PPG Paint Colors: Final Project

Example: read data, save, and reload model object

Grishma Palkar

Overview

This RMarkdown shows how to read in the final project data. It also shows how to calculate the logit-transformed response and setup the binary outcome for use with `caret` or `tidymodels`. It also demonstrates how to fit a simple model (with `lm()`), save that model, and load it back into the workspace. You may find these actions helpful as you work through the project.

You must download the data from Canvas and save the data in the same directory as this RMarkdown file.

Load packages

This example uses the `tidyverse` suite of packages.

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
## ✔ ggplot2 3.4.0      ✔ purrr   1.0.1
## ✔ tibble  3.2.1      ✔ dplyr   1.1.2
## ✔ tidyr   1.3.0      ✔ stringr 1.5.0
## ✔ readr   2.1.3      ✔ forcats 0.5.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(rstanarm)
```

```
## Loading required package: Rcpp
## This is rstanarm version 2.21.4
## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!
## - Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.
## - For execution on a local, multicore CPU with excess RAM we recommend calling
##   options(mc.cores = parallel::detectCores())
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(coefplot)
```

```
##
## Attaching package: 'coefplot'
##
## The following object is masked from 'package:rstanarm':
##
##     invlogit
```

Read data

Please download the final project data from Canvas. If this Rmarkdown file is located in the same directory as the downloaded CSV file, it will be able to load in the data for you. It is **highly** recommended that you use an RStudio RProject to easily manage the working directory and file paths of the code and objects associated with the final project.

The code chunk below reads in the final project data.

```
df <- readr::read_csv("paint_project_train_data.csv", col_names = TRUE)
```

```
## Rows: 835 Columns: 8
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (2): Lightness, Saturation
## dbl (6): R, G, B, Hue, response, outcome
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The `readr::read_csv()` function displays the data types and column names associated with the data. However, a glimpse is shown below that reveals the number of rows and also shows some of the representative values for the columns.

```
df %>% glimpse()
```

```
## Rows: 835
## Columns: 8
## $ R          <dbl> 172, 26, 172, 28, 170, 175, 90, 194, 171, 122, 0, 88, 144, …
## $ G          <dbl> 58, 88, 94, 87, 66, 89, 78, 106, 68, 151, 121, 140, 82, 163…
## $ B          <dbl> 62, 151, 58, 152, 58, 65, 136, 53, 107, 59, 88, 58, 132, 50…
## $ Lightness  <chr> "dark", "dark", "dark", "dark", "dark", "dark", "dark", "da…
## $ Saturation <chr> "bright", "bright", "bright", "bright", "bright", "bright",…
## $ Hue        <dbl> 4, 31, 8, 32, 5, 6, 34, 10, 1, 21, 24, 22, 36, 16, 26, 12, …
## $ response   <dbl> 12, 10, 16, 10, 11, 16, 10, 19, 14, 25, 14, 19, 14, 38, 15,…
## $ outcome    <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,…
```

The data consist of continuous and categorical inputs. The `glimpse()` shown above reveals the data type for each variable which state to you whether the input is continuous or categorical. The RGB color model inputs, `R`, `G`, and `B` are continuous (dbl) inputs. The HSL color model inputs consist of 2 categorical inputs, `Lightness` and `Saturation`, and a continuous input, `Hue`. Two outputs are provided. The continuous output, `response`, and the Binary output, `outcome`. However, the data type of the Binary outcome is numeric because the Binary `outcome` is **encoded** as `outcome = 1` for the EVENT and `outcome = 0` for the NON-EVENT.

Regression task

(iiB) Bayesian Linear model

As stated in the project guidelines, you will **not** model the continuous output, `response`, directly. The `response` is a bounded variable between 0 and 100. The `response` must be transformed to an unbounded variable to appropriately be modeled by a Gaussian likelihood. We are making this transformation because we want the **uncertainty** in the predicted output to also satisfy output constraints. If we did not make this transformation the uncertainty could violate the bounds, which would mean the model is providing unphysical results! By logit-transforming `response`, we will fully respect the bounds of the output variable.

The code chunk below assembles the data for Part ii) of the project. You should use this data set for all regression modeling tasks. The logit-transformed output is named `y`. The `dfii` dataframe as the original `response` and Binary output, `outcome`, removed. This way you can focus on the variables specific to the regression task.

```
dfii <- df %>%
  mutate(y = boot::logit( (response - 0) / (100 - 0) ) ) %>%
  select(R, G, B,
         Lightness, Saturation, Hue,
         y)
```

```
dfii_train <- dfii %>%
  select(R,G,B,Hue,y) %>%
  scale() %>% as.data.frame() %>%
  bind_cols(dfii %>% select(Lightness,Saturation))
```

##(1) Fit 2 Bayesian linear models – one must be the best model from iiA) and the second must be another model you fit in iiA).State why you chose the second model. Ans: I chose model 9 as my second model because it had the second lowest RMSE value in Part iiA.

```
#model 07
BLM_07 <- stan_lm(y ~ (R+G+B+Hue)*(R+G+B+Hue)*(Lightness+Saturation), data = dfii_train,
                 prior = R2(location = 0.5),
                 seed = 432123)
```

```
##
## SAMPLING FOR MODEL 'lm' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.8e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.28 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 17.0706 seconds (Warm-up)
## Chain 1:                5.22327 seconds (Sampling)
## Chain 1:                22.2939 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'lm' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.4e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 7.90315 seconds (Warm-up)
## Chain 2:                8.23185 seconds (Sampling)
## Chain 2:                16.135 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'lm' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 8.83454 seconds (Warm-up)
## Chain 3:                5.48406 seconds (Sampling)
## Chain 3:                14.3186 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'lm' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 10.0616 seconds (Warm-up)
## Chain 4:                10.7238 seconds (Sampling)
## Chain 4:                20.7854 seconds (Total)
## Chain 4:
```

```
BLM_07 %>% readr::write_rds("BLM_07.rds")
```

```
#model 09
BLM_09 <- stan_lm(y ~ (R+G+B+Lightness + Saturation + Hue)*(R+G+B+Lightness + Saturation + Hue) , data = d
fii_train,
                 prior = R2(location = 0.5),
                 seed = 432123)
```

```
##
## SAMPLING FOR MODEL 'lm' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.2 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 6.31451 seconds (Warm-up)
## Chain 1:                5.34278 seconds (Sampling)
## Chain 1:                11.6573 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'lm' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 5.74915 seconds (Warm-up)
## Chain 2:                4.24222 seconds (Sampling)
## Chain 2:                9.99137 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'lm' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 8.66729 seconds (Warm-up)
## Chain 3:                4.25505 seconds (Sampling)
## Chain 3:                12.9223 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'lm' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.6e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 7.42072 seconds (Warm-up)
## Chain 4:                5.9132 seconds (Sampling)
## Chain 4:                13.3339 seconds (Total)
## Chain 4:
```

```
BLM_09 %>% readr::write_rds("BLM_09.rds")
```

##(2) After fitting the 2 models, you must identify the best model. Which performance metric did you use to make your selection? (a) I used the WAIC metric. The Widely Applicable Information Criterion (WAIC) is an approach that makes no assumptions about the posterior distribution. It calculates the performance of the model and penalizes that performance based on effective number of parameters. The penalty fully accounts for the joint posterior distribution of the parameters and the effect on the predictive behavior.

```
BLM_07$waic <- waic(BLM_07)
```

```
## Warning:
## 73 (8.7%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
BLM_09$waic <- waic(BLM_09)
```

```
## Warning:
## 50 (6.0%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

The model objects are assigned as elements in the stanreg_list() function in the code chunk below. The model_names argument allows specified more informative names associated with each model.

```
my_models <- stanreg_list(BLM_07, BLM_09,
                          model_names = c("Model 7", "Model 9"))
```

We can now directly compare the five models based on their information criterion. The results are ordered automatically from the best to the worst models based on the WAIC metric.

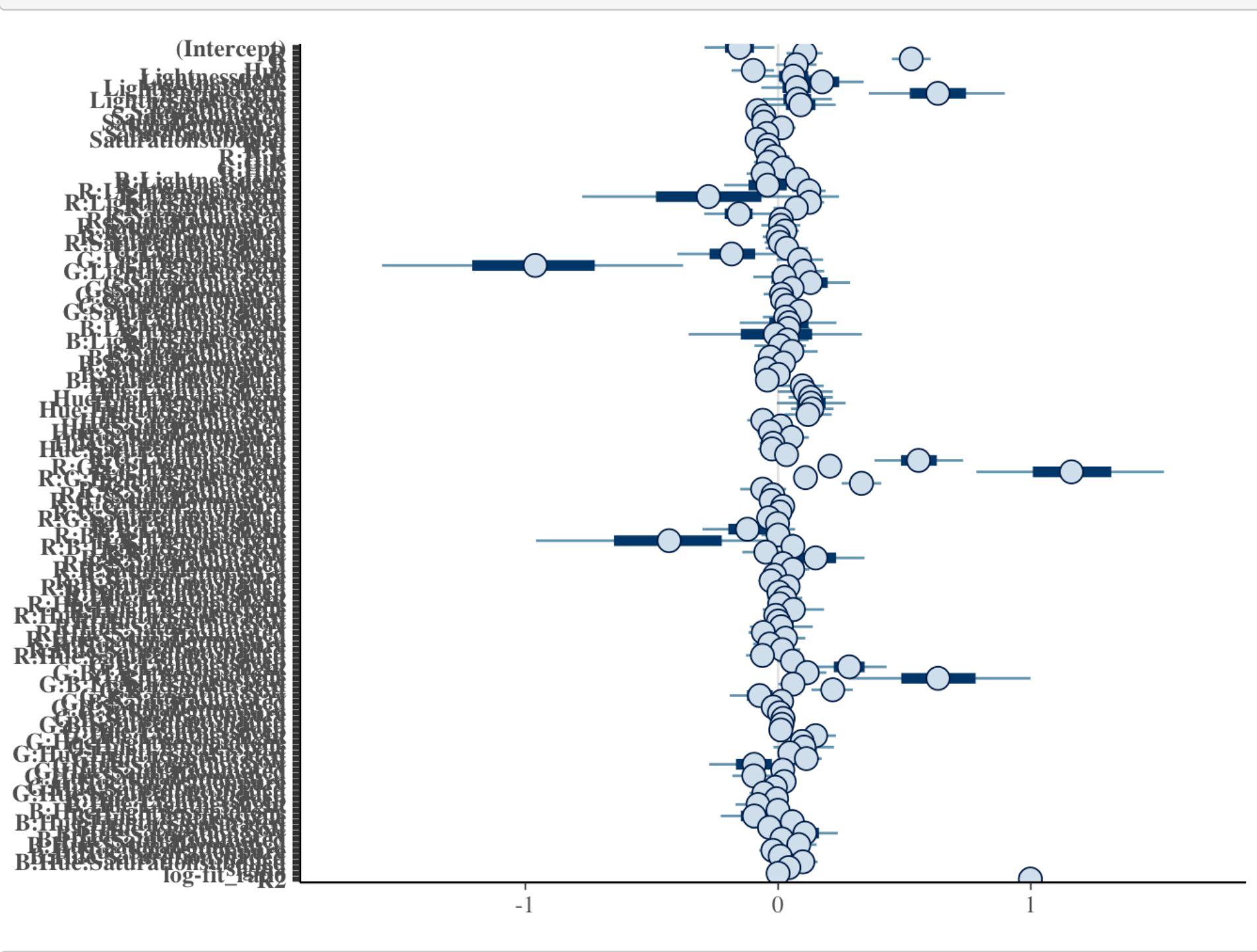
```
loo_compare(my_models, criterion = "waic")
```

```
## Model comparison based on WAIC:
##         elpd_diff se_diff
## Model 7    0.0       0.0
## Model 9 -126.1      20.8
```

So, according to WAIC, Model 7 is the best model.

(3) Visualize the regression coefficient posterior summary statistics for your best model

```
#model 7
plot(BLM_07)
```



```
bayes_R2(BLM_07) %>% quantile(c(0.05, 0.5, 0.95))
```

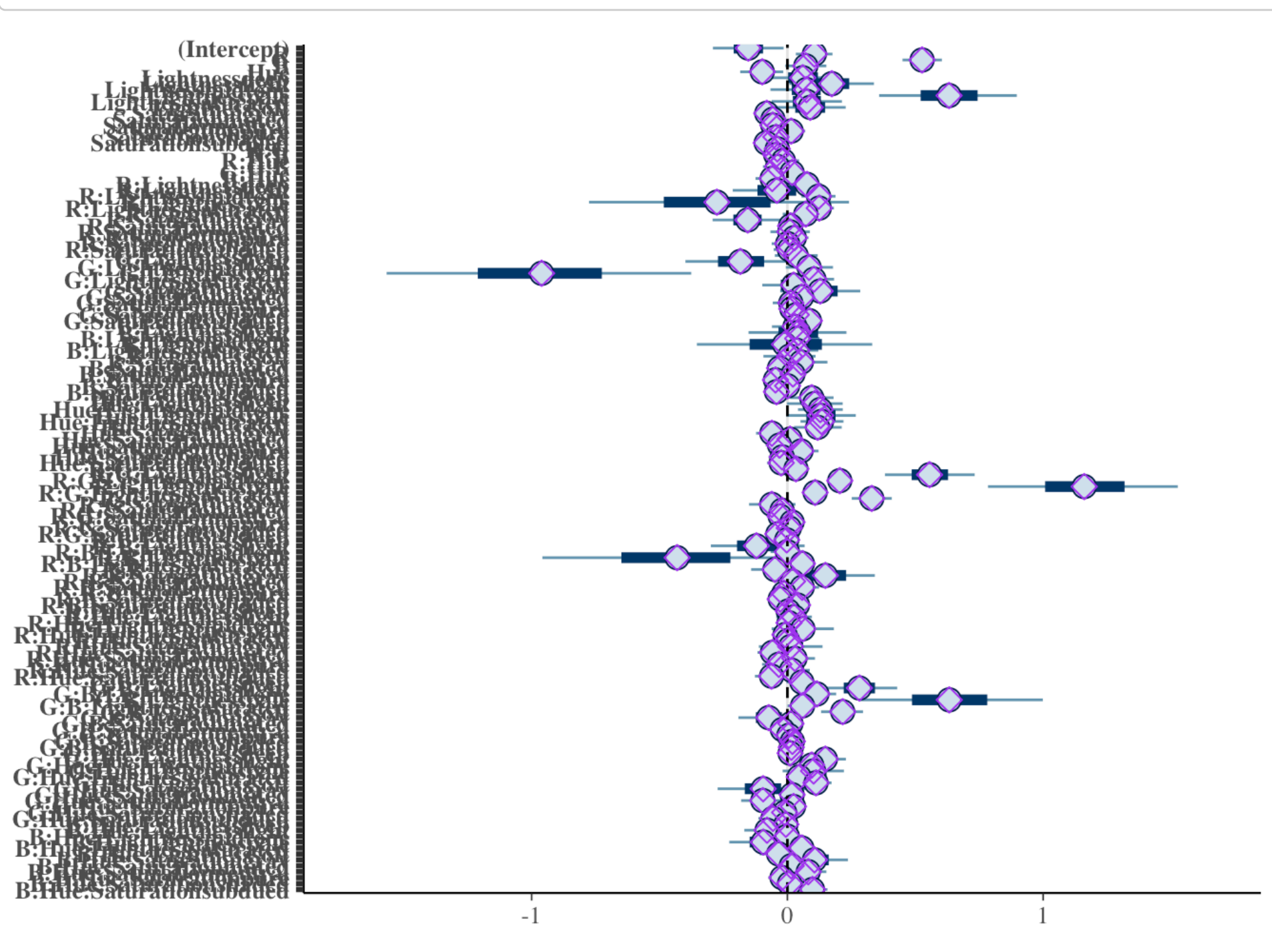
```
##        5%       50%       95%
## 0.9979948 0.9981962 0.9983606
```

The above data changed our prior belief about the model uncertainty (R-squared). For R-squared, posterior lower 5th quantile at 0.9979610, while our prior belief was 0.5.

##Let us visualize the posterior regression coefficient plot.

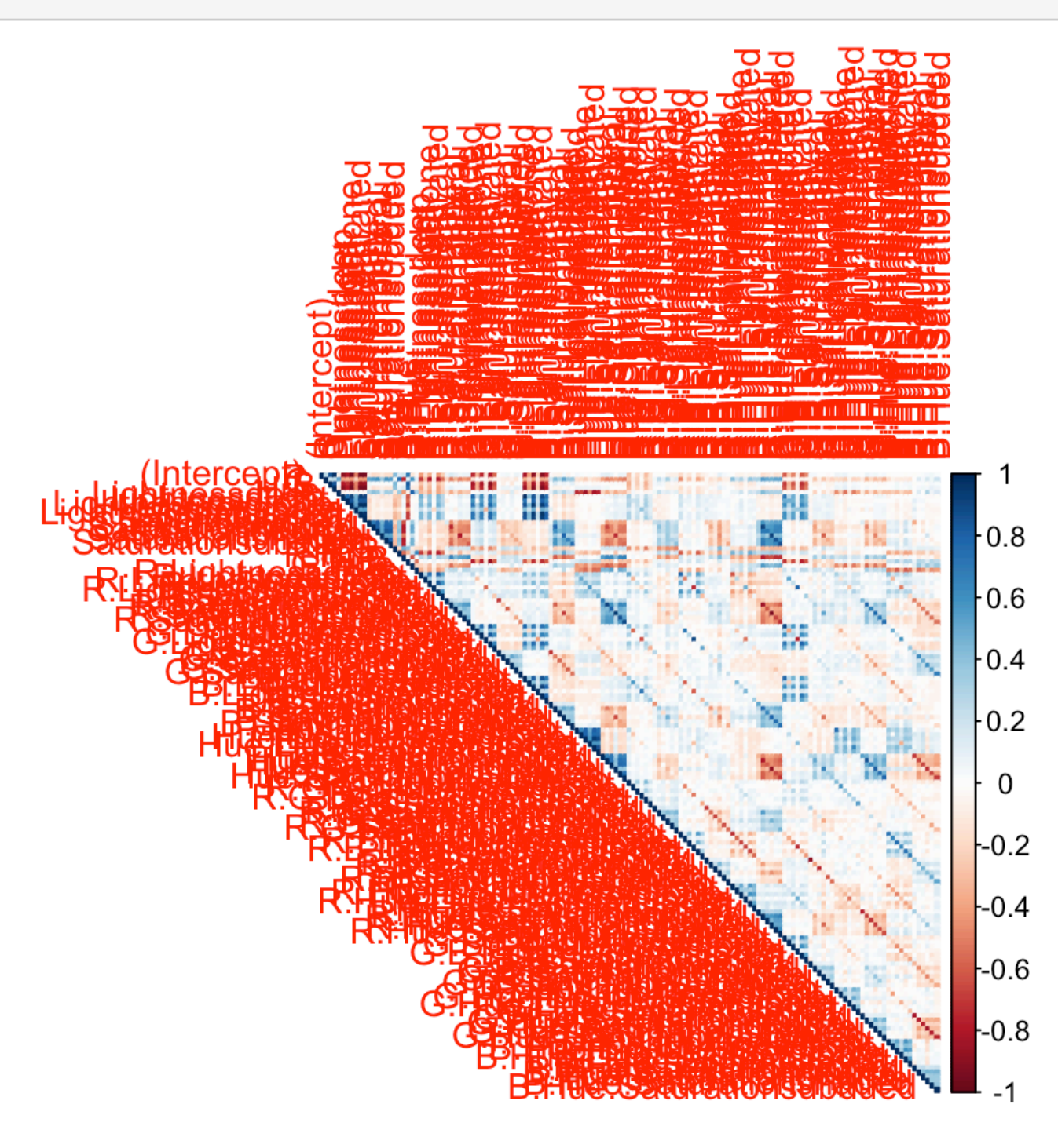
```
plot(BLM_07, pars = names(BLM_07$coefficients)) +
  geom_vline(xintercept = 0, color = "black", linetype = "dashed", size = 0.5) +
  geom_point(
    data = as_tibble(coef(BLM_07), rownames="coef"), aes(x = value, y = coef),
    shape = 5,
    size = 3,
    color = "purple"
  )
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



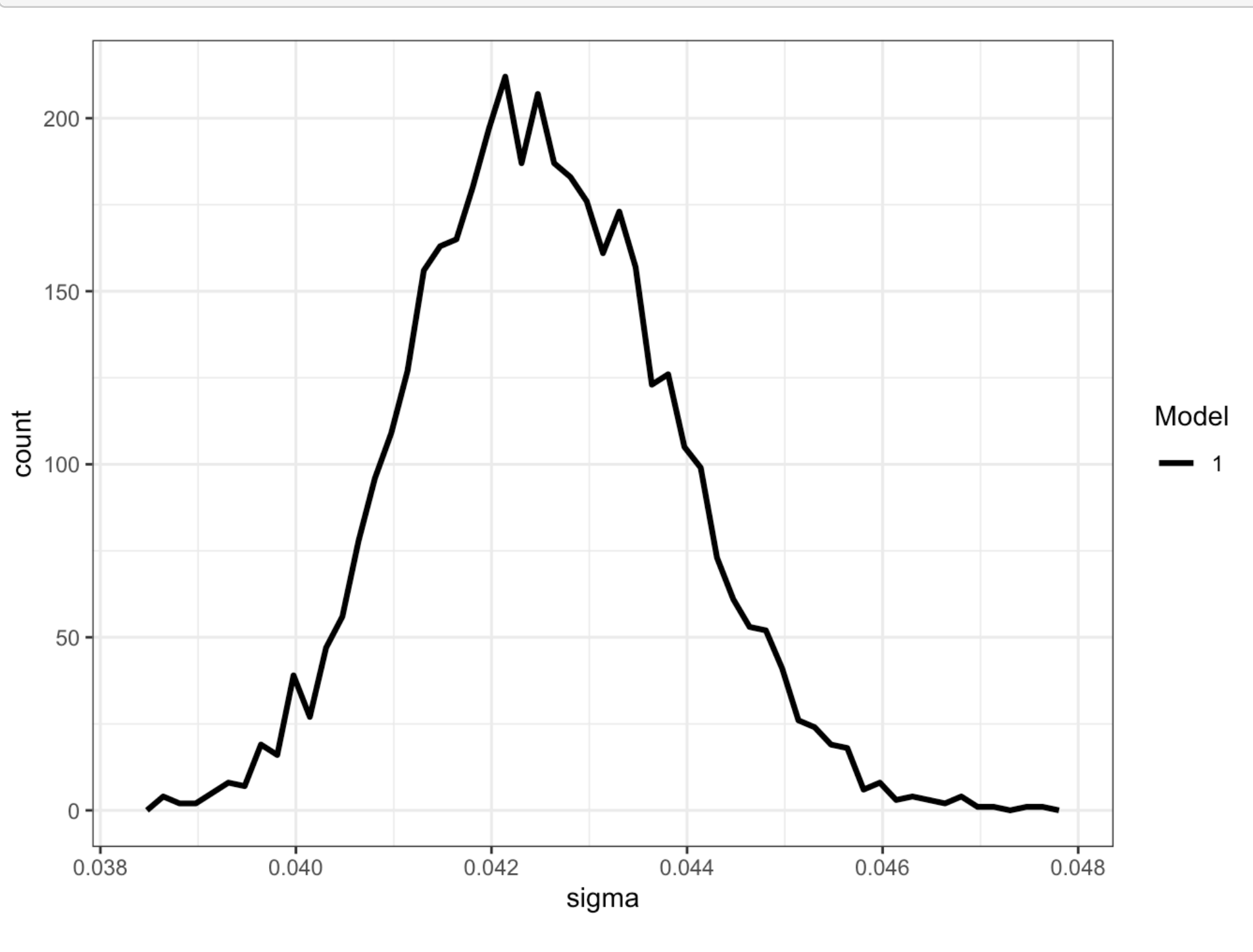
##Visualize the correlation of the posterior coefficients.

```
BLM_07 %>% as_tibble() %>%
  select(all_of(names(BLM_07$coefficients))) %>%
  cor() %>% corrplot(method="color", type="upper")
```



##(4)For your best model: Study the posterior UNCERTAINTY on the likelihood noise (residual error), σ. (a) How does the lm() maximum likelihood estimate (MLE) on σ relate to the posterior UNCERTAINTY on σ? Ans: let us visualize and compare posterior distribution of σ for our best model.

```
purrr::map2_dfr(list(BLM_07),
                as.character(1),
                function(mod, mod_name){as.data.frame(mod) %>% tibble::as_tibble() %>%
                    select(sigma) %>%
                    mutate(model_name = mod_name)}) %>%
  ggplot(mapping = aes(x = sigma)) +
  geom_freqpoly(bins = 55,
                mapping = aes(color = model_name),
                size = 1.1) +
  ggthemes::scale_color_colorblind("Model") +
  theme_bw()
```



b. Do you feel the posterior is precise or are we quite uncertain about σ? Ans.

```
mod07<-readr::read_rds("model07.rds")
sprintf("Posterior mode of sigma : %.3f", sigma(BLM_07))
```

```
## [1] "Posterior mode of sigma : 0.042"
```

```
sprintf("MLE estimated of sigma : %.3f", sigma(mod07))
```

```
## [1] "MLE estimated of sigma : 0.038"
```

```
mod_sigma <- BLM_07 %>% as_tibble() %>% select(sigma)
sprintf("Probability of posterior sigma > MLE sigma: %.3f%%", mean(mod_sigma >= sigma(mod07))*100)
```

```
## [1] "Probability of posterior sigma > MLE sigma: 100.000%"
```

For the best model identified using WAIC BLM_a (model 07): The posterior mode of σ from the Bayesian model is around 0.042 . The corresponding MLE estimated of σ is 0.036. The MLE underestimated σ i.e there is 100% chance σ is greater than the MLE estimate, judging from the posterior samples.