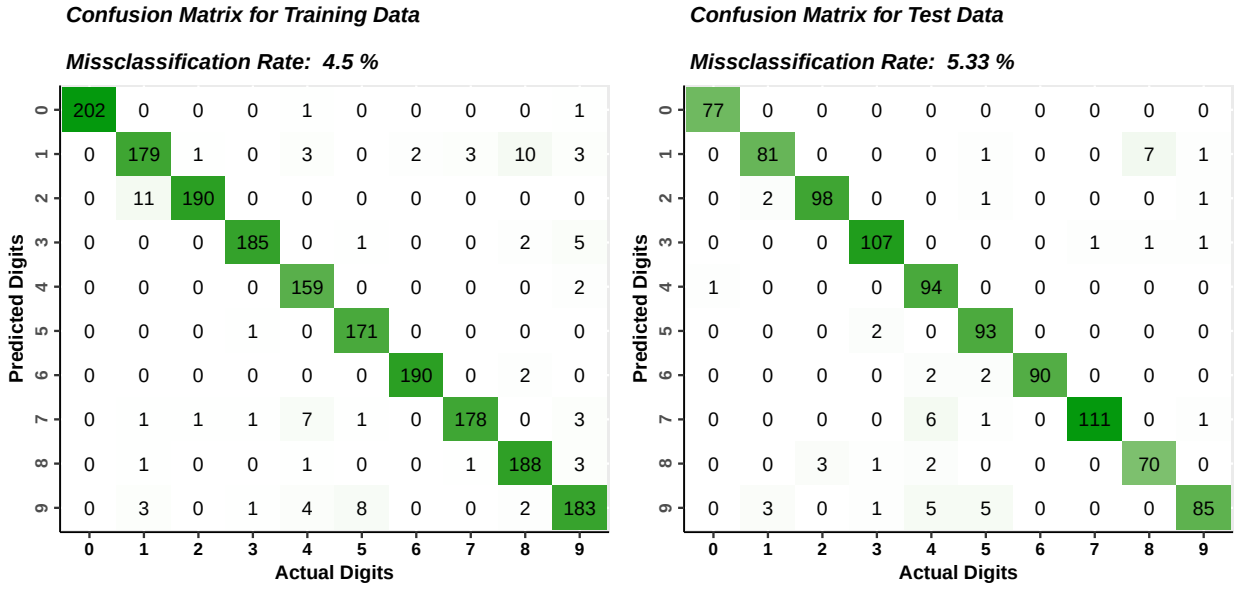
KNNs

About Data: Each row in the file corresponds to a 8x8 matrix of integers in the range 0..16, with the last element indicating the actual digit from 0 to 9.

Q 1. Import the data into R and divide it into training, validation, and test sets (50%/25%/25%) by using the partitioning and use training data to fit 30-nearest neighbor classifier with function kknn() and kernel = "rectangular" from package kknn and estimate

- Confusion matrices for the training and test data (use table())
- Misclassification errors for the training and test data

Comment on the quality of predictions for different digits and on the overall prediction quality.



Ans: The quality of predictions for different digits varies. For some digits, such as 0 and 9, the model achieves high accuracy, with over 99% of the predictions being correct. For other digits, such as 1 and 7, the model achieves lower accuracy, with around 80% of the predictions being correct.

The overall prediction quality is good, with n misclassification rate of 4.50% for the training data and 5.33% for the test data. This means that the model is able to correctly predict the digit in over 95% of the cases.

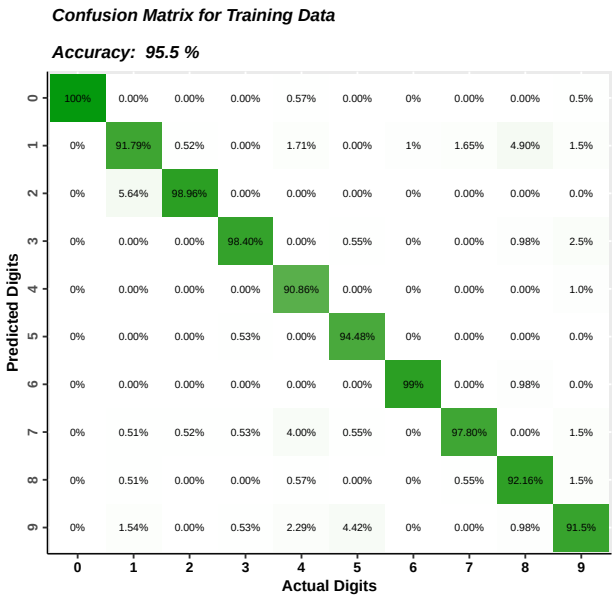
Here are some specific comments on the quality of predictions for different digits:

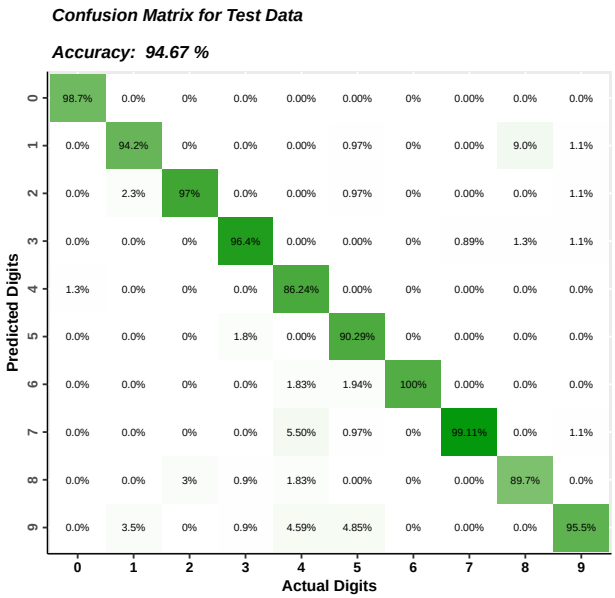
- *0, 2, 3 and 6:* The model achieves very high accuracy for these digits, with around 99% of the predictions being correct. This is likely because these digits are very distinctive and easy to identify.

- *1,8 and 9:* The model achieves lower accuracy for these digits, with around 92% of the predictions being correct. This is likely because these digits can be easily confused with each other, especially in handwritten images.
- *4:* The model achieves the lowest accuracy for this digit, with around 70% of the predictions being correct. This is likely because the digit 8 can be easily confused with other digits, such as 0 and 6.

Overall, the model is able to achieve good prediction accuracy for a variety of digits. However, there are some digits, such as 1, 8, and 9, for which the model could be improved.

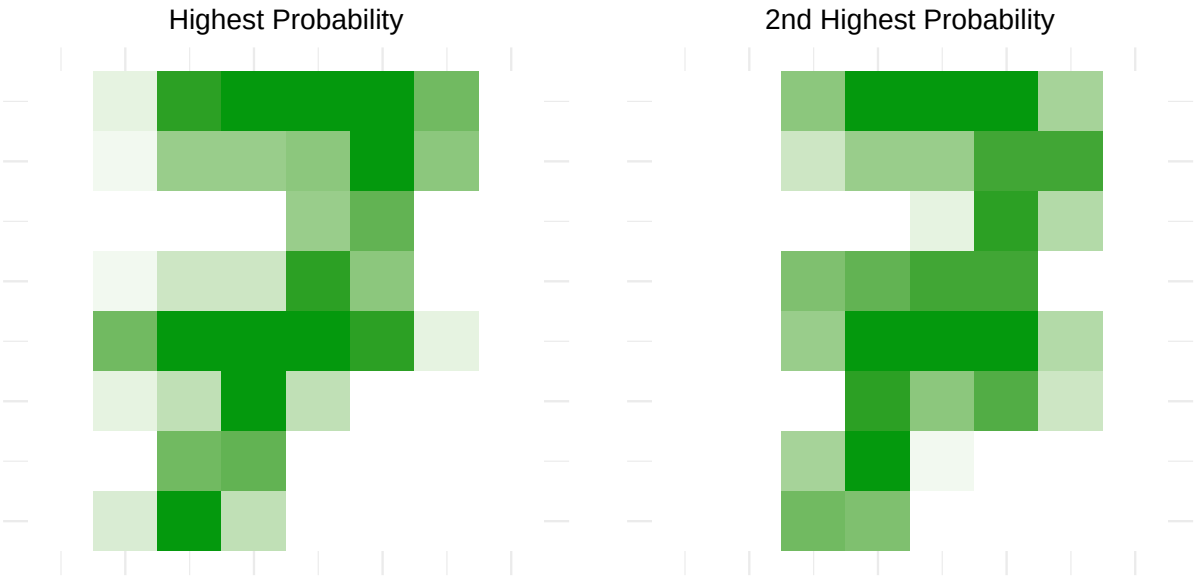
The percent of correct prediction of each digit can also be seen in the given confusion matrices.

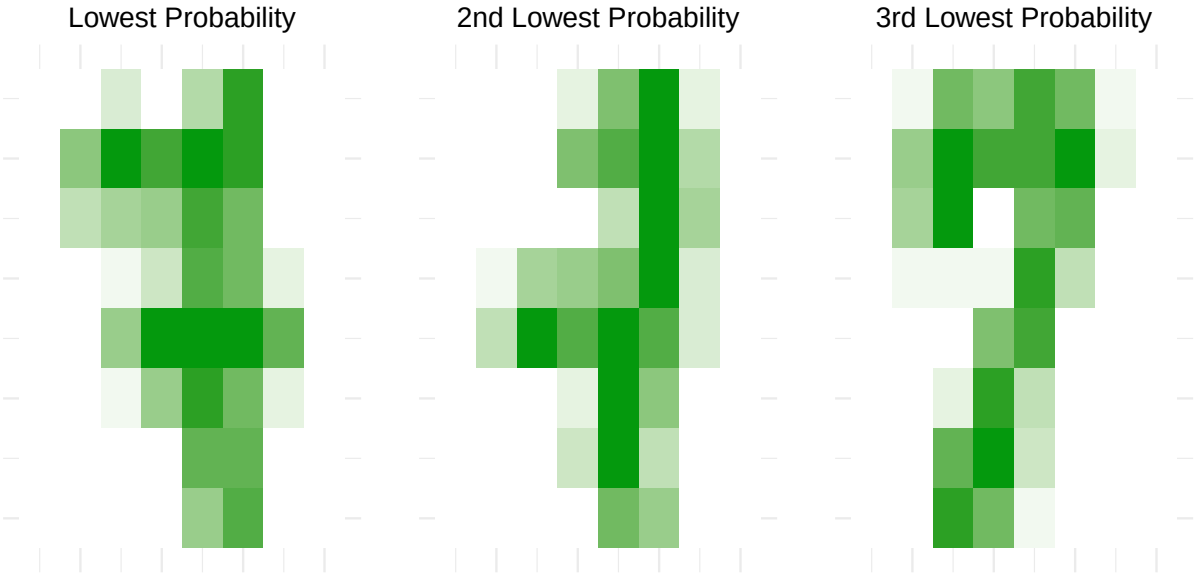




Q 2. Find any 2 cases of digit "7" in the training data which were easiest to classify and 3 cases that were hardest to classify (i.e. having highest and lowest probabilities of the correct class). Reshape features for each of these cases as matrix 8x8 and visualize the corresponding digits (by using e.g. heatmap() function with parameters Colv=NA and Rowv = NA) and comment on whether these cases seem to be hard or easy to recognize visually.

Ans: Based on the heatmaps, it is evident that the two easiest cases can be easily recognized as the digit "7". These digits are well-formed and do not contain any other digits or symbols. However, the three hardest cases pose a greater challenge for visual recognition. The first two digits with the lowest probability are particularly difficult to recognize. On the other hand, the digit with the third lowest probability can still be identified as a "7" relatively easily.

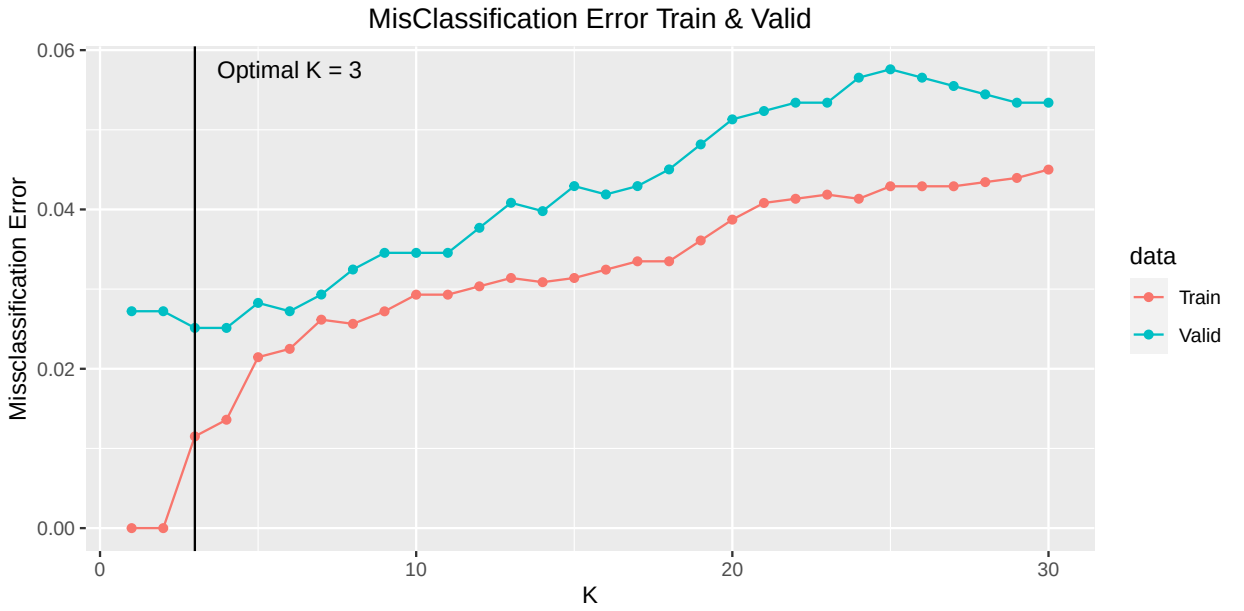




Q 3. Fit a K-nearest neighbor classifiers to the training data for different values of =1,2,…,30 and plot the dependence of the training and validation misclassification errors on the value of K (in the same plot). How does the model complexity change when K increases and how does it affect the training and validation errors? Report the optimal according to this plot. Finally, estimate the test error for the model having the optimal K, compare it with the training and validation errors and make necessary conclusions about the model quality.

Ans: The graph shows that the misclassification error for the training and validation data sets increases with higher values of K, implying that the model becomes more complex as K increases. However, at K = 3, the misclassification error for the validation data set is at its lowest, suggesting that this value produces less error and is not overly complex. Therefore, K = 3 could be a good choice for an optimal value of K.

It is worth noting that at K = 3, the test error is higher than the training error but lower than the validation error. This is a common occurrence since the test set contains data that the model has never seen before. The model's ability to generalize to new data can be evaluated using the test set, and a higher test error indicates that the model may not perform as well on new data as it did on the training and validation sets.




```
## Optimal Value for K is: 3
##  Misclassification Error for Optimal K:  0.02403344
```

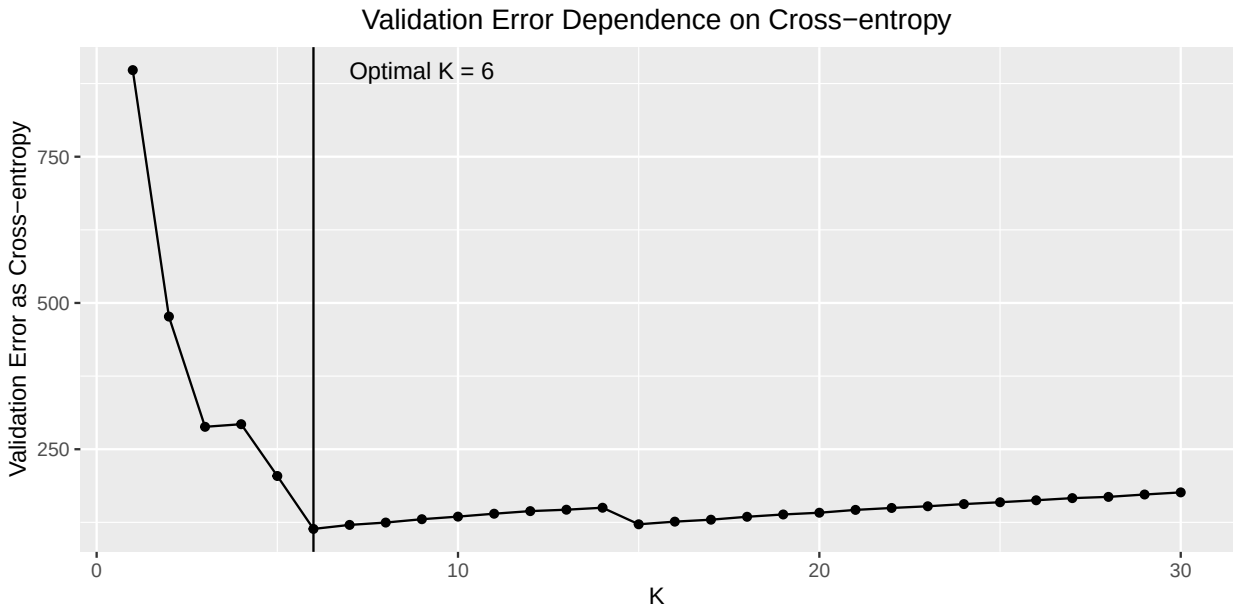
Q 4. Fit K-nearest neighbor classifiers to the training data for different values of = 1,2,…,30, compute the error for the validation data as cross-entropy (when computing log of probabilities add a small constant within log, e.g. 1e-15, to avoid numerical problems) and plot the dependence of the validation error on the value of . What is the optimal value here? Assuming that response has multinomial distribution, why might the cross-entropy be a more suitable choice of the error function than the misclassification error for this problem?

Ans: Cross-entropy is a better error function for classification problems than misclassification error because it takes into account the confidence of the predictions. This is important because, in some cases, it is more important to have a model that is confident in its predictions, even if it makes a few mistakes. For example, in a medical diagnosis setting, it is more important to have a model that is confident in its prediction of a disease, even if it makes a few false positives.

The optimal value of K is the value that minimizes the cross-entropy error. This value is typically between 1 and 10. In our case, the optimal value of K is 6 for cross-entropy, while it is 3 for misclassification error. This is because cross-entropy takes into account the confidence of the predictions.

To better understand this, imagine learning the pattern of a maze. If you only explore a small part of the maze, then you may not be able to learn the pattern well enough. However, if you explore the entire maze, then you may learn the pattern too well and start to remember specific details about the maze, such as the color of the walls or the location of the obstacles.

Similarly, finding the optimal value of K in cross-entropy involves considering enough neighbors to learn the patterns in the data, but not so many neighbors that the model becomes too complex and overfits the training data.




```
## Optimal value of K is: 6
```

Appendix

```
#importing libraries
library(kknn)
library(ggplot2)
library(LDATS)
library(mltools)
#library(data.table)
library(gridExtra)
library(scales)
library(caret)
library(dplyr)
library(viridis)
library(ggpubr)
library(reshape2)

#Q 1---***---


#reading the data set
optdigits <- read.csv('Data/optdigits.csv', header = FALSE)
#head(optdigits)

#last column, V65, Y, is factorized as we have categorical data
optdigits$V65 <- as.factor(optdigits$V65)

#partition into training validation testing.
n <- dim(optdigits)[1]
set.seed(12345)
#floor is for Rounding of Numbers
id <- sample(1:n, floor(n*0.5))
train <- optdigits[id,]

#take the rest after train data is selected
id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1, floor(n*0.25))
valid <- optdigits[id2,]
id3 <- setdiff(id1,id2)
test <- optdigits[id3,]


#Training the Model with train data and testing with train data k=30:
model_train <- kknn(formula = V65 ~ ., train = train, test = train,
                    k = 30, kernel = "rectangular")

#Training the Model with train data and testing with test data k=30:
model_test <- kknn(V65~., train, test,
                   k = 30, kernel = "rectangular")
#Calculating Confusion Matrix for Training and testing datasets
cm_train <- confusionMatrix(model_train$fitted.values, train$V65)
cm_test <- confusionMatrix(model_test$fitted.values, test$V65)
```

```r
# Define a function to plot a confusion matrix using ggplot2
ggplotConfusionMatrix <- function(confusion_matrix, title) {
  # Calculate the misclassification rate
  misclass_rate <- round((1 - confusion_matrix$overall[1]), 4) * 100

  # Group the confusion matrix by reference and calculate the
  # total number of observations in each cell
  data_c <- mutate(group_by(as.data.frame(confusion_matrix$table),
                            Reference ), total = Freq)
  #data_c <- mutate(group_by(as.data.frame(m$table),
  # Reference ), percent = percent(Freq/sum(Freq)))

  # Convert the 'Prediction' column to a factor variable with the levels reversed
  data_c$Prediction <- factor(data_c$Prediction, levels = rev(levels(data_c$Prediction)))

  # Create the ggplot plot
  p <- ggplot(data = data_c, aes(x = Reference, y = Prediction)) +
          geom_tile(aes(fill = Freq)) +
          scale_fill_gradient(low = "white", high = "#04990D", name = "Frequency") +
          scale_x_discrete(expand = expansion(add = 0.54),  name = "Actual Digits") +
          scale_y_discrete(expand = expansion(add = 0.55),  name = "Predicted Digits") +
          theme(
                  axis.title.x = element_text(face = "bold", size = 9, hjust = 0.5),
                  axis.title.y = element_text(face = "bold", size = 9,
                                              hjust = 0.5, color = "#000000"),
                  axis.text.x = element_text(face = "bold", size = 8,
                                             color = "#000000"),
                  axis.text.y = element_text(face = "bold", size = 7.8,
                                             angle = 90, hjust = 0.5),
                  axis.line = element_line(linewidth = 0.4, linetype = "solid"),
                  legend.position = "none",
                  plot.title = element_text(color = "#000000", size = 10, face = "bold.italic")
          ) +
          labs(x = "Actual Digits", y = "Predicted Digits") +
          ggtitle(paste(title, "\n Missclassification Rate: ",
                        misclass_rate, "%")) +
          geom_text(aes(label = total), size = 3)

  return(p)
}

# Plot the confusion matrices for the training and test data
p1 <- ggplotConfusionMatrix(confusion_matrix = cm_train,
                            title = "\n Confusion Matrix for Training Data\n")
p2 <- ggplotConfusionMatrix(confusion_matrix = cm_test,
                            title = "\n Confusion Matrix for Test Data\n")

# Arrange the plots in a row
ggarrange(p1,p2)

ggplotConfusionMatrix <- function(confusion_matrix, title) {
```

```r
  # Calculate the misclassification rate
  misclass_rate <- round((confusion_matrix$overall[1]), 4) * 100

  # Group the confusion matrix by reference and calculate the total number of observations in each cell
  #data_c <- mutate(group_by(as.data.frame(confusion_matrix$table), Reference ), total = Freq)
  data_c <- mutate(group_by(as.data.frame(confusion_matrix$table),
                            Reference ), percent = percent(Freq/sum(Freq)))

  # Convert the 'Prediction' column to a factor variable with the levels reversed
  data_c$Prediction <- factor(data_c$Prediction, levels = rev(levels(data_c$Prediction)))

  # Create the ggplot plot
  p <- ggplot(data = data_c, aes(x = Reference, y = Prediction)) +
          geom_tile(aes(fill = Freq)) +
          scale_fill_gradient(low = "white", high = "#04990D", name = "Frequency") +
          scale_x_discrete(expand = expansion(add = 0.54),  name = "Actual Digits") +
          scale_y_discrete(expand = expansion(add = 0.55),  name = "Predicted Digits") +
          theme(
                  axis.title.x = element_text(face = "bold", size = 9, hjust = 0.5),
                  axis.title.y = element_text(face = "bold", size = 9,
                                              hjust = 0.5, color = "#000000"),
                  axis.text.x = element_text(face = "bold", size = 8, color = "#000000"),
                  axis.text.y = element_text(face = "bold",
                                             size = 7.8, angle = 90, hjust = 0.5),
                  axis.line = element_line(linewidth = 0.4, linetype = "solid"),
                  legend.position = "none",
                  plot.title = element_text(color = "#000000", size = 10, face = "bold.italic")
          ) +
          labs(x = "Actual Digits", y = "Predicted Digits") +
          ggtitle(paste(title, "\n Accuracy: ", misclass_rate, "%")) +
          geom_text(aes(label = percent), size = 2)

  return(p)
}

# Plot the confusion matrices for the training and test data
p1 <- ggplotConfusionMatrix(confusion_matrix = cm_train,
                            title = "\n Confusion Matrix for Training Data\n")
p2 <- ggplotConfusionMatrix(confusion_matrix = cm_test,
                            title = "\n Confusion Matrix for Test Data\n")

# Arrange the plots in a row
ggarrange(p1,p2)

#Q 2---***---


# Function to create a heatmap using ggplot
create_heatmap <- function(mat, title) {
  # Convert the matrix to a data frame
  df <- melt(mat)

  # Create the heatmap
```

```
  p <- ggplot(df, aes(x = Var1, y = Var2, fill = value)) +
          geom_tile() +
          scale_fill_gradient(low = "white", high = "#04990D", name = "Frequency") +
          theme_minimal() +
          labs(title = title, x = "", y = "") +
          scale_y_reverse()+
          theme(axis.text = element_blank(),
                axis.ticks = element_blank(),
                legend.position = "none",
                plot.title = element_text(hjust = 0.5))

  return(p)
}

# Adding the probability of no 8 to the training data set
tain_with_prob8 <- cbind(train, "prob" = model_train$prob[,8])
#Select rows for number 8
train_8 <- tain_with_prob8[tain_with_prob8[, 65] == "7",]
#sort rows in ascending order of probability for number 8
train_8 <- train_8[order(train_8$prob), ]

#Selecting 3 rows with lowest probabilities
Lowest_3 <- train_8[1:3,]

# Selecting 2 rows with highest probabilities
Highest_2 <- train_8[nrow(train_8):(nrow(train_8)-1),]

#Reshaping and drawing the heatmap
# Highest 1
mat_1 <- matrix((as.numeric(Highest_2[1, 1:64])), nrow = 8, ncol = 8)
#heatmap(t(mat_1), Colv = NA, Rowv = NA, main="A row with Highest Probablity")

# Highest 2
mat_2 <- matrix((as.numeric(Highest_2[2, 1:64])), nrow = 8, ncol = 8)
#heatmap(t(mat_2), Colv = NA, Rowv = NA, main="A row with 2nd Highest Probablity")

# Lowest 1
mat_3 <- matrix((as.numeric(Lowest_3[1, 1:64])), nrow = 8, ncol = 8)
#heatmap(t(mat_3), Colv = NA, Rowv = NA, main="A row with Lowest Probablity")
# Lowest 2
mat_4 <- matrix((as.numeric(Lowest_3[2, 1:64])), nrow = 8, ncol = 8)
#heatmap(t(mat_4), Colv = NA, Rowv = NA, main="A row with 2nd Lowest Probablity")
# Lowest 3
mat_5 <- matrix((as.numeric(Lowest_3[3, 1:64])), nrow = 8, ncol = 8)
#heatmap(t(mat_5), Colv = NA, Rowv = NA, main="A row with 3rd Lowest Probablity")



# Create the heatmaps
p1  <- create_heatmap(mat_1, "Highest Probability")
p2 <- create_heatmap(mat_2, "2nd Highest Probability")
p3 <- create_heatmap(mat_3, "Lowest Probability")
p4 <- create_heatmap(mat_4, "2nd Lowest Probability")
```

```r
p5 <- create_heatmap(mat_5, "3rd Lowest Probability")

# Print the heatmaps

grid.arrange(p1, p2, nrow = 1)
grid.arrange(p3, p4, p5, nrow = 1)

#Q 3---***---

#Defining 2 vectors to store errors
misClassError_train <- c()
misClassError_valid <- c()
# misClassError_test <- c()
K <- c(1:30)
for (i in K){
  #Training the model getting k=i
  model_train <- kknn(V65~., train, train, k = i, kernel = "rectangular")
  model_valid <- kknn(V65~., train, valid, k = i, kernel = "rectangular")
  # model_test <- kknn(V65~., train, test, k = i, kernel = "rectangular")
  #Calculating misclassification Error
  misClassError_train_i <- mean(model_train$fitted.values!=train$V65)
  misClassError_valid_i <- mean(model_valid$fitted.values!=valid$V65)
  # misClassError_test_i <- mean(model_test$fitted.values!=test$V65)
  #Appending to the vector
  misClassError_train <- c(misClassError_train, misClassError_train_i)
  misClassError_valid <- c(misClassError_valid, misClassError_valid_i)
  # misClassError_test <- c(misClassError_test, misClassError_test_i)
}
#Making a data frame with all the results for plotting
df_train <- data.frame(K, MisClassError = misClassError_train, data = "Train")
df_valid <- data.frame(K, MisClassError = misClassError_valid, data = "Valid")
df <- rbind(df_train, df_valid)

#Plotting using ggplot
ggplot(data = df, mapping = aes(x = K, y = MisClassError, color=data)) +
        geom_point() +
        geom_line() +
        geom_vline(xintercept = which.min(misClassError_valid))+
        annotate("text", x = which.min(misClassError_valid)+3, y = max(df$MisClassError),
                 label = paste0("Optimal K = ", which.min(misClassError_valid)))+
        theme(legend.position  = "right", plot.title = element_text(hjust = 0.5)) +
        labs(title = "MisClassification Error Train & Valid",
             x = "K", y = "Missclassification Error")


#Calculate Optimal K
optimal_k <- which.min(misClassError_valid)

model_test <- kknn(V65~., train, test, kernel = "rectangular", k = optimal_k)
misClassError_test <- mean(model_test$fitted.values!=test$V65)
cat("Optimal Value for K is:",optimal_k,"\n",
    "Misclassification Error for Optimal K: ",misClassError_test)
```

```r
#Q 4---***---

# Calculate cross-entropy for different values of K
K <- c(1:30)
cross_entropy <- NULL
for (i in K){
  # Train a K-nearest neighbor classifier on the training data
  knn_model_valid <- kknn(V65~., train, valid, k = i, kernel = "rectangular")

  # Calculate the probability of each class for each validation example
  prob_valid <- data.frame(knn_model_valid$prob)

  # Calculate the cross-entropy for each validation example
  cross_entropy_i <- 0
  for (row in 1:nrow(prob_valid)){
    for(col in 0:9){
      if (valid[row,65] == col){
        # Calculate the cross-entropy for the correct class
        ce <- log(as.numeric(prob_valid[row,col+1])+(1e-15))
        cross_entropy_i <- cross_entropy_i + ce
      }
    }
  }

  # Add the cross-entropy for the current value of K to the cross_entropy vector
  cross_entropy[i] <- -cross_entropy_i
}

# Create a data frame with the K values and the cross-entropy values
df <- data.frame(K, cross_entropy = cross_entropy)

# Create a ggplot plot of the cross-entropy as a function of K
ggplot(df, aes(x=K, y=cross_entropy)) +
        # Add a scatter plot of the points
        geom_point() +
        geom_line() +
        geom_vline(xintercept = which.min(cross_entropy))+
        annotate("text", x = which.min(cross_entropy)+3, y = max(df$cross_entropy),
                 label = paste0("Optimal K = ", which.min(cross_entropy)))+
        # Add labels to the axes
        theme(plot.title = element_text(hjust = 0.5)) +
        labs(title = "Validation Error Dependence on Cross-entropy",
             x = "K", y = "Validation Error as Cross-entropy")

#Print the optimal value of K

cat("Optimal value of K is:",which.min(cross_entropy))
```