Supplementary Appendix

July 31, 2023

Contents

1) Code generating simulated datasets used in Figure 2.

This section includes the code used to generate the simulated datasets in Figure 2. I also plot alternative figures that use different values for the `degree_overlap` and `n_subjects` parameters. In words, this means that the plots vary the share of actions that are coded by more than one URA and the number of actions actually assigned to each URA.

```
knitr::opts_chunk$set(cache = T)
# see here re global seed: https://stackoverflow.com/questions/47897435/how-to-use-set-seed-globally-in
```

```
library(tidyverse)
library(furrr)


# install ura from Github:
# devtools::install_github("bengoehring/ura")
library(ura)

# set seed and number of cores for parallel processing
plan(multisession, workers = 8)
```

```
# Create a function, make_ex_data, to generate simulated coding datasets.

# Arguments:
#   n_subjects:       The number of subjects being coded
#   n_raters:         The number of raters doing the coding
#   degree_overlap:   A value of 1 means that all of the subjects are coded by
#                     all of the raters. As 1 increases, an increasingly large
#                     subset of the subjects are coded by only one rater.
#   rater_i_accuracy: The accuracy of the bad apple rater who diverges from the
#                     other raters' coding. A value of 1 means that the rater is
#                     just as accurate as his peers, while a value of zero means
#                     that he codes subjects exactly opposite the other raters.

make_ex_data <- function(n_subjects,
```

```
                        n_raters,
                        degree_overlap,
                        rater_i_accuracy) {

  # set up initial data framework
  initial_data <- data.frame(matrix(nrow = n_subjects,
                                    ncol = n_raters))
  names(initial_data) <- str_c('rater_', 1:n_raters)

  # randomly generate the subjects being coded by each rater
  example_data <- initial_data %>%
    as_tibble() %>%
    mutate(across(everything(),
                  ~sample(1:(n_subjects * degree_overlap),
                          size = n_subjects))) %>%
  pivot_longer(cols = everything(),
               values_to = "subject",
               names_to = "rater") %>%
  mutate(rater = as.numeric(str_remove(rater,
                                       "rater_")))

  # Assign the binary coding values. For each multi-coded subject, the given
  #   raters will always agree on the coding The exception is rater i, whose
  #   accuracy depends on the argument rater_i_accuracy.
  example_data <- example_data %>%
  group_by(subject) %>%
  mutate(coding = list_c(sample(list(rep(0, n()),
                                     rep(1, n())),
                                size = 1))) %>%
  ungroup() %>%
  mutate(rater_i = max(rater)) %>%
  group_by(rater,
           coding) %>%
  mutate(coding = case_when(
    rater == rater_i & coding == 0 ~ sample(c(0, 1),
                                      size = n(),
                                      prob = c(rater_i_accuracy,
                                               1 - rater_i_accuracy),
                                      replace = T),
    rater == rater_i & coding == 1 ~ sample(c(1, 0),
                                      size = n(),
                                      prob = c(rater_i_accuracy,
                                               1 - rater_i_accuracy),
                                      replace = T),
    TRUE ~ coding
  )) %>%
  ungroup() %>%
    select(-rater_i)

  return(example_data)
}
```

```
# generate simulated datasets using all possible combinations of inputs
```

```r
# set inputs
input_list <- expand_grid(n_subjects = c(50, 100, 500),
                          n_raters = c(3, 6, 10, 20),
                          degree_overlap = c(1, 2, 5, 10),
                          rater_i_accuracy = seq(.01, 1, .01))

# simulate data and return kripp alpha for all sets of inputs
kripp_out <- input_list %>%
  future_pmap(make_ex_data,
              .options = furrr_options(seed = TRUE)) %>%
  future_map(~ura::irr_stats(.,
                         rater_column = 'rater',
                         subject_column = 'subject',
                         coding_column = 'coding'),
             .options = furrr_options(seed = TRUE))

names(kripp_out) <- unite(input_list,
                          "input_name",
                          everything()) %>%
                          pull(input_name)

# turn into dataframe with named input columns
kripp_out_df <- kripp_out %>%
  bind_rows(.id = 'id') %>%
  filter(statistic == "Krippendorf's Alpha") %>%
  mutate(n_subjects = as.numeric(str_extract(id,
                                             "^\\d*"))) %>%
  mutate(n_raters = as.numeric(str_remove_all(str_extract(id,
                                               "_\\d*_"),
                                    "_"))) %>%
  mutate(id_2 = str_remove(str_extract(id,
                           "_\\d*_.*$"),
                          "_\\d*_")) %>%
  mutate(degree_overlap = as.numeric(str_extract(id_2,
                                                 "^\\d*"))) %>%
  mutate(rater_i_accuracy = as.numeric(str_extract(id_2,
                                                   "(?<=_).*$"))) %>%
  select(-id,
         -id_2)
```

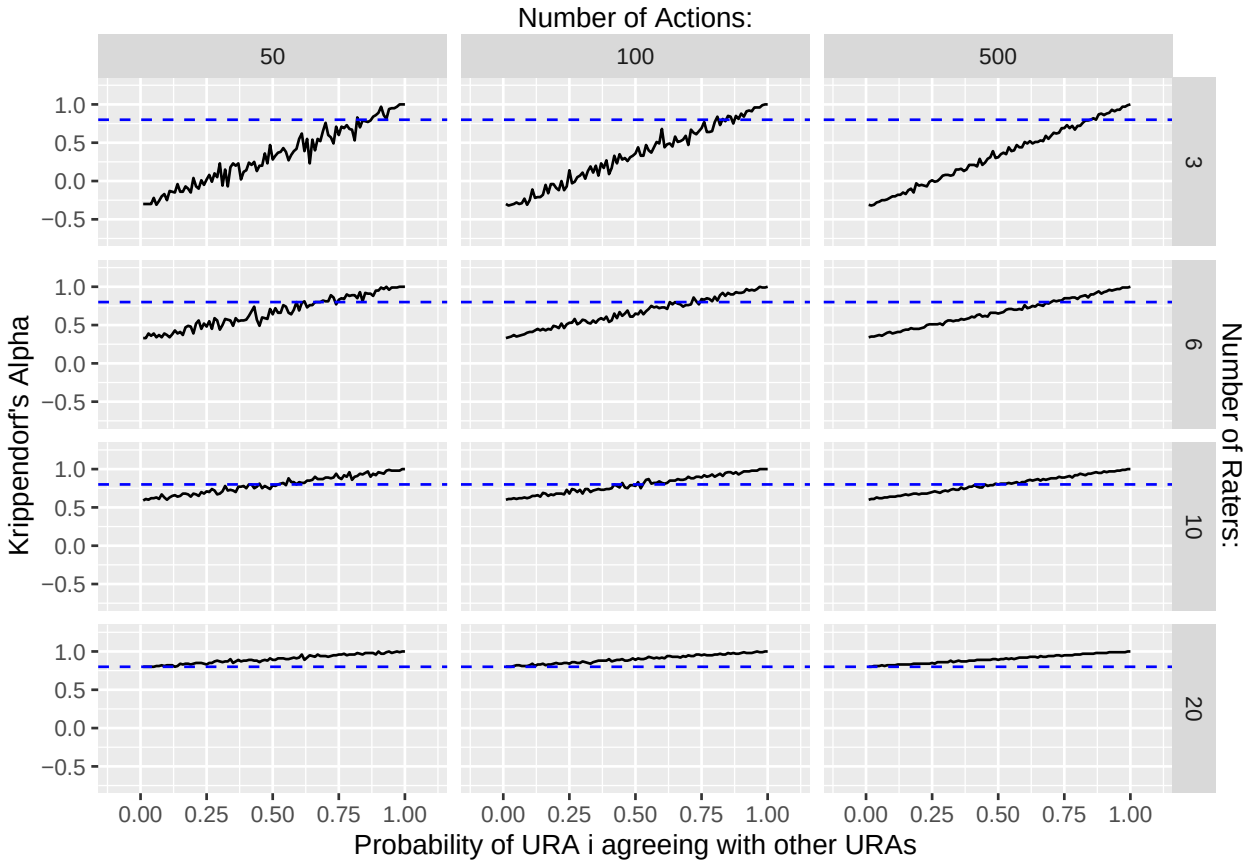
```r
# create plotting function to plot kripp alpha against accuracy for different
#   input sets
plot_kripp <- function(plot_degree_overlap,
                       plot_n_subjects = 100,
                       data = kripp_out_df,
                       si = FALSE) {

  # figure for paper
  if(isFALSE(si)) {
    out <- kripp_out_df %>%
      filter(degree_overlap == {{plot_degree_overlap}}) %>%
      filter(n_subjects == {{plot_n_subjects}}) %>%
      ggplot() +
      geom_line(aes(x = rater_i_accuracy,
```

```
                  y = value)) +
    facet_wrap(~n_raters,
               labeller = as_labeller(function(string,
                                               suffix = " URAs") paste0(string,
                                                                        suffix))) +
    geom_hline(aes(yintercept = .8),
                   color = 'blue',
                   linetype = 'dashed') +
    scale_y_continuous(limits = c(-.75, 1.25),
                       breaks = seq(-.5, 1, .5)) +
    scale_x_continuous(limits = c(-.1, 1.1),
                       breaks = seq(0, 1, .25)) +
    labs(y = "Krippendorf's Alpha",
         x = "Probability of URA i agreeing with other URAs")
} else {

  # figure for supplementary information
  if(plot_degree_overlap == 1) {
  si_caption <- str_wrap(str_glue("Note: This figure shows how one poorly performing URA can affect t
                                                  width = 100)
} else {
  si_caption <- str_wrap(str_glue("Note: This figure shows how one poorly performing URA can affect t
                                                  width = 100)
}

  out <- kripp_out_df %>%
    filter(degree_overlap == {{plot_degree_overlap}}) %>%
    ggplot() +
    geom_line(aes(x = rater_i_accuracy,
                  y = value)) +
    geom_hline(aes(yintercept = .8),
                   color = 'blue',
                   linetype = 'dashed') +
    facet_grid(n_raters ~ n_subjects) +
    scale_y_continuous(limits = c(-.75, 1.25),
                       breaks = seq(-.5, 1, .5),
                       sec.axis = sec_axis(~ . ,
                                           name = "Number of Raters:",
                                           breaks = NULL,
                                           labels = NULL)) +
  scale_x_continuous(limits = c(-.1, 1.1),
                       breaks = seq(0, 1, .25),
                       sec.axis = sec_axis(~ . ,
                                           name = "Number of Actions:",
                                           breaks = NULL,
                                           labels = NULL)) +
    theme(plot.caption = element_text(hjust = 0)) +
    labs(y = "Krippendorf's Alpha",
         x = "Probability of URA i agreeing with other URAs",
         title = "The Effects of One Poorly Performing URA on IRR",
         subtitle = str_glue("With degree_overlap == {plot_degree_overlap}"),
         caption = si_caption)
}
```

```
  return(out)
}

# figure used in paper
#BeRn::set_my_theme()
#ggsave("submission-figures/kripp_alpha_prelim.pdf",
#       plot = plot_kripp(2),
#       device = cairo_pdf,
#       height = 6, width = 7)
```

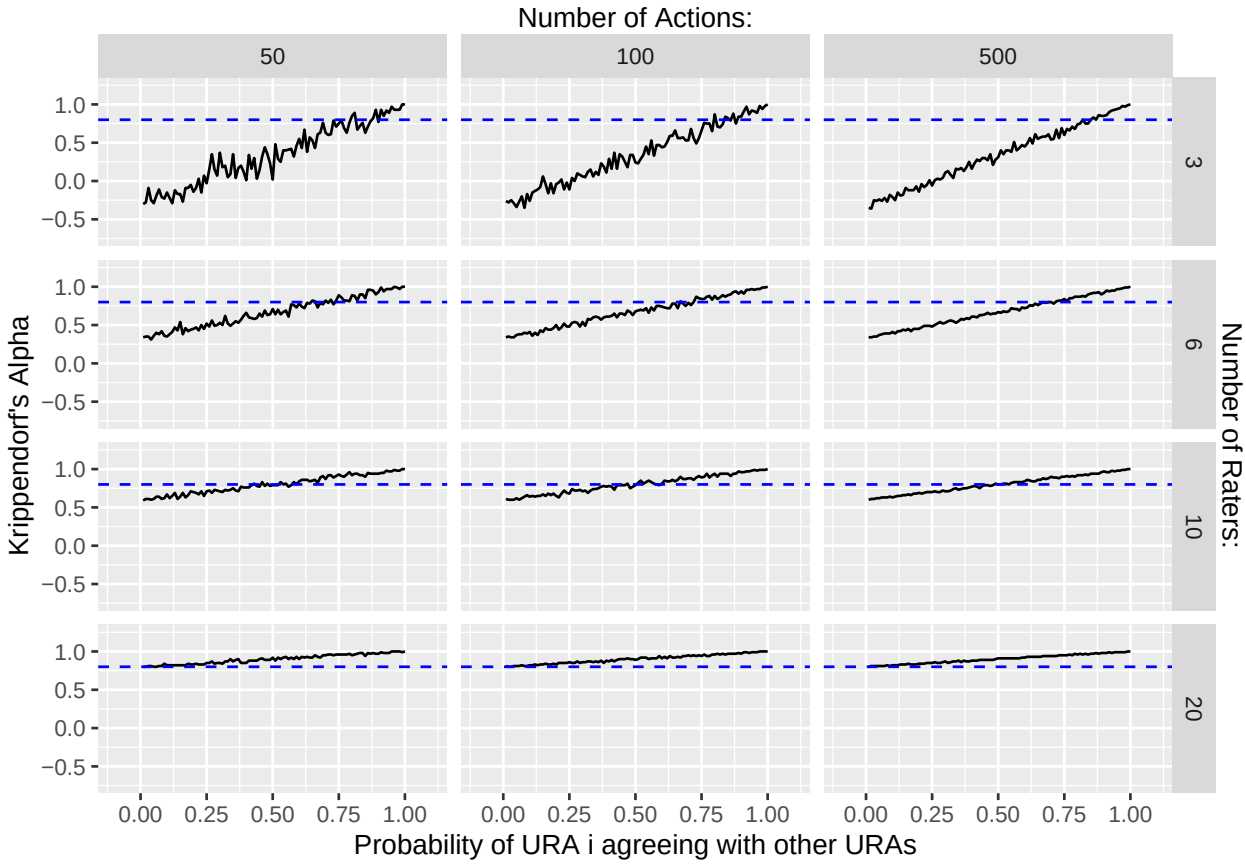
2) Additional simulation figures

```
plot_kripp(1,
           si = TRUE)
```



Note: This figure shows how one poorly performing URA can affect the IRR of a dataset. Each facet shows, for a given number of URAs and actions, how the (in)accuracy of one URA affects the Krippendorf Alpha of the final dataset. The blue dashed line at 0.8 represents the conventional level of reliability for Krippendorf's Alpha. The data are generated by randomly assigning a given number of actions to a given number of URAs. Variation in the number of actions is shown in the horizonal facets, while variation in the number of URAs is shown in the vertical facets. The actions are drawn from a set of 100 actions. Therefore, all of the actions are coded by more than one URA and thus suitable for IRR testing. The URAs always agree on actions that are coded by more than one URA, with the exception of URA i, who agrees with the others with some probability (shown on horizontal axis). If the probability of URA i agreeing with the other URAs is 1, then all the URAs assign the same coding to actions. If the probability of URA i agreeing with the other URAs is 0, then URA i assigns the opposite coding as the other URAs.

```
plot_kripp(2,
           si = TRUE)
```

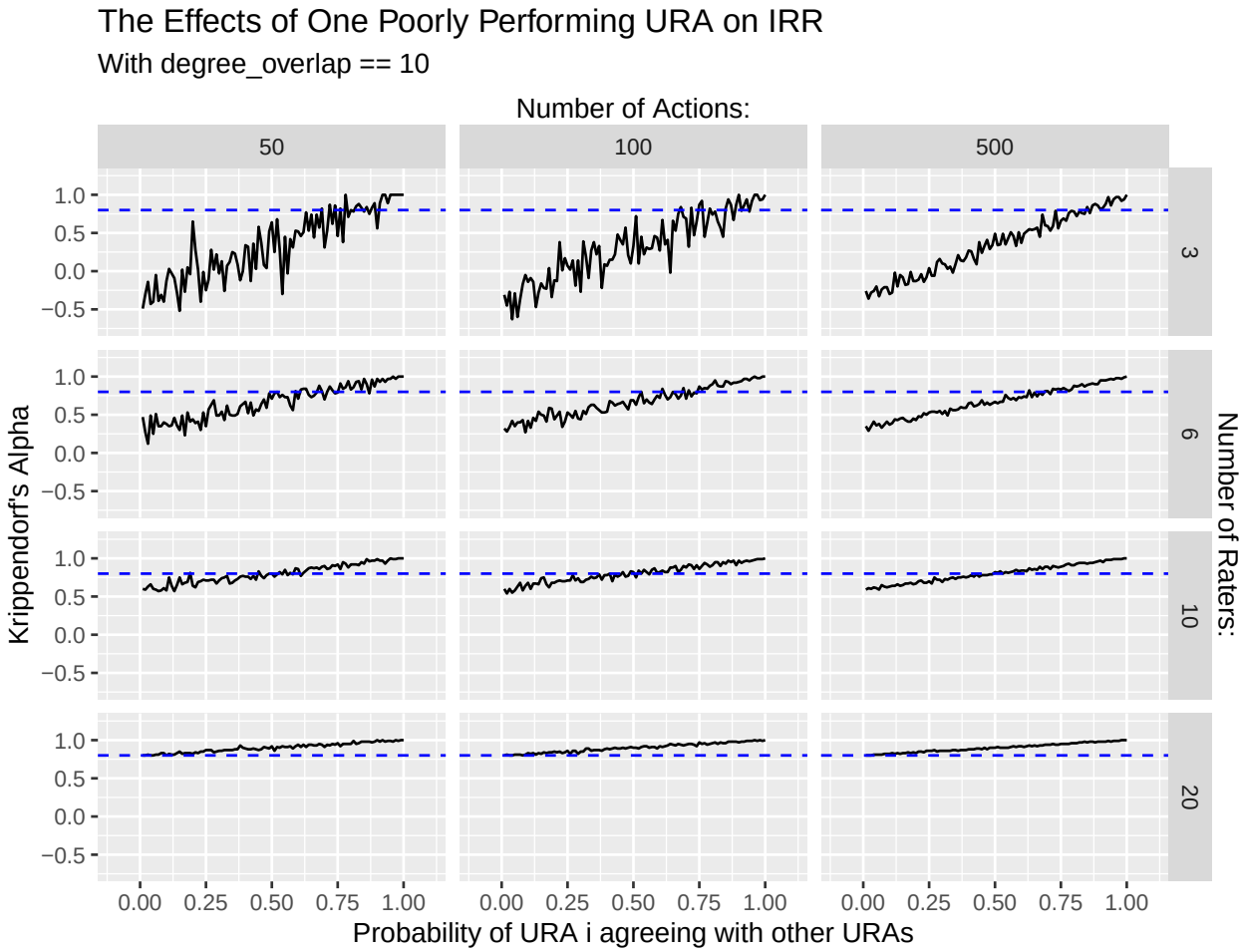


Note: This figure shows how one poorly performing URA can affect the IRR of a dataset. Each facet shows, for a given number of URAs and actions, how the (in)accuracy of one URA affects the Krippendorf Alpha of the final dataset. The blue dashed line at 0.8 represents the conventional level of reliability for Krippendorf's Alpha. The data are generated by randomly assigning a given number of actions to a given number of URAs. Variation in the number of actions is shown in the horizonal facets, while variation in the number of URAs is shown in the vertical facets. The actions are drawn from a set of 200 actions. Therefore, some but not all of the actions are coded by more than one URA and thus suitable for IRR testing. The URAs always agree on actions that are coded by more than one URA, with the exception of URA i, who agrees with the others with some probability (shown on horizontal axis). If the probability of URA i agreeing with the other URAs is 1, then all the URAs assign the same coding to actions. If the probability of URA i agreeing with the other URAs is 0, then URA i assigns the opposite coding as the other URAs.

```
plot_kripp(5,
           si = TRUE)
```

Note: This figure shows how one poorly performing URA can affect the IRR of a dataset. Each facet shows, for a given number of URAs and actions, how the (in)accuracy of one URA affects the Krippendorf Alpha of the final dataset. The blue dashed line at 0.8 represents the conventional level of reliability for Krippendorf's Alpha. The data are generated by randomly assigning a given number of actions to a given number of URAs. Variation in the number of actions is shown in the horizonal facets, while variation in the number of URAs is shown in the vertical facets. The actions are drawn from a set of 500 actions. Therefore, some but not all of the actions are coded by more than one URA and thus suitable for IRR testing. The URAs always agree on actions that are coded by more than one URA, with the exception of URA i, who agrees with the others with some probability (shown on horizontal axis). If the probability of URA i agreeing with the other URAs is 1, then all the URAs assign the same coding to actions. If the probability of URA i agreeing with the other URAs is 0, then URA i assigns the opposite coding as the other URAs.

```
plot_kripp(10,
           si = TRUE)
```



Note: This figure shows how one poorly performing URA can affect the IRR of a dataset. Each facet shows, for a given number of URAs and actions, how the (in)accuracy of one URA affects the Krippendorf Alpha of the final dataset. The blue dashed line at 0.8 represents the conventional level of reliability for Krippendorf's Alpha. The data are generated by randomly assigning a given number of actions to a given number of URAs. Variation in the number of actions is shown in the horizonal facets, while variation in the number of URAs is shown in the vertical facets. The actions are drawn from a set of 1000 actions. Therefore, some but not all of the actions are coded by more than one URA and thus suitable for IRR testing. The URAs always agree on actions that are coded by more than one URA, with the exception of URA i, who agrees with the others with some probability (shown on horizontal axis). If the probability of URA i agreeing with the other URAs is 1, then all the URAs assign the same coding to actions. If the probability of URA i agreeing with the other URAs is 0, then URA i assigns the opposite coding as the other URAs.

3) Comparing URA reliability using the `ura` package

To show the R package in action, consider the following example: Six URAs are coding 200 unilateral actions for whether they receive coverage by the media. Each of the six URAs codes 100 actions, so many actions are coded by more than one URA and thus suitable for IRR diagnostic testing. Here is a snapshot of 20 rows from the example dataset:

```
# make example data and show first 20 rows
example_data <- make_ex_data(n_subjects = 100,
                             n_raters = 6,
                             degree_overlap = 2,
                             rater_i_accuracy = .5)

example_data <- example_data %>%
```

```
  rename(ura = rater,
         action = subject,
         coverage = coding)

print(example_data,
      n = 20)
```

```
## # A tibble: 600 x 3
##      ura action coverage
##    <dbl>  <int>    <dbl>
##  1     1    142        1
##  2     2    148        1
##  3     3    170        1
##  4     4     73        0
##  5     5    199        0
##  6     6     12        0
##  7     1     51        0
##  8     2    140        1
##  9     3     66        0
## 10     4    148        1
## 11     5      1        1
## 12     6    192        0
## 13     1    152        0
## 14     2     98        0
## 15     3     20        0
## 16     4     59        1
## 17     5    169        1
## 18     6    139        1
## 19     1     58        1
## 20     2     68        1
## # i 580 more rows
```

The first thing a researcher is likely interested in upon seeing these data is how reliable they are. Did the URAs code the actions in a consistent, precise way? Or were they just guessing or inaccurately ascribing coverage to the actions? The `irr_stats()` function provides an easy-to-use means of calculating IRR statistics, such as percentage agreement and Krippendorf's Alpha. A wrapper around lower-level functions in the `irr` package, `irr_stats()` performs all of the data pre-processing steps under the hood. All the user has to do is specify the data and the respective column names. The function returns a dataframe with the respective IRR diagnostic values and the number of actions used to calculate the statistics (i.e., the number of actions that were coded by more than one URA).

In our example, the URAs agree with one another on the coding of the 183 actions coded multiple times 73.8 percent of the time. The Krippendorf Alpha diagnostic, which is a more rigorous metric for evaluating reliability that takes into account that raters might agree solely due to chance, is .67. A Krippendorf Alpha value of .8 and above is usually seen as a conventional marker of high reliability, so the results seem OK but not great.

```
irr_stats(example_data,
          rater_column = 'ura',
          subject_column = 'action',
          coding_column = 'coverage')
```

```
## # A tibble: 2 x 3
##   statistic            value n_subjects
##   <chr>                <dbl>      <int>
```

```
## 1 Percentage agreement 73.8         183
## 2 Krippendorf's Alpha   0.67        183
```

Since these statistics measure group-level reliability, they mask the performance of individual members. As shown in Figure 2 and the above simulations, one poorly-performing URA can dramatically affect overall reliability. As such, `ura` provides a simple interface for digging further into these IRR statistics by seeing how each URA performed relative to one another. In particular, the `rater_agreement()` function returns the percent share of a given URA's actions that were coded in the same way by other URAs coding the same actions:

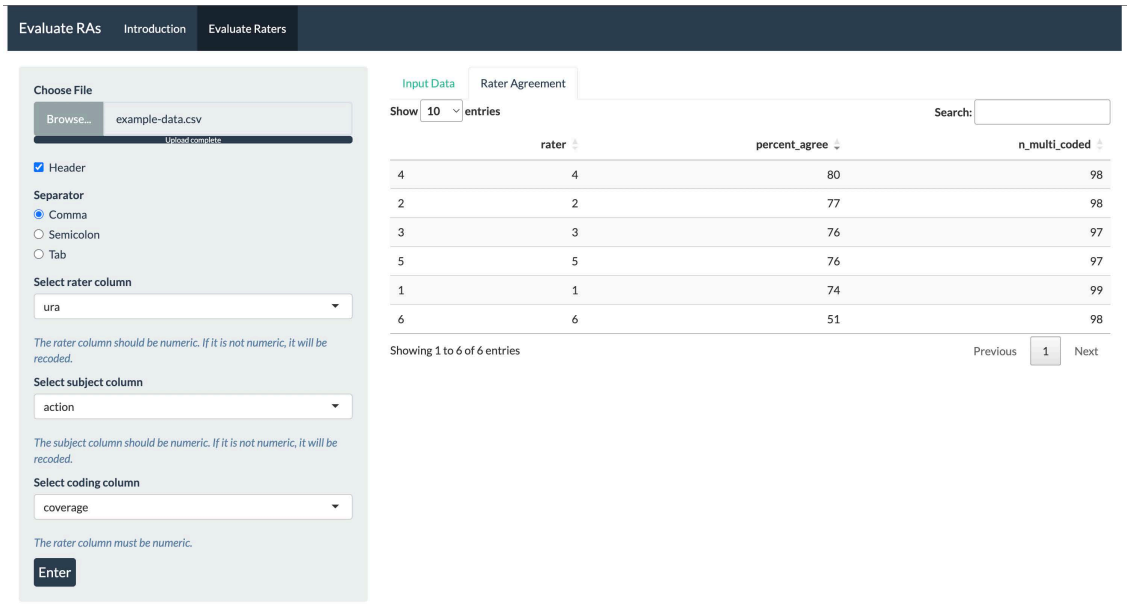
```
rater_agreement(example_data,
                rater_column = 'ura',
                subject_column = 'action',
                coding_column = 'coverage')
```

```
## # A tibble: 6 x 3
##   rater percent_agree n_multi_coded
##   <int>         <dbl>         <int>
## 1     4            80            98
## 2     2            77            98
## 3     3            76            97
## 4     5            76            97
## 5     1            74            99
## 6     6            51            98
```

The output above shows that URA 6 is performing especially poor relative to the rest of the team. Of the actions coded by URA 6, only 51% were coded the same by other URAs. This deviation from the rest of the team is in fact entirely responsible for the relatively low overall IRR statistics returned by `irr_stats()`. `example_data` was generated such that all URAs agree with each other at all times, except for URA 6 who only agrees with the others with a probability of .5. Via `ura`, this poor performance can be quickly uncovered and addressed before it affects the entire project's reliability.

4) Comparing URA reliability using web-based application

It is also possible to examine the relative reliability of coders without using R via a web-based application. This application implements the `rater_agreement()` function, but the user only needs to upload the data and select the necessary variable names. It is available via this link. Below is a screenshot of me calculating the percentage agreement of the URAs in `example_data` via the web application. It only takes a few steps to use the application. The researcher first loads a csv file containing the coding dataset into the application. Then, they select the rater, subject, and coding columns via dropdown boxes. In the example above, the rater column is `ura`, the subject column is `action`, and the coding column is `coverage`. After pressing `Enter` the input dataset will appear on the right-hand side of the screen. Clicking on the `Rater Agreement` tab will pull up the reliability data, which is identical to the output from the `rater_agreement()` function above.


Evaluate RAs
Introduction
Evaluate Raters
Choose File
Browse...
example-data.csv
Upload complete
Header
Separator
Comma
Semicolon
Tab
Select rater column
ura
The rater column should be numeric. If it is not numeric, it will be recoded.
Select subject column
action
The subject column should be numeric. If it is not numeric, it will be recoded.
Select coding column
coverage
The rater column must be numeric.
Enter
Input Data
Rater Agreement
Show 10 entries
Search:
rater
percent_agree
n_multi_coded
4 4 80 98
2 2 77 98
3 3 76 97
5 5 76 97
1 1 74 99
6 6 51 98
Showing 1 to 6 of 6 entries
Previous
1
Next