```
### QUESTION 2.1###
#1.     Situation: To obtain a nail technician license, candidates undergo a practical
examination where their skills are assessed by an examiner.  During the acrylic nail
enhancement portion of the exam, the technician must demonstrate proficiency in creating a
flawless and consistent acrylic mixture. The examiner closely observes the process of
applying the acrylic to the nail, evaluating and classifying the mixture as "ideal,"
"runny," or "dry" based on pre-defined criteria. This classification determines the
technician's score and ultimately their licensing status.

#The predictors to the problem:

  #a.   Liquid to powder ratio: this requires a lot of practice to do it right and
quickly. If the liquid is too much compared to the powder, the mixture would be too
“runny”, difficult to form the shape of the nail, and take too long to dry. If the liquid
is too little compared to the powder, the mixture is going to dry super-fast, before the
shape of the nail is formed.

#b.     Waiting time before applying to the nails: this requires a lot of practice. If the
wait time is too short even though you got the perfect ratio, the mixture can be dry
before applying to the nails.

#c.     The type of the powder: some powder brand requires more liquid than others, so you
must experience a lot to feel it when you first try it.

#d.     The type of the liquid: there are runny liquid and regular liquid.

#e.     The room temperature: the winter makes it harder for the mixture to dry so during
wintertime (I don’t know why, it’s just a fact and I don’t have enough knowledge in
Chemistry to explain), so during wintertime, be mindful to use different liquid or less
liquid than normal.

#The technician requires to practice a lot to make an “ideal” mixture consistently before
working on a client. If the mixture turns out “runny” or “dry”, they’ll not get the
license until they master it.




### QUESTION 2.2 ###

#Set as working directory to load the txt file into R
setwd("~/Downloads/IYSE6501/hw1-SP22/data 2.2")

#read the txt file and call it data:
data <- read.table("~/Downloads/IYSE6501/hw1-SP22/data 2.2/credit_card_data-headers.txt",
header= TRUE)
View(data) #Just to see how it looks like




### QUESTION 2.2.1 ###
# call ksvm.  Vanilladot is a simple linear kernel.
library(kernlab)

set.seed(42) #This is from office hour to make sure every thing run the same in case we
use any randomize function
#Convert data into matrix called x:
x <- as.matrix(data[,1:10]) #this code is referenced from the homework

#Use the last comlumn as factor called y:
y <- as.factor(data[,11])
```

```
#Now we creater the SVM model called model based on matrix x and factor y:
model <- ksvm(x,y,type="C-svc", kernal="vanilladot",C=1000, scaled=TRUE) #this code was
provided in the homework, C=1000 provided the highest accuracy after testing manually.

#Calculate a1…am based on xmatrix at model times coefficient at model:
a <- colSums(model@xmatrix[[1]] * model@coef[[1]]) #this code was provided in the
homework, used to calculate the factors a by summing up the total of vectors in xmatrix
multiplying by the corresponding coefficients in the model.
print(a)

#A1         A2         A3         A8         A9        A10        A11        A12        A14
A15
#-51.35778 -21.62044 -58.63362 122.24606  86.19287 -92.95451  96.16194 -64.88062 -71.71629
101.59822


#Calculate a0, provided in the homework
a0 <- -model@b #b is the intercept number that is provided in the model
print(a0)
# 0.6875617

#See what the model predicts
pred <- predict(model,data[,1:10]) #provided in the homework, used to predict the response
of the model
print(pred)

#See what fraction of the model's predictions match the actual classification (accuracy).
print(sum(pred == data[,11]) / nrow(data)) #provided in the homework, used to calculate
the accuracy of the model by comparing the total rights in the prediction to the actual
value in the dataset.
# 0.9831804


### CONCLUSION###
#The equation for credit_card_data is:
-51.35778A1 -21.62044A2 -58.63362A3 +122.24606A8 +86.19287A9 -92.95451A10 +96.16194A11
-64.88062A12 -71.71629A14 +101.59822A15 + 0.6875617 = 0
#The accuracy for SVM model is 98.32%




### QUESTION 2.2.3###

#install kknn package in order to use kknn function as requested:
install.packages("kknn")
library(kknn)

#First of all, test if we could use the entire data as test and train:

#Create a range for k to run:
k_value <- 1:25

# The k value is an integer, but we want to convert it to numeric and store it in a
vector...
# ...because every time we test k, the result might be a float, which is not an integer...
# ...and we need a vector to store the results after testing each k, which we will call
'results'.

results <- numeric(length(k_value))

#We are testing a lot of k to see what k gives the highest accuracy, so we use for loop:
#Those codes are referenced from the office hour, and this site:
https://www.learnbymarketing.com/tutorials/k-nearest-neighbors-in-r-example/ and chatGPT.
```

```
set.seed(42)
#We are trying to run k from 1 to 25, so we use for loop:
for (i in 1:length(k_value)) {
  k <- k_value[i]
  model_kknn <- kknn(R1~., train=data, test= data, k=k, distance = 2, kernel = "optimal",
scale=TRUE) #train the model with the current value of k, call the model model_kknn
  prediction <- predict(model_kknn) #get the prediction from model_kknn
  accuracy <- sum(prediction==data$R1)/ length(data$R1) #
  print(paste("k=",k,"Accuracy=", accuracy)) #Print k and accuracy together
  results[i] <- accuracy
}

###results###
"k= 1 Accuracy= 1"
"k= 2 Accuracy= 1"
"k= 3 Accuracy= 1"
"k= 4 Accuracy= 1"
"k= 5 Accuracy= 0.951070336391437"
"k= 6 Accuracy= 0.943425076452599"
"k= 7 Accuracy= 0.934250764525994"
"k= 8 Accuracy= 0.935779816513762"
"k= 9 Accuracy= 0.932721712538226"
"k= 10 Accuracy= 0.923547400611621"
"k= 11 Accuracy= 0.920489296636086"
"k= 12 Accuracy= 0.915902140672783"
"k= 13 Accuracy= 0.914373088685015"
"k= 14 Accuracy= 0.909785932721712"
"k= 15 Accuracy= 0.906727828746177"
"k= 16 Accuracy= 0.903669724770642"
"k= 17 Accuracy= 0.90519877675841"
"k= 18 Accuracy= 0.90519877675841"
"k= 19 Accuracy= 0.90519877675841"
"k= 20 Accuracy= 0.90519877675841"
"k= 21 Accuracy= 0.903669724770642"
"k= 22 Accuracy= 0.902140672782875"
"k= 23 Accuracy= 0.900611620795107"
"k= 24 Accuracy= 0.902140672782875"
"k= 25 Accuracy= 0.900611620795107"


#The result show that there are 4 k values that give 100% accuracy, and the rest of them
give less than 90% accuracy...
#...so we need to SPLIT the data to a TEST set and a TRAIN set.

#I am going to split the data into 80% for training, and 20% for testing.
#To make sure the data is split THE SAME WAY EVERY TIME, I use set.seed():

set.seed(42)
data_split <- sample(1:nrow(data), size = 0.8*nrow(data)) #split the data into 80-20
data_train <- data[data_split,] #the train data uses 80% of the row, with all the column
followed.
data_test <- data[-data_split,] #the test data uses 20% of the row, or the rest of the row
after split, with all the column followed.

#The 2 new data set don't know that they each have a target variable in them, so...
#...this step is for defining the response and the predictors for them:

#In this case, we only use the test data so: defining the response for test data:

test_data_response <- data_test$R1 #the column R1 is the response

set.seed(42)
data$R1 <- factor(data$R1) #define the response for the origional dataset
#Those codes are learned from https://www.learnbymarketing.com/tutorials/k-nearest-
```

```
neighbors-in-r-example/ and chatGPT#
for (i in 1:length(k_value)) {
  k <- k_value[i]  # Get the current value of k

  # Train the k-NN model with the current k
  model_knn <- kknn(R1~.,train=data_train, test = data_test, distance=2, kernel="optimal",
k=k, scale=TRUE) #this code is referenced from office hour

  # Get predictions from model_knn
  prediction <- predict(model_knn)

  # Calculate accuracy
  accuracy <- sum(prediction == data_test$R1) / length(data_test$R1) #Store the accuracy
in the results vector
  results[i] <- accuracy  #Store the accuracy for the current k
}

print(results)

###result###
###[1]0.8244275 0.8244275 0.8244275 0.8244275 0.8396947 0.8473282 0.8549618 0.8625954
0.8625954 0.8625954
###[11]0.8625954 0.8549618 0.8625954 0.8625954 0.8625954 0.8702290 0.8702290 0.8702290
0.8702290 0.8625954
###[21] 0.8625954 0.8625954 0.8625954 0.8625954 0.8625954


### CONCLUSION###
#The highest accuracy for kknn model is 87.23% with k= 16, 17, 18, 19




#Back to question 2.2.2:
#try the polynomial method:
#Reuse matrix x and factor y and the data above:
degrees <- c(2,3,4,5,6) #create a wide range of degree to test
results_poly <- data.frame(degree= integer(0), accuracy=numeric(0)) # this code is
referenced from chatGPT
scaled_data <- scale(data[, 1:10])

#with a range of degree, we use for loop:
for (degree in degrees) {
  model_poly <- ksvm(scaled_data, as.factor(data[,11]), type="C-svc", kernel="polydot",
degree=degree, C=100, scaled=FALSE) #This code is referenced from the code provided in the
homework
  prediction <- predict(model_poly,scaled_data)
  accuracy_poly <- sum(prediction==as.factor(data[,11])) / length(y)
  results_poly <- rbind(results_poly, data.frame(degree=degree, accuracy=accuracy_poly))
#This code is referenced from chatGPT
}
print(results_poly)


###result###
#degree  accuracy
#1      2 0.8639144
#2      3 0.8639144
#3      4 0.8639144
#4      5 0.8639144
#5      6 0.8639144
```

```
#It shows that with each degree, the accuracy of this model stays the same at 86.39%.
#There are 2 types of issue here:
#1) My codes are wrong
#2) The information is not enough to do a polynomial method.
```