Assignment 5

Natali Tckvitishvili

2024-07-13

```
library(rethinking)
library(dplyr)
library(ggplot2)
library(tidyr)
```

Task Set 1

Load the data set `RiskyChoice.csv` to solve the Task Set 1. Use the `read_csv2()` function instead of `read_csv()`.

```
risk <- read.csv2("/Users/nataly/Management & Technology/Introduction to Bayesian Data Analysis/bayes/ba
head(risk)
```

```
##   Subject AgeGroup ItemID Position CorrectChoice RiskyChoice Gender
## 1       1  younger      1       18             0           0 female
## 2       1  younger      2       92             0           1 female
## 3       1  younger      3       73             1           1 female
## 4       1  younger      4       44             1           0 female
## 5       1  younger      5        9             0           0 female
## 6       1  younger      6       78             1          NA female
##   NegativeAffect Numeracy
## 1           1.75        9
## 2           1.75        9
## 3           1.75        9
## 4           1.75        9
## 5           1.75        9
## 6           1.75        9
```

```
summary(risk)
```

```
##     Subject        AgeGroup            ItemID      Position
##  Min.   :  1.0   Length:12810       Min.   :  1   Min.   :  1
##  1st Qu.: 31.0   Class :character   1st Qu.: 27   1st Qu.: 27
##  Median : 61.5   Mode  :character   Median : 53   Median : 53
##  Mean   : 61.5                      Mean   : 53   Mean   : 53
##  3rd Qu.: 92.0                      3rd Qu.: 79   3rd Qu.: 79
##  Max.   :122.0                      Max.   :105   Max.   :105
##
##  CorrectChoice     RiskyChoice        Gender          NegativeAffect
```

```
##  Min.   :0.0000   Min.   :0.0000   Length:12810       Length:12810
##  1st Qu.:0.0000   1st Qu.:0.0000   Class :character   Class :character
##  Median :1.0000   Median :0.0000   Mode  :character   Mode  :character
##  Mean   :0.6518   Mean   :0.4717
##  3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000
##  NA's   :122      NA's   :244
##    Numeracy
##  Min.   : 2.000
##  1st Qu.: 7.000
##  Median : 8.000
##  Mean   : 8.025
##  3rd Qu.:10.000
##  Max.   :10.000
##  NA's   :315
```

Task 1.1

Create a reduced data table with only one row per subject that shows the number of solved choices problems (`nChoice`) and the number of correct choices (`nCorrect`) for each subject along with the other variables. Remove the subjects with missing values. Print the data of the first 10 subjects.

```r
risk_cut <- risk %>%
  na.omit() %>%
  group_by(Subject) %>%
  summarise(
    nChoice = n(),
    nCorrect = sum(CorrectChoice),
    ageGroup = max(AgeGroup), # they're all the same, so we can take max
    gender = max(Gender),
    numeracy = max(Numeracy))

print(risk_cut, n = 10)
```

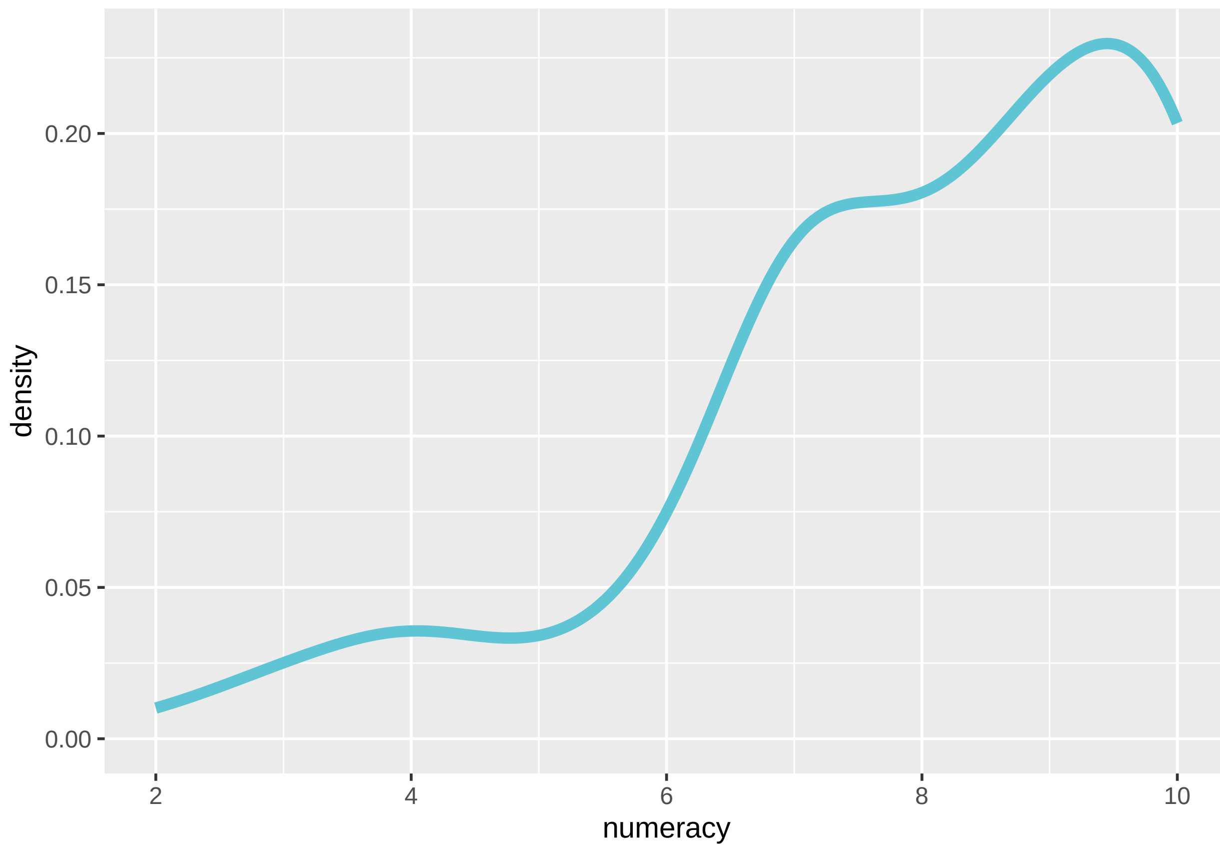
```
## # A tibble: 119 x 6
##    Subject nChoice nCorrect ageGroup gender numeracy
##      <int>   <int>    <int> <chr>    <chr>     <int>
##  1       1     102       59 younger  female        9
##  2       2     102       64 younger  female        7
##  3       3     102       78 younger  male         10
##  4       4     102       69 younger  female       10
##  5       5     102       56 younger  male          9
##  6       6     102       68 younger  female        9
##  7       7     102       73 younger  male         10
##  8       8     102       64 younger  female       10
##  9       9     102       60 younger  female        9
## 10      10     102       61 younger  female        7
## # i 109 more rows
```

Task 1.2

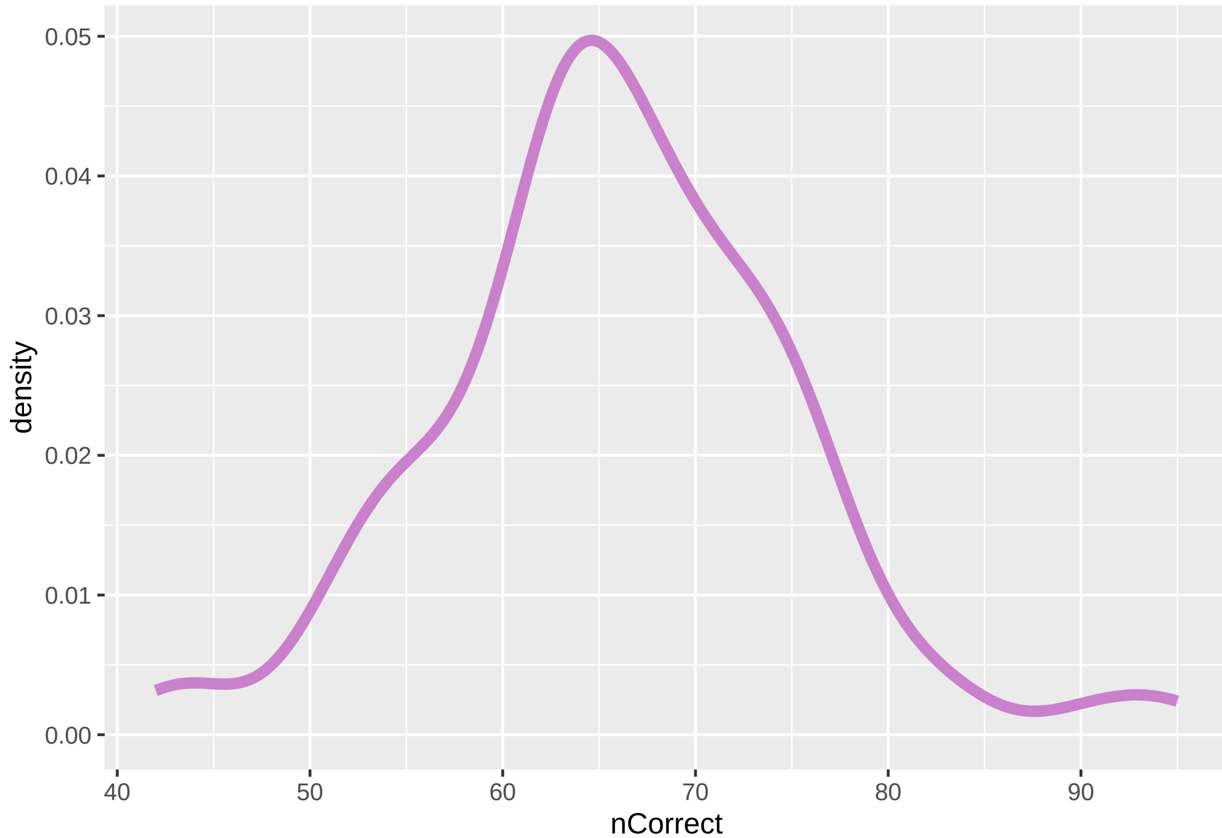
Run a Bayesian regression model that predicts `nCorrect` from `Numeracy` using fixed intercepts and fixed slopes. Standardize the predictor before running the model and compute the WAIC of the model.

```r
# make a list
risk_cut_list <- list(
  nCorrect = risk_cut$nCorrect,
  numeracy_s = scale(risk_cut$numeracy),
  subject = as.integer(as.factor(risk_cut$Subject))
)

# plotting to understand the distribution
ggplot(risk_cut, aes(numeracy)) +
  geom_density(color = "#5fc5d5", linewidth = 2)
```




```r
ggplot(risk_cut, aes(nCorrect)) +
  geom_density(color = "#ca81cb", linewidth = 2)
```




```
numeracy_model <- ulam(
  alist(
    nCorrect ~ dnorm(mu, sigma),
    mu <- a + b * numeracy_s,
    a ~ dnorm(0, 0.6),
    b ~ dnorm(0, 0.5),
    sigma ~ dnorm(0, 1)
  ), data = risk_cut_list, chains = 4, cores = 4, log_lik = TRUE
)
```

```
## Running MCMC with 4 parallel chains, with 1 thread(s) per chain...
##
## Chain 1 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2 Iteration: 501 / 1000 [ 50%]  (Sampling)
```

```
## Chain 2 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1 finished in 0.6 seconds.
## Chain 2 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2 finished in 0.7 seconds.
## Chain 3 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3 finished in 0.7 seconds.
## Chain 4 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4 finished in 0.7 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.7 seconds.
## Total execution time: 1.0 seconds.
```

```
precis(numeracy_model, depth = 2)
```

```
##             mean        sd       5.5%      94.5%     rhat ess_bulk
## a      4.2415430 0.6006702  3.2949512  5.2036750 1.000468 1736.182
## b      0.1625134 0.5113315 -0.6479768  0.9587778 1.004780 1900.657
## sigma 24.9920496 0.4843840 24.2276000 25.7510165 1.000677 1779.212
```

```
WAIC(numeracy_model)
```

```
##       WAIC      lppd  penalty  std_err
## 1 1728.527 -862.5864 1.677229 20.15274
```

Task 1.3

Run a Bayesian regression model that predicts `nCorrect` from `Numeracy` using random intercepts and fixed slopes. Standardize the predictor before running the model and compute the WAIC of the model.

```
numeracy_model_subject <- ulam(
  alist(
    nCorrect ~ dnorm(mu, sigma),
    mu <- a[subject] + b * numeracy_s,
    a[subject] ~ dnorm(a_bar, tau_a),
    a_bar ~ dnorm(0, 0.6),
    tau_a ~ dnorm(0, 1),
    b ~ dnorm(1, 0.6),
    sigma ~ dexp(2)
  ), data = risk_cut_list, chains = 4, cores = 4, log_lik = TRUE
)
```

```
## Running MCMC with 4 parallel chains, with 1 thread(s) per chain...
##
## Chain 1 Iteration:    1 / 1000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 1000 [  0%]  (Warmup)
## Chain 3 Iteration:    1 / 1000 [  0%]  (Warmup)
## Chain 4 Iteration:    1 / 1000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 1000 [ 10%]  (Warmup)
## Chain 2 Iteration:  100 / 1000 [ 10%]  (Warmup)
## Chain 3 Iteration:  100 / 1000 [ 10%]  (Warmup)
## Chain 1 Iteration:  200 / 1000 [ 20%]  (Warmup)
## Chain 3 Iteration:  200 / 1000 [ 20%]  (Warmup)
## Chain 1 Iteration:  300 / 1000 [ 30%]  (Warmup)
## Chain 2 Iteration:  200 / 1000 [ 20%]  (Warmup)
## Chain 1 Iteration:  400 / 1000 [ 40%]  (Warmup)
## Chain 2 Iteration:  300 / 1000 [ 30%]  (Warmup)
## Chain 3 Iteration:  300 / 1000 [ 30%]  (Warmup)
## Chain 4 Iteration:  100 / 1000 [ 10%]  (Warmup)
## Chain 1 Iteration:  500 / 1000 [ 50%]  (Warmup)
## Chain 1 Iteration:  501 / 1000 [ 50%]  (Sampling)
## Chain 3 Iteration:  400 / 1000 [ 40%]  (Warmup)
## Chain 4 Iteration:  200 / 1000 [ 20%]  (Warmup)
## Chain 1 Iteration:  600 / 1000 [ 60%]  (Sampling)
## Chain 3 Iteration:  500 / 1000 [ 50%]  (Warmup)
## Chain 3 Iteration:  501 / 1000 [ 50%]  (Sampling)
## Chain 4 Iteration:  300 / 1000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 1000 [ 70%]  (Sampling)
## Chain 2 Iteration:  400 / 1000 [ 40%]  (Warmup)
## Chain 3 Iteration:  600 / 1000 [ 60%]  (Sampling)
## Chain 4 Iteration:  400 / 1000 [ 40%]  (Warmup)
## Chain 1 Iteration:  800 / 1000 [ 80%]  (Sampling)
## Chain 1 Iteration:  900 / 1000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2 Iteration:  500 / 1000 [ 50%]  (Warmup)
## Chain 2 Iteration:  501 / 1000 [ 50%]  (Sampling)
## Chain 3 Iteration:  700 / 1000 [ 70%]  (Sampling)
## Chain 4 Iteration:  500 / 1000 [ 50%]  (Warmup)
## Chain 4 Iteration:  501 / 1000 [ 50%]  (Sampling)
```

```
## Chain 1 finished in 1.3 seconds.
## Chain 2 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3 finished in 1.5 seconds.
## Chain 2 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4 finished in 1.7 seconds.
## Chain 2 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2 finished in 1.8 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 1.6 seconds.
## Total execution time: 1.9 seconds.
```

```
precis(numeracy_model_subject, depth = 2)
```

```
##              mean        sd       5.5%     94.5%     rhat   ess_bulk
## a[1]    1.3065271 1.2570272 -0.5333489  3.037832 1.062208  83.467462
## a[2]    1.3481924 1.2205800 -0.5315219  3.127767 1.070753 106.615753
## a[3]    1.3175835 1.2909547 -0.7241622  3.192973 1.080209 121.919445
## a[4]    1.3363521 1.2586745 -0.4483804  3.040700 1.065594 110.657397
## a[5]    1.2813798 1.2828847 -0.6350112  3.152837 1.053543  75.004759
## a[6]    1.2989007 1.2715896 -0.7168493  3.187041 1.061681  99.356216
## a[7]    1.3559261 1.1692699 -0.4504705  3.149575 1.067156 112.479161
## a[8]    1.3186709 1.2771606 -0.6404766  3.206268 1.073010 139.072443
## a[9]    1.3041020 1.1755950 -0.4745914  3.029965 1.057128  80.807530
## a[10]   1.3243537 1.2383442 -0.6860163  3.067839 1.066864  76.414781
## a[11]   1.2921990 1.2699798 -0.6058620  3.106030 1.073371  83.501147
## a[12]   1.3214666 1.2663631 -0.6555854  3.107608 1.071853 116.477269
## a[13]   1.3262290 1.3590743 -0.6467916  3.278705 1.068223 100.215271
## a[14]   1.3026854 1.2239174 -0.5587417  3.105429 1.075927 144.255184
## a[15]   1.2716282 1.3897455 -0.7937395  3.132704 1.086962  88.731418
## a[16]   1.2871096 1.3225986 -0.7426868  3.173245 1.091088 121.042347
## a[17]   1.3175100 1.2095588 -0.5047731  3.041040 1.051813  73.524416
## a[18]   1.2931280 1.2503978 -0.6985124  3.099603 1.069046  86.705496
## a[19]   1.3079172 1.2530213 -0.6372359  3.009445 1.062132  85.802990
## a[20]   1.3206056 1.2450827 -0.6356315  3.092163 1.075333  94.781255
## a[21]   1.2916522 1.2530168 -0.6205843  3.206180 1.066638  88.667789
## a[22]   1.3134628 1.2871898 -0.6638352  3.067928 1.070095 101.559730
## a[23]   1.3094239 1.2598479 -0.7285002  3.141993 1.073939  90.461482
## a[24]   1.3385538 1.3505481 -0.6667681  3.256707 1.089520 132.926656
## a[25]   1.2981965 1.2679208 -0.6381127  3.066709 1.049686  76.075509
## a[26]   1.3073529 1.2112960 -0.6269851  3.087842 1.070189 105.021855
## a[27]   1.2875216 1.2116054 -0.5858857  2.995516 1.053775  99.154936
## a[28]   1.3409493 1.2708014 -0.7193861  3.304925 1.070741  94.376536
## a[29]   1.2908908 1.3334922 -0.7153308  3.265037 1.070106  94.430677
```

```
## a[30]    1.2905685 1.3063689 -0.7458065  3.306734 1.071652  94.968856
## a[31]    1.3464771 1.2882423 -0.7265115  3.334385 1.099240 132.015679
## a[32]    1.3320781 1.2125628 -0.5982792  3.171647 1.072964 110.816252
## a[33]    1.3072167 1.3319105 -0.6424213  2.941001 1.069423  99.056303
## a[34]    1.3632255 1.1571566 -0.3809317  3.104105 1.069916  91.067721
## a[35]    1.3190632 1.2458877 -0.6136722  3.184068 1.078037  90.931233
## a[36]    1.2885025 1.2771444 -0.7817765  3.122689 1.089106  88.514321
## a[37]    1.3016190 1.2402185 -0.6710447  3.092883 1.055930  93.668965
## a[38]    1.3139742 1.1819589 -0.4890373  3.081174 1.065743  93.091827
## a[39]    1.3117744 1.2036339 -0.5891000  3.010307 1.068802 111.329453
## a[40]    1.3014829 1.3183255 -0.7726438  3.227961 1.057950  91.576901
## a[41]    1.3576376 1.3279464 -0.6472350  3.421044 1.097012 128.126443
## a[42]    1.3203220 1.2409035 -0.5896149  3.192498 1.076476  96.228495
## a[43]    1.2914374 1.2496624 -0.5695445  3.056912 1.069390  81.200134
## a[44]    1.3114309 1.2939191 -0.7475476  3.194071 1.089498 119.402278
## a[45]    1.3400583 1.2590142 -0.5896930  3.152305 1.077790  95.583956
## a[46]    1.3084991 1.2286956 -0.6696171  3.180922 1.082163  91.481063
## a[47]    1.3589067 1.2037577 -0.5138048  3.148137 1.085004 121.957275
## a[48]    1.2781411 1.2119478 -0.6101324  3.090696 1.042830  88.764794
## a[49]    1.2941129 1.2273308 -0.6412594  3.091790 1.056775  79.684571
## a[50]    1.3047084 1.2671663 -0.6033281  3.104904 1.064733 102.782527
## a[51]    1.3070801 1.2139439 -0.5518158  3.135377 1.069125  97.079067
## a[52]    1.3126735 1.1962218 -0.5361338  2.995203 1.054353  82.802788
## a[53]    1.3311856 1.2874963 -0.6200356  3.073051 1.097547  75.276090
## a[54]    1.3119808 1.2566907 -0.6713508  3.073801 1.079458 115.583492
## a[55]    1.3445907 1.2371906 -0.4671066  3.203954 1.054524  78.450425
## a[56]    1.3147107 1.2263884 -0.6426222  3.157135 1.081081 116.446561
## a[57]    1.3503112 1.2561738 -0.6180119  3.232671 1.072087  92.997073
## a[58]    1.2871404 1.2672215 -0.7088823  3.165586 1.071798  86.157556
## a[59]    1.2983845 1.2394493 -0.6459859  3.118763 1.054416  80.287742
## a[60]    1.3350916 1.3920519 -0.8108827  3.336057 1.089157 117.375027
## a[61]    1.3242209 1.3536668 -0.7886257  3.365070 1.083137  88.973731
## a[62]    1.3451129 1.2786267 -0.6366733  3.262120 1.079617 104.304561
## a[63]    1.3342352 1.2802097 -0.5664427  3.351086 1.076443 117.138927
## a[64]    1.3245318 1.2443385 -0.6428810  3.288258 1.076185  97.322611
## a[65]    1.3235613 1.2718445 -0.6096099  3.217115 1.075232 108.218213
## a[66]    1.3171461 1.2903624 -0.7738921  3.283313 1.088674 112.749325
## a[67]    1.2969133 1.2082797 -0.6035056  2.991513 1.058589  79.707355
## a[68]    1.3313636 1.2187768 -0.5921883  3.168503 1.082211 104.129764
## a[69]    1.3012283 1.2818357 -0.5528477  3.179698 1.054460  76.681040
## a[70]    1.3051214 1.2575353 -0.6425660  3.086897 1.060802  98.701517
## a[71]    1.3208858 1.3112544 -0.6892380  3.300317 1.107231 103.555896
## a[72]    1.3287894 1.2089424 -0.6160288  3.109178 1.073010  84.174099
## a[73]    1.3046323 1.2530986 -0.7828859  3.255925 1.065648  90.370860
## a[74]    1.3141399 1.2930523 -0.7405587  3.253029 1.067605 100.164424
## a[75]    1.3092424 1.2608015 -0.6392014  3.203466 1.068299  94.172379
## a[76]    1.3495869 1.2028934 -0.5679287  3.150915 1.070660 103.709660
## a[77]    1.3072934 1.2327979 -0.5744730  3.034349 1.057841  91.277854
## a[78]    1.3273650 1.2926226 -0.7460186  3.296787 1.095020 158.688587
## a[79]    1.3004395 1.3047043 -0.7752515  3.353696 1.073712  96.035133
## a[80]    1.3307836 1.3564023 -0.6754199  3.370247 1.082846  89.619822
## a[81]    1.3237514 1.2953421 -0.6979317  3.242362 1.076973  96.308692
## a[82]    1.2924646 1.3133565 -0.8240281  3.169147 1.080501  93.082713
## a[83]    1.3211464 1.2608065 -0.7045209  3.240262 1.082067  87.738032
```

```
## a[84]   1.2802716 1.2970708 -0.6657169  3.062913 1.077460 116.978224
## a[85]   1.3173128 1.2609018 -0.6015835  3.125924 1.089661  99.252030
## a[86]   1.2857669 1.2228446 -0.6568591  3.019795 1.053167  88.167983
## a[87]   1.3282890 1.2951445 -0.6389398  3.296775 1.064470  77.473130
## a[88]   1.3226565 1.2850758 -0.5662940  3.165502 1.076342  87.047559
## a[89]   1.3132362 1.2645538 -0.6321357  3.155876 1.083828 112.354507
## a[90]   1.3155432 1.2985668 -0.6218053  3.247906 1.096231 106.881090
## a[91]   1.3528029 1.3298011 -0.6955595  3.402566 1.100125 143.145366
## a[92]   1.3012259 1.2664053 -0.6265960  3.079046 1.066422  74.155286
## a[93]   1.3448084 1.2465377 -0.5367866  3.110567 1.079790  85.230292
## a[94]   1.3500298 1.3374590 -0.6114927  3.250858 1.088999 120.936021
## a[95]   1.3382273 1.2989610 -0.6275671  3.280864 1.072637  82.227480
## a[96]   1.3013982 1.2814941 -0.7369003  3.190960 1.073907 100.934940
## a[97]   1.3058970 1.3009117 -0.7316766  3.045063 1.082967  79.879882
## a[98]   1.3415793 1.4129940 -0.7916497  3.426068 1.079392  80.818464
## a[99]   1.3462707 1.3547541 -0.6421942  3.234516 1.087188  90.359849
## a[100]  1.2992445 1.2702677 -0.7341806  3.158414 1.070889 119.902550
## a[101]  1.3294080 1.2312038 -0.5908657  3.165080 1.073305  79.919213
## a[102]  1.2894333 1.2799986 -0.7266323  3.138500 1.072612 121.617772
## a[103]  1.3229488 1.2879133 -0.7493468  3.279972 1.071288  92.643907
## a[104]  1.2967276 1.2869070 -0.7194065  3.213153 1.078869 127.375815
## a[105]  1.2857373 1.2251805 -0.6710709  2.976792 1.077667 125.685160
## a[106]  1.3584534 1.4242807 -0.7188777  3.339890 1.079648  95.550660
## a[107]  1.3006722 1.2578058 -0.5890231  3.250644 1.067108 117.533375
## a[108]  1.3393335 1.2978661 -0.7459788  3.283021 1.090446  85.988102
## a[109]  1.3234651 1.2240238 -0.6101085  3.202699 1.076633 100.866233
## a[110]  1.3065702 1.3050398 -0.7640229  3.252360 1.078473 110.387344
## a[111]  1.3052002 1.3125368 -0.7036514  3.180286 1.071787  95.992821
## a[112]  1.3086829 1.2448238 -0.5955417  3.043024 1.073045 121.566988
## a[113]  1.3123372 1.2933163 -0.7364179  3.131740 1.078959  97.550831
## a[114]  1.2939464 1.2623441 -0.6402131  3.087914 1.075633 130.442332
## a[115]  1.3475232 1.3594216 -0.6543924  3.420326 1.088689 105.267043
## a[116]  1.3207472 1.2709518 -0.6326915  3.063355 1.080944 101.319936
## a[117]  1.3105466 1.2598424 -0.6719756  3.326367 1.075703 127.891657
## a[118]  1.2872687 1.3019074 -0.6844068  3.097287 1.091246 158.033901
## a[119]  1.2821792 1.2054442 -0.6338555  3.104968 1.072391  85.984982
## a_bar   1.2765546 0.5486016  0.3492527  1.968484 1.139441  20.076980
## tau_a   0.8947608 0.7202672  0.1001180  2.237215 1.828731   5.935259
## b       1.0925461 0.5962908  0.1062486  2.011393 1.048885 109.408225
## sigma  48.7065947 1.9607859 45.5364395 51.674028 1.009065 258.433872
```

```
WAIC(numeracy_model_subject)
```

```
##       WAIC      lppd   penalty  std_err
## 1 1358.302 -678.7997 0.3513837 5.425908
```

```
compare(numeracy_model, numeracy_model_subject)
```

```
##                            WAIC        SE    dWAIC      dSE     pWAIC
## numeracy_model_subject 1358.302  5.425908   0.0000       NA 0.3513837
## numeracy_model         1728.527 20.152737 370.2251 14.81962 1.6772292
##                              weight
## numeracy_model_subject 1.000000e+00
## numeracy_model         4.042407e-81
```

Looks like if we consider each subject as a group, our model gets better

Task Set 2

Task 2.1

Create a data table that entails 10,000 posterior samples (rows) for each subject-specific (columns) intercept. Convert the sampled values into probabilities and print the first 10 samples of the first 10 subjects.

```
set.seed(22394)
subject_samples <- extract.samples(numeracy_model_subject, n=10000)$a
probabilities <- as.data.frame(1 / (1 + exp(-subject_samples)))
print(probabilities[1:10, 1:10])
```

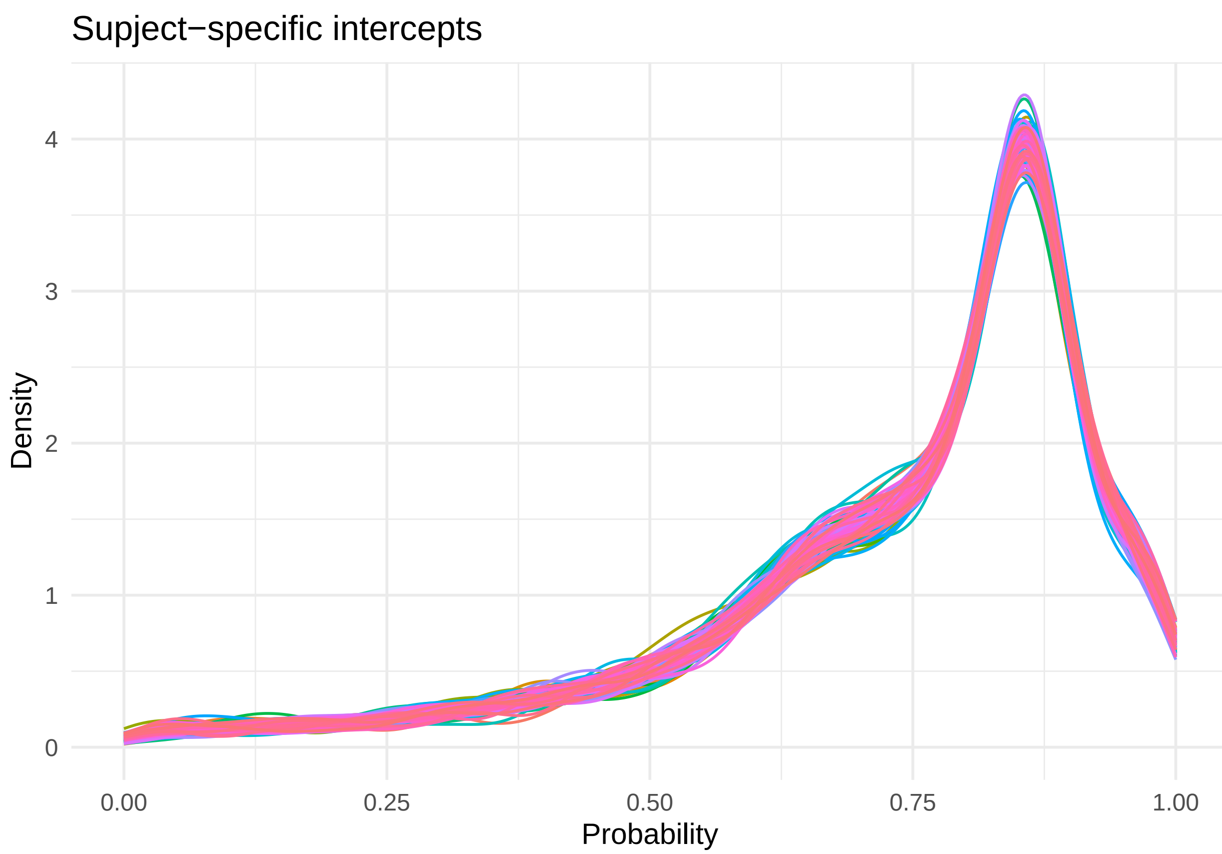
```
##           V1        V2        V3        V4        V5        V6        V7
## 1  0.6160661 0.5852580 0.5476396 0.6057645 0.6243971 0.5938902 0.5731266
## 2  0.6331295 0.6191238 0.5041113 0.6371210 0.6112401 0.6141056 0.6679720
## 3  0.6027970 0.6150185 0.6996110 0.6097485 0.6219951 0.6497535 0.5985733
## 4  0.6108292 0.6858032 0.6497988 0.6326867 0.6595200 0.6101819 0.6304535
## 5  0.6655922 0.6578719 0.6961439 0.6122218 0.6100178 0.6454868 0.6037907
## 6  0.5741511 0.5878947 0.6913418 0.6101217 0.6302441 0.5866510 0.6490245
## 7  0.6689443 0.6293131 0.6481935 0.6074212 0.6734740 0.6271358 0.5737195
## 8  0.5741352 0.6105876 0.5859117 0.6303151 0.5841252 0.6261347 0.6716720
## 9  0.6895334 0.6625358 0.6588746 0.6193872 0.6720179 0.6139333 0.5706293
## 10 0.5730541 0.6072580 0.6324117 0.6172975 0.6018221 0.6195902 0.6987259
##           V8        V9       V10
## 1  0.6501860 0.6241693 0.6266163
## 2  0.5824327 0.5288773 0.6131247
## 3  0.6907137 0.6909997 0.6453092
## 4  0.6909194 0.6222250 0.6906987
## 5  0.6879060 0.5747461 0.7008297
## 6  0.6026081 0.6096002 0.6440261
## 7  0.6837091 0.6597014 0.5927166
## 8  0.5926075 0.6297007 0.6674966
## 9  0.6605735 0.6088236 0.5838793
## 10 0.6134240 0.6468222 0.6510153
```

Task 2.2

Use the posterior samples to plot the posterior distribution of all subject-specific intercepts to show the variability in the performance among subjects. Use the converted values (probabilities).

```
probabilities_long <- probabilities %>%
  pivot_longer(cols = everything(), names_to = "subject", values_to = "probability")

ggplot(probabilities_long, aes(x = probability, color = subject)) +
  geom_density(alpha = 0.5) +
  theme_minimal() +
  theme(legend.position = "none") +
  labs(title = "Supject-specific intercepts",
       x = "Probability",
       y = "Density")
```



Most of the intercepts follow the similar distribution, however, looks like some subjects are different from the others

Task 2.3

Consider the following posterior summaries and traceplots. Which model was estimated and what might be the cause of the convergence problems?

```
# precis(m3)
# traceplot_ulam(m3, pars = c("mu_a", "tau_a", "mu_b", "tau_b"))
```

The model with random intercepts and slopes was estimated. Maybe too complicated model was estimated and it would be enough to have either random intercept or random slope model Other cause could be incorrect specification of the priors for those coefficients