

# Portfolio 1 - Study Group 10

2023-02-24

```
pacman::p_load(tidyverse, rethinking, dagitty)
```

## Loading packages

**1) Testing efficiency** **A)** *Estimate the probabilities of testing positive given that you're infected, and given that you're not infected. Use the grid approximation method as in the book. Use a prior you can defend using. Report the full posterior probability distribution for each case (we can do better than just a single value!).*

(NV)

```
# Set up the number of observations for both cases
n_truepos <- 20
n_trueneg <- 20
observed_pos_trueneg <- 7
observed_pos_truepos <- 11

# Set up a grid of possible true positive probabilities
p_grid <- seq(0, 1, length.out = 1000)

# Set up a prior distribution on p, assuming a uniform prior. The prior is taken from the actual empirical data
prior <- rep(0.73, 1000)

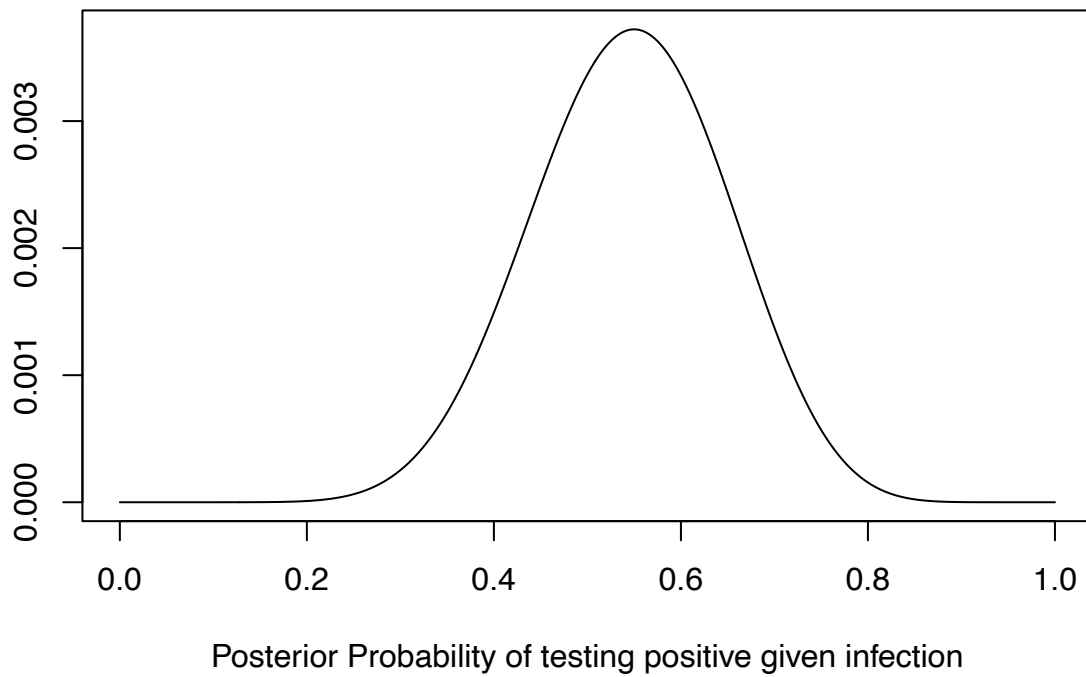
# Calculate the likelihood of the data for each value of p, assuming a binomial distribution
likelihood_infected <- dbinom(observed_pos_truepos, n_truepos, p_grid)
likelihood_not_infected <- dbinom(observed_pos_trueneg, n_trueneg, p_grid)

# Calculate the unnormalized posterior distribution for each case by multiplying the prior and likelihood
posterior_unnormalized_infected <- likelihood_infected * prior
posterior_unnormalized_not_infected <- likelihood_not_infected * prior

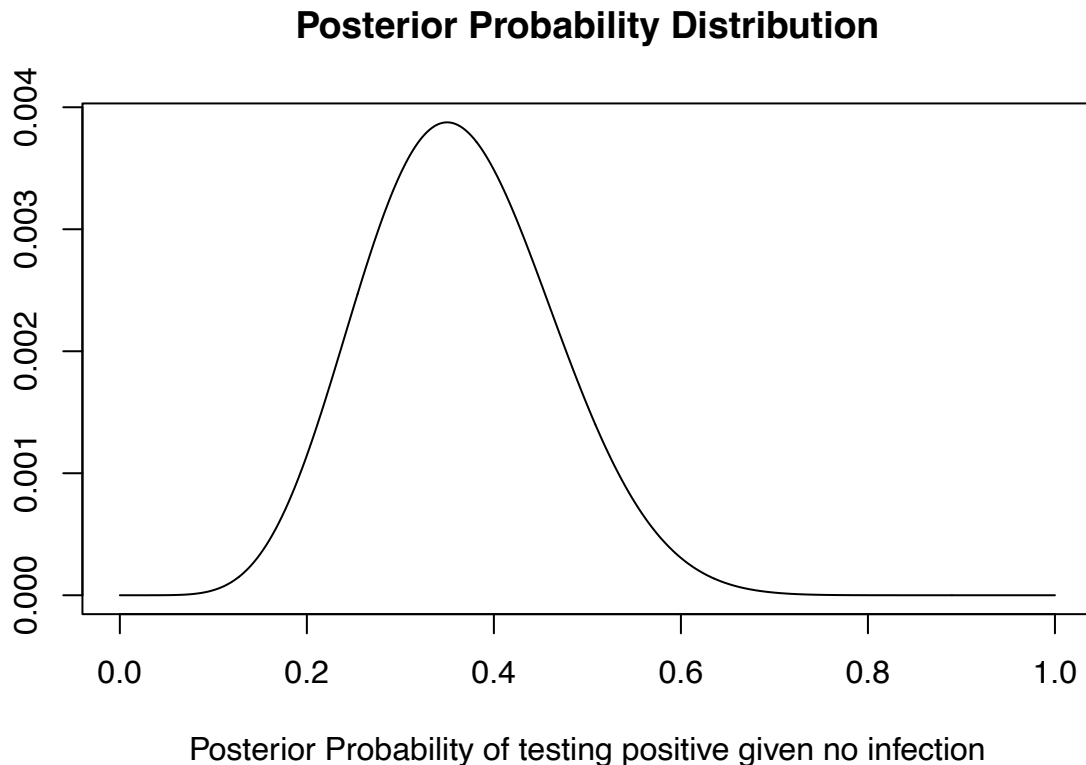
# Normalize the posterior distribution for each case
posterior_infected <- posterior_unnormalized_infected / sum(posterior_unnormalized_infected)
posterior_not_infected <- posterior_unnormalized_not_infected / sum(posterior_unnormalized_not_infected)

# Plot the posterior distribution for each case
plot(p_grid, posterior_infected, type = "l", xlab = "Posterior Probability of testing positive given infected", ylab = "Posterior Probability of testing positive given infected")
```

## Posterior Probability Distribution



```
plot(p_grid, posterior_not_infected, type = "l", xlab = "Posterior Probability of testing positive given infection")
```



B) The government says that they find probability distributions difficult to use. They ask you to provide them with a confidence interval of 95% within which the ‘real’ probability can be found. Do it.

```
# The code generates posterior samples using the grid approximation method and the posterior distribution
samples_infected <- sample(p_grid, size = 1e4, replace = TRUE, prob = posterior_infected)
samples_not_infected <- sample(p_grid, size = 1e4, replace = TRUE, prob = posterior_not_infected)

# Obtaining the 95% credible interval for the 'real' probability in each case
ci_infected <- HPDI(samples_infected, prob = 0.95)
ci_not_infected <- HPDI(samples_not_infected, prob = 0.95)
```

C) The government says that their voters find confidence intervals difficult to read. In addition, they are so wide that it looks like the government doesn’t know what they’re doing. They want a point estimate instead. Give them one.

```
# We choose to use the mean to estimate the probability values. The median would also make sense, but g
mean(ci_infected)
```

```
## [1] 0.548048
```

```
mean(ci_not_infected)
```

```
## [1] 0.3643644
```

## 2) Dark Cellars

Months pass. Thousands of people are tested by the wizards of the world governments. A fancy company analyses the data, and determine, with very high confidence they say, the probability of testing positive with the current test. They give the following point estimates:

- A 53% chance of testing positive if you are infected.
- A 45% chance of testing positive if you are not infected.

A) You are sitting in your dark cellar room, writing an apology to the Danish government, when you receive a positive test result on your phone. Oh, that party last weekend. In order to fight the boredom of isolation life, you start doing statistical inference. Using Bayes theorem, estimate the probability that you are infected, given that it is a priori equally likely to be infected or not to be.

(DKA)

```
# First, it calculates the probability of being infected, which is set to 50%. (P(A))

# Then, it calculates the probability of testing positive, which is the average between the probability

# The probability of testing positive given being infected (B given A) is set to 0.53. (P(B|A))

# The code then applies Bayes' theorem to calculate the probability of being infected after testing pos

# Building a function for bayes theorem
bayes <- function(A, B, BA) {
  AB <- (A * BA) / B
  return(AB)
}

A1 <- 0.5
B1 <- (0.53 * 0.5) + (0.45 * 0.5)
BA1 <- 0.53

bayes(A1,B1,BA1)
```

```
## [1] 0.5408163
```

```
# Which finally puts  $P(A|B) = 0.54$ 
```

B) A quick Google search tells you that about 546.000<sup>[2]</sup> people in Denmark are infected right now. Use this for a prior instead.

```
# Probability of being infected: infected (546.000) divided by danish population (5,85 mio). (P(A))

# Probability of testing positive: 53% + 45% divided by 2. P(B)

# Probability of testing positive given being infected: 0.53 (P(B|A))

# Probability of being infected after testing positive:  $(P(A) * P(B|A)) / P(B) = (58.5/5.46) * 0.53 / ($ 
```

```
A2 <- 5.46/58.57

B2 <- (0.53 * A2) + (0.45 * (1-A2))

BA2 <- 0.53

bayes(A2,B2,BA2)
```

```
## [1] 0.1080046
```

C) A friend calls and says that they have been determined by a wizard to be infected. You and your friend danced tango together at the party last weekend. It has been estimated that dancing tango with an infected person leads to an infection 32% of the time<sup>[3]</sup>. Use this information to construct a prior instead.

```
# Probability of being infected from tango dancing: 32%. (P(A))

# Probability of testing positive: 53% + 45% divided by 2. P(B)

# Probability of testing positive given being infected: 0.53. P(B|A)

# Probability of being infected after testing positive: (P(A) * P(B|A)) / P(B) = (0.32 * 0.53 / (0.53 + 0.45))

A3 <- 0.32

B3 <- (0.53 * A3) + (0.45 * (1-A3))

BA3 <- 0.53

bayes(A3,B3,BA3)
```

```
## [1] 0.3566022
```

D) You quickly run and get two more tests. One is negative, the other positive. Update your estimate.

(LA)

```
#probability of being infected from tango dancing - A - 32%

#probability of testing positive - B - 53% + 45%

#probability of testing positive given being infected - B given A - 0.53

#probability of being infected after testing positive - P of A given B = (P(A) * P(B|A)) / P(B) = (0.32 * 0.53 / (0.53 + 0.45))

A3 <- 0.32

B3 <- (0.53 * A3) + (0.45 * (1-A3))

BA3 <- 0.53
```

```

new_belief1 <- bayes(A3,B3,BA3)

#probability of being infected from tango dancing - A - 32%

#probability of testing negative - B - (1-0.53) * A + (1-0.45) * (1-A)

#probability of testing negative given being infected - B given A - 0.55

#probability of being infected after testing negative - P of A given B = (P(A) * P(B/A)) / P(B) = (0.32

A3.1 <- new_belief1

B3.1 <- (1-0.53) * A3.1 + (1-0.45) * (1-A3.1)

BA3.1 <- 0.47

new_belief2 <- bayes(A3.1,B3.1,BA3.1)

A3.2 <- new_belief2

B3.3 <- (0.53 * A3.2) + (0.45 * (1-A3.2))

bayes(A3.2,B3.3,BA3)

```

```
## [1] 0.358082
```

E) In a questionnaire someone sent out for their exam project, you have to answer if you think you are infected. You can only answer yes or no (a bit like making a point estimate). What do you answer?

```

# As the questionnaire doesn't penalize answering wrongly, it would make sense to set the threshold f
# newest posterior of being infected

infect <- bayes(A3.2,B3,BA3)
if (infect > 0.5){
  print("I am infected")
} else {
  print("I am not infected")
}

```

```
## [1] "I am not infected"
```

F) You are invited to a party. They ask if you are infected. They also think that if you are in doubt, staying home is safer. Because they studied statistics, they formulate this as preferring that you used an asymmetric loss function when making your decision: it is three times worse to falsely answer not infected and come sick to the party, as compared to to falsely answering infected and staying home while healthy. What do you answer?

(PM)

```
# chance of not being infected
not_infec <- 1-bayes(A3.2,B3,BA3)

# The same question as the one before, however; in this one we are penalizing the risk of falsely showing
if (infect*3 > not_infec){
  print("Stay home")
} else {
  print("Go Party Uartig bbbby")
}
```

## [1] "Stay home"

### 3) Causal Models

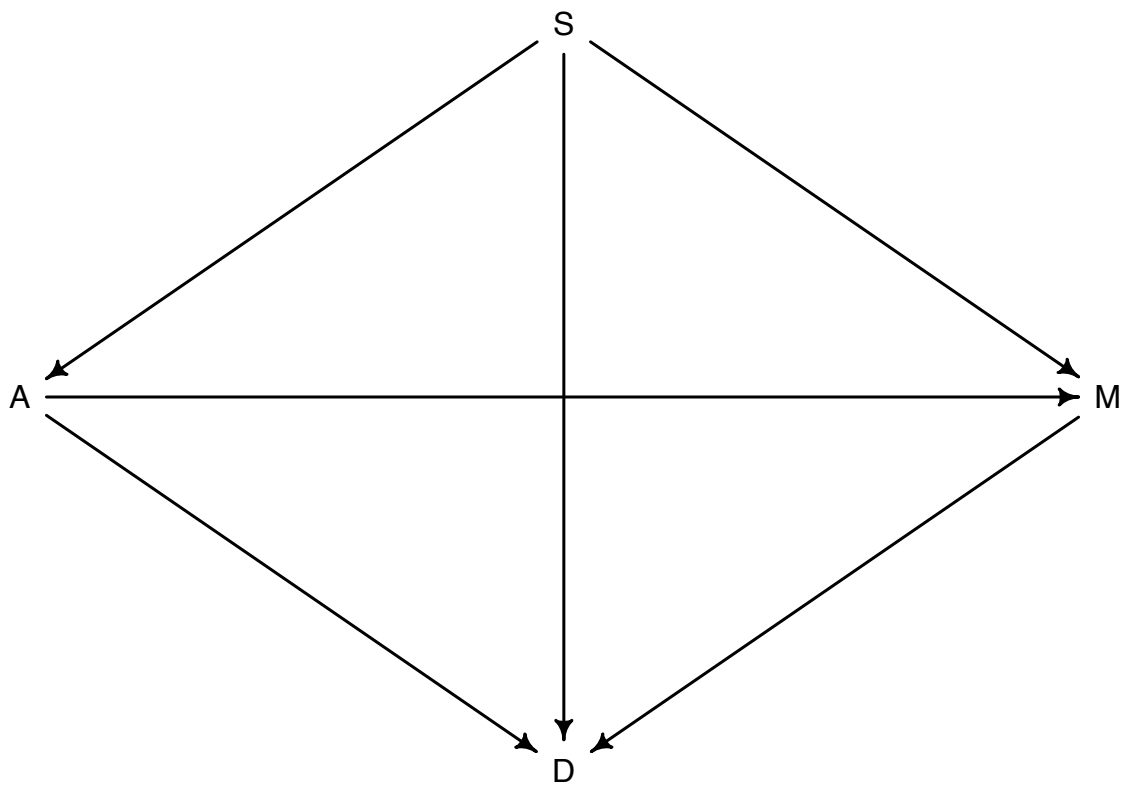
A problem from our textbook *Statistical Rethinking* (2nd ed.) (p. 160):

**5H4.** Here is an open practice problem to engage your imagination. In the divorce data, states in the southern United States have many of the highest divorce rates. Add the *South* indicator variable to the analysis. First, draw one or more DAGs that represent your ideas for how Southern American culture might influence any of the other three variables (*D*, *M*, or *A*). Then list the testable implications of your DAGs, if there are any, and fit one or more models to evaluate the implications. What do you think the influence of “Southernness” is?

```
data(WaffleDivorce)
d <- WaffleDivorce
# standardize variables
d$D <- standardize(d$Divorce)
d$M <- standardize(d$Marriage)
d$A <- standardize(d$MedianAgeMarriage)
d$S <- d$South

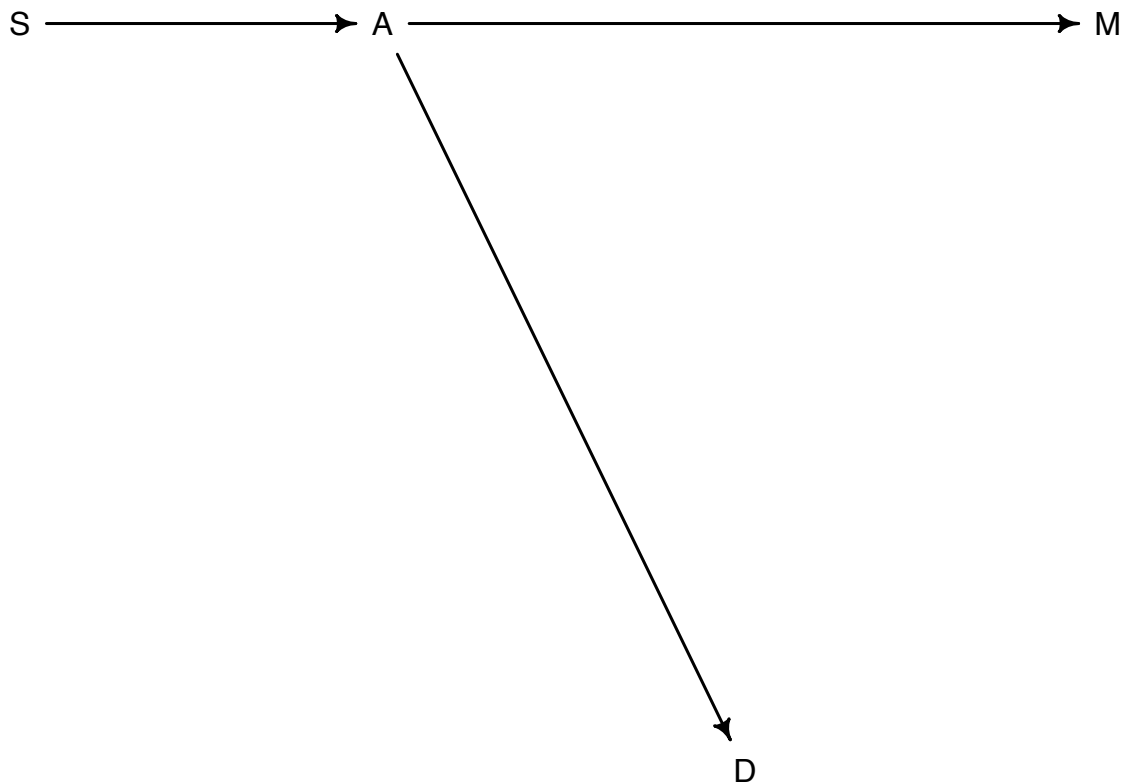
DMA_dag1 <- dagitty('dag{D<-S->M->D}')
impliedConditionalIndependencies( DMA_dag1)

#Dag 1
dag_1 <- dagitty( "dag{S ->D;S->M;S->A;A -> D; A -> M; M -> D }" )
coordinates(dag_1) <- list( x=c(A=0,D=1,M=2,S=1) , y=c(A=0,D=1,M=0,S=-1) )
drawdag( dag_1 )
```



```
#Dage 2  
dag_2 <- dagitty( "dag{S ->A;A -> D; A -> M }" )  
coordinates(dag_2) <- list( x=c(A=0,D=1,M=2,S=-1) , y=c(A=0,D=1,M=0,S=0) )  
drawdag( dag_2 )
```





#Testable implications

```
SDMA_dag1 <- dagitty( "dag{S ->A;S->M;S->A;A -> D; A -> M; M -> D }" )
impliedConditionalIndependencies( SDMA_dag1)
```

```
## D _||_ S | A, M
```

```
SDMA_dag2 <- dagitty( "dag{S ->A;A -> D; A -> M }" )
impliedConditionalIndependencies( SDMA_dag2 )
```

```
## D _||_ M | A
```

```
## D _||_ S | A
```

```
## M _||_ S | A
```

In plain terms, the testable implications of the first DAG is that D is conditionally independent of S given variables A and M. That is, if you know the values of A and M, then information about S does not provide much or any additional information about D.

The testable implications of the second DAG are following three statements: D is conditionally independent of M, given A. D is conditionally independent of S, given A. M is conditionally independent of S, given A.

(KM)

```

m1 <- quap(
  alist(
    D ~ dnorm( mu , sigma ) ,
    mu <- a + bM*M + bA*A ,
    a ~ dnorm( 0 , 0.2 ) ,
    bM ~ dnorm( 0 , 0.5 ) ,
    bA ~ dnorm( 0 , 0.5 ) ,
    sigma ~ dexp( 1 )
  ) , data = d )
precis( m1 )

```

| ##       |  | mean          | sd         | 5.5%       | 94.5%      |
|----------|--|---------------|------------|------------|------------|
| ## a     |  | -2.514032e-09 | 0.09707605 | -0.1551463 | 0.1551463  |
| ## bM    |  | -6.538074e-02 | 0.15077309 | -0.3063453 | 0.1755838  |
| ## bA    |  | -6.135135e-01 | 0.15098363 | -0.8548145 | -0.3722125 |
| ## sigma |  | 7.851182e-01  | 0.07784345 | 0.6607093  | 0.9095271  |

```

m2 <- quap(
  alist(
    D ~ dnorm( mu , sigma ) ,
    mu <- a + bA*A + bS*S ,
    a ~ dnorm( 0 , 0.2 ) ,
    bA ~ dnorm( 0 , 0.5 ) ,
    bS ~ dnorm( 0 , 0.5 ) ,
    sigma ~ dexp( 1 )
  ) , data = d )
precis( m2 )

```

| ##       |  | mean        | sd         | 5.5%        | 94.5%       |
|----------|--|-------------|------------|-------------|-------------|
| ## a     |  | -0.07720307 | 0.10604738 | -0.24668726 | 0.09228112  |
| ## bA    |  | -0.53179227 | 0.10918474 | -0.70629057 | -0.35729397 |
| ## bS    |  | 0.35627152  | 0.21489784 | 0.01282327  | 0.69971977  |
| ## sigma |  | 0.76436902  | 0.07587958 | 0.64309879  | 0.88563925  |

```

m3 <- quap(
  alist(
    D ~ dnorm( mu , sigma ) ,
    mu <- a + bM*M + bS*S ,
    a ~ dnorm( 0 , 0.2 ) ,
    bM ~ dnorm( 0 , 0.5 ) ,
    bS ~ dnorm( 0 , 0.5 ) ,
    sigma ~ dexp( 1 )
  ) , data = d )
precis( m3 )

```

| ##       |  | mean       | sd         | 5.5%       | 94.5%      |
|----------|--|------------|------------|------------|------------|
| ## a     |  | -0.1022073 | 0.11466917 | -0.2854708 | 0.08105617 |
| ## bM    |  | 0.3348697  | 0.11982281 | 0.1433697  | 0.52636970 |
| ## bS    |  | 0.5005156  | 0.23135965 | 0.1307582  | 0.87027305 |
| ## sigma |  | 0.8615998  | 0.08577342 | 0.7245174  | 0.99868234 |

```

m4 <- quap(
  alist(
    D ~ dnorm( mu , sigma ) ,
    mu <- a + bM*M + bA*A + bS*S ,
    a ~ dnorm( 0 , 0.2 ) ,
    bM ~ dnorm( 0 , 0.5 ) ,
    bA ~ dnorm( 0 , 0.5 ) ,
    bS ~ dnorm( 0 , 0.5 ) ,
    sigma ~ dexp( 1 )
  ) , data = d )
precis( m4 )

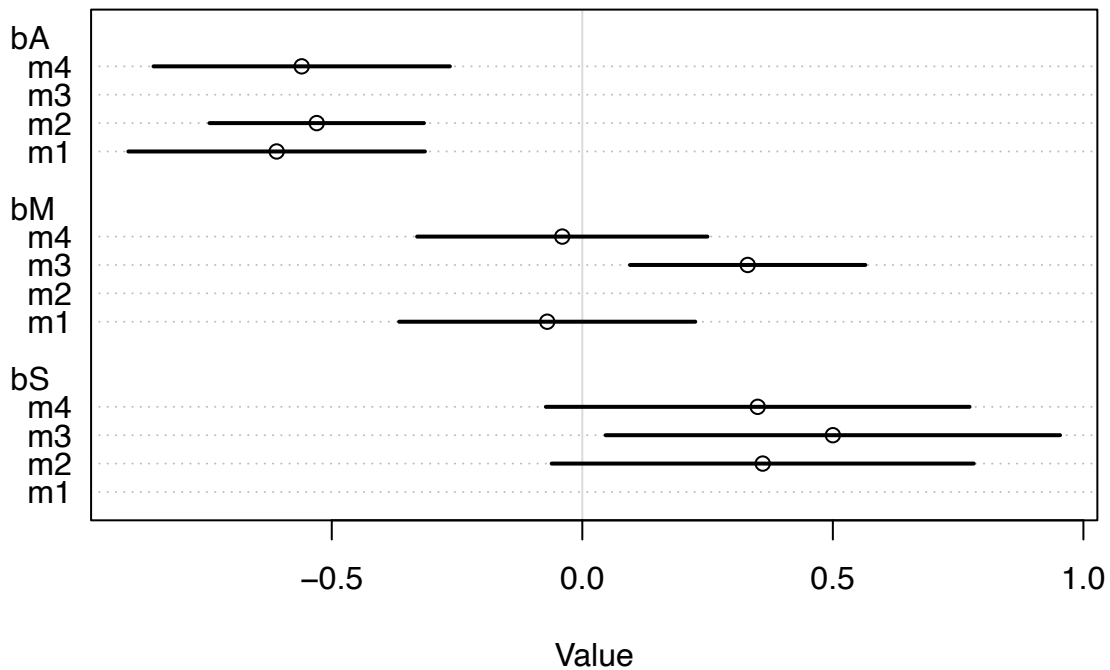
```

| ##       | mean        | sd         | 5.5%         | 94.5%       |
|----------|-------------|------------|--------------|-------------|
| ## a     | -0.07587186 | 0.10600078 | -0.245281586 | 0.09353787  |
| ## bM    | -0.04225886 | 0.14778442 | -0.278446905 | 0.19392919  |
| ## bA    | -0.56168871 | 0.15089645 | -0.802850379 | -0.32052704 |
| ## bS    | 0.34993853  | 0.21571500 | 0.005184286  | 0.69469276  |
| ## sigma | 0.76287476  | 0.07579564 | 0.641738682  | 0.88401084  |

```

plot( coeftab(m1,m2, m3, m4), par=c("bA","bM", "bS") )

```



The code above is used to fit four different models to evaluate the implications of the DAGs. The models examine the relationship between the variables Divorce rate, Marriage rate, Median age of marriage and S Southernness which was fit as an indicator variable. The first model (m1) examines the relationship between D, M and A. The second model (m2) examines the relationship between D, A and S. The third

model (m3) examines the relationship between D, M and S. Finally, the fourth model (m4) examines the relationship between D, M, A and S.

The results of the models show that Southernness has a very little to no impact on the divorce rate, when one or both of the other predictors are present. Marriage rate also tells us very little when the median age predictor is present. However, the median age is associated with the divorce rate regardless of which of the other predictors are present. The plot of the coefficients of the models shows that the Southernness indicator variable (S) has no impact.

To sum up, this suggests that Southernness does not offer any additional predictive power of the divorce rate. We think these results indicate that DAG number 2 is preferable. Given that M does not seem to add any predictive power for Divorce rate, which is implied by the DAG, and that S only adds predictive power when A is not included in the model which makes sense because S is only has an effect through A (pipe).

# Portfolio 2 - Study group 10

2023-04-16

```
# (KM)
### Loading packages
pacman::p_load(tidyverse, dagitty, rethinking, truncnorm)
```

## Task 1: The DAG

A) Come up with an incredibly interesting and scientifically important made-up *example* for a phenomenon to investigate. Decide on two variables (an outcome and a predictor) that you would like to investigate the relation between.

If in doubt, you can be inspired by Peter's example in the file `example.md`. Note that in Peter's example there are a lot of different variables - this is only to make you start thinking - it is not necessary to include as many.

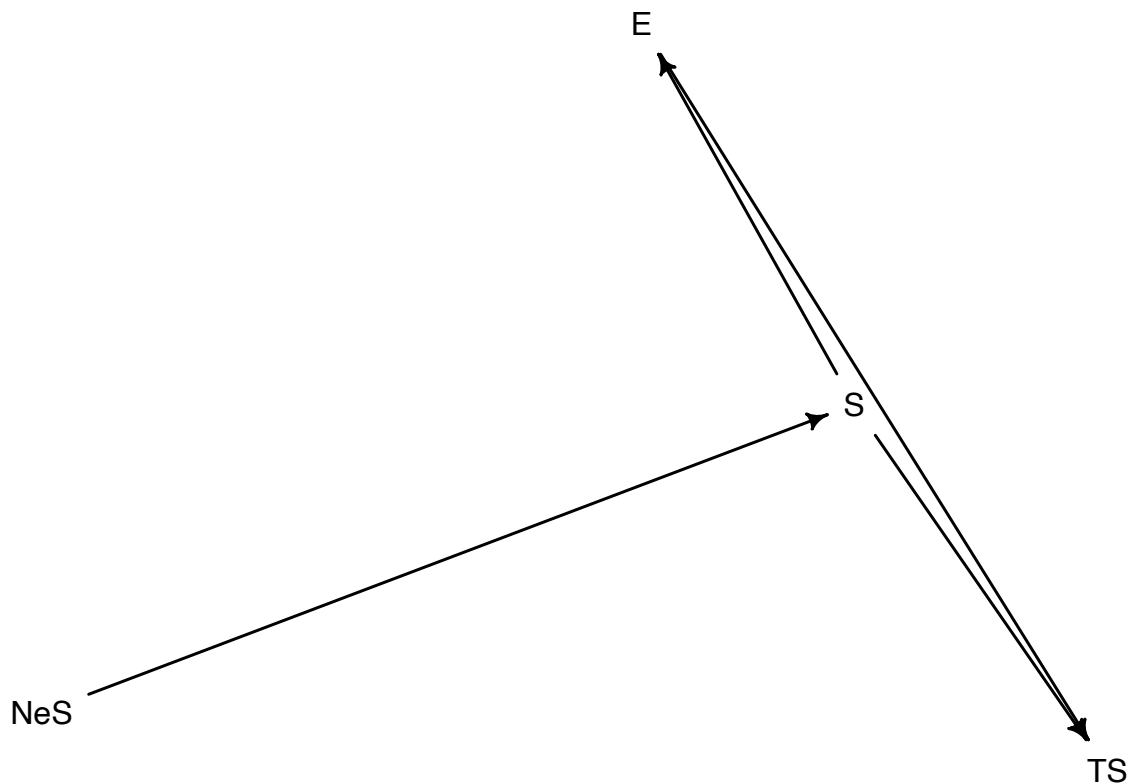
- The phenomenon we want to investigate is the relation between exam anxiety (E) and test score (TS). In our example, we believe that time spent studying per week (S) has an effect on exam anxiety and the test score. Furthermore, non-educational socializing per week (NeS) is also believed to be related to the time spent studying.

B) Make a DAG for the phenomenon. Make it medium complicated: that means, make sure there are some different kinds of relations (see next step). Change it if you don't get anything interesting for the next steps.

*Draw it* somehow (on paper, in R, laser engraved in diamond). *Code it* using dagitty (this is a nice tool: <http://dagitty.net/dags.html> )

```
dag <- dagitty( "dag {
  NeS -> S -> TS
  S -> E -> TS
}" )

drawdag(dag)
```



C) Find elemental forms of variable relations in the DAG (i.e., forks, pipes, colliders, and their descendants).

- There are various elemental forms of variable relations in our DAG. We have a pipe from  $\text{NeS} \rightarrow \text{S} \rightarrow \text{TS}$ , and a pipe running from  $\text{NeS} \rightarrow \text{S} \rightarrow \text{E} \rightarrow \text{TS}$ . There's a fork in  $\text{E} \leftarrow \text{S} \rightarrow \text{TS}$

D) Find out what variables to include (and not include) in a multiple linear regression to avoid “back door” (AKA non-causal) paths. Do this first with your eyes and your mind. Then you can use dagitty’s function `adjustmentSets()`.

We want to investigate the relation between exam anxiety (predictor) and Test score (outcome). Given our DAG we will condition on Studying time given it is a confounder and by conditioning on Studying time we block the back door path.

```
# (DKA)
adjustmentSets( dag , exposure="E" , outcome="TS" )
```

```
## { S }
```

The `adjustmentSets` agrees that we should include S (studying time) in our model

E) Find out which conditional independencies the DAG implies.

When looking at the DAG, we infer the following conditional independencies:

$\text{E} \parallel \text{NeS} \mid \text{S}$   $\text{TS} \parallel \text{NeS} \mid \text{S}$

E is independent of NeS conditioned on S TS is independent of NeS conditioned on S.

That is, there is no additional value in knowing NeS, once we already know S.

```
impliedConditionalIndependencies(dag)
```

```
## E _||_ NeS | S  
## NeS _||_ TS | S
```

```
E // NeS | S NeS // TS | S
```

The function suggests that NeS is independent of TS conditioned on S. However, to us it seems counterintuitive for the conditional independency to move opposite the causal path.

The statement -  $NeS \parallel TS \mid S$  - seems to imply if we know S, then knowing TS does not add information about NeS, which to us does not make sense, given that it goes against the causal path.

To us it would make sense to say, if we know S, then knowing NeS does not add any additional information about TS. In a more formal way:

```
TS // NeS | S
```

So for our conditional independencies, we proceed with our first suggestions.

## Task 2: The data

Simulate some data that fits the DAG. There are many ways to do this. A simple way is just to sample one variable from a normal distribution which has another variable as mean. McElreath does this in the book a few times, and you can use this as inspiration.

```
# (LA)  
N <- 1:100  
  
NeS <- rep(1,100)  
  
S <- rep(1,100)  
  
E <- rep(1,100)  
  
TS <- rep(1,100)  
  
d <- data.frame(N,NeS,S,E,TS)  
  
for (i in 1:nrow(d)) {  
  d$NeS[i] <- rnorm(1, mean = 28, sd = 6)  
  
  #NeS is on a "theoretical scale" of 0-56 because we find it reasonable that one third  
  #of a week can go to NeS, we put the mean in the middle at 28 hours per week,  
  #sd of 6 makes the 95 within 10 - 46 hours  
  
  d$S[i] <- rnorm(1, mean = 56 - d$NeS[i], sd = 6)  
  
  #in our DAG Studying time is dependent on NeS, because social interaction  
  #is of higher importance, the studying time is then also sampled on a scale  
  #from 0-56, but inverse from the NeS, the sd is the same  
  
  d$E[i] <- rnorm(1, mean = 0.6 + d$S[i]*(-0.005), sd = 0.1)
```

```

#Exam anxiety is sampled from a linear model based on S, we set the
#intercept at 0,6, and a slope of -0.005, because we think the
#less you have studied the more anxious you will be on the day of exam,
#so Studying time lowers Exam anxiety. Exam anxiety has a sd of 0.1,
#and on a "theoretical scale" of 0-1

d$TS[i] <- rtruncnorm(1, a = 0, b = 100, mean = (100/56 * d$S[i]) * (1-d$E[i]), sd = 10)}

#Time score is simulated from a truncnorm, which restrains the distribution between 0-100,
#that is how our virtual tests are scored. We divide the ultimate score 100 with the ultimate
#studying 56 hours per week, and multiply it by subject[i] studying time,
#to get a predicted score based only on studytime.
#We then inverse the exam anxiety of the subject, and multiply the two numbers.
#This implies that a if you have zero anxiety and study 56 hours a week you will get a perfect score,
#however no matter how much you study, if you have an exam anxiety score of 1 you're doomed.

```

### Task 3: Statistics

Run multiple linear regressions using the simulated data to test the conditional independencies implied by your DAG. Make sure to avoid backdoor paths. See that the linear model shows the conditional independencies implied by your DAG, implying that the data and the DAG are compatible.

```

# (PM)
# first we standardize the variables
d$NeS <- standardize(d$NeS)
d$S <- standardize(d$S)
d$E <- standardize(d$E)
d$TS <- standardize(d$TS)

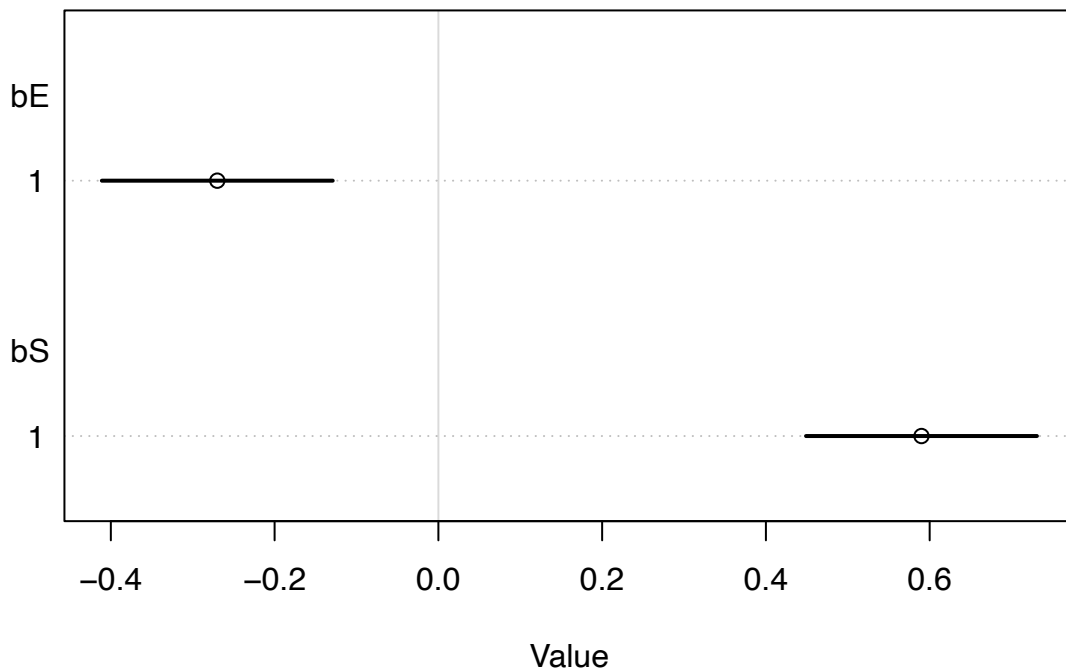
# We then evaluate the implications of the DAG. The model examines the relationship
#between the variables Test Score, Exam Anxiety and Study time.
model.1 <- quap(
  alist(
    TS ~ dnorm( mu , sigma ) ,
    mu <- a + bE*E + bS*S ,
    a ~ dnorm( 0 , 0.2 ) ,
    bE ~ dnorm( 0 , 0.5 ) ,
    bS ~ dnorm( 0 , 0.5 ) ,
    sigma ~ dexp( 1 )
  ) , data = d )
precis( model.1 )

```

| ##       | mean          | sd         | 5.5%       | 94.5%       |
|----------|---------------|------------|------------|-------------|
| ## a     | 3.632632e-06  | 0.06167648 | -0.0985673 | 0.09857456  |
| ## bE    | -2.704806e-01 | 0.07193412 | -0.3854452 | -0.15551600 |
| ## bS    | 5.861767e-01  | 0.07194699 | 0.4711915  | 0.70116186  |
| ## sigma | 6.483645e-01  | 0.04563518 | 0.5754307  | 0.72129835  |

```
plot( coeftab(model.1), par=c("bE", "bS"))
```





We want to investigate the effects of exam anxiety on test score, but given our DAG we need to include Study time in our model. We do this in order to close the back door, as the function `adjustmentSets()` also implies. The results fit our DAG

## Task 4: Messing it up

A) Change your model from *Task 3* to have an open back door path and see if you would make a false inference.

*# We exclude Study Time to have an open backdoor as implied by the adjustment sets.  
#To exclude the effects on S on TS*

```
model.2 <- quap(
  alist(
    TS ~ dnorm( mu , sigma ) ,
    mu <- a + bE*E ,
    a ~ dnorm( 0 , 0.2 ) ,
    bE ~ dnorm( 0 , 0.2 ) ,
    sigma ~ dexp( 1 )
  ) , data = d )
precis( model.2 )
```

```
##           mean          sd      5.5%      94.5%
## a      2.742948e-06 0.07742088 -0.1237308  0.1237363
## bE     -4.555739e-01 0.07836652 -0.5808187 -0.3303291
## sigma  8.396731e-01 0.05946454  0.7446373  0.9347089
```

When we do not include Study time in our model, ie. open the backdoor, we see that the estimate for bE becomes exaggerated.

B) Simulate some data that doesn't fit the DAG and fit the model from *Task 3* above to the new simulated data. You should now get results that doesn't fit the assumptions of your DAG.

```
# (NV)
dn <- data.frame(N,NeS,S,E,TS)

# We do the same simulation as before.
for (i in 1:nrow(dn)) {
  dn$NeS[i] <- rnorm(1, mean = 28, sd = 6)
  dn$S[i] <- rnorm(1, mean = 56 - dn$NeS[i], sd = 6)
  dn$E[i] <- rnorm(1, mean = 0.6 + dn$S[i]*(-0.005), sd = 0.1)

  #In this case, Exam Anxiety is excluded in the simulation of test score.
  #Therefore, study time is the only variable that affects the test score simulation.

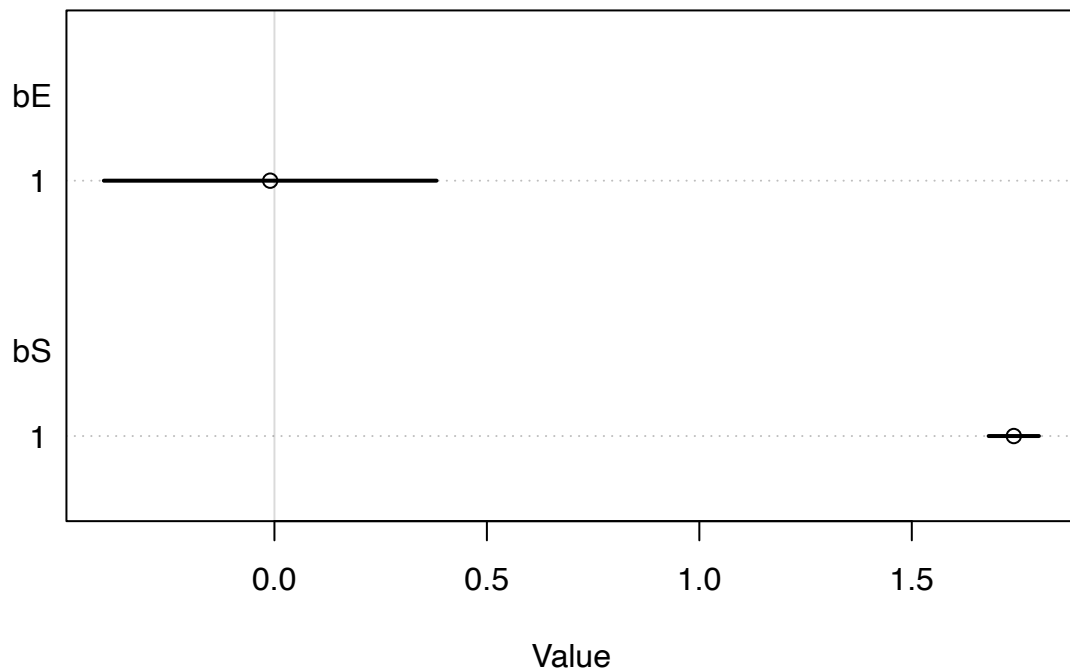
  dn$TS[i] <- rtruncnorm(1, a = 0, b = 100, mean = (100/56 * dn$S[i]), sd = 10)
}

# standarizing all the data for the "fake"-data
d$NeS <- standardize(dn$NeS)
d$S <- standardize(dn$S)
d$E <- standardize(dn$E)
d$TS <- standardize(dn$TS)
```

```
model.1 <- quap(
  alist(
    TS ~ dnorm( mu , sigma ) ,
    mu <- a + bE*E + bS*S ,
    a ~ dnorm( 0 , 0.2 ) ,
    bE ~ dnorm( 0 , 0.2 ) ,
    bS ~ dnorm( 0 , 0.5 ) ,
    sigma ~ dexp( 1 )
  ) , data = dn )
precis( model.1 )
```

| ##       |  | mean         | sd         | 5.5%       | 94.5%     |
|----------|--|--------------|------------|------------|-----------|
| ## a     |  | -0.004834182 | 0.19954147 | -0.3237400 | 0.3140716 |
| ## bE    |  | -0.006972174 | 0.19974802 | -0.3262081 | 0.3122637 |
| ## bS    |  | 1.741660889  | 0.03021832 | 1.6933662  | 1.7899556 |
| ## sigma |  | 8.755570067  | 0.58235428 | 7.8248554  | 9.6862847 |

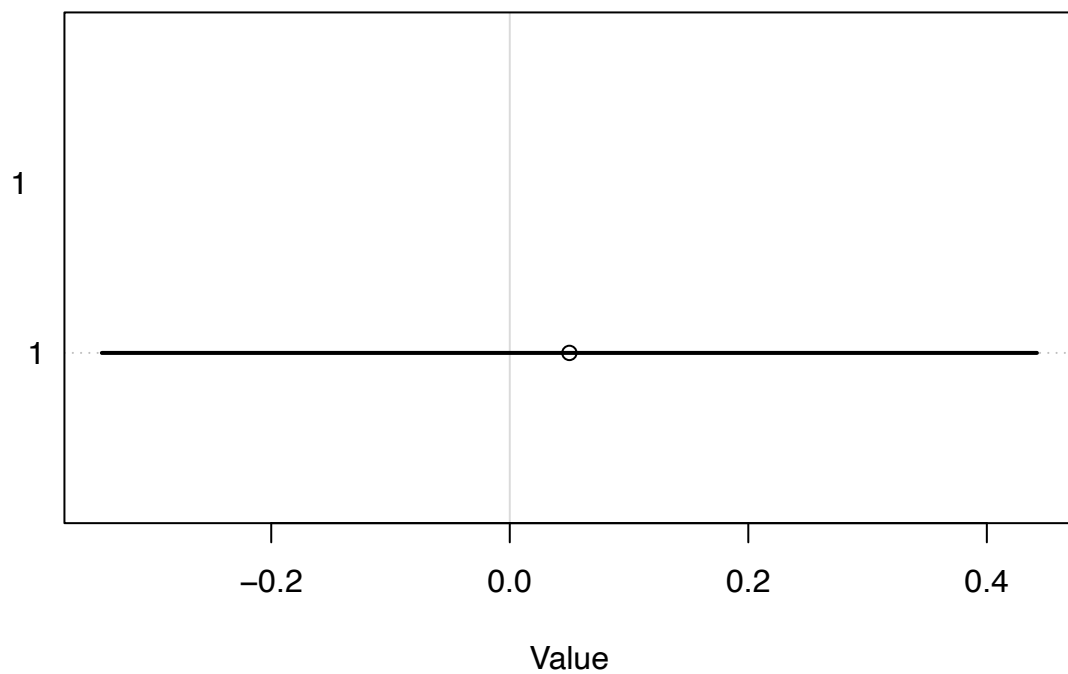
```
plot(coeftab(model.1), par=c("bE","bS"))
```



```
model.1 <- quap(
  alist(
    TS ~ dnorm( mu , sigma ) ,
    mu <- a + bE*E,
    a ~ dnorm( 0 , 0.2 ) ,
    bE ~ dnorm( 0 , 0.2 ) ,
    sigma ~ dexp( 1 )
  ) , data = dn )
precis( model.1 )
```

```
##           mean          sd      5.5%      94.5%
## a      0.1023115 0.2001080 -0.2174998  0.4221228
## bE      0.0450486 0.2000139 -0.2746123  0.3647095
## sigma 43.8916148 2.4144063 40.0329272 47.7503024
```

```
plot( coeftab(model.1), par="bE")
```



We simulate the data, so there is no connection between Test score and exam anxiety. When we run our regression we find that the model estimates the beta coefficient for Exam anxiety to zero.

# Methods 4 – Portfolio Assignment 3

- *Type:* Group assignment
- *Due:* 7 May 2023, 23:59
- *Instructions:* This is an exercise from *Statistical Rethinking* (p. 379). Please edit this file here and add your solutions.

```
#install packages
pacman::p_load(tidyverse, MASS, rethinking, gridExtra)
tinytex::install_tinytex()
```

```
## The directory /usr/local/bin is not writable. I recommend that you make it writ
able. See https://github.com/yihui/tinytex/issues/24 for more info.
```

```
## tlmgr install grffile
```

## Exercise 11H3

The data contained in `library(MASS); data(eagles)` are records of salmon pirating attempts by Bald Eagles in Washington State. See `?eagles` for details. While one eagle feeds, sometimes another will swoop in and try to steal the salmon from it. Call the feeding eagle the “victim” and the thief the “pirate.” Use the available data to build a binomial GLM of successful pirating attempts.

a. Consider the following model:

$$y_i \sim \text{Binomial}(n_i, p_i) \quad (1)$$

$$\log \frac{p_i}{1 - p_i} = \alpha + \beta_P P_i + \beta_V V_i + \beta_A A_i \quad (2)$$

$$\alpha \sim \text{Normal}(0, 1.5) \quad (3)$$

$$\beta_P \sim \text{Normal}(0, 0.5) \quad (4)$$

$$\beta_V \sim \text{Normal}(0, 0.5) \quad (5)$$

$$\beta_A \sim \text{Normal}(0, 0.5) \quad (6)$$

$$(7)$$

where  $y$  is the number of successful attempts,  $n$  is the total number of attempts,  $P$  is a dummy variable indicating whether or not the pirate had large body size,  $V$  is a dummy variable indicating whether or not the victim had large body size, and finally  $A$  is a dummy variable indicating whether or not the pirate was an adult. Fit the model above to the eagles data, using both `quap` and `ulam`. Is the quadratic approximation okay?

#DKA

```
# loading the data set
data <- eagles

#change dummy variables to index variables

data$P <- ifelse( data$P == 'L' , 2 , 1 ) #large is 2

data$A <- ifelse( data$A == 'A' , 2 , 1 ) #adult equals 2

data$V <- ifelse( data$V == 'L' , 2 , 1 ) #if victim is large equal 2
```

## #DKA

```
#fit the model using index variables instead of dummy variables

#logit(p) <- a + bp*P + bv*V + ba*A

eagle_model <- quap(
  alist(
    y ~ dbinom( n , p ) ,
    logit(p) <- a + bp[P] + bv[V] + ba[A] ,
    a ~ dnorm(0, 1.5),
    bp[P] ~ dnorm( 0 , 0.5 ) ,
    bv[V] ~ dnorm( 0 , 0.5 ) ,
    ba[A] ~ dnorm( 0 , 0.5 )
  ) , data=data )

precis( eagle_model , depth=2 , pars=c("bp","bv","ba") )
```

```
##           mean          sd      5.5%      94.5%
## bp[1] -1.0131911 0.3917739 -1.6393215 -0.3870607
## bp[2]  1.0757633 0.3908947  0.4510382  1.7004885
## bv[1]  1.1258310 0.3992853  0.4876960  1.7639659
## bv[2] -1.0632523 0.3956189 -1.6955277 -0.4309770
## ba[1] -0.3536793 0.3896532 -0.9764204  0.2690619
## ba[2]  0.4162564 0.3895095 -0.2062549  1.0387678
```

```
#check the quadratic estimation, did it turn out alright?
```

## #LA

```
#get the estimates in predictor space and plot the estimates,
#to get a sense of what the model has picked up on
```

```
post <- extract.samples(eagle_model)


estimates_intercept <- inv_logit( post$a )

estimates_p <- inv_logit( post$bp )



estimates_v <- inv_logit( post$bv )

estimates_a <- inv_logit( post$ba )



precis(as.data.frame(estimates_intercept))
```

```
##                mean          sd      5.5%      94.5% histogram
## estimates_intercept 0.6270966 0.1308053 0.3987601 0.8183531 
```



```
precis( as.data.frame(estimates_p))
```

```
##          mean          sd      5.5%      94.5%      histogram
## V1 0.2727480 0.07626250 0.1647086 0.4054909 
## V2 0.7397182 0.07394249 0.6104529 0.8452769 
```

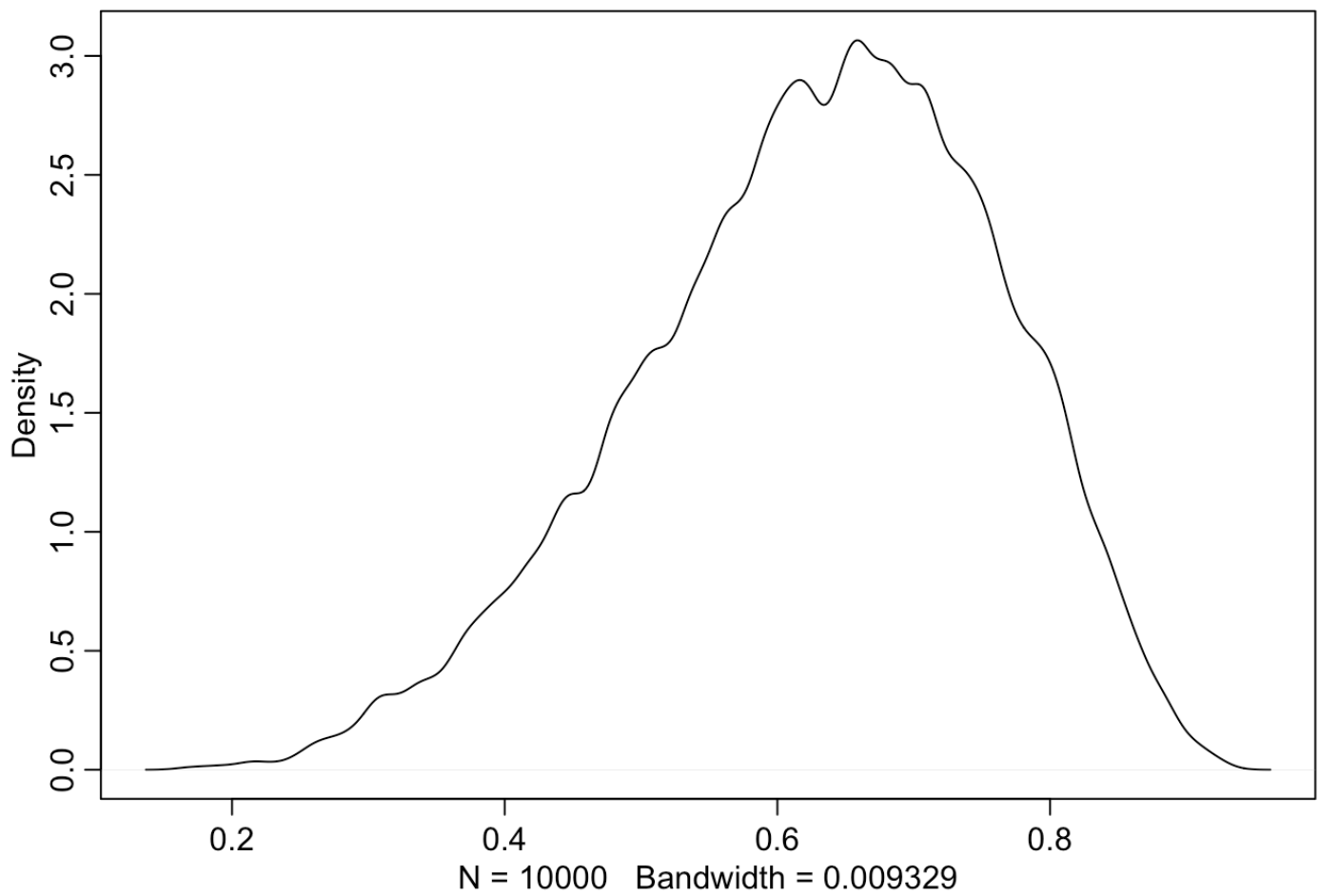
```
precis( as.data.frame(estimates_v))
```

```
##          mean          sd      5.5%      94.5%      histogram
## V1 0.7484108 0.07375767 0.6207327 0.8548932 
## V2 0.2645666 0.07544089 0.1534163 0.3938756 
```

```
precis( as.data.frame(estimates_a))
```

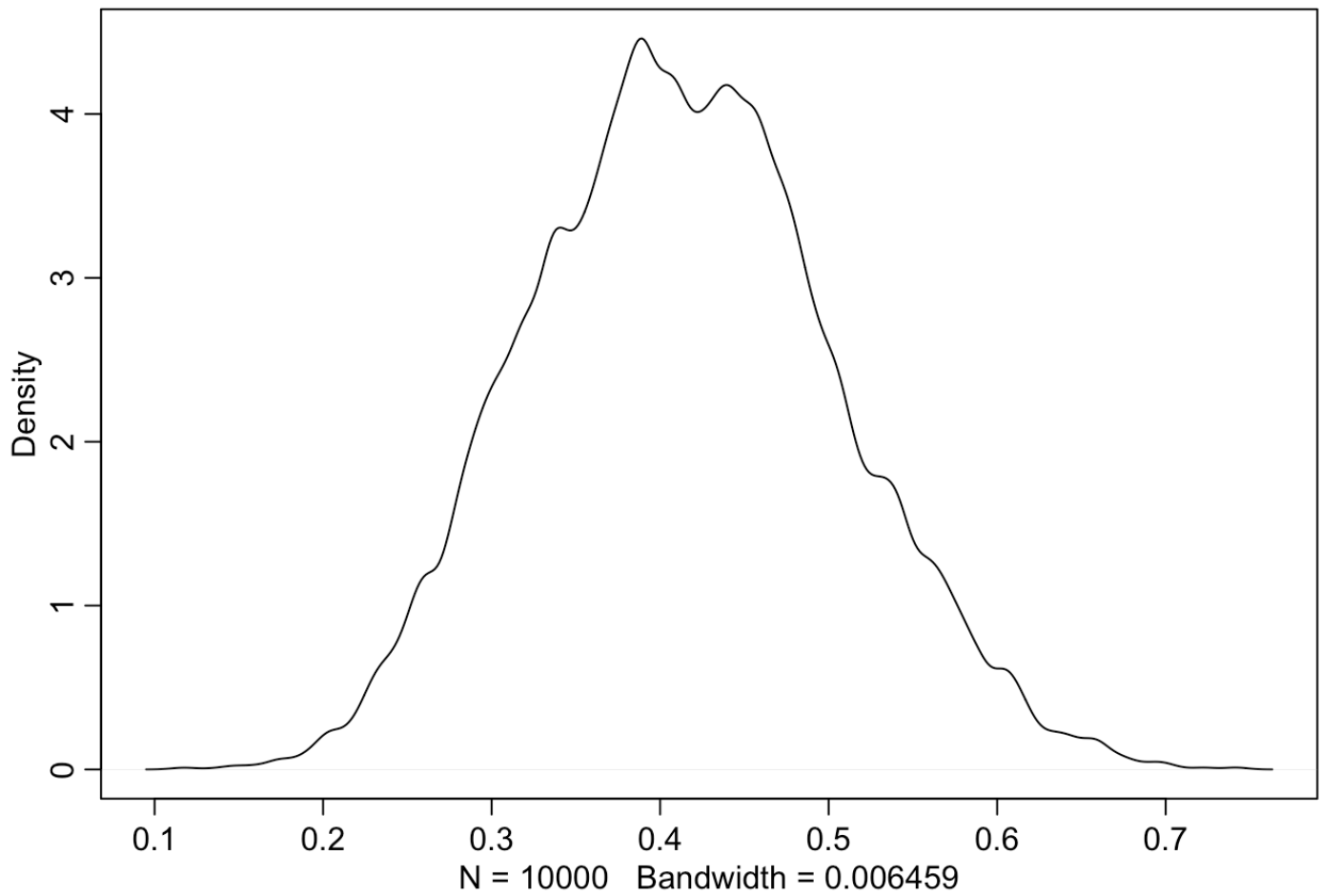
```
##          mean          sd      5.5%      94.5%      histogram
## V1 0.4136724 0.09056848 0.2735094 0.5644058 
## V2 0.5986924 0.08990812 0.4504870 0.7362263 
```

```
dens(estimates_intercept)
```

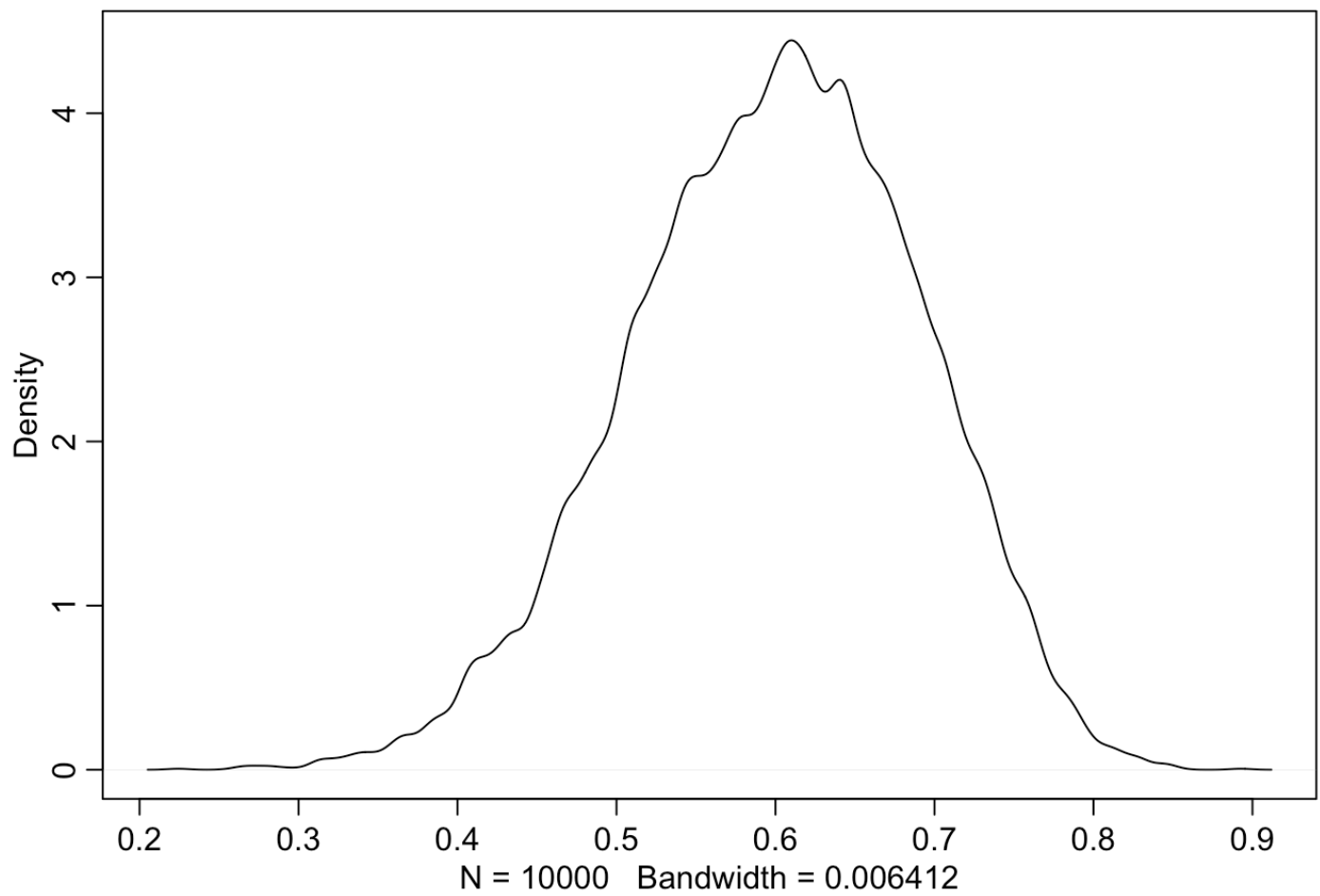


```
dens(estimates_a[,1])
```

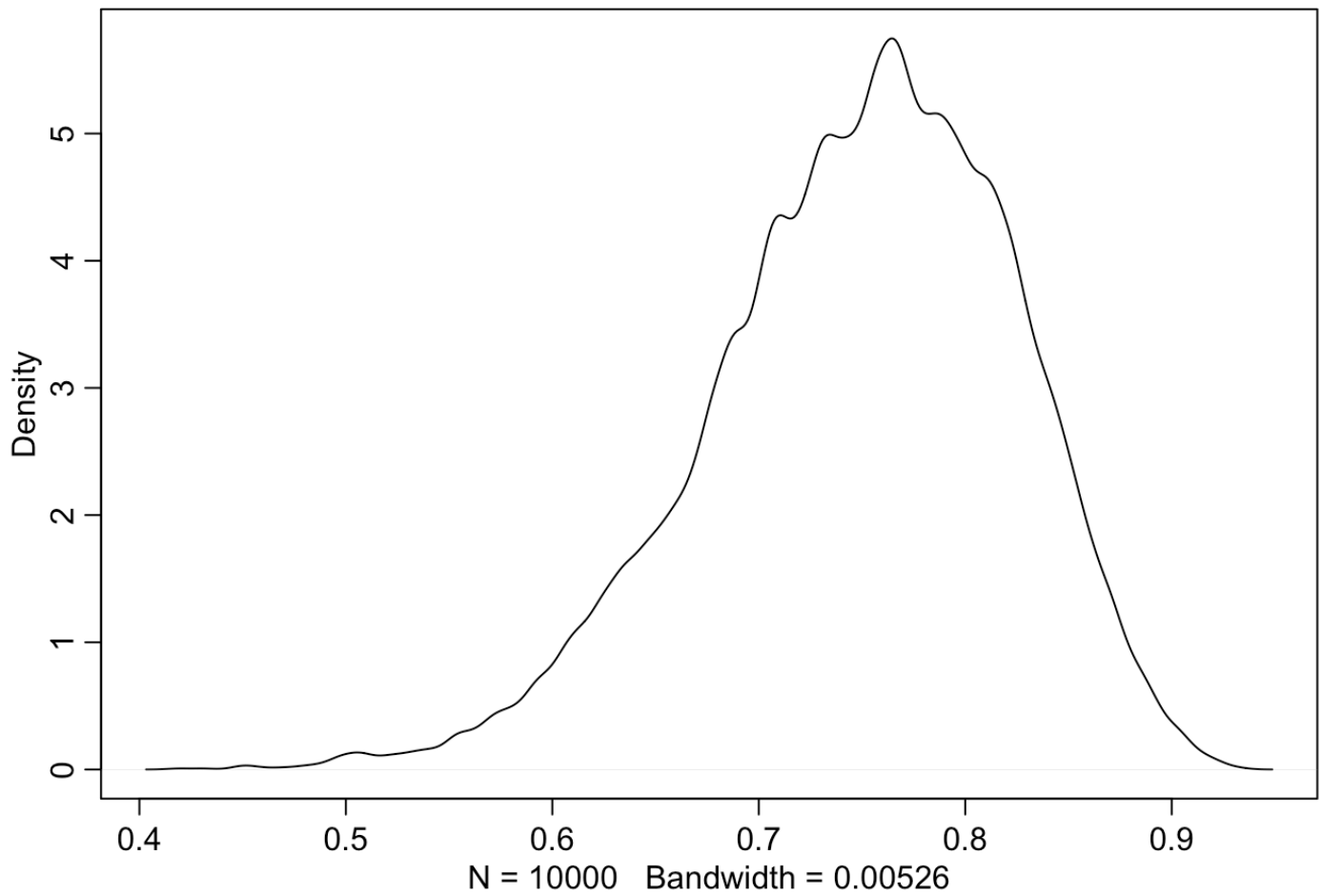




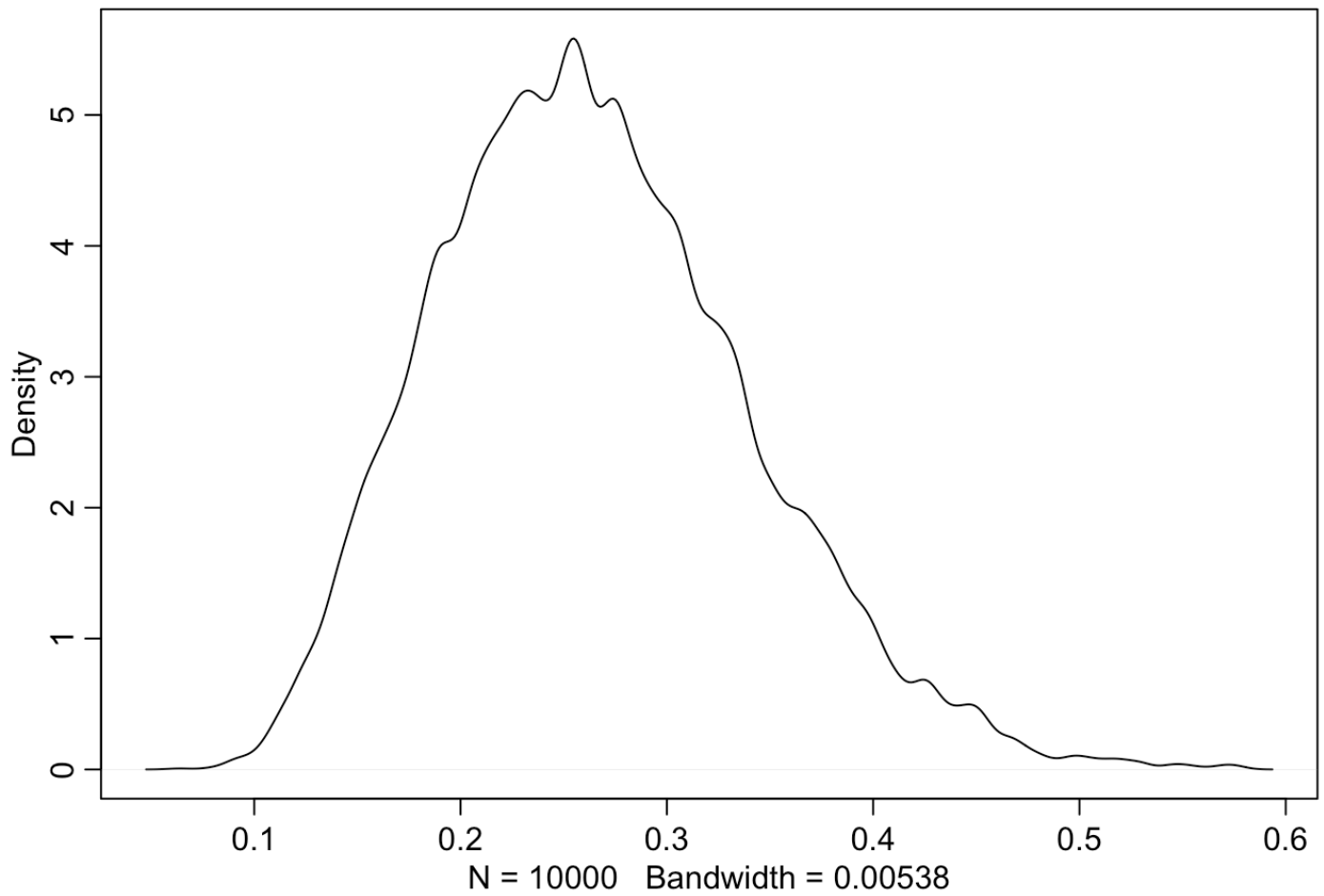
```
dens(estimates_a[,2])
```



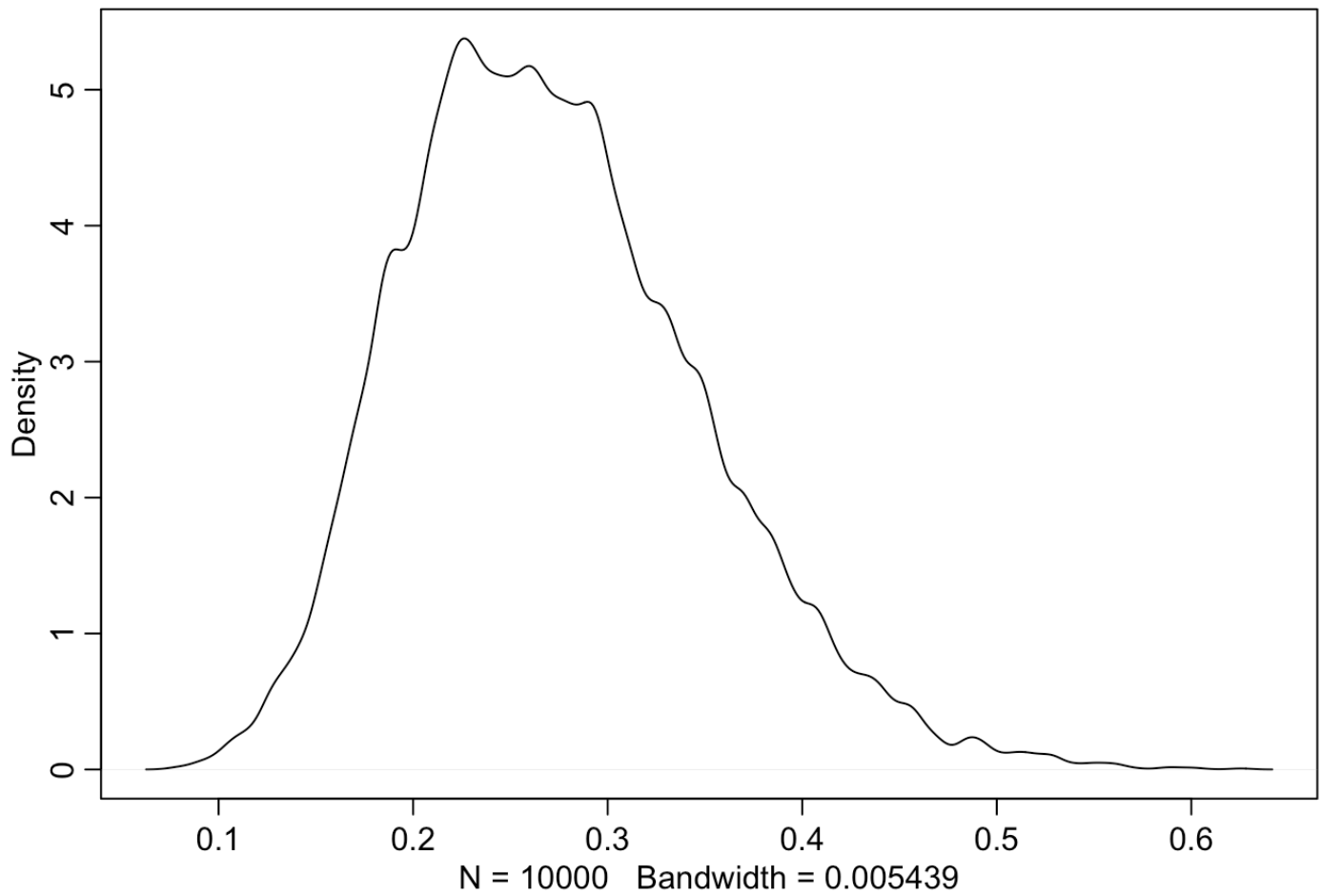
```
dens(estimates_v[,1])
```



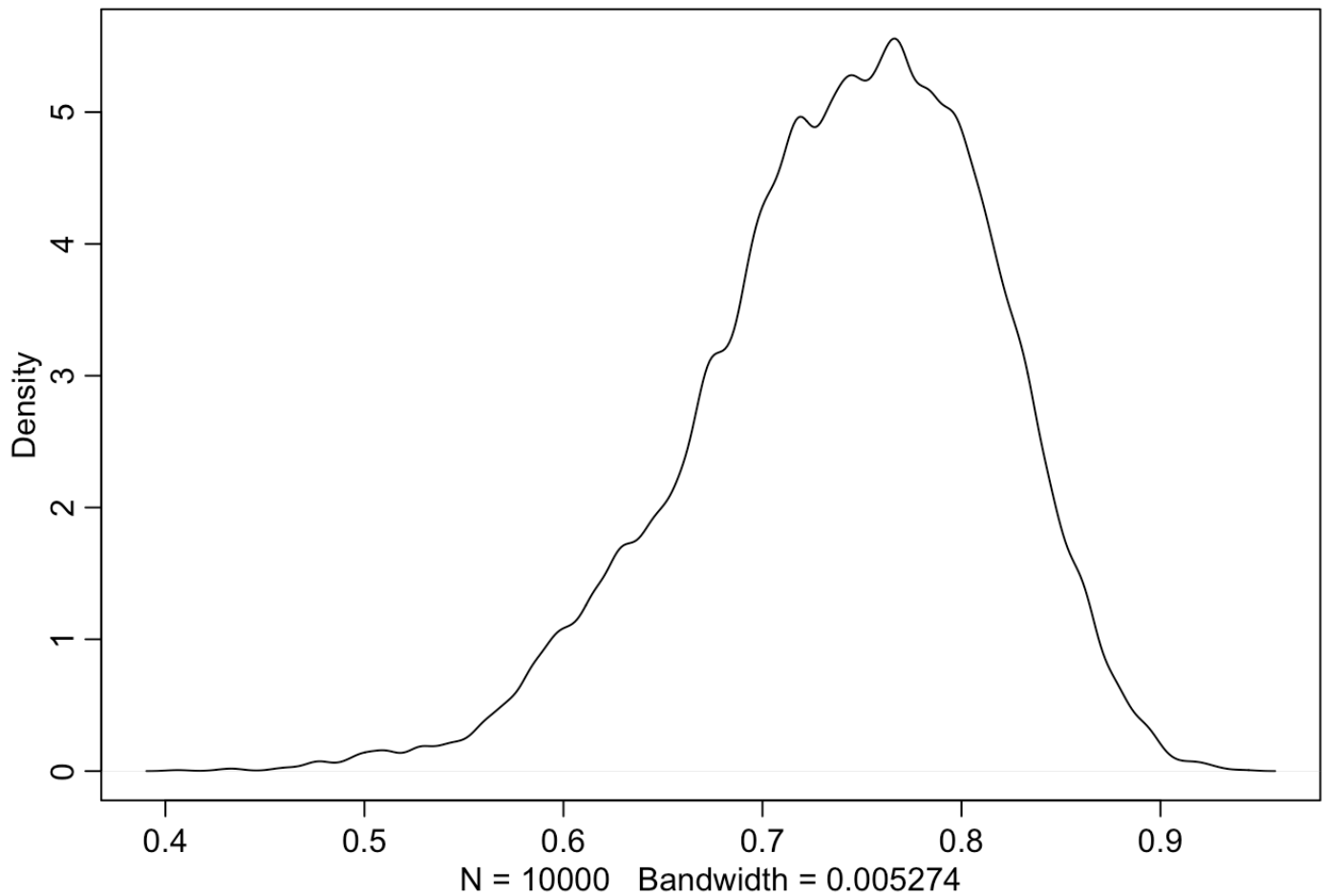
```
dens(estimates_v[,2])
```



```
dens(estimates_p[,1])
```



```
dens(estimates_p[,2])
```



#PM

```
eagle_model_2 <- ulam(
  alist(
    y ~ dbinom( n , p ) ,
    logit(p) <- a + bp[P] + bv[V] + ba[A] ,
    a ~ dnorm(0, 1.5),
    bp[P] ~ dnorm( 0 , 0.5 ) ,
    bv[V] ~ dnorm( 0 , 0.5 ) ,
    ba[A] ~ dnorm( 0 , 0.5 )
  ) , data=data, chains = 4, log_lik = TRUE )
```

```
## Warning in '/var/folders/hf/2cjsx77xd24b0lrtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b7e8e7652.stan', line 2, column 4: Declaration
##      of arrays by placing brackets after a variable name is deprecated and
##      will be removed in Stan 2.32.0. Instead use the array keyword before the
##      type. This can be changed automatically using the auto-format flag to
##      stanc
## Warning in '/var/folders/hf/2cjsx77xd24b0lrtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b7e8e7652.stan', line 3, column 4: Declaration
##      of arrays by placing brackets after a variable name is deprecated and
##      will be removed in Stan 2.32.0. Instead use the array keyword before the
##      type. This can be changed automatically using the auto-format flag to
##      stanc
## Warning in '/var/folders/hf/2cjsx77xd24b0lrtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b7e8e7652.stan', line 4, column 4: Declaration
##      of arrays by placing brackets after a variable name is deprecated and
##      will be removed in Stan 2.32.0. Instead use the array keyword before the
##      type. This can be changed automatically using the auto-format flag to
##      stanc
## Warning in '/var/folders/hf/2cjsx77xd24b0lrtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b7e8e7652.stan', line 5, column 4: Declaration
##      of arrays by placing brackets after a variable name is deprecated and
##      will be removed in Stan 2.32.0. Instead use the array keyword before the
##      type. This can be changed automatically using the auto-format flag to
##      stanc
## Warning in '/var/folders/hf/2cjsx77xd24b0lrtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b7e8e7652.stan', line 6, column 4: Declaration
##      of arrays by placing brackets after a variable name is deprecated and
##      will be removed in Stan 2.32.0. Instead use the array keyword before the
##      type. This can be changed automatically using the auto-format flag to
##      stanc
```

```
## Running MCMC with 4 sequential 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 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 0.0 seconds.
## 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 2 Iteration: 800 / 1000 [ 80%] (Sampling)
## Chain 2 Iteration: 900 / 1000 [ 90%] (Sampling)
## Chain 2 Iteration: 1000 / 1000 [100%] (Sampling)
## Chain 2 finished in 0.1 seconds.
## 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 3 Iteration: 900 / 1000 [ 90%] (Sampling)
## Chain 3 Iteration: 1000 / 1000 [100%] (Sampling)
## Chain 3 finished in 0.0 seconds.
## 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 4 Iteration: 900 / 1000 [ 90%] (Sampling)
## Chain 4 Iteration: 1000 / 1000 [100%] (Sampling)
## Chain 4 finished in 0.1 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.0 seconds.
## Total execution time: 0.6 seconds.

```

```

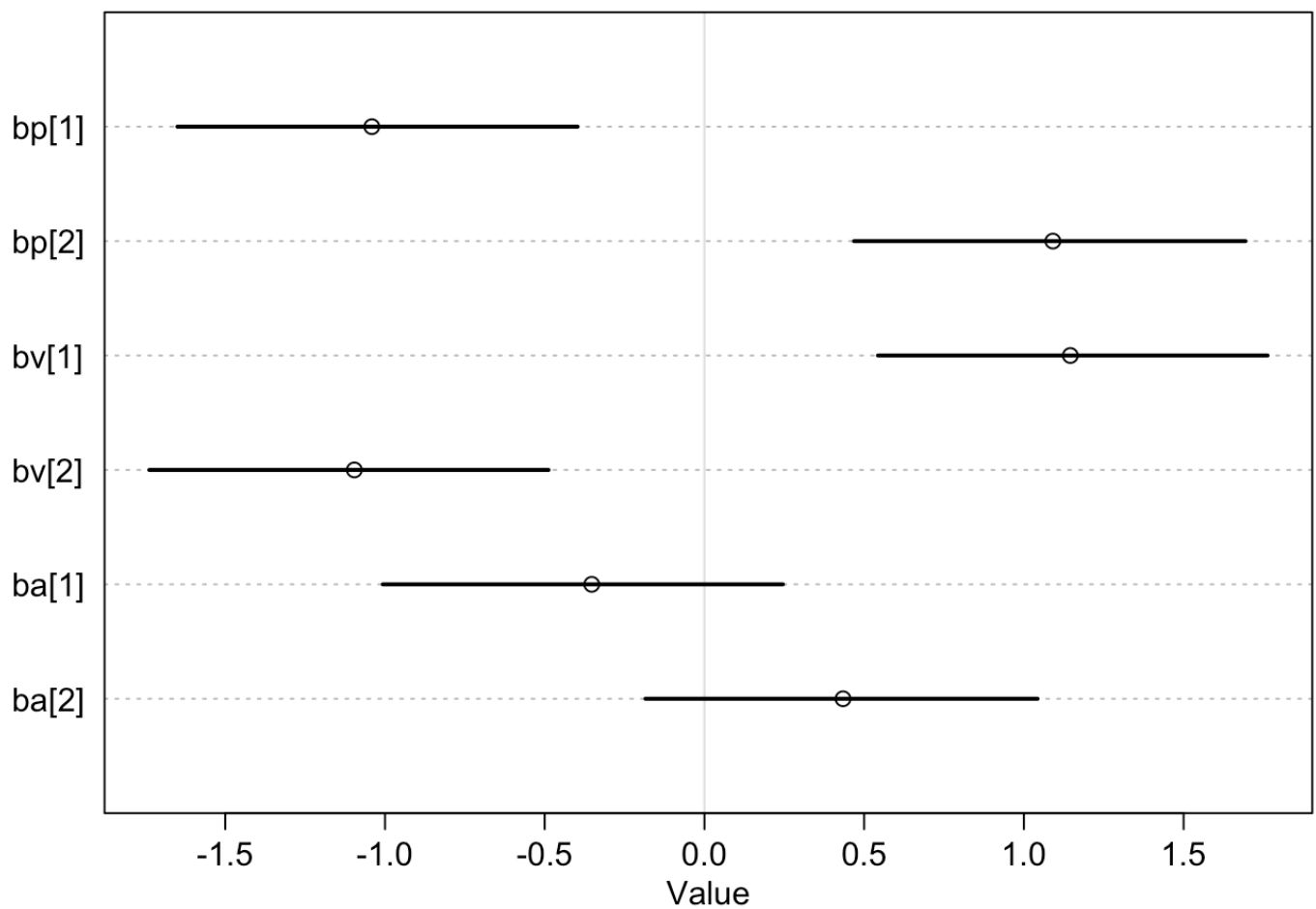
precis (eagle_model_2, depth = 2)

```



```
##          mean      sd      5.5%      94.5%    n_eff    Rhat4
## a      0.5834648 0.6000448 -0.3669407  1.5168820  967.4784 0.9999149
## bp[1] -1.0410484 0.3907042 -1.6498106 -0.3969870 1422.4337 0.9987770
## bp[2]  1.0907498 0.3839717  0.4682411  1.6938453 1245.3391 1.0002983
## bv[1]  1.1454259 0.3813302  0.5434165  1.7627229 1349.1281 0.9994017
## bv[2] -1.0958396 0.3942837 -1.7377337 -0.4880318 1317.4441 0.9992161
## ba[1] -0.3528038 0.3838388 -1.0068547  0.2457749 1317.9252 1.0023742
## ba[2]  0.4338391 0.3819855 -0.1846088  1.0427218 1279.4832 1.0002493
```

```
plot( precis( eagle_model_2 , depth=2 , pars=c("bp","bv","ba") ))
```



#PM

```
#get the estimates in predictor space



post_2 <- extract.samples(eagle_model_2)

estimates_p_2 <- inv_logit( post_2$bp )



estimates_v_2 <- inv_logit( post_2$bv )

estimates_a_2 <- inv_logit( post_2$ba )



precis( as.data.frame(estimates_p_2))
```

```
##           mean          sd      5.5%      94.5%    histogram
## V1 0.2676374 0.07535072 0.1611346 0.4020365 
## V2 0.7419201 0.07176726 0.6149674 0.8447292 
```

```
precis( as.data.frame(estimates_v_2))
```

```
##           mean          sd      5.5%      94.5%    histogram
## V1 0.7520580 0.06950333 0.6326068 0.8535503 
## V2 0.2574464 0.07376977 0.1496010 0.3803573 
```

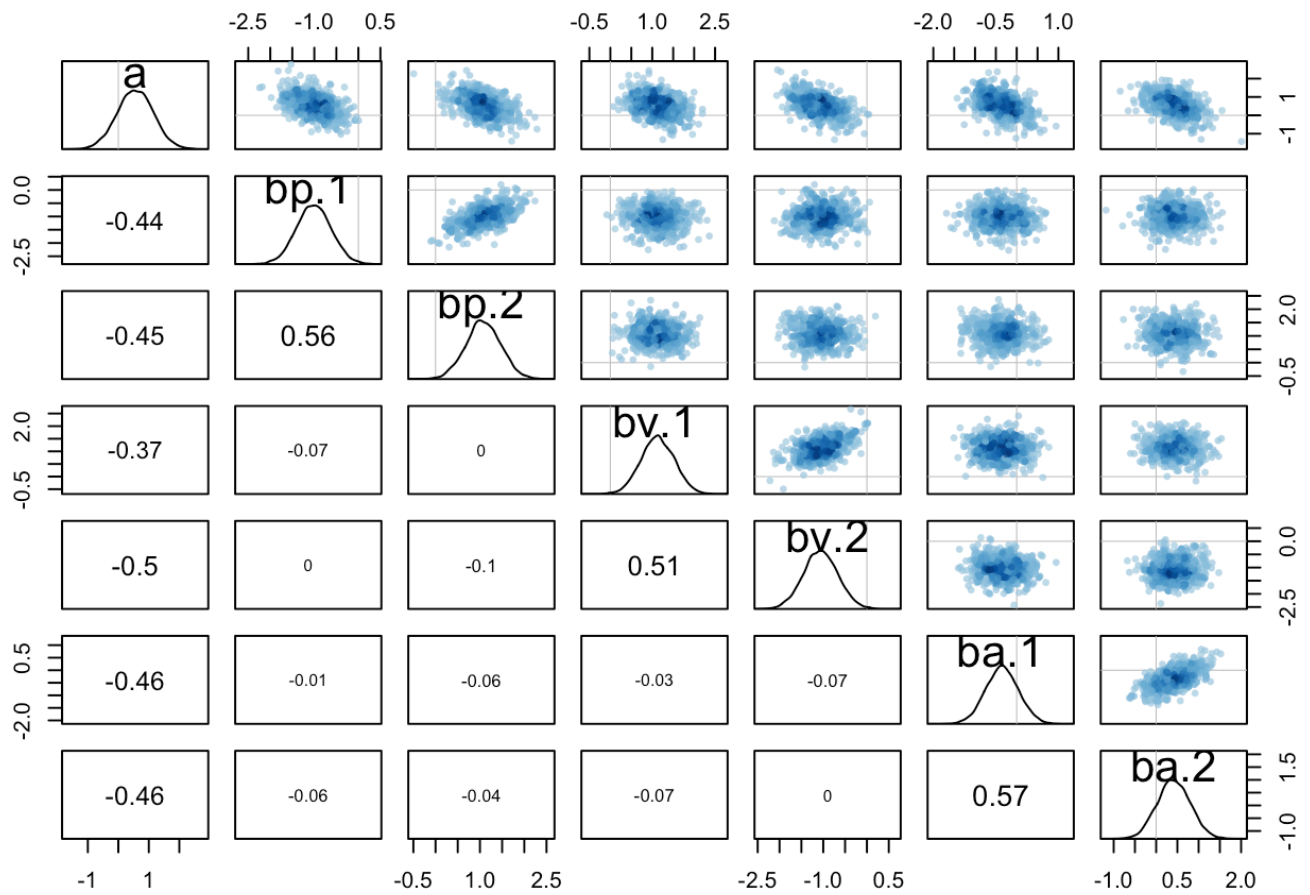
```
precis( as.data.frame(estimates_a_2))
```

```
##           mean          sd      5.5%      94.5%    histogram
## V1 0.4156922 0.08990270 0.2675958 0.5611363 
## V2 0.6032514 0.08830955 0.4539784 0.7393748 
```

- b. Now interpret the estimates. If the quadratic approximation turned out okay, then it's okay to use the quap estimates. Otherwise stick to `ulam` estimates. Then plot the posterior predictions. Compute and display both (1) the predicted probability of success and its 89% interval for each row (i) in the data, as well as (2) the predicted success count and its 89% interval. What different information does each type of posterior prediction provide?

#KM

```
# After the comparison of the outputs from both quap and ulam, we can see that the
y are not too different. That is a sign that quap worked out in a nice way and we
can most likely stick with it. We used the pairs() to check the the quap estimates
.
pairs(eagle_model)
```



```
#The new correct one!!!!!!!
data$success_rate <- data$y / data$n
y <- sim(eagle_model)
post <- extract.samples(eagle_model)
mu <- link(eagle_model, data = data)
mu_mean <- apply(mu, 2, mean)
mu_PI <- apply(mu, 2, PI, prob = 0.89)

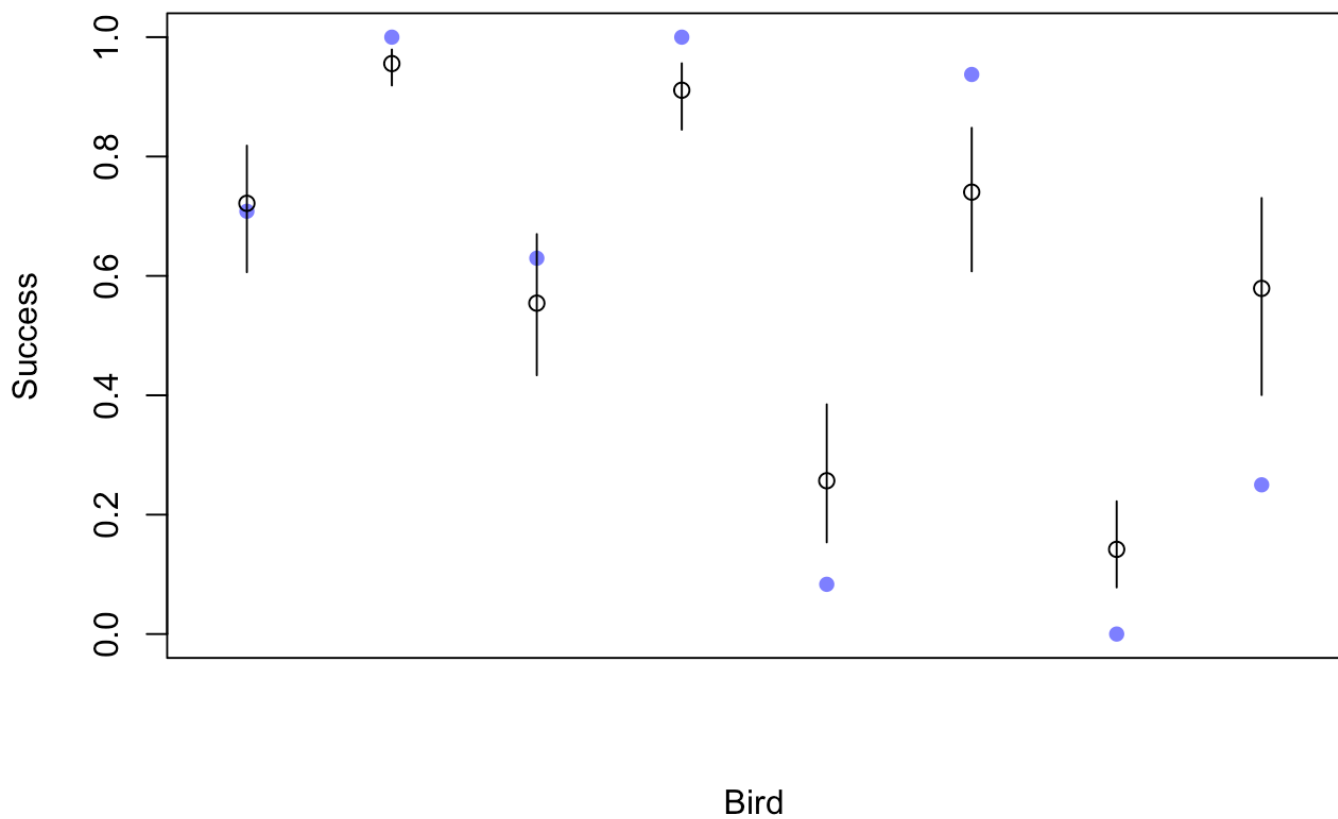
mu_mean_count <- data$n * mu_mean

mu_mean_count <- round(mu_mean_count)

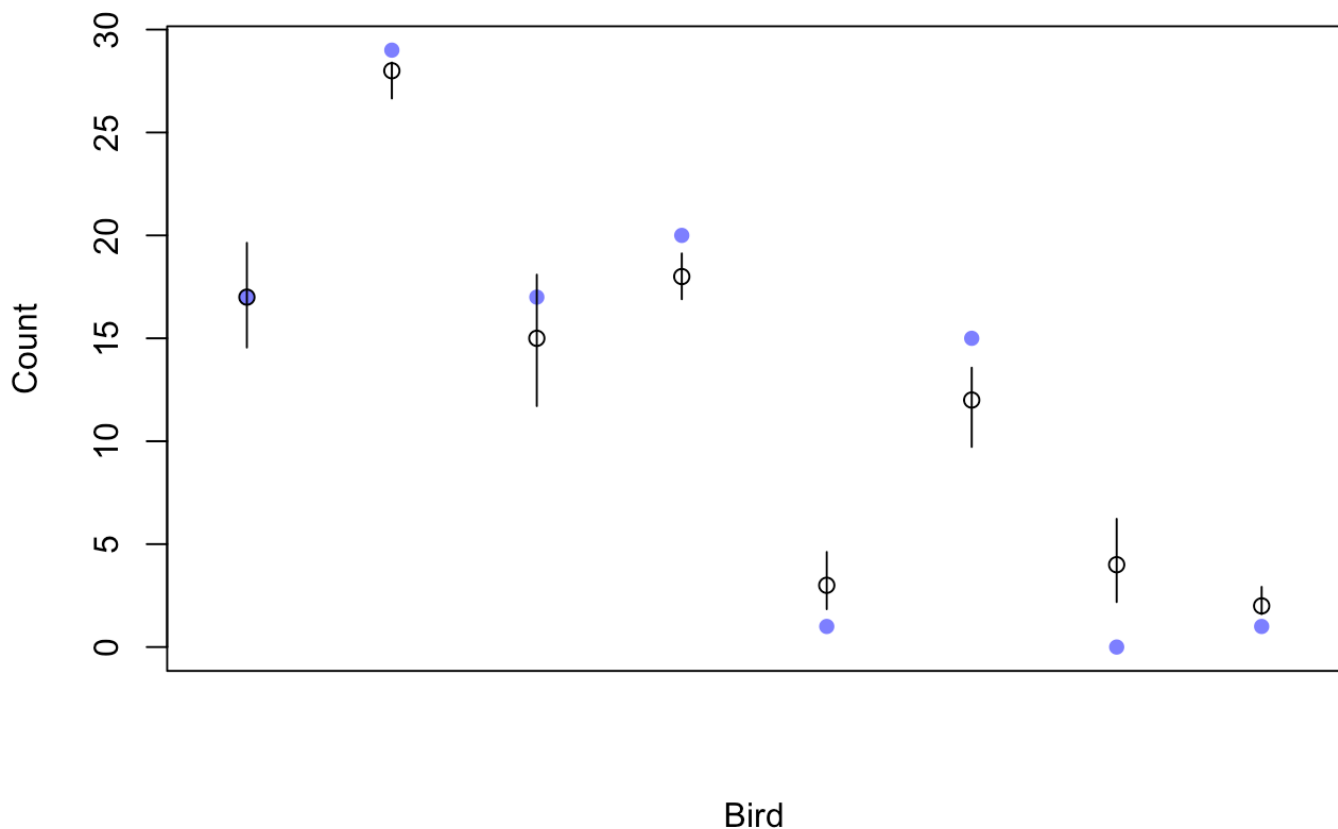
mu_PI_count <- data.frame(matrix(NA, nrow = nrow(mu_PI), ncol = ncol(mu_PI)))

for (i in 1:nrow(mu_PI_count)) {
  for (j in 1:ncol(mu_PI_count)) {
    mu_PI_count[i,j] = mu_PI[i,j]*data[j,2]
  }
}

# plot raw proportions success for each case
plot( data$success_rate , col=rainbow ,
      ylab="Success" , xlab="Bird" , xaxt="n" ,
      xlim=c(0.75,8.25) , pch=16 )
points( 1:8 , mu_mean )
for ( i in 1:8 ) lines( c(i,i) , mu_PI[,i] )
```



```
plot( data$y, col=range(2,
  ylab="Count" , xlab="Bird" , xaxt="n" ,
  xlim=c(0.75,8.25) , pch=16 )
points( 1:8 , mu_mean_count)
for (i in 1:8) lines( c(i,i) , mu_PI_count[,i])
```



We can see that the model makes fairly reasonable predictions. However, it has a hard time making sense of some of the birds that have very few attempts. Furthermore, the model seems reluctant to output extreme values, as for example the birds with either 0 or 100% success rate. Generally we find that, large body size of Pirate improves success rates, and small body size of victim improves success rate. The age of the bird also has a positive relationship with the success rate. We can be fairly certain about these claims, given none of the 89% intervals, relating to our parameter estimates, include zero. Given this information we will try to fit a new model including interaction effects.

(c) Now try to improve the model. Consider an interaction between the pirate's size and age (immature or adult). Compare this model to the previous one, using WAIC. Interpret.

#NV

```

set.seed(10)
eagle_model3 <- ulam(
  alist(
    y ~ dbinom(n, p),
    # setting up the interaction by making a matrix of the four combinations t
hat P and A can assume. We use the model from earlier and add the following intera
ction between age and size of the pirate a[P, A].
    logit(p) <- ba[A] + a[P,A] + bv[V] + bp[P],
    #a ~ dnorm(0, 1.5),
    ba[A] ~ dnorm(0, 0.5) ,
    bv[V] ~ dnorm(0, 0.5) ,
    bp[P] ~ dnorm(0, 0.5) ,
    matrix[P,A]:a ~ dnorm(0, 0.5)),
  data = data, chains = 4, cores = 4, log_lik = TRUE)

```

```

## Warning in '/var/folders/hf/2cjsx77xd24b01rtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b4de84ba8.stan', line 3, column 4: Declaration
## of arrays by placing brackets after a variable name is deprecated and
## will be removed in Stan 2.32.0. Instead use the array keyword before the
## type. This can be changed automatically using the auto-format flag to
## stanc
## Warning in '/var/folders/hf/2cjsx77xd24b01rtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b4de84ba8.stan', line 4, column 4: Declaration
## of arrays by placing brackets after a variable name is deprecated and
## will be removed in Stan 2.32.0. Instead use the array keyword before the
## type. This can be changed automatically using the auto-format flag to
## stanc
## Warning in '/var/folders/hf/2cjsx77xd24b01rtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b4de84ba8.stan', line 5, column 4: Declaration
## of arrays by placing brackets after a variable name is deprecated and
## will be removed in Stan 2.32.0. Instead use the array keyword before the
## type. This can be changed automatically using the auto-format flag to
## stanc
## Warning in '/var/folders/hf/2cjsx77xd24b01rtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b4de84ba8.stan', line 6, column 4: Declaration
## of arrays by placing brackets after a variable name is deprecated and
## will be removed in Stan 2.32.0. Instead use the array keyword before the
## type. This can be changed automatically using the auto-format flag to
## stanc
## Warning in '/var/folders/hf/2cjsx77xd24b01rtsfd8v4mmh0000gn/T/Rtmp5bh3Ok/model-8
30b4de84ba8.stan', line 7, column 4: Declaration
## of arrays by placing brackets after a variable name is deprecated and
## will be removed in Stan 2.32.0. Instead use the array keyword before the
## type. This can be changed automatically using the auto-format flag to
## stanc

```

```

## 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 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 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 2 Iteration: 800 / 1000 [ 80%] (Sampling)
## Chain 2 Iteration: 900 / 1000 [ 90%] (Sampling)
## Chain 2 Iteration: 1000 / 1000 [100%] (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 3 Iteration: 900 / 1000 [ 90%] (Sampling)
## Chain 3 Iteration: 1000 / 1000 [100%] (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 4 Iteration: 900 / 1000 [ 90%] (Sampling)
## Chain 4 Iteration: 1000 / 1000 [100%] (Sampling)
## Chain 1 finished in 0.0 seconds.
## Chain 2 finished in 0.0 seconds.
```



```
## Chain 3 finished in 0.0 seconds.
## Chain 4 finished in 0.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.0 seconds.
## Total execution time: 0.2 seconds.
```

```
precis( eagle_model3, depth = 3)
```

```
##              mean      sd      5.5%      94.5%    n_eff    Rhat4
## ba[1] -0.1965642 0.3867645 -0.78624722  0.430964455 1936.566 0.9996236
## ba[2]  0.4892420 0.3878831 -0.11278528  1.116219350 1815.814 1.0023120
## bv[1]  1.2700146 0.3829939  0.65267266  1.871995250 2216.365 0.9981787
## bv[2] -0.9296436 0.3603894 -1.50680030 -0.347356735 1996.898 0.9990720
## bp[1] -0.6449386 0.3962122 -1.28793410 -0.009884321 1888.209 1.0007875
## bp[2]  0.9588270 0.3869870  0.34903755  1.598666800 1955.233 0.9983101
## a[1,1] -0.7333052 0.4382262 -1.43920375 -0.034866386 2223.386 1.0020497
## a[1,2]  0.0986747 0.4272522 -0.57065656  0.796795300 1940.015 1.0005641
## a[2,1]  0.5844070 0.3917096 -0.02969234  1.201499800 1959.615 0.9998582
## a[2,2]  0.3732506 0.4038282 -0.30383331  1.004764100 2018.467 0.9998473
```

```
data$success_rate <- data$y / data$n
y <- sim(eagle_model3)
post <- extract.samples(eagle_model3)
mu <- link(eagle_model3, data = data)
mu_mean <- apply(mu, 2, mean)
mu_PI <- apply(mu, 2, PI, prob = 0.89)

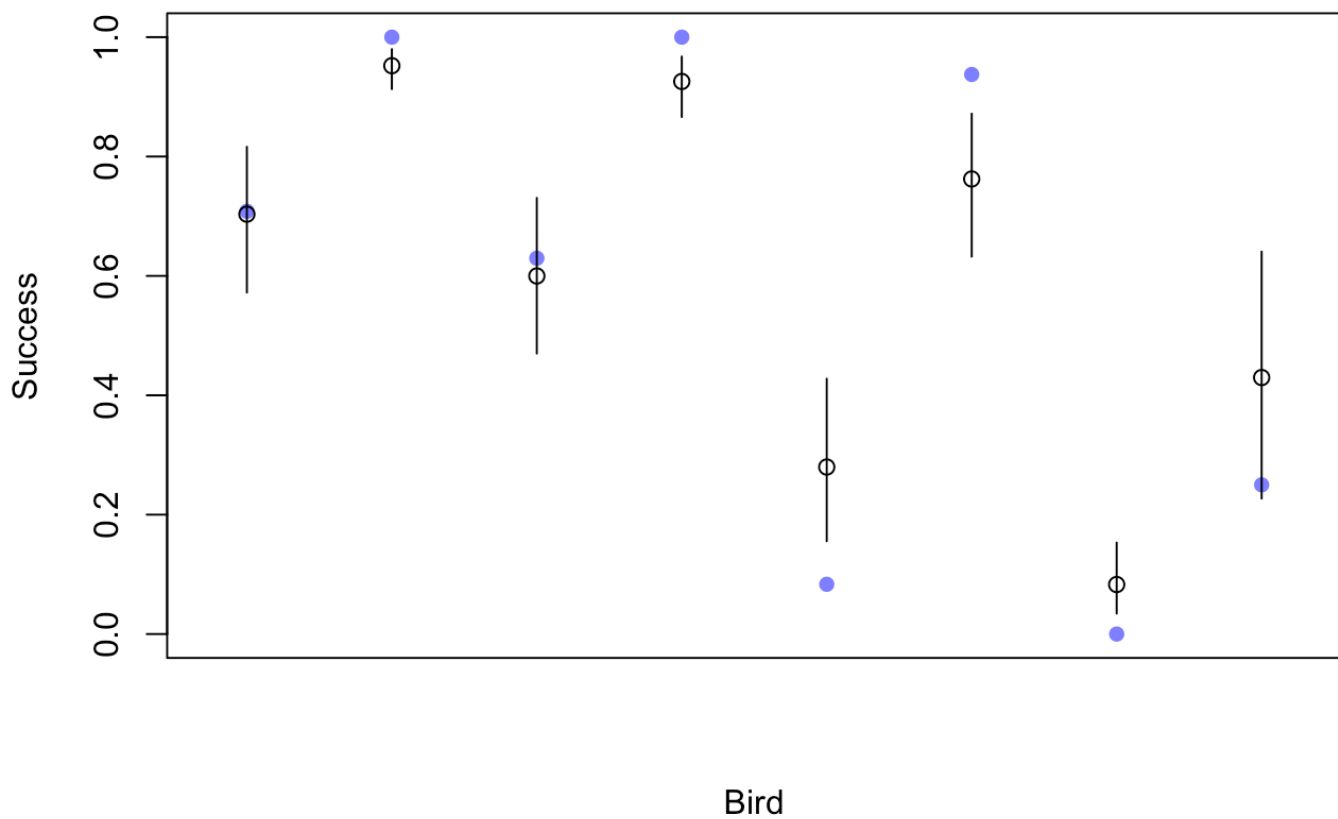
mu_mean_count <- data$n * mu_mean

mu_mean_count <- round(mu_mean_count)

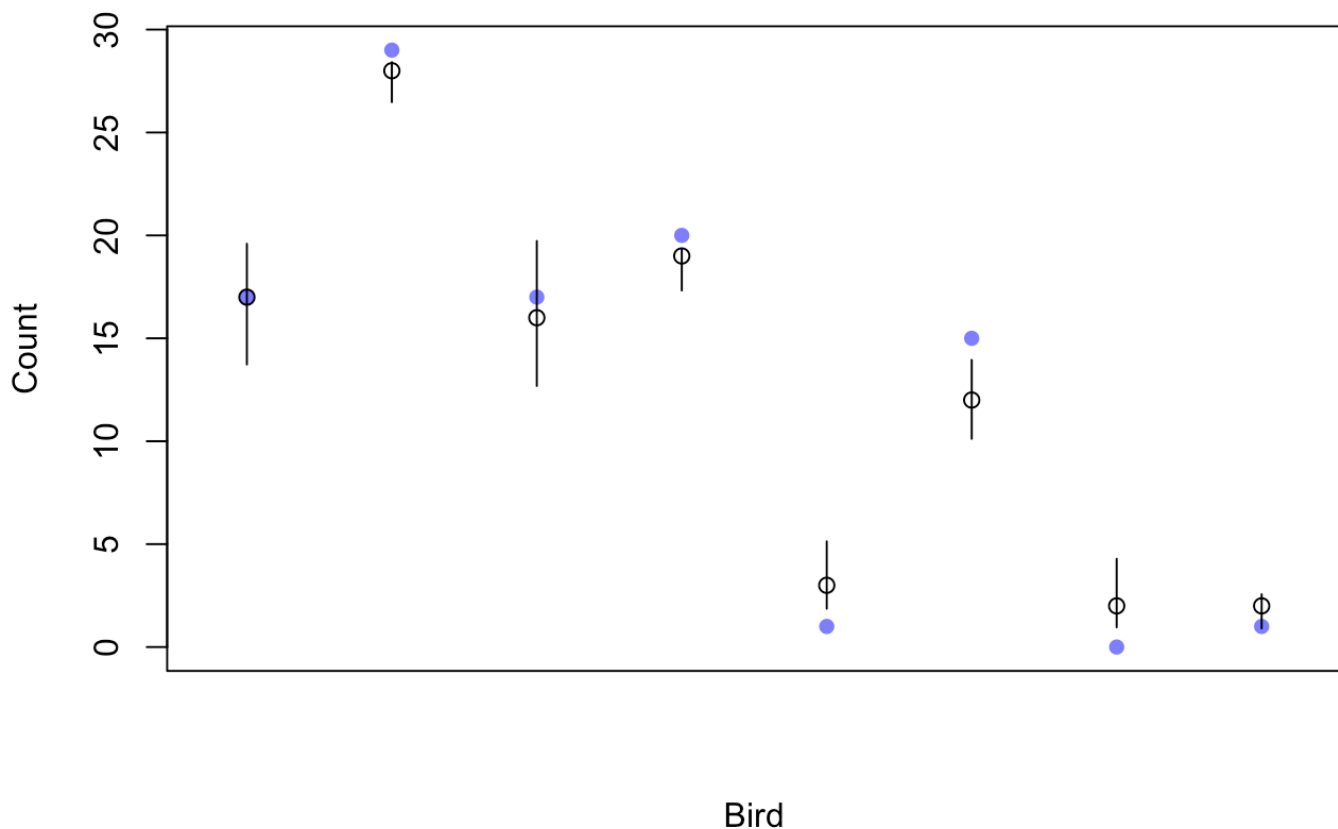
mu_PI_count <- data.frame(matrix(NA, nrow = nrow(mu_PI), ncol = ncol(mu_PI)))

for (i in 1:nrow(mu_PI_count)) {
  for (j in 1:ncol(mu_PI_count)) {
    mu_PI_count[i,j] = mu_PI[i,j]*data[j,2]
  }
}

# plot raw proportions success for each case
plot( data$success_rate , col=range12 ,
      ylab="Success" , xlab="Bird" , xaxt="n" ,
      xlim=c(0.75,8.25) , pch=16 )
points( 1:8 , mu_mean )
for ( i in 1:8 ) lines( c(i,i) , mu_PI[,i] )
```



```
plot( data$y, col=range(2,
  ylab="Count" , xlab="Bird" , xaxt="n" ,
  xlim=c(0.75,8.25) , pch=16 )
points( 1:8 , mu_mean_count)
for (i in 1:8) lines( c(i,i) , mu_PI_count[,i])
```



```
compare(eagle_model, eagle_model3)
```

```
## Warning in compare(eagle_model, eagle_model3): Not all model fits of same class
.
## This is usually a bad idea, because it implies they were fit by different algo-
rithms.
## Check yourself, before you wreck yourself.
```

| ##              | WAIC     | SE       | dWAIC    | dSE      | pWAIC    | weight     |
|-----------------|----------|----------|----------|----------|----------|------------|
| ## eagle_model3 | 40.29318 | 4.747018 | 0.000000 | NA       | 5.749695 | 0.96071862 |
| ## eagle_model  | 46.68704 | 7.369519 | 6.393862 | 4.770409 | 6.571742 | 0.03928138 |

```
#Output: Main model WAIC: 47.3, Interaction model WAIC: 40.7
```

The WAIC score improved after adding the interaction effect to the model. This tells us that there is valuable information in the interaction between the size and age of the pirate, though it does not imply which combinations of these variables lead to more successful attempts.