# Score TDS Qualtrics Data

*John Flournoy*

*2018-01-23*

## Contents



This presents both a walk-through for how you could go about adding a new questionnaire to be scored, and also should provide up-to-date descriptives and scored scales every time you recompile this Rmd document.

It would be nice if this were something completely automated. Unfortunately, the nature of data that is still under collection, and the fact that there are a lot of moving pieces means that something totally automated is likely to break pretty easily (at least, something automated coded by me). So this document will take a highly modular approach so hopefully if something goes wrong, you can see more exactly where that happens. I'll try to explain every step pretty verbosely too. You should probably be reading this document in R Studio. Make sure you're up to date with upgrades.

### Setting options

If you're reading the compiled HTML, you won't see the options below, because we've writtent `warning=F,echo=F,message=F,error=F` in the head of the chunk. You can change this by replacing the "F"s with "T"s.

### Install the `scorequaltrics` package

We need to have the scorequaltrics package installed. I wrote this so that I can maintain helpful functions related to scoring.

```
#this chunk won't evaluate. If you need to
#install the package, run this by hand.
devtools::install_github('jflournoy/qualtrics')
```

### Accessing qualtrics data

You also need to have a token in a YAML formatted file for accessing qualtrics via the API. It's formatted like:

```
user: username
token: apitoken
```

Once we've loaded this, we can get a list of questionnaires.

```r
library(scorequaltrics)
```

```
## Warning: replacing previous import 'data.table::first' by 'dplyr::first'
## when loading 'scorequaltrics'
```

```
## Warning: replacing previous import 'data.table::between' by
## 'dplyr::between' when loading 'scorequaltrics'
```

```
## Warning: replacing previous import 'data.table::last' by 'dplyr::last' when
## loading 'scorequaltrics'
```

```r
library(ggplot2)
library(dplyr)
library(tidyr)

credentials <- scorequaltrics::creds_from_file(cred_file_location)

rawSurveys <- scorequaltrics::get_surveys(credentials)
rawSurveysTDS <- filter(rawSurveys, grepl('.*(TDS1|TDS2|TDS3).*',SurveyName))

knitr::kable(arrange(select(rawSurveysTDS, SurveyName), SurveyName))
```

| SurveyName |
|---|
| QSupport TDS2 Session 3 - Parent - Copy - Copy |
| TDS1 AND TDS3 PSQI |
| TDS1 CBCL - Post |
| TDS1 CBCL - Pre |
| TDS1 PDS (Session2) - Post |
| TDS1 Saliva Questions - Post |
| TDS1 Saliva Questions - Pre |
| TDS1 Session 1 & 2 Makeups |
| TDS1 Session 1 - Post |
| TDS1 Session 1 - Pre |
| TDS1 Session 2 A - Post |
| TDS1 Session 2 A - Pre |
| TDS1 Session 2 B - Post |
| TDS1 Session 2 B - Pre |
| TDS1 Session 2 B - SHORTENED_A |
| TDS1 Session 2 B - SHORTENED_B |
| TDS1, Session 3 - Child |
| TDS1 Session 3 Child *A* |
| TDS1 Session 3 Child *B* |
| TDS1, Session 3 - Parent |
| TDS1, Session 3 - Parent part 2 |
| TDS1 Spinning Wheel Game - New |
| TDS1 Spinning Wheel Game - Post |
| TDS2 CBCL |
| TDS2 - PDS (Session 2) |
| TDS2 Saliva Questions |
| TDS2 Session 1 |
| TDS2 Session 2 A |
| TDS2 Session 2 B |
| TDS2 Session 2 B - SHORTENED_A |
| TDS2 Session 2 B - SHORTENED_B |
| TDS2 Session 3 - Child |

| SurveyName |
|---|
| TDS2 Session 3 - KSRQ & SAQ |
| TDS2 Session 3 - Parent |
| TDS2 Session 3 - Parent - Copy |
| TDS2 Spinning Wheel Game |

We have a lot of different questionnaires from different samples and different sessions. For simplicity, and to aid in diagnosing any problems, we can proceed through each sample and each wave of data collection. *Note*, however, that if we ensured that naming conventions were consistent across all questionnaires and rubrics, and if we had accurate session dates attached, we could score everything in one fell swoop.

## TDS 2

This is the first sample collected, so we can begin here. I'll demonstrate in this how to do a single massive data scoring. See below for how you can get more information about scales that have been constructed in a psychometric tradition, and that therefore are easy to evaluate using standard reliability metrics.

### Wave 1

First, we download the data for the surveys we want.

```
tds2_wave1_surveys <- rawSurveysTDS %>%
    filter(grepl('TDS2 (Session [12]|CBCL|- PDS)', SurveyName))

print(tds2_wave1_surveys$SurveyName)
```

```
## [1] "TDS2 - PDS (Session 2)"         "TDS2 CBCL"
## [3] "TDS2 Session 2 B"               "TDS2 Session 2 B - SHORTENED_B"
## [5] "TDS2 Session 2 A"               "TDS2 Session 1"
## [7] "TDS2 Session 2 B - SHORTENED_A"
```

```
tds2_wave1_long <- scorequaltrics::get_survey_data(tds2_wave1_surveys,
                                                   credentials,
                                                   pid_col = pid_column_name)

dim(tds2_wave1_long)
```

```
## [1] 86051     4
```

```
names(tds2_wave1_long)
```

```
## [1] "SID"         "item"        "value"       "survey_name"
```

The resulting data frame should have a lot of rows (the first part of the output of `dim`) and 4 columns.

```
## It looks like all is in order here. Note that the PID column is named "SID".
```

Before doing any scoring, we should take care of all the complex response recoding that may be specified. So We'll load all the response recoding rubrics and apply those. It's important that you pass the full path of the file to the next function, so if you use `dir` to collect filenames as I do below, make sure you set `full.names = TRUE`.

```
dir(file.path(tds2_wave1_rubric_dir), pattern = '.*response_recoding.*.csv')
```

```
## [1] "PAL2_response_recoding.csv"
## [2] "SES_response_recoding.csv"
## [3] "YRBS_response_recoding_TDS2_session_2.csv"
```

```
#You should see a result below -- if not, the path is likely wrong.

tds2_wave1_recoding_rubrics <- data.frame(file = dir(file.path(tds2_wave1_rubric_dir),
                                                     pattern = '.*response_recoding.*.csv',
                                                     full.names = TRUE))

tds2_wave1_recoding_data_long <- scorequaltrics::get_rubrics(tds2_wave1_recoding_rubrics,
                                                             type = 'recoding')

tds2_wave1_long_recoded <- scorequaltrics::recode_responses(tds2_wave1_long,
                                                            tds2_wave1_recoding_data_long)
```

Now let's load in the scoring rubrics.

```
tds2_wave1_scoring_rubrics <- data.frame(file = dir(file.path(tds2_wave1_rubric_dir),
                                                    pattern = '.*scoring_rubric.*.csv',
                                                    full.names = TRUE))

tds2_wave1_scoring_data_long <- scorequaltrics::get_rubrics(tds2_wave1_scoring_rubrics,
                                                            type = 'scoring')

head(tds2_wave1_scoring_data_long[, -1])
```

```
## # A tibble: 6 x 9
##   data_file_name scale_name column_name transform reverse min   max
##   <chr>          <chr>      <chr>       <chr>     <chr>   <chr> <chr>
## 1 TDS2_Session_1 ACE        ACE_1       0         0       0     0
## 2 TDS2_Session_1 ACE        ACE_2       0         0       0     0
## 3 TDS2_Session_1 ACE        ACE_3       0         0       0     0
## 4 TDS2_Session_1 ACE        ACE_4       0         0       0     0
## 5 TDS2_Session_1 ACE        ACE_5       0         0       0     0
## 6 TDS2_Session_1 ACE        ACE_6       0         0       0     0
## # ... with 2 more variables: scored_scale <chr>, include <chr>
```

### Cleaning

We can make sure we clean out duplicate responses which will help later with ensuring that scale scores are calculated from teh correct subset of items. This is a point at which, if there is something funky going on, you'll want to investigate it and make a decision. For example, if a participant has two conflicting answers to the same question for the same wave, it's likely that a small investigation should commence.

We can also ensure that we're only scoring data for participants with the correct ID numbers. The line in the middle of the first call, `filter(grepl('[1234]\\d\\d', SID))`, ensures we only keep people with ID's starting with "1".

Before we do that, we can ensure that we're only keeping the data in the scoring rubrics in the first place.

```
tds2_wave1_long_recoded_nodupes <- tds2_wave1_long_recoded %>%
    get_items_in_rubric(tds2_wave1_scoring_data_long) %>%
    filter(grepl('1\\d\\d', SID)) %>%
    scorequaltrics::clean_dupes(pid_col = 'SID')
```

If you get "NAs introduced by coercion" it probably means that one of the rubrics references a column

that has text input that is not transformable into a number. For example, if the questionnaire asks for ethnicity and someone writes in "White" it is not possible to turn that into a score to be used in a scale calculation (but there's a rubric that thinks it can). We can check that by using the function `scorequaltrics::get_uncoercibles()`.

```
tds2_wave1_uncoer <- tds2_wave1_long_recoded %>%
    get_items_in_rubric(tds2_wave1_scoring_data_long) %>%
    filter(grepl('[1234]\\d\\d', SID)) %>%
    scorequaltrics::get_uncoercibles() %>%
    distinct(item, value)

head(tds2_wave1_uncoer, 10)
```

```
## Empty data.table (0 rows) of 2 cols: item,value
```

```
unique(tds2_wave1_uncoer$item)
```

```
## character(0)
```

Now we can look at what rubrics have those items, if any.

```
tds2_wave1_scoring_data_long %>%
    filter(column_name %in% unique(tds2_wave1_uncoer$item),
           include %in% c(1, "1", "sum", "prod")) %>%
    ungroup() %>%
    select(scale_name, scored_scale, column_name, include)
```

```
## # A tibble: 0 x 4
## # ... with 4 variables: scale_name <chr>, scored_scale <chr>,
## #   column_name <chr>, include <chr>
```

If the above two chunks didn't result in output, we're good!

```
#Check that dropped values weren't ambiguous
tds2_wave1_long_recoded_nodupes %>%
  filter(dropped) %>%
  group_by(SID, item) %>%
  summarize(noinfo = all(length(unlist(old.value)) < 1)) %>%
  ungroup() %>%
  summarize(n_with_info = sum(!noinfo))
```

```
## # A tibble: 1 x 1
##   n_with_info
##         <int>
## 1          14
```

```
tds2_wave1_long_recoded_nodupes %>%
    filter(dropped) %>%
    group_by(SID, item) %>%
    filter(!all(length(unlist(old.value)) < 1)) %>%
    mutate(old.value = paste(old.value, collaps = ' ')) %>%
    knitr::kable()
```

| SID | item | value | survey_name | old.value | dropped |
|---|---|---|---|---|---|
| 109 | YRBS_10 | NA | TDS2 Session 2 B | c(1, 0) | TRUE |
| 124 | PDS_F3 | NA | TDS2 Session 1 | c(4, 3) | TRUE |
| 125 | YRBS_10 | NA | TDS2 Session 2 B | c(1, 0) | TRUE |
| 159 | YRBS_10 | NA | TDS2 Session 2 B | c(1, 0) | TRUE |

| SID | item | value | survey_name | old.value | dropped |
|---|---|---|---|---|---|
| 189 | YRBS_10 | NA | TDS2 Session 2 B | c(1, 0) | TRUE |
| 190 | YRBS_10 | NA | TDS2 Session 2 B | c(1, 0) | TRUE |
| 196 | BIS_1 | NA | TDS2 Session 2 A | c(3, 2) | TRUE |
| 196 | BIS_10 | NA | TDS2 Session 2 A | c(4, 2) | TRUE |
| 196 | BIS_14 | NA | TDS2 Session 2 A | c(2, 3) | TRUE |
| 196 | BIS_2 | NA | TDS2 Session 2 A | c(3, 2) | TRUE |
| 196 | BIS_6 | NA | TDS2 Session 2 A | c(2, 3) | TRUE |
| 196 | BIS_7 | NA | TDS2 Session 2 A | c(4, 3) | TRUE |
| 196 | SSS_3 | NA | TDS2 Session 2 A | c(1, 0) | TRUE |
| 196 | SSS_4 | NA | TDS2 Session 2 A | c(1, 0) | TRUE |

For now, if there are ambiguous entries, I'm going to ingore them. The value to be used for scoring is set to NA – we have to treat that data as missing since the responses are in conflict.

### Scoring

There are a few different options for scoring questionnaires. First, we can provide a rubric and data to `scorequaltrics::score_questionnaire(dataDF, rubricsDF, psych = TRUE)`, which will use the `psych` package to do the scoring. This has the advantage that you get back a lot of information about the measurement quality of the scale, but it only works for scales that follow certain psychometric principles (e.g., each item is rated on a continuous scale, and is an indicator of a latent construct). It won't work well for other kinds of data (like scales where you want to know the number of risky behaviors, for example).

The second option is to use `scorequaltrics::score_step_one_and_two(dataDF, rubricsDF)` which was created to take care of several special cases for the TDS project questionnaires. The RPI, and RSQ both require special handling because of their idiosyncratic questionnaire design.

```
tds2_wave1_scored <- scorequaltrics::score_step_one_and_two(tds2_wave1_long_recoded_nodupes,
                                                            tds2_wave1_scoring_data_long)

set.seed(9567)
tds2_wave1_scored %>% ungroup %>%
    dplyr::sample_n(size = 10) %>%
    select(-SID)
```

```
## # A tibble: 10 x 6
##    scale_name        scored_scale       score     n_items n_missing method
##    <chr>             <chr>              <chr>       <int>     <int> <chr>
##  1 CBCL              somatic_complaints 0.090909~      11         0 1
##  2 CARE-R Expected ~ risky_car          1               2         0 1
##  3 RSQ_part2         rsq_mean_anxious_~ 19.6           10         0 1
##  4 PAL-2             pal2_antisocial    1.090909~      11         0 1
##  5 CARE-R Willingne~ risky_sex_regular~ 1               5         0 1
##  6 SPSRQ-S           sensitivity_rewar~ 0.1            10         0 1
##  7 UPPS-P            pos_urgency        2.357142~      14         0 1
##  8 PEQ-R             prosocial_to_me    3.6             5         0 1
##  9 RPI_part2         rpi_mean           3.444444~       9         1 1
## 10 CARE-R Social     care_soc_not_risk~ 3               3         0 1
```

One thing missing still is the Pubertal Development Scale scored via the Shirtcliff method, so I'll calculate that now with the special function `scorequaltrics::score_pdss()`

```
tds2_wave1_scored_pdss <- scorequaltrics::score_pdss(tds2_wave1_long_recoded_nodupes,
                          gender_mix = pdss_gender_mix,
                          gendercode = pdss_gender_code)
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

That hopefully went off without a hitch. Now we can examine the scales we've scored, and along the way, see how we can get more information about these scales when we are able to use the `psych` package scoring function.

### Examine Wave 1 Scores

We can look at basic descriptive statistics using a function I'll write and REPLACE THE NAME OF HERE.

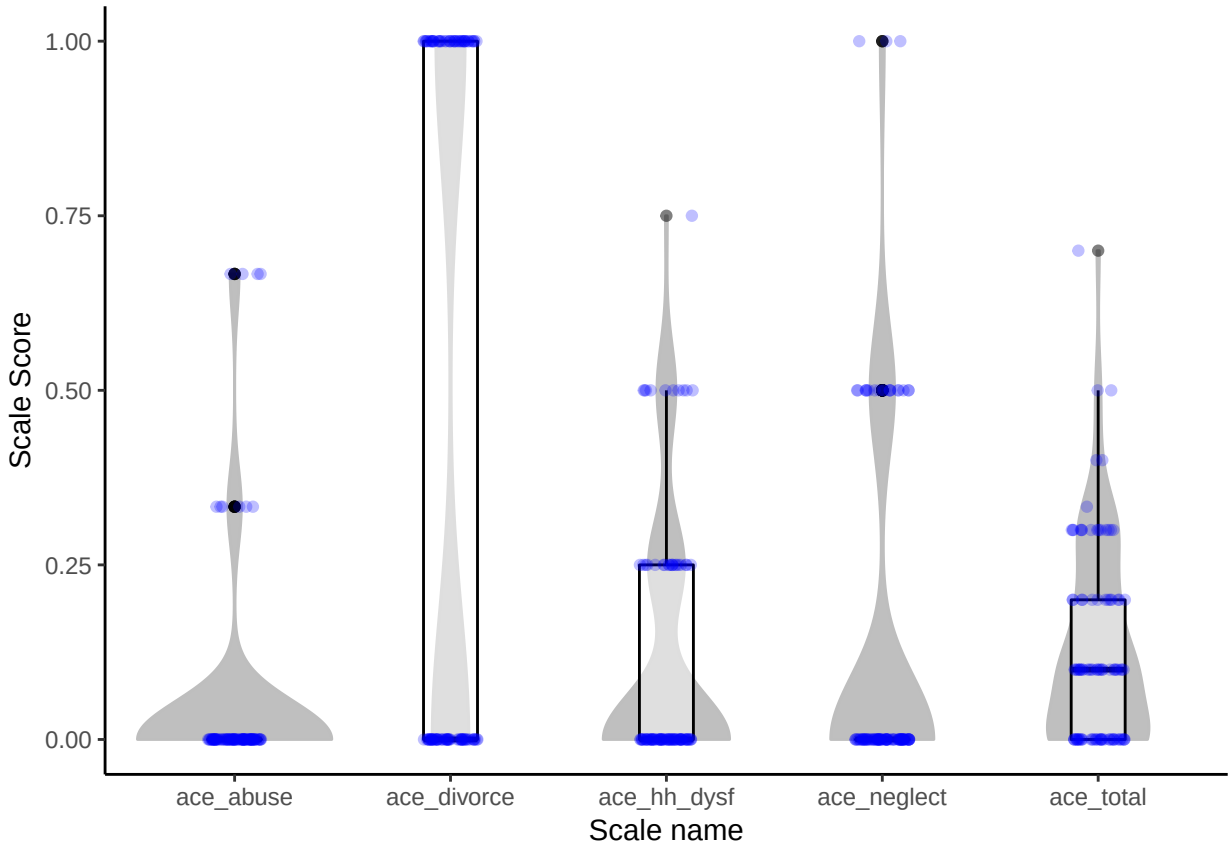
First, what scales do we have available?

```
tds2_wave1_scored %>%
    ungroup() %>%
    distinct(scale_name) %>%
    knitr::kable()
```

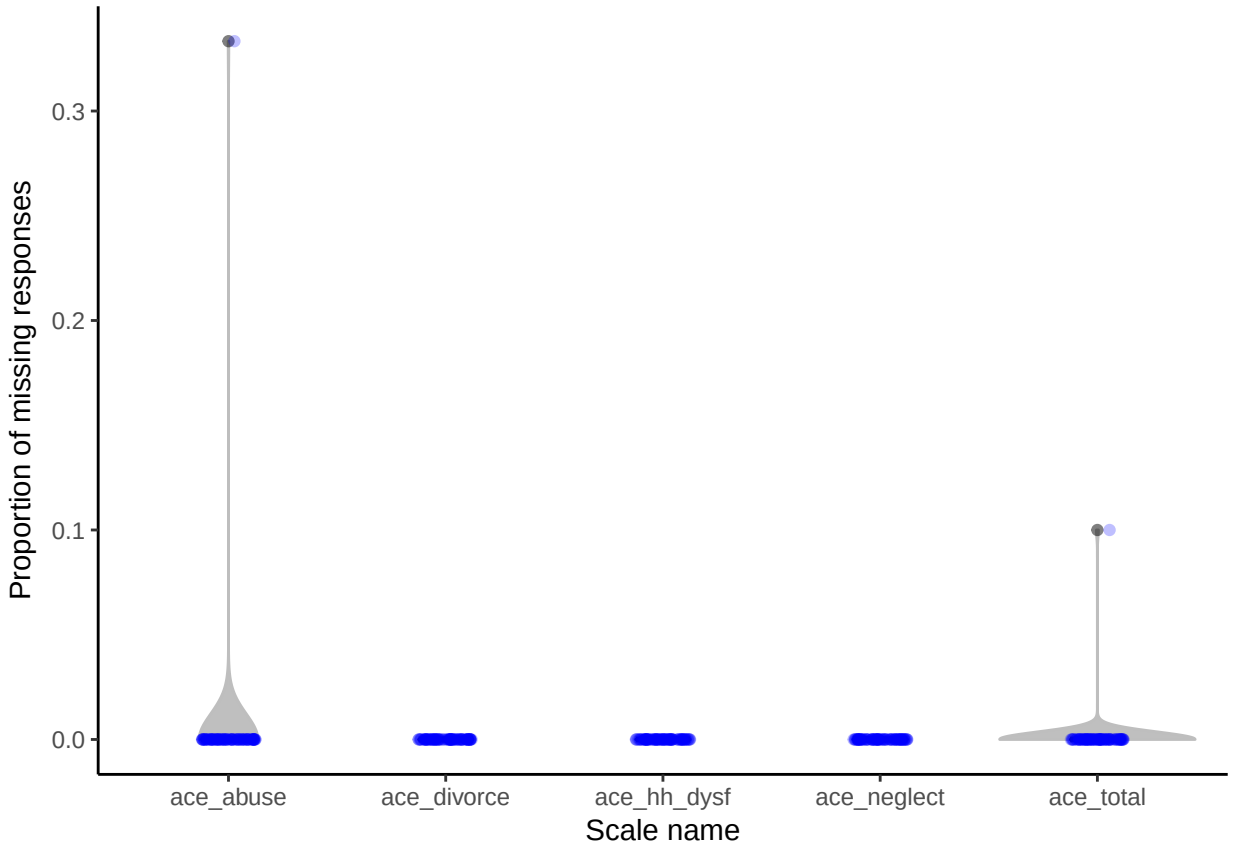
| scale_name |
|---|
| ACE |
| BFNE |
| BIS-15 |
| Brief SCARED |
| BSSS |
| CARE-R Expected Involvement |
| CARE-R Social |
| CARE-R Willingness to Engage |
| CBCL |
| CES-DC |
| MSSSS |
| NTS |
| PAL-2 |
| PDS |
| PEQ-R |
| SES |
| SPSRQ-S |
| UPPS-P |
| YRBS |
| RPI_part2 |
| RSQ_part2 |

**ACE**

```
plot_tds_scale <- function(aDF, scale_regx, type = 'score', by_gender = FALSE, gender_var = NA){
    aDF <- aDF %>%
        filter(grepl(scale_regx, scale_name)) %>%
        mutate_at(vars(score, n_missing, n_items), as.numeric)

    if(length(unique(aDF$scale_name)) > 1){
        warning('Matched multiple scales: "', paste(unique(aDF$scale_name), collapse = '", "'), '".')
    }

    if (type == 'score'){
        colname <- 'score'
        ylab <- 'Scale Score'
    } else if (type == 'n_missing') {
        colname <- 'n_missing'
        ylab <- 'Number of missing responses'
    } else if (type == 'p_missing') {
        aDF$p_missing <- aDF$n_missing/(aDF$n_items + aDF$n_missing)
        colname <- 'p_missing'
        ylab <- 'Proportion of missing responses'
    }

    p <- ggplot(aDF, aes_string(y = colname, x = 'scored_scale')) +
        geom_violin(fill = 'black', alpha = .25, color = 'gray') +
        geom_boxplot(alpha = .5, width = .25, color = 'black') +
        geom_point(position = position_jitter(w = .125, h = 0),
                   alpha = .25, color = 'blue') +
        labs(y = ylab, x = 'Scale name') +
        theme_classic()
    if(by_gender){
        p <- p + facet_grid(reformulate(gender_var, '.'))
    }
    return(p)
}


plot_tds_scale(tds2_wave1_scored,
               scale_regx = '^ACE$',
               type = 'score')
```

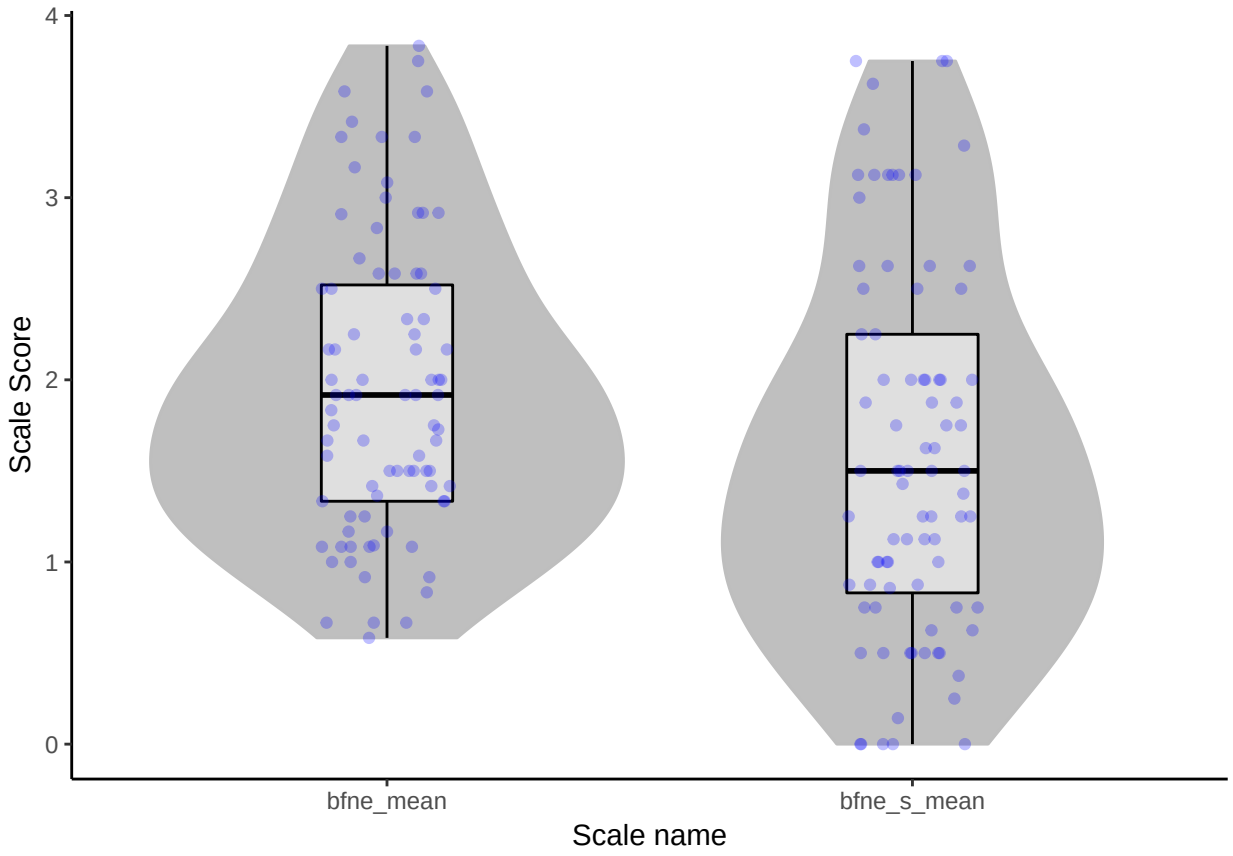



```
plot_tds_scale(tds2_wave1_scored,
               scale_regx = '^ACE$',
               type = 'p_missing')
```

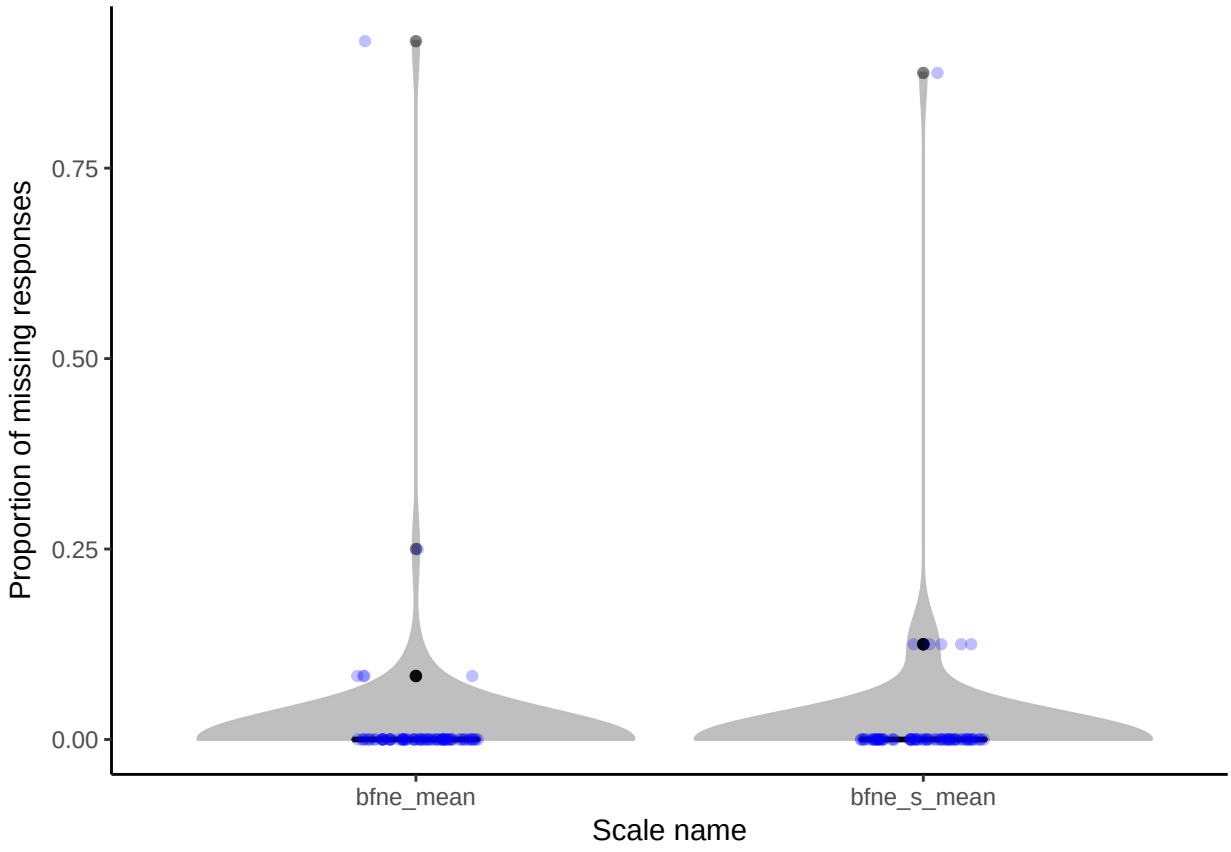



**BFNE**

```
plot_tds_scale(tds2_wave1_scored,
               scale_regx = '^BFNE$',
               type = 'score')
```




```
plot_tds_scale(tds2_wave1_scored,
               scale_regx = '^BFNE$',
               type = 'p_missing')
```



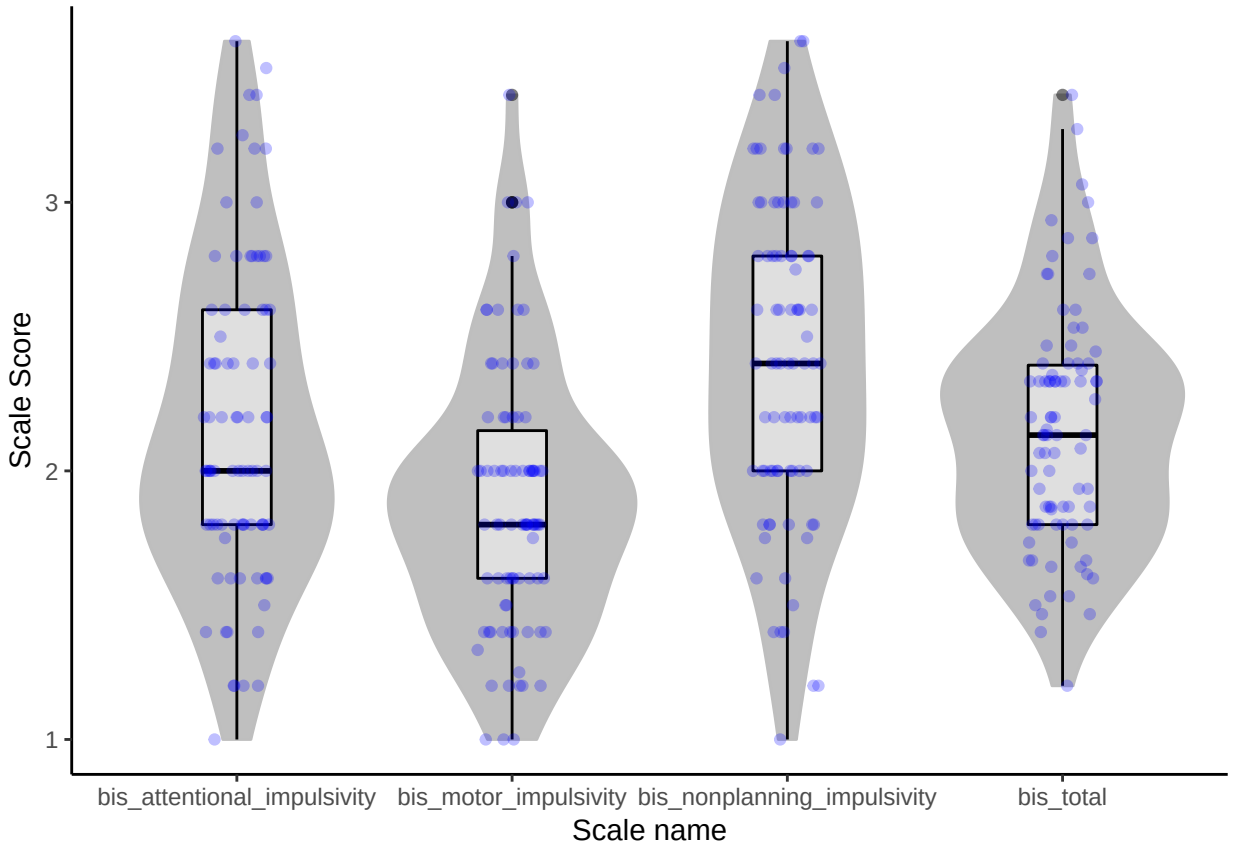

**BIS-15**

```
plot_tds_scale(tds2_wave1_scored,
               scale_regx = '^BIS-15$',
               type = 'score')
```

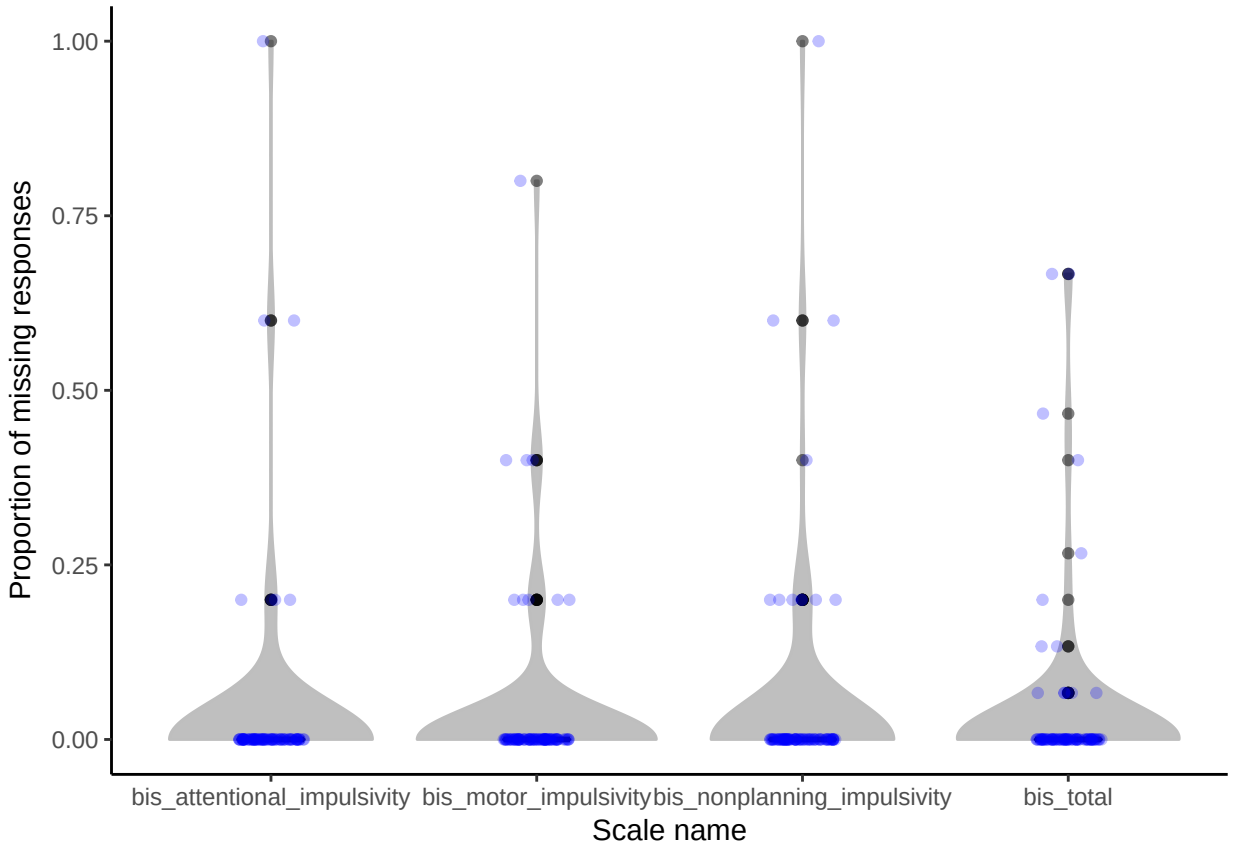
```
## Warning: Removed 2 rows containing non-finite values (stat_ydensity).
```

```
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```




```
plot_tds_scale(tds2_wave1_scored,
               scale_regx = '^BIS-15$',
               type = 'p_missing')
```


1.00
0.75
0.50
0.25
0.00
Proportion of missing responses
bis_attentional_impulsivity
bis_motor_impulsivity
bis_nonplanning_impulsivity
bis_total
Scale name