Final Project Analysis

* Brandon Good * Duncan McKinnon * Kirk Reese *

Loading Data

```r
# load packages
suppressPackageStartupMessages({
  library(tidyverse)
  library(randomForest)
  library(rpart)
  library(partykit)
  library(class)
})
```

```r
# load data set
cancer_data <- read_csv("C:/Users/dunca/OneDrive/Desktop/Course Work/Intro to Data Science/Data/FNA_canc
glimpse(cancer_data)
```

```
## Observations: 569
## Variables: 33
## $ id                      <dbl> 842302, 842517, 84300903, 84348301, 84...
## $ diagnosis               <chr> "M", "M", "M", "M", "M", "M", "M", "M"...
## $ radius_mean             <dbl> 17.990, 20.570, 19.690, 11.420, 20.290...
## $ texture_mean            <dbl> 10.38, 17.77, 21.25, 20.38, 14.34, 15....
## $ perimeter_mean          <dbl> 122.80, 132.90, 130.00, 77.58, 135.10,...
## $ area_mean               <dbl> 1001.0, 1326.0, 1203.0, 386.1, 1297.0,...
## $ smoothness_mean         <dbl> 0.11840, 0.08474, 0.10960, 0.14250, 0....
## $ compactness_mean        <dbl> 0.27760, 0.07864, 0.15990, 0.28390, 0....
## $ concavity_mean          <dbl> 0.30010, 0.08690, 0.19740, 0.24140, 0....
## $ `concave points_mean`   <dbl> 0.14710, 0.07017, 0.12790, 0.10520, 0....
## $ symmetry_mean           <dbl> 0.2419, 0.1812, 0.2069, 0.2597, 0.1809...
## $ fractal_dimension_mean  <dbl> 0.07871, 0.05667, 0.05999, 0.09744, 0....
## $ radius_se               <dbl> 1.0950, 0.5435, 0.7456, 0.4956, 0.7572...
## $ texture_se              <dbl> 0.9053, 0.7339, 0.7869, 1.1560, 0.7813...
## $ perimeter_se            <dbl> 8.589, 3.398, 4.585, 3.445, 5.438, 2.2...
## $ area_se                 <dbl> 153.40, 74.08, 94.03, 27.23, 94.44, 27...
## $ smoothness_se           <dbl> 0.006399, 0.005225, 0.006150, 0.009110...
## $ compactness_se          <dbl> 0.049040, 0.013080, 0.040060, 0.074580...
## $ concavity_se            <dbl> 0.05373, 0.01860, 0.03832, 0.05661, 0....
## $ `concave points_se`     <dbl> 0.015870, 0.013400, 0.020580, 0.018670...
## $ symmetry_se             <dbl> 0.03003, 0.01389, 0.02250, 0.05963, 0....
## $ fractal_dimension_se    <dbl> 0.006193, 0.003532, 0.004571, 0.009208...
## $ radius_worst            <dbl> 25.38, 24.99, 23.57, 14.91, 22.54, 15....
## $ texture_worst           <dbl> 17.33, 23.41, 25.53, 26.50, 16.67, 23....
## $ perimeter_worst         <dbl> 184.60, 158.80, 152.50, 98.87, 152.20,...
## $ area_worst              <dbl> 2019.0, 1956.0, 1709.0, 567.7, 1575.0,...
## $ smoothness_worst        <dbl> 0.1622, 0.1238, 0.1444, 0.2098, 0.1374...
## $ compactness_worst       <dbl> 0.6656, 0.1866, 0.4245, 0.8663, 0.2050...
## $ concavity_worst         <dbl> 0.71190, 0.24160, 0.45040, 0.68690, 0....
## $ `concave points_worst`  <dbl> 0.26540, 0.18600, 0.24300, 0.25750, 0....
## $ symmetry_worst          <dbl> 0.4601, 0.2750, 0.3613, 0.6638, 0.2364...
## $ fractal_dimension_worst <dbl> 0.11890, 0.08902, 0.08758, 0.17300, 0....
```

```
## $ X33                     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
```

EDA

Data Cleaning

```
# change concave points field names to fit pattern
cancer_data <- cancer_data %>%
  mutate(concave_points_mean = `concave points_mean`,
         concave_points_se = `concave points_se`,
         concave_points_worst = `concave points_worst`) %>%
  # remove old field names and unused field X33
  select(-`concave points_mean`, -`concave points_se`, -`concave points_worst`, -X33)

# check if there are any NA values left in the data
cancer_data %>% lapply( function(x){ return( c('NA Count' = sum( is.na( x ) ) ) ) } )
```

```
## $id
## NA Count
##        0
##
## $diagnosis
## NA Count
##        0
##
## $radius_mean
## NA Count
##        0
##
## $texture_mean
## NA Count
##        0
##
## $perimeter_mean
## NA Count
##        0
##
## $area_mean
## NA Count
##        0
##
## $smoothness_mean
## NA Count
##        0
##
## $compactness_mean
## NA Count
##        0
##
## $concavity_mean
## NA Count
##        0
##
```

```
## $symmetry_mean
## NA Count
##        0
##
## $fractal_dimension_mean
## NA Count
##        0
##
## $radius_se
## NA Count
##        0
##
## $texture_se
## NA Count
##        0
##
## $perimeter_se
## NA Count
##        0
##
## $area_se
## NA Count
##        0
##
## $smoothness_se
## NA Count
##        0
##
## $compactness_se
## NA Count
##        0
##
## $concavity_se
## NA Count
##        0
##
## $symmetry_se
## NA Count
##        0
##
## $fractal_dimension_se
## NA Count
##        0
##
## $radius_worst
## NA Count
##        0
##
## $texture_worst
## NA Count
##        0
##
## $perimeter_worst
## NA Count
```

```
##        0
##
## $area_worst
## NA Count
##        0
##
## $smoothness_worst
## NA Count
##        0
##
## $compactness_worst
## NA Count
##        0
##
## $concavity_worst
## NA Count
##        0
##
## $symmetry_worst
## NA Count
##        0
##
## $fractal_dimension_worst
## NA Count
##        0
##
## $concave_points_mean
## NA Count
##        0
##
## $concave_points_se
## NA Count
##        0
##
## $concave_points_worst
## NA Count
##        0
```

```r
# select attribute name
fields <- c('radius',
            'texture',
            'perimeter',
            'area',
            'smoothness',
            'compactness',
            'concavity',
            'concave_points',
            'symmetry',
            'fractal_dimension')

# create list for stats of each field attribute
data_by_field <- list()
for(i in fields){
  d <- cancer_data %>% select(starts_with(i))
```

```r
  data_by_field[[i]] <- d
}

names(data_by_field)
```

```
##  [1] "radius"            "texture"           "perimeter"        
##  [4] "area"              "smoothness"        "compactness"      
##  [7] "concavity"         "concave_points"    "symmetry"         
## [10] "fractal_dimension"
```

Summarize fields

```r
## summarize data fields by grouping
lapply(data_by_field, summary)
```

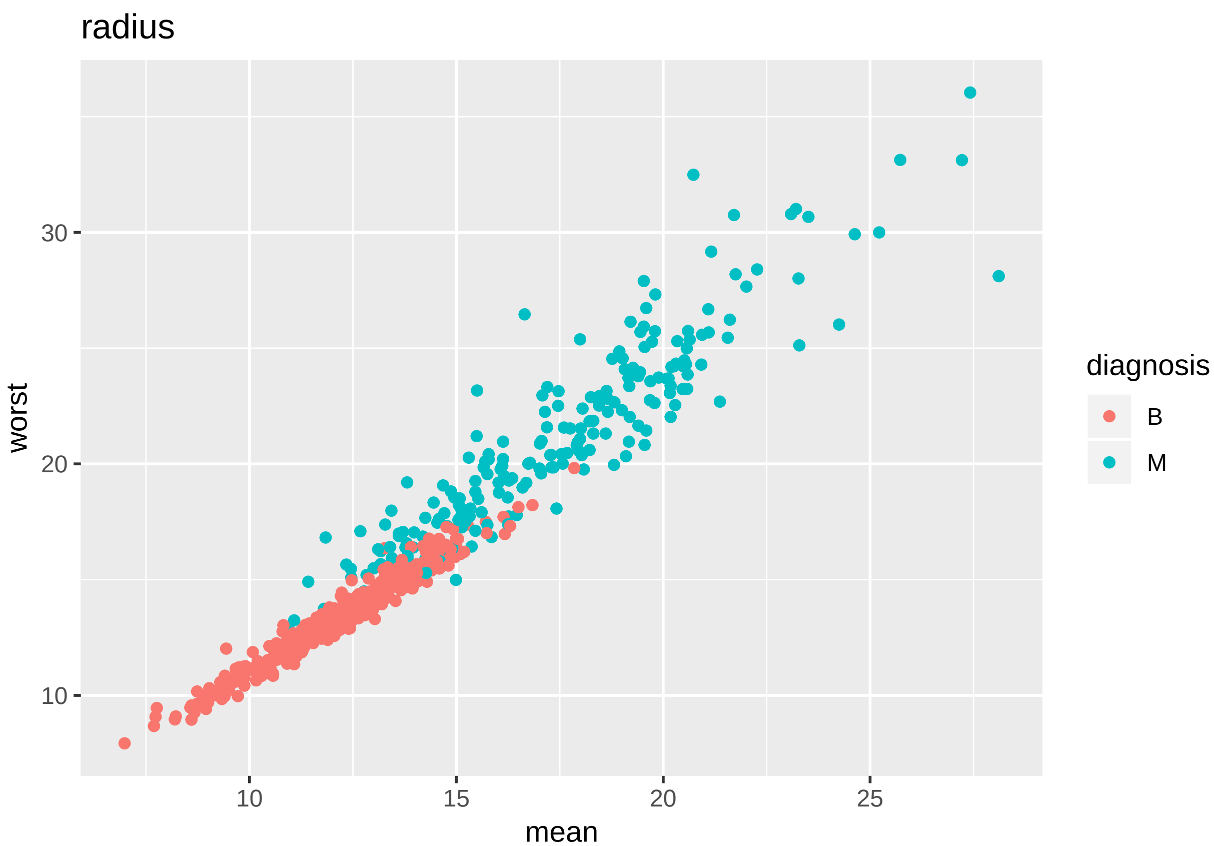
```
## $radius
##   radius_mean       radius_se       radius_worst  
##  Min.   : 6.981   Min.   :0.1115   Min.   : 7.93  
##  1st Qu.:11.700   1st Qu.:0.2324   1st Qu.:13.01  
##  Median :13.370   Median :0.3242   Median :14.97  
##  Mean   :14.127   Mean   :0.4052   Mean   :16.27  
##  3rd Qu.:15.780   3rd Qu.:0.4789   3rd Qu.:18.79  
##  Max.   :28.110   Max.   :2.8730   Max.   :36.04  
## 
## $texture
##   texture_mean     texture_se     texture_worst  
##  Min.   : 9.71   Min.   :0.3602   Min.   :12.02  
##  1st Qu.:16.17   1st Qu.:0.8339   1st Qu.:21.08  
##  Median :18.84   Median :1.1080   Median :25.41  
##  Mean   :19.29   Mean   :1.2169   Mean   :25.68  
##  3rd Qu.:21.80   3rd Qu.:1.4740   3rd Qu.:29.72  
##  Max.   :39.28   Max.   :4.8850   Max.   :49.54  
## 
## $perimeter
##  perimeter_mean    perimeter_se    perimeter_worst 
##  Min.   : 43.79   Min.   : 0.757   Min.   : 50.41  
##  1st Qu.: 75.17   1st Qu.: 1.606   1st Qu.: 84.11  
##  Median : 86.24   Median : 2.287   Median : 97.66  
##  Mean   : 91.97   Mean   : 2.866   Mean   :107.26  
##  3rd Qu.:104.10   3rd Qu.: 3.357   3rd Qu.:125.40  
##  Max.   :188.50   Max.   :21.980   Max.   :251.20  
## 
## $area
##    area_mean         area_se          area_worst    
##  Min.   : 143.5   Min.   :  6.802   Min.   : 185.2  
##  1st Qu.: 420.3   1st Qu.: 17.850   1st Qu.: 515.3  
##  Median : 551.1   Median : 24.530   Median : 686.5  
##  Mean   : 654.9   Mean   : 40.337   Mean   : 880.6  
##  3rd Qu.: 782.7   3rd Qu.: 45.190   3rd Qu.:1084.0  
##  Max.   :2501.0   Max.   :542.200   Max.   :4254.0  
## 
## $smoothness
##  smoothness_mean   smoothness_se      smoothness_worst 
```

```
##  Min.   :0.05263   Min.   :0.001713   Min.   :0.07117  
##  1st Qu.:0.08637   1st Qu.:0.005169   1st Qu.:0.11660  
##  Median :0.09587   Median :0.006380   Median :0.13130  
##  Mean   :0.09636   Mean   :0.007041   Mean   :0.13237  
##  3rd Qu.:0.10530   3rd Qu.:0.008146   3rd Qu.:0.14600  
##  Max.   :0.16340   Max.   :0.031130   Max.   :0.22260  
## 
## $compactness
##  compactness_mean  compactness_se     compactness_worst
##  Min.   :0.01938   Min.   :0.002252   Min.   :0.02729  
##  1st Qu.:0.06492   1st Qu.:0.013080   1st Qu.:0.14720  
##  Median :0.09263   Median :0.020450   Median :0.21190  
##  Mean   :0.10434   Mean   :0.025478   Mean   :0.25427  
##  3rd Qu.:0.13040   3rd Qu.:0.032450   3rd Qu.:0.33910  
##  Max.   :0.34540   Max.   :0.135400   Max.   :1.05800  
## 
## $concavity
##  concavity_mean     concavity_se     concavity_worst 
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.02956   1st Qu.:0.01509   1st Qu.:0.1145  
##  Median :0.06154   Median :0.02589   Median :0.2267  
##  Mean   :0.08880   Mean   :0.03189   Mean   :0.2722  
##  3rd Qu.:0.13070   3rd Qu.:0.04205   3rd Qu.:0.3829  
##  Max.   :0.42680   Max.   :0.39600   Max.   :1.2520  
## 
## $concave_points
##  concave_points_mean concave_points_se  concave_points_worst
##  Min.   :0.00000     Min.   :0.000000   Min.   :0.00000     
##  1st Qu.:0.02031     1st Qu.:0.007638   1st Qu.:0.06493     
##  Median :0.03350     Median :0.010930   Median :0.09993     
##  Mean   :0.04892     Mean   :0.011796   Mean   :0.11461     
##  3rd Qu.:0.07400     3rd Qu.:0.014710   3rd Qu.:0.16140     
##  Max.   :0.20120     Max.   :0.052790   Max.   :0.29100     
## 
## $symmetry
##  symmetry_mean     symmetry_se        symmetry_worst  
##  Min.   :0.1060   Min.   :0.007882   Min.   :0.1565  
##  1st Qu.:0.1619   1st Qu.:0.015160   1st Qu.:0.2504  
##  Median :0.1792   Median :0.018730   Median :0.2822  
##  Mean   :0.1812   Mean   :0.020542   Mean   :0.2901  
##  3rd Qu.:0.1957   3rd Qu.:0.023480   3rd Qu.:0.3179  
##  Max.   :0.3040   Max.   :0.078950   Max.   :0.6638  
## 
## $fractal_dimension
##  fractal_dimension_mean fractal_dimension_se fractal_dimension_worst
##  Min.   :0.04996        Min.   :0.0008948    Min.   :0.05504        
##  1st Qu.:0.05770        1st Qu.:0.0022480    1st Qu.:0.07146        
##  Median :0.06154        Median :0.0031870    Median :0.08004        
##  Mean   :0.06280        Mean   :0.0037949    Mean   :0.08395        
##  3rd Qu.:0.06612        3rd Qu.:0.0045580    3rd Qu.:0.09208        
##  Max.   :0.09744        Max.   :0.0298400    Max.   :0.20750        
```

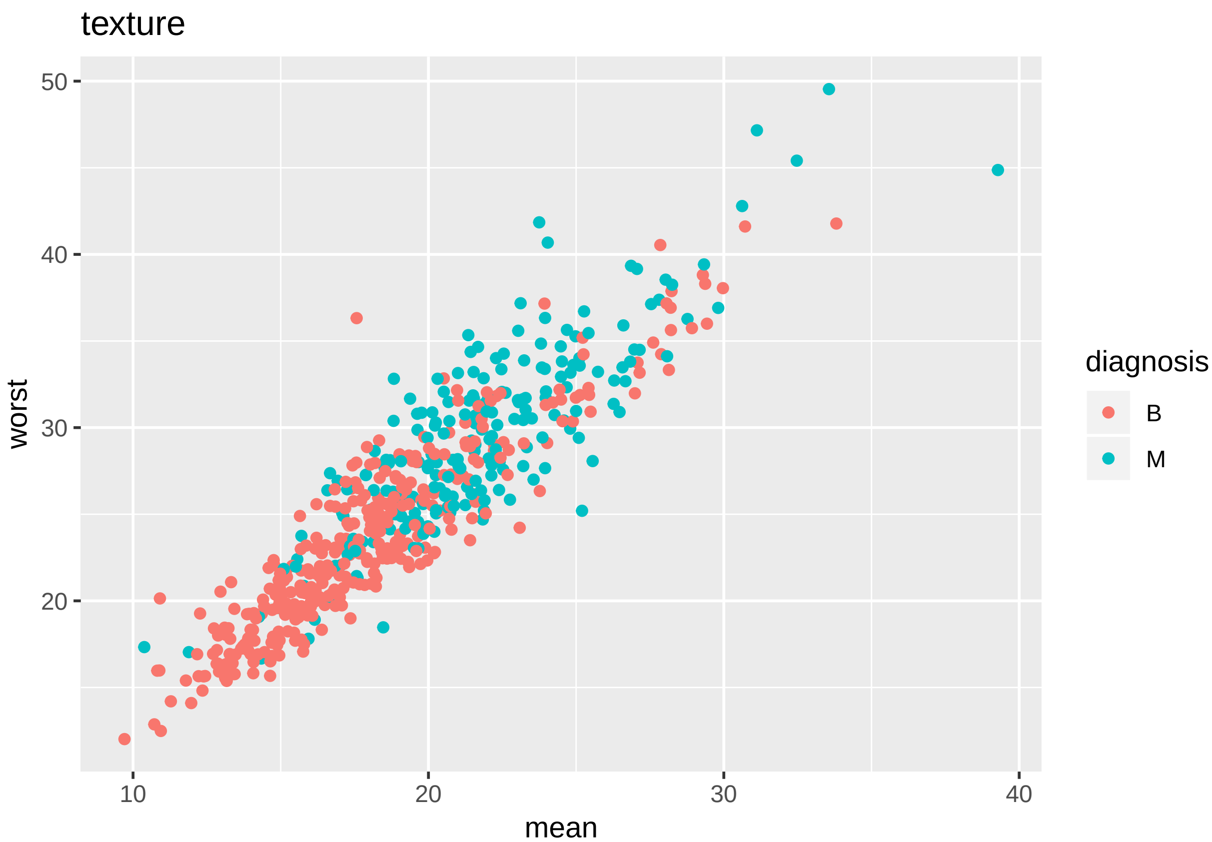
Compare Field Stats to Response

```
# plot diagnosis by mean/worst measurement for each variable
plot_f <- function(d){
  # names are _mean, _se, _worst
  name_f <- names(d)

  # plot points with diagnosis as color, change label
  p <- ggplot(d, aes(color = cancer_data$diagnosis)) +
    geom_point(aes_string(x = name_f[1], y = name_f[3])) +
    labs(title = str_remove(name_f[1], '_mean'),
         x = 'mean', y = 'worst', color='diagnosis')
  return(p)
}

# plot distribution for each field
lapply(data_by_field, plot_f)
```

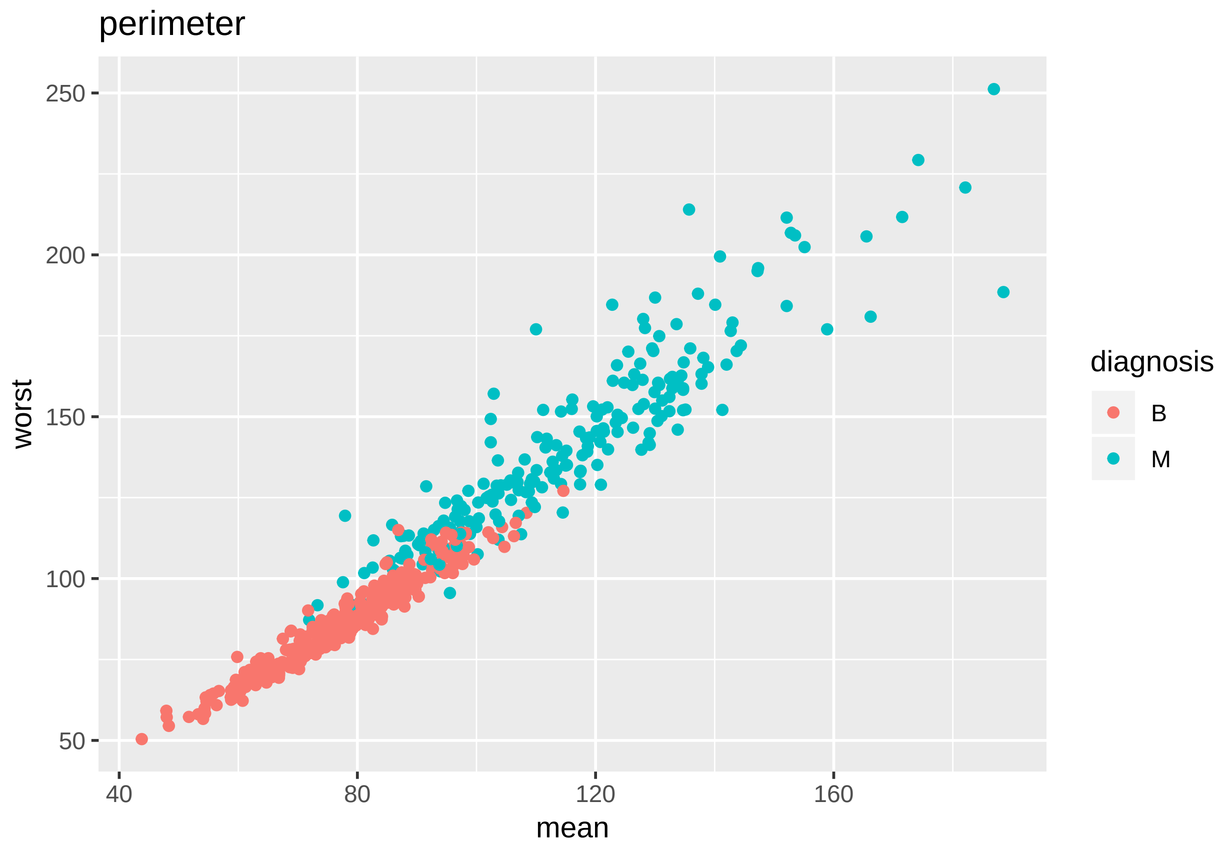
```
## $radius
```



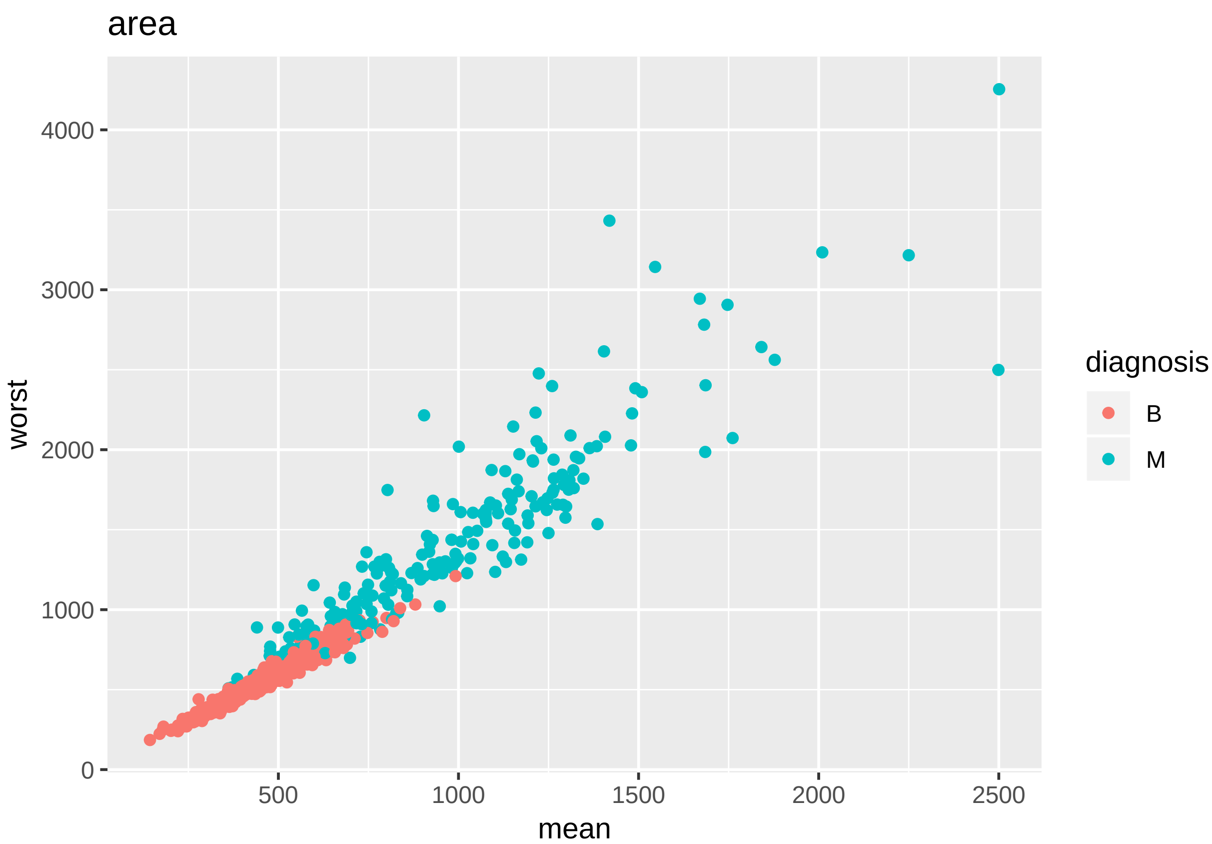

```
##
## $texture
```



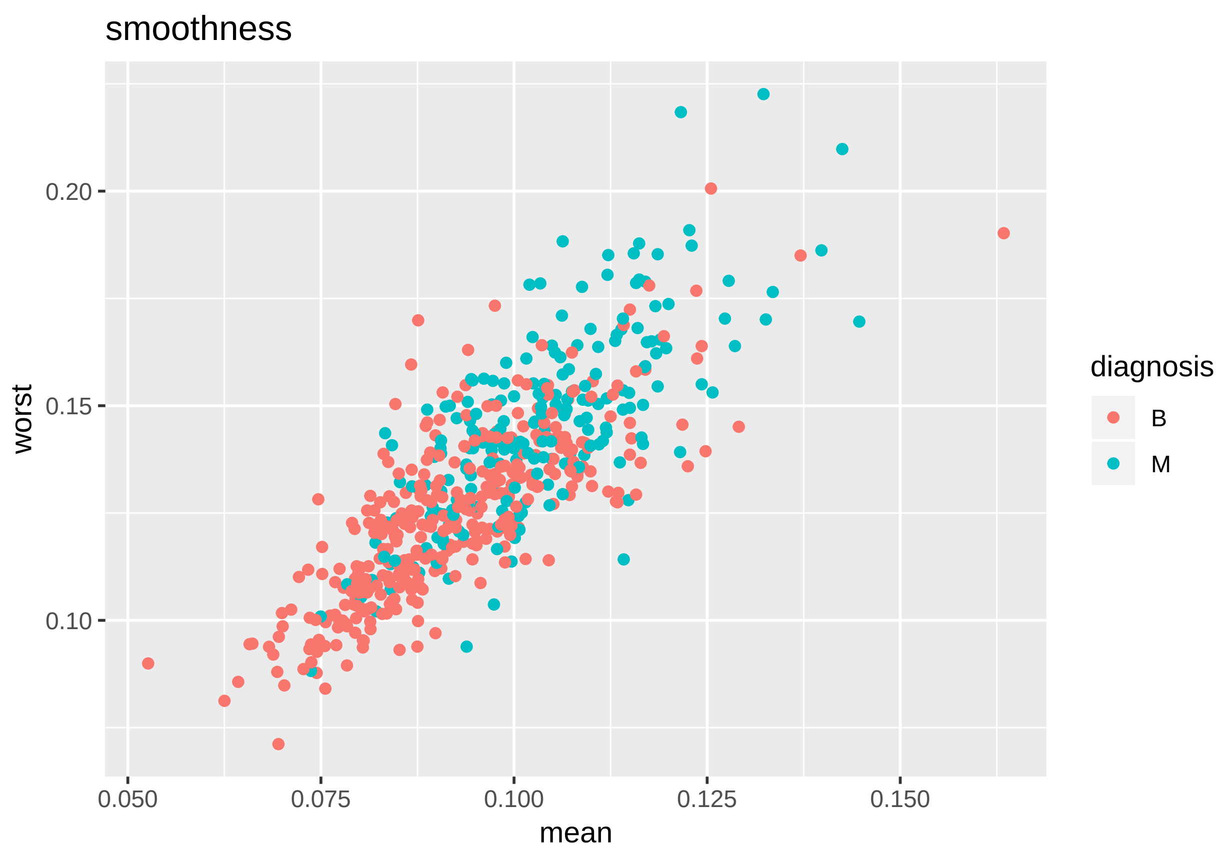

```
##
## $perimeter
```


perimeter
250
200
150
100
50
40
80
120
160
worst
mean
diagnosis
B
M

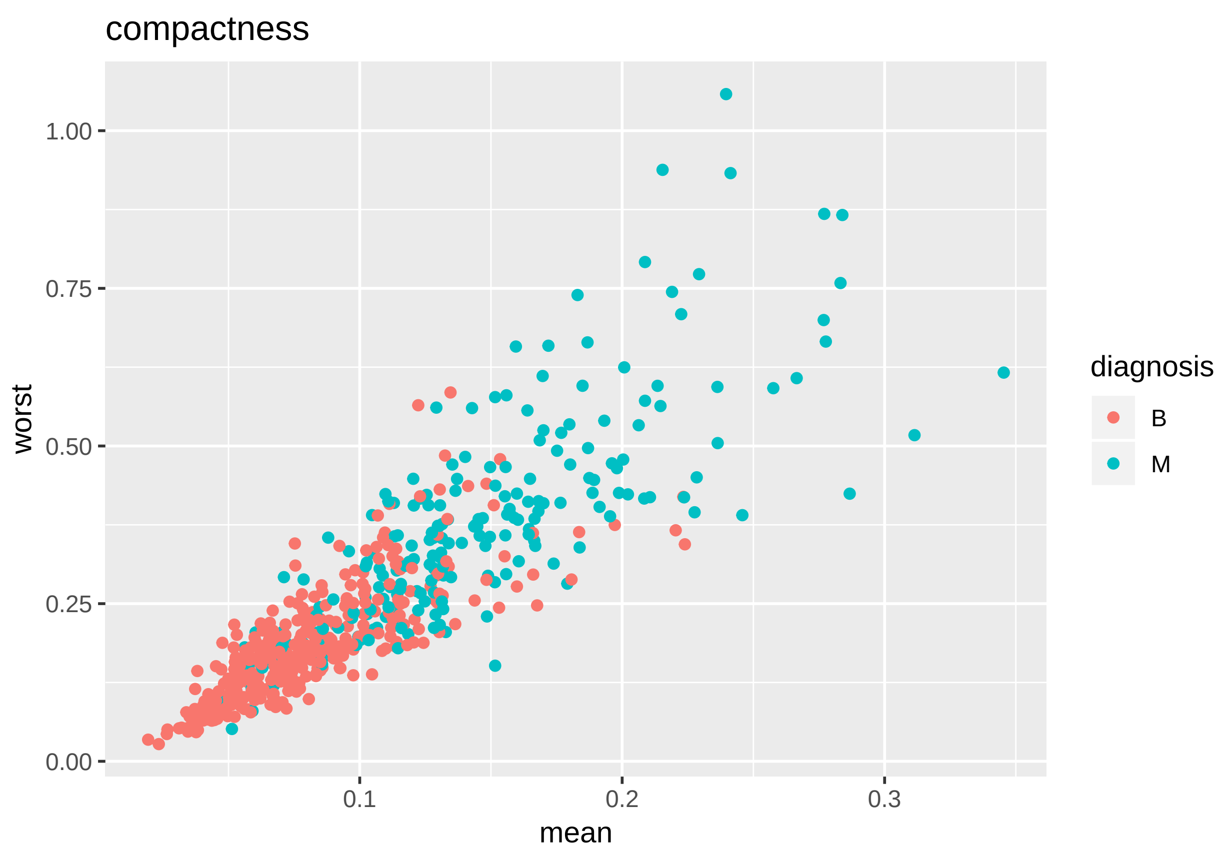
```
##
## $area
```


area
worst
mean
4000
3000
2000
1000
0
500
1000
1500
2000
2500
diagnosis
B
M

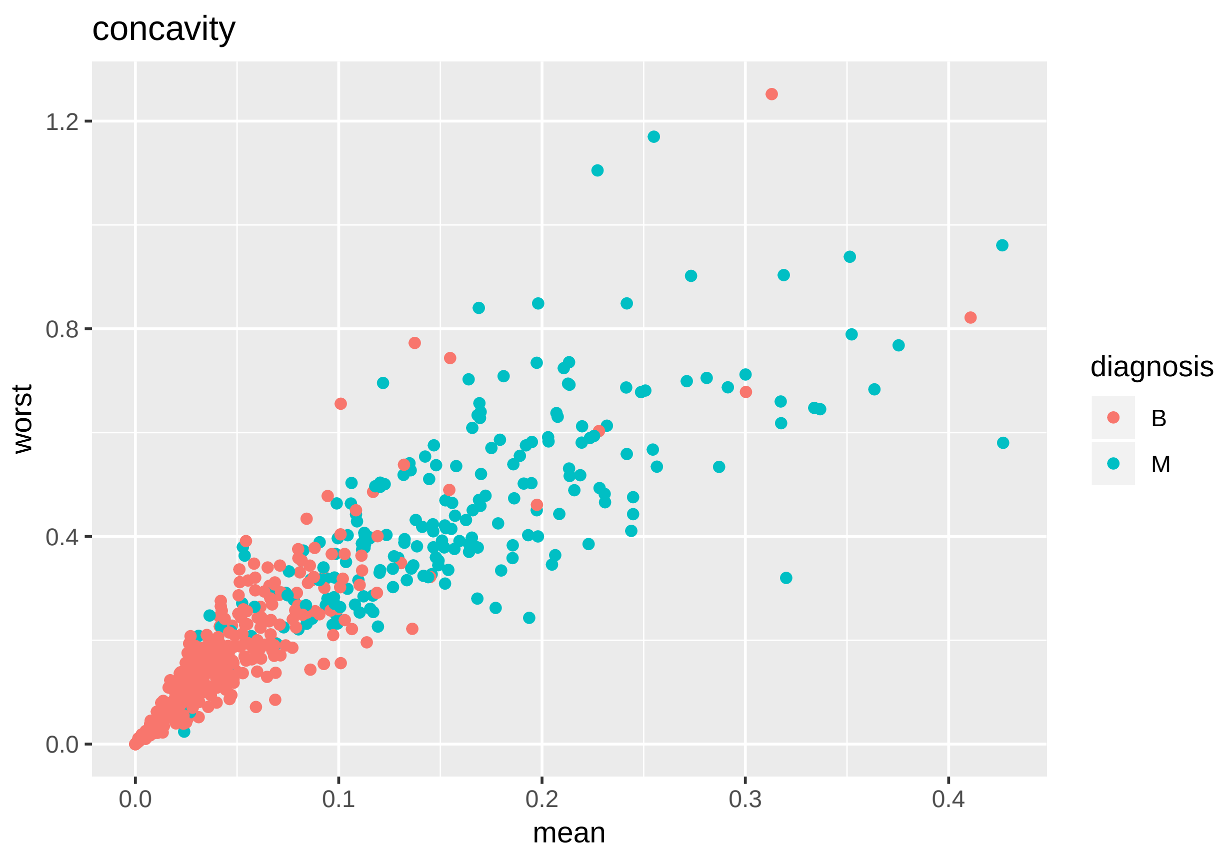
```
##
## $smoothness
```



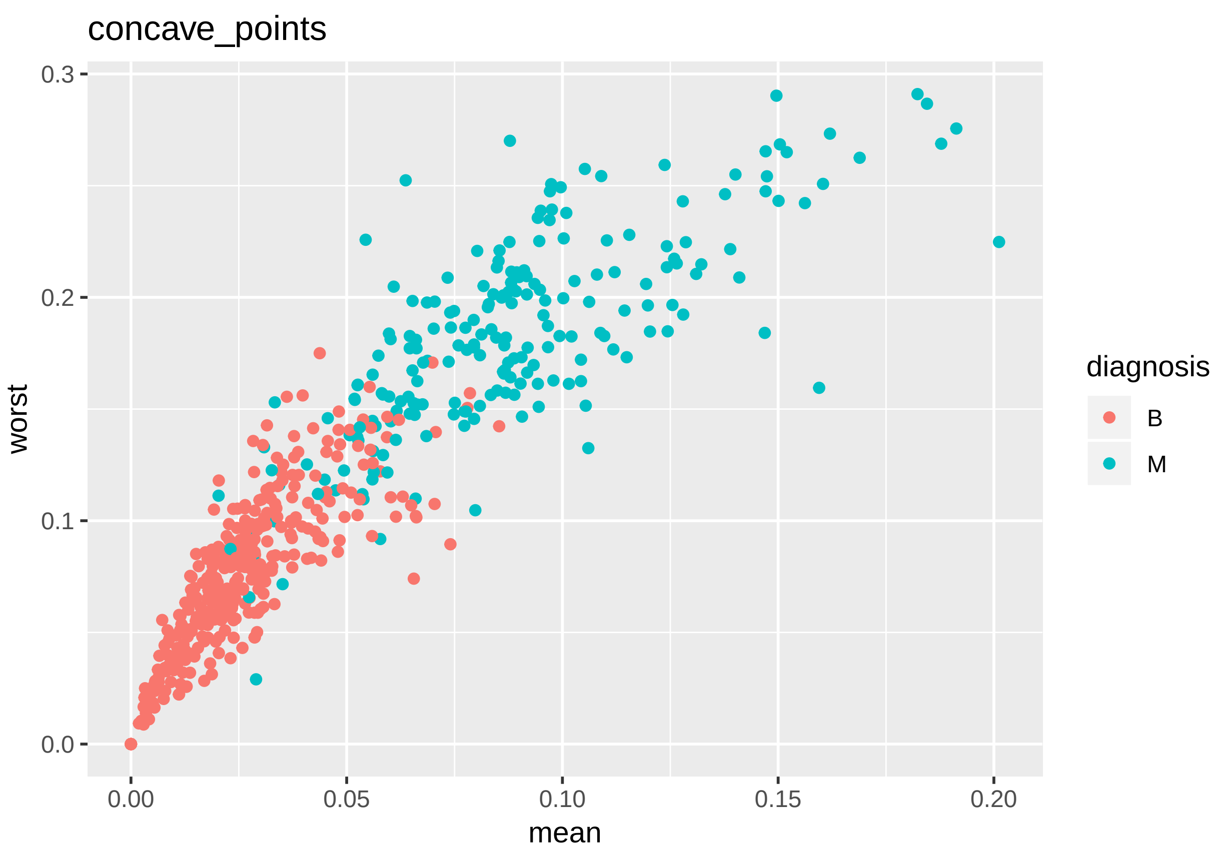

```
##
## $compactness
```


compactness
worst
1.00
0.75
0.50
0.25
0.00
0.1
0.2
0.3
mean
diagnosis
B
M

```
##
## $concavity
```

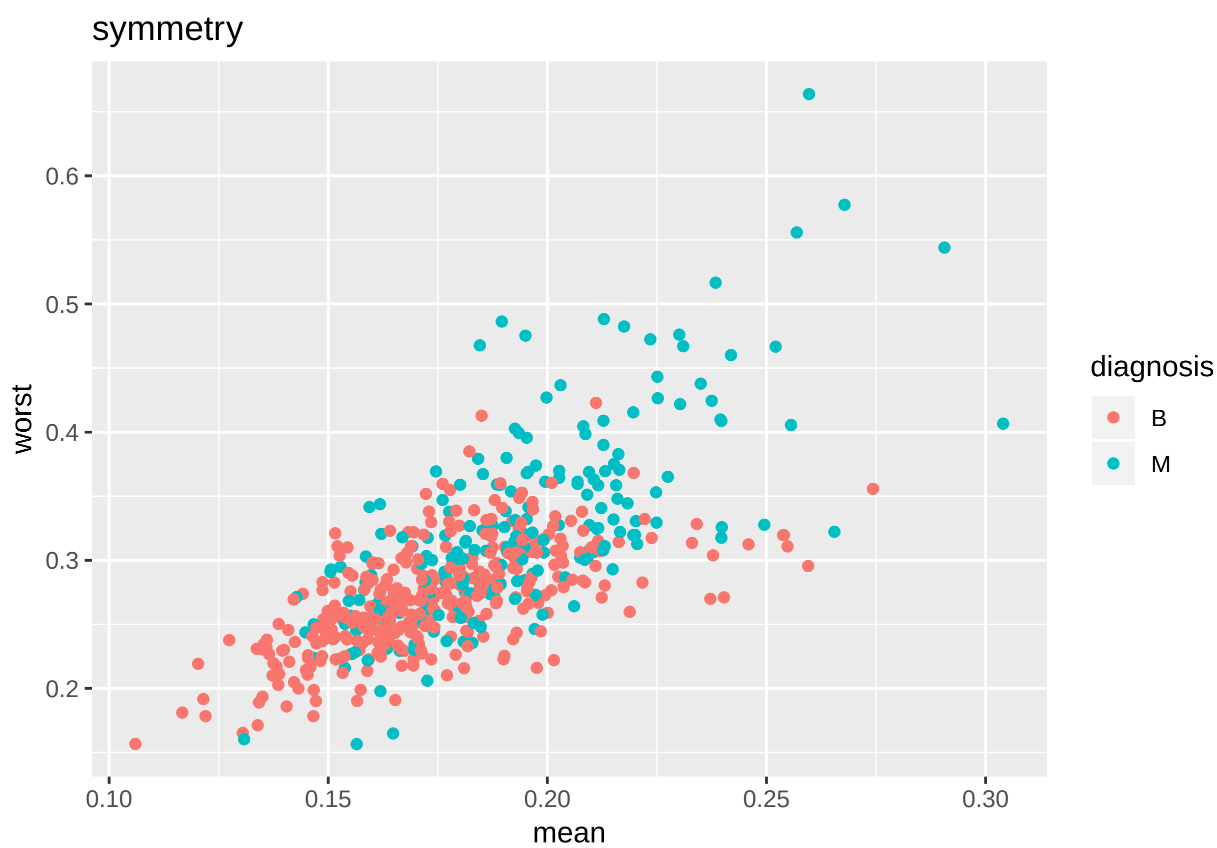



```
##
## $concave_points
```

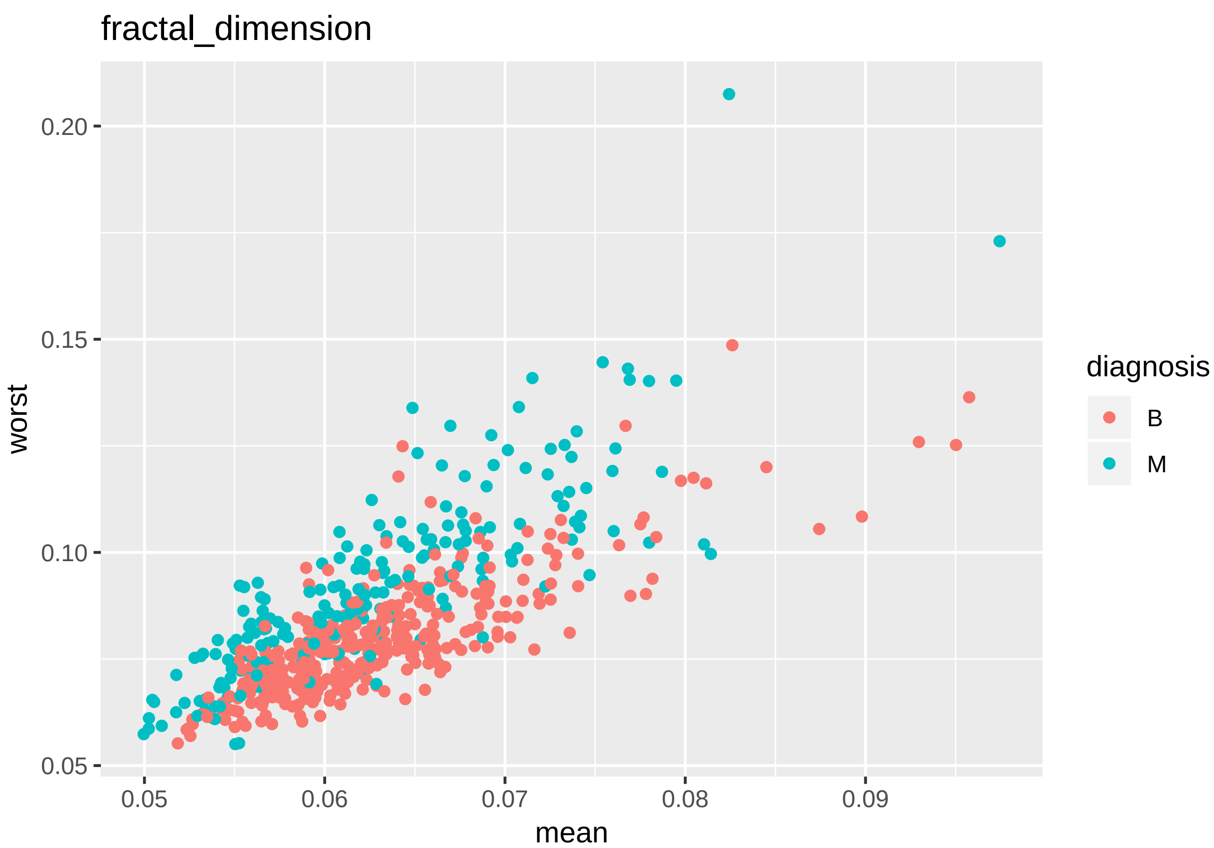


concave_points
0.3
0.2
0.1
0.0
worst
0.00
0.05
0.10
0.15
0.20
mean
diagnosis
B
M


```
##
## $symmetry
```




```
##
## $fractal_dimension
```



These plots of the mean and worst measurements for each field over the resultant diagnosis show a relatively consistent pattern of higher measurements resulting in more malignant diagnoses. Since the worst and mean relationships are intimately related to one another and to the standard error, it makes sense that the plots show direct correlation between statistics for many of the variables.
We can expect standard errors to be higher for observations where the worst measurement is >> than the mean of all measurements, but in this case, we left standard error out of the visualizations for the sake of clarity.

Form Training and Test Data Sets

```
# set random seed
set.seed(1847)
p <- 0.2 # proportion of test data
m <- 569 # number of observations

train_inds <- sample.int(m, (1-p)*m)
train_d <- cancer_data[train_inds,]
test_d <- cancer_data[-train_inds,]

dim(train_d)
```

```
## [1] 455  32
```

```
dim(test_d)
```

```
## [1] 114  32
```

Model Evaluation Function

```r
# Function for generating confusion matrix and performance evaluation statistics for a model
# * mod = statistical model
# * test = test data set to evaluate model with
# * y = response field from test dataset
confusion_eval <- function(mod, test, y){
  pred <- predict(mod, newdata = test, type = 'class')
  conf <- table(pred=pred, actual=y)
  accuracy <- sum(diag(conf)) / sum(conf)
  error_rate <- 1 - accuracy
  sensitivity <- conf[1,1] / sum(conf[,1])
  precision <- conf[1,1] / sum(conf[1,])
  miss_rate <- 1 - sensitivity
  fall_out <- 1 - precision
  f1 <- 2 * (precision * sensitivity) / (precision + sensitivity)
  return(list(
    prediction = pred,
    confusion = conf,
    stat = list(
      accuracy = accuracy,
      error_rate = error_rate,
      sensitivity = sensitivity,
      precision = precision,
      miss_rate = miss_rate,
      fall_out = fall_out,
      f1 = f1
    )
  ))
}
```

Decision Tree Classification

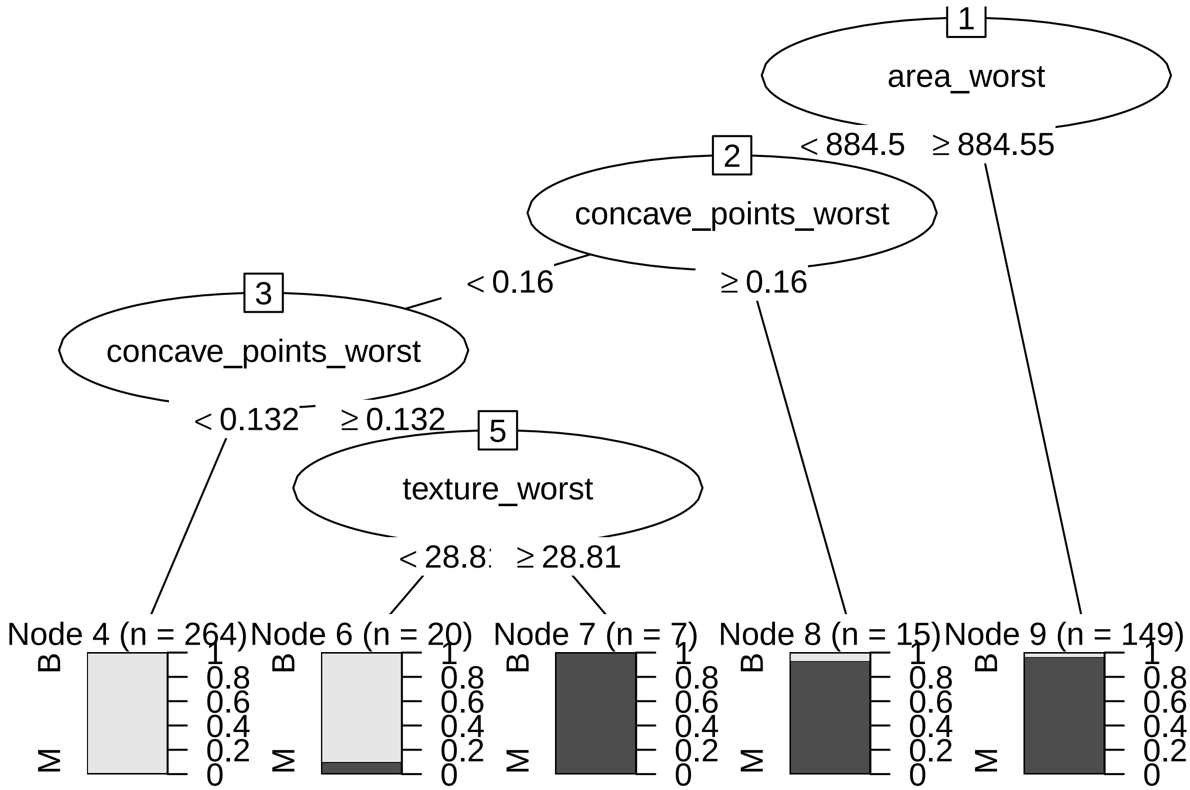
Training Model on All

```r
# create decision tree using all training data
dtm <- rpart(diagnosis~., data = train_d)
dtm
```

```
## n= 455
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 455 170 B (0.62637363 0.37362637)
##    2) area_worst< 884.55 306  26 B (0.91503268 0.08496732)
##      4) concave_points_worst< 0.1603 291  12 B (0.95876289 0.04123711)
##        8) concave_points_worst< 0.13235 264   3 B (0.98863636 0.01136364) *
##        9) concave_points_worst>=0.13235 27   9 B (0.66666667 0.33333333)
##         18) texture_worst< 28.81 20   2 B (0.90000000 0.10000000) *
##         19) texture_worst>=28.81 7   0 M (0.00000000 1.00000000) *
##      5) concave_points_worst>=0.1603 15   1 M (0.06666667 0.93333333) *
```

```
##    3) area_worst>=884.55 149   5 M (0.03355705 0.96644295) *
```

```r
# plot decision tree
plot(as.party(dtm))
```



Performance Evaluation

```r
# generate predictions and confusion matrix for decision tree model
confusion_eval(mod = dtm, test = test_d, y = test_d$diagnosis)
```

```
## $prediction
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##   M   M   M   M   B   M   M   B   M   M   M   M   M   B   M   B   M   M
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##   B   B   B   M   B   B   B   B   M   M   M   B   B   B   B   M   M   M
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##   M   M   M   B   M   B   B   B   M   M   B   B   B   M   B   B   M   M
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##   B   B   B   B   B   B   B   B   B   B   M   B   B   B   M   B   M   B
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   B   B   M   B   B   M   B   B   B   B   B   B   B   B   B   B   B   B
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
##   B   B   M   B   B   B   M   M   B   M   B   B   B   B   B   M   B   B
## 109 110 111 112 113 114
##   B   B   B   B   M   M
## Levels: B M
```

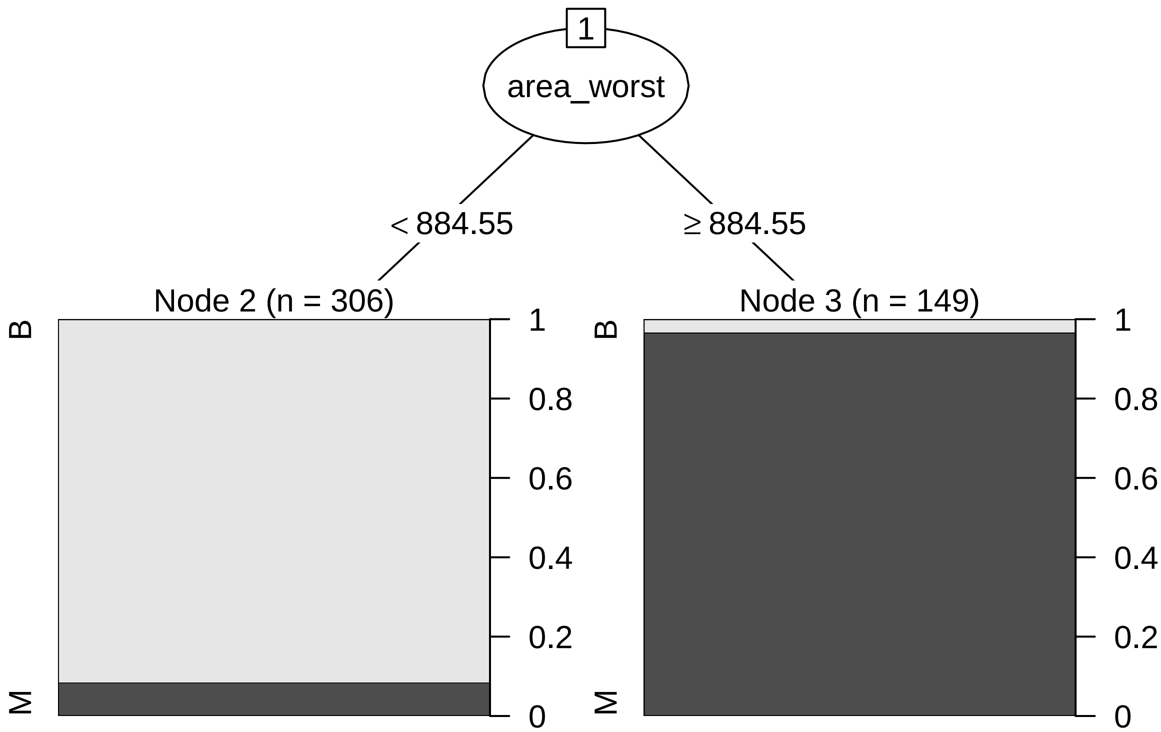
```
##
## $confusion
##     actual
## pred  B  M
##    B 68  4
##    M  4 38
##
## $stat
## $stat$accuracy
## [1] 0.9298246
##
## $stat$error_rate
## [1] 0.07017544
##
## $stat$sensitivity
## [1] 0.9444444
##
## $stat$precision
## [1] 0.9444444
##
## $stat$miss_rate
## [1] 0.05555556
##
## $stat$fall_out
## [1] 0.05555556
##
## $stat$f1
## [1] 0.9444444
```

Testing Pruning

```r
# Update decision tree,
# pruning such that branching must improve performance by at least 10% for each split
dtm_10 <- prune.rpart(dtm, cp = 0.1)
dtm_10
```

```
## n= 455
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 455 170 B (0.62637363 0.37362637)
##   2) area_worst< 884.55 306  26 B (0.91503268 0.08496732) *
##   3) area_worst>=884.55 149   5 M (0.03355705 0.96644295) *
```

```r
# plot updated model
plot(as.party(dtm_10))
```



Pruning Performance Evaluation

```
# generate predictions and confusion matrix for pruned decision tree model
conf <- confusion_eval(mod = dtm_10, test = test_d, y = test_d$diagnosis)
conf
```

```
## $prediction
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##   M   B   M   B   B   B   M   B   M   M   M   M   M   B   M   B   B   M
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##   B   B   B   M   B   B   B   B   M   M   M   B   B   B   B   M   M   M
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##   M   M   M   B   B   B   B   B   M   B   B   B   B   B   B   B   M   M
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##   B   B   B   B   B   B   B   B   B   B   M   B   B   B   M   B   M   B
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   B   B   M   B   B   M   B   B   B   B   B   B   B   B   B   B   B   B
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
##   B   B   M   B   B   B   M   M   B   M   B   B   B   B   B   B   B   B
## 109 110 111 112 113 114
##   B   B   B   B   M   M
## Levels: B M
##
## $confusion
##     actual
```

```
## pred  B  M
##    B 69 11
##    M  3 31
##
## $stat
## $stat$accuracy
## [1] 0.877193
##
## $stat$error_rate
## [1] 0.122807
##
## $stat$sensitivity
## [1] 0.9583333
##
## $stat$precision
## [1] 0.8625
##
## $stat$miss_rate
## [1] 0.04166667
##
## $stat$fall_out
## [1] 0.1375
##
## $stat$f1
## [1] 0.9078947
```

Pruning the decision tree by penalizing branches that didn't greatly improve the model accuracy ended up creating a much simpler model that only required one split to correctly categorize 87% of the test data. This is pretty interesting because it means that almost 90% of the decision in this decision tree is just looking at the worst area measurements to determine if a tumor is benign or malignant. One thing we may want to consider in this pruned model is that the simpler tree produced more false negatives, incorrectly predicting that a malignant tumor was benign. Since it would always be better in this situation to have false positive that start getting treated than false negatives that never get treated, it may be worth the added complexity of the original model if it can decrease the rate of false negatives.

Random Forest Classification

Training Model on All

```r
# set random seed
set.seed(1847)

# get data in form for random forest model
train_rf <- train_d %>% select(-diagnosis)
y_rf <- as.factor(train_d$diagnosis)
test_rf <- test_d %>% select(-diagnosis)
ytest_rf <- as.factor(test_d$diagnosis)

# create random forest model
rfm <- randomForest(x = train_rf, y = y_rf)
rfm
```

```
##
## Call:
```

```
##  randomForest(x = train_rf, y = y_rf)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 3.52%
## Confusion matrix:
##     B   M class.error
## B 280   5  0.01754386
## M  11 159  0.06470588
```

Performance Evaluation

```r
# generate predictions and confusion matrix for random forest model
confusion_eval(mod = rfm, test = test_rf, y = ytest_rf)
```

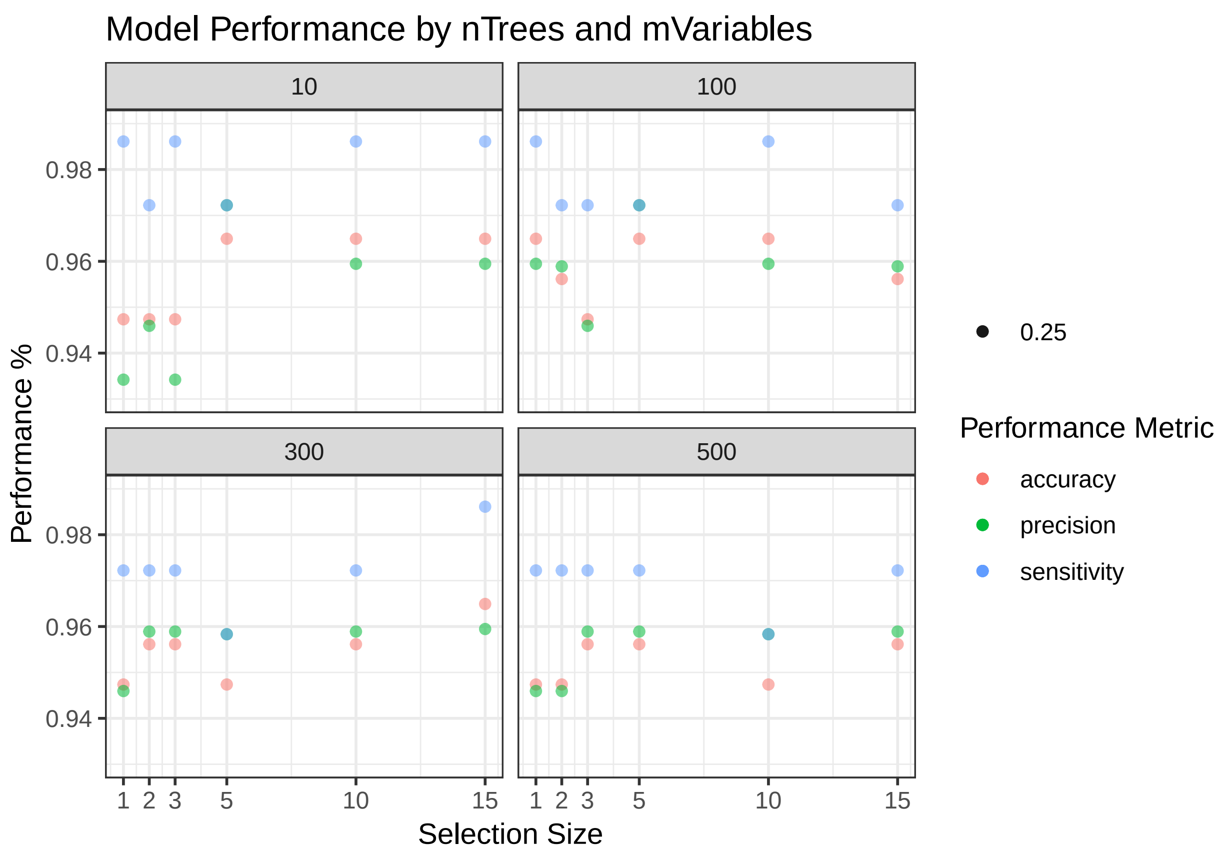
```
## $prediction
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##   M   M   M   M   M   M   M   B   M   M   M   M   M   B   M   B   M   M
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##   M   B   B   M   B   B   B   B   M   M   B   B   B   B   B   M   M   M
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##   M   M   M   B   M   B   B   B   M   M   B   B   B   M   B   B   M   M
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##   B   B   B   B   B   B   B   B   B   B   M   B   B   B   M   B   M   B
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   B   B   M   B   B   M   B   B   B   B   B   B   B   B   B   B   B   B
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
##   B   B   B   B   B   B   M   M   B   M   B   B   B   B   B   B   B   B
## 109 110 111 112 113 114
##   B   B   B   B   M   M
## Levels: B M
##
## $confusion
##     actual
## pred  B  M
##    B 70  3
##    M  2 39
##
## $stat
## $stat$accuracy
## [1] 0.9561404
##
## $stat$error_rate
## [1] 0.04385965
##
## $stat$sensitivity
## [1] 0.9722222
##
## $stat$precision
## [1] 0.9589041
##
## $stat$miss_rate
```

```
## [1] 0.02777778
##
## $stat$fall_out
## [1] 0.04109589
##
## $stat$f1
## [1] 0.9655172
```

Testing Random Forest with Different Hyper-Parameters

```r
# set random seed
set.seed(1847)

# set parameters to test
mtries <- c(1, 2, 3, 5, 10, 15)
ntrees <- c(10, 100, 300, 500)
nms <- c()
rf_models <- list()
rf_conf_matrices <- list()
rf_stats <- data.frame(accuracy = c(),
                       error_rate = c(),
                       sensitivity = c(),
                       precision = c(),
                       miss_rate = c(),
                       fall_out = c(),
                       f1 = c(),
                       mtry = c(),
                       ntree = c())

for(i in mtries){
  for(j in ntrees){
    nm <- paste('mtry:', i, 'ntree:', j)
    nms <- c(nms, nm)
    mod <- randomForest(x = train_rf,
                        y = y_rf,
                        mtry=i,
                        ntree = j)
    rf_models[[nm]] <- mod
    evalMod <- confusion_eval(mod = mod, test = test_rf, y = ytest_rf)
    rf_conf_matrices[[nm]] <- evalMod$confusion
    rf_stats <- rbind(rf_stats, cbind(as.data.frame(evalMod$stat), data.frame(mtry = i, ntree = j)))
  }
}

ggplot(rf_stats) +
  geom_point(aes(x = mtry, y = accuracy, color = 'accuracy', alpha = 0.25)) +
  geom_point(aes(x = mtry, y = precision, color = 'precision', alpha = 0.25)) +
  geom_point(aes(x = mtry, y = sensitivity, color = 'sensitivity', alpha = 0.25)) +
  facet_wrap(~ntree) +
  labs(title = 'Model Performance by nTrees and mVariables',
       x = 'Selection Size',
       y = 'Performance %',
       color = 'Performance Metric',
```

```
       alpha = NULL) +
  scale_x_continuous(breaks = mtries, labels = mtries) +
  lims(y = c(.93,0.99)) +
  theme_bw()
```



All of the models tested ended up with metrics that all fell between 93-99%. While all the models performed well, we wanted to select a model that would reduce the risk of false negatives (in favor of false positives), since it would be far more dangerous to have a malignant tumor misdiagnosed as benign. Since a high sensitivity indicates a lower rate of false negatives to true positives, we wanted to prioritize a high sensitivity over accuracy and precision. Among the models that scored the best in sensitivity, accuracy and precision (respectively), the simplest was the model with 10 trees trained with 10 variables each.

Choose Top Perfoming Random Forest

```
top_rf_model <- rf_models$`mtry: 10 ntree: 10`
top_rf_model
```

```
##
## Call:
##  randomForest(x = train_rf, y = y_rf, ntree = j, mtry = i)
##                Type of random forest: classification
##                      Number of trees: 10
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 5.33%
```

```
## Confusion matrix:
##     B   M class.error
## B 271  12  0.04240283
## M  12 155  0.07185629
```

K Nearest Neighbors

Setup for KNN Models

```r
# set random seed
set.seed(1847)

# set up training and test data
train_knn <- train_rf
y_knn <- y_rf
test_knn <- test_rf
ytest_knn <- ytest_rf

# function to run knn model with k and return the performance metrics
test_k <- function(k){
  # create knn model of data
  knnm <- knn(train_knn, test_knn, cl = y_knn, k = k)


  # confusion evaluation for knn and metrics
  confusion_knn <- table(pred=ytest_knn, actual=knnm)
  accuracy_knn <- sum(diag(confusion_knn)) / sum(confusion_knn)
  error_rate_knn <- 1 - accuracy_knn
  sensitivity_knn <- confusion_knn[1,1] / sum(confusion_knn[,1])
  precision_knn <- confusion_knn[1,1] / sum(confusion_knn[1,])
  miss_rate_knn <- 1 - sensitivity_knn
  fall_out_knn <- 1 - precision_knn
  f1_knn <- 2 * (precision_knn * sensitivity_knn) / (precision_knn + sensitivity_knn)

  # return model performance for k
  return(list(
    'k' = k,
    'confusion' = confusion_knn,
    'accuracy' = accuracy_knn,
    'error_rate' = error_rate_knn,
    'sensitivity' = sensitivity_knn,
    'precision' = precision_knn,
    'miss_rate' = miss_rate_knn,
    'fall_out' = fall_out_knn,
    'f1' = f1_knn))
}
```

Performance Evaluation

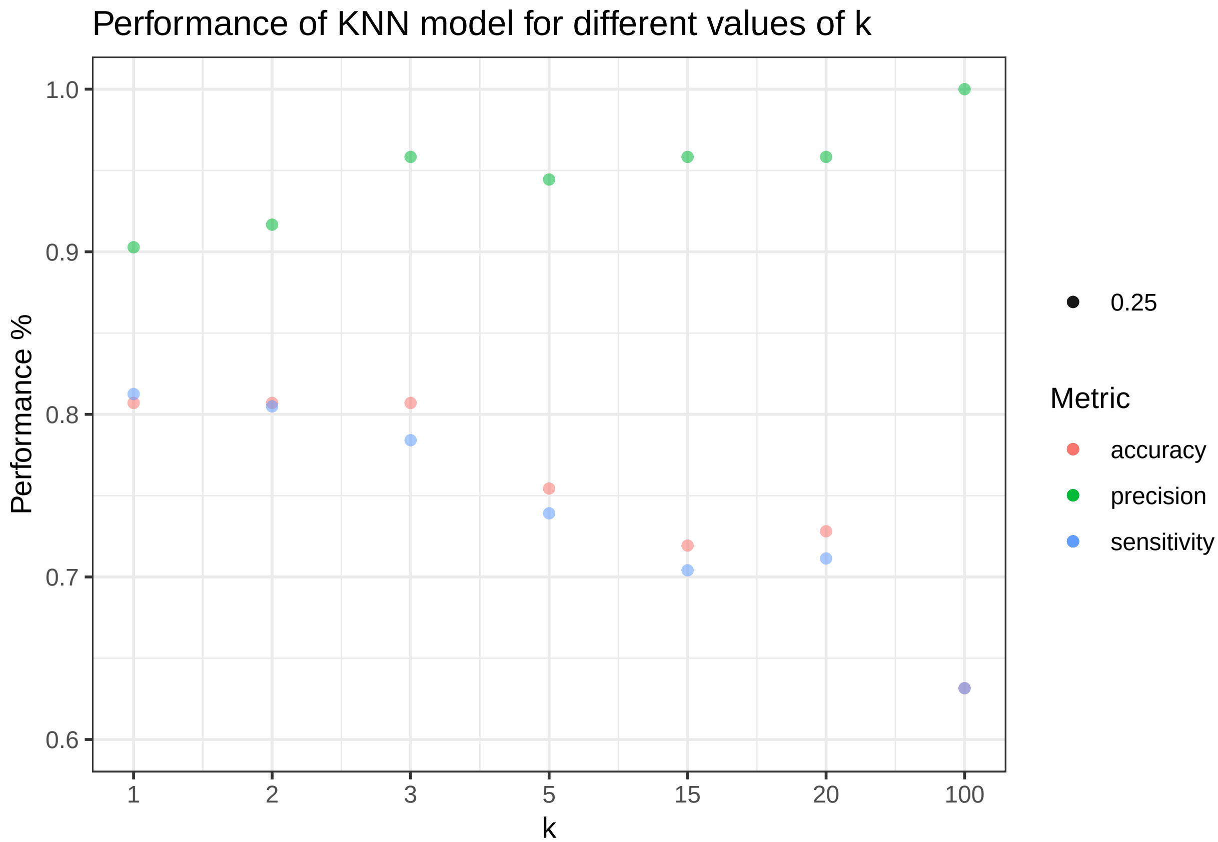
```r
# run knn modeling on values of k
ks <- c(1,2,3,5,15,20,100)
```

```r
knn_mods <- lapply(ks, test_k)

# recover model performance metrics
acc <- unlist(knn_mods %>% lapply('[[', 'accuracy'))
prec <- unlist(knn_mods %>% lapply('[[', 'precision'))
sens <- unlist(knn_mods %>% lapply('[[', 'sensitivity'))

# create performance data frame
knn_perf <- data.frame(ks, acc, prec, sens)

knn_perf
```

```
##    ks       acc      prec      sens
## 1   1 0.8070175 0.9027778 0.8125000
## 2   2 0.8070175 0.9166667 0.8048780
## 3   3 0.8070175 0.9583333 0.7840909
## 4   5 0.7543860 0.9444444 0.7391304
## 5  15 0.7192982 0.9583333 0.7040816
## 6  20 0.7280702 0.9583333 0.7113402
## 7 100 0.6315789 1.0000000 0.6315789
```

Choosing Top Performing KNN

```r
# plot performance metrics over k
ggplot(knn_perf, aes(x = c(1,2,3,4,5,6,7))) +
  geom_point(aes(y = acc, alpha = 0.25, color = 'accuracy')) +
  geom_point(aes(y = prec, alpha = 0.25, color = 'precision')) +
  geom_point(aes(y = sens, alpha = 0.25, color = 'sensitivity')) +
  labs(title='Performance of KNN model for different values of k',
       x = 'k', y = 'Performance %', color = 'Metric', alpha = NULL) +
  ylim(0.6, 1) +
  scale_x_continuous(breaks = c(1,2,3,4,5,6,7), labels = ks) +
  theme_bw()
```



Given that we want to find a model that is accurate but also prioritizes sensitivity over precision (since it would be more dangerous to falsely conclude that a tumor is benign than to falsely conclude that a tumor is malignant), we would want to choose the model for k=1. While k=1 had the highest rate of false positives on this data set, it alse had the lowest rate of false negatives, meaning that it would be unlikely to falsely conclude that a tumor is benign when it is actually malignant. Even so, the sensitivity was only around 80% in the best case, so this model would probably not work well for our use case.