Pilot 3

Alex Kale

8/8/2019

Recap of Pilot 2

In pilot 2 we asked participants to view distributions of predicted performance for two widget manufacturing machines and judge the probability that a new machine would produce more (defective) widgets the old machine (a.k.a., probability of superiority). We then asked them whether or not they would buy the new machine, where the utility optimal decision rule was such that they should buy the new machine on half of trials where effect sizes are most extreme. Each participant comlpeted 24 trials at different levels of effect size and using one of three visualizations: means and intervals, means only, or HOPs. Where the ground truth probability of superiority was above 50%, the task was framed as a possibility of gaining a contract above some threshold number of widgets produced. Where the ground truth probability of superiority was below 50%, the task was framed as a possibility of losing a contract above some threshold number of defective widgets produced.

There were a handful of issues with this study design: 1. The elicitation of probability of superiority judgments was confusing. Participants often responded on the wrong side of 50%, yielding poor data quality and inferential validity. 2. Our strategy for sampling ground truth effect size was optimized for modeling probability of superiority judgments but not for modeling decision-making. The values we sampled did not cluster near the utility optimal decision threshold, yielding psychometric function fits with higher entropy than is ideal. 3. Manipulating the baseline probability of gaining or keeping the contract with the old machine proved fruitless. Model comparisons indicate that including this manipulation as a predictor was not worth the increased risk of overfitting.

To remedy these issues, we redesign the interface to reduce confusion. We change the probability of superiority elicitation from a scale of 0 to 100 to a scale of 50 to 100. We also change the task from machine purchasing decisions to a fantasy sports scenario to make it more engaging. Last, we remove the baseline manipulation to simplify the study design.

Pilot 3

In pilot three, we the structure of the incentivized decision task we had for pilot two, but we change the narrative around the task.

Betting Task: Scenario and Payoff Scheme

In our new task, users play a fantasy sports game where they win or lose awards depending on how many points their team scores or gives up, respectively. They are presented with charts comparing how many points their team is predicted to score or give up with or without adding a new player to their team.

In the gain framing trails, users are told they will *win an award* worth \$X if their team *scores more than 100 points* (an arbitrary number). They can improve their chances of getting that award if they pay \$C for a new player The optimal decision rule in this case maximizes expected gains.

$$X * p(award| \sim player) < X * p(award|player) - C$$

If we assume a constant ratio between the value of the award and the cost of the new player $K = \frac{X}{C}$, the decision rule can be expressed as in terms of the difference in probability of winning the award with and without a new player.

$$p(award|player) - p(award| \sim player) > \frac{1}{K}$$

The more the new player increases their chances of winning the award, the more evidence there is in favor of intervening by paying for the new player.

The loss framing trails are similarly set up. Users are told they will *lose an award* worth $X if their team *gives up more than 75 points* (an arbitrary number). They can improve their chances of keeping that award if they pay $C for a new player. The optimal decision rule in this case minimizes expected losses.

$$X * p(\sim award| \sim player) > X * p(\sim award|player) + C$$

Again, if we assume a constant ratio between the value of the award and the cost of the new player $K = \frac{X}{C}$, the decision rule can be expressed as in terms of the difference in probability of losing the award with and without a new player

$$p(\sim award| \sim player) - p(\sim award|player) > \frac{1}{K}$$

The more the new player decrease their chances of losing the award, the more evidence there is in favor of intervening by paying for the new player.

Each trial, users will receive feedback based on their decision and a simulated outcome regarding the award in question. We will tally the dollar value of awards won/kept minus the cost of new players across trials to determine the user's fantasy sports account balance. Users will receive a bonus through MTurk that is porportional to the value of their account at the end of the HIT.

Visualization Conditions

Users will be shown predicted distributions of how many points their team will score or give up with vs without the new player. Number of points will be visualized as *means only, means with intervals, HOPs with means, densities, quantile dotplots, intervals only, and HOPs*. These conditions span a continuum of how much the make the mean available to users, from emphasizing the mean to only encoding it implicitly.

Data Conditions

We manipulate the probability of the team scoring or giving up more points with vs without the new player (p_superiority). We employ two sampling strategies, one which optimizes for each of the two questions we ask participants: 1. Linear intervals in logodds units to give perceptually uniform steps in probability of superiority. 2. Probability of superiority values near the utility optimal decision threshold (i.e., p_superiority == [0.13, 0.87]).

When p_superiority is greater than 0.5, the decision task is framed as a gain scenario where the user's team needs to score at least 100 points to win an award. When p_superiority is less than 0.5, the decision task is framed as a loss scenario where the user's team needs to give up fewer than 75 points to keep an award.

```
# linear sampling of log odds for full span of ground truth probability of superiorit
y between 0.025 and 0.975
n_trials.full_span <- 20
logodds.full_span <- seq(log(0.025 / (1 - 0.025)), log(0.975 / (1 - 0.975)), length.o
ut = n_trials.full_span)

# linear sampling of log odds near the decision threshold (p_superiority == [0.13, 0.
87])
n_trials.near_threshold <- 8
logodds.near_threshold <- c(seq(log(0.1 / (1 - 0.1)), log(0.2 /(1 - 0.2)), length.out
= n_trials.near_threshold / 2), # near threshold for loss frame
                            seq(log(0.8 / (1 - 0.8)), log(0.9 /(1 - 0.9)), length.out
= n_trials.near_threshold / 2)) # near threshold for gain frame

# combine the sampling strategies and convert from log odds to probability of superio
rity
logodds <- sort(c(logodds.full_span, logodds.near_threshold))
p_superiority <- 1 / (1 + exp(-logodds))
n_trials <- length(p_superiority)

print(p_superiority)
```

```
##  [1] 0.02500000 0.03633635 0.05253624 0.07539348 0.10000000 0.10707153
##  [7] 0.12709249 0.14990182 0.16021834 0.20000000 0.20591399 0.27605902
## [13] 0.35928769 0.45194404 0.54805596 0.64071231 0.72394098 0.79408601
## [19] 0.80000000 0.83978166 0.85009818 0.87290751 0.89292847 0.90000000
## [25] 0.92460652 0.94746376 0.96366365 0.97500000
```

This time around, we set the baseline probability of winning/keeping the award without the new player to a constant value of 0.5. The team is as likely as a coin flip to win or keep the award without the new player. This represents the scenario where there is the maximum uncertainty about outcomes without intervention.

```
# baseline probability of winning/keeping an award contract without the new player
baseline <- c(.5) # previously c(.15, .5, 8.5)

# initialize data conditions dataframe
conds_df <- data.frame(
  "p_superiority" = rep(p_superiority, length(baseline)),
  "baseline" = sort(rep(baseline, length(p_superiority))))

head(conds_df)
```

```
##   p_superiority baseline
## 1    0.02500000      0.5
## 2    0.03633635      0.5
## 3    0.05253624      0.5
## 4    0.07539348      0.5
## 5    0.10000000      0.5
## 6    0.10707153      0.5
```

As stated above, we set the threshold for winning the award in the gain frame at 100 points and the threshold for keeping the award in the loss frame at 75 points.

```
# label gain vs loss framing trials based on p_superiority, and add award thresholds
conds_df <- conds_df %>%
  mutate(frame = if_else(p_superiority > .5, "gain", "loss"),
         threshold = if_else(frame == "gain",
                             100, # points scored
                             75)) # points given up

head(conds_df)
```

```
##   p_superiority baseline frame threshold
## 1    0.02500000      0.5  loss        75
## 2    0.03633635      0.5  loss        75
## 3    0.05253624      0.5  loss        75
## 4    0.07539348      0.5  loss        75
## 5    0.10000000      0.5  loss        75
## 6    0.10707153      0.5  loss        75
```

We control the standard deviation of the distribution of the difference in points between the team with and without the new player (sd_diff) by setting it to 15. In the gain framing this is 15 points scored. In the loss framing, this is 15 points given up. We can think of this variable as constant across trials. We then derive the mean difference in the number of points scored by the team with minus without the new player (mean_diff) from sd_diff and p_superiority.

```
# add columns for the mean and standard deviation of the difference in the number of
 points for the team with vs without the new player
# depending on the gain vs loss frame, these values represent points scored vs points
given up
conds_df <- conds_df %>%
  mutate(sd_diff = 15, # std(new - old)
         mean_diff = sd_diff * qnorm(p_superiority)) # mean(new - old)

head(conds_df)
```

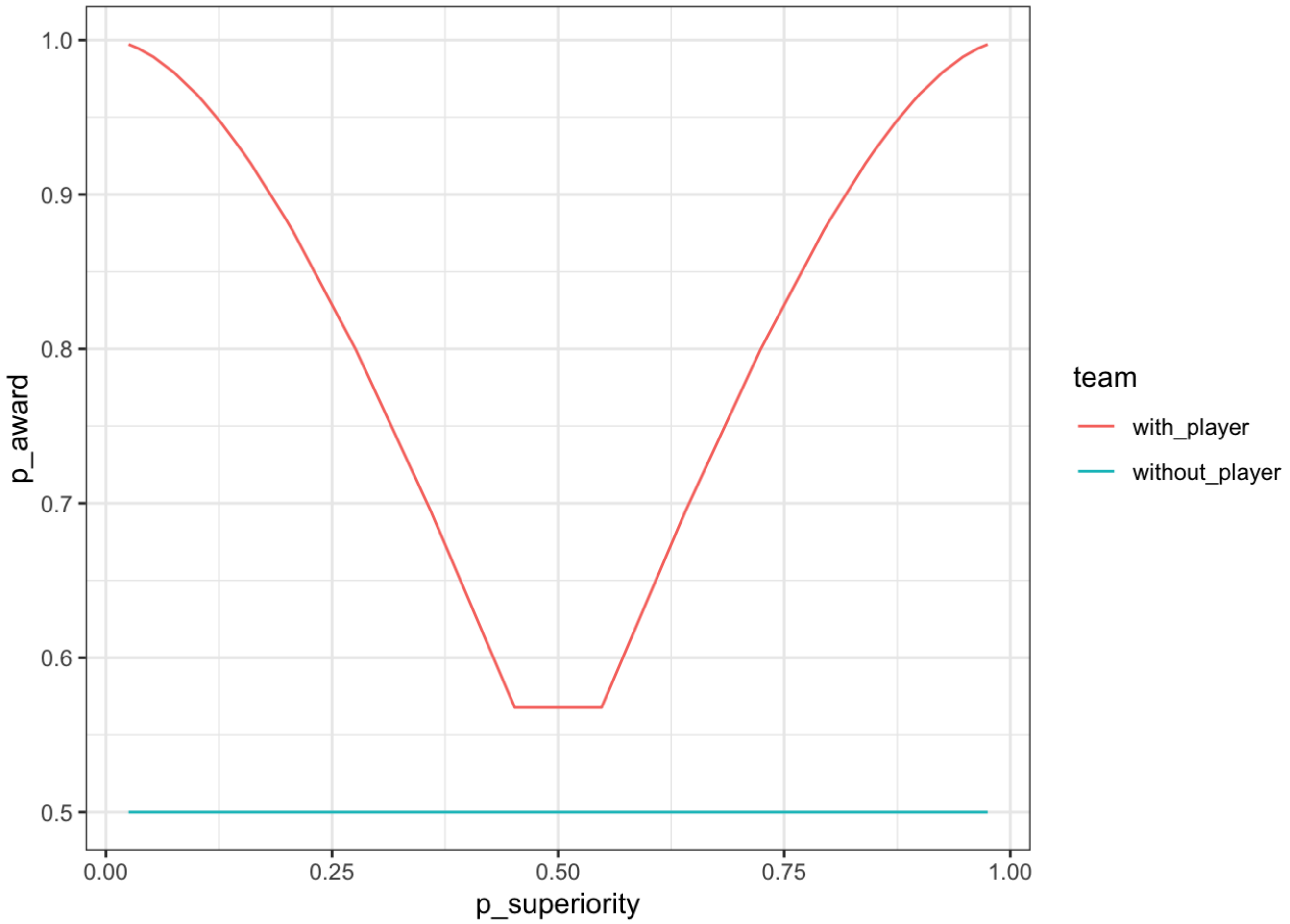
```
##   p_superiority baseline frame threshold sd_diff mean_diff
## 1    0.02500000      0.5  loss        75      15 -29.39946
## 2    0.03633635      0.5  loss        75      15 -26.92321
## 3    0.05253624      0.5  loss        75      15 -24.31117
## 4    0.07539348      0.5  loss        75      15 -21.55136
## 5    0.10000000      0.5  loss        75      15 -19.22327
## 6    0.10707153      0.5  loss        75      15 -18.63380
```

Now we calculate the summary statistics for the team with and without the new player, making the dataframe double its length up to this point. We derive the standard deviation of the points scored by the teams with and without the new player (sd) from sd_diff, variance sum law, and the assumption that the teams with or without the new player have equal and independent variances. We derive the mean number points scored by the teams with and without the new player (mean) from the threshold for winning/keeping the award, the sd of points for each version of the team, and the mean_diff between the number of points for with minus without the new player. We derive the probability of winning/keeping the award from the threshold, mean, and sd.

```
# double the length of the dataframe to add information per version of the team, crea
ting a stimulus dataframe with a row per distribution to visualize
stim_df <- map_df(seq_len(2), ~conds_df)
stim_df$team <- as.factor(sort(rep(c("with_player","without_player"), length(stim_df
$p_superiority) / 2)))


# add columns for the mean and standard deviation of points for each team and the pro
bability of winning/keeping the award
stim_df <- stim_df %>%
  mutate(sd = sqrt(stim_df$sd_diff ^ 2 / 2), # assume equal and independent variances
        mean = if_else(team == "without_player",
                       if_else(frame == "gain", # team without the new player is at b
aseline
                               threshold - sd * qnorm(1 - baseline),
                               threshold - sd * qnorm(baseline)),
                       if_else(frame == "gain", # team with new player is at differen
ce from baseline
                               threshold - sd * qnorm(1 - baseline) + mean_diff,
                               threshold - sd * qnorm(baseline) + mean_diff)),
        p_award = if_else(frame=="gain", # probability of exceeding threshold to win/
keep award
                          1 - pnorm((threshold - mean)/sd),
                          pnorm((threshold - mean)/sd)))

# spread values per machine across columns to get back to a conditions dataframe one
 row per trial
conds_df <- stim_df %>% # explanation: https://kieranhealy.org/blog/archives/2018/11/
06/spreading-multiple-values/
  gather(variable, value, -(p_superiority:team)) %>%
  unite(temp, team, variable) %>%
  spread(temp, value)

head(conds_df)
```

```
##   p_superiority baseline frame threshold sd_diff mean_diff
## 1    0.02500000      0.5  loss        75      15 -29.39946
## 2    0.03633635      0.5  loss        75      15 -26.92321
## 3    0.05253624      0.5  loss        75      15 -24.31117
## 4    0.07539348      0.5  loss        75      15 -21.55136
## 5    0.10000000      0.5  loss        75      15 -19.22327
## 6    0.10707153      0.5  loss        75      15 -18.63380
##   with_player_mean with_player_p_award with_player_sd without_player_mean
## 1         45.60054           0.9972127        10.6066                  75
## 2         48.07679           0.9944311        10.6066                  75
## 3         50.68883           0.9890494        10.6066                  75
## 4         53.44864           0.9789172        10.6066                  75
## 5         55.77673           0.9650368        10.6066                  75
## 6         56.36620           0.9605250        10.6066                  75
##   without_player_p_award without_player_sd
## 1                    0.5           10.6066
## 2                    0.5           10.6066
## 3                    0.5           10.6066
## 4                    0.5           10.6066
## 5                    0.5           10.6066
## 6                    0.5           10.6066
```

This results in an experimental design where the probability of winning/keeping the award with the new player increases monotonically with p_superiority. This means that users should intervene only at extreme values of p_superiority. Even though the decision rule is not defined in terms of p_superiority, users can user effect size as a proxy for the decision task.



Contract Values and Thresholds

Since we hold the cost of the new player constant at $1M, we can think of K as the *value of the award in millions of dollars*. We need to set K so that there is an *equal number of trials where users should vs shouldn't intervene*. A value that guarantees this balance for our sample of probability of superiority values is 2.25.

```
# ratio (K) of value of contract (X) over cost of intervention (C)
conds_df <- conds_df %>% mutate(K = 2.25)
```

Let's check that we have an equal number of trials where intervening is and is not the optimal choice. We want to make sure that this balance is maintained across both levels framing.

```
# determine whether or not intervention is utility optimal on each trial
conds_df <- conds_df %>%
  mutate(should_intervene = if_else(frame == "gain",
                                    (with_player_p_award - without_player_p_award) >
 (1 / K), # gain framing decision rule
                                    ((1 - without_player_p_award) - (1 - with_player_
p_award)) > (1 / K)) # loss framing decision rule
  )

conds_df %>%
  group_by(frame) %>%
  summarise(intervene = sum(should_intervene), n_trials = n())
```

```
## # A tibble: 2 x 3
##   frame intervene n_trials
##   <chr>     <int>    <int>
## 1 gain          7       14
## 2 loss          7       14
```

What exactly are the values of probability of superiority at these thresholds?

```
conds_df <- conds_df %>%
  mutate(
    p_sup_threshold = if_else(frame == "gain",
                              pnorm(sqrt(sd_diff ^ 2 / 2) / sd_diff * (qnorm((1 / K)
 + baseline) + qnorm(baseline))),
                              pnorm(-sqrt(sd_diff ^ 2 / 2) / sd_diff * (qnorm((1 / K)
+ baseline) - qnorm(baseline))))
  )

# unique(conds_df$p_sup_threshold)
conds_df %>%
  group_by(frame) %>%
  summarise(threshold = unique(p_sup_threshold))
```

```
## # A tibble: 2 x 2
##   frame threshold
##   <chr>     <dbl>
## 1 gain      0.870
## 2 loss      0.130
```

Expected Bonuses

We also need to set the *starting value of the fantasy sports account* and the *exchange rate* of actual dollars in MTurk bonus per million of dollars in account value.

First, we *set the starting value of the user's account equal to the maximum possible amount participants could lose*, if they buy the new player every trial and always fail to win/keep the award.

```
# starting value depends on maximum possible loss (if they pay for the new machine ev
ery time and never gain\keep the contract)
conds_df <- conds_df %>% mutate(starting_account_value = n_trials + n_trials / 2 * K)

conds_df %>% group_by(frame, K, starting_account_value) %>% summarise()
```

```
## # A tibble: 2 x 3
## # Groups:   frame, K [2]
##   frame     K starting_account_value
##   <chr> <dbl>                  <dbl>
## 1 gain   2.25                   59.5
## 2 loss   2.25                   59.5
```

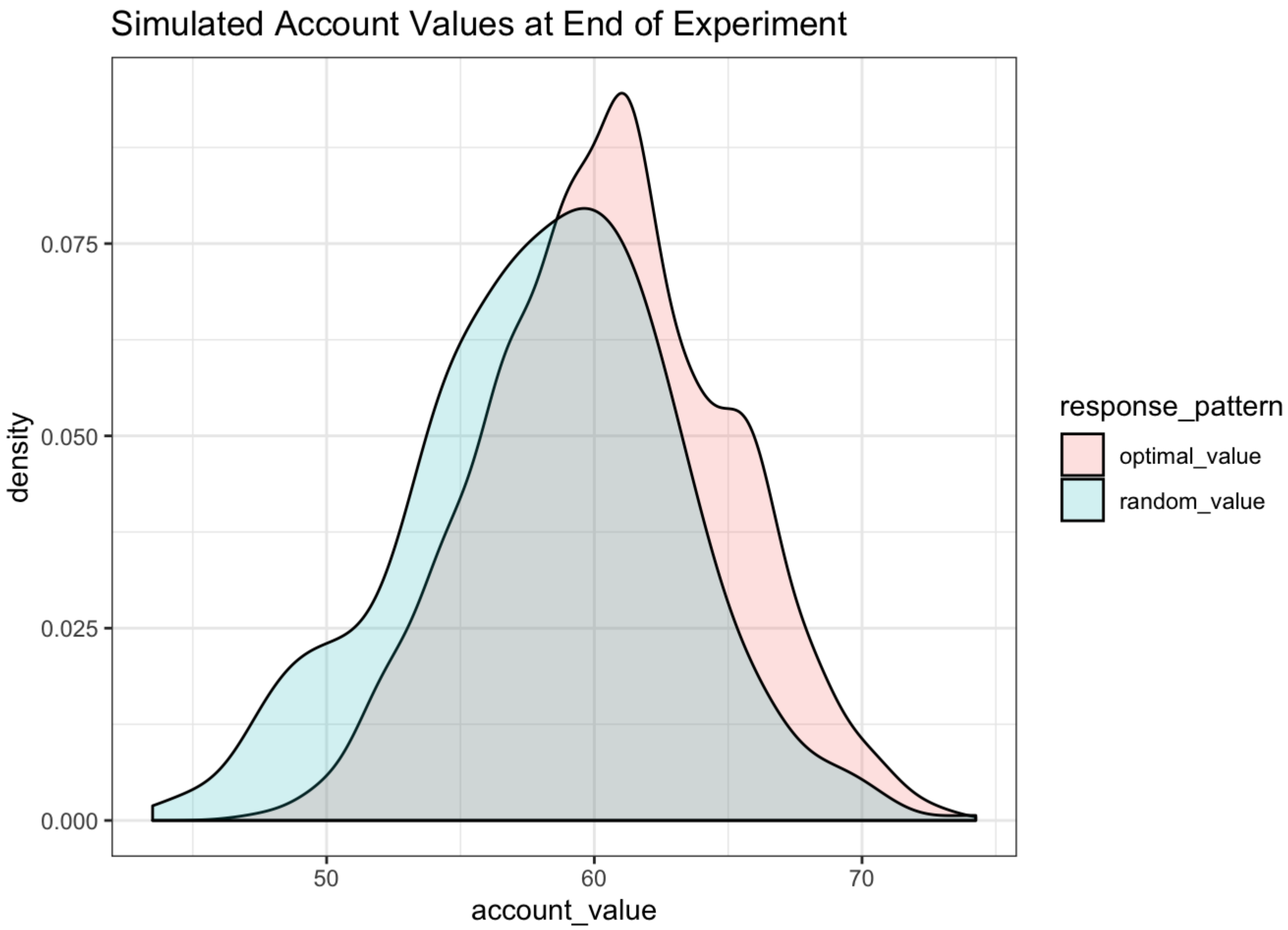
Next, *we set the exchange rate to range from $0 to $3 depending on decision quality*. To figure out how to calculate bonuses, we want to know what account values would look like at the end of the experiment if users guessed on every trial vs if they made the optimal choice on every trial. To learn this, we'll run a simulation of two response patterns: random guessing vs utility optimal decision-making.

```
# set up for simulation
n_iter <- 500
simulation_df <- NULL

# generate outcome, whether award is won/kept or not
outcome <- function(intervene, p_with, p_without) {
  if (intervene) {
    return(runif(1) <= p_with)
  } else {
    return(runif(1) <= p_without)
  }
}

for (i in 1:n_iter) {
  temp <- conds_df %>%
    mutate(
      # generate payoff for random guess
      random_guess = as.logical(rbinom(n(), 1, 0.5)),
      random_outcome = as.logical(pmap(list(random_guess, with_player_p_award, withou
t_player_p_award), outcome)),
      random_payoff = if_else(p_superiority > 0.5,
                              # gain frame
                              if_else(random_outcome,
                                      K - random_guess,
                                      as.numeric(-random_guess)),
                              # loss frame
                              if_else(random_outcome,
                                      as.numeric(-random_guess),
                                      -K - random_guess)),
      random_correct = (random_guess == should_intervene),
      # generate payoff for optimal guess
      optimal_outcome = as.logical(pmap(list(should_intervene, with_player_p_award, w
ithout_player_p_award), outcome)),
      optimal_payoff = if_else(p_superiority > 0.5,
                              # gain frame
                              if_else(optimal_outcome,
                                      K - should_intervene,
                                      as.numeric(-should_intervene)),
                              # loss frame
                              if_else(optimal_outcome,
                                      as.numeric(-should_intervene),
                                      -K - should_intervene)),
      optimal_correct = TRUE
    ) %>%
    summarise(
      iter = i,
      random_value = mean(starting_account_value) + sum(random_payoff),
      random_correct = sum(random_correct),
      optimal_value = mean(starting_account_value) + sum(optimal_payoff),
      optimal_correct = sum(optimal_correct)
    )
  simulation_df = rbind(simulation_df, temp)
}

head(simulation_df)
```

```
##   iter random_value random_correct optimal_value optimal_correct
## 1    1        48.00             14         61.25              28
## 2    2        49.00             15         63.50              28
## 3    3        61.00              8         65.75              28
## 4    4        56.75             14         61.25              28
## 5    5        60.00             13         56.75              28
## 6    6        55.50             15         61.25              28
```

Let’s plot the results of our simulation.

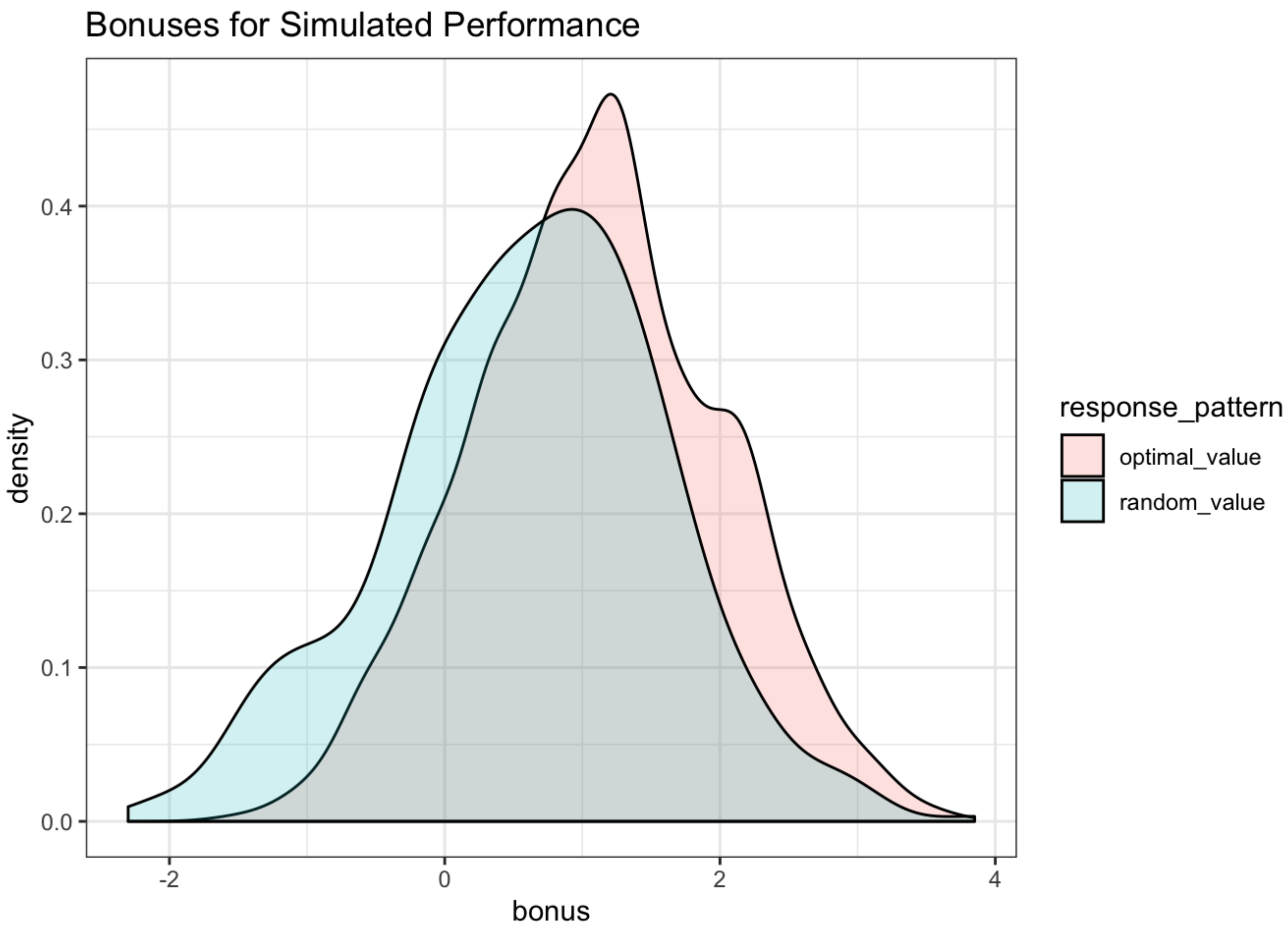
```
simulation_df %>%
  select(iter, random_value, optimal_value) %>%
  gather(response_pattern, account_value, -iter) %>%
  ggplot(aes(x = account_value, fill = response_pattern)) +
  geom_density(alpha = 0.2) +
  theme_bw() +
  labs(
    title = "Simulated Account Values at End of Experiment"
  )
```



We can see that account values aren’t reliably different between these two strategies. This is problem for differentiating between good and poor performance.

In order to make the incentives fair, we’ll can try setting the low end of likely account values under the optimal response distribution (i.e., an account value of 55) to correspond to no bonus. Above that every unit of company value should count for $0.20. This bonus structure would reward performance the most for account values that are unlikely to be obtainted by random guessing. However, luck still has a large impact on this payoff scheme.

```
# set exchange rate and cutoff
cutoff <- 55
exchange <- 0.2

simulation_df %>%
  select(iter, random_value, optimal_value) %>%
  gather(response_pattern, account_value, -iter) %>%
  mutate(bonus = (account_value - cutoff) * exchange) %>%
  ggplot(aes(x = bonus, fill = response_pattern)) +
  geom_density(alpha = 0.2) +
  theme_bw() +
  labs(
    title = "Bonuses for Simulated Performance"
  )
```



Alternatively, we could give participants feedback on their decisions in terms of expected utility. This is less realistic, but it should differentiate strongly between random guessing and optimal decisions. Let's simulate this behavior.

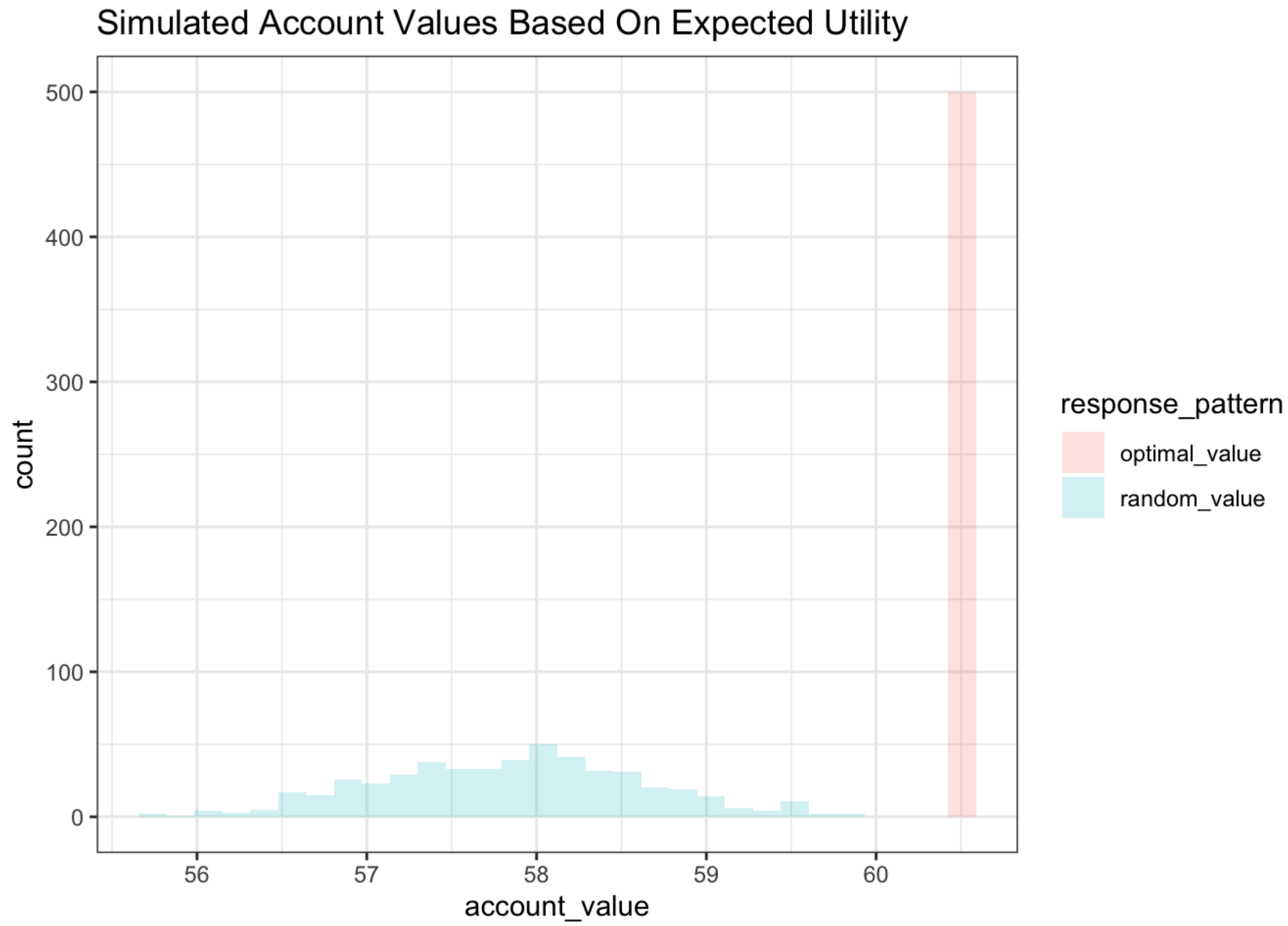
```
# set up for simulation
deterministic_df <- NULL

for (i in 1:n_iter) {
  temp <- conds_df %>%
    mutate(
      # generate deterministic for random guess
      random_guess = as.logical(rbinom(n(), 1, 0.5)),
      random_payoff = if_else(p_superiority > 0.5,
                              # gain frame
                              if_else(random_guess,
                                      K * with_player_p_award - 1,
                                      K * without_player_p_award),
                              # loss frame
                              if_else(random_guess,
                                      -K * (1 - with_player_p_award) - 1,
                                      -K * (1 - without_player_p_award))),
      random_correct = (random_guess == should_intervene),
      # generate deterministic payoff for optimal guess
      optimal_payoff = if_else(p_superiority > 0.5,
                               # gain frame
                               if_else(should_intervene,
                                       K * with_player_p_award - 1,
                                       K * without_player_p_award),
                               # loss frame
                               if_else(should_intervene,
                                       -K * (1 - with_player_p_award) - 1,
                                       -K * (1 - without_player_p_award))),
      optimal_correct = TRUE
    ) %>%
    summarise(
      iter = i,
      random_value = mean(starting_account_value) + sum(random_payoff),
      random_correct = sum(random_correct),
      optimal_value = mean(starting_account_value) + sum(optimal_payoff),
      optimal_correct = sum(optimal_correct)
    )
  deterministic_df = rbind(deterministic_df, temp)
}

head(deterministic_df)
```

```
##   iter random_value random_correct optimal_value optimal_correct
## 1    1     57.83304             13      60.49288              28
## 2    2     56.84677             10      60.49288              28
## 3    3     58.84001             17      60.49288              28
## 4    4     58.01401             13      60.49288              28
## 5    5     57.24298             12      60.49288              28
## 6    6     59.02835             16      60.49288              28
```

Let’s take a look at the results of our simulation. We can see that optimal responses obtain a completely separate distribution of account values.

```
# plot for deterministic outcomes
deterministic_df %>%
  select(iter, random_value, optimal_value) %>%
  gather(response_pattern, account_value, -iter) %>%
  ggplot(aes(x = account_value, fill = response_pattern)) +
  geom_histogram(alpha = 0.2) +
  theme_bw() +
  labs(
    title = "Simulated Account Values Based On Expected Utility"
  )
```

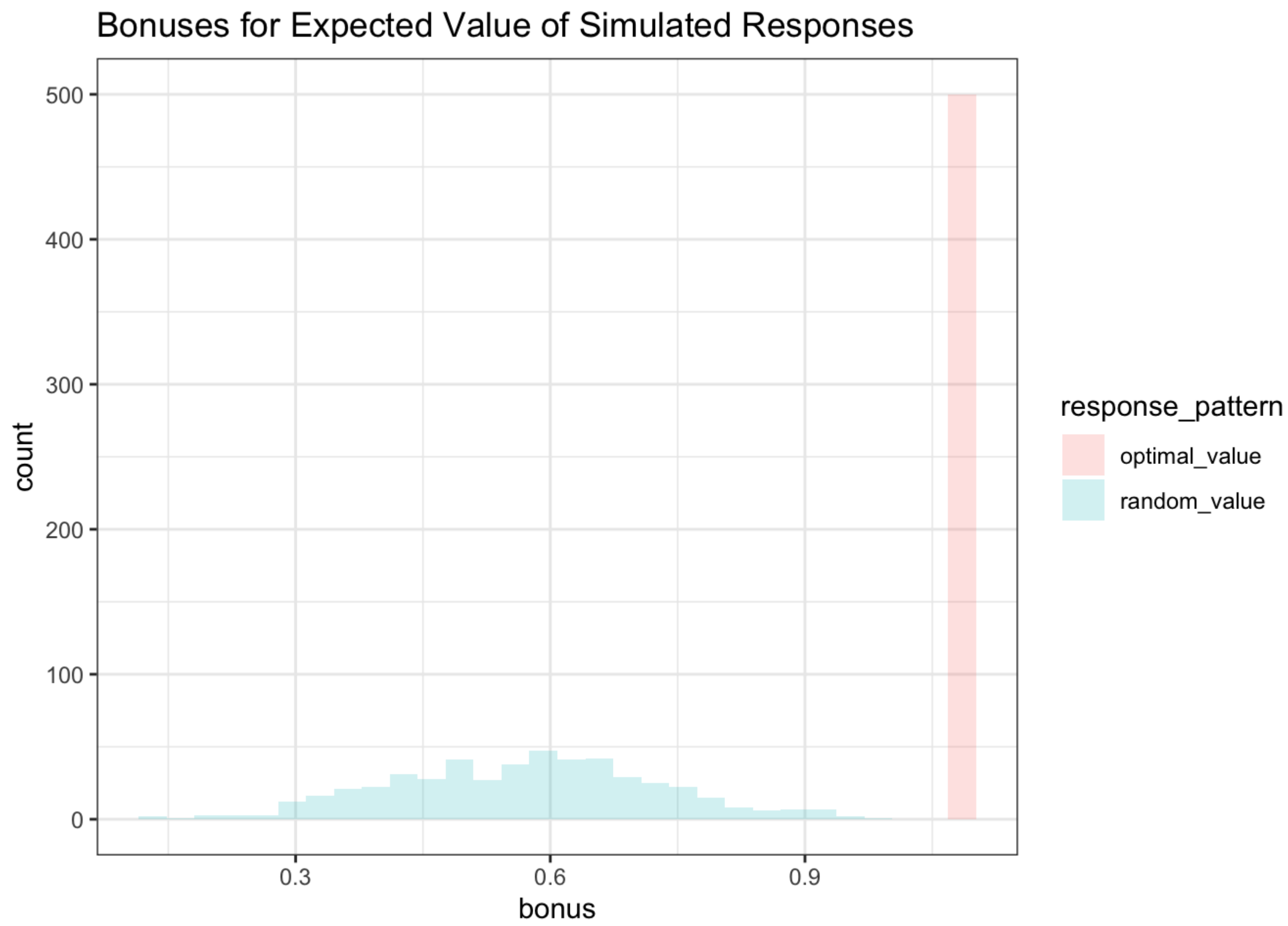
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



If we apply the same cutoff of exchange rate to this data, we can see that this approach is more rewarding for participants to try their best on the task compared to those who guess randomly.

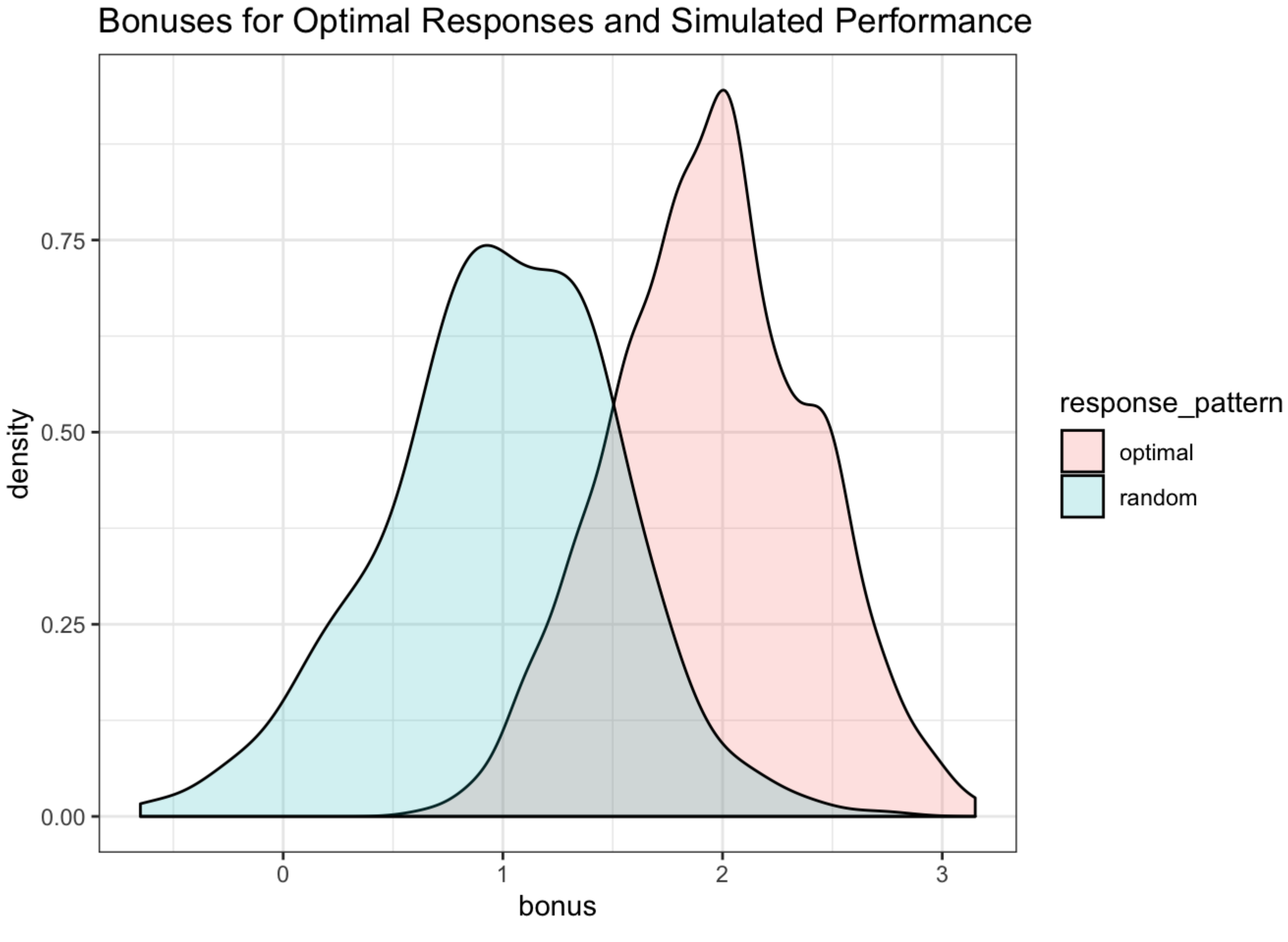
```
# set exchange rate and cutoff
cutoff <- 55
exchange <- 0.2

deterministic_df %>%
  select(iter, random_value, optimal_value) %>%
  gather(response_pattern, account_value, -iter) %>%
  mutate(bonus = (account_value - cutoff) * exchange) %>%
  ggplot(aes(x = bonus, fill = response_pattern)) +
  geom_histogram(alpha = 0.2) +
  theme_bw() +
  labs(
    title = "Bonuses for Expected Value of Simulated Responses"
  )
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



However, we do want users to pay attention to the probabilistic outcomes in the task to emphasize the uncertainty in effect size estimates. Ideally, we want to introduce the fairness of a deterministic payoff into a task which is still stochastic in nature. We can acheive this by rewarding users for their company value at a lower rate and giving them an additional 5 cents bonus for each utility optimal decision. Let's take a look at the results of this kind of mixed payoff scheme.

```
# set exchange rate and cutoff
cutoff <- 55
exchange <- 0.1
correct_bonus <- 0.05

simulation_df %>%
  gather(key, value, -iter) %>%
  separate(key, into = c("response_pattern", "value_type"), sep = "_") %>%
  spread(value_type, value) %>%
  rename(account_value = value) %>%
  mutate(bonus = (account_value - cutoff) * exchange + correct * correct_bonus) %>%
  ggplot(aes(x = bonus, fill = response_pattern)) +
  geom_density(alpha = 0.2) +
  theme_bw() +
  labs(
    title = "Bonuses for Optimal Responses and Simulated Performance"
  )
```



Something like this gives us the best of both worlds, however it will complicate feedback.