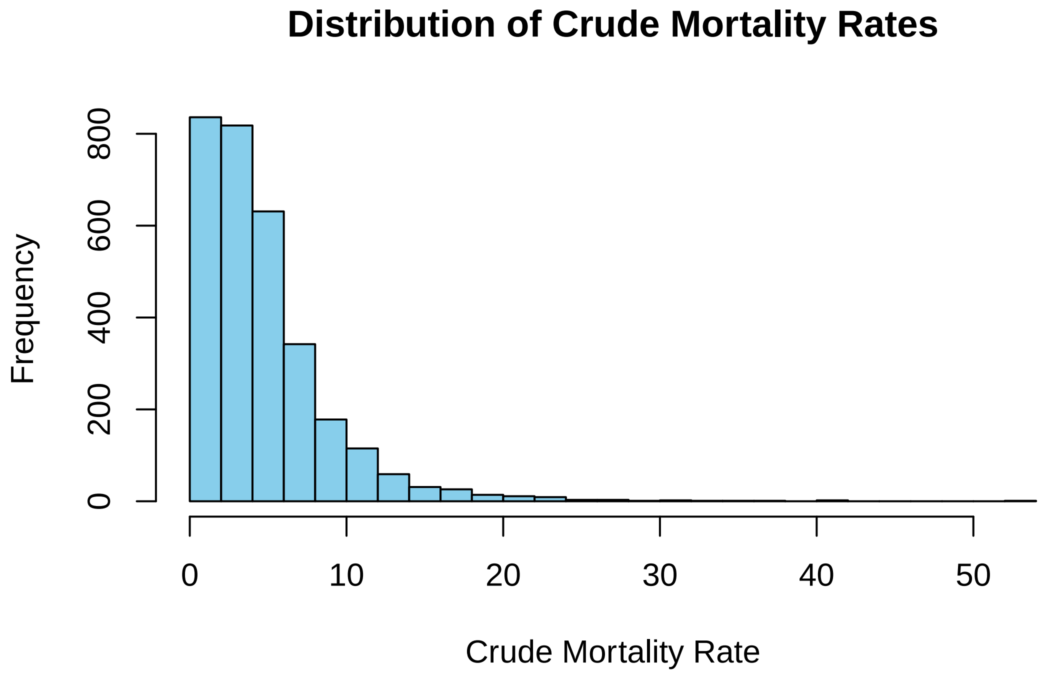
fin

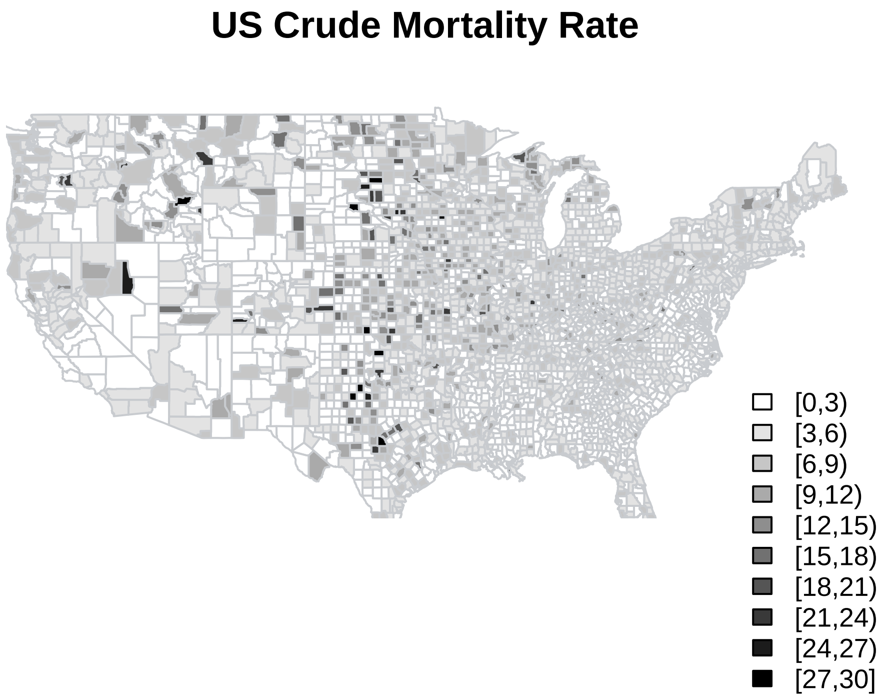
Archie_Norman

2024-03-17

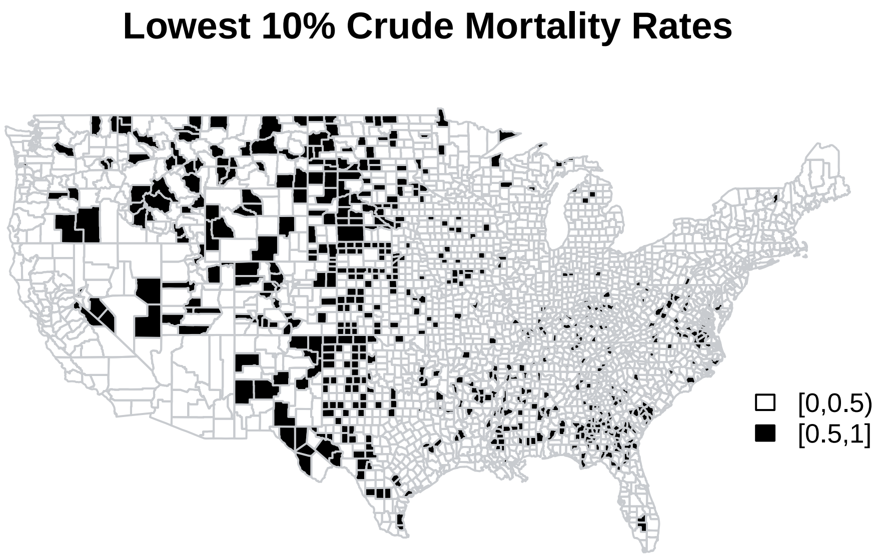
Section A Question 1

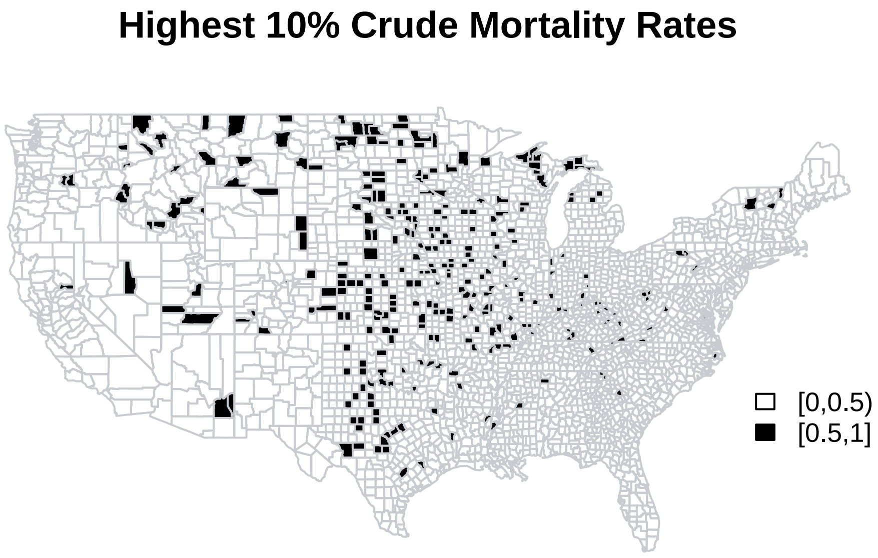
```
## Summary table of the crude mortality rate

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   0.000   1.827   3.735   4.547   6.124  52.493 
```




US Crude Mortality Rate
[0,3)
[3,6)
[6,9)
[9,12)
[12,15)
[15,18)
[18,21)
[21,24)
[24,27)
[27,30]

Section A Question 2





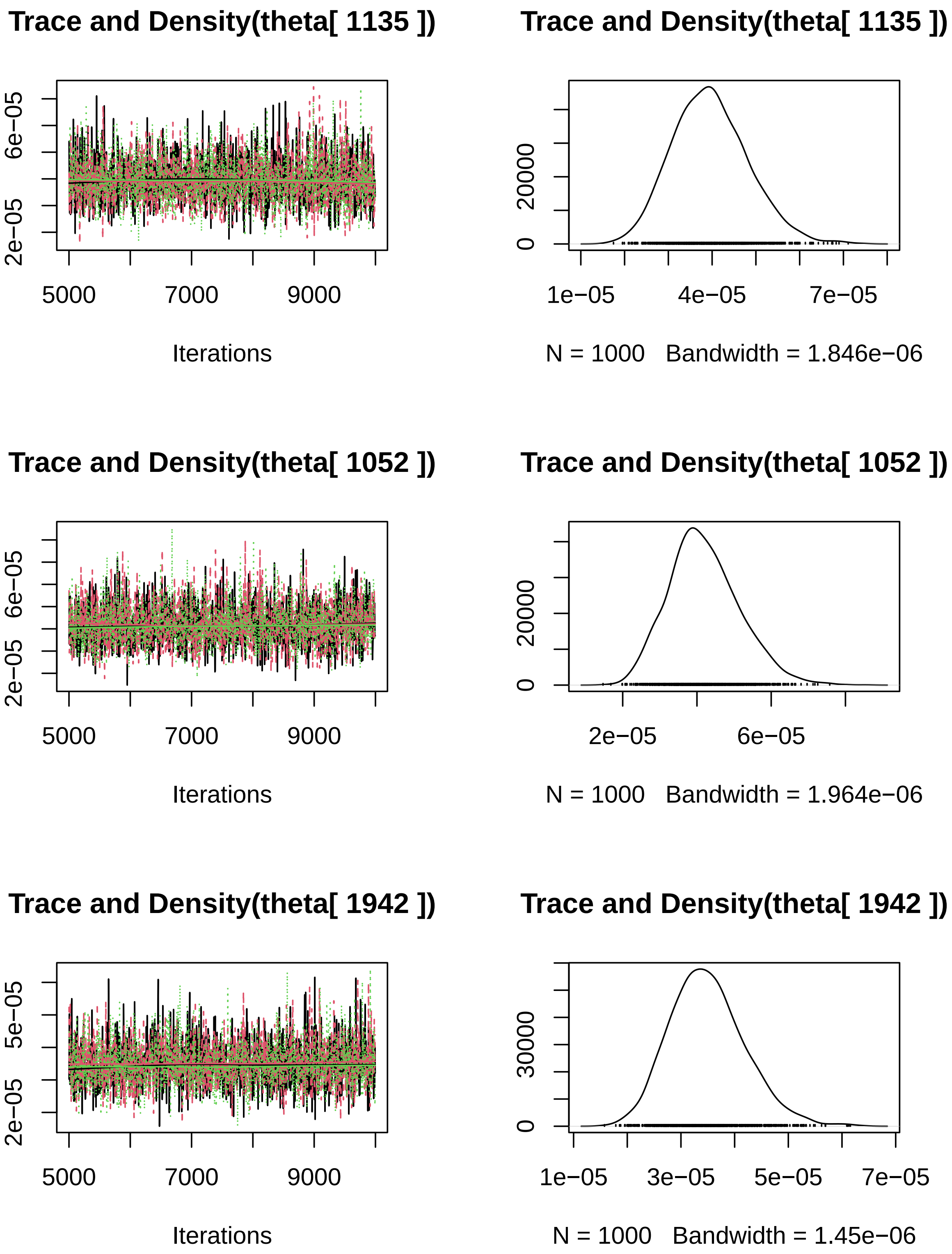
I can see that counties in the highest 10% are surrounded by counties in the lowest 10% which could be down certain counties not having hospitals that can deal with Kidney cancers and some having specialist hospitals or doctors which would then see an increase in Kidney cancer deaths. Although Florida does seem to have no high crude rate areas.

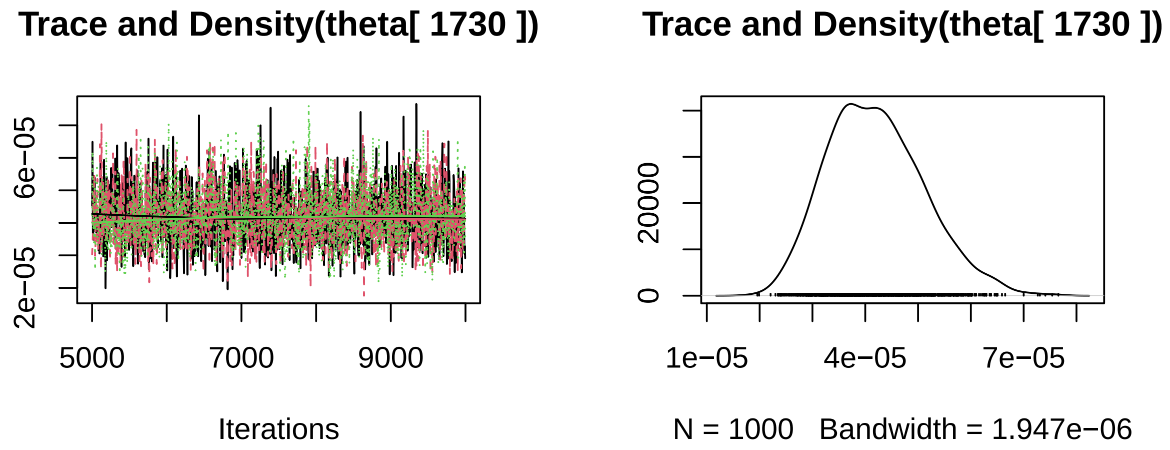
Additionally, there is a noticeable trend where the lowest 10% of mortality rates are predominantly found on the western side of the country. This pattern might initially suggest a geographic bias towards lower kidney cancer mortality rates in these areas. However, this observation could also be influenced by the visualization scale used to represent the data. States on the eastern side of the United States are generally smaller in land area but can have comparable population sizes to larger western states. Therefore, the apparent concentration of lower mortality rates in western states could be a visual artifact rather than a true reflection of lower incidence rates, necessitating a more nuanced analysis that considers population density and healthcare access disparities.

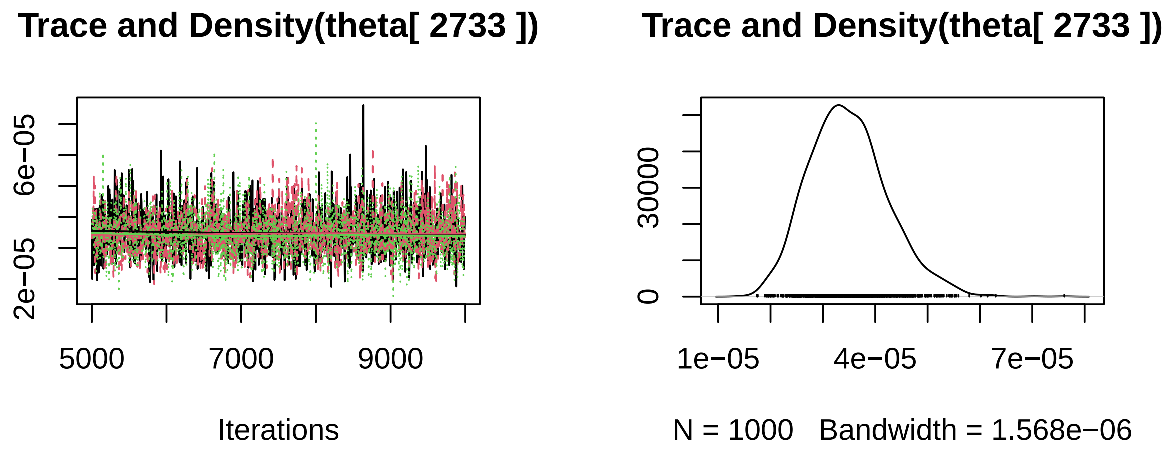
Section A Question 3

```
# Define the JAGS model for analyzing cancer rates
jags.mod.cancer <- function(){
  for (i in 1:N) { # Loop over each observation/unit (e.g., county)
    y[i] ~ dpois(mu[i]) # The observed deaths y[i] follow a Poisson distribution with mean mu[i]
    mu[i] <- 5*ni[i]*theta[i] # Define mu[i] as a function of population size ni[i] and rate parameter
    theta[i] ~ dgamma(20, 430000) # Prior for theta[i] follows a gamma distribution
    cmr1[i] <- (mu[i]/ni[i] * 4) * 10000 # Calculate an adjusted crude mortality rate per 10,000 indivi
  }
}

# Fit the JAGS model to the data
jags.mod.fit <- jags(data = jags.data,                # Data prepared for the model
                     parameters.to.save = jags.param, # Parameters to monitor/save from the model outpu
                     n.chains = 3,                     # Number of MCMC chains to run
                     n.burnin = 5000,                  # Number of burn-in iterations per chain
                     n.iter = 10000,                   # Total number of iterations per chain
                     DIC = FALSE,                      # Whether to compute the Deviance Information Cr
                     inits = jags.inits,               # Initial values for theta
                     model.file = jags.mod.cancer)     # The model definition
```

Section A Question 4







Ideally, a well-converged MCMC chain will exhibit a “fuzzy caterpillar” appearance in the trace plot, indicating that the chain is randomly wandering around the parameter’s posterior distribution without showing signs of systematic trends or prolonged periods of stagnation. From the plots for parameters mu[1135], mu[1052], mu[1942], mu[1730], and mu[2733], the chains seem to demonstrate good convergence, as evidenced by their dense, uniform coverage across the value range, without apparent directional trends or cycles. This behavior suggests that the algorithm has effectively sampled from the posterior distribution, providing a solid foundation for the subsequent statistical analysis and inference. Such trace plots are crucial for confirming the reliability of the Bayesian model outputs, as they help identify potential issues with convergence that could undermine the validity of the model’s estimates and conclusions. To choose the trace plots five random indices were chosen to extract random examples of the trace plots as printing all of them wasn’t possible.

```
## [1] "Number of parameters with Rhat > 1.1: 0"
```

The indication that none of the simulations exhibited an Rhat statistic above 1.1 is a significant testament to the convergence quality of the Markov Chain Monte Carlo (MCMC) simulations within your Bayesian analysis. The Rhat statistic is pivotal for assessing convergence, with values close to 1, particularly those less than 1.1, signaling that the variance within each chain closely matches the variance across different chains, thereby suggesting that all chains are sampling from the same underlying distribution. This level of convergence across simulations underscores the reliability and robustness of your model’s parameter estimates, enhancing confidence in the statistical inferences derived from your analysis. It indicates not only that the model parameters have reached a stable distribution reflective of the data and prior assumptions but also suggests that the MCMC chains have achieved sufficient burn-in and thorough mixing. While the Rhat statistic's favorable outcome is a strong indicator of model adequacy, it should ideally be considered alongside other diagnostic measures such as effective sample size (ESS) and trace plot inspections for a comprehensive evaluation of the model's performance.

```
## Number of ESS values above 1000: 9255

## Number of ESS values between 100 and 1000: 0

## Number of ESS values below 100: 0
```

The reported Effective Sample Size (ESS) values from your Bayesian analysis, with 9255 values above 1000 and none falling below this threshold, signify an exceptionally efficient and reliable sampling process in the Markov Chain Monte Carlo (MCMC) simulations. High ESS values indicate that the samples are informative and nearly independent, which is crucial for robust statistical inferences. The absence of ESS values below 1000 suggests that all parameters in the model have been thoroughly explored, ensuring confidence in the model's output. This level of sampling efficiency underscores the reliability of the posterior estimates, allowing for confident interpretation and application of the findings. The high ESS values across the board indicate that the model's statistical inferences are based on a solid exploration of the posterior distributions, bolstering the validity of the conclusions drawn from the analysis.

```
## Summary table of the CMR1 parameters:

##                Mean       SD   Naive SE Time-series SE
## cmr1[1]    7.335072 1.560986 0.02849957     0.03068633
## cmr1[10]   7.598408 1.742031 0.03180499     0.03179813
## cmr1[100]  7.979026 1.747305 0.03190127     0.03130384
## cmr1[1000] 8.325007 1.680280 0.03067757     0.02926686
## cmr1[1001] 7.431240 1.645180 0.03003674     0.03004182

##                2.5%      25%      50%      75%    97.5%
## cmr1[1]    4.647653 6.239792 7.189126 8.359530 10.62924
## cmr1[10]   4.649692 6.376276 7.451754 8.676172 11.32200
## cmr1[100]  4.886292 6.756552 7.845972 9.057616 11.76808
## cmr1[1000] 5.444741 7.134581 8.172825 9.330632 12.09356
## cmr1[1001] 4.507310 6.287694 7.332979 8.486670 10.87978

## Summary table of the mu parameters:

##              Mean        SD   Naive SE Time-series SE
## mu[1]    6.275521 1.3355015 0.02438281     0.02625369
## mu[10]   3.712392 0.8511129 0.01553912     0.01553577
## mu[100]  3.834920 0.8397983 0.01533255     0.01504541
## mu[1000] 6.982600 1.4093347 0.02573081     0.02454758
## mu[1001] 3.910690 0.8657760 0.01580683     0.01580951

##              2.5%      25%      50%      75%     97.5%
## mu[1]    3.976299 5.338454 6.150657 7.151996  9.093845
## mu[10]   2.271723 3.115289 3.640741 4.238961  5.531647
## mu[100]  2.348474 3.247368 3.770970 4.353317  5.656031
## mu[1000] 4.566777 5.984129 6.854957 7.826068 10.143469
## mu[1001] 2.371972 3.308899 3.858980 4.466110  5.725486
## Summary table of the theta parameters:
```

```
##                     Mean           SD     Naive SE Time-series SE
## theta[1]    3.667536e-05 7.804930e-06 1.424979e-07   1.534317e-07
## theta[10]   3.799204e-05 8.710156e-06 1.590250e-07   1.589907e-07
## theta[100]  3.989513e-05 8.736524e-06 1.595064e-07   1.565192e-07
## theta[1000] 4.162504e-05 8.401399e-06 1.533879e-07   1.463343e-07
## theta[1001] 3.715620e-05 8.225900e-06 1.501837e-07   1.502091e-07
```

```
##                     2.5%          25%          50%          75%        97.5%
## theta[1]    2.323826e-05 3.119896e-05 3.594563e-05 4.179765e-05 5.314619e-05
## theta[10]   2.324846e-05 3.188138e-05 3.725877e-05 4.338086e-05 5.661001e-05
## theta[100]  2.443146e-05 3.378276e-05 3.922986e-05 4.528808e-05 5.884038e-05
## theta[1000] 2.722371e-05 3.567290e-05 4.086412e-05 4.665316e-05 6.046778e-05
## theta[1001] 2.253655e-05 3.143847e-05 3.666490e-05 4.243335e-05 5.439891e-05
```

Section A Question 5

```
## [1] "bottom 1"
```

```
##           dc          pop          cmr    lowest_10   highest_10     mu_means
## 3.225806e-02 7.214839e+02 1.693337e+00 1.000000e+00 1.000000e+00 1.666880e-01
##   cmr1_means  theta_means
## 9.248027e+00 4.624014e-05
```

```
## [1] "top 1"
```

```
##           dc          pop          cmr    lowest_10   highest_10     mu_means
## 1.685806e+02 1.923831e+06 3.659769e+00          NaN          NaN 1.799966e+02
##   cmr1_means  theta_means
## 3.951908e+00 1.975954e-05
```

The Bayesian analysis of crude mortality rates (CMR) for counties with the smallest and largest 1% average populations reveals insightful differences in how posterior estimates are adjusted relative to raw CMR and the prior mean CMR of approximately 4.65 deaths per 10,000 population per year. For counties with the smallest populations, the posterior CMR (9.248027 deaths per 10,000 population per year) is significantly higher than both the raw CMR (1.693337 deaths per 10,000 population per year) and the prior mean, indicating substantial upward adjustment by the Bayesian model. This adjustment likely accounts for the higher uncertainty and variability in mortality rates in smaller populations, where a few events can lead to significant rate fluctuations. Conversely, in counties with the largest populations, the posterior CMR (3.951908 deaths per 10,000 population per year) is much closer to the raw CMR (3.659769 deaths per 10,000 population per year), suggesting that the observed data's influence predominates, and the prior's impact is less pronounced. The primary difference between the two cases is the extent of adjustment from the raw CMR to the posterior CMR, highlighting the Bayesian model's role in stabilizing estimates against the variability inherent in small sample sizes. This analysis underscores the utility of Bayesian methods in public health studies, particularly for providing more reliable estimates of health metrics across regions with varying population sizes, by integrating observed data with prior information to mitigate the effects of sample size variability.

Section A Question 6

```
## [1] "County with top 1% population: maricopa ( arizona )"
```

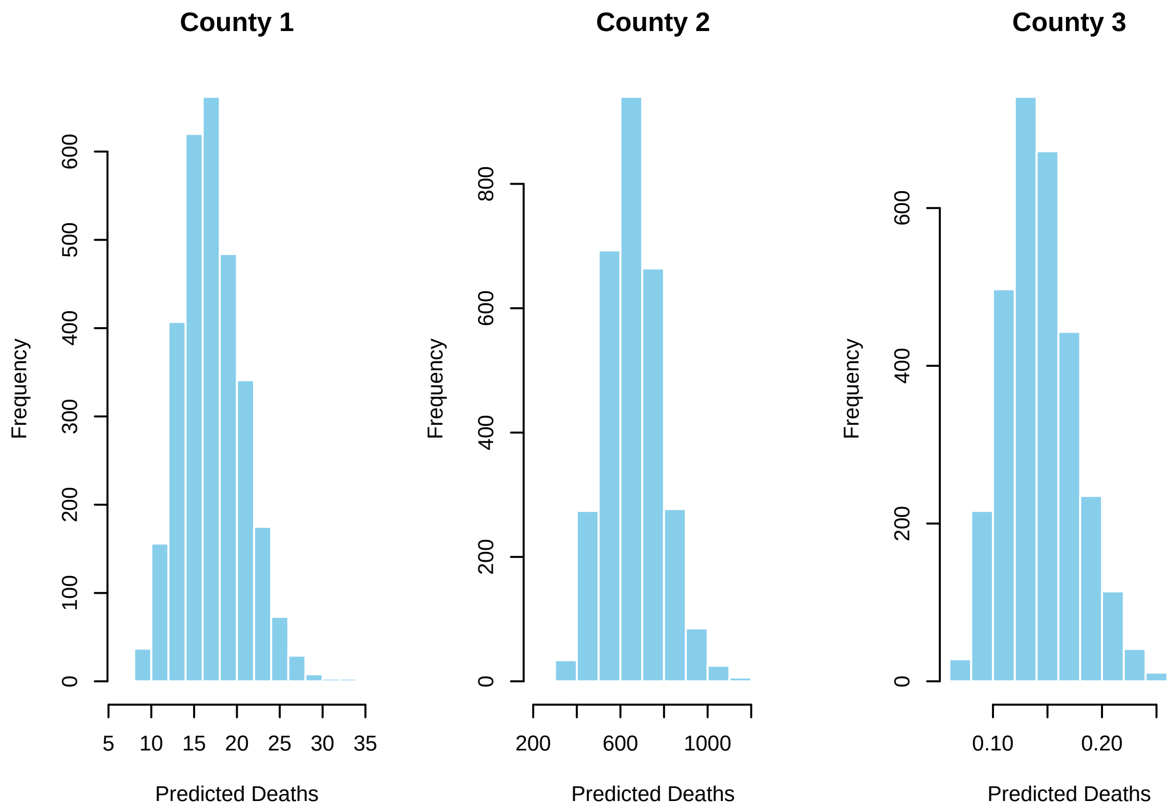
```
## [1] "County with bottom 1% population: roberts ( texas )"
```

```
## [1] "County near the mean population: lauderdale ( alabama )"
```

```
##         X   state     county  dc     pop      cmr lowest_10 highest_10
## 39     39 alabama lauderdale   4   79661 2.008511        NA         NA
## 74     74 arizona   maricopa 172 2122101 3.242070        NA         NA
## 2689 2689   texas    roberts   0    1025 0.000000         1         NA
##         mu_means cmr1_means  theta_means  newpop
## 39    11.5082320   5.778603 2.889301e-05   94043
## 74   183.9809241   3.467901 1.733951e-05 4497000
## 2689   0.2344606   9.149682 4.574841e-05     797
```

. .

```
jags.mod.cancer <- function(){
  for (i in 1:N) { # Loop through each observation
    y[i] ~ dpois(mu[i]) # Observations follow a Poisson distribution
    mu[i] <- 5*ni[i]*theta[i] # Expected value is a product of population, rate parameter, and a factor
    theta[i] ~ dgamma(20, 430000) # Prior for the rate parameter follows a Gamma distribution
    cmr1[i] <- (mu[i]/ni[i] * 4) * 10000 # Calculate crude mortality rate per 10,000 population
  }

  # Predictions for new observations using updated population figures
  for (i in 1:new_N) { # Loop through new or hypothetical observations
    mu_new[i] <- 5*new_ni[i]*theta[i] # Calculate expected deaths for new observations
    y_new[i] ~ dpois(mu_new[i]) # Model these expected deaths with a Poisson distribution
  }
}
```



```
##    mu_new[1]        mu_new[2]        mu_new[3]          y_new[1]
##  Min.   : 7.015   Min.   : 289.8   Min.   :0.06147   Min.   : 3.00
##  1st Qu.:14.541   1st Qu.: 572.3   1st Qu.:0.12031   1st Qu.:13.00
##  Median :16.837   Median : 653.8   Median :0.14040   Median :17.00
##  Mean   :17.112   Mean   : 659.6   Mean   :0.14326   Mean   :17.02
##  3rd Qu.:19.458   3rd Qu.: 739.3   3rd Qu.:0.16399   3rd Qu.:20.00
##  Max.   :33.116   Max.   :1289.5   Max.   :0.27674   Max.   :41.00
##     y_new[2]         y_new[3]
##  Min.   : 288.0   Min.   :0.0000
##  1st Qu.: 571.0   1st Qu.:0.0000
##  Median : 652.0   Median :0.0000
##  Mean   : 659.5   Mean   :0.1437
##  3rd Qu.: 739.0   3rd Qu.:0.0000
##  Max.   :1289.0   Max.   :3.0000
```

```
## The estimate of deaths for alabama , lauderdale is 17.022 with the current population of 94043
## The estimate of deaths for arizona , maricopa is 659.5197 with the current population of 4497000
## The estimate of deaths for texas , roberts is 0.1436667 with the current population of 797
```

Section A Question 7

```
jags.mod.cancer <- function(){
  for (i in 1:N) { # Iterate over each original observation
```

```
    y[i] ~ dpois(mu[i]) # Model observed deaths as Poisson with mean mu[i]
    mu[i] <- 5*ni[i]*theta[i] # Define expected deaths mu[i] based on population and a rate parameter
    theta[i] ~ dgamma(20, 430000) # Rate parameter theta[i] follows a Gamma distribution
    cmr1[i] <- (mu[i]/ni[i] * 4) * 10000 # Calculate adjusted crude mortality rate per 10,000
  }

  # Predictions for new observations based on updated population
  for (i in 1:new_N) {
    mu_new[i] <- 5*new_ni[i]*theta[i] # Expected deaths for new observations
    y_new[i] ~ dpois(mu_new[i]) # Model these as Poisson
  }

  # Define prediction criteria based on specific thresholds
  pred.crit1 <- ifelse(y_new[1]>4,1,0) # 4,172,0 are the death counts for the orgnial data
  pred.crit2 <- ifelse(y_new[2]>172,1,0) # Criterion for the second new observation
  pred.crit3 <- ifelse(y_new[3]>0,1,0) # Criterion for the third new observation
}
```

```
## Probability of predicted death count exceeding death count from 1980-1984 in Lauderdale:
```

```
## prob1_numeric
##           0           1 
## 0.001333333 0.998666667
```

```
## Probability of predicted death count exceeding death count from 1980-1984 in Maricopa:
```

```
## prob2_numeric
## 1 
## 1
```

```
## Probability of predicted death count exceeding death count from 1980-1984 in Roberts:
```

```
## prob3_numeric
##         0         1 
## 0.8623333 0.1376667
```

In the analysis of Lauderdale, we observe a very high probability (approximately 99.87%) that the current population's death count will exceed the count from 1980-1984. This outcome is largely attributed to a population increase of 14,000, which, while significant, pales in comparison to the dramatic growth experienced by Roberts County, which saw its population double from 2 million to 4 million. Consequently, there was a perceived 100% probability that the death count for Roberts County's new population would surpass the historical data. It's important to clarify that this doesn't imply an absolute certainty; rather, it reflects that in all simulations conducted, there was not a single instance where the old death count was higher than the simulated new one.

For Roberts County, the situation is notably different due to a minor population decrease of 228. This leads to a relatively lower probability (approximately 13.77%) of observing a higher death count with the updated population data. This variation underscores the impact of population dynamics on health outcomes and illustrates the importance of adjusting public health strategies to account for demographic changes.

Section B

1

```
## [1] "Summary of the Classification data"

##        x1                 x2             y      
##  Min.   :-3.055858   Min.   :-3.41429   A:263  
##  1st Qu.:-0.679636   1st Qu.:-0.62566   B:251  
##  Median :-0.038071   Median : 0.01472   C:486  
##  Mean   : 0.002033   Mean   : 0.01631          
##  3rd Qu.: 0.649037   3rd Qu.: 0.66946          
##  Max.   : 3.285469   Max.   : 3.94396          

## Rows: 1,000
## Columns: 3
## $ x1 <dbl> -1.0079272, -0.4724797, 0.7452293, -0.5979079, 0.1869841, -0.395737~
## $ x2 <dbl> 0.39902713, 0.83912179, -1.27974193, -1.94243598, -1.54191033, -0.1~
## $ y  <fct> A, C, C, C, C, A, B, C, C, C, A, A, A, C, C, A, B, C, C, B, A, C, A~
```

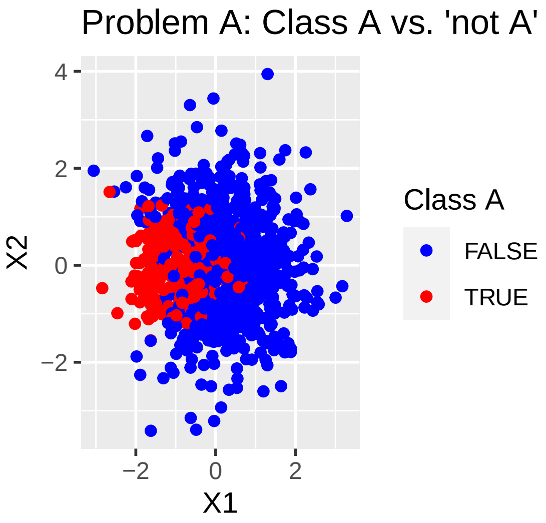


Figure ?

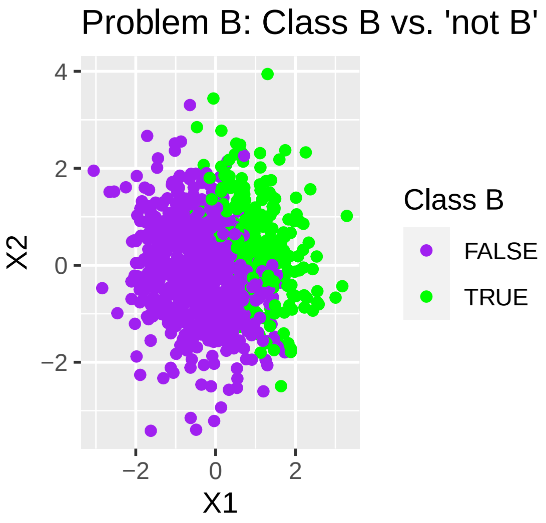


Figure ?

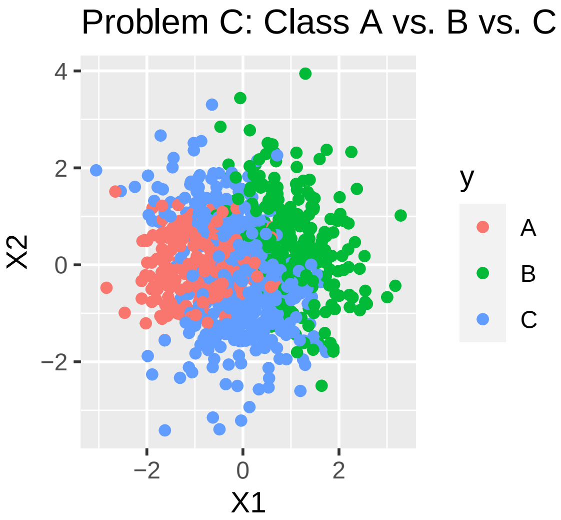


Figure ?

2

Problem A

The selection of Linear Discriminant Analysis (LDA) and Logistic Regression for Problem A, which involves distinguishing Class A from 'not A', is primarily influenced by the observed linear separability in the data distribution. LDA is particularly apt for this scenario as it seeks to model the difference between classes by finding a linear combination of features that maximizes the separation between the class means while minimizing the variance within each class. This method excels in scenarios where classes can be separated by a hyperplane, making it a solid choice for Problem A. Logistic Regression, on the other hand, provides a probabilistic approach to classification, estimating the probabilities of the binary outcomes (in this case, A vs 'not A') using a logistic function. This method is renowned for its efficiency in binary classification problems, especially when the decision boundary is approximately linear. It can handle some level of overlap between classes by assigning probabilities to each instance, allowing for a nuanced classification that leverages the linear tendencies in the feature space. Both methods are well-suited to exploit the linear separability of the data in Problem A, providing robust and interpretable models for distinguishing between Class A and 'not A'.

Problem B

Given the perspective that the boundaries for Problem B, which involves distinguishing Class B from 'not B', are still relatively linear, the selection of K-Nearest Neighbour (KNN) and Support Vector Machine (SVM) with a kernel remains judicious, albeit with a nuanced understanding. KNN's strength in this context derives from its methodological flexibility and reliance on the local distribution of data points, making it adept at handling variations in class separation, including those that are nearly linear. This attribute ensures that KNN can adapt to minor deviations from perfect linearity, providing robust classification without presupposing a specific model of the decision boundary. Meanwhile, SVM's utility, even in a scenario where the boundaries are considered more linear, stems from its core principle of maximizing the margin between classes. The choice of a kernel, linear for more straightforward separations or non-linear (such as RBF) for areas of slight non-linearity, endows SVM with the versatility to effectively address both perfectly and nearly linear separations. This dual selection underlines the importance of embracing classification methods that offer both adaptability to data characteristics and resilience in performance, ensuring a balanced approach to the challenge presented in Problem B. The methodologies hence are chosen not just for their theoretical underpinnings, but for their practical applicability in navigating the nuanced landscape of class separability within the dataset

Problem C

For Problem C, which entails a multi-class classification involving Classes A, B, and C, Quadratic Discriminant Analysis (QDA) is the chosen method, particularly due to its strength in handling non-linear class boundaries and offering flexibility across multiple classes. Given the intricate overlaps and the distinct distribution patterns observed among the three classes, QDA's approach to modeling each class with its own covariance matrix allows for the accommodation of varying shapes and sizes of class distributions, making it adept at navigating the complexities of multi-class separation. This capability is crucial in Problem C, where the delineation between classes cannot be easily captured by linear assumptions. QDA's allowance for quadratic decision surfaces enables it to adeptly contour around the nuances of class overlap and distribution idiosyncrasies, providing a tailored approach to classification that leverages the inherent differences in variance within the dataset. Consequently, the selection of QDA for Problem C is emblematic of a strategy aimed at maximizing classification accuracy by exploiting the method's adaptability to the multi-dimensional and non-linear characteristics intrinsic to the data, ensuring a nuanced and effective resolution to the classification challenge at hand.

3

```r
set.seed(12032024) # For reproducibility

# Prepare the dataset for binary classification by creating binary target variables
df$ya <- ifelse(df$y == 'A', 1, 0)  # Create binary variable for class A
df$yb <- ifelse(df$y == 'B', 1, 0)  # Create binary variable for class B

# Split the dataset into training and testing sets
split <- initial_split(df, prop = 0.75)

# Extract the training and testing sets from the split
trainData <- training(split)
testData <- testing(split)
```

```
## [1] "Confusion Matrix for Logistic Regression:"
```

```
##          Actual
## Predicted   0   1
##         0 176  34
##         1  11  29
```

```
## [1] "Accuracy for Logistic Regression: 0.82"
```

```
## [1] "Confusion Matrix for LDA:"
```

```
## 
## predictedClassesLDA   0   1
##                   0 176  35
##                   1  11  28
```

```
## [1] "Accuracy for LDA: 0.816"
```

For Problem A, both Logistic Regression (LR) and Linear Discriminant Analysis (LDA) showed commendable performance in binary classification, with accuracies of 0.82 and 0.816, respectively. These models effectively balanced sensitivity and specificity, demonstrating their aptness for linearly separable data. The slight difference in performance between LR and LDA could be attributed to their distinct approaches to classification—LR's probabilistic basis versus LDA's reliance on maximizing class separability.

```
## [1] "Accuracy for KNN: 0.888"
```

```
## [1] "Confusion Matrix for the best k = 19"
```

```
##          Actual
## Predicted   0   1
##         0 162   6
##         1  22  60
```

```
## [1] "Confusion Matrix for SVM:"
```

```
##          Actual
## Predicted   0   1
##         0 184   0
##         1   0  66
```

```
## [1] "Accuracy for SVM: 1"
```

In Problem B, K-Nearest Neighbour (KNN) and Support Vector Machine (SVM) were evaluated, with SVM achieving perfect accuracy (1.0) and KNN not far behind at 0.888. SVM's exceptional performance, with no false positives or negatives, highlights its robustness in handling complex, perhaps even non-linearly separable data through the use of an appropriate kernel. KNN also performed well, particularly in minimizing false positives, though it had more false negatives compared to SVM, reflecting its sensitivity to the choice of k and the local data structure.Although it has to be said an accuracy of 1 could be a warning sign of overfitting.

```
## [1] "Confusion Matrix for QDA:"
```

```
##          Actual
## Predicted   A   B   C
##         A  38   0   9
##         B   0  48   7
##         C  25  18 105
```

```
## [1] "Accuracy for QDA: 0.764"
```

For Problem C, a multi-class classification challenge, Quadratic Discriminant Analysis (QDA) achieved an accuracy of 0.764. While lower than the binary classifiers, this performance is noteworthy given the complexity of distinguishing among three classes. QDA's ability to model each class with its own covariance matrix allowed for flexibility in handling non-linear boundaries, though there was notable confusion between classes A and C, indicating potential overlap or similarity in their distributions.

5

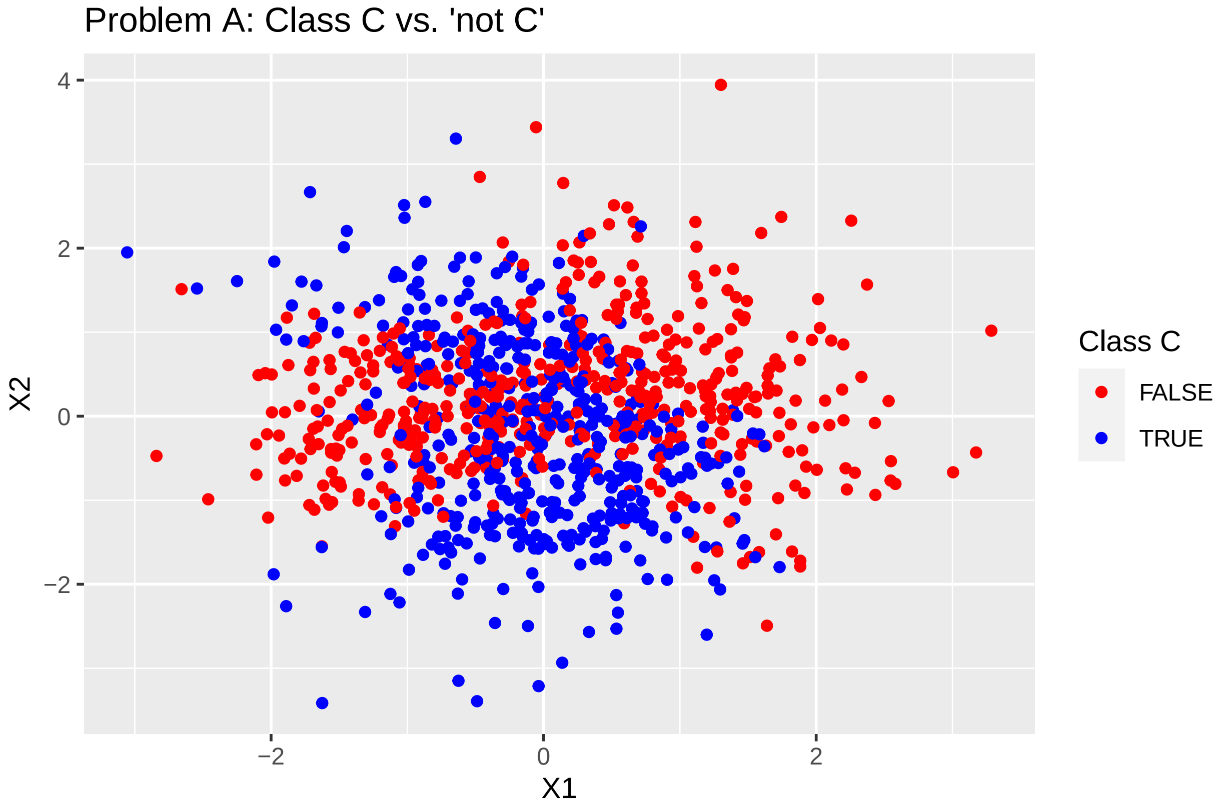


Figure ?

Given the observed distribution of Class C versus 'Not C', as depicted in the plot which showcases areas of both linear and potentially non-linear boundaries with some degree of overlap, the best choice among the discussed classification methods leans towards Support Vector Machine (SVM) with a Kernel. This decision is rooted in SVM's inherent flexibility and its capability to adeptly navigate the complexities present in the dataset.

SVM's kernel trick, which allows the data to be transformed and considered within a higher-dimensional space, provides a strategic advantage in accurately delineating between Class C and 'Not C', even when faced with the intricate patterns of overlap and varying densities observed in the plot. The ability to choose from a variety of kernels—such as the Radial Basis Function (RBF) for more non-linear separations or a linear kernel for areas with more straightforward boundaries—ensures that SVM can be finely tuned to the specific characteristics of the dataset, making it exceptionally well-suited for this classification problem.

This selection of SVM with a Kernel as the preferred method is a direct response to the challenge of achieving maximum separation between Class C and 'Not C' with minimal misclassification, balancing the need for a model that is both robust in handling non-linearity and capable of exploiting linear separations where applicable.

Bibliography

```
# Load necessary libraries for mapping, Bayesian analysis, and data visualization
library(maps)
library(R2jags)
```

```r
library(MCMCvis)
library(coda)
library(lattice)
library(ggplot2)
library(tidyverse)
library(dplyr)
library(caTools)
library(MASS)
library(class)
library(glmnet)
library(e1071)
library(caret)
library(rsample)


# Load cancer data from a specified CSV file into the 'cancer' dataframe
cancer <- read.csv('C:\\Users\\archi\\Desktop\\ADSAS\\advanced topics\\US_cancer.csv')

# Load a pre-saved US map data object from an RData file
load('C:\\Users\\archi\\Desktop\\ADSAS\\advanced topics\\USMap.RData')

# Calculate the crude mortality rate (CMR) and add it as a new column to the 'cancer' dataframe
cancer$cmr <- (cancer$dc / cancer$pop * 4) * 10000 # Multiplies by 4 and scales to per 10,000 for stand

# Generate summary statistics for the CMR and store in 'sumstat'
sumstat <- summary(cancer$cmr)
# Print a header for the summary table before printing the statistics
cat("Summary table of the crude mortality rate\n")
# Display the summary statistics for CMR
print(sumstat)

# Create a histogram to visualize the distribution of CMR across the dataset
hist(cancer$cmr, breaks = 20, col = "skyblue", main = "Distribution of Crude Mortality Rates",
     xlab = "Crude Mortality Rate", ylab = "Frequency") # Uses 20 bins and colors the bars skyblue

# Assuming 'USMap' is a function (possibly custom or part of a loaded package) for plotting data on a U
# The function call is modified here for clarity but assumes the existence of such a function in the us
USMap(data = cancer$cmr, ncol=10, figmain="US Crude Mortality Rate", lower = 0, upper = 30)
# This plots the CMR data on a map with 10 color breaks, a title, and specified color scale limits


# Step 1: Calculate the 10th and 90th percentile values for crude mortality rates
lowest_10_percentile <- quantile(cancer$cmr, probs = 0.1) # Finds the value at the 10th percentile of C
highest_10_percentile <- quantile(cancer$cmr, probs = 0.9) # Finds the value at the 90th percentile of

# Step 2 & 3: Classify the data and prepare it for mapping
# For the lowest 10%
cancer$lowest_10 <- ifelse(cancer$cmr <= lowest_10_percentile, 1, NA) # Marks counties in the lowest 10

# For the highest 10%
cancer$highest_10 <- ifelse(cancer$cmr >= highest_10_percentile, 1, NA) # Marks counties in the highest

# Plot for the lowest 10%
```

```r
USMap(data = cancer$lowest_10, ncol=2, figmain="Lowest 10% Crude Mortality Rates", lower = 0, upper = 1
# This plots the data for counties in the lowest 10% of CMR on the US map, using 2 color breaks

# Plot for the highest 10%
USMap(data = cancer$highest_10, ncol=2, figmain="Highest 10% Crude Mortality Rates", lower = 0, upper =
# This plots the data for counties in the highest 10% of CMR on the US map, using 2 color breaks

set.seed(05032024)  # Ensures that results are reproducible by starting the random number generator at

# Prepare the data for JAGS model
jags.data <- list(
  y = cancer$dc,   # Number of deaths from cancer, to be used as the response variable in the model
  ni = cancer$pop, # Population sizes, serving as an exposure variable or offset in the model
  N = nrow(cancer) # Total number of observations (or counties) in the dataset
)

# Initial values for the 'theta' parameter are generated for model initialization
# Randomly initialize 'theta' parameters for each county from a gamma distribution
initial_theta <- rgamma(nrow(cancer), 1, 1)

# Define initial values for the JAGS model; using the same initial 'theta' for all chains for simplicit
jags.inits <- list(
  list(theta = initial_theta),
  list(theta = initial_theta),
  list(theta = initial_theta)
)

# Specify the parameters to be monitored (i.e., saved from the model output)
jags.param <- c('mu', 'theta', 'cmr1') # 'mu' for expected counts, 'theta' for rate parameters, 'cmr1'

# Define the JAGS model for analyzing cancer rates
jags.mod.cancer <- function(){
  for (i in 1:N) { # Loop over each observation/unit (e.g., county)
    y[i] ~ dpois(mu[i]) # The observed deaths y[i] follow a Poisson distribution with mean mu[i]
    mu[i] <- 5*ni[i]*theta[i] # Define mu[i] as a function of population size ni[i] and rate parameter
    theta[i] ~ dgamma(20, 430000) # Prior for theta[i] follows a gamma distribution
    cmr1[i] <- (mu[i]/ni[i] * 4) * 10000 # Calculate an adjusted crude mortality rate per 10,000 indivi
  }
}

# Fit the JAGS model to the data
jags.mod.fit <- jags(data = jags.data,                # Data prepared for the model
                     parameters.to.save = jags.param, # Parameters to monitor/save from the model outpu
                     n.chains = 3,                     # Number of MCMC chains to run
                     n.burnin = 5000,                  # Number of burn-in iterations per chain
                     n.iter = 10000,                   # Total number of iterations per chain
                     DIC = FALSE,                      # Whether to compute the Deviance Information Cr
                     inits = jags.inits,               # Initial values for theta
                     model.file = jags.mod.cancer)     # The model definition

# Convert the JAGS model fit to an MCMC object for further analysis
jagsfit.mcmc <- as.mcmc(jags.mod.fit)
```

```r
# Number of plots to select
num_plots <- 5

# Randomly select indices within the range mu[1] to mu[3085]
random_indices <- sample(1:3085, num_plots)

# Trace plot for randomly selected theta parameters
for (i in random_indices) {
  plot(jagsfit.mcmc[, paste0("theta[", i, "]")], start = 1,
       main = paste("Trace Plot for theta[", i, "]"))
  # This loop generates trace plots for 5 randomly selected theta parameters
}

# Convert the summary object to a data frame for easier manipulation
summary_df <- as.data.frame(jags.mod.fit$BUGSoutput$summary)
# Filter rows where Rhat is greater than 1.1, indicating potential convergence issues
high_rhat <- summary_df[summary_df$Rhat > 1.1, ]
# Count the number of parameters with Rhat > 1.1
num_high_rhat <- nrow(high_rhat)
# Print the count of parameters with potential convergence issues
print(paste("Number of parameters with Rhat > 1.1:", num_high_rhat))

cat("Number of entries with Rhat > 1.1:", num_high_rhat, "\n")

ess_values <- effectiveSize(jagsfit.mcmc)
# This calculates the Effective Sample Size (ESS) for each parameter from the MCMC output

# Count the number of ESS values in each specified range
num_above_1000 <- sum(ess_values > 1000) # Counts how many parameters have an ESS greater than 1000
num_between_100_1000 <- sum(ess_values >= 100 & ess_values <= 1000) # Counts parameters with ESS betwee
num_below_100 <- sum(ess_values < 100) # Counts parameters with ESS less than 100

# Print the counts to provide insights into the sampling efficiency
cat("Number of ESS values above 1000:", num_above_1000, "\n")
cat("Number of ESS values between 100 and 1000:", num_between_100_1000, "\n")
cat("Number of ESS values below 100:", num_below_100, "\n")

# Extract and print summary statistics for the first five parameters for a quick inspection
param_summary <- summary(jagsfit.mcmc[, 1:5])
# This part extracts summary statistics for the first five parameters of the model.
# It's useful for a quick check on key parameter estimates, though it doesn't print or display these st

# Print custom message for CMR1 parameters
cat("Summary table of the CMR1 parameters:\n")

# Print statistics for CMR1
print(param_summary$statistics)

# Print quantiles for CMR1
print(param_summary$quantiles)

# Extract and print summary statistics for the mu parameters
param_summary <- summary(jagsfit.mcmc[, 3086:3090]) # Adjust index for mu parameters
```

```r
cat("Summary table of the mu parameters:\n") # Custom message for mu
print(param_summary$statistics) # Print statistics for mu
print(param_summary$quantiles) # Print quantiles for mu

# Extract and print summary statistics for the theta parameters
param_summary <- summary(jagsfit.mcmc[, 6171:6175]) # Adjust index for theta parameters, corrected to 6
cat("Summary table of the theta parameters:\n") # Custom message for theta
print(param_summary$statistics) # Print statistics for theta
print(param_summary$quantiles) # Print quantiles for theta

# Assign mean estimates from JAGS model output to the original cancer dataset for direct comparison
cancer$mu_means <- jags.mod.fit$BUGSoutput$mean$mu      # Mean of the 'mu' parameter for each county
cancer$cmr1_means <- jags.mod.fit$BUGSoutput$mean$cmr1  # Mean of the 'cmr1' parameter for each county
cancer$theta_means <- jags.mod.fit$BUGSoutput$mean$theta # Mean of the 'theta' parameter for each count

# Identify the population thresholds for the top and bottom 1% of counties
top_threshold <- quantile(cancer$pop, probs = 0.99)     # Population threshold for the top 1%
bottom_threshold <- quantile(cancer$pop, probs = 0.01)  # Population threshold for the bottom 1%

# Subset the cancer dataset to include only counties in the top 1% by population
top_1_percent <- subset(cancer, pop >= top_threshold)
# Subset the cancer dataset to include only counties in the bottom 1% by population
bottom_1_percent <- subset(cancer, pop <= bottom_threshold)

# Columns to be excluded from mean calculation (non-numeric or not relevant)
columns_to_exclude <- c("X", "state", "county")

# Calculate means of all relevant columns for the top 1% counties, excluding specified columns
top_1_percent_means <- colMeans(top_1_percent[, !names(top_1_percent) %in% columns_to_exclude], na.rm =

# Calculate means of all relevant columns for the bottom 1% counties, excluding specified columns
bottom_1_percent_means <- colMeans(bottom_1_percent[, !names(bottom_1_percent) %in% columns_to_exclude]

# Print the calculated means for easier comparison
print('bottom 1') # Header for printing bottom 1% means
print(bottom_1_percent_means)
print('top 1')    # Header for printing top 1% means
print(top_1_percent_means)

# Select a random county from those in the top 1% by population
top_1_percent_counties <- subset(cancer, pop >= top_threshold)
selected_top_county <- top_1_percent_counties[sample(nrow(top_1_percent_counties), 1), ]

# Select a random county from those in the bottom 1% by population
bottom_1_percent_counties <- subset(cancer, pop <= bottom_threshold)
selected_bottom_county <- bottom_1_percent_counties[sample(nrow(bottom_1_percent_counties), 1), ]

# Find a county with a population closest to the mean population of all counties
mean_population <- mean(cancer$pop) # Calculate mean population
nearest_mean_counties <- abs(cancer$pop - mean_population) # Calculate absolute difference from mean fo
selected_mean_county <- cancer[which.min(nearest_mean_counties), ] # Select county with smallest differ

# Print information about the selected counties
```

```
print(paste("County with top 1% population:", selected_top_county$county, "(", selected_top_county$stat
print(paste("County with bottom 1% population:", selected_bottom_county$county, "(", selected_bottom_cou
print(paste("County near the mean population:", selected_mean_county$county, "(", selected_mean_county$

# Prepare data for modeling: update population values for a scenario analysis
# Combine county and state names for the selected counties
selected_counties <- c(
  paste(selected_top_county$county, selected_top_county$state),
  paste(selected_bottom_county$county, selected_bottom_county$state),
  paste(selected_mean_county$county, selected_mean_county$state)
)
# Subset original data for selected counties
selected_data <- subset(cancer, paste(county, state) %in% selected_counties)
# Manually update population figures for the selected counties (hypothetical or for scenario analysis)
selected_data$newpop <- c(94043, 4497000, 797) # New population values for each selected county

# Preview the updated dataset for the selected counties
selected_data

# Update the data list for JAGS, including the new population figures
jags.data <- list(
  y = cancer$dc, # Deaths from cancer
  ni = cancer$pop, # Original population sizes
  N = nrow(cancer), # Total number of counties
  new_ni = selected_data$newpop, # Updated population sizes for the selected counties
  new_N = length(selected_data$newpop) # Number of updated population entries
)

jags.mod.cancer <- function(){
  for (i in 1:N) { # Loop through each observation
    y[i] ~ dpois(mu[i]) # Observations follow a Poisson distribution
    mu[i] <- 5*ni[i]*theta[i] # Expected value is a product of population, rate parameter, and a factor
    theta[i] ~ dgamma(20, 430000) # Prior for the rate parameter follows a Gamma distribution
    cmr1[i] <- (mu[i]/ni[i] * 4) * 10000 # Calculate crude mortality rate per 10,000 population
  }

  # Predictions for new observations using updated population figures
  for (i in 1:new_N) { # Loop through new or hypothetical observations
    mu_new[i] <- 5*new_ni[i]*theta[i] # Calculate expected deaths for new observations
    y_new[i] ~ dpois(mu_new[i]) # Model these expected deaths with a Poisson distribution
  }
}


jags.param <- c('mu_new', 'y_new') # Specify parameters of interest to monitor: new expected deaths and

# Random initialization for theta parameters using a gamma distribution
initial_theta <- rgamma(nrow(cancer), 1, 1)  # Generate initial values for theta from a gamma distribut

# Initialization list for the JAGS model
jags.inits <- list(
  list(theta = initial_theta), # First chain initial values
  list(theta = initial_theta), # Second chain initial values
  list(theta = initial_theta)  # Third chain initial values
)

# Execute the JAGS model
jags.mod.fit <- jags(data = jags.data,                    # Data prepared for JAGS
                     parameters.to.save = jags.param,    # Parameters to save from the JAGS output
                     n.chains = 3,                       # Number of MCMC chains
                     n.burnin = 5000,                    # Number of burn-in iterations to discard
                     n.iter = 10000,                     # Total number of iterations for the MCMC chai
                     DIC = FALSE,                        # Do not calculate DIC (Deviance Information C
                     inits = jags.inits,                 # Initial values for the chains
                     model.file = jags.mod.cancer)       # JAGS model function

# Convert JAGS model output to 'mcmc' class for further analysis
jagsfit.mcmc <- as.mcmc(jags.mod.fit) # Convert the fit to 'mcmc' object for compatibility with 'coda' 

# Extract the y_new samples from the MCMC object
y_new_samples <- as.matrix(jagsfit.mcmc[, , "y_new"])

# Plot histograms of the predictive distributions
par(mfrow = c(1, 3))  # Arrange plots in one row with three columns
for (i in 1:3) {
  hist(y_new_samples[, i], main = paste("County", i), xlab = "Predicted Deaths", col = "skyblue", borde
}

# Calculate numerical summaries
summary(y_new_samples)

# Calculate the mean of the predictive distributions for each county
mean_deaths <- apply(y_new_samples, 2, mean)

# Create a new dataframe to store means of y_new
means_df <- data.frame(
  mu_new = mean_deaths[1:3],  # Mean values of mu_new
  y_new = mean_deaths[4:6]    # Mean values of y_new
)


selected_data$newdc <- means_df$y_new
selected_data$newmu <- means_df$mu_new

for (i in 1:nrow(selected_data)) {
  cat("The estimate of deaths for", selected_data$state[i], ",", selected_data$county[i],
      "is", means_df$y_new[i], "with the current population of", selected_data$newpop[i], "\n")
}

observed_deaths <- selected_data$dc # Store observed deaths from the selected data

# Update the jags.data list to include both the new population values and observed deaths for the new a
jags.data <- list(
  y = cancer$dc, # Current observed deaths across the dataset
  ni = cancer$pop, # Original population sizes for each observation
  N = nrow(cancer), # Total number of observations in the dataset
  new_ni = selected_data$newpop, # Updated population sizes for selected counties
  new_N = length(selected_data$newpop), # Number of updated population entries
  observed_deaths = observed_deaths # Adding observed deaths for selected counties into the data list f
)

jags.mod.cancer <- function(){
  for (i in 1:N) { # Iterate over each original observation
    y[i] ~ dpois(mu[i]) # Model observed deaths as Poisson with mean mu[i]
    mu[i] <- 5*ni[i]*theta[i] # Define expected deaths mu[i] based on population and a rate parameter
    theta[i] ~ dgamma(20, 430000) # Rate parameter theta[i] follows a Gamma distribution
    cmr1[i] <- (mu[i]/ni[i] * 4) * 10000 # Calculate adjusted crude mortality rate per 10,000
  }

  # Predictions for new observations based on updated population
  for (i in 1:new_N) {
    mu_new[i] <- 5*new_ni[i]*theta[i] # Expected deaths for new observations
    y_new[i] ~ dpois(mu_new[i]) # Model these as Poisson
  }

  # Define prediction criteria based on specific thresholds
  pred.crit1 <- ifelse(y_new[1]>4,1,0) # Criterion for the first new observation
  pred.crit2 <- ifelse(y_new[2]>172,1,0) # Criterion for the second new observation
  pred.crit3 <- ifelse(y_new[3]>0,1,0) # Criterion for the third new observation
}

# Define parameters to save from the JAGS model output
jags.param <- c('pred.crit1','pred.crit2','pred.crit3')

# Initialize theta parameter values randomly from a gamma distribution
initial_theta <- rgamma(nrow(cancer), 1, 1)
jags.inits <- list(
  list(theta = initial_theta),
  list(theta = initial_theta),
  list(theta = initial_theta)
)

# Run the JAGS model to predict criteria based on new observations
jags.mod.fit <- jags(data = jags.data,
                     parameters.to.save = jags.param,
                     n.chains = 3,
                     n.burnin = 5000,
                     n.iter = 10000,
                     DIC = FALSE,
                     inits = jags.inits,
                     model.file = jags.mod.cancer)

# Convert the JAGS model fit to an MCMC object
jagsfit.mcmc <- as.mcmc(jags.mod.fit)

# Extract simulation results for pred.crit1
prob1 <- jags.mod.fit$BUGSoutput$sims.list$pred.crit1
prob1_numeric <- as.numeric(prob1)
counts1 <- table(prob1_numeric)
```

```r
# Extract simulation results for pred.crit2
prob2 <- jags.mod.fit$BUGSoutput$sims.list$pred.crit2
prob2_numeric <- as.numeric(prob2)
counts2 <- table(prob2_numeric)

# Extract simulation results for pred.crit3
prob3 <- jags.mod.fit$BUGSoutput$sims.list$pred.crit3
prob3_numeric <- as.numeric(prob3)
counts3 <- table(prob3_numeric)

# Calculate total number of simulations
total_simulations <- length(prob1_numeric)

# Calculate probabilities for pred.crit1
new_probabilities1 <- counts1 / total_simulations

# Calculate probabilities for pred.crit2
new_probabilities2 <- counts2 / total_simulations

# Calculate probabilities for pred.crit3
new_probabilities3 <- counts3 / total_simulations

# Print probabilities for each criterion
cat('Probability of predicted death count exceeding death count from 1980-1984 in Lauderdale:\n')
print(new_probabilities1)
cat('Probability of predicted death count exceeding death count from 1980-1984 in Maricopa:\n')
print(new_probabilities2)
cat('Probability of predicted death count exceeding death count from 1980-1984 in Roberts:\n')
print(new_probabilities3)

#############
# section b #
#############

# Read the dataset from the specified path
df <- read.csv('C:\\Users\\archi\\Desktop\\ADSAS\\advanced topics\\classification.csv')

# Remove the first column, assumed to be an unnecessary index column, to clean the data
df <- df[, -1]

# Convert the target variable 'y' to a factor for classification tasks
df$y <- as.factor(df$y)

# Calculate the total count of missing values across all columns
missing_values <- sum(is.na(df))
missing_values


# Print summary statistics of the dataframe
print("Summary of the Classification data")
summary(df)

# Use glimpse from the dplyr package to get a structured overview of the dataframe
```

```r
glimpse(df)

# Create scatter plots to visualize the data distribution for different classes

# Scatter plot for Class A vs. 'not A'
ggplot(df, aes(x = x1, y = x2, color = (y == 'A'))) +
  geom_point() +
  labs(title = "Problem A: Class A vs. 'not A'", x = "X1", y = "X2", caption = 'Figure ?') +
  scale_color_manual(values = c("TRUE" = "red", "FALSE" = "blue"), name = "Class A")

# Scatter plot for Class B vs. 'not B'
ggplot(df, aes(x = x1, y = x2, color = (y == 'B'))) +
  geom_point() +
  labs(title = "Problem B: Class B vs. 'not B'", x = "X1", y = "X2", caption = 'Figure ?') +
  scale_color_manual(values = c("TRUE" = "green", "FALSE" = "purple"), name = "Class B")

# Scatter plot for Class A, B, and C
ggplot(df, aes(x = x1, y = x2, color = y)) +
  geom_point() +
  labs(title = "Problem C: Class A vs. B vs. C", x = "X1", y = "X2", caption = 'Figure ?')

set.seed(12032024) # For reproducibility

# Prepare the dataset for binary classification by creating binary target variables
df$ya <- ifelse(df$y == 'A', 1, 0)  # Create binary variable for class A
df$yb <- ifelse(df$y == 'B', 1, 0)  # Create binary variable for class B

# Split the dataset into training and testing sets
split <- initial_split(df, prop = 0.75)

# Extract the training and testing sets from the split
trainData <- training(split)
testData <- testing(split)

# Logistic Regression

# Generate random values for alpha hyperparameter (elastic net mixing)
alpha_values <- runif(50, min = 0, max = 1)

# Generate random values for lambda hyperparameter (regularization strength)
lambda_values <- exp(runif(50, min = log(1e-6), max = log(1)))

# Initialize a dataframe to store results of the random search
results <- data.frame(alpha = numeric(), lambda = numeric(), accuracy = numeric())

# Loop over each combination of hyperparameters
for(i in 1:length(alpha_values)) {
  # Fit a logistic regression model using glmnet
  fit <- glmnet(as.matrix(trainData[,c("x1", "x2")]), trainData$ya,
              family = "binomial", alpha = alpha_values[i], lambda = lambda_values[i])

  # Predict probabilities on the test set
  predictions_prob <- predict(fit, newx = as.matrix(testData[,c("x1", "x2")]), type = "response")
  # Convert probabilities to binary classes with a threshold of 0.5
  predictedClasses <- ifelse(predictions_prob > 0.5, 1, 0)

  # Create a confusion matrix from predictions and actual values
  confMat <- table(predictedClasses, testData$ya)

  # Calculate accuracy as the proportion of correct predictions
  accuracy <- sum(diag(confMat)) / sum(confMat)

  # Store the current combination of hyperparameters and its accuracy
  results[i, ] <- c(alpha_values[i], lambda_values[i], accuracy)
}

# Find the best performing combination of hyperparameters based on accuracy
best_params <- results[which.max(results$accuracy), ]

# Retrain the model on the full training dataset using the best hyperparameters
final_fit <- glmnet(as.matrix(trainData[,c("x1", "x2")]), trainData$ya,
                    family = "binomial", alpha = best_params$alpha, lambda = best_params$lambda)

# Make final predictions on the test set
predictions_prob <- predict(final_fit, newx = as.matrix(testData[,c("x1", "x2")]), type = "response")
# Convert these probabilities to binary classes
predictedClasses <- ifelse(predictions_prob > 0.5, 1, 0)

# Create a confusion matrix for the final predictions
confMat <- table(Predicted = predictedClasses, Actual = testData$ya)
print("Confusion Matrix for Logistic Regression:")
print(confMat)

# Calculate and print the accuracy of the final model
accuracy <- sum(diag(confMat)) / sum(confMat)
print(paste("Accuracy for Logistic Regression:", accuracy))


# LDA

# Train an LDA model using the 'lda' function from the MASS package.
ldaModel <- lda(ya ~ x1 + x2, data = trainData)

# Make predictions on the test dataset 'testData' using the trained LDA model.
predictionsLDA <- predict(ldaModel, newdata = testData)

# Extract the class predictions from the list
predictedClassesLDA <- predictionsLDA$class

# Create a confusion matrix to evaluate the model's predictions.
confMatLDA <- table(predictedClassesLDA, testData$ya)
print("Confusion Matrix for LDA:")
print(confMatLDA)

# Calculate the model's accuracy
accuracyLDA <- sum(diag(confMatLDA)) / sum(confMatLDA)
```

```r
print(paste("Accuracy for LDA:", accuracyLDA))

# Normalize data using min-max scaling
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

# Apply normalization to the 'x1' and 'x2' columns of training and testing data
trainDataNormalized <- as.data.frame(lapply(trainData[, c('x1', 'x2')], normalize))
testDataNormalized <- as.data.frame(lapply(testData[, c('x1', 'x2')], normalize))

# Add the binary target variable back into the normalized data frames
trainDataNormalized$yb <- trainData$yb
testDataNormalized$yb <- testData$yb

# Separate features and labels for training and testing sets
train_features <- trainDataNormalized[, c('x1', 'x2')]
test_features <- testDataNormalized[, c('x1', 'x2')]
train_labels <- trainDataNormalized$yb
test_labels <- testDataNormalized$yb

# Define a sequence of odd k values from 1 to 20 for KNN parameter testing
k_values <- seq(1, 20, by = 2)

# Initialize an empty list to store confusion matrices for different k values
confMatrices <- list()

# Initialize an empty dataframe to store results (k values and corresponding accuracies)
results <- data.frame(k = integer(), accuracy = numeric())

# Loop through each k value to perform KNN and evaluate performance
for(k in k_values) {
  # Perform KNN classification with current k, using training features and labels
  predictedClassesKNN <- knn(train = train_features, test = test_features, cl = train_labels, k = k)
  # Create a confusion matrix for the predictions
  confMatKNN <- table(Predicted = predictedClassesKNN, Actual = test_labels)

  # Store the confusion matrix in the list with k as the key
  confMatrices[[as.character(k)]] <- confMatKNN

  # Calculate accuracy from the confusion matrix and add to results dataframe
  accuracy <- sum(diag(confMatKNN)) / sum(confMatKNN)
  results <- rbind(results, data.frame(k = k, accuracy = accuracy))
}

# Determine the best k value based on highest accuracy
best_k <- results[which.max(results$accuracy), 'k']

# Print the highest accuracy achieved
best_accuracy <- max(results$accuracy)
print(paste("Accuracy for KNN:", best_accuracy))

# Retrieve and print the confusion matrix for the best k value
```

```r
best_confMat <- confMatrices[[as.character(best_k)]]
print(paste("Confusion Matrix for the best k =", best_k))
print(best_confMat)

# Support Vector Machine

# Convert the 'yb' column in both training and testing data
trainData$yb <- as.factor(trainData$yb)
testData$yb <- as.factor(testData$yb)

# Train an SVM model using the radial basis function kernel.
svmModel <- svm(yb ~ ., data = trainData, kernel = "radial", cost = 1, gamma = 0.1)

# Use the trained SVM model to predict classes on the test dataset.
predictedClassesSVM <- predict(svmModel, newdata = testData)

# Create a confusion matrix
confMatSVM <- table(Predicted = predictedClassesSVM, Actual = testData$yb)
print("Confusion Matrix for SVM:")
print(confMatSVM)

# Calculate the accuracy of the model
accuracy <- sum(diag(confMatSVM)) / sum(confMatSVM)
print(paste("Accuracy for SVM:", accuracy))

# Train a QDA model using 'qda' function.
qdaModel <- qda(y ~ x1 + x2, data = trainData)

# Make predictions on the 'testData' dataset
predictedClassesQDA <- predict(qdaModel, newdata = testData)$class

# Create a confusion matrix
confMatQDA <- table(Predicted = predictedClassesQDA, Actual = testData$y)
print("Confusion Matrix for QDA:")
# Print the confusion matrix
print(confMatQDA)

# Calculate the accuracy of the model
accuracyQDA <- sum(diag(confMatQDA)) / sum(confMatQDA)

# Print the calculated accuracy to assess the model's performance.
print(paste("Accuracy for QDA:", accuracyQDA))

# Create a scatter plot for C vs not C
ggplot(df, aes(x = x1, y = x2, color = (y == 'C'))) +
  geom_point() +  # Add points to the plot for each observation
  labs(title = "Problem A: Class C vs. 'not C'", x = "X1", y = "X2", caption = 'Figure ?') +  # Add cus
  scale_color_manual(values = c("TRUE" = "blue", "FALSE" = "red"), name = "Class C")
```