Homework 4

Question 7E1

State the three motivating criteria that define information entropy. Try to express each in your own words

1 - information entropy should be countinous

2 - the mesure of uncertinity should increase as the number of the possible events increases

3 - uncertinity should be additive

Question 7E2

Suppose a coin is weighted such that, when it is tossed and lands on a table, it comes up heads 70% of the time. What is the entropy of this coin?

$H(p) = -Elog(p_i)$ $H(p) = -0.3 \times log(0.3) + 0.7 \times log(0.7)$

```
- 0.3 * log(0.3) + 0.7 * log(0.7)
```

```
## [1] 0.1115194
```

Question 7E3

Suppose a four-sided die is loaded such that, when tossed onto a table, it shows "1" 20%, "2" 25%, "3" 25%, and "4" 30% of the time. What is the entropy of this die?

$H(p) = -Elog(p_i)$ $H(p) = -0.2 \times log(0.2) + 0.25 \times log(0.25) + 0.25 \times log(0.25) + 0.30 \times log(0.30)$

```
- (0.2 * log(0.2) + 0.25 * log(0.25) + 0.25 * log(0.25) + 0.30 * log(0.30))
```

```
## [1] 1.376227
```

Question 7E4

Suppose another four-sided die is loaded such that it never shows "4". The other three sides show equally often. What is the entropy of this die?

$H(p) = -Elog(p_i)$ $H(p) = -1/3 \times log(1/3) + 1/3 \times log(1/3) + 1/3 \times log(1/3) + 1/3 \times log(1/3)$

```
- (1/3 * log(1/3) + 1/3 * log(1/3) + 1/3 * log(1/3))
```

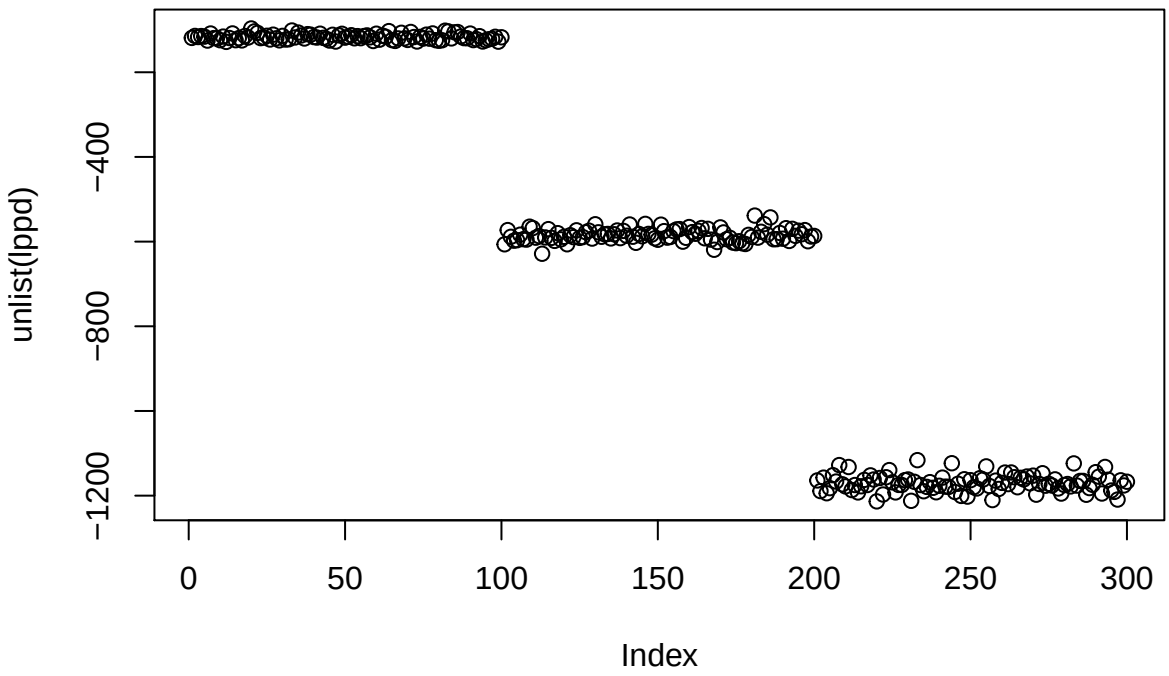
```
## [1] 1.098612
```

Question 7M3

When comparing models with an information criterion, why must all models be fit to exactly the same observations? What would happen to the information criterion values, if the models were fit to different numbers of observations? Perform some experiments, if you are not sure.

```
set.seed(2020)


number_of_observations <- rep(c(100, 500, 1000), each = 100)
```

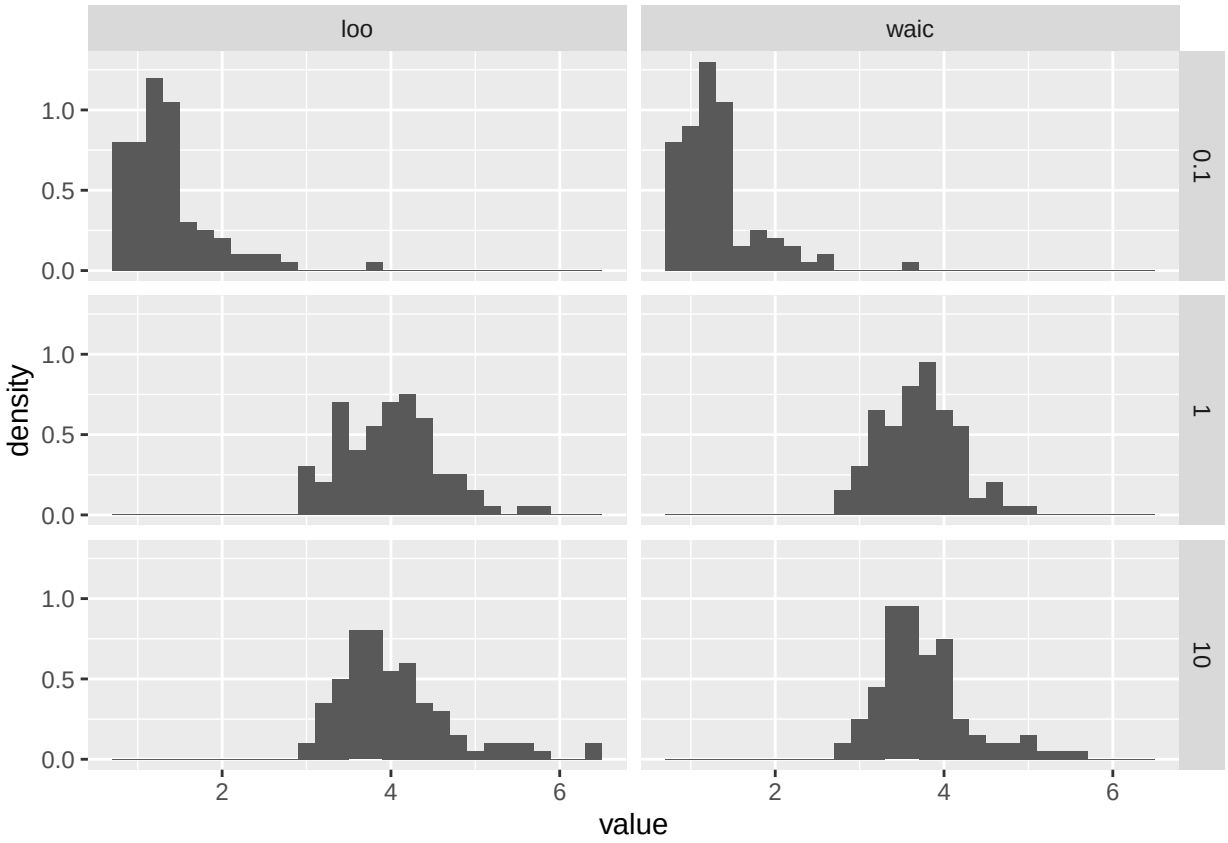
```r
sample_data <- purrr::map(number_of_observations, function (n) {
  tibble(x1 = rnorm(n=n)) %>%
    mutate(y = rnorm(n=n, mean=0.8*x1)) %>%
    mutate(across(everything(), standardize))
})


models <- purrr::map(sample_data,
  function(df) {
    mod <- quap(alist(y ~ dnorm(mu, sigma),
                      mu <- alpha + beta * x1,
                      alpha ~ dnorm(0, 0.2),
                      beta ~ dnorm(0, 0.5),
                      sigma ~ dexp(1)),
                data = df, start = list(alpha = 0, beta = 0))
    return(mod)
  })

lppd <- purrr::map(models, function(x) { sum(rethinking::lppd(x))  })

plot(unlist(lppd))
```



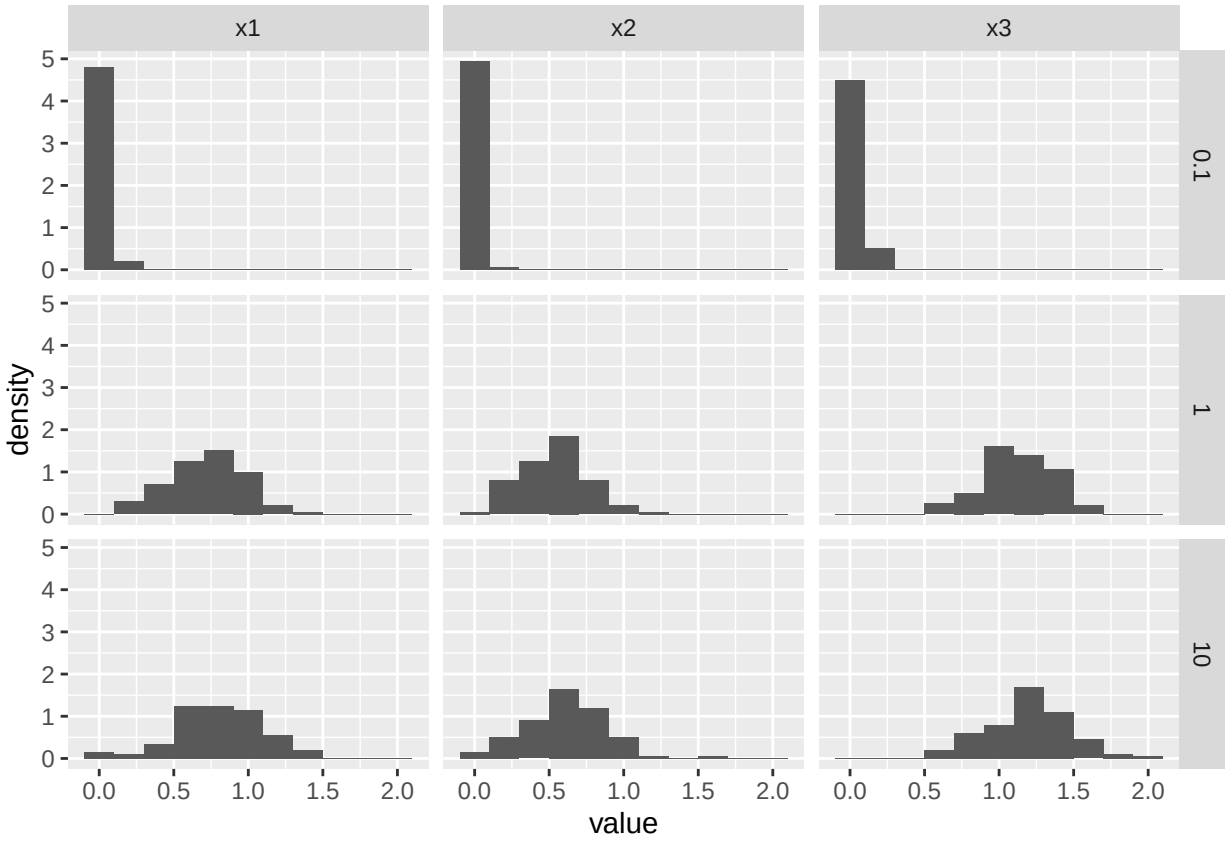
Question 7M4

What happens to the effective number of parameters, as measured by PSIS or WAIC, as a prior becomes more concentrated? Why? Perform some experiments, if you are not sure.

```
prior_factor <- rep(as.factor(priors), each=100)

df <- rbind(data.frame(prior=prior_factor, type='waic', value=waic_values %>% unlist),
            data.frame(prior=prior_factor, type='loo', value=loo_values %>% unlist))

ggplot(df, aes(x=value)) +
  facet_grid(rows = vars(prior), cols = vars(type)) + geom_histogram(aes(y = stat(density)), binwidth =
```




```
prior_factor <- rep(as.factor(priors), each=100)

df <- rbind(data.frame(prior=prior_factor, type='x1', value=x1),
            data.frame(prior=prior_factor, type='x2', value=x2),
            data.frame(prior=prior_factor, type='x3', value=x3))

ggplot(df, aes(x=value)) +
  facet_grid(rows = vars(prior), cols = vars(type)) + geom_histogram(aes(y = stat(density)), binwidth =
```



Question 7H1

```
data(Laffer)
laf_dat <- Laffer %>%
  mutate(tax_rate2 = tax_rate ^ 2,
         across(everything(), standardize))

laf_dat
```

```
##       tax_rate tax_revenue   tax_rate2
## 1  -3.00586397 -1.85313737 -2.03480375
## 2  -2.00741527 -0.47135219 -1.82970597
## 3  -1.54703217  0.15072640 -1.59913772
## 4  -1.15861963 -0.61448532 -1.33785549
## 5  -0.82275703 -0.46584707 -1.06266561
## 6  -0.81019074 -0.74660820 -1.05148279
## 7  -0.90615148 -1.13196662 -1.13525843
## 8   0.03974729 -0.94479253 -0.14667546
## 9  -0.45833467  0.03511887 -0.71239938
## 10 -0.11676012 -0.24013714 -0.33526235
## 11 -0.36351632  1.36735796 -0.61244688
## 12  0.22138727  3.69602382  0.08463272
## 13  0.44758045  1.32882212  0.39135659
## 14  0.22138727  0.83886642  0.08463272
## 15  0.12314174  0.10118031 -0.04213641
## 16  0.23052639 -0.07498353  0.09662393
## 17  0.32648713  0.17274688  0.22457379
```

```
## 18  0.45786196  0.27183904  0.40579090
## 19  0.78458545  1.20770948  0.88677727
## 20  0.45329240 -0.07498353  0.39937037
## 21  0.46243152 -0.22362178  0.41221990
## 22  0.90453638  0.15072640  1.07421249
## 23  1.02106014  0.04612911  1.26187207
## 24  0.96165587 -0.13003474  1.16551540
## 25  0.79029740 -0.30069346  0.89557062
## 26  0.78344306 -0.58695971  0.88502018
## 27  0.90339399 -0.59796996  1.07239991
## 28  1.03362643 -0.61999044  1.28243835
## 29  1.03019926 -0.28968322  1.27682303
```

```
laf_line <- brm(tax_revenue ~ 1 + tax_rate, data = laf_dat, family = gaussian,
                prior = c(prior(normal(0, 0.2), class = Intercept),
                          prior(normal(0, 0.5), class = b),
                          prior(exponential(1), class = sigma))
)
```

```
## Compiling Stan program...
```

```
## Start sampling
```

```
##
## SAMPLING FOR MODEL '7576e36c4c68af9defb1e24629c9e7ce' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.029 seconds (Warm-up)
## Chain 1:                0.029 seconds (Sampling)
## Chain 1:                0.058 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '7576e36c4c68af9defb1e24629c9e7ce' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.023 seconds (Warm-up)
## Chain 2:                0.018 seconds (Sampling)
## Chain 2:                0.041 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '7576e36c4c68af9defb1e24629c9e7ce' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.033 seconds (Warm-up)
## Chain 3:                0.032 seconds (Sampling)
## Chain 3:                0.065 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '7576e36c4c68af9defb1e24629c9e7ce' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.027 seconds (Warm-up)
## Chain 4:                0.02 seconds (Sampling)
## Chain 4:                0.047 seconds (Total)
## Chain 4:
```

```
laf_poly <- brm(tax_revenue ~ 1 + tax_rate + tax_rate2, data = laf_dat, family = gaussian,
                prior = c(prior(normal(0, 0.2), class = Intercept),
                          prior(normal(0, 0.5), class = b),
                          prior(exponential(1), class = sigma))
)
```

```
## Compiling Stan program...
```

```
## recompiling to avoid crashing R session
```

```
## Start sampling
```

```
##
## SAMPLING FOR MODEL '7576e36c4c68af9defb1e24629c9e7ce' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.056 seconds (Warm-up)
## Chain 1:                0.065 seconds (Sampling)
## Chain 1:                0.121 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '7576e36c4c68af9defb1e24629c9e7ce' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.053 seconds (Warm-up)
## Chain 2:                0.052 seconds (Sampling)
## Chain 2:                0.105 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '7576e36c4c68af9defb1e24629c9e7ce' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.046 seconds (Warm-up)
## Chain 3:                0.045 seconds (Sampling)
## Chain 3:                0.091 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '7576e36c4c68af9defb1e24629c9e7ce' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.001 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 10 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: 0.047 seconds (Warm-up)
## Chain 4:                0.06 seconds (Sampling)
## Chain 4:                0.107 seconds (Total)
## Chain 4:
```

```
laf_spln <- brm(tax_revenue ~ 1 + s(tax_rate, bs = "bs"), data = laf_dat,
                family = gaussian,
                prior = c(prior(normal(0, 0.2), class = Intercept),
                          prior(normal(0, 0.5), class = b),
                          prior(normal(0, 0.5), class = sds),
                          prior(exponential(1), class = sigma))
)
```

```
## Compiling Stan program...
## Start sampling
```

```
##
## SAMPLING FOR MODEL '81c0220dcd5f3fab08b55b129916f946' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.181 seconds (Warm-up)
## Chain 1:                0.229 seconds (Sampling)
## Chain 1:                0.41 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '81c0220dcd5f3fab08b55b129916f946' NOW (CHAIN 2).
```

```
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.221 seconds (Warm-up)
## Chain 2:                0.204 seconds (Sampling)
## Chain 2:                0.425 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '81c0220dcd5f3fab08b55b129916f946' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.231 seconds (Warm-up)
## Chain 3:                0.154 seconds (Sampling)
## Chain 3:                0.385 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '81c0220dcd5f3fab08b55b129916f946' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.238 seconds (Warm-up)
## Chain 4:                0.147 seconds (Sampling)
## Chain 4:                0.385 seconds (Total)
## Chain 4:
```

```
## Warning: There were 3 divergent transitions after warmup. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

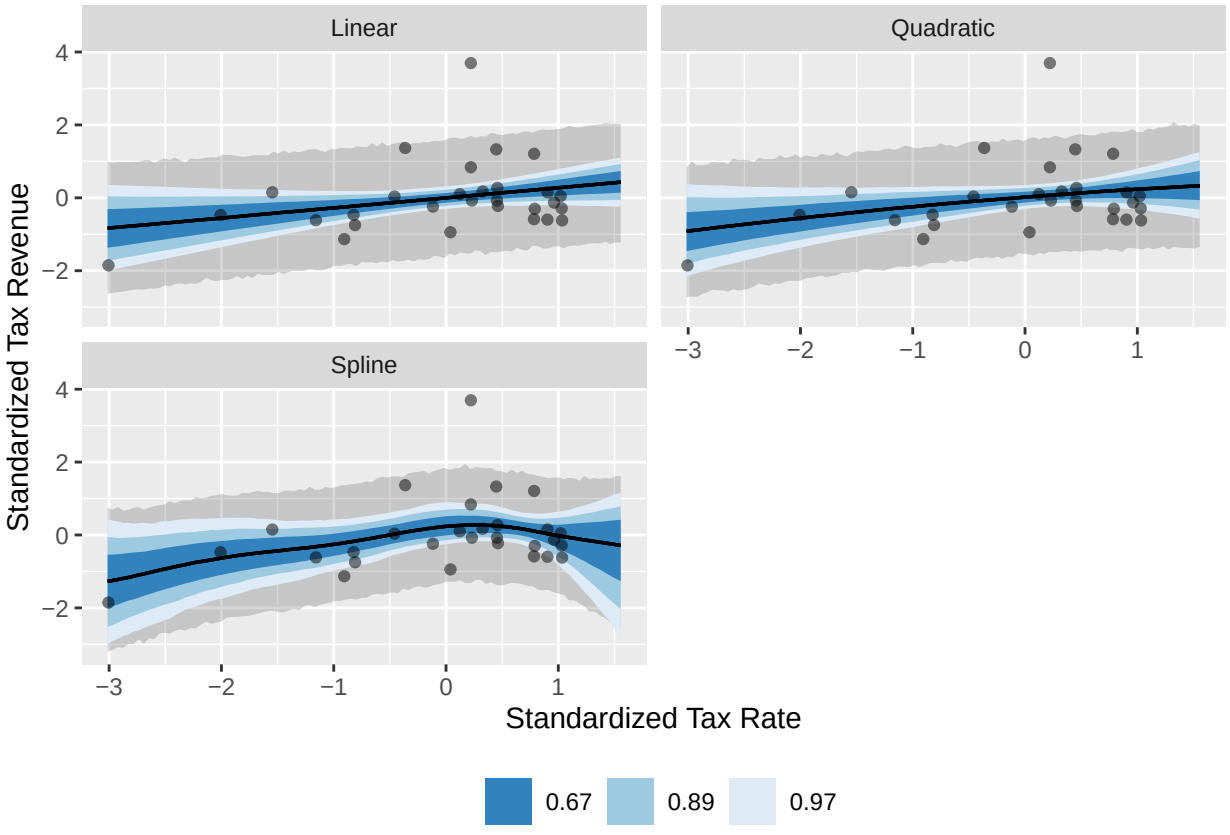
```
library(tidybayes)
```

```
##
## Attaching package: 'tidybayes'
```

```
## The following objects are masked from 'package:brms':
##
##     dstudent_t, pstudent_t, qstudent_t, rstudent_t
```

```
tr_seq <- tibble(tax_rate = seq(0, 40, length.out = 100)) %>%
  mutate(tax_rate2 = tax_rate ^ 2,
         tax_rate = (tax_rate - mean(Laffer$tax_rate)) / sd(Laffer$tax_rate),
         tax_rate2 = (tax_rate2 - mean(Laffer$tax_rate ^ 2)) /
           sd(Laffer$tax_rate ^ 2))

predictions <- bind_rows(
  predicted_draws(laf_line, newdata = tr_seq) %>%
    median_qi(.width = 0.89) %>%
    mutate(type = "Linear"),
  predicted_draws(laf_poly, newdata = tr_seq) %>%
    median_qi(.width = 0.89) %>%
    mutate(type = "Quadratic"),
  predicted_draws(laf_spln, newdata = tr_seq) %>%
    median_qi(.width = 0.89) %>%
    mutate(type = "Spline")
)


fits <- bind_rows(
  fitted_draws(laf_line, newdata = tr_seq) %>%
    median_qi(.width = c(0.67, 0.89, 0.97)) %>%
```

```
    mutate(type = "Linear"),
  fitted_draws(laf_poly, newdata = tr_seq) %>%
    median_qi(.width = c(0.67, 0.89, 0.97)) %>%
    mutate(type = "Quadratic"),
  fitted_draws(laf_spln, newdata = tr_seq) %>%
    median_qi(.width = c(0.67, 0.89, 0.97)) %>%
    mutate(type = "Spline")
)

ggplot() +
  facet_wrap(~type, ncol = 2) +
  geom_ribbon(data = predictions,
              aes(x = tax_rate, ymin = .lower, ymax = .upper),
              alpha = 0.2) +
  geom_lineribbon(data = fits,
                  aes(x = tax_rate, y = .value, ymin = .lower, ymax = .upper),
                  size = 0.6) +
  geom_point(data = laf_dat, aes(x = tax_rate, y = tax_revenue),
             alpha = 0.5) +
  scale_fill_brewer(palette = "Blues", breaks = c(0.67, 0.89, 0.97)) +
  labs(x = "Standardized Tax Rate", y = "Standardized Tax Revenue") +
  theme(legend.position = "bottom")
```




```
loo_compare(loo(laf_line), loo(laf_poly), loo(laf_spln))
```

```
## Warning: Found 1 observations with a pareto_k > 0.7 in model 'laf_line'. It is
```

```
## recommended to set 'moment_match = TRUE' in order to perform moment matching for
## problematic observations.
```

```
## Warning: Found 1 observations with a pareto_k > 0.7 in model 'laf_poly'. It is
## recommended to set 'moment_match = TRUE' in order to perform moment matching for
## problematic observations.
```

```
## Warning: Found 1 observations with a pareto_k > 0.7 in model 'laf_spln'. It is
## recommended to set 'moment_match = TRUE' in order to perform moment matching for
## problematic observations.
```

```
##          elpd_diff se_diff
## laf_poly  0.0       0.0
## laf_spln -0.1       0.7
## laf_line -0.7       0.6
```

Question 7H2

```
which.max(loo(laf_line)$pointwise[,'influence_pareto_k'])
```

```
## Warning: Found 1 observations with a pareto_k > 0.7 in model 'laf_line'. It is
## recommended to set 'moment_match = TRUE' in order to perform moment matching for
## problematic observations.
```

```
## [1] 12
```

```
laf_spln2 <- brm(bf(tax_revenue ~ 1 + s(tax_rate, bs = "bs"), nu = 1),
                 data = laf_dat, family = student,
                 prior = c(prior(normal(0, 0.2), class = Intercept),
                           prior(normal(0, 0.5), class = b),
                           prior(normal(0, 0.5), class = sds),
                           prior(exponential(1), class = sigma)))
```

```
## Compiling Stan program...
```

```
## Start sampling
```

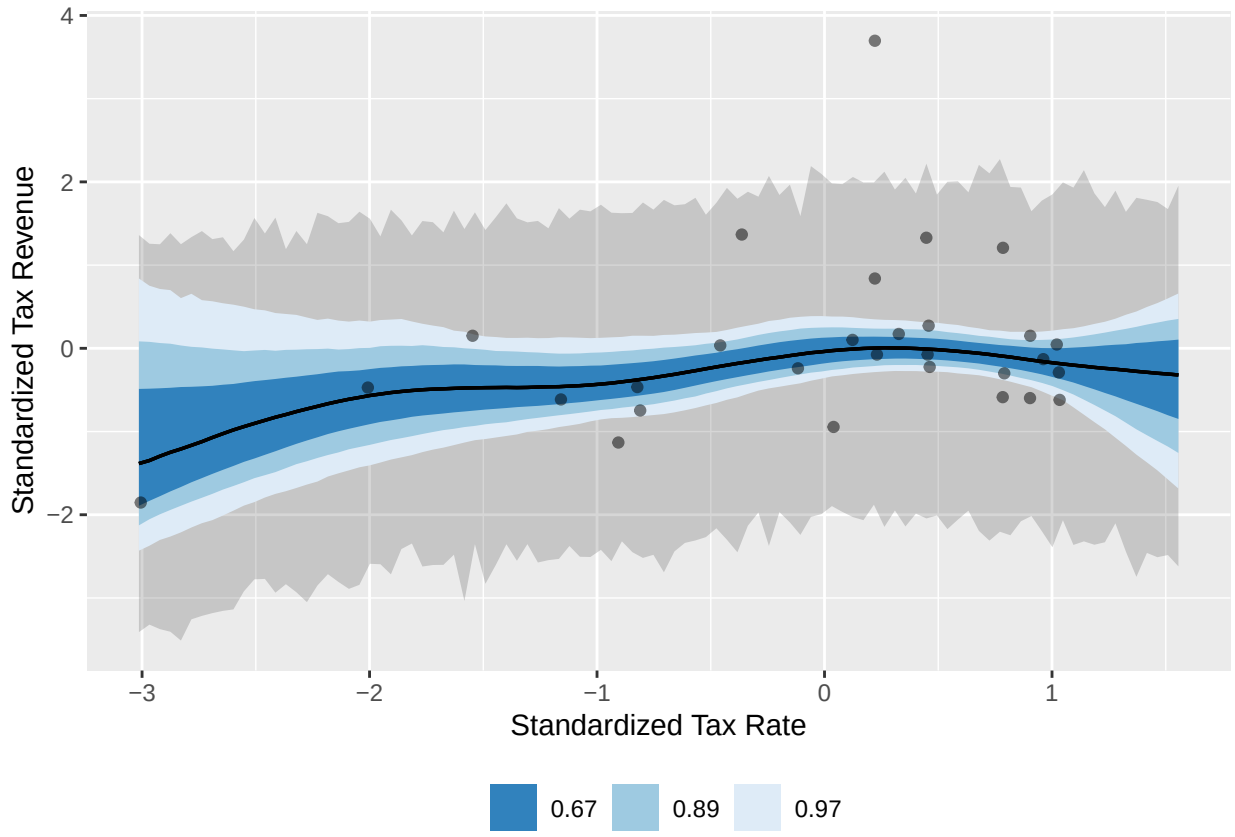
```
##
## SAMPLING FOR MODEL 'ff7f2f6d13f05a34d1691d48995773fd' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## 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: 0.343 seconds (Warm-up)
## Chain 1:                0.222 seconds (Sampling)
## Chain 1:                0.565 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'ff7f2f6d13f05a34d1691d48995773fd' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.332 seconds (Warm-up)
## Chain 2:                0.292 seconds (Sampling)
## Chain 2:                0.624 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'ff7f2f6d13f05a34d1691d48995773fd' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.308 seconds (Warm-up)
## Chain 3:                0.326 seconds (Sampling)
## Chain 3:                0.634 seconds (Total)
## Chain 3:
```

```
##
## SAMPLING FOR MODEL 'ff7f2f6d13f05a34d1691d48995773fd' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.352 seconds (Warm-up)
## Chain 4:                0.35 seconds (Sampling)
## Chain 4:                0.702 seconds (Total)
## Chain 4:
```

```
tr_seq <- tibble(tax_rate = seq(0, 40, length.out = 100)) %>%
  mutate(tax_rate2 = tax_rate ^ 2,
         tax_rate = (tax_rate - mean(Laffer$tax_rate)) / sd(Laffer$tax_rate),
         tax_rate2 = (tax_rate2 - mean(Laffer$tax_rate ^ 2)) /
           sd(Laffer$tax_rate ^ 2))

predictions <- bind_rows(
  predicted_draws(laf_spln2, newdata = tr_seq) %>%
    median_qi(.width = 0.89) %>%
    mutate(type = "Spline")
)


fits <- bind_rows(
  fitted_draws(laf_spln2, newdata = tr_seq) %>%
    median_qi(.width = c(0.67, 0.89, 0.97)) %>%
    mutate(type = "Spline")
)


ggplot() +
  geom_ribbon(data = predictions,
              aes(x = tax_rate, ymin = .lower, ymax = .upper),
              alpha = 0.2) +
  geom_lineribbon(data = fits,
                  aes(x = tax_rate, y = .value, ymin = .lower, ymax = .upper),
                  size = 0.6) +
  geom_point(data = laf_dat, aes(x = tax_rate, y = tax_revenue),
```

```
            alpha = 0.5) +
  scale_fill_brewer(palette = "Blues", breaks = c(0.67, 0.89, 0.97)) +
  labs(x = "Standardized Tax Rate", y = "Standardized Tax Revenue") +
  theme(legend.position = "bottom")
```




```
loo_compare(loo(laf_spln), loo(laf_spln2))
```

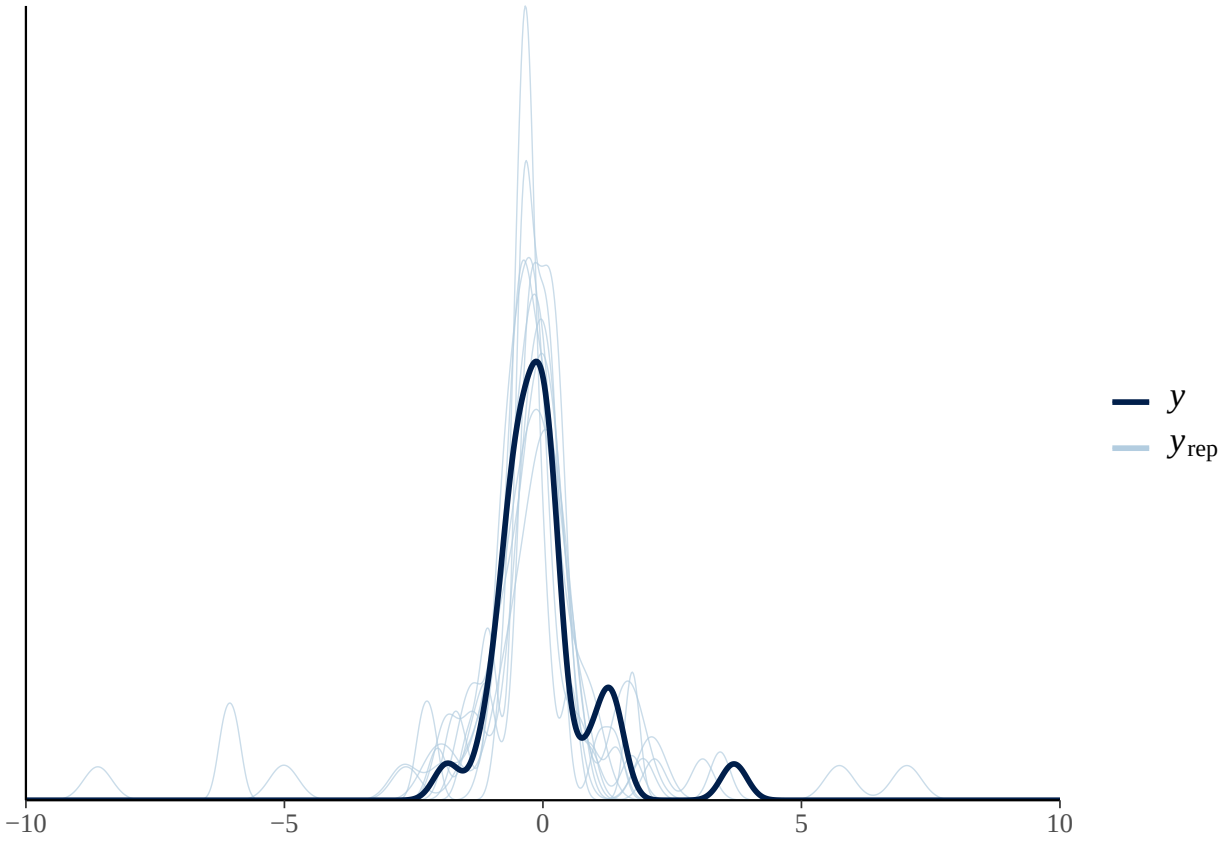
```
## Warning: Found 1 observations with a pareto_k > 0.7 in model 'laf_spln'. It is
## recommended to set 'moment_match = TRUE' in order to perform moment matching for
## problematic observations.
```

```
##           elpd_diff se_diff
## laf_spln2  0.0       0.0
## laf_spln  -6.1       6.8
```

```
pp_check(laf_spln2) + xlim(c(-10, 10))
```

```
## Using 10 posterior samples for ppc type 'dens_overlay' by default.
```

```
## Warning: Removed 8 rows containing non-finite values (stat_density).
```



Question 7H3

Consider three fictional Polynesian islands. On each there is a Royal Ornithologist charged by the king with surveying the bird population. They have each found the following proportions of 5 important bird species: Species A Species B Species C Species D Species E Island 1 0.2 0.2 0.2 0.2 0.2 Island 2 0.8 0.1 0.05 0.025 0.025 Island 3 0.05 0.15 0.7 0.05 0.05 Notice that each row sums to 1, all the birds. This problem has two parts. It is not computationally complicated. But it is conceptually tricky. First, compute the entropy of each island's bird distribution. Interpret these entropy values. Second, use each island's bird distribution to predict the other two. This means to compute the K-L Divergence of each island from the others, treating each island as if it were a statistical model of the other islands. You should end up with 6 different K-L Divergence values. Which island predicts the others best? Why?

H(p) = - E log(pi)

```
i1 <- c(0.2, 0.2, 0.2, 0.2, 0.2)
i2 <- c(0.8, 0.1, 0.05, 0.025, 0.025)
i3 <- c(0.05, 0.15, 0.7, 0.05, 0.05)

H1 <- i1 %>% sapply(FUN=function(x) {x * log(x) * -1}) %>% sum
H2 <- i2 %>% sapply(FUN=function(x) {x * log(x) * -1}) %>% sum
H3 <- i3 %>% sapply(FUN=function(x) {x * log(x) * -1}) %>% sum

KL <- function(to_island, from_island) {
    c(to_island, from_island) %>%
    matrix(byrow=TRUE, ncol=5) %>%
    apply(MARGIN=2, FUN=function(x) {x[1]*(log(x[1]) - log(x[2]))}) %>%
    sum
```

```
  }


result_matrix <- matrix(
  c(KL(i1,i1), KL(i1,i2), KL(i1,i3),
  KL(i2,i1), KL(i2,i2), KL(i2,i3),
  KL(i3,i1), KL(i3,i2), KL(i3,i3)),
  ncol = 3
)

head(result_matrix)
```

```
##           [,1]     [,2]      [,3]
## [1,] 0.0000000 0.866434 0.6258376
## [2,] 0.9704061 0.000000 1.8388452
## [3,] 0.6387604 2.010914 0.0000000
```

Question 7H4

7H4. Recall the marriage, age, and happiness collider bias example from Chapter 6. Run models m6.9 and m6.10 again (page 178). Compare these two models using WAIC (or PSIS, they will produce identical results). Which model is expected to make better predictions? Which model provides the correct causal inference about the influence of age on happiness? Can you explain why the answers to these two questions disagree?

```
loo_compare(loo(m6.9), loo(m6.10))
```

```
##       elpd_diff se_diff
## m6.9     0.0       0.0
## m6.10 -194.2      17.6
```

Question 7H5

7H5. Revisit the urban fox data, data(foxes), from the previous chapter's practice problems. Use WAIC or PSIS based model comparison on five different models, each using weight as the outcome, and containing these sets of predictor variables: (1) avgfood + groupsize + area (2) avgfood + groupsize (3) groupsize + area (4) avgfood (5) area Can you explain the relative differences in WAIC scores, using the fox DAG from last week's homework? Be sure to pay attention to the standard error of the score differences (dSE)

```
loo_compare(loo(b7h5_2), loo(b7h5_3), loo(b7h5_4), loo(b7h5_5))
```

```
##        elpd_diff se_diff
## b7h5_2  0.0       0.0
## b7h5_3  0.0       2.7
## b7h5_4 -4.8       3.1
## b7h5_5 -5.0       3.2
```