19.5/20
Congratulations!

Worksheet 05

Jan Hohenheim

```
library(tidyverse)
library(rjags)
library(coda)
library(bayesmeta)
library(pCalibrate)
library(ggthemes)
library(latex2exp)
library(glue)

theme_set(theme_solarized_2())
primary_colour <- scale_color_solarized()$palette(1)

# set seed
set.seed(42)
```

16/16

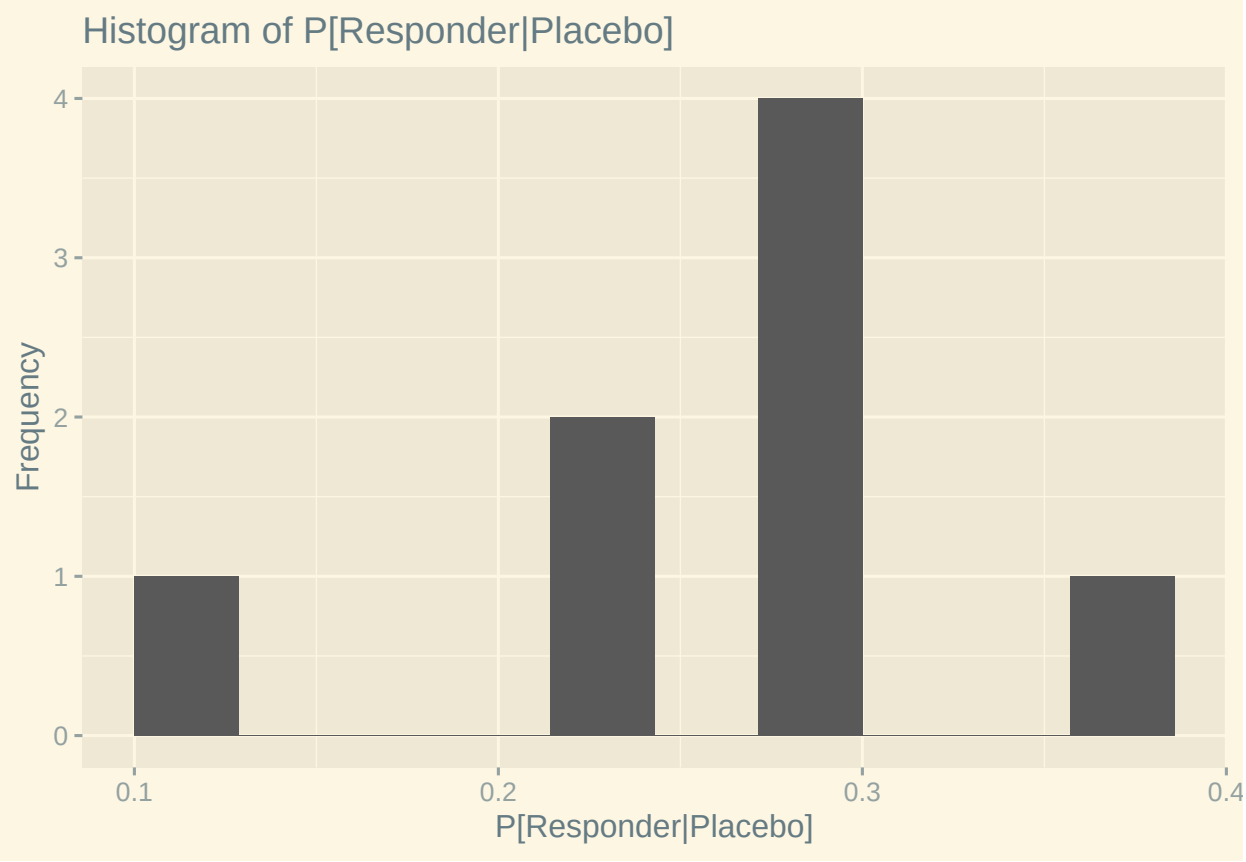
Exercise 1

a)

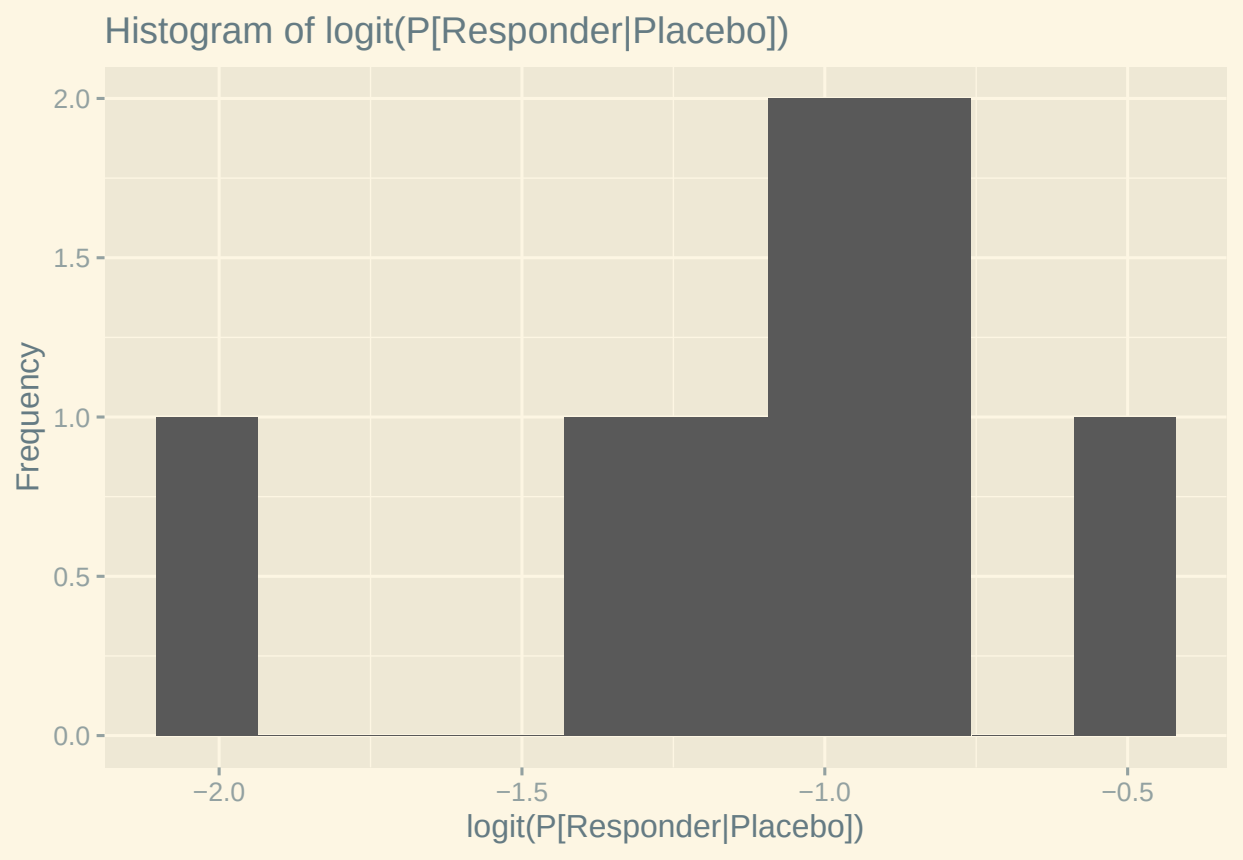
```
pl_total <- c(107,44,51,39,139,20,78,35)
pl_case <- c(23,12,19,9,39,6,9,10)
pl_p <- pl_case/pl_total
pl_logit <- log(pl_case/(pl_total - pl_case))
pl_se <- sqrt((1/pl_case) + (1/(pl_total - pl_case)))

tibble(p = pl_p) |>
  ggplot(aes(x = p)) +
  geom_histogram(bins = 10) +
  labs(title = "Histogram of P[Responder|Placebo]", x = "P[Responder|Placebo]", y = "Frequency") +
  scale_color_solarized()
```

If you want to check normality, a QQ-plot might be more convenient for such a small sample size.




```
tibble(logit = pl_logit) |>
  ggplot(aes(x = logit)) +
  geom_histogram(bins = 10) +
  labs(title = "Histogram of logit(P[Responder|Placebo])", x = "logit(P[Responder|Placebo])", y = "Freq
```



b)

```
pl1_modelString <- "
model {
  for(i in 1:length(y)){
    y[i] ~ dnorm(theta[i], prec_s[i]);
    theta[i] ~ dnorm(mu, prec_tau);
  }
  theta_new ~ dnorm(mu, prec_tau); # predictive distribution for theta
  p_new <- exp(theta_new)/(1+exp(theta_new)); # predictive distribution at the probability scale
  mu ~ dnorm(0, 1.0E-4); # just our assumption
  prec_tau ~ dgamma(1.0E-3, 1.0E-3); # just our assumption
}
"

# Random effects meta-analysis in JAGS
pl_prec <- 1/(pl_se^2)
model <- jags.model(
  textConnection(pl1_modelString),
  data = list(
    y = pl_logit,
    prec_s = pl_prec),
  n.chains = 4, # rule of thumb
)
```

```
## Compiling model graph
```

```
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 8
##    Unobserved stochastic nodes: 11
##    Total graph size: 34
##
## Initializing model
```

```
update(model, 5000)
model_samples <- coda.samples(model, c("theta_new", "p_new"), 40000)

model_samples |> summary()
```

```
##
## Iterations = 5001:45000
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 40000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean      SD  Naive SE Time-series SE
## p_new      0.2583 0.06347 0.0001587      0.0002236
## theta_new -1.0803 0.33950 0.0008488      0.0011882
##
## 2. Quantiles for each variable:
##
##              2.5%     25%     50%     75%   97.5%
## p_new      0.1414  0.2257  0.2541  0.2839  0.4080
## theta_new -1.8035 -1.2329 -1.0769 -0.9251 -0.3721
```

c)

Let's go through the code in chunks.

```
pl1_modelString <- "
model {
```

This is a setup for jags, which requires its own syntax and thus needs to be specified in a string. Let's take a look at the model specification.

```
  for(i in 1:length(y)){
    y[i] ~ dnorm(theta[i], prec_s[i]);
    theta[i] ~ dnorm(mu, prec_tau);
  }
```

This is the likelihood and prior specification. We assume that the data is normally distributed around the theta values, which are in turn normally distributed around mu. The `prec_s` and `prec_tau` are the precisions of the normal distributions. `y` and `prec_s` will be passed as data to the model later.

```
  theta_new ~ dnorm(mu, prec_tau); # predictive distribution for theta
```

`theta_new` is the new theta value we want to predict, i.e. our output. We assume that it is normally distributed around `mu` with precision `prec_tau`.

```
  p_new <- exp(theta_new)/(1+exp(theta_new)); # predictive distribution at the probability scale
```

We then transform `theta_new` to the probability scale. The transformation is the inverse logit function, i.e. the sigmoid function. We have to do this because we model the logit of the probability for better normality.

```
  mu ~ dnorm(0, 1.0E-4); # just our assumption
  prec_tau ~ dgamma(1.0E-3, 1.0E-3); # just our assumption
```

These are the priors for `mu` and `prec_tau`. We assume that `mu` is normally distributed around 0 with a very small precision and that `prec_tau` is gamma distributed with shape and rate parameters of 1e-3. These values will be updated during the MCMC sampling.

```
}
"
```

This is the end of the model specification.

```
pl_prec <- 1/(pl_se^2)
```

Since we model the data through the logit transformation, we need to calculate the precision of the ensuing normal distribution from the standard error. This is how this is done.

```
model <- jags.model(
  textConnection(pl1_modelString),
  data = list(
    y = pl_logit,
    prec_s = pl_prec),
  n.chains = 4, # rule of thumb
)
```

Here we just plug in the model string and the data described before into the jags model function. We also specify the number of chains to run, which is 4 by convention.

```
update(model, 5000)
```

We update the model in a burn-in phase. This is done by running the model for 5000 iterations.

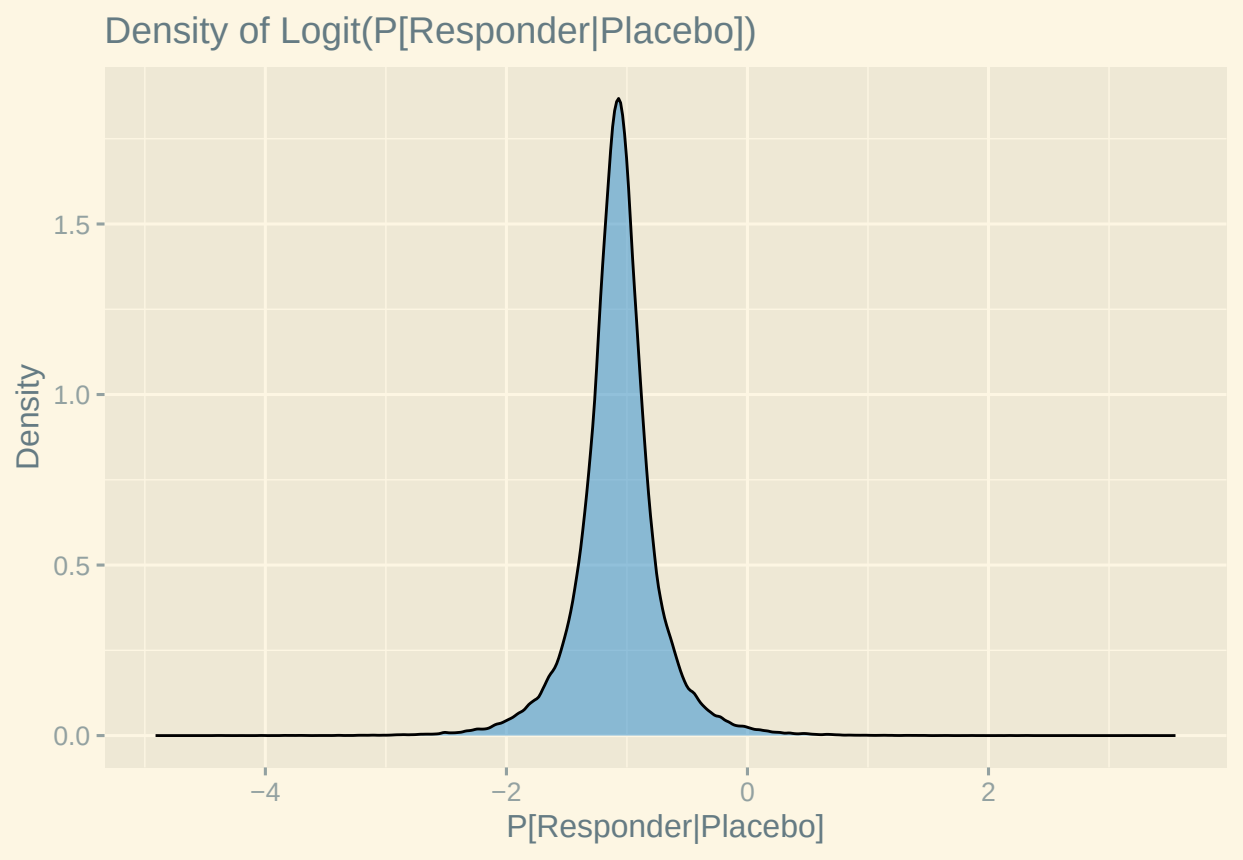
```
model_samples <- coda.samples(model, c("theta_new", "p_new"), 40000)
```

Finally, we sample from the model to get the posterior samples of `theta_new` and `p_new` after we run the model for 40000 iterations.

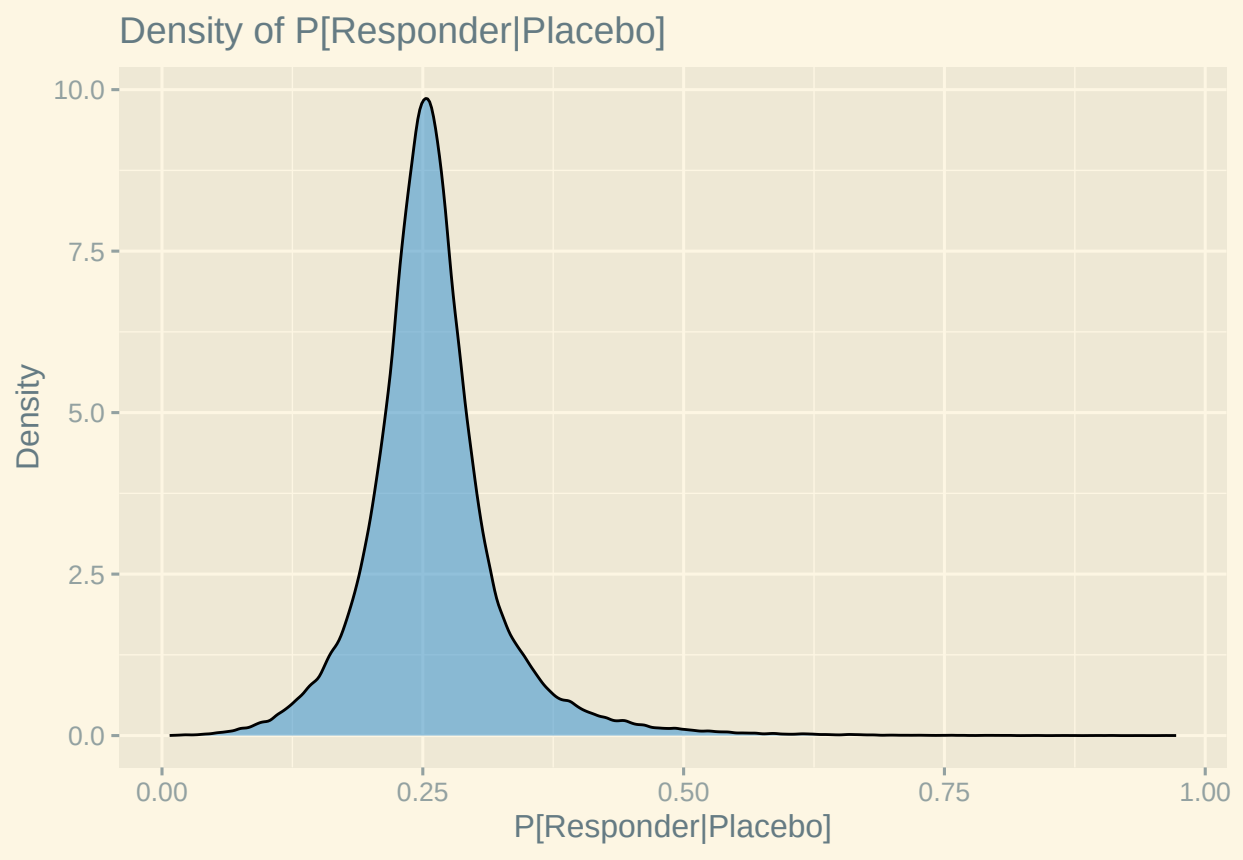
d)

```
model_samples.df <- do.call(rbind.data.frame, model_samples)
p_new <- model_samples.df$p_new
```

```
tibble(p = model_samples.df$theta_new) |>
  ggplot(aes(x = p)) +
  geom_density(fill = primary_colour, alpha = 0.5) +
  labs(title = "Density of Logit(P[Responder|Placebo])", x = "P[Responder|Placebo]", y = "Density") +
  scale_color_solarized()
```




```
tibble(p = p_new) |>
  ggplot(aes(x = p)) +
  geom_density(fill = primary_colour, alpha = 0.5) +
  labs(title = "Density of P[Responder|Placebo]", x = "P[Responder|Placebo]", y = "Density") +
  scale_color_solarized()
```



The posterior looks fairly narrow around 25%

e)

```
quantile(p_new, probs = c(0.025, 0.5, 0.975))
```

```
##      2.5%       50%     97.5%
## 0.1414224 0.2540890 0.4080318
```

It looks about right. My 95% credible interval is [14%, 41%] and the median is 25%, compared to the result of Baeten et al. of 25% and [13%, 40%].

f)

```
get_beta_params <- function(mu, sd) {
  alpha <- mu^2 * ((1 - mu) / sd^2 - 1 / mu)
  beta <- alpha * (1 - mu) / mu
  return(list(alpha = alpha, beta = beta))
}

mu <- p_new |> mean()
sd <- p_new |> sd()

beta_params <- get_beta_params(mu, sd)
alpha <- beta_params$alpha
beta <- beta_params$beta
```

g)

```
"Alpha: {alpha |> round(2)}" |> glue()
"Beta: {beta |> round(2)}" |> glue()
```

```
## Alpha: 12.03
## Beta: 34.54
```

The alpha and beta parameters reported by Baeten et al. were 11 and 32, which is fairly close to our result of 12 and 34.

h)

Our goal is to find out how likely a patient will respond to the placebo treatment. We've got historical data on previous placebo trials and will use these to estimate the underlying probability. First, we need to assume a distribution. Here, we use a normal distribution. The data is not normally distributed as-is, but can be transformed to a normal distribution by taking the logit. This is defined by the following equation:

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

After applying this transformation, we can model the data as normally distributed around the mean θ with the precision taken from the data's standard error, also transformed by the logit. θ itself is also assumed to be normally distributed around μ with precision τ, which are unknown parameters. We (semi-arbitrarily) assume that μ is normally distributed around 0 with a very small precision and that τ is gamma distributed with shape and rate parameters of 0.001. These are our priors. JAGS can be used to update these priors with the given data. The resulting posterior distribution of θ can then be transformed back to the probability scale using the inverse logit function. This will give us an estimate of the probability of a patient responding to the placebo treatment based on our data.

3.5/4

Exercise 2

a)

```
active_beta <- 1
placebo_alpha <- 11
placebo_beta <- 32

active_alphas <- seq(0, 10, by = 0.01)
n <- 5000

ps <- tibble(
  alpha = active_alphas,
  p = NA
)

for (active_alpha in active_alphas) {
  active_p <- rbeta(n, active_alpha, active_beta)
  placebo_p <- rbeta(n, placebo_alpha, placebo_beta)
  p <- mean(active_p > placebo_p)
  ps[ps$alpha == active_alpha, "p"] <- p
}

ps |>
  mutate(diff = abs(p - 0.5)) |>
```

Compute also the MCse for the estimate mean(active_p>placebo_p)

```
  arrange(diff) |>
  head(10)
```

```
## # A tibble: 10 x 3
##    alpha     p    diff
##    <dbl> <dbl>   <dbl>
##  1   0.5  0.503 0.00260
##  2   0.53 0.505 0.00480
##  3   0.48 0.490 0.0096
##  4   0.52 0.511 0.0110
##  5   0.49 0.488 0.0116
##  6   0.51 0.512 0.0122
##  7   0.54 0.527 0.0266
##  8   0.56 0.531 0.0314
##  9   0.47 0.466 0.0340
## 10   0.57 0.535 0.0352
```

We can see that the alpha that results in the closest P[Active > Placebo] to 50% is indeed 0.5.

b)

The high-level idea is to simulate outcomes from the beta distributions of the active and placebo groups and then calculate the proportion of times the active group has a higher response rate than the placebo group. We then repeat this process for different values of alpha and find the one that results in a P[Active > Placebo] closest to 50%. The reason we care about this outcome in particular is that it is the most conservative estimate of the treatment effect, i.e. treatment and placebo are equally likely to be better. As we have already fixed the β parameter for the active group to 1, we can only vary the α parameter to achieve this.

Let's go through it bit by bit.

```
active_beta <- 1
placebo_alpha <- 11
placebo_beta <- 32
```

These are the given parameters for the beta distributions of the active and placebo groups. Active is $\sim$ Beta(α, 1) and placebo is $\sim$ Beta(11, 32).

```
active_alphas <- seq(0, 10, by = 0.01)
```

We want to vary the alpha parameter of the active group from 0 to 10 in steps of 0.01.

```
n <- 5000
```

We will simulate 5000 outcomes for each alpha value.

```
ps <- tibble(
  alpha = active_alphas,
  p = NA
)
```

Here we set up a data frame to store the results.

```
for (active_alpha in active_alphas) {
```

We loop through the alpha values.

```
  active_p <- rbeta(n, active_alpha, active_beta)
  placebo_p <- rbeta(n, placebo_alpha, placebo_beta)
```

We simulate the outcomes for the active and placebo groups. The placebo group uses the fixed alpha and beta parameters discussed before, while the active group uses the current alpha value and a fixed beta value of 1.

```
  p <- mean(active_p > placebo_p)
  ps[ps$alpha == active_alpha, "p"] <- p
}
```

We calculate the proportion of times the active group has a higher response rate than the placebo group and store it in the data frame.

```
ps |>
  mutate(diff = abs(p - 0.5)) |>
  arrange(diff) |>
  head(10)
```

Finally, we calculate the difference between the proportion and 0.5, sort the data frame by this difference and show the 10 alpha values that result in the smallest difference.