Introduction to `rstanarm`

Bayesian Inference - Lab Sessions (2/3)

Marika D'Agostini
`marika.dagostini2@unibo.it`

University of Bologna

November-December 2023

Software implementation: `Stan` and `rstanarm`

The `Stan` software has represented a great improvement in the Bayesian computation framework.

It is based on the **Hamiltonian Monte Carlo (HMC)** algorithm.

N.B.: `Stan` is a very powerful tool since it is a *generic statistical model specification language* (e.g.: GLM, GLMM) and it implements also optimization methods

The `Stan` modeling language and statistical algorithms are exposed through interfaces into many popular computing environments:

- `RStanArm` (R)
- `PyStan` (Python)
- `StataStan` (Stata)

Prerequisites

Basic notions of statistical modelling that you have learnt in the *Statistical Models and Applications* course:

- Generalized linear models,
- Models including random effects.

Remark: which is the difference between a fixed effect and random effect in the Bayesian inferential framework?

Key steps

A basic inferential procedure in the Bayesian framework using `rstanarm` requires the following steps:

1) **Understand the data**
2) **Model formulation:** the model structure and hierarchy must be carefully studied and written
3) **Prior specification**
4) Specification of *initial values* for the simulation algorithm
5) Write the *model equation* in `rstanarm` and start the computation.
6) Check the *convergence* of the algorithm
7) Check the model assumptions (**posterior predictive check**) and/or choice of the better model (WAIC)
8) Analyse the posterior results (**summarize the posterior distribution**)

1. Understand the data - Gaussian Linear Model

See the script: Introduction_rstanarm.R

```
library(rstanarm)
library(rstan)

## Data generation
set.seed(123)
n <- 100

x1 <- rnorm(n = n,mean = 1, sd = 0.5)
x2 <- rnorm(n = n,mean = 0.5, sd = 0.25)
X <- cbind(x1, x2)

sigma <- 1
beta <- c(1, 2)

y <- rnorm(n = n, mean = X %*% beta, sd = sigma)
```

2. Model formulation

Once the problem is stated and the data explored, the first task is to carefully specify the model hierarchy:

1. **Likelihood:** you need to express the distributional assumption and the model equation you are considering. E.g. for a *Gaussian linear model*:

$$\begin{aligned} y_i|\mu_i, \sigma^2 &\sim \mathcal{N}\left(\mu_i, \sigma^2\right), \; i = 1, ..., n \\ \mu_i|\boldsymbol{\beta} &= \beta_0 + x_{i1}\beta_1 + x_{i2}\beta_2 \; i = 1, ..., n. \end{aligned}$$

2. **Priors:** you need to state the prior distributions specified for the parameters. In this case:

$$\begin{aligned} &\beta_p \sim \mathcal{N}(0, c), \; p = 1, 2, 3; \\ &\sigma \sim \text{half-Cauchy}. \end{aligned}$$

3. **Hyperpriors:** priors for the parameters of the model parameters priors (i.e. hyperparameters). In this case they are not present (c is a fixed constant).

N.B. Be careful about the conditioned quantities

2. `rstanarm` functions

Once the model scheme is clear, we need to implement it with the `rstanarm` package syntax. We will use two functions:

- `stan_glm`: allows to make Bayesian inference for GLM,
- `stan_glmer`: allows to make Bayesian inference for GLM with *random effects*.

The first block of arguments that allows to implement the desired models include:

- `formula`: useful to express the model equation (also group random effects).
- `data`: the `data.frame` that includes the variables declared in the `formula` statement.
- `family`: the distributional assumption. We will consider `"gaussian"`, `"binomial"` and `"poisson"`.

2. `formula` statement

The same syntax of the lm and glm formulas is required when only *fixed effects* are included in the model. In our example:

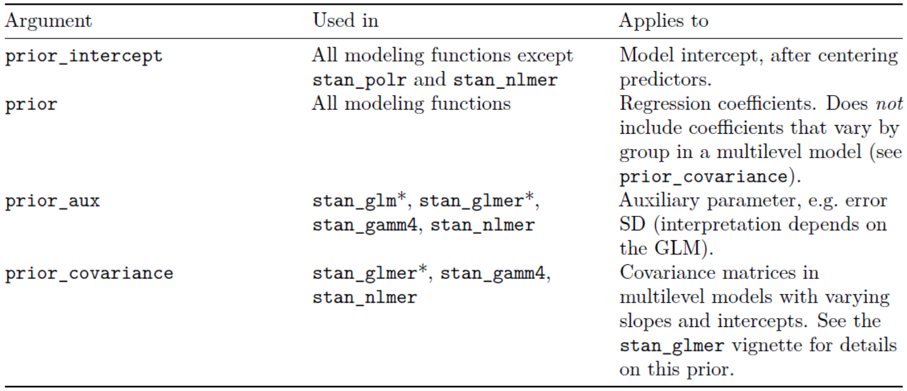
```
mod_formula <- y ~ x1 + x2
```

When a *random effect* determined by a generic grouping variable `group` is required, the same syntax of `lme4` package is adopted.

For example, a **random intercept** can be included in the formula as `(1|group)` and a **random coefficient** for the covariate `x` is declared as `(x|group)`.
$\rightarrow$ This will be faced later

3. Prior specification

The following arguments allow you to specify the prior distributions (from the vignette *Prior*):



Argument	Used in	Applies to
`prior_intercept`	All modeling functions except `stan_polr` and `stan_nlmer`	Model intercept, after centering predictors.
`prior`	All modeling functions	Regression coefficients. Does *not* include coefficients that vary by group in a multilevel model (see `prior_covariance`).
`prior_aux`	`stan_glm`*, `stan_glmer`*, `stan_gamm4`, `stan_nlmer`	Auxiliary parameter, e.g. error SD (interpretation depends on the GLM).
`prior_covariance`	`stan_glmer`*, `stan_gamm4`, `stan_nlmer`	Covariance matrices in multilevel models with varying slopes and intercepts. See the `stan_glmer` vignette for details on this prior.

4. Initialization

Usually, the **multichain** approach is used: two or more parallel Monte Carlo Markov Chains are simulated in order to check the convergence to the target distribution by comparison.

Each chain for every parameter needs to be initialized: choosing different starting values the robustness of the algorithm is tested.

By default, `Stan` and `rstanarm` randomly generates appropriate starting values. However in case of particularly complex models the algorithm might requires a more precise initialization.

Some arguments to specify various options for the estimation:

- `chains`: number of parallel chains.
- `warmup`: number of warmup iteration not included in the inference
- `iter`: number of total iterations
- `init`: list of lists with the fixed starting values (optional)

5. Start the computation

The Stan engine can be invoked in R through the functions stan_glm or stan_glmr generating a StanFit object

Example:

```
data_example <- data.frame(y,x1,x2)
mod_equation <- y ~ x1+x2

c <- 10

stan_fit_lr <-stan_glm(formula = mod_equation,
                       data = data_example,
                       family = "gaussian",
                       prior = normal(0,c),
                       prior_intercept = normal(0,c),
                       prior_aux = cauchy(0,1))
```

6. Check the convergence of the algorithm: traceplot

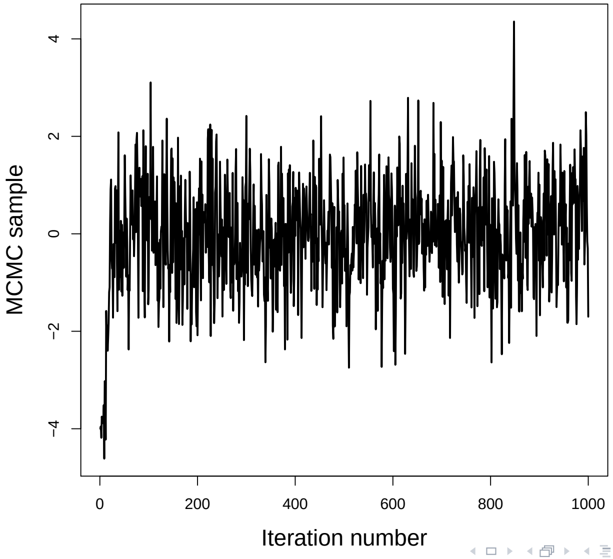
- The first step is the visual analysis of the **traceplot**: the iterations are plotted as a time series. **In case of convergence, the series must be regular and stationary and the different chain are overlapped.** (Ideally it should look like a caterpillar or bar code)

- The warm-up iterations should be included in order to check if the selected number is sufficient.

- After convergence the samples should not converge to a single point!

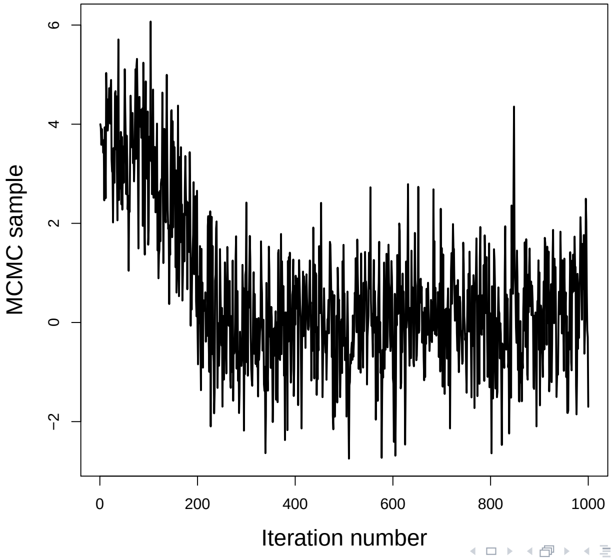
6. Traceplot examples (I)

→ Convergence in a few iterations



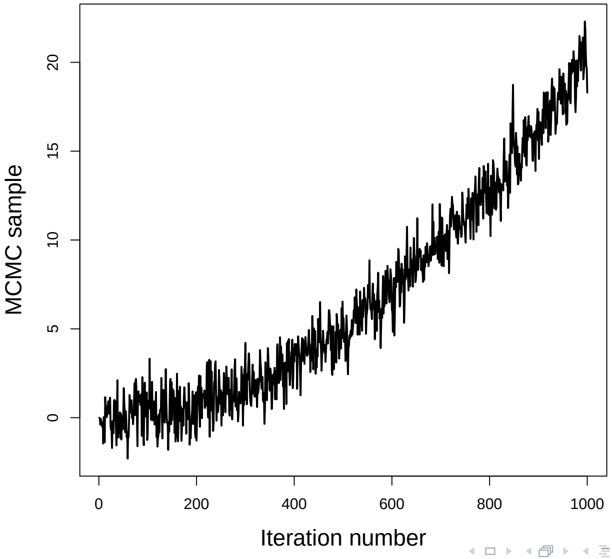
6. Traceplot examples (II)

→ Convergence in a few hundred iterations



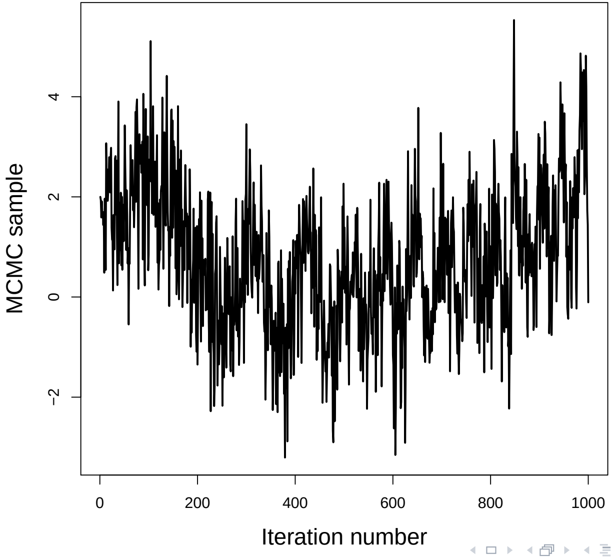
6. Traceplot examples (III)

→ This one never converged



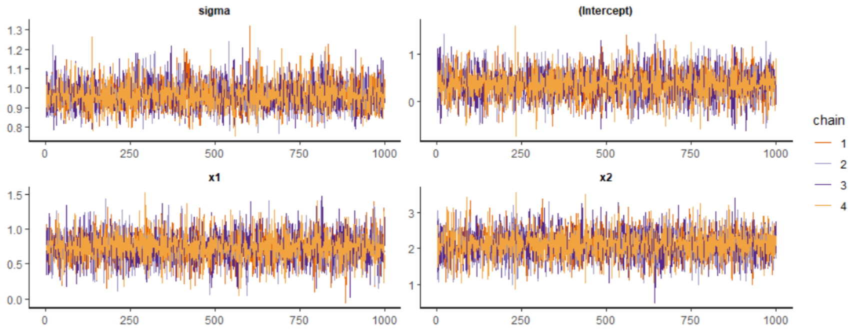
6. Traceplot examples (IV)

→ Convergence is questionable



6. Check the convergence of the algorithm: code

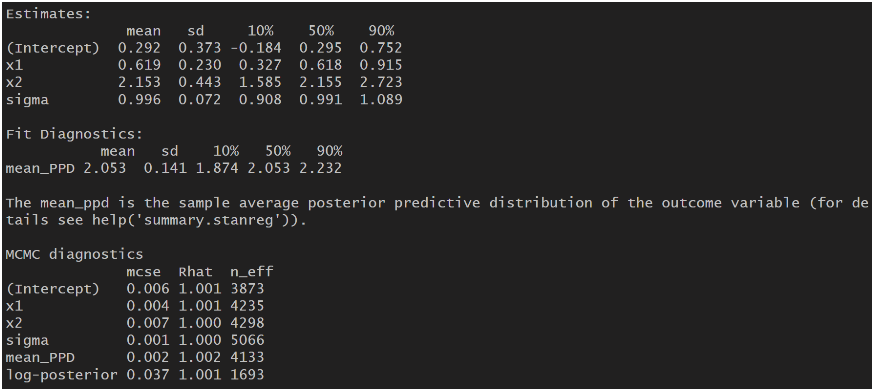
```
stan_trace(stan_fit_lr,
           pars=c("sigma","(Intercept)", "x1", "x2"))
```



6. Algorithm convergence: results overview

Other important indications can be deduced from the output of the `summary` command:

```
summary(stan_fit_lr,
          pars=c("sigma","(Intercept)", "x1", "x2"))
```



```
Estimates:
              mean   sd    10%   50%   90% 
(Intercept)  0.292  0.373 -0.184  0.295  0.752
x1           0.619  0.230  0.327  0.618  0.915
x2           2.153  0.443  1.585  2.155  2.723
sigma        0.996  0.072  0.908  0.991  1.089

Fit Diagnostics:
           mean   sd    10%   50%   90% 
mean_PPD 2.053  0.141 1.874 2.053 2.232

The mean_ppd is the sample average posterior predictive distribution of the outcome variable (for de
tails see help('summary.stanreg')).

MCMC diagnostics
              mcse  Rhat  n_eff
(Intercept)   0.006 1.001 3873
x1            0.004 1.001 4235
x2            0.007 1.000 4298
sigma         0.001 1.000 5066
mean_PPD      0.002 1.002 4133
log-posterior 0.037 1.001 1693
```

6. Algorithm convergence: $\hat{R}$ statistic (I)

Adopting a multichain approach, the $\hat{R}$ statistic (known also as *Gelman and Rubin statistic* or *Potential Scale Reduction Factor*) allows to check if the chains have reached the target distribution.

Considering M different chains with B realizations $\theta_m^{(n)}$, then the between sample variance is:

$$Be = \frac{B}{M-1}\sum_{m=1}^{M}\left[\bar{\theta}_m^{(\bullet)} - \bar{\theta}_\bullet^{(\bullet)}\right]^2;$$

whereas the within sample variance is:

$$Wi = \frac{1}{M(B-1)}\sum_{m=1}^{M}\sum_{b=1}^{B}\left[\theta_m^{(b)} - \bar{\theta}_m^{(\bullet)}\right]^2.$$

It is now possible to define an estimator of the average variance of samples:

$$\hat{V}^{+}[\theta] = \frac{B-1}{B}Wi + \frac{1}{B}Be.$$

6. Algorithm convergence: $\hat{R}$ statistic (II)

Note that $\hat{V}^+[\theta]$ is an unbiased estimate of the variance in case of convergence, otherwise it overestimates it.

Then, the $\hat{R}$ statistic is defined as:

$$\hat{R} = \sqrt{\frac{\hat{V}^+[\theta]}{Wi}}$$

The values assumed by the statistic must be interpreted as follows:

- $\underline{\hat{R} \sim 1\text{: all the chains reached the equilibrium distribution}}$;
- $\hat{R} > 1$: at least one chain did not converged (*RStan*: $\hat{R} > 1.05$).

Important: the $\hat{R}$ statistic is based on the underlying assumption of normality, so it is not appropriated in case of random variables with distribution largely different from the Gaussian.

N.B.: if many parameters are present in the model (e.g. posterior predictive), it is possible to plot the $\hat{R}$ statistic with

```
plotfun = "rhat"
```

6. Algorithm convergence: $\hat{N}_{eff}$

As known, one of the problems of MCMC is the autocorrelation among the realizations drawn.

One of the effects of the presence of autocorrelation in the sample is that uncertainty increases.

A way to measure theautocorrelation is the **effective sample size:**

$$N_{eff} = \frac{N}{\sum_{t=-\infty}^{+\infty} \rho_t},$$

where ρ_t is the autocorrelation at lag t for a chain.

It can be estimated through $\hat{N}_{eff}$ from the samples (multichain approach required) using the variogram V_t, at lag t.

A value of $\hat{N}_{eff}$ particularly lower than N highlights potential issues of autocorrelation. In this case, more samples might be required, in order to compute reliable estimates of the posterior distribution key quantities.

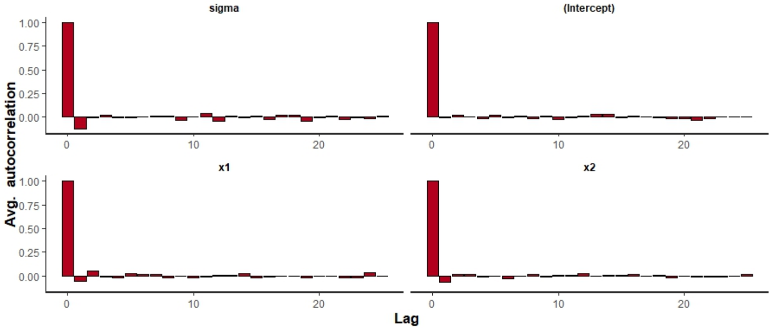
$\rightarrow$ Paragraph 30.3 of the Stan reference manual for more details about $\hat{R}$ and $\hat{N}_{eff}$

6. Algorithm convergence: autocorrelation

Another way to check for the presence of autocorrelation is to plot the autocorrelation function.

```
library(ggplot2)
stan_ac(stan_fit_lr,
        pars=c("sigma","(Intercept)", "x1", "x2"))
```

In case of slow decay of the ACF it is possible to *thin* the simulations.
If the thinning interval of T is chosen, then a simulation every T is considered and the other discarded.



7. Check model assumptions

Assured that the algorithm reached the correct sampling distribution, it is possible to perform the usual checks of the model assumptions (as in the frequentist world).

Some important remarks:

- The usual *residuals checks* are **still valid**.
- The usual *statistical tests* are **not valid**.
- In this context our "*best friend*" is the **Posterior Predictive Distribution (PPD)**. It is the basement of the **posterior predictive model checking** (**Bayesian p-value**) since it allow to evaluate the goodness of fit of a model comparing the predicted quantities to the observed ones.
- *Deviance based methods* similar to AIC and BIC are studied (e.g. DIC and **WAIC**). Cross validation methods are also available (`R` package `loo`).
- *Variable selection procedures*: different prior specifications could be used for this purposes.

7. Posterior predictive distribution (PPD)

The PPD is defined as:

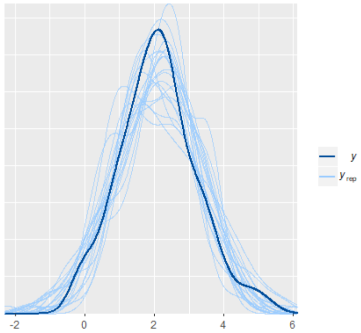
$$\begin{aligned} p\left(y_{rep}|y\right) &= \int_{\Theta} p\left(\mathbf{y}_{rep}|\mathbf{y},\theta\right) p\left(\theta|\mathbf{y}\right) \mathrm{d}\theta \\ &= \int_{\Theta} p\left(\mathbf{y}_{rep}|\theta\right) p\left(\theta|\mathbf{y}\right) \mathrm{d}\theta. \end{aligned}$$

Even if it appears to be a complex integral, it could be easily deduced from the simulations of θ. In this way it is possible to **generate a new dataset** that replicates the observed one **at each Monte Carlo iteration**.

Therefore, it represents a natural starting point to evaluate the discrepancy between the real data and the replicated ones: in case of systematic deviations it is possible to suppose that the model is not suitable to fit the dataset.

After a model is fitted through an `rstanarm` function, it is possible to generate new samples from its posterior predictive distribution using `posterior_predict`.

7. PPD: Empirical Density



A first visual check consists in the comparison between the **empirical density functions** of real data and the empirical distributions of the replicated datasets.

```
library(bayesplot)

post_pred <- posterior_predict(
        stan_fit_lr)

ppc_dens_overlay(y = y,
  yrep = post_pred[40:100,])
```

7. Posterior Predictive Checks (I)

If the goal is to check particular aspects of the model it is possible to consider an appropriate **discrepancy measure**.

Discrepancy measures are functions of the parameters vector and data and they could be specified as $D(y, \theta)$ (remember that a *test statistics* must be functions of data only).

It is possible to define as **posterior predictive p-value** (**Bayesian p-value**) the following generalization of the classical p-value:

$$p_B = \mathbb{P}\left[D(\mathbf{y}_{rep}, \theta) \geq D(\mathbf{y}, \theta) | \mathbf{y}\right],$$

where the conditioning is only with respect to the data.

Equivalently, we can define the following integral:

$$p_B = \int \int \mathbb{1}_{\{D(\mathbf{y}_{rep}, \theta) \geq D(\mathbf{y}, \theta)\}} p\left(\mathbf{y}_{rep} | \theta\right) p\left(\theta | \mathbf{y}\right) \mathrm{d}\mathbf{y}_{rep} \mathrm{d}\theta.$$

7. Posterior predictive checks (II)

The integral that cannot be solved analytically $\rightarrow$ solve it by simulation

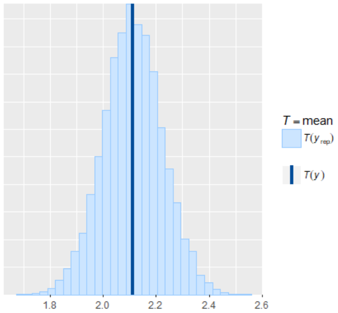
In the k-th **MCMC** step:

1) A value $\theta^*_{(k)}$ is drawn from the posterior $p(\theta|\mathbf{y})$
2) A value (i.e. a vector) $\mathbf{y}_{rep}$ is drawn from the PPD, given the value $\theta^*_{(k)}$.
3) The values $D(\mathbf{y}_{rep,(k)}, \theta^*_{(k)})$ and $D(\mathbf{y}, \theta^*_{(k)})$, are computed and compared.

Then, the Bayesian p-value is estimated through the proportion of simulations in which: $D(\mathbf{y}_{rep,(k)}, \theta^*_{(k)}) \geq D(\mathbf{y}, \theta^*(k))$.

If the result is near 1 or 0 the model should be considered not appropriate to fit the data.

A graphical output is usually enough and it could be obtained with the function `ppc_stat(y = y, yrep = post_pred, stat = "mean")`.



7. Information criterion: WAIC (I)

The **W**idely **A**pplicable **I**nformation **C**riteria is an estimate of the expected log density of a new dataset and it is composed by two quantities:

- **Log Pointwise Predictive Density**: a measure aimed at summarizing the predictive accuracy of the fitted model to the sample:

$$\widehat{lppd} = \sum_{i=1}^{n} \log\left(\frac{1}{K}\sum_{k=1}^{K} p(y_i|\theta^*_{(k)})\right).$$

- **A correction for the effective number of parameters** based on the posterior variance of the predictive density of each data point:

$$p_{WAIC} = \sum_{i=1}^{n}\left(\hat{\mathbb{V}}\left[\log p(y_i|\theta^*_{(\cdot)})\right]\right).$$

7. Information criterion: WAIC (II)

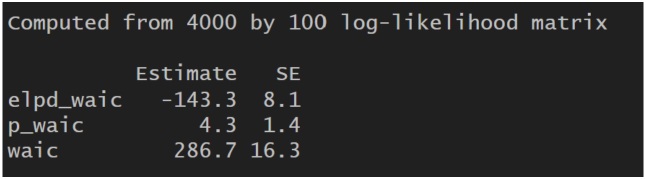
The final WAIC value is the difference of these two quantities:

$$WAIC = -2\left(\widehat{lppd} - p_{WAIC}\right)$$

In practice, it is useful to compare two or more model, and the best one is suggested by the lower WAIC value.

It can be easily computed by means of the function `waic` of the `loo` package

```
waic(stan_fit_lr)
```

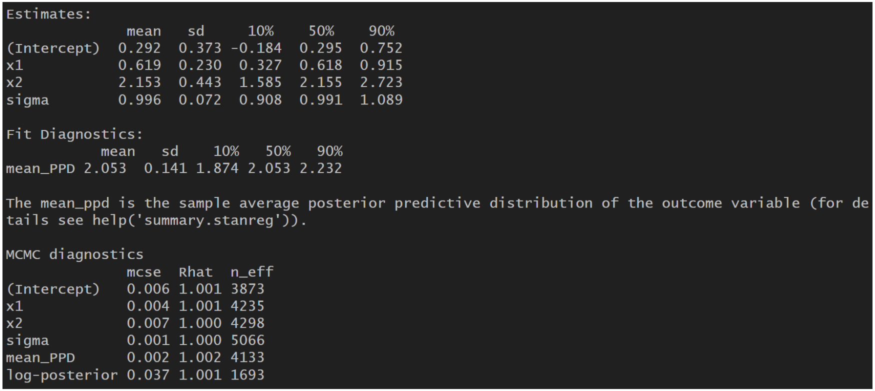


```
Computed from 4000 by 100 log-likelihood matrix

          Estimate   SE
elpd_waic   -143.3  8.1
p_waic         4.3  1.4
waic         286.7 16.3
```

8. Summarize the posterior distribution (I)

After the usual checks of the model assumptions, it is possible to derive easily the Bayes estimators (both point and intervals) of the quantity of interest.

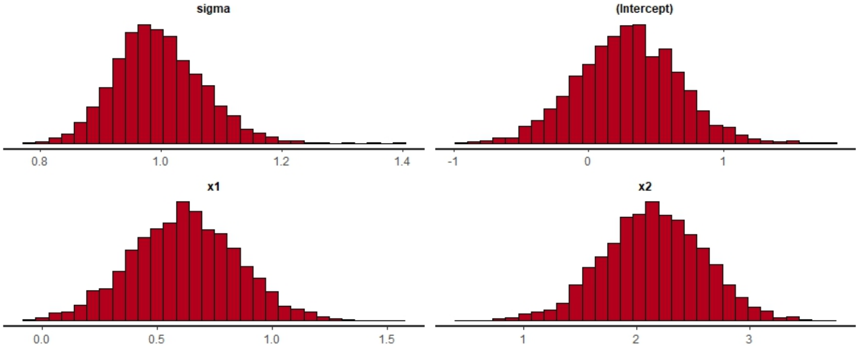
```
summary(stan_fit_lr,
        pars=c("sigma","(Intercept)", "x1", "x2"))
```



```
Estimates:
              mean   sd     10%    50%    90% 
(Intercept)  0.292  0.373 -0.184  0.295  0.752
x1           0.619  0.230  0.327  0.618  0.915
x2           2.153  0.443  1.585  2.155  2.723
sigma        0.996  0.072  0.908  0.991  1.089

Fit Diagnostics:
           mean   sd    10%   50%   90%
mean_PPD 2.053  0.141 1.874 2.053 2.232

The mean_ppd is the sample average posterior predictive distribution of the outcome variable (for de
tails see help('summary.stanreg')).

MCMC diagnostics
              mcse  Rhat  n_eff
(Intercept)   0.006 1.001 3873 
x1            0.004 1.001 4235 
x2            0.007 1.000 4298 
sigma         0.001 1.000 5066 
mean_PPD      0.002 1.002 4133 
log-posterior 0.037 1.001 1693 
```

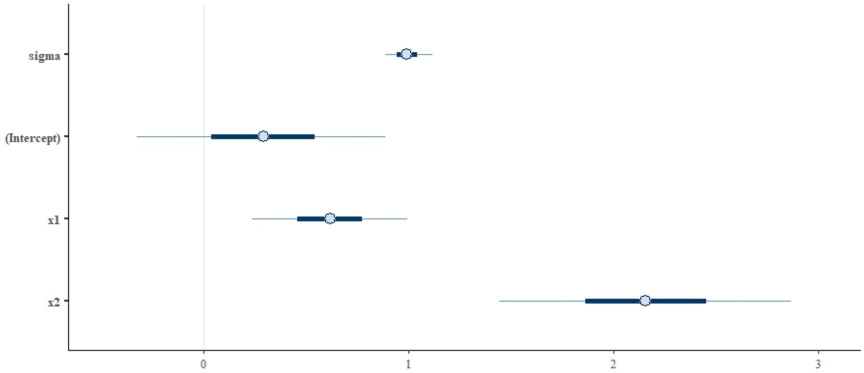
8. Summarize the posterior distribution (II)

```
stan_hist(stan_fit_lr,
          pars = c("sigma","(Intercept)", "x1", "x2"))
```



8. Summarize the posterior distribution (III)

```
plot(stan_fit_lr,
        pars = c("sigma","(Intercept)", "x1", "x2"))
```


sigma
(Intercept)
x1
x2