# Comprehensive Predictive Modeling of Chronic Kidney Disease from Routine Medical Data

Ebad Akhter, Nima Amin Taghavi, and Conor Fitzpatrick

University of San Diego

ADS 503: Applied Predictive Modeling

June 25, 2022

## Abstract

Chronic Kidney Disease (CKD) is a serious health burden in the United States, with an estimated fifteen percent of adults being affected, ninety percent of which do not know they have it until it is in its later stages (*Centers for Disease Control and Prevention,* 2019). CKD involves the gradual loss of function of both kidneys, which filter the blood, leading to an accumulation of toxins and fluid, increasing the risk of stroke and heart attack(Silverstein, 2009). CKD is typically diagnosed by bloodwork, with physicians measuring complex readouts of kidney function and determining diagnosis based on individual interpretation. Because most people with CKD are not aware of it, this diagnosis is typically done during routine screenings during annual checkups or unrelated visits. It would be beneficial for clinicians therefore if an algorithm could be developed to automatically flag the presence of CKD in routine medical checkup data.

Nine machine learning models were created to classify the presence or absence of Chronic Kidney Disease using routine medical data gathered in India that contains information related to age and blood pressure, sodium level, and white blood cell count. The models consisted of: Linear Discriminant Analysis, Neural Network, Stacked Autoencoder Deep Neural Network, XGBoost, Bagged ADA Boost, Nearest Shrunken Centroids, and Lasso and Elastic-net Regularized and Generalized Linear Models. The Bagged ADA Boost model had the best results on our test set, with an Accuracy of 97%, Sensitivity of 94%, and a Specificity of 100%.

## Table of Contents

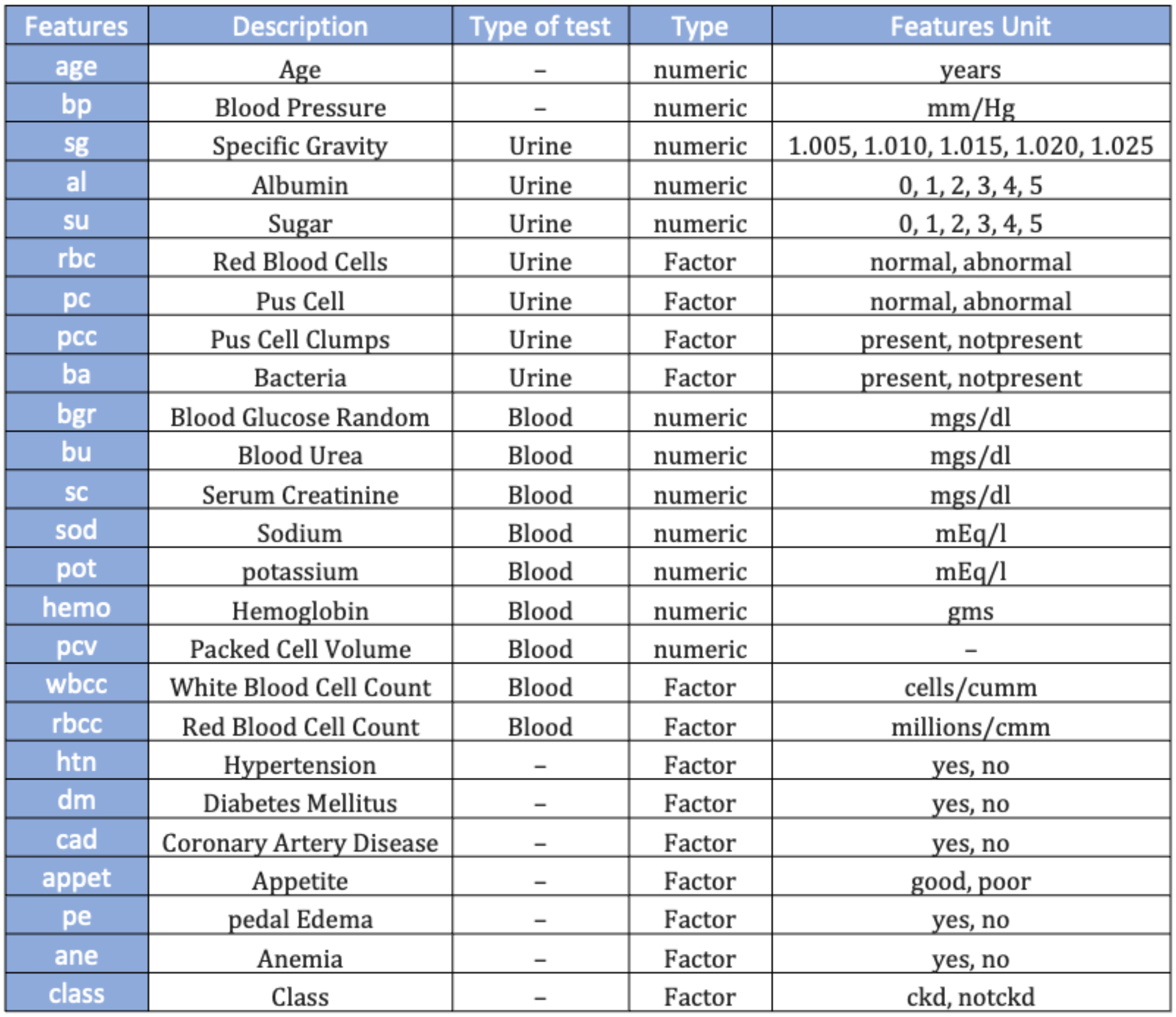
## List of Figures

## List of Tables

## Introduction

Most patients are not aware that they have chronic kidney disease (CDC, 2019). When patients perform a routine medical checkup, their blood and urine characteristics could suggest whether they have a high likelihood of having the disease. Predictive modeling, as part of preventive care, can be utilized for initial screening and help doctors to be pro-active and perform required testing and treatments. Below is a list of characteristics that are gathered when the Blood and Urine tests are performed, along with relevant patient histories.

*Table 1 - List of Features in Kidney Disease Dataset*



| Features | Description | Type of test | Type | Features Unit |
|---|---|---|---|---|
| age | Age | – | numeric | years |
| bp | Blood Pressure | – | numeric | mm/Hg |
| sg | Specific Gravity | Urine | numeric | 1.005, 1.010, 1.015, 1.020, 1.025 |
| al | Albumin | Urine | numeric | 0, 1, 2, 3, 4, 5 |
| su | Sugar | Urine | numeric | 0, 1, 2, 3, 4, 5 |
| rbc | Red Blood Cells | Urine | Factor | normal, abnormal |
| pc | Pus Cell | Urine | Factor | normal, abnormal |
| pcc | Pus Cell Clumps | Urine | Factor | present, notpresent |
| ba | Bacteria | Urine | Factor | present, notpresent |
| bgr | Blood Glucose Random | Blood | numeric | mgs/dl |
| bu | Blood Urea | Blood | numeric | mgs/dl |
| sc | Serum Creatinine | Blood | numeric | mgs/dl |
| sod | Sodium | Blood | numeric | mEq/l |
| pot | potassium | Blood | numeric | mEq/l |
| hemo | Hemoglobin | Blood | numeric | gms |
| pcv | Packed Cell Volume | Blood | numeric | – |
| wbcc | White Blood Cell Count | Blood | Factor | cells/cumm |
| rbcc | Red Blood Cell Count | Blood | Factor | millions/cmm |
| htn | Hypertension | – | Factor | yes, no |
| dm | Diabetes Mellitus | – | Factor | yes, no |
| cad | Coronary Artery Disease | – | Factor | yes, no |
| appet | Appetite | – | Factor | good, poor |
| pe | pedal Edema | – | Factor | yes, no |
| ane | Anemia | – | Factor | yes, no |
| class | Class | – | Factor | ckd, notckd |

This project consists of multiple stages of preprocessing, splitting data, modeling, etc. Below shows the workflow of this project.

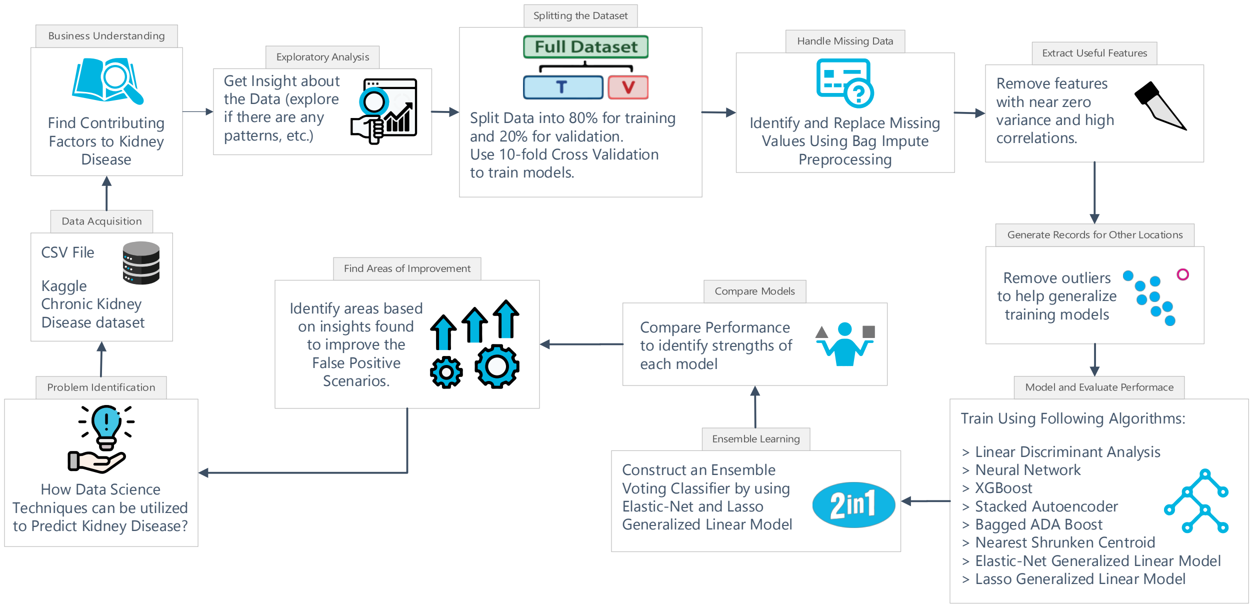

Business Understanding
Find Contributing Factors to Kidney Disease
Exploratory Analysis
Get Insight about the Data (explore if there are any patterns, etc.)
Splitting the Dataset
Full Dataset
T
V
Split Data into 80% for training and 20% for validation.
Use 10-fold Cross Validation to train models.
Handle Missing Data
Identify and Replace Missing Values Using Bag Impute Preprocessing
Extract Useful Features
Remove features with near zero variance and high correlations.
Data Acquisition
CSV File
Kaggle Chronic Kidney Disease dataset
Find Areas of Improvement
Identify areas based on insights found to improve the False Positive Scenarios.
Compare Models
Compare Performance to identify strengths of each model
Generate Records for Other Locations
Remove outliers to help generalize training models
Problem Identification
How Data Science Techniques can be utilized to Predict Kidney Disease?
Ensemble Learning
Construct an Ensemble Voting Classifier by using Elastic-Net and Lasso Generalized Linear Model
2in1
Model and Evaluate Performace
Train Using Following Algorithms:
> Linear Discriminant Analysis
> Neural Network
> XGBoost
> Stacked Autoencoder
> Bagged ADA Boost
> Nearest Shrunken Centroid
> Elastic-Net Generalized Linear Model
> Lasso Generalized Linear Model

*Figure 1 – Project Life Cycle*

## Methodology

This paper explores the classification of Chronic Kidney Disease from a clinical cohort in India which collected bloodwork, along with age and blood pressure. Our goal is to use this data to accurately classify Chronic Kidney Disease. The data was obtained from Kaggle dataset, which consists of 400 observations across 26 variables (refer to appendix for the list of features along with their description and unit).

Data in the bloodwork panels includes, but is not limited to, sugar, glucose, red blood cells, bacteria, pus cells, potassium hemoglobin, white blood cell count, and albumin. Outside of the bloodwork, other routine variables were collected as well such as age, blood pressure, and history of diabetes or coronary artery disease. Performance of model listed below were compared. The models are: Linear Discriminant Analysis, Neural Network, Stacked Autoencoder Deep Neural Network, XGBoost, Bagged ADA Boost, Nearest Shrunken Centroids, Lasso and Elastic-net Regularized and Generalized Linear. For consistency, each model was trained on the same data preprocessing and 80/20 split.

## Missing Data

The dataset contains several blank spaces, NAs, and “?” as missing data. The “?” and blank spaces were removed by using simple *if* checks. The top five columns that had the greatest number of missing values were Red Blood Cells (rbc), Red Blood Cell Count (rc), White Blood Cell Count (wc), Potassium (pot), and Sodium (sod).

The Figure 2 represents the missing value in the dataset shared between these columns, along with a mini bar chart that shows the missing values for each of the top five columns. Red Blood Cell column has the greatest number of missing values. There are 28 records where all five of these columns have null values.

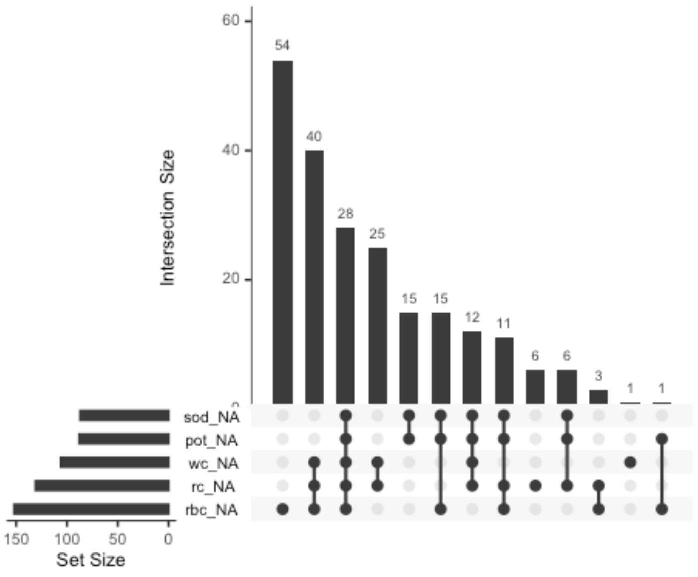


*Figure 2 - Missing Values*

## Linear Discriminant Analysis

A Linear Discriminant Analysis (LDA) was performed as a baseline model. LDA, when used for classification, is an algorithm that minimizes the probability of misclassification, which is dependent on multivariate predictor distribution and the class probabilities (Welch, 1939). Utilizing LDA on the test dataset, an Accuracy of 83%, a Sensitivity of 66%, and a Specificity of 100% was achieved.

## Neural Network

A Neural Network model was used from the caret package. Neural Networks are a class of machine learning algorithms that use a series of nodes and layers to find underlying structures in data, distantly resembling the processes of the human brain. The two available tuning parameters were utilized, which are the size of the hidden layers and the decay, which is the regularization parameter designed to prevent overfitting. Performance on the test dataset resulted in an accuracy of 91%, a Sensitivity of 86%, and a Specificity of 97%.

## Stacked Autoencoder Deep Neural Network.

Stacked Autoencoders are a type of deep learning neural network model that uses multiple (or stacked) autoencoders, which form compressed representations of the original data using a "bottleneck" of encoded layers (Lopez Pinaya et al., 2020). This is an unsupervised, feed-forward system, where input and output are equal. For tuning parameters, a grid of varied sizes for the three hidden layers was evaluated, as well as the hidden and visible dropout layers. On the test data, it was observed that Accuracy of 89%, a Sensitivity of 78%, and a Specificity of 100%.

## XGBoost

XGBoost model in caret was used, which stands for eXtreme Gradient Boosting, and is an ensemble machine learning algorithm utilizing many parallel decision trees (Chen & Guestrin, 2016). This model has seven tuning parameters which were used a range of values for, including nrounds (number of decision trees in final model, max_depth (maximum depth of the tree), eta (step size shrinkage), gamma (minimum loss reduction), colsample_bytree (subsample ratio of columns when constructing the tree), min_child_weight (minimum sum of instance weight in a child), and subsample (subsample ratio of training instances). Performance of the XGBoost model on test data resulted in an Accuracy of 95%, a Sensitivity of 90%, and a Specificity of 100%.

## Bagged ADA Boost

Bagged ADA Boost, or Adaptive Boosting, was used here as well. ADA Boost is an ensemble learning method for performance boosting with decision trees, like XGBoost (CAO et al., 2013). In this case, the ADA Boost model was bagged, meaning it utilized Bootstrap aggregation to help further improve performance. Two tuning parameters, mfinal, which is the number of trees used for the final model, and maxdepth, which is the maximum depth of the trees. Performance of the ADA Boost model on test data resulted in an Accuracy of 97%, a Sensitivity of 94%, and a Specificity of 100%.

## Nearest Shrunken Centroids

Nearest Shrunken Centroids (NSC) model is a linear classification model where for each class, the centroid is found based on the average value of each class of the predictor (Kuhn & Johnson, 2013). For tuning, the threshold was utilized, which effectively shrinks the centroids for each class towards the overall centroid. On the test dataset, this model returned an Accuracy of 89%, a Sensitivity of 78%, and a Specificity of 100%.

## Lasso

Lasso, or *least absolute shrinkage and selection operator*, is a model that penalizes absolute values, using regularization to improve the model and for feature extraction in generalized linear models (Kuhn & Johnson, 2013). On the test dataset, this model returned an Accuracy of 93%, a Sensitivity of 90%, and a. Specificity of 97%.

## Elastic net

Elastic net is a generalization of the Lasso and combines the two penalties from Lasso and Ridge Regression into one (Kuhn & Johnson, 2013). It uses the regularization from ridge regression and utilizes the feature selection of the lasso penalty. The two tuning parameters used were fraction, which is the ratio of the L1 norm for the coefficient, compared with the norm of the LS solution, and Lambda, which is the penalty for the quadratic parameter. On the test data, this model returned an Accuracy of 89%, a Sensitivity of 78%, and a Specificity of 100%.

### Ensemble

An Ensemble model was run as well, which combined the Lasso and Elastic net models together, to see if it could increase performance. The caretEnsemble package was utilized to run the ensemble. This, however, did not better the individual models very much, with an Accuracy of 93%, a Sensitivity of 90%, and a Specificity of 97%.

## Results

Nine models were reviewed using accuracy, precision, recall, sensitivity, specificity and F1 (shown in Table 2). Accuracy was used to show the proportion of correct classifications. Sensitivity was used to measure each model's ability to correctly classify positive results, whereas specificity was chosen to measure correctly classified negative results. F1 score was used to measure each model’s precision and recall, which are listed separately as well. Based on these results, the Bagged Ada Boost had the best results, with an Accuracy of 97%, a Sensitivity of 94%, a Specificity of 100%, and an F1 score of 97%. The worst model was the Linear Discriminant Analysis, with an Accuracy of 83%, a Sensitivity of 66%, a Specificity of 100%, and an F1 score of 80%, which was used as our baseline. According to our ADA Boost model, the most important variables were serum creatinine, albumin, and red blood cell count, highlighting these variables as potential prognostic markers for Chronic Kidney Disease.

*Table 2 - Model Evaluation Table for Chronic Kidney Disease*

| Model | Accuracy | Precision | Recall | Sensitivity | Specificity | F1 |
|---|---|---|---|---|---|---|
| Linear Discriminant Analysis | 0.83 | 1.00 | 0.66 | 0.66 | 1.00 | 0.80 |
| Neural Network | 0.91 | 0.98 | 0.86 | 0.86 | 0.97 | 0.91 |
| XGBoost | 0.95 | 1.00 | 0.90 | 0.90 | 1.00 | 0.95 |
| Stacked Autoencoder | 0.89 | 1.00 | 0.78 | 0.78 | 1.00 | 0.88 |
| Bagged ADA Boost | 0.97 | 1.00 | 0.94 | 0.94 | 1.00 | 0.97 |
| Nearest Shrunken Centroid | 0.89 | 1.00 | 0.78 | 0.78 | 1.00 | 0.88 |
| Lasso | 0.93 | 0.98 | 0.90 | 0.90 | 0.97 | 0.94 |
| Generalized Linear Model | 0.89 | 1.00 | 0.78 | 0.78 | 1.00 | 0.88 |
| Ensemble (glm + glmnet) | 0.93 | 0.98 | 0.90 | 0.90 | 0.97 | 0.94 |

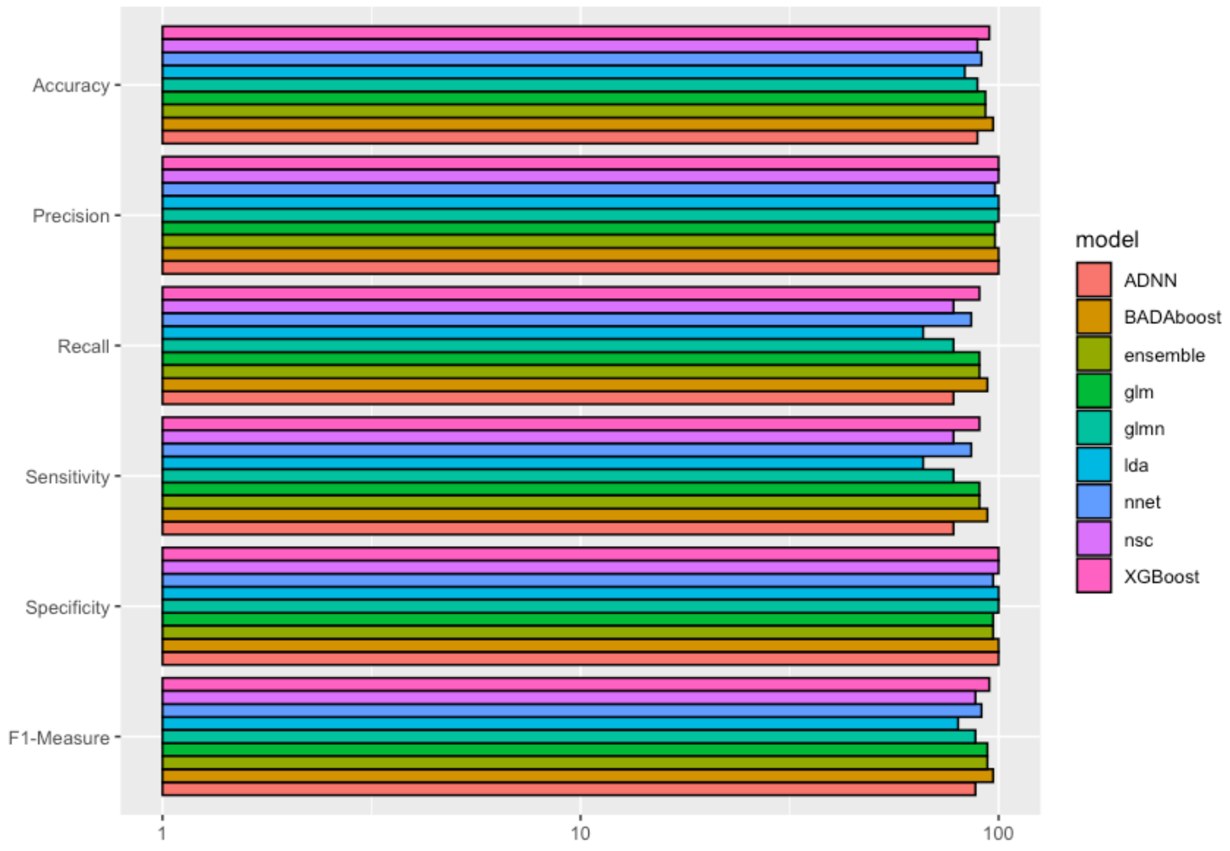


*Figure 3 - Model Performance Metrices Bar plot*

## Conclusion

Based on these results, it is recommended to utilize Bagged ADA Boost machine learning models as part of the initial screening for chronic kidney disease. With an accuracy of 97% and sensitivity of 94%, clinicians could utilize this model for screening patients at risk of chronic kidney disease during routine medical checkups.

For more information refer to GitHub page at:

github.com/nimaamintaghavi/predictive_modeling_of_kidney_disease

## References

CAO, Y., MIAO, Q.-G., LIU, J.-C., & GAO, L. (2013). Advance and Prospects of AdaBoost Algorithm. *Acta Automatica Sinica*, *39*(6), 745–758. https://doi.org/10.1016/S1874-1029(13)60052-X

*Centers for Disease Control and Prevention. Chronic Kidney Disease Surveillance System*. (2019). Centers for Disease Control.

Chen, T., & Guestrin, C. (2016). XGBoost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020). Autoencoders. In *Machine Learning* (pp. 193–208). Elsevier. https://doi.org/10.1016/B978-0-12-815739-8.00011-0

Silverstein, D. M. (2009). Inflammation in chronic kidney disease: role in the progression of renal and cardiovascular disease. *Pediatric Nephrology*, *24*(8), 1445–1452. https://doi.org/10.1007/s00467-008-1046-0

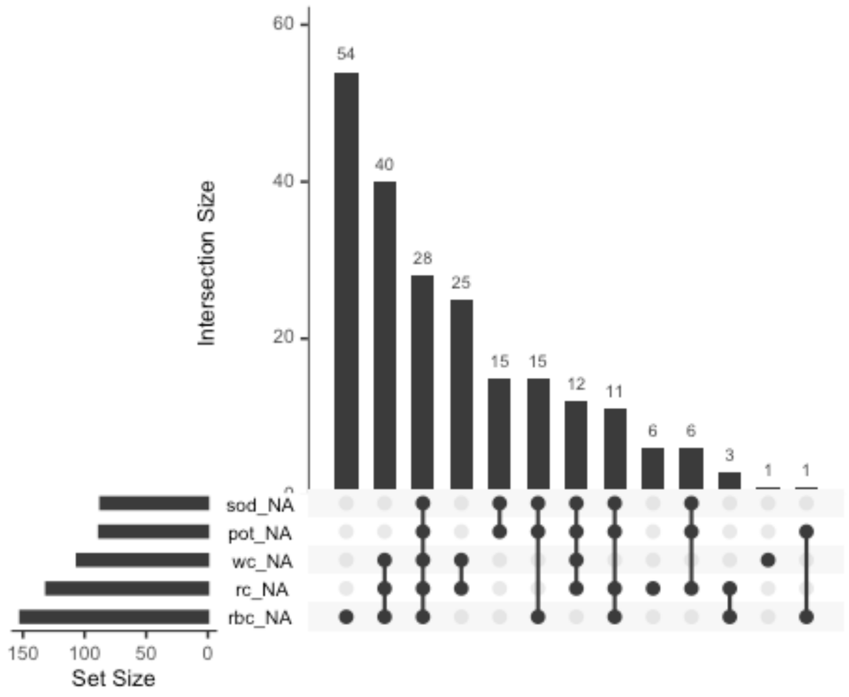
Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. Springer New York. https://doi.org/10.1007/978-1-4614-6849-3

## Appendix

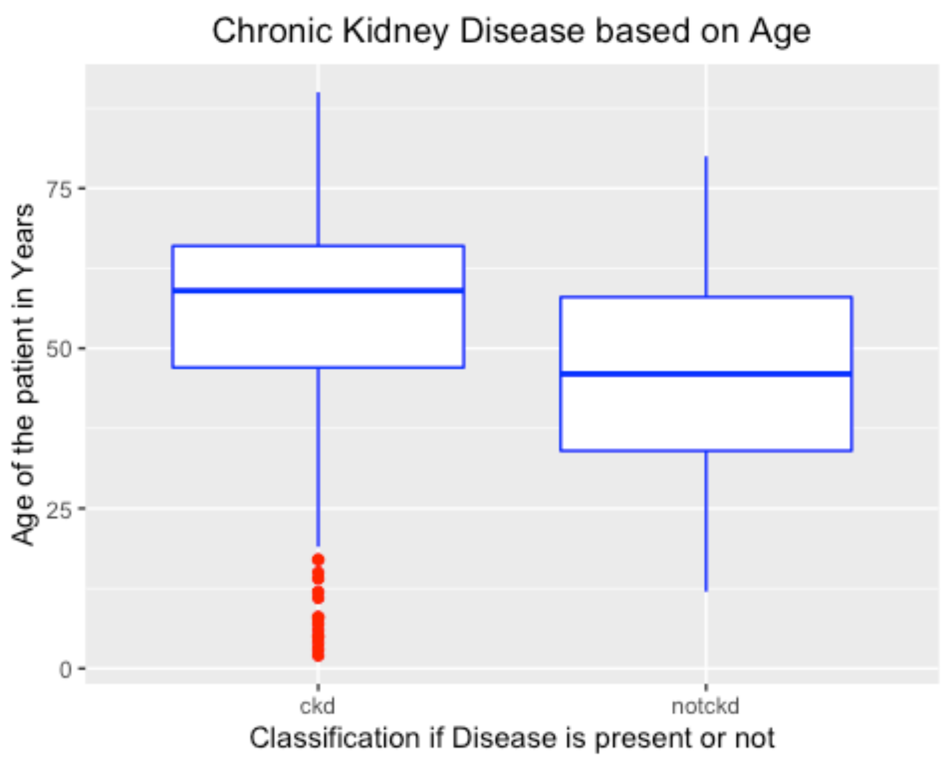
```
library(caret)
library(RANN)
library(corrplot)
library(ggplot2)
library(naniar)
library(caretEnsemble)
library(reshape2)

#Read in the dataset
df_org = read.csv('kidney_disease.csv')
df = df_org



# Replace the {tab} character (which is unnecessary) with nothing for all records
for (c in colnames(df)) df[,c] = gsub('\t', '', df[,c])

# Replace Empty values with NA - this will also take care of the records that have
# question mark (?) as its value.
for (c in colnames(df)) df[which(!df[,c] >= 0), c] = NA

#Check for NAs
percent = function(x, digits = 0, format = "f", ...) {
       paste0(formatC(100 * x, format = format, digits = digits, ...), "%")
}

empty_count_table = data.frame(count = colSums(is.na(df)))
empty_count_table$percent_empty = percent(empty_count_table$count / nrow(df))
head(empty_count_table[order(empty_count_table[1], decreasing = TRUE),], 10)

##       count percent_empty
## rbc     152           38%
## rc      131           33%
## wc      106           26%
## pot      88           22%
## sod      87           22%
## pcv      71           18%
## pc       65           16%
## hemo     52           13%
## su       49           12%
## sg       47           12%
```

```
# UpSetR package can be used to visualize the patterns of missingness, or the combinations of
missingness across cases. To see combinations of missingness and intersections of missingness
amongst variables, gg_miss_upset function was used

gg_miss_upset(df, nsets=5)
```

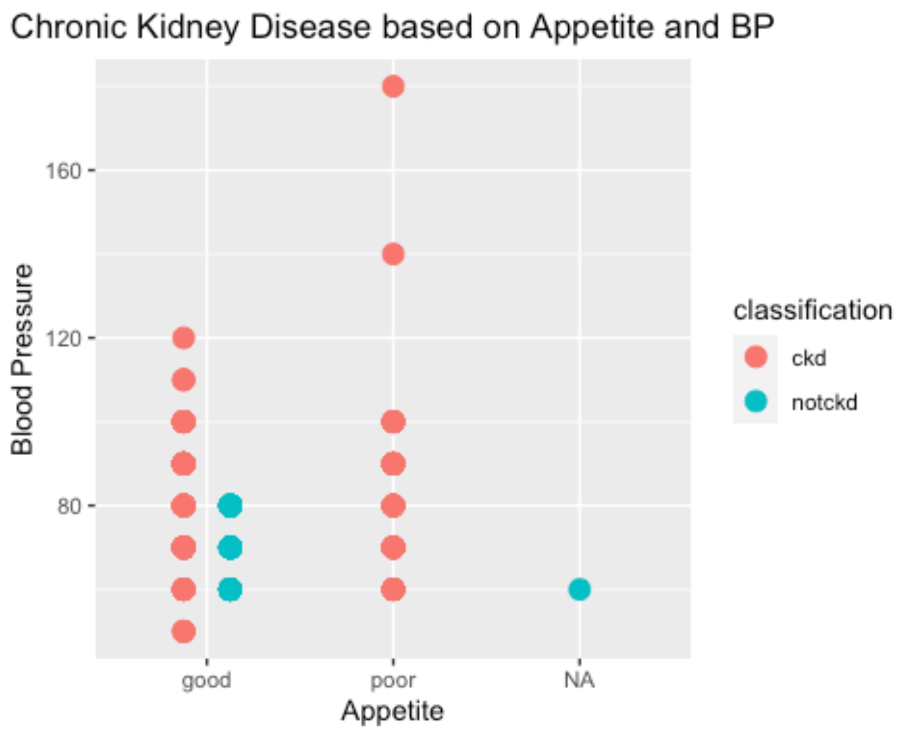
```
#This tells us:

#Bottom Left Corner bar plot shows the top 5 features/variables with missing values
#rbc has the most missing values
#There is 1 case where all top 5 variables have missing values together
```



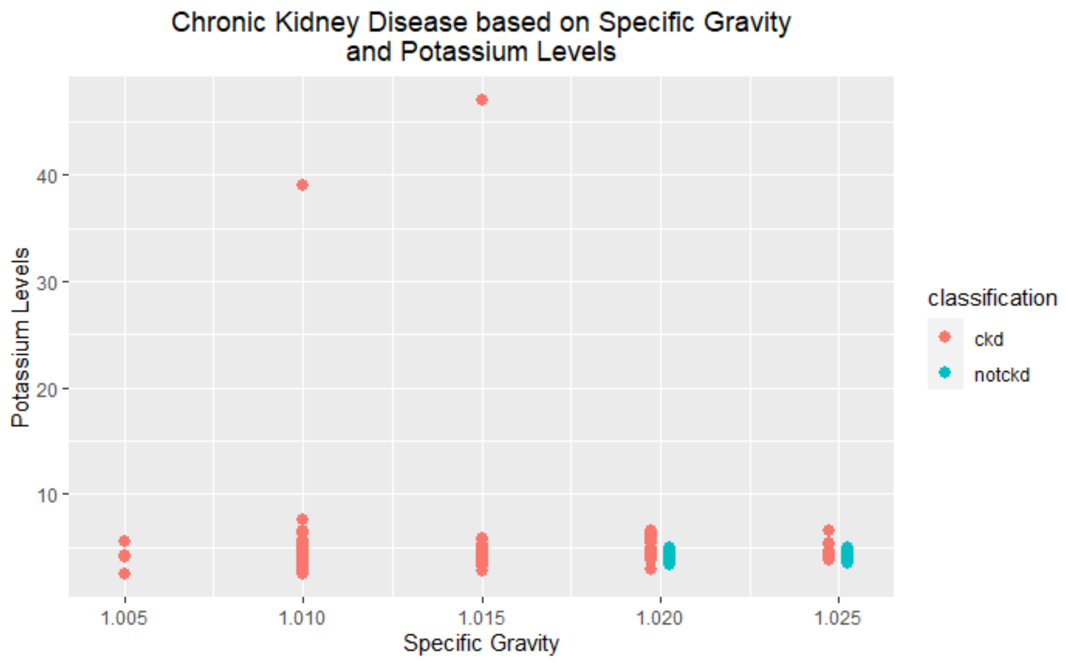
```
cols_numeric = c('age','bp','sg','al','su','bgr','bu','sc','sod','pot','hemo','pcv','wc','rc')

for (c in colnames(df)){
       if (c %in% cols_numeric) df[,c] = sapply(df[,c], as.numeric)
       else df[,c] = sapply(df[,c], as.factor)
}

#Plots for final
#Checking relation between features

ggplot(df, aes(classification, age)) +
       geom_boxplot(colour = "blue", outlier.colour = "red") +
       ggtitle("Chronic Kidney Disease based on Age") +
       xlab("Classification if Disease is present or not") +
       ylab("Age of the patient in Years") +
       theme(plot.title = element_text(hjust = 0.5))
```



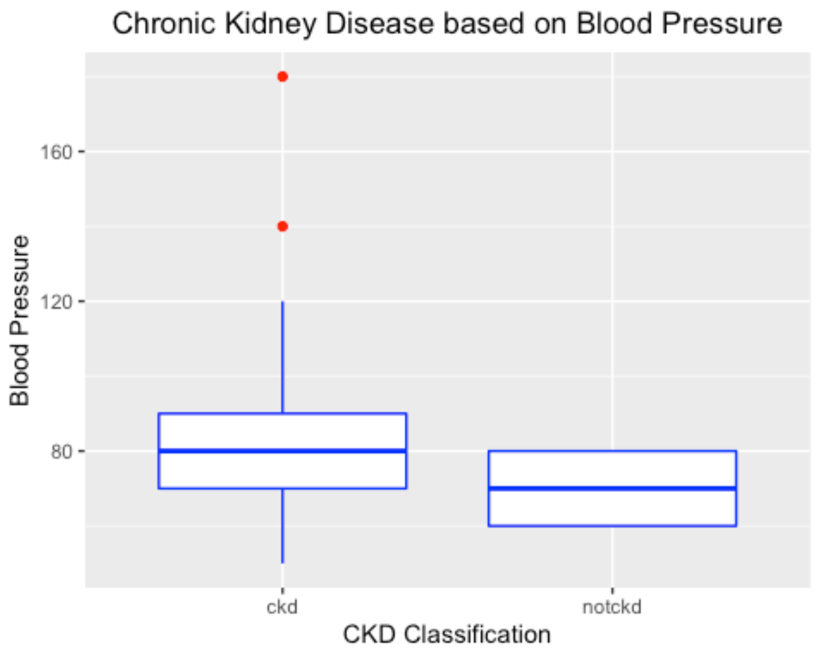
```
ggplot(df, aes(appet, bp, colour = classification)) +
       geom_point(size = 3.5, position = position_dodge(width = 0.5)) +
       ggtitle("Chronic Kidney Disease based on Appetite and BP") +
       xlab("Appetite") +
       ylab("Blood Pressure") +
       theme(plot.title = element_text(hjust = 0.5))
```



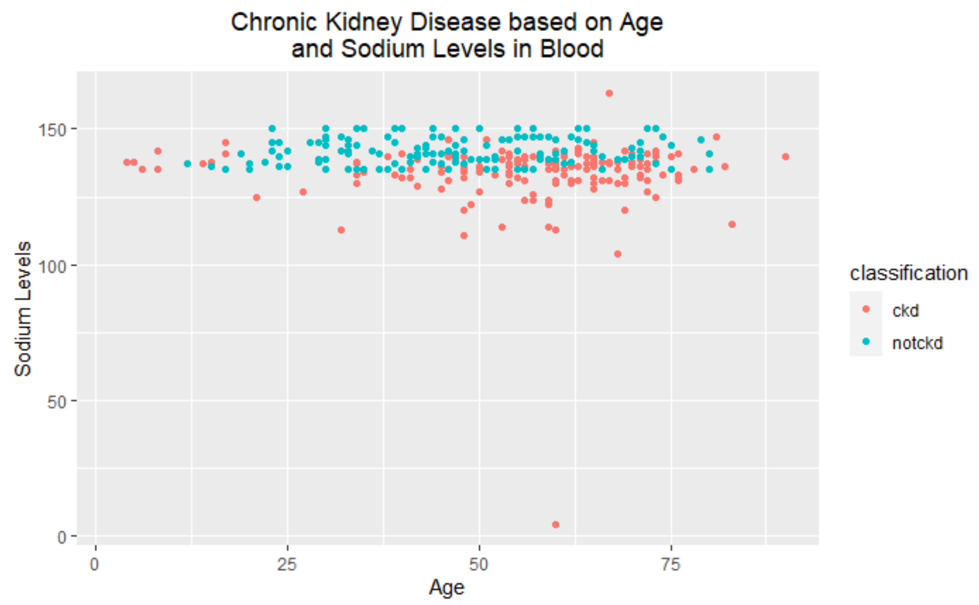
```
geom_point(size = 2.5, position = position_dodge(width = 0.001)) +
       ggtitle("Chronic Kidney Disease based on Specific Gravity\nand Potassium Levels") +
       xlab("Specific Gravity") +
       ylab("Potassium Levels") +
       theme(plot.title = element_text(hjust = 0.5))
```



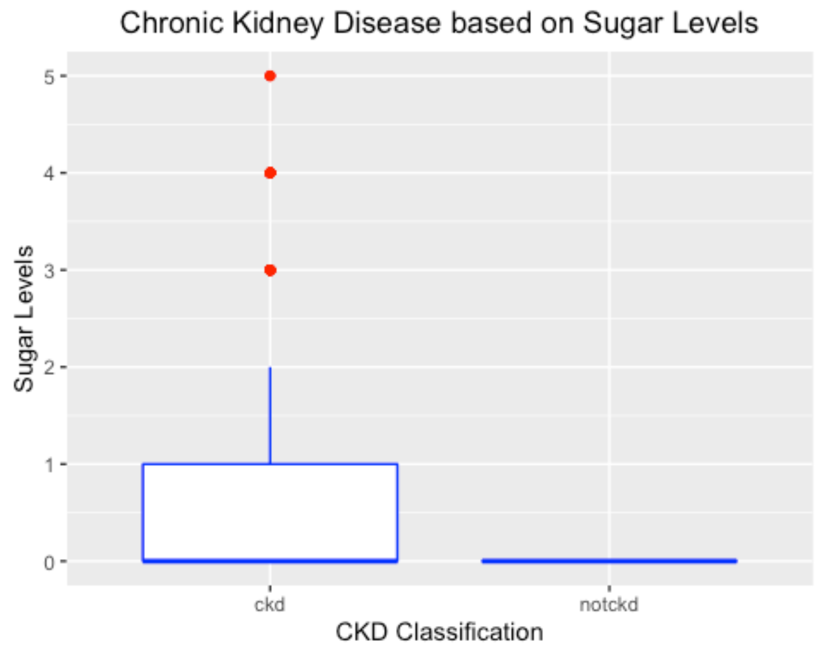
```
ggplot(df, aes(classification, bp)) +
        geom_boxplot(colour = "Blue", outlier.colour = "red") +
 ggtitle("Chronic Kidney Disease based on Blood Pressure") +
        xlab("CKD Classification") +
        ylab("Blood Pressure") +
        theme(plot.title = element_text(hjust = 0.5))
```



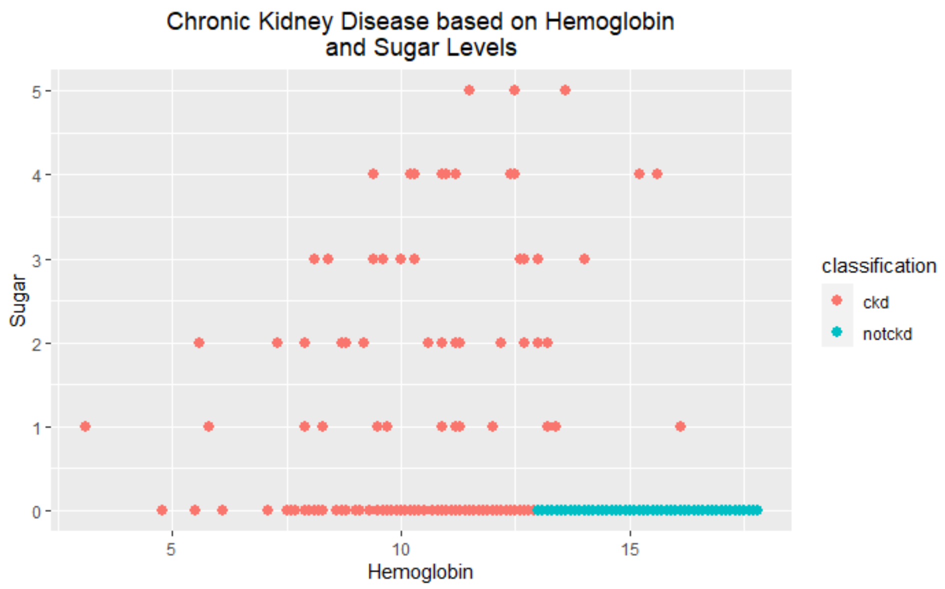
```
ggplot(df, aes(age, sod, colour = classification)) +
       geom_point(size = 1.5) +
        ggtitle("Chronic Kidney Disease based on Age\nand Sodium Levels in Blood") +
        xlab("Age") +
        ylab("Sodium Levels") +
        theme(plot.title = element_text(hjust = 0.5))
```



```
ggplot(df, aes(classification, su)) +
        geom_boxplot(colour = "Blue", outlier.colour = "red") +
       ggtitle("Chronic Kidney Disease based on Sugar Levels") +
        xlab("CKD Classification") +
        ylab("Sugar Levels") +
        theme(plot.title = element_text(hjust = 0.5))
```

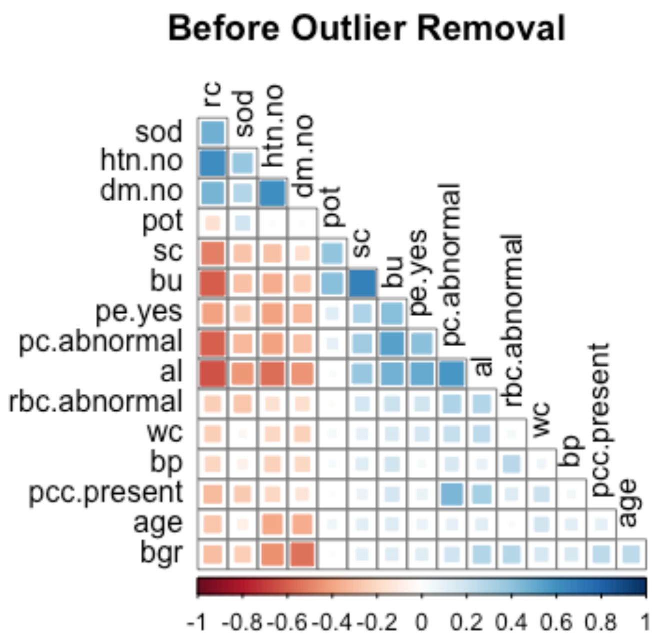


```
ggplot(df, aes(hemo, su, colour = classification, width = 0.75, position = "dodge")) +
        geom_point(size = 2.5, position = position_dodge(width = 0.001)) +
       ggtitle("Chronic Kidney Disease based on Hemoglobin\nand Sugar Levels") +
        xlab("Hemoglobin") +
        ylab("Sugar") +
       theme(plot.title = element_text(hjust = 0.5))
```

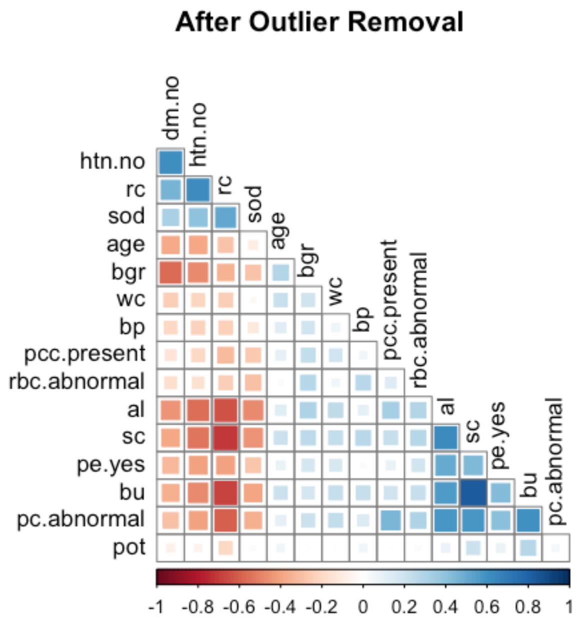


```r
set.seed(110)

cols_with_nearZeroVar = nearZeroVar(df, freqCut = 90/10)
df = df[,-cols_with_nearZeroVar]

trainingRows = createDataPartition(df$classification, p = .8, list = FALSE)
# Subset the data into training and testing.
trainSet = df[trainingRows, ]
testSet  = df[-trainingRows, ]

classification_proportion = summary(trainSet$classification)


trainSet_features = subset (trainSet, select = -c(id, ane, sg, hemo, appet, classification))

dummy = dummyVars(" ~ .", data=trainSet_features)
trainSet_features =  data.frame(predict(dummy, newdata = trainSet_features))

# Replace missing values using imputation
bagImpute_model = preProcess(trainSet_features, method = "bagImpute")
trainSet_features = predict(bagImpute_model, trainSet_features)

# find highly correlated predictors
highCorr = findCorrelation(cor(trainSet_features), cutoff = .75)

trainSet_features = trainSet_features[, -highCorr] # remove highly correlated predictors

# smooth out the extremes
remove_outliers = function(df, cols) {
    for (col in cols) {
        IQR_value = IQR(df[,col])
        low_side = quantile(unlist(df[,col]), probs = 0.25) - (1.5 * IQR_value)
        high_side = quantile(unlist(df[,col]), probs = 0.75) + (1.5 * IQR_value)

        df[which(df[col] <= low_side), col] = low_side
        df[which(df[col] >= high_side), col] = high_side
    }
return (df)
}


cols_numeric = c('age','bp','al','bgr','bu','sc','sod','pot','wc','rc')
trainSet_features_removed_outliers = remove_outliers(trainSet_features,cols_numeric)
```

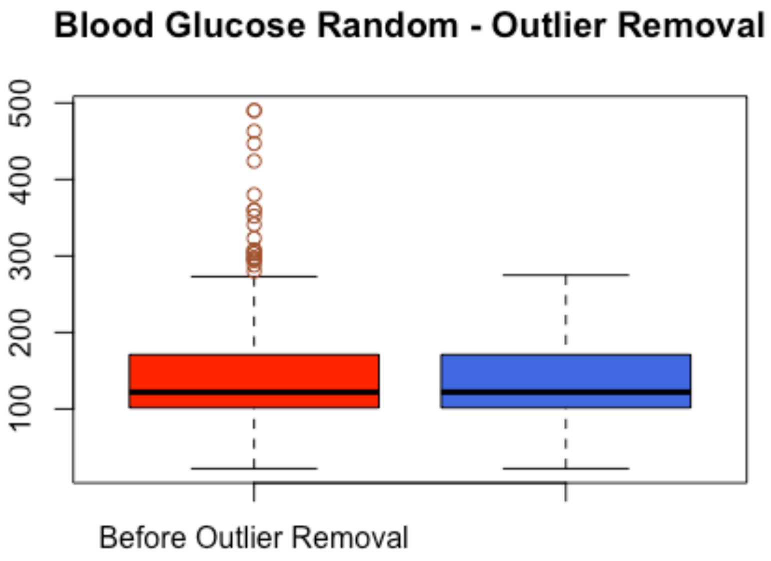
```
corrplot( cor( trainSet_features ),  #main = "Before Outlier Removal",
          method = 'square', order = 'AOE', type = 'lower', diag = FALSE,
          addgrid.col = "gray50", tl.cex=1, tl.col = "black")


title("Before Outlier Removal", line = 1)
```



```
corrplot( cor( trainSet_features_removed_outliers ), #main = "After Outlier Removal",
          method = 'square', order = 'AOE', type = 'lower', diag = FALSE,
          addgrid.col = "gray50", tl.cex=1, tl.col = "black")
title("After Outlier Removal", line = 1)
```



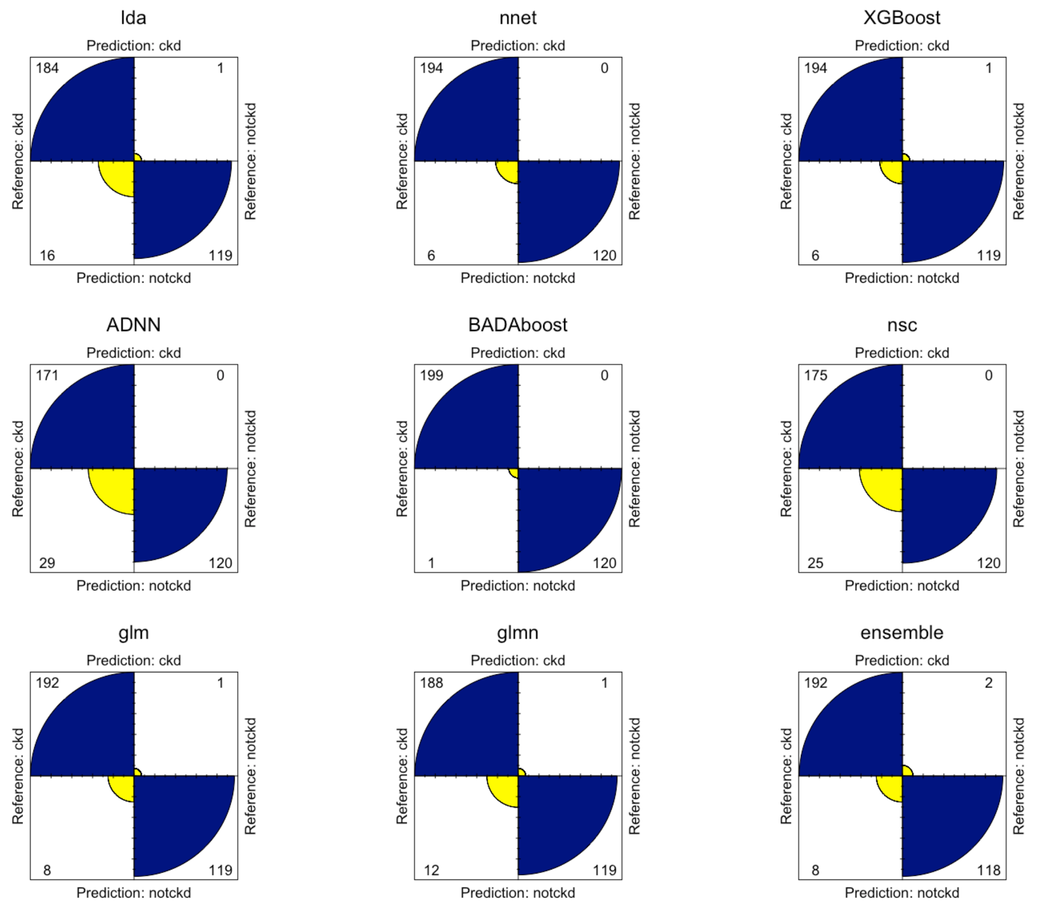
```
boxplot(trainSet_features$bgr, trainSet_features_removed_outliers$bgr,

        col= c("red", "royalblue"),
        main = ("Blood Glucose Random - Outlier Removal"),
        names = c("Before Outlier Removal", "After Outlier Removal"),
        outcol="sienna")
```



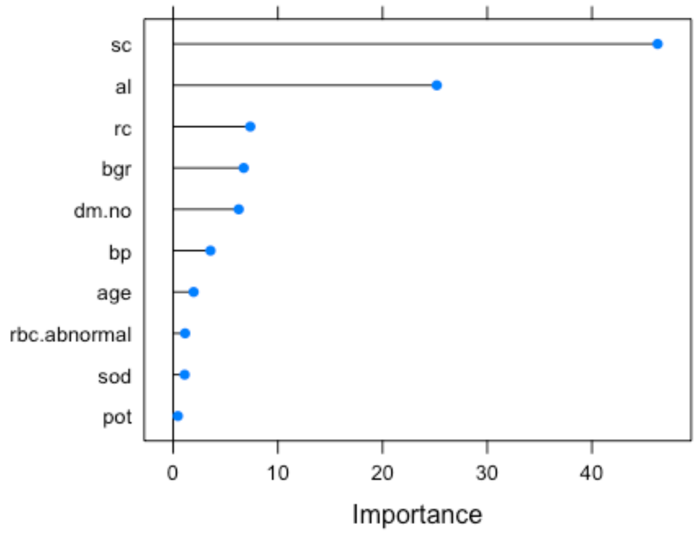
```
#Removing outliers helps in generalization of data set to avoid over fitting that data.

trainSet_features = trainSet_features_removed_outliers

trainSet = cbind(trainSet_features, classification = trainSet$classification)

#Stacked Autoencoder Deep Neural Network

ctrl = trainControl(method = "cv", number=10,
                  summaryFunction = twoClassSummary,
                  classProbs = TRUE, savePredictions = TRUE)

# Neural Network
nnetGrid = expand.grid(decay = c(0, 0.01, .1), size = c(3, 7, 10, 15))
model_nnet = train(classification ~ ., data = trainSet, method = "nnet",
                   preProc = c("center", "scale"), tuneGrid = nnetGrid,
                   trControl = ctrl, trace = FALSE, metric = "ROC",
                   MaxNWts = 15 * (ncol(trainSet) + 1) + 15 + 1,
                   maxit = 1000)

# Linear Discriminant Analysis
model_lda = train(classification ~ ., data = trainSet, method = "lda",
                  preProc = c("center", "scale"),
                  metric = "ROC", trControl = ctrl)
```

```
# GX Boost
XGGrid = expand.grid(nrounds = c(1, 20),
                     max_depth = c(1, 9),
                     eta = c(.1, .8),
                     gamma = 0,
                     colsample_bytree = 1,
                     min_child_weight = 2,
                     subsample = c(.5, 1))

model_XGBoost = train(classification ~ ., data = trainSet, method = "xgbTree",
                      trControl = ctrl,
                      metric = "ROC",
                      preProc = c("center", "scale"),
                      tuneGrid = XGGrid)

# Stacked Autoencoder Deep Neural Network
DNNTune = expand.grid(layer1 = c(75,150),
                      layer2 = c(0,75,150),
                      layer3 = c(0,75,150),
                      hidden_dropout = c(0, 0.5),
                      visible_dropout = c(0, 0.5))

model_ADNN = train(classification ~ ., data = trainSet,method="dnn",
                      metric = "ROC",
                      tuneGrid=DNNTune,
                      trControl= ctrl,
                      preProc = c("scale", "center"))

#Bagged ADA Boost Model
BADAgrid = expand.grid(mfinal = (1:10), maxdepth = c(1, 10))

model_BADAboost = train(classification ~ ., data = trainSet, method = "AdaBag",
                        trControl = ctrl,
                        tuneGrid = BADAgrid,
                        metric = "ROC",
                        preProc = c("center", "scale"))

# Nearest Shrunken Centroids
pamGrid = data.frame(threshold = seq(0, 25, length = 20))
model_nsc = train(classification ~ ., data=trainSet, method="pam",
                  preProc = c("center", "scale"), tuneGrid = pamGrid,
                  metric = "ROC", trControl = ctrl)

# Elastic-Net Generalized Linear Model
glmnGrid = expand.grid(alpha = c(0, .1, .2, .4, .6, .8, 1),
                       lambda = seq(.01, .2, length = 10))

model_glmn = train(classification ~ ., data = trainSet, method = "glmnet",
                   preProc = c("center", "scale"), tuneGrid = glmnGrid,
                   metric = "ROC", trControl = ctrl)

# Lasso Generalized Linear Model
model_glm = train(classification ~ ., data=trainSet, method="glm",
                   metric = "ROC", trControl = ctrl)

# Ensemble Model between GLM and GLMNet
model_list = caretList(classification~., data=trainSet, metric="ROC",
                       trControl=ctrl, methodList=c("glm", "glmnet"))


model_ensemble = caretEnsemble(model_list, metric="ROC", trControl=ctrl)
```

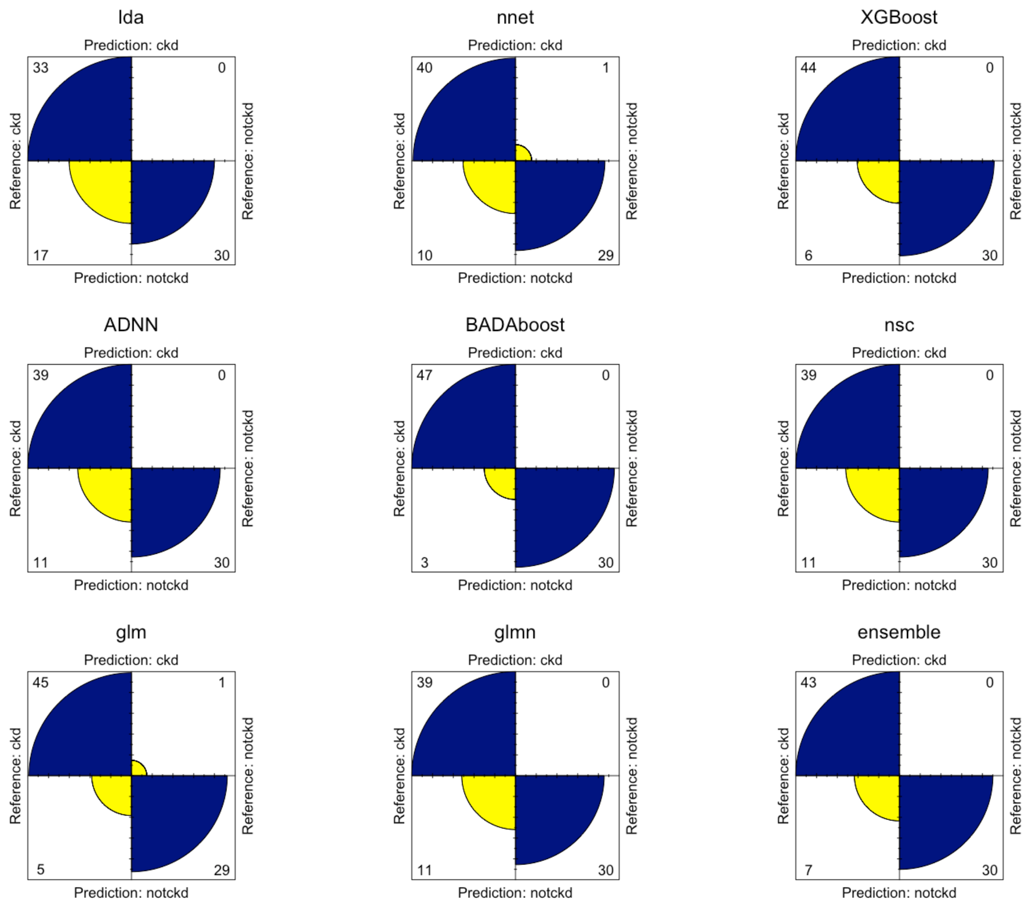
```r
# Check performance of each model
eval = data.frame(actual = trainSet$classification)


eval$lda = predict(model_lda, trainSet_features)
eval$nnet = predict(model_nnet, trainSet_features)
eval$XGBoost = predict(model_XGBoost, trainSet_features)
eval$ADNN = predict(model_ADNN, trainSet_features)
eval$BADAboost = predict(model_BADAboost, trainSet_features)
eval$nsc = predict(model_nsc, trainSet_features)
eval$glm = predict(model_glm, trainSet_features)
eval$glmn = predict(model_glmn, trainSet_features)
eval$ensemble = predict(model_ensemble, trainSet_features)

visualize_confusion_matrix = function (df){
  for (c in colnames(eval)){
        cfmx = confusionMatrix(df[,c], df$actual, positive = "ckd")

        fourfoldplot(cfmx$table, color = c("yellow", "navyblue"),
                 conf.level = 0, margin = 1, main = paste(c))
  }
}
# Check Performance based on Train Data
visualize_confusion_matrix(eval)
```



```
# Since Bagged ADA Boost is the best model compared to others, check variable importance
BADAimportance = varImp(model_BADAboost, scale = FALSE)
plot(BADAimportance, top = 10)
```

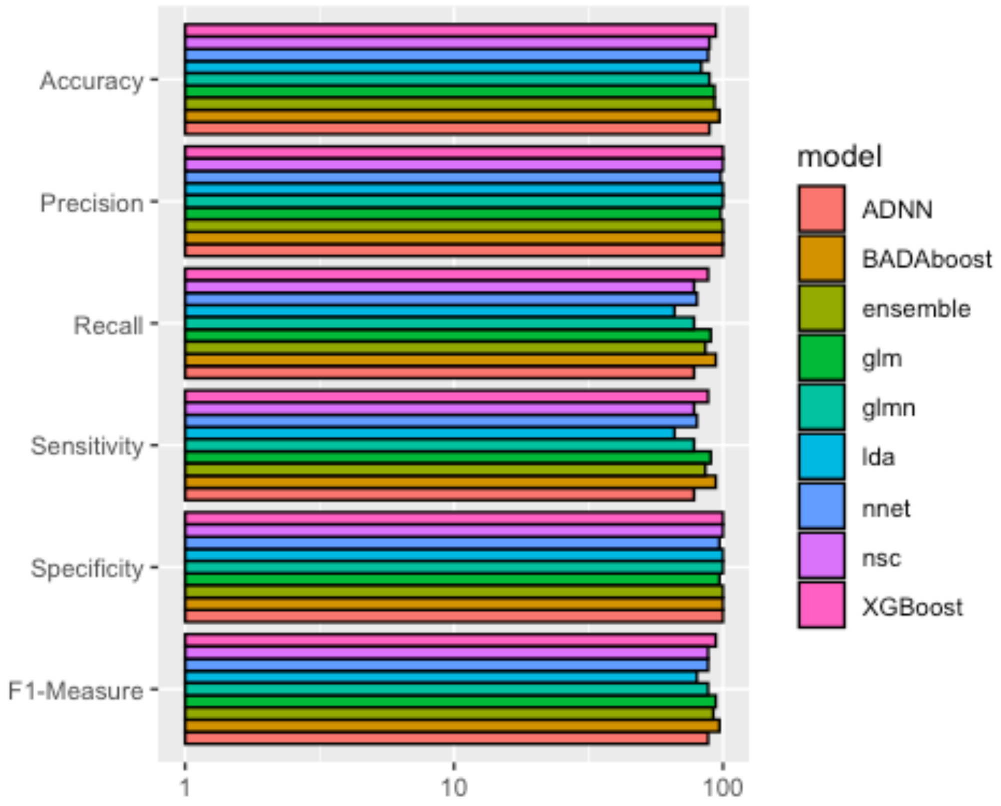


```
dummy = dummyVars(" ~ .", data=testSet)
testSet_features =  data.frame(predict(dummy, newdata = testSet))

testSet_features = testSet_features[colnames(trainSet_features)]

bagImpute_model = preProcess(testSet_features, method = "bagImpute")
testSet_features = predict(bagImpute_model, testSet_features)

eval_test = data.frame(actual = testSet$classification)

eval_test$lda = predict(model_lda, testSet_features)
eval_test$nnet = predict(model_nnet, testSet_features)
eval_test$XGBoost = predict(model_XGBoost, testSet_features)
eval_test$ADNN = predict(model_ADNN, testSet_features)
eval_test$BADAboost = predict(model_BADAboost, testSet_features)
eval_test$nsc = predict(model_nsc, testSet_features)
eval_test$glm = predict(model_glm, testSet_features)
eval_test$glmn = predict(model_glmn, testSet_features)
eval_test$ensemble = predict(model_ensemble, testSet_features)

# Check Performance based on Test Data
visualize_confusion_matrix(eval_test)
```



```
# juxtapose performance metric for each model
# and tabulate for plotting purposes


tabulate_performance_metrics = function(eval_test){
  df = data.frame()
  for (col in colnames(eval_test)) {
        cnfmx = confusionMatrix(eval_test[,col], eval_test$actual, positive = "ckd")
        last_row = nrow(df)+1
        for (i in c(1:length(cnfmx$byClass))) {
               metric = cnfmx$byClass[i]
               name = unlist(strsplit(deparse(metric),'='))[1]
               for (str_replace in c('c', '"', ' ' , '(', '\" '))
                   name = sub(str_replace, '', name,fixed = TRUE)

               df[last_row,'model'] = col
               df[last_row,name] = round(metric,2)
               }
  }
  return(df)
}


performance_table = tabulate_performance_metrics(eval_test)
```

```
performance_list_to_show = data.frame(
  current = c('model','F1','Specificity','Sensitivity','Recall','Precision','BalancedAccuracy'),
  change_to = c('', 'F1-Measure','Specificity','Sensitivity','Recall','Precision','Accuracy')

  )

perf_test = melt(performance_table[performance_list_to_show$current],id.vars = 7)

ggplot(perf_test,aes(x = variable, fill = model ,y = value*100)) +
       geom_bar(stat = "identity",position = "dodge", color = 'black') + scale_y_log10() +
       scale_x_discrete(labels = performance_list_to_show$change_to[-1]) +
       xlab(NULL) + ylab(NULL) + coord_flip()
```


Accuracy
Precision
Recall
Sensitivity
Specificity
F1-Measure
model
ADNN
BADAboost
ensemble
glm
glmn
lda
nnet
nsc
XGBoost