Journal of
open research software

Allon, A S and Luria, R 2016 prepdat- An R Package for Preparing Experimental Data for Statistical Analysis. *Journal of Open Research Software,* 4: e43, DOI: http://dx.doi.org/10.5334/jors.134

SOFTWARE METAPAPER

prepdat- An R Package for Preparing Experimental Data for Statistical Analysis

Ayala S. Allon[1] and Roy Luria[1,2]

[1] School of Psychological Sciences, Tel-Aviv University, IL

[2] Sagol School of Neuroscience, Tel-Aviv University, IL

Corresponding author: Ayala S. Allon (ayalaall@post.tau.ac.il)

In many research fields the outcome of running an experiment is a raw data file for each subject, containing a table in which each row describes one trial conducted during the experiment. The next step is to merge all files into one big table, and then aggregate it into one finalized table in which each row corresponds (usually) to the averaged performance of each subject. prepdat- An R package- enables to easily perform these steps, including several possibilities for dependent measures and trimming procedures. prepdat helps researchers to optimize and speedup their analysis, and to better understand the results.

Keywords: R; Data manipulation; Data aggregation; Data analysis; Trimming procedures; Experimental designs

Funding statement: This work was supported by the Israel Science Foundation Grant number 1696–13 awarded to Roy Luria.

(1) Overview

Introduction

Preparing data for statistical analysis is a very common task in experimental research fields. For example in Experimental Psychology, often the outcome of running each subject in an experiment is a file (e.g., a text file) containing a raw data table in a long format with numerical description of the subject's performance in the various experimental conditions. The columns in this raw data table describe the independent variables, dependent variables, and various characteristics of the subject and the experiment (e.g., age, gender, and a numerical description of the stimulus in the experiment). The rows in this raw data table describe the observations (i.e., trials) conducted during the experiment, such that each row in the table corresponds to one observation. An example for such a raw data table of a single subject in an experiment can be found in Appendix A Table 1. Usually, this raw data table has over a hundred lines, and the number of raw data files corresponds to the number of subjects in a given experiment. Next, the researcher is interested in conducting both descriptive and inferential statistical analysis. However first, in order to run this statistical analysis, the raw data needs to be merged and aggregated.

Merging individual raw data tables requires vertically concatenating the tables into one big table containing raw data from all subjects (i.e., the merged table). Next, aggregating the merged table includes reducing the amounts of data to the desired level of information, resulting in a finalized table, usually in a wide format, in which each row in the table refers to a specific subject (which is the variable that identifies the unit upon which the measurement took place; i.e., the id variable), and each cell in the table usually reflects the averaged performance of that subject according to the desired grouping variables (i.e., the independent and dependent variables). This finalized table often contains only selected variables relative to the merged table. An example for such finalized table can be found in Appendix A Table 2.

These two steps pose a major problem for students and researchers who many times find themselves doing these procedures in Excel by pasting one raw data table after the other to merge the data and then use Pivot tables to aggregate it, which takes a lot of time and energy, and is prone to various mistakes. For example, a standard procedure in analyzing psychological data is to remove (i.e., trim) outliers before conducting descriptive and inferential statistical analysis, assuming these outliers do not reflect the investigated process in question. One procedure to trim outliers is by computing the arithmetic mean (i.e., mean) for each cell in the aggregated table after rejecting observations that fell outside the range of a predefined criterion, which is usually a number of standard deviations (*SDs*) from the mean of that cell (e.g., computing the mean of a cell after rejecting observations that were more than 2.5 *SDs* from the mean of that cell; a procedure also known as the restricted means procedure). This procedure requires a combination of a number of Excel functions that are implemented in a few steps because the user needs to iterate and compute the criterion for each

subject, reject observations that fell outside that criterion, and then aggregate the trimmed data. Note that each trimmed observation changes the mean, leaving a lot of room for ad-hoc decisions and changes by the user about deciding when to stop trimming and is prone to mistakes if the user does not use the same criteria for all subjects.

R [1], which is a free language and computing environment for statistical analysis and graphics, has some functions for aggregating data such as the tapply() function form the base package [1], dcast() from the reshape2 package [2] and group_by() and summarise() from the dplyr package [3]. In principle, the user can aggregate the data using these kinds of functions. However, this will require detailed and serious thinking about how to put together a code that will provide the desired result. In addition, these types of functions can handle only a few scenarios, and each function requires the input arguments in a somewhat different manner. Moreover, even after tailoring the code to produce the desired result for the analysis in hand, it will require changes in order to produce the desired results in future analysis. In addition, to the best of our knowledge, until now there was no particular function in R that enabled to import individual raw data tables from an external folder or folders and vertically concatenate them into one big table. The user had to write its own function, or to use external tools to R such as Excel Automation [4], EXMERG [5], or PEBL Data File Combiner [6]. While these kind of tools are worthy to use, they do not enable to conduct the whole analysis from within the R platform and create a unified chain of analysis. Avoiding all these shortcomings, prepdat [7] – an R Package for Preparing Experimental Data for Statistical Analysis- provides a general framework to overcome these problems because it is specially designed to merge and aggregate data with any number of grouping variables, requiring minor changes from one analysis to the other, and provides various measures of the dependent variable while using identical criteria for all subjects.

The goal of the present paper is to introduce prepdat, an R package that enables the user to easily and quickly merge (using the file_merge() function) and aggregate raw data tables (using the prep() function) while keeping track and summarizing every step of the preparation. Except for means, which are very common dependent measures, prep() includes several other possibilities for the aggregated values of the dependent variable such as medians, percentiles, and means after rejecting observations above *SD* criterion. In addition, prep() includes special trimming procedures for measurements of reaction-times (RTs) [8]. In this paper we provide a manual for using prepdat. We first overview prepdat and then demonstrate how to implement prepdat using example data. prepdat is a free R package, which makes it available for use to any user from his or her own computer. We hope prepdat will help researchers to optimize and speedup their analysis, and help to better understand the results.

Implementation and architecture

The two functions the user needs for preparing the finalized table ready for further statistical analysis are the file_merge() and the prep() functions. Except for these functions, we also made the internal functions in prep available (such as the non_recursive_mc(), modified_recursive_mc(), and the hybrid_recursive_mc() functions), however the user does not need to call these functions by because prep() will call them if needed.

The file_merge() vertically concatenates files containing data tables in a long format into one big single table (i.e., the merged table). Then, the prep() function aggregates the merged table or any other table in a long format according to any number of grouping variables, which makes it suitable for various types of experimental designs such as between-subjects designs, within-subjects (i.e., repeated measures) designs, and mixed designs (i.e., designs that combine between-subjects and within-subjects independent variables). prep() is very easy to use, and only involves filling various arguments and when needed, making changes to default procedures for removing outliers. Moreover, because all calculations and procedures are done at once for all subjects, prep() ensures that the criteria for trimming procedures are identical for all subjects. Furthermore, prep() creates a summary file that helps to keep track of the finalized table it created, and this finalized table can be further analyzed in R or it can be exported to other statistical programs for further analysis. As already mentioned, prep() includes several possibilities for dependent measures including harmonic means and *SD*s for each cell in the finalized table. Please note that the *SD* for this procedure and for all trimming procedures in prep() is calculated using denominator N and not $N-1$, which is the denominator in the sd() function in R. Furthermore, prep() provides the 5th, 25th, 50th, 75th, and 95th percentile of the dependent variable, and the user can specify any other desirable percentile (e.g., the 66th percentile). Looking at different percentiles of the dependent variable can provide information about the source of the effect in question.

In addition, prep() enables to compute the dependent measure for each cell after rejecting outliers according to several trimming procedures. The user can specify a window of acceptable observations for computing the dependent measures according to any desirable logical condition/s. For example, when measuring RTs, the user can specify a window between 100 milliseconds (ms) and 2000 ms for valid observations, and prep() will do all further computations only on observations that were within this window. Additional trimming procedures include means when the analyzed window is determined by a range of number of *SD*s below and above the mean of each cell (e.g., ±2 *SD*). The default criteria for this procedure are 1 *SD*, 1.5 *SD*s, and 2 *SD*s. Namely, prep() provides the means according to each of these criteria. However, the user can specify any desirable number criteria of *SD*s (e.g., 2.5 *SD*s). Moreover, prep() provides for each criterion the number of observations rejected for each cell, number of observations for each cell before rejection, and proportions of rejected observations for

each cell. This additional information can help track subjects with high rejection rates.

Moreover, `prep()` includes three unique trimming procedures for RTs [8]. These procedures were created to overcome the effect of the number of observations for each experimental cell on outlier removal. As Miller [9] demonstrated, RTs distributions are characterized by a positive skew. When calculating means after rejecting outliers according to *SD* criterion, these means are sensitive to the amount of skewness in the distribution and to the number of observations on which they are calculated. Using Monte Carlo simulations, Van-Selst and Jolicoeur [8] developed new outlier eliminations procedures by applying recursive procedures and *SD* criteria that take into account the number of observations in each cell, which enabled them to decrease the effect of sample size on the means.

The first procedure is the Non-Recursive Procedure with Moving Criterion, which computes the mean for each cell after rejecting observations that were more than a criterion number of *SD*s from the overall mean of that cell. The notable aspect of this procedure is that the criterion for each cell is determined according to the number of observations in that cell. These criteria were originally determined by calculations done on theoretical distributions of RTs that were examined by Van-Selst and Jolicoeur [8] (for more information, see Table 4 in [8]). The second procedure is the Modified-Recursive Procedure with Moving Criterion, in which the mean and *SD* of each cell and for each round are computed based on observations of the cell after temporarily removing the highest observation. Then, for each round if the highest or lowest observations are more than a criterion number of *SD*s from the overall mean of the cell, they are removed from the sample. This procedure is continued until the stopping rule is met. The *SD* criterion is determined in the same way as in the Non-Recursive Procedure with Moving Criterion. Lastly, `prep()` can implement the Hybrid-Recursive Procedure with Moving Criterion (this procedure was suggested by Van-Selst and Jolicoeur [8]), which takes the average for each cell of the means produced in the Non-Recursive and Modified-Recursive procedures with moving criterion. For each of these three procedures `prep()` also provides the number of observations rejected for each cell, the number of observations for each cell before rejection, and the percent of rejected observations for each cell.

Installing `prepdat`

This section is intended for non-professionals `R` users, such that even users that are unfamiliar with the `R` software could use `prepdat`. To install `prepdat`, start `R` and then type in the console[1]:

```
> install.packages("prepdat")
```

This will install `prepdat` and its dependencies on your system. After installing `prepdat`, type the following in the console to load `prepdat` into `R` for use in the current session (note that each time the user starts a new `R` session `prepdat` needs to be loaded for that session):

```
> library("prepdat")
```

Example Data

The example data, which can be downloaded from the Example Data repository on GitHub, contains individual raw data files collected from 21 subjects in our lab. These subjects performed a visual search task similar to the one used in Lavie and Cox [11], in which they had to indicate on each trial which one of two possible letters was present in a circular array of letters (e.g., deciding whether an X or N appeared in the array). On some trials, a peripheral distractor letter (flanker) appeared outside the circular array and either matched the target for that trial (the Compatible condition), matched the other possible target (the Incompatible condition), or was not presented (the No-flanker condition). On half of the blocks, the flanker was presented at the same peripheral location marked by a placeholder throughout the whole task (Cue condition), and in the remaining blocks it was presented at random peripheral locations and was not marked by a placeholder (No-cue condition). The order of the Cue and No-cue blocks was counterbalanced between subjects. An example for part of a raw data file collected from one subject in the experiment can be found in Appendix A Table 1. An index for each of the columns in the raw data can be found in Appendix A Table 3.

Merging Raw Data Files Using `file_merge()`

The `file_merge()` merges the individuals raw data files in which each row corresponds to a single trial (i.e., observation; see Appendix A Table 1) into one big raw data table containing a 'chain' of raw data from all subjects, one after the other (i.e., the merged table). For the function to work, the raw data files should be in the same format that is either txt or csv format. This function accepts the following arguments:

```
file_merge(
            folder_path = NULL
            , has_header = TRUE
            , new_header = c()
            , raw_file_name = NULL
            , raw_file_extension = NULL
            , file_name = "dataset.txt"
            , save_table = TRUE
            , dir_save_table = folder_path
            , notification = TRUE
           )
```

In the `folder_path` argument, the user should enter the path of the folder in which the files to be merged will be searched. The search is recursive, which means files can be located in different subdirectories and the search will continue until relevant files are found. In the `has_header` argument, the user should enter using `logicals` whether the files to be merged have headers or not. The default for this argument is `TRUE`. The `new_header` argument is suitable to a situation in which the files to be merged do not have headers, or in case the user wants to replace the current header. Specifying the new header can be done using a string vector in this argument. Next are the `raw_file_name` and `raw_file_extension` arguments. In the `raw_file_name` the user should enter the shared name of the files to be searched and merged, not including the file extension. The file extension (i.e., `csv` or `txt`)

should be entered using the raw_file_extension argument. The next three arguments concern the merged table. In the file_name argument, the user should enter a string with the name of the file file_merge() creates that will contain the merged table (default for this argument is "dataset.txt"). The file_merge() function will save the merged table into a txt or csv file only in case the save_table argument is set to TRUE, which is also the default for this argument. We recommend saving the merged table into a file because this will save the user from merging the raw data files again if in case aggregation of the merged table is needed in different R sessions. In the dir_save_table argument the user can enter a path in which the merged table will be saved. In case no such path is provided, file_merge() will save the merged table in the path provided in folder_path. Lastly, in case the notification argument is set to TRUE (which is also the default for this argument), file_merge() will print messages to the console about the progress of the function. We recommend keeping the default value for this argument as well. To merge the individual raw data files of the example data (note that you might need to set a different folder_path):

```
> merged_t <- file_merge(
+           folder_path = getwd()
+           , has_header = TRUE
+           , new_header = c()
+           , raw_file_name = "sub"
+           , raw_file_extension = "txt"
+           , file_name = "merged_t.txt"
+           , save_table = TRUE
+           , dir_save_table = NULL
+           , notification = TRUE
+        )
Found 21 files
21 files were merged and saved into merged_t.txt
file_merge() finished!
```

After running this function, a new text file called merged_t.txt containing the merged table should appear in the appropriate folder. This file can also be downloaded from the Example Data repository on GitHub.

Aggregating the Merged Raw Data Table Using prep()

After merging raw data files using the file_merge() function, the user is ready to continue preparing the finalized table using prep(), which is the main function of prepdat. prep() expects that values in the merged table upon which it performs calculations will be numeric (except for names of the columns) and will give an error message if it finds other values. prep() contains the following arguments:

```
prep(
   dataset = NULL
   , file_name = NULL
   , file_path = NULL
   , id = NULL
   , within_vars = c()
   , between_vars = c()
   , dvc = NULL
   , dvd = NULL
   , keep_trials = NULL
   , drop_vars = c()
   , keep_trials_dvc = NULL
   , keep_trials_dvd = NULL
   , id_properties = c()
   , sd_criterion = c(1, 1.5, 2)
   , percentiles = c(0.05, 0.25, 0.75, 0.95)
   , outlier_removal = NULL
   , keep_trials_outlier = NULL
   , decimal_places = 4
   , notification = TRUE
   , dm = c()
   , save_results = TRUE
   , results_name = "results.txt"
   , results_path = NULL
   , save_summary = TRUE
)
```

The user needs to notify prep() which table to aggregate. This is done by the dataset or the file_name arguments, which one of them must be provided. The dataset argument specifies the name of the merged table created using file_merge() (i.e., dataset = merged_t in the example data). The file_name argument specifies the name of a txt or csv file containing the merged table after merging the files using file_merge() (i.e., file_name = "merged_t.txt" in the example data) or other external function to R. In case the file_name argument was used, then the file_path argument, which specifies a string with the path of the folder in which the file entered in file_name is located, must be provided.

The next steps are to let prep() know what are the grouping variables for aggregating the merged table, or in other words to specify the independent and dependent variables in the experiment. This is done using the id, within_vars, between_vars, dvc, and dvd arguments. The id argument specifies the name of the column in the merged table that contains the variable indicating the case identifier (i.e., subject number; id = "subject" in the example data). The values for id should be unique for each subject in the experiment, and this argument must be provided. The within_vars argument is a string vector specifying the name/s of column/s containing independent variables manipulated or observed within-ids (i.e., repeated measurements, also known as within-subjects). In the example data this will be:

```
within_vars = c("cue_nocue", "compatibility")
```

Note that the order of the columns names in the within_vars argument is highly important, because it determines the order of the columns for each dependent measure in the finalized table. Namely, prep() aggregates the columns for the dependent measures by first dividing them to the levels of the first independent variable in within_vars, and then within each level prep() divides the columns according to the next variable in within_vars and so forth. Therefore, the order of the independent variables in within_vars should be according to the hierarchical order the user wishes. The between_vars argument is a string vector specifying the name/s of column/s containing independent variables manipulated or observed between-ids (i.e., between-subjects), and the order of the columns names for this argument does not matter. In the example data this will be:

```
between_vars = c("order")
```

Since the within_vars and the between_vars denote the independent variables in the experiment, which are the grouping variables for the aggregation, at least one of these arguments must be provided. Next, the dvc and the dvd arguments denote the names of the dependent variables in the merged table, and should be entered as strings. The dvc argument stands for continuous dependent variable, and therefore is the name of the column in the merged table that contains a dependent variable in an interval or ratio scale (e.g., RTs; dvc = "rt" in the example data). The dvd argument stands for discrete dependent variable, and therefore is the name of the column in the merged table that contains a dependent variable with discrete values (e.g., 0 and 1 when measuring accuracy; dvd = "ac" in the example data). Moreover, it is very important to make sure to enter the name of the continuous dependent variable in the dvc argument and the discrete dependent variable in the dvd argument and not the other way around in order for prep() to work properly. Please note that at least one of these arguments must be provided.

The aforementioned arguments are the ones the user must provide in order for prep() to work properly, while the following arguments are the ones that enable the user to produce the dependent measure according to desirable specific needs. The keep_trials and the drop_vars arguments allow deleting unnecessary observations (i.e., rows) and variables (i.e., columns) in the merged table, before the aggregation takes place. In the keep_trials argument the user should specify with logical conditions provided as one string, which observations are desirable to keep for further calculations. For example, if the merged table contains practice trials for each subject, the user should remove these observations by specifying how these observations were coded in the merged table. In the example data the practice trials are the ones for which the "block" column equals to zero and therefore the keep_trials argument will be:

```
keep_trials = "raw_data$block > 0"
```

raw_data is the internal object in prep() representing the merged table. All logical conditions should be put in the same string and be concatenated by & or |. For example, if the user wants to keep observations except for the ones in block 0 and block 3 this argument will be as follow:

```
keep_trials = "raw_data$block > 0 & raw_
data$block < 3"
```

In addition, note that the logical conditions for this argument can relate to different columns in the merged table. For example if the user for a specific analysis is interested in keeping observations except for the ones in block 0 and observations in each block after the 10th trial, this argument will be as follows:

```
keep_trials = "raw_data$block > 0 & raw_
data$trial_num > 10"
```

In the drop_vars argument the user should specify the names of the variables in the merged table that need to be deleted. Names should be entered as strings separated by a comma. In the example data, for instance, this can be

```
drop_vars = c("fix1_duration", "distance")
```

Note that all further arguments of prep() will relate to the remaining rows and columns in the merged table. Therefore, the user should carefully choose which (if any) rows and columns need to be removed from further calculations using the keep_trials and drop_vars arguments. We recommend using these arguments in order to delete observations and variables that have no significance for further analysis (such as practice trials, as mentioned above).

The next arguments are the keep_trials_dvc and the keep_trials_dvd that enable the user to keep specific observations for calculations of the dependent measures specified in dvc and dvd (respectively; except for means according to the outlier removal procedures; to remove observations for these procedures see the keep_trials_outlier argument below). The user can specify a window of acceptable observations for computing the dependent measures according to any logical conditions. These arguments should be specified as logical conditions provided as strings as in the keep_trials argument. In the example data the dependent variable for dvc is RT and the dependent variable for dvd is accuracy. A reasonable window of observations for these dependent measures in these kinds of experiments is to keep observations in which RT was above 100 ms and below 2000 ms for both RT and accuracy, and in addition for RTs, to keep only observations in which the subject's response was correct. Therefore, the keep_trials_dvc argument which relates to the continuous dependent variable (i.e., the variable entered in dvc) will be:

```
keep_trials_dvc = "raw_data$rt > 100 & raw_
data$rt < 2000 & raw_data$ac == 1"
```

and the keep_trials_dvd argument which relates to the discrete dependent variable (i.e., the variable entered in dvd) will be:

```
keep_trials_dvd = "raw_data$rt > 100 & raw_
data$rt < 2000"
```

The next argument is id_properties. This argument is suitable for a situation in which the user logged for each trial and for each subject in the experiment also other important details such as age and gender as in the example data. In this case, the user can specify the name of these variables as strings in the id_properties vector. For instance in the example data this will be:

```
id_properties = c("age", "gender")
```

In return, the values for these variables for each subject will appear in the finalized table.

The sd_criterion and the percentiles arguments allow the user to get the dependent measure specified in dvc according to different cross sections. In the sd_criterion argument the user can specify a number of *SD*s for which prep() will calculate the mean for each cell of dvc after rejecting observations that were below and above the *SD* from the mean of that cell. As already mentioned, default criteria for this procedure are sd_criterion = c(1, 1.5, 2), and the user can specify any desirable criterion number of *SD*s (e.g., 2.5 *SD*s). The following argument is percentiles, which will give the percentile of dvc according

to any percentile in percentiles. The default for this argument is:

```
percentiles = c(0.05, 0.25, 0.75, 0.95)
```

Note that the median, which is the 50th percentile, will be calculated by default even if the user does not enter it to percentiles.

The outlier_removal argument is relevant when the dependent measure in dvc is reaction-time (i.e., RT). This argument enables the user to get the mean of the dependent variable in dvc after rejecting outliers according to one of the three outlier removal procedures with moving criteria [8]. By assigning 1, 2, or 3 to the outlier_removal argument, prep() will execute the Non-Recursive, the Modified-Recursive, or the Hybrid-Recursive procedures, respectively. By default, prep() will not execute any of these procedures. In addition, perp() can perform only one of these procedures at a time. In the example data, we decided to perform the Modified-Recursive procedure, and therefore this argument will be:

```
outlier_removal = 2
```

The keep_trials_outlier enables to keep specific observations for performing one of the outlier removal procedures. In the example data for instance, since dvc is RT, only correct observations (i.e., observations for which the "ac" column equals to 1) are analyzed and therefore this argument will be:

```
keep_trials_outlier = "raw_data$ac == 1"
```

This argument should be specified as logical conditions provided as a string as in the keep_trials argument.

The remaining arguments are decimal_places, notification, dm, save_results, results_name, and save_summary, and are aimed to adjust the finalized table prep() returns. decimal_places, as its name suggests, allows determining the number of decimal places for the dependent measures of dvc, with the default being four places. We recommend using decimal_places = 0 whenever measuring RTs in ms. For the dependent measures of dvd, prep() gives three decimal places. The notification argument, as in the file_merge() function, prints messages about the progress of prep(). The dm argument, which is a shortcut for dependent measures, is a string vector that allows the user to specify the desirable dependent measures in case the user does not want all the dependent measures perp() outputs by default. For instance, if the user wants only the mean and median of dvc and the mean and error rate of dvd to be in the finalized tale, the dm argument will be as follows:

```
dm = c("mdvc", "meddvc", "mdvd", "merr")
```

mdcv stands for mean dvc, meddvc stands for median dvc, and mdvd and merr stand for mean dvd and mean error, respectively. The complete list of the names of the dependent measures prep() outputs can be found in Appendix A Table 5. All dependent measures except for the outlier removal procedures can be specified in dm.

prep() will save the finalized table into a txt or csv file only in case the save_results argument is set to TRUE (i.e., save_results = TRUE, which is also the default for this argument). In case save_results is TRUE, the user can specify the name of the file with the finalized table as a string (including txt or csv extension) in results_name (the default for this argument is results_name = "results.txt"). In addition, in case the user wants the file with the finalized table to be saved in a different folder than the one specified in file_path, then the path to this folder should be entered as a string in results_path. Finally, if the save_summary is set to TRUE, prep() will also create a summary txt or csv file (depending on the file extension of results_name) that helps to keep track of the type and the number of observations that were deleted from the merged table (see below for more details). The complete function of prep() in order to aggregate the example data is as follows (the complete txt file, results_n21.txt, with the finalized table of the example data can be downloaded form the Example Data repository on GitHub):

```
> results_n21 <- prep(
+ dataset = merged_t
+ , file_name = NULL
+ , file_path = NULL
+ , id = "subject"
+ , within_vars = c("cue_nocue",
    "compatibility")
+ , between_vars = c("order")
+ , dvc = "rt"
+ , dvd = "ac"
+ , keep_trials = "raw_data$block > 0"
+ , drop_vars = c()
+ , keep_trials_dvc = "raw_data$rt > 100 & raw_
    data$rt < 2000 & raw_data$ac == 1"
+ , keep_trials_dvd = "raw_data$rt > 100 & raw_
    data$rt < 2000"
+ , id_properties = c("age", "gender")
+ , sd_criterion = c(1, 1.5, 2)
+ , percentiles = c(0.05, 0.25, 0.75, 0.95)
+ , outlier_removal = 2
+ , keep_trials_outlier = NULL
+ , decimal_places = 0
+ , notification = FALSE
+ , dm = c()
+ , save_results = TRUE
+ , results_name = "results_n21.txt"
+ , results_path = getwd()
+ , save_summary = TRUE
+ )
results_n21.txt has 21 observations and 148
variables
prep() returned a data frame to console
Hip Hip Hooray! prep() finished
Have a great day and may all your results be
significant!
```

Overview of the Finalized Table prep() Returns

In this section we will go over the finalized table prep() returns. The first six rows of the finalized table for the example data can be found in Appendix A Table 4. The first column in the finalized table will always be the id column ("subject" in the example data), and the next columns will be the ones entered in the id_properties argument ("age" and "gender" in the example data).

The next columns of the finalized table contain the dependent measures according to the order specified in Appendix A Table 5. The number of columns for each dependent measure in the finalized table is according to the experimental design. In the example data there was one

between-subjects variable (Order: Cue blocks first, No-cue blocks first), and therefore the next column in the finalized table will contain a column named "order" specifying the level of that variable for each subject. Next, there were two within-subjects independent variables (Cue, with two levels: Cue, No-cue; Compatibility, with three levels: Compatible, Incompatible, No-flanker), and the hierarchical order in which these variables were entered into the within_vars argument was first the Cue condition and then the Compatibility condition. Thus, for each dependent measure in the finalized table there will be six columns according to Cue × Compatibility. The first dependent measure is the mean for the dependent variable specified in dvc. In the example data this was RT and therefore the first six columns are mean RTs. For example, the "mdvc1" is the mean RT for Cue Compatible trials, the "mdvc2" is the mean RT for Cue Incompatible trials, and the "mdvc3" is the mean RT for Cue No-flanker trials. Columns "mdvc4" to "mdvc6" will be the mean RT for the Compatible, Incompatible, and No-flanker trials for the No-cue condition, respectively.

Overview of the Summary File prep() Creates

As already mentioned, if save_summary is set to TRUE, prep() will also create a summary txt or csv file that helps to keep track on the finalized table. The summary file for the Example Data can be found in Appendix A Figure 1 (this file, results_n21_summary.txt, can also be downloaded from GitHub). The file contains the name of the merged table and the date in which prep() was run. The following lines specify how many observations the merged table had in each step of prep(). The first line gives the number of observations and variables of the merged table before any observations or columns were removed (7980 observations and 13 variables in the example data). Then, you will find which logical conditions and columns were entered into the keep_trials and drop_vars arguments and how many observations and columns were left after removing observations according to these arguments (7560 observations and 13 variables in the example data). The next few lines specify which logical conditions were entered into the keep_trials_dvc, keep_trials_outlier, and keep_trials_dvd; how many observations and variables were left for each of the dependent variables entered in dvc, dvd, and how many observations and variables were left for the outlier removal procedure in case the outlier_removal() argument in prep() was used. The last part of the summary file describes which subjects were entered into the analysis (i.e., which ids where included in the finalized table), what were the independent between-subjects variables and their levels, what were the id properties (e.g., age and gender), and finally what were the within-subjects independent variables, their levels, and their hierarchical order (i.e., what were the grouping variables for the aggregation).

This summary file can be later used in order to remember how the finalized table is organized when running both descriptive and inferential statistical analysis. In addition, this file can come in handy when writing the methods section of the manuscript. Often, it takes a while from the time the researcher finished analyzing the data until starting to write the manuscript. The researcher can easily extract from the summary file the percent of observations removed due to window procedures for each of the dependent variables reported in the manuscript, and also other important details for the methods section.

Concluding Remarks

The package- prepdat- introduced in the current paper enables the user to easily and quickly organize raw data while keeping the same procedures for all cells in the aggregated data, and avoiding mistakes often caused when manually organizing raw data. In addition, the package offers a look at dependent variables across different sections and measures. This offers researchers an easy way to locate the source of the effects in question. Furthermore, prepdat offers unique recursive trimming procedures for reaction-times. Moreover, it enables the user to keep track of every step in the organization of the data, which can be useful later when writing a manuscript. We hope prepdat will help beginning and advanced researchers and R users to better organize and understand their results.

Quality control

All the functions of prepdat were tested to see they produce the desired results by comparing outputs form the package and other statistical programs. In addition, prep() was also tested using a test unit form the testthat [12] package. The full script for this test can be found at the tests folder on the prepdat repository on GitHub. The structure of the package was checked using devtools::check(document = FALSE) [10] that provides R CMD check for R packages. This check was performed both on Mac OS X and Windows operating systems. Most importantly, since the package is available on CRAN, it has also successfully passed the CRAN R CMD check. The results from this check can be found here.

(2) Availability

Operating system

The package can work with either Windows, Mac OS X or Linux.

Programming language

R version 3.0.3 or higher.

Additional system requirements

An Internet connection is required to install prepdat and download the individual raw data files from the Example Data repository on GitHub in case the user wants to run the example analysis as detailed in the manuscript. The individual raw data files come in a zip file and will take 258 KB of memory after extraction.

Dependencies

R version 3.0.3 or higher.

List of contributors
This package was created by Ayala S. Allon and Roy Luria.

Software location:
Archive The Comprehensive R Archive Network
Name: CRAN
Persistent identifier: https://cran.r-project.org/web/packages/prepdat/index.html
Licence: GPL-3
Version published: 1.0.8
Date published: 09/23/2016

Code repository
Name: GitHub
Persistent identifier: https://github.com/ayalaallon/prepdat
Licence: GPL-3.
Date published: 02/03/2016

Language
R

(3) Reuse potential
prepdat is relevant for any situation in which repetitive data is collected (both human and animal) and one wishes to aggregate it to get different measures of the dependent variable. By calculating different measures of the dependent variable, prepdat enables to examine the data from multiple perspectives, locating the source of the effect in question. In addition, it is possible to use the finalized aggregated table for further statistical analysis in R such as Analysis of Variance (ANOVA) using aov(), T-Tests using t.test() or linear regression using lm(). Furthermore, the finalized table prep() outputs can be exported for further analysis in other statistical softwares such as SPSS and STATISTICA. Many research students do most of their analysis in Excel, which is prone to mistakes and ad-hoc decisions. However, prepdat ensures that all calculations are done at once using the same criteria for all subjects, does not require previous knowledge in programming, and thus can serve as a first step in the data analysis streamline for research students.

All functions and code of prepdat can be found at the prepdat repository[2] on GitHub, enabling the user to further develop and inspect the code. Support mechanisms for this package are the package maintainer's email: ayalaallon@gmail.com. Bug reports should be done using http://github.com/ayalaallon/prepdat/issues.

Additional Files
The additional files for this article can be found as follows:

- **Additional File 1: Appendix A.** http://dx.doi.org/10.5334/jors.134.s1

Acknowledgements
Nachshon Meiran's SAS macro code inspired the writing of this package. We would also like to thank James A. Grange for allowing us to use parts of his trimr [13] code for programing the outlier removal procedures.

Competing Interests
The authors have no competing interests to declare.

Notes
[1] Advanced users can install the most current version of prepdat, sometimes even before its official release on CRAN. In order to do that, the user should first install devtools [10] (a package for developing packages in R and allows installation of packages directly hosted in GitHub repository) by typing in the console:

```
> install.packages("devtools")
```

Then, install prepdat directly from GitHub by typing in the console:

```
> devtools::install_github("ayalaallon/prepdat")
```

[2] https://github.com/ayalaallon/prepdat

References
1. **Team, R C** 2015 R: A language and environment for statistical computing [Internet]. Vienna, Austria: R Foundation for Statistical Computing; 2013. Available at: http://www.r-project.org.
2. **Wickham, H** 2007 Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12): 1–20. DOI: http://dx.doi.org/10.18637/jss.v021.i12
3. **Wickham, H** and **Francois, R** 2014 dplyr: A grammar of data manipulation. Retrieved from: http://CRAN.R-project.org/package=dplyr (R package version 0.2).
4. **Excel Automation** n.d. Merge all CSV or TXT files in a folder in one worksheet. Available at: http://www.rondebruin.nl/win/s3/win021.htm (Accessed 20 September 2016).
5. **SemiColonWeb** 2014 EXMERG merge data online – www.Exmerg.Com. Available at: http://www.exmerg.com (Accessed: 20 September 2016).
6. **Mueller, S T** and **Piper, B J** 2014 The psychology experiment building language (PEBL) and PEBL test battery. *Journal of neuroscience methods*, 222: 250–259. DOI: http://dx.doi.org/10.1016/j.jneumeth.2013.10.024
7. **Allon, A S** and **Luria, R** 2016 prepdat: Preparing Experimental Data for Statistical Analysis. Retrieved from: http://CRAN.R-project.org/package=prepdat (R package version 1.0.8).
8. **Van Selst, M** and **Jolicoeur, P** 1994 A solution to the effect of sample size on outlier elimination. *The quarterly journal of experimental psychology*, 47(3): 631–650. DOI: http://dx.doi.org/10.1080/14640749408401131
9. **Miller, J** 1991 Short report: Reaction time analysis with outlier exclusion: Bias varies with sample size. *The quarterly journal of experimental psychology*, 43(4): 907–912. DOI: http://dx.doi.org/10.1080/14640749108400962
10. **Wickham, H** and **Chang, W** 2015 devtools: Tools to Make Developing R Packages Easier. p. 185. Retrieved from: http://CRAN.R-project.org/package=devtools (R package version, 1(0)).
11. **Lavie, N** and **Cox, S** 1997 On the efficiency of visual selective attention: Efficient visual search leads to inefficient distractor rejection. *Psychological Science*,

8(5): 395–396. DOI: http://dx.doi.org/10.1111/j.1467-9280.1997.tb00432.x

12. **Wickham, H** 2011 testthat: Get Started with Testing. *The R Journal*, 3(1): 5–10.

13. **Grange, J** 2015 trimr: An Implementation of Common Response Time Trimming Methods. Retrieved from: https://cran.r-project.org/package=trimr (R package version 1.0.1).

How to cite this article: Allon, A S and Luria, R 2016 prepdat- An R Package for Preparing Experimental Data for Statistical Analysis. *Journal of Open Research Software,* 4: e43, DOI: http://dx.doi.org/10.5334/jors.134

Submitted: 09 June 2016 **Accepted:** 11 October 2016 **Published:** 25 November 2016

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS