Behavior Research Methods (2022) 54:2001–2024
https://doi.org/10.3758/s13428-021-01718-y

# The *lrd* package: An *R* package and Shiny application for processing lexical data

Nicholas P. Maxwell[1] · Mark J. Huff[1] · Erin M. Buchanan[2]

Accepted: 26 September 2021 / Published online: 30 November 2021

## Abstract
Recall testing is a common assessment to gauge memory retrieval. Responses from these tests can be analyzed in several ways; however, the output generated from a recall study typically requires manual coding that can be time intensive and error-prone before analyses can be conducted. To address this issue, this article introduces *lrd* (Lexical Response Data), a set of open-source tools for quickly and accurately processing lexical response data that can be used either from the *R* command line or through an *R Shiny* graphical user interface. First, we provide an overview of this package and include a step-by-step user guide for processing both cued- and free-recall responses. For validation of *lrd,* we used *lrd* to recode output from cued, free, and sentence-recall studies with large samples and examined whether the results replicated using *lrd*-scored data. We then assessed the inter-rater reliability and sensitivity and specificity of the scoring algorithm relative to human-coded data. Overall, *lrd* is highly reliable and shows excellent sensitivity and specificity, indicating that recall data processed using this package are remarkably consistent with data processed by a human coder.

**Keywords** Memory · Cued recall · Free recall · Lexical retrieval · Recall processing · Recall scoring

People are generally able to acquire new knowledge with relative ease. Much of our understanding of how individuals organize and store learned information comes from the use of recall tests (see Polyn et al., 2009 for review). Typical recall paradigms present participants with a set of to-be-remembered items, and participants are asked to recall them on a later test. Recall can either be assessed via free report, in which individuals report information from memory with few, if any, cues or constraints (free recall), or by the presentation of a cue that is used to direct their retrieval (cued recall). Recall tests are routine in memory research, including studies investigating the effectiveness of different memory strategies (e.g., deep vs. shallow encoding tasks; Craik & Lockhart, 1972), survival processing (e.g., assessing memory for contaminated objects; Gretz & Huff, 2019), and metacognition (e.g., accuracy between judgments of learning and recall; Koriat & Bjork, 2005). Furthermore, because studies often employ words as stimuli (i.e., cue-target pairs), a large body of research has been conducted to explore how the lexical properties of the cue and target can later recall (word frequency, Criss et al., 2011; e.g., concreteness, Paivio et al., 1988) or how the semantic relationships between pairs affect recall (Maxwell & Buchanan, 2020). Though the research questions differ, recall studies generally employ lexical information in some capacity, either as the stimuli that participants are required to study, the dependent variable of interest, or more commonly, through a combination of the two.

Cued-recall testing is a well-known paradigm and has been used extensively in psychological research. For example, cursory searches of Google Scholar for the keywords "cued recall" and "free recall" conducted in April of 2021 yielded approximately 18,000 and 48,000 publications, respectively, published since the year 2000. These results spanned multiple subfields of psychology including neuroscience, psycholinguistics, and cognitive aging. Additionally, the rise of the Internet, combined with more powerful computers, has made it easier for researchers to conduct recall testing by providing access to platforms for participant recruitment and computer-based testing. Furthermore, in addition to aiding data collection, the Internet has allowed information about lexical characteristics of stimuli (such as word length or frequency) to be

✉ Nicholas P. Maxwell
nicholas.maxwell@usm.edu

[1] The University of Southern Mississippi, School of Psychology, 118 College Dr, Hattiesburg, MS 39406, USA

[2] Harrisburg University of Science and Technology, Harrisburg, PA, USA

more efficiently collected and organized. As a result, the past two decades have provided researchers with access to a growing number of normed databases with which to construct lexical stimuli for use within recall studies (e.g., The English Lexicon Project, Balota et al., 2007; The Small World of Words, De Deyne et al., 2019; The Semantic Priming Project, Hutchison et al., 2013). Recently, online tools to aid researchers in selecting stimuli from the appropriate normed database have been made available (e.g., The Linguistic Annotated Bibliography, Buchanan et al., 2019) and computer applications such as the *lexOPS* package for *R* (Taylor et al., 2020) have been developed to automate the stimuli selection process entirely while controlling for several types of word properties. Though there has been a proliferation of datasets and tools used to aid researchers with stimuli creation, little attention has been given to developing tools that assist researchers with processing the large amounts of data that are generated from these studies.

Since studies investigating memory using both cued- and free-recall testing typically generate large amounts of lexical text data, processing the output is often time-consuming and tedious. Furthermore, the number of participants recruited to take part in these studies has drastically increased within the past decade, partially as a response to the replication crisis (Maxwell et al., 2015), which has resulted in a greater need for efficient and accurate methods for processing recall data. As such, the purpose of this paper is to introduce the *lrd* (lexical response data) package, which has been designed to provide researchers with a set of simple and freely accessible tools that can be used to speed up scoring of text responses from recall studies.

Output from cued- and free-recall tests is generally scored by matching participants' retrievals of study stimuli to a scoring key containing the correctly studied memory items. Though typed responses are unquestionably easier to process relative to handwritten responses, each item retrieved must be manually checked against the key to determine accuracy. For large datasets, manually scoring data is arduous, resulting in hours of checking participant responses against an answer key. While such tasks can generally be divided across research assistants, manual scoring is still a time-consuming endeavor depending on the amount of data requiring processing. Furthermore, multiple scorers can potentially introduce error in the coded responses, as inconsistencies across raters may arise if not properly controlled for (i.e., addressing participant misspellings, plural vs. singular nouns, alternate tenses, etc.) and scoring discrepancies are not resolved. Finally, if scoring is not conducted blind to experimental conditions, potential biases may influence the final scored output.

To reduce the overall time spent processing raw output and potential coder inaccuracies, an alternative method is to automate the data coding processes by employing a computer application that can automatically compare participant responses relative to a scoring key. However, simple text matching of responses does not account for participant errors in responses, such as misspellings or embedded coding provided by the survey software (e.g., extra spaces, tabs, newlines, etc.). These items still represent a correct memory; however, more sophisticated text processing is required. While a human scorer can easily correct any minor character additions, omissions, or misspellings to correctly score retrieved memory items, an automated one-to-one matching program may not score these items correctly unless a sufficient degree of flexibility is programmed into the scoring package. Instead, *lrd* takes a *fuzzy string-matching* approach in which response strings are counted as correct if they closely approximate the key rather than match it exactly (see Singla & Garg, 2012, for review).

By using fuzzy string-matching, the functions comprising the *lrd* package have been specifically designed to accurately score lexical text data while granting increased flexibility for minor errors that may be present in recall output, as determined by the user. The goal of this article is two-fold. First, we provide brief overviews of each function contained in the *lrd* package in the *R* environment and detail the corresponding *R Shiny* application by providing step-by-step guides on how to implement each of these tools to process several types of recall data. Second, we test the accuracy and reliability of the scoring algorithm by comparing output obtained from this package with human-coded data using four large data sets. Specifically, we test this package's reliability by using its core scoring functions to recode cued-recall data derived from two recently published cued-recall studies (Maxwell & Buchanan, 2020; Maxwell & Huff, 2021), a study employing a free-recall task (Huff et al., 2018), and a study using sentence-recall task (Geller et al., 2020). For each study, we then compare data processed using *lrd* to the findings in the original human-coded datasets and tested whether the original findings reported in these studies replicate using *lrd*. By including functions for multiple recall test types, *lrd* can be applied to a wide variety of memory studies. Additionally, this allowed us to assess whether scoring accuracy changes as a function of test type.

For the two cued-recall studies, participants studied lists of paired associates and judged either how related the words in each pair were (Maxwell & Buchanan, 2020) or how likely they would remember the second word if cued by the first at test using a judgment of learning (JOL) rating (Maxwell & Huff, 2021). Upon conclusion of the study/judgment tasks, participants completed a distractor task followed by a cued-recall task in which the first word in each pair was presented and participants were asked to respond with the item it was originally paired with (e.g., *mouse*- ?). Next, for the free-recall data derived from Huff et al. (2018), participants studied six word lists in which list items were either semantically related or unrelated. Following study of each list, participants then engaged in a free-recall task. Finally, for the sentence data taken from Geller et al. (2020), participants listened to a sentence and, following the conclusion of each audio presentation, were instructed to type as much of the sentence as they recalled hearing. The recall data reported in each of the above studies was initially scored by manually checking responses against a scoring key via human coders. We rescored this output using *lrd* to illustrate that output generated automatically from this package can replicate human-scored results across each recall paradigm with a high degree of precision.

## Overview of the *lrd* package

*lrd* is an open-source tool developed for the *R* environment and an interactive application that consists of several functions for scoring lexical recall data and assessing the output. This package's primary goal is to automate the scoring process by matching participant responses to a list of correct responses stored in a key. Critically, this package has been designed to accomplish this task while granting flexibility towards participant response errors (e.g., misspellings or incorrect tenses). We additionally provide functions for specific analyses tied to free-recall data including serial position curves, conditional response probabilities, and probability of first recall (Kahana, 1996; Wahlheim & Huff, 2015).

We begin by providing a set of general instructions for downloading and installing the *lrd* package within the *R* environment. Next, we provide examples of the scoring functions for three types of recall as well as a set of functions that can be used to compute recall proportions for each test type. Third, we provide a general guide on how to use the package within both the *R* environment and through the use of a graphical user interface (GUI) implemented in *Shiny* and *shinydashboard* (Chang et al., 2021; Chang & Ribeiro, 2018). Finally, we conclude by assessing the validity of this package by using the cued-recall, free-recall, and sentence-scoring functions to process sets of each data type that have been scored by human coders.

## Installation and setup

The latest version of *lrd* (including all applicable documentation and source code for each function) can be accessed via GitHub (https://github.com/npm27/lrd). While proficiency with *R* is not required to run this package, it is assumed that users will have some familiarity with the *R* environment and/or basic experience with object-oriented programming. Installation is relatively straightforward, but currently requires the use of the *devtools* package (Wickham et al., 2020) to download and install the files from GitHub. Typing the following command, `devtools::install_github('npm27/lrd')` will begin the installation process by downloading and installing the latest version of *lrd*. By providing this package via GitHub, researchers can contribute, fork (i.e., make a copy), and modify functions of this package as needed. Installation using *devtools* will always be supported, and updated installation instructions will be provided when applicable on the README for the package. To begin using *lrd*, be sure to first load the package by using `library(lrd)`. Each function has been documented with information about the arguments and outputs stored within that function. Use ?function name to view the documentation and examples provided within the *R* working environment (i.e., `?prop_correct_cued`). Several example datasets are also provided within the package to demonstrate the three main scoring functions.

### Cued-recall scoring functions example

In this section, we provide a general guide to using *lrd* to score cued-recall data. This example uses a set of simulated response data that was designed to mimic output that might be obtained from a cued-recall study. While the dataset is smaller than what is typically generated from psychological experiments, we note that this set of responses is sufficient for our purpose of illustrating how *lrd* scores participant output. We begin this section by detailing the creation of this dataset before providing a step-by-step walkthrough of the *lrd* package's cued-recall scoring functions. Code and data for all examples have been made available at https://osf.io/admyx/ and within the packages as vignettes.

## Materials and dataset creation

To simulate a set of cued-recall data, 40 words were randomly generated using *LexOPS* (Taylor et al., 2020) to serve as target items (i.e., the scoring key containing correct responses). To simplify the stimuli selection process, we followed the general example provided by Taylor et al. by controlling for word prevalence and concreteness when generating this set of items. First, only highly concrete words were included (concreteness $\geq 4$, Brysbaert et al., 2014). Pairs were then evenly split based on word prevalence (e.g., the proportion of individuals who are familiar with a word, Brysbaert et al., 2019). Thus, the final stimuli consisted of 20 concrete, high-prevalence words (i.e., prevalence $\geq 4$) and 20 concrete, low-prevalence words (i.e., prevalence $\leq 2$).

We next simulated six sets of participant responses to these items. These response simulations varied in their degree of accuracy to cover a broad spectrum of potential participant responses, including no response errors (Participant 1), minor misspellings (Participants 2, 3, and 4), and major response errors (e.g., blank responses, incorrect answers, misspellings of more than two letters; Participants 5 and 6). For Participant 1, all responses matched the key to simulate a situation in which a participant correctly recalls all items. Data for Participants 2 and 3 were manipulated to simulate situations in which participants make minor mistakes at recall that do not necessarily preclude their responses from being counted as correct (e.g., misspellings where it is evident what the intended word is). These were generated by removing, replacing, or doubling specific letters. As such, the letter *e* was removed from all responses for subject 2 (e.g., *hey* becomes *hy*). For Participant 3, all instances of the letter *t* were doubled (e.g., *edit* becomes *editt*). Next, for Participant 4, all instances of the letter *a* were replaced with an *e*. This procedure allowed us to simulate a range of common participant errors such as omitting a letter, typing the wrong letter, or double pressing a key by mistake. Finally, data for Participants 5 and 6 were manipulated to simulate situations in which participants make major mistakes on recall (e.g., responding with an incorrect word). To simulate this type of response error for Participant 5, five responses from the answer key were randomly changed to a different but conceptually similar word (e.g., *financial* becomes *money*). The simulated data for Participant 6 increased the number of incorrect responses and added three instances of missing data.

## Formatting and loading the dataset

To view this example in the package, use `vignette ("Cued_Recall", package = "lrd")` to load the script and output in one file. When processing data, *lrd* requires that the input data be arranged in long format, wherein each row is one trial of participant responses. The package includes a function to convert wide format data (i.e., one row per participant), and an example of the data conversion is shown in the free-recall section below. We can use the following code to load and view the first six rows of the data[1]:

```
> library (lrd)
> data ("cued_recall_manuscript")
> head (cued_recall_manuscript)
```

| | Sub.ID | Trial_num | Cue | Target | Answer |
|---|---|---|---|---|---|
| 1 | 1 | 1 | chlorination | ideological | ideological |
| 2 | 1 | 2 | bendy | financial | financial |
| 3 | 1 | 3 | topography | editing | editing |
| 4 | 1 | 4 | enquiry | buzzing | buzzing |
| 5 | 1 | 5 | draconian | statistic | statistic |
| 6 | 1 | 6 | speedball | stopwatch | stopwatch |

[1] When copying code, please note that the arguments in quotes change color (usually green), as not all quote symbols are recognized by *R*. Simply delete them and retype the quotes if they do not copy correctly. > symbols indicate code has been executed in R.

The following information is required to analyze cued-recall data with the corresponding column name in this example in parentheses: a unique identifier for each participant (`Sub.ID`), a participant response column (`Answer`), and a unique identifier for each tested item (e.g., such as a trial number, `Trial_num`). Additionally, this function requires an answer key containing the correct responses and a unique identifier for each key item; however, these columns can either be stored as part of the input data or may be stored in a separate dataframe. The cued-recall dataframe may contain additional columns (e.g., columns denoting experimental conditions) that can be used to group the output. These columns must be between-subject values to be included in the output, with a one-to-one match between participant identifier and column information. Because scoring is case sensitive, the response and answer key columns will need to be checked for case discrepancies. For simplicity, we suggest converting both the answer key and response columns to lowercase before scoring the data.

```
> cued_recall_manuscript$Target <- tolower
(cued_recall_manuscript$Target)
> cued_recall_manuscript$Answer <- tolower
(cued_recall_manuscript$Answer)
```

## Scoring cued-recall data

Scoring cued-recall data with *lrd* is a relatively straightforward process with the ability to tweak the analysis to different desired outputs. We will run the `prop_correct_cued()` and save the output as a new object (`cued_output`)

```
> cued_output <- prop_correct_cued (data =
cued_recall_manuscript,
+ responses = "Answer",
+ key = "Target",
+ key.trial = "Trial_num",
+ id = "Sub.ID",
+ id.trial = "Trial_num",
+ cutoff = 1,
+ flag = TRUE,
+ group.by = NULL)
>
```

The `data` argument should list the dataframe of the participant answers. The `responses` argument indicates the column to find the participant answer, while the `key` column indicates the expected answer for that trial. The columns are listed in quotes if they are in the same dataframe as the data argument; however, the answer key columns could be listed in another dataframe to match to the participant answers (i.e., `answer_ key$Target`). The `key.trial` and `id.trial` arguments indicate how to match the answer key trial numbers to the participant data trial numbers. The `id` argument indicates the participant unique identifier. The `cutoff` column indicates the Levenshtein distance used for scoring, wherein 0 indicates a perfect answer to key match, and non-zero numbers indicate the number of substitutions, deletions, or changes allowed to consider for matching (Levenshtein, 1966). Levenshtein distance provides a method of fuzzy string matching wherein distance values represent the number of character changes required to transform the first word into the second. For example, two identical words such as *cat* and *cat* would have distance of 0, while *cat* and *bat* would have a distance of 1, and *cat* and *dog* would have a distance of 3. Levenshtein distances are sensitive to changes in character order, which provides an advantage over simple character matching, as *bear* and *bare* would be computed as 100% matching but have a Levenshtein distance score of 3. Given our data simulation, we used a cutoff score of 1 in this example to demonstrate how *lrd* can account for simple misspellings. The `flag` argument can be set to `TRUE` to provide the *z*-score values for each participant for their final cued-recall score. Last, the `group.` argument is used to include grouping variables to calculate percent recall by group or condition, and more than one variable can be used by concatenating a vector of column names (i.e., `c("column1", "column2")`).

This function provides up to three pieces of output stored in list format. First, this function provides trial-level data showing participant responses to each test item, the corresponding answer key item, and whether the program scored the response as correct (denoted as 1 for correct, 0 for incorrect):

```
> cued_output$DF_Scored
     Trial.   Sub.        Cue        Target     Responses     Answer     Scored
      ID       ID
1      1       1   chlorination  ideological  ideological  ideological     1
2      1       3   chlorination  ideological  ideological  ideological     1
3      1       5   chlorination  ideological  ideological  ideological     1
4      1       2   chlorination  ideological   idological  ideological     1
5      1       4   chlorination  ideological  ideologicel  ideological     1
6      1       6   chlorination  ideological               ideological     0
```

The second output consists of participant-level data, summarizing the proportion correct for each participant and the optional *z*-score for outlier detection:

```
> cued_output$DF_Participant
        Sub.ID      Proportion.Correct      Z.Score.Participant
1         1               1.00                   1.0259784
2         2               0.80                   0.0000000
3         3               0.85                   0.2564946
4         4               0.95                   0.7694838
5         5               0.75                  -0.2564946
6         6               0.45                  -1.7954621
```

The last output includes statistics of the grouping conditions if they were included in the scoring function. An example of this output is included below.

## Free-recall scoring functions example

The next section provides a general guide for using *lrd* to score free-recall data. For this example, we simulated a set of free-recall responses. The sample data were modeled after output obtained by Gretz and Huff (2019) in which participants watched videos of either healthy or sick individuals interacting with a variety of household objects and were tested via free-recall. First, we by detail the creation of this dataset. We then provide a detailed walkthrough of the *lrd* package's free-recall scoring function.

## Materials and dataset creation

To simulate a set of free-recall data, a list of 22 common household objects was first generated. This list was based on the "bedroom" video used by Gretz and Huff (2019), which can be viewed at https://osf.io/qbrgm/. Next, to simulate a set of responses, data were generated for six participants. To capture response variability, we varied the number of responses each participant provided, spelling errors of correct items, and inclusion of incorrect items. The full sample dataset and answer key files as well as all code used in the following examples have been made available at https://osf.io/admyx/. You can use `vignette("Free_Recall", package = "lrd")` to view this example within the package.

## Formatting and loading the dataset

In this example, we will load a free-recall dataset using `data(wide_data)` and the answer key for the free-recall lists with `data(answer_key_free)`. The dataset is structured in wide format, such that each participant is the one row in the data frame:such that each participant

```
> head(wide_data)
Sub.ID  Response                                        Disease.Condition
1       basket, chair, clothes, flowrs, glasses, fan,   healthy
          windows, …
2       windows, bed, books, shelf, pictures            healthy
3       bacpack, chair, glasses, mirror, iphone,        healthy
          pillow, …
4       vase, blinds, computer, magazine, books,        sick
          bed, blanket, …
5       bed, blankets, closet, windows, books, fan      sick
6       bed, blankets, dreser, nightstand, end table,   sick
          stereo, …
```

For all scoring functions, the data should be converted to long format, which includes one trial per row to properly match the answer key to the answer for each trial. In the example above, the `Response` column includes all the answers a participant listed using a comma-separated format. If the data are structured so that each concept is in a separate column, the data can be restructured into long format several ways: *reshape* (Wickham, 2007) or *data.table* (Dowle & Srinivasan, 2020) using the `melt()` function or *tidyverse* (Wickham et al., 2019) using the `pivot_longer()` function. In *lrd*, the `arrange_data()` function was added to assist in reformatting participants answers that were entered as one text string. To convert the `wide_data` into a useable long format, use:

```
> DF_long <- arrange_data(data =
wide_data,
+ responses = "Response",
+ sep = ",",
+ id = "Sub.ID")
```

The `data` argument indicates the dataframe containing the variables defined in the next arguments. The column name of the responses is denoted in quotes for the `responses` argument, and the `sep` argument indicates the separator between responses (i.e., a comma, semicolon, space, tab, or other text delimitator). The `id` variable is a column name in the dataframe for the unique participant identifier. In addition, this function contains an option to include repeated measures column indicators using the `repeated` argument for studies with randomized or multiple study lists. Our new dataframe `DF_long` will then be converted to long format:

```
> head(DF_long)
   Sub.ID  response  position  Disease.Condition
1       1    basket         1            healthy
2       1     chair         2            healthy
3       1   clothes         3            healthy
4       1    flowrs         4            healthy
5       1   glasses         5            healthy
6       1       fan         6            healthy
```

This function first splits the original response column by the separator, strips out additional whitespaces (i.e., two or more spaces become one between tokens), and trims whitespace characters before and after the token(s). The position column is added to denote the order of responses for each participant, and the unique identifier for each participant is repeated for each of their answers. The last component of this function is that all between-subject columns will be added back into the restructured dataframe, provided they have a one-to-one match with the participant identifier. In this example, the `Disease.Condition` variable is included because each participant was only assigned into one of the groups. If there are multiple trials or conditions for free responses, they should be separated into different dataframes and this process repeated for each trial-answer key pairing.

The answer key is structured as a dataframe with one column of information (shown below). However, the answer key can also be imported by simply typing the answers as a single vector using the concatenate function `c()`.

```
> head(answer_key_free)
 Answer_Key
1 backpack
2 basket
3 bed
4 blanket
5 blinds
6 books
```

As with cued-recall scoring, the free-recall functions are case sensitive and cannot process missing responses. As such, we again recommend converting both the answer key and response columns to lowercase.

## Scoring free-recall data

With the restructured data and answer key, the `prop_correct_free()` function can be used to score the free-recall data. This function will compare the answer key to the response column created above, and therefore, each trial of free-recall responses should be analyzed separately.

```
> free_output <- prop_correct_free(data =
DF_long,
+       responses = "response",
+       key = answer_key_free$Answer_Key,
+       id = "Sub.ID",
+       cutoff = 1,
+       flag = TRUE,
+       group.by = "Disease.Condition")
```

The arguments for this function are the same as the `prop_correct_cued()` function, minus the trial arguments for matching individual trials, as each trial should be analyzed separately. It is important to note that a non-space delimiter should be used, as spaces may interfere with multiple word tokens (i.e., picture frame is one correctly recalled concept in the answer key). We can then view the separate outputs by printing out the overall dataframe scored:

```
> free_output$DF_Scored
  Responses  Sub.ID  position  Disease.Condition    Answer  Scored
1   bacpack       3         1            healthy  backpack       1
2    basket       1         1            healthy    basket       1
3       bed       2         2            healthy       bed       1
4       bed       4         6               sick       bed       1
5       bed       5         1               sick       bed       1
6       bed       6         1               sick       bed       1
```

This dataframe can also be used to ensure that the answers appear to match the appropriate target word, as two target concepts that are within one substitution of each other may present scoring issues within this framework (i.e., if the answer list included both *bed* and *bet*). In this example, we included a group.by argument, and therefore, the participant data shows the grouping variable and calculates the *z*-scores for both participants overall (`Z.Score.Participant`) and within their own group (`Z.Score.Group`):

```
> free_output$DF_Participant
  Disease.Condition Sub.ID Proportion.Correct Z.Score.Group Z.Score.Participant
1           healthy      1          0.3928571     0.5773503           1.0819232
2           healthy      2          0.1428571    -1.1547005          -1.3096965
3           healthy      3          0.3928571     0.5773503           1.0819232
4              sick      4          0.3214286     1.1547005           0.3986033
5              sick      5          0.2142857    -0.5773503          -0.6263766
6              sick      6          0.2142857    -0.5773503          -0.6263766
```

Last, we find a group summary of mean, standard deviation, and sample size by using:

```
> free_output$DF_Group
  Disease.Condition      Mean         SD N
1           healthy 0.3095238 0.14433757 3
2              sick 0.2500000 0.06185896 3
```

## Scoring free-recall responses from multiple lists

Although `prop_correct_free()` was designed for scoring output from free-recall studies, it is primarily limited to research designs in which all participants study items taken from a single list. This function can be used in situations where participants study multiple lists or lists are randomized within participants by scoring each list separately. This approach, however, quickly becomes tedious as the number of separate lists and answer keys increases. Therefore, `prop_correct_multiple()` can be used to score free-recall responses when participants study multiple sets of lists or lists are randomly generated by software. This function takes the same basic inputs as `prop_correct_free()`. The input data must be a dataframe in long format, and the user must specify which dataframe columns contain the participant responses, the answer key, and the participant identifier. Additionally, `prop_correct_multiple()` also requires a list identifier (i.e., a unique value indicating which list response items and key items belong to). Thus, both the dataset and answer key will require additional columns denoting this information to indicate which answer key matches each trial. While key and trial ID information can be either character strings or numeric values, the list identifiers must match between the dataset and the key. Use `key.trial` and `id.trial` arguments to specify columns containing list identifiers for the answer key and input data columns, respectively. These arguments are demonstrated below in other *lrd* functions, and for a complete example of multiple list recall scoring in *R*, use `vignette("Multi_Recall", package = lrd)`.

## Calculating serial-position-based measures in free recall

The order of the answer key can be used to calculate serial position estimates, conditional response probabilities, and probability of first recall. Serial-position analyses compute proportions of correct recall as a function of the list position presented at study and are often used to plot serial-position curves wherein participants are more likely to remember the first and last items of a list best (Murdock, 1962). Lag-conditional response probabilities (lag-CRPs), compute probabilities for correctly recalled items conditionalized on the distance (i.e., lag) between items presented at study (Kahana, 1996; Kahana et al., 2002). Typically, lag-CRPs indicate that recall is highest at adjacent lags (+1 and -1) and lowest for more distant lags. For instance, if an item from list position 4 is recalled, recall from adjacent positions (3 and 5) is more likely than from more distant positions. Finally, probability of first recall (PFR) refers to the recall probability of the first recalled item from each study set as a function of the studied list position. PFRs typically indicate greater recall from late position items on immediate tests (recency effect), but greater recall for early position items if the test is delayed (primacy effect; Wahlheim & Huff, 2015). For each of these functions, the output from `free_output$DF_Scored` is the expected input. Vignette "Free_Recall" contains examples for plotting each function with *R* using *ggplot2* (Wickham et al., 2021).

To create a dataframe of the percent correct by serial position, we can use the `serial_position()` function:

```
> serial_output <- serial_position(data = free_output$DF_Scored,
+                                  key = answer_key_free$Answer_Key,
+                                  position = "position",
+                                  scored = "Scored",
+                                  answer = "Answer",
+                                  group.by = "Disease.Condition")
> head(serial_output)
  Disease.Condition  Tested.Position  Freq  Proportion.Correct         SE
1           healthy                1     1           0.3333333  0.2721655
2           healthy                2     1           0.3333333  0.2721655
3           healthy                2     1           0.3333333  0.2721655
4              sick                3     0           0.0000000  0.0000000
5              sick                4     0           0.0000000  0.0000000
6              sick                5     0           0.0000000  0.0000000
```

In the function, you use similar arguments as our previous examples. The `data` is likely a dataframe processed from the free-recall functions. The `key` column represents the ordered answer key. The `position` argument denotes the position the participant listed each answer, while the `scored` argument indicates the 0 or 1 denoting whether the answer was scored correct. Last, you can include grouping variables with the `group.by` argument. The output is shown next, which returns information about the grouping conditions (if included), the position a word was tested (`Tested.Position`), the total number of times the word was correctly indicated within the acceptable position window (`Freq`), the proportion correct (i.e., sum divided by participants, `Proportion.Correct`), and the standard error (`SE`). Items are added to the `Sum` column if they are scored as correct and are listed within one item lag of the original tested position (i.e., item 10 would be correct in position 9 or 11), except for the first and last item, which are only considered correct listed as first or last. These data can be used to create a serial position curve visualization or further descriptive statistic calculations based on researcher interest. Similar outputs for conditional response probabilities (CRPs) using the `crp()` function:

```
> crp_output <- crp(data = free_output$DF_Scored,
+ key = answer_key_free$Answer_Key,
+ position = "position",
+ scored = "Scored",
+ answer = "Answer",
+ id = "Sub.ID")
> crp_output[ c(28:54), ]
```

```
    Sub.ID  participant_  Freq  Possible.Freq  Disease.Condition        CRP
                    lags
28       1             1     1              8            healthy  0.1250000
29       1             2     1              9            healthy  0.1111111
30       1             3     1              9            healthy  0.1111111
31       1             4     1              9            healthy  0.1111111
32       1             5     1              9            healthy  0.1111111
33       1             6     1              9            healthy  0.1111111
34       1             7     0              0            healthy  0.0000000
35       1             8     0              0            healthy  0.0000000
36       1             9     0              0            healthy  0.0000000
37       1            10     1              8            healthy  0.1250000
38       1            11     0              0            healthy  0.0000000
39       1            12     0              0            healthy  0.0000000
40       1            13     0              0            healthy  0.0000000
41       1            14     1              5            healthy  0.2000000
42       1            15     0              0            healthy  0.0000000
43       1            16     0              0            healthy  0.0000000
44       1            17     0              0            healthy  0.0000000
45       1            18     0              0            healthy  0.0000000
46       1            19     0              0            healthy  0.0000000
47       1            20     0              0            healthy  0.0000000
48       1            21     0              0            healthy  0.0000000
49       1            22     0              0            healthy  0.0000000
50       1            23     0              0            healthy  0.0000000
51       1            24     0              0            healthy  0.0000000
52       1            25     0              0            healthy  0.0000000
53       1            26     0              0            healthy  0.0000000
54       1            27     0              0            healthy  0.0000000
```

The above example displays CRP values across all positive lags for participant 1 (note that this function also returns negative lags, but for concision, they have been omitted from this example). Each lag between subsequent named items is tallied and then divided by the possible combinations of subsequent lags given their response patterns. Therefore, the column `participant_lags` represents the lag between the studied position and the test position the item is recalled (e.g., *chair* was reported in the second position on the test, which represents a lag of 6 from spot number 8 on the answer key list which corresponds to the studied order). The column `Freq` represents the frequency of the lags between listed and shown position, while the `Possible.Freq` column indicates the number of times that frequency could occur given each answer listed (e.g., given the current answer, a tally of the possible lags that could still occur). The `CRP` column calculates the conditional response probability, or the frequency column divided by the possible frequencies of lags. Given that conditional response probability is calculated by participant, there is no `group.by` argument; however, the columns are returned as part of the dataframe for further analysis if desired. Last, we provide calculations for the probability of first response:

```
> pfr_output <- pfr(data = free_output$DF_Scored,
+ key = answer_key_free$Answer_Key,
+ position = "position",
+ scored = "Scored",
+ answer = "Answer",
+ id = "Sub.ID",
+ group.by = "Disease.Condition")
> head(pfr_output)
```

| | Tested.Position | Disease.Condition | Freq | pfr |
|---|---|---|---|---|
| 1 | 1 | healthy | 1 | 0.3333333 |
| 2 | 2 | healthy | 1 | 0.3333333 |
| 3 | 3 | healthy | 0 | 0.0000000 |
| 4 | 27 | healthy | 0 | 0.0000000 |
| 5 | 28 | healthy | 1 | 0.3333333 |
| 6 | 1 | sick | 0 | 0.0000000 |

Participant answers are first filtered for their first response, and these are matched to the original order on the answer key list (`Tested.Position`). Then the frequency (`Freq`) of each of those answers is tallied and divided by the number of participants overall or by group if the `group.by` argument is included (`pfr`). Each of these functions can be used to calculate further statistics or create visualizations, and the included *Shiny* application automatically displays basic visualizations of these functions as part of the free-recall output. Each of these functions contains a corresponding multiple list version (`serial_position_multiple()`, `crp_multiple()`, and `pfr_multiple()`) in order to accommodate output from `prop_correct_multiple()`. In each function, use the `key.trial` and `id.trial` arguments to match the answer key to the specific trial list. The sentence scoring functions below show an example of these arguments.

## Sentence scoring function example

In addition to scoring cued-recall and free-recall responses, *lrd* can also be used to score the match between two sentences. In this section, we provide a general overview of using *lrd* to score sentences. This example uses a set of simulated sentence responses generated for six participants. We begin this section by detailing the creation of this dataset. We then provide a detailed walkthrough of the *lrd* package's sentence processing functionality.

## Materials and dataset creation

To simulate a set of sentence responses, we first generated a list of five simple sentences to serve as the answer key. Next, to simulate a set of responses, sample data were generated for six participants, leading to a total of 30 observations. To capture response variability, we varied the amount of error within each response, such that some sentences included spelling errors, inclusion of extra words, omission of words, and/or semantically similar words. The full sample dataset and all code used in the following example has been made available at https://osf.io/admyx/.

## Formatting and loading the dataset

To view this example in *R*, use `vignette("Sentence_Recall", package = "lrd")`. As with our cued-recall example, the data will need a participant identifier, a trial identifier, the answer key, the participant's answer, and any other grouping columns that may be present. The answer key can be stored separately from the participant answers, as the trial identifier will be used to merge them together. Unless sentence case is part of the requirements for the study, we would recommend normalizing all text to lowercase. In our example below, we simulated extra whitespaces that participants may use, which will be eliminated as part of the scoring function.

| 1 | Sub.ID | Trial.ID | Sentence | Response | Condition |
|---|---|---|---|---|---|
| 2 | 1 | 1 | This is a sentence. | This is a sentence. | a |
| 3 | 1 | 2 | Woo more sentences! | Woo more sentences! | a |
| 4 | 1 | 3 | This is the thing the participant typed. | This is thing the participant typed. | a |
| 5 | 1 | 4 | This is another example sentence. | This is another example sentence | a |
| 6 | 1 | 5 | Okay this is the final thing that they typed. | Okay this is the final thing they typed. | a |
| 7 | 2 | 1 | This is a sentence. | This is sentence. | b |

## Scoring sentence data

To score sentence data with *lrd*, begin by running `prop_correct_sentence()` and save the output as a new object. This function follows the same general format as the cued- and free-recall scoring functions. You should specify the dataframe (`data`), the columns containing the participant responses (`responses`), the answer key (`key`), the participant identifier (`id`), and the trial identifiers (`id.trial`, `key.trial`). Each token will be scored individually against the target answer, and the `cutoff` indicates the allowable Levenshtein distance. Again, you can `flag` outliers by participant and/or grouping variable (`group.by`). The `token.split` argument is used to specify the character that separates words in each sentence, and the default is a single space.

```
> sentence_ouptut <- prop_correct_sentence(data = sentence_data,
  + responses = "Response",
  + key = "Sentence",
+ key.trial = "Trial.ID",
+ id = "Sub.ID",
+ id.trial = "Trial.ID",
+ cutoff = 1,
+ flag = TRUE,
+ group.by = "Condition",
+ token.split = " ")
```

The overall output contains the scoring for each participant and item. The entire output can be viewed on our vignette, and for our example here, we display the new columns that are added to the dataframe and a few rows with specific examples of the issues one might encounter when scoring sentences.

```
> sentence_ouptut$DF_Scored
      Proportion.Match  Shared.Items              Corrected.Items  Omitted.Items           Extra.Items
2     0.7500000         this is sentence          <NA>             a                       <NA>
19    0.8000000         this is example sentence  <NA>             another                 an extra
24    0.6000000         this is another           <NA>             example sentence        one
29    0.4444444         this is final             th               okay thing that typed   one
22    1.0000000         is another sentence       tis xample       <NA>                    <NA>
10    1.0000000         woo more sentences        <NA>             <NA>                    <NA>
```

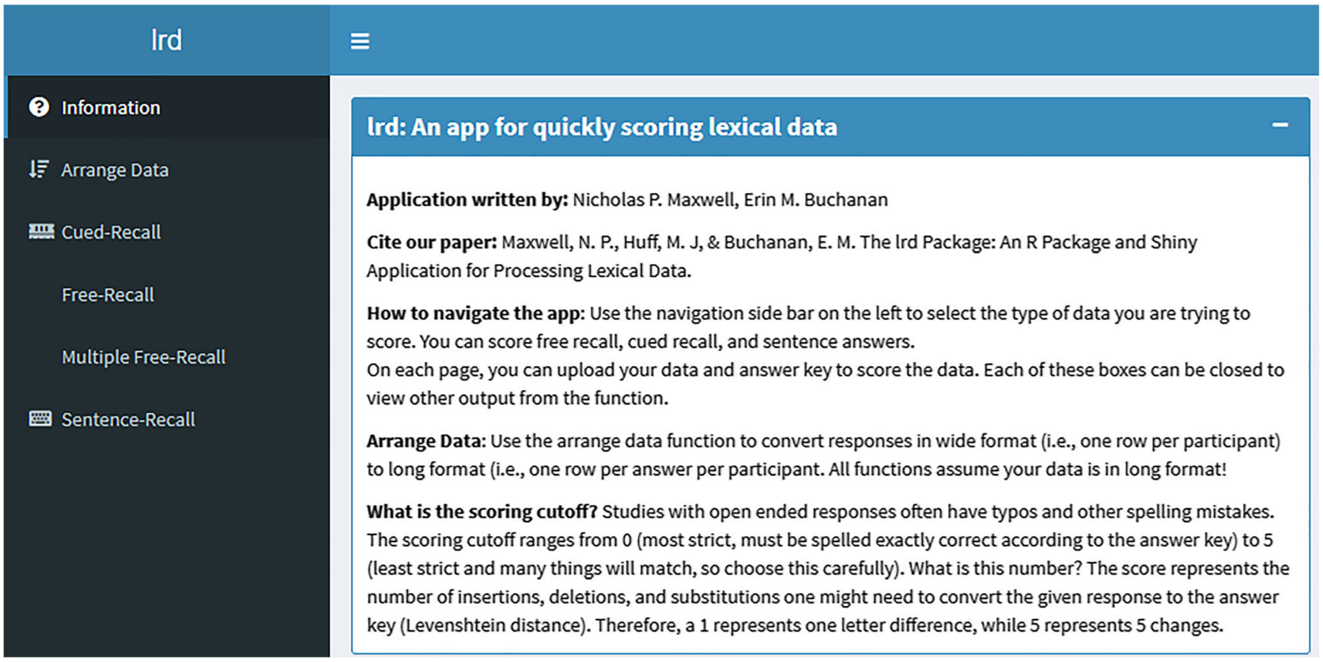
First, each sentence is stripped of punctuation and extra white space within the function. The total number of tokens, as split by `token.split` are tallied for calculating `Proportion.Match`. Then, the tokens are matched using the Levenshtein distance indicated in `cutoff`, as with the cued and free-recall functions. The key difference in this function is how each type of token is handled. The `Shared.Items` column includes all the items that were matched completely with the original answer (i.e., a `cutoff` of 0). The tokens not matched in the participant answer are then compared to the tokens not matched from the answer key to create the `Corrected.Items` column. This column indicates answers that were misspelled but within the cutoff score and were matched to the answer key (i.e., *th* for *the*, *ths* for *this*). The non-matched items are then separated into `Omitted.Items` (i.e., items in the answer key not found in the participant answer), and `Extra.Items` (i.e., items found in the participant answer that were not found in the answer key). The `Proportion.Match` is calculated by summing the number of tokens matched in `Shared.Items` and `Corrected.Items` and dividing by the total number of tokens in the answer key. Both `sentence_ouptut$DF_Participant` and `sentence_ouptut$DF_Group` can be used to view output that has the same information as shown in the cued- and free-recall sections for participant and group summaries of proportion correct.

## R shiny application

While *lrd* was initially designed as a package to be used within the *R* command environment, we recognized the need for an easy-to-access option that can be used independent of *R*. As such, we have developed a *Shiny* application that provides researchers with a programming-free alternative to using this tool that can be operated using basic Excel skills. Furthermore, because this application is web based (available at https://npm27.shinyapps.io/lrd_shiny/), no software downloads are required. The *lrd Shiny* application is structured as a series of tabs. Upon opening the application, you will be directed to the Information Tab (see Fig. 1). From here, the menu on the left can be used to navigate to the appropriate scoring task. Selecting a task will open a new tab in which the data and answer key can be uploaded, and other parameters can be set. For all scoring tasks, data will need to be structured in long-format, and the Arrange Data tab can be used prior to scoring to first convert wide-format data into the appropriate format. Additionally, each of the three scoring tabs provides options for downloading scored output for use in *R*, SPSS, Excel, or other programs. In the following sections, we provide detailed explanations for each of the three scoring tabs.

## Cued-recall tab

When using the *Shiny* application to score cued-recall data, the uploaded dataset needs to be arranged in long format and must contain the following columns: A unique participant identifier, participant responses, and a trial number for each recall trial. Additionally, cued-recall scoring requires an answer key and trial identifier for each key item (such as a trial number). If the answer key is embedded in the original participant data, simply upload the participant data in the answer file section. Finally, the dataset being scored may contain other columns (e.g., such as those denoting experimental conditions, demographics, etc.),



**lrd: An app for quickly scoring lexical data**

**Application written by:** Nicholas P. Maxwell, Erin M. Buchanan

**Cite our paper:** Maxwell, N. P., Huff, M. J, & Buchanan, E. M. The lrd Package: An R Package and Shiny Application for Processing Lexical Data.

**How to navigate the app:** Use the navigation side bar on the left to select the type of data you are trying to score. You can score free recall, cued recall, and sentence answers.
On each page, you can upload your data and answer key to score the data. Each of these boxes can be closed to view other output from the function.

**Arrange Data:** Use the arrange data function to convert responses in wide format (i.e., one row per participant) to long format (i.e., one row per answer per participant. All functions assume your data is in long format!

**What is the scoring cutoff?** Studies with open ended responses often have typos and other spelling mistakes. The scoring cutoff ranges from 0 (most strict, must be spelled exactly correct according to the answer key) to 5 (least strict and many things will match, so choose this carefully). What is this number? The score represents the number of insertions, deletions, and substitutions one might need to convert the given response to the answer key (Levenshtein distance). Therefore, a 1 represents one letter difference, while 5 represents 5 changes.

**Fig. 1** Illustration of the *lrd Shiny* application's Instructions tab prior to uploading a dataset

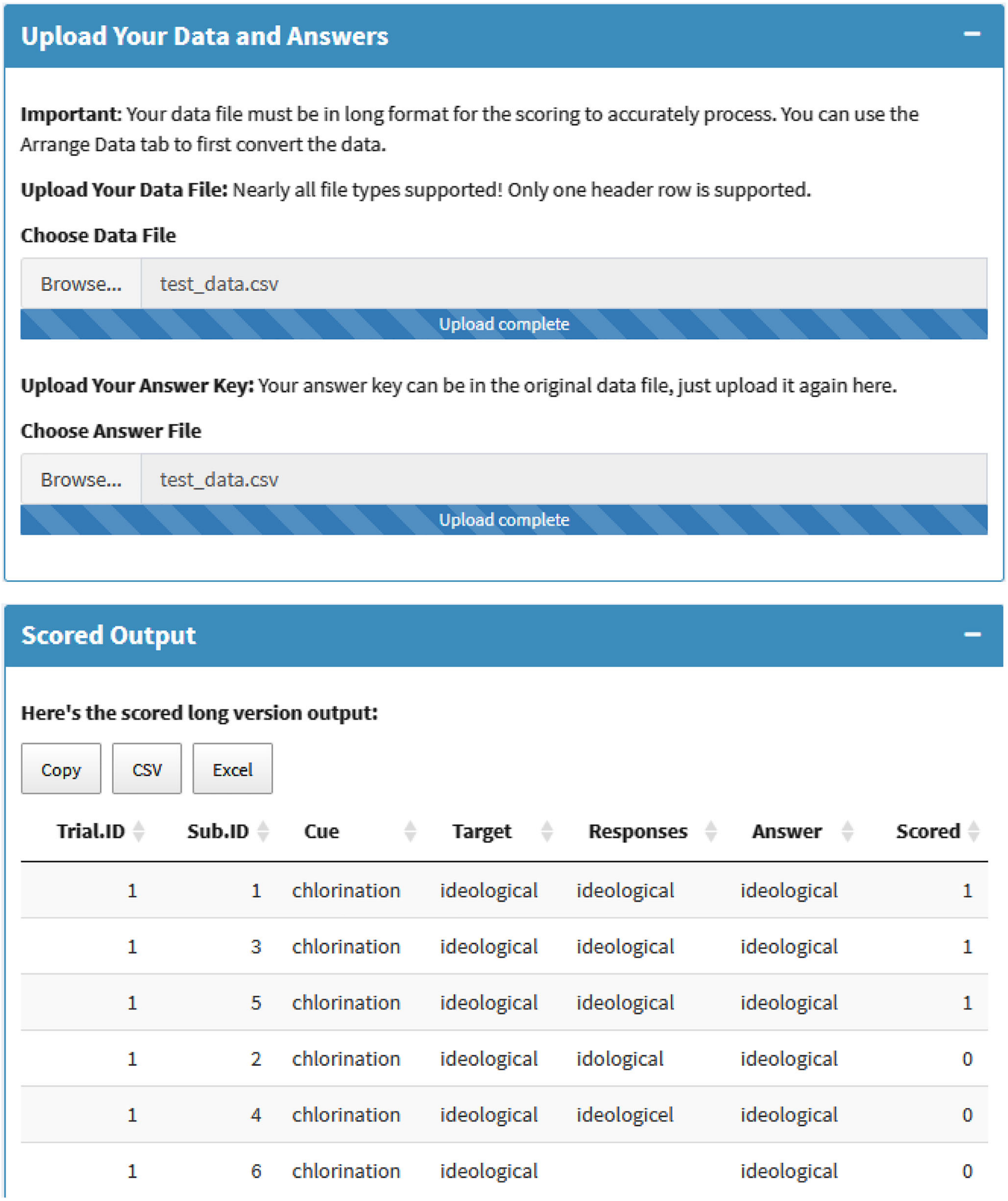
which can be selected using the "group by" option. Figure 2 (top panel) displays the cued-recall tab following data upload. The Check Your Data box can be used to determine that the data were uploaded correctly. The *rio* library is used to import data and generally provides a correct interpretation of the data, but errors can occur, generally with data that does not have a clear column header row (Becker et al., 2021).

After uploading both the dataset and answer key, the Scoring Set Up box can be used to indicate the columns for each relevant argument, the scoring cutoff, and the ability to provide *z*-scores for outliers. Like the *R* package, the scoring denotes the Levenshtein distance between the participant response and key item and represents the total number of insertions, deletions, and substitutions one might need to convert the given response to the answer key. Therefore, a selection of 1 represents a one-letter difference between the response item and the key, while a selection 5 represents five changes.

Once the appropriate settings have been selected, clicking "Score Your Data" will begin the scoring process. Scored output will then be displayed in the Scored Output box located below the data upload panel corresponding to the `DF_Scored` information from the *R* function (see Fig. 2 bottom panel). Next, the Summarized Output box displays the participant dataframe with the correct recall proportions. If a grouping variable was selected, recall proportions at the group level will also be displayed. Finally, the bottom panel displays recall proportions plotted as a function of the grouping variable. If no grouping variable is selected, this panel will display a histogram of recall proportion. At the top of each dataframe output, a set of buttons can be used: Copy for simple cut and paste of the dataframe, CSV for comma separated download, or Excel for .xlsx download. Graphs can be saved by right clicking to save or copy the graph but cannot be edited within the application.

## Free-recall tab

As with cued-recall scoring, free-recall data will also need to be uploaded in long-format. For simplicity, we suggest uploading the response data and answer key as two separate files. First, the participant response data will need to contain at minimum the responses and a unique participant identifier, along with grouping variables if desired. Next, the answer key file should at least contain one column of answers ordered by serial position. Free-recall data are then scored using the same general procedure as cued recall. Use the Scoring Set Up box to denote the appropriate columns for each argument, select a cut-off score, and flag for outliers. If the upload data contain a column denoting the order in which items were recalled, this column can be selected using the "position answered" box and will be used for creating serial position curves, lag-CRPs, and PFRs. Note that this column is automatically generated if the Arrange Data tab is used to convert the upload data from wide to long-format (Fig. 3). The scored dataset is then previewed in the Scored Output box, along with the Summarized Output box at the participant and grouping levels (Fig. 4). A graph of the free-recall proportions will be automatically created as a function of the grouping variable(s) selected or a histogram of participant-level responses. The last three boxes include the serial position, lag-CRP, and PFR



**Important:** Your data file must be in long format for the scoring to accurately process. You can use the Arrange Data tab to first convert the data.

**Upload Your Data File:** Nearly all file types supported! Only one header row is supported.

**Choose Data File**

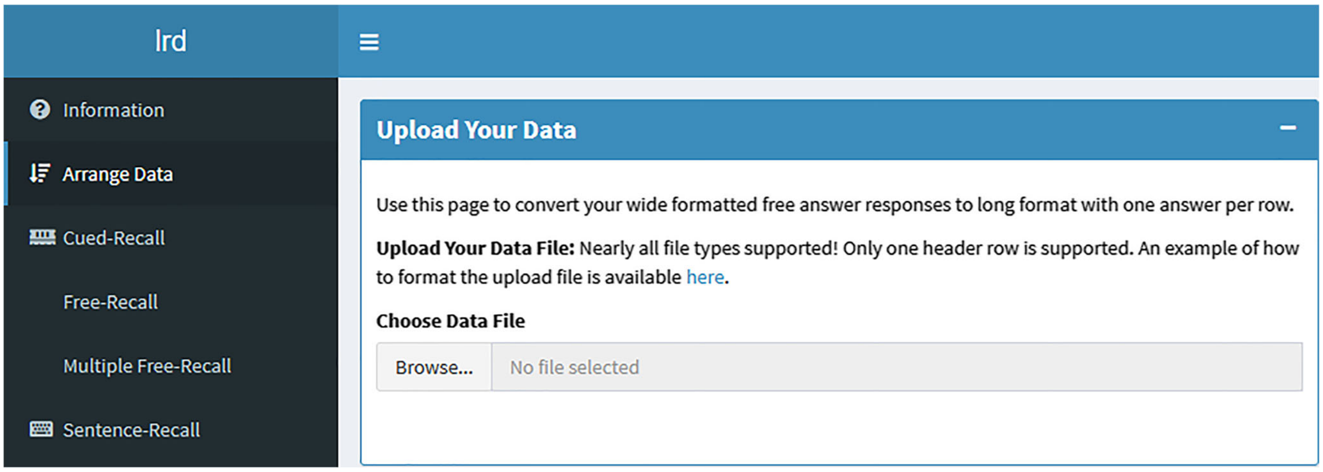
**Upload Your Answer Key:** Your answer key can be in the original data file, just upload it again here.

**Choose Answer File**

**Here's the scored long version output:**

| Trial.ID | Sub.ID | Cue | Target | Responses | Answer | Scored |
|---|---|---|---|---|---|---|
| 1 | 1 | chlorination | ideological | ideological | ideological | 1 |
| 1 | 3 | chlorination | ideological | ideological | ideological | 1 |
| 1 | 5 | chlorination | ideological | ideological | ideological | 1 |
| 1 | 2 | chlorination | ideological | idological | ideological | 0 |
| 1 | 4 | chlorination | ideological | ideologicel | ideological | 0 |
| 1 | 6 | chlorination | ideological |  | ideological | 0 |

**Fig. 2** Illustration of the *lrd Shiny* application's cued-recall tab prior to uploading a dataset and answer key (*top panel*) and the Scored Output panel after scoring (*bottom panel*). Data in this is example is scored using a Levenshtein distance of 1



Upload Your Data

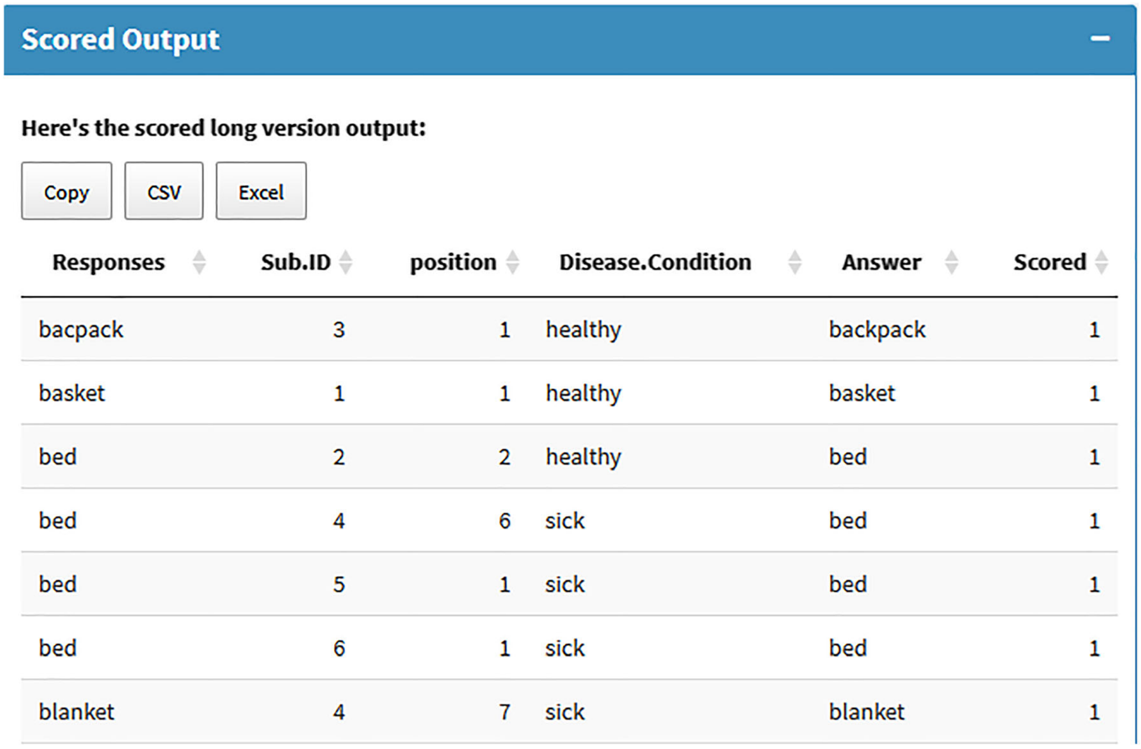
Use this page to convert your wide formatted free answer responses to long format with one answer per row.

**Upload Your Data File:** Nearly all file types supported! Only one header row is supported. An example of how to format the upload file is available here.

**Choose Data File**

Browse... No file selected

Fig. 3 Illustration of the *lrd Shiny* application's Arrange Data tab. This tab can be used to quickly convert wide format free-recall data into long format for scoring



Scored Output

**Here's the scored long version output:**

| Responses | Sub.ID | position | Disease.Condition | Answer | Scored |
|---|---|---|---|---|---|
| bacpack | 3 | 1 | healthy | backpack | 1 |
| basket | 1 | 1 | healthy | basket | 1 |
| bed | 2 | 2 | healthy | bed | 1 |
| bed | 4 | 6 | sick | bed | 1 |
| bed | 5 | 1 | sick | bed | 1 |
| bed | 6 | 1 | sick | bed | 1 |
| blanket | 4 | 7 | sick | blanket | 1 |

Fig. 4 Illustration of the *lrd Shiny* application's Free-Recall scoring tab after scoring. Data were scored using a Levenshtein distance of 1

dataframes for download and their corresponding plots for visualization.

## Multiple free-recall tab

Because the Free-Recall tab is based on the `prop_correct_free()` function, it can only be used to score free-recall data in which all test items are derived from the same study list. The Multiple Free-Recall tab can be used to score free-recall data in which participants or groups are tested on items taken from several study lists. Like single list free-recall scoring, multiple list free-recall requires that the uploaded data be arranged in long format. Additionally, both the answer key and dataset will each need to contain a column denoting unique list identifiers. Data upload follows the same general process as used in the standard free-recall tab, and the Scoring Set Up box contains the same inputs for selecting the appropriate columns for each argument (e.g., participant responses, participant identifiers, group options, etc.) as well as two additional inputs for selecting the scoring key and dataset list identifiers (see Fig. 5). As with the Free-Recall tab, multiple recall scoring will return the scored dataset along with a summary of the output along with a plot of the free-recall proportions, either at the participant or group level. Finally, if a column containing serial position data is selected using the "position answered" box, serial position curves, lag-CRPs, and PFRs will be automatically generated and displayed in the last three boxes.

## Sentence-recall tab

When scoring sentence recall, the data upload process closely follows cued-recall data scoring. The data will need to be arranged in long-format such that each row corresponds to one participant response. Moreover, the response column should be structured such that each cell contains the full participant response (i.e., each cell in the response column contains a full sentence). Example data illustrating the format required for the upload data is available at: https://osf.io/mcg9f/. Finally, this data will need to contain each of the necessary columns from the cued-recall tab. The answer key can either be included as a column with the upload data or it can be uploaded as a separate file. When scoring sentence data, in addition to selecting the cutoff criteria, you will need to select the delimiter for sentence tokens (i.e., the character(s) separating each word within the sentences). The token delimiter can be typed into the box below the scoring cutoff selection, and the default is a single space. If a different character separates words, you will need to delete this space before typing a different character.

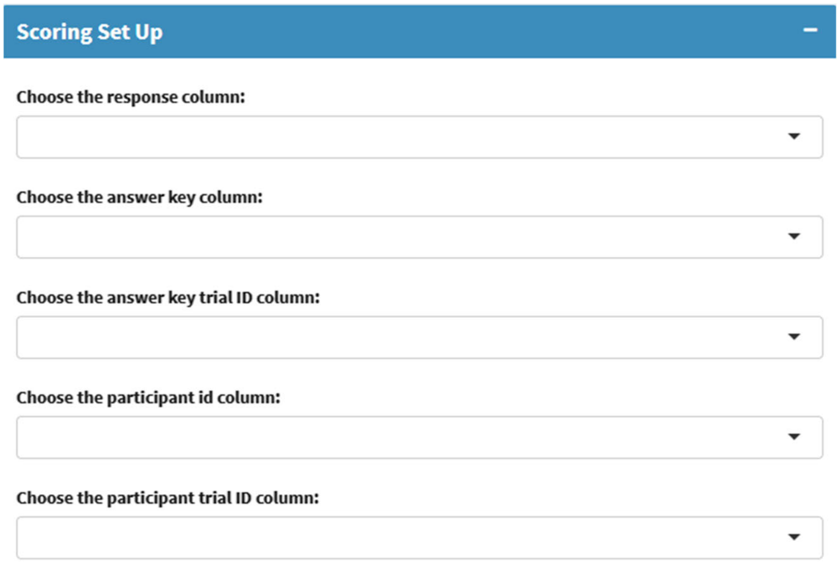
After scoring, the Scored Output box can be used to preview the final dataset. In addition to showing whether the sentence was correctly recalled, this panel will also return any tokens omitted from the answer key and any extra words included in the response (see Fig. 6). Next, the Summarized Output box displays participant-level and group-level (if specified) recall proportions. Finally, plots can be viewed via the graph box located at the bottom of this tab.

## Validation of scoring functions

We now turn to set of analyses designed to validate *lrd* package's scoring functions by using *lrd* to score cued-recall, free-recall, and sentence-recall datasets and comparing the output to human-coded data. For each recall task, we first conducted sensitivity and specificity analyses to determine the optimal Levenshtein distance for scoring before conducting additional analyses to assess accuracy and reliability. These analyses served as additional assessments to ensure that *lrd* can consistently produce accurate scoring across different sets of stimuli. We then tested whether the results of these studies would differ significantly from the original findings after the raw data were processed and scored using *lrd*, allowing us to test the accuracy of this package at the participant level. Finally, we computed Cohen's $\kappa$ to assess reliability between the different coding sources. We begin this section by testing *lrd*'s ability to score cued-recall data. Subsequent sections test scoring of free recall and sentence recall.

## Cued-recall scoring functions validation

In this section, we report results from two sets of analyses in which we tested the cued-recall scoring accuracy of *lrd*. First,


Scoring Set Up

Choose the response column:

Choose the answer key column:

Choose the answer key trial ID column:

Choose the participant id column:

Choose the participant trial ID column:

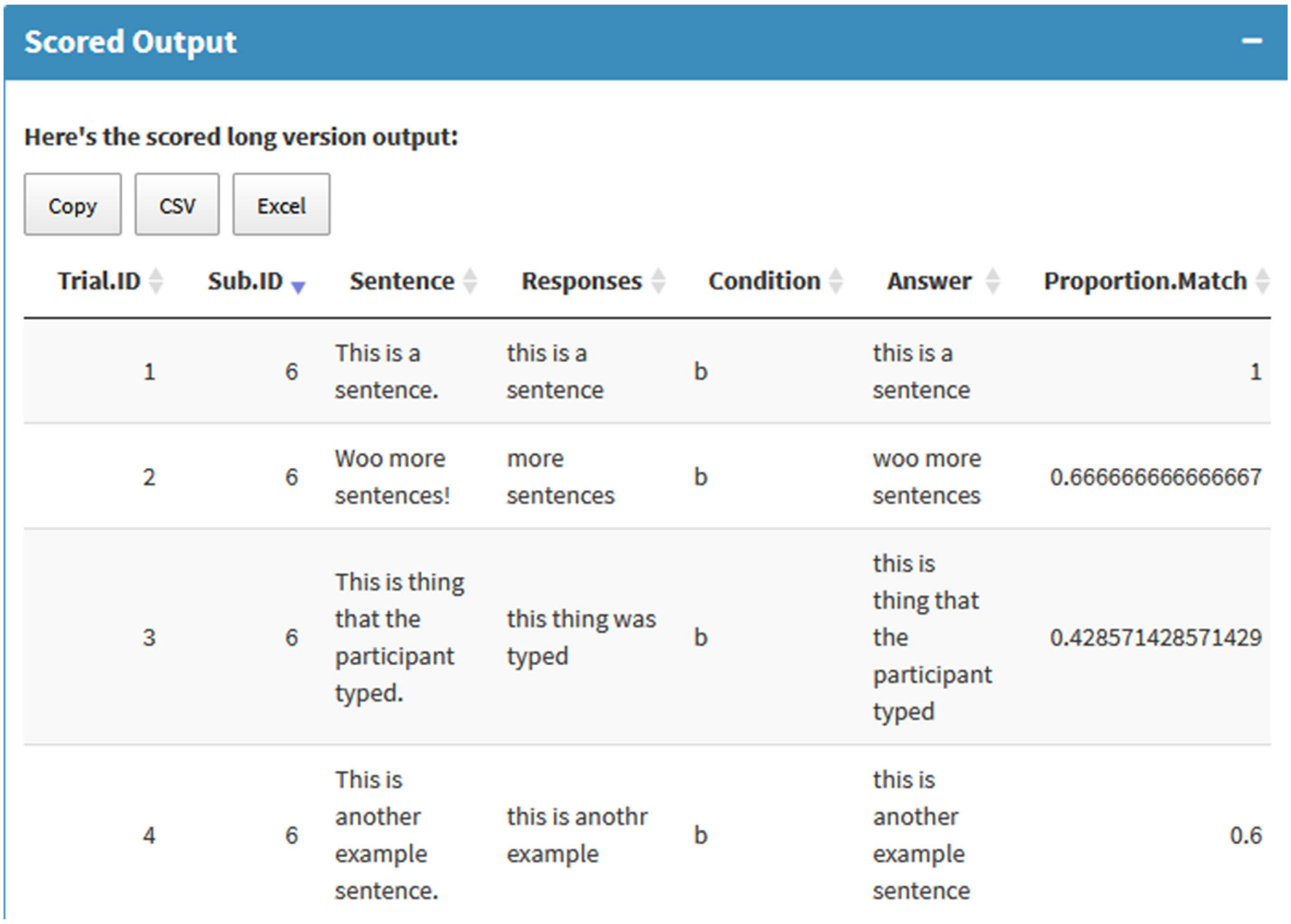
**Fig. 5** Illustration of the Scoring Set Up box for the Multiple Free-Recall tab. Columns denoting the answer key list ID and dataset list ID are selected using the "answer key trial ID" and "participant trial ID" columns

we use *lrd* to score the datasets used for each set of analyses. These data were derived from two sources: Maxwell and Buchanan (2020) and Maxwell and Huff (2021). We begin by providing details for each dataset, including participant and stimuli characteristics for each study. We then discuss the selection criteria for the Levenshtein distance value used at scoring and detail the results of a set of sensitivity and specificity analyses that were used to test potential cutoff values and provide a step-by-step walkthrough of the scoring process. Finally, we conclude this section by detailing each of the analyses described above.

## Participants and materials

Each dataset was collected separately across two different experimental settings. The first set of participants was originally reported in Maxwell and Buchanan (2020; dataset available at https://osf.io/y8h7v/). This dataset consists of 222 participants who were recruited online via Amazon's Mechanical Turk, a site which allows researchers to access a large pool of participants who complete surveys in exchange for small sums of money (Buhrmester et al., 2011). Next, Maxwell and Huff's (2021) data consists of 112 undergraduate students who were recruited from a psychology research pool at a large Southern university and tested in lab (dataset available at https://osf.io/hvdma/). These participants completed the study in exchange for partial course credit and were recruited to take part in one of four experiments. For purposes of this analysis, we collapsed across experiment to include all 112 subjects in one dataset. Combining datasets across studies resulted in 31,301 recall entries from 334 participants.

Datasets were selected due to their similarity in design. Each study presented participants with paired-associate study lists and later had them complete cued-recall tasks. Furthermore, each study contained reasonably sized samples (all *n*s > 90) and presented participants with at least 60 item pairs to study, providing us with a sufficient number of observations with which to test the reliability of this package. Each study presented participants with a set of cue-target paired associates (e.g., *credit–card*). Participants were asked to study



| Trial.ID | Sub.ID | Sentence | Responses | Condition | Answer | Proportion.Match |
|---|---|---|---|---|---|---|
| 1 | 6 | This is a sentence. | this is a sentence | b | this is a sentence | 1 |
| 2 | 6 | Woo more sentences! | more sentences | b | woo more sentences | 0.666666666666667 |
| 3 | 6 | This is thing that the participant typed. | this thing was typed | b | this is thing that the participant typed | 0.428571428571429 |
| 4 | 6 | This is another example sentence. | this is anothr example | b | this is another example sentence | 0.6 |

**Fig. 6** Illustration of the *lrd Shiny* application's Sentence-Recall scoring tab after scoring. Data were scored using a Levenshtein distance of 1

each pair before making a judgment of either the pair's relatedness or their ability to recall the pair at test. After completing the study and judgment tasks, participants then completed a cued-recall test. While participant judgments were collected in each experiment, they are not included in the following analyses as we are only interested in analyzing the accuracy of *lrd* in scoring recall responses.

## Data processing and scoring

To assess the reliability of the cued-recall scoring functions, we first used *lrd* to process and score the two cued-recall datasets introduced above. We then compared output obtained through this scoring process to the original, manually coded output reported in these studies and tested whether the initial findings would replicate. Prior to running the scoring algorithm, .csv files containing participant responses, answer key, trial numbers, and unique identifiers for each participant were generated for each dataset. Data from each study were then scored using the `prop_correct_cued()` function. Scoring was an iterative process, which used each of the suggested six Levenshtein distance values (i.e., 0–5). Thus, each dataset was scored six times, once for each scoring criterion, which allowed us to track how changing the Levenshtein distance affected scoring accuracy.

## Determining the optimal scoring criterion

Given the *lrd* package's scoring functions work by computing the Levenshtein distance between two strings, we first determined the optimal distance score that would maximize the number of correct hits (e.g., true positives) while minimizing the number of false positives and false negatives. To this end, we conducted a set of sensitivity and specificity analyses for each dataset (see Altman & Bland, 1994, for review) comparing each level of *lrd*-scored data to the original, human-coded data. Within the context of this study, sensitivity refers to the proportion of true positives that *lrd* correctly identifies (i.e., a participant correctly responds to the cue item with the correct target word and the program correctly identifies it) divided by the true positives plus false negatives (i.e., a miss, when the hand scoring indicates the item was right but *lrd* does not). Specificity refers to the proportion of true negatives identified by the program (i.e., the program correctly identifies that a participant missed an item at test) divided by the true negatives plus the false positives (i.e., *lrd* indicates the item was correct when the hand scoring did not).

Sensitivity and specificity analyses were computed in *R* using the *caret* package (Kuhn, 2008). Table 1 reports sensitivity and specificity percentages for each dataset computed across of the six Levenshtein distance cutoff values. Overall, both datasets displayed a consistent pattern of results: Sensitivity and specificity were each maximized when the scoring cutoff used a Levenshtein distance of 1, suggesting that this value allowed the scoring algorithm to achieve maximum accuracy. We therefore suggest that a Levenshtein distance of 1 provides the optimal cutoff value for minimizing false positives and negatives; however, the program allows researchers to increase or decrease the cutoff value as desired.

## Analyses and results

After determining the optimal range of cutoff values to use with the scoring functions, we now turn to a set of analyses that test whether data scored using *lrd* can successfully reproduce the results from each of the original, manually scored datasets. We begin this section by providing descriptive statistics of recall rates for both the original and rescored datasets and then test whether these recall rates differ as a function of coder. Finally, we compute the inter-rater reliability between the human-coded and *lrd*-scored data. Each dataset was analyzed individually, providing us with two separate tests of the *lrd* package's scoring accuracy. Generalized-eta squared ($\eta^2_G$) and Cohen's *d* effect sizes are reported for significant analyses of variance (ANOVAs) and *t*-tests, respectively. As effect size estimates provide a more useful interpretation of this package's scoring accuracy, we elected to set significance for analyses at the standard $\alpha < .05$ level. We note, however, that all significant main effects/interactions hold when using a more stringent $\alpha < .001$.

## Replication of cued-recall studies

To test whether cued-recall data scored using *lrd* could successfully replicate human-coded data, we conducted two one-way analysis of variance (ANOVA) models, which tested whether recall cued-rates differed between the seven scoring types (the six *lrd* scoring criteria ranging from 0 to 5 plus the human-coded data). For completeness, means, 95% CIs, and Cohen's *d* effect size indices for all comparisons are reported in Tables 2 and 3. Starting with the Maxwell and Buchanan (2020) dataset, a significant effect of scoring type was detected between the human-coded data and the *lrd* scored, $F(6, 1320) = 558.12$, $MSE = 3115.42$, $p < .001$, $\eta^2_G = .26$. However, post hoc analyses indicated that this effect was largely driven by differences between the higher Levenshtein distances (i.e., scored using a cutoff of 3 or greater) and the human-coded data ($ts \geq 3.19$, $ds \geq 0.29$). Recall rates from the *lrd*-scored data did not significantly differ from the human-coded data ($M = 54.14$) when *lrd* scoring used a Levenshtein distance of 0 ($M = 50.23$), 1 ($M = 52.14$), or 2 ($M = 53.37$; $ts \leq 1.23$, $ps \geq .209$).

**Table 1** Sensitivity and specificity results for cued recall

| Scoring criteria | Maxwell and Buchanan (2020) | | Maxwell and Huff (2021) | |
|---|---|---|---|---|
| | Sensitivity (%) | Specificity (%) | Sensitivity (%) | Specificity (%) |
| *lrd* 0 | 99 | 94 | 99 | 93 |
| *lrd* 1 | 99 | 96 | 99 | 97 |
| *lrd* 2 | 97 | 96 | 97 | 98 |
| *lrd* 3 | 80 | 97 | 87 | 99 |
| *lrd* 4 | 47 | 98 | 62 | 99 |
| *lrd* 5 | 24 | 99 | 40 | 99 |

*Note.* Column labels indicate Levenshtein distance used at scoring. Values denote percentages.

Next, for the Maxwell and Huff (2021) dataset, an effect of scoring type was also detected, $F(6, 666) = 1433.93$, $MSE = 14.82$, $p < .001$, $\eta^2_G = .53$. Post hoc analyses again showed that this effect was largely driven by differences in mean recall between the human-coded data ($M = 43.96$) data that was scored with *lrd* using a Levenshtein distance cutoff of 3 or greater, $t\text{s} \geq 3.86$, $d\text{s} \geq 0.52$. Recall rates did not differ between the human-coded data and any of the other *lrd* cutoff points, $t\text{s} \leq 1.60$, $p\text{s} \geq .110$. Thus, using *lrd* to score participant responses did not result in significant changes in outcome across any of the experiments, particularly when an optimized scoring criterion as based on the sensitivity and specificity analyses was used. As such, these findings suggest that *lrd* is able to code cued-recall data equivalently to human coders.

## Inter-rater reliability

To test the inter-rater reliability between the original data and the rescored data, we computed $\kappa$ values for all data sets at the individual trial level. These values were computed in *R* using the *psych* package (Revelle, 2020). The $\kappa$ statistic ranges from -1 to 1, and inter-rater reliability is considered strong if $\kappa$ exceeds .80 (Cohen, 1960). Beginning with the Maxwell and Buchanan (2020) data, a strong agreement was detected between the human-coded data and response sets scored using Levenshtein distances of 0, 1, and 2, $\kappa\text{s} \geq .96$, with this agreement weakening when the data were scored using higher Levenshtein distances. The Maxwell and Huff (2021) dataset showed a similar pattern of agreement between coding methods, with strong agreement for Levenshtein distances less than 3, $\kappa\text{s} \geq .94$, and weaker agreement when more liberal scoring criteria were used ($\kappa\text{s} \leq .85$). Table 4 reports individual $\kappa$ statistics for all comparisons within each dataset. Across datasets, reliability between human- and *lrd*-scored data was highest when a Levenshtein distance of 1 was used, and lowest when scoring used a Levenshtein distance of 5. These results provide further evidence that using *lrd* to score cued-recall responses results in output that is highly consistent with what is produced by human coders.

## Free-recall scoring functions validation

We now turn to a set of analyses in which we evaluated the *lrd* package's ability to accurately score free-recall data. First, we detail the dataset, including all participant and stimuli characteristics. We follow the same general procedure used to validate the cued-recall functions, including the use of sensitivity and specificity analyses to test potential cutoff values and comparing the *lrd*-scored output to the original human-coded data as a test of whether the original results can replicate. Finally, we conclude the analyses by computing Cohen's $\kappa$ to assess reliability between the various coding sources.

## Participants and materials

All data used in these analyses were originally published in Experiment 4A of Huff et al. (2018), who recruited 120 participants to complete the study online via Amazon's

**Table 2** Mean cued-recall rates as a function of human-coded and *lrd*-scored data collapsed across item type in Maxwell and Buchanan (2020)

| Group | *M* | HC | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 |
|---|---|---|---|---|---|---|---|
| Human-coded | 54.14 (3.47) | -- | | | | | |
| *lrd* 0 | 50.72 (3.58) | 0.13 | -- | | | | |
| *lrd* 1 | 52.14 (3.57) | 0.07 | 0.05 | -- | | | |
| *lrd* 2 | 53.37 (3.53) | 0.03 | 0.10 | 0.05 | -- | | |
| *lrd* 3 | 61.30 (2.91) | 0.29* | 0.43* | 0.37* | 0.32* | -- | |
| *lrd* 4 | 77.42 (1.86) | 1.10* | 1.24* | 1.17* | 1.12* | 0.87* | -- |
| *lrd* 5 | 88.48 (1.09) | 1.76* | 1.88* | 1.82* | 1.77* | 1.63* | 0.96* |

*Note.* Mean recall rates for each scoring condition. *95% CIs* are in parentheses. HC = Human-coded data. HC and *lrd* columns indicate Cohen's *d* effect sizes for post-hoc comparisons, * = $p < .05$.

**Table 3** Mean cued-recall rates as a function of human-coded and *lrd*-scored data collapsed across associative direction items in Maxwell and Huff (2021)

| Group | *M* | HC | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 |
|---|---|---|---|---|---|---|---|
| Human-coded | 43.96 (6.57) | -- | | | | | |
| *lrd* 0 | 41.06 (6.59) | 0.21 | -- | | | | |
| *lrd* 1 | 43.11 (6.59) | 0.06 | 0.15 | -- | | | |
| *lrd* 2 | 44.86 (6.58) | 0.06 | 0.28* | 0.13 | -- | | |
| *lrd* 3 | 50.83 (6.42) | 0.52* | 0.75* | 0.59* | 0.46* | -- | |
| *lrd* 4 | 64.84 (5.51) | 1.79* | 2.08* | 1.67* | 1.74* | 1.33* | -- |
| *lrd 5* | 77.69 (4.21) | 3.15* | 3.49* | 3.15* | 3.12* | 2.81* | 1.76* |

*Note.* Mean recall rates for each scoring condition. *95% CI*s are in parentheses. HC = Human-coded data. HC and *lrd* columns indicate Cohen's *d* effect sizes for post hoc comparisons, * = $p < .05$.

Mechanical Turk. Recall was assessed across three types of study lists: Categorical lists in which items were strongly related, ad hoc lists in which items were weakly related, and unrelated lists. Six lists of 20 items were generated, and participants studied two lists of each type (i.e., 40 of each item type). Following presentation of each list, participants completed a free-recall test. This experiment provided us with 720 individual free-recall tests (120 participants × 6 list presentations). Across tests, participants made a total of 6520 correct responses (out of 14,400 potentially correct responses).

## Data processing and scoring

To assess the reliability of the free-recall scoring functions, we began by generating a scoring key for each of the six lists. Lists were combined by type (ad hoc, categorical, unrelated) resulting in three unique key lists. Lists were then arranged into long format using the *reshape* package (Wickham, 2007). The final answer key file contained each list along with a "List Type" column denoting whether the key item belonged to the ad hoc, categorial, or unrelated list. We then used *lrd*'s `arrange_data()` function to convert participant responses into long format. Following reshaping of the dataset, we created a "List Type" column denoting which answer key to use for scoring, which corresponded to the "List Type" column in the answer key. The dataset was then scored using `prop_correct_multiple()`. This scoring was an iterative process which used each of the six cutoff values as used in the sensitivity and specificity analyses, allowing us to monitor how changes to the Levenshtein distance selected for scoring affected the scored output. The final dataset was created by combining the scored output at each of the six cutoff values with the original human-coded data.

**Table 4** Inter-rater reliability statistics (Cohen's κ) for Maxwell and Buchanan (2020) and Maxwell and Huff (2021)

| Experiment | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 | *lrd* 5 |
|---|---|---|---|---|---|---|
| MB | .93 | .95 | .94 | .79 | .49 | .24 |
| MH | .94 | .97 | .96 | .85 | .59 | .36 |

*Note.* MB = Maxwell and Buchanan, 2020; MH = Maxwell and Huff (2021). *lrd* columns indicate Levenshtein distance used at scoring. All values are Cohen's $\kappa$ between human-scored data and data scored at each *lrd* cutoff.

## Determining the optimal scoring criterion

Before testing whether *lrd* could successfully replicate human-coded free-recall data, we again needed to determine the optimal cutoff value for this function that would maximize the number of correct hits (e.g., true positives) while minimizing the number of false positives and false negatives. To do so, we again turn to a series of sensitivity/specificity analyses for each dataset. These analyses followed the same design used when validating the cued-recall functions. Table 5 displays sensitivity and specificity percentages for each dataset for each of the selected cutoff values. Each of the three datasets displayed similar results. Sensitivity and specificity were maximized when the Levenshtein distance was set at either 1 (ad hoc and categorical lists) or 0 (unrelated lists), though both cutoffs were very similar, indicating that either cutoff would be appropriate. We note, however, that both `prop_correct_free()` and `prop_correct_multiple()` allow this value to be edited as desired, providing users with maximum control over the scoring process.

## Analyses and results

We next conducted a series of analyses that tested whether free-recall data scored with *lrd* successfully replicates the results from the original human-coded dataset. First, we provide descriptive statistics of recall rates for both the original and rescored datasets. Next, we test whether these recall rates differ as a function of coding. Finally, we conclude this section by computing the inter-rater reliability between the human- and *lrd*-coded datasets.

## Replication of free-recall studies

First, data from each of the three list types were scored with *lrd* using all six Levenshtein distance cutoff values between 0 and 5. Next, three one-way ANOVAs were used to test whether recall rates differed between the seven scoring types (the six *lrd* scoring criteria plus the human-coded data) for each of the three study list types. For completeness, means, 95% CIs and Cohen's *d* effect size estimates for all comparisons are reported in Table 6. Beginning with the categorical list items, significant differences were detected between the human-coded data and the *lrd* scored, $F(6, 833) = 2.37$, $MSE = 261.02$, $p = .028$, $\eta^2_G = .017$. Post hoc testing, however, indicated that this effect was primarily driven by differences in mean recall between the human-coded data and the 0 cutoff, $t(237) = 1.99$, $SEM = .019$, $p = .048$, $d = 0.25$, and differences between the *lrd*-scored datasets at the strict and lenient cutoffs ($t$s $\geq 2.10$, $d$s $\geq 0.27$). Correct recall did not significantly differ between the human-coded data and any of the other *lrd* cutoff points, $t$s $\leq 1.63$, $p$s $\geq .105$. Furthermore, this pattern failed to extend to the ad hoc lists, $F(6, 833) = 1.19$, $MSE = 273.41$, $p = .310$, $\eta^2_G = .008$, and the unrelated lists, $F(6, 833) = 1.40$, $MSE = 294.51$, $p = .269$, $\eta^2_G = .009$, as no effect of scoring type was detected. As such, using *lrd* to score free-recall responses did not result in significant changes in outcome across any of the datasets, regardless of whether a strict or lenient scoring criterion was selected. Thus, the results of these analyses indicate that *lrd* can score free-recall data equivalently to human coders.

**Table 5** Sensitivity and specificity results for Huff et al.'s (2018) free-recall data

| List type | Scoring criteria | Sensitivity (%) | Specificity (%) |
|---|---|---|---|
| Ad-hoc | 0 | 96 | 94 |
| | 1 | 96 | 95 |
| | 2 | 94 | 95 |
| | 3 | 95 | 96 |
| | 4 | 91 | 96 |
| | 5 | 90 | 96 |
| Categorical | 0 | 98 | 90 |
| | 1 | 98 | 91 |
| | 2 | 96 | 91 |
| | 3 | 93 | 93 |
| | 4 | 91 | 93 |
| | 5 | 91 | 93 |
| Unrelated | 0 | 98 | 96 |
| | 1 | 97 | 96 |
| | 2 | 96 | 96 |
| | 3 | 93 | 96 |
| | 4 | 92 | 96 |
| | 5 | 92 | 96 |

*Note:* Analyses are split by list type. Scoring criteria indicates Levenshtein distance used when running `prop_correct_multiple()`.

## Inter-rater reliability

Finally, we computed $\kappa$ values for all data sets at the individual trial level as a test of inter-rater reliability. Starting with the categorical list, a strong agreement was detected between the human-coded data and the *lrd*-scored data when using each of the six scoring conditions, $\kappa$s $\geq .89$. Next, for the ad hoc dataset, a strong pattern of agreement was again detected, with agreement being strongest when scoring used cutoffs of 0, 1, and 2 ($\kappa$s $\geq .90$) while strength of agreement decreased when scoring used a cutoff of 3 or greater ($\kappa$s $\leq .87$). Finally, the unrelated list exhibited a pattern similar to the categorical lists, as the agreement observed between the human- and *lrd*-coded data was strongest when scored using cutoffs of 0, 1, and 2 ($\kappa$s $\geq .92$) and again decreased when using more lenient cutoffs ($\kappa$s $\leq .87$). Table 7 reports individual $\kappa$ statistics for all comparisons between human- and *lrd*-scored responses within each dataset. Based on the results of these analyses, we suggest using a Levenshtein cutoff of 1 when scoring free-recall responses. Taken together, the results of these analyses provide further evidence that free-recall data scored with *lrd* to is consistent to what is generated by human coders.

## Sentence scoring functions validation

We now detail a set of analyses that were designed to test *lrd*'s ability to accurately score sentence recall. We begin by providing a description of the dataset and note that these analyses closely follow the procedure used to validate both the cued- and free-recall functions by testing potential cutoff values for scoring, testing whether the *lrd*-scored output can replicate the original human-coded data, and assessing the reliability between coding sources.

## Participants and materials

Data in the following analyses were originally published as part of Geller et al. (2020) and are available at https://osf.io/ag7nc/. Geller et al. (2020) included 100 participants who listened to 20 sentences taken from AzBio (Spahr et al., 2012), a large, open-set database of recorded sentences.

**Table 6** Mean correct free-recall as a function of human-coded and *lrd*-scored data for each list type used in Huff et al. (2018)

| List Type | Group | *M* | HC | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd 4* |
|---|---|---|---|---|---|---|---|---|
| Ad hoc | Human coded | 50.00 (2.78) | -- | | | | | |
| | *lrd* 0 | 49.29 (2.74) | 0.05 | -- | | | | |
| | *lrd* 1 | 49.83 (2.73) | 0.01 | 0.03 | -- | | | |
| | *lrd* 2 | 50.69 (2.75) | 0.04 | 0.09 | 0.06 | -- | | |
| | *lrd* 3 | 52.48 (3.16) | 0.15 | 0.19 | 0.16 | 0.11 | -- | |
| | *lrd* 4 | 53.10 (3.24) | 0.18 | 0.23 | 0.20 | 0.14 | 0.03 | -- |
| | *lrd 5* | 53.15 (3.24) | 0.19 | 0.23 | 0.20 | 0.15 | 0.02 | < 0.01 |
| Categorical | Human coded | 47.85 (2.50) | -- | | | | | |
| | *lrd* 0 | 44.15 (2.67) | 0.25* | -- | | | | |
| | *lrd* 1 | 44.83 (2.64) | 0.21 | 0.05 | -- | | | |
| | *lrd* 2 | 45.58 (2.70) | 0.14 | 0.10 | 0.05 | -- | | |
| | *lrd* 3 | 48.54 (3.13) | 0.04 | 0.27* | 0.23 | 0.18 | -- | |
| | *lrd* 4 | 49.56 (3.25) | 0.11 | 0.33* | 0.29* | 0.24 | 0.06 | -- |
| | *lrd* 5 | 49.56 (3.25) | 0.11 | 0.33* | 0.29* | 0.24 | 0.06 | 0.00 |
| Unrelated | Human-coded | 37.99 (2.68) | -- | | | | | |
| | *lrd* 0 | 37.98 (2.76) | < 0.01 | -- | | | | |
| | *lrd* 1 | 38.08 (2.77) | < 0.01 | < 0.01 | -- | | | |
| | *lrd* 2 | 38.94 (2.83) | 0.06 | 0.06 | 0.05 | -- | | |
| | *lrd* 3 | 40.88 (3.37) | 0.17 | 0.17 | 0.16 | 0.11 | -- | |
| | *lrd* 4 | 41.17 (3.48) | 0.22 | 0.21 | 0.21 | 0.16 | 0.04 | -- |
| | *lrd 5* | 41.17 (3.48) | 0.22 | 0.21 | 0.21 | 0.16 | 0.04 | 0.00 |

*Note.* Mean recall rates for each scoring condition. *95% CI*s are in parentheses. HC = Human-coded data. *lrd* columns and row labels indicate each of the tested cutoff criteria. HC and percentage columns indicate Cohen's *d* effect sizes for post-hoc comparisons, * = *p* < .05.

After listening to each sentence, participants were instructed to immediately type each sentence from memory exactly as heard. Typed responses were then manually coded by two independent reviewers, leading to two sets of human-coded data (each consisting of 2000 responses) with which to test *lrd*'s sentence scoring functions.

## Data processing and scoring

To test the reliability of this package's sentence scoring capabilities, we began by using *lrd* to process the dataset described above using each of the six Levenshtein distance cutoffs. Because Geller et al. (2020) scored their output using two independent coders, we treated each coder as a separate dataset. Afterwards, we compared output obtained using *lrd* to each set of manually coded output and tested whether the *lrd*-scored data would replicate the original findings. Before running the scoring algorithm, we generated two .csv files (one for each human coder) containing participant responses, answer key, trial numbers, and unique identifiers for each participant. We then scored each dataset using the `prop_correct_sentence()` function. Consistent with the previous analyses, this scoring process was iterative such that we used each of the six Levenshtein distances. This resulted in each dataset being scored six times, allowing us again to track how changing the cutoff criteria affected scoring accuracy.

**Table 7** Inter-rater reliability statistics (Cohen's $\kappa$) for Huff et al's. (2018) free-recall data

| List type | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 | *lrd* 5 |
|---|---|---|---|---|---|---|
| Ad hoc | .90 | .91 | .90 | .87 | .86 | .86 |
| Categorical | .89 | .89 | .88 | .86 | .85 | .85 |
| Unrelated | .93 | .94 | .92 | .88 | .86 | .86 |

*Note.* List type corresponds to the three study lists conditions used in Huff et al. (2018). *lrd* columns indicate each of the tested cutoff criteria All values are Cohen's $\kappa$ between human-scored data and data scored at each *lrd* cutoff

## Determining the optimal scoring criterion

Before scoring the data, we again needed to determine the optimal cutoff value for the sentence-scoring function that would maximize the number of true positives while minimizing the number of false positives and false negatives. To do so, we again turn to a series of sensitivity and specificity analyses, comparing the *lrd*-scored data at each Levenshtein distance cutoff to each of the two human coders who originally scored the Geller et al. (2020) dataset. Table 8 displays sensitivity and specificity percentages for each of the six selected values. Overall, sensitivity and specificity were maximized when low Levenshtein distances ($\leq 1$) were selected, suggesting that these values maximized correct hits while simultaneously limiting false positives and negatives. As such, we propose that a value of 1 be selected when using *lrd* to perform sentence matching as this will provide some correction for minor discrepancies between the key and response (e.g., spelling errors), but note that as with the other scoring functions, this value can be modified as needed to provide flexibility in scoring.

## Analyses and results

After determining the optimal cutoff value for scoring, we next conducted a series of analyses testing whether sentence data scored with *lrd* successfully replicates the human-coded dataset. We begin this section by providing descriptive statistics for recall rates in both the human- and *lrd*-scored datasets and test whether these datasets significantly differ as a function of coding source. We then conclude our analyses of the sentence-recall data by assessing the inter-rater reliability between each dataset.

## Replication of sentence-recall studies

First, data from each of the three list types were scored with *lrd* using all six Levenshtein distance cutoff values between 0 and 5. Next, two one-way ANOVAs were conducted, testing whether recall rates differed between the seven scoring types (the six *lrd* scoring criteria plus the human-coded data) for each of the two human coders. Table 9 reports means, 95% CIs, and Cohen's *d* effect sizes for all comparisons.

Beginning with data scored by the first human coder, a significant difference was detected between the manually and *lrd*-scored data, $F(6, 594) = 204.37$, $MSE = 22.02$, $p < .001$, $\eta^2_G = .12$; however, post hoc *t* tests revealed that this effect was largely driven by differences between the human-coded data and the data scored with *lrd* using more lenient cutoffs. Specifically, recall rates differed from the human-scored data ($M = 31.80$) when a cutoff of 3 ($M = 40.15$), 4 ($M = 44.55$), or 5 ($M = 46.45$) were selected ($t$s $\geq 3.58$, $d$s $\geq 0.51$). However, the *lrd*-scored data did not differ from the human-coded data when it was scored using cutoffs of 0 ($M = 29.10$), 1 ($M = 32.90$), or 2 ($M = 34.25$; $t$s $\leq 1.12$, $p$s $\geq .265$). When compared to the second human coder, an effect of coding source was again detected, $F(6, 594) = 209.77$, $MSE = 21.89$, $p < .001$, $\eta^2_G = .12$. This effect largely followed the same patterns as the first human coder such that mean recall rates differed from the human-scored data ($M = 31.30$) when *lrd* scoring used cutoffs of 3 or greater ($t$s $\geq 3.75$, $d$s $\geq 0.53$). However, the *lrd*-scored data again did not differ from the human-coded data when scored using cutoffs less than 3 ($t$s $\leq 1.33$, $p$s $\geq .186$). Given the result of these analyses, using *lrd* to score sentence-recall did not result in significant changes when scoring used more stringent cutoff values (e.g., using a Levenshtein distance < 3). The results of these analyses suggest that when using the appropriate settings, *lrd* can score sentence responses with similar accuracy to human coders.

## Inter-rater reliability

Finally, we tested the inter-rater between each human coder and *lrd* by computing $\kappa$ values for at the individual trial level. Table 10 reports individual $\kappa$ statistics for all comparisons within each for each of the human coders. Beginning with sentences scored by the first human coder, a strong agreement was detected between the human- and *lrd*-scored data when a cutoff value of at least 2 was used, $\kappa$s $\geq .90$, and a moderate agreement was found when sentences were scored using a

**Table 8** Sensitivity and specificity results for Geller et al.'s (2020) sentence-recall data

| Scoring criteria | Coder 1 | | Coder 2 | |
|---|---|---|---|---|
| | Sensitivity (%) | Specificity (%) | Sensitivity (%) | Specificity (%) |
| *lrd* 0 | 99 | 91 | 99 | 93 |
| *lrd* 1 | 97 | 96 | 97 | 98 |
| *lrd* 2 | 95 | 97 | 95 | 98 |
| *lrd* 3 | 87 | 99 | 87 | 99 |
| *lrd* 4 | 81 | 99 | 80 | 99 |
| *lrd* 5 | 78 | 99 | 78 | 99 |

*Note.* Column labels indicate Levenshtein distance used at scoring. Values denote percentages. For completeness, we compare *lrd* sensitivity and specificity to both human coders from Geller et al. (2020)

**Table 9** Mean correct sentence-recall as a function of human-coded and lrd-scored data for each coder in Geller et al. (2020)

| Group | *M* | HC 1 | HC 2 | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd 4* |
|---|---|---|---|---|---|---|---|---|
| Human-coded 1 | 31.80 (2.88) | -- | | | | | | |
| Human-coded 2 | 31.30 (2.97) | 0.03 | -- | | | | | |
| *lrd* 0 | 29.10 (2.81) | 0.18 | 0.15 | -- | | | | |
| *lrd* 1 | 32.90 (3.10) | 0.07 | 0.10 | 0.25 | -- | | | |
| *lrd* 2 | 34.25 (3.18) | 0.16 | 0.19 | 0.34* | 0.08 | -- | | |
| *lrd* 3 | 40.15 (3.55) | 0.51* | 0.53* | 0.66* | 0.43* | 0.34* | -- | |
| *lrd* 4 | 44.45 (3.79) | 0.74* | 0.76* | 0.91* | 0.66* | 0.58* | 0.23 | -- |
| *lrd 5* | 46.45 (3.90) | 0.84* | 0.86* | 1.00* | 0.75* | 0.67* | 0.33* | 0.10 |

*Note.* Mean recall rates for each scoring condition. *95% CI*s are in parentheses. HC = Human-coded data. *lrd* columns and row labels indicate each of the tested cutoff criteria. HC and percentage columns indicate Cohen's *d* effect sizes for post hoc comparisons, * = *p* < .05.

cutoff of 3 or higher, *κ*s ≥ .69. This pattern extended to the second human coder. Again, a strong agreement between the *lrd*- and human-coded data emerged when a cutoff of at least 2 was used, *κ*s ≥ .91. A moderate agreement was again detected when sentences were scored using a cutoff value of 3 or higher, *κ*s ≥ .69. Thus, based on this set of results of these analyses, we propose using a Levenshtein cutoff of 1 when using *lrd* to score sentence recall, as this value provided strong agreement between both sets of human-coded data while still granting flexibility in participant responses due to minor errors. Given the strong agreement detected by these analyses, sentence data scored with *lrd* to is comparable to output generated by human coders.

## General discussion

The goal of *lrd* is to provide researchers with a free, open-source method for quickly and accurately processing lexical output from cued-recall, free-recall, and sentence-recall studies. Across each of these three memory tasks, data scored using *lrd* consistently matched output from human coders reliably and accurately. Importantly, our use of fuzzy string matching via Levenshtein distances provides *lrd* with increased flexibility in scoring compared to direct string matching. Furthermore, because researchers can select from a range of Levenshtein distances when setting up the scoring functions, this flexibility can be adjusted as needed. However, care must be taken when selecting this scoring cutoff, as our analyses showed that findings will change if too lenient (or strict) of a cutoff is used. Thus, an interesting question becomes which cutoff value provides sufficient flexibility to account for common participant errors without being so lenient that blatantly incorrect responses are counted as correct?

In general, our analyses suggest the use of a stricter cutoff when scoring recall data across the three test types. For example, across recall tasks, *lrd* provided the closest match to human coders when scoring used a Levenshtein distance of 1. We note, however, that both the type of recall task and the nature of the stimuli may need to be considered when selecting the cutoff value. This result is evident in our free-recall analyses, as scoring of categorical and ad hoc lists was most accurate when using a cutoff of 1, while scoring of unrelated pairs most closely matched human coders when a cutoff of 0 was used. However, we note that the difference between cutoffs of 0 and 1 were small and would likely produce fewer difference in the final analyses. Importantly, the flexibility of the *lrd* scoring criteria settings allows researchers to adjust scoring based on the materials and the frequency of response errors provided by participants. For instance, a more lenient cutoff may be appropriate if list items are prone to spelling errors, and a stricter cutoff may be appropriate if the study list contains several words that are within one or two characters of one another (e.g., *bear* and *tear*). Thus, while we recommend using a cutoff of 1 in most situations, the choice of cutoff value will ultimately be dictated by the nature of the recall task and the stimuli used.

Though we developed *lrd* to process lexical output from memory studies, recent work in the domain of speech perception has also led to the development of tools for automatically scoring lexical responses. Specifically, the *Autoscore* package for *R* (Borrie et al., 2019) and Bosker's (in press) application for computing Token Sort Ratio (TSR) each provide

**Table 10** Inter-rater reliability statistics (Cohen's *κ*) for each human coder used in Geller et al.'s (2020) sentence-recall data

| Coder | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 | *lrd* 5 |
|---|---|---|---|---|---|---|
| One | .93 | .92 | .90 | .81 | .72 | .69 |
| Two | .94 | .94 | .91 | .80 | .72 | .69 |

*Note. lrd* columns indicate Levenshtein distance used at scoring. All values are Cohen's *κ* between human-scored data and data scored at each *lrd* cutoff.

researchers with alternative methods for matching lexical text data and could be potentially applied to memory research. Like *lrd*, *Autoscore* and TSR were designed to expedite scoring of lexical text responses while maintaining the accuracy of a human coder. However, these applications differ both from each other and from *lrd* in their general approach to string matching, and furthermore, neither *Autoscore* nor TSR were specifically designed to score lexical output from memory studies. Instead, these tools were developed to assist with matching transcribed audio (typically sentences or phrases) to a key. Thus, while these tools can be used for single-word matching, they are most appropriate for sentence-matching tasks and are therefore akin to *lrd*'s sentence-recall function.

In addition to being designed primarily for matching responses from audio transcripts, these applications differ from *lrd* in other key aspects. First, *Autoscore* controls for participant errors via a set of hard-coded spelling and grammar rules that are available for the user to select from (e.g., match words based on root words, ignore double letters, etc.). While a rule-based approach can account for several common response errors, it requires researchers to guess which type of errors will be most likely to occur, rather having the flexibility to account for errors as they are detected. TSR, on the other hand, uses a fuzzy-matching approach in which individual tokens (i.e., words) comprising the key and the response are each sorted alphabetically before comparing the match between the two ordered strings (Bosker, in press). However, while *lrd* and TSR provide means of fuzzy string matching, *lrd*'s use of Levenshtein distance is more appropriate for the single-word responses commonly used in memory research (e.g., cued- and free-recall testing).

Further, while *Autoscore* and TSR were each designed for matching sentences, *lrd*'s sentence scoring matching provides additional functionality by including output for omitted and extra items from each participant's response, allowing researchers to whether certain words in each sentence are commonly forgotten. Neither Autoscore nor TSR were designed to handle open-ended response tasks such as free recall, in which there is not always a direct one-to-one correspondence between participant responses and answer key items. Finally, *lrd* includes several plotting and summary functions, allowing researchers to quickly assess trends in their scored output. Taken together, *lrd* provides researchers with a comprehensive set of tools that are particularly tailored towards scoring lexical output from a variety of recall paradigms quickly and accurately while also providing options for data visualization and exploration.

## Summary and conclusion

Although recall tests are widely used in psychology, few open-access tools currently exist to quickly process the large amounts of lexical text data that these studies generate. The *lrd* package addresses this need by providing researchers with a means of automating multiple types of recall scoring as a means of saving time and minimize coding errors, while also being able to control for minor errors in participant responses. By using this package to replicate results from cued-recall, free-recall, and sentence-recall experiments, we show that *lrd* can accurately reproduce each type of data. We hope that *lrd* will both drastically reduce the amount of time spent coding lexical data while ensuring near-perfect accuracy and assist the reproducibility measures being adopted by the field by providing researchers with a standardized, open-source method for processing lexical output across psychological studies.

**Author Note** The source code for this package is available at https://github.com/npm27/lrd/. All applicable analysis files have been made available at https://osf.io/admyx/. The Shiny application can be accessed at: https://npm27.shinyapps.io/lrd_shiny/ and a user guide is available at www.macapsych.com/lexical-re. We would like to thank Jason Geller for providing sentence-recall data for this manuscript.

## References

Altman, D. G., & Bland, J. M. (1994). Diagnostic tests. 1: Sensitivity and specificity. *BMJ :British Medical Journal*, *308*(6943), 1552. https://doi.org/10.1136/bmj.308.6943.1552

Balota, D. A., Yap, M. J., Hutchison, K. A., Cortese, M. J., Kessler, B., Loftis, B., Neely, J. H., Nelson, D. L., Simpson, G. B., & Treiman, R. (2007). The English Lexicon Project. *Behavior Research Methods*, *39*(3), 445–459. https://doi.org/10.3758/BF03193014

Becker, J., Chan, C., Chan, G. C., Leeper, T. J., Gandrud, C., MacDonald, A., Zahn, I., Stadlmann, S., Williamson, R., Kennedy, P., Price, R., Davis, T. L., Day, N., Denney, B., & Bokov, A. (2021). *rio: A Swiss-Army Knife for Data I/O* (0.5.26) [Computer software]. https://CRAN.R-project.org/package=rio

Borrie, S. A., Barrett, T. S., & Yoho, S. E. (2019). Autoscore: An open-source automated tool for scoring listener perception of speech. *The Journal of the Acoustical Society of America*, *145*(1), 392–399. https://doi.org/10.1121/1.5087276

Bosker, H. R. (in press). Using fuzzy string matching for automated assessment of listener transcripts in speech intelligibility studies. *Behavior Research Methods*https://doi.org/10.3758/s13428-021-01542-4

Brysbaert, M., Mandera, P., McCormick, S. F., & Keuleers, E. (2019). Word prevalence norms for 62,000 English lemmas. *Behavior Research Methods*, *51*(2), 467–479. https://doi.org/10.3758/s13428-018-1077-9

Brysbaert, M., Warriner, A. B., & Kuperman, V. (2014). Concreteness ratings for 40 thousand generally known English word lemmas. *Behavior Research Methods*, *46*(3), 904–911. https://doi.org/10.3758/s13428-013-0403-5

Buchanan, E. M., Valentine, K. D., & Maxwell, N. P. (2019). LAB: Linguistic Annotated Bibliography – a searchable portal for normed database information. *Behavior Research Methods*, *51*(4), 1878–1888. https://doi.org/10.3758/s13428-018-1130-8

Buhrmester, M., Kwang, T., & Gosling, S. D. (2011). Amazon's Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data? *Perspectives on Psychological Science*, *6*(1), 3–5. https://doi.org/10.1177/1745691610393980

Chang, W., Cheng, J., Allaire, J. J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., & Borges, B. (2021). *shiny: Web Application Framework for R* (1.6.0) [Computer software]. https://CRAN.R-project.org/package=shiny

Chang, W., & Ribeiro, B. B. (2018). *shinydashboard: Create Dashboards with "Shiny"* (0.7.1) [Computer software]. https://CRAN.R-project.org/package=shinydashboard

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, *20*(1), 37–46. https://doi.org/10.1177/001316446002000104

Craik, F. I. M., & Lockhart, R. S. (1972). Levels of processing: A framework for memory research. *Journal of Verbal Learning and Verbal Behavior*, *11*(6), 671–684. https://doi.org/10.1016/S0022-5371(72)80001-X

Criss, A. H., Aue, W. R., & Smith, L. (2011). The effects of word frequency and context variability in cued recall. *Journal of Memory and Language*, *64*(2), 119–132. https://doi.org/10.1016/j.jml.2010.10.001

De Deyne, S., Navarro, D. J., Perfors, A., Brysbaert, M., & Storms, G. (2019). The "Small World of Words" English word association norms for over 12,000 cue words. *Behavior Research Methods*, *51*(3), 987–1006. https://doi.org/10.3758/s13428-018-1115-7

Dowle, M., & Srinivasan, A. (2020). data.table: Extension of 'data.frame'. Retrieved fromhttps://CRAN.R-project.org/package=data.table. Accessed 13 April 2021

Geller, J., McMurray, B., Choi, I., & Holmes, A. (2020). *Validation of the Iowa Test of Consonant Perception* [Preprint]. PsyArXiv. https://doi.org/10.31234/osf.io/wxd93

Gretz, M. R., & Huff, M. J. (2019). Did you wash your hands? Evaluating memory for objects touched by healthy individuals and individuals with contagious and noncontagious diseases. *Applied Cognitive Psychology*, *33*(6), 1271–1278. https://doi.org/10.1002/acp.3604

Huff, M. J., Yates, T. J., & Balota, D. A. (2018). Evaluating the contributions of task expectancy in the testing and guessing benefits on recognition memory. *Memory (Hove, England)*, *26*(8), 1065–1083. https://doi.org/10.1080/09658211.2018.1467929

Hutchison, K. A., Balota, D. A., Neely, J. H., Cortese, M. J., Cohen-Shikora, E. R., Tse, C.-S., Yap, M. J., Bengson, J. J., Niemeyer, D., & Buchanan, E. (2013). The semantic priming project. *Behavior Research Methods*, *45*(4), 1099–1114. https://doi.org/10.3758/s13428-012-0304-z

Kahana, M. J. (1996). Associative retrieval processes in free recall. *Memory & Cognition*, *24*(1), 103–109. https://doi.org/10.3758/BF03197276

Kahana, M. J., Howard, M. W., Zaromb, F., & Wingfield, A. (2002). Age dissociates recency and lag recency effects in free recall. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *28*(3), 530–540. https://doi.org/10.1037/0278-7393.28.3.530

Koriat, A., & Bjork, R. A. (2005). Illusions of Competence in Monitoring One's Knowledge During Study. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *31*(2), 187–194. https://doi.org/10.1037/0278-7393.31.2.187

Kuhn, M. (2008). Building Predictive Models in R Using the caret Package. *Journal of Statistical Software*, *28*(1), 1–26. https://doi.org/10.18637/jss.v028.i05

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics-Doklady, 10*(8), 707–710

Maxwell, N. P., & Buchanan, E. M. (2020). Investigating the interaction of direct and indirect relation on memory judgments and retrieval. *Cognitive Processing*, *21*(1), 41–53. https://doi.org/10.1007/s10339-019-00935-w

Maxwell, N. P., & Huff, M. J. (2021). The deceptive nature of associative word pairs: The effects of associative direction on judgments of learning. *Psychological Research*, *85*, 1757–1775. https://doi.org/10.1007/s00426-020-01342-z

Maxwell, S. E., Lau, M. Y., & Howard, G. S. (2015). Is psychology suffering from a replication crisis? What does "failure to replicate" really mean? *American Psychologist*, *70*(6), 487–498. https://doi.org/10.1037/a0039400

Murdock, B. B. (1962). The serial position effect of free recall. *Journal of Experimental Psychology*, *64*(5), 482–488. https://doi.org/10.1037/h0045106

Paivio, A., Clark, J. M., & Khan, M. (1988). Effects of concreteness and semantic relatedness on composite imagery ratings and cued recall. *Memory & Cognition*, *16*(5), 422–430. https://doi.org/10.3758/BF03214222

Polyn, S. M., Norman, K. A., & Kahana, M. J. (2009). A context maintenance and retrieval model of organizational processes in free recall. *Psychological Review*, *116*(1), 129–156. https://doi.org/10.1037/a0014420

Revelle, W. (2020). *psych: Procedures for Psychological, Psychometric, and Personality Research* (2.0.12) [Computer software]. https://CRAN.R-project.org/package=psych

Singla, N., & Garg, D. (2012). String Matching Algorithms and their Applicability in various Applications. *International Journal of Soft Computing and Engineering*, *1*(6), 218–222.

Spahr, A. J., Dorman, M. F., Litvak, L. M., Van Wie, S., Gifford, R. H., Loizou, P. C., Loiselle, L. M., Oakes, T., & Cook, S. (2012). Development and validation of the AzBio sentence lists. *Ear and Hearing*, *33*(1), 112–117. https://doi.org/10.1097/AUD.0b013e31822c2549

Taylor, J. E., Beith, A., & Sereno, S. C. (2020). LexOPS: An R package and user interface for the controlled generation of word stimuli. *Behavior Research Methods*, *52*(6), 2372–2382. https://doi.org/10.3758/s13428-020-01389-1

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L.D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., et al. (2019). Welcome to the tidyverse. Journal of Open Source Software, 4, 1686.

Wahlheim, C. N., & Huff, M. J. (2015). Age differences in the focus of retrieval: Evidence from dual-list free recall. *Psychology and Aging*, *30*(4), 768–780. https://doi.org/10.1037/pag0000049

Wickham, H. (2007). Reshaping Data with the reshape Package. *Journal of Statistical Software*, *21*(1), 1–20. https://doi.org/10.18637/jss.v021.i12

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., Dunnington, D., & RStudio. (2021). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics* (3.3.5) [Computer software]. https://CRAN.R-project.org/package=ggplot2

Wickham, H., Hester, J., & Chang, W. (2020). *devtools: Tools to Make Developing R Packages Easier* (2.3.2) [Computer software]. https://CRAN.R-project.org/package=devtools