Module 11: Model checking

This Rmd contains model checking for parts 1, doing in-sample checks, and part 2, approximate leave-one-out validation.

Radon data

Read in the radon data and process (copied from earlier module)

```r
# house level data
d <- read.table(url("http://www.stat.columbia.edu/~gelman/arm/examples/radon/srrs2.dat"),
                header=T, sep=",")

# deal with zeros, select what we want, make a fips (county) variable to match on
d <- d %>%
  mutate(activity = ifelse(activity==0, 0.1, activity)) %>%
  mutate(fips = stfips * 1000 + cntyfips) %>%
  dplyr::select(fips, state, county, floor, activity)

# county level data
cty <- read.table(url("http://www.stat.columbia.edu/~gelman/arm/examples/radon/cty.dat"),
                  header = T, sep = ",")
cty <-
  cty %>%
  mutate(fips = 1000 * stfips + ctfips) %>%
  dplyr::select(fips, Uppm) %>%
  rename(ura_county = (Uppm))

dmn <- d %>%
  filter(state=="MN") %>% # Minnesota data only
  dplyr::select(fips, county, floor, activity) %>%
  left_join(cty)

dat <-
  dmn%>%
  mutate(y = log(activity), log_ur = log(ura_county))
head(dat)
```

```
##    fips               county floor activity ura_county         y     log_ur
## 1 27001 AITKIN                   1      2.2   0.502054 0.7884574 -0.6890476
## 2 27001 AITKIN                   0      2.2   0.502054 0.7884574 -0.6890476
## 3 27001 AITKIN                   0      2.9   0.502054 1.0647107 -0.6890476
## 4 27001 AITKIN                   0      1.0   0.502054 0.0000000 -0.6890476
## 5 27003 ANOKA                    0      3.1   0.428565 1.1314021 -0.8473129
## 6 27003 ANOKA                    0      2.5   0.428565 0.9162907 -0.8473129
```

Model fitting

To illustrate the use of model checks, we fit the following models:

```
fit <- brm(y ~ 1, family = gaussian(),
            data = dat, #sample_prior = T,
              seed = 123, # need to add seed here to make this reproducible
            chains = 4,
            iter = 1000, thin = 1,
            cores = getOption("mc.cores", 4),
            file = "output/mod11_fit")
```

Model with county intercepts only

```
fit_county <- brm(y ~ (1|county), family = gaussian(),
            data = dat, #sample_prior = T,
              seed = 123, # need to add seed here to make this reproducible
            chains = 4,
            iter = 1000, thin = 1,
            cores = getOption("mc.cores", 4),
            file = "output/mod11_fit_county")
```

Everything

```
fit_all <- brm(y ~ (1 + floor|county) + log_ur*floor, family = gaussian(),
            data = dat, #sample_prior = T,
              seed = 123, # need to add seed here to make this reproducible
            chains = 4,
            iter = 2000, thin = 1,
            cores = getOption("mc.cores", 4),
             file = "output/mod11_fit_all")
```

In-sample checks using replicated data sets

Obtain replicated data sets

Note that a lot of outputs can be generated directly with the brm-fit-object as well, as illustrated below. However, when fitting your own stan model, your starting point is working with repeated data sets. So I'm adding that approach here as well. (It may also help, even if you use brm, to be able to produce your own results).

```
ynew_si <- posterior_predict(fit) # adding si to indicate the dimension used
dim(ynew_si)
```

```
## [1] 2000  927
```

```
ynew_all_si <- posterior_predict(fit_all)
```

```
ynew_county_si <- posterior_predict(fit_county)
```

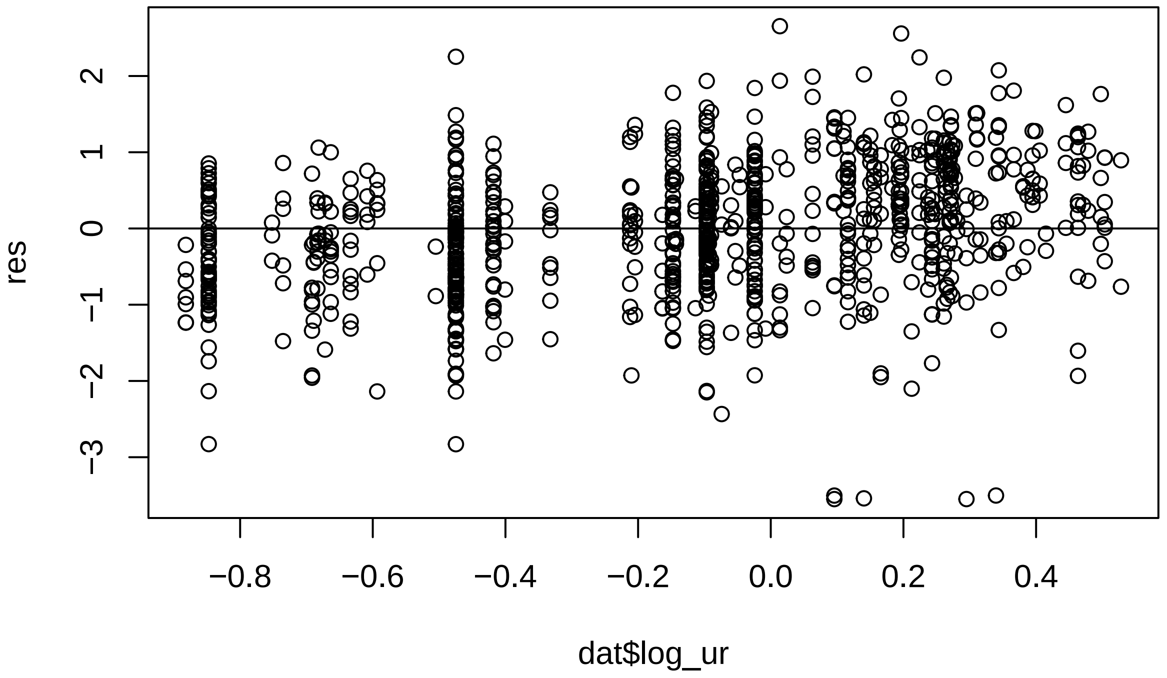
Residuals

Obtain point estimates from the replicated data

```
ytildehat_i <- apply(ynew_si, 2, mean)
res <- dat$y - ytildehat_i
```

and make some old-school plots (that I am sure you can improve upon :))

```
plot(res ~ dat$log_ur)
abline(h=0)
```



brm-fit specific function

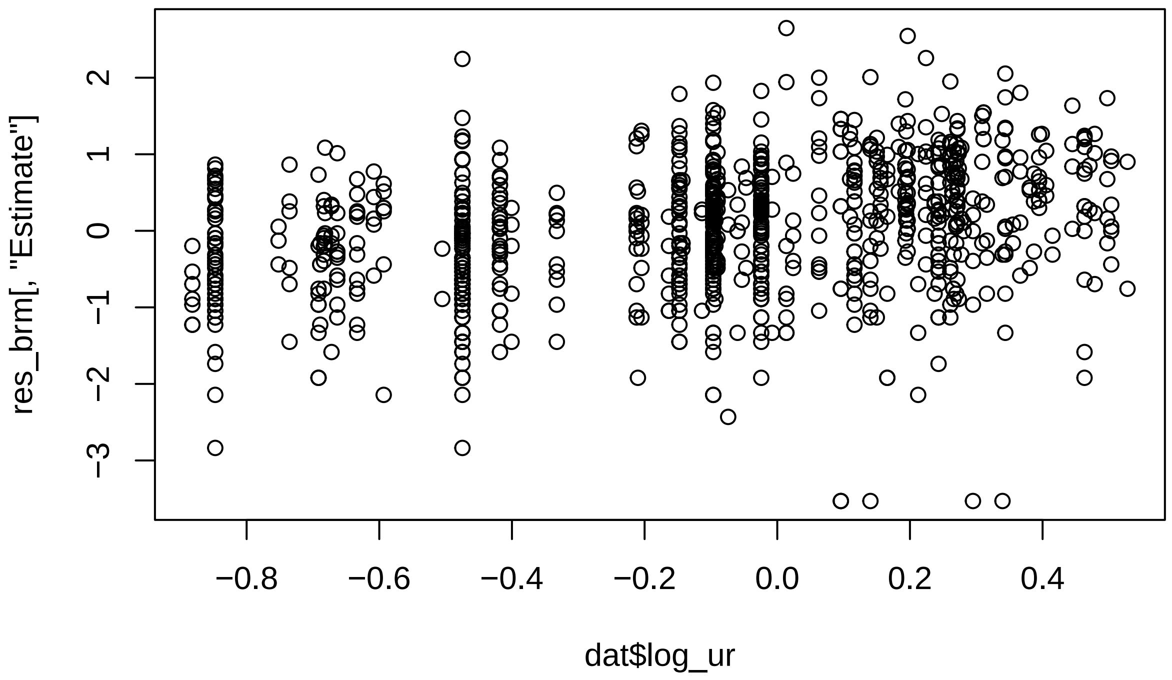
The residuals function, when applied to a brm-fit-object, returns the average residuals, as well as their SD and percentiles.

```
#?residuals.brmsfit # if you want to see the help function
res_brm <- residuals(fit)
head(res_brm)
```

```
##         Estimate  Est.Error       Q2.5       Q97.5
## [1,] -0.43872220 0.02879028 -0.4948257 -0.38072291
## [2,] -0.43872220 0.02879028 -0.4948257 -0.38072291
## [3,] -0.16246883 0.02879028 -0.2185723 -0.10446953
## [4,] -1.22717956 0.02879028 -1.2832830 -1.16918027
## [5,] -0.09577745 0.02879028 -0.1518809 -0.03777816
## [6,] -0.31088883 0.02879028 -0.3669923 -0.25288954
```

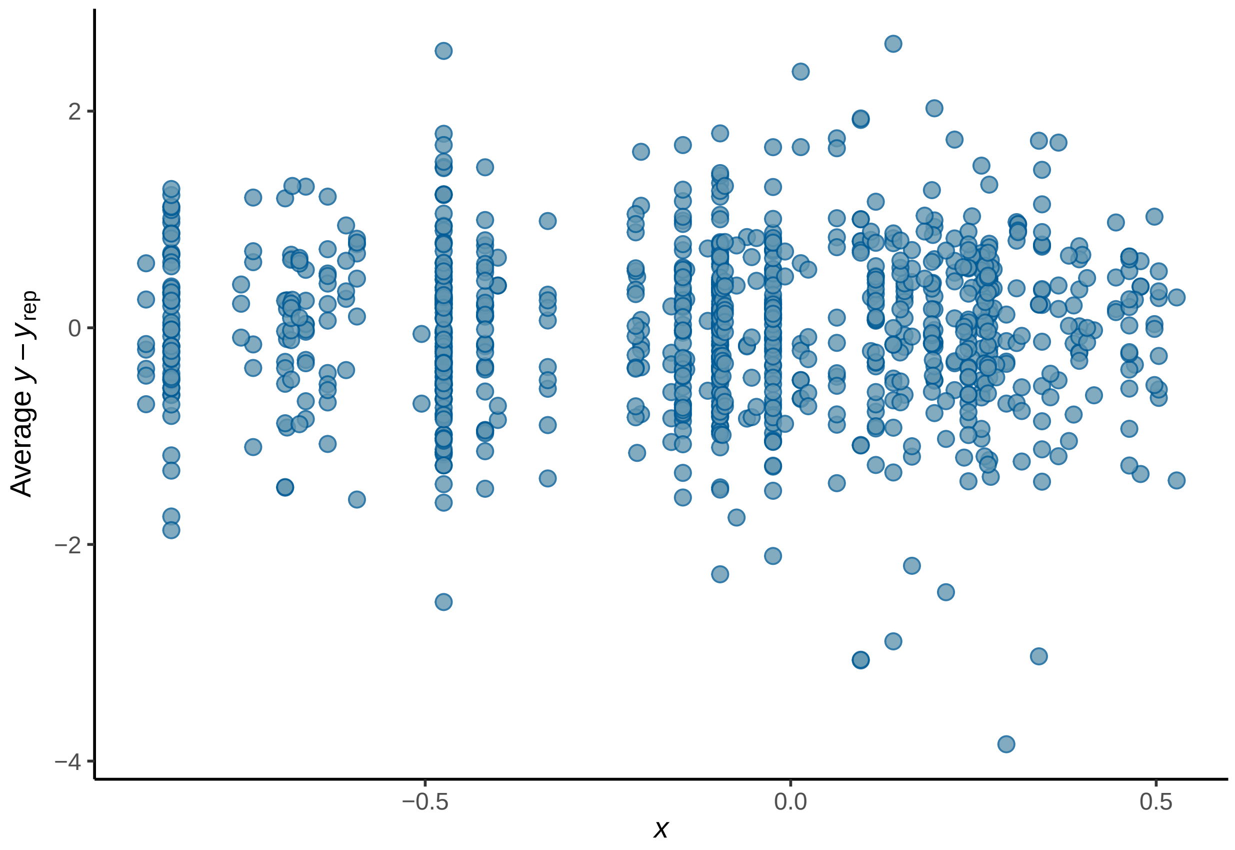
Then you can work with these residuals, e.g.

```
plot(res_brm[,"Estimate"] ~ dat$log_ur)
```



Last but not least, an automated way to do so, using functionality from the Bayesplot package:

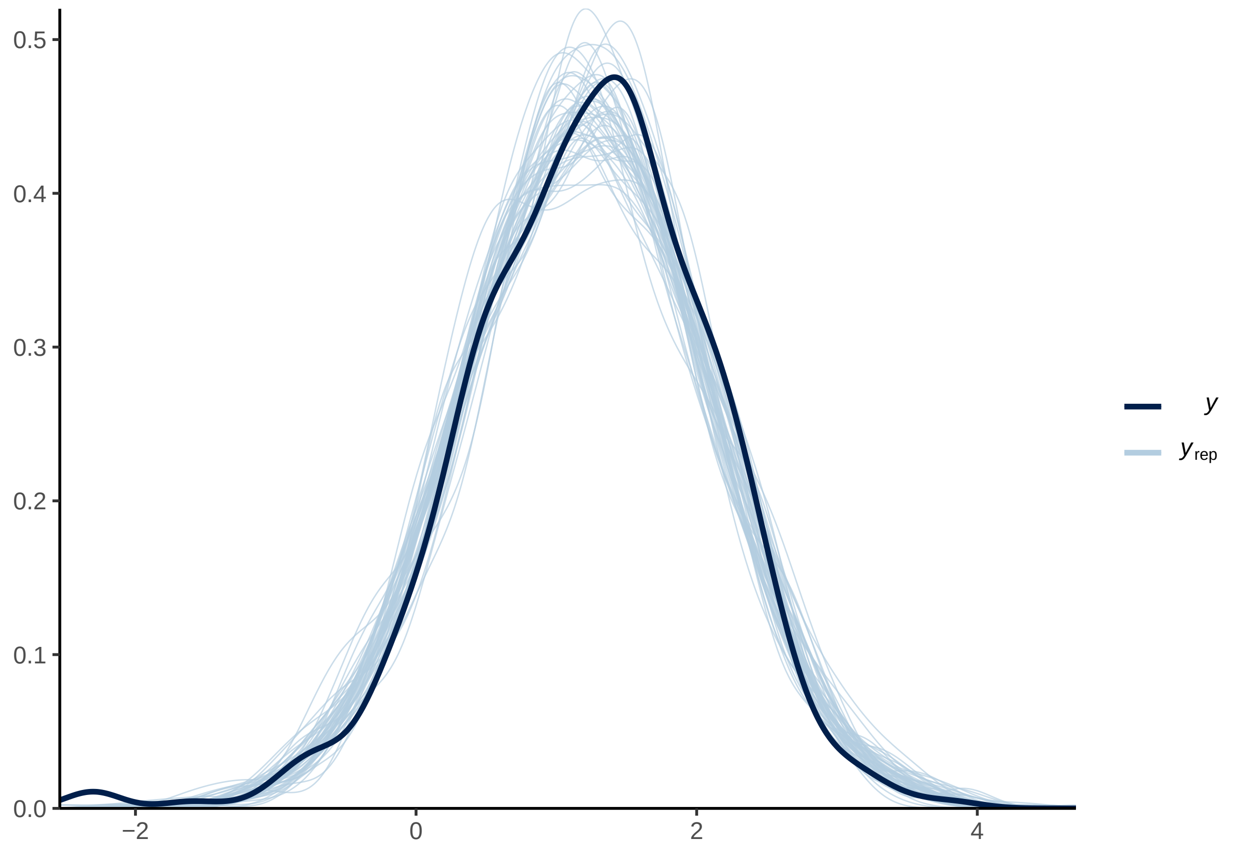
```
pp_check(fit_all, type = "error_scatter_avg_vs_x",
         x = "log_ur") +
        theme_classic()
```



Graphical checks

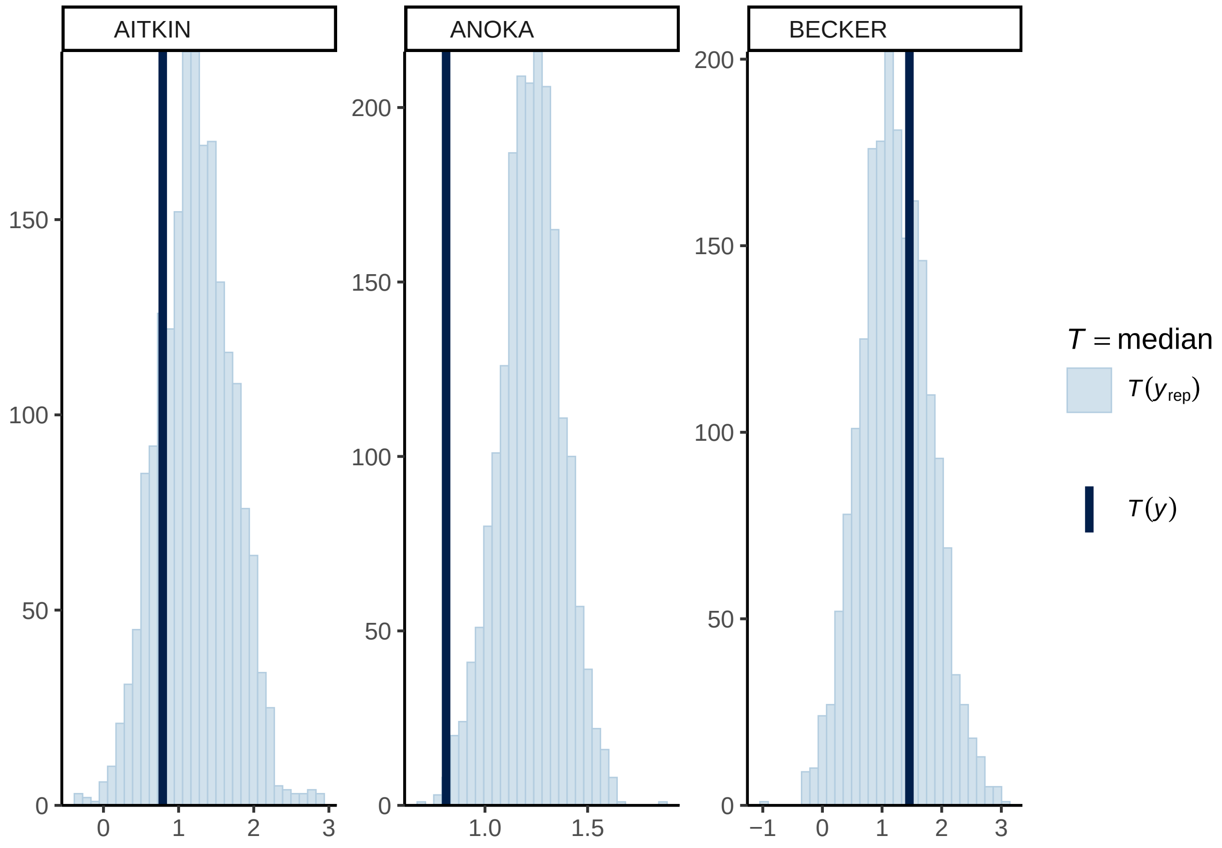
Density of observed data, and densities based on 50 replicated data sets

```
set.seed(123)
select_samp <- sample(1:dim(ynew_si)[1], 50)
ppc_dens_overlay(y = dat$y, yrep = ynew_si[select_samp, ]) +
  theme_classic()
```



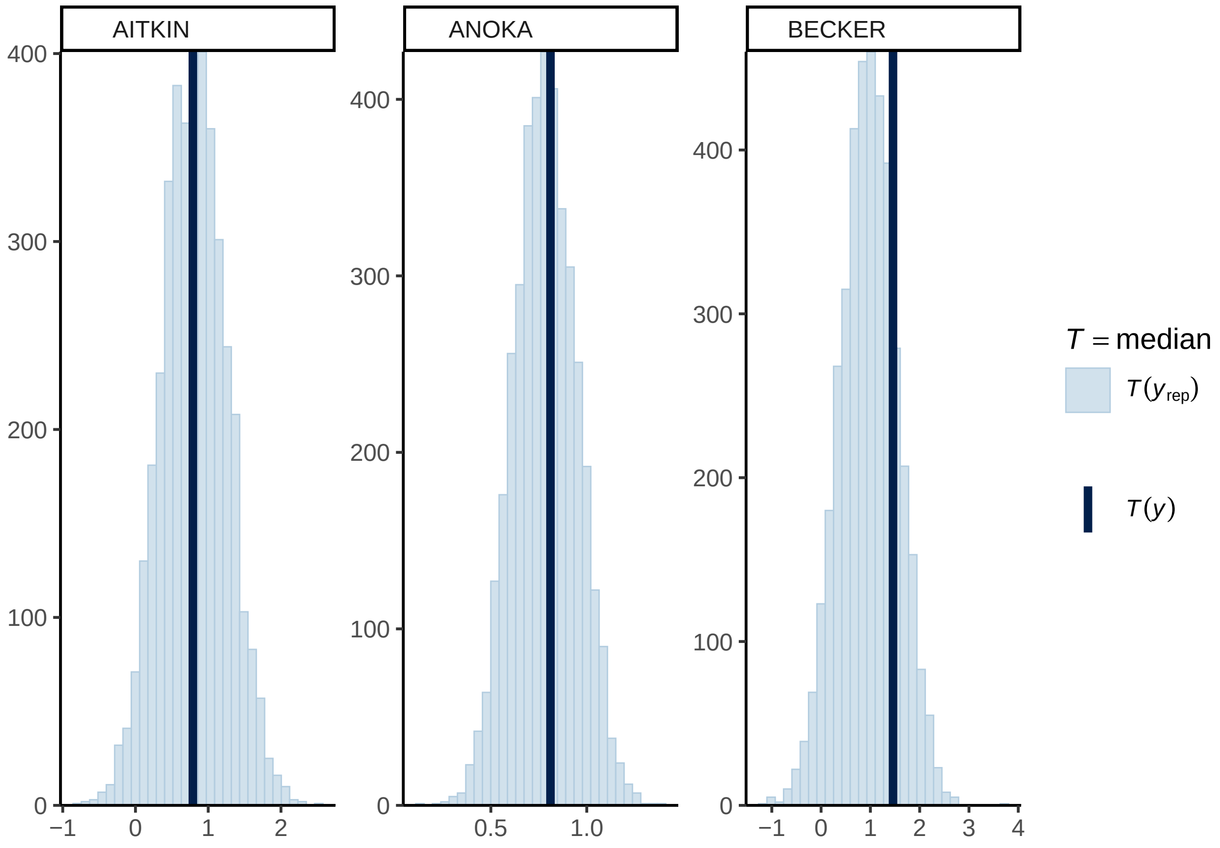
Now let's consider a test quantity, here a group-specific median, plotted for 3 counties. Results for the model w/o county intercepts show that observed median is extreme for one county (Anoka).

```
# select data for 3 counties
select_i <- which(is.element(dat$county, unique(dat$county)[seq(1,3)]))
color_scheme_set("blue")
ppc_stat_grouped(dat$y[select_i], yrep = ynew_si[, select_i],
                 group = dat$county[select_i],
                 stat = "median") + theme_classic()
```



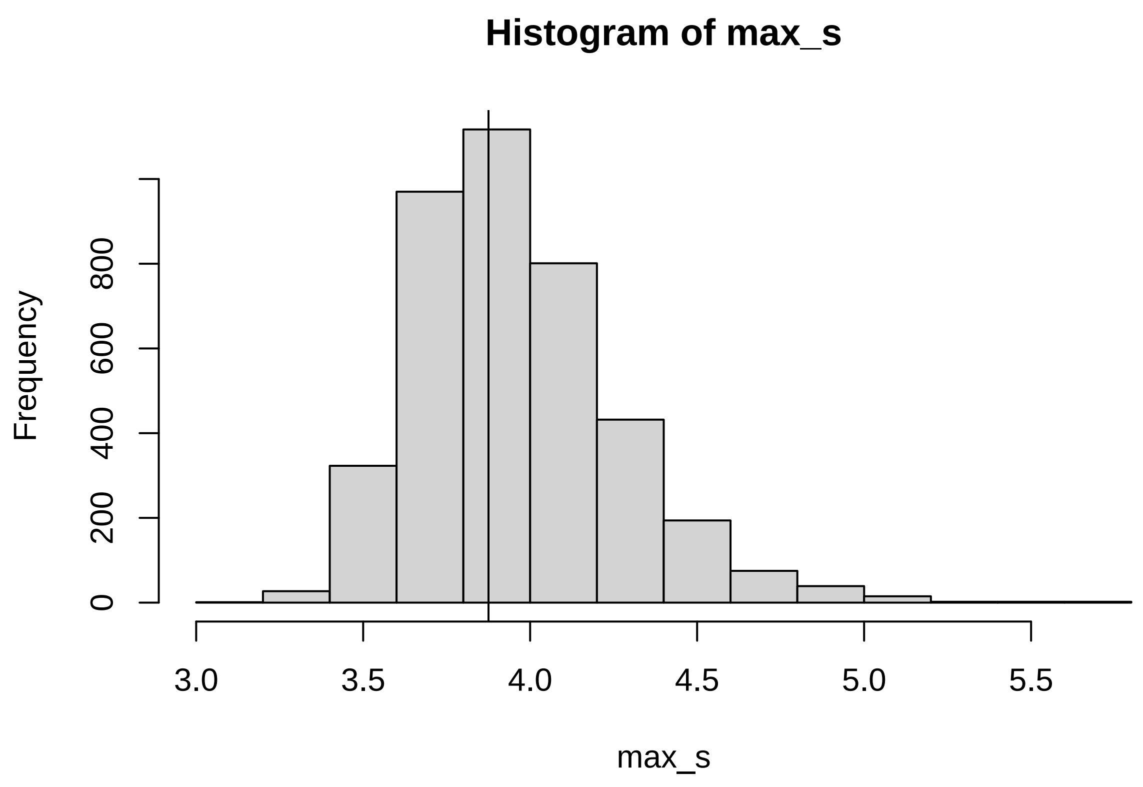
When adding the county level information, the observed median is not longer extreme:

```
ppc_stat_grouped(dat$y[select_i], yrep = ynew_all_si[, select_i],
                 group = dat$county[select_i],
                 stat = "median") + theme_classic()
```



Now let's check how extreme the observed maximum and 95th percentile of the data are, compared to these outcomes in replicated data sets.

```
max_s <- apply(ynew_all_si, 1, max) # maximum values in each of the replicated data sets
hist(max_s)
abline(v = max(dat$y))
```

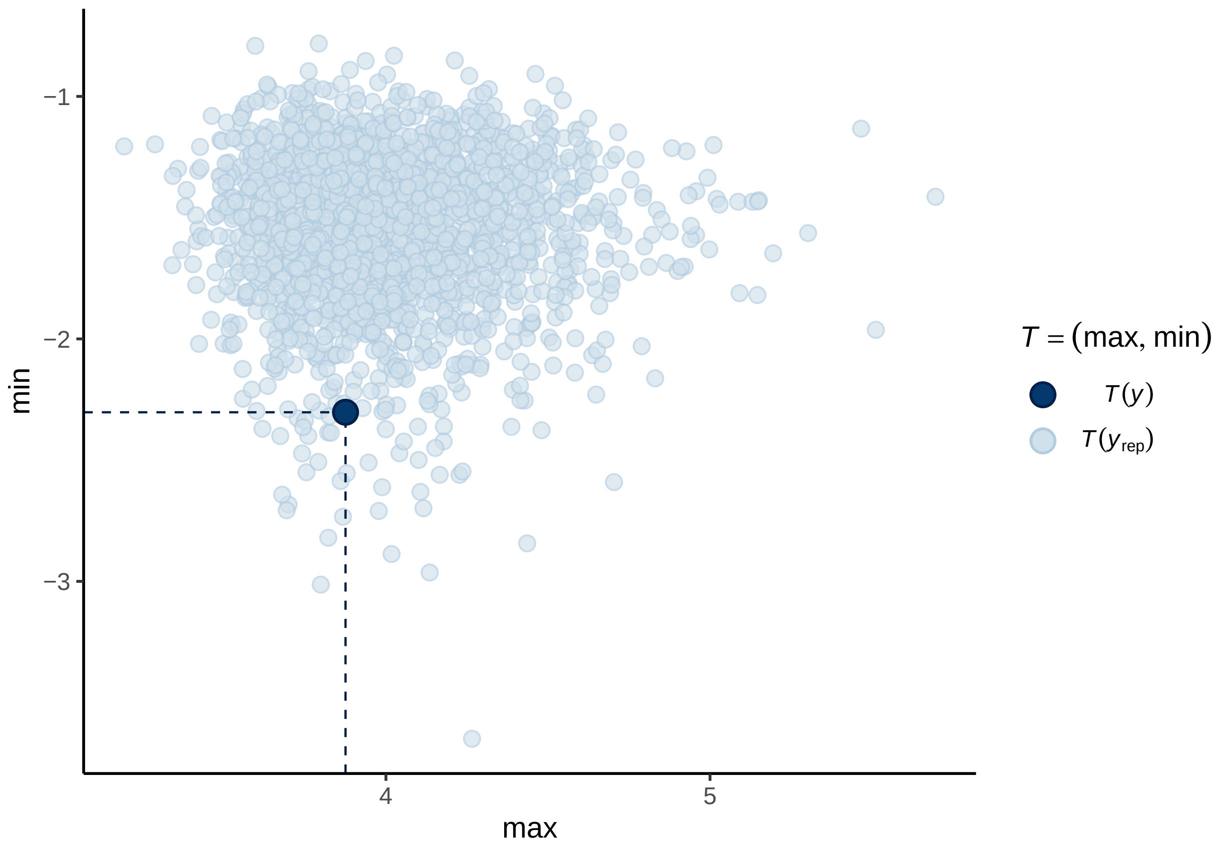


```r
mean(max_s >= max(dat$y))
```

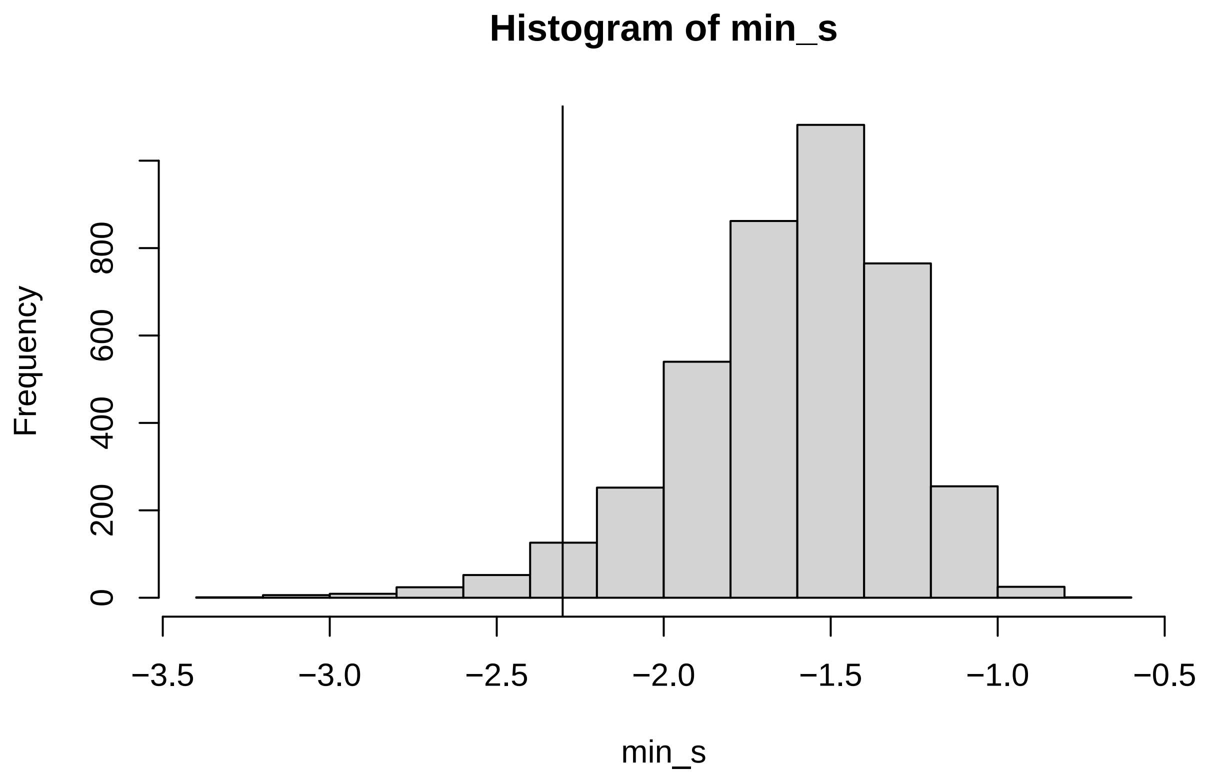
```
## [1] 0.5715
```

Using brm-fit-object, adding minimum as well: we see that the observed maximum is not extreme but the observed minimum may be.

```r
pp_check(fit, type = "stat_2d", stat = c("max", "min")) +
  theme_classic()
```



```
min_s <- apply(ynew_all_si, 1, min) # maximum values in each of the replicated data sets
hist(min_s)
abline(v = min(dat$y))
```



```
mean(min_s >= min(dat$y))
```

```
## [1] 0.96775
```

Part 2: PSIS-LOO

See
https://mc-stan.org/loo/index.html
https://mc-stan.org/bayesplot/reference/PPC-loo.html

```
library(loo)
```

get psis-loo results for our models:

```
loo_res <- loo(fit, save_psis = TRUE)
loo_county_res <- loo(fit_county, save_psis = TRUE)
loo_all_res <- loo(fit_all, save_psis = TRUE)
```

Check summaries, focus on note on Pareto k diagnostics:

```
loo_res
```

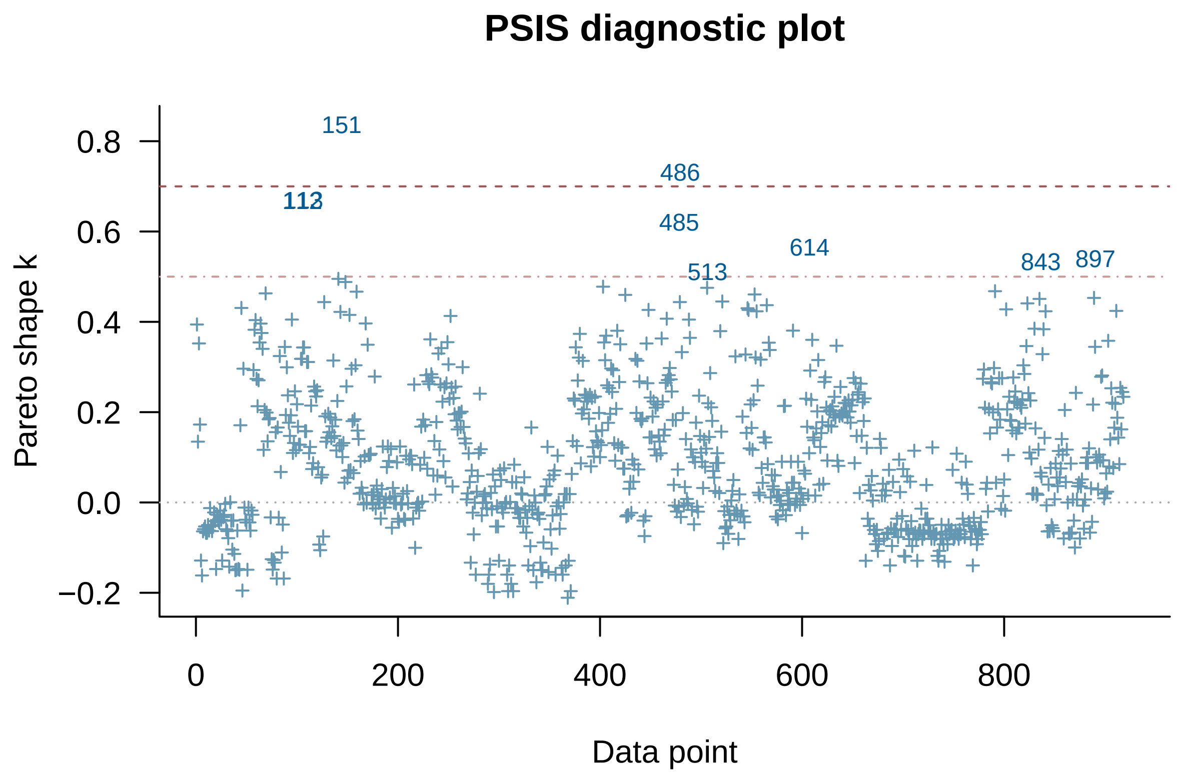
```
##
## Computed from 2000 by 927 log-likelihood matrix
##
```

```
##          Estimate   SE
## elpd_loo  -1181.2 26.8
## p_loo         2.7  0.4
## looic      2362.3 53.5
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
loo_all_res
```

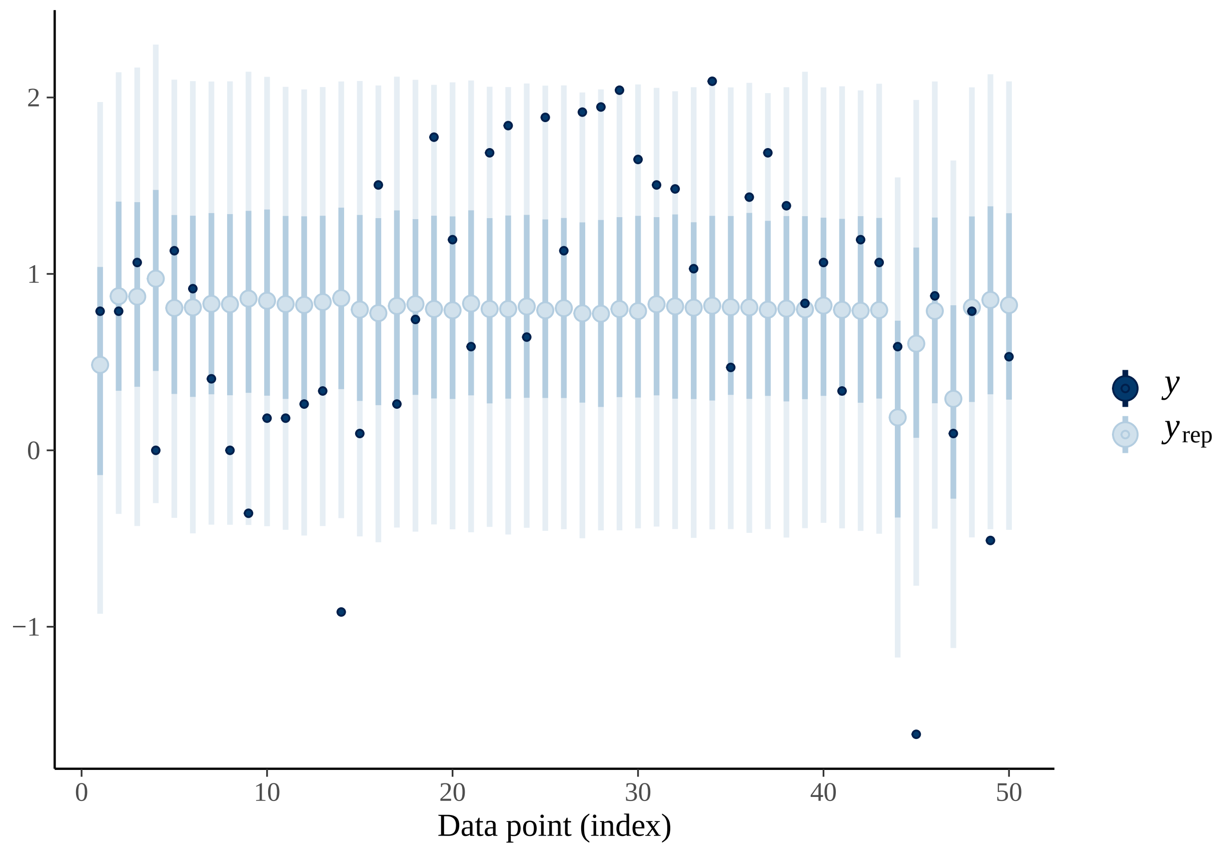
```
##
## Computed from 4000 by 927 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -1088.8 30.9
## p_loo        48.9  5.7
## looic      2177.5 61.7
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     918   99.0%   443
##  (0.5, 0.7]   (ok)         7    0.8%   190
##    (0.7, 1]   (bad)        2    0.2%   65
##    (1, Inf)   (very bad)   0    0.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

```
plot(loo_all_res,
  diagnostic = c("k"),
  label_points = TRUE,
  main = "PSIS diagnostic plot"
)
```



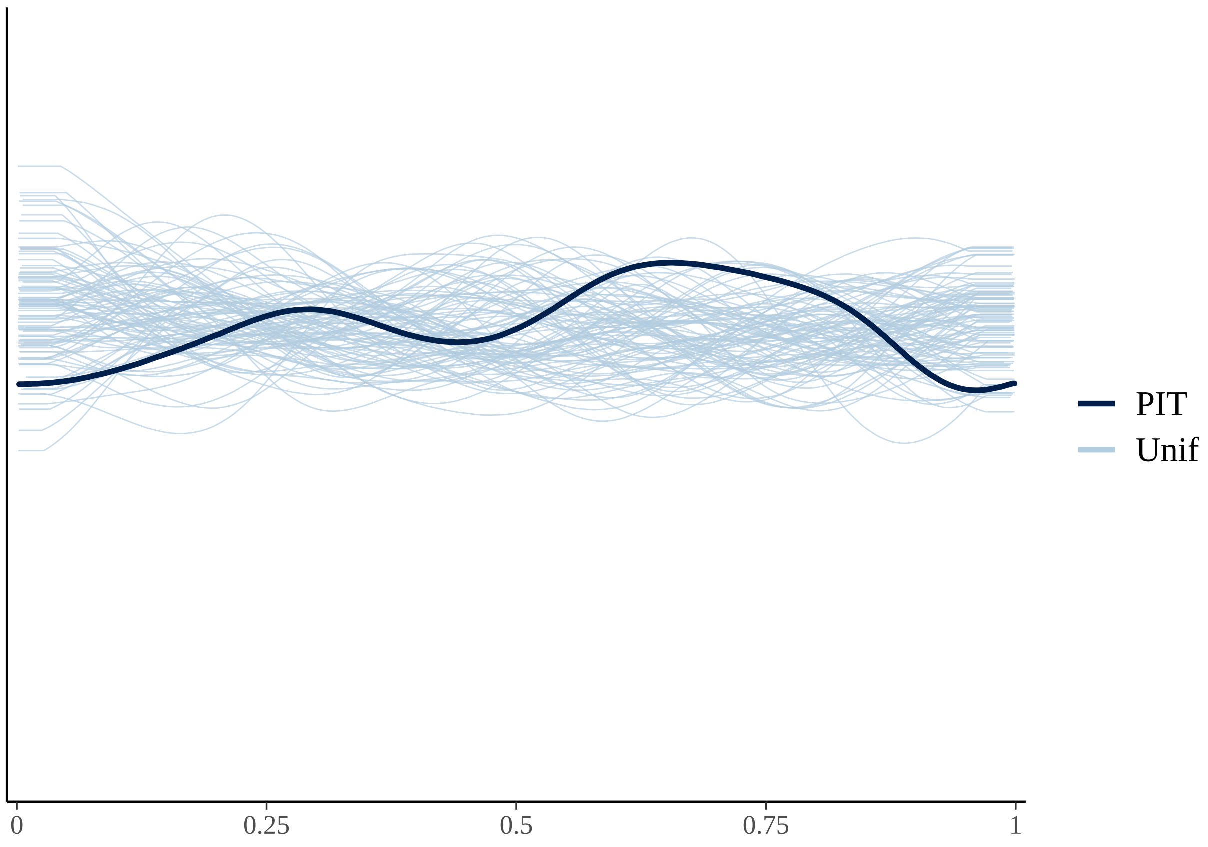
Using Bayesplot to visualize some checks

```
yrep <- posterior_predict(fit_all)
keep_obs <- 1:50
psis_all <- loo_all_res$psis_object
ppc_loo_intervals(y = dat$y, yrep, psis_object = psis_all, subset = keep_obs)
```



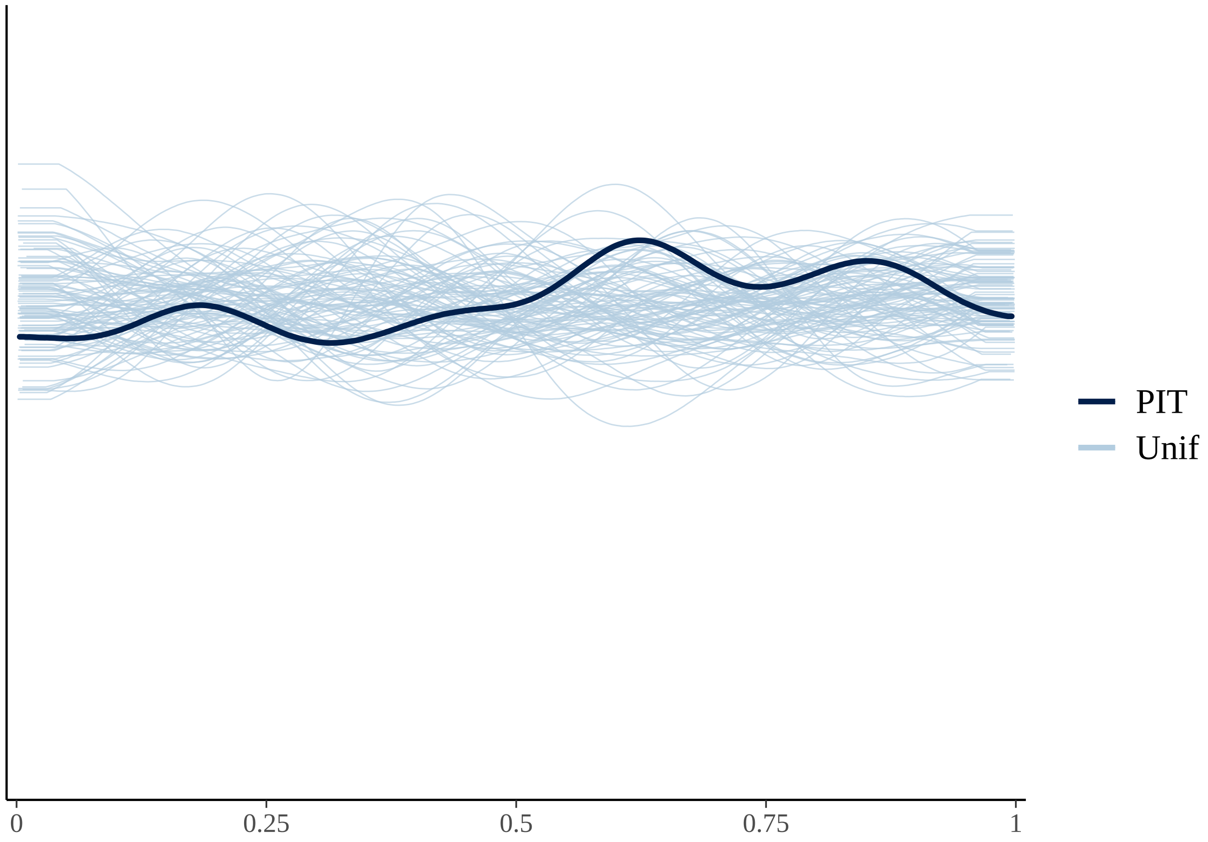
PIT

```r
lw_all <- weights(psis_all)
ppc_loo_pit_overlay(y = dat$y, yrep, lw = lw_all)
```



For simpler model

```
yrep <- posterior_predict(fit)
psis <- loo_res$psis_object
lw <- weights(psis)
ppc_loo_pit_overlay(y = dat$y, yrep, lw = lw)
```



Illustrating generated quantities in rstan

```
library(rstan)
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
```

```
dat <- dat %>%
  mutate(county_id = as.numeric(as_factor(county)))
stan_dat2 <- list(y = dat$y, N = length(dat$y), x = dat$floor, county_id = dat$county_id,
                  J = max(dat$county_id))
```

```
fit2 <- stan(file = 'module11_hier_regression.stan', seed = 123, data = stan_dat2)
```

```
y_new_list <- extract(fit2, pars = "y_new")
y_new2_si <- y_new_list$y_new
dim(y_new2_si)
```

```
## [1] 4000  927
```

```
log_lik_list <- extract(fit2, pars = "log_lik")
log_lik_si <- log_lik_list$log_lik
dim(log_lik_si)
```

```
## [1] 4000  927
```