James Valles
Final Project Report
DSC 441: Fundamentals of Data Science

Machine Learning Models Detect Breast Cancer

In this report, I will provide an overview of my data set, any pre-processing steps I took to prepare my data before modeling, the lessons learned, and the challenges faced along the way. I wil also discuss the creation and performance of three models: Model 1 - Decision Tree (Unpruned/Pruned), Model 2 - KNN, and Model 3 - Naive Bayes.

Problem Statement

According to statistics released on BreastCancer.org, in 2019, an estimated 268,600 women in the U.S. will be diagnosed with invasive breast cancer along with 62,930 new cases of non-invasive (in situ) breast cancer. When breast cancer is detected early, and is in the localized stage, the 5-year relative survival rate is 100%. I am thrilled to be working on creating models that can help detect breast cancer quickly and accurately to help save lives. The three supervised models in this report (decision tree, k-nearest neighbors, and naive bayes) make a prediction to classify whether a patient's breast tumor is malignant (M) or benign (B) using 17 out of 32 variables in the dataset (data.csv) describing the tumor's cell nuclei. They are as follows: radius_mean, texture_mean, smoothness mean, compactness_mean, symmetry_mean, fractal_dimension_mean, radius_se, texture_se, smoothness_se, compactness_se, concavity_se, concavity.points_se, symmetry_se, fractal_dimension_se, smoothness worst, symmetry_worst, fractal_dimension_worst. The models will train and test on records from 569 patients. Since the data set comes with a 'diagnosis' variable, the classification technique will be used as the results are known. Clustering is not applicable.

Data Description

The data for this project originally comes from the University of Wisconsin, which I found on Kaggle. A link to the data set can be found here: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data. According to the University of Wisconsin website, the data was donated on 11-01-1995 by Nick Street of the Computer Sciences Department at the University of Wisconsin. Additional creators of the data set include: Dr. William H Wolbery, General Surgery Dept. and Olvi L Mangasarian of the Computer Sciences Department both at the University of Wisconsin. As outlined in the article "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets," the data is derived from a "digitized image of a fine needle aspirate (FNA) of a breast mass" and describes characteristics of the cell nuclei present (refer to picture on the left). The cells are extracted from a small drop of fluid collected from the breast tumor using a needle. It is then spread on a glass slide, stained, and then photographed using a JVC TK-1070U color video camera, which is mounted on a Olyumpus microscope and then captured by a Computer Eyes RT color frame grabber. You can find a fascinating paper describing the complete collection process for the data set titled "Nuclear feature extraction for breast tumor diagnosis" by W.N. Street, W.H. Wolberg and O.L. Mangasarian in the final project's repository for review. This report provides a wealth of knowledge in understanding who and how the data was

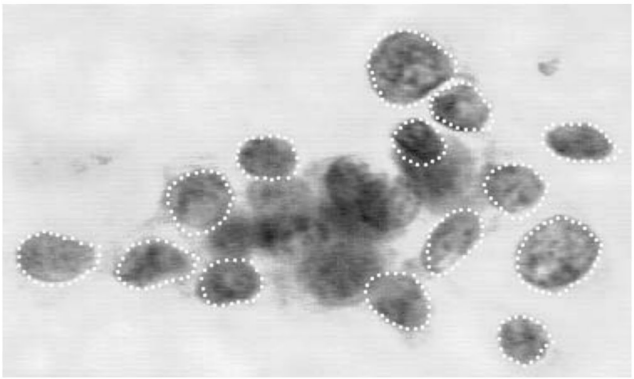


Figure 1: Initial Approximate Boundaries of Cell Nuclei
The user first draws a rough initial outline of some cell nucleus boundaries. Each outline serves as the initial position for a deformable spline which converges to an accurate boundary of the nucleus.

collected. Furthermore, it helps me understand what the attributes represent and gives me some domain knowledge to better understand the data and problem.

Overview of Data

The dataset contains 32 different variables broken down into three groups: mean, standard error, and worst (the mean of the three largest values). For each cell nucleus, 10 variables are calculated, the average is taken: radius_mean, texture_mean, perimeter_mean, area_mean, smoothness_mean, compactness_mean, Concavity_mean, concave points_mean, symmetry_mean, fractal_dimension_mean. According to the University of Wisconsin website, radius is the distance from the center to points on the perimeter." Texture is "standard deviation of gray-scale values." Smoothness has to do with "local variation in radius lengths." Compactness is calculated by squaring the perimeter dividing it by area - 1.0. Last, concavity is defined as "the severity of concave portions of the contour." Concave points is described as the "sum of concave portions of the contour." And, finally, fractal dimension is equivalent to "coastline approximation - 1." Additionally, the data set contains two variables that describe the patient (id) and (diagnosis) whether or not the tumor was M (malignant) or B (benign). The remaining 20 variables are as follows (standard error is denoted using "se" and the mean of the three largest values is denoted using "worst": radius_se, texture_se, perimeter_se, area_se, smoothness_se, compactness_se, concavity_se, concave, points_se, symmetry_se, fractal_dimension_se, radius_worst, texture_worst, perimeter_worst, area_worst, smoothness_worst, compactness_worst, concavity_worst, concave points_worst, symmetry_worst, fractal_dimension_worst. Overall, the data set contains 569 rows and 32 attributes. Furthermore, there is a class distribution of 357 benign, 212 malignant. The estimated file size is small - only 127KB.

Data Understanding

The second step of the CRISP-DM process is data understanding. I downloaded the data set and imported it in R. As part of the data understanding process, I need to see if the data set I obtained is usable and evaluate what I have. The first step, I ran some summary statistics using the str(df) command. I can see that there are 33 variables and 569 rows, of which most are of type num, except for id, which is an int. The variable 'Diagnosis' is of type Factor and has two levels B (Benign) and M (Malignant), and then, I notice a column labeled 'X,' which all of its rows are empty. I will immediately decide to drop that row as it has a number of missing values and may skew my model if I leave it in. Additionally, I can drop the id column as again that won't help much and is irrelevant to making a prediction. Below is a summary when running the str() function.

```
> str(df)
'data.frame':   569 obs. of  33 variables:
 $ id                     : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844981 84501001 ...
 $ diagnosis              : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ radius_mean            : num  18 20.6 19.7 11.4 20.3 ...
 $ texture_mean           : num  10.4 17.8 21.2 20.4 14.3 ...
 $ perimeter_mean         : num  122.8 132.9 130 77.6 135.1 ...
 $ area_mean              : num  1001 1326 1203 386 1297 ...
 $ smoothness_mean        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
 $ compactness_mean       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
 $ concavity_mean         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
 $ concave.points_mean    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
 $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 ...
 $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
 $ radius_se              : num  1.095 0.543 0.746 0.496 0.757 ...
 $ texture_se             : num  0.905 0.734 0.787 1.156 0.781 ...
 $ perimeter_se           : num  8.59 3.4 4.58 3.44 5.44 ...
 $ area_se                : num  153.4 74.1 94 27.2 94.4 ...
 $ smoothness_se          : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
 $ compactness_se         : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
 $ concavity_se           : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
 $ concave.points_se      : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
 $ symmetry_se            : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
 $ fractal_dimension_se   : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
 $ radius_worst           : num  25.4 25 23.6 14.9 22.5 ...
 $ texture_worst          : num  17.3 23.4 25.5 26.5 16.7 ...
 $ perimeter_worst        : num  184.6 158.8 152.5 98.9 152.2 ...
 $ area_worst             : num  2019 1956 1709 568 1575 ...
 $ smoothness_worst       : num  0.162 0.124 0.144 0.21 0.137 ...
 $ compactness_worst      : num  0.666 0.187 0.424 0.866 0.205 ...
 $ concavity_worst        : num  0.712 0.242 0.45 0.687 0.4 ...
 $ concave.points_worst   : num  0.265 0.186 0.243 0.258 0.163 ...
 $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
 $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
 $ X                      : logi  NA NA NA NA NA NA ...
>
```

As I continue to evaluate the data, I want to get a more comprehensive look at my data's consistency, completeness, plausibility, and whether or not there are any outliers, which can impact my model's performance. From the statistics outlined above, I am noticing the data is consistent, numbers appears to be numbers, I'm not noticing any strings, or weird formatting issues. So, we can mark that off my list of things to fix.

Next, I will run skim() function, to get a more complete picture of missing values, the variable's mean, standard deviation, and percentiles.

```
> skim(df)
Skim summary statistics
 n obs: 569
 n variables: 33

── Variable type:factor ─────────────────────────────────────────────────────────────────────────────────────────────────────────
  variable missing complete   n n_unique           top_counts ordered
 diagnosis       0      569 569        2 B: 357, M: 212, NA: 0   FALSE

── Variable type:integer ────────────────────────────────────────────────────────────────────────────────────────────────────────
 variable missing complete   n  mean      sd   p0    p25    p50     p75    p100     hist
       id       0      569 569 3e+07 1.3e+08 8670 869218 906024 8813129 9.1e+08 ▇▁▁▁▁▁▁▁

── Variable type:logical ────────────────────────────────────────────────────────────────────────────────────────────────────────
 variable missing complete   n mean count
        X     569        0 569  NaN   569
```

```
── Variable type:numeric ────────────────────────────────────────────────────────────────────────────────────────────────────────
                variable missing complete   n     mean       sd        p0      p25      p50      p75     p100     hist
               area_mean       0      569 569 654.89   351.91   143.5     420.3    551.1    782.7    2501     ▇▃▁▁▁▁▁▁
                 area_se       0      569 569  40.34    45.49     6.8      17.85    24.53    45.19    542.2    ▇▁▁▁▁▁▁▁
              area_worst       0      569 569 880.58   569.36   185.2     515.3    686.5    1084     4254     ▇▃▁▁▁▁▁▁
            compactness_mean       0      569 569   0.1      0.053    0.019     0.065    0.093    0.13     0.35     ▅▇▃▁▁▁▁▁
              compactness_se       0      569 569   0.025    0.018    0.0023    0.013    0.02     0.032    0.14     ▇▃▁▁▁▁▁▁
            compactness_worst       0      569 569   0.25     0.16     0.027     0.15     0.21     0.34     1.06     ▇▇▃▁▁▁▁▁
          concave.points_mean       0      569 569   0.049    0.039    0         0.02     0.034    0.074    0.2      ▇▇▅▂▁▁▁▁
            concave.points_se       0      569 569   0.012    0.0062   0         0.0076   0.011    0.015    0.053    ▇▇▂▁▁▁▁▁
         concave.points_worst       0      569 569   0.11     0.066    0         0.065    0.1      0.16     0.29     ▇▇▅▂▁▁▁▁
               concavity_mean       0      569 569   0.089    0.08     0         0.03     0.062    0.13     0.43     ▇▃▂▁▁▁▁▁
                 concavity_se       0      569 569   0.032    0.03     0         0.015    0.026    0.042    0.4      ▇▁▁▁▁▁▁▁
               concavity_worst       0      569 569   0.27     0.21     0         0.11     0.23     0.38     1.25     ▇▅▂▁▁▁▁▁
        fractal_dimension_mean       0      569 569   0.063    0.0071   0.05      0.058    0.062    0.066    0.097    ▇▇▂▁▁▁▁▁
          fractal_dimension_se       0      569 569   0.0038   0.0026   0.00089   0.0022   0.0032   0.0046   0.03     ▇▂▁▁▁▁▁▁
       fractal_dimension_worst       0      569 569   0.084    0.018    0.055     0.071    0.08     0.092    0.21     ▇▇▂▁▁▁▁▁
                perimeter_mean       0      569 569  91.97    24.3     43.79     75.17    86.24    104.1    188.5    ▇▇▃▁▁▁▁▁
                  perimeter_se       0      569 569   2.87     2.02     0.76      1.61     2.29     3.36     21.98    ▇▁▁▁▁▁▁▁
               perimeter_worst       0      569 569 107.26    33.6     50.41     84.11    97.66    125.4    251.2    ▇▇▃▂▁▁▁▁
                   radius_mean       0      569 569  14.13     3.52     6.98     11.7     13.37    15.78    28.11    ▇▇▃▁▁▁▁▁
                     radius_se       0      569 569   0.41     0.28     0.11      0.23     0.32     0.48     2.87     ▇▁▁▁▁▁▁▁
                   radius_worst       0      569 569  16.27     4.83     7.93     13.01    14.97    18.79    36.04    ▇▇▅▂▁▁▁▁
              smoothness_mean       0      569 569   0.096    0.014    0.053     0.086    0.096    0.11     0.16     ▁▇▇▂▁▁▁▁
                smoothness_se       0      569 569   0.007    0.003    0.0017    0.0052   0.0064   0.0081   0.031    ▇▇▁▁▁▁▁▁
             smoothness_worst       0      569 569   0.13     0.023    0.071     0.12     0.13     0.15     0.22     ▁▃▇▂▁▁▁▁
                symmetry_mean       0      569 569   0.18     0.027    0.11      0.16     0.18     0.2      0.3      ▁▆▇▂▁▁▁▁
                  symmetry_se       0      569 569   0.021    0.0083   0.0079    0.015    0.019    0.023    0.079    ▇▇▁▁▁▁▁▁
               symmetry_worst       0      569 569   0.29     0.062    0.16      0.25     0.28     0.32     0.66     ▇▇▁▁▁▁▁▁
                 texture_mean       0      569 569  19.29     4.3      9.71     16.17    18.84    21.8     39.28    ▃▇▇▂▁▁▁▁
                   texture_se       0      569 569   1.22     0.55     0.36      0.83     1.11     1.47     4.88     ▇▅▁▁▁▁▁▁
                texture_worst       0      569 569  25.68     6.15    12.02     21.08    25.41    29.72    49.54    ▃▇▇▃▂▁▁▁
>
```

From the chart above, I am noticing some outliers in this dataset; for example, for the variable 'area_worst,' the mean is 880.59, but I am showing values of 4254. Additionally, area_se has a mean of 40.34, but I'm seeing a value at percentile 100 of 542.2. I think the best way to visualize outliers is by plotting - so I will do that next!

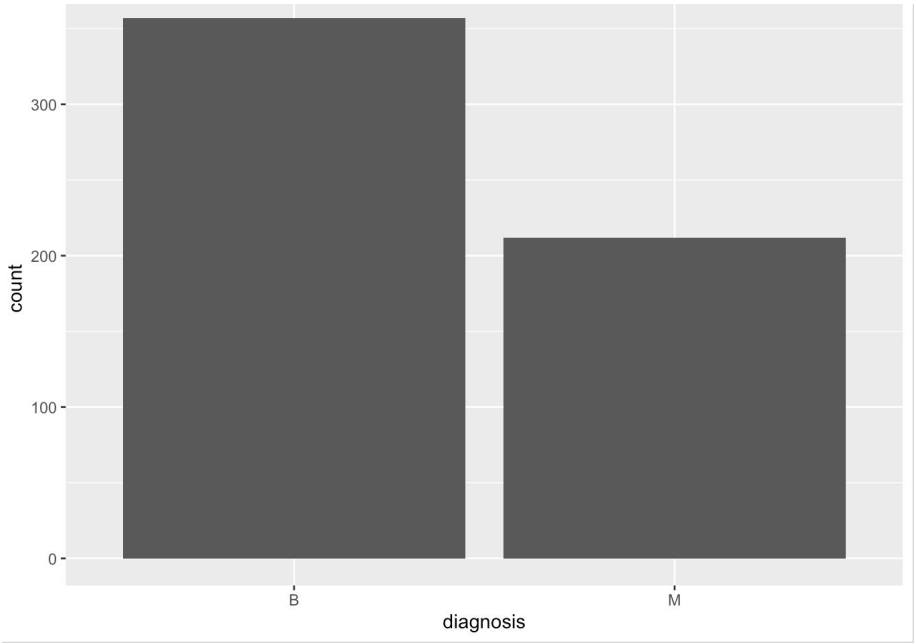
Furthermore, there appears to be no missing values at a glance, but this isn't always the case as there could still be values with weird formatting to indicate a missing value, such as 9999 or 0. In this datasets case, there are

569 rows with a value of NA for a variable called ‘X’. I received this information by running summary(df) and will remove them in the pre-processing step. Last, I will run my last function summary() which will provide additional information such as the minimum/maximum values for a given attribute. This is helpful as it will show possible outliers, and also reveal if there are any values that may not be plausible such as 0, a negative number, or a value that is extraordinarily high.

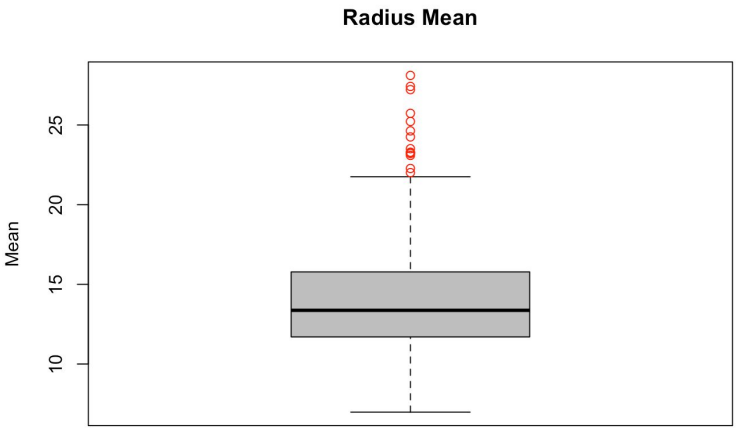
```
> summary(df)
       id              diagnosis  radius_mean      texture_mean    perimeter_mean     area_mean      smoothness_mean
 Min.   :     8670   B:357     Min.   : 6.981   Min.   : 9.71   Min.   : 43.79   Min.   : 143.5   Min.   :0.05263
 1st Qu.:   869218   M:212     1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17   1st Qu.: 420.3   1st Qu.:0.08637
 Median :   906024             Median :13.370   Median :18.84   Median : 86.24   Median : 551.1   Median :0.09587
 Mean   : 30371831             Mean   :14.127   Mean   :19.29   Mean   : 91.97   Mean   : 654.9   Mean   :0.09636
 3rd Qu.:  8813129             3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10   3rd Qu.: 782.7   3rd Qu.:0.10530
 Max.   :911320502             Max.   :28.110   Max.   :39.28   Max.   :188.50   Max.   :2501.0   Max.   :0.16340
 compactness_mean  concavity_mean    concave.points_mean symmetry_mean    fractal_dimension_mean   radius_se
 Min.   :0.01938   Min.   :0.00000   Min.   :0.00000     Min.   :0.1060   Min.   :0.04996        Min.   :0.1115
 1st Qu.:0.06492   1st Qu.:0.02956   1st Qu.:0.02031     1st Qu.:0.1619   1st Qu.:0.05770        1st Qu.:0.2324
 Median :0.09263   Median :0.06154   Median :0.03350     Median :0.1792   Median :0.06154        Median :0.3242
 Mean   :0.10434   Mean   :0.08880   Mean   :0.04892     Mean   :0.1812   Mean   :0.06288        Mean   :0.4052
 3rd Qu.:0.13040   3rd Qu.:0.13070   3rd Qu.:0.07400     3rd Qu.:0.1957   3rd Qu.:0.06612        3rd Qu.:0.4789
 Max.   :0.34540   Max.   :0.42680   Max.   :0.20120     Max.   :0.3040   Max.   :0.09744        Max.   :2.8730
   texture_se      perimeter_se      area_se       smoothness_se       compactness_se      concavity_se
 Min.   :0.3602   Min.   : 0.757   Min.   :  6.802   Min.   :0.001713   Min.   :0.002252   Min.   :0.00000
 1st Qu.:0.8339   1st Qu.: 1.606   1st Qu.: 17.850   1st Qu.:0.005169   1st Qu.:0.013080   1st Qu.:0.01509
 Median :1.1080   Median : 2.287   Median : 24.530   Median :0.006380   Median :0.020450   Median :0.02589
 Mean   :1.2169   Mean   : 2.866   Mean   : 40.337   Mean   :0.007041   Mean   :0.025478   Mean   :0.03189
 3rd Qu.:1.4740   3rd Qu.: 3.357   3rd Qu.: 45.190   3rd Qu.:0.008146   3rd Qu.:0.032450   3rd Qu.:0.04205
 Max.   :4.8850   Max.   :21.980   Max.   :542.200   Max.   :0.031130   Max.   :0.135400   Max.   :0.39600
 concave.points_se   symmetry_se       fractal_dimension_se  radius_worst   texture_worst   perimeter_worst
 Min.   :0.0000000   Min.   :0.007882   Min.   :0.0008948   Min.   : 7.93   Min.   :12.02   Min.   : 50.41
 1st Qu.:0.0007638   1st Qu.:0.015160   1st Qu.:0.0077480   1st Qu.:13.01   1st Qu.:21.08   1st Qu.: 84.11
 Median :0.0109930   Median :0.018730   Median :0.0031870   Median :14.97   Median :25.41   Median : 97.66
 Mean   :0.0111796   Mean   :0.020542   Mean   :0.0037949   Mean   :16.27   Mean   :25.68   Mean   :107.26
 3rd Qu.:0.0147100   3rd Qu.:0.023480   3rd Qu.:0.0045580   3rd Qu.:18.79   3rd Qu.:29.72   3rd Qu.:125.40
 Max.   :0.0529900   Max.   :0.078950   Max.   :0.0298400   Max.   :36.04   Max.   :49.54   Max.   :251.20
   area_worst     smoothness_worst  compactness_worst concavity_worst  concave.points_worst symmetry_worst
 Min.   : 185.2   Min.   :0.07117   Min.   :0.02729   Min.   :0.0000   Min.   :0.00000      Min.   :0.1565
 1st Qu.: 515.3   1st Qu.:0.11600   1st Qu.:0.14720   1st Qu.:0.1145   1st Qu.:0.06493      1st Qu.:0.2504
 Median : 686.5   Median :0.13130   Median :0.21190   Median :0.2267   Median :0.09993      Median :0.2822
 Mean   : 880.6   Mean   :0.13237   Mean   :0.25427   Mean   :0.2722   Mean   :0.11461      Mean   :0.2901
 3rd Qu.:1084.0   3rd Qu.:0.14600   3rd Qu.:0.33910   3rd Qu.:0.3829   3rd Qu.:0.16140      3rd Qu.:0.3179
 Max.   :4254.0   Max.   :0.22260   Max.   :1.05800   Max.   :1.2520   Max.   :0.29100      Max.   :0.6638
 fractal_dimension_worst    X
 Min.   :0.05504         Mode:logical
 1st Qu.:0.07146         NA's:569
 Median :0.08004
```

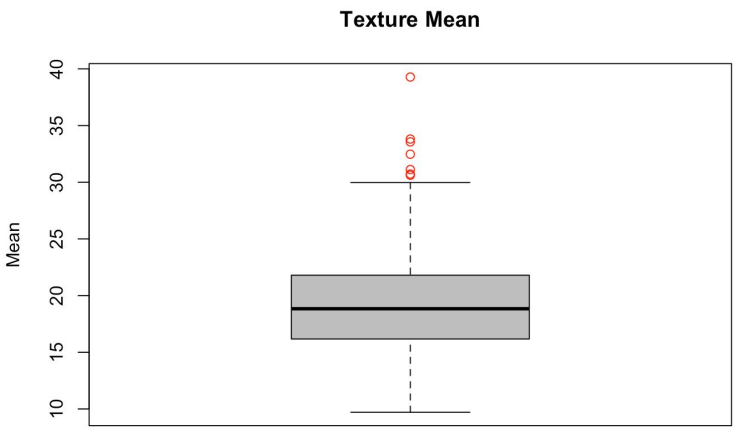
In looking at the summary() stats reveal there are a few variables that have a min value of 0: concave.points_mean, concavity mean, concavity_se, concave.points_se, concavity_worse, concavity.points_worst. This is where domain knowledge comes into play. What does this mean? Is this truly implausible to have zero values for these variables? According to the University of Wisconsin website the original source of the dataset, concavity is defined as “the severity of concave portions on the contour.” So, it could be possible that there were no concave portions; thus, a value of 0 may be reasonable.

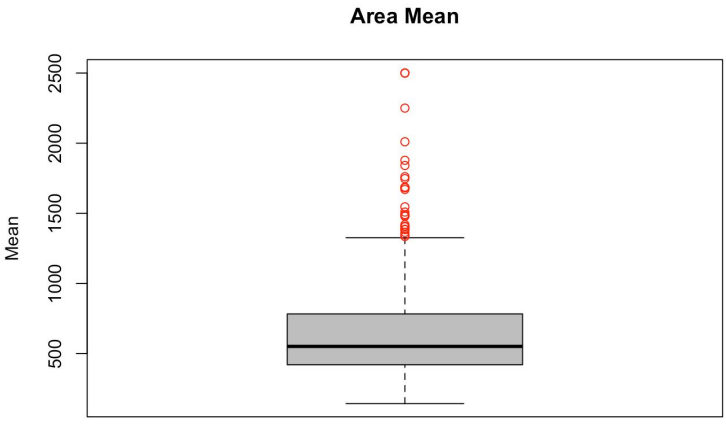
Additionally, since the dataset contains a label, ‘Diagnosis,’ which is what I want to predict (B or M), it is important that that the dataset be balanced. By running summary(df) I can see the class distribution of our categorical variable ‘Diagnosis.’ There are 357 rows that classify as benign (B) and 212 as malignant (M). There are a total of 569 rows. So the classification label has a proportion of 357/569 = 62.7% for B and 212/569 = 37.3% for M. This doesn’t appear to be a huge imbalance issue. However, it is important to consider as imbalanced data can lead to misleading metrics (i.e accuracy). In cases of severe imbalance, I may try over/under sampling, cross validation, or try focusing on other metrics such as AUC, f1 score, Kappa, which are balanced across all classes. Below, you will see a barplot of the variable ‘diagnosis’ that illustrates its distribution.

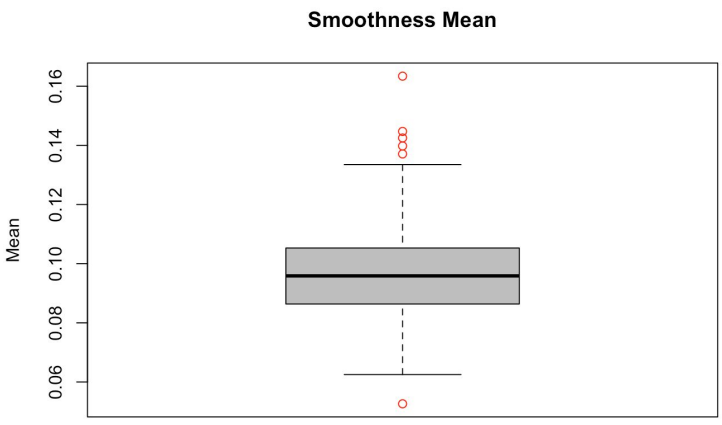


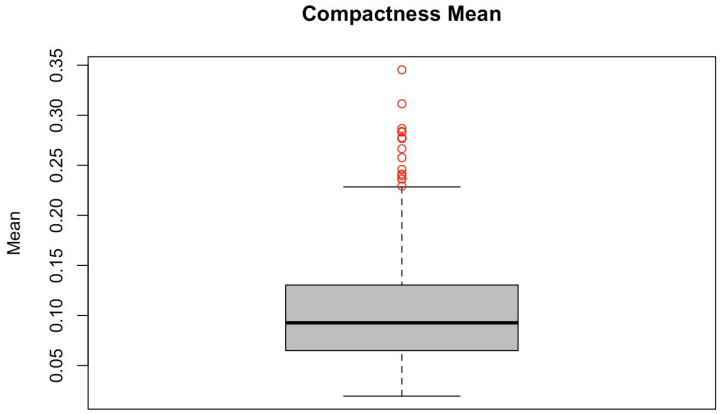
Next, I will plot the data to identify outliers and see the overall distribution of my variables. I will create a boxplot for each of the 31 variables. And, yes, there are outliers, which are outlined in the color red, but should I be concerned?

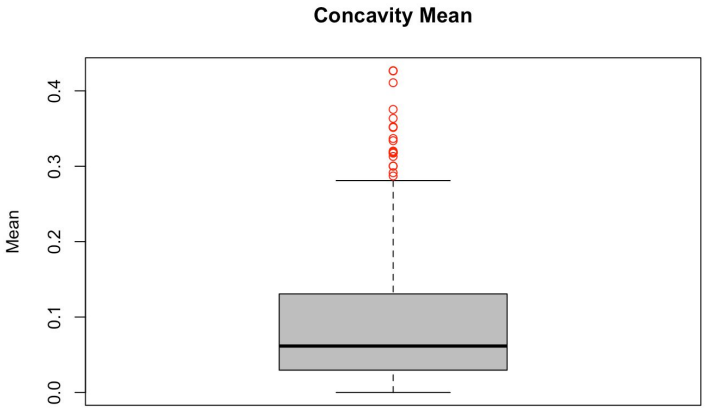


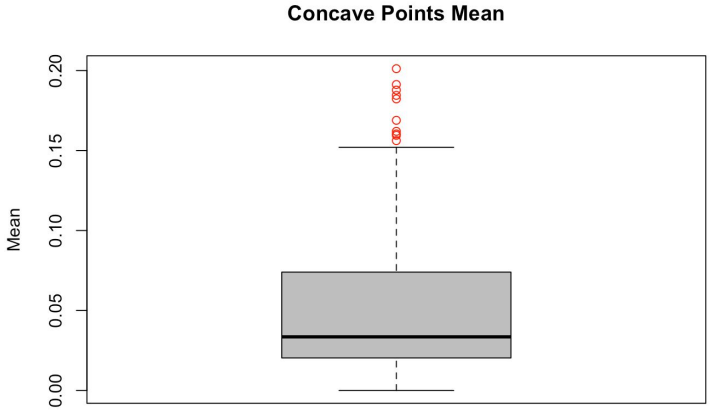


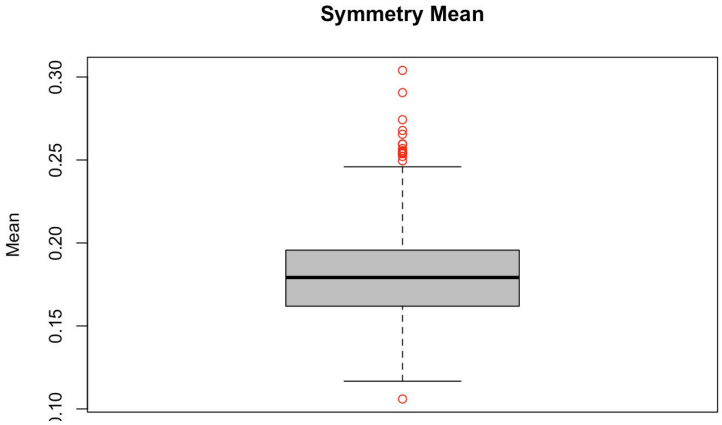


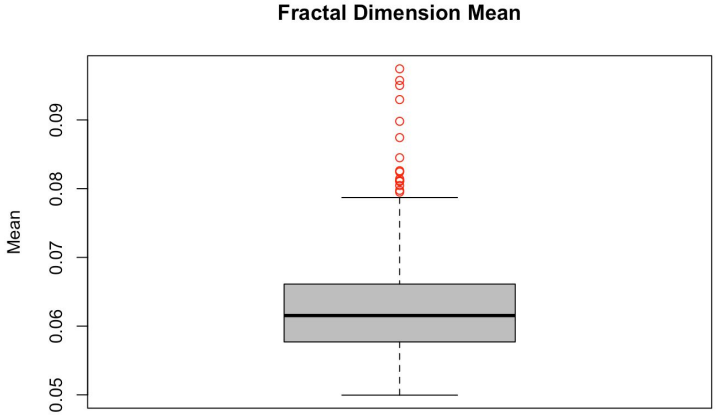


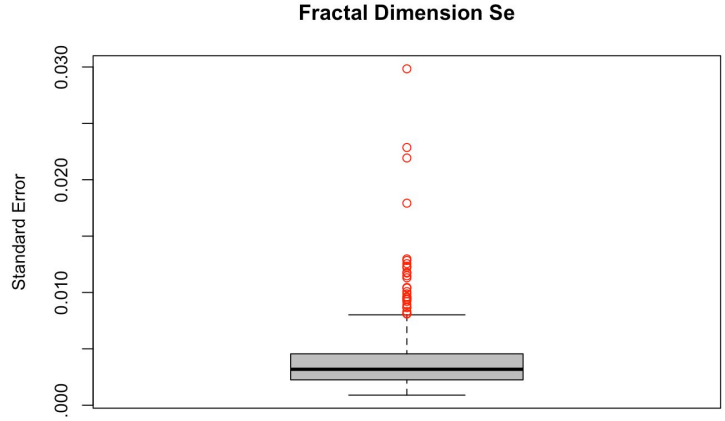

Compactness Mean
Mean
0.05
0.10
0.15
0.20
0.25
0.30
0.35

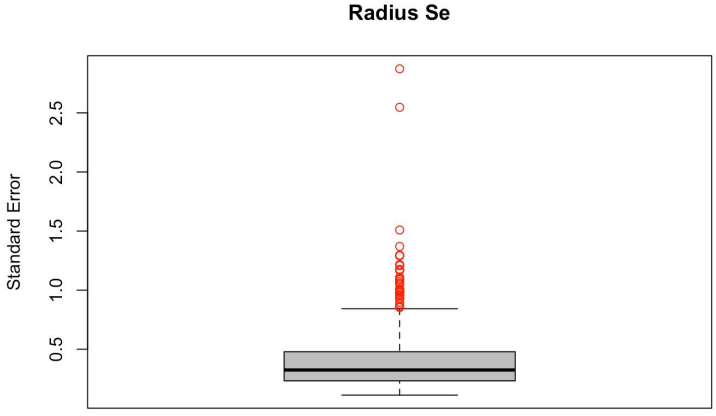

Concavity Mean
Mean
0.0
0.1
0.2
0.3
0.4

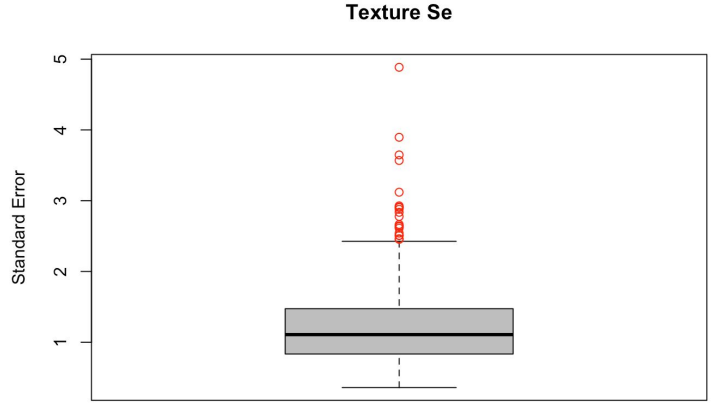

Concave Points Mean
Mean
0.00
0.05
0.10
0.15
0.20

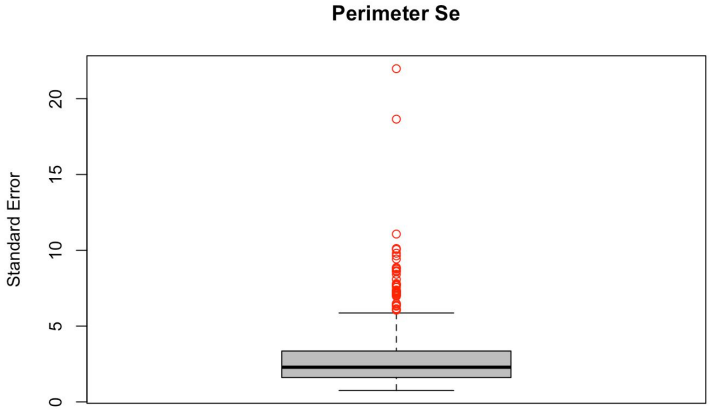

Symmetry Mean
Mean
0.10
0.15
0.20
0.25
0.30

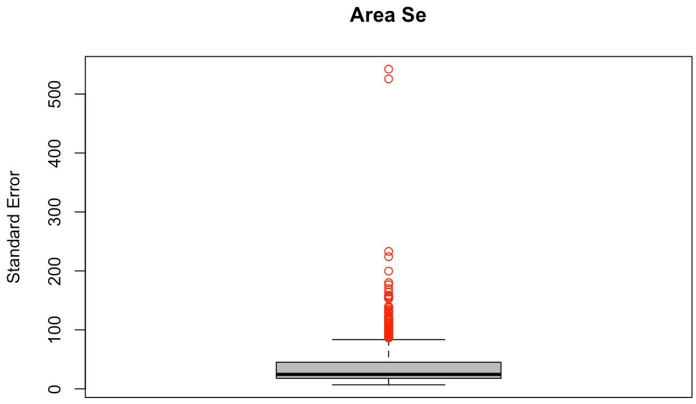

Fractal Dimension Mean
Mean
0.05
0.06
0.07
0.08
0.09

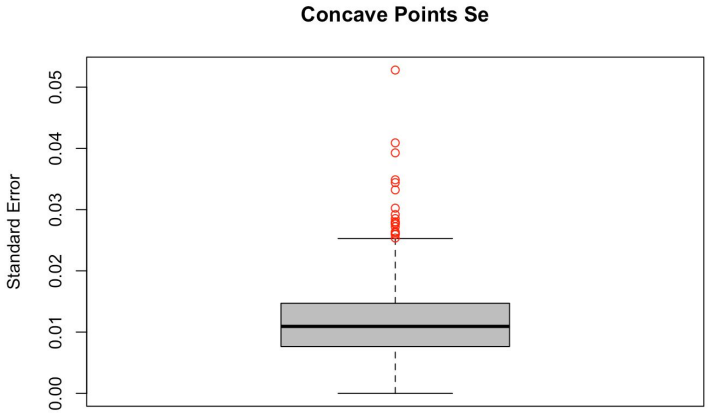

Fractal Dimension Se
Standard Error
.000
0.010
0.020
0.030

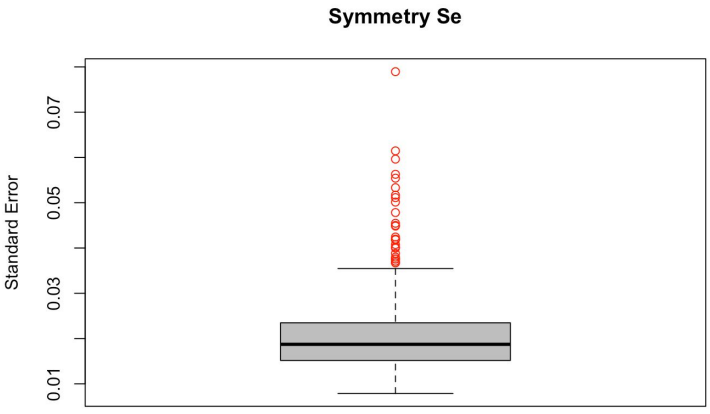

Radius Se
Standard Error
0.5
1.0
1.5
2.0
2.5

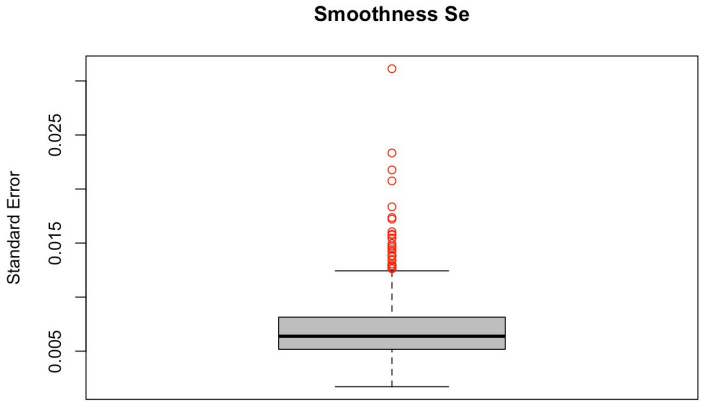

Texture Se
Standard Error
1
2
3
4
5

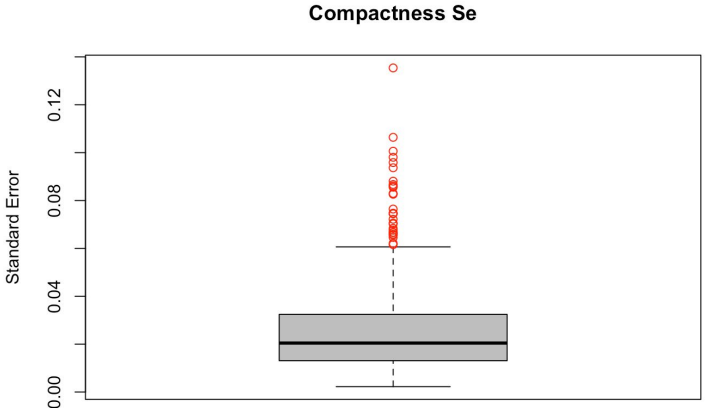

Perimeter Se
Standard Error
0
5
10
15
20

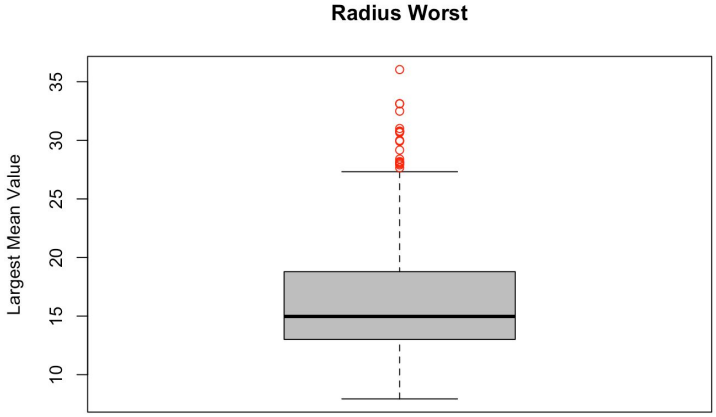

Area Se
Standard Error
0
100
200
300
400
500

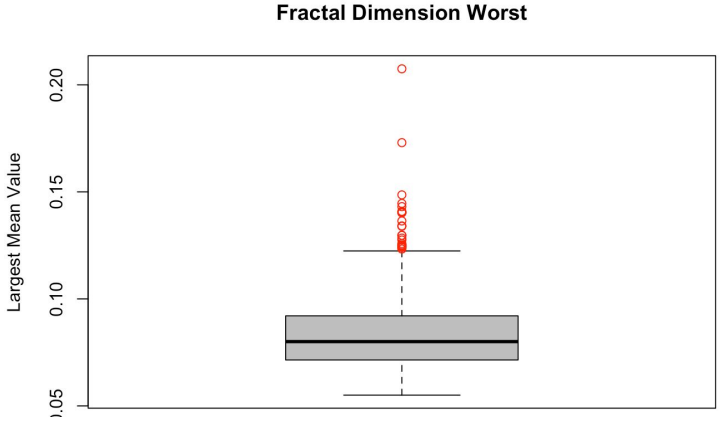

Concave Points Se
Standard Error
0.00
0.01
0.02
0.03
0.04
0.05

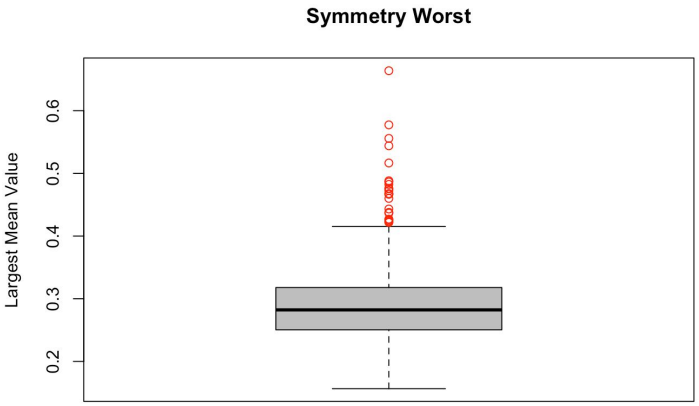

Symmetry Se
Standard Error
0.01
0.03
0.05
0.07

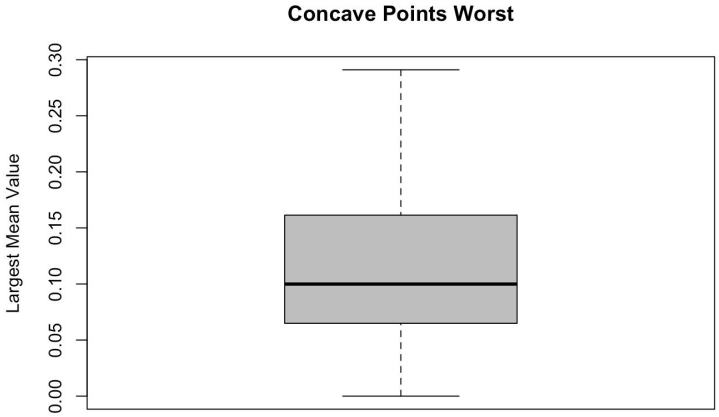

Smoothness Se
Standard Error
0.005
0.015
0.025

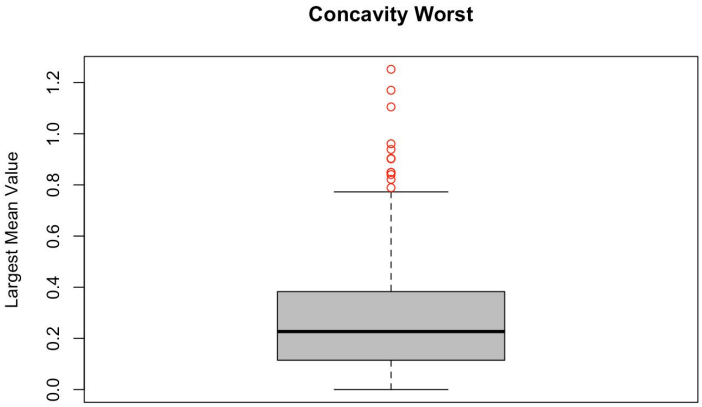

Compactness Se
Standard Error
0.00
0.04
0.08
0.12

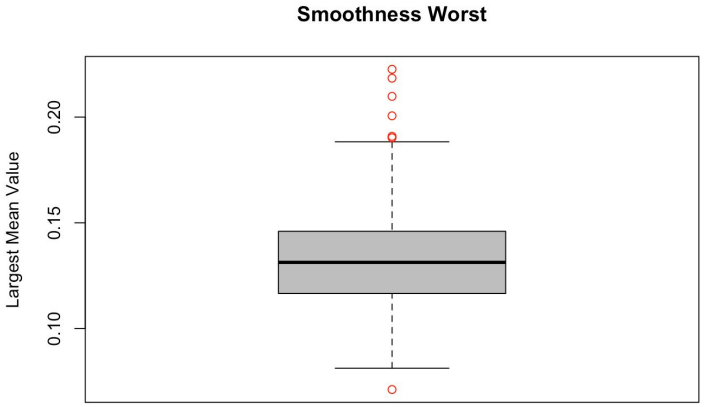

Radius Worst
Largest Mean Value
10
15
20
25
30
35

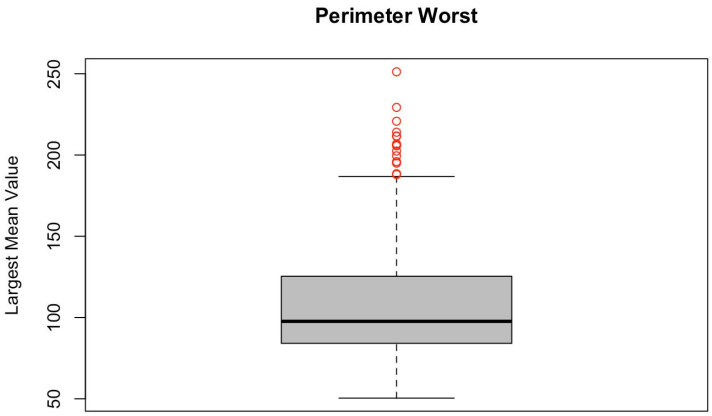

Fractal Dimension Worst
Largest Mean Value
0.10
0.15
0.20

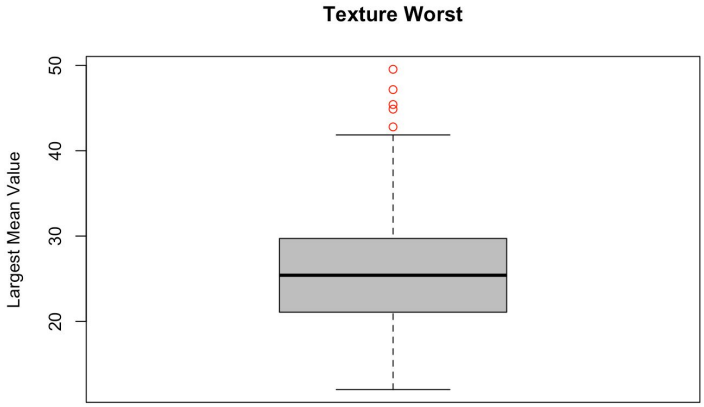


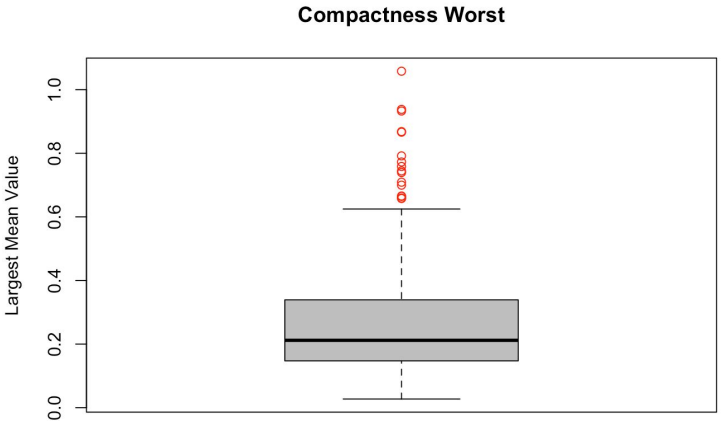












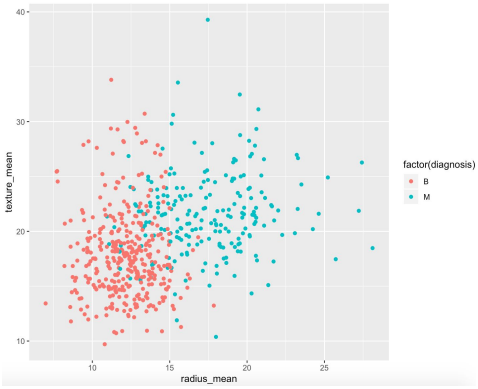
When evaluating the boxplots for each of the variables, it is clear that each has outliers, except for one 'Concave Points Worse.' But, as I learned in class, just dropping them can be a bad idea, since they can contain critical information. These values could be legitimate. So, I will dig deeper and determine what is the percentage of outliers for each variable and see if it makes up a large or small percentage (less than 5% of values) to decide if outliers are a true concern in this data set.

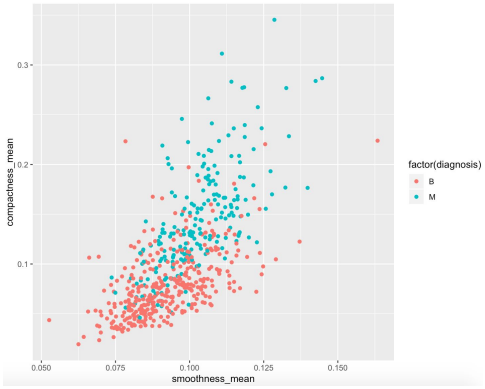
Get a summarized % of the number of outliers in each variable, this will allow me to determine whether outliers are a problem. n=3 df %>% summarise_each(funs(sum(. > mean(.) + n*sd(.) | . < mean(.) - n*sd(.))/n()),

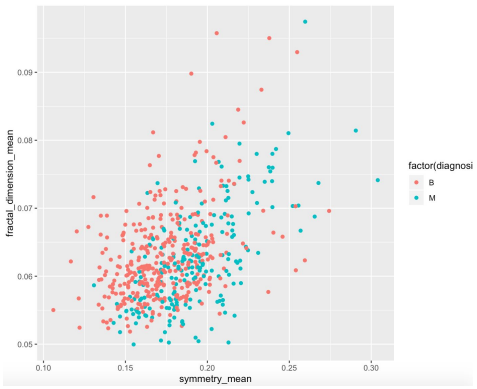
c(3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32))

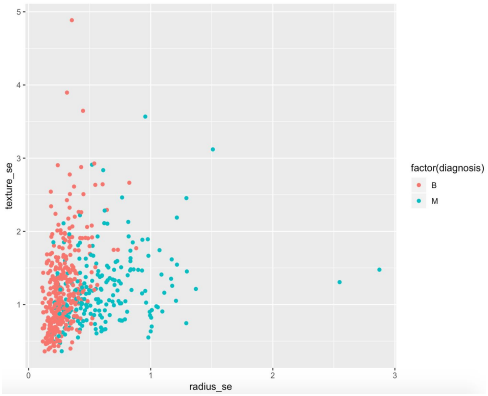
```
> n=3
> df %>% summarise_each(funs(sum(. > mean(.) + n*sd(.) | . < mean(.) - n*sd(.))/n()),
+                       c(3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32))
  radius_mean texture_mean perimeter_mean  area_mean smoothness_mean compactness_mean concavity_mean concave.points_mean
1 0.008787346  0.007029877     0.01230228 0.01405975     0.008787346       0.01581722     0.01581722          0.01054482
  symmetry_mean fractal_dimension_mean  radius_se texture_se perimeter_se    area_se smoothness_se compactness_se concavity_se
1   0.008787346             0.01230228 0.01230228 0.01581722   0.01405975 0.01054482    0.01230228     0.02108963   0.01054482
  concave.points_se symmetry_se fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst smoothness_worst
1        0.01054482  0.01933216           0.01757469   0.01054482   0.007029877      0.01054482 0.01757469      0.005272408
  compactness_worst concavity_worst concave.points_worst symmetry_worst fractal_dimension_worst
1        0.01757469      0.01230228                    0     0.01581722              0.01581722
> 
```

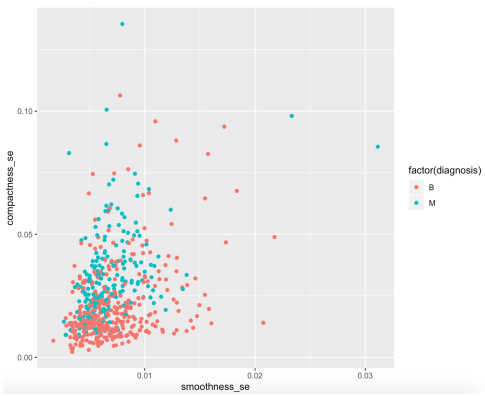
The percentage of outliers for each variable is below 2%, which means the data contains a small percent of outliers - less than 5%; therefore, I will leave them as they could contain meaningful information. If I discover that my model's performance is suffering because of the outliers, this is something I could revisit, but in this case I will not. Additionally, if I refer to the paper "Nuclear feature extraction for breast tumor diagnosis," these large values are not unexpected. In fact, the authors state "the features are numerically modeled such that larger values will typically indicate a higher likelihood of malignancy." So handling these outliers may do more harm than good as they appear to contain critical information. So, there is no need to correct using techniques such as binning, substitution, or imputing values. Another great way to evaluate my data is to plot, here are some interesting scatter plots illustrating variable distribution. The dots in red represent benign and the turquoise dots are indicative of malignant. When I look at the first box below, I've plotted radius_mean vs. text_mean. There is some nice separation of the classes. It appears the larger the radius_mean, I see greater separation of the classes as most appear to be malignant points.

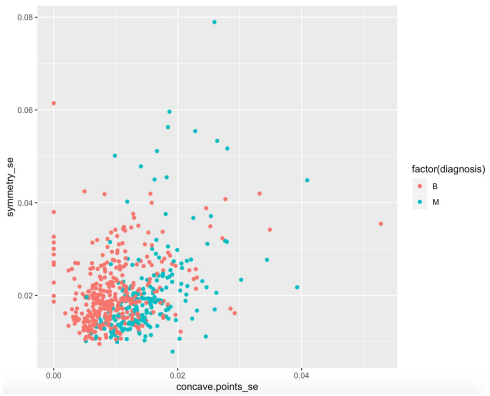


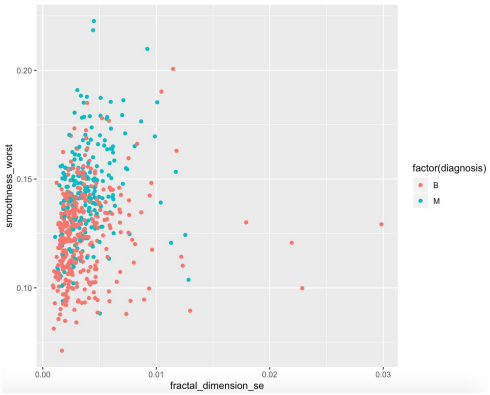


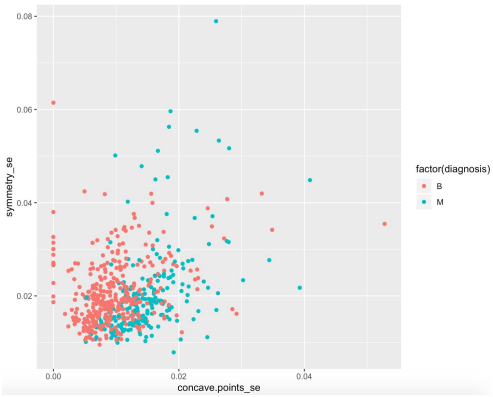


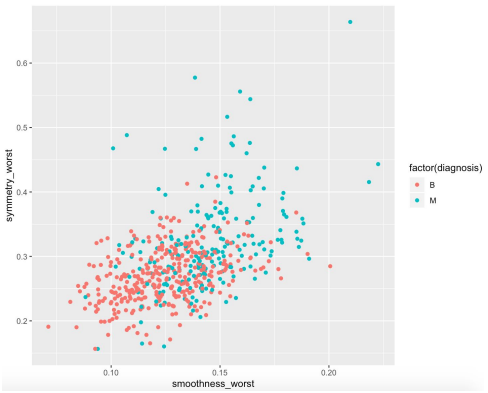




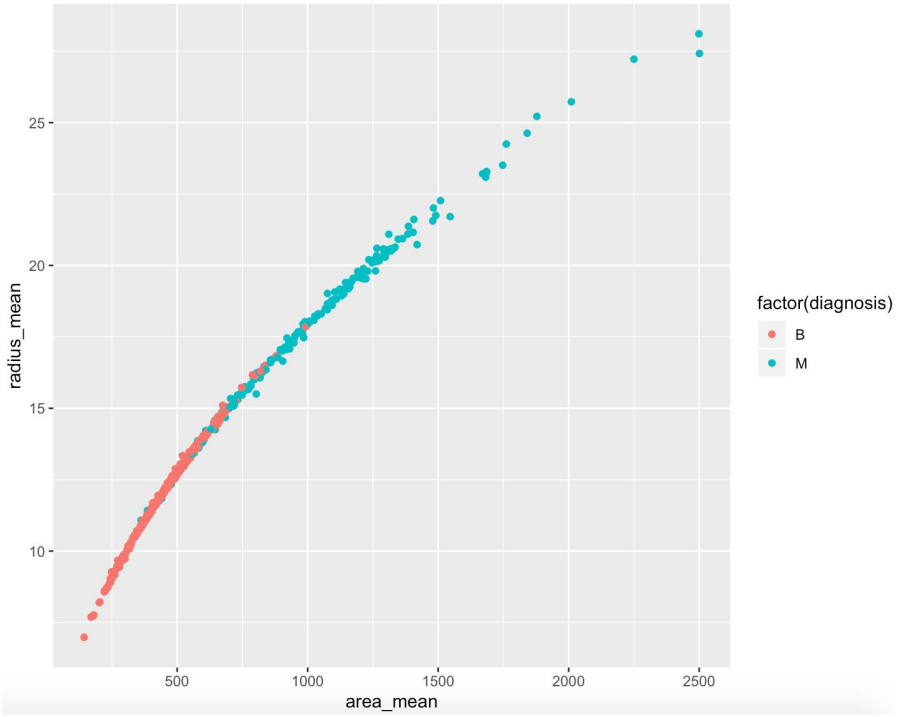








Pre-processing



During evaluation, I discovered there is a column 'X' that is irrelevant and has a bunch of values marked NA's I have dropped that column. Additionally, I have dropped id, which represents the patient id as it will not add any value in making a prediction. I have decided to keep the outliers that I discovered since the percentage of them is so small below 2% and appear to contain critical information.

#Drop rows that aren't needed: id & X
newdf <- subset(df, select = -c(`id`, `X`))

This scatter plot was cool showing the relationship between radius mean and area_mean. It's almost a perfect linear line. Once you hit a radius_mean above 15, most points are classified as malignant. As we get to the decision tree, you will see how the models use this attribute and its cut off point at 15 to make predictions.

Next, I wonder, are there any redundant or similar variables we can drop, since there are 30. The curse of dimensionality (too many variables, redundant rows) refers to the amount of data we need to accurately interpret and how it grows exponentially, which can confuse machine learning algorithms and may become computationally expensive limiting performance. To explore if there are any similar attributes I can discard, I will first create a correlation matrix and then plot the results using a heat map. To create the matrix, I will use the package "corrplot."

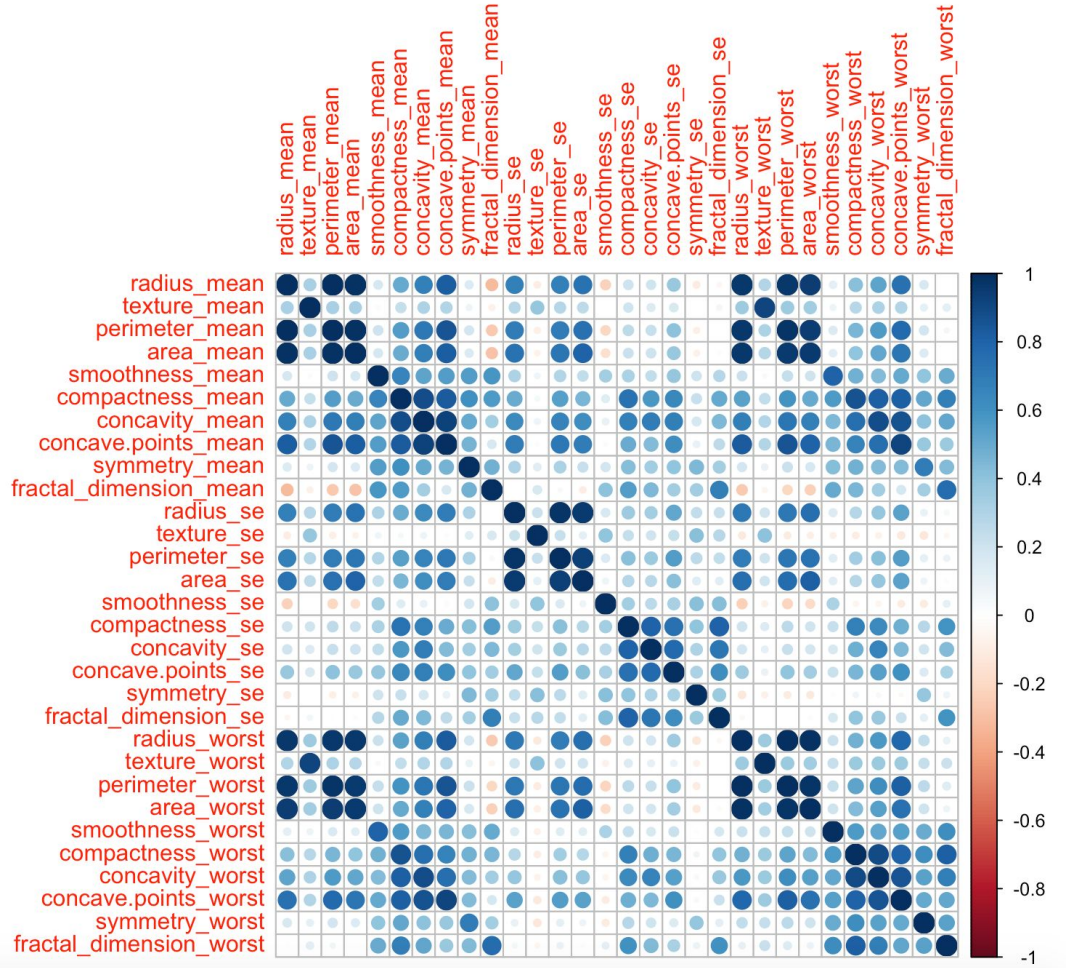
#Create a correlation matrix
cor_matrix <- cor(newdf)

This is a sample of the correlation matrix using the Pearson correlation algorithm. Values closer to 1 represent a strong relationship. Those values that are negative represent an inverse relation. And, values of 0 represents no relation.

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave.points_mean	syn
radius_mean	1.000000000	0.323781891	0.997855281	0.987357170	0.17058119	0.50612358	0.67676355	0.82252852	0.1
texture_mean	0.323781891	1.000000000	0.329533059	0.321085696	-0.02338852	0.23670222	0.30241783	0.29346405	0.0
perimeter_mean	0.997855281	0.329533059	1.000000000	0.986506804	0.20727816	0.55693621	0.71613565	0.85097704	0.1
area_mean	0.987357170	0.321085696	0.986506804	1.000000000	0.17702838	0.49850168	0.68598283	0.82326887	0.1
smoothness_mean	0.170581187	-0.023388516	0.207278164	0.177028377	1.00000000	0.65912322	0.52198377	0.55369517	0.5
compactness_mean	0.506123578	0.236702222	0.556936211	0.498501682	0.65912322	1.00000000	0.88312067	0.83113504	0.6
concavity_mean	0.676763550	0.302417828	0.716135650	0.685982829	0.52198377	0.88312067	1.00000000	0.92139103	0.5
concave.points_mean	0.822528522	0.293464051	0.850977041	0.823268869	0.55369517	0.83113504	0.92139103	1.00000000	0.4
symmetry_mean	0.147741242	0.071400980	0.183027212	0.151293079	0.55777479	0.60264105	0.50066662	0.46249739	1.0
fractal_dimension_mean	-0.311630826	-0.076437183	-0.261476908	-0.283109812	0.58479200	0.56536866	0.33678336	0.16691738	0.4
radius_se	0.679090388	0.275868676	0.691765014	0.732562227	0.30146710	0.49747345	0.63192482	0.69804983	0.3
texture_se	-0.097317443	0.386357623	-0.086761078	-0.066280214	0.06840645	0.04620483	0.07621835	0.02147958	0.1
perimeter_se	0.674171616	0.281673115	0.693134890	0.726628328	0.29609193	0.54890526	0.66039079	0.71064987	0.3
area_se	0.735863663	0.259844987	0.744982694	0.800085921	0.24655243	0.45565285	0.61742681	0.69029854	0.2
smoothness_se	-0.222600125	0.006613777	-0.202694026	-0.166776667	0.33237544	0.13529927	0.09856375	0.02765331	0.1
compactness_se	0.205999980	0.191974611	0.250743681	0.212582551	0.31894330	0.73872179	0.67027882	0.49042425	0.4
concavity_se	0.194203623	0.143293077	0.228082345	0.207660060	0.24839568	0.57051687	0.69127021	0.43916707	0.3
concave.points_se	0.376168956	0.163851025	0.407216916	0.372320282	0.38067569	0.64226185	0.68325992	0.61563413	0.3
symmetry_se	-0.104320881	0.009127168	-0.081629327	-0.072496588	0.20077438	0.22997659	0.17800921	0.09535079	0.4
fractal_dimension_se	-0.042641269	0.054457520	-0.005523391	-0.019886963	0.28360670	0.50731813	0.44930075	0.25758375	0.3

I will definitely need a heat map to make sense of the information in the table. I can generate one using the following:

corrplot(cor_matrix)



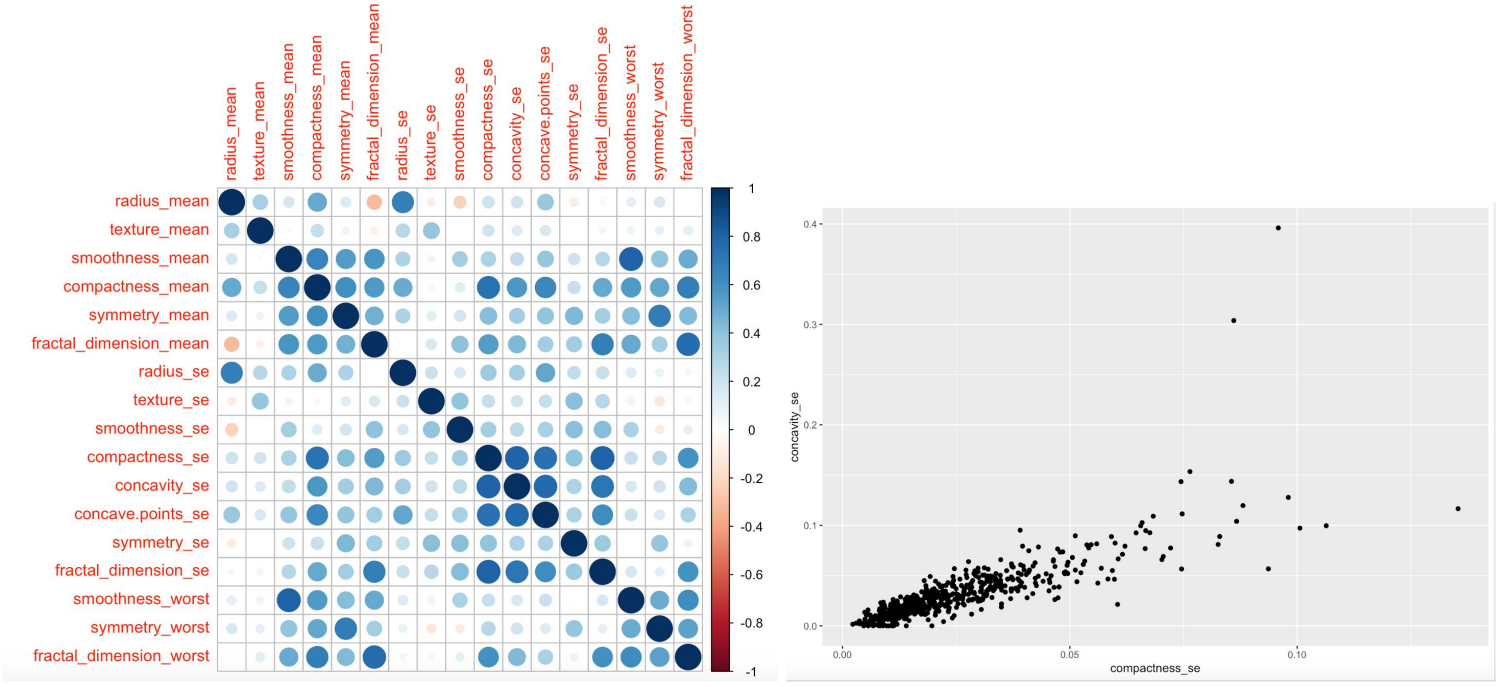
Aside from the pretty colors, which makes this map amazing, I am impressed by the power of being able to see the correlation of the variables. Based on this, the following variables are similar to each other.

Radius_mean is strongly related to area_mean and perimeter_mean, radius_worst, perimeter_worst, and area_worst. Perimeter_mean is strongly related to radius_mean, area_mean, radius_worst, area_worst, and perimeter_worst. Area mean is strong related to radius_mean and perimeter_mean, radius_perimeter, and area_worst. concavity_mean is strongly related to concave.points_mean, compactness_mean, compactness_worst, concavity worst, and concave.points_worst.

As a result, I can reduce redundant variables and am left with the following variables for prediction: radius_mean, texture_mean, smoothness mean, compactness_mean, symmetry_mean, fractal_dimension_mean, radius_se, texture_se, smoothness_se, compactness_se, concavity_se, concavity.points_se, symmetry_se, fractal_dimension_se, smoothness worst, symmetry_worst, fractal_dimension_worst. So, I was able to eliminate 15 variables (from 30 down to 17). This will not only help my machine learning algorthim's performance, but should reduce the computational expense.

```
reducedf <- subset(newdf, select = c(1,2,5,6,9,10,11,12,15,16,17,18,19,20,25,29,30))
```

Check out the updated heat map below.



Above, this is a new heat map of the reduced data frame. I just found this interesting to point out, for example, when looking at the correlation between concavity_se and compactness_se, they have a correlation score of 0.8012683, which is almost 1 (strongly related). I just wanted to check and see if a scatter plot of both variables would show the strong correlation. And, it does, you can see all records for the most part appear in one cluster, which is pretty neat.

Modeling

Now, that I have removed missing, irrelevant variables, and redundant variables, I am ready to begin the modeling step. Since we have a right or wrong variable 'Diagnosis, I will build a (supervised learning) classifier to

predict whether a patient has or does not have breast cancer. We explored three different classifiers in class: Decision Tree, KNN, and Naive-Bayes. I will start with decision tree.

Model 1: Decision Tree

First, I will begin setting a seed (refer to R script for all code), once that is complete, I create my training split which is usually between 66-80% and the rest will be used for training. Since my data does not appear to be imbalanced, I will not worry about methods to correct this such as over/under sampling, etc. I will set the p value in the createDataPartition function to .70, which means I want 70% of the data to be used in the training set (so 399 records to train my model) and the remainder of the set (170 records) to test.

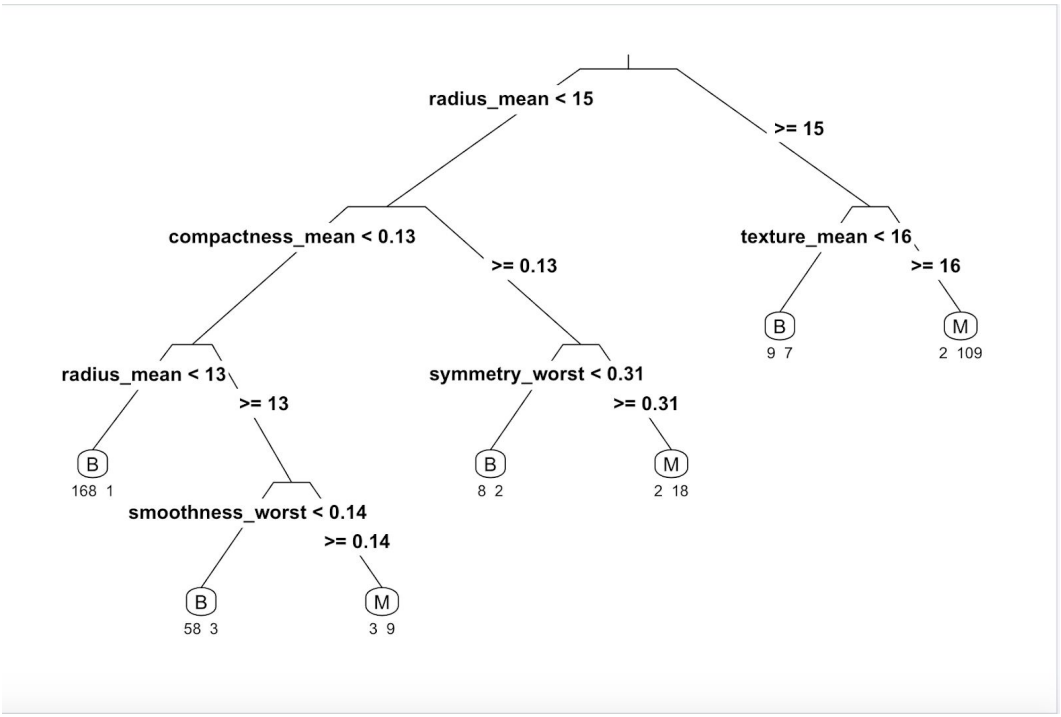
```
inTrain <- createDataPartition(y = reducedf$diagnosis, p = 0.7, list = FALSE)
training <- reducedf[ inTrain,]
testing <- reducedf[-inTrain,]
```

Next, I will create the decision tree model using the training set. However, before I get into the results, while the decision tree is faster to build and is robust (handles noise, redundant, irrelevant attributes, which I should have already removed), I need to be wary of over/under fitting. Moreover, I must also be mindful that the best tree may not be found since the classifier uses the Greedy algorithm (choosing the most optimal decision at each given step instead of choosing what is best for the overall outcome).

```
tree <- rpart(diagnosis ~ ., data = training)
prp(tree, under=TRUE, type=3, varlen = 0, faclen = 0, extra = TRUE)
```

And, here it is the tree. I expected something larger. And I am only seeing 6 of the 17 variables being used. I remind myself - a decision tree is unstable to small changes and produces a different tree each time. In this case, the tree has 4 levels and a depth of 3 with radius mean at the variable at the root.

The initial tree



I'm curious - what does the model consider to be the most important variables when making a prediction? To help me get a better sense of what is going on I run the following command:

```
> tree$variable.importance
            radius_mean               radius_se        compactness_mean
             113.803336                60.236801                44.896278
 fractal_dimension_mean            texture_mean       concave.points_se
              28.729140                20.266483                19.720366
      smoothness_worst          symmetry_worst fractal_dimension_worst
              17.872280                15.681138                12.490473
        compactness_se           symmetry_mean           smoothness_se
               5.348269                 4.773202                 3.722619
       smoothness_mean              texture_se             symmetry_se
               1.676719                 1.554668                 0.315659
> |
```

It appears radius_mean is the most important variable the model uses for predicting; therefore, it is at the root. As mentioned previously, most patients who have a radius_mean of 15 or greater are found to have cancer. The percentage of variability that this variable accounts for is 113.80%. It is followed by the following variables: radius_se, compactness_mean, fractal_dimension_mean, texture_mean, concave.points_se, smoothness_worst, symmetry_worst, fractal_dimension_worst and so on … to the least used symmetry_se with a percentage of 0.316%. But, what happened to the following attributes: concavity_se, fractal_dimension_se as they are not displayed in the variable important list?. The decision tree picks what it wants to use and this can vary each time a tree is created. So, perhaps, this time it just decided not to use those attributes.

After creating the tree, it was time to test it using the training data. After running a confusion matrix on the training data, the accuracy is 94.9% and sensitivity is 97.2%. Pretty good. But, I hope my tree isn't overfitted. Now, I will make predictions based on the testing set and see how it performs. If it performs about as well, then I can say my tree isn't overfitted. However, if it performs poorly on the testing set, then overfitting is definitely a concern.

```
tree.pred = predict(tree, training, type="class")
confusionMatrix(tree.pred, training$diagnosis)
```

```
> confusionMatrix(tree.pred, training$diagnosis)
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 243  13
         M   7 136

               Accuracy : 0.9499
                 95% CI : (0.9236, 0.9691)
    No Information Rate : 0.6266
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.892

 Mcnemar's Test P-Value : 0.2636

            Sensitivity : 0.9720
            Specificity : 0.9128
         Pos Pred Value : 0.9492
         Neg Pred Value : 0.9510
             Prevalence : 0.6266
         Detection Rate : 0.6090
   Detection Prevalence : 0.6416
      Balanced Accuracy : 0.9424

       'Positive' Class : B

>
```

```
> confusionMatrix(tree.pred, training$diagnosis, mode = "prec_recall")
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 243  13
         M   7 136

               Accuracy : 0.9499
                 95% CI : (0.9236, 0.9691)
    No Information Rate : 0.6266
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.892

 Mcnemar's Test P-Value : 0.2636

              Precision : 0.9492
                 Recall : 0.9720
                     F1 : 0.9605
             Prevalence : 0.6266
         Detection Rate : 0.6090
   Detection Prevalence : 0.6416
      Balanced Accuracy : 0.9424

       'Positive' Class : B

>
```

```
tree.pred = predict(tree, testing, type="class")
confusionMatrix(tree.pred, testing$diagnosis)
```

And, after running the commands above, here are the results of the model's performance using the testing set, as detailed in the confusion matrix below. Wow, 91.76% accuracy along with a 95% confidence interval that accuracy will lie anywhere between 86.57% to 95.42%. Not bad! While it is nice to see the model have an accuracy of 91.76%, I'm particularly interested in the sensitivity score, which is a 95.3%. Since we are detecting breast cancer, it is far more important that the classifier identify all possible instances of cancer (true positives) as it would cost more if our model fails to diagnose a patient who has cancer and is then left untreated. My model's recall score if 95.33%, Kappa 82.11%, and f1 93.58%.

```
> confusionMatrix(tree.pred, testing$diagnosis)
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 102   9
         M   5  54

               Accuracy : 0.9176
                 95% CI : (0.8657, 0.9542)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8211

 Mcnemar's Test P-Value : 0.4227

            Sensitivity : 0.9533
            Specificity : 0.8571
         Pos Pred Value : 0.9189
         Neg Pred Value : 0.9153
             Prevalence : 0.6294
         Detection Rate : 0.6000
   Detection Prevalence : 0.6529
      Balanced Accuracy : 0.9052

       'Positive' Class : B

>
```

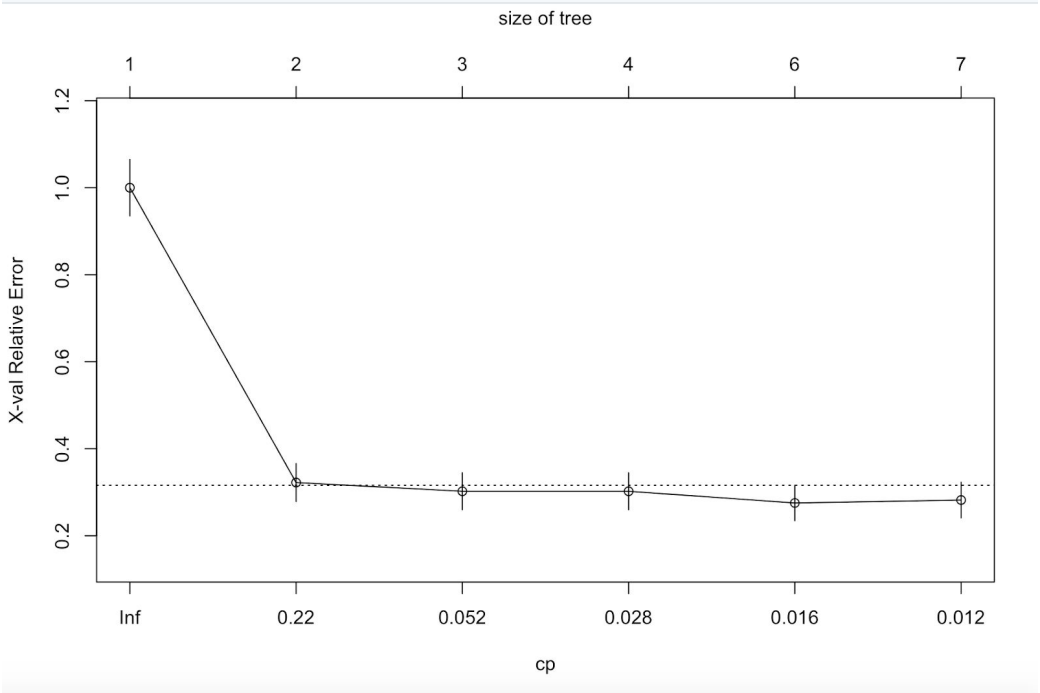
```
> confusionMatrix(tree.pred, testing$diagnosis, mode = "prec_recall")
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 102   9
         M   5  54

               Accuracy : 0.9176
                 95% CI : (0.8657, 0.9542)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8211

 Mcnemar's Test P-Value : 0.4227

              Precision : 0.9189
                 Recall : 0.9533
                     F1 : 0.9358
             Prevalence : 0.6294
         Detection Rate : 0.6000
   Detection Prevalence : 0.6529
      Balanced Accuracy : 0.9052

       'Positive' Class : B

>
```

Overall, the decision tree model did well, but could we do better? I will now display the complexity parameters of the tree which factors in how many splits/branches (the size of the tree) and how it relates to relative error.

CP table:

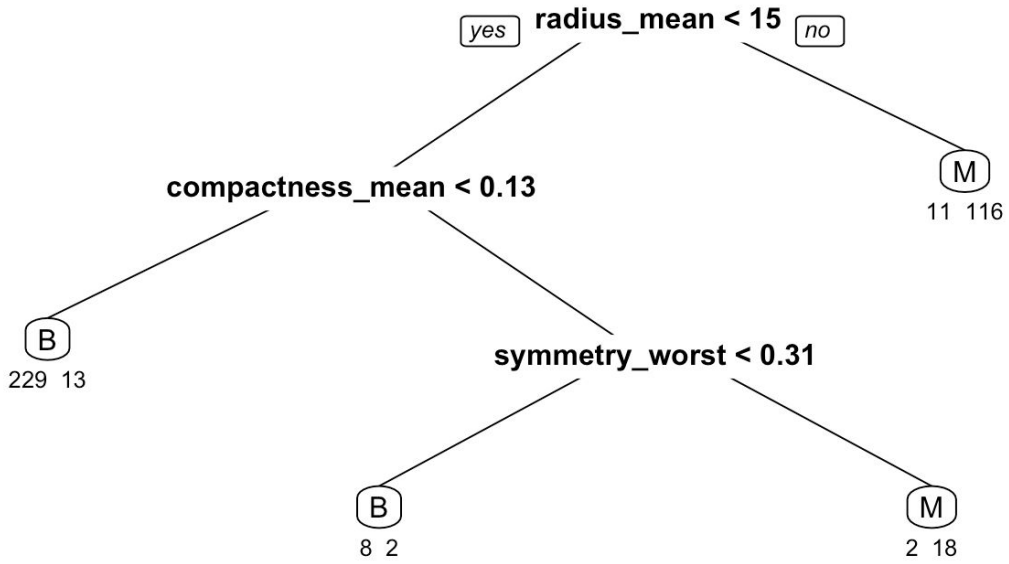
```
Variables actually used in tree construction:
[1] compactness_mean radius_mean      smoothness_worst symmetry_worst
[5] texture_mean

Root node error: 149/399 = 0.37343

n= 399

        CP nsplit rel error  xerror     xstd
1 0.704698      0   1.00000 1.00000 0.064847
2 0.067114      1   0.29530 0.32215 0.043612
3 0.040268      2   0.22819 0.30201 0.042407
4 0.020134      3   0.18792 0.30201 0.042407
5 0.013423      5   0.14765 0.27517 0.040706
6 0.010000      6   0.13423 0.28188 0.041142
```

The graph below shows the size of tree vs. the complexity parameter, which is used to decide which is the most optimal complexity parameter to prune the tree. Look for the leftmost point underneath the line and number of split = (size of tree -1). This is the optimal parameter that will be used for pruning. In this case, the leftmost point under the line is 3, so nsplit= 2, which the optimal complexity parameter = 0.0402268. Pruning the tree we are making it more general, and therefore the accuracy is expected to decrease. But, this is ok as it useful so that the tree can perform on a wider variety of data and settings it hasn't seen before. Moreover, it prevents the tree from overfitting by choosing a simpler tree with smaller X-val relative error and one which is not overly specific and fitted to the training data.



Using the cp value of 0.040, I obtained the following pruned tree. This tree now has 3 levels, one less than the original , and a depth of 2, radius_mean is still at the root. The pruned tree shows the following attributes as

parent nodes used for splitting: radius mean, compactness mean, and symmetry worst. The model chooses the attributes to split which in the end will reduce node impurity.



When looking at the variable importance : pruned$variable.importance, the radius_mean is still being used the most for predicting 111.24%, but slightly less than the original tree model. The most noticeable difference, text_mean is being used more in the new model than concave.points.se. You can see the differences below. Overall , for both models, it is clear the following three variables are used the most in making predictions: radius_mean, radius_se, compactness mean, and fractal_dimension_mean. The attribute with the least importance percentage was symmetry_se with a 0.31%. Interestingly, I don't see radius_se in the tree even though it has an importance percentage of 58.16%

```
> pruned$variable.importance
            radius_mean               radius_se        compactness_mean
             111.242990               58.163911               44.861205
 fractal_dimension_mean       concave.points_se          symmetry_worst
              24.989896               19.720366               13.218669
           texture_mean fractal_dimension_worst        smoothness_worst
              11.974923               10.028004                8.022403
         compactness_se           smoothness_se           symmetry_mean
               5.348269                3.266667                2.613333
```

Original tree:

```
> tree$variable.importance
            radius_mean               radius_se        compactness_mean
             113.803336                60.236801                44.896278
 fractal_dimension_mean            texture_mean       concave.points_se
              28.729140                20.266483                19.720366
      smoothness_worst          symmetry_worst fractal_dimension_worst
              17.872280                15.681138                12.490473
         compactness_se           symmetry_mean           smoothness_se
               5.348269                 4.773202                 3.722619
       smoothness_mean              texture_se             symmetry_se
               1.676719                 1.554668                 0.315659
> |
```

Now, it is time to test the pruned tree.

```
> confusionMatrix(pruned.pred, training$diagnosis)
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 237  15
         M  13 134

               Accuracy : 0.9298
                 95% CI : (0.9002, 0.9529)
    No Information Rate : 0.6266
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8496

 Mcnemar's Test P-Value : 0.8501

            Sensitivity : 0.9480
            Specificity : 0.8993
         Pos Pred Value : 0.9405
         Neg Pred Value : 0.9116
             Prevalence : 0.6266
         Detection Rate : 0.5940
   Detection Prevalence : 0.6316
      Balanced Accuracy : 0.9237

       'Positive' Class : B
```

```
> confusionMatrix(pruned.pred, training$diagnosis, mode = "prec_recall")
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 237  15
         M  13 134

               Accuracy : 0.9298
                 95% CI : (0.9002, 0.9529)
    No Information Rate : 0.6266
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8496

 Mcnemar's Test P-Value : 0.8501

              Precision : 0.9405
                 Recall : 0.9480
                     F1 : 0.9442
             Prevalence : 0.6266
         Detection Rate : 0.5940
   Detection Prevalence : 0.6316
      Balanced Accuracy : 0.9237

       'Positive' Class : B

> |
```

So when I run the training set through the pruned model, I get an accuracy of 92.98% and sensitivity of 94.80% as listed above. Compare that to the result of running my training set through the un-pruned tree - 94.99% accuracy and 97.20% sensitivity. The pruned tree is less accurate than than the original tree.

pruned.pred = predict(pruned, testing, type="class")
confusionMatrix(pruned.pred, testing$diagnosis)

Now, I will try running the testing set through the pruned tree to make predictions. And, as expected, the tree is slightly less accurate than the un-pruned tree, 90.59% and 91.76% respectively. This is because the tree is less complex and less specific (reduced number of branches and splits). Recall also decreased compared to the un-pruned tree, 94.80% and 95.33% respectively. And, Kappa also decreased from 82.11% to 79.14%. Kappa compares the Observed Accuracy with Expected Accuracy. See the confusion matrix below.

```
> confusionMatrix(pruned.pred, testing$diagnosis)
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 104  13
         M   3  50

               Accuracy : 0.9059
                 95% CI : (0.8517, 0.9452)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.7914

 Mcnemar's Test P-Value : 0.02445

            Sensitivity : 0.9720
            Specificity : 0.7937
         Pos Pred Value : 0.8889
         Neg Pred Value : 0.9434
             Prevalence : 0.6294
         Detection Rate : 0.6118
   Detection Prevalence : 0.6882
      Balanced Accuracy : 0.8828

       'Positive' Class : B
```

```
> confusionMatrix(pruned.pred, testing$diagnosis, mode = "prec_recall")
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 104  13
         M   3  50

               Accuracy : 0.9059
                 95% CI : (0.8517, 0.9452)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.7914

 Mcnemar's Test P-Value : 0.02445

              Precision : 0.8889
                 Recall : 0.9720
                     F1 : 0.9286
             Prevalence : 0.6294
         Detection Rate : 0.6118
   Detection Prevalence : 0.6882
      Balanced Accuracy : 0.8828

       'Positive' Class : B

>
```

Sometimes there are trade-offs, one must consider when looking at performance metrics. Because it is important for me to catch all cases of cancer (true positives) even if we guess incorrectly (false positives), this may not always be the case. So it really depends on the domain. A ROC curve helps me identify the tradeoff - what must specificity be to achieve the desired sensitivity. If the ROC curve aligns itself with the left and top corner of the plot, then I can conclude that the classifier does a very good job of separating the two classes B and M.

Generating ROC Curves

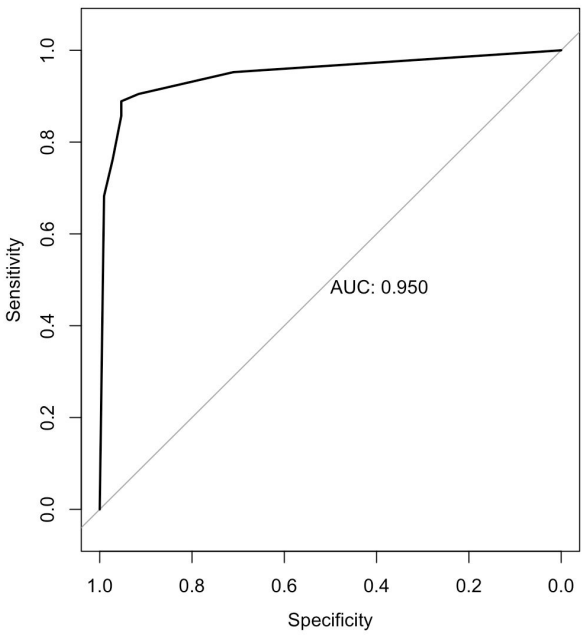
#Generate the probabilities for each row whether M or B for both the original and pruned trees

```
tree_prob = predict(tree, testing, type="prob")
pruned_prob = predict(pruned, testing, type="prob")
```

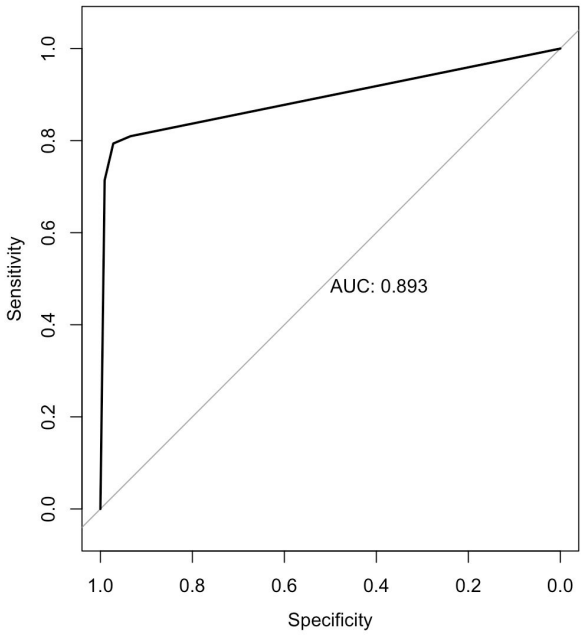
#Generate ROC plots for both trees

```
plot(roc((testing$diagnosis),tree_prob[,1]), print.auc=TRUE)
plot(roc((testing$diagnosis),pruned_prob[,1]), print.auc=TRUE)
```

The ROC plots for both the original tree and the pruned tree are seen on the next page.



Original Tree ROC Curve



Pruned Tree ROC Curve

For the original tree, the highest sensitivity is reached when specificity is at 0. The same goes for the pruned tree. However, for the original tree, we see a higher sensitivity value approximately 0.9 at a specificity value of 1.0 than we do with the pruned tree. At a specificity at 1.0, I see a sensitivity value of 0.8. The AUC for the original tree is 0.950 and for the pruned tree 0.893. Based on AUC (area under the curve) it appears the original tree performed better at spotting true positives and separating the classes compared to the pruned decision tree.

Model 2: KNN

I will now turn to try and create a KNN classifier and see if I can improve on the results obtained from the decision trees. KNN, a distanced based classifier, like a decision tree, is relatively simple and robust to noise since it averages its neighbors. It does not handle missing variables by default like a decision tree. Luckily, we do not have that issue with this data set. The downside, KNN can be quite slow to make predictions especially if we have a large number of variables or attributes that are irrelevant or redundant. Since my variable are all numbers we don't have to create any dummy variables.

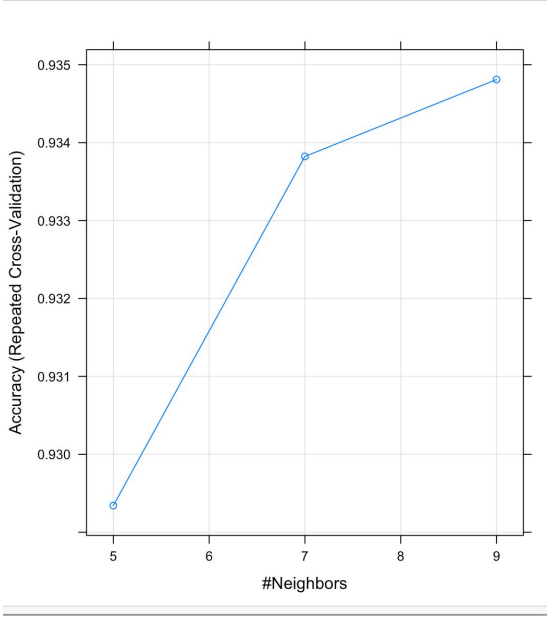
To get started, I will create my training and testing sets. So I can compare both models equally, I will select a p value of 0.7. And, since a KNN classifier is distanced based, I will make sure all values are on the same scale. Scaling will prevent distance measurements from being dominated by one of the variables. I will scale using z-score normalization and center my values.

```
preprocess <- preProcess(KNN.training, method = c("center", "scale"))
train_transformed <- predict(preprocess, KNN.training)
test_transformed <- predict(preprocess, KNN.testing)
```

Next, I will use 5-fold cross-validation sliced into 5 segments and cycle through all cycles to get an averaged overall performance. This technique is used to estimate how well the model generalizes to unseen data especially in cases where there is a limited data sample. A complete model is built each time a different fold is used. The performance score is then averaged. I picked 5-folds arbitrarily, and retested with 3-fold and received the same performance results. The model learned on 399 samples using 17 predictors to classify as either one of two

classes, B or M. The classifier used a final k value of 9. Our accuracy appears to be the highest when there are 9 neighbors as illustrated in the diagrams below. As learned in class, if K is too small, the model will be sensitive to noise points. On the other hand, if K is too large, it may confuse the model as there will be points from other classes voting. As summarized below k= 9, accuracy 93.48%.

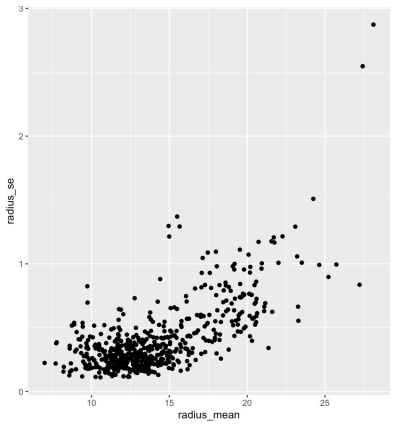
```
k-Nearest Neighbors

399 samples
 17 predictor
  2 classes: 'B', 'M'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 5 times)
Summary of sample sizes: 319, 319, 319, 320, 319, 320, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.9293418  0.8445192
  7  0.9338228  0.8545957
  9  0.9348101  0.8565009

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
>
```



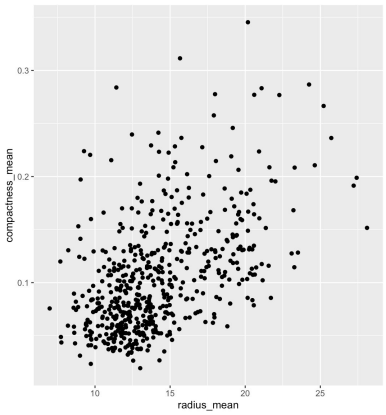
When looking at the most important variables, KNN just like the decision tree, which is interesting, relied heavily on radius_mean (100% importance), followed by radius_se (88.46% importance), and compactness_mean (83.46% importance) for predictions. The variable with the least importance is smoothness_se.

```
                       Importance
radius_mean               100.0000
radius_se                  88.4624
compactness_mean           83.4619
concave.points_se          68.5455
concavity_se               66.7979
texture_mean               65.4690
smoothness_worst           54.9957
symmetry_worst             54.9895
compactness_se             52.7191
smoothness_mean            49.6017
symmetry_mean              44.9255
fractal_dimension_worst    39.4433
fractal_dimension_se       28.2394
texture_se                  9.7711
symmetry_se                 3.1961
fractal_dimension_mean      0.5668
smoothness_se               0.0000
>
```

radius_se vs radius_mean



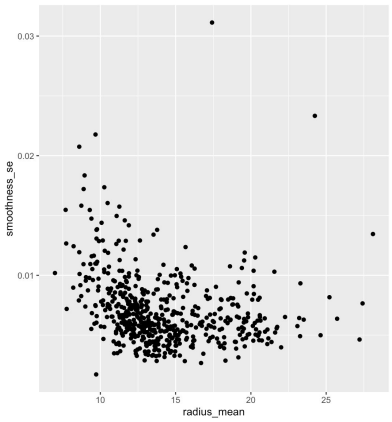
radius_mean vs. compactness mean



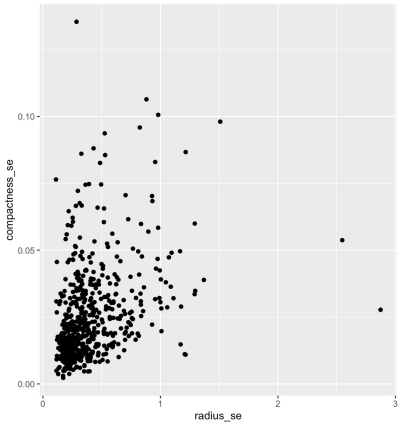
radius mean vs

radius_se vs

smoothness_se



compactness_se



Now, it is time to test my model using the training set. The process went surprisingly fast. Then again, there were only 170 records to test. Wow, the results are pretty impressive and better than the decision tree. The KNN has an accuracy measurement of 93.53% compared to 90.5% for the pruned decision tree, and 91.7% for the un-pruned decision tree. This definitely is a significant improvement. When it came to recall, though, our KNN model had a recall reading of 96.26% compared to 95.33% for the unpruned tree and 97.20% for the pruned tree. In this case, the pruned decision tree model did better than KNN, but not by much. The KNN model had a precision of 93.6% while the unpruned tree has a precision of 88.91% and the pruned tree 91.89. So while, the pruned decision tree did fairly better with sensitivity, the KNN model performed better when it comes to precision and accuracy.

```
> confusionMatrix(pred, KNN.testing$diagnosis)
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 103   7
         M   4  56

               Accuracy : 0.9353
                 95% CI : (0.8872, 0.9673)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8599

 Mcnemar's Test P-Value : 0.5465

            Sensitivity : 0.9626
            Specificity : 0.8889
         Pos Pred Value : 0.9364
         Neg Pred Value : 0.9333
             Prevalence : 0.6294
         Detection Rate : 0.6059
   Detection Prevalence : 0.6471
      Balanced Accuracy : 0.9258

       'Positive' Class : B
```

```
> #Get precision/recall information via confusion matrix, set mode
> confusionMatrix(pred, KNN.testing$diagnosis, mode = "prec_recall")
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 103   7
         M   4  56

               Accuracy : 0.9353
                 95% CI : (0.8872, 0.9673)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8599

 Mcnemar's Test P-Value : 0.5465

              Precision : 0.9364
                 Recall : 0.9626
                     F1 : 0.9493
             Prevalence : 0.6294
         Detection Rate : 0.6059
   Detection Prevalence : 0.6471
      Balanced Accuracy : 0.9258

       'Positive' Class : B

>
```

Next, I will create and produce a ROC plot to see the trade-off between sensitivity and specificity and how well the classifier does at separating the two classes as (area under the curve).

I create a fit control with class probability set to true.

```
fitControl_prob <- trainControl(method = "repeatedcv", number = 5, repeats = 5,  classProbs = TRUE, summaryFunction = twoClassSummary)
```

I will then create the KNN model using probabilities.

```
knn_prob <- train(diagnosis ~ ., data = train_transformed, method = "knn",  trControl = fitControl_prob, metric='ROC')
```

Next, I predict the probabilities of the testing data

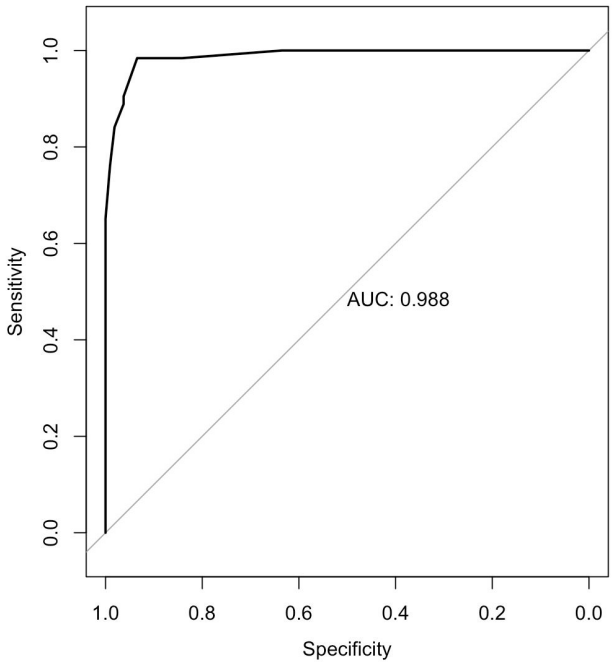
```
pred_prob <- predict(knn_prob, newdata = test_transformed, type = "prob")
```

And, finally, create an ROC curve for the KNN model.

```
plot(roc((KNN.testing$diagnosis),pred_prob[,1]), print.auc=TRUE)
```

The ROC curve for KNN has an AUC of 0.988 which is fantastic. This result is higher than what observed for the decision trees, which means the KNN does a way better job at distinguishing between the two diagnostic groups (malignant/benign). The AUC for the original tree is 0.950 and for the pruned tree 0.893. As for the trade-off between sensitivity, for the most part, sensitivity stays constant near 1.0 once specificity drops below 0.9. As a result, the KNN model seems to perform better at detecting cancer.

```
> knn_prob
k-Nearest Neighbors

399 samples
 17 predictor
  2 classes: 'B', 'M'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 5 times)
Summary of sample sizes: 319, 320, 319, 319, 319, 319, ...
Resampling results across tuning parameters:

  k  ROC        Sens    Spec
  5  0.9751549  0.9808  0.8441379
  7  0.9776556  0.9840  0.8388046
  9  0.9798634  0.9808  0.8414253

ROC was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
>
```



Model 3: Naive Bayes

Finally, the last model that I explored with the data set is Naive Bayes, which is a classifier that relies on probability of different scenarios occurring. It makes the naive assumption that the probabilities of each variable are independent. Compared to the tree models, this one may be more computationally expensive as the model needs to calculate different probabilities for the combinations of variables. Unlike KNN, the Naive Bayes classifier does not need dummy variables. The model does not work well with modeling dependencies or with variables with low variance. Therefore, the first step is to remove any near zero variance records.

```
#Getting rid of near zero variance in data frame
nearzero_variance <- nearZeroVar(reducedf, freqCut = 98/2, names=FALSE)
reducedf_nzv <- reducedf[, -nearzero_variance]
reducedf_nzv
```

Next, I will create by training and test splits. Keeping it consistent with the other models, I will set p to 0.7 and invoke the createDataPartition method: NB.inTrain <- createDataPartition(y = reducedf$diagnosis, p = 0.7, list = FALSE). I will then select my training and testing splits: NB.training <- reducedf[NB.inTrain,] NB.testing <- reducedf[-NB.inTrain,]

I have arbitrarily set my cross-validation for resampling to cross-validated 3-fold, repeated 3 times. The model selected a fL value was held constant at 0. The tuning parameter 'adjust' was held constant at 1.

```
Naive Bayes

399 samples
 17 predictor
  2 classes: 'B', 'M'

No pre-processing
Resampling: Cross-Validated (3 fold, repeated 3 times)
Summary of sample sizes: 266, 267, 265, 266, 266, 266, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.9005884  0.7848392
   TRUE      0.8964303  0.7805394

Tuning parameter 'fL' was held constant at a value of 0
Tuning parameter 'adjust' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = FALSE and adjust = 1.
>
```

Examining the variable importance summary could be a way to determine which low-important variables can be dropped manually although this method may not be efficient. When looking at the variables the Naive Bayes model considers most important when making predictions: like the decision tree and KNN models, radius_mean is most important (100.00), compactness_mean (85.300), and radius_se (84.51). Fractal_dimension_mean was deemed the least important (0.00).

Refer to the full list on the next page.

```
                         Importance
radius_mean                 100.0000
compactness_mean             85.3000
radius_se                    84.5075
concave.points_se            69.2414
concavity_se                 66.8575
smoothness_worst             59.8786
texture_mean                 56.2712
smoothness_mean              56.2178
symmetry_worst               55.1673
compactness_se               52.4185
symmetry_mean                49.3899
fractal_dimension_worst      42.5305
fractal_dimension_se         30.5573
symmetry_se                   6.8468
smoothness_se                 3.1199
texture_se                    0.7863
fractal_dimension_mean        0.0000
> 
```

Next, I will test the model using the training data using the test set. Out of all three models, the Naive Bayes was the least accurate with an accuracy of 89.4, compared to that of KNN 93.53%, and the pruned 90.59% and unpruned 91.76% decision trees. As for recall Naive Bayes also performed the worst 91.59% compared to KNN 96.26%, and the pruned 97.20, and unpruned 95.33% decision trees. Naive Bayes had precision measurement of 91.59% compared to KNN 93.64% and pruned 88.89% and unpruned 91.89%.

```
> confusionMatrix(NB.pred, NB.test_transformed$diagnosis)
Confusion Matrix and Statistics

          Reference
Prediction  B  M
         B 98  9
         M  9 54

               Accuracy : 0.8941
                 95% CI : (0.8378, 0.936)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : 5.13e-15

                  Kappa : 0.773

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9159
            Specificity : 0.8571
         Pos Pred Value : 0.9159
         Neg Pred Value : 0.8571
             Prevalence : 0.6294
         Detection Rate : 0.5765
   Detection Prevalence : 0.6294
      Balanced Accuracy : 0.8865

       'Positive' Class : B

> 
```

```
          Reference
Prediction  B  M
         B 98  9
         M  9 54

               Accuracy : 0.8941
                 95% CI : (0.8378, 0.936)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : 5.13e-15

                  Kappa : 0.773

 Mcnemar's Test P-Value : 1

              Precision : 0.9159
                 Recall : 0.9159
                     F1 : 0.9159
             Prevalence : 0.6294
         Detection Rate : 0.5765
   Detection Prevalence : 0.6294
      Balanced Accuracy : 0.8865

       'Positive' Class : B

> 
```

Model Evaluation

Here is a final summary of the performance metrics for the three different models tested. The KNN classifier overall appears to have performed the best. Although the pruned decision tree had the highest recall. When looking at precision, the KNN performed the best. The highest accuracy out of all three models used was 93.5%, followed by the highest recall 97.2%, and the highest precision 93.64% While it's not 100% it is definitely a great start.

Model	Data Set	Accuracy	Recall	Precision
Decision Tree - UP	**Testing**	**0.9176**	**0.9533**	**0.9189**
Decision Tree - P	**Testing**	**0.9059**	**0.9720**	**0.8889**
KNN	**Testing**	**0.9353**	**0.9626**	**0.9364**
Naive Bayes	**Training**	**0.8941**	**0.9159**	**0.9159**

I think there were many factors that contributed to such great results. There was less than 2% of outliers (noise) the models had to deal with. Redundant variables were cut down by 57% from 30 to 17. And, overall the data was relatively clean with no data inconsistencies or implausible values. The one big takeaway from all of this is how preprocessing is so important. The more time I spend pre-processing or cleaning my data to deal with missing values, outliers, inconsistencies, etc., the better the results.

I believe that all three models did a decent job of separating out the classes to make predictions. The time it took to perform each prediction took under 5 seconds, so computationally speaking, it would be feasible to deploy. It is clear that radius_mean played the biggest factor in making predictions, most records with a radius_mean above 15 was, for the most part, classified as malignant.

Discussion

Overall, I cannot believe I built three machine learning models from scratch all in an effort to detect one of the deadliest diseases that affects women and men around the world. This was a fantastic learning experience of tackling the problem from all steps of the CRISP-DM life cycle: Business Understanding, Data Understanding, Preprocessing, Modeling, Evaluation, and Deployment (the final report).

There were definitely moments when I asked myself am I doing this right? I'm not sure I understand this metric. The pattern I found, is it accurate? But, when I began seeing the results come in, the picture began to become clearer and clearer. And, finally, I could starting putting some of the pieces to the puzzle together. The most challenging part was understanding what the variables represent as my domain knowledge in the medical field is limited. But, with the help of Google, I am thankful to have found the paper online that provides a really detailed explanation as to how the data was collected. I must say the entire process is fascinating.

What surprised me the most was the accuracy of my models. After the evaluation step, I knew that my data was relatively clean. But, I had no idea what that would mean in terms of performance. While I achieved an overall maximum accuracy of 93% and a sensitivity that ranged between 96-97%, I cannot say I feel truly confident yet to rely on my model to detect cancer. There is still much work to do.

So, for my next steps, I would definitely be interested in obtaining another 5,000 more records I could add to further train and test my models. Moreover, I would explore additional ways to improve my models performance by tuning the parameters further, maybe trying different cp values, k, cross-validation variables, or even reducing my variable count even further.

Some of the most important things I learned throughout this process is 1) how to look at my data critically - looking specifically for data inconsistencies, formatting issues, whether or not my data is plausible, missing values. 2)I really enjoyed plotting. I created some pretty cool looking plots and would like to get better at interpreting them. 3) Understanding the importance of preprocessing and the effects it can have on performance 4) Making sense of the metrics given the problem. In my case, I focused on sensitivity and not so much on accuracy. 4) This experience gave me confidence. It is truly a good feeling to say I create three machine learning models from scratch. I look forward to furthering my experience in the field of data science.