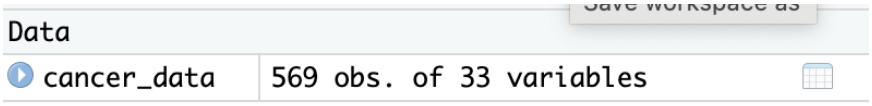
Read CSV file and load packages:

```
1  #install packages and import dataset ####
2  library(cluster)
3  library(factoextra)
4  library(pastecs)
5  library(clustMixType)
6
7  library(tree)
8  library(randomForest)
9  library(pastecs)
10 library(caret)
11
12 cancer_data = read.csv("/Users/lauren/Desktop/DataMining/data.csv")
13
```



We first need to load the necessary packages and the dataset we will be using into R. From lines 2-5, these are the packages we will be using for clustering, and from lines 7-10 they will be used for classification. Packages like “cluster” and “clustMixType” allows us to create clusters and visualize it for our dataset. Another helpful visual for clustering is “factoextra” since it can create a heat map type of plot that displays the similarities and differences between certain parts of the data and can also provide a plot to find the optimal number of clusters. “Pastecs” is used as a descriptive statistical analysis of our data to provide information such as the min, max, range, and variance. For “tree”, it can make decision trees when we train and test our data for predictions while “randomForest” is another type of modeling when we train and test our data. We use “randomForest” to see if our predictions from our training and test sets improve. As for “caret,” it allows us to split our data for training and testing our model for classification.

Investigating the Data:

```
#investigate data ####
head(cancer_data) #view the first few rows of the data
pastecs::stat.desc(cancer_data) #each row provides statistics (all numeric)
names(cancer_data)
summary(cancer_data) #notice X is all NA's, will remove X column and id column in pre-processing
str(cancer_data) #besides ID, diagnosis, and X, every variable is numeric. Change diagnosis to factor (will be our classifier)
```

For investigating our data, we use head(cancer_data) to see the first few rows of our data set. This allows us to see briefly what our data consists of or what it looks like. Then, to get a more statistical summary of our data, pastecs::stat.desc(cancer_data) lets us see some statistics on our data set per variable such as the minimums, maximums, ranges, means, standard errors, and variances. Summary(cancer_data) and str(cancer_data) do provide similar results as pastecs but summary(cancer_data) includes different statistics such as the quartiles and str(cancer_data) lets us know what data types each of the variables are and the first couple values our data has. Names(cancer_data) then provides the list of names of our variables in the data.

```
> head(cancer_data) #view the first few rows of the data
       id diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean compactness_mean
1   842302         M       17.99        10.38         122.80    1001.0         0.11840          0.27760
2   842517         M       20.57        17.77         132.90    1326.0         0.08474          0.07864
3 84300903         M       19.69        21.25         130.00    1203.0         0.10960          0.15990
4 84348301         M       11.42        20.38          77.58     386.1         0.14250          0.28390
5 84358402         M       20.29        14.34         135.10    1297.0         0.10030          0.13280
6   843786         M       12.45        15.70          82.57     477.1         0.12780          0.17000
  concavity_mean concave.points_mean symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
1         0.3001             0.14710        0.2419                0.07871    1.0950     0.9053        8.589
2         0.0869             0.07017        0.1812                0.05667    0.5435     0.7339        3.398
3         0.1974             0.12790        0.2069                0.05999    0.7456     0.7869        4.585
4         0.2414             0.10520        0.2597                0.09744    0.4956     1.1560        3.445
5         0.1980             0.10430        0.1809                0.05883    0.7572     0.7813        5.438
6         0.1578             0.08089        0.2087                0.07613    0.3345     0.8902        2.217
  area_se smoothness_se compactness_se concavity_se concave.points_se symmetry_se fractal_dimension_se
1  153.40      0.006399        0.04904      0.05373           0.01587     0.03003             0.006193
2   74.08      0.005225        0.01308      0.01860           0.01340     0.01389             0.003532
3   94.03      0.006150        0.04006      0.03832           0.02058     0.02250             0.004571
4   27.23      0.009110        0.07458      0.05661           0.01867     0.05963             0.009208
5   94.44      0.011490        0.02461      0.05688           0.01885     0.01756             0.005115
6   27.19      0.007510        0.03345      0.03672           0.01137     0.02165             0.005082
  radius_worst texture_worst perimeter_worst area_worst smoothness_worst compactness_worst concavity_worst
1        25.38         17.33          184.60     2019.0           0.1622            0.6656          0.7119
2        24.99         23.41          158.80     1956.0           0.1238            0.1866          0.2416
3        23.57         25.53          152.50     1709.0           0.1444            0.4245          0.4504
4        14.91         26.50           98.87      567.7           0.2098            0.8663          0.6869
5        22.54         16.67          152.20     1575.0           0.1374            0.2050          0.4000
6        15.47         23.75          103.40      741.6           0.1791            0.5249          0.5355
  concave.points_worst symmetry_worst fractal_dimension_worst  X
1               0.2654         0.4601                 0.11890 NA
2               0.1860         0.2750                 0.08902 NA
3               0.2430         0.3613                 0.08758 NA
4               0.2575         0.6638                 0.17300 NA
5               0.1625         0.2364                 0.07678 NA
6               0.1741         0.3985                 0.12440 NA
```

As mentioned before, we would like to view the first few records of the data to see what we are working with. Notice that the data set has X (the last column) and consists of NA values for the first few rows of data. We must further investigate if X is a valid column for clustering and classification, meaning that if X does have non-NA columns and provides numerical or categorical data, we could include it if there are not too many NA values.

```
> pastecs::stat.desc(cancer_data) #each row provides statistics (all numeric)
                      id diagnosis  radius_mean texture_mean perimeter_mean    area_mean smoothness_mean
nbr.val     5.690000e+02        NA  569.0000000 5.690000e+02   5.690000e+02 5.690000e+02    5.690000e+02
nbr.null    0.000000e+00        NA    0.0000000 0.000000e+00   0.000000e+00 0.000000e+00    0.000000e+00
nbr.na      0.000000e+00        NA    0.0000000 0.000000e+00   0.000000e+00 0.000000e+00    0.000000e+00
min         8.670000e+03        NA    6.9810000 9.710000e+00   4.379000e+01 1.435000e+02    5.263000e-02
max         9.113205e+08        NA   28.1100000 3.928000e+01   1.885000e+02 2.501000e+03    1.634000e-01
range       9.113118e+08        NA   21.1290000 2.957000e+01   1.447100e+02 2.357500e+03    1.107700e-01
sum         1.728157e+10        NA 8038.4290000 1.097581e+04   5.233038e+04 3.726319e+05    5.482900e+01
median      9.060240e+05        NA   13.3700000 1.884000e+01   8.624000e+01 5.511000e+02    9.587000e-02
mean        3.037183e+07        NA   14.1272917 1.928965e+01   9.196903e+01 6.548891e+02    9.636028e-02
SE.mean     5.241136e+06        NA    0.1477358 1.803088e-01   1.018666e+00 1.475301e+01    5.895989e-04
CI.mean.0.95 1.029437e+07       NA    0.2901752 3.541534e-01   2.000813e+00 2.897711e+01    1.158060e-03
var         1.563015e+16        NA   12.4189201 1.849891e+01   5.904405e+02 1.238436e+05    1.977997e-04
std.dev     1.250206e+08        NA    3.5240488 4.301036e+00   2.429898e+01 3.519141e+02    1.406413e-02
coef.var    4.116333e+00        NA    0.2494497 2.229712e-01   2.642083e-01 5.373645e-01    1.459536e-01
            compactness_mean concavity_mean concave.points_mean symmetry_mean fractal_dimension_mean
nbr.val         5.690000e+02   5.690000e+02        5.690000e+02  5.690000e+02           5.690000e+02
nbr.null        0.000000e+00   1.300000e+01        1.300000e+01  0.000000e+00           0.000000e+00
nbr.na          0.000000e+00   0.000000e+00        0.000000e+00  0.000000e+00           0.000000e+00
```

While investigating with the pastecs::stat.desc function, we can see that the columns are numerical except diagnosis. Even ID is considered numerical since it has numerical values for each patient. However, for classification and clustering we do not use ID, so we will exclude it in the pre-processing step.

```
> names(cancer_data)
 [1] "id"                "diagnosis"           "radius_mean"         "texture_mean"
 [5] "perimeter_mean"    "area_mean"           "smoothness_mean"     "compactness_mean"
 [9] "concavity_mean"    "concave.points_mean" "symmetry_mean"       "fractal_dimension_mean"
[13] "radius_se"         "texture_se"          "perimeter_se"        "area_se"
[17] "smoothness_se"     "compactness_se"      "concavity_se"        "concave.points_se"
[21] "symmetry_se"       "fractal_dimension_se" "radius_worst"       "texture_worst"
[25] "perimeter_worst"   "area_worst"          "smoothness_worst"    "compactness_worst"
[29] "concavity_worst"   "concave.points_worst" "symmetry_worst"     "fractal_dimension_worst"
[33] "X"
```

Names(cancer_data) as mentioned above, lets us see what variables the data set gives us. As of now, we have 33 variables and I am interested in viewing what column X consists of since it had some NA values when we used head(cancer_data). In addition to viewing the names of columns, I can see that there are different types of area, radius, or smoothness types such as area_mean vs area_se, and area_worst. In the context of breast cancer analysis, when analyzing a tumor, the analysis of the tumor is split into different sections and different measurements of data is recorded per section of the tumor. Thus, the area means are the means per tumor section, standard error (SE) per tumor section, and worst is the largest mean value among the other sections of the tumor. We want to know what makes a tumor or parts of the tumor benign or malignant from these attributes.

```
> summary(cancer_data) #notice X is all NA's, will remove X column and id column in pre-processing
       id            diagnosis          radius_mean      texture_mean   perimeter_mean     area_mean
 Min.   :     8670   Length:569         Min.   : 6.981   Min.   : 9.71   Min.   : 43.79   Min.   : 143.5
 1st Qu.:   869218   Class :character   1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17   1st Qu.: 420.3
 Median :   906024   Mode  :character   Median :13.370   Median :18.84   Median : 86.24   Median : 551.1
 Mean   : 30371831                      Mean   :14.127   Mean   :19.29   Mean   : 91.97   Mean   : 654.9
 3rd Qu.:  8813129                      3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10   3rd Qu.: 782.7
 Max.   :911320502                      Max.   :28.110   Max.   :39.28   Max.   :188.50   Max.   :2501.0
 smoothness_mean   compactness_mean  concavity_mean    concave.points_mean symmetry_mean
 Min.   :0.05263   Min.   :0.01938   Min.   :0.00000   Min.   :0.00000     Min.   :0.1060
 1st Qu.:0.08637   1st Qu.:0.06492   1st Qu.:0.02956   1st Qu.:0.02031     1st Qu.:0.1619
 Median :0.09587   Median :0.09263   Median :0.06154   Median :0.03350     Median :0.1792
 Mean   :0.09636   Mean   :0.10434   Mean   :0.08880   Mean   :0.04892     Mean   :0.1812
 3rd Qu.:0.10530   3rd Qu.:0.13040   3rd Qu.:0.13070   3rd Qu.:0.07400     3rd Qu.:0.1957
 Max.   :0.16340   Max.   :0.34540   Max.   :0.42680   Max.   :0.20120     Max.   :0.3040
 fractal_dimension_mean   radius_se        texture_se      perimeter_se       area_se         smoothness_se
 Min.   :0.04996        Min.   :0.1115   Min.   :0.3602   Min.   : 0.757   Min.   :  6.802   Min.   :0.001713
 1st Qu.:0.05770        1st Qu.:0.2324   1st Qu.:0.8339   1st Qu.: 1.606   1st Qu.: 17.850   1st Qu.:0.005169
 Median :0.06154        Median :0.3242   Median :1.1080   Median : 2.287   Median : 24.530   Median :0.006380
 Mean   :0.06280        Mean   :0.4052   Mean   :1.2169   Mean   : 2.866   Mean   : 40.337   Mean   :0.007041
```

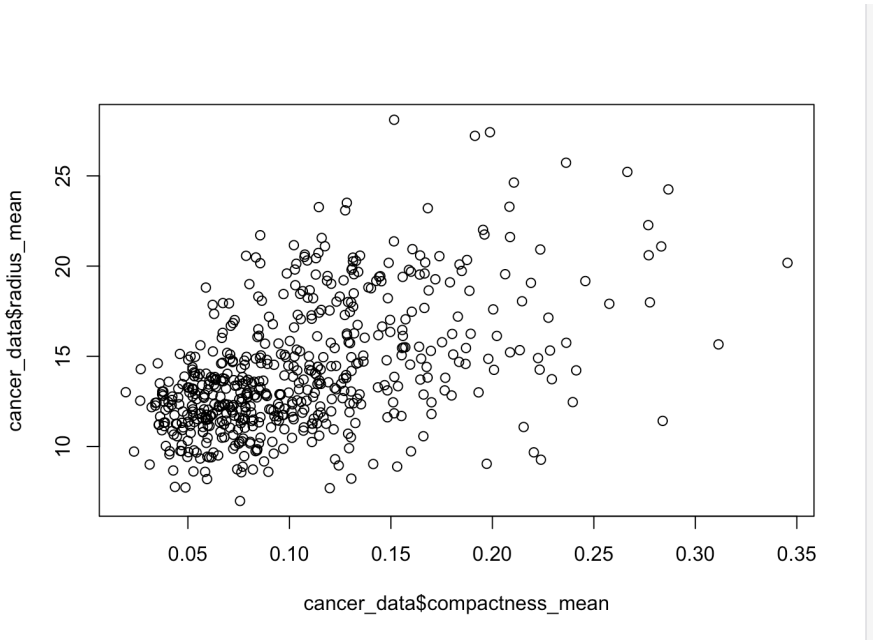
This is a small snippet of summary(cancer_data), when looking through the summary statistics for the data, the X column consisted of NA's for the min, 1st quartile, median, mean, 3rd quartile, and max. This can indicate that X could have all NA values; hence, we may need to remove it in the pre-processing step. As for the rest of the data, notice how the means vary between one another such that the area_mean is 654 while symmetry_mean is 0.18. Because the data varies so much, we need to scale our data in the clustering process.

```
> str(cancer_data) #besides ID, diagnosis, and X, every variable is numeric. Change diagnosis to factor (will be our cl
sifier)
'data.frame':   569 obs. of  33 variables:
 $ id                     : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844981 84501001 ...
 $ diagnosis              : chr  "M" "M" "M" "M" ...
 $ 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 ...
```

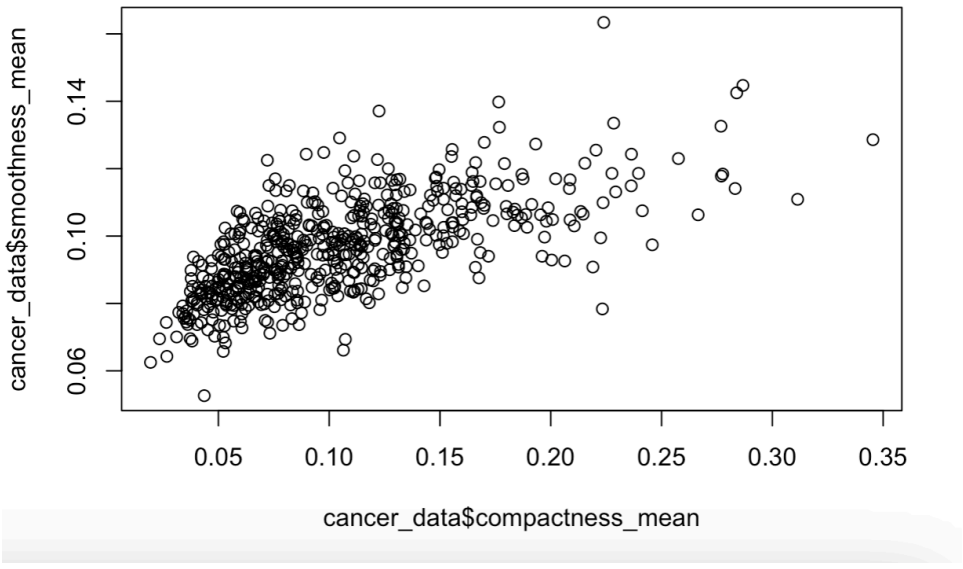
Looking more closely at the data, we notice that X is a logical datatype, diagnosis is character, and ID is integer. Because ID is an integer data type, it was able to calculate means, quartiles, and more. Though as mentioned, we do not need it but it is interesting to see how ID can be considered as a numeric data type. Then, we can see that diagnosis will be our test attribute for classification. Seeing that it is a character data type, we will convert it to a factor data type in the classification process.

```
> lapply(cancer_data[1:33], mean)
$id
[1] 30371831

$diagnosis
[1] NA

$radius_mean
[1] 14.12729

$texture_mean
[1] 19.28965

$perimeter_mean
[1] 91.96903

$area_mean
[1] 654.8891
```

lapply(cancer_data[1:33], mean) lets us see the mean values of the data. Similarly to when we looked at summary(cancer_data), we saw that the means varied by a lot. So, knowing that there is a drastic difference in means from radius_mean and area_mean for example, scaling is needed in clustering our data.



Going more into investigating our data, looking into the data to see if there are any correlations between different variables would be interesting to see. I chose compactness and radius as I believe that the size and the feel of the tumor would heavily influence whether or not a tumor is benign or malignant.



The same idea follows with compactness and smoothness. Normally it is important to know that if a tumor is solid as if it feels like solid (compact), then it is likely to assume that it is malignant or cancerous. This comes with the assumption that if a tumor is solid, it would feel smooth and not lumpy. Thus, seeing if there is a relationship between these two variables was ideal based on my understanding of how cancer is identified in tumors.

```
> cor(cancer_data$compactness_mean, cancer_data$radius_mean) #0.5 (not
[1] 0.5061236
> cor(cancer_data$compactness_mean, cancer_data$smoothness_mean) #0.6
[1] 0.6591232
```

To understand if there were any relationships between the two plots I made with the following variables, calculating the correlation between both of them gave me a better idea. It seemed that in the plots that both did have somewhat of a relationship between each other when plotted against each other. However, while looking at the correlation, the compactness_mean and radius_mean is about 0.51; showing that it is not strongly correlated. As for compactness_mean and smoothness_mean, it is somewhat more strongly correlated with about 0.66 correlation, though still not as strong as I thought it would be.

Pre-Processing the Data:

```
> sum(is.na(cancer_data)) #sum of total missing values,
the X column with 569 missing values
[1] 569
```

After investigating the data, we would like to see if there are any missing or NA values in the data set. Knowing that there are 569 missing values, we must investigate where they exist and get rid of them. In doing so, I noticed in the investigation process that there are 569 observations while doing str(cancer_data) and while looking into the global environment tab. It is possible that all of the NA values are coming from the X column since the X column had some NA values in head(cancer_data) and did not provide any statistical information when doing summary or pastecs.

```r
#removing id and X variables
cancer_data = subset(cancer_data, select = -id)
cancer_data = subset(cancer_data, select = -X)
str(cancer_data) #check to see if they were removed
```

To test to see if X had all NA values, I removed it from the data set and also removed ID since we do not use it for classification and clustering.

```
> #removing id and X variables
> cancer_data = subset(cancer_data, select = -id)
> cancer_data = subset(cancer_data, select = -X)
> str(cancer_data) #check to see if they were removed
'data.frame':   569 obs. of  31 variables:
 $ diagnosis              : chr  "M" "M" "M" "M" ...
 $ 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 ...
```

Using str(cancer_data) I can see now that ID and X have been successfully removed from the data set. We can proceed further by looking at how many null values we must work with.

```
> sum(is.na(cancer_data))
able
[1] 0
```

Now knowing that our data has 0 null values, we can continue working on our data set. This also meant that the X column did have all 569 NA values.

```
> #changing variable diagnosis to factor
> cancer_data$diagnosis = as.factor(cancer_data$diagnosis) #change diagnosis to factor variable
> str(cancer_data) #check to see if diagnosis is changed to factor
'data.frame':   569 obs. of  31 variables:
 $ diagnosis              : Factor w/ 2 levels "B","M": 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 ...
```

As we noticed in the investigation process, diagnosis was a character data type. Since it is nominal, we must change it to factor. Hence we use the as.factor() function.

```
> #diagnosis is categorical so we need to remove it for now to do clustering
> cancer_data_no_diagnosis = subset(cancer_data, select = -diagnosis)
> str(cancer_data_no_diagnosis) #now that we have all numerical values, scale the data
'data.frame':   569 obs. of  30 variables:
 $ 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 ...
```

Now that we changed diagnosis to factor, in clustering, all variables must be numeric. So, for now we will subset the cancer_data into cancer_data_no_diagnosis (excludes diagnosis from the data set) and work with the numerical variables. We will work with the regular cancer_data data set as we transition into classification.

```
#noticed in investigating our data that the means range differently
#need to scale/normalize the data; normalizing the data allows us to make better measurements
d.scale = data.frame(scale(cancer_data_no_diagnosis))
```

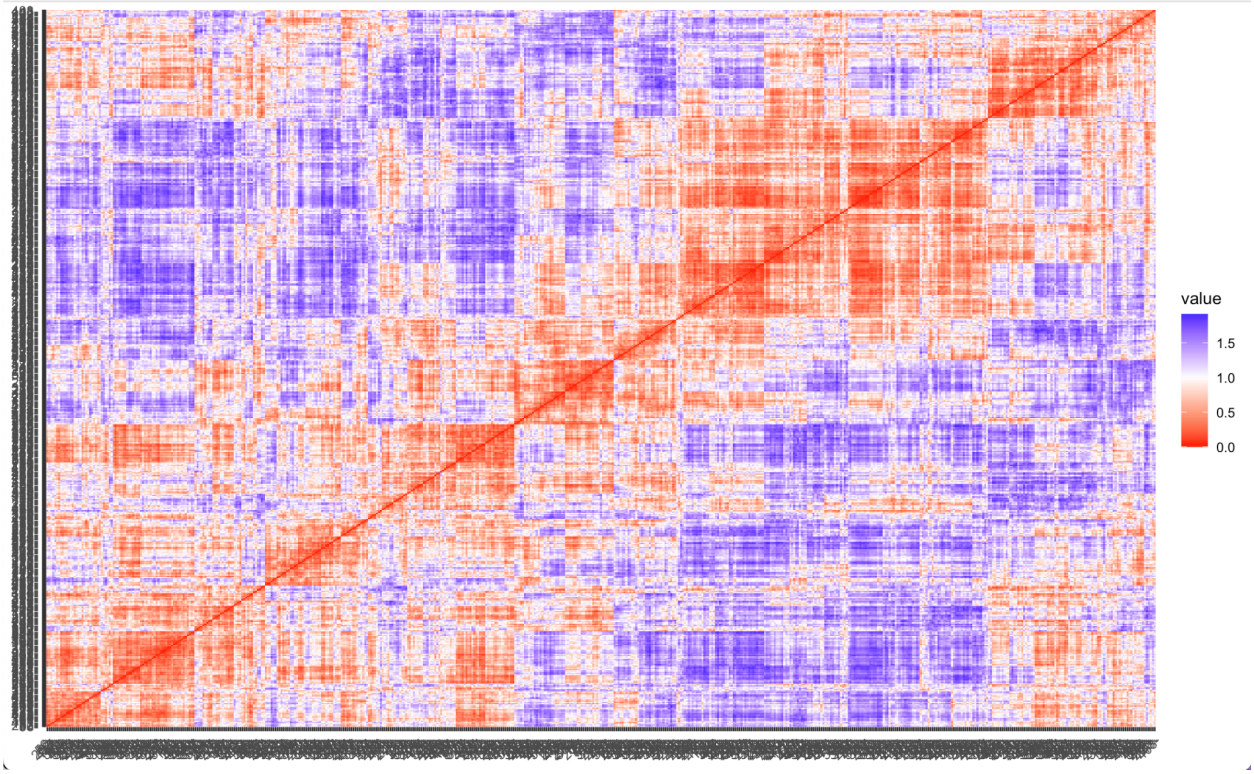
Referring back to the investigation process, when we used lapply(), we saw that the means for each variable varied by a lot. Since they range by a lot, we need to scale the data. Scaling the data normalizes the data to make better measurements.

```
#get the distance matrix between data (getting the distances between the variables)
d.dist = factoextra::get_dist(d.scale, method = 'pearson')
d.dist
```

```
> d.dist
          1          2          3          4          5          6          7          8
2 0.78816931
          9         10         11         12         13         14         15         16
2
         17         18         19         20         21         22         23         24
2
         25         26         27         28         29         30         31         32
2
         33         34         35         36         37         38         39         40
2
```

We now want the distance matrix between the data using the pearson method. Getting the distance allows us to visually see which variables are similar or dissimilar to each other. We can see this with the factoextra() function using d.scale variable which does not include diagnosis.

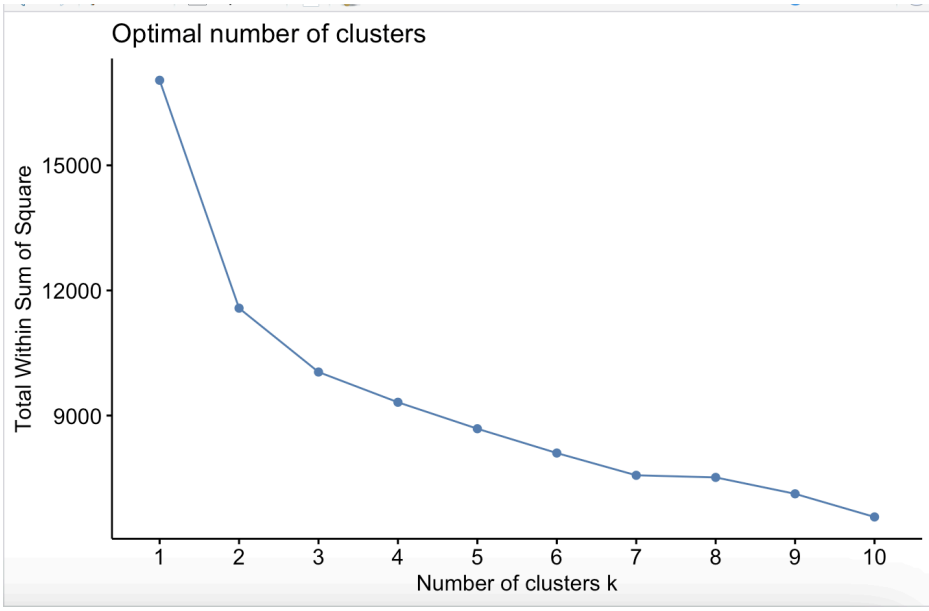
```
#tells us if there are certain
factoextra::fviz_dist(d.dist)
```



Factoextra() function gives a heatmap-like plot that shows whether certain points in the data are similar or dissimilar/naturally cluster together. However, since we have a lot of data, we need to

look more into this with the cluster visualization. But, it is good to see that there is red which indicates that there are certain parts of the data that are similar.

```
#k-means clustering ####

#1. identify or find optimal k clusters
#optimal k is where the elbow point does not significantly
factoextra::fviz_nbclust(d.scale, kmeans, method = 'wss')
```

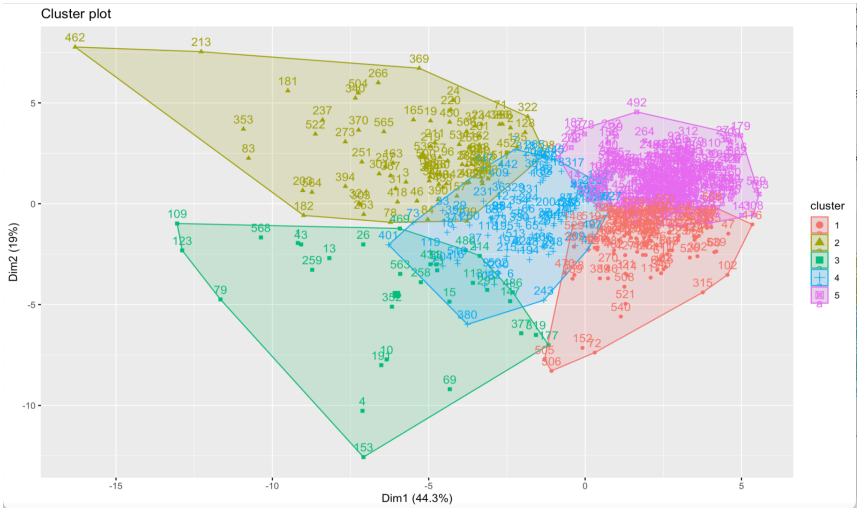


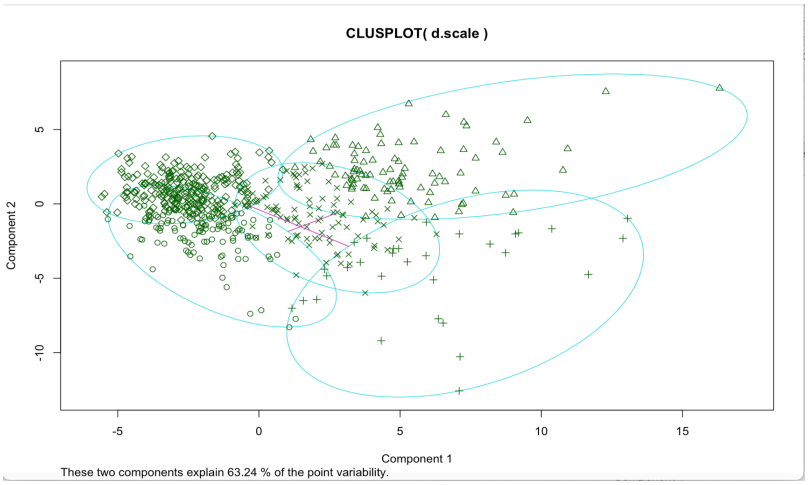
To begin clustering our data, we use the factoextra::fviz_nbclust() function to find the optimal k clusters. By looking at the plot, we see from the "elbow" technique that between 4-6 clusters may be the best to separate the data. We do not want to do more clustering past that as it could lead to overfitting.

```
> k5 = kmeans(d.scale, centers = 5)
> k5$size #gives us size of the 4 clusters
[1] 119  90  31  96 233
> k5$tot.withinss #total wss ; 8709.396
[1] 8709.396
> k5$cluster
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
  3   2   2   3   2   4   4   4   4   3   5   4   3   5   3   4   4   4   2   5   5   1   3   2
 25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48
  2   3   4   2   4   4   2   4   4   2   4   4   4   5   5   4   5   4   3   4   4   2   1   4
 49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
```

For this data set, we will work with 5 clusters; hence, we do k5 = kmeans(d.scale, centers = 5) to indicate our 5 clusters. Then, to see how many data points are within each cluster, we use k5$size and can use k5$cluster to see the different data points. k5$tot.withinss means shows how far apart the clusters are. Given that our value is about 8709, it is relatively high so there could be less separation between these clusters (may have overlap perhaps)?

```
fviz_cluster(k5, data = d.scale) #visualize the k4 cluster with d.scale data
#we can see there are some distincy clusters but also some overlap between the clusters
cluster::clusplot(d.scale, k5$cluster)
```





To visualize our clusters, we use fviz_cluster and cluster::clusplot. We see the overlap with some of the clusters which may indicate why our totwithinss was high. We can also see that there are

data points that could be seen as outliers which may skew the cluster's centers a bit. Nevertheless, while looking at the clusters, we can see that cluster 2 has the most separation in comparison to the other clusters. Investigating cluster 2 would be helpful in our predictions.

```
c1 = cancer_data[k5$cluster==1,] #slice/subset data set to see where the cluster value is 1
c2 = cancer_data[k5$cluster==2,]
c3 = cancer_data[k5$cluster==3,]
c4 = cancer_data[k5$cluster==4,]
c5 = cancer_data[k5$cluster==5,]
```

Data	
c1	119 obs. of 31 variables
c2	90 obs. of 31 variables
c3	31 obs. of 31 variables
c4	96 obs. of 31 variables
c5	233 obs. of 31 variables

We now subset the clusters into their own. However, we now want to bring back our original data set with diagnosis to help create class labels in the classification process. Looking at each of the clusters, we can see that they match the sizes of the clusters when we use k5$size.

```
> c1
    diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
22          B       9.504        12.44          60.34     273.9         0.10240
47          B       8.196        16.84          51.71     201.9         0.08600
60          B       8.618        11.79          54.34     224.5         0.09752
61          B      10.170        14.88          64.55     311.9         0.11340
62          B       8.598        20.98          54.66     221.8         0.12430
64          B       9.173        13.86          59.20     260.9         0.07721
67          B       9.465        21.01          60.11     269.4         0.10440
72          B       8.888        14.64          58.79     244.0         0.09783
77          B      13.530        10.94          87.91     559.2         0.12910
81          B      11.450        20.97          73.81     401.5         0.11020
89          B      12.360        21.80          79.78     466.1         0.08772
98          B       9.787        19.94          62.11     294.5         0.10240
102         B       6.981        13.43          43.79     143.5         0.11700
104         B       9.876        19.40          63.95     298.3         0.10050
```

For better context, when we use c1, it is the data points that are within the original data set that are clustered in cluster 1. The same follows when we run c2-c5.

Classification Process:

```
> #before creating training
> table(c1$diagnosis)

  B   M
119   0
> table(c2$diagnosis)

 B  M
 0 90
> table(c3$diagnosis)

 B  M
 8 23
> table(c4$diagnosis)

 B  M
17 79
> table(c5$diagnosis)

  B   M
213  20
```

While looking at the clusters against the diagnosis, we can see that cluster 1 and 5 have a high benign count and cluster 2 has a high malignant count. As for cluster 3 and 4, it is somewhat in between even though malignant has more counts. So, for our class attribute, we will assign the following for our class attribute: high benign, high malignant, medium malignant, and all_others for when we do classification.

```
> #class attribute
> class.attribute = ifelse(k5$cluster==1 & k5$cluster == 5, 'high_B',
+                          ifelse(k5$cluster==2, 'high_M',
+                                 ifelse(k5$cluster==3 & k5$cluster == 4, 'medium_M', 'all_the_others')))
> #check data type for class.attribute
> class(class.attribute) #need to convert to factor
[1] "character"
```

To create our class attribute, we need to do ifelse(). After writing the following conditions based on what our clusters provide, we see that class.attribute is character. We need to convert it into factor for the classification process.

```
> cancer_data = data.frame(cancer_data,class.attribute) #adding "class.attribute"
> str(cancer_data)
'data.frame':   569 obs. of  32 variables:
 $ 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 ...
 $ class.attribute        : chr  "medium_M" "high_M" "high_M" "medium_M" ...
```

Before converting it, we needed to insert class.attribute into the cancer_data set. This is because since we have diagnosis and our class.attribute that provide different diagnosis conditions, we need to remove the diagnosis column since they are essentially the same and we would like to focus on class.attribute with its given conditions.

```
> #need to change or remove diagnosis since it is similar to class.attribute
> d1 = subset(cancer_data, select = -diagnosis) # removes diagnosis from the data frame
> str(d1)
'data.frame':   569 obs. of  31 variables:
 $ 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 ...
 $ class.attribute        : chr  "medium_M" "high_M" "high_M" "medium_M" ...
```

To remove diagnosis from our data set, we subset a new data frame from the original cancer data called d1 which excludes the diagnosis column.

```
> #convert class attribute to factor
> d1$class.attribute = as.factor(d1$class.attribute)
> str(d1)
'data.frame':   569 obs. of  31 variables:
 $ 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 ...
 $ class.attribute        : Factor w/ 3 levels "high_B","high_M",..: 3 2 2 3 2 3 3 3 3 3 ...
```

After creating our new data frame, we then convert and check to see if class.attribute has changed from character to factor. Now that we have factor as our data type for class.attribute, we can go ahead and begin splitting our data into training and test sets.

```
#training set
train.index = sample(1:nrow(d1), .8*nrow(d1))
train.index
train.set = d1[train.index,] #create train set
train.set #view train set
```

train.set	455 obs. of 31 variables
Values	
class.attribute	chr [1:569] "medium_M" "all_the_others" "all_t…
d.dist	Large dist (161596 elements, 1.3 MB)
train.index	int [1:455] 121 110 158 64 483 477 480 67 85 1…

```
> train.index = sample(1:nrow(d1), .8*nrow(d1)) #use 80% of data for training
> train.index
  [1] 121 110 158  64 483 477 480  67  85 165  51  74 178 362 236 330 127 212 310 243 113 564 151
 [24] 160 391 155 426   5 326 280 238 339  39 137 455  83 537 196 286 500 344 551 459  20 195 164
 [47]  52 177  84 392 302 430 428 250 429 398 381  33  40 424  10 473 200 466 125 501 265 263 545
 [70] 186  61 252 458 152 319  54 407  26 235 289 185 253 413 115 509 309 494 205 363 267  25 523
 [93] 282 504 409 215 346 546 468 408  57 457 105 357 279 270 366 134 347 129 218 106 369 530 337
[116] 285 492  27   7 444 245 154  41 552 390 222 159 349 145 421 469 148 163 539 161  66   4 554
[139] 225 389 117 479 136  55 217 561  45 146 170 415 462 199 361 176 401 449 484 104 487 495 418
[162] 210 442 420 431 258 522 386 141  24 130 560 191  76 377 269 440 198 556 234 368 566  80  36
[185] 291 488 343 323  48 111 465 317 295 450 287 376  73 292 226 278 172 297 348  93 505 299 533
[208] 237 514 107 512 534 354 277 209 396  94 385  30 425 175 382 486 338  96 393 345 453 230 438
[231] 202  81 232 550 497  11 549 406  31 461 370 353 538  16 197 557 403  12 434  50 204 493 419
[254] 122 315 259 248 490 513 100 108 301 485 529 540 355 536 411   8 114 261  29 558 548 373 563
```

```
> train.set = d1[train.index,] #create train set variable
> train.set #view train set
    radius_mean texture_mean perimeter_mean area_mean smoothness_mean compactness_mean
121      11.410        10.82          73.34     403.3         0.09373           0.06685
110      11.340        21.26          72.48     396.5         0.08759           0.06575
158      16.840        19.46         108.40     880.2         0.07445           0.07223
64        9.173        13.86          59.20     260.9         0.07721           0.08751
483      13.470        14.06          87.32     546.3         0.10710           0.11550
477      14.200        20.53          92.41     618.4         0.08931           0.11080
480      16.250        19.51         109.80     815.8         0.10260           0.18930
67        9.465        21.01          60.11     269.4         0.10440           0.07773
85       12.000        15.65          76.95     443.3         0.09723           0.07165
165      23.270        22.04         152.10    1686.0         0.08439           0.11450
51       11.760        21.60          74.72     427.9         0.08637           0.04966
```

For this project, we will split our dataset by training 80% of the data and testing 20% of it. Thus we have .8*nrow(d1). We can also see in the global environment that 80% of the data is being trained (455/569). Note that when we split our data, the data is randomized; thus, when we view the train index and train set, the rows have been randomized.

```
> #test set
> test.set = d1[-train.index,] #excludes training set
> test.set
    radius_mean texture_mean perimeter_mean area_mean smoothness_mean compactness_mean
1         17.990        10.38         122.80    1001.0         0.11840           0.27760
3         19.690        21.25         130.00    1203.0         0.10960           0.15990
9         13.000        21.82          87.50     519.8         0.12730           0.19320
14        15.850        23.95         103.70     782.7         0.08401           0.10020
15        13.730        22.61          93.60     578.3         0.11310           0.22930
18        16.130        20.68         108.10     798.8         0.11700           0.20220
22         9.504        12.44          60.34     273.9         0.10240           0.06492
34        19.270        26.47         127.90    1162.0         0.09401           0.17190
35        16.130        17.88         107.00     807.2         0.10400           0.15590
42        10.950        21.35          71.90     371.1         0.12270           0.12180
44        13.280        20.28          87.32     545.2         0.10410           0.14360
53        11.940        18.24          75.71     437.6         0.08261           0.04751
56        11.520        18.75          73.34     409.0         0.09524           0.05473
58        14.710        21.59          95.55     656.9         0.11370           0.13650
```

test.set	114 obs. of 31 variables
train.set	455 obs. of 31 variables

Once we have created our training set, we create our test set which is the remaining parts of the d1 data frame (-train.index). When we view it in our global environment we can see that 20% of the data has been randomized into the test.set (114/569).

```
> #creating our test attribute (class attribute)
> cancer.test = d1$class.attribute[-train.index]
> cancer.test
  [1] medium_M high_M   medium_M high_B   medium_M medium_M high_B   high_M   medium_M medium_M
 [11] medium_M high_B   high_B   medium_M high_B   high_B   medium_M high_M   high_B   medium_M
 [21] high_B   medium_M medium_M high_B   high_B   high_B   high_B   high_B   medium_M high_B
 [31] high_B   medium_M high_M   high_B   high_M   medium_M high_B   high_B   medium_M high_B
 [41] high_B   medium_M high_B   high_B   high_M   medium_M high_B   high_M   high_B   high_B
 [51] high_B   medium_M high_B   high_B   high_B   high_B   high_B   high_B   high_B   high_B
 [61] high_B   high_B   medium_M high_B   high_B   high_B   high_B   high_M   medium_M high_B
 [71] high_B   high_B   medium_M high_B   high_B   high_B   high_B   high_M   high_B   high_M
 [81] high_B   high_B   high_B   high_B   high_B   high_B   high_M   high_B   medium_M high_B
 [91] high_M   high_B   high_B   high_B   high_B   high_B   high_B   high_B   high_B   high_B
[101] medium_M high_B   high_M   high_B   high_B   high_B   medium_M high_B   high_B   high_B
[111] medium_M high_B   high_B   high_B
Levels: high_B high_M medium_M
```

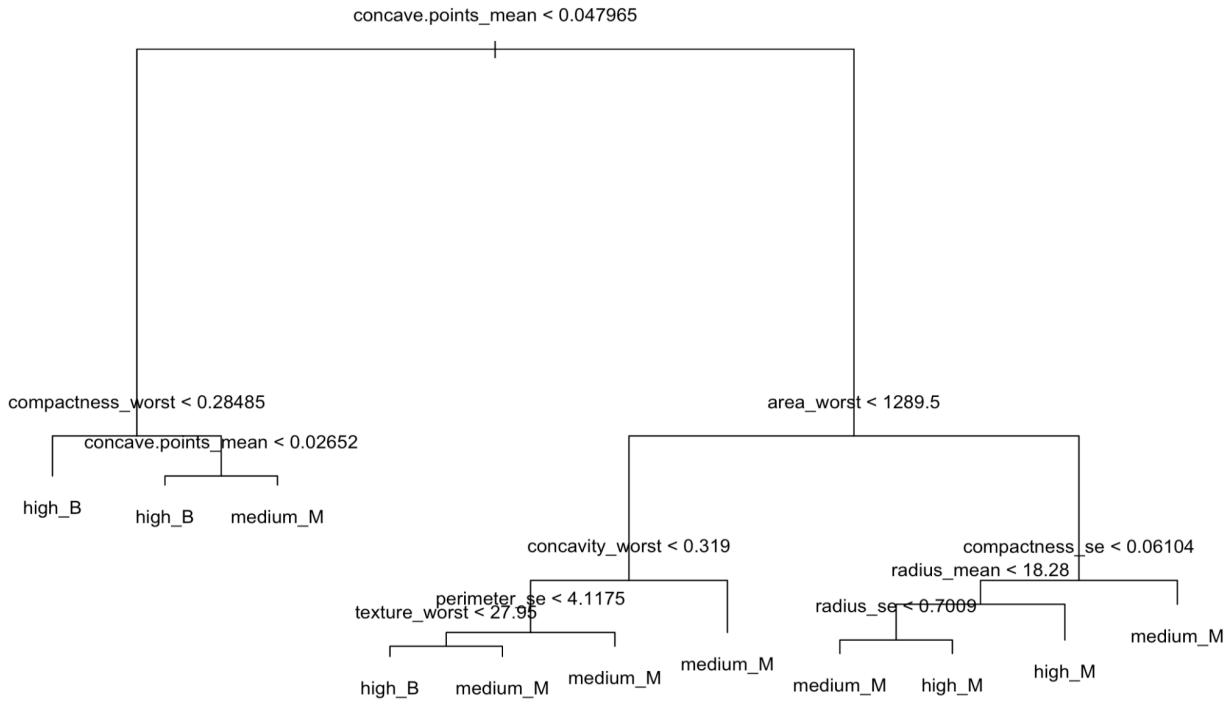
Considering that our class.attribute is what are target is for classification, we now need to make it our test attribute. To do this we do d1$class.attribute[-train.index]. Our test attribute is the class label that is being predicted. So when we test our model, a confusion matrix for cancer.test will be made based on cancer.test to see how well our model does for predictions.

```
> #train model ####
> dtree = tree(class.attribute~., train.set) #decision tree with train set
> dtree
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 455 852.600 high_B ( 0.60879 0.16923 0.22198 )
   2) concave.points_mean < 0.047965 264  71.700 high_B ( 0.96970 0.00000 0.03030 )
     4) compactness_worst < 0.28485 242   0.000 high_B ( 1.00000 0.00000 0.00000 ) *
     5) compactness_worst > 0.28485 22  28.840 high_B ( 0.63636 0.00000 0.36364 )
      10) concave.points_mean < 0.02652 8   0.000 high_B ( 1.00000 0.00000 0.00000 ) *
      11) concave.points_mean > 0.02652 14  19.120 medium_M ( 0.42857 0.00000 0.57143 ) *
   3) concave.points_mean > 0.047965 191 366.500 medium_M ( 0.10995 0.40314 0.48691 )
     6) area_worst < 1289.5 98 120.400 medium_M ( 0.21429 0.02041 0.76531 )
      12) concavity_worst < 0.319 35  54.320 high_B ( 0.60000 0.02857 0.37143 )
        24) perimeter_se < 4.1175 29  34.160 high_B ( 0.72414 0.00000 0.27586 )
          48) texture_worst < 27.95 23  17.810 high_B ( 0.86957 0.00000 0.13043 ) *
          49) texture_worst > 27.95 6   5.407 medium_M ( 0.16667 0.00000 0.83333 ) *
        25) perimeter_se > 4.1175 6   5.407 medium_M ( 0.00000 0.16667 0.83333 ) *
      13) concavity_worst > 0.319 63  10.270 medium_M ( 0.00000 0.01587 0.98413 ) *
     7) area_worst > 1289.5 93  91.390 high_M ( 0.00000 0.80645 0.19355 )
      14) compactness_se < 0.06104 86  65.770 high_M ( 0.00000 0.87209 0.12791 )
        28) radius_mean < 18.28 20  27.530 medium_M ( 0.00000 0.45000 0.55000 )
          56) radius_se < 0.7009 11   6.702 medium_M ( 0.00000 0.09091 0.90909 ) *
          57) radius_se > 0.7009 9   6.279 high_M ( 0.00000 0.88889 0.11111 ) *
        29) radius_mean > 18.28 66   0.000 high_M ( 0.00000 1.00000 0.00000 ) *
      15) compactness_se > 0.06104 7   0.000 medium_M ( 0.00000 0.00000 1.00000 ) *
```

Now that we have set up our training and test sets, we train our model and create a decision tree using the tree package. Looking into the decision tree, we can see which variables under certain conditions are the terminal nodes (with the highest purity in terms of the Gini index). For example, if concave.points_mean < 0.047965 and compactness_worst < 0.28485, then the tumor is diagnosed high benign (high_b). Meaning that the tumor is likely to not be dangerous. In addition, we can also see from the decision tree that if concave.points_mean < 0.047965, compactness_worst < 0.28485, and concave.points_mean > 0.02652 then the tumor is likely to be medium malignant (medium_M). Indicating that if the tumor's measurements follow this pattern, there is somewhat of a chance that the tumor is malignant or serious. Doctors may need to further test the patient or examine the tumor to see whether or not it is dangerous to be sure it is malignant.

```
> summary(dtree) #get info about dtree

Classification tree:
tree(formula = class.attribute ~ ., data = train.set)
Variables actually used in tree construction:
[1] "concave.points_mean" "compactness_worst"   "area_worst"          "concavity_worst"
[5] "perimeter_se"        "texture_worst"       "compactness_se"      "radius_mean"
[9] "radius_se"
Number of terminal nodes:  11
Residual mean deviance:  0.1599 = 71 / 444
Misclassification error rate: 0.03077 = 14 / 455
```

To dive more deeply into the decision tree summary(dtree) let us see the number of nodes and the MSE. Seeing that the number of terminal nodes is 11, we would like to prune the tree as a result of the decision tree being large. A large and complex decision tree could mean that the decision tree model is overfitting the training data. Notice also that the MSE is 0.03/3%, showing that only 3% of the errors were misclassified. Hence, because of the complexity of the data, we will prune it to reduce the number of terminal nodes and choose the most optimal number of nodes for our model.



With plot(dtree) and text(dtree) we can see what our decision tree looks like visually. As mentioned before, 11 terminal nodes are complex of a model, so we would like to still reduce

complexity as much as possible. Hence, we will soon begin pruning our model to make it less complex and hopefully be more accurate.

```
> #testing the trained model ####
> dtree.test = predict(dtree, test.set, type = 'class')
> table(dtree.test,cancer.test) #confusion matrix
           cancer.test
dtree.test high_B high_M medium_M
  high_B       69      0        2
  high_M        1     13        1
  medium_M      5      0       23
```

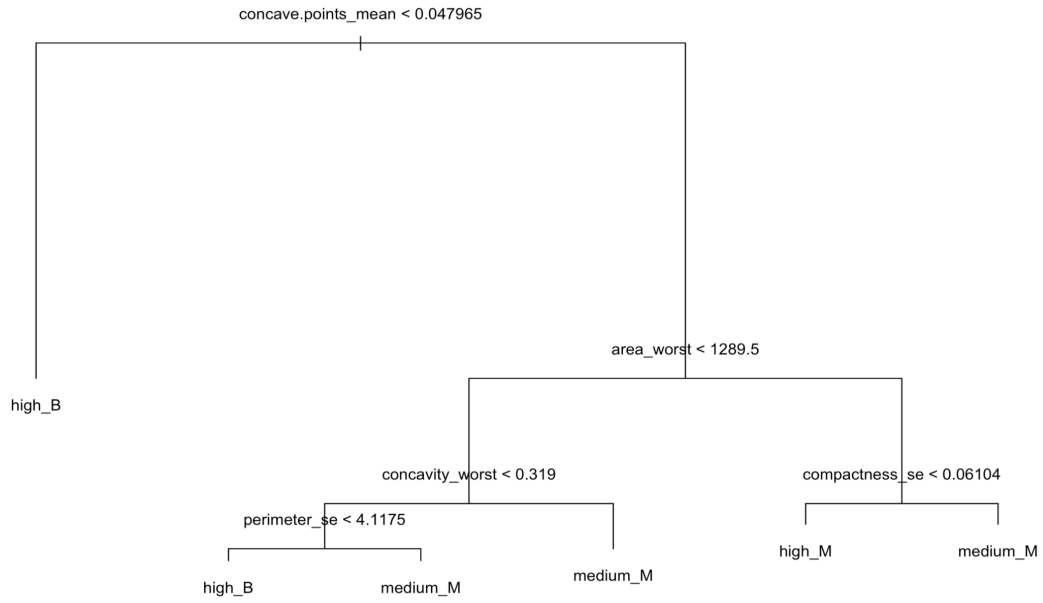
Before pruning our model, we would like to see how accurate our initial model is when predicting the severity of the tumor. Based on the calculation (69+13+23)/114, our testing accuracy for this model is about 92%. In terms of prediction and on a medical basis, this is strongly accurate. However, we would like to see if pruning or running randomForest would help improve our testing accuracy.

```
> #Cross-validation
> cv.tree(dtree, FUN = prune.misclass) #find what is the most optimal node for our tree
$size
[1] 11  9  8  6  5  4  3  2  1

$dev
[1]  41  41  55  61  62  65  65 113 173

$k
[1] -Inf  1.0  4.0  4.5  5.0  7.0  8.0 57.0 72.0

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

To find the number of optimal terminal nodes, cross-validation or cv.tree will help us indicate which number of nodes we should use to prune our model. As we can see in the $size vs $dev section, the deviation decreases significantly from 1 to 2 and slowly goes down from 3 and beyond. In this case, we will select 6 terminal nodes for our pruned model since it is the right amount of nodes to avoid under and overfitting the data.

```
> #pruning the trained model
> dtree.pruned = prune.misclass(dtree, best = 6) #best tree of size 6
> plot(dtree.pruned)
> text(dtree.pruned)
>
```

After selecting our optimal number of terminal nodes, we then plot to see what the new pruned model looks like.




```
> dtree.pruned
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 455 852.600 high_B ( 0.60879 0.16923 0.22198 )
   2) concave.points_mean < 0.047965 264  71.700 high_B ( 0.96970 0.00000 0.03030 ) *
   3) concave.points_mean > 0.047965 191 366.500 medium_M ( 0.10995 0.40314 0.48691 )
     6) area_worst < 1289.5 98 120.400 medium_M ( 0.21429 0.02041 0.76531 )
      12) concavity_worst < 0.319 35  54.320 high_B ( 0.60000 0.02857 0.37143 )
        24) perimeter_se < 4.1175 29  34.160 high_B ( 0.72414 0.00000 0.27586 ) *
        25) perimeter_se > 4.1175 6   5.407 medium_M ( 0.00000 0.16667 0.83333 ) *
      13) concavity_worst > 0.319 63  10.270 medium_M ( 0.00000 0.01587 0.98413 ) *
     7) area_worst > 1289.5 93  91.390 high_M ( 0.00000 0.80645 0.19355 )
      14) compactness_se < 0.06104 86  65.770 high_M ( 0.00000 0.87209 0.12791 ) *
      15) compactness_se > 0.06104 7   0.000 medium_M ( 0.00000 0.00000 1.00000 ) *
```

In our pruned model, we can see that the data is less complex. To interpret some of the data points, we can see that if concave.points_mean < 0.047965, then the tumor is likely to be highly

benign (high_B) showing that the tumor may not be serious. However, on the right side of the tree, if the concave.points_mean > 0.047965, area_worst > 1289.5, and the compactness_se < 0.06104 then the tumor is highly malignant (high_M). Thus, patients with tumors under those conditions are likely to have highly malignant tumors/is serious.

```
> summary(dtree.pruned)

Classification tree:
snip.tree(tree = dtree, nodes = c(2L, 24L, 14L))
Variables actually used in tree construction:
[1] "concave.points_mean" "area_worst"          "concavity_worst"     "perimeter_se"
[5] "compactness_se"
Number of terminal nodes:  6
Residual mean deviance:  0.4172 = 187.3 / 449
Misclassification error rate: 0.06374 = 29 / 455
```

Furthermore, when we run summary(dtree.pruned), we can see that the MSE did go up by 3% as it is now 6%. in comparison to the decision tree. This is likely to occur because we are preventing the model from overfitting our training data. Regardless of the MSE going up, the idea that 6% of the data was misclassified shows how strong of a model it is when we are running classification with our pruned model.

```
> #test the pruned tree
> dtree.pruned.test = predict(dtree.pruned, test.set, type='class')
> table(cancer.test, dtree.pruned.test)
            dtree.pruned.test
cancer.test high_B high_M medium_M
   high_B       74      1        0
   high_M        0     13        0
   medium_M      9      3       14
```

To calculate the dtree.pruned model's accuracy, (74+13+14)/114 = 0.886 or roughly 89%. The pruned model's accuracy is still strong and can be useful when medical professionals are diagnosing a patient's tumor. Even if the pruned model's accuracy is less than the original decision tree model's accuracy by 3%, we can still say that 89% accuracy is strong.

```
> #training set with Random Forest
> dtree.rf = randomForest(class.attribute~., train.set) #train model with random forest
> dtree.rf

Call:
 randomForest(formula = class.attribute ~ ., data = train.set)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 5

        OOB estimate of  error rate: 5.71%
Confusion matrix:
         high_B high_M medium_M class.error
high_B      271      0        6  0.02166065
high_M        0     71        6  0.07792208
medium_M      8      6       87  0.13861386
```

Finally, to see if our testing accuracy can still increase, we will use random forest. Based on the OOB estimate being 5.71%, our trained model accuracy when trained and tested at the same time is 94.29%.

```
> #testing the trained model with Random Forest
> dtree.rf.test = predict(dtree.rf, test.set, type='class')
> table(cancer.test, dtree.rf.test) #confusion matrix
           dtree.rf.test
cancer.test high_B high_M medium_M
   high_B       74      0        1
   high_M        0     13        0
   medium_M      1      0       25
```

Using random forest, the accuracy percentage is (74+13+25)/114 = 0.98 or 98%! This means that when we use random forest, our model's accuracy in terms of predicting the tumor's diagnosis is 98%. In the medical field, this high accuracy for prediction is crucial as it can help detect malignant tumors or benign tumors and examine the behavior of the tumor if medium malignant.

Concluding thoughts:

This data set for breast cancer diagnoses was published by the University of California Irvine. It investigates certain conditions for example the area, compactness, and texture of different sections of a tumor to determine whether or not the patient's tumor is malignant (cancerous) or benign (not cancerous). When medical professionals analyze the tumor, they split up sections of the tumor to analyze the cells and its components (area, radius, texture, etc.). This then leads to them calculating the means, standard errors, and "worse" measurements of the tumor. Mean (average amongst the measurements), standard error (how much the measurements vary; smaller value = consistent, large value = inconsistent), and worst (highest mean amongst the different sections of the tumor). For instance, area_mean is the average radius of the cells, area_se is the standard error or how much the cell's area measurements vary, and area_worst is the mean of the three largest area measurements.

So, given the context of how the tumor is analyzed, medical professionals can use these data mining techniques to help accurately predict whether or not a tumor is likely to be malignant or benign. We were able to see that when using random forest the testing accuracy was 98%. Given its high accuracy, doctors will be able to quickly diagnose the tumor based on the defined conditions that determine if it is malignant or benign. Even if random forest has an extremely high prediction accuracy, the other modeling techniques such as our original decision tree and pruning our model had high testing accuracy scores of 92% and 89% respectively. This comes to show that with the following data, using either technique (preferably random forest) could still lead to accurate and quick results.

With a quick response to detect either possibility, this can lead to patients who were able to catch it early getting treatment as soon as possible; thus, increasing the chances of the patient to go into remission. Otherwise, if it is benign, the doctor can reassure the patient that it is not serious, but may ask the patient to have schedules on a monthly or yearly basis to keep track of the tumor's behavior. Furthermore, if the patient falls within the medium_malignant category, extra steps such as more testing/monitoring can be taken to determine whether or not it is malignant. An example of this would be that if we use the decision tree from the pruned model, if a patient's tumor has concave.points_mean > 0.047965, area_worst < 1289.5, concavity_worst < 0.319, and perimeter_se > 4.1175 the tumor would be diagnosed as medium malignant.

Indicating that there is a chance that the tumor may be malignant/cancerous. Because there is some uncertainty but it is somewhat likely that it is malignant, doctors may ask the patient to undergo more tests to see what else needs to be done to diagnose it.

The idea of quick detection especially when it comes to cancer can help many individuals go into remission. Many have lost their lives to cancer, so if using data mining can tremendously help predict these outcomes, it can save many lives. Not only can accurate predictions lead to saving lives, they can also encourage further research in the healthcare industry since identifying these patterns may spark new research regarding patients' diet, history, or lifestyle.

Thus far, seeing how accurate predictions using classification and identifying similarities between data points with clustering, the future of healthcare can improve tremendously.