Practicum 1

Dr. Nishat Mohammad

02/03/2024

1 / Predicting Diabetes.

```
## 'data.frame':    768 obs. of  9 variables:
##  $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
##  $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
##  $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
```

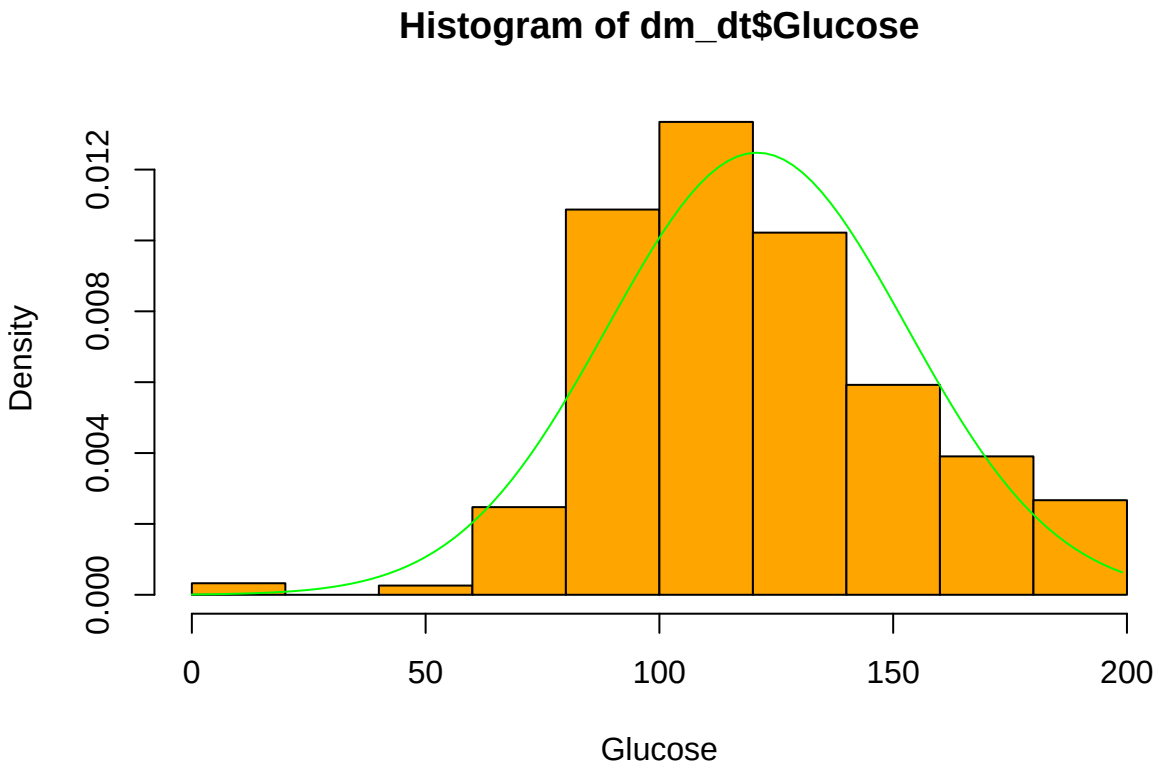
```
## [1] 768   9
```

```
##   Pregnancies        Glucose      BloodPressure    SkinThickness
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
##  Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
##  Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
##  3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
##     Insulin           BMI        DiabetesPedigreeFunction      Age
##  Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
##  1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
##  Median : 30.5   Median :32.00   Median :0.3725           Median :29.00
##  Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24
##  3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
##  Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
##     Outcome
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.349
##  3rd Qu.:1.000
##  Max.   :1.000
```

There is no Id Column in this data set. I found zero values which looked out of the ordinary for Glucose, BloodPressure, SkinThickness, Insulin, BMI. Glucose has a range for the blood concentration , it cannot be zero even after death. Insulin concentrations fluctuate with circadian rhythm and also cannot be zero even in type I Diabetes but for simplicity sake I will consider that at the time of compilation of this dataset for

some reason the researchers were not able to document close to 0 values of insulin for type I diabetes. Blood pressure also has a range and cannot be zero as well. SkinThickness can only be zero if there is no skin so I thought this was also not realistic. BMI cannot be zero as it is a measure of mass. There were zero values in Pregnancies which is possibility and not out of the ordinary. Outcome is zero for negative instances of the target variable. There are a total of 768 observations in total.

1.1 / Analysis of Data Distribution.



The glucose distribution appears reasonably normal although the left tail is slightly longer and narrower than the right tail. it is skewed to the right slightly and this is usually the case for real life data due to sample variability due to method and timing of sampling among many other discrepancies that bring a slight bias which can be overlooked. Severe skewness is however detrimental to statistical evaluations of analysis of the data.
Coming back to the summary of the data shown above we did have a couple of zero values under the glucose column, these are also contributing to the skewness slightly shifting to the right and it would be standard to impute these values and then look at the data distribution again. The data seems to have heavy outlier population on the right tail, this could imply they are cases with diabetes with uncontrolled blood glucose or some errors related to blood glucose estimation from sampling to testing and documentation processes.

The Shapiro-Wilk test result is “0.970103837977483” which shows that the data is normally distributed. The p-value is “1.98634754201744e-11”. which is pretty low and thus strong to support that the data is strongly normally distributed.

1.2 / Identification of Outliers.

```
# (5 pts) Add header ### 1.2 / Identification of Outliers. Identify any outliers for the columns using
#Which are your outliers for each column? What would you do? Summarize the results of your analysis and

# Get at the z scores

z <- as.data.frame(apply(dm_dt, 2, function(dm_dt) (abs(dm_dt-mean(dm_dt))/sd(dm_dt))))

cols = colnames(dm_dt)
for (col in cols){
  outlyr = dm_dt[,col][z[,col] > 2.5]
  print(paste("The outliers for the", col, "are:"))
  print(outlyr)
}
```

```
## [1] "The outliers for the Pregnancies are:"
##  [1] 13 13 13 15 17 13 14 13 13 14 13 13 13 13
## [1] "The outliers for the Glucose are:"
## [1] 0 0 0 0 0
## [1] "The outliers for the BloodPressure are:"
##  [1]   0   0   0   0   0   0 122   0   0   0   0   0   0   0   0   0   0   0
## [20]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [1] "The outliers for the SkinThickness are:"
## [1] 63 99
## [1] "The outliers for the Insulin are:"
##  [1] 543 846 495 485 495 478 744 370 680 402 375 545 465 415 579 474 480 600 440
## [20] 540 480 387 392 510
## [1] "The outliers for the BMI are:"
##  [1]  0.0  0.0  0.0  0.0 53.2 55.0  0.0 67.1 52.3 52.3 52.9  0.0  0.0 59.4  0.0
## [16]  0.0 57.3  0.0  0.0
## [1] "The outliers for the DiabetesPedigreeFunction are:"
##  [1] 2.288 1.441 1.390 1.893 1.781 1.400 1.321 2.329 1.318 1.353 1.391 1.476
## [13] 2.137 1.731 1.600 2.420 1.699 1.698 1.461 1.394
## [1] "The outliers for the Age are:"
##  [1] 69 65 66 63 65 63 65 67 72 81 63 67 66 64 67 66 70 68 69 66 63
## [1] "The outliers for the Outcome are:"
## integer(0)
```

From the code chunk above, The outliers in the data have been identified and printed out.

Z_score calculation: I took the columns of the data (using dm_dta, 2) and applied a function that calculates the zscore. this function subtracts the mean from the value, gets the absolute value and then divides by the standard deviation.

Outlier Identification: This was done using a for loop, in which each column in the data was iterated upon by its name. the value was checked if it was 2.5 sd away form the z-score, if yes, then the it was added to the `outlyr` object. These were printed based on the column name they fell under. Hope this explanation was ample to understand the code.

Looking at the outliers:

Pregnancies: have outliers ranging from 13 to 17, since we are considering women of at least 21 years, it would be impossible to have had that number of pregnancies in that short time. This can be imputed by

methods which will be subsequently discussed. Unless ofcurse we consisder that ther may be outliers in the age variable as well which we will look at subsequently

Glucose: the oultiers are zero values just as I had pointed out earlier. these need to be imputed as well.

Blood Pressure: The outliers are mainly zeros which are not compatible with realistic scenarios. 122 for diastolic blood pressure is also an outlier, it may be imputed as well to have a better analysis although it may be indicative of diastolic hypertension which is one of the signs of right sided heart failure. but that is not the focus of this data and analysis so we may go ahead to impute this.

Skin Thickness: The skin thickness of 63 and 99 are found to be outliers, they are indicative of excess subcutaneous fat and tell us about the nutritional category of the subjects in the study, these individuals may have high BMI and a very low insulin. We may take this off for this analysis.

Insulin and Diabetes Pedigree function: The outliers are many in these variables and will be imputed as well.

Age outliers are above 21 yrs and will require imputation. These outliers could account for those in pregnancies varaible.

Techniques for handling Outliers:

1) Deleting data points that are outliers from the data entirely is a way of handling that must be used with caution to avoid drastic reduction in the size of the data set.

2) Impute with the mean, median implies the replacement of outliers with the mean or the median, the choice of value to replace the outliers with is subjective.

3) The 10th quantile to replace the outliers at the lower end of the data and 90th percentile to replace the outliers at the other end.

With the next code chunk I will go ahead to remove the rows with outliers.

```
outlyr_preprocesd_dt <- dm_dt[!rowSums(z>2.5), ]
num_rows1 <- nrow(outlyr_preprocesd_dt)
summary(outlyr_preprocesd_dt)
```

```
##   Pregnancies        Glucose      BloodPressure    SkinThickness
##  Min.   : 0.000   Min.   : 44.0   Min.   : 24.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.00   1st Qu.: 0.00
##  Median : 3.000   Median :114.0   Median : 72.00   Median :23.00
##  Mean   : 3.686   Mean   :119.3   Mean   : 72.02   Mean   :21.04
##  3rd Qu.: 6.000   3rd Qu.:137.0   3rd Qu.: 80.00   3rd Qu.:32.00
##  Max.   :12.000   Max.   :198.0   Max.   :110.00   Max.   :60.00
##     Insulin           BMI        DiabetesPedigreeFunction      Age
##  Min.   :  0.00   Min.   :18.20   Min.   :0.0780           Min.   :21.00
##  1st Qu.:  0.00   1st Qu.:27.40   1st Qu.:0.2447           1st Qu.:24.00
##  Median : 45.00   Median :32.00   Median :0.3670           Median :29.00
##  Mean   : 71.42   Mean   :32.12   Mean   :0.4435           Mean   :32.11
##  3rd Qu.:125.25   3rd Qu.:36.12   3rd Qu.:0.5985           3rd Qu.:39.00
##  Max.   :360.00   Max.   :50.00   Max.   :1.2920           Max.   :62.00
##     Outcome
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3245
```

```
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

Now, 644 observations are left after removing outliers. It will be convenient to handle the missing values i.e. the zero values in some of the variables, I will do this in the next chunk of code. From the summary we can see the only columns with zero values now are the Pregnancies which is not out of the ordinary, the SkinThickness which I will consider missing values and have them imputed witht the median value for that varaible.

The Insulin variable is predominantly zeros, I have reservations about this considering that in recent years Type I Diabetes patients(insulin deficient) have been found to have minimal concentrations of insulin. I am tempted to assume at the time of compilation of this dataset these minimal concentrations of insulin were not detected by the researchers because if I am to impute insulin variable it will also be to a very large extent since most of the observations for this variable have 0 value. I considered checking the values of Outcome variable for insulin 0 in the code chunk below.

```
# check the outcome for insulin = 0
insulin_outcome_var <- outlyr_preprocesd_dt %>%
  filter(Insulin==0) %>%
  select(Outcome) %>%
  filter(Outcome==1)
  #filter(Outcome==0)
```

I found that about half of the cases with insulin 0 have outcome 1 and the rest have outcome 1. I am thus taking the zeros in insulin as missing values and will impute them with the median. Also, I found out that for imputation there is no limit of missingness unlike the case of removing the data from the the dataset entirely, so I feel more comfortable imputing the values for insulin and the skin thickness variables in the code chunk below.

```
# Impute skinThickness with median
# replace 0 with NA
outlyr_preprocesd_dt$SkinThickness[outlyr_preprocesd_dt$SkinThickness == 0] <-
  NA
#any(is.na(outlyr_preprocesd_dt$SkinThickness))
#head(outlyr_preprocesd_dt$SkinThickness)

# Get the median
mid_skinthickness <- median(outlyr_preprocesd_dt$SkinThickness, na.rm = TRUE)

# Replace the NA with median
ind_NA_ST <- which(is.na(outlyr_preprocesd_dt$SkinThickness)   == TRUE)
outlyr_preprocesd_dt$SkinThickness[ind_NA_ST] <- mid_skinthickness
head(outlyr_preprocesd_dt$SkinThickness)
```

```
## [1] 35 29 29 23 29 32
```

```
# Impute insulin with median
outlyr_preprocesd_dt$Insulin[outlyr_preprocesd_dt$Insulin == 0] <-
  NA
#any(is.na(outlyr_preprocesd_dt$Insulin))
#head(outlyr_preprocesd_dt$Insulin)

# Get the median
```

```r
mid_insulin <- median(outlyr_preprocesd_dt$Insulin, na.rm = TRUE)

# Replace the NA with median
ind_NA_i <- which(is.na(outlyr_preprocesd_dt$Insulin)   == TRUE)
outlyr_preprocesd_dt$Insulin[ind_NA_i] <- mid_insulin
head(outlyr_preprocesd_dt$Insulin)
```

```
## [1] 120 120 120  94 120  88
```

1.3 / Data Preparation.

```r
# (5 pts + 1 bonus) Add header ### 1.3 / Data Preparation. Normalize all numeric columns using z-score

scald_dt <- as.data.frame(scale(outlyr_preprocesd_dt[,1:(ncol(outlyr_preprocesd_dt)-1)]))
#head(scald_dt)
Outcome <- outlyr_preprocesd_dt$Outcome
# Add outcome
scald_dt <- cbind(scald_dt, Outcome)
head(scald_dt)
```

```
##   Pregnancies    Glucose BloodPressure SkinThickness    Insulin        BMI
## 1   0.7419492  0.9743387  -0.002100934    0.75375251 -0.1258414  0.2287961
## 2  -0.8614578 -1.1610800  -0.509476499    0.04164989 -0.1258414 -0.8500875
## 3   1.3833120  2.1606824  -0.678601687    0.04164989 -0.1258414 -1.3587041
## 4  -0.8614578 -1.0254978  -0.509476499   -0.67045272 -0.5965938 -0.6188982
## 6   0.4212678 -0.1103184   0.167024254    0.04164989 -0.1258414 -1.0042138
## 7  -0.2200950 -1.3983487  -1.862478005    0.39770120 -0.7052290 -0.1719321
##   DiabetesPedigreeFunction         Age Outcome
## 1                0.6883743  1.72999352       1
## 2               -0.3471847 -0.10736441       0
## 3                0.8572154 -0.01066136       1
## 4               -1.0375573 -1.07439490       0
## 6               -0.9099884 -0.20406746       0
## 7               -0.7336432 -0.59087965       1
```

The table above shows a few values of the data after standardization with all the values in the same range. I have used the `scale()` function to carry this step out. This step is important because knn is sensitive to the scale of values in the data and normalizing the data will prevent bias since all the features are equally learnt by the model and a fair comparison between them occurs making it easier to weigh the importance of these features.

1.4 / Sampling Training and Validation Data.

```r
# (5 pts) Add header ### 1.4 / Sampling Training and Validation Data. Randomize (shuffle) the data and
# get the stratified data
Outcome0 <- scald_dt[scald_dt$Outcome == 0,]
Outcome1 <- scald_dt[scald_dt$Outcome == 1,]

out_neg <- sample(rownames(Outcome0), 0.20 * nrow(Outcome0))
```

```r
out_pos <- sample(rownames(Outcome1), 0.20 * nrow(Outcome1))
#head(scald_dt[out_neg,])
#head(scald_dt[out_pos,])

# Create the validation set
validatn_set <- rbind(scald_dt[out_neg,],scald_dt[out_pos,])
nrow(validatn_set)
```

```
## [1] 128
```

```r
#head(validatn_set)

# create the training set
train_set <- scald_dt[!(row.names(scald_dt) %in% c(out_neg,out_pos)),]
nrow(train_set)
```

```
## [1] 516
```

```r
head(train_set)
```

```
##    Pregnancies    Glucose BloodPressure SkinThickness    Insulin        BMI
## 1    0.7419492  0.9743387  -0.002100934    0.75375251 -0.1258414  0.2287961
## 2   -0.8614578 -1.1610800  -0.509476499    0.04164989 -0.1258414 -0.8500875
## 3    1.3833120  2.1606824  -0.678601687    0.04164989 -0.1258414 -1.3587041
## 4   -0.8614578 -1.0254978  -0.509476499   -0.67045272 -0.5965938 -0.6188982
## 6    0.4212678 -0.1103184   0.167024254    0.04164989 -0.1258414 -1.0042138
## 11   0.1005864 -0.3136916   1.689150948    0.04164989 -0.1258414  0.8453010
##    DiabetesPedigreeFunction         Age Outcome
## 1                 0.6883743  1.72999352       1
## 2                -0.3471847 -0.10736441       0
## 3                 0.8572154 -0.01066136       1
## 4                -1.0375573 -1.07439490       0
## 6                -0.9099884 -0.20406746       0
## 11               -0.9475087 -0.20406746       0
```

```r
# cross checking for my understanding
nrow(validatn_set) / sum(nrow(validatn_set), nrow(train_set))
```

```
## [1] 0.1987578
```

In this code chunk I have stratified the data based on outcome classes. created the validation set from 20% of both classes of Outcome. the rest of the data is the training data.

1.5 / Predictive Modeling.

```r
# (5 pts) Add header ### 1.5 / Predictive Modeling. Apply the knn function from the class package with

#Pregnancies    Glucose BloodPressure   SkinThickness   Insulin BMI DiabetesPedigreeFunction    Age
#4   178 82   32   NA   37   0.602    54
```

```r
# Define the given data
nw_data <- c(4, 178, 82, 32, NA, 37, 0.602, 54)

#Replace the NA with median for insulin
nw_ind_NA_i <- which(is.na(nw_data)   == TRUE)
nw_data[nw_ind_NA_i] <- mid_insulin
#nw_data

# Standardize the new data
nw_scald_dt <- scale(nw_data)
nw_scald_dt
```

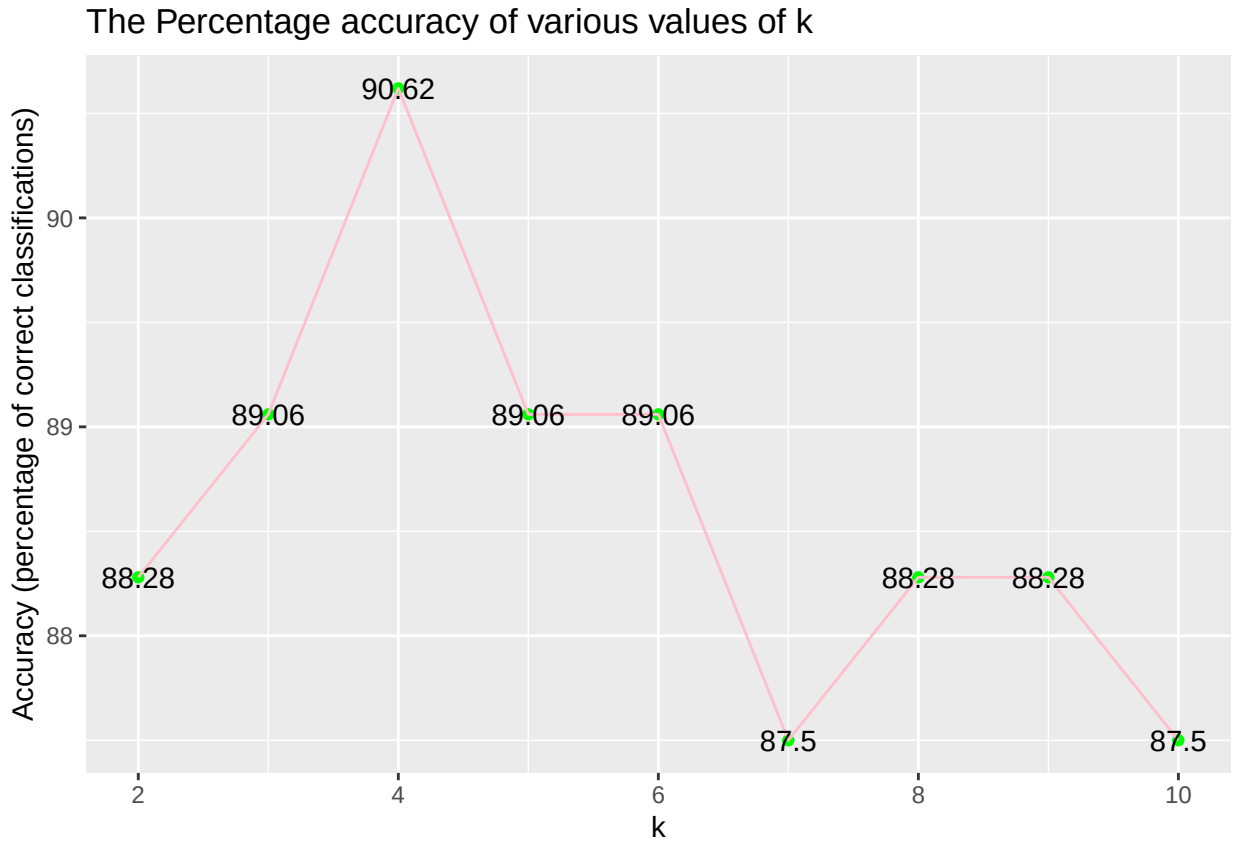
```
##            [,1]
## [1,] -0.9768680
## [2,]  1.8822458
## [3,]  0.3048037
## [4,] -0.5167807
## [5,]  0.9292079
## [6,] -0.4346223
## [7,] -1.0327029
## [8,] -0.1552836
## attr(,"scaled:center")
## [1] 63.45025
## attr(,"scaled:scale")
## [1] 60.85802
```

```r
# apply knn form class package: with the trainin set, test data as the new data and the k as 5
library(class)
get_knn_outcome <- knn(train = train_set[-9],
                       test = t(nw_scald_dt),
                       cl = train_set$Outcome, k=5)
```

The outcome of the prediction using knn is "1". which is positive.

1.6 / Model Accuracy.

```
##   k_accuracy  k
## 1      88.28  2
## 2      89.06  3
## 3      90.62  4
## 4      89.06  5
## 5      89.06  6
## 6      87.50  7
## 7      88.28  8
## 8      88.28  9
## 9      87.50 10
```



To generate the plot above I calculated the accuracies of for values of k = 2 to k = 10 via a for loop which also calculated the accuracies in percentages.

We can see the accuracy increases with increase in k until k= 5. Thereafter it fluctuates a little but does not go higher than 91.41% accuracy for increase in k.

From this plot, the best choice of k is 5, because if we take a smaller value of k the accuracy depreciates and if we take a higher value of k the accuracy does not go above that at k=5. Also, keeping in mind that the knn works better with smaller values of k, k = 5 is the best.

2 / Predicting Age of Abalones using Regression kNN

```
# This problem requires analysis of a data set about abalonesLinks to an external site. -- a type of ma

#While you may download the data set for inspection, be sure to load the data into R from its URL.

#The columns of the data are:

#Sex / nominal / -- / M, F, and I (infant).
#Length / continuous / mm / Longest shell measurement
#Diameter / continuous / mm / perpendicular to length.
#Height / continuous / mm / with meat in shell.
#Whole weight / continuous / grams / whole abalone
#Shucked weight / continuous / grams / weight of meat
#Viscera weight / continuous / grams / gut weight (after bleeding)
```

```r
#Shell weight / continuous / grams / after being dried
#Rings / integer / -- / +1.5 gives the age in years

#Instructions:

#Add the level 2 header ## 2 / Predicting Age of Abalones using Regression kNN to your notebook. For ea
url_ab <- "https://s3.us-east-2.amazonaws.com/artificium.us/datasets/abalone.csv"

ab_data <- read.csv(url_ab, stringsAsFactors = FALSE)
str(ab_data)
```

```
## 'data.frame':    4178 obs. of  9 variables:
##  $ Length       : num  0.455 0.35 0.53 0.44 0.33 0.425 0.53 0.545 0.475 0.55 ...
##  $ Diameter     : num  0.365 0.265 0.42 0.365 0.255 0.3 0.415 0.425 0.37 0.44 ...
##  $ Height       : num  0.095 0.09 0.135 0.125 0.08 0.095 0.15 0.125 0.125 0.15 ...
##  $ ShuckedWeight: num  0.2245 0.0995 0.2565 0.2155 0.0895 ...
##  $ VisceraWeight: num  0.101 0.0485 0.1415 0.114 0.0395 ...
##  $ ShellWeight  : num  0.15 0.07 0.21 0.155 0.055 0.12 0.33 0.26 0.165 0.32 ...
##  $ WholeWeight  : num  0.514 0.226 0.677 0.516 0.205 ...
##  $ NumRings     : int  15 7 9 10 7 8 20 16 9 19 ...
##  $ Sex          : chr  "M" "M" "F" "F" ...
```

```r
dim(ab_data)
```

```
## [1] 4178    9
```

```r
any(is.na(ab_data))
```

```
## [1] FALSE
```

```r
head(ab_data)
```

```
##   Length Diameter Height ShuckedWeight VisceraWeight ShellWeight WholeWeight
## 1  0.455    0.365  0.095        0.2245        0.1010       0.150      0.5140
## 2  0.350    0.265  0.090        0.0995        0.0485       0.070      0.2255
## 3  0.530    0.420  0.135        0.2565        0.1415       0.210      0.6770
## 4  0.440    0.365  0.125        0.2155        0.1140       0.155      0.5160
## 5  0.330    0.255  0.080        0.0895        0.0395       0.055      0.2050
## 6  0.425    0.300  0.095        0.1410        0.0775       0.120      0.3515
##   NumRings Sex
## 1       15   M
## 2        7   M
## 3        9   F
## 4       10   F
## 5        7   I
## 6        8   I
```

2.1 / Picking useful data for further work:

```
#(0 pts) Save the values of the "Rings" column in a separate vector called target_data and then also cr


# Create a vector for target_data
target_data <- ab_data$NumRings

# Create a dataframe without NumRings
train_data <- subset(ab_data, select= -NumRings)
str(train_data)
```

```
## 'data.frame':    4178 obs. of  8 variables:
##  $ Length       : num  0.455 0.35 0.53 0.44 0.33 0.425 0.53 0.545 0.475 0.55 ...
##  $ Diameter     : num  0.365 0.265 0.42 0.365 0.255 0.3 0.415 0.425 0.37 0.44 ...
##  $ Height       : num  0.095 0.09 0.135 0.125 0.08 0.095 0.15 0.125 0.125 0.15 ...
##  $ ShuckedWeight: num  0.2245 0.0995 0.2565 0.2155 0.0895 ...
##  $ VisceraWeight: num  0.101 0.0485 0.1415 0.114 0.0395 ...
##  $ ShellWeight  : num  0.15 0.07 0.21 0.155 0.055 0.12 0.33 0.26 0.165 0.32 ...
##  $ WholeWeight  : num  0.514 0.226 0.677 0.516 0.205 ...
##  $ Sex          : chr  "M" "M" "F" "F" ...
```

Looking at the train_data, we confirm that the NumRings column has been removed.

2.2 Categorical Variable Encoding:

```
#(5 pts) Encode all categorical columns using an encoding scheme of your choice but document (in markdo

# Categorical variable encoding
train_data$Sex <- ifelse(train_data$Sex == "M", 1, 0)
```

The Categorical variable I found was the “Sex” column. I thus assigned 1 to M and anything other than that 0, so F will be 0.
Reason for choice:
The Sex variable is clearly divide into 2 groups; M and F. They are not related to each other and they are the direct opposite of each other (answering the yes or no question). of course The Sex variable is not numeric and does not follow any sequence like continuous variables are. thus I chose this as a categorical variable. All other variables look like continuous variables. Thus I chose binary coding here with M as 1 and F as 0. Other forms of encoding such as One-Hot and integral encoding do not fit in this context since the Sex variable is nominal categorical variable with only two categories.

2.3 Data Normalization:

```
# (5 pts) Normalize all the columns in train_data using min-max normalization.

# Create a function for scaling
scalr <- function(x) {
(x - min(x)) / (max(x) - min(x))
}
```

```r
# Apply scaling
train_data <- as.data.frame(lapply(train_data, scalr))
#str(train_data)
head(train_data)
```

```
##      Length  Diameter     Height ShuckedWeight VisceraWeight ShellWeight
## 1 0.5135135 0.5210084 0.08407080    0.15030262    0.13232390  0.14798206
## 2 0.3716216 0.3529412 0.07964602    0.06624075    0.06319947  0.06826109
## 3 0.6148649 0.6134454 0.11946903    0.17182246    0.18564845  0.20777280
## 4 0.4932432 0.5210084 0.11061947    0.14425017    0.14944042  0.15296462
## 5 0.3445946 0.3361345 0.07079646    0.05951580    0.05134957  0.05331340
## 6 0.4729730 0.4117647 0.08407080    0.09414929    0.10138249  0.11808670
##   WholeWeight Sex
## 1  0.18133522   1
## 2  0.07915707   1
## 3  0.23906499   0
## 4  0.18204356   0
## 5  0.07189658   0
## 6  0.12378254   0
```

Here the data has been normalized using the min max method and the first few values can be seen above.

2.4 KNN Modeling:

```r
# (15 pts) Build (write) a function called knn.reg that implements a regression version of kNN that ave
#It must have the following signature: knn.reg (new_data, target_data, train_data, k where new_data is
#This question requires that you write your own version of kNN and not use one from a package.


# Create the knn.reg() function
# Get weighted average
wtd_av <- function(closest, k) {
  wt_fac1 <- closest[1] * 2
  wt_fac2 <- closest[2] * 1.5
  wtd_sum <- sum(closest[3:k]) + wt_fac1 + wt_fac2
  wts_sum <- (k - 2) + 2 + 1.5
  wtd_avg <- wtd_sum/wts_sum
  return(wtd_avg)
}
# Create a function for the distance between points
get_euc_dist <- function(v1, new_data) sqrt(sum((v1 - new_data)^2))

# Create the knn.reg function
knn.reg <- function(new_data, target_data, train_data, k){
  #get the euclidean distance using the function
  euc_dists <- apply(train_data, 1, FUN=function(x1) get_euc_dist(x1, new_data))
  # assign the names form the target data
  names(euc_dists) <- target_data
  # sort to get the nearest ones close by
  sort_euc_dist <- sort(euc_dists)
  # get the neighbours
```

```
  kns <- sort_euc_dist[1:k]
  # get the weighted average using the function
  weighted_avg <- wtd_av(as.numeric(names(kns)), k)

  return(as.integer(weighted_avg))
}
```

This chunk of code shows three functions:

1) The wtd_av function encompases the calculation of weigted averages with the nearest having a weight of 2 the second nearest having a weight of 1.5 and the rest weighted by wiegth factor of 1. the sum of the product of the value and the weight factor is divided by the total weight factors.

2) The get_euc_dist function calculates the euclidean distance between points. these 2 functions are prerequisites for the knn.reg function requested.

3) the knn.reg function is one which takes a set of new data, calculates the euclidean distance between the points by using the get_euc_dist() function, assigns names/labels based on the target_data created earlier from the dataset. sorts the distances to keep the closest nearest, then predicts the knn by using the wtd_avg() function.

2.5 Forecasting with a Given Data:

```
# (5 pts) Forecast the number of Rings of this new abalone using your regression kNN using k = 3:
#Sex: F | Length: 0.82 | Diameter: 0.491 | Height: 0.361 | Whole weight: 0.5538 | Shucked weight: 0.324

# new data
nw_data <- scalr(c(0, 0.82, 0.491, 0.361, 0.5538, 0.3245, 0.0921, 0.305))
k <- 3
forcasted_rings <- knn.reg(nw_data, target_data, train_data, k)
#forcasted_rings
```

In this forcasting, the knn.reg() function is fed the given data, and the value of k as 3 to forecast and the forecasted number of rings are “11”.

2.6 Root Mean Squared:

(5 pts) Calculate the Root Mean Squared Error (RMSE) using a random sample of 20% of the data set as test data.

```
# test set
test_set_labs <- sample(rownames(ab_data), 0.20 * nrow(ab_data))
test_dt_x <- train_data[test_set_labs,]
test_dt_y <- target_data[as.numeric(test_set_labs)]
#head(test_dt_x)
#head(test_dt_y)


# Train set
train_dt_x <- train_data[!(row.names(train_data) %in% test_set_labs),]
train_dt_y <- target_data[-as.numeric(test_set_labs)]
```

```r
#head(train_dt_x)
#head(train_dt_y)


# Knn
forcasted_number_rings <- apply(test_dt_x, 1, FUN=function(x1) knn.reg(x1, train_dt_y,train_dt_x, k))
#head(forcasted_number_rings)

# Get MSE
mse <- sum((test_dt_y - forcasted_number_rings)^2) / length(test_dt_y)

# Get RMSE
rmse <- sqrt(mse)
```

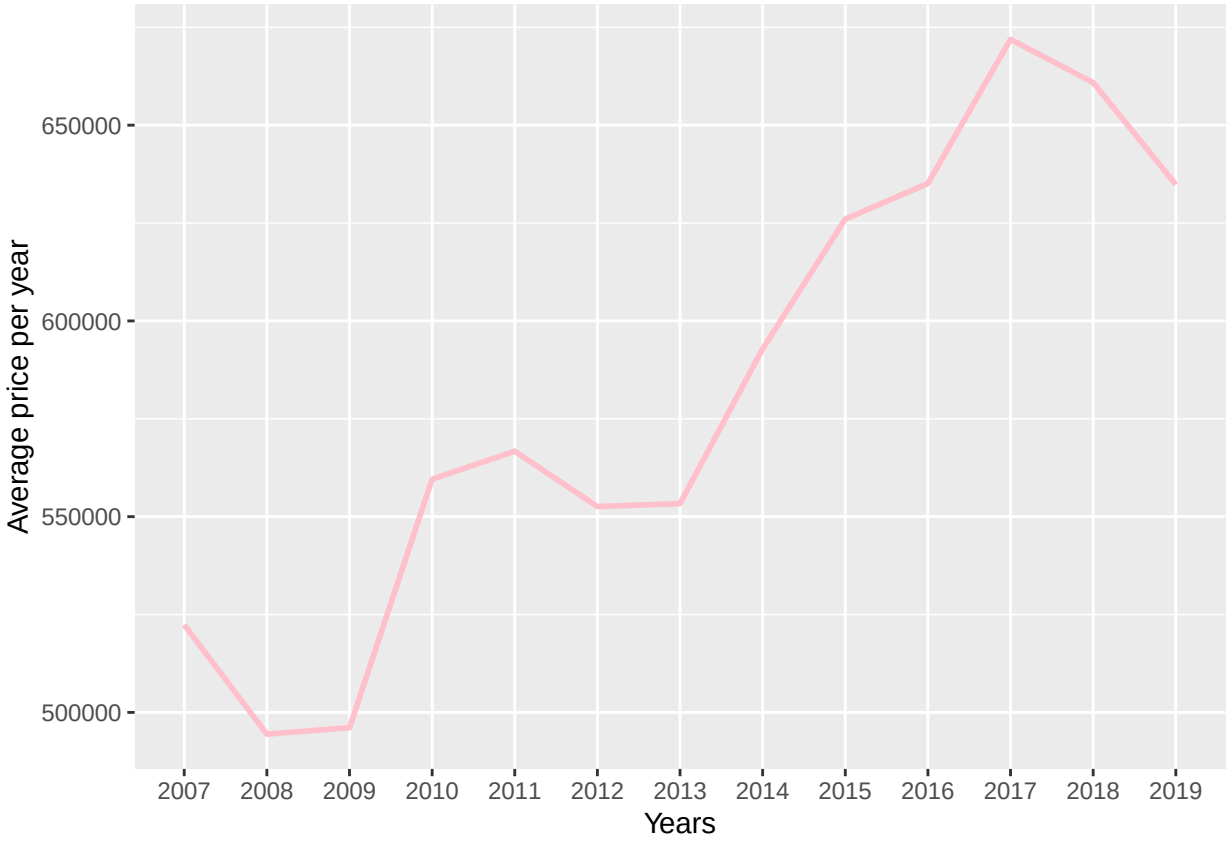
The Code chunk above has picked up a random 20% of the data for the testing set and the rest for training set. then forecasted using the knn.reg() function. After this the mse is gotten and its square root is taken to get the RMSE which is "2.49526497698926".

3 / Forecasting Future Sales Price

We obtained a data set containing "29550" sales transactions for the years "2007" to "2019". The mean sales price for the entire time frame was "609804.725414552"(sd = "281704.520009819").

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

years	average_price_per_year
2007	522377.2
2008	494447.1
2009	496092.0
2010	559564.8
2011	566715.1
2012	552575.4
2013	553330.3
2014	592861.8
2015	626014.7
2016	635097.4
2017	671921.1
2018	660875.2
2019	634830.9



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 “649233.777556601”.