KNN

Objective: to predict heart disease in patients.

Dataset: processed.cleveland.data from https://archive.ics.uci.edu/ml/datasets/Heart+Disease

For this assignment I prepared the environment, loaded libraries (class, gmodels, and caret) ad loaded the data in the data frame using the following code:

```
> #MSDS680 Week 2 Assignment: KNN  Weakly,Natalia
>
> rm(list=ls()) #Clear the environment
> setwd("E:/Dropbox/RU DataScience/MSDS680/Week2/Assignment") #Set working directory for the assi
gnment
> getwd() #Check working directory
[1] "E:/Dropbox/RU DataScience/MSDS680/Week2/Assignment"
>
> #Load libraries
>
> library("class")
> library("gmodels")
> library("caret")
> data1 <- read.table("processed.cleveland.data", header=FALSE, sep=",", stringsAsFactors = FALSE
)
```

To make sure that the data was loaded correctly, I looked at the first few rows using head() and displayed the structure of the data frame using str():

```
> #Check that the data was loaded correctly
> head(data1)
  V1 V2 V3  V4  V5 V6 V7  V8 V9 V10 V11 V12 V13 V14
1 63  1  1 145 233  1  2 150  0 2.3   3 0.0 6.0   0
2 67  1  4 160 286  0  2 108  1 1.5   2 3.0 3.0   2
3 67  1  4 120 229  0  2 129  1 2.6   2 2.0 7.0   1
4 37  1  3 130 250  0  0 187  0 3.5   3 0.0 3.0   0
5 41  0  2 130 204  0  2 172  0 1.4   1 0.0 3.0   0
6 56  1  2 120 236  0  0 178  0 0.8   1 0.0 3.0   0
> str(data1)
'data.frame':	303 obs. of  14 variables:
 $ V1 : num  63 67 67 37 41 56 62 57 63 53 ...
 $ V2 : num  1 1 1 1 0 1 0 0 1 1 ...
 $ V3 : num  1 4 4 3 2 2 4 4 4 4 ...
 $ V4 : num  145 160 120 130 130 120 140 120 130 140 ...
 $ V5 : num  233 286 229 250 204 236 268 354 254 203 ...
 $ V6 : num  1 0 0 0 0 0 0 0 0 1 ...
 $ V7 : num  2 2 2 0 2 0 2 0 2 2 ...
 $ V8 : num  150 108 129 187 172 178 160 163 147 155 ...
 $ V9 : num  0 1 1 0 0 0 0 1 0 1 ...
 $ V10: num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ V11: num  3 2 2 3 1 1 3 1 2 3 ...
 $ V12: chr  "0.0" "3.0" "2.0" "0.0" ...
 $ V13: chr  "6.0" "3.0" "7.0" "3.0" ...
 $ V14: int  0 2 1 0 0 0 3 0 2 1 ...
```

It appeared that the data was loaded correctly. Next, for convenience purposes I added labels to the column names so they corresponded to the description of the data set:

```
 > #Name the columns
> colnames(data1) <- c("age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach", "exan
g", "oldpeak", "slope", "ca", "thal", "num")
> str(data1)
'data.frame':	303 obs. of  14 variables:
 $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp      : num  1 4 4 3 2 2 4 4 4 4 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg : num  2 2 2 0 2 0 2 0 2 2 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ slope   : num  3 2 2 3 1 1 3 1 2 3 ...
 $ ca      : chr  "0.0" "3.0" "2.0" "0.0" ...
 $ thal    : chr  "6.0" "3.0" "7.0" "3.0" ...
 $ num     : int  0 2 1 0 0 0 3 0 2 1 ...
```

Next, I grouped the target variable (num) into two classes: 0, meaning no heart disease, and 1, that included all levels above 0, meaning presence of heart disease.

```
> #Group target variable into 2 classes
> data1$num[data1$num >0] = 1
> str(data1)
'data.frame':	303 obs. of  14 variables:
 $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp      : num  1 4 4 3 2 2 4 4 4 4 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg : num  2 2 2 0 2 0 2 0 2 2 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ slope   : num  3 2 2 3 1 1 3 1 2 3 ...
 $ ca      : chr  "0.0" "3.0" "2.0" "0.0" ...
 $ thal    : chr  "6.0" "3.0" "7.0" "3.0" ...
 $ num     : num  0 1 1 0 0 0 1 0 1 1 ...
```

Next, I proceeded with data type conversions. Num, as the target variable, needed to be converted to factors, variable ca and thal initially loaded as char data type and needed to be converted to numeric.

```
> #Group target variable into 2 classes
> data1$num[data1$num >0] = 1
> str(data1)
'data.frame':	303 obs. of  14 variables:
 $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp      : num  1 4 4 3 2 2 4 4 4 4 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg : num  2 2 2 0 2 0 2 0 2 2 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : num  0 1 1 0 0 0 1 0 1 1 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ slope   : num  3 2 2 3 1 1 3 1 2 3 ...
 $ ca      : chr  "0.0" "3.0" "2.0" "0.0" ...
 $ thal    : chr  "6.0" "3.0" "7.0" "3.0" ...
 $ num     : num  0 1 1 0 0 0 1 0 1 1 ...
```

Since not all values (ex., question mark and empty fields) from the initial data set were able to be converted to numeric, NA were automatically added in these cells. To find missing values I used the following:

```
 >###Missing values detection
>#Total number of missing values
> sum(is.na(data1))
[1] 6
>
> sum(is.na(data1$thal)) #missing thal values
[1] 2
> sum(is.na(data1$ca))   #missing ca values
[1] 4
>
> #list rows of data that have missing values
> data1[!complete.cases(data1),]
    age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal num
88   53   0  3      128  216   0       2     115     0     0.0     1  0   NA   0
167  52   1  3      138  223   0       0     169     0     0.0     1 NA    3   0
193  43   1  4      132  247   1       2     143     1     0.1     2 NA    7   1
267  52   1  4      128  204   1       0     156     1     1.0     2  0   NA   1
```

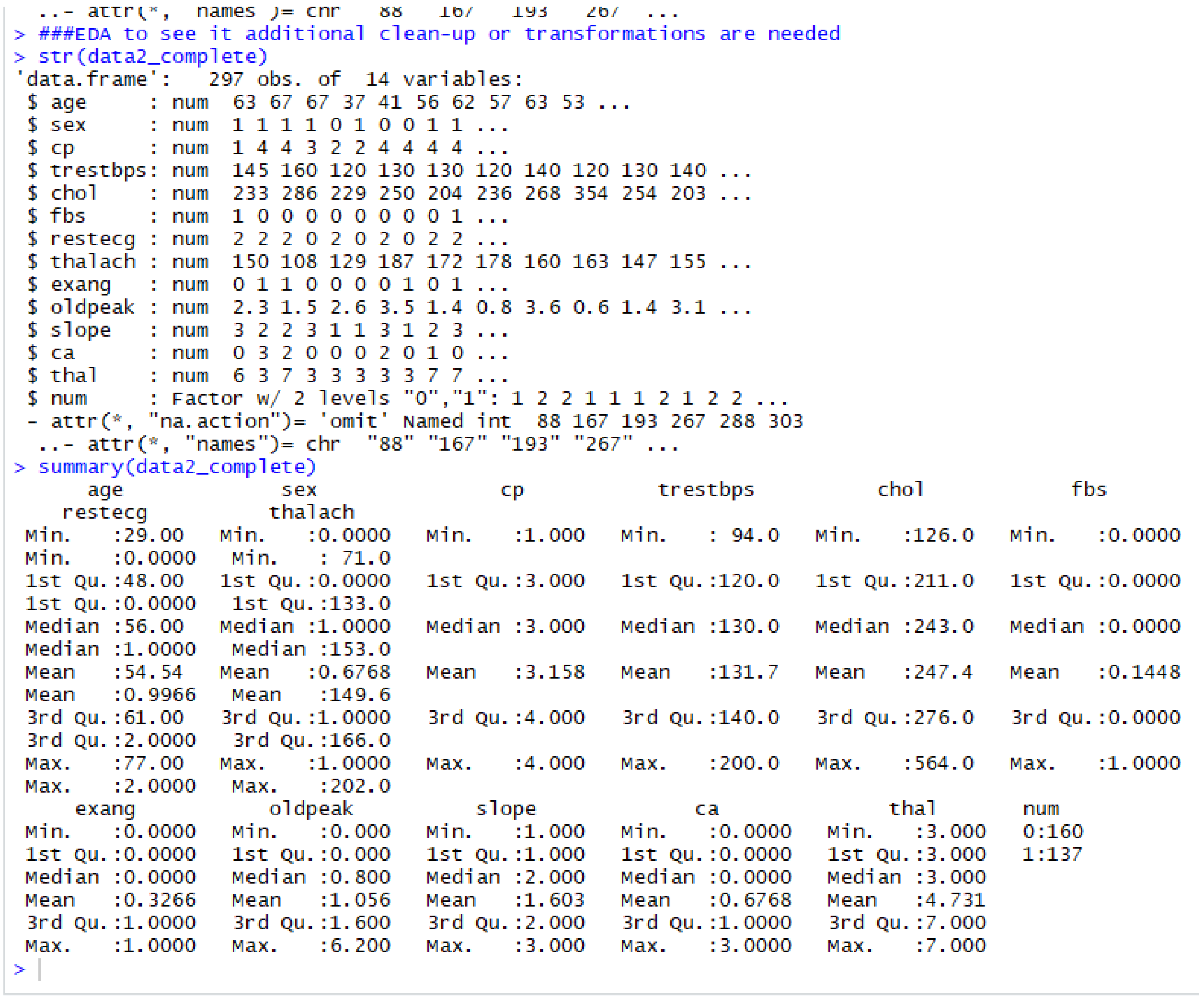
```
288   58    1   2       125   220    0        0      144      0      0.4      2 NA     7    0
303   38    1   3       138   175    0        0      173      0      0.0      1 NA     3    0
>
> #check number of heart disease and no heart disease diagnosis for the target variable
> table(data1$num)

  0   1
164 139
```

There were a total of 6 missing values: 4 in the ca column and 2 in the thal column. It affected 4 observations with negative heart disease diagnosis and 2 observations with positive heart disease diagnosis. Since there is no reliable and meaningful way to impute these values (categorical variable ca- for number of major vessels (0-3) colored by fluoroscopy and categorical variable thal for type of defect) and the proportions between no diagnosis and positive diagnosis observations in the dataset would not be considerably skewed, I decided just to delete the records with missing values. I used the following code:

```
> #remove incomplete records from the data set
> data2_complete <- na.omit(data1)
> sum(is.na(data2_complete))
[1] 0
> str(data2_complete) #This is the full DF before transformations
'data.frame':	297 obs. of  14 variables:
 $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp      : num  1 4 4 3 2 2 4 4 4 4 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg : num  2 2 2 0 2 0 2 0 2 2 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ slope   : num  3 2 2 3 1 1 3 1 2 3 ...
 $ ca      : num  0 3 2 0 0 0 2 0 1 0 ...
 $ thal    : num  6 3 7 3 3 3 3 3 7 7 ...
 $ num     : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
 - attr(*, "na.action")= 'omit' Named int  88 167 193 267 288 303
  ..- attr(*, "names")= chr  "88" "167" "193" "267" ...
```

The resulting data2_complete data set included all full record from the initial dataset. To see what potential transformations might be needed I did some exploratory data analysis.

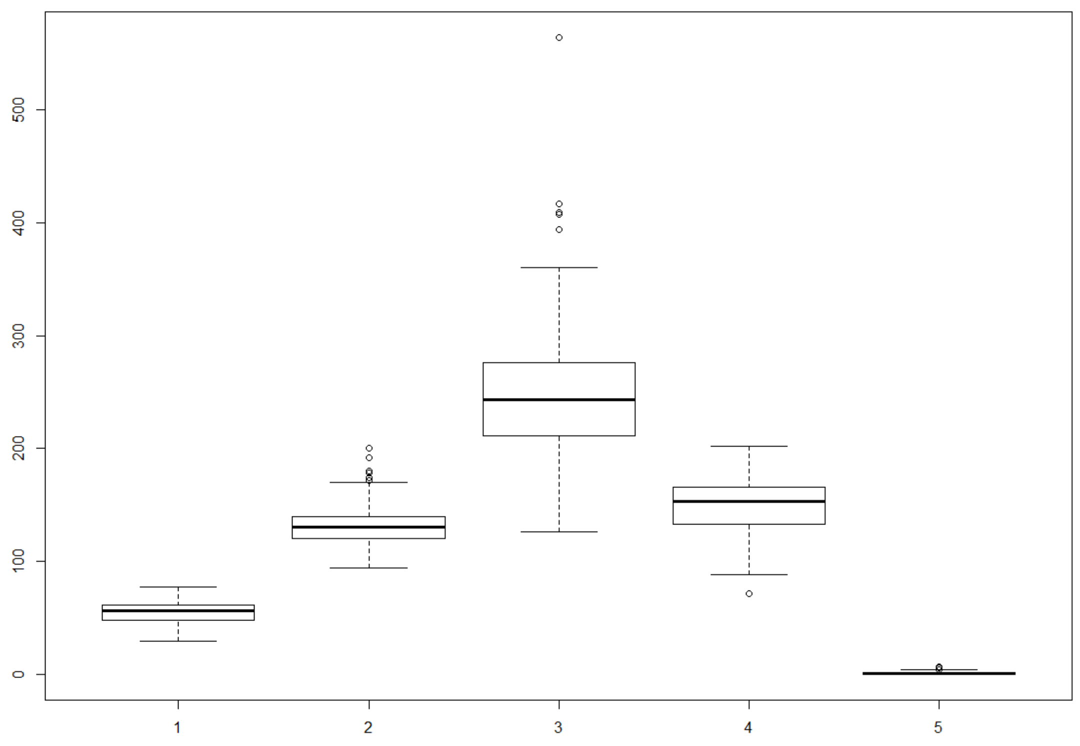


```
 ..- attr(*, "names")= chr  "88" "167" "193" "267" ...
> ###EDA to see it additional clean-up or transformations are needed
> str(data2_complete)
'data.frame':	297 obs. of  14 variables:
 $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp      : num  1 4 4 3 2 2 4 4 4 4 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg : num  2 2 2 0 2 0 2 0 2 2 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ slope   : num  3 2 2 3 1 1 3 1 2 3 ...
 $ ca      : num  0 3 2 0 0 0 2 0 1 0 ...
 $ thal    : num  6 3 7 3 3 3 3 3 7 7 ...
 $ num     : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
 - attr(*, "na.action")= 'omit' Named int  88 167 193 267 288 303
  ..- attr(*, "names")= chr  "88" "167" "193" "267" ...
> summary(data2_complete)
      age             sex               cp           trestbps          chol            fbs        
    restecg          thalach     
 Min.   :29.00   Min.   :0.0000   Min.   :1.000   Min.   : 94.0   Min.   :126.0   Min.   :0.0000  
 Min.   :0.0000   Min.   : 71.0  
 1st Qu.:48.00   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:120.0   1st Qu.:211.0   1st Qu.:0.0000  
 1st Qu.:0.0000   1st Qu.:133.0  
 Median :56.00   Median :1.0000   Median :3.000   Median :130.0   Median :243.0   Median :0.0000  
 Median :1.0000   Median :153.0  
 Mean   :54.54   Mean   :0.6768   Mean   :3.158   Mean   :131.7   Mean   :247.4   Mean   :0.1448  
 Mean   :0.9966   Mean   :149.6  
 3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:140.0   3rd Qu.:276.0   3rd Qu.:0.0000  
 3rd Qu.:2.0000   3rd Qu.:166.0  
 Max.   :77.00   Max.   :1.0000   Max.   :4.000   Max.   :200.0   Max.   :564.0   Max.   :1.0000  
 Max.   :2.0000   Max.   :202.0  
     exang           oldpeak          slope             ca              thal        num    
 Min.   :0.0000   Min.   :0.000   Min.   :1.000   Min.   :0.0000   Min.   :3.000   0:160  
 1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:1.000   1st Qu.:0.0000   1st Qu.:3.000   1:137  
 Median :0.0000   Median :0.800   Median :2.000   Median :0.0000   Median :3.000          
 Mean   :0.3266   Mean   :1.056   Mean   :1.603   Mean   :0.6768   Mean   :4.731          
 3rd Qu.:1.0000   3rd Qu.:1.600   3rd Qu.:2.000   3rd Qu.:1.0000   3rd Qu.:7.000          
 Max.   :1.0000   Max.   :6.200   Max.   :3.000   Max.   :3.0000   Max.   :7.000          
> 
```

The above result showed from str() and summary() commands showed no significant problems with the data. At the same time, they indicated that categorical variable will require substitution with dummy variables to exclude possibility of arithmetical operations (ex, 0.5 as a sex type) and that the continuous variable variables have significantly different ranges.

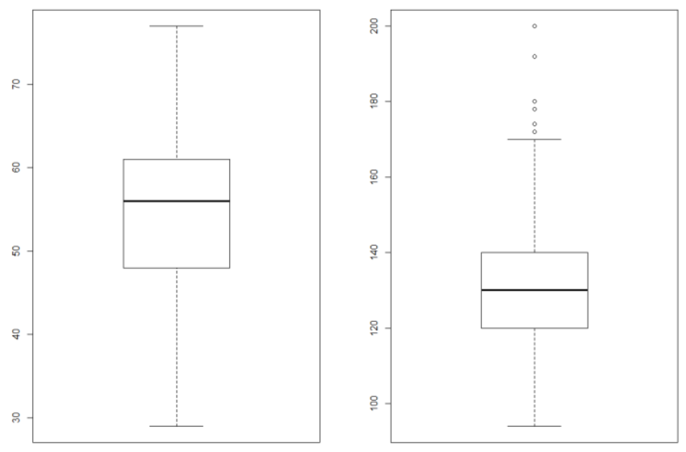
Side-by-side box plots for continuous variables visually confirmed differences in scales.

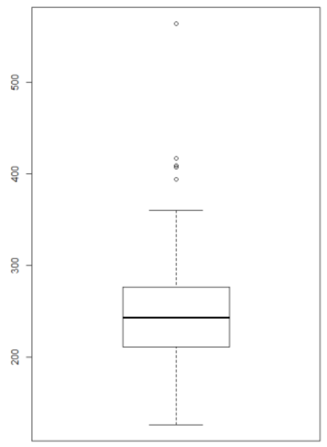
```
> #Side-by-side box plots
> boxplot(data2_complete$age,data2_complete$trestbps, data2_complete$chol, data2_complete$thalac
h, data2_complete$oldpeak)
```

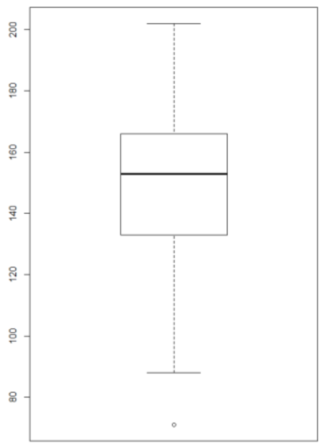


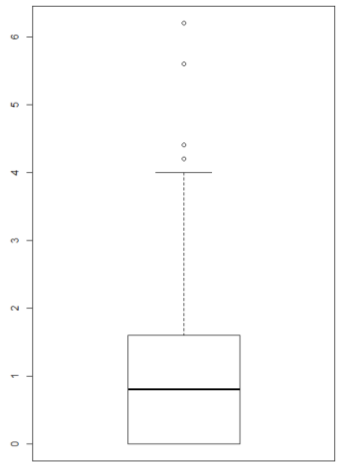
I also looked at boxplots and histograms for each individual continuous variable.

```
> #Box plots for each continuolus variable
> par(mfrow=c(1,2)) #Display 2 graphs in row
> boxplot(data2_complete$age)
> boxplot(data2_complete$trestbps)
> boxplot(data2_complete$chol)
> boxplot(data2_complete$thalach)
> boxplot(data2_complete$oldpeak)
```



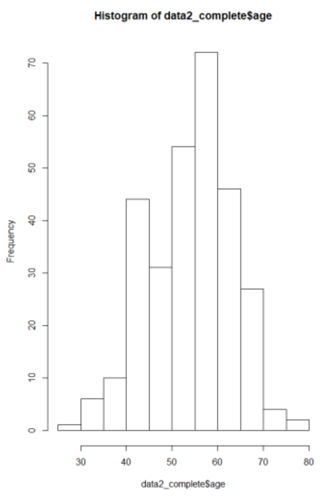


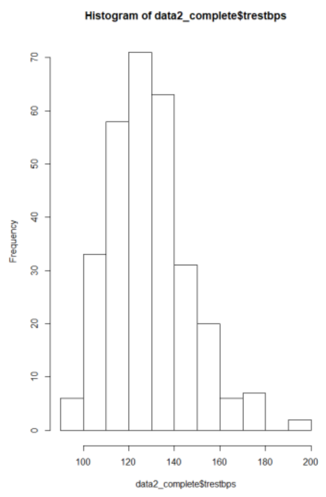


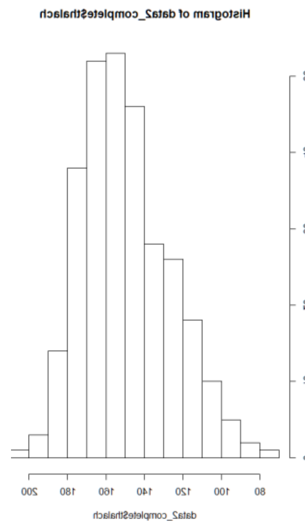


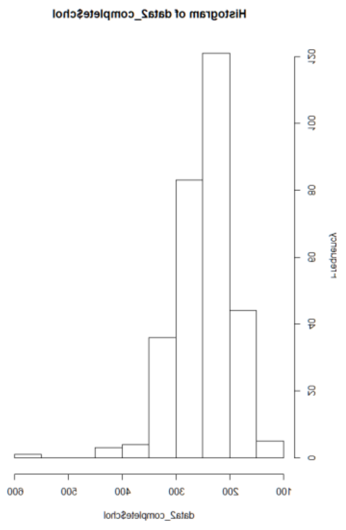
```
> #Histograms for all continuous variables
> hist(data2_complete$age)
> hist(data2_complete$trestbps)
> hist(data2_complete$chol)
> hist(data2_complete$thalach)
> hist(data2_complete$oldpeak)
```

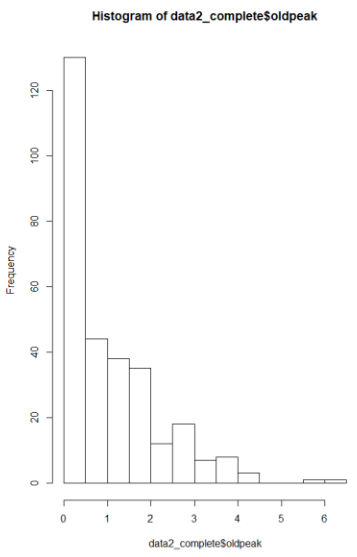
KNN


Histogram of data2_complete$age
Frequency
data2_complete$age


Histogram of data2_complete$trestbps
Frequency
data2_complete$trestbps


Histogram of data2_complete$thalach
Frequency
data2_complete$thalach


Histogram of data2_complete$chol
Frequency
data2_complete$chol


Histogram of data2_complete$oldpeak
Frequency
data2_complete$oldpeak

This analysis did not show any trends that would prevent us from using KNN algorithm for classification.

Next step – to code categorical variables as dummy variables. First, I created a data frame with 9 variables (8 categorical variables and the target num variable).

```
> #convert categorical variables to dummy variable
> #create a df with variables that need to be converted to dummy variables
>
> data3_dummies <- data2_complete[ , -1]
> data3_dummies$trestbps <- NULL
> data3_dummies$chol <- NULL
> data3_dummies$thalach <- NULL
> data3_dummies$oldpeak <- NULL
>
>
> str(data3_dummies)
'data.frame':	297 obs. of  9 variables:
 $ sex     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp      : num  1 4 4 3 2 2 4 4 4 4 ...
 $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg: num  2 2 2 0 2 0 2 0 2 2 ...
 $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ slope   : num  3 2 2 3 1 1 3 1 2 3 ...
 $ ca      : num  0 3 2 0 0 0 2 0 1 0 ...
 $ thal    : num  6 3 7 3 3 3 3 3 7 7 ...
 $ num     : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
```

Before any variables could be coded as dummy variable using dummyVars() function from the caret package, they need to be converted to factors. I used lapply function to apply as.factor to all 8 variables in the data frame:

```
> #convert variables from num to factors
> ##something to try later
> #numvar <- ("names of the colomns", "  ", )
> todummies <- c("sex", "cp", "fbs", "restecg", "exang", "slope", "thal", "ca")
> #variables that need to be substituted with dummies need to be converted to factors
> dumvars <- as.data.frame(lapply(data3_dummies[todummies], as.factor))
> #dumvars - dataframe with 8 variables as factors
> str(dumvars)
'data.frame':	297 obs. of  8 variables:
 $ sex     : Factor w/ 2 levels "0","1": 2 2 2 2 1 2 1 1 2 2 ...
 $ cp      : Factor w/ 4 levels "1","2","3","4": 1 4 4 3 2 2 4 4 4 4 ...
 $ fbs     : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 2 ...
 $ restecg: Factor w/ 3 levels "0","1","2": 3 3 3 1 3 1 3 1 3 3 ...
 $ exang   : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 1 2 1 2 ...
 $ slope   : Factor w/ 3 levels "1","2","3": 3 2 2 3 1 1 3 1 2 3 ...
 $ thal    : Factor w/ 3 levels "3","6","7": 2 1 3 1 1 1 1 1 3 3 ...
 $ ca     : Factor w/ 4 levels "0","1","2","3": 1 4 3 1 1 1 3 1 2 1 ...
```

The resulting data frame dumvars contains 8 variables that need to be coded as dummies. The data frame also needs to include the target variable, so I added it by creating a target factor and using the cbind() function:

```
> #create vector with the target num variable
> target <- c(data3_dummies$num)
> target <- as.factor(target)
> #add target vector back to the df with categorical variables
> data3_dummies <- cbind(dumvars, target)
```

Next step is to create dummy variables:

```
> #use dummyVars() from caret to create dummy variables
> dV <- dummyVars(formula = target ~., data =data3_dummies)
> dV
Dummy Variable Object

Formula: target ~ .
9 variables, 9 factors
Variables and levels will be separated by '.'
A less than full rank encoding is used
> data4_dummies <- as.data.frame(predict(object=dV, newdata = data3_dummies))
Warning message:
In model.frame.default(Terms, newdata, na.action = na.action, xlev = object$lvls) :
  variable 'target' is not a factor
> str(data4_dummies)
'data.frame':	297 obs. of  23 variables:
 $ sex.0     : num  0 0 0 0 1 0 1 1 0 0 ...
 $ sex.1     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp.1      : num  1 0 0 0 0 0 0 0 0 0 ...
 $ cp.2      : num  0 0 0 0 1 1 0 0 0 0 ...
 $ cp.3      : num  0 0 0 1 0 0 0 0 0 0 ...
 $ cp.4      : num  0 1 1 0 0 0 1 1 1 1 ...
 $ fbs.0     : num  0 1 1 1 1 1 1 1 1 0 ...
 $ fbs.1     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg.0: num  0 0 0 1 0 1 0 1 0 0 ...
 $ restecg.1: num  0 0 0 0 0 0 0 0 0 0 ...
 $ restecg.2: num  1 1 1 0 1 0 1 0 1 1 ...
 $ exang.0   : num  1 0 0 1 1 1 1 0 1 0 ...
 $ exang.1   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ slope.1   : num  0 0 0 0 1 1 0 1 0 0 ...
 $ slope.2   : num  0 1 1 0 0 0 0 0 1 0 ...
 $ slope.3   : num  1 0 0 1 0 0 1 0 0 1 ...
 $ thal.3    : num  0 1 0 1 1 1 1 1 0 0 ...
 $ thal.6    : num  1 0 0 0 0 0 0 0 0 0 ...
 $ thal.7    : num  0 0 1 0 0 0 0 0 1 1 ...
 $ ca.0      : num  1 0 0 1 1 1 0 1 0 1 ...
 $ ca.1      : num  0 0 0 0 0 0 0 0 1 0 ...
 $ ca.2      : num  0 0 1 0 0 0 1 0 0 0 ...
 $ ca.3      : num  0 1 0 0 0 0 0 0 0 0 ...
```

The resulting data frame data4_dummies contains all 297 observations for the initial 8 categorical variables now coded as 23 variables.

Now we need to normalize continuous variables. First, I created a data frame (data5_numeric) containing 5 continuous variables and the target variable (factor).

```
> #########
> #Create a data frame with remaining numeric variables and the target variable
> #str(data2_complete)
> data5_numeric <- data2_complete[-2]
> data5_numeric$cp <- NULL
> data5_numeric$fbs <- NULL
> data5_numeric$restecg <- NULL
> data5_numeric$exang <- NULL
> data5_numeric$slope <- NULL
> data5_numeric$thal <- NULL
> data5_numeric$ca<- NULL


> str(data5_numeric) #DF with continuous variables and target
'data.frame':	297 obs. of  6 variables:
 $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
 $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
 $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
 $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
 $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ num     : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
```

In order to normalize the continuous variables I used the following function:

```
> ######Normalization
>
> #create normalize function
> normalize <- function(x) {
+   return((x-min(x)) / (max(x) - min(x)))
+ }
```

Then, I applied it to all 5 columns than needed normalization using lapply() function:

```
> #columns that need normalization
> neednorm <- c("age", "trestbps", "chol", "thalach", "oldpeak")
>
> data7_norm <-as.data.frame(lapply(data5_numeric[neednorm], normalize))
> str(data7_norm)
'data.frame':	297 obs. of  5 variables:
 $ age     : num  0.708 0.792 0.792 0.167 0.25 ...
 $ trestbps: num  0.481 0.623 0.245 0.34 0.34 ...
 $ chol    : num  0.244 0.365 0.235 0.283 0.178 ...
 $ thalach : num  0.603 0.282 0.443 0.885 0.771 ...
 $ oldpeak : num  0.371 0.242 0.419 0.565 0.226 ...
```

The resulting data7_norm data frame contains 297 observations for 5 variables that were normalized.

Next, I put together all columns for the dummy variables, normalized continuous variables, and the target variable using cbind(0 function:

```
> #Put together full normalized DF
> data_normbound <- cbind(data4_dummies, data7_norm, target)
> str(data_normbound) #This is FULL NORMALIZED df before splitting
'data.frame':	297 obs. of  29 variables:
 $ sex.0    : num  0 0 0 0 1 0 1 1 0 0 ...
 $ sex.1    : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp.1     : num  1 0 0 0 0 0 0 0 0 0 ...
 $ cp.2     : num  0 0 0 0 1 1 0 0 0 0 ...
 $ cp.3     : num  0 0 0 1 0 0 0 0 0 0 ...
 $ cp.4     : num  0 1 1 0 0 0 1 1 1 1 ...
 $ fbs.0    : num  0 1 1 1 1 1 1 1 1 0 ...
 $ fbs.1    : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg.0: num  0 0 0 1 0 1 0 1 0 0 ...
 $ restecg.1: num  0 0 0 0 0 0 0 0 0 0 ...
 $ restecg.2: num  1 1 1 0 1 0 1 0 1 1 ...
 $ exang.0  : num  1 0 0 1 1 1 1 0 1 0 ...
 $ exang.1  : num  0 1 1 0 0 0 0 1 0 1 ...
 $ slope.1  : num  0 0 0 0 1 1 0 1 0 0 ...
 $ slope.2  : num  0 1 1 0 0 0 0 0 1 0 ...
 $ slope.3  : num  1 0 0 1 0 0 1 0 0 1 ...
 $ thal.3   : num  0 1 0 1 1 1 1 1 0 0 ...
 $ thal.6   : num  1 0 0 0 0 0 0 0 0 0 ...
 $ thal.7   : num  0 0 1 0 0 0 0 0 1 1 ...
 $ ca.0     : num  1 0 0 1 1 1 0 1 0 1 ...
 $ ca.1     : num  0 0 0 0 0 0 0 0 1 0 ...
 $ ca.2     : num  0 0 1 0 0 0 1 0 0 0 ...
 $ ca.3     : num  0 1 0 0 0 0 0 0 0 0 ...
 $ age      : num  0.708 0.792 0.792 0.167 0.25 ...
 $ trestbps : num  0.481 0.623 0.245 0.34 0.34 ...
 $ chol     : num  0.244 0.365 0.235 0.283 0.178 ...
 $ thalach  : num  0.603 0.282 0.443 0.885 0.771 ...
 $ oldpeak  : num  0.371 0.242 0.419 0.565 0.226 ...
 $ target   : Factor w/ 2 levels "1","2": 1 2 2 1 1 1 2 1 2 2 ...
```

The resulting data_normbound contains all the variables for the complete data set.

Next, I used createDataPartition() from the caret package to split the data set into training (70%) and testing data (30%). I also created labels for training and deleted the target variables from the data frame.

```
> #Create training and testing data sets
> #using createDataPartition from caret package
```

```
>
> trainingindex <-  createDataPartition(data_normbound$target,p=0.7, list=FALSE)
>
> data_train <- data_normbound[trainingindex,] #full training data set
> data_test <- data_normbound[-trainingindex,] #full testing data set
>
> #labels
> data_trainlbl <- data_train$target
> data_testlbl <- data_test$target
>
> #training and testing sets without the target variable
> data_trainmodel <- data_train[, -29]
> data_testmodel <- data_test[ , -29]
```

The training data set contains 208 observations and the testing data set contains 89 observations.

Finally, the data is ready for model fitting/classification using the following code starting with the k=1:

```
> ###Fit the model
> #use k=1
> knn_pred <- knn(train=data_trainmodel, test = data_testmodel, cl=data_trainlbl, k=1)
```

In order to evaluate the model, I used CrossTab() function from the gmaodel package:

```
> #Evaluate the model using CrossTable() from gmodels package
> CrossTable(x = data_testlbl, y = knn_pred, prop.chisq = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  89


                | knn_pred
data_testlbl |         1 |         2 | Row Total |
-------------|-----------|-----------|-----------|
           1 |        37 |        11 |        48 |
             |     0.771 |     0.229 |     0.539 |
             |     0.822 |     0.250 |           |
             |     0.416 |     0.124 |           |
-------------|-----------|-----------|-----------|
           2 |         8 |        33 |        41 | 
             |     0.195 |     0.805 |     0.461 | 
             |     0.178 |     0.750 |           | 
             |     0.090 |     0.371 |           | 
-------------|-----------|-----------|-----------|
Column Total |        45 |        44 |        89 | 
             |     0.506 |     0.494 |           | 
-------------|-----------|-----------|-----------|
```

The above output shows that with k=1 the model accurately classifies 37 cases of absence heart disease and 33 cases of presence of heart disease. However, in 11 cases the model erroneously misclassified healthy patients as patients with heart disease (false positives), and 8 patients, who actually had disease went undetected. (false negatives). It means that the accuracy was (33+37)/89=78.65%.

I used the same model to test the data for k equal to 3, 5, 7, 9, 11, 13, 15, 17 and 19. The results of the classifier included in the table below. K=1 showed the worst prediction accuracy. The best predication accuracy was reached with k=5 at 92.13% accuracy.

Accuracy of KNN classification (using normalization):

K	**Correct classification**	**Incorrect Classification (FN + FP)**	**Accuracy**
1	70	8+11	78.65%
3	76	6+7	85.39%
5	82	5+2	92.13%
7	80	6+3	89.88%
9	81	5+3	91.01%
11	80	5+4	89.88%
13	80	6+3	89.88%
15	80	6+3	89.88%
17	81	6+2	91.01
19	79	8+2	88.76%

FP – false positive

FN- false negative

In addition, I decided to check if there are any differences between using normalization vs. standardization. I used the built-in scale() function to transform the continuous variables using z-score standardization:

```
> #Create a data frame with standardized values
> data7_z <-as.data.frame(scale(data5_numeric[neednorm]))
> str(data7_z)
'data.frame':	297 obs. of  5 variables:
 $ age     : num  0.935 1.377 1.377 -1.938 -1.496 ...
 $ trestbps: num  0.7491 1.5936 -0.6583 -0.0953 -0.0953 ...
 $ chol    : num  -0.276 0.743 -0.353 0.051 -0.834 ...
 $ thalach : num  0.0175 -1.8133 -0.8979 1.6303 0.9764 ...
 $ oldpeak : num  1.067 0.381 1.324 2.096 0.295 ...
```

The rest of the steps for creating training and testing datasets were similar to the previous case:

```
> #Put together full standardized DF
> data_zbound <- cbind(data4_dummies, data7_z, target)
> str(data_zbound) #This is FULL STANDARTIZED df before splitting
'data.frame':	297 obs. of  29 variables:
 $ sex.0     : num  0 0 0 0 1 0 1 1 0 0 ...
 $ sex.1     : num  1 1 1 1 0 1 0 0 1 1 ...
 $ cp.1      : num  1 0 0 0 0 0 0 0 0 0 ...
 $ cp.2      : num  0 0 0 0 1 1 0 0 0 0 ...
 $ cp.3      : num  0 0 0 1 0 0 0 0 0 0 ...
 $ cp.4      : num  0 1 1 0 0 0 1 1 1 1 ...
 $ fbs.0     : num  0 1 1 1 1 1 1 1 1 0 ...
 $ fbs.1     : num  1 0 0 0 0 0 0 0 0 1 ...
 $ restecg.0: num  0 0 0 1 0 1 0 1 0 0 ...
 $ restecg.1: num  0 0 0 0 0 0 0 0 0 0 ...
 $ restecg.2: num  1 1 1 0 1 0 1 0 1 1 ...
 $ exang.0   : num  1 0 0 1 1 1 1 0 1 0 ...
 $ exang.1   : num  0 1 1 0 0 0 0 1 0 1 ...
 $ slope.1   : num  0 0 0 0 1 1 0 1 0 0 ...
 $ slope.2   : num  0 1 1 0 0 0 0 0 1 0 ...
 $ slope.3   : num  1 0 0 1 0 0 1 0 0 1 ...
 $ thal.3    : num  0 1 0 1 1 1 1 1 0 0 ...
 $ thal.6    : num  1 0 0 0 0 0 0 0 0 0 ...
 $ thal.7    : num  0 0 1 0 0 0 0 0 1 1 ...
 $ ca.0      : num  1 0 0 1 1 1 0 1 0 1 ...
 $ ca.1      : num  0 0 0 0 0 0 0 0 1 0 ...
 $ ca.2      : num  0 0 1 0 0 0 1 0 0 0 ...
 $ ca.3      : num  0 1 0 0 0 0 0 0 0 0 ...
 $ age       : num  0.935 1.377 1.377 -1.938 -1.496 ...
 $ trestbps  : num  0.7491 1.5936 -0.6583 -0.0953 -0.0953 ...
 $ chol      : num  -0.276 0.743 -0.353 0.051 -0.834 ...
 $ thalach   : num  0.0175 -1.8133 -0.8979 1.6303 0.9764 ...
 $ oldpeak   : num  1.067 0.381 1.324 2.096 0.295 ...
 $ target    : Factor w/ 2 levels "1","2": 1 2 2 1 1 1 2 1 2 2 ...
```

KNN

```
>
> #Create training and testing data sets
> #using createDataPartition from caret package
>
> data_ztrain <- data_zbound[trainingindex,] #full training data set
> data_ztest <- data_zbound[-trainingindex,] #full testing data set
>
> #training and testing sets without target variable
> data_ztrainmodel <- data_ztrain[, -29]
> data_ztestmodel <- data_ztest[ , -29]
```

Next, I fitted the model and used CrossTable() from the gmodel package to create the following confusion matrix:

```
> ###Fit the model
> #use k=1
> knn_zpred <- knn(train=data_ztrainmodel, test = data_ztestmodel, cl=data_trainlbl, k=1)
>
> #Evaluate the model using CrossTable() from gmodels package
> CrossTable(x = data_testlbl, y = knn_zpred, prop.chisq = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  89


             | knn_zpred
data_testlbl |         1 |         2 | Row Total |
-------------|-----------|-----------|-----------|
           1 |        37 |        11 |        48 |
             |     0.771 |     0.229 |     0.539 |
             |     0.787 |     0.262 |           |
             |     0.416 |     0.124 |           |
-------------|-----------|-----------|-----------|
           2 |        10 |        31 |        41 |
             |     0.244 |     0.756 |     0.461 |
             |     0.213 |     0.738 |           |
             |     0.112 |     0.348 |           |
-------------|-----------|-----------|-----------|
Column Total |        47 |        42 |        89 |
             |     0.528 |     0.472 |           |
-------------|-----------|-----------|-----------|
```

The above output demonstrates that the classifier accurately predicted 37 cases of no disease and 31 cases of the presence of heart disease. At the same time, it misclassified 11 healthy patients as having heart disease (false positive) and 10 cases of heart disease were mistakenly classified as healthy patients (false negatives). It means that the accuracy of the classification was (31+37)/89 = 76.4%.

I ran the same classifier for k equal to 3, 5, 7, 9, 11, 13, and 15. The results are included in the table below. The accuracy increased with increasing K reaching its highest at k=17 - 92.13%.

Accuracy of KNN classification (using z-score standardization):

K	**Correct classification**	**Incorrect Classification (FN + FP)**	**Accuracy**
1	68	10+11	76.4%
3	73	9+7	82.02%
5	78	7+4	87.64%
7	77	9+3	86.52%
9	78	8+3	87.64%
11	78	8+3	87.64%
13	80	8+1	89.89%
15	81	6+2	91.01%
17	82	6+1	92.13%
19	81	7+1	91.01%

If we compare result for the two transformation techniques (normalization and z-score standardization), neither of them had considerably changed the outcome. The best accuracy with normalization was reached with lower K value, but the results are inconsistent for these training and testing datasets.