```r
# load in dataset
library(ggplot2)
library(tidyr)
library(caret)

## Loading required package: lattice

library(stats)
library(factoextra)

## Welcome! Want to learn more? See two factoextra-related books at
https://goo.gl/ve3WBa

library(e1071)
library(pROC)

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

library(xgboost)

## Warning: package 'xgboost' was built under R version 4.3.1

library(stringr)
library(randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

library(gridExtra)

##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:randomForest':
##
##     combine

library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

library(reshape2)

##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths

library(ROSE)

## Warning: package 'ROSE' was built under R version 4.3.1

## Loaded ROSE 0.0-4

library(Rtsne)
library(corrplot)

## corrplot 0.92 loaded

library(car)

## Loading required package: carData

df <-
read.csv("C:/MIDS/ADS-503_Applied_Predictive_Modeling/ADS_503_team_2_final_pr
oject/data/breast_cancer_FNA_data.csv")
# remove the x and Id from the end of the data set since it is all null
values
# "x" column is an error from csv column spacing, and can be removed
entirely.
df$X <- NULL
df$id <- NULL
df_diag <- df
```

Summary Stats, Dimensions & NA values

```
# get descriptive statistics for the data set
summary(df)

##   diagnosis          radius_mean      texture_mean   perimeter_mean
##  Length:569         Min.   : 6.981   Min.   : 9.71   Min.   : 43.79
##  Class :character   1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17
##  Mode  :character   Median :13.370   Median :18.84   Median : 86.24
##                     Mean   :14.127   Mean   :19.29   Mean   : 91.97
##                     3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10
##                     Max.   :28.110   Max.   :39.28   Max.   :188.50
##    area_mean      smoothness_mean   compactness_mean  concavity_mean
##  Min.   : 143.5   Min.   :0.05263   Min.   :0.01938   Min.   :0.00000
##  1st Qu.: 420.3   1st Qu.:0.08637   1st Qu.:0.06492   1st Qu.:0.02956
```

```
##  Median : 551.1   Median :0.09587   Median :0.09263   Median :0.06154
##  Mean   : 654.9   Mean   :0.09636   Mean   :0.10434   Mean   :0.08880
##  3rd Qu.: 782.7   3rd Qu.:0.10530   3rd Qu.:0.13040   3rd Qu.:0.13070
##  Max.   :2501.0   Max.   :0.16340   Max.   :0.34540   Max.   :0.42680
##  concave.points_mean symmetry_mean    fractal_dimension_mean   radius_se
##  Min.   :0.00000     Min.   :0.1060   Min.   :0.04996        Min.
:0.1115
##  1st Qu.:0.02031     1st Qu.:0.1619   1st Qu.:0.05770        1st
Qu.:0.2324
##  Median :0.03350     Median :0.1792   Median :0.06154        Median
:0.3242
##  Mean   :0.04892     Mean   :0.1812   Mean   :0.06280        Mean
:0.4052
##  3rd Qu.:0.07400     3rd Qu.:0.1957   3rd Qu.:0.06612        3rd
Qu.:0.4789
##  Max.   :0.20120     Max.   :0.3040   Max.   :0.09744        Max.
:2.8730
##    texture_se      perimeter_se       area_se        smoothness_se
##  Min.   :0.3602   Min.   : 0.757   Min.   :  6.802   Min.   :0.001713
##  1st Qu.:0.8339   1st Qu.: 1.606   1st Qu.: 17.850   1st Qu.:0.005169
##  Median :1.1080   Median : 2.287   Median : 24.530   Median :0.006380
##  Mean   :1.2169   Mean   : 2.866   Mean   : 40.337   Mean   :0.007041
##  3rd Qu.:1.4740   3rd Qu.: 3.357   3rd Qu.: 45.190   3rd Qu.:0.008146
##  Max.   :4.8850   Max.   :21.980   Max.   :542.200   Max.   :0.031130
##  compactness_se      concavity_se     concave.points_se   symmetry_se
##  Min.   :0.002252   Min.   :0.00000   Min.   :0.000000   Min.   :0.007882
##  1st Qu.:0.013080   1st Qu.:0.01509   1st Qu.:0.007638   1st Qu.:0.015160
##  Median :0.020450   Median :0.02589   Median :0.010930   Median :0.018730
##  Mean   :0.025478   Mean   :0.03189   Mean   :0.011796   Mean   :0.020542
##  3rd Qu.:0.032450   3rd Qu.:0.04205   3rd Qu.:0.014710   3rd Qu.:0.023480
##  Max.   :0.135400   Max.   :0.39600   Max.   :0.052790   Max.   :0.078950
##  fractal_dimension_se  radius_worst   texture_worst   perimeter_worst
##  Min.   :0.0008948    Min.   : 7.93   Min.   :12.02   Min.   : 50.41
##  1st Qu.:0.0022480    1st Qu.:13.01   1st Qu.:21.08   1st Qu.: 84.11
##  Median :0.0031870    Median :14.97   Median :25.41   Median : 97.66
##  Mean   :0.0037949    Mean   :16.27   Mean   :25.68   Mean   :107.26
##  3rd Qu.:0.0045580    3rd Qu.:18.79   3rd Qu.:29.72   3rd Qu.:125.40
##  Max.   :0.0298400    Max.   :36.04   Max.   :49.54   Max.   :251.20
##    area_worst     smoothness_worst  compactness_worst concavity_worst
##  Min.   : 185.2   Min.   :0.07117   Min.   :0.02729   Min.   :0.0000
##  1st Qu.: 515.3   1st Qu.:0.11660   1st Qu.:0.14720   1st Qu.:0.1145
##  Median : 686.5   Median :0.13130   Median :0.21190   Median :0.2267
##  Mean   : 880.6   Mean   :0.13237   Mean   :0.25427   Mean   :0.2722
##  3rd Qu.:1084.0   3rd Qu.:0.14600   3rd Qu.:0.33910   3rd Qu.:0.3829
##  Max.   :4254.0   Max.   :0.22260   Max.   :1.05800   Max.   :1.2520
##  concave.points_worst symmetry_worst   fractal_dimension_worst
##  Min.   :0.00000      Min.   :0.1565   Min.   :0.05504
##  1st Qu.:0.06493      1st Qu.:0.2504   1st Qu.:0.07146
##  Median :0.09993      Median :0.2822   Median :0.08004
##  Mean   :0.11461      Mean   :0.2901   Mean   :0.08395
```

```
##  3rd Qu.:0.16140      3rd Qu.:0.3179   3rd Qu.:0.09208
##  Max.   :0.29100      Max.   :0.6638   Max.   :0.20750
```

```
# data set dimensions
cat("Dimensions of dataset:", dim(df))
```

```
## Dimensions of dataset: 569 31
```

```
# NA values
df_na_counts <- sum(is.na(df))
cat("NA Sum:", df_na_counts)
```

```
## NA Sum: 0
```

Distribution of Outcomes

```
# Calculate percentages
percentage_M <- sum(df$diagnosis == "M") / nrow(df) * 100
percentage_B <- sum(df$diagnosis == "B") / nrow(df) * 100

# Print the percentages
cat("Percentage of Malignant diagnosis:", percentage_M,"%\n")
```

```
## Percentage of Malignant diagnosis: 37.25835 %
```

```
cat("Percentage of Benign diagnosis:", percentage_B,"%\n")
```

```
## Percentage of Benign diagnosis: 62.74165 %
```

Exploring Possible Near Zero Variances

```
degeneratecols <- nearZeroVar(df)
degeneratecols
```

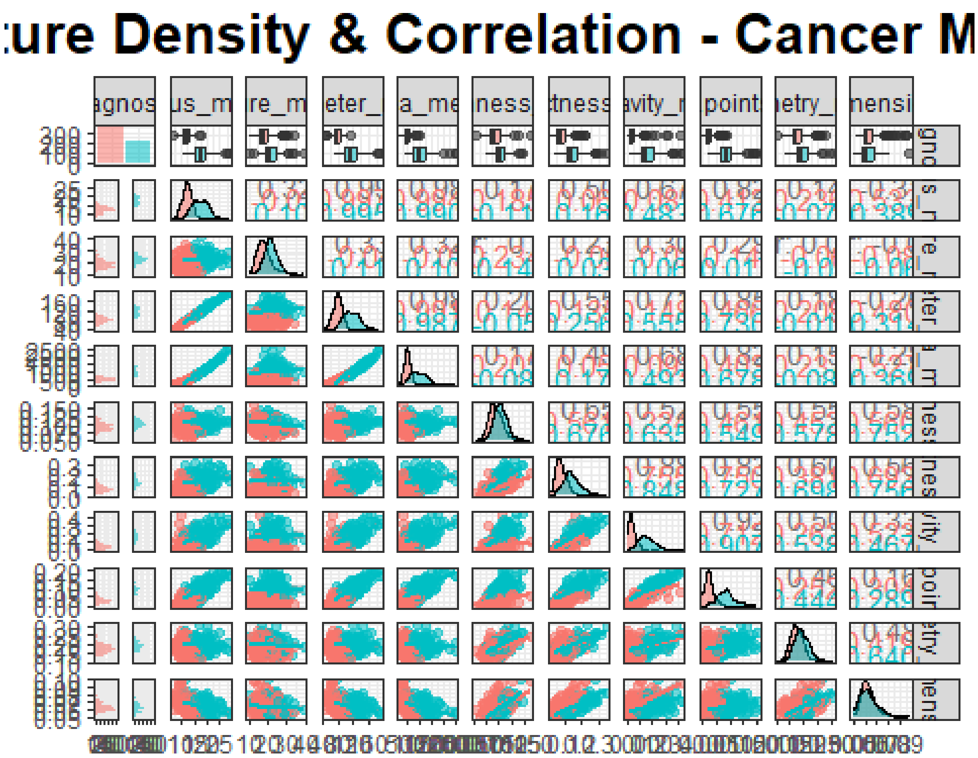
```
## integer(0)
```

There appears to be no degenerate variables.

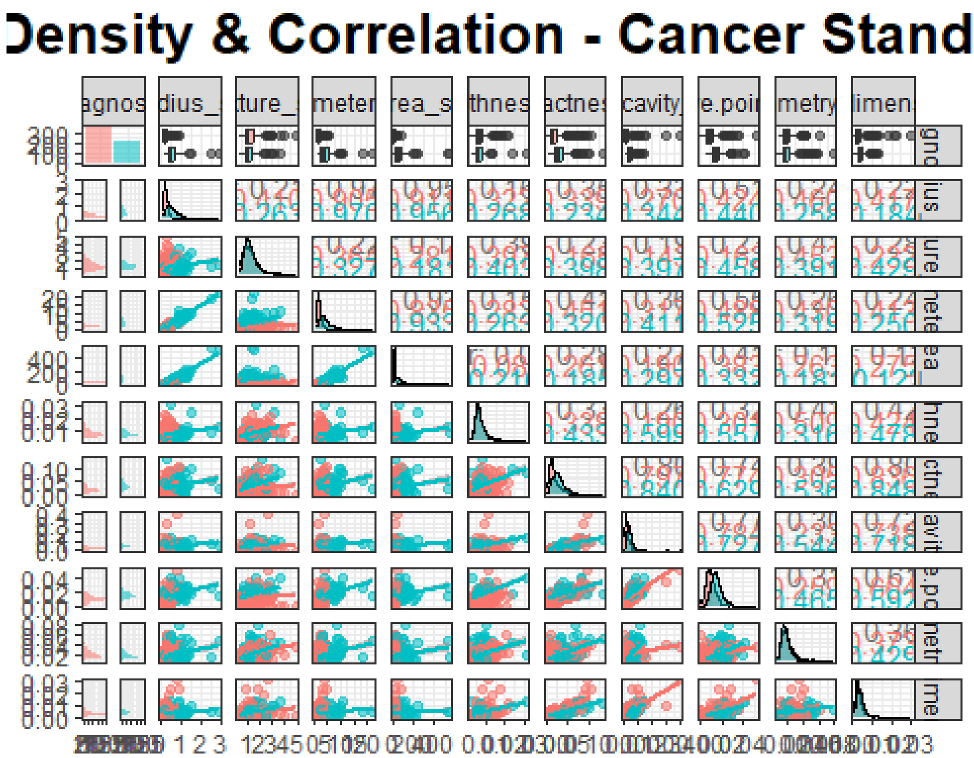
Splitting into groups to allow for easier visualizations

```
# Identify the columns containing "mean"
mean_columns <- grep("mean", names(df), value = TRUE)

# Identify the columns containing "se"
se_columns <- grep("se", names(df), value = TRUE)

# Identify the columns containing "worst"
worst_columns <- grep("worst", names(df), value = TRUE)

# Split the dataframe into three groups based on the keywords
df_mean <- df[, mean_columns]
df_se <- df[, se_columns]
df_worst <- df[, worst_columns]
```

```r
# Prepare the dataframe
df_clean <- df[, !(names(df) %in% c("X", "id"))]

# Create empty data frames
df_mean <- data.frame(diagnosis = df_clean$diagnosis)
df_se <- data.frame(diagnosis = df_clean$diagnosis)
df_worst <- data.frame(diagnosis = df_clean$diagnosis)

# Loop over variable names
variable_names <- colnames(df_clean)
for (variable_name in variable_names) {
  # Extract the part after the "_" delimiter
  variable_type <- sub(".+_([^_]+)$", "\\1", variable_name)
  # Assign the column to the appropriate data frame based on variable type
  if (variable_type == "mean") {
    df_mean[[variable_name]] <- df_clean[[variable_name]]
  } else if (variable_type == "se") {
    df_se[[variable_name]] <- df_clean[[variable_name]]
  } else if (variable_type == "worst") {
    df_worst[[variable_name]] <- df_clean[[variable_name]]
  }
}

### Analyze the Feature Density & Correlation between variables
# Visualization for the "Mean" Variables
ggpairs(df_mean, aes(color=diagnosis, alpha=0.75),
lower=list(continuous="smooth"))+ theme_bw()+
  labs(title="Feature Density & Correlation - Cancer
Means")+theme(plot.title=element_text(face='bold',color='black',hjust=0.5,siz
e=20))

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

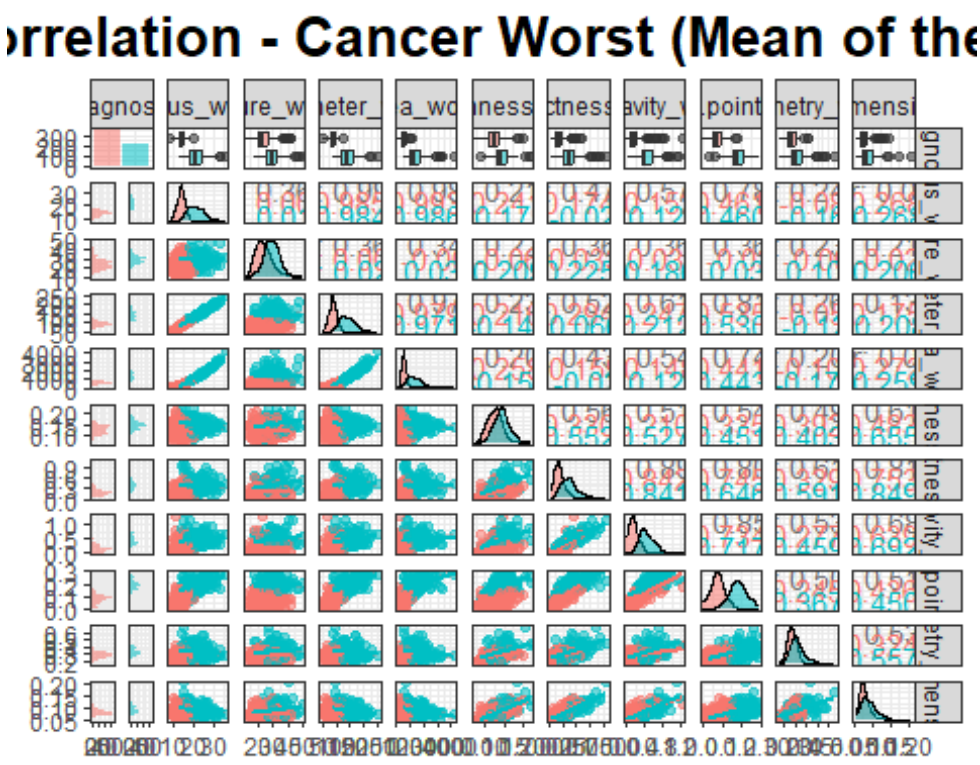



```r
# Visualization for the "Standard Error" Variables
ggpairs(df_se, aes(color=diagnosis, alpha=0.75),
lower=list(continuous="smooth"))+ theme_bw()+
  labs(title="Feature Density & Correlation - Cancer Standard
Error")+theme(plot.title=element_text(face='bold',color='black',hjust=0.5,siz
e=20))

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

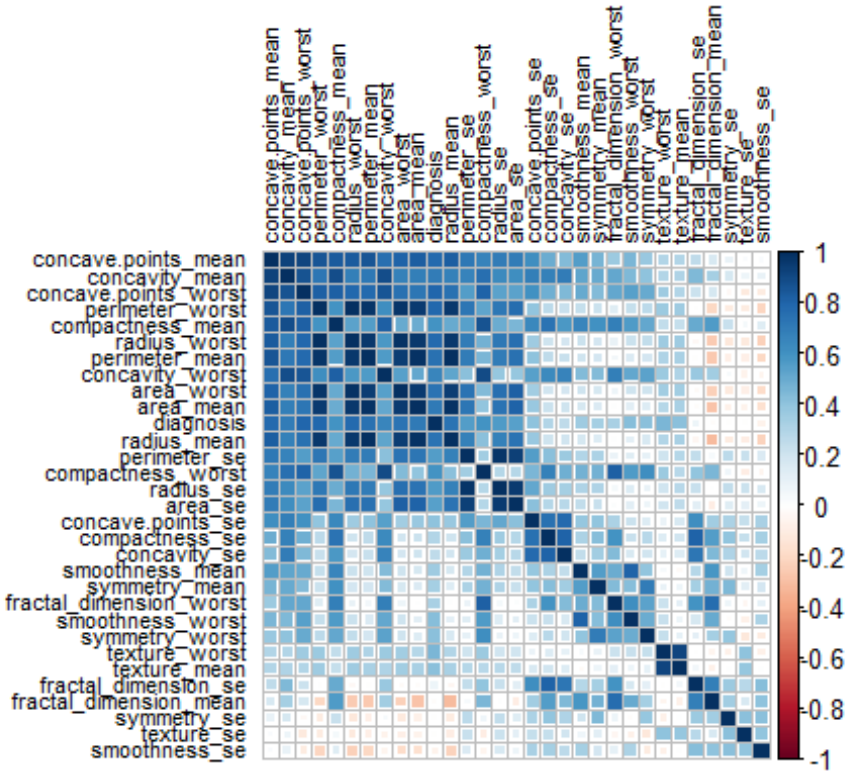



```r
# Visualization for the "Worst" Variables
ggpairs(df_worst, aes(color=diagnosis, alpha=0.75),
lower=list(continuous="smooth"))+ theme_bw()+
  labs(title="Feature Density & Correlation - Cancer Worst (Mean of the three
largest
values)")+theme(plot.title=element_text(face='bold',color='black',hjust=0.5,s
ize=20))
```

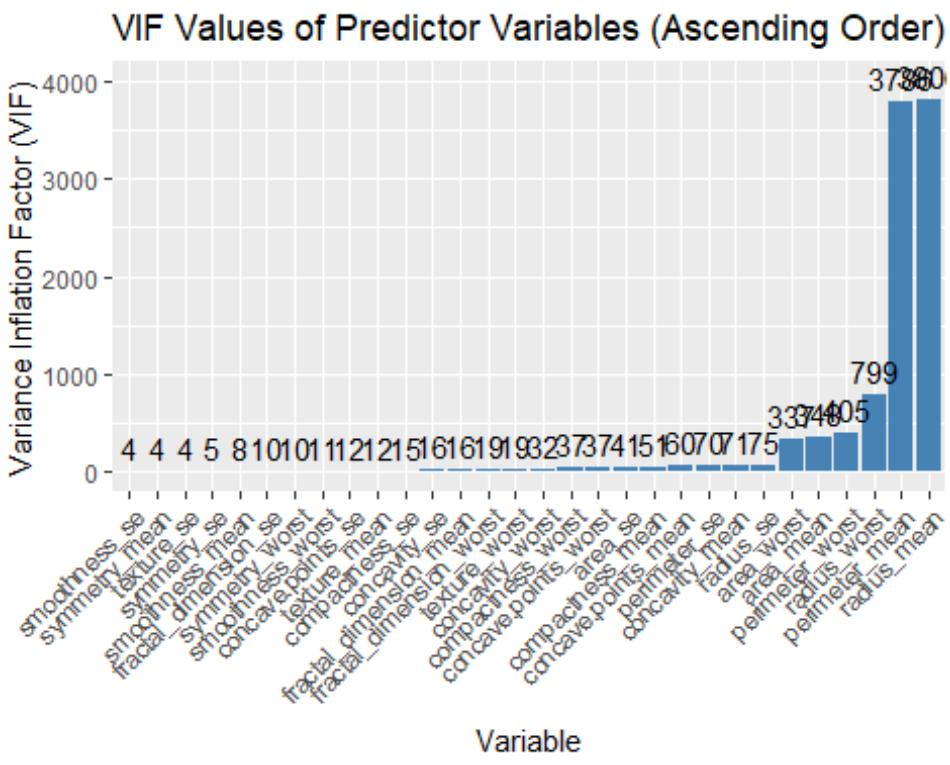
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



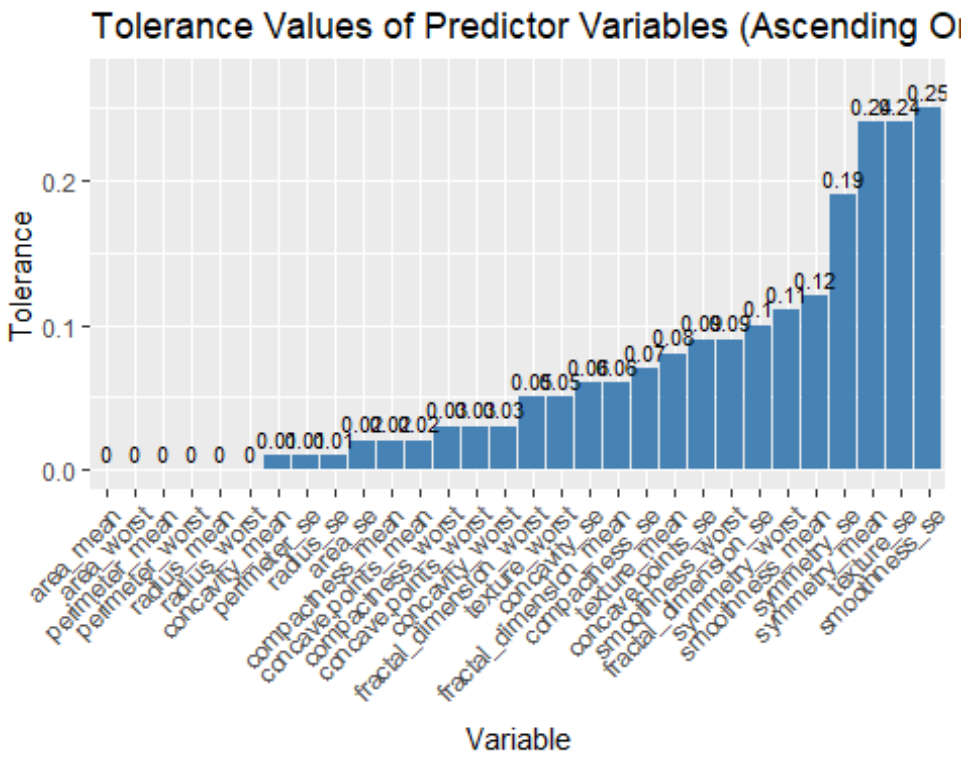

```r
# Pearson Correlation of all features
df_clean$diagnosis <- as.integer(factor(df_clean$diagnosis))-1
correlations <- cor(df_clean,method="pearson")
corrplot(correlations, number.cex = .9, method = "square",
         hclust.method = "ward", order = "FPC",
         type = "full", tl.cex=0.7,tl.col = "black")
```



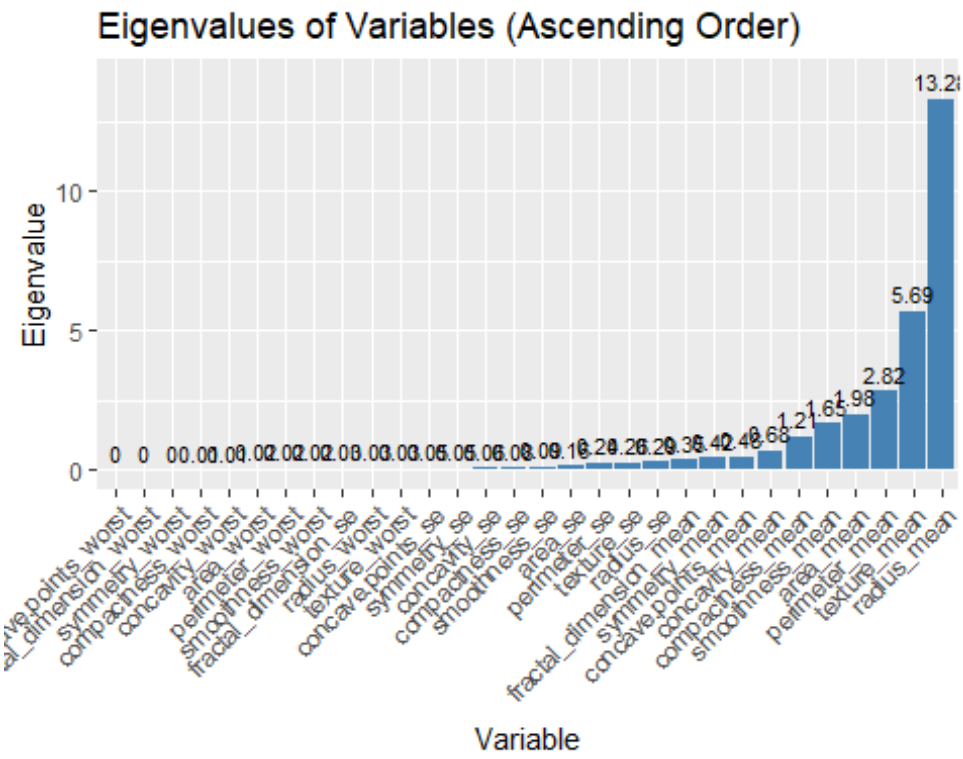

```r
# Calculate the VIF values (using linear model) for the predictor variables to look for multicollinearity
# VIF below 5 are generally considered acceptable, 5-10 suggest moderate multicollinearity, and above 10 show high multicollinearity
lm_model <- lm(diagnosis ~ ., data = df_clean)
vif_values <- vif(lm_model)
vif_table <- data.frame(Variable = names(vif_values), VIF = vif_values)
vif_table <- vif_table[order(vif_table$VIF), ]
vif_table$VIF <- round(vif_table$VIF)
y_axis_limit <- 4000
vif_plot <- ggplot(vif_table, aes(x = reorder(Variable, VIF), y = VIF)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  geom_text(aes(label = VIF), vjust = -0.5) +  # Add integer value labels
  labs(x = "Variable", y = "Variance Inflation Factor (VIF)") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("VIF Values of Predictor Variables (Ascending Order)") +
  ylim(0, y_axis_limit)  # Set the y-axis limits
print(vif_plot)
```



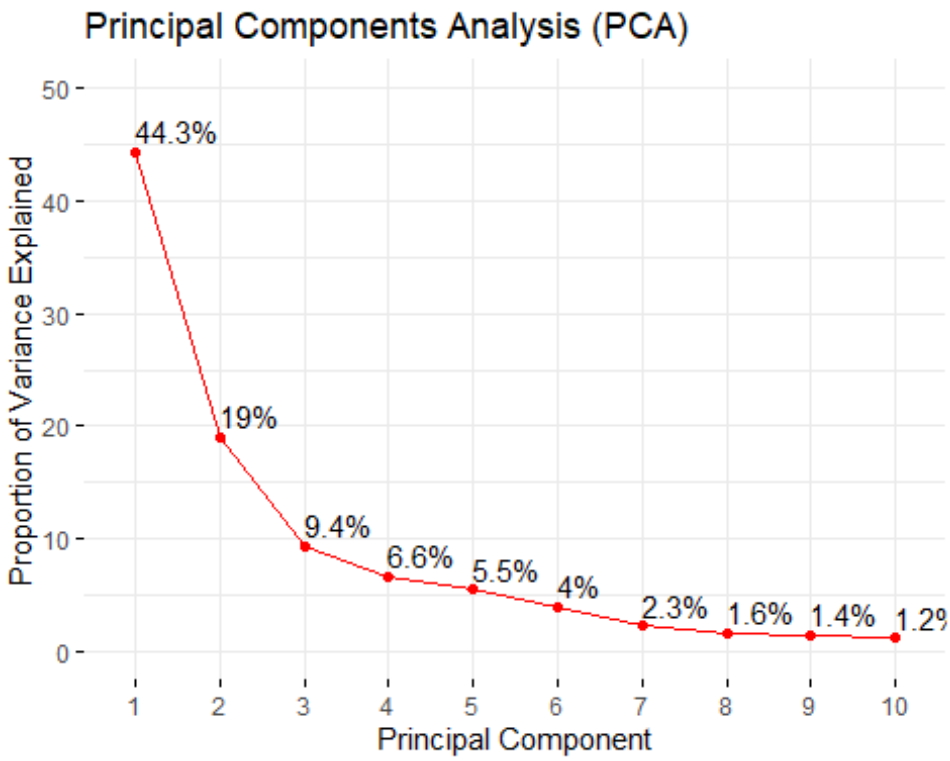

```
### Calculate the tolerance (reciprocal of VIF) for the predictor variables
to look for multicollinearity
# Generally, a tolerance value less than 0.1 or 0.2 is often considered
indicative of multicollinearity.
tolerance <- 1 / vif_values
tolerance_table <- data.frame(Variable = names(tolerance), Tolerance =
tolerance)
tolerance_table <- tolerance_table[order(tolerance_table$Tolerance), ]
tolerance_table$Tolerance <- round(tolerance_table$Tolerance, 2)
y_axis_limit <- 0.27
tolerance_plot <- ggplot(tolerance_table, aes(x = reorder(Variable,
Tolerance), y = Tolerance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  geom_text(aes(label = Tolerance), vjust = -0.5, hjust = 0.5, color =
"black", size = 3) +  # Add value labels
  labs(x = "Variable", y = "Tolerance") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("Tolerance Values of Predictor Variables (Ascending Order)") +
  ylim(0, y_axis_limit)  # Set the y-axis limits
tolerance_plot
```



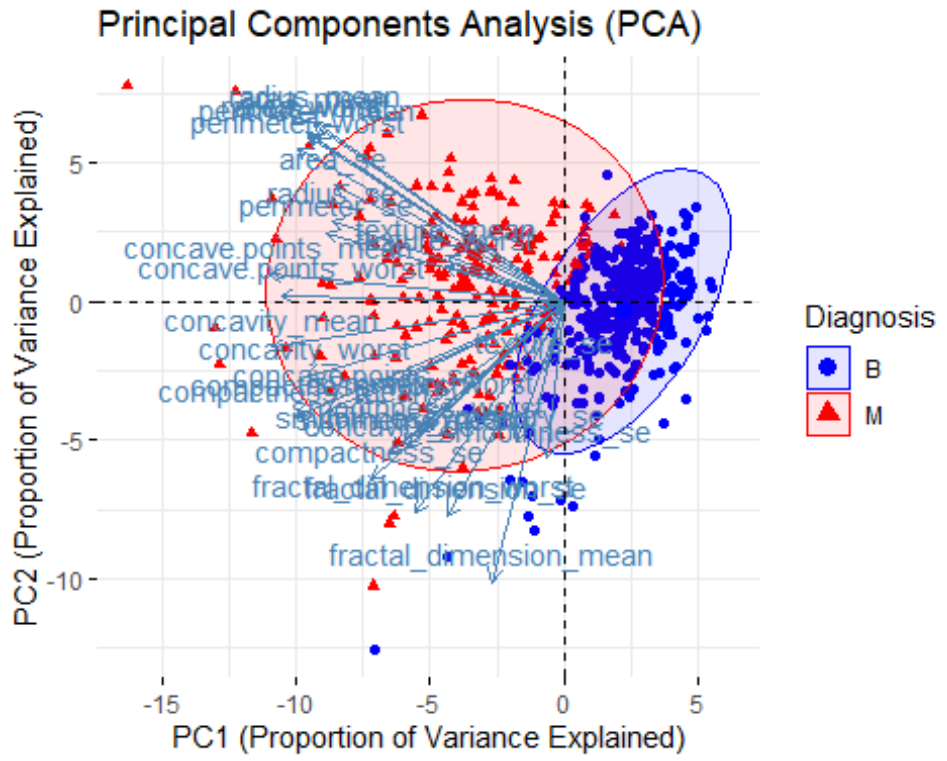

```r
# Calculate the eigenvalues for the predictor variables to look for
multicollinearity
# Small eigenvalues suggest potential multicollinearity issues
correlations <- cor(df_clean[-1], method = "pearson")
eigenvalues <- eigen(correlations)$values
eigen_table <- data.frame(Variable = colnames(correlations), Eigenvalue =
eigenvalues)
eigen_table <- eigen_table[eigen_table$Variable != "diagnosis", ]
eigen_table <- eigen_table[order(eigen_table$Eigenvalue), ]
# Round the Eigenvalue values for better aesthetics
eigen_table$Eigenvalue <- round(eigen_table$Eigenvalue, 2)
y_axis_limit <- 14
eigen_plot <- ggplot(eigen_table, aes(x = reorder(Variable, Eigenvalue), y =
Eigenvalue)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  geom_text(aes(label = Eigenvalue), vjust = -0.5, hjust = 0.5, color =
"black", size = 3) +  # Add value labels
  labs(x = "Variable", y = "Eigenvalue") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("Eigenvalues of Variables (Ascending Order)") +
  ylim(0, y_axis_limit)  # Set the y-axis limits
eigen_plot
```




```
### Principal Components Analysis (PCA) transform
raw.data <-
read.csv('C:/MIDS/ADS-503_Applied_Predictive_Modeling/ADS_503_team_2_final_pr
oject/data/breast_cancer_FNA_data.csv')
var_only <- df_clean[, !(names(df_clean) %in% c("diagnosis"))] # predictor
variables only
diagnosis <- raw.data[, 2] # target variable only
var_pca <- prcomp(var_only, center = TRUE, scale. = TRUE)
fviz_eig(var_pca, addlabels = TRUE, ylim = c(0, 50),
         title = "Principal Components Analysis (PCA)",
         subtitle = NULL,
         xlab = "Principal Component",
         ylab = "Proportion of Variance Explained",
         geom = "line",
         linecolor = "red",
         pointsize = 2,
         pointshape = 21,
         pointfill = "white",
         pointcolor = "red",
         legend.title = "Principal Components",
         legend.position = "right")
```



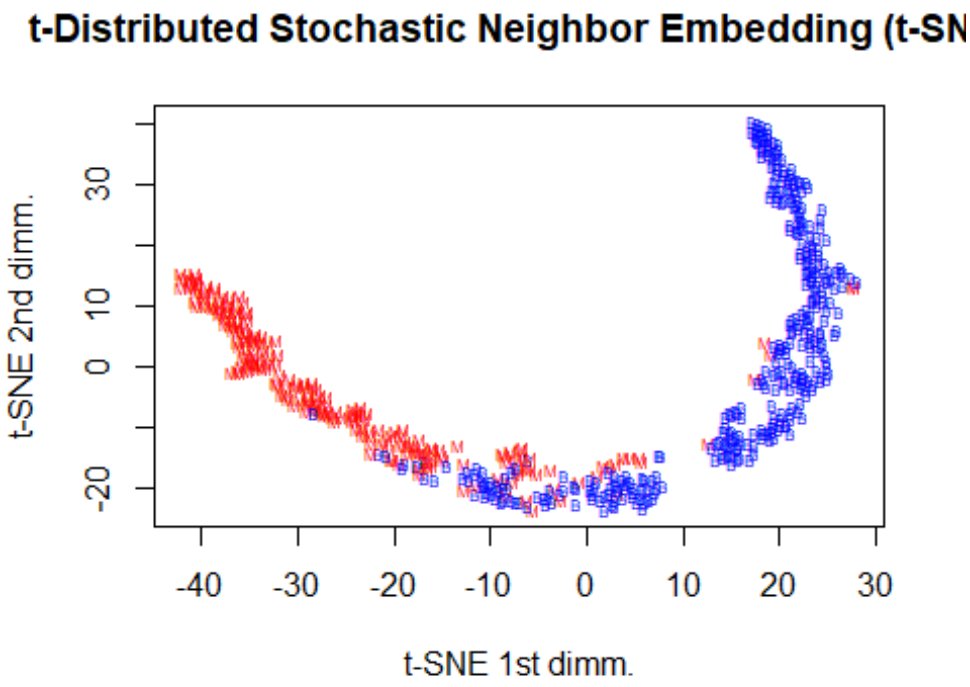

```r
# Adjust plot margins
par(mar = c(5, 5, 4, 2) + 0.1)

# Create PCA biplot with customized aesthetics
pca_biplot <- fviz_pca_biplot(var_pca,
                              geom.ind = "point",
                              col.ind = diagnosis,
                              palette = c("blue", "red"),
                              addEllipses = TRUE,
                              axes.linetype = "dashed",
                              title = "Principal Components Analysis (PCA)",
                              xlab = "PC1 (Proportion of Variance Explained)",
                              ylab = "PC2 (Proportion of Variance Explained)",
                              legend.title = "Diagnosis",
                              legend.position = "right",
                              legend.shape = "circle",
                              legend.label = c("Benign", "Malignant"))
pca_biplot
```




```
### t-SNE transform for dimensionality reduction

colors <- c("red", "blue")
names(colors) = unique(diagnosis)
set.seed(0)
tsne <- Rtsne(var_only, dims=2, perplexity=30,
              verbose=TRUE, pca=TRUE,
              theta=0.01, max_iter=1000)

## Performing PCA
## Read the 569 x 30 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.010000
## Computing input similarities...
## Building tree...
## Done in 0.13 seconds (sparsity = 0.189300)!
## Learning embedding...
## Iteration 50: error is 52.793956 (50 iterations in 0.54 seconds)
## Iteration 100: error is 48.892861 (50 iterations in 0.51 seconds)
## Iteration 150: error is 48.176855 (50 iterations in 0.48 seconds)
## Iteration 200: error is 47.908522 (50 iterations in 0.49 seconds)
## Iteration 250: error is 47.787718 (50 iterations in 0.49 seconds)
## Iteration 300: error is 0.384715 (50 iterations in 0.56 seconds)
## Iteration 350: error is 0.296547 (50 iterations in 0.63 seconds)
## Iteration 400: error is 0.272270 (50 iterations in 0.59 seconds)
## Iteration 450: error is 0.263214 (50 iterations in 0.59 seconds)
```

```
## Iteration 500: error is 0.259213 (50 iterations in 0.51 seconds)
## Iteration 550: error is 0.257035 (50 iterations in 0.48 seconds)
## Iteration 600: error is 0.255582 (50 iterations in 0.47 seconds)
## Iteration 650: error is 0.254480 (50 iterations in 0.49 seconds)
## Iteration 700: error is 0.253590 (50 iterations in 0.47 seconds)
## Iteration 750: error is 0.252851 (50 iterations in 0.49 seconds)
## Iteration 800: error is 0.252216 (50 iterations in 0.50 seconds)
## Iteration 850: error is 0.251664 (50 iterations in 0.54 seconds)
## Iteration 900: error is 0.251177 (50 iterations in 0.51 seconds)
## Iteration 950: error is 0.250739 (50 iterations in 0.52 seconds)
## Iteration 1000: error is 0.250342 (50 iterations in 0.49 seconds)
## Fitting performed in 10.35 seconds.
```

```
plot(tsne$Y, t='n', main="t-Distributed Stochastic Neighbor Embedding
(t-SNE)",
     xlab="t-SNE 1st dimm.", ylab="t-SNE 2nd dimm.")
text(tsne$Y, labels=diagnosis, cex=0.5, col=colors[diagnosis])
```



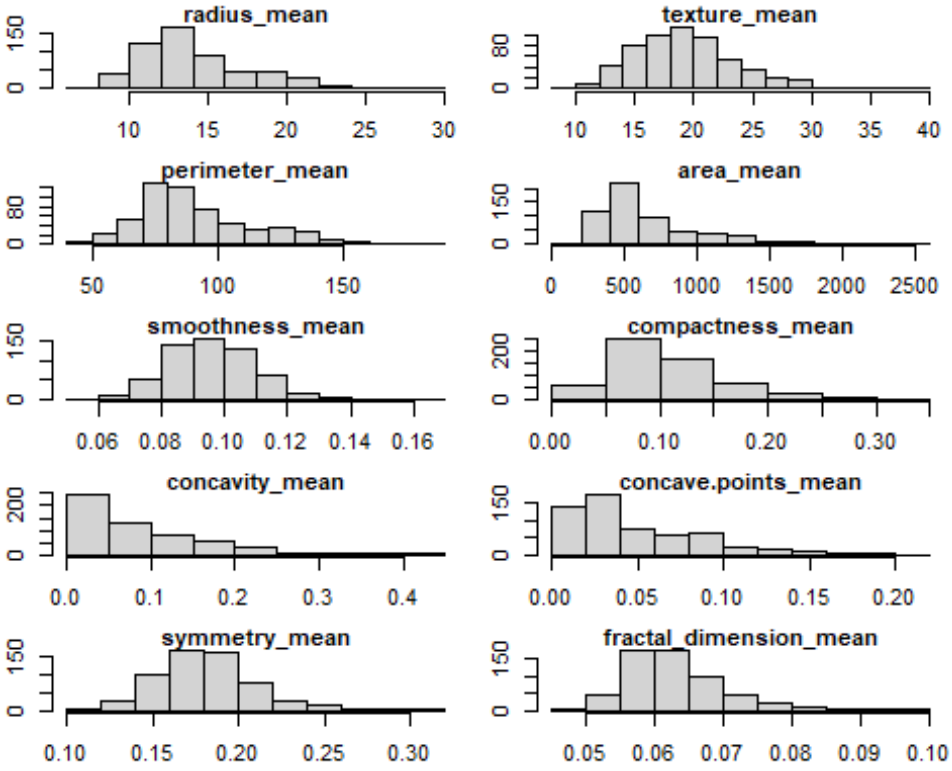
df_var data frame for visualizing just predictors

```
remove <- c("diagnosis")
df_var <- df[, !(colnames(df) %in% remove)]
```

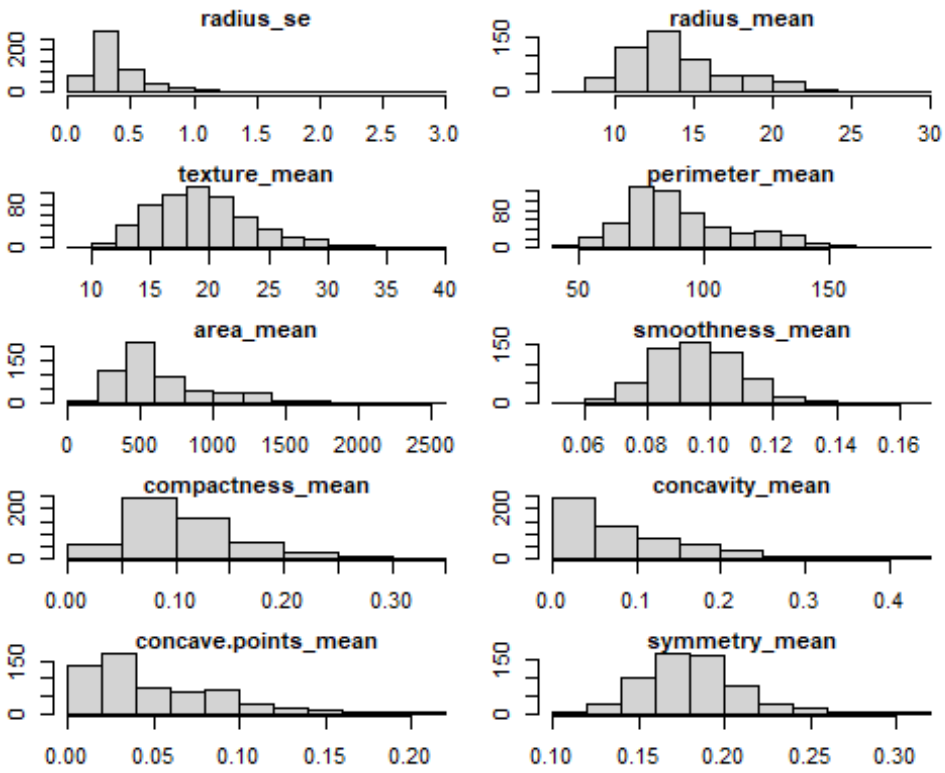
Histograms of all predictor variables

```
par(mfrow = c(5, 2))
par(mar = c(2.5, 2, 1, 1))
par(cex.main = 1)
```

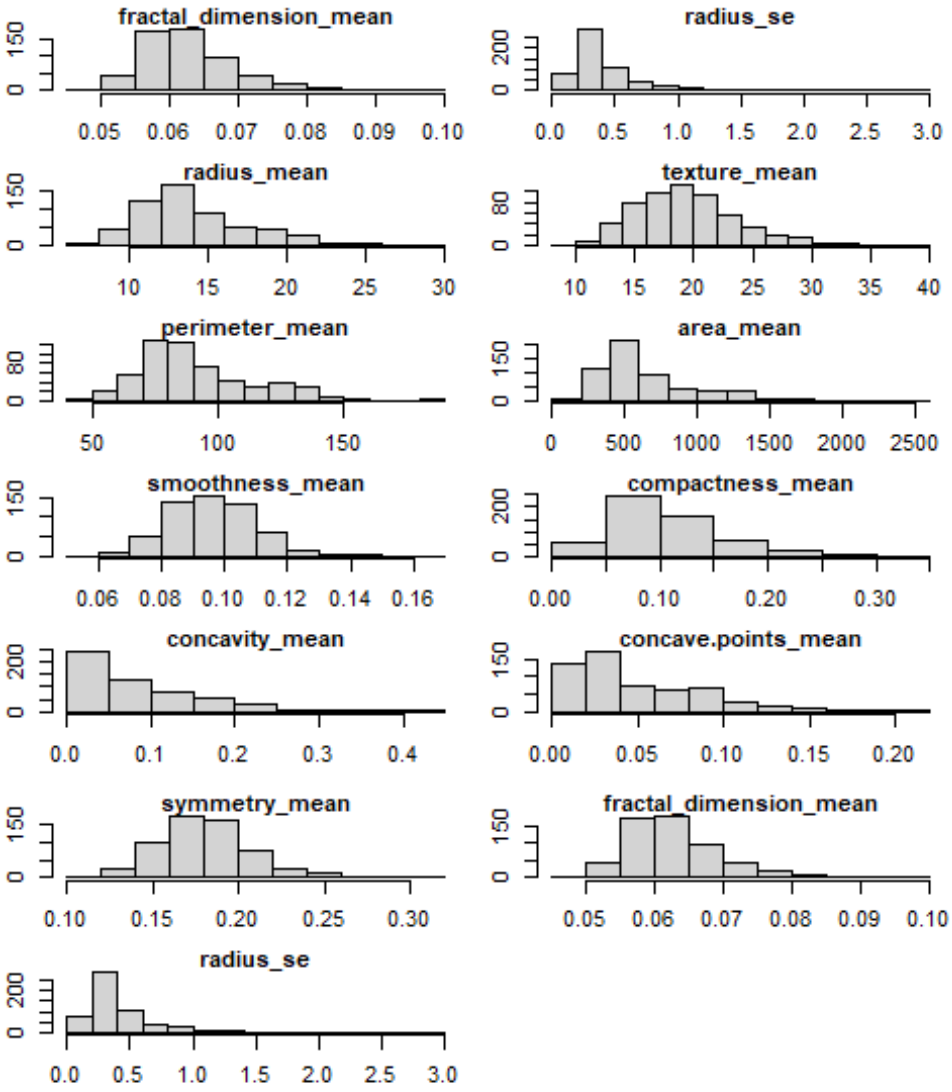
```r
for (i in 1:ncol(df_mean)) {
  hist(df_var[, i], xlab = names(df_var)[i], main = paste(names(df_var)[i]))
}
```




```r
for (i in 1:ncol(df_se)) {
  hist(df_var[, i], xlab = names(df_var)[i], main = paste(names(df_var)[i]))
}
```




```
for (i in 1:ncol(df_worst)) {
  hist(df_var[, i], xlab = names(df_var)[i], main = paste(names(df_var)[i]))
}
```



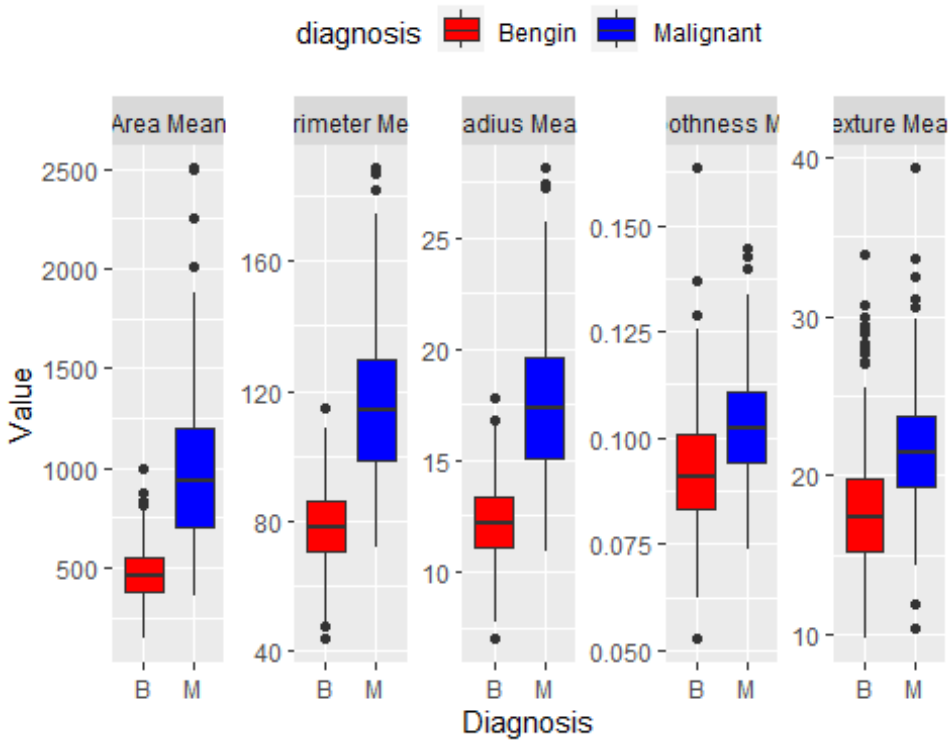

```
# build a boxplot to identity the outliers that were produced in the summary
statistics
# split the data into groups of 5 and create 6 graphs for the plots since
```

```
there are 30 features and 1 target
# data will be split into variables bx_1, bx_2, bx_3, bx_4, bx_5 for boxplot
purposes
colors <- c('red', 'blue')
bx_1 <- df[, 2:6]
bx_2 <- df[, 7:12]
bx_3 <- df[, 13:18]
bx_4 <- df[, 19:24]
bx_5 <- df[, 25:31]
```

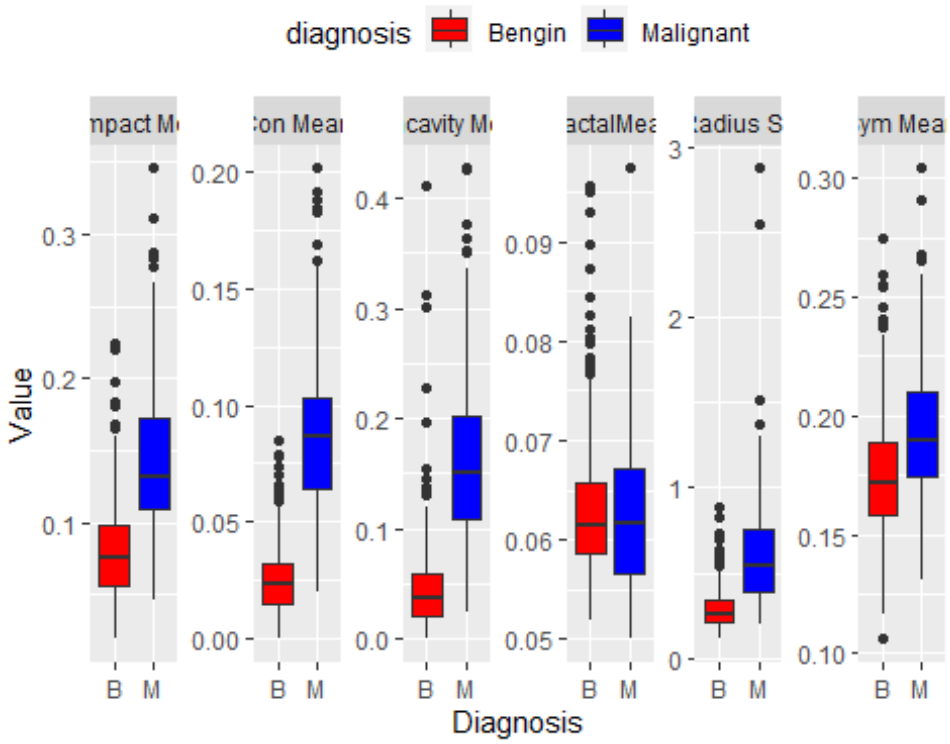
All of these predictors have outliers however the predictor with the most are texture_mean, area_mean ad smoothness mean.

For bx_2 all of these also have outliers and I would say they all have some pretty significant outliers.

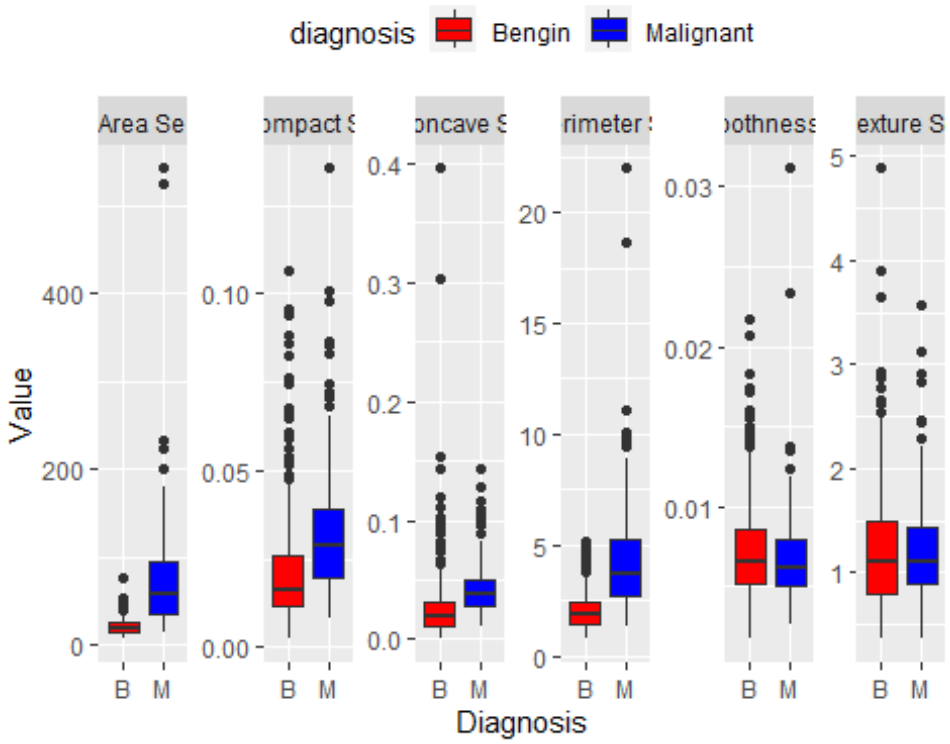
```
# Create a data frame for plotting
plot_data_1 <- data.frame(
  value = c(bx_1$radius_mean, bx_1$texture_mean, bx_1$perimeter_mean,
bx_1$smoothness_mean, bx_1$area_mean),
  diagnosis = rep(df$diagnosis, 5),
  feature = rep(c("Radius Mean", "Texture Mean", "Perimeter Mean",
"Smoothness Mean", "Area Mean"), each = nrow(df))
)

# Create the boxplot using ggplot2
ggplot(plot_data_1, aes(x = diagnosis, y = value, fill = diagnosis)) +
  geom_boxplot() +
  facet_wrap(~ feature, scales = "free_y", nrow = 1) +
  scale_fill_manual(values = colors, labels = c('Bengin', 'Malignant')) +
  xlab("Diagnosis") +
  ylab("Value") +
  theme(legend.position = "top")
```



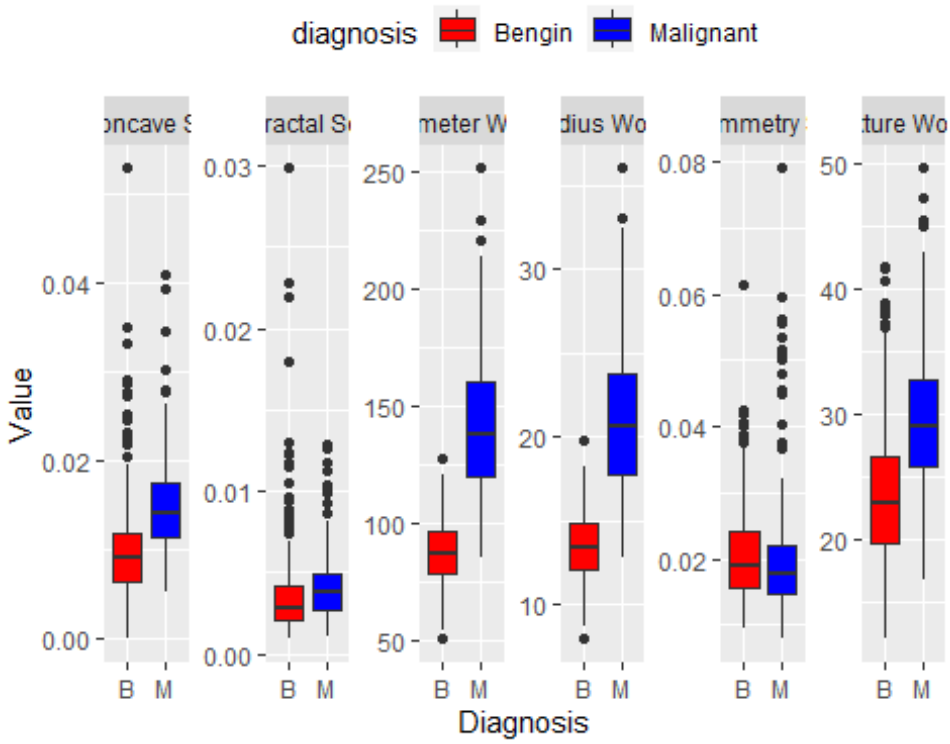

```r
# Create a data frame for plotting
plot_data_2 <- data.frame(
  value = c(bx_2$fractal_dimension_mean,bx_2$symmetry_mean, bx_2$radius_se,
bx_2$concave.points_mean, bx_2$concavity_mean,bx_2$compactness_mean  ),
  diagnosis = rep(df$diagnosis, 6),
  feature = rep(c("FractalMean", "Sym Mean", "Radius Se", "Con Mean",
"Concavity Mean", 'Compact Mean'), each = nrow(df))
)

# Create the boxplot using ggplot2
ggplot(plot_data_2, aes(x = diagnosis, y = value, fill = diagnosis)) +
  geom_boxplot() +
  facet_wrap(~ feature, scales = "free_y", nrow = 1) +
  scale_fill_manual(values = colors, labels = c('Bengin', 'Malignant')) +
  xlab("Diagnosis") +
  ylab("Value") +
  theme(legend.position = "top")
```



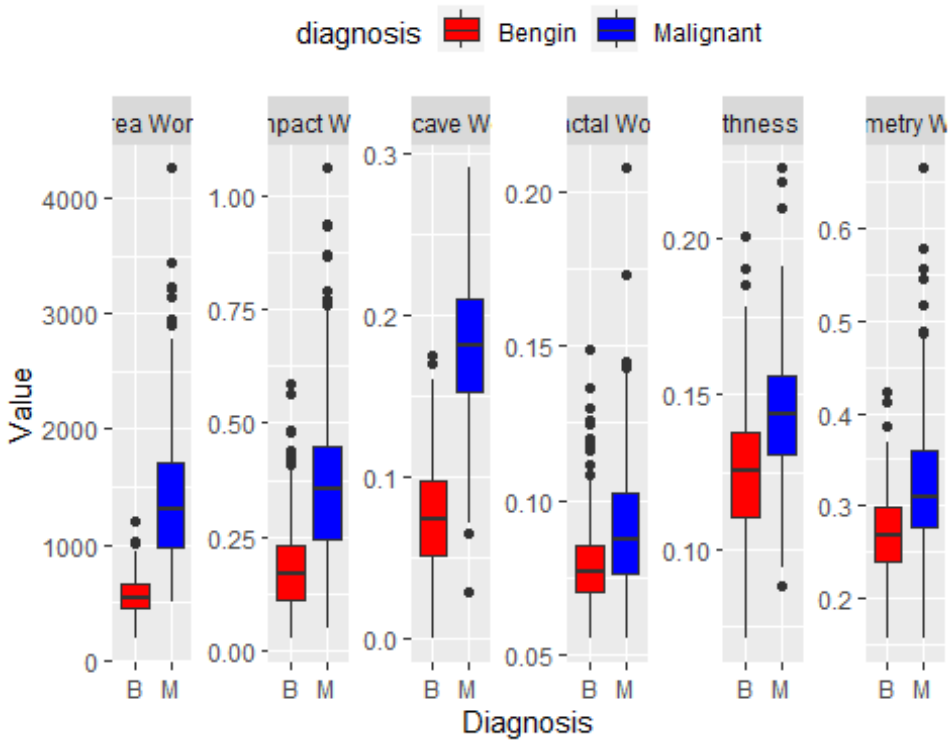

```r
# Create a data frame for plotting
plot_data_3 <- data.frame(
  value = c(bx_3$texture_se,bx_3$perimeter_se, bx_3$area_se,
bx_3$smoothness_se, bx_3$compactness_se, bx_3$concavity_se),
  diagnosis = rep(df$diagnosis, 6),
  feature = rep(c("Texture Se", "Perimeter Se", "Area Se", "Smoothness Se",
"Compact Se", 'Concave Se'), each = nrow(df))
)

# Create the boxplot using ggplot2
ggplot(plot_data_3, aes(x = diagnosis, y = value, fill = diagnosis)) +
  geom_boxplot() +
  facet_wrap(~ feature, scales = "free_y", nrow = 1) +
  scale_fill_manual(values = colors, labels = c('Bengin', 'Malignant')) +
  xlab("Diagnosis") +
  ylab("Value") +
  theme(legend.position = "top")
```




```r
# Create a data frame for plotting
plot_data_4 <- data.frame(
  value = c(bx_4$concave.points_se,bx_4$symmetry_se,
bx_4$fractal_dimension_se, bx_4$radius_worst, bx_4$texture_worst,
bx_4$perimeter_worst),
  diagnosis = rep(df$diagnosis, 6),
  feature = rep(c("Concave Se", "Symmetry Se", "Fractal Se", "Radius Worst",
"Texture Worst", 'Perimeter Worst'), each = nrow(df))
)

# Create the boxplot
ggplot(plot_data_4, aes(x = diagnosis, y = value, fill = diagnosis)) +
  geom_boxplot() +
  facet_wrap(~ feature, scales = "free_y", nrow = 1) +
  scale_fill_manual(values = colors, labels = c('Bengin', 'Malignant')) +
  xlab("Diagnosis") +
  ylab("Value") +
  theme(legend.position = "top")
```



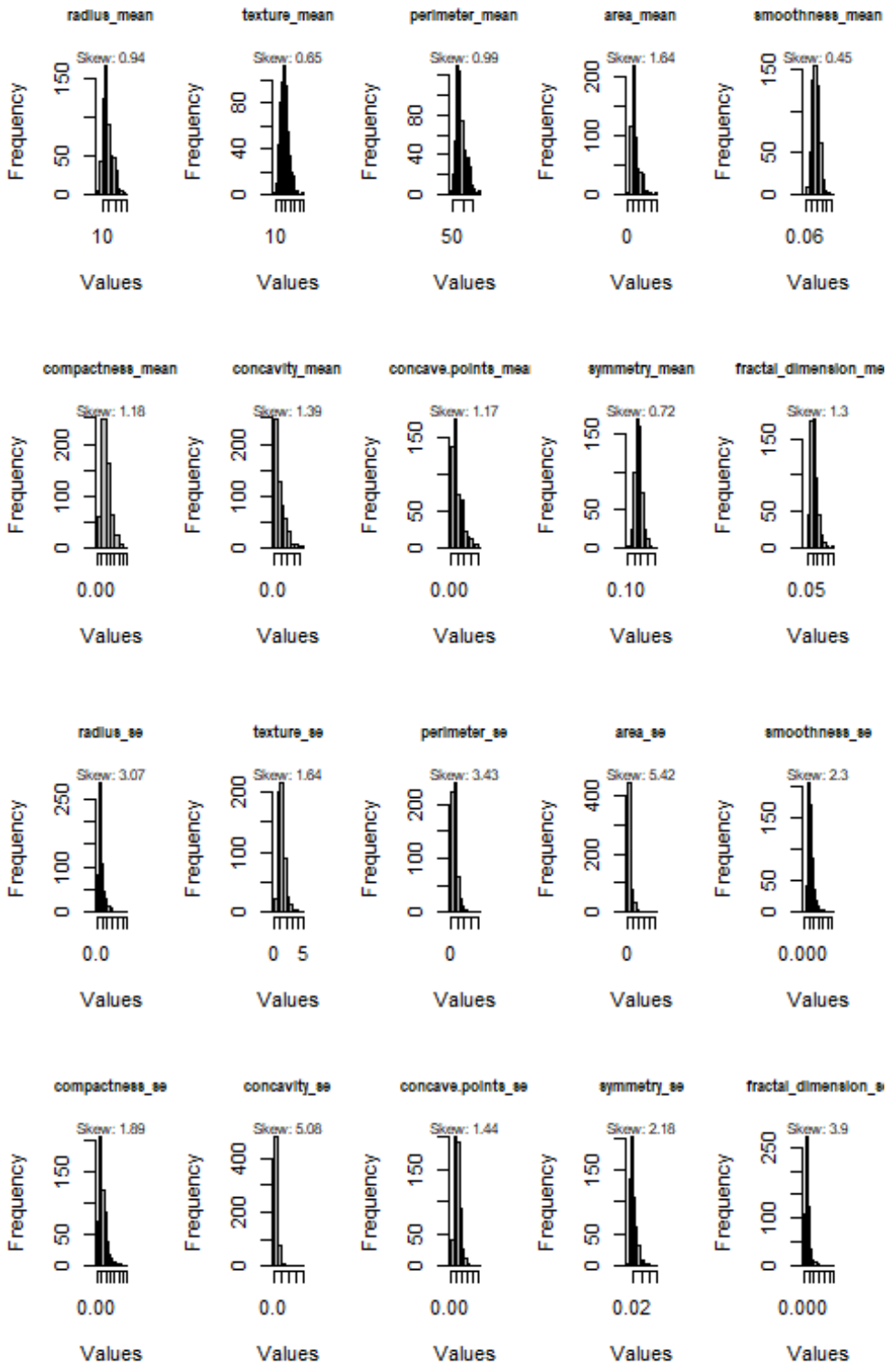

```r
# Create a data frame for plotting
plot_data_5 <- data.frame(
  value = c(bx_5$area_worst ,bx_5$smoothness_worst, bx_5$compactness_worst,
bx_5$concave.points_worst, bx_5$symmetry_worst,
bx_5$fractal_dimension_worst),
  diagnosis = rep(df$diagnosis, 6),
  feature = rep(c("Area Worst", "Smoothness Worst", "Compact Worst", "Concave
Worst", "symmetry Worst", 'Fractal Worst'), each = nrow(df))
)

# Create the boxplot
ggplot(plot_data_5, aes(x = diagnosis, y = value, fill = diagnosis)) +
  geom_boxplot() +
  facet_wrap(~ feature, scales = "free_y", nrow = 1) +
  scale_fill_manual(values = colors, labels = c('Bengin', 'Malignant')) +
  xlab("Diagnosis") +
  ylab("Value") +
  theme(legend.position = "top")
```

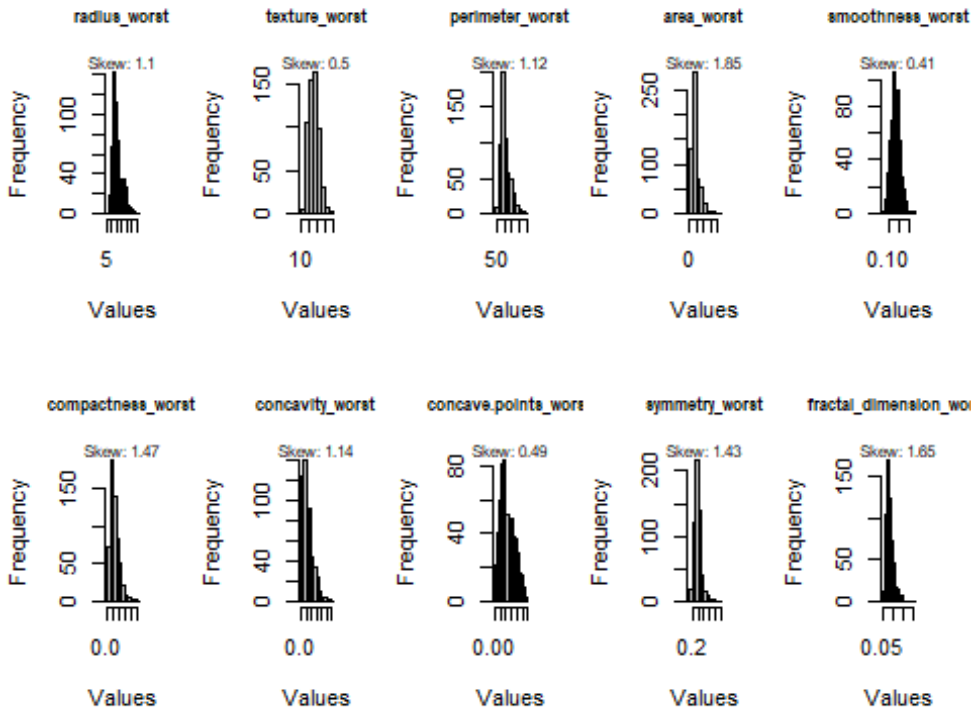


Skew

```
# List of group names
group_names <- c("mean", "se", "worst")

# Set the main title font size
par(cex.main = 0.8)

# Loop through each group
for (group_name in group_names) {
  # Subset the columns based on group name
  group_columns <- grep(group_name, names(df_var), value = TRUE)

  # Calculate skewness and create plots for each column in the group
  num_plots <- length(group_columns)
  num_rows <- ceiling(num_plots / 5)
  num_cols <- min(num_plots, 5)

  # Set the plotting layout for the current group
  par(mfrow = c(num_rows, num_cols))

  # Loop through each column in the group
  for (i in 1:num_plots) {
    col_name <- group_columns[i]
    skewness <- skewness(df_var[[col_name]])

    # Create a histogram to visualize skewness
```

```
    hist(df_var[[col_name]], main = col_name,
         xlab = "Values", ylab = "Frequency",
         col = "gray", border = "black")

    # Add skewness value to the plot
    mtext(paste("Skew:", round(skewness, 2)), side = 3, line = -.25, cex =
0.5, font = 1)
  }

  # Reset the plotting layout
  par(mfrow = c(1, 1))
}
```


radius_mean
Skew: 0.94
texture_mean
Skew: 0.65
perimeter_mean
Skew: 0.99
area_mean
Skew: 1.64
smoothness_mean
Skew: 0.45
compactness_mean
Skew: 1.18
concavity_mean
Skew: 1.39
concave.points_mea
Skew: 1.17
symmetry_mean
Skew: 0.72
fractal_dimension_me
Skew: 1.3
radius_se
Skew: 3.07
texture_se
Skew: 1.64
perimeter_se
Skew: 3.43
area_se
Skew: 5.42
smoothness_se
Skew: 2.3
compactness_se
Skew: 1.89
concavity_se
Skew: 5.08
concave.points_se
Skew: 1.44
symmetry_se
Skew: 2.18
fractal_dimension_s
Skew: 3.9
Frequency
Values



Skew in dataframe format for easy reference

```
# List of group names
group_names <- c("mean", "se", "worst")

# Create an empty list to store the skewness values
skewness_list <- list()

# Loop through each group
for (group_name in group_names) {
  # Subset the columns based on group name
  group_columns <- grep(group_name, names(df_var), value = TRUE)

  # Loop through each column in the group
  for (col_name in group_columns) {
    skewness <- skewness(df_var[[col_name]])

    # Print skewness value
    cat("Skewness for", col_name, ":", skewness, "\n")

    # Store the skewness value in the list
    skewness_list[[col_name]] <- skewness
  }
}
```

```
## Skewness for radius_mean : 0.9374168
## Skewness for texture_mean : 0.6470241
```

```
## Skewness for perimeter_mean : 0.9854334
## Skewness for area_mean : 1.637065
## Skewness for smoothness_mean : 0.4539207
## Skewness for compactness_mean : 1.183856
## Skewness for concavity_mean : 1.393801
## Skewness for concave.points_mean : 1.165012
## Skewness for symmetry_mean : 0.7217877
## Skewness for fractal_dimension_mean : 1.297619
## Skewness for radius_se : 3.072347
## Skewness for texture_se : 1.637773
## Skewness for perimeter_se : 3.42548
## Skewness for area_se : 5.4185
## Skewness for smoothness_se : 2.302262
## Skewness for compactness_se : 1.892203
## Skewness for concavity_se : 5.08355
## Skewness for concave.points_se : 1.43707
## Skewness for symmetry_se : 2.183573
## Skewness for fractal_dimension_se : 3.903304
## Skewness for radius_worst : 1.097306
## Skewness for texture_worst : 0.495697
## Skewness for perimeter_worst : 1.122223
## Skewness for area_worst : 1.849581
## Skewness for smoothness_worst : 0.4132383
## Skewness for compactness_worst : 1.465795
## Skewness for concavity_worst : 1.144179
## Skewness for concave.points_worst : 0.4900213
## Skewness for symmetry_worst : 1.426376
## Skewness for fractal_dimension_worst : 1.653824
```

```
# Convert the skewness list to a dataframe
skewness_df <- data.frame(Skewness = unlist(skewness_list))

# Print the skewness dataframe
print(skewness_df)
```

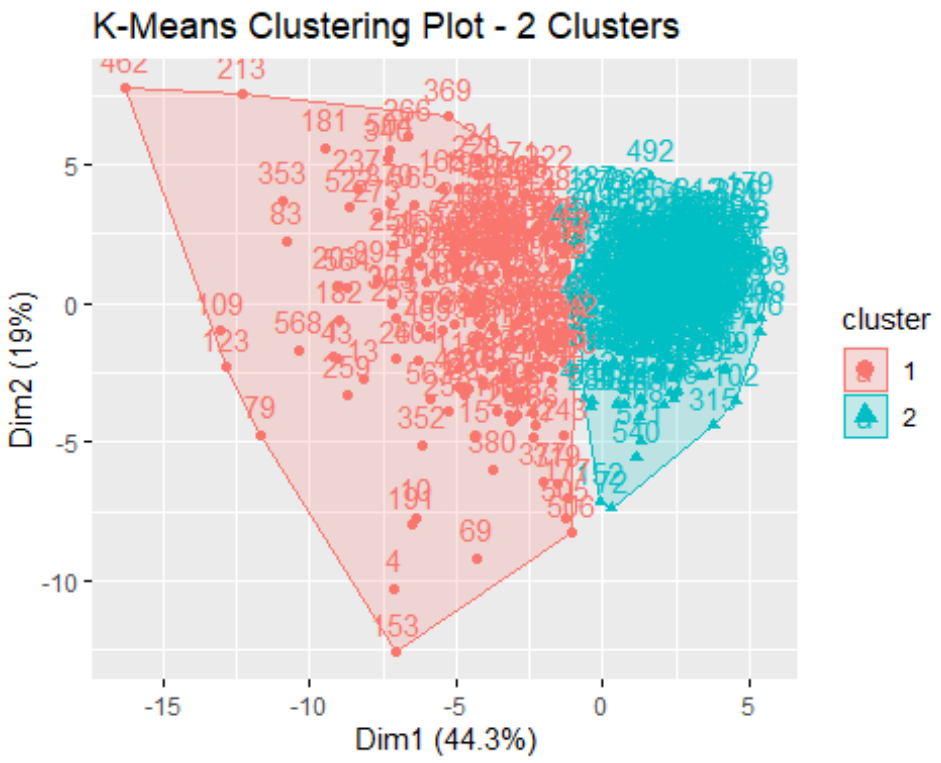
```
##                          Skewness
## radius_mean             0.9374168
## texture_mean            0.6470241
## perimeter_mean          0.9854334
## area_mean               1.6370654
## smoothness_mean         0.4539207
## compactness_mean        1.1838556
## concavity_mean          1.3938008
## concave.points_mean     1.1650124
## symmetry_mean           0.7217877
## fractal_dimension_mean  1.2976191
## radius_se               3.0723468
## texture_se              1.6377733
## perimeter_se            3.4254803
## area_se                 5.4185001
```

```
## smoothness_se           2.3022616
## compactness_se          1.8922032
## concavity_se            5.0835502
## concave.points_se       1.4370701
## symmetry_se             2.1835728
## fractal_dimension_se    3.9033041
## radius_worst            1.0973059
## texture_worst           0.4956970
## perimeter_worst         1.1222227
## area_worst              1.8495814
## smoothness_worst        0.4132383
## compactness_worst       1.4657948
## concavity_worst         1.1441794
## concave.points_worst    0.4900213
## symmetry_worst          1.4263764
## fractal_dimension_worst 1.6538237
```

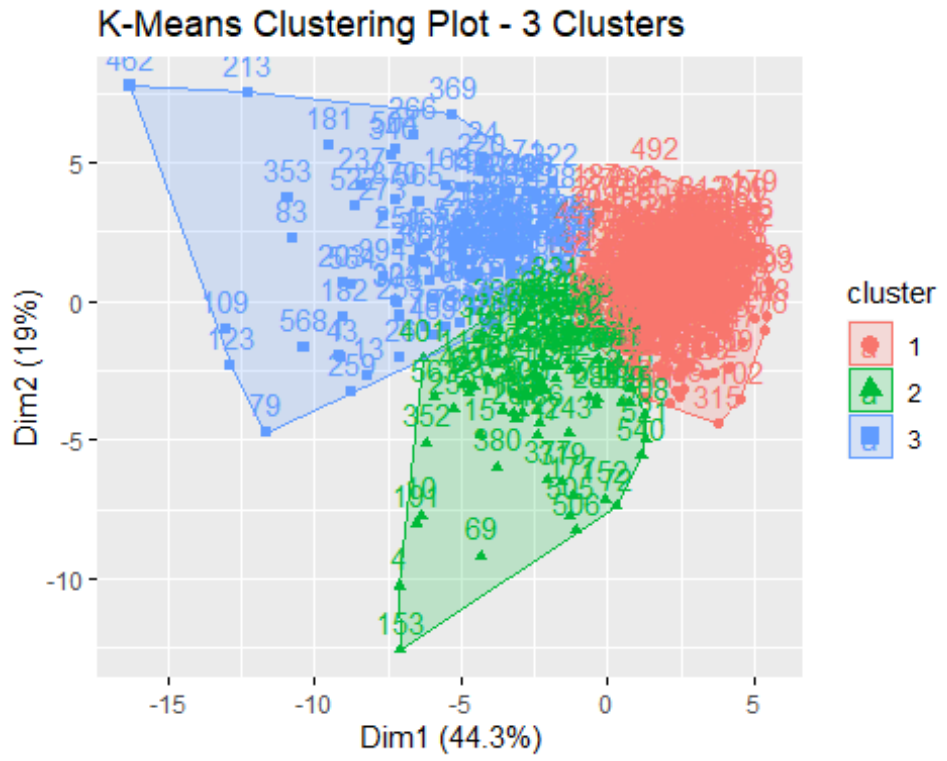
2 or 3 principal components. revisit & decide later.

K-means Clustering

```
df_scale <- scale(df_var)

#Kmeans for 2 clusters
km2.res <- kmeans(df_scale, 2)

km2.plot <- fviz_cluster(km2.res, data = df_scale)

km2.plot <- km2.plot + ggtitle("K-Means Clustering Plot - 2 Clusters")

km2.plot
```




```r
#Kmeans for 3 clusters
km3.res <- kmeans(df_scale, 3)

km3.plot <- fviz_cluster(km3.res, data = df_scale)

km3.plot <- km3.plot + ggtitle("K-Means Clustering Plot - 3 Clusters")

km3.plot
```



Data Transformations - this may need to be removed

```
# define the transformation or pre-processing
df_trans <- preProcess(df_var, method = c("BoxCox", "center", "scale"))
#apply the transformation
df_boxcox <- predict(df_trans, df_var)
head(df_boxcox[,1:4])
```

```
##   radius_mean texture_mean perimeter_mean  area_mean
## 1   1.1312223   -2.6966342      1.2560773  1.1292186
## 2   1.6105399   -0.2615935      1.5213622  1.7111779
## 3   1.4576801    0.5484335      1.4483646  1.5096864
## 4  -0.7554306    0.3590997     -0.5111072 -0.8425905
## 5   1.5629828   -1.2329217      1.5751647  1.6654085
## 6  -0.3631870   -0.8225400     -0.2467828 -0.4045600
```

```
#Rearranging variables for easiest continuity
df_original <- df_var
df <- df_boxcox

# List of group names
group_names <- c("mean", "se", "worst")

# Create an empty list to store the skewness values
skewness_list <- list()

# Loop through each group
```

```r
for (group_name in group_names) {
  # Subset the columns based on group name
  group_columns <- grep(group_name, names(df), value = TRUE)

  # Loop through each column in the group
  for (col_name in group_columns) {
    skewness <- skewness(df[[col_name]])

    # Print skewness value
    cat("Skewness for", col_name, ":", skewness, "\n")

    # Store the skewness value in the list
    skewness_list[[col_name]] <- skewness
  }
}
```

```
## Skewness for radius_mean : -0.018084
## Skewness for texture_mean : -0.01380153
## Skewness for perimeter_mean : -0.01825973
## Skewness for area_mean : 0.2834568
## Skewness for smoothness_mean : -0.0674592
## Skewness for compactness_mean : -0.03390649
## Skewness for concavity_mean : 1.393801
## Skewness for concave.points_mean : 1.165012
## Skewness for symmetry_mean : 0.001737667
## Skewness for fractal_dimension_mean : 0.1506466
## Skewness for radius_se : 0.02717609
## Skewness for texture_se : 0.02903681
## Skewness for perimeter_se : 0.06922794
## Skewness for area_se : 0.1153034
## Skewness for smoothness_se : -0.02401198
## Skewness for compactness_se : -0.004019758
## Skewness for concavity_se : 5.08355
## Skewness for concave.points_se : 1.43707
## Skewness for symmetry_se : 0.05491059
## Skewness for fractal_dimension_se : 0.01219151
## Skewness for radius_worst : 0.0263996
## Skewness for texture_worst : -0.003876732
## Skewness for perimeter_worst : 0.06122523
## Skewness for area_worst : 0.06768204
## Skewness for smoothness_worst : 0.02612512
## Skewness for compactness_worst : -0.2206758
## Skewness for concavity_worst : 1.144179
## Skewness for concave.points_worst : 0.4900213
## Skewness for symmetry_worst : -0.05654899
## Skewness for fractal_dimension_worst : 0.04705346
```

```r
# Convert the skewness list to a dataframe
skewness_df <- data.frame(Skewness = unlist(skewness_list))
```

```r
# Print the skewness dataframe
print(skewness_df)
```

```
##                            Skewness
## radius_mean             -0.018084005
## texture_mean            -0.013801528
## perimeter_mean          -0.018259725
## area_mean                0.283456808
## smoothness_mean         -0.067459204
## compactness_mean        -0.033906489
## concavity_mean           1.393800804
## concave.points_mean      1.165012377
## symmetry_mean            0.001737667
## fractal_dimension_mean   0.150646585
## radius_se                0.027176088
## texture_se               0.029036809
## perimeter_se             0.069227942
## area_se                  0.115303422
## smoothness_se           -0.024011982
## compactness_se          -0.004019758
## concavity_se             5.083550174
## concave.points_se        1.437070137
## symmetry_se              0.054910585
## fractal_dimension_se     0.012191507
## radius_worst             0.026399596
## texture_worst           -0.003876732
## perimeter_worst          0.061225231
## area_worst               0.067682043
## smoothness_worst         0.026125116
## compactness_worst       -0.220675829
## concavity_worst          1.144179410
## concave.points_worst     0.490021300
## symmetry_worst          -0.056548989
## fractal_dimension_worst  0.047053460
```

script k-fold cross, this one works

```r
#y <- df_diag$diagnosis
df_var_lr <- df_var
df_var_lr$diagnosis <- df_diag$diagnosis
#df_diag$diagnosis = as.integer(factor(df_diag$diagnosis))-1
#df_var$diagnosis <-  y #df_diag$diagnosis
#y <- df_diag$diagnosis
# create folds for k-fold cross validation
x_value_lr <- df_var_lr[ , -which(names(df_var_lr) == "diagnosis")]
y_value_lr <- df_var_lr$diagnosis
df_preprocess_lr <- preProcess(x_value_lr, method = c('BoxCox', 'center',
'scale'))
df_predict_lr <- predict(df_preprocess_lr,x_value_lr)

# create folds and apply them
```

```r
df_folds_lr <- createFolds(df_var_lr$diagnosis, returnTrain = TRUE)
crtl_df_lr <- trainControl(method = 'cv',
                           summaryFunction = twoClassSummary,
                           classProbs = TRUE,
                           savePredictions = TRUE,
                           index = df_folds_lr)

# cv split for xgboost
df_diag$diagnosis = as.integer(factor(df_diag$diagnosis))-1
df_var$diagnosis <- df_diag$diagnosis

# create folds for k-fold cross validation
df_preprocess <- preProcess(df_var, method = c('BoxCox', 'center', 'scale'))
df_predict <- predict(df_preprocess,df_var)

# create folds and apply them
df_folds <- createFolds(df_var$diagnosis, returnTrain = TRUE)
crtl_df <- trainControl(method = 'cv',
                           summaryFunction = twoClassSummary,
                           classProbs = TRUE,
                           savePredictions = TRUE,
                           index = df_folds)

# 1st will be LR cross validation
pen_grid <-  expand.grid(alpha = c(0, .4, .8, 1),
                           lambda = seq(.01, .2, length = 10))

set.seed(0)
log_reg_fit <- train(df_predict_lr,
                     y_value_lr,
                     method = 'glmnet',
                     tuneGrid = pen_grid,
                     metric = 'ROC',
                     trControl = crtl_df_lr)
                     #family = "binomial")
log_reg_fit
```

```
## glmnet
##
## 569 samples
##  30 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 512, 512, 512, 511, 512, 511, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda      ROC        Sens       Spec
##   0.0    0.01000000  0.9955394  0.9943651  0.9484848
```

```
##   0.0    0.03111111  0.9955394  0.9943651  0.9484848
##   0.0    0.05222222  0.9955356  0.9943651  0.9484848
##   0.0    0.07333333  0.9954094  0.9943651  0.9439394
##   0.0    0.09444444  0.9951448  0.9943651  0.9391775
##   0.0    0.11555556  0.9951410  0.9943651  0.9298701
##   0.0    0.13666667  0.9946104  0.9943651  0.9203463
##   0.0    0.15777778  0.9940933  0.9943651  0.9203463
##   0.0    0.17888889  0.9939671  0.9915079  0.9155844
##   0.0    0.20000000  0.9935725  0.9943651  0.9108225
##   0.4    0.01000000  0.9961925  0.9943651  0.9439394
##   0.4    0.03111111  0.9942249  0.9943651  0.9439394
##   0.4    0.05222222  0.9942271  0.9915079  0.9296537
##   0.4    0.07333333  0.9943616  0.9943651  0.9205628
##   0.4    0.09444444  0.9942256  0.9971429  0.9015152
##   0.4    0.11555556  0.9935725  0.9971429  0.9015152
##   0.4    0.13666667  0.9937085  0.9971429  0.8826840
##   0.4    0.15777778  0.9937107  0.9971429  0.8683983
##   0.4    0.17888889  0.9933259  0.9971429  0.8588745
##   0.4    0.20000000  0.9931921  0.9971429  0.8404762
##   0.8    0.01000000  0.9944774  0.9943651  0.9484848
##   0.8    0.03111111  0.9939566  0.9971429  0.9296537
##   0.8    0.05222222  0.9938423  0.9942857  0.9155844
##   0.8    0.07333333  0.9935943  0.9971429  0.8967532
##   0.8    0.09444444  0.9929449  0.9971429  0.8733766
##   0.8    0.11555556  0.9916485  1.0000000  0.8545455
##   0.8    0.13666667  0.9895350  1.0000000  0.8402597
##   0.8    0.15777778  0.9884692  1.0000000  0.8261905
##   0.8    0.17888889  0.9883369  1.0000000  0.8025974
##   0.8    0.20000000  0.9883407  1.0000000  0.7837662
##   1.0    0.01000000  0.9942151  0.9915873  0.9484848
##   1.0    0.03111111  0.9943654  0.9914286  0.9296537
##   1.0    0.05222222  0.9941249  0.9942857  0.9060606
##   1.0    0.07333333  0.9924316  1.0000000  0.8874459
##   1.0    0.09444444  0.9897004  1.0000000  0.8590909
##   1.0    0.11555556  0.9881138  1.0000000  0.8497835
##   1.0    0.13666667  0.9875809  1.0000000  0.8307359
##   1.0    0.15777778  0.9858576  1.0000000  0.8121212
##   1.0    0.17888889  0.9842709  1.0000000  0.7980519
##   1.0    0.20000000  0.9837418  1.0000000  0.7554113
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.4 and lambda = 0.01.

confusionMatrix(log_reg_fit, norm = 'none')

## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##           Reference
```

```
## Prediction   B   M
##          B 355  12
##          M   2 200
##
##  Accuracy (average) : 0.9754
```

```
cv_lr_roc <- roc(response = log_reg_fit$pred$obs,
                 predictor = log_reg_fit$pred$M,
                 levels = rev(levels(log_reg_fit$pred$obs)))
```

```
## Setting direction: controls > cases
```

```
print("Log Reg Cross-Val metrics")
```

```
## [1] "Log Reg Cross-Val metrics"
```

```
# Calculate accuracy
cv_lr_acc <- confusionMatrix(log_reg_fit$pred$obs,
log_reg_fit$pred$pred)$overall["Accuracy"]
print(paste("Accuracy:", cv_lr_acc))
```

```
## [1] "Accuracy: 0.956239015817223"
```

```
# Calculate sensitivity
cv_lr_sens <- confusionMatrix(log_reg_fit$pred$obs,
log_reg_fit$pred$pred)$byClass["Sensitivity"]
print(paste("Sensitivity:", cv_lr_sens))
```

```
## [1] "Sensitivity: 0.937722419928826"
```

```
# Calculate specificity
cv_lr_spec <- confusionMatrix(log_reg_fit$pred$obs,
log_reg_fit$pred$pred)$byClass["Specificity"]
print(paste("Specificity:", cv_lr_spec))
```

```
## [1] "Specificity: 0.993277089375165"
```

```
# Calculate precision (positive predictive value)
cv_lr_prec <- confusionMatrix(log_reg_fit$pred$obs,
log_reg_fit$pred$pred)$byClass["Pos Pred Value"]
print(paste("Precision:", cv_lr_prec))
```

```
## [1] "Precision: 0.996428571428572"
```

```
# Calculate F1-score
cv_lr_f1 <- 2 * (cv_lr_prec * cv_lr_sens) / (cv_lr_prec + cv_lr_sens)
print(paste("F1-score:", cv_lr_f1))
```

```
## [1] "F1-score: 0.966184558973314"
```

```
# Calculate AUC score
cv_lr_auc <- auc(cv_lr_roc)
print(paste("AUC/ROC-score:", cv_lr_auc))
```

```
## [1] "AUC/ROC-score: 0.990951811148988"

#Universal XGBoost Set Up

#parameters
param <- list(
  "objective"           = "binary:logistic",
  "eval_metric"         = "auc",
  "eta"                 = 0.01,
  "max_depth"           = 6,
  "subsample"           = 0.8,
  "colsample_bytree"    = 0.8,
  "min_child_weight"    = 1,
  "gamma"               = 0
)

xgb.nround <- 1000
earlyStoppingRound <- 250
xgb.nfold <- 5
```

#Cross Validation XGBoost

```
#matrix prep
set.seed(0)
for (i in 1:xgb.nfold) {
  # Split the data into training and testing sets based on the fold
  cvtrain_indices <- unlist(df_folds[-i])
  cvtest_indices <- df_folds[[i]]
  cvtrain_data <- df_var[cvtrain_indices, ]
  cvtest_data <- df_var[cvtest_indices, ]}

# Create xgb.DMatrix for training and testing data
  cvtrain_data_matrix <- as.matrix(cvtrain_data[, -1])
  cvtrain_data_label <- as.numeric(cvtrain_data$diagnosis)
  cv_xgbtrain <- xgb.DMatrix(data = cvtrain_data_matrix, label =
cvtrain_data_label)

  cvtest_data_matrix <- as.matrix(cvtest_data[, -1])
  cvtest_data_label <- as.numeric(cvtest_data$diagnosis)
  cv_xgbtest <- xgb.DMatrix(data = cvtest_data_matrix, label =
cvtest_data_label)

# Train the xgboost model using xgb.cv
  cv_model_xgb_crossval <- xgb.cv(
    params = param,
    data = cv_xgbtrain,
    nrounds = xgb.nround,
    maximize = TRUE,
    nfold = xgb.nfold,
    prediction = TRUE,
```

```
    early_stopping_round = earlyStoppingRound,
    verbose = 0
  )

# Extract the best iteration from the cross-validated model
  best_iteration <- cv_model_xgb_crossval$best_iteration

# Train the xgboost model using xgboost with the best iteration
  cv_model_xgb <- xgboost(
    params = param,
    data = cv_xgbtrain,
    nrounds = best_iteration,
    maximize = TRUE,
          verbose = 0

  )

# Make predictions on the test set using the xgboost model
  cv_predictions <- predict(cv_model_xgb, newdata = cv_xgbtest)

#Metrics
# Convert predicted labels and actual labels to factors with the same levels
cv_predicted_labels <- factor(ifelse(cv_predictions > 0.5, 1, 0), levels =
c(0, 1))
cvtest_data$diagnosis <- factor(cvtest_data$diagnosis, levels = c(0, 1))

# Create a confusion matrix
cv_CM <- confusionMatrix(data = cv_predicted_labels, reference =
cvtest_data$diagnosis)
cv_CM

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 327   0
##          1   0 185
##
##                Accuracy : 1
##                  95% CI : (0.9928, 1)
##     No Information Rate : 0.6387
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
```

```
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6387
##          Detection Rate : 0.6387
##    Detection Prevalence : 0.6387
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

```
# Convert predicted probabilities and actual labels to vectors
cv_predicted_probs <- as.numeric(cv_predictions)
cv_actual_labels <- as.numeric(as.character(cvtest_data$diagnosis))

# Calculate the ROC curve
cv_roc_results_xgb <- roc(cv_actual_labels, cv_predicted_probs)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
# Get the sensitivity value (True Positive Rate)
cv_sens <- cv_CM$byClass["Sensitivity"]
cv_sens
```

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

```
cv_spec <- cv_CM$byClass["Specificity"]
cv_spec
```

```
## Specificity
##           1
```

```
# Convert predicted labels and actual labels to factors with the same levels
cv_predicted_labels <- factor(ifelse(cv_predictions > 0.5, 1, 0), levels =
c(0, 1))
cvtest_data$diagnosis <- factor(cvtest_data$diagnosis, levels = c(0, 1))

# Create a confusion matrix
cv_CM <- confusionMatrix(data = cv_predicted_labels, reference =
cvtest_data$diagnosis)
cv_CM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 327   0
##          1   0 185
##
##                Accuracy : 1
##                  95% CI : (0.9928, 1)
```

```
##     No Information Rate : 0.6387
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6387
##          Detection Rate : 0.6387
##    Detection Prevalence : 0.6387
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

```
# Calculate accuracy
cv_lr_acc <- cv_CM$overall["Accuracy"]
print(paste("Accuracy:", cv_lr_acc))
```

```
## [1] "Accuracy: 1"
```

```
# Calculate sensitivity
cv_lr_sens <- cv_CM$byClass["Sensitivity"]
print(paste("Sensitivity:", cv_lr_sens))
```

```
## [1] "Sensitivity: 1"
```

```
# Calculate specificity
cv_lr_spec <- cv_CM$byClass["Specificity"]
print(paste("Specificity:", cv_lr_spec))
```

```
## [1] "Specificity: 1"
```

```
# Calculate precision (positive predictive value)
cv_lr_prec <- cv_CM$byClass["Pos Pred Value"]
print(paste("Precision:", cv_lr_prec))
```

```
## [1] "Precision: 1"
```

```
# Calculate F1-score
cv_lr_f1 <- 2 * (cv_lr_prec * cv_lr_sens) / (cv_lr_prec + cv_lr_sens)
print(paste("F1-score:", cv_lr_f1))
```

```
## [1] "F1-score: 1"
```

```
# Calculate AUC score
cv_lr_auc <- auc(cv_roc_results_xgb)
print(paste("AUC/ROC-score:", cv_lr_auc))
```

```
## [1] "AUC/ROC-score: 1"
```

70/30 Split

```
#randomly split the data 70/30 LR
trainingRows <- createDataPartition(df_var_lr$diagnosis, p = 0.7, list =
FALSE)
df_train_lr <- df_var_lr[trainingRows, ]
df_test_lr <- df_var_lr[-trainingRows, ]
train_imp_lr <- preProcess(df_train_lr, method = c("BoxCox", 'center',
'scale'))
trainpre_lr <- predict(train_imp_lr, df_train_lr)
testpre_lr <- predict(train_imp_lr, df_test_lr)
X_train_lr <- trainpre_lr[, -which(names(trainpre_lr) == "diagnosis")]
y_train_lr <- trainpre_lr$diagnosis
X_test_lr <- testpre_lr[, -which(names(testpre_lr) == "diagnosis")]
y_test_lr <- testpre_lr$diagnosis

# 70/30 split for xgboost
df_diag$diagnosis = as.integer(factor(df_diag$diagnosis))-1
trainingRows <- createDataPartition(df_diag$diagnosis, p = 0.7, list = FALSE)
df_train <- df_diag[trainingRows, ]
df_test <- df_diag[-trainingRows, ]
X_train <- df_train[, -which(names(df_train) == "diagnosis")]
y_train <- df_train$diagnosis
X_test <- df_test[, -which(names(df_test) == "diagnosis")]
y_test <- df_test$diagnosis

# create model 3 random 70/30 split for LR
set.seed(0)
pen_grid_2 <-  expand.grid(alpha = c(0, .4, .8, 1),
                           lambda = seq(.01, .2, length = 10))
log_reg_fit_3 <- train(x =X_train_lr, y= y_train_lr,
                       method = 'glmnet',
                       metric = 'ROC',
                       tuneGrid = pen_grid_2,
                       trControl = trainControl(classProbs = TRUE,
summaryFunction = twoClassSummary)) #, type.measure = "class"))

log_reg_fit_3
```

```
## glmnet
##
## 399 samples
##  30 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 399, 399, 399, 399, 399, 399, ...
## Resampling results across tuning parameters:
```

```
##
##   alpha  lambda      ROC        Sens       Spec
##   0.0    0.01000000  0.9931210  0.9897600  0.9279517
##   0.0    0.03111111  0.9931210  0.9897600  0.9279517
##   0.0    0.05222222  0.9928581  0.9910899  0.9248031
##   0.0    0.07333333  0.9924102  0.9905741  0.9203651
##   0.0    0.09444444  0.9921980  0.9905741  0.9137264
##   0.0    0.11555556  0.9918384  0.9905741  0.9097054
##   0.0    0.13666667  0.9916308  0.9905741  0.9038363
##   0.0    0.15777778  0.9914186  0.9897530  0.9010876
##   0.0    0.17888889  0.9912778  0.9897530  0.8957404
##   0.0    0.20000000  0.9911079  0.9901697  0.8899787
##   0.4    0.01000000  0.9927730  0.9906195  0.9303132
##   0.4    0.03111111  0.9912046  0.9909102  0.9193857
##   0.4    0.05222222  0.9906472  0.9923482  0.9147058
##   0.4    0.07333333  0.9904835  0.9932290  0.9021255
##   0.4    0.09444444  0.9904872  0.9941888  0.8908536
##   0.4    0.11555556  0.9904739  0.9955977  0.8799423
##   0.4    0.13666667  0.9902900  0.9969998  0.8642704
##   0.4    0.15777778  0.9901757  0.9974164  0.8468695
##   0.4    0.17888889  0.9899062  0.9982457  0.8337117
##   0.4    0.20000000  0.9895444  0.9982457  0.8234229
##   0.8    0.01000000  0.9908624  0.9900962  0.9267482
##   0.8    0.03111111  0.9901374  0.9905227  0.9125506
##   0.8    0.05222222  0.9902103  0.9938791  0.8993617
##   0.8    0.07333333  0.9899226  0.9966455  0.8804631
##   0.8    0.09444444  0.9894498  0.9978836  0.8640579
##   0.8    0.11555556  0.9884698  0.9979253  0.8390434
##   0.8    0.13666667  0.9873457  0.9983335  0.8096513
##   0.8    0.15777778  0.9860702  0.9987545  0.7844212
##   0.8    0.17888889  0.9850518  0.9991506  0.7610968
##   0.8    0.20000000  0.9844216  0.9991506  0.7311463
##   1.0    0.01000000  0.9897748  0.9860436  0.9191658
##   1.0    0.03111111  0.9897257  0.9897408  0.9084557
##   1.0    0.05222222  0.9892096  0.9934846  0.8915483
##   1.0    0.07333333  0.9878328  0.9943358  0.8624132
##   1.0    0.09444444  0.9859294  0.9943347  0.8412648
##   1.0    0.11555556  0.9837028  0.9944599  0.8183402
##   1.0    0.13666667  0.9819233  0.9970595  0.7898444
##   1.0    0.15777778  0.9807946  0.9966428  0.7634316
##   1.0    0.17888889  0.9797845  0.9992040  0.7351079
##   1.0    0.20000000  0.9787950  0.9996000  0.6983532
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0 and lambda =
0.03111111.

# obtain predictions
predictions_lr_3 <- predict(log_reg_fit_3, newdata = X_test_lr)
```

```r
# Create confusion matrix
y_test_factor_3 <- as.factor(y_test_lr)
confusion_matrix_lr_3 <- confusionMatrix(data = predictions_lr_3, reference =
y_test_factor_3)
confusion_matrix_lr_3
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   B   M
##          B 105   1
##          M   2  62
##
##                Accuracy : 0.9824
##                  95% CI : (0.9493, 0.9963)
##     No Information Rate : 0.6294
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9623
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9813
##             Specificity : 0.9841
##          Pos Pred Value : 0.9906
##          Neg Pred Value : 0.9688
##              Prevalence : 0.6294
##          Detection Rate : 0.6176
##    Detection Prevalence : 0.6235
##       Balanced Accuracy : 0.9827
##
##        'Positive' Class : B
##
```

```r
# calculate the ROC scores
y_test_num_3 <- as.numeric(y_test_factor_3)
roc_results_lr_3 <- roc(response = predictions_lr_3, predictor =
y_test_num_3)
```

```
## Setting levels: control = B, case = M
```

```
## Setting direction: controls < cases
```

```r
roc_results_lr_3
```

```
##
## Call:
## roc.default(response = predictions_lr_3, predictor = y_test_num_3)
##
## Data: y_test_num_3 in 106 controls (predictions_lr_3 B) < 64 cases
```

```
(predictions_lr_3 M).
## Area under the curve: 0.9797

print("Log Reg random 70/30 split Performance metrics:")

## [1] "Log Reg random 70/30 split Performance metrics:"

# Calculate accuracy
cv_lr_acc_3 <- confusion_matrix_lr_3$overall["Accuracy"]
print(paste("Accuracy:", cv_lr_acc_3))

## [1] "Accuracy: 0.982352941176471"

# Calculate sensitivity
cv_lr_sens_3 <- confusion_matrix_lr_3$byClass["Sensitivity"]
print(paste("Sensitivity:", cv_lr_sens_3))

## [1] "Sensitivity: 0.981308411214953"

# Calculate specificity
cv_lr_spec_3 <- confusion_matrix_lr_3$byClass["Specificity"]
print(paste("Specificity:", cv_lr_spec_3))

## [1] "Specificity: 0.984126984126984"

# Calculate precision (positive predictive value)
cv_lr_prec_3 <- confusion_matrix_lr_3$byClass["Pos Pred Value"]
print(paste("Precision:", cv_lr_prec_3))

## [1] "Precision: 0.990566037735849"

# Calculate F1-score
cv_lr_f1_3 <- 2 * (cv_lr_prec_3 * cv_lr_sens_3) / (cv_lr_prec_3 +
cv_lr_sens_3)
print(paste("F1-score:", cv_lr_f1_3))

## [1] "F1-score: 0.985915492957746"

set.seed(0)
#Random Split (70/30) XGBoost

#matrix prep
rs_xgbtrain <- xgb.DMatrix(data = as.matrix(X_train), label = y_train)
rs_xgbtest <- xgb.DMatrix(data = as.matrix(X_test), label = y_test)

rs_model_xgb_crossval <- xgb.cv(
  params = param,
  data = rs_xgbtrain,
  nrounds = xgb.nround,
  maximize = TRUE,
  nfold = xgb.nfold,
  prediction = TRUE,
  early_stopping_round=earlyStoppingRound,
```

```
  verbose = 0
)

#Model
rs_model_xgb <- xgboost(
  params = param,
  data = rs_xgbtrain,
  nrounds = xgb.nround,
  maximize = TRUE,
  early_stopping_round=earlyStoppingRound,
  verbose = 0
)

#Predict
X_test$predicted <- round(predict(object = rs_model_xgb ,newdata =
rs_xgbtest),0)

#Metrics
RS_xgb_AUC <- auc(y_test, X_test$predicted)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

RS_xgb_AUC

## Area under the curve: 0.9515

rs_CM <- confusionMatrix(factor(X_test$predicted),factor(y_test))
rs_CM

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 104   5
##          1   2  59
##
##                Accuracy : 0.9588
##                  95% CI : (0.917, 0.9833)
##     No Information Rate : 0.6235
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9115
##
##  Mcnemar's Test P-Value : 0.4497
##
##             Sensitivity : 0.9811
##             Specificity : 0.9219
##          Pos Pred Value : 0.9541
##          Neg Pred Value : 0.9672
```

```
##              Prevalence : 0.6235
##          Detection Rate : 0.6118
##    Detection Prevalence : 0.6412
##       Balanced Accuracy : 0.9515
##
##        'Positive' Class : 0
##
```

```
print("XGB Random 70-30 split Performance Metrics")
```

```
## [1] "XGB Random 70-30 split Performance Metrics"
```

```
# Get the sensitivity value (True Positive Rate)
rs_sens <- rs_CM$byClass["Sensitivity"]
rs_sens
```

```
## Sensitivity
##   0.9811321
```

```
rs_spec <- rs_CM$byClass["Specificity"]
rs_spec
```

```
## Specificity
##    0.921875
```

```
# calculate the ROC scores
rs_roc_results_xgb <- roc(response = X_test$predicted, predictor = y_test)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
rs_roc_results_xgb
```

```
##
## Call:
## roc.default(response = X_test$predicted, predictor = y_test)
##
## Data: y_test in 109 controls (X_test$predicted 0) < 61 cases
(X_test$predicted 1).
## Area under the curve: 0.9607
```

```
# Calculate precision (positive predictive value)
rs_prec <- rs_CM$byClass["Pos Pred Value"]
print(paste("Precision:", rs_prec))
```

```
## [1] "Precision: 0.954128440366972"
```

```
# Calculate F1-score
rs_f1 <- 2 * (rs_prec * rs_sens) / (rs_prec + rs_sens)
print(paste("F1-score:", rs_f1))
```

```
## [1] "F1-score: 0.967441860465116"
```

```r
# perform a random oversampling on the dataset
# Splitting the data into training and testing sets
trainingRows_2 <- createDataPartition(df_var_lr$diagnosis, p = 0.7, list =
FALSE)
df_train_2_lr <- df_var_lr[trainingRows_2, ]
df_test_2_lr <- df_var_lr[-trainingRows_2, ]
# scale and transform the data
trainimp_lr_2 <- preProcess(df_train_2_lr, method = c("BoxCox", 'center',
'scale'))
train_pre_lr_2 <- predict(trainimp_lr_2, df_train_2_lr)
test_pre_lr_2 <- predict(trainimp_lr_2, df_test_2_lr)

# Separate predictors and target variables in the training set
#X_train_2 <- df_train_2[, -which(names(df_train_2) == "diagnosis")]
#y_train_2 <- df_train_2$diagnosis

# Perform oversampling on the training set
oversampled_data_lr <- ovun.sample(diagnosis ~ ., data = train_pre_lr_2,
method = "over")

# Extract the oversampled predictors and target variables
X_train_oversampled_lr <- oversampled_data_lr$data[,
-which(names(oversampled_data_lr$data) == "diagnosis")]
y_train_oversampled_lr <- oversampled_data_lr$data$diagnosis

# Separate predictors and target variables in the testing set
X_test_oversampled_lr <- test_pre_lr_2[, -which(names(test_pre_lr_2) ==
"diagnosis")]
y_test_oversampled_lr <- test_pre_lr_2$diagnosis

#70/30 oversampled split
# Splitting the data into training and testing sets
trainingRows_2 <- createDataPartition(df_diag$diagnosis, p = 0.7, list =
FALSE)
df_train_2 <- df_diag[trainingRows_2, ]
df_test_2 <- df_diag[-trainingRows_2, ]

# Separate predictors and target variables in the training set
X_train_2 <- df_train_2[, -which(names(df_train_2) == "diagnosis")]
y_train_2 <- df_train_2$diagnosis

# Perform oversampling on the training set
oversampled_data <- ovun.sample(diagnosis ~ ., data = df_train_2, method =
"over")

# Extract the oversampled predictors and target variables
X_train_oversampled <- oversampled_data$data[,
-which(names(oversampled_data$data) == "diagnosis")]
y_train_oversampled <- oversampled_data$data$diagnosis
```

```r
# Separate predictors and target variables in the testing set
X_test_oversampled <- df_test_2[, -which(names(df_test_2) == "diagnosis")]
y_test_oversampled <- df_test_2$diagnosis

# create model 2 using the random over sampler technique on a 70/30 split
pen_grid_2 <-  expand.grid(alpha = c(0, .4, .8, 1),
                           lambda = seq(.01, .2, length = 10))
set.seed(0)
log_reg_fit_2 <- train(x =X_train_oversampled_lr, y= y_train_oversampled_lr,
                       method = 'glmnet',
                       metric = 'ROC',
                       tuneGrid = pen_grid_2,
                       trControl = trainControl(classProbs = TRUE,
summaryFunction = twoClassSummary))

log_reg_fit_2
```

```
## glmnet
##
## 507 samples
##  30 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 507, 507, 507, 507, 507, 507, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda      ROC        Sens       Spec
##   0.0    0.01000000  0.9983411  0.9717500  0.9874170
##   0.0    0.03111111  0.9983411  0.9717500  0.9874170
##   0.0    0.05222222  0.9981443  0.9698265  0.9866000
##   0.0    0.07333333  0.9978155  0.9680309  0.9857530
##   0.0    0.09444444  0.9975038  0.9648268  0.9861877
##   0.0    0.11555556  0.9971680  0.9635352  0.9857917
##   0.0    0.13666667  0.9969464  0.9622493  0.9857917
##   0.0    0.15777778  0.9967518  0.9622493  0.9857917
##   0.0    0.17888889  0.9965557  0.9618283  0.9854143
##   0.0    0.20000000  0.9963431  0.9618283  0.9850183
##   0.4    0.01000000  0.9989045  0.9776897  0.9865012
##   0.4    0.03111111  0.9980989  0.9746407  0.9851678
##   0.4    0.05222222  0.9972036  0.9684193  0.9816255
##   0.4    0.07333333  0.9966758  0.9651863  0.9791204
##   0.4    0.09444444  0.9962139  0.9634428  0.9786976
##   0.4    0.11555556  0.9957657  0.9595606  0.9813839
##   0.4    0.13666667  0.9952920  0.9574812  0.9807726
##   0.4    0.15777778  0.9948426  0.9540114  0.9807913
##   0.4    0.17888889  0.9944048  0.9503985  0.9792329
##   0.4    0.20000000  0.9940526  0.9469936  0.9776762
```

```
##   0.8    0.01000000  0.9986692  0.9800459  0.9848857
##   0.8    0.03111111  0.9968244  0.9692020  0.9830294
##   0.8    0.05222222  0.9958967  0.9580859  0.9839322
##   0.8    0.07333333  0.9949153  0.9488130  0.9857749
##   0.8    0.09444444  0.9940129  0.9399139  0.9850094
##   0.8    0.11555556  0.9933675  0.9366865  0.9830065
##   0.8    0.13666667  0.9930914  0.9375860  0.9830545
##   0.8    0.15777778  0.9928674  0.9358704  0.9814563
##   0.8    0.17888889  0.9928473  0.9368167  0.9809781
##   0.8    0.20000000  0.9926360  0.9368167  0.9790508
##   1.0    0.01000000  0.9982243  0.9775176  0.9840145
##   1.0    0.03111111  0.9961452  0.9593960  0.9838637
##   1.0    0.05222222  0.9944638  0.9485455  0.9844904
##   1.0    0.07333333  0.9929488  0.9364160  0.9846554
##   1.0    0.09444444  0.9920077  0.9345733  0.9839529
##   1.0    0.11555556  0.9917044  0.9332809  0.9818808
##   1.0    0.13666667  0.9914651  0.9333846  0.9795124
##   1.0    0.15777778  0.9913267  0.9328002  0.9791465
##   1.0    0.17888889  0.9910388  0.9312598  0.9740529
##   1.0    0.20000000  0.9909643  0.9310535  0.9663106
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.4 and lambda = 0.01.
```

```
# obtain predictions
predictions_lr <- predict(log_reg_fit_2, newdata = X_test_oversampled_lr)
# build confusion matrix
# set y_test as a facotr
y_test_factor_lr <- as.factor(y_test_oversampled_lr)
confusionMatrix(data = predictions_lr, reference = y_test_factor_lr)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   B   M
##          B 104   3
##          M   3  60
##
##                Accuracy : 0.9647
##                  95% CI : (0.9248, 0.9869)
##     No Information Rate : 0.6294
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9243
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9720
##             Specificity : 0.9524
##          Pos Pred Value : 0.9720
```

```
##          Neg Pred Value : 0.9524
##              Prevalence : 0.6294
##          Detection Rate : 0.6118
##    Detection Prevalence : 0.6294
##       Balanced Accuracy : 0.9622
##
##        'Positive' Class : B
##
```

```
# calculate the ROC scores
y_test_num_lr <- as.numeric(y_test_factor_lr)
roc_results_lr <- roc(response = predictions_lr, predictor = y_test_num_lr)
```

```
## Setting levels: control = B, case = M
```

```
## Setting direction: controls < cases
```

```
print("Oversampled Log Reg Performance Metrics")
```

```
## [1] "Oversampled Log Reg Performance Metrics"
```

```
# Calculate accuracy
lr_acc <- confusionMatrix(predictions_lr,
y_test_factor_lr)$overall["Accuracy"]
print(paste("Accuracy:", lr_acc))
```

```
## [1] "Accuracy: 0.964705882352941"
```

```
# Calculate sensitivity (True Positive Rate)
lr_sens <- confusionMatrix(predictions_lr,
y_test_factor_lr)$byClass["Sensitivity"]
print(paste("Sensitivity:", lr_sens))
```

```
## [1] "Sensitivity: 0.97196261682243"
```

```
# Calculate specificity (True Negative Rate)
lr_spec <- confusionMatrix(predictions_lr,
y_test_factor_lr)$byClass["Specificity"]
print(paste("Specificity:", lr_spec))
```

```
## [1] "Specificity: 0.952380952380952"
```

```
# Calculate precision (Positive Predictive Value)
lr_prec <- confusionMatrix(predictions_lr, y_test_factor_lr)$byClass["Pos
Pred Value"]
print(paste("Precision:", lr_prec))
```

```
## [1] "Precision: 0.97196261682243"
```

```
# Calculate F1-score
lr_f1 <- 2 * (lr_prec * lr_sens) / (lr_prec + lr_sens)
print(paste("F1-score:", lr_f1))
```

```
## [1] "F1-score: 0.97196261682243"

# Calculate AUC/ROC score
lr_auc <- roc_results_lr$auc
print(paste("AUC/ROC score:", lr_auc))

## [1] "AUC/ROC score: 0.962171784601691"

set.seed(0)
#Oversampled XGBoost

#matrix prep
os_xgbtrain <- xgb.DMatrix(data = as.matrix(X_train_oversampled), label =
y_train_oversampled)
os_xgbtest <- xgb.DMatrix(data = as.matrix(X_test_oversampled), label =
y_test_oversampled)

os_model_xgb_crossval <- xgb.cv(
  params = param,
  data = os_xgbtrain,
  nrounds = xgb.nround,
  maximize = TRUE,
  nfold = xgb.nfold,
  prediction = TRUE,
  early_stopping_round=earlyStoppingRound,
  verbose = 0
)

#Model
os_model_xgb <- xgboost(
  params = param,
  data = os_xgbtrain,
  nrounds = xgb.nround,
  maximize = TRUE,
  early_stopping_round=earlyStoppingRound,
  verbose = 0
)

#Predict
X_test$predicted <- round(predict(object = os_model_xgb ,newdata =
os_xgbtest),0)

os_CM <- confusionMatrix(factor(X_test$predicted),factor(y_test_oversampled))
os_CM

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 106   1
##          1   0  63
```

```
##
##                Accuracy : 0.9941
##                  95% CI : (0.9677, 0.9999)
##     No Information Rate : 0.6235
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9874
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9844
##          Pos Pred Value : 0.9907
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6235
##          Detection Rate : 0.6235
##    Detection Prevalence : 0.6294
##       Balanced Accuracy : 0.9922
##
##        'Positive' Class : 0
##
```

```
# Get the sensitivity value (True Positive Rate)
os_sens <- os_CM$byClass["Sensitivity"]
os_sens
```

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

```
os_spec <- os_CM$byClass["Specificity"]
os_spec
```

```
## Specificity
##    0.984375
```

```
# calculate the ROC scores
os_roc_results_xgb <- roc(response = X_test$predicted, predictor =
y_test_oversampled)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
os_roc_results_xgb
```

```
##
## Call:
## roc.default(response = X_test$predicted, predictor = y_test_oversampled)
##
## Data: y_test_oversampled in 107 controls (X_test$predicted 0) < 63 cases
(X_test$predicted 1).
## Area under the curve: 0.9953
```

```r
# Calculate precision
os_prec <- os_CM$byClass["Pos Pred Value"]
print(paste("Precision:", os_prec))
```

```
## [1] "Precision: 0.990654205607477"
```

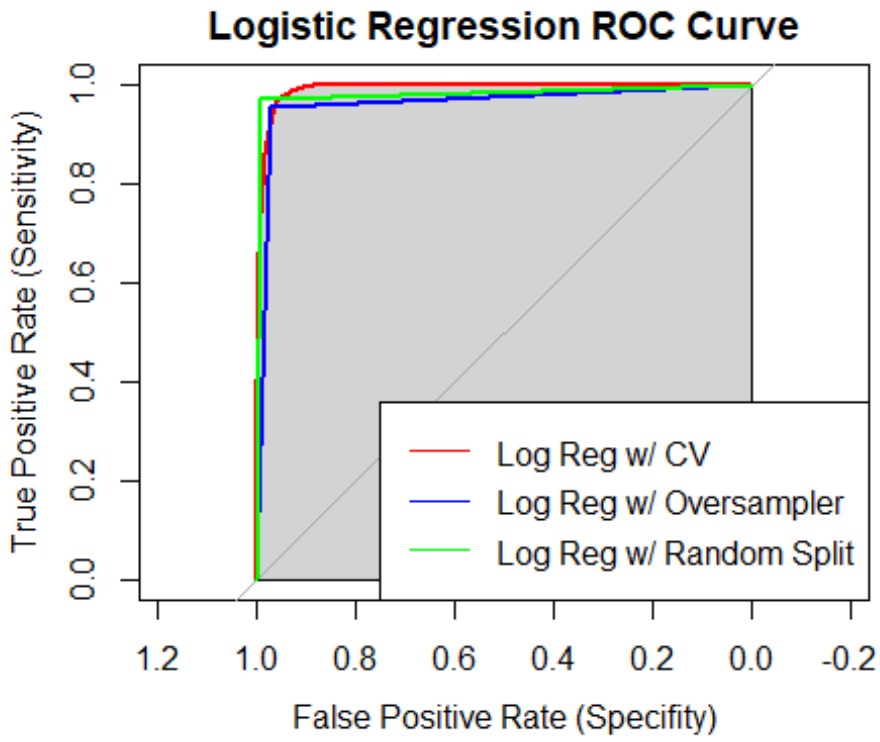
```r
# Calulate F1-score
os_f1 <- 2 * (os_prec * os_sens) / (os_prec + os_sens)
print(paste("F1-score:", os_f1))
```

```
## [1] "F1-score: 0.995305164319249"
```

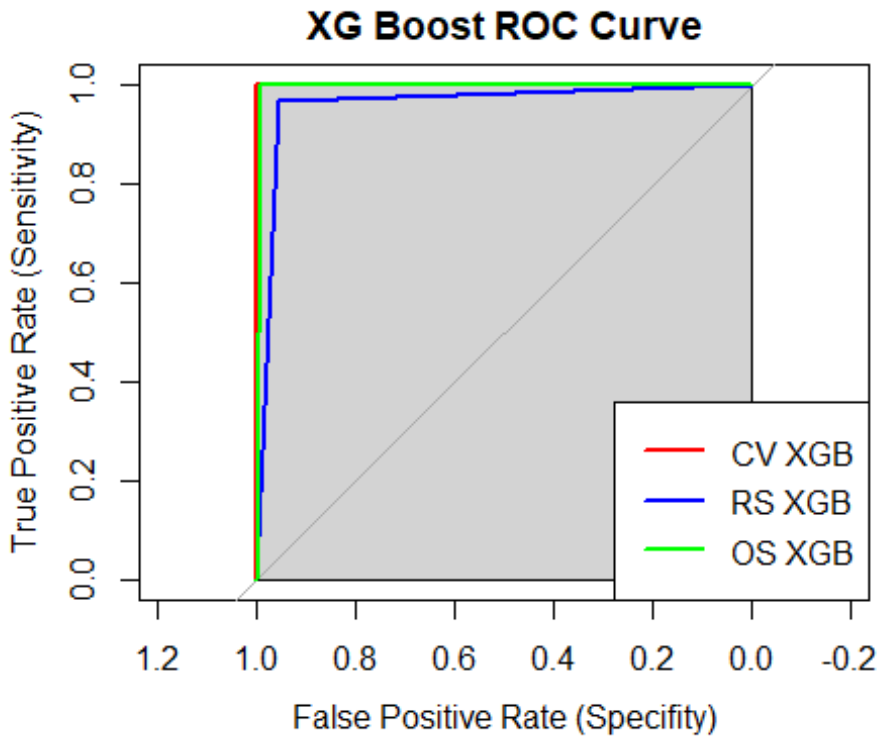
```r
# Calculate AUC/ROC score
os_auc <- os_roc_results_xgb$auc
print(paste("AUC/ROC score:", os_auc))
```

```
## [1] "AUC/ROC score: 0.995327102803738"
```

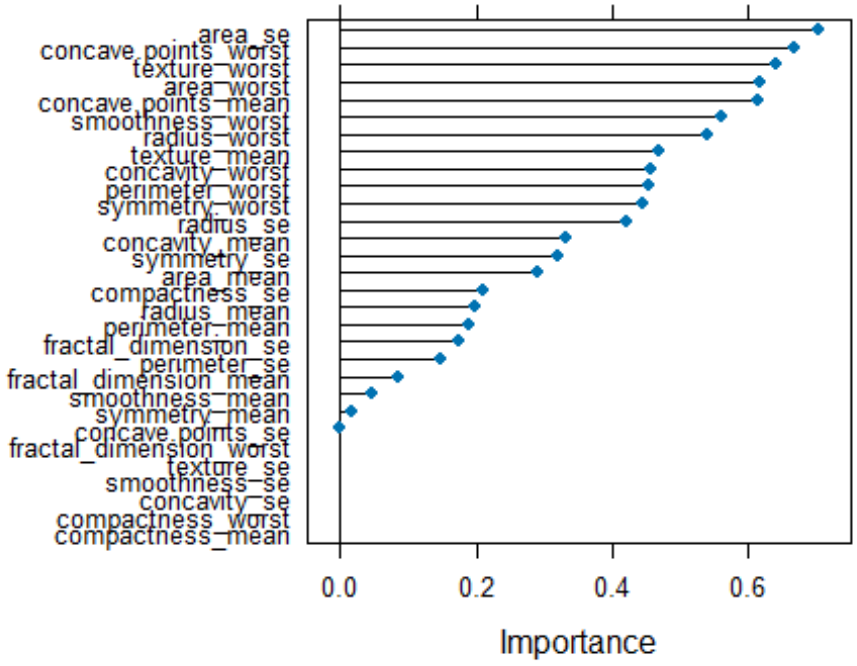
```r
# plot the ROC curve for each of the 3 Lr models
plot(cv_lr_roc, col = "red", main = "Logistic Regression ROC Curve", xlab =
"False Positive Rate (Specifity)", ylab = "True Positive Rate (Sensitivity)",
print.auc = FALSE, auc.polygon = TRUE, auc.polygon.col = "lightgray")
lines(roc_results_lr, col = "blue", print.auc = FALSE, auc.polygon = TRUE,
auc.polygon.col = "lightgray")
lines(roc_results_lr_3, col = "green", print.auc = FALSE, auc.polygon = TRUE,
auc.polygon.col = "lightgray")

# Add a legend
legend("bottomright", legend = c("Log Reg w/ CV", "Log Reg w/ Oversampler" ,
"Log Reg w/ Random Split"), col = c("red", "blue", "green"), lty = 1)
```



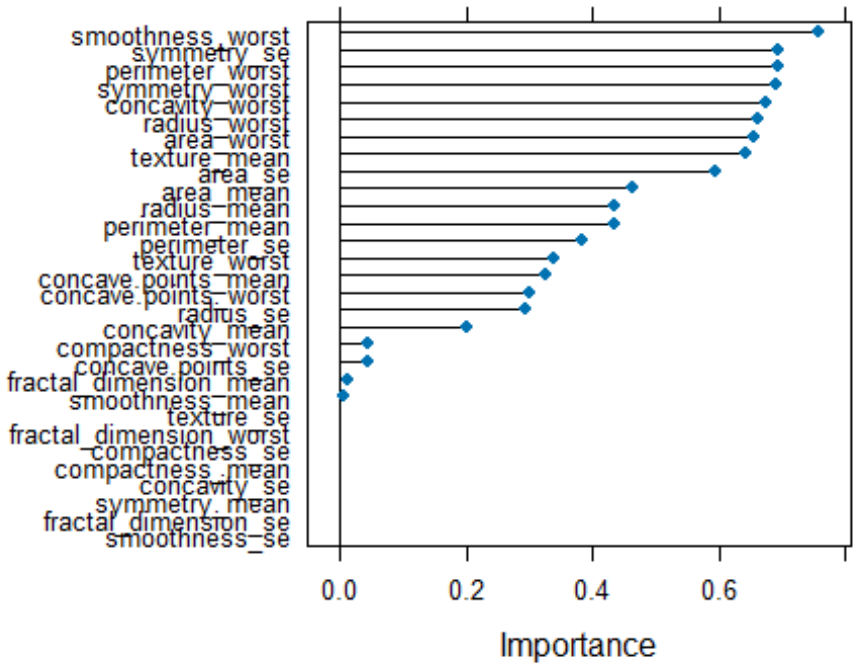

```r
# plot the ROC curve for each of the 3 XBoost models
plot(cv_roc_results_xgb, col = "red", main = "XG Boost ROC Curve", xlab =
"False Positive Rate (Specifity)", ylab = "True Positive Rate (Sensitivity)",
print.auc = FALSE, auc.polygon = TRUE, auc.polygon.col = "lightgray")
lines(rs_roc_results_xgb, col = "blue", print.auc = FALSE, auc.polygon =
TRUE, auc.polygon.col = "lightgray")
lines(os_roc_results_xgb, col = "green", print.auc = FALSE, auc.polygon =
TRUE, auc.polygon.col = "lightgray")
legend("bottomright", legend = c("CV XGB", "RS XGB", "OS XGB"), col =
c("red", "blue", "green"), lty = 1, lwd = 2)
```



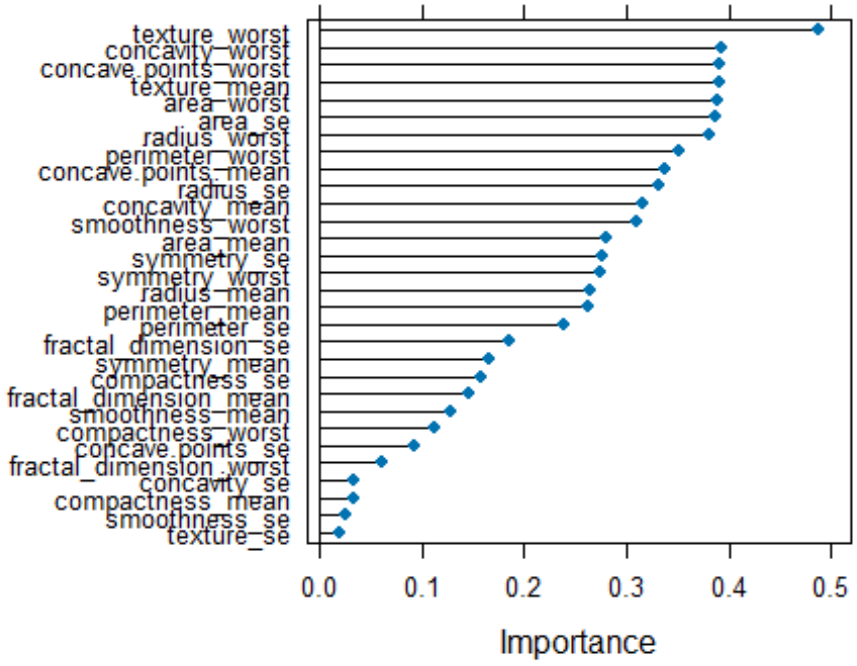

```
lr_1 <- varImp(log_reg_fit, scale = FALSE)
lr_2 <- varImp(log_reg_fit_2, scale = FALSE)
lr_3 <- varImp(log_reg_fit_3, scale = FALSE)
plot(lr_1)
```




```
plot(lr_2)
```




```
plot(lr_3)
```




```
set.seed(0)

# Load the data
df <-
read.csv("C:/MIDS/ADS-503_Applied_Predictive_Modeling/ADS_503_team_2_final_pr
oject/data/breast_cancer_FNA_data.csv")
data_rf <- df[, !(names(df) %in% c("X", "id"))]
print(names(data_rf))
```

```
##  [1] "diagnosis"               "radius_mean"
##  [3] "texture_mean"            "perimeter_mean"
##  [5] "area_mean"               "smoothness_mean"
##  [7] "compactness_mean"        "concavity_mean"
##  [9] "concave.points_mean"     "symmetry_mean"
## [11] "fractal_dimension_mean"  "radius_se"
## [13] "texture_se"              "perimeter_se"
## [15] "area_se"                 "smoothness_se"
## [17] "compactness_se"          "concavity_se"
## [19] "concave.points_se"       "symmetry_se"
## [21] "fractal_dimension_se"    "radius_worst"
## [23] "texture_worst"           "perimeter_worst"
## [25] "area_worst"              "smoothness_worst"
## [27] "compactness_worst"       "concavity_worst"
## [29] "concave.points_worst"    "symmetry_worst"
## [31] "fractal_dimension_worst"
```

```r
trainingRows_rf <- createDataPartition(data_rf$diagnosis, p = 0.7, list =
FALSE)
df_train_rf <- data_rf[trainingRows_rf, ]
df_test_rf <- data_rf[-trainingRows_rf, ]

# Preprocess the training and testing data for Random Forest
train_imp_rf <- preProcess(df_train_rf, method = c("BoxCox", 'center',
'scale'))
train_pre_rf <- predict(train_imp_rf, df_train_rf)
test_pre_rf <- predict(train_imp_rf, df_test_rf)
X_train_rf <- train_pre_rf[, -which(names(train_pre_rf) == "diagnosis")]
y_train_rf <- factor(df_train_rf$diagnosis, levels = c("B", "M"))
X_test_rf <- test_pre_rf[, -which(names(test_pre_rf) == "diagnosis")]
y_test_rf <- factor(df_test_rf$diagnosis, levels = c("B", "M"))

# Model 1: Random Forest with Random Oversampling
# Oversample the training set
data_rf_oversampled <- ROSE::ovun.sample(diagnosis ~ ., data = df_train_rf,
method = "over", N = nrow(df_train_rf), seed = 0)$data

# Split the oversampled data into features and labels for training set
X_train_rf_oversampled <- data_rf_oversampled[,
-which(names(data_rf_oversampled) == "diagnosis")]
y_train_rf_oversampled <- factor(data_rf_oversampled$diagnosis, levels =
c("B", "M"))

# Oversample the test set
data_test_rf_oversampled <- ROSE::ovun.sample(diagnosis ~ ., data =
df_test_rf, method = "over", N = nrow(df_test_rf), seed = 0)$data

# Split the oversampled data into features and labels for test set
X_test_rf_oversampled <- data_test_rf_oversampled[,
-which(names(data_test_rf_oversampled) == "diagnosis")]
y_test_rf_oversampled <- factor(data_test_rf_oversampled$diagnosis, levels =
c("B", "M"))

# Model 1: Random Forest with Random Oversampling
rf_fit_oversampling <- train(
  x = X_train_rf_oversampled,
  y = y_train_rf_oversampled,
  method = 'rf',
  metric = 'ROC',
  trControl = trainControl(classProbs = TRUE, summaryFunction =
twoClassSummary)
)

# Evaluate Model 1 on oversampled test set
predicted_probabilities_rf_1 <- predict(rf_fit_oversampling, newdata =
X_test_rf_oversampled, type = "prob")
```

```r
predicted_classes_rf_1 <- ifelse(predicted_probabilities_rf_1[, "M"] > 0.5,
"M", "B")
predicted_classes_rf_1 <- factor(predicted_classes_rf_1, levels = c("B",
"M"))
confusion_matrix_1 <- confusionMatrix(data = predicted_classes_rf_1,
reference = y_test_rf_oversampled)
precision_1 <- confusion_matrix_1$byClass["Pos Pred Value"]
f1_score_1 <- confusion_matrix_1$byClass["F1"]
sensitivity_1 <- confusion_matrix_1$byClass["Sensitivity"]
specificity_1 <- confusion_matrix_1$byClass["Specificity"]

# Get the predicted probabilities for the "M" class
predicted_probs_M <- predicted_probabilities_rf_1[, "M"]
# Create a new factor for the reference labels with levels in the correct
order
y_test_rf_oversampled_reordered <- factor(y_test_rf_oversampled, levels =
c("B", "M"))
# Calculate AUC/ROC
auc_roc_1 <- roc(response = as.numeric(y_test_rf_oversampled_reordered ==
"M"), predictor = predicted_probs_M)$auc
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
# Print results for Model 1
print("Model 1: Random Forest with Random Oversampling")
```

```
## [1] "Model 1: Random Forest with Random Oversampling"
```

```r
print(confusion_matrix_1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   B   M
##          B 104   2
##          M   3  61
##
##                Accuracy : 0.9706
##                  95% CI : (0.9327, 0.9904)
##     No Information Rate : 0.6294
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9372
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9720
##             Specificity : 0.9683
##          Pos Pred Value : 0.9811
```

```
##          Neg Pred Value : 0.9531
##              Prevalence : 0.6294
##          Detection Rate : 0.6118
##    Detection Prevalence : 0.6235
##       Balanced Accuracy : 0.9701
##
##        'Positive' Class : B
##
```

```
print(paste("Precision:", precision_1))
```

```
## [1] "Precision: 0.981132075471698"
```

```
print(paste("F1-score:", f1_score_1))
```

```
## [1] "F1-score: 0.976525821596244"
```

```
print(paste("AUC/ROC:", auc_roc_1))
```

```
## [1] "AUC/ROC: 0.994362854175938"
```

```
# Model 2: Random Forest with random 70/30 split

# Split the data into training and testing sets (70/30 split)
data_rf$diagnosis <- factor(data_rf$diagnosis)
train_indices <- createDataPartition(data_rf$diagnosis, p = 0.7, list =
FALSE)
train_data <- data_rf[train_indices, ]
test_data <- data_rf[-train_indices, ]

# Random Forest with 70-30 split
rf_model_split <- randomForest(
  x = train_data[, -which(names(train_data) == "diagnosis")],
  y = train_data$diagnosis,
  ntree = 1000,
  importance = TRUE
)

# Predict on the test set
test_data$predicted_split <- predict(rf_model_split, newdata = test_data[,
-which(names(test_data) == "diagnosis")])

confusion_matrix_2 <- confusionMatrix(data = test_data$predicted_split,
reference = test_data$diagnosis)

# Calculate precision, recall, and F1-score
precision_2 <- confusion_matrix_2$byClass["Pos Pred Value"]
recall_2 <- confusion_matrix_2$byClass["Sensitivity"]
f1_score_2 <- 2 * (precision_2 * recall_2) / (precision_2 + recall_2)
sensitivity_2 <- confusion_matrix_2$byClass["Sensitivity"]
specificity_2 <- confusion_matrix_2$byClass["Specificity"]
```

```r
# Extract predicted probabilities for the positive class ("M")
predicted_probs <- predict(rf_model_split, newdata = test_data[,
-which(names(test_data) == "diagnosis")], type = "prob")
predicted_probs_M <- predicted_probs[, "M"]

# Calculate AUC/ROC
auc_roc_2 <- roc(test_data$diagnosis, predicted_probs_M)$auc
```

```
## Setting levels: control = B, case = M
## Setting direction: controls < cases
```

```r
# Print results for Model 2
print("Model 2: Random Forest with 70-30 split")
```

```
## [1] "Model 2: Random Forest with 70-30 split"
```

```r
print(confusion_matrix_2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   B   M
##          B 107   7
##          M   0  56
##
##                Accuracy : 0.9588
##                  95% CI : (0.917, 0.9833)
##     No Information Rate : 0.6294
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9097
##
##  Mcnemar's Test P-Value : 0.02334
##
##             Sensitivity : 1.0000
##             Specificity : 0.8889
##          Pos Pred Value : 0.9386
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6294
##          Detection Rate : 0.6294
##    Detection Prevalence : 0.6706
##       Balanced Accuracy : 0.9444
##
##        'Positive' Class : B
##
```

```r
print(paste("Precision:", precision_2))
```

```
## [1] "Precision: 0.93859649122807"
```

```r
print(paste("F1-score:", f1_score_2))
```

```
## [1] "F1-score: 0.968325791855204"
```

```r
print(paste("AUC/ROC:", auc_roc_2))
```

```
## [1] "AUC/ROC: 0.988206497552292"
```

```r
# Model 3: Random Forest with K-fold cross-validation
trainingRows_rf <- createDataPartition(data_rf$diagnosis, p = 0.7, list =
FALSE)
df_train_rf <- data_rf[trainingRows_rf, ]
df_test_rf <- data_rf[-trainingRows_rf, ]

# Preprocess the training and testing data for Random Forest
train_imp_rf <- preProcess(df_train_rf, method = c("BoxCox", 'center',
'scale'))
train_pre_rf <- predict(train_imp_rf, df_train_rf)
test_pre_rf <- predict(train_imp_rf, df_test_rf)
X_train_rf <- train_pre_rf[, -which(names(train_pre_rf) == "diagnosis")]
y_train_rf <- factor(df_train_rf$diagnosis, levels = c("B", "M"))
X_test_rf <- test_pre_rf[, -which(names(test_pre_rf) == "diagnosis")]
y_test_rf <- factor(df_test_rf$diagnosis, levels = c("B", "M"))

# Random Forest with k-fold cross-validation
rf_model <- train(
  x = X_train_rf,
  y = y_train_rf,
  method = 'rf',
  metric = 'Accuracy',
  trControl = trainControl(
    method = "cv",
    number = 10,
    classProbs = TRUE,
    summaryFunction = twoClassSummary
  )
)
```

```
## Warning in train.default(x = X_train_rf, y = y_train_rf, method = "rf", :
The
## metric "Accuracy" was not in the result set. ROC will be used instead.
```

```r
# Evaluate Model 3
predictions_rf_3 <- predict(rf_model, newdata = X_test_rf)
confusion_matrix_3 <- confusionMatrix(data = predictions_rf_3, reference =
y_test_rf)
precision_3 <- confusion_matrix_3$byClass["Pos Pred Value"]
f1_score_3 <- confusion_matrix_3$byClass["F1"]
auc_roc_3 <- roc(predictions_rf_3, as.numeric(y_test_rf))$auc
```

```
## Setting levels: control = B, case = M
## Setting direction: controls < cases
```

```
sensitivity_3 <- confusion_matrix_3$byClass["Sensitivity"]
specificity_3 <- confusion_matrix_3$byClass["Specificity"]
# Print results for Model 3
print("Model 3: Random Forest with K-fold cross-validation")
```

```
## [1] "Model 3: Random Forest with K-fold cross-validation"
```

```
print(confusion_matrix_3)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   B   M
##          B 103   3
##          M   4  60
##
##                Accuracy : 0.9588
##                  95% CI : (0.917, 0.9833)
##     No Information Rate : 0.6294
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.912
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9626
##             Specificity : 0.9524
##          Pos Pred Value : 0.9717
##          Neg Pred Value : 0.9375
##              Prevalence : 0.6294
##          Detection Rate : 0.6059
##    Detection Prevalence : 0.6235
##       Balanced Accuracy : 0.9575
##
##        'Positive' Class : B
##
```

```
print(paste("Precision:", precision_3))
```

```
## [1] "Precision: 0.971698113207547"
```

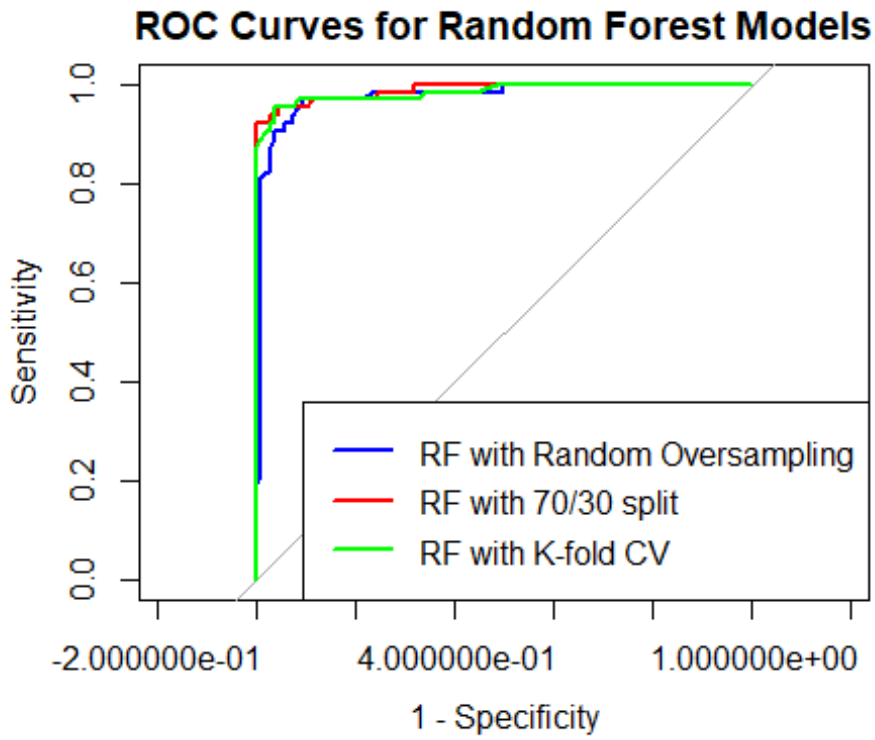
```
print(paste("F1-score:", f1_score_3))
```

```
## [1] "F1-score: 0.967136150234742"
```

```
print(paste("AUC/ROC:", auc_roc_3))
```

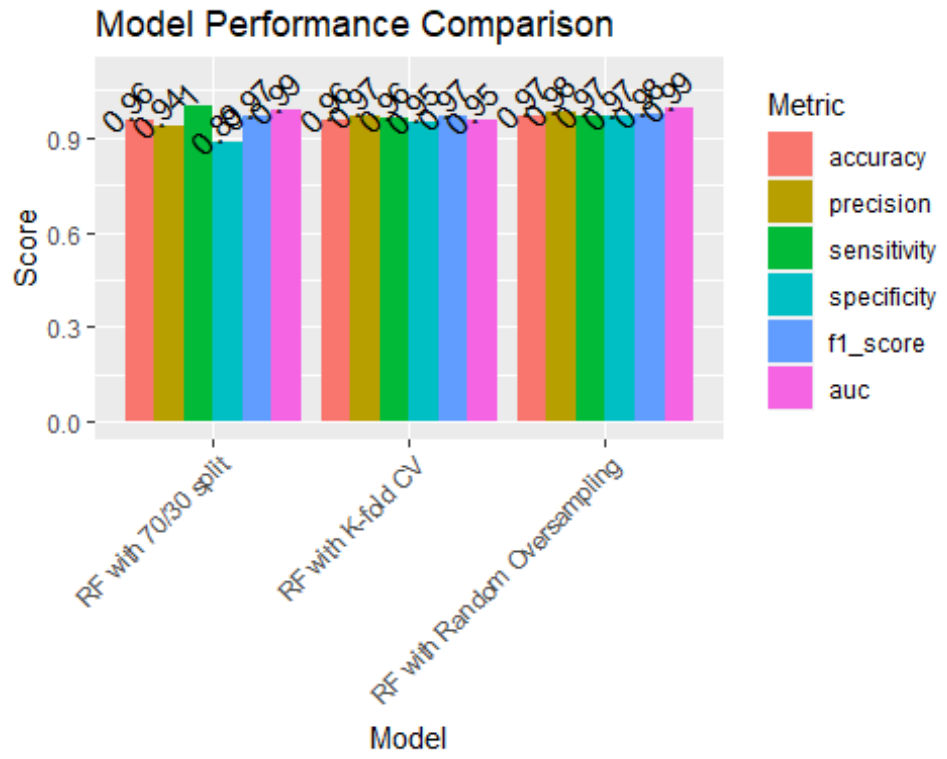
```
## [1] "AUC/ROC: 0.954599056603774"
```

```r
# Get the predicted probabilities for each model
probs_rf_1 <- predict(rf_fit_oversampling, newdata = X_test_rf,
type="prob")[, "M"]
probs_rf_2 <- predicted_probs_M  # You already calculated this
probs_rf_3 <- predict(rf_model, newdata = X_test_rf, type="prob")[, "M"]

# Generate the ROC curves for each model
roc_obj_rf_1 <- roc(y_test_rf, probs_rf_1)

## Setting levels: control = B, case = M

## Setting direction: controls < cases

roc_obj_rf_2 <- roc(test_data$diagnosis, probs_rf_2)

## Setting levels: control = B, case = M
## Setting direction: controls < cases

roc_obj_rf_3 <- roc(y_test_rf, probs_rf_3)

## Setting levels: control = B, case = M
## Setting direction: controls < cases

# Plot the ROC curves
plot(roc_obj_rf_1, col="blue", lwd=2, main="ROC Curves for Random Forest
Models", legacy.axes=TRUE)
lines(roc_obj_rf_2, col="red", lwd=2)
lines(roc_obj_rf_3, col="green", lwd=2)
legend("bottomright", legend=c("RF with Random Oversampling", "RF with 70/30
split", "RF with K-fold CV"),
       col=c("blue", "red", "green"), lwd=2)
```




```r
# Construct a data frame
performance_df <- data.frame(
  model = c("RF with Random Oversampling", "RF with 70/30 split", "RF with
K-fold CV"),
  accuracy = c(confusion_matrix_1$overall["Accuracy"],
confusion_matrix_2$overall["Accuracy"],
confusion_matrix_3$overall["Accuracy"]),
  precision = c(precision_1, precision_2, precision_3),
  sensitivity = c(sensitivity_1, sensitivity_2, sensitivity_3),
  specificity = c(specificity_1, specificity_2, specificity_3),
  f1_score = c(f1_score_1, f1_score_2, f1_score_3),
  auc = c(auc_roc_1, auc_roc_2, auc_roc_3)
)

# Melt the data frame to long format for plotting

performance_melted_df <- melt(performance_df, id.vars = "model")

# Plot the metrics
ggplot(performance_melted_df, aes(x = model, y = value, fill = variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(aes(label = round(value, 2)), position = position_dodge(width =
0.9), vjust = -0.25, angle = 45) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(y = "Score", x = "Model", fill = "Metric") +
```

```
  ggtitle("Model Performance Comparison") +
  ylim(0, max(performance_melted_df$value) + 0.1)
```

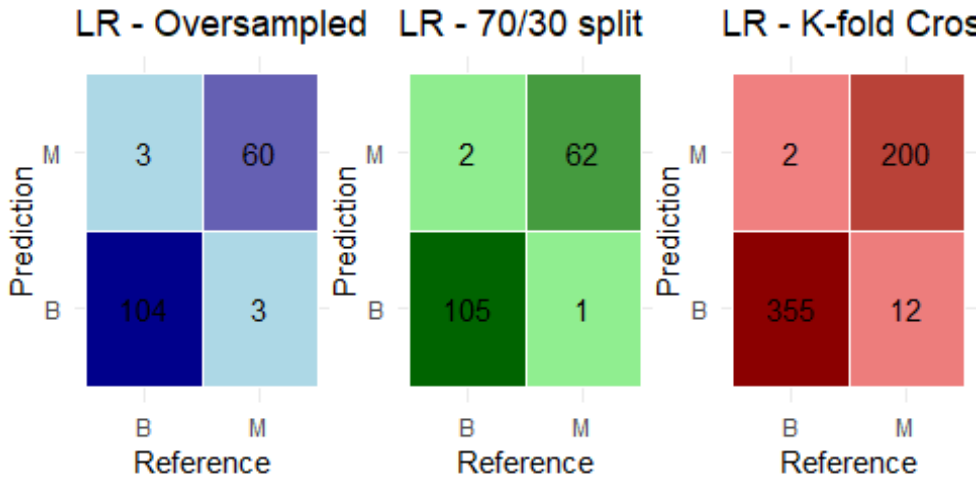



```
# Construct a data frame
performance_df <- data.frame(
  model = c(rep("RF with Random Oversampling", 6), rep("RF with 70/30 split",
6), rep("RF with K-fold CV", 6)),
  metric = c(rep(c("accuracy", "precision", "sensitivity", "specificity",
"f1_score", "auc"), 3)),
  score = c(confusion_matrix_1$overall["Accuracy"], precision_1,
sensitivity_1, specificity_1, f1_score_1, auc_roc_1,
            confusion_matrix_2$overall["Accuracy"], precision_2,
sensitivity_2, specificity_2, f1_score_2, auc_roc_2,
            confusion_matrix_3$overall["Accuracy"], precision_3,
sensitivity_3, specificity_3, f1_score_3, auc_roc_3)
)

# Plot the metrics
ggplot(performance_df, aes(x = metric, y = score, fill = model)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(aes(label = round(score, 2)), position = position_dodge(width =
0.9), vjust = -0.25, angle = 45, hjust = 0.5) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(y = "Score", x = "Metric", fill = "Model") +
  ggtitle("Model Performance Comparison") +
  ylim(0, max(performance_df$score) + 0.1)
```


```
# LR confusion matrix plots
# Convert the confusion matrices to data frames
confusion_matrix_1_lr <- confusionMatrix(log_reg_fit, norm = 'none')
confusion_matrix_2_lr <- confusionMatrix(data = predictions_lr_3, reference =
y_test_factor_3)
confusion_matrix_3_lr <- confusionMatrix(data = predictions_lr, reference =
y_test_factor_lr)
df1_lr <- as.data.frame(as.table(confusion_matrix_1_lr$table))
df2_lr <- as.data.frame(as.table(confusion_matrix_2_lr$table))
df3_lr <- as.data.frame(as.table(confusion_matrix_3_lr$table))

# Generate the plots with different color schemes and without legend
plot1_lr <- ggplot(data = df3_lr, aes(x = Reference, y = Prediction)) +
         geom_tile(aes(fill = Freq), color = "white") +
         geom_text(aes(label = Freq), color = "black") +
         scale_fill_gradient(low = "lightblue", high = "darkblue") +
         theme_minimal() +
         labs(title = "LR - Oversampled") +
         guides(fill = FALSE) +
         theme(plot.margin = unit(c(2, 0, 2, 0), "cm"))

## Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use
"none" instead as
## of ggplot2 3.3.4.
## This warning is displayed once every 8 hours.
```

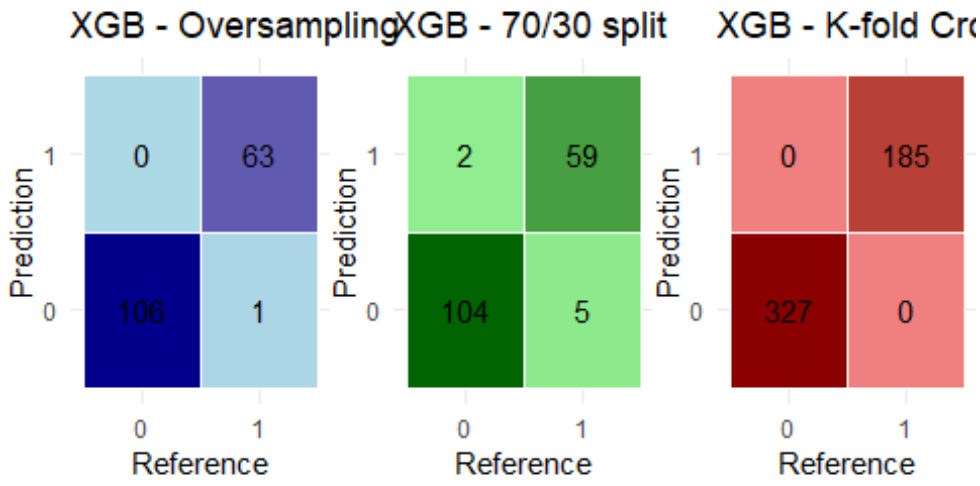
```
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
plot2_lr <- ggplot(data = df2_lr, aes(x = Reference, y = Prediction)) +
          geom_tile(aes(fill = Freq), color = "white") +
          geom_text(aes(label = Freq), color = "black") +
          scale_fill_gradient(low = "lightgreen", high = "darkgreen") +
          theme_minimal() +
          labs(title = "LR - 70/30 split") +
          guides(fill = FALSE) +
          theme(plot.margin = unit(c(2, 0, 2, 0), "cm"))

plot3_lr <- ggplot(data = df1_lr, aes(x = Reference, y = Prediction)) +
          geom_tile(aes(fill = Freq), color = "white") +
          geom_text(aes(label = Freq), color = "black") +
          scale_fill_gradient(low = "lightcoral", high = "darkred") +
          theme_minimal() +
          labs(title = "LR - K-fold Cross Val") +
          guides(fill = FALSE) +
          theme(plot.margin = unit(c(2, 0, 2, 0), "cm"))

# Create a grid with all three plots
grid_lr <- grid.arrange(plot1_lr, plot2_lr, plot3_lr, nrow = 1)
```



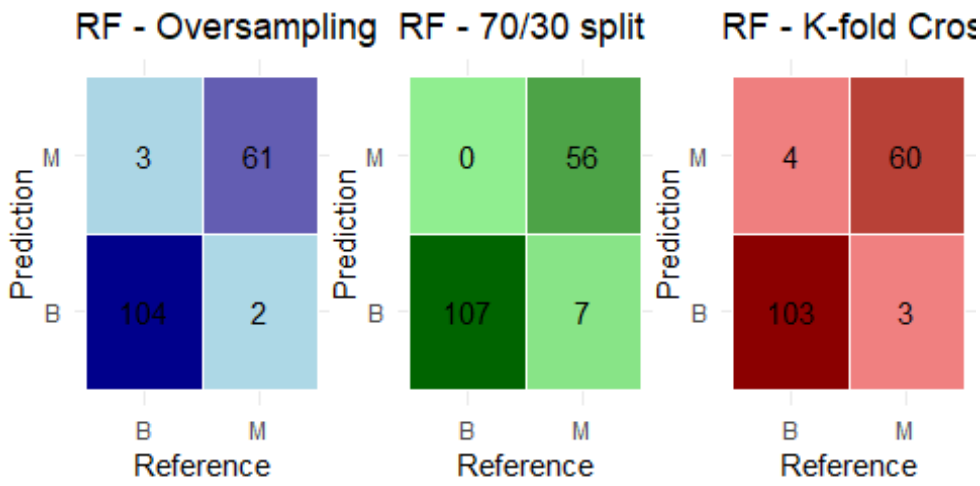

```
# Print the grid
print(grid_lr)
```

```
## TableGrob (1 x 3) "arrange": 3 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
## 3 3 (1-1,3-3) arrange gtable[layout]

# Convert the confusion matrices to data frames
os_CM_df <- as.data.frame(as.table(os_CM$table))
rs_CM_df  <- as.data.frame(as.table(rs_CM$table))
cv_CM_df <- as.data.frame(as.table(cv_CM$table))
# Generate the plots with different color schemes and without legend
os_CM_plot1 <- ggplot(data = os_CM_df, aes(x = Reference, y = Prediction)) +
         geom_tile(aes(fill = Freq), color = "white") +
         geom_text(aes(label = Freq), color = "black") +
         scale_fill_gradient(low = "lightblue", high = "darkblue") +
         theme_minimal() +
         labs(title = "XGB - Oversampling") +
         guides(fill = FALSE) +
         theme(plot.margin = unit(c(2, 0, 2, 0), "cm"))
rs_CM_plot2 <- ggplot(data = rs_CM_df, aes(x = Reference, y = Prediction)) +
         geom_tile(aes(fill = Freq), color = "white") +
         geom_text(aes(label = Freq), color = "black") +
         scale_fill_gradient(low = "lightgreen", high = "darkgreen") +
         theme_minimal() +
         labs(title = "XGB - 70/30 split") +
         guides(fill = FALSE) +
         theme(plot.margin = unit(c(2, 0, 2, 0), "cm"))
cv_CM_plot3 <- ggplot(data = cv_CM_df, aes(x = Reference, y = Prediction)) +
         geom_tile(aes(fill = Freq), color = "white") +
         geom_text(aes(label = Freq), color = "black") +
         scale_fill_gradient(low = "lightcoral", high = "darkred") +
         theme_minimal() +
         labs(title = "XGB - K-fold Cross-Val") +
         guides(fill = FALSE) +
         theme(plot.margin = unit(c(2, 0, 2, 0), "cm"))
# Create a grid with all three plots
xgb_grid <- grid.arrange(os_CM_plot1, rs_CM_plot2, cv_CM_plot3, nrow = 1)
```




```r
# Print the grid
print(xgb_grid)

## TableGrob (1 x 3) "arrange": 3 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
## 3 3 (1-1,3-3) arrange gtable[layout]

# Convert the confusion matrices to data frames
df1 <- as.data.frame(as.table(confusion_matrix_1$table))
df2 <- as.data.frame(as.table(confusion_matrix_2$table))
df3 <- as.data.frame(as.table(confusion_matrix_3$table))

# Generate the plots with different color schemes and without legend
plot1 <- ggplot(data = df1, aes(x = Reference, y = Prediction)) +
         geom_tile(aes(fill = Freq), color = "white") +
         geom_text(aes(label = Freq), color = "black") +
         scale_fill_gradient(low = "lightblue", high = "darkblue") +
         theme_minimal() +
         labs(title = "RF - Oversampling") +
         guides(fill = FALSE) +
         theme(plot.margin = unit(c(2, 0, 2, 0), "cm"))

plot2 <- ggplot(data = df2, aes(x = Reference, y = Prediction)) +
         geom_tile(aes(fill = Freq), color = "white") +
```

```
          geom_text(aes(label = Freq), color = "black") +
          scale_fill_gradient(low = "lightgreen", high = "darkgreen") +
          theme_minimal() +
          labs(title = "RF - 70/30 split") +
          guides(fill = FALSE) +
          theme(plot.margin = unit(c(2, 0, 2, 0), "cm"))

plot3 <- ggplot(data = df3, aes(x = Reference, y = Prediction)) +
          geom_tile(aes(fill = Freq), color = "white") +
          geom_text(aes(label = Freq), color = "black") +
          scale_fill_gradient(low = "lightcoral", high = "darkred") +
          theme_minimal() +
          labs(title = "RF - K-fold Cross-Val") +
          guides(fill = FALSE) +
          theme(plot.margin = unit(c(2, 0, 2, 0), "cm"))

# Create a grid with all three plots
grid <- grid.arrange(plot1, plot2, plot3, nrow = 1)
```




```
# Print the grid
print(grid)

## TableGrob (1 x 3) "arrange": 3 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
```

```
## 2 2 (1-1,2-2) arrange gtable[layout]
## 3 3 (1-1,3-3) arrange gtable[layout]
```