HW #2 - KELLY “SCOTT” SIMS

Question 3.1

Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier: (a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional)

```
#read in the data
data <- read.csv('credit_card_data-headers.csv', header = TRUE)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(e1071)
#set the seed for reproducibility
set.seed(42)

#split the data into independent and dependent variables
X <- as.matrix(data[1:10])
y <- as.matrix(data[11])

#setup the model training parameters for crossvalidation
train_control <- trainControl(method = 'repeatedcv', number = 10, repeats =3)
#train the model using crossvalidation, scale the data first and report on the accuracy of each metric 
model <- train(X,as.factor(y),method='knn', trControl = train_control, metric = 'Accuracy', tuneGrid = 

model
```

```
## k-Nearest Neighbors
##
## 654 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 589, 588, 588, 589, 588, 589, ...
## Resampling results across tuning parameters:
##
##   k    Accuracy   Kappa
##    1   0.8083528  0.6120403
##    2   0.7987646  0.5922332
##    3   0.8302953  0.6568654
##    4   0.8374048  0.6720875
##    5   0.8476379  0.6934779
##    6   0.8430692  0.6842709
##    7   0.8466045  0.6922552
##    8   0.8445221  0.6883149
##    9   0.8353924  0.6698339
##   10   0.8389510  0.6764702
##   11   0.8389510  0.6765608
##    12  0.8364103  0.6709513
##    13  0.8333566  0.6647722
##    14  0.8368998  0.6721899
##    15  0.8267211  0.6513536
##    16  0.8344056  0.6662047
##    17  0.8292774  0.6564478
##    18  0.8338617  0.6657675
##    19  0.8364103  0.6704028
##    20  0.8410023  0.6801204
##    21  0.8415074  0.6804596
##    22  0.8430070  0.6830984
##    23  0.8389588  0.6747284
##    24  0.8322922  0.6605916
##    25  0.8358664  0.6676028
##    26  0.8353225  0.6663346
##    27  0.8399689  0.6752910
##    28  0.8409868  0.6772831
##    29  0.8363947  0.6678796
##    30  0.8358819  0.6668471
##    31  0.8384305  0.6715414
##    32  0.8414841  0.6776118
##    33  0.8425019  0.6796728
##    34  0.8404895  0.6755235
##    35  0.8430303  0.6807431
##    36  0.8435664  0.6818444
##    37  0.8409946  0.6766459
##    38  0.8414841  0.6774735
##    39  0.8430225  0.6806196
##    40  0.8435276  0.6816097
##    41  0.8430458  0.6807284
##    42  0.8450816  0.6848096
##    43  0.8425330  0.6796337
##    44  0.8450738  0.6848330
##    45  0.8450738  0.6845180
##    46  0.8445610  0.6834185
##    47  0.8440326  0.6822374
##    48  0.8440326  0.6821114
##    49  0.8445532  0.6830415
##    50  0.8430070  0.6800068
##    51  0.8420047  0.6778078
##    52  0.8405051  0.6747179
##    53  0.8384538  0.6704750
##    54  0.8379332  0.6691988
##    55  0.8369153  0.6673936
##    56  0.8389510  0.6715862
##    57  0.8399845  0.6735964
##    58  0.8404895  0.6745544
##    59  0.8414996  0.6766082
##    60  0.8399845  0.6734731
##    61  0.8389666  0.6714261
##    62  0.8399767  0.6734691
##    63  0.8374281  0.6681535
##    64  0.8394716  0.6722666
##    65  0.8384460  0.6702164
##    66  0.8389588  0.6711898
##    67  0.8409946  0.6753399
##    68  0.8399922  0.6732074
##    69  0.8394716  0.6722306
##    70  0.8384615  0.6700710
##    71  0.8384693  0.6700252
##    72  0.8389744  0.6711075
##    73  0.8394794  0.6721101
##    74  0.8374514  0.6678158
##    75  0.8374514  0.6678943
##    76  0.8389744  0.6709841
##    77  0.8369386  0.6667260
##    78  0.8374437  0.6677404
##    79  0.8379643  0.6688551
##    80  0.8338928  0.6605701
##    81  0.8364336  0.6656431
##    82  0.8364336  0.6657025
##    83  0.8354079  0.6634702
##    84  0.8369542  0.6666796
##    85  0.8354079  0.6634995
##    86  0.8359285  0.6645414
##    87  0.8384615  0.6697292
##    88  0.8364180  0.6654593
##    89  0.8359285  0.6646191
##    90  0.8369308  0.6665633
##    91  0.8369308  0.6665930
##    92  0.8369308  0.6665991
##    93  0.8369153  0.6663262
##    94  0.8359130  0.6642467
##    95  0.8348796  0.6621404
##    96  0.8354002  0.6632015
##    97  0.8343667  0.6610313
##    98  0.8333489  0.6589283
##    99  0.8333489  0.6588908
##   100  0.8338384  0.6598223
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

***Analysis** Using cross validation and with 10 folds, a knn model was trained, investigating k-nearest neighbor values from 1 neighbor to 100 neighbors. Ultimately during CV, 5 nearest neighbors resulted in the model with the best accuracy on the data

(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM;

```r
library(caret)
data <- read.csv('credit_card_data-headers.csv', header = TRUE)
data[['R1']] = factor(data[["R1"]])

set.seed(49)
#Create a partion for 70% Training Data
intrain <- createDataPartition(y=data$R1, p = 0.7, list=FALSE)
train_set <- data[intrain,]
```

```r
not_train_set <- data[-intrain,]

#For the remaining 30% of data,split it 50/50 between a train and validation set
invalidate <- createDataPartition(y=not_train_set$R1, p=0.5, list=FALSE)
test_set <- not_train_set[invalidate,]
valid_set <- not_train_set[-invalidate,]

#train model and predict
model <- train(R1~., data=train_set, method ='knn', preProcess = c('center','scale'), tuneGrid = expand
val_pred <- predict(model, valid_set)
test_pred <- predict(model, test_set)

print(model)
```

```
## k-Nearest Neighbors
##
## 459 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 459, 459, 459, 459, 459, 459, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.7981305  0.5923394
##    2  0.7965999  0.5887534
##    3  0.7969241  0.5900716
##    4  0.8020342  0.6007195
##    5  0.8189725  0.6351510
##    6  0.8229729  0.6434208
##    7  0.8262140  0.6502153
##    8  0.8305380  0.6589893
##    9  0.8297149  0.6572141
##   10  0.8330612  0.6638179
##   11  0.8320106  0.6615264
##   12  0.8354320  0.6683371
##   13  0.8347281  0.6671354
##   14  0.8358289  0.6694443
##   15  0.8369958  0.6716752
##   16  0.8377146  0.6730455
##   17  0.8382396  0.6742760
##   18  0.8409922  0.6797860
##   19  0.8403078  0.6782159
##   20  0.8417451  0.6812128
##   21  0.8400326  0.6775841
##   22  0.8417193  0.6809592
##   23  0.8393922  0.6762512
##   24  0.8396014  0.6765003
##   25  0.8398385  0.6769226
##   26  0.8350342  0.6670152
##   27  0.8362295  0.6694827
##   28  0.8373787  0.6715588
##    29  0.8384899  0.6737666
##    30  0.8358463  0.6684968
##    31  0.8358319  0.6683878
##    32  0.8352437  0.6672649
##    33  0.8347330  0.6659835
##    34  0.8326132  0.6617427
##    35  0.8340129  0.6643693
##    36  0.8350741  0.6664307
##    37  0.8363510  0.6690000
##    38  0.8368479  0.6699714
##    39  0.8380439  0.6721155
##    40  0.8388012  0.6736092
##    41  0.8380315  0.6720585
##    42  0.8401608  0.6767402
##    43  0.8413187  0.6789911
##    44  0.8384483  0.6728935
##    45  0.8383280  0.6725994
##    46  0.8413606  0.6786332
##    47  0.8371070  0.6697932
##    48  0.8376082  0.6709782
##    49  0.8375995  0.6708813
##    50  0.8385083  0.6727770
##    51  0.8387693  0.6732070
##    52  0.8375288  0.6705582
##    53  0.8356418  0.6668208
##    54  0.8368846  0.6692826
##    55  0.8390158  0.6735646
##    56  0.8373673  0.6701037
##    57  0.8383501  0.6720152
##    58  0.8347733  0.6648376
##    59  0.8371814  0.6696607
##    60  0.8367265  0.6687031
##    61  0.8364386  0.6679607
##    62  0.8359800  0.6670001
##    63  0.8341224  0.6633289
##    64  0.8339447  0.6628354
##    65  0.8332432  0.6613519
##    66  0.8330113  0.6608714
##    67  0.8324522  0.6596654
##    68  0.8334524  0.6617053
##    69  0.8315692  0.6578082
##    70  0.8332840  0.6612525
##    71  0.8323502  0.6591855
##    72  0.8302788  0.6550887
##    73  0.8317200  0.6578914
##    74  0.8330771  0.6607760
##    75  0.8314663  0.6574825
##    76  0.8312344  0.6569989
##    77  0.8302102  0.6549307
##    78  0.8298235  0.6541197
##    79  0.8311935  0.6567689
##    80  0.8314223  0.6572917
##    81  0.8313935  0.6570980
##    82  0.8318381  0.6579752
##    83  0.8316127  0.6576069
##    84  0.8299450  0.6541362
##    85  0.8299032  0.6540420
##    86  0.8290320  0.6522628
##    87  0.8281186  0.6503515
##    88  0.8292795  0.6527257
##    89  0.8283805  0.6508557
##    90  0.8297903  0.6537130
##    91  0.8295963  0.6532952
##    92  0.8300782  0.6542852
##    93  0.8293363  0.6527138
##    94  0.8284848  0.6509885
##    95  0.8289016  0.6517939
##    96  0.8279054  0.6498714
##    97  0.8269821  0.6478887
##    98  0.8274709  0.6489869
##    99  0.8277306  0.6495025
##   100  0.8267707  0.6475188
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 20.
```

```
cat('\n\n','***** VALIDATION DATA METRICS *******','\n')
```

```
##
##
##  ***** VALIDATION DATA METRICS *******
```

```
confusionMatrix(table(val_pred, as.matrix(valid_set['R1'])), positive='1')
```

```
## Confusion Matrix and Statistics
##
##
## val_pred  0  1
##        0 45  8
##        1  8 36
##
##                Accuracy : 0.8351
##                  95% CI : (0.746, 0.9027)
##     No Information Rate : 0.5464
##     P-Value [Acc > NIR] : 1.81e-09
##
##                   Kappa : 0.6672
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8182
##             Specificity : 0.8491
##          Pos Pred Value : 0.8182
##          Neg Pred Value : 0.8491
##              Prevalence : 0.4536
##          Detection Rate : 0.3711
##    Detection Prevalence : 0.4536
##       Balanced Accuracy : 0.8336
##
##        'Positive' Class : 1
```

```
##
```

```
cat('\n\n','***** TEST DATA METRICS *****','\n')
```

```
##
##
##  ***** TEST DATA METRICS *****
```

```
confusionMatrix(table(test_pred, as.matrix(test_set['R1'])), positive='1')
```

```
## Confusion Matrix and Statistics
##
##
## test_pred  0  1
##         0 43  7
##         1 11 37
##
##                Accuracy : 0.8163
##                  95% CI : (0.7253, 0.8874)
##     No Information Rate : 0.551
##     P-Value [Acc > NIR] : 3.029e-08
##
##                   Kappa : 0.6319
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.8409
##             Specificity : 0.7963
##          Pos Pred Value : 0.7708
##          Neg Pred Value : 0.8600
##              Prevalence : 0.4490
##          Detection Rate : 0.3776
##    Detection Prevalence : 0.4898
##       Balanced Accuracy : 0.8186
##
##        'Positive' Class : 1
##
```

***Analysis** Interestingly enough, when performing a Train/Validation/Test split, the accuracy is comparable with cross validation. However the optimal k nearest neighbor has changed greatly from 5 to 20. The model is pretty good at picking the negative class ("0"), but is not as optimal with the positive class ("1") when analyzing the test set metrics. it picked 37 / 48 positive classes for a positive prediciton value of 77%. Conversely we can see the negative prediciton accuarcy is 86%.

Question 4.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a clustering model would be appropriate. List some (up to 5) predictors that you might use.

In business analytics, there are cumbersome amounts of data, both structured and unstructured. Because of this, some online retailers will utilize clustering techniques to refine what specific merchandise they show you as suggestions on their websites. For example they'll keep track of what your past purchases were, what you currently have in your cart, your clickstream through their sites looking for specific brand patterns, how long you stay on an item's page before either adding it to the cart or moving on to another item, as well as how long you hover over a specific item before clicking the link (or not clicking the link). From this, they can group you into a cluster of other clients and compare you to what those other clients have purchased in the *past. If you haven't also already purchased those items, they will suggest those items to you since you seem to fit the cluster of customers who typically buy those items.*

Question 4.2

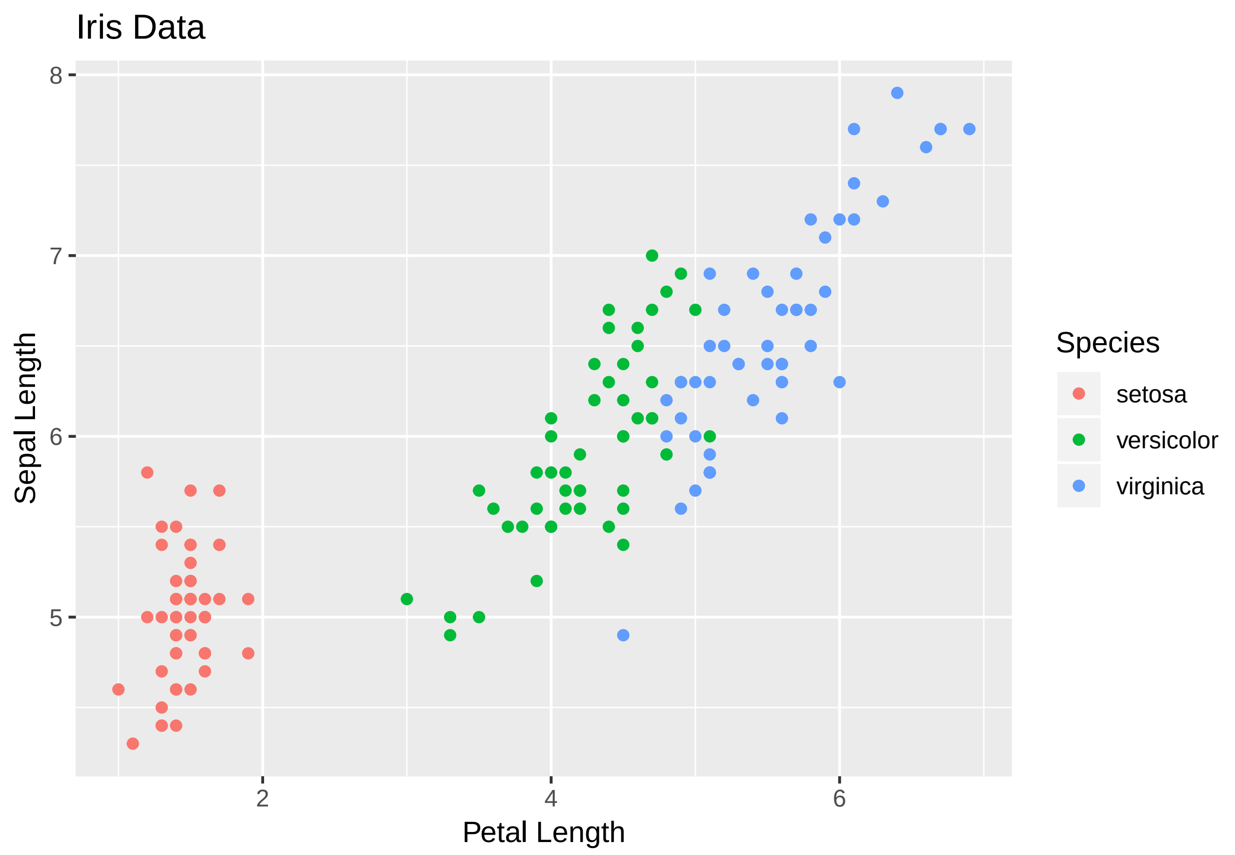
The iris data set iris.txt contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers and the response is the type of flower. The data is available from the R library datasets and can be accessed with iris once the library is loaded. It is also available at the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris). The response values are only given to see how well a specific method performed and should not be used to build the model. Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

Explore the Data Visually

```r
data(iris)
head(iris)
```

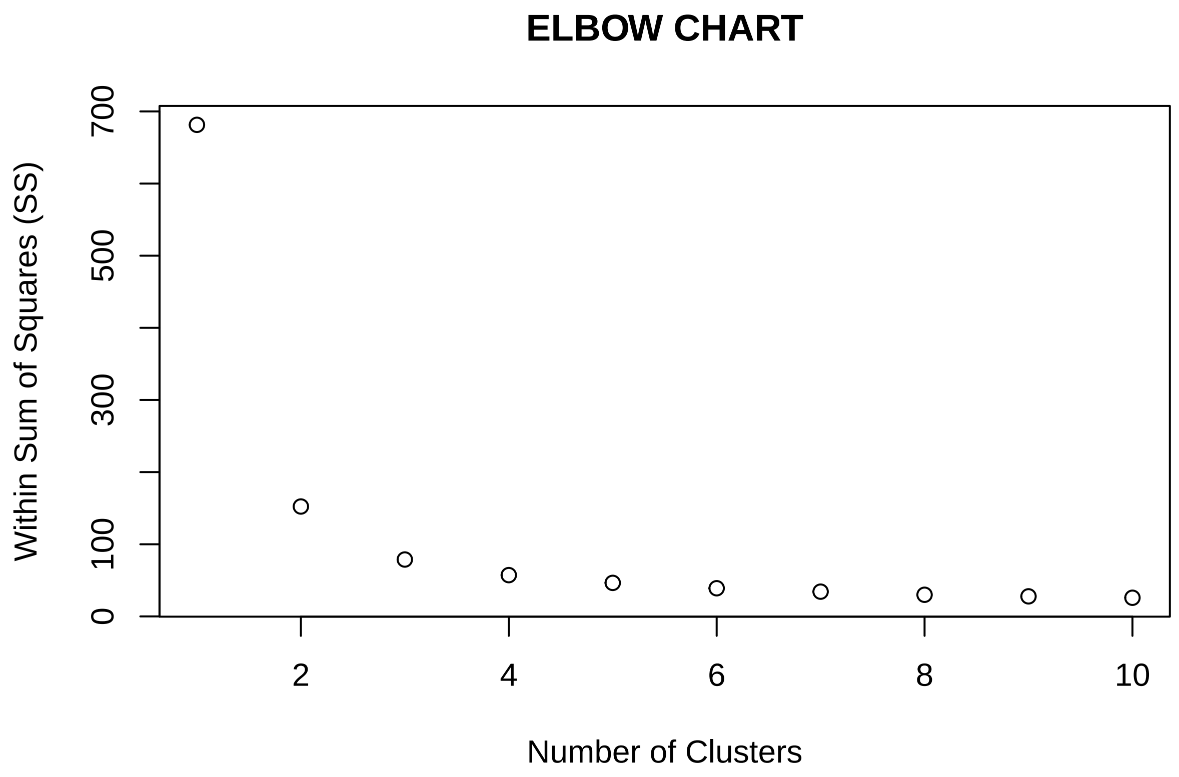
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
#ggplot the data
plot <- ggplot(iris, aes(x=Petal.Length, y=Sepal.Length, colour = Species)) + geom_point() + ggtitle('I
plot + labs(x = "Petal Length", y = "Sepal Length")
```



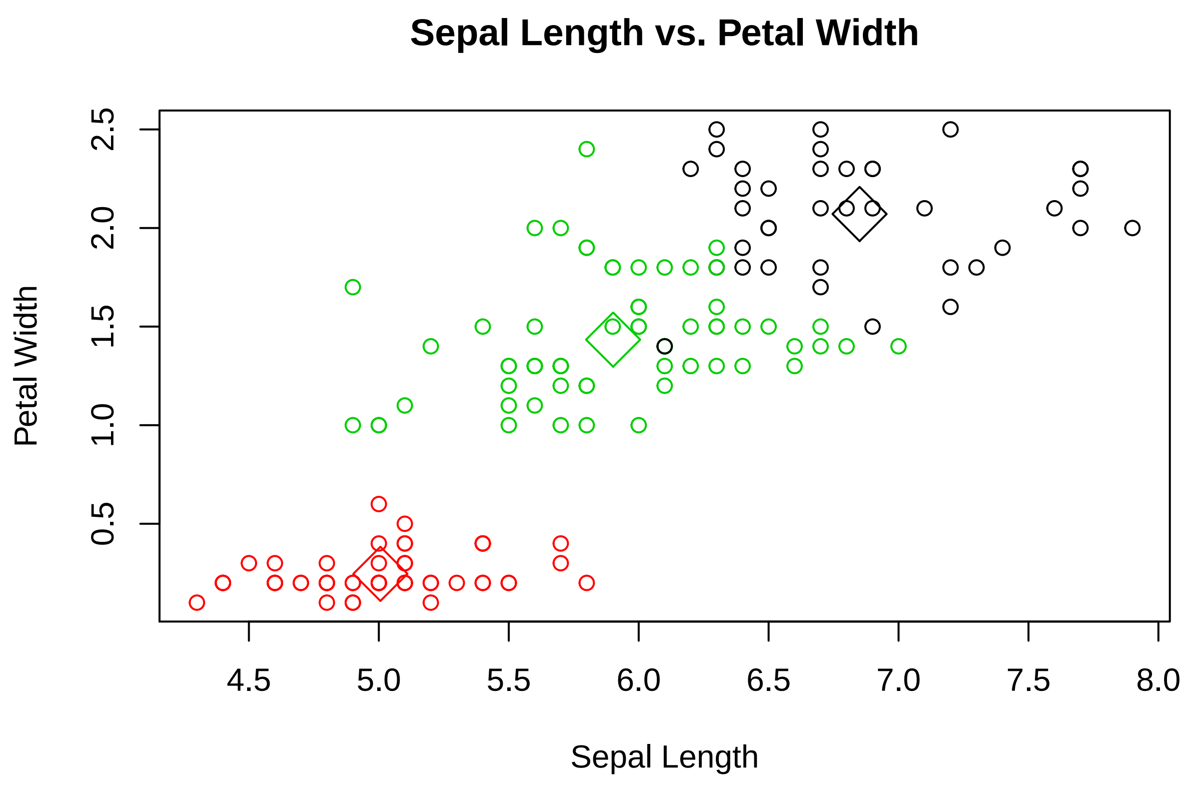
***Analysis** Investigating the data visually, we can see a clear distinction amongst the 3 different iris types. The Versicolor and Virginica however appear to have an overlapping zone which may cause somewhat of a hit in model accuracy once it is trained. Before we train our model, let's see how many clusters is optimal. We won't scale our data since it is all within the same relative range as far as scale is concerned for each feature.

```
data(iris)

#extract the independent variables
X <- iris[,1:4]

#intialize and empty vector to store all the calculated wss values
wss <- rep(0,10)
#try kmeans models using 1 to 10 clusters
for(i in 1:10){

model <- kmeans(X, i, nstart = 50)
wss[i] <- model$tot.withinss

}
#create elbow chart
plot(1:10, wss, main='ELBOW CHART', xlab = 'Number of Clusters', ylab = 'Within Sum of Squares (SS)')
```

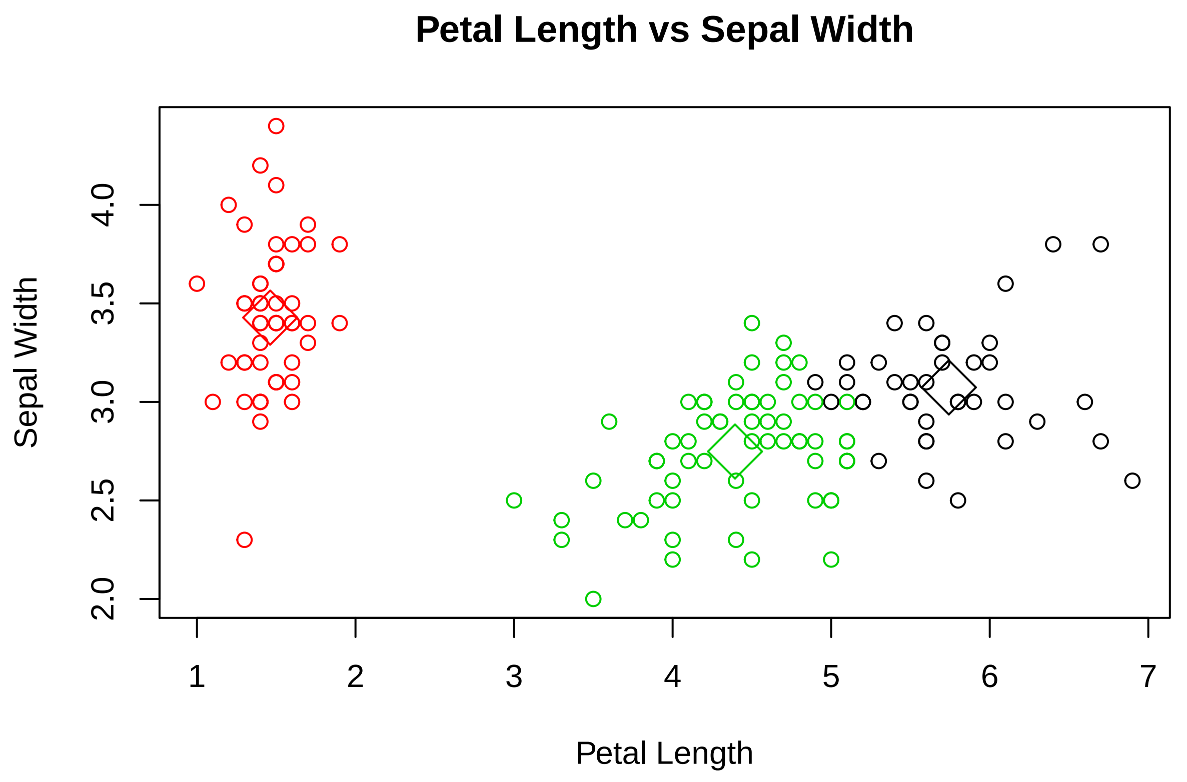


***Analysis** From the elbow chart above it appears that either 3 or 4 clusters is optmial since they form the "elbow" of the graph. We are going to cheat and train the model using only 3 clusters since we know there are only 3 iris types.

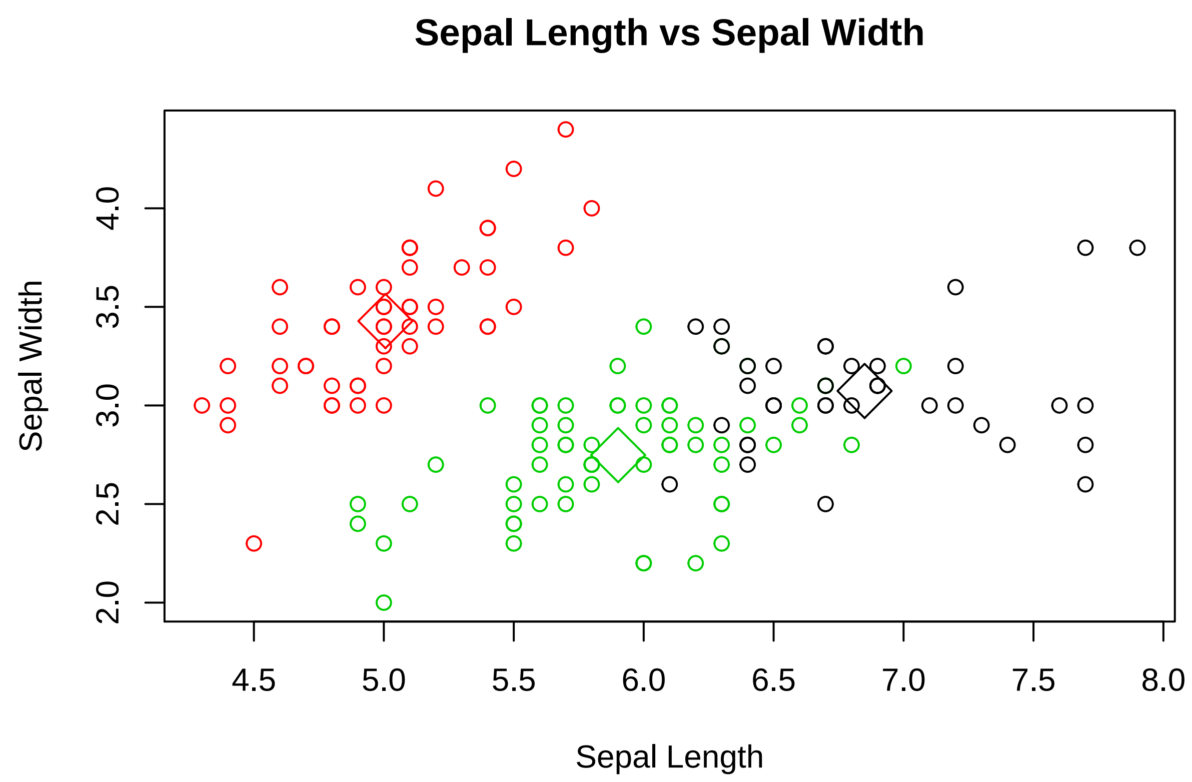
```
data(iris)

#separate the independent variables
X <- iris[,1:4]

#train the model
model <- kmeans(X, 3, nstart = 50)

#plot the data with the corresponding cluster centroids from the trained model
plot(X[c("Sepal.Length", "Petal.Width")], col=model$cluster, main="Sepal Length vs. Petal Width", xlab =
points(model$centers[,c("Sepal.Length", "Petal.Width")], col=1:3, pch=23, cex=3)
```



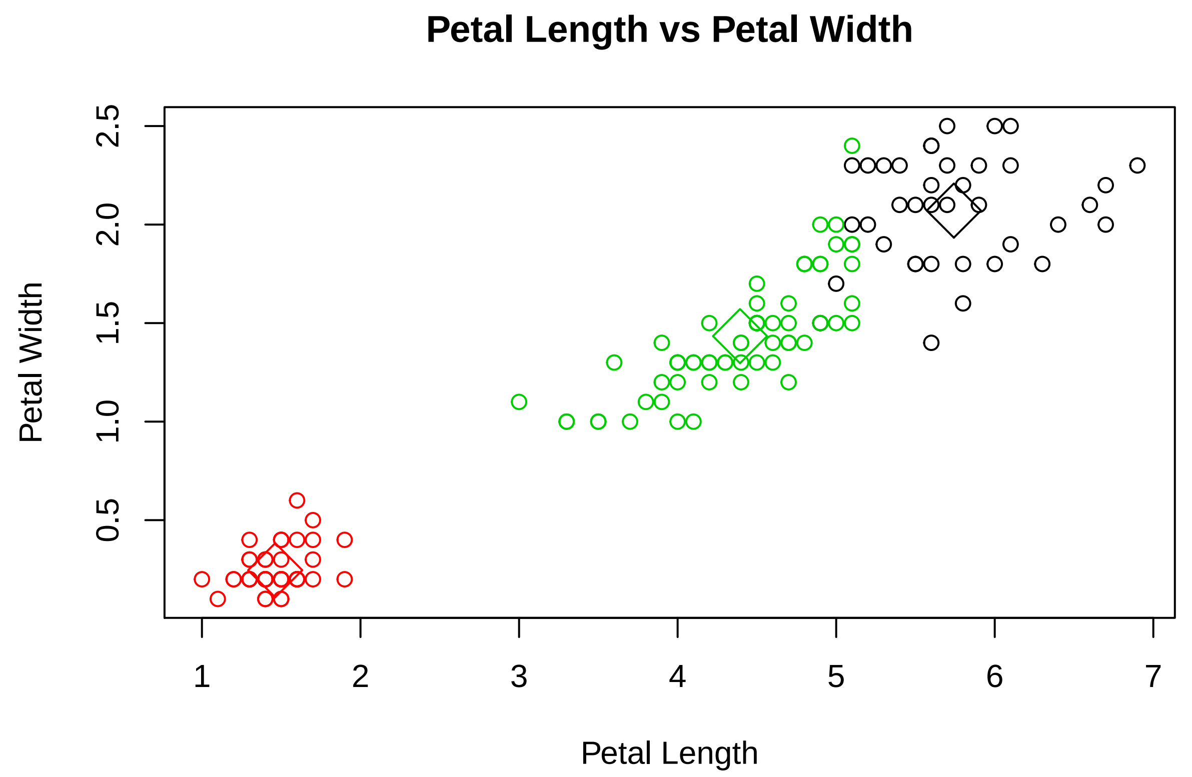
```r
plot(X[c("Petal.Length", "Sepal.Width")], col=model$cluster, main="Petal Length vs Sepal Width", xlab =
points(model$centers[,c("Petal.Length", "Sepal.Width")], col=1:3, pch=23, cex=3)
```



```r
plot(X[c("Sepal.Length", "Sepal.Width")], col=model$cluster, main="Sepal Length vs Sepal Width", xlab =
points(model$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=23, cex=3)
```



```
plot(X[c("Petal.Length", "Petal.Width")], main = "Petal Length vs Petal Width", col=model$cluster, xlab
points(model$centers[,c("Petal.Length", "Petal.Width")], col=1:3, pch=23, cex=3)
```



***Analysis** We can see from all the charts above the the cluster centroids do coincide with their proper classification. This reaffirms our analysis from the elbow chart that 3 clusters was indeed the optimium model.

From this we can conclude that the model will predict corresponding flowers accurately, as the centroids are well embedded in the middle of the correct classifications