An R Markdown program to analyse responses to a Discrete Choice Experiment (DCE) survey exploring the online help seeking preferences of socially anxious young people

Part 1: Pre-process Data

Matthew P Hamilton[1,*]

19 October 2022

[1] Orygen, Parkville, Australia

[*] Correspondence: Matthew P Hamilton <matthew.hamilton@orygen.org.au>

Suggested citation: "Matthew Hamilton (2022). dce_sa_analysis: An R Markdown program to analyse responses to a Discrete Choice Experiment (DCE) survey exploring online help seeking in socially anxious young people."

1 About this code

1.1 Purpose

This program is used to ingest and preprocess survey response data for a Discrete Choice Experiment study that is currently being written up. Future versions of this program will include details of the parent study.

The raw versions of the data to be analysed exist in multiple files. These files need to be appropriately merged and reformatted before we can undertake descriptive and choice modelling analyses. In addition to creating a composite dataset from multiple source files, the program implements pre-processing steps to:

- restrict the dataset to valid responses;
- account for choice sets being organised into two blocks, only one (randomly selected) of which was presented to respondents;
- derive variables from participant responses (e.g. calculate a total SIAS score and SIAS based anxiety band from the SIAS assessment questions);
- format the dataset so that it is compatible with the R packages selected for analysing the data.

1.2 Status

This code is the same version as was used to analyse study data. The only items that have been modified are those that remove references to the local directory structure on which study data was stored.

1.3 Use

This code can be run either by "knitting" the parent RMD file or by manually executing each code chunk. The code as is currently optimised for being knit in one step, which means that in a number of instance the value "Y" has been supplied to the `consent_1L_chr` argument of program functions. Supplying this value overrides the default behaviour of functions that write files which is to prompt users for their active consent prior to to write files and directories to their machine. If running this code interactively, we recommend omitting the `consent_1L_chr` argument as this will provide you with greater transparency about what files and directories are being written to your machine.

2 Prepare workspace

2.1 Install and load required libraries

If you do not already have the required libraries to run this program installed, you can do so by un-commenting and running the following lines.

```
# devtools::install_github("ready4-dev/ready4")
# devtools::install_github("ready4-dev/mychoice") # add lwgeom to imports
```

Next we load the libraries required to run this program.

```
library(ready4)
library(mychoice)
```

2.2 Specify data directories

We begin by specifying where our input data can be located and where we wish to write our outputs to. You must supply these details or the rest of this code will not work.

```
paths_ls <- list(input_data_dir_1L_chr = "PROVIDE DETAILS HERE",
                 output_data_dir_1L_chr = "PROVIDE DETAILS HERE",
                 raw_data_fl_nms_chr = "PROVIDE DETAILS HERE")
```

2.3 Reproducibility

We now set a seed to aid reproducibility.

```
set.seed(1001)
```

Having set the seed, it is now likely that if you run the syntax described in this document on your own installation of R you will get identical results to those reported in this document. However, if you do not, it may be that you have a different version of R, or of some of the packages that we used to produce this analysis. We therefore save a record of the software that we have on the machine used for this analysis so this can be made available for comparisons.

```
session_ls <- sessionInfo()
```

3 Create custom functions

We now create a number of functions that we will use in subsequent parts of this program.

4 Data ingest and processing

4.1 Create preprocessing log

We create an object into which we will save key milestones from our pre-processing algorithm.

```
preprocessing_log_ls <- list(session_ls = session_ls)
```

4.2 Ingest and validate data

4.2.1 Ingest data on choice features

We next retrieve data on the choices that participants were presented with from the code we previously ran to create the choice cards.

```
dce_design_ls <- readRDS(paste0(paths_ls$Replication,"/dce_design_ls.RDS")) # Need to change to get data from online repo
```

4.2.2 Import raw responses

We have three databases of participant responses - one from individuals that were sent a survey invite because they had participated in the Entourage pilot study and two of respondents to publicly advertised invitations. The reason for having two databases of responses to public invitations is that during the initial phases of our survey it became apparent that an individual or individuals was/were trying to game the survey in response to the survey participation incentive. To address this we temporarily paused data collection before relaunching the survey with additional security features. One of these additional features was the addition of one free text question in the relaunched public invitation version, which was used solely for the purposes of helping to screen out potential non-genuine respondents. Other than this additional screening question, the surveys were identical for all three sets of respondents.

The following commands ingest and merge the three databases in their raw format.

```
records_ls <- bind_tables_from_loc_files(paste0(paths_ls$input_data_dir_1L_chr,
                                                      paste0("/",paths_ls$raw_data_fl_nms_chr)),
                                                force_numeric_1L_lgl = T, force_tb_1L_lgl = T, heading_rows_1L_int = 3L) %>%
  make_records_ls(choice_vars_pfx_1L_chr = "DCE_B")
```

```
preprocessing_log_ls <- list(raw_case_nbrs_1L_int = nrow(records_ls$ds_tb))
```

4.2.3 Drop missing choice responses

We next drop all records for which no choice card questions were answered.

```
records_ls$ds_tb <- remove_no_choice_responses(records_ls$ds_tb, choice_card_pfx_1L_chr = records_ls$choice_vars_pfx_1L_chr)
```

```
preprocessing_log_ls$raw_choice_resps_1L_int <- nrow(records_ls$ds_tb)
```

4.2.4 Restrict each dataset to valid responses

We next want to restrict response data to valid responses by dropping responses assessable as non-genuine. To do this we combine qualitative and automated checks.

Qualitative assessments were undertaken by one study investigator who reviewed participant free text responses, creating a dataset (`qltv_ax_tb`) in which flags ("green" or "red") were added to any individual responses that the reviewer had high degrees of confidence were likely to be genuine or not-genuine. Examples of where responses were red-flagged were for free-text entries that were either nonsensical or which poorly addressed the survey question with highly generic responses that may have been cut-and-pasted from published resources on social anxiety.

Automated checks added red flags if:

- a participant's stated age was inconsistent with the age calculated by combining their stated date of birth and the date of their response;
- the survey was completed in under 5 minutes;
- the participants submitted an email address which included 5 or more consecutive digits;
- different individuals supplied identical free text respondents; and
- the geo-code associated with a respondents' IP address was from outside Australia.

```
records_ls <- add_flags(records_ls,
                        qltv_ax_tb = {if(file.exists(paste0(paths_ls$input_data_dir_1L_chr,"/qltv_ax_tb.RDS"))){
                          qltv_ax_tb <- readRDS(paste0(paths_ls$input_data_dir_1L_chr,"/qltv_ax_tb.RDS"))}else{
                            qltv_ax_tb <- NULL}},
                       # Variables for age inconsistency red flag
                       age_var_nm_1L_chr = "Ent_DCE_Age",
                       date_stamp_format_1L_chr = "DMY",
                       date_stamp_var_nm_1L_chr = "StartDate",
                       dob_format_1L_chr = "DMY",
                       dob_var_nm_1L_chr = "Ent_DCE_DOB",
                       # Variables for too short completion time red flag
                       attempt_dur_min_1L_dbl = 300, #
```

```r
                       attempt_dur_var_nm_1L_chr = "Duration (in seconds)",
                       # Variables for unusual email address red flag
                       email_max_cnstv_digits_1L_int = 5,
                       email_var_nm_1L_chr = "Ent_DCE_TY_1",
                       # Variables for ineligbile location red flag
                       lat_lng_var_nms_chr = c("LocationLongitude", "LocationLatitude"),
                       valid_countries_chr = "Australia",
                       # Variables for duplicate responses red flag
                       unique_by_case_var_nms_chr = "Ent_DCE_SAsupport",
                       # Variables for qualitative assessment flags
                       flags_max_1L_int = 0,
                       qltv_ax_green_flag_1L_chr = "Green",
                       qltv_ax_red_flag_1L_chr = "Red",
                       qltv_ax_var_nm_1L_chr = "Colour_code")
```

```r
records_ls$ds_tb <- rlang::exec(add_red_flags, records_ls$ds_tb,
                                   !!!(records_ls$flags_ls %>% purrr::list_modify("flags_max_1L_int" = NULL,
                                                                                  "qltv_ax_green_flag_1L_chr" = NULL)))
```

```r
preprocessing_log_ls <- make_flags_smry_ls(records_ls$ds_tb,
                                           qltv_ax_green_flag_1L_chr = records_ls$flags_ls$qltv_ax_green_flag_1L_chr,
                                           qltv_ax_var_nm_1L_chr = records_ls$flags_ls$qltv_ax_var_nm_1L_chr,
                                           preprocessing_log_ls = preprocessing_log_ls,
                                           flags_max_1L_int = records_ls$flags_ls$flags_max_1L_int)
```

We then dropped responses that had more than one red flag, unless it had been green-flagged in the qualitative assessment.

```r
records_ls$ds_tb <- records_ls$ds_tb %>%
      remove_red_flag_cases(flags_max_1L_int = records_ls$flags_ls$flags_max_1L_int,
                            qltv_ax_green_flag_1L_chr = records_ls$flags_ls$qltv_ax_green_flag_1L_chr,
                            qltv_ax_var_nm_1L_chr = records_ls$flags_ls$qltv_ax_var_nm_1L_chr)
```

```r
preprocessing_log_ls$raw_unflagged_1L_int <- nrow(records_ls$ds_tb)
```

4.3 Pre-process merged dataset of valid responses

The merged dataset of valid responses we created in the last step is not yet ready for analysis. To prepare the dataset for the analyses that we want to run we have to add information about the design of the survey, perform some transformations on the individual characteristics of survey respondents and reformat our data.

4.3.1 Prepare respondent characteristic data

We first need to create a number of new, derived or recoded or variables relating to respondent characteristics. We therefore appy a transformation function that performs the following tasks:

- creates a dichotomous age variable (under 20 years of age);
- creates dummy gender variables (Male and Other, Prefer Not To Say) with Female as the base gender category;
- calculate total SIAS scores and create Normal Range (under 34 total SIAS score) and Social Anxiety (SIAS scores above 43) dummy SIAS variables with Social Phobia (34-42 total SIAS score) the base category;
- create a SEIFA disadvantage quartile variable (based on participant postcode) and dummy variables for SEIFA disadvantage quartiles 1,2 and 4 (base category, SEIFA disadvantage quartile 3);

- create a State and Territory variable (based on participant postcode) and dummy variables for Victoria, Queensland, South Australia, Western Australia, ACT and Tasmania (base category New South Wales); and
- drop unnecessary variables.

```r
preprocessing_log_ls$transformation_fn_ls <- list(fn = transform_repln_ds_for_analysis,
                                                  args_ls = list(consent_1L_chr = "Y",
                                                                 write_to_1L_chr = paths_ls$Replication))
```

```r
records_ls$ds_tb <- rlang::exec(preprocessing_log_ls$transformation_fn_ls$fn,
                                records_ls$ds_tb,
                                !!!preprocessing_log_ls$transformation_fn_ls$args_ls)
```

4.3.2 Create database of respondent choices

The next step is to extract participant's responses to the choice cards as a matrix.

The design matrix we loaded in a previous step is not yet in the correct order. That is because we randomly sampled from that design matrix to create the choice cards in each block. We therefore need to reorder our design matrix to match how choices were presented to participants in the survey. Our survey included an opt out choice option on each choice card. We need to identify that variable prior to analysi by adding an alternative specific constant. Our database of valid responses is currently in a wide format - each case contains data on all choices presented in each choice card. For the analyses we plan on running, our choice response matrix needs to be in long format - each choice in a choice set needs to be its own case. We also need to selectively apply transformations to attribute variables depending on whether they are continuous or factors. In our survey, only the cost attribute is a continuous variable. All the factor attributes require the creation of dummy variables.

```
records_ls$case_choices_mat <- make_case_choices_mat(records_ls$ds_tb, block_indcs_ls = dce_design_ls$block_indcs_ls,
                                                     choice_sets_ls = dce_design_ls$choice_sets_ls,
                                                     design_mat = dce_design_ls$design_mat,
                                                     choice_vars_pfx_1L_chr = records_ls$choice_vars_pfx_1L_chr)
```

```
preprocessing_log_ls$cases_wide_1L_int <- nrow(records_ls$case_choices_mat)
```

4.3.3 Specify modelling parameters

We now specify detail about the variables that we plan to use in our choice modelling.

```
mdl_params_ls <- tibble::tibble(domain_chr = c(rep("demographic",5),
                                               rep("clinical",4),
                                               rep("spatial",13),
                                               rep("study",5)),
                                concept_chr = c(rep("age",2),
                                                rep("gender",3),
                                                rep("SIAS",4),
                                                rep("area disadvantage",4),
                                                rep("urbanicity",2),
                                                rep("jurisdiction",7),
                                                "pilot participant",
                                                rep("incomplete choice tasks",2),
                                                rep("time taken",2)),
                                type_chr = c("continuous", "logical",
                                             "factor", "dummy", "dummy",
                                             "continuous", "factor", "dummy", "dummy",
                                             "factor",
                                             "dummy", "dummy","dummy",
                                             "factor", "logical",
                                             "factor", "dummy", "dummy","dummy", "dummy",
                                             "dummy", "dummy",
                                            "logical",
                                            "continuous","logical",
                                            "continuous", "logical"),
                               short_name_chr = c("rc_age", "der_age_u20",
                                                  "der_gender", "der_gender_male", "der_gender_opnts",
                                                  "der_SIAS_ttl", "der_SIAS_ctg", "der_SIAS_nr", "der_SIAS_sa",
                                                  "der_SEIFA_Quartile",
                                                  "der_SEIFA_Q1", "der_SEIFA_Q2", "der_SEIFA_Q4",
                                                  "der_SOS", "der_urban",
                                                  "der_STE", "der_STE_VIC","der_STE_QLD","der_STE_SA","der_STE_WA",
                                                  "der_STE_ACT","der_STE_TAS",
                                                  "rc_pilot_participant",
                                                  "der_Missing_Tasks","der_All_Tasks",
                                                  "rc_time_taken_secs", "der_under_ten_mins"),
                               long_name_chr = c("age","aged 15-19",
                                                 "gender", "male gender", "other gender or prefer not to say",
                                                 "total SIAS score", "overal SIAS category", "SIAS normal range", "SIAS social anxiety",
                                                 "SEIFA disadvantage (by postcode, national comparisons) quartile",
                                                 "SEIFA Quartile 1", "SEIFA Quartile 2", "SEIFA Quartile 4",
                                                 "Section of State", "resides in an urban area",
                                                 "State or Territory", "Victoria", "Queensland","South Australia","Western Australia",
                                                 "Australian Capital Territory", "Tasmania",
                                                 "Entourage pilot programme participant",
                                                 "number of incomplete choice tasks","completed all choice tasks",
                                                 "time taken to complete survey in seconds", "completed survey in under ten minutes"
                                                 )) %>%
 dplyr::mutate(false_value_chr = dplyr::case_when(long_name_chr == "aged 15-19" ~ "aged 20-25",
                                                  long_name_chr == "resides in an urban area" ~ "resides in a rural or remote area",
                                                  long_name_chr == "Entourage pilot programme participant" ~ "responded to public adverti
                                                  long_name_chr == "completed all choice tasks" ~ "did not complete all choice tasks",
                                                  long_name_chr == "completed survey in under ten minutes" ~ "took at least ten minutes t
                                                  T ~ NA_character_)) %>%
 dplyr::mutate(units_chr = dplyr::case_when(short_name_chr == "rc_time_taken_secs" ~ "seconds",
                                            T ~ NA_character_)) %>%
make_mdl_params_ls(as_selection_ls_1L_lgl = T,
                   concepts_chr = c("age","gender","SIAS",
                                    "urbanicity", "jurisdiction", "area disadvantage",
                                    "pilot participant", "incomplete choice tasks", "time taken" ),
                     types_chr = c("logical","dummy","dummy",
                                   "logical", "dummy", "dummy","logical",
                                   "logical",
                                   "logical"),
                     dce_design_ls = dce_design_ls,
                     records_ls = records_ls)
```

4.3.4 Create database of respondents

We next create a composite, indexed database that combines the respondents' characteristics that we will use in choice modelling, the choice sets presented to respondents and their choices.

```
records_ls$ds_dfidx <- make_choice_mdlng_ds(case_choices_mat = records_ls$case_choices_mat,
                                            candidate_predrs_tb = mdl_params_ls$candidate_predrs_tb,
                                            card_id_var_nm_1L_chr = records_ls$card_id_var_nm_1L_chr,
                                            choice_sets_ls = dce_design_ls$choice_sets_ls,
                                            ds_tb = records_ls$ds_tb,
                                            person_card_uid_var_nm_1L_chr = records_ls$person_card_uid_var_nm_1L_chr,
                                            person_uid_var_nm_1L_chr = records_ls$person_uid_var_nm_1L_chr,
                                            concepts_chr = character(0),
                                            types_chr = character(0),
                                            as_selection_ls_1L_lgl = F)
```

5 Write output files

We now save our output. If running this code interactively, we recommend omitting the `consent_1L_chr` argument.

```
paths_ls <- write_preprocessing_outp(paths_ls,
                                     mdl_params_ls = mdl_params_ls,
                                     preprocessing_log_ls = preprocessing_log_ls,
                                     records_ls = records_ls,
                                     consent_1L_chr = "Y")
```