IDS-Final Project

Avisek Choudhury, Aldo Adriazola, Kait Arnold

3/08/2020

Dataset

The physicians have identified a data set that consists of over 500 measurements from Fine Needle Aspiration (FNA) of breast tissue masses. In an FNA, a small needle is used to extract a sample of cells from a tissue mass. The cells are then photographed under a microscope. The resulting photographs are entered into graphical imaging software. A trained technician uses a mouse pointer to draw the boundary of the nuclei. The software then calculates each of ten characteristics for the nuclei. This process is repeated for most or all of the nuclei in the sample.

The data consists of measurements of the cell nuclei for the following characteristics:

1. radius
2. texture
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter^2 / area - 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension ("coastline approximation" - 1)

Measurements of these ten characteristics are summarized for all cells in the sample. The dataset consists of the mean, standard error of the mean, and maximum of the 10 characteristics, for a total of 30 observations for each. Additionally, the data set includes an identification number and a variable that indicates if the tissue mass is malignant (M) or benign (B).

```
# Load the necessary libraries
library(tidyverse)
library(class)
library(caret)
library(rpart)
library(partykit)
library(randomForest)
library(readr)
library(e1071)
```

1. Download the data from NeXus: FNA_cancer.csv

```
#Load the dataset that was previously downloaded
#cancer_df <- read_csv('C:/MSDS/Spring 2020/IDS/Project/FNA_cancer.csv')

# Path for Kait to download data
```

```r
cancer_df <- read_csv("FNA_cancer.csv")

# Print the dataset
head(cancer_df)
```

```
## # A tibble: 6 x 32
##       id diagnosis radius_mean texture_mean perimeter_mean area_mean
##    <dbl> <chr>           <dbl>        <dbl>          <dbl>     <dbl>
## 1 8.42e5 M                18.0         10.4          123.      1001
## 2 8.43e5 M                20.6         17.8          133.      1326
## 3 8.43e7 M                19.7         21.2          130       1203
## 4 8.43e7 M                11.4         20.4           77.6      386.
## 5 8.44e7 M                20.3         14.3          135.      1297
## 6 8.44e5 M                12.4         15.7           82.6      477.
## # ... with 26 more variables: smoothness_mean <dbl>, compactness_mean <dbl>,
## #   concavity_mean <dbl>, `concave points_mean` <dbl>, symmetry_mean <dbl>,
## #   fractal_dimension_mean <dbl>, radius_se <dbl>, texture_se <dbl>,
## #   perimeter_se <dbl>, area_se <dbl>, smoothness_se <dbl>,
## #   compactness_se <dbl>, concavity_se <dbl>, `concave points_se` <dbl>,
## #   symmetry_se <dbl>, fractal_dimension_se <dbl>, radius_worst <dbl>,
## #   texture_worst <dbl>, perimeter_worst <dbl>, area_worst <dbl>,
## #   smoothness_worst <dbl>, compactness_worst <dbl>, concavity_worst <dbl>,
## #   `concave points_worst` <dbl>, symmetry_worst <dbl>,
## #   fractal_dimension_worst <dbl>
```

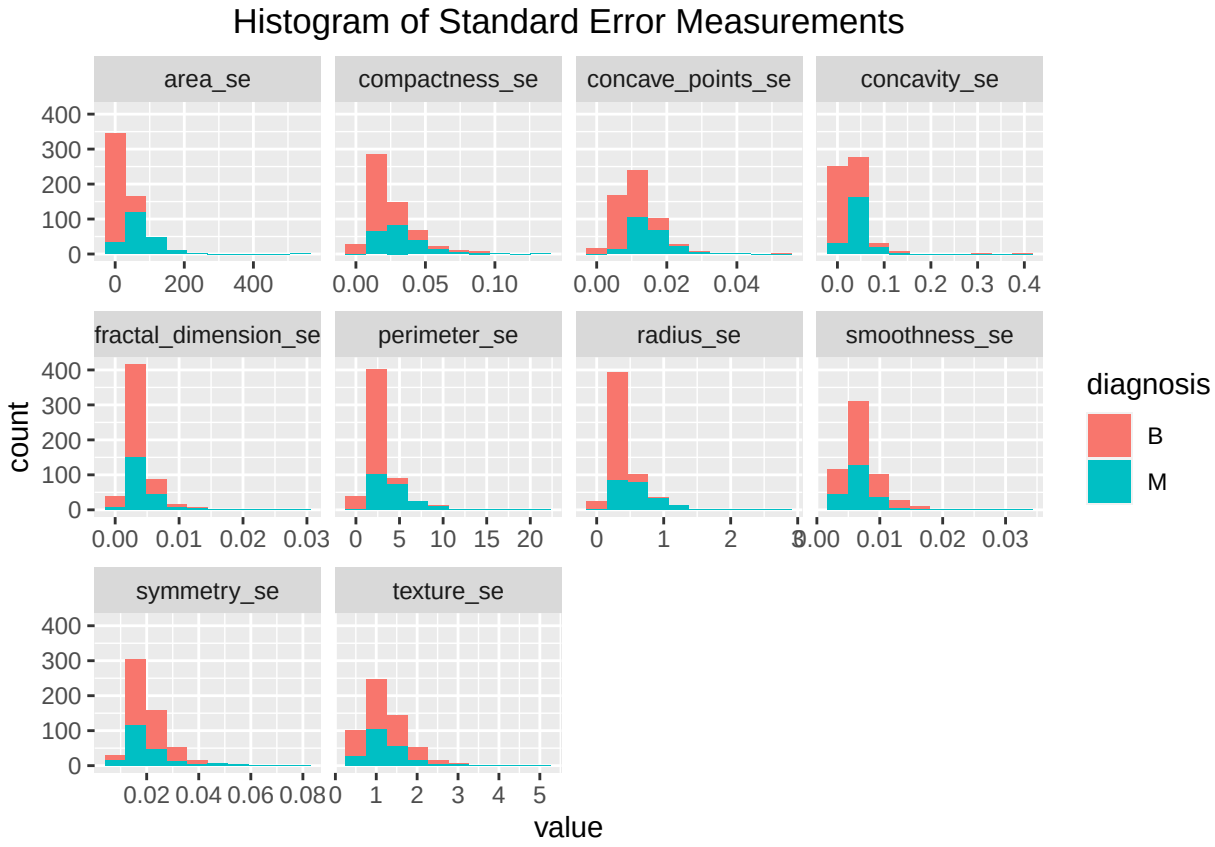
2. Perform basic exploratory data analysis.

Exploratory data analysis (EDA)

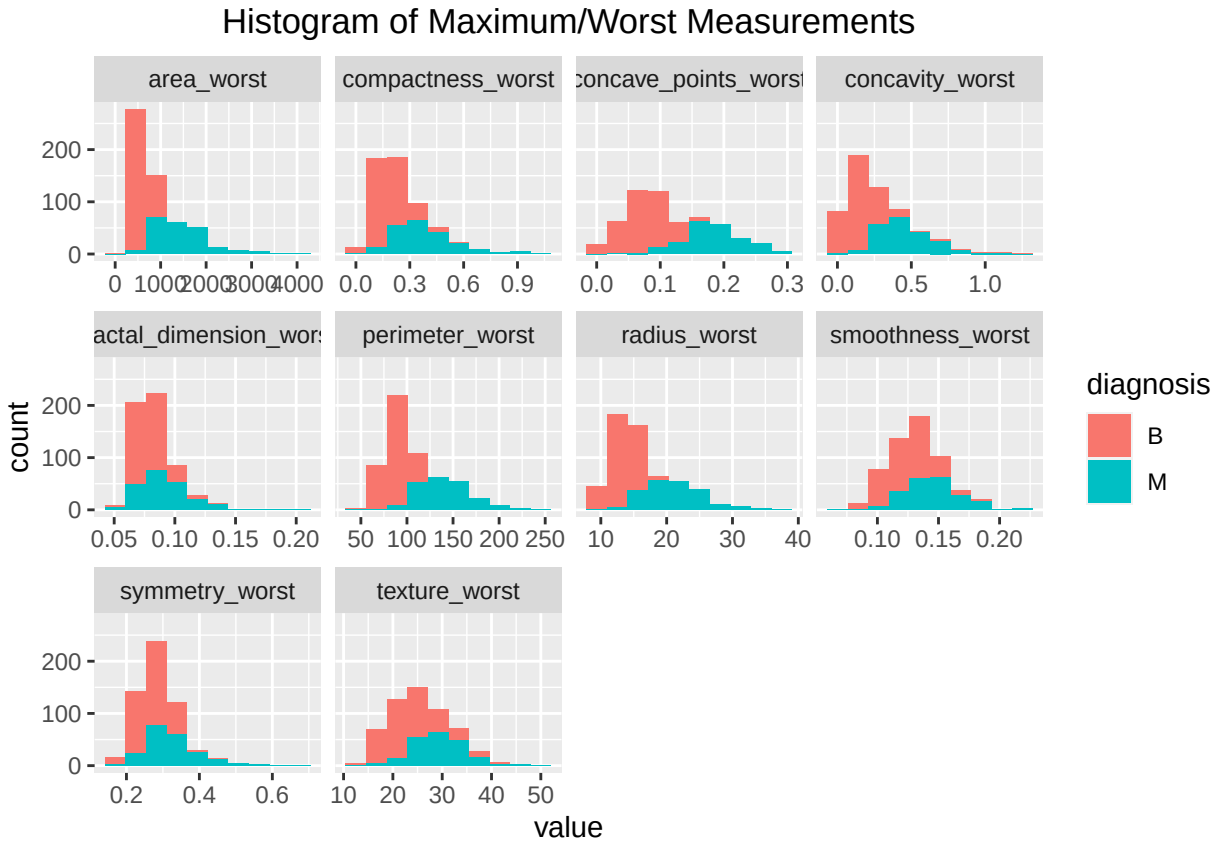
```r
# Make the Diagnosis as Factor
cancer_df$diagnosis <- as.factor(cancer_df$diagnosis)

# Fix the column names for consistency
colnames(cancer_df) <- str_replace(colnames(cancer_df), ' ', '_')
```

```r
# create a histogram for each of the mean measurements to visualize the
# distribution of each field. we will color by the diagnosis to see the distribution
#of each field's mean broken down by diagnosis, B and M.
ggplot(cancer_df[ , c(2:12)] %>%
         pivot_longer(cols = radius_mean:fractal_dimension_mean),
       aes(value, fill = diagnosis)) +
  geom_histogram(bins = 10) +  ggtitle("Histogram of Mean Measurements") +
  theme(plot.title = element_text(hjust = 0.5)) +
  facet_wrap(~name, scales = 'free_x')
```


```
# create a histogram for the standard error measurements.
# similar to above, we will color by the diagnosis to see the distribution
# of each field's standard error measurements broken down by diagnosis, B and M.
ggplot(cancer_df[ , c(2,13:22)] %>%
         pivot_longer(cols = radius_se:fractal_dimension_se),
       aes(value, fill = diagnosis)) +
  geom_histogram(bins = 10)  +  ggtitle("Histogram of Standard Error Measurements") +
  theme(plot.title = element_text(hjust = 0.5)) +
  facet_wrap(~name, scales = 'free_x')
```



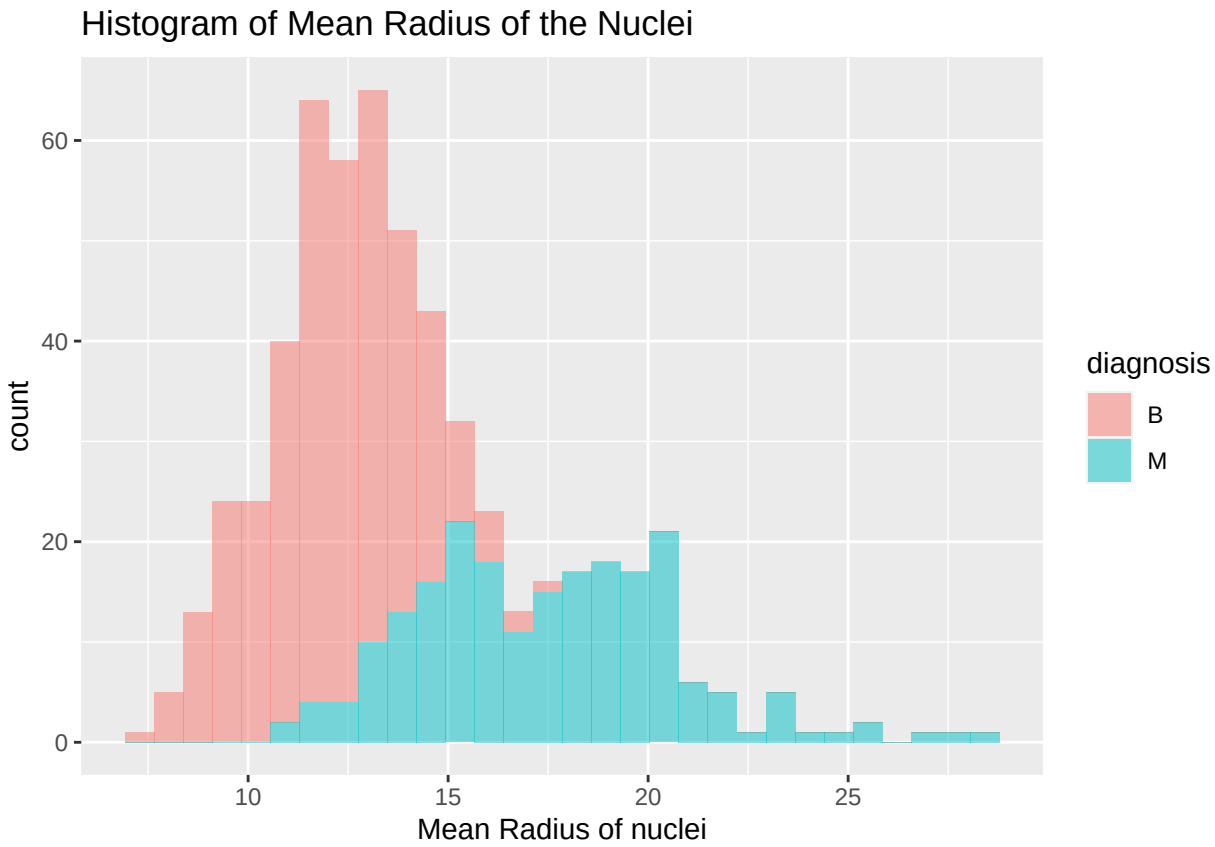

```
# create a histogram for the maximum/worst measurements colored by diagnosis.
ggplot(cancer_df[ , c(2,23:32)] %>%
         pivot_longer(cols = radius_worst:fractal_dimension_worst),
       aes(value, fill = diagnosis)) +
  geom_histogram(bins = 10)  +  ggtitle("Histogram of Maximum/Worst Measurements") +
  theme(plot.title = element_text(hjust = 0.5)) +
  facet_wrap(~name, scales = 'free_x')
```



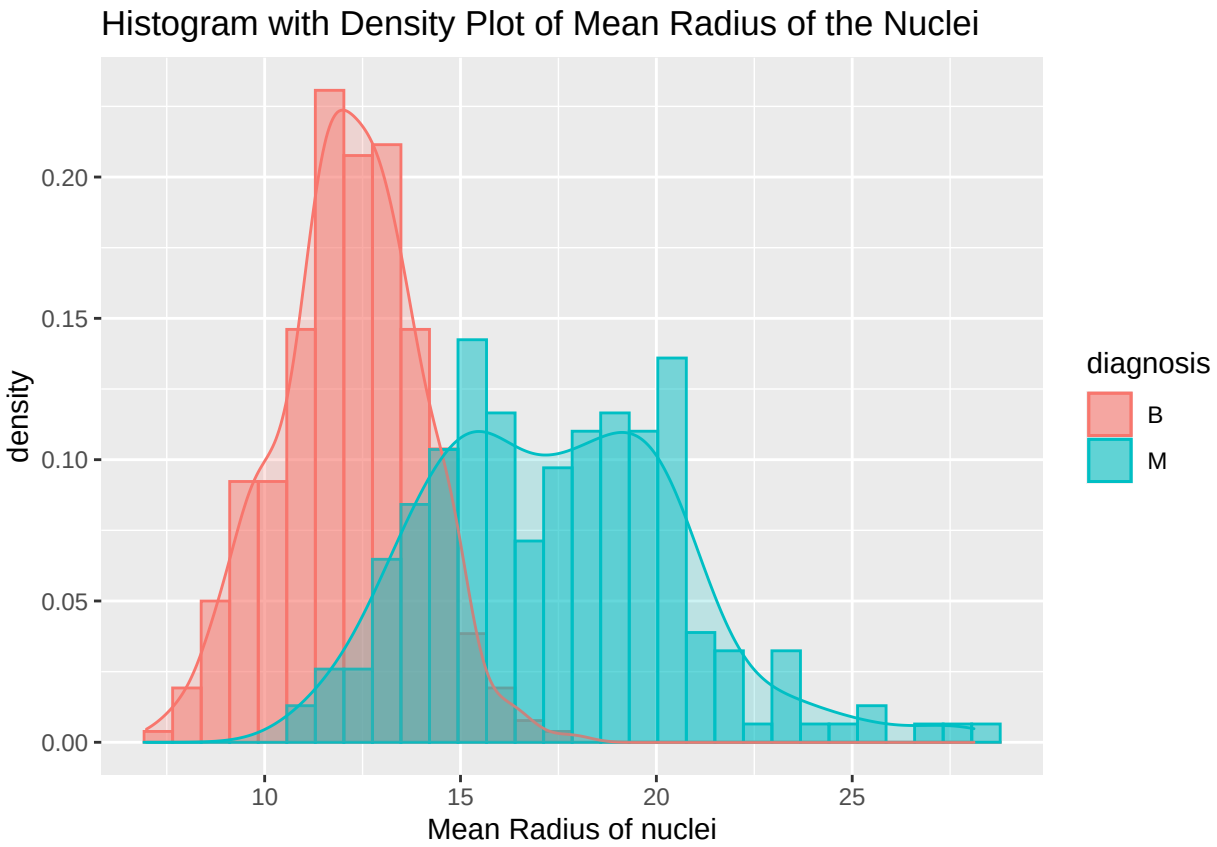
Based on the output above, we chose to focus on the mean measurements. The standard error and maximum/worst measurements do not appear to add much value compared to what is already visible in our mean measurements analysis.

From the preliminary EDA of the mean above, we learned of the vast differences between the histograms for benign and malignant tumors in all of the tumor metadata.

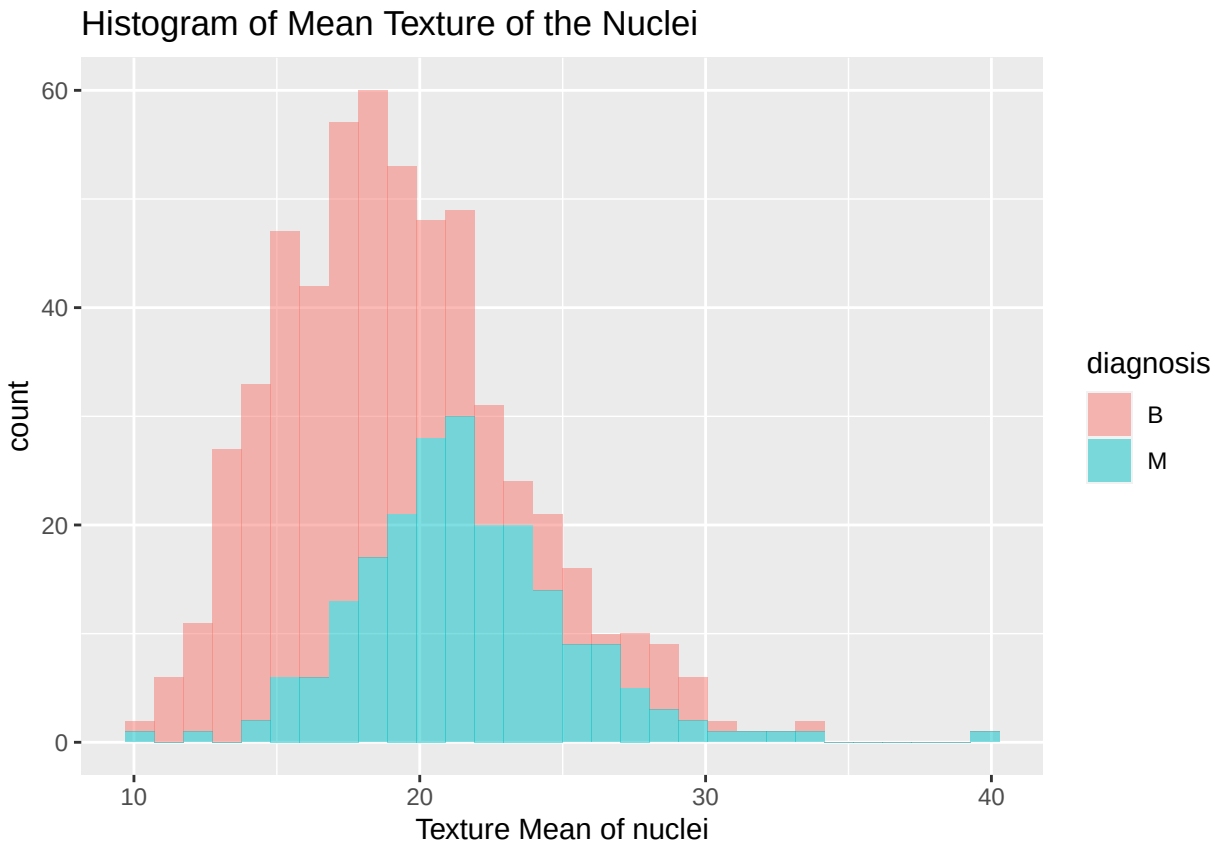
```
# Histogram of the Mean Radius of nuclei colored by diagnosis.
ggplot(data = cancer_df, aes(x = radius_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Mean Radius of nuclei') +
  ggtitle("Histogram of Mean Radius of the Nuclei")
```



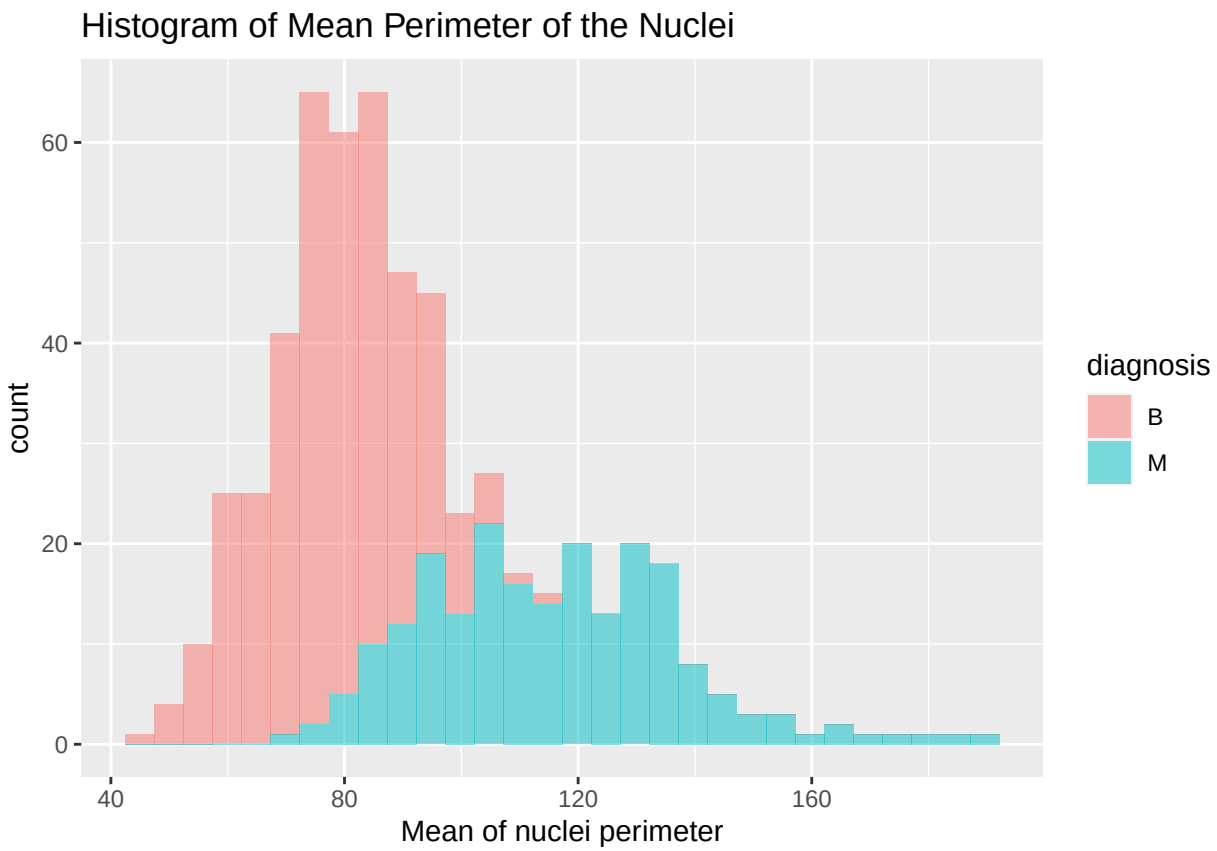

```
# Histogram of the Mean Radius of nuclei colored by diagnosis and distribution outlined
ggplot(cancer_df, aes(x = radius_mean, color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Mean Radius of nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Mean Radius of the Nuclei")
```



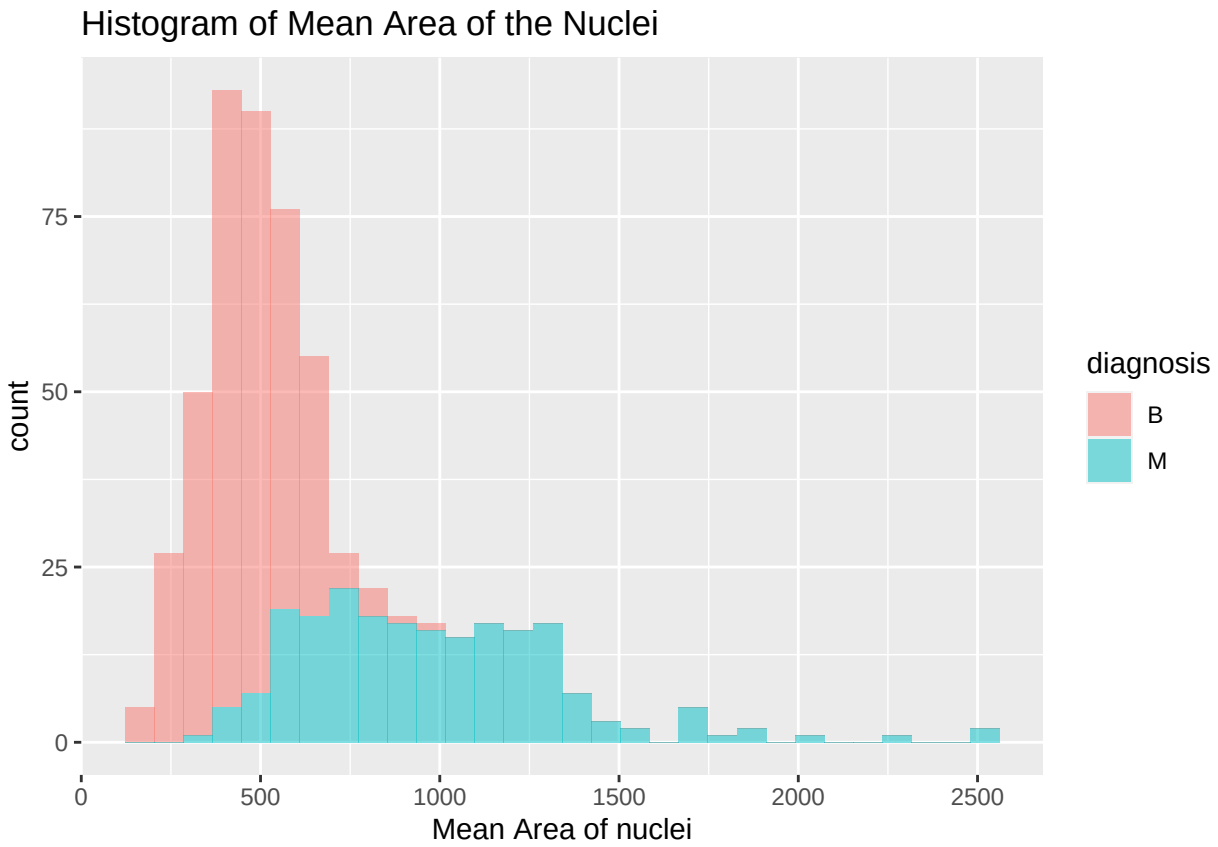

```
# Histogram of the texture_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = texture_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Texture Mean of nuclei') +
  ggtitle("Histogram of Mean Texture of the Nuclei")
```



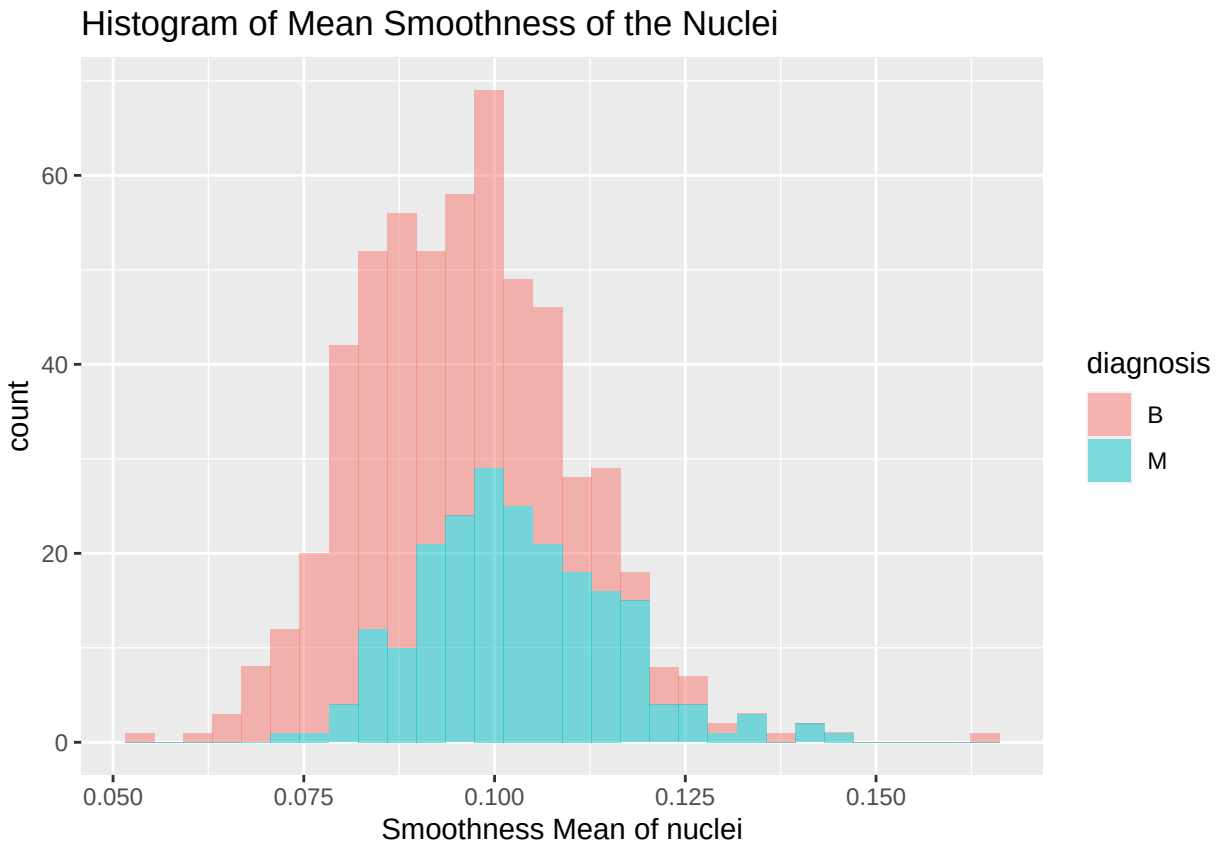

```
# Histogram of the perimeter_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = perimeter_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Mean of nuclei perimeter') +
  ggtitle("Histogram of Mean Perimeter of the Nuclei")
```



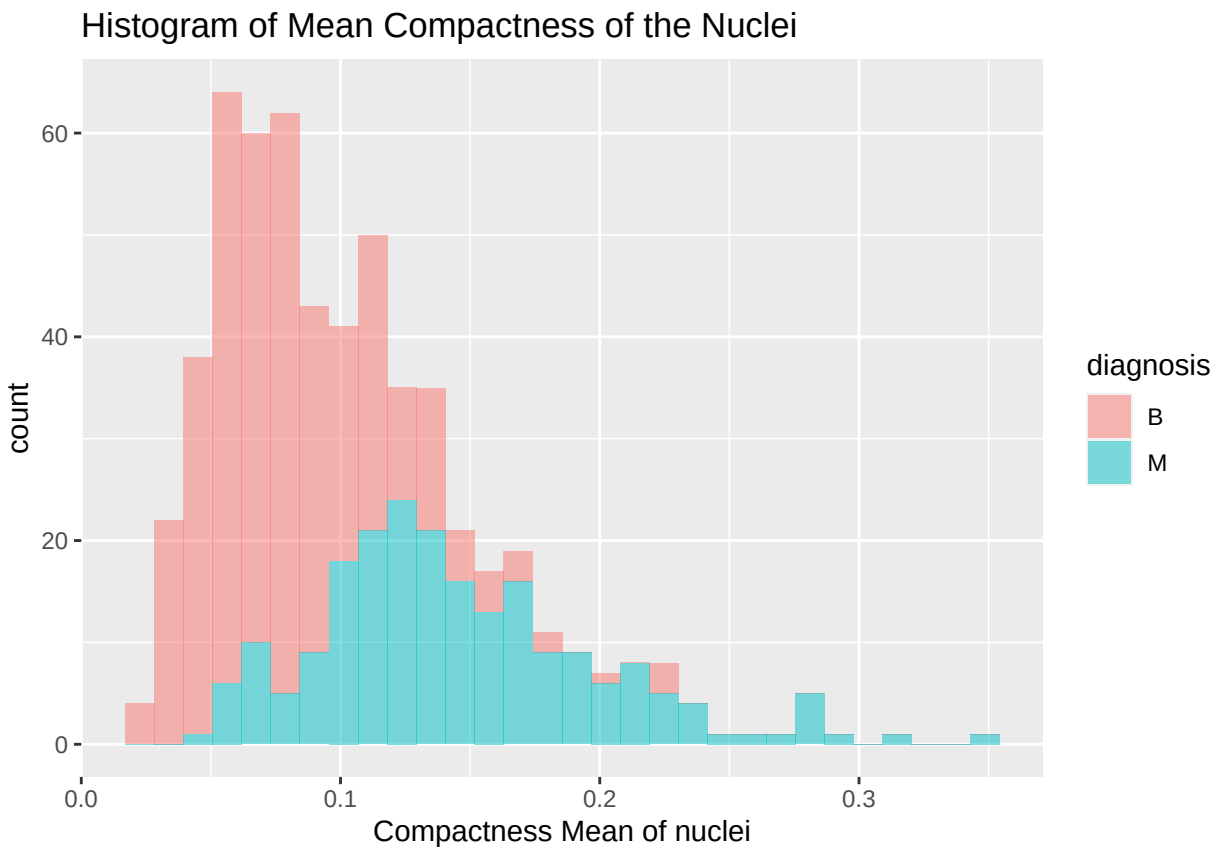

```
# Histogram of the area_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = area_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Mean Area of nuclei') +
  ggtitle("Histogram of Mean Area of the Nuclei")
```



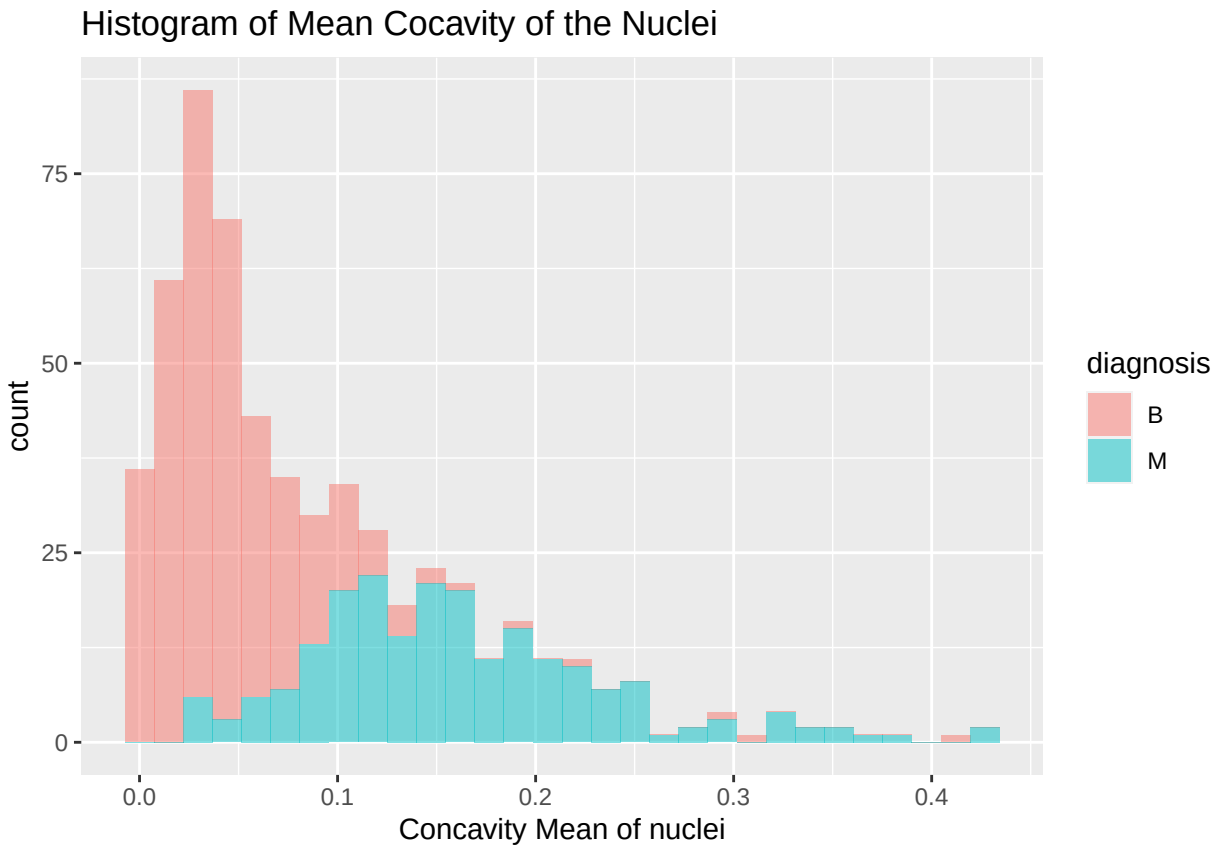

```
# Histogram of the smoothness_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = smoothness_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Smoothness Mean of nuclei') +
  ggtitle("Histogram of Mean Smoothness of the Nuclei")
```



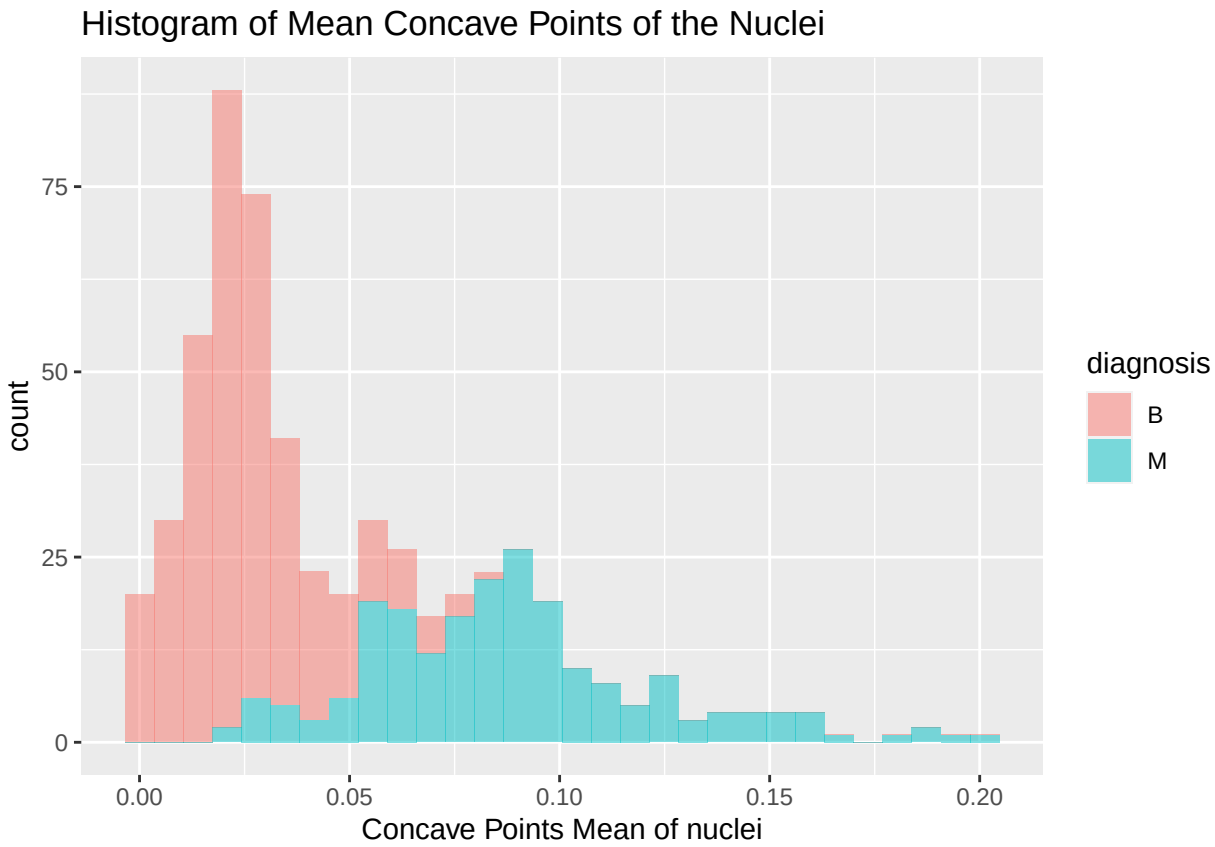

```
# Histogram of the compactness_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = compactness_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Compactness Mean of nuclei') +
  ggtitle("Histogram of Mean Compactness of the Nuclei")
```



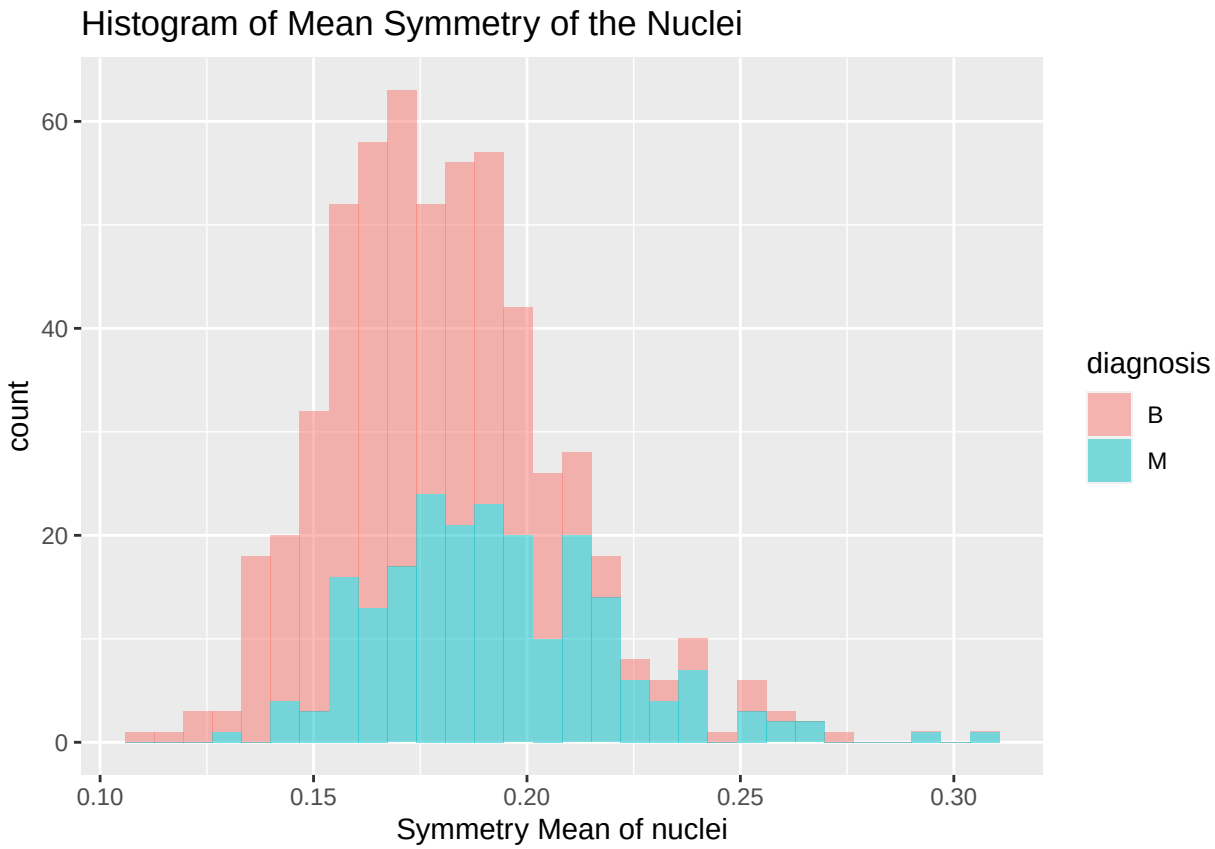

```
# Histogram of the concavity_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = concavity_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Concavity Mean of nuclei') +
  ggtitle("Histogram of Mean Cocavity of the Nuclei")
```



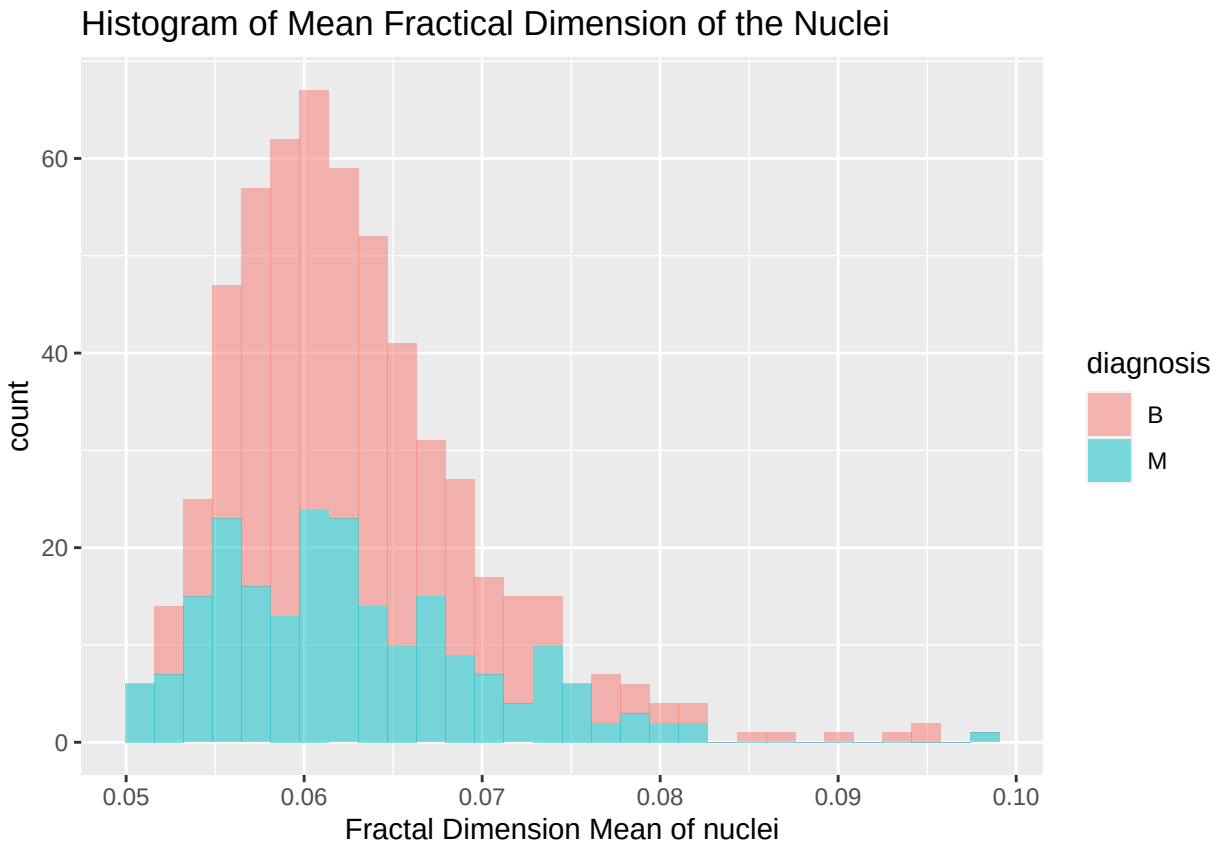

```
# Histogram of the concave points_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = concave_points_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Concave Points Mean of nuclei') +
  ggtitle("Histogram of Mean Concave Points of the Nuclei")
```



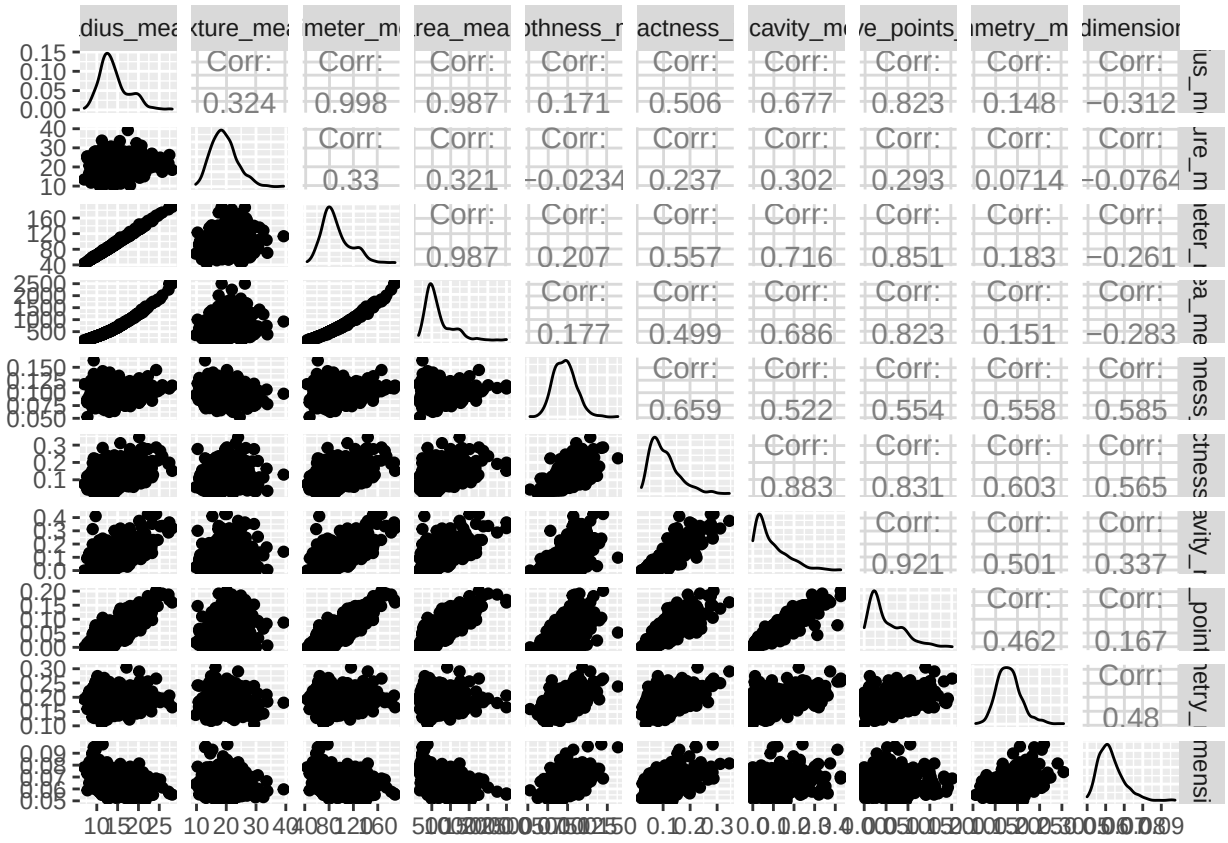

```
# Histogram of the symmetry_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = symmetry_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Symmetry Mean of nuclei') +
  ggtitle("Histogram of Mean Symmetry of the Nuclei")
```




```
# Histogram of the symmetry_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = fractal_dimension_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Fractal Dimension Mean of nuclei') +
  ggtitle("Histogram of Mean Fractical Dimension of the Nuclei")
```




```
# import library to use ggpairs function
library(GGally)

# Use the ggpairs function (from GGally) which which creates
#a matrix of plots within a given dataset
ggpairs(cancer_df[ , c(3:12)])
```



3. Split the data into test and training data.

```
# First, rescale the data
# create the rescaling function we have been using thus far
rescale_x <- function(x){(x-min(x))/(max(x)-min(x))}

# create a copy of the df
rescaled_df <- cancer_df

# retain only the first two columns and all the 'mean' data
rescaled_df <- rescaled_df[1:12]

# apply the rescale function to all columns except id and diagnosis
rescaled_df[3:12] <- sapply(rescaled_df[3:12],rescale_x)

# confirm rescaling worked correctly
# all rescaled vars should be within [0,1]
summary(rescaled_df)
```

```
##        id            diagnosis  radius_mean      texture_mean
##  Min.   :     8670   B:357     Min.   :0.0000   Min.   :0.0000
##  1st Qu.:   869218   M:212     1st Qu.:0.2233   1st Qu.:0.2185
##  Median :   906024             Median :0.3024   Median :0.3088
##  Mean   : 30371831             Mean   :0.3382   Mean   :0.3240
##  3rd Qu.:  8813129             3rd Qu.:0.4164   3rd Qu.:0.4089
```

```
##  Max.   :911320502             Max.   :1.0000   Max.   :1.0000
##  perimeter_mean     area_mean      smoothness_mean  compactness_mean
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2168   1st Qu.:0.1174   1st Qu.:0.3046   1st Qu.:0.1397
##  Median :0.2933   Median :0.1729   Median :0.3904   Median :0.2247
##  Mean   :0.3329   Mean   :0.2169   Mean   :0.3948   Mean   :0.2606
##  3rd Qu.:0.4168   3rd Qu.:0.2711   3rd Qu.:0.4755   3rd Qu.:0.3405
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##  concavity_mean    concave_points_mean symmetry_mean    fractal_dimension_mean
##  Min.   :0.00000   Min.   :0.0000      Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.06926   1st Qu.:0.1009      1st Qu.:0.2823   1st Qu.:0.1630
##  Median :0.14419   Median :0.1665      Median :0.3697   Median :0.2439
##  Mean   :0.20806   Mean   :0.2431      Mean   :0.3796   Mean   :0.2704
##  3rd Qu.:0.30623   3rd Qu.:0.3678      3rd Qu.:0.4530   3rd Qu.:0.3404
##  Max.   :1.00000   Max.   :1.0000      Max.   :1.0000   Max.   :1.0000
```

```
# Now split the data
# set the seed to Notre Dame's founding year
set.seed(1842)

# determine the number of rows in the dataframe
n <- nrow(rescaled_df)

# get a list of 20% of the rows in combined to use as indices
test_idx <- sample.int(n, size = round(0.2 * n))

# set the the training data to be those rows not matching the index list
training <- rescaled_df[-test_idx,]

# show the number of training rows
nrow(training)
```

```
## [1] 455
```

```
# get a glimpse of the data using tibble's, glimpse function
glimpse(training)
```

```
## Observations: 455
## Variables: 12
## $ id                    <dbl> 842302, 842517, 84300903, 84348301, 843786, ...
## $ diagnosis             <fct> M, M, M, M, M, M, M, M, M, M, M, M, M, M,...
## $ radius_mean           <dbl> 0.5210374, 0.6431445, 0.6014956, 0.2100904, ...
## $ texture_mean          <dbl> 0.0226581, 0.2725736, 0.3902604, 0.3608387, ...
## $ perimeter_mean        <dbl> 0.5459885, 0.6157833, 0.5957432, 0.2335015, ...
## $ area_mean             <dbl> 0.36373277, 0.50159067, 0.44941676, 0.102905...
## $ smoothness_mean       <dbl> 0.5937528, 0.2898799, 0.5143089, 0.8113208, ...
## $ compactness_mean      <dbl> 0.7920373, 0.1817680, 0.4310165, 0.8113613, ...
## $ concavity_mean        <dbl> 0.70313964, 0.20360825, 0.46251172, 0.565604...
## $ concave_points_mean   <dbl> 0.7311133, 0.3487575, 0.6356859, 0.5228628, ...
## $ symmetry_mean         <dbl> 0.6863636, 0.3797980, 0.5095960, 0.7762626, ...
## $ fractal_dimension_mean <dbl> 0.60551811, 0.14132266, 0.21124684, 1.000000...
```

```
# set the the test data to be those rows matching the index list
testing <- rescaled_df[test_idx,]

# show the number of test rows
```

```
nrow(testing)
```

```
## [1] 114
```

```
# get a glimpse of the training data
glimpse(training)
```

```
## Observations: 455
## Variables: 12
## $ id                     <dbl> 842302, 842517, 84300903, 84348301, 843786, ...
## $ diagnosis              <fct> M, M, M, M, M, M, M, M, M, M, M, M, M, M, M,...
## $ radius_mean            <dbl> 0.5210374, 0.6431445, 0.6014956, 0.2100904, ...
## $ texture_mean           <dbl> 0.0226581, 0.2725736, 0.3902604, 0.3608387, ...
## $ perimeter_mean         <dbl> 0.5459885, 0.6157833, 0.5957432, 0.2335015, ...
## $ area_mean              <dbl> 0.36373277, 0.50159067, 0.44941676, 0.102905...
## $ smoothness_mean        <dbl> 0.5937528, 0.2898799, 0.5143089, 0.8113208, ...
## $ compactness_mean       <dbl> 0.7920373, 0.1817680, 0.4310165, 0.8113613, ...
## $ concavity_mean         <dbl> 0.70313964, 0.20360825, 0.46251172, 0.565604...
## $ concave_points_mean    <dbl> 0.7311133, 0.3487575, 0.6356859, 0.5228628, ...
## $ symmetry_mean          <dbl> 0.6863636, 0.3797980, 0.5095960, 0.7762626, ...
## $ fractal_dimension_mean <dbl> 0.60551811, 0.14132266, 0.21124684, 1.000000...
```

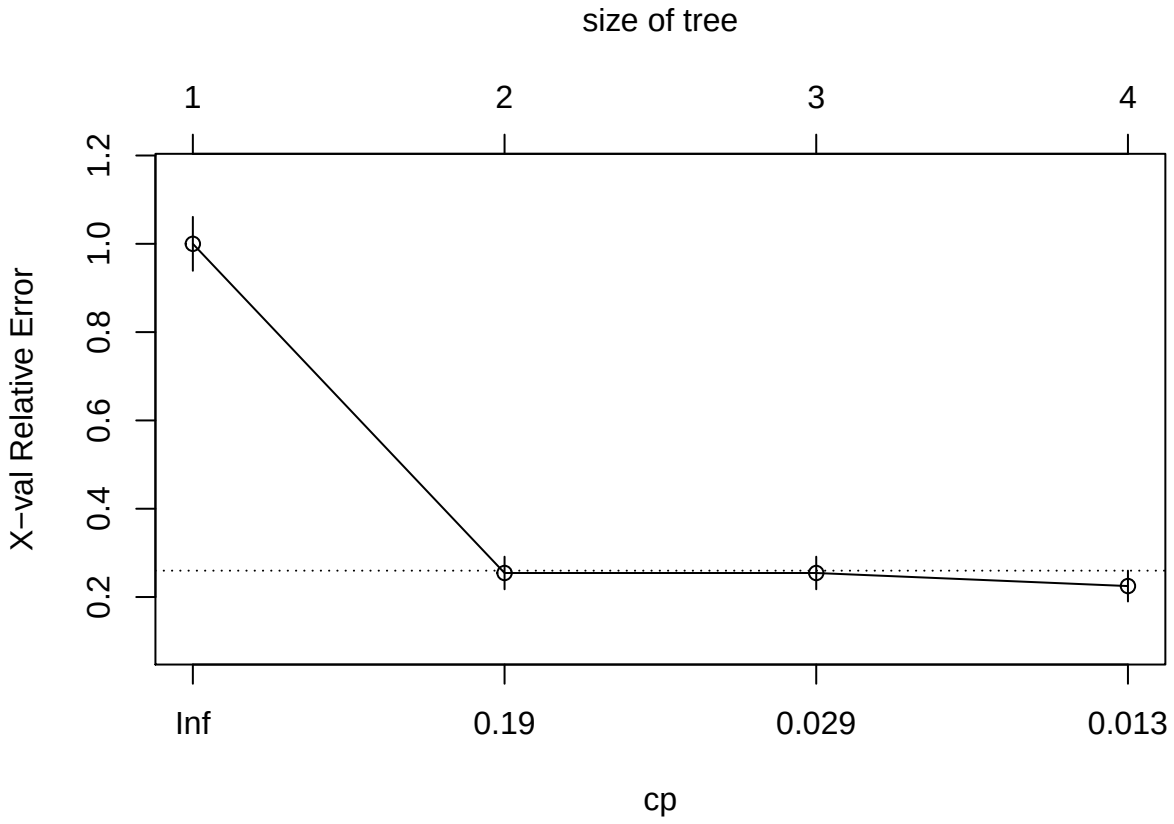
4. Build a classification algorithm using decision trees. Prune your tree appropriately.

```
# set the seed for consistent results
set.seed(1842)

# Define the formula
form <- as.formula(diagnosis ~ radius_mean + texture_mean +
                     perimeter_mean + area_mean + smoothness_mean +
                     compactness_mean + concavity_mean +
                     concave_points_mean + symmetry_mean +
                     fractal_dimension_mean)

# Generate the Decision tree
diag.tree <- rpart(form, data=training)

# Print the Tree CP
printcp(diag.tree)
```

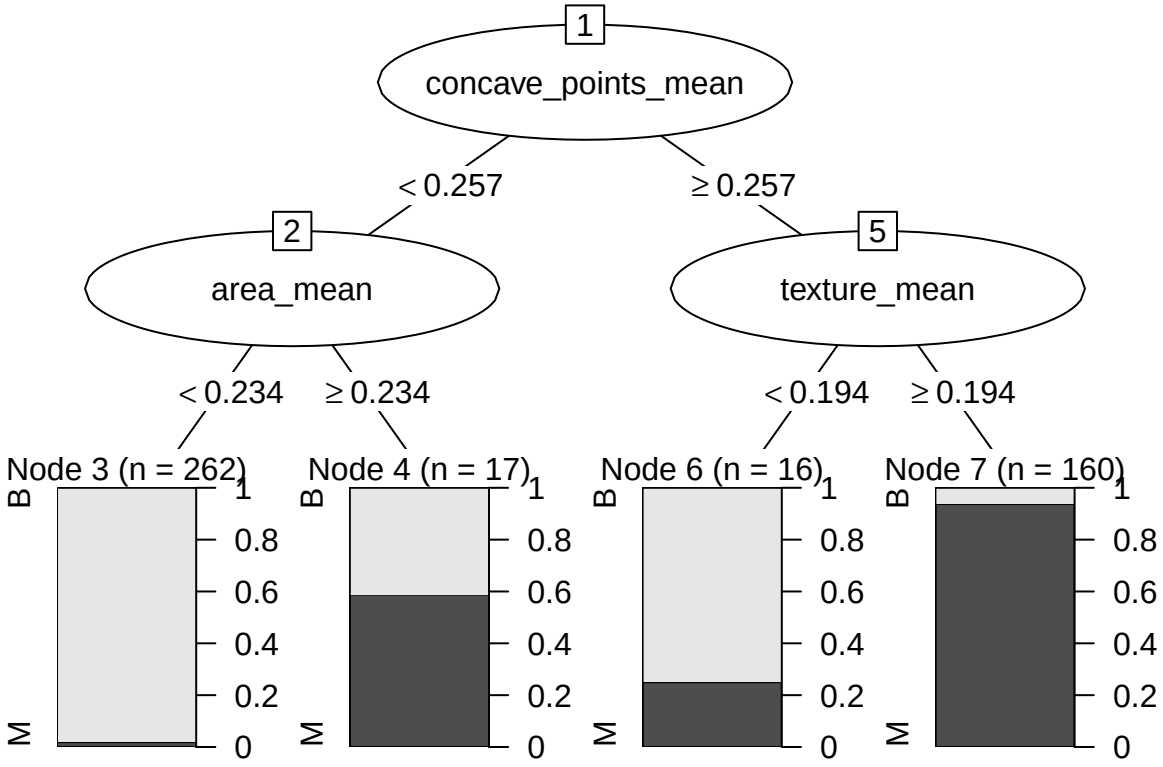
```
##
## Classification tree:
## rpart(formula = form, data = training)
##
## Variables actually used in tree construction:
## [1] area_mean           concave_points_mean texture_mean
##
## Root node error: 169/455 = 0.37143
##
## n= 455
##
##         CP nsplit rel error  xerror     xstd
```

```
## 1 0.781065      0   1.00000 1.00000 0.060987
## 2 0.047337      1   0.21893 0.25444 0.036922
## 3 0.017751      2   0.17160 0.25444 0.036922
## 4 0.010000      3   0.15385 0.22485 0.034919
```

```
# Plotting the Tree CP
plotcp(diag.tree)
```

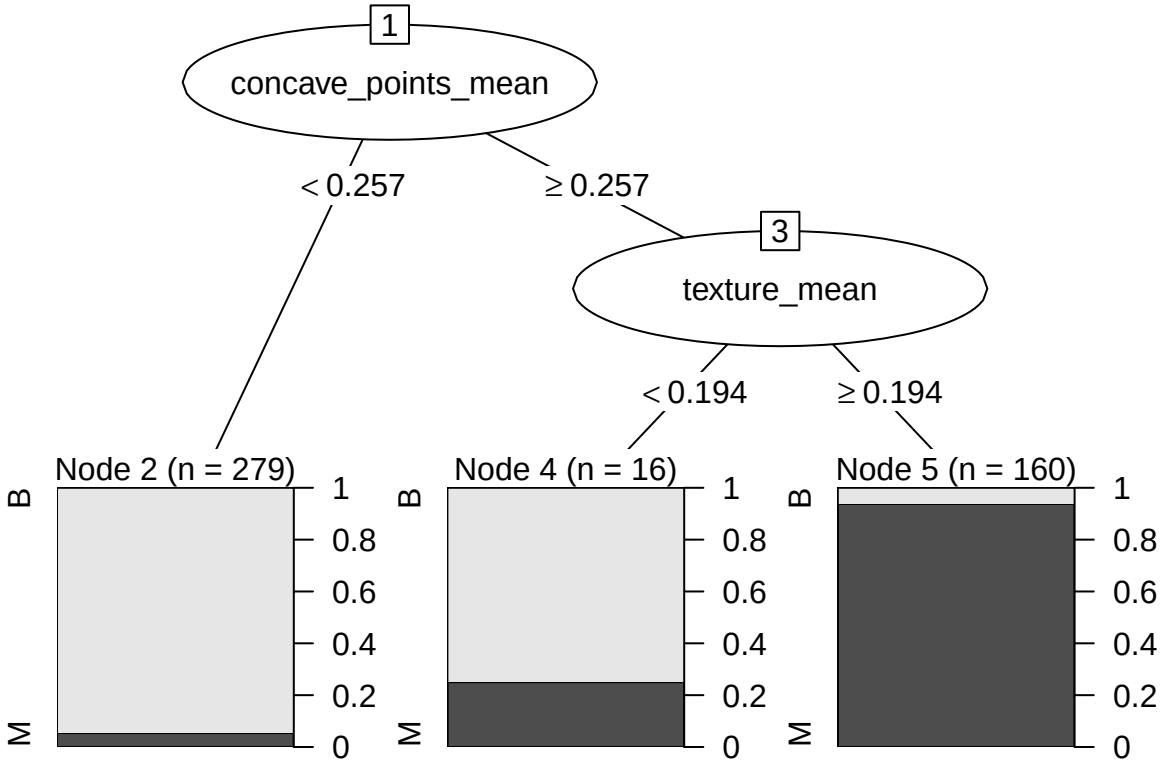



```
# Partykit plot of the Tree
plot(as.party(diag.tree))
```

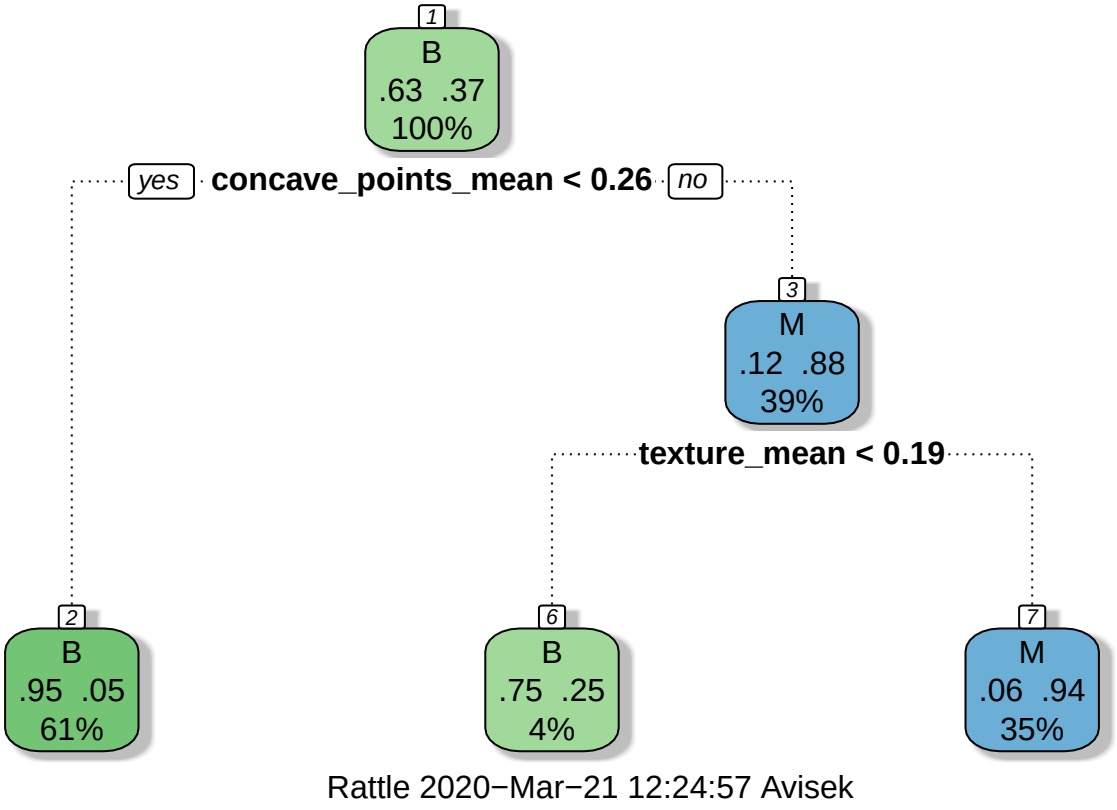


Prune the tree using CP 0.035 using the prune function

```
diag.new.tree <- prune(diag.tree, cp = 0.035)

#Partykit plot of the Pruned Tree
plot(as.party(diag.new.tree))
```

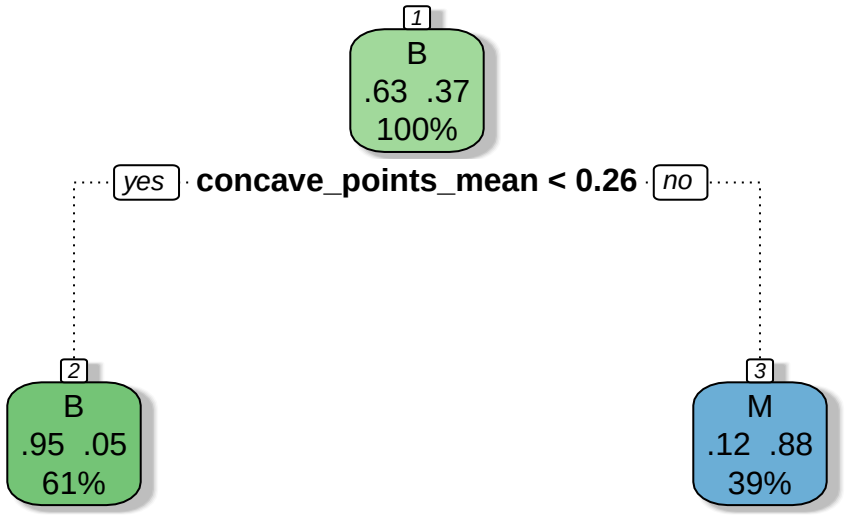



```
library(rattle)
#Generate the fancy plot
fancyRpartPlot(diag.new.tree)
```



Let's use the cross-validation method from `caret` package to compare the models.

```
# fit the model
diag.tree.caret = train(form,
                        data = training,
                        method = 'rpart',
                        trControl = trainControl(method = "cv"))

#Plot the Final Tree
fancyRpartPlot(diag.tree.caret$finalModel)
```



Rattle 2020−Mar−21 12:24:58 Avisek

```
# use the decision tree created above to predict values in the test data
# and then store the results
testing$tree_predict <- predict(diag.new.tree,
                                newdata=testing,
                                type="class")

# create the confusion matrix using the table function
confusion_tree <- table(testing$tree_predict,
                        testing$diagnosis)
#Print the confusion matrix
confusion_tree
```

```
##
##      B  M
##   B 67  7
##   M  4 36
```

```
# show the accuracy of the decision tree
cat("Overall accuracy of prediction:\t",
    sum(diag(confusion_tree)/nrow(testing)) %>%
      round(4),"\n")
```

```
## Overall accuracy of prediction:   0.9035
```

```
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_tree[1,2] /
       (confusion_tree[1,1] + confusion_tree[1,2])) %>%
```

```
      round(4),"\n")
```

```
## Rate of misclassifying M as B:    0.0946
```

```
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_tree[2,1] /
       (confusion_tree[2,1] + confusion_tree[2,2])) %>%
      round(4),"\n")
```

```
## Rate of misclassifying B as M:    0.1
```

5. Build a classification algorithm using random forests/bagging. Adjust the parameters of the forest appropriately.

```
# set the seed for consistent results
set.seed(1842)

#Generate the Random Forest
diag.forest <- randomForest(form, mtry = 3,
                            ntree = 500,
                            data=training,
                            na.action = na.roughfix)

#Print the Random Forest
diag.forest
```

```
##
## Call:
##  randomForest(formula = form, data = training, mtry = 3, ntree = 500,      na.action = na.roughfix)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 4.84%
## Confusion matrix:
##     B   M class.error
## B 277   9  0.03146853
## M  13 156  0.07692308
```

```
# use the Random Forest created above to predict values in the test data
# and then store the results
testing$rf_pred <- predict(diag.forest,
                           newdata=testing,
                           type="class")

# create the confusion matrix using the table function
confusion_rf <- table(testing$rf_pred,
                      testing$diagnosis)
# Print the confusion matrix
confusion_rf
```

```
##
##      B  M
```

```
##   B 68  6
##   M  3 37
```

```
# show the accuracy of the Random Forest
cat("Overall accuracy of prediction:\t",
     sum(diag(confusion_rf)/nrow(testing)) %>%
      round(4),"\n")
```

```
## Overall accuracy of prediction:   0.9211
```

```
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_rf[1,2] / (confusion_rf[1,1]+confusion_rf[1,2])) %>%
      round(4),"\n")
```

```
## Rate of misclassifying M as B:    0.0811
```

```
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_rf[2,1] / (confusion_rf[2,1] + confusion_rf[2,2])) %>%
      round(4),"\n")
```

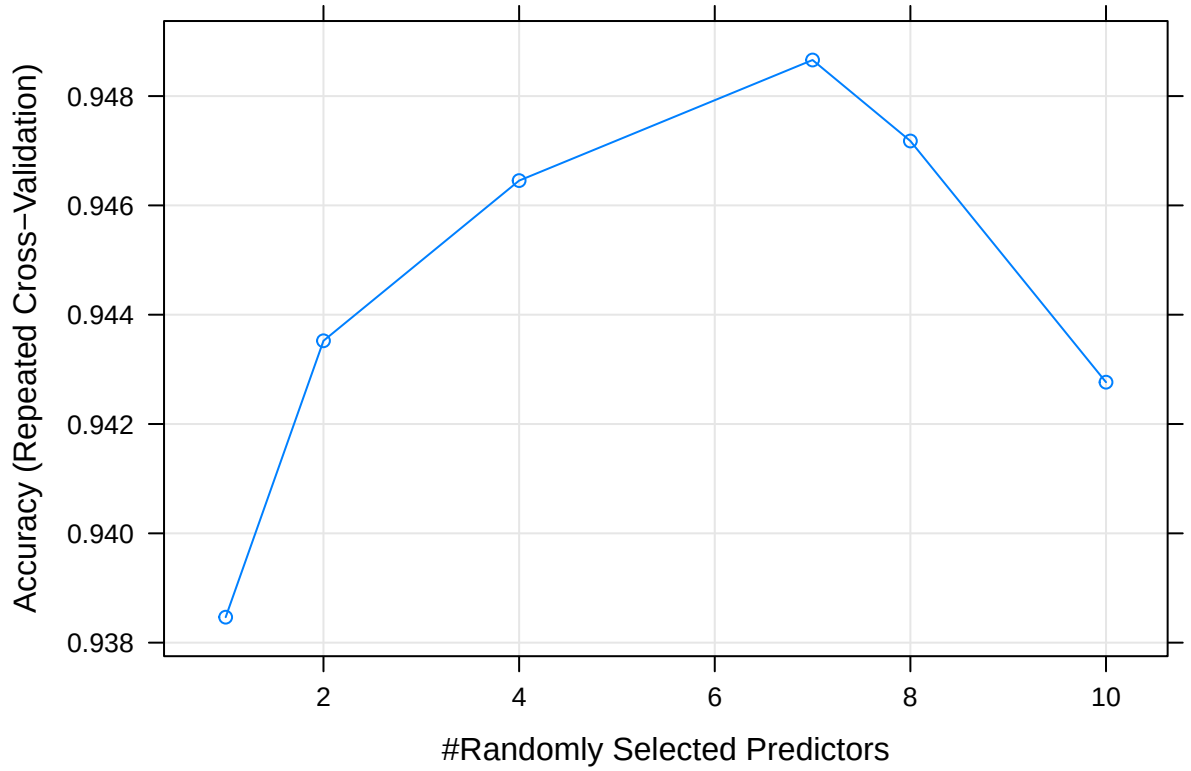
```
## Rate of misclassifying B as M:    0.075
```

```
#10 folds repeat 3 times
control <- trainControl(method='repeatedcv',
                        number=10,
                        repeats=3,
                        search = 'random')
#Metric compare model is Accuracy
metric <- "Accuracy"

# set the seed for consistent results
set.seed(1842)

#Number randomely variable selected is mtry
diag.rf.caret <- train(form,
                       data=training,
                       method='rf',
                       metric='Accuracy',
                       tuneLength = 10,
                       trControl=control)

#Print the random forest cross validation results
print(diag.rf.caret)
```

```
## Random Forest
##
## 455 samples
##  10 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 409, 409, 409, 409, 410, 410, ...
## Resampling results across tuning parameters:
##
```

```
##   mtry  Accuracy   Kappa
##    1    0.9384658  0.8674654
##    2    0.9435229  0.8791329
##    4    0.9464537  0.8857706
##    7    0.9486598  0.8898941
##    8    0.9471783  0.8869705
##   10    0.9427653  0.8774122
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 7.
```

```r
#Plot the mtry vs Accuracy chart
plot(diag.rf.caret)
```



So we can see the highest accuracy can be achived by using mtry = 7 i.e. considering 7 predictors at a time for spliting to generate the trees in the random forest.

```r
# use the optimized Random Forest created above to predict values in the test data
# and then store the results
testing$rf_opt_pred <- predict(diag.rf.caret$finalModel,
                               newdata=testing,
                               type="class")

# create the confusion matrix using the table function
confusion_rf_opt <- table(testing$rf_opt_pred,
                          testing$diagnosis)
# Print the confusion matrix
confusion_rf_opt
```

```
##
##      B  M
##   B 68  6
##   M  3 37
```

```
# show the accuracy of the Random Forest
cat("Overall accuracy of prediction:\t",
    sum(diag(confusion_rf_opt)/nrow(testing)) %>%
      round(4),"\n")
```

```
## Overall accuracy of prediction:   0.9211
```

```
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_rf_opt[1,2] /
       (confusion_rf_opt[1,1] + confusion_rf_opt[1,2])) %>%
      round(4),"\n")
```

```
## Rate of misclassifying M as B:    0.0811
```

```
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_rf_opt[2,1] /
       (confusion_rf_opt[2,1] + confusion_rf_opt[2,2])) %>%
      round(4),"\n")
```

```
## Rate of misclassifying B as M:    0.075
```

There appears to be no difference between the initial random forest and the adjusted/optimized one.

6. Build a classification algorithm using Kth Nearest Neighbors. Tune the value of K appropriately.

```
# Choose a value for K that is equal to the square root of n,
# the nummber of onservations in the training set
k_try = sqrt(nrow(training[3:12]))
k_try
```

```
## [1] 21.33073
```

```
# We'll use 21 as our value of K
diag_knn_21 <- knn(training[3:12],
                   testing[3:12],
                   cl = training$diagnosis,
                   k=21)

# create and display the confusion matrix
confusion_knn21 <- table(predicted = diag_knn_21,
                         actual = testing$diagnosis)
# Print the confusion matrix
confusion_knn21
```

```
##          actual
## predicted  B  M
##         B 69  7
```

```
##         M  2 36
```

```
# show the accuracy of the KNN classification
cat("Overall accuracy of prediction:\t",
    sum(diag(confusion_knn21) / nrow(testing)) %>%
      round(4),"\n")
```

```
## Overall accuracy of prediction:   0.9211
```

```
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_knn21[1,2] /
       (confusion_knn21[1,1] + confusion_knn21[1,2])) %>%
      round(4),"\n")
```

```
## Rate of misclassifying M as B:    0.0921
```

```
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_knn21[2,1] /
       (confusion_knn21[2,1] + confusion_knn21[2,2])) %>%
      round(4),"\n")
```

```
## Rate of misclassifying B as M:    0.0526
```

Let's tune K to see if we can get better accuracy

```
# set the seed for consistent results
set.seed(1842)

# set the train control to use 5-fold cross validation
# choosing 5-fold as a good middle ground
trControl <- trainControl(method  = "cv",
                          number  = 5)

#find the best knn fit using values of K of all odd numbers from 1 to 99
knn_fit <- train(diagnosis ~ .,
             method     = "knn",
             tuneGrid   = expand.grid(k = c((1:50)*2 - 1)),
             trControl  = trControl,
             metric     = "Accuracy",
             data       = training[-1])
#Print the Model
knn_fit
```

```
## k-Nearest Neighbors
##
## 455 samples
##  10 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 364, 363, 364, 365, 364
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9427579  0.8771948
```

```
##    3  0.9427584  0.8767277
##    5  0.9426851  0.8775809
##    7  0.9405118  0.8725297
##    9  0.9471779  0.8859015
##   11  0.9427579  0.8763228
##   13  0.9449318  0.8809589
##   15  0.9427340  0.8761268
##   17  0.9427340  0.8755511
##   19  0.9427340  0.8755511
##   21  0.9471296  0.8848759
##   23  0.9427335  0.8754956
##   25  0.9427335  0.8754956
##   27  0.9427335  0.8754956
##   29  0.9450040  0.8798233
##   31  0.9428062  0.8746801
##   33  0.9427579  0.8744835
##   35  0.9361639  0.8603202
##   37  0.9383862  0.8649461
##   39  0.9361884  0.8597384
##   41  0.9339906  0.8551269
##   43  0.9383862  0.8644052
##   45  0.9361884  0.8595135
##   47  0.9383862  0.8647212
##   49  0.9361884  0.8595135
##   51  0.9361884  0.8595135
##   53  0.9339906  0.8546754
##   55  0.9339906  0.8546754
##   57  0.9339906  0.8546754
##   59  0.9361884  0.8595135
##   61  0.9340144  0.8548800
##   63  0.9339906  0.8546754
##   65  0.9339906  0.8546754
##   67  0.9339906  0.8546754
##   69  0.9339906  0.8546754
##   71  0.9362128  0.8593618
##   73  0.9340150  0.8544113
##   75  0.9340150  0.8544113
##   77  0.9340150  0.8544113
##   79  0.9318172  0.8495144
##   81  0.9296194  0.8445041
##   83  0.9273971  0.8398176
##   85  0.9296194  0.8445041
##   87  0.9296433  0.8444692
##   89  0.9252715  0.8345782
##   91  0.9296433  0.8441205
##   93  0.9252715  0.8345782
##   95  0.9274693  0.8391264
##   97  0.9252715  0.8341697
##   99  0.9230737  0.8296215
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

Revising our solution to use K = 9

```r
# We'll use 9 as our value of K
diag_knn_9 <- knn(training[3:12],testing[3:12],
                  cl=training$diagnosis,k=9)

# create and display the confusion matrix
confusion_knn9 <- table(predicted = diag_knn_9,
                        actual = testing$diagnosis)
# Print the confusion matrix
confusion_knn9
```

```
##          actual
## predicted  B  M
##         B 70  7
##         M  1 36
```

```r
# show the accuracy of the KNN classification
cat("Overall accuracy of model:\t",
    sum(diag(confusion_knn9)/nrow(testing)) %>%
      round(4),"\n")
```

```
## Overall accuracy of model:    0.9298
```

```r
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_knn9[1,2] /
       (confusion_knn9[1,1] + confusion_knn9[1,2])) %>%
      round(4),"\n")
```

```
## Rate of misclassifying M as B:    0.0909
```

```r
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_knn9[2,1] /
       (confusion_knn9[2,1] + confusion_knn9[2,2])) %>%
      round(4),"\n")
```

```
## Rate of misclassifying B as M:    0.027
```