Diabetes Data Analysis

Luc Chen
Curtis Pan
Hengyuan Liu
Weilin Cheng
Ka Wai Sit

University of California, Davis

1 Introduction

The prevalence of diabetes has been a growing concern worldwide, affecting millions of individuals and posing significant health challenges. In this project, our dataset was provided by the US National Institute of Diabetes and Digestive and Kidney Diseases. This dataset captures diabetes status from a population of women aged 21+ who are not only of Pima Indian heritage but also reside near the city of Phoenix, Arizona. Our primary goal is to explore and analyze the data to develop a predictive model for determining diabetes status. With our project, we seek to promote greater understanding among the general public regarding the influential factors that influence the probability of diabetes, ultimately emphasizing the vital significance of prioritizing personal well-being.

2 Dataset Description

We will utilize the Pima Indians Diabetes dataset from the mlbench package, consisting of 768 observations and 8 predictor variables: pregnant, glucose, pressure, triceps, insulin, mass, pedigree, and age. The response variable is a binary indicator of diabetes presence or absence. During exploratory analysis, we discovered zero values in certain variables, which were set to NA as per the R Documentation guidelines. To address the data imbalance of 262 observations without diabetes and 130 observations with diabetes, we employed k-fold cross-validation for a more accurate evaluation of the model's performance on unseen data. Consequently, anomalous observations with zero values were excluded, resulting in a dataset of 392 observations for analysis.

3 Visualization

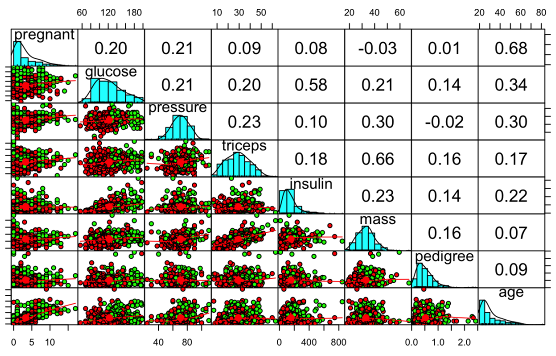


Figure 1: Diabetes Data Visualization

Once we eliminated the missing values from the original dataset, we proceeded to create paired plots for visualization. In these paired plots, the variable names are presented along the outer edges of the matrix. The diagonal boxes contain density plots for each variable, providing a visual representation of their distributions. The boxes in the lower left corner represent scatterplots, displaying the relationships between each pair of variables. Lastly, the boxes in the upper right corner indicate the Pearson correlation coefficient, which quantifies the strength and direction of the linear relationship between each pair of variables. For example, the highest correlation is between pregnant and age which is 0.68.

4 Methodologies

In our analysis, we aim to utilize either Linear Discriminant Analysis (LDA) or Quadratic Discriminant Analysis (QDA). To determine which method is most suitable, we conducted a preliminary assessment using Box's M test to examine whether the two classes (positive and negative) have the same covariance matrices. Box's M test is suitable in this case as it is more sensitive to large datasets. We want to test

$$H_0 : \Sigma_{pos} = \Sigma_{neg} \ vs. \ H_a : \Sigma_{pos} \neq \Sigma_{neg} \tag{1}$$

The test resulted in a chi-square statistic of 184.78, with 36 degrees of freedom. The p-value associated with the test was found to be less than 2.2e-16. Therefore, we can conclude that there is strong evidence that the covariance matrices of those with diabetes and sans-diabetes are significantly different from each other, violating the validity of LDA. Instead, we use QDA in order to test whether the covariance matrices are equal to one another.

We employed QDA to categorize observations as either positive or negative. Initially, we partitioned the data into a 70% training set and a 30% test set. Subsequently, performing QDA on the training data, conducting 1000 iterations of random sampling. From these iterations, we computed the average result of 79.90% with the full model. We conducted additional experiments with various train-test splits to assess the impact on the results. Our final analysis involved 1000 iterations, and we observed that a 50-50 train-test split yielded a result of 79.29% accuracy, 60-40 train-test split yielded a result of 79.69% accuracy, 80-20 train-test split yielded a result of 80.22% accuracy, and 90-10 train-test split yield 80.22%. In order to combat overfitting of our model and more robust model performance, especially working with the limited data, we used k folds cross validation to test our model's performance. We performed a 5-fold cross-validation on the dataset for 1000 iterations, and returned the average accuracy of the model over these iterations, the accuracy is 77.69%. For this project, a 70-30 train-test split was utilized for evaluating the model's performance.

4.1 QDA-Decision Boundary

$$Y_{QDA} = \begin{cases} S_1(X) - S_2(X) > 0, & Negative \\ S_1(X) - S_2(X) < 0, & Positive \end{cases} \tag{2}$$

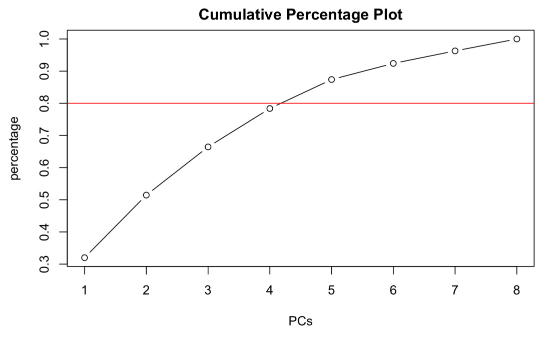
	Positive	**Negative**
Positive	25	9
Negative	15	69

Table 1: Confusion Matrix for QDA Decision Boundary

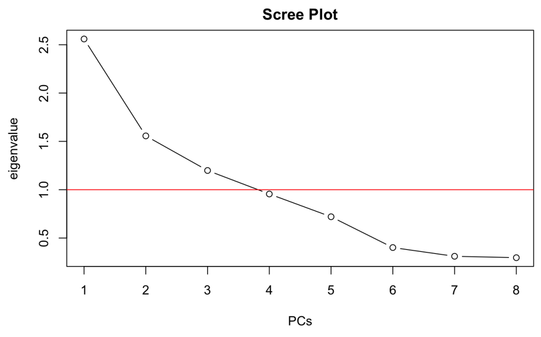
As Equation(2) shows above, our decision boundary is when $S_1(X) - S_2(X)$ (See detail in **Appendix** [**B**]) is larger than 0, classify diabetes as negative, and if it's less than 0, classify it as positive. In our case, we didn't observe $S_1(X) - S_2(X)$ equal to 0, which would mean data point is located on the decision boundary, thus having an equal chance of being classified as either positive or negative. In Table 1 confusion matrix, the column means the actual positive and negative diabetes and the row means the predicted positive and negative diabetes.
This is how we got this confusion table and the accuracy rate by QDA decision boundary. First, we randomly split the full data set into 70-30 training and testing across 100 iterations in order to test our decision boundary. Second, for each time, we obtained the coefficients of the decision boundary based on the training data and tested them on the testing data 100 times. Lastly, we obtained an average confusion table (See Table 1), and an average accuracy rate of 79%. We find the result consistent with the QDA package that we showed previously. The detailed calculations for the accuracy rate and the equations used for determining the QDA decision boundary can be found in **Appendix [A] and [B]** respectively.

4.2 Principal Component Analysis (PCA)

We are now focused on comparing the error rates between the QDA with the full model and the QDA with different PCA reductions. Through principal component analysis, we can reduce the number of dimensions of the dataset, and isolate which health factors contribute the most to the explained variation within the dataset.
Illustrated by the Scree Plot we can implement the Overall Threshold method, where we choose the number of principal components based on which eigenvalues are greater than the average of all the eigenvalues, thus



(a) Scree plot displaying the eigenvalues of the Principal Components (PCs)



(b) Cumulative Percentage of Variance Explained by Principal Components (PCs)

Figure 2: Method for choosing number of PCs

we chose three principal components. However, based on the Contribution Ratio approach we try to retain a sufficient amount of components up to 80% of total explained variance. According to the Cumulative Percentage plot, we found that the first four principal components contained up to 78.38% of the explained variation within the dataset. To meet the cutoff for the 80% of total explained variance, we could use the first five principal components.

Based on the findings, there is a discrepancy observed between the number of principal components recommended by the Cumulative Percentage Plot and the Scree Plot. We repeated the process of Box M on the full model on the three different PC models as well and found they all corroborate the result of rejecting the null hypothesis, thus we must use QDA on all three reduced models as well. To ensure optimal performance, we conducted a comparison between the accuracy obtained using three principal components, four principal components, and five principal components. First, we want to see the performance of PCA when the number of PCs equals to three. The projects of our data on the space defined by the third principal component. Following the PCA transformation, we apply the QDA to our new dataset, the resulting accuracy of which is 74.6% for three PC's. Compared with the performance of four PCs with an accuracy rate of 75.7%, with five PC's the accuracy is around 78.5%. (See **Appendix [C]** first 3-5 PCAs confusion table) All of these accuracy rates were taken from the average of 1000 simulations. The reduced model seems to lose accuracy compared to the full model since some useful information may be lost from reducing the number of dimensions.

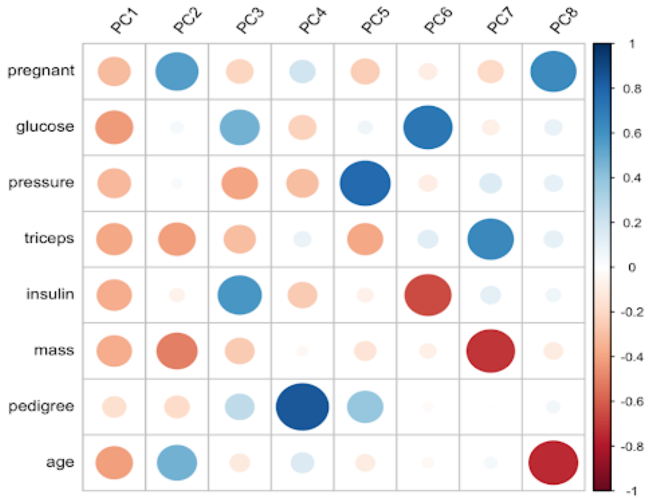


Figure 3: Correlation and Heat Map of Variable Relationships

In order to conduct a comprehensive comparison, we employed the QDA default function on a multiple number of principal components. In order to ensure robustness and reliability, we performed 1000 iterations for each setting, and we generated an average confusion table. This approach allowed us to contrast the results obtained from our custom function with those produced by the QDA default function, facilitating a thorough evaluation of their performance.

The correlation heat map (Figure 3) reinforces the findings from the scree plot and cumulative percentage plot, indicating that the first four principal components (PC1, PC2, PC3, and PC4) effectively capture the majority of the information across all variables.

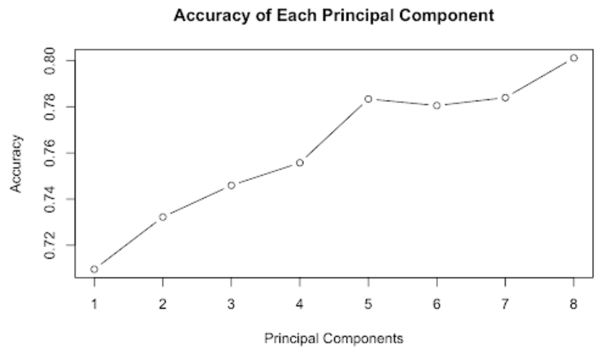


Figure 4: Accuracy of QDA with Different Principal Components

Based on Figure 4, we conducted a comparison of the accuracy of QDA with different numbers of principal components. The results indicate that an increased number of principal components corresponds to higher accuracy in QDA classification. After analyzing the data, we determined that utilizing five principal components yields the highest accuracy. Interestingly, we observed a decrease in accuracy when using six or seven principal components, which is an unexpected finding. We suspect that the additional principal components may contain more noise or less relevant information compared to the initial five components. This noise can introduce inconsistencies or misleading patterns, leading to a decrease in accuracy.

5 Summary and Conclusions

Overall, we found that the quadratic discriminant analysis classification method resulted in an accuracy rate of the full model's decision boundary of around 79%. From PCA, we reduced the dimensions from eight dependent variables to principal components aiming for the lowest number of principal components, and combinations of the variables, while still maintaining a high degree of explained variance within the new PCA dataset. We found one PC has the lowest accuracy which is about 71% and the highest accuracy is eight PC which is about 80%. The accuracy of the five PC models results in 78% accuracy while also reducing the number of dimensions to a suitable amount, making it a suitable candidate for the QDA model. The decrease in accuracy could be attributed to the loss of some valuable information from reducing the dimensions into principal components; we found that each reduction in dimensions resulted in a lower accuracy each time. The first four principal components capture most of the explained variation from their combination of variables: PC1 captures the majority of variables besides pedigree, PC2 mainly captures: pregnant, triceps, mass, and age, PC3: glucose, insulin, pressure, PC4 - pedigree, and PC5-pressure. In total the four principal components capture information from all variables. However, there are unseen errors that may cause misclassification, hence the accuracy rate our models produced. There may be better methods of classification regarding the use of these specific health factors in classifying diabetes, however, those are beyond the scope of this project which focuses mainly on the quadratic discriminant analysis method.

6 Appendix

A Accuracy Rate

$$AccuracyRate = \frac{(TruePredictedPositive + TruePredictedNegative)}{(TotalPredictions)} \tag{3}$$

B QDA Decision Boundary

n_1 is the number of data points belong to negative
n_2 is the number of data points belong to positive
$n = n_1 + n_2$
C_1: Condition of negative situation
C_2: Condition of positive situation

$$Given : \hat{\Pi}_1 = \frac{n_1}{n}, \hat{\Pi}_2 = \frac{n_2}{n}, \hat{\mu}_1 = \frac{1}{n_1}\sum_{i \in C_1} X_i, \hat{\mu}_2 = \frac{1}{n_2}\sum_{i \in C_2} X_i$$

$$S_1 = \hat{\Sigma}_1 = \frac{1}{n_1 - 1}\sum_{i \in C_1}(X_i - \hat{\mu}_1)(X_i - \hat{\mu}_1)^T \tag{4}$$

$$S_2 = \hat{\Sigma}_2 = \frac{1}{n_2 - 1}\sum_{i \in C_2}(X_i - \hat{\mu}_2)(X_i - \hat{\mu}_2)^T \tag{5}$$

$$S_1(X) = log(\Pi_1) - \frac{1}{2}\mu_1^T\Sigma_1^{-1}\mu_1 + X^T\Sigma_1^{-1}\mu_1 - \frac{1}{2}X^T\Sigma_1^{-1}X - \frac{1}{2}log(|\Sigma_1|) \tag{6}$$

$$S_2(X) = log(\Pi_2) - \frac{1}{2}\mu_2^T\Sigma_2^{-1}\mu_2 + X^T\Sigma_2^{-1}\mu_2 - \frac{1}{2}X^T\Sigma_2^{-1}X - \frac{1}{2}log(|\Sigma_2|) \tag{7}$$

$$\begin{aligned} S_1(X) - S_2(X) = &\frac{1}{2}X^T\left(\Sigma_2^{-1} - \Sigma_1^{-1}\right)X + X^T\left(\Sigma_1^{-1}\mu_1 - \Sigma_2^{-1}\mu_2\right) + \log\left(\frac{\Pi_1}{\Pi_2}\right) \\ &+ \frac{1}{2}\left(\mu_2^T\Sigma_2^{-1}\mu_2 - \mu_1^T\Sigma_1^{-1}\mu_1\right) + \frac{1}{2}\log\left(\frac{|\Sigma_2|}{|\Sigma_1|}\right) \end{aligned} \tag{8}$$

C Tables

	Positive	**Negative**
Positive	19	10
Negative	20	69

Table 2: Confusion Matrix for First 3 Principal Components by QDA Package

	Positive	**Negative**
Positive	21	10
Negative	19	69

Table 3: Confusion Matrix for First 4 Principal Components by QDA Package

	Positive	**Negative**
Positive	23	10
Negative	16	69

Table 4: Confusion Matrix for First 5 Principal Components by QDA Package

D Contribution

Name:	Contributions
Luc Chen	Implementation for QDA, PCA, K-folds Cross Validation
Hengyuan Liu:	Implementation to Coding and QDA Decision Boundary
Willie Cheng:	Coding for visualization, writing and formatting in Overleaf
Curtis Pan:	Report writing and offer project guidelines
Ka Wai Sit:	Implementation Box's M test, Methodologies/Graph Interpretations

References

[1] R Core Team (2021). PimaIndiansDiabetes: Pima Indians Diabetes Data. R package version 2.1 - 3.1 Retrieved from https://search.r-project.org/CRAN/refmans/mlbench/html/PimaIndiansDiabetes.html

Code Appendix

```
library(mlbench)
library(ggplot2)
library(cluster)
library(psych)
library(mvtnorm)
library(klaR)
library(psych)
library(MASS)
library(devtools)
library(heplots)
library(caret)
library(corrplot)
library(FactoMineR)
library(factoextra)
data(PimaIndiansDiabetes)
head(PimaIndiansDiabetes)
# Data Cleanning
df=subset(PimaIndiansDiabetes,mass!=0)
df=subset(df,triceps!=0)
df=subset(df, insulin!=0)
df=subset(df,glucose!=0)
rownames(df)=NULL
pairs.panels(df[1:8],
             gap = 0,
             bg = c("red", "green", "blue","yellow","orange","purple")[df$diabetes],
             pch = 21)
# Method 1: Package
res <- boxM(df[, 1:8], df[, "diabetes"])
res
#summary(res)
########################################

# Method 2: Manually: Chi-sq and F-approx
#df = PimaIndiansDiabetes
X = df[,1:8]
Y <- as.factor(df$diabetes)
k <- length(levels(Y))
table(df$diabetes)
n1 <- sum(Y == levels(Y)[1])
n2 <- sum(Y == levels(Y)[2])
p <- dim(df)[2] - 1
v1 <- n1 - 1
v2 <- n2 - 1
X_neg_df = df[df$diabetes == "neg",1:8]
X_pos_df = df[df$diabetes == "pos",1:8]
S1 <- cov(X_neg_df) #neg
S2 <- cov(X_pos_df) #pos
Spl <- (v1*S1+v2*S2)/(v1+v2)
vi = c(v1, v2)
C1 = (sum(1/vi)-1/sum(vi))*((2*p^2+3*p-1)/(6*(p+1)*(k-1)))
lnM = 0.5*(v1*log(det(S1))+v2*log(det(S2)))-0.5*(v1+v2)*log(det(Spl))
```

```
M = exp(0.5*(v1*log(det(S1))+v2*log(det(S2)))-0.5*(v1+v2)*log(det(Spl)))
# chi-sq approximation
u = -2*(1-C1)*(lnM)
u # Result: 164.115 > 41.33714 so we reject H0, conlude QDA
# chi-sq statistics
alpha = 0.05
qchisq(1-alpha,1/2*p*(p-1))

###############################################################
#F approximation:
# For F Approximation
C2 = (((p-1)*(p+2))/(6*(k-1)))*(sum(1/vi^2)-1/(sum(vi)^2))
a1 = 1/2*(k-1)*p*(p+1)
a2 = (a1 + 2)/abs(C2 - C1^2)
b1 = (1-C1-(a1/a2))/a1
b2 = (1-C1 + 2/a2)/a2
if(C2 > C1^2){
  F_stat = -2*b1*log(M)
}else{
  F_stat = -(2*a2*b2*log(M))/(a1*(1+2*b2*log(M)))
}
#F statistics
qf(1-alpha, a1, a2)
F_stat
# 4.558044 > 1.416674, we should reject H0, conlude QDA
# Define the repeated random subsampling function when
random_sampling <- function(data, split,n_iter, table = FALSE) {
  n <- nrow(data)
  trainsize <- round(split*n)
  accuracy <- numeric(n_iter)
  for(i in 1:n_iter) {
    traindata <- data[sample(1:n,round(trainsize)),]
    testdata <- data[-sample(1:n, trainsize), ]
    model <- qda(diabetes ~ ., data = traindata)
    p2 <- predict(model, testdata)$class
    tab1 <- table(Predicted = p2, Actual = testdata$diabetes)
    accuracy[i] <- sum(diag(tab1))/sum(tab1)
  }
  if (table == FALSE){
    return(sum(accuracy)/n_iter)
  }else{
    return(tab1)
  }
}


random_sampling(df,0.5,1000)
random_sampling(df,0.6,1000)
random_sampling(df,0.7,1000)
random_sampling(df,0.8,1000)
random_sampling(df,0.9,1000)
k_fold_cv <- function(data, k, n_iter) {
  n <- nrow(data)
  fold_size <- round(n/k)
  accuracy <- numeric(n_iter)

  for(i in 1:n_iter) {
    accuracy_fold <- numeric(k)

    for(j in 1:k){
      test_indices <- ((j-1) * fold_size + 1):(j * fold_size)
      testdata <- data[test_indices,]
      traindata <- data[-test_indices,]
      model <- qda(diabetes ~ ., data = traindata)
      p2 <- predict(model, testdata)$class
      tab1 <- table(Predicted = p2, Actual = testdata$diabetes)
      accuracy_fold[j] <- sum(diag(tab1))/sum(tab1)
    }

    accuracy[i] <- mean(accuracy_fold)
  }
  return(mean(accuracy))
}

k_fold_cv(df,5,1000)
# splite data in 70% and 30%
n <- nrow(df)
trainsize <- round(0.7*n)
traindata <- df[sample(1:n,round(trainsize)),]
testdata <- df[-sample(1:n, trainsize), ]


QDA_coeff <- function(df){
  # Parameters
  n0 <- sum(df$diabetes == "neg")
  n1 <- sum(df$diabetes == "pos")
  n = n1 + n0
  p=dim(df)[2]-1
  pi_hat0 = n0/n
  pi_hat1 = n1/n
  negdata <- df[df$diabetes == "neg", ][-dim(df)[2]]
  column_sums0 <- as.vector(colSums(negdata))
  posdata <- df[df$diabetes == "pos", ][-dim(df)[2]]
  column_sums1 <- as.vector(colSums(posdata))
  mu_hat0 = (1/n0)*column_sums0
  mu_hat1 = (1/n1)*column_sums1
  #estimator for the population covariance matrices
  S0 <- matrix(0,p,p)
  for(i in 1:n0){
    S0 <- S0 + (as.matrix(negdata)[i,]-mu_hat0)%*%t(as.matrix(negdata)[i,]-mu_hat0)
  }
  S0 <- S0/(n0-1)
  S1 <- matrix(0,p,p)
  for(i in 1:n1){
    S1 <- S1 + (as.matrix(posdata)[i,]-mu_hat1)%*%t(as.matrix(posdata)[i,]-mu_hat1)
  }
```

```r
  S1 <- S1/(n1-1)
  my_list <- list(S0,S1,mu_hat0, mu_hat1, pi_hat0, pi_hat1)
  return(my_list)
}

#parameters for train data:
S0 <- QDA_coeff(traindata)[[1]]
S1 <- QDA_coeff(traindata)[[2]]
mu_hat0 <- QDA_coeff(traindata)[[3]]
mu_hat1 <- QDA_coeff(traindata)[[4]]
pi_hat0 <- QDA_coeff(traindata)[[5]]
pi_hat1 <- QDA_coeff(traindata)[[6]]


# input X vector should be 1 by 8 vector
Desision_boundary <- function(X){
  QDA_DB = (1/2)*t(X)%*%(solve(S1)-solve(S0))%*%X + t(X)%*%(solve(S0)%*%mu_hat0 - solve(S1)%*%mu_hat1) 
  if ((QDA_DB) > 0){
    return("neg")
  }else if ((QDA_DB) < 0){
    return("pos")
  }
}

# Check the test prediction 30%
pred <- c() # Create an empty vector to store the predictions
for (i in 1:nrow(testdata)) {
  pred <- c(pred, Desision_boundary(as.numeric(testdata[i,][-9])))  # Append the prediction to the 'pre
}

pred_diabetes <- c(pred)
true_diabetes <- c(testdata$diabetes)
tab <- table(Predicted = pred_diabetes, Actual = testdata$diabetes)
accuracy <- sum(diag(tab))/sum(tab)
accuracy
QDA_Decision_Boundary <- function(df, X){
  # Parameters
  n0 <- sum(df$diabetes == "neg")
  n1 <- sum(df$diabetes == "pos")
  n = n1 + n0
  p=dim(df)[2]-1
  pi_hat0 = n0/n
  pi_hat1 = n1/n
  negdata <- df[df$diabetes == "neg", ][-dim(df)[2]]
  column_sums0 <- as.vector(colSums(negdata))
  posdata <- df[df$diabetes == "pos", ][-dim(df)[2]]
  column_sums1 <- as.vector(colSums(posdata))
  mu_hat0 = (1/n0)*column_sums0
  mu_hat1 = (1/n1)*column_sums1
  #estimator for the population covariance matrices
  S0 <- matrix(0,p,p)
  for(i in 1:n0){
    S0 <- S0 + (as.matrix(negdata)[i,]-mu_hat0)%*%t(as.matrix(negdata)[i,]-mu_hat0)
  }
  S0 <- S0/(n0-1)
  S1 <- matrix(0,p,p)
  for(i in 1:n1){
    S1 <- S1 + (as.matrix(posdata)[i,]-mu_hat1)%*%t(as.matrix(posdata)[i,]-mu_hat1)
  }
  S1 <- S1/(n1-1)

  QDA_DB = (1/2)*t(X)%*%(solve(S1)-solve(S0))%*%X + t(X)%*%(solve(S0)%*%mu_hat0 - solve(S1)%*%mu_hat1)
  if ((QDA_DB) > 0){
    return("neg")
  }else if ((QDA_DB) < 0){
    return("pos")
  }
}

# Define the repeated random subsampling function when
random_sampling_QDA <- function(data, split,n_iter) {
  n <- nrow(data)
  trainsize <- round(split*n)
  accuracy <- numeric(n_iter)
  for(i in 1:n_iter) {
    traindata <- data[sample(1:n,round(trainsize)),]
    testdata <- data[-sample(1:n, trainsize), ]
    #model <- qda(diabetes ~ ., data = traindata)
    pred <- c() # Create an empty vector to store the predictions
    for (j in 1:nrow(testdata)) {
      pred <- c(pred, QDA_Decision_Boundary(df = traindata, X = as.numeric(testdata[j,][-dim(df)[2]])))
    }
    pred_diabetes <- c(pred)
    true_diabetes <- c(testdata$diabetes)
    length(pred_diabetes)
    length(true_diabetes)
    tab <- table(Predicted = as.factor(pred_diabetes), Actual = as.factor(testdata$diabetes))
    accuracy[i] <- sum(diag(tab))/sum(tab)
  }
  acc = sum(accuracy)/n_iter
  return(acc)
}

#random_sampling_QDA(data = newpc5, split=0.7, n_iter=1)
# Define the repeated random subsampling function when
random_sampling_QDA <- function(data, split,n_iter, table = FALSE) {
  n <- nrow(data)
  trainsize <- round(split*n)
  accuracy <- numeric(n_iter)
  confusion_tables <- list()
  for(i in 1:n_iter) {
    traindata <- data[sample(1:n,round(trainsize)),]
    testdata <- data[-sample(1:n, trainsize), ]
    #model <- qda(diabetes ~ ., data = traindata)
    pred <- c() # Create an empty vector to store the predictions
    for (j in 1:nrow(testdata)) {
      pred <- c(pred, QDA_Decision_Boundary(df = traindata, X = as.numeric(testdata[j,][-9])))  # Appen
    }
    pred_diabetes <- c(pred)
    true_diabetes <- c(testdata$diabetes)
    length(pred_diabetes)
    length(true_diabetes)
    tab <- table(Predicted = as.factor(pred_diabetes), Actual = as.factor(testdata$diabetes))
    accuracy[i] <- sum(diag(tab))/sum(tab)
    confusion_tables[[i]] <- tab
  }
  # Calculate the average confusion table
  avg_table <- round(Reduce(`+`, confusion_tables) / n_iter )
  acc = sum(accuracy)/n_iter
  if (table == FALSE) {
    return(acc)
  } else {
    return(avg_table)
  }
}

print(random_sampling_QDA(data = df, split=0.7, n_iter=100))
print(random_sampling_QDA(data = df, split=0.7, n_iter=100, table=TRUE))
#PCA
# Select the variables
X <- df[1:8]
Y <- df[9]
table(Y)
ev=eigen((cov(X)))$values
eve = eigen((cov(X)))$vectors
S <- cor(X)
val<-eigen(S)$values
vec<-eigen(S)$vectors

#Method of Scree Plot: Choose PC3
plot(val,type="b",main="Scree Plot") +
abline(h=mean(val), col = "red")

#Method of Scree Plot of Difference
plot(-diff(val), type = "b", main = "Eigenvalue Differences Plot")

#method of CPP: Choose first PC4 or PC5
plot(cumsum(val)/8,type="b",main="Cumulative percentage Plot") +
abline(h=0.8, col = "red")

PCA<-princomp(X,cor=T)
loadings(PCA)
summary(PCA)
par(mfrow=c(2,2))
biplot(PCA,choices=c(1,3))
biplot(PCA,choices=c(3,3))
biplot(PCA,choices=c(1,2))
biplot(PCA,choices=c(3,2))
```

```r
########################
summary(PCA)
dim<-as.data.frame(eigen(S)$vector)
names(dim)[names(dim) == "V1"] <- "PC1"
names(dim)[names(dim) == "V2"] <- "PC2"
names(dim)[names(dim) == "V3"] <- "PC3"
names(dim)[names(dim) == "V4"] <- "PC4"
names(dim)[names(dim) == "V5"] <- "PC5"
names(dim)[names(dim) == "V6"] <- "PC6"
names(dim)[names(dim) == "V7"] <- "PC7"
names(dim)[names(dim) == "V8"] <- "PC8"
row.names(dim)[row.names(dim) == "1"] <- "pregnant"
row.names(dim)[row.names(dim) == "2"] <- "glucose"
row.names(dim)[row.names(dim) == "3"] <- "pressure"
row.names(dim)[row.names(dim) == "4"] <- "triceps"
row.names(dim)[row.names(dim) == "5"] <- "insulin"
row.names(dim)[row.names(dim) == "6"] <- "mass"
row.names(dim)[row.names(dim) == "7"] <- "pedigree"
row.names(dim)[row.names(dim) == "8"] <- "age"

corrplot(as.matrix(dim), order = "original",
         tl.col = "black", tl.srt = 50)

res.pca = PCA(X, scale.unit = TRUE, ncp = 2, quali.sup=c(1),graph = T)
my_palette <- c("neg" = "red", "pos" = "blue")
# Assuming your column name is "outcome" and it contains "neg" and "pos" values
Y <- ifelse(Y == "neg", 0, 1)
Y <- factor(Y)
# Create the PCA plot with colored points
fviz_pca(res.pca, geom.ind = "point", col.ind = Y, palette = my_palette,
         addEllipses = TRUE, legend.title = "Diabetes")
PCA_QDA_Acc <- function(df, n){
  X <- df[1:8]
  Y <- df[9]
  S <- cor(X)
  diabetes <- df[,9]
  PC_matrix=eigen(S)$vectors
  PC=PC_matrix[,1:n]
  df_matrix=df
  colnames(df_matrix) <- NULL
  df_matrix= as.matrix(df_matrix[-9])
  new_data=df_matrix %*% PC
  newpc=as.data.frame(new_data)
  newpc["diabetes"] <- diabetes
  return (random_sampling(newpc,0.7,1000))
}


PCA_data <- function(df, n){
  X <- df[1:8]
  Y <- df[9]
  S <- cor(X)
  diabetes <- df[,9]
  PC_matrix=eigen(S)$vectors
  PC=PC_matrix[,1:n]
  df_matrix=df
  colnames(df_matrix) <- NULL
  df_matrix= as.matrix(df_matrix[-9])
  new_data=df_matrix %*% PC
  newpc=as.data.frame(new_data)
  newpc["diabetes"] <- diabetes
  return (newpc)
}


accuracy_list <- c()
for (i in 1:8){
  accuracy <- PCA_QDA_Acc(df, i)
  accuracy_list <- c(accuracy_list, accuracy)
}

plot(accuracy_list, type = "b",xlab = "Principal Components",ylab = "Accuracy", main = "Accuracy of Eac
accuracy_list
PCA_data <- function(df, n){
  X <- df[1:8]
  Y <- df[9]
  S <- cor(X)
  diabetes <- df[,9]
  PC_matrix=eigen(S)$vectors
  PC=PC_matrix[,1:n]
  df_matrix=df
  colnames(df_matrix) <- NULL
  df_matrix= as.matrix(df_matrix[-9])
  new_data=df_matrix %*% PC
  newpc=as.data.frame(new_data)
  newpc["diabetes"] <- diabetes
  return (newpc)
}
newpc3 <- PCA_data(df, 3)
newpc4 <- PCA_data(df, 4)
newpc5 <- PCA_data(df, 5)
# Method 1: Package
res <- boxM(newpc3[, 1:3], newpc3[, "diabetes"])
res
########################################

# Method 2: Manually: Chi-sq and F-approx
#df = PimaIndiansDiabetes
X = newpc3[,1:3]
Y <- as.factor(newpc3$diabetes)
k <- length(levels(Y))
table(newpc3$diabetes)
n1 <- sum(Y == levels(Y)[1])
n2 <- sum(Y == levels(Y)[2])
p <- dim(newpc3)[2] - 1
v1 <- n1 - 1
```

```
v2 <- n2 - 1
X_neg_df = newpc3[newpc3$diabetes == "neg",1:3]
X_pos_df = newpc3[newpc3$diabetes == "pos",1:3]
S1 <- cov(X_neg_df) #neg
S2 <- cov(X_pos_df) #pos
Spl <- (v1*S1+v2*S2)/(v1+v2)
vi = c(v1, v2)
C1 = (sum(1/vi)-1/sum(vi))*((2*p^2+3*p-1)/(6*(p+1)*(k-1)))
lnM = 0.5*(v1*log(det(S1))+v2*log(det(S2)))-0.5*(v1+v2)*log(det(Spl))
M = exp(0.5*(v1*log(det(S1))+v2*log(det(S2)))-0.5*(v1+v2)*log(det(Spl)))
# chi-sq approximation
u = -2*(1-C1)*(lnM)
u # Result: 30.5138 > 12.59159 so we reject H0, conlude QDA
# chi-sq statistics
alpha = 0.05
qchisq(1-alpha,1/2*p*(p-1))
#################################################################
#F approximation:
# For F Approximation
C2 = (((p-1)*(p+2))/(6*(k-1)))*(sum(1/vi^2)-1/(sum(vi)^2))
a1 = 1/2*(k-1)*p*(p+1)
a2 = (a1 + 2)/abs(C2 - C1^2)
b1 = (1-C1-(a1/a2))/a1
b2 = (1-C1 + 2/a2)/a2
if(C2 > C1^2){
  F_stat = -2*b1*log(M)
}else{
  F_stat = -(2*a2*b2*log(M))/(a1*(1+2*b2*log(M)))
}
#F statistics
qf(1-alpha, a1, a2)
F_stat
# 3.051283 > 1.830733, we should reject H0, conlude QDA
random_sampling(newpc3,0.5,100)
random_sampling(newpc3,0.6,100)
random_sampling(newpc3,0.7,100)
random_sampling(newpc3,0.8,100)
random_sampling(newpc3,0.9,100)
# Method 1: Package
res <- boxM(newpc4[, 1:4], newpc4[, "diabetes"])
res
summary(res)
########################################

# Method 2: Manually: Chi-sq and F-approx
#df = PimaIndiansDiabetes
X = newpc4[,1:4]
Y <- as.factor(newpc4$diabetes)
k <- length(levels(Y))

table(newpc4$diabetes)
n1 <- sum(Y == levels(Y)[1])
n2 <- sum(Y == levels(Y)[2])
```

```r
p <- dim(newpc4)[2] - 1

v1 <- n1 - 1
v2 <- n2 - 1

X_neg_df = newpc4[newpc4$diabetes == "neg",1:4]
X_pos_df = newpc4[newpc4$diabetes == "pos",1:4]
S1 <- cov(X_neg_df) #neg
S2 <- cov(X_pos_df) #pos
det(S1)
det(S2)

Spl <- (v1*S1+v2*S2)/(v1+v2)
det(Spl)

vi = c(v1, v2)
C1 = (sum(1/vi)-1/sum(vi))*((2*p^2+3*p-1)/(6*(p+1)*(k-1)))

lnM = 0.5*(v1*log(det(S1))+v2*log(det(S2)))-0.5*(v1+v2)*log(det(Spl))
M = exp(0.5*(v1*log(det(S1))+v2*log(det(S2)))-0.5*(v1+v2)*log(det(Spl)))

# chi-sq approximation
u = -2*(1-C1)*(lnM)
u # Result: 30.5138 > 12.59159 so we reject H0, conlude QDA
# chi-sq statistics
alpha = 0.05
qchisq(1-alpha,1/2*p*(p-1))


##################################################################
#F approximation:
# For F Approximation
C2 = (((p-1)*(p+2))/(6*(k-1)))*(sum(1/vi^2)-1/(sum(vi)^2))
C2

a1 = 1/2*(k-1)*p*(p+1)
a2 = (a1 + 2)/abs(C2 - C1^2)

b1 = (1-C1-(a1/a2))/a1
b2 = (1-C1 + 2/a2)/a2

if(C2 > C1^2){
  F_stat = -2*b1*log(M)
}else{
  F_stat = -(2*a2*b2*log(M))/(a1*(1+2*b2*log(M)))
}
F_stat

#F statistics
qf(1-alpha, a1, a2)
F_stat
# 3.051283 > 1.830733, we should reject H0, conlude QDA
random_sampling(newpc4,0.5,1000)
random_sampling(newpc4,0.6,1000)
random_sampling(newpc4,0.7,1000)
random_sampling(newpc4,0.8,1000)
random_sampling(newpc4,0.9,1000)
# Method 1: Package
res <- boxM(newpc5[, 1:5], newpc5[, "diabetes"])
res
summary(res)
########################################

# Method 2: Manually: Chi-sq and F-approx
#df = PimaIndiansDiabetes
X = newpc5[,1:5]
Y <- as.factor(newpc5$diabetes)
k <- length(levels(Y))

table(newpc5$diabetes)
n1 <- sum(Y == levels(Y)[1])
n2 <- sum(Y == levels(Y)[2])
p <- dim(newpc5)[2] - 1

v1 <- n1 - 1
v2 <- n2 - 1

X_neg_df = newpc5[newpc5$diabetes == "neg",1:5]
X_pos_df = newpc5[newpc5$diabetes == "pos",1:5]
S1 <- cov(X_neg_df) #neg
S2 <- cov(X_pos_df) #pos
det(S1)
det(S2)

Spl <- (v1*S1+v2*S2)/(v1+v2)
det(Spl)

vi = c(v1, v2)
C1 = (sum(1/vi)-1/sum(vi))*((2*p^2+3*p-1)/(6*(p+1)*(k-1)))

lnM = 0.5*(v1*log(det(S1))+v2*log(det(S2)))-0.5*(v1+v2)*log(det(Spl))
M = exp(0.5*(v1*log(det(S1))+v2*log(det(S2)))-0.5*(v1+v2)*log(det(Spl)))

# chi-sq approximation
u = -2*(1-C1)*(lnM)
u # Result: 84.323 > 18.30704 so we reject H0, conlude QDA
# chi-sq statistics
alpha = 0.05
qchisq(1-alpha,1/2*p*(p-1))


################################################################
#F approximation:
# For F Approximation
C2 = (((p-1)*(p+2))/(6*(k-1)))*(sum(1/vi^2)-1/(sum(vi)^2))
C2
```

```
a1 = 1/2*(k-1)*p*(p+1)
a2 = (a1 + 2)/abs(C2 - C1^2)

b1 = (1-C1-(a1/a2))/a1
b2 = (1-C1 + 2/a2)/a2

if(C2 > C1^2){
  F_stat = -2*b1*log(M)
}else{
  F_stat = -(2*a2*b2*log(M))/(a1*(1+2*b2*log(M)))
}
F_stat

#F statistics
qf(1-alpha, a1, a2)
F_stat
# 5.621194 > 1.666422, we should reject H0, conlude QDA
random_sampling(newpc5,0.5,100)
random_sampling(newpc5,0.6,100)
random_sampling(newpc5,0.7,100)
random_sampling(newpc5,0.8,100)
random_sampling(newpc5,0.9,100)
random_sampling <- function(data, split, n_iter, table = FALSE) {
  n <- nrow(data)
  trainsize <- round(split * n)
  accuracy <- numeric(n_iter)
  confusion_tables <- list()  # Initialize a list to store confusion tables

  for (i in 1:n_iter) {
    traindata <- data[sample(1:n, round(trainsize)), ]
    testdata <- data[-sample(1:n, trainsize), ]
    model <- qda(diabetes ~ ., data = traindata)
    p2 <- predict(model, testdata)$class
    tab1 <- table(Predicted = p2, Actual = testdata$diabetes)
    accuracy[i] <- sum(diag(tab1))/sum(tab1)
    confusion_tables[[i]] <- tab1  # Store each confusion table in the list
  }

  # Calculate the average confusion table
  avg_table <- round(Reduce(`+`, confusion_tables) / n_iter )

  if (table == FALSE) {
    return(sum(accuracy) / n_iter)
  } else {
    return(avg_table)
  }
}

print(random_sampling(newpc3,0.7,1000, table = TRUE))
print(random_sampling(newpc3,0.7,1000, table = FALSE))

print(random_sampling(newpc4,0.7,1000, table = TRUE))
print(random_sampling(newpc4,0.7,1000, table = FALSE))
```

```r
print(random_sampling(newpc5,0.7,1000, table = TRUE))
print(random_sampling(newpc5,0.7,1000, table = FALSE))
```