NETCapstone

Netaverner

13/12/2020

Overview

This project is related to the Choose-your-own project of the HervardX: PH125.9x Data Science: Capstone course.

The report starts by giving a general idea of the project. It then shows how the Dataset will be prepared and setup. An exploratory data analysis is carried out on the Dataset. This is performed in order to help facilitate the develop of a machine learning algorithm(s) that can predict whether the Biomechanical features of orthopedic patients is Abnormal or Normal. The results of the models created will be examined and explaing. Lastly, the report have some concluding remarks and ideas for future work.

Introduction

This project focuses on a Biomechanical features of orthopedic patients Dataset aqquired from kaggle (**???**). This data contains patients that have medical diagnosis that covers, Disk Hernia and Spondylolisthesis.

Spondylolisthesis is a spinal condition that affects the lower vertebrae (spinal bones). This disease causes one of the lower vertebrae to slip forward onto the bone directly beneath it. It's a painful condition but treatable in most cases (Moore 2018).

A herniated disk refers to a problem with one of the rubbery cushions (disks) that sit between the individual bones (vertebrae) that stack to make your spine (Staff, n.d.).

With the medical field introducing more and more machinery and computer oriented technology to perform diagnostics on the human body. This project tries to show that by using machine learning one should be able to having computers or other technology performing diagnosis on humans, without the need for human intervention. This could possibly help human doctors in the future as machines could possible achieve a medical diagnosis more rapidly, if provided with the needed inputs.

This project will make a performance comparison between different machine learning algorithms in order to to assess the correctness in classifying data with respect to efficiency and effectiveness of each algorithm in terms of accuracy, precision, sensitivity and specificity, in order to find the best class of patient.

The major models used and tested will be supervised learning models (algorithms that learn from labeled data), which are generally used in these kinds of data analysis.

Aim

The objective of this report is to train machine learning models to predict whether Biomechanical features of a orthopedic patient is Abnormal or Normal. Data will be transformed and its dimension reduced to reveal patterns in the Dataset and create a more robust analysis.

The optimal model will be selected by selecting the Model that produces the best results in the following categories: * accuracy * sensitivity * f1 score

Other factors will also be reviewed when select the optimal model.

Though the use of machine learning method the features of orthopedic patients will be extracted and classified. The goal is determine whether a given sample of patients have Normal or Abnormal features.

The machine learning models in this report try to create a classifier that provides a high accuracy level combined with a low rate of false-negatives (high sensitivity).

Dataset

The report covers the Biomechanical features of orthopedic patients Dataset acquired from (https://www.kaggle.com/uciml/biomechanical-features-of-orthopedic-patients) and created and maintained by UCI Machine Learning (Learning, n.d.).

This report focuses on the .csv file "column2Cweka.csv (file with two class labels)" this file contains the following:

The categories Disk Hernia and Spondylolisthesis were merged into a single category labelled as 'abnormal'. Thus, task consists in classifying patients as belonging to one out of two categories: Normal (100 patients) or Abnormal (210 patients).

The .csv format file containing the data is loaded from my personal computer, should one wish to run the .Rmd file please download the "column2Cweka.csv" from the URL displayed earlier and load it.

```r
#Libraries
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(corrplot)) install.packages("recorrplotadr", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.org")
if(!require(pROC)) install.packages("pROC", repos = "http://cran.us.r-project.org")
if(!require(caTools)) install.packages("caTools", repos = "http://cran.us.r-project.org")
if(!require(caretEnsemble)) install.packages("caretEnsemble", repos = "http://cran.us.r-project.org")
if(!require(grid)) install.packages("grid", repos = "http://cran.us.r-project.org")
if(!require(readr)) install.packages("readr", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(ggfortify)) install.packages("ggfortify", repos = "http://cran.us.r-project.org")
if(!require(glmnet)) install.packages("glmnet", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(nnet)) install.packages("nnet", repos = "http://cran.us.r-project.org")
if(!require(funModeling)) install.packages("funModeling", repos = "http://cran.us.r-project.org")
if(!require(RefManageR)) install.packages("RefManageR", repos = "http://cran.us.r-project.org")
if(!require(kableExtra)) install.packages("kableExtra", repos = "http://cran.us.r-project.org")

# The data file will be loaded from my personal computer
#Please add own .csv below if wanting to recreate
data <- read.csv("/Users/user/Documents/DataScience/FinalCapstone/NETCapstone/DataFiles/column_2C_weka.
```

Methods And Analysis

Data Analysis

From observation of the Dataset, one can determine that the Dateset consists of 310 observations and 7 variables

```r
str(data)
```

```
## 'data.frame':    310 obs. of  7 variables:
##  $ pelvic_incidence        : num  63 39.1 68.8 69.3 49.7 ...
##  $ pelvic_tilt.numeric     : num  22.55 10.06 22.22 24.65 9.65 ...
##  $ lumbar_lordosis_angle   : num  39.6 25 50.1 44.3 28.3 ...
##  $ sacral_slope            : num  40.5 29 46.6 44.6 40.1 ...
##  $ pelvic_radius           : num  98.7 114.4 106 101.9 108.2 ...
##  $ degree_spondylolisthesis: num  -0.254 4.564 -3.53 11.212 7.919 ...
##  $ class                   : chr  "Abnormal" "Abnormal" "Abnormal" "Abnormal" ...
```

```r
head(data)
```

```
##   pelvic_incidence pelvic_tilt.numeric lumbar_lordosis_angle sacral_slope
## 1         63.02782           22.552586              39.60912     40.47523
## 2         39.05695           10.060991              25.01538     28.99596
## 3         68.83202           22.218482              50.09219     46.61354
## 4         69.29701           24.652878              44.31124     44.64413
## 5         49.71286            9.652075              28.31741     40.06078
## 6         40.25020           13.921907              25.12495     26.32829
##   pelvic_radius degree_spondylolisthesis    class
## 1      98.67292                -0.254400 Abnormal
## 2     114.40543                 4.564259 Abnormal
## 3     105.98514                -3.530317 Abnormal
## 4     101.86850                11.211523 Abnormal
## 5     108.16872                 7.918501 Abnormal
## 6     130.32787                 2.230652 Abnormal
```

```r
summary(data)
```

```
##  pelvic_incidence pelvic_tilt.numeric lumbar_lordosis_angle  sacral_slope
##  Min.   : 26.15   Min.   :-6.555      Min.   : 14.00        Min.   : 13.37
##  1st Qu.: 46.43   1st Qu.:10.667      1st Qu.: 37.00        1st Qu.: 33.35
##  Median : 58.69   Median :16.358      Median : 49.56        Median : 42.40
##  Mean   : 60.50   Mean   :17.543      Mean   : 51.93        Mean   : 42.95
##  3rd Qu.: 72.88   3rd Qu.:22.120      3rd Qu.: 63.00        3rd Qu.: 52.70
##  Max.   :129.83   Max.   :49.432      Max.   :125.74        Max.   :121.43
##  pelvic_radius    degree_spondylolisthesis    class
##  Min.   : 70.08   Min.   :-11.058          Length:310
##  1st Qu.:110.71   1st Qu.:  1.604          Class :character
##  Median :118.27   Median : 11.768          Mode  :character
##  Mean   :117.92   Mean   : 26.297
##  3rd Qu.:125.47   3rd Qu.: 41.287
##  Max.   :163.07   Max.   :418.543
```

One needs to determine if the Dataset contains any missing values

```
## $pelvic_incidence
## [1] 0
##
```

```
## $pelvic_tilt.numeric
## [1] 0
##
## $lumbar_lordosis_angle
## [1] 0
##
## $sacral_slope
## [1] 0
##
## $pelvic_radius
## [1] 0
##
## $degree_spondylolisthesis
## [1] 0
##
## $class
## [1] 0
```

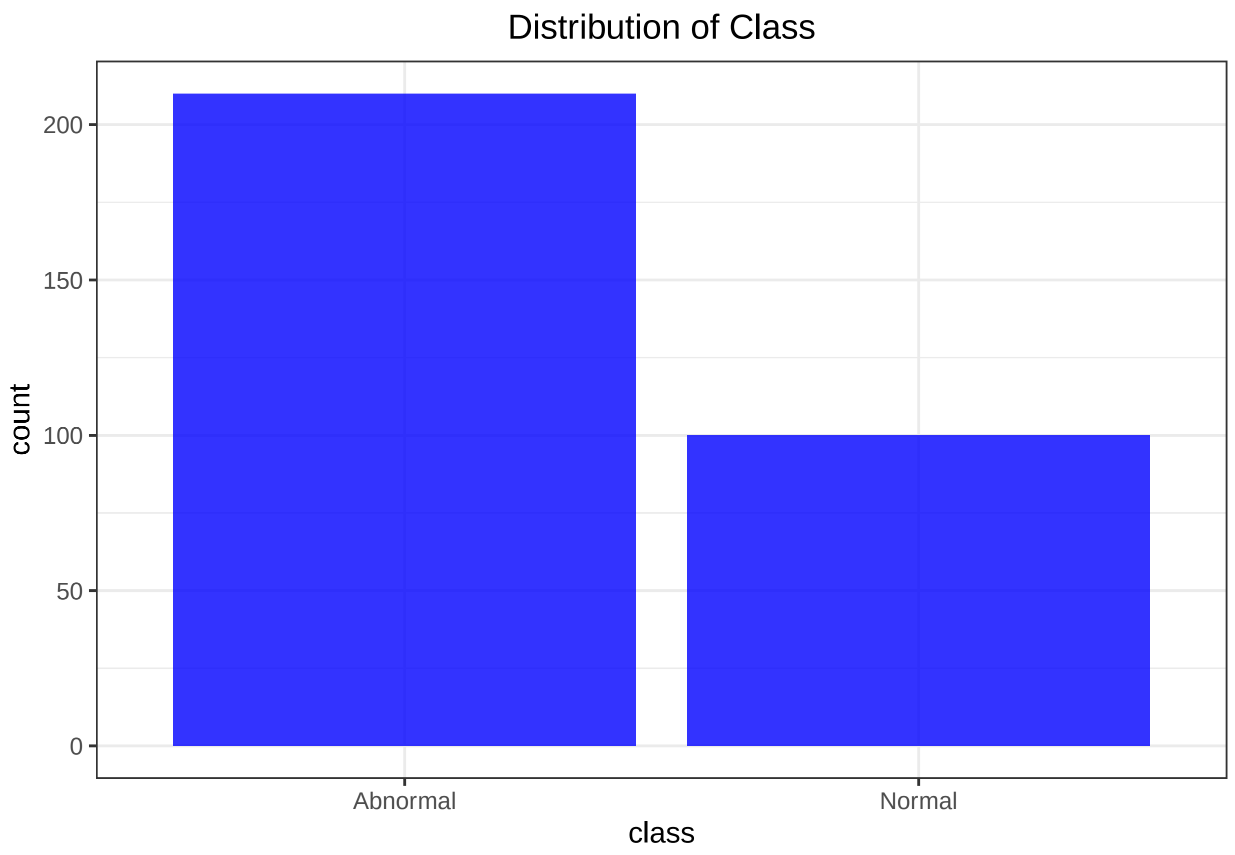
There are no NA values in this dataset, but there is an unbalance between the datasets proportions :

```
#check proportions of income groups
prop.table(table(data$class))
```

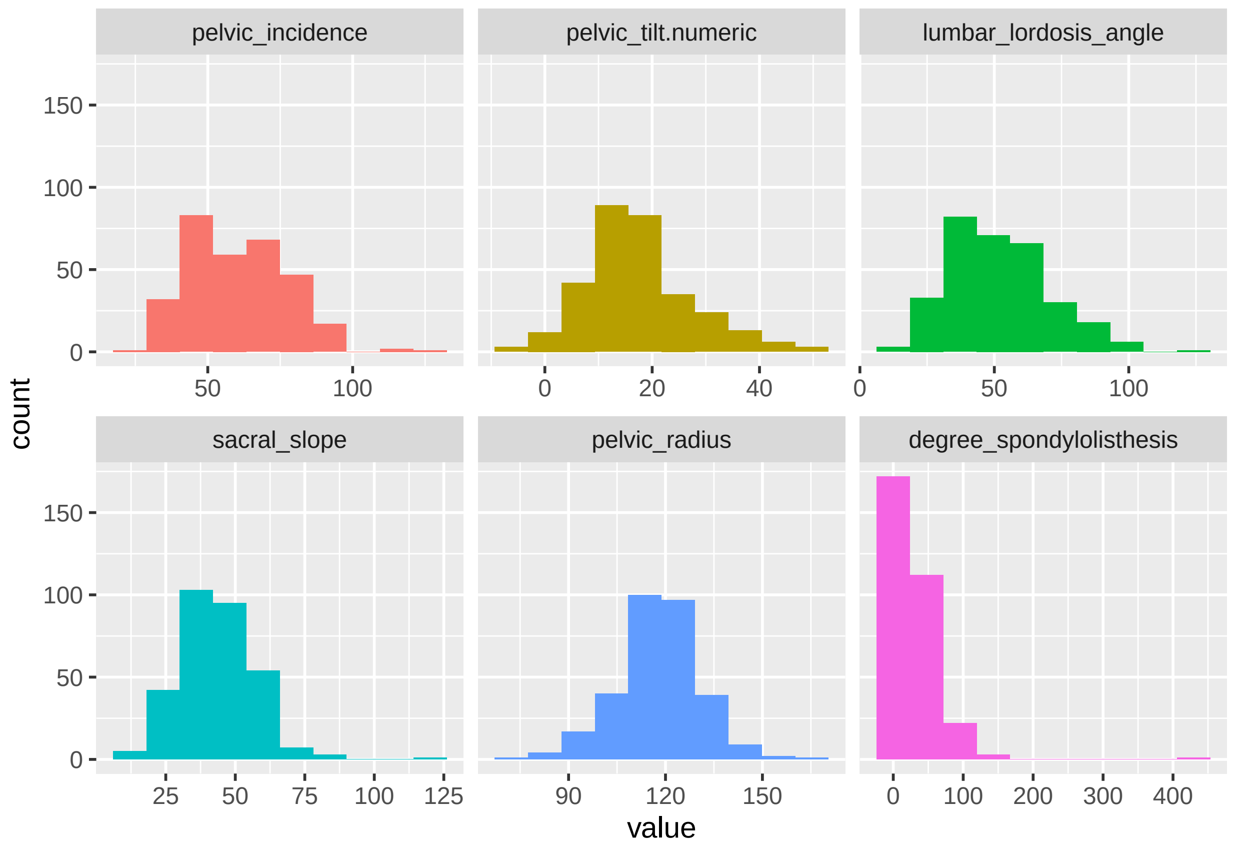
```
##
##  Abnormal    Normal
## 0.6774194 0.3225806
```

The graph below gives and indication to this disproportion in the target variable class:

```
#plot proportion
options(repr.plot.width=4, repr.plot.height=4)

ggplot(data, aes(class))+
  geom_bar(fill="blue",alpha=0.8)+
  theme_bw() +
  labs(title="Distribution of Class") +
  theme(plot.title = element_text(hjust = 0.5))
```



Most variables in the dataset are normally distributed as shown in the below plot, except for degree_spondylolisthesis:



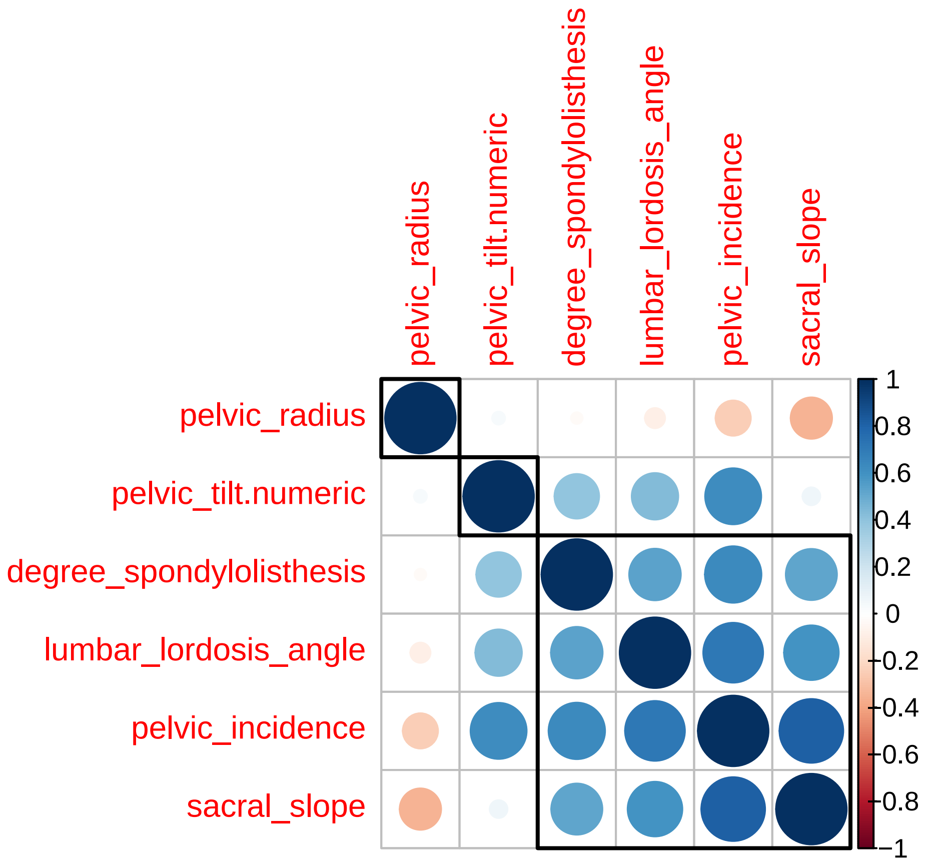
Correlations

Check correlation

```
correlationMatrix <- cor(data[,0:6])
correlationMatrix %>%
  knitr::kable() %>% kable_styling(latex_options = c("striped", "scale_down"))
```

	pelvic_incidence	pelvic_tilt.numeric	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
pelvic_incidence	1.0000000	0.6291988	0.7172824	0.8149600	-0.2474672	0.6387427
pelvic_tilt.numeric	0.6291988	1.0000000	0.4327639	0.0623453	0.0326678	0.3978623
lumbar_lordosis_angle	0.7172824	0.4327639	1.0000000	0.5983869	-0.0803436	0.5336670
sacral_slope	0.8149600	0.0623453	0.5983869	1.0000000	-0.3421283	0.5235575
pelvic_radius	-0.2474672	0.0326678	-0.0803436	-0.3421283	1.0000000	-0.0260650
degree_spondylolisthesis	0.6387427	0.3978623	0.5336670	0.5235575	-0.0260650	1.0000000

```
corrplot(correlationMatrix, order = "hclust", tl.cex = 1, addrect = 3)
```



As seen in plot there seems to be three variables that are highly correlated with each other (cor >= 0.7) (University, n.d.) , these three variables are pelvic_incidence ,lumbar_lordosis_angle and sacral_slope. Due to this we can assume that methods that usually fail due to high correlation variable maybe be impacted on badly by the current variables. Thus the highly correlated variable will be removed as to much correlation can cause some machine learning models to fail.

The Caret R package provides the "findCorrelation", which analyses the correlation matrix of a data's attributes, it then reports on which attributes can be removed.

Following method below proves the assumption of no highly correlated variables:

```
# find Variables that are highly corrected (>0.7)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.7)
# print indexes of highly correlated attributes
print(highlyCorrelated)
```

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

There is one variables to be removed.

Remove highly correlated variables

By carefully selecting features in ones data can mean the difference between poor performance with long training times and great performance with short training times.

The highly correlated variable is removed as shown in the code below:

```r
data2 <- data %>% select(!highlyCorrelated)
# number of columns after removing correlated variables
ncol(data2)
```

```
## [1] 6
```

One variable is lost, this being pelvic_incidence. By removing this one variable the data is cleaned from highly correlated variables. Hense forth the dataset with the highly correlated variable removed will be refered to as the Cleaned Dataset

MODELING Approach

Principal Component Analysis [PCA]

For this project the function "prncomp" will be used to calculate the PCA, this function is chosen as to avoid relecancy and redundancy. The "prncomp" function helps to select components that will avoid correlated variables. This is importance as mentioned before highly correlated variable can cause problems with clustering analysis.

PCA is usually used with data that contains a large number of variables. Although this particular dataset has at max 7 variables the PCA technique will be used on the dataset as to see how it performs. PCA works by reduces the dimensions of the feature space by feature extraction.

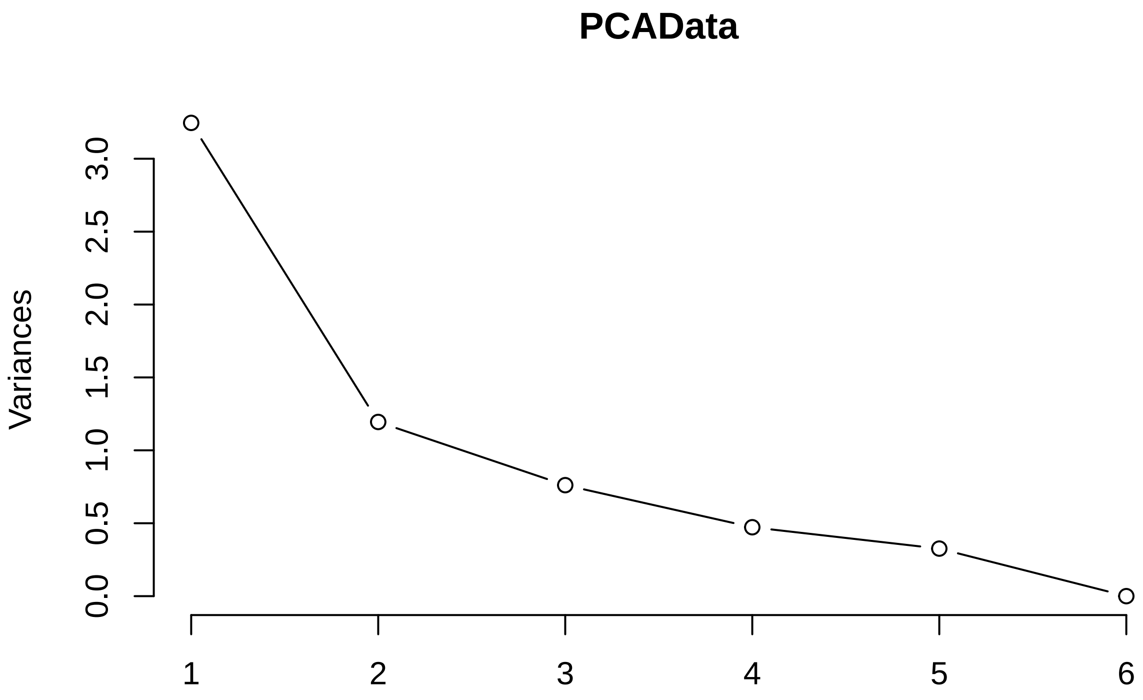
PCA is used to reduce the dimensionality of a dataset which consists of a large number of variables correlated with each other. The variables can be either heavily or lightly correlated. The reduction of dimensionality must be done while retaining the variation present in the dataset, up to the maximum extent.

The same is done by transforming the variables to a new set of variables, which are known as the principal components [PCs]. Pcs are orthogonal and are ordered in such a way that the retention of variation present in the original variables decreases when moving down the order. By transforming variables in this mannger, the 1st principal component retains maximum variation that was present in the original components. The principal components are the eigenvectors of a covariance matrix, and are therefore are orthogonal.

It is important to note that the dataset on which PCA technique is used on, must be scaled. The results are also sensitive to the relative scaling. (ProjectPro n.d.)

PCA on original data

```r
PCAData <- prcomp(data[,0:6], center = TRUE, scale = TRUE)
plot(PCAData, type="l")
```



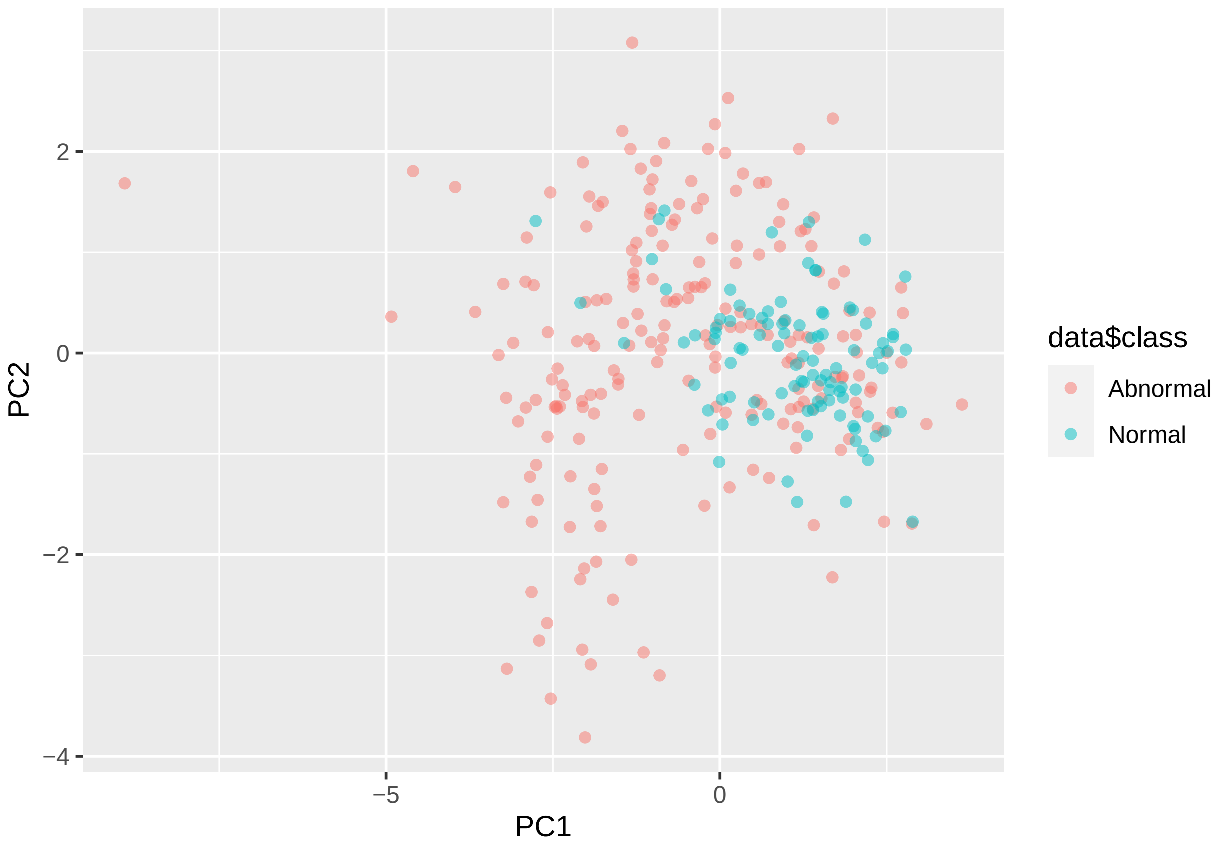
```
#summary
summary(PCAData)
```

```
## Importance of components:
##                          PC1    PC2    PC3     PC4     PC5       PC6
## Standard deviation     1.802 1.0930 0.8724 0.68741 0.57098 1.935e-10
## Proportion of Variance 0.541 0.1991 0.1268 0.07875 0.05434 0.000e+00
## Cumulative Proportion  0.541 0.7401 0.8669 0.94566 1.00000 1.000e+00
```

As seen in the table above the first component can explain 0.541 of the variance after applying 4 PCs, 0.94566 of the variance can be explained. According to the summary above, 1.0 of the variance can be explained after 5 PCS, only a small number of PCs are required as the dataset has so few variables

Plot of PC1 vs PC2

```
pcaDf <- as.data.frame(PCAData$x)
ggplot(pcaDf, aes(x=PC1, y=PC2, col=data$class)) + geom_point(alpha=0.5)
```

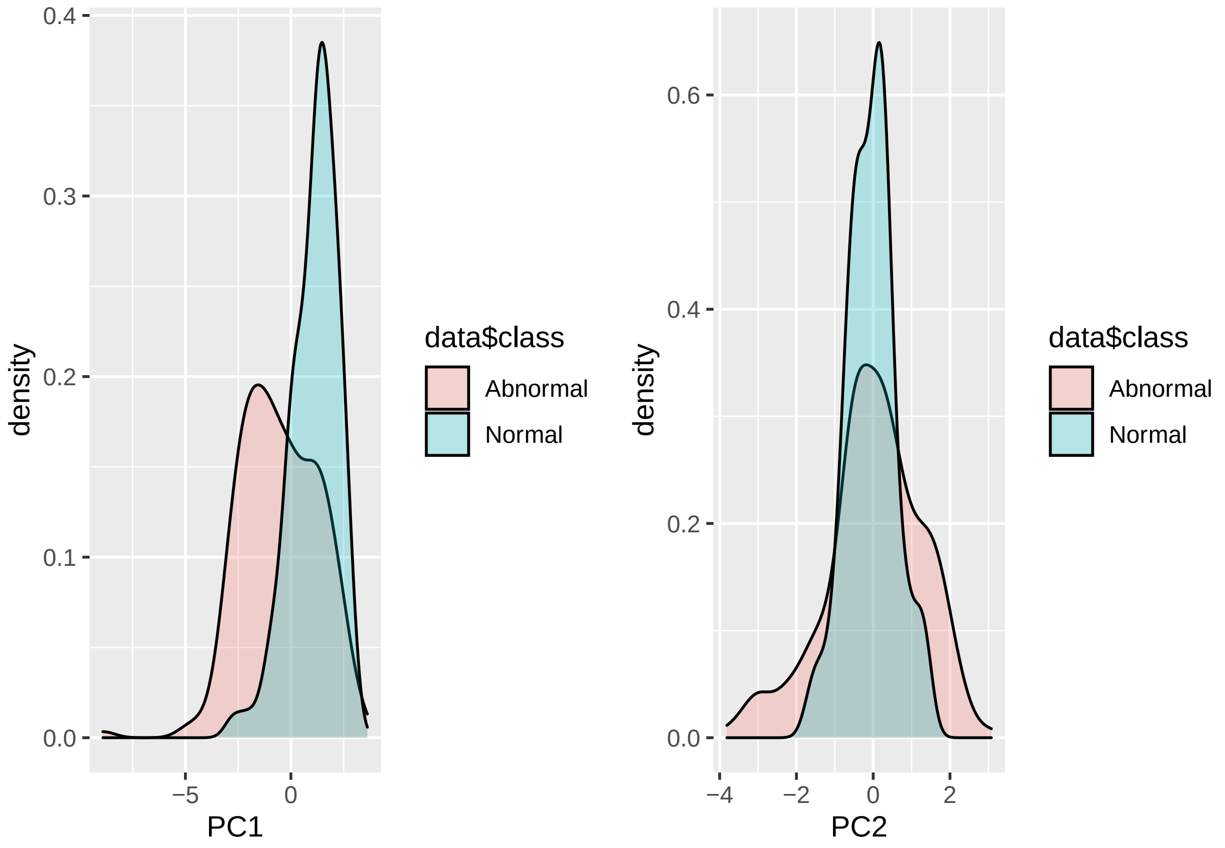


From the plot above it can be determined that the first two components somewhat separated into two classes. This is caused by the fact that the variance explained by these components is not large.

Plot of densitys

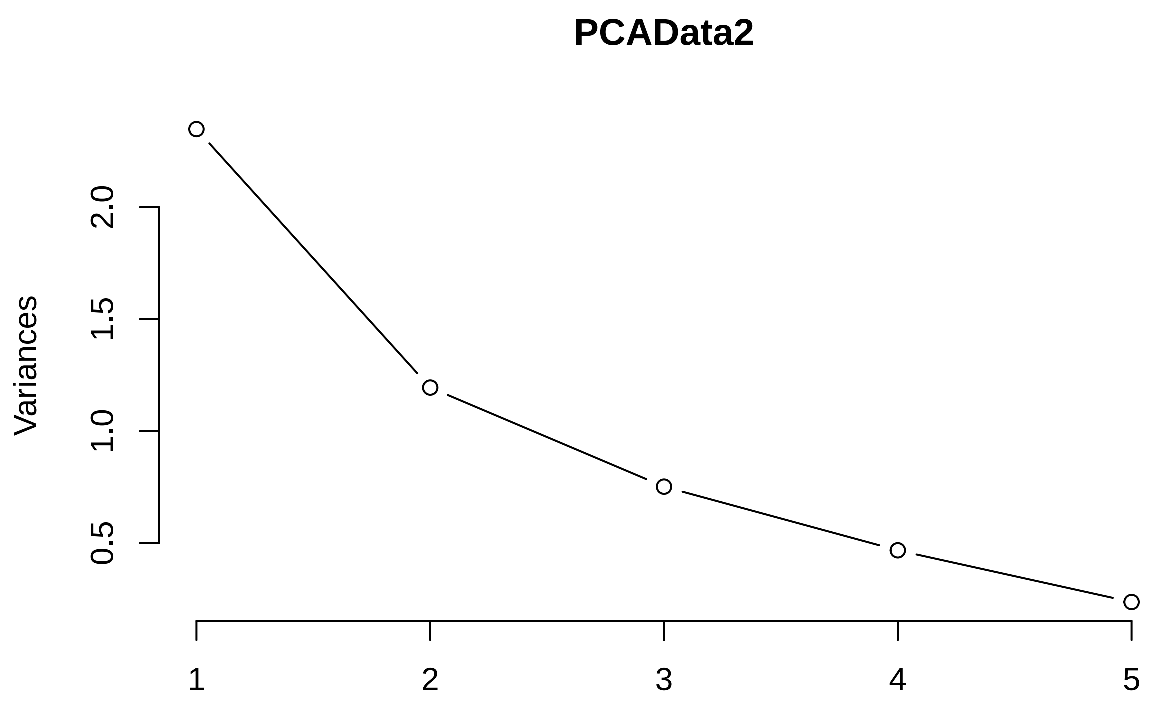
```
pc1 <- ggplot(pcaDf, aes(x=PC1, fill=data$class)) + geom_density(alpha=0.25)
pc2 <- ggplot(pcaDf, aes(x=PC2, fill=data$class)) + geom_density(alpha=0.25)
```

```
grid.arrange(pc1, pc2, ncol=2)
```



PCA Cleaned Dataset

```
PCAData2 <- prcomp(data2[,0:5], center = TRUE, scale = TRUE)
plot(PCAData2, type="l")
```



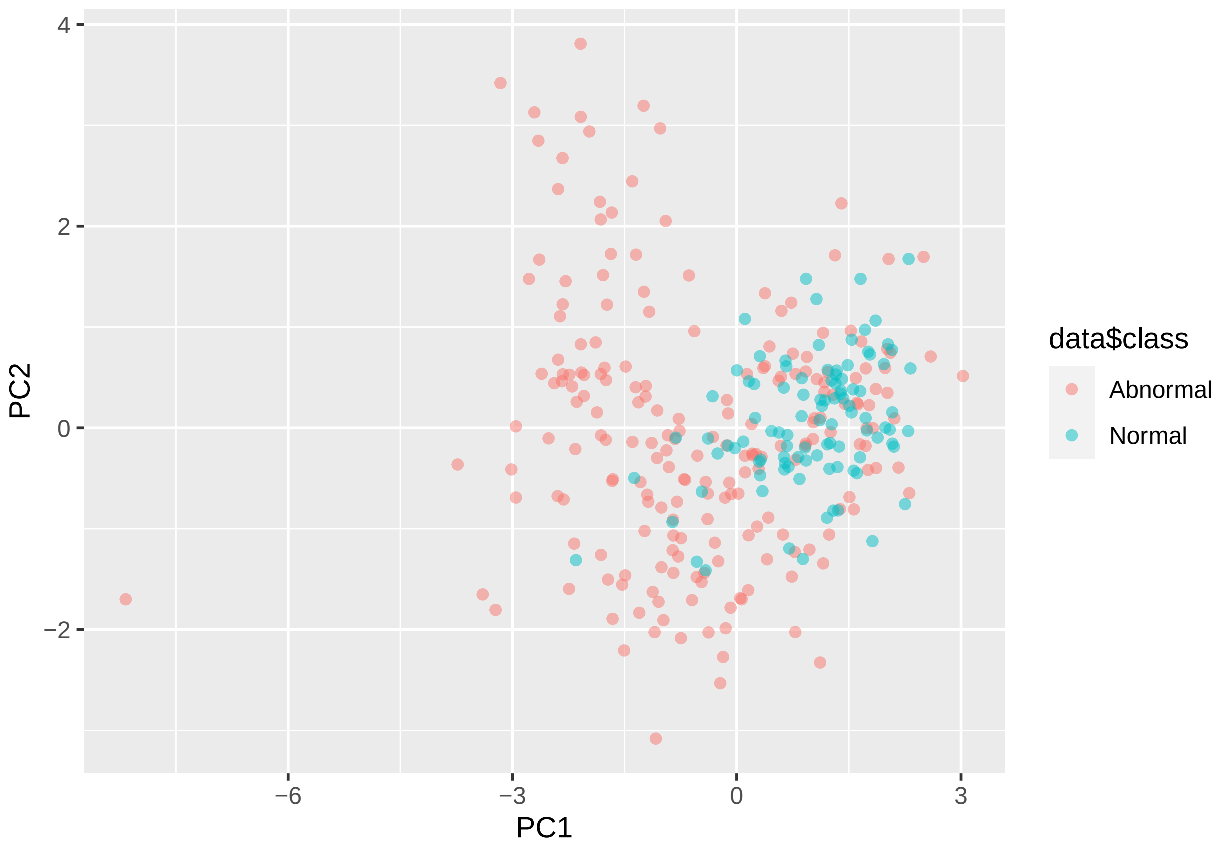
```
summary(PCAData2)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5
## Standard deviation     1.5325 1.0930 0.8672 0.68395 0.48679
## Proportion of Variance 0.4697 0.2389 0.1504 0.09356 0.04739
## Cumulative Proportion  0.4697 0.7086 0.8590 0.95261 1.00000
```

The above table shows that 0.95261 of the variance is explained with 4 PCs in the transformed dataset data2. There does not seem to be must change with having removed the highly correlated variable but the is some change in the amount of variance explained by the 4th PCs. Thus there seems there is some improvement.

Plot of PC1 vs PC2

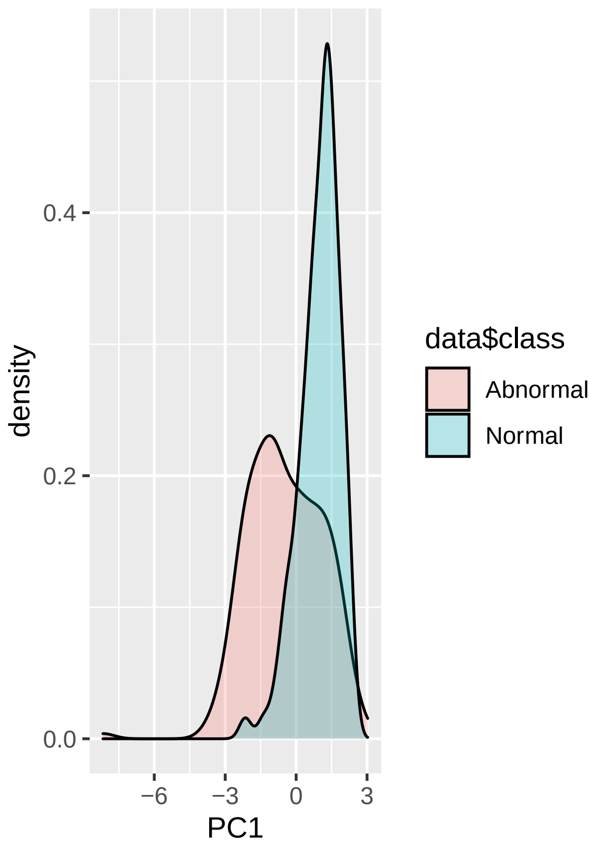
```
PcaDf2 <- as.data.frame(PCAData2$x)
ggplot(PcaDf2, aes(x=PC1, y=PC2, col=data$class)) + geom_point(alpha=0.5)
```

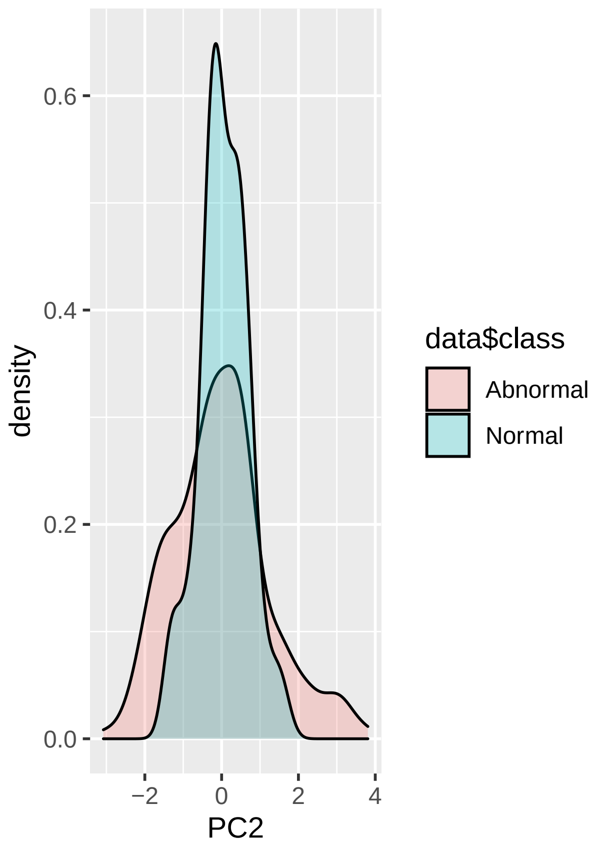


From the plot above it can be determined that the first two components are still somewhat separated into two classes. This is caused by the fact that the variance explained by these components is not large. There is some change in the density plots, the density plots have narrowed for the components and thus this can be see as some improvement.

Plot of densitys

```
pc12 <- ggplot(PcaDf2, aes(x=PC1, fill=data$class)) + geom_density(alpha=0.25)
pc22 <- ggplot(PcaDf2, aes(x=PC2, fill=data$class)) + geom_density(alpha=0.25)
grid.arrange(pc12, pc22, ncol=2)
```





Linear Discriminant Analysis [LDA]

Other than PCA, there is LDA. LDA takes in consideration the different classes and could possibly get better results.

The particularity of LDA is that it models the distribution of predictors separately in each of the response classes, and then it uses Bayes' theorem to estimate the probability. It is important to note that LDA assumes a normal distribution for each class, a class-specific mean, and a common variance. (Peixeiro, n.d.)

LDA with original data

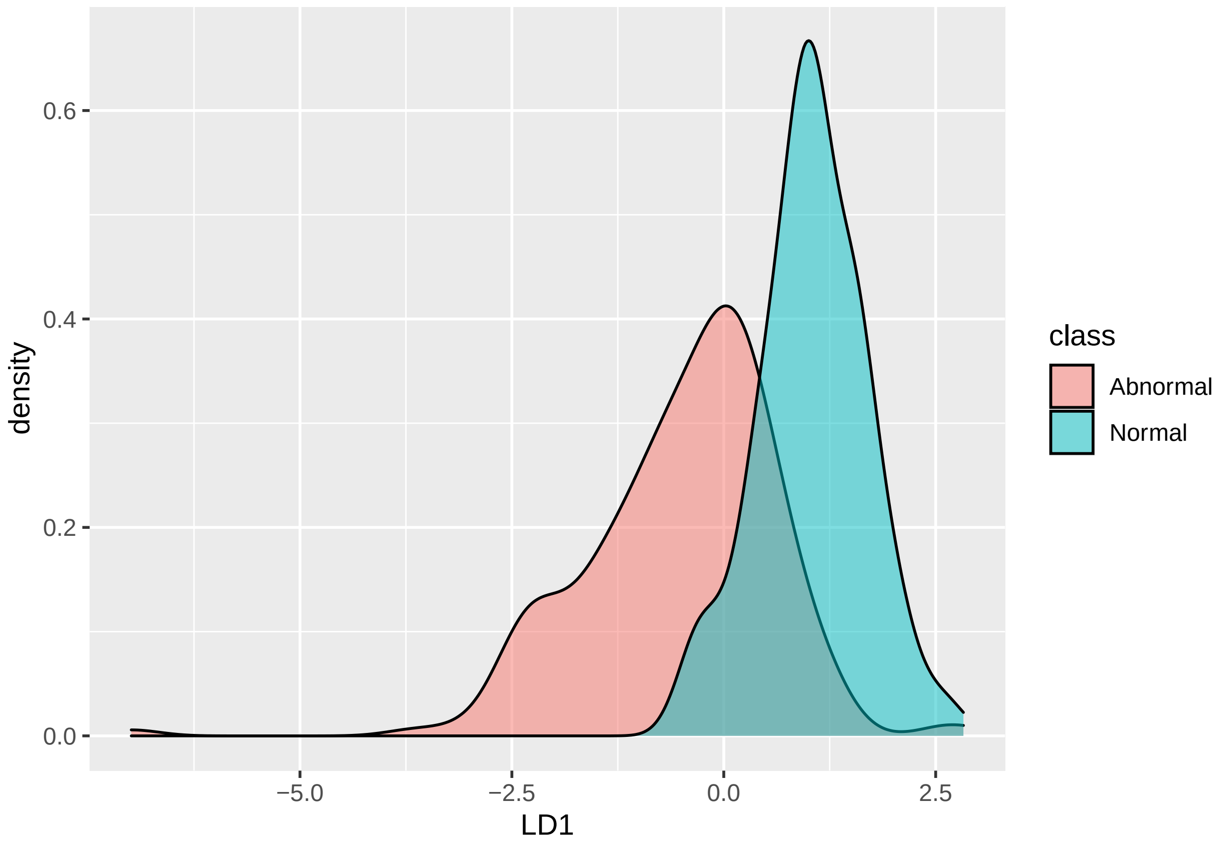
```
LdaData <- MASS::lda(class~., data = data, center = TRUE, scale = TRUE)
LdaData
```

```
## Call:
## lda(class ~ ., data = data, center = TRUE, scale = TRUE)
##
## Prior probabilities of groups:
##  Abnormal    Normal
## 0.6774194 0.3225806
##
## Group means:
##          pelvic_incidence pelvic_tilt.numeric lumbar_lordosis_angle
## Abnormal         64.69256            19.79111              55.92537
## Normal           51.68524            12.82141              43.54260
##          sacral_slope pelvic_radius degree_spondylolisthesis
## Abnormal     44.90145      115.0777                37.777705
## Normal       38.86383      123.8908                 2.186572
##
## Coefficients of linear discriminants:
##                                   LD1
## pelvic_incidence          0.007884392
## pelvic_tilt.numeric      -0.032981545
## lumbar_lordosis_angle    -0.015839324
## sacral_slope              0.029051984
## pelvic_radius             0.058779826
## degree_spondylolisthesis -0.024406328
```

```
#Data frame of the LDA for visualization purposes
ldaDataPredict <- predict(LdaData, data)$x %>% as.data.frame() %>% cbind(class=data$class)
```

Plot density of LD1

```
ggplot(ldaDataPredict, aes(x=LD1, fill=class)) + geom_density(alpha=0.5)
```



LDA on Cleaned Dataset

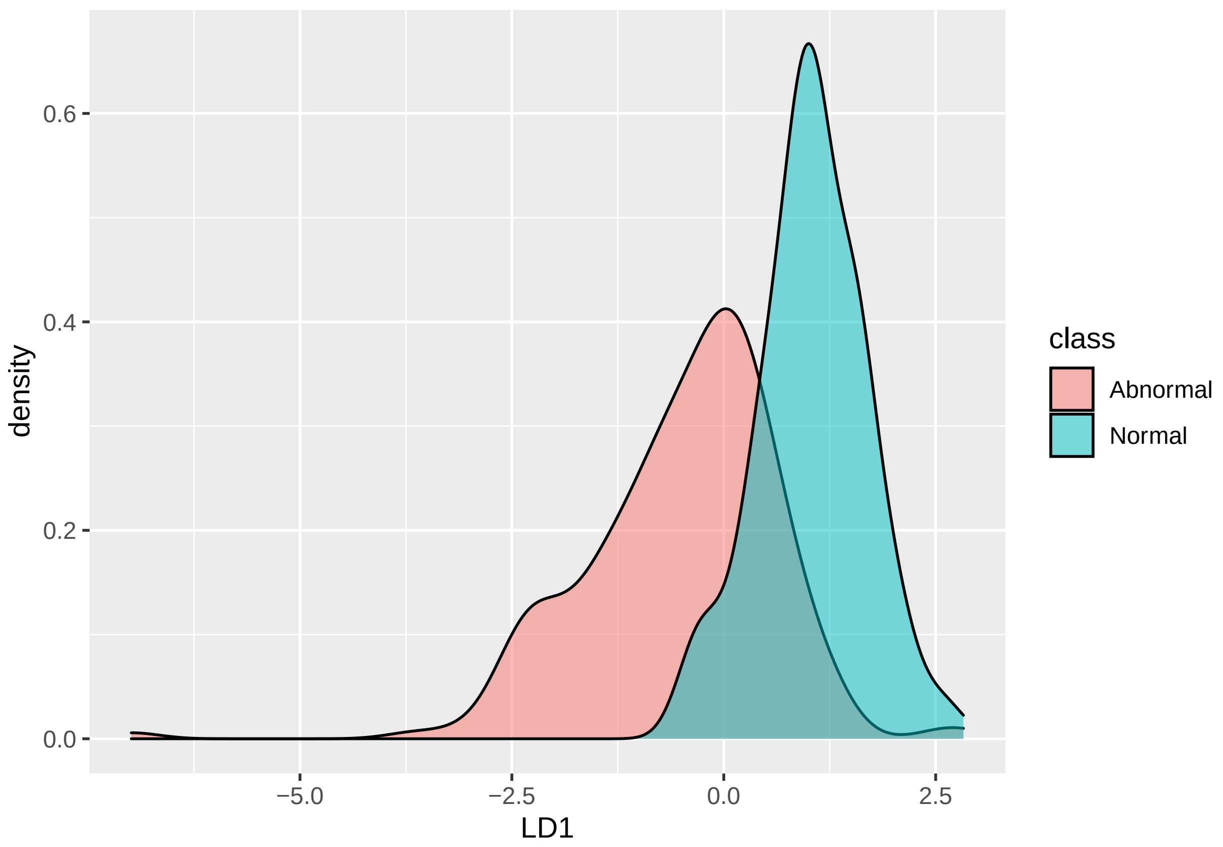
```
LdaData2 <- MASS::lda(class~., data = data2, center = TRUE, scale = TRUE)
LdaData2
```

```
## Call:
## lda(class ~ ., data = data2, center = TRUE, scale = TRUE)
##
## Prior probabilities of groups:
##  Abnormal    Normal
## 0.6774194 0.3225806
##
## Group means:
##          pelvic_tilt.numeric lumbar_lordosis_angle sacral_slope pelvic_radius
## Abnormal            19.79111              55.92537     44.90145      115.0777
## Normal              12.82141              43.54260     38.86383      123.8908
##          degree_spondylolisthesis
## Abnormal                37.777705
## Normal                   2.186572
##
## Coefficients of linear discriminants:
##                                  LD1
## pelvic_tilt.numeric      -0.02509715
## lumbar_lordosis_angle    -0.01583932
## sacral_slope              0.03693638
## pelvic_radius             0.05877983
## degree_spondylolisthesis -0.02440633
```

```
#Data frame of the LDA for visualization purposes
ldaDataPredict2 <- predict(LdaData2, data2)$x %>% as.data.frame() %>% cbind(class=data2$class)
```

Plot density of LD1

```
ggplot(ldaDataPredict2, aes(x=LD1, fill=class)) + geom_density(alpha=0.5)
```



There seems to be very little difference in the LDA outputs and plot when using the original and cleaned Datasets. The only changes that can be noted is the changes in the Coefficients of linear discriminates.

There is a clearer difference between the Normal and Abnormal classes in the density plots when using LDA

Methods

Original dataset

Creating train and training/validation datasets

```
set.seed(1, sample.kind="Rounding") #if using R 3.5 or earlier, use `set.seed(1)
#Train (80% of data)
datSamplingIndex <- createDataPartition(data$class, times=1, p=0.8, list = FALSE)
trainData <- data[datSamplingIndex, ]
#Test (20%)
testData <- data[-datSamplingIndex, ]
```

Control the data

```
fitControl <- trainControl(method="cv",
                           number = 20,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary)
```

Cleaned Dataset

Creating train and training/validation datasets

```
#Train (80% of data)
data3 <- cbind (class=data$class, data2)
trainData2 <- data3[datSamplingIndex, ]
#Test (20%)
testData2 <- data3[-datSamplingIndex, ]
```

Control the data

```
fitControl2 <- trainControl(method="cv",
                           number = 20,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary)
```

Naive Bayes Model

The Naive Bayesian classifier is based on Bayes' theorem with the independence assumptions between predictors. A Naive Bayesian model is simple to build, as there are no complicated iterative parameter estimation. Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c). Naive Bayes classifier assumes that the effect of a predictor (x) value on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.

Although the Naive Bayesian classifier is simplistic it often works well. Thus it is widely use as it often outperforms more sophisticated classification methods. ("Naive Bayesian," n.d.)

Original Dataset

```
NaiveBayesModel <- train(class~.,
                      trainData,
                      method="nb",
                      metric="ROC",
                      preProcess=c('center', 'scale'), #in order to normalize the data
                      trace=FALSE,
                      importance = TRUE,
                      trControl=fitControl)

NBPrediction <- predict(NaiveBayesModel, testData)
NBConfMatrix <- confusionMatrix(NBPrediction, as.factor(testData$class), positive = "Abnormal")
NBConfMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       25      2
##   Normal         17     18
##
##                Accuracy : 0.6935
##                  95% CI : (0.5635, 0.8044)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.452354
##
##                   Kappa : 0.4139
##
##  Mcnemar's Test P-Value : 0.001319
##
##             Sensitivity : 0.5952
##             Specificity : 0.9000
##          Pos Pred Value : 0.9259
##          Neg Pred Value : 0.5143
##              Prevalence : 0.6774
##          Detection Rate : 0.4032
##    Detection Prevalence : 0.4355
##       Balanced Accuracy : 0.7476
##
##        'Positive' Class : Abnormal
##
```

Cleaned Dataset

```
NaiveBayesModel2 <- train(class~.,
                      trainData2,
                      method="nb",
                      metric="ROC",
                      preProcess=c('center', 'scale'), #in order to normalize the data
                      trace=FALSE,
                      importance = TRUE,
                      trControl=fitControl2)
```

```r
NBPrediction2 <- predict(NaiveBayesModel2, testData2)
NBConfMatrix2 <- confusionMatrix(NBPrediction2, as.factor(testData2$class), positive = "Abnormal")
NBConfMatrix2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       27      1
##   Normal         15     19
##
##                Accuracy : 0.7419
##                  95% CI : (0.615, 0.8447)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.171207
##
##                   Kappa : 0.501
##
##  Mcnemar's Test P-Value : 0.001154
##
##             Sensitivity : 0.6429
##             Specificity : 0.9500
##          Pos Pred Value : 0.9643
##          Neg Pred Value : 0.5588
##              Prevalence : 0.6774
##          Detection Rate : 0.4355
##    Detection Prevalence : 0.4516
##       Balanced Accuracy : 0.7964
##
##        'Positive' Class : Abnormal
##
```

There is a positive effect on the Naive Bayes Model from the removal of the highly correlated variable.

The accuracy of the Cleaned Dataset model can be noted being 0.7419 where as the Original Dataset model had and accuracy of 0.6935. It seems that the Naive Bayes Model is performing fairly well

In a later discussion other metrics will be discussed, such as:

- Sensitivity (recall) represent the true positive rate: the proportions of actual positives correctly identified.
- Specificity is the true negative rate: the proportion of actual negatives correctly identified.
- Accuracy is the general score of the classifier model performance as it is the ratio of how many samples are correctly classified to all samples.
- F1 score: the harmonic mean of precision and sensitivity.
- Accuracy and F1 score would be used to compare the result with the benchmark model.
- Precision: the number of correct positive results divided by the number of all positive results returned by the classifier.

Random Forest [RF]

The RF is one of the most powerful machine learning algorithms available. RF is a supervised machine learning algorithm that can be used for both classification and regression tasks. The algorithm addresses the shortcomings of decision trees by using a clever tick. Its goal is to improve prediction performance and reduce instability by averaging multiple decision trees. RF is made from a group of individual decision trees, this technique is called Ensemble Learning. A large group of uncorrelated decision trees can produce more accurate and stable results than any of individual decision trees.

Training a RF for a classification task, is actually training a group of decision trees. Then by obtaining all the predictions of each individual trees and can use these predictions to predict the class that gets the most votes. Although some individual trees produce wrong predictions, many can produce accurate predictions. As a group, they can move towards accurate predictions. (Pramoditha, n.d.)

Original Dataset

```
RandomforestModel <- train(class~.,
                           trainData,
                           method="rf",
                           metric="ROC",
                           #tuneLength=10,
                           #tuneGrid = expand.grid(mtry = c(2, 3, 6)),
                           preProcess = c('center', 'scale'),
                           trControl=fitControl)

RFPrediction <- predict(RandomforestModel, testData)
#Check results
RFConfMatrix <- confusionMatrix(RFPrediction, as.factor(testData$class), positive = "Abnormal")
RFConfMatrix
```

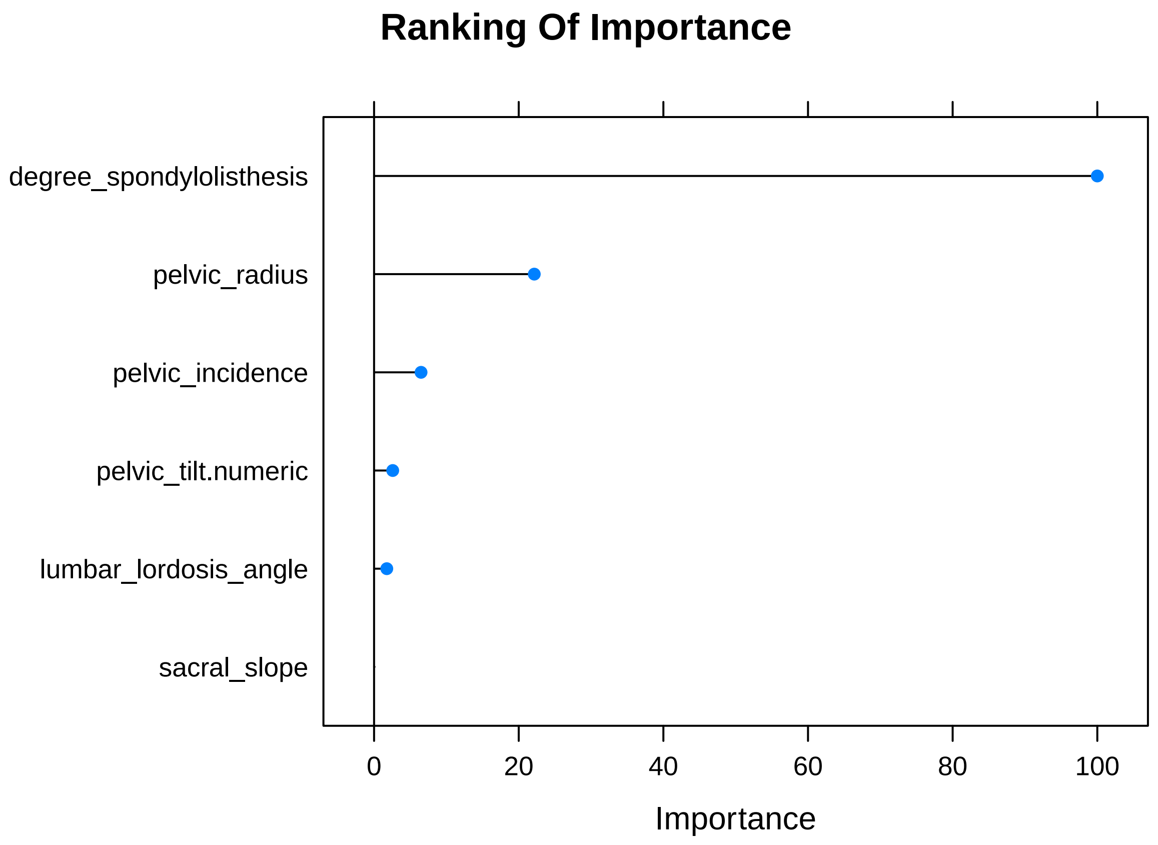
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       35      5
##   Normal          7     15
##
##                Accuracy : 0.8065
##                  95% CI : (0.6863, 0.8958)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.01765
##
##                   Kappa : 0.5684
##
##  Mcnemar's Test P-Value : 0.77283
##
##             Sensitivity : 0.8333
##             Specificity : 0.7500
##          Pos Pred Value : 0.8750
##          Neg Pred Value : 0.6818
##              Prevalence : 0.6774
##          Detection Rate : 0.5645
##    Detection Prevalence : 0.6452
##       Balanced Accuracy : 0.7917
##
##        'Positive' Class : Abnormal
##
```

```r
varImp(RandomforestModel)
```

Plot of Variable Importance

```
## rf variable importance
##
##                          Overall
## degree_spondylolisthesis 100.000
## pelvic_radius             22.152
## pelvic_incidence           6.497
## pelvic_tilt.numeric        2.580
## lumbar_lordosis_angle      1.760
## sacral_slope               0.000
```

```r
plot(varImp(RandomforestModel), main="Ranking Of Importance")
```



Cleaned Dataset

```r
RandomforestModel2 <- train(class~.,
                            trainData2,
                            method="rf",
                            metric="ROC",
                            preProcess = c('center', 'scale'),
                            trControl=fitControl2)

RFPrediction2 <- predict(RandomforestModel2, testData2)
```

```r
#Check results
RFConfMatrix2 <- confusionMatrix(RFPrediction2, as.factor(testData2$class), positive = "Abnormal")
RFConfMatrix2
```

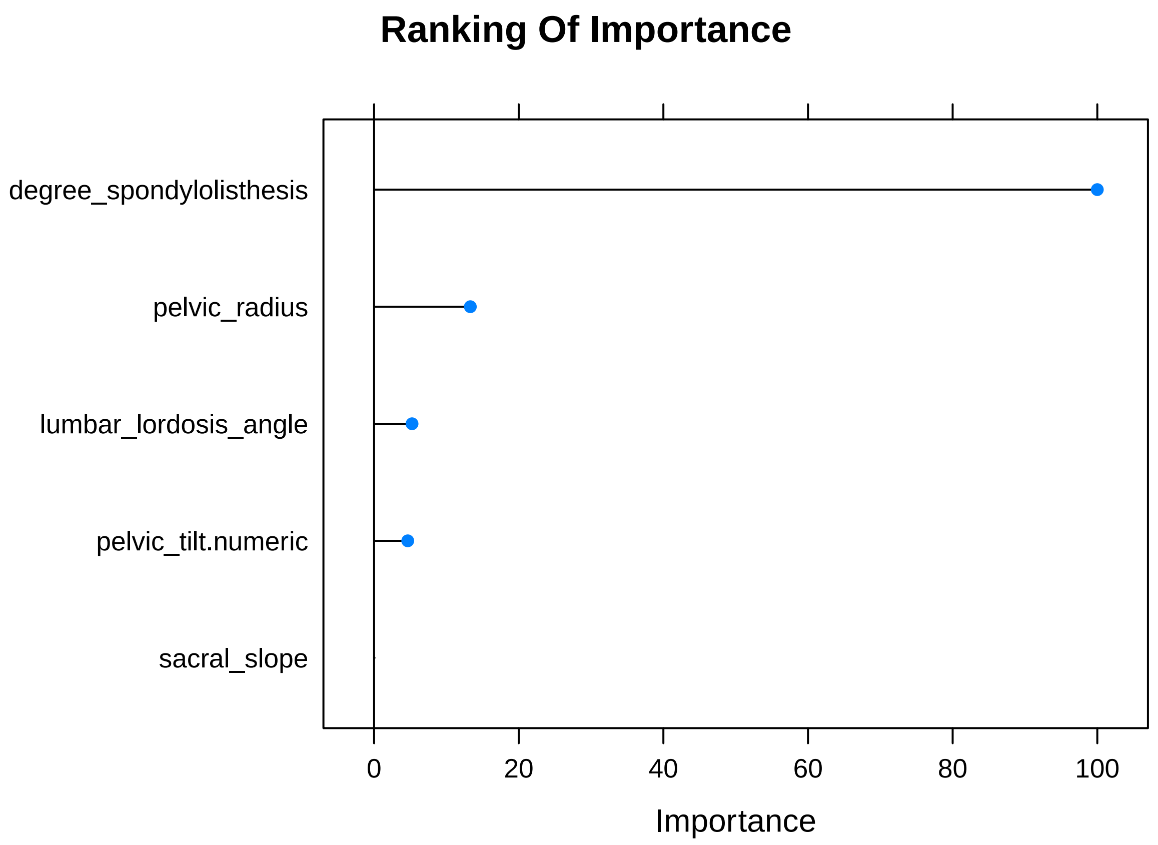
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       35      6
##   Normal          7     14
##
##                Accuracy : 0.7903
##                  95% CI : (0.6682, 0.8834)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.03518
##
##                   Kappa : 0.5264
##
##  Mcnemar's Test P-Value : 1.00000
##
##             Sensitivity : 0.8333
##             Specificity : 0.7000
##          Pos Pred Value : 0.8537
##          Neg Pred Value : 0.6667
##              Prevalence : 0.6774
##          Detection Rate : 0.5645
##    Detection Prevalence : 0.6613
##       Balanced Accuracy : 0.7667
##
##        'Positive' Class : Abnormal
##
```

```r
varImp(RandomforestModel2)
```

Plot of Variable Importance

```
## rf variable importance
##
##                          Overall
## degree_spondylolisthesis 100.000
## pelvic_radius             13.310
## lumbar_lordosis_angle      5.252
## pelvic_tilt.numeric        4.658
## sacral_slope               0.000
```

```r
plot(varImp(RandomforestModel2), main="Ranking Of Importance")
```



The removal of the highly correlated variable from the original dataset had negitive impact on the Random Forest Model. This can be seen from the decrease in the Accuracy and Specificity.

Logistic Regression Model [LogReg]

LogReg is mainly used for binary classification. A binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features).

Original Dataset

```
LogRegModel<- train(class~., data = trainData,
                    method = "glm",
                    metric = "ROC",
                    preProcess = c("scale", "center"),  # in order to normalize the data
                    trControl= fitControl)

LogRegPred <- predict(LogRegModel, testData)
# Check results
LogRegConfMatrix <- confusionMatrix(LogRegPred, as.factor(testData$class), positive = "Abnormal")
LogRegConfMatrix
```

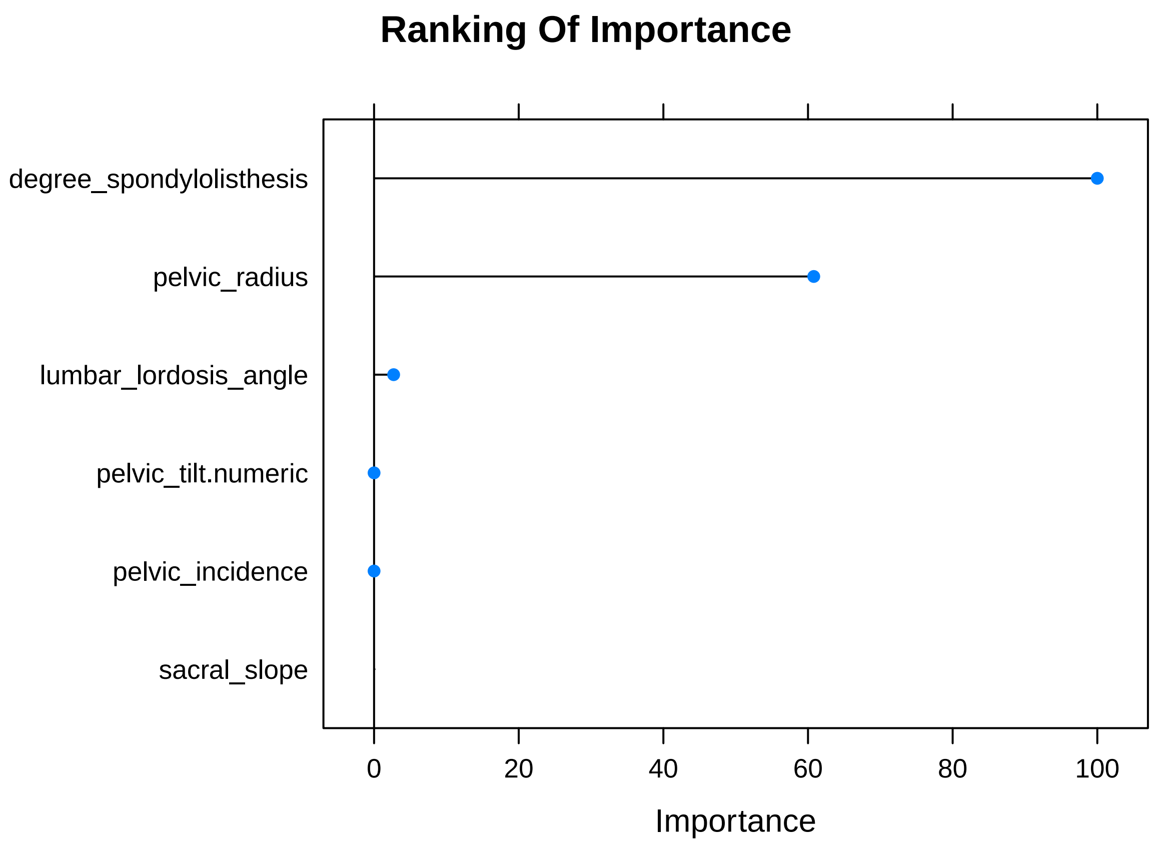
```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction Abnormal Normal
##   Abnormal       36      5
##   Normal          6     15
##
##                Accuracy : 0.8226
##                  95% CI : (0.7047, 0.908)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.008074
##
##                   Kappa : 0.5993
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8571
##             Specificity : 0.7500
##          Pos Pred Value : 0.8780
##          Neg Pred Value : 0.7143
##              Prevalence : 0.6774
##          Detection Rate : 0.5806
##    Detection Prevalence : 0.6613
##       Balanced Accuracy : 0.8036
##
##        'Positive' Class : Abnormal
##
```

```r
varImp(LogRegModel)
```

Plot of Variable Importance

```
## glm variable importance
##
##                            Overall
## degree_spondylolisthesis 1.000e+02
## pelvic_radius            6.079e+01
## lumbar_lordosis_angle    2.707e+00
## pelvic_tilt.numeric      6.557e-08
## pelvic_incidence         3.134e-08
## sacral_slope             0.000e+00
```

```r
plot(varImp(LogRegModel), main="Ranking Of Importance")
```



Cleaned Dataset

```
LogRegModel2<- train(class~., data = trainData2,
                     method = "glm",
                     metric = "ROC",
                     preProcess = c("scale", "center"),  # in order to normalize the data
                     trControl= fitControl2)

LogRegPred2 <- predict(LogRegModel2, testData2)
# Check results
LogRegConfMatrix2 <- confusionMatrix(LogRegPred2, as.factor(testData2$class), positive = "Abnormal")
LogRegConfMatrix2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       36      5
##   Normal          6     15
##
##                Accuracy : 0.8226
##                  95% CI : (0.7047, 0.908)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.008074
##
##                   Kappa : 0.5993
```

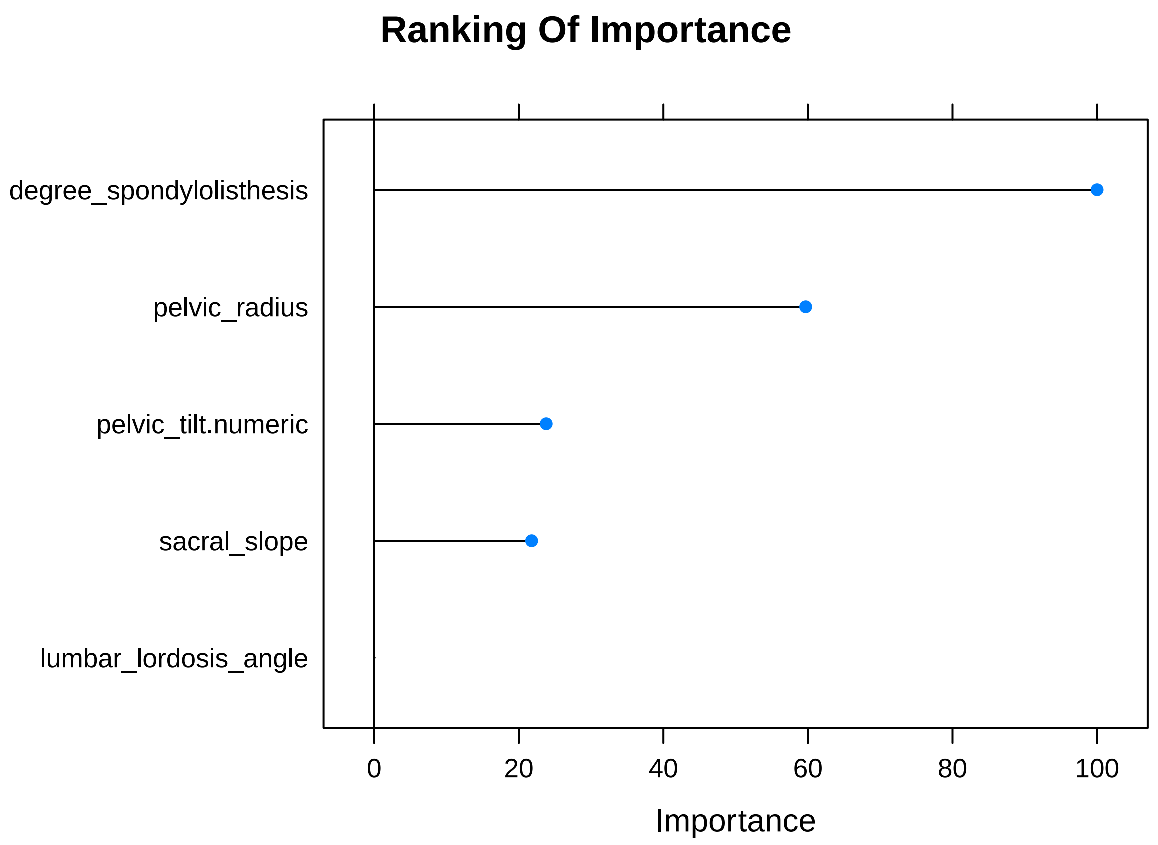
```
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8571
##             Specificity : 0.7500
##          Pos Pred Value : 0.8780
##          Neg Pred Value : 0.7143
##              Prevalence : 0.6774
##          Detection Rate : 0.5806
##    Detection Prevalence : 0.6613
##       Balanced Accuracy : 0.8036
##
##        'Positive' Class : Abnormal
##
```

```r
varImp(LogRegModel2)
```

Plot of Variable Importance

```
## glm variable importance
##
##                         Overall
## degree_spondylolisthesis  100.00
## pelvic_radius             59.69
## pelvic_tilt.numeric       23.80
## sacral_slope              21.78
## lumbar_lordosis_angle      0.00
```

```r
plot(varImp(LogRegModel2), main="Ranking Of Importance")
```



There is no impact on the LogReg model with the removal of the highly correlated variable.

K Nearest Neighbor (KNN) Model

KNN is a supervised learning algorithms that is commonly used in data mining and machine learning. It is a classifier algorithm, learning is based "how similar" is a data from one another. KNN is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure. The KNN algorithm assumes that similar things exist in close proximity (Harrison, n.d.).

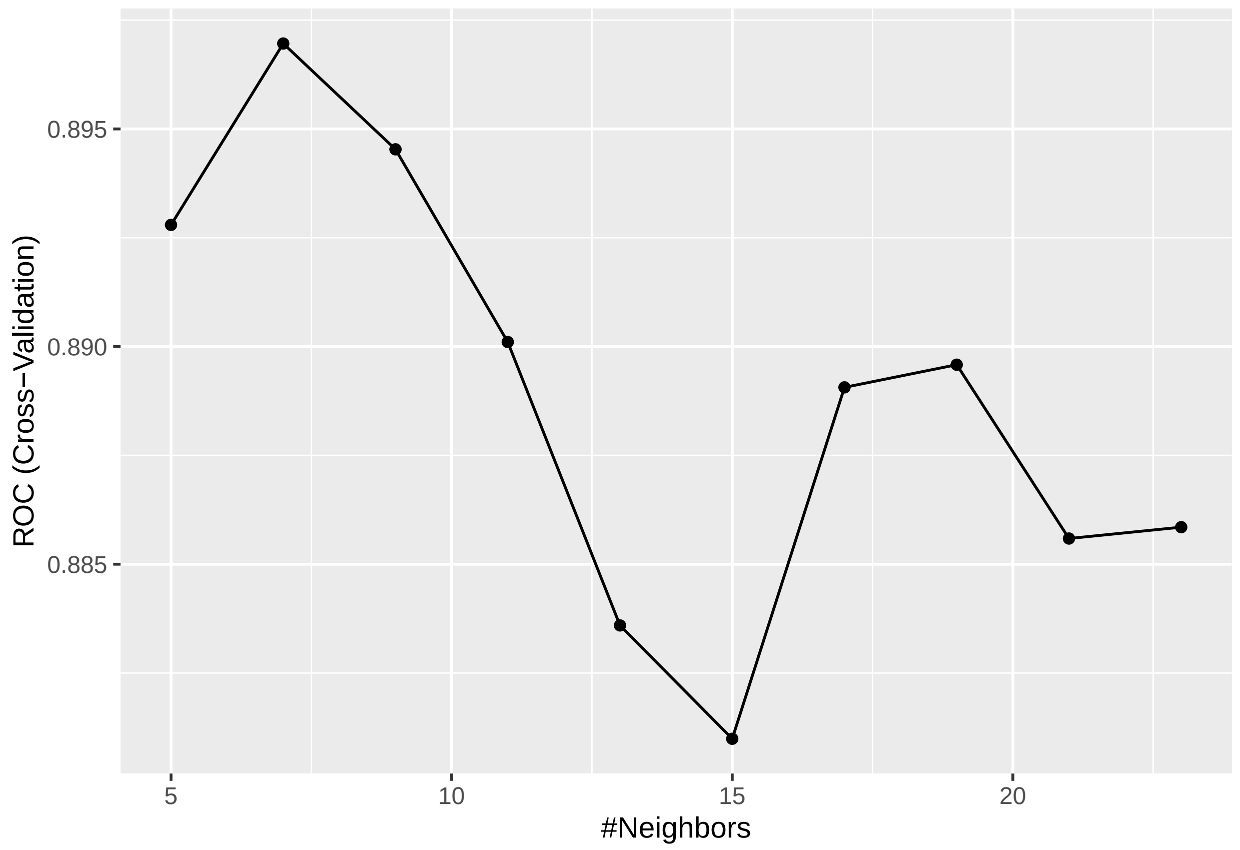
Original Dataset

```
KNNModel <- train(class~.,
                  trainData,
                  method="knn",
                  metric="ROC",
                  preProcess = c('center', 'scale'),
                  tuneLength=10,
                  trControl=fitControl)

KNNPred <- predict(KNNModel, testData)
KNNConfMatrix <- confusionMatrix(KNNPred, as.factor(testData$class), positive = "Abnormal")
KNNConfMatrix
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction Abnormal Normal
##   Abnormal       32      6
##   Normal         10     14
##
##                Accuracy : 0.7419
##                  95% CI : (0.615, 0.8447)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.1712
##
##                   Kappa : 0.4389
##
##  Mcnemar's Test P-Value : 0.4533
##
##             Sensitivity : 0.7619
##             Specificity : 0.7000
##          Pos Pred Value : 0.8421
##          Neg Pred Value : 0.5833
##              Prevalence : 0.6774
##          Detection Rate : 0.5161
##    Detection Prevalence : 0.6129
##       Balanced Accuracy : 0.7310
##
##        'Positive' Class : Abnormal
##
```

```
print(ggplot(KNNModel))
```

Plot of ROC vs Number of Neighbors



Cleaned Dataset

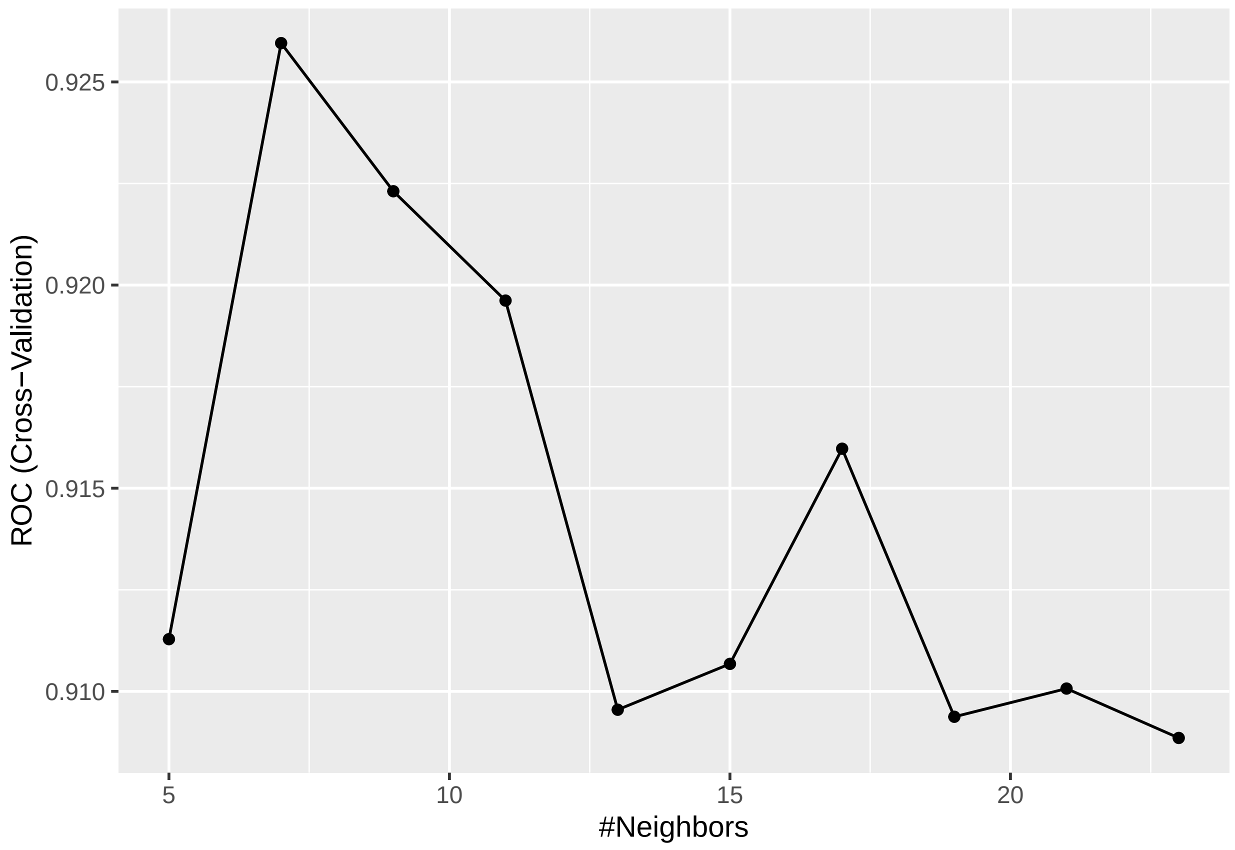
```
KNNModel2 <- train(class~.,
                   trainData2,
                   method="knn",
                   metric="ROC",
                   preProcess = c('center', 'scale'),
                   tuneLength=10,
                   trControl=fitControl2)

KNNPred2 <- predict(KNNModel2, testData2)
KNNConfMatrix2 <- confusionMatrix(KNNPred2, as.factor(testData2$class), positive = "Abnormal")
KNNConfMatrix2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       32      5
##   Normal         10     15
##
##                Accuracy : 0.7581
##                  95% CI : (0.6326, 0.8578)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.1089
##
```

```
##                   Kappa : 0.4804
##
##  Mcnemar's Test P-Value : 0.3017
##
##             Sensitivity : 0.7619
##             Specificity : 0.7500
##          Pos Pred Value : 0.8649
##          Neg Pred Value : 0.6000
##              Prevalence : 0.6774
##          Detection Rate : 0.5161
##    Detection Prevalence : 0.5968
##       Balanced Accuracy : 0.7560
##
##        'Positive' Class : Abnormal
##
```

```
print(ggplot(KNNModel2))
```

Plot of ROC vs Number of Neighbors



Once again there is very little impact cause by removing the highly correlated variables, there is only a small increase in the accuracy from the Original to the Cleaned Dataset.

Neural Network with LDA Model

Artificial Neural Networks [NN] is a type of mathematical algorithms that imitates the simulation of networks of biological neurons. It tries to imatate the human brains neural pathways.

An NN consists of nodes (called neurons) and edges (called synapses). Input data is transmitted through the weighted synapses to the neurons. The neurons make calculations that are processed and then passed onto the next neurons passed to a neuron representing the output (this implise that end).

NN creates a weighting for connections between neurons. Once all weightings have been trained, the NN is able to use these connection make predictions from the input data. NN make use of both forward and Backpropagation. Backpropagations the set of learning rules used to guide NN. (Yiu 2019)

Original Dataset

```
trainDataLda <- ldaDataPredict[datSamplingIndex, ]
testDataLda <- ldaDataPredict[-datSamplingIndex, ]
```

Test and Training data

```
NNLDAModel <- train(class~.,
                 trainDataLda,
                 method="nnet",
                 metric="ROC",
                 preProcess=c('center', 'scale'),
                 tuneLength=10,
                 trace=FALSE,
                 trControl=fitControl)

NNLdaPred <- predict(NNLDAModel, testDataLda)
NNLdaConfMatrix <- confusionMatrix(NNLdaPred, as.factor(testDataLda$class), positive = "Abnormal")
NNLdaConfMatrix
```

Model

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       35      3
##   Normal          7     17
##
##                Accuracy : 0.8387
##                  95% CI : (0.7233, 0.9198)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.003344
##
##                   Kappa : 0.6493
##
##  Mcnemar's Test P-Value : 0.342782
##
##             Sensitivity : 0.8333
##             Specificity : 0.8500
##          Pos Pred Value : 0.9211
##          Neg Pred Value : 0.7083
##              Prevalence : 0.6774
##          Detection Rate : 0.5645
##    Detection Prevalence : 0.6129
##       Balanced Accuracy : 0.8417
##
##        'Positive' Class : Abnormal
##
```

Cleaned Dataset

```
trainDataLda2 <- ldaDataPredict[datSamplingIndex, ]
testDataLda2 <- ldaDataPredict[-datSamplingIndex, ]
```

Test and Training data

```
NNLDAModel2 <- train(class~.,
                 trainDataLda2,
                 method="nnet",
                 metric="ROC",
                 preProcess=c('center', 'scale'),
                 tuneLength=10,
                 trace=FALSE,
                 trControl=fitControl2)

NNLdaPred2 <- predict(NNLDAModel2, testDataLda2)
NNLdaConfMatrix2 <- confusionMatrix(NNLdaPred2, as.factor(testDataLda2$class), positive = "Abnormal")
NNLdaConfMatrix2
```

Model

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       35      3
##   Normal          7     17
##
##                Accuracy : 0.8387
##                  95% CI : (0.7233, 0.9198)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.003344
##
##                   Kappa : 0.6493
##
##  Mcnemar's Test P-Value : 0.342782
##
##             Sensitivity : 0.8333
##             Specificity : 0.8500
##          Pos Pred Value : 0.9211
##          Neg Pred Value : 0.7083
##               Prevalence : 0.6774
##           Detection Rate : 0.5645
##     Detection Prevalence : 0.6129
##        Balanced Accuracy : 0.8417
##
##         'Positive' Class : Abnormal
##
```

There has been no impact from using the Cleaned Dataset

Neural Network with PCA Model

Original Dataset

```
NNPCAModel <- train(class~.,
                        trainData,
                        method="nnet",
                        metric="ROC",
                        preProcess=c('center', 'scale', 'pca'),
                        tuneLength=10,
                        trace=FALSE,
                        trControl=fitControl)

NNPcaPred <- predict(NNPCAModel, testData)
NNPcaConfMatrix <- confusionMatrix(NNPcaPred, as.factor(testData$class), positive = "Abnormal")
NNPcaConfMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       36      5
##   Normal          6     15
##
##                Accuracy : 0.8226
##                  95% CI : (0.7047, 0.908)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.008074
##
##                   Kappa : 0.5993
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8571
##             Specificity : 0.7500
##          Pos Pred Value : 0.8780
##          Neg Pred Value : 0.7143
##              Prevalence : 0.6774
##          Detection Rate : 0.5806
##    Detection Prevalence : 0.6613
##       Balanced Accuracy : 0.8036
##
##        'Positive' Class : Abnormal
```

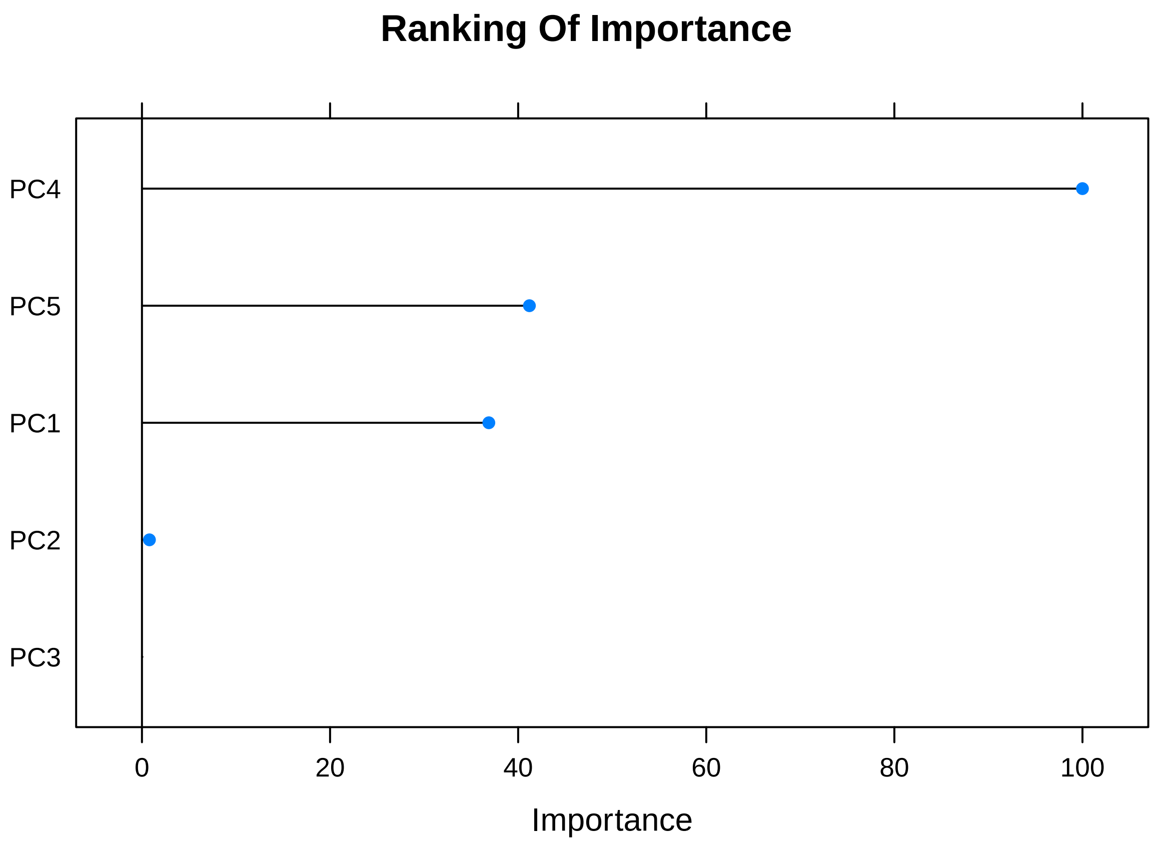
```
##
```

```r
varImp(NNPCAModel)
```

Plot of Variable Importance

```
## nnet variable importance
##
##       Overall
## PC4 100.0000
## PC5  41.1974
## PC1  36.8820
## PC2   0.7919
## PC3   0.0000
```

```r
plot(varImp(NNPCAModel), main="Ranking Of Importance")
```



Cleaned Dataset

```r
NNPCAModel2 <- train(class~.,
                        trainData2,
                        method="nnet",
                        metric="ROC",
                        preProcess=c('center', 'scale', 'pca'),
                        tuneLength=10,
                        trace=FALSE,
                        trControl=fitControl2)

NNPcaPred2 <- predict(NNPCAModel2, testData2)
NNPcaConfMatrix2 <- confusionMatrix(NNPcaPred2, as.factor(testData2$class), positive = "Abnormal")
NNPcaConfMatrix2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       35      5
##   Normal          7     15
##
##                Accuracy : 0.8065
##                  95% CI : (0.6863, 0.8958)
##     No Information Rate : 0.6774
##     P-Value [Acc > NIR] : 0.01765
##
##                   Kappa : 0.5684
##
##  Mcnemar's Test P-Value : 0.77283
##
##             Sensitivity : 0.8333
##             Specificity : 0.7500
##          Pos Pred Value : 0.8750
##          Neg Pred Value : 0.6818
##              Prevalence : 0.6774
##          Detection Rate : 0.5645
##    Detection Prevalence : 0.6452
##       Balanced Accuracy : 0.7917
##
##        'Positive' Class : Abnormal
##
```

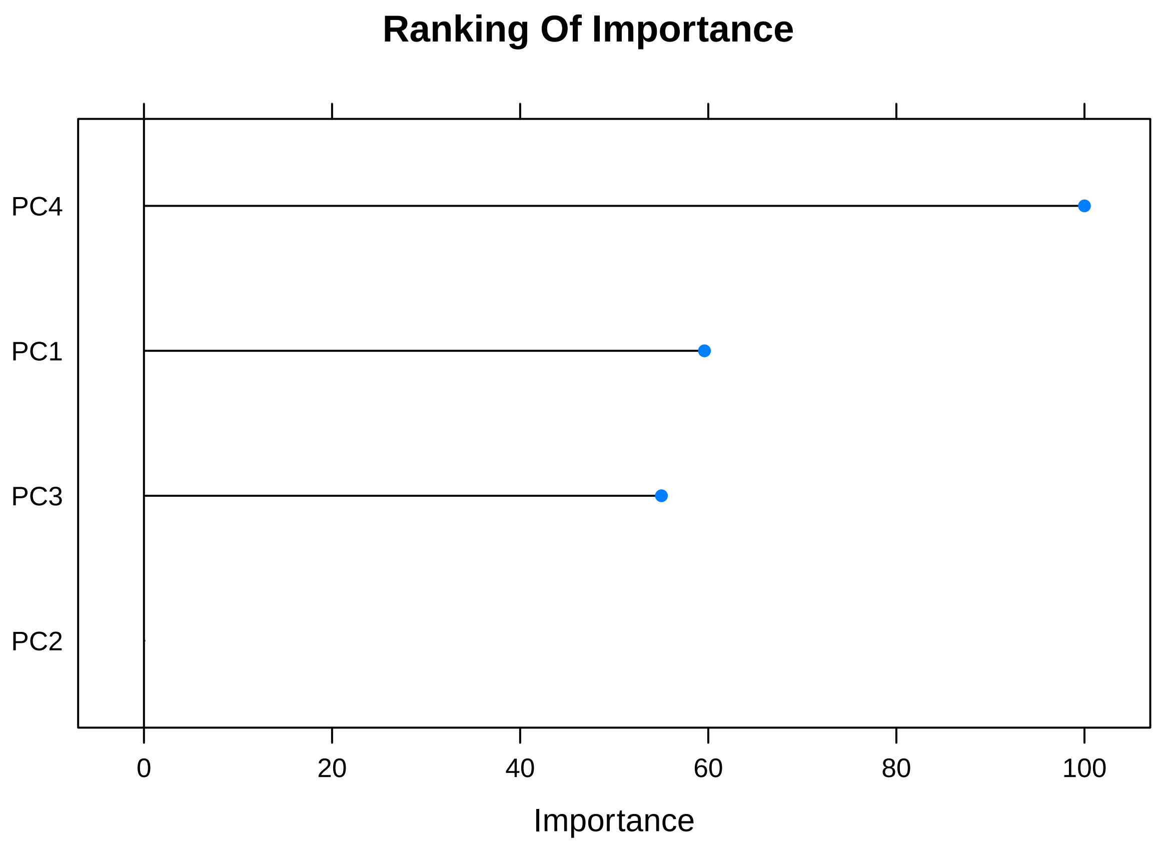
There has been a negative impact from using the Cleaned Dataset on the NN using PCA. This can be seen via the decrease in accuracy from 0.8226 to 0.8065, from the Original to the Clean Dataset respectively.

```
varImp(NNPCAModel2)
```

Plot of Variable Importance

```
## nnet variable importance
##
##     Overall
## PC4  100.00
## PC1   59.60
## PC3   55.02
## PC2    0.00
```

```r
plot(varImp(NNPCAModel2), main="Ranking Of Importance")
```



Results

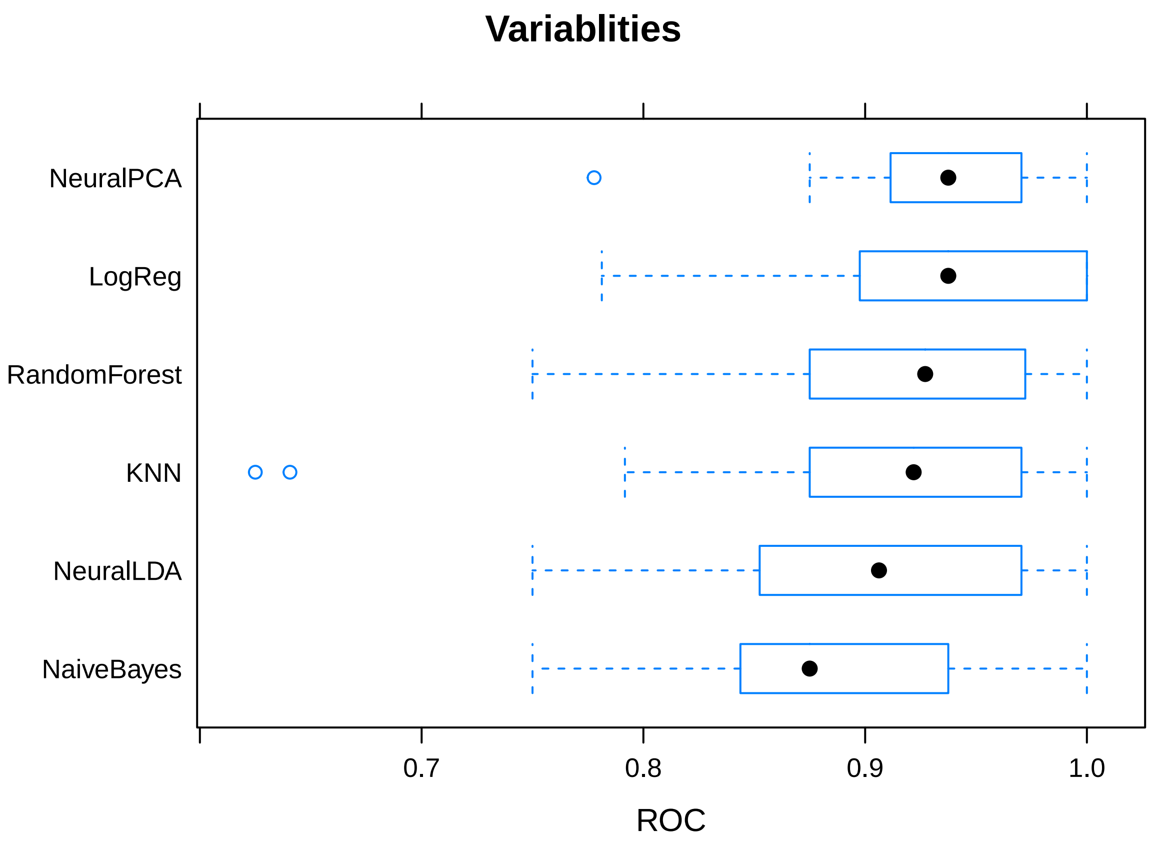
Original DataSet

Model Comparisions

```
models <- list(NaiveBayes = NaiveBayesModel,
               LogReg = LogRegModel,
               RandomForest = RandomforestModel,
               KNN = KNNModel,
               NeuralPCA = NNPCAModel,
               NeuralLDA = NNLDAModel)

modelsResults <- resamples(models)
summary(modelsResults)
```

```
##
## Call:
## summary.resamples(object = modelsResults)
##
## Models: NaiveBayes, LogReg, RandomForest, KNN, NeuralPCA, NeuralLDA
## Number of resamples: 20
##
## ROC
##                   Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## NaiveBayes   0.7500000 0.8437500 0.8750000 0.8727431 0.9375000    1    0
## LogReg       0.7812500 0.9019097 0.9375000 0.9303819 1.0000000    1    0
## RandomForest 0.7500000 0.8750000 0.9270833 0.9119792 0.9722222    1    0
## KNN          0.6250000 0.8750000 0.9218750 0.8969618 0.9696181    1    0
## NeuralPCA    0.7777778 0.9140625 0.9375000 0.9361111 0.9696181    1    0
## NeuralLDA    0.7500000 0.8567708 0.9062500 0.9100694 0.9696181    1    0
##
## Sens
##                   Min.   1st Qu.    Median      Mean   3rd Qu.  Max. NA's
## NaiveBayes   0.5555556 0.6666667 0.7500000 0.7465278 0.7777778 0.875    0
## LogReg       0.6250000 0.7708333 0.8819444 0.8631944 1.0000000 1.000    0
## RandomForest 0.7500000 0.8506944 0.8819444 0.8854167 1.0000000 1.000    0
## KNN          0.6250000 0.7708333 0.8750000 0.8659722 1.0000000 1.000    0
## NeuralPCA    0.5000000 0.7777778 0.8819444 0.8763889 1.0000000 1.000    0
## NeuralLDA    0.7500000 0.8506944 0.8888889 0.8819444 0.9166667 1.000    0
##
## Spec
##              Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
## NaiveBayes   0.50  0.7500  0.750 0.8375  1.0000    1    0
## LogReg       0.25  0.5000  0.875 0.7875  1.0000    1    0
## RandomForest 0.50  0.5000  0.750 0.7375  1.0000    1    0
## KNN          0.25  0.5000  0.750 0.7250  0.8125    1    0
## NeuralPCA    0.50  0.7500  0.750 0.8250  1.0000    1    0
## NeuralLDA    0.00  0.6875  0.750 0.7375  1.0000    1    0
```

```r
print(bwplot(modelsResults, metric="ROC",main = "Variablities"))
```



As we can observe from the plot above, Three models, Naive Bayes , Neural LDA and Random Forest have great variability.

The Receiver Operating characteristic Curve [ROC] is a graph that shows the performance of a classification model at all classification thresholds. AUC is the metric measure of the ROC curve of each model. This metric is independent of any threshold (Narkhede, n.d.).

The NN LDA model managed to achieve a great Area Under the ROC Curve [AUC] but has a high variability. Secondly the NN PCA managed to also achieve a good Area Under the ROC Curve but with less variability that the NN LDA which indicates that it is performing better.

```r
confMatrixs <- list(
  NaiveBayes = NBConfMatrix,
  LogReg = LogRegConfMatrix,
  RandomForest = RFConfMatrix,
  KNN = KNNConfMatrix,
  NeuralPCA = NNPcaConfMatrix,
  NeuralLDA = NNPcaConfMatrix)

ConfMatrixResults <- sapply(confMatrixs, function(x) x$byClass)
ConfMatrixResults %>% knitr::kable()
```

	NaiveBayes	LogReg	RandomForest	KNN	NeuralPCA	NeuralLDA
Sensitivity	0.5952381	0.8571429	0.8333333	0.7619048	0.8571429	0.8571429
Specificity	0.9000000	0.7500000	0.7500000	0.7000000	0.7500000	0.7500000
Pos Pred Value	0.9259259	0.8780488	0.8750000	0.8421053	0.8780488	0.8780488
Neg Pred Value	0.5142857	0.7142857	0.6818182	0.5833333	0.7142857	0.7142857
Precision	0.9259259	0.8780488	0.8750000	0.8421053	0.8780488	0.8780488
Recall	0.5952381	0.8571429	0.8333333	0.7619048	0.8571429	0.8571429
F1	0.7246377	0.8674699	0.8536585	0.8000000	0.8674699	0.8674699
Prevalence	0.6774194	0.6774194	0.6774194	0.6774194	0.6774194	0.6774194
Detection Rate	0.4032258	0.5806452	0.5645161	0.5161290	0.5806452	0.5806452
Detection Prevalence	0.4354839	0.6612903	0.6451613	0.6129032	0.6612903	0.6612903
Balanced Accuracy	0.7476190	0.8035714	0.7916667	0.7309524	0.8035714	0.8035714

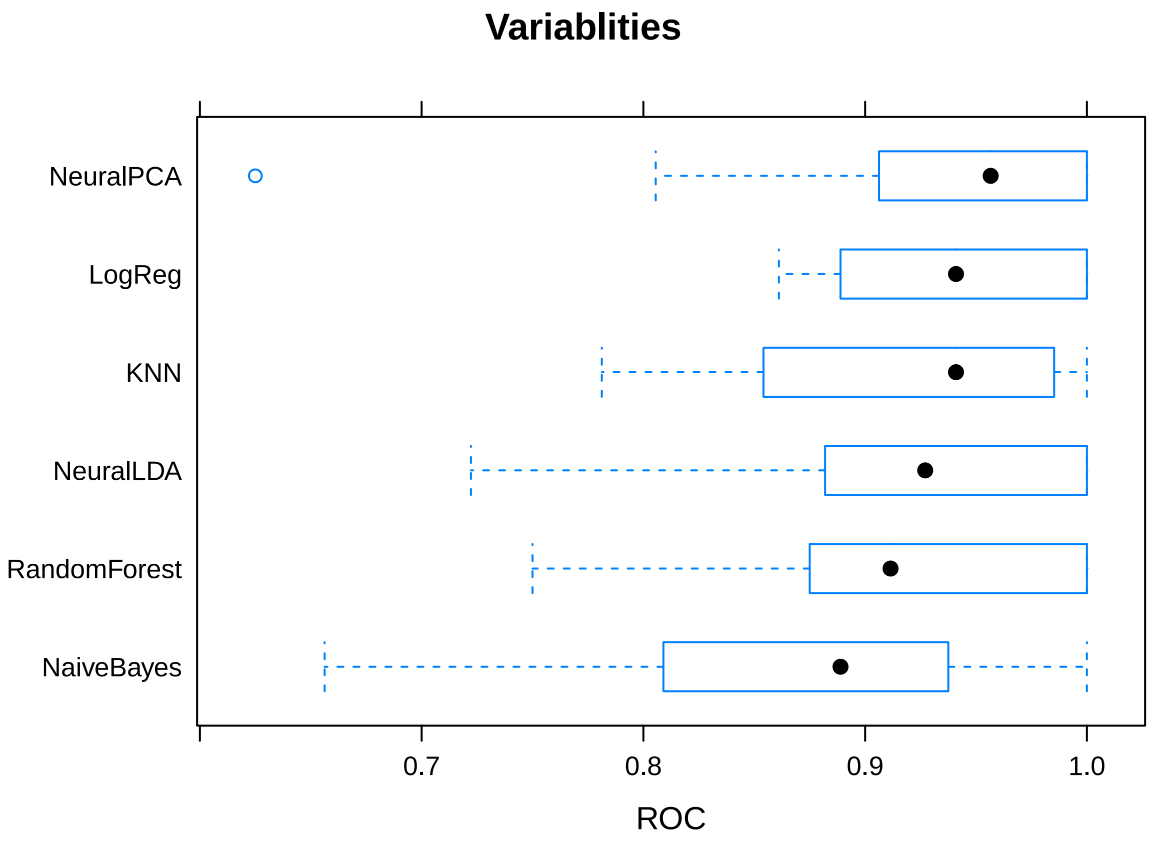
Cleaned DataSet

Model Comparisions

```r
models2 <- list(NaiveBayes = NaiveBayesModel2,
                    LogReg = LogRegModel2,
                    RandomForest = RandomforestModel2,
                    KNN = KNNModel2,
                    NeuralPCA = NNPCAModel2,
                    NeuralLDA = NNLDAModel2)

modelsResults2 <- resamples(models2)
summary(modelsResults2)
```

```
##
## Call:
## summary.resamples(object = modelsResults2)
##
## Models: NaiveBayes, LogReg, RandomForest, KNN, NeuralPCA, NeuralLDA
## Number of resamples: 20
##
## ROC
##                   Min.   1st Qu.    Median      Mean  3rd Qu. Max. NA's
## NaiveBayes   0.6562500 0.8107639 0.8888889 0.8736111 0.937500    1    0
## LogReg       0.8611111 0.8888889 0.9409722 0.9371528 1.000000    1    0
## RandomForest 0.7500000 0.8750000 0.9114583 0.9211806 1.000000    1    0
## KNN          0.7812500 0.8576389 0.9409722 0.9259549 0.984809    1    0
## NeuralPCA    0.6250000 0.9062500 0.9565972 0.9305556 1.000000    1    0
## NeuralLDA    0.7222222 0.8854167 0.9270833 0.9208333 1.000000    1    0
##
## Sens
##                   Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## NaiveBayes   0.3750000 0.6250000 0.7500000 0.7409722 0.8784722    1    0
## LogReg       0.5000000 0.7777778 0.8819444 0.8743056 1.0000000    1    0
## RandomForest 0.6250000 0.7777778 0.8819444 0.8798611 1.0000000    1    0
## KNN          0.7500000 0.8506944 0.8888889 0.9048611 1.0000000    1    0
## NeuralPCA    0.7500000 0.8750000 0.9444444 0.9229167 1.0000000    1    0
## NeuralLDA    0.6666667 0.8437500 0.8888889 0.8930556 1.0000000    1    0
##
## Spec
##              Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
## NaiveBayes   0.50  0.7500   1.00 0.8875  1.0000    1    0
## LogReg       0.50  0.7500   0.75 0.8000  1.0000    1    0
## RandomForest 0.50  0.5000   0.75 0.7625  1.0000    1    0
## KNN          0.50  0.6875   0.75 0.7500  0.8125    1    0
## NeuralPCA    0.00  0.7500   0.75 0.7500  1.0000    1    0
## NeuralLDA    0.25  0.5000   0.75 0.7375  1.0000    1    0
```

```r
bwplot(modelsResults2, metric="ROC",main = "Variablities")
```



Comparing the Cleaned Dataset to the original it is important to note although the had been no real differences seen when running the models. There is a clear difference when comparing the variability plots. It is clear that by removing the highly correlated variable reduced some of the variability in the ROC for particular Models which is good, but also has seemed to increase variability in some of the other models

From the plot above it is clear that the models now experiencing the greatest variability are Naive Bayes and Neural LDA. The models now with the best ROC are the LogReg and Neural PCA.

```r
confMatrixs2 <- list(
  NaiveBayes = NBConfMatrix2,
  LogReg = LogRegConfMatrix2,
  RandomForest = RFConfMatrix2,
  KNN = KNNConfMatrix2,
  NeuralPCA = NNPcaConfMatrix2,
  NeuralLDA = NNPcaConfMatrix2)

ConfMatrixResults2 <- sapply(confMatrixs2, function(x) x$byClass)
ConfMatrixResults2 %>% knitr::kable()
```

	NaiveBayes	LogReg	RandomForest	KNN	NeuralPCA	NeuralLDA
Sensitivity	0.6428571	0.8571429	0.8333333	0.7619048	0.8333333	0.8333333
Specificity	0.9500000	0.7500000	0.7000000	0.7500000	0.7500000	0.7500000
Pos Pred Value	0.9642857	0.8780488	0.8536585	0.8648649	0.8750000	0.8750000
Neg Pred Value	0.5588235	0.7142857	0.6666667	0.6000000	0.6818182	0.6818182
Precision	0.9642857	0.8780488	0.8536585	0.8648649	0.8750000	0.8750000
Recall	0.6428571	0.8571429	0.8333333	0.7619048	0.8333333	0.8333333
F1	0.7714286	0.8674699	0.8433735	0.8101266	0.8536585	0.8536585
Prevalence	0.6774194	0.6774194	0.6774194	0.6774194	0.6774194	0.6774194
Detection Rate	0.4354839	0.5806452	0.5645161	0.5161290	0.5645161	0.5645161
Detection Prevalence	0.4516129	0.6612903	0.6612903	0.5967742	0.6451613	0.6451613
Balanced Accuracy	0.7964286	0.8035714	0.7666667	0.7559524	0.7916667	0.7916667

Discussion

The following metrics will be compared in this discussion:

- Accuracy: It is the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to turn it into a percentage.
- Precision is the number of True Positives divided by the number of True Positives and False Positives. Or can be referred to as the number of positive predictions divided by the total number of positive class values predicted. This is refered to as the Positive Predictive Value (PPV). A low precision can indicate a large number of False Positives.
- Sensitivity or True Positive Rate [recall] is the number of True Positives divided by the number of True Positives and the number of False Negatives. Or can be seen as the number of positive predictions divided by the number of positive class values in the test data. Recall can be thought of as a measure of a classifiers completeness. A low recall indicates many False Negatives.
- The F1 Score is calculated from: 2 x ((precision x recall) / (precision + recall)). It is also called the F Score or the F Measure. The F1 score conveys the balance between the precision and the recall.

Original Dataset

From within the original datset :

The best results for sensitivity (detection of Abnormal features of orthopedic patients) is NN Model with the PCA model which also has the best F1 score. It should be note the NN Model with PCA model and the NN Model with LDA have the same Sensitivity score. As the PCA model has the best F1 store it has been rated as best overall.

```
MaxConfMatrix <- apply(ConfMatrixResults, 1, which.is.max)
OutputReport <- data.frame(metric=names(MaxConfMatrix),
                           bestModel=colnames(ConfMatrixResults)[MaxConfMatrix],
                           value=mapply(function(x,y) {ConfMatrixResults[x,y]},
                                        names(MaxConfMatrix),
                                        MaxConfMatrix))
rownames(OutputReport) <- NULL
OutputReport
```

```
##                  metric  bestModel     value
## 1           Sensitivity  NeuralLDA 0.8571429
## 2           Specificity NaiveBayes 0.9000000
## 3        Pos Pred Value NaiveBayes 0.9259259
## 4        Neg Pred Value     LogReg 0.7142857
## 5             Precision NaiveBayes 0.9259259
## 6                Recall     LogReg 0.8571429
## 7                    F1  NeuralPCA 0.8674699
## 8            Prevalence  NeuralPCA 0.6774194
## 9        Detection Rate  NeuralLDA 0.5806452
## 10 Detection Prevalence  NeuralLDA 0.6612903
## 11    Balanced Accuracy     LogReg 0.8035714
```

Cleaned Dataset

From within the Cleaned Dataset datset :

The best results for sensitivity (detection of Abnormal features of orthopedic patients) is the LogReg which also has the best F1 score.

```r
MaxConfMatrix2 <- apply(ConfMatrixResults2, 1, which.is.max)
OutputReport2 <- data.frame(metric=names(MaxConfMatrix2),
                            bestModel=colnames(ConfMatrixResults2)[MaxConfMatrix2],
                            value=mapply(function(x,y) {ConfMatrixResults2[x,y]},
                                         names(MaxConfMatrix2),
                                         MaxConfMatrix2))
rownames(OutputReport2) <- NULL
OutputReport2
```

```
##                  metric    bestModel     value
## 1           Sensitivity       LogReg 0.8571429
## 2           Specificity   NaiveBayes 0.9500000
## 3        Pos Pred Value   NaiveBayes 0.9642857
## 4        Neg Pred Value       LogReg 0.7142857
## 5             Precision   NaiveBayes 0.9642857
## 6                Recall       LogReg 0.8571429
## 7                    F1       LogReg 0.8674699
## 8            Prevalence RandomForest 0.6774194
## 9        Detection Rate       LogReg 0.5806452
## 10 Detection Prevalence       LogReg 0.6612903
## 11    Balanced Accuracy       LogReg 0.8035714
```

Conclusion

This paper treats the Biomechanical features of orthopedic patients diagnosis problem as a pattern classification problem. This report investigates several machine learning models the optimal model is selected by selecting a high accuracy level combined with a low rate of false-negatives (meaning high sensitivity).

From this it can be concluded that from the Original Dataset the NN Model with the PCA Model had the optimal results for F1 (0.8674699), Sensitivity (0.8571429) and Balanced Accuracy (0.8035714).

Secondly it can also be concluded that from the Cleaned Dataset the LogReg Model had the optimal results for F1 (0.8674699), Sensitivity (0.8571429) and Balanced Accuracy (0.8035714).

To draw a conclusion in good machine learning practise it is always best to remove highly correlated variable. Therefore one should theoretical use the Cleaned Dataset with the LogReg model when making predictions.

Future work

Future work on this particular project should focus on checking whether the predictions made by the models that had the optimal results actually give outputs that can be trusted. This could be done by supplying the Models with unfinished data and comparing the models results to verified data.

Also future work should focus on trying to improve the models accuracy and reducing the variability further.

Appendix

Environment

```
print("Operating System:")
```

```
## [1] "Operating System:"
```

```
version
```

```
##                _
## platform       x86_64-apple-darwin17.0
## arch           x86_64
## os             darwin17.0
## system         x86_64, darwin17.0
## status
## major          4
## minor          0.3
## year           2020
## month          10
## day            10
## svn rev        79318
## language       R
## version.string R version 4.0.3 (2020-10-10)
## nickname       Bunny-Wunnies Freak Out
```

References

Harrison, Onel. n.d. "Machine Learning Basics with the K-Nearest Neighbors Algorithm." https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

Learning, UCI Machine. n.d. "Biomechanical Features of Orthopedic Patients." https://www.kaggle.com/uciml/biomechanical-features-of-orthopedic-patients.

Moore, Kristeen. 2018. "Spondylolisthesis." https://www.healthline.com/health/spondylolisthesis.

"Naive Bayesian." n.d. https://www.saedsayad.com/naive_bayesian.htm.

Narkhede, Sarang. n.d. "Understanding Auc - Roc Curve." https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

Peixeiro, Marco. n.d. "The Complete Guide to Classification in Python." https://towardsdatascience.com/the-complete-guide-to-classification-in-python-b0e34c92e455.

Pramoditha, Rukshan. n.d. "Random Forests — an Ensemble of Decision Trees." https://towardsdatascience.com/random-forests-an-ensemble-of-decision-trees-37a003084c6c.

ProjectPro. n.d. "Principal Component Analysis Tutorial." Accessed 2020. https://www.dezyre.com/data-science-in-python-tutorial/principal-component-analysis-tutorial#:~:text=The%20main%20idea%20of%20principal,up%20to%20the%20maximum%20extent.&text=As%20a%20layman%2C%20it%20is%20a%20method%20of%20summarizing%20data.

Staff, Mayo Clinic. n.d. "Herniated Disk." https://www.mayoclinic.org/diseases-conditions/herniated-disk/symptoms-causes/syc-20354095.

University, Andrews. n.d. “Applied Statistics - Lesson 5.” https://www.andrews.edu/~calkins/math/edrm611/edrm05.htm#:~:text=Correlation%20coefficients%20whose%20magnitude%20are,can%20be%20considered%20highly%20correlated.

Yiu, Tony. 2019. “Understanding Neural Networks.” https://towardsdatascience.com/understanding-neural-networks-19020b758230.