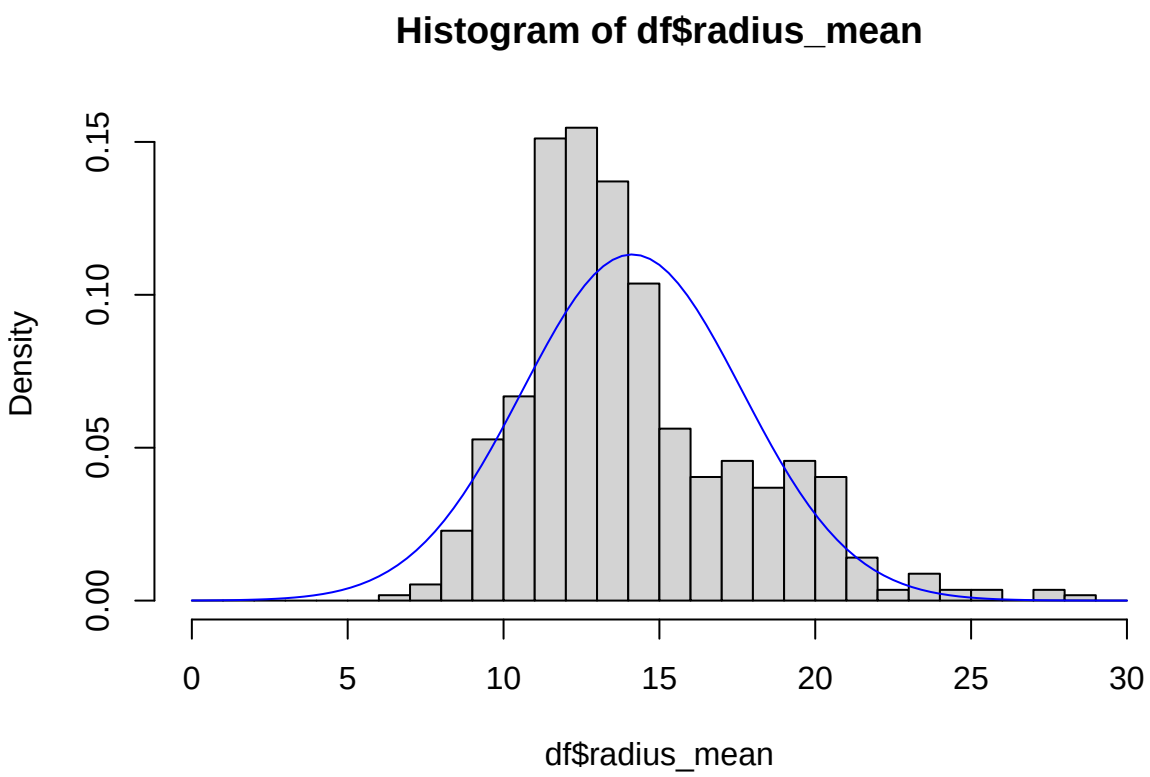
Practicum 1

John Keith

October 4 2023

1 / Predicting Breast Cancer

1.1 / Analysis of Data Distribution



It seems like the radius_mean data is fairly left skewed, not following a normal distribution. It matters to see if the data is normally distributed because you can only perform certain statistical/machine learning tests if the data is normally distributed.

From the Shapiro-Wilk test, we find that the p-value from the test is 3.10564350835627e-14, which is much less than $p = 0.05$, therefore showing that the data is not normally distributed.

1.2 / Identification of Outliers

```
find_outliers <- function(column){
  m <- mean(column)
  std_column <- sd(column)
  z_score <- abs(m - column[1:length(column)]) / std_column
  return(which(z_score > 2.5))
  }
num_outliers <- 0
for (i in 2:ncol(df)){
  outliers <- find_outliers(df[,i])
  print(paste('Outliers for column', colnames(df[i]), 'are in rows'))
  print(outliers)
  num_outliers <- num_outliers + length(outliers)

}
```

```
## [1] "Outliers for column radius_mean are in rows"
##  [1]  83 123 165 181 203 213 237 340 353 462 504 522
## [1] "Outliers for column texture_mean are in rows"
## [1] 220 233 240 260 266 456 563
## [1] "Outliers for column perimeter_mean are in rows"
##  [1]  83 109 123 181 203 213 237 340 353 462 522
## [1] "Outliers for column area_mean are in rows"
##  [1]  83 123 165 181 203 213 237 340 353 369 462 504 522
## [1] "Outliers for column smoothness_mean are in rows"
## [1]   4 106 109 123 204 258 505 521 569
## [1] "Outliers for column compactness_mean are in rows"
##  [1]   1   4  10  13  79  83 109 123 182 191 259 352 401 568
## [1] "Outliers for column concavity_mean are in rows"
##  [1]   1  69  79  83 109 113 123 153 203 213 259 352 353 401 462 564 568
## [1] "Outliers for column concave.points_mean are in rows"
##  [1]   1  79  83  84 109 123 181 182 203 213 324 353 370 394 462 522 564 568
## [1] "Outliers for column symmetry_mean are in rows"
##  [1]   4  23  26  61  79 109 123 147 151 153 289 324 425 562
## [1] "Outliers for column fractal_dimension_mean are in rows"
##  [1]   4  10  69  72  79 152 153 177 259 319 377 505 506
## [1] "Outliers for column radius_se are in rows"
##  [1]  39 109 123 139 213 259 266 273 301 369 418 462 504 565
## [1] "Outliers for column texture_se are in rows"
##  [1]  13  84 123 153 193 246 315 390 417 472 474 529 558 560 562
## [1] "Outliers for column perimeter_se are in rows"
##  [1]   1  13  39  43  79 109 123 139 213 259 273 418 462 504 564
## [1] "Outliers for column area_se are in rows"
##  [1] 109 123 213 237 266 273 340 369 418 462 504 565
## [1] "Outliers for column smoothness_se are in rows"
##  [1]  72 117 123 174 214 246 289 315 346 392 417 470 506 521 540
## [1] "Outliers for column compactness_se are in rows"
##  [1]   4  10  13  43  63  69  72 109 113 123 153 177 191 214 289 291 377 469 486
## [1] "Outliers for column concavity_se are in rows"
##  [1]  69  79 113 123 153 177 191 214 377 486
## [1] "Outliers for column concave.points_se are in rows"
##  [1]  13  69 139 153 162 211 214 259 289 291 377 390 529
```

```
## [1] "Outliers for column symmetry_se are in rows"
##  [1]   4  13  43  61  64  69  79 120 123 139 147 191 213 215 315 333 352
## [1] "Outliers for column fractal_dimension_se are in rows"
##  [1]  13  72 113 152 153 177 191 214 291 377 389 469 505 506
## [1] "Outliers for column radius_worst are in rows"
##  [1]  24  83 109 181 237 266 340 353 369 462 504 522
## [1] "Outliers for column texture_worst are in rows"
## [1] 204 220 233 240 260 266 456 563
## [1] "Outliers for column perimeter_worst are in rows"
##  [1]  83 109 181 237 266 273 340 353 369 370 462 504 522
## [1] "Outliers for column area_worst are in rows"
##  [1]  19  24  83 109 165 181 213 220 237 266 273 340 353 369 462 504 522
## [1] "Outliers for column smoothness_worst are in rows"
## [1]   4  42 193 204 380 505 506
## [1] "Outliers for column compactness_worst are in rows"
##  [1]   1   4  10  15  16  27  34  43  73 109 182 191 380 431 563 568
## [1] "Outliers for column concavity_worst are in rows"
##  [1]  10  69 109 153 191 253 380 401 431 563 568
## [1] "Outliers for column concave.points_worst are in rows"
## [1]  83 109 182
## [1] "Outliers for column symmetry_worst are in rows"
##  [1]   1   4  23  32  36  43  79 120 147 191 200 215 324 371 490
## [1] "Outliers for column fractal_dimension_worst are in rows"
##  [1]   4  10  15  16  32  73 106 152 191 243 253 380 506 563
```

The above code outputs all of the outliers per column. As you can see, there are a decent amount of outliers for each column, making it important to deal with/normalize. If I were performing a statistical/machine learning task that assumed a normal distribution (such as linear regression), I would simply standardize the data using the z-score method. Otherwise, I would probably use the min-max normalization method in order to get the entire column of data on a scale of 0 to 1, in order to allow the machine learning task to not get thrown off by a difference in scales. To find the outliers in each column, I used the z-score formula in combination wtih the which function to see which data points in a column were greater than 2.5. The amount of outliers I found were 388.

1.3 / Data Preparation

```
for (i in 2:ncol(df)){
  df[,i] <- (df[,i] - mean(df[,i])) / sd(df[,i])

}
```

I normalized each column of data by simply applying the z-score formula to each element in each row. You do this in order to allow any machine learning method to not be thrown off by changes in scales between different measurements/types of data.

1.4 / Sampling Training and Validation Data

```
set.seed(202)
df_M <- df[which(df$diagnosis == 'M'),]
df_B <- df[which(df$diagnosis == 'B'),]
```

```
sample <- sample.int(n = nrow(df_M), size = floor(.20*nrow(df_M)))
validation_df <- df_M[sample,]
training_df <- df_M[-sample,]
sample_b <- sample.int(n = nrow(df_B), size = floor(.20*nrow(df_B)))
validation_df <- rbind(validation_df, df_B[sample_b,])
training_df <- rbind(training_df, df_B[-sample_b,])
```

1.5 / Predictive Modeling

```
library(class)
df_training_labels <- training_df[,1]
df_training <- training_df[,-1]
df_validation_labels <- validation_df[,1]
df_validation <- validation_df[,-1]

knn_test_pred <- knn(train = df_training, test = df_validation, cl = df_training_labels, k = 5)

#checking if the knn model works well
library(gmodels)
CrossTable(x = df_validation_labels, y = knn_test_pred, prop.chisq = FALSE)
```

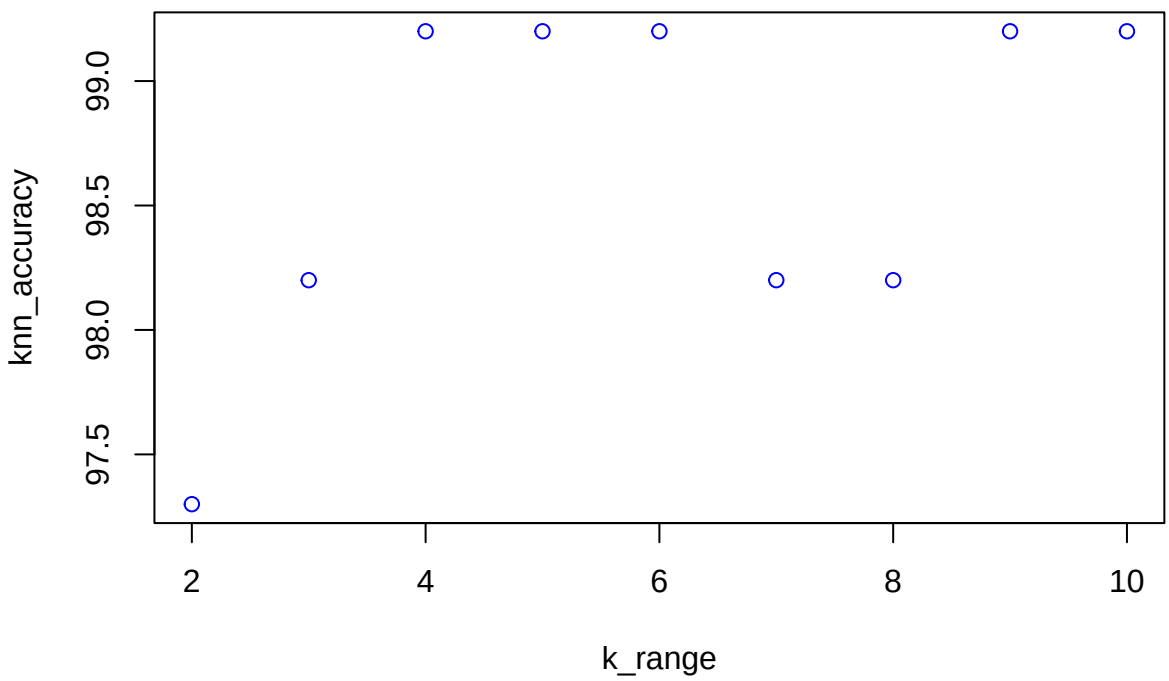
```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  113
##
##
##                      | knn_test_pred
## df_validation_labels |         B |         M | Row Total |
## ---------------------|-----------|-----------|-----------|
##                    B |        71 |         0 |        71 |
##                      |     1.000 |     0.000 |     0.628 |
##                      |     0.986 |     0.000 |           |
##                      |     0.628 |     0.000 |           |
## ---------------------|-----------|-----------|-----------|
##                    M |         1 |        41 |        42 |
##                      |     0.024 |     0.976 |     0.372 |
##                      |     0.014 |     1.000 |           |
##                      |     0.009 |     0.363 |           |
## ---------------------|-----------|-----------|-----------|
##         Column Total |        72 |        41 |       113 |
##                      |     0.637 |     0.363 |           |
## ---------------------|-----------|-----------|-----------|
```

```
##
##
```

```
# Around 99% accuracy, so the model works very well


#load original df for non z-score values
df_orig <- read.csv('Wisonsin_breast_cancer_data.csv')[-1][-32]

new_data <- list(14.5, 17.0, 87.5, 561.3, 0.098, 0.105, 0.085, 0.050, 0.180, 0.065, 0.351, 1.015, 2.457

df_new_data <- df_training[1,]
for (i in 1:ncol(df_new_data)){
  df_new_data[1,i] <- new_data[i]
}

# z-score noramlize the data (except for 27 and 28)
for (i in 2:ncol(df_new_data)){
  df_new_data[,i-1] <- (df_new_data[,i-1] - mean(df_orig[,i])) / sd(df_orig[,i])
}

# knn test for new values
new_data_knn_pred <- knn(train = df_training, test = df_new_data, cl = df_training_labels, k = 5)
```

I used the knn method from the class package to predict what the diagnosis (Malignant or Benign) would be for the new cancer data given in the assignment. I firs used the testing data to make the model, then saw how well the model predicted the validation data, and it correctly predicted the diagnosis of the validation set 99% of the time, so it worked very well. Then I applied the model to the new data (z-score standardized), and the model predicted that the diagnosis from the new data would be B, which would be benign.

1.6 / Model Accuracy



The chart shows the accuracy of the model based on the K values chosen between 2 and 10. As you can see, the model itself among all of these K values is fairly accurate(around 97-99% accuracy for k=2-10). However, if I were to apply this model and chose one K value for future knn predictions, I would chose a K of 10 because a good starting point for k is the square root of the training sample, and 10 is the closest of these values to that while being among the most accurate.

2 / Predicting Age of Abalones using Regression kNN

2.1 / Load csv and save target vector and training data

```
df_abalone <- read.csv('https://s3.us-east-2.amazonaws.com/artificium.us/datasets/abalone.csv')

target_data <- as.vector(df_abalone$NumRings)
train_data <- df_abalone[-8]
```

2.2 / Encoding Categorical Variables

```
#the only categorical variable is sex, which is a good case for using one-hot encoding. We will create

train_data$Male <- 0
train_data$Female <- 0
```

```r
for (i in 1:nrow(train_data)){
  if (train_data$Sex[i] == 'I'){
    train_data$Male[i] <- FALSE
    train_data$Female[i] <- FALSE
  }
  if (train_data$Sex[i] == 'M'){
    train_data$Male[i] <- TRUE
    train_data$Female[i] <- FALSE
  }
  if (train_data$Sex[i] == 'F'){
    train_data$Male[i] <- FALSE
    train_data$Female[i] <- TRUE
  }
}

#get rid of the Sex column, now that sex is stored in two seperate columns.
train_data <- train_data[-8]
```

2.3 / Normalize all colums using min-max normalization

```r
#Remember not to normalize the two sex columns; train_data[8:9]
for (i in 1:(ncol(train_data)-2)){
  train_data[,i] <- (train_data[,i] - min(train_data[,i])) / (max(train_data[,i]) - min(train_data[,i]))
}
```

2.4 / Build knn.reg function

```r
#establish variables/data needed for knn.reg function
knn_dist_list <- list()
new_data <- train_data[1,]
new_data_list <- list(0.44, 0.391,0.254, 0.2132, 0.0878, 0.21,0.5853, 0, 1)
for (i in 1:ncol(new_data)){
  new_data[1,i] <- new_data_list[i]
}
#make the knn.reg function
knn.reg <- function(new_data, target_data, train_data, k){
  # knndist <- sqrt((xnewdata - xtraindata)^2 + (ynewdata - ytraindata)^2 ...)
  for (i in 1:nrow(train_data)){
    knn_dist_presqrt <- 0
    knn_dist_unsqrt <- train_data[i,] - new_data[1,]
    for (q in 1:length(knn_dist_unsqrt)){
      knn_dist_presqrt <- knn_dist_presqrt + (knn_dist_unsqrt[,q])^2
    }
    knn_dist_final <- sqrt(knn_dist_presqrt)
    knn_dist_list <- append(as.numeric(knn_dist_list), knn_dist_final)

  }
  top_k_neighbors <- order(knn_dist_list, decreasing = TRUE)[1:3]
  #now find the ring values then multiply the weights and divide by the total of the weights (6)
  ring_k_neighbors <- ((3 * target_data[top_k_neighbors[1]]) +
```

```
    ((2) * target_data[top_k_neighbors[2]]) + (1* target_data[top_k_neighbors[3]]))/6
return(ring_k_neighbors)

}

#call the knn.reg function using the correct variable/data
knn.reg(new_data = new_data, target_data = target_data, train_data = train_data, k = 3)
```

```
## [1] 11.83333
```

2.5 / use knn.reg function to predict rings from new data

```
#call the knn.reg function using the correct variable/data
knn.reg(new_data = new_data, target_data = target_data, train_data = train_data, k = 3)
```

```
## [1] 11.83333
```

2.6 / calculate MSE

```
predicted_knn_reg <- list()
# generate random sample from train_data
sample <- sample.int(n = nrow(train_data), size = floor(.25*nrow(train_data)))
test_data <- train_data[sample,]
test_rings <- target_data[sample]
```

```
#not evaluating because it took too long when knitting, so I saved to
#a .Rdata file instead before knitting
for (i in 1:nrow(test_data)){
  knn_val <- knn.reg(new_data = test_data[i,] , target_data = target_data, train_data = train_data)
  predicted_knn_reg <- append(as.numeric(predicted_knn_reg), knn_val)
}
```

```
load(file = 'predicted_knn_reg.Rdata')
error_total <- 0
#MSE calc
for (i in 1:length(predicted_knn_reg)){
  error_total <- error_total + (test_rings[i] - predicted_knn_reg[i])^2
}
MSE_knn <- sum(error_total)/length(predicted_knn_reg)
print(MSE_knn)
```

```
## [1] 21.58852
```

3 / Forecasting Future Sales Price

We obtained a data set containing 29580sales transactions for the years 2007 to 2019. The mean sales price for the entire time frame was 6.0973626×10^5 (sd $= 2.8170791 \times 10^5$).

Broken down by year, we have the following average sales prices per year:

As the graph below shows, the average sales price per year has been increasing.

Using a weighted moving average forecasting model that averages the prior 3 years (with weights of 4, 3, and 1), we predict next year's average sales price to be around 6.379666×10^5.