MTHM017 - Advanced Topics in Statistics

Joshua Harrison

2025-03-13

AI-supported/AI-integrated use is permitted in this assessment. I acknowledge the following uses of GenAI tools in this assessment:

- I have used GenAI tools to proofread and correct grammar or spelling errors.
- I have used GenAI to help debug my code.

I declare that I have referenced use of GenAI outputs within my assessment in line with the University referencing guidelines.

A. Bayesian Inference

1.

We will fit a finite model using the `rtimes` dataset, which contains the reaction times of 17 people (11 non-schizophrenics and 6 schizophrenics, stored in this order) in a psychological experiment. Each person's reaction time was measured 30 times.

We convert the data into long format so that each row is representing a single observation. Reaction Time is shown by one column as a list which makes mapping the histograms much easier using `facet_wrap()` in `ggplot2` using a consistent x-axis range.

```r
# Renaming the first column to from "X" to Person"
colnames(rtimes)[1] <- "Person"

# Converting the data into long format as explained above
rtimes_long <- pivot_longer(rtimes,
                            cols = starts_with("T"),
                            names_to = "Trial",
                            values_to = "ReactionTime")

head(rtimes_long)
```

```
## # A tibble: 6 x 3
##   Person Trial ReactionTime
##    <int> <chr>        <int>
## 1      1 T1             312
## 2      1 T2             272
## 3      1 T3             350
## 4      1 T4             286
## 5      1 T5             268
## 6      1 T6             328
```

```r
#Adding a group column that allows us to differentiate between non-schizophrenics and schizophrenics
rtimes_long <- rtimes_long %>%
  mutate(Group = ifelse(Person <= 11, "Non-Schizophrenic", "Schizophrenic"))
```

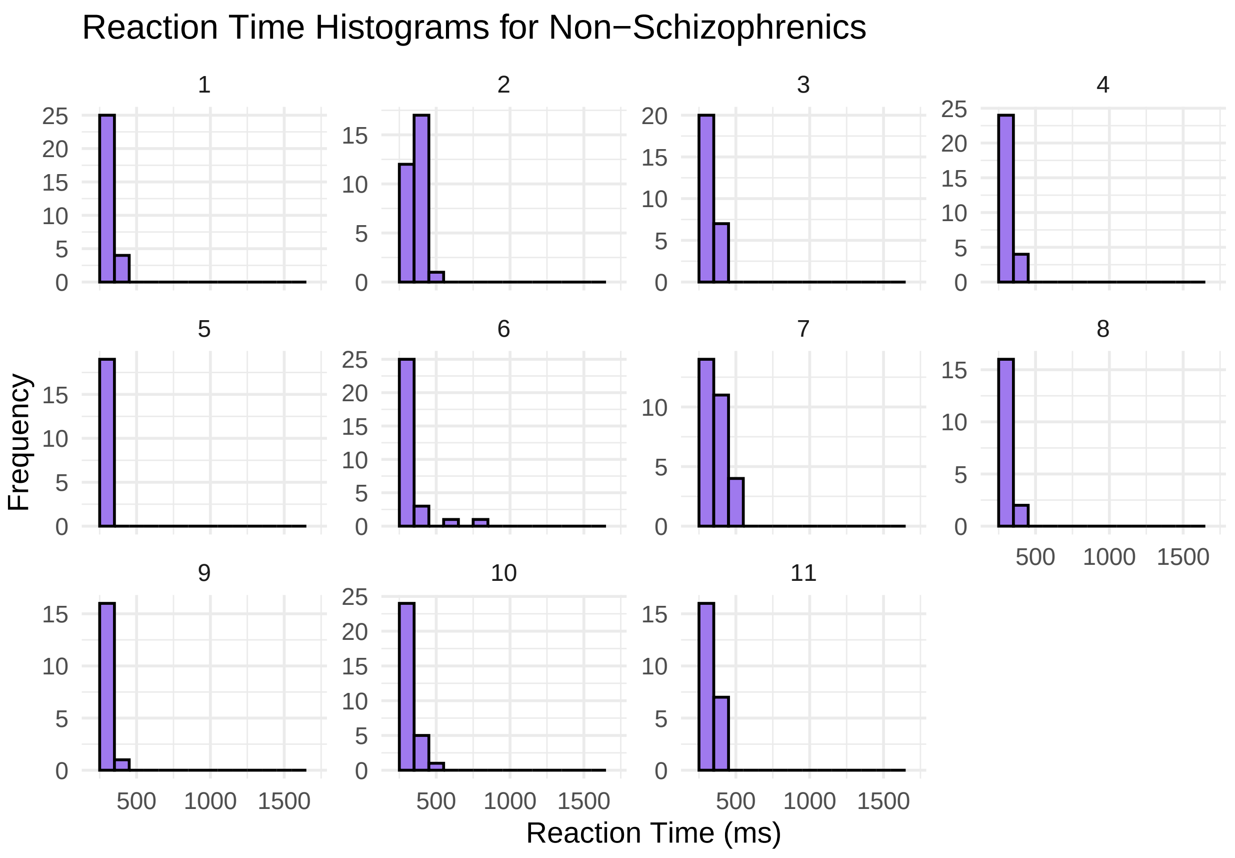
```r
#Checking the range of the data values in the x axis in order to create a conssitent x-axis range
range(rtimes_long$ReactionTime, na.rm = TRUE)
```

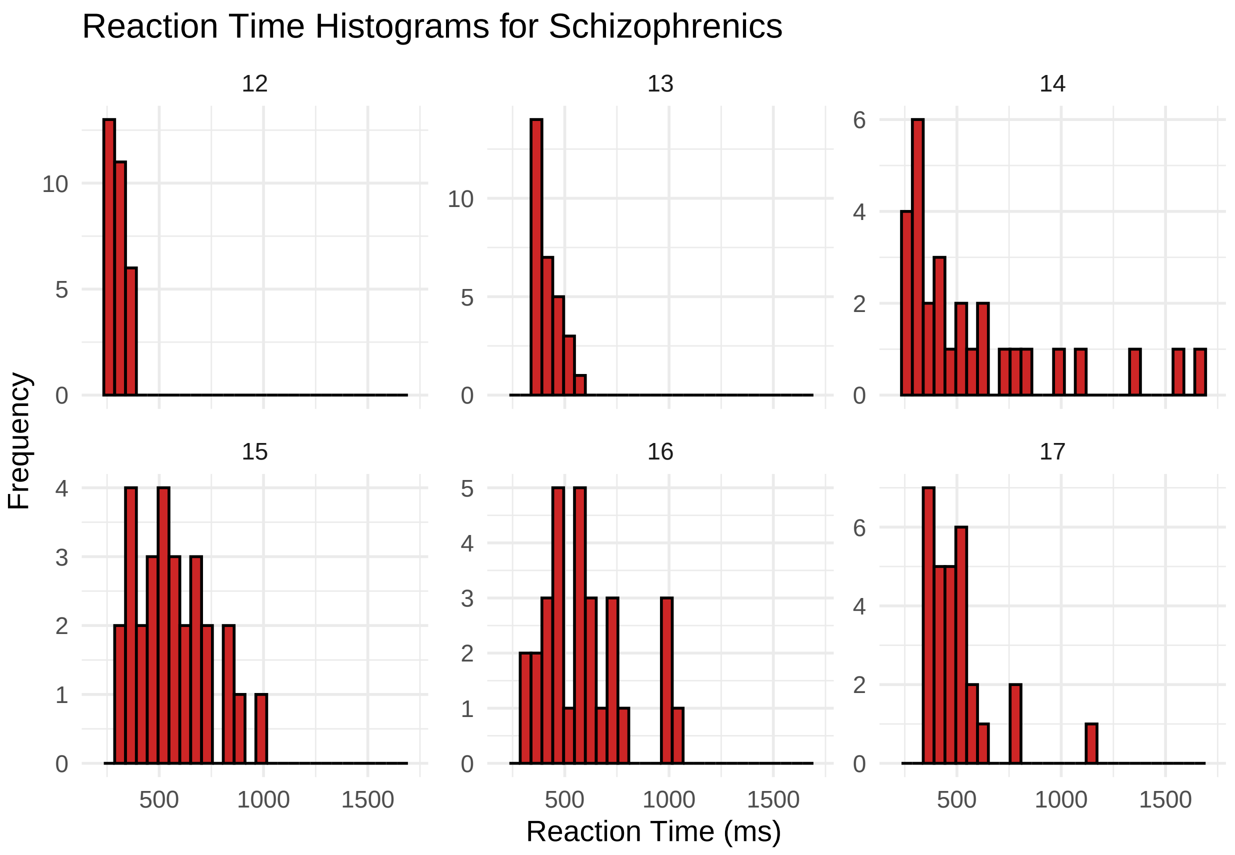
```
## [1]  204 1714
```

This prints out the x min and x max values for the reaction times in the dataset. We can store them below so that we can create a consistent x axis for the histograms.

```r
x_min <- 204
x_max <- 1714
```

```r
# Plotting histograms for Non-Schizophrenics

histogram_nonschizophrenic <- ggplot(filter(rtimes_long, Group == "Non-Schizophrenic"),
                                     aes(x = ReactionTime)) +
  geom_histogram(binwidth = 100, color = "black", fill = "mediumpurple2") +
  facet_wrap(~ Person, scales = "free_y") +
  xlim(x_min, x_max) + # Ensures the consistent x-axis range
  theme_minimal() +
  labs(title = "Reaction Time Histograms for Non-Schizophrenics",
       x = "Reaction Time (ms)",
       y = "Frequency")
  invisible(theme(panel.spacing = unit(0.5, "lines")))

#Plotting histograms for Schizophrenics

histogram_schizophrenic <- ggplot(filter(rtimes_long, Group == "Schizophrenic"),
                                 aes(x = ReactionTime)) +
  geom_histogram(binwdith = 100, color = "black", fill = "firebrick3") +
  facet_wrap(~ Person, scales = "free_y") +
  xlim(x_min, x_max) +
  theme_minimal() +
  labs(title = "Reaction Time Histograms for Schizophrenics",
       x = "Reaction Time (ms)",
       y = "Frequency") +
  invisible(theme(panel.spacing = unit(0.5, "lines")))
```


Reaction Time Histograms for Non−Schizophrenics
1
2
3
4
5
6
7
8
9
10
11
Frequency
Reaction Time (ms)
500
1000
1500


Reaction Time Histograms for Schizophrenics
12
13
14
15
16
17
Frequency
Reaction Time (ms)
500
1000
1500

As we can see from the histograms above, the non schizophrenic individuals (people 1-11) show highly right-skewed data, with most data points concentrated on the left (lower reaction times). Almost all non-schizophrenic individuals have reaction times below 500ms. However, Person 6 and Person 10 show anomalies with a few readings over 500ms. Despite this, there is a consistent pattern and fast reaction times for non-schizophrenic people with minimal variability.

In contrast, the schizophrenic individuals (people 12-17) display inconsistent reaction times and patterns. There is some consistency across certain individuals such as Person 12, 13 and 17 who display mostly right-skewed results much alike the non-schizophrenic individuals. However, the variability between schizophrenic individuals is much greater than the non-schizophrenic individuals. Person 14, 15 and 16 show higher frequency of these extreme reaction times. Person 14 in particular shows a wider distribution with many results over 1000ms with multiple peaks, possibly due to fluctuating attention or motor control issues. The schizophrenic individuals display much more extreme values and inconsistent reaction times than the non-schizophrenic people, with multiple modes and longer tails.

The peak modes for both schizophrenic and non-schizophrenic groups are towards the lower reaction times, although they are much sharper for the non-schizophrenic groups. Both groups demonstrate right-skewed distributions, however the skewness is more pronounced for the non-schizophrenic individuals. The schizophrenic group display multimodal, flatter distributions with a broader range and more outliers compared to the non-schizophrenics people.

Specifically comparing Person 5 (non-schizophrenic) to Person 14 (schizophrenic), Person 5 has a sharp peak at lower values, whereas Person 14 displays a much wider spread of reaction times with multiple peaks. Comparing Person 10 (non-schizophrenic) to Person 16 (schizophrenic), we can see that Person 10 has a tight consistent distribution, while Person 16 shows a bimodal distribution with multiple peaks and inconsistent reaction times. This corroborates the claim that non-schizophrenic individuals demonstrate faster, more consistent reaction times with limited variability, indicating efficient cognitive processing. In contrast, schizophrenic individuals show inconsistent patterns and greater variation, reflecting potential attention deficit and cognitive challenges such as motor reflex retardation.

2.

To reflect the attention deficit, let y_{ij} denote the logarithm of the j-th measured reaction time of person i.

Then:

Non-Schizophrenic Individuals For the responses of the i-th non-schizophrenic person (i = *1, 2, . . . , 11*) we have:

$$y_{ij} \sim N(\alpha_i, \sigma_y^2), \quad i = 1, 2, ..., 11, \; j = 1, 2, ..., 30.$$

That is, the responses are normally distributed with person-specific mean α_i and a common variance σ_y^2.

Schizophrenic Individuals For the responses of the i-th schizophrenic individual (i = *12, 13, . . . , 17*), there are two possibilities: - With probability $(1 - \lambda)$, there is no delay, and the response is normally distributed with mean α_i and variance σ_y^2. - With probability λ, the response is delayed, and the observations have mean $\alpha_i + \tau$ and variance σ_y^2.

This can be modeled as:

$$y_{ij} \sim N(\alpha_i + \tau z_{ij}, \sigma_y^2),$$

where

$$z_{ij} \sim \text{Bernoulli}(\lambda), \quad i = 12, 13, ..., 17; \; j = 1, 2, ..., 30.$$

- $z_{ij} = 0$ with probability $(1 - \lambda)$ (no delay)
- $z_{ij} = 1$ with probability λ (delay occurs)

The magnitude of the schizophrenics' motor retardation is captured by the distribution of the α_i parameters. In particular:

Non-Schizophrenic Individuals For non-schizophrenic individuals, we assume that α_i follows a normal distribution with mean μ and variance σ_α^2. Specifically,

$$\alpha_i \sim N(\mu, \sigma_\alpha^2), \quad i = 1, 2, \ldots, 11.$$

Schizophrenic Individuals For the schizophrenics, we assume that the mean of α_i is $\mu + \beta$, while the variance remains σ_α^2. Specifically,

$$\alpha_i \sim N(\mu + \beta, \sigma_\alpha^2), \quad i = 12, 13, \ldots, 17.$$

The above model uses the logarithm of measured reaction times. This is relevant since the data shown in the histograms above is predominately right-skewed data. Particularly for the schizophrenic people, the data is highly skewed with long tails. Therefore, taking the logarithm will reduce the skewness, allowing the distribution of results to become more symmetric and closer to normality. This is particularly important as we are modelling using a normal distribution which assumes symmetry. As well as this, the variance is stabilised, reducing heteroscedasticity. This is important for Bayesian models which assume both normality and constant variance.

```r
rtimes_long <- rtimes_long %>%
  mutate(LogReactionTime = log(ReactionTime))


head(rtimes_long)
```

```
## # A tibble: 6 x 5
##   Person Trial ReactionTime Group             LogReactionTime
##    <int> <chr>        <int> <chr>                       <dbl>
## 1      1 T1             312 Non-Schizophrenic            5.74
## 2      1 T2             272 Non-Schizophrenic            5.61
## 3      1 T3             350 Non-Schizophrenic            5.86
## 4      1 T4             286 Non-Schizophrenic            5.66
## 5      1 T5             268 Non-Schizophrenic            5.59
## 6      1 T6             328 Non-Schizophrenic            5.79
```

```r
log_sd <- rtimes_long %>%
  group_by(Person) %>%
  summarise(SD_LogReactionTime = sd(LogReactionTime, na.rm = TRUE))

head(log_sd)
```

```
## # A tibble: 6 x 2
##   Person SD_LogReactionTime
##    <int>              <dbl>
## 1      1              0.127
## 2      2              0.129
## 3      3              0.169
## 4      4              0.150
## 5      5              0.145
## 6      6              0.224
```

3.

Model Parameters

When choosing our non-informative uniform priors, it is worth noting that we are choosing bounds for the logged values of the dataset. The parameters used within this model are:

$\boldsymbol{\mu}$ - The mean reaction times for non-schizophrenics. This can take any real value: $-\infty < \mu < +\infty$. JAGS does not support infinite bounds, so we will use a wide range in the context of the logged reaction times.

- mu ~ dunif(-10, 10)

  $\boldsymbol{\sigma_y^2}$ - The variance of reaction times. This value must be positive. JAGS uses precision:

$$\tau = \frac{1}{\sigma_y^2}$$

- sigma_y2 ~ dunif(0, 10)
- tau_y <- 1/ sigma_y2 (Precision derived from variance for JAGS implementation)

$\boldsymbol{\alpha_i}$ - The person specific mean. This can take any real value: $-\infty < \mu < +\infty$.

- alpha[i] ~ dunif(-10, 10)

$\boldsymbol{\sigma_\alpha^2}$ - This represents the variance of the person-specific means shown above. We convert to precision for JAGS.

- sigma_alpha2 ~ dunif(0, 5)

- p.alpha <- 1 / sigma_alpha2

$\boldsymbol{\lambda}$ - Probability of delay for schizophrenics. The probability must be between 0 and 1.

- lambda ~ dunif(0, 1)

$\boldsymbol{\tau}$ - Delay magnitude for response of schizophrenics. This is restricted to be positive to ensure the model is identifiable.

- tau ~ dunif(0, 10)

$\boldsymbol{\beta}$ - Difference in mean for schizophrenics. This can take any value any real value $-\infty < \mu < +\infty$.

- beta ~ dunif(-10, 10)

4.

Fitting JAGS Model

```r
# Data list for jags

extractedRtimes <- rtimes [, 2:31]
log_rtimes <- log(extractedRtimes)

jags.data <- list(
  y.ns = log_rtimes[1:11, ],
  y.s = log_rtimes[12:17, ]
)
```

```r
set.seed(123)

jags.model <- function(){
  # Reaction time of non-schizophrenics
  for(i in 1:11){
    for(j in 1:30){
      y.ns[i, j] ~ dnorm(alpha.ns[i], p.y)
    }
    alpha.ns[i] ~ dnorm(mu, p.alpha)
}
# Reaction time of schizophrenics
  for(i in 1:6){
    for(j in 1:30){
      z[i, j] ~ dbern(lambda)
      y.s[i, j] ~ dnorm(alpha.s[i] + tau * z[i, j], p.y)
      }
      alpha.s[i] ~ dnorm(mu + beta, p.alpha)
  }


# Priors

mu ~ dunif(-10, 10)
beta ~ dunif(0, 10)
tau ~ dunif(0, 10)
lambda ~ dunif(0, 1)
sigma_y2 ~ dunif(0, 10)
p.y <- 1/ sigma_y2 # converting variance to precision as mentioned above.
sigma_alpha2 ~ dunif(0, 5)
p.alpha <- 1 / sigma_alpha2 #converting variance to precision.

}

# Parameters we want to monitor

jags.param <- c("mu", "beta", "tau", "lambda", "sigma_y2", "sigma_alpha2")
```

Its important to choose initial values to reflect the observed distribution of reaction times and avoid extreme values that slow down the MCMC sampling. This will help JAGS start close to true values and improve convergence.

In order to choose a reasonable initial value for the mean of the non-schizophrenics ($\boldsymbol{\mu}$), we will compute the mean reaction time of non-schizophrenic individuals using `rnorm`. We set a seed beforehand to ensure reproducibility. This therefore makes sure that mu start at a central value.

```
set.seed(123)  # Ensures reproducibility

rnorm(1, mean = 6, sd = 1)  # Mean reaction time (log-scale)
```

```
## [1] 5.439524
```

For the difference in mean for schizophrenics ($\boldsymbol{\beta}$), we compute the difference between the means for schizophrenic and non-schizophrenic individuals.

```
# Compute mean log reaction times for each group
mu_ns <- mean(rtimes_long$LogReactionTime[rtimes_long$Person <= 11], na.rm = TRUE)
mu_s <- mean(rtimes_long$LogReactionTime[rtimes_long$Person >= 12], na.rm = TRUE)

# Compute the difference (beta)
```

```r
mu_s - mu_ns
```

```
## [1] 0.4165507
```

For delay magnitude for response of schizophrenics ($\boldsymbol{\tau}$), we choose to set tau based on observation from the histograms. If we calculate the difference between the 75th percentile of reaction times and subtract this from the 50th percentile representing a typical response time, we estimate the delay.

```r
quantile(rtimes_long$LogReactionTime[rtimes_long$Person >= 12], 0.75, na.rm = TRUE)[[1]] -
  quantile(rtimes_long$LogReactionTime[rtimes_long$Person >= 12], 0.50, na.rm = TRUE)[[1]]
```

```
## [1] 0.2710224
```

For the probability delay for schizophrenics ($\boldsymbol{\lambda}$), looking at the histograms and the proportion of delayed response, we will assume that roughly 30% of schizophrenic responses are delayed.

For the variance of the reaction times ($\boldsymbol{\sigma_y^2}$), we will estimate the variance in the log-transformed reaction times.

```r
var(log(rtimes_long$ReactionTime), na.rm = TRUE)
```

```
## [1] 0.1217484
```

For the variance of the person specific means ($\boldsymbol{\sigma_\alpha^2}$), this refers to how much the mean reactions varies across individuals. We choose a value larger than $\boldsymbol{\sigma_y^2}$ because individuals differ more than trial to trial variance.

```r
inits1 <- list(
  mu = 5.44, beta = 0.42, tau = 0.27, lambda = 0.3, sigma_y2 = 0.2,  sigma_alpha2 = 0.2
)

# The initial values for the parameters for chain 2 have been selected randomly.

inits2 <- list(
  mu = 6.5, beta = 0.7, tau = 0.3, lambda = 0.6, sigma_y2 = 0.35, sigma_alpha2 = 0.4
)

jags.inits <- list(inits1, inits2)
```

```r
jags.mod.fit <- jags(data = jags.data, inits = jags.inits,
                     parameters.to.save = jags.param, n.chains = 2, n.iter = 10000,
                     n.burnin = 5000, n.thin=1, model.file = jags.model, DIC = FALSE)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 510
##    Unobserved stochastic nodes: 203
##    Total graph size: 1081
##
## Initializing model
```

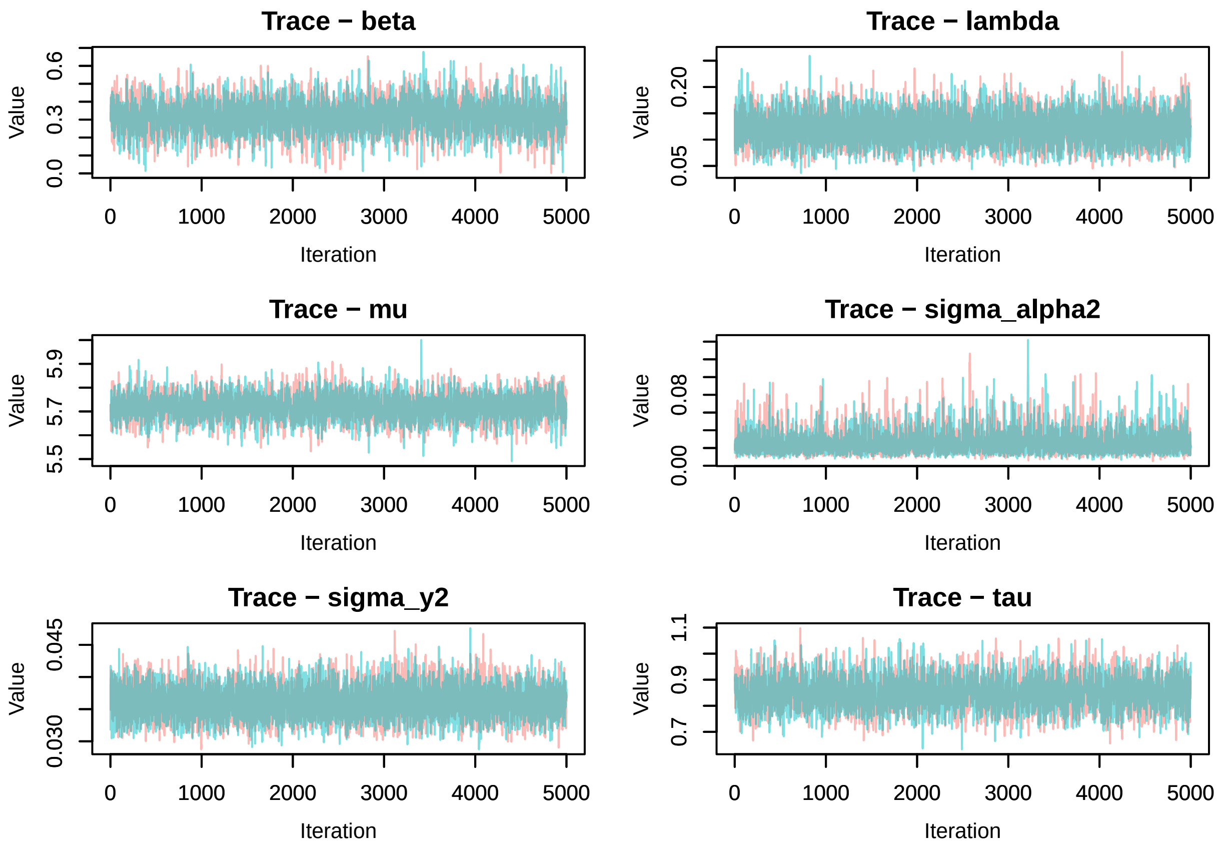
```r
summary(jags.mod.fit)
```

```
##                    Length Class  Mode
## model               8     jags   list
## BUGSoutput         22     bugs   list
## parameters.to.save  6     -none- character
## model.file          1     -none- character
## n.iter                1      -none- numeric
## DIC                   1      -none- logical
```

5.

Markov Chain Monte Carlo

```r
jagsfit.mcmc <- as.mcmc(jags.mod.fit)

MCMCtrace(jagsfit.mcmc, type = 'trace', ind = TRUE, pdf = FALSE)
```



In the traceplots above, we can see good mixing between the chains. The chains look like a random scatter around a stable value and the behavior of the two chains are both indistinguishable. All the chains have a caterpillar shape, suggesting convergence.

```r
# Computing Gelman diagnostics
gelman_results <- gelman.diag(jagsfit.mcmc)$psrf

# Converting to dataframe
gelman_df <- as.data.frame(gelman_results) %>%
  rownames_to_column(var = "Parameter") %>%
  rename(`Point est.` = `Point est.`, `Upper C.I.` = `Upper C.I.`)

kable(gelman_df, digits = 3, caption = "Gelman Diagnostic Results")
```

Table 1: Gelman Diagnostic Results

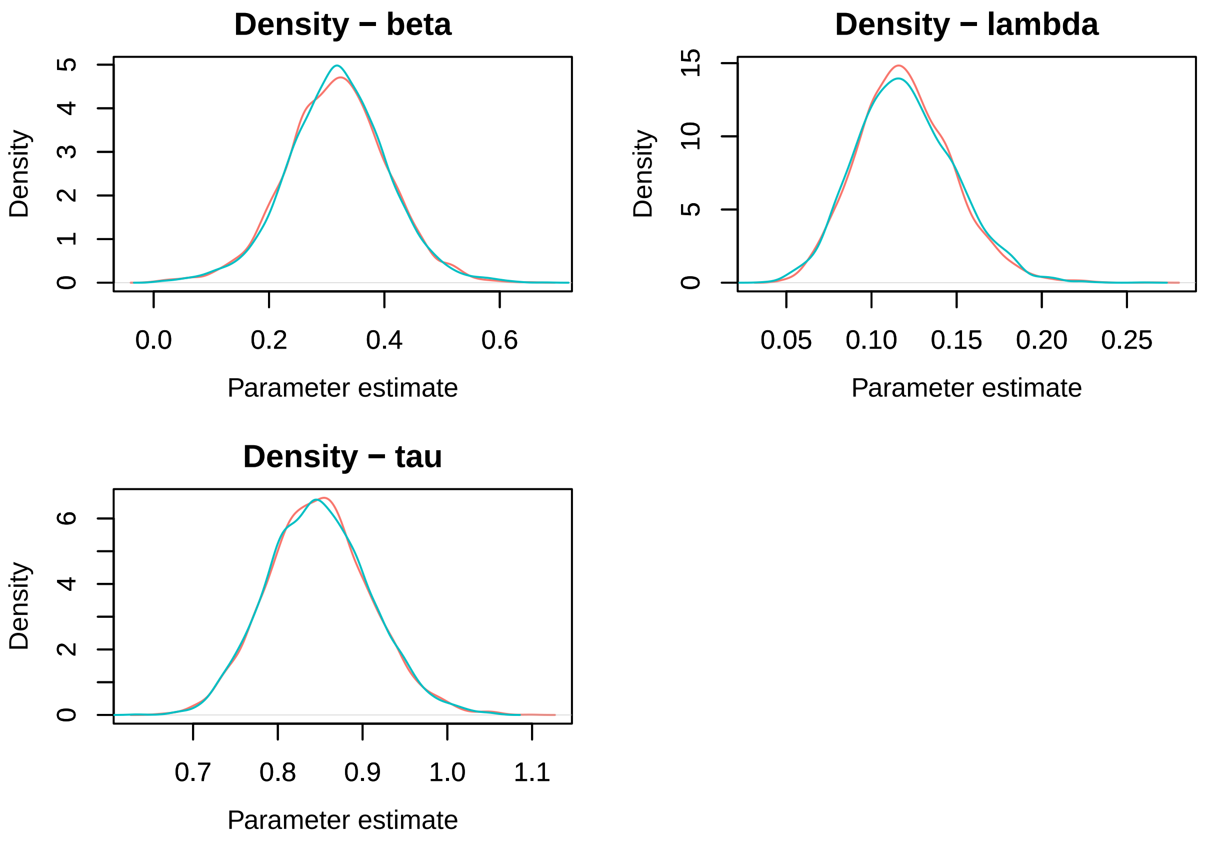
Parameter	Point est.	Upper C.I.
beta	1.000	1.001
lambda	1.001	1.001
mu	1.000	1.002
sigma_alpha2	1.001	1.002
sigma_y2	1.000	1.002
tau	1.000	1.000

Looking at the Gelman diagnostics results in the above table, all of the upper confidence interval values for all parameters are close to 1.00. This shows that the chains have mixed well and converged.

6.

Posterior Density Plots

```r
MCMCtrace(jagsfit.mcmc,
          params = c("beta", "lambda", "tau"),
          type = 'density',
          ind = TRUE, ISB = FALSE,
          exact = FALSE, pdf = FALSE)
```



Posterior Summary Statistics

```r
summary(jagsfit.mcmc)
```

```
##
## Iterations = 5001:10000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                  Mean       SD  Naive SE Time-series SE
## beta         0.31799 0.087085 8.709e-04      0.0017401
## lambda       0.11999 0.028257 2.826e-04      0.0004732
## mu           5.71882 0.050535 5.054e-04      0.0009443
## sigma_alpha2 0.02678 0.013197 1.320e-04      0.0002633
## sigma_y2     0.03615 0.002435 2.435e-05      0.0000357
## tau          0.84855 0.060464 6.046e-04      0.0012526
##
## 2. Quantiles for each variable:
##
##                  2.5%     25%     50%     75%   97.5%
## beta         0.14127 0.26181 0.31842 0.37413 0.49129
## lambda       0.06949 0.10027 0.11819 0.13843 0.17971
## mu           5.61807 5.68619 5.71894 5.75132 5.82006
## sigma_alpha2 0.01095 0.01779 0.02366 0.03225 0.06127
## sigma_y2     0.03157 0.03449 0.03606 0.03771 0.04124
## tau          0.73460 0.80696 0.84702 0.88789 0.97202
```

```r
# We want 'Time-Series SE (MCMC SE) = 1-5% of 'SD'
```

```r
# Checking enough samples

# Create a data frame for better presentation
MC_errors <- data.frame(
  Parameter = c("Beta", "Lambda", "Tau"),
  MC_Error = c(0.0017401, 0.0004732, 0.0012526),
  Std_Dev = c(0.087085, 0.028257, 0.060464),
  MC_Error_Percent = c(0.0017401/0.087085 * 100,
                       0.0004732/0.028257 * 100,
                       0.0012526/0.060464 * 100)
)
# Present table using kable()
kable(MC_errors, col.names = c("Parameter", "MC Error", "Standard Deviation", "MC Error (%)"),
      caption = "Checking Sample Size")
```

Table 2: Checking Sample Size

Parameter	MC Error	Standard Deviation	MC Error (%)
Beta	0.0017401	0.087085	1.998163
Lambda	0.0004732	0.028257	1.674629
Tau	0.0012526	0.060464	2.071646

From the above table, we can see that the MC error % for Beta, Lambda and Tau are between 1-5%. Therefore, we have enough samples for posterior inference.

```r
# Extracting position of the beta, lambda, and tau parameters
pos_beta <- substr(rownames(jags.mod.fit$BUGSoutput$summary), 1, 4) == 'beta'
pos_lambda <- substr(rownames(jags.mod.fit$BUGSoutput$summary), 1, 6) == 'lambda'
pos_tau <- substr(rownames(jags.mod.fit$BUGSoutput$summary), 1, 3) == 'tau'

# Extracting posterior mean for each parameter
beta_median <- exp(jags.mod.fit$BUGSoutput$summary[pos_beta, 5]) # exponentiate beta
lambda_median <- jags.mod.fit$BUGSoutput$summary[pos_lambda, 5] # lambda remains unchanged
tau_median <- exp(jags.mod.fit$BUGSoutput$summary[pos_tau, 5]) # exponentiate tau

# Extracting 2.5 percentile of the posterior
beta_lcr <- exp(jags.mod.fit$BUGSoutput$summary[pos_beta, 3])
lambda_lcr <- jags.mod.fit$BUGSoutput$summary[pos_lambda, 3]
tau_lcr <- exp(jags.mod.fit$BUGSoutput$summary[pos_tau, 3])

# Extracting 97.5 percentile of the posterior
beta_ucr <- exp(jags.mod.fit$BUGSoutput$summary[pos_beta, 7])
lambda_ucr <- jags.mod.fit$BUGSoutput$summary[pos_lambda, 7]
tau_ucr <- exp(jags.mod.fit$BUGSoutput$summary[pos_tau, 7])

summary_df <- data.frame(
  Parameter = c("Beta", "Lambda", "Tau"),
  Median = c(beta_median, lambda_median, tau_median),
  `2.5%` = c(beta_lcr, lambda_lcr, tau_lcr),
  `97.5%` = c(beta_ucr, lambda_ucr, tau_ucr)
)


kable(summary_df, digits = 3, caption = "Posterior Summary Statistics
      (Beta and Tau exponentiated to Original Scale)")
```

Table 3: Posterior Summary Statistics (Beta and Tau exponentiated to Original Scale)

Parameter	Median	X2.5.	X97.5.
Beta	1.375	1.152	1.634
Lambda	0.118	0.069	0.180
Tau	2.333	2.085	2.643

Looking at the posterior density plots and summary statistics above, we can conclude that we have enough samples for posterior inference for β, τ and λ, because the chains have mixed well and converged with smooth posterior densities. We have also checked the MC error is less than 1-5% of the posterior standard deviation.

We can see in the posterior summary statistics table above that β has a median value of 1.375. This means that schizophrenic people with no attention deficit take 37% longer to react than non-schizophrenics. For the Schizophrenic people with attention deficits, we times β by τ.

```r
1.375*2.333
```

```
## [1] 3.207875
```

This value shows that in trials where attention lapse occurs, reaction times are 3.2% longer than those of non-schizophrenics. This is a 220% percentage increase in reaction times from non-schizophrenic individuals.

This therefore concludes that schizophrenia affects both motor retardation and cognitive focus, leading to schizophrenics having a slower reaction time than non-schizophrenics on average.

7.

Prediction as Model Checking

(a)

In order to add 30 additional logged predictions for each schizophrenic individual, we will add a new term (y.pred[i, j]) for each person (i = 12 to 17). Using the posterior of $\alpha.s[i]$, the predicted response times maintain individual variability. As well as this, the Bernoulli-distributed delay will be retained for these individuals' predictions which ensures that the updated model reflects the observed pattern from the data.

```r
set.seed(123)
```

```r
jags.model2 <- function(){
  # Reaction time of non-schizophrenics
  for(i in 1:11){
    for(j in 1:30){
      y.ns[i, j] ~ dnorm(alpha.ns[i], p.y)
    }
    alpha.ns[i] ~ dnorm(mu, p.alpha)
}
# Reaction time of schizophrenics
  for(i in 1:6){
    for(j in 1:30){
      z[i, j] ~ dbern(lambda)
      y.s[i, j] ~ dnorm(alpha.s[i] + tau * z[i, j], p.y)
      }
      alpha.s[i] ~ dnorm(mu + beta, p.alpha)

   # **Predicted Log Reaction Times (y.pred)**
    for(j in 1:30){
      y.pred[i, j] ~ dnorm(alpha.s[i] + tau * z[i, j], p.y) # extra node
    }

  }

# Priors

mu ~ dunif(-10, 10)
beta ~ dunif(0, 10)
tau ~ dunif(0, 10)
lambda ~ dunif(0, 1)
sigma_y2 ~ dunif(0, 10)
p.y <- 1/ sigma_y2
sigma_alpha2 ~ dunif(0, 5)
p.alpha <- 1 / sigma_alpha2

}

# Parameters we want to monitor

jags.param2 <- c("y.pred")
```

(b)

In order to assess the variability in the predicted reaction times, we will compute the standard deviation of the 30 predicted values (y.pred[i, j]) within the JAGS model for each schizophrenic individual (i = 1, . . . , 6). The standard deviation (sd_pred[i]) shows the spread of predictred response times for each person.

After computing the six standard deviations, we compute the smallest (Smin) and largest (Smax) values to assess the range of predicted variability between each individual.

```
jags.model2 <- function(){
  # Reaction time of non-schizophrenics
  for(i in 1:11){
    for(j in 1:30){
      y.ns[i, j] ~ dnorm(alpha.ns[i], p.y)
    }
    alpha.ns[i] ~ dnorm(mu, p.alpha)
}
# Reaction time of schizophrenics
  for(i in 1:6){
    for(j in 1:30){
      z[i, j] ~ dbern(lambda)
      y.s[i, j] ~ dnorm(alpha.s[i] + tau * z[i, j], p.y)
      }
      alpha.s[i] ~ dnorm(mu + beta, p.alpha)

   # **Predicted Log Reaction Times (y.pred)**
    for(j in 1:30){
      y.pred[i, j] ~ dnorm(alpha.s[i] + tau * z[i, j], p.y) # extra node
      z.pred[i, j] ~ dbern(lambda)
    }
      sd_pred[i] <- sd(y.pred[i, ]) # computed standard deviation
  }
  Smin <- min(sd_pred[]) #standard deviation min
  Smax <- max(sd_pred[]) # standard deviation max


# Priors

mu ~ dunif(-10, 10)
beta ~ dunif(0, 10)
tau ~ dunif(0, 10)
lambda ~ dunif(0, 1)
sigma_y2 ~ dunif(0, 10)
p.y <- 1/ sigma_y2
sigma_alpha2 ~ dunif(0, 5)
p.alpha <- 1 / sigma_alpha2

}

# Parameters we want to monitor

jags.param2 <- c("y.pred", "sd_pred", "Smin", "Smax")
```

(c)

```
inits3 <- list(
  mu = 5.44, beta = 0.42, tau = 0.27, lambda = 0.3, sigma_y2 = 0.2,  sigma_alpha2 = 0.2
)

# The initial values for the parameters for chain 2 have been selected randomly.

inits4 <- list(
  mu = 6.5, beta = 0.7, tau = 0.3, lambda = 0.6, sigma_y2 = 0.35, sigma_alpha2 = 0.4
)

jags.inits2 <- list(inits3, inits4)
```

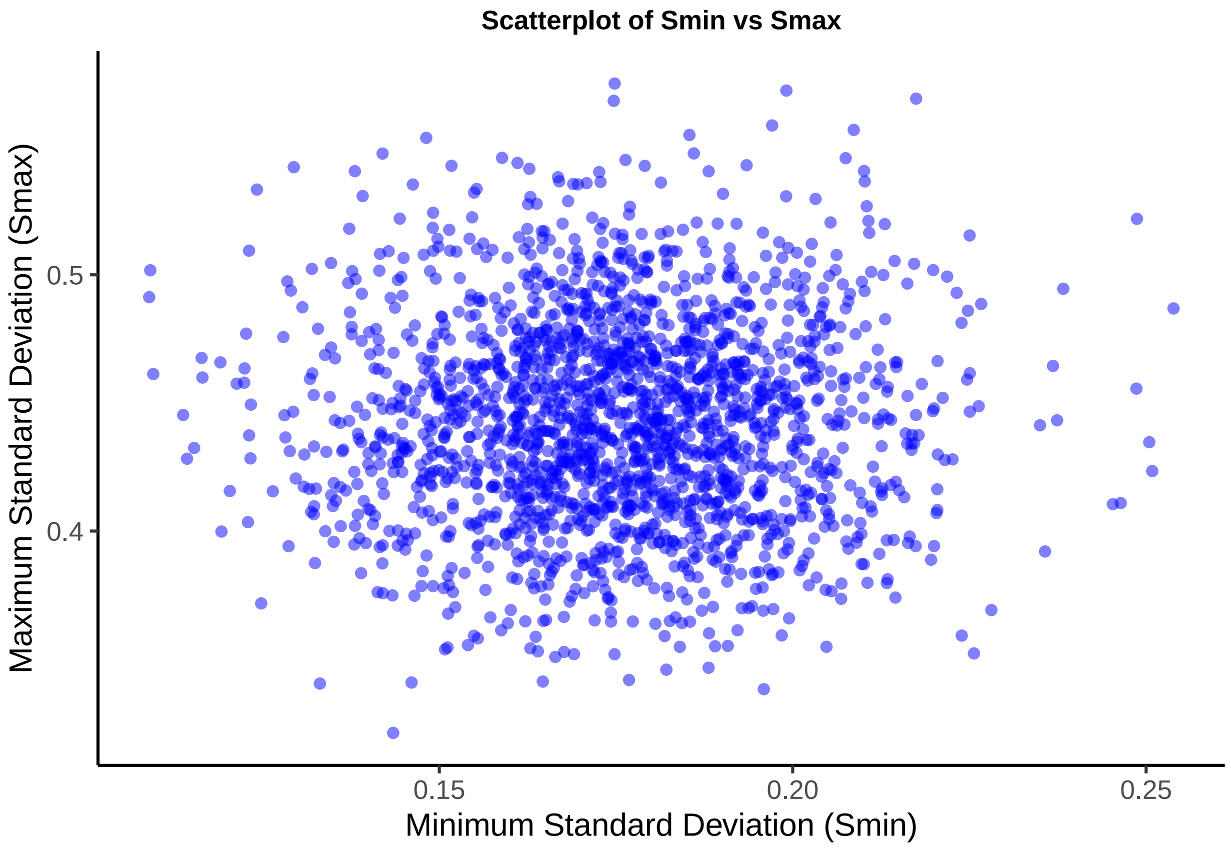
```
jags.mod.fit2 <- jags(data = jags.data, inits = jags.inits2,
                     parameters.to.save = jags.param2, n.chains = 2, n.iter = 6000,
                     n.burnin = 5000, n.thin=1, model.file = jags.model2, DIC = FALSE)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 510
##    Unobserved stochastic nodes: 563
##    Total graph size: 1456
##
## Initializing model
```

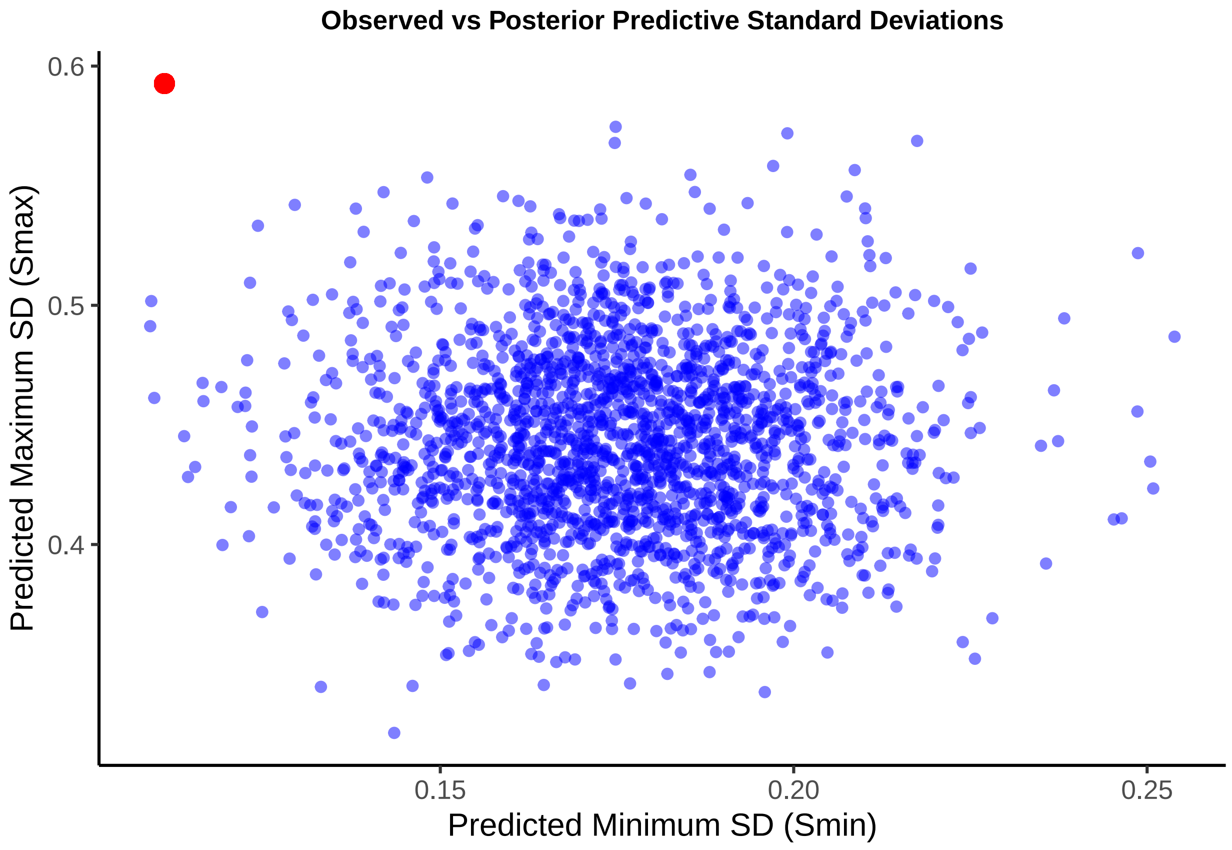
```
summary(jags.mod.fit2)
```

```
##                     Length Class  Mode
## model                8     jags   list
## BUGSoutput          22     bugs   list
## parameters.to.save  4      -none- character
## model.file           1     -none- character
## n.iter               1     -none- numeric
## DIC                  1     -none- logical
```

```
# Extract `Smin` and `Smax` samples
Smin_samples <- as.numeric(jags.mod.fit2$BUGSoutput$sims.list$Smin)
Smax_samples <- as.numeric(jags.mod.fit2$BUGSoutput$sims.list$Smax)

# Converting to a data frame
Smin_Smax_df <- data.frame(Smin = Smin_samples, Smax = Smax_samples)

# Scatterplot of Smin vs Smax
ggplot(Smin_Smax_df, aes(x = Smin, y = Smax)) +
  geom_point(alpha = 0.5, color = "blue") +
  labs(title = "Scatterplot of Smin vs Smax",
       x = "Minimum Standard Deviation (Smin)",
       y = "Maximum Standard Deviation (Smax)") +
  custom_theme
```



```r
# Computing the minimum and maximum observed SD's from log_sd

observed_Smin <- min(log_sd$SD_LogReactionTime)
observed_Smax <- max(log_sd$SD_LogReactionTime)

# Scatterplot comparing posterior predictive SDs and observed SDs
ggplot(Smin_Smax_df, aes(x = Smin, y = Smax)) +
  geom_point(alpha = 0.5, color = "blue") +  # Posterior predictive SDs
  geom_point(aes(x = observed_Smin, y = observed_Smax),
             color = "red", size = 3) +  # Observed SDs
  labs(title = "Observed vs Posterior Predictive Standard Deviations",
       x = "Predicted Minimum SD (Smin)",
       y = "Predicted Maximum SD (Smax)") +
  custom_theme
```



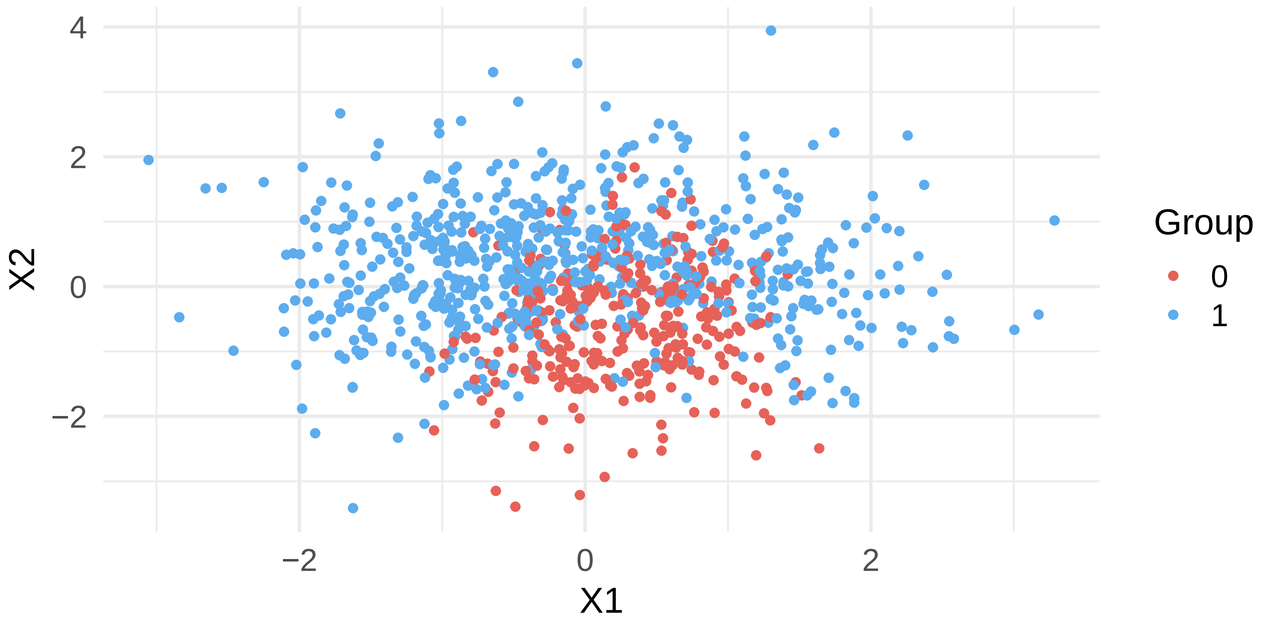
The above scatterplot displays the relationship between the minimum (Smin) and maximum (Smax) standard deviations of the predicted response times for schizophrenic individuals. The red point refers to the observed raw standard deviation estimates. The red observed point is far outside the cluster of blue points toward the top left portion of the plot. This suggests that the observed standard deviation values are significantly different from the predicted values.

The range of the predicted SD values do not seem to capture the observed SD points. The observed SD values are larger than the predicted values and so the model underestimates and fails to fully explain the within-person variability in reaction times. This observed raw minimum-maximum pair is just one observation and therefore there is a lack of observed data points to accurately explain the variation of the predicted data points. Thus, inference of the model's variance is unreliable.

B. Classification

The figure below shows the information in the dataset `Classification.csv` - it shows two different groups, plotted against two explanatory variables.

```r
# Read the dataset
classification_data <- read.csv("Classification.csv")
```



1.

```r
# Summary table for X1
summary_X1 <- classification_data %>%
  group_by(Group) %>%
  summarise(
    Count = n(),
    Mean = mean(X1),
    SD = sd(X1),
    Min = min(X1),
    Q1 = quantile(X1, 0.25),
    Median = median(X1),
    Q3 = quantile(X1, 0.75),
    IQR = Q3 - Q1,
    Max = max(X1)
  )

# Summary table for X2
summary_X2 <- classification_data %>%
  group_by(Group) %>%
  summarise(
    Count = n(),
    Mean = mean(X2),
    SD = sd(X2),
    Min = min(X2),
    Q1 = quantile(X2, 0.25),
    Median = median(X2),
    Q3 = quantile(X2, 0.75),
    IQR = Q3 - Q1,
    Max = max(X2)
  )


kable(summary_X1, caption = "Summary Statistics for X1")
```

Table 4: Summary Statistics for X1

Group	Count	Mean	SD	Min	Q1	Median	Q3	IQR	Max
0	318	0.2595396	0.5445669	-1.089792	-0.1344959	0.2611133	0.6447207	0.7792166	1.638613
1	682	-0.1180367	1.1190846	-3.055858	-0.9339504	-0.3030602	0.6534641	1.5874145	3.285469

```r
kable(summary_X2, caption = "Summary Statistics for X2")
```

Table 5: Summary Statistics for X2

Group	Count	Mean	SD	Min	Q1	Median	Q3	IQR	Max
0	318	-0.6009877	0.8852118	-3.392178	-1.2038826	-0.6009561	-0.0063410	1.197542	1.836123
1	682	0.3041380	0.9327773	-3.414288	-0.2662734	0.3058376	0.8870264	1.153300	3.943965

If we look at the variable X1, we can see that group 0 has a higher mean of 0.2595 than group 1 with a mean of -0.1180. This suggests that group 0 tends to have slightly larger values of X1. In comparison, group 0 has a negative mean of -0.6009 compared to group 1 with a mean of 0.3041. The difference here is greater and therefore this indicates that X2 plays a stronger role in distinguishing the two groups.

We can compare the variability in the two variables by comparing the standard deviations of the variable groups respectively. X1 has a much larger standard deviation for group 1 with a value of 1.1191 compared to group 0 with a value of 0.5447. X2 on the other hand shows relatively similar variability across groups with values of 0.8852 and 0.9328 respectively. This insinuates that X2 is more stable in distinguishing between the two groups.

Looking at quartiles and skewness, we can see that X1 has a big difference between the interquartile ranges of group 0 and group 1. Group 0 for X1 is more concentrated with a smaller IQR of 0.7792. Group 1 for X1 has a wider spread with a larger IQR of 1.5876, meaning there is more variability in X1 values. For the variable X2, IQR ranges are 1.1975 and 1.1533 across the two groups, suggesting that the spread of the middle 50% of the data is roughly the same between the two groups.

X1 and X2 both contribute significantly to group separation respectively. X2 has a strong shift in its distribution, making it useful for classification. X1 has a high variance in group 1, which means a non-linear method such as random forest, quadratic discriminant analysis or support vector machines could be suitable for better classification of the data.

Linear discriminant analysis (LDA) could work since the data between X1 and X2 differs between groups, suggesting LDA could work. However, it would not be suitable because the variance differences in X1 suggest that a non-linear model might perform better and the scatterplot shows too much complexity

Quadratic discriminant analysis (QDA) can be suitable because of variance differences and because the data supports a non-linear model due to the complexity of the data.

Considering the plot above and the numerical summaries, k-nearest neighbor classification would be suitable because the scatter plot shows complex patterns rather than a simple linear/quadratic separation. X1 has high variability in group 1 and therefore a simple linear decision boundary such as with linear discriminant analysis may not be suitable in capturing the separation well.

Support Vector Machines (SVM) is a suitable classification method because there is overlap in X1 and X2. Since X1 has a higher variance in group 1, a flexible decision boundary might be needed. This is a good method if the data is not cleanly separable by methods such as LDA or QDA.

Random Forests could be suitable for classifying this data because there are complex non-linear relationships. This method will give high accuracy and robustness. Due to X1 having high variance in group 1, decision trees can handle the variability. The issue of X2 having similar IQR values across groups will be automatically handled by random forests which find the best splits for the groups.

2.

Splitting data for testing and training

```r
# Define function that will split the data into training and test sets
trainTestSplit <- function(df,trainPercent,seed1){
## Sample size percent
smp_size <- floor(trainPercent/100 * nrow(df))
## set the seed
set.seed(seed1)
train_ind <- sample(seq_len(nrow(df)), size = smp_size)
train_ind
}

# Split as training and test sets
train_ind <- trainTestSplit(classification_data,trainPercent=75,seed=123)
train <- classification_data[train_ind, ]
test <- classification_data[-train_ind, ]

dim(train)
```

```
## [1] 750   4
```

```r
dim(test)
```

```
## [1] 250   4
```

The classification data is therefore split into two parts. The training set which accounts for 75% of the data and the test set used to evaluate the model, accounting for 25% of the data. The function `trainTestSplit()` is used to randomly select a portion of the data for training. In order to ensure the same split is generated each time the code is run, the function includes a seed, which controls the randomness and ensures reproducibility.

3.

Quadratic discriminant analysis

Quadratic discriminant analysis (QDA) is a classification technique similar to linear discriminant analysis (LDA) but estimates separate variances and covariances for each class. QDA assumes that the observations from each class are drawn from a Gaussian distribution, but it allows for class-specific covariance measures. The different covariance matrices lead to curved decision boundaries rather than straight lines found in LDA.

Quadratic Discriminant Analysis is most effective when the variances are very different between classes and there are enough observations to accurately estimate the variances.

The discriminant function for QDA is:

$$\delta_k(x) = -\frac{1}{2}x^T\Sigma_k^{-1}x + x^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k + \log(\pi_k)$$

QDA involves plugging estimates for Σ_k, μ_k and π_k.

Based on Bayes' Theorem, the classifier assigns an observation $X = x$ to the class k that maximises $\delta_k(x)$. This means that an observation is assigned to the class for which the discriminant function is the largest.

```
library(MASS)
library(caret)
library(class)
library(e1071)

# Fitting QDA model

fit.qda <- qda(Group ~ X1 + X2, data = train)
fit.qda
```

```
## Call:
## qda(Group ~ X1 + X2, data = train)
##
## Prior probabilities of groups:
##     0     1
## 0.304 0.696
##
## Group means:
##           X1         X2
## 0  0.2655471 -0.5822295
## 1 -0.1109920  0.3304992
```

```
# Making predictions on the test set

qda.predict <- predict(fit.qda, newdata = test)

# Confusion Matrix

confusionMatrix(qda.predict$class, as.factor(test$Group))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  44  10
##          1  46 150
##
##                Accuracy : 0.776
##                  95% CI : (0.7192, 0.8261)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 2.435e-06
##
##                   Kappa : 0.4673
##
```

```
##  Mcnemar's Test P-Value : 2.910e-06
##
##             Sensitivity : 0.4889
##             Specificity : 0.9375
##          Pos Pred Value : 0.8148
##          Neg Pred Value : 0.7653
##              Prevalence : 0.3600
##          Detection Rate : 0.1760
##    Detection Prevalence : 0.2160
##       Balanced Accuracy : 0.7132
##
##        'Positive' Class : 0
##
```

The confusion matrix above shown from the QDA is shown to have an accuracy score of 77.6%. This implies that 77.6% of test cases were classified correctly. This true accuracy as shown by the confidence interval lies between 71.9% and 82.6%. The confusion matrix displays that there are 44 true negatives, suggesting that the model predicted class '0' when it was actually '0'. The model predicts 10 false negatives, implying that the model predicted '0' when it was actually '1'. There are 46 false positives where the model predicted '1' when it was actually '0'. The model predicted 150 true positives, correctly predicting class '1' when it was actually '1'. Overall, while the model performs well in predicted class 1, it makes more errors for class 0.

The confusion matrix displays a sensitivity score of 48.89%, confirming that the model struggles to detect '0' cases. However, the model has a high specificity score of 93.75%, which shows that it is excellent at detecting '1' cases. The Negative Predictive Value (NPV) given is 76.53%, indicating that 76.53% of predicted '1' cases were correct. The model has a Positive Predictive Value (PPV) of 81.48%, implying the model predicted 81.48% of '0' cases correctly.

The Kappa score of 0.4673 refers to the measure of agreement between the predicted and actual classifications beyond what would be expected by chance. This value displays a moderate agreement and that the model performs better than random chance, but suggests there is potentially better model fits.

Overall, the model is able to capture the underlying structure in the data well, with an accuracy score of 77.6% and a moderate Kappa value. The higher specificity value indicates that the model is more effective at correctly classifying class '1' instances, while is struggles with class '0' as evidenced by the sensitivity value. The results imply that the differences in covaraiance between the classes are significant and therefore the quadratic decision boundary is appropriate for this dataset.

K-nearest neighbour classification

K-nearest neighbor (KNN) classification works by computing the distance between a test point and all training data points. The algorithm identifies the 'k' closest training points to the test point and assigns the test point the most common class label among them. This approach allows us to classify new data based on the majority vote of its nearest neighbours. In our analysis, we will be using K-nearest neighbor classification to classify observations into one of the two groups based on the explanatory variables X1 and X2. This method aims to determine the underlying structure that separates the two groups.

We will use the `knn` function of the `class` package to fit the model, specifying the matrix of training predictors (train), matrix of test predictors (test), the train set labels and the number of neigbours considered (k).

```r
# Extracting predictor variables from training and test sets
xTrain <- train[, c("X1", "X2")]
xTest <- test[, c("X1", "X2")]

# Extracting class labels from training set
yTrain <- train$Group
```

```r
yTest <- test$Group

# fitting the knn model
set.seed(123)

fit = knn(train=xTrain,test=xTest,cl=yTrain,k=3)

fit <- as.factor(fit)
yTest <- as.factor(yTest)
levels(fit) <- levels(yTest)

confusionMatrix(fit, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  58  28
##          1  32 132
##
##                Accuracy : 0.76
##                  95% CI : (0.7021, 0.8116)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 3.145e-05
##
##                   Kappa : 0.4741
##
##  Mcnemar's Test P-Value : 0.6985
##
##             Sensitivity : 0.6444
##             Specificity : 0.8250
##          Pos Pred Value : 0.6744
##          Neg Pred Value : 0.8049
##              Prevalence : 0.3600
##          Detection Rate : 0.2320
##    Detection Prevalence : 0.3440
##       Balanced Accuracy : 0.7347
##
##        'Positive' Class : 0
##
```

The model gives an accuracy of 76%. This implies that the model correctly classifies 76% of cases. The model detects 64.44% of actual '0' cases as shown by the sensitivity score. Whereas the model detects 82.5% of actual '1' cases, shown by the specificity score. The positive predictive value shows that 67.44% of predicted '0' cases are correct, while the negative predictive value shows that 80.49% of the predicted '1' cases are correct.

In order to improve the model performance by optimising the number of neighbours used in the fitting procedure, we will use K-fold cross validation. K-fold cross validation is a technique used to divide the data set into K folds or K partitions. The Machine Learning model is trained on K - 1 folds and tested on the Kth fold. We will apply repeated cross-validation to fine-tune the KNN hyperparameter (k) and determine the optimal number of neighbours. Through training the model on our prepared dataset, we can identify the best value of k to maximise classification accuracy.

```r
# Setting training options
# Repeating 5-fold cross-valiation, ten times
opts <- trainControl(method = 'repeatedcv', number = 10, repeats = 5)

# Find optimal k (model)
mdl <- train(x=xTrain, y=yTrain,
            method='knn',
            trControl=opts,
            tuneGrid=data.frame(k=seq(2, 15)))

print(mdl)
```

```
## k-Nearest Neighbors
##
## 750 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 675, 675, 675, 675, 675, 675, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    2  0.3979451  0.3309209  0.2139111
##    3  0.3850484  0.3412856  0.2266667
##    4  0.3759945  0.3571007  0.2318133
##    5  0.3677060  0.3761670  0.2338489
##    6  0.3648134  0.3821979  0.2364444
##    7  0.3629518  0.3859801  0.2379238
##    8  0.3609975  0.3897946  0.2397926
##    9  0.3598307  0.3919795  0.2410459
##   10  0.3581236  0.3965011  0.2413964
##   11  0.3575850  0.3975690  0.2425737
##   12  0.3563276  0.4010278  0.2431265
##   13  0.3561531  0.4011321  0.2446681
##   14  0.3561553  0.4006646  0.2463733
##   15  0.3567181  0.3987898  0.2484111
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 13.
```

```r
# Refitting the model with k=13
set.seed(123)

fit = knn(train=xTrain,test=xTest,cl=yTrain,k=13)

fit <- as.factor(fit)
yTest <- as.factor(yTest)
levels(fit) <- levels(yTest)

confusionMatrix(fit, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  61  22
##          1  29 138
##
##                Accuracy : 0.796
##                  95% CI : (0.7407, 0.8442)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 6.024e-08
##
##                   Kappa : 0.5496
##
##  Mcnemar's Test P-Value : 0.4008
##
##             Sensitivity : 0.6778
##             Specificity : 0.8625
##          Pos Pred Value : 0.7349
##          Neg Pred Value : 0.8263
##              Prevalence : 0.3600
##          Detection Rate : 0.2440
##    Detection Prevalence : 0.3320
##       Balanced Accuracy : 0.7701
##
##        'Positive' Class : 0
##
```

The confusion matrix above shows the result of the KNN model after using the optimal k value of 13. The accuracy of the model is shown to be 79.6%, implying that 79.6% of the test cases were correctly classified.

The matrix shows that there were 61 correct predictions of 0 with 22 incorrect predictions of 0 when it was actually 1. The matrix displays there were 138 true positive predictions of 1 when it was actually 1, with 29 false positive predictions of 1 when it was actually 0.

The model has a sensitivity score of 67.78% for its correct classification of '0' cases and a specificity score of 86.25% for the correct classification of '1' cases. Therefore overall, the model is better at identifying '1's as it has fewer false positives but tends to miss some '0' cases. This is further shown by the positive predictive value (PPV) of 73.49,% indicating that 73.49% of instances predicted as '0' instances were actually '0'. The negative predictive value (NPV) of 82.63% shows that 82.63% of instances predicted as '1' were actually '1'. The Kappa value has improved with this refined model to a value of 0.5496, showing that there is moderate agreement between predicted and actual classifications. Overall, the model performed well with a good balance between sensitivity and specificity. The accuracy score of 79.6% with a 95% confidence interval between 74-84% shows a good fit for the data, but could imply that alternative models may be preferable.

Support vector machines

Support Vector Machines (SVM's) are a generalisation of support vector classifiers. Support vector classifiers were developed using a soft margin approach and placing the hyperplane in a way that correctly classifies most of the data points. SVM's work by finding the maximum margin hyperplane that separates the data into different classes while maximising the margin. Therefore, the decision boundary is defined by support vectors that are the closest data points from both classes to the hyperplane.

SVM's make use of the kernel trick, which enables it to handle non-linearity separable data by mapping the original feature space to a higher dimensional space where the classes are linearly separable. Increasing dimensionality is usually undesirable. However, SVM only implicitly works in this higher-dimensional space using the kernel function. This allows us to define complex decision boundaries without the computational cost of transforming the data explicitly. Kernels are therefore a generalised distance measure.

SVM's are inherently binary classifiers. This means that they can only separate two classes at a time. Multi-class classification is handled using one-versus-all or one-versus-one classifiers.

```r
library(kernlab)

yTrain <- as.factor(yTrain)

# Fitting the Support Vector Machine

SVM <- train(x=xTrain, y=yTrain, method='svmLinear')
SVM
```

```
## Support Vector Machines with Linear Kernel
##
## 750 samples
##   2 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 750, 750, 750, 750, 750, 750, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7307685  0.2427153
##
## Tuning parameter 'C' was held constant at a value of 1
```

```r
# Testing model on testing data
yTestPred <- predict(SVM, newdata=xTest)

yTestPred <- as.factor(yTestPred)
yTest <- as.factor(yTest)

confusionMatrix(yTestPred, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  34  13
##          1  56 147
##
##                Accuracy : 0.724
##                  95% CI : (0.6641, 0.7785)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.002997
##
##                   Kappa : 0.3311
##
##  Mcnemar's Test P-Value : 4.277e-07
##
##             Sensitivity : 0.3778
##             Specificity : 0.9187
##          Pos Pred Value : 0.7234
##          Neg Pred Value : 0.7241
##               Prevalence : 0.3600
##           Detection Rate : 0.1360
##     Detection Prevalence : 0.1880
##        Balanced Accuracy : 0.6483
##
##         'Positive' Class : 0
##
```

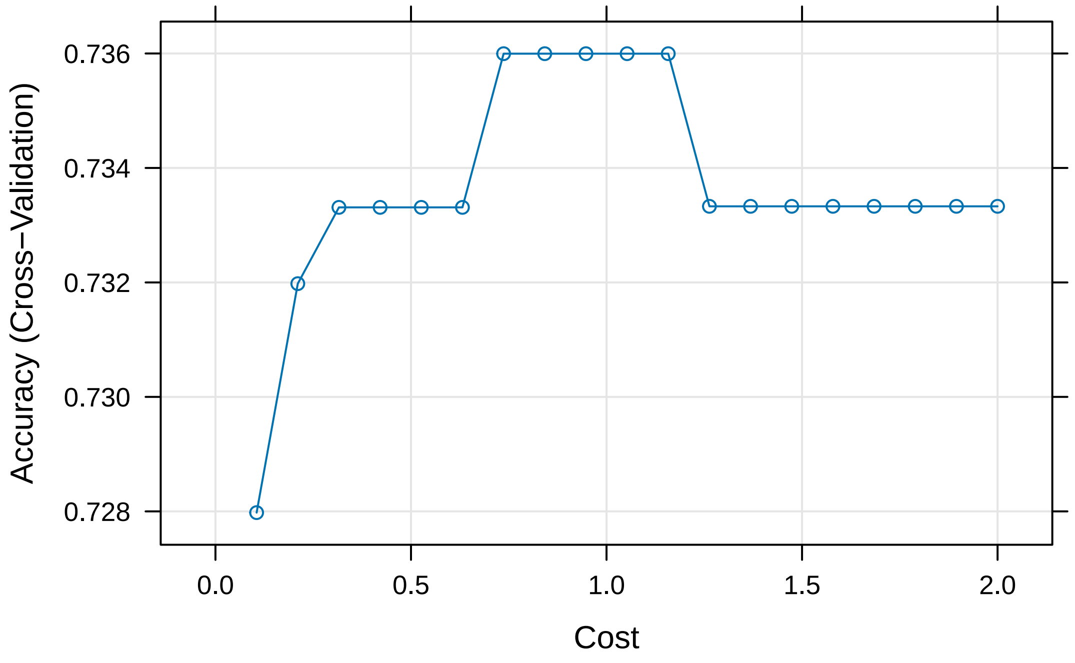
The confusion matrix and statistics summary above created from fitting the SVM model before undergoing cross-validation displays an accuracy score of 72.4%. This suggests that the mode correctly classifies 72.4% of cases.

The model predicts 34 true negatives, accounting for 34 correctly classified ‘0’ instances. It predicts 13 false negatives, accounting for the model predicted ‘0’ instances when it was actually ‘1’. There are 147 correctly classified ‘1’ cases. There are 56 false positives where the model predicted ‘1’ but the actual instance was ‘0’.

As a result of this, the model has a sensitivity score of 37.78%, correctly detecting 37.78% of actual ‘0’ cases. This is very low implying the model misses many ‘0’ cases. The specificity score shows that the model correctly detects 91.87% of ‘1’ cases. Both the positive predictive value and negative predictive values are 72% implying that 72% of predicted cases were actually correct as evidenced by the overall accuracy score. The Kappa score is 0.3311 which displays a weak agreement between predictions and true values.

In order to improve the model, we will undergo cross-validation and hyperparamter tuning to optimise ’C’and kernel selection to maximise the model’s cross-validation accuracy. In the model above, we have previously used C=1 as by default the SVM linear classifier uses this. Therefore, we will use a 5-fold cross validation, trying 20 different C values between 0 and 2.

```
set.seed(123)

# Fitting model with 5-fold cross validation and trying 20 different C values
SVM <- train(x=xTrain, y=yTrain, method = "svmLinear",
[illegible]
# Plotting model accuracy vs different values of cost
plot(SVM)
```



```r
# Printing best tuning parameter C which maximises the model accuracy
SVM$bestTune
```

```
##           C
## 8 0.7368421
```

The cross-validation plot and `SVM$bestTune` function above show that the optimal C value is 0.736. The plot specifically shows that at very small C values such as 0.1-0.2, the model has low accuracy, indicating under fitting. This is because small C values allow for more misclassification, leading to wider margins but weaker separation. The accuracy increases sharply until the value of 0.7 where the accuracy plateaus, suggesting that further increasing C does not improve the model performance. This implies that the margin has been optimised.

```r
# Test model on testing data
yTestPred <- predict(SVM, newdata=xTest)
confusionMatrix(yTestPred, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  34  13
##          1  56 147
##
##                Accuracy : 0.724
##                  95% CI : (0.6641, 0.7785)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.002997
##
##                   Kappa : 0.3311
##
##  Mcnemar's Test P-Value : 4.277e-07
##
##             Sensitivity : 0.3778
##             Specificity : 0.9187
##          Pos Pred Value : 0.7234
##          Neg Pred Value : 0.7241
##              Prevalence : 0.3600
##          Detection Rate : 0.1360
##    Detection Prevalence : 0.1880
##       Balanced Accuracy : 0.6483
##
##        'Positive' Class : 0
##
```

The confusion matrix and statistics with the SVMLinear kernel method are therefore the same. This is because $C = 1.26$ is still close to 1, so the decision boundary using this method remains similar. We will now try fitting an SVM model using a non-linear polynomial kernel.

```r
# Fitting model with polynomial kernel
set.seed(123)
SVM <- train(x = xTrain, y = yTrain, method = "svmPoly")

SVM$bestTune
```

```
##    degree scale C
## 27      3   0.1 1
```

```r
# Testing model on testing data

yTestPred <- predict(SVM, newdata=xTest)
confusionMatrix(yTestPred, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  52  15
##          1  38 145
##
##                Accuracy : 0.788
##                  95% CI : (0.7321, 0.837)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 2.835e-07
##
##                   Kappa : 0.5127
##
##  Mcnemar's Test P-Value : 0.002512
##
##             Sensitivity : 0.5778
##             Specificity : 0.9062
##          Pos Pred Value : 0.7761
##          Neg Pred Value : 0.7923
##              Prevalence : 0.3600
##          Detection Rate : 0.2080
##    Detection Prevalence : 0.2680
##       Balanced Accuracy : 0.7420
##
##        'Positive' Class : 0
##
```

As a result of using a non-linear polynomial model with optimised C value of 1 with a degree of 3 and scale 0.1, the model's accuracy has been improved to 78.8% as shown by the confusion matrix above. The sensitivity score increased from 36.67% to 57.78% and therefore this updated model is better at detecting class '0' cases. However, the specificity score slightly decreased by 1%. Noticeably, the Kappa score improved from 0.3396 to 0.5127 which shows that there is stronger agreement between predictions and actual values for this model.

We can also attempt to further optimise the model by fitting the SVM using a Radial Basis Function kernel.

```r
# Fitting the model with a Radial Basis Function kernel.
set.seed(123)
SVM <- train(x = xTrain, y = yTrain, method = "svmRadial")

SVM$bestTune
```

```
##      sigma    C
## 1 1.444326 0.25
```

```r
# Testing model on testing data

yTestPred <- predict(SVM, newdata=xTest)
confusionMatrix(yTestPred, yTest)
```

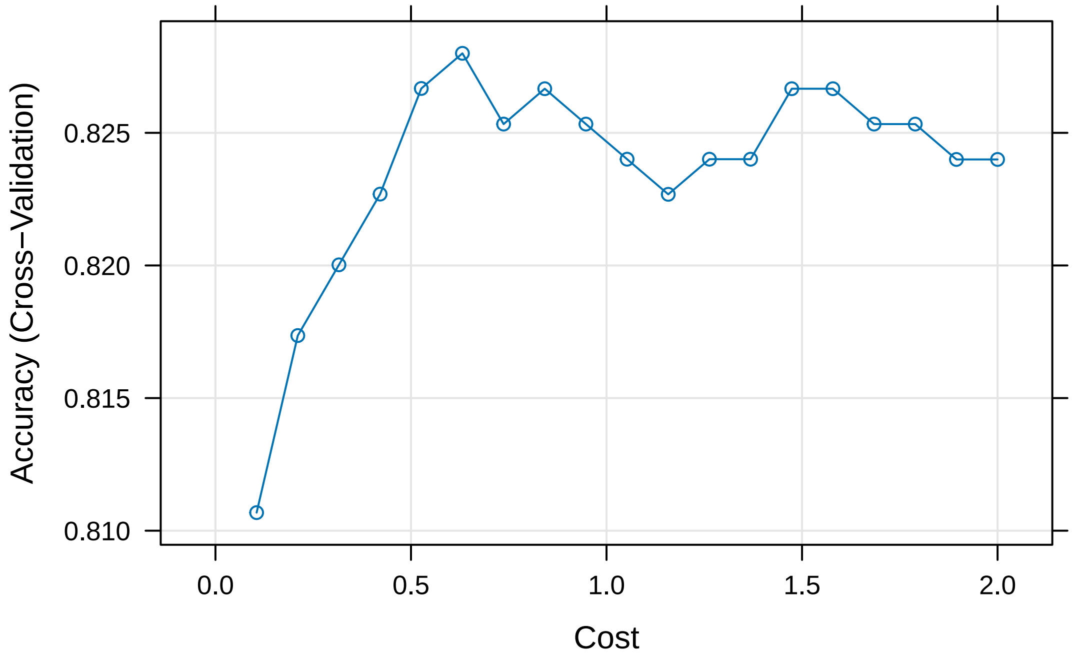
```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction   0   1
##          0  50  13
##          1  40 147
##
##                Accuracy : 0.788
##                  95% CI : (0.7321, 0.837)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 2.835e-07
##
##                   Kappa : 0.5076
##
##  Mcnemar's Test P-Value : 0.0003551
##
##             Sensitivity : 0.5556
##             Specificity : 0.9187
##          Pos Pred Value : 0.7937
##          Neg Pred Value : 0.7861
##              Prevalence : 0.3600
##          Detection Rate : 0.2000
##    Detection Prevalence : 0.2520
##       Balanced Accuracy : 0.7372
##
##        'Positive' Class : 0
##
```

After fitting the model using the Radial Basis Function kernel, the accuracy has come out as 78.8%, meaning the overall classification is good. The sensitivity has is 56%, showing a moderate detection of class '0' cases. The Kappa score is 0.5076, indicating a moderate agreement between predictions and actual values. There is a slight decrease false negatives with 50 values rather than 52 being recorded in the previous model.

However, to fully evaluate the effectiveness of the Radial Basis Function kernel, we can optimise the

```
set.seed(123)

# Fitting the Radial Model with 5-fold cross validation and trying 20 different values
SVM <- train(x=xTrain, y=yTrain, method = "svmRadial",
             trControl = trainControl("cv", number =5),
             tuneGrid = expand.grid(C = seq(0, 2, length = 20),
                                    sigma = 1.3))
plot(SVM)
```



This cross-validation accuracy plot against cost (C) parameter for the Radial Basis Function kernel shows that the best C value appears to be between 0.6-1.0, balancing accuracy and generalisation. Higher C values show to not improve accuracy and could lead to an increase in overfitting.

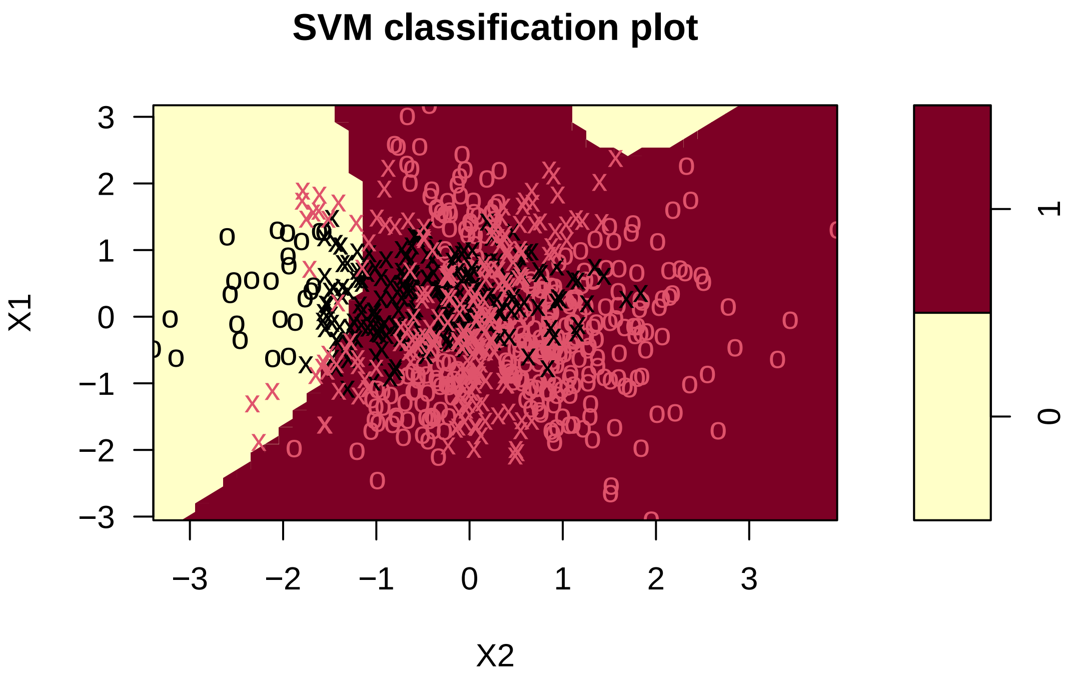
```r
# Test model on testing data
yTestPred_SVM <- predict(SVM, newdata=xTest)
confusionMatrix(yTestPred_SVM, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  55  14
##          1  35 146
##
##                Accuracy : 0.804
##                  95% CI : (0.7493, 0.8513)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 1.163e-08
##
##                   Kappa : 0.5518
##
##  Mcnemar's Test P-Value : 0.004275
##
##             Sensitivity : 0.6111
##             Specificity : 0.9125
##          Pos Pred Value : 0.7971
##          Neg Pred Value : 0.8066
##              Prevalence : 0.3600
##          Detection Rate : 0.2200
##    Detection Prevalence : 0.2760
##       Balanced Accuracy : 0.7618
##
##        'Positive' Class : 0
```

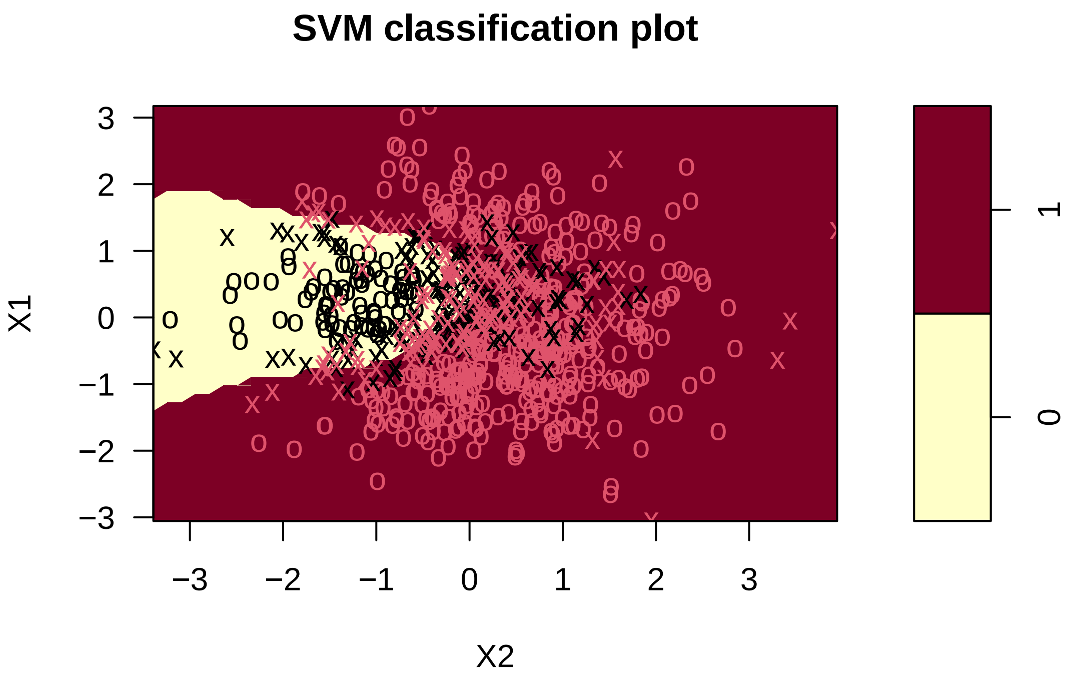
##

Having undergone cross-validation for the Radial Basis Function kernel, we have improved the accuracy of the model to 80.4%. The sensitivity score has increased to 61% with a specificity score of 91%. As well as this, the model has an improved Kappa score of 0.5518 which indicates a stronger agreement between predictions and actual values. Despite minor trade-off's such as a slight increase in false negatives with 55 values rather than 50 being recorded in the previous model, the model still performs well as accuracy has improved overall. This model has optimised the hyperparameters and provided the most robust SVM model fit, addressing the limitations of the other SVM models.

```r
# Plotting polynomial vs Radial Basis kernel types side-by-side for visual analysis of best model fit

df <- data.frame(xTrain, yTrain)

# Train & plot Polynomial SVM
mdl_poly <- svm(yTrain ~ ., data = df, kernel = "polynomial")
plot(mdl_poly, df, main = "Polynomial Kernel SVM")
```



```r
# Train & plot Radial SVM
mdl_rbf <- svm(yTrain ~ ., data = df, kernel = "radial")
plot(mdl_rbf, df, main = "Radial Basis Function SVM")
```



The classification plots above further display that the choice of Radial Basis Function kernel is the best option. The polynomial SVM in the first plot above has a curved and more flexible boundary but struggles with complex decision regions. The radial SVM has the most flexible decision boundary as shown by the plot and evidenced by the accuracy score. It adapts well to class distribution, minimising misclassification.

Random forests

Random forests is an ensemble method that has been developed in order to mitigate the problem of overfitting in decision trees whilst improving predictive performance. Random forests build multiple decision trees and combines their predictions to create a more stable and robust model.

Rather than training one tree, random forests grow T trees and aggregate their predictions. Each tree is known as a weak learner, but their combined output results in a strong overall model. Randomness is then added for better generalisation through bagging (bootstrap aggregation) where each tree is trained on a random subset of the data with replacement. At each split, only a random subset of predictors is considered which reduces the correlation between trees. For classification, random forests take a majority vote for all decision tress and for regression the predictions are averaged across all trees.

There is built-in cross-validation within the random forests model process as every tree is trained only on a subset of the original data. Random forests automatically estimate generalisation error using out-of-bag (OOB) samples. These samples are data points not used in training a particular tree, allowing error estimation without additional validation sets. The OOB data are also used when computing the estimate of the importance of each predictor which can then be used for feature selection.

```
library(randomForest)
# Fit Random Forest model
# Fix ntree and mtry

set.seed(123)

mdl <- train(x=xTrain, y=yTrain,
             method='rf',
             ntree=200,
             tuneGrid=data.frame(mtry=2))
```

```r
print(mdl)
```

```
## Random Forest
##
## 750 samples
##   2 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 750, 750, 750, 750, 750, 750, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.803138  0.5218263
##
## Tuning parameter 'mtry' was held constant at a value of 2
```

As shown by the Random Forest model above, the accuracy score of the model is 79.5%, implying that the model correctly classifies 79.5% of the test samples. The model has a kappa score of 0.5085. This indicates moderate agreement, suggesting that the model is making meaningful predictions yet it still has room for improvement. The model was trained using 25 bootstrap resampling iterations, meaning that the trees were built using randomly sampled subsets of data. This improves generalisation and reduces overfitting as we have mentioned. The hyperparameter 'mtry' is fixed at 2, meaning at each split, 2 predictors were randomly chosen.

```r
# Test model on testing data
yTestPred_RF <- predict(mdl, newdata=xTest)
confusionMatrix(yTestPred_RF, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  56  28
##          1  34 132
##
##                Accuracy : 0.752
##                  95% CI : (0.6937, 0.8043)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 9.955e-05
##
##                   Kappa : 0.4538
##
##  Mcnemar's Test P-Value : 0.5254
##
##             Sensitivity : 0.6222
##             Specificity : 0.8250
##          Pos Pred Value : 0.6667
##          Neg Pred Value : 0.7952
##              Prevalence : 0.3600
##          Detection Rate : 0.2240
##    Detection Prevalence : 0.3360
##       Balanced Accuracy : 0.7236
##
```

```
##        'Positive' Class : 0
##
```

After testing the model on the testing data, the confusion matrix above shows that the random forest model has a final accuracy of 76%. This lower accuracy suggests that there is some overfitting to the training data. The Kappa score has also dropped to 0.4662 indicating a loss of predictive agreement when generalised. The model has a sensitivity score of 62% suggesting it handles both classes fairly well and correctly identifies 62% of '0' cases. Whilst the specificity score indicates that 83% of class '1' cases are correctly identified. The PPV and NPV respectively show that the model correctly predicts '0' cases 67% of the time and '1' cases 79% of the time.

```r
# Variable importance by mean decrease in gini index
varImp(mdl$finalModel)
```

```
##     Overall
## X1 175.6713
## X2 141.4425
```

The output above displays the variable importance based on the mean decrease in gini index. The X1 value of 175 shows that X1 is the most important feature outranking X2 with a value of 141. X2 is also important as the values are similar, but X2 contributes less than X1 does. Therefore, X1 is the strongest predictor, followed by X2.

4.

Comparing all of the methods together from the four different approaches, the Support Vector Machine using a Radial Basis Function (RBF) kernel performed the best with an accuracy score of 80.4% after optimising hyperparameters. This method also provided the best sensitivity out of all the methods with a high specificity and the highest Kappa score of 0.54.

The Quadratic Discriminant Analysis method had an accuracy score of 77.6%, but had low sensitivity. However, QDA had very high specificity and so was good at detecting '1' cases. This method would work well if the data was normally distributed. QDA had the worst Kappa score of 0.46 suggesting that it struggled to find strong agreement between predicted and actual values. If the dataset is following a Gaussian distribution, then QDA is a good choice of method for classification.

K-nearest neighbour classification had an accuracy score of 76% with a moderate sensitivity and high specificity. This simple model performed well but its performance was very sensitive to k-value tuning. It had an overall Kappa score of 0.47 which was the second lowest out of the 4 different methods. This model can be most effective if computational efficiency is required because it is the easiest to implement.

Random Forest method provided us with a strong performance, handling non-linearity within the data as well as being interpretable with feature importance. This method had an accuracy score of 79.5%, just below the accuracy score of SVM (RBF). Random forest provided moderate sensitivity and high specificity scores of 61% and 84% respectively. Random Forest had a high Kappa score of 0.51 indicating that the model is making meaningful predictions.

Overall, if we are prioritising model accuracy then the SVM (RBF) model is the best method for classification because it achieves the highest accuracy (80.4%), balancing sensitivity and specificity, while handling complex decision boundaries well. This method is able to capture complex structures in the data more effectively than linear or polynomial models. SVM (RBF) also achieves the highest Kappa score of 0.54, suggesting that it has a strong agreement between predicted and actual values. Despite this, if our modelling objectives require feature importance analysis, then Random Forest is a strong alternative method due to its interpretability and robust generalisation. Therefore, the final model choice depends on the modelling objectives and trade-off between accuracy, interpretability, and computational efficiency for the particular classification task.

5.

Evaluation of Classification Methods Against the True Classifications

```r
# Read the dataset
true_labels <- read.csv("ClassificationTrue.csv")

yTrue <- true_labels$Group[-train_ind]  # Selecting only test set labels
yTrue <- as.factor(yTrue)  # Converting to factor

# Loading predictions

yTestPred_SVM <- predict(SVM, newdata=xTest) # SVM
yTestPred_RF <- predict(mdl, newdata=xTest) # Random Forest
yTestPred_KNN <- fit # KNN
yTestPred_QDA <- qda.predict$class #QDA


yTestPred_KNN <- as.factor(yTestPred_KNN)
yTestPred_QDA <- as.factor(yTestPred_QDA)
yTestPred_SVM <- as.factor(yTestPred_SVM)
yTestPred_RF  <- as.factor(yTestPred_RF)

# Creating Confusion Matrices for true labels

confusionMatrix(yTestPred_KNN, yTrue)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  65  18
##          1  15 152
##
##                Accuracy : 0.868
##                  95% CI : (0.8196, 0.9074)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 5.288e-12
##
##                   Kappa : 0.6997
##
##  Mcnemar's Test P-Value : 0.7277
##
##             Sensitivity : 0.8125
##             Specificity : 0.8941
##          Pos Pred Value : 0.7831
##          Neg Pred Value : 0.9102
##              Prevalence : 0.3200
##          Detection Rate : 0.2600
##    Detection Prevalence : 0.3320
##       Balanced Accuracy : 0.8533
##
##        'Positive' Class : 0
##
```

```r
confusionMatrix(yTestPred_QDA, yTrue)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  53   1
##          1  27 169
##
##                Accuracy : 0.888
##                  95% CI : (0.8422, 0.9243)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 1.174e-14
##
##                   Kappa : 0.7184
##
##  Mcnemar's Test P-Value : 2.306e-06
##
##             Sensitivity : 0.6625
##             Specificity : 0.9941
##          Pos Pred Value : 0.9815
##          Neg Pred Value : 0.8622
##              Prevalence : 0.3200
##          Detection Rate : 0.2120
##    Detection Prevalence : 0.2160
##       Balanced Accuracy : 0.8283
##
##        'Positive' Class : 0
##
```

```r
confusionMatrix(yTestPred_SVM, yTrue)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  61   8
##          1  19 162
##
##                Accuracy : 0.892
##                  95% CI : (0.8468, 0.9276)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 3.089e-15
##
##                   Kappa : 0.7425
##
##  Mcnemar's Test P-Value : 0.05429
##
##             Sensitivity : 0.7625
##             Specificity : 0.9529
##          Pos Pred Value : 0.8841
##          Neg Pred Value : 0.8950
##              Prevalence : 0.3200
##          Detection Rate : 0.2440
##    Detection Prevalence : 0.2760
##       Balanced Accuracy : 0.8577
##
##        'Positive' Class : 0
##
```

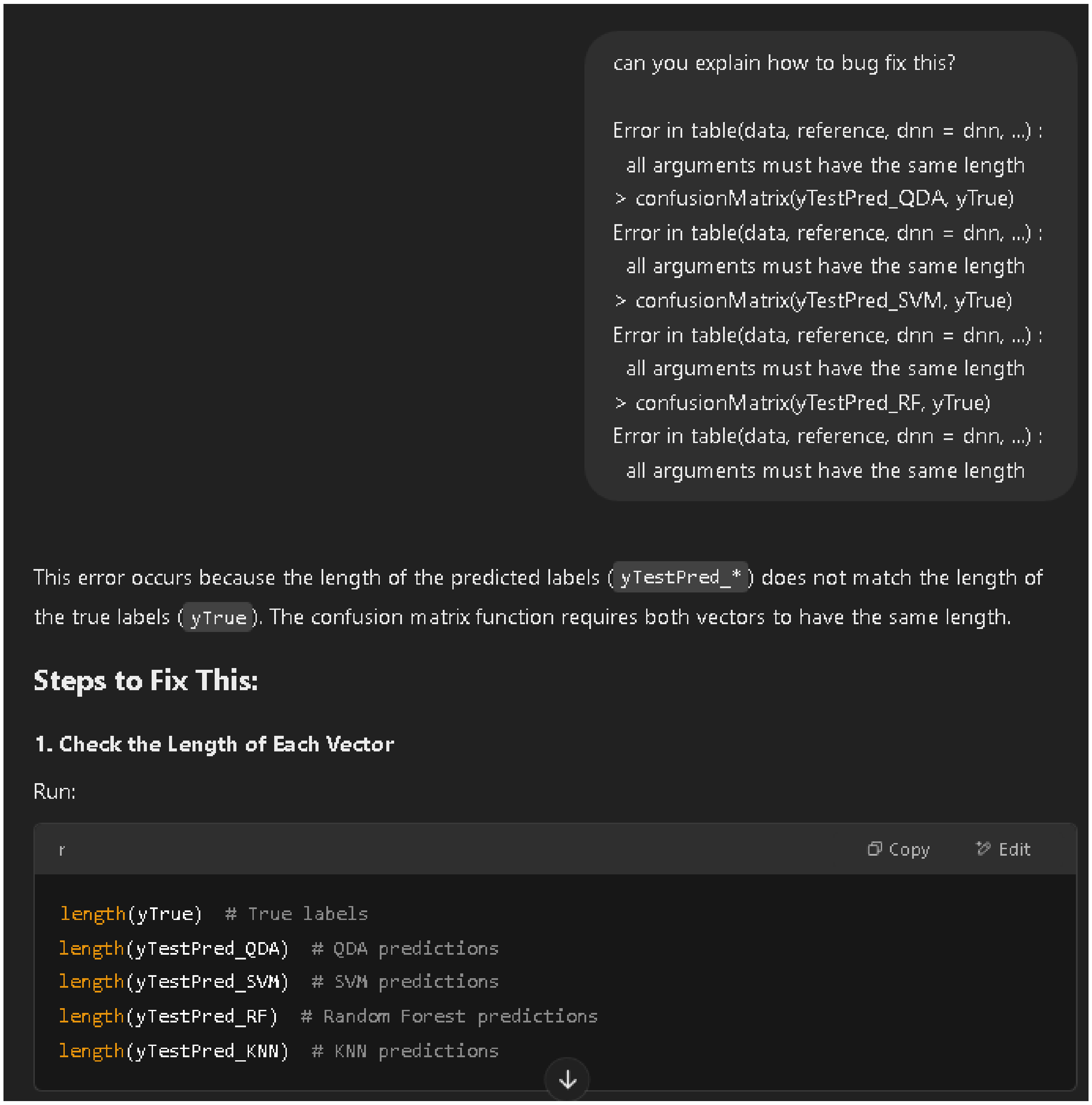
```r
confusionMatrix(yTestPred_RF, yTrue)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  60  24
##          1  20 146
##
##                Accuracy : 0.824
##                  95% CI : (0.771, 0.8691)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 2.06e-07
##
##                   Kappa : 0.6009
##
##  Mcnemar's Test P-Value : 0.6511
##
##             Sensitivity : 0.7500
##             Specificity : 0.8588
##          Pos Pred Value : 0.7143
##          Neg Pred Value : 0.8795
##              Prevalence : 0.3200
##          Detection Rate : 0.2400
##    Detection Prevalence : 0.3360
##       Balanced Accuracy : 0.8044
##
##        'Positive' Class : 0
##
```

After computing the confusion matrices for all four chosen methods with the true classifications, based on X1 and X2 without the noise, the method with the best accuracy score is still SVM (RBF) with a score of 89.6%. However, methods such as QDA improved the most with an accuracy score of 88.8%. K-nearest neighbour classification had an accuracy of 86.8% which also showed great improvement but did not out perform SVM. The Random Forest method showed the least improvement with an accuracy of 82.4%. Overall, the method with the best classification for this dataset is SVM (RBF), as it consistently outperforms the other methods in accuracy while maintaining a well-balanced classification performance. It has the highest Kappa score of 0.75, confirming the flexibility the RBF kernel allows in capturing the complex decision boundary of the true classification function much more effectively than the other models.

Appendix

GenAI Prompts



can you explain how to bug fix this?

```
Error in table(data, reference, dnn = dnn, ...) :
  all arguments must have the same length
> confusionMatrix(yTestPred_QDA, yTrue)
Error in table(data, reference, dnn = dnn, ...) :
  all arguments must have the same length
> confusionMatrix(yTestPred_SVM, yTrue)
Error in table(data, reference, dnn = dnn, ...) :
  all arguments must have the same length
> confusionMatrix(yTestPred_RF, yTrue)
Error in table(data, reference, dnn = dnn, ...) :
  all arguments must have the same length
```

This error occurs because the length of the predicted labels (`yTestPred_*`) does not match the length of the true labels (`yTrue`). The confusion matrix function requires both vectors to have the same length.

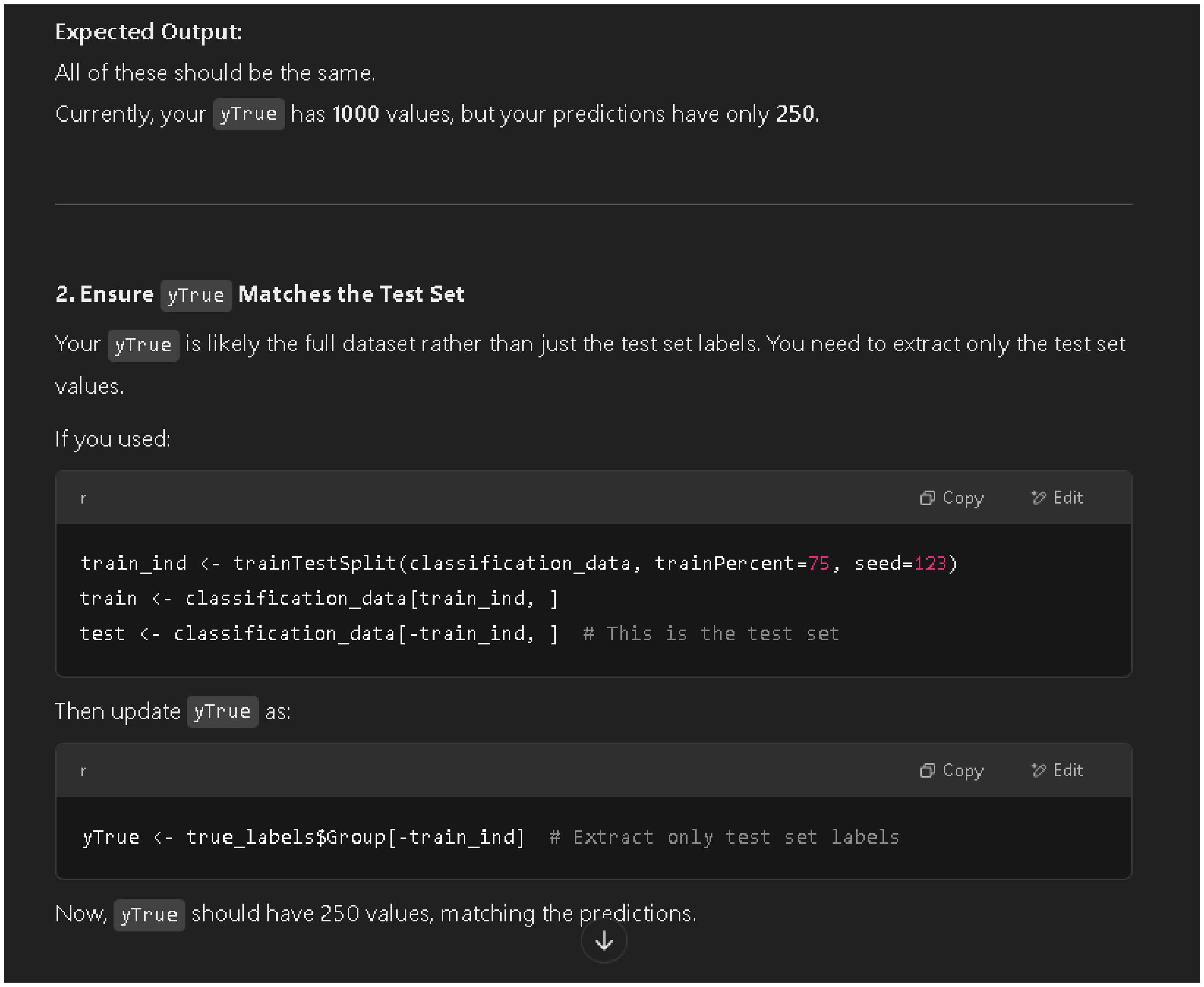
Steps to Fix This:

1. Check the Length of Each Vector

Run:

```r
length(yTrue)  # True labels
length(yTestPred_QDA)  # QDA predictions
length(yTestPred_SVM)  # SVM predictions
length(yTestPred_RF)  # Random Forest predictions
length(yTestPred_KNN)  # KNN predictions
```

Figure 1: Prompt 1



Expected Output:

All of these should be the same.

Currently, your `yTrue` has **1000** values, but your predictions have only **250**.

---

2. Ensure `yTrue` Matches the Test Set

Your `yTrue` is likely the full dataset rather than just the test set labels. You need to extract only the test set values.

If you used:

```r
train_ind <- trainTestSplit(classification_data, trainPercent=75, seed=123)
train <- classification_data[train_ind, ]
test <- classification_data[-train_ind, ]  # This is the test set
```

Then update `yTrue` as:

```r
yTrue <- true_labels$Group[-train_ind]  # Extract only test set labels
```

Now, `yTrue` should have 250 values, matching the predictions.

Figure 2: Prompt 2