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)

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

Part a

Using cross validation on the k nearest neighbors model, I found that a k-value of 57 produced the lowest mean squared error. As noted in class, this is based on the common value of cross validations being 10. Even though I ended up using 10 cross validations as my final number, I played around with different values for it to see if the answer of k = 57 would change at all. It turned out that it was a constant value of k = 57 regardless of how many cross validations I chose. In terms of k-values, I tested from k = 1 to k = 100 and skipped by 2 (so tested only odd numbers). I did this because the knn model works best with an odd number of k-values since there cannot be any ties between the number of different points in a group. I tested these k-values against what their minimal mean squared error was. In other words, for each k-value, I looked at what was the chance that the point was misclassified instead of what was the chance that the point was correctly classified. The minimal mean squared error occurred at k = 57 where it was 0.1073804. Below is a table showing each k-value tested and their corresponding minimal mean squared error around k = 57. The minimal mean squared error was largest at k = 1 (MSE = .1850153) and shrunk as it go to k = 57 and after k = 57 it started to grow larger again. I found that if I did not set the seed in the R code, then a different best k-value would come out every time, so in this specific cross validation set in this seed, the best k-value turned out to be 57.

K = 53	MSE = 0.1074282
K = 55	MSE = 0.1073986
K = 57	MSE = 0.1073804
K = 59	MSE = 0.1073850
K = 61	MSE = 0.1074153

R Code and Output

```
set.seed(123)

pacman::p_load(kernlab, dplyr, kknn, caTools)

data <- read.table("credit_card_data-headers.txt", header = TRUE)

k_list <- c(seq(from = 1, to = 100, by = 2))

knn_model <- train.kknn(R1~., data, ks=k_list, kcv = 10, kernel = "optimal", distance = 2, scale = TRUE)

print(knn_model)

print(knn_model$MEAN.SQU)
```

Type of response variable: continuous

minimal mean absolute error: 0.1850153

Minimal mean squared error: 0.1073804

Best kernel: optimal

Best k: 57

```
> print(knn_model$MEAN.SQU)
    optimal
1  0.1850153
3  0.1441704
5  0.1274106
7  0.1193859
9  0.1159121
11 0.1139085
13 0.1126801
```

15 0.1117159

17 0.1109290

19 0.1102460

21 0.1096646

23 0.1090944

25 0.1086863

27 0.1084168

29 0.1082095

31 0.1080277

33 0.1078638

35 0.1077567

37 0.1076855

39 0.1076509

41 0.1076072

43 0.1075653

45 0.1075203

47 0.1074917

49 0.1074710

51 0.1074555

53 0.1074282

55 0.1073986

57 0.1073804

59 0.1073850

61 0.1074153

63 0.1074609

65 0.1075103

67 0.1075566

69 0.1075962

71 0.1076398

73 0.1076807

75 0.1077263

77 0.1077767

79 0.1078371

81 0.1079073

83 0.1079835

85 0.1080658

87 0.1081492

89 0.1082304

91 0.1083153

93 0.1084074

95 0.1085026

97 0.1085990

99 0.1086954

Part B

By randomly splitting the data into training, validation, and testing data, I found that a k-value of k = 7 resulted in the best knn prediction. To start, I split the data first into the training data. I decided to use 60% of the data to create the training data. That means that remaining 40% of the data was used on the validation and testing data (20% each). I picked 60% 20% 20% because I personally liked giving more into the validation and testing data rather than something like 15% or smaller. The k-values I tested were from 1 to 100 and I skipped by two so that there were only odd values for k tested. I ran my knn model first using the training data and the validation data sets and saved each k-values corresponding accuracy value. Using these two, I found that a k-value of 7 provided the best knn prediction with the accuracy value being 0.857143. Using this information, I then tested the

knn model again using the k-value of 7, the training data, and the testing data. After running this model, I found that the accuracy value for k = 7 decreased over 3% for the new value for it was 0.820513 after using the test data. This means that the observed performance, or the accuracy after the validation data, had about 3% random effect attached to it which made the value higher. I was fine with using the randomness method on splitting this data because the data is not based off certain points in time, so certain days/months/years could not be in one set more than the other to skew the accuracy number.

R Code and Output

```
set.seed(123)

pacman::p_load(kernlab, kknn, caret )

data <- read.table("credit_card_data-headers.txt", header = TRUE)

data_part_1 <- createDataPartition(data$R1, p=.60, list = FALSE)

str(data_part_1)

train_data <- data[data_part_1,]

dim(train_data)

remaining_data <- data[-data_part_1,]

data_part_2 <- createDataPartition(y=remaining_data$R1, p=.40, list = FALSE)

validation_data <- remaining_data[data_part_2,]

dim(validation_data)

test_data <- remaining_data[-data_part_2,]

dim(test_data)
```

```
k_list <- c(seq(from = 1, to = 100, by = 2))

knn_prediction <- c()


for (k in 1:length(k_list)){

  knn_model <- kknn(R1~., train_data, validation_data, k=k_list[[k]], kernel = "optimal",
distance = 2, scale = TRUE)

    knn_predict <- as.integer(fitted(knn_model)+0.5)

  knn_prediction[k] <- sum(knn_predict == validation_data$R1)/nrow(validation_data)

}


do.call(rbind, Map(data.frame, K=k_list, knn_prediction = knn_prediction))


sprintf("k = %f provides the best knn prediction of %f", k_list[which(knn_prediction ==
max(knn_prediction))[1]], max(knn_prediction))


knn_model_best <- kknn(R1~., train_data, test_data, k=k_list[which(knn_prediction ==
max(knn_prediction))[1]], kernel = "optimal", distance = 2, scale = TRUE)

knn_predict <- as.integer(fitted(knn_model_best)+0.5)

knn_predict_best <- sum(knn_predict == test_data$R1)/nrow(test_data)

sprintf("k = %f provides the best knn prediction of %f", k_list[which(knn_prediction ==
max(knn_prediction))[1]], max(knn_predict_best))


> do.call(rbind, Map(data.frame, K=k_list, knn_prediction = knn_prediction))

   K knn_prediction

1  1     0.8095238

2  3     0.8095238

3  5     0.8380952
```

4	7	0.8571429
5	9	0.8571429
6	11	0.8571429
7	13	0.8476190
8	15	0.8476190
9	17	0.8476190
10	19	0.8380952
11	21	0.8380952
12	23	0.8380952
13	25	0.8380952
14	27	0.8380952
15	29	0.8380952
16	31	0.8380952
17	33	0.8285714
18	35	0.8285714
19	37	0.8190476
20	39	0.8190476
21	41	0.8190476
22	43	0.8190476
23	45	0.8190476
24	47	0.8190476
25	49	0.8190476
26	51	0.8285714
27	53	0.8285714
28	55	0.8285714
29	57	0.8285714

30 59 0.8380952

31 61 0.8380952

32 63 0.8380952

33 65 0.8380952

34 67 0.8380952

35 69 0.8380952

36 71 0.8476190

37 73 0.8476190

38 75 0.8476190

39 77 0.8476190

40 79 0.8571429

41 81 0.8571429

42 83 0.8571429

43 85 0.8571429

44 87 0.8476190

45 89 0.8476190

46 91 0.8380952

47 93 0.8380952

48 95 0.8476190

49 97 0.8476190

50 99 0.8571429

> sprintf("k = %f provides the best knn prediction of %f", k_list[which(knn_prediction == max(knn_prediction))[1]], max(knn_prediction))

[1] "k = 7.000000 provides the best knn prediction of 0.857143"

```
> sprintf("k = %f provides the best knn prediction of %f", k_list[which(knn_prediction ==
max(knn_prediction))[1]], max(knn_predict_best))

[1] "k = 7.000000 provides the best knn prediction of 0.820513"
```

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.

Working in financial advising, me and my company are always looking for new clients and prospects. To do so, marketing is required. A clustering model would be a great choice to find out what people in certain groups would like to hear for them to want to work with us. Some of the predictors that come to mind would be:

1. Early retirement – a lot of our current clients are early retirees. We have found that this is a pretty niche thing to know how to work around in this field, so telling others that we deal with it every day could bring some possible prospects.
2. Retirement – most of the other clients that we have are people trying to navigate retirement. They are unaware of things like how to best take Social Security, make sure they don't have to pay IRMAA, etc. Promoting this knowledge to potential clients could bring them to work with us.
3. Our firm size – because we are not a huge firm (about 8 people), clients see all our faces at some point. Larger financial companies can be overwhelming to some people, so finding out if there are people who would rather work with a smaller firm would be beneficial to market to.

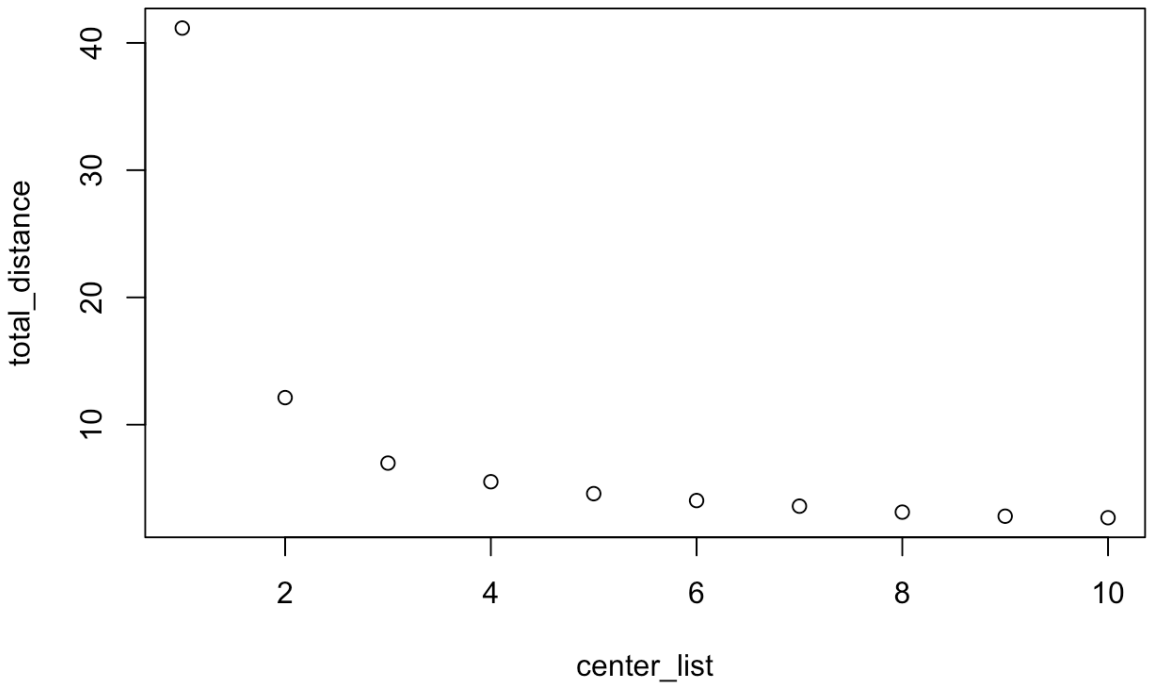
It would be interesting to see if a more targeted marketing strategy using clustering would result in more prospect meetings than our current marketing to everybody and anybody strategy.

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.

Using my suggested value of k = 3, meaning 3 centers, I found that the one center predicts 3 Versicolors and 36 Virginicas, the second center predicts 47 Versicolors and 14 Virginicas, and the third center predicts 50 Setosas. To start, I normalized all of the values I was going to work with to keep them all within the same range. I then tested the clustering model using multiple different total amount of centers – I tested from 1 center to 10 centers. From this model, I was able to calculate the total distances of each data point to its cluster center for each value of k. Using the different k-values and these total distance values, I was able to construct an elbow diagram to help me figure out which k-value looks like the best one. From the graph, I found that the marginal benefit of adding another center started to be small from 3 centers to 4 centers. I then ran another kmeans using 3 centers instead of a range of 1 to 10 centers to get the amount of each iris in each cluster. To find the accuracy of the kmeans model, I took the sum of the largest cluster in each cluster (50+47+36) and divided it by the total data points (150). The accuracy of the kmeans model using 3 centers turned out to be 0.8866667. Just to see if there really was not much of a difference, I ran the second kmeans equation with the other center values, and the accuracy value did not increase enough for it to make sense to create more cluster centers.



R Code and Output

```
pacman::p_load(kernlab, caret, ggplot2)

iris <- read.table("iris.txt", header = TRUE)

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

```
iris_normalized <- as.data.frame(lapply(iris[,c(1,2,3,4)],normalize))

center_list <- c(seq(from = 1, to = 10, by = 1))
total_distance <- c()

for(i in 1:length(center_list)){
 iris_cluster <- kmeans(iris_normalized, centers = center_list[[i]], nstart = 10)
 total_distance[i] <- iris_cluster$tot.withinss
}

plot(center_list, total_distance)

iris_cluster_3 <- kmeans(iris_normalized, centers = 3, nstart = 10)

table(iris$Species, iris_cluster_3$cluster)

accuracy <- (50+47+36)/150
print(accuracy)
```

```
> plot(center_list, total_distance)
> table(iris$Species, iris_cluster_3$cluster)

                1  2  3
 setosa         0  0 50
 versicolor    47  3  0
 virginica     14 36  0
```

```
> accuracy <- (50+47+36)/150


> print(accuracy)
[1] 0.8866667
```