SDS2_Project

Simone Boesso

2023-06-16

Bayesian Analysis of the risk of being a survivor on the Titanic
Everyone knows what happened during the Titanic accident, and its data are continuously studied, during my analysis I want to find what were the most influencing features for surviving and using a logistic regression model tuned with **Monte Carlo Markow Chain method** I want to predict if a person would survive or not.
The objective of the analysis is to find the model that maximize the accuracy in predicting if a passenger would survive or not.

The dataset analyzed for this analysis is taken from CRAN package Titanic.

Two **statistical models for binary classification** are applied:
-Logistic Regression (with Logit link function) with 5 features
-Logistic Regression (with Logit link function) with 2 features

As I said the parameters of the models are tuned with Monte Carlo Markov Chain methods, and a Bayesian analysis is applied to them.

The analysis is divided in **4 parts:**

1. **Exploratory Data Analysis**

2. For each of the models:
   - **Parameter tuning** with Monte Carlo Markov Chain
   - Visualization of the MCMC with **Traceplots**, **Density plots**, **Autocorrelation plots**, **Correlation matrix** of the parameters
   - Evaluation of the model on new data
   - Comparison between **Bayesian approach** and **Frequentist approach**
3. Choice of the best model among the 4 analyzed
4. Conclusions

Now, let’s start.

Exploratory Data Analysis and Feature Engineering

The dataset is composed of 10 variables:

-**PassengerId**: ID number of every passenger
-**Survived** : the label if has survived or not, 0 = No, 1 = Yes
-**Pclass** : the Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd
-**Name** : the name
-**Sex** : the sex
-**Age** : the age
-**SibSp**: # of siblings / spouses aboard the Titanic
-**Parch** : # of parents / children aboard the Titanic
-**Ticket** : Ticket number
-**Fare** : Passenger fare
-**Cabin**: Cabin number
-**Embarked** : Port of Embarkation, C = Cherbourg, Q = Queenstown, S = Southampton

Data cleaning

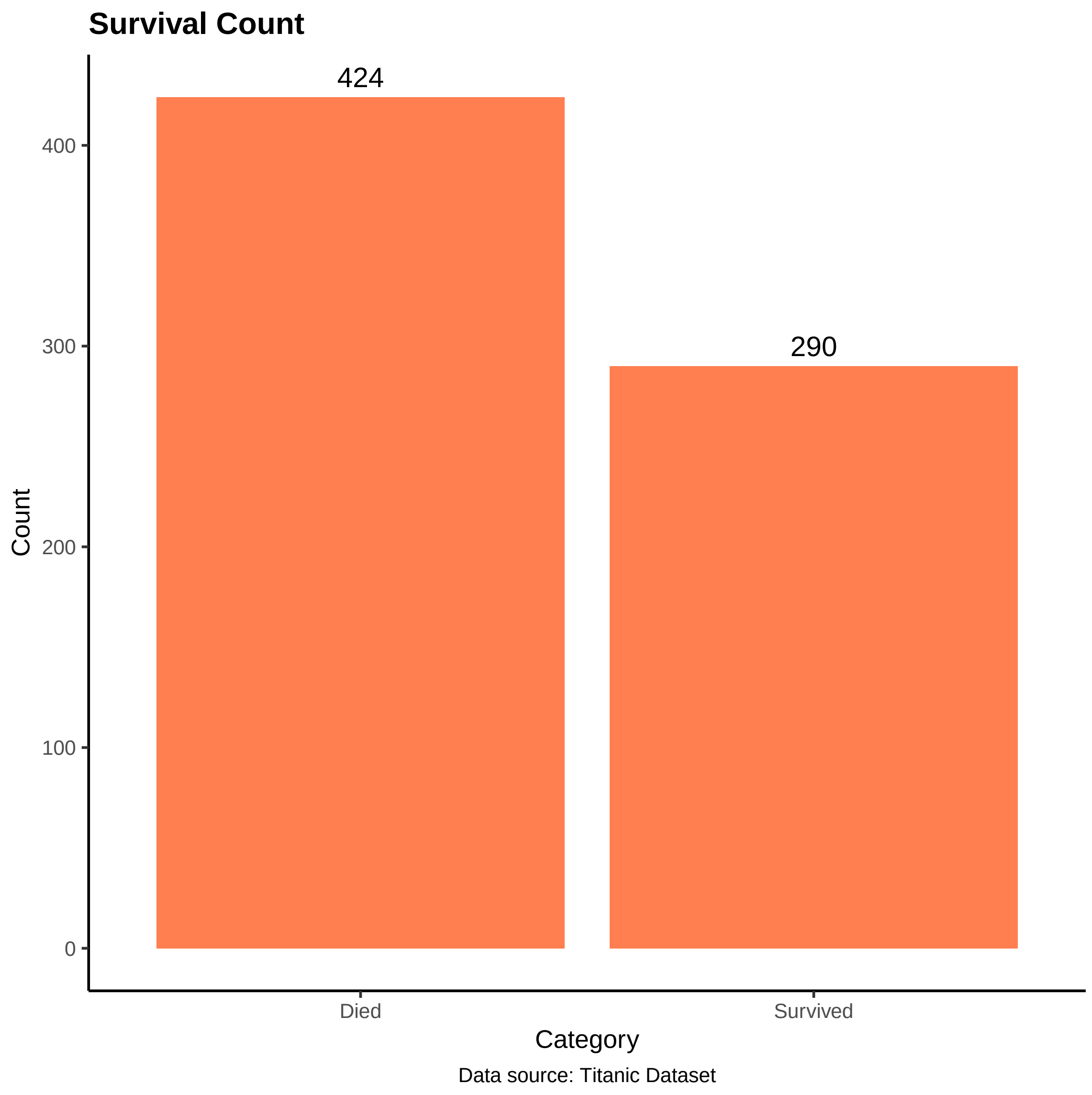
I start the pre-processing of the data.

- The **Cabin column** has many NA values, I will drop it.

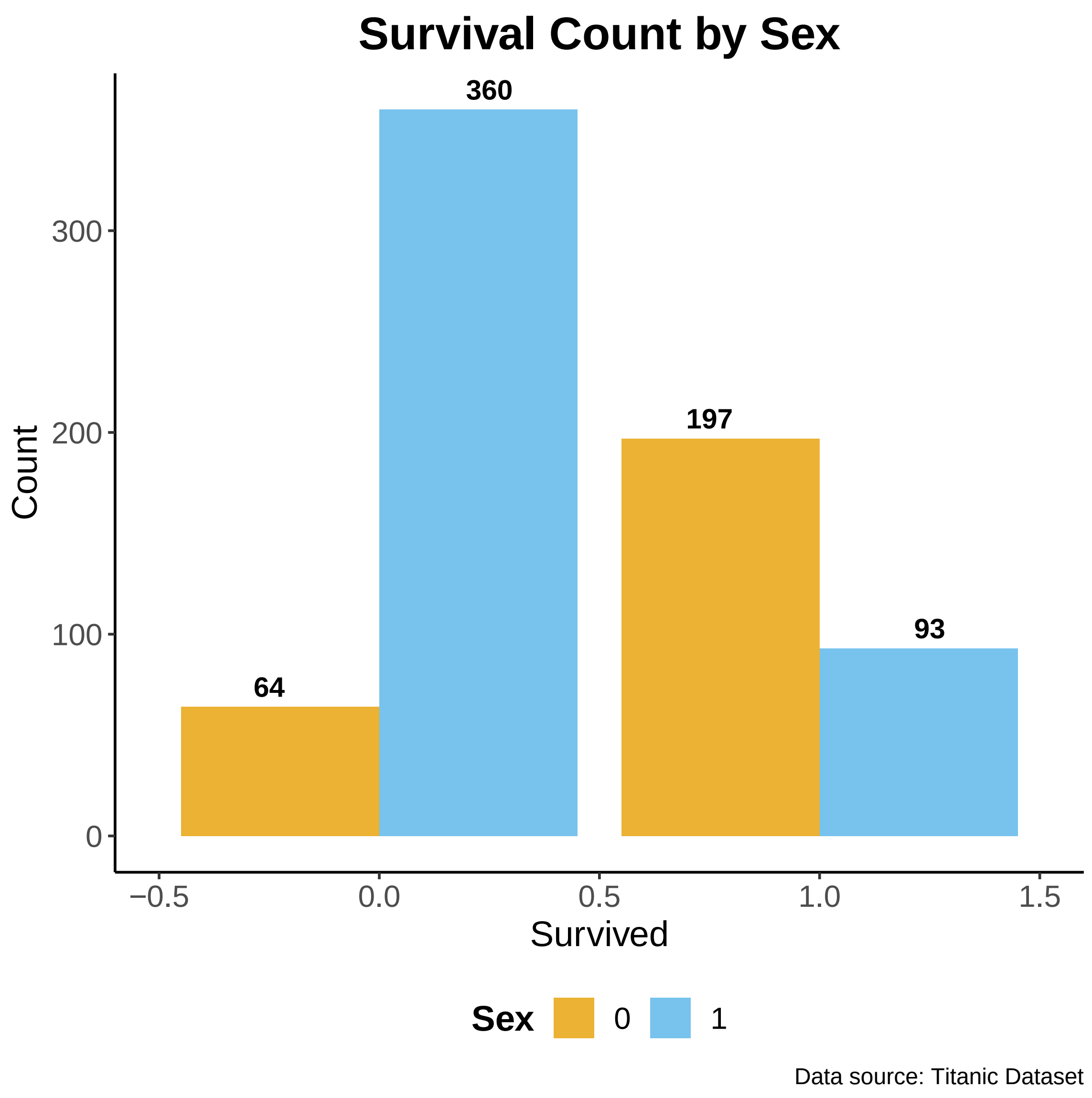
- The **PassengerID** is a unique identifier for the records, I will drop it.
  I will select the remaining columns and I omit the Na.

Data Visualization and Exploration

Let's start with a first count of how many passengers died and survived

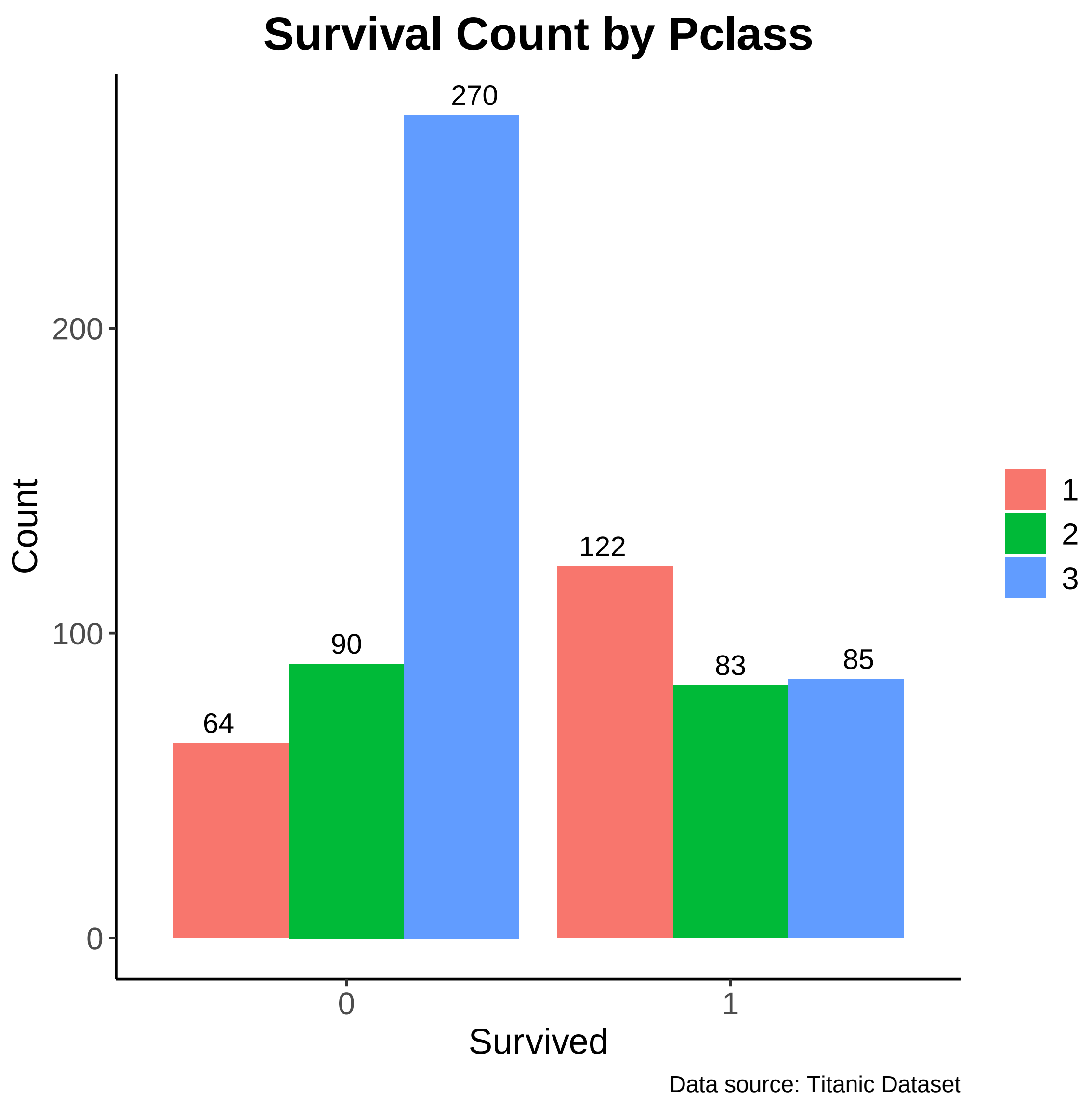


Let's see if the sex is relevant for our analysis:



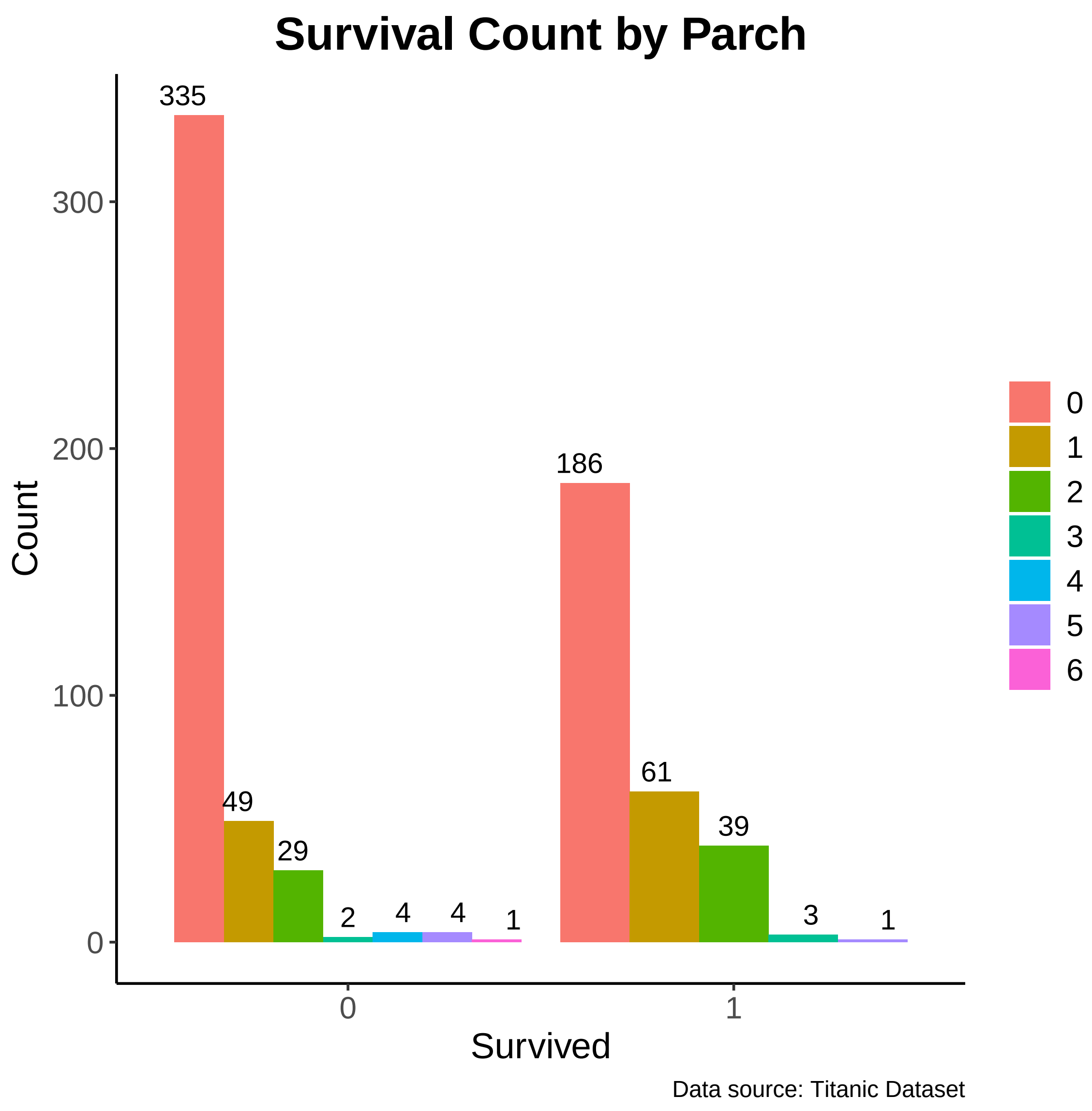
Yep a lot, more females survived than males.

Now let's look at the type of class they were in:

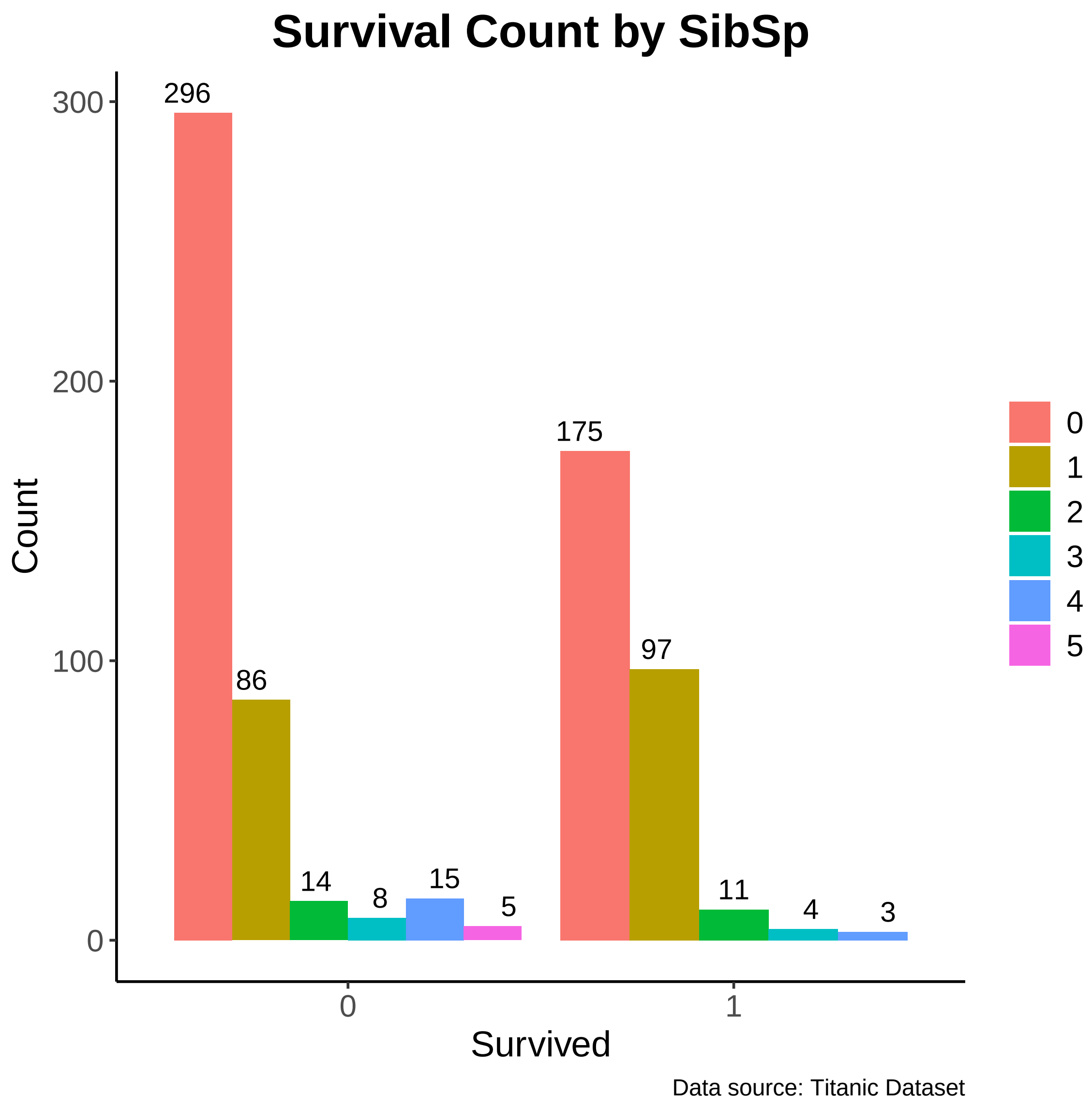


Being in **1st class** was a big advantage.

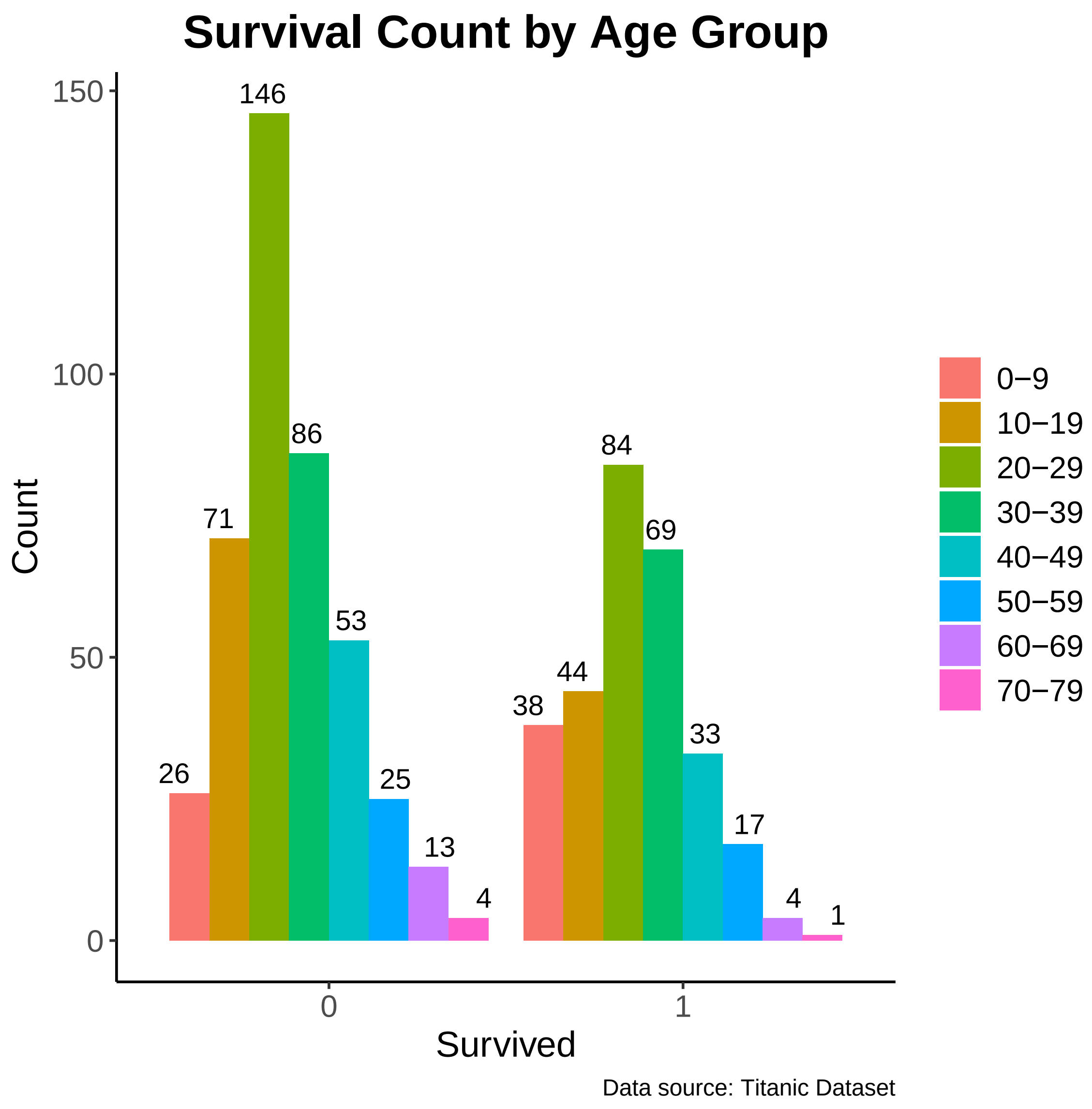
Let's do the same but for the **Parch attribute**: # of parents / children aboard the Titanic



So it doesn't seems as relevant as the Pclass and Sex let's do the same for **SibSp attribute**: # of siblings / spouses aboard the Titanic

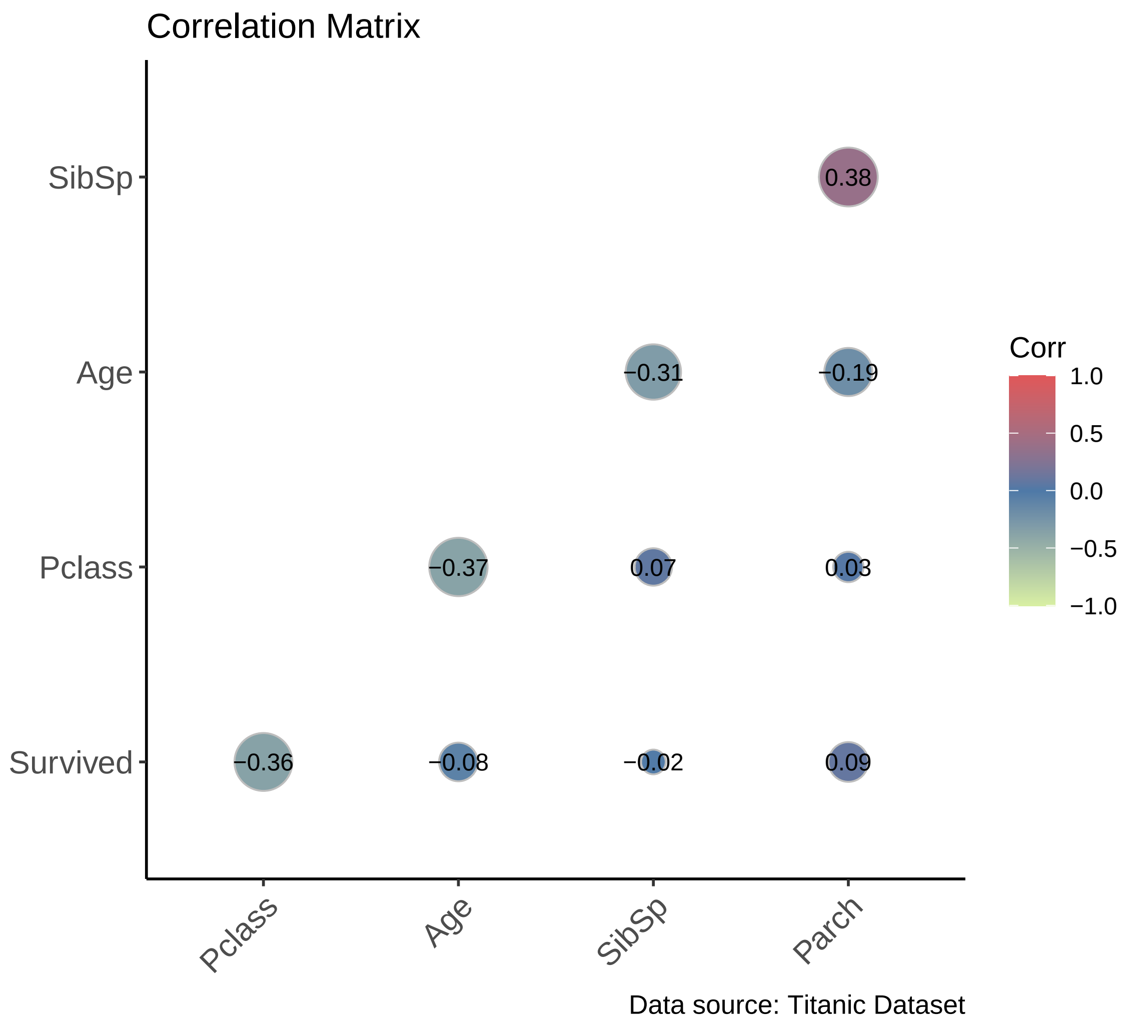


Finally, I'm looking at the **age attribute** grouping by 10 years



Well, as before I don't see it as really relevant like sex and Pclass

Now, let's look at the **correlation matrix** of my variables:



Looking at the plot we see there are some linear dependencies. Conversely, as the "Pclass" value decreases (from 3rd class to 2nd class or 1st class), the likelihood of being a "survived" passenger (i.e., 1) tends to increase.

Standardization of the variables: Before the inferential analysis, I proceed to standardize my input variables, in fact they have different unit of measure.

The Inferential Analysis

As I said, the purpose of the analysis is to forecast whether a person will survive or not maximizing the accuracy.
Before starting, I split the database into a train and eval parts. I randomly split the dataset into a 80% train-set and 20% eval-set and I check their proportions of GTs.

```
## label
##         0         1
## 0.5874126 0.4125874
```

```
## label
##         0         1
## 0.6197183 0.3802817
```

Now i can start with the analysis.

Modelling

Model: Logistic Regression (Logit)

Gibbs sampling of regression models use normal prior distributions over the weights, so the **assumptions** are the following:

$$Y_i \sim Ber(logit(p_i))$$

$$logit(p_i) = log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{1_i} + \beta_2 x_{2_i} + \beta_3 x_{3_i} + \beta_4 x_{4_i} + \beta_5 x_{5_i}$$

the prior parameters β will have a prior distribution with $\mu = 0$ and $\sigma = 0.0001$

$$\beta_i \sim N(0, 0.0001)$$

for $i = 1, 2, 3, 4, 5$.

Implementation of the model with RJags For the implementation i use Rjags with 2 chains and a burnin of 1000 iterations.

```
set.seed(123)

#Write the model in BUGS language
N = length(train$Survived)
model_func <- function(){
  #likelihood
  for (i in 1:N){
    y[i] ~ dbern(p[i])
    #The logit function can be used to convert any real number to a probability.
    logit(p[i]) =  beta0 + beta1*x1[i] + beta2*x2[i] + beta3*x3[i] + beta4*x4[i] +beta5*x5[i]
  }
  #prior beta parameters ( a non informative prior)
  beta0 ~ dnorm(0, 0.0001)
  beta1 ~ dnorm(0, 0.0001)
  beta2 ~ dnorm(0, 0.0001)
  beta3 ~ dnorm(0, 0.0001)
  beta4 ~ dnorm(0, 0.0001)
  beta5 ~ dnorm(0, 0.0001)

}
```

```
set.seed(123)
model = jags(data = data_sim,
             model.file = model_func,
             parameters.to.save = params,
             n.chains = 2,
             n.iter = 10000,
             n.burnin = 1000,
             n.thin = 10)
```

```
## module glm loaded
```

These are the results we get:

	mean.vect	sd.vect	Quantile_0.25	Quantile_0.50	Quantile_0.75	R.hat	N_eff
Beta_0	-0.506	0.115	-0.579	-0.506	-0.430	1.001	1800
Beta_1-Pclass	-1.110	0.130	-1.199	-1.112	-1.020	1.001	1800
Beta_2-Sex	-1.287	0.123	-1.370	-1.285	-1.205	1.002	1200
Beta_3-Age	-0.653	0.132	-0.741	-0.650	-0.562	1.000	1800
Beta_4-SibSp	-0.292	0.134	-0.382	-0.291	-0.195	1.001	1800
Beta_5-Parch	0.012	0.113	-0.063	0.011	0.090	1.000	1800
Deviance	518.303	3.575	515.767	517.706	520.053	1.001	1800

The **DIC** (Deviance Information Criterion) is used for comparing different models based on their goodness of fit and complexity. Similar to AIC, DIC incorporates a penalty for model complexity, aiming to avoid overfitting. It's represented as

$$DIC = D_{\hat{\theta}}(y) + 2p_D = \hat{D}_{avg}(y) + p_D = 2\hat{D}_{avg}(y) - D_{\hat{\theta}}(y)$$

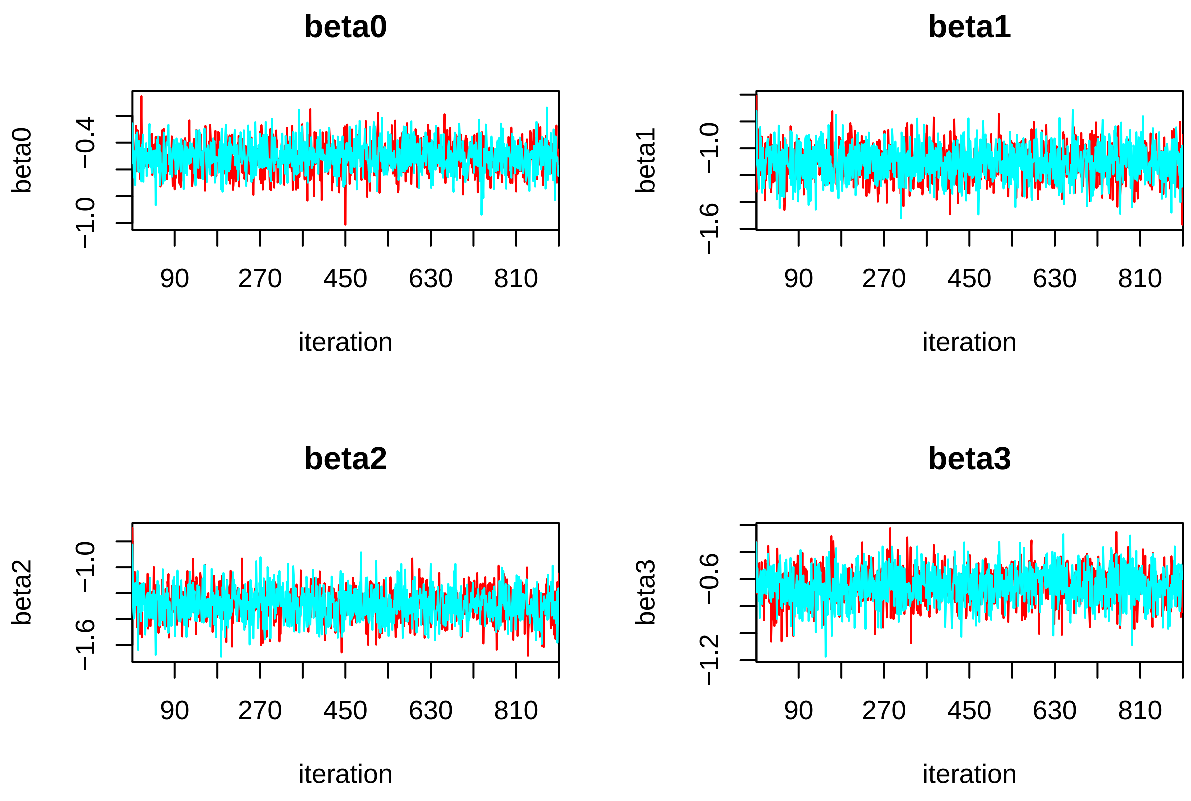
Where
$D_{\hat{\theta}}(y) = -2logf(y, \hat{\theta}(y))$,
$\hat{D}_{avg}(y) \approx \frac{1}{M}\sum_j -2logf(y|\theta^{(j)})$
and p_D is the effective number of parameters.

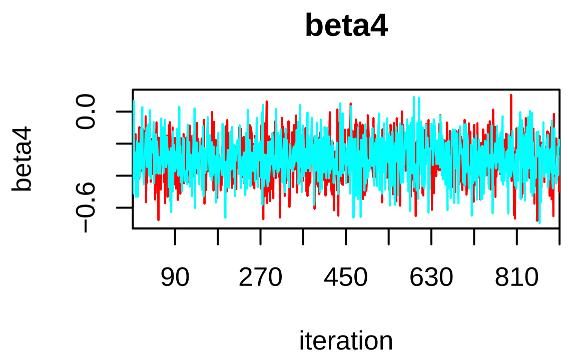
DIC's asymptotic approximation becomes more reliable as the sample size increases, making it particularly useful for larger datasets
The minimum DIC estimates the "best" model in the same spirit as AIC.
When comparing different models using DIC, the model with the lower DIC value is preferred. This corresponds to a model with a better balance between goodness of fit (larger likelihood) and complexity (smaller number of parameters).

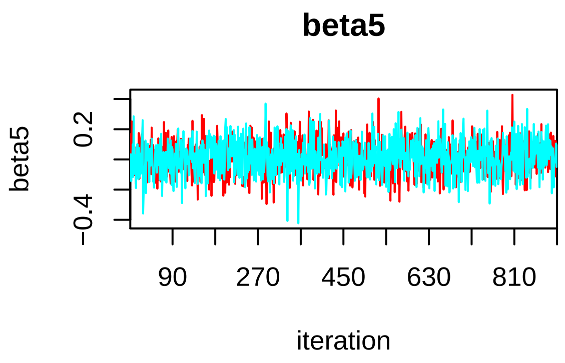
The DIC is a comparative index, so I will compare it with all the models in order to choose the best one.

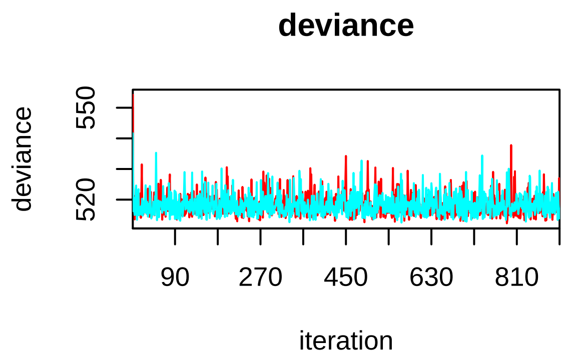
If the Rhat values are substantially larger than one, the chains haven't mixed properly and the posterior estimates cannot be trusted. We can also visually assess how well the chains have mixed using the traceplots

Trace-plots Let's observe now the stationary regions. When the Gibbs sampler has converged to the posterior distribution, it means that the samples of each parameter exhibit less fluctuation or variability, indicating a more stable estimate. In particular the 2 chains give the possibility to see if given different initialization of the parameters, we reach the same target distribution.





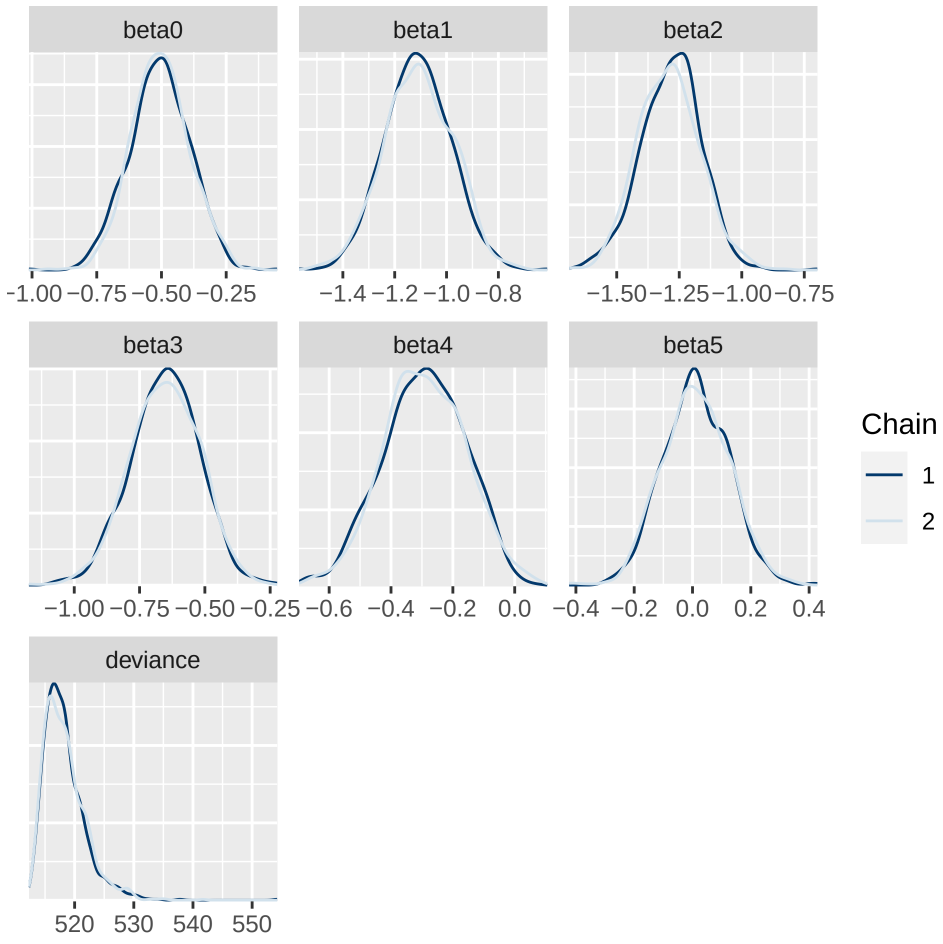




The traceplot explains the pattern followed by the parameter for every iteration of the Markov Chains. From the plot we can see that the processes look stationary: it means that the trend of the parameter, with infinite iterations, becomes constant.

Density From the density plot we can observe the distribution of the estimated parameters.

The mean of the distribution represents the estimate of the parameters ("$\mu.vect$") since the distributions are quite symmetric and so the medians are not needed.



The next step is to study the probability of the parameters with the **Credible Intervals**.
I use **Highest Posterior Density interval (HPD)**.

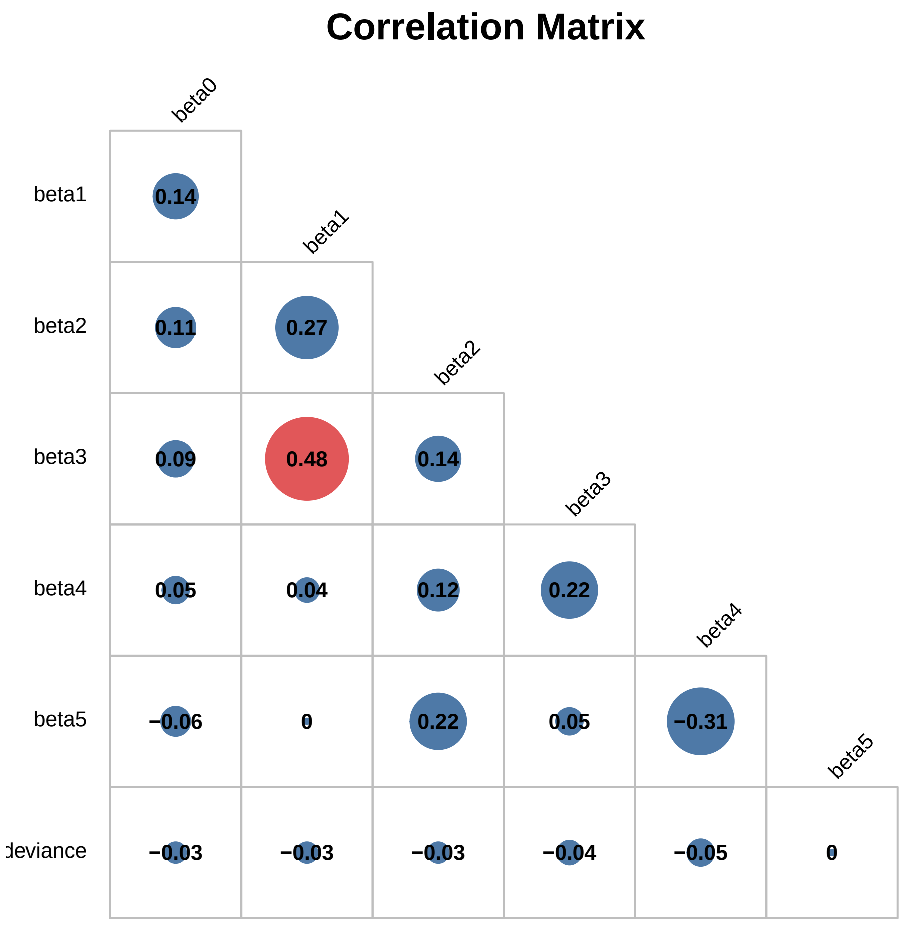
The particularity of the HPD interval is that the posterior density for every point in the confidence region I_α is higher than the posterior density for any point outside of this set.

The HPD intervals at level $\alpha = 0.05$ are the following

	lower	upper
beta0	-0.7148321	-0.2630843
beta1	-1.3583719	-0.8633822
beta2	-1.5410792	-1.0741599
beta3	-0.9032111	-0.4124246
beta4	-0.5472041	-0.0507275
beta5	-0.2069794	0.2229874
deviance	513.0652453	525.2029400

The intervals effectively contain the values of the estimated parameters.

Correlation between parameters Above we've seen the plots regarding the behavior of the Markov Chains from different point of view.
Furthermore, looking closely to the graphs, we can notice that the parameters follow very similar patterns and have a similar distribution.
These aspects may suggest us to look at the correlation between them and to see whether these values are high:



As expected, there is a correlation between them!

Approximation Error To evaluate the approximation error I use the **MCSE** estimate: the MCSE (Monte Carlo Standard Error) is an estimate of the inaccuracy of Monte Carlo samples, usually regarding the expectation of posterior samples from Monte Carlo Markov Chain algorithms.

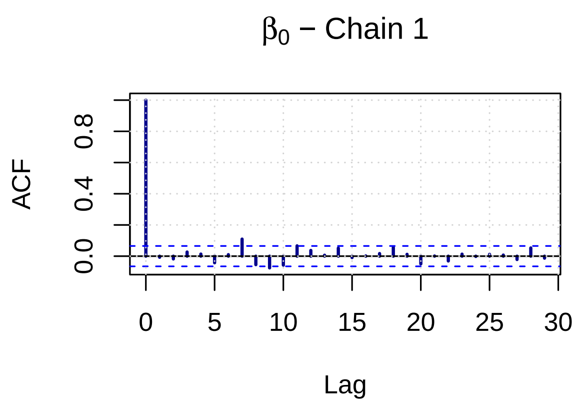
I estimated the MCSE with the *MCSE* function from LaplacesDemon package obtaining the following results:

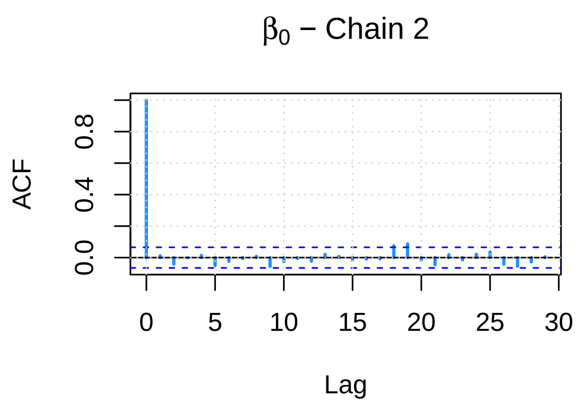
	Approximation_Error
beta_0	0.0038818
beta_1	0.0038103
beta_2	0.0037832
beta_3	0.0043308
beta_4	0.0040950
beta_5	0.0037136

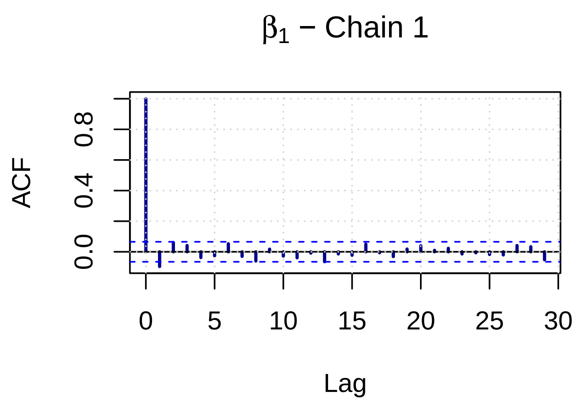
When the MCSE is close to 0, it suggests that the Monte Carlo sampling procedure has converged, and the estimates are stable. It implies that the algorithm has generated a sufficient number of samples to approximate the posterior distribution accurately, resulting in reliable parameter estimates.

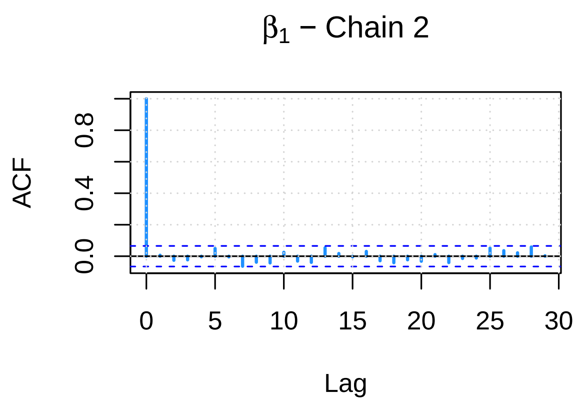
Auto-correlation

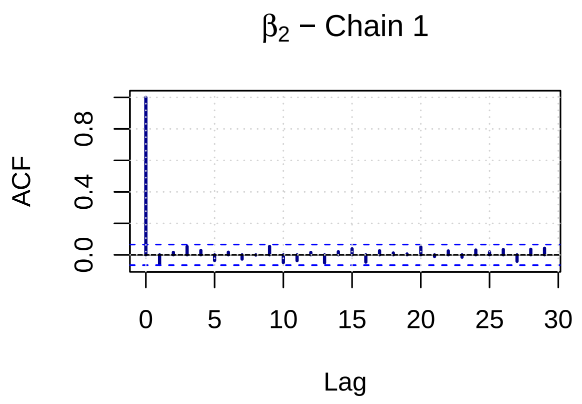
In general, lower autocorrelation values are preferred, as they indicate a higher degree of independence between consecutive samples in the chain. This indicates better mixing and convergence.For the lag1 autocorrelation, values close to 0 or within the range of -0.1 to 0.1 are generally considered acceptable. Larger absolute values of autocorrelation, such as above 0.2, may indicate slow mixing and convergence

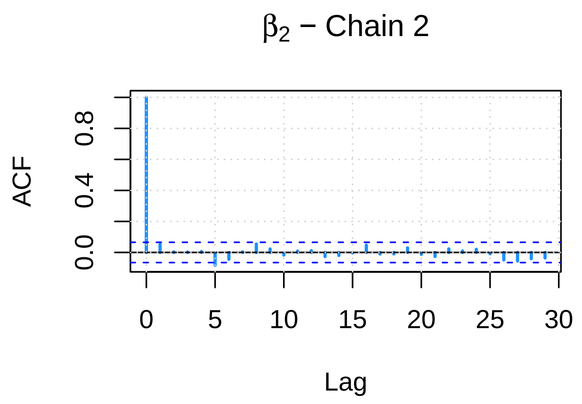


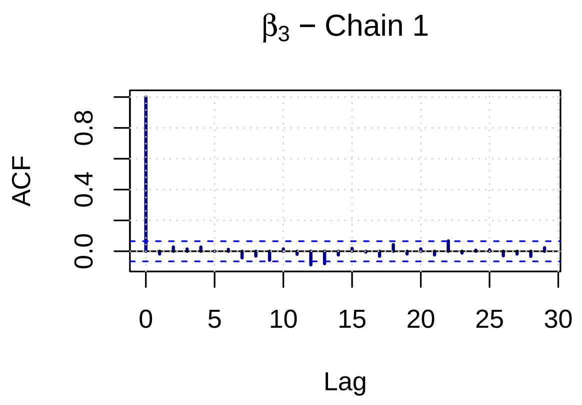


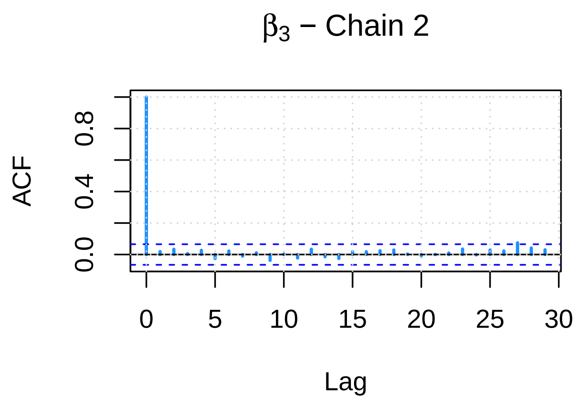

β_1 − Chain 1
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag

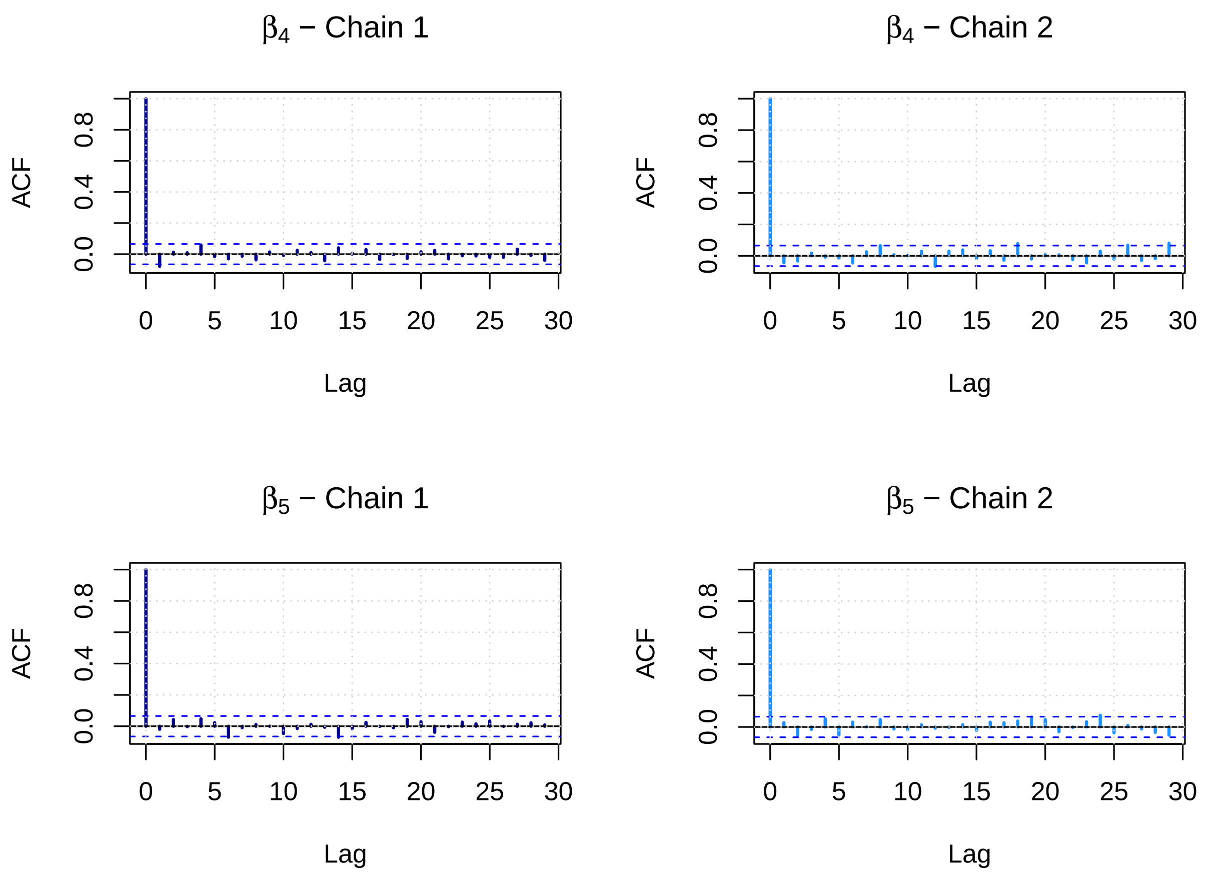

β_1 − Chain 2
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag


β_2 − Chain 1
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag


β_2 − Chain 2
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag


β_3 − Chain 1
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag


β_3 − Chain 2
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag



```
autocorr.diag(as.mcmc(model))
```

```
##                 beta0        beta1        beta2         beta3        beta4
## Lag 0     1.000000000  1.000000000  1.000000000  1.0000000000  1.000000000
## Lag 10    0.002489458 -0.044906868 -0.004093179  0.0004957907 -0.061158562
## Lag 50   -0.046268262  0.011666579 -0.058905869 -0.0123433207 -0.016281160
## Lag 100  -0.041179688 -0.001505898 -0.032826395  0.0094277711 -0.003136480
## Lag 500   0.006959112 -0.016744945 -0.033435199 -0.0110580005 -0.009182178
##                 beta5     deviance
## Lag 0     1.000000000 1.0000000000
## Lag 10    0.004021263 0.0053178726
## Lag 50   -0.013763697 0.0004087712
## Lag 100  -0.031713204 0.0026014211
## Lag 500   0.043440546 0.0100753055
```

so we get quite good samples that are expressing the assumption of being independent with larger lag!

Geweke Diagnostic

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    beta0    beta1    beta2    beta3    beta4    beta5 deviance
## -0.41910 -0.37770  0.65926 -0.03605  1.10347 -2.39083  0.21125
```

This is based on a test for equality of the means of the first and last part of a Markov Chain (by default the first 10% and the last 50%).
The null hypothesis in the test is that two parts of the chain come from the same distribution, and to study it it's used a mean difference test.
If the two averages are equal it is likely that the chain will converge.

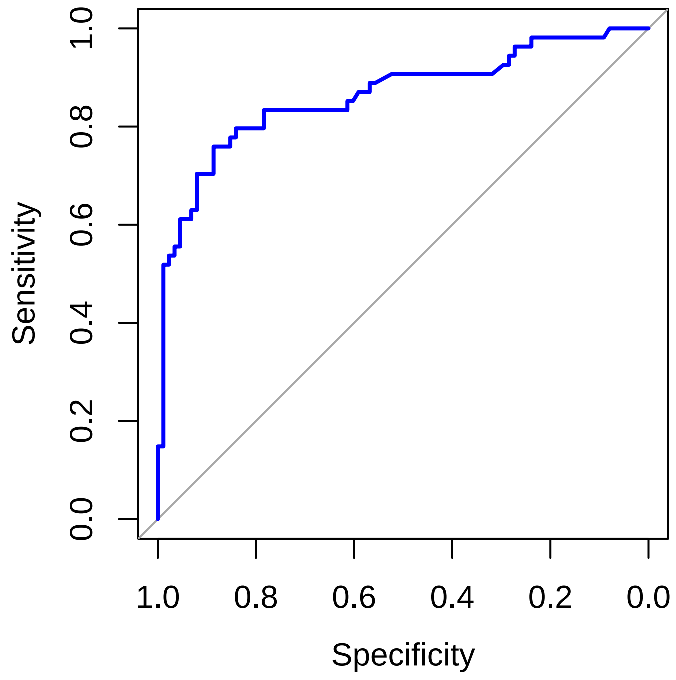
The test statistic is a standard Z-score: the difference between the two sample means divided by its estimated standard error adjusted for autocorrelation.
If the Z-scores are close to zero and within a reasonable range (-2 to 2), it generally indicates convergence. But even if the Geweke test result is slightly outside the typical range, it's important to consider it alongside other convergence diagnostics and visualizations to form a comprehensive assessment of the MCMC results.

	Z.score_Chain1	Z.score_Chain2
Beta_0	0.7100	-0.41910
Beta_1	0.7147	0.37770
Beta_2	0.2883	0.65926
Beta_3	0.6285	-0.03605
Beta_4	0.3892	1.10347
Beta5	0.1774	-2.39083
deviance	1.9707	0.21125

Evaluation time for the bayesian linear regression model Now that the parameters have been estimated, it's time to do the first forecast on the eval set:

```r
#Parameters
beta0 = model$BUGSoutput$summary["beta0", "mean"]
beta1 = model$BUGSoutput$summary["beta1", "mean"]
beta2 = model$BUGSoutput$summary["beta2", "mean"]
beta3 = model$BUGSoutput$summary["beta3", "mean"]
beta4 = model$BUGSoutput$summary["beta4", "mean"]
beta5 = model$BUGSoutput$summary["beta5", "mean"]




x = eval[2:6]
y = eval[,1]
xTb = beta0 + beta1*x[1] + beta2*x[2] + beta3*x[3] + beta4*x[4] +  beta4*x[5]




sig_fun = function(xTb){
  probs = rep(NA, length(xTb[,1]))
  for(i in 1:length(xTb[,1])){
    probs[i] = 1/(1+exp(-xTb[i,1]))
  }
  return (probs)
}


predictions_t = function(probs, t){
  predictions = rep(NA, length(probs))
  for(i in 1:length(probs)){
    if(probs[i]>t)
      predictions[i] = 1
    else
      predictions[i] = 0
  }
  return(predictions)
}

probs = sig_fun(xTb)
roc <- roc(y, probs)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```



```
##
## Call:
```

```
## roc.default(response = y, predictor = probs)
##
## Data: probs in 88 controls (y 0) < 54 cases (y 1).
## Area under the curve: 0.8633
```

Before selecting the best threshold, it's good to observe the ROC curves for having an idea of the performance of our model.
The ROC curve plots **Sensitivity** (true positive rate) against the **Specificity** (true negative rate)
This seems good, and we can get an estimate of its average performance using the Auroc 0.8608.
Since we are dealing with probabilities, the obvious choice is to set the threshold to 0.5.
However in some context this is not the best idea, so we take the threshold that maximize the accuracy with our training dataset.

Now i find the threshold that maximizes the accuracy.

```
## [1] "The maximum accuracy is:"
```

```
## [1] 0.84
```

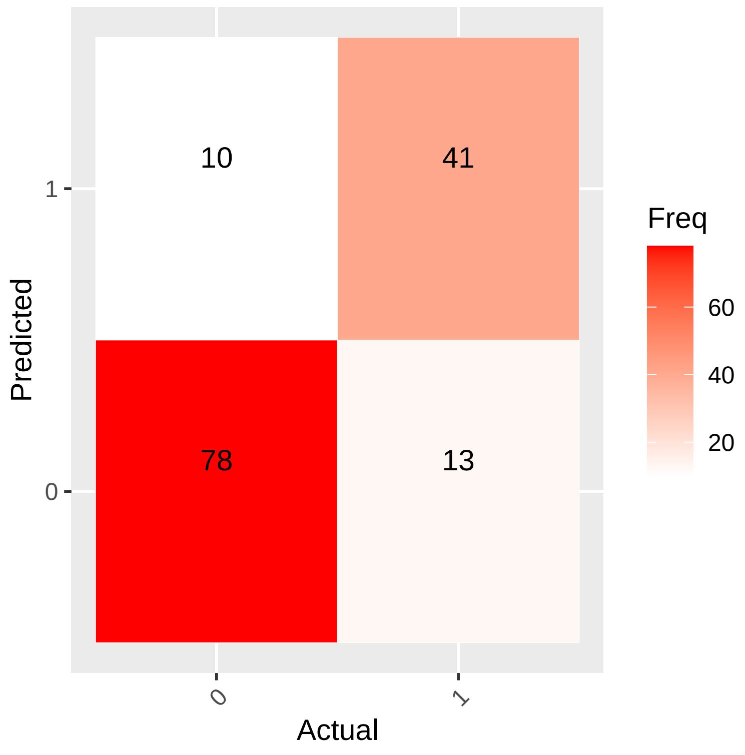
```
## [1] "The corresponding threshold is:"
```

```
## [1] 0.55
```

Confusion Matrix

```
## Precision
##      0.86
```

```
## Recall
##   0.89
```



Comparison with the Frequentist approach After observing the behaviour and the goodness of the parameters, it's time to compare the obtained model with the estimated parameters of the frequentist approach.
In order to do that I decide to use the *glm* package from R.

```
freq = glm(Survived ~ Pclass + Sex +  Age +  SibSp + Parch, family = binomial(link = "logit"),data=trai
```

Traning phase

```
##    Variable Coefficient
## 1 Intercept     -0.4979
## 2    Pclass     -1.0948
## 3       Sex     -1.2704
## 4       Age     -0.6462
## 5     SibSp     -0.2816
## 6     Parch      0.0122
```

Residual Deviance: 512.2 and AIC: 524.2

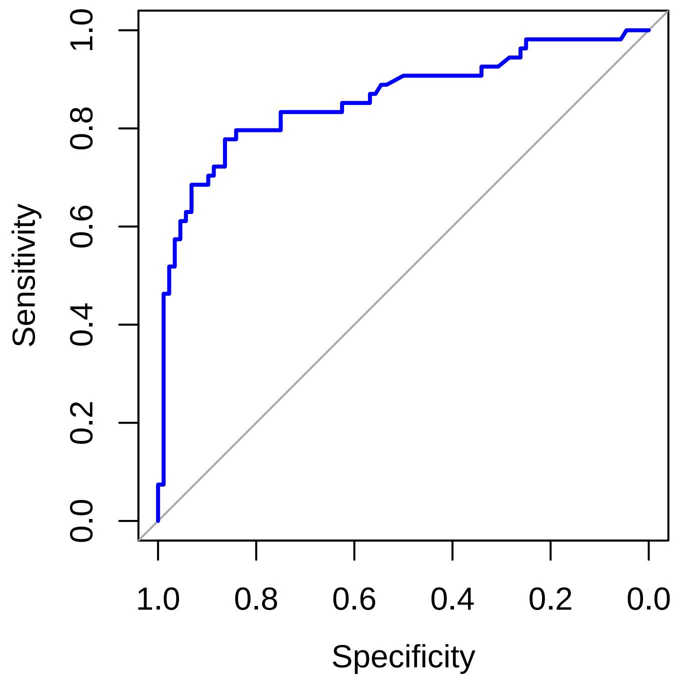
The estimated parameters are pretty close to the parameters studied in the Bayesian approach.

In order to compare the models, I use the same techniques used above on the eval (ROC curve + evaluation of the indices from the confusion matrix)

Evaluation time for the frequentist Model

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

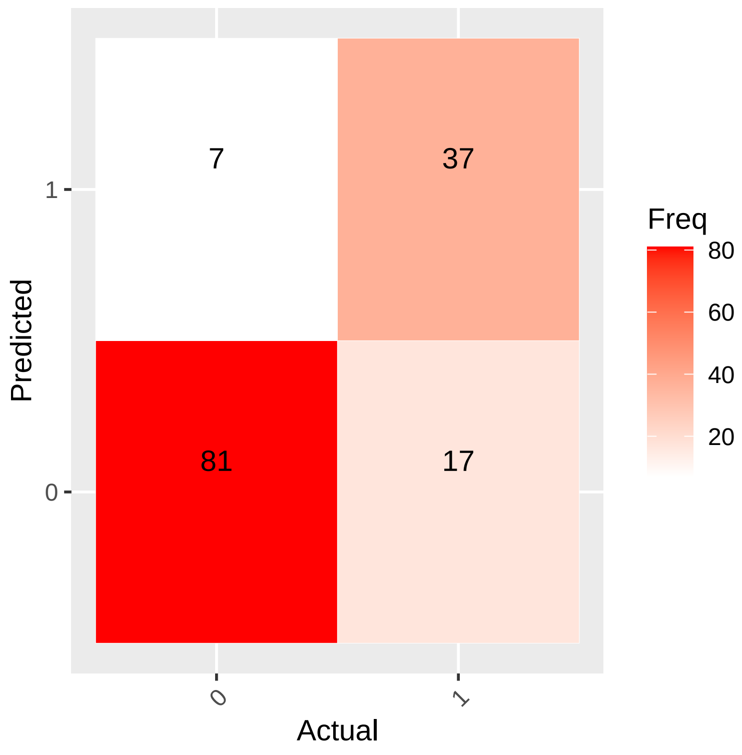


```
##
## Call:
## roc.default(response = y, predictor = preds)
##
## Data: preds in 88 controls (y 0) < 54 cases (y 1).
## Area under the curve: 0.8595
```

Now i find the best threshold that maximizes the accuracy.

```
## [1] "The maximum accuracy is:"

## [1] 0.83

## [1] "The corresponding threshold is:"

## [1] 0.63

## Precision
##      0.83

## Recall
##   0.92
```

Confusion Matrix



Let's now prooced with the same analysis, but using less parameters. In particular, i want to use only the sex and the ticket class because i think they are the most meaningful variables.

Model1:Logistic Regression(Logit)

Gibbs sampling of regression models typically use normal prior distributions over the weights, so the assumptions are the following:

$$Y_i \sim Ber(logit(p_i))$$
$$logit(p_i) = log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{1_i} + \beta_2 x_{2_i}$$

the prior parameters β will have a prior distribution with $\mu = 0$ and $\sigma = 0.0001$

$$\beta_i \sim N(0, 0.0001)$$

for $i = 1, 2$.

Implementation of the model with RJags I repet the same steps as before but reducing the number of parameters to 4.

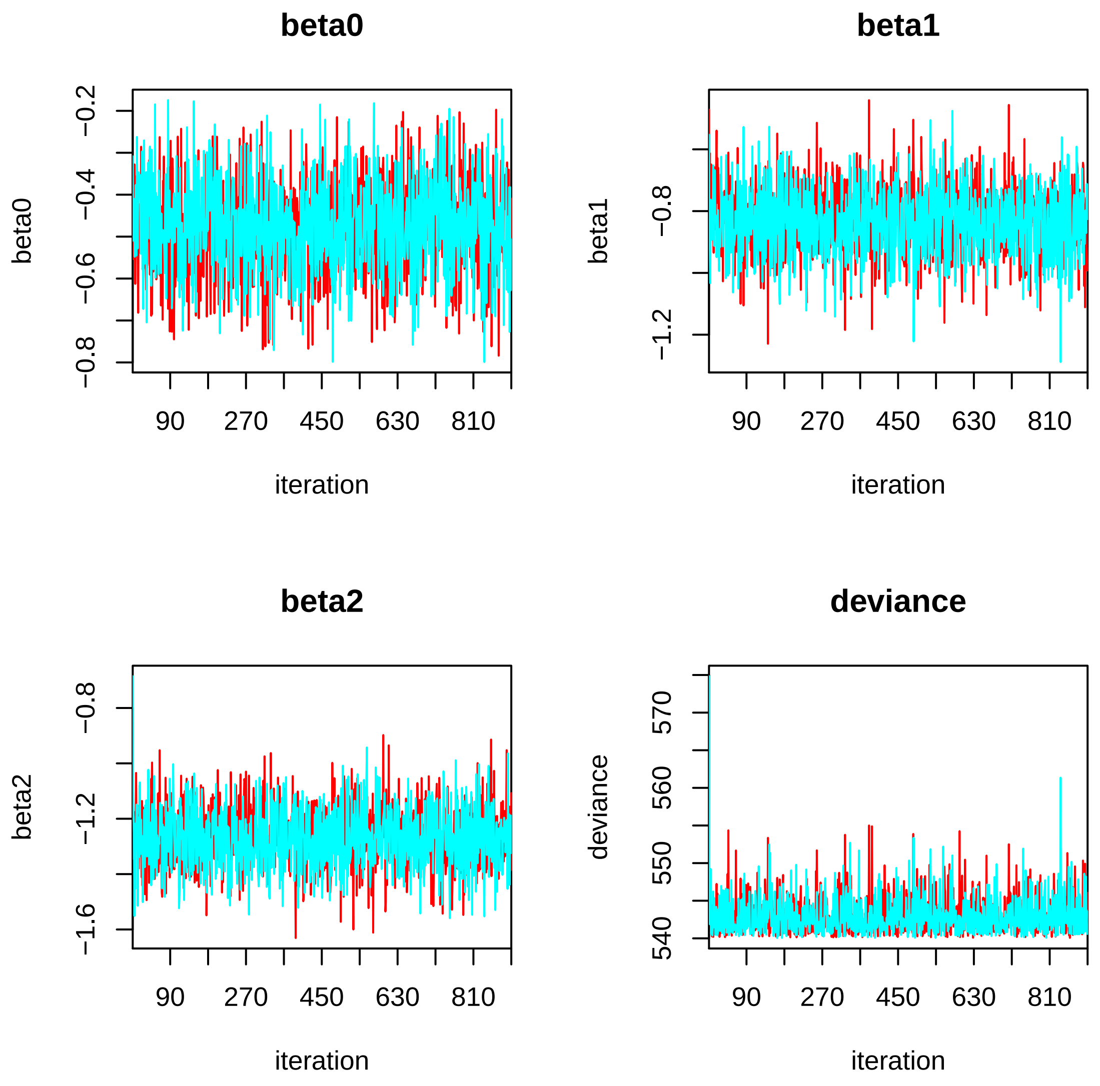
```
set.seed(123)
#Write the model in BUGS language
N = length(train$Survived)
model_func <- function(){
  #model
  for (i in 1:N){
    y[i] ~ dbern(p[i])
    #The logit function can be used to convert any real number to a probability.
    logit(p[i]) =  beta0 + beta1*x1[i] + beta2*x2[i]
  }

  #prior beta parameters ( a non informative prior)
  beta0 ~ dnorm(0, 0.0001)
  beta1 ~ dnorm(0, 0.0001)
  beta2 ~ dnorm(0, 0.0001)

}
```

These are the results we get:

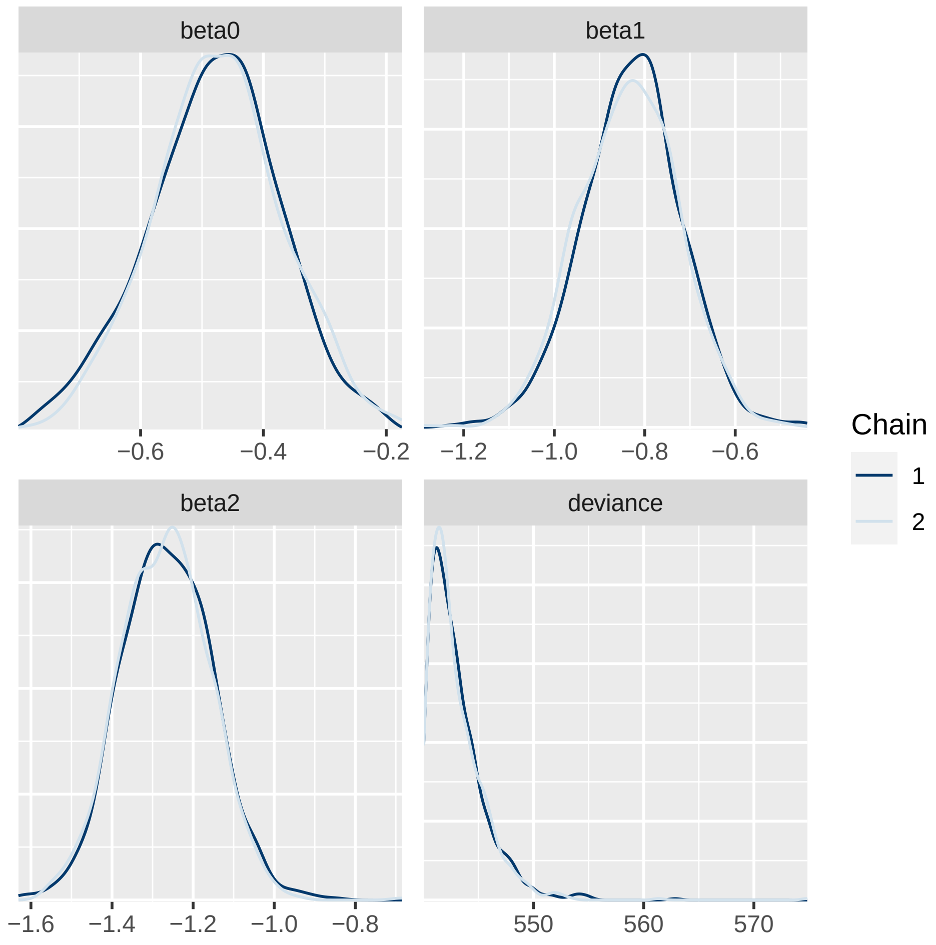
```
kable(df)
```

	mean.vect	sd.vect	Quantile_0.25	Quantile_0.50	Quantile_0.75	R.hat	N_eff
Beta_0	-0.478	0.108	-0.548	-0.476	-0.408	1.002	990
Beta_1-Pclass	-0.832	0.110	-0.907	-0.830	-0.759	1.002	1000
Beta_2-Sex	-1.267	0.111	-1.343	-1.267	-1.191	1.001	1800
Deviance	543.003	2.571	541.192	542.333	544.170	1.001	1800

```
pD = 3.3
DIC = 546.3
```


beta0
beta0
−0.8
−0.6
−0.4
−0.2
90
270
450
630
810
iteration
beta1
beta1
−1.2
−0.8
90
270
450
630
810
iteration
beta2
beta2
−1.6
−1.2
−0.8
90
270
450
630
810
iteration
deviance
deviance
540
550
560
570
90
270
450
630
810
iteration

Density



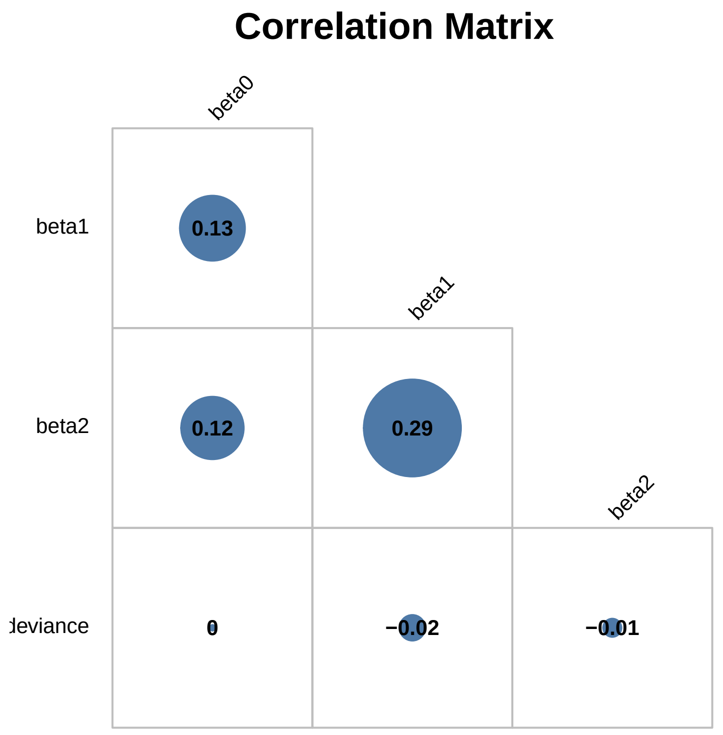
to check if the mean is inside the HPD intervals:

HPD intervals

	lower	upper
beta0	-0.6968517	-0.2614287
beta1	-1.0510551	-0.6250918
beta2	-1.4727488	-1.0440750
deviance	540.0587324	547.8436503

Let's plot the correlation matrix:

Correlation between parameters



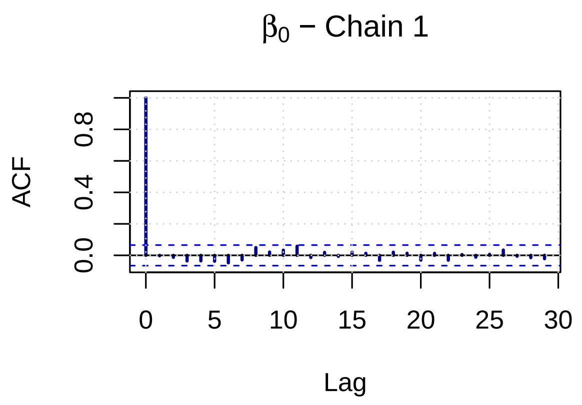
Approximation Error

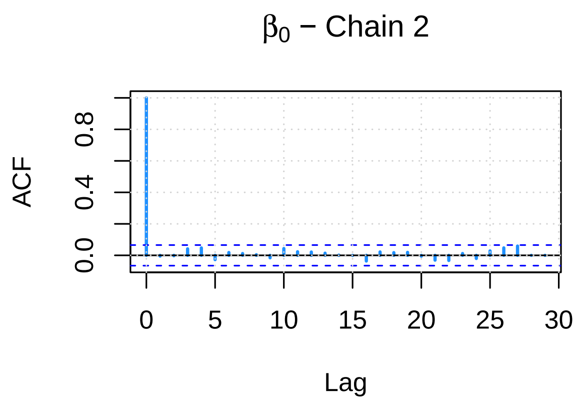
To evaluate the approximation error I use the **MCSE** estimate: the MCSE (Monte Carlo Standard Error) is an estimate of the inaccuracy of Monte Carlo samples, usually regarding the expectation of posterior samples from Monte Carlo Markov Chain algorithms.

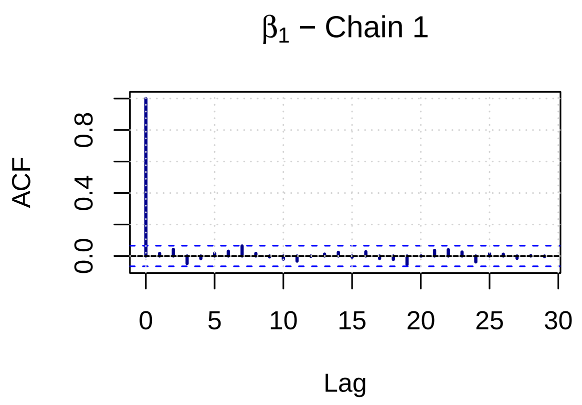
	Approximation_Error
beta_0	0.0036056
beta_1	0.0039239
beta_2	0.0038758

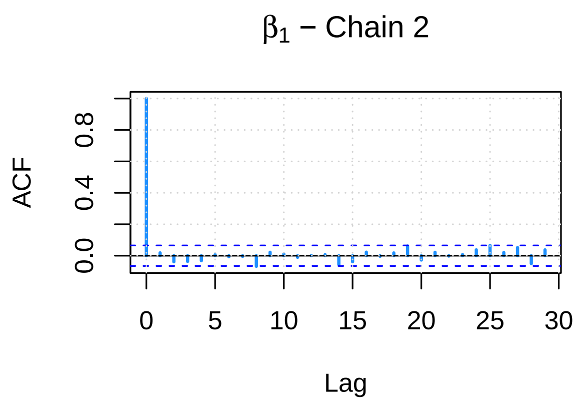
Auto-correlation

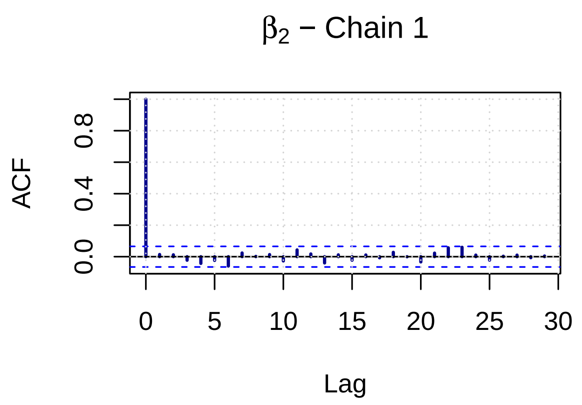
In general, lower autocorrelation values are preferred, as they indicate a higher degree of independence between consecutive samples in the chain. This indicates better mixing and convergence.For the lag1 autocorrelation, values close to 0 or within the range of -0.1 to 0.1 are generally considered acceptable. Larger absolute values of autocorrelation, such as above 0.2, may indicate slow mixing and convergence

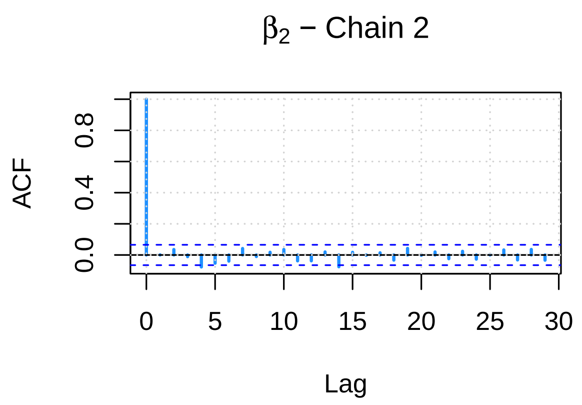












```
autocorr.diag(as.mcmc(model1))
```

```
##                beta0         beta1        beta2     deviance
## Lag 0    1.000000000  1.000000e+00  1.000000000  1.000000000
## Lag 10  -0.003715438  1.711667e-02  0.007405392  0.006996056
## Lag 50  -0.032264690  1.233926e-02 -0.038213157 -0.021457740
## Lag 100  0.038215773 -4.642466e-03  0.004010971  0.018845700
## Lag 500  0.007510411  9.398448e-05  0.004717901  0.014180359
```

so we get quite good samples that are expressing the assumption of being independent!

Geweke Diagnostic
This is based on a test for equality of the means of the first and last part of a Markov Chain (by default the first 10% and the last 50%).
The null hypothesis in the test is that two parts of the chain come from the same distribution, and to study it it's used a mean difference test.
If the two averages are equal it is likely that the chain will converge.

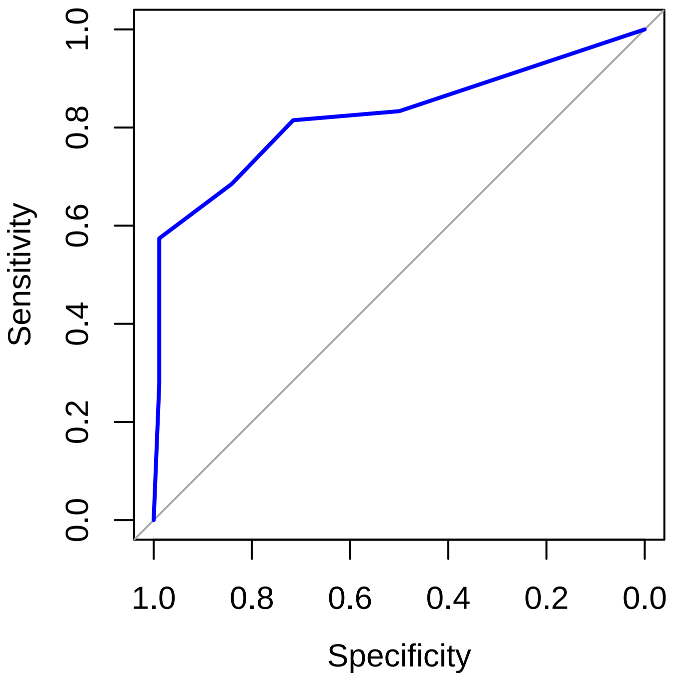
The test statistic is a standard Z-score: the difference between the two sample means divided by its estimated standard error adjusted for autocorrelation.
If the Z-scores are close to zero and within a reasonable range (-2 to 2), it generally indicates convergence. But even if the Geweke test result is slightly outside the typical range, it's important to consider it alongside other convergence diagnostics and visualizations to form a comprehensive assessment of the MCMC results.

```
## 
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5 
## 
##    beta0    beta1    beta2 deviance 
## -1.28125  0.56991  0.06225  0.45143
```

	Z.score_Chain1	Z.score_Chain2
Beta_0	-0.6649	0.8563
Beta_1	-0.1951	1.7230
Beta_2	0.2668	0.8705
Deviance	-0.9564	0.5528

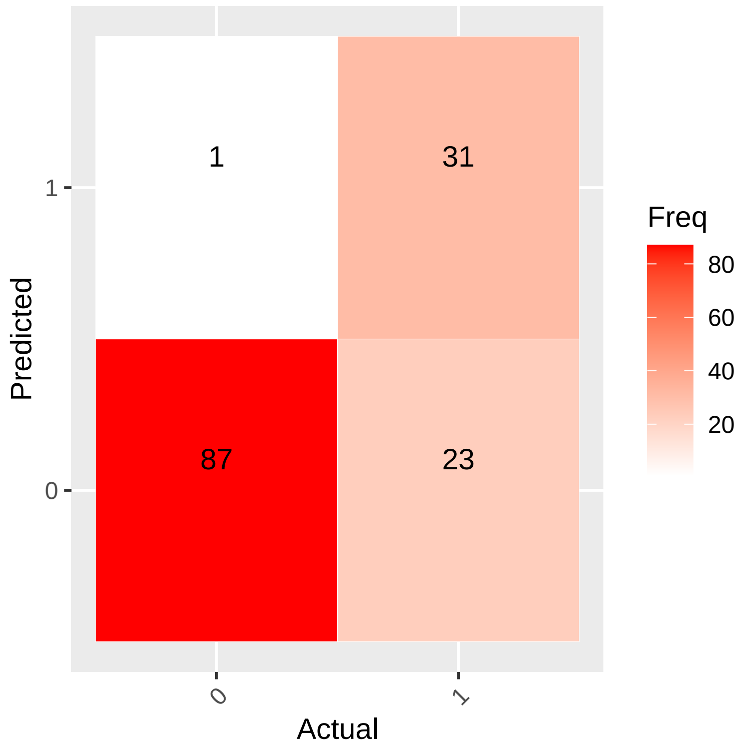
Evaluatio time for the bayasian linear regression Model1 I proceed with the same analysis as before, i plot the ROC and visualize the AUROC value

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```



```
##
## Call:
## roc.default(response = y, predictor = probs)
##
## Data: probs in 88 controls (y 0) < 54 cases (y 1).
## Area under the curve: 0.8246

## [1] "The maximum accuracy is:"

## [1] 0.83

## [1] "The corresponding threshold is:"

## [1] 0.61

## Precision
##      0.79

## Recall
##   0.99
```

Confusion Matrix



```r
freq2 = glm(Survived ~ Pclass + Sex , family = binomial(link = "logit"),data=train)
```

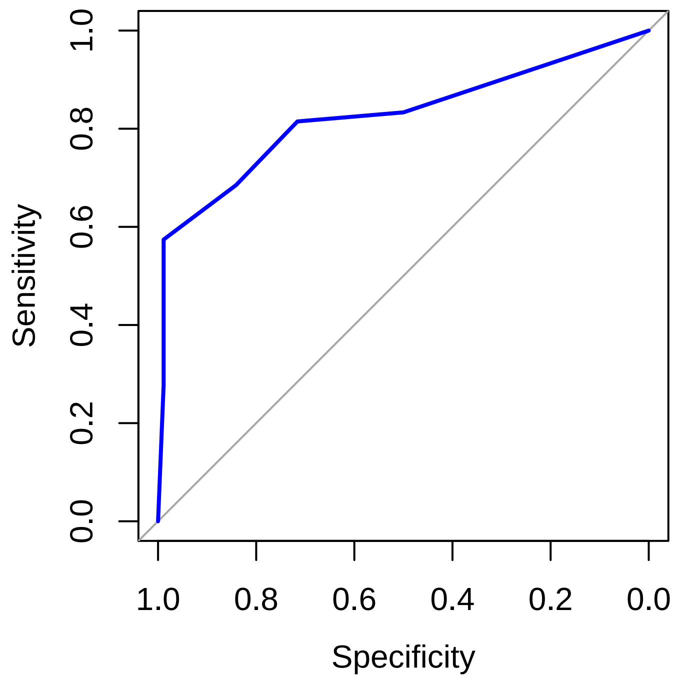
Comparison with the Frequentist model

```
##    Variable Coefficient
## 1 Intercept     -0.4744
## 2    Pclass     -0.8269
## 3       Sex     -1.2536
```

Residual Deviance: 540 AIC: 546
even here the parameters are quite similar!

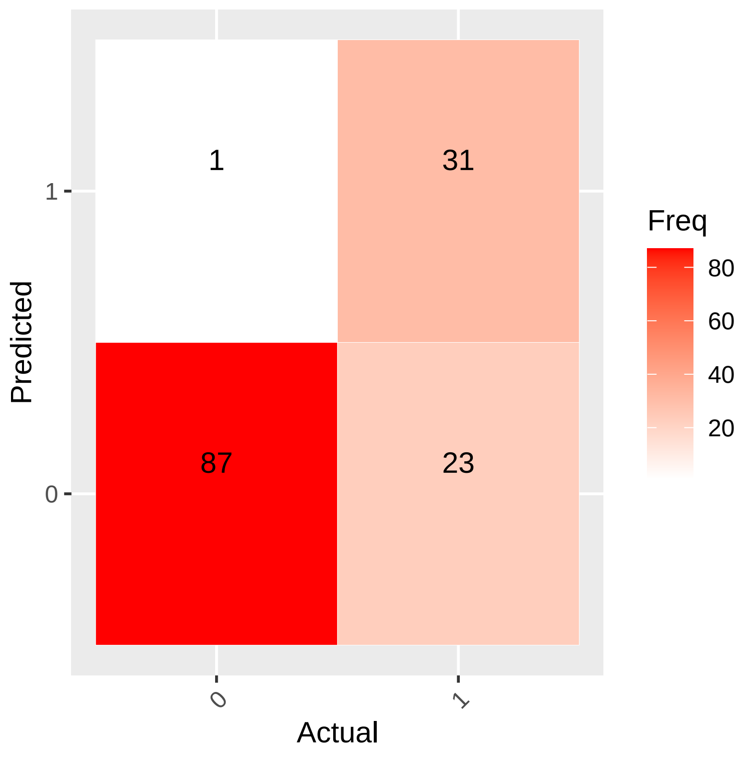
Evaluation time for the frequentist Model1

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```



```
##
## Call:
## roc.default(response = y, predictor = preds2)
##
## Data: preds2 in 88 controls (y 0) < 54 cases (y 1).
## Area under the curve: 0.8246
## [1] "The maximum accuracy is:"

## [1] 0.83

## [1] "The corresponding threshold is:"

## [1] 0.61

## Precision
##      0.79

## Recall
##   0.99
```

Confusion Matrix



Now let's look at the models with the optimal thresholds in order to choose the best one for this problem

```
##                     BestThreshold Accuracy Precision Recall AUROC
## Model Bayesian               0.55     0.84      0.86   0.89 0.863
## Model Frequentist            0.63     0.83      0.83   0.92 0.860
## Model1 Bayesian              0.61     0.83      0.79   0.99 0.825
## Model1 Frequentist           0.61     0.83      0.79   0.99 0.825
```

```
##        Bayesian_DIC Frequentist_AIC
## Model         524.7           524.2
## Model1        546.3           546.0
```

Conclusions After my analysis, I can say the model with the highest accuracy is the Model with 6 parameters tuned with MCMC. However this is quite similar in terms of accuracy with the other models both Frequentist and Bayesian with 3 parameters, so the really effective features were Sex and Pclass.