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); and

Methodology:

I have used the kknn package in R and specifically the train.kknn and cv.kknn functions to perform cross-validation on the kknn model. The models are evaluated by the metric classification errors. In order to force the functions to perform classification algorithm, I changed the response variable's data type to factors. For the train.kknn function, I set kmax parameter as 20 so the model tries fitting models with number of neighbors (k) from 1 to 20 and find the best k value through leave one out cross-validation. For the cv.kknn function, I set number of folds as 10 as it is generally a good practice and I looped through values for number of neighbors(k) from 1 to 20.

Results:

The function results of train.kknn are as follows:

```
> #use train.kknn function to perform leave one out cross validation to select
> model_cv_1 <- train.kknn(R1~.,data, kmax=20, kernel = "optimal", scale=TRUE)
> model_cv_1

Call:
train.kknn(formula = R1 ~ ., data = data, kmax = 20, kernel = "optimal",     scale = TRUE)

Type of response variable: nominal
Minimal misclassification: 0.146789
Best kernel: optimal
Best k: 12
```

The best k is chosen at 12 and gives the highest accuracy at 0.853211.

```
> 1-model_cv_1$MISCLASS
     optimal
1  0.8149847
2  0.8149847
3  0.8149847
4  0.8149847
5  0.8516820
6  0.8455657
7  0.8470948
8  0.8486239
9  0.8470948
10 0.8516820
11 0.8516820
12 0.8532110
13 0.8516820
14 0.8516820
15 0.8532110
16 0.8532110
17 0.8532110
18 0.8516820
19 0.8501529
20 0.8501529
```

The function results of cv.kknn are as follows:

```
> acc <- rep(0,20)
>
> for(i in 1:20){
+ model_cv_2 <- cv.kknn(R1~.,data, kcv=10, k=i, scale=TRUE)
+
+ acc[i] <- mean(model_cv_2[[1]][,1] == model_cv_2[[1]][,2])
+
+ }
>
> which.max(acc)
[1] 5
```

```
> acc
 [1] 0.8119266 0.8073394 0.8134557 0.8103976 0.8547401 0.8455657 0.8455657 0.8425076 0.8425076 0.8409786
[11] 0.8455657 0.8470948 0.8440367 0.8547401 0.8516820 0.8470948 0.8440367 0.8425076 0.8455657 0.8516820
```

For this particular k-fold CV run, when value of k equals to 5, the model produces the highest accuracy at 0.8547401.

Discussion:
The two cross-validation functions gave different results for best K value. The reasons could be the following:

1. the train.kknn method performs leave one out cross validation and the cv.kknn method performs k-fold cross validation.
2. The cv.kknn function only performs cross-validation once on a single value of k (number of neighbors), and looping the function through multiple k values cannot ensure the cross validation folds are consistent, so there is a lot of instability to use cv.kknn to find the best k Value. It could be the case that for a particular loop of the function and a particular k value, the cross validation folds are selected by chance that artificially boost the performance of the model. Thus, I would prefer to use train.kknn function to have higher confidence in selecting the best k value.

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

Methodology:
I used the sample method to randomly select data points from the dataset and assigned 60%, 20% and 20% of the population to the train, validation and test data subsets respectively. I then used the kknn function with the train and validation data subsets to find the optimal value of k. Once I obtained the optimal k value from validation step, I again used the kknn function, this time with train and test data subsets along with the optimal k value at 10 to test the model's accuracy on unseen data to get a more objective evaluations of the model's performance in correctly classifying responses.

Results:
The validation step results are as follows:

```
> n = nrow(data)
> idx <- sample(n)
>
> train_idx <- idx[1:floor(0.6 * n)]
> val_idx   <- idx[(floor(0.6 * n) + 1):floor(0.8 * n)]
> test_idx  <- idx[(floor(0.8 * n) + 1):n]
>
> train <- data[train_idx, ]
> val   <- data[val_idx, ]
> test  <- data[test_idx, ]
>
> max_k <- 20
> val_acc  <- rep(0,20)
>
> for (i in 1:max_k){
+
+   model <- kknn(R1 ~ ., train = train, test = val, k = i, scale = TRUE)
+
+
+   val_acc[i] <- mean(model$fitted.values == val[, "R1"])
+ }
>
> which.max(val_acc)
[1] 10
> val_acc
 [1] 0.7862595 0.7862595 0.7862595 0.7862595 0.8167939 0.8167939 0.8320611 0.8473282 0.8473282 0.8549618
[11] 0.8549618 0.8549618 0.8549618 0.8473282 0.8396947 0.8396947 0.8396947 0.8320611 0.8320611 0.8320611
```

It gives the optimal k value at 10 with accuracy at 0.8549618.

The test step results are as follows:

```
> model_final <- kknn(R1 ~ ., train = train, test = test, k =which.max(val_acc) , scale = TRUE)
>
>
> test_acc <- mean(model_final$fitted.values == test[, "R1"])
> test_acc
[1] 0.8549618
```

The test step gives the same accuracy of 0.8549618 as that of the best model in the validation step, thus I am more confident the selected model has good stability in performance.

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.

For example, in a large graduate program course, the instructor may want to assign students into groups to complete group projects. Instead of randomly assign them, in order to ensure well rounded and diversified groups are put together, the instructor can cluster students into different groups and then try as much as possible to ensure each group has at least one student from each of the cluster. This practice can ensure the groups are diversified in backgrounds and skillsets. The predictors can be, for example, employment length, worked industry, whether held a managerial position, number of years of coding experiences and gender (if this information is available).

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.

Methodology:

First, I used the kmeans function to perform clustering on all predictors in the iris dataset and looped through value for number of clusters between 1 and 10 inclusively to find out a possible good number of clusters. I also plotted the within-cluster variance for each value of number of clusters from 1 to 10.

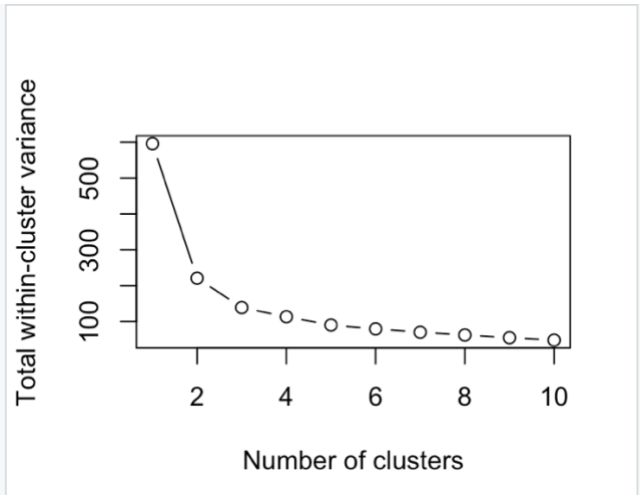
In terms of finding the best combination of predictors, I started by having a closer look at the predictors by summarizing the predictor values per each response category since the response value is available in this case and note that this is usually not possible in most situations where clustering is applied.

Then based on the intuition I got from above, I used the kmeans function with just 2 predictors Petal.Length and Petal.Width and set number of clusters as 3 to create another model.

Next, I will compare the clustering results of the 2 models I built above against the response value that is available in this case to see which combinations of predictors will perform better in that the cluster results match the flower type.

Results:

```
> set.seed(123)
>
> iris_data <- read.table("/Users/jeremyli/Documents/Master's Program/6501/HW 2/Homework2_ISYE6501/data 4.2/iris.txt",
+                   stringsAsFactors = FALSE, header = TRUE)
>
> iris_scaled_data <- scale(iris_data[,1:4])
>
> within_cluster_var <- rep(0,10)
>
> for (i in 1:10) {
+
+   cluster <- kmeans(iris_scaled_data, centers = i, nstart = 25)
+   within_cluster_var[i] <- cluster$tot.withinss
+
+ }
>
> within_cluster_var
 [1] 596.00000 220.87929 138.88836 113.33162  90.20190  79.46523  70.18758  62.37581  54.93464  48.12018
> plot(1:10, within_cluster_var, type = "b",
+      xlab = "Number of clusters",
+      ylab = "Total within-cluster variance")
```



As shown above, the within-cluster variance drops slower as number of clusters increases, which mimic an elbow shape, and it looks like 3 clusters is a good value for the model.

The summarized data below shows the average value of predictors for each response category:

```
> iris_data %>%
+     group_by(Species) %>%
+     summarise(across(c(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width),
+                      ~ mean(.x, na.rm = TRUE), .names = "mean_{.col}"))
# A tibble: 3 × 5
  Species    mean_Sepal.Length mean_Sepal.Width mean_Petal.Length mean_Petal.Width
  <chr>                  <dbl>            <dbl>             <dbl>            <dbl>
1 setosa                  5.01             3.43              1.46            0.246
2 versicolor              5.94             2.77              4.26            1.33
3 virginica               6.59             2.97              5.55            2.03
```

Just a quick glance, it seems that the predictors Petal.Length and Petal.Width have more different average values for each species of flowers, for which one can argue that the two predictors can be used separate the species of flowers more effectively and accurately.

Below are the results for how well the Petal only predictors model and the all-predictors model perform in matching the correct flower type.

```
> # model with all predictors
>
> cluster_all_predictors <- kmeans(iris_scaled_data, centers = 3, nstart = 25)
>
> # model with just Petal.Length and Petal.Width
>
> cluster_petal_only <- kmeans(iris_scaled_data[, c(3,4)], centers = 3, nstart = 25)
>
>
> #compare cluster results against flower types
>
> petal_only %>%
+   count(Species, cluster_petal_only$cluster) %>%
+   group_by(Species) %>%
+   mutate(prop = n / sum(n))
# A tibble: 5 × 4
# Groups:   Species [3]
  Species    `cluster_petal_only$cluster`     n  prop
  <chr>                             <int> <int> <dbl>
1 setosa                                2    50  1
2 versicolor                            1     2  0.04
3 versicolor                            3    48  0.96
4 virginica                             1    46  0.92
5 virginica                             3     4  0.08
>
> petal_only %>%
+   count(Species, cluster_all_predictors$cluster) %>%
+   group_by(Species) %>%
+   mutate(prop = n / sum(n))
# A tibble: 5 × 4
# Groups:   Species [3]
  Species    `cluster_all_predictors$cluster`     n  prop
  <chr>                                <int> <int> <dbl>
1 setosa                                   2    50  1
2 versicolor                               1    39  0.78
3 versicolor                               3    11  0.22
4 virginica                                1    14  0.28
5 virginica                                3    36  0.72
```

As shown above, the Petal-only 2 predictors model results are superior in correctly matching the flower types than the all-predictors model where both models perfectly match the flower type setosa with one of the 3 clusters, but the petal only model more accurately matches flower type versicolor and flower type virginica to one of the remaining 2 clusters.

Discussion:

This is a very interesting question to work on, especially that the response variable is available in the dataset so that we can more objectively measure validity of the combinations of predictors and number of clusters to use for the clustering exercise. However, in real life examples, the professional's domain knowledge will need to play a significant role solving specific clustering problems.