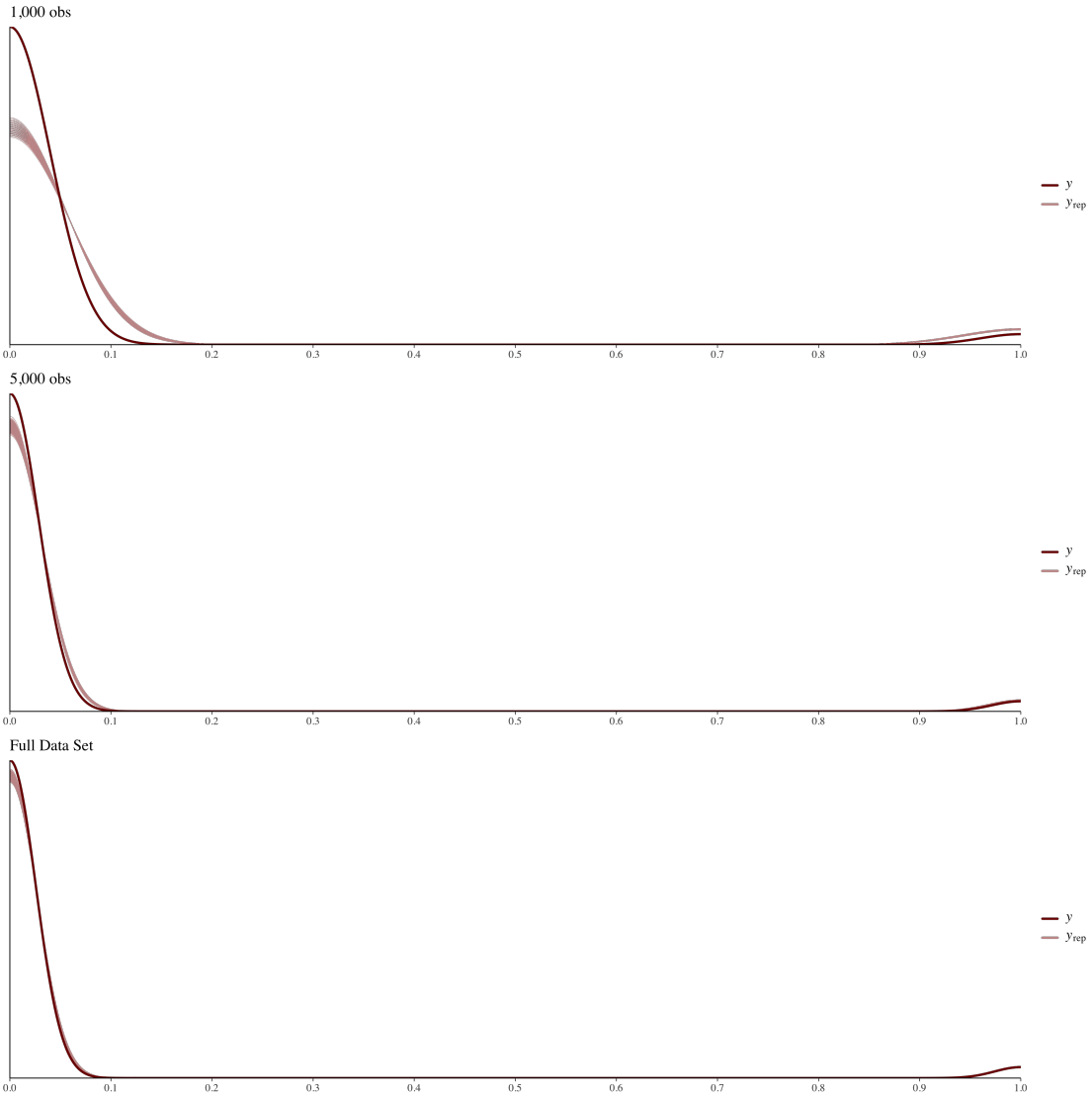
Final Assignment for Introduction to Software in Econometrics (EBS2072)

Tobias Schnabel (i6255807)

January 31, 2023 | Academic Year 2022/23

Tutorial Group 2
Dr. Nalan Baştürk



Contents

List of Figures

List of Tables

Intoduction

This report compares a Frequentist logistic regression model with several corresponding Bayesian models. The purpose of these varying implementations is to contrast the advantages and difficulties of specifying varying implementations of Bayesian models using two sets of priors and varying sample sizes. Specifically, I seek to assess whether incorrectly specified priors are overwhelmed by the likelihood at different sample sizes and the absolute and relative posterior predictive accuracy of the respective models.

1 Data

The data set used in the remainder of this report is the simulated *Default* data set, taken from *An introduction to statistical learning*. The stated aim of this data set is he aim here is to "predict which [credit card] customers will default on their credit card debt" (James, Witten, Hastie, and Robert Tibshirani 2021, p. 133). It spans $10,000$ simulated observations over 4 variables: $default$, the binary dependent variable, *student*, a binary indicator of whether a customer is a student, *balance*, a (non-negative) continuous variable which stores a customer's credit card balance, and *income*, a (non-negative) continuous variable which records a customer's income.

1.1 Summary Statistics

Table 1 below displays summary statistics. No variables have missing observations. About one third of customers are students, whereas only about 3.3% of customers default on their credit card payment, making this data set highly imbalanced, which presents certain challenges when splitting the data set.

Table 1: Summary Statistics

Statistic	N	Mean	St. Dev.	Min	Max
default	10,000	0.033	0.179	0	1
student	10,000	0.294	0.456	0	1
balance	10,000	835.000	484.000	0.000	2,654.000
income	10,000	33,517.000	13,337.000	772.000	73,554.000

Figure 1: Correlation of Student and Balance

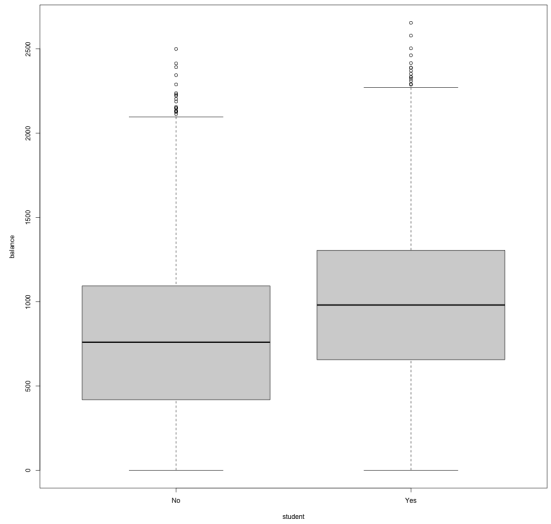
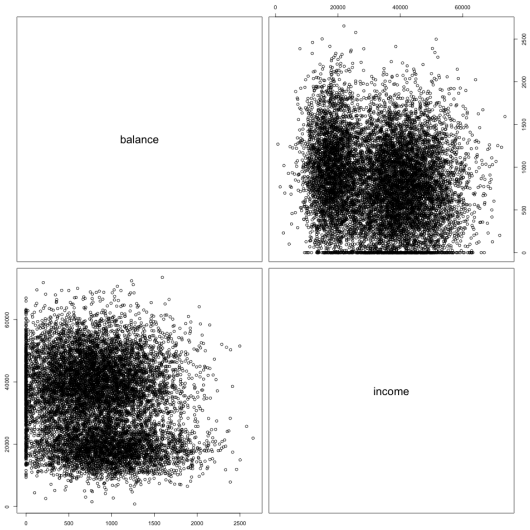


Figure 2: Correlation of Balance and Income



One interesting property of these data is that two variables, *student* and *balance*, are correlated, as can be seen in Figure 1: students tend to hold higher levels of debt, which we would a priori expect to be associated with a higher default probability. As shown in Figure 2, *balance* and *income* show no clear correlation.

1.2 Sample Splitting

To be able to compare the varying implementations of the Bayesian model introduced in section 2, I generate two subsamples from the data, which hold $1,000$ and $5,000$ observations, respectively. The random split preserves the proportion of $default$, the response, in all subsamples by using *caret: Classification and Regression Training*'s `create_partition()` function. Table 2

below shows the proportion of the response in these subsamples and the full data set.

Table 2: Data Splits

	n_obs	Proportion of Defaults
Subset 1	1000	0.033
Subset 2	5000	0.032
Original Data	10000	0.034

2 Model

This section describes the specification of the Frequentist model as well as two specifications of the same logistic regression as Bayesian models with two different priors.

2.1 Frequentist Baseline

The main interest of this model lies in predicting default on outstanding credit card balance given all three predictors, or, more formally,

$$\mathbf{P}(y = 1|\mathbf{x}) = \mathbf{P}(y = 1|student, balance, income) \tag{1}$$

(cf. Wooldridge 2020, p. 560) To achieve this, and given the binary nature of the response, I use a simple logistic regression model estimated by Maximum Likelihood of the form

$$default = \alpha + \beta_1 \times student + \beta_2 \times balance + \beta_3 \times income \tag{2}$$

using `glm(form, data = data, family = binomial(link = "logit"))`. Estimation Results are reported in Section 4.

2.2 Bayesian Model with Flat Priors

The first Bayesian model implements (2) by assuming uninformative, that is, *flat*, priors. The R implementation requires *rstanarm: Bayesian Applied Regression Modeling via Stan*'s wrapper function `stan_glm()`, as *rstan: R Interface to Stan*'s `sampling()` implementation did not progress past the first iteration using either of the two available (NUTS and HMC) Markov Chain Monte Carlo (MCMC) sampling algorithms [1]. The implementation in R is fairly straightforward:

```
stan_glm(default ~ student + balance + income, data = data,
                     family = binomial(link = "logit"), y = T,
                     algorithm = "sampling",
                   warmup = 1000, iter = 10000, chains = 4, refresh = 10000)
```

As no priors are specified, *rstanarm*'s default priors are used by default:

```
    Priors for model 'flat.fit'
    ------
    Intercept (after predictors centered)
     ~ normal(location = 0, scale = 2.5)

    Coefficients
  Specified prior:
    ~ normal(location = [0,0,0], scale = [2.5,2.5,2.5])
  Adjusted prior:
    ~ normal(location = [0,0,0], scale = [5.48492,0.00517,0.00019])
------
See help('prior_summary.stanreg') for more details
```

Estimation Results are reported in Section 4.

2.3 Bayesian Model with Strong Priors

To evaluate the performance and advantages of using a Bayesian modelling approach, I also specify a second model which uses priors that are in a sense *data-driven*. Using estimation results of the Frequentist baseline (reported fully later), I specify a set of priors that are purposefully wrong in order to see how many observations are needed by this Bayesian model to successfully update these priors to a good posterior approximation of the actual data. The parameters used are deliberately changed based on the Frequentist baseline, from

[1] Appendix A reports details.

```
      term             estimate  std.error statistic   p.value
  <chr>               <dbl>      <dbl>     <dbl>     <dbl>
1 (Intercept) -10.9         0.492        -22.1   4.91e-108
2 student       -0.647      0.236         -2.74  6.19e-  3
3 balance        0.00574    0.000232      24.7   4.22e-135
4 income         0.00000303 0.00000820     0.370 7.12e-  1
```

to

```
data_driven_prior = normal(location = c(0.5, -0.1, -0.011),
       scale = c(0.236, 0.000232, 0.00000820), autoscale = F)
```

The means for the coefficients' respective normal distributions were chosen to have a mean that is opposite in sign and different in magnitude to the point estimates, but to have the same scale. The implementation is again simple:

```
stan_glm(default ~ student + balance + income, data = data,
                   family = binomial(link = "logit"), y = T,
                   algorithm = "sampling",
                   prior = data_driven_prior,
                   warmup = 1000, iter = 10000, chains = 4, refresh = 10000)
```

2.4 Strong Priors with Different Sample Sizes

This second model is re-estimated with identical informative priors on the two subsamples discussed earlier:

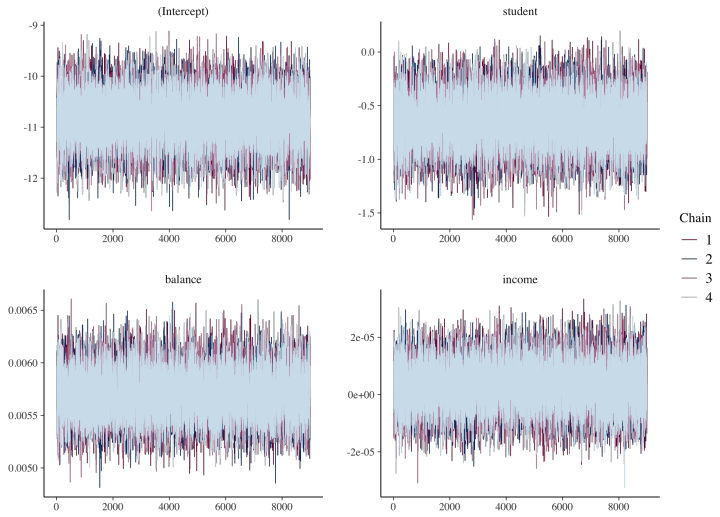
```
stan_glm(default ~ student + balance + income, data = subset1,
                    family = binomial(link = "logit"), y = T,
                    algorithm = "sampling",
                    prior = data_driven_prior,
                    warmup = 1000, iter = 10000, chains = 4, refresh = 0)

stan_glm(default ~ student + balance + income, data = subset2,
                    family = binomial(link = "logit"), y = T,
                    algorithm = "sampling",
                    prior = data_driven_prior,
                    warmup = 1000, iter = 10000, chains = 4, refresh = 0)
```

3 MCMC Diagnostics

3.1 Flat Priors

The Bayesian model with flat priors displays good MCMC covergence:

Figure 3: MCMC Trace Plot, Flat Priors



As Figure 3 shows, all 4 MCMC chains do not show flat portions or a trend over time, but rather display a white noise-like trace, indicating good mixing.

Figure 4: Geweke Diagnostic p-values, Flat Priors

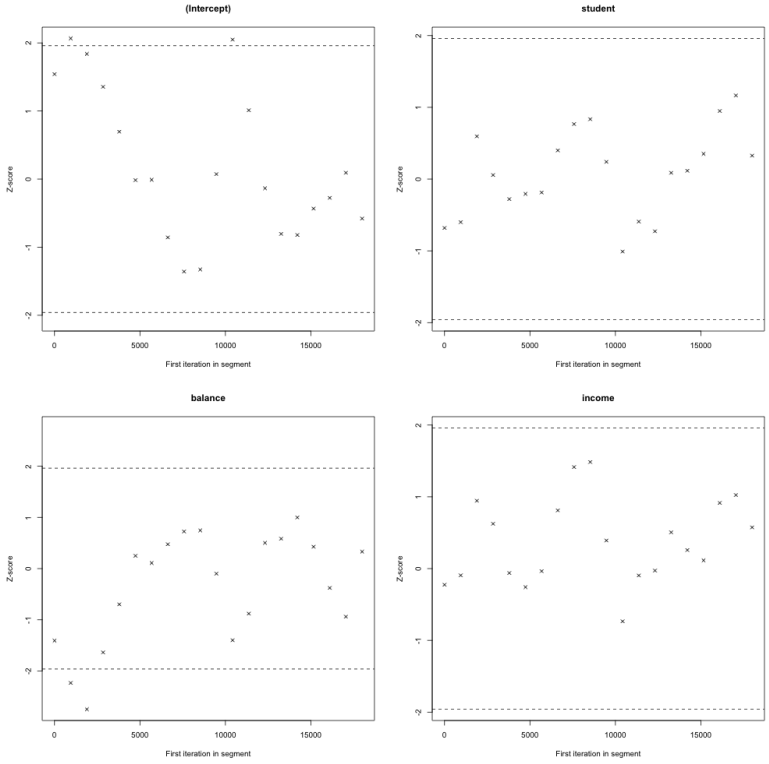


Figure 4 shows the z-scores produced by the Geweke Diagnostic (cf. Geweke 1991, p.9). As the vast majority of z-scores lie between the critical values at the 5% confidence level, we can conclude that there is no statistically significant difference between the means of samples drawn at the beginning of each chain and in the subsequent portions, which is indicative of good convergence to the target (i.e. posterior) distribution. It also shows that there is no burn-in required to achieve convergence for this model.

Figure 5: MCMC Autocorrelation, Flat Priors

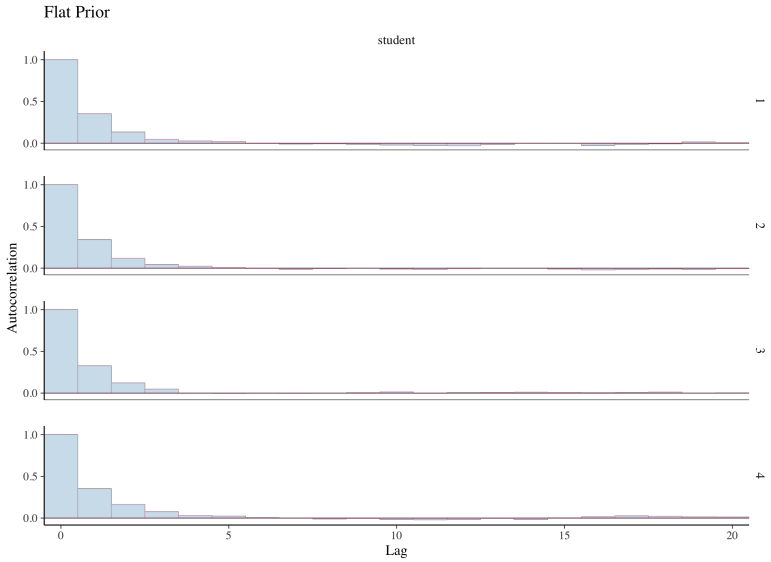


Figure 5 shows that there is barely any autocorrelation in the MCMC chains for predictor *student*, which shows the highest autocorrelation. The strongest autocorrelation is detected at lag 1, with autocorrelation subsiding to zero by lag 5 at the latest across all variables.

Figure 6: $\hat{R}$, Flat Priors

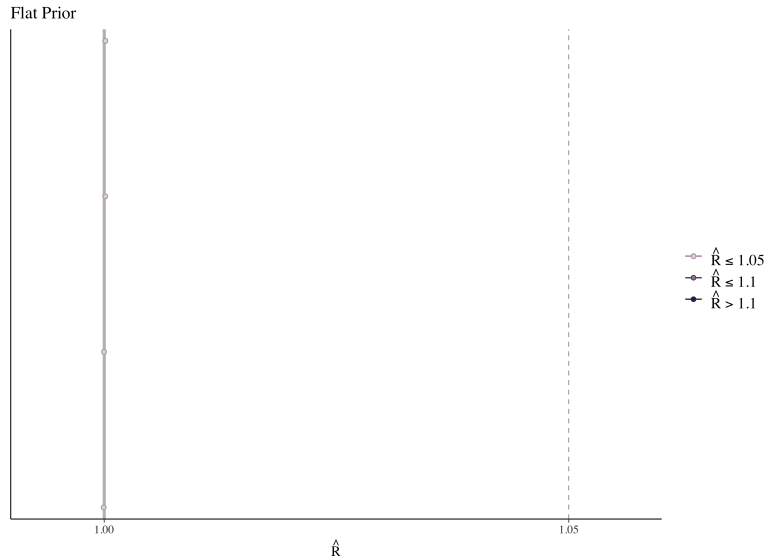


Figure 6 shows the Gelman-Rubin statistic (cf. Gelman and Rubin 1992) across all four variables. `Rrstan` recommends using only samples that show an $\hat{R} \leq 1.05$. As this model shows $\hat{R} = 1$ across all variables, this is again indicative of good convergence.

Figure 7: N_{eff}, Flat Priors

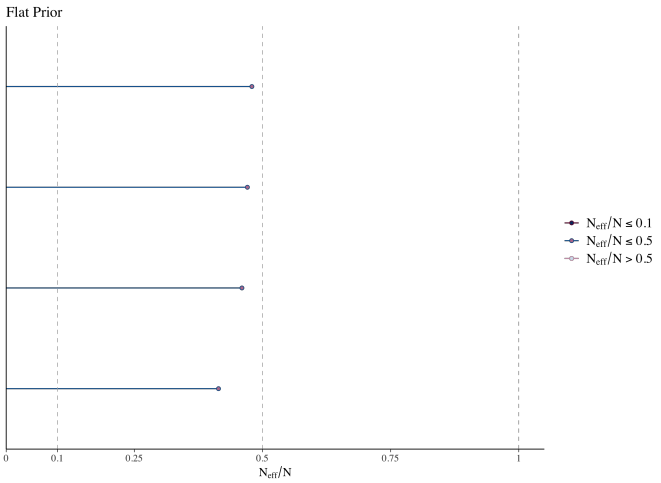
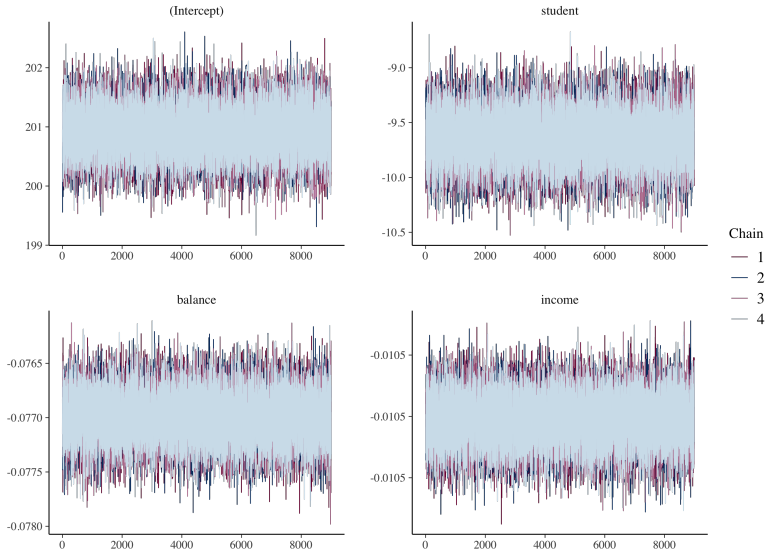


Figure 7 shows the ratio of the effective sample size N_{eff} to N across all four variables. Values of $\frac{N_{eff}}{N} \leq 0.5$ are commonly viewed as favorable for convergence, which this model displays.

3.2 Strong Priors

For the sake of brevity, this subsection simply displays the same plots as the previous subsection for the strong prior model, which also shows (very) favorable convergence properties.

Figure 8: MCMC Trace Plot, Strong Priors



As Figure 8 shows, all 4 MCMC chains do not show flat portions or a trend over time, but rather display a white noise-like trace, indicating good mixing.

Figure 9: Geweke Diagnostic p-values, Strong Priors

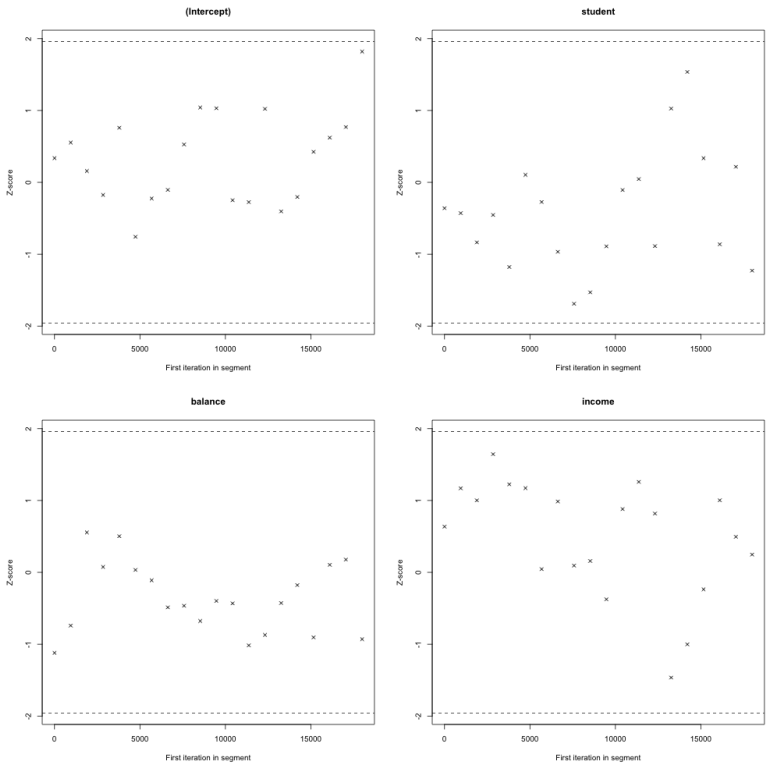


Figure 9 is again indicative of good convergence.

Figure 10: MCMC Autocorrelation, Strong Priors

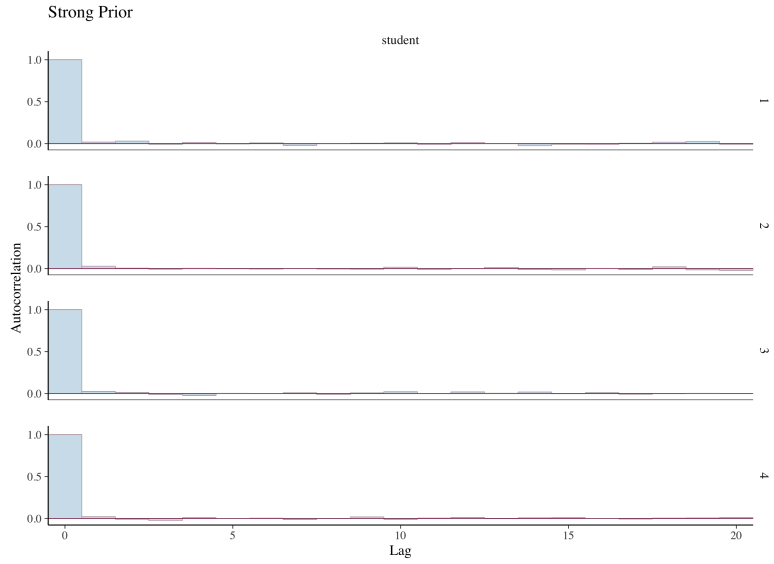


Figure 10 shows less autocorrelation for predictor *student*.

Figure 11: $\hat{R}$, Strong Priors

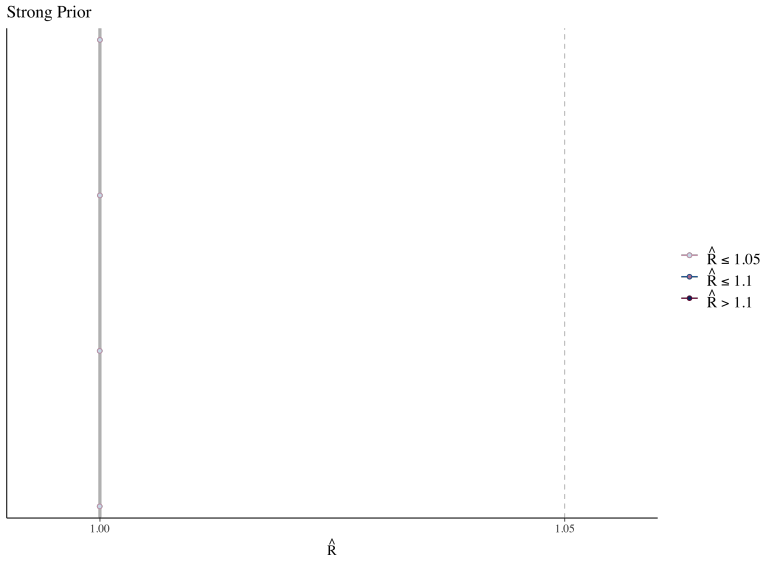


Figure 11 shows identical $\hat{R} = 1$ across all variables, this is again indicative of good convergence.

Figure 12: N_{eff}, Strong Priors

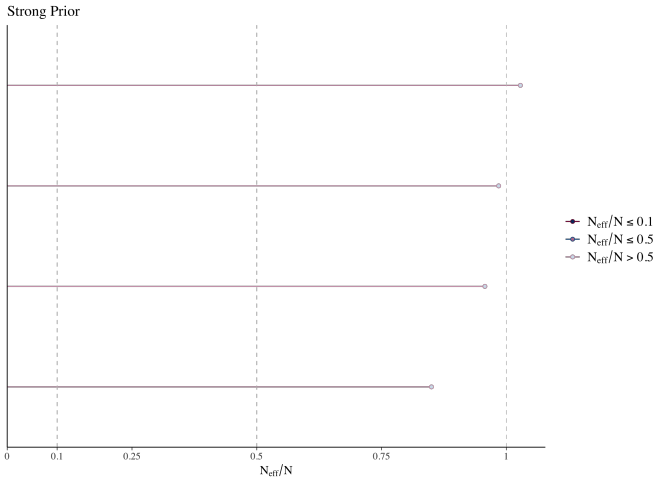


Figure 12 shows that $\frac{N_{eff}}{N}$ is higher than for the model using flat priors.

Figures 13 - 15 below display the same diagnostics for the model with strong priors estimated on the subsets with $1,000$ and $5,000$ observations. There are no distinguishable differences between these models in terms of autocorrelation, $\hat{R}$, and N_{eff}. This is somewhat surprising, as one could reasonably expect the model that uses only $1,000$ observations to converge less well than the ones using more data, but this seems not to be the case.

Figure 13: MCMC Autocorrelation by Sample Size

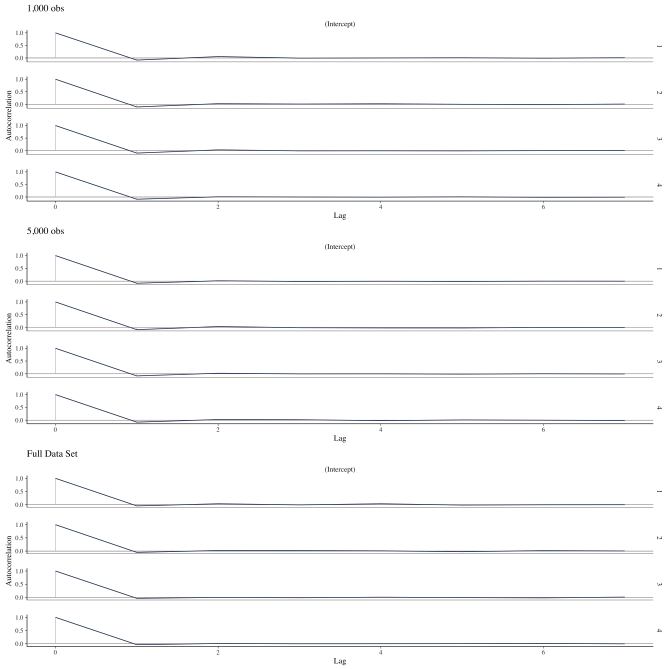


Figure 14: $\hat{R}$ by Sample Size

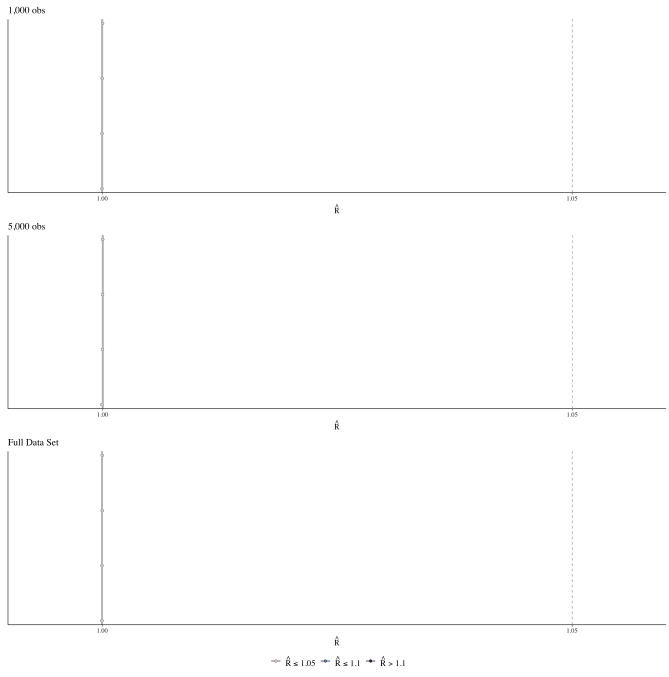
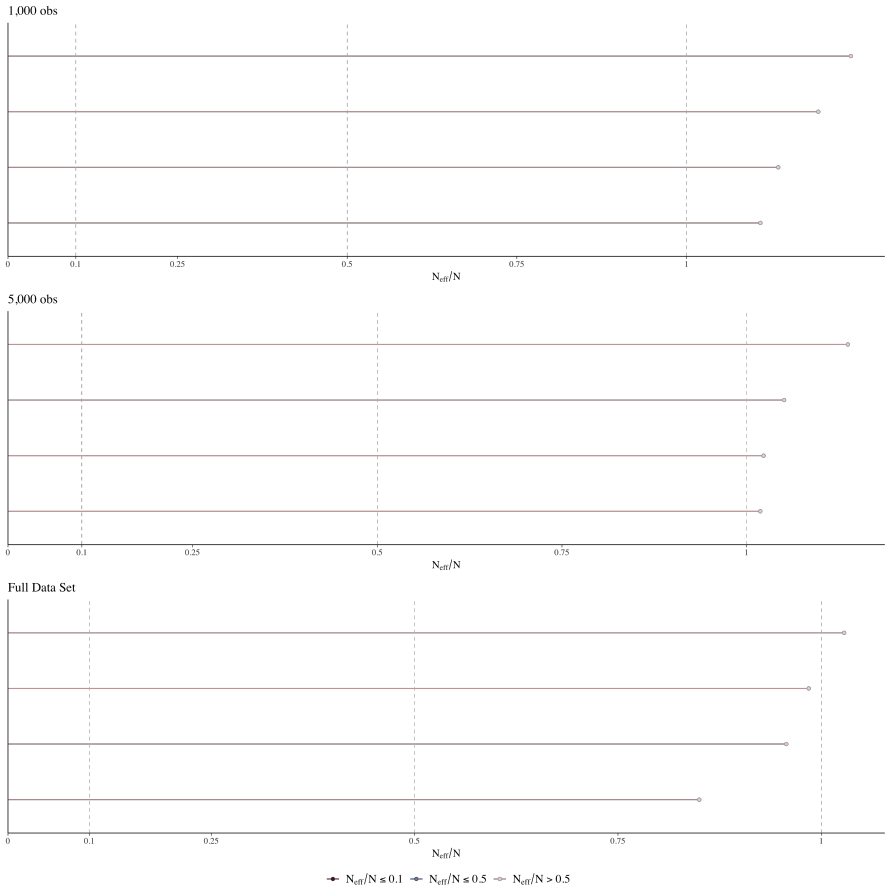


Figure 15: N_{eff} by Sample Size



In conclusion, there is no indication that either model needs burn-in draws or thinning to converge or mix, and therefore to predict well.

4 Estimation Results

This section reports coefficient estimates / posterior samples from all three models. As the main topic of interest of this report is the comparison of the models' respective strengths, and given that the data used are simulated, I will not dwell too much on the interpretation of parameter estimates.

4.1 Frequentist Baseline

Table 3 shows the point estimates and standard errors for eq. (2). Interestingly, *income* has an estimated coefficient of 0 and is the only variable that is not statistically significant. This can plausibly be explained by the correlated nature of *student* and *income*. Due to this correlation, *student*, the estimated coefficient of which is highly statistically significant, picks up the variance in *income*, which renders the predictive power of *income* useless.

Table 3: Baseline Estimation Results

	Dependent variable:
	default
student	-0.647^{***} (0.236)
balance	0.006^{***} (0.0002)
income	0.00000 (0.00001)
Constant	-10.900^{***} (0.492)
Observations	10,000
Log Likelihood	-786.000
Akaike Inf. Crit.	1,580.000
Note:	*p<0.1; **p<0.05; ***p<0.01

4.2 Bayesian Model with Flat Priors

Figure 16 shows histograms and correlation scatter plots for a sample ($n = 1,000$) of the posterior distribution of eq. (2) estimated using flat priors. As the histograms show, this sample of the posterior is approximately normal in distribution, which corresponds to the flat priors. The scatter plots do show some signs of correlation in the posterior sample, for instance between *student* and *income*. Overall, these results are to be expected: estimating this model with flat, i.e. $\mathcal{N}$-distributed priors, leads to a normally distributed posterior sample. Table 4 below reports parameter means, Standard Deviations, and confidence intervals.

Figure 16: Posterior Sample, Flat Priors

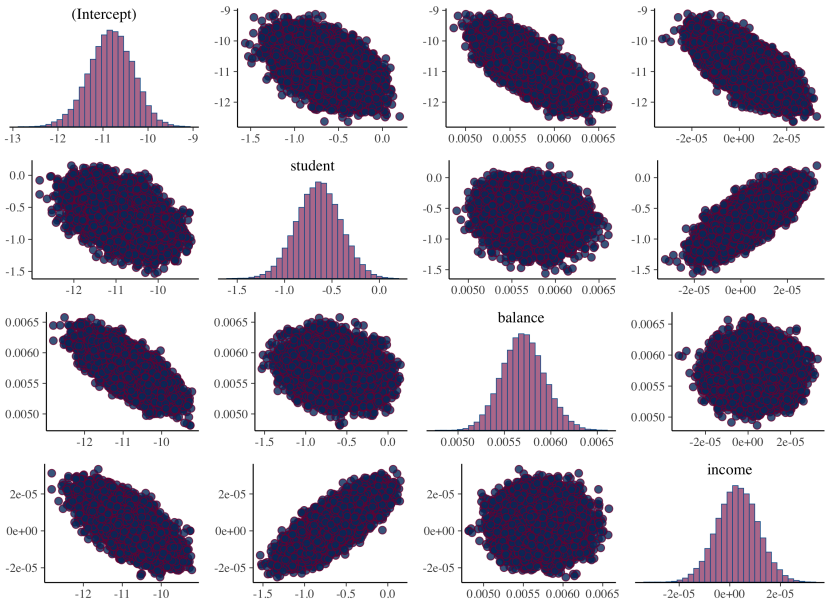


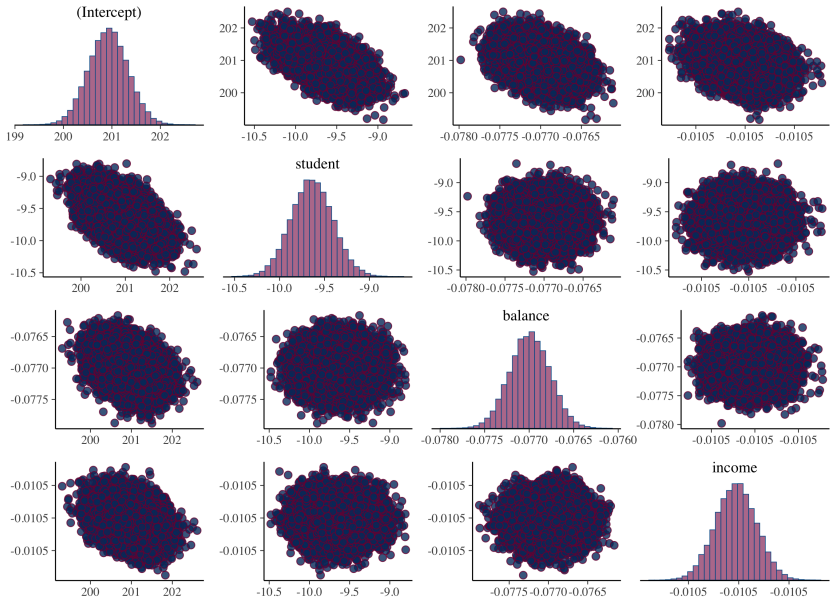
Table 4: Fit with Flat Priors

	mean	mcse	sd	10%	50%	90%	n_eff	Rhat
(Intercept)	-10.8220	0.0038	0.4861	-11.4481	-10.8138	-10.2072	16558	1
student	-0.6413	0.0018	0.2328	-0.9393	-0.6406	-0.3430	16941	1
balance	0.0057	0.0000	0.0002	0.0054	0.0057	0.0060	14912	1
income	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	17256	1
mean_PPD	0.0334	0.0000	0.0021	0.0307	0.0334	0.0361	37053	1
log-posterior	-796.0163	0.0128	1.4124	-797.9135	-795.6960	-794.5339	12209	1

4.3 Bayesian Model with Strong Priors

Figure 17 shows histograms and correlation scatterplots for a sample ($N = 1,000$) of the posterior distribution of eq. (2) estimated using data-driven / strong priors. Similarly to the flat prior model, the posterior sample is normally distributed for every parameter. Interestingly, this posterior does not show any correlation between any parameters to the extent that the one with flat priors does. Table 5 again shows summary values. The estimated parameter means are noticeably different from Table 4, while the SD are similar. This is explained by the specification of the priors, which use an incorrect mean, but accurate SD. As Tables 6 and 7 below show, the model with strong priors achieves results that are similar to the flat prior baseline using only $n = 5,000$, though the results are closer for the full data set.

Figure 17: Posterior Sample, Strong Priors



Tables 5-7 below show concrete values.

Table 5: Fit with Strong Priors, Full Data Set

	mean	mcse	sd	10%	50%	90%	n_eff	Rhat
(Intercept)	2.01e+02	0.0021	0.4031	2.00e+02	2.01e+02	2.01e+02	37011	1
student	-9.64e+00	0.0013	0.2320	-9.94e+00	-9.64e+00	-9.34e+00	34451	1
balance	-7.70e-02	0.0000	0.0002	-7.73e-02	-7.70e-02	-7.67e-02	35443	1
income	-1.05e-02	0.0000	0.0000	-1.05e-02	-1.05e-02	-1.05e-02	30597	1
mean_PPD	3.68e-02	0.0000	0.0005	3.62e-02	3.68e-02	3.74e-02	38346	1
log-posterior	-1.13e+05	0.0112	1.4083	-1.13e+05	-1.13e+05	-1.13e+05	15949	1

Table 6: Fit with Strong Priors, Subset 1

	mean	mcse	sd	10%	50%	90%	n_eff	Rhat
(Intercept)	2.32e+02	0.004	0.789	2.31e+02	2.32e+02	2.33e+02	40872	1
student	-1.47e+00	0.001	0.236	-1.77e+00	-1.47e+00	-1.17e+00	39924	1
balance	-9.70e-02	0.000	0.000	-9.70e-02	-9.70e-02	-9.70e-02	44727	1
income	-1.10e-02	0.000	0.000	-1.10e-02	-1.10e-02	-1.10e-02	42997	1
mean_PPD	6.70e-02	0.000	0.002	6.40e-02	6.70e-02	6.90e-02	40229	1
log-posterior	-1.44e+04	0.011	1.406	-1.44e+04	-1.44e+04	-1.44e+04	15914	1

Table 7: Fit with Strong Priors, Subset 2

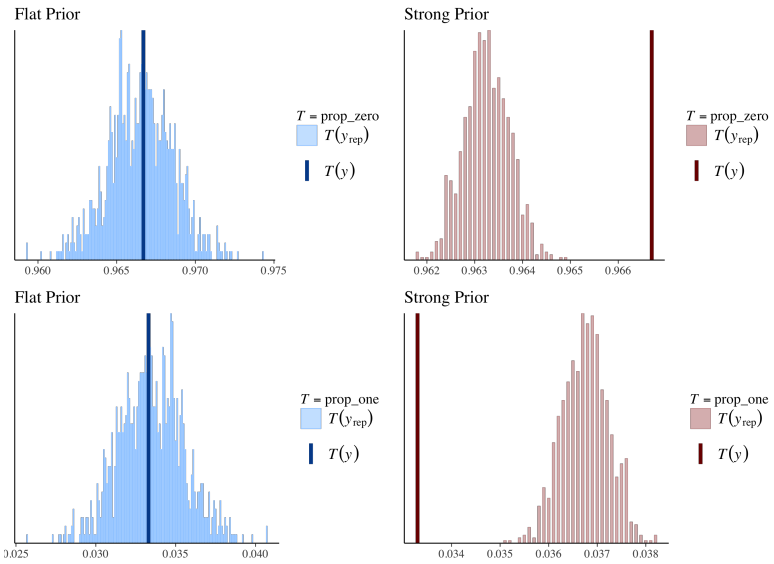
	mean	mcse	sd	10%	50%	90%	n_eff	Rhat
(Intercept)	2.08e+02	0.002	0.486	2.07e+02	2.08e+02	2.08e+02	40934	1
student	-4.76e+00	0.001	0.232	-5.06e+00	-4.76e+00	-4.46e+00	36665	1
balance	-8.90e-02	0.000	0.000	-8.90e-02	-8.90e-02	-8.80e-02	37825	1
income	-1.10e-02	0.000	0.000	-1.10e-02	-1.10e-02	-1.10e-02	36824	1
mean_PPD	3.80e-02	0.000	0.001	3.70e-02	3.80e-02	3.90e-02	38697	1
log-posterior	-5.83e+04	0.011	1.422	-5.83e+04	-5.83e+04	-5.83e+04	15305	1

5 Model Evaluation

5.1 Posterior Predictive Checks

Figure 18 below gives an impression of how well the two models fit the data. The left-hand side panels show the distribution of the posterior predictive distribution (PPD), which is the distribution of the outcome implied by the model after using the observed data to update our beliefs about the unknown parameters. The two dark-blue bars / dark-red in the upper and lower panels represent the true (i.e. observed) proportion of $default = 0$ and $default = 1$ in the data. The light blue /red distributions around this proportion are the respective PPD with $1,000$ draws each. As the left-hand panel shows, the model with flat priors generates a PPD that fairly accurately represents the true proportion. On the right-hand side panels, the same plots are displayed for the model with strong priors. For this model, which was specified with intentionally "wrong" means, the PPD is clearly off from the true proportion. Even though this model converges just fine and produces identical-looking posterior samples, it clearly fits the data substantially worse, even after updating the (wrong) priors with $10,000$ observations.

Figure 18: Response Proportion Comparison between Priors



Figures 19 and 20 below display an overlay of the PPD density for the models with flat and strong priors onto the empirical density. For both models, the PPD sample density is virtually indistinguishable from the observed density, which means that both models, even the one with wrong priors, fit the data seemingly well, as they can be used to simulate data that at least in density is close to identical to the observed data. However, Figure 18 above paints a different picture, as the model with strong priors fails to accurately predict the response proportions. As

this would be the main interest of any practical application, the density overlays are somewhat deceiving for this model.

Figure 19: Density Overlay Comparison between Priors

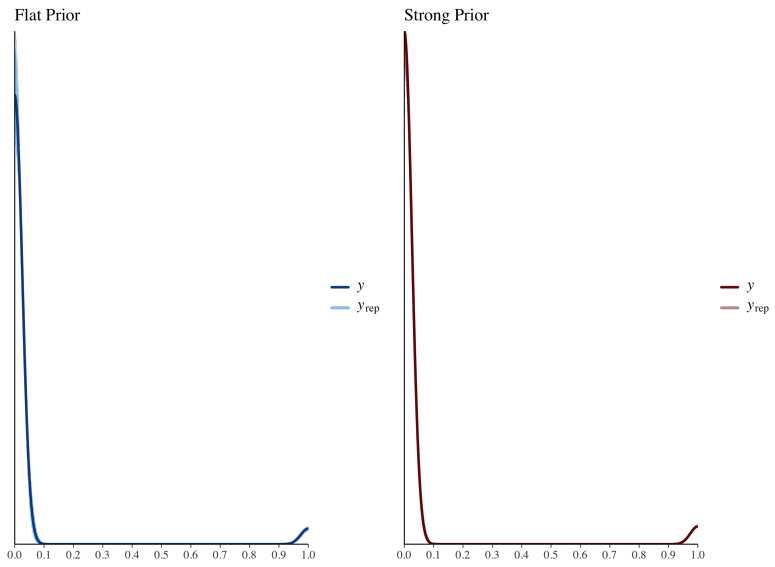


Figure 20: Discrete Density Overlay Comparison between Priors

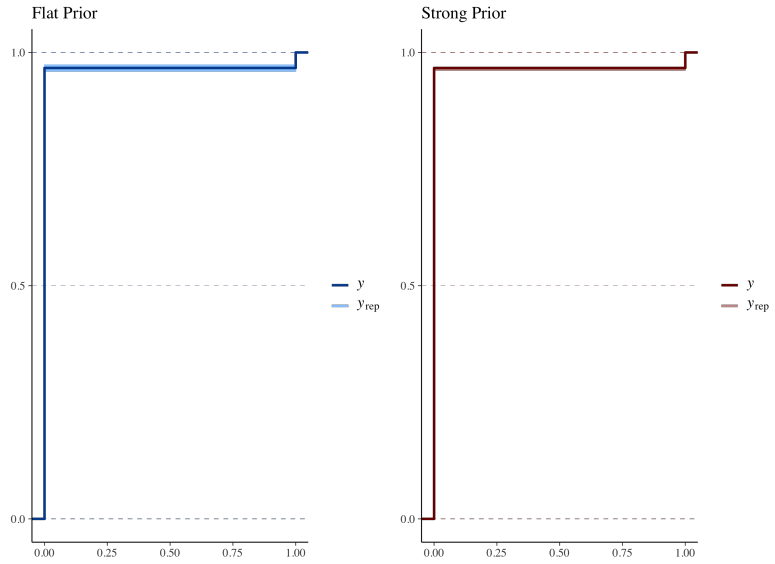
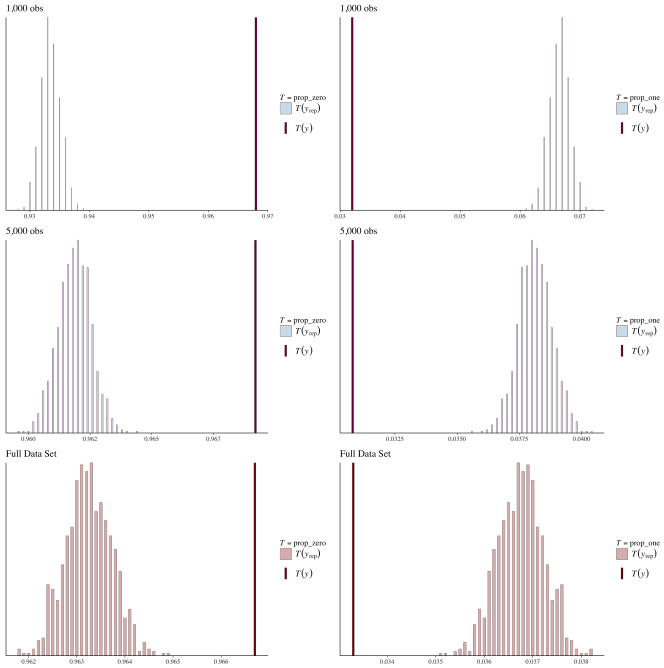


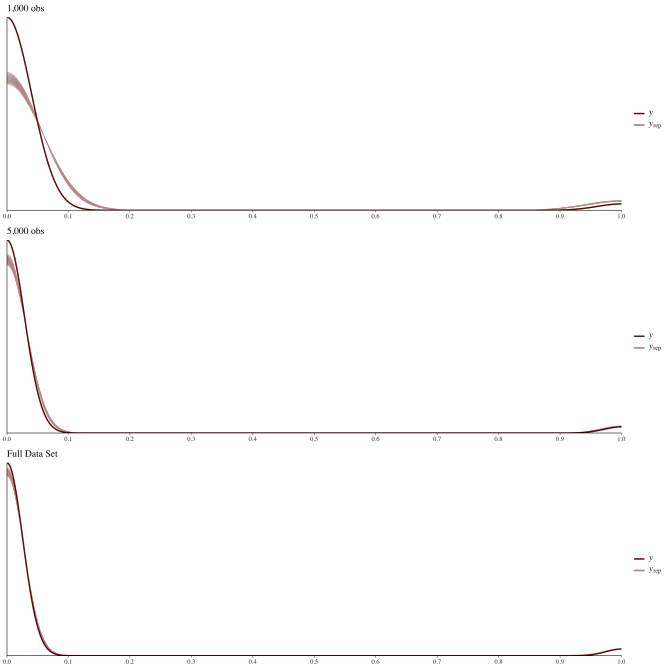
Figure 21 below compares the respective proportions of $default = 0$ and $default = 1$ between the respective PPDs and the actual data (sub)sets. While none of the models are particularly accurate, there is a visible improvement each time the model is given more data to update its intentionally wrong priors. This improvement can be spotted by the posterior sample moving closer to the true proportion the more observations there are.

Figure 21: Response Proportion Comparison by Sample Size



Finally, Figure 22 displays the overlay of the PPD density for the model with strong priors onto the empirical density by subsample. Similarly to the proportions, there is a clearly visible improvement every time we increase the sample size.

Figure 22: Density Overlay Comparison by Sample Size



5.2 Cross-Validation

Moving beyond graphical Posterior Predictive Checks, which ultimately rely on heuristics and judgement, this section attempts to quantify the goodness of fit of the models by using 10-fold Cross-Validation (cf. James, Witten, Hastie, and Robert Tibshirani 2021, ch. 5.1.3). To keep the computation tractable, I use $k = 10$ instead of $k = N$. a.k.a leave-one-out CV. Using `rstanarm::kfold()`, I can compare the performance of all Bayesian models:

Table 8: 10-fold CV Comparison

	ELPD	ELPD SE	# Params	# Params SE	N
Strong Priors, 1k obs	-2859	293.6	100.43	17.744	1000
Strong Priors, 5k obs	-9925	558.4	147.69	19.288	5000
Strong Priors, 10k obs	-19983	790.0	176.85	21.259	10000
Flat Priors	-790	38.9	3.98	0.678	10000

The first column displays the expected log pointwise predictive density (ELPD). Following Vehtari, Gelman, and Gabry (2017, section 2.3), `rstanarm::kfold()` partitions the data into $k = 10$ subsets $y\_k, k = 1, \dots K$ and fits the model separately to each trainig set y_{-k}, yielding a posterior $p_{post(-k)}(\theta) = p(\theta|y_{(-k)})$. For the typical value of $k = 10$, it is computationally cheap to refit the model separately each time. Vehtari, Gelman, and Gabry (2017, eq. 19) define predictive accuracy for each data point, which yields the log predictive density for $y_i \in k$:

$$logp(y_i|y_{(-k)}) = log \int p(y_i|\theta)p(\theta|y_{(-k)})d\theta, \; i \in k \tag{3}$$

If the posterior $[(\theta|y_{(k)}$ is summarized by S simulation draws $\theta^{k,s}$, the log predictive density is computed as

$$\widehat{elpd}_i = log\Big(\frac{1}{S}\sum_{i=1}^{S} p(y_i|\theta^{k,s})\Big) \tag{4}$$

using the simulations which correspond to subset $k \ni i$. Lastly, one sums to obtain the estimate

$$\widehat{elpd}_{xval} = \sum_{i=1}^{n} \widehat{elpd}_i \tag{5}$$

ELPD values themselves are mainly useful for comparing models. Though such a comparison is nontrivial, the difference between ELPD values shown in Table 8 are substantial. Following Sivula et al. (2022, p. 10ff), it is necessary to assume that both ELPD and the associated standard error (SE) based on normal approximation are are well-calibrated, which a.o. necessitates no gross misspecification in the models to be compared. Although one can reasonably conclude that the

Strong Prior model used here does not satisfy this condition, I will attempt to use Sivula et al. (2022)'s rough heuristic of significant differences between models for $|elpd_{diff}| > 4$, if$N > 100$ (cf. *If elpd_ diff/se_ diff > |2|, is this noteworthy?* 2021). Using this heuristic, there are very strong differences between the predicitvce accuracy of all models. The Flat Prior model shows the largest ELPD estimate with a narrow standard error (SE). The smallest difference to this baseline is surprisingly given by comparing this baseline to the model using strong priors fit to the smallest subset of $n = 1,000$. The difference is largest (i.e. several orders of magnitude) when comparing the flat prior model to the strong prior model fit on more data. This is yet again due to the (intentional) misspecification. Following Spiegelhalter et al. (2002, p. 583), *# Params* in Table 8 refers to the measure p_d of the "effective number of parameters in a model as the difference between the posterior mean of the deviance and the deviance at the posterior means of the parameters of interest". By this measure as well, the model with flat priors fits the data best by far.

6 Conclusion

This report has contrasted the implementation, results, posterior predictive checks, and cross-validation performance of one Bayesian model implemented with two different sets of priors. Both models mix and converge well, and at first glance yield similar results. This is somewhat unexpected given that the model estimated using strong priors was deliberately specified using incorrect priors to see how many observations are necessary for those priors to be dominated by the Likelihood. When comparing the models side-by-side, it is clear that while both have excellent MCMC diagnostics, only the flat prior model achieves good accuracy in representing the correct proportions of response values in the posterior. As expected, for larger sample sizes, the model with strong priors achieves better accuracy here. Similarly, for larger sample sizes, the strong prior model performs more closely to the baseline in all graphical Posterior Predictive Checks. It is only when comparing the 10-fold CV ELPD estimates that it becomes clear that the more observations are used to fit the strong prior model, the worse its predictive accuracy vis-à-vis the flat prior baseline becomes. Overall, the models yield results as expected, with the strong (that is, wrong) priors overwhelmed by the likelihood for larger N, but the resulting models still performing worse than the one fit using flat priors in terms of predictive accuracy as a result of the misspecified priors.

7 Software used

The implementation of this project used *R: A Language and Environment for Statistical Computing* v. 4.2.2 and relies on the following packages:

1. *tidyverse: Easily Install and Load the Tidyverse* v.1.3.2
2. *rstan: R Interface to Stan* v.2.21.8
3. *rstanarm: Bayesian Applied Regression Modeling via Stan* v.2.21.3
4. *bayesplot: Plotting for Bayesian Models* v.1.10.0
5. *coda: Output Analysis and Diagnostics for MCMC* v.0.19-4
6. *kableExtra: Construct Complex Table with kable and Pipe Syntax* v.1.3.4
7. *stargazer: Well-Formatted Regression and Summary Statistics Tables* v.5.2.3
8. *ISLR: Data for an Introduction to Statistical Learning with Applications in R* v.1.3
9. *caret: Classification and Regression Training* v.6.0-93

References

Gelman, Andrew and Donald B. Rubin (Nov. 1992). "Inference from Iterative Simulation Using Multiple Sequences". In: *Statistical Science* 7.4, pp. 457–472. ISSN: 0883-4237, 2168-8745. DOI: 10.1214/ss/1177011136. URL: https://projecteuclid.org/journals/statistical-science/volume-7/issue-4/Inference-from-Iterative-Simulation-Using-Multiple-Sequences/10.1214/ss/1177011136.full (visited on 01/29/2023).

Geweke, John F. (1991). "Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments". In: *Staff Report*. Number: 148 Publisher: Federal Reserve Bank of Minneapolis. URL: https://ideas.repec.org//p/fip/fedmsr/148.html (visited on 01/28/2023).

If elpd_ diff/se_ diff > |2|, is this noteworthy? (Jan. 31, 2021). The Stan Forums. URL: https://discourse.mc-stan.org/t/if-elpd-diff-se-diff-2-is-this-noteworthy/20549 (visited on 01/30/2023).

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2021). *An introduction to statistical learning: with applications in R*. Second edition. 1 online resource : illustrations (chiefly color) vols. Springer texts in statistics. New York: Springer. ISBN: 9781071614181 9781071614174. URL: https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=2985424 (visited on 01/27/2023).

Sivula, Tuomas et al. (Mar. 17, 2022). *Uncertainty in Bayesian Leave-One-Out Cross-Validation Based Model Comparison*. DOI: 10.48550/arXiv.2008.10296. arXiv: 2008.10296[stat]. URL: http://arxiv.org/abs/2008.10296 (visited on 01/30/2023).

Spiegelhalter, David J. et al. (2002). "Bayesian measures of model complexity and fit". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 64.4, pp. 583–639. ISSN: 1467-9868. DOI: 10.1111/1467-9868.00353. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00353 (visited on 01/30/2023).

Vehtari, Aki, Andrew Gelman, and Jonah Gabry (Sept. 2017). "Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC". In: *Statistics and Computing* 27.5, pp. 1413–1432. ISSN: 0960-3174, 1573-1375. DOI: 10.1007/s11222-016-9696-4. arXiv: 1507.04544[stat]. URL: http://arxiv.org/abs/1507.04544 (visited on 01/30/2023).

Wooldridge, Jeffrey M. (2020). *Introductory econometrics: a modern approach*. Seventh edition. 1 online resource (xxii, 826 pages : illustrations, black and white, and colour) vols. Victoria, Australia: Cengage. ISBN: 978-1-337-67133-0. URL: http://www.vlebooks.com/vleweb/product/openreader?id=none&isbn=9781337671330 (visited on 01/27/2023).

8 Appendix A: Failed *rstan* Attempt

(Back to Model) This Appendix contrasts the machine-generated `rstanrm::stan_glm()` *STAN* ode with my attempt at implementing the same model in `rstan::sampling()`, documented by the R and STAN files used. It is unclear to me why implementing the model in *rstan* did not work. The only sampling algorithm available in *rstanarm*, the wrapper method that worked perfectly fine, is Hamiltonian Monte Carlo (HMC) (cf. cf. *rstanarm: Bayesian Applied Regression Modeling via Stan*). Using `rstan::get_stanmodel(flat.fit\$stanfit)` yields the following result:

Machine-Generated STAN code

```
    4 class stanmodel 'bernoulli' coded as follows:
#include /pre/Columbia_copyright.stan
#include /pre/license.stan

// GLM for a Bernoulli outcome
functions {
#include /functions/common_functions.stan
#include /functions/bernoulli_likelihoods.stan
}
data {
  // dimensions
  int<lower=0> K;        // number of predictors
  int<lower=0> N[2];     // number of observations where y = 0 and y = 1 respectively
  vector[K] xbar;        // vector of column-means of rbind(X0, X1)
  int<lower=0,upper=1> dense_X; // flag for dense vs. sparse
  matrix[N[1],K] X0[dense_X];   // centered (by xbar) predictor matrix | y = 0
  matrix[N[2],K] X1[dense_X];   // centered (by xbar) predictor matrix | y = 1

  int<lower=0, upper=1> clogit; // 1 iff the number of successes is fixed in each stratum
  int<lower=0> J; // number of strata (possibly zero)
  int<lower=1,upper=J> strata[clogit == 1 ? N[1] + N[2] : 0];

  // stuff for the sparse case
  int<lower=0> nnz_X0;                      // number of non-zero elements in the implicit X0 ma
  vector[nnz_X0] w_X0;                      // non-zero elements in the implicit X0 matrix
```

```
  int<lower=1, upper = K> v_X0[nnz_X0];      // column indices for w_X0
  // where the non-zeros start in each row of X0
  int<lower=1, upper = rows(w_X0) + 1> u_X0[dense_X ? 0 : N[1] + 1];
  int<lower=0> nnz_X1;                       // number of non-zero elements in the implicit X1 ma
  vector[nnz_X1] w_X1;                       // non-zero elements in the implicit X1 matrix
  int<lower=1, upper = K> v_X1[nnz_X1];      // column indices for w_X1
  // where the non-zeros start in each row of X1
  int<lower=1, upper = rows(w_X1) + 1> u_X1[dense_X ? 0 : N[2] + 1];
  // declares prior_PD, has_intercept, link, prior_dist, prior_dist_for_intercept
#include /data/data_glm.stan

  int<lower=0> K_smooth;
  matrix[N[1], K_smooth] S0;
  matrix[N[2], K_smooth] S1;
  int<lower=1> smooth_map[K_smooth];

  int<lower=5,upper=5> family;

[ommited for brevity]

generated quantities {
  real mean_PPD = compute_mean_PPD ? 0 : negative_infinity();
  real alpha[has_intercept];

  if (has_intercept == 1) {
    if (dense_X) alpha[1] = gamma[1] - dot_product(xbar, beta);
    else alpha[1] = gamma[1];
  }

  if (compute_mean_PPD) {
    vector[N[1]] pi0;
    vector[N[2]] pi1;
    // defines eta0, eta1
#include /model/make_eta_bern.stan
    if (has_intercept == 1) {
```

```
      if (link != 4) {
        eta0 += gamma[1];
        eta1 += gamma[1];
      }
      else {
        real shift;
        shift = fmax(max(eta0), max(eta1));
        eta0 += gamma[1] - shift;
        eta1 += gamma[1] - shift;
        alpha[1] -= shift;
      }
    }
    if (clogit) for (j in 1:J) mean_PPD += successes[j]; // fixed by design
    else {
      pi0 = linkinv_bern(eta0, link);
      pi1 = linkinv_bern(eta1, link);
      for (n in 1:N[1]) mean_PPD += bernoulli_rng(pi0[n]);
      for (n in 1:N[2]) mean_PPD += bernoulli_rng(pi1[n]);
    }
    mean_PPD /= NN;
  }
}
```

Upon comparing this code to the one I tried to use (Section 8 below), I can spot no clear explanation why *rstanarm*'s HMC algorithm worked while *rstan*'s did not. I can also not see any explanation as to why *rstan*'s NUTS sampling algorithm did not work, as it is supposedly more easily cmputationally tractable (cf. *rstan: R Interface to Stan*, `sampling()` documentation).

R code

```
## Tobias Schnabel ##
## i6255807 ##

rm(list = ls(all = TRUE)) #CLEAR ALL

####Housekeeping####
packages <- c("tidyverse", "broom", "ggpubr", "stargazer", "caret", "rstan",
              "rstanarm", "kableExtra", "bayesplot", "coda", "ISLR")
```

```r

#Comment in lines below to Install packages not yet installed
# installed_packages <- packages %in% rownames(installed.packages())
# if (any(installed_packages == FALSE)) {
#   install.packages(packages[!installed_packages])
# }

#load packages
invisible(lapply(packages, library, character.only = TRUE))

#set options
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)

#load Credit Card Default Data Set
attach(Default)

# tidy
#make factors numerical
data = Default %>%
  mutate(default=ifelse(default=="No", 0,1)) %>%
  mutate(student=ifelse(student=="No", 0,1))

#df summary statistics
stargazer(data, type = "text")
#prepare data for STAN
#make recipe
rec = recipe(default ~ student + balance + income, data = data) %>%
  prep(retain = T)

#extract X matrix and y vector
X = juice(rec, all_predictors(), composition = 'matrix')
y = juice(rec, all_outcomes(), composition = 'matrix') %>% drop()

#feed data into STAN
stan_data <- list(
  X = X,
  K = ncol(X),
  N = nrow(X),
  y = y,
  use_y_rep = T,
  use_log_lik = F
)

```

```
#initialize models
stan.mod = stan_model('Final_Assignment_Schnabel.stan')

#flat priors WITHOUT income variable
# does not work
# flat.fit.hmc = sampling(stan.mod, data = stan_data,
#                         algorithm = "HMC",
#                      warmup = 1000, iter = 10000, chains = 1, thin = 1)

#does work, but does not perform MCMC
flat.fit.fixedparam = sampling(stan.mod, data = stan_data,
                               algorithm = "Fixed_param",
                               warmup = 1000, iter = 10000, chains = 4, thin = 1)

#does not work
# flat.fit.nuts = sampling(stan.mod, data = stan_data,
#                          algorithm = "NUTS",
#                          warmup = 1000, iter = 10000, chains = 1, thin = 1)


fixedparam_params = rstan::extract(flat.fit.fixedparam)
monitor(flat.fit.fixedparam)

#check proportions of 0s and ones
ppc_stat(y, yrep.flat, stat = "prop_zero", binwidth = 0.00005)
ppc_stat(y, yrep.flat, stat = "prop_one", binwidth = 0.00005)

#check posterior.flat trace
color_scheme_set("mix-blue-pink")
mcmc_trace(flat.fit.fixedparam)
mcmc_pairs(flat.fit.fixedparam)

# mcmc diagnostics
# rhat
plot(flat.fit.fixedparam, "rhat")
plot(flat.fit.fixedparam, "rhat_hist")
# ratio of effective sample size to total posterior.flat sample size
plot(flat.fit.fixedparam, "neff")
plot(flat.fit.fixedparam, "neff_hist")
# autocorrelation by chain
plot(flat.fit.fixedparam, "acf", pars = "(Intercept)")
plot(flat.fit.fixedparam, "acf_bar", pars = "(Intercept)")
mcmc_acf(flat.fit.fixedparam)
```

STAN code

```
data {
  int <lower = 0> N; // Defining the number of defects in the test dataset
  // response
  int <lower = 0, upper = 1> y [N];
  // number of columns in the design matrix X
  int <lower = 0> K;
  // design matrix X
  // does not include an intercept
  matrix [N, K] X;
  //keep responses
  int use_log_lik;
  int use_y_rep;
}
parameters {
  // The (unobserved) model parameters that we want to recover
  real alpha;
  vector[K] beta;
}
transformed parameters {
  vector[N] eta;
  eta = alpha + X * beta;
}
model {
  // multiple logistic regression model
  y ~ bernoulli_logit(eta);

  // Prior models for the unobserved parameters
  // alpha ~ normal(0, 1);
  // beta ~ normal(1, 1);
}
generated quantities {
  // simulate data from the posterior
  vector[N * use_y_rep] y_rep;
  // log-likelihood posterior
  vector[N * use_log_lik] log_lik;
  for (i in 1:num_elements(y_rep)) {
    y_rep[i] = bernoulli_rng(inv_logit(eta[i]));
  }
  for (i in 1:num_elements(log_lik)) {
    log_lik[i] = bernoulli_logit_lpmf(y[i] | eta[i]);
  }
}
```

9 Appendix B: Main Script

Last executed January 31, 2023, 00:36, runtime of 7.89 minutes on 10 cores.

```
## Tobias Schnabel ##
## i6255807 ##

rm(list = ls(all = TRUE)) #CLEAR ALL

####Housekeeping####
packages <- c("tidyverse", "broom", "ggpubr", "stargazer", "caret", "rstan",
              "rstanarm", "kableExtra", "bayesplot", "coda", "ISLR")

#Comment in lines below to Install packages not yet installed
# installed_packages <- packages %in% rownames(installed.packages())
# if (any(installed_packages == FALSE)) {
#   install.packages(packages[!installed_packages])
# }

#load packages
invisible(lapply(packages, library, character.only = TRUE))

#set options
options(digits = 3)
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)

#load Credit Card Default Data Set
attach(Default)

#record start time
start.time = Sys.time()

# tidy
#make factors numerical
data = Default %>%
  mutate(default=ifelse(default=="No", 0,1)) %>%
  mutate(student=ifelse(student=="No", 0,1))

#df summary statistics
stargazer(data, type = "text")

#generate datasets with fewer obs to compare models
intrain_1k = caret::createDataPartition(
```

```
  y = data$default,
  p = 0.1,
  list = F
)

intrain_5k = caret::createDataPartition(
  y = data$default,
  p = 0.5,
  list = F
)

subset1 = data[intrain_1k,]
subset2 = data[intrain_5k,]

#verify proportions
props = rbind(table(subset1$default)[2]/table(subset1$default)[1],
              table(subset2$default)[2]/table(subset2$default)[1],
              table(data$default)[2]/table(data$default)[1])
nrows = rbind(nrow(subset1), nrow(subset2), nrow(data))

#create matrix to export later
data_integrity = as.matrix(cbind(nrows, props))
rownames(data_integrity) = c("Subset 1", "Subset 2", "Original Data")
colnames(data_integrity) = c("n_obs", "Proportion of Defaults")
print(data_integrity)

#estimate logit baseline
form = formula(default ~ student + balance + income)

baseline = glm(form, data = data, family = binomial(link = "logit"))
tidy(baseline)


####flat priors####

#fit stan model
flat.fit = stan_glm(default ~ student + balance + income, data = data,
                    family = binomial(link = "logit"), y = T,
                    algorithm = "sampling",
                  warmup = 1000, iter = 10000, chains = 4, refresh = 10000)

#generate yrep for this prior
yrep.flat = posterior_predict(flat.fit, draws = 1000)

```

```
#extract posterior
posterior.flat = as.matrix(flat.fit)

#generate tidy df for use with ggplot
plotposterior.flat = as.data.frame(flat.fit) %>%
  reshape2::melt(measure.vars = 1:4)



####strong / data-driven priors####
tidy(baseline)
#set data-driven priors: means and SD taken from baseline logit output
data_driven_prior = normal(location = c(0.5, -0.1, -0.011),
           scale = c(0.236, 0.000232, 0.00000820), autoscale = F)

#estimate models with data-driven / strong prior
#fit strong priors (data-driven) on ***FULL data set***
strong.fit = stan_glm(default ~ student + balance + income, data = data,
                      family = binomial(link = "logit"), y = T,
                      algorithm = "sampling",
                      prior = data_driven_prior,
                      warmup = 1000, iter = 10000, chains = 4, refresh = 10000)

yrep.strong = posterior_predict(strong.fit, draws = 1000) #gen yrep
posterior.strong = as.matrix(strong.fit) #extract posterior

#generate tidy df for ggplot
plotposterior.strong = as.data.frame(strong.fit) %>%
  reshape2::melt(measure.vars = 1:4)

#fit strong priors (data-driven) on  ***SUBSET 1***
strong.fit.s1 = stan_glm(default ~ student + balance + income, data = subset1,
                       family = binomial(link = "logit"), y = T,
                       algorithm = "sampling",
                       prior = data_driven_prior,
                       warmup = 1000, iter = 10000, chains = 4, refresh = 0)

yrep.strong.s1 = posterior_predict(strong.fit.s1, draws = 1000) #gen yrep
posterior.strong.s1 = as.matrix(strong.fit.s1) #extract posterior

#fit strong priors (data-driven) on ***SUBSET 2***
strong.fit.s2 = stan_glm(default ~ student + balance + income, data = subset2,
                       family = binomial(link = "logit"), y = T,
                       algorithm = "sampling",
```

```
                        prior = data_driven_prior,
                        warmup = 1000, iter = 10000, chains = 4, refresh = 0)

yrep.strong.s2 = posterior_predict(strong.fit.s2, draws = 1000) #gen yrep
posterior.strong.s2 = as.matrix(strong.fit.s2) #extract posterior


####COMPARE RESULTS####
#monitor results
monitor(posterior.flat)

#look at flat priors
prior_summary(flat.fit)

#look at strong priors
prior_summary(strong.fit)

#monitor results
monitor(posterior.strong)
cat("", sep = "\n") # print empty line for readability
#monitor results subset 1
monitor(posterior.strong.s1)
cat("", sep = "\n") # print empty line for readability

#monitor results subset 2
monitor(posterior.strong.s2)
cat("", sep = "\n") # print empty line for readability

#Geweke Test
geweke.diag(posterior.flat)
geweke.diag(posterior.strong)
geweke.diag(posterior.strong.s1)
geweke.diag(posterior.strong.s2)

#Geweke plots
geweke.plot(as.mcmc(posterior.flat))
geweke.plot(as.mcmc(posterior.strong))
geweke.plot(as.mcmc(posterior.strong.s1))
geweke.plot(as.mcmc(posterior.strong.s2))


#compare 10-fold cv with diff sample sizes, this way of performing 10-fold cv
#adds an attribute to each model
######NOTE*****this will take quite a while to run
```

```r
#I would recommend lowering k to 3-4 unless strong compute is available
# reason why I chose k = 10: my machine has 10 cores, which this function utilizes
# reason why not LOOCV: too computationally expensive
flat.fit$loo = kfold(flat.fit, k = nrow(data))
strong.fit$loo = kfold(strong.fit, k = nrow(data))
strong.fit.s1$loo = kfold(strong.fit.s1, k = 10)
strong.fit.s2$loo = kfold(strong.fit.s2, k = nrow(subset2))

#compare
loocv.comp = loo_compare(flat.fit, strong.fit)
strong.fit.s1$loo
strong.fit.s2$loo

####Graphical PPC####
# do plots
#define custom functions for plots below
prop_zero <- function(x) mean(x == 0)
prop_one <- function(x) mean(x == 1)
source('Plots.R')

####Housekeeping Pt2####
#export plots and Tables and copy code files
if (Sys.info()[7] == "ts") {
  #this code only executes on my machine to prevent errors
  source('Tables.R')
  source('Plot.Export.R')
  setwd('/Users/ts/Git/ise')

  #copy code files to overleaf
  file.copy('Main.R', '/Users/ts/Library/CloudStorage/Dropbox/Apps/Overleaf/ISE_Assignment/Code',
  ↪  overwrite = T)
  file.copy('scrap_file.R',
  ↪  '/Users/ts/Library/CloudStorage/Dropbox/Apps/Overleaf/ISE_Assignment/Code', overwrite = T)
  file.copy('Final_Assignment_Schnabel.stan',
  ↪  '/Users/ts/Library/CloudStorage/Dropbox/Apps/Overleaf/ISE_Assignment/Code', overwrite = T)

  #copy bib of packages and dependencies
  knitr::write_bib(c(.packages()),
                   "/Users/ts/Dropbox/Apps/Overleaf/ISE_Assignment/packages.bib")
}

#record end time
end.time = Sys.time()
print("Time Elapsed: ")
```

```
214 print(end.time-start.time)
215 ####Show Plots####
216 #display plots (run each line to show plots, might take a few seconds)
217 phf
218 dof
219 dodf
220 propcomp
221 denscomp
222 discretedenscomp
223 rhatcomp
224 neffcomp
225 acfcomp
226 do_sample_comp
227
228 #Baseline Regression Diagnostic Plots
229 plot(baseline)
230
231
```