Guillermo Romera | Practicum 3 | DA5030 | Intro to Data Mining/Machine Learning| Term : Fall 2017

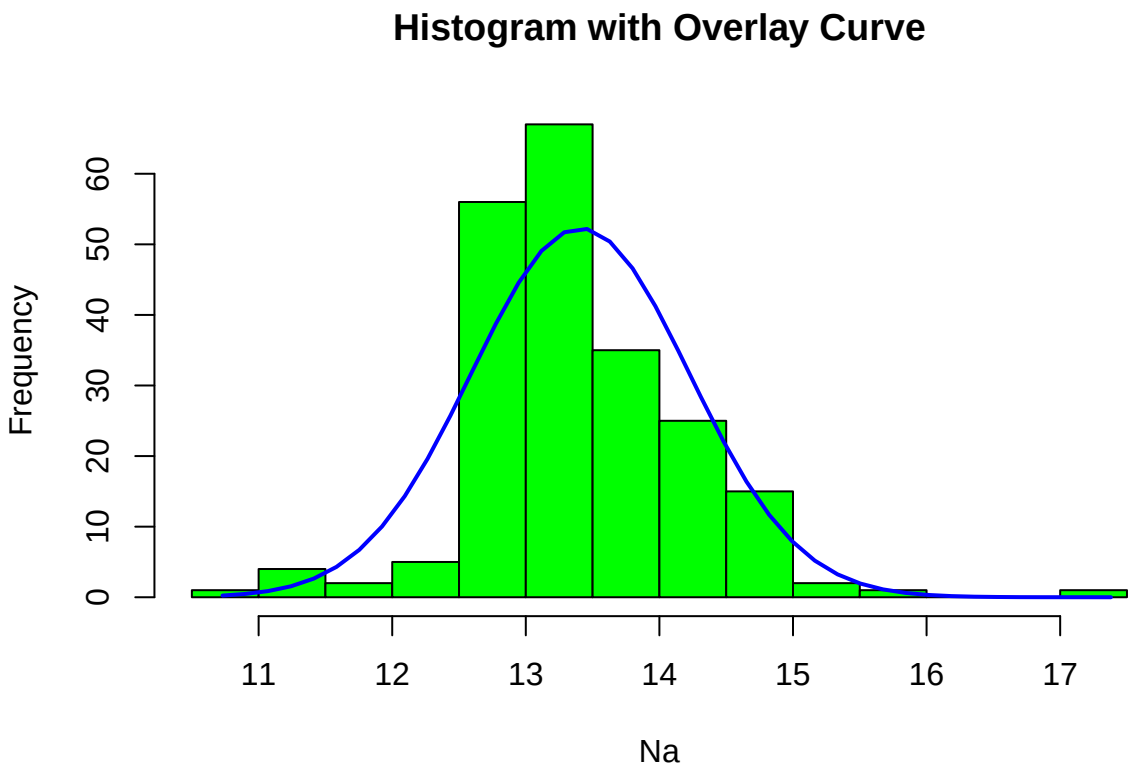
1. (0 pts) Download the data set Glass Identification Database along with its explanation. Note that the data file does not contain header names; you may wish to add those. The description of each column can be found in the data set explanation. This assignment must be completed within an R Markdown Notebook.

2. (0 pts) Explore the data set as you see fit and that allows you to get a sense of the data and get comfortable with it.

```
library(readr)
glass <- read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data",
    col_names = FALSE)

#This line changes the names of the columns for it's appripiate names.
names(glass) <- c("Id","Ri","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type")
write.csv(glass, "glass.csv")
```

3. (5 pts) Create a histogram of the Na column and overlay a normal curve; visually determine whether the data is normally distributed. You may use the code from this tutorial. Does the k-NN algorithm require normally distributed data or is it a non-parametric method? Comment on your findings.

```
x <- glass$Na
h<-hist(x, breaks=10, col="green", xlab="Na",
    main="Histogram with Overlay Curve")
xfit<-seq(min(x),max(x),length=40)
yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
yfit <- yfit*diff(h$mids[1:2])*length(x)
lines(xfit, yfit, col="blue", lwd=2)
```



It seem as if the data is not normally distributed. The second plot gives us a better representation of how the curve is for the Na column. For the second question no, K-NN algorithm doesn't need a normal distribution and is classified as a non-parametric method.

4.(5 pts) After removing the ID column (column 1), normalize the first two columns in the data set using min-max normalization.

```
#Removing the ID column
glass<-glass[-1]

glass.Type<-glass$Type

glass$Type=NULL

#min max normalization function
normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }



#Min max normalization of the first two columns
glass1 <- as.data.frame(lapply(glass[1:2], normalize))
```

5. (5 pts) Normalize the remaining columns, except the last one, using z-score standardization. The last column is the glass type and so it is excluded.

```
#z-score normalization function
zscore<-function(x){
  return ((x- mean(x))/sd(x))
}

#normalizing columns 3 through 9
glass2<-as.data.frame(lapply(glass[3:9], zscore))

#Putting all normalized data together in one data frame
glass_sorted<-data.frame(glass1,glass2,glass.Type)
```

6. (10 pts) The data set is sorted, so creating a validation data set requires random selection of elements. Create a stratified sample where you randomly select 50% of each of the cases for each glass type to be part of the validation data set. The remaining cases will form the training data set.

```
set.seed(20)
#subsetting the data for a certain glass type, in this particular case Type1
type1<-subset(glass_sorted, glass_sorted$glass.Type==1)
#creating a random sample of certain glass Type , in this case Type1
sample <-sample(nrow(type1), 35)
sample1 <-sample(nrow(type1), 35)
#using the sample to fill out a validation variable
validation1 <- type1[sample, ]
#using the sample to substract what is left and filling out the training variable
training1  <- type1[-sample, ]

type2<-subset(glass_sorted, glass_sorted$glass.Type==2)
sample2 <-sample(nrow(type2), 38)
validation2 <- type2[sample2, ]
training2  <- type2[-sample2, ]

type3<-subset(glass_sorted, glass_sorted$glass.Type==3)
sample3 <-sample(nrow(type3), 9)
validation3 <- type3[sample3, ]
training3  <- type3[-sample3, ]

type5<-subset(glass_sorted, glass_sorted$glass.Type==5)
sample5 <-sample(nrow(type5), 7)
validation5 <- type5[sample5, ]
training5  <- type5[-sample5, ]

type6<-subset(glass_sorted, glass_sorted$glass.Type==6)
sample6 <-sample(nrow(type6), 4)
validation6 <- type6[sample6, ]
training6  <- type6[-sample6, ]

type7<-subset(glass_sorted, glass_sorted$glass.Type==7)
sample7 <-sample(nrow(type7), 14)
```

```
validation7 <- type7[sample7, ]
training7  <- type7[-sample7, ]


#putting together all samples from the validation variables into one data frame
glass_validation<-rbind.data.frame(validation1,
                                   validation2,validation3,
                                   validation5,validation6,validation7)

row.names(glass_validation)<-1:nrow(glass_validation)

#putting together all samples from the training variables into one data frame
glass_training<-rbind.data.frame(training1,
                                 training2,training3,training5,training6,training7)

row.names(glass_training)<-1:nrow(glass_training)
```

7. (30 pts) Implement the k-NN algorithm in R (do not use an implementation of k-NN from a package) and use your algorithm with a k=10 to predict the glass type for the following two cases:

```
library(readr)
glass <- read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data",
    col_names = FALSE)
names(glass) <- c("Id","Ri","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type")

#the first case of the problem
p1<-data.frame (1.51621,12.53 ,3.48, 1.39 , 73.39 , 0.60 , 8.55 , 0.00 , 0.05)
#it's proper nomenglature
names(p1) <- c("Ri","Na","Mg","Al","Si","K","Ca","Ba","Fe")
#the second case of the problem
p2<-data.frame ( 1.5098 , 12.77, 1.85, 1.81, 72.69, 0.59,10.01, 0.00,0.01)
names(p2) <- c("Ri","Na","Mg","Al","Si","K","Ca","Ba","Fe")

#deleting ID column
glass<-glass[-1]

glass.Type<-glass$Type

glass$Type=NULL

#min max normalization
normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }


glass<-rbind.data.frame(glass,p1,p2)

glass1 <- as.data.frame(lapply(glass[1:2], normalize))

#zscore normalization
zscore<-function(x){
```

```
  return ((x- mean(x))/sd(x))
}

glass2<-as.data.frame(lapply(glass[3:9], zscore))

x<-glass1[215:216, ]
glass1<-glass1[-215: -216, ]
x<-data.frame(x,glass2[215:216, ])

glass2<-glass2[-215: -216, ]

#creating a new data set with everything normalized
glass_sorted1<-data.frame(glass1,glass2,glass.Type)


test<-x
training<-glass_sorted1

#changing the levels of the glass type for names to make it easier to work with
training$glassie <- factor(training$glass.Type,
                           levels = c(1, 2,3,5,6,7),
                           labels = c("Type1", "Type2","Type3","Type5","Type6","Type7"))

#function for the euclidean distance
euclideanDistance = function(a, b) {
    sqrt(sum((a - b)^2))
}
#function for the Knn algorithm
knn1 = function(train, target,test, k) {
    n = nrow(train)

eucDist = vector()
    for(i in 1:n) {
        eucDist = c(eucDist, euclideanDistance(train[i, ], test))
    }

    #sorting the distances in ascending order
firstK = order(eucDist)[1:k]
    #using the kj value so we can get the target values
fTarget = target[firstK]
#sorting the variable above based on the frecuency so we can get the most repeated target case
fTarget = names(sort(table(fTarget), decreasing = TRUE))[1]
    return(fTarget)
}
#the column of the target cases
target = training$glassie
#emptying the training dataset of the glass type column
training$glass.Type=NULL
training$glassie=NULL


#empty vector to be filled out with the prediction
predictedTarget = vector()
```

```
#loop to run the knn function over all rows of the test data set
for(i in 1:nrow(test)) {
    predictedTarget = c(predictedTarget,
                        knn1(train = training,
                                              target = target, test = test[i,], k = 10))
}

#the prediction of the new cases, Type1 for the 1st and Type6 for the 2nd
predictedTarget
```

```
## [1] "Type1" "Type6"
```

8. (10 pts) Apply the knn function from the class package with k=14 and redo the cases from Question (7).

```
library(class)
library(gmodels)

row.names(x)<-1:nrow(x)
x<-data.matrix(x)

glass_sorted_label <-as.data.frame(glass_sorted1[, 10])

names(glass_sorted_label) <- c("glassie")


glass_sorted1$glassie<-NULL
glass_sorted1$glass.Type<-NULL



test_pred1 <- knn(glass_sorted1, x,glass_sorted_label$glassie, k=14)

table(test_pred1)
```

```
## test_pred1
## 1 2 3 5 6 7
## 1 0 0 0 1 0
```

```
#First case to predict
test_pred1[1]
```

```
## [1] 1
## Levels: 1 2 3 5 6 7
```

```
#Second case to predict
test_pred1[2]
```

```
## [1] 6
## Levels: 1 2 3 5 6 7
```

9. (12 pts) Determine the accuracy of the knn function with k=14 from the class package by applying it against each case in the validation data set. What is the percentage of correct classifications?

```
library(class)

#taking the target from the training data set
glass_training_label <-as.data.frame( glass_training[ , 10])
#taking the target from the validation data set
glass_validation_label<- as.data.frame( glass_validation[ , 10])

#emptying the training and validation data set of the target
glass_training$glass.Type<-NULL
glass_validation$glass.Type<-NULL


names(glass_training_label) <- c("glass.Type")
names(glass_validation_label) <- c("glass.Type")


test_pred <- knn(train = glass_training,
                 test = glass_validation,
                 cl = glass_training_label$glass.Type, k=14)

predc<-data.frame(as.numeric(test_pred))


library(OneR)
library(caret)
glass_validation_label1<-unlist(glass_validation_label)
test_pred1<-unlist(test_pred)
confusionMatrix(glass_validation_label1, test_pred1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  5  6  7
##          1 24 11  0  0  0  0
##          2 11 25  0  0  1  1
##          3  6  3  0  0  0  0
##          5  3  1  0  0  0  3
##          6  4  0  0  0  0  0
##          7  2  2  0  0  0 10
##
## Overall Statistics
##
##                Accuracy : 0.5514
##                  95% CI : (0.4522, 0.6477)
##     No Information Rate : 0.4673
##     P-Value [Acc > NIR] : 0.04993
##
##                   Kappa : 0.3501
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
## Sensitivity            0.4800   0.5952       NA       NA 0.000000  0.71429
## Specificity            0.8070   0.8000  0.91589  0.93458 0.962264  0.95699
## Pos Pred Value         0.6857   0.6579       NA       NA 0.000000  0.71429
## Neg Pred Value         0.6389   0.7536       NA       NA 0.990291  0.95699
## Prevalence             0.4673   0.3925  0.00000  0.00000 0.009346  0.13084
## Detection Rate         0.2243   0.2336  0.00000  0.00000 0.000000  0.09346
## Detection Prevalence   0.3271   0.3551  0.08411  0.06542 0.037383  0.13084
## Balanced Accuracy      0.6435   0.6976       NA       NA 0.481132  0.83564
```

```
#The percentage of correct classifications is
((24+25+10)*100)/107
```

```
## [1] 55.14019
```

10. (7 pts) Determine an optimal k by trying all values from 5 through 14 for your own k-NN algorithm implementation against the cases in the validation data set. What is the optimal k, i.e., the k that results in the best accuracy?

```
set.seed(45)
library(OneR)
library(caret)

glass_trat<-glass_training

   n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
                 test = glass_validation[i,], k = 5))
}
 glass_trat$k5 <- n
glass_ks<- glass_trat[, -c(1:8) ]



results<-confusionMatrix (glass_ks$k5,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
##          1 27 10  7  0  0  2
##          2  8 26  2  0  3  1
##          3  0  0  0  0  1  0
##          5  0  1  0  6  0  0
##          6  0  0  0  0  0  0
##          7  0  1  0  1  0 11
```

```
m5<-as.matrix(results, what = "overall")
m5
```

```
##                       [,1]
## Accuracy       6.542056e-01
## Kappa          5.099641e-01
## AccuracyLower  5.561046e-01
## AccuracyUpper  7.435301e-01
## AccuracyNull   3.551402e-01
## AccuracyPValue 3.118510e-10
## McnemarPValue           NaN
```

```
   n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
                 test = glass_validation[i,], k = 6))
}
 glass_ks$k6 <- n
glass_ks<-glass_ks[-1]
results<-confusionMatrix (glass_ks$k5,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
##          1 27 10  7  0  0  2
##          2  8 26  2  0  3  1
##          3  0  0  0  0  1  0
##          5  0  1  0  6  0  0
##          6  0  0  0  0  0  0
##          7  0  1  0  1  0 11
```

```
m5<-as.matrix(results, what = "overall")
m5
```

```
##                       [,1]
## Accuracy       6.542056e-01
## Kappa          5.099641e-01
## AccuracyLower  5.561046e-01
## AccuracyUpper  7.435301e-01
## AccuracyNull   3.551402e-01
## AccuracyPValue 3.118510e-10
## McnemarPValue           NaN
```

```
n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
                 test = glass_validation[i,], k = 7))
}
 glass_ks$k7 <- n

results<-confusionMatrix (glass_ks$k7,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
```

```
##          1 27  9  8  0  3  2
##          2  8 27  1  2  1  1
##          3  0  0  0  0  0  0
##          5  0  0  0  1  0  0
##          6  0  1  0  3  0  0
##          7  0  1  0  1  0 11
```

```
m5<-as.matrix(results, what = "overall")
m5
```

```
##                       [,1]
## Accuracy       6.168224e-01
## Kappa          4.522412e-01
## AccuracyLower  5.178405e-01
## AccuracyUpper  7.091640e-01
## AccuracyNull   3.551402e-01
## AccuracyPValue 3.290674e-08
## McnemarPValue           NaN
```

```
n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
                 test = glass_validation[i,], k = 8))
}
 glass_ks$k8 <- n

results<-confusionMatrix (glass_ks$k8,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
##          1 26  9  8  2  2  2
##          2  9 27  1  2  2  1
##          3  0  0  0  0  0  0
##          5  0  1  0  0  0  0
##          6  0  0  0  1  0  0
##          7  0  1  0  2  0 11
```

```
m5<-as.matrix(results, what = "overall")
m5
```

```
##                       [,1]
## Accuracy       5.981308e-01
## Kappa          4.198714e-01
## AccuracyLower  4.989201e-01
## AccuracyUpper  6.917665e-01
## AccuracyNull   3.551402e-01
## AccuracyPValue 2.682090e-07
## McnemarPValue           NaN
```

```
n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
```

```
                test = glass_validation[i,], k = 9))
}
 glass_ks$k9 <- n

results<-confusionMatrix (glass_ks$k9,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
##          1 24  7  8  2  3  3
##          2 11 29  1  2  1  1
##          3  0  0  0  0  0  0
##          5  0  1  0  0  0  0
##          6  0  0  0  1  0  0
##          7  0  1  0  2  0 10
```

```
m5<-as.matrix(results, what = "overall")
m5
```

```
##                       [,1]
## Accuracy       5.887850e-01
## Kappa          4.041261e-01
## AccuracyLower  4.895112e-01
## AccuracyUpper  6.830158e-01
## AccuracyNull   3.551402e-01
## AccuracyPValue 7.237220e-07
## McnemarPValue           NaN
```

```
n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
                 test = glass_validation[i,], k = 10))
}
 glass_ks$k10 <- n

results<-confusionMatrix (glass_ks$k10,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
##          1 25  9  8  2  3  1
##          2 10 27  1  2  1  2
##          3  0  0  0  0  0  0
##          5  0  0  0  0  0  0
##          6  0  1  0  1  0  0
##          7  0  1  0  2  0 11
```

```
m5<-as.matrix(results, what = "overall")
m5
```

```
##                       [,1]
## Accuracy       5.887850e-01
## Kappa          4.063800e-01
## AccuracyLower  4.895112e-01
```

```
## AccuracyUpper  6.830158e-01
## AccuracyNull   3.551402e-01
## AccuracyPValue 7.237220e-07
## McnemarPValue           NaN
```

```
n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
                 test = glass_validation[i,], k = 11))
}
 glass_ks$k11 <- n

results<-confusionMatrix (glass_ks$k11,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
##          1 24 10  9  2  4  2
##          2 11 26  0  2  0  2
##          3  0  0  0  0  0  0
##          5  0  0  0  1  0  0
##          6  0  1  0  0  0  0
##          7  0  1  0  2  0 10
```

```
m5<-as.matrix(results, what = "overall")
m5
```

```
##                       [,1]
## Accuracy       5.700935e-01
## Kappa          3.779856e-01
## AccuracyLower  4.707944e-01
## AccuracyUpper  6.654125e-01
## AccuracyNull   3.551402e-01
## AccuracyPValue 4.715644e-06
## McnemarPValue           NaN
```

```
n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
                 test = glass_validation[i,], k = 12))
}
 glass_ks$k12 <- n

results<-confusionMatrix (glass_ks$k12,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
##          1 25 11  6  2  4  2
##          2 10 25  3  2  0  2
##          3  0  0  0  0  0  0
##          5  0  0  0  0  0  0
```

```
##          6  0  1  0  0  0  0
##          7  0  1  0  3  0 10
```

```
m5<-as.matrix(results, what = "overall")
m5
```

```
##                       [,1]
## Accuracy       5.607477e-01
## Kappa          3.636594e-01
## AccuracyLower  4.614859e-01
## AccuracyUpper  6.565606e-01
## AccuracyNull   3.551402e-01
## AccuracyPValue 1.139433e-05
## McnemarPValue           NaN
```

```
n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
                 test = glass_validation[i,], k = 13))
}
 glass_ks$k13 <- n

results<-confusionMatrix (glass_ks$k13,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
##          1 24 12  7  2  4  2
##          2 11 24  2  2  0  2
##          3  0  0  0  0  0  0
##          5  0  0  0  0  0  0
##          6  0  1  0  0  0  0
##          7  0  1  0  3  0 10
```

```
m5<-as.matrix(results, what = "overall")
m5
```

```
##                       [,1]
## Accuracy       5.420561e-01
## Kappa          3.368328e-01
## AccuracyLower  4.429675e-01
## AccuracyUpper  6.387575e-01
## AccuracyNull   3.551402e-01
## AccuracyPValue 5.969591e-05
## McnemarPValue           NaN
```

```
n<-vector()
  for(i in 1:nrow(glass_validation)) {

     n<-c(n,knn1(train = glass_training,
                 target = glass_training_label$glass.Type,
                 test = glass_validation[i,], k = 14))
}
 glass_ks$k14 <- n
```

```
results<-confusionMatrix (glass_ks$k14,glass_validation_label$glass.Type)
as.table(results)
```

```
##           Reference
## Prediction  1  2  3  5  6  7
##          1 25 12  6  4  4  2
##          2 10 24  3  0  0  2
##          3  0  0  0  0  0  0
##          5  0  0  0  0  0  0
##          6  0  1  0  0  0  0
##          7  0  1  0  3  0 10
```

```
m5<-as.matrix(results, what = "overall")
m5
```

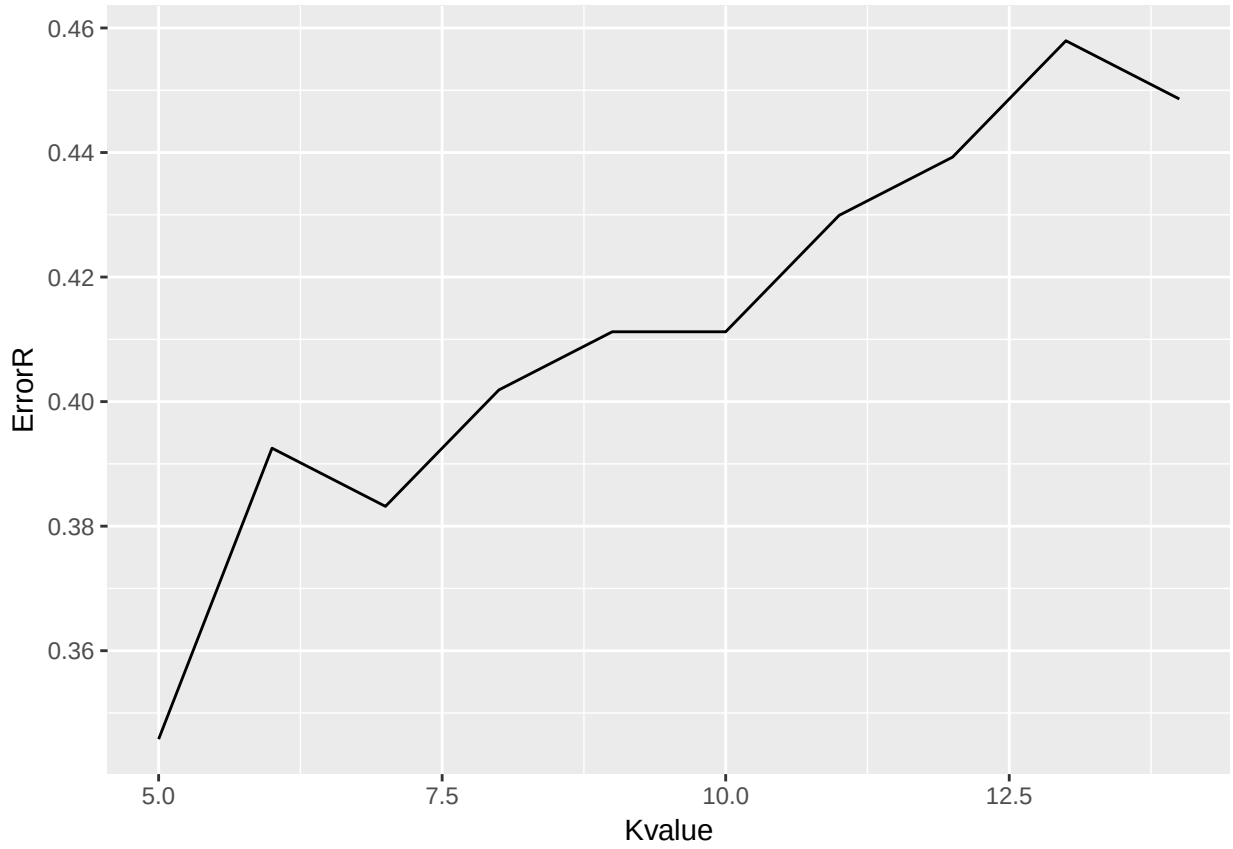
```
##                       [,1]
## Accuracy       0.5514018692
## Kappa          0.3508594540
## AccuracyLower  0.4522103294
## AccuracyUpper  0.6476755020
## AccuracyNull   0.3551401869
## AccuracyPValue 0.0000265532
## McnemarPValue           NaN
```

The K with the better accuracy are both 5 and 6,a thlough it seems rather strange that they both have the same accuracy, but that can be because of the choice of K and the sample randomness at that moment.

11. (5 pts) Create a plot of k (x-axis) versus error rate (percentage of incorrect classifications) using ggplot.

```
#Storing the error rate for each K in multiple variables
er5<-( mean (glass_validation_label != glass_ks$k5))
er6<-( mean (glass_validation_label != glass_ks$k6))
er7<-( mean (glass_validation_label != glass_ks$k7))
er8<-( mean (glass_validation_label != glass_ks$k8))
er9<-( mean (glass_validation_label != glass_ks$k9))
er10<-( mean (glass_validation_label != glass_ks$k10))
er11<-( mean (glass_validation_label != glass_ks$k11))
er12<-( mean (glass_validation_label != glass_ks$k12))
er13<-( mean (glass_validation_label != glass_ks$k13))
er14<-( mean (glass_validation_label !=glass_ks$k14))

#the values used for K in the prediction
Knumber<- cbind.data.frame(c(5,6,7,8,9,10,11,12,13,14))
names(Knumber) <- c("Kvalue")
#putting all the error rates ina  single data frame
erroRs<-rbind.data.frame(er5,er6,er7,er8,er9,er10,er11,er12,er13,er14)
names(erroRs) <- c("ErrorR")

#creating a dataframe with both the error rate and its respective value of k
plotD<- cbind.data.frame(Knumber,erroRs)


library(ggplot2)
```

```
ggplot(data = plotD, aes(x = Kvalue, y = ErrorR))+
  geom_line()
```



\begin{tcolorbox}[colframe=green!50, colback=green!10] **\textcolor{black} { 12.(5 pts) Produce a cross-table confusion matrix showing the accuracy of the classification using a package of your choice and a k of your choice.**

```
library(OneR)
library(caret)

#Confusion matrix
confusionMatrix (glass_ks$k5,glass_validation_label$glass.Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  5  6  7
##          1 27 10  7  0  0  2
##          2  8 26  2  0  3  1
##          3  0  0  0  0  1  0
##          5  0  1  0  6  0  0
##          6  0  0  0  0  0  0
##          7  0  1  0  1  0 11
##
## Overall Statistics
##
##                Accuracy : 0.6542
```

```
##                  95% CI : (0.5561, 0.7435)
##     No Information Rate : 0.3551
##     P-Value [Acc > NIR] : 3.119e-10
##
##                   Kappa : 0.51
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
## Sensitivity            0.7714   0.6842 0.000000  0.85714  0.00000   0.7857
## Specificity            0.7361   0.7971 0.989796  0.99000  1.00000   0.9785
## Pos Pred Value         0.5870   0.6500 0.000000  0.85714      NaN   0.8462
## Neg Pred Value         0.8689   0.8209 0.915094  0.99000  0.96262   0.9681
## Prevalence             0.3271   0.3551 0.084112  0.06542  0.03738   0.1308
## Detection Rate         0.2523   0.2430 0.000000  0.05607  0.00000   0.1028
## Detection Prevalence   0.4299   0.3738 0.009346  0.06542  0.00000   0.1215
## Balanced Accuracy      0.7538   0.7407 0.494898  0.92357  0.50000   0.8821
```

13. (3 pts) Comment on the run-time complexity of the k-NN for classifying w new cases using a training data set of n cases having m features. Assume that m is "large". How does this algorithm behave as w, n, and m increase? Would this algorithm be "fast" if the training data set and the number of features are large?

I would say it won't be fast, after some research for the practicum I did find that K-NN it's a simple and useful way of prediction, but nonethless one of its disadvantages is that it is not a fast working algorithm. One of the issues being working with Euclidean Distance where the algorithm has to run through all the cases all of datasets and calculate it. Because of this the algorithm would get slower with each increment of w,n or m.

14. (10 pts) Investigate this data set of home prices in King County (USA). How many cases are there? How many features? Imagine you are a real estate broker and are advising home sellers on how much their home is worth. Research and think about how you might use kNN to forecast (predict) the likely sales price for a home? Build a forecasting model with kNN and then forecast the price of some home (you can determine its features).

```
library(class)
library(FNN)
library(readr)

hd<- read_csv("C:/Users/Guillermo/Desktop/Practicum House sale/kc_house_data.csv")

#This line is to show the number of cases
nrow(hd)
```

```
## [1] 21613
```

```
#This line is to show the number of features
ncol(hd)
```

```
## [1] 21
```

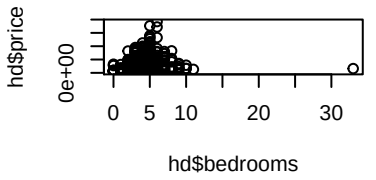
```r
hd[1:2] <- NULL


corH <- cor(hd)

#it does display the table of correlations between all the variables
print(round(corH, digits=2))
```

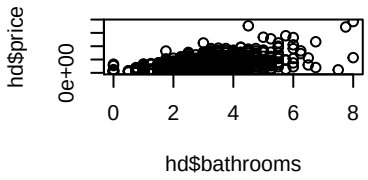
```
##               price bedrooms bathrooms sqft_living sqft_lot floors
## price          1.00     0.31      0.53        0.70     0.09   0.26
## bedrooms       0.31     1.00      0.52        0.58     0.03   0.18
## bathrooms      0.53     0.52      1.00        0.75     0.09   0.50
## sqft_living    0.70     0.58      0.75        1.00     0.17   0.35
## sqft_lot       0.09     0.03      0.09        0.17     1.00  -0.01
## floors         0.26     0.18      0.50        0.35    -0.01   1.00
## waterfront     0.27    -0.01      0.06        0.10     0.02   0.02
## view           0.40     0.08      0.19        0.28     0.07   0.03
## condition      0.04     0.03     -0.12       -0.06    -0.01  -0.26
## grade          0.67     0.36      0.66        0.76     0.11   0.46
## sqft_above     0.61     0.48      0.69        0.88     0.18   0.52
## sqft_basement  0.32     0.30      0.28        0.44     0.02  -0.25
## yr_built       0.05     0.15      0.51        0.32     0.05   0.49
## yr_renovated   0.13     0.02      0.05        0.06     0.01   0.01
## zipcode       -0.05    -0.15     -0.20       -0.20    -0.13  -0.06
## lat            0.31    -0.01      0.02        0.05    -0.09   0.05
## long           0.02     0.13      0.22        0.24     0.23   0.13
## sqft_living15  0.59     0.39      0.57        0.76     0.14   0.28
## sqft_lot15     0.08     0.03      0.09        0.18     0.72  -0.01
##               waterfront  view condition grade sqft_above sqft_basement
## price               0.27  0.40      0.04  0.67       0.61          0.32
## bedrooms           -0.01  0.08      0.03  0.36       0.48          0.30
## bathrooms           0.06  0.19     -0.12  0.66       0.69          0.28
## sqft_living         0.10  0.28     -0.06  0.76       0.88          0.44
## sqft_lot            0.02  0.07     -0.01  0.11       0.18          0.02
## floors              0.02  0.03     -0.26  0.46       0.52         -0.25
## waterfront          1.00  0.40      0.02  0.08       0.07          0.08
## view                0.40  1.00      0.05  0.25       0.17          0.28
## condition           0.02  0.05      1.00 -0.14      -0.16          0.17
## grade               0.08  0.25     -0.14  1.00       0.76          0.17
## sqft_above          0.07  0.17     -0.16  0.76       1.00         -0.05
## sqft_basement       0.08  0.28      0.17  0.17      -0.05          1.00
## yr_built           -0.03 -0.05     -0.36  0.45       0.42         -0.13
## yr_renovated        0.09  0.10     -0.06  0.01       0.02          0.07
## zipcode             0.03  0.08      0.00 -0.18      -0.26          0.07
## lat                -0.01  0.01     -0.01  0.11       0.00          0.11
## long               -0.04 -0.08     -0.11  0.20       0.34         -0.14
## sqft_living15       0.09  0.28     -0.09  0.71       0.73          0.20
## sqft_lot15          0.03  0.07      0.00  0.12       0.19          0.02
##               yr_built yr_renovated zipcode   lat  long sqft_living15
## price             0.05         0.13   -0.05  0.31  0.02          0.59
## bedrooms          0.15         0.02   -0.15 -0.01  0.13          0.39
## bathrooms         0.51         0.05   -0.20  0.02  0.22          0.57
## sqft_living       0.32         0.06   -0.20  0.05  0.24          0.76
## sqft_lot          0.05         0.01   -0.13 -0.09  0.23          0.14
```

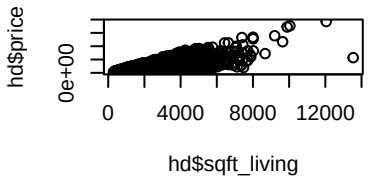
```
## floors            0.49         0.01   -0.06  0.05  0.13          0.28
## waterfront       -0.03         0.09    0.03 -0.01 -0.04          0.09
## view             -0.05         0.10    0.08  0.01 -0.08          0.28
## condition        -0.36        -0.06    0.00 -0.01 -0.11         -0.09
## grade             0.45         0.01   -0.18  0.11  0.20          0.71
## sqft_above        0.42         0.02   -0.26  0.00  0.34          0.73
## sqft_basement    -0.13         0.07    0.07  0.11 -0.14          0.20
## yr_built          1.00        -0.22   -0.35 -0.15  0.41          0.33
## yr_renovated     -0.22         1.00    0.06  0.03 -0.07          0.00
## zipcode          -0.35         0.06    1.00  0.27 -0.56         -0.28
## lat              -0.15         0.03    0.27  1.00 -0.14          0.05
## long              0.41        -0.07   -0.56 -0.14  1.00          0.33
## sqft_living15     0.33         0.00   -0.28  0.05  0.33          1.00
## sqft_lot15        0.07         0.01   -0.15 -0.09  0.25          0.18
##               sqft_lot15
## price               0.08
## bedrooms            0.03
## bathrooms           0.09
## sqft_living         0.18
## sqft_lot            0.72
## floors             -0.01
## waterfront          0.03
## view                0.07
## condition           0.00
## grade               0.12
## sqft_above          0.19
## sqft_basement       0.02
## yr_built            0.07
## yr_renovated        0.01
## zipcode            -0.15
## lat                -0.09
## long                0.25
## sqft_living15       0.18
## sqft_lot15          1.00
```

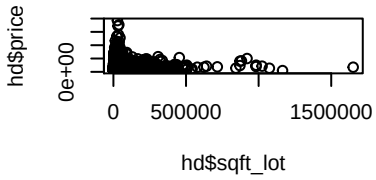
```r
#The correlation for price and sqft_living seems rather strong, that means we can use it
#The correlation between price and grade is also strong (0.67) so this is another variable to use
#Price and sqft above have a decent correlation too
#although the correlations are getting lower, price and sqft15 living seems good
#Correlation betwwen price and bathrooms is strong too

#After some more scrutiny of the correlations I have noticed that:
#The year that a house was built has also a strong correlation on the other features
#The number of floors laos has this correlation

#Sqft_above also has a strong correlation between multiple variables as it can be seen in the table


#Some plots to visually see the correlation of the variables with the price
par(mfrow=c(3,3))
plot (hd$bedrooms,hd$price)
plot (hd$bathrooms,hd$price)
plot (hd$sqft_living,hd$price)
plot (hd$sqft_lot,hd$price)
```

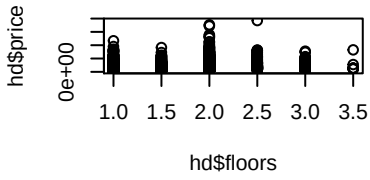
```
plot (hd$floors,hd$price)
plot (hd$waterfront,hd$price)
plot (hd$view ,hd$price)
plot (hd$condition,hd$price)
plot (hd$grade,hd$price)
```

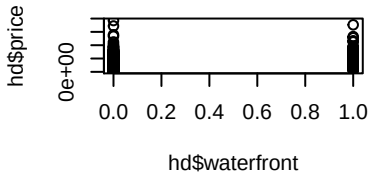


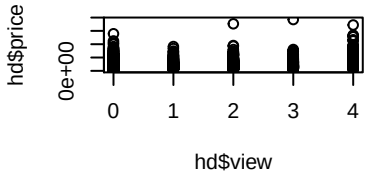


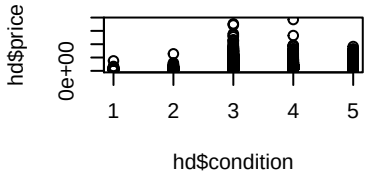


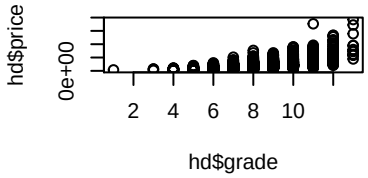




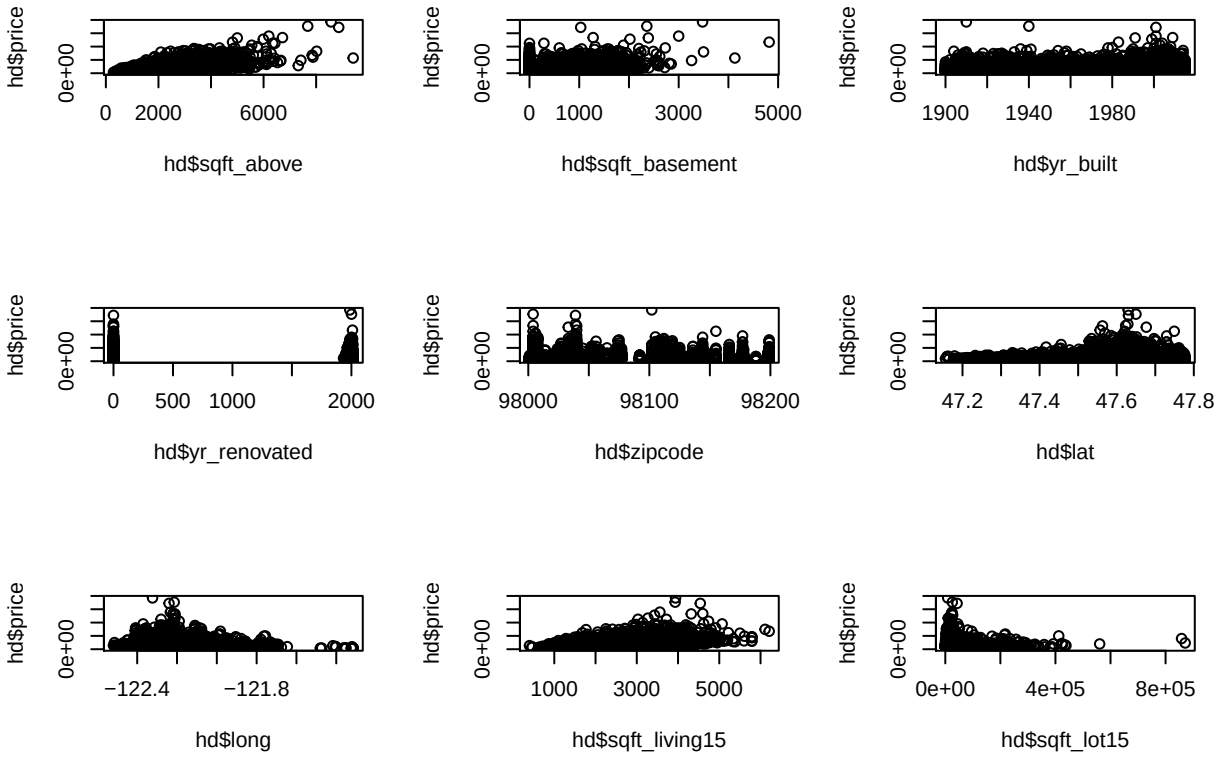








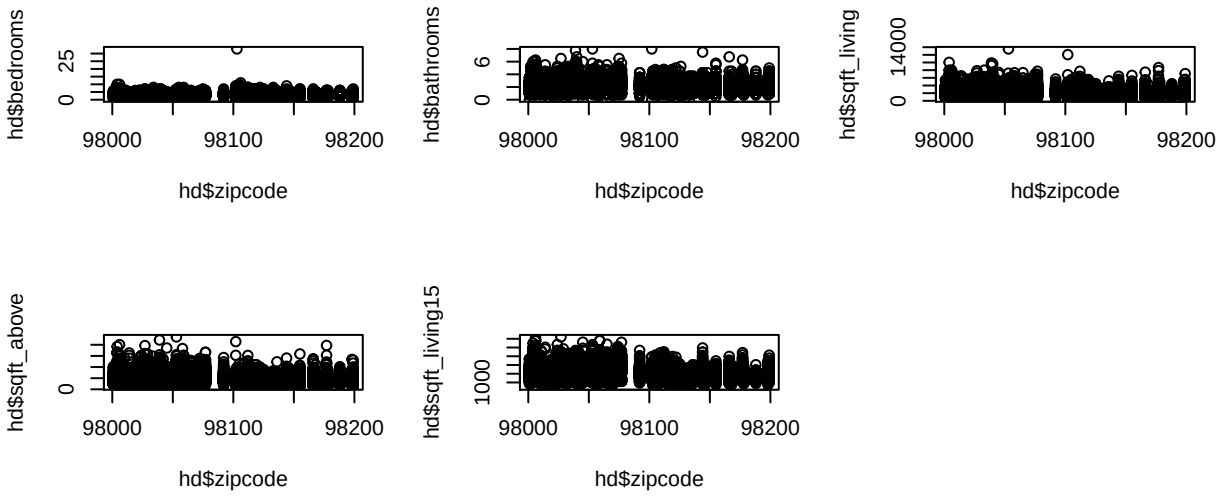

```
plot (hd$sqft_above,hd$price)
plot (hd$sqft_basement ,hd$price)
plot (hd$yr_built,hd$price)
plot (hd$yr_renovated,hd$price)
plot (hd$zipcode,hd$price)
plot (hd$lat,hd$price)
plot (hd$long,hd$price)
plot (hd$sqft_living15,hd$price)
plot (hd$sqft_lot15,hd$price)
```




```
#Now I have deiced to see what role plays the Zipcode
#When it comes to buying a house the zipcode can be very importat so let's see
par(mfrow=c(3,3))
plot (hd$zipcode,hd$bedrooms)
plot (hd$zipcode,hd$bathrooms)
plot (hd$zipcode,hd$sqft_living)
plot (hd$zipcode,hd$sqft_above)
plot (hd$zipcode,hd$sqft_living15)

#Selecting the features that I think are more important or do make an impact
hd2<- data.frame(hd$price,hd$bathrooms,hd$sqft_living15,hd$sqft_living,
                 hd$grade,hd$sqft_above,hd$zipcode, hd$yr_built, hd$floors)
names(hd2) <- c("Price","bathrooms","sqft_living15","sqft_living","grade","sqft_above","zipcode", "yrbu
hd3<-hd2[-1]

#Taking a sample from different columns to create my random house
sample2<-sample(hd2[ ,2],1)
sample3<-sample(hd2[ ,3],1)
sample4<-sample(hd2[ ,4],1)
sample5<-sample(hd2[ ,5],1)
sample6<-sample(hd2[ ,6],1)
sample7<-sample(hd2[ ,7],1)
sample8<-sample(hd2[ ,8],1)
sample9<-sample(hd2[ ,9],1)

new<-cbind.data.frame(sample2,sample3,sample4,sample5,sample6,sample7,sample8,sample9)
```

```r
names(new) <- c("bathrooms","sqft_living15","sqft_living","grade","sqft_above","zipcode","yrbuilt","floo

hd3<-rbind.data.frame(hd3,new)

zscore<-function(x){
  return ((x- mean(x))/sd(x))
}
#normalizing with zscore columns 1,4 &8
hdbath<-as.data.frame(lapply(hd3[1], zscore))
hdgrad<-as.data.frame(lapply(hd3[4], zscore))
hdfl<-as.data.frame(lapply(hd3[8], zscore))

normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }

#normalizing with min max normalization columns 2,3,5
hdliv15<-as.data.frame(lapply(hd3[2:3], zscore))
hdliv<-as.data.frame(lapply(hd3[5], zscore))

#I have decided to leave the zipcode and the eyar without normalizing
hd4<-cbind.data.frame(hdbath,hdliv15,hdgrad,hdliv,hd3$zipcode, hd3$yrbuilt,hdfl)

train1<-hd4[1:21613,-1 ]
test1<-hd4[21614,-1 ]

target1<-hd[1:21613, ]$price

#This is a regression model to be used with KNN
prediction <- knn.reg(train1, test1, target1, k = 147, algorithm="kd_tree")
prediction$pred
```

```
## [1] 372492.6
```

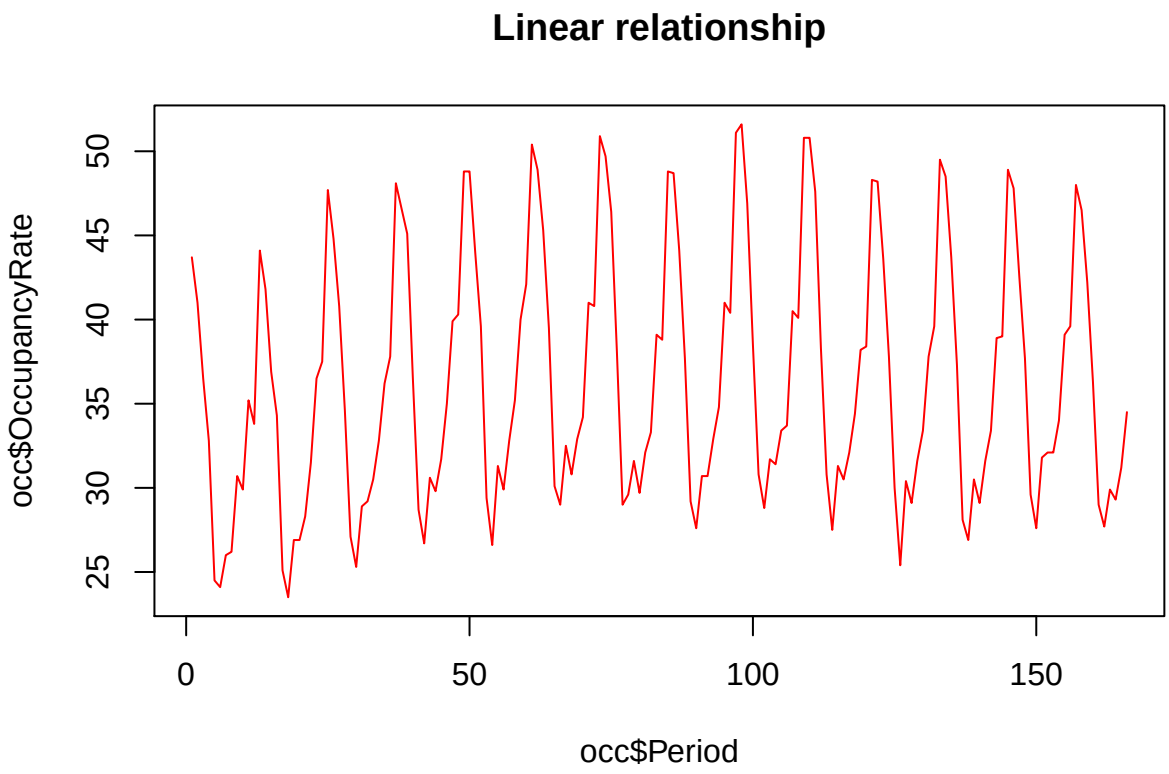


After some research it does seem like the main use with knn and continuos variables is the dimensionality. With continuous variables you'll have more a lot more points than with categorical variables hence a dimension reduction should be used. Also another issue that would rise up is that maybe, Euclidean Distance might not be the preferable choice

15. (10 pts) Inspect the data set of occupancy rates for a series of time periods. Which forecasting method is most appropriate to use for forecast the next time period? Calculate a forecast for the next time period with a 95% prediction interval. Comment on the bias of your forecasting model.

At first I thought the Montecarlo Simulation would be the best idea given the plot, but after some reviewing the lectures over and over again while looking at the plot, I have come to the conclusion that indeed in here there is a trend, even if that trend haves a lot of ups and downs you can see that the up and downs follow a pattern.

```
library(FNN)
library(readr)
occ <- read_csv("C:/Users/Guillermo/Desktop/occupancyratestimeseries (1).csv")

plot (occ$Period, occ$OccupancyRate,type='l',col='red',main='Linear relationship')
```




```
###Lineal Regression Trendline Forecast

md1<-data.frame(occ[161:166, ])

     md11<- lm(OccupancyRate ~ Period, md1)

     summary(md11)
```

```
##
## Call:
## lm(formula = OccupancyRate ~ Period, data = md1)
##
## Residuals:
##       1       2       3       4       5       6
##  1.4048 -0.9638  0.1676 -1.5010 -0.6695  1.5619
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -144.4448    55.5210  -2.602   0.0599 .
## Period         1.0686     0.3396   3.147   0.0346 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.42 on 4 degrees of freedom
## Multiple R-squared:  0.7123, Adjusted R-squared:  0.6404
## F-statistic: 9.903 on 1 and 4 DF,  p-value: 0.03462
```

```
     md11
```

```
##
## Call:
## lm(formula = OccupancyRate ~ Period, data = md1)
##
## Coefficients:
## (Intercept)       Period
##    -144.445        1.069
```

```
     md1$ft<-0
     md1$e<-0

#This line adds the row for the period 167
   md1[nrow(md1) + 1,] = c(167,NA,0,NA)

     for (i in 1:nrow(md1))

     {
       md1$ft[i]<-1.069*md1$Period[i] -144.445
       md1$e[i]<-md1[i,2]- md1$ft[i]
       #md3$e[12]<-0

     }


      frcst<- -144.445   + 1.069  *(167)
     frcst
```

```
## [1] 34.078
```

```
     # 95% Prediction interval for Linear Regression is a default
newdata<-data.frame(Period=167)
     predict(md11, newdata, interval="predict")
```

```
##        fit     lwr      upr
## 1 34.00667 28.6183 39.39504
```

```
     rownames(md1) <- 161:167

                                              ###Exponential Smoothing  Forecast

md2 <- read_csv("C:/Users/Guillermo/Desktop/occupancyratestimeseries (1).csv")

md2[nrow(md2) + 1,] = c(167,0,0,0)

a<-c(0.2)
md2$ft<- as.numeric(0)
md2$e<-as.numeric(0)

md2$ft[1]<-as.numeric(md2[1,2])

md2$ft[2]<-as.numeric(md2$ft[1] + (a*md2$e[1]))

md2$e[2]<-as.numeric(md2[2,2]- md2$ft[2])
```

```
for (i in 2:nrow(md2))
     {
     md2$ft[i]<-as.numeric(md2$ft[i-1]+ a*md2$e[i-1])
       md2$e[i]<-as.numeric(occ[i,2]- md2$ft[i])
      md2$e[167]<-0
}

#Polished data frame with the forecast for the period 167 and the rest of the data
md2
```

```
## # A tibble: 167 × 4
##    Period OccupancyRate       ft          e
## *   <dbl>         <dbl>    <dbl>      <dbl>
## 1       1          43.7 43.70000   0.000000
## 2       2          41.0 43.70000  -2.700000
## 3       3          36.5 43.16000  -6.660000
## 4       4          32.8 41.82800  -9.028000
## 5       5          24.5 40.02240 -15.522400
## 6       6          24.1 36.91792 -12.817920
## 7       7          26.0 34.35434  -8.354336
## 8       8          26.2 32.68347  -6.483469
## 9       9          30.7 31.38678  -0.686775
## 10     10          29.9 31.24942  -1.349420
## # ... with 157 more rows
```

```
n<-nrow(md2)
frcst167<-md2$ft[n]+a*md2$e[n]

#This is the exponential smoothing forecast for the period 167
frcst167
```

```
## [1] 33.25209
```

```
                                          ###Weighted Moving Average

md3<- read_csv("C:/Users/Guillermo/Desktop/occupancyratestimeseries (1).csv")

md3[nrow(md3) + 1,] = c(167,NA,0,NA)

md3$ft<-0
md3$e<-0
for (i in 4:nrow(md3))
{

  md3$ft[i]<-as.numeric(md3$OccupancyRate[i-1]*(4/6))+
    (md3$OccupancyRate[i-2]*(1/6))+(md3$OccupancyRate[i-3]*(1/6))
  md3$e[i]<-as.numeric(md3[i,2] - md3$ft[i])
  #md1$e[12]<-0

}
#Weighted Moving Average Forecast for the year 2017
md3$ft[167]
```

```
## [1] 33.08333
```

```
                          #Average Absolute Error for Linear Regression Trendline


md1$ae<-abs(md1$e)
mean(md1$ae[1:6])
```

```
## [1] 1.044833
```

```
                            #Average Absolute Error for Exponential Smoothing

md2$ae<-abs(md2$e)
mean(md2$ae[1:166])
```

```
## [1] 6.290716
```

```
                                 #Average Absolute Error for Weighted Moving Average

md3$ae<-abs(md3$e)
mean(md3$ae[4:166], na.rm=TRUE)
```

```
## [1] 4.654397
```

```
#Confidence Interval for Exponential Smoothing
sd1<-0.8*6.290716
sd1
```

```
## [1] 5.032573
```

```
se<-(5.032573)/(sqrt(166))
se
```

```
## [1] 0.3906034
```

```
##Confidence Interval for 95%
(33.25209)+(1.96*0.3906034)
```

```
## [1] 34.01767
```

```
##Confidence Interval for 95%
(33.25209)-(1.96*0.3906034)
```

```
## [1] 32.48651
```

```
##Confidence Interval for Weigthed Moving Average

sd1<-0.8*4.654397
sd1
```

```
## [1] 3.723518
```

```
se<-(3.723518)/(sqrt(166))
se
```

```
## [1] 0.289001
```

```
##Confidence Interval for 95%
(33.25209)+(1.96*0.289001)
```

```
## [1] 33.81853
```

```
##Confidence Interval for 95%
(33.25209)-(1.96*0.289001)
```

```
## [1] 32.68565
```

```
                          #Average Absolute Error for Linear Regression Trendline


md1$ae<-abs(md1$e)
mean(md1$ae[1:6])
```

```
## [1] 1.044833
```

```
                            #Average Absolute Error for Exponential Smoothing

md2$ae<-abs(md2$e)
mean(md2$ae[1:166])
```

```
## [1] 6.290716
```

```
                                 #Average Absolute Error for Weighted Moving Average

md3$ae<-abs(md3$e)
mean(md3$ae[4:166], na.rm=TRUE)
```

```
## [1] 4.654397
```

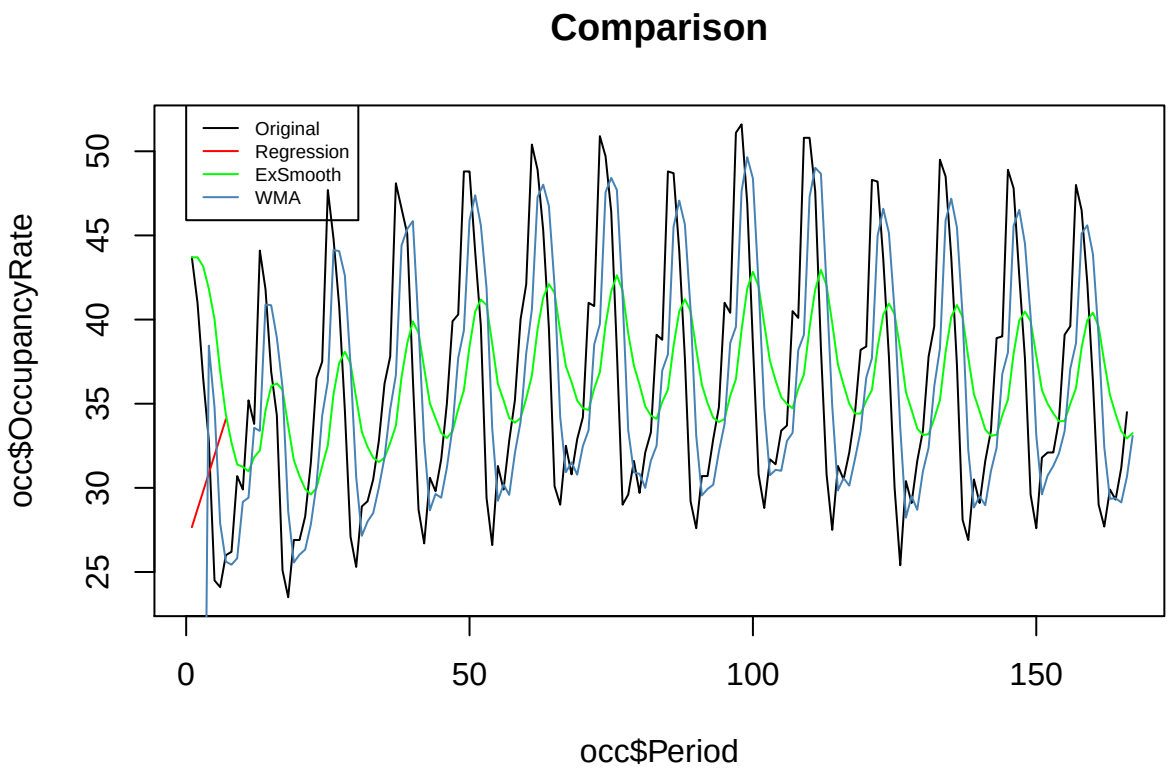
```
#MAD in a Data frame for easier recognition & and or manipulation

md1ae<-mean(md1$ae[1:6], na.rm=TRUE)
md2ae<-mean(md2$ae[1:166])
md3ae<- mean(md3$ae[4:166])

small <- data.frame(Method =
                      c("Lineal Regression Trendline",
                        "Exponential Smoothing","Weighted Mean"
                              ),
                MAD = c(md1ae,md2ae,md3ae))
#This line will give us the smalle mean absolute error (MAD)
small[which.min(small$MAD), ]
```

```
##                         Method      MAD
## 1 Lineal Regression Trendline 1.044833
```

```
#This line gives us the whole data frame for us to see it clearly
small
```

```
##                         Method      MAD
## 1 Lineal Regression Trendline 1.044833
## 2       Exponential Smoothing 6.290716
## 3               Weighted Mean 4.654397
```

```
#PLOT
plot (occ$Period, occ$OccupancyRate,type="l",col="black",main= "Comparison")
lines(md1$ft,type="l",col="red")
lines(md2$ft,type="l",col="green")
lines(md3$ft,type="l",col="steel blue")
legend(0,52.75, legend=c("Original", "Regression", "ExSmooth", "WMA"),col=c("black", "red","green", "st
```



The bias here I think is because of the seasonality that is present in the data set, but not yet explained. If we see the plot we see that the there is a trend of cycles of up and down, but yet there is not a clear time period defined.

I think the best choice to forecast the enxt period would be the linear regression and just pick un the last period. The issue is that if we would like to forecast for a longer period the lineal regression would only go upwards and not down, hence why maybe if we were to forecast a longer period of time we would need to select a cycle.

Note that I havent been able to move the Regression Line in the plot to the left but it seems it would give us a better forecast than any other model. Looking also at the MAD we can see that the Linear Regression performs better when picking a certain period of time.