MTHM017 - Advanced Topics in Statistics, Assignment

James R Lewis

14-03-2025

Contents

Declaration of AI Assistance: I have used OpenAI's ChatGPT tool in creating this report.

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

1. I have used GenAI tools to check and debug my code.
2. I have used GenAI tools to proofread and correct grammar or spelling errors.
3. I have used GenAI tools to give me feedback on a draft.

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

A. Bayesian Inference

1. Schizophrenic and Non-Schizophrenic individual's reaction times

We begin by making the wrangling the data into a more useable structure.

```
# Renaming first column
names(rtimes)[1] <- "PatientType"
# Classifying the patient type, 1-11 non-schiz, 12-17 schiz
rtimes$PatientType <- c(rep("non-schizophrenic", 11), rep("schizophrenic", 6))
rtimes1 <- rtimes
head(rtimes1)
```

```
##         PatientType  T1  T2  T3  T4  T5  T6  T7  T8  T9 T10 T11 T12 T13 T14 T15
## 1 non-schizophrenic 312 272 350 286 268 328 298 356 292 308 296 372 396 402 280
## 2 non-schizophrenic 354 346 384 342 302 312 322 376 306 402 320 298 308 414 304
## 3 non-schizophrenic 256 284 320 274 324 268 370 430 314 312 362 256 342 388 302
## 4 non-schizophrenic 260 294 306 292 264 290 272 268 344 362 330 280 354 320 334
## 5 non-schizophrenic 204 272 250 260 314 308 246 236 208 268 272 264 308 236 238
## 6 non-schizophrenic 590 312 286 310 778 364 318 316 316 298 344 262 274 330 312
##   T16 T17 T18 T19 T20 T21 T22 T23 T24 T25 T26 T27 T28 T29 T30
## 1 330 254 282 350 328 332 308 292 258 340 242 306 328 294 272
## 2 422 388 422 426 338 332 426 478 372 392 374 430 388 354 368
## 3 366 298 396 274 226 328 274 258 220 236 272 322 284 274 356
## 4 276 418 288 338 350 350 324 286 322 280 256 218 256 220 356
## 5 350 272 252 252 236 306 238 350 206 260 280 274 318 268 210
## 6 310 376 326 346 334 282 292 282 300 290 302 300 306 294 444
```

Creating Histograms

We want to extract the data from these columns **rtimes[, 2:31]**, for all 17 patients. We will plot 17 histograms, displaying the distribution of the 30 reaction times.

In order to keep a consistent range across the 17 histograms, we need to find the absolute minimum and maximum values in this dataset. To do this, we extract all the data, 17 rows * 30 trials.

```
DataValues <- unlist(rtimes1[, 2:31])
range(DataValues)
```

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

```
# We want to store these values

x_min <- 204
x_max <- 1714
```

Below, we can observe two sets of histograms, the first containing the eleven non-schizophrenic patients, and the seconds containing the 6 schizophrenic patients.

We reshaped to data frame to long format, as having the reaction time data in one column (a list) makes much easier to code the histogram. To create a separate plot for each patient, we use facet_wrap() to group data by PatientID.

```
# Reshaping to long format
rtimes_long <- rtimes1 %>%
```

```r
  pivot_longer(cols = starts_with("T"),  # Columns T1 to T30
               names_to = "Trial",
               values_to = "ReactionTime") %>%
  mutate(PatientID = rep(1:17, each = 30)) %>%  # Add patientID
  select(PatientID, PatientType, ReactionTime)

# Splitting into two datasets so I can have seperate histogramws
non_schizo <- rtimes_long %>% filter(PatientType == "non-schizophrenic")
schizo <- rtimes_long %>% filter(PatientType == "schizophrenic")
```

```r
# Plotting non-schizophrenic histograms
hist1 <- ggplot(non_schizo, aes(x = ReactionTime)) +
  geom_histogram(binwidth = 150, fill = "dodgerblue1", color = "black") +
  facet_wrap(~ PatientID, ncol = 4, scales = "free_y") +  # 11 plots, 4 columns
  coord_cartesian(xlim = c(x_min, x_max)) +  # Fixed x-axis
  labs(title = "Figure 1: Non-Schizophrenic Reaction Times",
       x = "Reaction Time (ms)",
       y = "Count") +
  custom_theme

# Plotting schizophrenic histograms
hist2 <- ggplot(schizo, aes(x = ReactionTime)) +
  geom_histogram(binwidth = 150, fill = "sienna2", color = "black") +
  facet_wrap(~ PatientID, ncol = 3, scales = "free_y") +  # 6 plots, 3 columns
  coord_cartesian(xlim = c(x_min, x_max)) +  # Fixed x-axis
  labs(title = "Figure 2: Schizophrenic Reaction Times",
       x = "Reaction Time (ms)",
       y = "Count") +
  custom_theme
```

Figure 1: Non−Schizophrenic Reaction Times

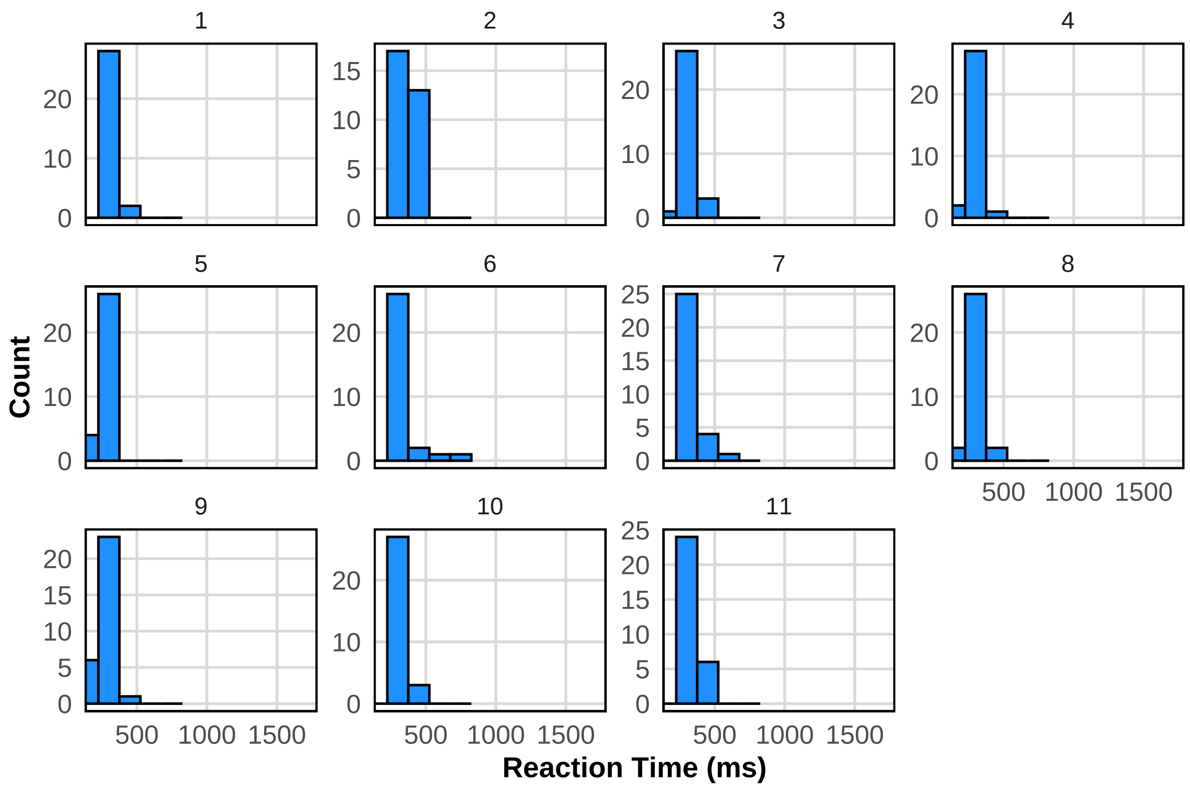
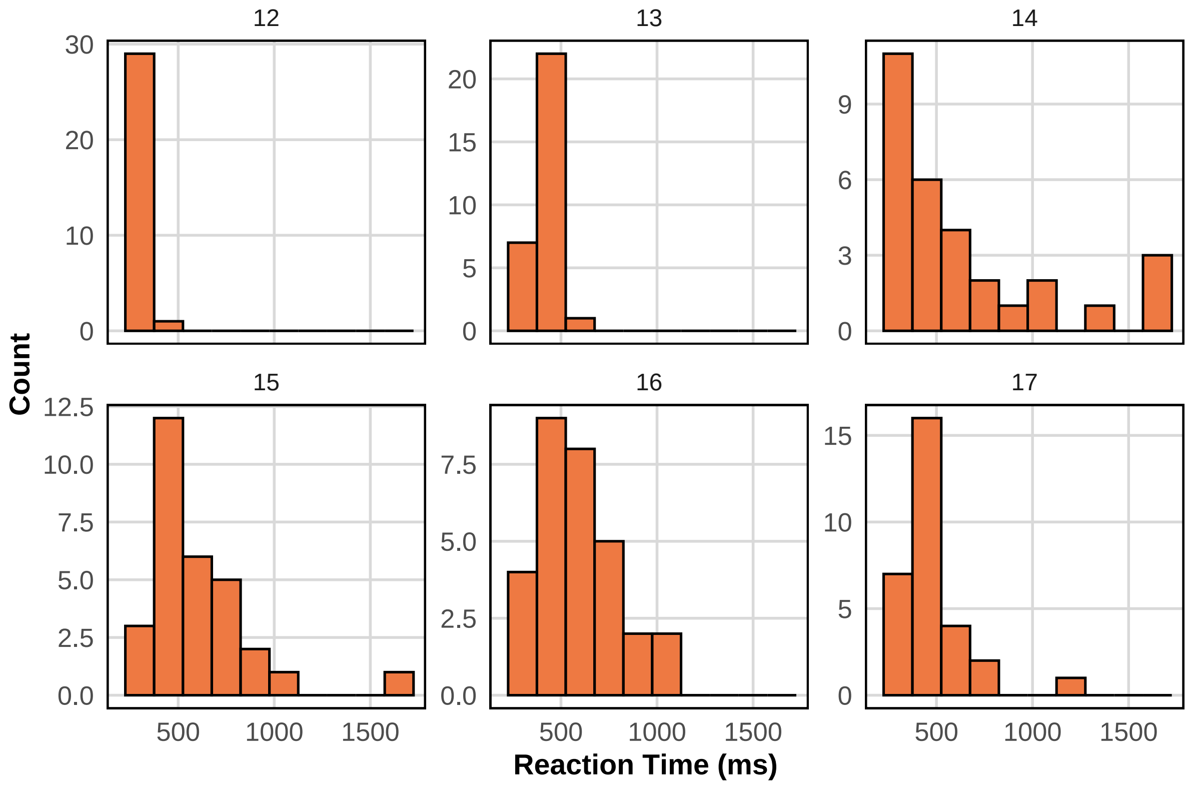


Figure 2: Schizophrenic Reaction Times



In Figure 1, non-schizophrenic patients, such as **Patient 1**, display histograms with denser distributions and lower variability. Reaction times are clustered around the second bin (**150–300ms**), with most patients having at least **two-thirds** of their reaction times falling below **300ms**. This indicates consistent and faster reaction times within this group.

In contrast, the histograms for schizophrenic patients in Figure 2 show a much wider spread of reaction times, with some exceeding **1500ms**. For instance, **Patient 14** demonstrates significant variability, with a large proportion of reaction times exceeding **1000ms**, but also a notable count below **500ms**. This suggests a mixture of both normal and delayed responses, aligning with the theory of attention deficits and motor retardation in schizophrenic individuals. These characteristics are reflected in their inconsistent distributions, right-skewed shapes, and greater variance.

Despite these differences, both groups share some similarities. Both have peaks within the first two bins **(0–500ms**), as seen when comparing **Patient 8** (non-schizophrenic) to **Patient 13** (schizophrenic). However, only schizophrenic patients display reaction times in the higher bands (greater than **800ms**), indicating greater variability and more extreme delays.

Overall, these histograms support the hypothesis that schizophrenic patients experience attention deficits and motor reflex retardation. This is evidenced by the wider spread of values, higher maximum reaction times, and the more irregular, skewed distributions observed in the schizophrenic group.

2. Preparing the Data

Log-transforming the reaction times is necessary for two key reasons. Firstly, log transformation **stabilises variance**, as reaction times y_{ij} are positively skewed. Applying $\log(y_{ij})$ helps to approximate a normal distribution, which is essential for meeting the model's assumptions. This skewness is evident in the histograms of schizophrenic patients, particularly **Patients 14** and **15**, where right-skewed distributions with extreme values (exceeding **1500ms**) are observed. By reducing skewness, the data better align with the model's assumption of normality.

Secondly, log transformation scales the data, making the **mean** and **variance** more interpretable. For example, the original range of **204ms** to **1714ms** becomes **5.31** to **7.44** (to three significant figures) when log-transformed. This makes the data easier to interpret and handle, as working with smaller numbers simplifies analysis and reduces the influence of extreme values.

```r
log_rtimes_long <- rtimes_long %>% # Log Transforming
  mutate(LogReactionTime = log(ReactionTime))
range(log_rtimes_long$LogReactionTime)
```

```
## [1] 5.318120 7.446585
```

```r
# Checking the range also lets us know if any observations are negative.
```

Now we compute the standard deviation of each Patient's 30 log-transformed reaction times. This is useful as it measures the within personal variability on the log scale.

```r
# Compute standard deviation of log-transformed reaction times for each person
sd_log <- log_rtimes_long %>%
  group_by(PatientID) %>%
  summarise(SD_Log_RT = sd(LogReactionTime))

# Print results
head(sd_log)
```

```
## # A tibble: 6 x 2
##   PatientID SD_Log_RT
##       <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. Parameters and Prior Distributions

Non-informative uniform prior distributions are used due to the lack of prior knowledge about the parameters. By specifying a broad and equally likely range of values, these priors **minimise bias** and allow the data to strongly influence the posterior distribution, ensuring that the model is driven by the observed evidence rather than subjective assumptions.

For Non-Schizophrenic Individuals:

- α_i (for $i = 1$ to 17): Person-specific means on the log scale.

```
alpha[i] ~ dunif(0, 10)
```

- μ: **Mean** of α_i for **non-schizophrenics**, which represents the mean log reaction time.

A broad range of [**-10,10**] was selected to remain non-informative and capture outliers, as the transformed log reaction times realistically range between approximately 5.3 and 7.5.

```
mu ~ dunif(-10, 10)
```

- (_y^2): Varience of log-reaction times. The lower bound (**0.001** rather than 0) avoids computational instability from precision going to infinity. An upper bound of **5** is sufficiently large to accommodate considerable variability.

```
sigma_y2 ~ dunif(0.001, 5)
# tau_y <- 1 / sigma_y2
```

- (_ ^2): Varience of α_i. The same reasoning as sigma_y2.

```
sigma_alpha2 ~ dunif(0.001, 5)
# tau_alpha <- 1 / sigma_alpha2
```

For Schizophrenic Individuals:

- β: Additional variable in α_i mean for **schizophrenics**, which represents the increase in reaction time due to schizophrenia. Although we expect schizophrenics to be slower, there is still the possibility of motor retardation having no difference, or faster reactions (unlikely, but possible). The chosen range [**0, 10**] ensures positivity while being wide enough to accommodate large variations in reaction times.

```
beta ~ dunif(0, 10)
```

- τ: Log-transformed **delay parameter** for **schizophrenics** ($\tau > 0$). This is a wide range covers all reasonable values, but remains uninformative and is contained to positive values.

```
tau ~ dunif(0, 10)
```

- λ: **Probability of delay** ($0 \leq \lambda \leq 1$). Value must lie between **0** and **1**, as λ is a probability.

```
lambda ~ dunif(0, 1)
```

4. JAGS Setup

```
library(R2jags)
```

```
## Loading required package: rjags

## Loading required package: coda

## Linked to JAGS 4.3.1

## Loaded modules: basemod,bugs

##
## Attaching package: 'R2jags'

## The following object is masked from 'package:coda':
##
##     traceplot
```

```
library(coda)
library(lattice)
library(MCMCvis)
library(tidyverse)
```

First, a data matrix is created for JAGS processing. The raw reaction times are extracted in their original wide format and log-transformed to meet model assumptions. A data list is then constructed, defining separate matrices for schizophrenic and non-schizophrenic reaction times, the number of patients in each group, and the number of trials. Defining these components at this stage is essential, as they will be referenced in the model definition.

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

data_list <- list(
  y.ns = log_rtimes[1:11, ],  # Non-schizophrenic reaction times (11 x 30)
  y.s = log_rtimes[12:17, ],  # Schizophrenic reaction times (6 x 30)
  N_ns = 11,                  # Number of non-schizophrenics
  N_s = 6,                    # Number of schizophrenics
  T = 30                      # Number of trials
)
```

Model Specification

Now we define the Hierarchical Bayesian model. The model is accounting for the two groups of patients, and their individual variability.

Reaction times for Non-Schizophrenic patients (N.S) are modeled as:

$(i = 1, 2, \dots, 11)$

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

Where:

- y_{ij}^{ns} is the **reaction time** for individual i and trial j
- α_i is individual specific **mean** reaction time
- σ_y^2 is the shared **varience** across all trials.

Individuals **mean** (α_i) reaction time is:

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

- μ is the mean reaction time for N.S - σ_α^2 is the intragroup variance in reaction times

Reaction times for Schizophrenic patients (S):

$$y_{ij} \sim N(\alpha_i + \tau z_{ij}, \sigma_y^2) \quad i = 12, 13, \dots, 17, \quad j = 1, 2, \dots, 30$$

Where:

- τ is the extra delay
- z_{ij} is an indicator (one or zero)

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

Schizophrenic patients can have delayed responses due to **attention deficit**. This is modelled through the Bernoulli distributed indicator function. λ is the probability of a trial being delayed.

Schizophrenics mean reaction time:

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

Where:

- β is the increase in mean reaction time for Schizophrenic patients

Defining the JAGS Model

```r
set.seed(123) # Setting Seed for reproducibility

jags.model <- function(){
    # Reaction time of non-schizophrenics
    for (i in 1:N_ns){
        for (j in 1:T) {
            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:N_s){
        for (j in 1:T) {
            z[i,j] ~ dbern(lambda)  # Bernoulli for delay indicator
            y.s[i,j] ~ dnorm(alpha.s[i] + tau * z[i,j], p.y)
            # Used ChatGPT  to bug fix indentation error.
        }
        alpha.s[i] ~ dnorm(mu + beta, p.alpha)
    }

    # Priors
    mu ~ dunif(-10, 10)
    sigma_y2 ~ dunif(0.001, 5)
    sigma_alpha2 ~ dunif(0.001, 5)
    beta ~ dunif(0, 10)
    tau ~ dunif(0, 10)
    lambda ~ dunif(0, 1)

    # Precision parameters
    p.y <- 1 / sigma_y2
    p.alpha <- 1 / sigma_alpha2
}
```

Defining the Parameters, what we what to track, and initial values

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

set.seed(123)

inits1 <- list(
    mu = 4, beta = 1, tau = 1, lambda = 0.5,
    sigma_y2 = 1, sigma_alpha2 = 1,
    z = matrix(0, nrow = 6, ncol = 30)
)
inits2 <- list(
    mu = 6, beta = 4, tau = 5, lambda = 0.3,
    sigma_y2 = 2, sigma_alpha2 = 2,
    z = matrix(0, nrow = 6, ncol = 30)
)
jags.inits <- list(inits1, inits2)
```

Two sets of initial values were chosen to help the model explore different starting points and ensure proper convergence. Parameters `mu`, `beta`, `tau`, `lambda`, `sigma_y2`, and `sigma_alpha2` were given different realistic values to cover a broad range. The `z` matrix, which indicates delayed responses, was initialized with zeros to start with the assumption of no delays, allowing the model to update this based on the data.

Fitting the JAGS Model

In the first JAGS model, we are running two MCMC chains, with **10,000** iterations and discarding the first **5000** iterations to view the convergence.

```
jags.mod.fit <- jags(data = data_list, 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)
```

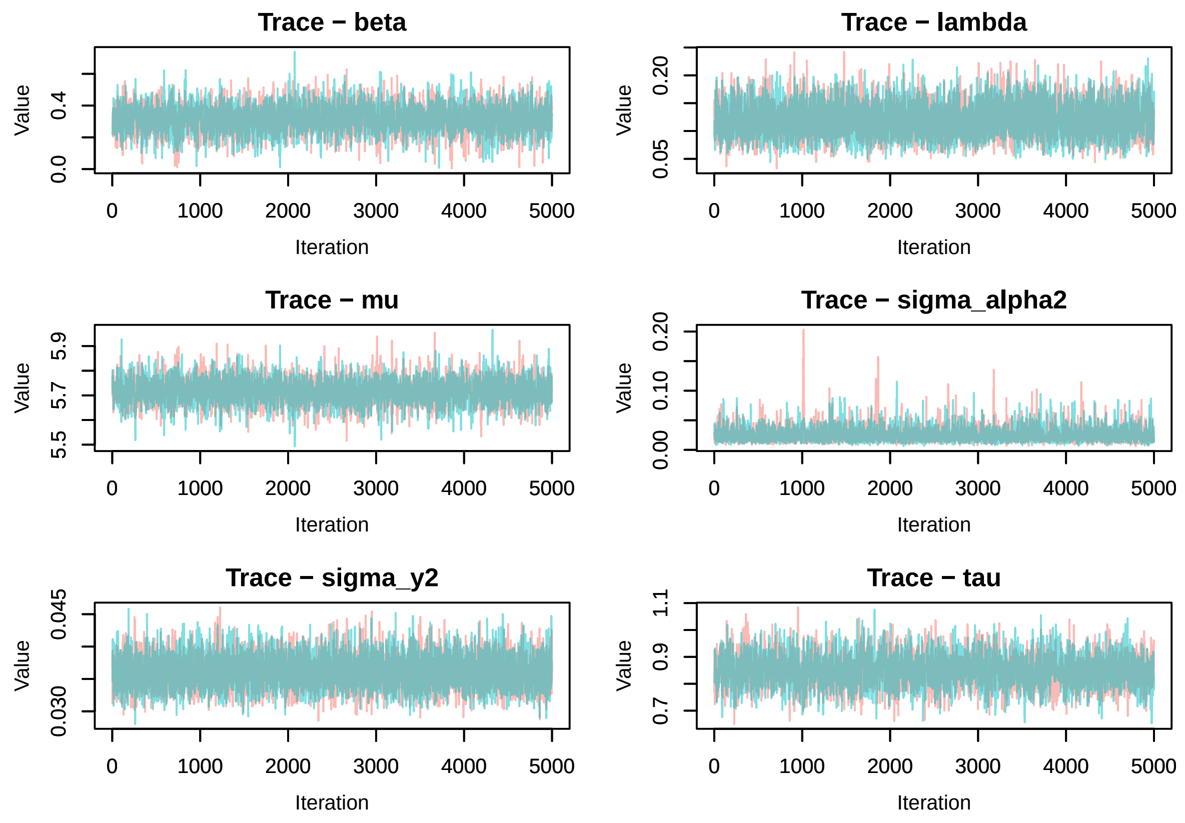
```
## module glm loaded
```

```
## module dic loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 510
##    Unobserved stochastic nodes: 203
##    Total graph size: 1085
##
## Initializing model
```

5. Monte Carlo Markov Chain

```
jagsfit.mcmc <- as.mcmc(jags.mod.fit)
 MCMCtrace(jagsfit.mcmc,type = 'trace',ind = TRUE, pdf = FALSE)
```



The trace plots for all key parameters (`beta`, `lambda`, `mu`, `sigma_alpha2`, `sigma_y2`, and `tau`) show good overall mixing and convergence. The parameters `beta`, `mu`, `tau`, and `sigma_y2` display well-mixed chains with stable fluctuations around a consistent mean, indicating effective exploration of the posterior distribution. The `lambda` trace plot shows slightly more fluctuation at the start but quickly stabilizes, suggesting successful convergence after the burn-in period.

Although `sigma_alpha2` shows a bit more variability with some occasional spikes, the chain still appears well-distributed and doesn't indicate any major convergence issues. Overall, the absence of clear trends or poor mixing across all parameters suggests that the **MCMC algorithm has converged effectively**.

To check for convergence more accurately, we can extract the **Gelman-Rubin** test statistic values. This is in the code below.

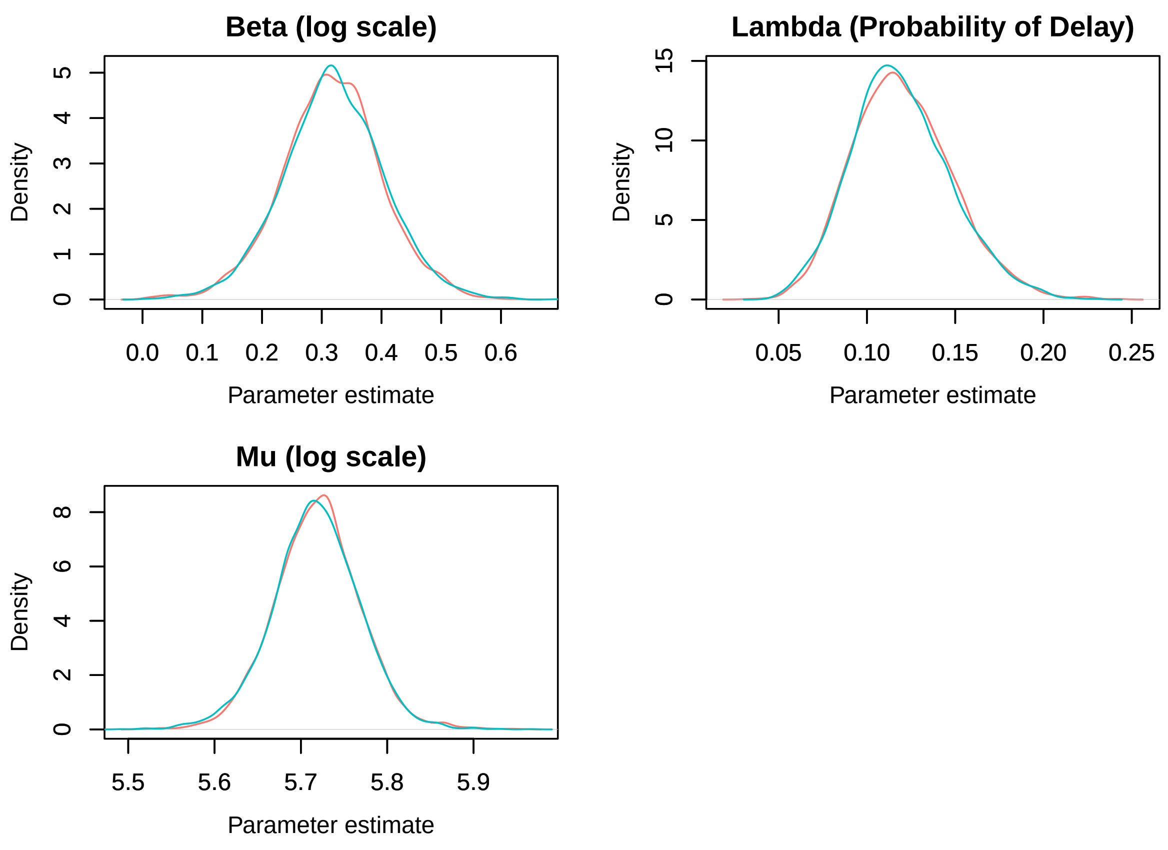
```
gelman_stats<- gelman.diag(jagsfit.mcmc,multivariate=FALSE)
gelman_df <- as.data.frame(gelman_stats$psrf)
gelmam.params <- c("mu", "beta", "tau", "lambda", "sigma_y2", "sigma_alpha2")
gelman_subset <- gelman_df[gelmam.params, , drop = FALSE]
```

```
## Potential scale reduction factors:
##
##                Point est. Upper C.I.
## beta                 1.00       1.00
## lambda               1.00       1.00
## mu                   1.00       1.00
## sigma_alpha2         1.01       1.01
## sigma_y2             1.00       1.00
## tau                  1.00       1.00
```

The Gelman-Rubin diagnostics show values close to 1 for all parameters, indicating good convergence. The slight deviation for `sigma_alpha2` (1.01) is minimal and not concerning. If a value was above 1.1, this would suggest a lack of convergence.

6. Posterior Distributions

```
library(MCMCvis)
MCMCtrace(jagsfit.mcmc,
          params = c("beta", "lambda", "mu"),
          type = "density",
          ind = TRUE,
          pdf = FALSE,
          main_den = c("Beta (log scale)",
                       "Lambda (Probability of Delay)",
                       "Mu (log scale)"))
```



Next, we check to summary statistics.

```
print(jags.mod.fit$BUGSoutput$summary[c("beta", "lambda", "mu"), ])
```

```
##              mean         sd       2.5%       25%       50%       75%     97.5%
## beta    0.3181462 0.08436591 0.14833526 0.2647345 0.3180889 0.3721589 0.4878396
## lambda  0.1198850 0.02818665 0.06948152 0.1001758 0.1178340 0.1378933 0.1801819
## mu      5.7183935 0.05033801 5.61767490 5.6864151 5.7187429 5.7507063 5.8162062
##             Rhat n.eff
## beta    1.001889  3000
## lambda  1.001403  3000
## mu      1.001504  2500
```

The **n.eff** statistics in the table above is a crude measurement for **effective sample size**, if the value is above 1000, it indicates that you have sample adequacy, enough independent samples for posterior inference. In our case, we can confidently assume that our model has plenty of valid samples shown by the n.eff values greater than 1000(**3000, 3000, 2500**).

To confirm this, the **MCMC Standard Error** should be within 1-5% of the posterior standard deviation, indicating reliable and stable estimates. This is calculated by dividing the Time-series Standard Error by the posterior Standard Deviation and expressing it as a percentage.

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

```
SampleCheck <- summary(jagsfit.mcmc)
stats <- SampleCheck$statistics

sd_values <- SampleCheck$statistics[, "SD"]
se_values <- SampleCheck$statistics[, "Time-series SE"]
```

```
MC_Error <- (se_values / sd_values) * 100
```

```
MC_Error[c("beta", "lambda", "tau")]
```

```
##     beta   lambda      tau
## 1.921942 1.667309 2.068671
```

The **MC errors** for Beta, Lambda and Tau range from **1.6**% to **2.1**%. This falls within the acceptable threshold, suggesting that the MCMC chains have **converged sufficiently**, and the number of iterations provided **reliable posterior estimates**.

Summary Statistics

Now we extract the relevant statistics for **Beta**, **Lamba** and **Tau**. It is important to note that Beta and Tau have been in log scale up until this point, so we multiply them by the exponential function.

This is done in the code below.

Beta

```
pos_beta<-substr(rownames(jags.mod.fit$BUGSoutput$summary),1,4)=='beta'
# Extract posterior mean for mu
beta_mean<-jags.mod.fit$BUGSoutput$summary[pos_beta,1]
# Extract posterior median
beta_median<-jags.mod.fit$BUGSoutput$summary[pos_beta,5]
# Extract 2.5 percentile of the posterior
beta2.5 <- jags.mod.fit$BUGSoutput$summary[pos_beta,3]
# Extract 97.5 percentile of the posterior
beta97.5 <- jags.mod.fit$BUGSoutput$summary[pos_beta,7]
```

Lambda

```
pos_lambda <- substr(rownames(jags.mod.fit$BUGSoutput$summary), 1, 6) == "lambda"
lambda_mean <- jags.mod.fit$BUGSoutput$summary[pos_lambda, 1]
lambda_median <- jags.mod.fit$BUGSoutput$summary[pos_lambda, 5]
lambda2.5 <- jags.mod.fit$BUGSoutput$summary[pos_lambda,3]
lambda97.5 <- jags.mod.fit$BUGSoutput$summary[pos_lambda,7]
```

Tau

```
pos_tau <- substr(rownames(jags.mod.fit$BUGSoutput$summary), 1, 3) == "tau"
tau_mean <- jags.mod.fit$BUGSoutput$summary[pos_tau, 1]
tau_median <- jags.mod.fit$BUGSoutput$summary[pos_tau, 5]
tau2.5 <- jags.mod.fit$BUGSoutput$summary[pos_tau,3]
tau97.5 <- jags.mod.fit$BUGSoutput$summary[pos_tau,7]
```

We add the summary statistics to a data frame.

```
df_stats <- data.frame(
  Parameter = c("Beta", "Lambda", "Tau"),
  Median = c(exp(beta_median), lambda_median, exp(tau_median)),
```

Posterior Summaries for Key Parameters

Parameter	Median	Lower 95% CI	Upper 95% CI
Beta	1.374	1.160	1.629
Lambda	0.118	0.069	0.180
Tau	2.333	2.089	2.632

```
  Lower_95_CI = c(exp(beta2.5), lambda2.5, exp(tau2.5)),
  Upper_95_CI = c(exp(beta97.5), lambda97.5, exp(tau97.5))
)
```

Based on the "**Posterior Summaries**" table, conclusions about schizophrenic versus non-schizophrenic reaction times emerge, aligning with **motor retardation and attention deficit theories**. The median β of **1.374** (95% CI: 1.169–1.626) indicates schizophrenic mean reaction times are **37**% longer, with the CI above **1** confirming significant slowing, as seen in **Person 14**'s wide histogram spread (Question 1a). This supports motor retardation across all trials.

The median λ of **0.118** (CI: 0.070–0.181) suggests **11.8**% of trials are delayed (**7.0%–18.1%**), consistent with attention deficits, matching mixed fast/slow times in Person 15.

The median τ of **2.337** (CI: 2.082–2.643) shows delayed responses are **2.34** times longer, reinforcing significant delays.

Compared to the tighter distributions seen in non-schizophrenic individuals, such as **Person 1**, the combined effects of β, τ, λ among schizophrenics validate the model's two-component structure, capturing both **consistent and delayed response patterns**.

7. Preditions

A. Predicting Additional Reaction Times

To evaluate the model's predictive performance, additional reaction times should be simulated for each schizophrenic individual. This is done by extending the model to **generate predicted values** (`y_pred`) while accounting for the possibility of independant delayed responses through the use of an indicator variable (`z_pred`).

```
set.seed(123) # Setting Seed for reproducibility

jags.model2 <- function(){
 for (i in 1:N_ns){
        for (j in 1:T) {
            y.ns[i,j] ~ dnorm(alpha.ns[i], p.y)
        }
        alpha.ns[i] ~ dnorm(mu, p.alpha)
    }
     for (i in 1:N_s){
        for (j in 1:T) {
            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)
    }
```

```
     # Prediction for additional response times for schizophrenic
      #individuals i = 12 to 17
      for (i in 1:N_s) {
        for (j in 1:T) {
          # Predicted response times
            y.pred[i,j] ~ dnorm(alpha.s[i] + tau * z.pred[i,j], p.y)
            z.pred[i,j] ~ dbern(lambda)
        }
    }

    # Priors
    mu ~ dunif(-10, 10)
    sigma_y2 ~ dunif(0.001, 5)
    sigma_alpha2 ~ dunif(0.001, 5)
    beta ~ dunif(0, 10)
    tau ~ dunif(0, 10)
    lambda ~ dunif(0, 1)

    # Precision parameters
    p.y <- 1 / sigma_y2
    p.alpha <- 1 / sigma_alpha2
}
```

`y_pred` represents the predicted log-transformed reaction times. We use z_pred instead of z, as it ensures that the delayed reaction times are **independent** from the observed information. Even with the same initial values, this extra step can reduce bias within the model.

B.

To assess the variability in the predicted reaction times, the model includes nodes to track the **standard deviations of the 30 predicted measurements for each schizophrenic individual**. Specifically, `sd_pred[i]` calculates the standard deviation of the 30 predicted reaction times for individual `i`, using the formula:

sd_pred[i] <- sd(y_pred[i,])

This allows the model to capture **within-individual variability** in the predictions.

Additionally, to summarise this variability across all individuals, two nodes were defined to track the minimum and maximum standard deviations:

- `Smin` captures the smallest standard deviation across all individuals:
- `Smax` captures the largest standard deviation across all individuals:

These values (`Smin` and `Smax`) provide a summary of the range of variability in the predicted reaction times, which will be used to compare against the observed variability in the original data.

```
set.seed(123) # Setting Seed for reproducibility

jags.model2 <- function(){
 for (i in 1:N_ns){
        for (j in 1:T) {
            y.ns[i,j] ~ dnorm(alpha.ns[i], p.y)
        }
        alpha.ns[i] ~ dnorm(mu, p.alpha)
    }
     for (i in 1:N_s){
        for (j in 1:T) {
            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)
    }

     # Prediction for additional response times for schizophrenic
      #individuals i = 12 to 17
      for (i in 1:N_s) {
          for (j in 1:T) {
          # Predicted response times
            y.pred[i,j] ~ dnorm(alpha.s[i] + tau * z.pred[i,j], p.y)
            z.pred[i,j] ~ dbern(lambda)
          }
                sd.pred[i] <- sd(y.pred[i, ])
    }
    # Compute min and max of standard deviations
    Smin <- min(sd.pred[])
    Smax <- max(sd.pred[])

    # Priors
    mu ~ dunif(-10, 10)
    sigma_y2 ~ dunif(0.001, 5)
    sigma_alpha2 ~ dunif(0.001, 5)
    beta ~ dunif(0, 10)
    tau ~ dunif(0, 10)
    lambda ~ dunif(0, 1)

    # Precision parameters
    p.y <- 1 / sigma_y2
    p.alpha <- 1 / sigma_alpha2
}
```

C. Fitting the Extended Model

```
jags.param2 <- c("mu", "beta", "tau", "lambda", "p.y", "p.alpha", "Smin", "Smax")

set.seed(123)

inits1 <- list(
    mu = 4, beta = 1, tau = 1, lambda = 0.5,
    sigma_y2 = 1, sigma_alpha2 = 1,
    z = matrix(0, nrow = 6, ncol = 30),
    z.pred = matrix(0, nrow = 6, ncol = 30)
)
```

```
inits2 <- list(
    mu = 6, beta = 4, tau = 5, lambda = 0.3,
    sigma_y2 = 2, sigma_alpha2 = 2,
    z = matrix(0, nrow = 6, ncol = 30),
    z.pred = matrix(0, nrow = 6, ncol = 30)
)
jags.inits <- list(inits1, inits2)
```

The extended model, which includes the prediction of additional reaction times and the tracking of standard deviations (`sd_pred`, `Smin`, and `Smax`), is fitted using two MCMC chains. The model ran for **6,000** iterations, discarding the first **5,000** as burn-in to ensure convergence and reliable posterior samples.

The same initial values and prior distributions from the original model were used to maintain consistency, with additional initial values specified for the new `z_pred` matrix. This matrix was initialized with zeros, allowing the model to determine the occurrence of delays in the predictions based on the posterior estimate of `lambda`.

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

```
# Display results
print(jags.mod.fit2)
```

```
## Inference for Bugs model at "C:/Users/james/AppData/Local/Temp/RtmpyMOiTS/model5cfc41f01c
##  2 chains, each with 6000 iterations (first 5000 discarded)
##  n.sims = 2000 iterations saved. Running time = 6.46 secs
##         mu.vect sd.vect   2.5%    25%    50%    75%  97.5% Rhat n.eff
## Smax      0.401   0.042  0.320  0.373  0.400  0.430  0.483 1.002  1400
## Smin      0.251   0.047  0.161  0.219  0.253  0.285  0.339 1.002  2000
## beta      0.318   0.082  0.156  0.267  0.318  0.369  0.496 1.001  2000
## lambda    0.122   0.029  0.071  0.102  0.119  0.140  0.183 1.001  2000
## mu        5.717   0.048  5.618  5.688  5.717  5.747  5.817 1.002  2000
## p.alpha  46.997  20.830 17.138 32.201 43.371 58.282 98.884 1.006   380
## p.y      27.878   1.865 24.316 26.543 27.907 29.141 31.462 1.001  2000
## tau       0.843   0.059  0.735  0.802  0.841  0.882  0.964 1.002  1100
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
```

D. Interpreting Predicted Variability

To assess the model's ability to capture variability in predicted reaction times, the minimum and maximum standard deviations (`Smin` and `Smax`) were extracted from the MCMC samples of the fitted model.

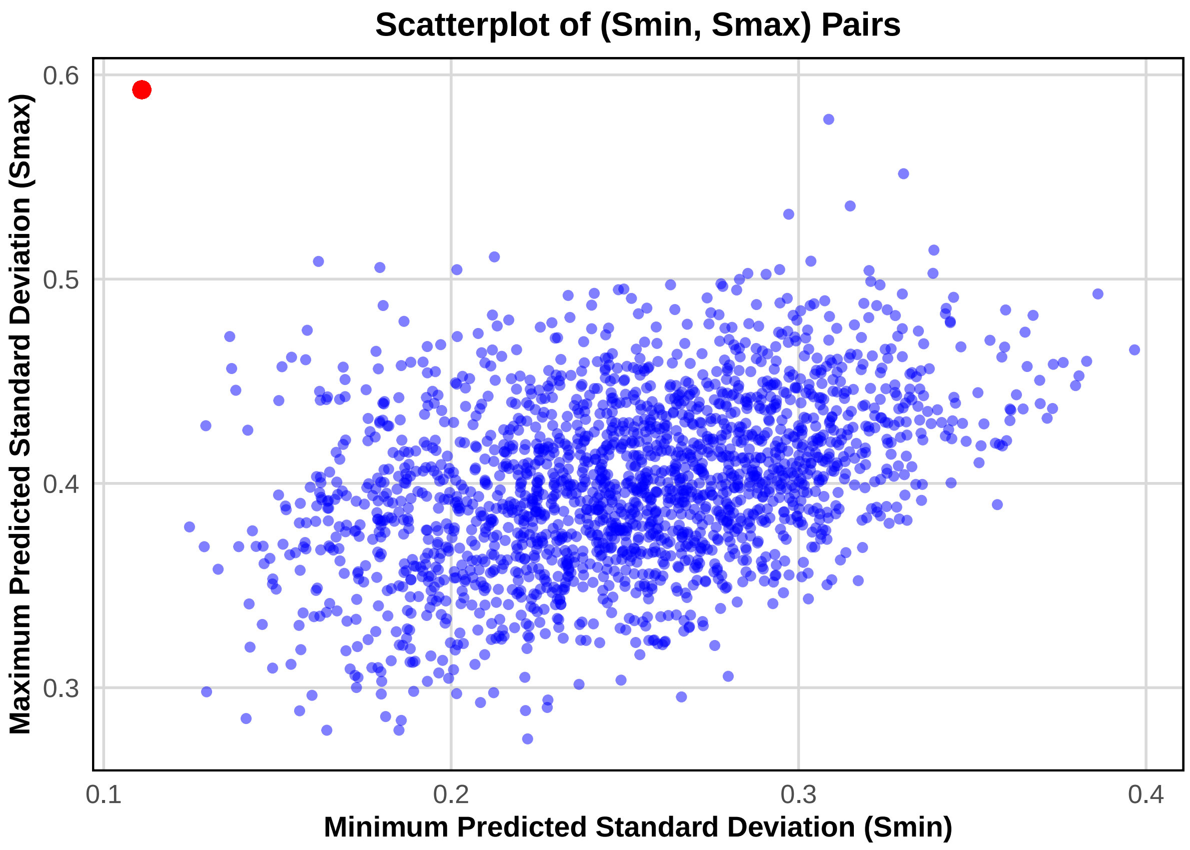
```
# Extract Smin and Smax from MCMC samples
smin_samples <- jags.mod.fit2$BUGSoutput$sims.list$Smin
smax_samples <- jags.mod.fit2$BUGSoutput$sims.list$Smax

scatter_data <- data.frame(Smin = smin_samples, Smax = smax_samples)
```

```r
# Calculate raw minimum and maximum standard deviation values from Question 2
raw_min_sd <- min(sd_log$SD_Log_RT[12:17])  # Minimum SD for schizophrenic group
raw_max_sd <- max(sd_log$SD_Log_RT[12:17])  # Maximum SD for schizophrenic group

# Create scatter plot
ggplot(scatter_data, aes(x = Smin, y = Smax)) +
  geom_point(alpha = 0.5, color = "blue") +  # MCMC Samples
  geom_point(aes(x = raw_min_sd, y = raw_max_sd), color = "red", size = 3) +
  labs(
    title = "Scatterplot of (Smin, Smax) Pairs",
    x = "Minimum Predicted Standard Deviation (Smin)",
    y = "Maximum Predicted Standard Deviation (Smax)"
  ) +
  custom_theme
```

```
## Warning in geom_point(aes(x = raw_min_sd, y = raw_max_sd), color = "red", : All aesthetic
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
```



The scatterplot above displays the relationship between the predicted minimum (`Smin`) and maximum (`Smax`) standard deviations of schizophrenic individuals, obtained from the posterior samples of the JAGS model. Each **blue point** represents an (`Smin`, `Smax`) pair from a single MCMC iteration, while the **red point** corresponds to the minimum and maximum standard deviations calculated from the observed data.

Most of the posterior samples (blue points) form a dense cluster, indicating the range of predicted within-person variability. However, the red point lies outside this distribution, suggesting that the model may underestimate the true variability observed in the original data.

Conclusion

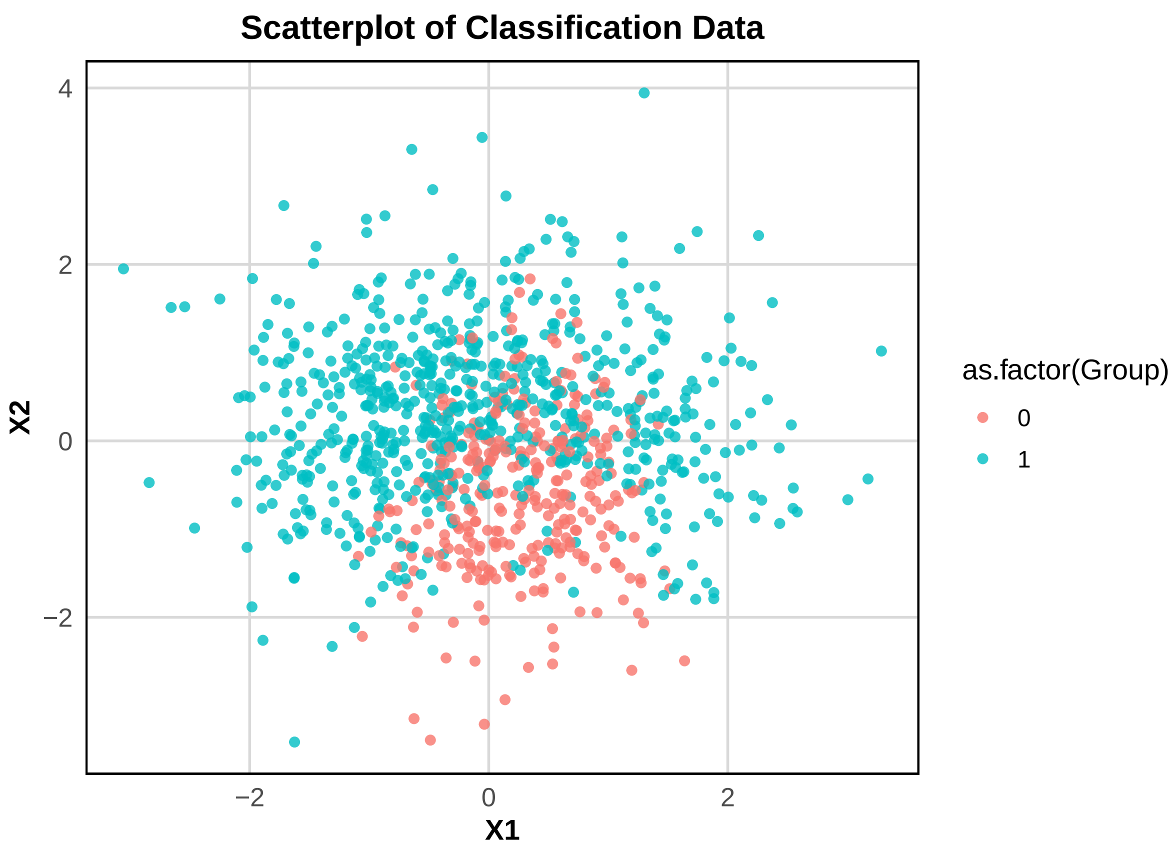
The position of the red point outside the cloud of posterior samples suggests that the model **does not** fully explain the observed variability in schizophrenic reaction times. While the model captures general trends, it appears to **underestimate** the true dispersion of reaction times.

However, this conclusion should be treated with caution. With only a single point of comparison, it is difficult to confidently assess the model's accuracy. If a larger dataset were available—such as 1,000 observed standard deviation pairs—it is possible that the observed distribution would align more closely with the model's predictions. Therefore, while the current results suggest some underestimation, more observed data would be needed to validate this conclusively.

B. Classification

The following figure shows the information in the dataset Classification.csv. It shows two different groups, plotted with two explanatory variables. This is simulated data - the groupings are determined by a known function of **X1** and **X2** with added **noise/random error**.

```r
ggplot(Classification, aes(x = X1, y = X2, color = as.factor(Group))) +
  geom_point(alpha = 0.8) +
  labs(
    title = "Scatterplot of Classification Data",
    x = "X1",
    y = "X2"
  ) +
  custom_theme
```



1. Numerical Summary

First, we use the summary function to get a brief overview of the dataset. We can see that the X1 and X2 are the explanatory variables. To understand how the groups are classified, we will create a more detailed numerical summary.

```r
summary(Classification)
```

```r
# Load libraries
library(gt)
library(dplyr)
```

```r
# Create separate summaries for X1 and X2
classification_summary_X1 <- Classification %>%
  group_by(Group) %>%
  summarise(
    `Mean X1` = mean(X1),
    `SD X1` = sd(X1),
    `Median X1` = median(X1),
    `Min X1` = min(X1),
    `Max X1` = max(X1)
  ) %>%
  mutate(Group = ifelse(Group == 0, "Group 0", "Group 1")) %>%
  relocate(Group)

classification_summary_X2 <- Classification %>%
  group_by(Group) %>%
  summarise(
    `Mean X2` = mean(X2),
    `SD X2` = sd(X2),
    `Median X2` = median(X2),
    `Min X2` = min(X2),
    `Max X2` = max(X2)
  ) %>%
  mutate(Group = ifelse(Group == 0, "Group 0", "Group 1")) %>%
  relocate(Group)
```

```r
# Code to create Summary stat table

# GT table for X1
gt_table_X1 <- classification_summary_X1 %>%
  gt() %>%
  tab_header(
    title = md("**Summary Statistics for Group 0 and Group 1**"),
    subtitle = md("*Comparison of X1 Across Groups*")
  ) %>%
  fmt_number(
    columns = where(is.numeric),
    decimals = 2
  ) %>%
  tab_options(
    table.font.size = px(10),
    table.width = pct(90),
    heading.align = "center",
    column_labels.font.weight = "bold",
    column_labels.border.bottom.width = px(1),
    column_labels.border.bottom.color = "black",
    table.border.top.width = px(1),
    table.border.top.color = "black",
    table.border.bottom.width = px(1),
    table.border.bottom.color = "black",
    data_row.padding = px(4)
  ) %>%
  cols_align(
    align = "center", columns = where(is.numeric)
  ) %>%
  opt_table_lines() %>%
  opt_row_striping()

# GT table for X2
gt_table_X2 <- classification_summary_X2 %>%
  gt() %>%
  tab_header(
    title = md("**Summary Statistics for Group 0 and Group 1**"),
    subtitle = md("*Comparison of X2 Across Groups*")
  ) %>%
  fmt_number(
    columns = where(is.numeric),
    decimals = 2
  ) %>%
  tab_options(
    table.font.size = px(10),
    table.width = pct(90),
    heading.align = "center",
    column_labels.font.weight = "bold",
    column_labels.border.bottom.width = px(1),
    column_labels.border.bottom.color = "black",
    table.border.top.width = px(1),
    table.border.top.color = "black",
    table.border.bottom.width = px(1),
    table.border.bottom.color = "black",
    data_row.padding = px(4)
  ) %>%
  cols_align(
    align = "center", columns = where(is.numeric)
  ) %>%
  opt_table_lines() %>%
  opt_row_striping()

# Used ChatGPT here to suggest improvements on how to improve the tables formatting.
```

Summary Statistics for Group 0 and Group 1

Comparison of X1 Across Groups

Group	Mean X1	SD X1	Median X1	Min X1	Max X1
Group 0	0.26	0.54	0.26	-1.09	1.64
Group 1	-0.12	1.12	-0.30	-3.06	3.29

Summary Statistics for Group 0 and Group 1

Comparison of X2 Across Groups

Group	Mean X2	SD X2	Median X2	Min X2	Max X2
Group 0	-0.60	0.89	-0.60	-3.39	1.84
Group 1	0.30	0.93	0.31	-3.41	3.94

Numerical Summaries and Interpretation

Variable X1:

- **Mean and Variability:** Group 1 shows higher variability in X1 with a standard deviation of **1.12**, compared to **0.59** for Group 0. This suggests that Group 1 has greater dispersion around its mean.
- **Median and Range**: The median for Group 1 is **-0.30**, which is lower than Group 0's median of **0.06**, indicating a shift towards lower values for Group 1. Additionally, Group 1 exhibits a broader range (**-3.06 to 3.29**) compared to Group 0 (**-1.09 to 1.64**), suggesting more extreme values and greater variability.

Variable X2

X2 shows greater similarity between the two groups compared to X1:

- **Central Tendency**: The median value for Group 1 is **0.31**, slightly higher than Group 0's median of **-0.05**, suggesting that Group 1 tends to have higher X2 values.
- **Spread and Range:** Both groups display similar variability in terms of standard deviation (**0.93** for Group 1 vs. **0.89** for Group 0). However, Group 1 again exhibits a wider range (**-3.41 to 3.94**) compared to Group 0 (**-3.39 to 1.84**), indicating that Group 1 includes more extreme values.

Implications for Classification

The observed numerical summaries and scatter plot highlight several key points:

- The differences in distributions and notable overlap suggest that the classes are not linearly separable.
- The variability, particularly in X1, suggests complex interactions or non-linear boundaries between groups.

Recommended Classification Methods

Given the distribution and structure of the data, suitable classification methods include:

1. **K-Nearest Neighbors (KNN):**
   - Flexible in handling non-linear boundaries.
   - The optimal choice of neighbors (K) can be tuned to balance flexibility and overfitting risks.

2. **Quadratic Discriminant Analysis (QDA):**
    - Effective at modeling non-linear class boundaries if the assumption of normality holds.
3. **Random Forest (RF):**
    - Capable of capturing complex interactions and nonlinear patterns in the data without strong assumptions.
4. **Support Vector Machines (SVM):**
    - Suitable for data where clear linear boundaries do not exist, leveraging kernels to create non-linear separations.

Linear Discriminant Analysis (LDA) is not ideal here as it assumes linear boundaries and equal covariance across groups, which is unsuitable for this dataset given the observed overlap and variability.

2. Training and Test Data

```r
library(dplyr)
 library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.3
```

```r
 library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.4.3
```

```r
Classification <- Classification %>%
dplyr::select(Group, X1, X2) %>% # Explicitly calling dplyr here as I was
  # getting a package clash error. ChatGPT helped bug fix this.
  mutate(Group = as.factor(Group))

y <- Classification[, "Group"]
x <- Classification[, c("X1", "X2")]

set.seed(123) # Setting seed ensures reproducible splits

# 75% training data
ii <- createDataPartition(y, p = 0.75, list = FALSE)

# Creating train and test sets
xTrain <- x[ii, ]
yTrain <- y[ii]

xTest <- x[-ii, ]
yTest <- y[-ii]

# Final training dataframe
df <- data.frame(xTrain,yTrain)
```

The dataset was split into **training and test subsets** to support model evaluation. The data was randomly partitioned with **75%** allocated for training and **25%** for testing. A fixed random seed (set.seed(123) is used to ensure reproducibility. The training data is used to fit the models, while the test data provides an unbiased evaluation of model performance.

3. Classification Methods

To determine the best classification method, it's crucial to evaluate and compare the performance, strengths, and weaknesses of each method applied: K-Nearest Neighbors (KNN), Quadratic Discriminant Analysis (QDA), Random Forest (RF), and Support Vector Machines (SVM).

3.1 K-Nearest Neighbors (KNN):

```
library(class) # This is the package which has the KNN function
```

The **KNN classifier** is a supervised, non- parametric classification algorithm. It classifies a new observation by examining the classes of its **k** nearest neighbours, and assigning the class by through a "majority vote" (most common class of neighbours).

The value of k controls the **flexibility** of the model:

- Smaller k leads to a flexible decision boundary (low bias but high variance, prone to overfitting).
- Larger k smooths the decision boundary (high bias but lower variance, risk underfitting).

The goal is to select an **optimal k** that balances bias and variance, typically done via cross-validation.

Fitting the model

```
knn_fit1 = knn(train=xTrain, test=xTest, cl=yTrain, k=7)

# Confusion Matrix
confusionMatrix(knn_fit1, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  50  17
##          1  29 153
##
##                Accuracy : 0.8153
##                  95% CI : (0.7614, 0.8615)
##     No Information Rate : 0.6827
##     P-Value [Acc > NIR] : 1.802e-06
##
##                   Kappa : 0.5555
##
##  Mcnemar's Test P-Value : 0.1048
##
##             Sensitivity : 0.6329
##             Specificity : 0.9000
##          Pos Pred Value : 0.7463
##          Neg Pred Value : 0.8407
##              Prevalence : 0.3173
##          Detection Rate : 0.2008
##    Detection Prevalence : 0.2691
##       Balanced Accuracy : 0.7665
##
##        'Positive' Class : 0
##
```

Initial testing with an arbitrary choice of $\mathbf{k = 7}$, resulted in an accuracy of **81.53**%. Although this initial accuracy was good, further hyperparameter tuning is necessary to optimise model performance.

Hyperparameters Optimisation

Hyperparameter tuning improves the structure of a machine learning model to balance flexibility and overfitting. For KNN, this involves selecting the optimal value of **k**.

A **10-fold cross-validation, repeated 5 times**, is used to estimate prediction error and identify the optimal k value. This method ensures robust performance evaluation by averaging results across multiple partitions.

We use the train() function here which does the fitting and tuning simultaneously.

```r
set.seed(123)
# Set training options
# Repeat 5-fold cross-validation, ten times
opts <- trainControl(method='repeatedcv', number=10, repeats=5)
# Find optimal k (model)
mdl <- train(x=xTrain, y=yTrain, # training data
method='knn', # machine learning model
trControl=opts, # training options
tuneGrid=data.frame(k=seq(4, 30))) # range of k's to try
print(mdl) # print the outcome
```

```
## k-Nearest Neighbors
##
## 751 samples
##   2 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 676, 675, 676, 676, 676, 676, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    4  0.7901716  0.5093445
##    5  0.7989361  0.5250251
##    6  0.8037289  0.5348628
##    7  0.8120280  0.5545077
##    8  0.8133616  0.5593923
##    9  0.8162704  0.5612139
##   10  0.8175757  0.5666458
##   11  0.8207754  0.5704364
##   12  0.8138419  0.5550297
##   13  0.8162315  0.5592418
##   14  0.8180664  0.5649665
##   15  0.8143610  0.5567516
##   16  0.8156946  0.5608058
##   17  0.8188947  0.5681544
##   18  0.8167507  0.5634111
##   19  0.8183402  0.5664944
##   20  0.8191507  0.5691983
##   21  0.8194246  0.5702634
##   22  0.8146419  0.5603220
##   23  0.8143752  0.5589839
##   24  0.8181085  0.5672467
##   25  0.8149085  0.5593909
##   26  0.8159680  0.5636286
##   27  0.8154243  0.5607340
##   28  0.8175610  0.5665536
##   29  0.8154241  0.5617290
##   30  0.8156766  0.5635613
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 11.
```

Cross-validation identified an optimal value of **k = 11**, slightly increasing accuracy.

Optimised KNN Model

```
set.seed(123)
knn_fit2 = knn(train=xTrain, test=xTest, cl=yTrain, k=11)
# Confusion Matrix
confusionMatrix(knn_fit2, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  49  14
##          1  30 156
##
##                Accuracy : 0.8233
##                  95% CI : (0.7701, 0.8686)
##     No Information Rate : 0.6827
##     P-Value [Acc > NIR] : 3.946e-07
##
##                   Kappa : 0.5687
##
##  Mcnemar's Test P-Value : 0.02374
##
##             Sensitivity : 0.6203
##             Specificity : 0.9176
##          Pos Pred Value : 0.7778
##          Neg Pred Value : 0.8387
##              Prevalence : 0.3173
##          Detection Rate : 0.1968
##    Detection Prevalence : 0.2530
##       Balanced Accuracy : 0.7690
##
##        'Positive' Class : 0
##
```

Interpretation of Results

The optimized KNN model achieved an overall accuracy of **82.3%**, with high specificity (**0.917**), effectively minimizing false positives. However, sensitivity (**0.62**) was lower, indicating limitations in correctly identifying positive instances. This trade-off is important in contexts where false negatives carry significant consequences.

3.2 Quadratic Discriminant Analysis (QDA):

Quadratic Discriminant Analysis (QDA) is a classification method that extends Linear Discriminant Analysis (LDA) by allowing each class to have its own covariance matrix. Like LDA, QDA assumes observations within each class are generated from a multivariate Gaussian distribution; however, unlike LDA, QDA does not assume these covariance matrices to be equal across classes. Due to this flexibility, QDA produces **quadratic decision boundaries**, enabling it to capture more complex, non-linear separations between classes effectively. Consequently, QDA is particularly suitable when the classes exhibit differing covariance structures, as it can adapt to variability within individual classes to achieve improved classification accuracy.

Fitting the Model

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
set.seed(123)
# Fitting QDA model
qda_fit1 <- qda(yTrain ~ ., data=xTrain)

# Predicting classes using fitted model
qda_predict <- predict(qda_fit1,xTest)
confusionMatrix(qda_predict$class, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  42  10
##          1  37 160
##
##                Accuracy : 0.8112
##                  95% CI : (0.757, 0.8579)
##     No Information Rate : 0.6827
##     P-Value [Acc > NIR] : 3.703e-06
##
##                   Kappa : 0.5204
##
##  Mcnemar's Test P-Value : 0.0001491
##
##             Sensitivity : 0.5316
##             Specificity : 0.9412
##          Pos Pred Value : 0.8077
##          Neg Pred Value : 0.8122
##              Prevalence : 0.3173
##          Detection Rate : 0.1687
##    Detection Prevalence : 0.2088
##       Balanced Accuracy : 0.7364
##
##        'Positive' Class : 0
##
```

No hyperparameter tuning is required for QDA, as it does not involve adjustable parameters.

Interpretation of Results

QDA achieved an accuracy of 81.1%, with high specificity (94%), making it reliable for correctly identifying negative cases. However, its lower sensitivity (53%) indicates limitations in detecting positive instances, which could be a concern in contexts where false negatives are costly.

3.3 Random Forest (RF):

Random Forests (RF) is an ensemble learning method that enhances classification accuracy by constructing multiple decision trees and aggregating their predictions. It extends **bagging** by de-correlating the trees, reducing model variance and improving generalisation.

Bagging: RF builds multiple trees using **bootstrap samples** (sampling with replacement) from the original dataset. Each tree is unpruned, reducing variance through averaging. Predictions are aggregated using majority voting for classification.

Random Feature Selection: To reduce correlation between trees, each split considers only a random subset of predictors (typically $\sqrt{p}$). This **de-correlates** trees, enhancing model robustness and lowering variance.

Out-of-Bag (OOB) Error: On average, one-third of observations are left out of each bootstrap sample. These "out-of-bag" observations provide a natural method for estimating model error without a separate validation set.

Fitting the model

```r
library(randomForest)

# Fitting random forest model
set.seed(123)

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

```
## Random Forest
##
## 751 samples
##   2 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 751, 751, 751, 751, 751, 751, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.786498  0.500471
##
## Tuning parameter 'mtry' was held constant at a value of 2
```

Optimised RF Model

Further optimization by adjusting the number of trees and mtry yielded minimal improvements. Therefore, the initial model with **200 trees** and **mtry = 2** was retained.

```
set.seed(123)
# Test model on testing data
TreePred <- predict(mdl, newdata=xTest)
confusionMatrix(TreePred, yTest) # predicted/true
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  49  21
##          1  30 149
##
##                Accuracy : 0.7952
##                  95% CI : (0.7397, 0.8435)
##     No Information Rate : 0.6827
##     P-Value [Acc > NIR] : 5.158e-05
##
##                   Kappa : 0.5123
##
##  Mcnemar's Test P-Value : 0.2626
##
##             Sensitivity : 0.6203
##             Specificity : 0.8765
##          Pos Pred Value : 0.7000
##          Neg Pred Value : 0.8324
##              Prevalence : 0.3173
##          Detection Rate : 0.1968
##    Detection Prevalence : 0.2811
##       Balanced Accuracy : 0.7484
##
##        'Positive' Class : 0
##
```

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

```
##     Overall
## X1 178.7594
## X2 146.9377
```

Variable Importance

Variable importance, assessed by the mean decrease in **Gini index**, indicated that X1 (**178.75**) contributed more significantly to the classification compared to X2 (**146.93**), suggesting that X1 is the most influential predictor.

Interpretation of Results

Random Forest provides good accuracy (**79.5**%) and insightful interpretability, highlighting predictor X1 as particularly influential. While accuracy and specificity were good, its lower sensitivity compared to KNN and QDA suggests potential limitations in classifying positive cases. RF's interpretability via variable importance remains valuable for understanding the predictive dynamics.

3.4 Support Vector Machines (SVM):

Support Vector Machines (SVM) are supervised classification algorithms designed to find the **optimal hyperplane** that best separates data points belonging to different classes. SVM seeks the hyperplane that maximizes the margin between groups 0 and 1 while minimizing classification errors.

When data cannot be separated linearly, SVM uses **kernel functions** to map the data into a higher-dimensional space where a linear separator can be found. This is known as the kernel trick, which allows SVM to handle complex, non-linear relationships efficiently without explicitly increasing dimensionality. Common kernel types include **linear, polynomial, and radial basis function (RBF)**. The choice of kernel is a hyperparameter that can significantly affect model performance and is typically optimized through cross-validation.

Since the data exhibits non-linear relationships, linear kernels are excluded. Instead, Radial Basis Function (RBF), Polynomial, and Sigmoid kernels will be evaluated to determine the most suitable option.

Fitting the model

1. **RFB Kernal**

The RBF kernel is chosen first due to its flexibility and ability to handle non-linear boundaries. We will also perform hyperparamater tuning, using 5-fold cross-validation method.

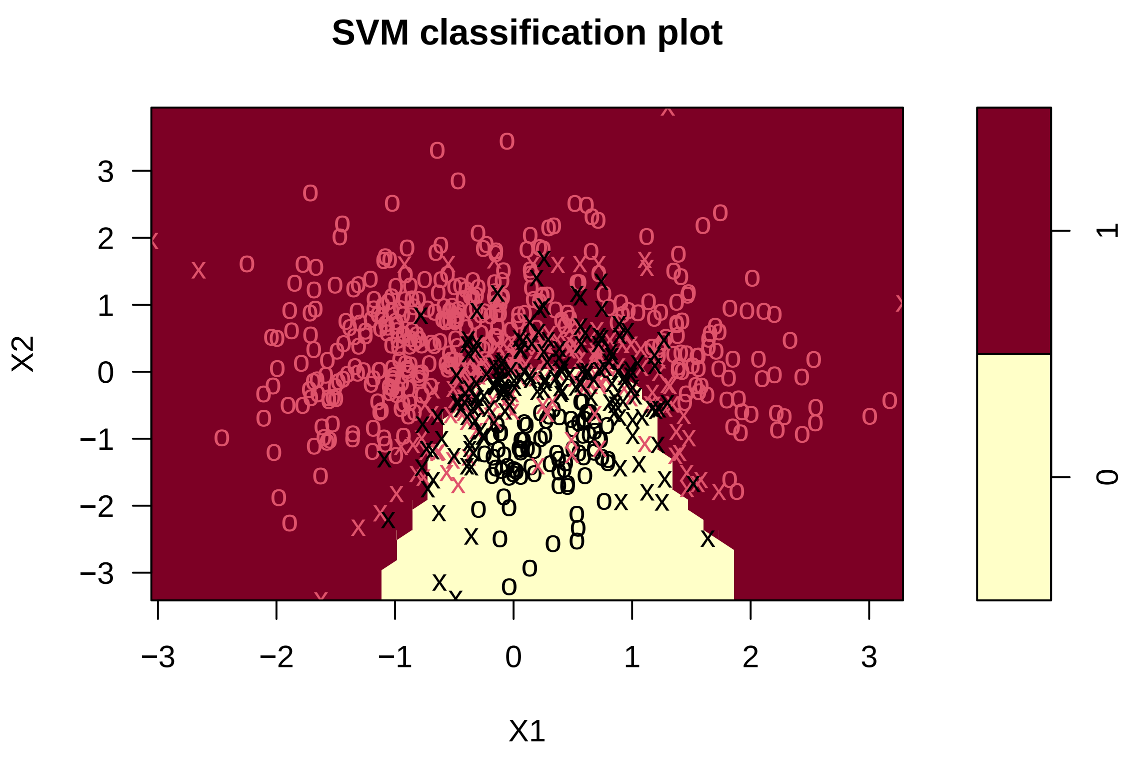
```
library(kernlab)
set.seed(123)

mdl_rad <- svm(yTrain~., data = df, kernel = "radial")
```

```r
print(mdl_rad)
```

```
##
## Call:
## svm(formula = yTrain ~ ., data = df, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  325
```

```r
plot(mdl_rad, df, X2 ~ X1)
```



```r
yTestPred_rad <- predict(mdl_rad, newdata=xTest)
confusionMatrix(yTestPred_rad, yTest) # predicted/true
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  48  13
##          1  31 157
##
##                Accuracy : 0.8233
##                  95% CI : (0.7701, 0.8686)
##     No Information Rate : 0.6827
##     P-Value [Acc > NIR] : 3.946e-07
##
##                   Kappa : 0.5656
##
##  Mcnemar's Test P-Value : 0.01038
##
##             Sensitivity : 0.6076
##             Specificity : 0.9235
##          Pos Pred Value : 0.7869
##          Neg Pred Value : 0.8351
##              Prevalence : 0.3173
##          Detection Rate : 0.1928
##    Detection Prevalence : 0.2450
##       Balanced Accuracy : 0.7656
##
##        'Positive' Class : 0
##
```

2. **Polynomial Kernal**

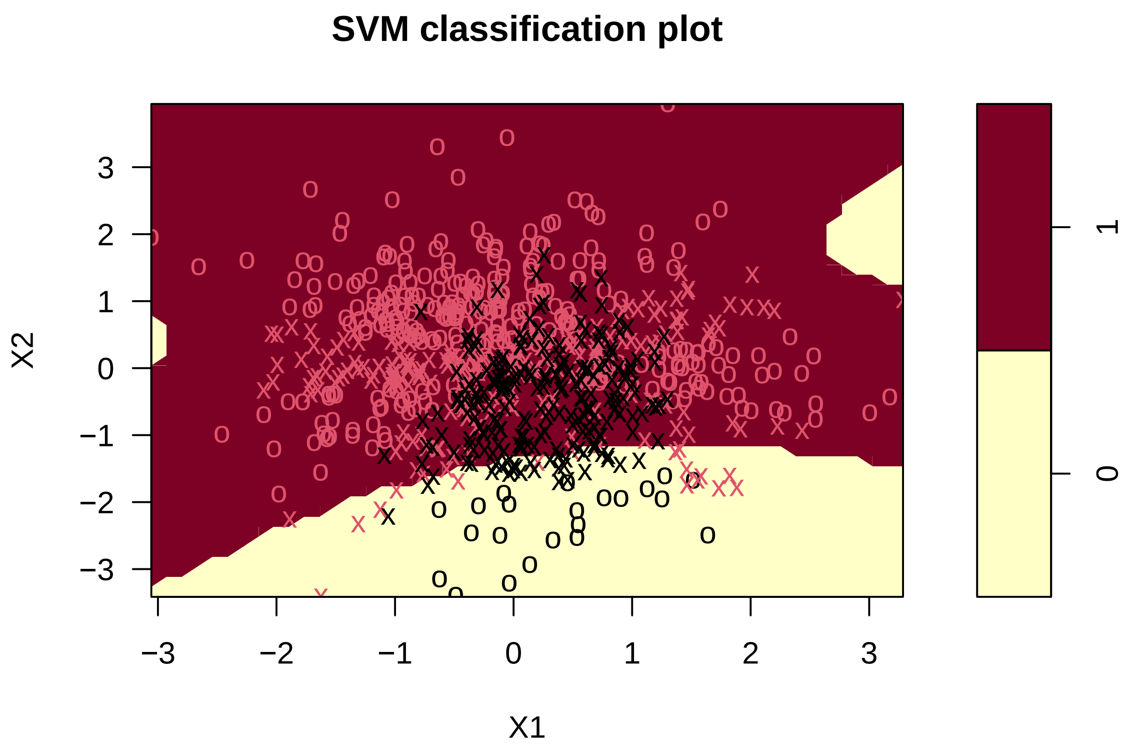
```r
set.seed(123)

mdl_poly <- svm(yTrain~., data = df, kernel = "polynomial")

print(mdl_poly)
```

```
##
## Call:
## svm(formula = yTrain ~ ., data = df, kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  435
```

```r
plot(mdl_poly, df, X2 ~ X1)
```



```r
yTestPred_poly <- predict(mdl_poly, newdata=xTest)
confusionMatrix(yTestPred_poly, yTest) # predicted/true
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  11   4
##          1  68 166
##
##                Accuracy : 0.7108
##                  95% CI : (0.6502, 0.7663)
##     No Information Rate : 0.6827
##     P-Value [Acc > NIR] : 0.1886
##
##                   Kappa : 0.1477
##
##  Mcnemar's Test P-Value : 1.131e-13
##
##             Sensitivity : 0.13924
##             Specificity : 0.97647
##          Pos Pred Value : 0.73333
##          Neg Pred Value : 0.70940
##              Prevalence : 0.31727
##          Detection Rate : 0.04418
##    Detection Prevalence : 0.06024
##       Balanced Accuracy : 0.55786
##
```

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

3. **Sigmoid Kernel**

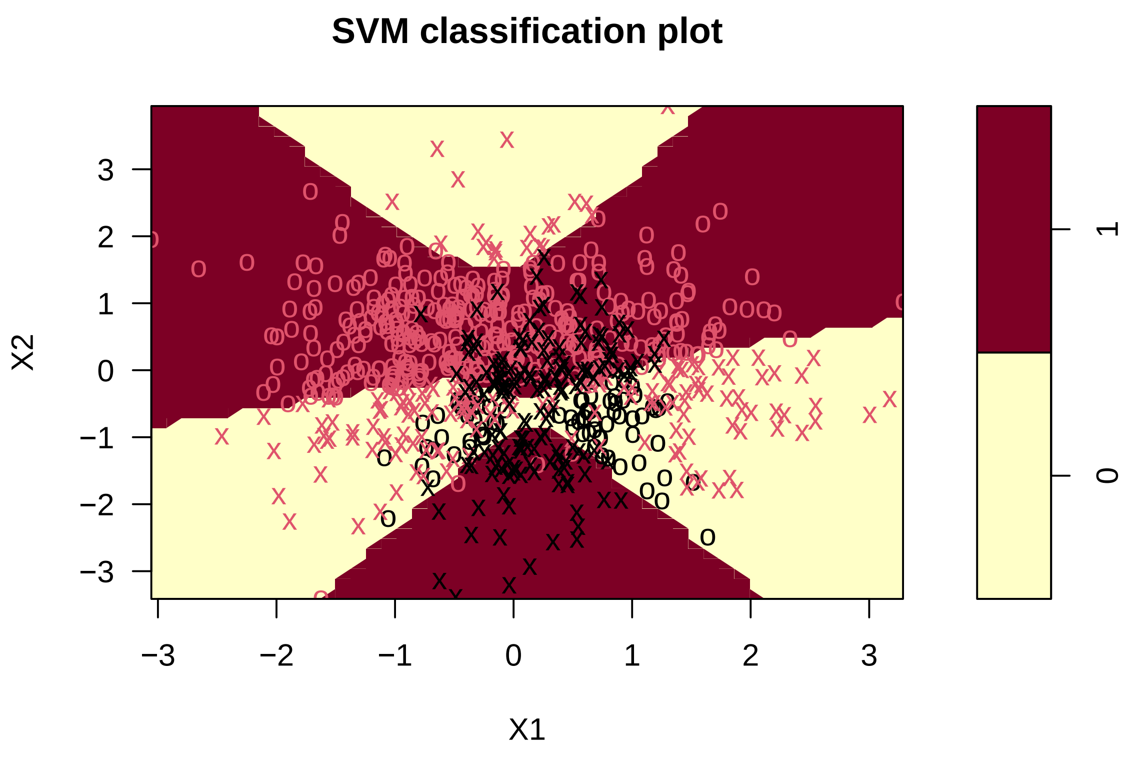
```
set.seed(123)

mdl_sig <- svm(yTrain~., data = df, kernel = "sigmoid")

print(mdl_sig)
```

```
##
## Call:
## svm(formula = yTrain ~ ., data = df, kernel = "sigmoid")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  sigmoid
##        cost:  1
##      coef.0:  0
##
## Number of Support Vectors:  328
```

```
plot(mdl_sig, df, X2 ~ X1)
```



```
yTestPred_sig <- predict(mdl_sig, newdata=xTest)
confusionMatrix(yTestPred_sig, yTest) # predicted/true
```

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

```
##
##           Reference
## Prediction   0   1
##          0  32  57
##          1  47 113
##
##                Accuracy : 0.5823
##                  95% CI : (0.5184, 0.6443)
##     No Information Rate : 0.6827
##     P-Value [Acc > NIR] : 0.9997
##
##                   Kappa : 0.0675
##
##  Mcnemar's Test P-Value : 0.3775
##
##             Sensitivity : 0.4051
##             Specificity : 0.6647
##          Pos Pred Value : 0.3596
##          Neg Pred Value : 0.7063
##              Prevalence : 0.3173
##          Detection Rate : 0.1285
##    Detection Prevalence : 0.3574
##       Balanced Accuracy : 0.5349
##
##        'Positive' Class : 0
##
```

Kernel Evaluation:

The RBF kernel achieved the highest accuracy (**82.33**%) with strong sensitivity and specificity, demonstrating its effectiveness in handling non-linear class separations. The Polynomial kernel showed moderate accuracy (**71.08**%), while the Sigmoid kernel performed poorly (**58.23**%), indicating it is less suited to this dataset.

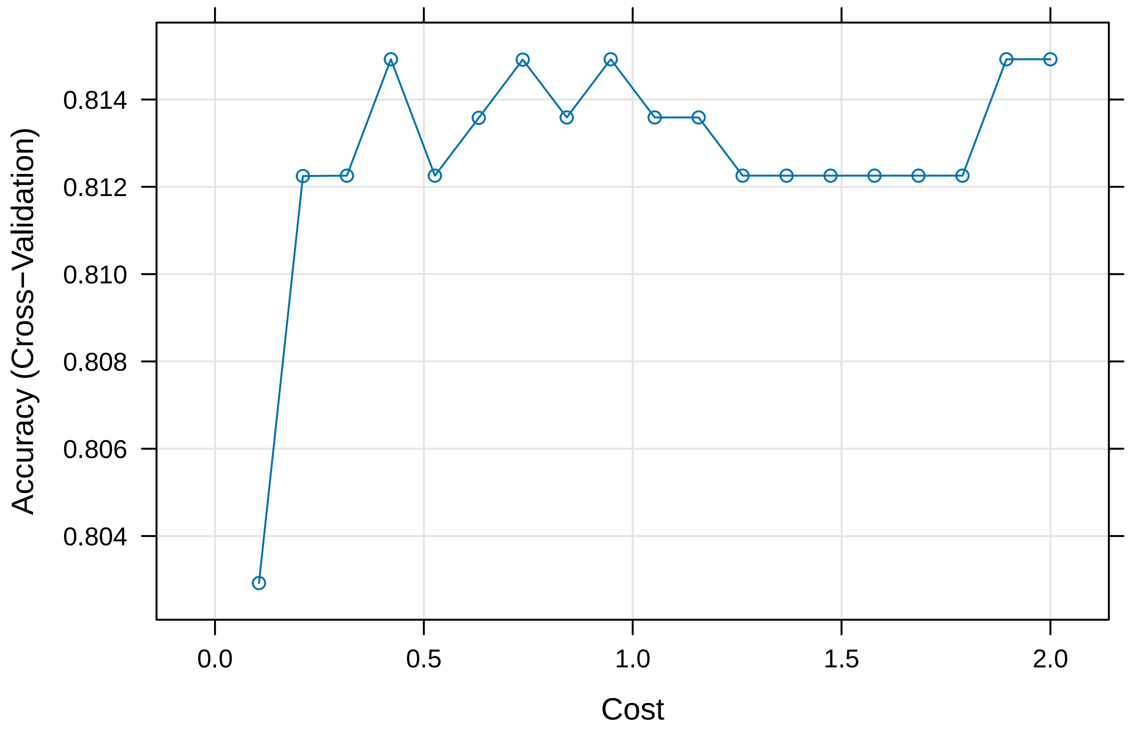
Based on the decision boundaries and the original scatter plot, the **RBF kernel** provided the best fit, effectively capturing the non-linear relationships. While the Polynomial kernel performed reasonably, it was less effective than the RBF, and the Sigmoid kernel produced a poor decision boundary with low accuracy.

Optimised SVM Model

Implementing the optimal hyper parameter values:

```r
set.seed(123)
# Tuning the hyperparameter C
rbf_mdl2 <- train(x = xTrain, y = yTrain,
                  method = "svmRadial",
                  trControl = trainControl(method = "cv", number = 5),
                  tuneGrid = expand.grid(C = seq(0, 2, length = 20), sigma = 1))

# Plot model accuracy vs different values of Cost and Sigma
plot(rbf_mdl2)
```



```
print(rbf_mdl2$bestTune)
```

```
##   sigma         C
## 5     1 0.4210526
```

```
# Evaluate the optimized model on the test data
yTestPred_rbf2<- predict(rbf_mdl2, newdata = xTest)
confusionMatrix(yTestPred_rbf2, yTest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  48  12
##          1  31 158
##
##                Accuracy : 0.8273
##                  95% CI : (0.7745, 0.8721)
##     No Information Rate : 0.6827
##     P-Value [Acc > NIR] : 1.775e-07
##
##                   Kappa : 0.574
##
##  Mcnemar's Test P-Value : 0.006052
##
##             Sensitivity : 0.6076
##             Specificity : 0.9294
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.8360
##              Prevalence : 0.3173
##          Detection Rate : 0.1928
##    Detection Prevalence : 0.2410
##       Balanced Accuracy : 0.7685
##
##        'Positive' Class : 0
##
```

Hyperparameter tuning through cross-validation identified an optimal cost parameter (C = **0.421**), further increasing accuracy to **82.73**%. Sigma is set to one, as the model wouldn't run unless it was set to a value.

Interpretation of Results

The optimized SVM with radial kernel achieved the highest accuracy (**82.7**%) and excellent specificity (**92.9**%) among tested kernels, effectively minimizing false positives. While sensitivity decreased slightly after optimisation, the increase in specificity and overall accuracy suggests a more reliable model. This trade-off is beneficial in scenarios where minimizing false positives is critical. The clear and effective non-linear boundary produced by this kernel underscores its suitability for this dataset.

4. Model Comparison

This section aims to compare the classification results obtained from the four methods evaluated previously—KNN, QDA, Random Forest, and SVM—and select the best method(s) according to different modeling objectives.

Model Performance Table

Model	Accuracy (%)	Specificity (%)	Sensitivity (%)	Balanced Accuracy (%)	Cohen's Kappa
KNN (k=11)	82.33	91.76	**62.03**	76.90	0.57
QDA	81.12	**94.12**	53.16	73.64	0.52
Random Forest	79.52	87.65	**62.03**	74.84	0.51
SVM (Radial)	**82.73**	92.94	60.76	**76.85**	**0.57**

This table clearly summarises the performance metrics for each classification model, highlighting the optimized SVM with the radial kernel as achieving the highest overall accuracy and specificity, demonstrating a robust predictive performance.

Consideration of Modelling Objectives

The selection of a classification method strongly depends on practical and contextual requirements:

1. Prioritizing High Specificity (Minimizing False Positives): When minimizing false positives is critical, QDA and SVM (Radial) are the top choices.

- QDA achieved the highest specificity (**94.12**%), making it suitable for scenarios where false positives carry significant costs, such as in medical diagnostics or fraud detection.

- SVM (Radial) also performed well in specificity (**92.94**%) while maintaining high overall accuracy.

2. Prioritizing Balanced Accuracy (Balancing Sensitivity and Specificity): If the goal is to balance the correct identification of both classes, SVM (Radial) and KNN are preferable.

- Both achieved similar balanced accuracy (**77**%) and Cohen's Kappa (**0.57**), indicating consistent classification performance across both positive and negative cases.

3. Interpretability and Insight into Variable Importance:

- Although Random Forest had slightly lower accuracy, it provided valuable variable importance measures, offering insights into which features contribute most to classification decisions.

Best Method

Considering these objectives, the **SVM with the Radial Kernel** is recommended as the overall best classification method. It achieved the highest accuracy, strong specificity, and solid balanced accuracy, demonstrating reliability and robustness for this classification task.

The radial kernel effectively handles complex, non-linear class boundaries, reinforcing its suitability. While the model showed slightly lower sensitivity, its superior specificity ensures fewer false positives, which is crucial in contexts where misclassification is costly.

However, potential limitations, such as reduced sensitivity, should be acknowledged. For future improvements, exploring additional feature engineering, alternate kernel functions, or ensemble methods could enhance performance further.

5. Evaluation against True Classification

To evaluate the models against the **true classification** (without noise), the Classification-True.csv dataset was used. This comparison helps assess each model's robustness in differentiating true class boundaries from noisy variations.

```r
# Loading data
true_class <- read.csv("ClassificationTrue.csv")
yTrue <- as.factor(true_class$Group)

set.seed(123)
ii <- createDataPartition(y, p = 0.75, list = FALSE)
# Explicitly defining this again to make sure its correct
# I was getting an bug here when i first tried, so i used ChatGPT to help problem solve thi

# Ensuring True data is the same length as the predicted (249)
yTrue_test <- yTrue[-ii]

# Prediction Against True Class
KNN_CM <- confusionMatrix(knn_fit2, yTrue_test)
QDA_CM <- confusionMatrix(qda_predict$class, yTrue_test)
TREE_CM <- confusionMatrix(TreePred, yTrue_test)
SVM_CM <- confusionMatrix(yTestPred_rbf2, yTrue_test)
```

Comparison to True Classification and Final Recommendation

Model	Accuracy (with noise) (%)	Accuracy (true labels) (%)
KNN (k=11)	82.33	90.36
QDA	81.12	**92.37**
Random Forest	79.52	89.96
SVM (Radial)	**82.73**	90.76

- QDA achieved the highest accuracy with true labels (**92.37**%), suggesting it can effectively classify clean, noise-free data. However, its performance drops when noise is introduced, indicating susceptibility to data variability.
- SVM (Radial) showed consistent accuracy across both noisy (**82.73**%) and noise-free (**90.76**%) conditions, highlighting its robustness and better generalization to real-world, noisy scenarios.
- KNN and Random Forest also improved when evaluated against true labels, but the increase was less pronounced compared to SVM.

Final Recommendation

Considering performance across both noisy and noise-free datasets, SVM (Radial Kernel) emerges as the most reliable and robust classification method. Its consistent accuracy demonstrates strong generalization, making it well-suited for practical applications where data variability is expected.

ChatGPT Prompts: A range of different prompts I used:

1. Here is an error in my code " ", What is the issue and how can I fix it?
2. Can you help improve the format of this GT table?
3. Is my code clear and reproducible?
4. Can you check my writing and suggest corrrections/improvements?
5. Can you give me feedback on my work " ", and suggest ways I could improve it.