Prostate Cancer Project

G. Capurso, D. Di Labbio, A. Garbin

2023-07-12

Contents

Introduction

Having medical images of tumors and access to medical data in general can be immensely beneficial in medicine, particularly when combined with statistical analysis and machine learning techniques. The availability of medical images of tumors, along with comprehensive medical data, presents a remarkable opportunity for improving healthcare through the application of statistics and machine learning. By leveraging these advanced technologies, medical professionals can extract valuable insights, enhance diagnostic accuracy, and devise personalized treatment plans. In particular, statistical analysis enables the identification of patterns, correlations, and trends within large datasets, empowering researchers and clinicians to derive meaningful conclusions. When applied to medical images, statistical methods can help uncover subtle patterns and features that might be overlooked by the human eye alone. By quantifying and analyzing these image characteristics, statistical techniques aid in early tumor detection, accurate classification, and monitoring of treatment response.

Problem presentation

In this project we present a binary classification problem based on the "Prostate Cancer Dataset", freely available on Kaggle.

Our objective was to study a group of patients and determine whether a tumor is benign or malignant based on specific characteristics observed during each patient's diagnosis. To achieve this, we aimed to identify the most informative features and evaluate their effectiveness in various classification models for distinguishing between the two tumor types.

The dataset constists of 10 features and 100 instances corresponding to the patients that have been analized.

The set of features is composed by two informative features, *ID* and *diagnosis_results*, corresponding respectively to the ID of the patient and to the type of tumor ("Benignant" or "Malignant"), and eight numerical features, divided as follows:

- *radius* : mean of distances from center to points on the perimeter
- *texture* : standard deviation of gray-scale values
- *perimeter*
- *area*
- *smoothness* : local variation in radius lengths (by measuring the difference between the length of a radial line and the mean lenght of the lines surrounding it)
- *compactness* : perimeterˆ2 / area - 1.0

- *symmetry*
- *fractal_dimension* : "coastline approximation" - 1

Data exploration

After downloading the dataset locally, we import it on R and check at first if there is any missing values. We then check if any of the row is duplicated. Since each row is unique, we get rid of the first column, referring to the ID of the patients, to have a dataset with only relevant informations.

```
library(readr)
prostate_cancer <- read.csv("Prostate_Cancer.csv")
View(prostate_cancer)

# Check NA
anyNA(prostate_cancer)
```

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

```
prostate_cancer$id <- as.character(prostate_cancer$id)
# Check duplicate in ID
sum(duplicated(prostate_cancer$id) == TRUE)
```

```
## [1] 0
```

```
# No duplicates => we can remove id column
prostate_cancer <- prostate_cancer[,-1]
```

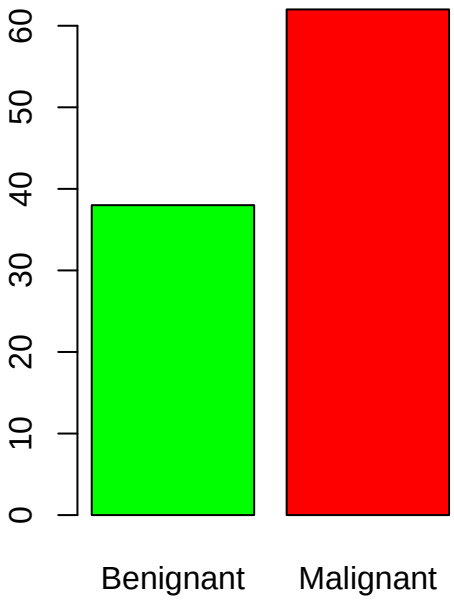
We re-organize the *diagnosis_result* column with two levels named "Benignant" and "Malignant" (corresponding to the original values "B" and "M" respectively) and check the distribution of the two classes.

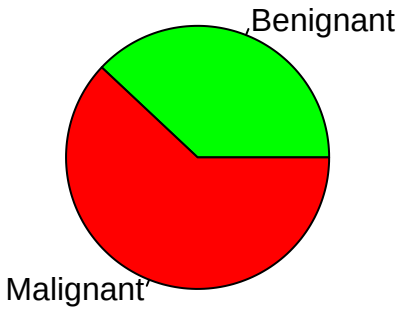
```
prostate_cancer$diagnosis_result <- as.factor(prostate_cancer$diagnosis_result)
levels(prostate_cancer$diagnosis_result) <- c('Benignant', 'Malignant')

# Proportion between Benignant and Malignant data
table(prostate_cancer$diagnosis_result)/length(prostate_cancer$diagnosis_result)
```

```
##
## Benignant Malignant
##      0.38      0.62
```

To see the distribution of the classes in a more intuitive way, we do some plots.

```
par(mfrow = c(1,2))

l <- prostate_cancer$diagnosis_result =="Benignant"
plot(prostate_cancer$diagnosis_result, col = c("green", "red")[l+1], pch=l*15+1)

class_counts <- table(prostate_cancer$diagnosis_result)
class_percentages <- prop.table(class_counts) * 100

par(mar = c(5, 5, 2, 2))
labels <- names(class_counts)
pie(class_percentages, labels = labels, col=c("green","red"))
```





We produce a summary of our dataset to see how the different values of every features were distributed, and then we calculate the variance and the standard deviation of each variables using the functions var() and sd().

```
summary(prostate_cancer)
```

```
##   diagnosis_result     radius         texture         perimeter
##  Benignant:38      Min.   : 9.00   Min.   :11.00   Min.   : 52.00
##  Malignant:62      1st Qu.:12.00   1st Qu.:14.00   1st Qu.: 82.50
##                    Median :17.00   Median :17.50   Median : 94.00
##                    Mean   :16.85   Mean   :18.23   Mean   : 96.78
##                    3rd Qu.:21.00   3rd Qu.:22.25   3rd Qu.:114.25
##                    Max.   :25.00   Max.   :27.00   Max.   :172.00
##       area          smoothness      compactness        symmetry
##  Min.   : 202.0   Min.   :0.0700   Min.   :0.0380   Min.   :0.1350
##  1st Qu.: 476.8   1st Qu.:0.0935   1st Qu.:0.0805   1st Qu.:0.1720
##  Median : 644.0   Median :0.1020   Median :0.1185   Median :0.1900
##  Mean   : 702.9   Mean   :0.1027   Mean   :0.1267   Mean   :0.1932
##  3rd Qu.: 917.0   3rd Qu.:0.1120   3rd Qu.:0.1570   3rd Qu.:0.2090
##  Max.   :1878.0   Max.   :0.1430   Max.   :0.3450   Max.   :0.3040
##  fractal_dimension
##  Min.   :0.05300
##  1st Qu.:0.05900
##  Median :0.06300
##  Mean   :0.06469
##  3rd Qu.:0.06900
##  Max.   :0.09700
```

```
#Variance and Standard Deviation

var(prostate_cancer$radius)
```

```
## [1] 23.80556
```

```
sd(prostate_cancer$radius)
```

```
## [1] 4.879094
```

```
var(prostate_cancer$texture)
```

```
## [1] 26.96677
```

```
sd(prostate_cancer$texture)
```

```
## [1] 5.192954
```

```
var(prostate_cancer$perimeter)
```

```
## [1] 560.5572
```

```
sd(prostate_cancer$perimeter)
```

```
## [1] 23.67609
```

```
var(prostate_cancer$area)
```

```
## [1] 102215.1
```

```
sd(prostate_cancer$area)
```

```
## [1] 319.7109
```

```
var(prostate_cancer$smoothness)
```

```
## [1] 0.0002143809
```

```
sd(prostate_cancer$smoothness)
```

```
## [1] 0.01464175
```

```
var(prostate_cancer$compactness)
```

```
## [1] 0.003738535
```

```
sd(prostate_cancer$compactness)
```

```
## [1] 0.06114356
```

```
var(prostate_cancer$symmetry)
```

```
## [1] 0.0009477183
```

```
sd(prostate_cancer$symmetry)
```

```
## [1] 0.03078503
```

```
var(prostate_cancer$fractal_dimension)
```

```
## [1] 6.643828e-05
```

```
sd(prostate_cancer$fractal_dimension)
```

```
## [1] 0.008150968
```

Density plots, histograms and Normal Q-Q plots

We now concentrate on numerical variables. After a first look of the Normal Q-Q plot of our variables, we decide to rescale them logaritmically, because some of them didn't present a normal behaviour.

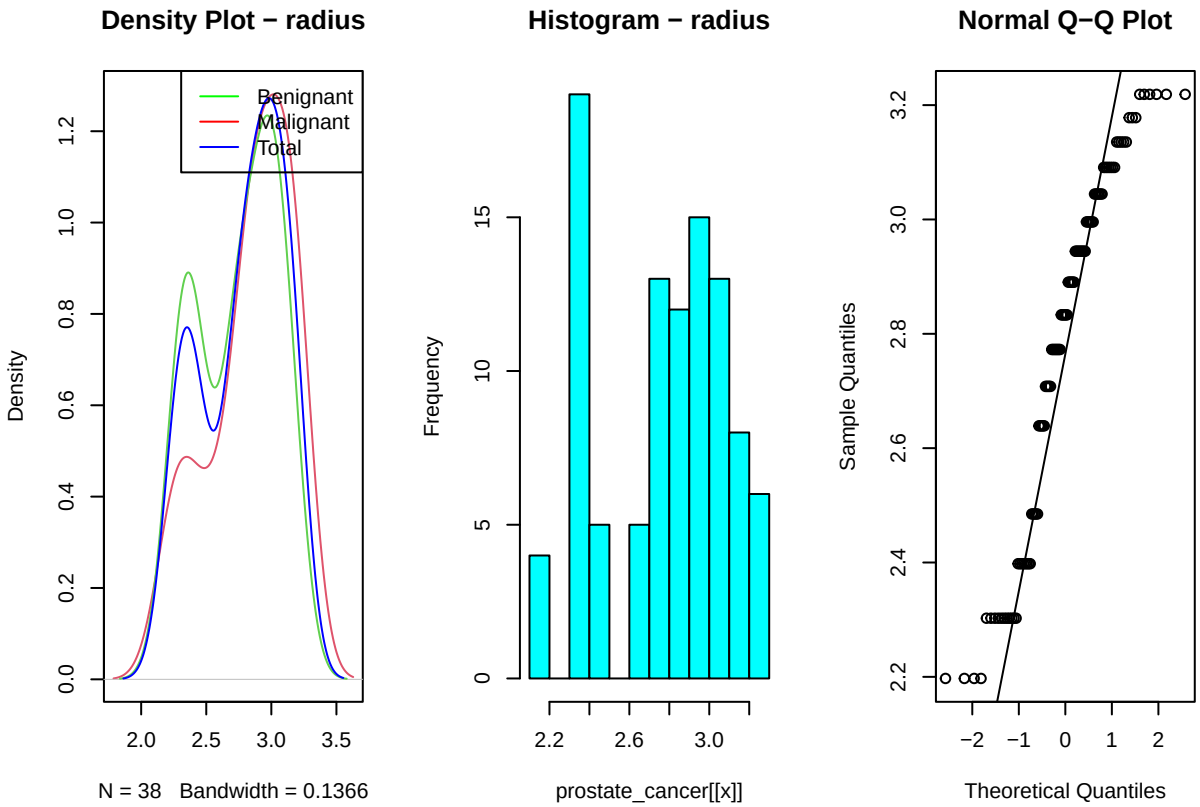
We are now ready to run the code with all the significant plots we are looking for: density plot (with respect to the total data, the data referred to the "Benignant" diagnosis and the ones referred to the "Malignant" diagnosis), histogram and normal Q-Q plot.

```
# Numeric columns log-scaled
num_col <- sapply(prostate_cancer, is.numeric)
prostate_cancer[num_col] <- lapply(prostate_cancer[num_col], log)

# Preparing data for density plots
cancer_benign <- prostate_cancer[prostate_cancer$diagnosis_result == "Benignant",]
cancer_malign <- prostate_cancer[prostate_cancer$diagnosis_result == "Malignant",]

# Plots for every numeric features
for (x in colnames(prostate_cancer)) {
  print(x)
  if (is.numeric(prostate_cancer[[x]])==TRUE){
    par(mfrow=c(1,3))
    plot(density(cancer_benign[[x]]), col = 2, main = paste("Density Plot -", x), type = 'l')
    lines(density(cancer_malign[[x]]), col = 3)
    lines(density(prostate_cancer[[x]]), col = "blue")
    legend("topright", legend = c("Benignant","Malignant","Total"),
           col = c("green", "red","blue"), lwd = 1)
    print(hist(prostate_cancer[[x]], col="cyan",
               main = paste("Histogram -", x)))
    qqnorm(prostate_cancer[[x]])
    qqline(prostate_cancer[[x]])
  }
}
```

```
## [1] "diagnosis_result"
## [1] "radius"
```

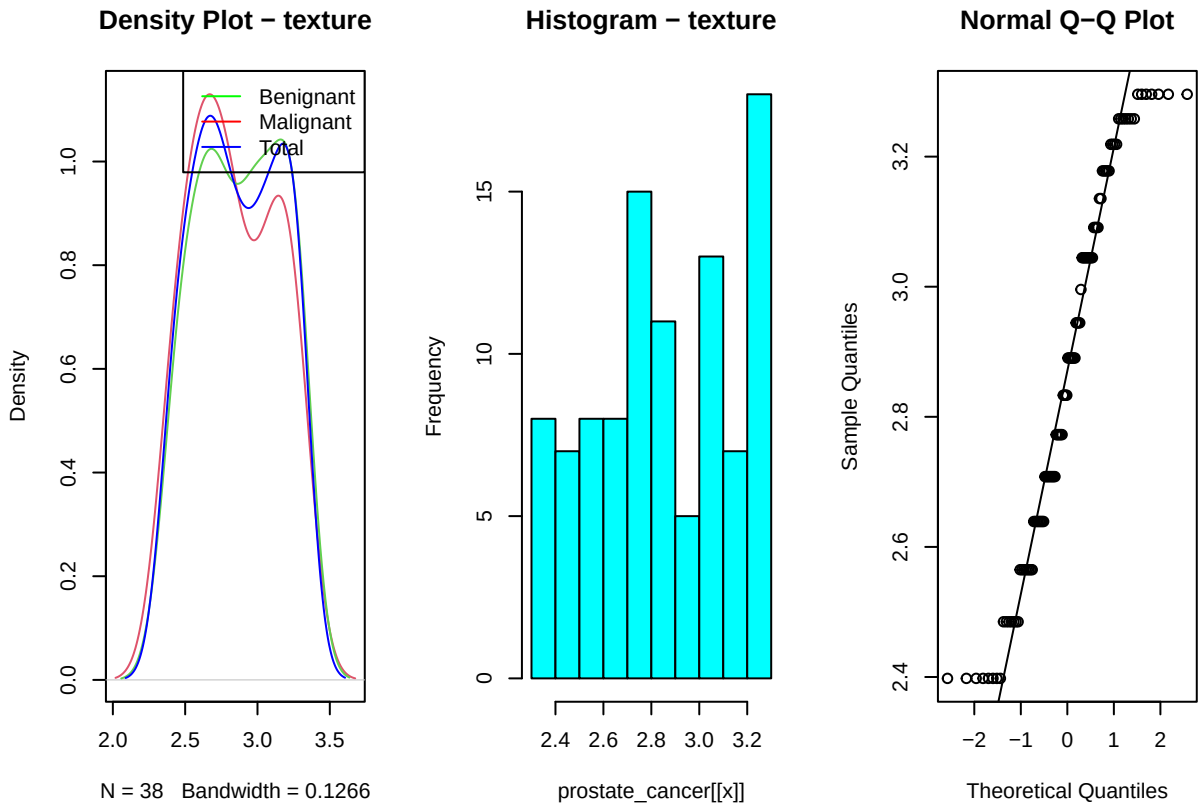
```
## $breaks
##  [1] 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3
##
## $counts
##  [1]  4  0 19  5  0  5 13 12 15 13  8  6
##
## $density
##  [1] 0.4 0.0 1.9 0.5 0.0 0.5 1.3 1.2 1.5 1.3 0.8 0.6
##
## $mids
##  [1] 2.15 2.25 2.35 2.45 2.55 2.65 2.75 2.85 2.95 3.05 3.15 3.25
##
## $xname
## [1] "prostate_cancer[[x]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
```

```
## [1] "histogram"
```



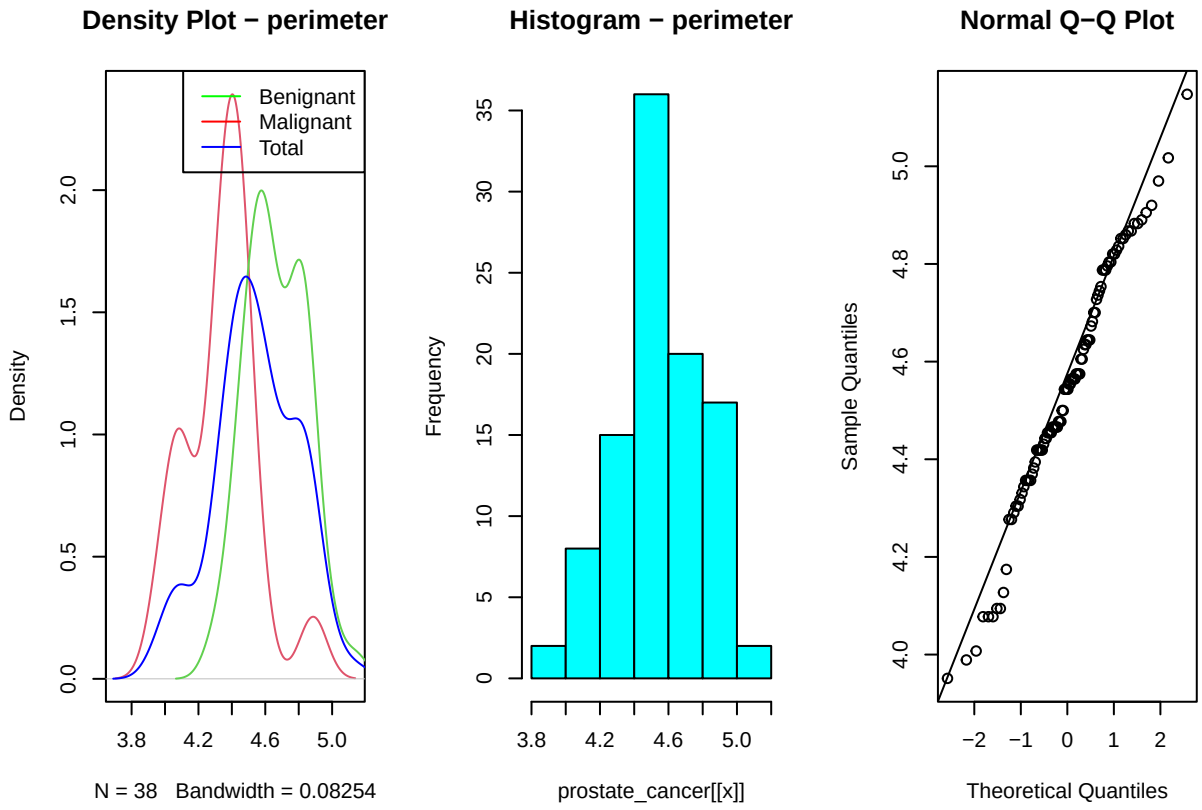

```
## [1] "texture"
```

```
## $breaks
##  [1] 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3
##
## $counts
##  [1]  8  7  8  8 15 11  5 13  7 18
##
## $density
##  [1] 0.8 0.7 0.8 0.8 1.5 1.1 0.5 1.3 0.7 1.8
##
## $mids
##  [1] 2.35 2.45 2.55 2.65 2.75 2.85 2.95 3.05 3.15 3.25
##
## $xname
## [1] "prostate_cancer[[x]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```



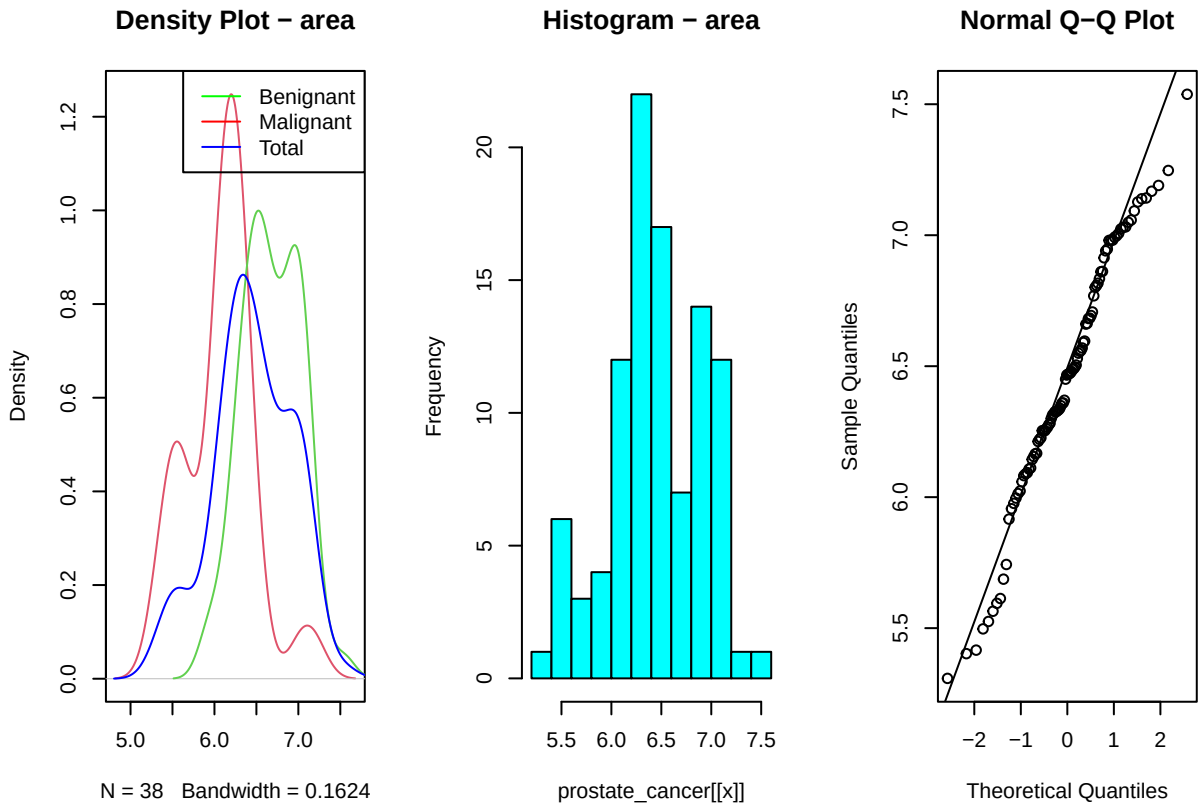

```
## [1] "perimeter"
```

```
## $breaks
## [1] 3.8 4.0 4.2 4.4 4.6 4.8 5.0 5.2
##
## $counts
## [1]  2  8 15 36 20 17  2
##
## $density
## [1] 0.10 0.40 0.75 1.80 1.00 0.85 0.10
##
## $mids
## [1] 3.9 4.1 4.3 4.5 4.7 4.9 5.1
##
## $xname
## [1] "prostate_cancer[[x]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```



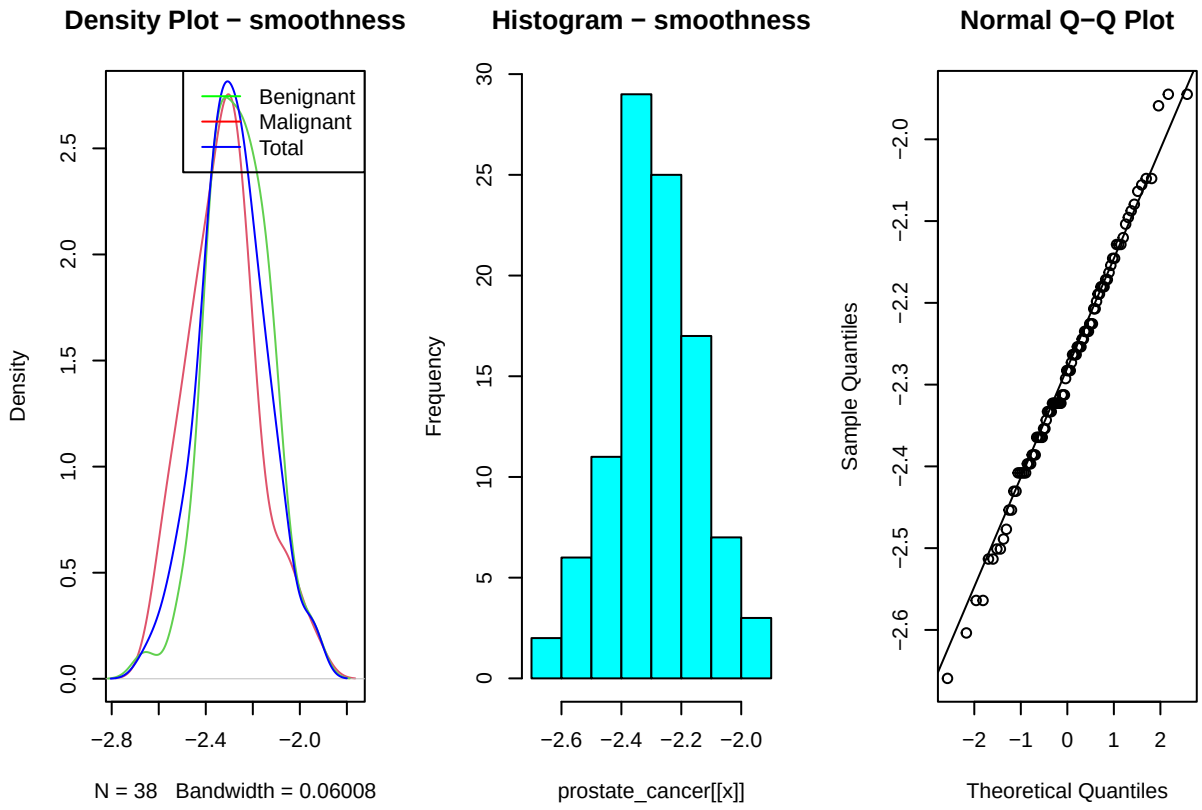

```
## [1] "area"
```

```
## $breaks
##  [1] 5.2 5.4 5.6 5.8 6.0 6.2 6.4 6.6 6.8 7.0 7.2 7.4 7.6
##
## $counts
##  [1]  1  6  3  4 12 22 17  7 14 12  1  1
##
## $density
##  [1] 0.05 0.30 0.15 0.20 0.60 1.10 0.85 0.35 0.70 0.60 0.05 0.05
##
## $mids
##  [1] 5.3 5.5 5.7 5.9 6.1 6.3 6.5 6.7 6.9 7.1 7.3 7.5
##
## $xname
## [1] "prostate_cancer[[x]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```



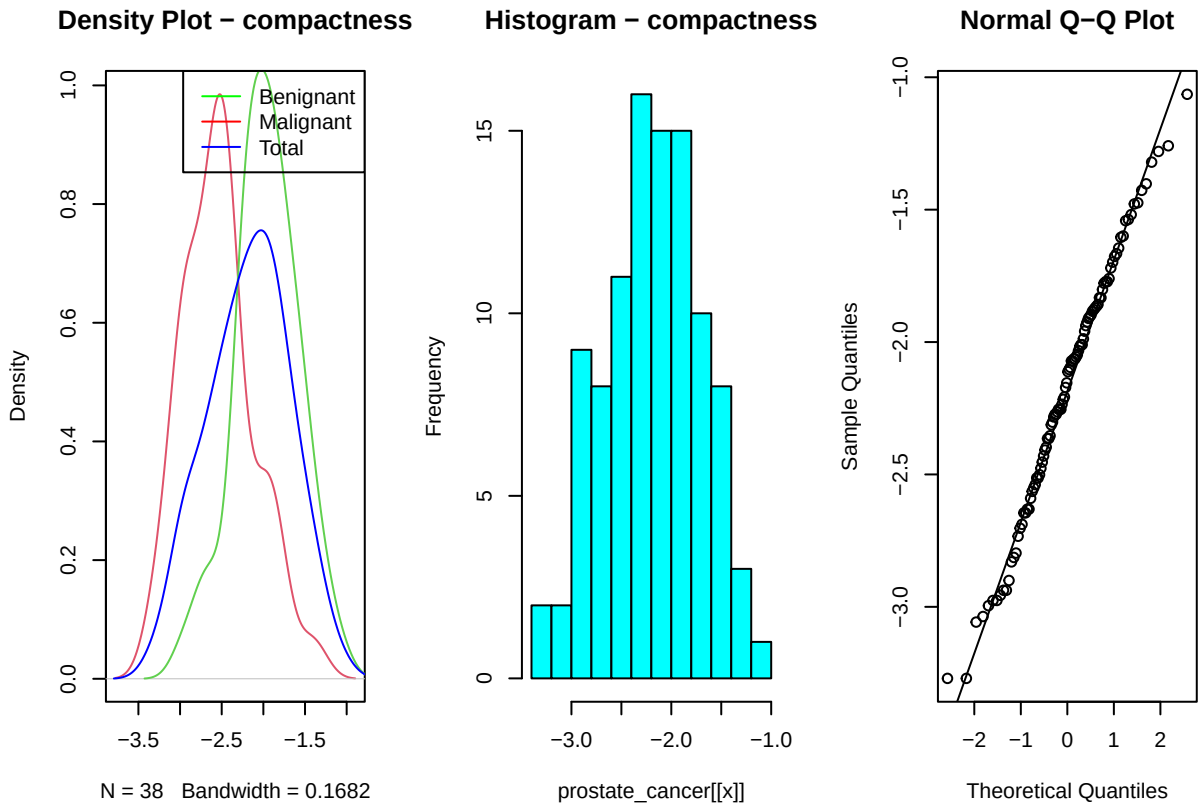

```
## [1] "smoothness"
```

```
## $breaks
## [1] -2.7 -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2.0 -1.9
##
## $counts
## [1]  2  6 11 29 25 17  7  3
##
## $density
## [1] 0.2 0.6 1.1 2.9 2.5 1.7 0.7 0.3
##
## $mids
## [1] -2.65 -2.55 -2.45 -2.35 -2.25 -2.15 -2.05 -1.95
##
## $xname
## [1] "prostate_cancer[[x]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```



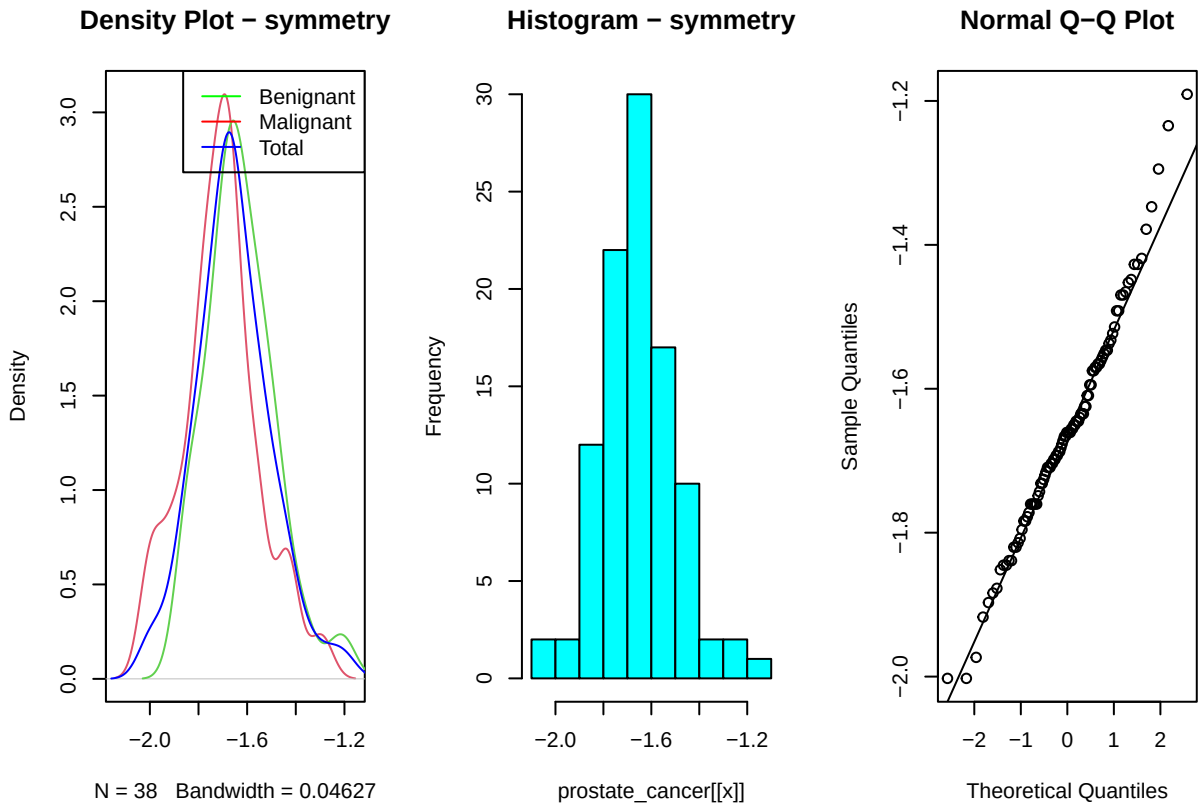

```
## [1] "compactness"
```

```
## $breaks
##  [1] -3.4 -3.2 -3.0 -2.8 -2.6 -2.4 -2.2 -2.0 -1.8 -1.6 -1.4 -1.2 -1.0
##
## $counts
##  [1]  2  2  9  8 11 16 15 15 10  8  3  1
##
## $density
##  [1] 0.10 0.10 0.45 0.40 0.55 0.80 0.75 0.75 0.50 0.40 0.15 0.05
##
## $mids
##  [1] -3.3 -3.1 -2.9 -2.7 -2.5 -2.3 -2.1 -1.9 -1.7 -1.5 -1.3 -1.1
##
## $xname
## [1] "prostate_cancer[[x]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```



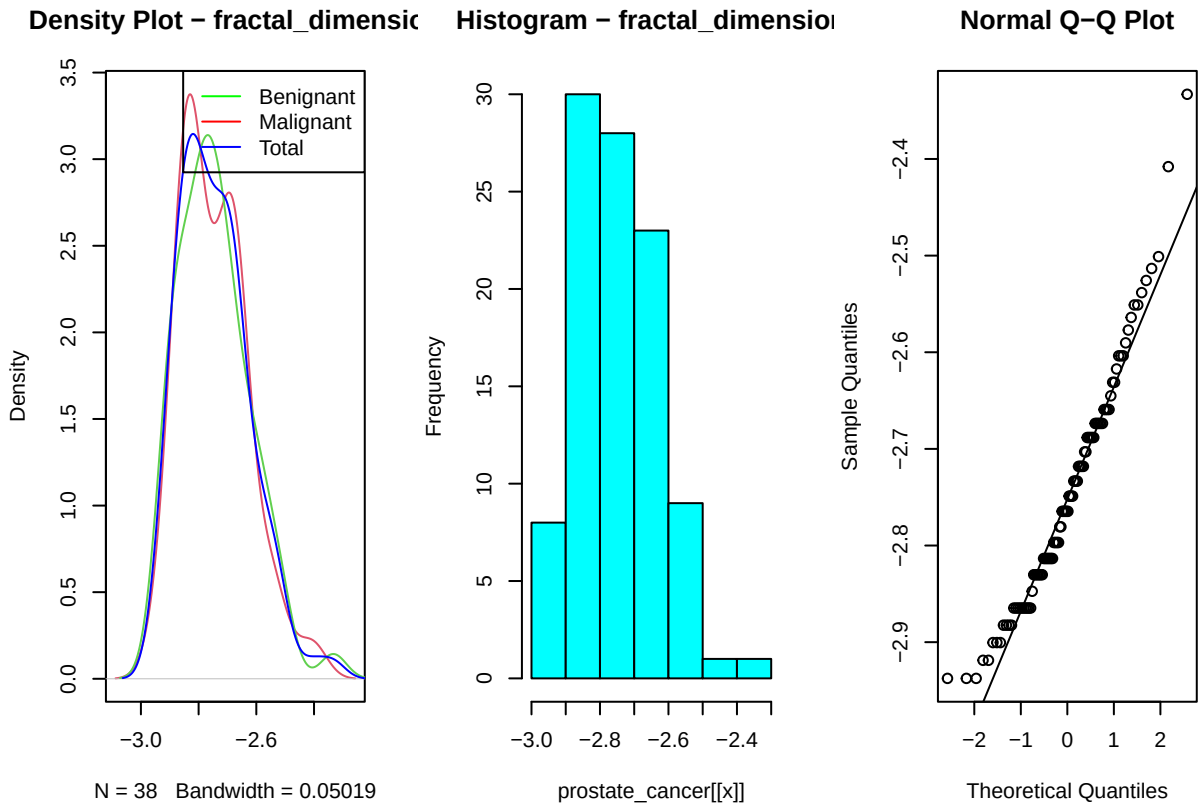

```
## [1] "symmetry"
```

```
## $breaks
##  [1] -2.1 -2.0 -1.9 -1.8 -1.7 -1.6 -1.5 -1.4 -1.3 -1.2 -1.1
##
## $counts
##  [1]  2  2 12 22 30 17 10  2  2  1
##
## $density
##  [1] 0.2 0.2 1.2 2.2 3.0 1.7 1.0 0.2 0.2 0.1
##
## $mids
##  [1] -2.05 -1.95 -1.85 -1.75 -1.65 -1.55 -1.45 -1.35 -1.25 -1.15
##
## $xname
## [1] "prostate_cancer[[x]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```




```
## [1] "fractal_dimension"
```

```
## $breaks
## [1] -3.0 -2.9 -2.8 -2.7 -2.6 -2.5 -2.4 -2.3
##
## $counts
## [1]  8 30 28 23  9  1  1
##
## $density
## [1] 0.8 3.0 2.8 2.3 0.9 0.1 0.1
##
## $mids
## [1] -2.95 -2.85 -2.75 -2.65 -2.55 -2.45 -2.35
##
## $xname
## [1] "prostate_cancer[[x]]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

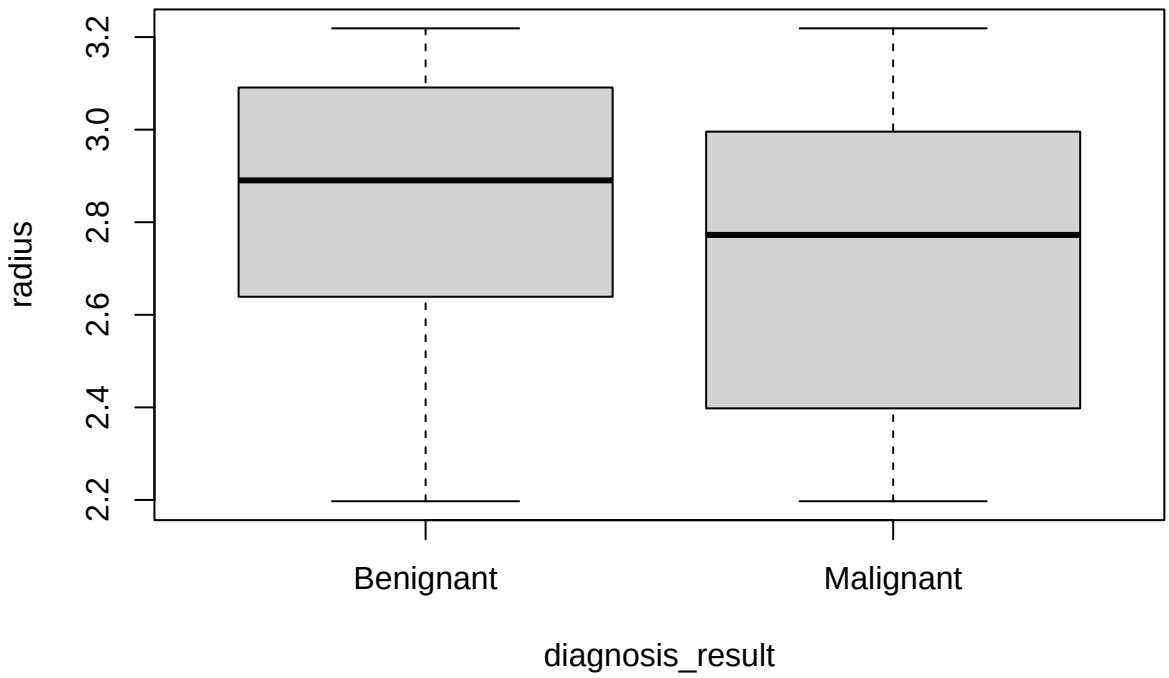


We notice that, even after having applied the logaritmic scale to the variables, "radius" and "texture" ones remain not strongly normal distributed.

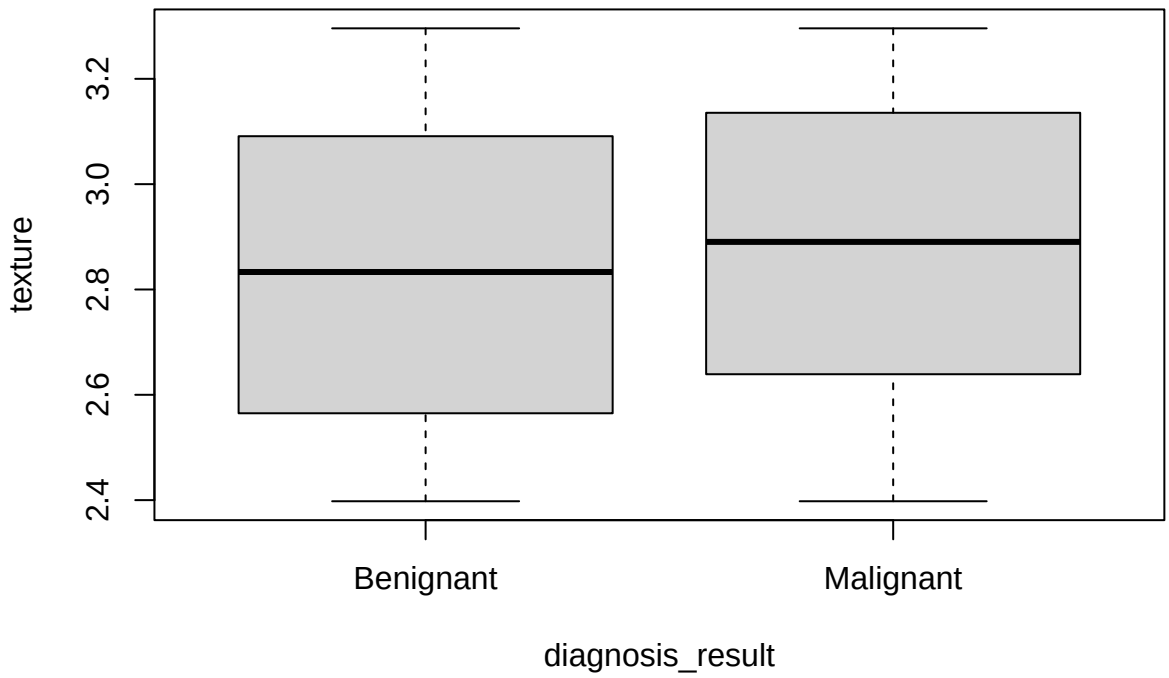
Boxplots

Finally, to have a concise summary of the distribution of our variables, we do some boxplots for a quick understanding of the data's central tendencies and spread.

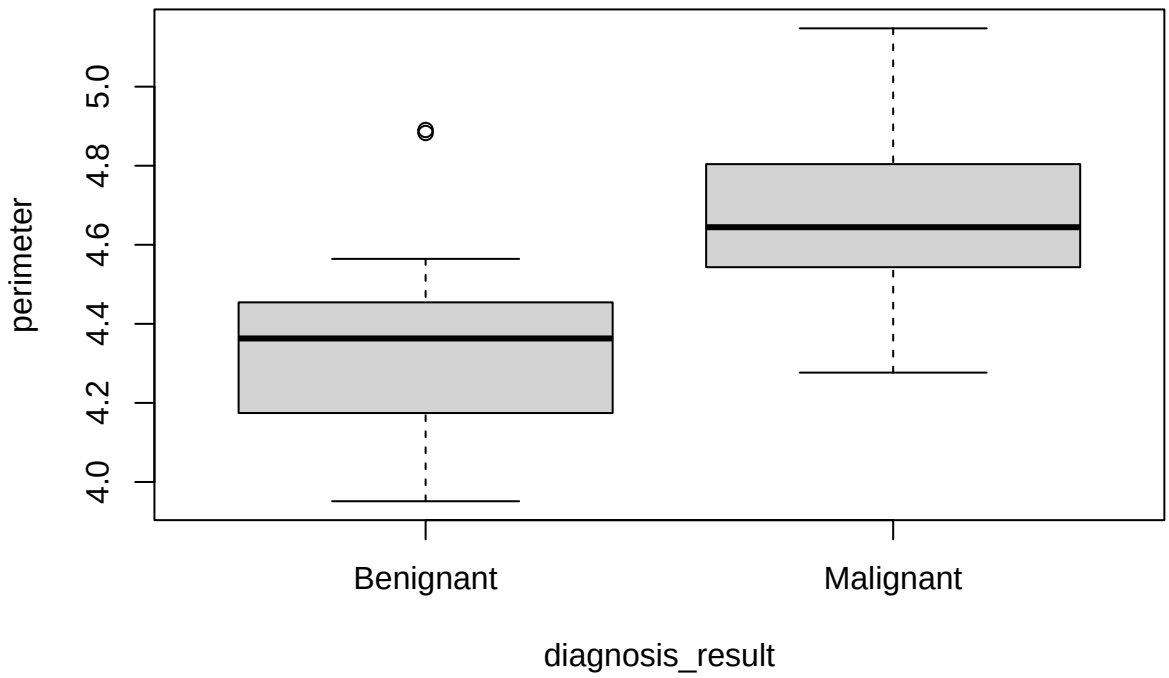
```
boxplot(radius ~ diagnosis_result, data = prostate_cancer)
```



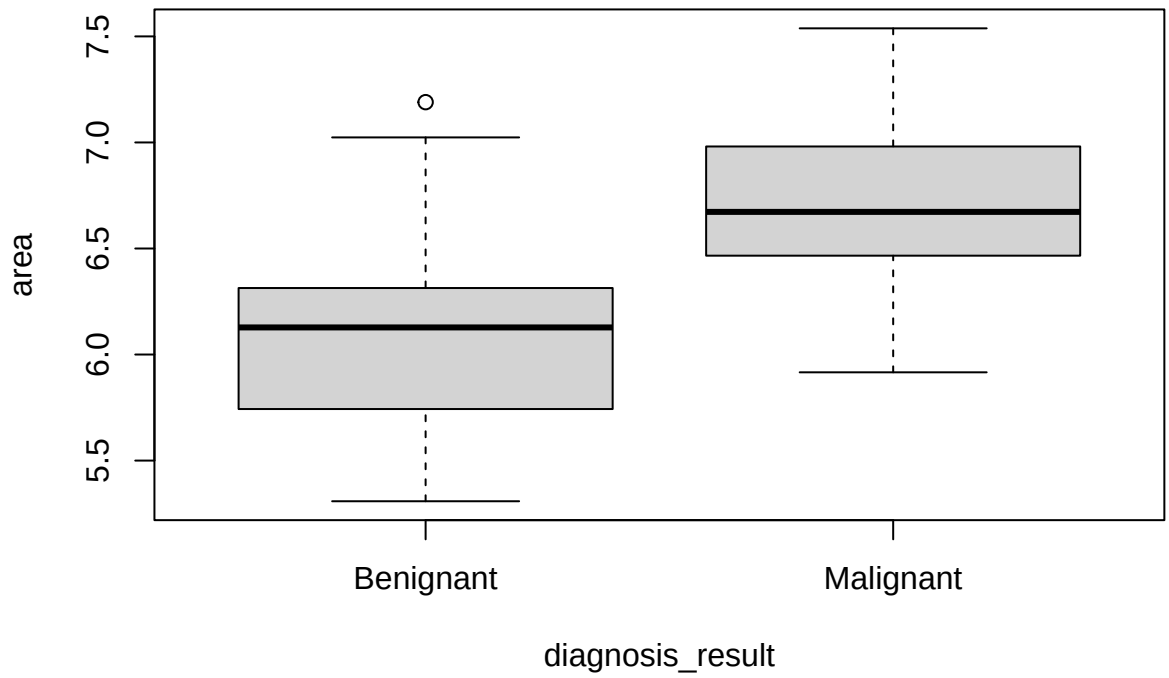

```
boxplot(texture ~ diagnosis_result, data = prostate_cancer)
```



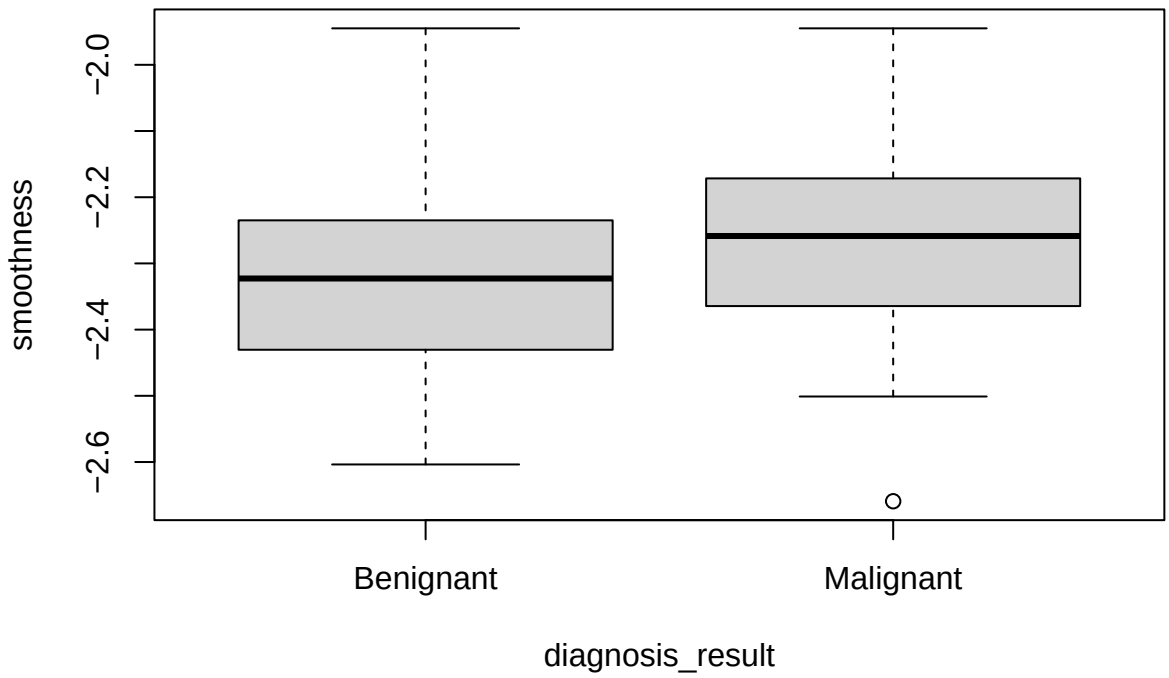

```
boxplot(perimeter ~ diagnosis_result, data = prostate_cancer)
```



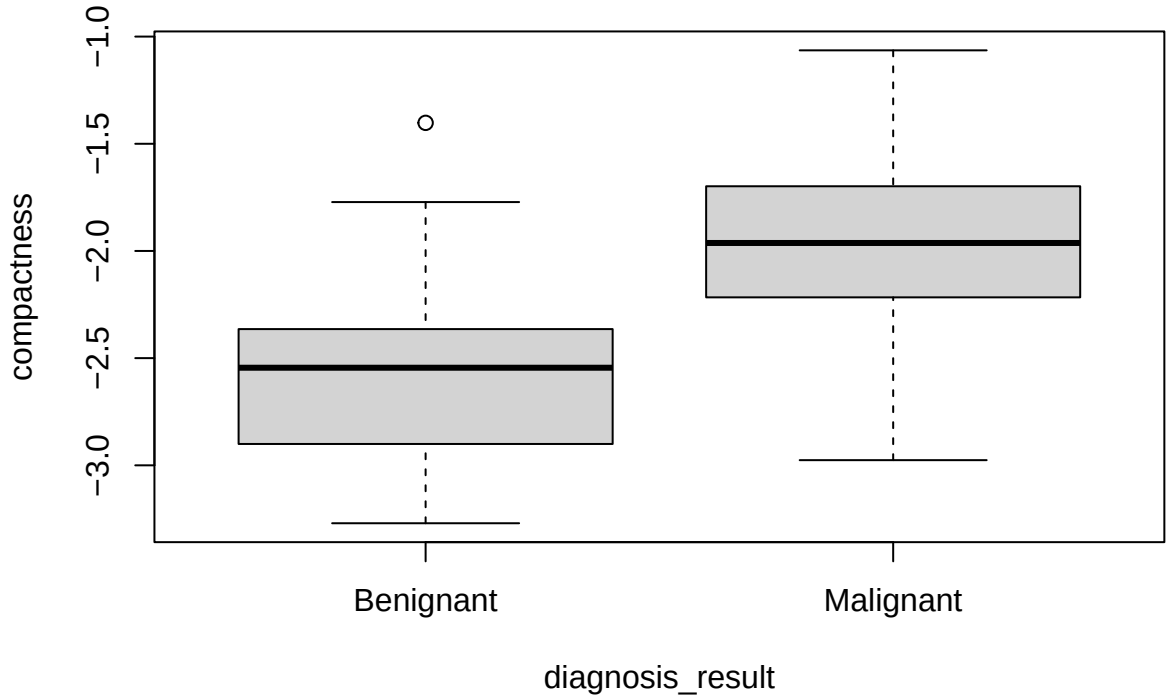

```
boxplot(area ~ diagnosis_result, data = prostate_cancer)
```



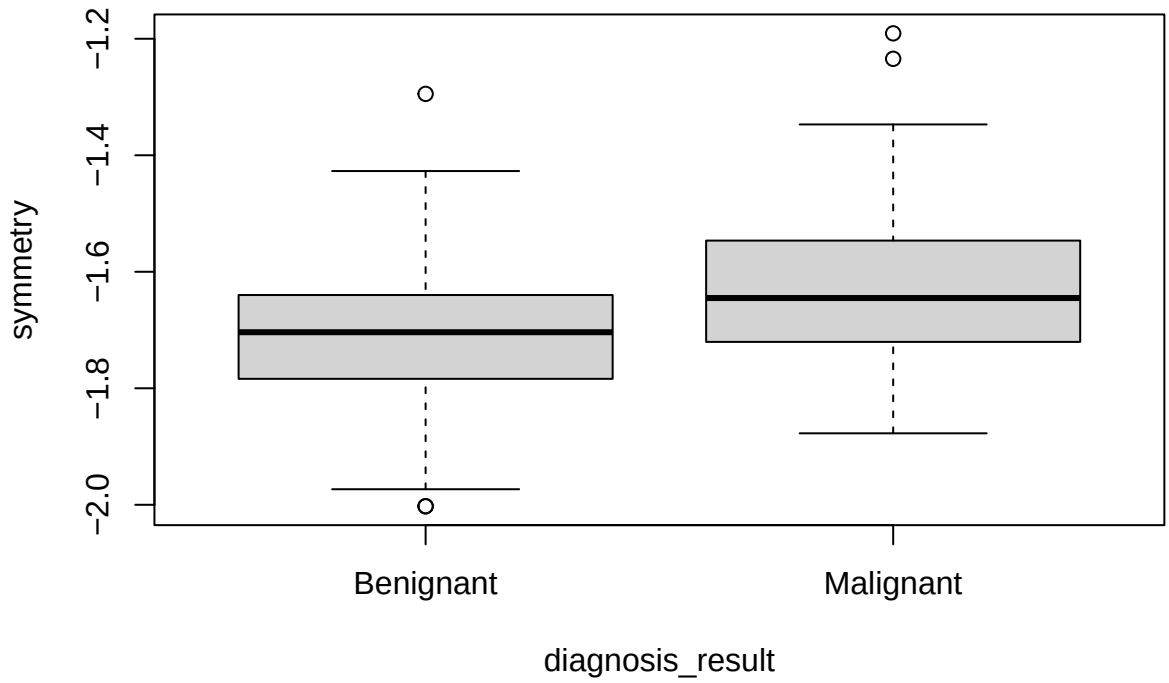

```
boxplot(smoothness ~ diagnosis_result, data = prostate_cancer)
```



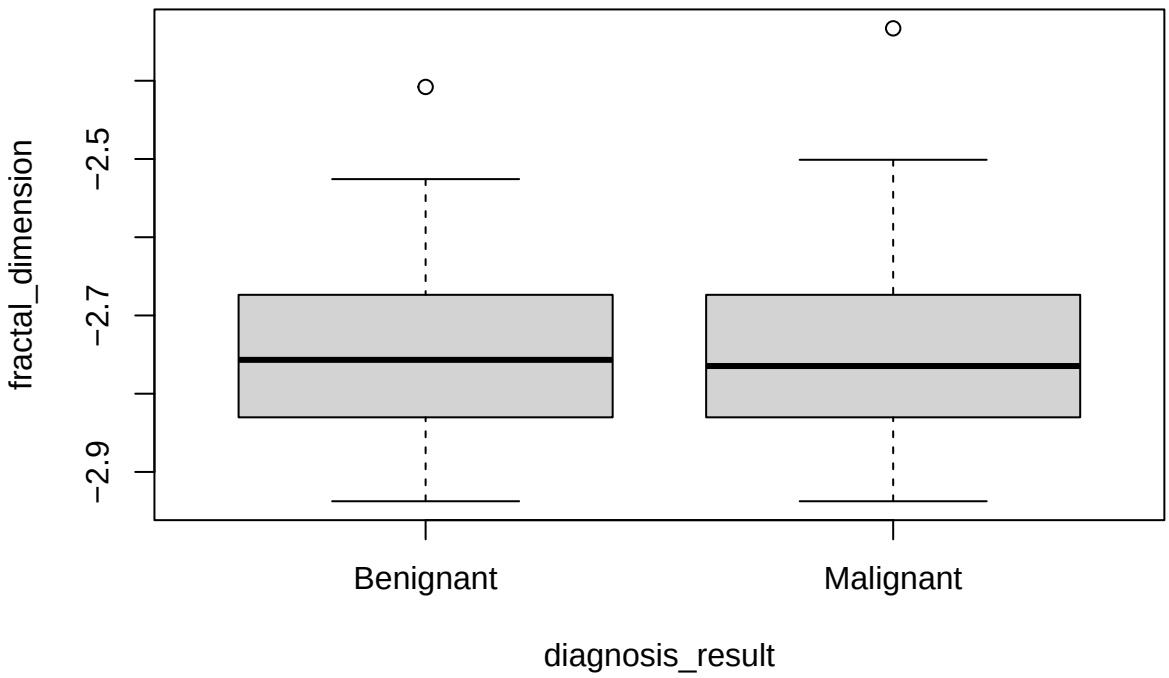

```
boxplot(compactness ~ diagnosis_result, data = prostate_cancer)
```




```
boxplot(symmetry ~ diagnosis_result, data = prostate_cancer)
```



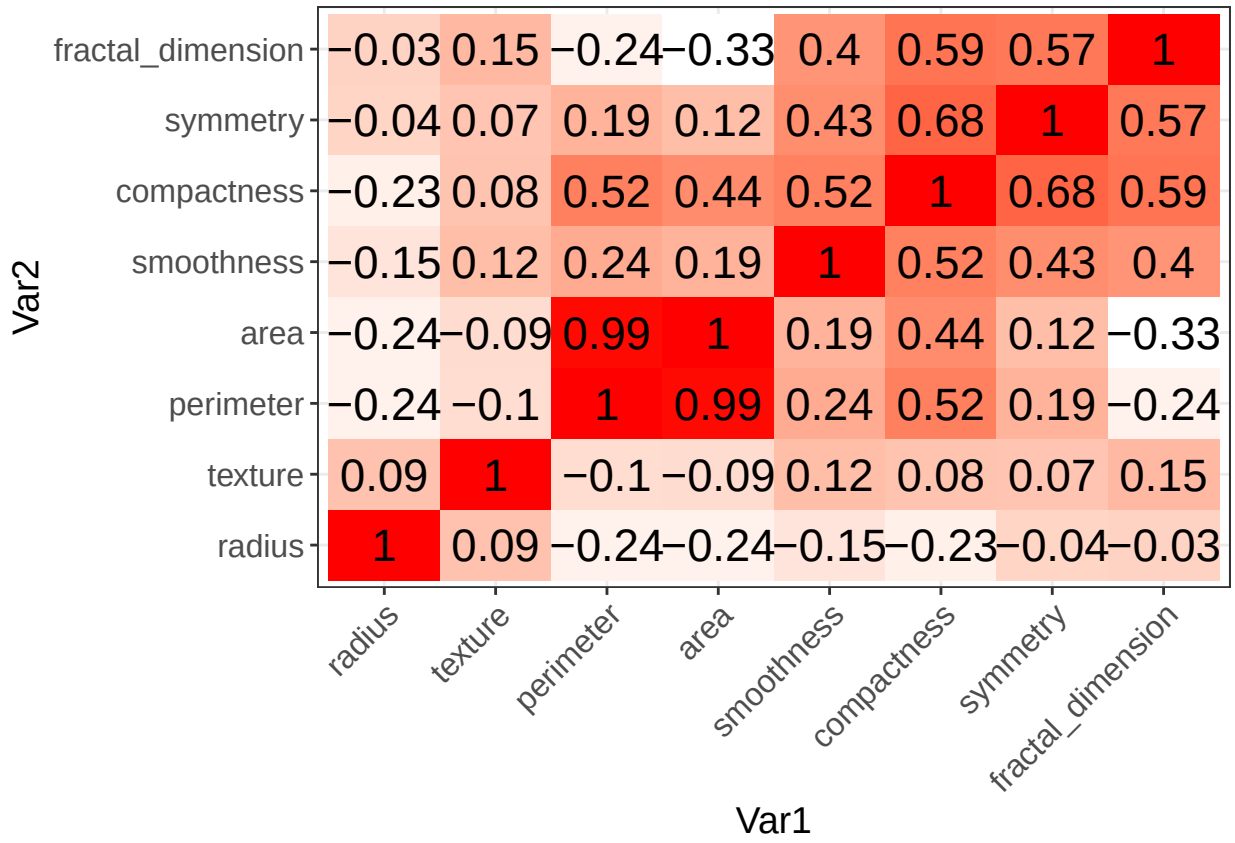

```
boxplot(fractal_dimension ~ diagnosis_result, data = prostate_cancer)
```



Correlation and Covariance

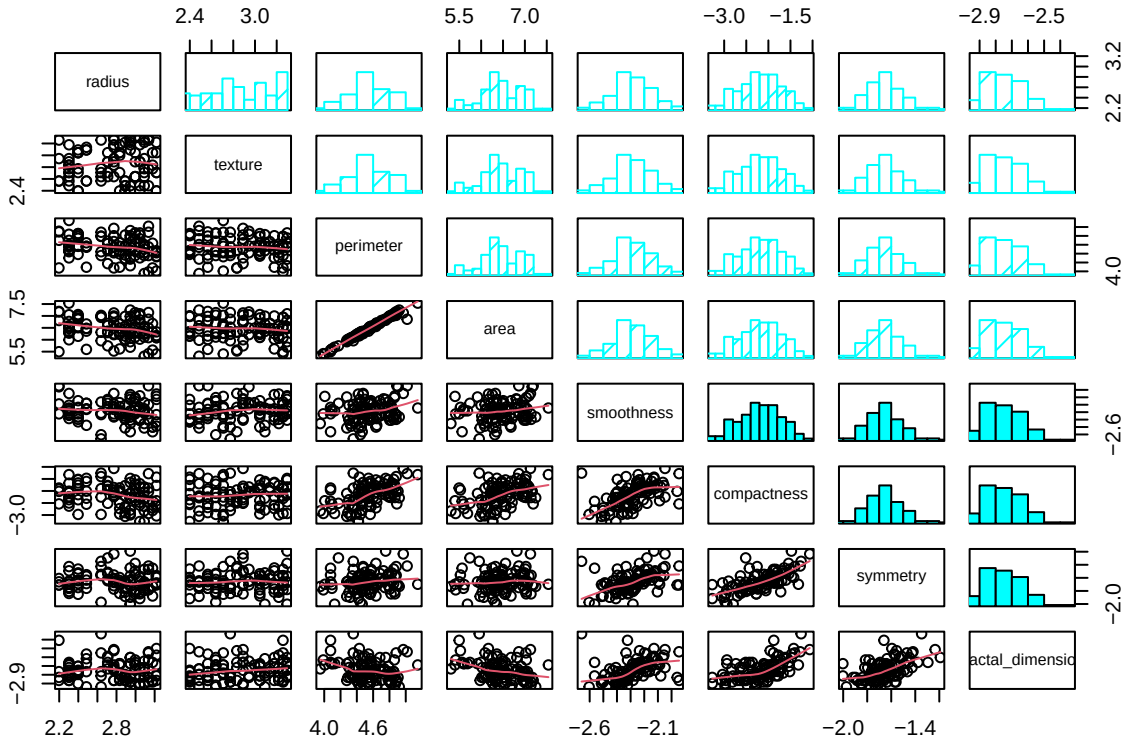
We now check the correlation between all the numerical variables. We do this with with the function *cor()*, that gives us the correlation matrix of the variables, and the package *ggplot*. The plot will display correlation coefficients as numbers, showing the strength and direction of the relationships between the variables.

```
library(dplyr)
library(ellipse)
library(ggplot2)

prostate_cancer_num <- select_if(prostate_cancer, is.numeric)

cor_matrix<-cor(prostate_cancer_num)
cor_df <- as.data.frame(as.table(cor_matrix))
names(cor_df) <- c("Var1", "Var2", "Correlation")

ggplot(data = cor_df, aes(x = Var1, y = Var2, fill = Correlation)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red") +
  geom_text(aes(label = round(Correlation, 2)), size = 6, color = "black") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "none", axis.text = element_text(size = 12),
        axis.title = element_text(size = 14))
```



To see it more graphically, we implement the following code (which includes correlation informations and histograms).

```
panel.hist <- function(x, ...)
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5) )
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks; nB <- length(breaks)
  y <- h$counts; y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col = "cyan", ...)
}

panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor, ...)
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y))
  txt <- format(c(r, 0.123456789), digits = digits)[1]
  txt <- paste0(prefix, txt)
  if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
  text(0.5, 0.5, txt, cex = cex.cor * r)
}

pairs(prostate_cancer_num, upper.panel=panel.hist, lower.panel=panel.smooth)
```



Models

We will now implement some of the principal algorithms and models for binary classification problems: Logistic Regression, Ridge and Lasso Regression, Naive Bayes, Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA) and K-Nearest Neighbors (K-NN).

Before trying all the models, we divide our dataset into training set and test set, so that we can effectively verify the correctness and the performance of our models.

```
library(car)
set.seed(42)
samp <- sample(1:nrow(prostate_cancer), ceiling(0.80*nrow(prostate_cancer)))
training.data<-prostate_cancer[samp,]
test.data<-prostate_cancer[-samp,]
```

Logistic Regression

Let's proceed implementing Logistic Regression, the most famous algorithm for binary classification problems.

```
modelDIAG <- glm(diagnosis_result ~. , data = training.data, family = "binomial")
summary(modelDIAG)
```

```
##
## Call:
## glm(formula = diagnosis_result ~ ., family = "binomial", data = training.data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
```

```
## -1.92266  -0.11409   0.05036   0.21825   2.49190
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)        101.641    109.198   0.931   0.3520
## radius              -1.311      1.413  -0.928   0.3536
## texture              2.238      1.729   1.295   0.1953
## perimeter         -108.751     78.307  -1.389   0.1649
## area                59.133     39.532   1.496   0.1347
## smoothness          -9.927      5.194  -1.911   0.0560 .
## compactness         11.622      5.031   2.310   0.0209 *
## symmetry             6.070      5.942   1.022   0.3070
## fractal_dimension   -8.210     10.034  -0.818   0.4132
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 100.893  on 79  degrees of freedom
## Residual deviance:  34.893  on 71  degrees of freedom
## AIC: 52.893
##
## Number of Fisher Scoring iterations: 7
```

To decide what variable to remove to update our model, we check the VIF values.

```
vif(modelDIAG)
```

```
##            radius           texture         perimeter              area
##          1.256135          1.403616        796.001801        843.568136
##        smoothness       compactness          symmetry fractal_dimension
##          3.083757         19.420121          2.720315          8.662770
```

VIF helps to identify variables that are highly correlated with each other. By examing the VIF values we can see how the removal of certain variables affects the multicollinearity among the remaining predictor variables. High VIF values suggest that a predictor variable may not be providing unique or independent information beyond what other predictors already capture. In such cases, it may be appropriate to remove or transform the highly correlated variables to improve the model's interpretability and stability.

In this case, the variable with the highest VIF value is "area", so we remove it.

```
modelDIAG_update <- update(modelDIAG, ~. -area)
```

We proceed in the same way until all the variables in our model no longer have high VIF values.

```
vif(modelDIAG_update)
```

```
##            radius           texture         perimeter        smoothness
##          1.282738          1.429350          2.787761          2.887704
##       compactness          symmetry fractal_dimension
##          6.309447          2.694029          8.146498
```

```
modelDIAG_update1 <- update(modelDIAG_update, ~. -fractal_dimension)
vif(modelDIAG_update1)
```

```
##      radius     texture   perimeter  smoothness compactness    symmetry
##    1.187040    1.307771    1.658558    2.870762    2.315677    2.839002
```

```
summary(modelDIAG_update1)
```

```
##
## Call:
## glm(formula = diagnosis_result ~ radius + texture + perimeter +
##     smoothness + compactness + symmetry, family = "binomial",
##     data = training.data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.07223  -0.18730   0.07798   0.26440   2.06294
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -59.713     20.509  -2.912  0.00360 **
## radius        -1.144      1.291  -0.886  0.37568
## texture        2.232      1.629   1.371  0.17044
## perimeter     12.350      3.631   3.401  0.00067 ***
## smoothness    -9.858      4.949  -1.992  0.04640 *
## compactness    4.309      1.711   2.519  0.01177 *
## symmetry       6.437      5.516   1.167  0.24320
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 100.893  on 79  degrees of freedom
## Residual deviance:  38.188  on 73  degrees of freedom
## AIC: 52.188
##
## Number of Fisher Scoring iterations: 7
```

We now decide to remove the variable with the highest p-value (Pr(>|z|)). We proceed in the same way until we find a model where all variables have a significantly small p-value.

```
modelDIAG_update2 <- update(modelDIAG_update1, ~. -radius)
summary(modelDIAG_update2)
```

```
##
## Call:
## glm(formula = diagnosis_result ~ texture + perimeter + smoothness +
##     compactness + symmetry, family = "binomial", data = training.data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.14956  -0.20559   0.08383   0.28739   1.94680
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -59.573     20.236  -2.944 0.003241 **
## texture        1.842      1.585   1.162 0.245367
## perimeter     11.901      3.479   3.421 0.000624 ***
## smoothness    -8.600      4.500  -1.911 0.055998 .
## compactness    4.302      1.782   2.414 0.015789 *
## symmetry       4.861      5.389   0.902 0.367001
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 100.893  on 79  degrees of freedom
## Residual deviance:  38.975  on 74  degrees of freedom
## AIC: 50.975
##
## Number of Fisher Scoring iterations: 7
```

```
modelDIAG_update3 <- update(modelDIAG_update2, ~. -symmetry)
summary(modelDIAG_update3)
```

```
##
## Call:
## glm(formula = diagnosis_result ~ texture + perimeter + smoothness +
##     compactness, family = "binomial", data = training.data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.06155  -0.13535   0.09129   0.35553   1.95291
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -58.097     19.757  -2.941  0.00328 **
## texture        1.436      1.478   0.971  0.33135
## perimeter     11.192      3.278   3.414  0.00064 ***
## smoothness    -7.107      4.075  -1.744  0.08111 .
## compactness    5.183      1.666   3.111  0.00187 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 100.893  on 79  degrees of freedom
## Residual deviance:  39.761  on 75  degrees of freedom
## AIC: 49.761
##
## Number of Fisher Scoring iterations: 7
```

```
modelDIAG_update4 <- update(modelDIAG_update3, ~. -texture)
summary(modelDIAG_update4)
```

```
##
## Call:
## glm(formula = diagnosis_result ~ perimeter + smoothness + compactness,
##     family = "binomial", data = training.data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.74882  -0.16443   0.09671   0.36508   2.06046
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept)  -51.064     17.689  -2.887 0.003893 **
## perimeter     10.654      3.174   3.356 0.000789 ***
## smoothness    -7.120      4.096  -1.738 0.082196 .
## compactness    5.425      1.692   3.207 0.001343 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 100.893  on 79  degrees of freedom
## Residual deviance:  40.742  on 76  degrees of freedom
## AIC: 48.742
##
## Number of Fisher Scoring iterations: 7
```

```
resultsDIAG1<- Anova(modelDIAG_update4, modelDIAG, type = "III", test.statistic = NULL)
print(resultsDIAG1)
```

```
## Analysis of Deviance Table (Type III tests)
##
## Response: diagnosis_result
##             LR Chisq Df Pr(>Chisq)
## perimeter    22.7928  1  1.804e-06 ***
## smoothness    2.9936  1    0.08359 .
## compactness  18.3507  1  1.838e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
modelDIAG_update5 <- update(modelDIAG_update4, ~. -smoothness)
summary(modelDIAG_update5)
```

```
##
## Call:
## glm(formula = diagnosis_result ~ perimeter + compactness, family = "binomial",
##     data = training.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7774  -0.2368   0.1199   0.3424   1.8714
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -36.348     13.735  -2.646  0.00813 **
## perimeter     10.094      3.084   3.273  0.00106 **
## compactness    3.617      1.131   3.197  0.00139 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 100.893  on 79  degrees of freedom
## Residual deviance:  43.736  on 77  degrees of freedom
## AIC: 49.736
##
## Number of Fisher Scoring iterations: 6
```

```
resultsDIAG <- Anova(modelDIAG_update5, modelDIAG, type = "III", test.statistic = NULL)
print(resultsDIAG)
```

```
## Analysis of Deviance Table (Type III tests)
##
## Response: diagnosis_result
##             LR Chisq Df Pr(>Chisq)
## perimeter     22.239  1  2.408e-06 ***
## compactness   16.583  1  4.657e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(modelDIAG_update5)
```

```
##
## Call:
## glm(formula = diagnosis_result ~ perimeter + compactness, family = "binomial",
##     data = training.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7774  -0.2368   0.1199   0.3424   1.8714
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -36.348     13.735  -2.646  0.00813 **
## perimeter     10.094      3.084   3.273  0.00106 **
## compactness    3.617      1.131   3.197  0.00139 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 100.893  on 79  degrees of freedom
## Residual deviance:  43.736  on 77  degrees of freedom
## AIC: 49.736
##
## Number of Fisher Scoring iterations: 6
```

Based on this updated model, the significant predictors for diagnosing prostate cancer are perimeter and compactness of the mass.

The results of the analysis of deviance table (Type III tests) indicate the statistical significance of the predictor variables in the logistic regression models `modelDIAG` and `modelDIAG_update5`.

These variables are crucial factors in predicting the outcome and should be considered when analyzing and interpreting the diagnostic results in the context of the logistic regression model.

Predictions We can now use this model to make predictions on the test set and see how it works.

```
glm.probs.test <- predict(modelDIAG_update5, test.data, type="response")
table(test.data$diagnosis_result)
```

```
##
## Benignant Malignant
##        12         8
```

```
glm.pred.test <- rep("Benignant", 12)
glm.pred.test[glm.probs.test>.5] <- "Malignant"
confusion_matrix_glm <- table(glm.pred.test, test.data$diagnosis_result)
confusion_matrix_glm <- addmargins(confusion_matrix_glm, margin = c(1, 2))
confusion_matrix_glm
```

```
##
## glm.pred.test Benignant Malignant Sum
##     Benignant         5         0   5
##     Malignant         4         8  12
##     Sum               9         8  17
```

```
overall_error_rate_test_glm <- (confusion_matrix_glm["Benignant", "Malignant"] + confusion_matrix_glm["
print(overall_error_rate_test_glm)
```

```
## [1] 0.05882353
```

```
# FPR: False Positive Rate = FP/N
fpr_glm <- (confusion_matrix_glm["Benignant", "Malignant"])/(confusion_matrix_glm["Sum", "Malignant"])
fpr_glm
```

```
## [1] 0
```

```
# Specificity
specif_glm <- 1 - fpr_glm
specif_glm
```

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

```
# Sensitivity
tpr_glm <- (confusion_matrix_glm["Benignant", "Benignant"])/(confusion_matrix_glm["Sum", "Benignant"])
tpr_glm
```

```
## [1] 0.5555556
```

The overall error rate is calculated as the sum of false positives and false negatives divided by the total number of cases. In this case, the overall error rate is 0.0588 ans it indicates that the model misclassified approximately 5.88% of the cases.

False Positive Rate (FPR) is the ratio of false positives to the total number of actual negative cases. In this case, the FPR is 0, indicating that the model did not incorrectly classify any actual "Benignant" cases as "Malignant".

Specificity is the complement of the FPR and represents the model's ability to correctly identify negative cases. The specificity, as we expected from the calculation of FPR, is 1, which indicates that the model had a perfect identification of true "Benignant" cases.

True Positive Rate (TPR) is the ratio of true positives to the total number of actual positive cases. In this case, the TPR (also known as sensitivity or recall) is 0.5556: the model correctly identified 55.56% of the actual "Benignant" cases.

Overall, the model demonstrates reasonably good performance, with a low overall error rate and a high specificity (no false positives). However, the true positive rate (sensitivity) is relatively moderate, suggesting that there is room for improvement in correctly identifying positive cases.

Regularized Regression

In this section, we applied regularized regression techniques, specifically Lasso and Ridge regularization, to our generalized models. These methods enable automatic variable selection by shrinking the coefficients of predictors towards zero or even eliminating them entirely. The purpose of these techniques is to handle

multicollinearity and prevent overfitting, where estimators with large variances can result in poor estimates. Based on the results obtained so far, we decided to continue testing our model only on the balanced dataset. In terms of feature selection, Lasso performs variable selection by shrinking coefficients to zero, while Ridge has a similar behavior but gradually shrinks coefficients towards zero rather than directly setting them to zero.

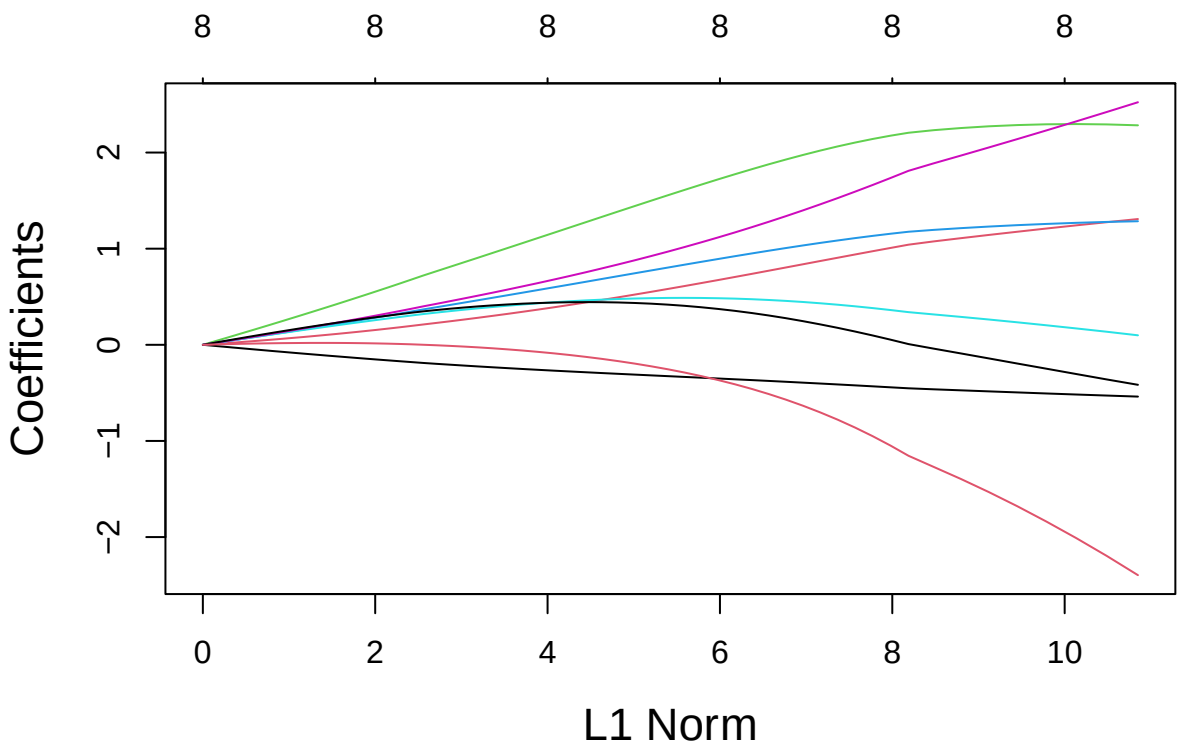
```
library(glmnet)
```

Ridge Regression

```
## Caricamento del pacchetto richiesto: Matrix
```

```
## Loaded glmnet 4.1-7
```

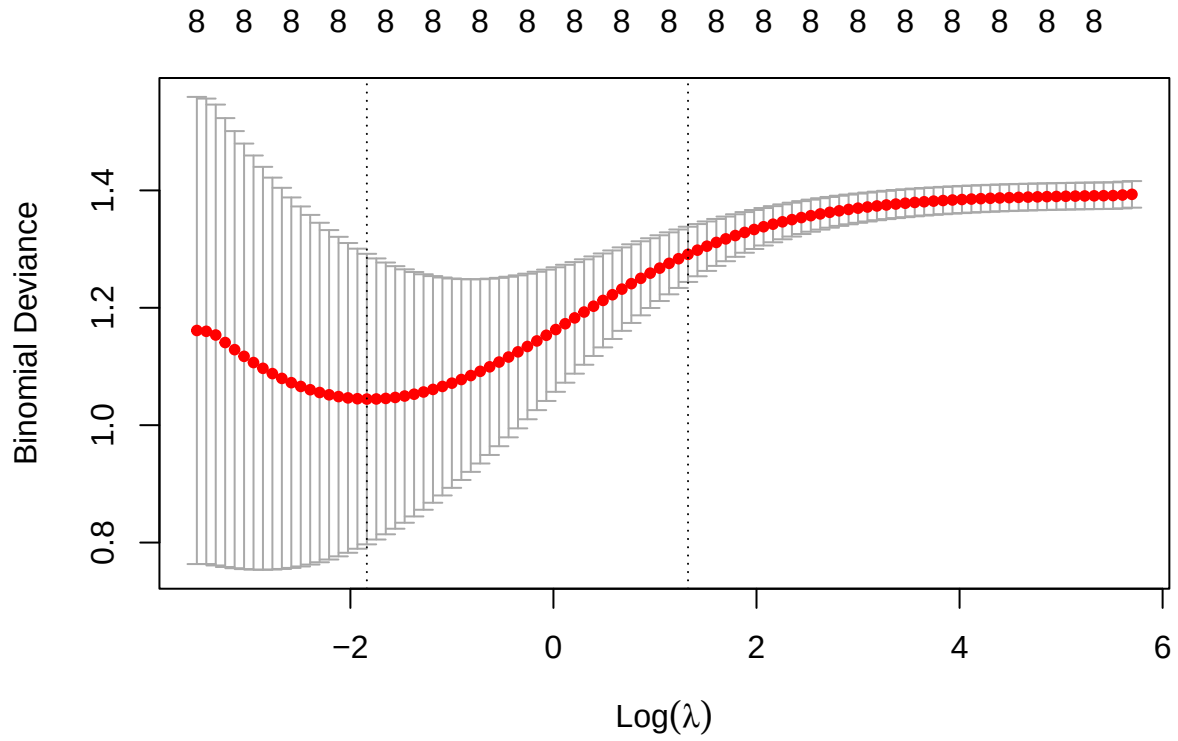
```
set.seed(0607)
#we use a balanced training set beacause Ridge and Lasso regression are sensitive to class imbalances i
train <- sample(nrow(prostate_cancer), nrow(prostate_cancer)/2)
train_data <- prostate_cancer[train, ]
test<-prostate_cancer[-train,]
N<-dim(train_data)[1]
X<-as.matrix(cbind(rep(1,N),train_data[,2:9]))

p<-train_data$diagnosis_result

mod_glmnet<-glmnet(x=X[,-1],y=p,family="binomial", alpha=0)
plot(mod_glmnet,cex.lab=1.4)
```



```
cvfit<-cv.glmnet(x=X[,-1],y=p, family="binomial", alpha=0)
coef(cvfit)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                               s1
## (Intercept)       -0.88222150
## radius            -0.07910443
## texture            0.06835953
## perimeter          0.26808719
## area               0.13626782
## smoothness         0.13272711
## compactness        0.14436181
## symmetry           0.15258727
## fractal_dimension  0.01808046
```

```
plot(cvfit)
```




```
#search for the best lambda
lambda.opt<-cvfit$lambda.min
lambda.opt
```

```
## [1] 0.15917
```

```
ridge.pred <- predict(cvfit, as.matrix(test[,-1]), s =lambda.opt, type="class")
ridge.probs <- predict(cvfit, as.matrix(test[,-1]), s =lambda.opt, type="response")
table(test$diagnosis_result,ridge.pred)
```

```
##            ridge.pred
```

```
##             Benignant Malignant
##   Benignant        13         3
##   Malignant         2        32
```

```
mean( ridge.pred != test$diagnosis_result)
```

```
## [1] 0.1
```

```
coef(cvfit,s="lambda.min")
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                              s1
## (Intercept)       -11.4509716
## radius             -0.3586223
## texture             0.7053843
## perimeter           1.7740396
## area                0.9214143
## smoothness          0.4803323
## compactness         1.1687916
## symmetry            0.3533453
## fractal_dimension  -0.4102882
```

```
# confusion matrix
confusion_matrix_ridge<- table(test$diagnosis_result, ridge.pred)
confusion_matrix_ridge<- addmargins(confusion_matrix_ridge, margin = c(1, 2))
confusion_matrix_ridge
```

```
##            ridge.pred
##             Benignant Malignant Sum
##   Benignant        13         3  16
##   Malignant         2        32  34
##   Sum              15        35  50
```

```
# FPR: False Positive Rate = FP/N
fpr_ridge <- (confusion_matrix_ridge["Benignant", "Malignant"])/(confusion_matrix_ridge["Sum", "Malignan
fpr_ridge
```

```
## [1] 0.08571429
```

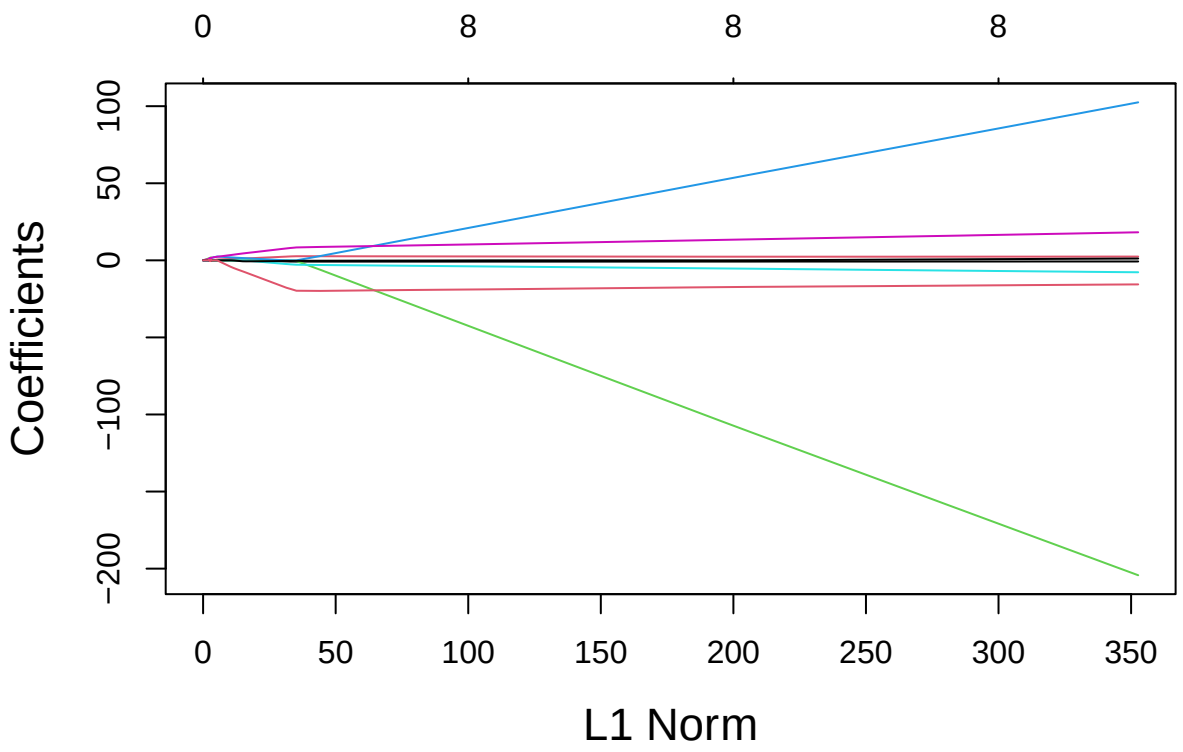
```
#Specificity
specif_ridge <- 1 - fpr_ridge
specif_ridge
```

```
## [1] 0.9142857
```

```
#Sensitivity
tpr_ridge <- (confusion_matrix_ridge["Benignant", "Benignant"])/(confusion_matrix_ridge["Sum", "Benignan
tpr_ridge
```

```
## [1] 0.8666667
```

```
mod_glmnet_lasso<-glmnet(x=X[,-1],y=p,family="binomial", alpha=1)
plot(mod_glmnet_lasso,cex.lab=1.4)
```

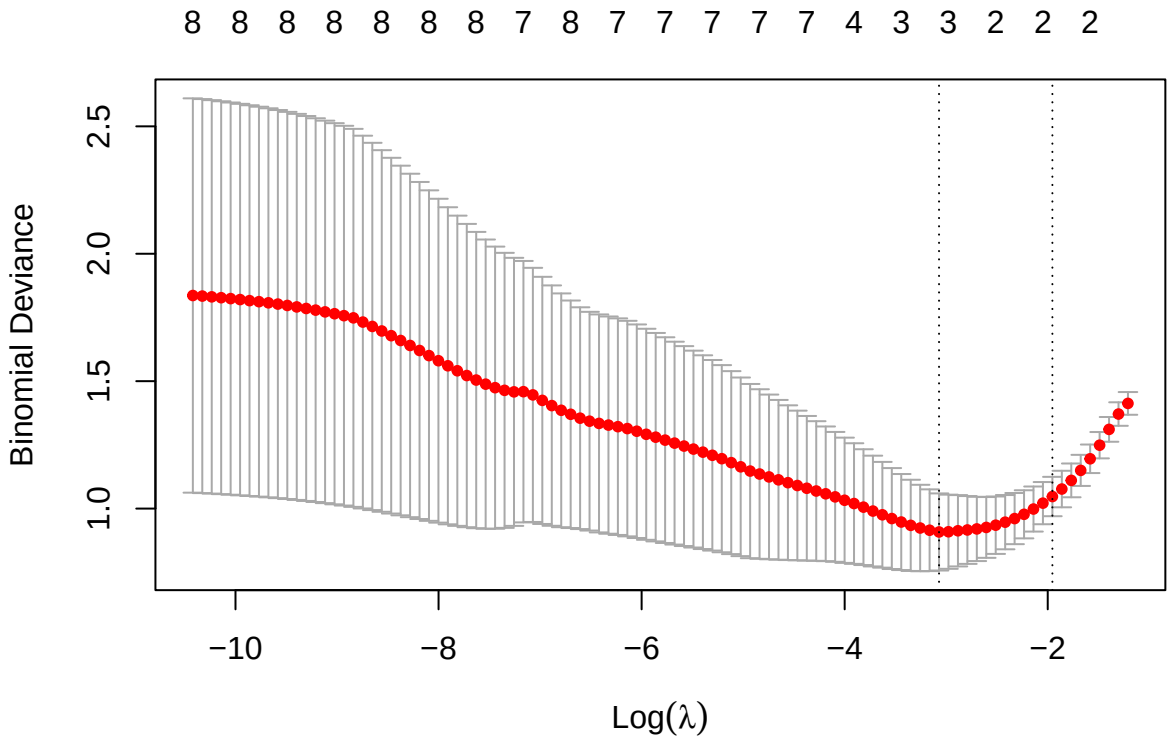


Lasso Regression

```
cvfit_lasso<-cv.glmnet(x=X[,-1],y=p, family="binomial", alpha=1)
coef(cvfit_lasso)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                              s1
## (Intercept)       -5.143436
## radius             .
## texture            .
## perimeter          1.701809
## area               .
## smoothness         .
## compactness        1.017585
## symmetry           .
## fractal_dimension  .
```

```
plot(cvfit_lasso)
```




```
# We identify th best lambda value
lambda.opt.lasso<-cvfit_lasso$lambda.min
lambda.opt.lasso
```

```
## [1] 0.04639891
```

```
# We compute predictions with the lasso model using the best value for# lambda that we have obtained
lasso.pred <- predict(cvfit_lasso, as.matrix(test[,-1]), s =lambda.opt.lasso, type="class")
#confusion matrix
table(test$diagnosis_result, lasso.pred)
```

```
##             lasso.pred
##              Benignant Malignant
##   Benignant         13         3
##   Malignant          4        30
```

```
#error rate
mean( lasso.pred != test$diagnosis_result)
```

```
## [1] 0.14
```

```
#confusion matrix
confusion_matrix_lasso<- table(test$diagnosis_result, lasso.pred)
confusion_matrix_lasso<- addmargins(confusion_matrix_lasso, margin = c(1, 2))
confusion_matrix_lasso
```

```
##             lasso.pred
##              Benignant Malignant Sum
##   Benignant         13         3  16
```

```
##   Malignant         4        30  34
##   Sum              17        33  50
```

```
#FPR: False Positive Rate = FP/N
fpr_lasso <- (confusion_matrix_lasso["Benignant", "Malignant"])/(confusion_matrix_lasso["Sum", "Malignan
fpr_lasso
```

```
## [1] 0.09090909
```

```
#Specificity
specif_lasso <- 1 - fpr_lasso
specif_lasso
```

```
## [1] 0.9090909
```

```
#Sensitivity
tpr_lasso <- (confusion_matrix_lasso["Benignant", "Benignant"])/(confusion_matrix_lasso["Sum", "Benignan
tpr_lasso
```

```
## [1] 0.7647059
```

When comparing Lasso and Ridge models, using the Ridge alternative allows us to achieve lower total error and True Positive rates compared to the Lasso model.

Bayes Classifier

The Bayes Classifier is a probabilistic model used for classification problems. It is based on the Bayes Theorem, which allows us to predict the probability of an event happening given the occurrence of certain events.

To build the Bayes Classifier model, we can use the features selected through the VIF (Variance Inflation Factor) selection method. The VIF helps identify relevant features that contribute to the classification task.

We'll use the `e1071` library in R to implement the Bayes Classifier.

```
# Load the necessary libraries
library(e1071)

# Train the Naive Bayes Classifier
nb.fit <- naiveBayes(diagnosis_result ~ . - area - fractal_dimension, data = training.data)
nb.fit
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
## Benignant Malignant
##     0.325     0.675
##
## Conditional probabilities:
##            radius
## Y               [,1]      [,2]
##   Benignant 2.802889 0.3133678
##   Malignant 2.718552 0.3065060
##
```

```
##            texture
## Y               [,1]      [,2]
##   Benignant 2.804656 0.3101136
##   Malignant 2.898281 0.2784325
##
##            perimeter
## Y               [,1]      [,2]
##   Benignant 4.342824 0.1903640
##   Malignant 4.644660 0.1775257
##
##            smoothness
## Y                [,1]      [,2]
##   Benignant -2.327380 0.1458493
##   Malignant -2.262204 0.1302679
##
##            compactness
## Y                [,1]      [,2]
##   Benignant -2.559157 0.3696100
##   Malignant -1.957651 0.3782103
##
##            symmetry
## Y                [,1]      [,2]
##   Benignant -1.734351 0.1470999
##   Malignant -1.622015 0.1423746
```

```
# Make predictions on the test data
nb.class <- predict(nb.fit, test.data)

# Obtain raw predicted probabilities
nb.preds <- predict(nb.fit, test.data, type = "raw")
nb.preds[1:5,]
```

```
##         Benignant Malignant
## [1,] 3.259223e-05 0.9999674
## [2,] 3.193568e-02 0.9680643
## [3,] 8.799800e-01 0.1200200
## [4,] 1.219948e-04 0.9998780
## [5,] 7.919202e-02 0.9208080
```

```
confusion_matrix_nb <- table(nb.class,test.data$diagnosis_result)
confusion_matrix_nb <- addmargins(confusion_matrix_nb, margin = c(1, 2))
confusion_matrix_nb
```

```
##
## nb.class    Benignant Malignant Sum
##   Benignant         8         1   9
##   Malignant         4         7  11
##   Sum              12         8  20
```

```
# error rate
err_nb <- mean(nb.class != test.data$diagnosis_result)
err_nb
```

```
## [1] 0.25
```

```
# FPR: False Positive Rate = FP/N
```

```r
fpr_nb <- (confusion_matrix_nb["Benignant", "Malignant"])/(confusion_matrix_nb["Sum", "Malignant"])
fpr_nb
```

```
## [1] 0.125
```

```r
#Specificity

specif_nb <- 1 - fpr_nb
specif_nb
```

```
## [1] 0.875
```

```r
#Sensitivity

tpr_nb <- (confusion_matrix_nb["Benignant", "Benignant"])/(confusion_matrix_nb["Sum", "Benignant"])
tpr_nb
```

```
## [1] 0.6666667
```

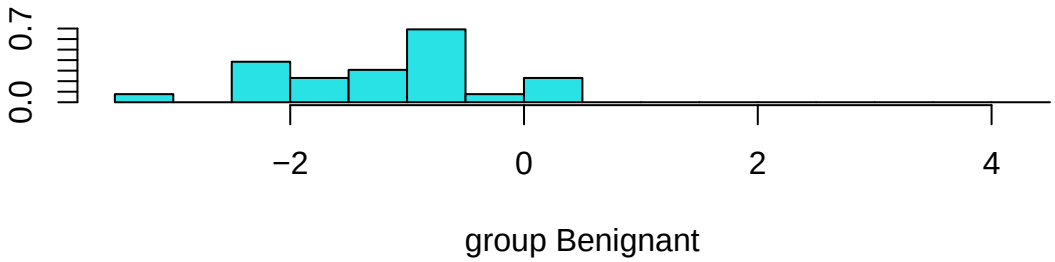
Linear Discriminant Analysis (LDA)

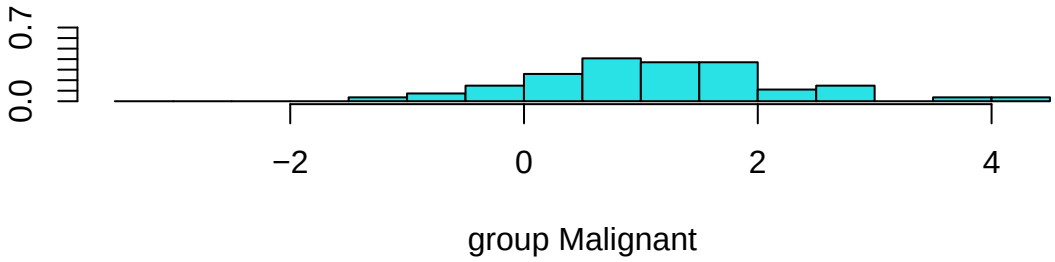
The Linear Discriminant Analysis (LDA) model is a powerful technique commonly used for classification tasks. We incorporated the features selected through the VIF (Variance Inflation Factor) method, which helps identify relevant features for the analysis.

```r
# Load the necessary libraries
library(MASS)

# Train the LDA model
prostate.lda <- lda(diagnosis_result ~ . - area - fractal_dimension, data = training.data)
prostate.lda
```

```
## Call:
## lda(diagnosis_result ~ . - area - fractal_dimension, data = training.data)
##
## Prior probabilities of groups:
## Benignant Malignant
##     0.325     0.675
##
## Group means:
##             radius  texture perimeter smoothness compactness  symmetry
## Benignant 2.802889 2.804656  4.342824  -2.327380   -2.559157 -1.734351
## Malignant 2.718552 2.898281  4.644660  -2.262204   -1.957651 -1.622015
##
## Coefficients of linear discriminants:
##                     LD1
## radius      -0.02892069
## texture      0.67594571
## perimeter    3.87798048
## smoothness  -1.74570719
## compactness  1.85248056
## symmetry     0.11618038
```

```r
plot(prostate.lda, type="histogram")
```





In the plot we can see that the coefficients of the linear discriminant analysis are distributed using a histogram.The histogram visualizes how the coefficients are distributed and provides insights into the separation between the two groups. The two groups are well separated and overlap just a bit. As before we computed the Confusion Matrix, the error and the other measures.

```
lda.pred <- predict(prostate.lda,test.data)
names(lda.pred)
```

Evaluation Metrics

```
## [1] "class"     "posterior" "x"
```

```
lda.class <- lda.pred$class

confusion_matrix_lda <- table(lda.class,test.data$diagnosis_result)
confusion_matrix_lda <- addmargins(confusion_matrix_lda, margin = c(1, 2))
confusion_matrix_lda
```

```
##
## lda.class   Benignant Malignant Sum
##   Benignant         9         0   9
##   Malignant         3         8  11
##   Sum              12         8  20
```

```
# error rate
err_lda <- mean(lda.class!=test.data$diagnosis_result)
err_lda
```

```
## [1] 0.15
```

```
# FPR: False Positive Rate = FP/N
fpr_lda <- (confusion_matrix_lda["Benignant", "Malignant"])/(confusion_matrix_lda["Sum", "Malignant"])
fpr_lda
```

```
## [1] 0
```

```
#Specificity
specif_lda <- 1 - fpr_lda
specif_lda
```

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

```
#Sensitivity
tpr_lda <- (confusion_matrix_lda["Benignant", "Benignant"])/(confusion_matrix_lda["Sum", "Benignant"])
tpr_lda
```

```
## [1] 0.75
```

In this particular case, we achieved satisfactory results in terms of classification accuracy and precision. The model performed well in accurately classifying the observations, and the precision value indicates a high proportion of correctly predicted positive cases relative to false positives.

Quadratic Discriminant Analysis (QDA)

In addition to Linear Discriminant Analysis (LDA), we also implemented Quadratic Discriminant Analysis (QDA) for classification. QDA is a powerful technique that allows for more flexibility in capturing complex relationships between features.

```
prostate.qda <- qda(diagnosis_result~.-area-fractal_dimension, data=training.data)
prostate.qda
```

```
## Call:
## qda(diagnosis_result ~ . - area - fractal_dimension, data = training.data)
##
## Prior probabilities of groups:
## Benignant Malignant
##     0.325     0.675
##
## Group means:
##             radius  texture perimeter smoothness compactness  symmetry
## Benignant 2.802889 2.804656  4.342824  -2.327380   -2.559157 -1.734351
## Malignant 2.718552 2.898281  4.644660  -2.262204   -1.957651 -1.622015
```

```
qda.pred <- predict(prostate.qda, test.data)
names(qda.pred)
```

```
## [1] "class"     "posterior"
```

```
qda.class <- qda.pred$class

confusion_matrix_qda <- table(qda.class,test.data$diagnosis_result)
confusion_matrix_qda <- addmargins(confusion_matrix_qda, margin = c(1, 2))
confusion_matrix_qda
```

```
##
## qda.class   Benignant Malignant Sum
##   Benignant         8         1   9
```

```
##   Malignant         4         7  11
##   Sum              12         8  20
```

```
# error rate
err_qda <- mean(qda.class!=test.data$diagnosis_result)
err_qda
```

```
## [1] 0.25
```

```
# FPR: False Positive Rate = FP/N
fpr_qda <- (confusion_matrix_qda["Benignant", "Malignant"])/(confusion_matrix_qda["Sum", "Malignant"])
fpr_qda
```

```
## [1] 0.125
```

```
#Specificity
specif_qda <- 1 - fpr_qda
specif_qda
```

```
## [1] 0.875
```

```
#Sensitivity
tpr_qda <- (confusion_matrix_qda["Benignant", "Benignant"])/(confusion_matrix_qda["Sum", "Benignant"])
tpr_qda
```

```
## [1] 0.6666667
```

In this case we observe an overall increase in the total error. The False Positive rate is 0,125. This suggests, that, even if the LDA model has better values, also the QDA model performs well for our classification task and can be considered a suitable solution.

K-Nearest Neighbors

The K-Nearest Neighbors (K-NN) model is a classification model that selects the k nearest neighbors to a new point and assigns the label based on the majority vote of those neighbors. To determine the optimal value of k, we experiment with different values and choose the one that result in the lowest error on the training set. In our case, the best-performing value of k is 4.

```
library(caret)
library(class)
pcancer_train_vif <- scale(training.data[-c(1,5,9)])
pcancer_test_vif <- scale(test.data[-c(1,5,9)])

kmax <- 80
err <- rep(0,kmax)
for (l in 1:kmax){
  knn_predictor <- knn(pcancer_train_vif, pcancer_test_vif, prostate_cancer$diagnosis_result[samp], k=l
  err[l] <- mean(knn_predictor != prostate_cancer$diagnosis_result[-samp])
}
k<- which.min(err)
k
```

```
## [1] 4
```

In our implemented model, we opt to use a subset of features obtained through VIF (Variance Inflation Factor) selection. By doing so, we aim to eliminate collinearity among the features, which could have a negative impact on the classifier's performance.

```
knn_predictor <- knn(pcancer_train_vif, pcancer_test_vif, prostate_cancer$diagnosis_result[samp], k)
knn_predictor
```

```
##  [1] Malignant Malignant Malignant Malignant Malignant Malignant Malignant
##  [8] Malignant Benignant Benignant Malignant Benignant Malignant Benignant
## [15] Malignant Malignant Benignant Malignant Malignant Benignant
## Levels: Benignant Malignant
```

Similarly, we compute the confusion matrix for evaluating the performance of the classifier in this scenario as well.

```
confusion_matrix_knn <- table(test.data$diagnosis_result, knn_predictor)
confusion_matrix_knn <- addmargins(confusion_matrix_knn, margin = c(1, 2))
confusion_matrix_knn
```

```
##            knn_predictor
##             Benignant Malignant Sum
##   Benignant         6         6  12
##   Malignant         0         8   8
##   Sum               6        14  20
```

```
# error rate
err_k <- mean(knn_predictor != prostate_cancer$diagnosis_result[-samp])
err_k
```

```
## [1] 0.3
```

```
# FPR: False Positive Rate = FP/N
fpr_knn <- (confusion_matrix_knn["Benignant", "Malignant"])/(confusion_matrix_knn["Sum", "Malignant"])
fpr_knn
```

```
## [1] 0.4285714
```

```
#Specificity
specif_knn <- 1 - fpr_knn
specif_knn
```

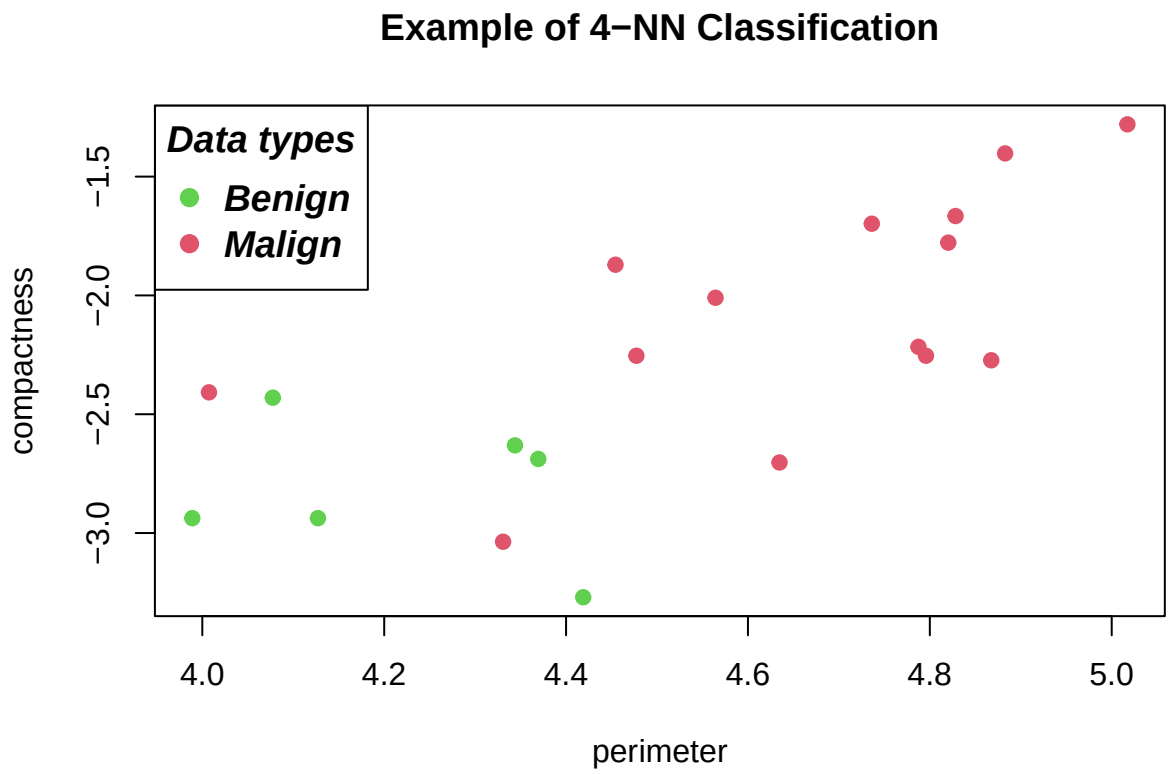
```
## [1] 0.5714286
```

```
#Sensitivity
tpr_knn <- (confusion_matrix_knn["Benignant", "Benignant"])/(confusion_matrix_knn["Sum", "Benignant"])
tpr_knn
```

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

Upon examining the Confusion Matrix, it becomes evident that the model makes errors specifically in classifying benign tumors. Given the context of dealing with tumors, it is crucial not to misclassify a malignant tumor as benign. Consequently, when comparing all the models, this particular result must be taken into consideration.

```
# KNN PLOT
BB <- knn_predictor =="Benignant"
plot(prostate_cancer$perimeter[-samp], prostate_cancer$compactness[-samp],
     col = BB+2,xlab="perimeter",ylab="compactness",
     main="Example of 4-NN Classification", pch=19, cex=1)
legend("topleft", legend = c("Benign", "Malign"),
       col = c(3, 2), pch = 19, cex = 1.2,
       title = "Data types", text.font = 4)
```



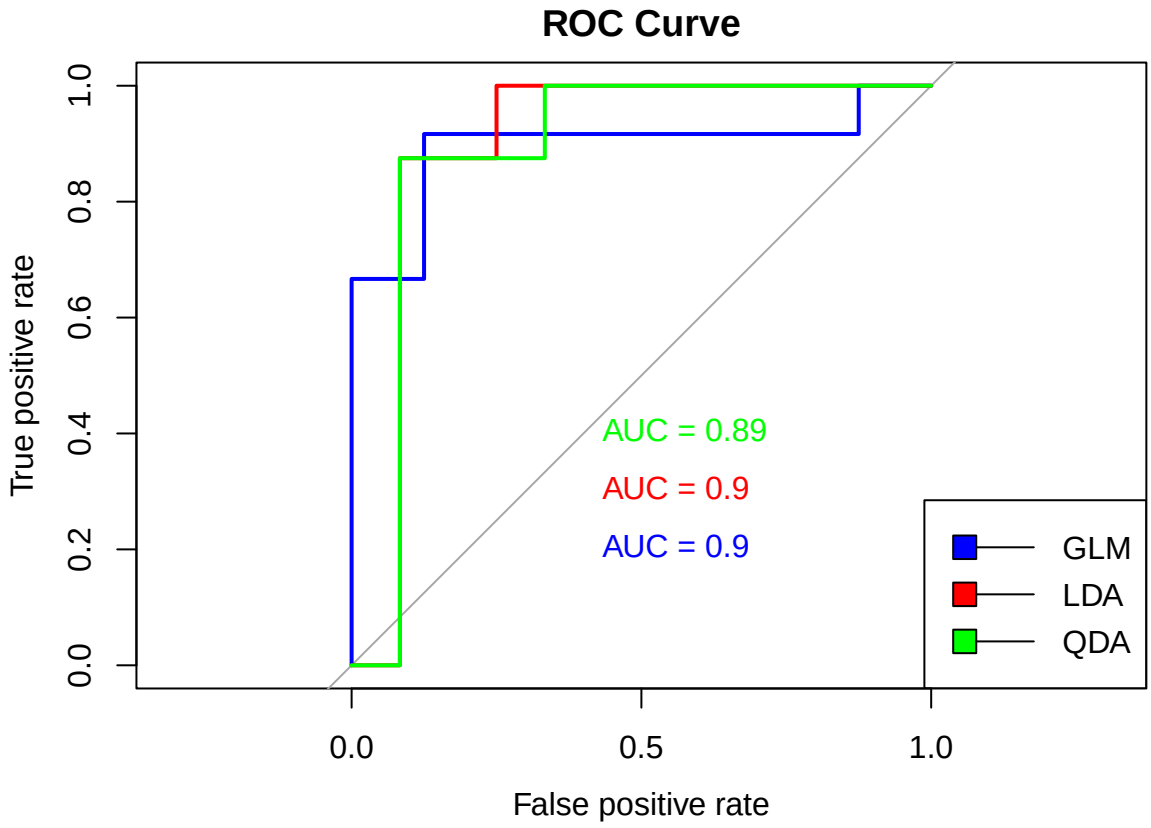
K-NN PLOT

In the plot, we can observe the classification of points by the model. It is clear that the model is making some errors, which aligns with the values obtained in the confusion matrix.

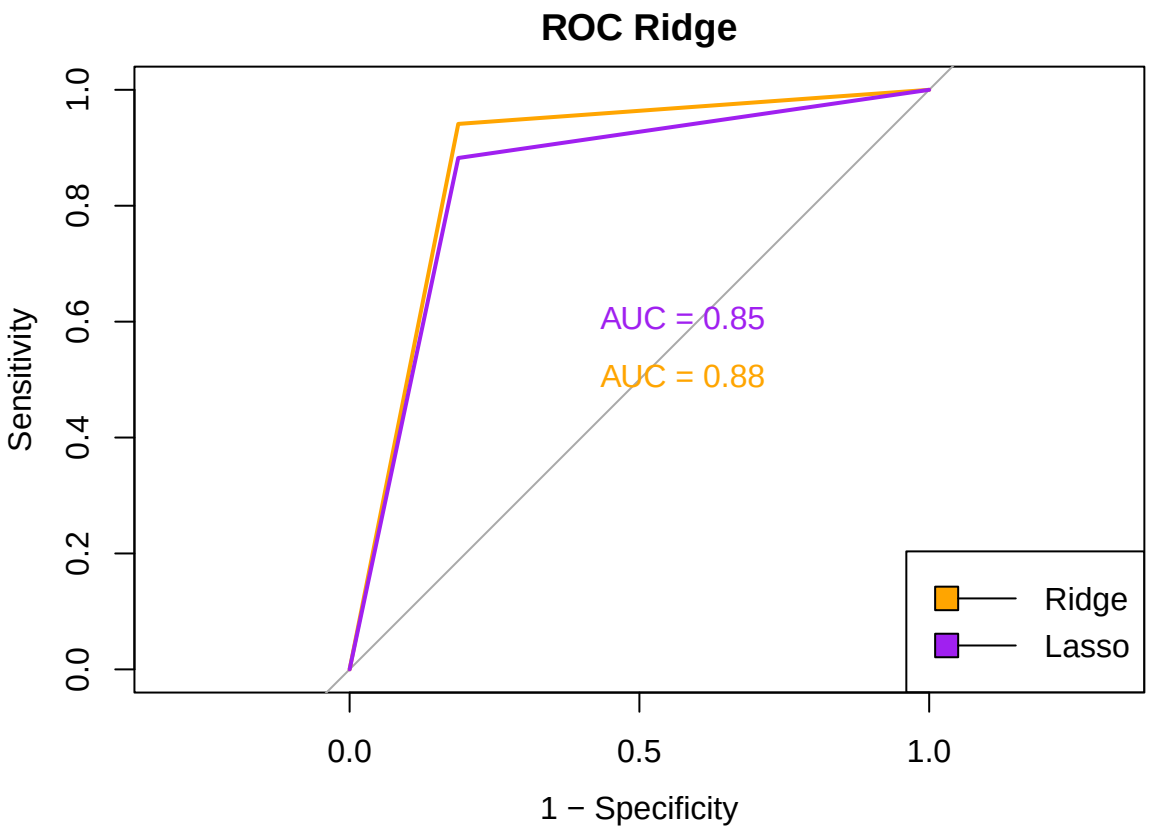
Comparison of the different ROC curves

```
library(pROC)

true_labels <- as.numeric(test.data$diagnosis_result == "Malignant")

# ROC curve - GLM
roc.out_glm <- roc(test.data$diagnosis_result, glm.probs.test, levels = c("Malignant", "Benignant"))
plot(roc.out_glm, legacy.axes = TRUE, xlab = "False positive rate", ylab = "True positive rate",
     col = "blue", main = "ROC Curve")
auc_value_glm <- auc(roc.out_glm)
text(0.6, 0.2, paste("AUC =", round(auc_value_glm, 2)), col = "blue", cex = 1, pos = 4)

# ROC curve - LDA
lda_pred_pos <- lda.pred$posterior[, "Malignant"]
roc_obj_lda <- roc(true_labels, lda_pred_pos)
plot(roc_obj_lda, legacy.axes = TRUE, col = "red", add = TRUE)
auc_value_lda <- auc(roc_obj_lda)
text(0.6, 0.3, paste("AUC =", round(auc_value_lda, 2)), col = "red", cex = 1, pos = 4)

# ROC curve - QDA
qda_pred_pos <- qda.pred$posterior[, "Malignant"]
roc_obj_qda <- roc(true_labels, qda_pred_pos)
```

```r
plot(roc_obj_qda, legacy.axes = TRUE, col = "green", add = TRUE)
auc_value_qda <- auc(roc_obj_qda)
text(0.6, 0.4, paste("AUC =", round(auc_value_qda, 2)), col = "green", cex = 1, pos = 4)

legend("bottomright", legend = c("GLM", "LDA", "QDA"),
       fill = c("blue", "red", "green"), lty = 1, cex = 1)
```




```r
true_labels_rl = as.numeric(test$diagnosis_result == "Malignant")

# ROC curve - RIDGE
ridge_pred_pos <- as.numeric(ridge.pred == "Malignant")
roc_obj_ridge <- roc(true_labels_rl, ridge_pred_pos)
plot(roc_obj_ridge, legacy.axes = TRUE, col = "orange", main = "ROC Ridge")
auc_value_ridge <- auc(roc_obj_ridge)
text(0.6, 0.5, paste("AUC =", round(auc_value_ridge, 2)), col = "orange", cex = 1, pos = 4)

# ROC curve - LASSO
lasso_pred_pos <- as.numeric(lasso.pred == "Malignant")
roc_obj_lasso <- roc(true_labels_rl, lasso_pred_pos)
plot(roc_obj_lasso, legacy.axes = TRUE, col = "purple", add = TRUE)
auc_value_lasso <- auc(roc_obj_lasso)
text(0.6, 0.6, paste("AUC =", round(auc_value_lasso, 2)), col = "purple", cex = 1, pos = 4)


legend("bottomright", legend = c("Ridge", "Lasso"),
       fill = c( "orange", "purple"), lty = 1, cex = 1)
```



After analyzing the ROC curve plot and the Area Under the Curve (AUC), we can conclude that the QDA and LDA models perform the best among all the models for our classification task. While the other models exhibit good accuracy.

Final considerations

We can now summarize the results obtained from all the implemented models in order to compare their performances on a similar level.

Considering the error values, it is evident that Logistic Regression and Ridge Regression exhibit the smallest errors among the models.

However, our primary goal is to ensure accurate classification of tumors, which means we want sensitivity and specificity values as close to one as possible. In terms of sensitivity, the best model is KNN, while Logistic Regression and LDA perform best in terms of specificity.

Taking all factors into account, including error, sensitivity, and specificity, we can conclude that Logistic Regression is the most favorable model for predicting the nature of a tumor in this particular case.