Exercise 11.2

Dipika Sharma

June 5, 2021

Add Citations

- R for Everyone (Lander 2014)
- Discovering Statistics Using R (Field, Miles, and Field 2012)

In this problem, you will use the nearest neighbors algorithm to fit a model on two simplified datasets. The first dataset (found in binary-classifier-data.csv) contains three variables; label, x, and y. The label variable is either 0 or 1 and is the output we want to predict using the x and y variables (You worked with this dataset last week!). The second dataset (found in trinary-classifier-data.csv) is similar to the first dataset except that the label variable can be 0, 1, or 2.

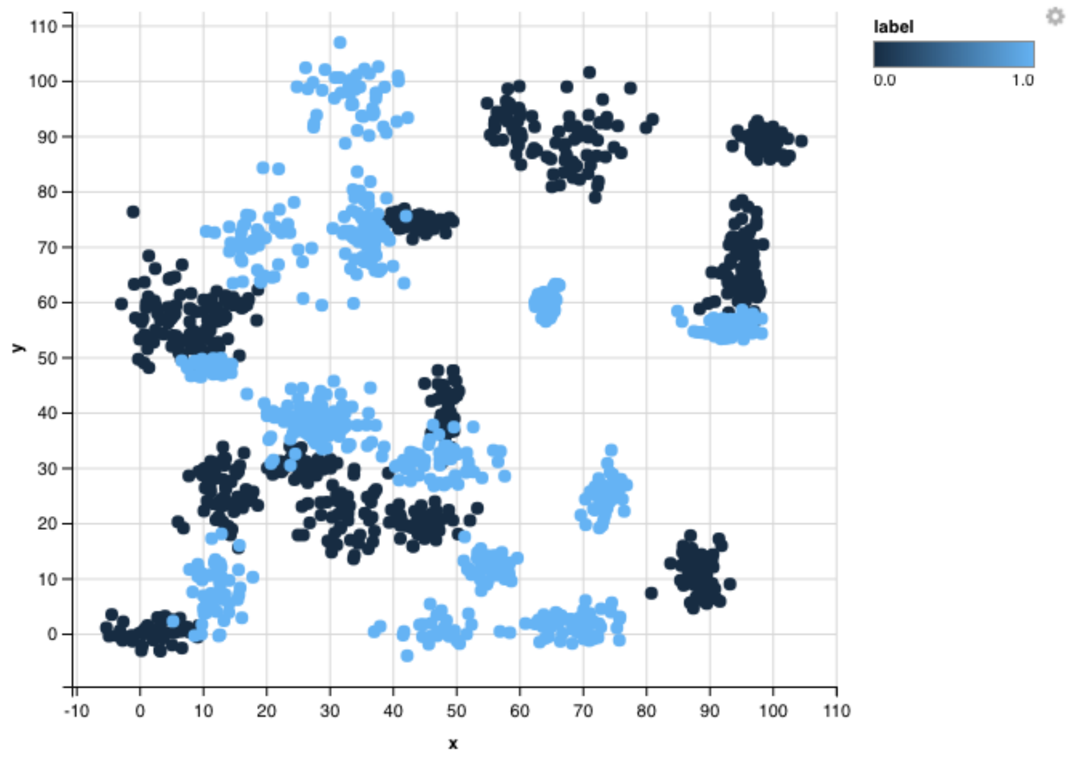
```
## Set the working directory to the root of your DSC 520 directory
setwd("/Users/dipikasharma/R_Projects/DSC520")

## Load the data to df
Binary_df <- read.csv("data/binary-classifier-data.csv")
head(Binary_df)

##   label        x        y
## 1     0 70.88469 83.17702
## 2     0 74.97176 87.92922
## 3     0 73.78333 92.20325
## 4     0 66.40747 81.10617
## 5     0 69.07399 84.53739
## 6     0 72.23616 86.38403

trinary <- read.csv("data/trinary-classifier-data.csv")
head(trinary)

##   label        x        y
## 1     0 30.08387 39.63094
## 2     0 31.27613 51.77511
## 3     0 34.12138 49.27575
## 4     0 32.58222 41.23300
## 5     0 34.65069 45.47956
## 6     0 33.80513 44.24656
```

Note that in real-world datasets, your labels are usually not numbers, but text-based descriptions of the categories (e.g. spam or ham). In practice, you will encode categorical variables into numeric values.

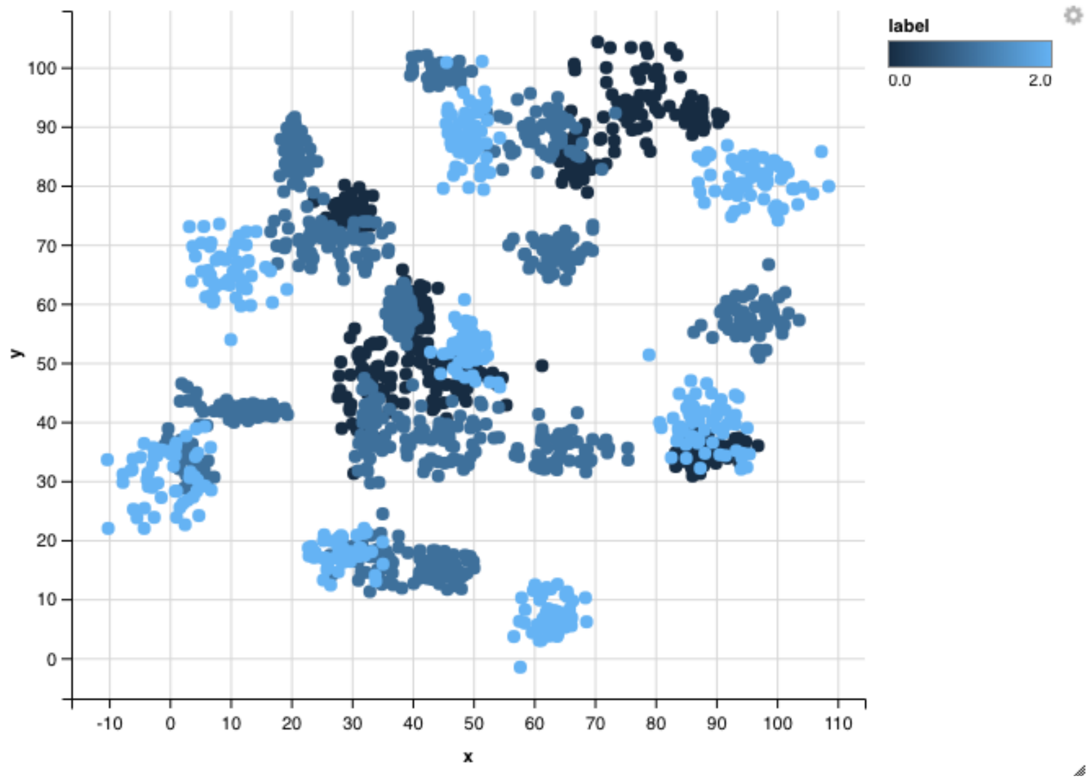
i. Plot the data from each dataset using a scatter plot.

Using the scatterplot we can see the correlation between labels and variable x and y.

```
Binary_df %>% ggvis(~x, ~y, fill = ~label) %>% layer_points()
```



```
trinary %>% ggvis(~x, ~y, fill = ~label) %>% layer_points()
```



The k nearest neighbors algorithm categorizes an input value by looking at the labels for the k nearest points and assigning a category based on the most common label. In this problem, you will determine which points are nearest by calculating the Euclidean distance between two points. As a refresher, the Euclidean distance between two points:

```
set.seed(123)
dat.d <- sample(1:nrow(Binary_df),size=nrow(Binary_df)*0.7,replace = FALSE) #
random selection of 70% data.
dat.d_trinary <- sample(1:nrow(trinary),size=nrow(trinary)*0.7,replace = FALS
E)

train.Binary <- Binary_df[dat.d,] # 70% training data
test.Binary <- Binary_df[-dat.d,] # remaining 30% test data

train.trinary <- trinary[dat.d_trinary,] # 70% training data
test.trinary <- trinary[-dat.d_trinary,] # remaining 30% test data

#Creating separate dataframe for 'Creditability' feature which is our target.
```

```
train.binary_labels <- Binary_df[dat.d,1]
test.binary_labels <-Binary_df[-dat.d,1]

train.trinary_labels <- trinary[dat.d_trinary,1]
test.trinary_labels <-trinary[-dat.d_trinary,1]


#Find the number of observation
NROW(train.binary_labels)

## [1] 1048

sqrt(NROW(train.binary_labels) )

## [1] 32.37283
```

So, we have 1048 observations in our training data set. The square root of 1048 is around 32.37, therefore ww will create two models. One with ‘K’ value as 32 and the other model with a ‘K’ value as 33

```
NROW(train.trinary_labels)

## [1] 1097

sqrt(NROW(train.trinary_labels) )

## [1] 33.12099
```

For Trinary data set, So, we have 1097 observations in our training data set. The square root of 1097 is around 33.12, therefore we will create two models. One with ‘K’ value as 33 and the other model with a ‘K’ value as 34

```
library(class)
knn.32 <- knn(train=train.Binary, test=test.Binary, cl=train.binary_labels, k
=32)
knn.33 <- knn(train=train.Binary, test=test.Binary, cl=train.binary_labels, k
=33)

knn.33.trinary <- knn(train=train.trinary, test=test.trinary, cl=train.trinar
y_labels, k=33)
knn.34.trinary <- knn(train=train.trinary, test=test.trinary, cl=train.trinar
y_labels, k=34)
```

Fitting a model is when you use the input data to create a predictive model. There are various metrics you can use to determine how well your model fits the data. For this problem, you will focus on a single metric, accuracy. Accuracy is simply the percentage of how often the model predicts the correct result. If the model always predicts the correct result, it is 100% accurate. If the model always predicts the incorrect result, it is 0% accurate.

Calculate the proportion of correct classification for k = 32, 33 for binary data set

```
ACC.32 <- 100 * sum(test.binary_labels == knn.32)/NROW(test.binary_labels)
ACC.33 <- 100 * sum(test.binary_labels == knn.33)/NROW(test.binary_labels)
ACC.32

## [1] 97.77778

ACC.33

## [1] 97.55556
```

As shown above, the accuracy for K = 32 is 97.55 and for K = 33 it is 97.55

Calculate the proportion of correct classification for k = 33, 34 for Trinary dataset

```
ACC.33.trinary <- 100 * sum(test.trinary_labels == knn.33.trinary)/NROW(test.
trinary_labels)
ACC.34.trinary <- 100 * sum(test.trinary_labels == knn.34.trinary)/NROW(test.
trinary_labels)
ACC.33.trinary

## [1] 87.26115

ACC.34.trinary

## [1] 86.41189
```

As shown above, the accuracy for K = 33 is 84.71 and for K = 34 it is 84.92.

We can also check the predicted outcome against the actual value in tabular form:

Check prediction against actual value in tabular form for k=26 for Binary dataset

```
table(knn.32 ,test.binary_labels)

##       test.binary_labels
## knn.32   0   1
##      0 227   6
##      1   4 213

knn.32

##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [112] 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [149] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0
## [186] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [223] 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1
1 1 1
## [260] 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [445] 1 1 1 1 1 1
## Levels: 0 1
```

Check prediction against actual value in tabular form for k=27 for Binary dataset

```
table(knn.33 ,test.binary_labels)

##       test.binary_labels
## knn.33   0   1
##       0 226   6
##       1   5 213
```

```
knn.33
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [112] 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [149] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0
## [186] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [223] 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1
1 1 1
## [260] 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [445] 1 1 1 1 1 1
## Levels: 0 1
```

Check prediction against actual value in tabular form for k=26 for Trinary dataset

```
table(knn.33.trinary ,test.trinary_labels)
```

```
##                 test.trinary_labels
## knn.33.trinary   0   1   2
##                0  99  16   9
##                1  14 186   2
##                2   5  14 126
```

```
knn.33.trinary
```

```
##   [1] 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0
0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
```

```
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 1 1 0 0 1 0 0
2 0 1
## [112] 2 0 0 2 0 0 2 1 1 1 1 1 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 2
## [149] 1 1 2 1 1 2 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 2 2 2 2 1 1 1
1 1 1
## [223] 1 1 1 1 1 1 0 1 1 2 1 1 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1
1 1 1
## [260] 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [297] 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 0 1
1 0 1
## [334] 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [371] 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 2 0 2 2 2 2 0 0 0 0 2 0 2 2 2 2 2 2 2
2 2 2
## [408] 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [445] 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2
## Levels: 0 1 2
```

Check prediction against actual value in tabular form for k=27 for Trinary dataset

```
table(knn.34.trinary ,test.trinary_labels)
```

```
##                 test.trinary_labels
## knn.34.trinary   0   1   2
##                0  98  16  12
##                1  15 186   2
##                2   5  14 123
```

```
knn.34.trinary
```

```
##   [1] 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 1 1 0 0 1 2 0
2 0 1
## [112] 2 0 0 2 0 0 2 1 1 1 1 1 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 2
## [149] 1 1 2 1 1 2 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 2 2 2 2 1 1 1
1 1 1
## [223] 1 1 1 1 1 1 0 1 1 2 1 1 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1
```

```
1 1 1
## [260] 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [297] 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 0 1
1 0 1
## [334] 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
0 2 2
## [371] 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 2 0 2 0 2 0 0 2 0 0 0 0 2 0 2 2 2 2 2 2 2 2
2 2 2
## [408] 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [445] 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## Levels: 0 1 2
```

We can also use the confusion matrix to calculate the accuracy. To do this we must first install the infamous Caret package:

```
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:ggvis':
##
##     resolution

confusionMatrix(table(knn.32 ,test.binary_labels))

## Confusion Matrix and Statistics
##
##       test.binary_labels
## knn.32   0   1
##      0 227   6
##      1   4 213
##
##                Accuracy : 0.9778
##                  95% CI : (0.9595, 0.9893)
##     No Information Rate : 0.5133
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9555
##
##  Mcnemar's Test P-Value : 0.7518
##
##             Sensitivity : 0.9827
##             Specificity : 0.9726
##          Pos Pred Value : 0.9742
##          Neg Pred Value : 0.9816
##              Prevalence : 0.5133
##          Detection Rate : 0.5044
##    Detection Prevalence : 0.5178
##       Balanced Accuracy : 0.9776
##
##        'Positive' Class : 0
##
```

For Binary dataset, we can see that our model predicts the outcome with an accuracy of 97.5% which is very good with small data set

```
confusionMatrix(table(knn.33.trinary ,test.trinary_labels))

## Confusion Matrix and Statistics
##
##                test.trinary_labels
## knn.33.trinary   0   1   2
##              0  99  16   9
##              1  14 186   2
##              2   5  14 126
##
## Overall Statistics
##
##                Accuracy : 0.8726
##                  95% CI : (0.8391, 0.9014)
##     No Information Rate : 0.4586
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8034
##
##  Mcnemar's Test P-Value : 0.01636
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity            0.8390   0.8611   0.9197
## Specificity            0.9292   0.9373   0.9431
## Pos Pred Value         0.7984   0.9208   0.8690
## Neg Pred Value         0.9452   0.8885   0.9663
## Prevalence             0.2505   0.4586   0.2909
## Detection Rate         0.2102   0.3949   0.2675
## Detection Prevalence   0.2633   0.4289   0.3079
## Balanced Accuracy      0.8841   0.8992   0.9314
```

For Trinary dataset, we can see that our model predicts the outcome with an accuracy of 84.71% which is very good with small data set

Fit a k nearest neighbors' model for each dataset for k=3, k=5, k=10, k=15, k=20, and k=25. Compute the accuracy of the resulting models for each value of k. Plot the results in a graph where the x-axis is the different values of k and the y-axis is the accuracy of the model.

```
library("kknn")

##
## Attaching package: 'kknn'

## The following object is masked from 'package:caret':
##
##     contr.dummy

library(class)

i=3                         # declaration to initiate for loop
k.optm=1   # declaration to initiate for loop
val <- c(3,5,10,15,20,25)
for (i in val){
  knn.mod <-  knn(train=train.Binary, test=test.Binary, cl=train.binary_label
s, k=i)
  k.optm[i] <- 100 * sum(test.binary_labels == knn.mod)/NROW(test.binary_labe
ls)
  k=i
  cat(k,'=',k.optm[i],'\n')       # to print % accuracy
}

## 3 = 98
## 5 = 97.55556
## 10 = 98.22222
## 15 = 97.55556
## 20 = 97.77778
## 25 = 98.44444
```

From the output you can see that for K = 25 in Binary dataset, we achieve the maximum accuracy of 98.44%.

```
i=3                             # declaration to initiate for loop
k.optm.trinary=1   # declaration to initiate for loop
val <- c(3,5,10,15,20,25)
for (i in val){
  knn.mod.trinary <-  knn(train=train.trinary, test=test.trinary, cl=train.tr
inary_labels, k=i)
  k.optm.trinary[i] <- 100 * sum(test.trinary_labels == knn.mod)/NROW(test.tr
inary_labels)
  k=i
  cat(k,'=',k.optm[i],'\n')       # to print % accuracy
}
```

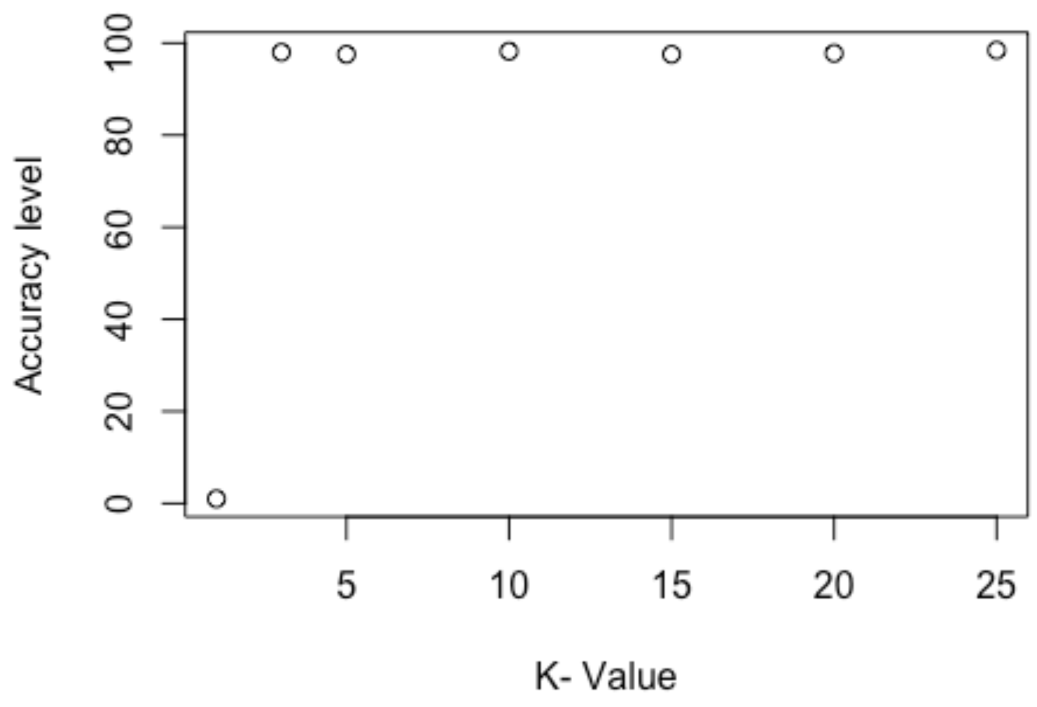
```
## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple o
f
## shorter object length

## 3 = 98

## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## 5 = 97.55556

## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## 10 = 98.22222

## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## 15 = 97.55556
## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## 20 = 97.77778

## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## Warning in `==.default`(test.trinary_labels, knn.mod): longer object lengt
h is
## not a multiple of shorter object length

## 25 = 98.44444
```

From the output you can see that for K = 3 in Trinary dataset, we achieve the maximum accuracy of 90.23%.

We can also represent this graphically, like so:

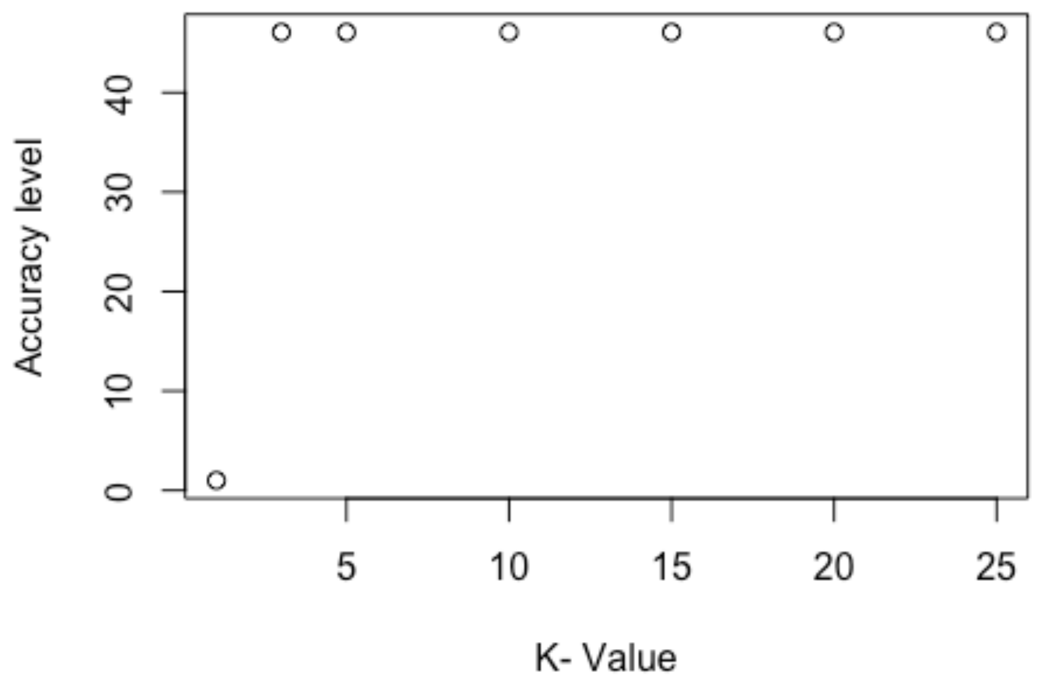
Accuracy plot Binary Dataset

```
plot(k.optm, type="b", xlab="K- Value",ylab="Accuracy level")
```



Accuracy plot Trinary Dataset

```
plot(k.optm.trinary, type="b", xlab="K- Value",ylab="Accuracy level")
```



Looking back at the plots of the data, do you think a linear classifier would work well on these datasets?

Linear Classifier is used for making classification decision based on the value characteristic, to identify which group or class value belongs to. We mainly use linear classifier for problems with many variables or features. For Binary and Trinary dataset, linear classifier will not work well as they they few variables and looking at the plots we do not see huge variance in accuracy with different k-value.

How does the accuracy of your logistic regression classifier from last week compare? Why is the accuracy different between these two methods?

The accuracy of logistic regression classifier was 57.5 % only where as with KNN the accuracy of an model 98.44%. Both model are good, it depends on type of problem we are working on. Binary data set have categorical variable which work best with non-parametric model and KNN is non-parametric model where as Logistic regression work best on parametric model.

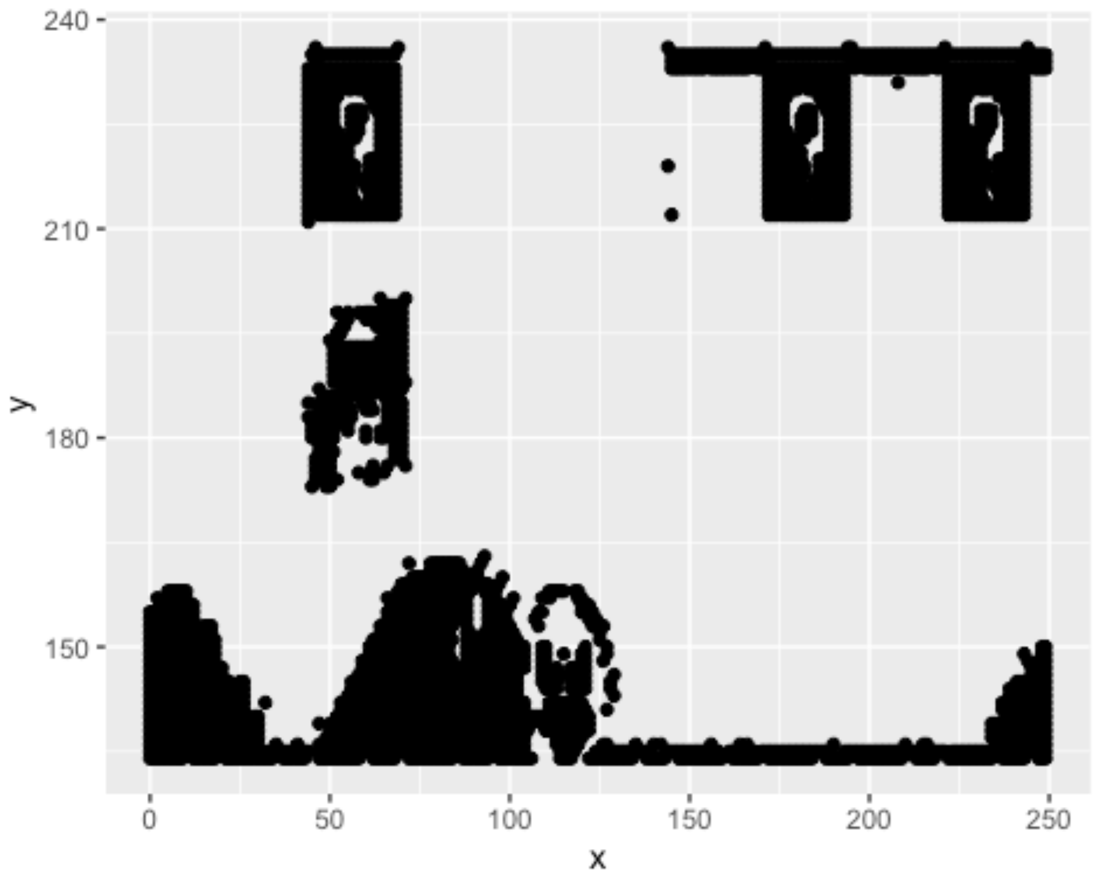
Clustering

In this problem, you will use the k-means clustering algorithm to look for patterns in an unlabeled dataset. The dataset for this problem is found at data/clustering-data.csv.

```r
## Set the working directory to the root of your DSC 520 directory
setwd("/Users/dipikasharma/R_Projects/DSC520")

## Load the `data/r4ds/heights.csv` to
clustering_df <- read.csv("data/clustering-data.csv")
```

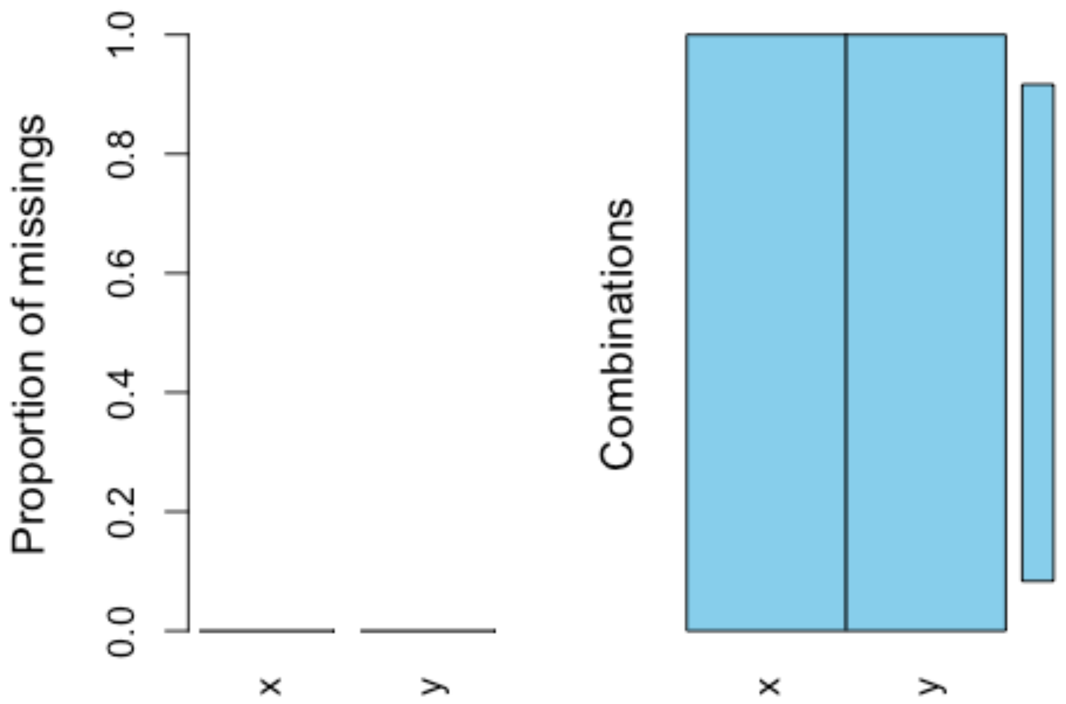
Plot the dataset using a scatter plot.

```r
library(ggplot2)
ggplot(clustering_df, aes(x = x, y = y)) +
  geom_point()
```



```
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

glimpse(clustering_df)

## Rows: 4,022
## Columns: 2
## $ x <int> 46, 69, 144, 171, 194, 195, 221, 244, 45, 47, 48, 49, 50, 51, 52
, 53…
## $ y <int> 236, 236, 236, 236, 236, 236, 236, 236, 235, 235, 235, 235, 235,
235…

summary(clustering_df)
```

```
##        x               y        
##  Min.   :  0.0   Min.   :134.0  
##  1st Qu.: 56.0   1st Qu.:141.0  
##  Median : 82.0   Median :154.0  
##  Mean   :109.6   Mean   :175.7  
##  3rd Qu.:180.0   3rd Qu.:218.0  
##  Max.   :249.0   Max.   :236.0

# 'aggr' plots the amount of missing/imputed values in each column
library(VIM)

## Loading required package: colorspace

## Loading required package: grid

## VIM is ready to use.

## Suggestions and bug-reports can be submitted at: https://github.com/statis
tikat/VIM/issues

## 
## Attaching package: 'VIM'

## The following object is masked from 'package:datasets':
## 
##     sleep

aggr(clustering_df)
```



As you can see, the dataset has no missing values.

```
k2 <- kmeans(clustering_df, centers = 3, nstart = 25)
str(k2)

## List of 9
##  $ cluster     : int [1:4022] 1 1 2 2 2 2 2 2 1 1 ...
##  $ centers     : num [1:3, 1:2] 38.7 209.2 87.3 180.5 205.2 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "1" "2" "3"
##   .. ..$ : chr [1:2] "x" "y"
##  $ totss       : num 28608985
##  $ withinss    : num [1:3] 2400602 2798177 815599
##  $ tot.withinss: num 6014378
##  $ betweenss   : num 22594607
##  $ size        : int [1:3] 1347 1273 1402
##  $ iter        : int 2
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"

print(k2)

## K-means clustering with 3 clusters of sizes 1347, 1273, 1402
##
```

```
## Cluster means:
##           x        y
## 1  38.73348 180.4610
## 2 209.16575 205.1791
## 3  87.29886 144.3702
##
## Clustering vector:
##    [1] 1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2
2 2 2
##   [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##   [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [149] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [186] 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
2 2 2
##  [223] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [260] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [297] 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
2 2 2
##  [334] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [371] 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [408] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1
1 1 1
##  [445] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [482] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
2 2 2
##  [519] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2
2 2 2
##  [556] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 2
##  [593] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1
1 1 1
##  [630] 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 1 1
##  [667] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [704] 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [741] 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2
2 2 2
##  [778] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
1 2 2
##  [815] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1
1 1 1
##  [852] 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 1
##  [889] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
##  [926] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [963] 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [1000] 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
2 2 2
## [1037] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1
1 1 1
## [1074] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [1111] 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
2 2 2
## [1148] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1
## [1185] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [1222] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 2 2
## [1259] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1
1 1 1
## [1296] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [1333] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1
1 1 1
## [1370] 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [1407] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [1444] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [1481] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [1518] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [1555] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [1592] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [1629] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [1666] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [1703] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
1 1 1
## [1740] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3
3 3 3
## [1777] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [1814] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 3 3 3 3 3 3
3 3 3
## [1851] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 3 3 3
3 3 3
## [1888] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 3
3 3 3
## [1925] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1
1 1 1
## [1962] 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 1
## [1999] 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2036] 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2073] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [2110] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1
1 1 1
## [2147] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2184] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
3 3 3
## [2221] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2258] 3 3 3 3 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2295] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
2 2 2
## [2332] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2369] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2
1 1 1
## [2406] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2443] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
2 2 2
## [2480] 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2517] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2554] 3 3 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3
3 3 3
## [2591] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2628] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
```

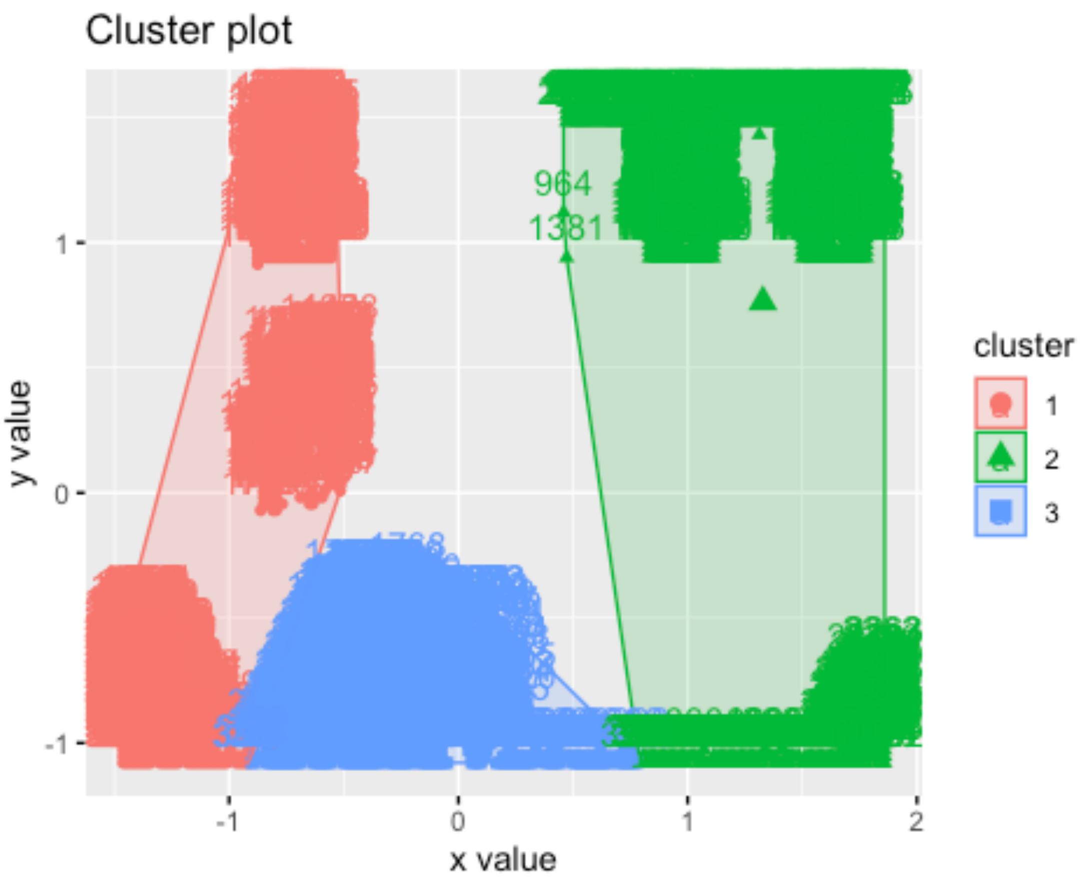
```
1 1 1
## [2665] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2702] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2
2 2 2
## [2739] 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3
3 3 3
## [2776] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2813] 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [2850] 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2887] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 2 2
## [2924] 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [2961] 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [2998] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2
2 2 2
## [3035] 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
3 3 3
## [3072] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3109] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2
2 2 2
## [3146] 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [3183] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3220] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 2 2
## [3257] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [3294] 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3331] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3368] 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [3405] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3442] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3479] 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [3516] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3553] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

```
3 3 3
## [3590] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1
1 1 1
## [3627] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 3 3
## [3664] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3701] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3738] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [3775] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [3812] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [3849] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3886] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3923] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3
## [3960] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [3997] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 2400602.0 2798177.2  815598.7
##  (between_SS / total_SS =  79.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withi
nss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

If we print the results we'll see that our groupings resulted in 3 cluster sizes of 1347, 1402 and 1273.

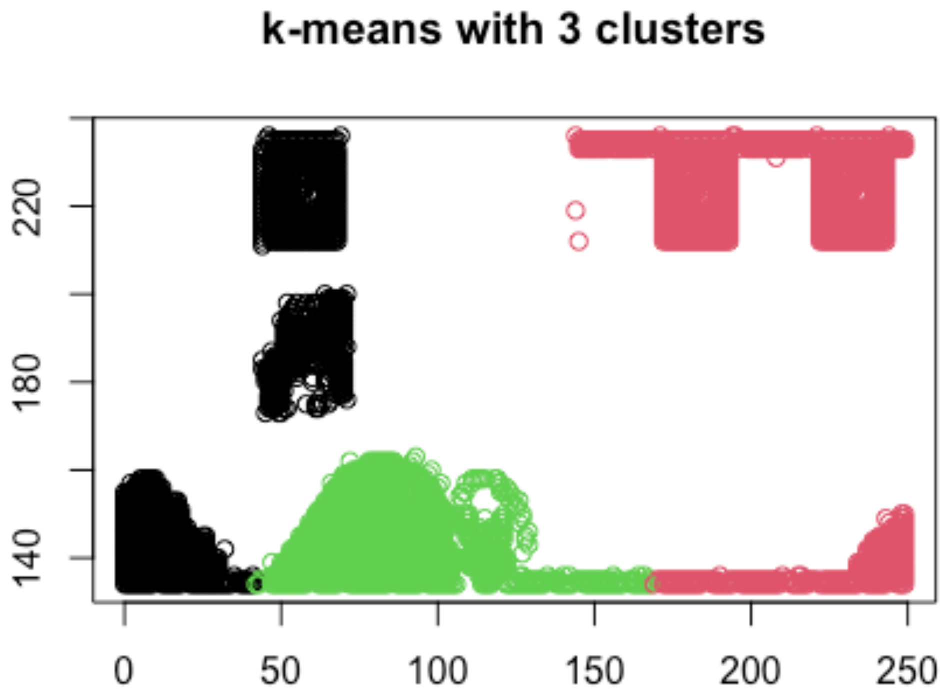
We can also view our results by using fviz_cluster

```
library(factoextra)

## Welcome! Want to learn more? See two factoextra-related books at https://g
oo.gl/ve3WBa

fviz_cluster(k2, data = clustering_df)
```



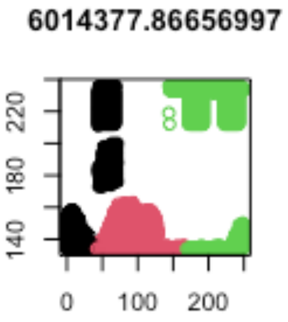
Alternatively, you can use standard pairwise scatter plots to illustrate the clusters compared to the original variables.

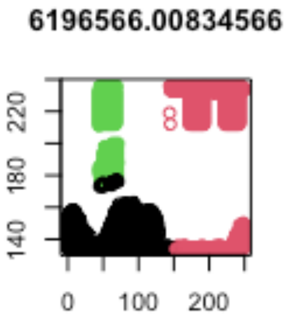
```
plot(clustering_df,col=k2$cluster, main="k-means with 3 clusters", xlab="", y
lab="")
```

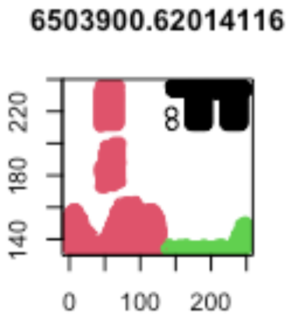


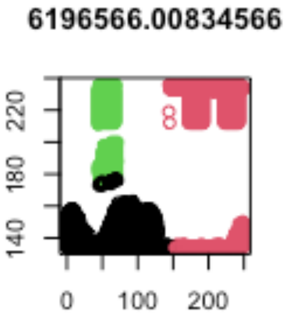
Fit the dataset using the k-means algorithm from k=2 to k=12. Create a scatter plot of the resultant clusters for each value of k.

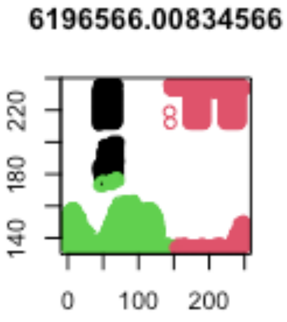
```
# Set up 2 x 3 plotting grid
par(mfrow = c(2, 3))

# Set seed
set.seed(1)

for(i in 2:12) {
  # Run kmeans() on x with three clusters and one start
  km.out <- kmeans(clustering_df, centers=3, nstart=1)

  # Plot clusters
  plot(clustering_df, col = km.out$cluster,
       main = km.out$tot.withinss,
       xlab = "", ylab = "")
}
```

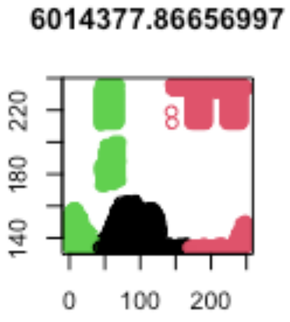

6014377.86656997
140
180
220
0
100
200

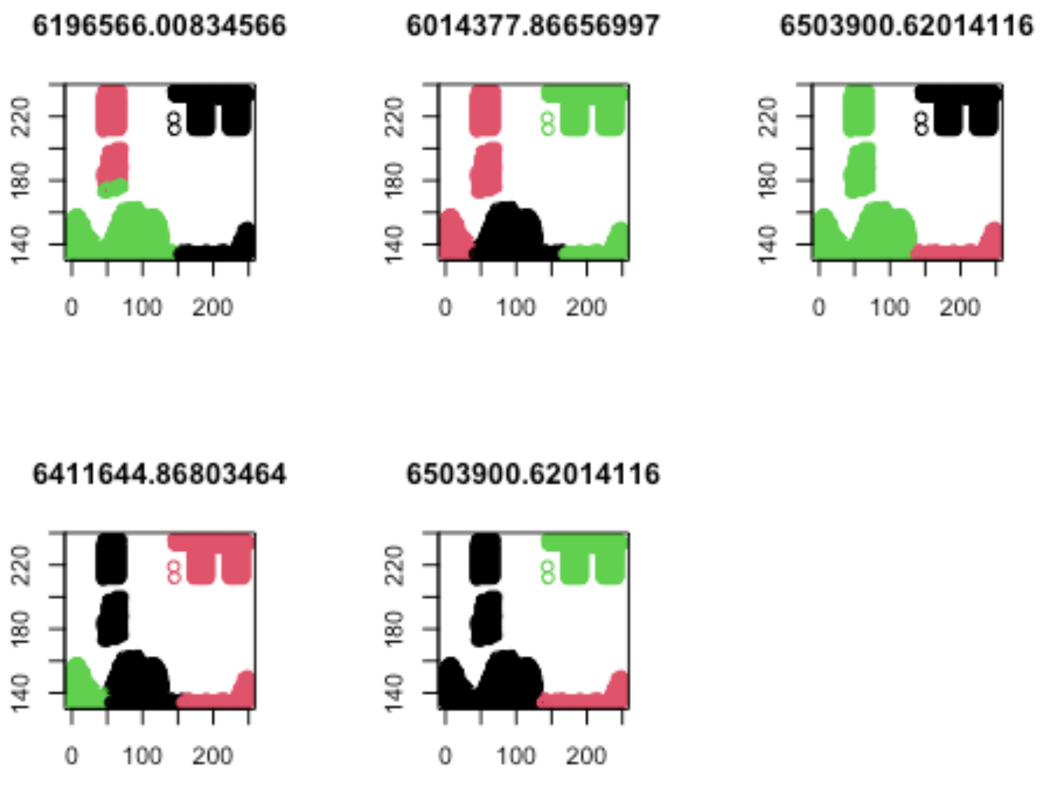

6196566.00834566
140
180
220
0
100
200


6503900.62014116
140
180
220
0
100
200


6196566.00834566
140
180
220
0
100
200


6196566.00834566
140
180
220
0
100
200


6014377.86656997
140
180
220
0
100
200



As k-means is an unsupervised algorithm, you cannot compute the accuracy as there are no correct values to compare the output to. Instead, you will use the average distance from the center of each cluster as a measure of how well the model fits the data. To calculate this metric, simply compute the distance of each data point to the center of the cluster it is assigned to and take the average value of all of those distances.

```
kmeans.wss.k <- function(clustering_df, k){
  km = kmeans(clustering_df, k)
  return (km$tot.withinss)
}

kmeans.wss.k(clustering_df,5)

## [1] 2171613

kmeans.wss.k(clustering_df,10)

## [1] 554716.4
```

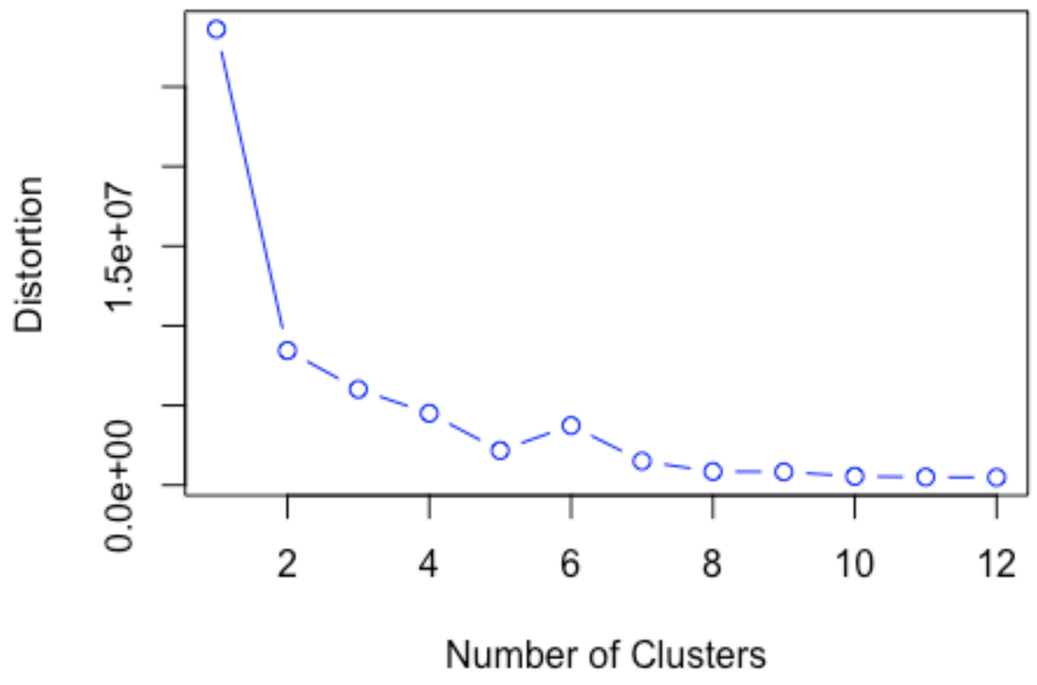
It can be seen that as the value of K increases, distortion decreases.

```
kmeans.dis <- function(clustering_df, maxk){
   dis=(nrow(clustering_df)-1)*sum(apply(clustering_df,2,var))
  dis[2:maxk]=sapply (2:maxk, kmeans.wss.k, clustering_df=clustering_df)
   return(dis)}
 maxk = 12
 dis = kmeans.dis(clustering_df, maxk);
 dis
```

```
##  [1] 28608984.7  8443681.1  6014377.9  4509358.0  2171612.8  3741267.6
##  [7]  1505729.6   853160.1   833518.5   554632.2   504767.0   482688.3
```

Calculate this average distance from the center of each cluster for each value of k and plot it as a line chart where k is the x-axis and the average distance is the y-axis.

```
 plot(1:maxk, dis, type='b', xlab="Number of Clusters",
        ylab="Distortion",
        col="blue")
```



One way of determining the “right” number of clusters is to look at the graph of k versus average distance and finding the “elbow point.” Looking at the graph you generated in the previous example, what is the elbow point for this dataset?

This is the plot between ‘k,’ the number of clusters and the ‘totwithinss’ (or distortion) for each value of k. You can see when the number of cluster is less, there is a gradual decrease in distortion but as we keep on increasing the value of k, the rate of reduction of distortion values becomes constant. This value of k beyond which the distortion rate becomes constant is the optimal value. Here k=7.

References

Field, A., J. Miles, and Z. Field. 2012. *Discovering Statistics Using r*. SAGE Publications. https://books.google.com/books?id=wd2K2zC3swIC.

Lander, J. P. 2014. *R for Everyone: Advanced Analytics and Graphics*. Addison-Wesley Data and Analytics Series. Addison-Wesley. https://books.google.com/books?id=3eBVAgAAQBAJ.