# 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\_project/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)

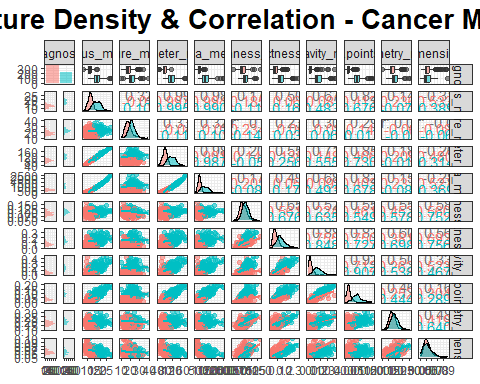
There appears to be no degenerate variables.

#### 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]

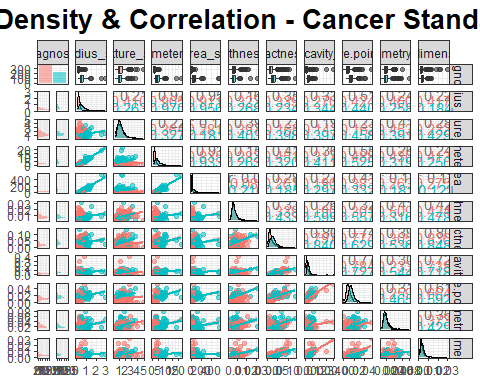
# Prepare the dataframe   
df\_clean <- df[, !(names(df) %in% c("X", "id"))]  
  
# Create empty data frames  
df\_mean <- data.frame(diagnosis = df\_clean$diagnosis)  
df\_se <- data.frame(diagnosis = df\_clean$diagnosis)  
df\_worst <- data.frame(diagnosis = df\_clean$diagnosis)  
  
# Loop over variable 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,size=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`.



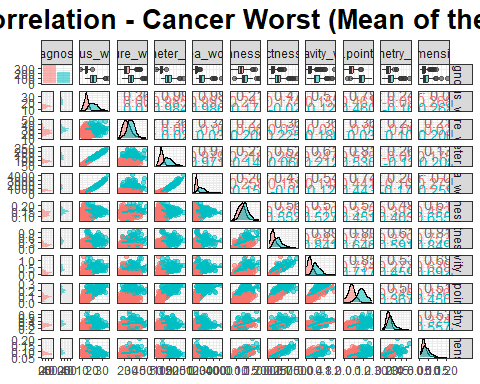
# Visualization for the "Standard Error" Variables  
ggpairs(df\_se, aes(color=diagnosis, alpha=0.75), lower=list(continuous="smooth"))+ theme\_bw()+  
 labs(title="Feature Density & Correlation - Cancer Standard Error")+theme(plot.title=element\_text(face='bold',color='black',hjust=0.5,size=20))

## `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`.

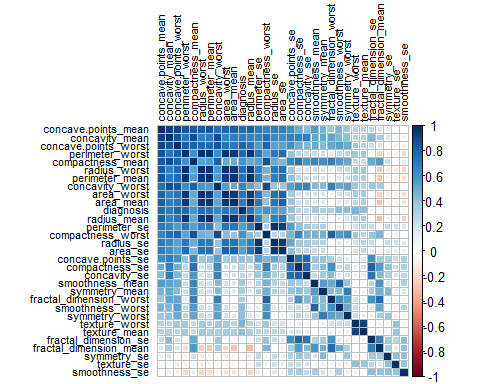


# Visualization for the "Worst" Variables  
ggpairs(df\_worst, aes(color=diagnosis, alpha=0.75), lower=list(continuous="smooth"))+ theme\_bw()+  
 labs(title="Feature Density & Correlation - Cancer Worst (Mean of the three largest values)")+theme(plot.title=element\_text(face='bold',color='black',hjust=0.5,size=20))

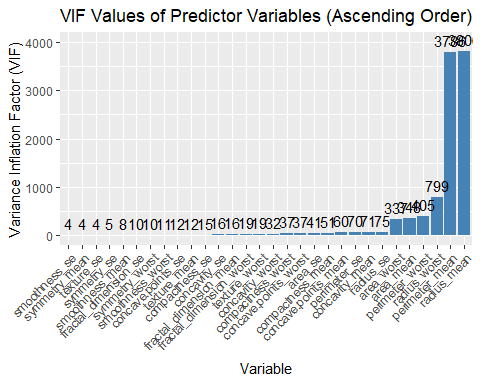
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` 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`.



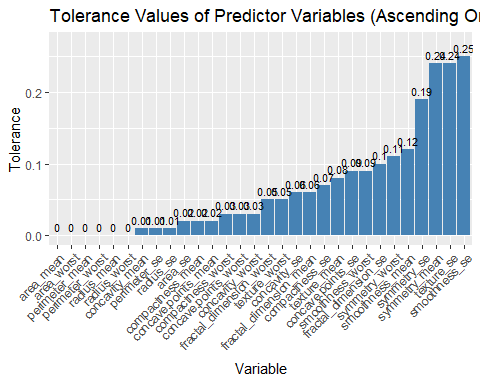
# Pearson Correlation of all features  
df\_clean$diagnosis <- as.integer(factor(df\_clean$diagnosis))-1  
correlations <- cor(df\_clean,method="pearson")  
corrplot(correlations, number.cex = .9, method = "square",   
 hclust.method = "ward", order = "FPC",  
 type = "full", tl.cex=0.7,tl.col = "black")



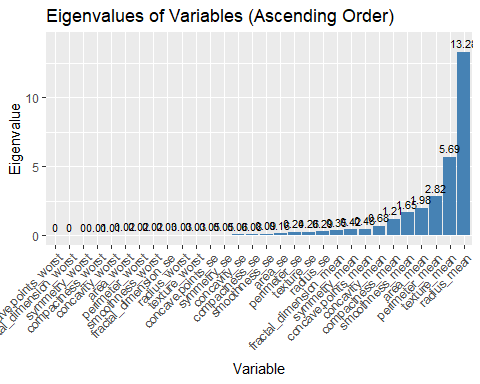
# Calculate the VIF values (using linear model) for the predictor variables to look for multicollinearity  
# VIF below 5 are generally considered acceptable, 5-10 suggest moderate multicollinearity, and above 10 show high multicollinearity  
lm\_model <- lm(diagnosis ~ ., data = df\_clean)  
vif\_values <- 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)



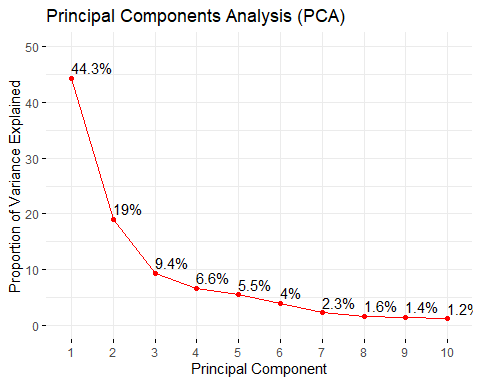
### Calculate the tolerance (reciprocal of VIF) for the predictor variables to look for multicollinearity   
# Generally, a tolerance value less than 0.1 or 0.2 is often considered indicative of multicollinearity.  
tolerance <- 1 / vif\_values  
tolerance\_table <- data.frame(Variable = names(tolerance), 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



# 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

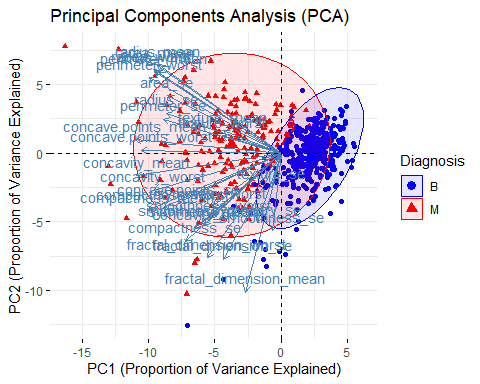


### Principal Components Analysis (PCA) transform  
raw.data <- read.csv('C:/MIDS/ADS-503\_Applied\_Predictive\_Modeling/ADS\_503\_team\_2\_final\_project/data/breast\_cancer\_FNA\_data.csv')  
var\_only <- df\_clean[, !(names(df\_clean) %in% c("diagnosis"))] # predictor variables only  
diagnosis <- 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")



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

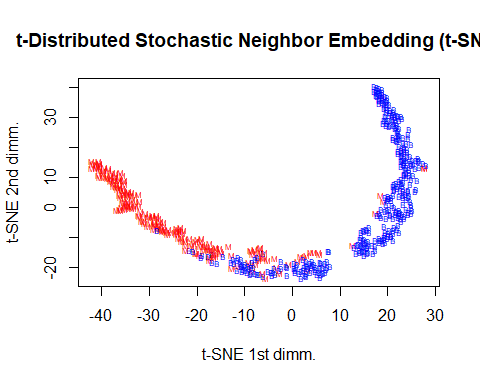
# Create PCA biplot with customized aesthetics  
pca\_biplot <- fviz\_pca\_biplot(var\_pca,  
 geom.ind = "point",  
 col.ind = diagnosis,  
 palette = c("blue", "red"),  
 addEllipses = TRUE,  
 axes.linetype = "dashed",  
 title = "Principal Components Analysis (PCA)",  
 xlab = "PC1 (Proportion of 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])

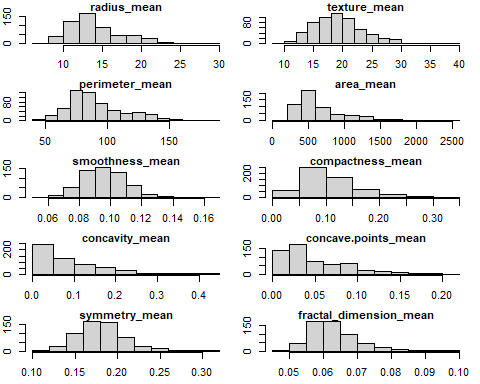


#### df\_var data frame for visualizing just predictors

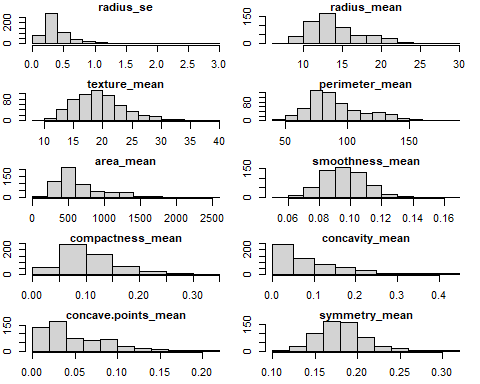
remove <- c("diagnosis")  
df\_var <- df[, !(colnames(df) %in% remove)]

#### Histograms of all predictor variables

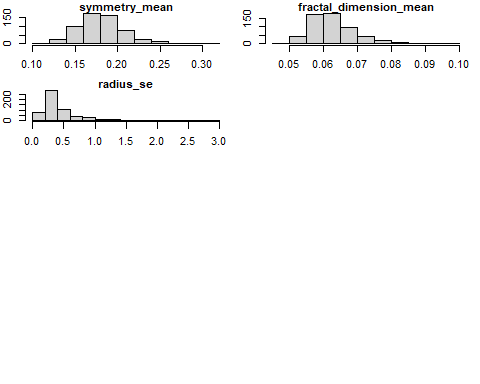
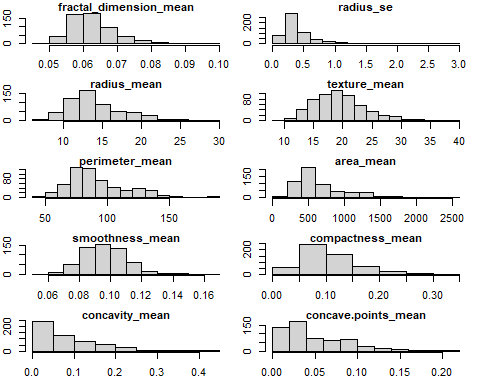
par(mfrow = c(5, 2))  
par(mar = c(2.5, 2, 1, 1))  
par(cex.main = 1)  
  
for (i in 1:ncol(df\_mean)) {  
 hist(df\_var[, i], xlab = names(df\_var)[i], main = paste(names(df\_var)[i]))  
}



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]))  
}

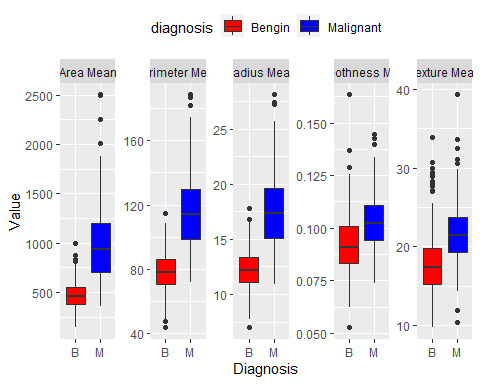


# build a boxplot to identity the outliers that were produced in the summary statistics  
# split the data into groups of 5 and create 6 graphs for the plots since there are 30 features and 1 target  
# data will be split into variables bx\_1, bx\_2, bx\_3, bx\_4, bx\_5 for boxplot purposes  
colors <- 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]

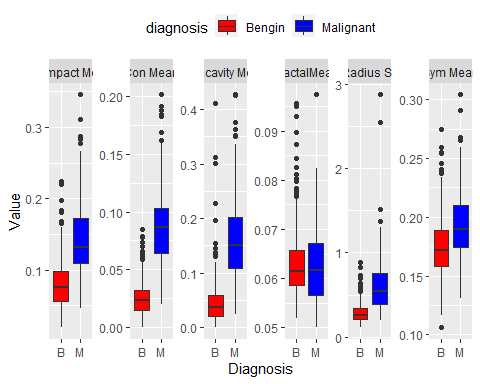
All of these predictors have outliers however the predictor with the most are texture\_mean, area\_mean ad smoothness mean.

For bx\_2 all of these also have outliers and I would say they all have some pretty significant outliers.

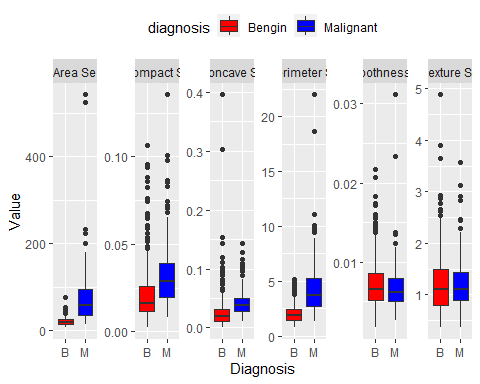
# Create a data frame for plotting  
plot\_data\_1 <- data.frame(  
 value = c(bx\_1$radius\_mean, bx\_1$texture\_mean, bx\_1$perimeter\_mean, bx\_1$smoothness\_mean, bx\_1$area\_mean),  
 diagnosis = rep(df$diagnosis, 5),  
 feature = rep(c("Radius Mean", "Texture Mean", "Perimeter Mean", "Smoothness 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")



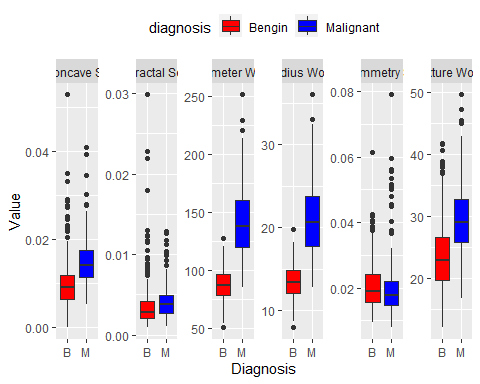
# Create a data frame for plotting  
plot\_data\_2 <- data.frame(  
 value = c(bx\_2$fractal\_dimension\_mean,bx\_2$symmetry\_mean, bx\_2$radius\_se, bx\_2$concave.points\_mean, bx\_2$concavity\_mean,bx\_2$compactness\_mean ),  
 diagnosis = rep(df$diagnosis, 6),  
 feature = rep(c("FractalMean", "Sym 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")



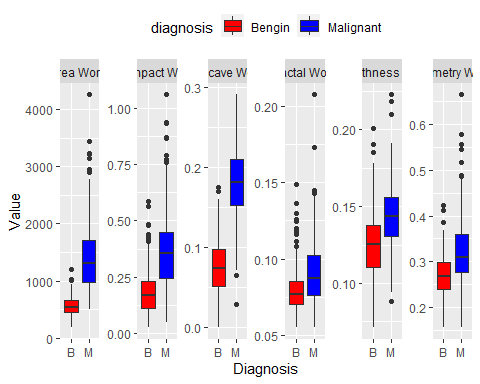
# Create a data frame for plotting  
plot\_data\_3 <- data.frame(  
 value = c(bx\_3$texture\_se,bx\_3$perimeter\_se, bx\_3$area\_se, bx\_3$smoothness\_se, bx\_3$compactness\_se, bx\_3$concavity\_se),  
 diagnosis = rep(df$diagnosis, 6),  
 feature = rep(c("Texture Se", "Perimeter Se", "Area Se", "Smoothness 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")



# 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")

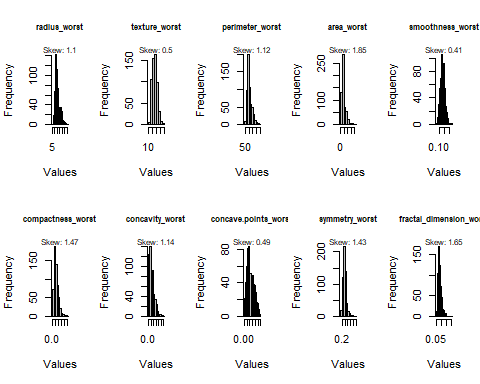
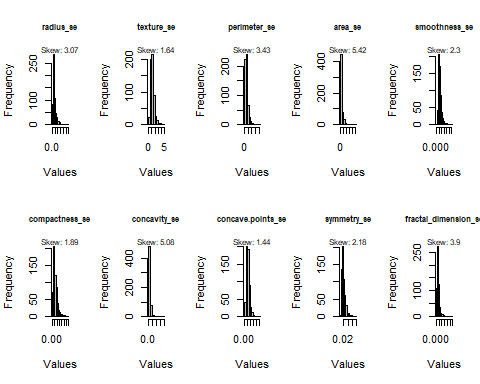
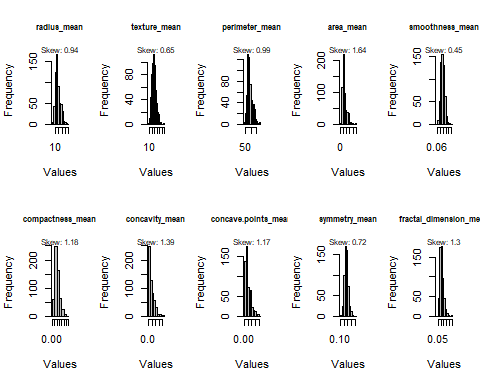


# Create a data frame for plotting  
plot\_data\_5 <- data.frame(  
 value = c(bx\_5$area\_worst ,bx\_5$smoothness\_worst, bx\_5$compactness\_worst, bx\_5$concave.points\_worst, bx\_5$symmetry\_worst, bx\_5$fractal\_dimension\_worst),  
 diagnosis = rep(df$diagnosis, 6),  
 feature = rep(c("Area Worst", "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))  
}



#### 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

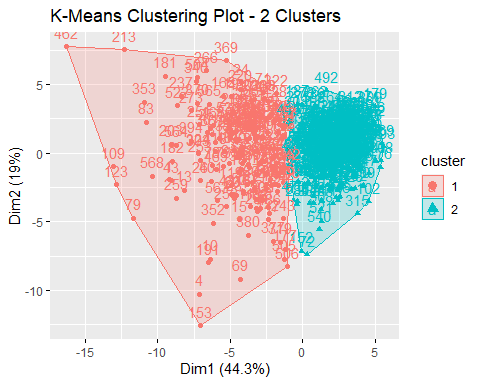
# Convert the skewness list to a dataframe  
skewness\_df <- data.frame(Skewness = unlist(skewness\_list))  
  
# Print the skewness dataframe  
print(skewness\_df)

## 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

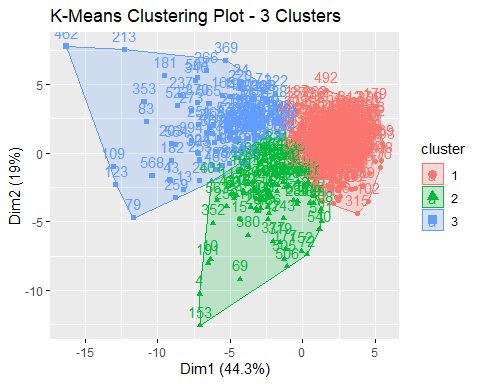
2 or 3 principal components. revisit & decide later.

#### 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



#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  
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

# 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 -0.018084005  
## 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

#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  
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)  
  
# 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   
##

# 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

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"

# 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  
  
# 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

# Calculate precision  
os\_prec <- os\_CM$byClass["Pos Pred Value"]  
print(paste("Precision:", os\_prec))

## [1] "Precision: 0.990654205607477"

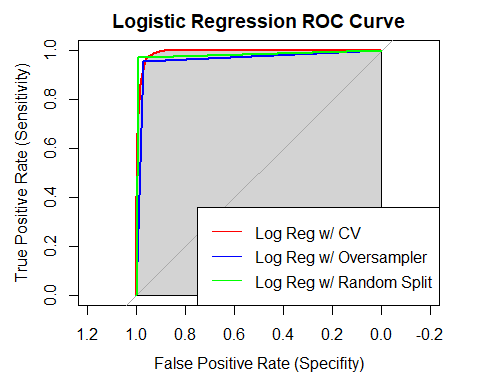
# Calulate F1-score  
os\_f1 <- 2 \* (os\_prec \* os\_sens) / (os\_prec + os\_sens)  
print(paste("F1-score:", os\_f1))

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

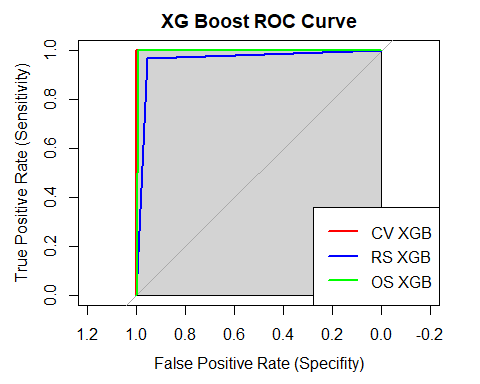
# Calculate AUC/ROC score  
os\_auc <- os\_roc\_results\_xgb$auc  
print(paste("AUC/ROC score:", os\_auc))

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

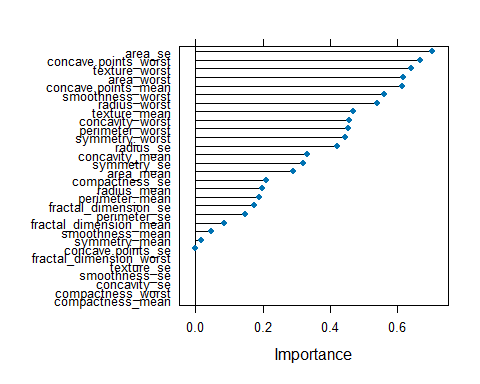
# plot the ROC curve for each of the 3 Lr models  
plot(cv\_lr\_roc, col = "red", main = "Logistic Regression ROC Curve", xlab = "False Positive Rate (Specifity)", ylab = "True Positive Rate (Sensitivity)", print.auc = FALSE, auc.polygon = TRUE, auc.polygon.col = "lightgray")  
lines(roc\_results\_lr, 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)



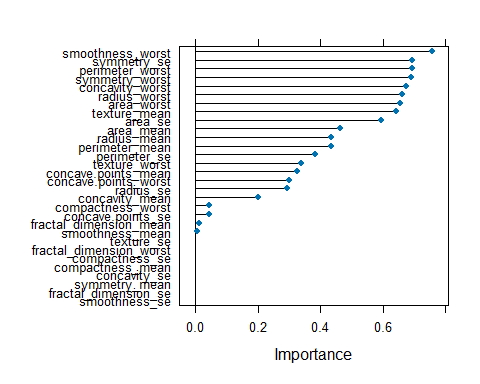
# plot the ROC curve for each of the 3 XBoost models  
plot(cv\_roc\_results\_xgb, col = "red", main = "XG Boost ROC Curve", xlab = "False Positive Rate (Specifity)", ylab = "True Positive Rate (Sensitivity)", print.auc = FALSE, auc.polygon = TRUE, auc.polygon.col = "lightgray")  
lines(rs\_roc\_results\_xgb, 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)



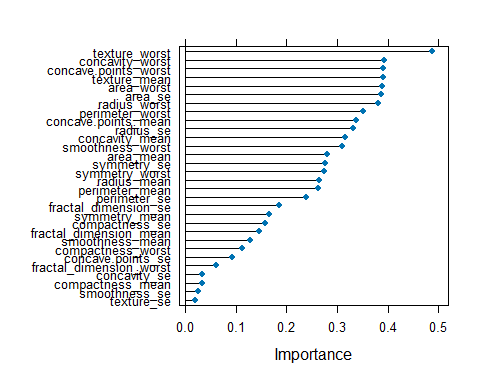
lr\_1 <- varImp(log\_reg\_fit, scale = FALSE)  
lr\_2 <- varImp(log\_reg\_fit\_2, scale = FALSE)  
lr\_3 <- varImp(log\_reg\_fit\_3, scale = FALSE)  
plot(lr\_1)



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\_project/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"

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")  
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

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

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

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"]  
  
# 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

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

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

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   
##

print(paste("Precision:", precision\_2))

## [1] "Precision: 0.93859649122807"

print(paste("F1-score:", f1\_score\_2))

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

print(paste("AUC/ROC:", auc\_roc\_2))

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

# 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.

# 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"

# 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

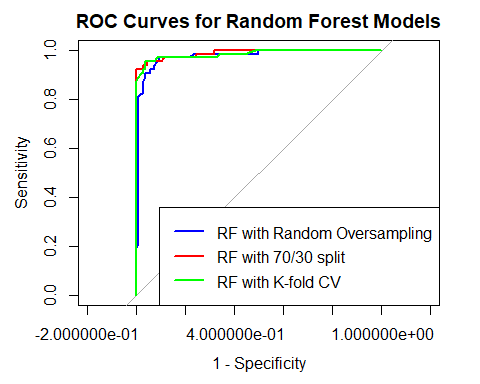
roc\_obj\_rf\_2 <- roc(test\_data$diagnosis, probs\_rf\_2)

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

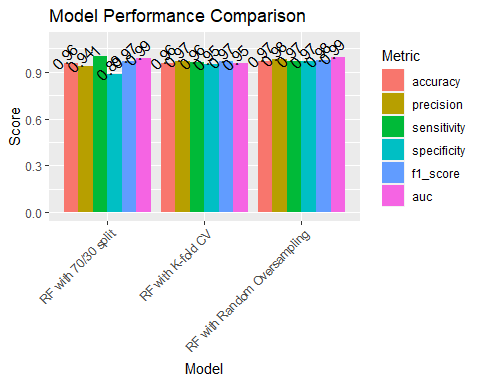
roc\_obj\_rf\_3 <- roc(y\_test\_rf, probs\_rf\_3)

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

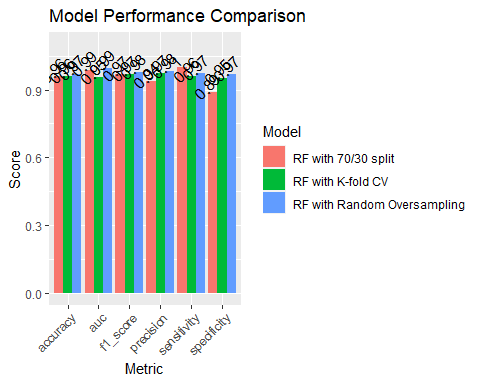
# Plot the ROC curves  
plot(roc\_obj\_rf\_1, col="blue", lwd=2, main="ROC Curves for Random Forest Models", legacy.axes=TRUE)  
lines(roc\_obj\_rf\_2, col="red", lwd=2)  
lines(roc\_obj\_rf\_3, col="green", lwd=2)  
legend("bottomright", legend=c("RF with Random Oversampling", "RF with 70/30 split", "RF with K-fold CV"),   
 col=c("blue", "red", "green"), lwd=2)



# 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)



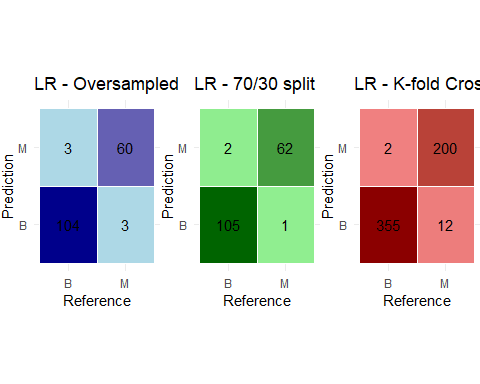
# Construct a data frame  
performance\_df <- data.frame(  
 model = c(rep("RF with Random Oversampling", 6), rep("RF with 70/30 split", 6), rep("RF with K-fold CV", 6)),  
 metric = c(rep(c("accuracy", "precision", "sensitivity", "specificity", "f1\_score", "auc"), 3)),  
 score = c(confusion\_matrix\_1$overall["Accuracy"], precision\_1, sensitivity\_1, specificity\_1, f1\_score\_1, auc\_roc\_1,  
 confusion\_matrix\_2$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.

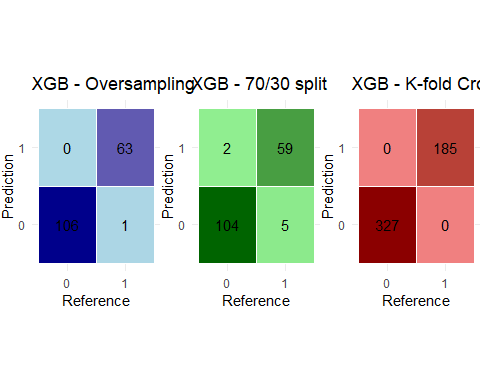
plot2\_lr <- ggplot(data = df2\_lr, aes(x = Reference, y = Prediction)) +  
 geom\_tile(aes(fill = Freq), color = "white") +  
 geom\_text(aes(label = Freq), color = "black") +  
 scale\_fill\_gradient(low = "lightgreen", high = "darkgreen") +  
 theme\_minimal() +  
 labs(title = "LR - 70/30 split") +  
 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]

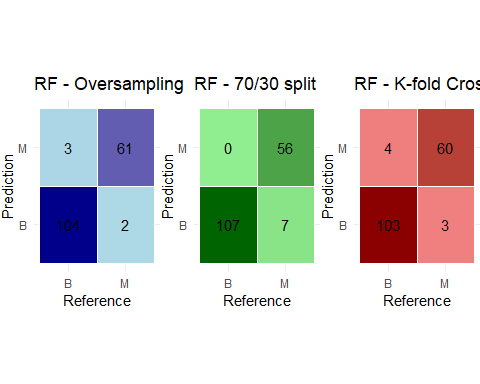
# Convert the confusion matrices to data frames  
os\_CM\_df <- as.data.frame(as.table(os\_CM$table))  
rs\_CM\_df <- as.data.frame(as.table(rs\_CM$table))  
cv\_CM\_df <- as.data.frame(as.table(cv\_CM$table))  
# Generate the plots with different color schemes and without legend  
os\_CM\_plot1 <- ggplot(data = 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)



# 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]