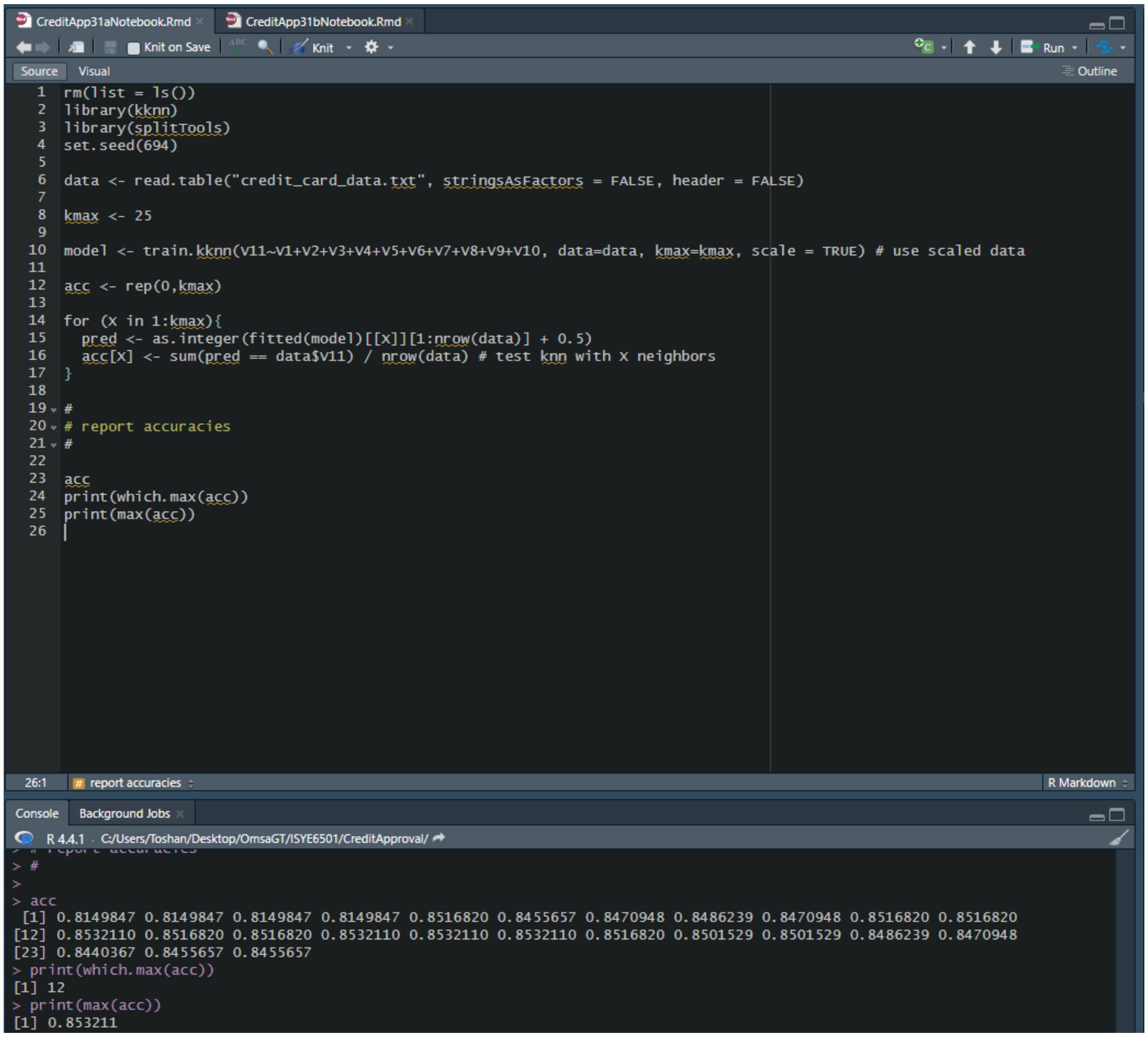
Question 3.1

a) Cross validation for the k-nearest-neighbors model was used to determine the accuracy in classifying the credit approval dataset. Although not required, I utilized both methods of cross validation available to use in the kknn package. The first method, with accuracies per k-value generated in Figure 1, was the leave-one-out (LOO) cross validation procedure done with train.kknn. The second method, with data, visualization, and results for kcv=15, was performed using k-fold cross validation. The rationale behind picking either one is based on the size of the dataset for which the model is being tested. By this, I mean that LOO approaches the cross validation problem by swapping a single portion of the data set out as a training/testing collection, and then constantly rotating through to make sure all portions of the data have been used as a training or testing set. Whereas k-fold takes the method of creating k subsets (A.K.A k-folds) to partition the data into a training/testing collection. Which means that as the dataset gets larger, using the k-fold method will typically result in a sufficiently accurate classification based on the length of time it would take to train the model. For smaller datasets (such as credit approval), LOO should generate more accurate results since it will use more training samples in each iteration. However, for my data analysis this was shown to be untrue. A possible reason for this could be due to random effects, as the max accuracy between the two methods only differed by about $7.0*10^{-3}$.



```r
rm(list = ls())
library(kknn)
library(splitTools)
set.seed(694)

data <- read.table("credit_card_data.txt", stringsAsFactors = FALSE, header = FALSE)

kmax <- 25

model <- train.kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10, data=data, kmax=kmax, scale = TRUE) # use scaled data

acc <- rep(0,kmax)

for (X in 1:kmax){
  pred <- as.integer(fitted(model)[[X]][1:nrow(data)] + 0.5)
  acc[X] <- sum(pred == data$V11) / nrow(data) # test knn with X neighbors
}

#
# report accuracies
#

acc
print(which.max(acc))
print(max(acc))
```

```
> #
>
> acc
 [1] 0.8149847 0.8149847 0.8149847 0.8149847 0.8516820 0.8455657 0.8470948 0.8486239 0.8470948 0.8516820 0.8516820
[12] 0.8532110 0.8516820 0.8516820 0.8532110 0.8532110 0.8532110 0.8516820 0.8501529 0.8501529 0.8486239 0.8470948
[23] 0.8440367 0.8455657 0.8455657
> print(which.max(acc))
[1] 12
> print(max(acc))
[1] 0.853211
```

Figure 1. This figure denotes leave-one-out cross validation method of k-nearest neighbor (most computationally efficient) on the credit approval dataset.

K-cv	Accuracy	K-value
2	**0.8593272**	**19**
3	**0.8593272**	**16**
4	**0.8608563**	**14**
5	**0.8608563**	**15**
10	**0.8608563**	**13**
15	**0.8547401**	**10**
20	**0.851682**	**10**
25	**0.8593272**	**19**
30	**0.8577982**	**11**

Figure 2. Data table of varying k-folds in the k-fold cross validation model per largest accuracy.

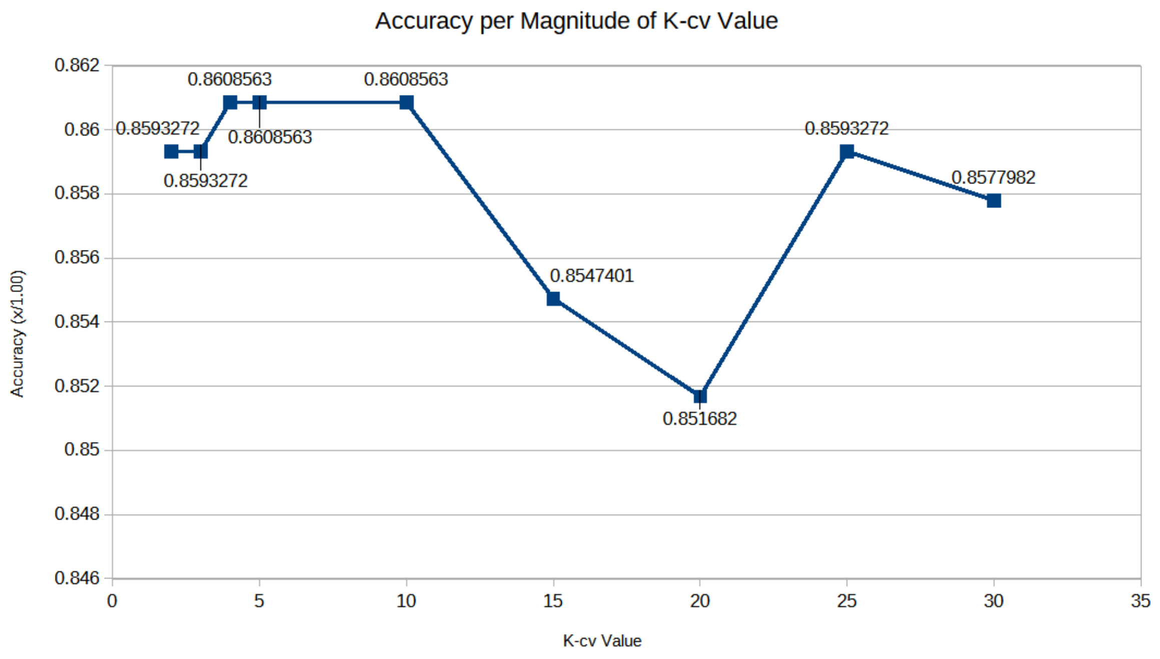
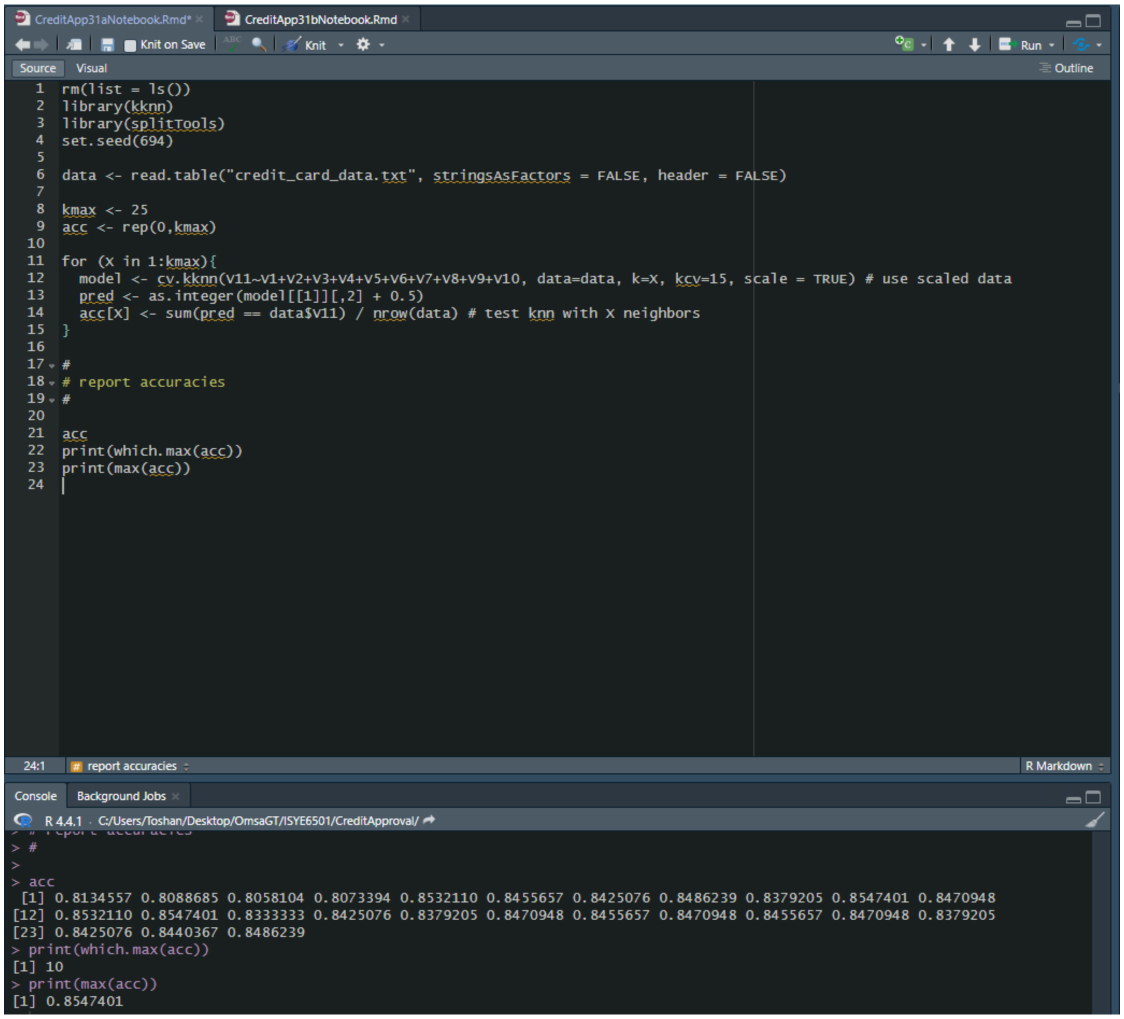


Figure 3. Visualization of Figure 2, demonstrating the relationship specifically between the accuracy and the k-cv value. This graph demonstrates that a higher fold value does not correlate to a higher accuracy.

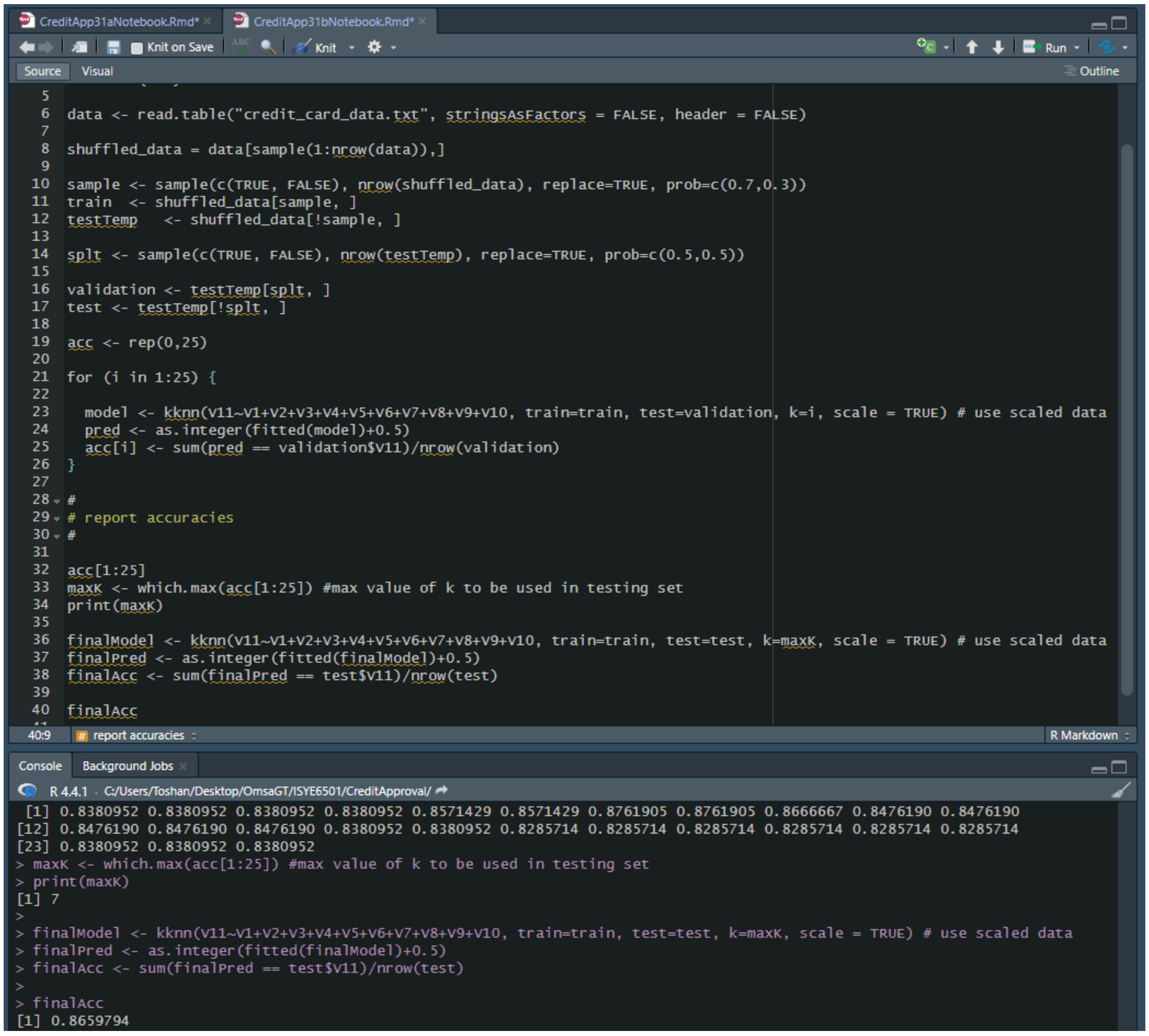


```r
rm(list = ls())
library(kknn)
library(splitTools)
set.seed(694)

data <- read.table("credit_card_data.txt", stringsAsFactors = FALSE, header = FALSE)

kmax <- 25
acc <- rep(0,kmax)

for (X in 1:kmax){
  model <- cv.kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10, data=data, k=X, kcv=15, scale = TRUE) # use scaled data
  pred <- as.integer(model[[1]][,2] + 0.5)
  acc[X] <- sum(pred == data$V11) / nrow(data) # test knn with X neighbors
}

#
# report accuracies
#

acc
print(which.max(acc))
print(max(acc))

```

```
> #
>
> acc
 [1] 0.8134557 0.8088685 0.8058104 0.8073394 0.8532110 0.8455657 0.8425076 0.8486239 0.8379205 0.8547401 0.8470948
[12] 0.8532110 0.8547401 0.8333333 0.8425076 0.8379205 0.8470948 0.8455657 0.8470948 0.8455657 0.8470948 0.8379205
[23] 0.8425076 0.8440367 0.8486239
> print(which.max(acc))
[1] 10
> print(max(acc))
[1] 0.8547401
```

Figure 4. This figure denotes the k-fold cross validation method of k-nearest neighbors on the credit approval dataset.

b) In this approach to classifying the credit approval dataset, the k-nearest-neighbors model was used in tandem with a training, testing, and validation set. The split chosen for this was 0.70, 0.15, and 0.15 for the training, testing, and validation, respectively. The reason for this choice was so that the training set occupied a clear majority over the testing and validation split, thus ensuring that the model could see as many cases as possible while retaining a sizable amount to test itself. After running the training and validation sets on the model for k-values from 1 to 25, the greatest accuracy was produced at k=7. The final accuracy generated on the test set was 0.8659794. The highest accuracy k-value being an odd number is supported by the notion that it includes an automatic tie break between selecting which class the data point would belong to.


CreditApp31aNotebook.Rmd*
CreditApp31bNotebook.Rmd*
Knit on Save
Knit
Run
Source
Visual
Outline
40:9
report accuracies
R Markdown
Console
Background Jobs
R 4.4.1 · C:/Users/Toshan/Desktop/OmsaGT/ISYE6501/CreditApproval/

```r

data <- read.table("credit_card_data.txt", stringsAsFactors = FALSE, header = FALSE)

shuffled_data = data[sample(1:nrow(data)),]

sample <- sample(c(TRUE, FALSE), nrow(shuffled_data), replace=TRUE, prob=c(0.7,0.3))
train  <- shuffled_data[sample, ]
testTemp   <- shuffled_data[!sample, ]

splt <- sample(c(TRUE, FALSE), nrow(testTemp), replace=TRUE, prob=c(0.5,0.5))

validation <- testTemp[splt, ]
test <- testTemp[!splt, ]

acc <- rep(0,25)

for (i in 1:25) {

  model <- kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10, train=train, test=validation, k=i, scale = TRUE) # use scaled data
  pred <- as.integer(fitted(model)+0.5)
  acc[i] <- sum(pred == validation$V11)/nrow(validation)
}

#
# report accuracies
#

acc[1:25]
maxK <- which.max(acc[1:25]) #max value of k to be used in testing set
print(maxK)

finalModel <- kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10, train=train, test=test, k=maxK, scale = TRUE) # use scaled data
finalPred <- as.integer(fitted(finalModel)+0.5)
finalAcc <- sum(finalPred == test$V11)/nrow(test)

finalAcc
```

```
 [1] 0.8380952 0.8380952 0.8380952 0.8380952 0.8571429 0.8571429 0.8761905 0.8761905 0.8666667 0.8476190 0.8476190
[12] 0.8476190 0.8476190 0.8476190 0.8380952 0.8380952 0.8285714 0.8285714 0.8285714 0.8285714 0.8285714 0.8285714
[23] 0.8380952 0.8380952 0.8380952
> maxK <- which.max(acc[1:25]) #max value of k to be used in testing set
> print(maxK)
[1] 7
>
> finalModel <- kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10, train=train, test=test, k=maxK, scale = TRUE) # use scaled data
> finalPred <- as.integer(fitted(finalModel)+0.5)
> finalAcc <- sum(finalPred == test$V11)/nrow(test)
>
> finalAcc
[1] 0.8659794
```

Figure 5. This figure denotes the accuracies of the 25 k’s used to determine the most optimal for the test set.

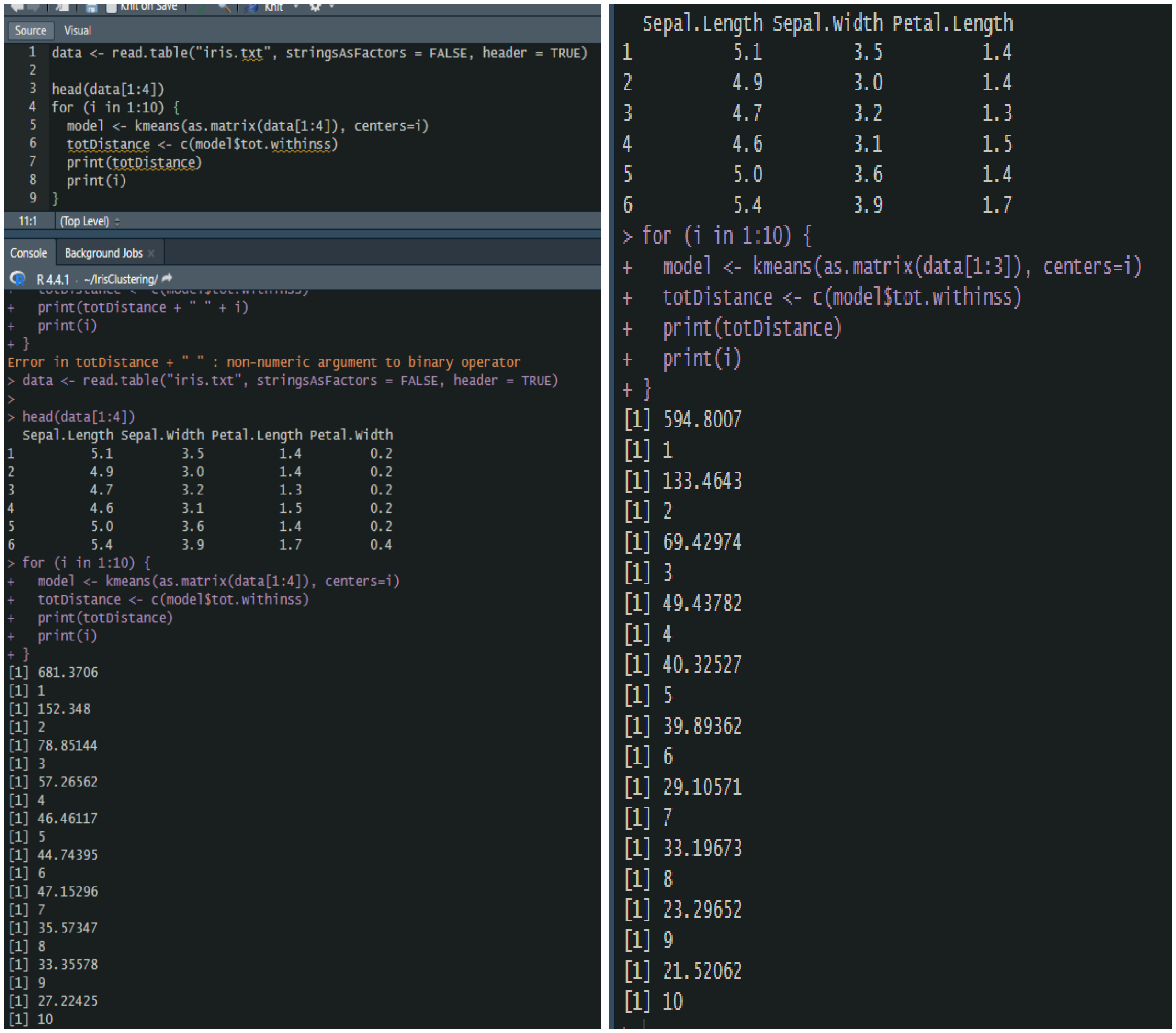
Question 4.1

As a prospective data scientist with the goal of working in the field of bioinformatics, working on projects that are encompassed by the field gives me valuable experience so that I may succeed in my endeavor. There are many cases in this field for which selecting a clustering model would result in a beneficial analysis, one such case is the identification of different unannotated sequences. In order to determine where each sequence belongs, there are many predictors that can be identified to ensure the success of the clustering algorithm. They are:

1. Length of Sequence
2. Order of Sequence
3. Base Composition of Sequence
4. Hamming Distance of Sequence
5. Location of Promoter Sequence

Question 4.2

This question required the use of k-means to cluster the data points of the iris dataset so that it will accurately predict the flower type. By testing many combinations of predictors and k-values, the most accurate prediction can be made by the clustering model. In Figures 6&7, the dataset is reduced in each image by removing a single column (as seen in the console output) and iterating over the model with k-values from 1 to 10. The numbers generated above the k-values correspond to that k-value and the total distance between the total sum of the squares distances in all the clusters of the points from the assigned centroid. This result is significant due to its use in determining the optimal k-value. As Professor Sokol mentioned in the lecture video, it does this via the kink in the elbow curve that would form as a result of plotting the numbers per k-value. Each figure is labeled with the quantitatively determined k-value, however this result may or may not agree with the qualitative k-value of 3. The k-value of 3 was determined due the dataset only containing 3 types of flowers. As such, having clusters greater than or less than three would automatically reject any notion of the clusters being correct with respect to the given data. Based on these factors, it is seen that the hyperparameters that produced the greatest accuracy while retaining the qualitatively and quantitatively k-value of 3 were utilizing only the sepal length, width, and petal length. The accuracy of this model was 69.42974 which was more accurate than the four column 78.85144 accuracy. This is due to the fact that the centroid of the cluster was, on average, closer in the three column model than the fourth. Additionally, the petal width seemed to have misled the grouping of the clusters as proven by the four column model having greater spread in its clusters.

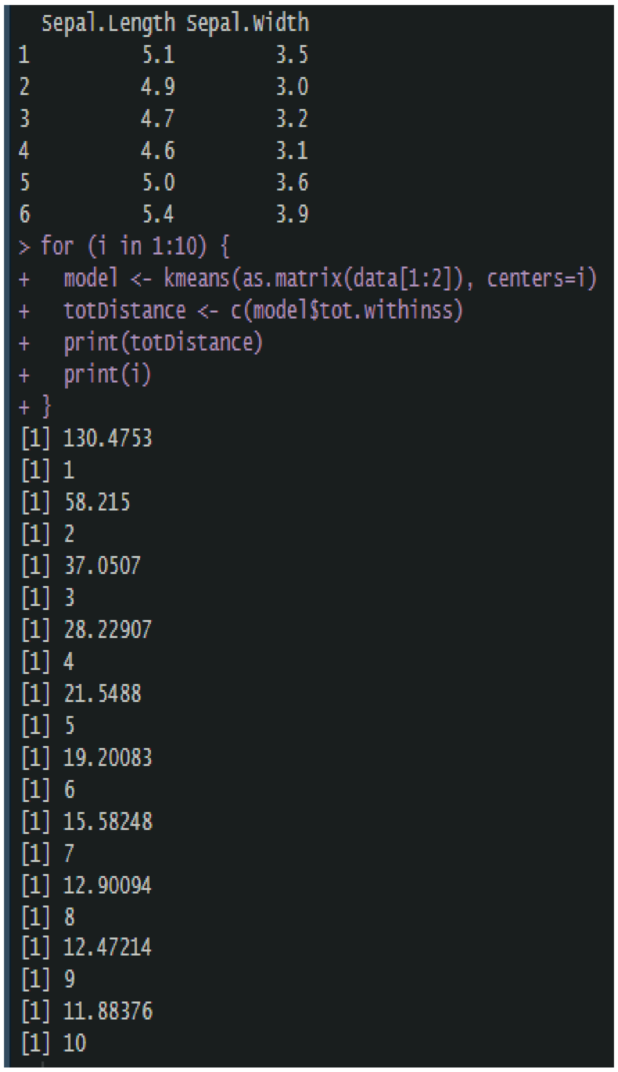


```
data <- read.table("iris.txt", stringsAsFactors = FALSE, header = TRUE)

head(data[1:4])
for (i in 1:10) {
  model <- kmeans(as.matrix(data[1:4]), centers=i)
  totDistance <- c(model$tot.withinss)
  print(totDistance)
  print(i)
}
```

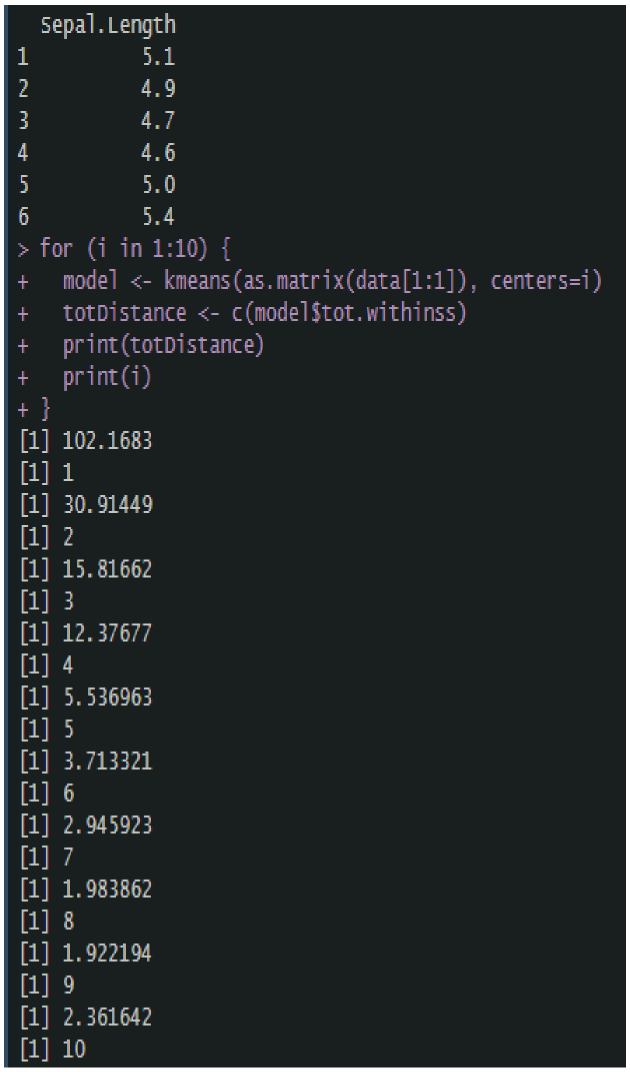
```
+   print(totDistance + " " + i)
+   print(i)
+ }
Error in totDistance + " " : non-numeric argument to binary operator
> data <- read.table("iris.txt", stringsAsFactors = FALSE, header = TRUE)
>
> head(data[1:4])
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
4          4.6         3.1          1.5         0.2
5          5.0         3.6          1.4         0.2
6          5.4         3.9          1.7         0.4
> for (i in 1:10) {
+   model <- kmeans(as.matrix(data[1:4]), centers=i)
+   totDistance <- c(model$tot.withinss)
+   print(totDistance)
+   print(i)
+ }
[1] 681.3706
[1] 1
[1] 152.348
[1] 2
[1] 78.85144
[1] 3
[1] 57.26562
[1] 4
[1] 46.46117
[1] 5
[1] 44.74395
[1] 6
[1] 47.15296
[1] 7
[1] 35.57347
[1] 8
[1] 33.35578
[1] 9
[1] 27.22425
[1] 10
```

```
  Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5          1.4
2          4.9         3.0          1.4
3          4.7         3.2          1.3
4          4.6         3.1          1.5
5          5.0         3.6          1.4
6          5.4         3.9          1.7
> for (i in 1:10) {
+   model <- kmeans(as.matrix(data[1:3]), centers=i)
+   totDistance <- c(model$tot.withinss)
+   print(totDistance)
+   print(i)
+ }
[1] 594.8007
[1] 1
[1] 133.4643
[1] 2
[1] 69.42974
[1] 3
[1] 49.43782
[1] 4
[1] 40.32527
[1] 5
[1] 39.89362
[1] 6
[1] 29.10571
[1] 7
[1] 33.19673
[1] 8
[1] 23.29652
[1] 9
[1] 21.52062
[1] 10
```

Figure 6. This figure denotes the generated values and code used to run the k-means model. For the left image, using all four columns of the dataset, the kink appears to be at k=3. The same is true for the image with only 3/4 columns on the right.



```
  Sepal.Length Sepal.Width
1          5.1         3.5
2          4.9         3.0
3          4.7         3.2
4          4.6         3.1
5          5.0         3.6
6          5.4         3.9
> for (i in 1:10) {
+   model <- kmeans(as.matrix(data[1:2]), centers=i)
+   totDistance <- c(model$tot.withinss)
+   print(totDistance)
+   print(i)
+ }
[1] 130.4753
[1] 1
[1] 58.215
[1] 2
[1] 37.0507
[1] 3
[1] 28.22907
[1] 4
[1] 21.5488
[1] 5
[1] 19.20083
[1] 6
[1] 15.58248
[1] 7
[1] 12.90094
[1] 8
[1] 12.47214
[1] 9
[1] 11.88376
[1] 10
```



```
  Sepal.Length
1          5.1
2          4.9
3          4.7
4          4.6
5          5.0
6          5.4
> for (i in 1:10) {
+   model <- kmeans(as.matrix(data[1:1]), centers=i)
+   totDistance <- c(model$tot.withinss)
+   print(totDistance)
+   print(i)
+ }
[1] 102.1683
[1] 1
[1] 30.91449
[1] 2
[1] 15.81662
[1] 3
[1] 12.37677
[1] 4
[1] 5.536963
[1] 5
[1] 3.713321
[1] 6
[1] 2.945923
[1] 7
[1] 1.983862
[1] 8
[1] 1.922194
[1] 9
[1] 2.361642
[1] 10
```

Figure 7. This figure denotes the k-value to be equal to 2 for the right and left image.