AdvancedStatsAssignment

Table of contents

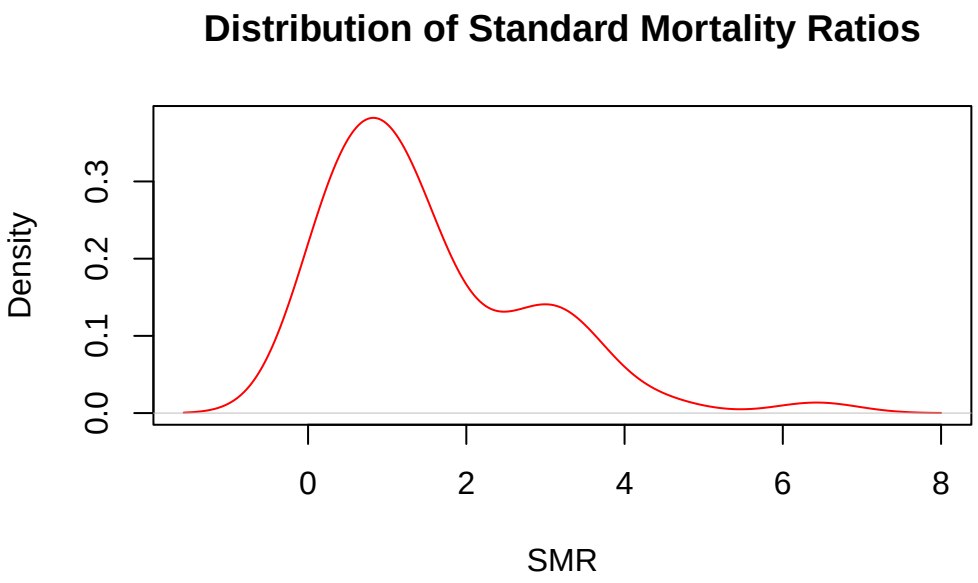
Question 1

```
source("ScotlandMap.R")  # need to read in the Scotland Map function
testdat = runif(56)  # generate random numbers to use as observations
ScotlandMap(testdat, figtitle = "Scotland random numbers")
```

Question 1 a)

```
ScotlandDF$SMR = ScotlandDF$Observed/ScotlandDF$Expected
```

```
plot(density(ScotlandDF$SMR), main = "Distribution of Standard Mortality Ratios",
    xlab = "SMR", ylab = "Density", col = "red")
```



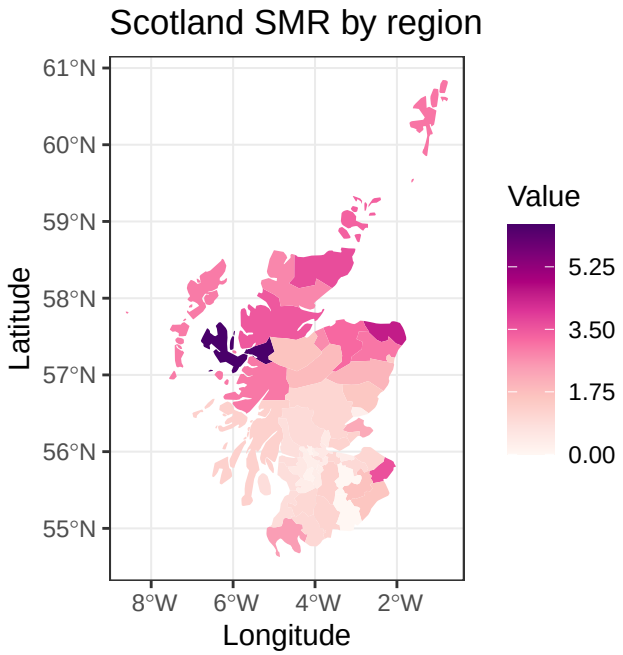

```
source("ScotlandMap.R")  # need to read in the ScotlandMap function
testdat = ScotlandDF$SMR  # generate random numbers to use as observations
ScotlandMap(testdat, figtitle = "Scotland SMR by region")
```

```
Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others
```

```
Warning: OGR support is provided by the sf and terra packages among others
```

```
Warning: OGR support is provided by the sf and terra packages among others
```

```
OGR data source with driver: ESRI Shapefile
Source: "C:\AppliedDataScienceAndStatistics\Applied-Data-Science-and-Statistics\Term2\Advanc
with 56 features
It has 1 fields
```



Question 1 b)

$\beta 0$ is the intercept that represents the baseline for when all the other variables are 0 which can also be interpreted as the log number expected of cases per administrative area.

θi is the effect that captures the variation in the observed number of cases that cannot be explained by the expected numbers alone. θi also represents the deviation of the log observed number of cases from the log expected number of cases for area i.

In the Poisson model, θi represents the deviation of the observed number of cases in area i from the expected number of cases in area i, given the reference rates. In other words, it captures how much higher or lower the observed number of cases is compared to what we would expect based on the reference rates

The role of i in the estimation of the relative risk is to allow for variation in the risk of disease across different areas, over and above what is explained by the reference rates. By including a random effect for each area in the model, we are able to estimate the relative risk for each area, while accounting for the fact that some areas may have higher or lower risk due to factors that are not captured by the reference rates alone.

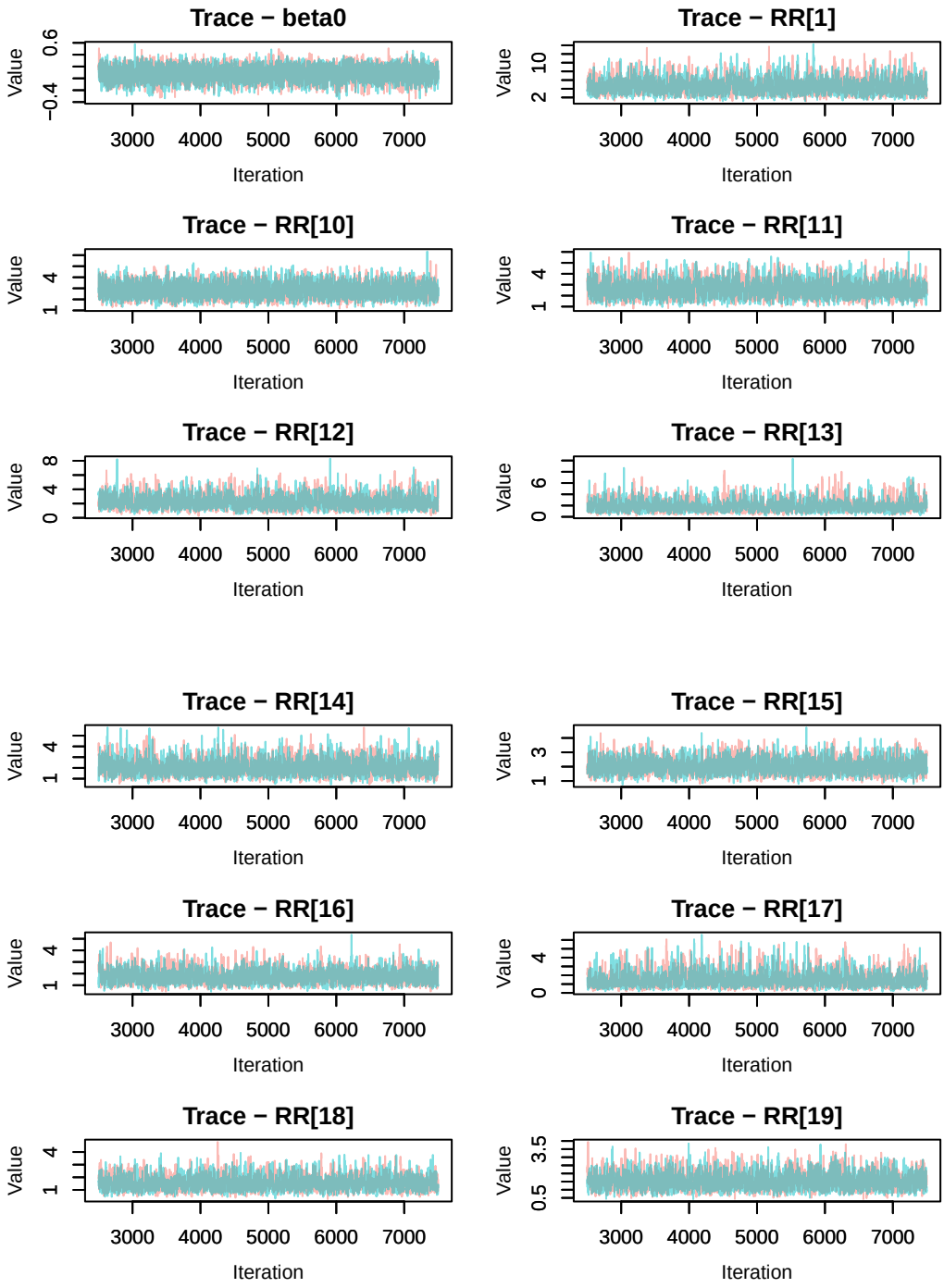
Question 1 c)

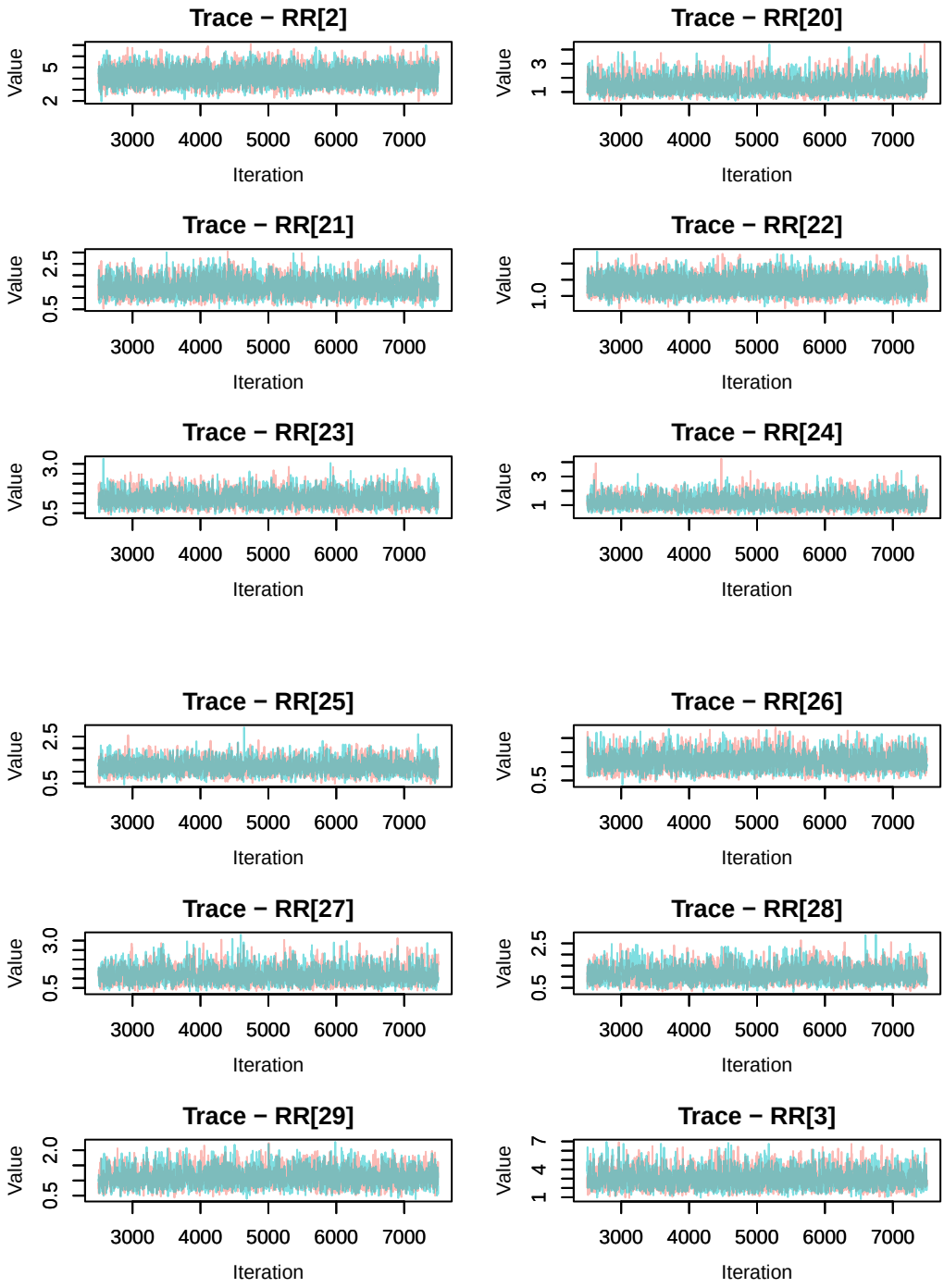
```
jags.mod.lipCancer = function() {
    # Priors
    beta0 ~ dnorm(0, 1/1000)
    tau ~ dgamma(1, 0.05)

    # Likelihood
    for (i in 1:N) {
        obs[i] ~ dpois(mu[i])
        log(mu[i]) = log(expo[i]) + beta0 + theta[i]
        theta[i] ~ dnorm(0, tau)
        RR[i] = exp(beta0) * exp(theta[i])
    }
}


# Parameters to monitor
jags.param.lipCancer = c("beta0", "theta", "tau", "RR")

# Data
jags.data.lipCancer <- list(N = nrow(ScotlandDF), obs = ScotlandDF$Observed,
    expo = ScotlandDF$Expected)

## initial values, have to be for the stochaic for thetta it will be a
## repetition because of the i
inits1 <- list(tau = 1, beta0 = 1, theta = rep(1, 56))
inits2 <- list(tau = 17.20806, beta0 = -0.5604756, theta = rep(19.38524,
    56))

## these initial values for inits2 came from these functions random
## generation set.seed(123) y <- rgamma(n = 1, shape = 1, rate = 0.05)
## value <- rnorm(1, mean = 0, sd = sqrt(1000)) tau_init <- 1/rgamma(n
## = 1, shape = 2, rate = 0.1)
```

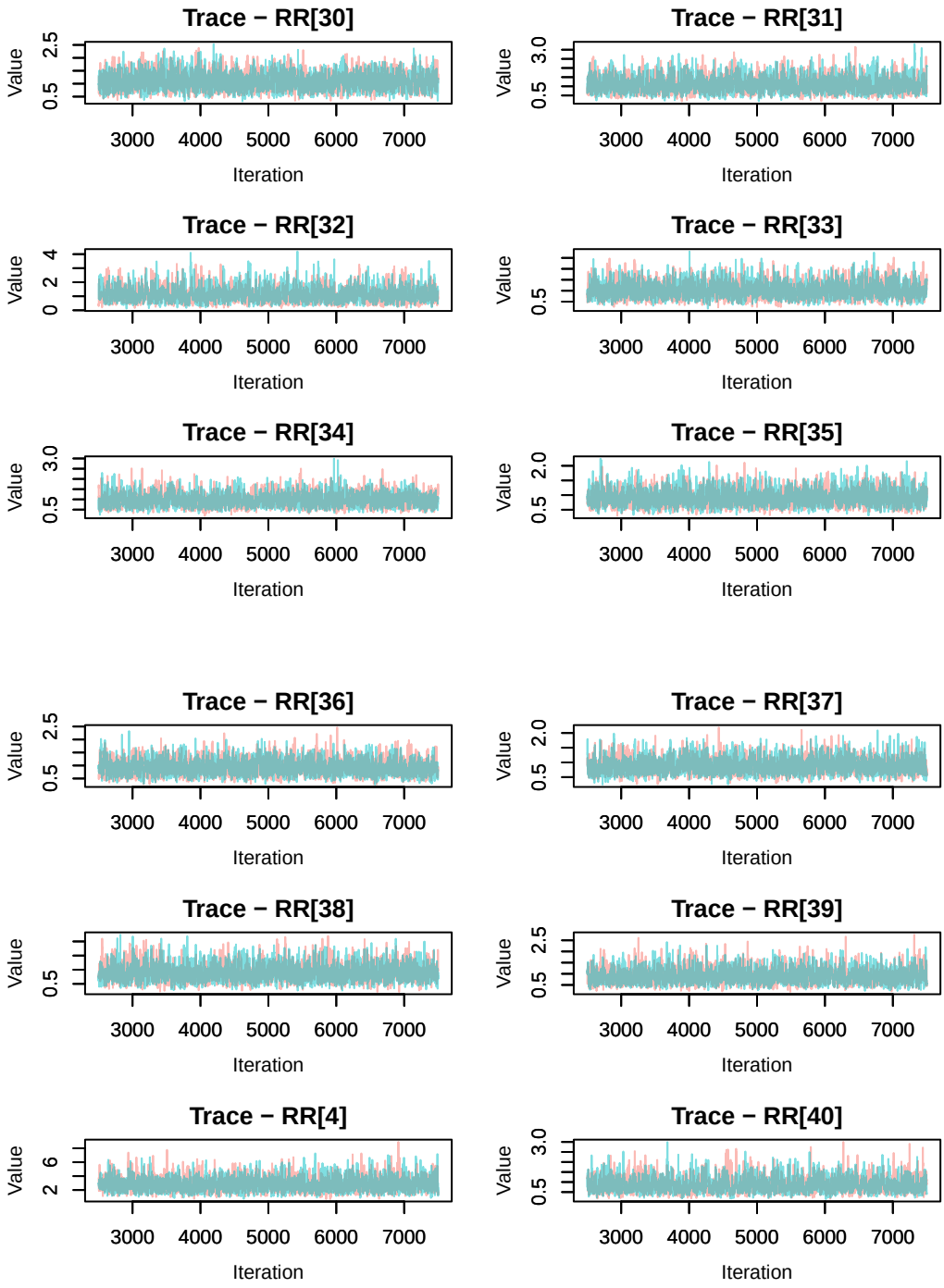
```r
jags.inits.lipCancer = list(inits1, inits2)

## fit the jags model

jags.mod.fit.lipCancer <- jags(data = jags.data.lipCancer, inits = jags.inits.lipCancer,
    parameters.to.save = jags.param.lipCancer, n.chains = 2, n.iter = 15000,
    n.burnin = 7500, n.thin = 1, model.file = jags.mod.lipCancer, DIC = FALSE)
```

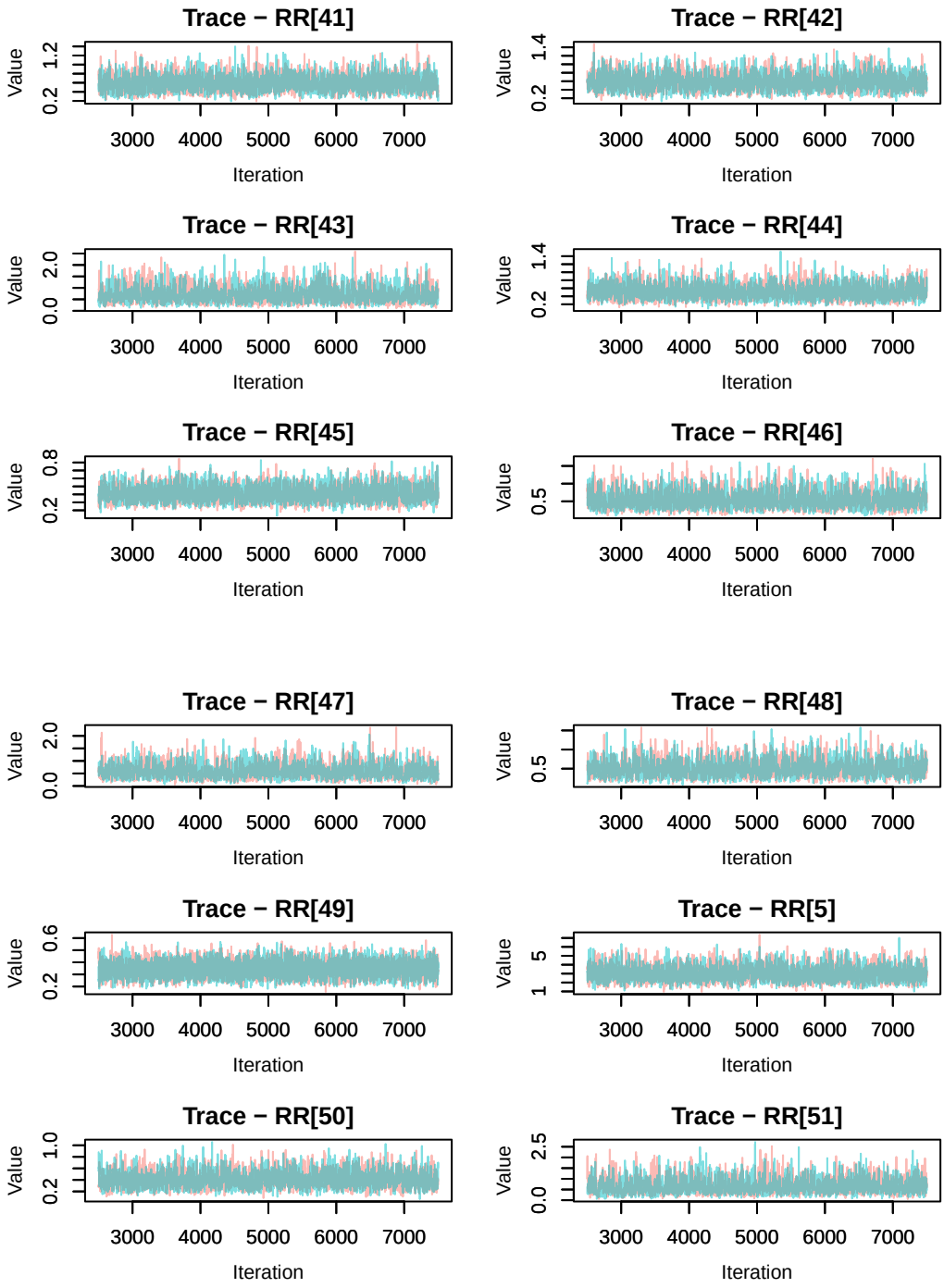
```
module glm loaded

module dic loaded

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 56
   Unobserved stochastic nodes: 58
   Total graph size: 450

Initializing model
```

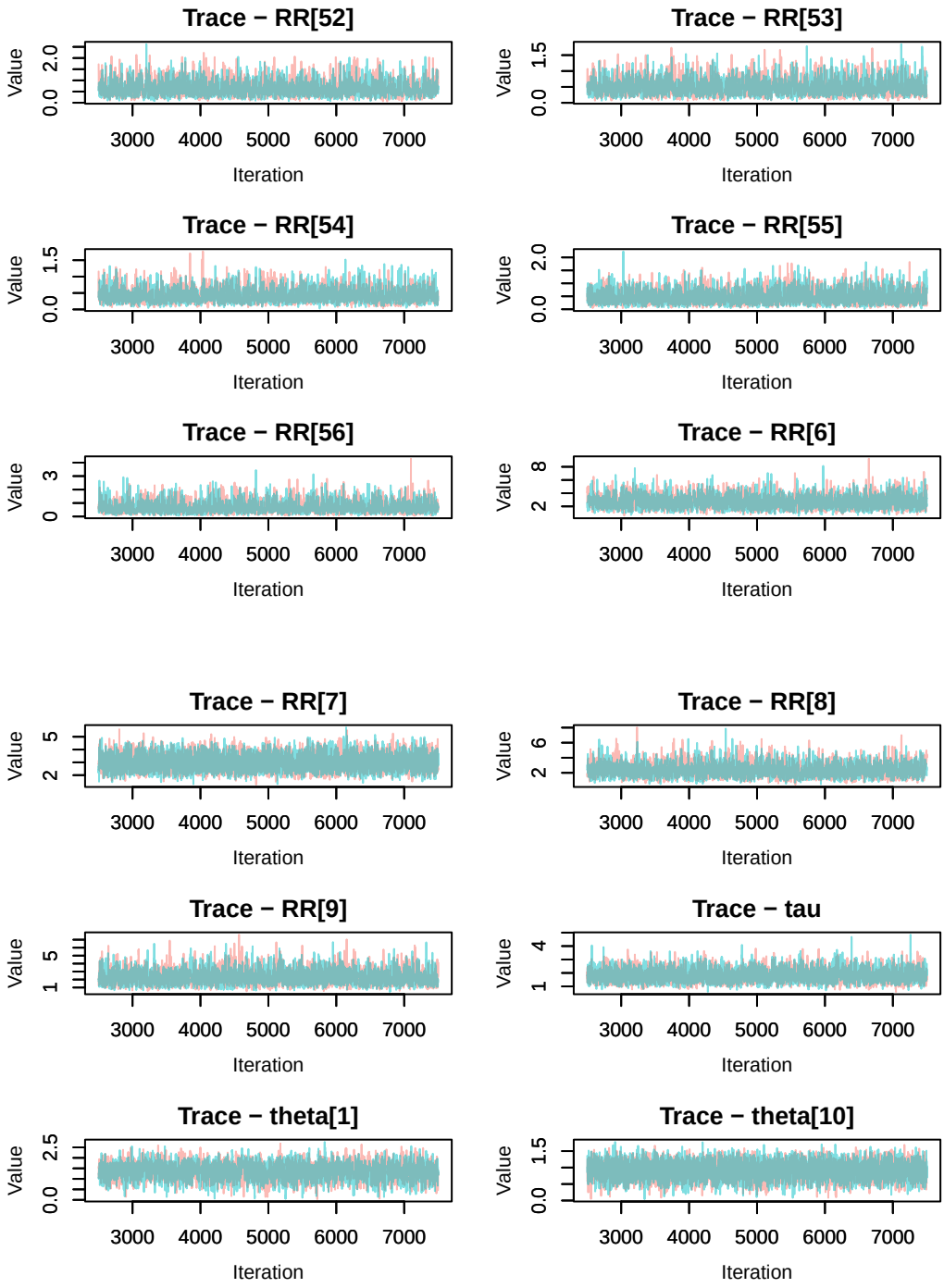
```r
jagsfitlipCancer.mcmc <- as.mcmc(jags.mod.fit.lipCancer)
MCMCtrace(jagsfitlipCancer.mcmc, type = "trace", ind = TRUE, pdf = FALSE)
```

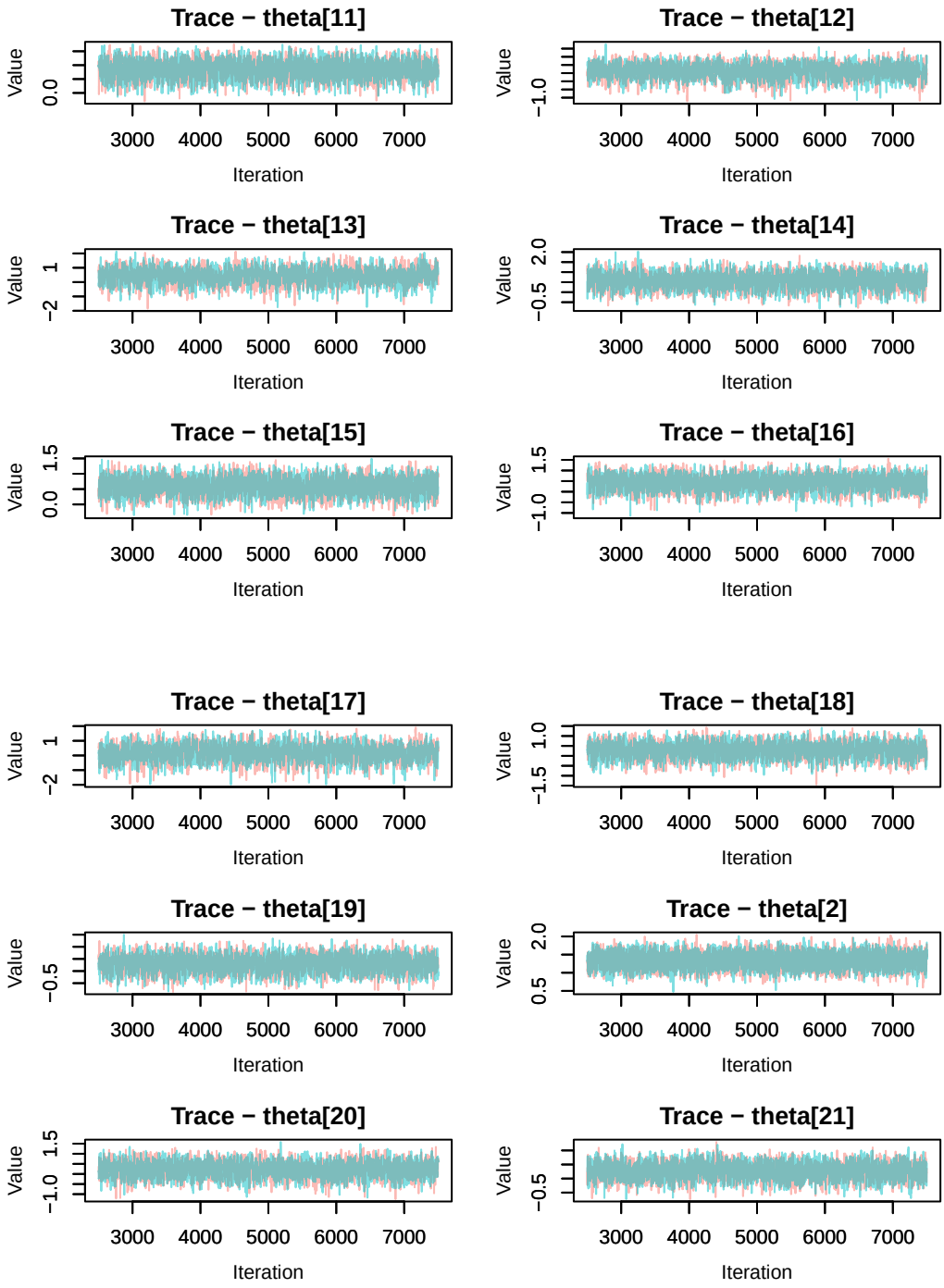

Trace − beta0
Value
−0.4
0.6
3000
4000
5000
6000
7000
Iteration
Trace − RR[1]
Value
2
10
3000
4000
5000
6000
7000
Iteration
Trace − RR[10]
Value
1
4
3000
4000
5000
6000
7000
Iteration
Trace − RR[11]
Value
1
4
3000
4000
5000
6000
7000
Iteration
Trace − RR[12]
Value
0
4
8
3000
4000
5000
6000
7000
Iteration
Trace − RR[13]
Value
0
6
3000
4000
5000
6000
7000
Iteration
Trace − RR[14]
Value
1
4
3000
4000
5000
6000
7000
Iteration
Trace − RR[15]
Value
1
3
3000
4000
5000
6000
7000
Iteration
Trace − RR[16]
Value
1
4
3000
4000
5000
6000
7000
Iteration
Trace − RR[17]
Value
0
4
3000
4000
5000
6000
7000
Iteration
Trace − RR[18]
Value
1
4
3000
4000
5000
6000
7000
Iteration
Trace − RR[19]
Value
0.5
3.5
3000
4000
5000
6000
7000
Iteration

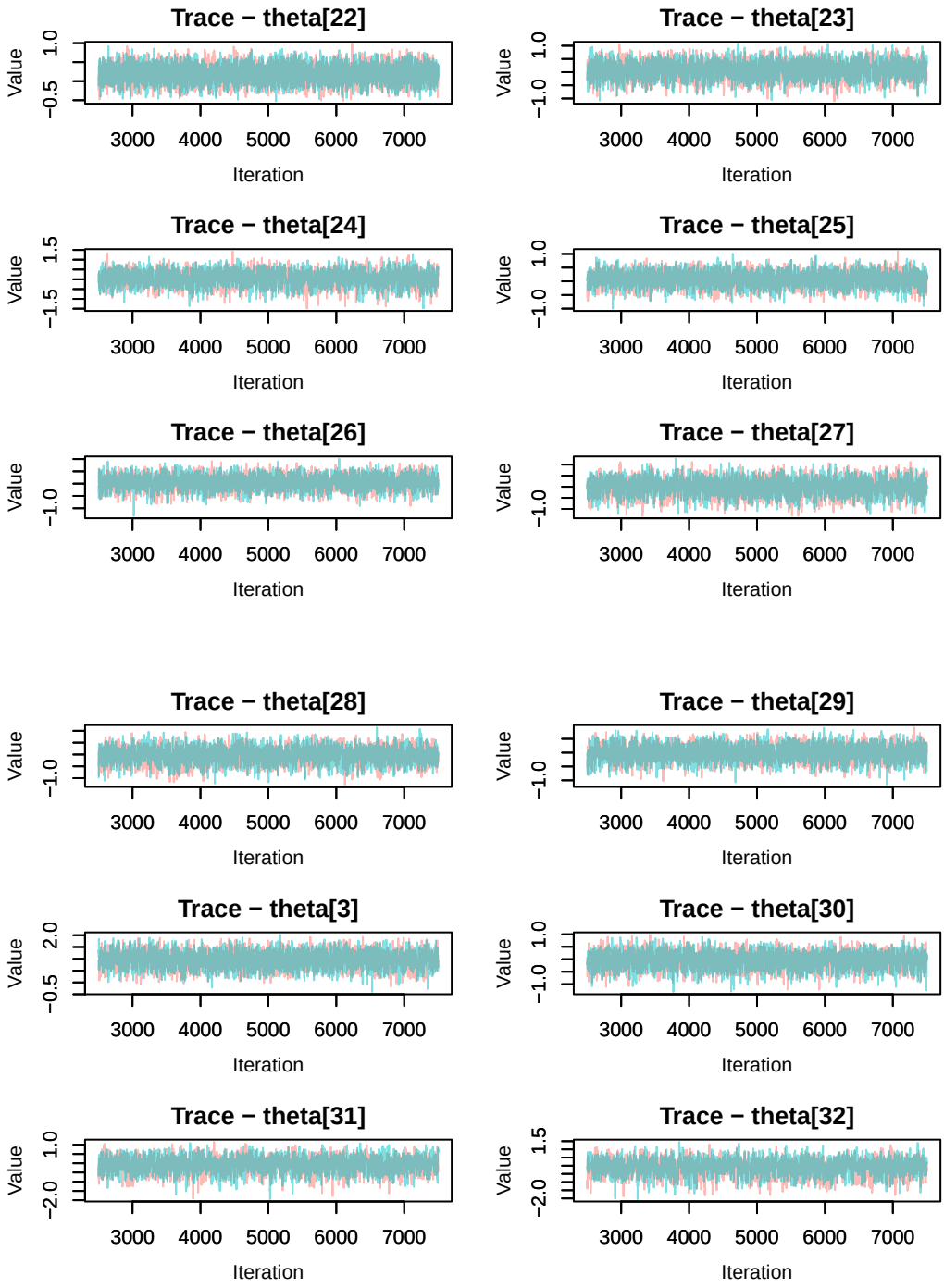

Trace − RR[2]
Trace − RR[20]
Trace − RR[21]
Trace − RR[22]
Trace − RR[23]
Trace − RR[24]
Trace − RR[25]
Trace − RR[26]
Trace − RR[27]
Trace − RR[28]
Trace − RR[29]
Trace − RR[3]
Value
Iteration
3000
4000
5000
6000
7000

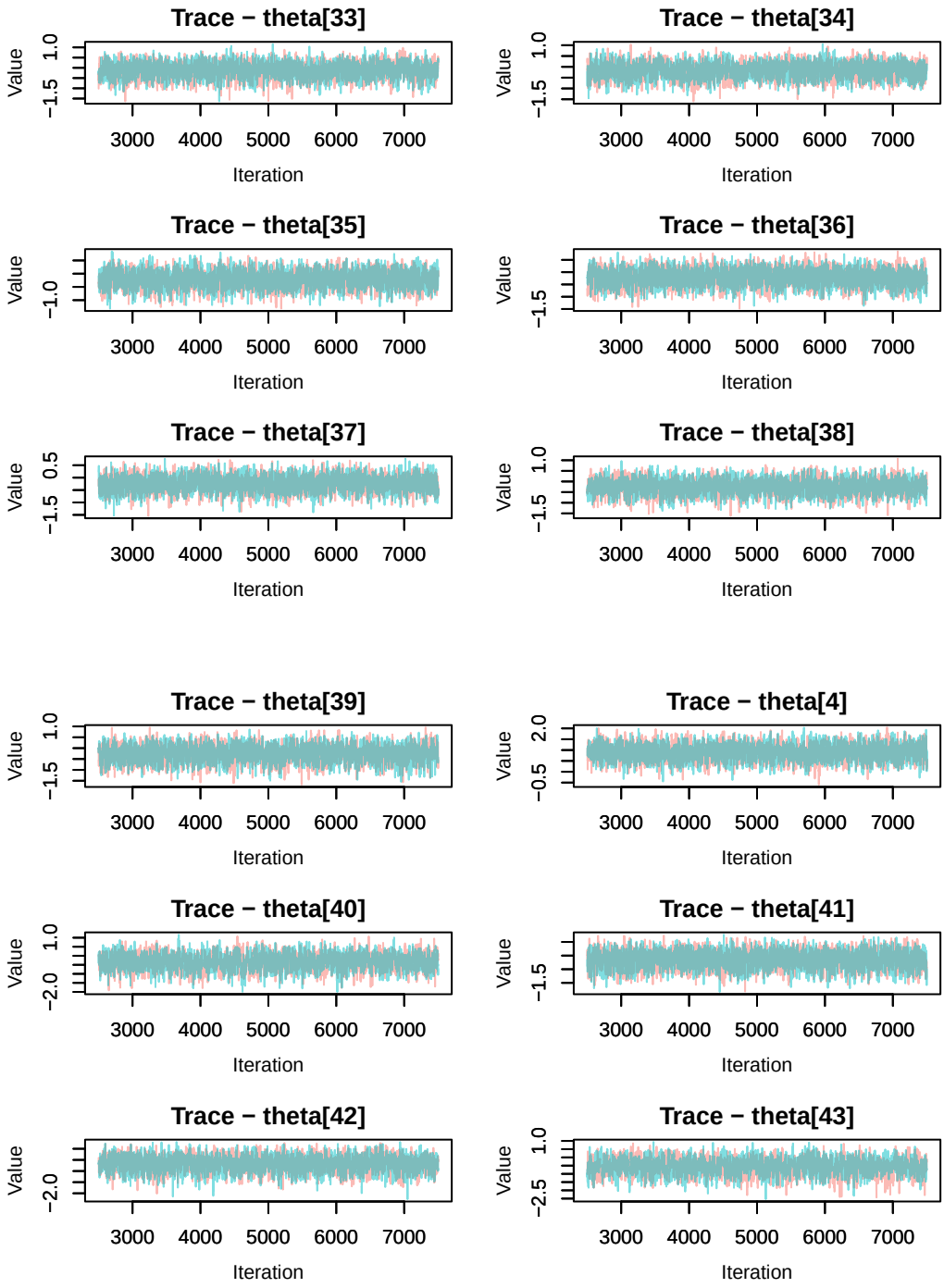

Trace − RR[30]
Value
0.5 2.5
3000 4000 5000 6000 7000
Iteration
Trace − RR[31]
Value
0.5 3.0
3000 4000 5000 6000 7000
Iteration
Trace − RR[32]
Value
0 2 4
3000 4000 5000 6000 7000
Iteration
Trace − RR[33]
Value
0.5
3000 4000 5000 6000 7000
Iteration
Trace − RR[34]
Value
0.5 3.0
3000 4000 5000 6000 7000
Iteration
Trace − RR[35]
Value
0.5 2.0
3000 4000 5000 6000 7000
Iteration
Trace − RR[36]
Value
0.5 2.5
3000 4000 5000 6000 7000
Iteration
Trace − RR[37]
Value
0.5 2.0
3000 4000 5000 6000 7000
Iteration
Trace − RR[38]
Value
0.5
3000 4000 5000 6000 7000
Iteration
Trace − RR[39]
Value
0.5 2.5
3000 4000 5000 6000 7000
Iteration
Trace − RR[4]
Value
2 6
3000 4000 5000 6000 7000
Iteration
Trace − RR[40]
Value
0.5 3.0
3000 4000 5000 6000 7000
Iteration

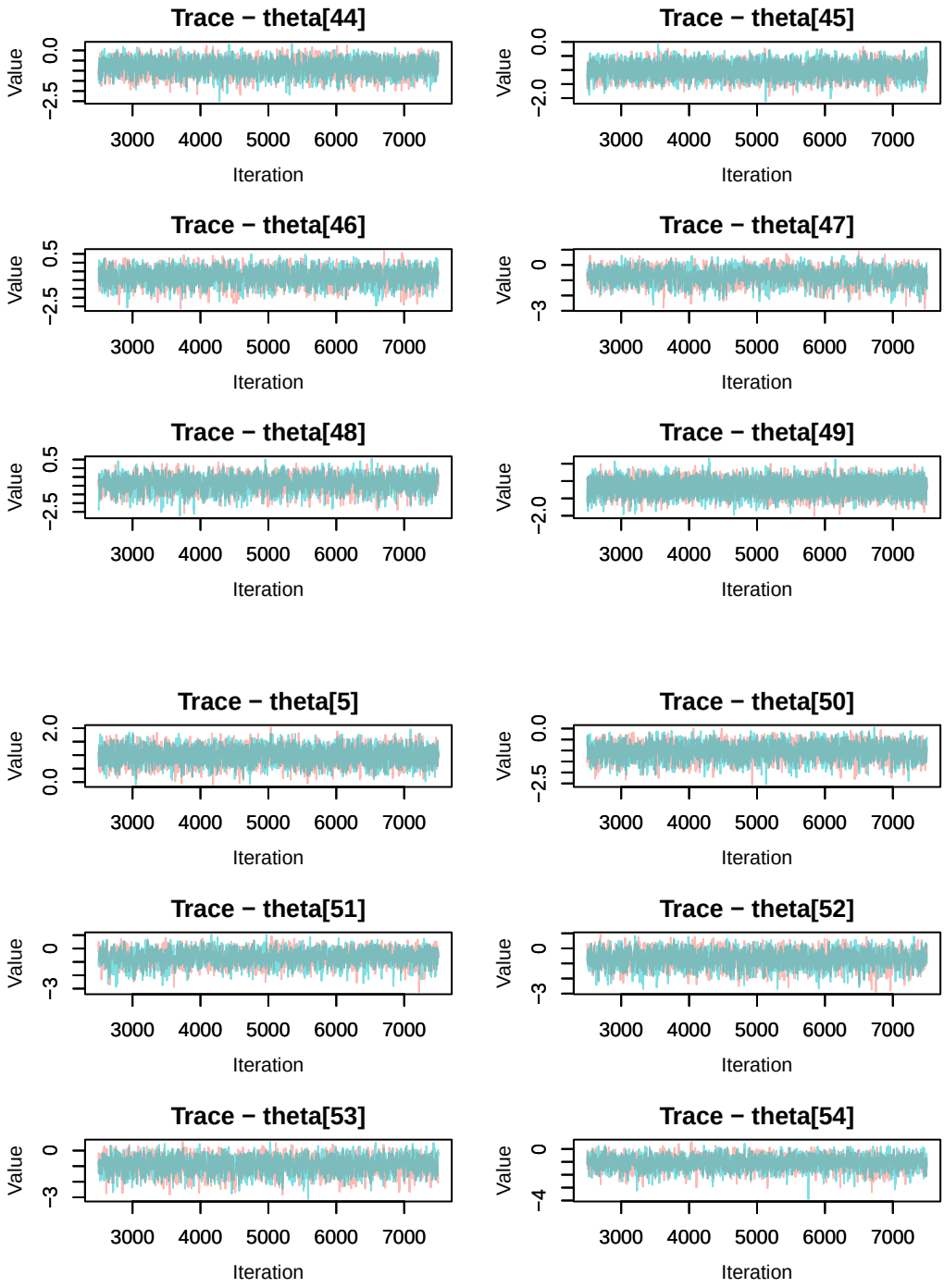

Trace − RR[41]
Value
0.2 1.2
3000 4000 5000 6000 7000
Iteration
Trace − RR[42]
Value
0.2 1.4
3000 4000 5000 6000 7000
Iteration
Trace − RR[43]
Value
0.0 2.0
3000 4000 5000 6000 7000
Iteration
Trace − RR[44]
Value
0.2 1.4
3000 4000 5000 6000 7000
Iteration
Trace − RR[45]
Value
0.2 0.8
3000 4000 5000 6000 7000
Iteration
Trace − RR[46]
Value
0.5
3000 4000 5000 6000 7000
Iteration
Trace − RR[47]
Value
0.0 2.0
3000 4000 5000 6000 7000
Iteration
Trace − RR[48]
Value
0.5
3000 4000 5000 6000 7000
Iteration
Trace − RR[49]
Value
0.2 0.6
3000 4000 5000 6000 7000
Iteration
Trace − RR[5]
Value
1 5
3000 4000 5000 6000 7000
Iteration
Trace − RR[50]
Value
0.2 1.0
3000 4000 5000 6000 7000
Iteration
Trace − RR[51]
Value
0.0 2.5
3000 4000 5000 6000 7000
Iteration

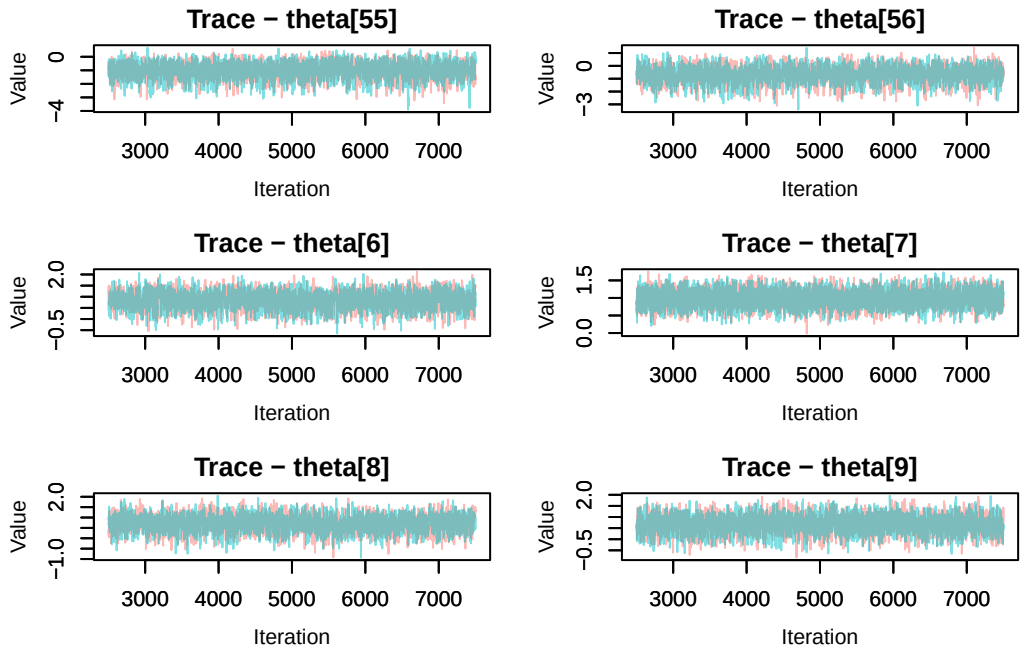

Trace − RR[52]
Value
Iteration
Trace − RR[53]
Value
Iteration
Trace − RR[54]
Value
Iteration
Trace − RR[55]
Value
Iteration
Trace − RR[56]
Value
Iteration
Trace − RR[6]
Value
Iteration
Trace − RR[7]
Value
Iteration
Trace − RR[8]
Value
Iteration
Trace − RR[9]
Value
Iteration
Trace − tau
Value
Iteration
Trace − theta[1]
Value
Iteration
Trace − theta[10]
Value
Iteration


Trace − theta[11]
Value
0.0
3000 4000 5000 6000 7000
Iteration
Trace − theta[12]
Value
−1.0
3000 4000 5000 6000 7000
Iteration
Trace − theta[13]
Value
−2
1
3000 4000 5000 6000 7000
Iteration
Trace − theta[14]
Value
−0.5
2.0
3000 4000 5000 6000 7000
Iteration
Trace − theta[15]
Value
0.0
1.5
3000 4000 5000 6000 7000
Iteration
Trace − theta[16]
Value
−1.0
1.5
3000 4000 5000 6000 7000
Iteration
Trace − theta[17]
Value
−2
1
3000 4000 5000 6000 7000
Iteration
Trace − theta[18]
Value
−1.5
1.0
3000 4000 5000 6000 7000
Iteration
Trace − theta[19]
Value
−0.5
3000 4000 5000 6000 7000
Iteration
Trace − theta[2]
Value
0.5
2.0
3000 4000 5000 6000 7000
Iteration
Trace − theta[20]
Value
−1.0
1.5
3000 4000 5000 6000 7000
Iteration
Trace − theta[21]
Value
−0.5
3000 4000 5000 6000 7000
Iteration


Trace − theta[22]
Value
Iteration
Trace − theta[23]
Value
Iteration
Trace − theta[24]
Value
Iteration
Trace − theta[25]
Value
Iteration
Trace − theta[26]
Value
Iteration
Trace − theta[27]
Value
Iteration
Trace − theta[28]
Value
Iteration
Trace − theta[29]
Value
Iteration
Trace − theta[3]
Value
Iteration
Trace − theta[30]
Value
Iteration
Trace − theta[31]
Value
Iteration
Trace − theta[32]
Value
Iteration


Trace − theta[33]
Trace − theta[34]
Trace − theta[35]
Trace − theta[36]
Trace − theta[37]
Trace − theta[38]
Trace − theta[39]
Trace − theta[4]
Trace − theta[40]
Trace − theta[41]
Trace − theta[42]
Trace − theta[43]
Value
Iteration
3000
4000
5000
6000
7000


Trace − theta[44]
Trace − theta[45]
Trace − theta[46]
Trace − theta[47]
Trace − theta[48]
Trace − theta[49]
Trace − theta[5]
Trace − theta[50]
Trace − theta[51]
Trace − theta[52]
Trace − theta[53]
Trace − theta[54]
Value
Iteration
3000
4000
5000
6000
7000




```
gelman.diag(jagsfitlipCancer.mcmc)
```

```
Potential scale reduction factors:

          Point est. Upper C.I.
beta0              1       1.00
RR[1]              1       1.01
RR[10]             1       1.00
RR[11]             1       1.00
RR[12]             1       1.00
RR[13]             1       1.00
RR[14]             1       1.00
RR[15]             1       1.00
RR[16]             1       1.01
RR[17]             1       1.01
RR[18]             1       1.00
RR[19]             1       1.00
RR[2]              1       1.00
RR[20]             1       1.00
RR[21]             1       1.01
RR[22]             1       1.00
RR[23]             1       1.00
```

RR[24]	1	1.00
RR[25]	1	1.00
RR[26]	1	1.00
RR[27]	1	1.00
RR[28]	1	1.00
RR[29]	1	1.00
RR[3]	1	1.00
RR[30]	1	1.00
RR[31]	1	1.01
RR[32]	1	1.01
RR[33]	1	1.00
RR[34]	1	1.00
RR[35]	1	1.00
RR[36]	1	1.01
RR[37]	1	1.00
RR[38]	1	1.00
RR[39]	1	1.00
RR[4]	1	1.00
RR[40]	1	1.00
RR[41]	1	1.01
RR[42]	1	1.00
RR[43]	1	1.00
RR[44]	1	1.00
RR[45]	1	1.01
RR[46]	1	1.00
RR[47]	1	1.00
RR[48]	1	1.00
RR[49]	1	1.00
RR[5]	1	1.00
RR[50]	1	1.00
RR[51]	1	1.01
RR[52]	1	1.00
RR[53]	1	1.00
RR[54]	1	1.00
RR[55]	1	1.00
RR[56]	1	1.00
RR[6]	1	1.00
RR[7]	1	1.00
RR[8]	1	1.00
RR[9]	1	1.00
tau	1	1.00
theta[1]	1	1.00
theta[10]	1	1.00

theta[11]	1	1.00
theta[12]	1	1.01
theta[13]	1	1.00
theta[14]	1	1.00
theta[15]	1	1.00
theta[16]	1	1.01
theta[17]	1	1.01
theta[18]	1	1.00
theta[19]	1	1.00
theta[2]	1	1.00
theta[20]	1	1.00
theta[21]	1	1.01
theta[22]	1	1.00
theta[23]	1	1.00
theta[24]	1	1.00
theta[25]	1	1.00
theta[26]	1	1.00
theta[27]	1	1.00
theta[28]	1	1.00
theta[29]	1	1.00
theta[3]	1	1.00
theta[30]	1	1.00
theta[31]	1	1.01
theta[32]	1	1.02
theta[33]	1	1.00
theta[34]	1	1.00
theta[35]	1	1.00
theta[36]	1	1.00
theta[37]	1	1.00
theta[38]	1	1.00
theta[39]	1	1.00
theta[4]	1	1.00
theta[40]	1	1.00
theta[41]	1	1.00
theta[42]	1	1.00
theta[43]	1	1.00
theta[44]	1	1.00
theta[45]	1	1.00
theta[46]	1	1.00
theta[47]	1	1.00
theta[48]	1	1.00
theta[49]	1	1.00
theta[5]	1	1.00

```
theta[50]          1       1.00
theta[51]          1       1.01
theta[52]          1       1.00
theta[53]          1       1.00
theta[54]          1       1.01
theta[55]          1       1.00
theta[56]          1       1.00
theta[6]           1       1.00
theta[7]           1       1.00
theta[8]           1       1.00
theta[9]           1       1.00

Multivariate psrf

1.03
```

question 1.d)

we know have to monitor RR as well

```
## extract posterior RR
posRR <- substr(rownames(jags.mod.fit.lipCancer$BUGSoutput$summary), 1, 2) ==
    "RR"

RRRegions = jags.mod.fit.lipCancer$BUGSoutput$summary[posRR, 1]

ScotlandDF$RR = RRRegions
```

Checking the difference between the average RR and the SMR for each region

I firstly do RR/SMR and see how close it is to 1

```
ScotlandDF$SMR/ScotlandDF$RR
```

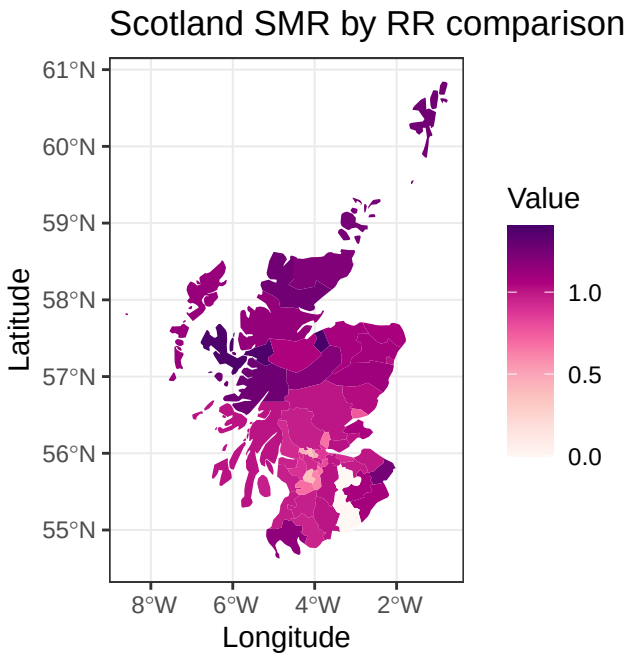
```
    RR[1]     RR[2]     RR[3]     RR[4]     RR[5]     RR[6]     RR[7]     RR[8]
1.3827601 1.0692991 1.2060200 1.2431525 1.1403501 1.2358500 1.0744491 1.2560451
    RR[9]    RR[10]    RR[11]    RR[12]    RR[13]    RR[14]    RR[15]    RR[16]
1.2656534 1.0860679 1.1253242 1.2606595 1.4014936 1.1566445 1.0634921 1.1015585
   RR[17]    RR[18]    RR[19]    RR[20]    RR[21]    RR[22]    RR[23]    RR[24]
1.1873541 1.0749224 1.0546081 1.0798189 1.0291039 1.0173163 1.0089986 1.0108396
   RR[25]    RR[26]    RR[27]    RR[28]    RR[29]    RR[30]    RR[31]    RR[32]
1.0086968 1.0131638 1.0040717 0.9958646 0.9981068 0.9954143 0.9677997 0.9499865
```

```
   RR[33]    RR[34]    RR[35]    RR[36]    RR[37]    RR[38]    RR[39]    RR[40]
0.9776905 0.9724289 0.9765388 0.9576960 0.9687556 0.9508922 0.9232974 0.8887376
   RR[41]    RR[42]    RR[43]    RR[44]    RR[45]    RR[46]    RR[47]    RR[48]
0.8927977 0.8676651 0.6742266 0.8097408 0.9120022 0.6833366 0.6138317 0.6568076
   RR[49]    RR[50]    RR[51]    RR[52]    RR[53]    RR[54]    RR[55]    RR[56]
0.9295735 0.7565476 0.4569503 0.4503310 0.3697826 0.3495811 0.0000000 0.0000000
```

And now I plot it on the map to see how close the posterior is to SMR likelihood

```
comparison = ScotlandDF$SMR/ScotlandDF$RR

source("ScotlandMap.R")  # need to read in the ScotlandMap function
testdat = comparison  # generate random numbers to use as observations
ScotlandMap(testdat, figtitle = "Scotland SMR by RR comparison by region")
```

```
Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

OGR data source with driver: ESRI Shapefile
Source: "C:\AppliedDataScienceAndStatistics\Applied-Data-Science-and-Statistics\Term2\Advanc
with 56 features
It has 1 fields
```



Question 1.e)

Now I will have to check when RR is > 1 using similarly the summary Confidicence intervals

```
jags.mod.lipCancer = function() {
    # Priors
    beta0 ~ dnorm(0, 1/1000)
    tau ~ dgamma(1, 0.05)

    # Likelihood
    for (i in 1:N) {
        obs[i] ~ dpois(mu[i])
        log(mu[i]) = log(expo[i]) + beta0 + theta[i]
        theta[i] ~ dnorm(0, tau)
        RR[i] = exp(beta0) * exp(theta[i])
    }

    PRRAbove = ifelse(RR > 1, 1, 0)

}

# Parameters to monitor
```

```r
jags.param.lipCancer = c("beta0", "theta", "tau", "RR", "PRRAbove")


# Data
jags.data.lipCancer <- list(N = nrow(ScotlandDF), obs = ScotlandDF$Observed,
    expo = ScotlandDF$Expected)

## initial values, have to be for the stochaic for thetta it will be a
## repetition because of the i
inits1 <- list(tau = 1, beta0 = 1, theta = rep(1, 56))
inits2 <- list(tau = 17.20806, beta0 = -0.5604756, theta = rep(19.38524,
    56))

## these initial values for inits2 came from these functions random
## generation set.seed(123) y <- rgamma(n = 1, shape = 1, rate = 0.05)
## value <- rnorm(1, mean = 0, sd = sqrt(1000)) tau_init <- 1/rgamma(n
## = 1, shape = 2, rate = 0.1)


jags.inits.lipCancer = list(inits1, inits2)

## fit the jags model

jags.mod.fit.lipCancer <- jags(data = jags.data.lipCancer, inits = jags.inits.lipCancer,
    parameters.to.save = jags.param.lipCancer, n.chains = 2, n.iter = 15000,
    n.burnin = 7500, n.thin = 1, model.file = jags.mod.lipCancer, DIC = FALSE)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 56
   Unobserved stochastic nodes: 58
   Total graph size: 453

Initializing model
```

```r
## to see which regions RR are in the 95% C.I we will have to see if
## the 2.5% is small or equal to 1 and if 97.5% is equal or greater
## than 1
```

```r
## extract posterior RR
posProbRR <- substr(rownames(jags.mod.fit.lipCancer$BUGSoutput$summary),
    1, 2) == "RR"

# jags.mod.fit.lipCancer$BUGSoutput$summary[posProbRR,1] ##mean
lowerCI = jags.mod.fit.lipCancer$BUGSoutput$summary[posProbRR, 3]  # 2.5 percentile
upperCI = jags.mod.fit.lipCancer$BUGSoutput$summary[posProbRR, 7]  # 97.5 percentile
```

Since we know that the 97,5 percentile is always bigger than the the 2,5 percentile to check if the 95% CI of RR is lower than 1, meaning there is no excessive risk, all we have to do is check if a region 97,5% percentile is smaller than 1

```r
ScotlandDF$AboveRR = upperCI
ScotlandDF$isAboveRR = 0

for (i in 1:nrow(ScotlandDF)) {

    if (ScotlandDF$AboveRR[i] > 1) {
        ScotlandDF$isAboveRR[i] = 1
    }

    if (upperCI[i] < 1) {
        print(ScotlandDF$Name[i])
    }
}
```

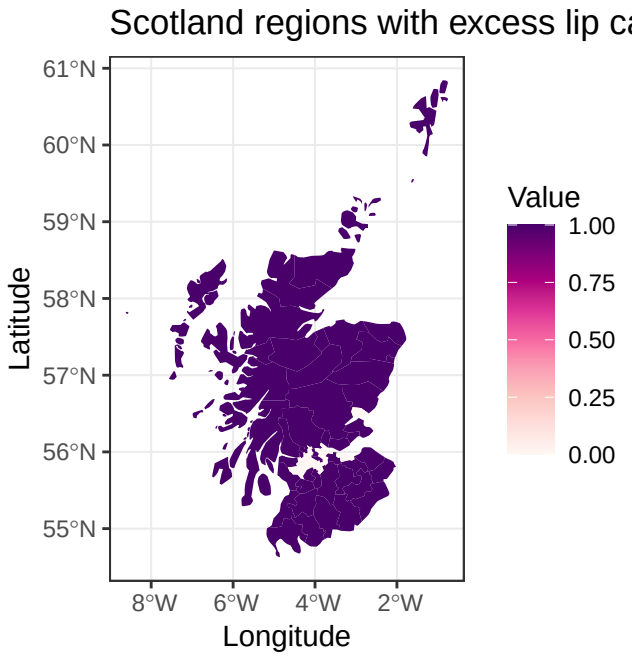
```
[1] "Renfrew"
[1] "Falkirk"
[1] "Motherwell"
[1] "Edinburgh"
[1] "Hamilton"
[1] "Glasgow"
[1] "Dundee"
[1] "Strathkelvin"
```

Here is the list of the regions where we are 95% certain that RR is bellow 1

So now to map the regions with and without excess risk of lip cancer

```r
source("ScotlandMap.R")  # need to read in the ScotlandMap function
testdat = ScotlandDF$isAboveRR  # generate random numbers to use as observations
ScotlandMap(testdat, figtitle = "Scotland regions with excess lip cancer")
```

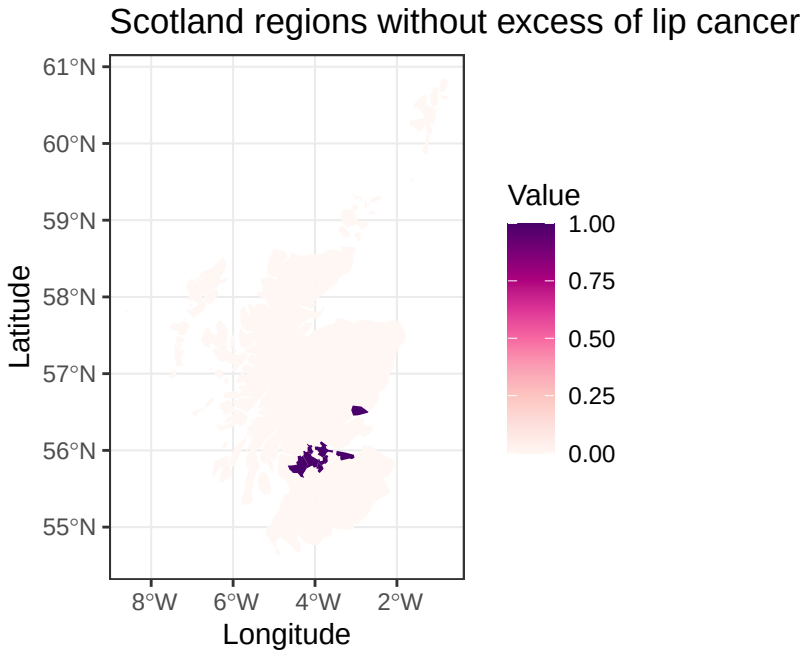
```
Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

OGR data source with driver: ESRI Shapefile
Source: "C:\AppliedDataScienceAndStatistics\Applied-Data-Science-and-Statistics\Term2\Advance
with 56 features
It has 1 fields
```




```
source("ScotlandMap.R")  # need to read in the ScotlandMap function
testdat = as.integer(!ScotlandDF$isAboveRR)  # generate random numbers to use as observati
```

```
ScotlandMap(testdat, figtitle = "Scotland regions without excess of lip cancer")
```

```
Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

Warning: OGR support is provided by the sf and terra packages among others

OGR data source with driver: ESRI Shapefile
Source: "C:\AppliedDataScienceAndStatistics\Applied-Data-Science-and-Statistics\Term2\Advance
with 56 features
It has 1 fields
```



Here I have decided to map separetly without and without the risk of lip cancer due to the default colour scheme of the above risk cancer regions making it very hard to see which regions have risk bellow 1.

Question 1f)

#exponential? normal/gamma prior? # for beta0 normal?

Question 2 a)

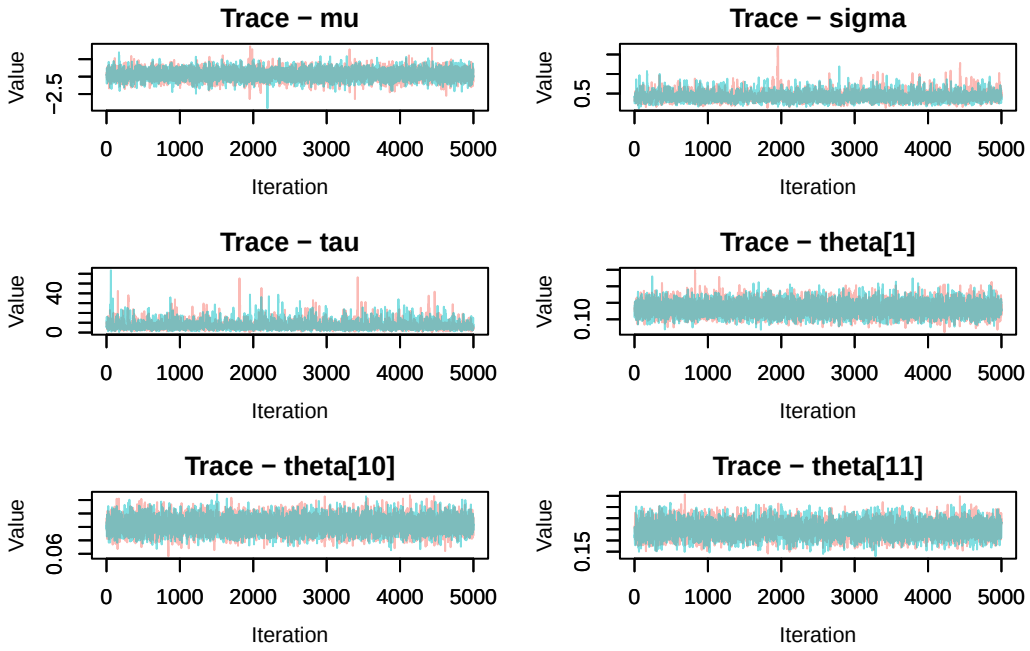
```
jags.mod.infantDeath <- function(){
for(i in 1:12){
r[i] ~ dbin(theta[i],n[i])
logit(theta[i]) <- logit.theta[i]
logit.theta[i] ~ dnorm(mu, tau)
}
mu ~ dnorm(0,0.001)
tau <- 1/sigma^2
sigma ~ dunif(0,100)
}

# Parameters to monitor
jags.param.infantDeath <- c("mu", "sigma", "theta", "tau")

## initial values
inits1 <- list('tau' = 100, 'sigma' = 20, 'theta'=c(10,-5,-25))
inits2 <- list('tau'=100000,'sigma'=-100,'theta'=c(-100,100,500))

jags.inits = list(inits1,inits2)

## data

jags.data.infantDeath <- list(r = surgicalDF$r, n = surgicalDF$n)

## fit the jags model

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

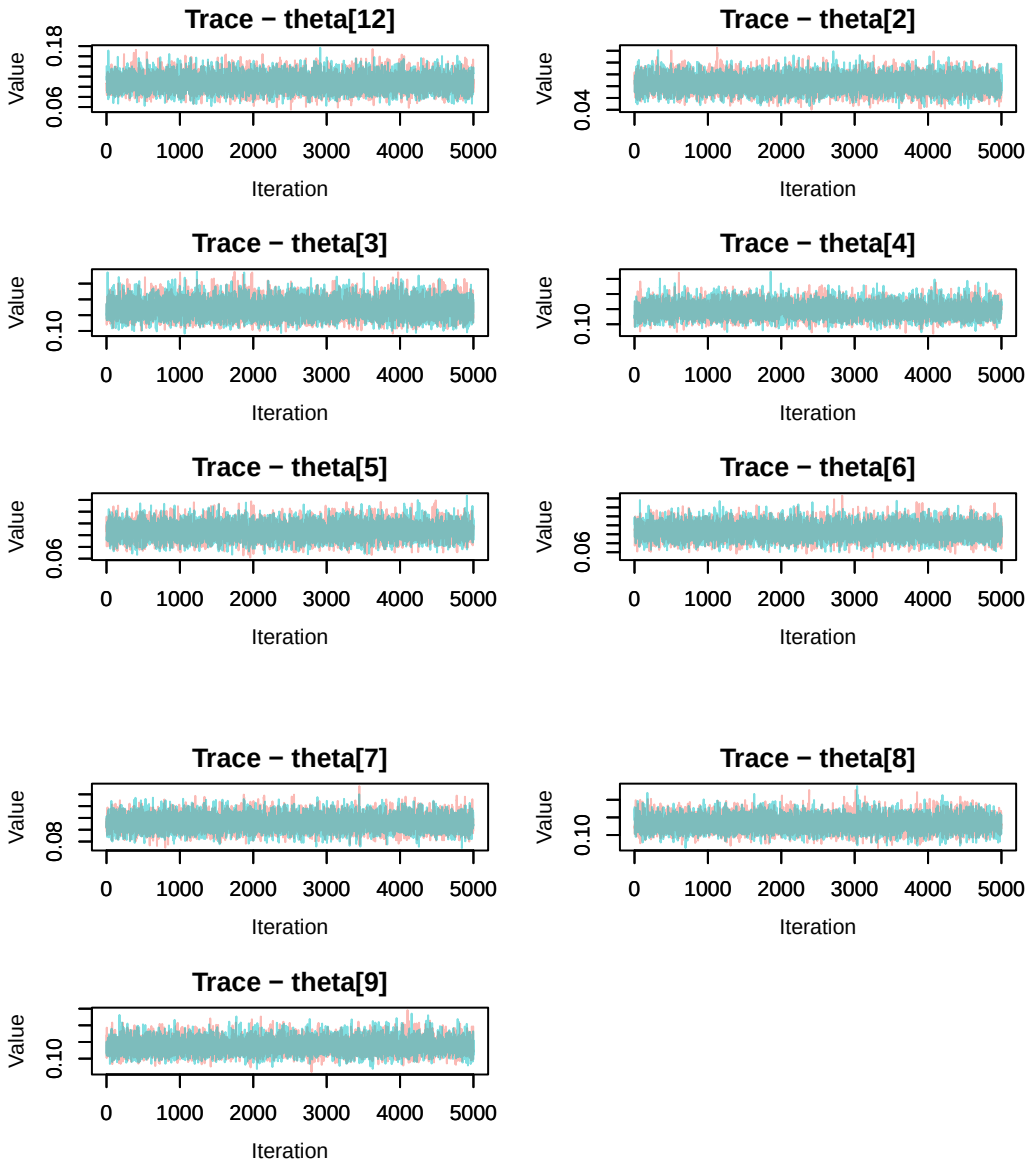
```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 12
```

```
   Unobserved stochastic nodes: 14
   Total graph size: 57

Initializing model
```

Now that we have the fitted the model we will observe the plots to check for possible convergence during the recorded 5000 iterations.

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





As we can see from the plots, there seems to be no visible patterns on the chains and all parameters have stable oscillation around a common value. We can also observe that the

chains seem to be well mixed with each other as it is difficult to distinguish between chains.

However visualization alone is not enough to fully access the convergence so we will use the Gelman-Rubin diagnostics to check for convergence.

```
gelman.diag(jagsfitinfantDeath.mcmc)
```

```
Potential scale reduction factors:

          Point est. Upper C.I.
mu                 1       1.00
sigma              1       1.01
tau                1       1.01
theta[1]           1       1.00
theta[10]          1       1.00
theta[11]          1       1.00
theta[12]          1       1.00
theta[2]           1       1.00
theta[3]           1       1.00
theta[4]           1       1.00
theta[5]           1       1.00
theta[6]           1       1.00
theta[7]           1       1.00
theta[8]           1       1.00
theta[9]           1       1.00

Multivariate psrf

1
```

As we can see from the Upper Confidence Interval, the values of all parameters are approximately 1, indicating that the parameters have indeed converged.

Question 2 b)

```
jags.mod.infantDeath <- function(){
for(i in 1:12){
r[i] ~ dbin(theta[i],n[i])
rpred[i] ~ dbin(theta[i],n[i])
logit(theta[i]) <- logit.theta[i]
logit.theta[i] ~ dnorm(mu, tau)
```

```
  }
  mu ~ dnorm(0,0.001)
  tau <- 1/sigma^2
  sigma ~ dunif(0,100)

  ## pi = P (rpred i > ri) + 1/2 * P (rpred i = ri)



  }

  # Parameters to monitor
  jags.param.infantDeath <- c("mu", "sigma", "theta", "tau")

  ## initial values
  inits1 <- list('tau' = 100, 'sigma' = 20, 'theta'=c(10,-5,-25))
  inits2 <- list('tau'=100000,'sigma'=-100,'theta'=c(-100,100,500))

  jags.inits = list(inits1,inits2)

  ## data

  jags.data.infantDeath <- list(r = surgicalDF$r, n = surgicalDF$n)

  ## fit the jags model

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

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 12
   Unobserved stochastic nodes: 26
   Total graph size: 69

Initializing model
```

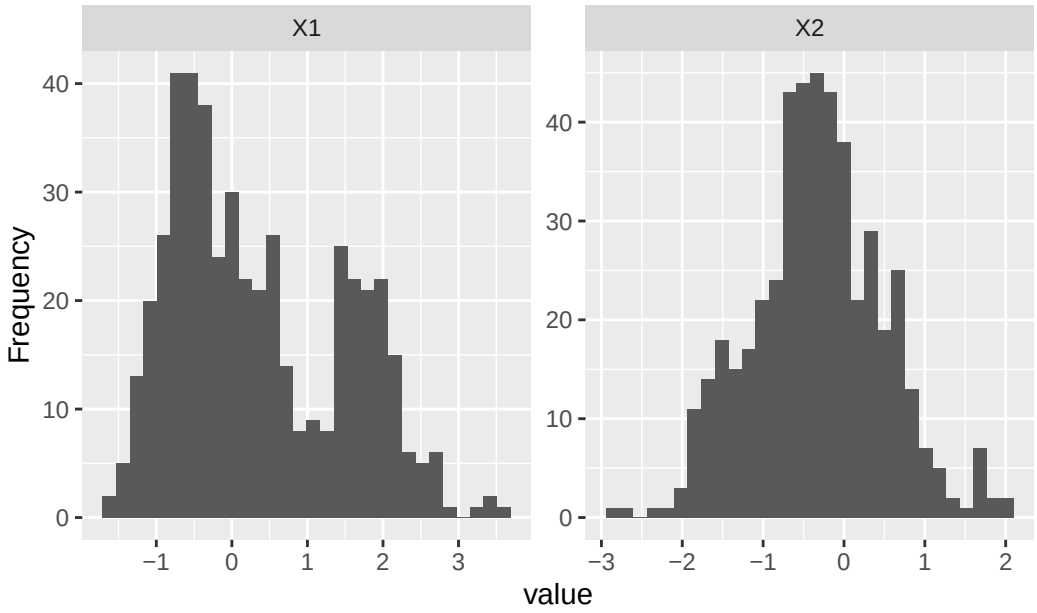
Question 2 c)

Question B - classification

Question B.1

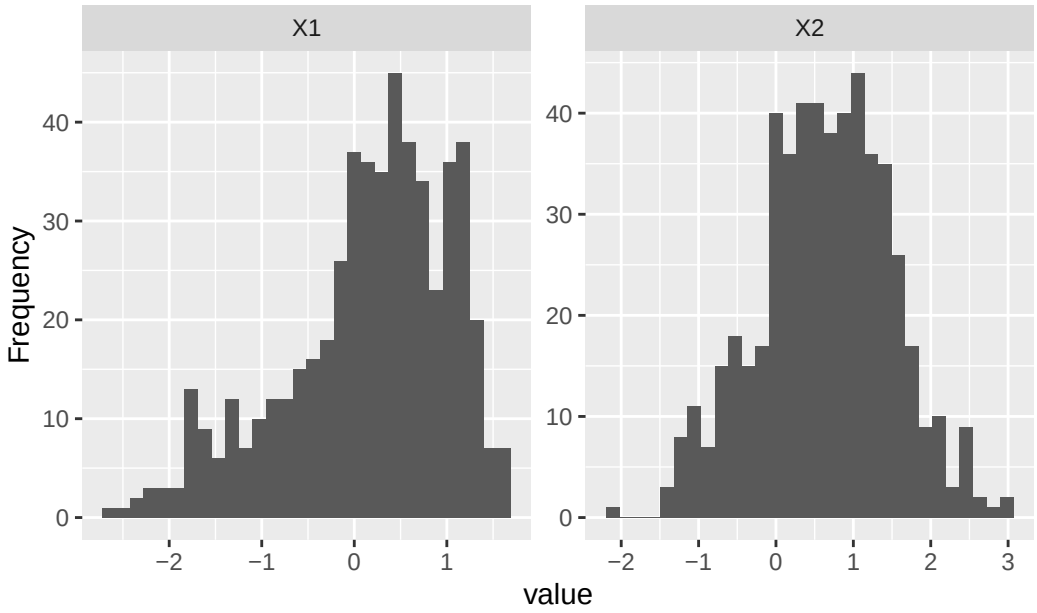
Group 0 is characterised for always having x1 > -1,65 and x2 < 2,05 with the majority of the occurrences of group 0 having

```
## I am just going to split the dataframe in two so its easier to show
## an histogram of each variable for both groups

group0 = subset(classificationDF, Group == 0)
group1 = subset(classificationDF, Group == 1)

## removing the id variable from both of these new data frames so they
## are not included in the histograms
group0 = group0 %>%
    select(-(1))
group1 = group1 %>%
    select(-(1))


plot_histogram(group0)
```



As we can see from the histograms from group 0 it seems that X2 seems to follow a normal distribution that is very slightly skewed to the left and has mean -0,5. X1 is clearly skewed to the right possibly following a normal distribution and again mean equal to -0,5.

```
plot_histogram(group1)
```



Group 1 histograms reveal that X2 is normally distributed with mean equal to 0,5 and sightly skewed to the left. The variable X1 also seems to follow a normal distribution that has a very prolonged skewness to the left and mean 0,5.0

Question B.2

Both LDA and QDA assume normality which seems to hold from what we can see from the histograms on question B.1

Starting with Linear discriminant analysis (LDA) we can easily see from the initial graph that we can see that the any sensible decision boundary is highly non-linear. As analysed from histograms this data violates the initial assumption of homogeneity of variance (homoscedasticity). The non-linearity of the data would also lead to LDA having terrible performance. As such we can conclude that LDA is not suitable for this data.

Secondly, quadratic discriminant analysis (QDA) follow the assumption of normality of each of the variables which does seem to hold even with how highly non-linear the data is. However judging from the plot with the observations, it does seem that QDA does not have enough flexibility to account for all the variability in what is the theoretical Bayes decision boundary due to the highly non-linear nature of the data. As such we can conclude that QDA is not suitable for this data.

Thirdly, K-nearest neighbour classification (KNN) has no assumptions made about the shape of the decision, as such as we choose a adequate level of smoothness to prevent overfitting KNN classification is an adequate method for the data.

Support Vector Machines (SVM) are most commonly used for binary classification, which holds true for our data. The high non-linear decision boundary however requires a careful selection of a non linear kernel function to ensure that the boundary becomes non-linear when converted back to the regular space. As such I deem SVM an appropriate method to classify the data.

Finally, Random forest also do not make any assumptions about the underlying distribution present on the data set. A random forest would also have the added bonus of letting us understand which of the variables X1 or X2 has the biggest impact on the classification of data points which could be useful for future applications of this model. As such I deem Random forest an appropriate method to classify the data.

Question B.3

```
# make this example reproducible
set.seed(26041999)
# use 80% of dataset as training set and 20% as test set
sample = sample.split(classificationDF$Group, SplitRatio = 0.8)
## note that the column selected above can be any column.
train = classificationDF[sample, ]
test = classificationDF[!sample, ]

# split the data using the indices returned by the createDataPartition
# function
X_train = train[, c("X1", "X2")]
y_train = train$Group
X_test = test[, c("X1", "X2")]
y_test = test$Group

# check the dimensions
dim(train)
```

```
[1] 800   4
```

```
dim(test)
```

```
[1] 200   4
```

Question B.4

The 3 method I will be using are Random forest, KNN and SVM as deemed to be the most appropriate methods for this data.

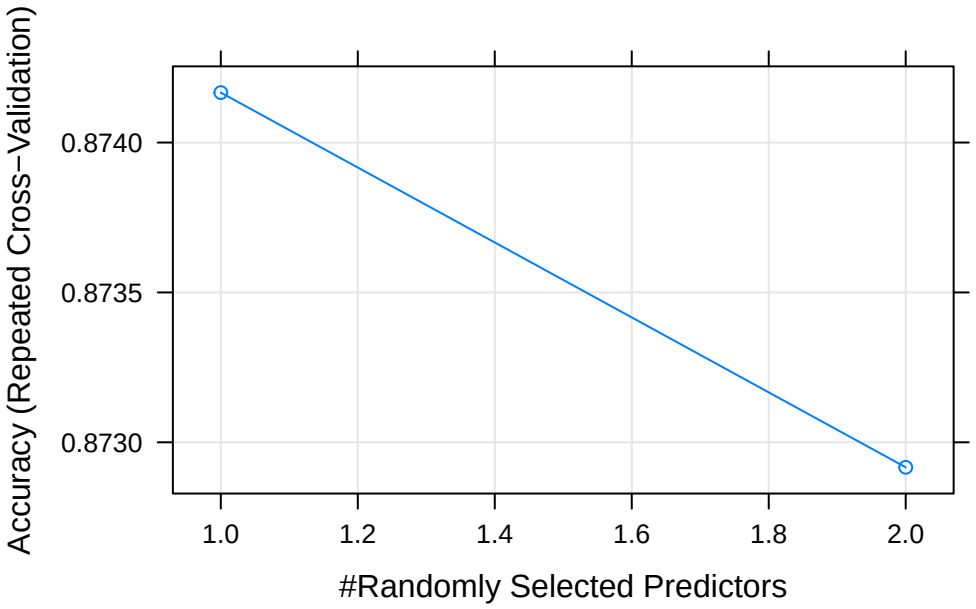
Random Forest

For random Forest we will tune the number of variables randomly split for each of the random samples, as such we will try if only 1 split variable or both variables give a better accuracy

```
# Random Search for best number of variables
control <- trainControl(method = "repeatedcv", number = 10, repeats = 3,
    search = "random")

rf_random <- train(Group ~ ., data = trainNum, method = "rf", metric = "Accuracy",
    tuneLength = 15, trControl = control)
print(rf_random)
```

```
Random Forest

800 samples
  2 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 720, 720, 720, 720, 720, 720, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  1     0.8741667  0.7478208
  2     0.8729167  0.7452588

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 1.
```

```
plot(rf_random)
```



As we can see the accuracy is almost identical with a difference of less that 0,1% and is actually variable depending on the randomly selected values during the training process due to the nature of the random search as such I will try both mtry = 1 and mtry = 2.

```r
# Train the model
rf_model <- randomForest(Group ~ ., data = trainNum, tuneGrid = data.frame(mtry = 1))

# Make predictions on the test data
predictions <- predict(rf_model, testNum)

confusionMatrix(as.factor(predictions), as.factor(y_test))
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 79 20
         1 16 85

               Accuracy : 0.82
                 95% CI : (0.7596, 0.8706)
    No Information Rate : 0.525
```

```
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6398

 Mcnemar's Test P-Value : 0.6171

            Sensitivity : 0.8316
            Specificity : 0.8095
         Pos Pred Value : 0.7980
         Neg Pred Value : 0.8416
             Prevalence : 0.4750
         Detection Rate : 0.3950
   Detection Prevalence : 0.4950
      Balanced Accuracy : 0.8206

       'Positive' Class : 0
```

```r
# Train the model
rf_model <- randomForest(Group ~ ., data = trainNum, tuneGrid = data.frame(mtry = 2))


# Make predictions on the test data
predictions <- predict(rf_model, testNum)

## Manually making the cut at 0,5 of predicition for the groups
## predictions <- ifelse(predictions >= 0.5, 1, 0)

confusionMatrix(as.factor(predictions), as.factor(y_test))
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 80 21
         1 15 84

               Accuracy : 0.82
                 95% CI : (0.7596, 0.8706)
    No Information Rate : 0.525
    P-Value [Acc > NIR] : <2e-16
```

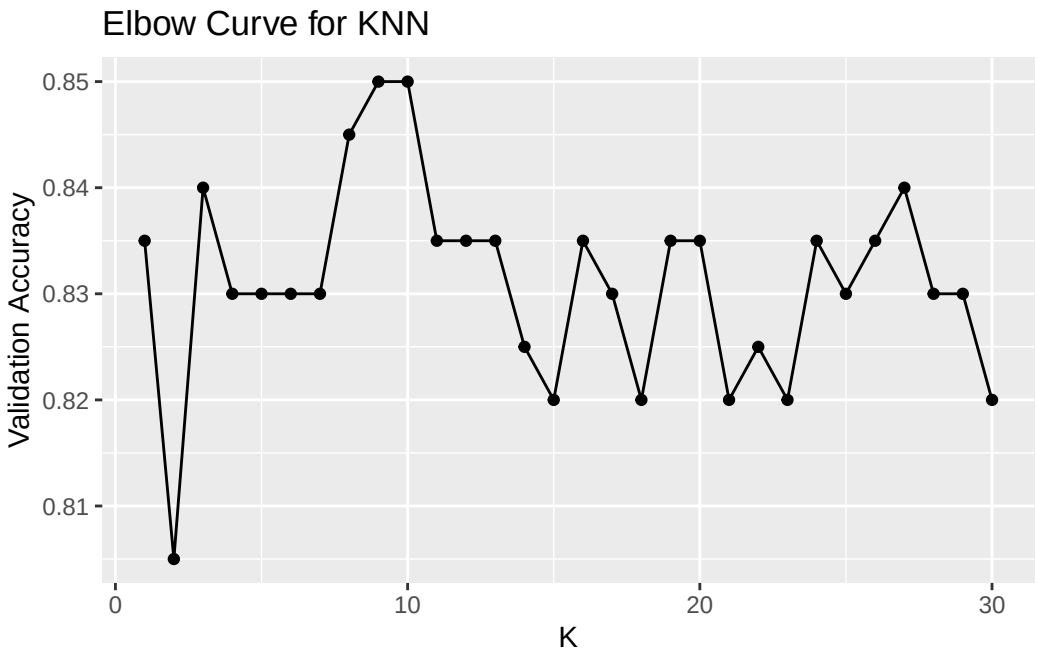
```
                 Kappa : 0.6402

 Mcnemar's Test P-Value : 0.4047

            Sensitivity : 0.8421
            Specificity : 0.8000
         Pos Pred Value : 0.7921
         Neg Pred Value : 0.8485
             Prevalence : 0.4750
         Detection Rate : 0.4000
   Detection Prevalence : 0.5050
      Balanced Accuracy : 0.8211

       'Positive' Class : 0
```

As we can see from the summary tables both fine tuned values lead to approximately the same result of 82% accuracy which is actually lower than the random search was predicting.

KNN - K-nearest neighbour

First we make an graph displaying the accuracy for each k from 1 to 30 to see which K is the most appropriate by giving us the best accuracy.

```r
# Define range of K values to test
k_range <- 1:30

# Empty list to store validation accuracy scores
accuracy_scores <- c()

# Train and evaluate KNN model for each value of K
for (k in k_range) {
    knn <- knn(X_train, X_test, y_train, k = k)
    accuracy_scores <- c(accuracy_scores, sum(knn == y_test)/length(y_test))
}

# Plot the elbow curve
df <- data.frame(k = k_range, accuracy = accuracy_scores)
ggplot(df, aes(x = k, y = accuracy)) + geom_line() + geom_point() + xlab("K") +
    ylab("Validation Accuracy") + ggtitle("Elbow Curve for KNN")
```



As we can see from the graph, k= 9 is the one that give us the most accuracy and most other values have not been as highly accurate.

```
# fit the model
fit = knn(train = X_train, test = X_test, cl = y_train, k = 9)
# fit = knn(xTrain,xTest,yTrain,k=3)

# produce the confusion matrix when numbers are used as class labels we
# might need
confusion = confusionMatrix(as.factor(fit), as.factor(y_test))

confusion
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 82 17
         1 13 88

               Accuracy : 0.85
                 95% CI : (0.7928, 0.8965)
```

```
   No Information Rate : 0.525
   P-Value [Acc > NIR] : <2e-16

                 Kappa : 0.6998

 Mcnemar's Test P-Value : 0.5839

            Sensitivity : 0.8632
            Specificity : 0.8381
         Pos Pred Value : 0.8283
         Neg Pred Value : 0.8713
             Prevalence : 0.4750
         Detection Rate : 0.4100
   Detection Prevalence : 0.4950
      Balanced Accuracy : 0.8506

       'Positive' Class : 0
```

```r
test$predicted_group = fit
test$correct_prediction = test$predicted_group == test$Group

# plot the data points
p = ggplot() + geom_point(data = train, aes(x = X1, y = X2, color = "Training"),
    size = 3) + geom_point(data = test, aes(x = X1, y = X2, color = correct_prediction,
    shape = factor(correct_prediction)), size = 3) + scale_color_manual(values = c("red",
    "blue", "green"), labels = c("Incorrect", "Correct", "Training")) + scale_shape_manual
    16)) + labs(x = "X1", y = "X2", color = "", shape = "") + ggtitle("KNN Classification

# display the plot
print(p)
```

SVM - Support Vector Machines

Since we don't have a linear kernel due to the nature of the data we have a choice of using either a polynomial kernel or a radial kernel, however there is no simple test to differentiate which kernel function has the best performance for this data, as such I would be fine tuning both kernel functions and then see which kernel function has the best performance.

However since the polynomial fine tuning does not converge I will use the non tuned kernels to show that the radial kernel is better suited to the data.

Startin with the polynomial kernel

```r
mdlSVMPoly = svm(Group ~ .,
                 data = train,
                 type = 'C-classification',
                 kernel = 'polynomial',
                 #cost = svmPolynoml_tuned$bestTune$C,
                 #sigma = svmPolynoml_tuned$bestTune$sigma
                 )

# Test model on testing data
yTestPredPoly = predict(mdlSVMPoly, newdata=test)
# yTestPred = mdl %>% predict(xTest)
confusionMatrix(as.factor(yTestPredPoly), as.factor(y_test)) # predicted/true
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 59 18
         1 36 87

               Accuracy : 0.73
                 95% CI : (0.6628, 0.7902)
    No Information Rate : 0.525
    P-Value [Acc > NIR] : 2.343e-09

                  Kappa : 0.4537

 Mcnemar's Test P-Value : 0.0207

            Sensitivity : 0.6211
            Specificity : 0.8286
         Pos Pred Value : 0.7662
         Neg Pred Value : 0.7073
             Prevalence : 0.4750
         Detection Rate : 0.2950
   Detection Prevalence : 0.3850
      Balanced Accuracy : 0.7248

       'Positive' Class : 0
```

As we can see this kernel function only has an average accuracy of 73% which is significanly

lower than the other methods.

```
mdlSVM = svm(Group ~ .,
                 data = train,
                 type = 'C-classification',
                 kernel = 'radial')

# Test model on testing data
yTestPred = predict(mdlSVM, newdata=test)
# yTestPred = mdl %>% predict(xTest)
confusionMatrix(as.factor(yTestPred), as.factor(y_test)) # predicted/true
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 80 20
         1 15 85

               Accuracy : 0.825
                 95% CI : (0.7651, 0.875)
    No Information Rate : 0.525
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.65

 Mcnemar's Test P-Value : 0.499

            Sensitivity : 0.8421
            Specificity : 0.8095
         Pos Pred Value : 0.8000
         Neg Pred Value : 0.8500
             Prevalence : 0.4750
         Detection Rate : 0.4000
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.8258

       'Positive' Class : 0
```

As we can see, the Radial kernel function even when not fine tuned has a significantly better accuracy than the polynomial kernel by almost 10%, such gap would have not been narrowed

by the fine tuning of the parameters alone and as such a Radial kernel function is the most appropriate for this data, as such we will be fine tuning the hyper parameters of the model.

Firstly I will start by fine tuning the common constant C and the sigma parameter which is specific to the radial kernel SVM

```
fitControl = trainControl(## 10-fold CV
                          method = "repeatedcv",
                          number = 10,
                          ## repeated ten times
                          repeats = 10)

param_grid <- expand.grid(C = seq(0.1, 10, length = 10),
                          sigma = seq(0.1, 1, length = 10))


svmRadial_tuned = train(Group ~ .,
                 data = trainNum,
                 method = 'svmRadial',
                 trControl = fitControl,
                 tuneGrid = param_grid)

svmRadial_tuned
```

```
Support Vector Machines with Radial Basis Function Kernel

800 samples
  2 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 10 times)
Summary of sample sizes: 720, 720, 720, 720, 720, 720, ...
Resampling results across tuning parameters:

  C     sigma  Accuracy  Kappa
   0.1  0.1    0.774125  0.5474824
   0.1  0.2    0.804375  0.6082202
   0.1  0.3    0.816750  0.6330858
   0.1  0.4    0.828500  0.6568911
   0.1  0.5    0.838000  0.6759468
   0.1  0.6    0.841625  0.6832838
```

0.1	0.7	0.845875	0.6917003
0.1	0.8	0.847625	0.6951959
0.1	0.9	0.848625	0.6972030
0.1	1.0	0.851000	0.7020040
1.2	0.1	0.814625	0.6288121
1.2	0.2	0.852000	0.7037885
1.2	0.3	0.863125	0.7263778
1.2	0.4	0.865750	0.7317670
1.2	0.5	0.868250	0.7368202
1.2	0.6	0.868625	0.7376100
1.2	0.7	0.870375	0.7411153
1.2	0.8	0.872875	0.7461434
1.2	0.9	0.873375	0.7471350
1.2	1.0	0.874500	0.7493192
2.3	0.1	0.825875	0.6514549
2.3	0.2	0.858125	0.7162464
2.3	0.3	0.863500	0.7271185
2.3	0.4	0.867250	0.7348227
2.3	0.5	0.869500	0.7393521
2.3	0.6	0.871750	0.7438767
2.3	0.7	0.873375	0.7471450
2.3	0.8	0.875875	0.7520855
2.3	0.9	0.878250	0.7567092
2.3	1.0	0.880625	0.7612476
3.4	0.1	0.834375	0.6686225
3.4	0.2	0.862625	0.7252902
3.4	0.3	0.864750	0.7297386
3.4	0.4	0.867500	0.7353467
3.4	0.5	0.869875	0.7401243
3.4	0.6	0.875250	0.7509467
3.4	0.7	0.875000	0.7503502
3.4	0.8	0.878125	0.7563653
3.4	0.9	0.881875	0.7636264
3.4	1.0	0.886500	0.7726123
4.5	0.1	0.843875	0.6875661
4.5	0.2	0.864000	0.7280358
4.5	0.3	0.865750	0.7317767
4.5	0.4	0.867375	0.7350838
4.5	0.5	0.871500	0.7433916
4.5	0.6	0.873500	0.7474097
4.5	0.7	0.876250	0.7526861
4.5	0.8	0.882250	0.7643886
4.5	0.9	0.887375	0.7743615

4.5	1.0	0.891750	0.7829283
5.6	0.1	0.847375	0.6944760
5.6	0.2	0.865125	0.7302821
5.6	0.3	0.867000	0.7342874
5.6	0.4	0.868625	0.7375996
5.6	0.5	0.873250	0.7469459
5.6	0.6	0.874375	0.7490668
5.6	0.7	0.880000	0.7600253
5.6	0.8	0.886000	0.7716813
5.6	0.9	0.892750	0.7850086
5.6	1.0	0.893875	0.7871524
6.7	0.1	0.851625	0.7029540
6.7	0.2	0.865000	0.7300621
6.7	0.3	0.867875	0.7360572
6.7	0.4	0.870375	0.7411338
6.7	0.5	0.873750	0.7479047
6.7	0.6	0.875500	0.7512072
6.7	0.7	0.884250	0.7683498
6.7	0.8	0.890125	0.7798265
6.7	0.9	0.893625	0.7867012
6.7	1.0	0.896125	0.7916334
7.8	0.1	0.853250	0.7062708
7.8	0.2	0.866875	0.7338577
7.8	0.3	0.868000	0.7363082
7.8	0.4	0.871375	0.7431444
7.8	0.5	0.874125	0.7485991
7.8	0.6	0.877000	0.7541060
7.8	0.7	0.886500	0.7726783
7.8	0.8	0.892375	0.7842332
7.8	0.9	0.895500	0.7904077
7.8	1.0	0.896250	0.7918742
8.9	0.1	0.855750	0.7113351
8.9	0.2	0.867250	0.7346342
8.9	0.3	0.868500	0.7372914
8.9	0.4	0.871125	0.7426332
8.9	0.5	0.873750	0.7478349
8.9	0.6	0.879375	0.7587404
8.9	0.7	0.889375	0.7783491
8.9	0.8	0.891875	0.7831710
8.9	0.9	0.896125	0.7916140
8.9	1.0	0.895875	0.7911165
10.0	0.1	0.855875	0.7115987
10.0	0.2	0.867375	0.7349059

```
  10.0  0.3    0.869375  0.7390316
  10.0  0.4    0.871250  0.7428916
  10.0  0.5    0.873750  0.7477788
  10.0  0.6    0.882125  0.7641143
  10.0  0.7    0.890875  0.7812984
  10.0  0.8    0.893750  0.7869032
  10.0  0.9    0.896625  0.7926159
  10.0  1.0    0.895875  0.7910950

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.9 and C = 10.
```

This is the non-converging polynomial SVM where I would fine tuning the common constant C and the specific fine tuning parameters from the polynomial function, degree and scale.

```
# Set up the tuning grid
tuningGrid <- expand.grid(degree = c(2, 3, 4), C = 10^(0:2), scale = c(0.1,
    1, 10))

# Set up the control parameters
fitControl <- trainControl(method = "repeatedcv", number = 10, repeats = 3,
    classProbs = TRUE)
## I had to convert the variable names from scientific notation to
## decimal because it is the only way for this to work for this kernel
trainNumTest = trainNum

levels(trainNumTest$Group) <- make.names(levels(trainNumTest$Group))


# Train the SVM with polynomial kernel
svmPoly_tuned <- train(Group ~ ., data = trainNumTest, method = "svmPoly",
    tuneGrid = tuningGrid, trControl = fitControl, minstep = 1e-08)


# Print the results
svmPoly_tuned
```

Now that we have the fine tuned parameters I will be using these to fit a model for the Radial kernel.

```
mdlSVM = svm(Group ~ .,
                 data = train,
```

```
                  type = 'C-classification',
                  kernel = 'radial',
                  cost = svmRadial_tuned$bestTune$C,
                  sigma = svmRadial_tuned$bestTune$sigma)

# Test model on testing data
yTestPred = predict(mdlSVM, newdata=test)
# yTestPred = mdl %>% predict(xTest)
confusionMatrix(as.factor(yTestPred), as.factor(y_test)) # predicted/true
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 83 23
         1 12 82

               Accuracy : 0.825
                 95% CI : (0.7651, 0.875)
    No Information Rate : 0.525
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.651

 Mcnemar's Test P-Value : 0.09097

            Sensitivity : 0.8737
            Specificity : 0.7810
         Pos Pred Value : 0.7830
         Neg Pred Value : 0.8723
             Prevalence : 0.4750
         Detection Rate : 0.4150
   Detection Prevalence : 0.5300
      Balanced Accuracy : 0.8273

       'Positive' Class : 0
```

As we can see from the fine tuned model summary, the usage of C and sigma managed to improve the model by approximately 0,5% making this model slightly better than the prevviously untuned model.

Question B.5

In our data we have 475 observations belonging to the group 0 and the remaining 525 belonging to group 1, since there is only a small difference in number of observations between both groups, accuracy will be a sensible metric to compare all 3 different methods.

From the 3 final fine tuned models we can see that the Radial kernel SVM has an accuracy of 83%, the KNN with k=9 has an accuracy of 85% and the random forest has an accuracy of 82%, as such with overall accuracy in mind we can determine that KNN with 9 clusters is the better predictive model.

This also holds true if we were to compare the specificity of the 3 models as KNN has the highest value of 83,81% compared to the 80,00% from random forest and 78,10% from SVM.

However for sensitivity the SVM has 88,42% which beats both the SVM 86,32% and the random forest 84,32%.

To conclude unless there was a specific need to focus on the specificity of the model which is not given for this dataset the best method for classification is the K-nearest neighbour classification.