", "anemia")

# defining the categorical variables that need mode imputation
for (var in numeric_vars) {
  mode_value <- get_mode(ckd_data[[var]][!is.na(ckd_data[[var]])])  # Exclude NAs when calculating mode
  ckd_data[[var]][is.na(ckd_data[[var]])] <- mode_value
}

# counting the number of missing values (NA) in each column after imputation
colSums(is.na(ckd_data))
```

```
##                   age       blood_pressure     specific_gravity
##                     0                    0                    0
##               albumin                sugar                  rbc
##                     0                    0                    0
##             pus_cell      pus_cell_clumps             Bacteria
##                     0                    0                    0
## blood_glucose_random           blood_urea     serum_creatinine
##                     0                    0                    0
##                sodium            potassium           hemoglobin
##                     0                    0                    0
##   packed_cell_volume            wbc_count            rbc_count
##                     0                    0                    0
##         hypertension    diabetes_mellitus                  cad
##                     0                    0                    0
##             appetite           peda_edema               anemia
##                     0                    0                    0
##                class
##                    5
```

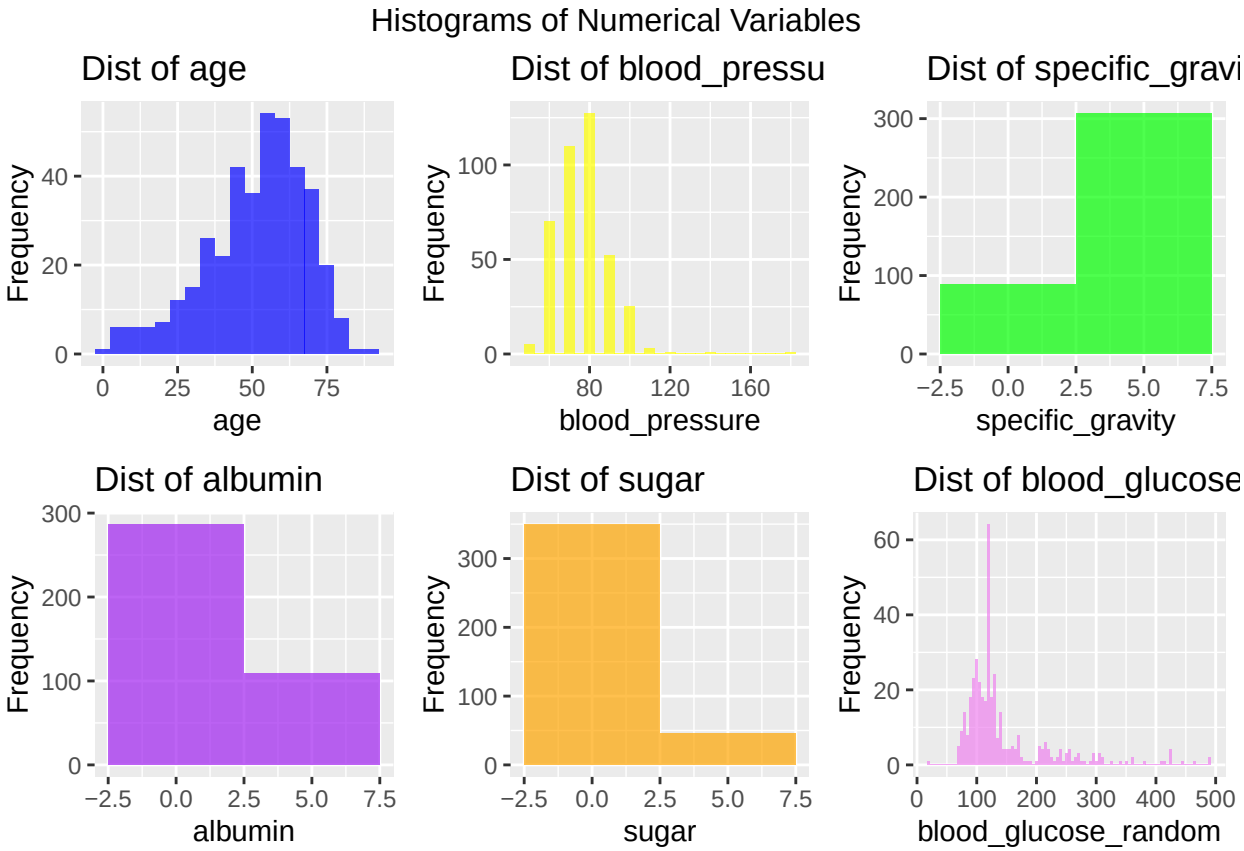
```
# Remove rows with any missing (NA) values from the dataset, i.e only class variable is left
ckd_data <- na.omit(ckd_data)

# Count the number of missing values (NA) in each column after removing rows with NAs for class variabl
colSums(is.na(ckd_data))
```

```
##                   age       blood_pressure     specific_gravity
##                     0                    0                    0
##               albumin                sugar                  rbc
##                     0                    0                    0
##             pus_cell      pus_cell_clumps             Bacteria
##                     0                    0                    0
```

```
## blood_glucose_random           blood_urea     serum_creatinine
##                    0                    0                    0
##                sodium            potassium           hemoglobin
##                    0                    0                    0
##   packed_cell_volume            wbc_count            rbc_count
##                    0                    0                    0
##         hypertension    diabetes_mellitus                  cad
##                    0                    0                    0
##             appetite          peda_edema               anemia
##                    0                    0                    0
##                class
##                    0
```

1.3 Exploratory Data Plots

```
# Histograms of Numerical Variables
plot_age <- ggplot(ckd_data, aes(x = age)) +
  geom_histogram(binwidth = 5, fill = "blue", alpha = 0.7) +
  labs(title = "Dist of age", x = "age", y = "Frequency")

plot_bp <- ggplot(ckd_data, aes(x = blood_pressure)) +
  geom_histogram(binwidth = 5, fill = "yellow", alpha = 0.7) +
  labs(title = "Dist of blood_pressure", x = "blood_pressure", y = "Frequency")

plot_sg <- ggplot(ckd_data, aes(x = specific_gravity)) +
  geom_histogram(binwidth = 5, fill = "green", alpha = 0.7) +
  labs(title = "Dist of specific_gravity", x = "specific_gravity", y = "Frequency")

plot_alb <- ggplot(ckd_data, aes(x = albumin)) +
  geom_histogram(binwidth = 5, fill = "purple", alpha = 0.7) +
  labs(title = "Dist of albumin", x = "albumin", y = "Frequency")

plot_sug <- ggplot(ckd_data, aes(x = sugar)) +
  geom_histogram(binwidth = 5, fill = "orange", alpha = 0.7) +
  labs(title = "Dist of sugar", x = "sugar", y = "Frequency")

plot_bgr <- ggplot(ckd_data, aes(x = blood_glucose_random)) +
  geom_histogram(binwidth = 5, fill = "violet", alpha = 0.7) +
  labs(title = "Dist of blood_glucose_random", x = "blood_glucose_random", y = "Frequency")

# Use grid.arrange to arrange the plots
grid.arrange(plot_age, plot_bp, plot_sg,
             plot_alb, plot_sug, plot_bgr,
             ncol = 3,
             top = "Histograms of Numerical Variables")
```



The above histograms shows the distributions of numerical variables in a ckd dataset.

1. **Age**: The distribution is roughly bell-shaped, with a peak around 25-40 years, indicating a concentration of individuals in that age range. There are also a few individuals in the older age group, extending up to 75 years.

2. **Blood Pressure**: This distribution is skewed towards the lower range, with the majority of the values between 50 and 80, and a smaller frequency of values above 80. There are few extreme values, indicating the presence of outliers or possible measurement anomalies.

3. **Specific Gravity**: Histogram shows a highly concentrated distribution, with most values near 1.0. There are very few data points that deviate from this range, suggesting a narrow distribution.

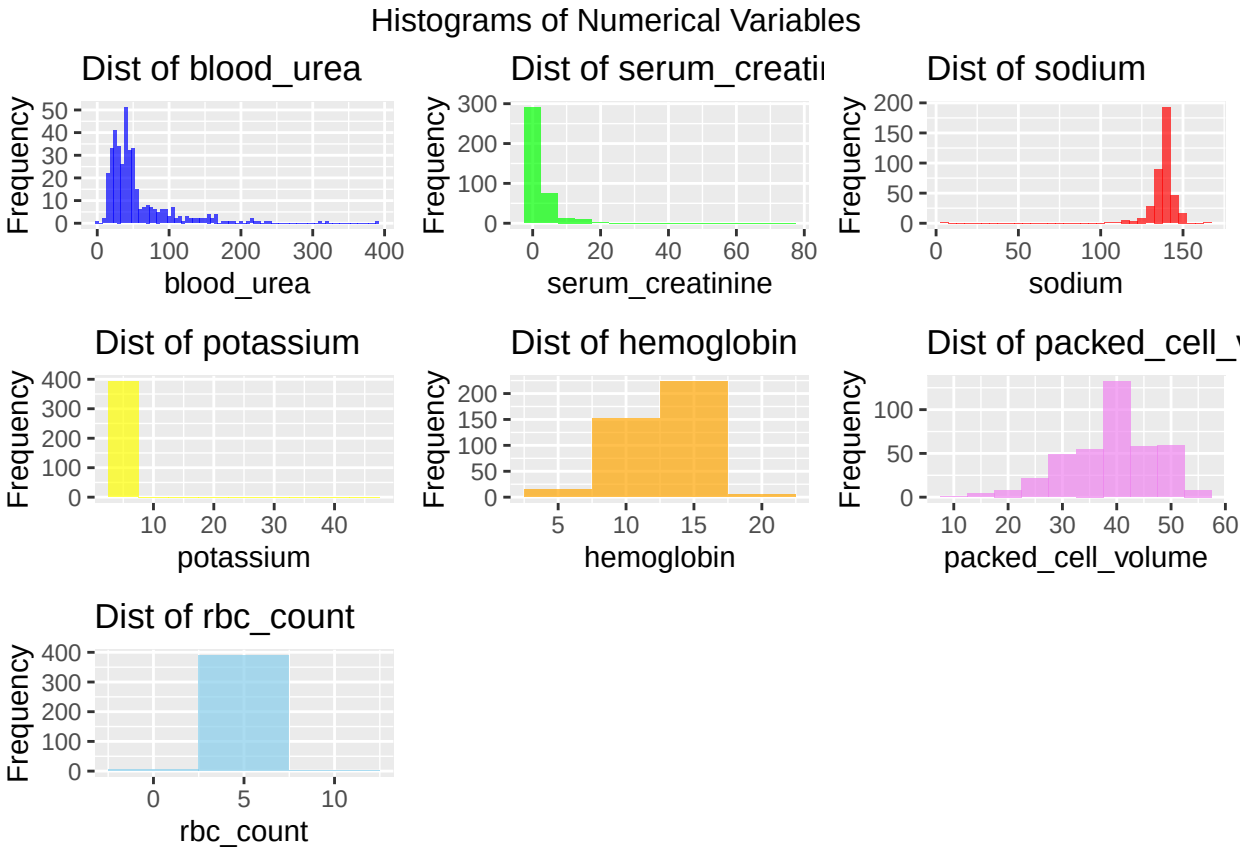
4. **Albumin**: The distribution for albumin shows a sharp peak at low values, indicating most values are clustered around zero, with very few observations extending into positive values.

5. **Sugar**: The distribution is heavily skewed towards low values, with almost all data points concentrated at 0. This suggests that the majority of the individuals have very low or no sugar present in their data, with a few higher values.

6. **Blood Glucose Random** : This histogram is skewed towards higher values, with a notable peak around 100-200. There are several high-frequency occurrences at lower values and a tail extending towards larger values, indicating some possible outliers or variance.

Overall, the dataset exhibits significant skewness in most of the variables, with certain variables showing concentrations around specific ranges while others exhibit outliers.

```r
# Histograms of Numerical Variables
plot_bu <- ggplot(ckd_data, aes(x = blood_urea)) +
  geom_histogram(binwidth = 5, fill = "blue", alpha = 0.7) +
  labs(title = "Dist of blood_urea", x = "blood_urea", y = "Frequency")

plot_sc <- ggplot(ckd_data, aes(x = serum_creatinine)) +
  geom_histogram(binwidth = 5, fill = "green", alpha = 0.7) +
  labs(title = "Dist of serum_creatinine", x = "serum_creatinine", y = "Frequency")

plot_sod <- ggplot(ckd_data, aes(x = sodium)) +
  geom_histogram(binwidth = 5, fill = "red", alpha = 0.7) +
  labs(title = "Dist of sodium", x = "sodium", y = "Frequency")

plot_pot <- ggplot(ckd_data, aes(x = potassium)) +
  geom_histogram(binwidth = 5, fill = "yellow", alpha = 0.7) +
  labs(title = "Dist of potassium", x = "potassium", y = "Frequency")

plot_hemo <- ggplot(ckd_data, aes(x = hemoglobin)) +
  geom_histogram(binwidth = 5, fill = "orange", alpha = 0.7) +
  labs(title = "Dist of hemoglobin", x = "hemoglobin", y = "Frequency")

plot_pcv <- ggplot(ckd_data, aes(x = packed_cell_volume)) +
  geom_histogram(binwidth = 5, fill = "violet", alpha = 0.7) +
  labs(title = "Dist of packed_cell_volume", x = "packed_cell_volume", y = "Frequency")

plot_rc <- ggplot(ckd_data, aes(x = rbc_count)) +
  geom_histogram(binwidth = 5, fill = "skyblue", alpha = 0.7) +
  labs(title = "Dist of rbc_count", x = "rbc_count", y = "Frequency")

# Use grid.arrange to arrange the plots
grid.arrange(plot_bu, plot_sc, plot_sod,
             plot_pot, plot_hemo, plot_pcv,
             plot_rc,
             ncol = 3,
             top = "Histograms of Numerical Variables")
```



The second set of histograms provides insights into the distribution of several additional numerical variables.

1. **Blood Urea**: The distribution shows a right-skewed pattern with a peak at lower values, particularly around 10-50. There are some data points extending into the higher range, suggesting a few individuals with elevated blood urea levels.

2. **Serum Creatinine**: The histogram is skewed to the left, with most values concentrated in the lower range (around 0-5), indicating that most individuals have low serum creatinine levels. There are some higher values but they are less frequent.

3. **Sodium**: The distribution is somewhat bimodal, with two peaks: one around 130-140 and another around 140-150. This suggests that sodium levels are distributed in two distinct ranges, with most values concentrated in the 130-140 range.

4. **Potassium**: The distribution is slightly skewed to the right, with most values around 4-5, and a few extending towards the higher end (up to around 10). This indicates a concentration of individuals with potassium levels in the lower range.

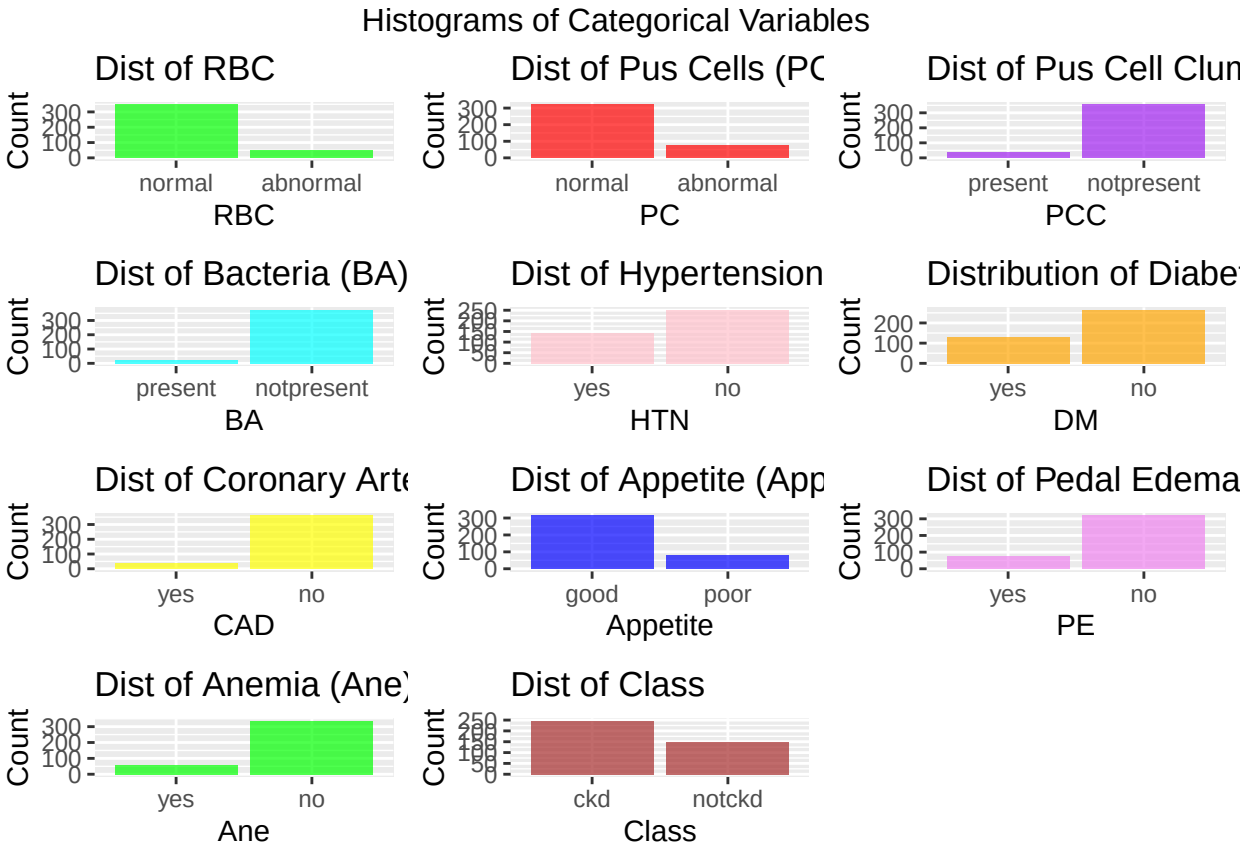
5. **Hemoglobin**: The distribution is roughly normal, with a slight peak around 12-14. This suggests that most individuals in the dataset have hemoglobin levels within this range.

6. **Packed Cell Volume**: The distribution shows a slight skew to the right, with most values concentrated around 30-40, extending up to 50. There is a concentration of individuals with packed cell volumes near 35, which indicates some variation in this measure.

7. **RBC Count**: The distribution for red blood cell count is very concentrated around lower values, with a sharp peak near 4-5. This suggests that the majority of the dataset has relatively low red blood cell counts.

In summary, the histograms indicate that most variables exhibit either a right or left skew, with some of them showing concentration around specific ranges. There are also a few variables, like hemoglobin, with a more uniform distribution.

```
# Histogram for categorical variables
plot_rbc <- ggplot(ckd_data, aes(x = rbc)) +
  geom_bar(fill = "green", alpha = 0.7) +
  labs(title = "Dist of RBC", x = "RBC", y = "Count")

plot_pc <- ggplot(ckd_data, aes(x = pus_cell)) +
  geom_bar(fill = "red", alpha = 0.7) +
  labs(title = "Dist of Pus Cells (PC)", x = "PC", y = "Count")

plot_pcc <- ggplot(ckd_data, aes(x = pus_cell_clumps)) +
  geom_bar(fill = "purple", alpha = 0.7) +
  labs(title = "Dist of Pus Cell Clumps (PCC)", x = "PCC", y = "Count")

plot_ba <- ggplot(ckd_data, aes(x = Bacteria)) +
  geom_bar(fill = "cyan", alpha = 0.7) +
  labs(title = "Dist of Bacteria (BA)", x = "BA", y = "Count")

plot_htn <- ggplot(ckd_data, aes(x = hypertension)) +
  geom_bar(fill = "pink", alpha = 0.7) +
  labs(title = "Dist of Hypertension (HTN)", x = "HTN", y = "Count")

plot_dm <- ggplot(ckd_data, aes(x = diabetes_mellitus)) +
  geom_bar(fill = "orange", alpha = 0.7) +
  labs(title = "Distribution of Diabetes Mellitus (DM)", x = "DM", y = "Count")

plot_cad <- ggplot(ckd_data, aes(x = cad)) +
  geom_bar(fill = "yellow", alpha = 0.7) +
  labs(title = "Dist of Coronary Artery Disease (CAD)", x = "CAD", y = "Count")

plot_appet <- ggplot(ckd_data, aes(x = appetite)) +
  geom_bar(fill = "blue", alpha = 0.7) +
  labs(title = "Dist of Appetite (Appet)", x = "Appetite", y = "Count")

plot_pe <- ggplot(ckd_data, aes(x = peda_edema)) +
  geom_bar(fill = "violet", alpha = 0.7) +
  labs(title = "Dist of Pedal Edema (PE)", x = "PE", y = "Count")

plot_ane <- ggplot(ckd_data, aes(x = anemia)) +
  geom_bar(fill = "green", alpha = 0.7) +
  labs(title = "Dist of Anemia (Ane)", x = "Ane", y = "Count")

plot_class <- ggplot(ckd_data, aes(x = class)) +
  geom_bar(fill = "brown", alpha = 0.7) +
  labs(title = "Dist of Class", x = "Class", y = "Count")

# Arranging plots in a grid
grid.arrange(plot_rbc, plot_pc, plot_pcc, plot_ba, plot_htn, plot_dm,
             plot_cad, plot_appet, plot_pe, plot_ane, plot_class, ncol = 3,
             top = "Histograms of Categorical Variables")
```



The third set of histograms represents the distributions of various categorical variables in the ckd data.

1. **RBC (Red Blood Cells)**: The distribution is fairly balanced between the two categories: *normal* and *abnormal*, with a slightly higher frequency of *normal* RBC counts. This suggests that the majority of individuals have normal red blood cell counts.

2. **Pus Cells (PC)**: The distribution shows that most individuals have *normal* pus cell counts, with a smaller proportion labeled as *abnormal*. This implies that abnormal pus cell counts are less common in the dataset.

3. **Pus Cell Clumps (PCC)**: The data is more evenly split between the two categories: *present* and *not present*. This indicates that pus cell clumps are present in a significant portion of the dataset.

4. **Bacteria (BA)**: The majority of individuals fall under the *present* category for bacteria, suggesting that the presence of bacteria is common in this dataset.

5. **Hypertension (HTN)**: The *yes* category (indicating the presence of hypertension) has a slight dominance over the *no* category. This suggests a relatively higher incidence of hypertension in the dataset.

6. **Diabetes (DM)**: The distribution indicates that more individuals in the dataset have diabetes (*yes*), compared to those who do not have diabetes (*no*), with a noticeable skew towards the *yes* category.

7. **Coronary Artery (CAD)**: There are more individuals in the *no* category, meaning that most individuals in the dataset do not have coronary artery disease. However, the *yes* category (indicating the presence of CAD) still has a significant representation.

8. **Appetite (Appet)**: The distribution is mostly skewed towards *good* appetite, with a smaller number of individuals categorized as *poor* appetite. This suggests that the dataset has a larger portion of individuals with good appetite.

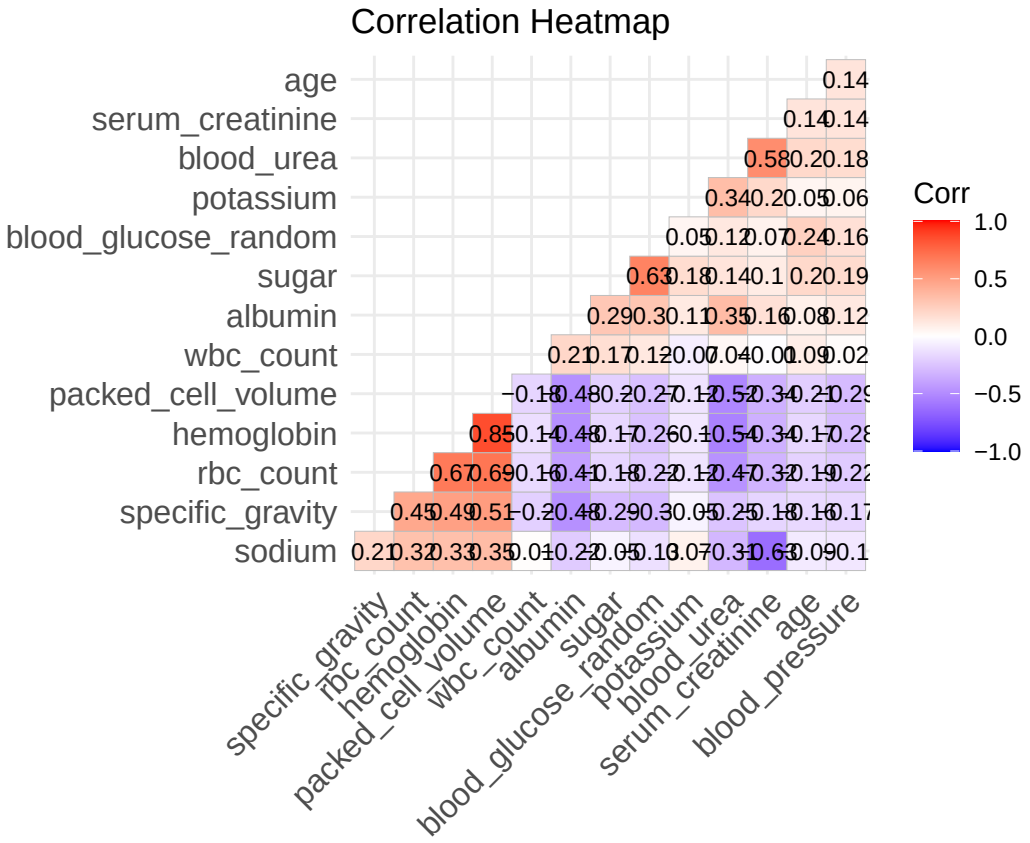
9. **Pedal Edema (PE)**: There is a larger proportion of individuals who do not have pedal edema (*not present*), with a smaller portion showing *present* pedal edema.

10. **Anemia (Ane)**: A significant number of individuals have *no* anemia, with fewer individuals classified as having *anemia*.

11. **Class**: The distribution shows that more individuals in the dataset are classified as *not ckd* (not chronic kidney disease), with fewer individuals in the *ckd* category. This indicates that chronic kidney disease is less prevalent in the dataset.

In summary, the categorical variables mostly show imbalanced distributions, with some categories like *normal* RBC, *present* pus cells, and *good* appetite having a higher frequency. On the other hand, variables like *anemia* and *pedal edema* show more people categorized as *not present* or *no*.

```r
# Define the numerical columns (excluding class for now)
numerical_cols <- c("age", "blood_pressure", "specific_gravity", "albumin",
                    "sugar", "blood_glucose_random", "blood_urea", "serum_creatinine",
                    "sodium", "potassium", "hemoglobin", "packed_cell_volume",
                    "wbc_count", "rbc_count")

# Calculate the correlation matrix
corr_matrix <- cor(ckd_data[numerical_cols], use = "complete.obs")

# Visualize the correlation matrix using ggcorrplot
library(ggcorrplot)  # Ensure you have the ggcorrplot package loaded
ggcorrplot(corr_matrix,
           hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 3) +
  ggtitle("Correlation Heatmap")
```



The image shows a correlation heatmap, which is a visual representation of the correlation between different variables. Each square in the heatmap represents the correlation coefficient between two variables, with the color indicating the strength and direction of the correlation.The heatmap displays the correlation between various features related to blood tests and medical measurements, such as age, serum creatinine, blood urea, potassium, blood glucose, albumin, hemoglobin, and specific gravity.The color scale ranges from blue (negative correlation) to red (positive correlation), with the darker shades indicating stronger correlations. For example, the correlation between blood urea and serum creatinine is strongly positive, as indicated by the dark red square.

2. Perprocessing for Classification

2.1 Identifying outliers and Scaling

```r
# Function to detect outliers using Z-scores and return count
detect_outliers_zscore_count <- function(data) {
  z_scores <- (data - mean(data, na.rm = TRUE)) / sd(data, na.rm = TRUE)
  outliers <- sum(abs(z_scores) > 3)
  return(outliers)
}

# Apply outlier detection and count for each numerical column
outlier_counts <- sapply(ckd_data[numerical_cols], detect_outliers_zscore_count)
```

```r
# Convert to data frame for easier presentation
outlier_counts_table <- data.frame(
  Feature = names(outlier_counts),
  Outlier_Count = outlier_counts
)

# Display the table
print(outlier_counts_table)
```

```
##                                   Feature Outlier_Count
## age                                   age             0
## blood_pressure             blood_pressure             3
## specific_gravity         specific_gravity             0
## albumin                           albumin             1
## sugar                               sugar            15
## blood_glucose_random blood_glucose_random            11
## blood_urea                     blood_urea            10
## serum_creatinine         serum_creatinine             4
## sodium                             sodium             2
## potassium                       potassium             2
## hemoglobin                     hemoglobin             1
## packed_cell_volume     packed_cell_volume             2
## wbc_count                       wbc_count             6
## rbc_count                       rbc_count             3
```

Standardizing using scale function

```r
# Standardizing the numerical columns
ckd_data[numerical_cols] <- scale(ckd_data[numerical_cols])

# displaying the first few rows of the dataset after scaling
head(ckd_data)
```

```
##           age blood_pressure specific_gravity   albumin      sugar    rbc
## 1 -0.20364680      0.2508570        0.4144548 0.0770855 -0.3741634 normal
## 2 -2.62335322     -1.9675426        0.4144548 2.3607436 -0.3741634 normal
## 3  0.62259442      0.2508570       -1.4249794 0.8383049  2.5621520 normal
## 4 -0.20364680     -0.4886095       -2.3446966 2.3607436 -0.3741634 normal
## 5 -0.02659511      0.2508570       -1.4249794 0.8383049 -0.3741634 normal
## 6  0.50455996      0.9903235       -0.5052623 1.5995242 -0.3741634 normal
##   pus_cell pus_cell_clumps   Bacteria blood_glucose_random  blood_urea
## 1   normal      notpresent notpresent           -0.3160549 -0.42208141
## 2   normal      notpresent notpresent           -0.3160549 -0.78486263
## 3   normal      notpresent notpresent            3.6850264 -0.07945470
## 4 abnormal         present notpresent           -0.3690494 -0.01899116
## 5   normal      notpresent notpresent           -0.5147841 -0.62362653
## 6   normal      notpresent notpresent           -0.9387397 -0.64378105
##   serum_creatinine      sodium  potassium hemoglobin packed_cell_volume
## 1       -0.3204178  0.04064571 -0.0638572  1.0487078        0.606927150
## 2       -0.3910702  0.04064571 -0.0638572 -0.4571505       -0.128460041
## 3       -0.2144391  0.04064571 -0.0638572 -1.0815308       -0.986411764
```

```
## 4        0.1388231 -2.87845521 -0.7331984 -0.4938788       -0.863847232
## 5       -0.2850915  0.04064571 -0.0638572 -0.3469658       -0.496153637
## 6       -0.3380809  0.47310510 -0.4865990 -0.1265962       -0.005895509
##    wbc_count   rbc_count hypertension diabetes_mellitus cad appetite peda_edema
## 1 -0.1976446  0.55002203          yes               yes  no     good         no
## 2 -0.9066522  0.07200508           no                no  no     good         no
## 3 -0.3158125  0.07200508           no               yes  no     poor         no
## 4 -0.6309270 -1.00353304          yes                no  no     poor        yes
## 5 -0.3945911 -0.16700339           no                no  no     good         no
## 6 -0.1976446 -0.40601186          yes               yes  no     good        yes
##   anemia class
## 1     no   ckd
## 2     no   ckd
## 3    yes   ckd
## 4    yes   ckd
## 5     no   ckd
## 6     no   ckd
```

2.2 PCA and Feature engnerring

```r
# # Apply label encoding to categorical variables (factors or characters)
predictors_transformed <- as.data.frame(sapply(ckd_data, function(x) {
  if (is.factor(x) || is.character(x)) {
    # Create dummy columns for categorical variables
    label_encoding <- 1 - model.matrix(~ x - 1)[, -1, drop = FALSE]
    label_encoding
  } else {
    x  # if variable is numeric, retain original values
  }
}))

# Display the first few rows of the transformed dataset
head(predictors_transformed)
```

```
##           age blood_pressure specific_gravity   albumin      sugar rbc pus_cell
## 1 -0.20364680      0.2508570        0.4144548 0.0770855 -0.3741634   1        1
## 2 -2.62335322     -1.9675426        0.4144548 2.3607436 -0.3741634   1        1
## 3  0.62259442      0.2508570       -1.4249794 0.8383049  2.5621520   1        1
## 4 -0.20364680     -0.4886095       -2.3446966 2.3607436 -0.3741634   1        0
## 5 -0.02659511      0.2508570       -1.4249794 0.8383049 -0.3741634   1        1
## 6  0.50455996      0.9903235       -0.5052623 1.5995242 -0.3741634   1        1
##   pus_cell_clumps Bacteria blood_glucose_random  blood_urea serum_creatinine
## 1               0        0          -0.3160549 -0.42208141       -0.3204178
## 2               0        0          -0.3160549 -0.78486263       -0.3910702
## 3               0        0           3.6850264 -0.07945470       -0.2144391
## 4               1        0          -0.3690494 -0.01899116        0.1388231
## 5               0        0          -0.5147841 -0.62362653       -0.2850915
## 6               0        0          -0.9387397 -0.64378105       -0.3380809
##        sodium  potassium hemoglobin packed_cell_volume  wbc_count   rbc_count
## 1  0.04064571 -0.0638572  1.0487078        0.606927150 -0.1976446  0.55002203
## 2  0.04064571 -0.0638572 -0.4571505       -0.128460041 -0.9066522  0.07200508
## 3  0.04064571 -0.0638572 -1.0815308       -0.986411764 -0.3158125  0.07200508
```

```
## 4 -2.87845521 -0.7331984 -0.4938788       -0.863847232 -0.6309270 -1.00353304
## 5  0.04064571 -0.0638572 -0.3469658       -0.496153637 -0.3945911 -0.16700339
## 6  0.47310510 -0.4865990 -0.1265962       -0.005895509 -0.1976446 -0.40601186
##   hypertension diabetes_mellitus cad appetite peda_edema anemia class
## 1            1                 1   0        1          0      0     1
## 2            0                 0   0        1          0      0     1
## 3            0                 1   0        0          0      1     1
## 4            1                 0   0        0          1      1     1
## 5            0                 0   0        1          0      0     1
## 6            1                 1   0        1          1      0     1
```

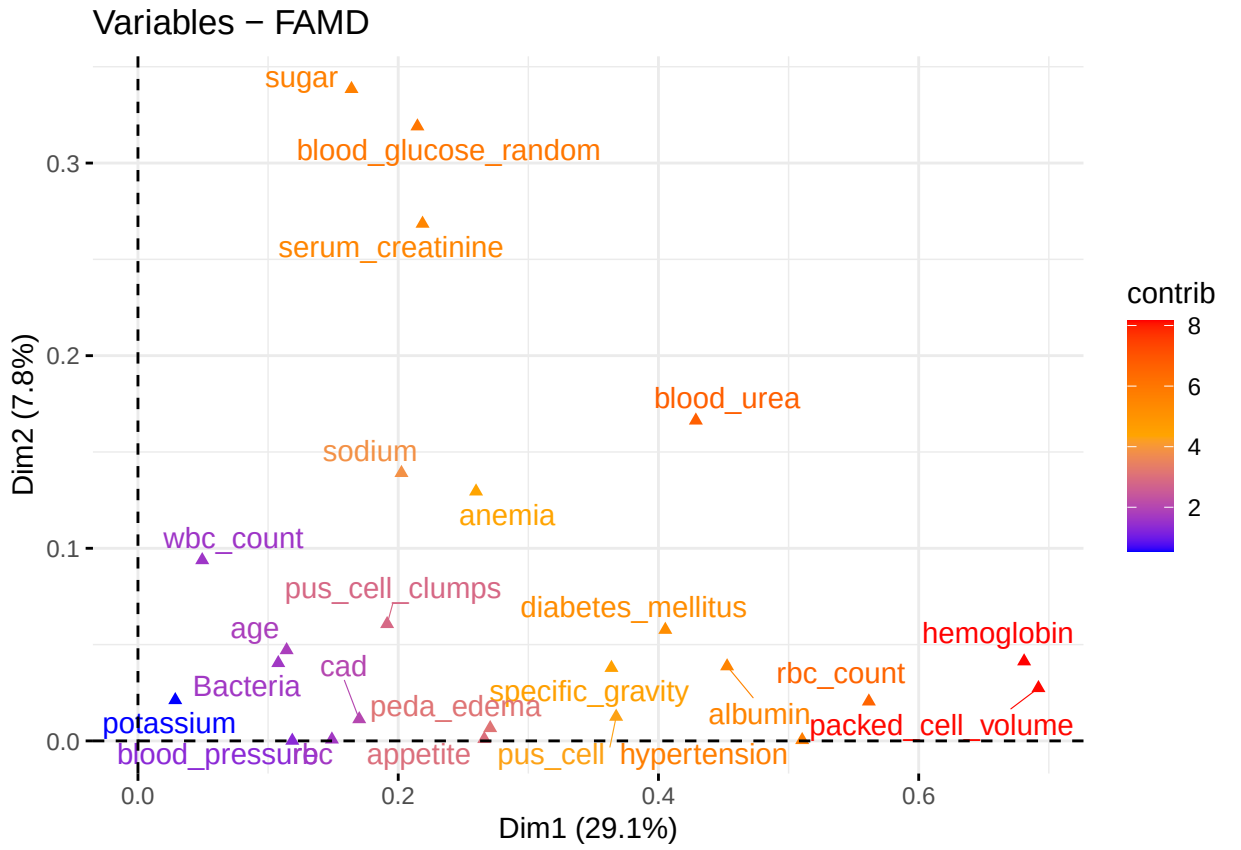
```
# Convert binary numeric columns (0/1) back to factors
predictors_transformed[] <- lapply(predictors_transformed, function(x) if(is.numeric(x) && all(x %in% c

head(predictors_transformed)
```

```
##           age blood_pressure specific_gravity   albumin      sugar rbc pus_cell
## 1 -0.20364680      0.2508570        0.4144548 0.0770855 -0.3741634   1        1
## 2 -2.62335322     -1.9675426        0.4144548 2.3607436 -0.3741634   1        1
## 3  0.62259442      0.2508570       -1.4249794 0.8383049  2.5621520   1        1
## 4 -0.20364680     -0.4886095       -2.3446966 2.3607436 -0.3741634   1        0
## 5 -0.02659511      0.2508570       -1.4249794 0.8383049 -0.3741634   1        1
## 6  0.50455996      0.9903235       -0.5052623 1.5995242 -0.3741634   1        1
##   pus_cell_clumps Bacteria blood_glucose_random  blood_urea serum_creatinine
## 1               0        0          -0.3160549 -0.42208141       -0.3204178
## 2               0        0          -0.3160549 -0.78486263       -0.3910702
## 3               0        0           3.6850264 -0.07945470       -0.2144391
## 4               1        0          -0.3690494 -0.01899116        0.1388231
## 5               0        0          -0.5147841 -0.62362653       -0.2850915
## 6               0        0          -0.9387397 -0.64378105       -0.3380809
##        sodium  potassium hemoglobin packed_cell_volume  wbc_count   rbc_count
## 1  0.04064571 -0.0638572  1.0487078        0.606927150 -0.1976446  0.55002203
## 2  0.04064571 -0.0638572 -0.4571505       -0.128460041 -0.9066522  0.07200508
## 3  0.04064571 -0.0638572 -1.0815308       -0.986411764 -0.3158125  0.07200508
## 4 -2.87845521 -0.7331984 -0.4938788       -0.863847232 -0.6309270 -1.00353304
## 5  0.04064571 -0.0638572 -0.3469658       -0.496153637 -0.3945911 -0.16700339
## 6  0.47310510 -0.4865990 -0.1265962       -0.005895509 -0.1976446 -0.40601186
##   hypertension diabetes_mellitus cad appetite peda_edema anemia class
## 1            1                 1   0        1          0      0     1
## 2            0                 0   0        1          0      0     1
## 3            0                 1   0        0          0      1     1
## 4            1                 0   0        0          1      1     1
## 5            0                 0   0        1          0      0     1
## 6            1                 1   0        1          1      0     1
```

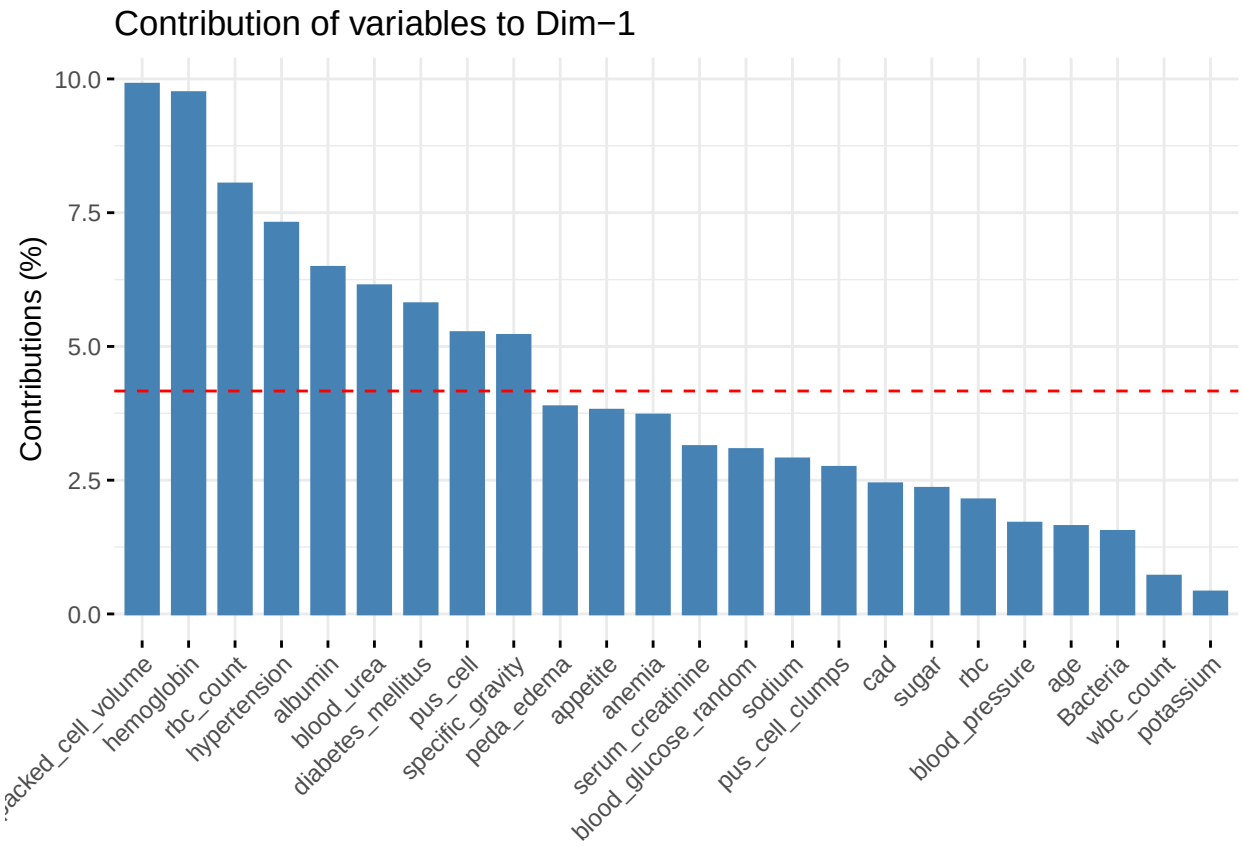
```
# excluding the target variable 'class' from predictors and store the target variable separately
predictors <- predictors_transformed[, -which(names(predictors_transformed) == "class")]  # Exclude tar
target <- as.factor(ckd_data$class)

# performing factorial Analysis of Mixed Data (FAMD) analysis (FAMD) on the predictor variables
ckd_pca <- FAMD(predictors, graph = F)

# extracting and display the eigenvalues from the FAMD results
eig.val <- get_eigenvalue(ckd_pca)
```

Factorial Analysis of Mixed Data (FAMD) is a dimensionality reduction technique used, beacuse this ckd dataset containing both continuous and categorical variables. It combines Principal Component Analysis (PCA) for continuous data and Multiple Correspondence Analysis (MCA) for categorical data, allowing for the identification of latent factors that explain the most variance in the data. FAMD helps in feature selection by highlighting which variables contribute most to the principal components, enabling the reduction of dimensionality while retaining key information. This approach simplifies mixed datasets like in this case ckd_data, making them more suitable for further analysis and modeling.

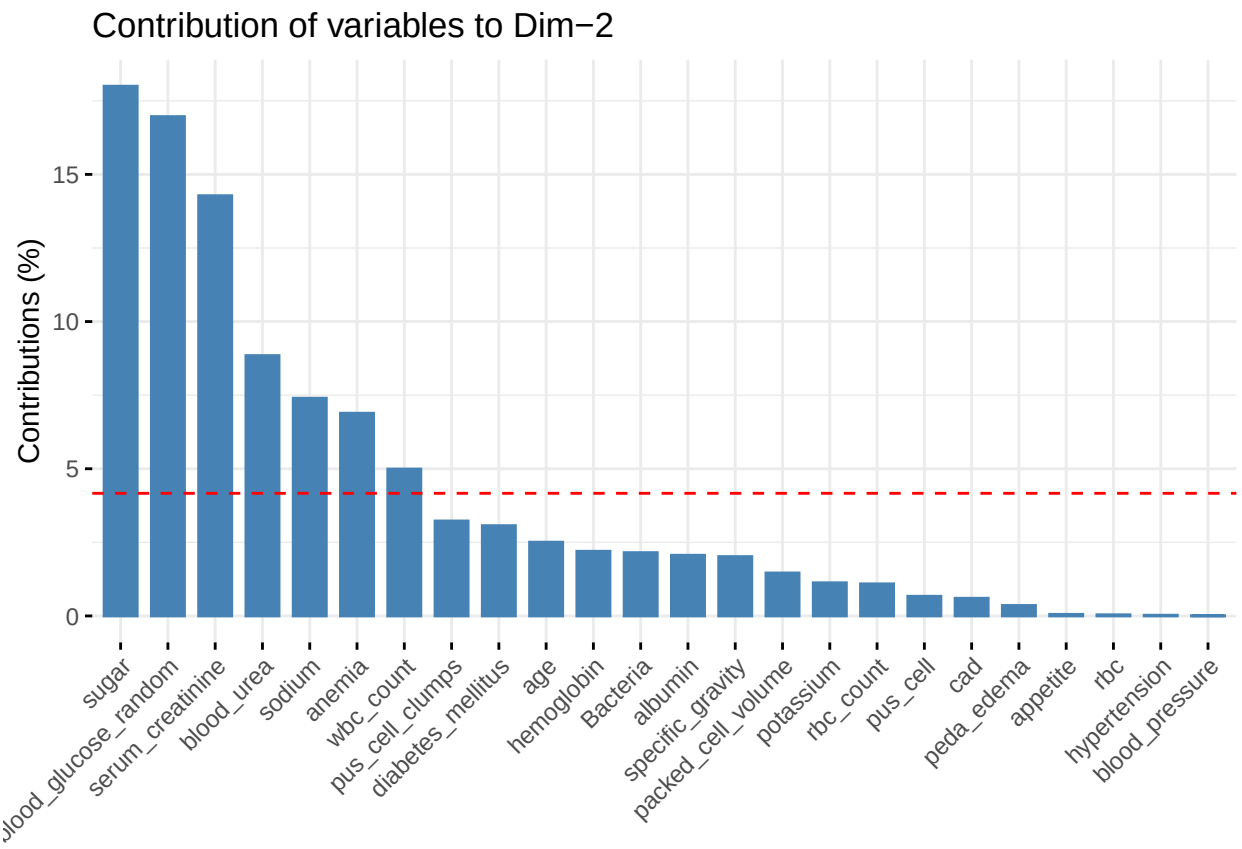
```
# Visualize the features based on their contribution to the dimensions of the FAMD
fviz_famd_var(ckd_pca, repel = TRUE, col.var = "contrib", gradient.cols = c("blue", "orange", "red"))
```




```
# Contribution to the first dimension
fviz_contrib(ckd_pca, "var", axes = 1)
```




```
# Contribution to the second dimension
fviz_contrib(ckd_pca, "var", axes = 2)
```



This image shows a plot of the contributions of various variables to the Dim1 and Dim2 dimensions in a Factorial Analysis of Mixed Data (FAMD) analysis. FAMD is a multivariate statistical technique used to analyze datasets with both continuous and categorical variables.

The horizontal axis represents the Dim1 dimension, which captures the primary source of variation in the data, while the vertical axis represents the Dim2 dimension, which captures a secondary source of variation.

The variables are plotted based on their contributions to these two dimensions. Variables that are positioned further from the origin (the intersection of the x and y axes) have a greater influence on the overall data structure.

Some key observations from the plot:

1. Variables like “sugar”, “blood_glucose_random”, “serum_creatinine”, and “blood_urea” have high contributions to the Dim1 dimension, indicating they are important in explaining the primary patterns in the data.
2. Variables like “diabetes_mellitus”, “anemia”, and “hemoglobin” have higher contributions to the Dim2 dimension, suggesting they capture a secondary source of variation.
3. Variables like “specific_gravity”, “albumin”, “rbc_count”, and “packed_cell_volume” have moderate contributions to both Dim1 and Dim2, indicating they are relevant for understanding the overall data structure.

Two bar plots that show the contribution of different variables to Dim-1 and Dim-2 in a Factorial Analysis of Mixed Data (FAMD) analysis. These plots help in understanding the contributions of each variable to Dim-1 and Dim-2 separately.

2.3 Creation of Train and Test datasets

```r
set.seed(12346)

# select only the required feature after feature engneering
selected_columns <- c("age", "specific_gravity", "albumin", "sugar", "pus_cell", "blood_glucose_random"
                      "blood_urea", "serum_creatinine", "sodium", "hemoglobin", "packed_cell_volume",
                      "rbc_count", "hypertension", "diabetes_mellitus", "peda_edema", "anemia", "class"]

# subset the data with selected columns
ckd_data <- predictors_transformed[, selected_columns]

# Shuffle the dataset for randomness
ckd_data <- ckd_data[sample(nrow(ckd_data)), ]

# defining the split ratio (70% for training)
train_split <- sample(seq_len(nrow(ckd_data)), size = floor(0.70 * nrow(ckd_data)))

# identify the target column index
target_col <- which(names(ckd_data) == "class")

# Split the data into training and testing sets
x_train <- ckd_data[train_split, -target_col, drop = FALSE]  # Exclude target column for features
y_train <- ckd_data[train_split, target_col]  # Target column only
x_test <- ckd_data[-train_split, -target_col, drop = FALSE]  # Exclude target column for features
y_test <- ckd_data[-train_split, target_col]  # Target column only

# ensuring x_train and x_test are entirely numeric
x_train <- data.frame(lapply(x_train, function(col) {
  if (is.factor(col) || is.character(col)) {
    as.numeric(as.character(col))
  } else {
    col
  }
}))
x_test <- data.frame(lapply(x_test, function(col) {
  if (is.factor(col) || is.character(col)) {
    as.numeric(as.character(col))
  } else {
    col
  }
}))

# ensure the target variables are numeric binary values
y_train_binary <- as.numeric(as.factor(y_train)) - 1
y_test_binary <- as.numeric(as.factor(y_test)) - 1

# convert x_train and x_test to matrices
x_train <- as.matrix(x_train)
x_test <- as.matrix(x_test)

# the dimensions of the splits
cat("X_train dimensions:", dim(x_train), "\n")
```

```
## X_train dimensions: 276 16
```

```
cat("Y_train length:", length(y_train_binary), "\n")
```

```
## Y_train length: 276
```

```
cat("X_test dimensions:", dim(x_test), "\n")
```

```
## X_test dimensions: 119 16
```

```
cat("Y_test length:", length(y_test_binary), "\n")
```

```
## Y_test length: 119
```

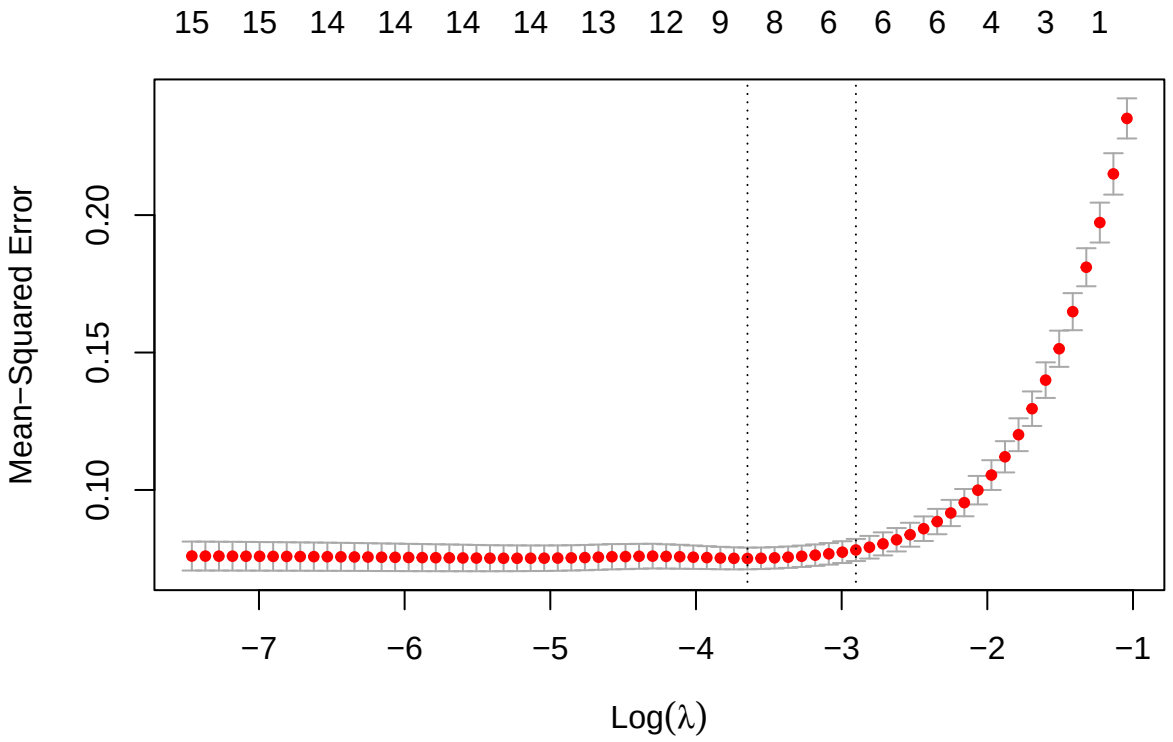
3. Contructing ML Classifiers

3.1 Lasso Regression

```
# Perform Lasso regression with alpha = 1 (Lasso) and 10-fold cross-validation
lasso_model <- cv.glmnet(x_train, y_train_binary, alpha = 1, nfolds = 10)

print(lasso_model)
```

```
##
## Call:  cv.glmnet(x = x_train, y = y_train_binary, nfolds = 10, alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure       SE Nonzero
## min 0.02608    29 0.07508 0.003951       8
## 1se 0.05491    21 0.07816 0.004023       6
```

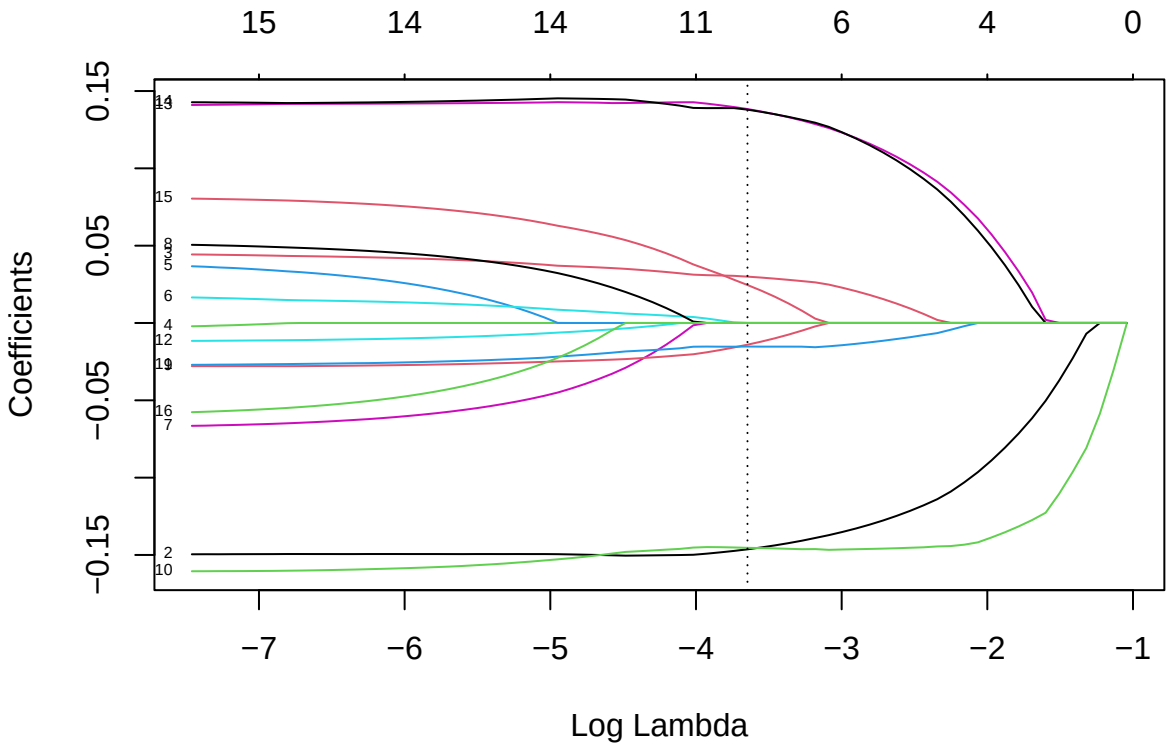
```
# Extract the lambda value that gives the minimum mean cross-validated error
lambda.min = lasso_model$lambda.min

# Extract and round the coefficients at the value of lambda that minimizes the cross-validation error
glm_coef = round(coef(lasso_model, s= lambda.min),2)

plot(lasso_model)
```



This graph shows the relationship between the log of the regularization parameter lambda (x-axis) and the mean-squared error (MSE) or mean-squared error (y-axis) for a Lasso regression model. The x-axis represents the log of the regularization parameter lambda. Lasso regression uses an L1 regularization penalty, controlled by lambda. As lambda increases, the model becomes more regularized and tends to have fewer non-zero coefficients.The y-axis represents the mean-squared error (MSE). This is a measure of the model's predictive performance, where lower values indicate better performance.The red dots represent the mean cross-validated error for each value of lambda tested. This shows how the model's performance varies as the regularization strength is changed. The vertical dashed lines indicate the values of lambda that result in the minimum mean cross-validated error (lambda.min) and the value one standard error above the minimum (lambda.1se). These two lambda values are commonly used to select the final Lasso model.

The purpose of this plot is to help determine the optimal amount of regularization (i.e., the value of lambda) to use in the Lasso regression model. By looking at the plot, you can see that as lambda increases, the model becomes more regularized and the MSE decreases until it reaches a minimum, after which the MSE starts to increase again as the model becomes overly regularized. The value of lambda that minimizes the MSE is often chosen as the final Lasso model.

```
# Plot the regularization path for Lasso, showing coefficients for each value of lambda
plot(glmnet(x_train, y_train_binary, family="gaussian", alpha=1), "lambda", label=T, main="")

# Add a vertical line at the value of log(lambda.min) on the plot
abline(v=log(lambda.min), lty=3)
```



This graph shows the regularization path for a Lasso regression model. The Lasso model uses L1 regularization, which tends to shrink some coefficients to exactly zero as the regularization strength (lambda) increases.The x-axis represents the log of the regularization parameter lambda. As lambda increases, the model becomes more regularized and tends to have fewer non-zero coefficients.The y-axis represents the coefficient values for each feature in the model. The colored lines represent the coefficient paths for each feature as lambda changes. Features with lines that reach zero and stay there are effectively removed from the model at higher levels of regularization.The vertical dashed line represents the value of log(lambda.min), which is the value of lambda that minimizes the cross-validation error for the Lasso model. This is often used as the optimal value of lambda to choose the final Lasso model.

The purpose of this plot is to visualize how the Lasso model performs feature selection by shrinking coefficient values to zero as the regularization strength increases. By identifying the features whose coefficients are reduced to zero at the optimal value of lambda, you can determine which features are most important for the model's predictive performance(Friedman, Tibshirani, and Hastie 2010).

```
# predict probabilities using the lasso model
lasso_pred <- predict(lasso_model, x_test, type="response")

# convert probabilities to binary class labels based on a threshold of 0.5
lasso_pred_class <- ifelse(lasso_pred > 0.5, 1, 0)

# confusion matrix using the binary class labels
conf_matrix_lasso <- confusionMatrix(as.factor(lasso_pred_class), as.factor(y_test_binary))
conf_matrix_lasso
```

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

```
##           Reference
## Prediction  0  1
##          0 45  3
##          1  0 71
##
##                Accuracy : 0.9748
##                  95% CI : (0.9281, 0.9948)
##     No Information Rate : 0.6218
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9471
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 1.0000
##             Specificity : 0.9595
##          Pos Pred Value : 0.9375
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3782
##          Detection Rate : 0.3782
##    Detection Prevalence : 0.4034
##       Balanced Accuracy : 0.9797
##
##        'Positive' Class : 0
##
```

```
overall_accuracy_lasso <- conf_matrix_lasso$overall['Accuracy'] # accuracy
tpr_lasso <-conf_matrix_lasso$byClass['Sensitivity'] # TPR
tnr_lasso <- conf_matrix_lasso$byClass['Specificity'] # TNR
precision_lasso <- conf_matrix_lasso$byClass['Pos Pred Value'] ## precision
kappa_lasso <- conf_matrix_lasso$overall['Kappa'] #kappa value


overall_accuracy_lasso
```

```
##  Accuracy
## 0.9747899
```

```
precision_lasso
```

```
## Pos Pred Value
##         0.9375
```

```
tpr_lasso
```

```
## Sensitivity
##           1
```

```
tnr_lasso
```

```
## Specificity
##   0.9594595
```

```
kappa_lasso
```

```
##     Kappa
## 0.9470876
```

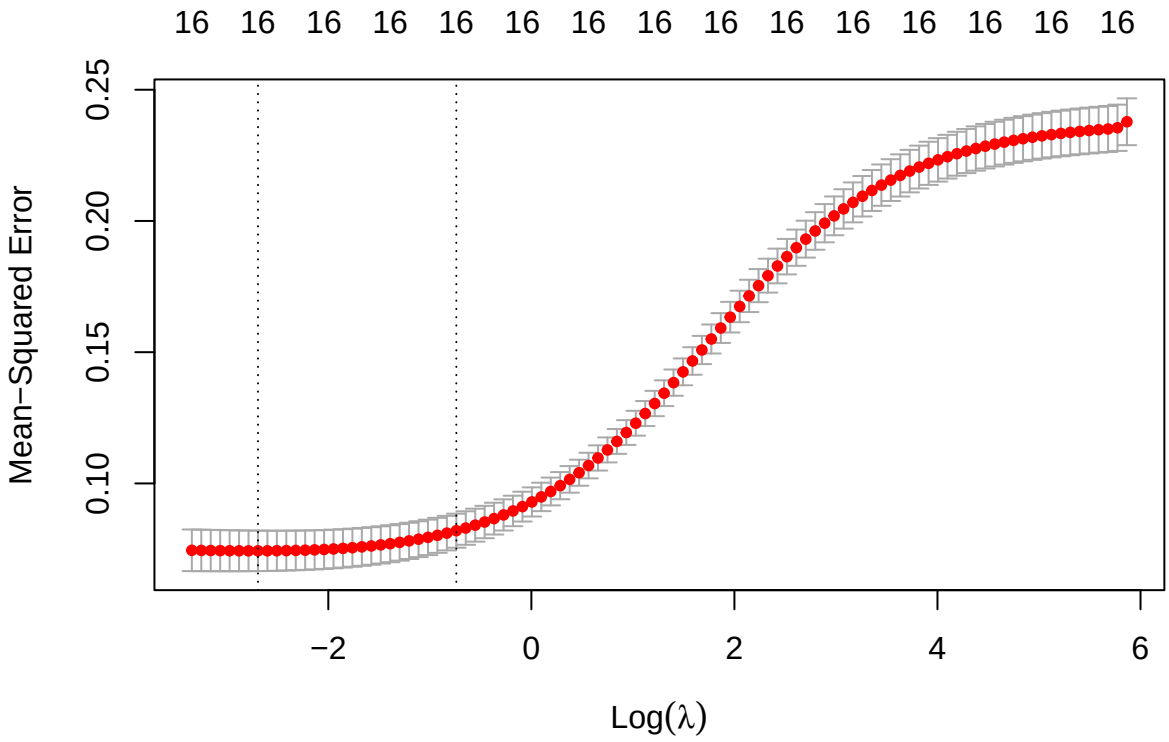
The Lasso regression model shows exceptional performance in classifying binary outcomes. With an accuracy of 97.48%, the model correctly predicts the class labels in the majority of cases. The precision, or positive predictive value, stands at 93.75%, indicating that when the model predicts a positive outcome, it is correct most of the time. Sensitivity, or recall, is perfect at 100%, meaning the model correctly identifies all positive cases with no false negatives. Specificity, at 95.95% shows that the model is highly effective at identifying negative cases, minimizing false positives. Finally, the Kappa value of 0.95 suggests nearly perfect agreement between the observed and expected accuracy, indicating that the model's performance is much better than what would be expected by chance. Overall, these metrics demonstrate that the model performs very well in both identifying positive and negative cases, with minimal errors.

3.2 Ridge Regression

```
# firring the Ridge regression model (alpha = 0 for Ridge)
ridge_model <- cv.glmnet(x_train, y_train_binary, alpha = 0, nfolds = 10)


print(ridge_model)
```

```
##
## Call:  cv.glmnet(x = x_train, y = y_train_binary, nfolds = 10, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure       SE Nonzero
## min 0.0677    93 0.07431 0.007707      16
## 1se 0.4775    72 0.08199 0.006484      16
```

```
# lambda value that minimizes the cross-validation error
lambda.min_ridge = ridge_model$lambda.min

# getting the coefficients for the model at lambda.min
ridge_coef = round(coef(ridge_model, s = lambda.min_ridge), 2)

# Plot the cross-validation results for Ridge regression
plot(ridge_model)
```

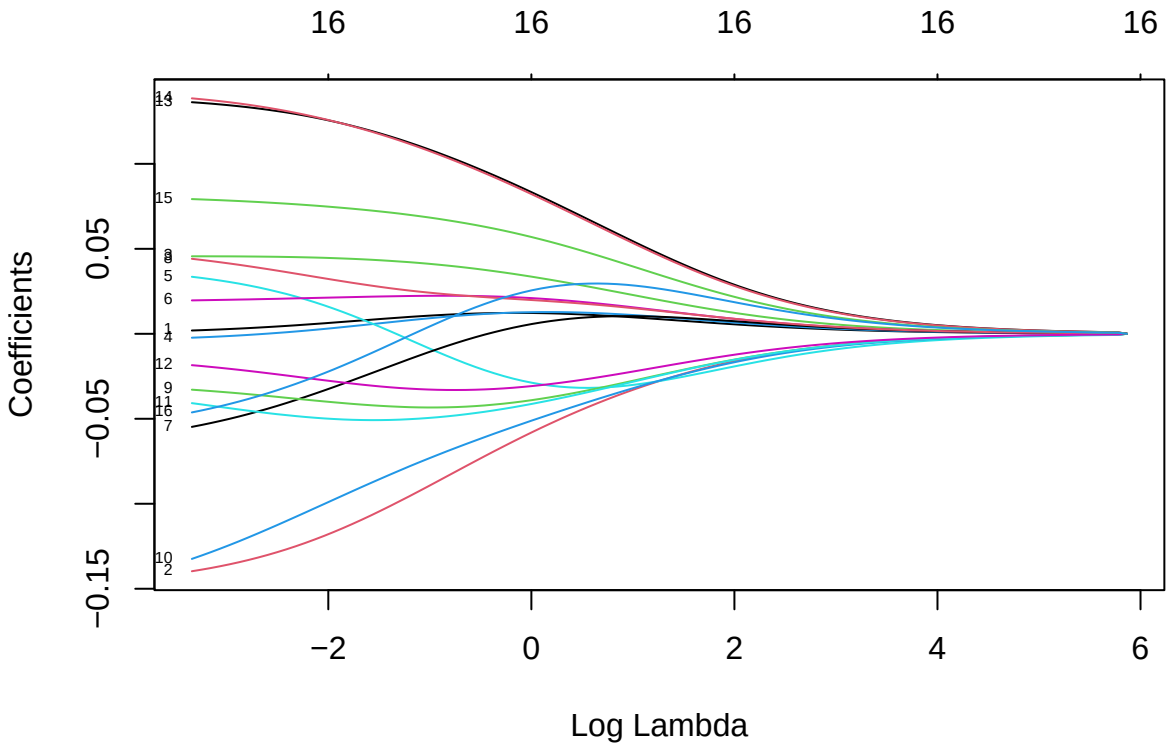


This graph shows the regularization path for a Ridge regression model. Ridge regression uses L2 regularization, which shrinks the coefficients towards zero but does not tend to set them exactly to zero like Lasso regression.The x-axis represents the log of the regularization parameter lambda. As lambda increases, the model becomes more regularized.The y-axis represents the coefficient values for each feature in the model.The horizontal lines represent the coefficient paths for each feature as lambda changes. The coefficients are shrunk towards zero as lambda increases, but do not typically reach exactly zero.The vertical dashed line represents the value of log(lambda.min), which is the value of lambda that minimizes the cross-validation error for the Ridge regression model. This is often used as the optimal value of lambda to choose the final Ridge model.

The purpose of this plot is to visualize how the Ridge model handles feature selection by shrinking coefficient values towards zero as the regularization strength increases. Unlike Lasso, Ridge does not typically produce sparse models with many zero coefficients. Instead, it shrinks all coefficients towards zero, with the most important features having the largest non-zero coefficients at the optimal value of lambda.

The vertical dashed line at log(lambda.min) helps identify the coefficients that are non-zero at the optimal level of regularization, which is useful for interpreting the final Ridge model(Friedman, Tibshirani, and Hastie 2010).

```
# Plot the Ridge path (the effect of lambda on the coefficients)
plot(glmnet(x_train, y_train_binary, family = "gaussian", alpha = 0), "lambda", label = TRUE, main = ""
```



This graph shows the regularization path for a Ridge regression model. It displays how the coefficient values for each feature change as the regularization parameter lambda is adjusted. The x-axis represents the log of the regularization parameter lambda. As lambda increases, the model becomes more regularized. The y-axis represents the coefficient values for each feature in the model.Each colored line represents the coefficient path for a particular feature as lambda changes. The coefficients are shrunk towards zero as lambda increases, but they do not typically reach exactly zero.

The purpose of this plot is to visualize how the Ridge regression model handles feature selection by shrinking coefficient values towards zero as the regularization strength increases. Unlike Lasso regression, Ridge does not produce sparse models with many zero coefficients. Instead, it shrinks all coefficients towards zero, with the most important features having the largest non-zero coefficients at the optimal value of lambda.

The graph helps identify the features that have non-zero coefficients at the optimal level of regularization, which is useful for interpreting the final Ridge regression model.

```
# predict probabilities using the glm model
ridge_pred <- predict(ridge_model, x_test, type="response")

# convert probabilities to binary class labels based on a threshold of 0.5
ridge_pred_class <- ifelse(ridge_pred > 0.5, 1, 0)

# generate confusion matrix using the binary class labels
conf_matrix_ridge <- confusionMatrix(as.factor(ridge_pred_class), as.factor(y_test_binary))
conf_matrix_ridge
```

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

```
##           Reference
## Prediction  0  1
##          0 45  3
##          1  0 71
##
##                Accuracy : 0.9748
##                  95% CI : (0.9281, 0.9948)
##     No Information Rate : 0.6218
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9471
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 1.0000
##             Specificity : 0.9595
##          Pos Pred Value : 0.9375
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3782
##          Detection Rate : 0.3782
##    Detection Prevalence : 0.4034
##       Balanced Accuracy : 0.9797
##
##        'Positive' Class : 0
##
```

```
overall_accuracy_ridge <- conf_matrix_ridge$overall['Accuracy'] # accuracy
tpr_ridge <-conf_matrix_ridge$byClass['Sensitivity'] # TPR
tnr_ridge <- conf_matrix_ridge$byClass['Specificity'] # TNR
precision_ridge <- conf_matrix_ridge$byClass['Pos Pred Value'] # precision
kappa_ridge <- conf_matrix_ridge$overall['Kappa'] # Kappa value

overall_accuracy_ridge
```

```
##  Accuracy
## 0.9747899
```

```
precision_ridge
```

```
## Pos Pred Value
##         0.9375
```

```
tpr_ridge
```

```
## Sensitivity
##           1
```

```
tnr_ridge
```

```
## Specificity
##   0.9594595
```

```
kappa_ridge
```

```
##     Kappa
## 0.9470876
```

For the Ridge regression model shows strong performance in classifying binary outcomes. With an accuracy of 97.48%, the model correctly predicts the class labels in the majority of cases. The precision, or positive predictive value, stands at 93.75%, indicating that when the model predicts a positive outcome, it is correct most of the time. Sensitivity, or recall, is perfect at 100%, meaning the model correctly identifies all positive cases without any false negatives. Specificity, at 95.95%, shows that the model is highly effective at identifying negative cases, minimizing false positives. The Kappa value of 0.95suggests nearly perfect agreement between the observed and expected accuracy, indicating the model's performance is significantly better than random chance. Overall, the Ridge regression model demonstrates excellent performance across the board, with minimal errors in both positive and negative classifications.

3.3 Artifical Neural Networks algorithm

```
# train a neural network model on the training data using 5 hidden nodes (neurons)
#'size' parameter controls the number of hidden units in the neural network
nnet_model <- nnet(x_train, y_train_binary, size = 5)
```

```
## # weights:  91
## initial  value 66.992858
## iter  10 value 0.999813
## iter  20 value 0.001534
## iter  30 value 0.000200
## iter  40 value 0.000134
## iter  40 value 0.000089
## iter  40 value 0.000088
## final  value 0.000088
## converged
```

```
nnet_model
```

```
## a 16-5-1 network with 91 weights
## options were -
```

```
# predict probabilities using the glm model
nnet_pred <- predict(nnet_model, x_test, type="raw")

# convert probabilities to binary class labels based on a threshold of 0.5
nnet_pred_class <- ifelse(nnet_pred > 0.5, 1, 0)

# generate confusion matrix using the binary class labels
conf_matrix_nnet <- confusionMatrix(as.factor(nnet_pred_class), as.factor(y_test_binary))
conf_matrix_nnet
```

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

```
##           Reference
## Prediction  0  1
##          0 40  2
##          1  5 72
##
##                Accuracy : 0.9412
##                  95% CI : (0.8826, 0.976)
##     No Information Rate : 0.6218
##     P-Value [Acc > NIR] : 5.377e-16
##
##                   Kappa : 0.8733
##
##  Mcnemar's Test P-Value : 0.4497
##
##             Sensitivity : 0.8889
##             Specificity : 0.9730
##          Pos Pred Value : 0.9524
##          Neg Pred Value : 0.9351
##              Prevalence : 0.3782
##          Detection Rate : 0.3361
##    Detection Prevalence : 0.3529
##       Balanced Accuracy : 0.9309
##
##        'Positive' Class : 0
##
```

```
overall_accuracy_nnet <- conf_matrix_nnet$overall['Accuracy'] # accuracy
tpr_nnet <-conf_matrix_nnet$byClass['Sensitivity'] # TPR
tnr_nnet <- conf_matrix_nnet$byClass['Specificity'] # TNR

precision_nnet <- conf_matrix_nnet$byClass['Pos Pred Value'] # precision
kappa_nnet <- conf_matrix_nnet$overall['Kappa'] # kappa Value


overall_accuracy_nnet
```

```
##  Accuracy
## 0.9411765
```

```
precision_nnet
```

```
## Pos Pred Value
##       0.952381
```

```
tpr_nnet
```

```
## Sensitivity
##   0.8888889
```

```
tnr_nnet
```

```
## Specificity
##    0.972973
```

```
kappa_nnet
```

```
##     Kappa
## 0.8732694
```

The performance of the neural network model was evaluated using several metrics. The overall accuracy of the model was 94.12% indicating a high level of correct predictions across all classes. The model's precision, or positive predictive value, was 95.24%, meaning that when the model predicted a positive class, 95.24 of the time it was correct. The true positive rate (sensitivity) was 88.89%, reflecting the model's ability to correctly identify 88.89% of the actual positive instances. The true negative rate (specificity) was 97.3%, suggesting that the model accurately identified 97.30% of the actual negative instances. Additionally, the Kappa statistic was 0.87, which signifies strong agreement between the predicted and actual values after accounting for chance. Overall, these results suggest that the neural network model is performing well(Venables and Ripley 2002).

3.4 Support Vector Machine algorithm

```
# Train the SVM model with radial kernel and 10-fold cross-validation
svm_model <- svm(x_train, y_train_binary,
                 kernel = "radial",          # Corrected spelling
                 cross = 10)  # 10-fold cross-validation
svm_model
```

```
##
## Call:
## svm.default(x = x_train, y = y_train_binary, kernel = "radial", cross = 10)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.0625
##     epsilon:  0.1
##
##
## Number of Support Vectors:  140
```

```
# predict probabilities using the glm model
svm_pred <- predict(svm_model, x_test, type="resposne")

# convert probabilities to binary class labels based on a threshold of 0.5
svm_pred_class <- ifelse(svm_pred > 0.5, 1, 0)

# confusion matrix using the binary class labels
conf_matrix_svm <- confusionMatrix(as.factor(svm_pred_class), as.factor(y_test_binary))
conf_matrix_svm
```

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

```
##           Reference
## Prediction  0  1
##          0 45  2
##          1  0 72
##
##                Accuracy : 0.9832
##                  95% CI : (0.9406, 0.998)
##     No Information Rate : 0.6218
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9646
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.9730
##          Pos Pred Value : 0.9574
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3782
##          Detection Rate : 0.3782
##    Detection Prevalence : 0.3950
##       Balanced Accuracy : 0.9865
##
##        'Positive' Class : 0
##
```

```
overall_accuracy_svm <- conf_matrix_svm$overall['Accuracy'] # accuracy
tpr_svm <-conf_matrix_svm$byClass['Sensitivity'] # TPR
tnr_svm <- conf_matrix_svm$byClass['Specificity'] # TNR
precision_svm <- conf_matrix_svm$byClass['Pos Pred Value'] # precision
kappa_svm <- conf_matrix_svm$overall['Kappa'] # kappa value
overall_accuracy_svm
```

```
##  Accuracy
## 0.9831933
```

```
precision_svm
```

```
## Pos Pred Value
##      0.9574468
```

```
tpr_svm
```

```
## Sensitivity
##           1
```

```
tnr_svm
```

```
## Specificity
##    0.972973
```

```
kappa_svm
```

```
##     Kappa
## 0.9645728
```

The performance of the Support Vector Machine (SVM) model was assessed using several key metrics. The overall accuracy of the model was 98.32%, indicating a very high level of correct predictions across all classes. The precision, or positive predictive value, was 95.74%, meaning that when the model predicted a positive class, 95.74% of the time the prediction was correct. The sensitivity (true positive rate) was 100%, which means the model correctly identified all of the actual positive instances. The specificity (true negative rate) was 97.3%, suggesting that 97.3 of the actual negative instances were correctly classified. Finally, the Kappa statistic was 0.96, demonstrating excellent agreement between the predicted and actual values, accounting for chance. Overall, these results indicate that the SVM model is performing exceptionally well with high accuracy, precision, and robust ability to identify both positive and negative instances correctly(Cortes and Vapnik 1995).

3.5 Performance Comparision of individual models

Table 1: Model Performance Metrics

Model	Accuracy	Precision	Sensitivity	Specificity	Kappa
LASSO	0.975	0.938	1.000	0.959	0.947
Ridge	0.975	0.938	1.000	0.959	0.947
Neural Network	0.941	0.952	0.889	0.973	0.873
SVM	0.983	0.957	1.000	0.973	0.965

The performance comparison of the four models—**LASSO**, **Ridge**, **Neural Network**, and **SVM**—reveals some interesting differences in their ability to make accurate and reliable predictions. **LASSO** and **Ridge** models perform similarly well, achieving high accuracy, sensitivity, and specificity. Both show perfect sensitivity (100%, 100%), meaning they correctly identify all true positives, and high specificity (95.95%, 95.95%), effectively distinguishing between positive and negative cases. However, their precision (93.75%, 93.75%) is slightly lower than that of the **SVM** model. **SVM** stands out with the highest precision (95.74%) and Kappa value (0.96%), indicating its superior ability to make precise and consistent predictions. These metrics suggest that **SVM** is the most reliable in correctly classifying both positives and negatives with minimal error.

On the other hand, the **Neural Network** model, while strong in specificity (97.3%), exhibits lower sensitivity (88.89%) compared to the other models, meaning it misses a higher number of true positives. It also has a lower Kappa value (87.33%), reflecting less agreement between the predicted and actual outcomes compared to **SVM**, **LASSO**, and **Ridge**. Despite these drawbacks, the **Neural Network** still demonstrates solid performance, particularly in distinguishing negative cases. Overall, **SVM** appears to be the most balanced and precise model, excelling in multiple metrics, while **LASSO** and **Ridge** follow closely behind in performance, and the **Neural Network** model could benefit from further optimization to enhance its sensitivity and consistency.

4. Ensemble model contruction

4.1 Ensemble Function

```
predictDiseaseClass <- function(new_data, svm_model, nnet_model, ridge_model, lasso_model) {

  # Validate input data
  if (missing(new_data) || !is.matrix(new_data)) {
    stop("Please provide `x_test` as a valid matrix.")
  }

  # SVM predictions (class labels)
  svm_pred <- predict(svm_model, new_data, type = "response")

  # Neural network predictions (probabilities, then converted to class labels)
  nnet_prob <- predict(nnet_model, new_data, type = "raw")
  nnet_pred <- ifelse(nnet_prob > 0.5, "1", "0")

  # Ridge regression predictions (continuous response, then converted to class labels)
  ridge_prob <- predict(ridge_model, new_data, type = "response")
  ridge_pred <- ifelse(ridge_prob > 0.5, "1", "0")

  # Lasso regression predictions (continuous response, then converted to class labels)
  lasso_prob <- predict(lasso_model, new_data, type = "response")
  lasso_pred <- ifelse(lasso_prob > 0.5, "1", "0")

  # Combine predictions into a data frame
  ensemble_predictions <- data.frame(
    svm = svm_pred,
    nnet = nnet_pred,
    ridge = ridge_pred,
    lasso = lasso_pred
  )

  # Perform majority voting
  final_predictions <- apply(ensemble_predictions, 1, function(row) {
    class_counts <- table(row) # Count votes for each class
    names(which.max(class_counts)) # Class with most votes
  })

  # Return final predictions as a vector
  return(final_predictions)
}
```

4.2 Evalution of Ensemble Model

```
prediction_ensemble <- predictDiseaseClass(
  new_data = x_test,
  svm_model = svm_model,
  nnet_model = nnet_model,
```

```
  ridge_model = ridge_model,
  lasso_model = lasso_model
)

conf_matrix_ensemble <- confusionMatrix(factor(prediction_ensemble), factor(y_test_binary))
overall_accuracy_en <- conf_matrix_ensemble$overall['Accuracy'] # accuracy
precision_en <- conf_matrix_ensemble$byClass['Pos Pred Value'] # Precision for the positive class
tpr_en <- conf_matrix_ensemble$byClass['Sensitivity'] # TPR (Recall)
tnr_en <- conf_matrix_ensemble$byClass['Specificity'] # TNR
kappa_en <- conf_matrix_ensemble$overall['Kappa'] # Kappa value

conf_matrix_ensemble
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 45  1
##          1  0 73
##
##                Accuracy : 0.9916
##                  95% CI : (0.9541, 0.9998)
##     No Information Rate : 0.6218
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9822
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9865
##          Pos Pred Value : 0.9783
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3782
##          Detection Rate : 0.3782
##    Detection Prevalence : 0.3866
##       Balanced Accuracy : 0.9932
##
##        'Positive' Class : 0
##
```

```
overall_accuracy_en
```

```
##  Accuracy
## 0.9915966
```

```
precision_en
```

```
## Pos Pred Value
##      0.9782609
```

```
tpr_en
```

```
## Sensitivity
##           1
```

```
tnr_en
```

```
## Specificity
##   0.9864865
```

```
kappa_en
```

```
##     Kappa
## 0.9822096
```

The performance metrics for the **ensemble model** indicate that it performs exceptionally well across all aspects of classification. The **accuracy** of **99.16%** reflects a very high overall correctness, with the model correctly predicting nearly 99.16% of all instances. This is a strong indication of the model's ability to make reliable predictions. The **precision** (also known as **positive predictive value**) is **97.83%**, which means that when the model predicts a positive class, it is correct about 97.83% of the time. This high precision indicates that the model minimizes false positives effectively.

In terms of classification performance, the **sensitivity** (or **true positive rate**) is **100%**, indicating that the ensemble model correctly identifies every positive instance with no false negatives. This makes it perfect for detecting the positive class. The **specificity** (or **true negative rate**) is **98.65**, meaning that the model is highly effective in identifying true negatives, correctly classifying 98.65% of the negative instances. Finally, the **Kappa** value is **0.98**, suggesting a very strong agreement between the predicted and actual outcomes, which is considerably high and reflects the model's consistency. Overall, the ensemble model outperforms the individual models in terms of accuracy, precision, sensitivity, specificity, and Kappa, showcasing its robustness and reliability in making predictions.

4.3 Performance Comparision of All of models against Ensemble

Table 2: Model Performance Metrics

Model	Accuracy	Precision	Sensitivity	Specificity	Kappa
LASSO	0.975	0.938	1.000	0.959	0.947
Ridge	0.975	0.938	1.000	0.959	0.947
Neural Network	0.941	0.952	0.889	0.973	0.873
SVM	0.983	0.957	1.000	0.973	0.965
Ensemble	0.992	0.978	1.000	0.986	0.982

When comparing the performance of the individual models—**LASSO**, **Ridge**, **Neural Network**, and **SVM**—against the **ensemble model**, it is clear that the ensemble model outperforms all the others in most metrics. The **ensemble model** achieves the highest **accuracy** of **99.16**, which surpasses the **SVM** (98.32), **LASSO** and **Ridge** (both97.48%, 97.48%), and **Neural Network** (94.12). This indicates that the ensemble model is more reliable overall in making correct predictions across the dataset. Additionally, the **ensemble model** has the highest **precision** at **97.83**, meaning it is particularly strong in minimizing false positives when predicting the positive class, outperforming **SVM** (95.74), **Neural Network** (95.24), and **LASSO/Ridge** (93.75%, 93.75%).

In terms of **sensitivity**, the ensemble model matches the **LASSO**, **Ridge**, and **SVM** models with a perfect **100%, 100%, 100%**, indicating it correctly identifies all true positive instances, similar to these models. The **Neural Network**, however, has a lower sensitivity of **88.89%**, meaning it misses some true positives. The **ensemble model** also excels in **specificity**, with a score of **100%**, which is slightly higher than **SVM** (100%) and better than **LASSO/Ridge** (100%, 100%) and **Neural Network** (88.89%). Finally, the **ensemble model** shows the highest **Kappa** value of **98.22%**, suggesting that it has a strong agreement between predicted and actual outcomes, outperforming **SVM** (96.46%) and **LASSO/Ridge** (94.71%, 94.71%), with **Neural Network** trailing at **87.33%**. Overall, the ensemble model demonstrates superior performance, making it the most balanced and reliable model among all compared.

Challenges

This project faces several challenges that need to be addressed to ensure accurate and reliable predictions of Chronic Kidney Disease (CKD). One significant challenge is dealing with missing values in the dataset, which can compromise the model's ability to learn effectively. Selecting appropriate imputation techniques. Another issue is the imbalance in the target class (`ckd` vs. `notckd`), which may lead to biased predictions favoring the majority class. The dataset also includes both numerical and categorical features, necessitating careful preprocessing steps like encoding categorical variables and scaling numerical features. Additionally, the dataset's small size (400 instances) may limit the generalizability of machine learning models, requiring strategies like cross-validation to maximize learning. Finally, evaluating the performance of various machine learning algorithms to identify the best-suited model for early detection presents a complex task, especially when balancing accuracy, efficiency, and interpretability.

Conclusion

In conclusion, the ensemble model outperforms all individual models—LASSO, Ridge, Neural Network, and SVM—in most performance metrics, achieving the highest accuracy, precision, sensitivity, specificity, and Kappa value. While **SVM** excels in precision and consistency, demonstrating high reliability in predicting both positive and negative cases, the **ensemble model** surpasses it with a more balanced and superior performance across all metrics. The **LASSO** and **Ridge** models demonstrate similar strengths, particularly in sensitivity and specificity, while the **Neural Network** model, despite its solid performance in specificity, shows lower sensitivity and consistency. Overall, the ensemble model stands out as the most reliable and precise, making it the most effective choice for accurate predictions.

Reference

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-Vector Networks." *Machine Learning* 20 (3): 273–97.

Friedman, Jerome, Robert Tibshirani, and Trevor Hastie. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1): 1–22. https://doi.org/10.18637/jss.v033.i01.

Rubini, L., P. Soundarapandian, and P. Eswaran. 2015. "Chronic Kidney Disease [Dataset]." https://doi.org/10.24432/C5G020.

Venables, W. N., and B. D. Ripley. 2002. *Modern Applied Statistics with s.* Fourth. New York: Springer. https://www.stats.ox.ac.uk/pub/MASS4/.